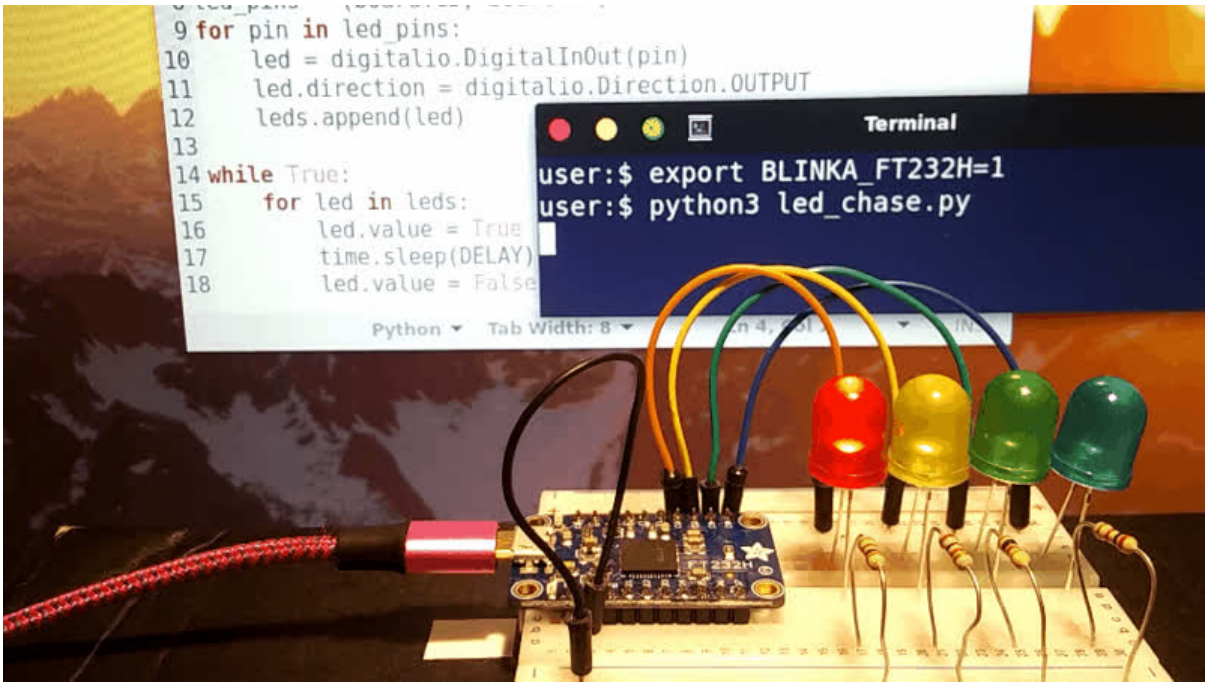
CircuitPython Libraries on any Computer with FT232H

Created by Carter Nelson



https://learn.adafruit.com/circuitpython-on-any-computer-with-ft232h

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

Table of Contents

Overview 5

- CircuitPython and CircuitPython Libraries
- CircuitPython on Microcontrollers
- CircuitPython on Single Board Computers
- CircuitPython 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 11

- Have Python 3 Installed
- Plug in FT232H & Fix Driver with Zadig
- Option 1 - Install pyftdi and pyusb
- Option 2 - Install pyftdi and (fixed) pyusb
- Test pyusb and pyftdi
- Install Adafruit Blinka
- Set Environment Variable
- Check Platform was detected

Mac OSX 18

- Install libusb
- Install pyftdi and Blinka
- Test!

Linux 20

- Install libusb
- Setup udev rules
- Install pyftdi
- Install Blinka
- Set environment variable
- Run the sanity check.

Post Install Checks 23

- Check pyftdi is installed correctly
- Check environment variable within python
- Board not supported None
- BLINKA_FT232H environment variable set, but no FT232H device found
- The device has no langid
- NotImplementedError: Operation not supported or unimplemented on this platform
- pyftdi.ftdi.FtdiError: UsbError: [Errno 13] Access denied (insufficient permissions)

• Versions of OS X up to and including 10.8
• Versions of OS X 10.9 through 10.13

Pinout 27

• Power Pins
• GPIO Pins
• I2C Pins
• SPI Pins
• STEMMA QT

Examples 29

• Installing Libraries for Breakouts

GPIO 30

• Digital Output
• Digital Input
• Digtal Input and Output

I2C 34

• Install BME280 Library
• Run Example
• Using STEMMA QT

SPI 39

• Install BME280 Library
• Run Example

NeoPixels 41

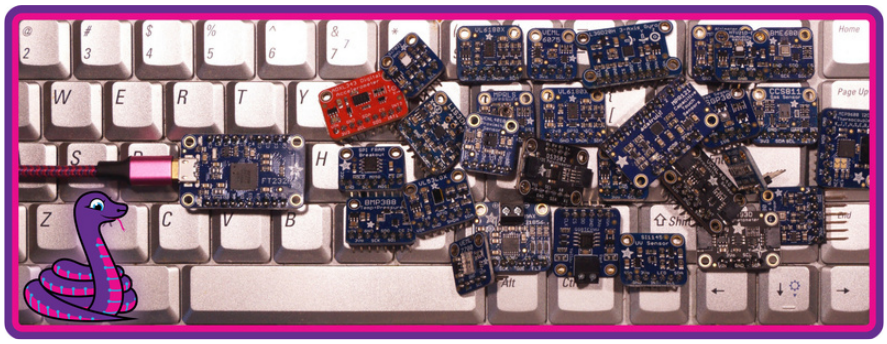
TFT Display 41

• Connect the Display
• Install Pillow/PIL
• Install CircuitPython RGB Display Library
• Example Image
• The Code
• Fancier Demo for Linux

FAQ & Troubleshooting 46

• Update Blinka/Platform Libraries

Overview



This guide will show you how to use an FT232H to connect to I2C and SPI sensors and breakouts from your desktop PC running Windows, Mac OSX, or Linux. The FT232H also allows for general purpose digital input and output (GPIO) for things like buttons and LEDs.

The cool part about this is that you can then use any of the CircuitPython Libraries that have been written for the numerous sensors and breakouts.

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). And here's the main CircuitPython guide (https://adafru.it/cpy-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).

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. To explain this further, let's go over the main hardware platforms and explain where the FT232H fits in to all this.

CircuitPython on Microcontrollers

This is the most straight forward setup.

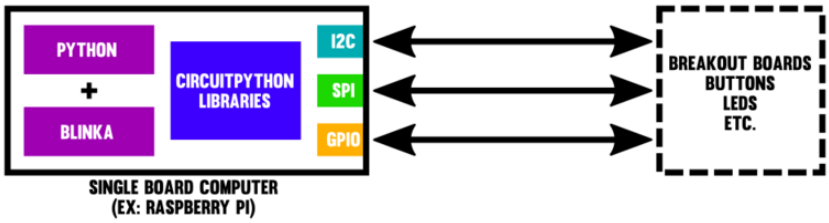
If you are using a microcontroller (https://adafru.it/Em8) that can run CircuitPython, then you will most likely have the low level hardware interfaces needed to access the many sensors and other breakout boards - I2C, SPI, and GPIO. Even better, there is most likely a CircuitPython library written for the sensor.

In this case, you simply install CircuitPython (https://adafru.it/Amd), add the libraries (https://adafru.it/ABU), and then follow whatever guide (https://adafru.it/dlu) goes along with your sensor.

CircuitPython on Single Board Computers

This setup requires a special shim library called Blinka (https://adafru.it/EQa).

Single Board Computers (SBC), like the Raspberry Pi and Beagle, also typically have I2C, SPI, and GPIO interfaces available. These boards are also powerful enough to run complete operating systems, like Linux. They can't run CircuitPython directly, but generally don't need to. They can run the much larger Python implementations, like CPython. To allow use of SBCs running Python to use CircuitPython libraries to access sensors over I2C/SPI/GPIO, the Blinka (https://adafru.it/EQa) library was created.

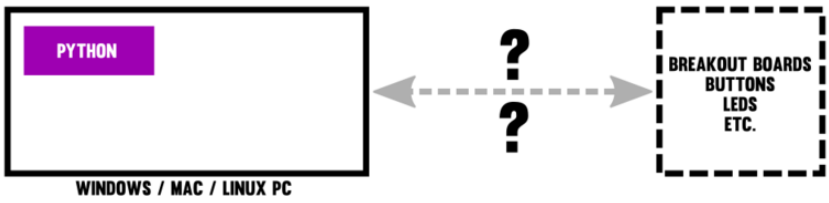


In this case, you pip install Blinka (https://adafru.it/EQa), pip install libraries (https://adafru.it/Deo), and then follow whatever guide (https://adafru.it/dlu) goes along with your sensor.

CircuitPython on Personal Computers

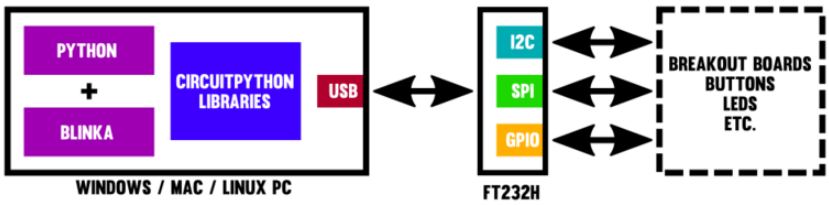
This is where the FT232H comes in. Here's why.

So what about your super powerful desktop or laptop PC? They can most definitely run Python. But can they also use CircuitPython libraries and talk to I2C/SPI sensors? Can they blink LEDs? Read buttons? Generally, no. As powerful as your Windows, Mac, or Linux PC is, it most likely does not have those low level hardware interfaces. Look on the back of your computer. Do you see an I2C port? A SPI port? A cluster of pins labeled GPIO? Nope.



So what can we do? Well, look again at the back of your PC. See any USB ports? Most likely there are several. Heck, there are probably even several USB ports on the front of your PC! Can we use USB? Yes, thanks to a specialized USB bridge chip made by FTDI - the FT232H (http://adafru.it/2264).

This will allow us to do something like this:



With FT232H support added to Blinka, we can follow a similar approach as with the SBCs. The FT232H attached to the USB port acts as our surrogate set of low level hardware interfaces.

Neat! Let's see how we can get this all setup and working.

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 that provide hardware pins you can control. These boards include MCP221A (https://adafru.it/IfV) and FT232H (https://adafru.it/xia) breakout boards, and Raspberry Pi Pico boards running the **u2if** software (https://adafru.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://adafru.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 down 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) 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).
- When you import the library, you write `import adafruit_lis3dh`.
- The name you use with `pip3` is `adafruit-circuitpython-lis3dh`. This 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) for more information about the bundles, and also see the Libraries page on circuitPython.org (https://adafru.it/ENC).

Setup

The support for the FT232H in Blinka utilizes the pyftdi library (https://adafru.it/FVF) by eblot (https://adafru.it/FVG). This in turn relies on a few other things, like libusb. So before we can actually use the FT232H, 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 install info from the pyftdi library itself:

pyftdi Installation

https://adafru.it/FVH

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

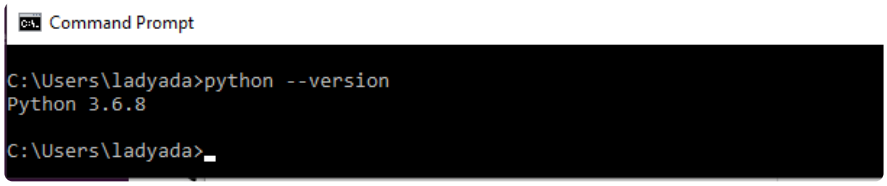
Windows

Getting this all set up on Windows is not fun - but it is possible. Follow each step below to get it working on 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`



Plug in FT232H & Fix Driver with Zadig

Unlike Mac or Linux, Windows doesn't like devices that are 'driverless' - every hardware device must have a driver attached to it. To fix the driver on Windows, we must use a tool called Zadig

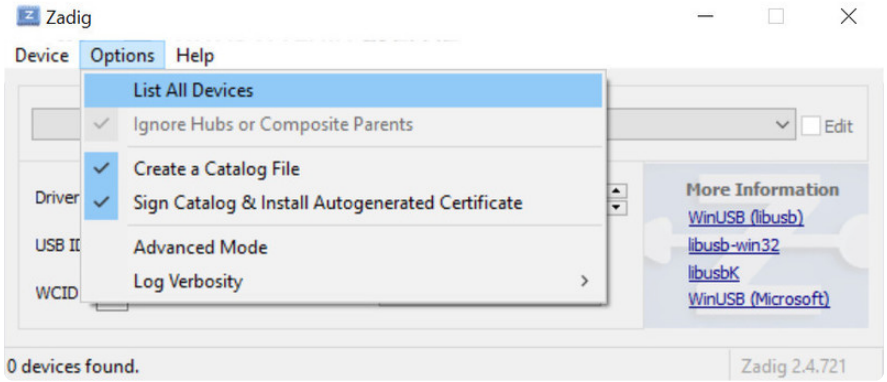
You only have to 'Zadig' the board once per computer. If you get another FT232H board, you will need to run Zadig again to set the driver for the second board. Other than that, you only have to run it once.

Download the appropriate Zadig tool for your version of Windows at its homepage (https://adafru.it/eal). This tool simplifies the installation of a libusb driver for the FT232H device.

Before you run the tool unplug all FTDI devices from your computer. This includes devices like Arduinos which include an FTDI chip as a USB to serial converter. You want to unplug these devices to make sure you don't accidentally select one with the tool and replace its driver.

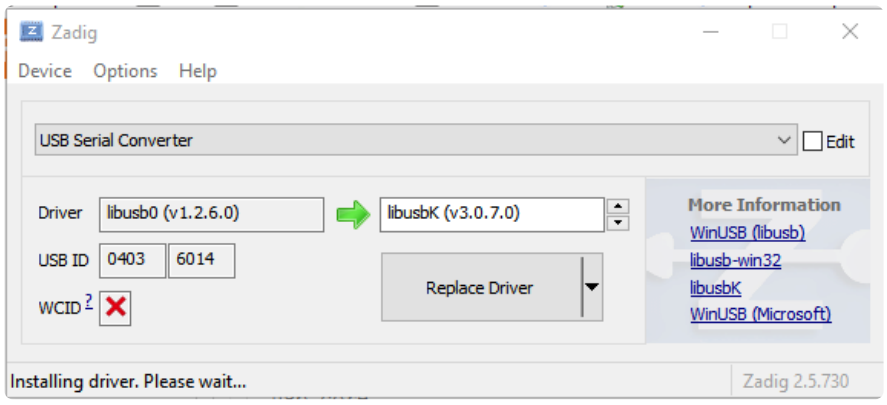
After all the FTDI devices are unplugged, plug in your FT232H breakout to the computer so it is the only FTDI device connected to the computer.

Now run the Zadig tool executable you just downloaded (there is no installation necessary, the executable is the program). Click the **Options** menu and select the **List All Devices** item below:



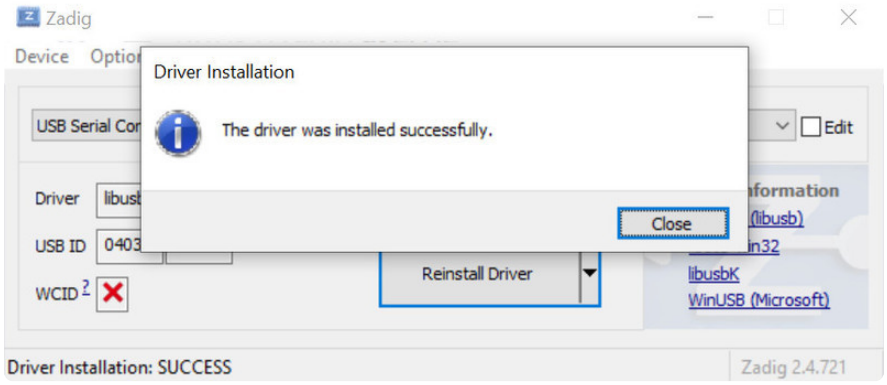
The list box of devices should populate with many devices. Select the USB Serial Converter device shown below.

Note: Make sure to select the device with Driver equal to FTDIBUS and USB ID equal to 0403 6014! If you pick the wrong device you might accidentally uninstall another device's driver and make it inoperable.

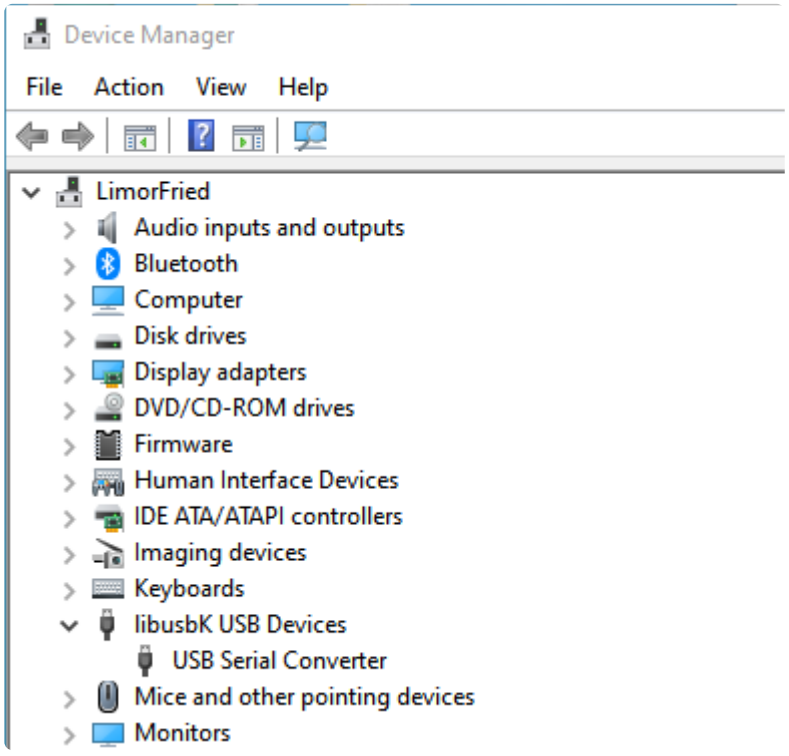


Click the up/down arrows on the driver select box to the right of the green arrow and select the **libusbK** driver as shown above.

Now click the Replace Driver button to replace the FTDI driver with the libusbK-based driver. After the driver replacement finishes you can close Zadig tool.



To check that the driver was successfully replaced, open Device Manager from Control Panel or searching in the Start menu. You should see a new top level node **libusbK devices** and the **USB Serial Converter** underneath it as shown below.



If you see the libusb-win32 node and USB serial device, move on to the next step to install libftdi.

If you don't see the libusb-win32 node, try unplugging and plugging back in the FT232H breakout. If you still don't see the node, run Zadig tool again and follow the

steps above again to make sure you replace the FTDI driver for the device with the libusb-win32 driver.

Option 1 - Install pyftdi and pyusb

As a first attempt, try simply pip installing pyftdi and pyusb.

```
pip3 install pyusb
```

```
pip3 install pyftdi
```

And then move to the **Test pyusb and pyftdi** section below. If the install runs into issues or doesn't seem to work, you may need to try **Option 2**. This is what it originally took to set things up on Windows. However, it has been reported that the latest version of pyusb "just works", so you might get lucky :)

Option 2 - Install pyftdi and (fixed) pyusb

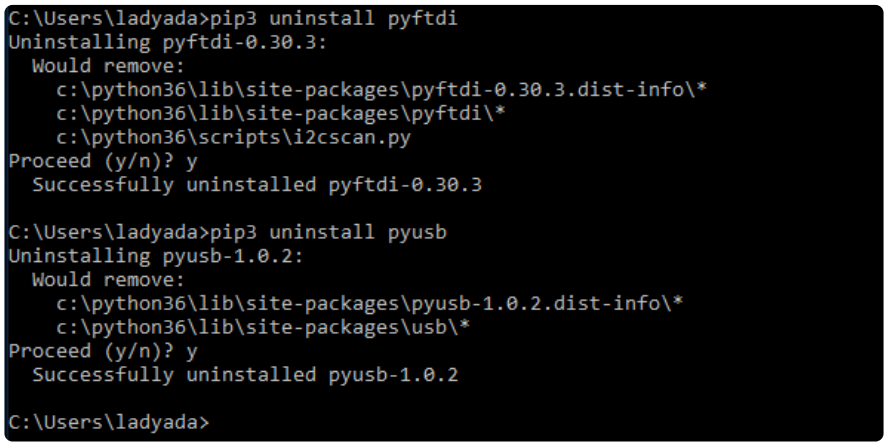
Next lets install pyusb and pyftdi. We have to do a little trickery here so before you start run

pip3 uninstall pyusb

pip3 uninstall pyftdi

(or pip instead of pip3 if that's how you have it named)

To make sure you do not have pyusb and pyftdi installed



Run them again to make absolutely sure!

```
C:\Users\ladyada>pip3 uninstall pyusb
WARNING: Skipping pyusb as it is not installed.

C:\Users\ladyada>pip3 uninstall pyftdi
WARNING: Skipping pyftdi as it is not installed.
```

We need to get a fork of pyusb to fix a bug that affects windows:

`git clone https://github.com/minkustree/pyusb.git`

```
C:\Users\ladyada>git clone https://github.com/minkustree/pyusb.git
Cloning into 'pyusb'...
remote: Enumerating objects: 2090, done.
remote: Total 2090 (delta 0), reused 0 (delta 0), pack-reused 2090
Receiving objects: 100% (2090/2090), 1011.18 KiB | 8.36 MiB/s, done.
Resolving deltas: 100% (1197/1197), done.

C:\Users\ladyada>
```

Then

`cd pyusb`

`python setup.py install`

if you get a permission denied error, try

`python setup.py install --user`

```
C:\Users\ladyada>cd pyusb

C:\Users\ladyada\pyusb>python setup.py install
running install
running bdist_egg
running egg_info
creating pyusb.egg-info
writing pyusb.egg-info\PKG-INFO
writing dependency_links to pyusb.egg-info\dependency_links.txt
writing top-level names to pyusb.egg-info\top_level.txt
writing manifest file 'pyusb.egg-info\SOURCES.txt'
reading manifest file 'pyusb.egg-info\SOURCES.txt'
reading manifest template 'MANIFEST.in'
```

Now you can run `pip install pyftdi`

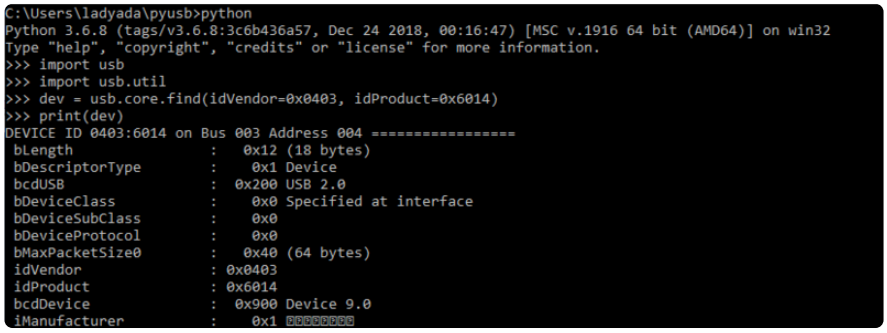
```
C:\Users\ladyada\pyusb>pip install pyftdi
Collecting pyftdi
Requirement already satisfied: pyserial>=3.0 in c:\python36\lib\site-packages (from pyftdi) (3.4)
Requirement already satisfied: pyusb>=1.0.0 in c:\python36\lib\site-packages\pyusb-1.0.2-py3.6.egg

Installing collected packages: pyftdi
Successfully installed pyftdi-0.30.3
```

Test pyusb and pyftdi

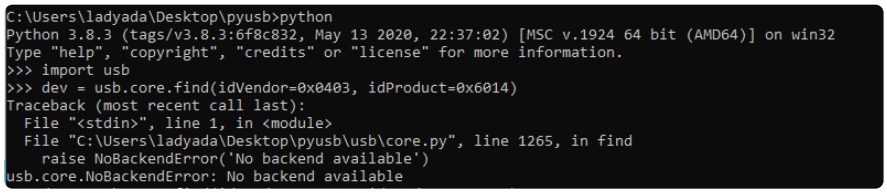
Now that you have pyusb and pyftdi installed correctly, run python and paste in the following (with the FT232H plugged in)

```
import usb
import usb.util
dev = usb.core.find(idVendor=0x0403, idProduct=0x6014)
print(dev)
```

You should get something like the following, not that `dev` is `None` or any other weird failure



If you get `usb.core.NoBackendError: No backend available`



Download libusb 1.0.23 (https://adafru.it/LxD), uncompress it with WinRar or some other decompression tool that can open 7z files.

If you are on 64-bit Windows, copy the `MS64\dll\libusb-1.0.dll` file into `C:\Windows\System32` and `C:\Windows\SysWOW64I`

If you are on 32-bit Windows, copy the `MS32\dll\libusb-1.0.dll` file into `C:\Windows\System32`

Install Adafruit Blinka

Run `pip install adafruit-blinka`

```
C:\Users\ladyada>pip install adafruit-blinka
Collecting adafruit-blinka
Requirement already satisfied: Adafruit-PureIO in c:\python36\lib\site-packages (
Requirement already satisfied: Adafruit-PlatformDetect in c:\python36\lib\site-pa
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_FT232H=1
```

```
C:\Users\ladyada>set BLINKA_FT232H=1

C:\Users\ladyada>
```

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

```
$env:BLINKA_FT232H=1
```

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

Check Platform was detected

In the same command window you `set BLINKA_FT232H=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_FT232H=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)
['C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'D4', 'D5', 'D6', 'D7', 'I
2C', 'MISO', 'MOSI', 'SCK', 'SCL', 'SCLK', 'SDA', 'SPI', '__builtins__', '_
_cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'ap_board', 'board_id', 'detector', 'pin', 'sys']
>>>
```

Mac OSX

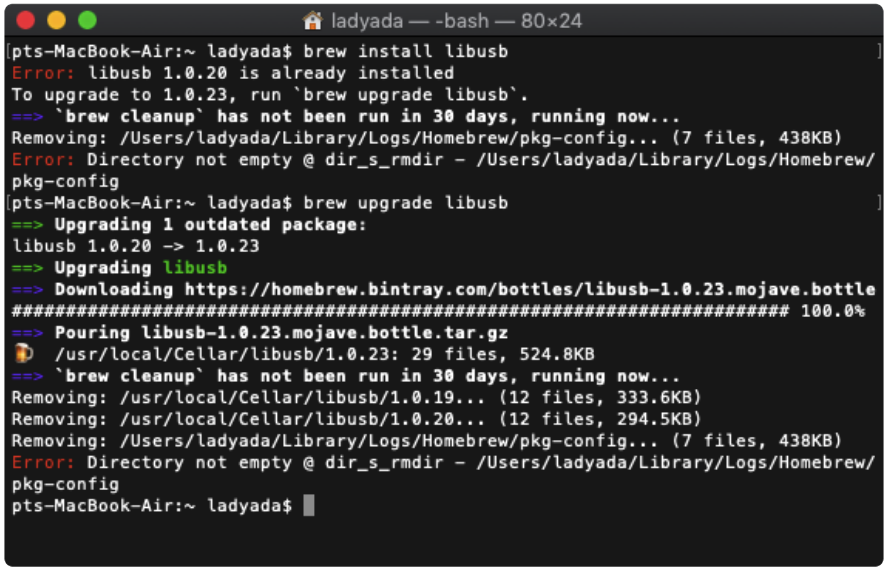
We assume you already have Python 3 and brew available on your Mac. Thankfully, setup on MacOS X is not so bad!

Note: If you are running VMWare Fusion on MacOS, then you can also try the Windows install (https://adafru.it/FX6) process.

Install libusb

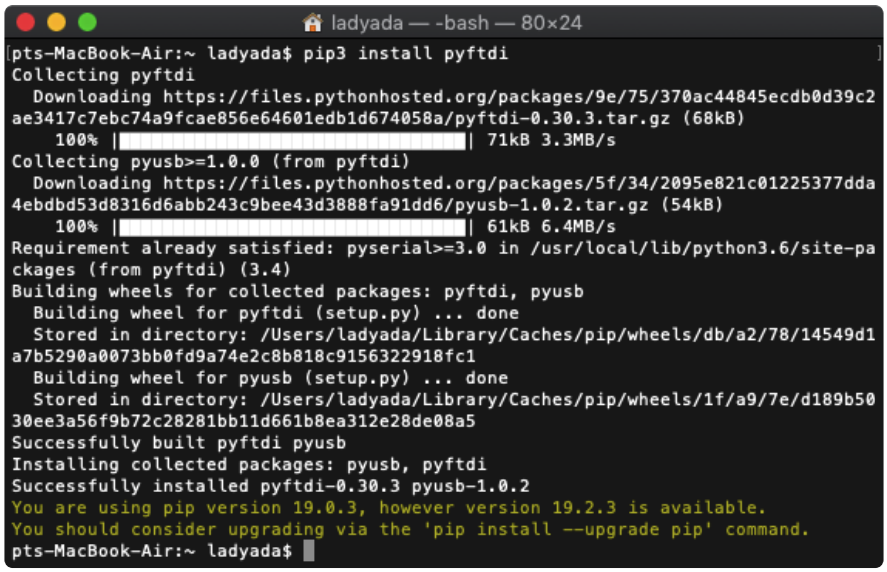
Start by installing libusb with

```
brew install libusb
```

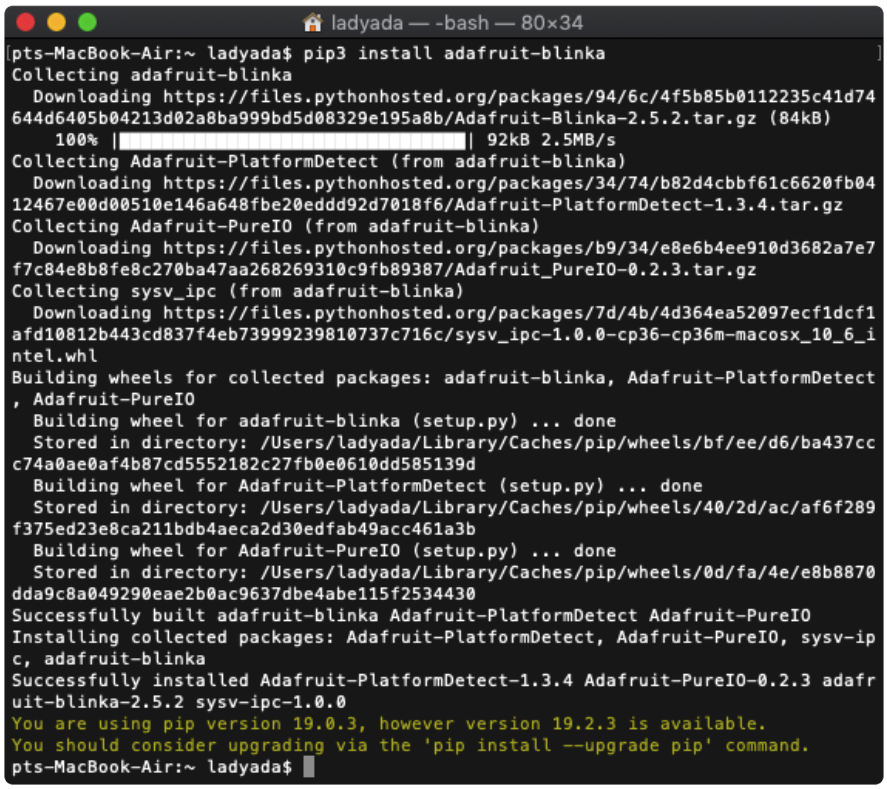


Install pyftdi and Blinka

Then, `pip3 install pyftdi` which will also install some other libraries



Then `pip3 install adafruit-blinka`



Test!

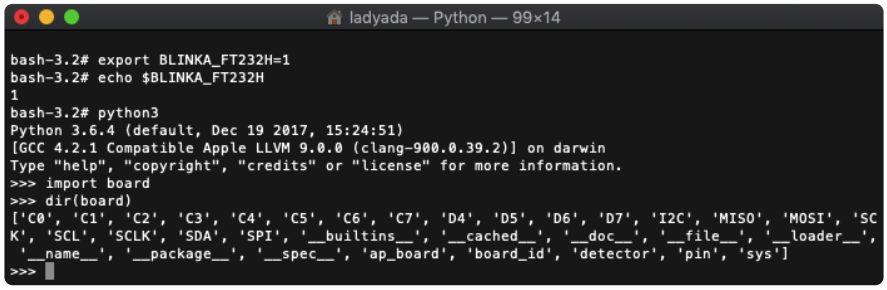
Finally, set the environment variable with

```
export BLINKA_FT232H=1
```

and run python3. In the REPL, type

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

You should get no errors, and a list of the pins available



Linux

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

Install libusb

Run the following:

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

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

Setup udev rules

Use a text editor to create and edit the file **/etc/udev/rules.d/11-ftdi.rules** and add the following contents.

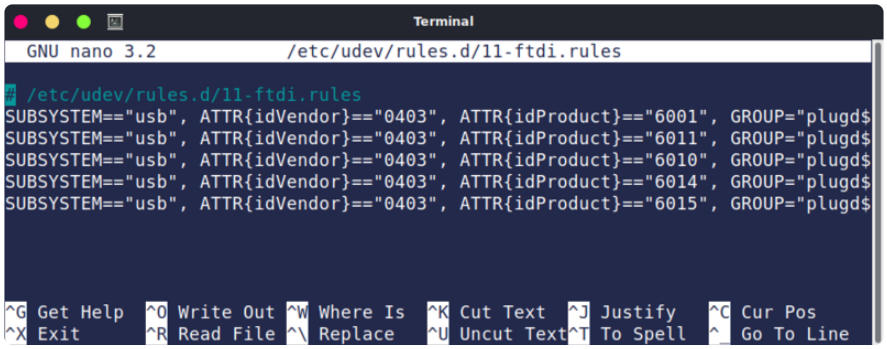
```
# /etc/udev/rules.d/11-ftdi.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6001", GROUP="plugdev",
MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6011", GROUP="plugdev",
MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6010", GROUP="plugdev",
MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6014", GROUP="plugdev",
MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6015", GROUP="plugdev",
MODE="0666"
```

Here we use nano, so run:

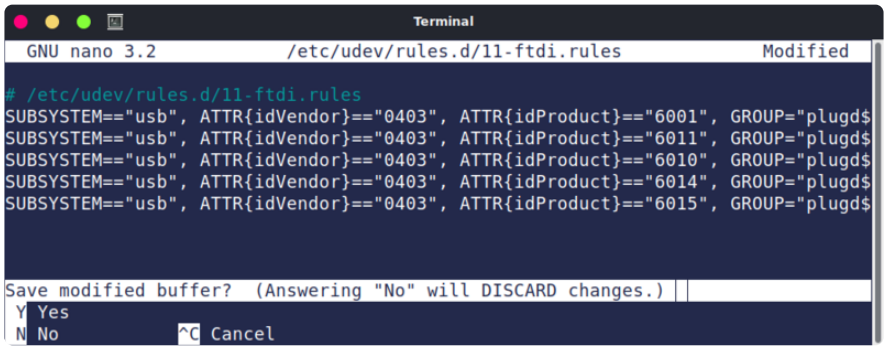
```
sudo nano /etc/udev/rules.d/11-ftdi.rules
```

like this:

add the contents from above:



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



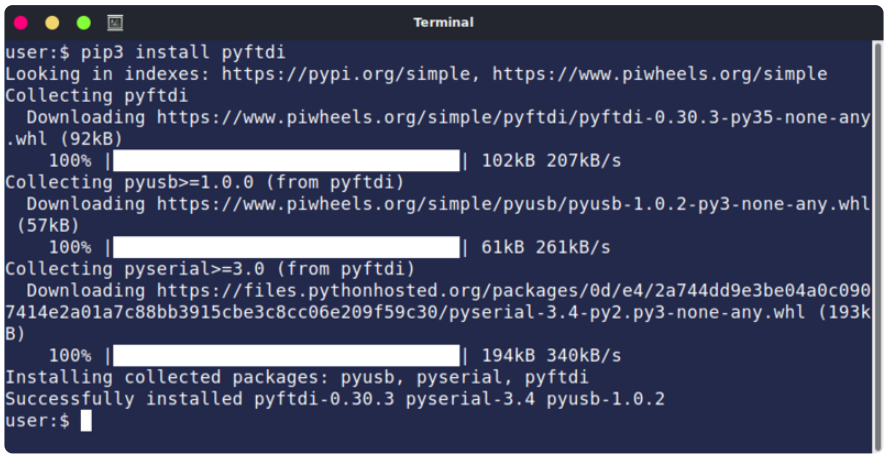
The settings will take effect the next time you plug in the FT232H.

The settings will take effect the next time you plug in the FT232H.

Install pyftdi

To install pyftdi and its dependencies, run:

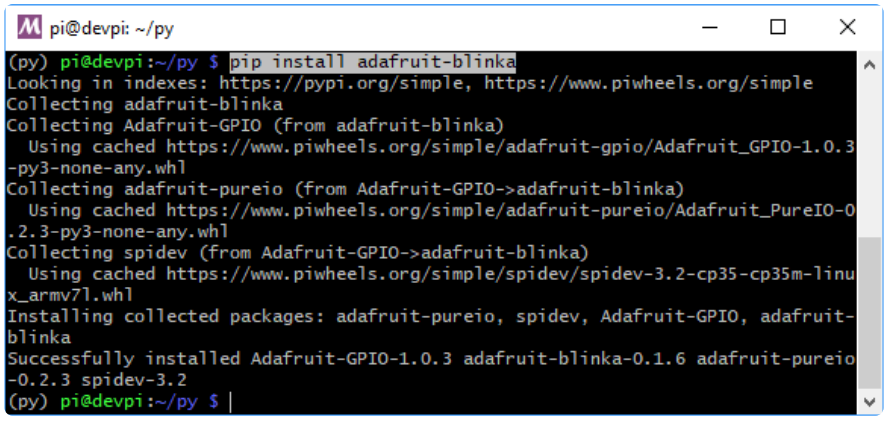
```
pip3 install pyftdi
```



Install Blinka

To install Blinka and its dependencies, run:

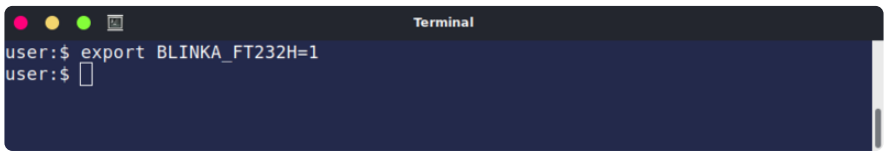
```
pip3 install adafruit-blinka
```



Set environment variable

We need to manually signal to Blinka that we have a FT232H attached. To do this, we set the environment variable **BLINKA_FT232H**. The value doesn't matter, just use 1:

```
export BLINKA_FT232H=1
```



Don't forget this step. Things won't work unless BLINKA_FT232H is set.

Run the sanity check.

Now move on to the Sanity Check 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 this as a simple test 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 FT232H to a USB port on your PC.

Check pyftdi is installed correctly

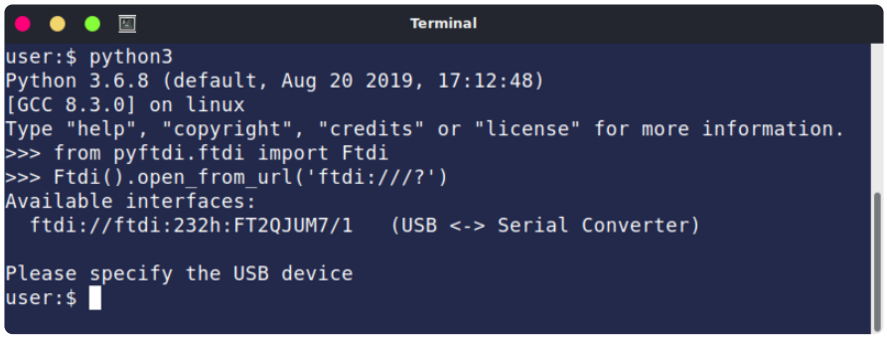
Launch Python:

```
python3
```

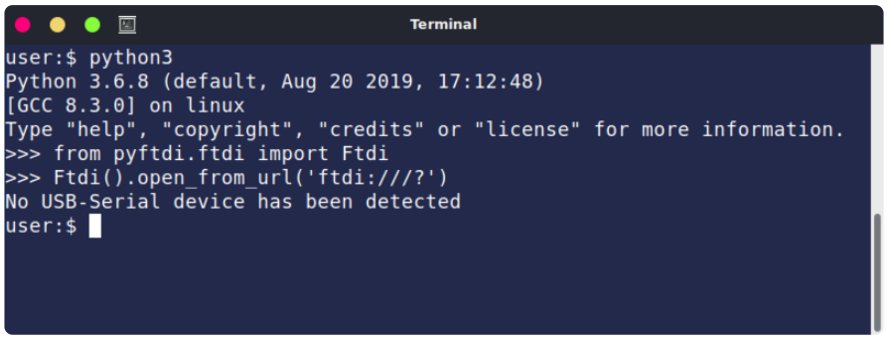
and enter these commands:

```
from pyftdi.ftdi import Ftdi
Ftdi().open_from_url('ftdi:///?')
```

You should get a list of connected FTDI devices. Most likely, there will be only one.



If an FTDI device can not be found for some reason, you'll see something like this:



If you get

`NotImplementedError: Operation not supported or unimplemented on this platform`

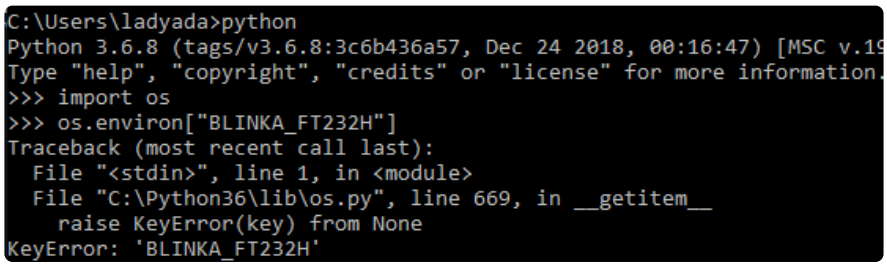
Go back to the setup steps and Zadig your board if on Windows

Check environment variable within python

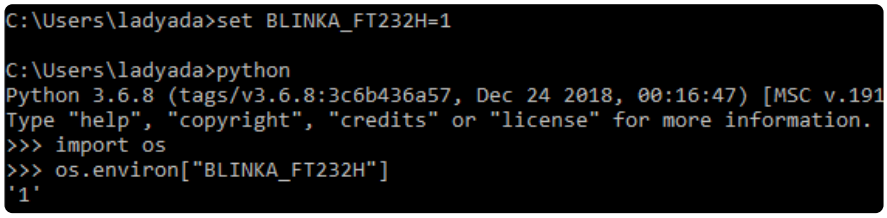
At the Python REPL, type

```
import os
os.environ["BLINKA_FT232H"]
```

if you get a KeyError it means you did not set the environment variable right



If you see some other errors, go back through the install process for your OS and make sure everything ran correctly.



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

Now type in `import board`

```
C:\Users\ladyada>python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916
Type "help", "copyright", "credits" or "license" for more information.
>>> import board
>>>
```

You should get no errors at all, in which case you can continue onto the next steps!

Board not supported None

If you get

`NotImplementedError: Board not supported None`

that could mean you did not set the FT232H environmental variable or you don't have the latest python libraries or the FT232H is not plugged in.

```
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
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python36\lib\site-packages\board.py", line 107, in <module>
    raise NotImplementedError("Board not supported {}".format(board_id))
NotImplementedError: Board not supported None
>>>
```

BLINKA_FT232H environment variable set, but no FT232H device found

If you get this error, check your USB cable - it could be that you have a charge-only not charge+sync cable.

If you're running Windows, verify you ran Zadig to install the libusbK driver

```
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
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python36\lib\site-packages\board.py", line 32, in <module>
    from adafruit_blinka.agnostic import board_id, detector
  File "C:\Python36\lib\site-packages\adafruit_blinka\agnostic\__init__.py", line 18, in <module>
    chip_id = detector.chip.id
  File "C:\Python36\lib\site-packages\adafruit_platformdetect\chip.py", line 47, in id
    'set, but no FT232H device found')
RuntimeError: BLINKA_FT232H environment variableset, but no FT232H device found
```

The device has no langid

On Windows, if you get `ValueError: The device has no langid` on an open_from_url() call that may mean you dont have **libusbk.dll** placed in **C:\Windows\System32**

```
>>> from pyftdi.ftdi import Ftdi
>>> Ftdi().open_from_url('ftdi:///?')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 399, in open_from_url
    vendor, product, index, serial, interface = self.get_identifiers(url)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 311, in get_identifiers
    cls.SCHEME, cls.VENDOR_IDS, cls.PRODUCT_IDS, cls.DEFAULT_VENDOR)
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 352, in parse_url
    devices = devclass.find_all(vps)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 392, in find_all
    return UsbTools.find_all(vps, nocache)
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 68, in find_all
    devs.update(UsbTools._find_devices(v, p, nocache))
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 271, in _find_devices
    sn = UsbTools.get_string(dev, dev.iSerialNumber)
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 508, in get_string
    return usb.util.get_string(device, strname)
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\util.py", line 301, in get_string
ValueError: The device has no langid
```

NotImplementedError: Operation not supported or unimplemented on this platform

If you're running Windows, run Zadig to select/install the libusbK driver on this device

```
>>> import board
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python36\lib\site-packages\board.py", line 32, in <module>
    from adafruit_blinka.agnostic import board_id, detector
  File "C:\Python36\lib\site-packages\adafruit_blinka\agnostic\__init__.py", line 18, in <module>
    chip_id = detector.chip.id
  File "C:\Python36\lib\site-packages\adafruit_platformdetect\chip.py", line 44, in id
    count = len(UsbTools.find_all([(0x0403, 0x6014)]))
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 68, in find_all
    devs.update(UsbTools._find_devices(v, p, nocache))
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 271, in _find_devices
    sn = UsbTools.get_string(dev, dev.iSerialNumber)
  File "C:\Python36\lib\site-packages\pyftdi\usbtools.py", line 508, in get_string
    return usb.util.get_string(device, strname)
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\util.py", line 298, in get_string
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\core.py", line 807, in langids
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\util.py", line 255, in get_langids
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\control.py", line 160, in get_descriptor
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\core.py", line 1027, in ctrl_transfer
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\core.py", line 89, in wrapper
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\core.py", line 107, in managed_open
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\backend\libusb1.py", line 786, in open_device
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\backend\libusb1.py", line 635, in __init__
  File "C:\Python36\lib\site-packages\pyusb-1.0.2-py3.6.egg\usb\backend\libusb1.py", line 585, in _check
NotImplementedError: Operation not supported or unimplemented on this platform
```

pyftdi.ftdi.FtdiError: UsbError: [Errno 13] Access denied (insufficient permissions)

If you get this on a Mac, it could be you have conflicting ftdi drivers installed, you'll need to remove them!

Versions of OS X up to and including 10.8

The easiest way we found to do that is run

```
brew remove libftdi
```

This may have been installed by `avrdude`, `open-ocd` or other tools! You may have to --force uninstall it depending on what tools you are using.

For OS X versions 10.3 to 10.8, there is a procedure outlined in section 4.1 of Application Note AN 134 from FTDI:

AN 134 FTDI Drivers Installation guide for MAC OS X

https://adafru.it/ucE

In summary, start a terminal session:

Go -> Applications -> Utilities -> Terminal

and run the following commands:

```
cd /System/Library/Extensions
sudo rm -r FTDIUSBSerialDriver.kext
cd /Library/Receipts
sudo rm -r ftdiusbserialdriver.pkg
sudo rm -r ftdiusbserialdriverinstallerPostflight.pkg
sudo rm -r ftdiusbserialdriverinstallerPreflight.pkg
```

and then reboot the system for the changes to take effect.

Versions of OS X 10.9 through 10.13

For OS X versions 10.9 though 10.13, you can unload the kext driver by running the following command:

```
sudo kextunload -v -bundle com.apple.driver.AppleUSBFTDI
```

Please note, the system will automatically reload the kext driver upon restart, so this would need to be run each time before using the board.

You may want to use an alias or script to easily run this command.

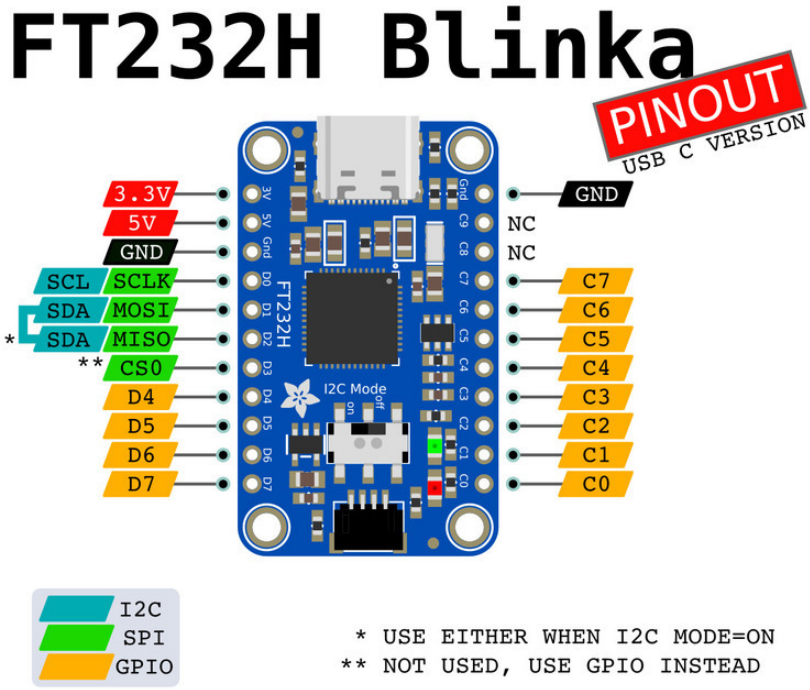
PyFtdi Troubleshooting Guide

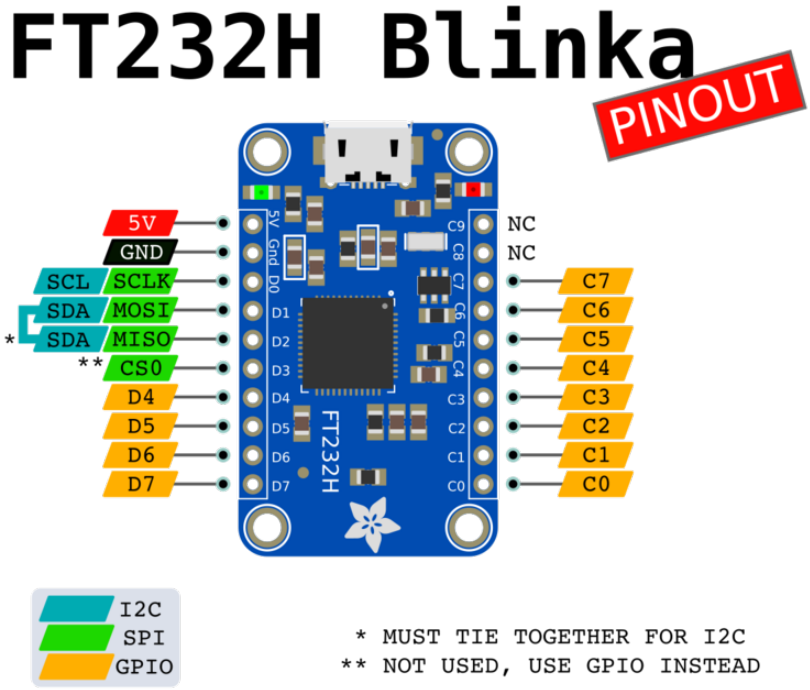
https://adafru.it/19fw

Pinout

Note that there are two versions of this board in the wild. It was updated on Feb. 12, 2020 to have a USB C connector, additional pinouts, and other features (see product page for more info (http://adafru.it/2264)).

Pinouts for both the **new version** (USB C) and the **original version** are below.





I2C and SPI share the same pins, so only one mode can be used at a time.

Power Pins

- **5V** - this is the 5V power from the USB input.

- **GND** - this is the common ground for all power and logic.
- **3V power output** - The **new version** has a 3.3V power output pin for up to 500mA

GPIO Pins

- **D4** to **D7** - can be used as either digital inputs or outputs.
- **C0** to **C7** - can be used as either digital inputs or outputs.

I2C Pins

- **SCL** - the I2C clock signal is on **D0**.
- **SDA** - the I2C data is on **D1+D2.**
- **I2C switch** - The **new version** has a switch that connects D1 and D2 for easy I2C interfacing. **Move the switch to ON to use I2C** and/or the STEMMA QT connector. You can then use either D1 or D2 for SDA.

On the original version only: Note that there are two pins (D1 and D2) which must be tied together and treated as one to use I2C.

SPI Pins

- **SCLK** - the SPI clock signal is on **D0**.
- **MOSI** - Microcontroller Out, Serial In is on **D1**.
- **MISO** - Microcontroller In, Serial Out is on **D2**.
- **CS0** - Chip Select is on **D3**. **This is not used by Blinka, instead use one of the GPIO pins from above (see example section).**

STEMMA QT

- **STEMMA QT connector** - The **new version** has a Stemma QT connector (https://adafru.it/HMF) that lets you plug & play any Stemma QT (https://adafru.it/HMF) or Qwiic devices, sensors and displays. To use: make sure the I2C switch is set to ON to use the STEMMA QT connector.

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, and SPI.

Make sure you've set the BLINKA_FT232H environment variable.

Installing Libraries for Breakouts

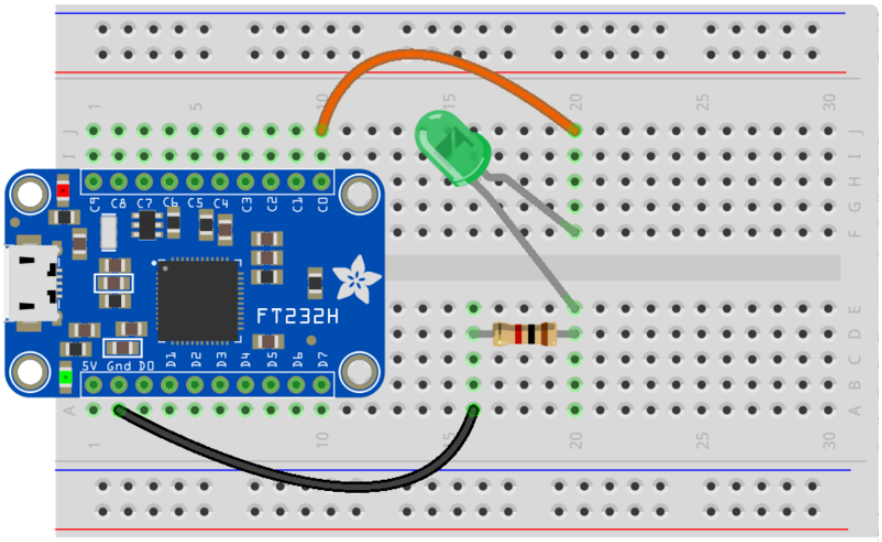
The general process for installing the CircuitPython library you are interested in will be the same as shown in the Python section of the Learn guide for your sensor. Just use pip3.

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.C0)
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:

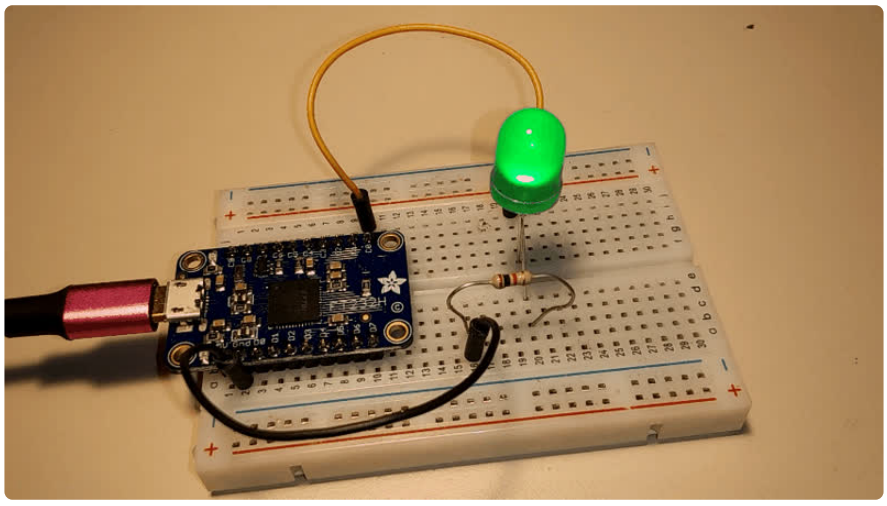
```
led.value = False
```

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.C0)
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:

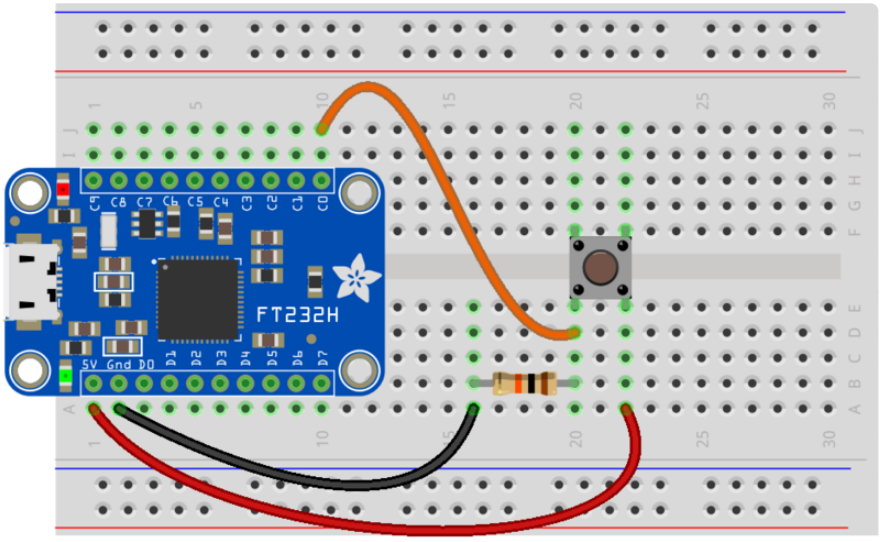
```
python3 blink.py
```



Digital Input

Let's read a button!

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



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.C0)
button.direction = digitalio.Direction.INPUT
```

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

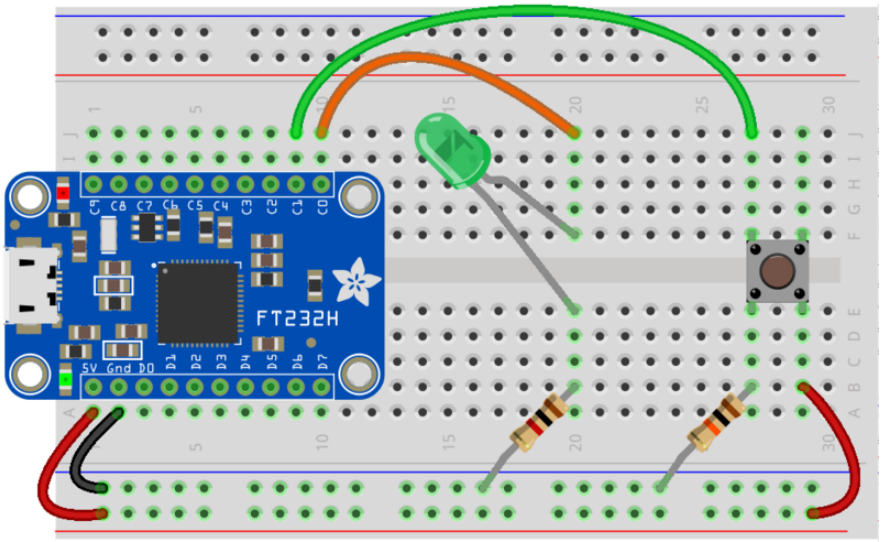
```
button.value
```

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

Digtal 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.



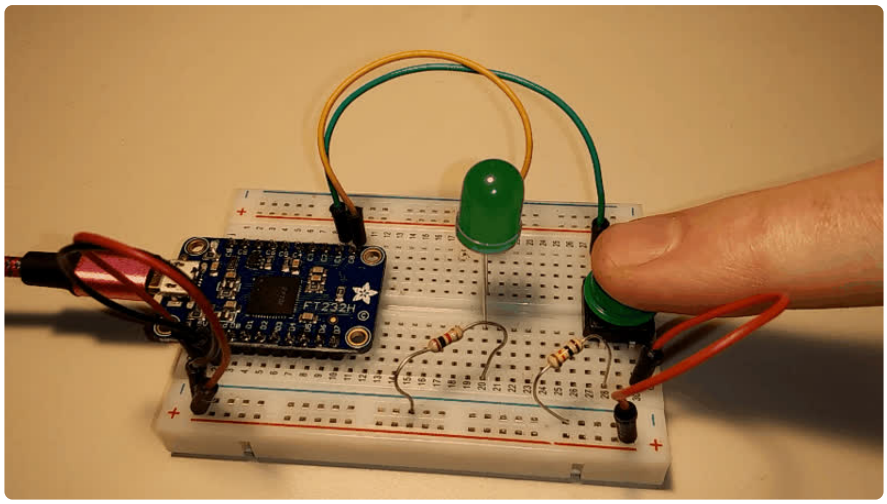
And here's the code.

```
import board
import digitalio

led = digitalio.DigitalInOut(board.C0)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.C1)
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
```

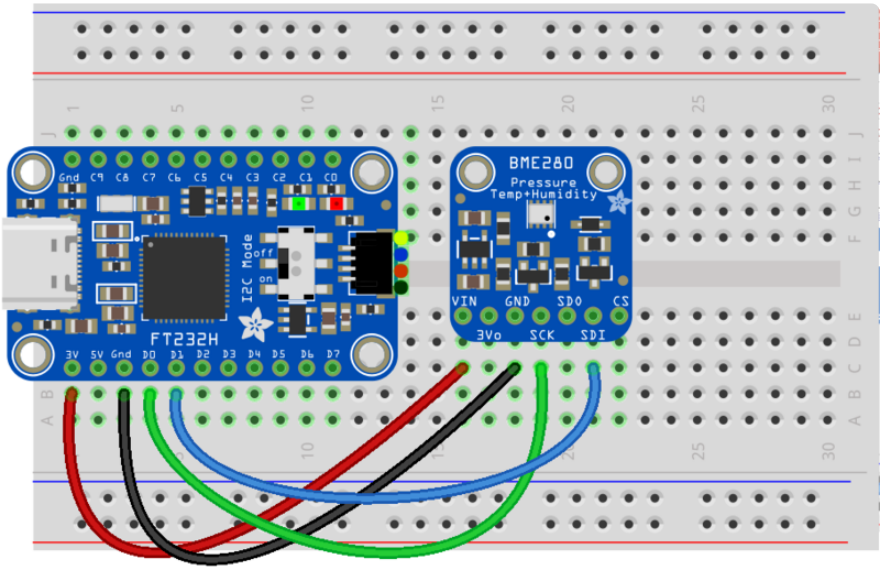
and the button should turn on the LED when pressed.



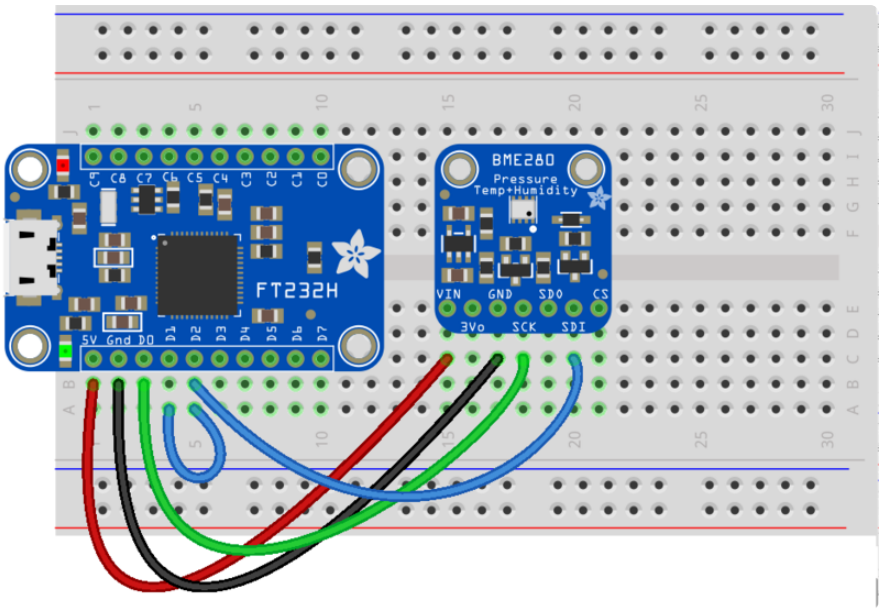
I2C

Let's talk to an I2C sensor.

We'll use the BME280 sensor (http://adafru.it/2652) which can read temperature, humidity, and pressure. Here's the wiring diagram:



For the older version of the board, you'll need to jumper D1 and D2 together:



If you have the original board version, don't forget to jumper the D1 and D2 pins together.

If you have the new board version, don't forget to set the I2C MODE switch to ON.

Install BME280 Library

To install the BME280 library, run the following:

```
sudo pip3 install adafruit-circuitpython-bme280
```

Note that this step is the same as shown in the main BME280 guide (https://adafru.it/FUP). You would do the same thing for any other sensor.

Run Example

Now we can run the simple test example from the library. Here's the code:

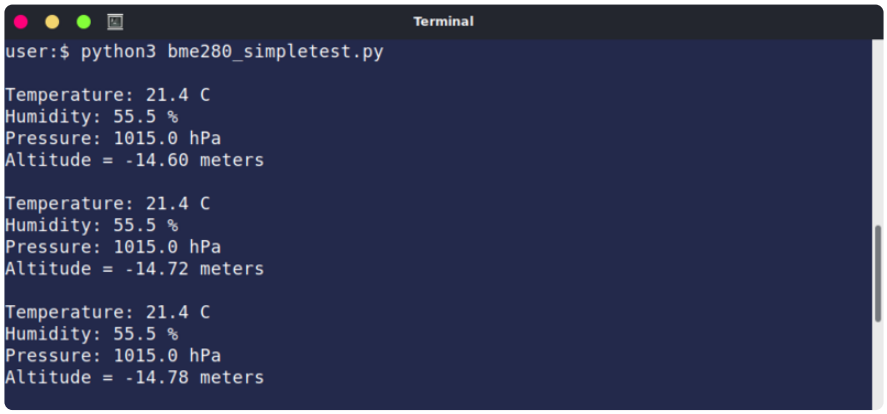
```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
from adafruit_bme280 import basic as adafruit_bme280

# Create sensor object, using the board's default I2C bus.
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
```

```
# OR create sensor object, using the board's default SPI bus.
# spi = board.SPI()
# bme_cs = digitalio.DigitalInOut(board.D10)
# bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, bme_cs)

# change this to match the location's pressure (hPa) at sea level
bme280.sea_level_pressure = 1013.25

while True:
    print("\nTemperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %%" % bme280.relative_humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
    print("Altitude = %0.2f meters" % bme280.altitude)
    time.sleep(2)
```

Copy and save the above code and then run it with:

```
python3 bme280_simpletest.py
```

and you should see it print out sensor readings over and over:



Using STEMMA QT

The new version of the FT232H features a STEMMA QT (https://adafru.it/JqB) connector. This makes connecting to newer breakouts that also have a STEMMA QT connector super easy. Just use a STEMMA QT cable:

STEMMA QT / Qwiic JST SH 4-Pin Cable - 200mm Long

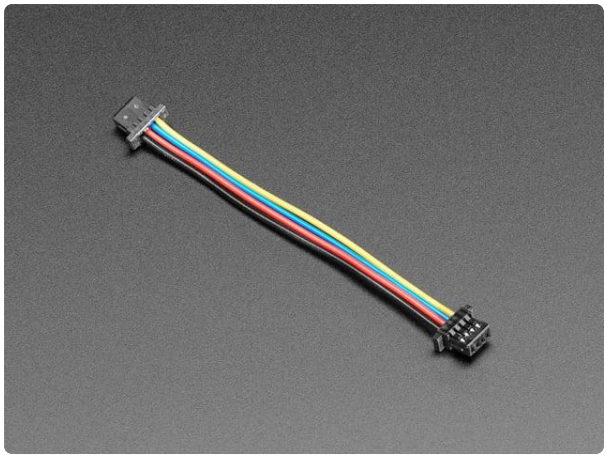
This 4-wire cable is a little over 200mm / 7.8" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...

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

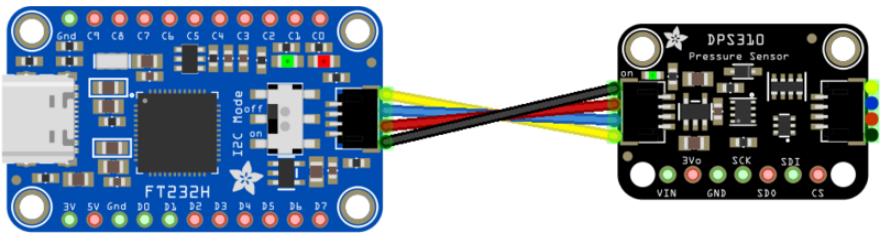


STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

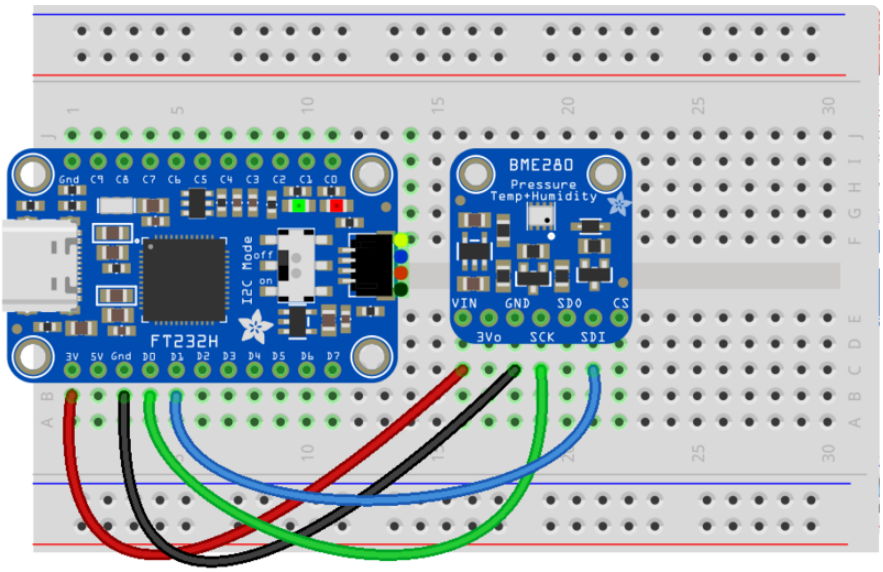
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

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

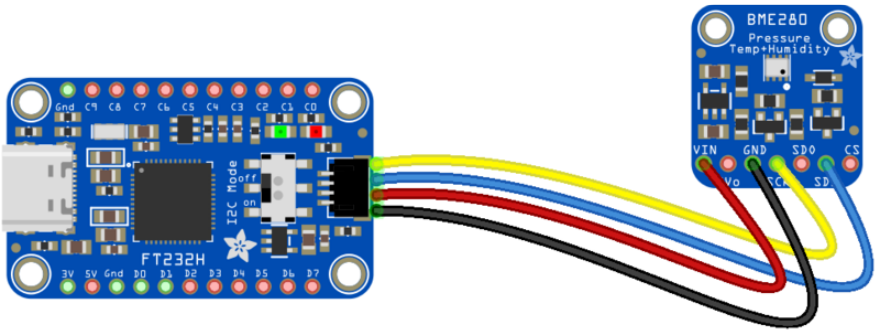
And then you can just wire the FT232H and the STEMMA QT breakout directly together. You don't even need a breadboard or soldering.



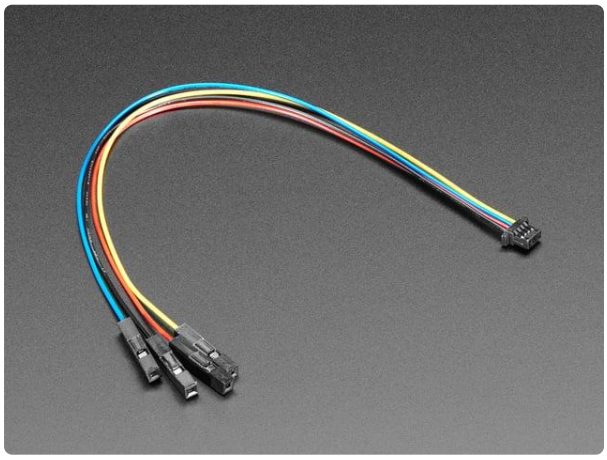
For breakout boards without a STEMMA QT connector, you can either use the header pins on both boards, that is, don't even use the STEMMA QT connector:



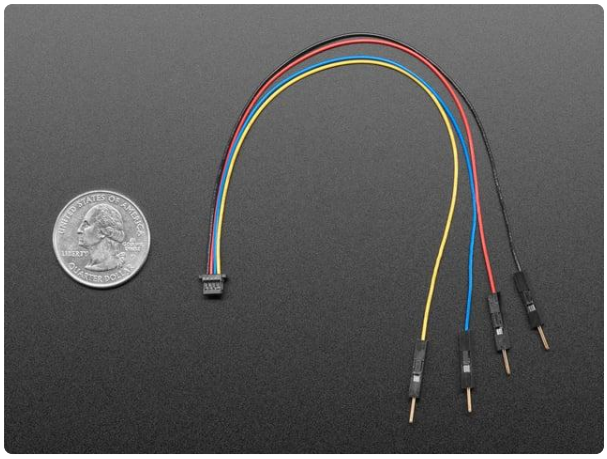
Or use the STEMMA QT connector on the FT232H breakout along with a STEMMA QT cable with one end of the cable cut off. You can then terminate the cut off end to the breakout in whatever way works best for your application.



There are also some STEMMA QT cables with header connectors which might be useful:



[STEMMA QT / Qwiic JST SH 4-pin Cable with Premium Female Sockets]
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium female headers on the other. Compared with the chunkier...
https://www.adafruit.com/product/4397

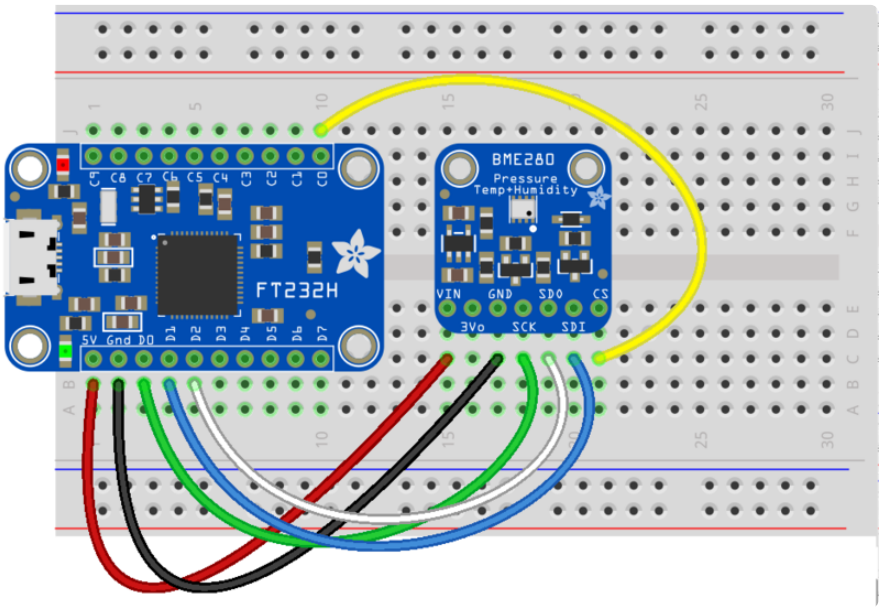


STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other. Compared with the...
https://www.adafruit.com/product/4209

SPI

Let's talk to a SPI sensor.

We can use the BME280 again, since it supports both I2C and SPI. Here's the SPI wiring:



If you have the new version of the board, make sure the I2C MODE switch is OFF for SPI usage.

Install BME280 Library

To install the BME280 library, run the following:

```
sudo pip3 install adafruit-circuitpython-bme280
```

Note that this step is the same as shown in the main BME280 guide (https://adafru.it/FUP). You would do the same thing for any other sensor.

Run Example

We can run the simple test example (https://adafru.it/FUQ) from the library. It is set up to use I2C by default, so we need to comment out the I2C stuff and uncomment the SPI stuff.

We also add an import for **digitalio** (since it wasn't in there) and change the CS pin to the one we are using with the FT232H - **C0**.

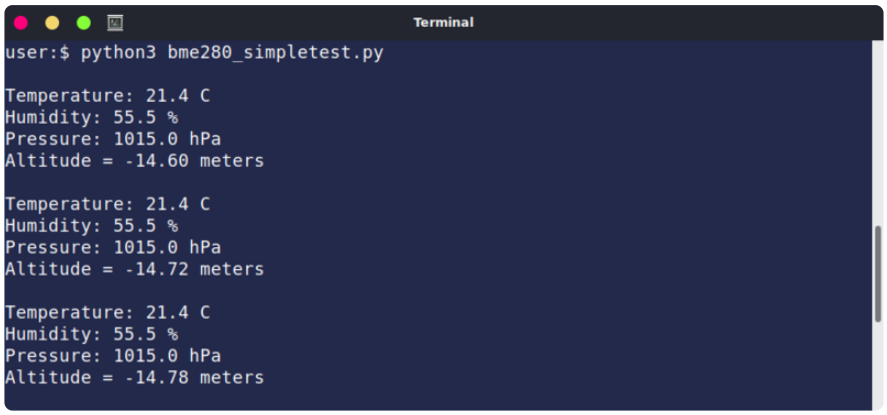
Here's the complete code:

```
import time

import board
import busio
import digitalio
import adafruit_bme280

# Create library object using our Bus I2C port
#i2c = busio.I2C(board.SCL, board.SDA)
#bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# OR create library object using our Bus SPI port
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
bme_cs = digitalio.DigitalInOut(board.C0)
bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, bme_cs)

# change this to match the location's pressure (hPa) at sea level
bme280.sea_level_pressure = 1013.25

while True:
    print("\nTemperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %%" % bme280.humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
    print("Altitude = %0.2f meters" % bme280.altitude)
    time.sleep(2)
```

Copy and save that code and then run it with:

```
python3 bme280_simpletest.py
```

and you should see it print out sensor readings over and over:



NeoPixels

NeoPixels (https://adafru.it/Gbx)

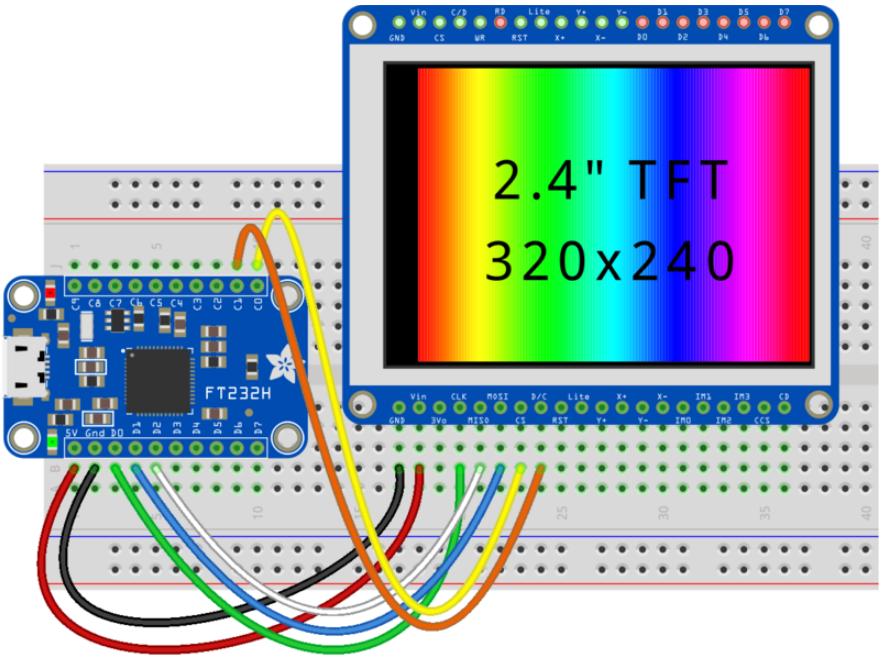
TFT Display

Let's draw to a display!

Here we'll use SPI to talk to a TFT display. And since we are on a powerful desktop PC, we can even use Pillow (https://adafru.it/FUR) (PIL) to do the graphical heavy lifting.

Connect the Display

Here is the wiring diagram for connecting a 2.4" TFT LCD display (http://adafru.it/2478) via SPI. We aren't using touch, so don't need to worry about those pins.



Install Pillow/PIL

We'll just point you to the official Pillow/PIL page so you can follow the install instruction there.

Pillow Homepage

https://adafru.it/FUR

You should be able to just use pip (https://adafru.it/FUT) though.

Install CircuitPython RGB Display Library

This library (https://adafru.it/u1C) provides support for the TFT display. To install, run:

```
sudo pip3 install adafruit-circuitpython-rgb-display
```

Example Image

In this example, we'll load and display an image. Download the image below and save it to the same directory where the Python code will be stored. Make sure it is named **blinka.bmp**.

blinka.bmp

https://adafru.it/FUU

The Code

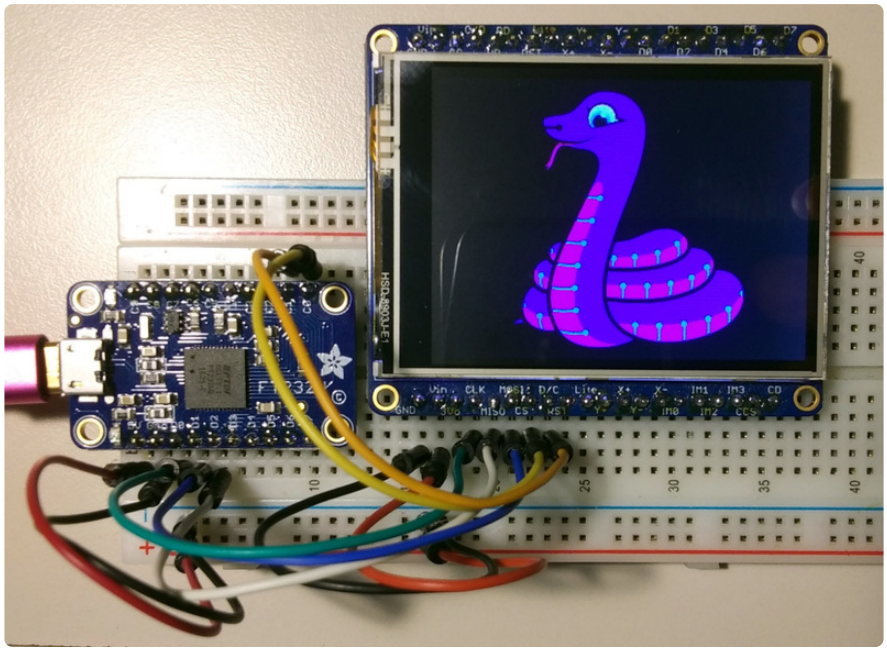
And here is the complete code listing to load and display the BMP file.

```
import board
import digitalio
import adafruit_rgb_display.ili9341 as ili9341
from PIL import Image

# Setup display
cs_pin = digitalio.DigitalInOut(board.C0)
dc_pin = digitalio.DigitalInOut(board.C1)
disp = ili9341.ILI9341(board.SPI(), cs=cs_pin, dc=dc_pin, baudrate=64000000)

# Load image and convert to RGB
image = Image.open('blinka.bmp').convert('RGB')

# Display it (rotated by 90 deg)
disp.image(image, 90)
```

Save the above with a filename like **tft_image.py** and then run it with:

```
python3 tft_image.py
```

and boom! you should get a happy little Blinka showing up on your display.



Fancier Demo for Linux

There's a lot of power in Pillow/PIL. It can do more than just load and display images. Here's another example that creates a Draw object, uses a TTF font, and then grabs some stats from the PC and displays them.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This will show some Linux Statistics on the attached display. Be sure to adjust
to the display you have connected. Be sure to check the learn guides for more
usage information.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!
"""

import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                           # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                          # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                          # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,      # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,   # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                        # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                        # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long
```

```
# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height

image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)

# First define some constants to allow easy positioning of text.
padding = -2
x = 0

# Load a TTF font.  Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24)

while True:
    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-
usage-disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d' ' -f1"
    IP = "IP: " + subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\",
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'df -h | awk \'$NF=="/"{printf "Disk: %d/%d GB  %s", $3,$2,$5}\''
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk '{printf \"CPU Temp: %.
1f C\", $(NF-0) / 1000}'"  # pylint: disable=line-too-long
    Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.
    y = padding
    draw.text((x, y), IP, font=font, fill="#FFFFFF")
    y += font.getsize(IP)[1]
    draw.text((x, y), CPU, font=font, fill="#FFFF00")
    y += font.getsize(CPU)[1]
    draw.text((x, y), MemUsage, font=font, fill="#00FF00")
    y += font.getsize(MemUsage)[1]
    draw.text((x, y), Disk, font=font, fill="#0000FF")
    y += font.getsize(Disk)[1]
    draw.text((x, y), Temp, font=font, fill="#FF00FF")

    # Display image.
    disp.image(image)
    time.sleep(0.1)
```

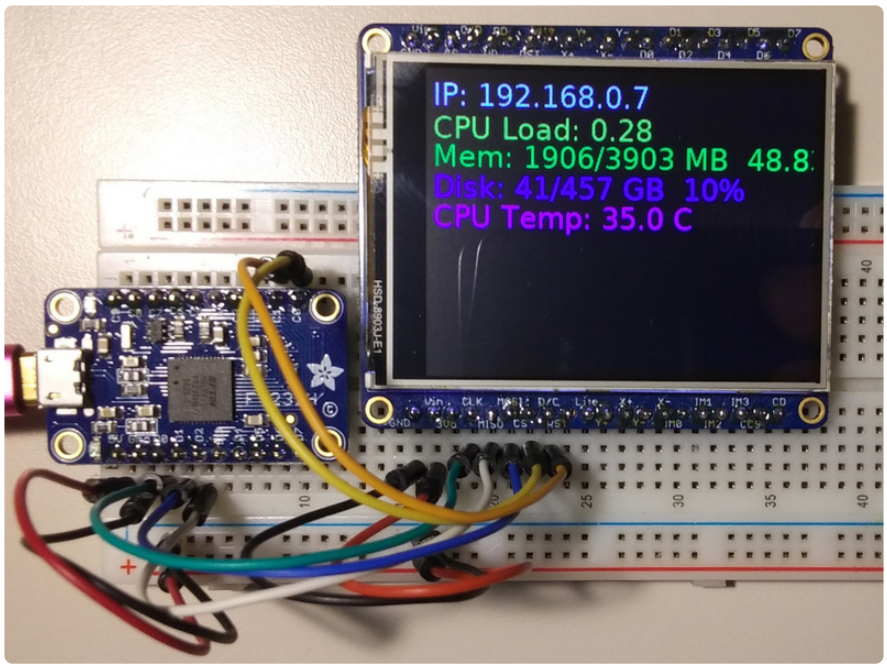
The example has import and code initialization for multiple displays. Uncomment the one for the display you are using, and comment out all others. If you are using the

same 2.4" TFT (ILI9341) from above, then you don't need to change anything. The example uses that as default.

You will need to change the pins being used to match those on the FT232H. Find these lines of code and make things changes:

```
# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.C0)
dc_pin = digitalio.DigitalInOut(board.C1)
reset_pin = None
```

With those changes in place, you should be able to run it and get some system stats showing up.



And it will update in near real-time.

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). You need at least version 3.4. This should have been taken care of (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 then 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) will give you `AnalogIn` objects. For PWM outputs, try the PCA9685 (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) 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"

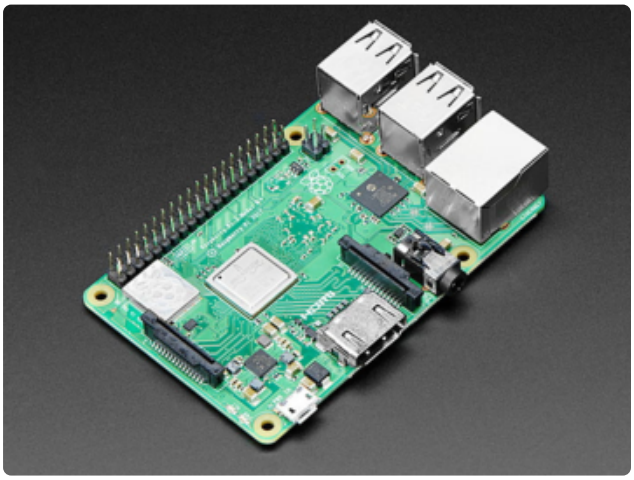
It looks like **libgpiod** may not be installed on your board.

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"

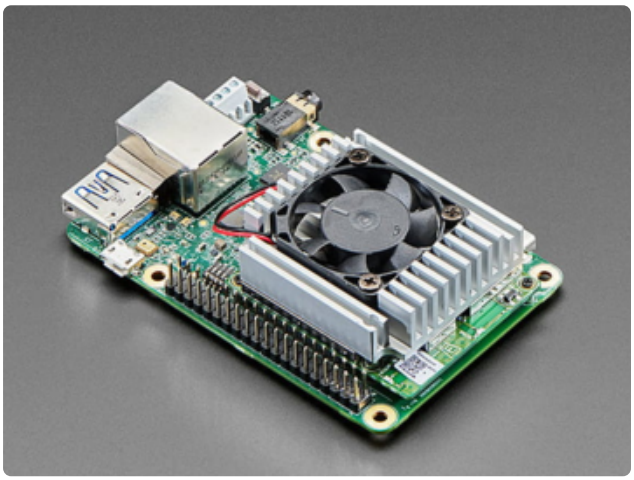
Be sure you have the latest libgpiod.py script and run it with the `-l` or `--legacy` flag:

```
sudo python3 libgpiod.py --legacy
```



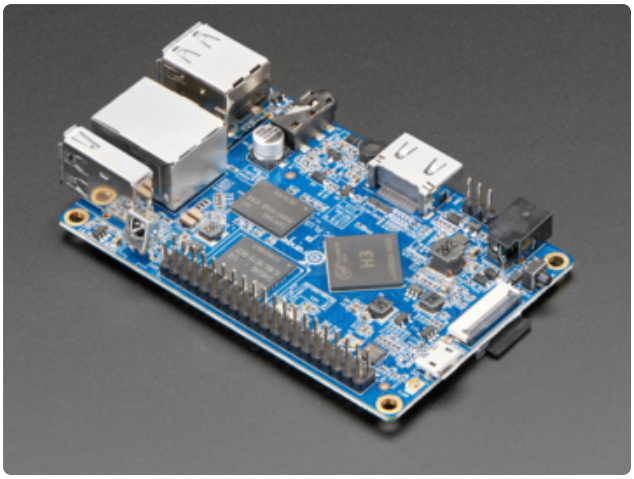
All Raspberry Pi Computers Have:

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)



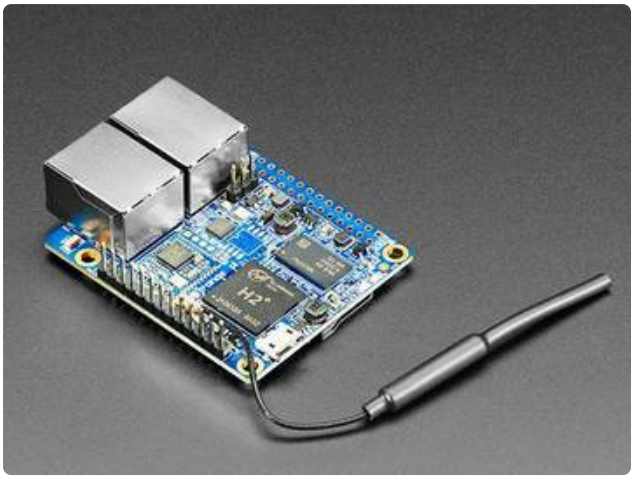
Google Coral TPU Dev Boards Have:

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)



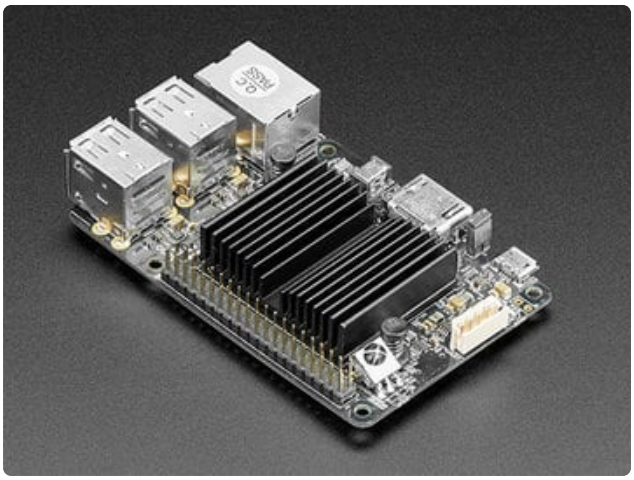
Orange Pi PC Plus Boards Have:

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)



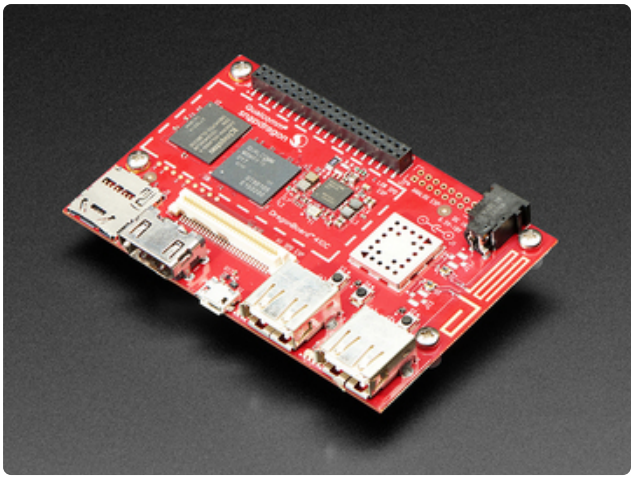
Orange Pi R1 Boards Have:

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)

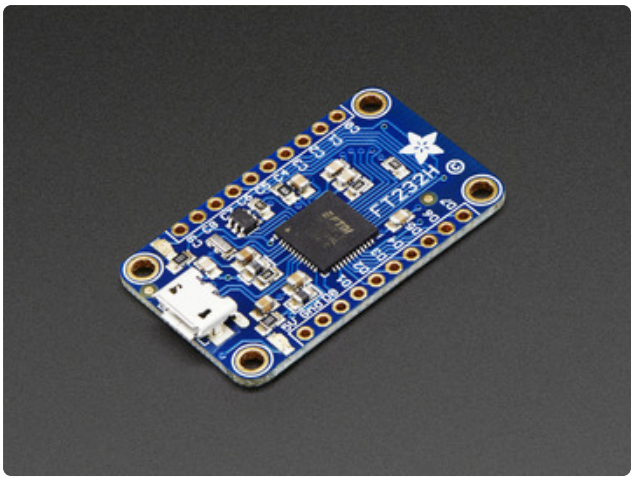


DragonBoard 410c Boards Have:

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:

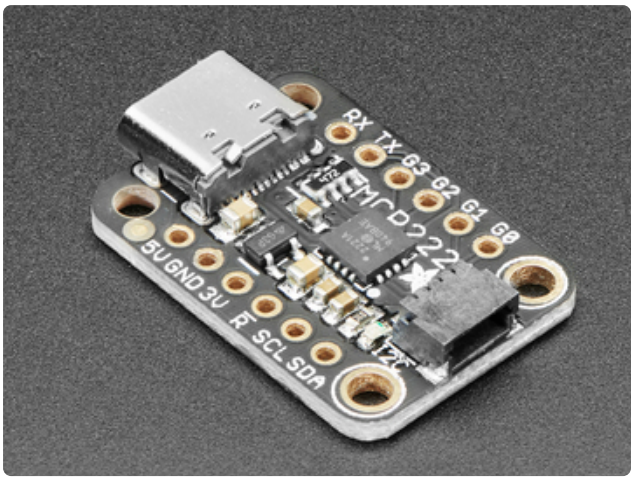
2 x I2C port with **busio**
2 x SPI ports with **busio**
2 x UART port with **serial** - note one of these is shared with the hardware console
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



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