# Table of Contents

## Overview
- Why CircuitPython?
- CircuitPython on Microcontrollers
- CircuitPython Libraries on Desktop Linux

## Running CircuitPython Code without CircuitPython
- 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

## CircuitPython & Odroid
- CircuitPython Libraries on Linux & Odroid
- Wait, isn’t there already something that does this - libgpiod?
- What about other Linux SBCs?

## Initial Setup
- Install ARMBian on your Odroid C2
- Logging in
- Update Your Packages
- Install Python and set Python 3 to Default
- Update Python Packages
- Install libgpiod
- Enable UART, I2C and SPI
- Install Python Libraries

## Digital I/O
- Parts Used
- Wiring
- Blinky Time!
- Button It Up

## I2C Sensors & Devices
- Parts Used
- Wiring
- Install the CircuitPython BME280 Library
- Run that code!

## UART / Serial
- The Easy Way - An External USB-Serial Converter
- The Hard Way - Using Built-in UART
- Install the CircuitPython GPS Library
- Run that code!

## More To Come!
FAQ & Troubleshooting

- Update Blinka/Platform Libraries
Overview

Here at Adafruit we're always looking for ways to make making easier - whether that's making breakout boards for hard-to-solder sensors or writing libraries to simplify motor control. Our new favorite way to program is CircuitPython.

Why CircuitPython?

CircuitPython is a variant of MicroPython, a very small version of Python that can fit on a microcontroller. Python is the fastest-growing programming language. It's taught in schools, used in coding bootcamps, popular with scientists and of course programmers at companies use it a lot!

CircuitPython adds the Circuit part to the Python part. It lets you program in Python and talk to Circuitry like sensors, motors, and LEDs!

CircuitPython on Microcontrollers

CircuitPython runs on microcontroller boards, such as our Feather, Metro, QT Py, and ItsyBitsy boards, using a variety of chips, such as the MicroChip SAMD21 SAMD51, the
Raspberry Pi RP2040, the Nordic nRF52840, and the Espressif ESP32-S2 and ESP32-S3.

All of these chips have something in common - they are microcontrollers with hardware peripherals like SPI, I2C, ADCs etc. We squeeze Python into ‘em and can then make the project portable.

But...sometimes you want to do more than a microcontroller can do. Like HDMI video output, or camera capture, or serving up a website, or just something that takes more memory and computing than a microcontroller board can do...

**CircuitPython Libraries on Desktop Linux**

By adding a software layer, you can use CircuitPython hardware control capabilities with "regular Python", as found on your desktop or single-board Linux computer/ There are tons of projects, libraries and example code for CircuitPython on microcontrollers, and thanks to the flexibility and power of Python its' pretty easy to get that code working on micro-computers like the Raspberry Pi or other single-board Linux computers with GPIO pins available.

You'll use a special library called adafruit_blinka () (named after Blinka, the CircuitPython mascot ()) that provides a layer that translates the CircuitPython hardware API to whatever library the Linux board provides. For example, on Raspberry Pi we use the python RPi.GPIO () library. For any I2C interfacing we'll use ioctl messages to the /dev/i2c device. For SPI we'll use the spidev python library, etc. These details don't matter so much because they all happen underneath the adafruit_blinka layer.

The upshot is that most code we write for CircuitPython will be instantly and easily runnable on Linux computers like Raspberry Pi.

In particular, you'll be able to use all of our device drivers - the sensors, led controllers, motor drivers, HATs, bonnets, etc. And nearly all of these use I2C or SPI!

The rest of this guide describes how to install and set up Blinka, and then how to use it to run CircuitPython code to control hardware.
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 () and FT232H () breakout boards, and Raspberry Pi Pico boards running the u2if software (). 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 (). 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` 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 () 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`.
- 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`.

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 () for more information about the bundles, and also see the Libraries page on circuitPython.org.
CircuitPython & Odroid

CircuitPython Libraries on Linux & Odroid

The next obvious step is to bring CircuitPython ease of use back to 'desktop Python'. We've got tons of projects, libraries and example code for CircuitPython on microcontrollers, and thanks to the flexibility and power of Python its pretty easy to get it working with micro-computers like Odroid or other 'Linux with GPIO pins available' single board computers.

We'll use a special library called adafruit_blinka () (named after Blinka, the CircuitPython mascot ()) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. For example, on Odroid we use the python libgpiod bindings. For any I2C interfacing we'll use ioctl messages to the /dev/i2c device. These details don't matter so much because they all happen underneath the adafruit_blinka layer.

The upshot is that any code we have for CircuitPython will be instantly and easily runnable on Linux computers like Odroid.

In particular, we'll be able to use all of our device drivers - the sensors, led controllers, motor drivers, HATs, bonnets, etc. And nearly all of these use I2C or SPI!
Wait, isn't there already something that does this - libgpiod?

*libgpiod is a python hardware interface class* that works on Odroid, it works just fine for I2C, SPI and GPIO but doesn't work with our drivers as its a different API.

By letting you use CircuitPython libraries on Odroid via adafruit_blinka, you can unlock all of the drivers and example code we wrote! And you can keep using libgpiod if you like. We save time and effort so we can focus on getting code that works in one place, and you get to reuse all the code we've written already.

What about other Linux SBCs?

Yep! Blinka can easily be updated to add other boards. We've started with the one we've got, so we could test it thoroughly. If you have other SBC board you'd like to adapt check out the adafruit_blinka code on github, pull requests are welcome as there's a ton of different Linux boards out there!

**Initial Setup**

Right now, Blinka only supports the Odroid C2 (because that's the only board we've got for testing).

At any time after armbian is installed, it's easy to tell what board you have: simply cat /etc/armbian-release and look for `BOARD_NAME`.

```
$ cat /etc/armbian-release
BOARD=odroidc2
BOARD_NAME="Odroid C2"
BOARDFAMILY=odroidc2
VERSION=5.53
LINUXFAMILY=odroidc2
BRANCH=next
ARCH=arm64
IMAGE_TYPE=stable
BOARD_TYPE=conf
INITIAL_ARCH=arm64
KERNEL_IMAGE_TYPE=Image
```

**Install ARMBian on your Odroid C2**

We're only going to be using armbian, other distros could be made to work but you'd probably need to figure out how to detect the platform since we rely on `/etc/armbian-release` existing. Using other operating systems and CircuitPython is your call, we cannot provide support for that.
Download and install the latest armbian, for example we’re using https://www.armbian.com/odroid-c2/.

There’s some documentation to get started at https://docs.armbian.com/User-Guide_Getting-Started/.

Blinka only supports ARMBian because that's the most stable OS we could find and it's easy to detect which board you have.

Logging in

We've found the easiest way to connect is through a console cable, wired to the UART Serial port, and then on your computer, use a serial monitor at 115200 baud.

USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi
The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...
https://www.adafruit.com/product/954
Once powered correctly and with the right SD card you should get a login prompt.

After logging in you may be asked to create a new username, we recommend pi - if our instructions end up adding gpio access for the pi user, you can copy and paste them.

Once installed, you may want to enable mdns so you can ssh `pi@odroidc2.local` instead of needing to know the IP address:

```
sudo apt-get install avahi-daemon
```

then reboot

**Update Your Packages**

Run the standard updates:
Install Python and set Python 3 to Default

There's a few ways to do this. Since Python 2 is no longer installed by default, we recommend something like this:

```
sudo apt-get install -y python3 git python3-pip
sudo update-alternatives --install /usr/bin/python python $(which python3) 2
```

This will make it so typing `python` runs `python3`.

Update Python Packages

```
sudo pip3 install --upgrade setuptools
```

Update all your Python 3 packages with

```
pip3 freeze -l | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip3 install -U
```

and

```
sudo bash
pip3 freeze -l | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip3 install -U
```

Install libgpiod

libgpiod is what we use for gpio toggling. To install run this command:

```
sudo apt-get install libgpiod2
pip3 install gpiod
```
After installation, you should be able to `import gpiod` from within Python 3:

```
root@odroidc2:/home/pi# sudo apt-get install libgpiod2 python3-libgpiod
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libgpiod2 python3-libgpiod
0 to upgrade, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 50.4 kB of archives.
After this operation, 206 kB of additional disk space will be used.
Get:1 http://ports.ubuntu.com focal/universe arm64 libgpiod2 arm64 1.4.1-4 [32.0 kB]
Get:2 http://ports.ubuntu.com focal/universe arm64 python3-libgpiod arm64 1.4.1-4 [18.4 kB]
Fetched 50.4 kB in 0s (52.0 kB/s)
Selecting previously unselected package libgpiod2:arm64.
(Reading database ... 59922 files and directories currently installed.)
Preparing to unpack .../libgpiod2_1.4.1-4_arm64.deb ...
Unpacking libgpiod2:arm64 (1.4.1-4) ...
Selecting previously unselected package python3-libgpiod:arm64.
Preparing to unpack .../python3-libgpiod_1.4.1-4_arm64.deb ...
Unpacking python3-libgpiod:arm64 (1.4.1-4) ...
Setting up libgpiod2:arm64 (1.4.1-4) ...
Setting up python3-libgpiod:arm64 (1.4.1-4) ...
Processing triggers for libc-bin (2.31-0ubuntu0.7) ...
```

Enable UART, I2C and SPI

A vast number of our CircuitPython drivers use UART, I2C and SPI for interfacing, so you'll want to get those enabled.

You only have to do this once per board, unfortunately by default all three interfaces are disabled!

Install the support software with:

```
sudo apt-get install -y python3-smbus python3-dev i2c-tools
sudo adduser pi i2c
```
The Odroid C2 does not have a hardware peripheral, but I2C is enabled by default, so if you run `ls /dev/i2c*`

You should see at least one i2c device.

Because the Odroid C2 is a newer board, currently /dev/ttyAML1 isn't working properly.

The UART Serial Port on the Odroid C2 is connected to /dev/ttyAML0. To enable the GPIO UART, edit /boot/armbianEnv.txt and add this line to the end.

```
overlays=uartA
```

After you have rebooted, verify that /dev/ttyAML1 is now enabled by typing:
ls /dev/ttyA*

```
pi@odroidc2:~$ ls /dev/ttyA*
/dev/ttyAML0  /dev/ttyAML1
pi@odroidc2:~$  
```

Even though the Odroid C2 has SPI, both of the hardware CS lines are in use, making it unavailable

## Install Python Libraries

Now you're ready to install all the Python support.

Run the following command to install adafruit_blinka:

```
sudo pip3 install adafruit-blinka  
```

```
pi@odroidc2:~$ sudo pip3 install adafruit-blinka
Collecting adafruit-blinka
  Collecting sysv-ipc; platform_system != "Windows" (from adafruit-blinka)
    Using cached https://files.pythonhosted.org/packages/b0/34/e8e6b4ee910d3582c557603456f168b1d84b85b5110360679e7a7e65e/adafruit-blinka-1.0.4.tar.gz
  Collecting Adafruit-PlatformDetect (from adafruit-blinka)
    Using cached https://files.pythonhosted.org/packages/b3/10/38ad70e947e659929e45d0c8ec840c045d7f0ba6f991d4d49/Adafruit-PlatformDetect-1.0.4.tar.gz
Building wheels for collected packages: sysv-ipc, spidev, Adafruit-PlatformDetect
  Running setup.py bdist_wheel for sysv-ipc ... done
  Stored in directory: /root/.cache/pip/wheels/ac/d6/5b/727b8b8e350038cf49d92796f9b60f0d83f31bd1f1df630b
  Running setup.py bdist_wheel for spidev ... done
  Stored in directory: /root/.cache/pip/wheels/1b/5f/9b/076c03edbb7f137b49b8b0a91d5b51e4d6b972b7d
  Running setup.py bdist_wheel for Adafruit-PlatformDetect ... done
  Stored in directory: /root/.cache/pip/wheels/3c/fc/26/e3f79b4bfa6d5bce9f3e17b5
  Running setup.py bdist_wheel for Adafruit-PureIO ... done
  Stored in directory: /root/.cache/pip/wheels/0d/fa/4e/e88870d9c8a049290e91e72656c71e37f367997
Successfully built sysv-ipc spidev Adafruit-PlatformDetect Adafruit-PureIO
Installing collected packages: sysv-ipc, spidev, Adafruit-PlatformDetect, Adafruit-PureIO
Successfully installed Adafruit-PlatformDetect-1.0.4 Adafruit-PureIO-0.2.3 adafruit-blinka

The computer will install a few different libraries such as adafruit-pureio (our ioctl-only i2c library), Adafruit-GPIO (for detecting your board) and of course adafruit-blinka.

That's pretty much it! You're now ready to test.

Create a new file called blinkatest.py with nano or your favorite text editor and put the following in:
import board
import digitalio
import busio

print("Hello blinka!")

# Try to great a Digital input
pin = digitalio.DigitalInOut(board.D7)
print("Digital IO ok!")

# Try to create an I2C device
i2c = busio.I2C(board.SCL, board.SDA)
print("I2C ok!")

# Try to create an SPI device
#spi = busio.SPI(board.SCLK, board.MOSI, board.MISO)
#print("SPI ok!")

print("done!")

Save it and run at the command line with

sudo python3 blinkatest.py

You should see the following, indicating digital i/o, I2C and SPI all worked

---

**Digital I/O**

The first step with any new hardware is the 'hello world' of electronics - blinking an LED. This is very easy with CircuitPython and Odroid. We'll extend the example to also show how to wire up a button/switch.
## Parts Used

Any old LED will work just fine as long as it’s not an IR LED (you can’t see those) and a 470 to 2.2K resistor.
**Diffused Blue 10mm LED (25 pack)**

Need some big indicators? We are big fans of these huge diffused blue LEDs. They are really bright so they can be seen in daytime, and from any angle. They go easily into a breadboard...

https://www.adafruit.com/product/847

**Through-Hole Resistors - 470 ohm 5% 1/4W - Pack of 25**

ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you! This is a 25 Pack of...

https://www.adafruit.com/product/2781

Some tactile buttons or switches:

**Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack**

Medium-sized clicky momentary switches are standard input "buttons" on electronic projects. These work best in a PCB but

https://www.adafruit.com/product/1119

We recommend using a breadboard and some female-male wires.
Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...
https://www.adafruit.com/product/826

You can use a Cobbler to make this a little easier, the pins will be labeled according to Raspberry Pi names so just check the Odroid name!

Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3
The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...

Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

Wiring

Connect the Odroid Ground pin to the blue ground rail on the breadboard.

- Connect one side of the tactile switch to Odroid GPIO 249 which is also called D 7
• Connect a ~10K pull up resistor from D7 to 3.3V
• Connect the other side of the tactile switch to the ground rail
• Connect the longer/positive pin of the LED to Odroid GPIO 238 which is also called D1
• Connect the shorter/negative pin of the LED to a 470ohm to 2.2K resistor, the other side of the resistor goes to ground rail

Double-check you have the right wires connected to the right location, it can be tough to keep track of GPIO pins as there are forty of them!

No additional libraries are needed so we can go straight on to the example code

However, we recommend running a pip3 update!

```bash
sudo pip3 install --upgrade adafruit_blinka
```

**Blinky Time!**

The finish line is right up ahead, lets start with an example that blinks the LED on and off once a second (half a second on, half a second off):

```python
import time
import board
import digitalio
```
print("hello blinky!")
led = digitalio.DigitalInOut(board.D1)
led.direction = digitalio.Direction.OUTPUT

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

Verify the LED is blinking. If not, check that it's wired to GPIO 238 or D1, the resistor is installed correctly, and you have a Ground wire to the Odroid.

Type Control-C to quit

**Button It Up**

Now that you have the LED working, lets add code so the LED turns on whenever the button is pressed

```python
import time
import board
import digitalio

print("press the button!")

led = digitalio.DigitalInOut(board.D1)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.D7)
button.direction = digitalio.Direction.INPUT
# use an external pullup since we don't have internal PU's
#button.pull = digitalio.Pull.UP

while True:
    led.value = not button.value # light when button is pressed!
```

Press the button - see that the LED lights up!

Type Control-C to quit

**I2C Sensors & Devices**

The most popular electronic sensors use I2C to communicate. This is a 'shared bus' 2 wire protocol, you can have multiple sensors connected to the two SDA and SCL pins as long as they have unique addresses ([check this guide for a list of many popular devices and their addresses](#))
Lets show how to wire up a popular BME280. This sensor provides temperature, barometric pressure and humidity data over I2C.

We're going to do this in a lot more depth than our guide pages for each sensor, but the overall technique is basically identical for any and all I2C sensors.

Honestly, the hardest part of using I2C devices is figuring out the I2C address and which pin is SDA and which pin is SCL!

Parts Used

Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor
Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...
https://www.adafruit.com/product/2652

We recommend using a breadboard and some female-male wires.

Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...
https://www.adafruit.com/product/826

You can use a Cobbler to make this a little easier, the pins are then labeled!
Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3
The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...

Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

Wiring

• Connect the Odroid 3.3V power pin to Vin
• Connect the Odroid GND pin to GND
• Connect the Pi SDA pin to the BME280 SDI
• Connect the Pi SCL pin to the BME280 SCK
Double-check you have the right wires connected to the right location, it can be tough to keep track of Pi pins as there are forty of them!

After wiring, we recommend running I2C detection with `sudo i2cdetect -y 0` to verify that you see the device, in this case its address 77.

Install the CircuitPython BME280 Library

OK onto the good stuff, you can now install the Adafruit BME280 CircuitPython library.

As of this writing, not all libraries are up on PyPI so you may want to search before trying to install. Look for circuitpython and then the driver you want.
(If you don't see it you can open up a github issue on circuitpython to remind us ()!)

Once you know the name, install it with

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

You'll notice we also installed a dependency called adafruit-circuitpython-busdevice. This is a great thing about pip, if you have other required libraries they'll get installed too!

We also recommend an adafruit-blinka update in case we've fixed bugs:

```bash
sudo pip3 install --upgrade adafruit_blinka
```

Run that code!

The finish line is right up ahead. You can now run one of the (many in some cases) example scripts we've written for you.
Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be https://github.com/adafruit/Adafruit_CircuitPython_BME280/tree/master/examples

As of this writing there's only two examples. Here's the first one:

```python
import time
import board
import busio
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.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.humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
    print("Altitude = %0.2f meters" % bme280.altitude)
    time.sleep(2)
```

Save this code to your Odroid by copying and pasting it into a text file, downloading it directly from the Odroid, etc.

Then in your command line run

```
sudo python3 bme280_simpletest.py
```
The code will loop with the sensor data until you quit with a Control-C

Here's the second example:

```python
###
Example showing how the BME280 library can be used to set the various parameters supported by the sensor.
Refer to the BME280 datasheet to understand what these parameters do
###

code

import time
import board
import busio
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.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
bme280.mode = adafruit_bme280.MODE_NORMAL
bme280.standby_period = adafruit_bme280.STANDBY_TC_500
bme280.iir_filter = adafruit_bme280.IIR_FILTER_X16
bme280.overscan_pressure = adafruit_bme280.OVERSCAN_X16
bme280.overscan_humidity = adafruit_bme280.OVERSCAN_X1
bme280.overscan_temperature = adafruit_bme280.OVERSCAN_X2

#The sensor will need a moment to gather initial readings
time.sleep(1)

while True:
    print("Temperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %" % bme280.humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
```

©Adafruit Industries
Save this code to your Odroid by copying and pasting it into a text file, downloading it directly from the Odroid, etc.

Then in your command line run

```
sudo python3 bme280_normal_mode.py
```

(The code will loop with the sensor data until you quit with a Control-C)

That's it! Now if you want to read the documentation on the library, what each function does in depth, visit our readthedocs documentation at


---

**UART / Serial**

After I2C and SPI, the third most popular "bus" protocol used is serial (also sometimes referred to as 'UART'). This is a non-shared two-wire protocol with an RX line, a TX line and a fixed baudrate. The most common devices that use UART are GPS units, MIDI interfaces, fingerprint sensors, thermal printers, and a scattering of sensors.

One thing you'll notice fast is that most linux computers have minimal UARTs, often only 1 hardware port. And that hardware port may be shared with a console.
There are two ways to connect UART / Serial devices to your Odroid. The easy way, and the hard way.

We'll demonstrate wiring up & using an Ultimate GPS with both methods

Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates
We carry a few different GPS modules here in the Adafruit shop, but none that satisfied our every desire - that's why we designed this little GPS breakout board. We believe this is...
https://www.adafruit.com/product/746

The Easy Way - An External USB-Serial Converter

By far the easiest way to add a serial port is to use a USB to serial converter cable or breakout. They're not expensive, and you simply plug it into the USB port. On the other end are wires or pins that provide power, ground, RX, TX and maybe some other control pads or extras.

Here are some options, they have varying chipsets and physical designs but all will do the job. We'll list them in order of recommendation.

The first cable is easy to use and even has little plugs that you can arrange however you like, it contains a CP2102

USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi
The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<>Serial conversion chip and at...
https://www.adafruit.com/product/954

The CP2104 Friend is low cost, easy to use, but requires a little soldering, it has an '6-pin FTDI compatible' connector on the end, but all pins are broken out the sides

©Adafruit Industries
Both the FTDI friend and cable use classic FTDI chips, these are more expensive than the CP2104 or PL2303 but sometimes people like them!

FTDI Friend + extras
Long gone are the days of parallel ports and serial ports. Now the USB port reigns supreme! But USB is hard, and you just want to transfer your everyday serial data from a...
https://www.adafruit.com/product/284

FTDI Serial TTL-232 USB Cable
Just about all electronics use TTL serial for debugging, bootloading, programming, serial output, etc. But it's rare for a computer to have a serial port anymore. This is a USB to...
https://www.adafruit.com/product/70

There is also a GPS module with integrated serial available which works like the GPS breakout connected to the USB to TTL Serial cable.
Adafruit Ultimate GPS GNSS with USB - 99 channel w/10 Hz updates

The Ultimate GPS module you know and love has a glow-up to let it be easily used with any computer, not just microcontrollers! With the built-in USB-to-Serial converter, you...

https://www.adafruit.com/product/4279

You can wire up the GPS by connecting the following

- GPS Vin to USB 5V or 3V (red wire on USB console cable)
- GPS Ground to USB Ground (black wire)
- GPS RX to USB TX (green wire)
- GPS TX to USB RX (white wire)

Once the USB adapter is plugged in, you'll need to figure out what the serial port name is. You can figure it out by unplugging-replugging in the USB and then typing `dmesg | tail -10` (or just `dmesg`) and looking for text like this:

```
pi@rodnodc2:~$ dmesg | tail -10
[ 3252.783684] usb 1-1.1: new full-speed USB device number 5 using dw2
[ 3252.885849] usb 1-1.1: New USB device found, idVendor=1645, idProduct=es08, bcdDevice= 1.00
[ 3252.885860] usb 1-1.1: new USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 3252.885877] usb 1-1.1: Product: CP2102 USB to UART Bridge Controller
[ 3252.885880] usb 1-1.1: Manufacturer: Silicon Labs
[ 3252.885923] usb 1-1.1: SerialNumber: 0001
[ 3252.913064] usbcore: registered new interface driver cp210x
[ 3252.913065] usbserial: USB Serial support registered for cp210x
[ 3252.913777] cp210x 1-1.1.1:0: cp210x converter detected
[ 3252.915003] usb 1-1.1:1.0: cp210x converter now attached to ttyUSB0
```

At the bottom, you'll see the 'name' of the attached device, in this case its `ttyUSB0`, that means our serial port device is available at `/dev/ttyUSB0`
The Hard Way - Using Built-in UART

As of this writing, there is an issue in the current armbian distribution with /dev/ttyAML1 not exchanging data properly. However, if using SSH, you can use the UART Serial port and /dev/ttyAML0 as a workaround.

If you don't want to plug in external hardware to the Odroid you can use the built in UART on the RX/TX pins.

You can use the serial console UART via /dev/ttyAML0

Wire the GPS as follows:

Install the CircuitPython GPS Library

OK onto the good stuff, you can now install the Adafruit GPS CircuitPython library.

As of this writing, not all libraries are up on PyPI so you may want to search before trying to install. Look for circuitpython and then the driver you want.
(If you don’t see it you can open up a github issue on circuitpython to remind us (!))

Once you know the name, install it with

```bash
sudo pip3 install pyserial adafruit-circuitpython-gps
```

You’ll notice we also installed a dependency called pyserial. This is a great thing about pip, if you have other required libraries they’ll get installed too!

We also recommend an adafruit-blinka update in case we’ve fixed bugs:

```bash
sudo pip3 install --upgrade adafruit_blinka
```

**Run that code!**

The finish line is right up ahead. You can now run one of the (many in some cases) example scripts we’ve written for you.

Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be [https://github.com/adafruit/Adafruit_CircuitPython_GPS/tree/master/examples](https://github.com/adafruit/Adafruit_CircuitPython_GPS/tree/master/examples)

Let’s start with the simplest, the echo example

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

# Simple GPS module demonstration.
# Will print NMEA sentences received from the GPS, great for testing connection
# Uses the GPS to send some commands, then reads directly from the GPS
import time
import board
import busio

import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
```
```python
# import serial
uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
i2c = board.I2C()  # uses board.SCL and board.SDA
i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart)  # Use UART/pyserial
gps = adafruit_gps.GPS_GtopI2C(i2c)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

gps.send_command(b"PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0"
# Turn on the basic GGA and RMC info (what you typically want)
gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Tuen on everything (not all of it is parsed!)
gps.send_command(b'PMTK314,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b"PMTK220,1000")
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
gps.send_command(b'PMTK220,2000')
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing. This would be twice a second (2hz, 500ms delay):
gps.send_command(b'PMTK220,500')

# Main loop runs forever printing data as it comes in
timestamp = time.monotonic()
while True:
    data = gps.read(32)  # read up to 32 bytes
    print(data)  # this is a bytearray type
    if data is not None:
        data_string = ".join([chr(b) for b in data])
        print(data_string, end="")
        if time.monotonic() - timestamp > 5:
            # every 5 seconds...
            # request firmware version
            gps.send_command(b"PMTK605")
            timestamp = time.monotonic()
```

We'll need to configure this code to work with our UART port name.

- If you're using a USB-to-serial converter, the device name is probably `/dev/ttyUSB0` - but check `dmesg` to make sure.
- If you're using the built-in UART on the Odroid, the device name is `/dev/ttyAML0`

Comment out the lines that reference `board.TX`, `board.RX` and `busio.uart` and uncomment the lines that import `serial` and define the `serial` device, like so:
# Define RX and TX pins for the board's serial port connected to the GPS. 
# These are the defaults you should use for the GPS FeatherWing. 
# For other boards set RX = GPS module TX, and TX = GPS module RX pins. 
# RX = board.RX 
# TX = board.TX 

# Create a serial connection for the GPS connection using default speed and 
# a slightly higher timeout (GPS modules typically update once a second). 
# uart = busio.UART(TX, RX, baudrate=9600, timeout=3000) 

# for a computer, use the pyserial library for uart access 
import serial 
uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=3000) 

And update the "/dev/ttyUSB0" device name if necessary to match your USB interface 

Whichever method you use, you should see output like this, with $GP "NMEA sentences" - there probably wont be actual location data because you haven't gotten a GPS fix. As long as you see those $GP strings sorta like the below, you've got it working!

```
pi@odroidc2:~$ sudo python3 gps_simpletest.py
$PMTK001,314,3*36
$PMTK001,220,3*30
$GPSA,182130.900,....,0.00,.,M,M,.*78
$GPSCM,182130.900,V,,,0.00,0.00,130619,,,N*41
$GPSCG,182131.900,....,0.00,.,M,M,.*79
$GPSCM,182131.900,V,,,0.00,0.00,130619,,,N*40
$GPSCG,182132.900,....,0.00,.,M,M,.*7A
$GPSCM,182132.900,V,,,0.00,0.00,130619,,,N*43
$GPSCG,182133.900,....,0.00,.,M,M,.*7B
$GPSCM,182133.900,V,,,0.00,0.00,130619,,,N*42
$GPSCG,182134.900,....,0.00,.,M,M,.*7C
```

---

More To Come!

That's just a taste of what we've got working so far

We're adding more support constantly, so please hold tight and visit the adafruit_blinka github repo to share your feedback and perhaps even submit some improvements!

If you'd like to contribute, but aren't sure where to start, check out the following guides:

- Adding a Single Board Computer to PlatformDetect for Blinka
- Adding a Single Board Computer to Blinka

©Adafruit Industries  Page 37 of 45
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 (!) 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 (!)**, please use in I2C mode instead of SPI
• **MAX31865** is phase 1, 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`. You need at least version 3.4. This should have been taken care of 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:

```bash
set BLINKA_MCP2221_RESET_DELAY=0.5
```

Linux:

```bash
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:


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

which describes copying the 32bit and 64bit DLLs into specific folders. (example for Win7)
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 will give you AnalogIn objects. For PWM outputs, try the PCA9685. For audio, use pygame or other Python3 libraries to play audio.

Some libraries, like Adafruit_CircuitPython_DHT 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, 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.