CircuitPython on Linux and Raspberry Pi
Created by lady ada

Last updated on 2019-06-16 07:08:21 PM UTC
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. Letting you program in Python and talk to Circuitry like sensors, motors, and LEDs!

CircuitPython on Microcontrollers

For a couple years now we've had CircuitPython for microcontrollers like our SAMD21 series with Feather/Trinket/CircuitPlayground/Metro M0, as well as the ESP8266 WiFi microcontroller, nRF52 bluetooth microcontroller and SAMD51 series.

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...

Please note! All the stuff in this guide works and we're improving and working on this code a bunch so be sure to check back for updates!
CircuitPython Libraries on Linux & Raspberry Pi

The next obvious step is to bring CircuitPython 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 it's pretty easy to get that code working with micro-computers like Raspberry Pi or other 'Linux with GPIO pins available' single board computers.

We are not running the CircuitPython interpreter itself on the Linux machine. But we are running Python code written to use the CircuitPython hardware API (busio.I2C, busio.SPI, etc.)

We'll use a special library called adafruit_blinka (https://adafru.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafru.it/BJT)) to provide the 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 (https://adafru.it/BJU) 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 any code we have for CircuitPython will be instantly and easily runnable on Linux computers like Raspberry Pi.

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 - GPIO Zero?

Yes! We like and use GPIO Zero a lot(https://adafru.it/BJV), its an excellent hardware interfacing library for Raspberry
Pi. It's great for digital in/out, analog inputs, servos, some basic sensors, etc. In particular, one cool thing it does is thread/event management so you can have code run, say, when a button is pressed.

GPIO Zero excels at that, but doesn't cover SPI/I2C sensors or drivers, which is where we got stuck: for each sensor we had we'd write a driver in C/C++ for Arduino, CircuitPython using our hardware API, and then Python using smbus or similar.

By letting you use CircuitPython on Raspberry Pi via adafruit_blinka, you can unlock all of the drivers and example code we wrote! And you can keep using GPIO Zero for pins, buttons and LEDs. 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?

Plus, we're adapting and extending adafruit_blinka to support other boards (https://adafruit.it/DbB) such as Allwinners and BeagleBone, even some smaller linux boards like Onion.io will be able to run CircuitPython code.

If you have a board you'd like to adapt check out the adafruit_blinka code on github (https://adafruit.it/BJX), pull requests are welcome as there's a ton of different Linux boards out there! You'll need to add a detection element (https://adafruit.it/Dyb) so we can tell what board you're running on, then the pin definitions into adafruit_blinka above. As long as you're running a modern kernel, you'll have libgpiod for GPIO, smbus for I2C and spidev for SPI all ready to go.
Prerequisite Pi Setup!

In this page we’ll assume you’ve already gotten your Raspberry Pi up and running and can log into the command line.

Here’s the quick-start for people with some experience:

1. Download the latest Raspbian or Raspbian Lite (https://adafru.it/fQi) to your computer
2. Burn the Raspbian image to your MicroSD card (https://adafru.it/dDL) using your computer
3. Re-plug the SD card into your computer (don’t use your Pi yet!) and set up your wifi connection by editing supplicant.conf (https://adafru.it/yuD)
4. Activate SSH support (https://adafru.it/yuD)
5. Plug the SD card into the Pi
6. If you have an HDMI monitor we recommend connecting it so you can see that the Pi is booting OK
7. Plug in power to the Pi - you will see the green LED flicker a little. The Pi will reboot while it sets up so wait a good 10 minutes
8. If you are running Windows on your computer, install Bonjour support so you can use .local names, you’ll need to reboot Windows after installation (https://adafru.it/lPE)
9. You can then ssh into raspberrypi.local (https://adafru.it/jvB)

The Pi Foundation has tons of guides as well (https://adafru.it/BJY)

Update Your Pi and Python

Run the standard updates:

`sudo apt-get update`

`sudo apt-get upgrade`

and

`sudo pip3 install --upgrade setuptools`

If above doesn't work try

`sudo apt-get install python3-pip`

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

You only have to do this *once* per Raspberry Pi but by default both interfaces are disabled!

- Enable I2C (https://adafruit.it/dEO)
- Enable SPI (https://adafruit.it/BJ2)

Once you're done with both and have rebooted, verify you have the I2C and SPI devices with the command:

```
ls /dev/i2c* /dev/spi*
```

You should see the response:

```
/dev/i2c-1 /dev/spidev0.0 /dev/spidev0.1
```

Enabling Second SPI

If you are using the main SPI port for a display or something and need another hardware SPI port, you can enable it by adding the line:

```
dtoverlay=spi1-3cs
```

to the bottom of `/boot/config.txt` and rebooting. You'll then see the addition of some /dev/spidev1.x devices:

```
pi@devpi:~ $ ls /dev/spi*
/dev/spidev0.0 /dev/spidev0.1 /dev/spidev1.0 /dev/spidev1.1 /dev/spidev1.2
```

Make sure you’re using Python 3!

The default python on your computer may not be python 3. Python 2 is officially discontinued and all our libraries are Python 3 only.

We'll be using `python3` and `pip3` in our commands, use those versions of python and pip to make sure you're using 3 and not 2

Install Python libraries

Now you're ready to install all the python support

Run the following command to install the Raspberry PI GPIO library:

```
pip3 install RPI.GPIO
```
Run the following command to install `adafruit_blinka`

```bash
top@devpi ~$ pip3 install adafruit-blinka
```

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

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

**Blinka Test**

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

```python
import board
import digitalio
import busio

print("Hello blinka!")

# Try to great a Digital input
pin = digitalio.DigitalInOut(board.D4)
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

```bash
python3 blinkatest.py
```

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

![Command Line Output](image-url)
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 Raspberry Pi. We'll extend the example to also show how to wire up a button/switch and enable a pull-up resistor.

Even if you use a different library to create digital in/outs like GPIO Zero, there's a number of sensor libraries that use a digital pin for resetting, or for a chip-select. So it's good to have this part working!

Parts Used

Any old LED will work just fine as long as its not an IR LED (you can't see those) and a 470 to 2.2K resistor

![Diffused Blue 10mm LED (25 pack)](image)

$9.95
IN STOCK
ADD TO CART

![Through-Hole Resistors - 470 ohm 5% 1/4W - Pack of 25](image)

$0.75
IN STOCK
ADD TO CART

Some tactile buttons or switches
We recommend using a breadboard and some female-male wires.

You can use a Cobbler to make this a little easier, the pins are then labeled!

© Adafruit Industries  https://learn.adafruit.com/circuitpython-on-raspberrypi-linux  Page 12 of 45
Wiring

Connect the Raspberry Pi **Ground** pin to the **blue ground rail** on the breadboard.

- Connect one side of the tactile switch to Raspberry Pi GPIO **#4**
- Connect the other side of the tactile switch to the **ground** rail
- Connect the longer/positive pin of the LED to Raspberry Pi GPIO **#18**
- 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 Pi 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!
Blinky Time!

The finish line is right up ahead, let's 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.D18)
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 #18, the resistor is installed correctly, and you have a Ground wire to the Raspberry Pi.

Type Control-C to quit

Button It Up

Now that you have the LED working, let's 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.D18)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.D4)
button.direction = digitalio.Direction.INPUT
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 (https://adafruit.it/BK0))

Let's 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 (https://adafruit.it/BK0) and which pin is SDA and which pin is SCL!

Don't forget you have to enable I2C with raspi-config!

Parts Used

Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor

OUT OF STOCK

Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

$3.95
IN STOCK

ADD TO CART
You can use a Cobbler to make this a little easier, the pins are then labeled!

Wiring

- Connect the Raspberry Pi 3.3V power pin to Vin
- Connect the Raspberry Pi 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 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.
Once you know the name, install it with

```
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:

```
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 one example. But that's cool, here it is:
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("Temperature: %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 Pi by copying and pasting it into a text file, downloading it directly from the Pi, etc.

Then in your command line run

python3 bme280_simpletest.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

I2C Clock Stretching

In order to use certain I2C sensors, such as the BNO055 (https://adafru.it/fE0) and the CCS811 (https://adafru.it/BlK), you'll need to enable I2C clock stretching 'support' by greatly slowing down the I2C clock on the Raspberry Pi using the device tree overlay.

This is done by adding a line in `/boot/config.txt`. Log in to a terminal on your Pi and open that file in Nano, or your text editor of choice:

```bash
sudo nano /boot/config.txt
```

Scroll down until you find a block like:

```bash
# Uncomment some of all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
dtparam=i2s=on
dtparam=spi=on
```

This block might vary depending on what you've enabled in `raspi-config`. Directly below it, add the following:

```bash
# Clock stretching by slowing down to 10KHz
dtparam=i2c_arm_baudrate=10000
```

The default baudrate may be 100KHz or 1MHz, by slowing it down to 10KHz or more, you may be able to be slow enough to avoid missing clocks.

In Nano, your screen should look like this:

```bash
# Enable audio (loads snd_bcm2835)
#dtparam=audio=on
dtparam=audio=on
dtparam=i2c_arm=on
```

Next, save the file and exit (in Nano, press Ctrl-X, y for yes, and Enter).

Now you can reboot your Pi and proceed to testing your I2C device:
If you still get bad data, try slowing it down more, maybe to 5 KHz or 1 KHz rate. Reboot after each change.

```bash
sudo reboot
```
SPI Sensors & Devices

SPI is less popular than I2C but still you’ll see lots of sensors and chips use it. Unlike I2C, you don’t have everything share two wires. Instead, there’s three shared wires (clock, data in, data out) and then a unique ‘chip select’ line for each chip.

The nice thing about SPI is you can have as many chips as you like, even the same kind, all share the three SPI wires, as long as each one has a unique chip select pin.

The formal/technical names for the 4 pins used are:

- **SPI clock** - called **SCLK, SCK or CLK**
- **SPI data out** - called **MOSI** for **Master Out Slave In**. This is the wire that takes data from the Linux computer to the sensor/chip. Sometimes marked **SDI or DI** on chips
- **SPI data in** - called **MISO** for **Master In Slave Out**. This is the wire that takes data to the Linux computer from the sensor/chip. Sometimes marked **SDO or DO** on chips
- **SPI chip select** - called **CS or CE**

Remember, connect all SCK, MOSI and MISO pins together (unless there’s some specific reason/instruction not to) and a unique CS pin for each device.

**WARNING! SPI on Linux/Raspberry PI**

SPI on microcontrollers is fairly simple, you have an SPI peripheral and you can transfer data on it with some low level command. Its ‘your job’ as a programmer to control the CS lines with a GPIO. That’s how CircuitPython is structured as well. **busio** does just the SPI transmit/receive part and **busdevice** handles the chip select pin as well.

Linux, on the other hand, doesn’t let you send data to SPI without a CS line, and the CS lines are fixed in hardware as well. For example on the Raspberry Pi, there’s only two CS pins available for the hardware SPI pins - **CE0** and **CE1** - and you have to use them. (In theory there’s an ioctl option called **no_cs** but this does not actually work)

The upshot here is - to let you use more than 2 peripherals on SPI, we decided to let you use any CS pins you like, CircuitPython will toggle it the way you expect. But when we transfer SPI data we always tell the kernel to use **CE0**. CE0 will toggle like a CS pin, but if we leave it disconnected, its no big deal.

**The upshot here is basically never connect anything to CE0 (or CE1 for that matter). Use whatever chip select pin you define in CircuitPython and just leave the CE pins alone, it will toggle as if it is the chip select line, completely on its own, so you shouldn’t try to use it as a digital input/output/whatever.**

**Don’t forget you have to enable SPI with raspi-config!**

Using the Second SPI Port

The Raspberry Pi has a 'main' SPI port, but not a lot of people know there's a second one too! This is handy if you are using the main SPI port for a PiTFT or other kernel-driven device. You can enable this SPI #1 by adding

```
dtoverlay=spi1-3cs
```
to the bottom of `/boot/config.txt` and rebooting. You'll then see the addition of some `/dev/spidev1.x` devices.

Here's the wiring for SPI #1:

- **SCK_1** on GPIO #21
- **MOSI_1** on GPIO #20
- **MISO_1** on GPIO #19
- **SPI #1 CS0** on GPIO 18
- **SPI #1 CS1** on GPIO 17
- **SPI #1 CS2** on GPIO 16

like the main SPI, we'll use `CE0` as our default but don't connect to it! Use any other pin and leave that one unused. Then update your scripts to use

```python
spi = busio.SPI(board.SCK_1, MOSI=board.MOSI_1, MISO=board.MISO_1)
```

### Parts Used

OK now that we've gone thru the warning, let's wire up an SPI MAX31855 thermocouple sensor, this particular device doesn't have a MOSI pin so we'll not connect it.

![Thermocouple Amplifier MAX31855 breakout board (MAX6675 upgrade)](image)

- **Thermocouple Amplifier MAX31855 breakout board (MAX6675 upgrade)**
  - $14.95
  - IN STOCK
  - ADD TO CART

![Thermocouple Type-K Glass Braid Insulated](image)

- **Thermocouple Type-K Glass Braid Insulated**
  - $9.95
  - IN STOCK
  - ADD TO CART

We recommend using a breadboard and some female-male wires.
You can use a Cobbler to make this a little easier, the pins are then labeled!

Wiring

- Connect the Raspberry Pi 3.3V power pin to Vin
- Connect the Raspberry Pi GND pin to GND
- Connect the Pi SCLK pin to the MAX31855 CLK
- Connect the Pi MISO pin to the MAX31855 DO
- Connect the Pi GPIO 5 pin to the MAX31855 CS
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!

Install the CircuitPython MAX31855 Library

OK onto the good stuff, you can now install the Adafruit MAX31855 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

pip3 install adafruit-circuitpython-max31855
You'll notice we also installed a few other dependencies called `spidev`, `adafruit-pureio`, `adafruit-circuitpython-busdevice` and more. 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:

```
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_MAX31855/tree/master/examples](https://github.com/adafruit/Adafruit_CircuitPython_MAX31855/tree/master/examples)

As of this writing there's only one example. But that's cool, here it is:
import time
import board
import busio
import digitalio
import adafruit_max31855

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
cs = digitalio.DigitalInOut(board.D5)

max31855 = adafruit_max31855.MAX31855(spi, cs)

while True:
    tempC = max31855.temperature
    tempF = tempC * 9 / 5 + 32
    print('Temperature: {} C {} F '.format(tempC, tempF))
    time.sleep(2.0)

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

Then in your command line run

```python
python3 max31855_simpletest.py
```

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

Make sure you have a K-type thermocouple installed into the sensor breakout or you will get an error like the one below!

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

https://learn.adafruit.com/circuitpython-on-raspberrypi-linux
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 Raspberry Pi. The easy way, and the hard way.

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

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
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.

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

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:

```
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

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

But, if you do this, you'll lose the serial console, so if you're using a PiUART or console cable or HAT that lets you connect directly to the console, that will no longer work and you'll have to use the HDMI+Keyboard or ssh method of running commands!

This isn't a big deal, in fact the serial login-console isn't even enabled by default on Raspbian anymore, but it's worth a warning!

Disabling Console & Enabling Serial

Before wiring up, make sure you have disabled the console.

Run `sudo raspi-config` and select the following:
Select No on enabling the login shell

Select Yes on enabling serial port hardware

Once complete you should have no console and yes on serial interface:

Then **reboot**

Once you've rebooted, you can use the built in UART via `/dev/ttyS0`

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[https://adafru.it/tB7]!)

Once you know the name, install it with

`pip3 install adafruit-circuitpython-gps`

You’ll notice we also installed a dependancy 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:

`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://adafruit.it/Ca9)

Let's start with the simplest, the echo example

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

# 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=30)

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

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart)

# 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
# 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 on just minimum info (RMC only, location):
#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')
# Turn on everything (not all of it is parsed!)
gps.send_command(b'PMTK314,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# 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 could be 1001 which is twice default.
```
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 a Pi, the device name is `/dev/ttyS0` - note that last character is a zero

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:

```python
# 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 won't 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!
root@orangecirc:/home/pi# python3 opptest.py
SPNTK001.3=36
SPNTK001.3=T30
SPNTK001.3=T30
SGPGGA,000013.800,....,0.00,..W,M,.,72
SGPRMC,000013.800,V,....,0.00,0.00,0600180,,N48
SGPGGA,000014.799,....,0.00,..W,M,.,77
SGPRMC,000014.799,V,....,0.00,0.00,0600180,,N49
SGPGGA,000015.799,....,0.00,..W,M,.,78
SGPRMC,000015.799,V,....,0.00,0.00,0600180,,N50
SGPGGA,000016.799,....,0.00,..W,M,.,78
SGPRMC,000016.799,V,....,0.00,0.00,0600180,,N51
SGPGGA,000017.799,....,0.00,..W,M,.,79
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 (https://adafruit.it/BJX) to share your feedback and perhaps even submit some improvements!
CircuitPython & OrangePi

CircuitPython & OrangePi (https://adafruit.it/DbB)
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!

- 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.
No Pullup/Pulldown support on some linux boards

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!
Not all peripherals supported

Some CircuitPython modules like neopixel, analogio, audioio and pulseio may not be supported. 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!
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
- `pulseio.pulseIn` using gpiod
- 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`
- `pulseio.pulseIn` using gpiod
- 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)