CircuitPython Libraries on Linux and the NVIDIA Jetson Nano

Created by Melissa LeBlanc-Williams

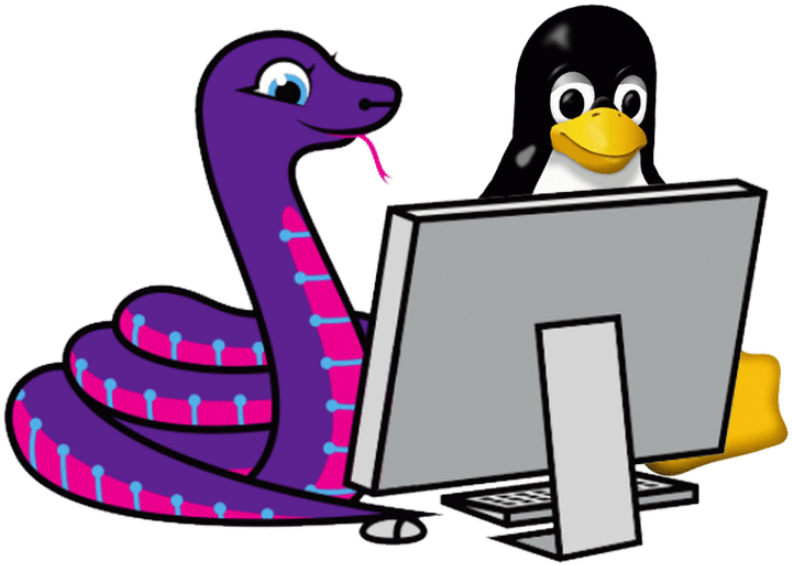
https://learn.adafruit.com/circuitpython-libraries-on-linux-and-the-nvidia-jetson-nano

Last updated on 2022-04-20 11:29:43 PM EDT

Table of Contents

Overview

This guide describes using CircuitPython -libraries- on small Linux computers, running under regular Python. It is not about running the CircuitPython firmware itself on those boards.



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 wi th "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.

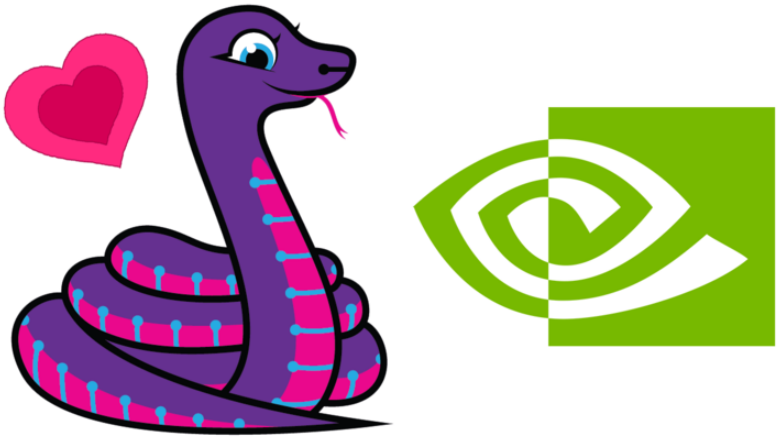
You'll use a special library called adafruit_blinka (https://adafru.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafru.it/BJT)) that provides a layer that translates the CircuitPython hardware API to whatever library the Linux board provides. For example, on Raspberry Pi we use the python RPi.GPIO (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 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.

CircuitPython & Jetson Nano



CircuitPython Libraries on Linux & NVIDIA Jetson Nano

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 microcomputers like the Jetson Nano or other 'Linux with GPIO pins available' single board computers.

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 the Jetson Nano 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 the Jetson Nano.

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 - Jetson.GPIO?

Jetson.GPIO is a pure python hardware interface class (https://adafru.it/FMQ) for the Jetson Nano. It works just fine for I2C, SPI and GPIO but doesn't work with our drivers as it's a different API

By letting you use CircuitPython libraries on the Jetson Nano via adafruit_blinka, you can unlock all of the drivers and example code we wrote! And you can keep using Jet son.GPIO 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 ones we've got, so we could test them thoroughly. If you have other SBC board you'd like to adapt check out the adafruit_blinka code on github (https://adafru.it/BJX), pull requests are welcome as there's a ton of different Linux boards out there!

Initial Setup

Be sure to use the very latest Jetson Nano Dev Kit image because earlier versions required you to re-flash the board to enable SPI.

Install Jetson Nano Developer Kit on your Jetson Nano

We decided to try getting Blinka running in the Jetson Nano Developer Kit because that's the recommended installation available for the Jetson Nano. Other distros could be made to work but you'd probably need to figure out how to detect the platform. Using other operating systems and CircuitPython is your call, we cannot provide support for that.

Due to the size of the Dev Kit Image, you will need at least a minimum of a 16GB SD card.

Download and install the latest Jetson Nano Developer Kit, for example we're using https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write (https://adafru.it/FMD)

There's some documentation to get started at https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit (https://adafru.it/FMD)

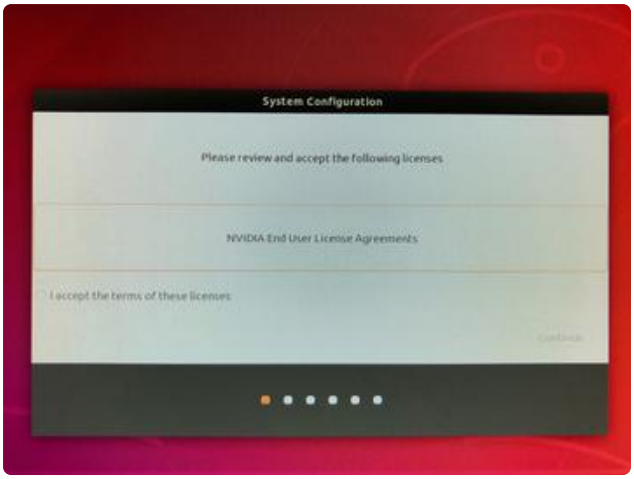
Blinka only supports the Jetson Nano Developer Kit on the Jetson Nano because that's the recommended OS we could find and it's easy to detect which board you have

Preparing the Board

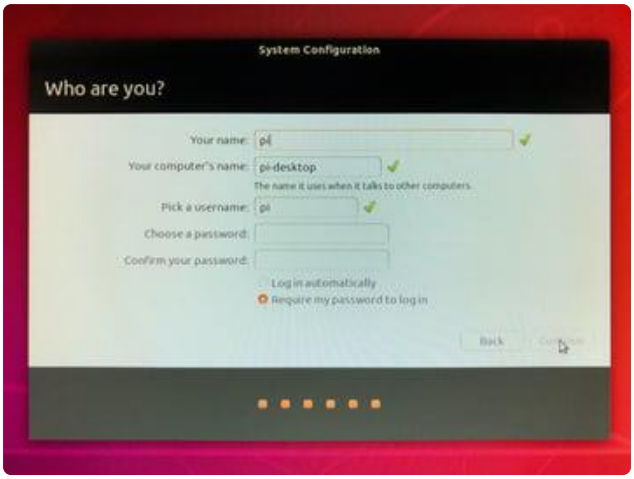
A monitor and keyboard are required to set this board up. A mouse is helpful too.

Wireless Keyboard and Mouse Combo - One USB Port!
Add a good quality, slim chiclet keyboard as well as an optical mouse to your Raspberry Pi, Beagle Bone Black or other mini-computer with this wireless combo set. Best of all,...
https://www.adafruit.com/product/1738

After you burn the image to the SD card, you'll need to install it into the Nano. The card slot is located on the underside of the large heatsink.



Connect a monitor (HDMI or DisplayPort), USB mouse, and keyboard to continue. An End User License Agreement will need to be accepted before you can continue.



As you go through the configuration wizard, take note of the fields your name and your computer's name. These will be your username and hostname for connecting by SSH.

Logging in

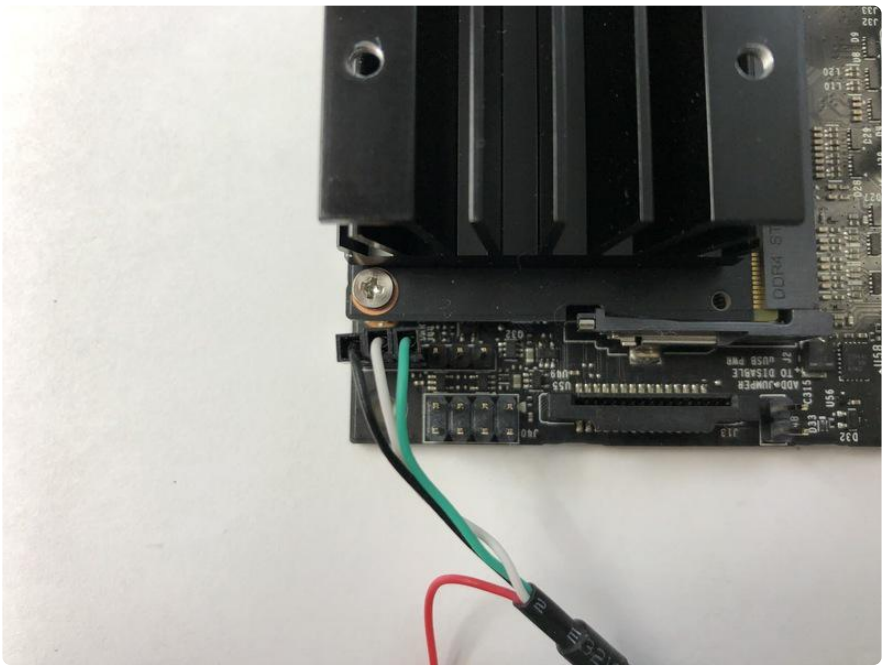
Once you have completed the wizard, you can either continue using that, connect a serial console cable, or log in through SSH. We've found the easiest way to connect is through a console cable, wired to the J44 connector and then on your computer, use a serial monitor at 115200 baud.

You can also connect via SSH using either the command prompt on a Mac or Linux computer or using a terminal program such as PuTTY on Windows. In either case, you will need your username and hostname from earlier in the setup wizard.

You may need to reboot the Jetson Nano after completing the wizard before the device will show up on your network.

[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

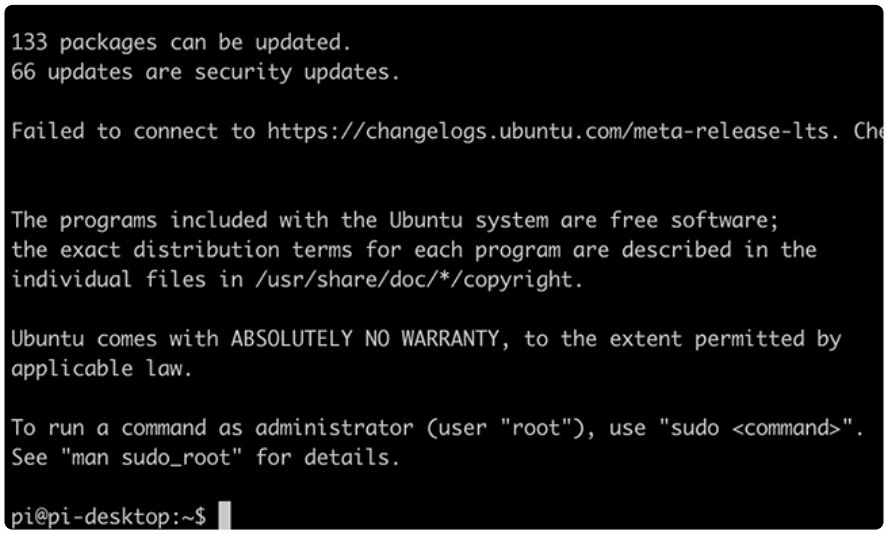
Connect the RX, TX and ground wires of the console cable like so:



```
Starting kernel ...

<hit enter to activate fiq debugger>
[    1.083572] tegradc tegradc.1: dpd enable lookup fail:-19
[    1.259482] imx219 6-0010: imx219_board_setup: error during i2c read probe (-1
[    1.259512] imx219 6-0010: board setup failed
[    2.425879] cgroup: cgroup2: unknown option "nsdelegate"
[    3.498786] using random self ethernet address
[    3.503744] using random host ethernet address
[    3.543160] using random self ethernet address
[    3.548250] using random host ethernet address
[    3.855343] random: crng init done
[    3.858781] random: 7 urandom warning(s) missed due to ratelimiting
[    4.122734] Please complete system configuration setup on desktop to proceed..

Ubuntu 18.04.2 LTS pi-desktop ttyS0

pi-desktop login: [  369.127001] using random self ethernet address
[  369.131530] using random host ethernet address
[  369.166407] using random self ethernet address
[  369.171485] using random host ethernet address

Ubuntu 18.04.2 LTS pi-desktop ttyS0

pi-desktop login:
```

Once powered correctly and with the right SD card you should get a command prompt as the username you selected in the configuration wizard. You may need to press enter if it appears to stop.

Log in to get to a shell. In this guide, we will be using the user `pi` just to keep things consistent with other guides. Change the commands to use whatever username you ended up using.



Set your Python install to Python 3 Default

There's a few ways to do this, we recommend something like this:

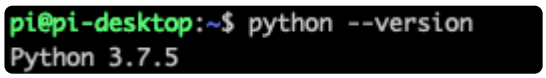
- `sudo apt install -y python3 git python3-pip`
- `sudo update-alternatives --install /usr/bin/python python $(which python2) 1`
- `sudo update-alternatives --install /usr/bin/python python $(which python3) 2`
- `sudo update-alternatives --config python`

Install Python 3.7 and Make Default

You can install Python 3.7, but it won't be the default Python. However, you can do something like above:

- `sudo apt install -y python3.7-dev`
- `sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.6 1`
- `sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.7 2`
- `sudo update-alternatives --config python`

Now run `python --version` and it should show some version of Python 3.7:



Update Your Board and Python

Run the standard updates:

```
sudo apt update
```

```
sudo apt upgrade
```

and

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

Update all your Python 3 packages with

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

and

```
sudo bash
```

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

Nano (no relationship to the board), which is very handy for quickly editing python scripts, isn't installed by default. So let's add that:

```
sudo apt install nano
```

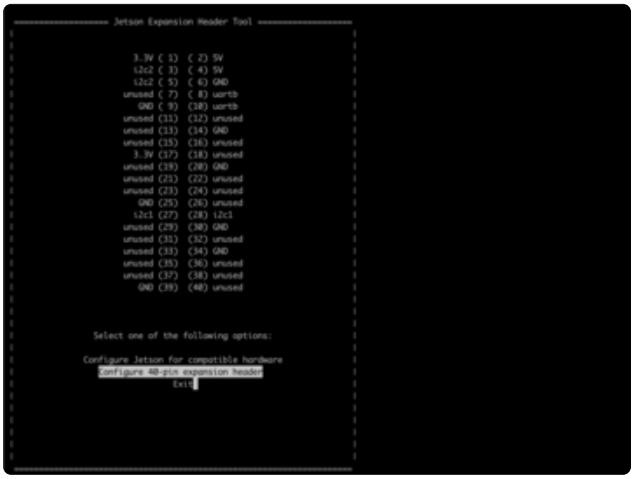
Enable UART, I2C and SPI

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

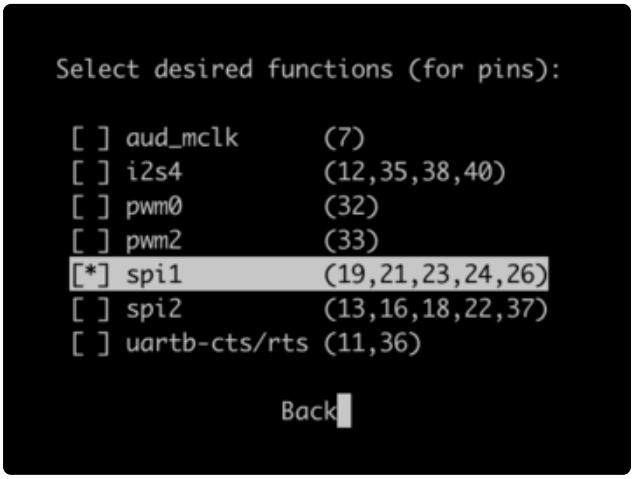
I2C and UART are enabled by default, so you don't need to take any additional steps.

To enable SPI, you will need to run the Jetson-IO utility. Be sure your terminal window is expanded first, or it may not display the full utility:

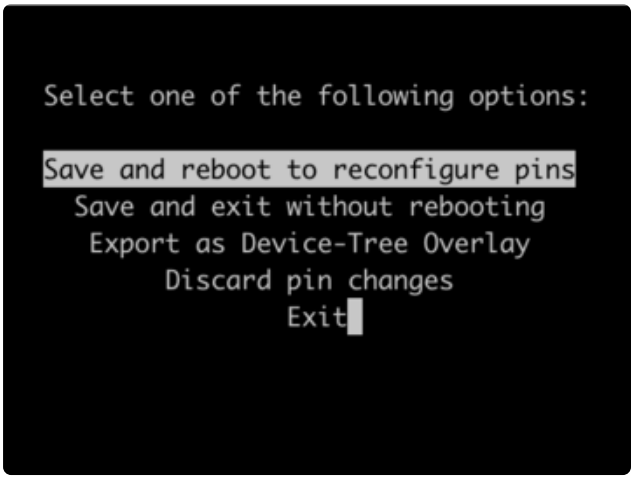
```
sudo /opt/nvidia/jetson-io/jetson-io.py
```



Select Configure 40-pin expansion header at the bottom. Then select Configure header pins manually.



Select spi1 (19, 21, 23, 24, 26) and then select Back



Finally select Save pin changes and then Save and reboot to reconfigure pins. This will create a config file and reboot the Jetson Nano.

After the Nano boots up again, verify you have the I2C devices with the command `ls /dev/i2c* /dev/spi*`

You should see at least one I2C device and one SPI device

```
pi@pi-desktop:~$ ls /dev/i2c* /dev/spi*
/dev/i2c-0  /dev/i2c-3  /dev/i2c-6  /dev/spidev0.0  /dev/spidev1.1
/dev/i2c-1  /dev/i2c-4  /dev/i2c-7  /dev/spidev0.1
/dev/i2c-2  /dev/i2c-5  /dev/i2c-8  /dev/spidev1.0
pi@pi-desktop:~$
```

```
pi@pi-desktop:~$ sudo i2cdetect -l
i2c-3   i2c             7000c700.i2c                            I2C adapter
i2c-1   i2c             7000c400.i2c                            I2C adapter
i2c-6   i2c             Tegra I2C adapter                       I2C adapter
i2c-4   i2c             7000d000.i2c                            I2C adapter
i2c-2   i2c             7000c500.i2c                            I2C adapter
i2c-0   i2c             7000c000.i2c                            I2C adapter
i2c-5   i2c             7000d100.i2c                            I2C adapter
pi@pi-desktop:~$
```

You can test to see what I2C addresses are connected by running `sudo i2cdetect -r -y 0` (on pins 27/28) or `sudo i2cdetect -r -y 1` (on pins 3/5)

In this case I do have a sensor on the 'standard' i2c port i2c-1 under address 0x77

The UART Serial Console on the Jetson Nano is connected to /dev/ttyS0. The UART GPIO Serial Port is connected to /dev/ttyTHS1.

Set User Permissions

In order to use the Jetson GPIO Library, the correct user permissions/groups must be set first. Start by creating a new gpio user group.

```
sudo groupadd -f -r gpio
sudo usermod -a -G gpio pi
```

Clone the Repo and copy the rules

```
cd ~
git clone https://github.com/NVIDIA/jetson-gpio.git
sudo cp ~/jetson-gpio/lib/python/Jetson/GPIO/99-gpio.rules /etc/udev/rules.d
```

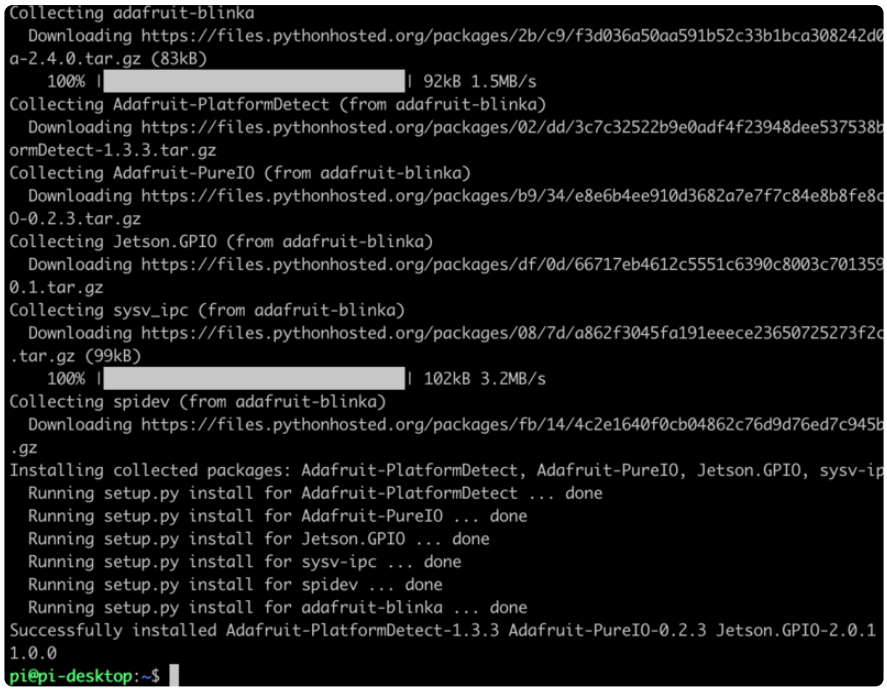
Now reboot so that the new rules take place.

Install Python Libraries

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

Next, run the following command to install adafruit_blinka:

```
pip3 install adafruit-blinka
```



The computer will install a few different libraries such as `Adafruit-PureIO` (our ioctl-only i2c library), `Jetson.GPIO` (for handling GPIO), `Adafruit-PlatformDetect` (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.D4)
print("Digital IO ok!")

# Try to create an I2C device
i2c = busio.I2C(board.SCL, board.SDA)
print("I2C 1 ok!")
i2c = busio.I2C(board.SCL_1, board.SDA_1)
print("I2C 2 ok!")

print("done!")
```

Save it and run at the command line with

`python3 blinkatest.py`

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

```
pi@pi-desktop:~$ sudo python3 blinkatest.py
Hello blinka!
Digital IO ok!
I2C 1 ok!
I2C 2 ok!
done!
Exiting...
Cleaning up pins
pi@pi-desktop:~$
```

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 Jetson Nano. We'll extend the example to also show how to wire up a button/switch.

Jetson Nano boards don't have any way to set the pullup/pulldown resistors, so you'll need to use external resistors instead of built-in pullups, whenever it makes sense!

Jetson Nano Dev-Board Expansion Header

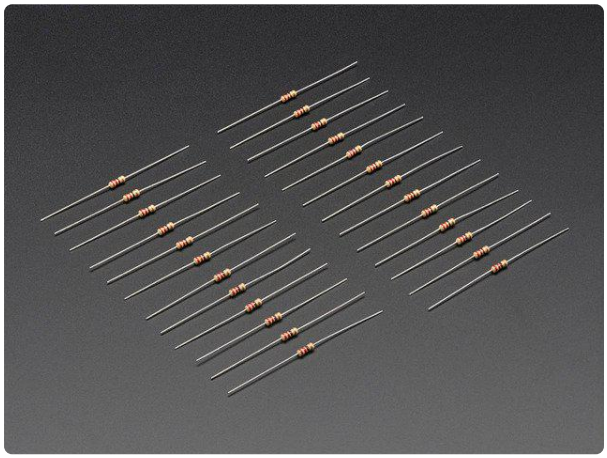
Alt Function	Linux(BCM)	Board Label			Board Label	Linux(BCM)	Alt Function
DAP4_DOUT	78(21)	D21	40	39	GND		
DAP4_DIN	77(20)	D20	38	37	D26	12(26)	SPI2_MOSI
UART2_CTS	51(16)	D16	36	35	D19	76(19)	DAP4_FS
		GND	34	33	D13	38(13)	GPIO_PE6
LCD_BL_PWM	168(12)	D12	32	31	D6	200(6)	GPIO_PZ0
		GND	30	29	D5	149(5)	CAM_AF_EN
		D1/ID_SC	28	27	D0/ID_SD		
SPI1_CS1	20(7)	D7	26	25	GND		
SPI1_CS0	19(8)	D8	24	23	D11	18(11)	SPI1_SCK
SPI2_MISO	13(25)	D25	22	21	D9	17(9)	SPI1_MISO
		GND	20	19	D10	16(10)	SPI1_MOSI
SPI2_CS0	15(24)	D24	18	17	3.3V		
SPI2_CS1	232(23)	D23	16	15	D22	194(22)	LCD_TE
		GND	14	13	D27	14(27)	SPI2_SCK
DAP4_SCLK	79(18)	D18	12	11	D17	50(17)	UART2_RTS
		RXD/D15	10	9	GND		
		TXD/D14	8	7	D4	216(4)	AUDIO_MCLK
		GND	6	5	SCL/D3		
		5V	4	3	SDA/D2		
		5V	2	1	3.3V		

Note that this graphic matches the 2x20 Raspberry Pi pinout but it's upside-down

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 220 to 2.2K resistor

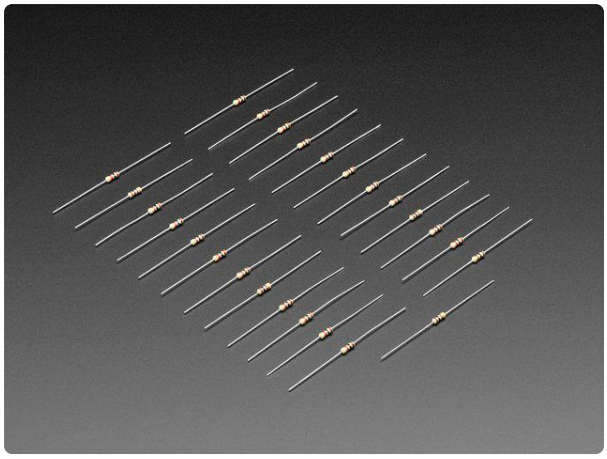
Diffused Blue 10mm LED (25 pack)
Need some big indicators? We are big fans of these huge diffused blue LEDs. They are really bright so they can be seen in daytime, and from any angle. They go easily into a breadboard...
https://www.adafruit.com/product/847



Through-Hole Resistors - 220 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/2780

Some tactile buttons or switches along with some pull-up resistors:

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

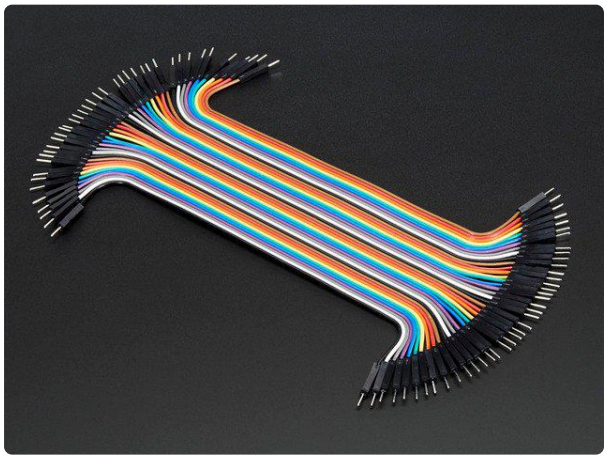


Through-Hole Resistors - 1.0K ohm 5% 1/4W - Pack of 25

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/4294

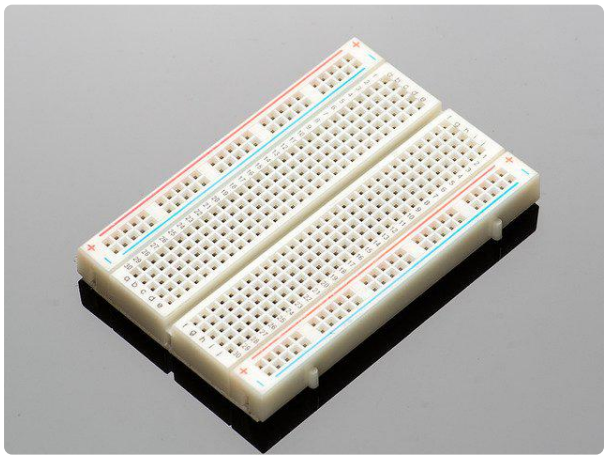
We recommend using a breadboard and some male-male wires.



Premium Male/Male 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/758



Half-size breadboard

This is a cute half-size breadboard, good for small projects. It's 2.2" x 3.4" (5.5 cm x 8.5 cm) with a standard double-strip in the middle and two power rails on both...

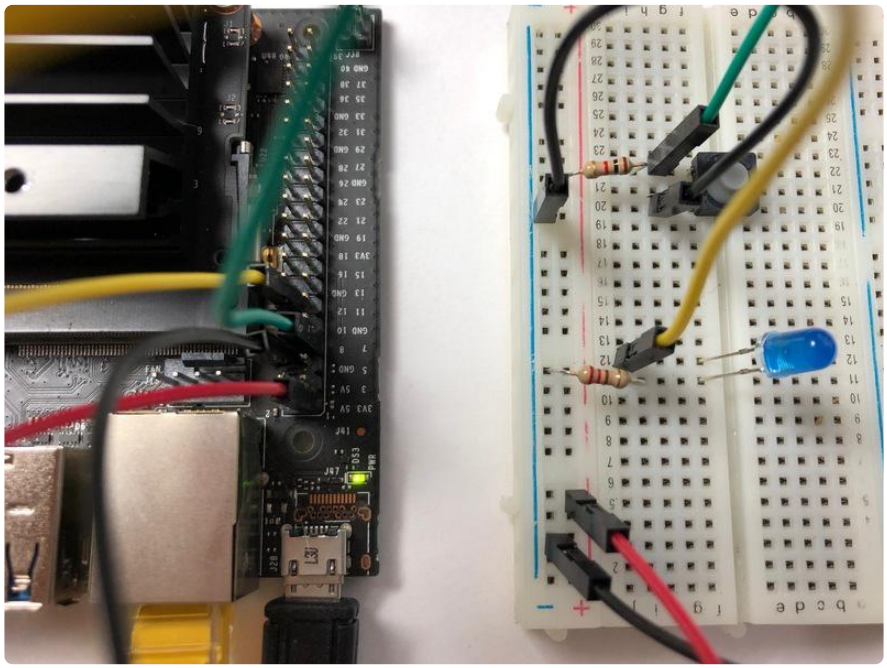
https://www.adafruit.com/product/64

Wiring

- Connect the Jetson Nano Ground pin to the blue ground rail on the breadboard.
- Connect one side of the tactile switch to Jetson Nano D4 (#7)
- Connect a ~1K-10K pull up resistor from D4 to 3.3V
- Connect the other side of the tactile switch to the ground rail
- Connect the longer/positive pin of the LED to Jetson Nano D18 (#12)

- Connect the shorter/negative pin of the LED to a 220 ohm-2.2K resistor, the other side of the resistor goes to ground rail

There's no Jetson Nano Fritzing object, so we sub'd a Raspberry Pi in



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!

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

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):

```
    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 the resistor is installed correctly, and you have Ground and +3.3v wires to the Jetson Nano. Also, check that the wires are going to the correct GPIO pins. If you are powering the board with an AC adapter, try powering it through the USB port.

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:

```
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
# 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

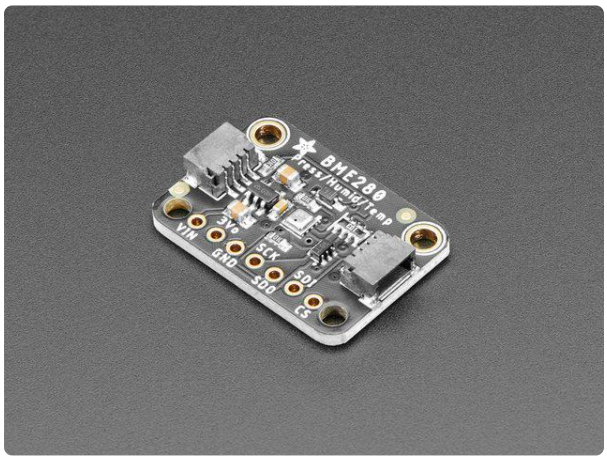
The most popular electronic sensors use I2C to communicate. This is a 'shared bus' 2 wire protocol, you can have multiple sensors connected to the two SDA and SCL pins as long as they have unique addresses (check this guide for a list of many popular devices and their addresses (https://adafru.it/BK0))

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.

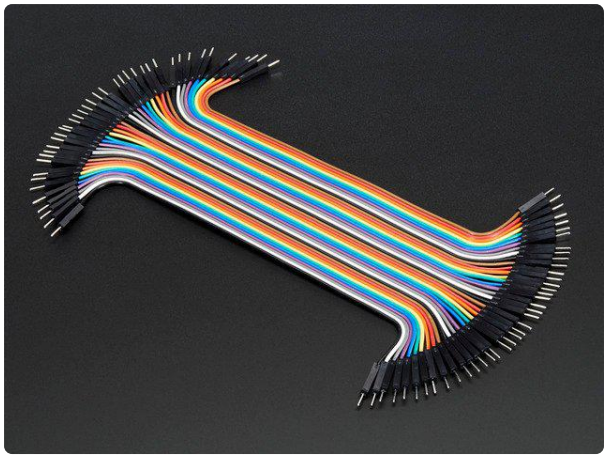
Honestly, the hardest part of using I2C devices is figuring out the I2C address (https://adafru.it/BK0) and which pin is SDA and which pin is SCL!

Parts Used

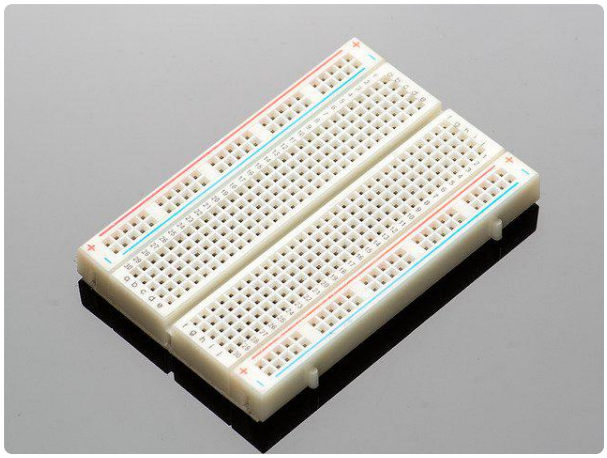


Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor
Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...
https://www.adafruit.com/product/2652

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



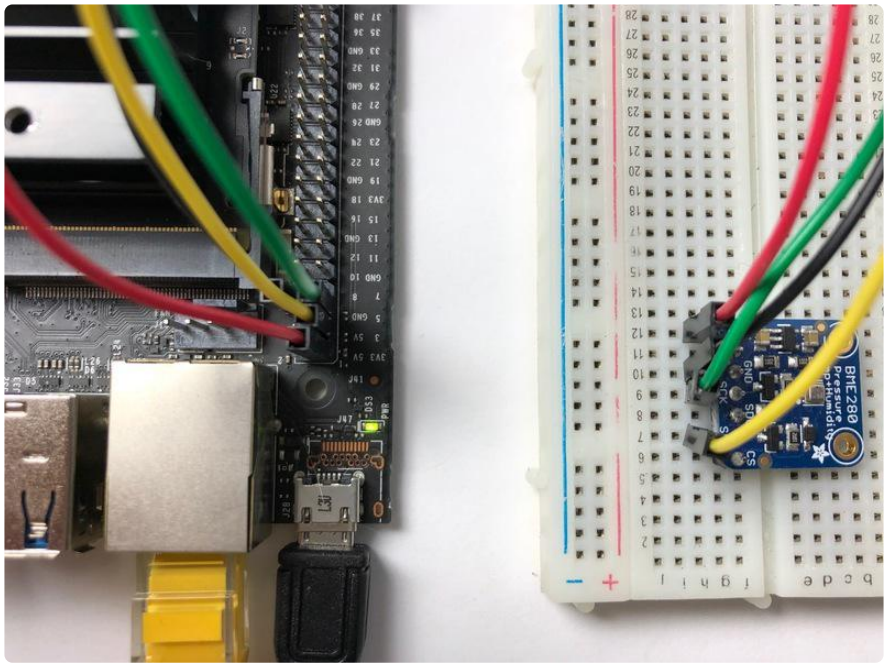
Premium Male/Male Jumper Wires - 40 x 6" (150mm)
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...
https://www.adafruit.com/product/758



Half-size breadboard
This is a cute half-size breadboard, good for small projects. It's 2.2" x 3.4" (5.5 cm x 8.5 cm) with a standard double-strip in the middle and two power rails on both...
https://www.adafruit.com/product/64

Wiring

- Connect the Jetson Nano 3.3V power pin to Vin
- Connect the Jetson Nano GND pin to GND
- Connect the Jetson Nano SDA (#3) pin to the BME280 SDI
- Connect the Jetson Nano SCL (#5) pin to to the BME280 SCK



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

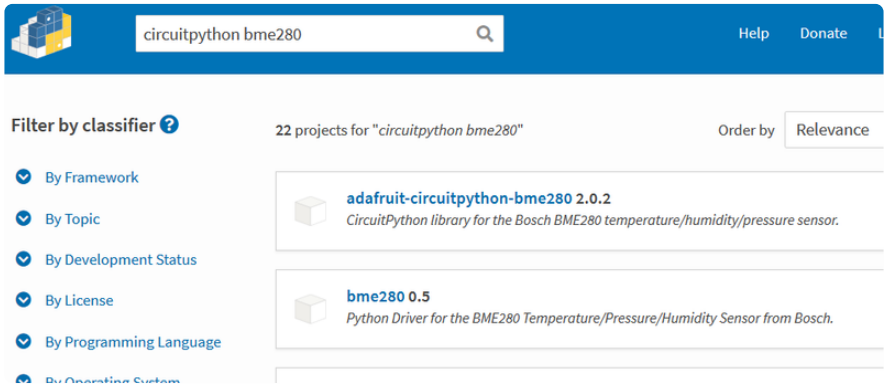
After wiring, we recommend running I2C detection with `i2cdetect -r -y 1` to verify that you see the device, in this case its address 77

```
pi@pi-desktop:~$ i2cdetect -r -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
pi@pi-desktop:~$
```

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 (https://adafru.it/tB7)!)

Once you know the name, install it with

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

```
pi@pi-desktop:~$ pip3 install adafruit-circuitpython-bme280
Collecting adafruit-circuitpython-bme280
Collecting Adafruit-Blinka (from adafruit-circuitpython-bme280)
Collecting adafruit-circuitpython-busdevice (from adafruit-circuitpython-bme280)
Collecting Adafruit-PureIO>=1.1.5 (from Adafruit-Blinka->adafruit-circuitpython-bme280)
Collecting Adafruit-PlatformDetect>=2.15.1 (from Adafruit-Blinka->adafruit-circuitpython
-bme280)
Collecting Jetson.GPIO (from Adafruit-Blinka->adafruit-circuitpython-bme280)
Collecting pyftdi>=0.40.0 (from Adafruit-Blinka->adafruit-circuitpython-bme280)
  Using cached https://files.pythonhosted.org/packages/b8/d4/5f8215300b1c8849ae3b614a312
6edd6f940571f1ce24c175b6ccad8c3df/pyftdi-0.52.0-py3-none-any.whl
Collecting pyserial>=3.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-b
me280)
  Using cached https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c0907414e2a
01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl
Collecting pyusb>=1.0.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-bm
e280)
Installing collected packages: Adafruit-PureIO, Adafruit-PlatformDetect, Jetson.GPIO, py
serial, pyusb, pyftdi, Adafruit-Blinka, adafruit-circuitpython-busdevice, adafruit-circu
itpython-bme280
Successfully installed Adafruit-Blinka-5.4.0 Adafruit-PlatformDetect-2.17.0 Adafruit-Pur
eIO-1.1.5 Jetson.GPIO-2.0.10 adafruit-circuitpython-bme280-2.5.0 adafruit-circuitpython-
busdevice-5.0.1 pyftdi-0.52.0 pyserial-3.4 pyusb-1.1.0
pi@pi-desktop:~$
```

You'll notice we also installed a few other dependancies 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_BME280/tree/master/examples (https://adafru.it/BK1)

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

```
# 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
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)
```

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

Then in your command line run

```
python3 bme280_simpletest.py
```

```
pi@pi-desktop:~$ python3 bme280_simpletest.py

Temperature: 22.8 C
Humidity: 29.7 %
Pressure: 809.2 hPa
Altitude = 1856.78 meters

Temperature: 22.8 C
Humidity: 29.9 %
Pressure: 809.2 hPa
Altitude = 1856.83 meters

Temperature: 22.8 C
Humidity: 29.9 %
Pressure: 809.2 hPa
Altitude = 1856.69 meters

Temperature: 22.8 C
Humidity: 29.9 %
Pressure: 809.2 hPa
Altitude = 1856.71 meters
```

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

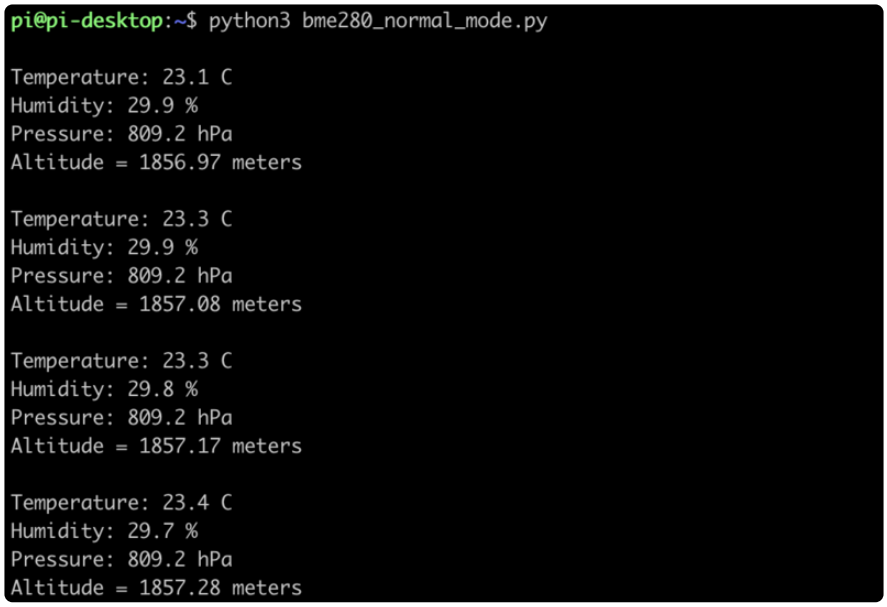
Here's the second example:

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

"""
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
"""
import time
import board
import adafruit_bme280.advanced as adafruit_bme280

# Create sensor object, using the board's default I2C bus.
i2c = board.I2C()  # uses board.SCL and board.SDA
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# OR create sensor object, using the board's default SPI bus.
# SPI setup
# from digitalio import DigitalInOut
# 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
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("\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)
```

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

Then in your command line run

```
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

https://circuitpython.readthedocs.io/projects/bme280/en/latest/ (https://adafru.it/BK2)

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 (cl ock, 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 Microcomputer Out Serial 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 Microcomputer In Serial 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/Jetson Nano WARNING!

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 Jetson Nano there are only two CS pins available for use as 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, it's no big deal

The downside 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 jetson-io!

Using the Second SPI Port

The Jetson Nano 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 the second SPI port by using the Jetson-IO utility and also selecting spi2. See the Initial Setup Page for more details.

Here's the wiring for SPI #2:

- SCK_1 on GPIO #27 (Pin 13)
- MOSI_1 on GPIO #26 (Pin 37)
- MISO_1 on GPIO #25 (Pin 22)
- SPI #2 CS0 on GPIO #24 (Pin 18)
- SPI #2 CS1 on GPIO #23 (Pin 16)

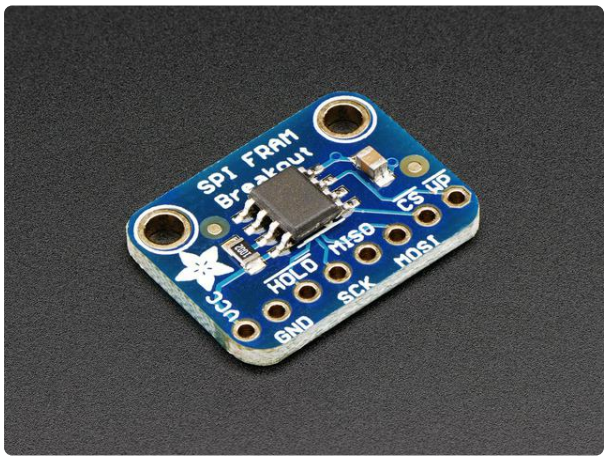
Like the main SPI, we'll use CE0 as our default but don't connect to it! Use any other pin and leave that one unused. Then update your scripts to use

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

Testing SPI with FRAM

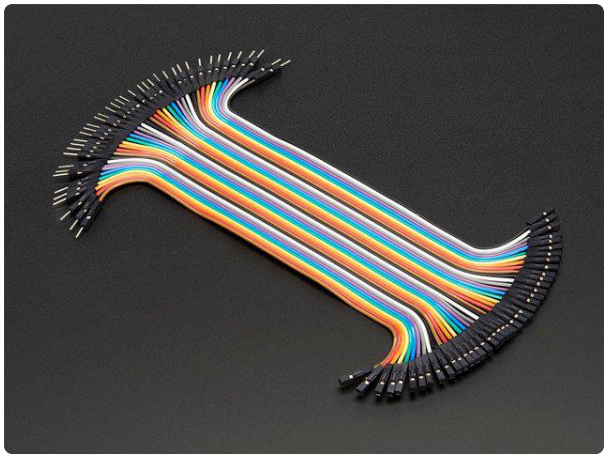
OK now that we've gone thru the warning, let's wire up an SPI FRAM Memory breakout. This will allow us to test both reading and writing.

Parts Used



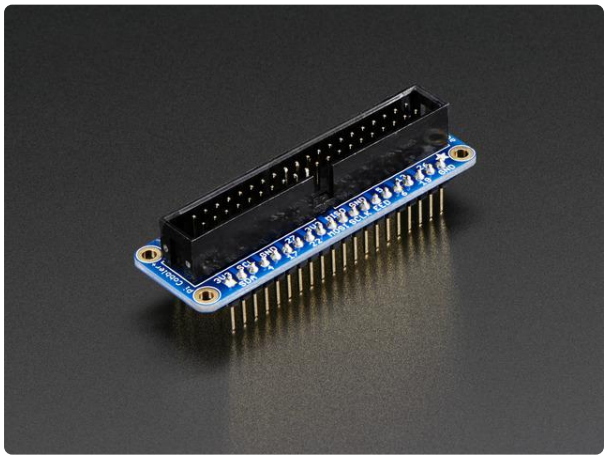
Adafruit SPI Non-Volatile FRAM Breakout - 64Kbit / 8KByte
FRAM, or Ferroelectric Ram, is the coolest new data storage method that all the fashion magazines are talking about. Oh...
https://www.adafruit.com/product/1897

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!

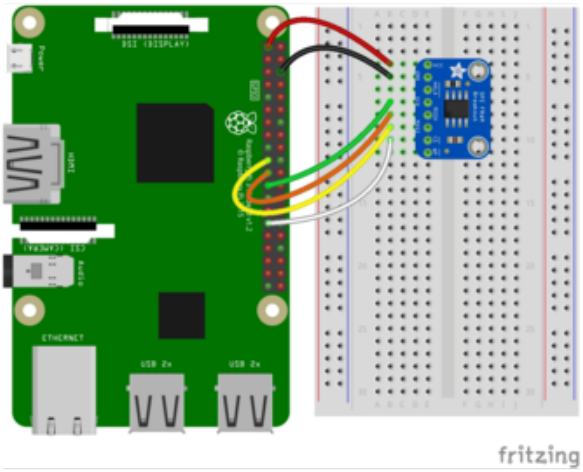


Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3
The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...
https://www.adafruit.com/product/1990

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

There's no Jetson Nano Fritzing object, so we'll show using a Raspberry Pi which has the same pinout



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 FRAM SCK
Connect the Pi MISO pin to to the FRAM MISO
Connect the Pi MOSI pin to to the FRAM MOSI
Connect the Pi GPIO 5 pin to to the FRAM CS

Fritzing Diagram

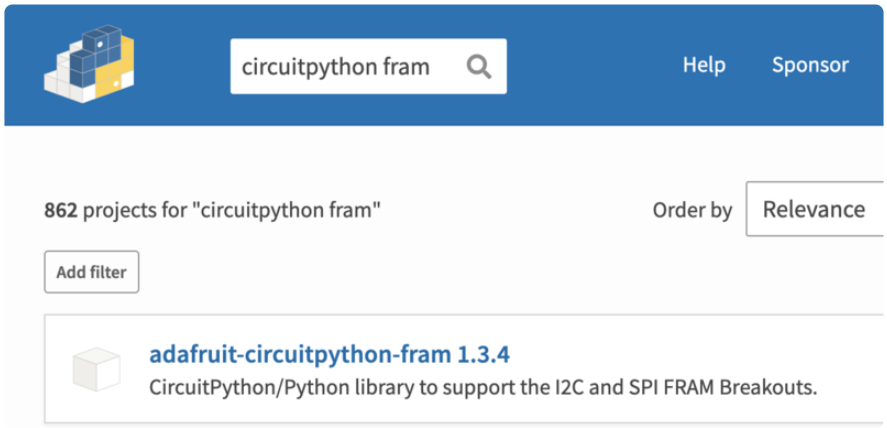
https://adafru.it/NCT

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!

Install the CircuitPython FRAM Library

OK, onto the good stuff. You can now install the Adafruit FRAM 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-fram
```

```
pi@pi-desktop:~$ pip3 install adafruit-circuitpython-fram
Collecting adafruit-circuitpython-fram
Collecting adafruit-circuitpython-busdevice (from adafruit-circuitpython-fram)
Collecting Adafruit-Blinka (from adafruit-circuitpython-fram)
Collecting Jetson.GPIO (from Adafruit-Blinka->adafruit-circuitpython-fram)
Collecting pyftdi>=0.40.0 (from Adafruit-Blinka->adafruit-circuitpython-fram)
  Using cached https://files.pythonhosted.org/packages/b8/d4/5f8215300b1c8849ae3b614a312
6edd6f940571f1ce24c175b6ccad8c3df/pyftdi-0.52.0-py3-none-any.whl
Collecting Adafruit-PlatformDetect>=2.15.1 (from Adafruit-Blinka->adafruit-circuitpython
-fram)
Collecting Adafruit-PureIO>=1.1.5 (from Adafruit-Blinka->adafruit-circuitpython-fram)
Collecting pyserial>=3.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-f
ram)
  Using cached https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c0907414e2a
01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl
Collecting pyusb>=1.0.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-fr
am)
Installing collected packages: Jetson.GPIO, pyserial, pyusb, pyftdi, Adafruit-PlatformDe
tect, Adafruit-PureIO, Adafruit-Blinka, adafruit-circuitpython-busdevice, adafruit-circu
itpython-fram
Successfully installed Adafruit-Blinka-5.4.0 Adafruit-PlatformDetect-2.17.0 Adafruit-Pur
eIO-1.1.5 Jetson.GPIO-2.0.10 adafruit-circuitpython-busdevice-5.0.1 adafruit-circuitpyth
on-fram-1.3.4 pyftdi-0.52.0 pyserial-3.4 pyusb-1.1.0
pi@pi-desktop:~$
```

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_FRAM/tree/master/examples (https://adafru.it/NDs)

As of this writing there's only one example that runs on SPI:

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

## Simple Example For CircuitPython/Python SPI FRAM Library

import board
import busio
import digitalio
import adafruit_fram

## Create a FRAM object.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs = digitalio.DigitalInOut(board.D5)
fram = adafruit_fram.FRAM_SPI(spi, cs)
```

```
## Write a single-byte value to register address '0'

fram[0] = 1

## Read that byte to ensure a proper write.
## Note: 'read()' returns a bytearray

print(fram[0])

## Or write a sequential value, then read the values back.
## Note: 'read()' returns a bytearray. It also allocates
##       a buffer the size of 'length', which may cause
##       problems on memory-constrained platforms.

# values = list(range(100))  # or bytearray or tuple
# fram[0:100] = values
# print(fram[0:100])
```

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

Then in your command line run:

```
python3 fram_spi_simpletest.py
```

The code will write a value of 1 to the first address of the FRAM memory and then read it back:

Feel free to edit the test script and change the value if you want to test it more thoroughly.

Now if you want to read the documentation on the library, what each function does in depth, visit our readthedocs documentation at:

https://circuitpython.readthedocs.io/projects/fram/en/latest/ (https://adafru.it/NCU)

Using a TFT

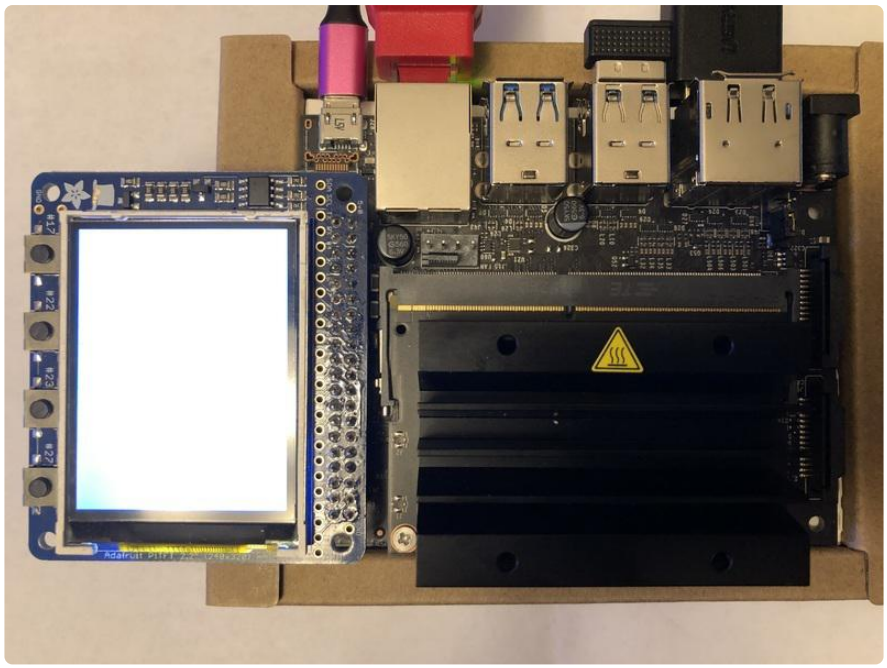
Now that we've done a simple test for reading and writing just a little data, let's test out a TFT which uses much more data, but only uses it one direction. In this example, we're going to use a PiTFT.

Parts

Adafruit PiTFT 2.2" HAT Mini Kit - 320x240 2.2" TFT - No Touch
The cute PiTFT got even more adorable with this little primary display for Raspberry Pi in HAT form! It features a 2.2" display with 320x240 16-bit color pixels. The HAT uses the...
https://www.adafruit.com/product/2315

Wiring

One of the reasons we wanted to use the PiTFT for this section is because it's so easy since you just press it onto the Jetson Nano's GPIO pins. The 40-pin GPIO header is backwards from a Pi, so the PiTFT just sits off to the side.



Install the CircuitPython RGB Display Library

Just like with the previous example, you'll want to install the RGB Display library. The command to install it is:

```
pip3 install adafruit-circuitpython-display
```

```
pi@pi-desktop:~$ pip3 install adafruit-circuitpython-rgb-display
Collecting adafruit-circuitpython-rgb-display
Collecting Adafruit-Blinka (from adafruit-circuitpython-rgb-display)
Collecting adafruit-circuitpython-busdevice (from adafruit-circuitpython-rgb-display)
Collecting Adafruit-PureIO>=1.1.5 (from Adafruit-Blinka->adafruit-circuitpython-rgb-disp
lay)
Collecting pyftdi>=0.40.0 (from Adafruit-Blinka->adafruit-circuitpython-rgb-display)
  Using cached https://files.pythonhosted.org/packages/b8/d4/5f8215300b1c8849ae3b614a312
6edd6f940571f1ce24c175b6ccad8c3df/pyftdi-0.52.0-py3-none-any.whl
Collecting Jetson.GPIO (from Adafruit-Blinka->adafruit-circuitpython-rgb-display)
Collecting Adafruit-PlatformDetect>=2.15.1 (from Adafruit-Blinka->adafruit-circuitpython
-rgb-display)
Collecting pyusb>=1.0.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-rg
b-display)
Collecting pyserial>=3.0 (from pyftdi>=0.40.0->Adafruit-Blinka->adafruit-circuitpython-r
gb-display)
  Using cached https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c0907414e2a
01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl
Installing collected packages: Adafruit-PureIO, pyusb, pyserial, pyftdi, Jetson.GPIO, Ad
afruit-PlatformDetect, Adafruit-Blinka, adafruit-circuitpython-busdevice, adafruit-circu
itpython-rgb-display
Successfully installed Adafruit-Blinka-5.4.0 Adafruit-PlatformDetect-2.17.0 Adafruit-Pur
eIO-1.1.5 Jetson.GPIO-2.0.10 adafruit-circuitpython-busdevice-5.0.1 adafruit-circuitpyth
on-rgb-display-3.10.2 pyftdi-0.52.0 pyserial-3.4 pyusb-1.1.0
pi@pi-desktop:~$
```

If you haven't already updated adafruit_blinka be sure to do that now:

```
pip3 install --upgrade adafruit_blinka
```

Install Pillow

We're going to show you an example that uses Pillow, a popular fork of the Python Imaging Library. First we need to install it:

```
sudo apt install python3-pil
```

Run that Code!

You can now run one of the many example scripts for displays that 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_RGB_Display/tree/master/examples (https://adafru.it/NCV)

We'll just go over one example here as other scripts are covered in other guides. Here's the stats example:

```
# 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,                 # 0.96" MiniTFT
ST7735R
# 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%%\",
```

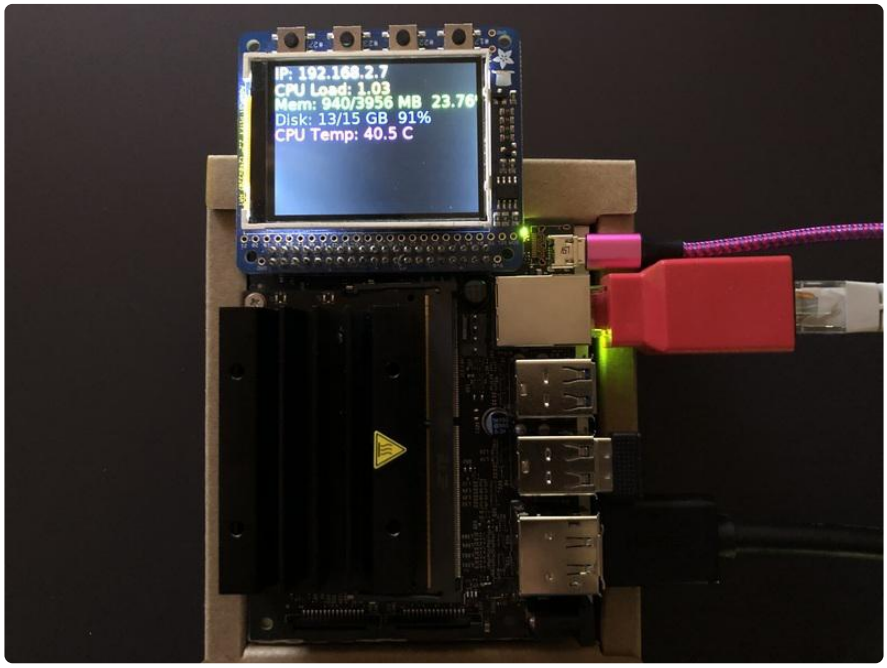
```
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'df -h | awk \'$NF=="/"{printf "Disk: %d/%d GB  %s", $3,$2,$5}\''
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk '{printf \"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)
```

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

Then in your command line run:

```
python3 rgb_display_pillow_stats.py
```

You should see some stats showing up:



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://circuitpython.readthedocs.io/projects/rgb_display/en/latest/ (https://adafru.it/NCW)

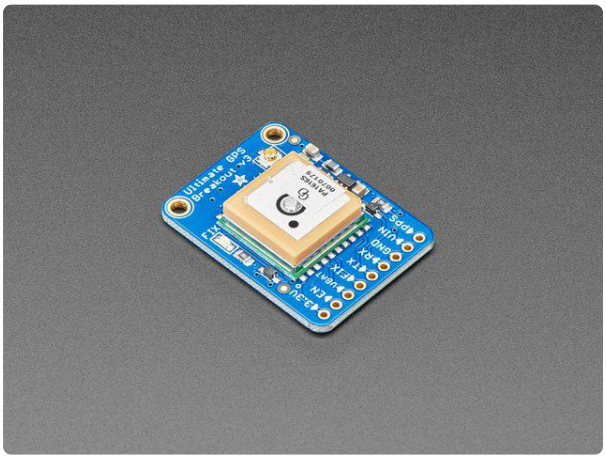
UART / Serial

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 Jetson Nano. 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.

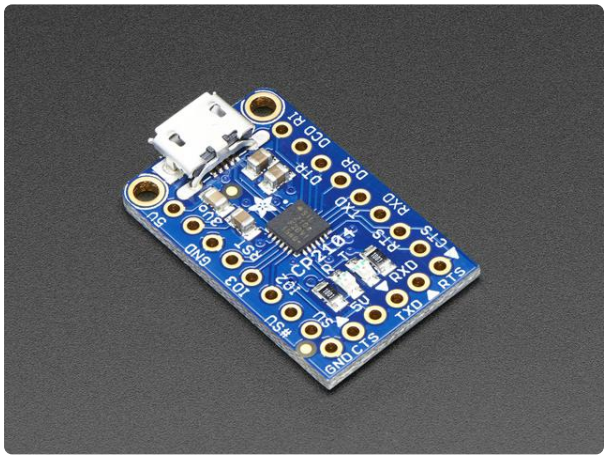
The first cable is easy to use and even has little plugs that you can arrange however you like, it contains a CP2102

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.

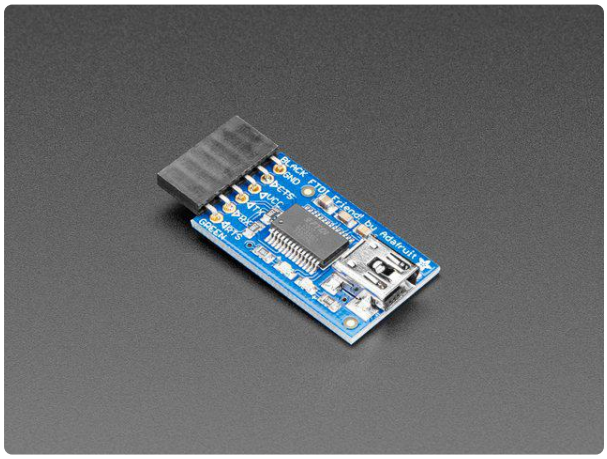


Adafruit CP2104 Friend - USB to Serial Converter

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 every-day serial data from a...

https://www.adafruit.com/product/3309

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



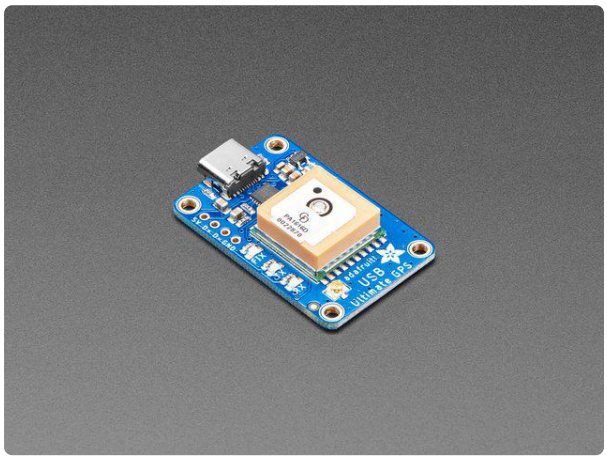
FTDI Friend + extras

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 every-day 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

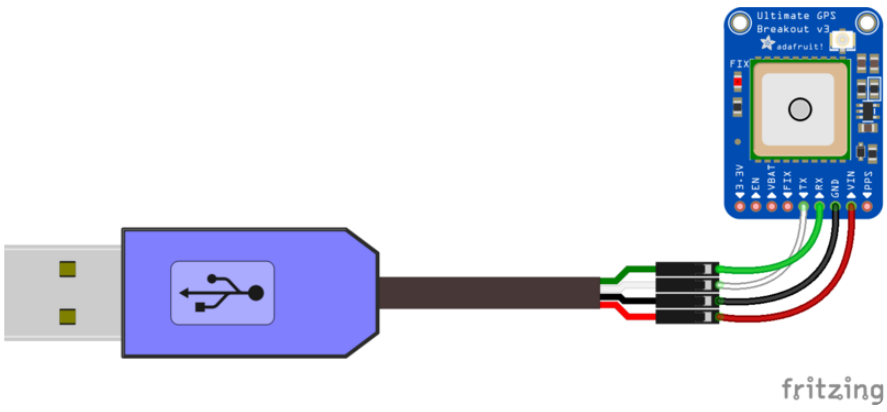
There is also a GPS module with integrated serial available which works like the GPS breakout connected to the USB to TTL Serial cable.



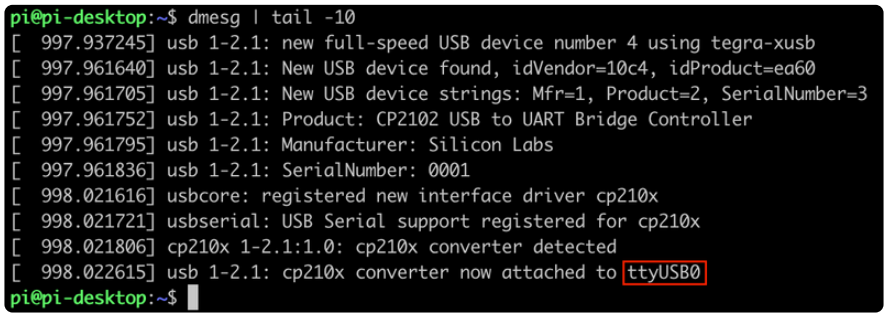
Adafruit Ultimate GPS with USB - 66 channel w/10 Hz updates
The Ultimate GPS module you know and love has a glow-up to let it be easily used with any computer, not just microcontrollers! With the built-in USB-to-Serial converter, you...
https://www.adafruit.com/product/4279

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:



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`

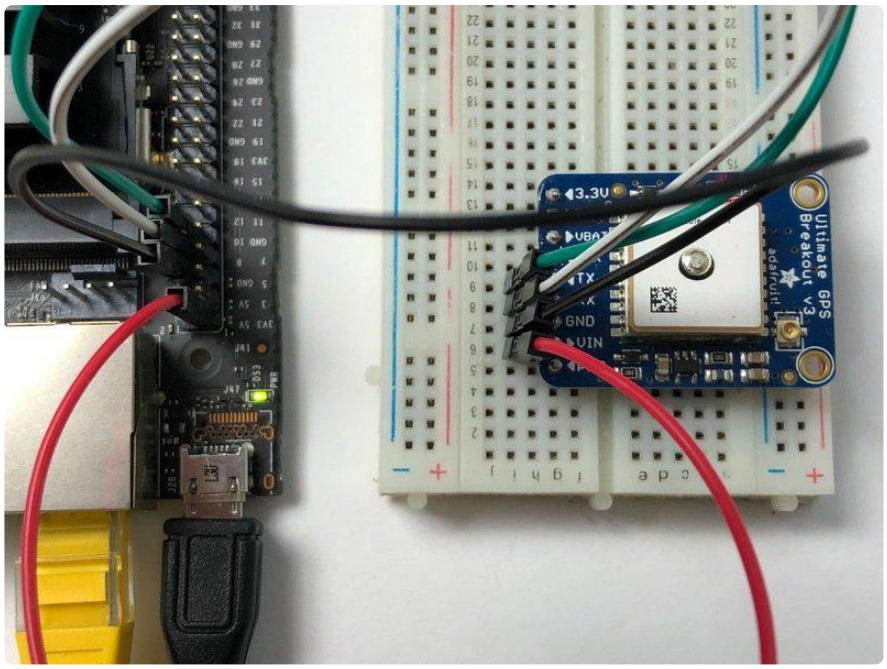
The Hard Way - Using Built-in UART

If you don't want to plug in external hardware to the Jetson Nano you can use the built in UART on the RX/TX pins. Unlike the Raspberry Pi, the Jetson Nano isn't using the RX/TX pins for a console, those are on a different UART peripheral, so you should be good to go!

You can use the built in UART via `/dev/ttyTHS1`

Wire the GPS as follows:

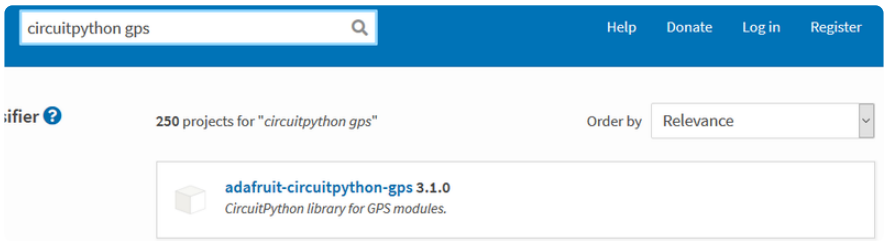
- Connect the Jetson Nano +3.3V pin to the Vin pin on the GPS
- Connect the Jetson Nano Ground pin to the GPS Ground pin
- Connect the Jetson Nano UART TX (#8) to the GPS RX pin
- Connect the Jetson Nano UART RX (#10) to the GPS TX pin



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 pyserial 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://adafru.it/Ca9)

Lets start with the simplest, the echo example:

```
# 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
# 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()

# 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

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

# 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:
        # convert bytearray to string
        data_string = "".join([chr(b) for b in data])
        print(data_string, end="")

    if time.monotonic() - timestamp > 5:
        # every 5 seconds...
        gps.send_command(b"PMTK605")  # request firmware version
        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 DragonBoard, the device name is `/dev/ttyTHS1`.

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!

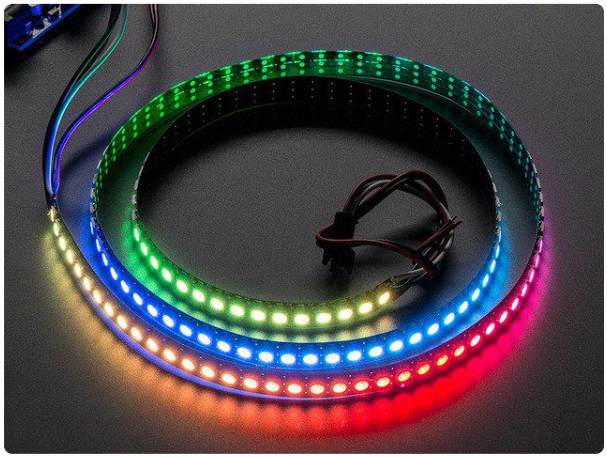
```
pi@pi-desktop:~$ python3 gps_echotest.py
$GPGGA,002749.799,,,,,0,00,,,M,,M,,*77
$GPRMC,002749.799,V,,,,,0.00,0.00,060180,,,N*4D
$PMTK001,314,3*36
$PMTK001,220,3*30
$GPGGA,002750.799,,,,,0,00,,,M,,M,,*7F
$GPRMC,002750.799,V,,,,,0.00,0.00,060180,,,N*45
$GPGGA,002751.799,,,,,0,00,,,M,,M,,*7E
$GPRMC,002751.799,V,,,,,0.00,0.00,060180,,,N*44
$GPGGA,002752.799,,,,,0,00,,,M,,M,,*7D
$GPRMC,002752.799,V,,,,,0.00,0.00,060180,,,N*47
$GPGGA,002753.799,,,,,0,00,,,M,,M,,*7C
$GPRMC,002753.799,V,,,,,0.00,0.00,060180,,,N*46
$GPGGA,002754.799,,,,,0,00,,,M,,M,,*7B
$GPRMC,002754.799,V,,,,,0.00,0.00,060180,,,N*41
$PMTK705,AXN_2.31_3339_13101700,5632,PA6H,1.0*6B
$GPGGA,002755.799,,,,,0,00,,,M,,M,,*7A
$GPRMC,002755.799,V,,,,,0.00,0.00,060180,,,N*40
$GPGGA,002756.799,,,,,0,00,,,M,,M,,*79
$GPRMC,002756.799,V,,,,,0.00,0.00,060180,,,N*43
$GPGGA,002757.799,,,,,0,00,,,M,,M,,*78
```

NeoPixels with SPI

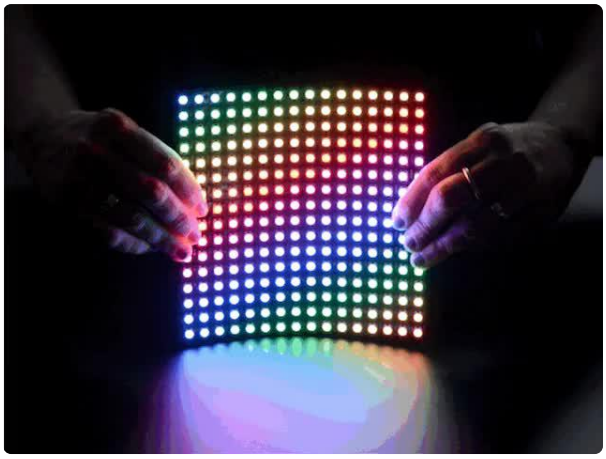
You can even use NeoPixels on the Jetson Nano using the CircuitPython NeoPixel_SPI library (https://adafru.it/NHd). Using this library makes it really easy. The only requirement is you need to enable SPI first. If you haven't already done that, be sure to take a look at the Initial Setup page. For more information about the the NeoPixel SPI library itself, check out our CircuitPython NeoPixel Library Using SPI (https://adafru.it/Gbx) guide.

Parts Used

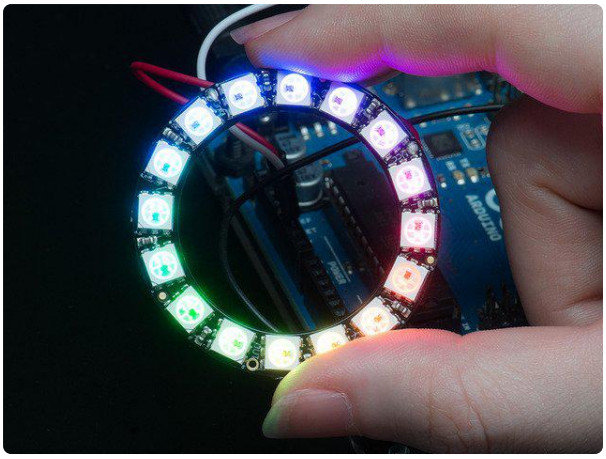
You want some NeoPixels of course. You can use strips, matrices, or even rings.



Adafruit NeoPixel Digital RGB LED Strip 144 LED - 1m Black
We crammed ALL THE NEOPIXELS into this strip! An unbelievable 144 individually-controllable LED pixels on a flexible PCB. It's completely out of control and ready for you to...
https://www.adafruit.com/product/1506



Flexible 16x16 NeoPixel RGB LED Matrix
For advanced NeoPixel fans, we now have a bendable, Flexible 16x16 NeoPixel LED Matrix! Control all 256 ultra-bright LEDs using a single microcontroller pin, set each...
https://www.adafruit.com/product/2547



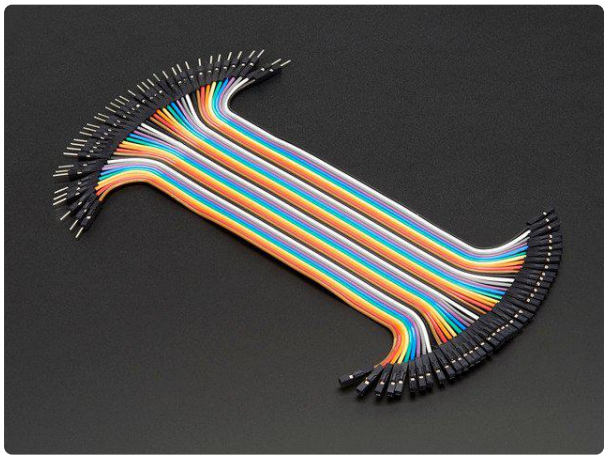
NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers
Round and round and round they go! 16 ultra bright smart LED NeoPixels are arranged in a circle with 1.75" (44.5mm) outer diameter. The rings are 'chainable' - connect the...
https://www.adafruit.com/product/1463

If you're using more than a few NeoPixels (and why wouldn't you want to?), then you'll want an external power supply. For more about power requirements see Powering NeoPixels (https://adafru.it/DCq).

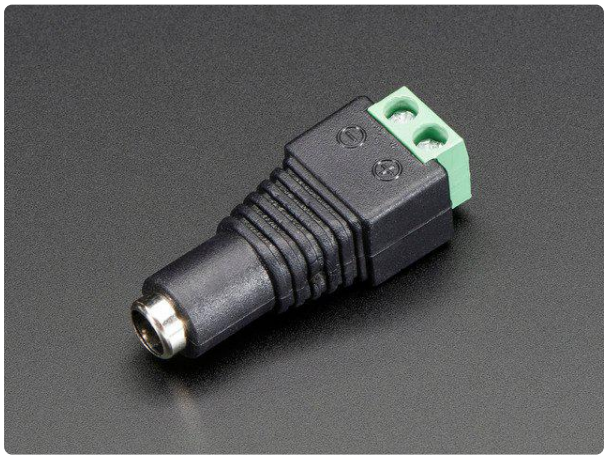
5V 10A switching power supply
This is a beefy switching supply, for when you need a lot of power! It can supply 5V DC up to 10 Amps, running from 110V or 220V power (the plug it comes with is for US/Canada/Japan...
https://www.adafruit.com/product/658

[5V 4A (4000mA) switching power supply - UL Listed]
Need a lot of 5V power? This switching supply gives a clean regulated 5V output at up to 4 Amps (4000mA). 110 or 240 input, so it works in any country. The plugs are "US...
https://www.adafruit.com/product/1466

And here's a couple more products to make it easier to hook it all up.



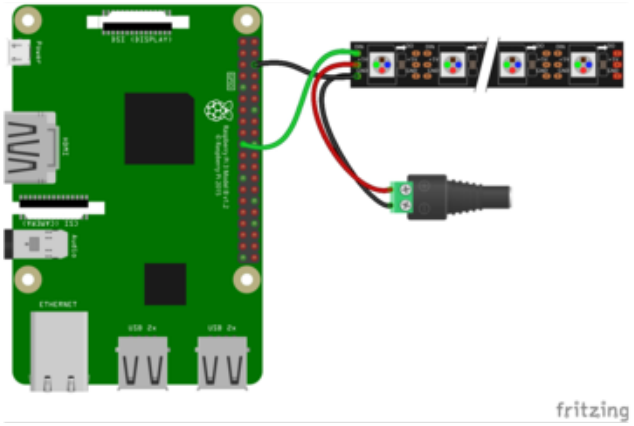
[Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)]
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...
https://www.adafruit.com/product/826



[Female DC Power adapter - 2.1mm jack to screw terminal block]
If you need to connect a DC power wall wart to a board that doesn't have a DC jack - this adapter will come in very handy! There is a 2.1mm DC jack on one end, and a screw terminal...
https://www.adafruit.com/product/368

Wiring

There's no Jetson Nano Fritzing object, so we'll show using a Raspberry Pi which has the same pinout



Connect +5V on the power supply to the NeoPixel VIN
Connect Ground on the power supply to the NeoPixel GND
Connect RPI GND to NeoPixel GND
Connect RPI MOSI to NeoPixel DIN

Fritzing Diagram

https://adafru.it/NHe

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!

Install the required libraries

The CircuitPython NeoPixel_SPI library is very easy to install. Just type:

```
pip3 install adafruit-circuitpython-neopixel-spi
```

If you're using a NeoPixel matrix, you may also want to try out the CircuitPython Pixel Framebuf library as well, which works well with the NeoPixel SPI library.

```
pip3 install adafruit-circuitpython-pixel-framebuf
```

For more information on using the Pixel Framebuf library, check out our Easy NeoPixel Graphics with the CircuitPython Pixel Framebuf Library (https://adafru.it/NHf) guide.

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_NeoPixel_SPI/tree/master/examples (https://adafru.it/NHA)

As of this writing there's only one example that runs NeoPixels on SPI:

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

import time
import board
import neopixel_spi as neopixel

NUM_PIXELS = 12
PIXEL_ORDER = neopixel.GRB
COLORS = (0xFF0000, 0x00FF00, 0x0000FF)
DELAY = 0.1

spi = board.SPI()

pixels = neopixel.NeoPixel_SPI(
    spi, NUM_PIXELS, pixel_order=PIXEL_ORDER, auto_write=False
)

while True:
    for color in COLORS:
        for i in range(NUM_PIXELS):
            pixels[i] = color
            pixels.show()
            time.sleep(DELAY)
            pixels.fill(0)
```

If you are using more than 128 NeoPixels, you will need to make some changes to your Jetson Nano first.

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

You'll want to edit the script and number of NeoPixels that you are actually using and possibly the type if the colors don't look correct.

Then in your command line run:

```
python3 neopixel_spi_simpletest.py
```

Each pixel should individually light up Red, then Green, then Blue and loop back to Red and so on.

Using Many NeoPixels

If you need to use a lot of them, like more than 170 RGB NeoPixels (about 128 for RGBW type), it will require you to make some changes on the Jetson Nano first. This

is because the SPI buffer is set to 4096 by default and each NeoPixel takes 24 bytes. If we use any more than that, it overflows and behaves oddly.

To increase this, we need to make changes in a couple of different places. Although this was tested on the Jetson Nano, it will likely work on other NVIDIA Jetson boards.

Increase the Jetson Nano's Spidev Buffer Size

The first place we increase it is in the Jetson Nano's spidev.bufsize parameter. This is done in the boot/extlinux/extlinux.conf file. We can check the current size with:

`cat /sys/module/spidev/parameters/bufsiz`

Let's change that. Start by opening the config file in an editor:

`sudo nano /boot/extlinux/extlinux.conf`

At the top of the file, you should see a line that looks like DEFAULT [labelname]. If you enabled spi1 using the directions in the Initial Setup, it likely is using JetsonIO which we'll go with for the rest of this section.

```
DEFAULT JetsonIO
```

Scroll down in the file to a section that starts with LABEL and has a name that matches.

```
LABEL JetsonIO
```

Just below that, there is a line that starts with APPEND. Go to the end of that line and at the end add

`spidev.bufsiz=65536`

It should look something like the following:

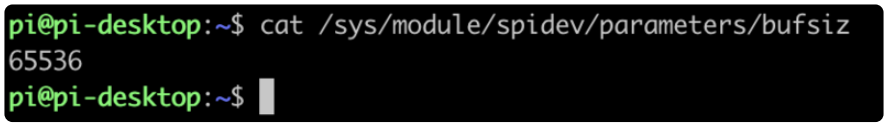
```
APPEND ${cbootargs} quiet root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
console=ttyS0,115200n8 console=tty0 fbcon=map:0 net.ifnames=0 spidev.bufsiz=65536
```

Go ahead and save the file, then exit and reboot:

```
sudo reboot
```

Once it restarts, type the following command to verify that it is now 65536:

```
cat /sys/module/spidev/parameters/bufsiz
```



Change Adafruit PureIO's buffer size

First of all, be sure you're running the latest version of Blinka. This will ensure that you have the latest PureIO as well, which was recently updated for this purpose:

```
pip3 install --upgrade adafruit-blinka
```

To change Adafruit PureIO's buffer size, we can simply set the SPI_BUFSIZE environment variable and it will use that. If you just want to use it once, you can type at the command line:

```
export SPI_BUFSIZE=65536
```

If you want to have it set the variable every time the Jetson Nano boots, you can add it to your .bashrc file, which is located in your home directory. So fi your username is p i, you would find it in /home/pi/.bashrc. Open it up in an editor:

```
nano ~/.bashrc
```

And just add the export line to the bottom of your file and save. Once it is added, you can either reboot or type the following command to immediately reload it:

```
source ~/.bashrc
```

That's it, you're ready to rock with lots of NeoPixels now.

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://adafru.it/BJX) 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 (https://adafru.it/JFy)
- Adding a Single Board Computer to Blinka (https://adafru.it/KEF)

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))

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"

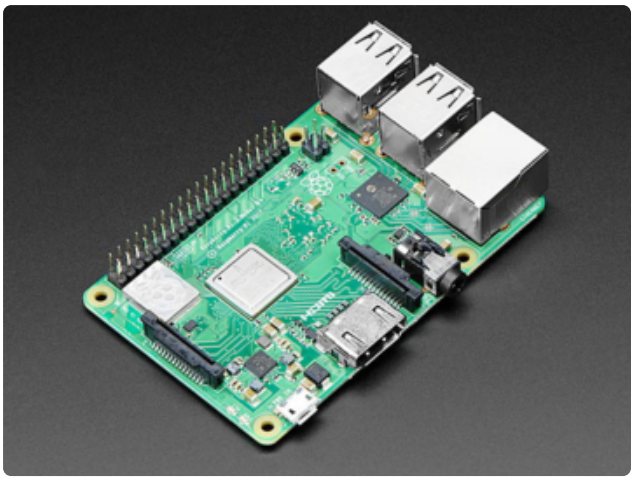
It looks like libgpiod may not be installed on your board.

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

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

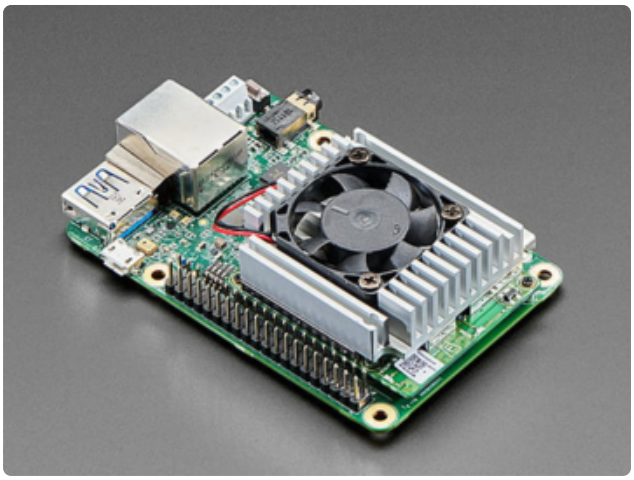
Be sure you have the latest libgpiod.sh script and run it with the `-l` or `--legacy` flag:

```
./libgpiod.sh --legacy
```



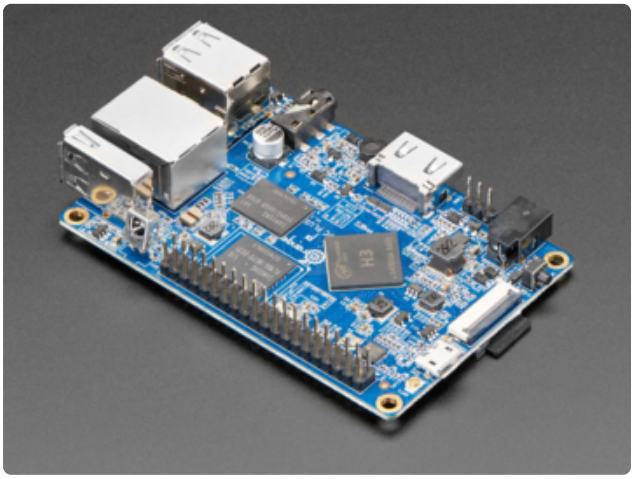
All Raspberry Pi Computers Have:

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
pulseio.pulseIn using gpiod
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)

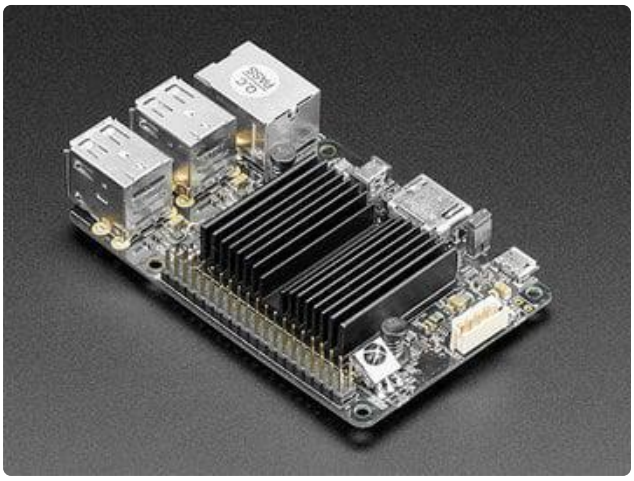


Orange Pi PC Plus Boards Have:

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:

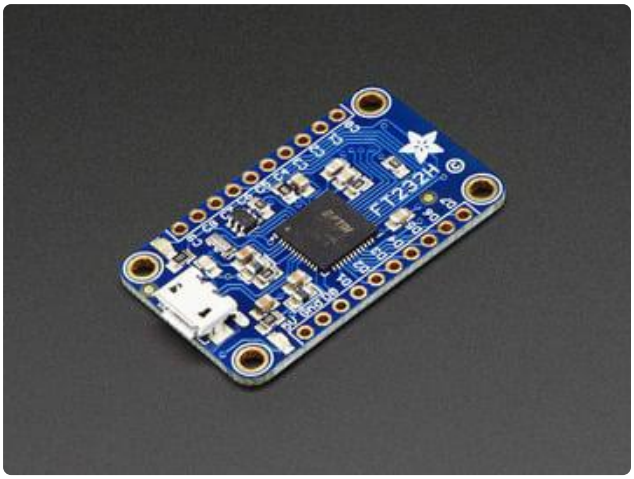
1 x I2C port with busio
No SPI support
1 x UART port with serial - note this is shared with the hardware console
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)

DragonBoard 410c Boards Have:

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

NVIDIA Jetson Nano Boards Have:

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



FT232H Breakouts Have:

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

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

MCP2221 Breakouts Have:

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

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

Downloads

Application Notes

- Customizing the Expansion Header Configuration (https://adafru.it/FMS)