

## CircuitPython on Linux and Orange Pi

Created by lady ada



Last updated on 2018-12-03 02:06:10 AM UTC

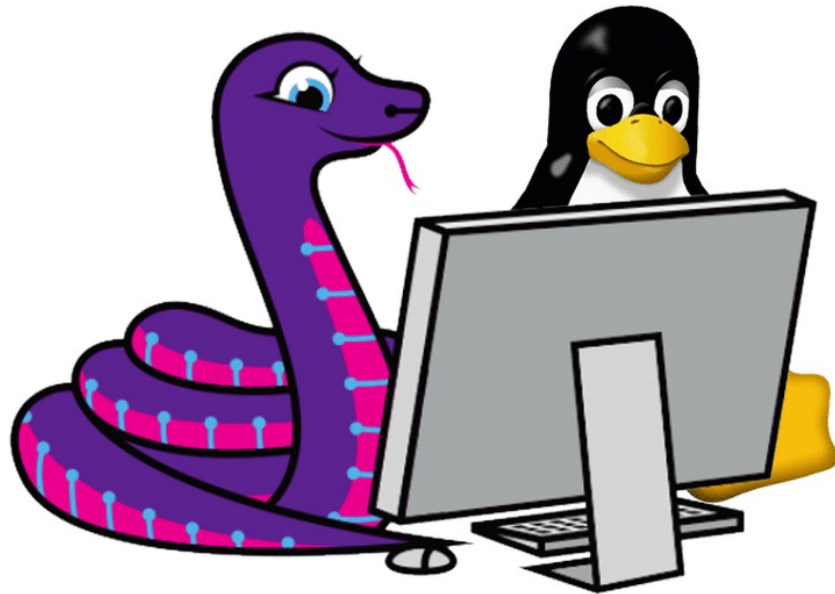
## Guide Contents

Guide Contents	2
Overview	4
Why CircuitPython?	4
CircuitPython on Microcontrollers	4
CircuitPython & OrangePi	6
CircuitPython on Linux & Orange Pi	6
Wait, isn't there already something that does this - WiringOP?	6
What about other Linux SBCs or other Orange Pi's?	7
Initial Setup	8
Install ARMBian on your Orange Pi	8
Logging in	8
USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi	9
Set your Python install to Python 3 Default	10
Install libgpod	10
Update Your Board and Python	11
Enable UART, I2C and SPI	11
Install Python libraries	13
Digital I/O	16
Parts Used	16
Diffused Blue 10mm LED (25 pack)	16
Through-Hole Resistors - 470 ohm 5% 1/4W - Pack of 25	17
Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack	17
Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)	17
Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3	18
Assembled Pi T-Cobbler Plus - GPIO Breakout	18
Wiring	18
Blinky Time!	19
Button It Up	20
I2C Sensors & Devices	21
Parts Used	21
Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor	21
Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)	21
Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3	22
Assembled Pi T-Cobbler Plus - GPIO Breakout	22
Wiring	22
Install the CircuitPython BME280 Library	23
Run that code!	24
SPI Sensors & Devices	26
Parts Used	26
Thermocouple Amplifier MAX31855 breakout board (MAX6675 upgrade)	27
Thermocouple Type-K Glass Braid Insulated	27
Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)	27
Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3	28
Assembled Pi T-Cobbler Plus - GPIO Breakout	28

Wiring	28
Install the CircuitPython MAX31855 Library	29
Run that code!	30
UART / Serial	33
Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates	33
The Easy Way - An External USB-Serial Converter	33
USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi	33
Adafruit CP2104 Friend - USB to Serial Converter	34
FTDI Friend + extras	34
FTDI Serial TTL-232 USB Cable	34
The Hard Way - Using Built-in UART	35
Install the CircuitPython GPS Library	36
Run that code!	36
More To Come!	40
FAQ & Troubleshooting	41
Mixed SPI mode devices	41
No Pullup/Pulldown support on some linux boards	41
Not all peripherals supported	41

## Overview

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!



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...



## CircuitPython & OrangePi



### CircuitPython on Linux & Orange 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 its pretty easy to get it working with micro-computers like Orange Pi or other 'Linux with GPIO pins available' single board computers.

We'll use a special library called [adafruit\\_blinka](https://adafru.it/BJS) (<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 Orange Pi we use the python `libgpiod` bindings. 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 Orange 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 - WiringOP?

WiringOP, a modification of WiringPi, is for GPIO usage only, it doesn't work well for various I2C or SPI devices.

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 WiringOP 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.

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

## What about other Linux SBCs or other Orange Pi's?

Yep! Blinka can easily be updated to add other boards. We've started with the one we've got, so we could test it thoroughly. If you have an OrangePi or other SBC board you'd like to adapt [check out the adafruit\\_blinka code on github \(https://adafru.it/BJX\)](https://adafru.it/BJX), pull requests are welcome as there's a *ton* of different Linux boards out there!

## Initial Setup

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

Once armbian is installed, its easy to tell what board you have, simply `cat /etc/armbian-release` and look for `BOARD_NAME`

```
root@orangeipc:/home/pi# cat /etc/armbian-release
# PLEASE DO NOT EDIT THIS FILE
BOARD=orangeipc
BOARD_NAME="Orange Pi PC"
BOARDFAMILY=sun8i
VERSION=5.65
LINUXFAMILY=sunxi
BRANCH=next
ARCH=arm
IMAGE_TYPE=stable
BOARD_TYPE=conf
INITRD_ARCH=arm
KERNEL_IMAGE_TYPE=zImage
root@orangeipc:/home/pi#
```

## Install ARMBian on your Orange Pi

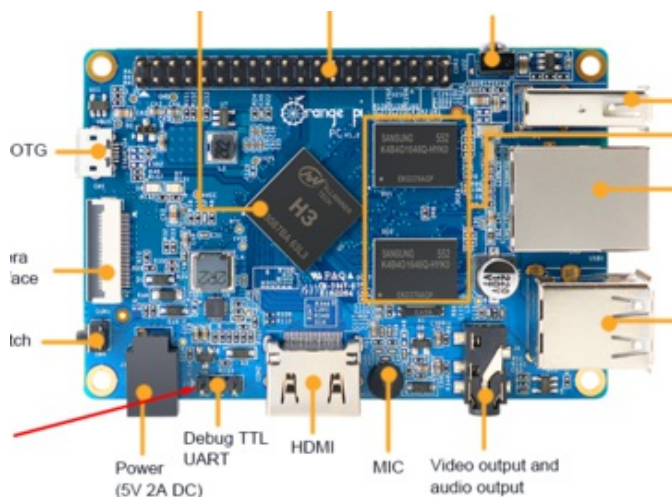
We're only going to be using armbian, other distros could be made to work but you'd probably need to figure out how to detect the platform since we rely on `/etc/armbian-release` existing.

Download and install the latest armbian, for example we're using <https://www.armbian.com/orange-pi-pc-plus/> (<https://adafru.it/Dbx>)

There's some documentation to get started at [https://docs.armbian.com/User-Guide\\_Getting-Started/](https://docs.armbian.com/User-Guide_Getting-Started/) (<https://adafru.it/Dby>)

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

## Logging in



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





### USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

\$9.95  
IN STOCK

ADD TO CART

Once powered correctly and with the right SD card you should get a login prompt

```
[ OK ] Started /etc/rc.local Compatibility.  
[ OK ] Started Serial Getty on ttyS0.  
[ OK ] Started Getty on tty1.  
[ OK ] Reached target Login Prompts.  
[ OK ] Started LSB: Start NTP daemon.  
[ OK ] Reached target Multi-User System.  
[ OK ] Reached target Graphical Interface.  
Starting Update UTMP about System Runlevel Changes..  
[ OK ] Started Update UTMP about System Runlevel Changes.  
Debian GNU/Linux 9 orangepi3c ttyS0  
orangepi3c login: █
```

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

```
Please provide a username (eg. your forename): pi  
Trying to add user pi  
Adding user 'pi' ...  
Adding new group 'pi' (1000) ...  
Adding new user 'pi' (1000) with group 'pi' ...  
Creating home directory '/home/pi' ...  
Copying files from '/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for pi
```

```
orangeipc login: pi
Password:

Welcome to ARMBIAN 5.65 stable Debian GNU/Linux 9 (stretch) 4.14.78-sunxi
System load:  0.91 0.46 0.18   Up time:       1 min
Memory usage: 5 % of 1000MB   IP:           10.0.1.188
CPU temp:     44°C
Usage of /:    6% of 15G

[ General system configuration (beta): armbian-config ]

pi@orangeipc:~$
```

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

- `sudo apt-get install avahi-daemon`

then `reboot`

## Set your Python install to Python 3 Default

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

- `sudo apt-get install -y python3 git python3-pip`
- `sudo update-alternatives --install /usr/bin/python python /usr/bin/python2.7 1`
- `sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.5 2`
- `sudo update-alternatives --config python`

Of course change the version numbers if newer Python is distributed

## Install libgpiod

libgpiod is what we use for gpio toggling, it doesn't come in installations yet but its easy to add by [running our script](#) (<https://adafru.it/Dbz>). You'll probably need to run this as root, so `sudo bash` before you...

- `cd ~`
- `wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/libgpiod.sh`
- `chmod +x libgpiod.sh`
- `./libgpiod.sh`

```

pi@orangeipc:~$ ./libgpiod.sh
Installing build requirements - this may take a few minutes!

Hit:1 http://security.debian.org stretch/updates InRelease
Ign:2 http://cdn-fastly.deb.debian.org/debian stretch InRelease
Hit:4 http://cdn-fastly.deb.debian.org/debian stretch-updates InRelease
Hit:5 http://cdn-fastly.deb.debian.org/debian stretch-backports InRelease
Hit:6 http://cdn-fastly.deb.debian.org/debian stretch Release
Hit:3 https://apt.armbian.com stretch InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package raspberrypi-kernel-headers
Cloning libgpiod repository to /tmp/libgpiod.jKdv

Cloning into '.'...
remote: Counting objects: 4018, done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 4018 (delta 27), reused 0 (delta 0)
Receiving objects: 100% (4018/4018), 633.31 KiB | 1.10 MiB/s, done.
Resolving deltas: 100% (2751/2751), done.
Building libgpiod

```

After installation you should be able to `import gpiod` from within Python3

```

root@orangeipc:/home/pi# python
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> import gpiod
>>>

```

## Update Your Board and Python

Run the standard updates:

```
sudo apt-get update
```

```
sudo apt-get 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
```

## Enable UART, I2C and SPI

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

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

---

Install the support software with

- `sudo apt-get install -y python-smbus python-dev i2c-tools`
- `sudo adduser pi i2c`

```
pi@orangepipc:~$ sudo apt-get install -y python-smbus python-dev i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
i2c-tools is already the newest version (3.1.2-3).
python-smbus is already the newest version (3.1.2-3).
python-dev is already the newest version (2.7.13-2).
0 upgraded, 0 newly installed, 0 to remove and 38 not upgraded.
pi@orangepipc:~$ sudo adduser pi i2c
Adding user 'pi' to group 'i2c' ...
Adding user pi to group i2c
Done.
pi@orangepipc:~$
```

---

Read `/etc/armbian-release` to figure out your board family, in this case its a **sun8i**

```
pi@orangepipc:~$ cat /etc/armbian-release
# PLEASE DO NOT EDIT THIS FILE
BOARD=orangepipc
BOARD_NAME="Orange Pi PC"
BOARDFAMILY=sun8i
VERSION=5.65
LINUXFAMILY=sunxi
BRANCH=next
ARCH=arm
IMAGE_TYPE=stable
BOARD_TYPE=conf
INITRD_ARCH=arm
KERNEL_IMAGE_TYPE=zImage
pi@orangepipc:~$
```

---

Edit `/boot/armbianEnv.txt` and add these lines at the end, adjusting the `overlay_prefix` for your particular board

```
overlay_prefix=sun8i-h3
overlays=uart3 i2c0 spi-spidev
param_spidev_spi_bus=0
```

```
overlay_prefix=sun8i-h3
overlays=uart3 i2c0 spi-spidev
param_spidev_spi_bus=0
```

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 at least one i2c device and one spi device

```
pi@orangepipc:~$ ls /dev/i2c* /dev/spi*
/dev/i2c-0 /dev/i2c-1 /dev/i2c-2 /dev/spidev0.0
pi@orangepipc:~$
```

```
pi@orangepipc:~$ sudo i2cdetect -l
i2c-1  i2c          mv64xxx_i2c adapter      I2C adapter
i2c-2  i2c          DesignWare HDMI         I2C adapter
i2c-0  i2c          mv64xxx_i2c adapter      I2C adapter
```

```

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

```

You can test to see what I2C addresses are connected by running `sudo i2cdetect -y 0` (on PA11/PA12) or `sudo i2cdetect -y 1` (on PA18/PA19)

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

You can also install WiringOP and then run `gpio readall` to see the MUX status. If you see **ALT3** next to a pin, its a plain GPIO. If you see **ALT4** or **ALT5**, its an SPI/I2C/UART peripheral

```

pi@orangepipc:~$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      |      | 3.3v |      |   | 1 | 2 |      | 5v |      |      | |
| 12 | 8 | SDA.0 | ALT5 | 0 | 3 | 4 |      | 5V |      |      |
| 11 | 9 | SCL.0 | ALT5 | 0 | 5 | 6 |      | 0v |      |      |
| 6 | 7 | GPIO.7 | ALT3 | 0 | 7 | 8 |      | ALT4 | TxD3 | 15 | 13 |
|      |      | 0v |      |   | 9 | 10 |      | ALT4 | RxD3 | 16 | 14 |
| 1 | 0 | RxD2 | ALT3 | 0 | 11 | 12 |      | ALT3 | GPIO.1 | 1 | 110 |
| 0 | 2 | TxD2 | ALT3 | 0 | 13 | 14 |      |      | 0v |      |      |
| 3 | 3 | CTS2 | ALT3 | 0 | 15 | 16 |      | ALT3 | GPIO.4 | 4 | 68 |
|      |      | 3.3v |      |   | 17 | 18 |      | ALT3 | GPIO.5 | 5 | 71 |
| 64 | 12 | MOSI | ALT4 | 0 | 19 | 20 |      |      | 0v |      |      |
| 65 | 13 | MISO | ALT4 | 0 | 21 | 22 |      | ALT3 | RTS2 | 6 | 2 |
| 66 | 14 | SCLK | ALT4 | 0 | 23 | 24 |      | ALT4 | CE0 | 10 | 67 |
|      |      | 0v |      |   | 25 | 26 |      | ALT3 | GPIO.11 | 11 | 21 |
| 19 | 30 | SDA.1 | ALT3 | 0 | 27 | 28 |      | ALT3 | SCL.1 | 31 | 18 |
| 7 | 21 | GPIO.21 | ALT3 | 0 | 29 | 30 |      |      | 0v |      |      |
| 8 | 22 | GPIO.22 | ALT3 | 0 | 31 | 32 |      | ALT3 | RTS1 | 26 | 200 |
| 9 | 23 | GPIO.23 | ALT3 | 0 | 33 | 34 |      |      | 0v |      |      |
| 10 | 24 | GPIO.24 | ALT3 | 0 | 35 | 36 |      | ALT3 | CTS1 | 27 | 201 |
| 20 | 25 | GPIO.25 | ALT3 | 0 | 37 | 38 |      | ALT3 | TxD1 | 28 | 198 |
|      |      | 0v |      |   | 39 | 40 |      | ALT3 | RxD1 | 29 | 199 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

## Install Python libraries

Now you're ready to install all the python support

Run the following command to install `adafruit_blinka`

```

pip3 install adafruit-blinka

```

```
pi@devpi: ~/py
(py) pi@devpi:~/py $ pip install adafruit-blinka
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting adafruit-blinka
Collecting Adafruit-GPIO (from adafruit-blinka)
  Using cached https://www.piwheels.org/simple/adafruit-gpio/Adafruit_GPIO-1.0.3-py3-none-any.whl
Collecting adafruit-pureio (from Adafruit-GPIO->adafruit-blinka)
  Using cached https://www.piwheels.org/simple/adafruit-pureio/Adafruit_PureIO-0.2.3-py3-none-any.whl
Collecting spidev (from Adafruit-GPIO->adafruit-blinka)
  Using cached https://www.piwheels.org/simple/spidev/spidev-3.2-cp35-cp35m-linux_armv7l.whl
Installing collected packages: adafruit-pureio, spidev, Adafruit-GPIO, adafruit-blinka
Successfully installed Adafruit-GPIO-1.0.3 adafruit-blinka-0.1.6 adafruit-pureio-0.2.3 spidev-3.2
(py) pi@devpi:~/py $ |
```

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.

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.PA6)
print("Digital IO ok!")

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

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

print("done!")
```

Save it and run at the command line with

```
sudo python3 blinkatest.py
```

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

```
root@orangepipc:/home/pi# python3 blinkatest.py
Hello blinka!
Digital IO ok!
I2C ok!
SPI ok!
done!
root@orangepipc:/home/pi#
```

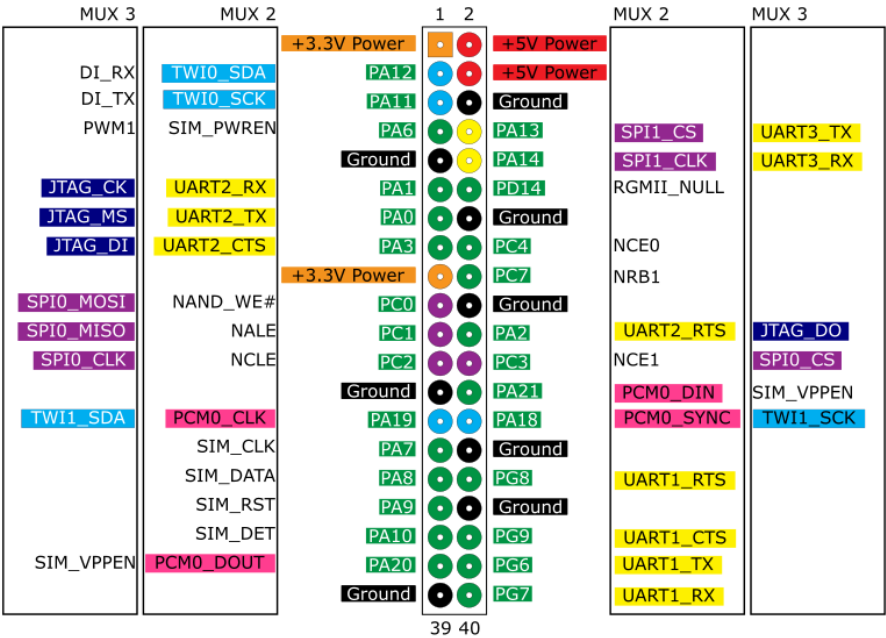


# Digital I/O

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

Orange Pi (Allwinner) 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!

## Orange Pi (H3 SoC) GPIO - pinout



NOTE: GPIO voltage levels are 3.3V.

JTAG I2C SPI +5V GPIO UART +3.3V Ground I2S/PCM

## Parts Used

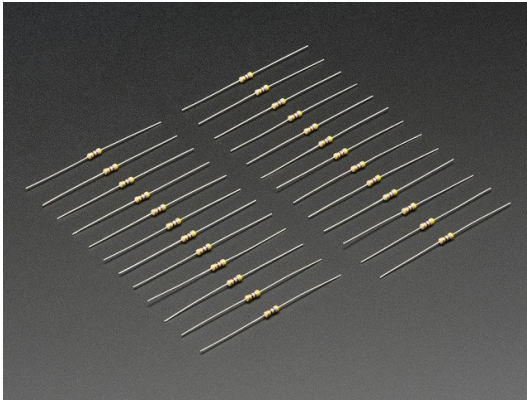
Any old LED will work just fine as long as its not an IR LED (you can't see those) and a 470 to 2.2K resistor



Diffused Blue 10mm LED (25 pack)

\$9.95  
OUT OF STOCK  
OUT OF STOCK





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

\$0.75  
IN STOCK

ADD TO CART

---

Some tactile buttons or switches



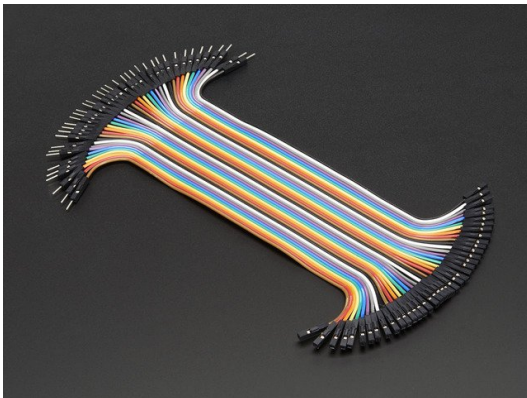
Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack

\$2.50  
IN STOCK

ADD TO CART

---

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



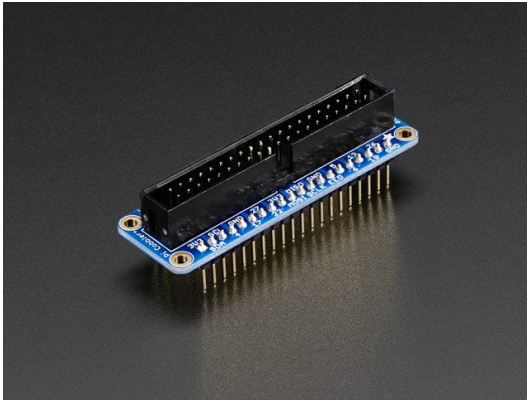
Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

\$3.95  
IN STOCK

ADD TO CART

---

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



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3

\$6.50  
OUT OF STOCK

OUT OF STOCK



Assembled Pi T-Cobbler Plus - GPIO Breakout

\$7.95  
IN STOCK

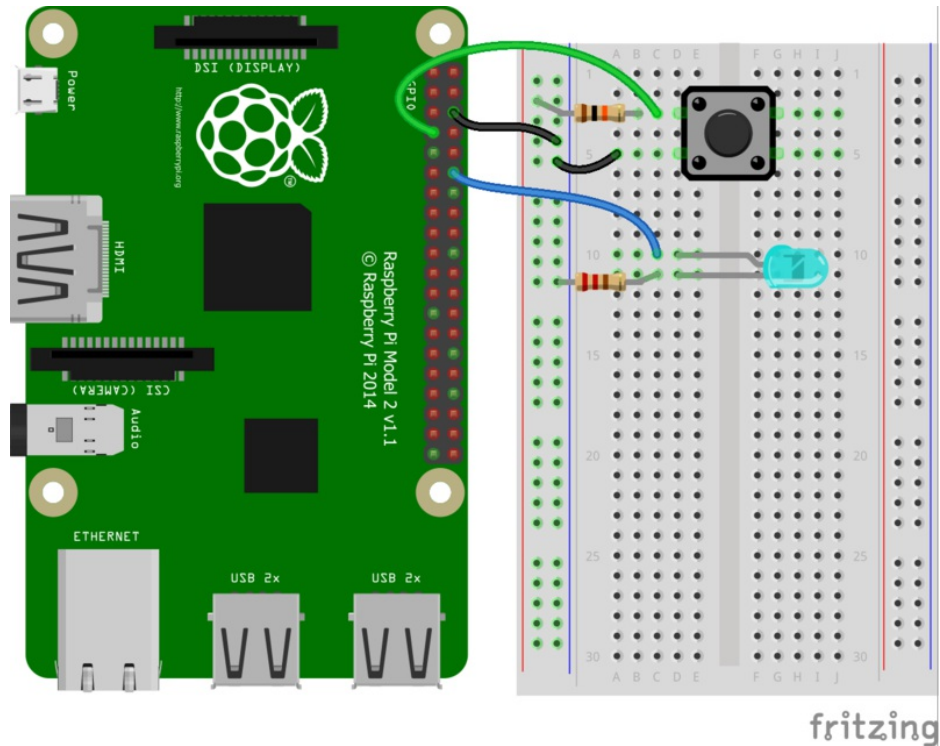
ADD TO CART

## Wiring

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

- Connect one side of the tactile switch to Orange Pi **GPIO PA6**
- Connect a ~10K pull up resistor from **PA6** to **3.3V**
- Connect the other side of the tactile switch to the **ground rail**
- Connect the longer/positive pin of the LED to Orange Pi **GPIO PD14**
- 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**

There's no Orange Pi PC Fritzing object, so we sub'd a Raspberry Pi in



<https://adafru.it/Dbw>

<https://adafru.it/Dbw>

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!

```
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.PD14)
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 PD14, the resistor is installed correctly, and you have a Ground wire to the Orange Pi.

Type Control-C to quit

## Button It Up

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

```

import time
import board
import digitalio

print("press the button!")

led = digitalio.DigitalInOut(board.PD14)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.PA6)
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 \(https://adafru.it/BK0\)](https://adafru.it/BK0))

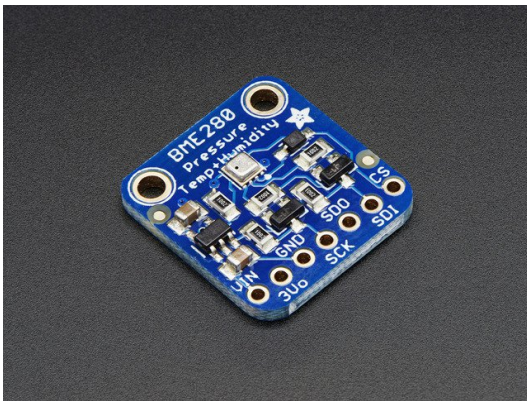
Lets show how to wire up a popular BME280. This sensor provides temperature, barometric pressure and humidity data over I2C

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 \(https://adafru.it/BK0\)](https://adafru.it/BK0) and which pin is SDA and which pin is SCL!

Don't forget you have to enable I2C overlay

### Parts Used

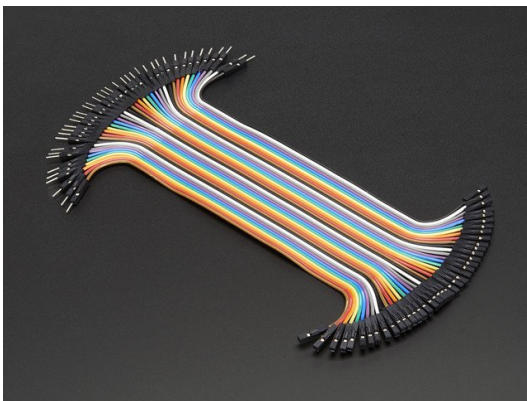


Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor

\$19.95  
IN STOCK

ADD TO CART

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

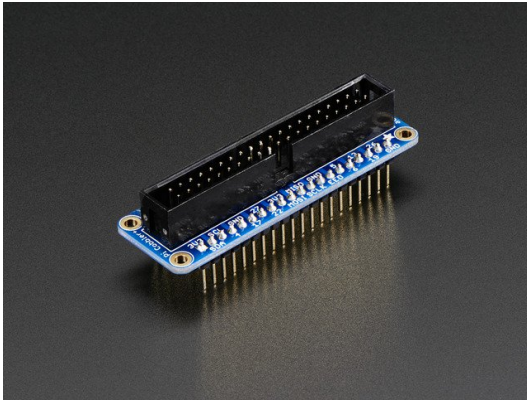


Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

\$3.95  
IN STOCK

ADD TO CART

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



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3

\$6.50  
OUT OF STOCK

OUT OF STOCK



Assembled Pi T-Cobbler Plus - GPIO Breakout

\$7.95  
IN STOCK

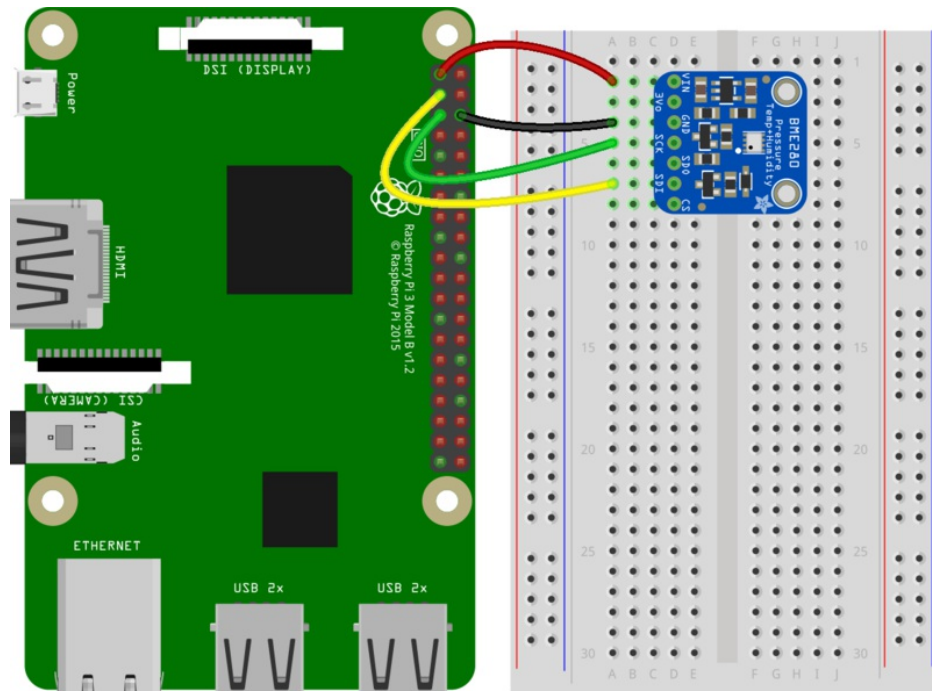
ADD TO CART

## Wiring

- Connect the Orange Pi **3.3V** power pin to **Vin**
- Connect the Orange Pi **GND** pin to **GND**
- Connect the Pi **SDA** pin to the BME280 **SDI**
- Connect the Pi **SCL** pin to to the BME280 **SCK**

There's no Orange Pi PC Fritzing object so we're showing Raspberry Pi which has the same pinout





fritzing

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

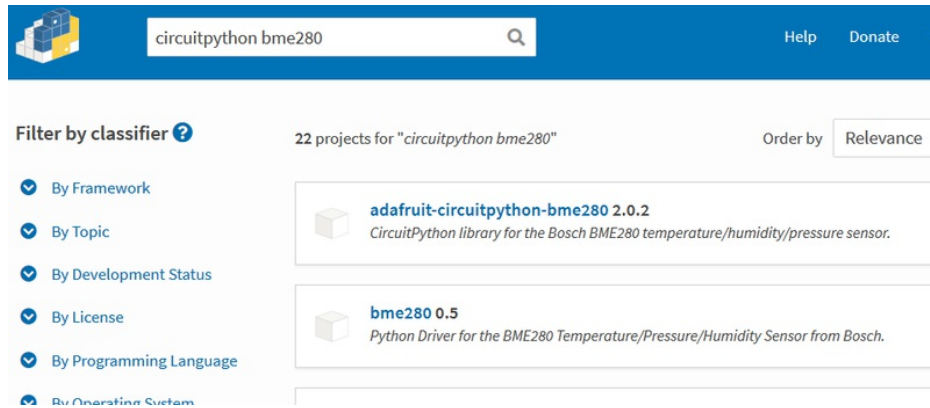
After wiring, we recommend running I2C detection with `sudo i2cdetect -y 0` to verify that you see the device, in this case its address **77**

```
root@orangeipc:/home/pi# i2cdetect -y 0
 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
```

## 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@devpi: ~/py
(py) pi@devpi:~/py $ pip install adafruit-circuitpython-bme280
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting adafruit-circuitpython-bme280
Requirement already satisfied: Adafruit-Blinka in ./lib/python3.5/site-packages
(from adafruit-circuitpython-bme280) (0.1.6)
Collecting adafruit-circuitpython-busdevice (from adafruit-circuitpython-bme280)
Requirement already satisfied: Adafruit-GPIO in ./lib/python3.5/site-packages (f
rom Adafruit-Blinka->adafruit-circuitpython-bme280) (1.0.3)
Requirement already satisfied: spidev in ./lib/python3.5/site-packages (from Ada
fruit-GPIO->Adafruit-Blinka->adafruit-circuitpython-bme280) (3.2)
Requirement already satisfied: adafruit-pureio in ./lib/python3.5/site-packages
(from Adafruit-GPIO->Adafruit-Blinka->adafruit-circuitpython-bme280) (0.2.3)
Installing collected packages: adafruit-circuitpython-busdevice, adafruit-circui
tpython-bme280
Successfully installed adafruit-circuitpython-bme280-2.0.2 adafruit-circuitpytho
n-busdevice-2.2.2
(py) pi@devpi:~/py $ |

```

You'll notice we also installed a *dependancy* 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](https://github.com/adafruit/Adafruit_CircuitPython_BME280/tree/master/examples) (<https://adafru.it/BK1>)

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

```

Save this code to your 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`

```

root@orangepipc:/home/pi# python3 bme280test.py

Temperature: 23.9 C
Humidity: 32.6 %
Pressure: 1015.2 hPa
Altitude = -16.39 meters

Temperature: 23.9 C
Humidity: 32.7 %
Pressure: 1015.2 hPa
Altitude = -16.31 meters
^CTraceback (most recent call last):
  File "bme280test.py", line 24, in <module>
    time.sleep(2)
KeyboardInterrupt
root@orangepipc:/home/pi#

```

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](https://circuitpython.readthedocs.io/projects/bme280/en/latest/) 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 (**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/Orange PI 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 Orange Pi PC, there's only one CS pins available for the hardware SPI pins - **SPI\_CS** known as **PC3** - and you *have* to use it. (In theory there's an `ioctl` option called `no_cs` but this does not actually work)

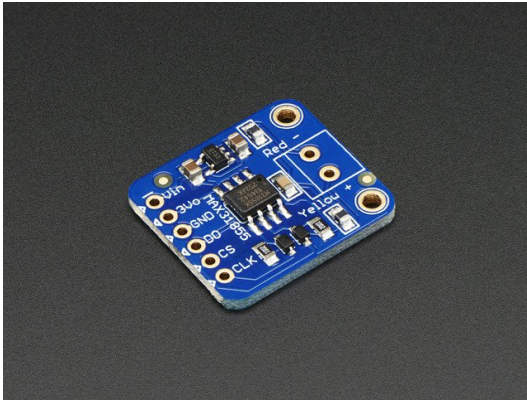
The upshot here is - to let you use more than 1 peripheral on SPI, we decided to **let you use any CS pins** you like, CircuitPython will toggle it the way you expect. But when we transfer SPI data we always tell the kernel to use **SPI\_CS**. **SPI\_CS** will toggle like a CS pin, but if we leave it disconnected, its no big deal

**The upshot here is basically never connect anything to SPI\_CS.** Use whatever chip select pin you define in CircuitPython and just leave the **CS** pin 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 an overlay!

## Parts Used

OK now that we've gone thru the warning, lets 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)

\$14.95  
IN STOCK

ADD TO CART

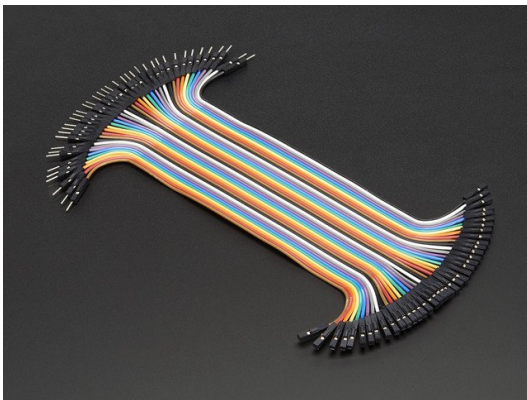


Thermocouple Type-K Glass Braid Insulated

\$9.95  
IN STOCK

ADD TO CART

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

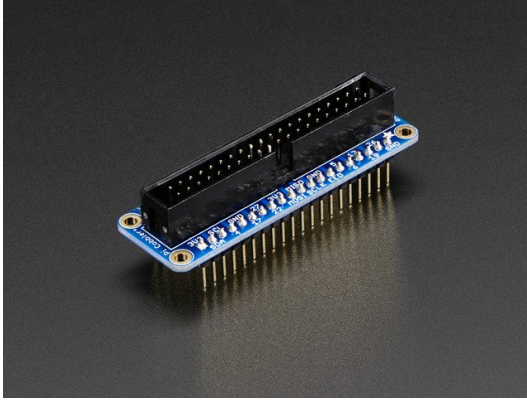


Premium Female/Male 'Extension' Jumper Wires - 40 x 6"  
(150mm)

\$3.95  
IN STOCK

ADD TO CART

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



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3

\$6.50  
OUT OF STOCK

OUT OF STOCK



Assembled Pi T-Cobbler Plus - GPIO Breakout

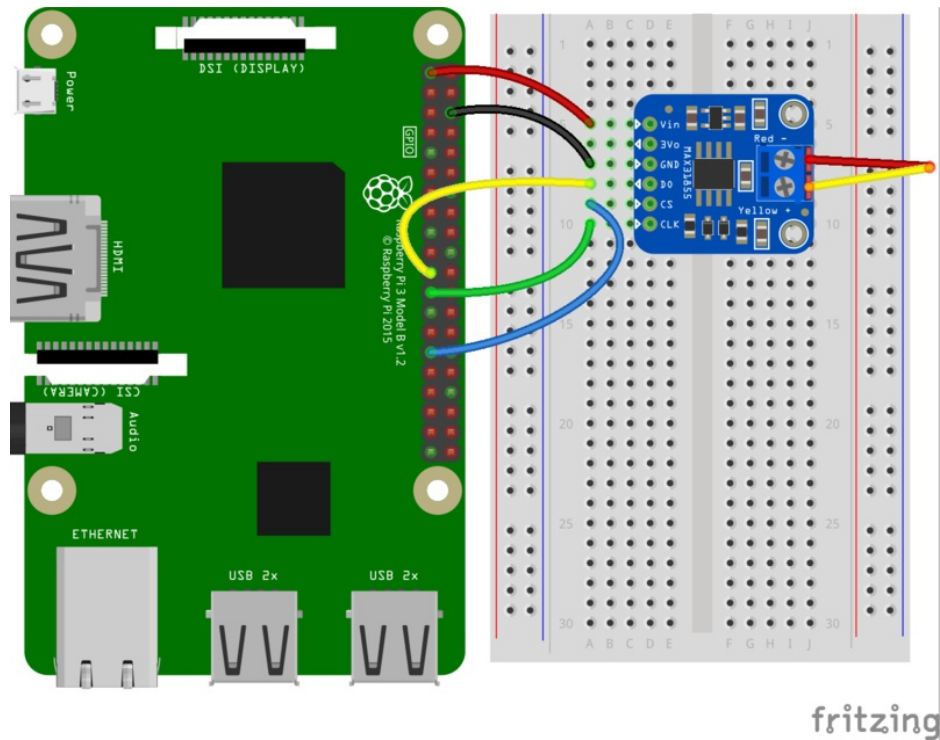
\$7.95  
IN STOCK

ADD TO CART

## Wiring

- Connect the Orange Pi **3.3V** power pin to **Vin**
- Connect the Orange Pi **GND** pin to **GND**
- Connect the Pi **SCLK** pin to the MAX31855 **CLK**
- Connect the Pi **MISO** pin to to the MAX31855 **DO**
- Connect the Pi **GPIO PA7** pin to to the MAX31855 **CS**

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

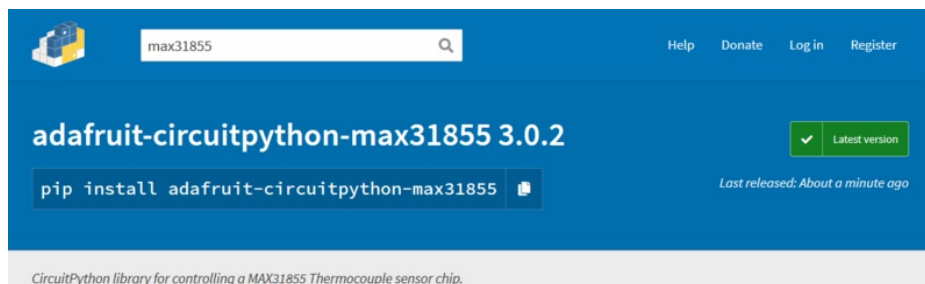


Double-check you have the right wires connected to the right location, it can be tough to keep track of Pi pins as there are forty of them!

## Install the CircuitPython MAX31855 Library

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

As of this writing, not *a*ll 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) (https://adafru.it/tB7)!) )

Once you know the name, install it with

```
pip3 install adafruit-circuitpython-max31855
```

```
COM24 - PuTTY
pi@devpi:~$ pip3 install adafruit-circuitpython-max31855
Collecting adafruit-circuitpython-max31855
  Downloading https://files.pythonhosted.org/packages/02/bc/2f306390163e25ef8f4413ff1a30dac50ebf9118413dbfa9d264b44a996a/adafruit-circuitpython-max31855-3.0.2.tar.gz
Collecting Adafruit-Blinka (from adafruit-circuitpython-max31855)
Collecting adafruit-circuitpython-busdevice (from adafruit-circuitpython-max31855)
  Cache entry deserialization failed, entry ignored
  Cache entry deserialization failed, entry ignored
  Downloading https://www.piwheels.org/simple/adafruit-circuitpython-busdevice/adafruit_circuitpython_busdevice-2.2.2-py3-none-any.whl
Collecting Adafruit-GPIO (from Adafruit-Blinka->adafruit-circuitpython-max31855)
  Using cached https://www.piwheels.org/simple/adafruit-gpio/Adafruit_GPIO-1.0.3-py3-none-any.whl
Collecting spidev (from Adafruit-GPIO->Adafruit-Blinka->adafruit-circuitpython-max31855)
  Using cached https://www.piwheels.org/simple/spidev/spidev-3.2-cp35-cp35m-linux_armv7l.whl
Collecting adafruit-pureio (from Adafruit-GPIO->Adafruit-Blinka->adafruit-circuitpython-max31855)
  Using cached https://www.piwheels.org/simple/adafruit-pureio/Adafruit_PureIO-0.2.3-py3-none-any.whl
Building wheels for collected packages: adafruit-circuitpython-max31855
  Running setup.py bdist_wheel for adafruit-circuitpython-max31855 ... done
  Stored in directory: /home/pi/.cache/pip/wheels/8b/93/a9/41c486048c873f75cf30f99669c86b5963ed8ef437e4904477
Successfully built adafruit-circuitpython-max31855
Installing collected packages: spidev, adafruit-pureio, Adafruit-GPIO, Adafruit-Blinka, adafruit-circuitpython-busdevice, adafruit-circuitpython-max31855
Successfully installed Adafruit-Blinka-0.1.8 Adafruit-GPIO-1.0.3 adafruit-circuitpython-busdevice-2.2.2 adafruit-circuitpython-max31855-3.0.2 adafruit-pureio-0.2.3 spidev-3.2
pi@devpi:~$
```

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) (<https://adafru.it/BKj>)

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.

Change the line that says

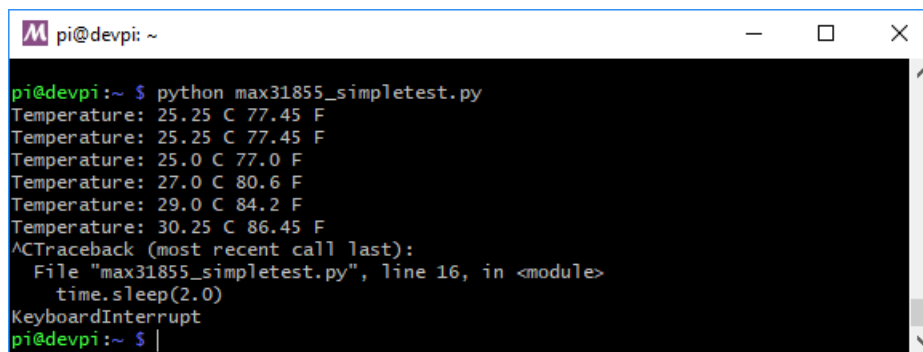
```
cs = digitalio.DigitalInOut(board.D5)
```

to

```
cs = digitalio.DigitalInOut(board.PA7)
```

Then in your command line run

```
python3 max31855_simpletest.py
```



```

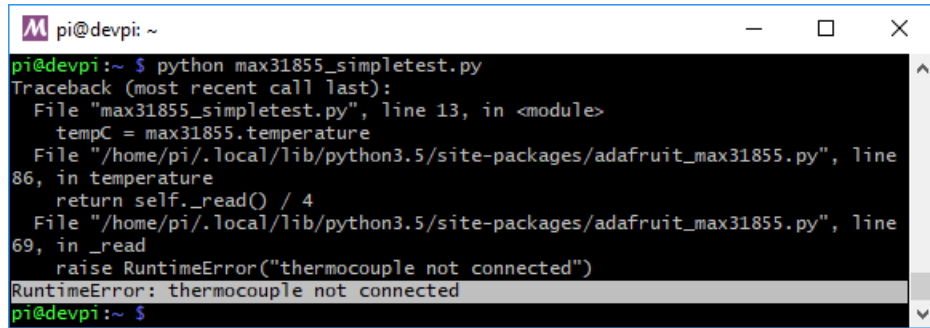
pi@devpi: ~
pi@devpi:~ $ python max31855_simpletest.py
Temperature: 25.25 C 77.45 F
Temperature: 25.25 C 77.45 F
Temperature: 25.0 C 77.0 F
Temperature: 27.0 C 80.6 F
Temperature: 29.0 C 84.2 F
Temperature: 30.25 C 86.45 F
^CTraceback (most recent call last):
  File "max31855_simpletest.py", line 16, in <module>
    time.sleep(2.0)
KeyboardInterrupt
pi@devpi:~ $

```

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!



A terminal window titled 'pi@devpi: ~' with standard window controls. It shows the execution of a Python script 'max31855\_simpletest.py'. A traceback is displayed, showing the error originates from the 'adafruit\_max31855.py' library file. The error is a 'RuntimeError: thermocouple not connected', which is highlighted in the terminal output.

```
pi@devpi:~ $ python max31855_simpletest.py
Traceback (most recent call last):
  File "max31855_simpletest.py", line 13, in <module>
    tempC = max31855.temperature
  File "/home/pi/.local/lib/python3.5/site-packages/adafruit_max31855.py", line
86, in temperature
    return self._read() / 4
  File "/home/pi/.local/lib/python3.5/site-packages/adafruit_max31855.py", line
69, in _read
    raise RuntimeError("thermocouple not connected")
RuntimeError: thermocouple not connected
pi@devpi:~ $
```

That's it! Now if you want to read the documentation on the library, what each function does in depth, visit our [readthedocs](https://circuitpython.readthedocs.io/projects/max31855/en/latest/) documentation at

<https://circuitpython.readthedocs.io/projects/max31855/en/latest/> (<https://adafru.it/BKk>)



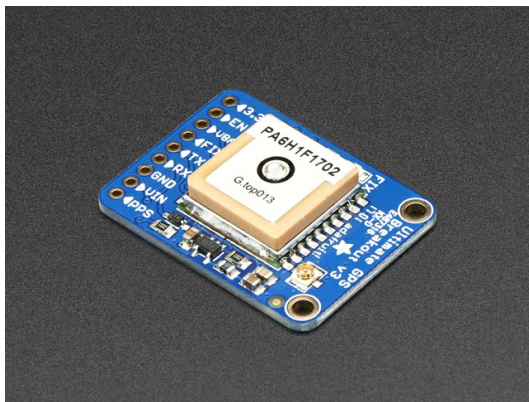
## UART / Serial

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 Orange Pi. 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

\$39.95  
IN STOCK

ADD TO CART

---

## 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

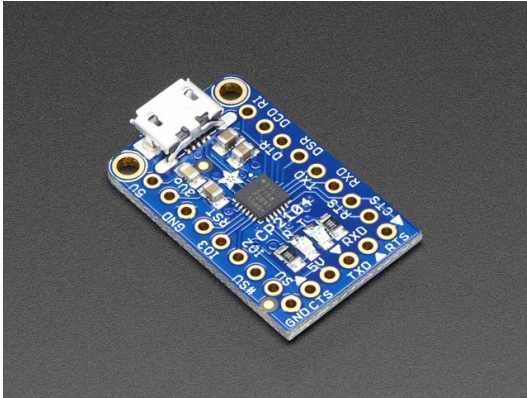
\$9.95  
IN STOCK

ADD TO CART

---

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

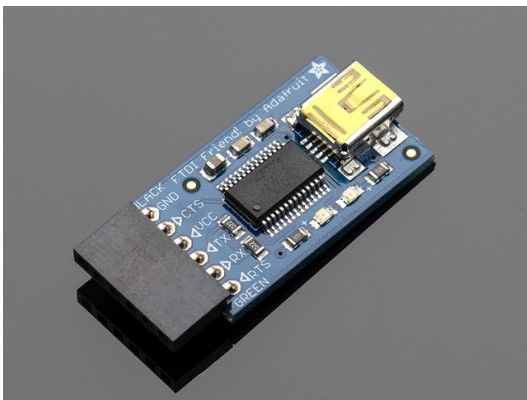


Adafruit CP2104 Friend - USB to Serial Converter

\$5.95  
IN STOCK

ADD TO CART

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

\$14.75  
IN STOCK

ADD TO CART



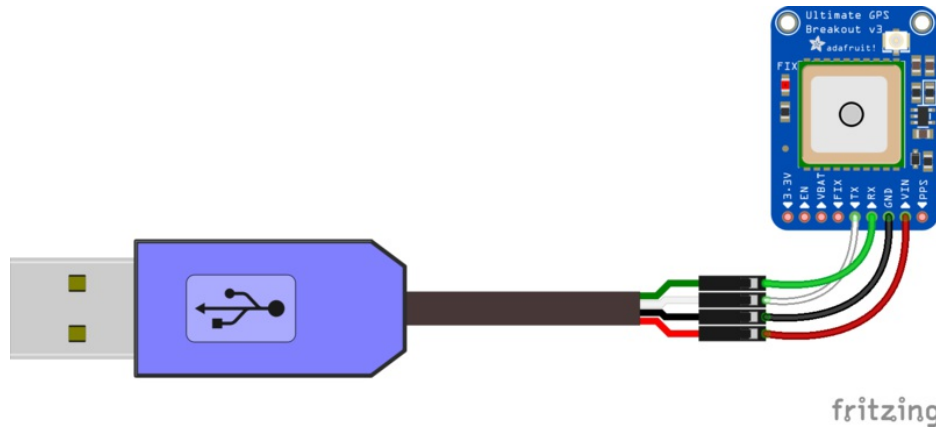
FTDI Serial TTL-232 USB Cable

\$17.95  
IN STOCK

ADD TO CART

You can wire up the GPS by connecting the following

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



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

```
pi@devpi:~$ dmesg | tail -10
[ 601.391424] usb 1-1.2: Manufacturer: FTDI
[ 601.391432] usb 1-1.2: SerialNumber: AL00FP25
[ 601.440489] usbcore: registered new interface driver usbserial
[ 601.440554] usbcore: registered new interface driver usbserial_generic
[ 601.440609] usbserial: USB Serial support registered for generic
[ 601.455895] usbcore: registered new interface driver ftdi_sio
[ 601.455970] usbserial: USB Serial support registered for FTDI USB Serial Device
[ 601.456248] ftdi_sio 1-1.2:1.0: FTDI USB Serial Device converter detected
[ 601.456383] usb 1-1.2: Detected FT232RL
[ 601.459259] usb 1-1.2: FTDI USB Serial Device converter now attached to ttyUSB0
```

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

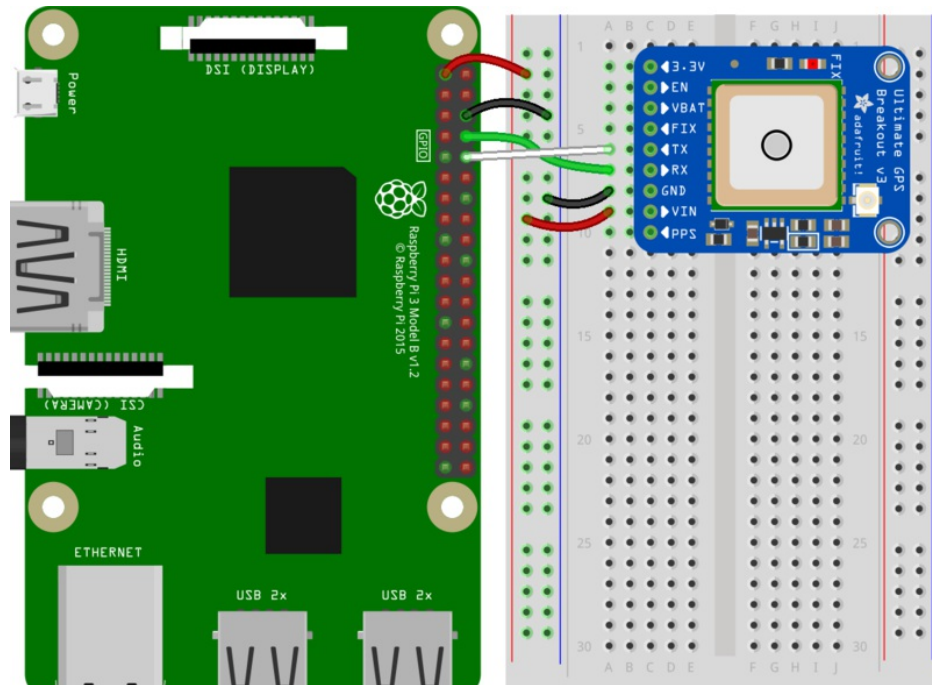
## The Hard Way - Using Built-in UART

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. Unlike the Raspberry Pi, the Orange Pi isn't using the RX/TX pins for a console, those are on a different UART peripheral, so as long as you've activated UART3 (see the Install page) you should be good to go!

You can use the built in UART via `/dev/ttyS3`

Wire the GPS as follows:

There's no Orange Pi Fritzing object, but the Raspberry Pi has the same pinout so we're using that instead

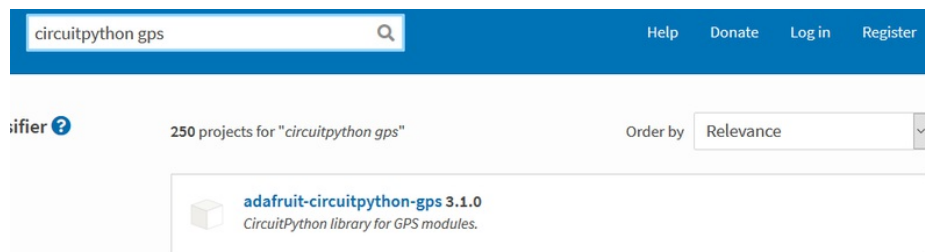


fritzing

## 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)(<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 *dependency* called **pyserial**. This is a great thing about pip, if you have other required libraries they'll get installed too!

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

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

## Run that code!

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

Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be

[https://github.com/adafruit/Adafruit\\_CircuitPython\\_GPS/tree/master/examples](https://github.com/adafruit/Adafruit_CircuitPython_GPS/tree/master/examples) (<https://adafru.it/Ca9>)

Lets start with the simplest, the echo example

```
# 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=3000)

# 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'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# 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')
# 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')
# Tuen on everything (not all of it is parsed!)
#gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# 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 = uart.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 Orange Pi PC, the device name is `/dev/ttyS3`

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 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@orangepipc:/home/pi# python3 gpstest.py
$PMTK001,314,3*36
$PMTK001,220,3*30
$GPGGA,000013.800,,,,,0,00,,M,M,*72
$GPRMC,000013.800,V,,,,,0.00,0.00,060180,,,N*48
$GPGGA,000014.799,,,,,0,00,,M,M,*7A
$GPRMC,000014.799,V,,,,,0.00,0.00,060180,,,N*40
$GPGGA,000015.799,,,,,0,00,,M,M,*7B
$GPRMC,000015.799,V,,,,,0.00,0.00,060180,,,N*41
$GPGGA,000016.799,,,,,0,00,,M,M,*78
$GPRMC,000016.799,V,,,,,0.00,0.00,060180,,,N*42
$GPGGA,000017.799,,,,,0,00,,M,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 \(https://adafru.it/BJX\)](#) to share your feedback and perhaps even submit some improvements!



## 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!