CircuitPython Libraries on any Computer with FT232H

Created by Carter Nelson

Last updated on 2019-12-02 05:22:28 PM UTC
Overview

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

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

CircuitPython and CircuitPython Libraries

As you are going through this guide, keep in mind the difference between CircuitPython and CircuitPython Libraries:

- **CircuitPython** - a microprocessor specific implementation of Python written in C. [Here’s the source code](https://adafru.it/tB7). And here’s the main [CircuitPython guide](https://adafru.it/cpy-welcome).
- **CircuitPython Libraries** - sensor and breakout specific code written in Python using the CircuitPython hardware API. There are a lot of these - [check out the bundle](https://adafru.it/zdx).

There are various hardware combinations that allow for running CircuitPython and CircuitPython Libraries. In this guide we will **not** be using the actual CircuitPython firmware. But we will be using CircuitPython Libraries. To explain this further, let’s go over the main hardware platforms and explain where the FT232H fits in to all this.

CircuitPython on Microcontrollers

This is the most straightforward setup.

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

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

CircuitPython on Single Board Computers

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

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

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

CircuitPython on Personal Computers

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

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

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

This will allow us to do something like this:
With FT232H support added to Blinka, we can follow a similar approach as with the SBCs. The FT232H attached to the USB port acts as our surrogate set of low level hardware interfaces.

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

The support for the FT232H in Blinka utilizes the pyftdi library (https://adafruit.it/FVF) by eblot (https://adafruit.it/FVG). This in turn relies on a few other things, like libusb. So before we can actually use the FT232H, we need to get everything setup. See the OS specific sections for what we went through to get things working for each.

Additional Information

Just for reference, here's the install info from the pyftdi library itself:

https://adafruit.it/FVH
https://adafruit.it/FVH

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

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

Have Python 3 Installed

We assume you already have Python 3 installed on your computer. **Note we do not support Python 2** - it's deprecated and no longer supported!

At your command line prompt of choice, check your Python version with `python --version`

![Command Prompt](image)

Plug in FT232H & Fix Driver with Zadig

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

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

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

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

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

Now run the Zadig tool executable you just downloaded (there is no installation necessary, the executable is the program). Click the **Options** menu and select the **List All Devices** item below:
The list box of devices should populate with many devices. Select the USB Serial Converter device shown below.

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

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

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

To check that the driver was successfully replaced, open Device Manager from Control Panel or searching in the Start menu. You should see a new top level node **libusbK USB Devices** and the **USB Serial Converter** underneath it as shown below.
If you see the libusbK node and USB serial device, move on to the next step to install libftdi.

If you don't see the libusbK node, try unplugging and plugging back in the FT232H breakout. If you still don't see the node, run Zadig tool again and follow the steps above again to make sure you replace the FTDI driver for the device with the libusbK driver.

Install pyftdi and (fixed) pyusb

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

`pip3 uninstall pyusb`

`pip3 uninstall pyftdi`

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

To make sure you do not have pyusb and pyftdi installed
Run them again to make absolutely sure!

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

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

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

Then

```
cd pyusb
python setup.py install
```

if you get a permission denied error, try

```
python setup.py install --user
```

Now you can run

```
pip install pyftdi
```

Test pyusb and pyftdi

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

```
import usb
import usb.util
```

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

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

Install Adafruit Blinka

Run `pip install adafruit-blinka`

Set Environment Variable

You must do this every time before running circuitpython code, you can set it permanently in windows if you like, for now just type into the same cmd window you're using with Python

`set BLINKA_FT232H=1`

Check Platform was detected

In the same command window you `set BLINKA_FT232H=1` env var, run `python` and run

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

at the Python REPL. If you get no errors, and you see a list of all the pins available - you're good to go!
Mac OSX

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

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

Install libusb

Start by installing libusb with

```
brew install libusb
```

Install pyftdi and Blinka

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

Test!

Finally, set the environment variable with

```bash
export BLINKA_FT232H=1
```

and run python3. In the REPL, type

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

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

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

Install libusb

Run the following:

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

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

Setup udev rules

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

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

Here we use nano, so run:

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

like this:

![Terminal window showing nano command](image)

add the contents from above:

![Terminal window showing nano command](image)

and then press `CTRL-X` and `Y` to save and exit.
The settings will take effect the next time you plug in the FT232H.

Install pyftdi

To install pyftdi and its dependencies, run:

```bash
pip3 install pyftdi
```

Install Blinka

To install Blinka and its dependencies, run:

```bash
pip3 install adafruit-blinka
```
Set environment variable

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

```
export BLINKA_FT232H=1
```

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

Run the sanity check.

Now move on to the Sanity Check section and run the commands there to make sure everything is installed correctly.
Post Install Checks

After going through all the install steps for your OS, run this as a simple test to make sure everything is installed correctly. See the rest of the page for some potential hiccups you may run into.

Go ahead and plug in your FT232H to a USB port on your PC.

Check pyftdi is installed correctly

Launch Python:

```
python3
```

and enter these commands:

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

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

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

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

If you get

Go back to the setup steps and Zadig your board if on Windows
Check environment variable within python

At the Python REPL, type

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

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

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

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

Now type in `import board`

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

Board not supported None

If you get

```
NotImplementedError: Board not supported None
```

that could mean you did not set the FT232H environmental variable or you don't have the latest python libraries or the FT232H is not plugged in.
BLINKA_FT232H environment variables set, but no FT232H device found

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

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

The device has no langid

On Windows, if you get `ValueError: The device has no langid` on an open_from_url() call that may mean you don't have libusbk.dll placed in C:\Windows\System32.

NotImplementedError: Operation not supported or unimplemented on this platform

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

```
>>> from pyftdi.ftdi import ftdi
>>> ftdi.open_from_url('ftdi:///f')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 399, in open_from_url
    vendor, product, index, serial, interface = self.get_identifiers(url)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 221, in get_identifiers
    class Enum, cls.VENDOR_ID, cls.PRODUCT_ID, cls.DEVICE_VENDER
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 302, in parse_url
    devices = devices_class.find_all(vpi)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 392, in find_all
    return UsbTools.find_all(vpe, nocache)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 60, in find_all
    return UsbTools.find_devices(vpe, nocache)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 271, in find_devices
    return UsbTools.get_string(dev, dev.IS_SERIAL_NUMBER)
  File "C:\Python36\lib\site-packages\pyftdi\ftdi.py", line 500, in get_string
    return usb.util.get_string(device, device_name)
ValueError: The device has no langid
```

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

The easiest way we found to do that is run

`brew remove libftdi`

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

FT232H Blinka

Power Pins
- **5V** - this is the 5V power from the USB input.
- **GND** - this is the common ground for all power and logic.

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

I2C Pins
- **SCL** - the I2C clock signal is on **D0**.
- **SDA** - the I2C data is on **D1+D2**. Note that there are two pins which must be tied together and treated as one.

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

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

All right, now that all that annoying install stuff is done, let's have some fun.

The following sections will provide some basic examples for the main use cases - GPIO, I2C, and SPI.

| Make sure you've set the BLINKA_FT232H environment variable. |

Installing Libraries for Breakouts

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

Digital Output

Let's blink a LED!

Here's the bread board layout. The resistor can be something around 1kOhm. We don't need to make the LED super bright.

First, let's do things interactively so you can see how it all works one line at a time. Start by launching Python:

```python
python3
```

Then, at the Python `>>>` prompt, enter the following to import the needed modules:

```python
import board
import digitalio
```

Next we'll create our LED digital pin and set the mode to output:

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

And that should be it. You should be able to turn ON the LED with:

```python
led.value = True
```

And turn it OFF with:

```python
led.value = False
```
And here's a complete blink program you can run to make the LED blink forever.

```python
import time
import board
import digitalio

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

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

Save it as something like `blink.py` and then you can run it with:

```bash
python3 blink.py
```

Digital Input

Let's read a button!

Here's the bread board layout. Use something like a 10kOhm resistor.
We'll do this interactively also. So launch python:

```
python3
```

Then, at the Python >>> prompt, enter the following to import the needed modules:

```
import board
import digitalio
```

And now we create our button digital pin and set it to input.

```
button = digitalio.DigitalInOut(board.C0)
button.direction = digitalio.Direction.INPUT
```

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

```
button.value
```

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

**Digital Input and Output**

Ok, let's put those two together and make the button turn on the LED. So we'll use two digital pins - one will be an input (button) and one will be an output (LED).

Here's the bread board layout.
And here’s the code.

```python
import board
import digitalio

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

button = digitalio.DigitalInOut(board.C1)
button.direction = digitalio.Direction.INPUT

while True:
    led.value = button.value
```

Save that to a file with a name like `button_and_led.py` and then you can run it with:

```bash
python3 button_and_led.py
```

and the button should turn on the LED when pressed.
I2C

Let's talk to an I2C sensor.

We'll use the BME280 sensor (https://adafruit.it/y8f) which can read temperature, humidity, and pressure. Here's the wiring diagram:

Don't forget to jumper the D1 and D2 pins together.

Install BME280 Library

To install the BME280 library, run the following:

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

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

Run Example

Now we can run the simple test example from the library. Here's the code:
import time
import board
import busio
import adafruit_bme280

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

# OR create library object using our Bus SPI port
#spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
#bme_cs = digitalio.DigitalInOut(board.D10)
#bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, bme_cs)

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

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

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

```bash
python3 bme280_simpletest.py
```

and you should see it print out sensor readings over and over:
Let's talk to a SPI sensor.

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

Install BME280 Library

To install the BME280 library, run the following:

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

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

Run Example

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

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

Here's the complete code:
import time
import board
import busio
import digitalio
import adafruit_bme280

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

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

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

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

Copy and save that code and then run it with:

    python3 bme280_simpletest.py

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

NeoPixels (https://adafruit.it/Gbx)
TFT Display

Let's draw to a display!

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

Connect the Display

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

Install Pillow/PIL

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

https://adafru.it/FUR

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

Install CircuitPython RGB Display Library

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

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

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

https://adafruit.it/FUU

The Code

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

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

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

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

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

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

```
python3 tft_image.py
```

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

There's a lot of power in Pillow/PIL. It can do more than just load and display images. Here's another example that creates a Draw object, uses a TTF font, and then grabs some stats from the PC and displays them.
# Quick test of TFT FeatherWing (ILI9341) with Feather M0 or M4
# Will fill the TFT black and put a red pixel in the center, wait 2 seconds,
# then fill the screen blue (with no pixel), wait 2 seconds, and repeat.
import time
import random
import busio
import digitalio
import board
from adafruit_rgb_display.rgb import color565
import adafruit_rgb_display.ili9341 as ili9341

# Configuration for CS and DC pins (these are FeatherWing defaults on M0/M4):
cs_pin = digitalio.DigitalInOut(board.D9)
dc_pin = digitalio.DigitalInOut(board.D10)

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

# Setup SPI bus using hardware SPI:
spi = busio.SPI(clock=board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Create the ILI9341 display:
display = ili9341.ILI9341(spi, cs=cs_pin, dc=dc_pin, baudrate=BAUDRATE)

# Main loop:
while True:
    # Fill the screen red, green, blue, then black:
    for color in ((255, 0, 0), (0, 255, 0), (0, 0, 255)):
        display.fill(color565(color))
    # Clear the display
    display.fill(0)
    # Draw a red pixel in the center.
    display.pixel(display.width//2, display.height//2, color565(255, 0, 0))
    # Pause 2 seconds.
    time.sleep(2)
    # Clear the screen a random color
    display.fill(color565(random.randint(0, 255),
                        random.randint(0, 255),
                        random.randint(0, 255)))
    # Pause 2 seconds.
    time.sleep(2)

The example uses a ST7789 based TFT, so if you are using the same 2.4" TFT (ILI9341) from above, you'll need to change those lines of code to match. Use the image load example above for reference. You'll need to change the display import to:

```python
import adafruit_rgb_display.ili9341 as ili9341
```

and the display setup to:
# Configuration for CS and DC pins (these are FeatherWing defaults on M0/M4):
cs_pin = digitalio.DigitalInOut(board.C0)
dc_pin = digitalio.DigitalInOut(board.C1)
reset_pin = None

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 64000000

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

# Create the ST7789 display:
disp = ili9341.ILI9341(spi, cs=cs_pin, dc=dc_pin, rst=reset_pin, baudrate=BAUDRATE,
width=240, height=320)

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

And it will update in near real-time.
FAQ & Troubleshooting

There's a few oddities when running Blinka/CircuitPython on linux. Here's a list of stuff to watch for that we know of!

### Mixed SPI mode devices

Due to the way we share an SPI peripheral, you cannot have two SPI devices with different 'mode/polarity' on the same SPI bus - you'll get weird data

95% of SPI devices are mode 0, check the driver to see mode or polarity settings. For example:

- **LSM9DS1 is mode 1**, please use in I2C mode instead of SPI
- **MAX31865 is phase 1**, try using this on a separate SPI device, or read data twice.
Why am I getting AttributeError: 'SpiDev' object has no attribute 'writebytes2'?

This is due to having an older version of spidev. You need at least version 3.4. This should have been taken care of when you installed Blinka, but in some cases it does not seem to happen.

To check what version of spidev Python is using:

```
$ python3
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import spidev
>>> spidev.__version__
'3.4'
```

If you see a version lower 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

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!
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
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
- No SPI support without reflashing the board
- 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://adafruit.github.io/FWD](https://adafruit.github.io/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)