

What. Not how.

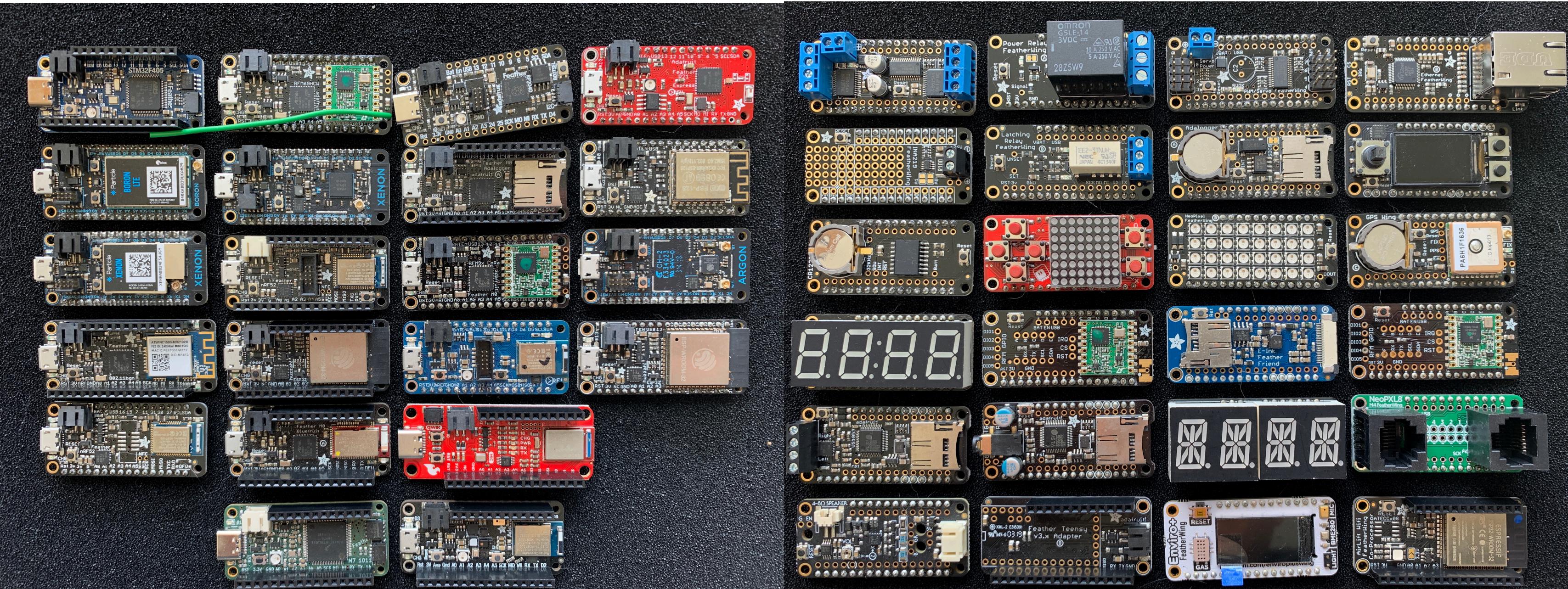
Scott (@tannewt)

An *interface* is a boundary between
two things

A well designed interface defines the *what* without revealing the *how*.

Abstraction lets one ignore the "how" of *what* the interface does. Abstract interfaces enable multiple uses of the same interface which increases compatibility between things. Open interfaces allow others to work against the same abstractions and collaborate on different things.

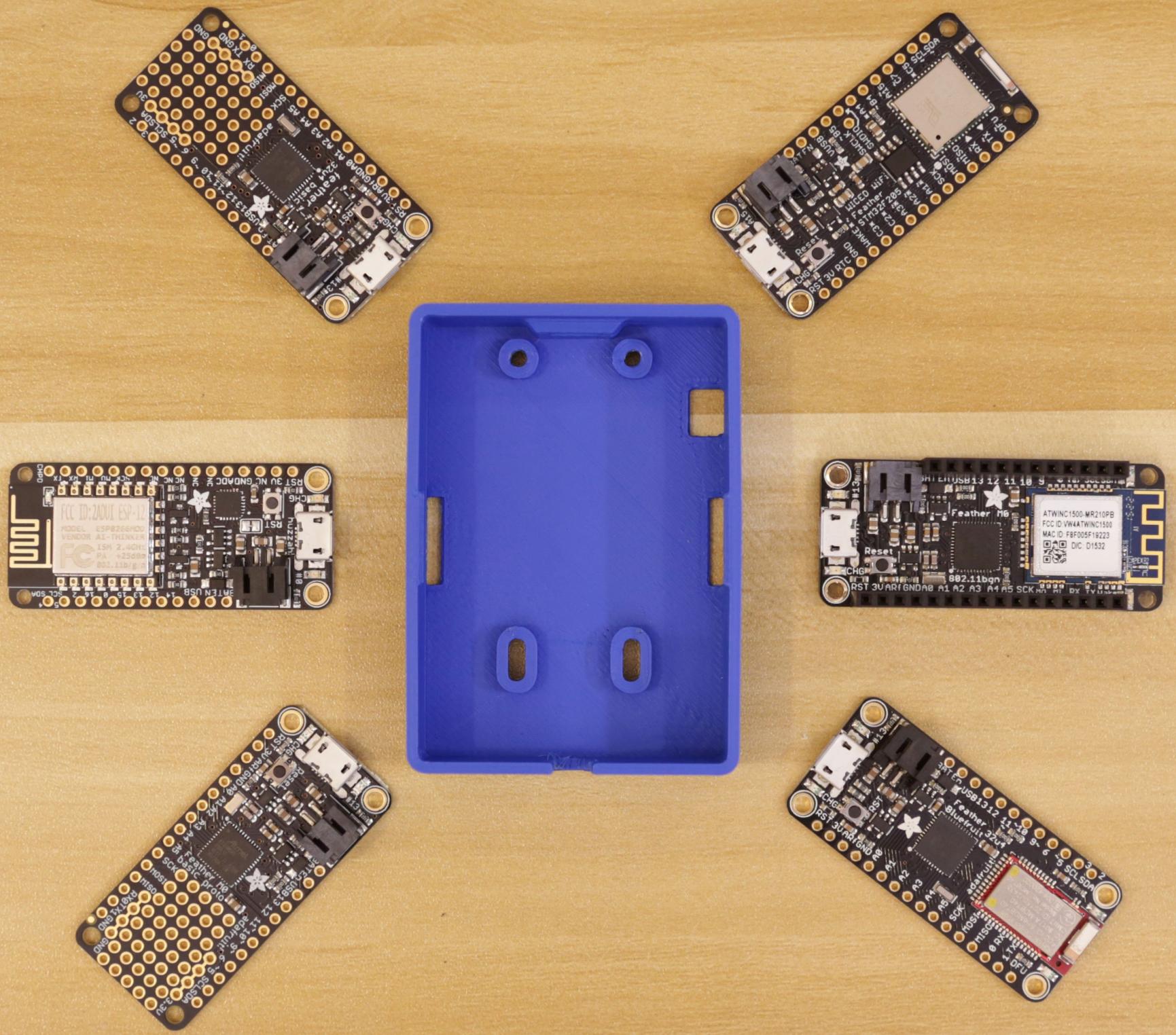
Abstraction of the *how* into *what*
allows for reuse.



<https://github.com/adafruit/awesome-feather/>

In open hardware there are three common category of interfaces:

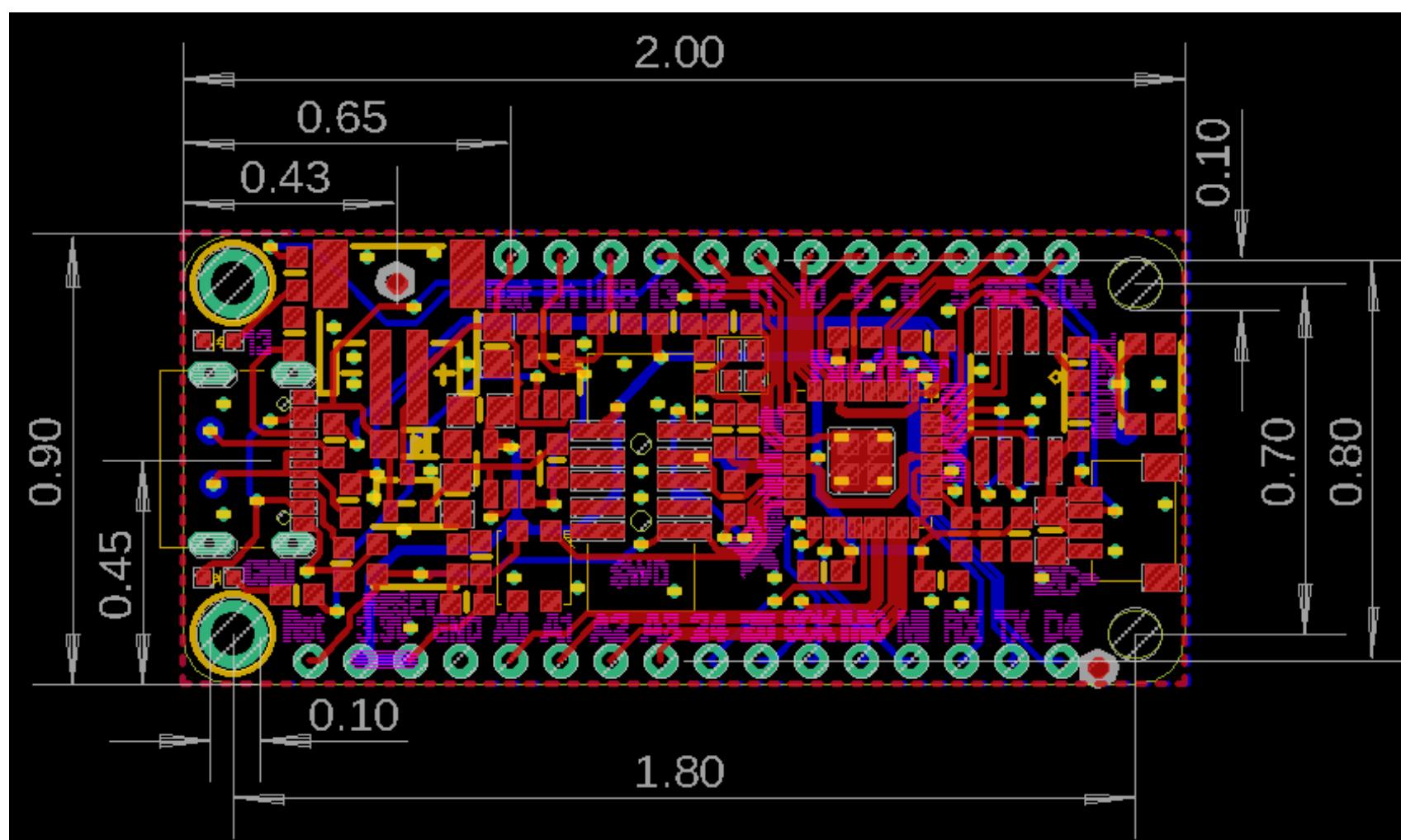
- Mechanical
- Electrical
- Software

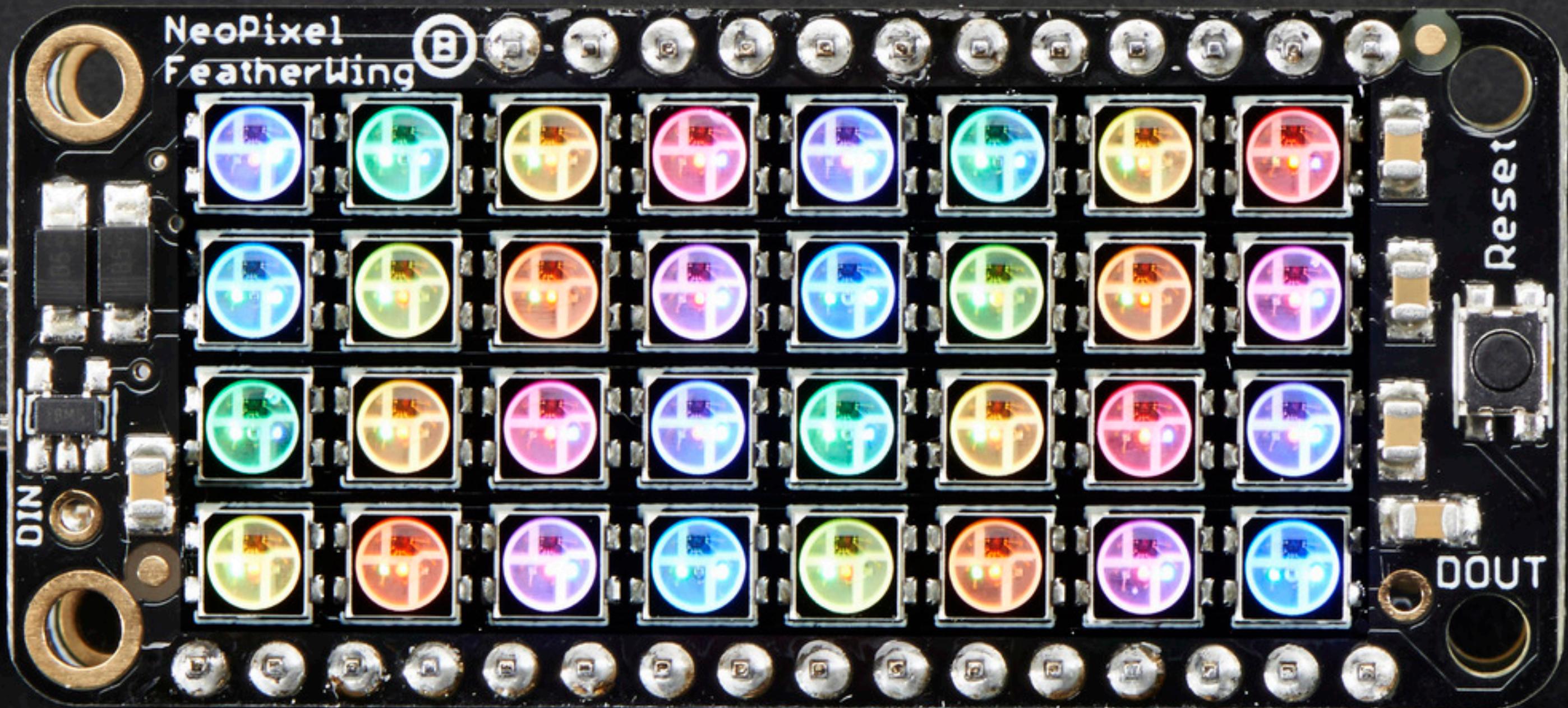


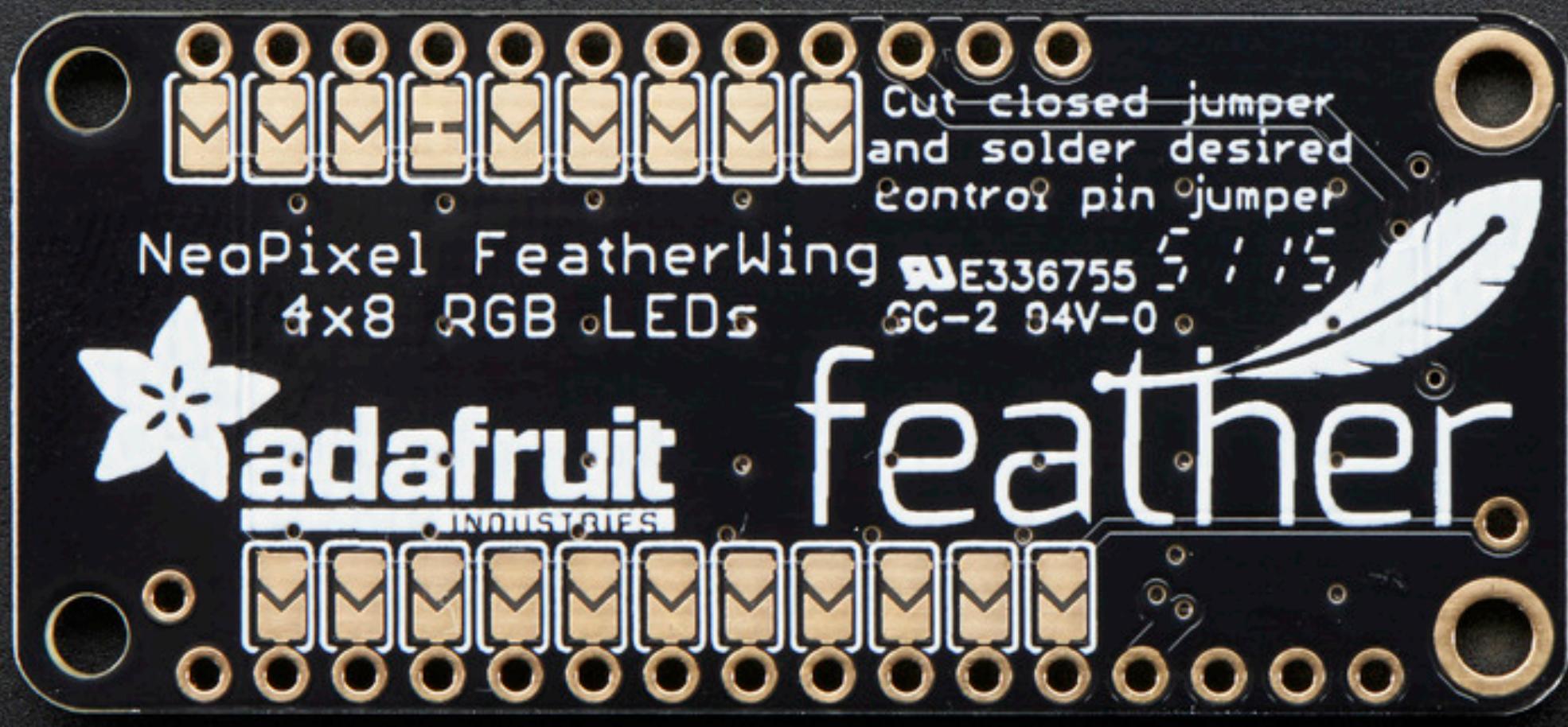
Outside In mechanical design may be designing for an existing case. In this case, the print was designed inside out. The second hole is a slot to accomodate different feather lengths. This is a sign of flexibility in the interface that if unbounded could lead to incompatibility and break the abstraction provide by the Feather interface.

Mechanical

- Board size
- Mounting holes
- Component size
- Electrical connection location







Electrical

- Depends on *mechanical* interfaces through contact position
- Designated function(s)/protocol(s)
- Voltage levels
- Input/output

Bat En USB 13 12 11 10 9 6 5 SCL SDA

Rst

3.3V

GND

A0

A1

A2

A3

24

25

SCK

M0

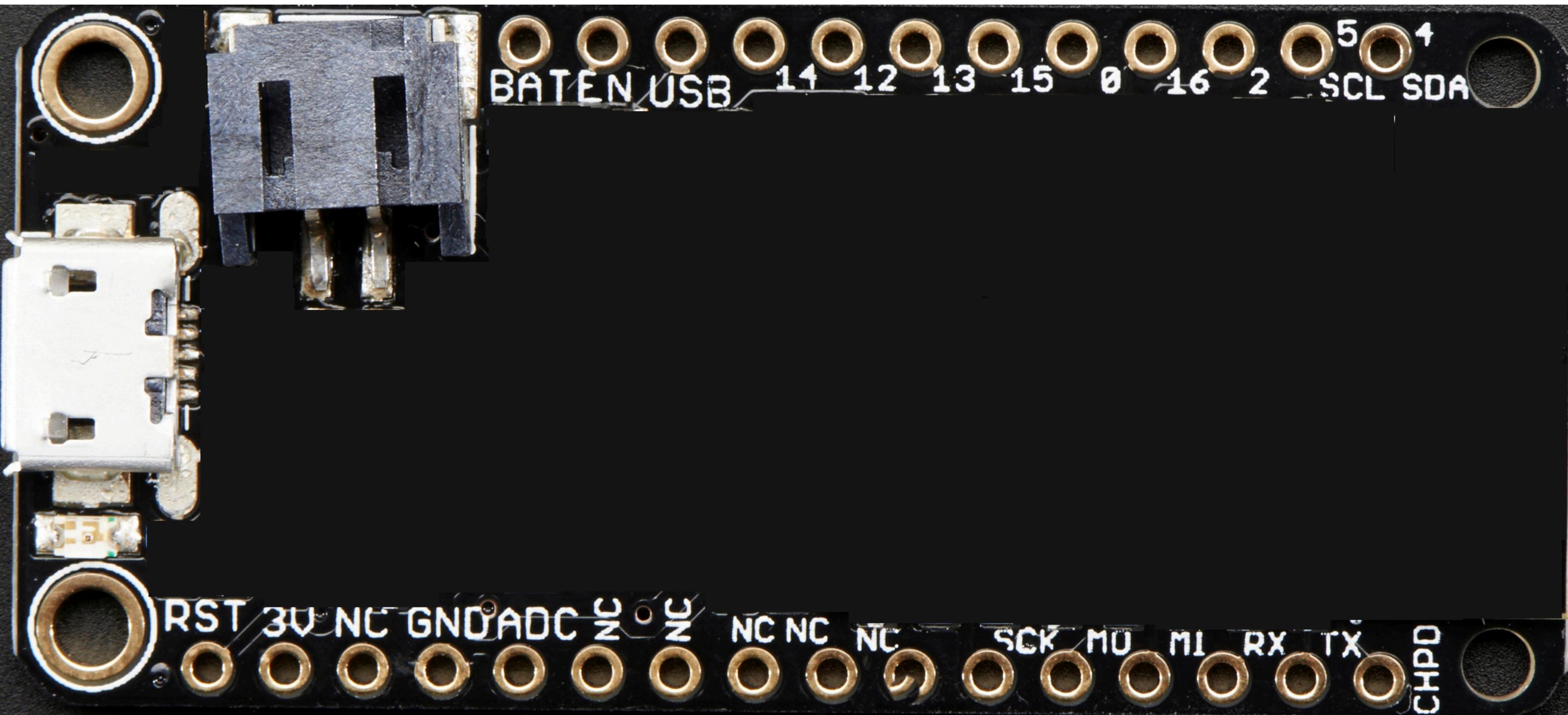
M1

RX

TX

D4

I²C





```
23
24 # If you are using a board with pre-defined ESP32 Pins:
25 esp32_cs = DigitalInOut(board.ESP_CS)
26 esp32_ready = DigitalInOut(board.ESP_BUSY)
27 esp32_reset = DigitalInOut(board.ESP_RESET)
28
29 # If you have an AirLift Shield:
30 # esp32_cs = DigitalInOut(board.D10)
31 # esp32_ready = DigitalInOut(board.D7)
32 # esp32_reset = DigitalInOut(board.D5)
33
34 # If you have an Airlift Featherwing or ItsyBitsy Airlift:
35 # esp32_cs = DigitalInOut(board.D13)
36 # esp32_ready = DigitalInOut(board.D11)
37 # esp32_reset = DigitalInOut(board.D12)
38
39 # If you have an externally connected ESP32:
40 # NOTE: You may need to change the pins to reflect your wiring
41 # esp32_cs = DigitalInOut(board.D9)
42 # esp32_ready = DigitalInOut(board.D10)
43 # esp32_reset = DigitalInOut(board.D5)
44
45 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
46 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,
47
48 requests.set_socket(socket, esp)
49
50 if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
51     print("ESP32 found and in idle mode")
52     print("Firmware vers.", esp.firmware_version)
53     print("MAC addr:", [hex(i) for i in esp.MAC_address])
54
```

Remember that pin naming impacts software! Software impacts libraries, examples and tutorial content.

Software

- Names!
- Interaction
- Default software
- Bootloader

A screenshot of a web browser window displaying a list of Adafruit CircuitPython libraries. The browser interface includes standard controls like back, forward, and search at the top, and a URL bar showing https://github.com/adafruit/CircuitPython_Bundle/. The main content area contains text and a bulleted list of library links.

Here is a listing of current Adafruit CircuitPython Libraries.
There are 269 libraries available.

Drivers:

- [Adafruit CircuitPython 74HC595 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython ADS1x15 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython ADT7410 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython ADXL34x \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AHTx0 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AM2320 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AMG88xx \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython APDS9960 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AS726x \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AS7341 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython ATECC \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython AW9523 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BD3491FS \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BH1750 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BME280 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BME680 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BMP280 \(PyPi\) \(Docs\)](#)
- [Adafruit CircuitPython BMP3XX \(PyPi\) \(Docs\)](#)

S ★ Adafruit Learning System

Shop Learn Blog Forums LIVE! AdaBox IO Sign In 0

adafruit Explore & Learn New Guides

All Categories > CircuitPython

LED Snowboard with Motion-Reactive Animation By Erin St Blaine

Intermediate

Zelda Guardian Robot Terrako Companion By Ruiz Brothers

Intermediate

Quickstart - Raspberry Pi RP2040 with BLE and... By Brent Rubell

Beginner

Black Lives Matter Education & Workshop Kit By lady ada

Beginner

Touch Deck: DIY Customizable TFT Control Pad By John Park

Beginner

Adafruit MagTag COVID Vaccination Percent Tracker By Dylan Herrada

Beginner

Your Very First Circuit Sculpture By Jeff Epler

Beginner

Retroreflective Greenscreen Light Ring QT Py... By John Park

Beginner

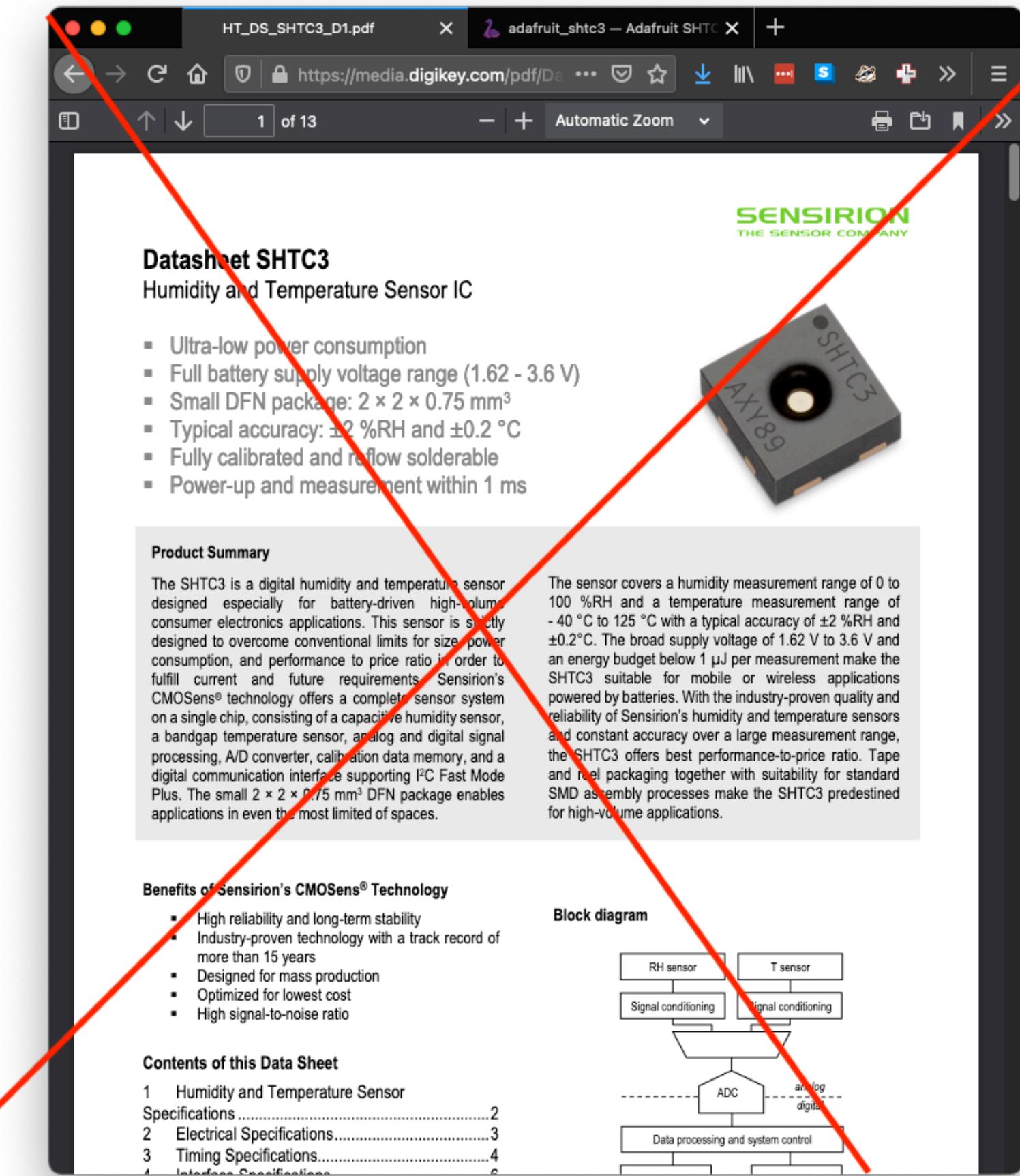
Breakout Board and TFT By John Park

Adafruit_CircuitPython_Display_Text Comprehensive usage guide with examples

Introduction to Adafruit Modules

CircuitPython Breakout Board

Think Outside In



The feather pin numbering was different because the inside numbers were different than those of the other feathers. In particular, the Espressif Arduino core doesn't do number mapping to allow consistent numbers across boards. It assumes inside out numbering. Vendors tend to do this a bunch (Pico does as well.) It's better to abstract the numbering from the get-go so that new boards in the same form factor don't need to break the inside out numbering.

The screenshot shows a GitHub repository page for the Adafruit_SHTC3 library. The specific file displayed is `shtc3_simpletest.py`. The page includes a header with the repository name and a pull request from user `siddacious` fixing missing copyright and license. Below this, it shows one contributor and a code editor area.

Raw **Blame**

Executable File | 17 lines (15 sloc) | 446 Bytes

```
1 # SPDX-FileCopyrightText: Copyright (c) 2020 Bryan Siepert for Adafruit Industries, LLC
2 #
3 # SPDX-License-Identifier: MIT
4 import time
5 import busio
6 import board
7 import adafruit_shtc3
8
9 i2c = busio.I2C(board.SCL, board.SDA)
10 sht = adafruit_shtc3.SHTC3(i2c)
11
12 while True:
13     temperature, relative_humidity = sht.measurements
14     print("Temperature: %0.1f C" % temperature)
15     print("Humidity: %0.1f %%" % relative_humidity)
16     print("")
17     time.sleep(1)
```

HT_DS_SHTC3_D1.pdf

adafruit_shtc3 – Adafruit SHTC3

Adafruit's Register library: https://github.com/adafruit/Adafruit_CircuitPython_Register

class adafruit_shtc3.SHTC3(i2c_bus)

A driver for the SHTC3 temperature and humidity sensor.

Parameters: `i2c_bus (I2C)` – The `busio.I2C` object to use. This is the only required parameter.

low_power

Enables the less accurate low power mode, trading accuracy for power consumption

measurements

both `temperature` and `relative_humidity`, read simultaneously

relative_humidity

The current relative humidity in % rH

reset()

Perform a soft reset of the sensor, resetting all settings to their power-on defaults

sleeping

Determines the sleep state of the sensor

temperature

The current temperature in degrees celcius

[Previous](#)

© Copyright 2020 Bryan Siepert Revision f3fc3048.
Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

Be Explicit

The screenshot shows a web browser window with the URL <https://learn.adafruit.com/>. The page is titled "Feather Specification | Introduct X". The main content is divided into sections:

- Bus Pins**:
 - To make it easy to create add-ons, we have fixed inter-chip bus pins:
 - **RX & TX** - these are UART pins. If your mainboard has a spare hardware UART, put these here. If your board uses the sole UART for bootloader/debug (e.g. ESP8266) then you can put those here as well. If designing a Feather, try to not use these as they are not always available
 - **SDA & SCL** - this should be your main I2C bus. I2C pullups are put on the Wings, not on the mainboard. These are 3.3V logic, we strongly recommend avoiding clock-stretch I2C on Featherwings. Repeated start is OK! This is our recommended interface for sensors and Wings, try to stick to it.
 - **SCK/MOSI/MISO** - this should be your main SPI bus. 3.3V logic. If you share these with devices on the main featherwing, keep the CS pin from being exposed so that you don't have SPI bus contentions.
 - **I2S** - we don't fix the I2S pins! I2S is not as common and often are very restricted. We don't have any guarantees on where the I2S pins may lay.
- Analog Pins**:
 - There are 6 reserved spots for Analog pins between the power pins and SPI pins. You do not have to have the analog pins in order (e.g. A0 first, then A1) but it is considered in good taste to do so. We do our best to put any DAC pins on the first two pins (e.g. Feather M0 has the DAC on A0, ESP32 has the DACs on A0 and A1)
 - The ESP8266 only has one analog pin and its 1.0V max**, so be aware that while every other Feather mainboard has 6 analog-ins, if you want your wing to work with the ESP8266 there are restrictions. We tend not to use the analog inputs on Wings for these reasons.
 - While we don't require all 6 pins to be analog inputs, it would be unusual if they were

When designing a new product or interface be explicit with requirements. Even when allowing for flexibility, set explicit bounds. Any omitted detail may lead to incompatibilities in the future.

Be Strict

The screenshot shows a web browser displaying the Adafruit CircuitPython documentation for the `time` module. The page title is "Adafruit CircuitPython" and the specific page is "Docs » Core Modules » time – time and timing related functions". The main content is titled "time – time and timing related functions". It states that the `time` module is a strict subset of the CPython `time` module. Below this, there are three code snippets with detailed descriptions:

- `time.monotonic() → float`**
Returns an always increasing value of time with an unknown reference point. Only use it to compare against other values from `monotonic`.
Returns: the current monotonic time
Return type: float
- `time.sleep(seconds: float) → None`**
Sleep for a given number of seconds.
Parameters: seconds (float) – the time to sleep in fractional seconds
- `class time.struct_time(time_tuple: Tuple[int, int, int, int, int, int, int, int, int])`**
Structure used to capture a date and time. Note that it takes a tuple!
Parameters: time_tuple (tuple) –

In CircuitPython we reimplement some desktop "CPython" APIs by being strict subsets. This allows for code written in CP to be reused in CPython. Do this for hardware too. Keep the same shape and silkscreen names as another board even if it isn't part of a formal interface. This can come up when a v2 version of a board is made.

Remember

Interfaces are:

- * Mechanical
- * Electrical
- * Software

To design one well:

- * Think outside in
- * Be explicit when designing an interface
- * Be strict when implementing an existing interface

Designing a perfect interface is impossible but a good interface isn't. By thinking outside in, being explicit on specs and strict in implementation you'll have the best shot at producing an interface that enables abstraction and reuse.