What. Not how.

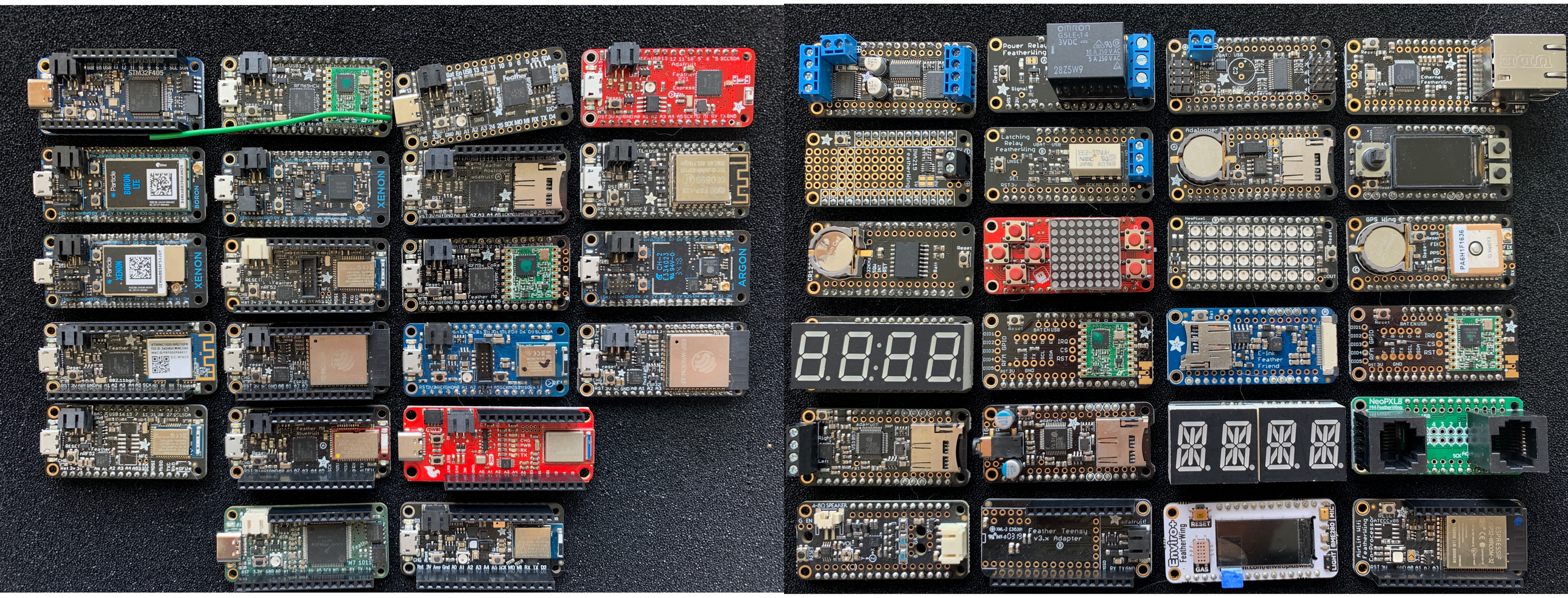
Scott (@tannewt)

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.

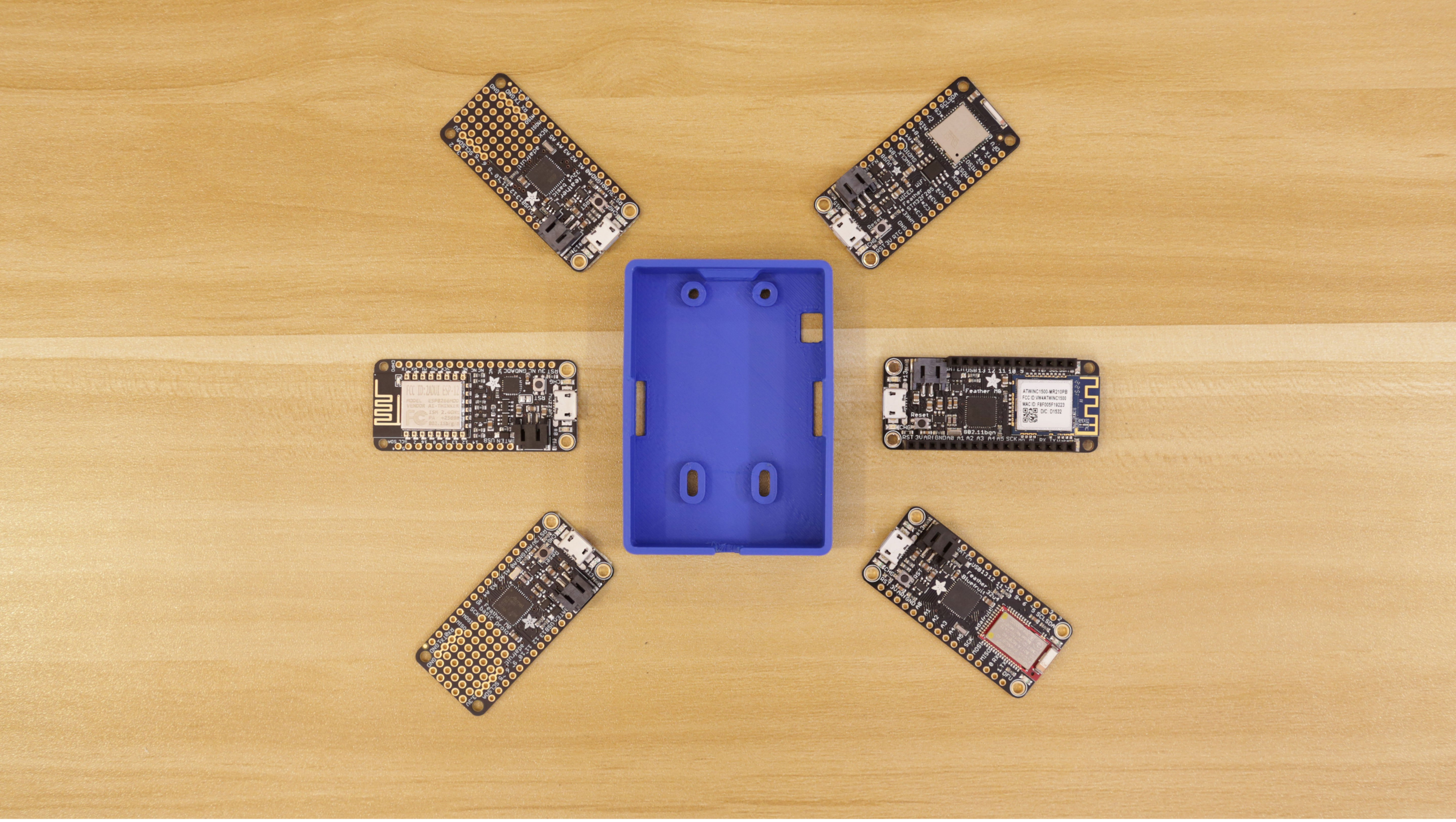
Abstraction of the *how* into *what* allows for reuse.


STM32F405
BORON
XENON
ARGON
Feather Teensy
v3.x Adapter
Power Relay
FeatherWing
Latching
Relay
FeatherWing
Adalogger
GPS Wing
NeoPixel
FeatherWing
E-Ink
Feather
Friend
NeoPXL8
M4 FeatherWing
4-8Ω SPEAKER
Envirox
FeatherWing
RESET
GAS
AirLift Wifi
Featherwing
ESP32 WROOM-32E

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

In open hardware there are three common category of interfaces:

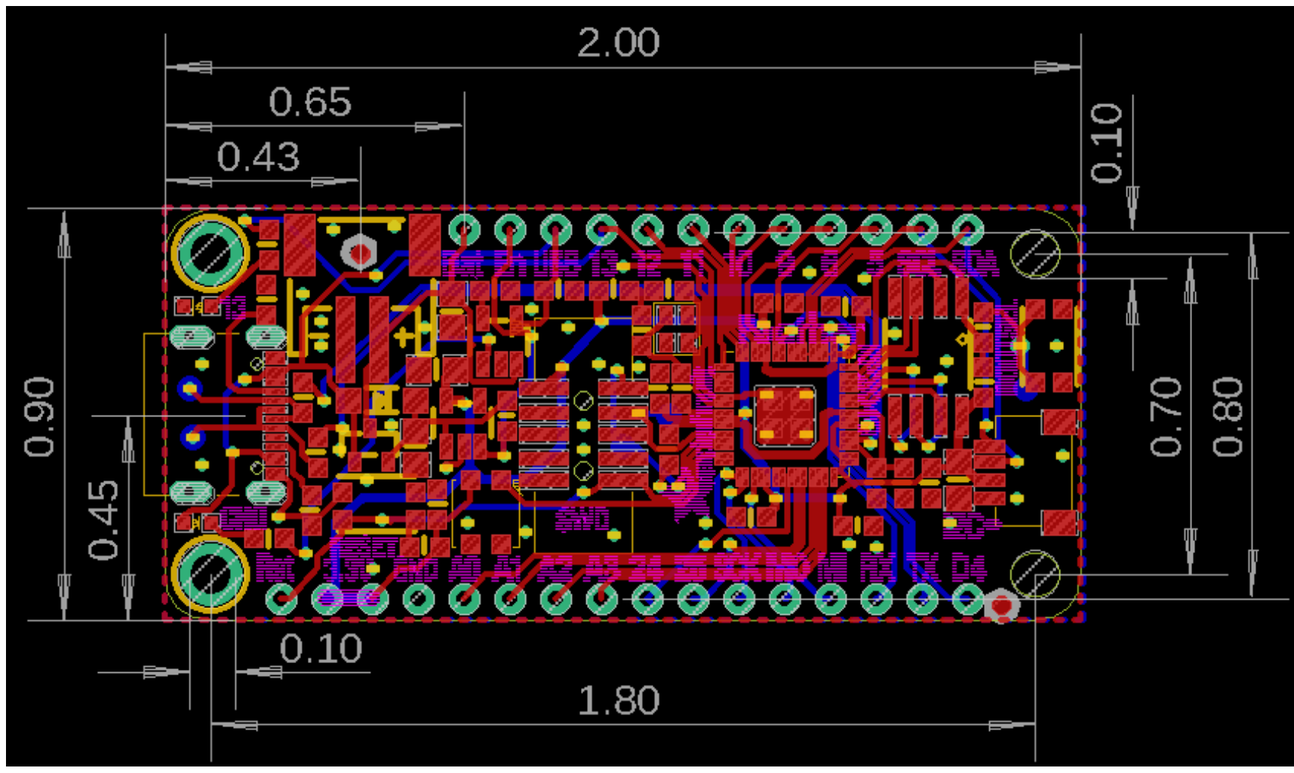
- Mechanical
- Electrical
- Software



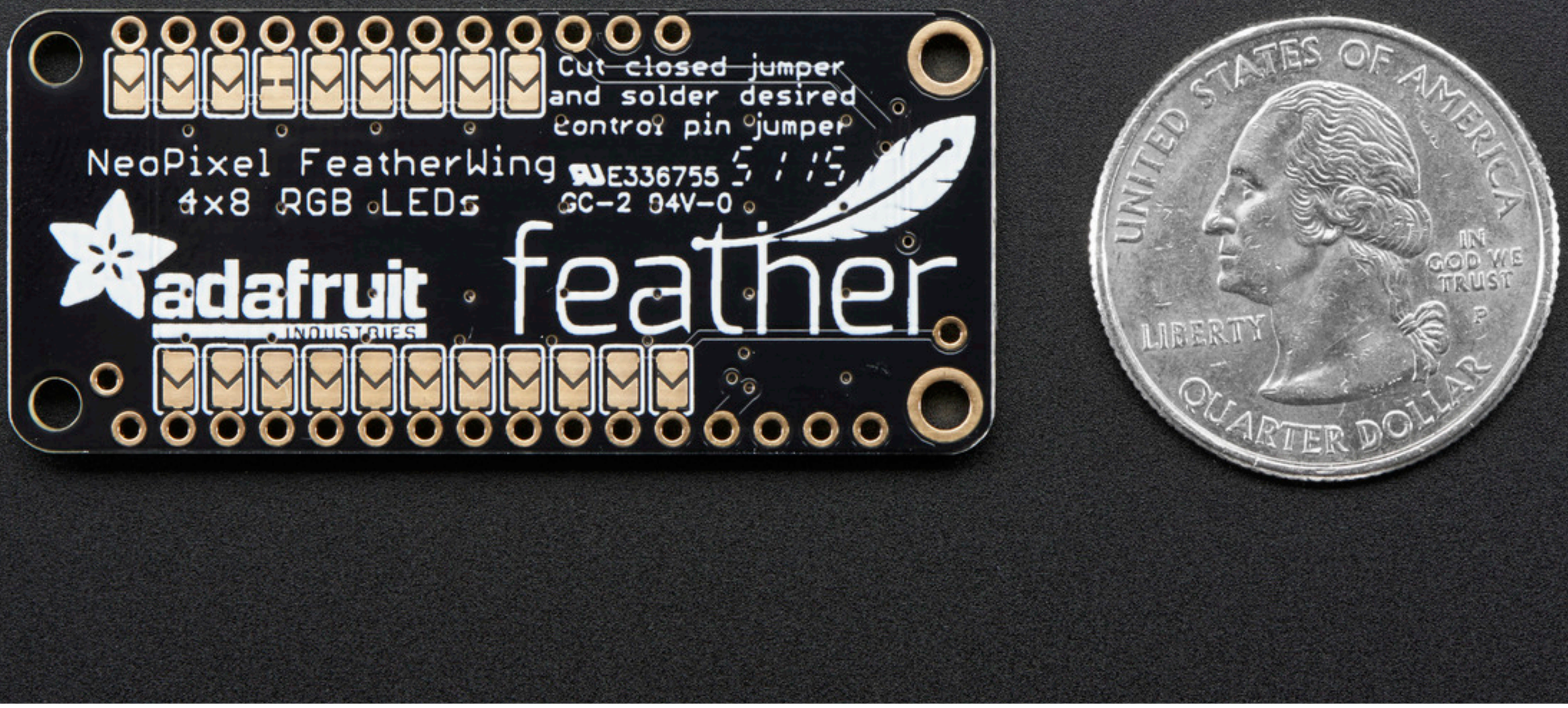
Outside In mechanical design may be designing for an existing case. In this case, the print was designed inside out. The second hole is a slot to accomodate different feather lengths. This is a sign of flexibility in the interface that if unbounded could lead to incompatibility and break the abstraction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

Mechanical

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

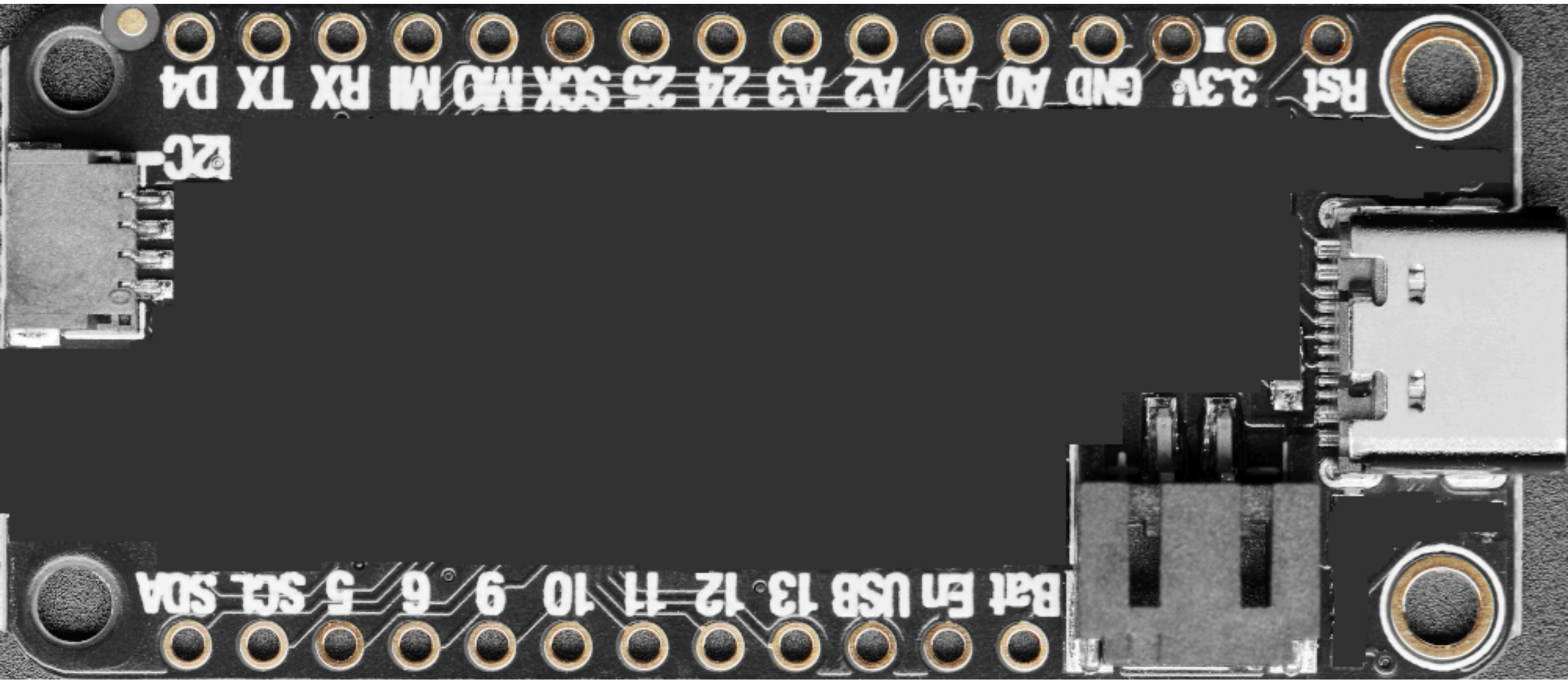


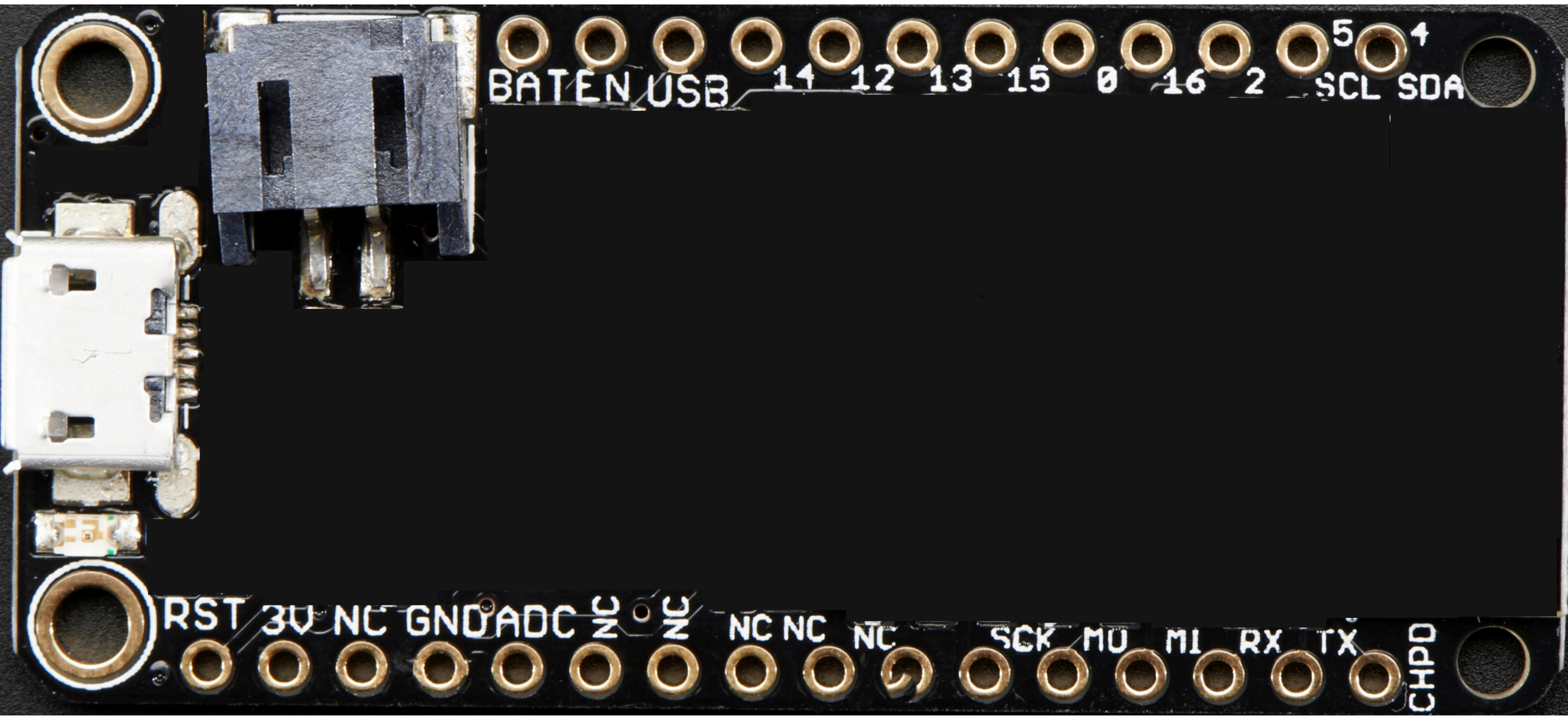

DOUT
DIN
Reset
NeoPixel
FeatherWing

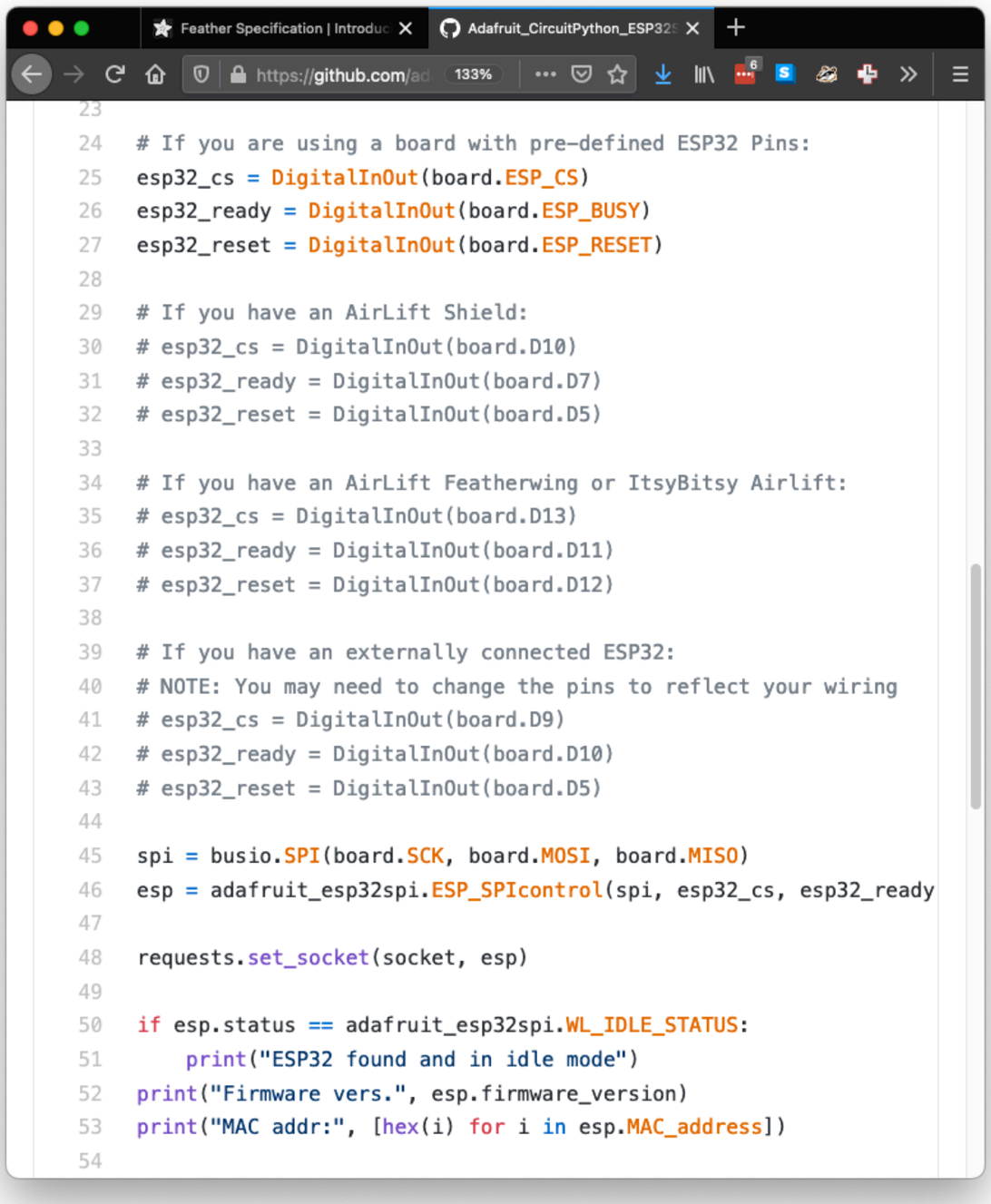

Cut closed jumper
and solder desired
control pin jumper
NeoPixel FeatherWing
4x8 RGB LEDs
E336755
GC-2 94V-0
adafruit
INDUSTRIES
feather
UNITED STATES OF AMERICA
IN GOD WE TRUST
LIBERTY
QUARTER DOLLAR

Electrical

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


Rst 3.3V GND A0 A1 A2 A3 24 25 SCK MO MI RX TX D4
I2C
Batt En USB 13 12 11 10 9 6 5 SCL SDA


BAT EN USB 11 12 13 15 0 16 2 5 4 SCL SDA
RST 3V NC GND ADC NC NC NC NC NC SCK MO MI RX TX CHPD

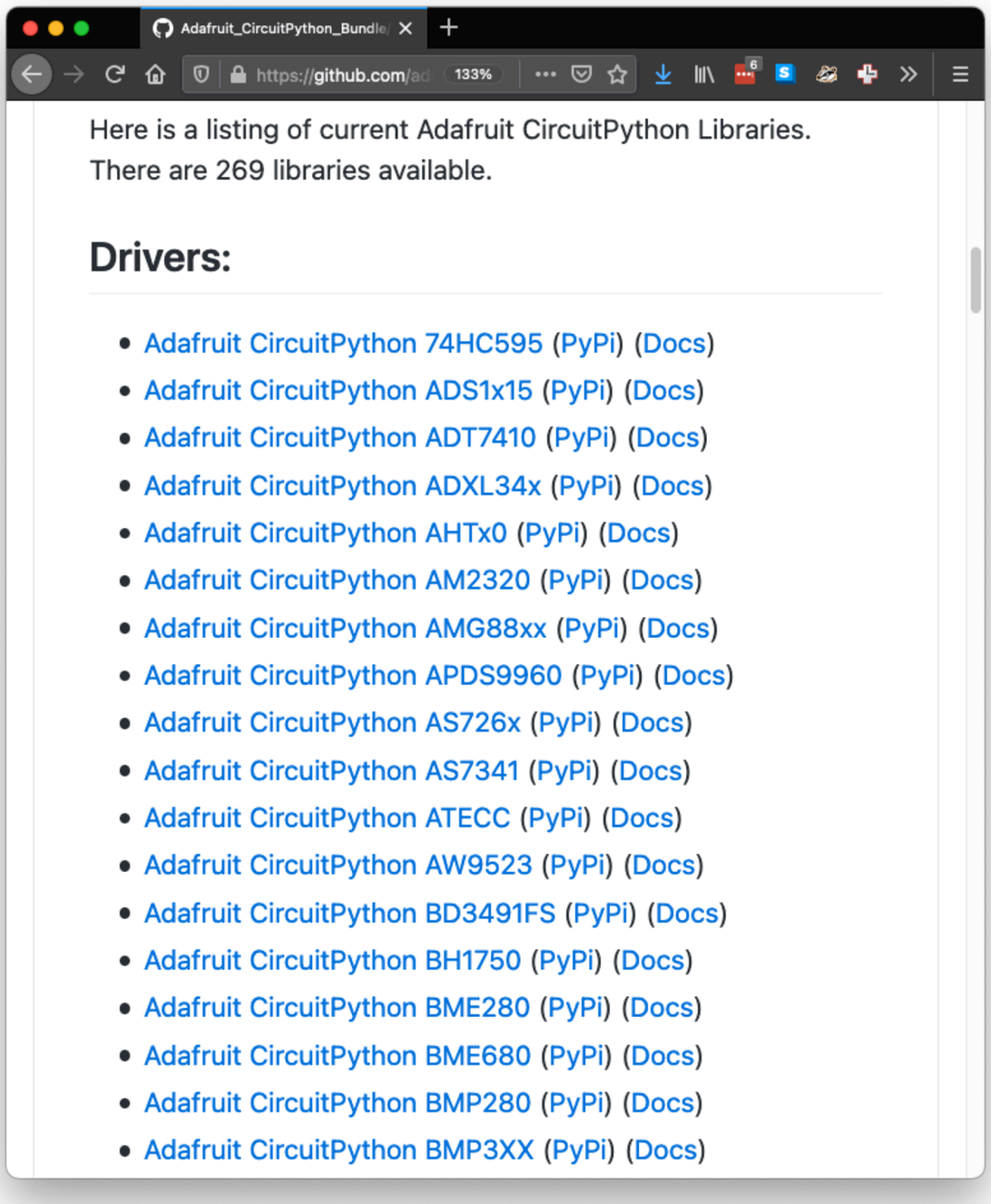



```
23
24  # If you are using a board with pre-defined ESP32 Pins:
25  esp32_cs = DigitalInOut(board.ESP_CS)
26  esp32_ready = DigitalInOut(board.ESP_BUSY)
27  esp32_reset = DigitalInOut(board.ESP_RESET)
28
29  # If you have an AirLift Shield:
30  # esp32_cs = DigitalInOut(board.D10)
31  # esp32_ready = 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

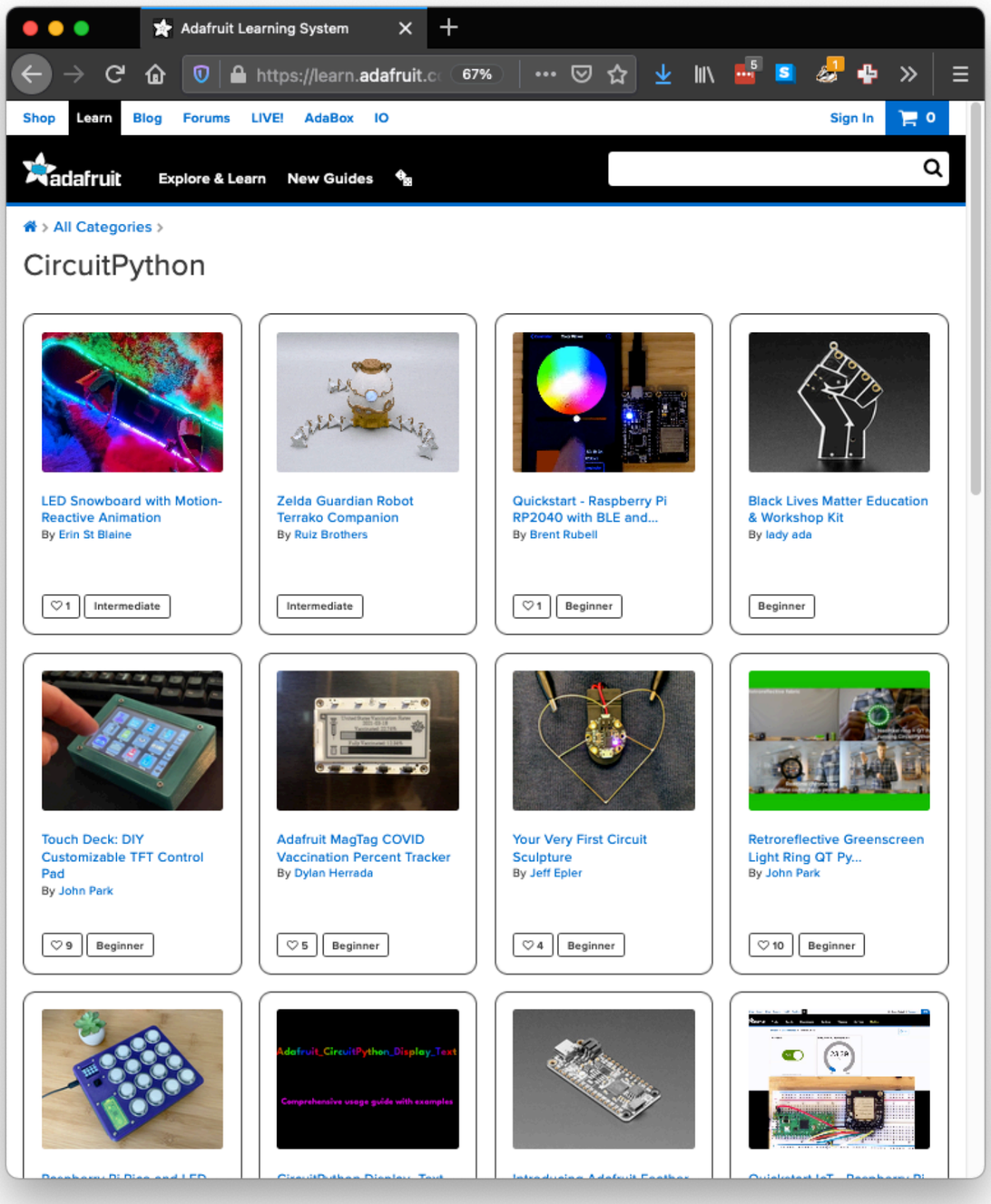
- Names!
- Interaction
- Default software
- Bootloader



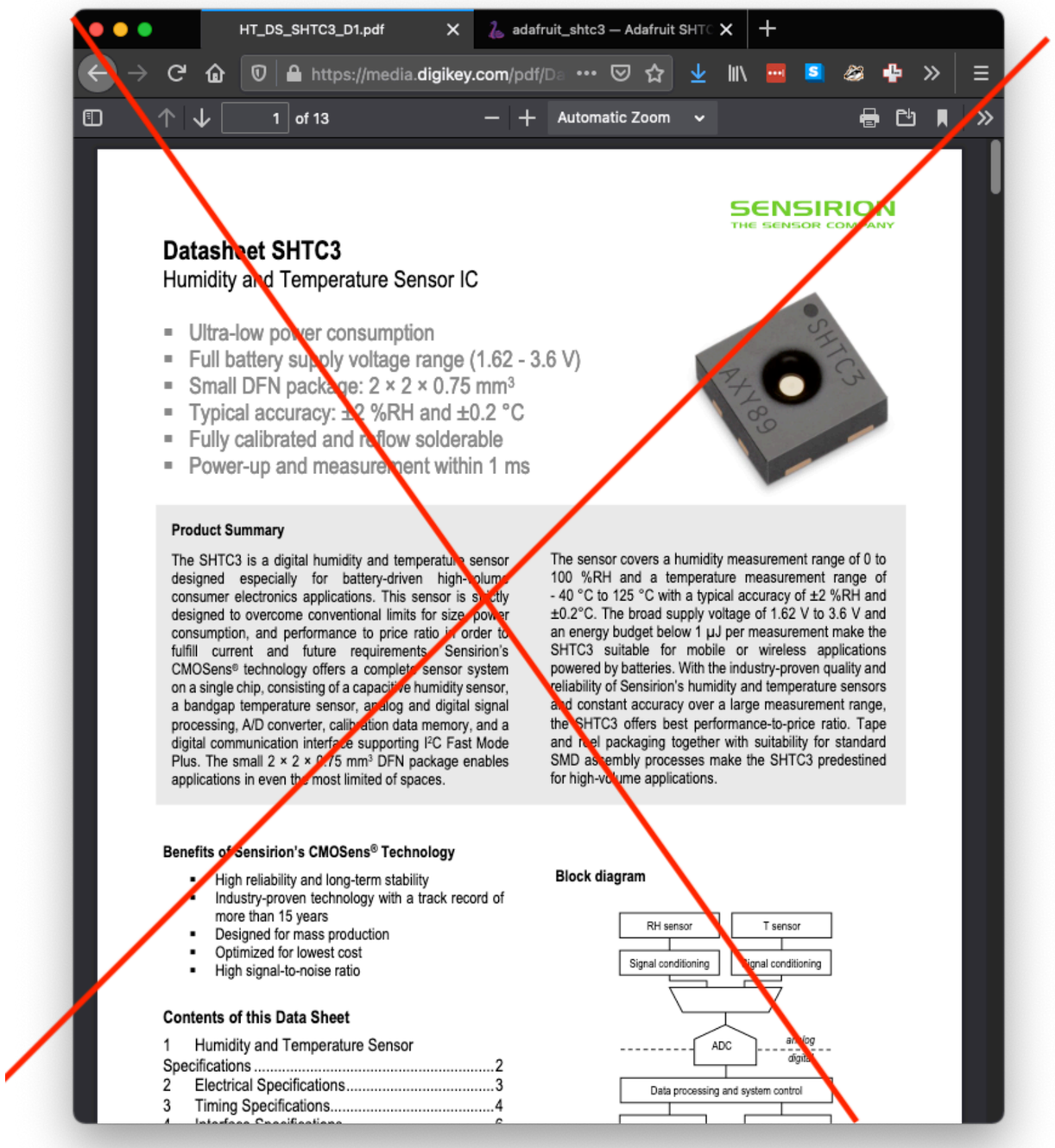
Here is a listing of current Adafruit CircuitPython Libraries.
There are 269 libraries available.

Drivers:

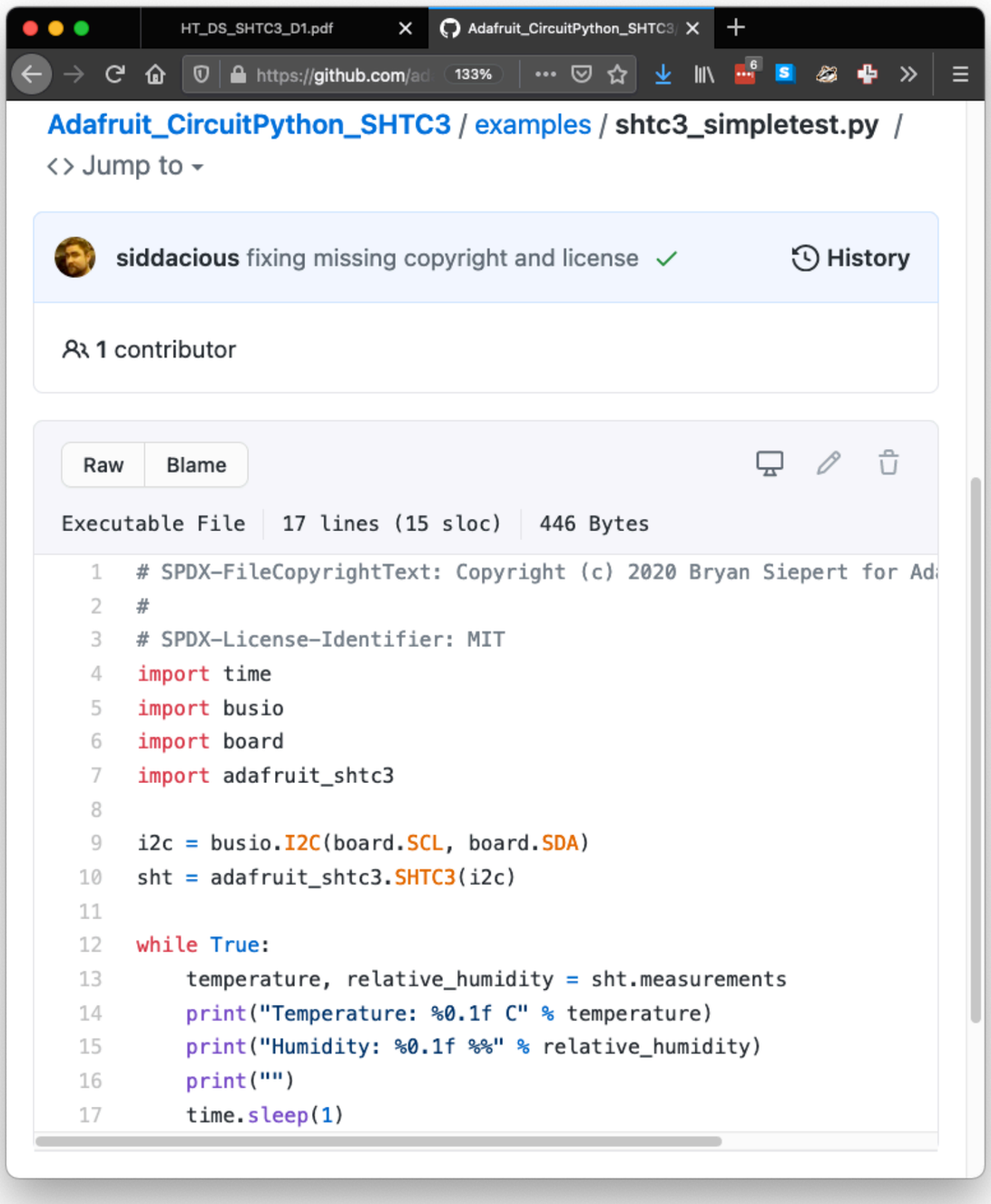
- Adafruit CircuitPython 74HC595 (PyPi) (Docs)
- Adafruit CircuitPython ADS1x15 (PyPi) (Docs)
- Adafruit CircuitPython ADT7410 (PyPi) (Docs)
- Adafruit CircuitPython ADXL34x (PyPi) (Docs)
- Adafruit CircuitPython AHTx0 (PyPi) (Docs)
- Adafruit CircuitPython AM2320 (PyPi) (Docs)
- Adafruit CircuitPython 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)


Adafruit Learning System
https://learn.adafruit.c
67%
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
1
Intermediate
Zelda Guardian Robot Terrako Companion
By Ruiz Brothers
Intermediate
Quickstart - Raspberry Pi RP2040 with BLE and...
By Brent Rubell
1
Beginner
Black Lives Matter Education & Workshop Kit
By lady ada
Beginner
Touch Deck: DIY Customizable TFT Control Pad
By John Park
9
Beginner
Adafruit MagTag COVID Vaccination Percent Tracker
By Dylan Herrada
5
Beginner
Your Very First Circuit Sculpture
By Jeff Epler
4
Beginner
Retroreflective Greenscreen Light Ring QT Py...
By John Park
10
Beginner
Adafruit_CircuitPython_Display_Text
Comprehensive usage guide with examples

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.


HT_DS_SHTC3_D1.pdf
Adafruit_CircuitPython_SHTC3/
https://github.com/ad
133%
Adafruit_CircuitPython_SHTC3 / examples / shtc3_simpletest.py /
Jump to
siddacious fixing missing copyright and license
History
1 contributor
Raw
Blame
Executable File
17 lines (15 sloc)
446 Bytes
SPDX-FileCopyrightText: Copyright (c) 2020 Bryan Siepert for Ad
#
# SPDX-License-Identifier: MIT
import time
import busio
import board
import adafruit_shtc3
i2c = busio.I2C(board.SCL, board.SDA)
sht = adafruit_shtc3.SHTC3(i2c)
while True:
temperature, relative_humidity = sht.measurements
print("Temperature: %0.1f C" % temperature)
print("Humidity: %0.1f %%" % relative_humidity)
print("")
time.sleep(1)

- 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

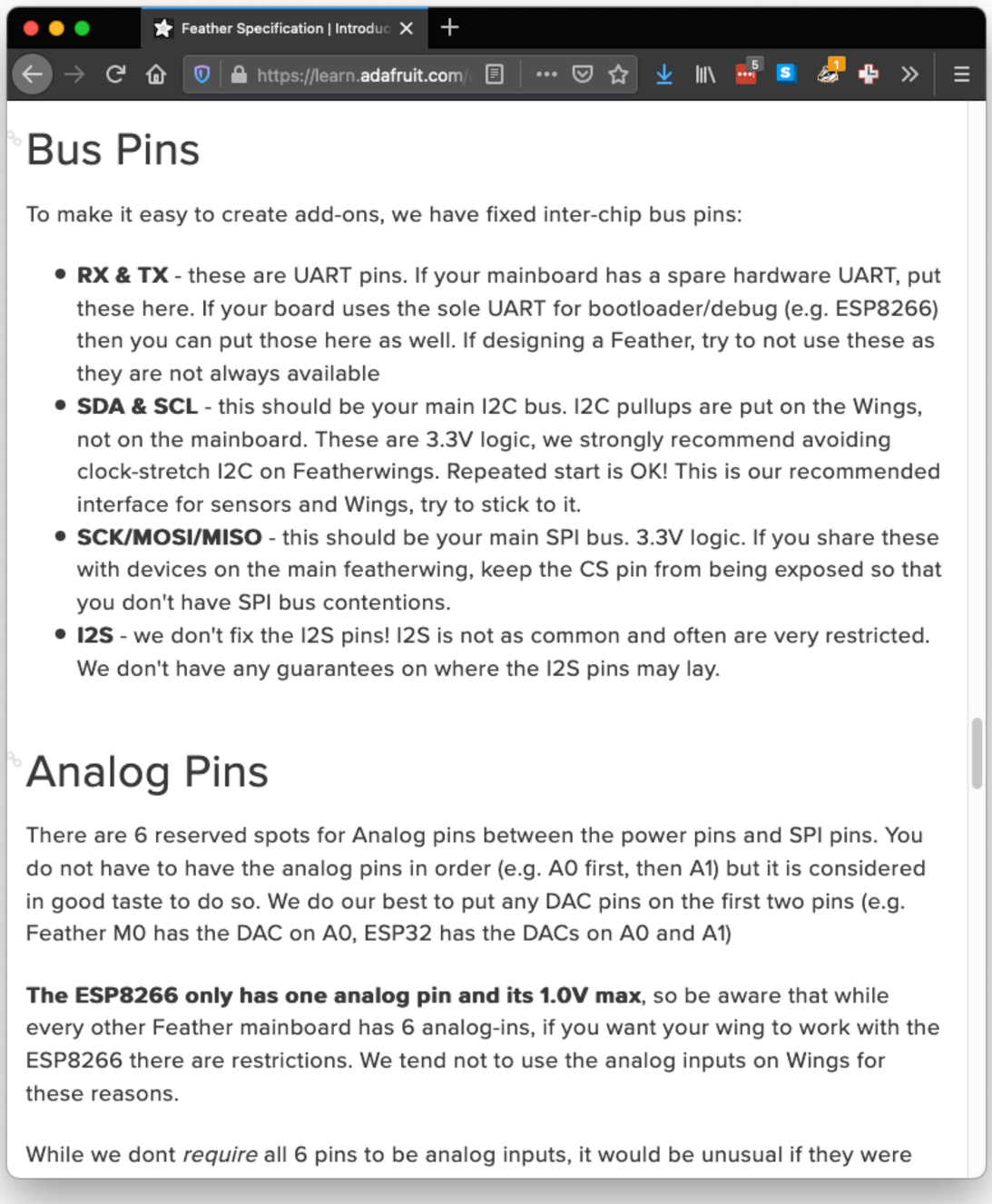
Determines the sleep state of the sensor

temperature

The current temperature in degrees celcius

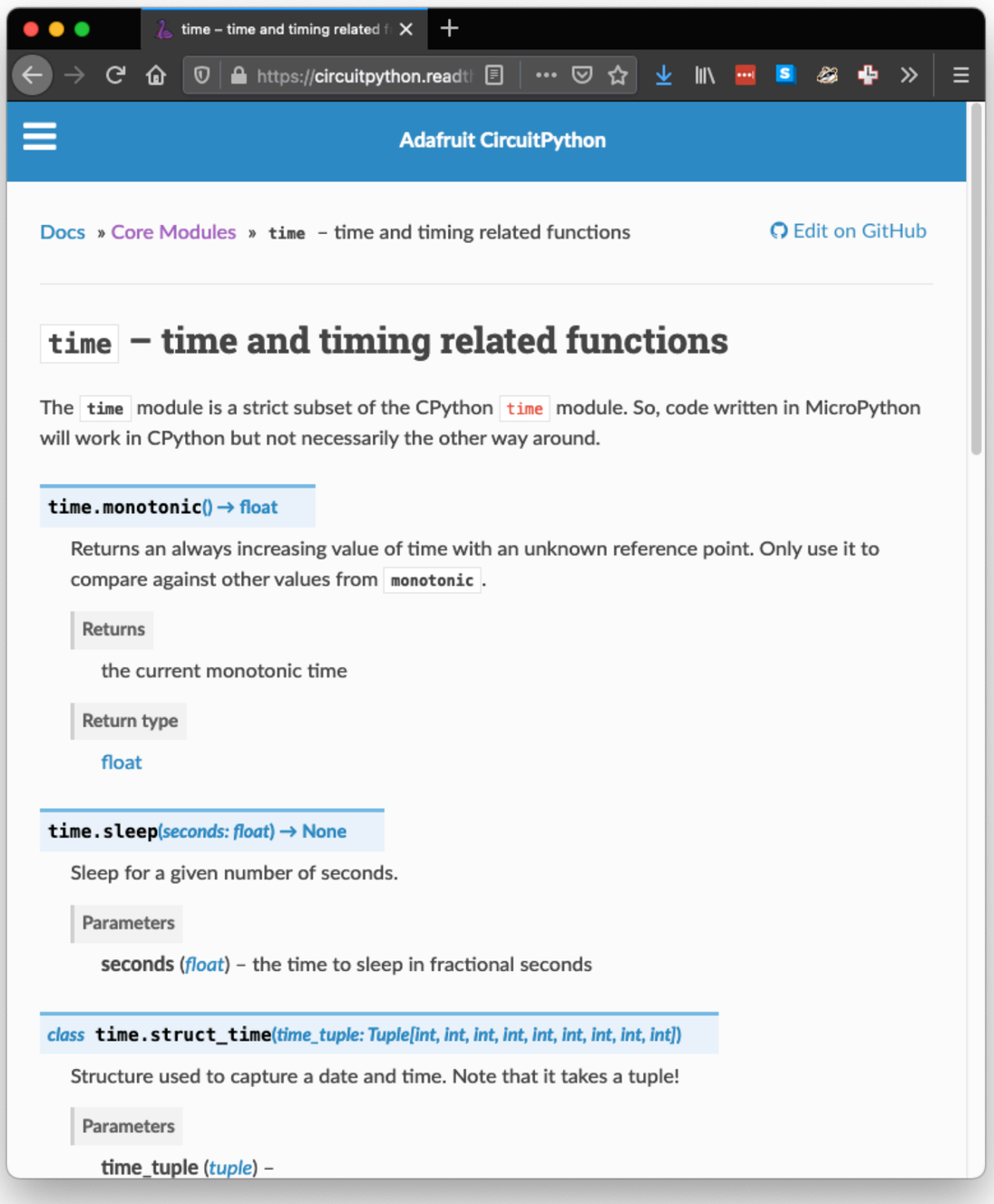
Previous

Be Explicit



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

Be Strict



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.