

Outside In

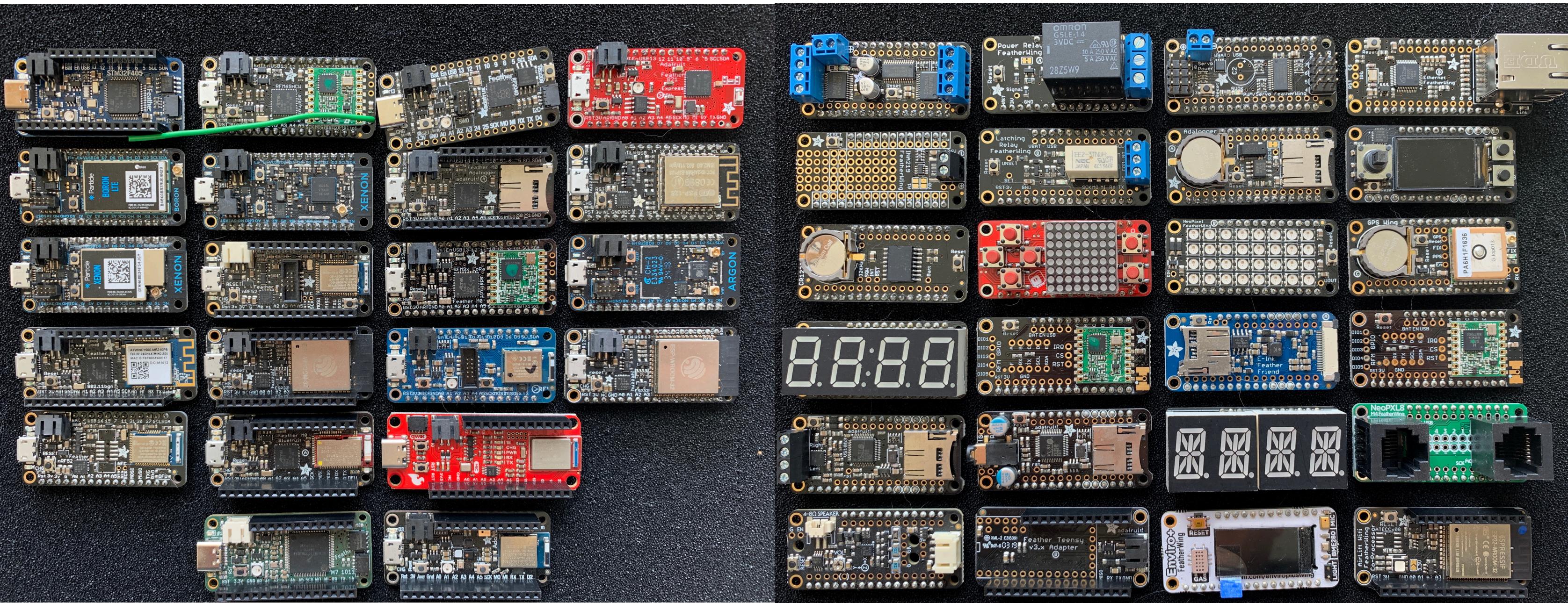
Better Interface Design

Scott (@tannewt)

An *interface* is a boundary between
two things

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

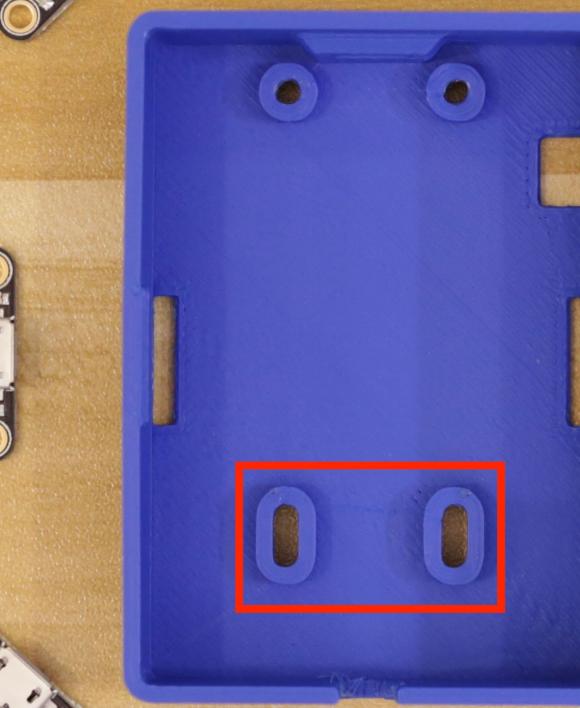
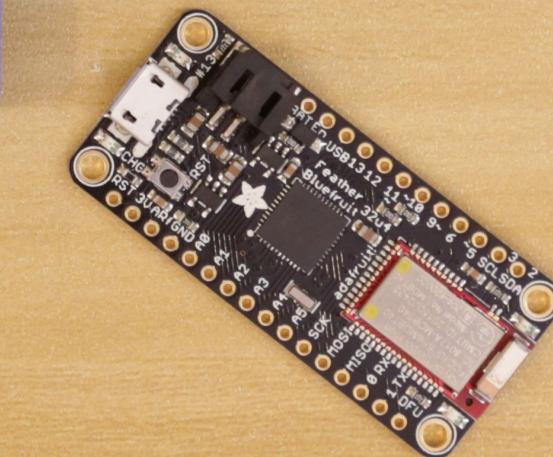
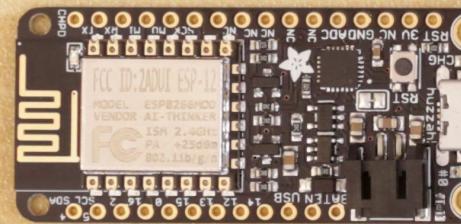
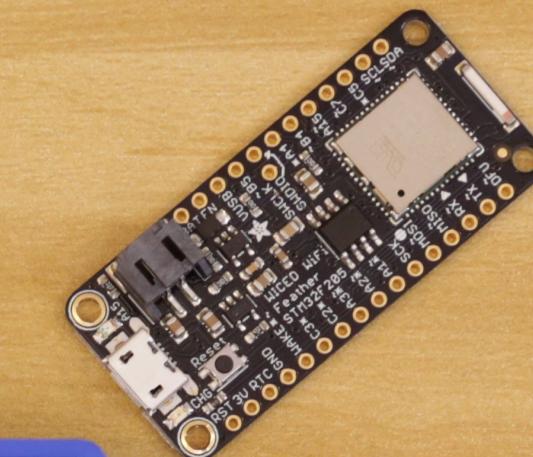
Abstraction of the *how* into *what*
allows for reuse.



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

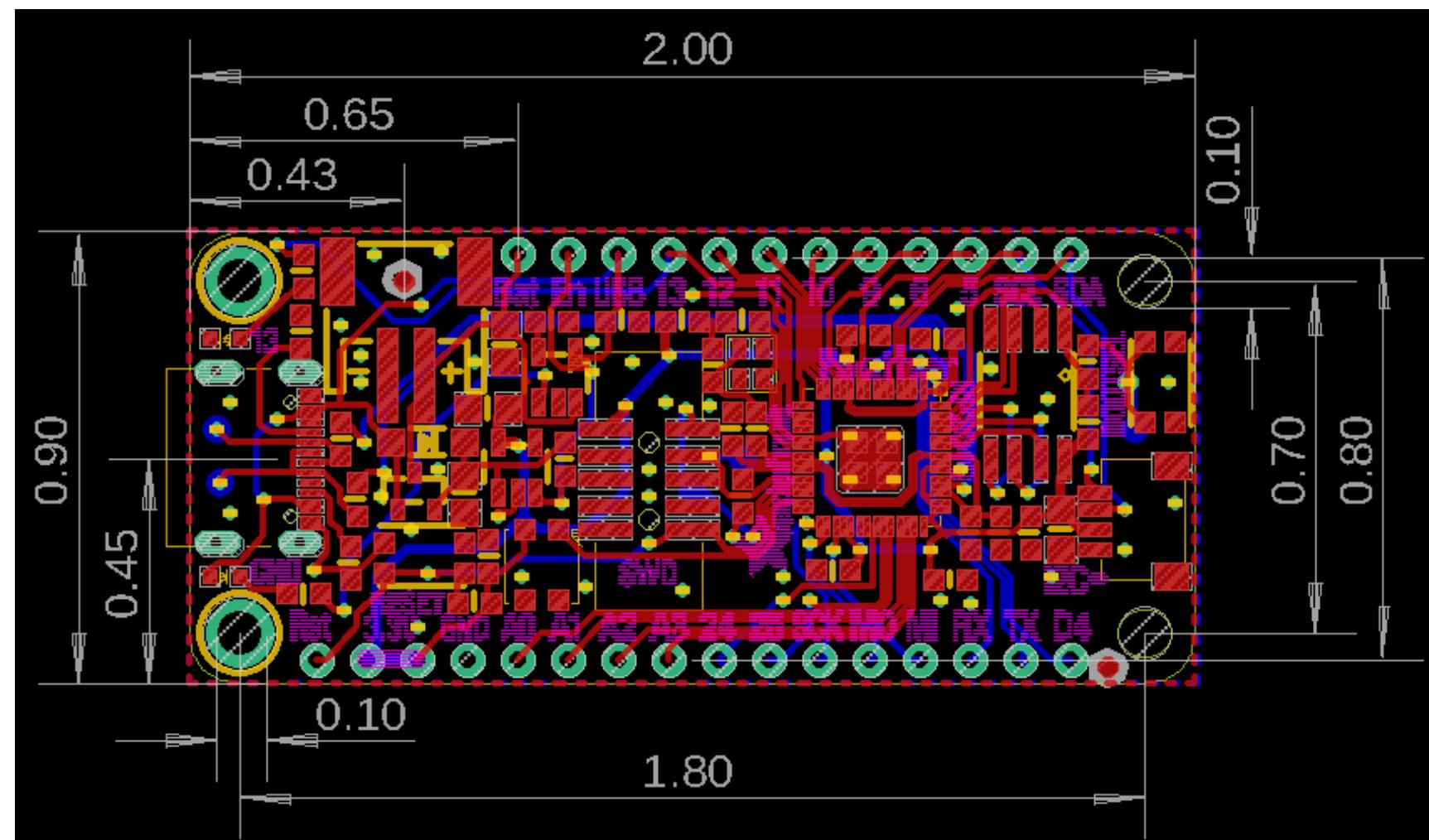
In open hardware there are three common category of interfaces:

- Mechanical
- Electrical
- Software



Mechanical

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



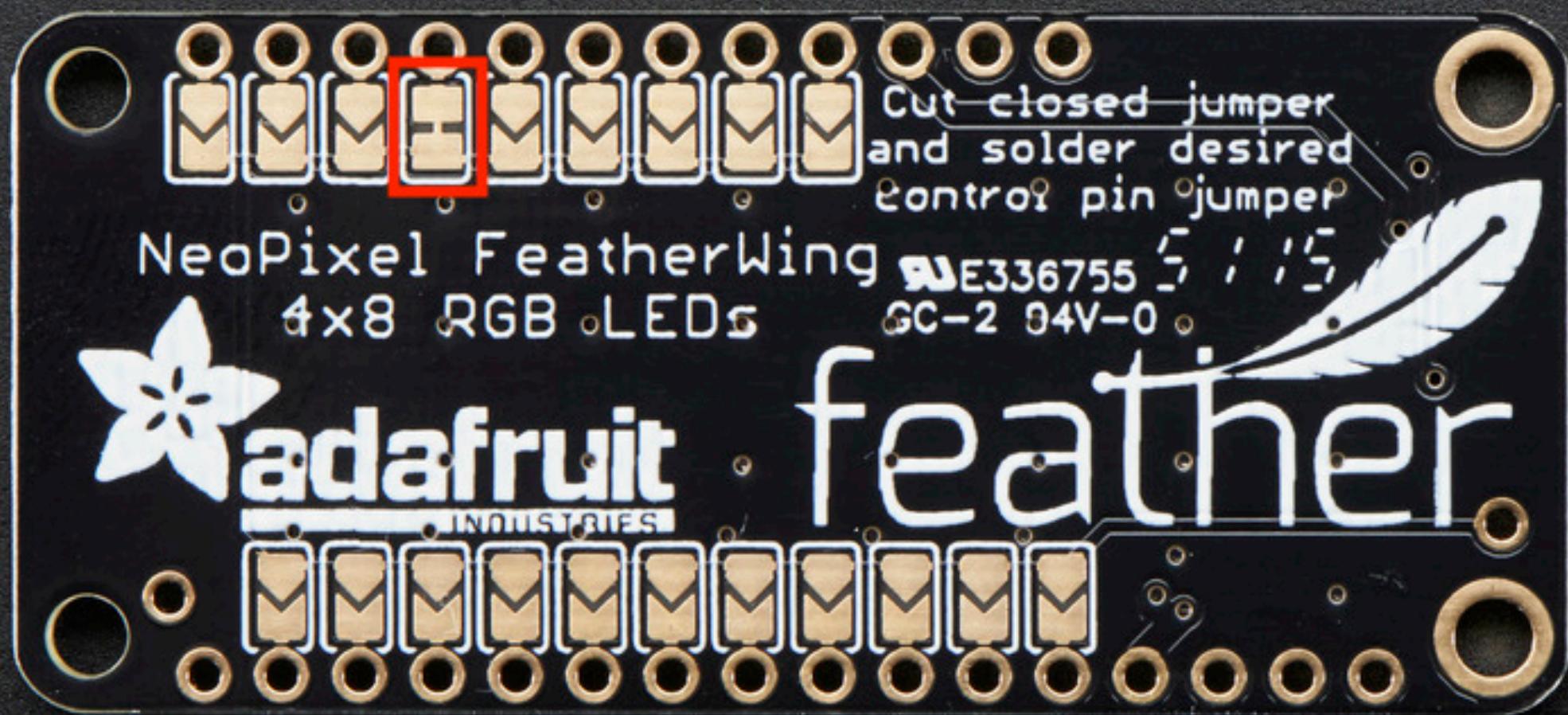
NeoPixel
FeatherWing

B

Reset

DIN

DOUT



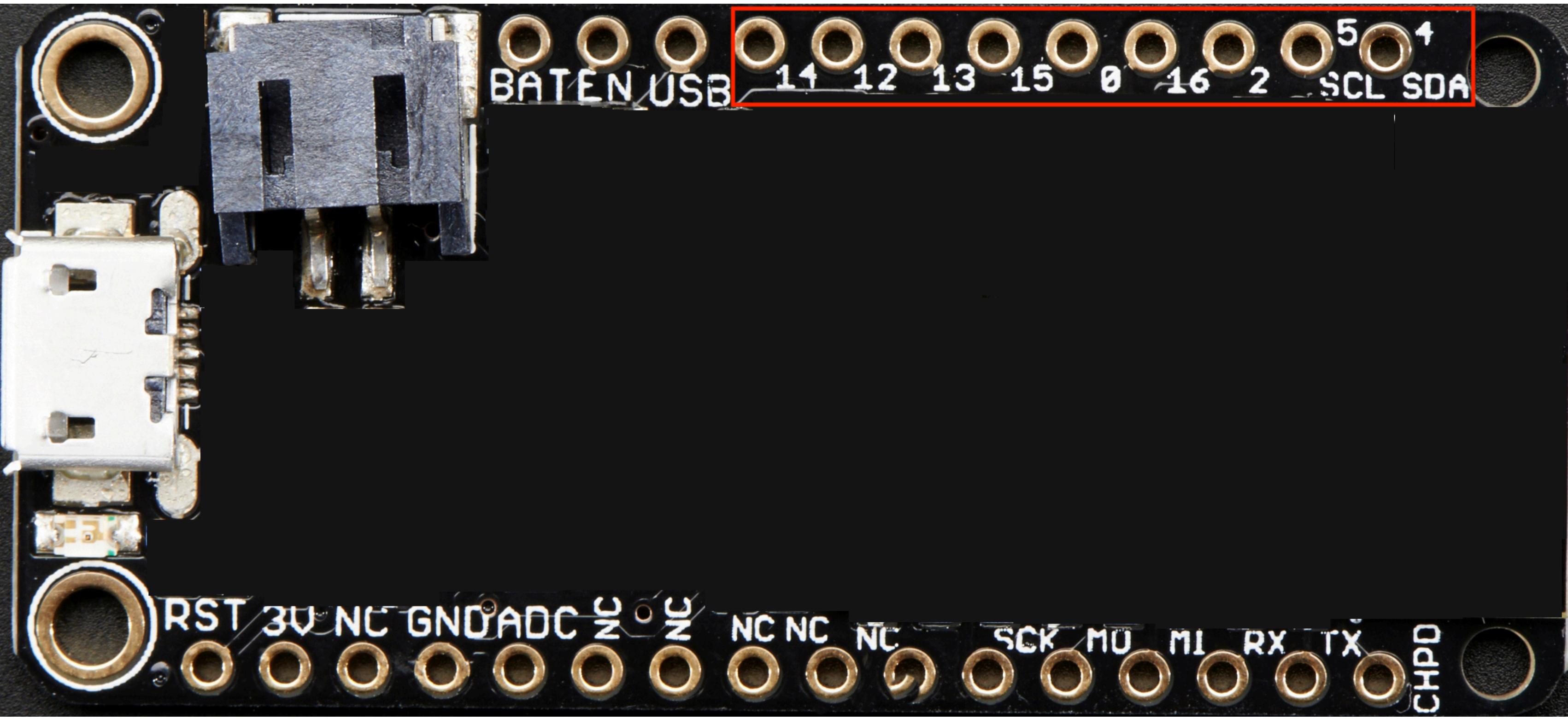
Electrical

- 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 M RX TX D4

I²C



A screenshot of a web browser window displaying a Python script. The browser's title bar shows "Feather Specification | Introduct X" and "Adafruit_CircuitPython_ESP32S X". The address bar indicates the URL is https://github.com/adafruit/Adafruit_CircuitPython_ESP32S. The script itself is a code example for connecting to an ESP32 using CircuitPython. It includes comments for different board configurations (ESP32 pins, AirLift Shield, AirLift Featherwing/ItsyBitsy Airlift), a note about pin changes, and a section for externally connected ESP32s. The code uses the `busio.SPI` module for SPI communication and the `adafruit_esp32spi.ESP_SPIcontrol` class to manage the ESP32.

```
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
```

Software

- Names!
- Interaction
- Default software
- Bootloader

Adafruit_CircuitPython_Bundle X +

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

Shop Learn Blog Forums LIVE! AdaBox IO Sign In Cart 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

Deepboard DI Buttons and LEDs

CircuitPython Display_Text
Comprehensive usage guide with examples

Introduction to Adafruit Fingerprint

Qualcomm IoT - Deepboard DI

Think Outside In

HT_DS_SHTC3_D1.pdf X adafruit_shtc3 — Adafruit SHTC X +

1 of 13 Automatic Zoom

Datasheet SHTC3

Humidity and Temperature Sensor IC

- Ultra-low power consumption
- Full battery supply voltage range (1.62 - 3.6 V)
- Small DFN package: 2 × 2 × 0.75 mm³
- Typical accuracy: ±2 %RH and ±0.2 °C
- Fully calibrated and reflow solderable
- Power-up and measurement within 1 ms

Product Summary

The SHTC3 is a digital humidity and temperature sensor designed especially for battery-driven high-volume consumer electronics applications. This sensor is specifically designed to overcome conventional limits for size, power consumption, and performance-to-price ratio in order to fulfill current and future requirements. Sensirion's CMOSens® technology offers a complete sensor system on a single chip, consisting of a capacitive humidity sensor, a bandgap temperature sensor, analog and digital signal processing, A/D converter, calibration data memory, and a digital communication interface supporting I²C Fast Mode Plus. The small 2 × 2 × 0.75 mm³ DFN package enables applications in even the most limited of spaces.

The sensor covers a humidity measurement range of 0 to 100 %RH and a temperature measurement range of -40 °C to 125 °C with a typical accuracy of ±2 %RH and ±0.2°C. The broad supply voltage of 1.62 V to 3.6 V and an energy budget below 1 µJ per measurement make the SHTC3 suitable for mobile or wireless applications powered by batteries. With the industry-proven quality and reliability of Sensirion's humidity and temperature sensors and constant accuracy over a large measurement range, the SHTC3 offers best performance-to-price ratio. Tape and reel packaging together with suitability for standard SMD assembly processes make the SHTC3 predestined for high-volume applications.

Benefits of Sensirion's CMOSens® Technology

- High reliability and long-term stability
- Industry-proven technology with a track record of more than 15 years
- Designed for mass production
- Optimized for lowest cost
- High signal-to-noise ratio

Contents of this Data Sheet

1 Humidity and Temperature Sensor Specifications	2
2 Electrical Specifications	3
3 Timing Specifications	4
4 Interface Specifications	5

Block diagram

```
graph TD; RH[RH sensor] --> SC1[Signal conditioning]; T[T sensor] --> SC2[Signal conditioning]; SC1 --> ADC[ADC]; SC2 --> ADC; ADC --> DPS[Data processing and system control];
```

HT_DS_SHTC3_D1.pdf X Adafruit_CircuitPython_SHTC3 X +

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

```
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: {:.1f} C" % temperature)
15     print("Humidity: {:.1f} %" % relative_humidity)
16     print("")
17     time.sleep(1)
```

HT_DS_SHTC3_D1.pdf X adafruit_shtc3 – Adafruit SHTC X +

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

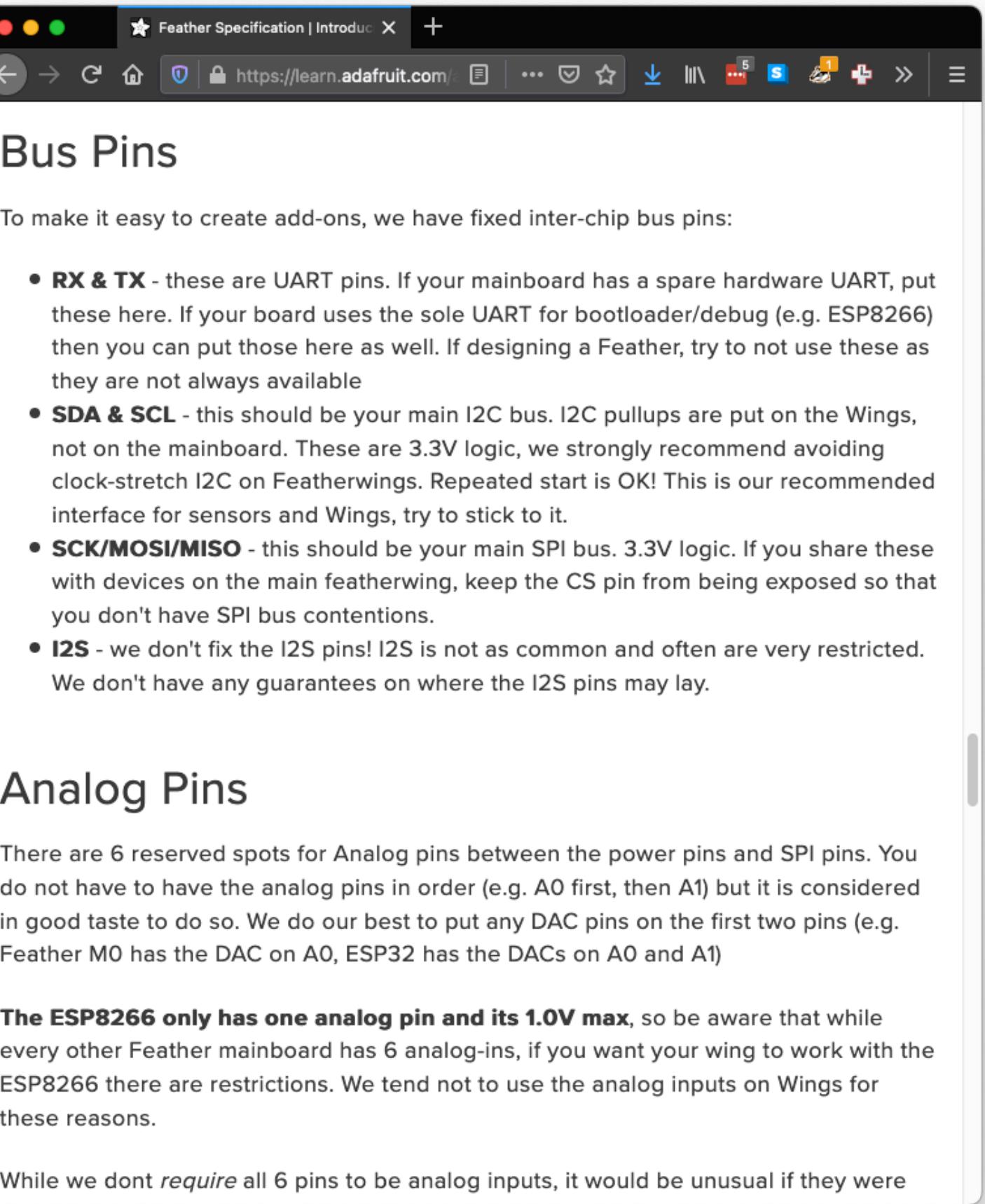
temperature

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 title "Feather Specification | Introduct X". The URL in the address bar is <https://learn.adafruit.com/>. The page content is titled "Bus Pins" and includes the following text and list:

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

Be Strict

The screenshot shows a web browser displaying the Adafruit CircuitPython documentation for the `time` module. The page title is "time – time and timing related functions". The URL in the address bar is https://circuitpython.readthedocs.io/en/latest/_modules/time.html. The page content includes a brief introduction stating that the `time` module is a strict subset of the CPython `time` module, followed by detailed descriptions of three functions: `monotonic()`, `sleep()`, and `struct_time`.

time – time and timing related functions

The `time` module is a strict subset of the CPython `time` module. So, code written in MicroPython will work in CPython but not necessarily the other way around.

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)` –

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