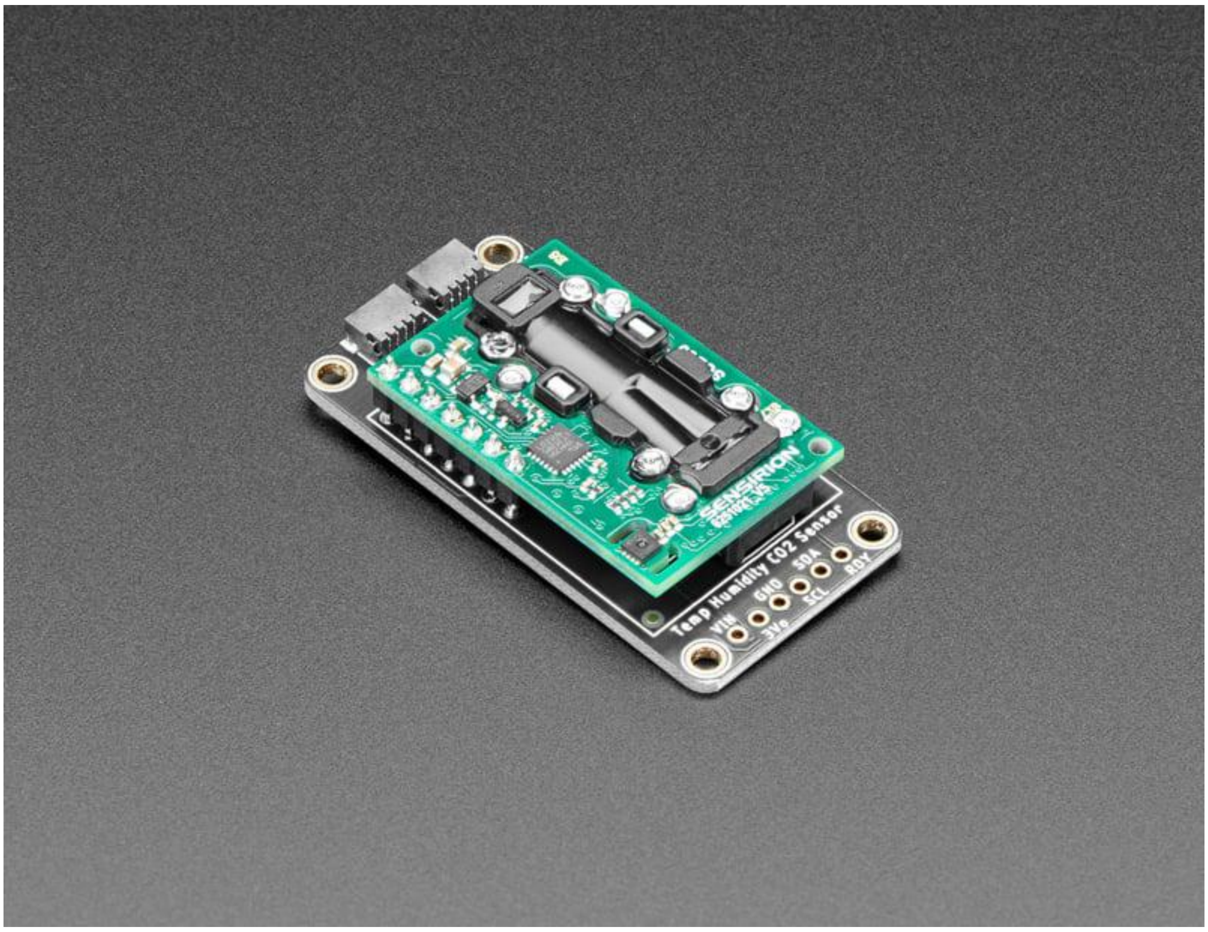
Adafruit SCD-30 - NDIR CO2 Temperature and Humidity Sensor

Created by Kattni Rembor



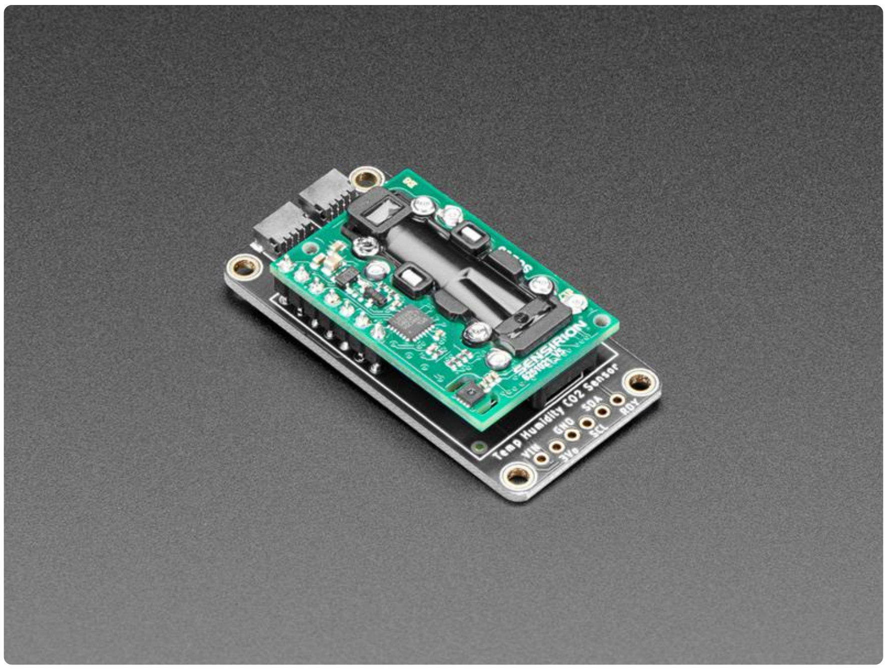
https://learn.adafruit.com/adafruit-scd30

Last updated on 2023-01-06 02:40:41 PM EST

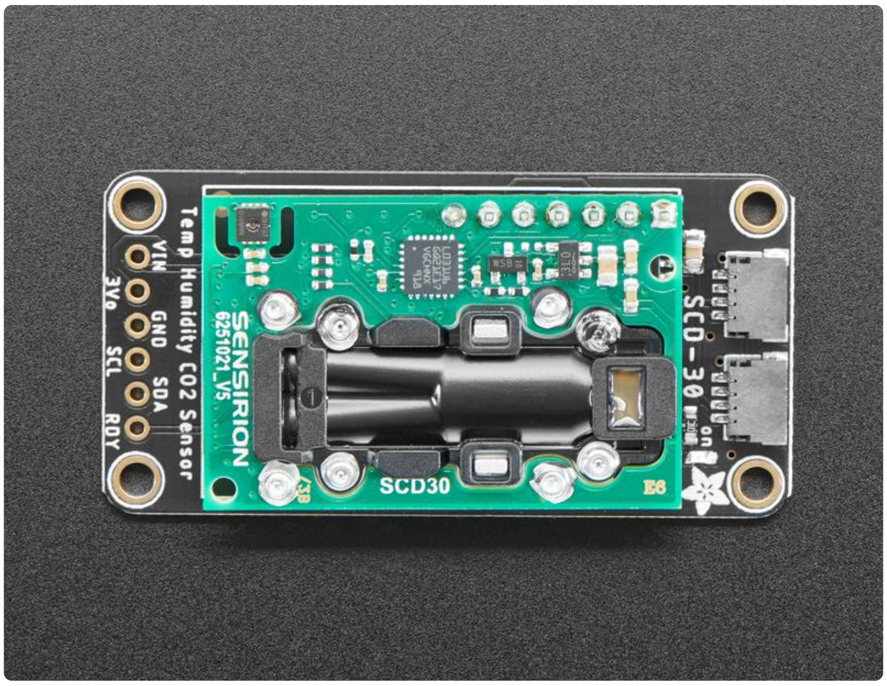
Table of Contents

Overview

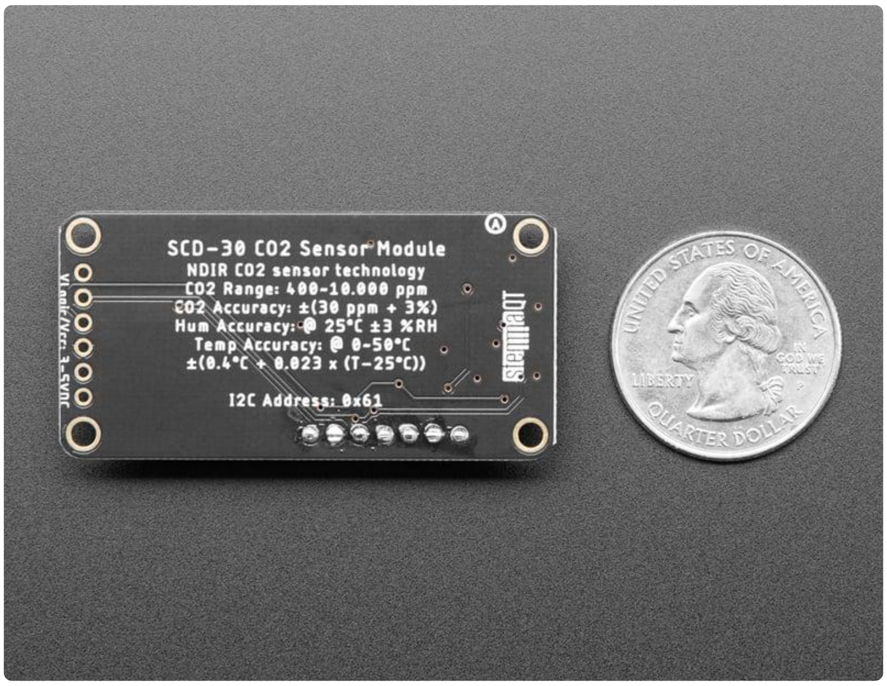
Take a deep breath in...now slowly breathe out. Mmm isn't it wonderful? All that air around us, which we bring into our lungs, extract oxygen from and then breathe out carbon dioxide. CO2 is essential for life on this planet we call Earth - us and plants take turns using and emitting CO2 in an elegant symbiosis. But it's important to keep that CO2 balanced - you don't want too much around, not good for humans and not good for our planet.



The SCD-30 is an NDIR sensor (), which is a 'true' CO2 sensor, that will tell you the CO2 PPM (parts-per-million) composition of ambient air. Unlike the SGP30, this sensor isn't approximating it from VOC gas concentration () - it really is measuring the CO2 concentration! That means its a lot bigger and more expensive, but it is the real thing. Perfect for environmental sensing, scientific experiments, air quality and ventilation studies and more.

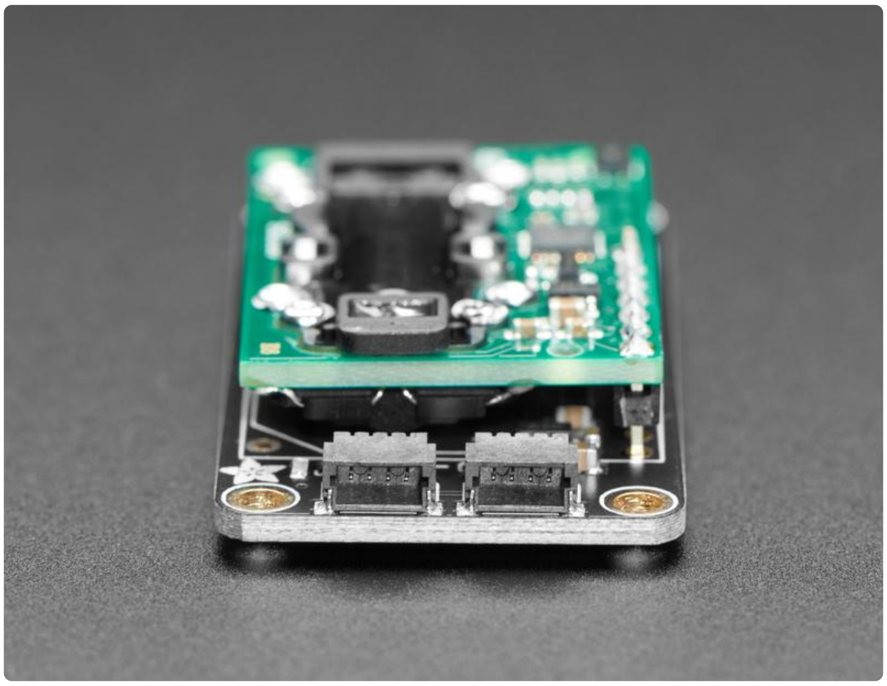


Data is read over I2C, so it works very nicely with just about any microcontroller or microcomputer. We've written both Arduino and Python/CircuitPython code so you can get started in a jiffy. Another nice element to this sensor is it comes with an SHT31 temperature and humidity sensor already built in (). The sensor is used to compensate the NDIR CO2 sensor, but its also readable so you get full environmental data.

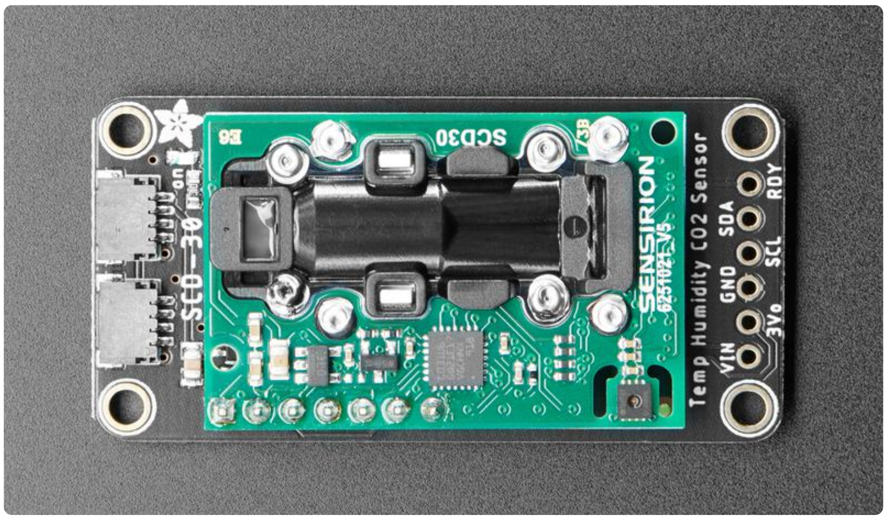


Nice sensor right? So we made it easy for you to get right into your next project. The sensor is hand-soldered onto a custom made PCB in the STEMMA QT form factor (), making them easy to interface with. The STEMMA QT connectors () on either side are compatible with the SparkFun Qwiic () I2C connectors. This allows you to make solderless connections between your development board and the SCD-30 or to chain it with a wide range of other sensors and accessories using a compatible cable ().

We've of course broken out all the pins to standard headers and added a 3.3V voltage regulator and level shifting so allow you to use it with either 3.3V or 5V systems such as the Raspberry Pi, or Metro M4 or Arduino Uno.



Pinouts



Power Pins

- VIN - this is the power pin. Since the sensor chip uses 3 VDC, we have included a voltage regulator on board that will take 3-5VDC and safely convert it down.

To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V microcontroller like Arduino, use 5V

- 3Vo - this is the 3.3V output from the voltage regulator, you can grab up to 100mA from this if you like
- GND - common ground for power and logic

I2C Logic Pins

- SCL - I2C clock pin, connect to your microcontroller I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.
- SDA - I2C data pin, connect to your microcontroller I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.
- STEMMA QT () - These connectors allow you to connect to dev boards with STEMMA QT connectors or to other things with various associated accessories ()

Other Pins

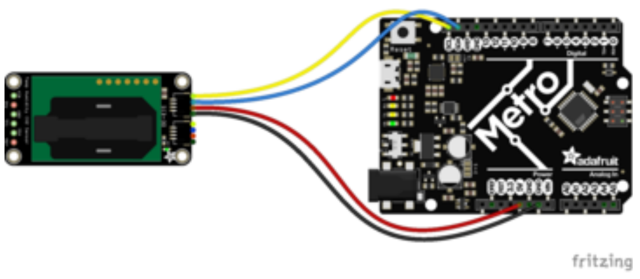
- RDY - Data Ready Pin. High when data is ready for read-out, it helps if you want to avoid polling the I2C port to verify data is ready.

Arduino

Using the SCD30 with Arduino is a simple matter of wiring up the sensor to your Arduino-compatible microcontroller, installing the Adafruit SCD30 () library we've written, and running the provided example code.

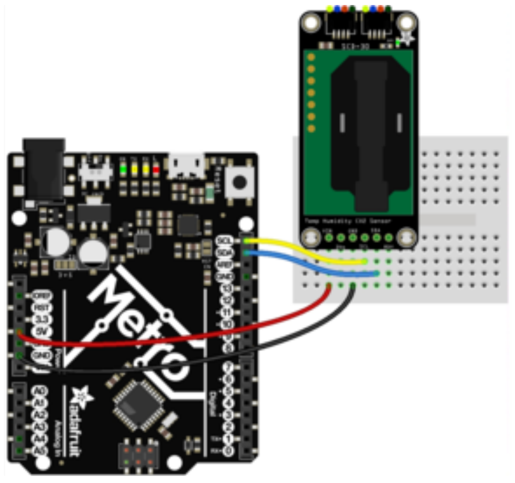
I2C Wiring

Here is how to wire up the sensor using one of the STEMMA QT () connectors. The examples show a Metro but wiring will work the same for an Arduino or other compatible board.



Connect board VIN (red wire) to Arduino 5V if you are running a 5V board Arduino (Uno, etc.). If your board is 3V, connect to that instead.
Connect board GND (black wire) to Arduino GND
Connect board SCL (yellow wire) to Arduino SCL
Connect board SDA (blue wire) to Arduino SDA

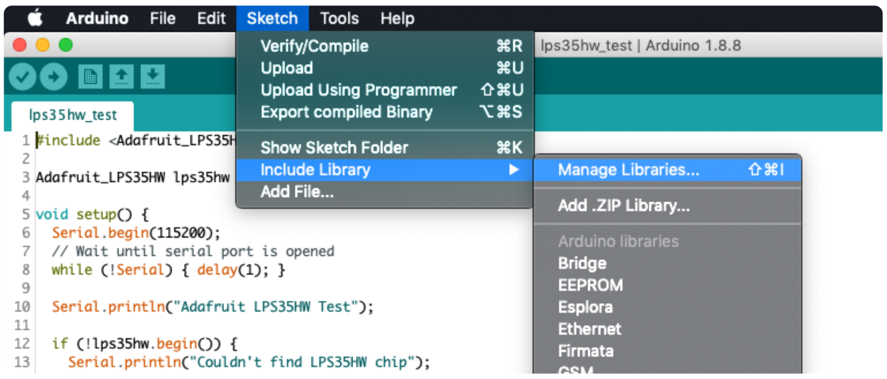
Here is how to wire the sensor to a board using a solderless breadboard:



Connect board VIN (red wire) to Arduino 5V if you are running a 5V board Arduino (Uno, etc.). If your board is 3V, connect to that instead.
Connect board GND (black wire) to Arduino GND
Connect board SCL (yellow wire) to Arduino SCL
Connect board SDA (blue wire) to Arduino SDA

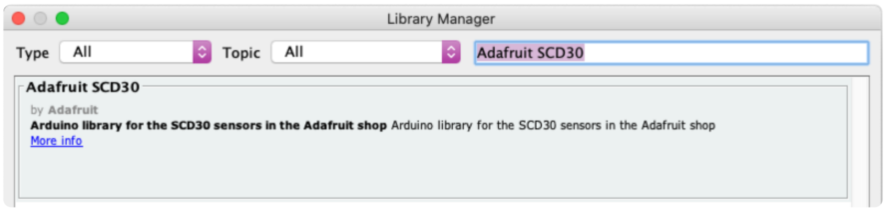
Library Installation

You can install the Adafruit SCD30 library for Arduino using the Library Manager in the Arduino IDE.



```
#include <Adafruit_LPS35H

Adafruit_LPS35HW lps35hw

void setup() {
  Serial.begin(115200);
  // Wait until serial port is opened
  while (!Serial) { delay(1); }

  Serial.println("Adafruit LPS35HW Test");

  if (!lps35hw.begin()) {
    Serial.println("Couldn't find LPS35HW chip");
```

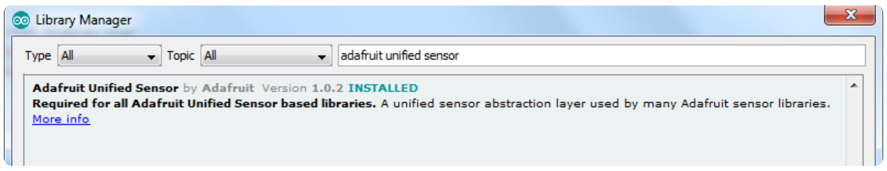
Click the Manage Libraries ... menu item, search for Adafruit SCD30 , and select the A dafruit SCD30 library:


Library Manager

Type All Topic All Adafruit SCD30

Adafruit SCD30
by Adafruit
Arduino library for the SCD30 sensors in the Adafruit shop Arduino library for the SCD30 sensors in the Adafruit shop
More info

Finally, search for Adafruit Unified Sensor and install that too (you may have to scroll a bit)

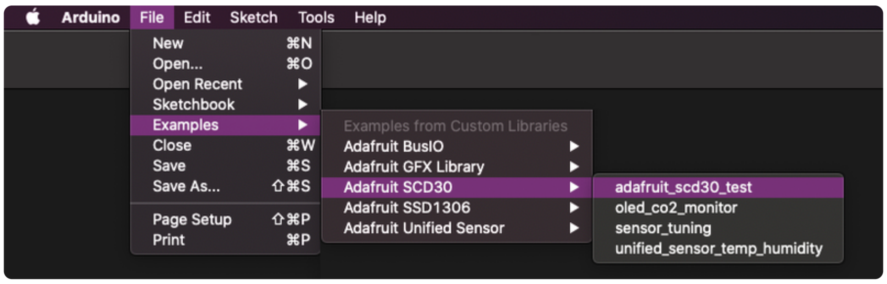

Library Manager

Type All Topic All adafruit unified sensor

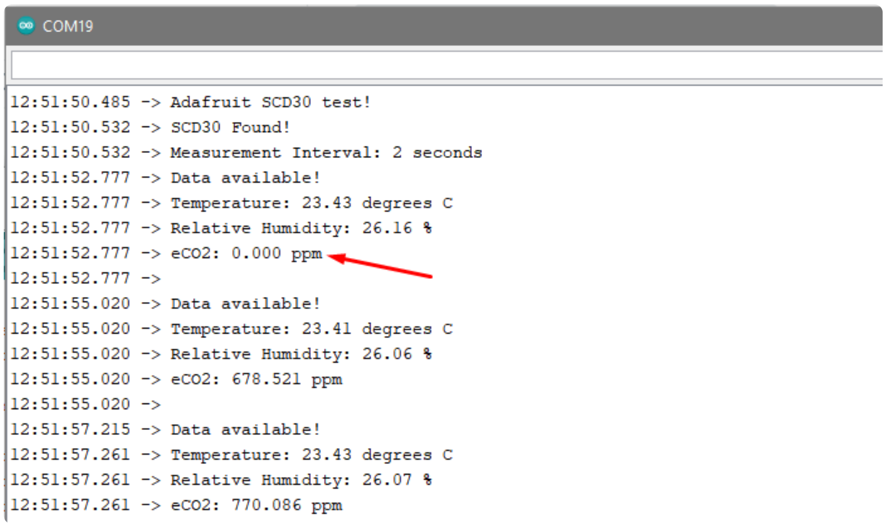
Adafruit Unified Sensor by Adafruit Version 1.0.2 INSTALLED
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

Load Example

Open up File -> Examples -> Adafruit SCD30 -> adafruit_scd30_test



After opening the demo file, upload to your Arduino wired up to the sensor. Once you upload the code, you will see the temperature, humidity and eCO2 data values being printed when you open the Serial Monitor (Tools->Serial Monitor) at 115200 baud, similar to this:


COM19

```
12:51:50.485 -> Adafruit SCD30 test!
12:51:50.532 -> SCD30 Found!
12:51:50.532 -> Measurement Interval: 2 seconds
12:51:52.777 -> Data available!
12:51:52.777 -> Temperature: 23.43 degrees C
12:51:52.777 -> Relative Humidity: 26.16 %
12:51:52.777 -> eCO2: 0.000 ppm
12:51:52.777 ->
12:51:55.020 -> Data available!
12:51:55.020 -> Temperature: 23.41 degrees C
12:51:55.020 -> Relative Humidity: 26.06 %
12:51:55.020 -> eCO2: 678.521 ppm
12:51:55.020 ->
12:51:57.215 -> Data available!
12:51:57.261 -> Temperature: 23.43 degrees C
12:51:57.261 -> Relative Humidity: 26.07 %
12:51:57.261 -> eCO2: 770.086 ppm
```

Its normal for the first CO2 reading to be 0, simply ignore the first reading when logging data.

The sensor has a lot going on, there's temperature and humidity reading thanks to an SHT31 sensor on board. These values are used internally to normalize the NDIR CO2 readings as well. You can only get data every 2 seconds, which is pretty fast for this kind of sensor! If you want to slow down the readings to reduce power usage, uncomment this section:

```
// if (!scd30.setMeasurementInterval(10)){
//   Serial.println("Failed to set measurement interval");
//   while(1){ delay(10);}
// }
```

The valid range is 2 seconds per reading up to 1800 seconds per reading.

Example Code

```
// Basic demo for readings from Adafruit SCD30
#include <Adafruit_SCD30.h>

Adafruit_SCD30  scd30;


void setup(void) {
  Serial.begin(115200);
  while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial
console opens

  Serial.println("Adafruit SCD30 test!");

  // Try to initialize!
  if (!scd30.begin()) {
    Serial.println("Failed to find SCD30 chip");
    while (1) { delay(10); }
  }
  Serial.println("SCD30 Found!");


  // if (!scd30.setMeasurementInterval(10)){
  //   Serial.println("Failed to set measurement interval");
  //   while(1){ delay(10);}
  // }
  Serial.print("Measurement Interval: ");
  Serial.print(scd30.getMeasurementInterval());
  Serial.println(" seconds");
}

void loop() {
  if (scd30.dataReady()){
    Serial.println("Data available!");

    if (!scd30.read()){ Serial.println("Error reading sensor data"); return; }

    Serial.print("Temperature: ");
    Serial.print(scd30.temperature);
    Serial.println(" degrees C");

    Serial.print("Relative Humidity: ");
    Serial.print(scd30.relative_humidity);
    Serial.println(" %");

    Serial.print("CO2: ");
    Serial.print(scd30.CO2, 3);
    Serial.println(" ppm");
    Serial.println("");
  } else {
    //Serial.println("No data");
  }

  delay(100);
}
```

Arduino Docs

Arduino Docs ()

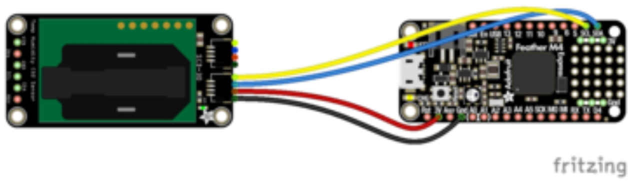
Python & CircuitPython

It's easy to use the SCD-30 with Python or CircuitPython, and the Adafruit CircuitPython SCD30 () module. This module allows you to easily write Python code that reads CO2, temperature, and humidity from the SCD30 sensor.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library ().

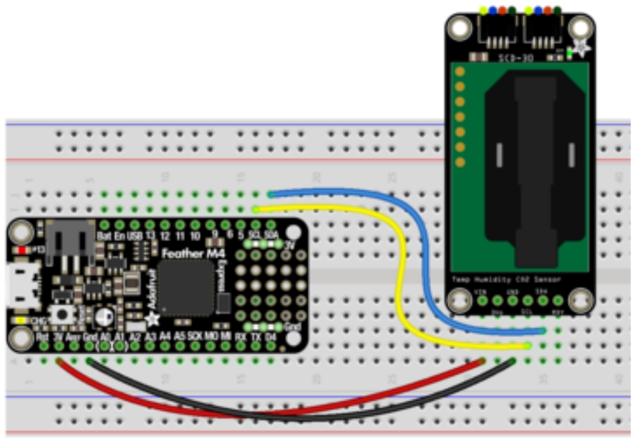
CircuitPython Microcontroller Wiring

First wire up a SCD-30 to your board exactly as shown below. Here's an example of wiring a Feather M4 to the sensor with I2C using one of the handy STEMMA QT () connectors:



Board 3V to sensor VIN (red wire)
Board GND to sensor GND (black wire)
Board SCL to sensor SCL (yellow wire)
Board SDA to sensor SDA (blue wire)

You can also use the standard 0.100" pitch headers to wire it up on a breadboard:

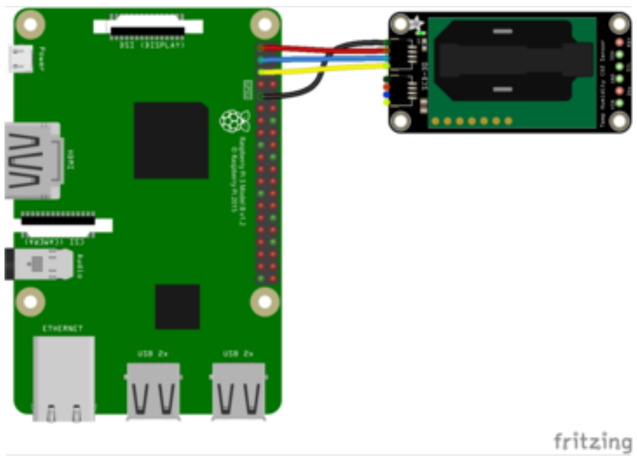


Board 3V to sensor VIN (red wire)
Board GND to sensor GND (black wire)
Board SCL to sensor SCL (yellow wire)
Board SDA to sensor SDA (blue wire)

Python Computer Wiring

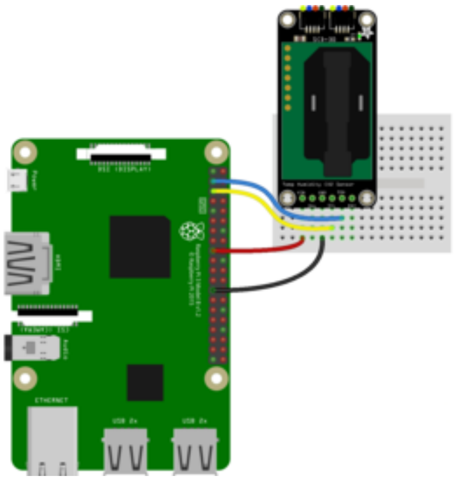
Since there's dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

Here's the Raspberry Pi wired to the sensor using I2C and a STEMMA QT () connector:



Pi 3V to sensor VIN (red wire)
Pi GND to sensor GND (black wire)
Pi SCL to sensor SCL (yellow wire)
Pi SDA to sensor SDA (blue wire)

Finally here is an example of how to wire up a Raspberry Pi to the sensor using a solderless breadboard



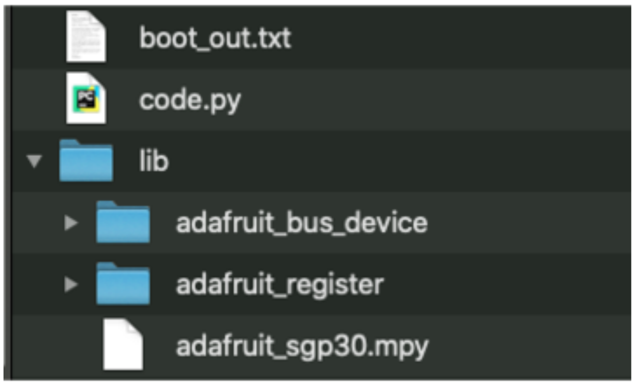
Pi 3V to sensor VIN (red wire)
Pi GND to sensor GND (black wire)
Pi SCL to sensor SCL (yellow wire)
Pi SDA to sensor SDA (blue wire)

CircuitPython Installation of SCD30 Library

You'll need to install the Adafruit CircuitPython SCD30 () library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (). Our CircuitPython starter guide has a great page on how to install the library bundle ().



Required libraries:

adafruit_scd30.mpy
adafruit_bus_device/
adafruit_register/
Your CIRCUITPY drive should look like the image.

Before continuing make sure your board's lib folder or root filesystem has the adafruit_scd30.mpy file, and the adafruit_bus_device and adafruit_register folders copied over.

Python Installation of SCD30 Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-scd30`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

Next connect to the board's serial REPL ()so you are at the CircuitPython >>> prompt.

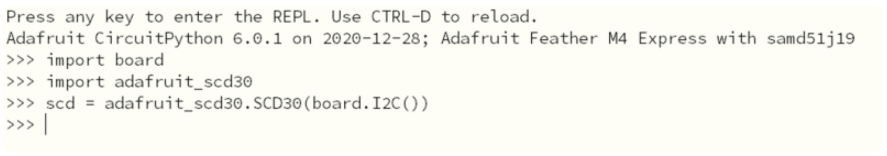
CircuitPython & Python Usage

To demonstrate the usage of the sensor we'll initialize it and read the CO2, temperature and humidity data from the board's Python REPL.

Run the following code to import the necessary modules and initialize the I2C connection with the sensor:

```
import board
import adafruit_scd30
```

```
scd = adafruit_scd30.SCD30(board.I2C())
```

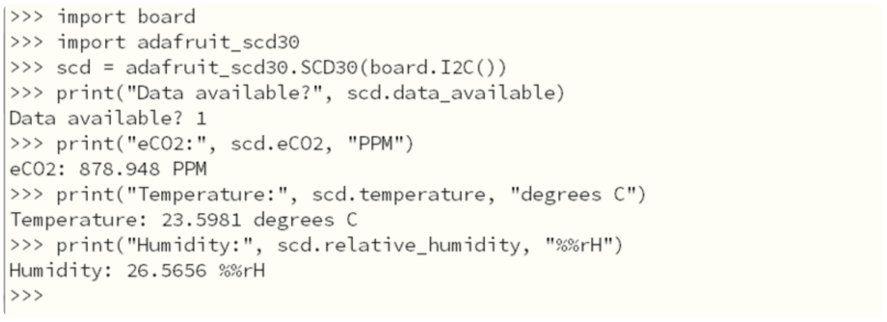


```
Press any key to enter the REPL. Use CTRL-D to reload.
Adafruit CircuitPython 6.0.1 on 2020-12-28; Adafruit Feather M4 Express with samd51j19
>>> import board
>>> import adafruit_scd30
>>> scd = adafruit_scd30.SCD30(board.I2C())
>>> 
```

Now you're ready to read values from the sensor using these properties:

- data_available - Check the sensor to see if new data is available.
- eCO2 - The CO2 concentration in PPM (parts per million).
- temperature - The current temperature in degrees Celsius.
- relative_humidity - The current relative humidity in %rH.

```
print("Data available?", scd.data_available)
print("CO2:", scd.CO2, "PPM")
print("Temperature:", scd.temperature, "degrees C")
print("Humidity:", scd.relative_humidity, "%%rH")
```



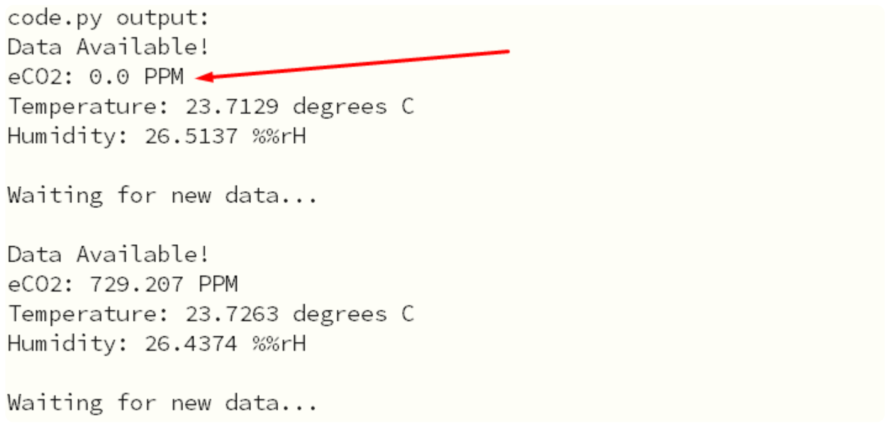
```
>>> import board
>>> import adafruit_scd30
>>> scd = adafruit_scd30.SCD30(board.I2C())
>>> print("Data available?", scd.data_available)
Data available? 1
>>> print("eCO2:", scd.eCO2, "PPM")
eCO2: 878.948 PPM
>>> print("Temperature:", scd.temperature, "degrees C")
Temperature: 23.5981 degrees C
>>> print("Humidity:", scd.relative_humidity, "%%rH")
Humidity: 26.5656 %%rH
>>>
```

Full Example Code

```
# SPDX-FileCopyrightText: 2020 by Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import busio
import adafruit_scd30

# SCD-30 has tempremental I2C with clock stretching, datasheet recommends
# starting at 50KHz
i2c = busio.I2C(board.SCL, board.SDA, frequency=50000)
scd = adafruit_scd30.SCD30(i2c)

while True:
    # since the measurement interval is long (2+ seconds) we check for new data
before reading
    # the values, to ensure current readings.
    if scd.data_available:
        print("Data Available!")
        print("CO2: %d PPM" % scd.CO2)
        print("Temperature: %0.2f degrees C" % scd.temperature)
        print("Humidity: %0.2f %% rH" % scd.relative_humidity)
        print("")
        print("Waiting for new data...")
        print("")

    time.sleep(0.5)
```

You'll be able to get a new reading every 2 seconds, that's as fast as data comes out of the sensor.

It's normal for the first reading to be 0, as the sensor 'warms up'. Simply skip that reading when logging data.



```
code.py output:
Data Available!
eCO2: 0.0 PPM
Temperature: 23.7129 degrees C
Humidity: 26.5137 %%rH

Waiting for new data...

Data Available!
eCO2: 729.207 PPM
Temperature: 23.7263 degrees C
Humidity: 26.4374 %%rH

Waiting for new data...
```

To change things like the interval delay (how often data is calculated) check out this example which shows how you can tweak the sensor to change the interval, or tune the sensor with things like the known altitude/barometric pressure. Check the datasheet for the SCD-30 for more details on tuning the sensor.

```
# SPDX-FileCopyrightText: 2020 by Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import busio
import adafruit_scd30

# SCD-30 has tempremental I2C with clock stretching, datasheet recommends
# starting at 50KHz
i2c = busio.I2C(board.SCL, board.SDA, frequency=50000)
scd = adafruit_scd30.SCD30(i2c)
# scd.temperature_offset = 10
print("Temperature offset:", scd.temperature_offset)

# scd.measurement_interval = 4
print("Measurement interval:", scd.measurement_interval)

# scd.self_calibration_enabled = True
print("Self-calibration enabled:", scd.self_calibration_enabled)

# scd.ambient_pressure = 1100
print("Ambient Pressure:", scd.ambient_pressure)

# scd.altitude = 100
print("Altitude:", scd.altitude, "meters above sea level")
```

```
# scd.forced_recalibration_reference = 409
print("Forced recalibration reference:", scd.forced_recalibration_reference)
print("")

while True:
    data = scd.data_available
    if data:
        print("Data Available!")
        print("CO2:", scd.CO2, "PPM")
        print("Temperature:", scd.temperature, "degrees C")
        print("Humidity::", scd.relative_humidity, "%%rH")
        print("")
        print("Waiting for new data...")
        print("")

    time.sleep(0.5)
```

Python Docs

Python Docs ()

WipperSnapper

What is WipperSnapper

WipperSnapper is a firmware designed to turn any WiFi-capable board into an Internet-of-Things device without programming a single line of code. WipperSnapper connects to Adafruit IO (), a web platform designed (by Adafruit! ()) to display, respond, and interact with your project's data.

Simply load the WipperSnapper firmware onto your board, add credentials, and plug it into power. Your board will automatically register itself with your Adafruit IO account.

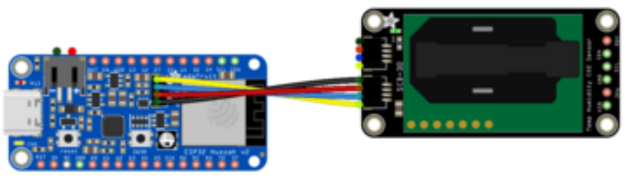
From there, you can add components to your board such as buttons, switches, potentiometers, sensors, and more! Components are dynamically added to hardware, so you can immediately start interacting, logging, and streaming the data your projects produce without writing code.

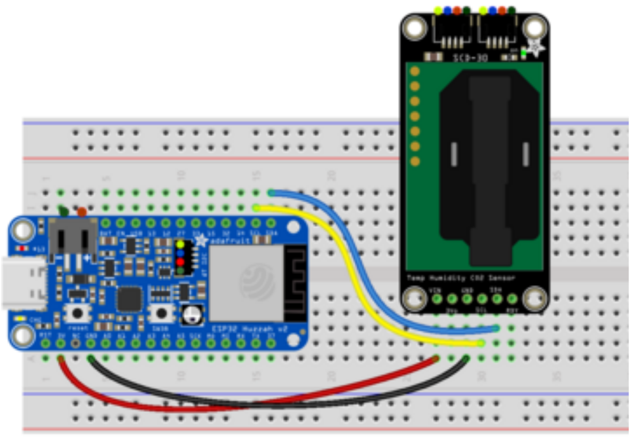
If you've never used WipperSnapper, click below to read through the quick start guide before continuing.

Quickstart: Adafruit IO WipperSnapper

Wiring

First, wire up a SCD30 to your board exactly as follows. Here is an example of the SCD30 wired to an Adafruit ESP32 Feather V2 () using I2C with a STEMMA QT cable (no soldering required) ()



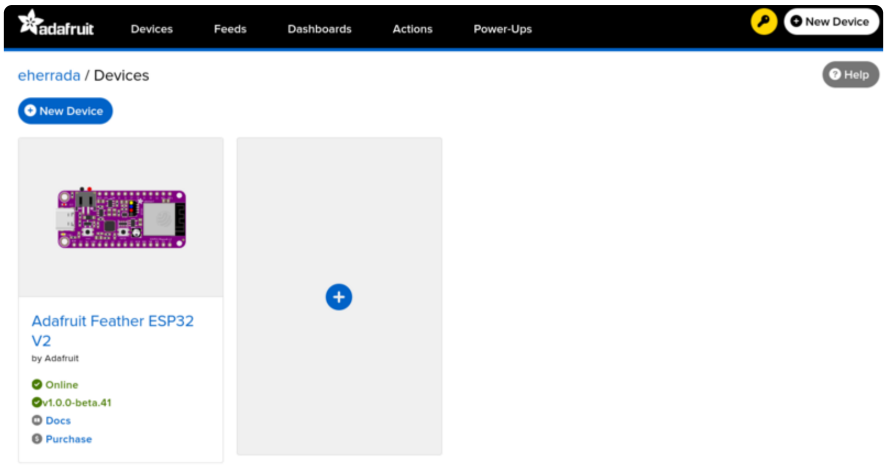


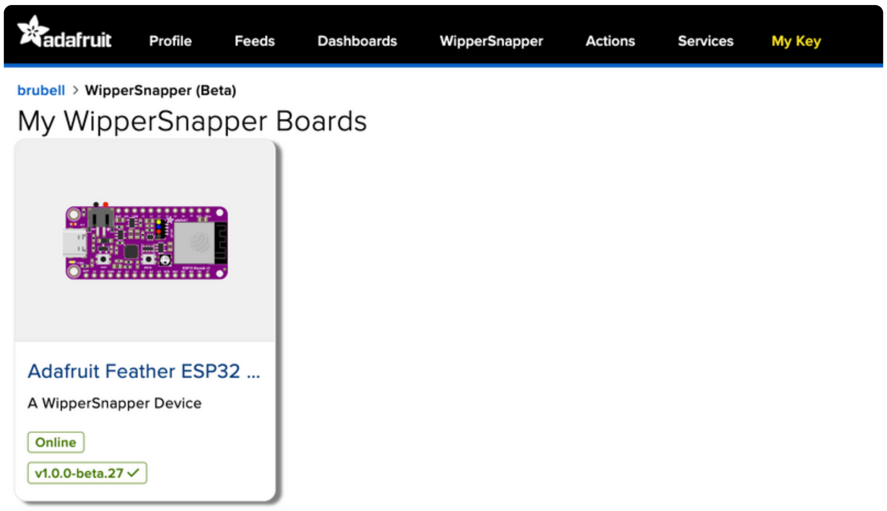
Board 3V to sensor VIN (red wire on STEMMA QT)
Board GND to sensor GND (black wire on STEMMA QT)
Board SCL to sensor SCL (yellow wire on STEMMA QT)
Board SDA to sensor SDA (blue wire on STEMMA QT)

Usage

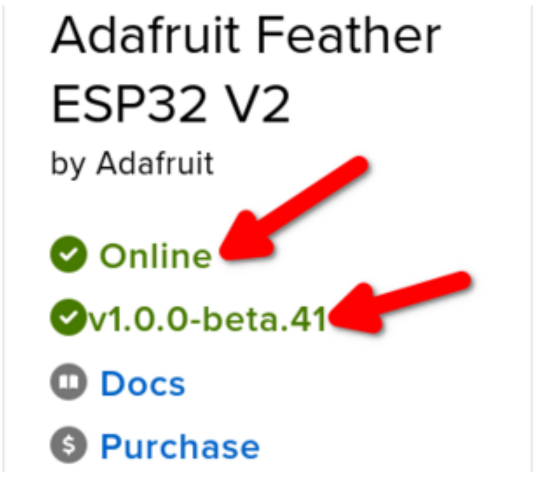
Connect your board to Adafruit IO Wippersnapper and navigate to the WipperSnapper board list ().

On this page, select the WipperSnapper board you're using to be brought to the board's interface page.





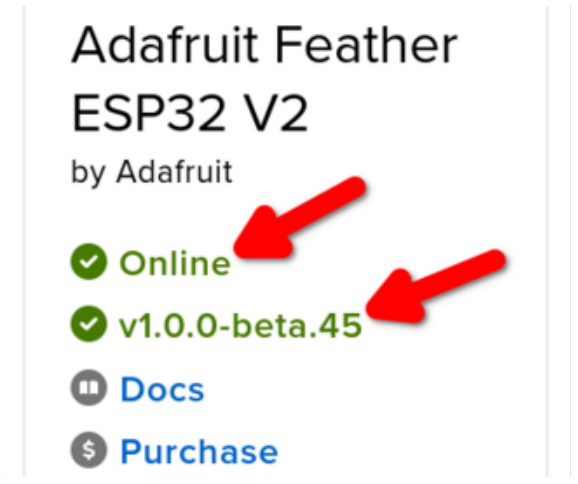
If you do not see your board listed here - you need to connect your board to Adafruit IO () first.



On the device page, quickly check that you're running the latest version of the WipperSnapper firmware.

The device tile on the left indicates the version number of the firmware running on the connected board.

If the firmware version is green with a checkmark - continue with this guide.
If the firmware version is red with an "X" - update to the latest WipperSnapper firmware () on your board before continuing.

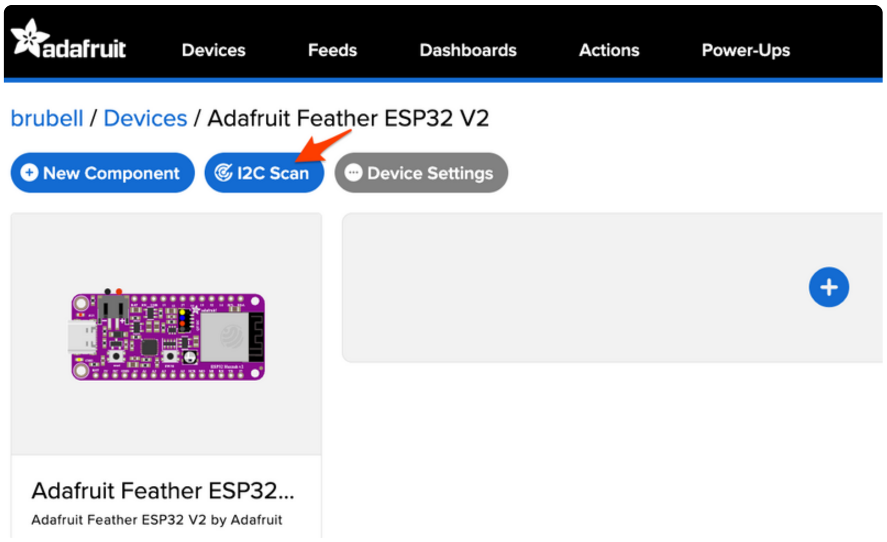


On the device page, quickly check that you're running the latest version of the WipperSnapper firmware.

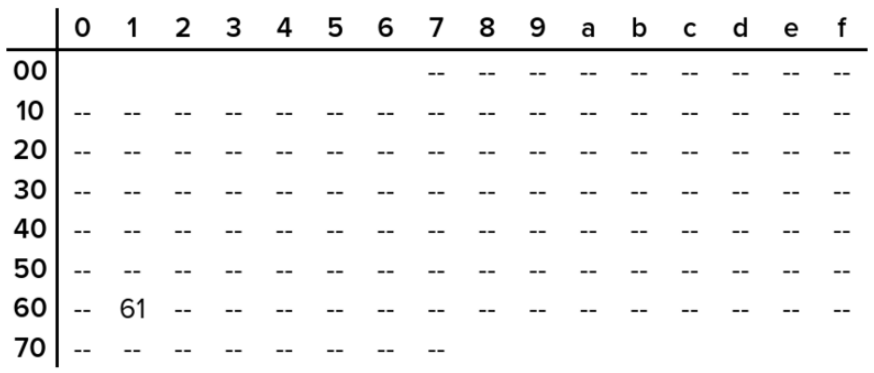
The device tile on the left indicates the version number of the firmware running on the connected board.

If the firmware version is green with a checkmark - continue with this guide.
If the firmware version is red with an "X" - update to the latest WipperSnapper firmware () on your board before continuing.

Next, make sure the sensor is plugged into your board and click the I2C Scan button.


adafruit
Devices
Feeds
Dashboards
Actions
Power-Ups
brubell / Devices / Adafruit Feather ESP32 V2
New Component
I2C Scan
Device Settings
Adafruit Feather ESP32...
Adafruit Feather ESP32 V2 by Adafruit

You should see the SCD30's default I2C address of `0x61` pop up in the I2C scan list.


```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00                       -- -- -- -- -- -- -- -- --
10  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60  -- 61 -- -- -- -- -- -- -- -- -- -- -- -- -- --
70  -- -- -- -- -- -- -- --
```

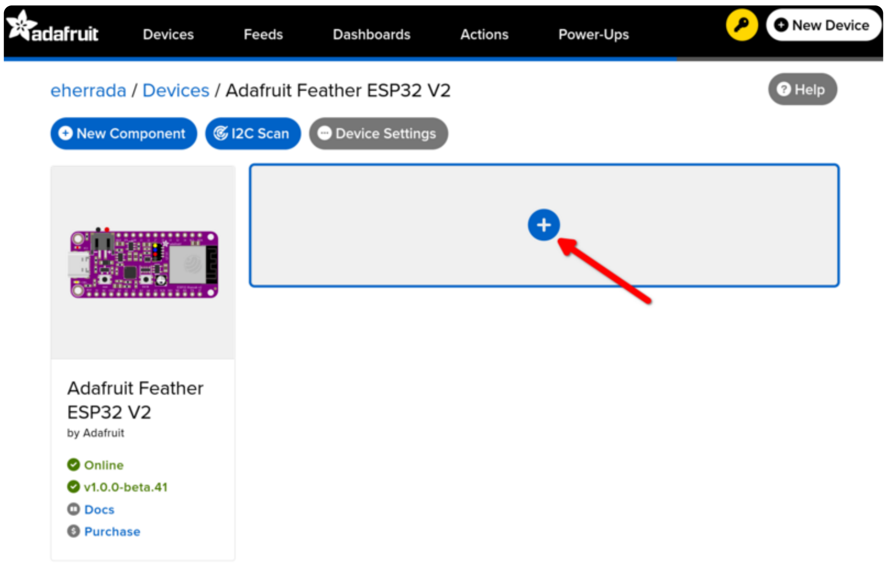
I don't see the sensor's I2C address listed!

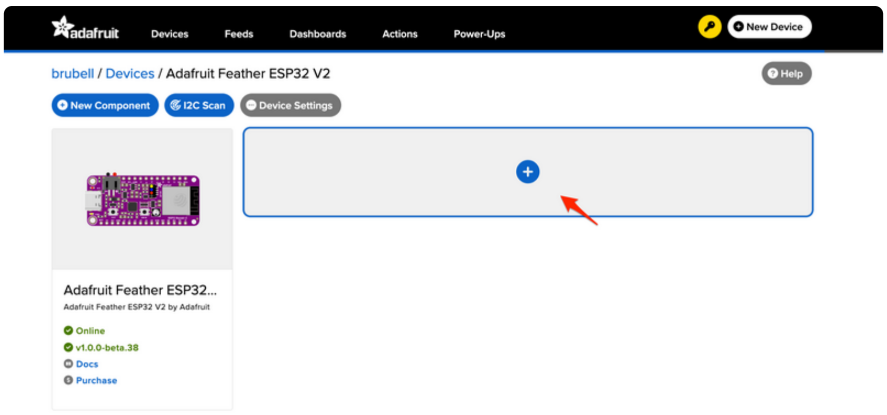
First, double-check the connection and/or wiring between the sensor and the board.

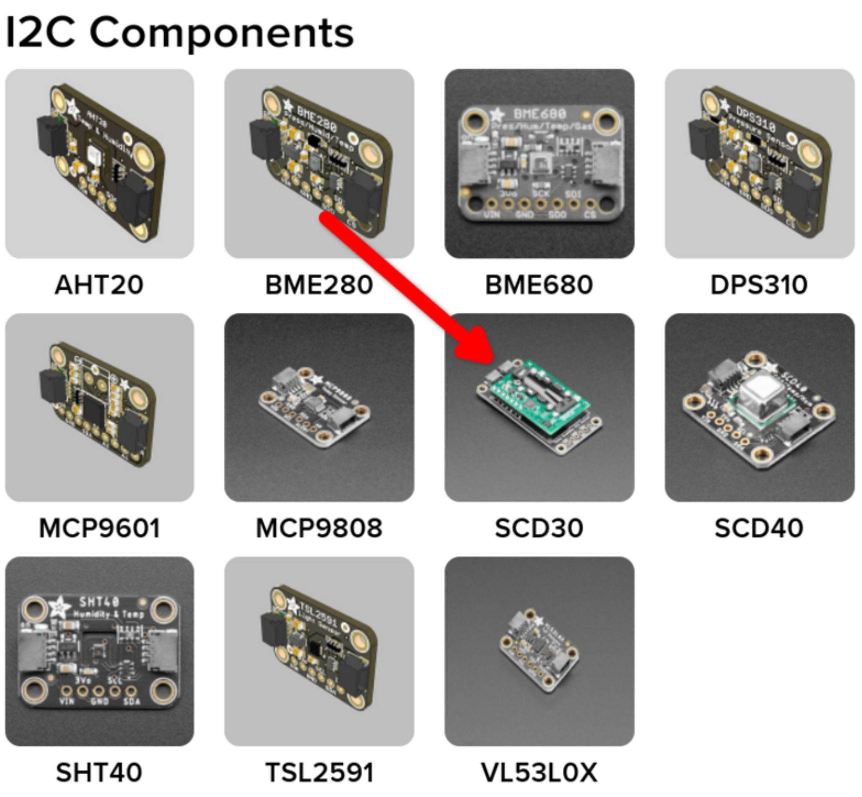
Then, reset the board and let it re-connect to Adafruit IO WipperSnapper.

With the sensor detected in an I2C scan, we're ready to add the sensor to your board.

Click the New Component button or the + button to bring up the component picker.



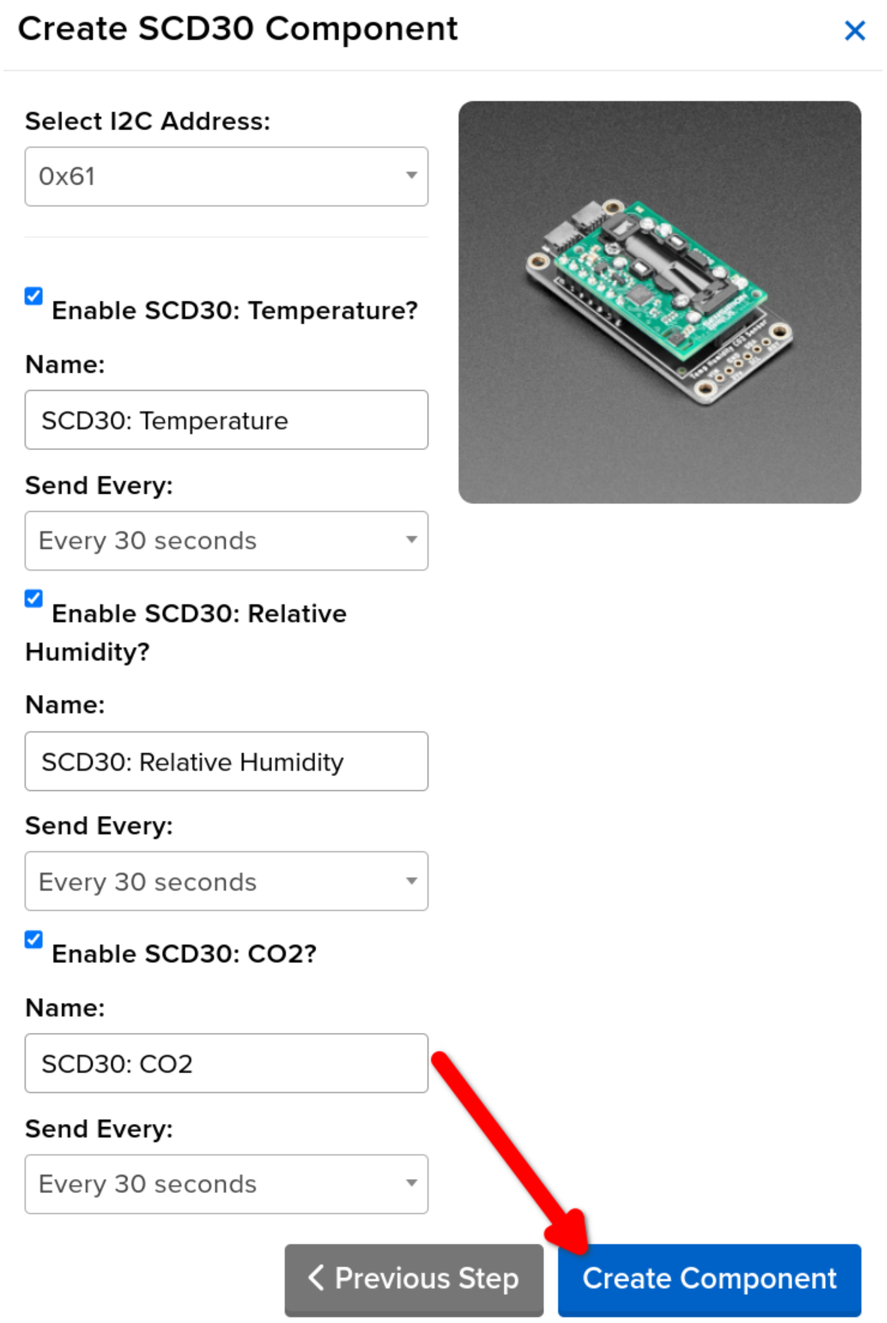




On the component configuration page, the SCD30's sensor address should be listed along with the sensor's settings.

The Send Every option is specific to each sensor's measurements. This option will tell the Feather how often it should read from each of the SCD30's three sensors and send the data to Adafruit IO. Measurements can range from every 30 seconds to every 24 hours.

For this example, set the Send Every interval for each sensor to every 30 seconds.


Create SCD30 Component

Select I2C Address: 0x61

[✓] Enable SCD30: Temperature?

Name: SCD30: Temperature

Send Every: Every 30 seconds

[✓] Enable SCD30: Relative Humidity?

Name: SCD30: Relative Humidity

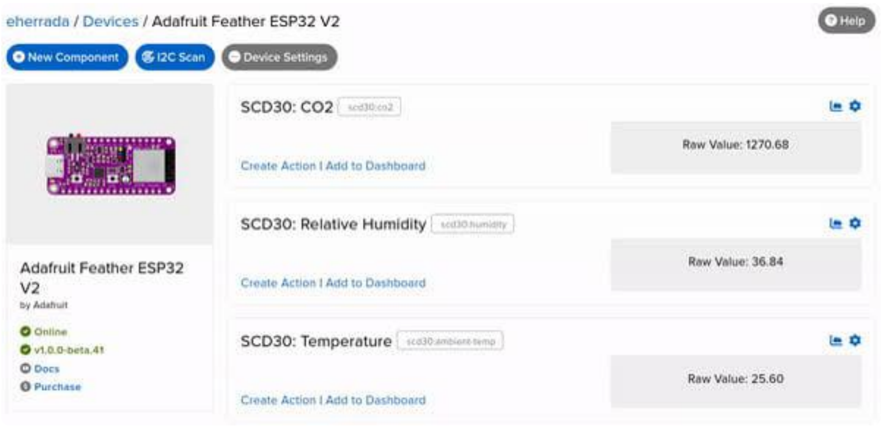
Send Every: Every 30 seconds

[✓] Enable SCD30: CO2?

Name: SCD30: CO2

Send Every: Every 30 seconds

Your device interface should now show the sensor components you created. After the interval you configured elapses, WipperSnapper will automatically read values from the sensor(s) and send them to Adafruit IO.

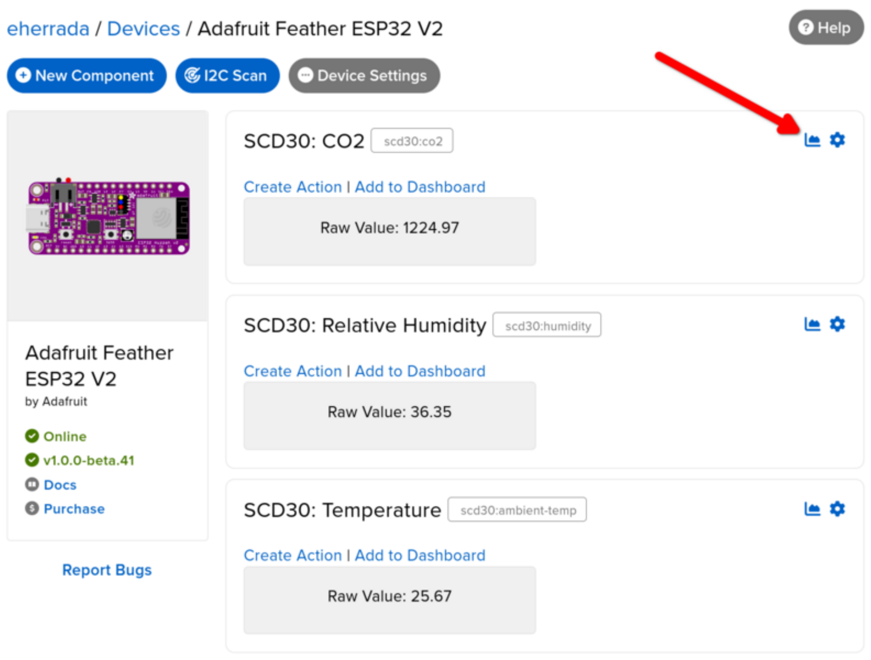

eherrada / Devices / Adafruit Feather ESP32 V2

SCD30: CO2 scd30:co2 — Raw Value: 1270.68

SCD30: Relative Humidity scd30:humidity — Raw Value: 36.84

SCD30: Temperature scd30:ambient-temp — Raw Value: 25.60

To view the data that has been logged from the sensor, click on the graph next to the sensor name.


eherrada / Devices / Adafruit Feather ESP32 V2

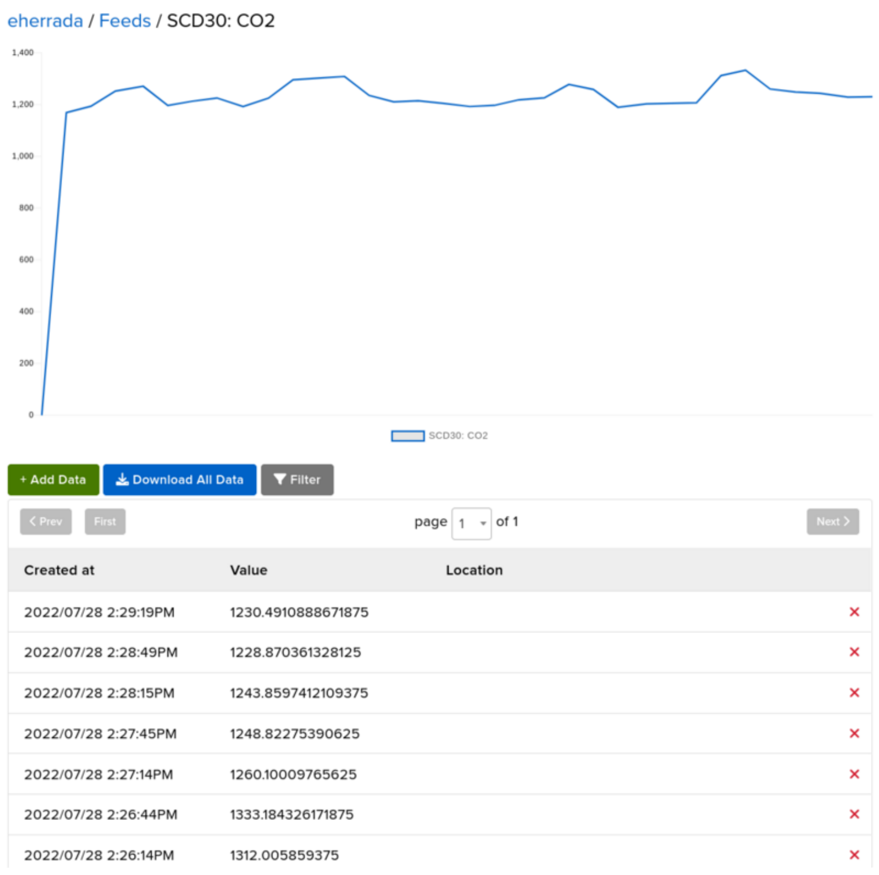
SCD30: CO2 scd30:co2 — Raw Value: 1224.97

SCD30: Relative Humidity scd30:humidity — Raw Value: 36.35

SCD30: Temperature scd30:ambient-temp — Raw Value: 25.67

Here you can see the feed history and edit things about the feed such as the name, privacy, webhooks associated with the feed and more. If you want to learn more about how feeds work, check out this page ().

The SCD30 has three sensors that each have their own feeds. In this picture, we're looking at the CO2 sensor, but if you click on the graph icon for the different sensors you'll see their feed history.


eherrada / Feeds / SCD30: CO2

Created at	Value	Location	
2022/07/28 2:29:19PM	1230.4910888671875		✕
2022/07/28 2:28:49PM	1228.870361328125		✕
2022/07/28 2:28:15PM	1243.8597412109375		✕
2022/07/28 2:27:45PM	1248.82275390625		✕
2022/07/28 2:27:14PM	1260.10009765625		✕
2022/07/28 2:26:44PM	1333.184326171875		✕
2022/07/28 2:26:14PM	1312.005859375		✕

For IO Free accounts, feed data is stored for a maximum of 30 days and there's a maximum of 10 feeds. In this guide, you created three feeds (one for each of the SCD30's sensors). If you'd like to store data for more than 30 days, increase the number of feeds (components) you can use with WipperSnapper, or increase your data rate to send more sensor measurements to Adafruit IO - upgrade your account to Adafruit IO Plus ().

Field Calibration

Performing a re-calibration of the SCD-30 can help maintain accurate CO2 readings over time. Various factors can cause the SCD-30 sensor reading to drift and there are two available re-calibration options: Forced Re-Calibration (FRC) and Automatic Self-Calibration (ASC).

This Application Note from Sensirion () goes into lots of detail and is worth reading:

Field calibration for SCD30

Here we summarizes the two approaches.

Forced Re-Calibration

This is the easiest approach. The SCD-30 is placed in an environment with a known CO2 concentration. Then the FRC routine is called and this known concentration value (in ppm) is supplied. But how do you come up with that known value? That is a caveat of this approach and Sensirion (see PDF linked above) suggests three approaches:

1. Using a separate secondary calibrated CO2 sensor to provide the value.
2. Exposing the SCD-30 to a controlled environment with a known value.
3. Exposing the SCD-30 to fresh outside air and using a value of 400 ppm.

However, once you have your reference value, performing a FRC is super easy. Assuming a reference CO2 concentration of 800 ppm has been determined, then with the CircuitPython library use:

```
scd30.forced_recalibration_reference = 800
```

or with the Arduino library use:

```
scd30.forceRecalibrationWithReference(800);
```

Automatic Self-Calibration

Hey, automatic! That sounds great! Set and forget, right? Well, not so fast. The ASC feature has some requirements which should be considered to determine if it is suitable for any given end use application. If the conditions can not be met, then the FRC mentioned above should be used.

1. The SCD-30 should regularly be exposed to fresh air with CO2 concentration of 400 ppm.
2. The SCD-30 needs to operate in continuous mode, i.e. do not power it down.
3. The ASC needs 7 good readings separated by at least 18 hours (that's ~5 days).

See the PDF linked above for many more details. If you want to use ASC, enabling it is very simple. In CircuitPython use:

```
scd30.self_calibration_enabled = True
```

or with the Arduino library use:

```
scd30.selfCalibrationEnabled(true);
```

With either, simply use `False` / `false` to disable ASC.

FRC vs. ASC

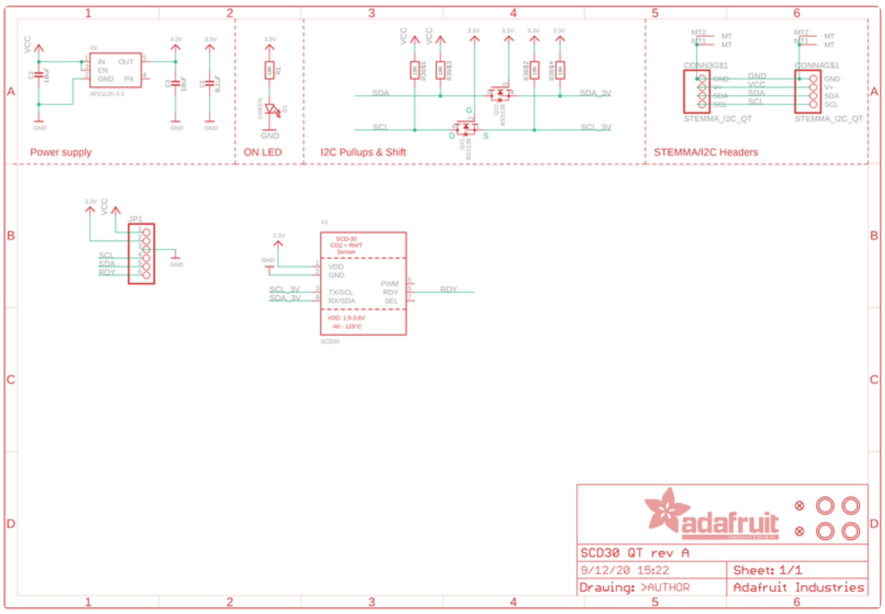
Both the Forced Re-Calibration (FRC) and Automatic Self-Calibration (ASC) are ways of arriving at the same "Reference Value" which is then used in determining the CO2 ppm reading reported by the SCD-30. With the FRC approach, the Reference Value is specified. With the ASC approach, the Reference Value is determined algorithmically. Either one will overwrite the Reference Value from the other one. For example, running a FRC will immediately change to the new Reference Value. However, if ASC is enabled, then it may replace the Reference Value at a later time.

Downloads

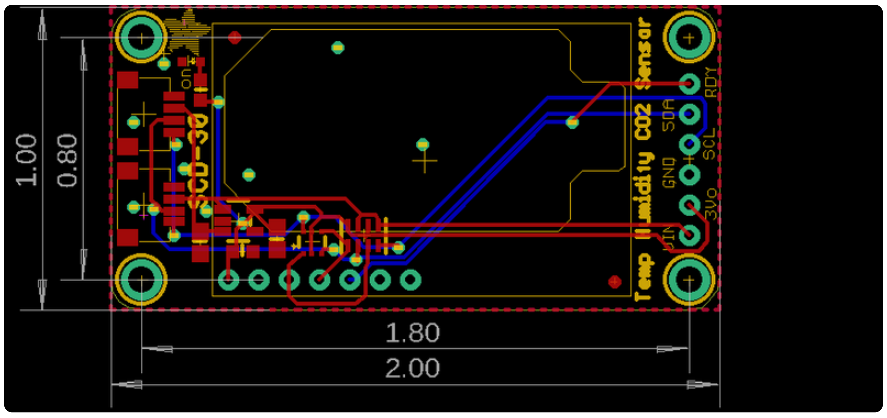
Files:

- SCD-30 Datasheet ()
- SCD-30 Design-in Guidelines ()
- SCD-30 Interface Description ()
- Fritzing object in the Adafruit Fritzing Library ()
- EagleCAD PCB files on GitHub ()
- 3D models on GitHub ()

Schematic



Fab Print


1.00
0.80
1.80
2.00
SCD-30
Temp Humidity CO2 Sensor
RDY
SDA
SCL
GND
3Vo
VIN