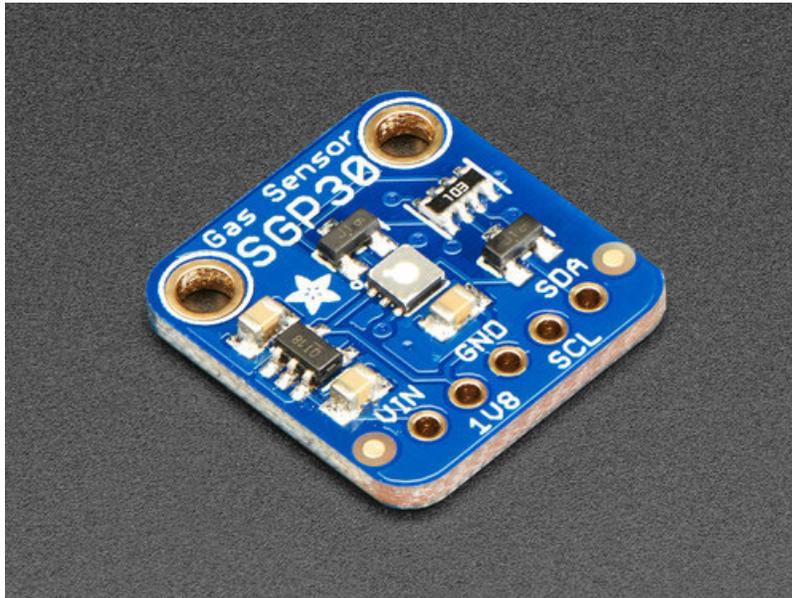


Adafruit SGP30 TVOC/eCO2 Gas Sensor

Created by lady ada

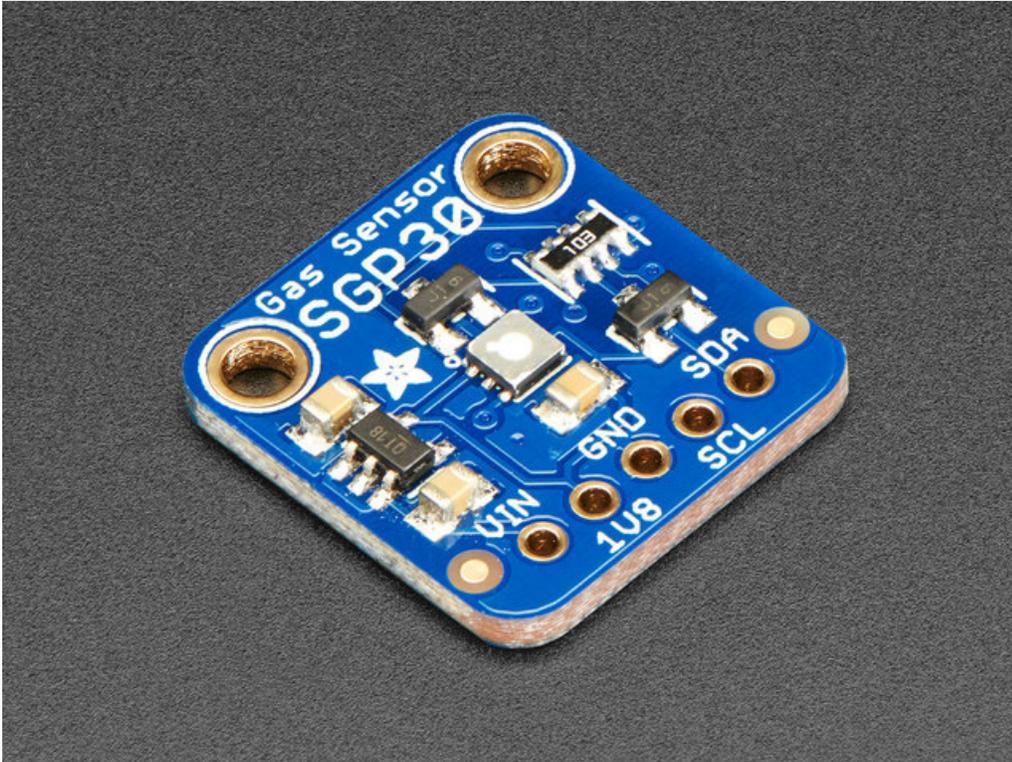


Last updated on 2018-04-03 04:03:06 PM UTC

Guide Contents

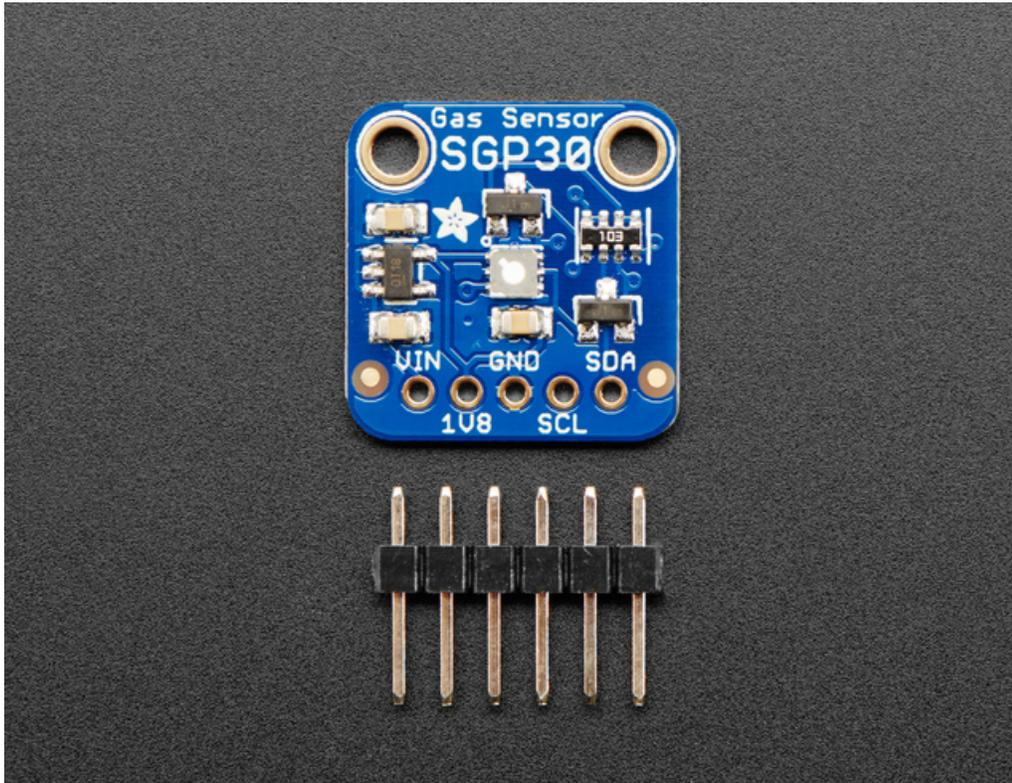
Guide Contents	2
Overview	3
Pinouts	6
Power Pins:	6
Data Pins	6
Wiring	7
Parts	7
Wiring	7
Arduino Code	9
Install Adafruit_SGP30 library	9
Load Demo	9
Baseline Set & Get	11
Arduino Library Docs	13
CircuitPython Wiring & Test	14
Installing Library	14
Usage	14
Baseline Set & Get	16
Download	18
Files:	18
Schematic & Fabrication Print	18

Overview



Breathe easy with the SGP30 Multi-Pixel Gas Sensor, a fully integrated MOX gas sensor. This is a very fine air quality sensor from the sensor experts at Sensirion, with I2C interfacing and fully calibrated output signals with a typical accuracy of 15% within measured values. The SGP combines multiple metal-oxide sensing elements on one chip to provide more detailed air quality signals.

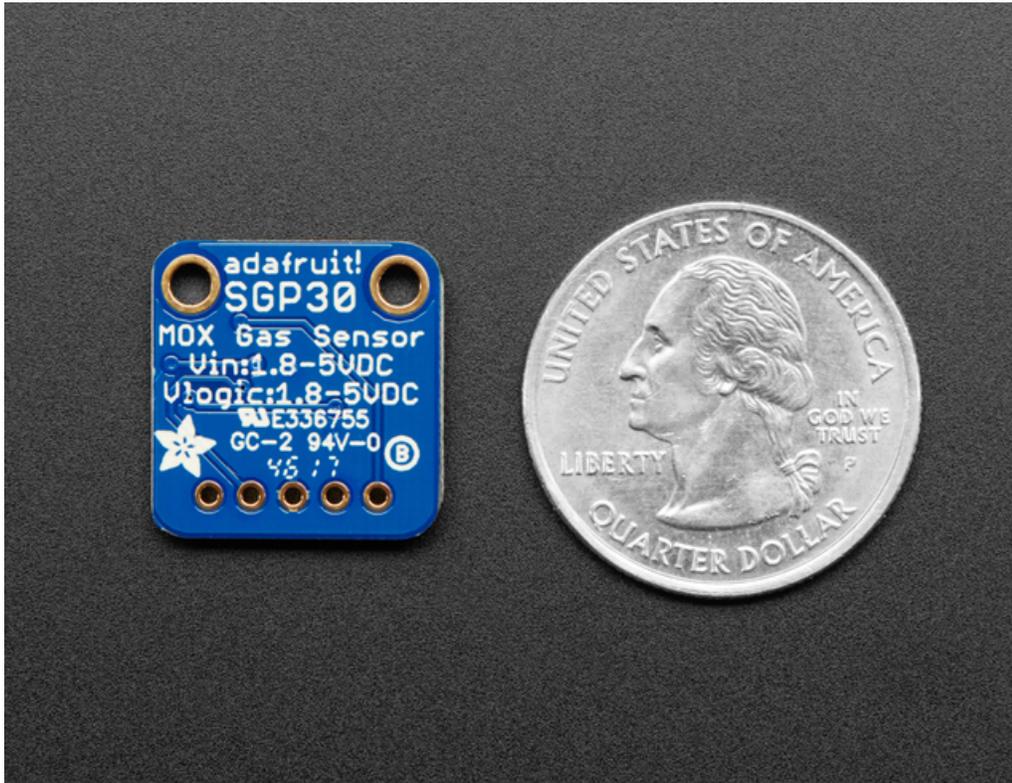
This is a gas sensor that can detect a wide range of Volatile Organic Compounds (VOCs) and H₂ and is intended for indoor air quality monitoring. When connected to your microcontroller (running our library code) it will return a Total Volatile Organic Compound (TVOC) reading and an equivalent carbon dioxide reading (eCO₂) over I2C.



The SGP30 has a 'standard' hot-plate MOX sensor, as well as a small microcontroller that controls power to the plate, reads the analog voltage, tracks the baseline calibration, calculates TVOC and eCO₂ values, and provides an I2C interface to read from. Unlike the CCS811, this sensor does not require I2C clock stretching.

This part will measure **eCO₂** (equivalent calculated carbon-dioxide) concentration within a range of 0 to 60,000 parts per million (ppm), and **TVOC** (Total Volatile Organic Compound) concentration within a range of 0 to 60,000 parts per billion (ppb).

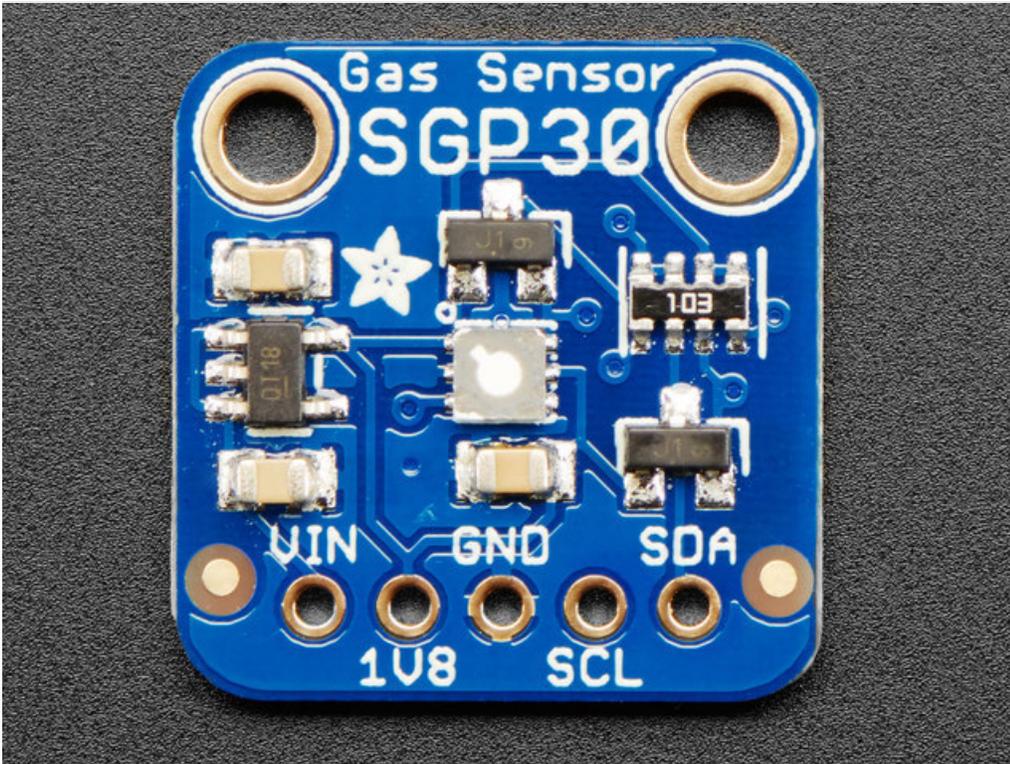
Please note, this sensor, like all VOC/gas sensors, has variability and to get precise measurements you will want to calibrate it against known sources! That said, for general environmental sensors, it will give you a good idea of trends and comparison. The SGP30 does have built in calibration capabilities, note that eCO₂ is calculated based on H₂ concentration, it is not a 'true' CO₂ sensor for laboratory use.



Another nice element to this sensor is the ability to set humidity compensation for better accuracy. An external humidity sensor is required and then the RH% is written over I2C to the sensor, so it can better calculate the TVOC/eCO2 values.

For your convenience we've pick-and-placed the sensor on a PCB with a 1.8V regulator and some level shifting so it can be easily used with your favorite 3.3V or 5V microcontroller.

Pinouts



Power Pins:

- **Vin** - this is the power pin. Since the sensor chip uses 3 VDC for logic, 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 micro like Arduino, use 5V
- **1V8** - this is the 1.8V output from the voltage regulator, you can grab up to 50mA from this if you like
- **GND** - common ground for power and logic

Data Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. Can use 3V or 5V logic, and has a 10K pullup to **Vin**
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. Can use 3V or 5V logic, and has a 10K pullup to **Vin**

Wiring

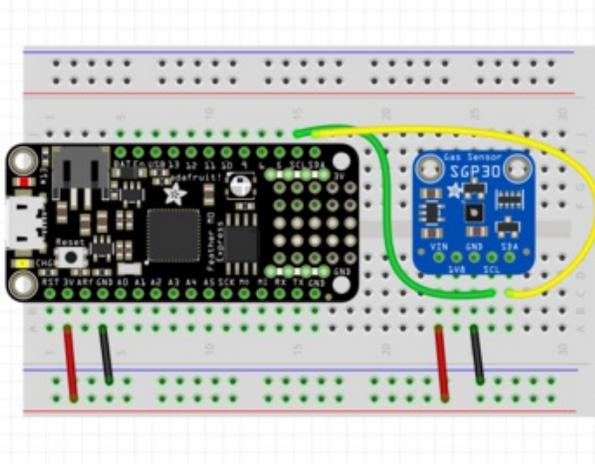
Parts

Start by soldering the headers to the SGP30 breakout board. Check out the [Adafruit guide to excellent soldering](#) if you're new to soldering. Then continue on below to learn how to wire it to a Feather or Metro

The sensor uses **I2C address 0x58** and cannot be changed

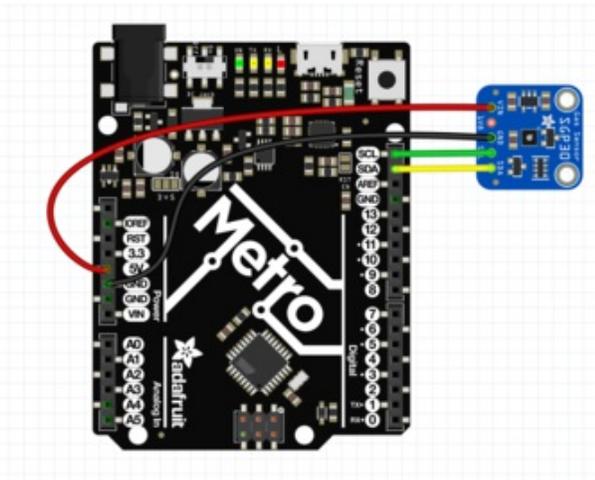
Wiring

Connect the SGP30 breakout to your board using an I2C connection. Here's an example with a Feather MO:



- **Board 3.3V to sensor Vin** (Feather is 3.3V logic)
- **Board ground / GND to sensor ground / GND.**
- **Board SCL to sensor SCL.**
- **Board SDA to sensor SDA.**

Or Arduino-compatible:



- **Board 5V to Sensor Vin.** (Metro is a 5V logic chip)
- **Board ground / GND to sensor ground / GND.**
- **Board SCL to sensor SCL.**
- **Board SDA to sensor SDA.**

Feather Fritzing file

<https://adafru.it/At8>

Metro Fritzing file

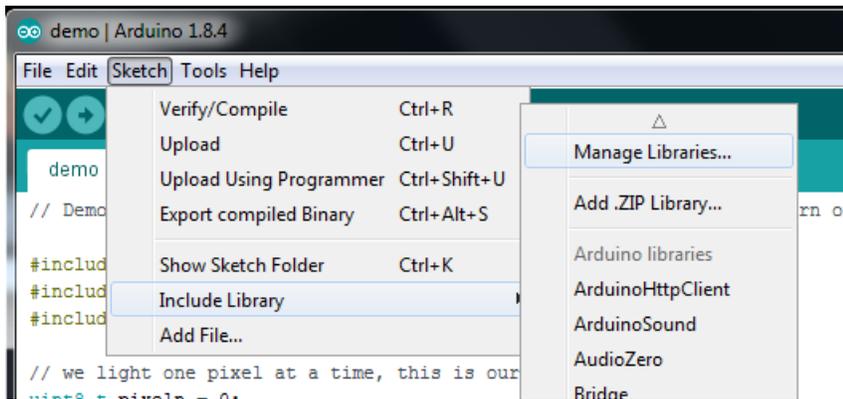
<https://adafru.it/At6>

Arduino Code

Install Adafruit_SGP30 library

To begin reading sensor data, you will need to [install the Adafruit_SGP30 library \(code on our github repository\)](#). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...

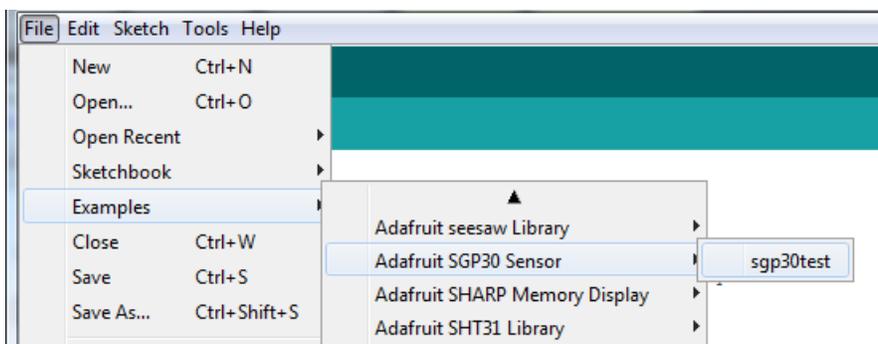


And type in `adafruit sgp30` to locate the library. Click **Install**

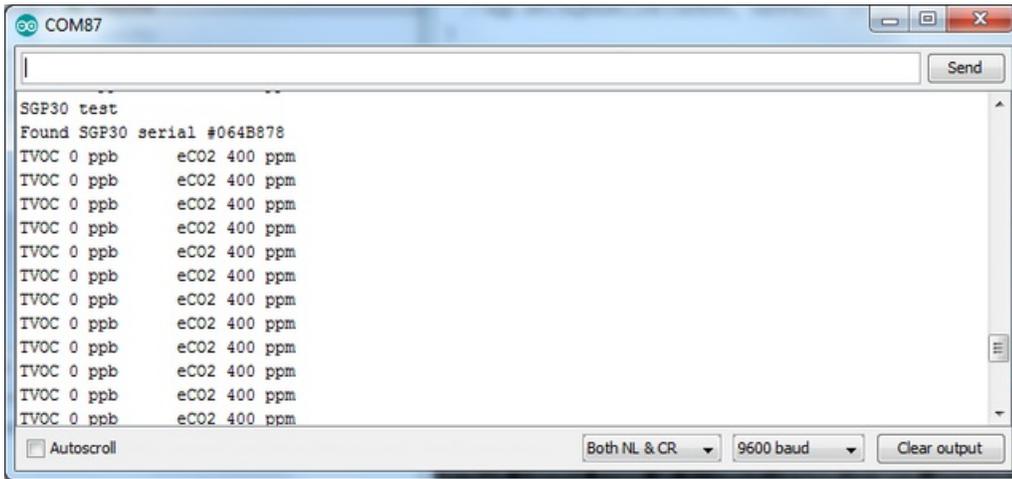


Load Demo

Open up `File->Examples->Adafruit_SGP30->sgp30test` and upload to your microcontroller wired up to the sensor

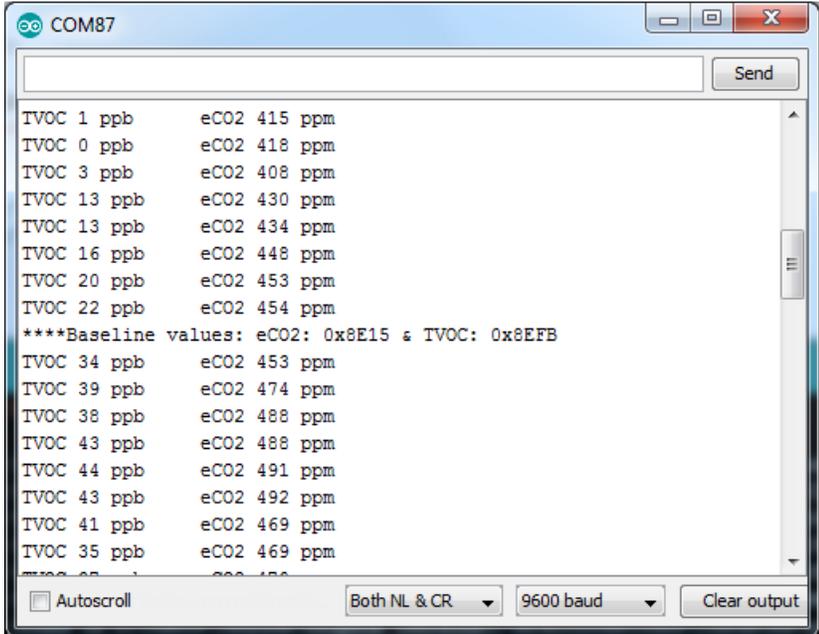


Then open up the serial console at 9600 baud, you'll see the **serial number** printed out - this is a unique 48-bit number burned into each chip. Since you may want to do per-chip calibration, this can help keep your calibration detail separate



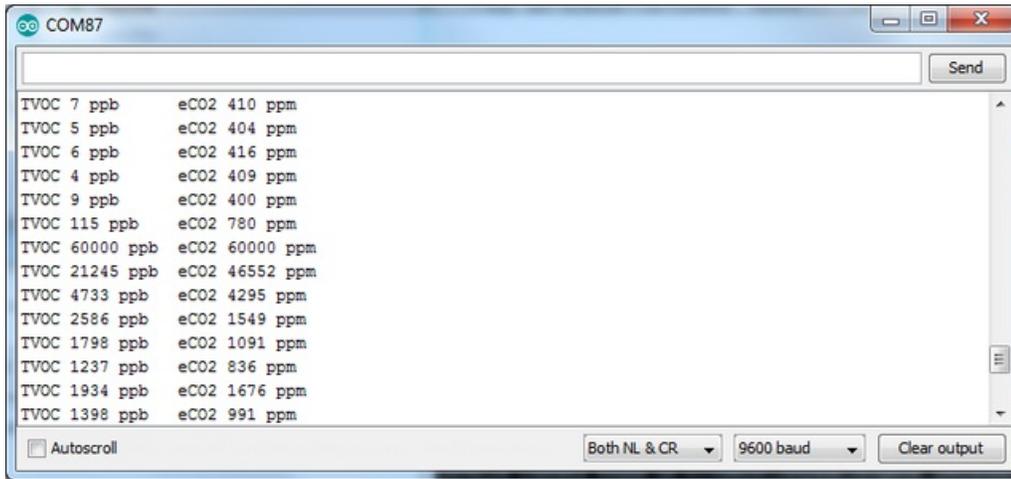
The first 10-20 readings will always be **TVOC 0 ppb eCO2 400 ppm** . That's because the sensor is warming up, so it will have 'null' readings.

After a few seconds, you will see the TVOC and eCO2 readings fluctuate:



Every minute or so you'll also get a **Baseline value** printed out. More about that later!

You can take a bit of alcohol on a swap and swipe it nearby to get the readings to spike



That's it! The sensor pretty much only does that - all the calculations for the TVOC and eCO2 are done within the sensor itself, no other data is exposed beyond the 'baseline' values

Baseline Set & Get

All VOC/gas sensors use the same underlying technology: a tin oxide element that, when exposed to organic compounds, changes resistance. The 'problem' with these sensors is that the baseline changes, often with humidity, temperature, and other non-gas-related-events. To keep the values coming out reasonable, you'll need to calibrate the sensor.

If no stored baseline is available after initializing the baseline algorithm, the sensor has to run for 12 hours until the baseline can be stored. This will ensure an optimal behavior for preceding startups. Reading out the baseline prior should be avoided unless a valid baseline is restored first. Once the baseline is properly initialized or restored, the current baseline value should be stored approximately once per hour. While the sensor is off, baseline values are valid for a maximum of seven days.

Restarting the sensor without reading back a previously stored baseline will result in the sensor trying to determine a new baseline. The adjustment algorithm will be accelerated for 12hrs which is the Maximum time required to find a new baseline.

The sensor adjusts to the best value it has been exposed to. So keeping it indoors the sensor thinks this is the best value and sets it to ~0ppb tVOC and 400ppm CO2eq. As soon as you expose the sensor to outside air it can adjust to the global H2 Background Signal. For normal Operation exposing the sensor to outside air for 10min cumulative time should be sufficient.

If you're experienced with sensors that don't have a baseline, you either won't be able to measure absolute values or you'll have to implement your own baseline algorithm.

The sensor to sensor variation of SGP30 in terms of sensitivity is very good as each of them is calibrated. But the baseline has to be determined for each sensor individually during the first Operation.

To make that easy, SGP lets you query the 'baseline calibration readings' from the sensor with code like this:

```
uint16_t TVOC_base, eCO2_base;
sgp.getIAQBaseline(&eCO2_base, &TVOC_base);
```

This will grab the two 16-bit sensor calibration words and place them in the variables so-named.

You should store these in EEPROM, FLASH or hard-coded. Then, next time you start up the sensor, you can pre-fill the calibration words with `sgp.setIAQBaseline(eCO2_baseline, TVOC_baseline);`

CircuitPython Wiring & Test

Installing Library

To use the SGP30 you'll need to install the [Adafruit CircuitPython SGP30](#) 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 introduction guide has [a great page on how to install the library bundle](#) for both express and non-express boards.

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- `adafruit_sgp30.mpy`
- `adafruit_bus_device`

You can also download the `adafruit_sgp.mpy` from [its releases page on Github](#).

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_sgp30.mpy`, and `adafruit_bus_device` files and folders copied over.

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

Usage

To demonstrate the usage of the sensor we'll initialize it and read the eCO2 and TVOC data and print it to the REPL

Save this example sketch as `main.py` on your CircuitPython board:

```

""" Example for using the SGP30 with CircuitPython and the Adafruit library"""

import time
import board
import busio
import adafruit_sgp30

i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)

# Create library object on our I2C port
sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)

print("SGP30 serial #", [hex(i) for i in sgp30.serial])

sgp30.iaq_init()
sgp30.set_iaq_baseline(0x8973, 0x8aae)

elapsed_sec = 0

while True:
    co2eq, tvoc = sgp30.iaq_measure()
    print("CO2eq = %d ppm \t TVOC = %d ppb" % (co2eq, tvoc))
    time.sleep(1)
    elapsed_sec += 1
    if elapsed_sec > 10:
        elapsed_sec = 0
        co2eq_base, tvoc_base = sgp30.get_iaq_baseline()
        print("**** Baseline values: CO2eq = 0x%x, TVOC = 0x%x" % (co2eq_base, tvoc_base))

```

In the REPL you'll see the **serial number** printed out: `[0x0, 0x64, 0xb878]` in this case. This is a unique 48-bit number burned into each chip. Since you may want to do per-chip calibration, this can help keep your calibration detail separate

```

Adafruit CircuitPython REPL
main.py:6: Output:
SGP30 serial # ['0x0', '0x64', '0xb878']
CO2eq = 400 ppm      TVOC = 0 ppb
**** Baseline values: CO2eq = 0x8973, TVOC = 0x8aae
CO2eq = 400 ppm      TVOC = 0 ppb
CO2eq = 400 ppm      TVOC = 0 ppb
CO2eq = 400 ppm      TVOC = 0 ppb
CO2eq = 400 ppm      TVOC = 774 ppb
CO2eq = 400 ppm      TVOC = 705 ppb
CO2eq = 400 ppm      TVOC = 682 ppb
CO2eq = 400 ppm      TVOC = 649 ppb
CO2eq = 400 ppm      TVOC = 639 ppb

```

The first 10-20 readings will always be `eCO2 400 ppm` `TVOC 0 ppb`. That's because the sensor is warming up, so it will have 'null' readings.

After a few seconds, you will see the TVOC and eCO2 readings fluctuate

Every minute or so you'll also get a **Baseline value** printed out. More about that later!

You can take a bit of alcohol on a swap and swipe it nearby to get the readings to spike

```
CO2eq = 400 ppm      TVOC = 0 ppb
**** Baseline values: CO2eq = 0x8973, TVOC = 0x8ab2
CO2eq = 400 ppm      TVOC = 0 ppb
CO2eq = 60000 ppm    TVOC = 60000 ppb
CO2eq = 38277 ppm    TVOC = 13302 ppb
CO2eq = 3208 ppm     TVOC = 3078 ppb
CO2eq = 994 ppm      TVOC = 1638 ppb
CO2eq = 415 ppm      TVOC = 799 ppb
CO2eq = 400 ppm      TVOC = 663 ppb
CO2eq = 400 ppm      TVOC = 666 ppb
```

That's it! The sensor pretty much only does that - all the calculations for the TVOC and eCO2 are done within the sensor itself, no other data is exposed beyond the 'baseline' values

Baseline Set & Get

All VOC/gas sensors use the same underlying technology: a tin oxide element that, when exposed to organic compounds, changes resistance. The 'problem' with these sensors is that the baseline changes, often with humidity, temperature, and other non-gas-related-events. To keep the values coming out reasonable, you'll need to calibrate the sensor.

If no stored baseline is available after initializing the baseline algorithm, the sensor has to run for 12 hours until the baseline can be stored. This will ensure an optimal behavior for preceding startups. Reading out the baseline prior should be avoided unless a valid baseline is restored first. Once the baseline is properly initialized or restored, the current baseline value should be stored approximately once per hour. While the sensor is off, baseline values are valid for a maximum of seven days.

Restarting the sensor without reading back a previously stored baseline will result in the sensor trying to determine a new baseline. The adjustment algorithm will be accelerated for 12hrs which is the Maximum time required to find a new baseline.

The sensor adjusts to the best value it has been exposed to. So keeping it indoors the sensor thinks this is the best value and sets it to ~0ppb tVOC and 400ppm CO2eq. As soon as you expose the sensor to outside air it can adjust to the global H2 Background Signal. For normal Operation exposing the sensor to outside air for 10min cumulative time should be sufficient.

If you're experienced with sensors that don't have a baseline, you either won't be able to measure absolute values or you'll have to implement your own baseline algorithm.

The sensor to sensor variation of SGP30 in terms of sensitivity is very good as each of them is calibrated. But the baseline has to be determined for each sensor individually during the first Operation.

To make that easy, SGP lets you query the 'baseline calibration readings' from the sensor with code like this:

```
co2eq_base, tvoc_base = sgp30.get_iaq_baseline()
```

This will grab the two 16-bit sensor calibration words and place them in the variables so-named.

You should store these in EEPROM, FLASH or hard-coded. Then, next time you start up the sensor, you can pre-fill the calibration words with `sgp30.set_iaq_baseline(co2eq_base, tvoc_base)`

Download

Files:

- [SGP30 Datasheet](#)
- [Fritzing object in Adafruit Fritzing library](#)

Schematic & Fabrication Print

