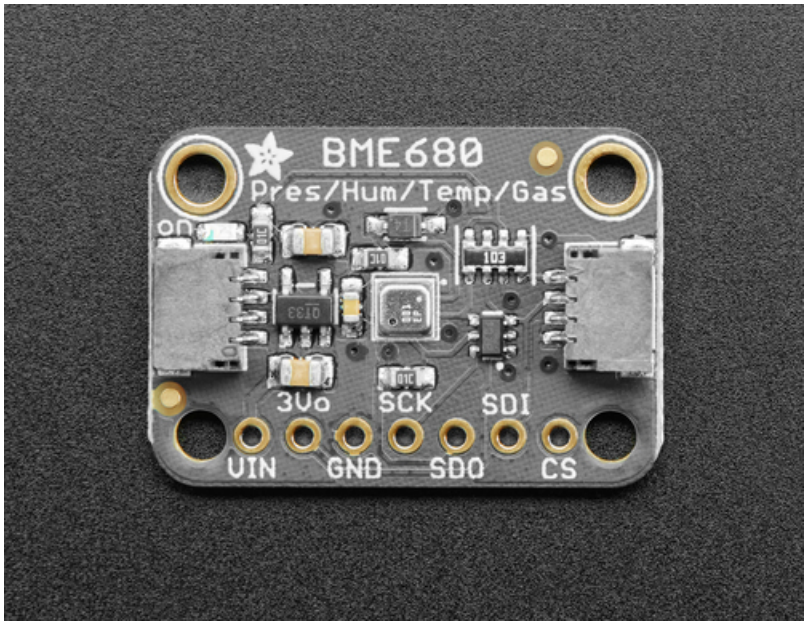
adafruit learning system

Adafruit BME680

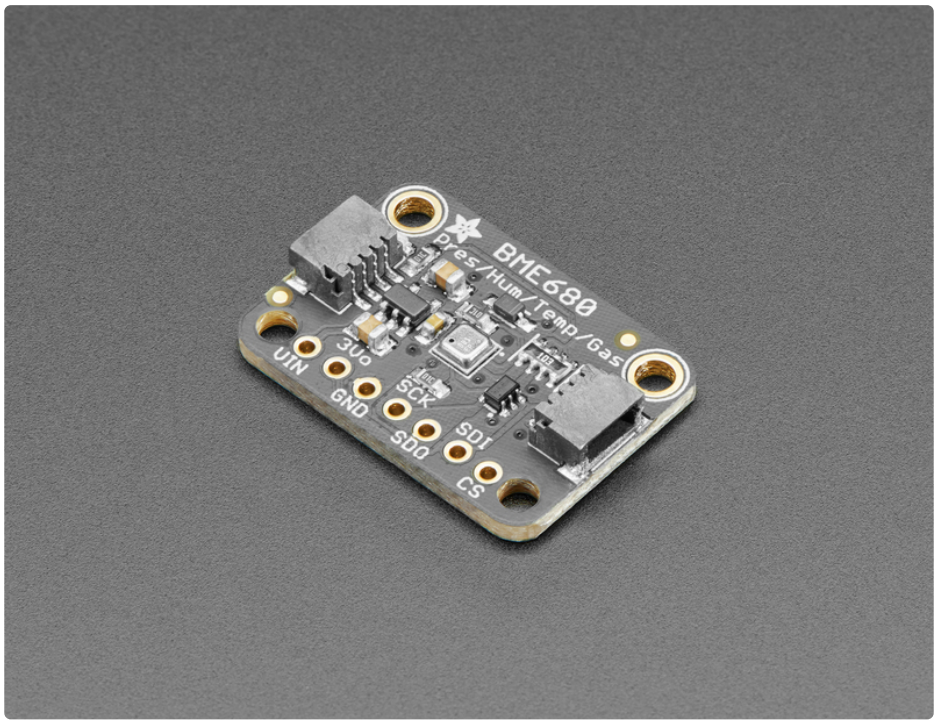
Created by lady ada



Last updated on 2021-10-22 11:21:19 AM EDT

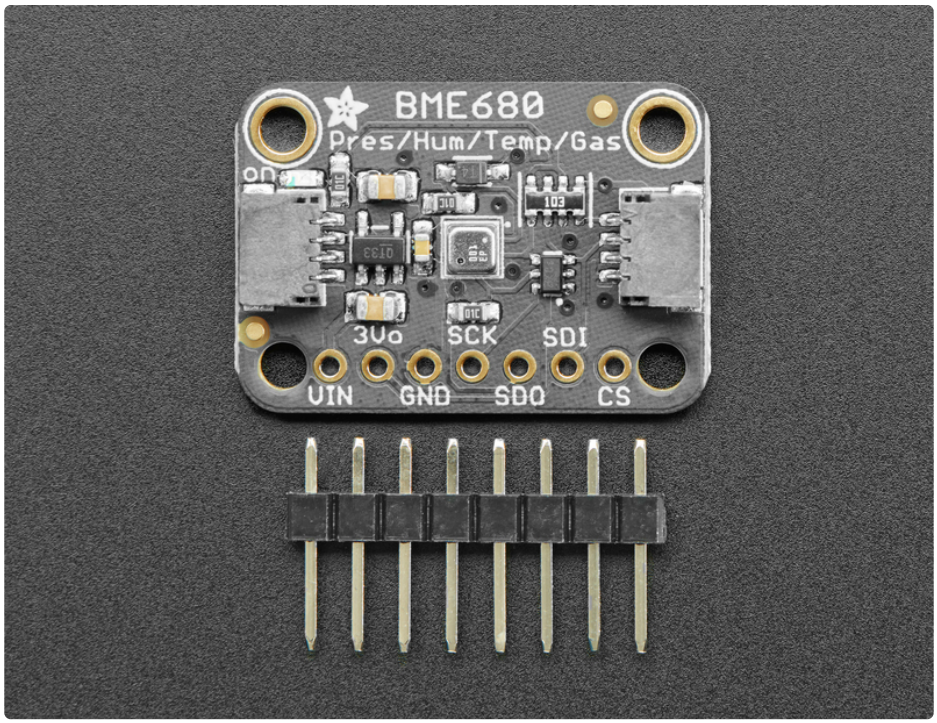
Guide Contents

Overview

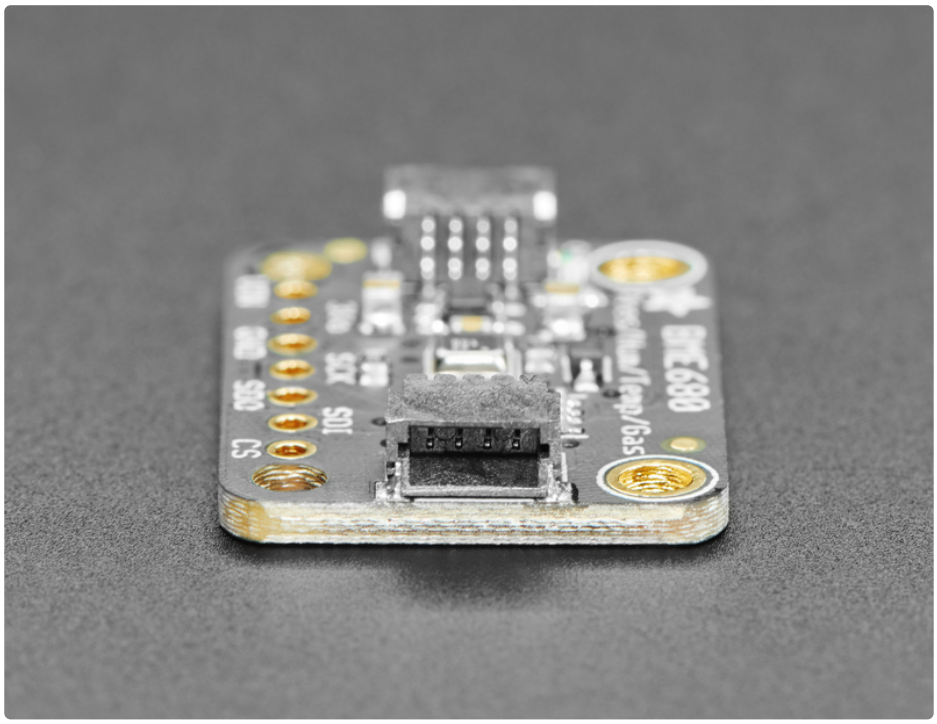


The long awaited BME680 from Bosch gives you *all the environmental sensing you want* in one small package. This little sensor contains **temperature**, **humidity**, **barometric pressure** and **VOC gas** sensing capabilities. All over SPI or I2C, at a great price!

Like the BME280 & BMP280, this precision sensor from Bosch can measure humidity with ±3% accuracy, barometric pressure with ±1 hPa absolute accuracy, and temperature with ±1.0°C accuracy. Because pressure changes with altitude, and the pressure measurements are so good, you can also use it as an altimeter with ±1 meter or better accuracy!

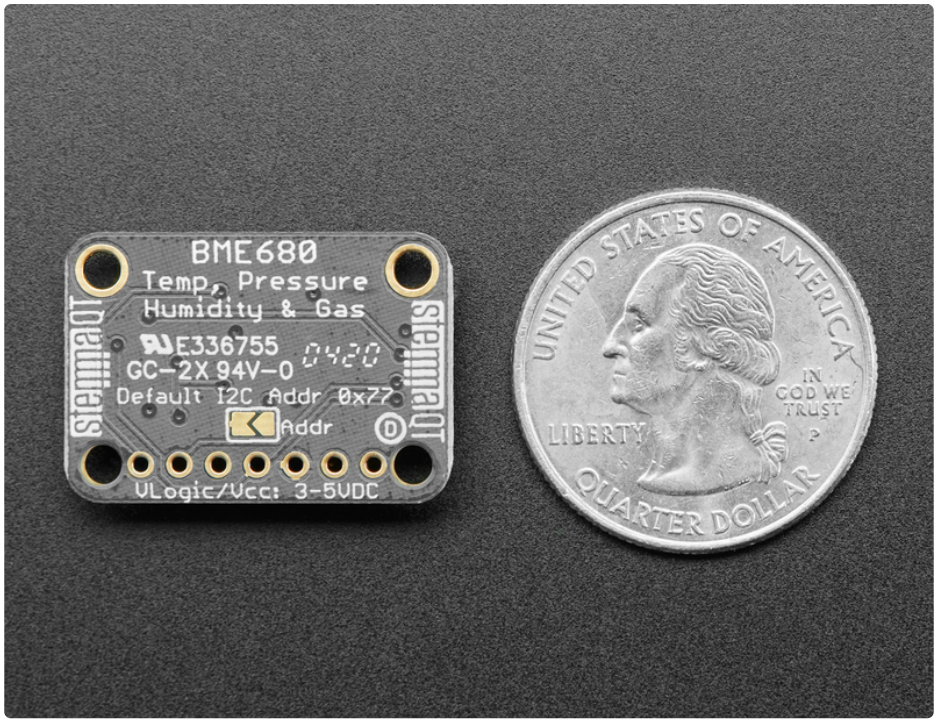


The BME680 takes those sensors to the next step in that it contains a small MOX sensor. The heated metal oxide changes resistance based on the volatile organic compounds (VOC) in the air, so it can be used to detect gasses & alcohols such as Ethanol, Alcohol and Carbon Monoxide and perform air quality measurements. Note it will give you one resistance value, with overall VOC content, it cannot differentiate gasses or alcohols.



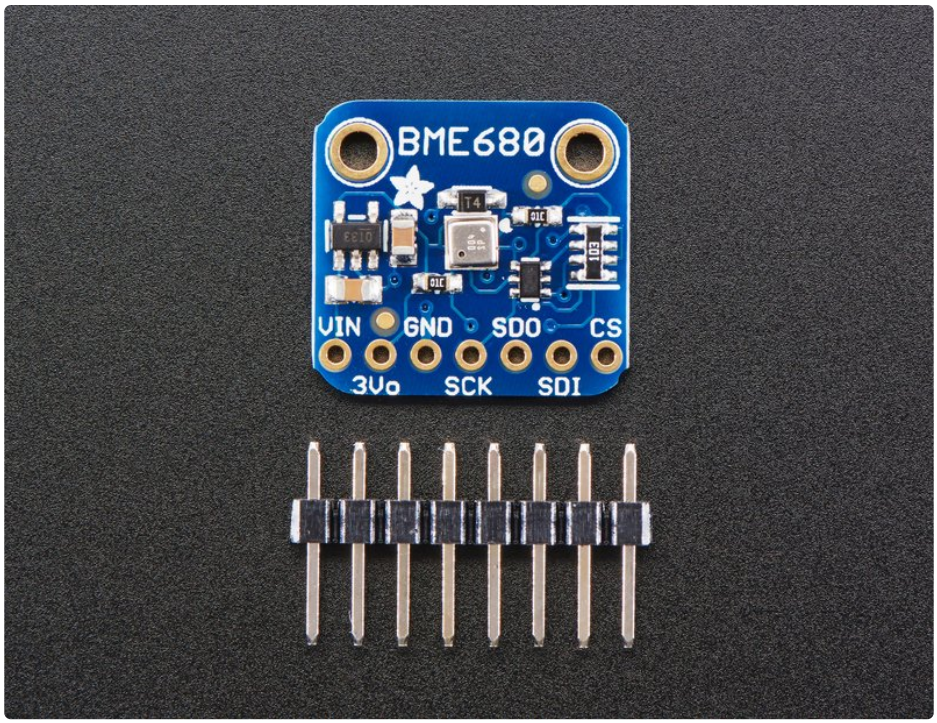
To make things easier and a bit more flexible, we've also included SparkFun

Qwiic (https://adafru.it/Fpw) compatible STEMMA QT (https://adafru.it/Ft4) connectors for the I2C bus so you don't even need to solder! Just plug in a compatible cable and attach it to your MCU of choice, and you're ready to load up some software and measure some light.



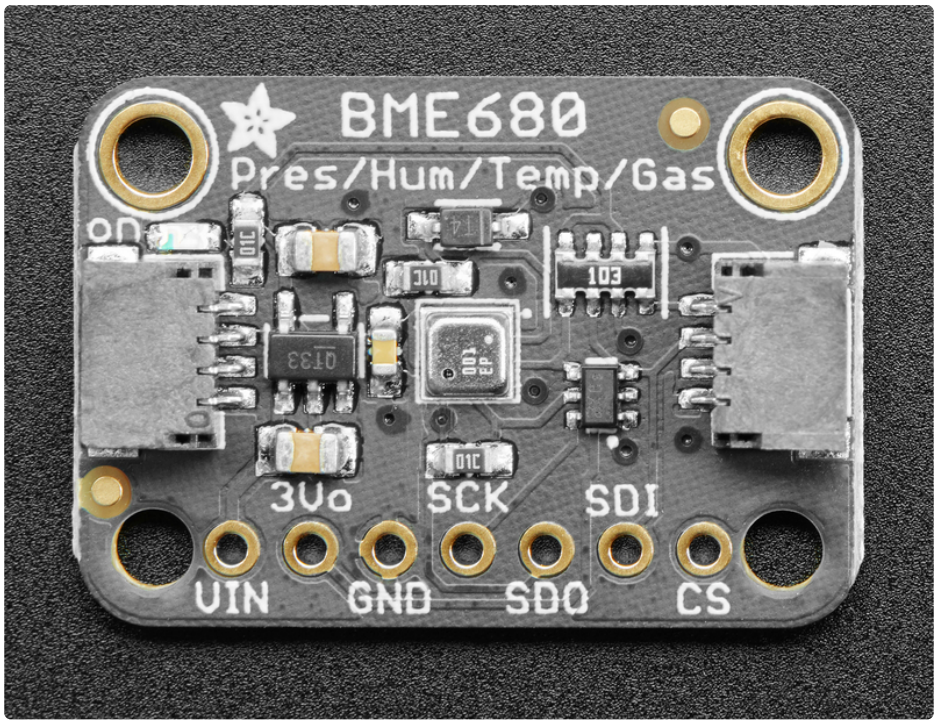
Please note, this sensor, like *all* VOC/gas sensors, has variability and to get precise measurements you will want to calibrate it against known sources! That said, for general environmental sensors, it will give you a good idea of trends and comparisons. We recommend that you run this sensor for 48 hours when you first receive it to "burn it in", and then 30 minutes in the desired mode every time the sensor is in use. This is because the sensitivity levels of the sensor will change during early use and the resistance will slowly rise over time as the MOX warms up to its baseline reading.

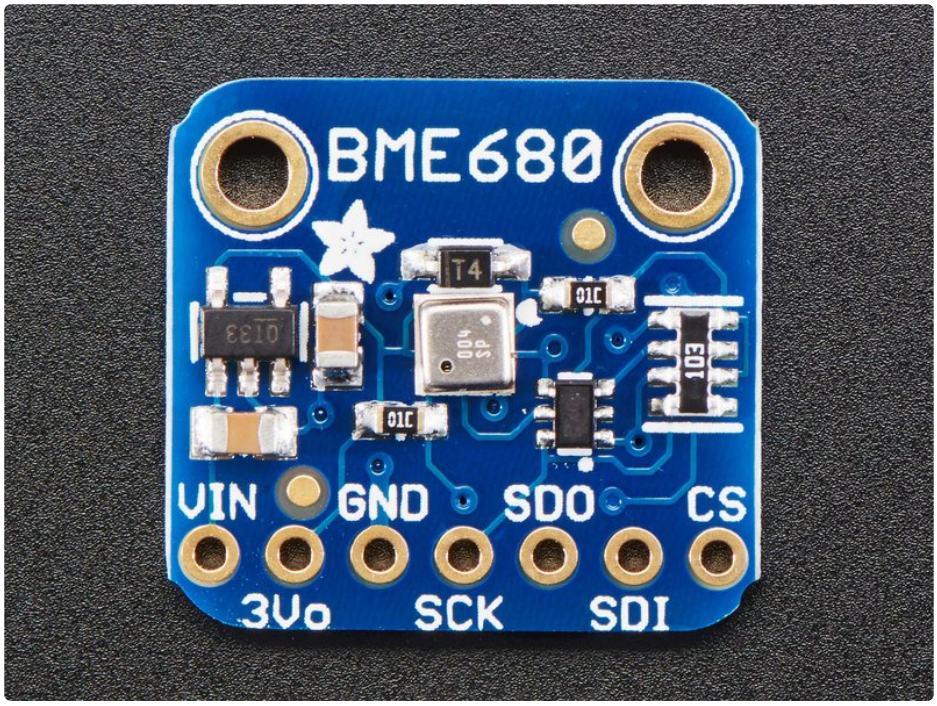
There are two versions of this board - the STEMMA QT version shown above, and the original header-only version shown below. Code works the same on both!



For your convenience we've pick-and-placed the sensor on a PCB with a 3.3V 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, 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
- **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

SPI Logic pins:

All pins going into the breakout have level shifting circuitry to make them 3-5V logic level safe. Use whatever logic level is on **Vin!**

- **SCK** - This is the **S**PI **C**lo**c**k pin, its an input to the chip
- **SDO** - this is the **S**erial **D**ata **O**ut / **M**icrocontroller **I**n **S**ensor **O**ut pin, for data sent from the BME680 to your processor
- **SDI** - this is the **S**erial **D**ata **I**n / **M**icrocontroller **O**ut **S**ensor **I**n pin, for data sent from your processor to the BME680
- **CS** - this is the **C**hip **S**elect pin, drop it low to start an SPI transaction. Its an input to the chip

If you want to connect multiple BME680's to one microcontroller, have them share the SDI, SDO and SCK pins. Then assign each one a unique CS pin.

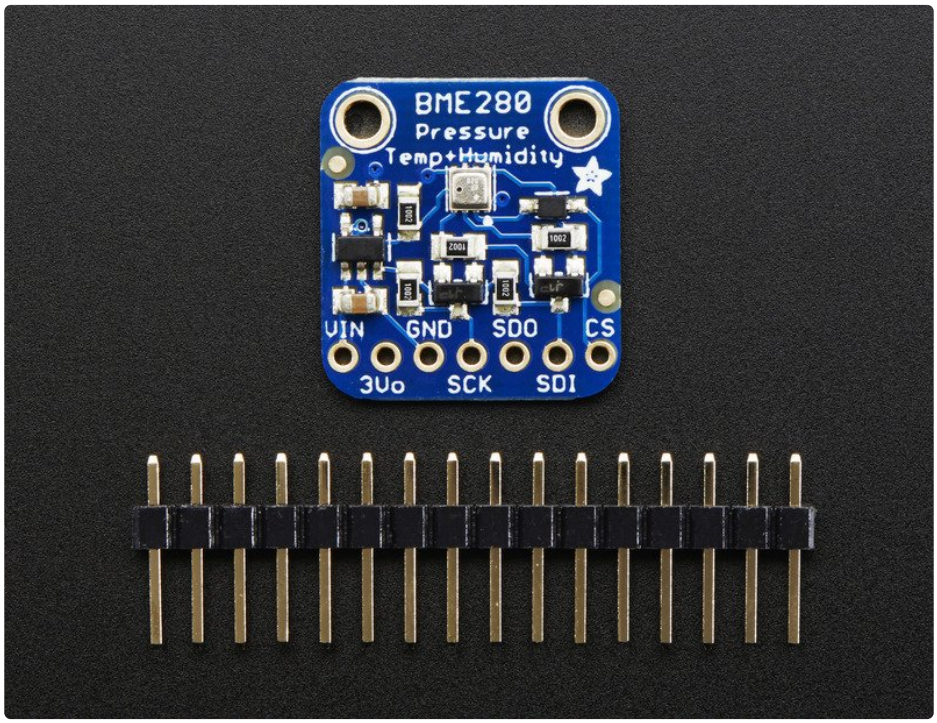
I2C Logic pins:

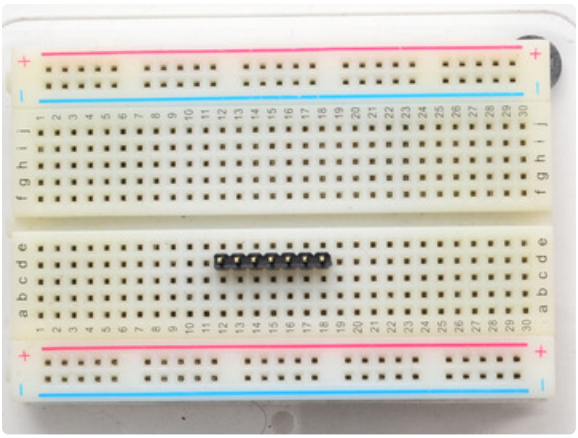
- **SCK** - this is *also* the I2C clock pin, connect to your microcontrollers I2C clock line.
- **SDI** - this is *also* the I2C data pin, connect to your microcontrollers I2C data line.
- **STEMMA QT** (https://adafru.it/Ft4) **-** These connectors allow you to connectors to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/Ft6)

Leave the other pins disconnected

Assembly

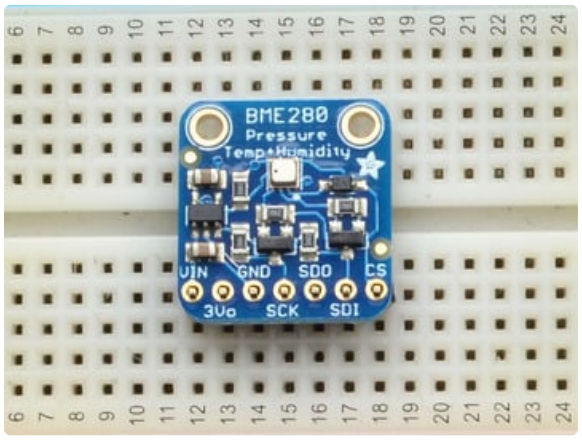
Your board may look a little different - the assembly process is the same!





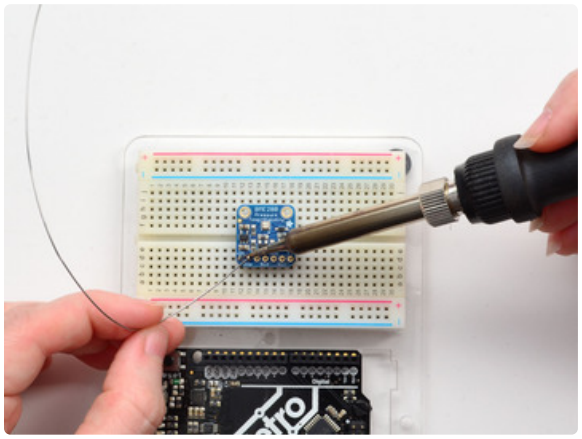
Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



Add the breakout board:

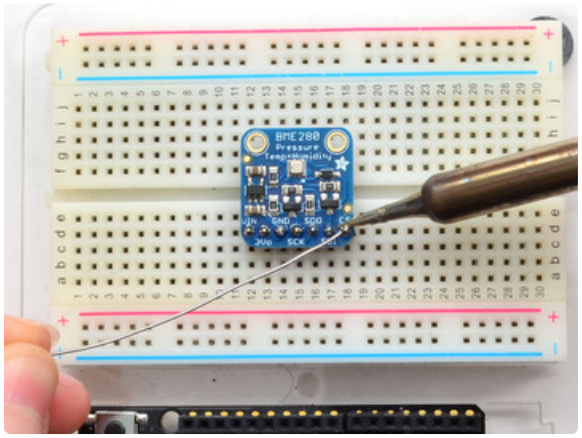
Place the breakout board over the pins so that the short pins poke through the breakout pads



And Solder!

Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).



You're done! Check your solder joints visually and continue onto the next steps

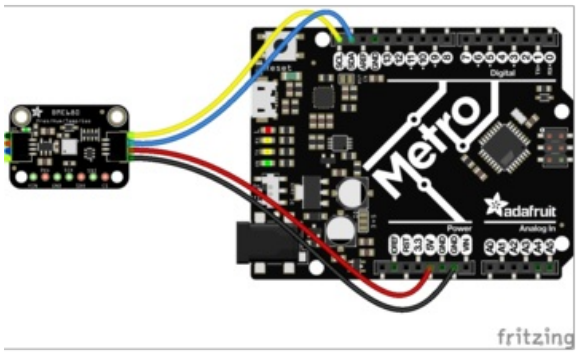
Arduino Wiring & Test

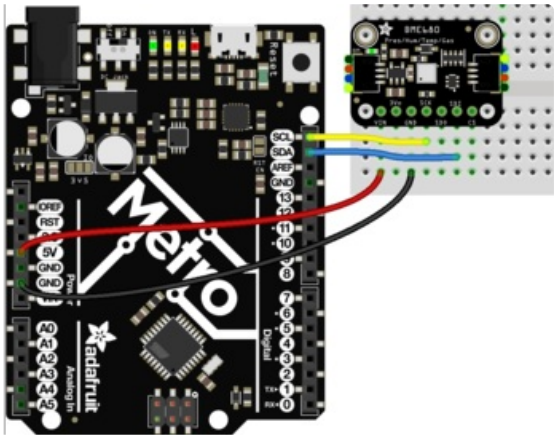
You can easily wire this breakout to any microcontroller, we'll be using an Arduino compatible. For another kind of microcontroller, as long as you have 4 available pins it is possible to 'bit-bang SPI' or you can use two I2C pins, but usually those pins are fixed in hardware. Just check out the library, then port the code.

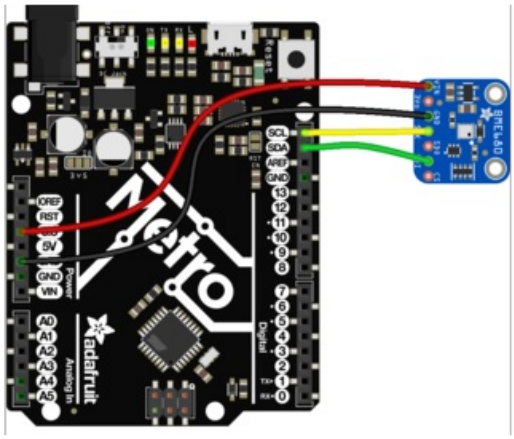
I2C Wiring

Use this wiring if you want to connect via I2C interface

By default, the i2c address is 0x77. If you add a jumper from SDO to GND, the address will change to 0x76.



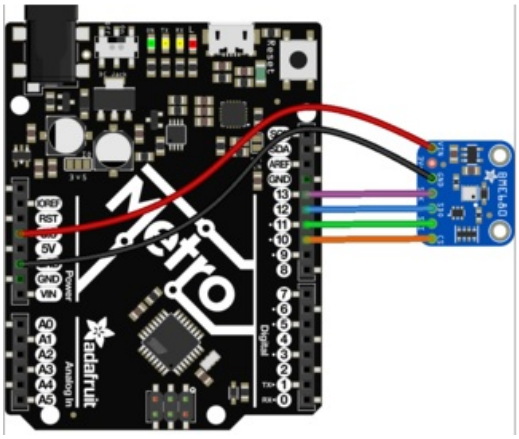




- Connect **Vin (red wire on STEMMA QT version)** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V. For 3.3V logic devices, use 3.3V
- Connect **GND (black wire on STEMMA QT version)** to common power/data ground
- Connect the **SCK** breakout pin to the I2C clock **SCL** pin on your Arduino compatible **(yellow wire on STEMMA QT version)**
- Connect the **SDI** breakout pin to the I2C data **SDA** pin on your Arduino compatible **(blue wire on STEMMA QT version)**

SPI Wiring

Since this is a SPI-capable sensor, we can use hardware or 'software' SPI. To make wiring identical on all microcontrollers, we'll begin with 'software' SPI. The following pins should be used:



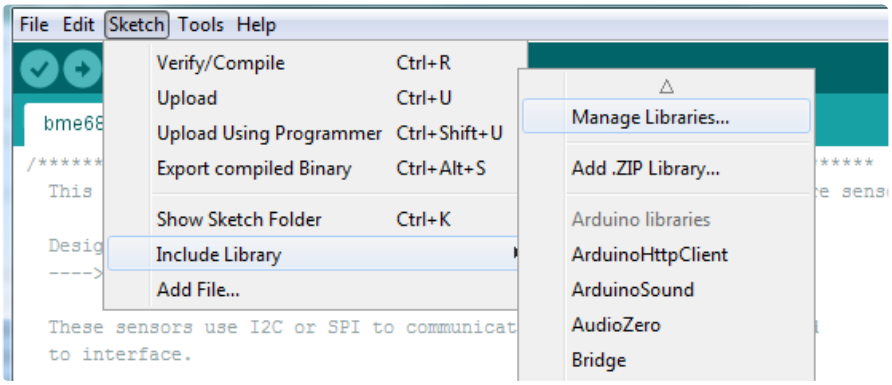
- Connect **Vin** to the power supply, 3V or 5V is fine. Use the same voltage that the microcontroller logic is based off of
- Connect **GND** to common power/data ground
- Connect the **SCK** pin to **Digital #13** but any pin can be used later
- Connect the **SDO** pin to **Digital #12** but any pin can be used later
- Connect the **SDI** pin to **Digital #11** but any pin can be used later
- Connect the **CS** pin **Digital #10** but any pin can be used later

Later on, once we get it working, we can adjust the library to use hardware SPI if you desire, or change the pins to others.

Install Adafruit_BME680 library

To begin reading sensor data, you will need to install the Adafruit_BME680 library (code on our github repository) (https://adafru.it/Btn). It is available from the Arduino library manager so we recommend using that.

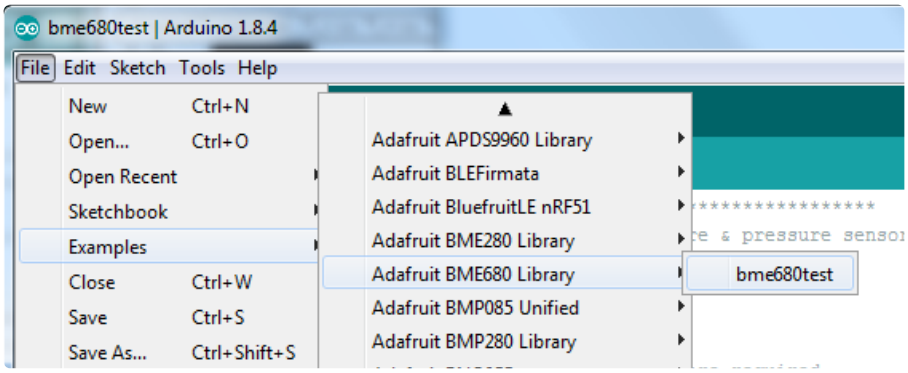
From the IDE open up the library manager...



And type in **adafruit bme680** to locate the library. Click **Install**

Load Demo

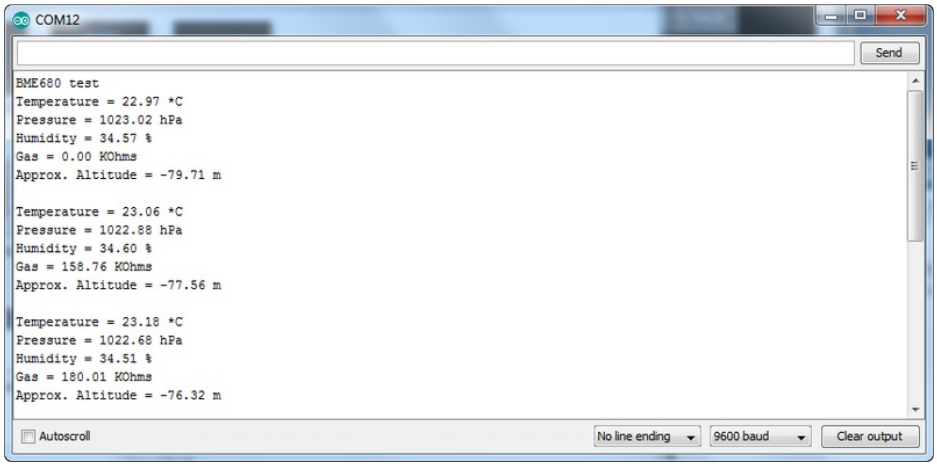
Open up **File->Examples->Adafruit_BME680->bmp680test** and upload to your microcontroller wired up to the sensor



Depending on whether you are using I2C or SPI, change the pin names and comment or uncomment the following lines.

```
#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO,  BME_SCK);
```

Once uploaded, open up the serial console at 9600 baud speed to see data being printed out



Temperature is calculated in degrees C, you can convert this to F by using the classic F = C * 9/5 + 32 equation.

Pressure is returned in the SI units of **Pascals**. 100 Pascals = 1 hPa = 1 millibar. Often times barometric pressure is reported in millibar or inches-mercury. For future reference 1 pascal =0.000295333727 inches of mercury, or 1 inch Hg = 3386.39 Pascal. So if you take the pascal value of say 100734 and divide by 3386.39 you'll get 29.72 inches-Hg.

Humidity is returned in Relative Humidity %

Gas is returned as a resistance value in ohms. This value takes up to 30 minutes to stabilize! Once it stabilizes, you can use that as your baseline reading. Higher concentrations of VOC will make the resistance lower.

You can also calculate Altitude. **However, you can only really do a good accurate job of calculating altitude if you know the hPa pressure at sea level for your location and day!** The sensor is quite precise but if you do not have the data updated for the current day then it can be difficult to get more accurate than 10 meters.

BSEC Air Quality Library

The BME680 doesn't have built-in air quality calcualtion capabilities like other sensors like the SGP30 or CCS811. Instead, you only get temperature, pressure, humidity and gas resistance (the raw resistance value of the sensor in the BME60. So we have to use a separate library from Bosch to perform the conversion to get Air Quality values like the VOC and equivalent CO2.

The Bosch BSEC library is an all-in-one Arduino library that will get you all the values from the sensor and also perform the AQI calculations. **It is not an open source library! You can *only* use it in Arduino and *only* with the chipsets supported.**

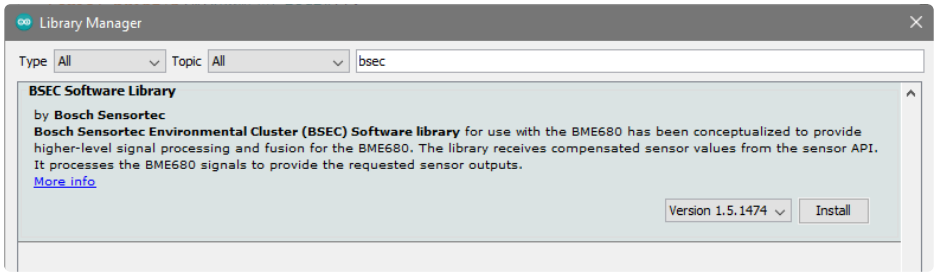
Not every chipset is supported by the closed-source BSEC library!

We have tested the Adafruit SAMD21 (M0) series of chips and these work great. You can use an Adafruit QT Py (https://adafru.it/Ofl), Trinket M0 (https://adafru.it/zya), Feather M0 (https://adafru.it/wRE), etc!

According to Bosch, ESP32 and ESP8266 are also supported (https://adafru.it/Ofm). AVR is not recommended - you definitely **cannot fit the library into an Uno/ATmega328 or ATmega32u4** , you could try using an Arduino Mega but it isnt suggested.

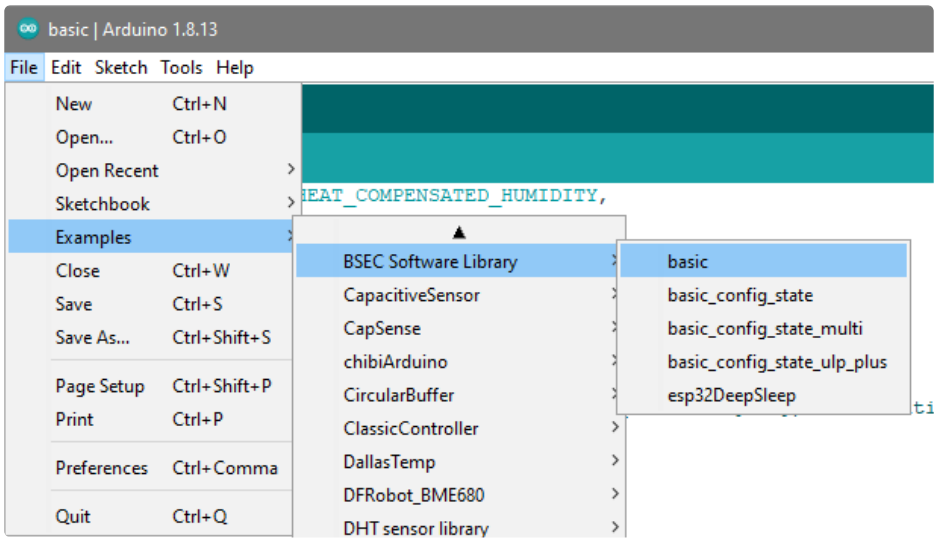
Really we recommend a SAMD21 or ESP board!

Install Library



Load Example & Adjust

Load up the `basic` example



Look for the part in `setup()` where the `iaqSensor` is started.

Change `BME680_I2C_ADDR_PRIMARY` to `BME680_I2C_ADDR_SECONDARY`

We also recommend adding

`while (!Serial) delay(10); // wait for console`

right after `Serial.begin` so that the console will print any errors out right after its opened

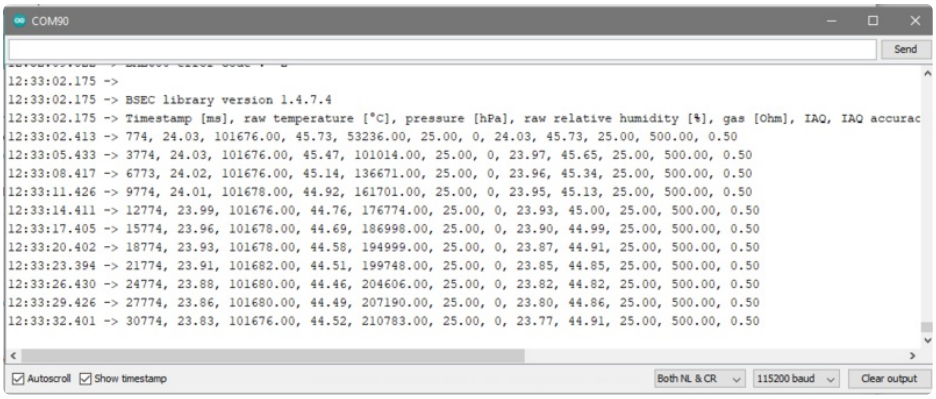
```
// Entry point for the example
void setup(void)
{
  Serial.begin(115200);
  while (!Serial) delay(10); // wait for console
  Wire.begin();

  iaqSensor.begin(BME680_I2C_ADDR_SECONDARY, Wire);
  output = "\nBSEC library version " + String(iaqSensor.version.major) + "
  Serial.println(output);
  checkIaqSensorStatus();
```

Upload to your board and open up the serial console at 115200 baud. You will see comma-separated values with the following heading

`Timestamp [ms], raw temperature [°C], pressure [hPa], raw relative humidity [%], gas [Ohm], IAQ, IAQ accuracy, temperature [°C], relative humidity [%], Static IAQ, CO2 equivalent, breath VOC equivalent`

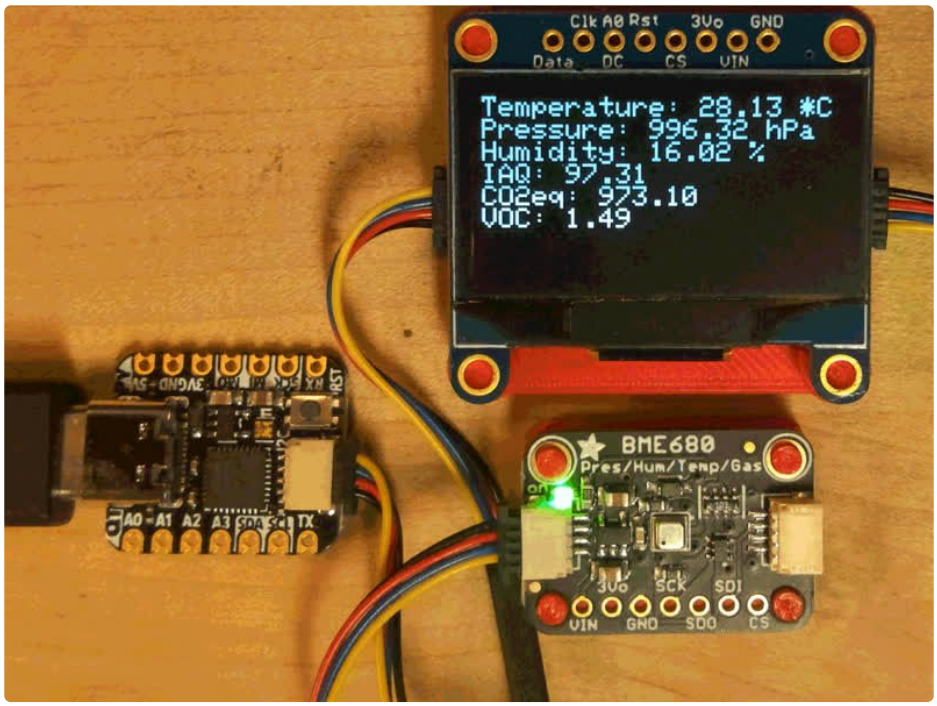
Note that the last 3 values are the calculated values. They will take a few minutes to normalize. Make sure to start your sensor in a clean air environment so it can normalize!

You can check the `basic_config_state` example if you want to calibrate the sensor, store the state in EEPROM, then re-write it on boot so you don't have to go through the normalization process.



QT Py + OLED Demo

Here's a demo that uses a 1.3" OLED + QT Py for a plug-n-play air quality display



Your browser does not support the video tag.

Adafruit QT Py - SAMD21 Dev Board with STEMMA QT

What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with our favorite lil chip, the SAMD21 (as made famous in our GEMMA M0 and Trinket M0 boards).This time it...

$7.50

In Stock

Add to Cart

Your browser does not support the video tag.

Monochrome 1.3" 128x64 OLED graphic display - STEMMA QT / Qwiic

These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This display is made of

128x64 individual white OLED pixels,...

$19.95

In Stock

Add to Cart

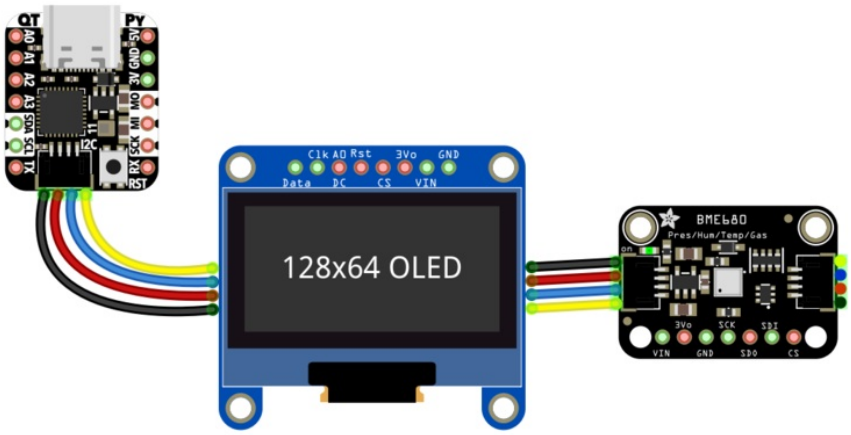
STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long

This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...

$0.95

In Stock

Add to Cart




```
/**************************************************************************
  This is a library for the BME680 gas, humidity, temperature & pressure sensor

  Designed specifically to work with the Adafruit BME680 Breakout
  ----> http://www.adafruit.com/products/3660

  These sensors use I2C or SPI to communicate, 2 or 4 pins are required
  to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 **************************************************************************/

#include <Adafruit_SSD1306.h>
#include "bsec.h"

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 64, &Wire);

Bsec iaqSensor;
```

```
String output;


void setup() {
  Serial.begin(9600);
  //while (!Serial);

  Serial.println(F("BME680 test"));

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3D)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  Serial.println("OLED begun");

  display.display();
  delay(100);
  display.clearDisplay();
  display.display();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setRotation(0);

  iaqSensor.begin(BME680_I2C_ADDR_SECONDARY, Wire);
  output = "\nBSEC library version " + String(iaqSensor.version.major) + "." +
String(iaqSensor.version.minor) + "." + String(iaqSensor.version.major_bugfix) + "." +
String(iaqSensor.version.minor_bugfix);
  Serial.println(output);
  checkIaqSensorStatus();
  bsec_virtual_sensor_t sensorList[10] = {
    BSEC_OUTPUT_RAW_TEMPERATURE,
    BSEC_OUTPUT_RAW_PRESSURE,
    BSEC_OUTPUT_RAW_HUMIDITY,
    BSEC_OUTPUT_RAW_GAS,
    BSEC_OUTPUT_IAQ,
    BSEC_OUTPUT_STATIC_IAQ,
    BSEC_OUTPUT_CO2_EQUIVALENT,
    BSEC_OUTPUT_BREATH_VOC_EQUIVALENT,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY,
  };

  iaqSensor.updateSubscription(sensorList, 10, BSEC_SAMPLE_RATE_LP);
  checkIaqSensorStatus();
  // Print the header
  output = "Timestamp [ms], raw temperature [°C], pressure [hPa], raw relative humidity [%], gas
[Ohm], IAQ, IAQ accuracy, temperature [°C], relative humidity [%], Static IAQ, CO2 equivalent,
breath VOC equivalent";
  Serial.println(output);
}

void loop() {
  display.setCursor(0,0);
  display.clearDisplay();

  unsigned long time_trigger = millis();
```

```
  if (! iaqSensor.run()) { // If no data is available
    checkIaqSensorStatus();
    return;
  }

  output = String(time_trigger);
  output += ", " + String(iaqSensor.rawTemperature);
  output += ", " + String(iaqSensor.pressure);
  output += ", " + String(iaqSensor.rawHumidity);
  output += ", " + String(iaqSensor.gasResistance);
  output += ", " + String(iaqSensor.iaq);
  output += ", " + String(iaqSensor.iaqAccuracy);
  output += ", " + String(iaqSensor.temperature);
  output += ", " + String(iaqSensor.humidity);
  output += ", " + String(iaqSensor.staticIaq);
  output += ", " + String(iaqSensor.co2Equivalent);
  output += ", " + String(iaqSensor.breathVocEquivalent);
  Serial.println(output);


  Serial.print("Temperature = "); Serial.print(iaqSensor.temperature); Serial.println(" *C");
  display.print("Temperature: "); display.print(iaqSensor.temperature); display.println(" *C");

  Serial.print("Pressure = "); Serial.print(iaqSensor.pressure / 100.0); Serial.println(" hPa");
  display.print("Pressure: "); display.print(iaqSensor.pressure / 100); display.println(" hPa");

  Serial.print("Humidity = "); Serial.print(iaqSensor.humidity); Serial.println(" %");
  display.print("Humidity: "); display.print(iaqSensor.humidity); display.println(" %");

  Serial.print("IAQ = "); Serial.print(iaqSensor.staticIaq); Serial.println("");
  display.print("IAQ: "); display.print(iaqSensor.staticIaq); display.println("");

  Serial.print("CO2 equiv = "); Serial.print(iaqSensor.co2Equivalent); Serial.println("");
  display.print("CO2eq: "); display.print(iaqSensor.co2Equivalent); display.println("");

  Serial.print("Breath VOC = "); Serial.print(iaqSensor.breathVocEquivalent);
Serial.println("");
  display.print("VOC: "); display.print(iaqSensor.breathVocEquivalent); display.println("");

  Serial.println();
  display.display();
  delay(2000);
}


// Helper function definitions
void checkIaqSensorStatus(void)
{
  if (iaqSensor.status != BSEC_OK) {
    if (iaqSensor.status < BSEC_OK) {
      output = "BSEC error code : " + String(iaqSensor.status);
      Serial.println(output);
      display.setCursor(0,0);
      display.println(output);
      display.display();
      for (;;)  delay(10);
    } else {
```

```
    } else {
      output = "BSEC warning code : " + String(iaqSensor.status);
      Serial.println(output);
    }
  }

  if (iaqSensor.bme680Status != BME680_OK) {
    if (iaqSensor.bme680Status < BME680_OK) {
      output = "BME680 error code : " + String(iaqSensor.bme680Status);
      Serial.println(output);
      display.setCursor(0,0);
      display.println(output);
      display.display();
      for (;;)  delay(10);
    } else {
      output = "BME680 warning code : " + String(iaqSensor.bme680Status);
      Serial.println(output);
    }
  }
}
```

Arduino Library Docs

Arduino Library Docs (https://adafru.it/Awf)

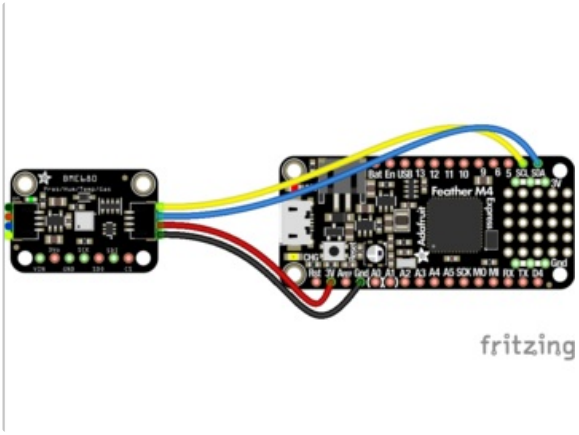
Python & CircuitPython

It's easy to use the BME680 sensor with Python and CircuitPython, and the Adafruit CircuitPython BME680 (https://adafru.it/Bto) module. This module allows you to easily write Python code that reads the humidity, temperature, pressure, and more from the sensor.

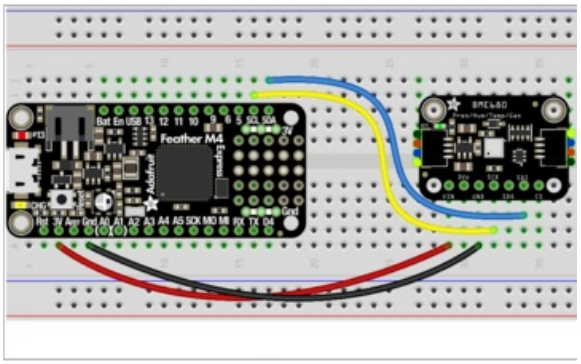
You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

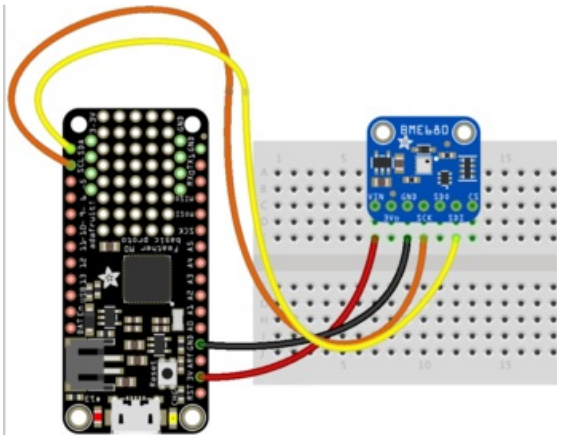
CircuitPython Microcontroller Wiring

First wire up a BME680 to your board exactly as shown on the previous pages for Arduino. Here's an example of wiring a Feather M0 to the sensor with I2C:



- **Board 3V** to **sensor VIN (red wire on STEMMA QT version)**
- **Board GND** to **sensor GND (black wire on STEMMA QT version)**
- **Board SCL** to **sensor SCK (yellow wire on STEMMA QT version)**
- **Board SDA** to **sensor SDI (blue wire on STEMMA QT version)**



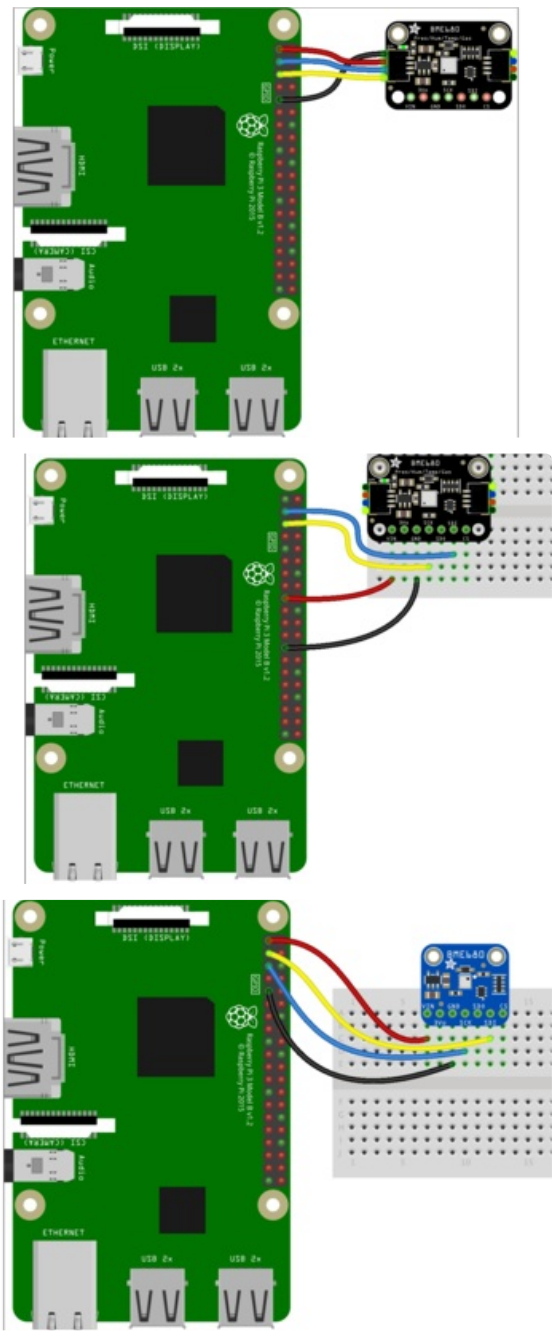


Python Computer Wiring

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](https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C:



- **Pi 3V3** to **sensor VIN (red wire on STEMMA QT version)**
- **Pi GND** to **sensor GND (black wire on STEMMA QT version)**
- **Pi SCL** to **sensor SCK (yellow wire on STEMMA QT version)**
- **Pi SDA** to **sensor SDI (blue wire on STEMMA QT version)**

CircuitPython Installation of BME680 Library

Next you'll need to install the [Adafruit CircuitPython BME680](https://adafru.it/Bto) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) 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 (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) 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_bme680.mpy**
- **adafruit_bus_device**

You can also download the **adafruit_bme680.mpy** from its releases page on Github (https://adafru.it/Btr).

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_bme680.mpy,** and **adafruit_bus_device** files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

Python Installation of BME680 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 (https://adafru.it/BSN)!

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

- `sudo pip3 install adafruit-circuitpython-bme680`

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!

CircuitPython & Python Usage

To demonstrate the usage of the sensor we'll initialize it and read the temperature, humidity, and more 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_bme680
i2c = board.I2C()
sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)
```

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

- **temperature** - The sensor temperature in degrees Celsius.
- **gas** - The resistance (in Ohms) of the gas sensor. This is proportional to the amount of VOC particles in the air.
- **humidity** - The percent humidity as a value from 0 to 100%.
- **pressure** - The pressure in hPa.
- **altitude** - The altitude in meters.

```
print('Temperature: {} degrees C'.format(sensor.temperature))
print('Gas: {} ohms'.format(sensor.gas))
print('Humidity: {}%'.format(sensor.humidity))
print('Pressure: {}hPa'.format(sensor.pressure))
```

```
>>> print('Temperature: {} degrees C'.format(sensor.temperature))
Temperature: 32.2855 degrees C
>>> print('Gas: {} ohms'.format(sensor.gas))
Gas: 11671 ohms
>>> print('Humidity: {}%'.format(sensor.humidity))
Humidity: 43.9525%
>>> print('Pressure: {}hPa'.format(sensor.pressure))
Pressure: 1017.28hPa
>>>
```

For altitude you'll want to set the pressure at sea level for your location to get the most accurate measure (remember these sensors can only infer altitude based on pressure and need a set calibration point). Look at your local weather report for a pressure at sea level reading and set the **seaLevelhPA** property:

```
sensor.seaLevelhPa = 1014.5
```

Then read the altitude property for a more accurate altitude reading (but remember this altitude will fluctuate based on atmospheric pressure changes!):

```
print('Altitude: {} meters'.format(sensor.altitude))
```

```
>>> sensor.seaLevelhPa = 1014.5
>>> print('Altitude: {} meters'.format(sensor.altitude))
Altitude: -25.3658 meters
>>>
```

That's all there is to using the BME680 sensor with CircuitPython!

Full Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_bme680

# Create sensor object, communicating over the board's default I2C bus
i2c = board.I2C()  # uses board.SCL and board.SDA
bme680 = adafruit_bme680.Adafruit_BME680_I2C(i2c, debug=False)

# change this to match the location's pressure (hPa) at sea level
bme680.sea_level_pressure = 1013.25

# You will usually have to add an offset to account for the temperature of
# the sensor. This is usually around 5 degrees but varies by use. Use a
# separate temperature sensor to calibrate this one.
temperature_offset = -5

while True:
    print("\nTemperature: %0.1f C" % (bme680.temperature + temperature_offset))
    print("Gas: %d ohm" % bme680.gas)
    print("Humidity: %0.1f %%" % bme680.relative_humidity)
    print("Pressure: %0.3f hPa" % bme680.pressure)
    print("Altitude = %0.2f meters" % bme680.altitude)

    time.sleep(1)
```

Python Docs

Python Docs (https://adafru.it/C42)

Downloads

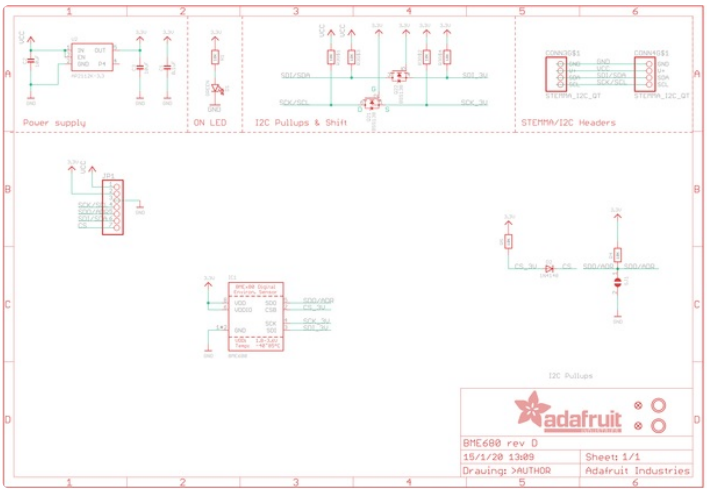
Files

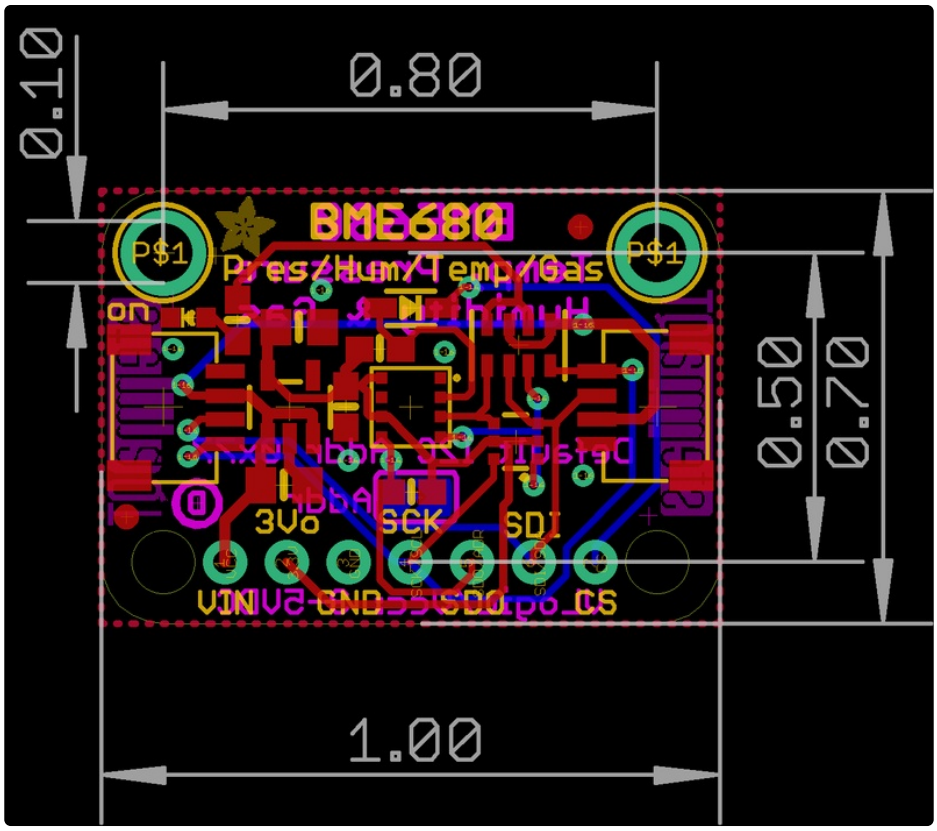
- Fritzing object in Adafruit Fritzing library (https://adafru.it/c7M)
- EagleCAD PCB files on github (https://adafru.it/Btt)
- BME680 Datasheet (https://adafru.it/Btu)

More reading:

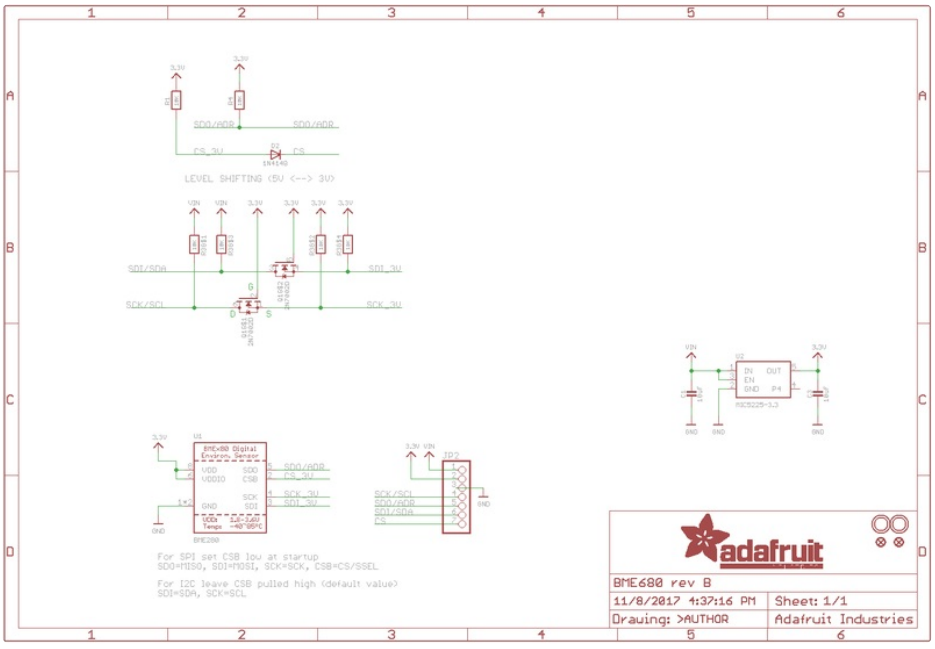
- The next generation of low-cost personal air quality sensors forquantitative exposure monitoring (https://adafru.it/Bts)
- New small, low-power MOX VOC sensors: how might they be used for indoor air quality monitoring? (https://adafru.it/Btv)

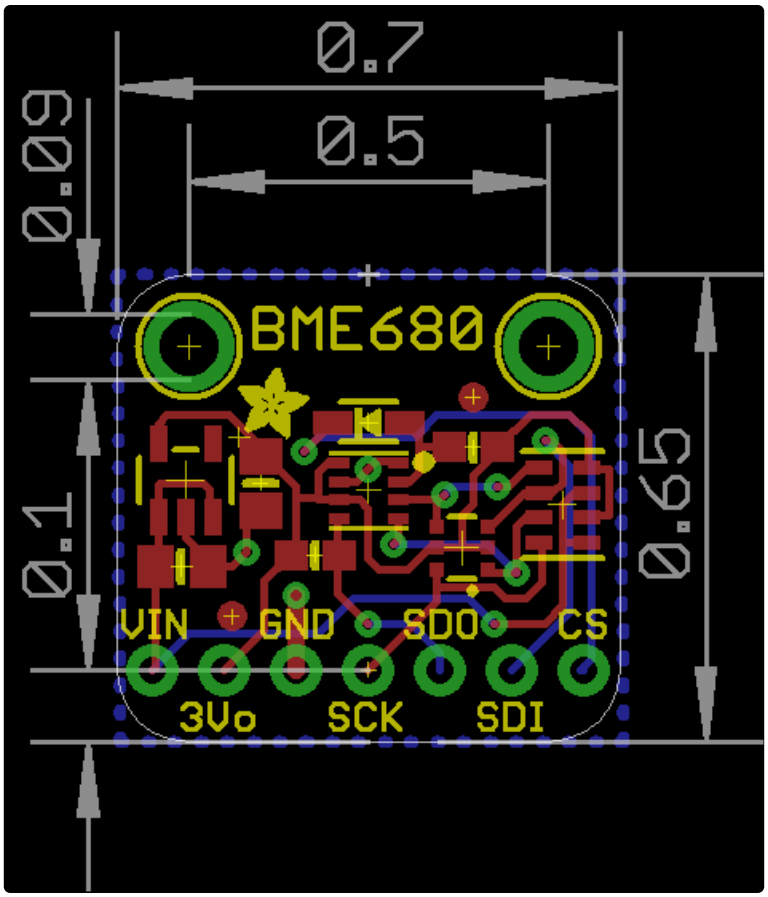
Schematic & Fabrication Print - STEMMA QT Version





Schematic & Fabrication Print - Original Version




0.7
0.5
0.09
0.1
0.65
BME680
VIN
GND
SDO
CS
3Vo
SCK
SDI