Getting Started with BME680 Breakout

Originalartikel

Backup

<html> <p>This tutorial will show you how to solder your BME680 breakout, install and use the <a href=„https://github.com/pimoroni/bme680“>Python library</a>, and suggest some uses for it. We'll go through the functions of the Python library, including how to read all of the various sensor values.</p> <p><img src=„https://learn.pimoroni.com/static/repos/learn/sandyj/bme680-1.jpg“ alt=„BME680 mounted on Pi Zero W“/></p> <h2>Soldering your BME680 breakout</h2> <p>There are a couple of options when it comes to soldering your BME680 breakout.</p> <p>We've included a piece of right-angle female header that you can solder on and then pop the breakout right onto pins 1, 3, 5, 7, and 9 of your Raspberry Pi (the bottom left 5 pins on the GPIO header, as shown above and below).</p> <p><strong>Note that it's important that you don't mount the sensor on your Pi's pins back to front, or it will cause damage to the sensor! The side of the board with the components on should be closest to the top edge of your Pi if you're using the right-angle header.</strong></p> <p><img src=„https://learn.pimoroni.com/static/repos/learn/sandyj/bme680-3.jpg“ alt=„BME680 with right-angle female header“/></p> <p>There's a piece of male header that you can use if you plan to use female-to-female jumper wires or a breadboard with your breakout.</p> <p><img src=„https://learn.pimoroni.com/static/repos/learn/sandyj/bme680-2.jpg“ alt=„BME680 with standard male header“/></p> <p><img src=„https://learn.pimoroni.com/static/repos/learn/sandyj/bme680-4.jpg“ alt=„BME680 in breadboard, connected to Arduino“/></p> <p>Or you can solder wires straight to the breakout if you plan to use the breakout in a more permanent, embedded setup.</p> <p>If you're not a confident solderer, then you can check out our <a href=„https://learn.pimoroni.com/tutorial/sandyj/the-ultimate-guide-to-soldering“>Ultimate Guide to Soldering</a> for a bunch of tips on how to get started and how to improve your soldering.</p> <h2>Changing the default I2C address</h2> <p>The breakout has a default I2C address of 0x76, but this can be changed so that you can use up to two breakouts on the same Raspberry Pi or Arduino. To change the I2C address to 0x77, simply flow a small blob of solder across the two solder pads so that it bridges the pads. If you decide to change it back at a later date, then you can use a solder sucker or some solder braid to remove the solder and un-bridge the pads.</p> <p><img src=„https://learn.pimoroni.com/static/repos/learn/sandyj/bme680-5.jpg“ alt=„Close-up of mounted BME680 and solder pads“/></p> <h2>Installing the software</h2> <p>We always recommend using the most up-to-date version of Raspbian, as this is what we test our boards and software against, and it often helps to start with a completely fresh install of Raspbian, although this isn't necessary.</p> <p>You'll need to have I2C enabled on your Raspberry Pi. You can do this in the Interfaces section of the Raspberry Pi Configuration menu, by typing

```
sudo raspi-config
```

in the terminal, or by using our one-line-configuration below, in the terminal:</p> <div class=„codehilite“ readability=„6“> <pre>curl https://get.pimoroni.com/i2c | bash </pre></div> <p>Next, in the terminal, type the following to clone the repository and install it:</p> <div class=„codehilite“ readability=„6“> <pre>git clone https://github.com/pimoroni/bme680 cd bme680/library sudo python setup.py install </pre></div> <p>The library should now be installed. You'll find the examples in

```
bme680/examples
```

.</p> <p>You can also use pip to install the BME680 library, as follows, although you'll have to download the examples separately:</p> <div class=„codehilite“ readability=„6“> <pre>sudo pip install bme680 </pre></div> <p>Or you can use our one-line-installer, which should set up and install everything in one go, and download the examples into the

```
/home/pi/Pimoroni/bme680/examples
```

folder:</p> <pre>curl https://get.pimoroni.com/bme680 | bash </pre> <h2>Running the built-in example and burning in the sensor</h2> <p>We've included an example that prints out all of the sensor readings - the temperature, pressure, humidity, and gas resistance value - continuously.</p> <p>In our testing, we've found that it helps to run the sensor for at least 20 minutes the first time that you use it, to „burn in“ the sensor. The sensor's readings, especially the gas resistance readings, will drift gradually and then stabilise after a while. This drift will happen every time you start taking readings but, after the initial burn-in, the readings should stabilise fairly quickly each time, usually after a couple of minutes.</p> <p>To run the example, type the following:</p> <div class=„codehilite“ readability=„6“> <pre>cd /home/pi/bme680/examples python read-all.py </pre></div> <p>Leave the example running for a couple of minutes, or for at least 20 minutes if you haven't run it before (watch for the gas resistance readings stabilising). You can press control-c to stop the example running.</p> <p>If you want to log the sensor values to a text file, then it's simple to do that by redirecting the output of the program to a text file in the terminal, as follows:</p> <div class=„codehilite“ readability=„6“> <pre>python read-all.py &gt; bme680-data.txt </pre></div> <p>Note that the values are comma-separated, so you should be able to import them into a spreadsheet to plot them fairly easily.</p> <h2>Imports and configuring the sensor</h2> <p>Reading the sensor values on the BME680 is fairly straightforward, but requires quite a few configuration values to be set initially. You can also run the sensor in two different „modes“ - with or without gas readings being taken - and if gas readings aren't taken and just temperature, pressure, and humidity readings are taken, then you can sample data much more quickly.</p> <p><strong>When gas readings are being taken, because the gas measurement requires the hot plate inside the sensor to be heated, the temperature, pressure, and humidity readings will be taken first to minimise the effect of the hot plate on the temperature reading.</strong></p> <p>We'll look first at the library import and the configuration settings. Open a terminal window, type

```
python
```

to open a Python prompt, and then type the following:</p> <div class=„codehilite“ readability=„9“> <pre>import bme680 import time sensor = bme680.BME680() sensor.set_humidity_oversample(bme680.OS_2X) sensor.set_pressure_oversample(bme680.OS_4X) sensor.set_temperature_oversample(bme680.OS_8X) sensor.set_filter(bme680.FILTER_SIZE_3) sensor.set_gas_status(bme680.ENABLE_GAS_MEAS) sensor.set_gas_heater_temperature(320) sensor.set_gas_heater_duration(150) sensor.select_gas_heater_profile(0) </pre></div> <p>We'll go through quickly what the above means, but most of it can be left unmeddled with in the majority of cases.</p> <p>First, we import the

```
bme680
```

and

```
time
```

libraries. We'll be using the

```
time
```

library to introduce a small delay between each reading of the sensor.</p> <p>

```
sensor = bme680.BME680()
```

creates an instance of the sensor, that we'll use to set the settings and take the readings from.</p> <p>The

```
_oversample
```

settings that are set for the humidity, pressure, and temperature set a balance between accuracy of reading and amount of noise. The higher the oversampling, the greater the reduction in noise, albeit with a loss of accuracy.</p> <p>The

```
filter
```

protects sensor readings against transient changes in conditions, e.g. a door slamming will cause the pressure to change momentarily, and the IIR filter will filter out these transient spiky values.</p> <p>The gas measurement has a few settings that can be tweaked. It can be enabled or disabled with

```
set_gas_status
```

. Disabling it allows the other readings to be taken more rapidly, as mentioned above. The temperature and duration that the hot plate is held at that temperature can be altered, although we'd recommend not altering these settings if your gas resistance readings look sensible.</p> <h2>Reading data from the sensor</h2> <p>Data can be read from the sensor as follows:</p> <div class=„codehilite“ readability=„14“> <pre>while True:

```
  if sensor.get_sensor_data():
      output = "{0:.2f} C,{1:.2f} hPa,{2:.2f}
%RH".format(sensor.data.temperature, sensor.data.pressure,
sensor.data.humidity)
      if sensor.data.heat_stable:
          print("{0},{1} Ohms".format(output, sensor.data.gas_resistance))
      else:
          print(output)
  time.sleep(1)
```

</pre></div> <p>You'll see that the temperature, pressure, and humidity can be read with

```
sensor.data.temperature
```

,

```
sensor.data.pressure
```

, and

```
sensor.data.humidity
```

respectively (the units are degrees Celsius, hectoPascals, and % relative humidity).</p> <p>The gas is read with

```
sensor.data.gas_resistance
```

and will return a resistance reading in Ohms, up to several hundred thousand Ohms (there's more on how to interpret these readings in the section below).</p> <p>There are a couple of extra lines -

```
if sensor.get_sensor_data():
```

and

```
if sensor.data.heat_stable:
```

- that ensure that readings are ready to be taken. They both return Boolean values,

```
True
```

or

```
False
```

.</p> <p>The lines that format the output look a little scary, but they're just placeholders that let you format the values nicely. We'll go through how it works.</p> <div class=„codehilite“ readability=„11“> <pre>output = „{0:.2f} C,{1:.2f} hPa,{2:.2f} %RH“.format(sensor.data.temperature, sensor.data.pressure, sensor.data.humidity) </pre></div> <p>The parts in the curly brackets -

```
{0:.2f}
```

- are placeholders. The first

```
0
```

is the index of the placeholder (you'll see the others are

```
1
```

and

```
2
```

). The remaining bit -

```
:.2f
```

- tells it that the number is a floating point number (

```
f
```

), and to use two decimal places after the decimal point (

```
.2
```

). Then in the round brackets, after

```
.format
```

, we pass it the three values that we want to replace the placeholders with -

```
sensor.data.temperature, sensor.data.pressure, sensor.data.humidity
```

.</p> <p>There's a great explanation of all of the different string formatting options at the <a href=„https://pyformat.info/“>PyFormat</a> site.</p> <p>After checking that the gas heater is stable, we also place the gas resistance reading into our formatted

```
output
```

string -

```
"{0},{1} Ohms".format(output, sensor.data.gas_resistance)
```

.</p> <p>Finally, a

```
time.sleep(1)
```

introduces a small pause between readings.</p> <h2>Interpreting the gas resistance readings</h2> <p>The sensor produces gas resistance readings in Ohms. These will range from the low thousands up to several hundred thousand Ohms. Every time that you use the sensor, it will take a few minutes (or more if it's the first time you've used it) to stabilise; you'll see the readings creeping upwards. When the readings stabilise, this will be your background reference reading.</p> <p>If the air quality increases, then the gas resistance reading will increase, and if the air quality decreases, then the gas resistance reading will decrease also. You can test it out by holding something like a permanent marker, or spraying a little perfume on a piece of tissue and holding it near the sensor (always take care when using solvents, don't let children do this part, and make sure that your room is well-ventilated).</p> <p>In our <a href=„https://github.com/pimoroni/bme680/blob/master/examples/indoor-air-quality.py“>example</a> of how to convert the BME680's gas resistance readings into a percentage indoor air quality (IAQ), we took a background reading for 5 minutes, then set this as the optimal gas resistance reading (100%), and also factored in humidity with a weighting of 75:25 for gas:humidity, as humidity also has an effect on IAQ (a relative humidity of 40% is optimal).</p> <h2>Taking it further</h2> <p>Why not visualise the gas resistance readings with one of our <a href=„https://shop.pimoroni.com/search?q=leds&amp;hPP=10&amp;idx=shop.pimoroni.com.products&amp;p=0&amp;hFR%5Btags%5D%5B0%5D=LED&amp;hFR%5Bvendor%5D%5B0%5D=Pimoroni&amp;type=product&amp;is_v=1“>LED boards</a>? There's an <a href=„https://github.com/pimoroni/bme680/blob/master/examples/indoor-air-

quality.py“>example</a> in the BME680 Python library of how you might translate the gas resistance readings into a friendlier percentage air quality, which you could then map to a colour.</p> <p>The gas reading in the BME680 should be sensitive to a wide range of different gases, like methane and other volatile organic compounds (VOCs), carbon monoxide, ethanol, human breath and sweat, and more, so you can really have some fun creating fart detectors or even a crude breathalyser!</p> </html>