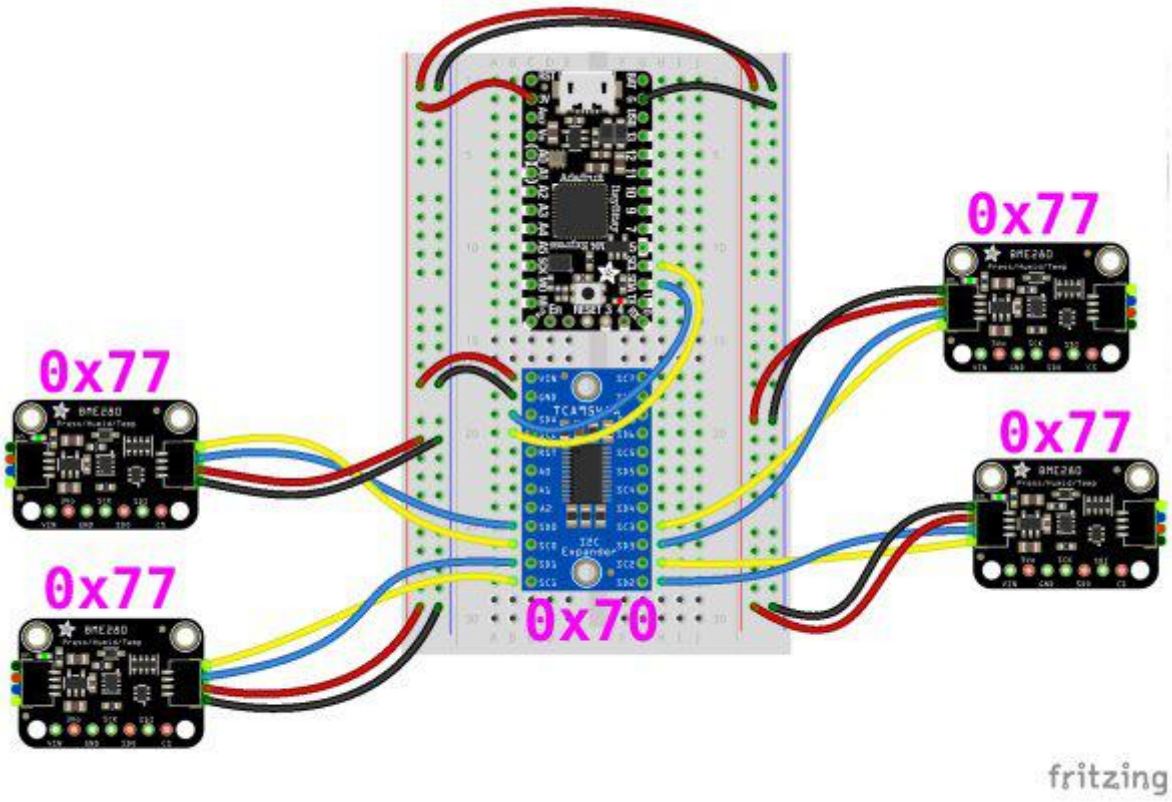
# Working with Multiple Same Address I2C Devices

Created by Carter Nelson

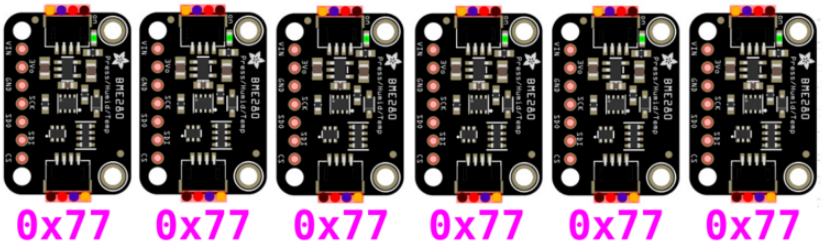



https://learn.adafruit.com/working-with-multiple-i2c-devices

Last updated on 2022-05-04 12:28:58 PM EDT

# Table of Contents

# Overview




A previous guide (https://adafru.it/YUF) discussed working with I2C devices in a general way, covering various topics. If you're new to I2C and need a broader overview, be sure to read that guide first, here's the link:

Working with I2C Devices

https://adafru.it/YUF

This guide goes more in depth on working with multiple copies of the same I2C device, which most likely have the same I2C address. Getting this general configuration working seems to be a common source of confusion.

As mentioned in the guide linked above, every I2C device on the I2C bus needs to have a unique address (https://adafru.it/ZYd). If you've only used a single I2C device, you may not even have realized this. Most drivers are written such that they use, without any further input from the user, a predefined default address. This may also have even been the case for using multiple, but different, I2C devices. In that case, each device's default address was different, so there was nothing else to do in terms of coding. Everything was just plug-and-play simple.


However, things get more interesting when address conflicts (https://adafru.it/ZYe) start happening. This is essentially unavoidable when attempting to use multiple copies of the same I2C device. As mentioned in the other guide, there are three main ways of dealing with this:

- Use an alternate address (if device allows)
- Use an I2C channel multiplexer if alternate address(es) not possible (recommended)
- Use alternate I2C ports (if desperate)

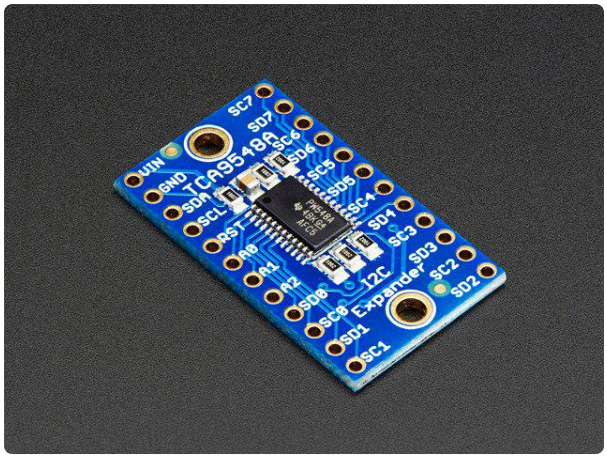
With any of these approaches, there is more work to be done. What is the alternate address? How do you actually "set" it? How do you change code to accommodate the alternate address? How do you incorporate using an I2C channel multiplexer? etc.

This guide aims to help answer those questions.

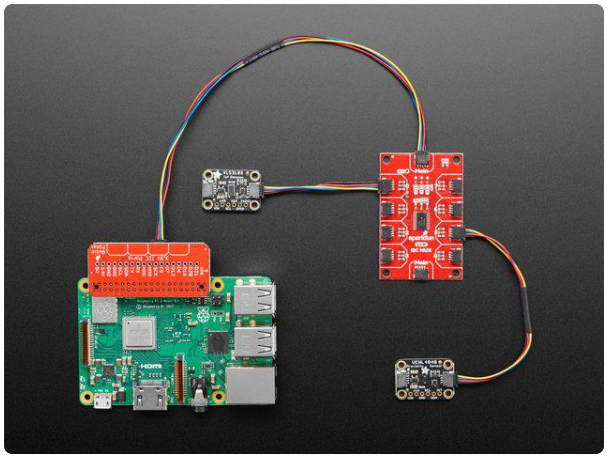
## Hardware

This guide uses the Adafruit BME280 breakout (https://adafru.it/y8f) in the examples as a representative I2C device. However, there is nothing special about this particular sensor. So it's not really "required hardware". The information here should generally apply to any I2C breakout.

When it comes to an I2C multiplexer, the TCA9548A is currently the best option. It's available in a couple of different breakouts:




TCA9548A I2C Multiplexer
You just found the perfect I2C sensor, and you want to wire up two or three or more of them to your Arduino when you realize "Uh oh, this chip has a fixed I2C address, and from...
https://www.adafruit.com/product/2717



SparkFun STEMMA QT / Qwiic TCA9548A Mux Breakout - 8 Channel
Do you have too many sensors with the same I2C address? Put them on the SparkFun Qwiic Mux Breakout to get them all talking on the same bus! The Qwiic Mux Breakout...
https://www.adafruit.com/product/4704

The Adafruit TCA9548A breakout (https://adafru.it/y6b)is used in this guide. However, the SparkFun breakout has the benefit of not needing any soldering. Everything can be connected using STEMMA QT cables.

Also, an ItsyBitsy M4 (https://adafru.it/BQC) is shown in the wiring diagrams. However, the information in this guide should apply to any Arduino or CircuitPython board with an I2C port. Which is pretty much most of them.

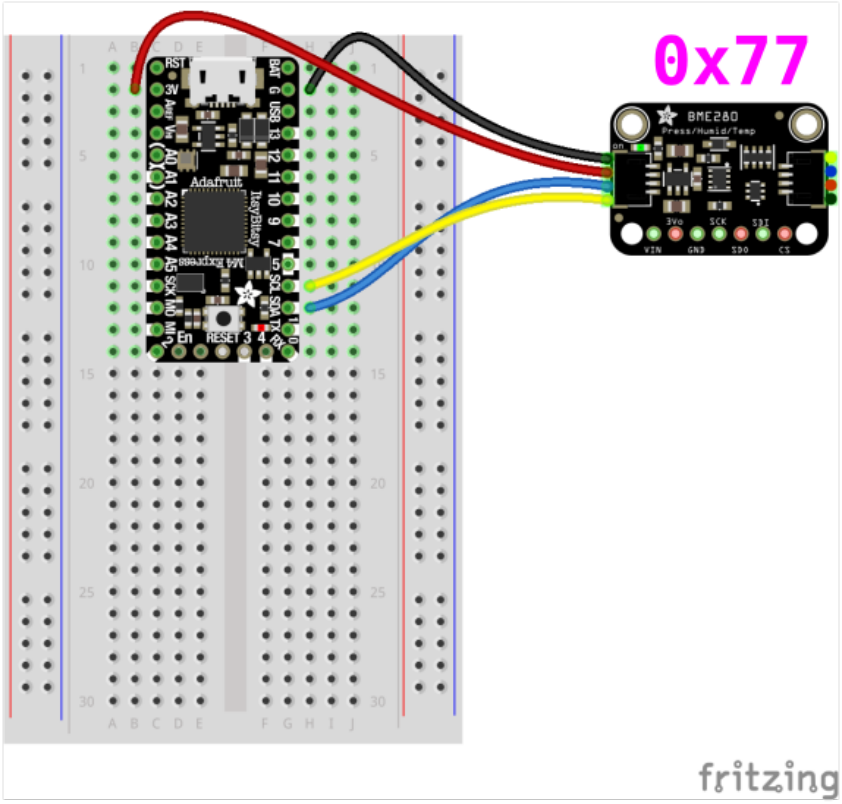
---

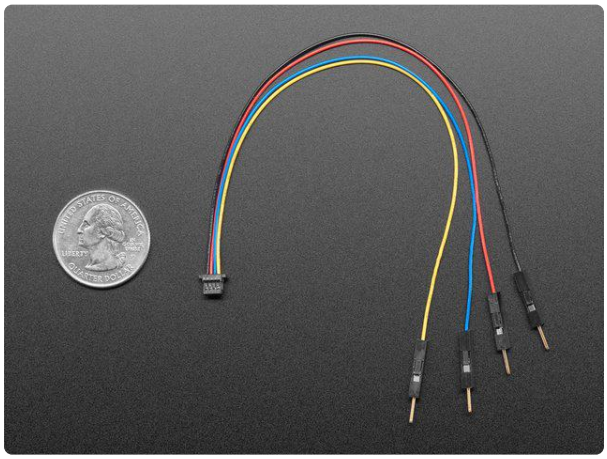
# Single Device using Default Settings

This is the trivial case. It should be possible to wire the breakout per its guide and run a simple example without any modifications. It should "just work". The I2C address itself never needs to be directly dealt with.

This configuration is covered here to simply provide a basic starting point. Here's the wiring:

The pink text is the default I2C address for this breakout board. Look on the back or check the vendor documentation to find the default!

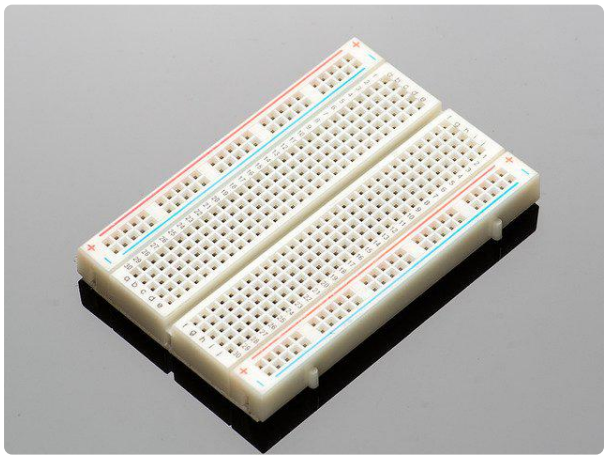


STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable

This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other. Compared with the...

https://www.adafruit.com/product/4209



Half-size breadboard

This is a cute half-size breadboard, good for small projects. It's 2.2" x 3.4" (5.5 cm x 8.5 cm) with a standard double-strip in the middle and two power rails on both...

https://www.adafruit.com/product/64

# Arduino

There's really not much to do other than run the basic example from the library, as is. No code changes should be needed. This is no different than what the BME280 primary guide has you do here:

BME280 Arduino Test

https://adafru.it/ZYf

Once the Adafruit BME280 library is installed, open the example in the Arduino IDE:

File->Examples->Adafruit_BME280->bme280test

and upload the sketch to the board. The output in the Serial Monitor should look like this:


# CircuitPython

Again, the basic example from the library should work as is without any code changes. This is no different than what the BME280 primary guide has you do here:

BME280 CircuitPython Test

https://adafru.it/FUP

With the bme280_simpletest.py (https://adafru.it/ZYA) example running, the output should look like this:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

Temperature: 19.3 C
Humidity: 46.9 %
Pressure: 1014.0 hPa
```

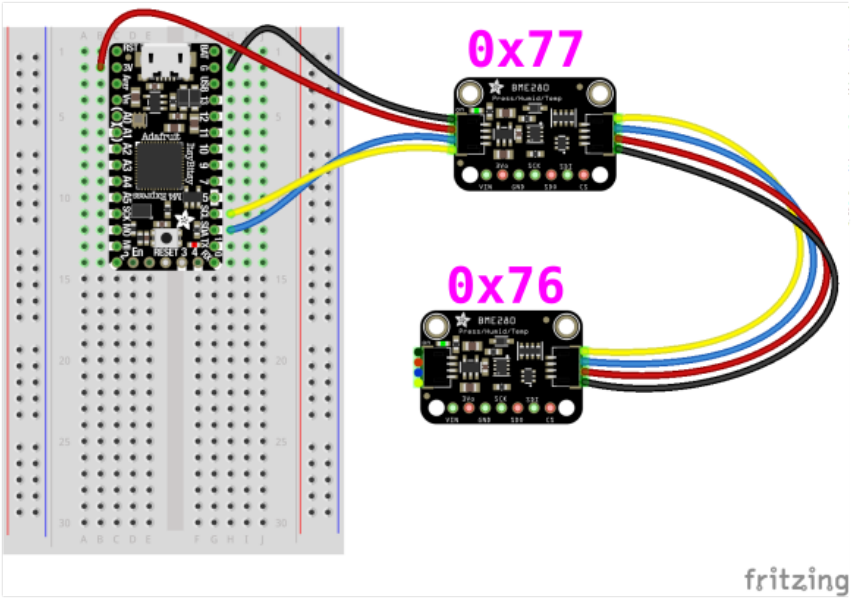
```
Altitude = -6.30 meters

Temperature: 19.3 C
Humidity: 46.8 %
Pressure: 1014.0 hPa
Altitude = -6.17 meters
```

# Two Devices using Alternate Address

OK, now let's try using two BME280s. Now there's some work to do. Both the hardware and the code will need to be changed.

Both BME280s can't use the default address of 0x77. Not all devices have this capability, but the BME280 allows setting a second I2C address ox 0x76. That allows for the following connection:

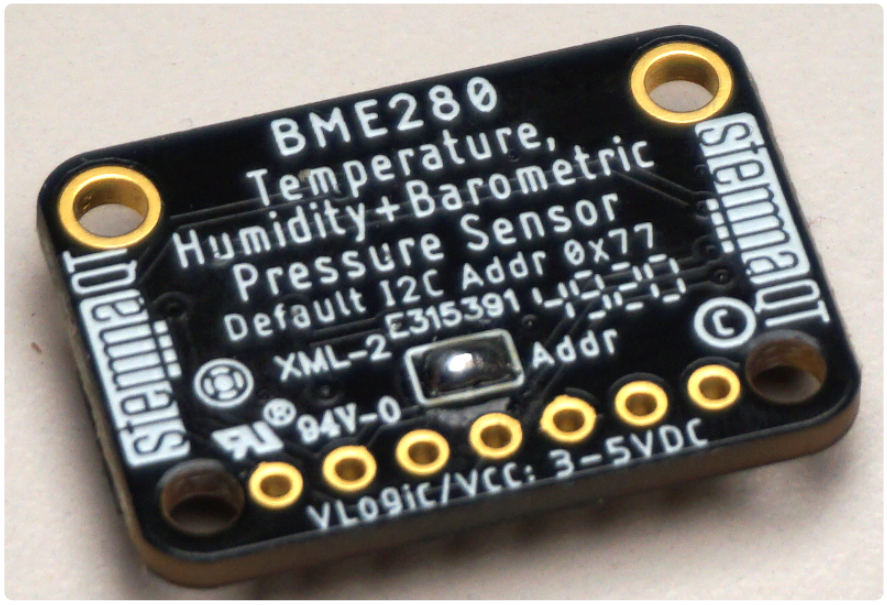



But how did we know the alternate address was 0x76? And how was the breakout altered to set the address to 0x76?

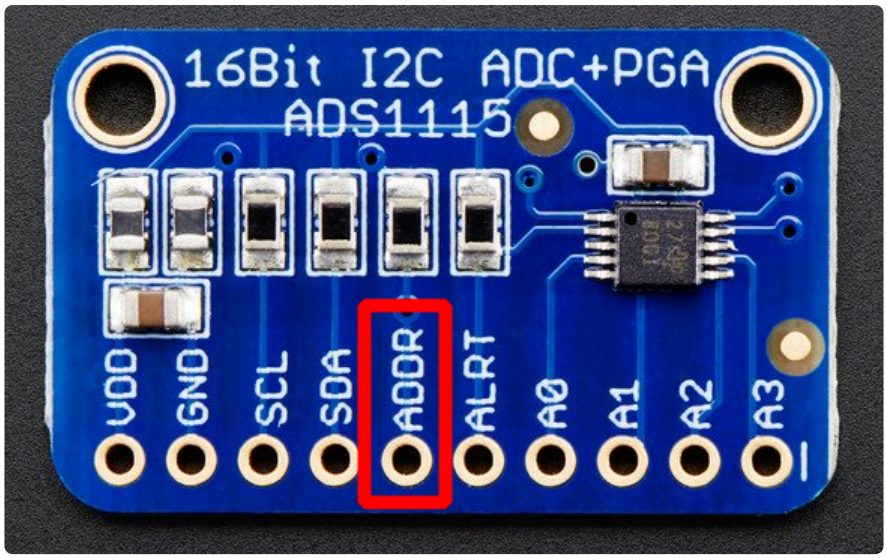
## Finding and Setting Alternate Addresses

How do you know if any given device has the ability to set an alternate I2C address? Ideally, it will be mentioned in the product guide or some other documentation. Sometimes this may required digging into the device's datasheet. Also, since the most common method of setting the alternate address is by using additional pins, there are some physical features to look for on any given breakout. For example, the BME280 has a solder pad for setting the alternate address on the backside of the board:


If nothing is done, the BME280 has the indicated default address of 0x77. By adding a blob of solder to electrically connect to the two copper pads, the alternate address of 0x76 is set. Here is what a blob of solder on the address jumper pad looks like:




Another place to look would be the row of header pins. For example, the ADS1105 and ADS1115 ADC breakouts have a dedicated pin for setting the alternate address.




This is done since setting alternate addresses for the ADS1015/1115 is not as simple as making/breaking a single connection - which is what solder pad jumpers are used for. A total of four different addresses can be set depending on what the ADDR pin is connected to.

**Table 5. ADDR Pin Connection and Corresponding Slave Address**

| ADDR PIN | SLAVE ADDRESS |
|---|---|
| Ground | 1001000 |
| VDD | 1001001 |
| SDA | 1001010 |
| SCL | 1001011 |

Trying to do that with solder jumpers would be a bit of a mess.

## Coding for Alternate Address Usage

Once the hardware has been modified to set an alternate I2C address, what are the required code changes? To use an alternate, non-default address, it must be explicitly called out in user code. But exactly where and how is that done? It's different for Arduino and CircuitPython.

### Arduino

For Adafruit Arduino libraries, the alternate address is passed in when calling `begin()` on the sensor object. This code is typically placed in an Arduino sketch's `setup()` function. For example, to specify the alternate address of 0x76 for the BME280:

```
bme.begin(0x76);  // specify the alternate address of 0x76
```

In Arduino, FOR MOST LIBRARIES, you can specify the I2C address in the call to the device's begin() library function.

NOTE: There can be some library-to-library variability in the specific format for calling `begin()`. So be sure to look at the associated library documents.

### CircuitPython

For Adafruit CircuitPython libraries, the alternate address is passed in when creating the instance for the sensor. For example, to specify the alternate address of 0x76 for the BME280:

```
bme = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)
```

In CircuitPython, specify the I2C address as a parameter when creating the device instance.

NOTE: Sometimes a parameter name, like `addr` or `address` is used.

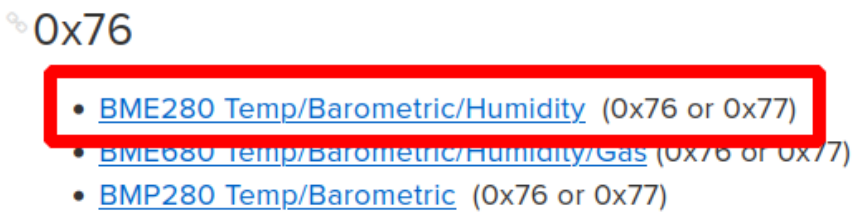
## Consult The List

The following guide attempts provide a list of I2C addresses for popular I2C devices:

I2C Addresses - The List

https://adafru.it/EnK

There are a lot. One could text search on that page and see if a given device is there. For example, search for "BME280" to find the entry:




And now it's known that the BME280 can have an I2C address of either 0x76 or 0x77.

However, this list requires manually updating. So the page may or may not have the sensor you're looking for.

## When In Doubt, Scan

Running an I2C scan (https://adafru.it/VBu) is a good way to sanity check the setup. Not only will it report the I2C addresses for all discovered devices, it's also a good way to verify the I2C connections themselves are OK. There's a dedicated guide on I2C scanning here:

How to Scan and Detect I2C Addresses

https://adafru.it/VBu

There are examples for scanning with Arduino, CircuitPython, and on Raspberry Pi. Here is what typical Arduino scan output looks like:


There's something at 0x76 and 0x77! That's the two BME280 breakouts as shown in the wiring diagram above. One is unaltered and has an I2C address of 0x77. One has the solder jumper closed and has an I2C address of 0x76.

---

# Arduino

OK, to actually code up a simple example using two BME280's, one without any changes having the default address of 0x77 and one with the solder pad connected to set the alternate address of 0x76.

Here's the sketch code:

```
// SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT
```

```
#include <Adafruit_BME280.h>

// For each device, create a separate instance.
Adafruit_BME280 bme1;  // BME280 #1 @ 0x77
Adafruit_BME280 bme2;  // BME280 #2 @ 0x76

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println(F("Two BME280 Example"));

  // NOTE: There's no need to manually call Wire.begin().
  // The BME280 library does that in its begin() method.

  // In the call to begin, pass in the I2C address.
  // If left out, the default address is used.
  // But also OK to just be explicit and specify.
  bme1.begin(0x77);  // address = 0x77 (default)
  bme2.begin(0x76);  // address = 0x76
}


void loop() {
  float pressure1, pressure2;

  // Read each device separately
  pressure1 = bme1.readPressure();
  pressure2 = bme2.readPressure();

  Serial.println("-------------------------------------");
  Serial.print("BME280 #1 Pressure = "); Serial.println(pressure1);
  Serial.print("BME280 #2 Pressure = "); Serial.println(pressure2);

  delay(1000);
}
```

With that sketch running on the Arduino board, the output in the Serial Monitor will look like this:


Note how for each device, there is a separate instance created:

```
Adafruit_BME280 bme1;  // BME280 #1 @ 0x77
Adafruit_BME280 bme2;  // BME280 #2 @ 0x76
```

And then for each, the `begin()` function is called and the address is specified:

```
bme1.begin(0x77);  // address = 0x77 (default)
bme2.begin(0x76);  // address = 0x76
```

Here we intentionally specify the default 0x77 address just to be explicit.

Once that is taken care of, the two instances can be used to directly read the sensor values:

```
pressure1 = bme1.readPressure();
pressure2 = bme2.readPressure();
```

---

# CircuitPython

Let's do the same thing in CircuitPython.

Here's the code:

```
# SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
from adafruit_bme280 import basic as adafruit_bme280

# Get the board's default I2C port
i2c = board.I2C()

#--------------------------------------------------------------------
# NOTE!!! This is the "special" part of the code
#
# Create each sensor instance
# If left out, the default address is used.
# But also OK to be explicit and specify address.
bme1 = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x77)  # address = 0x77
bme2 = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)  # address = 0x76
#--------------------------------------------------------------------

print("Two BME280 Example")

while True:
    # Access each sensor via its instance
    pressure1 = bme1.pressure
    pressure2 = bme2.pressure

    print("-"*20)
    print("BME280 #1 Pressure =", pressure1)
    print("BME280 #2 Pressure =", pressure2)

    time.sleep(1)
```

With that code running on the CircuitPython board, the output will look like this:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Two BME280 Example
--------------------
BME280 #1 Pressure = 1013.98
BME280 #2 Pressure = 1013.16
--------------------
BME280 #1 Pressure = 1013.98
BME280 #2 Pressure = 1013.13
--------------------
BME280 #1 Pressure = 1014.0
BME280 #2 Pressure = 1013.16
--------------------
BME280 #1 Pressure = 1014.01
BME280 #2 Pressure = 1013.2
```

These are the two important lines:

```
bme1 = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x77)  # address = 0x77
bme2 = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)  # address = 0x76
```

They create a separate sensor instance for each BME280 and specify the I2C address for each. We intentionally specify the default 0x77 address just to be explicit.

After that, each can be used to read the sensor values:

```
pressure1 = bme1.pressure
pressure2 = bme2.pressure
```

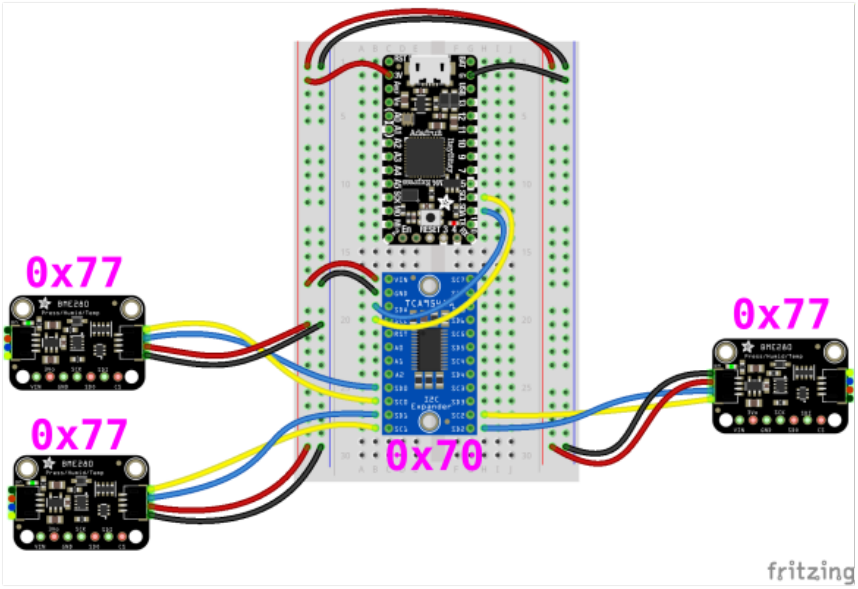
So things are pretty easy if alternate addresses can be used.

---

# Three Devices using Multiplexer

Now to look at using three BME280s. There are no easy options, since the BME280 only has two available I2C addresses. So now to introduce the TCA9548A I2C multiplexer (https://adafru.it/y6b). Also, to help illustrate the use of the TCA9548A, this will ignore the BME280's alternate address. Therefore, this example is exactly like what one would deal with when trying to use three copies of any I2C device with a sin gle fixed address.

Here is the wiring diagram for the setup. Note that the TCA9548A itself has an I2C address of 0x70. Each BME280 has the same address of 0x77.




## The TCA9548A Multiplexer

This is a simple device. It has only one trick up its sleeve. It has one input and eight outputs. The only I2C "command" it accepts is one to set what outputs are active. That's it. That's all it does.

It is an I2C device though, so does have an I2C address. The address can be changed, but in this guide we'll only use its default 0x70 address. No other device can have the same address as the TCA9548A. So the basic rule about unique addresses still applies.

### Changing Output Channels

While the TCA9548A can have more than one output channel active at a time, this only considers using a single channel at a time. The one "command" that the TCA9548A accepts is a single byte value, where each bit represents an output channel:

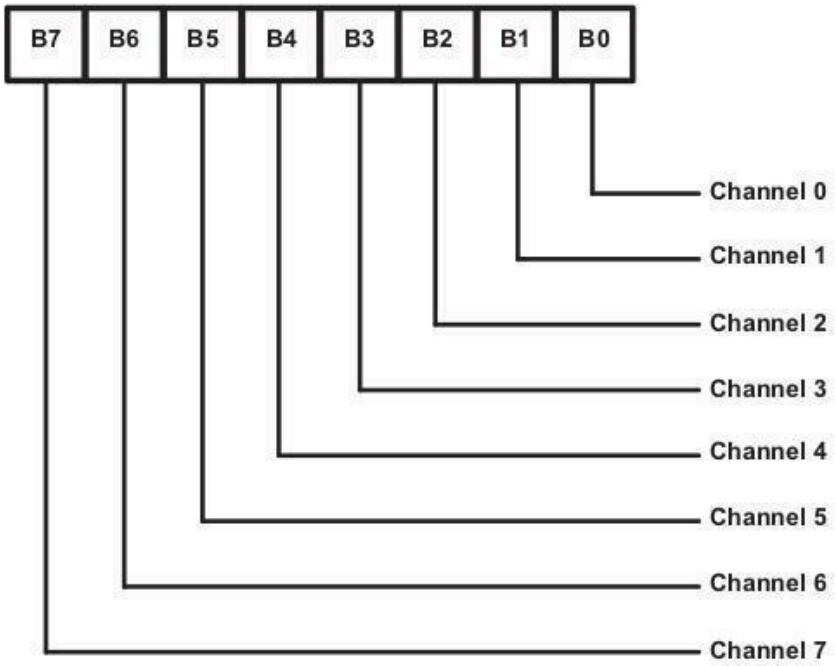



**Figure 11. Control Register**

If the bit is 1, the channel is active. If the bit is 0, the channel is not active. Super simple.

The easiest way to generate the proper byte to activate a single channel is to left shift (https://adafru.it/ZYB) 1 by the channel number. An example Arduino code snippet to do this would look like:

```
Wire.beginTransmission(TCA_ADDRESS);
Wire.write(1 &lt;&lt; CHANNEL_NUMBER);
Wire.endTransmission();
```

Where `TCA_ADDRESS` would be the 0x70 (or other) address of the TCA9548A and `CHANNEL_NUMBER` would be an integer value 0 to 7 for the desired output channel. The `<<` is the left shift operator.

This is the exact same code as shown in the TCA9548A guide (https://adafru.it/ZYC) and will also be used here in the Arduino examples. There is no dedicated Arduino library for the TCA9548A. Channel switching must be done manually in Arduino sketch code.

When using CircuitPython, things are a little fancier. There is a CircuitPython library for the TCA9548A (https://adafru.it/ZYD). Under the hood, it's no different than the Arduino example above. The left shifting can be seen here (https://adafru.it/ZYE). The

neat thing about the CircuitPython library is that it can take care of sending the channel switch byte to the TCA9548A automatically.

---

# Arduino

OK, now to code up the example with a TCA9548A multiplexer and three BME280s, all with I2C addresses of 0x77.

Here's the code:

```
// SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_BME280.h>

#define TCAADDR 0x70

// For each device, create a separate instance.
Adafruit_BME280 bme1;  // BME280 #1
Adafruit_BME280 bme2;  // BME280 #2
Adafruit_BME280 bme3;  // BME280 #3

// Helper function for changing TCA output channel
void tcaselect(uint8_t channel) {
  if (channel > 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << channel);
  Wire.endTransmission();
}

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println(F("Three BME280 Example"));

  // NOTE!!! VERY IMPORTANT!!!
  // Must call this once manually before first call to tcaselect()
  Wire.begin();

  // Before using any BME280, call tcaselect to set the channel.
  tcaselect(0);      // TCA channel for bme1
  bme1.begin();      // use the default address of 0x77

  tcaselect(1);      // TCA channel for bme2
  bme2.begin();      // use the default address of 0x77

  tcaselect(2);      // TCA channel for bme3
  bme3.begin();      // use the default address of 0x77
}


void loop() {
  float pressure1, pressure2, pressure3;

  // Read each device separately
  tcaselect(0);
  pressure1 = bme1.readPressure();
  tcaselect(1);
  pressure2 = bme2.readPressure();
  tcaselect(2);
```

```
  pressure3 = bme3.readPressure();

  Serial.println("-----------------------------------");
  Serial.print("BME280 #1 Pressure = "); Serial.println(pressure1);
  Serial.print("BME280 #2 Pressure = "); Serial.println(pressure2);
  Serial.print("BME280 #3 Pressure = "); Serial.println(pressure3);

  delay(1000);
}
```

With that sketch running on the Arduino board, the output in the Serial Monitor will look like this:


Similar to the two BME280s example, a separate instance is created for each sensor:

```
Adafruit_BME280 bme1;  // BME280 #1
Adafruit_BME280 bme2;  // BME280 #2
Adafruit_BME280 bme3;  // BME280 #3
```

A helper function is used to make switching TCA9548A channels easy. Note that the code is no different than what was discussed above about setting TCA9548A output channels:

```
void tcaselect(uint8_t channel) {
  if (channel &gt; 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 &lt;&lt; channel);
  Wire.endTransmission();
}
```

While simple, this function is using the Wire bus directly. Therefore, it is very important to remember to call `Wire.begin();` before the first call to `tcaselect()`. It only needs to be called once.

```
// NOTE!!! VERY IMPORTANT!!!
// Must call this once manually before first call to tcaselect()
Wire.begin();
```

Also important is noting that `tcaselect()` must be called each time to set the output channel to the specific BME280. This is done in `setup()` before calling `begin()` on each sensor:

```
tcaselect(0);      // TCA channel for bme1
bme1.begin();      // use the default address of 0x77

tcaselect(1);      // TCA channel for bme2
bme2.begin();      // use the default address of 0x77

tcaselect(2);      // TCA channel for bme3
bme3.begin();      // use the default address of 0x77
```

And also in the `loop()` before reading each sensor:

```
tcaselect(0);
pressure1 = bme1.readPressure();
tcaselect(1);
pressure2 = bme2.readPressure();
tcaselect(2);
pressure3 = bme3.readPressure();
```

Yes, that is a bit klunky. Code lines to deal with the TCA9548A must be interleaved in with the sensor reading code. Think in terms of "I want to use the sensor on TCA9548A channel X, so first must switch the TCA9548A to channel X." The `tcaselect()` function is what does that switching.

---

# CircuitPython

Here is the CircuitPython code for a TCA9548A and three BME280s:

```
# SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_tca9548a
from adafruit_bme280 import basic as adafruit_bme280

# Create I2C bus as normal
i2c = board.I2C()

# Create the TCA9548A object and give it the I2C bus
tca = adafruit_tca9548a.TCA9548A(i2c)

#-----------------------------------------------------------------
# NOTE!!! This is the "special" part of the code
#
# Create each BME280 using the TCA9548A channel instead of the I2C object
bme1 = adafruit_bme280.Adafruit_BME280_I2C(tca[0])   # TCA Channel 0
bme2 = adafruit_bme280.Adafruit_BME280_I2C(tca[1])   # TCA Channel 1
bme3 = adafruit_bme280.Adafruit_BME280_I2C(tca[2])   # TCA Channel 2
#-----------------------------------------------------------------
```

```
print("Three BME280 Example")

while True:
    # Access each sensor via its instance
    pressure1 = bme1.pressure
    pressure2 = bme2.pressure
    pressure3 = bme3.pressure

    print("-"*20)
    print("BME280 #1 Pressure =", pressure1)
    print("BME280 #2 Pressure =", pressure2)
    print("BME280 #3 Pressure =", pressure3)

    time.sleep(1)
```

With that code running on the CircuitPython board, the output will look like this:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Three BME280 Example
--------------------
BME280 #1 Pressure = 1013.96
BME280 #2 Pressure = 1013.72
BME280 #3 Pressure = 1013.76
--------------------
BME280 #1 Pressure = 1013.97
BME280 #2 Pressure = 1013.73
BME280 #3 Pressure = 1013.77
--------------------
BME280 #1 Pressure = 1013.98
BME280 #2 Pressure = 1013.73
BME280 #3 Pressure = 1013.73
```

The TCA9548A itself is setup like any other I2C device:

```
tca = adafruit_tca9548a.TCA9548A(i2c)
```

The board's I2C bus ( `i2c` ) is passed in. The I2C address for the TCA9548A would also be specified here. But in this case we leave it out to show how to use with the default 0x70 address. Remember - that's the address of the TCA9548A.

The important difference to note is what is being passed in when creating each instance of the BME280 device:

```
bme1 = adafruit_bme280.Adafruit_BME280_I2C(tca[0])   # TCA Channel 0
bme2 = adafruit_bme280.Adafruit_BME280_I2C(tca[1])   # TCA Channel 1
bme3 = adafruit_bme280.Adafruit_BME280_I2C(tca[2])   # TCA Channel 2
```

Instead of passing in the I2C bus ( `i2c` ), the `[]` operator is used on the `tca` instance to access and specify that output channel. So `tca[0]` is channel 0 of the TCA9548A, etc.

This is the fancy thing that the Adafruit CircuitPython TCA9548A library does. Because of this feature, once the initial setup is done, the instances can be used in a normal way:

```
pressure1 = bme1.pressure
pressure2 = bme2.pressure
pressure3 = bme3.pressure
```

There's no need to worry about dealing with the TCA9548A directly, changing its output channel, etc.

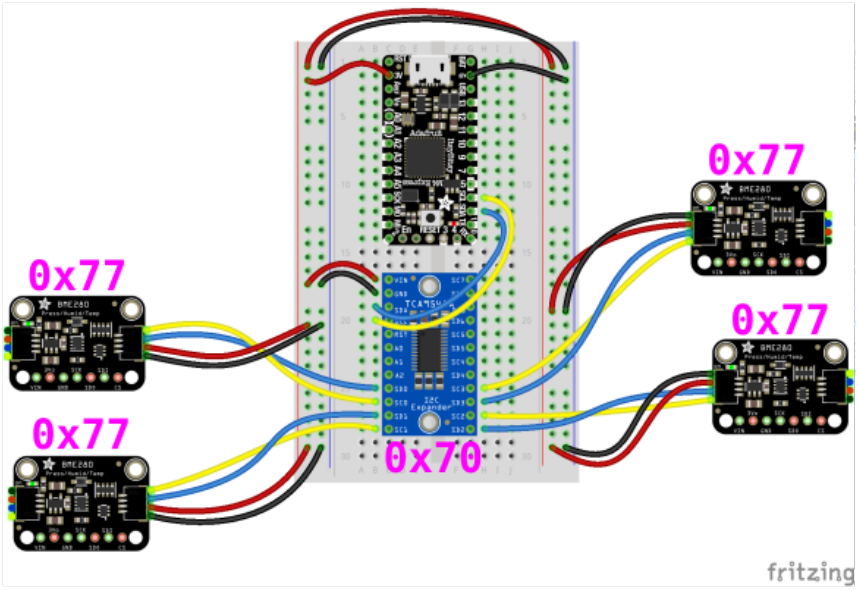
Exactly how the library does this is beyond the scope of this guide. But it relies on some Python specific tricks, which is why a similar capability is lacking in Arduino land.

---

# Four Devices using Multiplexer

This is really no different than the example with three devices. There's just one more BME280 device added. But hopefully this example helps reinforce the idea that adding yet-another-same-address-device is a simple matter of connecting it to an available channel on the TCA9548A and adding a few more lines of code. So by looking at the three BME280 example and then the four BME280 example, one can expand this to five, six, etc. BME280s.

Here's the setup:



Why stop at 4 BME280s? Or even 8? A single TCA9548A can support up to 8 total same address devices. By using multiple TCA9548As, each with its own I2C address, more than 8 same address devices can be used. There are 8 total settable addresses for the TCA9548A, with values 0x70 to 0x77. So the grand total of same address devices that could be used is 8 TCA9548As x 8 output channels each = 64.

---

# Arduino

OK, now to code up the example with a TCA9548A multiplexer and four BME280s, all with I2C addresses of 0x77.

Here's the code:

```
// SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_BME280.h>

#define TCAADDR 0x70

// For each device, create a separate instance.
Adafruit_BME280 bme1;  // BME280 #1
Adafruit_BME280 bme2;  // BME280 #2
Adafruit_BME280 bme3;  // BME280 #3
Adafruit_BME280 bme4;  // BME280 #4

// Helper function for changing TCA output channel
void tcaselect(uint8_t channel) {
  if (channel > 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << channel);
  Wire.endTransmission();
}

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println(F("Four BME280 Example"));

  // NOTE!!! VERY IMPORTANT!!!
  // Must call this once manually before first call to tcaselect()
  Wire.begin();

  // Before using any BME280, call tcaselect to select its output channel
  tcaselect(0);      // TCA channel for bme1
  bme1.begin();      // use the default address of 0x77

  tcaselect(1);      // TCA channel for bme2
  bme2.begin();      // use the default address of 0x77

  tcaselect(2);      // TCA channel for bme3
  bme3.begin();      // use the default address of 0x77

  tcaselect(3);      // TCA channel for bme4
  bme4.begin();      // use the default address of 0x77
}
```

```
void loop() {
  float pressure1, pressure2, pressure3, pressure4;

  // Read each device separately
  tcaselect(0);
  pressure1 = bme1.readPressure();
  tcaselect(1);
  pressure2 = bme2.readPressure();
  tcaselect(2);
  pressure3 = bme3.readPressure();
  tcaselect(3);
  pressure4 = bme4.readPressure();

  Serial.println("-----------------------------------");
  Serial.print("BME280 #1 Pressure = "); Serial.println(pressure1);
  Serial.print("BME280 #2 Pressure = "); Serial.println(pressure2);
  Serial.print("BME280 #3 Pressure = "); Serial.println(pressure3);
  Serial.print("BME280 #4 Pressure = "); Serial.println(pressure4);

  delay(1000);
}
```

With that sketch running on the Arduino board, the output in the Serial Monitor will look like this:


Hopefully by comparing the 4xBME280 code to the previous 3xBME280 code example, the basic code pattern can be seen. It's essentially just a copy-paste of the same code to add one more instance:

```
Adafruit_BME280 bme4;  // BME280 #4
```

And then call `tcaselect()` same as done for the other BME280s:

```
tcaselect(3);      // TCA channel for bme4
bme4.begin();      // use the default address of 0x77
```

Don't forget to call Wire.begin() once before calling tcaselect().

Don't forget to call tcaselect() before accessing each sensor.

# CircuitPython

Here's the same thing for CircuitPython:

```
# SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_tca9548a
from adafruit_bme280 import basic as adafruit_bme280

# Create I2C bus as normal
i2c = board.I2C()

# Create the TCA9548A object and give it the I2C bus
tca = adafruit_tca9548a.TCA9548A(i2c)

#-----------------------------------------------------------------------
# NOTE!!! This is the "special" part of the code
#
# Create each BME280 using the TCA9548A channel instead of the I2C object
bme1 = adafruit_bme280.Adafruit_BME280_I2C(tca[0])   # TCA Channel 0
bme2 = adafruit_bme280.Adafruit_BME280_I2C(tca[1])   # TCA Channel 1
bme3 = adafruit_bme280.Adafruit_BME280_I2C(tca[2])   # TCA Channel 2
bme4 = adafruit_bme280.Adafruit_BME280_I2C(tca[3])   # TCA Channel 3
#-----------------------------------------------------------------------

print("Four BME280 Example")

while True:
    # Access each sensor via its instance
    pressure1 = bme1.pressure
    pressure2 = bme2.pressure
    pressure3 = bme3.pressure
    pressure4 = bme4.pressure

    print("-"*20)
    print("BME280 #1 Pressure =", pressure1)
    print("BME280 #2 Pressure =", pressure2)
    print("BME280 #3 Pressure =", pressure3)
    print("BME280 #4 Pressure =", pressure4)

    time.sleep(1)
```

With that code running on the CircuitPython board, the output will look like this:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Four BME280 Example
--------------------
BME280 #1 Pressure = 1014.0
BME280 #2 Pressure = 1013.72
BME280 #3 Pressure = 1013.73
BME280 #4 Pressure = 1013.75
--------------------
BME280 #1 Pressure = 1013.99
BME280 #2 Pressure = 1013.72
BME280 #3 Pressure = 1013.76
BME280 #4 Pressure = 1013.73
--------------------
```

```
BME280 #1 Pressure = 1013.98
BME280 #2 Pressure = 1013.71
BME280 #3 Pressure = 1013.74
BME280 #4 Pressure = 1013.75
```

Again, there is little more done compared to the 3xBME280s example other than adding a new instance for the 4th sensor, specifying the TCA9548A channel:

```
bme4 = adafruit_bme280.Adafruit_BME280_I2C(tca[3])   # TCA Channel 3
```

And then using it like the others:

```
pressure4 = bme4.pressure
```