SmartElex Atmospheric Sensor Breakout Board - BME280

The BME280 Breakout Board is the easy way to measure pressure and humidity, and without taking up a lot of room. It gives you easy to solder 0.1" headers, runs I2C or SPI, takes measurements at less than 1mA and idles less than 5uA (yes, microamps!).

The BME280 can be used to take pressure, humidity, and temperature readings. Use the data to get relative altitude changes, or absolute altitude if the locally reported barometric pressure is known.

Ranges:

- Temp: -40C to 85C
- Humidity: 0 - 100% RH, =-3% from 20-80%
- Pressure: 30,000Pa to 110,000Pa, relative accuracy of 12Pa, absolute accuracy of 100Pa
- Altitude: 0 to 30,000 ft (9.2 km), relative accuracy of 3.3 ft (1 m) at sea level, 6.6 (2 m) at 30,000 ft.

Hardware Overview

The BME280 Breakout board has 10 pins, but no more than 6 are used at a single time.

Use one header for I2C connections, **or** the other for SPI connections -- no need to use both!

The left side of the board are power, ground, and I^2C pins.

Pin Label	Pin Function	Notes
GND	Ground	0V voltage supply.
3.3v	Power Supply	Supply voltage to the chip. Should be regulated between **1.8V and 3.6V**.
SDA	Data	I^2C: Serial data (bi-directional)
SCL	Serial Clock	I^2C serial clock.

The remaining pins are broken out on the other side. These pins break out SPI functionality and have another power and ground.

Pin Label	Pin Function	Notes
GND	Ground	0V voltage supply.
3.3v	Power Supply	Supply voltage to the chip. Should be regulated between **1.8V and 3.6V**.
SCK	Clock	Clock line, 3.6V max
SDO	Data out	Data comming out of the BME280 (MISO)

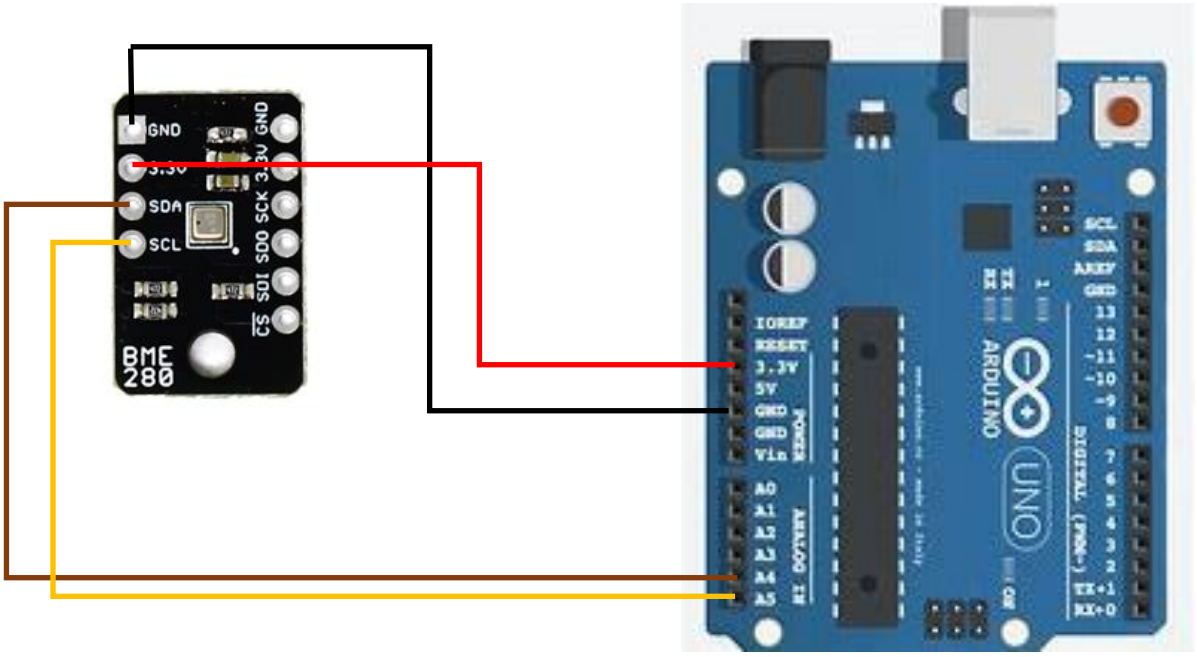
SDI	Data in	Data going into the BME280, 3.6V max (MOSI)
!CS	Chip Select (Slave Select)	Active low chip select, 3.6V max

On the other side of the board you'll find all the configuration jumpers. Pull-ups can be left connected even when using SPI mode, so you'll probably never have to touch these. If you do, here's what they're for.

Jumper Label	Jumper Function	Notes
ADR:	I²C Address	Select between addresses 0x77 (default, '1' side) and 0x76 by slicing the trace and bridging the '0' side. Controls the least significant bit.
CS PU	SPI chip select pull-up	Connects a 4.7k resistor to the CS line to make sure it is idle high. Can be disconnected by slicing between the jumper pads.
I²C	I²C pull-ups	Connects the I²C pull-up resistors to 3.3V. Cut the trace to disconnect them if necessary.

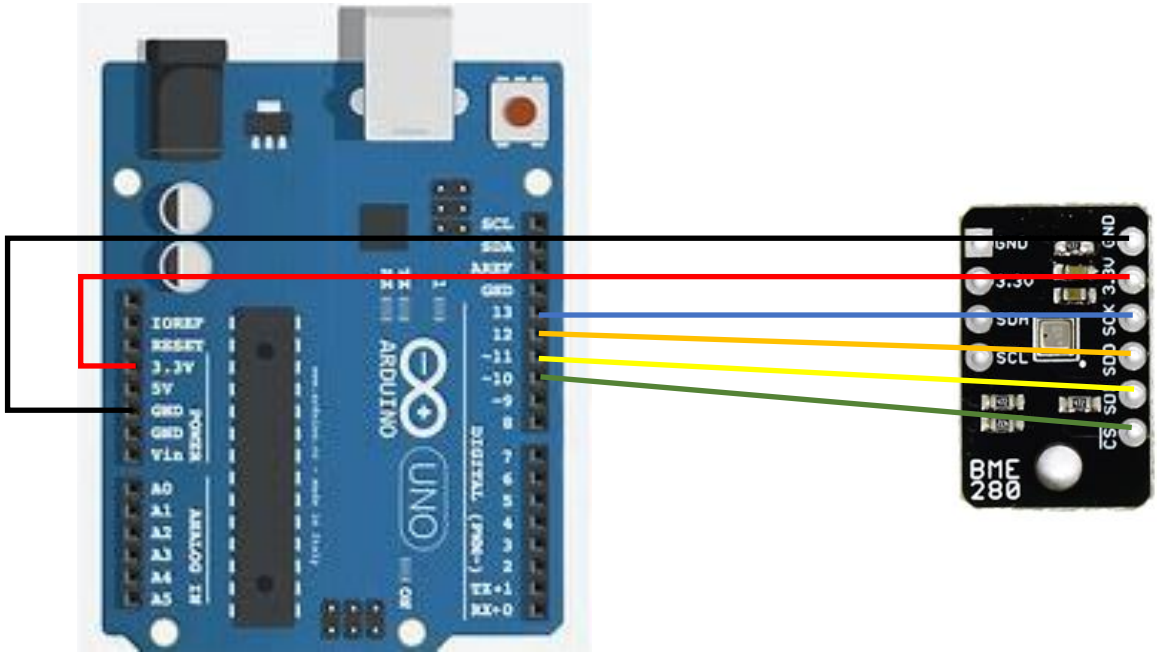
I²C Connection

The sensor pulls the I²C lines to 3.3V, so they can be directly connect to the UNO's A4/A5 pins, or the SDA/SCL pins (as long as they're configured by Wire). Make sure to power the sensor from 3.3v! The power and ground pins are connected, so you only need to connect to one side.



BME280	Arduino
SCL	SCL(A5)
SDA	SDA(A4)
3.3v	3.3v
GND	GND

SPI Connection



Arduino	BME280
D13	SCK
D12	SDO
D11	SDI
D10	CS
3.3v	3.3v
GND	GND

Install the BME280 library from library manager

Functions of the Arduino Library

Let's get started by looking at the functions that set up the BME280 Atmospheric Sensor:

Class

In the global scope, construct your sensor object (such as `mySensor` or `pressureSensorA`) without arguments.
`BME280 mySensor;`

Object Parameters and setup()

Rather that passing a bunch of data to the constructor, configuration is accomplished by setting the values of the BME280 type in the `setup()` function. They are exposed by being `public:` so use the `myName.aVariable = someValue;` syntax.
Settable variables of the class BME280:

```
COPY CODE//Main Interface and mode settings
uint8_t commInterface;
uint8_t I2CAddress;
uint8_t chipSelectPin;

uint8_t runMode;
uint8_t tStandby;
uint8_t filter;
uint8_t tempOverSample;
uint8_t pressOverSample;
uint8_t humidOverSample;
```

Functions

.begin();
Initialize the operation of the BME280 module with the following steps:

- Starts up the wiring library for I^2C by default
- Checks/Validates BME280 chip ID
- Reads compensation data
- Sets default settings from table
- Sets operational mode to *Normal Mode*

Output: uint8_t

Returns the BME280 chip ID stored in the ID register.

.begin() Needs to be run once during the setup, or after any settings have been modified. In order to let the sensor's configuration take place, the BME280 requires a minimum time of about 2 ms in the sketch before you take data.

.beginSPI(uint8_t csPin);
Begins communication with the BME280 over an SPI connection.
Input: uint8_t

csPin: Digital pin used for the CS.

Output: Boolean

True: Connected to sensor.
False: Unable to establish connection.

.beginI2C(TwoWire &wirePort); or **.beginI2C(SoftwareWire &wirePort);**
Begins communication with the BME280 over an I^2C connection. If #ifdef SoftwareWire_h is defined, then a software I^2C connection is used.
Input: &wirePort

&wirePort: Port for the I^2C connection.

Output: Boolean

True: Connected to sensor.
False: Unable to establish connection.

.setMode(uint8_t mode);
Sets the operational mode of the sensor. (*For more details, see section 3.3 of the datasheet.*)

Input: uint8_t

- **0:** Sleep Mode
- **1:** Forced Mode
- **3:** Normal Mode

.getMode();
Returns the operational mode of the sensor.
Output: uint8_t

- **0:** Sleep Mode
- **1:** Forced Mode
- **3:** Normal Mode

.setStandbyTime(uint8_t timeSetting);
Sets the standby time of the cycle time. (*For more details, see section 3.3 and Table 27 of the datasheet.*)
Input: uint8_t

- **0:** 0.5ms
- **1:** 62.5ms
- **2:** 125ms
- **3:** 250ms
- **4:** 500ms
- **5:** 1000ms
- **6:** 10ms
- **7:** 20ms

.setFilter(uint8_t filterSetting)
Sets the time constant of the IIR filter, which slows down the response time of the sensor inputs based on the number of samples required. (*For more details, see section 3.4.4, Table 6, and Figure 7 of the datasheet.*)
Input: uint8_t

- **0:** filter off
- **1:** coefficient of 2
- **2:** coefficient of 4
- **3:** coefficient of 8
- **4:** coefficient of 16

.setTempOverSample(uint8_t overSampleAmount);
Sets the oversampling option (osrs_t) for the temperature measurements. (*Directly influences the noise and resolution of the data.*)

Input: uint8_t

0: turns off temperature sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

.setPressureOverSample(uint8_t overSampleAmount);
Sets the oversampling option (osrs_p) for the pressure measurements. (*Directly influences the noise and resolution of the data.*)
Input: uint8_t

0: turns off pressure sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

.setHumidityOverSample(uint8_t overSampleAmount);
Sets the oversampling option (osrs_h) for the humidity measurements. (*Directly influences the noise of the data.*)
Input: uint8_t

0: turns off humidity sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

.setI2CAddress(uint8_t address);
Changes the I^2C address stored in the library to access the sensor.
Input: uint8_t

address: The new I^2C address.

.isMeasuring();
Checks the measuring bit of the status register for if the device is taking measurement.
Output: Boolean

True: A conversion is running.
False: The results have been transferred to the data registers.

.reset();
Soft resets the sensor. (*If called, the begin function must be called before using the sensor again.*)

.readFloatPressure();
Reads raw pressure data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)
Output: float

Returns pressure in Pa.

.readFloatHumidity();
Reads raw humidity data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)
Output: float

Returns humidity in %RH.

.readTempC();
Reads raw temperature data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)
Output: float

Returns temperature in Celsius.

.readTempF();
Reads raw temperature data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)
Output: float

Returns temperature in Fahrenheit.

Example Code

The examples are selectable from the drop-down menu in the Arduino IDE, or they will run stand-alone if you put the contents of the libraries /src dirctory in with the example.ino file.

```
#include <Wire.h>
#include "SparkFunBME280.h"
BME280 mySensor;
void setup()
{
  Serial.begin(115200);
  Serial.println("Reading basic values from BME280");
  Wire.begin();
  if (mySensor.beginI2C() == false) //Begin communication over I2C
  {
    Serial.println("The sensor did not respond. Please check wiring.");
    while(1); //Freeze
  }
}
void loop()
{
  Serial.print("Humidity: ");
  Serial.print(mySensor.readFloatHumidity(), 0);
  Serial.print(" Pressure: ");
  Serial.print(mySensor.readFloatPressure(), 0);
  Serial.print(" Alt: ");
  //Serial.print(mySensor.readFloatAltitudeMeters(), 1);
  Serial.print(mySensor.readFloatAltitudeFeet(), 1);
  Serial.print(" Temp: ");
  //Serial.print(mySensor.readTempC(), 2);
```

```
  Serial.print(mySensor.readTempF(), 2);

  Serial.println();

  delay(50);

}
```

I2C_and_SPI_Multisensor.ino

This example configures one BME280 on the SPI bus and another on the I2C bus. Then it gets the data and outputs from both sensors every second. If you only have 1 sensor connected the other channel reports garbage, so this can be a good troubleshooting and starting place.