SmartElex Environmental Sensor - BME688

Important Note: In order to avoid contamination of its gas scanning capabilities, **DO NOT** touch the metallic casing of the BME688 sensor.

The BME688 is a breakout for the 4-in-1 BME680 gas sensor from Bosch. The BME680 combines a gas sensor with temperature, humidity and barometric pressure sensing for a complete environmental sensor in a single package. The gas sensor on the BME680 can detect a wide variety of volatile organic compounds (or VOC for short) to monitor indoor air quality. Combine that with precise temperature, humidity and barometric pressure and the BME680 can work as a completely standalone environmental sensor all in a 1"x1" breakout! The BME680 communicates over either I^2C or SPI.

Hardware Overview

The heart of these breakout boards, Bosch's BME680 Gas Sensor, integrates four sensors (gas, pressure, temperature and humidity) into a tiny package. The BME68x measures just 3mm x 3mm x 0.93 mm and was specifically designed

for applications that depend on a small footprint and low power consumption. This makes the BME68x a great choice for remote or mobile environmental sensing applications. We will highlight some of the unique aspects of the BME68x in this section but for a full overview of the sensor package, check out the BME688 datasheet

BME688 Note: The BME688 is a drop in replacement for the BME680; with the added gas scanning functionality and support for AI algorithms. The parameters in highlighted in yellow, only apply to the BME688 sensor.

How does the gas scanner work?

The gas sensor takes measurements with different sensitivities during one gas scan. In doing so, it can generate a profile *(or fingerprint)* for different gas mixtures. This can be modified and optimized with BME AI-Studio.

Characteristic	**Description**
Operating Voltage	• V_{DD}: 1.71V to 3.6V • V_{DDIO}: 1.2 to 3.6V
Operational Modes	**Sleep** (**Default**) and Forced (*low power; single measurement*) Parallel (*Gas sensor heater operates in parallel to TPH measurement*)
Interface	I²C and SPI
I²C Address	BME688: **0x76** (**Default**) or 0x77
Average current consumption	2.1 µA at 1 Hz humidity and temperature 3.1 µA at 1 Hz pressure and temperature 3.7 µA at 1 Hz humidity, pressure and temperature 90 µA at ULP mode for p/h/T & air quality 0.9 mA at LP mode for p/h/T & air quality 3.9 mA in standard gas scan mode
Humidity Parameters	Range: 10 to 90 %RH Absolute Accuracy: ±3 %RH (from 20 - 80 %RH) Resolution: 0.008 %RH
Pressure Parameters	Range: 300 to 1100 hPa (30,000 - 110,000 Pa or approx. 4.35 - 15.95 PSI) Absolute Accuracy: ±0.6 hPa Resolution: 0.18 Pa

Temperature Parameters	Range: 0°C to 65°C (32°F to 149°F) Absolute Accuracy: ±(0.5 - 1.0)°C Resolution: 0.01°C
Gas Sensor Parameters	F1 score for H_2S scanning: 0.92 Standard scan speed: 10.8 s / scan Sensor-to-sensor deviation: +/- 15% +/- 15 Output data processing: • Index for Air Quality (IAQ) • bVOC-& CO_2-equivalents (ppm) • Gas scan result (%) • More listed in the **BSEC outputs** table: ○ *Table 20* in the BME688 datasheet

Power

The BME68x accepts a supply voltage between **1.71 to 3.6V**. Power can be supplied to the board either through one of the connectors or the dedicated **3.3V** and **GND** pins broken out on either side of the board.

I²C Interface

The Environmental Sensor - BME68x communicates over I²C by default. We have routed the BME68x's I²C pins to two connectors as well as broken them out to 0.1"-spaced the header pins.

Note: The default I²C address between the BME688 board is: **0x76**

Serial Peripheral Interface (SPI)

If you would prefer to communicate with your BME68x via SPI, we have broken those pins out as well to standard 0.1"-spade header pins. Communicating over SPI requires more connections than I²C but is more versatile and can be faster. It is particularly helpful if you need to use more than two BME68x's in your circuit or if you have other devices using the same I²C addresses.

BME688 SPI Jumpers: In order to communicate with the BME688 board over SPI, users will need to cut the `CSB` and `ADR` *(leave floating)* jumpers. **See the **CSB Jumper** section, below, for more information.*

Solder Jumpers

The Environmental Sensor - **BME688** has four solder jumpers which can be modified to alter the functionality of the sensor.

I²C Pull-Up Jumper

On the BME680 board, the SDA/SDI and SCL/SCK pins are pulled to VDDIO (**3.3V**) through a pair of **4.7kΩ** (**2.2kΩ** on the BME688) resistors. The jumper is normally **closed** so to disable the pull-up resistors, simply sever the traces between the three pads using a knife.

Power LED Jumper

This jumper connects the power LED to **3.3V** via a **1K Ohm** resistor. This jumper is normally **closed** so to disable the power LED, sever the trace between the two pads. This is particularly helpful for reducing the total current draw of your breakout for low-power applications.

I²C Address Jumper

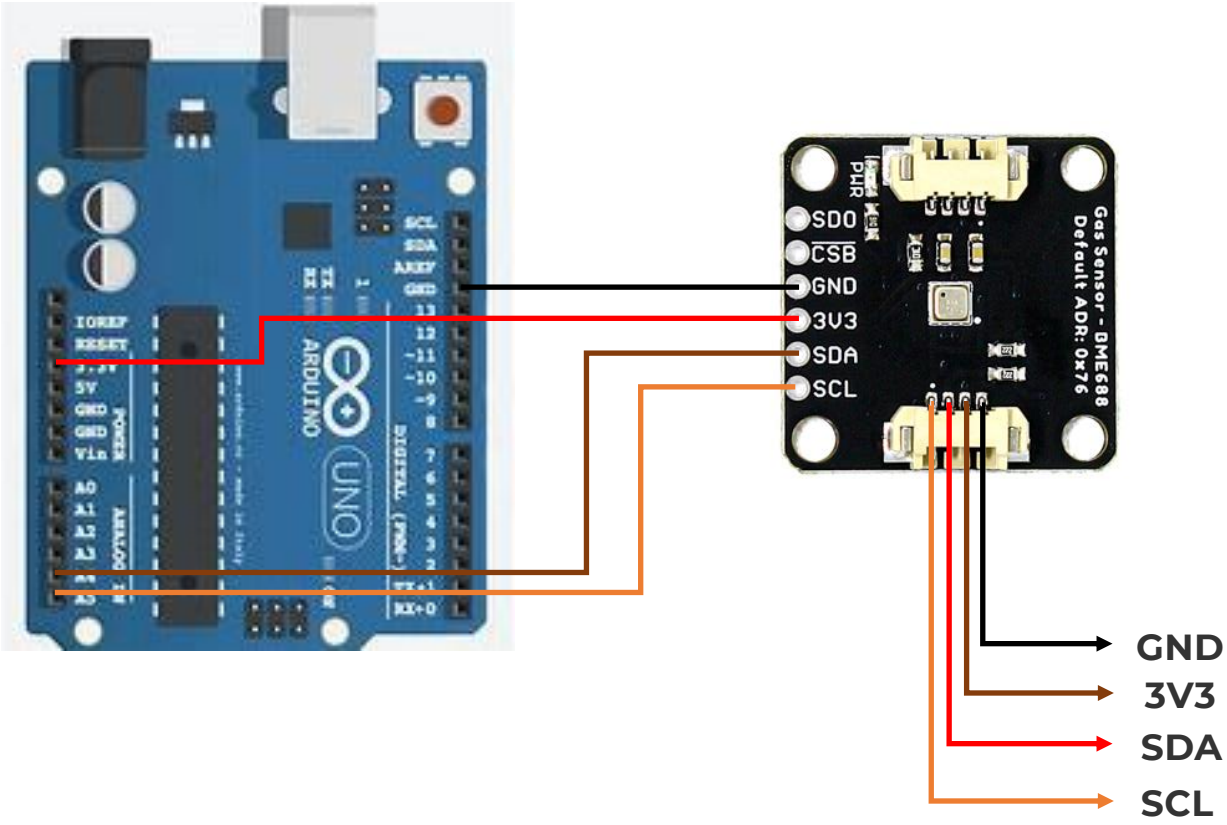
This jumper sets the 7-Bit unshifted I^2C address of the BME680 and is **open** by default. The default address is **0x77** and can be adjusted to **0x76** by closing this jumper.

This jumper sets the 7-Bit unshifted I^2C address of the BME688 and sets the default address to **0x76** and can be adjusted to **0x77** by cutting and soldering the jumper over to the `0x77` pad.

CSB Jumper

This jumper only applies to the BME688 board. The `CSB` pin is pulled up to V_{DDIO} in order to configure the board for I^2C communication by default. In order to communicate with the sensor over SPI, the `CSB` jumper must be cut along with the `ADR` jumper *(leave floating)*. Once the `CSB` pin has been pulled low during SPI communication, the sensor will communicate over SPI until there is a power reset.

Wiring



Arduino	BME688
SCL(A5)	SCL
SDA(A4)	SDA
3.3v	3V3
GND	GND

If you would prefer to communicate with the BME680 via SPI, you will need to connect to the SPI pins broken out on this board and route them to the respective pins for SPI communication on your development board (CIPO, COPI, SCK and CS). Also note that this breakout defaults to I^2C mode so your code will need to toggle the CS pin **LOW** once on power up to enable SPI mode. The BME680 will remain in SPI mode until the next power cycle. The SPI examples further on in this guide do that automatically so it's only necessary to note for writing your own code.

Note: On the BME688 board, users will need to cut the `ADR` and `CSB` jumpers to enable SPI communication. *(*See the **Hardware Overview** section for more information.)*

Soldering to the pins is the best option for a secure connection but you can also create temporary connections to those pins for prototyping using something like

these IC Hooks. If you are not familiar with through-hole soldering, take a look at this tutorial:

BME680 Arduino Library

For the scope of tutorial, we are going to use the BME680 Arduino Library created by Zanshin_BME680. You can download it with the Arduino Library Manager by searching **'BME688'** and selecting the one authored by SV-Zanshin.

Once you have the library installed you can move on to uploading the examples and gathering environmental data.

Example Code

```
#include "Zanshin_BME680.h" // Include the BME680 Sensor library

/*********************************************************************************************************

** Declare all program constants                                                                        **

*********************************************************************************************************/

const uint32_t SERIAL_SPEED = 115200; ///< Set the baud rate for Serial I/O


/*********************************************************************************************************

** Declare global variables and instantiate classes                                                     **

*********************************************************************************************************/

BME680_Class BME680; ///< Create an instance of the BME680 class


float altitude(const int32_t press, const float seaLevel = 1013.25); ///< Forward function declaration with default value for sea level

float altitude(const int32_t press, const float seaLevel)

{

 /*!

  * @brief     This converts a pressure measurement into a height in meters

  * @details   The corrected sea-level pressure can be passed into the function if it is know, otherwise the standard

  *            atmospheric pressure of 1013.25hPa is used (see https://en.wikipedia.org/wiki/Atmospheric_pressure)
```

```
  * @param[in] press    Pressure reading from BME680
  * @param[in] seaLevel Sea-Level pressure in millibars
  * @return    floating point altitude in meters.
  */
  static float Altitude;
  Altitude = 44330.0*(1.0-pow(((float)press/100.0)/seaLevel,0.1903)); // Convert into altitude in meters
  return(Altitude);
} // of method altitude()

void setup()
{
  /*!
  @brief    Arduino method called once at startup to initialize the system
  @details  This is an Arduino IDE method which is called first upon boot or restart. It is only called one time
            and then control goes to the main "loop()" method, from which control never returns
  @return   void
  */
  Serial.begin(SERIAL_SPEED); // Start serial port at Baud rate
  #ifdef  __AVR_ATmega32U4__  // If this is a 32U4 processor, then wait 3 seconds to initialize USB port
    delay(3000);
  #endif
  Serial.print(F("Starting I2CDemo example program for BME680\n"));
  Serial.print(F("- Initializing BME680 sensor\n"));
  while (!BME680.begin(I2C_STANDARD_MODE)) // Start BME680 using I2C protocol
  {
    Serial.print(F("-  Unable to find BME680. Trying again in 5 seconds.\n"));
    delay(5000);
  } // of loop until device is located
  Serial.print(F("- Setting 16x oversampling for all sensors\n"));
  BME680.setOversampling(TemperatureSensor,Oversample16); // Use enumerated type values
  BME680.setOversampling(HumiditySensor,   Oversample16); // Use enumerated type values
```

```
  BME680.setOversampling(PressureSensor,   Oversample16); // Use enumerated type values
  Serial.print(F("- Setting IIR filter to a value of 4 samples\n"));
  BME680.setIIRFilter(IIR4); // Use enumerated type values
  Serial.print(F("- Setting gas measurement to 320\xC2\xB0\x43 for 150ms\n")); // "°C" symbols
  BME680.setGas(320,150); // 320°c for 150 milliseconds
} // of method setup()
void loop()
{
  /*!
  @brief    Arduino method for the main program loop
  @details  This is the main program for the Arduino IDE, it is an infinite loop and keeps on repeating.
        The "sprintf()" function is to pretty-print the values, since floating point is not supported on the
        Arduino, split the values into those before and those after the decimal point.
  @return   void
  */
  static int32_t  temp, humidity, pressure, gas;                    // Variable to store readings
  static char     buf[16];                                          // Text buffer for sprintf
  static float    alt;                                              // temp variable for altitude
  static uint16_t loopCounter = 0;                                  // Display iterations
  if (loopCounter % 25 == 0)                                        // Display header every 25 loops
  {                                                                 //
    Serial.print(F("\nLoop Temp\xC2\xB0\x43 Humid% Press hPa   Alt m Air m"));     // Show header plus unicode
"°C"
    Serial.print(F("\xE2\x84\xA6\n==== ====== ====== ========= ======= ======\n")); // and "?" symbols
  } // if-then time to show headers                                 //
  BME680.getSensorData(temp,humidity,pressure,gas);                 // Get the most recent readings
  sprintf(buf, "%4d %3d.%02d", ++loopCounter%9999,                  // Clamp iterations to 9999,
        (int8_t)(temp/100),(uint8_t)(temp%100));                    // Temperature in decidegrees
  Serial.print(buf);                                                //
  sprintf(buf, "%3d.%03d", (int8_t)(humidity/1000),(uint16_t)(humidity%1000));     // Humidity in milli-percent
  Serial.print(buf);                                                //
```

```
  sprintf(buf, "%7d.%02d", (int16_t)(pressure/100),(uint8_t)(pressure%100));       // Pressure in Pascals
  Serial.print(buf);                                                              //
  alt = altitude(pressure);                                                       // temp variable for altitude
  sprintf(buf, "%5d.%02d", (int16_t)(alt),((uint8_t)(alt*100)%100));              // Altitude in meters
  Serial.print(buf);                                                              //
  sprintf(buf, "%4d.%02d\n", (int16_t)(gas/100),(uint8_t)(gas%100));              // Resistance in milliohms
  Serial.print(buf);                                                              //
  delay(10000);                                                                   // Wait 10s before repeating
} // of method loop()
```