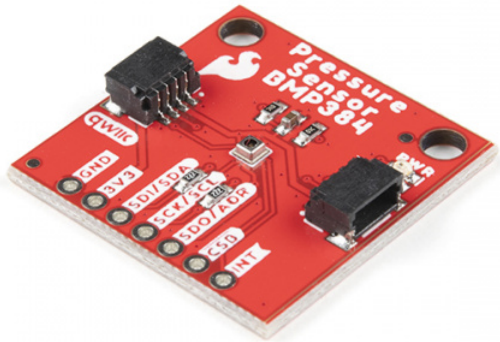
Qwiic Pressure Sensor (BMP384) Hookup Guide

Introduction

The SparkFun Pressure Sensor - BMP384 Qwiic features the BMP384 digital pressure sensor from Bosch©. The BMP384 excels at high-resolution measurements (up to 21-bit) and uses a gel-filled cavity to provide extra resistance to liquids (water and other chemicals) making it a great option for monitoring pressure in a wide variety of environments though the sensor is *not* water-proof.



SparkFun Pressure Sensor - BMP384 (Qwiic)
SEN-19662

This guide will take you through the hardware present on this Qwiic breakout, how to connect it to a Qwiic circuit and how to use it with the SparkFun BMP384 Arduino Library.

Required Materials

To follow along with this guide you will need a microcontroller to communicate with the BMP384. Below are a few options that come Qwiic-enabled out of the box:

SparkFun Thing Plus - ESP32 WROOM (Micro-B)
WRL-15663

SparkFun RedBoard Plus
DEV-18158

SparkFun RedBoard Artemis
DEV-15444

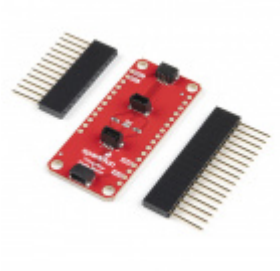
SparkFun Thing Plus - Artemis
WRL-15574

If your chosen microcontroller is not already Qwiic-enabled, you can add that functionality with one or more of the following items:

SparkFun Qwiic Cable Kit
KIT-15081

SparkFun Qwiic Adapter
DEV-14495

SparkFun Qwiic Shield for Arduino
○ DEV-14352



SparkFun Qwiic Shield for Thing Plus
◉ DEV-16790

You will also need at least one Qwiic cable to connect your sensor to your microcontroller.

Qwiic Cable - 100mm
◉ PRT-14427

Qwiic Cable - 50mm
◉ PRT-14426

Qwiic Cable - 500mm
◉ PRT-14429

Qwiic Cable - 200mm
◉ PRT-14428

Recommended Reading

If you aren't familiar with the Qwiic system, we recommend reading here for an overview.

We would also recommend taking a look at the following tutorials if you aren't familiar with them. If you are using one of the Qwiic Shields listed above, you may want to read through their respective Hookup Guides as well before you get started with the SparkFun Pressure Sensor - BMP384 (Qwiic).

I2C

An introduction to I2C, one of the main embedded communications protocols in use today.

Serial Terminal Basics

This tutorial will show you how to communicate with your serial devices using a variety of terminal emulator applications.

Qwiic Shield for Arduino & Photon Hookup Guide

Get started with our Qwiic ecosystem with the Qwiic shield for Arduino or Photon.

SparkFun Qwiic Shield for Arduino Nano Hookup Guide

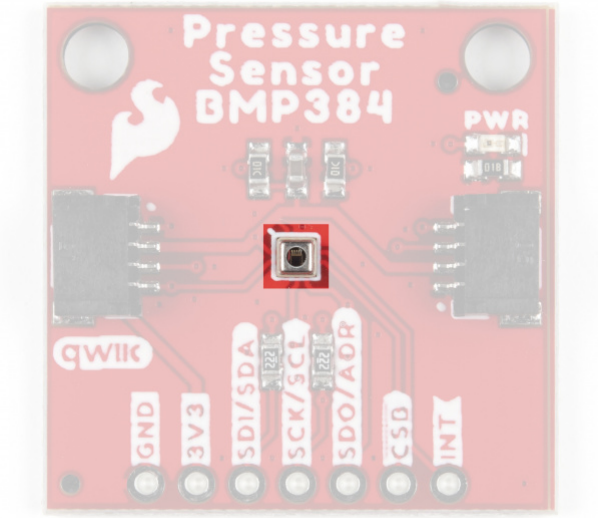
Hookup Guide for the SparkFun Qwiic Shield for Arduino Nano.

Hardware Overview

Let's take a closer look at the BMP384 sensor and other hardware present on these Qwiic breakouts.

BMP384 Pressure Sensor

The BMP384 is a high-resolution digital pressure sensor from Bosch with a wide measurement range (300hPa to 1250hPa) and excellent accuracy.



The sensor measures pressure and temperature with an average accuracy of 0.09hPa (300hPa to 1250hPa) and 0.35°C (at 25°C), respectfully. The sensor supports up to 21-bit resoluation as well as oversampling, low-pass filtering and a max sampling rate of 200Hz making it suitable for a wide range of applications. It uses a gel-filled cavity to improve the sensor's resistance to moisture (though not waterproof) so it works well in applications where the sensor may be exposed to liquids (outdoor sensor, drone, weather balloon etc.). For a complete overview of the sensor, refer to the datasheet.

The BMP384 accepts a supply voltage between **1.65V** to **3.6V**. The breakout runs the sensor at **3.3V** supply and logic when connected to a Qwiic system. The BMP384 supports data transfer speeds up to 3.4MHz over I^2C and speeds up to 10MHz over SPI. The sensor also includes a configurable Interrupt pin broken out to a pin on the PTH header.

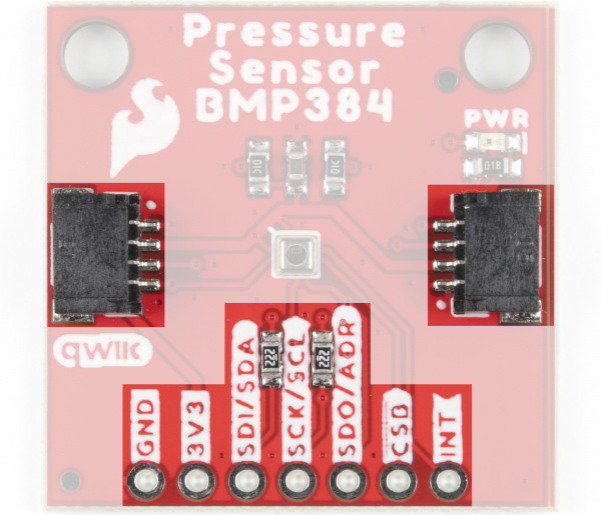
The BMP384 has three operating modes: Sleep Mode (Default after reset), Normal Mode and Forced Mode. While in Sleep Mode the sensor is idle and consumes ~2µA. While in Normal Mode, the sensor automatically cycles between measurement and standby periods and consumes ~700µA at peak current draw during measurements. Forced Mode allows direct control of measurements to wake the sensor from Sleep Mode, take a single-shot measurement and return the device to Sleep Mode.

Parameter	**Min.**	**Typ.**	**Max.**	**Units**	**Notes**
Operating Temperature	-40	25	85	°C	
Operating Pressure	300	-	1250	hPa	
Relative Accuracy	-	±9	-	Pa	At 900-1110hPA & 25-40°C
Absolute Accuracy	-	±50	-	Pa	At 300-1100hPa & 0-65°C
Temp. Coeff. Offset	-	±1.0	-	Pa/K	At 900hPa & 25-40°C
RMS Noise in Pressure [1]	-	1.2	-	Pa	Full bandwidth, highest resolution.
	-	0.03	-	Pa	Lowest bandwidth, highest resolution.
Sampling Rate [2]	-	-	200	Hz	Depends on oversampling settings *osr_t* and *osr_p*.>

1. Refer to section 3.4.4 of the BMP384 datasheet for more information.
2. Refer to section 3.9 of the BMP384 datasheet for more information.

Communication Interfaces - I^{2}C & SPI

The Qwiic Pressure Sensor (BMP384) communicates over I^{2}C by default but also supports using the BMP384 over SPI.



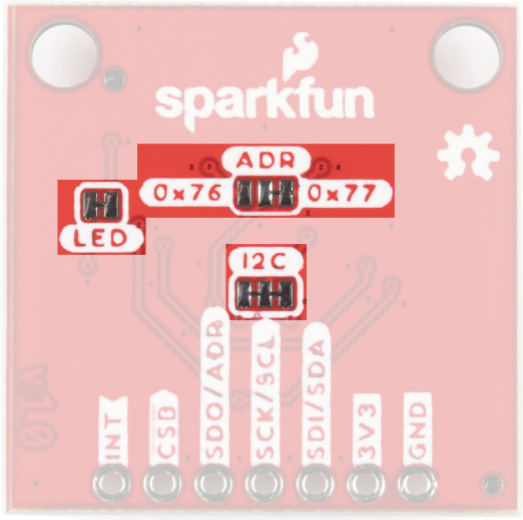
The breakout routes the I^{2}C interface to a pair of Qwiic connectors as well as a 0.1"-spaced PTH header for users who prefer a traditional, soldered connection. This PTH header shares the SPI connections and also includes the Interrupt pin.

The board sets the BMP384's I^{2}C address to **0x77** by default. Adjust the ADR jumper to change to the alternate address (**0x76**) or leave it completely open to use the SPI interface. More information on this jumper in the Solder Jumpers section below.

Solder Jumpers

If you have never worked with solder jumpers and PCB traces before or would like a quick refresher, check out our How to Work with Solder Jumpers and PCB Traces tutorial for detailed instructions and tips.

The breakout has three solder jumpers on the board labeled: **I2C**, **ADR** and **LED**.



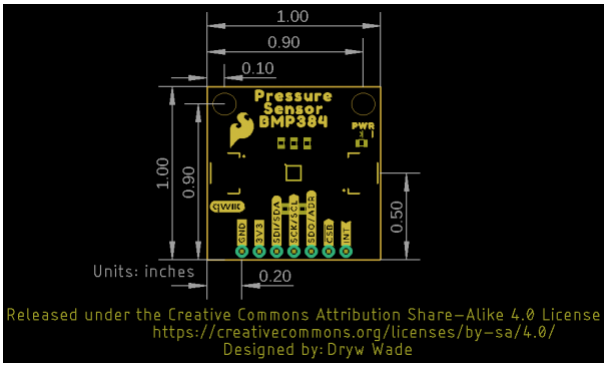
The I^{2}C jumper connects a pair of **2.2kΩ** resistors to the SDA/SCL lines. Leave these enabled unless you have a large amount of I^{2}C devices on the same bus.

The ADR jumper sets the I^{2}C address of the BMP384 to **0x77** by default (**0x76** alternate). It also controls whether it operates via I^{2}C or SPI. Open the jumper completely to set the BMP384 to communicate via SPI.

The LED jumper completes the Power LED circuit. Open the jumper to disable the Power LED if desired.

Board Dimensions

The breakout matches the 1.0" x 1.0" (25.4mm x 25.4mm) form factor for Qwiic breakouts with two mounting holes that fit a size 4-40 screw.

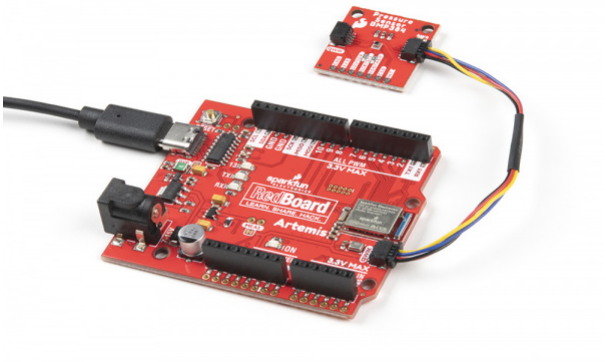


Hardware Assembly

Now that we're familiar with the Qwiic Pressure Sensor (BMP384), we can start assembling our circuit.

Qwiic/I^{2}C Assembly

The fastest and easiest way to get started using the breakout is using either of the Qwiic connectors, a Qwiic cable and a Qwiic-enabled development board like the SparkFun RedBoard Artemis.



If you would prefer a more secure and permanent connection, you can solder headers or wire to the PTH header on the board.

SPI Assembly

Setting the breakout up to communicate with the sensor over SPI requires completely opening the ADR jumper and we recommend soldering to the PTH header to make the connections. If you are not familiar with through-hole soldering, take a read through this tutorial:

How to Solder: Through-Hole Soldering

SEPTEMBER 19, 2013

This tutorial covers everything you need to know about through-hole soldering.

Along with tools for soldering, you'll need either some hookup wire or headers and jumper wires. Sever the trace between the "Center" and "Right" pads of the ADR jumper to switch to SPI mode. After opening this jumper, connect the BMP384 to your controller's SPI bus pins.

Remember, the BMP384 operates at **3.3V logic** so make sure to connect to a board running at the same logic level like the RedBoard Artemis or use a level shifter to adjust it to a safe voltage.

SparkFun BMP384 Arduino Library

> **Note:** This library assumes you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please check out our installation guide.

The SparkFun BMP384 Arduino Library is based off the API for the sensor from Bosch. Install the library through the Arduino Library Manager tool by searching for **"SparkFun BMP384"**. Users who prefer to manually install the library can download a copy of it from the GitHub repository by clicking the button below:

SPARKFUN BMP384 ARDUINO LIBRARY (ZIP)

> **Heads Up!** We recommend choosing a development board with plenty of available RAM like the RedBoard Artemis shown in the Hardware Assembly section if you want to use the FIFO buffer as it is read all at once which causes some microcontrollers like the ATMega328 on the RedBoard/Uno to run out of RAM after just a few samples. All other use cases of the Arduino Library will work with most microcontrollers.

Library Functions

The list below outlines and describes the functions available in the SparkFun BMP384 Library:

Sensor Initialization & Mode Selection

- `int8_t beginI2C(uint8_t address = BMP384_I2C_ADDRESS_DEFAULT, TwoWire& wirePort = Wire);` - Initialzes the BMP384 in I^2C at the specified address and on the specified Wire port.
- `int8_t beginSPI(uint8_t csPin, uint32_t clockFrequency = 100000);` - Initializes the BMP384 in SPI mode at the specified frequency (default is 100000) and sets the Chip Select pin.
- `int8_t init();` - Initialize the BMP384. The begin functions automatically perform this.

- `int8_t setMode(uint8_t mode);` - Manually set the operating mode of the BMP384. Set to Normal by default in the begin functions.
- `int8_t enablePressAndTemp(uint8_t pressEnable, uint8_t tempEnable);` - Enable pressure and temperature measurements. Enabled by default in the begin functions.

Sensor Data

- `int8_t getSensorData(bmp3_data* data);` - Returns pressure and temperature data from the sensor.
- `int8_t getSensorStatus(bmp3_sens_status* sensorStatus);` - Returns the status of command ready, data ready for pressure & temperature and power on reset parameters.
- `int8_t setODRFrequency(uint8_t odr);` - Set the Output Data Rate frequency.
- `int8_t getODRFrequency(uint8_t* odr);` - Retrieve the value stored for the Output Data Rate frequency.
- `int8_t setOSRMultipliers(bmp3_odr_filter_settings osrMultipliers);` - Set the Oversampling Rate multipliers.
- `int8_t getOSRMultipliers(bmp3_odr_filter_settings* osrMultipliers);` - Retreive the value set for the Oversampling multiplier.
- `int8_t setFilterCoefficient(uint8_t coefficient);` - Sets the low pass filter coeffecient.
- `int8_t setInterruptSettings(bmp3_int_ctrl_settings interruptSettings);` - Set the Interrupt Settings (output mode, level, latch and data ready). Refer to Example 3 - Interrupts in the Arduino Library for a detailed demonstration of setting and using the Interrupt pin.
- `int8_t getInterruptStatus(bmp3_int_status* interruptStatus);` - Returns the settings for the Interrupt.

FIFO Buffer Control

Refer to Example 6 - FIFO Buffer in the Arduino library for a detailed example of setting and using the FIFO buffer.

- `int8_t setFIFOSettings(bmp3_fifo_settings fifoSettings);` - Set the FIFO buffer settings.
- `int8_t setFIFOWatermark(uint8_t numData);` - Sets the number of samples for the FIFO watermark.
- `int8_t getFIFOLength(uint8_t* numData);` - Returns the number of data samples in the FIFO buffer.
- `int8_t getFIFOData(bmp3_data* data, uint8_t numData);` - Pull the FIFO data stored on the BMP384.
- `int8_t flushFIFO();` - Clears the FIFO buffer.

Arduino Examples

Let's take a closer look at a few of the examples included in the SparkFun BMP384 Arduino Library.

Example 1 - Basic Readings I^{2}C

The first example initializes the BMP384 to communicate over I^{2}C with default settings. Open the example by navigating to **File Examples > SparkFun BMP384 Arduino Library > Example_1_Basic_ReadingsI2C**. Select your Board and Port and click upload. Open the serial monitor after the upload completes with the baud set to **115200** to watch pressure data (in Pascals) print out.

If you have switched to the alternate address, comment/uncomment the line with the correct value:

```
uint8_t i2cAddress = BMP384_I2C_ADDRESS_DEFAULT; // 0x77
//uint8_t i2cAddress = BMP384_I2C_ADDRESS_SECONDARY; // 0x76
```

The code attempts to initialize the sensor with default settings in I^{2}C at the specified address and prints out an error message if it cannot initialize properly:

```
while(pressureSensor.beginI2C(i2cAddress) != BMP3_OK)
{
    // Not connected, inform user
    Serial.println("Error: BMP384 not connected, check wiring and I2C address!");

    // Wait a bit to see if connection is established
    delay(1000);
}
```

After initializing, the main loop polls the BMP384 for pressure and temperature data every second. If polling for data fails, the code will print out an error code for debugging. Try moving the sensor up and down and you should see noticeable differences in pressure readings with just a few inches of movement.

```
void loop()
{
    // Get measurements from the sensor
    bmp3_data data = {0};
    int8_t err = pressureSensor.getSensorData(&data);

    // Check whether data was acquired successfully
    if(err == BMP3_OK)
    {
        // Acquisistion succeeded, print temperature and pressure
        Serial.print("Temperature (C): ");
        Serial.print(data.temperature);
        Serial.print("\t\t");
        Serial.print("Pressure (Pa): ");
        Serial.println(data.pressure);
    }
    else
    {
        // Acquisition failed, most likely a communication error (code -2)
        Serial.print("Error getting data from sensor! Error code: ");
        Serial.println(err);
    }

    // Only print every second
    delay(1000);
}
```

Example 4 - Filtering

Example 4 demonstrates how to set up a low-pass filter for the BMP384 data to smooth out the output. After initializing the sensor, the code creates an case to print error codes returned by API calls and sets the filter coefficient:

```
// Variable to track errors returned by API calls
int8_t err = BMP3_OK;

// By default, the filter coefficient is set to 0 (no filtering). We can
// smooth out the measurements by increasing the coefficient
err = pressureSensor.setFilterCoefficient(BMP3_IIR_FILTER_COEFF_127);
if(err)
{
    // Setting coefficient failed, most likely an invalid coefficient (code -3)
    Serial.print("Error setting filter coefficient! Error code: ");
    Serial.println(err);
}
```

Example 5 - Oversampling

Example 5 shows how to set the oversampling rate on the BMP384 so it performs multiple samples between each measurement to boost resolution and reduce noise. The code sets the oversampling rate to 32x for pressure measurements and 2x for temperature:

```
bmp3_odr_filter_settings osrMultipliers =
{
    .press_os = BMP3_OVERSAMPLING_32X,
    .temp_os = BMP3_OVERSAMPLING_2X
};
err = pressureSensor.setOSRMultipliers(osrMultipliers);
if(err)
{
    // Setting OSR failed, most likely an invalid multiplier (code -3)
    Serial.print("Error setting OSR! Error code: ");
    Serial.println(err);
}
```

Adjusting the oversampling rate requires an adjustment to the output data rate as well. The `setOSRMultipliers()` function automatically adjusts it and the code polls the sensor to return the data rate in Hz:

```
uint8_t odr = 0;
err = pressureSensor.getODRFrequency(&odr);
if(err)
{
    // Interrupt settings failed, most likely a communication error (code -2)
    Serial.print("Error getting ODR! Error code: ");
    Serial.println(err);
}

// The true ODR frequency in Hz is [200 / (2^odr)]
Serial.print("ODR Frequency: ");
Serial.print(200 / pow(2, odr));
Serial.println("Hz");
```

Example 6 - FIFO Buffer

Reminder We recommend using a microcontroller with plenty of RAM like the RedBoard Artemis as the BMP384 reads the entire FIFO buffer all at once. This causes some microcontrollers like the ATMega328 on the RedBoard or Uno to run out of RAM after just a few samples.

Example 6 demonstrates how to enable, read and flush the FIFO buffer on the BMP384. The example uses the BMP384's interrupt pin to trigger an external interrupt for an attached microcontroller to monitor when the FIFO buffer reaches a specified threshold, in this case the code will trigger when the FIFO buffer has 5 samples stored in it.

The code sets up D2 as the interrupt pin so make sure to connect the Interrupt pin to D2 and ensure your microcontroller supports external interrupts on D2. If it does not, adjust the pin to one that can.

```
int interruptPin = 2;

// Flag to know when interrupts occur
volatile bool interruptOccurred = false;

// Create a buffer for FIFO data
// Note - on some systems (eg. Arduino Uno), warnings will be generated
// when numSamples is large (eg. >= 5)
const uint8_t numSamples = 5;
bmp3_data fifoData[numSamples];
```

The setup initializes the BMP384 on the I^2C bus and then sets the FIFO buffer settings:

```
bmp3_fifo_settings fifoSettings =
{
    .mode            = BMP3_ENABLE,  // Enable the FIFO buffer
    .stop_on_full_en = BMP3_DISABLE, // Stop writing to FIFO once full, or overwrite oldest data
    .time_en         = BMP3_DISABLE, // Enable sensor time, only 1 frame at end of buffer
    .press_en        = BMP3_ENABLE,  // Enable pressure sensor recording
    .temp_en         = BMP3_ENABLE,  // Enable temperature sensor recording
    .down_sampling   = BMP3_FIFO_NO_SUBSAMPLING, // Set downsampling factor
    .filter_en       = BMP3_DISABLE, // Enable data filtering
    .fwtm_en         = BMP3_ENABLE,  // Trigger interrupt on FIFO watermark
    .ffull_en        = BMP3_DISABLE  // Trigger interrupt on FIFO full
};
```

Note that the FIFO settings includes the interrupt conditions. In this case, the interrupt is configured to trigger on the FIFO watermark (5 samples) set further down in the code.

After setting everything up, the main loop monitors the FIFO buffer, prints the number of samples currently stored up to when the sample number hits the watermark threshold. Once the number of samples hits the watermark, the interrupt condition triggers and the code prints out the data for each sample stored in the FIFO buffer.

Troubleshooting

Temperature Offset

The BMP384 reports temperature measured *inside* the sensor package so it reads several degrees warmer than ambient. This temperature offset should be quite steady so users looking to get accurate ambient temperature from the sensor can subtract that offset.

Pressure Data as Altitude

If you want to use the pressure data from the BMP384 to determine the altitude of the sensor, refer to this section of our MPL3115A2 Breakout Hookup Guide for more information on how to manipulate and correctly interpret pressure data.

General Troubleshooting

Not working as expected and need help?

If you need technical assistance and more information on a product that is not working as you expected, we recommend heading on over to the SparkFun Technical Assistance page for some initial troubleshooting.

SPARKFUN TECHNICAL ASSISTANCE PAGE

If you don't find what you need there, the SparkFun Forums are a great place to find and ask for help. If this is your first visit, you'll need to create a Forum Account to search product forums and post questions.

CREATE NEW FORUM ACCOUNT | LOG INTO SPARKFUN FORUMS

Resources and Going Further

For more information about the SparkFun Pressure Sensor - BMP384 (Qwiic) check out the following resources:

- Schematic
- Eagle Files
- Board Dimensions
- Datasheet (BMP384)
- GitHub Hardware Repo
- BMP384 Arduino Library

Looking for inspiration for your next environmental sensing project? The tutorials below may help you get started:

Photon Remote Water Level Sensor

Learn how to build a remote water level sensor for a water storage tank and how to automate a pump based

HIH-4030 Humidity Sensor Hookup Guide

Measure relative humidity with SparkFun's HIH-4030 Humidity Sensor Breakout.

off the readings!

micro:climate Kit Experiment Guide

A weather station kit that is built on top of the inexpensive, easy-to-use micro:bit and Microsoft MakeCode.

SparkFun Humidity Sensor Breakout - SHTC3 (Qwiic) Hookup Guide

A Hookup Guide to get started using the SHTC3 breakout.