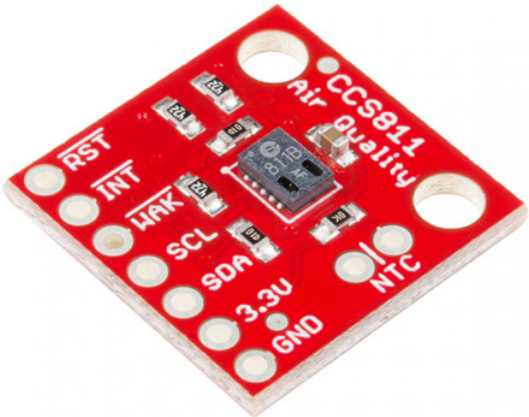
CCS811 Air Quality Breakout Hookup Guide

CONTRIBUTORS: *MTAYLOR*

Introduction

The CCS811 Air Quality Breakout is a digital gas sensor solution that senses a wide range of Total Volatile Organic Compounds (TVOCs), including equivalent carbon dioxide (eCO_2) and metal oxide (MOX) levels. It is intended for indoor air quality monitoring in personal devices such as watches and phones, but we've put it on a breakout board so you can use it as a regular I^2C device.



SparkFun Air Quality Breakout - CCS811
SEN-14193
$20.95
★★★★½ 5

Required Materials

To follow along with this project tutorial, you will need the following materials:

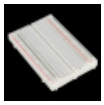
CCS811 Air Quality Breakout Hookup Guide SparkFun Wish List

SparkFun RedBoard - Programmed with Arduino
DEV-13975

At SparkFun we use many Arduinos and we're always looking for the simplest, most stable one. Each board is a bi…

Jumper Wires Premium 6" M/M Pack of 10
PRT-08431
This is a SparkFun exclusive! These are 155mm long jumpers with male connectors on both ends. Use these to ju…

Break Away Headers - Straight
PRT-00116
A row of headers - break to fit. 40 pins that can be cut to any size. Used with custom PCBs or general custom hea…



Breadboard - Self-Adhesive (White)
PRT-12002
This is your tried and true white solderless breadboard. It has 2 power buses, 10 columns, and 30 rows - a total of…

Resistor Kit - 1/4W (500 total)
COM-10969
Resistors are a good thing, in fact, they're actually crucial in a lot of circuit designs. The only problem seems to be…

SparkFun Air Quality Breakout - CCS811
SEN-14193
The CCS811 Air Quality Breakout is a digital gas sensor solution that senses a wide range of Total Volatile Organic…

Suggested Reading

If you aren't familiar with the following concepts, we recommend checking out these tutorials before continuing.

How to Solder: Through-Hole Soldering

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

Installing an Arduino Library

How do I install a custom Arduino library? It's easy! This tutorial will go over how to install an Arduino library using the Arduino Library Manager. For libraries not linked with the Arduino IDE, we will also go over manually installing an Arduino library.

How to Use a Breadboard

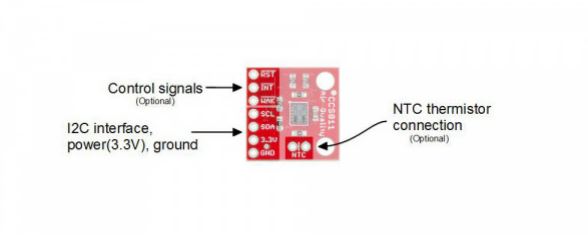
Welcome to the wonderful world of breadboards. Here we will learn what a breadboard is and how to use one to build your very first circuit.

I2C

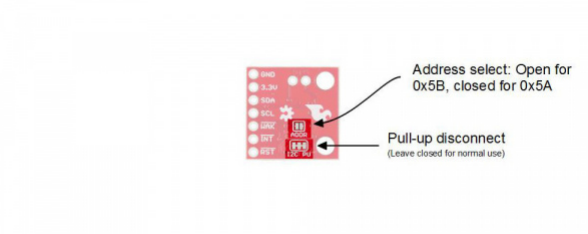
An introduction to I2C, one of the main embedded communications protocols in use today.

Hardware Overview

The CCS811 is supported by only a few passives, and so the breakout board is relatively simple. This section discusses the various pins on the board.



Connections available to the user are shown on the top side



Jumpers are available on the bottom

Pins

Pin	Description	Direction
$\overline{\text{RST}}$	Reset (active low)	In
$\overline{\text{INT}}$	Interrupt (active low)	Out
$\overline{\text{WAK}}$	Wake (active low)	In
SCL	Clock	In
SDA	Data	In
3.3V	Power	In
GND	Ground	In
NTC (2 pins)	Negative thermal coefficient resistor	N/A

Power and I^2C Bus

The minimum required connections are power, ground SDA and SCL. Supply a regulated 3.3V between the board's 3.3V pin and ground terminals. The sensor consumes an average of 12mA of current.

The I^2C bus has pull-up resistors enabled by default. If not desired, these can be removed by separating the "I^2C PU" triple jumper on the bottom side with a hobby knife.

An I^2C address can be either 0x5A or 0x5B. The "ADDR" jumper is connected with copper from the factory, corresponding to a default address of 0x5B. Separate this jumper to use the address 0x5A.

Settling time: This sensor takes about 20 minutes to get fully settled to a point where it generates good data. The I^{2}C bus is active, and data can be collected before the 20 minutes is up, but it may not be accurate.

Control lines

Additionally, the three control lines $\overline{\text{RST}}$, $\overline{\text{INT}}$ and $\overline{\text{WAK}}$ can be used to further the degree of control.

- $\overline{\text{RST}}$ — Pull this line low to reset the IC.
- $\overline{\text{INT}}$ — After configuring the sensor to emit interrupt requests, read this line to determine the state of the interrupt.
- $\overline{\text{WAK}}$ — Pull this line high to put the sensor to sleep. This can be used to save power but is not necessary if power is not an issue.

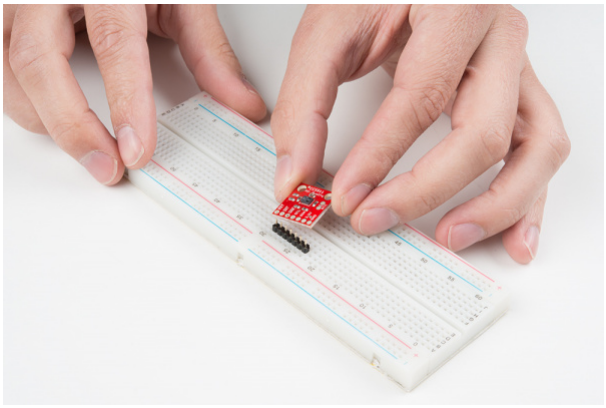
NTC Thermistor operation

A thermistor can be used to determine the temperature of the CCS811's surroundings, which can be used to help compensate the readings. You'll need your own 10K NTC thermistor, such as our 10K Thermistor, soldered between the "NTC" pins. A thermistor is a nonpolarized device, so it can go in either way.

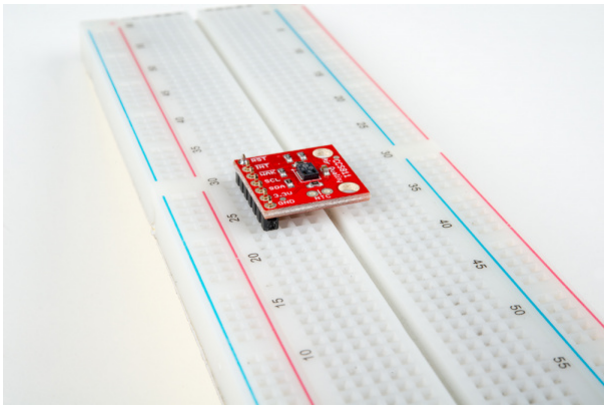
Hardware Assembly

Attach Headers

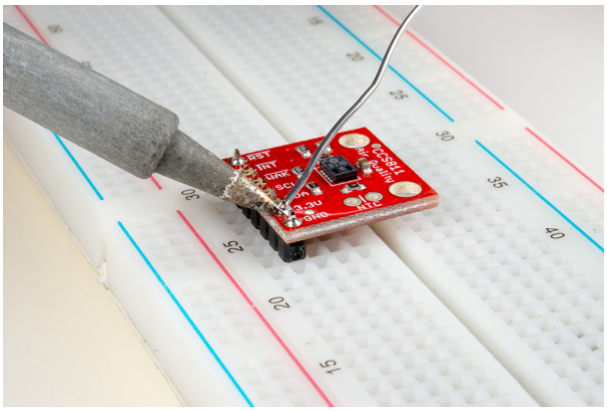
To prepare the sensor for the examples, attach seven pins from a Break Away Header to the through holes. Even though we only need the four I^{2}C pins, we'll populate all of them for this guide in case we want to try them out.



Place the strip of seven pins in a breadboard.



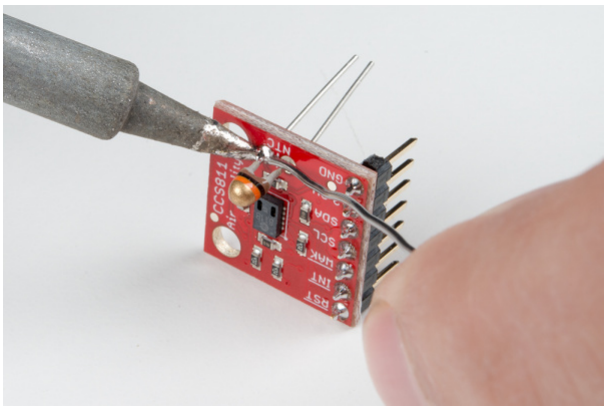
Solder a single pin and then check that the board is square to the pins.



Solder the remaining pins.

Attach NTC thermistor (Optional)

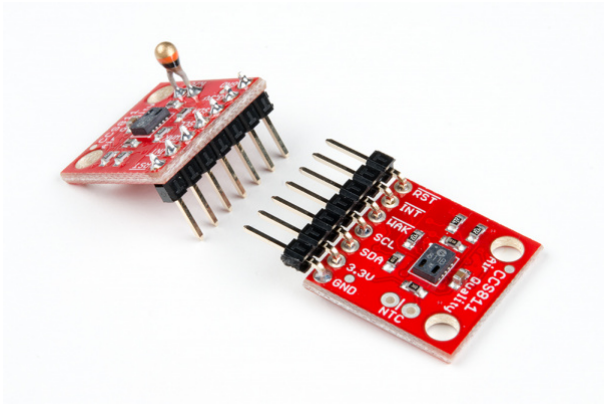
If you would like to use a thermistor to compensate for temperature, solder in a 10K Thermistor (Vishay part number NTCLE100E3103JB0).



Attaching an NTC thermistor

Example Assemblies

You're ready to start communicating with the CCS811! Here's an example with the NTC Thermistor populated, and one using right-angle headers instead.



Arduino Library and Usage

Getting the CCS811 Arduino Library

To get the Arduino library, download from GitHub or use the Arduino Library Manager.

Download the GitHub repository

Visit the GitHub repository to download the most recent version of the library, or click the button below:

DOWNLOAD THE SPARKFUN CCS811 ARDUINO LIBRARY

Use the Library Manager or install in the Arduino IDE

For help installing the library, check out our Installing an Arduino Library tutorial.

If you don't end up using the manager, you'll need to move the SparkFun_CCS811_Arduino_Library folder into a *libraries* folder within your Arduino sketchbook. If you downloaded the zip, you can remove "master" from the name, but it's not required.

Using the Library

The library is fairly normal to use compared with our other sensors. You'll have to include the library, create a sensor object in the global space, and then use functions of that object to begin and control the sensor. With this one, pass the I^2C address to the object during construction.

To get the library included, and to take care of all the gritty compiler stuff, place the following at the beginning of the sketch before `void setup()`

```
#include <SparkFunCCS811.h>

#define CCS811_ADDR 0x5B //Default I2C Address
//#define CCS811_ADDR 0x5A //Alternate I2C Address

CCS811 myCCS811(CCS811_ADDR);
```

Now, functions of the object named `myCCS811` can be called to set up and get data, while all the wire stuff is kept under the hood.

To make the sensor get ready during program boot, `myCCS811.begin()` must be called. Here's an example of the minimal usage of begin.

Error Status: The .begin() function has a special feature: it returns the status of the function call! If there was a problem during begin, it will return a non-zero code indicating what happened. It's optional, and is described in the "Custom Types and Literals" section below.

```
void setup()
{
    myCCS811.begin();
}
```

Then, in the main loop of the program, call sensor functions such as `mySensor.readAlgorithmResults()` to operate the sensor. The following snippet shows a simple check for data, to call on the sensor to calculate and get values, and to access those values. It doesn't do anything with the data, though! Check out the examples for fully functional code.

```
void loop()
{
  if (myCCS811.dataAvailable())
  {
    myCCS811.readAlgorithmResults();
    int tempCO2 = myCCS811.getCO2();
    int tempVOC = myCCS811.gettVOC();
  }
  else if (myCCS811.checkForStatusError())
  {
    while(1);
  }

  delay(1000); //Wait for next reading
}
```

Function Reference

The following functions exist for the CCS811 object.

Functions with scoped return type CCS811Core::status report an error state as defined in the literals section below. It is optional and can be used to determine success or failure of call.

- `CCS811Core::status begin( void )` — This starts `wire`, checks the ID register, checks for valid app data, starts the app, and establishes a drive mode.
- `CCS811Core::status readAlgorithmResults( void )` — Call to cause the sensor to read its hardware and calculate TVOC and eCO_2 levels.
- `bool checkForStatusError( void )` — Returns `true` if there is an error pending. This checks the status register.
- `bool dataAvailable( void )` — Returns `true` if a new sample is ready and hasn't been read.
- `bool appValid( void )` — Returns `true` if there is a valid application within the internal CCS811 memory.
- `uint8_t getErrorRegister( void )` — Returns the state of the ERROR_ID register.
- `uint16_t getBaseline( void )` — Returns the baseline value.
- `CCS811Core::status setBaseline( uint16_t )` — Apply a saved baseline to the CCS811.
- `CCS811Core::status enableInterrupts( void )` — Enables the interrupt pin for data ready.
- `CCS811Core::status disableInterrupts( void )` — Disables the interrupt pin.
- `CCS811Core::status setDriveMode( uint8_t mode )` — Sets the drive mode (`mode` can be 0 through 4).
    - 0: Measurement off
    - 1: Measurement every 1 second
    - 2: Measurement every 10 seconds
    - 3: Measurement every 60 seconds
    - 4: Measurement every 0.25 seconds — for use with external algorithms

- `CCS811Core::status setEnvironmentalData( float relativeHumidity, float temperature )` — Sets the environmental conditions for compensation.
    - relativeHumidity in units of %, 0.00 through 100.0
    - temperature in degrees C, -25.0 through 50.0

- `void setRefResistance( float )` — If you've changed the thermistor pull-up, call this to give the sensor the new resistor value. Otherwise, it will be 10000.
- `CCS811Core::status readNTC( void )` — Cause the CCS811 to get and calculate a temperature from the thermistor input.
- `uint16_t getTVOC( void )` — Collect the last calculated TVOC value, in parts per billion (ppb).
- `uint16_t getCO2( void )` — Collect the last calculated $eC0_2$ value, in parts per million (ppm).
- `float getResistance( void )` — Collect the last calculated resistance value of the NTC terminals.
- `float getTemperature( void )` — Collect the last calculated temperature.

Custom Types and Literals

The CCS811 library defines the following special data type to deal with error states of functions. In most places the library can be used without paying attention to the function return types, but if they are needed, here are the values the data type `status` can hold:

```
// Return values
typedef enum
{
    SENSOR_SUCCESS,
    SENSOR_ID_ERROR,
    SENSOR_I2C_ERROR,
    SENSOR_INTERNAL_ERROR
    //...
} status;
```

To avoid the possibility of multiple libraries using the same `status` name, the enum is actually inside the scope of the CCS811 object, buried in the CCS811Core, which is the base class. *Phew*, don't worry about that too much; just place `CCSCore::` before the status name when you want to use it, and use it like a regular enum (e.g., `CCS811Core::status myLocalReturnStatus;`). This just tells the compiler that the variable name is in a specific place. You'll also have to add the scope operator to the enum names.

Here's an example that shows how the status enum can be used:

```
CCS811Core::status returnCode = mySensor.beginCore();
Serial.print("beginCore exited with: ");
switch ( returnCode )
{
case CCS811Core::SENSOR_SUCCESS:
  Serial.print("SUCCESS");
  break;
case CCS811Core::SENSOR_ID_ERROR:
  Serial.print("ID_ERROR");
  break;
case CCS811Core::SENSOR_I2C_ERROR:
  Serial.print("I2C_ERROR");
  break;
case CCS811Core::SENSOR_INTERNAL_ERROR:
  Serial.print("INTERNAL_ERROR");
  break;
case CCS811Core::SENSOR_GENERIC_ERROR:
  Serial.print("GENERIC_ERROR");
  break;
default:
  Serial.print("Unspecified error.");
}
```

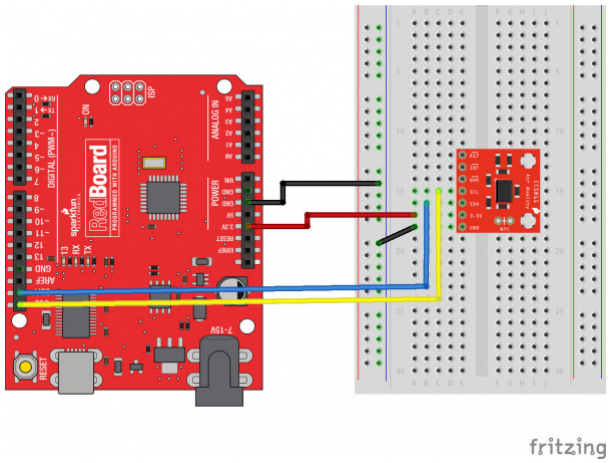
The library also defines names for CCS811 registers, if you're using direct read and write functions. These are globally scoped and can be used anywhere.

```
//Register addresses
#define CSS811_STATUS 0x00
#define CSS811_MEAS_MODE 0x01
#define CSS811_ALG_RESULT_DATA 0x02
#define CSS811_RAW_DATA 0x03
#define CSS811_ENV_DATA 0x05
#define CSS811_NTC 0x06
#define CSS811_THRESHOLDS 0x10
#define CSS811_BASELINE 0x11
#define CSS811_HW_ID 0x20
#define CSS811_HW_VERSION 0x21
#define CSS811_FW_BOOT_VERSION 0x23
#define CSS811_FW_APP_VERSION 0x24
#define CSS811_ERROR_ID 0xE0
#define CSS811_APP_START 0xF4
#define CSS811_SW_RESET 0xFF
```

Example: Basic Reading

After you've got pins attached to your breakout board, the first example to use should be *BasicReadings*. Select it from examples or copy from below.

Connect the sensor as follows as a starting place for the examples.



Wiring diagram showing basic connection to RedBoard. Click for a closer look.

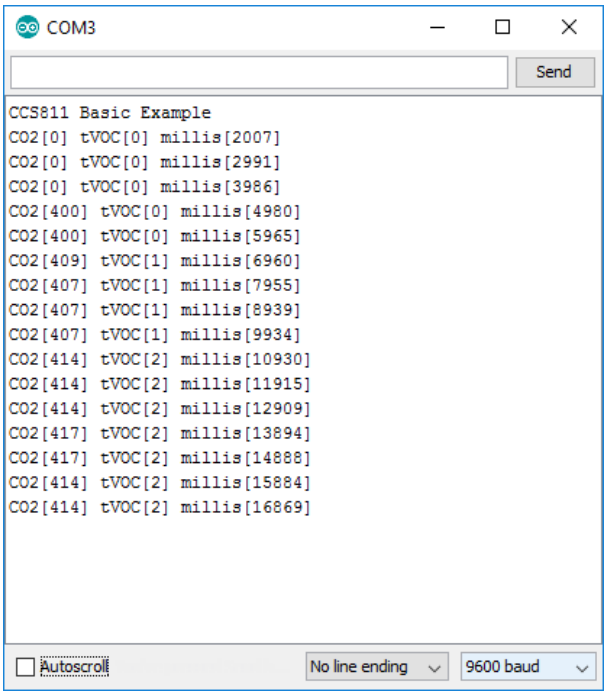
For this example, only 3.3V, GND, SDA and SCL are needed. The jumpers on the board are left in the default positions.

```
/******************************************************************************
  BasicReadings.ino

  Marshall Taylor @ SparkFun Electronics
  Nathan Seidle @ SparkFun Electronics

  April 4, 2017

  https://github.com/sparkfun/CCS811_Air_Quality_Breakout
  https://github.com/sparkfun/SparkFun_CCS811_Arduino_Library

  Read the TVOC and CO2 values from the SparkFun CSS811 breakout board

  This is the simplest example.  It throws away most error information and
  runs at the default 1 sample per second.

  A new sensor requires at 48-burn in. Once burned in a sensor requires
  20 minutes of run in before readings are considered good.

  Hardware Connections (Breakoutboard to Arduino):
  3.3V to 3.3V pin
  GND to GND pin
  SDA to A4
  SCL to A5

  Resources:
  Uses Wire.h for i2c operation

  Development environment specifics:
  Arduino IDE 1.8.1

  This code is released under the [MIT License](http://opensource.org/licenses/MIT).

  Please review the LICENSE.md file included with this example. If you have any question
s
  or concerns with licensing, please contact techsupport@sparkfun.com.

  Distributed as-is; no warranty is given.
******************************************************************************/
#include "SparkFunCCS811.h"

#define CCS811_ADDR 0x5B //Default I2C Address
//#define CCS811_ADDR 0x5A //Alternate I2C Address

CCS811 mySensor(CCS811_ADDR);

void setup()
{
  Serial.begin(9600);
  Serial.println("CCS811 Basic Example");

  //It is recommended to check return status on .begin(), but it is not
  //required.
```

```
  CCS811Core::status returnCode = mySensor.begin();
  if (returnCode != CCS811Core::SENSOR_SUCCESS)
  {
    Serial.println(".begin() returned with an error.");
    while (1); //Hang if there was a problem.
  }
}

void loop()
{
  //Check to see if data is ready with .dataAvailable()
  if (mySensor.dataAvailable())
  {
    //If so, have the sensor read and calculate the results.
    //Get them later
    mySensor.readAlgorithmResults();

    Serial.print("CO2[");
    //Returns calculated CO2 reading
    Serial.print(mySensor.getCO2());
    Serial.print("] tVOC[");
    //Returns calculated TVOC reading
    Serial.print(mySensor.getTVOC());
    Serial.print("] millis[");
    //Simply the time since program start
    Serial.print(millis());
    Serial.print("]");
    Serial.println();
  }

  delay(10); //Don't spam the I2C bus
}
```

At the beginning, an object is created in the global space `CCS811 mySensor(CCS811_ADDR);` and is constructed with the address as a parameter. During the setup phase, the library is told to configure with `CCS811Core::status returnCode = mySensor.begin();` Using the return parameter to check for errors is optional, and in this example if the return code is not valid, the program enters a captive while loop.

To get data from the sensor, `mySensor.dataAvailable()` is checked until a new reading is ready, `mySensor.readAlgorithmResults();` is called to have the sensor process the reading, then `mySensor.getCO2()` and `mySensor.getTVOC()` are used to retrieve the calculated values for gas levels.



Example terminal output

If everything is connected correctly, the serial window will report gas levels every second. Remember the sensor takes 20 minutes to properly warm up, so values reported will rise up in the early stages of operation!

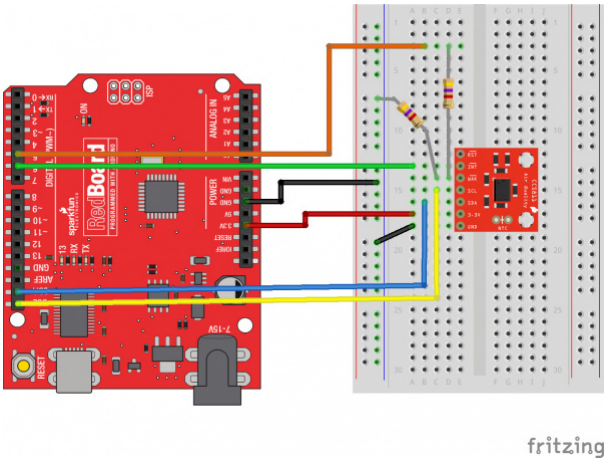
Summary:

To get data from the CCS811, these minimum requirements must be met:

- Create a `CCS811` object in the global space
- Run `.begin()` of your object (Return type monitoring optional)
- Check for the availability of new data with `.dataAvailable()`
- Use `.readAlgorithmResults()` to perform a measurement
    - `.getCO2()` to get the last equivalent CO_2 reading (no I^2C bus operation)
    - `.getTVOC()` to get the last TVOC reading (no I^2C bus operation)

Example: Additional Control Lines

The CCS811 has a couple extra control lines that are not part of the I^2C bus, which can be utilized to improve the system. There's a pin to flag that data is ready, and a pin to make the sensor go to sleep.



Wiring diagram including the wake and interrupt pins. Click for a closer look.

To connect the interrupt line, connect it directly to an input pin. This is a 3.3V output from the sensor, so it's OK to drive the input logic of a 5V device from it. The example has nInt connected to pin 6.

To connect the wake stat line, use a voltage divider to divide the 5V coming from the Arduino down to something below 3.3V for the sensor. The example has nWake connected to pin 5 through a voltage divider made from two 4.7K resistors for a 2.5V output.

The example for these additional lines is called *WakeAndInterrupt* and is listed here:

```
/******************************************************************************
  WakeAndInterrupt.ino

  Marshall Taylor @ SparkFun Electronics

  April 4, 2017

  https://github.com/sparkfun/CCS811_Air_Quality_Breakout
  https://github.com/sparkfun/SparkFun_CCS811_Arduino_Library

  This example configures the nWAKE and nINT pins.
  The interrupt pin is configured to pull low when the data is
  ready to be collected.
  The wake pin is configured to enable the sensor during I2C communications

  Hardware Connections (Breakoutboard to Arduino):
  3.3V to 3.3V pin
  GND to GND pin
  SDA to A4
  SCL to A5
  NOT_INT to D6
  NOT_WAKE to D5 (For 5V arduinos, use resistor divider)
    D5---
         |
         R1 = 4.7K
         |
         --------NOT_WAKE
         |
         R2 = 4.7K
         |
        GND

  Resources:
  Uses Wire.h for i2c operation

  Development environment specifics:
  Arduino IDE 1.8.1

  This code is released under the [MIT License](http://opensource.org/licenses/MIT).

  Please review the LICENSE.md file included with this example. If you have any question
s
  or concerns with licensing, please contact techsupport@sparkfun.com.

  Distributed as-is; no warranty is given.
******************************************************************************/
#include <SparkFunCCS811.h>

#define CCS811_ADDR 0x5B //Default I2C Address
//#define CCS811_ADDR 0x5A //Alternate I2C Address

#define PIN_NOT_WAKE 5
#define PIN_NOT_INT 6
```

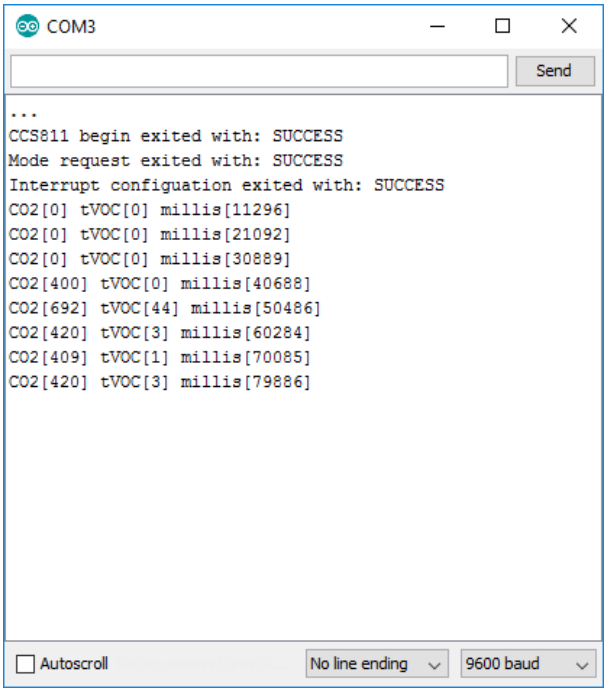
```
CCS811 myCCS811(CCS811_ADDR);

//Global sensor object
//----------------------------------------------------------------
void setup()
{
  //Start the serial
  Serial.begin(9600);
  Serial.println();
  Serial.println("...");

  CCS811Core::status returnCode;

  //This begins the CCS811 sensor and prints error status of .begin()
  returnCode = myCCS811.begin();
  Serial.print("CCS811 begin exited with: ");
  printDriverError( returnCode );
  Serial.println();

  //This sets the mode to 60 second reads, and prints returned error status.
  returnCode = myCCS811.setDriveMode(2);
  Serial.print("Mode request exited with: ");
  printDriverError( returnCode );
  Serial.println();

  //Configure and enable the interrupt line,
  //then print error status
  pinMode(PIN_NOT_INT, INPUT_PULLUP);
  returnCode = myCCS811.enableInterrupts();
  Serial.print("Interrupt configuation exited with: ");
  printDriverError( returnCode );
  Serial.println();

  //Configure the wake line
  pinMode(PIN_NOT_WAKE, OUTPUT);
  digitalWrite(PIN_NOT_WAKE, 1); //Start asleep

}
//----------------------------------------------------------------
void loop()
{
  //Look for interrupt request from CCS811
  if (digitalRead(PIN_NOT_INT) == 0)
  {
    //Wake up the CCS811 logic engine
    digitalWrite(PIN_NOT_WAKE, 0);
    //Need to wait at least 50 us
    delay(1);
    //Interrupt signal caught, so cause the CCS811 to run its algorithm
    myCCS811.readAlgorithmResults(); //Calling this function updates the global tVOC and
 CO2 variables

    Serial.print("CO2[");
```

```
    Serial.print(myCCS811.getCO2());
    Serial.print("] tVOC[");
    Serial.print(myCCS811.getTVOC());
    Serial.print("] millis[");
    Serial.print(millis());
    Serial.print("]");
    Serial.println();

    //Now put the CCS811's logic engine to sleep
    digitalWrite(PIN_NOT_WAKE, 1);
    //Need to be asleep for at least 20 us
    delay(1);
  }
  delay(1); //cycle kinda fast
}


//printDriverError decodes the CCS811Core::status type and prints the
//type of error to the serial terminal.
//
//Save the return value of any function of type CCS811Core::status, then pass
//to this function to see what the output was.
void printDriverError( CCS811Core::status errorCode )
{
  switch ( errorCode )
  {
    case CCS811Core::SENSOR_SUCCESS:
      Serial.print("SUCCESS");
      break;
    case CCS811Core::SENSOR_ID_ERROR:
      Serial.print("ID_ERROR");
      break;
    case CCS811Core::SENSOR_I2C_ERROR:
      Serial.print("I2C_ERROR");
      break;
    case CCS811Core::SENSOR_INTERNAL_ERROR:
      Serial.print("INTERNAL_ERROR");
      break;
    case CCS811Core::SENSOR_GENERIC_ERROR:
      Serial.print("GENERIC_ERROR");
      break;
    default:
      Serial.print("Unspecified error.");
  }
}

//printSensorError gets, clears, then prints the errors
//saved within the error register.
void printSensorError()
{
  uint8_t error = myCCS811.getErrorRegister();

  if ( error == 0xFF ) //comm error
  {
```

```
    Serial.println("Failed to get ERROR_ID register.");
  }
  else
  {
    Serial.print("Error: ");
    if (error & 1 << 5) Serial.print("HeaterSupply");
    if (error & 1 << 4) Serial.print("HeaterFault");
    if (error & 1 << 3) Serial.print("MaxResistance");
    if (error & 1 << 2) Serial.print("MeasModeInvalid");
    if (error & 1 << 1) Serial.print("ReadRegInvalid");
    if (error & 1 << 0) Serial.print("MsgInvalid");
    Serial.println();
  }
}
```

Notice that this example doesn't poll `dataAvailable()` to check if data is ready; instead it reads the state of a digital input. When the input is low, data is ready in the sensor and `readAlgorithmResults()`, then `.getCO2()` and `getTVOC()` are used as normal.

The $\overline{\text{WAK}}$ pin can be used to control the logic engine on the CCS811 to save a bit of power. When the $\overline{\text{WAK}}$ pin is low (or disconnected), the I^2C bus will respond to commands, but when the pin is high it will not. Tens of microseconds are required to wake or sleep, so in this example, commands are wrapped with a 1ms delay.



Example terminal output

The terminal displays the calculation every 10 seconds. You can see that it take a few samples for the algorithm to spit out data, even at a slow rate of acquisition. Between the sampling, power is decreased as much as possible.

Summary:

To use $\overline{\text{WAK}}$,

- Set up a digital output for the wake pin
- To communicate with a sleeping sensor,

- Set $\overline{\text{WAK}}$ low
- Wait 50us
- Do your communication
- Set $\overline{\text{WAK}}$ high
- Wait 20us

To use $\overline{\text{INT}}$,

- Set up a digital input for the interrupt pin
- Enable interrupts with enableInterrupts()
- Look for a falling edge on the input to detect the availability of new data

Example: Compensating for Climate

To have the CCS811 compensate for pressure and temperature conditions, obtain those metrics and pass to the sensor object with `setEnvironmentalData`.

The examples from the library show three different sources of data that can be used to calibrate the CCS811:

1. Randomly generated temperature and humidity data
2. Data from a supplemental BME280 sensor
3. Data collected by reading the NTC pins

This guide only shows the example that uses randomized data, as it can be used without additional components yet still illustrate the effects of different climates.

From Arduino Library and Usage,

- `status_t setEnvironmentalData( float relativeHumidity, float temperature )` — Sets the environmental conditions for compensation.
  - relativeHumidity in units of %, 0.00 through 100.0
  - temperature in degrees C, -25.0 through 50.0

Compensating with Random Data

A starting place for working with the compensation is the *setEnvironmentalReadings* example. After the same configuration from the basic example, this sketch applies a random temperature and humidity, then takes 10 reads and repeats.

```
/*************************************************************************
setEnvironmentalReadings.ino

Marshall Taylor @ SparkFun Electronics

April 4, 2017

https://github.com/sparkfun/CCS811_Air_Quality_Breakout
https://github.com/sparkfun/SparkFun_CCS811_Arduino_Library

Hardware Connections (Breakoutboard to Arduino):

  3.3V to 3.3V pin
  GND to GND pin
  SDA to A4
  SCL to A5

Generates random temperature and humidity data, and uses it to compensate the CCS811.
This just demonstrates how the algorithm responds to various compensation points.
Use NTCCompensated or BME280Compensated for real-world examples.

Resources:
Uses Wire.h for i2c operation

Development environment specifics:
Arduino IDE 1.8.1

This code is released under the [MIT License](http://opensource.org/licenses/MIT).

Please review the LICENSE.md file included with this example. If you have any questions
or concerns with licensing, please contact techsupport@sparkfun.com.

Distributed as-is; no warranty is given.
*************************************************************************/
float temperatureVariable = 25.0; //in degrees C
float humidityVariable = 65.0; //in % relative

#include <Wire.h>
#include "SparkFunCCS811.h"

#define CCS811_ADDR 0x5B //Default I2C Address
//#define CCS811_ADDR 0x5A //Alternate I2C Address

CCS811 myCCS811(CCS811_ADDR);

void setup()
{
    Serial.begin(9600);
    Serial.println("CCS811 EnvironmentalReadings Example");

    //This begins the CCS811 sensor and prints error status of .begin()
    CCS811Core::status returnCode = myCCS811.begin();
    Serial.print("begin exited with: ");
    printDriverError( returnCode );
```

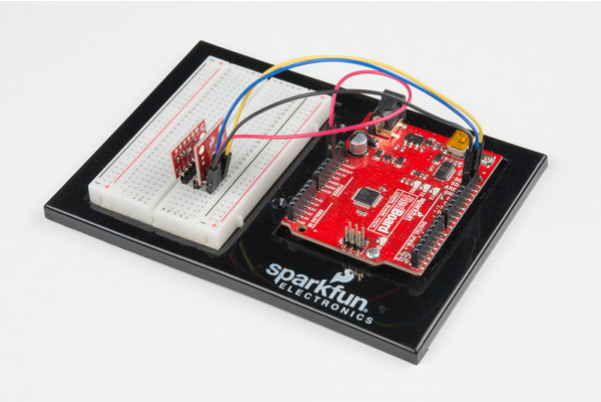
```
    Serial.println();

}

void loop()
{
    Serial.println();
    //Randomize the Temperature and Humidity
    humidityVariable = (float)random(0, 10000)/100; //0 to 100%
    temperatureVariable = (float)random(500, 7000)/100; // 5C to 70C
    Serial.println("New humidity and temperature:");
    Serial.print("  Humidity: ");
    Serial.print(humidityVariable, 2);
    Serial.println("% relative");
    Serial.print("  Temperature: ");
    Serial.print(temperatureVariable, 2);
    Serial.println(" degrees C");
    myCCS811.setEnvironmentalData(humidityVariable, temperatureVariable);

    Serial.println("Environmental data applied!");
    myCCS811.readAlgorithmResults(); //Dump a reading and wait
    delay(1000);
    //Print data points
    for( int i = 0; i < 10; i++)
    {
        if (myCCS811.dataAvailable())
        {
            //Calling readAlgorithmResults() function updates the global tVOC and CO2 va
riables
            myCCS811.readAlgorithmResults();

            Serial.print("CO2[");
            Serial.print(myCCS811.getCO2());
            Serial.print("] tVOC[");
            Serial.print(myCCS811.getTVOC());
            Serial.print("] millis[");
            Serial.print(millis());
            Serial.print("]");
            Serial.println();
        }
        else if (myCCS811.checkForStatusError())
        {
            //If the CCS811 found an internal error, print it.
            printSensorError();
        }
        delay(1000); //Wait for next reading
    }
}


//printDriverError decodes the CCS811Core::status type and prints the
//type of error to the serial terminal.
//
//Save the return value of any function of type CCS811Core::status, then pass
```

```
//to this function to see what the output was.
void printDriverError( CCS811Core::status errorCode )
{
    switch( errorCode )
    {
    case CCS811Core::SENSOR_SUCCESS:
        Serial.print("SUCCESS");
        break;
    case CCS811Core::SENSOR_ID_ERROR:
        Serial.print("ID_ERROR");
        break;
    case CCS811Core::SENSOR_I2C_ERROR:
        Serial.print("I2C_ERROR");
        break;
    case CCS811Core::SENSOR_INTERNAL_ERROR:
        Serial.print("INTERNAL_ERROR");
        break;
    case CCS811Core::SENSOR_GENERIC_ERROR:
        Serial.print("GENERIC_ERROR");
        break;
    default:
        Serial.print("Unspecified error.");
    }
}

//printSensorError gets, clears, then prints the errors
//saved within the error register.
void printSensorError()
{
    uint8_t error = myCCS811.getErrorRegister();

    if( error == 0xFF )//comm error
    {
        Serial.println("Failed to get ERROR_ID register.");
    }
    else
    {
        Serial.print("Error: ");
        if (error & 1 << 5) Serial.print("HeaterSupply");
        if (error & 1 << 4) Serial.print("HeaterFault");
        if (error & 1 << 3) Serial.print("MaxResistance");
        if (error & 1 << 2) Serial.print("MeasModeInvalid");
        if (error & 1 << 1) Serial.print("ReadRegInvalid");
        if (error & 1 << 0) Serial.print("MsgInvalid");
        Serial.println();
    }
}
```

Compensating with BME280 Data

If you have a BME280 sensor, they work great for getting the compensation parameters. Use the example *BME280Compensated* to see compensation using another sensor.

Connecting the two devices is as simple as putting them on the bus together.



A BME280 used in conjunction with the CCS811

View BME280Compensated.ino on github, or use the example from Arduino.

Compensating from NTC Thermistor Readings

Alternately, an NTC resistor can be placed in the provide PTH terminals, and the example *PTHCompensated* can be used to see how the internal ADC is used to calibrate for temperature only.

There is one caveat to this method: no humidity data! Partially compensated is better than uncompensated, so punch in an average humidity for your area, or leave the example's default at 50 percent.

View NTCCompensated.ino on github, or use the example from Arduino.

Resources and Going Further

Now that you've successfully got your CCS811 breakout up and running, and have scientifically proved which family member is the smelliest, here are some additional resources:

- CCS811 Datasheet-DS000459.pdf — PDF
- ASHRAE Allowable CO2 Levels.pdf — PDF
- CC-000774-AN-3-Assembly guidelines for CCS811.pdf — PDF
- CC-000783-AN-1-Mechanical Considerations for CCS811_0.pdf — PDF
- CCS811 Firmware Download AN000371.pdf — PDF
- CCS811 Programming Guide-AN000369.pdf — PDF
- CCS811 Thermistor Interface AN000372.pdf — PDF
- Indoor Air Quality Investigations TVOCs EU.pdf — PDF
- CCS811_Air_Quality_Breakout product repository — Design files and docs
- SparkFun_CCS811_Arduino_Library Repository — Source and example files for the Arduino library used in this tutorial.

Also, the following examples are included with the library but not discussed. They may help you on your way!

- *BaselineOperator* — Save and restore baselines to the EEPROM
- *Core* — Show how the underlying hardware object works
- *TwentyMinuteTest* — Report data with timestamp

Need some inspiration for your next project? Check out some of these sensor-related tutorials:

MAX31855K Thermocouple Breakout Hookup Guide

Learn how to take readings with a k-type thermocouple using the MAX31855K cold-junction-compensated k-type thermocouple-to-digital converter.

MPU-9250 Hookup Guide

Get up and running with the MPU-9250 9-axis MEMS sensor.

Qwiic Distance Sensor (RFD77402) Hookup Guide

The RFD77402 uses an infrared VCSEL (Vertical Cavity Surface Emitting Laser) TOF (Time of Flight) module capable of millimeter precision distance readings up to 2 meters. It's also part of SparkFun's Qwiic system, so you won't have to do any soldering to figure out how far away things are.

Interactive 3D Printed LED Diamond Prop

In this tutorial, we will learn about how to create an interactive theatrical prop for a performance by 3D printing a translucent diamond prop using a non-addressable RGB LED strip and AT42QT1011 capacitive touch sensing.

Or maybe check out our tests using the sensor at SparkFun!

Air Quality Sensor Experiment

JULY 19, 2017