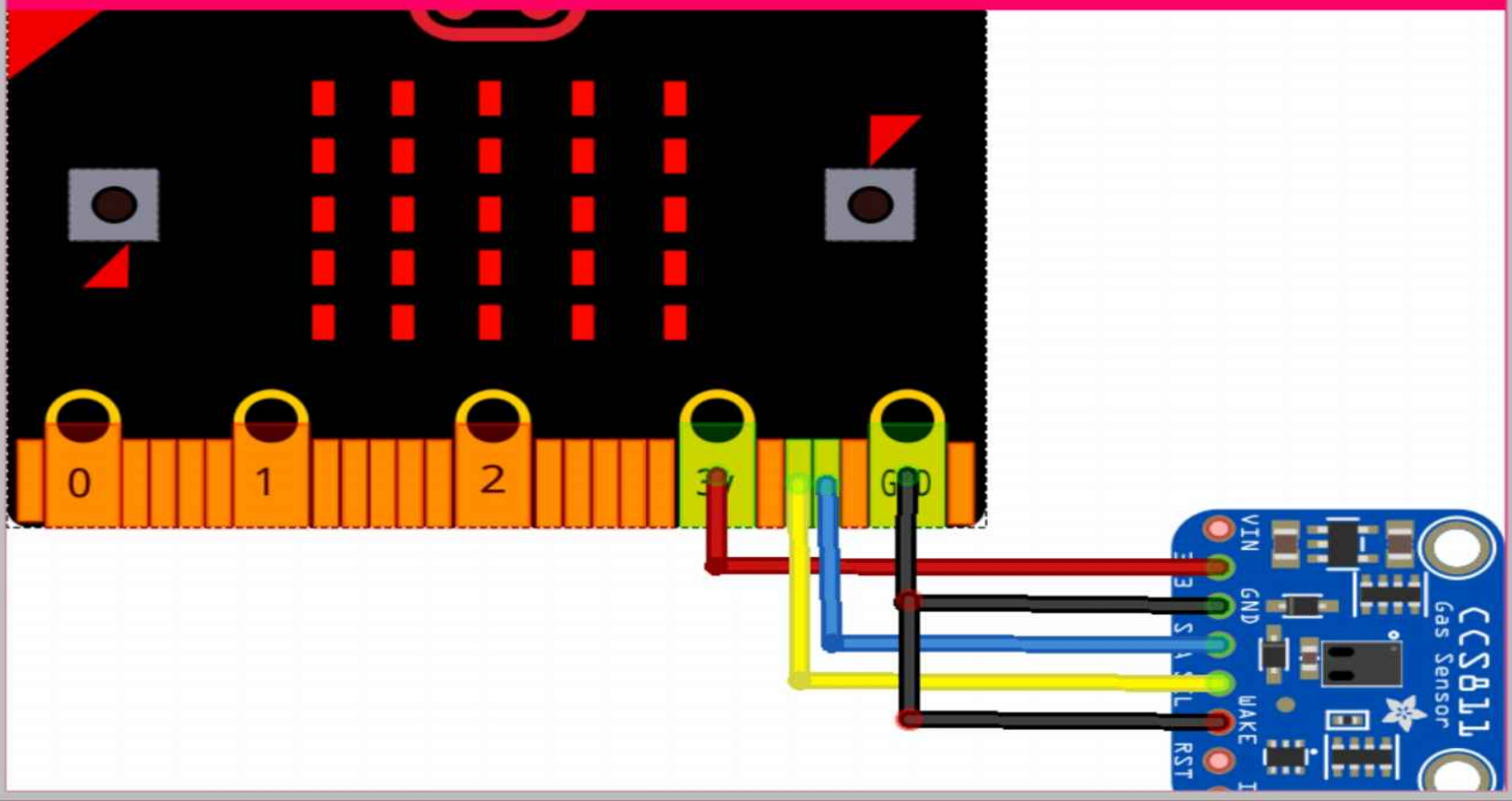
Learn the Micro:Bit using the Arduino IDE

IAIN HENDRY



Learn the Micro:Bit using the Arduino IDE

By

Iain Hendry

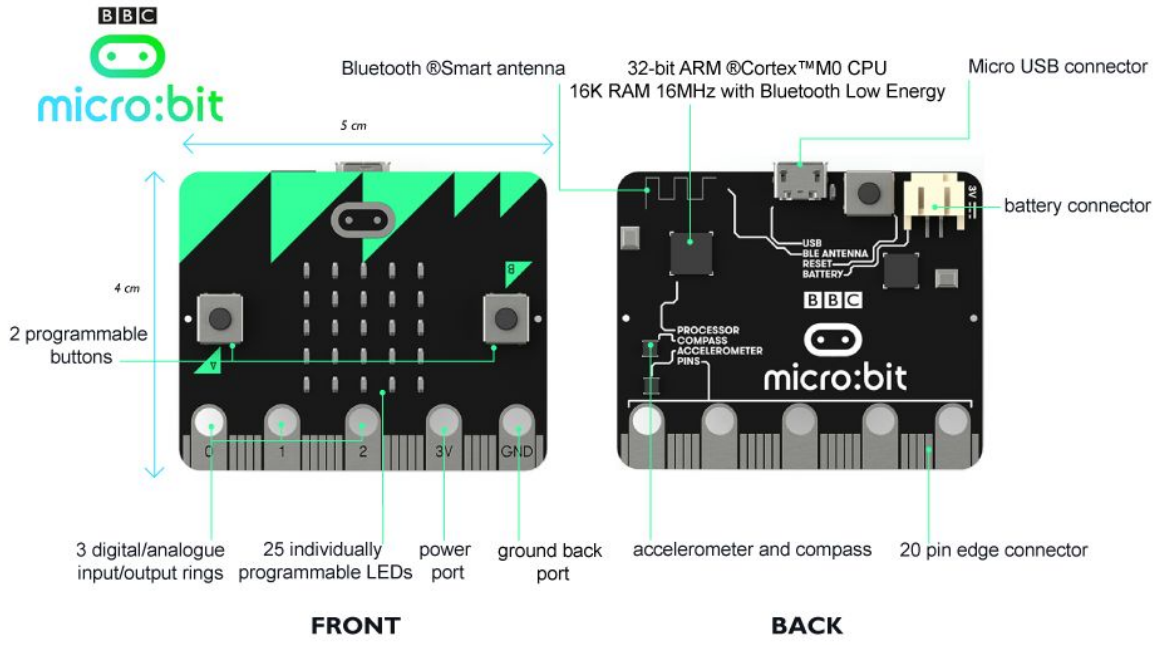
Table of Contents

OPT3001 Digital Ambient Light Sensor
MAG3110 magnetic sensor example
MMA7660 accelerometer example
HDC1080 humidity and temperature sensor example
micro:bit and OLED display example
bme280 environmental sensor example
MICRO:BIT and I2C LCD example
micro:bit and TM1637 7 segment display example
Si7021 I2C Humidity and Temperature Sensor example
TEMT6000 ambient light sensor
Microbit and GUVA-S12SD UV Sensor
AM2320 temperature and humidity sensor
DHT12 temperature sensor
micro:bit and TCS34725 Color Sensor
micro:bit and HMC5883L magnetometer example
MCP4725 DAC example
SHT31 temperature and humidity sensor
micro:bit and bmp180 sensor example
In review

Introducing the Micro:bit

If you have no idea what a Micro:Bit is hen this will introduce you to this intriguing board

The Micro Bit measures 4 × 5 cm the device has an ARM Cortex-M0 processor, accelerometer and magnetometer sensors, Bluetooth and USB connectivity, a display consisting of 25 LEDs, two programmable buttons, and can be powered by either USB or an external battery pack. The device inputs and outputs are through five ring connectors that are part of the 23-pin edge connector.



Hardware

The size of the device is described as half the size of a credit card, measuring 43 x 52 mm and, as of the start of final manufacturing,[18] includes: Nordic nRF51822 – 16 MHz 32-bit ARM Cortex-M0 microcontroller, 256 KB flash memory, 16 KB static ram, 2.4 GHz

Bluetooth low energy wireless networking. The ARM core has the capability to switch between 16 MHz or 32.768 kHz. NXP/Freescale KL26Z – 48 MHz ARM Cortex-M0+ core microcontroller, that includes a full-speed USB 2.0 On-The-Go (OTG) controller, used as a communication interface between USB and main Nordic microcontroller.

NXP/Freescale MMA8652 – 3-axis accelerometer sensor via I²C-bus.
NXP/Freescale MAG3110 – 3-axis magnetometer sensor via I²C-bus (to act as a compass and metal detector).
MicroUSB connector, battery connector, 23-pin edge connector. Display consisting of 25 LEDs in a 5×5 array.
Three tactile pushbuttons (two for user, one for reset).

I/O includes three ring connectors (plus one power one ground) which accept crocodile clips or 4 mm banana plugs as well as a 23-pin edge connector with two or three PWM outputs, six to 17 GPIO pins (depending on configuration), six analog inputs, serial I/O, SPI, and I²C.

Unlike early prototypes, which had an integral battery, an external battery pack (AAA batteries) can be used to power the device as a standalone or wearable product.

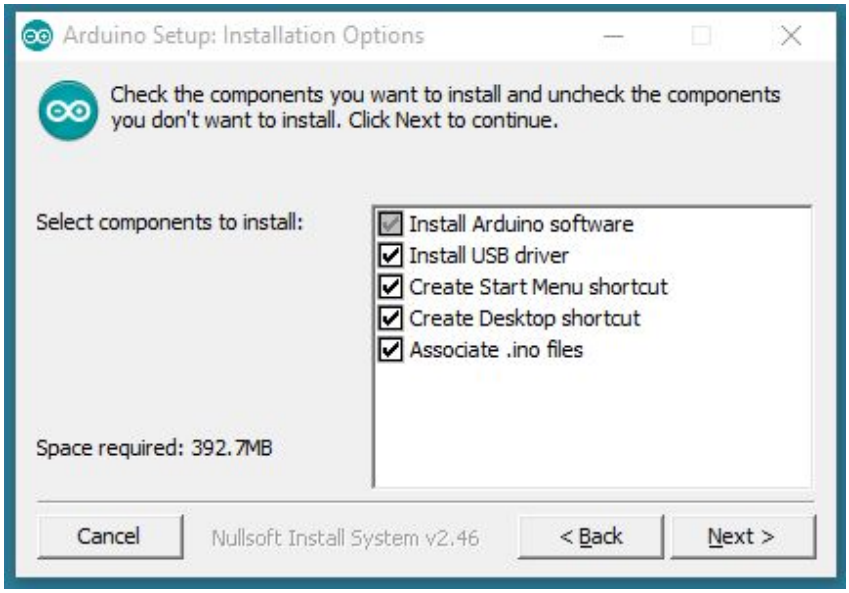
Setting up the Arduino IDE

Install the Arduino IDE

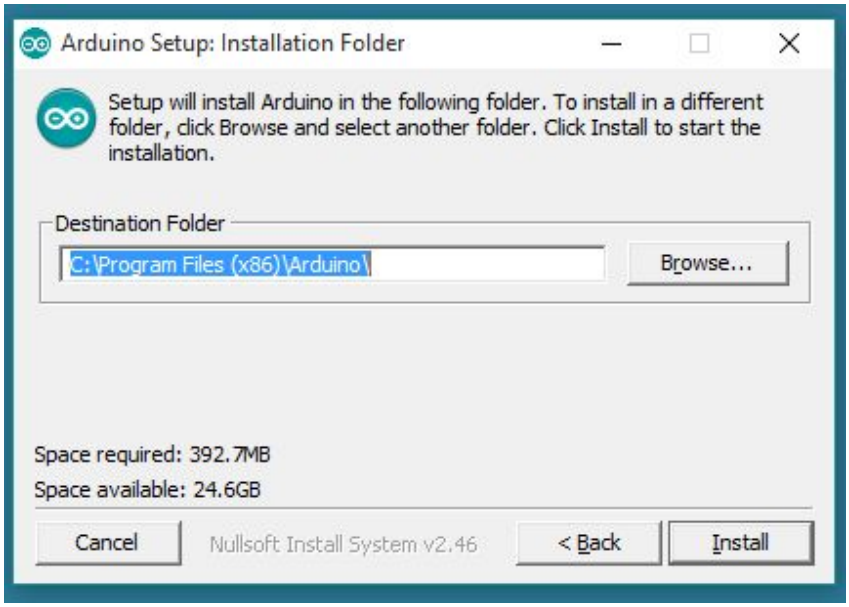
Download the latest version from https://www.arduino.cc/en/Main/Software

You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.

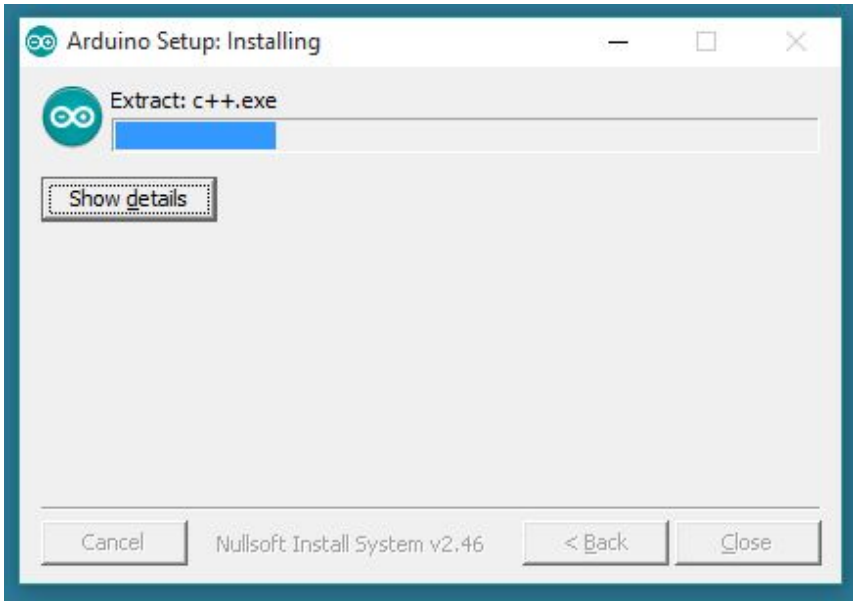
When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.



Choose the components to install



Choose the installation directory (we suggest to keep the default one)



The process will extract and install all the required files to execute properly the Arduino Software (IDE)

Adding support for the Micro:bit

You can now use the Arduino IDE to develop and program your Micro:Bit, in my view thats a good thing to add this support as its a popular development tool that is used for many other boards Arduino assumes there's a 'softdevice' radio already installed.

Flashing a SoftDevice

This is the instructions from https://github.com/sandeepmistry/arduino-nRF5 cd <SKETCHBOOK>, where <SKETCHBOOK> is your Arduino Sketch folder:

OS X: ~/Documents/Arduino Linux: ~/Arduino
Windows: ~/Documents/Arduino

Create the following directories: tools/nRF5FlashSoftDevice/tool/
Download nRF5FlashSoftDevice.jar to <SKETCHBOOK>/tools/nRF5FlashSoftDevice/tool/

Restart the Arduino IDE

Select your nRF board from the Tools -> Board menu

Select a SoftDevice from the Tools -> "SoftDevice: " menu

Select a Programmer (J-Link, ST-Link V2, or CMSIS-DAP) from the Tools -> "Programmer: " menu

Select Tools -> nRF5 Flash SoftDevice

Read license agreement

Click "Accept" to accept license and continue, or "Decline" to decline and abort

If accepted, SoftDevice binary will be flashed to the board or Download this zip file, extract it and drag it into your MICROBIT drive - http://www.microbitlearning.com/wp-content/uploads/2017/11/microbit-adv.zip

You should also add support for the BLE_Peripheral library, Adafruit micro:bit library via the Arduino library manager

Micro:Bit Features

LED matrix

The micro:bit contains 25 LEDs in a matrix, in this chapter we look at some examples

Code Examples

These use the adafruit library

Example 1

Flash all the LEDs on and off

```
#include <Adafruit_Microbit.h>

Adafruit_Microbit_Matrix microbit;

void setup()
{
  microbit.begin();
}

void loop()
{
  // Fill screen
  microbit.fillScreen(LED_ON);
  delay(1000);
  //empty screen
  microbit.fillScreen(LED_OFF);
  delay(1000);
}
```

Example 2

Display a yes and no (tick and cross)

```
#include <Adafruit_Microbit.h>

Adafruit_Microbit_Matrix microbit;

void setup()
{
  microbit.begin();
}

void loop()
{
  // draw a no
  microbit.show(microbit.NO);
  delay(1000);
  // draw a yes
  microbit.show(microbit.YES);
  delay(1000);
}
```

Example 3

Display a line and pixel in the middle of the screen

```
#include <Adafruit_Microbit.h>

Adafruit_Microbit_Matrix microbit;

void setup()
{
  microbit.begin();
}

void loop()
{
```

```
  microbit.drawPixel(2, 2, LED_ON); //draw a pixel
  microbit.drawLine(0, 0, 4, 0, LED_ON); //draw a line
}
```

Example 4

Draw a string on the led matrix

```
#include <Adafruit_Microbit.h>

Adafruit_Microbit_Matrix microbit;

void setup()
{
  microbit.begin();
}

void loop()
{
  microbit.print("HELLO WORLD");
}
```

Buttons

As we saw in the overview for the micro:bit there are 2 onboard buttons, its very easy to use this in the Arduino IDE

Code

Lets play with the push buttons You may have to increase the delay(150), I used this to try and get rid of simple multiple presses and switch bounce

```
const int buttonA = 5;
const int buttonB = 11;

void setup()
{
 Serial.begin(9600);
 pinMode(buttonA, INPUT);
 pinMode(buttonB, INPUT);
}

void loop()
{
 if (! digitalRead(buttonA))
 {
  Serial.println("Button A pressed");
 }
 if (! digitalRead(buttonB))
 {
  Serial.println("Button B pressed");
 }
 delay(150);
}
```

Testing

Open the serial monitor and press the buttons

Button A pressed
Button B pressed
Button A pressed
Button A pressed
Button B pressed
Button B pressed
Button B pressed

micro:bit onboard 3-axis accelerometer using the Arduino IDE

The MMA8653FC is an intelligent, low-power, three-axis, capacitive micromachined accelerometer with 10 bits of resolution. This accelerometer is packed with embedded functions with flexible user-programmable options, configurable to two interrupt pins. Embedded interrupt functions enable overall power savings, by relieving the host processor from continuously polling data. There is access to either low-pass or high-pass filtered data, which minimizes the data analysis required for jolt detection and faster transitions. The device can be configured to generate inertial wake-up interrupt signals from any combination of the configurable embedded functions, enabling the MMA8653FC to monitor inertial events while remaining in a low-power mode during periods of inactivity.

Features
• 1.95 V to 3.6 V supply voltage
• 1.62 V to 3.6 V digital interface voltage
• ±2 g, ±4 g, and ±8 g dynamically selectable full-scale ranges
• Output Data Rates (ODR) from 1.56 Hz to 800 Hz
• 10-bit digital output
• I2C digital output interface with programmable interrupts
• One embedded channel of configurable motion detection (Freefall)
• Orientation (Portrait/Landscape) detection with fixed hysteresis of 15°.
• Configurable automatic ODR change triggered by the Auto-Wake/Sleep state change
• Self-Test

Typical applications
• Tilt compensation in e-compass applications
• Static orientation detection (Portrait/Landscape, Up/Down, Left/Right, Back/ Front position identification)
• Notebook, tablet, e-reader, and laptop tumble and freefall detection
• Real-time orientation detection (virtual reality and gaming 3D user

orientation feedback)
• Real-time activity analysis (pedometer step counting, freefall drop detection for HDD, dead-reckoning GPS backup)
• Motion detection for portable product power saving (Auto-SLEEP and Auto-WAKE for cell phone, PDA, GPS, gaming)
• Shock and vibration monitoring (mechatronic compensation, shipping and warranty usage logging)
• User interface (tilt menu scrolling)

Code

You need to the https://github.com/hidnseek/hidnseek/tree/master/arduino/libraries/MMA8653

```
#include "Wire.h"
#include "MMA8653.h"

MMA8653 accel;

void setup()
{
 Serial.begin(9600);
 accel.begin(false, 2); // 8-bit mode, 2g range
}


void loop()
{
 accel.update();
 Serial.print((float)accel.getX() * 0.0156);
 Serial.print(" , ");
 Serial.print((float)accel.getY() * 0.0156);
 Serial.print(", ");
 Serial.println((float)accel.getZ() * 0.0156);
 delay(250);
}
```

Output

Open the serial monitor and move the micro:bit - the values are in g's

0.66 , -0.06, 0.69
0.22 , -0.19, 1.25
0.09 , 0.50, -0.50
0.86 , -0.17, 0.37

0.17 , 0.12, -1.03
0.23 , -0.31, -1.37
0.64 , 1.51, 1.03

micro:bit magnetometer with the Arduino IDE

Freescale's MAG3110 is a small, low-power, digital 3-axis magnetometer. The device can be used in conjunction with a 3-axis accelerometer to realize an orientation independent electronic compass that can provide accurate heading information. It features a standard I2C serial interface output and smart embedded functions.

The MAG3110 is capable of measuring magnetic fields with an output data rate (ODR) up to 80 Hz; these output data rates correspond to sample intervals from 12.5 ms to several seconds. The MAG3110 is available in a plastic DFN package and it is guaranteed to operate over the extended temperature range of -40°C to +85°C.

You have a micro:bit so you already have one on the board

Code

You need to import the Sparkfun library for the MAG3110 to run this example - https://github.com/sparkfun/SparkFun_MAG3110_Breakout_Board_Arduino_Library/archive/master.zip

```
#include <SparkFun_MAG3110.h>

MAG3110 mag = MAG3110();

void setup()
{
  Serial.begin(9600);
  mag.initialize(); //Initializes the mag sensor
  mag.start();      //Puts the sensor in active mode
}

void loop() {
```

```
  int x, y, z;
  //Only read data when it's ready
  if(mag.dataReady())
  {
    //Read the data
    mag.readMag(&x, &y, &z);

    Serial.print("X: ");
    Serial.print(x);
    Serial.print(", Y: ");
    Serial.print(y);
    Serial.print(", Z: ");
    Serial.println(z);

    Serial.println("--------");
    delay(1000);
  }
}
```

Testing

Open the serial monitor and move your micro:bit through the various axis, you should see something like this

```
 X: 218, Y: 65109, Z: 65274
--------
X: 71, Y: 64965, Z: 49
--------
X: 598, Y: 65036, Z: 248
--------
X: 423, Y: 65338, Z: 564
```

Basic Examples

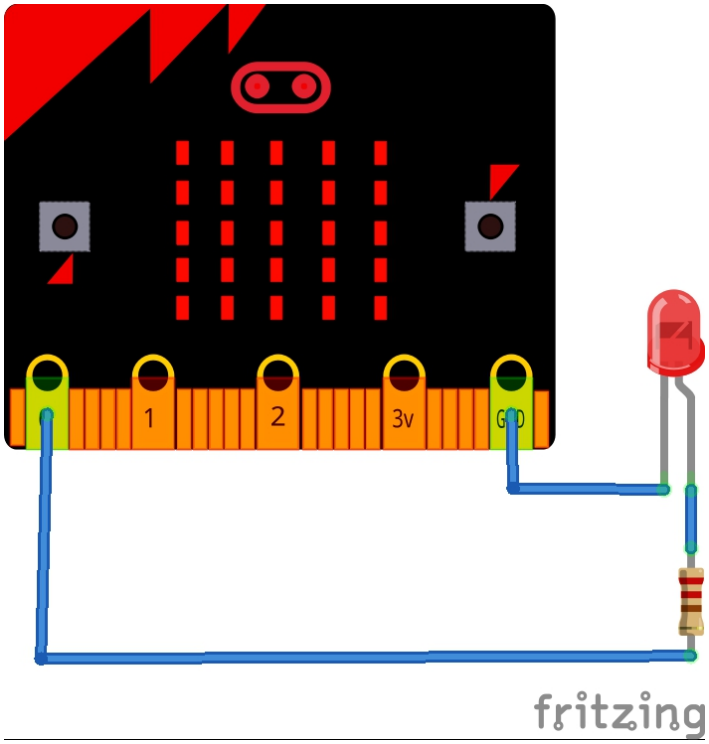
External LED example

The micro:bit contains many led's on the matrix but in this first basic example we connect an external LED to the micro:bit. You can use crocodile clips and connect up to a resistor and LED as per the schematic but I use an expansion board which allows you to connect modules using standard connecting cables.

In the picture below you can see one of these expansion boards connected to a micro:bit

Schematic

I used 0 you could use any I/O pin you want



Code

```
int led = 0;

void setup()
{
  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH);
}

void loop()
{
  digitalWrite(led, LOW); // turn the LED on
  delay(1000);
  digitalWrite(led, HIGH); // turn the LED off by making the voltage LOW
  delay(1000);
}
```

Arduino and RGB LED example

In this example we will connect an RGB led to our micr:bit, lets look at RGB leds first
RGB LEDs consist of one red, one green, and one blue LED.

By independently adjusting each of the three, RGB LEDs are capable of producing a wide color gamut. Unlike dedicated-color LEDs, however, these obviously do not produce pure wavelengths. Moreover, such modules as commercially available are often not optimized for smooth color mixing.

There are two primary ways of producing white light-emitting diodes (WLEDs), LEDs that generate high-intensity white light. One is to use individual LEDs that emit three primary colors[95]—red, green, and blue—and then mix all the colors to form white light. The other is to use a phosphor material to convert monochromatic light from a blue or UV LED to broad-spectrum white light, much in the same way a fluorescent light bulb works. It is important to note that the 'whiteness' of the light produced is essentially engineered to suit the human eye, and depending on the situation it may not always be appropriate to think of it as white light.

There are three main methods of mixing colors to produce white light from an LED:

blue LED + green LED + red LED (color mixing; can be used as backlighting for displays)
near-UV or UV LED + RGB phosphor (an LED producing light with a wavelength shorter than blue's is used to excite an RGB

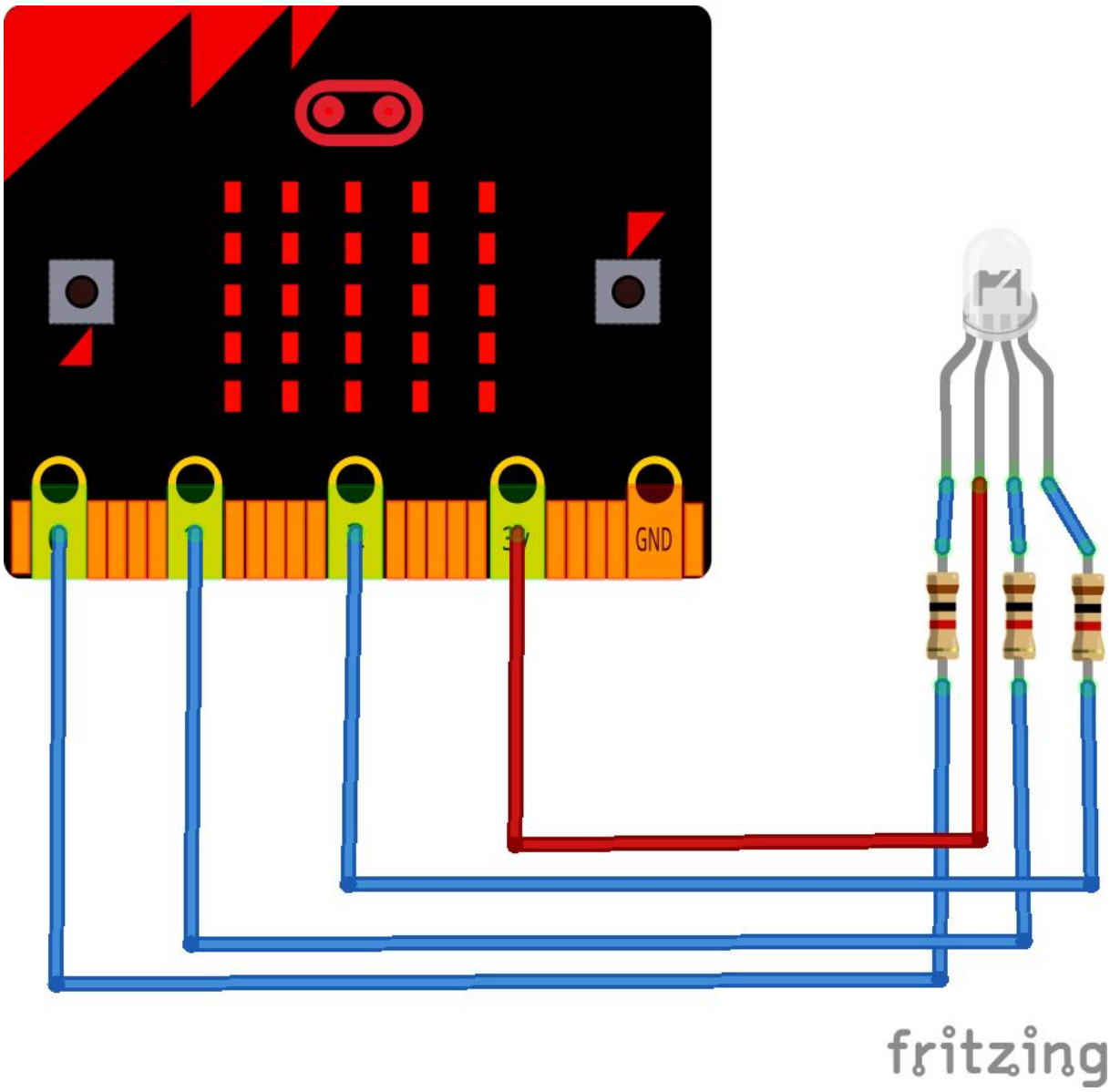
phosphor)
blue LED + yellow phosphor (two complementary colors combine to form white light; more efficient than first two methods and more commonly used)[96]

Because of metamerism, it is possible to have quite different spectra that appear white. However, the appearance of objects illuminated by that light may vary as the spectrum varies.

Here is a picture of the RGB LED module I used, this is a common anode type.

Schematic

Here is a rough layout, the LED and resistors are basically the same as the module above


3V
GND
fritzing

Code

Cycle through the 3 main LED colours

```
int red = 0;
int green = 1;
int blue = 2;

// the setup routine runs once when you press reset:
void setup()
{
// initialize the digital pin as an output.
pinMode(red, OUTPUT);
pinMode(green, OUTPUT);
pinMode(blue, OUTPUT);
digitalWrite(red, HIGH);
digitalWrite(green, HIGH);
digitalWrite(blue, HIGH);
}

// the loop routine runs over and over again forever:
void loop() {
digitalWrite(red, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(red, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
digitalWrite(green, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(green, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
digitalWrite(blue, LOW); // turn the LED on
delay(1000); // wait for a second
digitalWrite(blue, HIGH); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Micro:bit and LDR example using the Arduino IDE

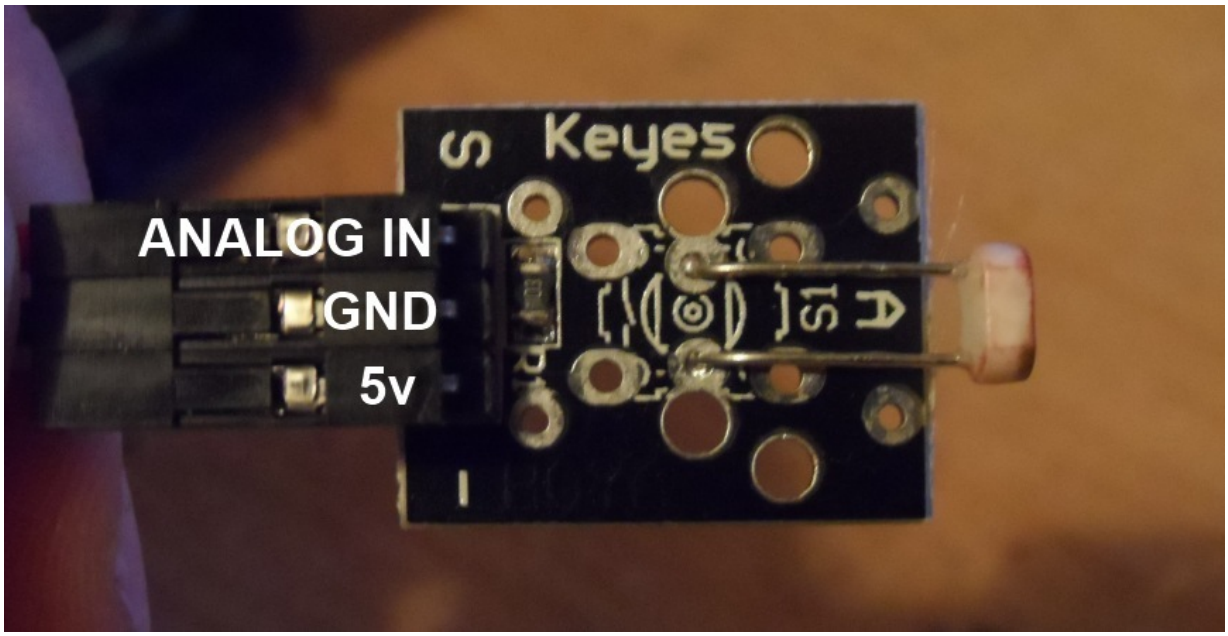
In this example we connect a photoresistor to a micro:bit, the value read from the photoresistor corresponds to the amount of light present. The photoresistor is connected to P3 in this example.

A **photoresistor** (or **light-dependent resistor**, **LDR**, or **photocell**) is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibitsphotoconductivity. A photoresistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits. A photoresistor is made of a high resistance semiconductor.

In the dark, a photoresistor can have a resistance as high as several megohms (MΩ), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

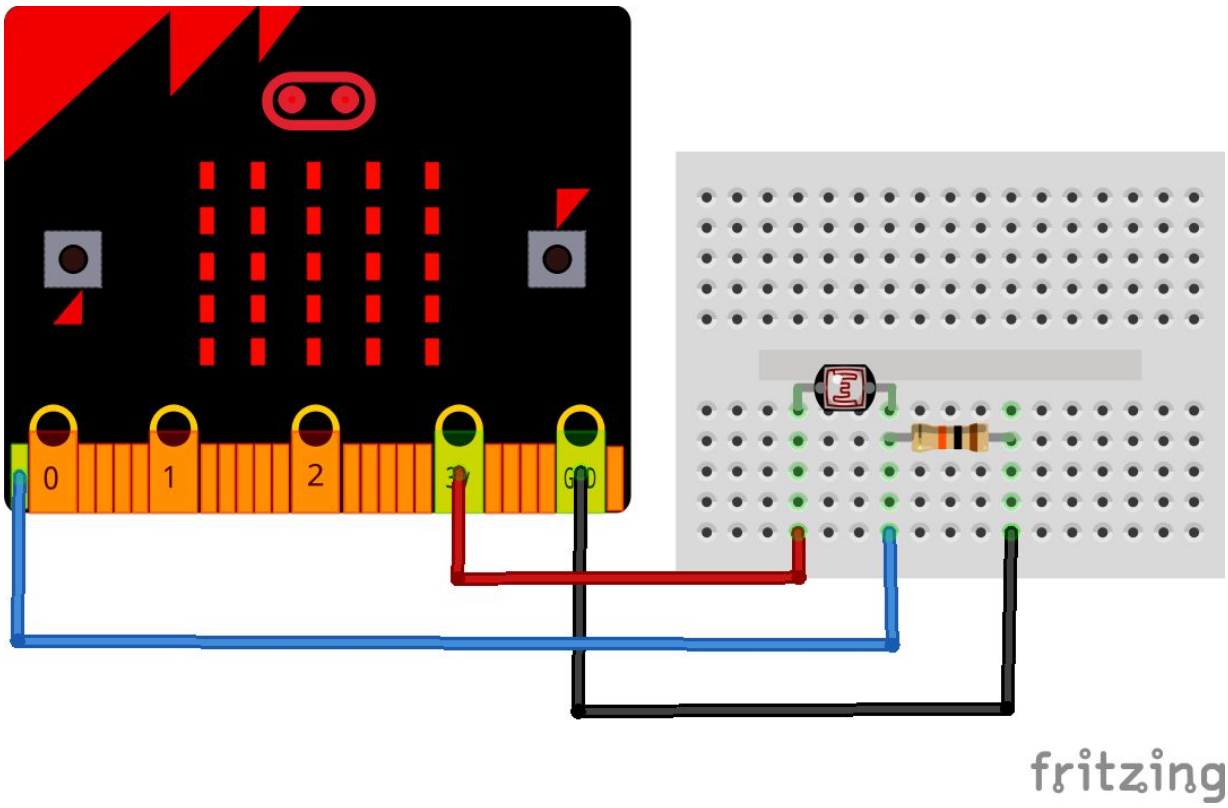
A practical example could be a dark room sensor for photography, if the reading approached a critical level an alarm could be activated or even a night light

Here is a sample module


Keyes
ANALOG IN
GND
5v

Schematic

In this example I connected 3v3 to the module and it worked fine



Code

In this example we simply output the reading via the serial port.

```
int sensorValue;

void setup()
{
  Serial.begin(9600); // starts the serial port at 9600
}

void loop()
```

```
{
  sensorValue = analogRead(A0); // read analog input pin 0
  Serial.print(sensorValue, DEC); // prints the value read
  Serial.print(" \n"); // prints a space between the numbers
  delay(1000); // wait 100ms for next reading
}
```

Testing

Open the serial monitor and move the LDR closer to a light, cover the LDR.

10
4
254
254
112
142
282
665
742
983

Basic Sensor Projects

Temperature readings using a TMP102

In this example we look at the TMP102 digital sensor and we will connect it up to a Micro:bit, again the code is written in the Arduino IDE

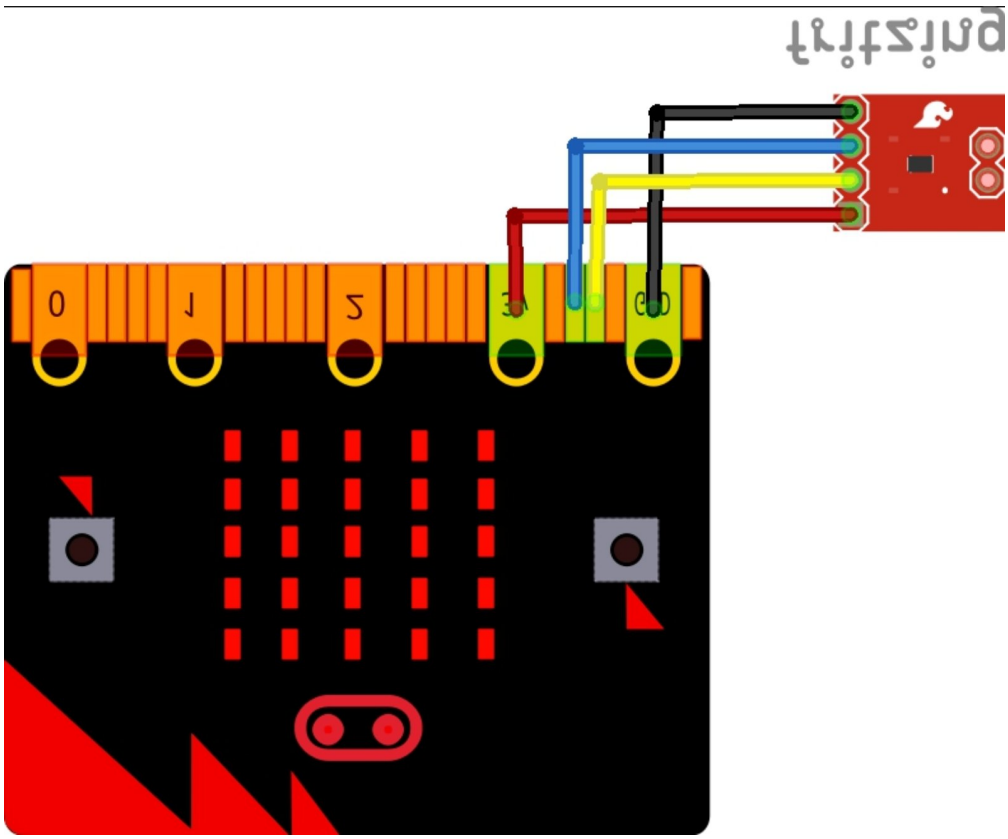
The TMP102 device is a digital temperature sensor ideal for NTC/PTC thermistor replacement where high accuracy is required. The device offers an accuracy of ±0.5°C without requiring calibration or external component signal conditioning. Device temperature sensors are highly linear and do not require complex calculations or lookup tables to derive the temperature. The on-chip 12-bit ADC offers resolutions down to 0.0625°C.

The TMP102 device features SMBus™, two-wire and I^2C interface compatibility, and allows up to four devices on one bus.

The device also features an SMBus alert function. The device is specified to operate over supply voltages from 1.4 to 3.6 V with the maximum quiescent current of 10 µA over the full operating range.

The TMP102 device is ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications. The device is specified for operation over a temperature range of –40°C to 125°C.

Layout



Parts List

Name	**Link**
Micro:bit	Micro:bit Development Board
expansion board	Edge Breakout I/O Expansion Extension Board for BBC micro:bit
TMP102 breakout	TMP102 Digital Temperature Sensor Breakout Board Module 12-bit 1.4V To 3.6VDC Sensor Module Temperature Module
connecting wire	Free shipping Dupont line 120pcs 20cm male to male + male to female and female to female jumper wire

Code

This is from the Arduino site with a few tweaks

```
#include "Wire.h"
#define TMP102_I2C_ADDRESS 72 /* This is the I2C address for our chip. This value is correct if you tie the ADD0 pin to ground. See the datasheet for some other values. */

void setup()
{
  Wire.begin(); // start the I2C library
  Serial.begin(115200); //Start serial communication at 115200 baud
}


void loop()
{
  getTemp102();
  delay(5000); //wait 5 seconds before printing our next set of readings.
}

void getTemp102()
{
  byte firstbyte, secondbyte; //these are the bytes we read from the TMP102 temperature registers
  int val; /* an int is capable of storing two bytes, this is where we "chuck" the two bytes together. */
  float convertedtemp; /* We then need to multiply our two bytes by a scaling factor, mentioned in the datasheet. */
  float correctedtemp;

  /* Reset the register pointer (by default it is ready to read temperatures)
You can alter it to a writeable register and alter some of the configuration -
the sensor is capable of alerting you if the temperature is above or below a specified threshold. */

  Wire.beginTransmission(TMP102_I2C_ADDRESS); //Say hi to the sensor.
  Wire.write((byte)0x00);
  Wire.endTransmission();
```

```
  Wire.requestFrom(TMP102_I2C_ADDRESS, 2);
  Wire.endTransmission();


  firstbyte      = (Wire.read());
  /*read the TMP102 datasheet - here we read one byte from
   each of the temperature registers on the TMP102*/
  secondbyte     = (Wire.read());
  /*The first byte contains the most significant bits, and
   the second the less significant */
    val = firstbyte;
    if ((firstbyte & 0x80) > 0)
    {
      val |= 0x0F00;
    }
    val <<= 4;
/* MSB */
    val |= (secondbyte >> 4);
/* LSB is ORed into the second 4 bits of our byte.
Bitwise maths is a bit funky, but there's a good tutorial on the playground*/
    convertedtemp = val*0.0625;
    correctedtemp = convertedtemp - 5;
    /* See the above note on overreading */


  Serial.print("firstbyte is ");
  Serial.print("\t");
  Serial.println(firstbyte, BIN);
  Serial.print("secondbyte is ");
  Serial.print("\t");
  Serial.println(secondbyte, BIN);
  Serial.print("Concatenated byte is ");
  Serial.print("\t");
  Serial.println(val, BIN);
  Serial.print("Converted temp is ");
  Serial.print("\t");
  Serial.println(val*0.0625);
```

```
  Serial.print("Corrected temp is ");
  Serial.print("\t");
  Serial.println(correctedtemp);
  Serial.println();
}
```

Output

Open the serial monitor window and you should see something like this

firstbyte is 10110 secondbyte is 11000000
Concatenated byte is 101101100
Converted temp is 22.75
Corrected temp is 17.75

and so on

Temperature, pressure and altitude with a BMP280

This time we look at a BMP280, once again we have a quick overview of the sensor

BMP280 is an absolute barometric pressure sensor especially designed for mobile applications. The sensor module is housed in an extremely compact package. Its small dimensions and its low power consumption allow for the implementation in battery powered devices such as mobile phones, GPS modules or watches.

As its predecessor BMP180, BMP280 is based on Bosch's proven Piezo-resistive pressure sensor technology featuring high accuracy and linearity as well as long term stability and high EMC robustness. Numerous device operation options offer highest flexibility to optimize the device regarding power consumption, resolution and filter performance.

A tested set of default settings for example use case is provided to the developer in order to make design-in as easy as possible.

Parts List

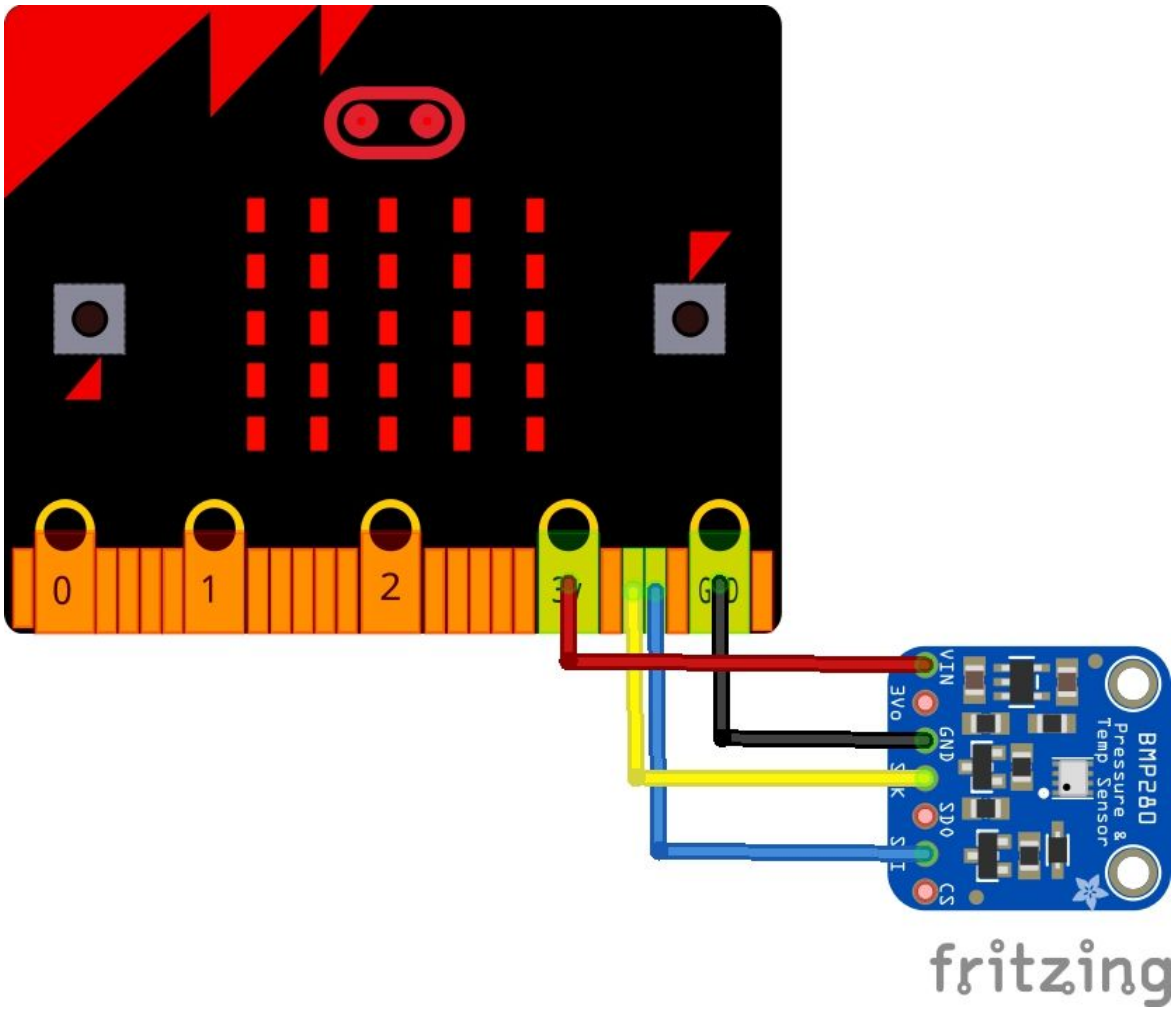
I chose a part that actually included 2 sensors on it a BMP280 and an Si7021, you can of course locate a module that has only a BMP280

name	**Link**
Micro:bit	Micro:bit Development Board
GY-21p module	Atmospheric Humidity Temperature Sensor Breakout Barometric Pressure BMP280 SI7021 for Arduino GY-21P

connecting wire	Free shipping Dupont line 120pcs 20cm male to male + male to female and female to female jumper wire

Layout

This is a layout diagram using an adafruit part for my sensor, my module had clearly marked SDA and SCL connections



Code

This example requires the following libraries to be installed, you can add these using the Arduino IDE- https://github.com/adafruit/Adafruit_BMP280_Library and https://github.com/adafruit/Adafruit_Sensor

Most, if not all the hard work is done in this library. Basically, in this example we simply print the values out via the serial monitor. The library has several functions for temperature, pressure and altitude

```
/***************************************************************************
This is a library for the BMP280 humidity, temperature & pressure sensor

Designed specifically to work with the Adafruit BMEP280 Breakout
----> http://www.adafruit.com/products/2651

These sensors use I2C or SPI to communicate, 2 or 4 pins are required
to interface.

Adafruit invests time and resources providing this open source code,
please support Adafruit andopen-source hardware by purchasing products
from Adafruit!

Written by Limor Fried & Kevin Townsend for Adafruit Industries.
BSD license, all text above must be included in any redistribution
***************************************************************************/

#include <Wire.h>
```

```
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bme; // I2C


void setup() {
Serial.begin(9600);
Serial.println(F("BMP280 test"));

if (!bme.begin()) {
Serial.println("Could not find a valid BMP280 sensor, check wiring!");
while (1);
}
}

void loop() {
Serial.print("Temperature = ");
Serial.print(bme.readTemperature());
Serial.println(" *C");

Serial.print("Pressure = ");
Serial.print(bme.readPressure());
Serial.println(" Pa");

Serial.print("Approx altitude = ");
Serial.print(bme.readAltitude(1013.25)); // this should be adjusted to your local forcase
Serial.println(" m");

Serial.println();
delay(2000);
}
```

Output

Open the serial monitor and you should see something like this

Temperature = 27.65 *C
Pressure = 100592.44
Pa Approx altitude = 61.17 m
and so on repeated every 2 seconds

Temperature measurement using an LM75 sensor

The LM75 temperature sensor includes a delta-sigma analog-to-digital converter, and a digital overtemperature detector. The host can query the LM75 through its I²C interface to read temperature at any time. The open-drain overtemperature output (OS) sinks current when the programmable temperature limit is exceeded.

The OS output operates in either of two modes, comparator or interrupt. The host controls the temperature at which the alarm is asserted (TOS) and the hysteresis temperature below which the alarm condition is not valid (THYST).

Also, the LM75's TOS and THYST registers can be read by the host. The address of the LM75 is set with three pins to allow multiple devices to work on the same bus. Power-up is in comparator mode, with defaults of TOS = +80°C and THYST = +75°C. The 3.0V to 5.5V supply voltage range, low supply current, and I²C interface make the LM75 ideal for many applications in thermal management and protection.

Parts List

name	**Link**
Micro:bit	Micro:bit Development Board
LM75	LM75A Temperature Sensor I2C Interface Temperature Module
connecting wire	Free shipping Dupont line 120pcs 20cm male to male + male to female and female to female jumper wire

Connection

LM75 sensor	**Micro:bit**

SDA	20 (SDA)
SCL	19 (SCL)
Vcc	3v
Gnd	GND
DS	N/C

Code

I used this library for this example
- https://github.com/thefekete/LM75

```
#include <Wire.h>
#include <LM75.h>

LM75 sensor;  // initialize an LM75 object
// You can also initiate with another address as follows:
//LM75 sensor(LM75_ADDRESS | 0b001);  // if A0->GND, A1->GND and A2->Vcc

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
  // get temperature from sensor
  Serial.print("Current temp: ");
  Serial.print(sensor.temp());
  Serial.println(" C");

  // Tos Set-point
  sensor.tos(47.5); // set at 47.5'C
  Serial.print("Tos set at ");
  Serial.print(sensor.tos());
  Serial.println(" C");
```

```
  // Thyst Set-point
  sensor.thyst(42); // set at 42'C
  Serial.print("Thyst set at ");
  Serial.print(sensor.thyst());
  Serial.println(" C");

  // shutdown the sensor and wait a while
  sensor.shutdown(true);
  delay(3000);
  // wake up sensor for next time around
  sensor.shutdown(false);

  Serial.println();
}
```

Output

Open the serial monitor

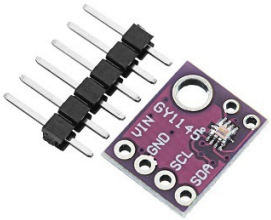
Current temp: 19.00
C Tos set at 47.50
C Thyst set at 42.00 C

Micro:bit and Si1145 sensor example

Another sensor reaches the desk, this time it is a Si1145 sensor, we will connect it to a Micro:bit and test it out, we will be using the Adduino IDE for this example

The Si1145/46/47 is a low-power, reflectance-based, infrared proximity, ultraviolet (UV) index, and ambient light sensor with I2C digital interface and programmableevent interrupt output. This touchless sensor IC includes an analog-to-digital converter, integrated high-sensitivity visible and infrared photodiodes, digital signal processor, and one, two, or three integrated infrared LED drivers with fifteen selectable drive levels. The Si1145/46/47 offers excellent performance under a wide dynamic range and

a variety of light sources including direct sunlight. The Si1145/46/47 can also work under dark glass covers.

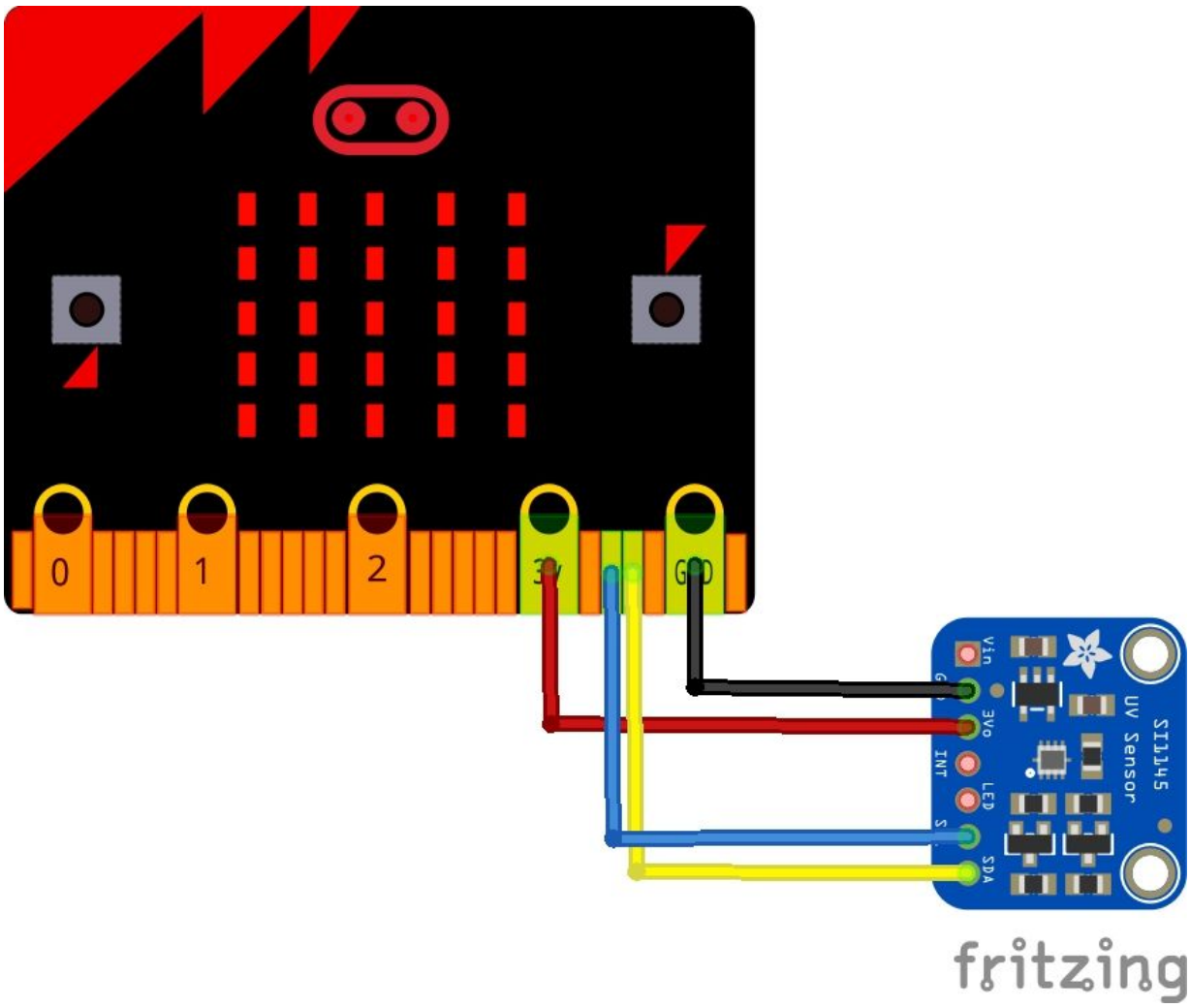


The photodiode response and associated digital conversion circuitry provide excellent immunity to artificial light flicker noise and natural light flutter noise. With two or more LEDs, the Si1146/47 is capable of supporting multiple-axis proximity motion detection. The Si1145/46/47 devices are provided in a 10-lead 2x2 mm QFN package and are capable of operation from 1.71 to 3.6 V over the –40 to +85 °C temperature range.

Parts List

name	**Link**
Microbit	Micro:bit Development Board
expansion board	Edge Breakout I/O Expansion Extension Board for BBC micro:bit
Si1145 breakout	SI1145 UV IR Visible Sensor I2C GY1145 Light Breakout Board Module
connecting wire	Free shipping Dupont line 120pcs 20cm male to male + male to female and female to female jumper wire

Layout



Code

This example requires the Adafruit Si1145 library, you can add this via the library manager or download from https://github.com/adafruit/Adafruit_SI1145_Library

```
#include <Wire.h>
#include "Adafruit_SI1145.h"

Adafruit_SI1145 uv = Adafruit_SI1145();

void setup() {
  Serial.begin(9600);
```

```
  Serial.println("Adafruit SI1145 test");

  if (! uv.begin()) {
    Serial.println("Didn't find Si1145");
    while (1);
  }

  Serial.println("OK!");
}

void loop() {
  Serial.println("====================");
  Serial.print("Vis: "); Serial.println(uv.readVisible());
  Serial.print("IR: "); Serial.println(uv.readIR());

  // Uncomment if you have an IR LED attached to LED pin!
  //Serial.print("Prox: "); Serial.println(uv.readProx());

  float UVindex = uv.readUV();
  // the index is multiplied by 100 so to get the
  // integer index, divide by 100!
  UVindex /= 100.0;
  Serial.print("UV: ");  Serial.println(UVindex);

  delay(1000);
}
```

Output

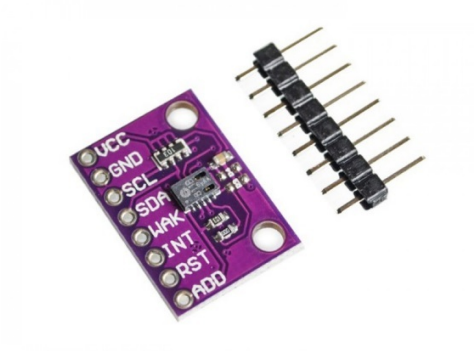
Open the serial monitor window and you will see something like this. These were my readings.

Vis: 265
IR: 375
UV: 0.05
====================
Vis: 266
IR: 384

UV: 0.05
==========
Vis: 267
IR: 379
UV: 0.05

CCS811 digital gas sensor and micro:bit example

CCS811 is a low-power digital gas sensor solution, which integrates a gas sensor solution for detecting low levels of VOCs typically found indoors, with a microcontroller unit (MCU) and an Analog-to-Digital converter to monitor the local environment and provide an indication of the indoor air quality via an equivalent CO2 or TVOC output over a standard I2C digital interface.



Features

Integrated MCU
On-board processing
Standard digital interface
Optimised low power modes
IAQ threshold alarms
Programmable baseline
2.7mm x 4.0mm LGA package
Low component count
Proven technology platform

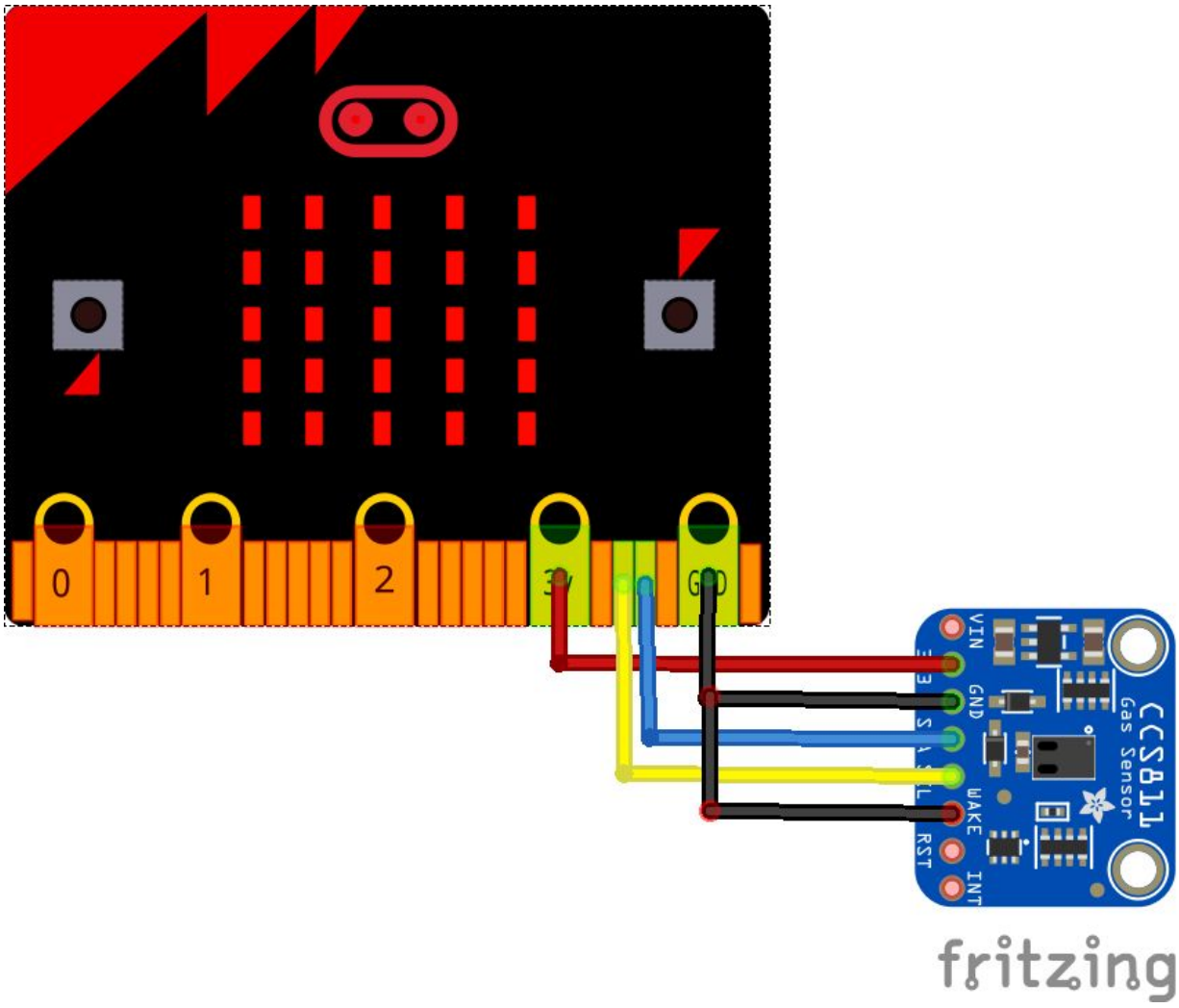
Specs

Interface	I²C
Supply Voltage [V]	1.8 to 3.6
Power Consumption [mW]	1.2 to 46
Dimension [mm]	2.7 x 4.0 x 1.1 LGA

Ambient Temperature Range [°C]	-40 to 85
Ambient Humidity Range [% r.h.]	10 to 95

Schematics/Layout

Remember and connect WAKE to gnd



Code

Again we use a library - And this is the out of the box example

```
#include "Adafruit_CCS811.h"
Adafruit_CCS811 ccs;

void setup()
{
Serial.begin(9600);
Serial.println("CCS811 test");
if(!ccs.begin()){
Serial.println("Failed to start sensor! Please check your wiring.");
while(1);
}
//calibrate temperature sensor
while(!ccs.available());
float temp = ccs.calculateTemperature();
ccs.setTempOffset(temp - 25.0);
}

void loop()
{
if(ccs.available()){
float temp = ccs.calculateTemperature();
if(!ccs.readData()){
Serial.print("CO2: ");
Serial.print(ccs.geteCO2());
Serial.print("ppm, TVOC: ");
Serial.print(ccs.getTVOC());
Serial.print("ppb Temp:");
Serial.println(temp);
}
else{
Serial.println("ERROR!");
while(1);
}
```

```
}
delay(500);
}
```

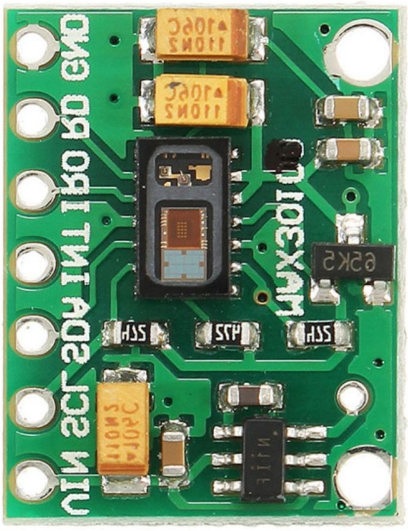
Output

Open the serial monitor - this is what I saw. The higher CO2 level was when I breathed on the sensor

CO2: 400ppm, TVOC: 0ppb Temp:35.34
CO2: 400ppm, TVOC: 0ppb Temp:34.42
CO2: 400ppm, TVOC: 0ppb Temp:34.04
CO2: 770ppm, TVOC: 56ppb Temp:32.90

MAX30102 pulse and heart-rate monitor sensor and micro:bit

The MAX30102 is an integrated pulse oximetry and heart-rate monitor biosensor module. It includes internal LEDs, photodetectors, optical elements, and low-noise electronics with ambient light rejection. The MAX30102 provides a complete system solution to ease the design-in process for mobile and wearable devices.



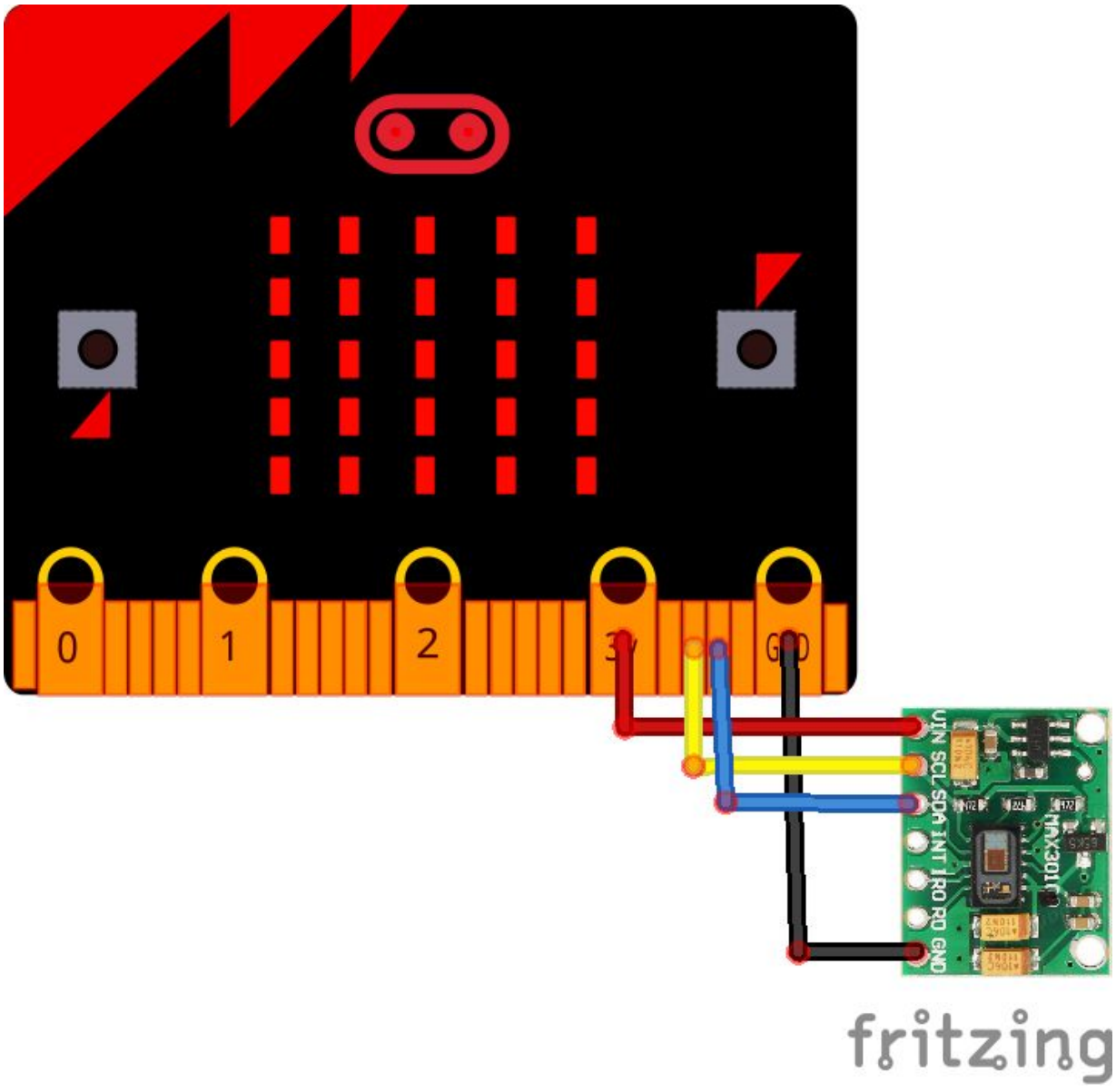
The MAX30102 operates on a single 1.8V power supply and a separate 5.0V power supply for the internal LEDs. Communication is through a standard I2C-compatible interface. The module can be shut down through software with zero standby current, allowing the power rails to remain powered at all times.

Key Features

Heart-Rate Monitor and Pulse Oximeter Biosensor in LED Reflective Solution
Tiny 5.6mm x 3.3mm x 1.55mm 14-Pin Optical Module
Integrated Cover Glass for Optimal, Robust Performance

Ultra-Low Power Operation for Mobile Devices
Programmable Sample Rate and LED Current for Power Savings
Low-Power Heart-Rate Monitor (< 1mW)
Ultra-Low Shutdown Current (0.7µA, typ)
Fast Data Output Capability
High Sample Rates
Robust Motion Artifact Resilience
High SNR
-40°C to +85°C Operating Temperature Range

Schematics/Layout



Code

Again we use a library and again its an sparkfun one - https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library

```
#include <Wire.h>
#include "MAX30105.h"

#include "heartRate.h"

MAX30105 particleSensor;

const byte RATE_SIZE = 4; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; //Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

void setup()
{
Serial.begin(115200);
Serial.println("Initializing...");

// Initialize sensor
if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
{
Serial.println("MAX30105 was not found. Please check wiring/power. ");
while (1);
}
Serial.println("Place your index finger on the sensor with steady pressure.");

particleSensor.setup(); //Configure sensor with default settings
particleSensor.setPulseAmplitudeRed(0x0A); //Turn Red LED to low to indicate sensor is running
particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED
}
```

```
void loop()
{
long irValue = particleSensor.getIR();

if (checkForBeat(irValue) == true)
{
//We sensed a beat!
long delta = millis() - lastBeat;
lastBeat = millis();

beatsPerMinute = 60 / (delta / 1000.0);

if (beatsPerMinute < 255 && beatsPerMinute > 20)
{
rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
rateSpot %= RATE_SIZE; //Wrap variable

//Take average of readings
beatAvg = 0;
for (byte x = 0 ; x < RATE_SIZE ; x++)
beatAvg += rates[x];
beatAvg /= RATE_SIZE;
}
}

Serial.print("IR=");
Serial.print(irValue);
Serial.print(", BPM=");
Serial.print(beatsPerMinute);
Serial.print(", Avg BPM=");
Serial.print(beatAvg);

if (irValue < 50000)
Serial.print(" No finger?");

Serial.println();
}
```

Output

Open the serial monitor - this is what I saw, it took a little bit of time for the readings to appear

IR=111297, BPM=32.68, Avg BPM=64
IR=111451, BPM=32.68, Avg BPM=64
IR=111519, BPM=32.68, Avg BPM=64
IR=111557, BPM=85.35, Avg BPM=66
IR=111449, BPM=85.35, Avg BPM=66

VEML6040 color sensor and micro:bit

VEML6040 color sensor senses red, green, blue, and white light and incorporates photodiodes, amplifiers, and analog / digital circuits into a single chip using CMOS process.

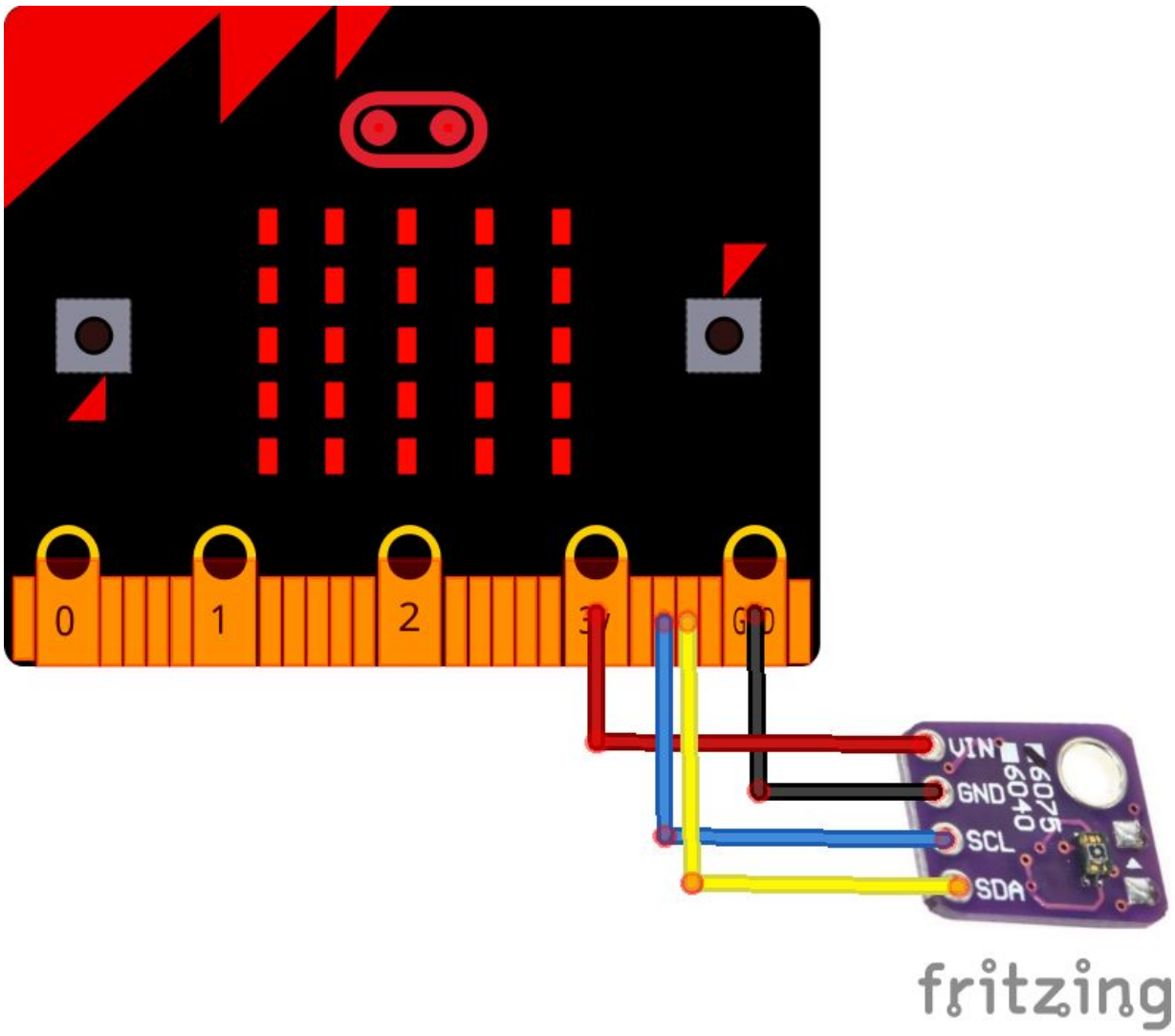
With the color sensor applied, the brightness, and color temperature of backlight can be adjusted base on ambient light source that makes panel looks more comfortable for end user's eyes.

VEML6040's adoption of FiltronTM technology achieves the closest ambient light spectral sensitivity to real human eye responses.

VEML6040 provides excellent temperature compensation capability for keeping the output stable under changing temperature. VEML6040's function are easily operated via the simple command format of I2C (SMBus compatible) interface protocol.

VEML6040's operating voltage ranges from 2.5 V to 3.6 V.

Schematics/Layout



Code

Again we use a library - https://github.com/thewknd/VEML6040

This example worked just fine

```
#include "Wire.h"
#include "veml6040.h"
VEML6040 RGBWSensor;
void setup() {
Serial.begin(9600);
Wire.begin();
if(!RGBWSensor.begin()) {
Serial.println("ERROR: couldn't detect the sensor");
while(1){}
}
/*
* init RGBW sensor with:
* - 320ms integration time
* - auto mode
* - color sensor enable
*/
RGBWSensor.setConfiguration(VEML6040_IT_320MS + VEML6040_AF_AUTO +
VEML6040_SD_ENABLE);
delay(1500);
Serial.println("Vishay VEML6040 RGBW color sensor auto mode example");
Serial.println("CCT: Correlated color temperature in \260K");
Serial.println("AL: Ambient light in lux");
delay(1500);
}
void loop() {
Serial.print("RED: ");
Serial.print(RGBWSensor.getRed());
Serial.print(" GREEN: ");
Serial.print(RGBWSensor.getGreen());
Serial.print(" BLUE: ");
Serial.print(RGBWSensor.getBlue());
```

```
Serial.print(" WHITE: ");
Serial.print(RGBWSensor.getWhite());
Serial.print(" CCT: ");
Serial.print(RGBWSensor.getCCT());
Serial.print(" AL: ");
Serial.println(RGBWSensor.getAmbientLight());
delay(400);
}
```

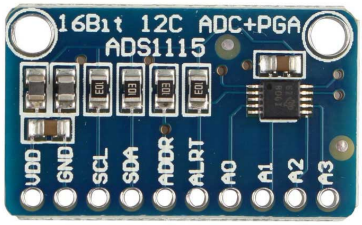
Output

Open the serial monitor - this is what I saw

RED: 2681 GREEN: 1532 BLUE: 657 WHITE: 4341 CCT: 2027 AL: 48.20
RED: 2696 GREEN: 1543 BLUE: 661 WHITE: 4362 CCT: 2031 AL: 48.54
RED: 2617 GREEN: 1493 BLUE: 641 WHITE: 4257 CCT: 2024 AL: 46.97

ADS1115 analog-to-digital converter and micro:bit

The ADS1115 device (ADS111x) is a precision, low-power, 16-bit, I2C-compatible, analog-to-digital converters (ADCs) offered in an ultra-small, leadless, X2QFN-10 package, and a VSSOP-10 package. The ADS111x devices incorporate a low-drift voltage reference and an oscillator. The ADS1114 and ADS1115 also incorporate a programmable gain amplifier (PGA) and a digital comparator. These features, along with a wide operating supply range, make the ADS111x well suited for power- and space-constrained, sensor measurement applications.



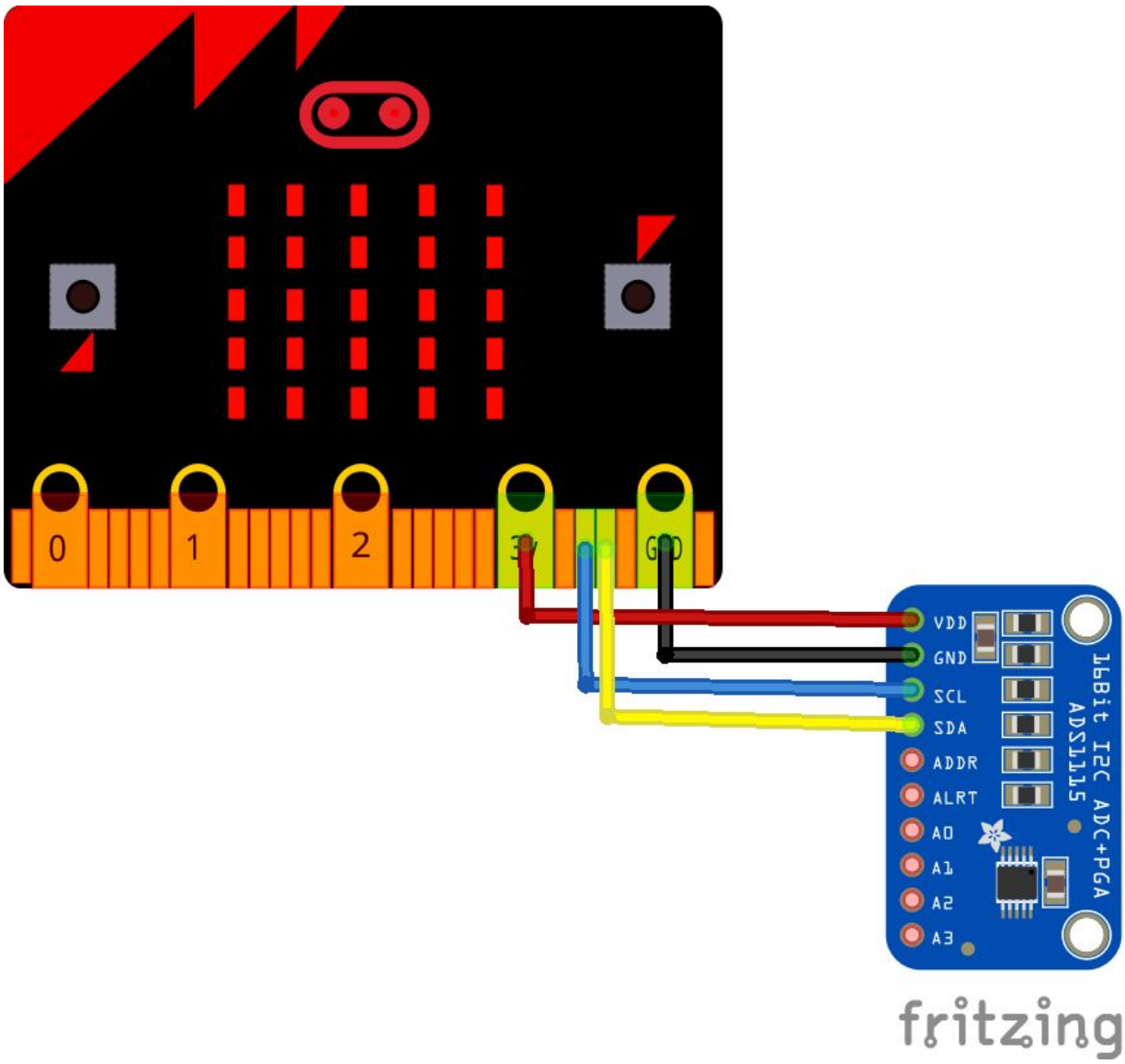
The ADS111x perform conversions at data rates up to 860 samples per second (SPS). The PGA offers input ranges from ±256 mV to ±6.144 V, allowing precise large- and small-signal measurements. The ADS1115 features an input multiplexer (MUX) that allows two differential or four single-ended input measurements. Use the digital comparator in the ADS1114 and ADS1115 for under- and overvoltage detection.

The ADS111x operate in either continuous-conversion mode or single-shot mode. The devices are automatically powered down after one conversion in single-shot mode; therefore, power consumption is significantly reduced during idle periods.

Features

Wide Supply Range: 2.0 V to 5.5 V
Low Current Consumption: 150 μA
(Continuous-Conversion Mode)
Programmable Data Rate: 8 SPS to 860 SPS
Single-Cycle Settling
Internal Low-Drift Voltage Reference
Internal Oscillator
I2C Interface: Four Pin-Selectable Addresses
Four Single-Ended or Two Differential Inputs (ADS1115)
Programmable Comparator (ADS1114 and ADS1115)
Operating Temperature Range: –40°C to +125°C

Schematics/Layout


0
1
2
3V
GND
VDD
GND
SCL
SDA
ADDR
ALRT
A0
A1
A2
A3
16Bit I2C ADC+PGA
ADS1115
fritzing

Code

Again we use a library and again its an adafruit one - https://github.com/adafruit/Adafruit_ADS1X15

```
#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads(0x48);

void setup(void)
{
Serial.begin(9600);
Serial.println("Hello!");

Serial.println("Getting single-ended readings from AIN0..3");
Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV/ADS1015, 0.1875mV/ADS1115)");

ads.begin();
}

void loop(void)
{
int16_t adc0, adc1, adc2, adc3;

adc0 = ads.readADC_SingleEnded(0);
adc1 = ads.readADC_SingleEnded(1);
adc2 = ads.readADC_SingleEnded(2);
adc3 = ads.readADC_SingleEnded(3);
Serial.print("AIN0: ");
Serial.println(adc0);
Serial.print("AIN1: ");
Serial.println(adc1);
Serial.print("AIN2: ");
Serial.println(adc2);
Serial.print("AIN3: ");
```

```
Serial.println(adc3);
Serial.println(" ");

delay(1000);
}
```

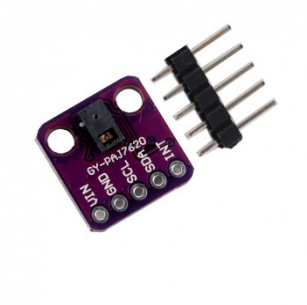
Output

Open the serial monitor - this is what I saw, I only had AIN2 connected to 0v, the other inputs were not connected

AIN0: 3052
AIN1: 3060
AIN2: -2
AIN3: 3052

PAJ7620 gesture sensor and micro:bit example

The PAJ7620 integrates gesture recognition function with general I2C interface into a single chip forming an image analytic sensor system. It can recognize 9 human hand gesticulations such as moving up, down, left, right, forward, backward, circle-clockwise, circle-counter clockwise, and waving.

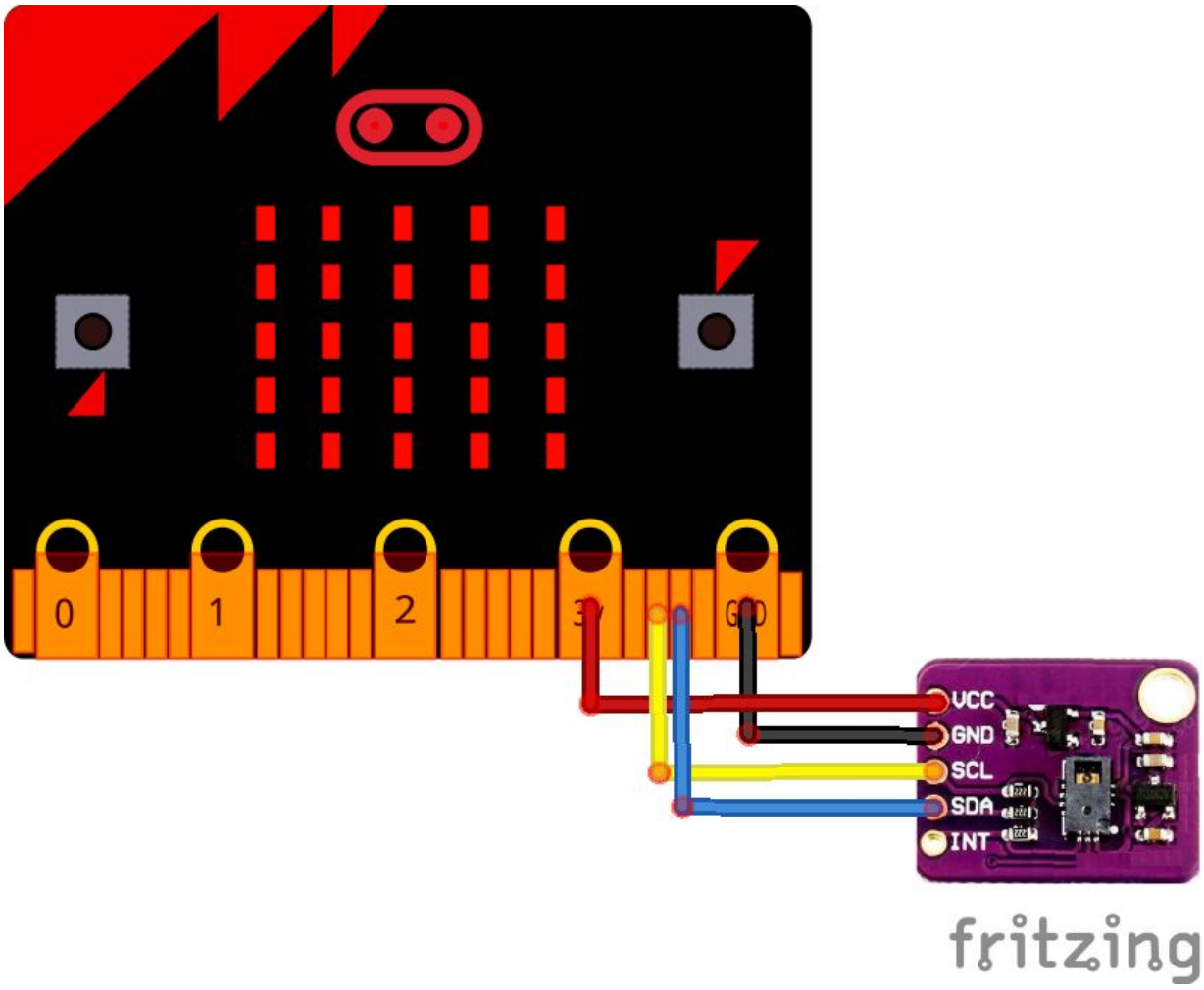


It also offers built-in proximity detection in sensing approaching or departing object from the sensor. The PAJ7620 is designed with great flexibility in power-saving mechanism, well suit for low power battery operated HMI devices.

The PAJ7620 is packaged into module form in-built with IR LED and optics lens as a complete sensor solution.

Gesture Mode : Single-Object Detection
Image Array Size : 60 x 60
Sampling Rate : 240fps
Operating Distance : 20cm
Output Types : 9 types of Gestures: Up, Down, Left, Right, Forward, Backward, Circle-Clockwise, Circle-Counter Clockwise, and Waving)
Cursor Mode : Yes
Interface : I2C/SPI
Light Source : Integrated IR LED with optics

Schematics/Layout



Code

Again we use a library - https://www.elecrow.com/wiki/images/7/72/Paj7620.zip

```
#include <Wire.h>
#include "paj7620.h"

void setup()
{
  Serial.begin(9600);
  paj7620Init();
}

void loop()
{
  uint8_t data = 0; // Read Bank_0_Reg_0x43/0x44 for gesture result.

  paj7620ReadReg(0x43, 1, &data);

  if (data == GES_UP_FLAG)
  {
  Serial.println("GES_UP_FLAG !");
  }

  if (data == GES_DOWN_FLAG)
  {
  Serial.println("GES_DOWN_FLAG !");
  }

  if(data == GES_FORWARD_FLAG)
  {
  Serial.println("GES_FORWARD_FLAG !");
  }

  if(data == GES_BACKWARD_FLAG)
```

```
  {
  Serial.println("GES_BACKWARD_FLAG !");
  }

  if(data == GES_RIGHT_FLAG)
  {
  Serial.println("GES_RIGHT_FLAG !");
  }

  if(data ==GES_LEFT_FLAG)
  {
  Serial.println("GES_LEFT_FLAG !");
  }

}
```

Output

Open the serial monitor - this is what I saw, move your hands in various directions

```
INIT SENSOR...
Addr0 =20, Addr1 =76
wake-up finish.
Paj7620 initialize register finished.
GES_UP_FLAG !
GES_BACKWARD_FLAG !
GES_UP_FLAG !
GES_UP_FLAG !
```

MLX90615 infrared thermometer and micro:bit example

The MLX90615 is a miniature infrared thermometer for non-contact temperature measurements. Both the IR sensitive thermopile detector chip and the signal conditioning ASIC are integrated in the same miniature TO-46 can.

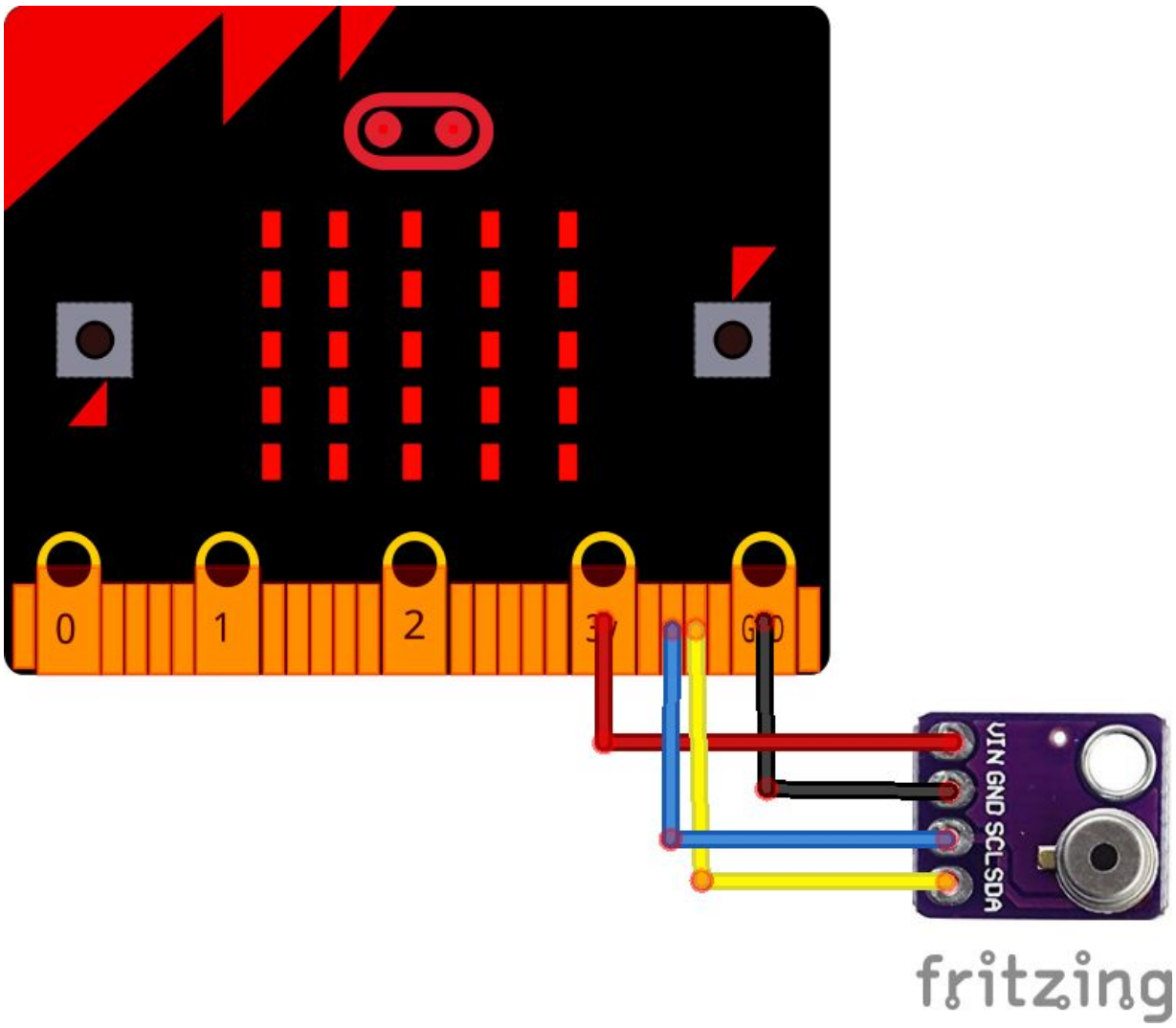
The infrared thermometer comes factory calibrated with a digital SMBus output giving full access to the measured temperature in the complete temperature range(s) with a resolution of 0.02 °C. The sensor achieves an accuracy of ±0.2°C within the relevant medical temperature range. The user can choose to configure the digital output to be PWM.

Features and benefits

Factory calibrated in wide temperature range: -20 to 85°C for sensor temperature and -40 to 115°C for object temperature
High accuracy of 0.5°C over wide temperature range (0..+50 C for both Ta and To)
Medical accuracy of 0.2°C in a limited temperature range
Measurement resolution of 0.02°C
SMBus compatible digital interface for fast temperature readings and building sensor networks
Customizable PWM output for continuous reading

3V supply voltage with power saving mode

Connection



Code

This particular example comes from the following library which I installed - https://github.com/skiselev/MLX90615

```
#include <Wire.h>
#include <mlx90615.h>

MLX90615 mlx = MLX90615();

void setup()
{
  Serial.begin(9600);
  Serial.println("Melexis MLX90615 infra-red temperature sensor test");
  mlx.begin();
  Serial.print("Sensor ID number = ");
  Serial.println(mlx.get_id(), HEX);
}

void loop()
{
  Serial.print("Ambient = ");
  Serial.print(mlx.get_ambient_temp());
  Serial.print(" *C\tObject = ");
  Serial.print(mlx.get_object_temp());
  Serial.println(" *C");

  delay(500);
}
```

Output

Open the serial monitor and you should see something like this displayed

Ambient = 24.97 *C Object = 20.95 *C
Ambient = 24.93 *C Object = 20.95 *C

Ambient = 24.93 *C Object = 20.95 *C
Ambient = 24.89 *C Object = 20.85 *C
Ambient = 24.97 *C Object = 51.13 *C
Ambient = 25.49 *C Object = 62.23 *C
Ambient = 26.15 *C Object = 55.59 *C
Ambient = 26.41 *C Object = 54.95 *C

MPL3115A2 absolute pressure sensor example

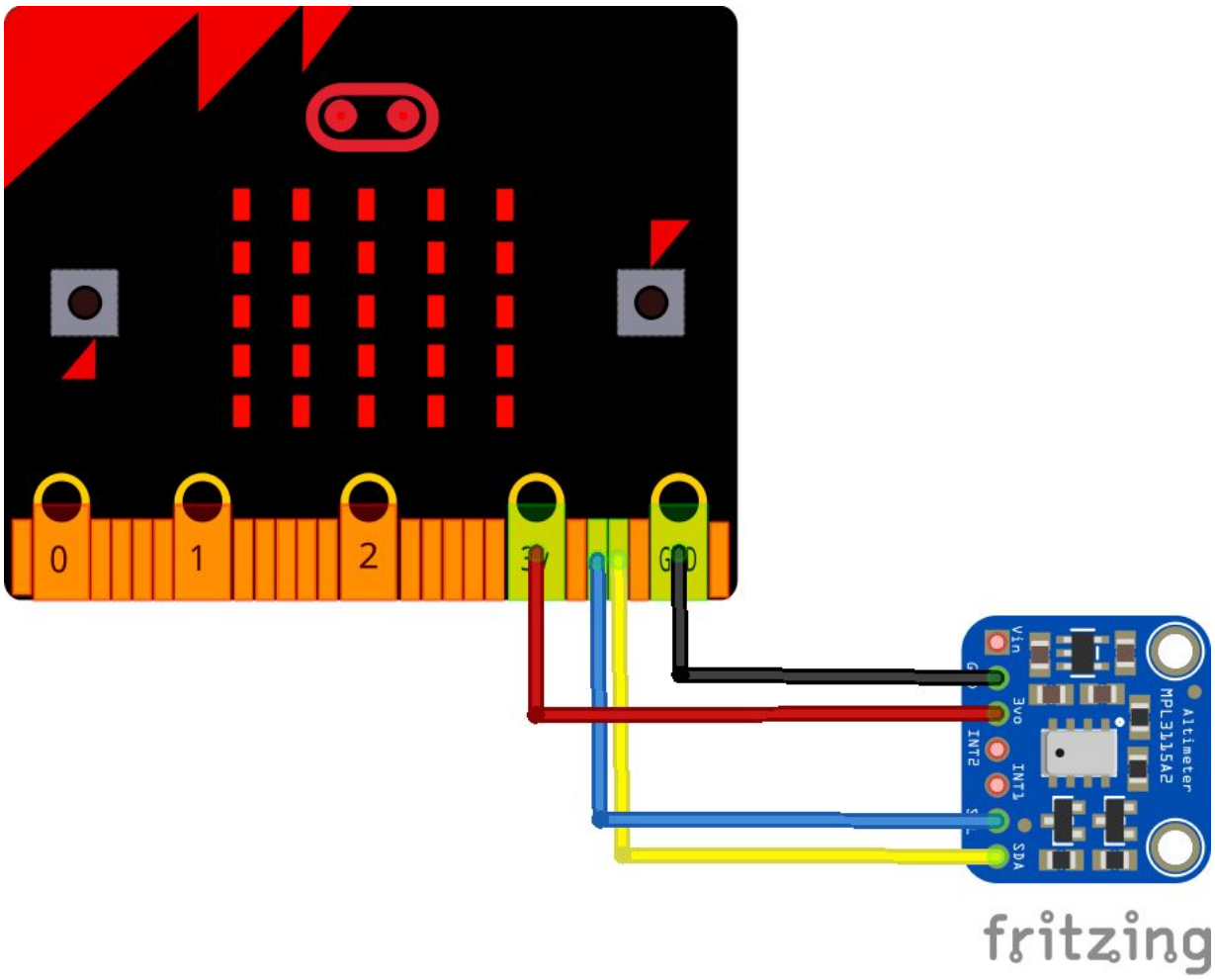
The MPL3115A2 is a compact, piezoresistive, absolute pressure sensor with an I2C digital interface. MPL3115A2 has a wide operating range of 20 kPa to 110 kPa, a range that covers all surface elevations on earth. The MEMS is temperature compensated utilizing an on-chip temperature sensor.

The pressure and temperature data is fed into a high resolution ADC to provide fully compensated and digitized outputs for pressure in Pascals and temperature in °C.

The compensated pressure output can then be converted to altitude, utilizing the formula stated in Section 9.1.3 "Pressure/altitude" provided in meters.

The internal processing in MPL3115A2 removes compensation and unit conversion load from the system MCU, simplifying system design

Schematics/Layout



Code

Again we use a library - https://github.com/adafruit/Adafruit_MPL3115A2_Library

```
#include <Wire.h>
#include <Adafruit_MPL3115A2.h>

// Power by connecting Vin to 3-5V, GND to GND
// Uses I2C - connect SCL to the SCL pin, SDA to SDA pin
// See the Wire tutorial for pinouts for each Arduino
// http://arduino.cc/en/reference/wire
Adafruit_MPL3115A2 baro = Adafruit_MPL3115A2();

void setup()
{
  Serial.begin(9600);
  Serial.println("Adafruit_MPL3115A2 test!");
}

void loop()
{
  if (! baro.begin())
  {
  Serial.println("Couldnt find sensor");
  return;


  float pascals = baro.getPressure();
  // Our weather page presents pressure in Inches (Hg)
  // Use http://www.onlineconversion.com/pressure.htm for other units
  Serial.print(pascals/3377); Serial.println(" Inches (Hg)");

  float altm = baro.getAltitude();
  Serial.print(altm); Serial.println(" meters");
```

```
  float tempC = baro.getTemperature();
  Serial.print(tempC); Serial.println("*C");

  delay(250);
  }
}
```

Output

Open the serial monitor and you should see something like this

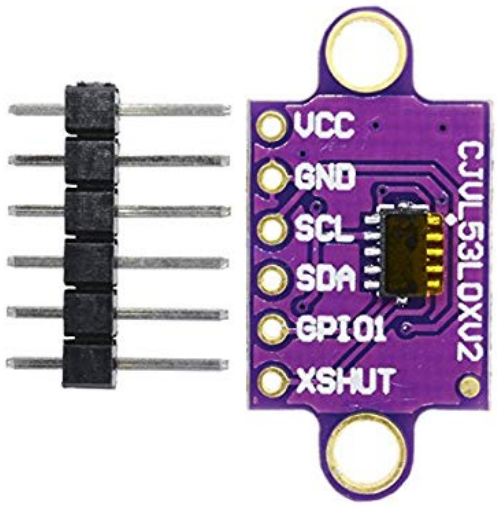
```
Adafruit_MPL3115A2 test!
30.17 Inches (Hg)
-48.25 meters
35.75*C
30.17 Inches (Hg)
-47.81 meters
35.63*C
30.17 Inches (Hg)
-48.00 meters
35.56*C
30.17 Inches (Hg)
-47.81 meters
35.44*C
```

VL53L0X Time-of-Flight sensor and micro:bit

The VL53L0X is a Time-of-Flight (ToF) laser-ranging module housed in the smallest package on the market today, providing accurate distance measurement whatever the target reflectances unlike conventional technologies.

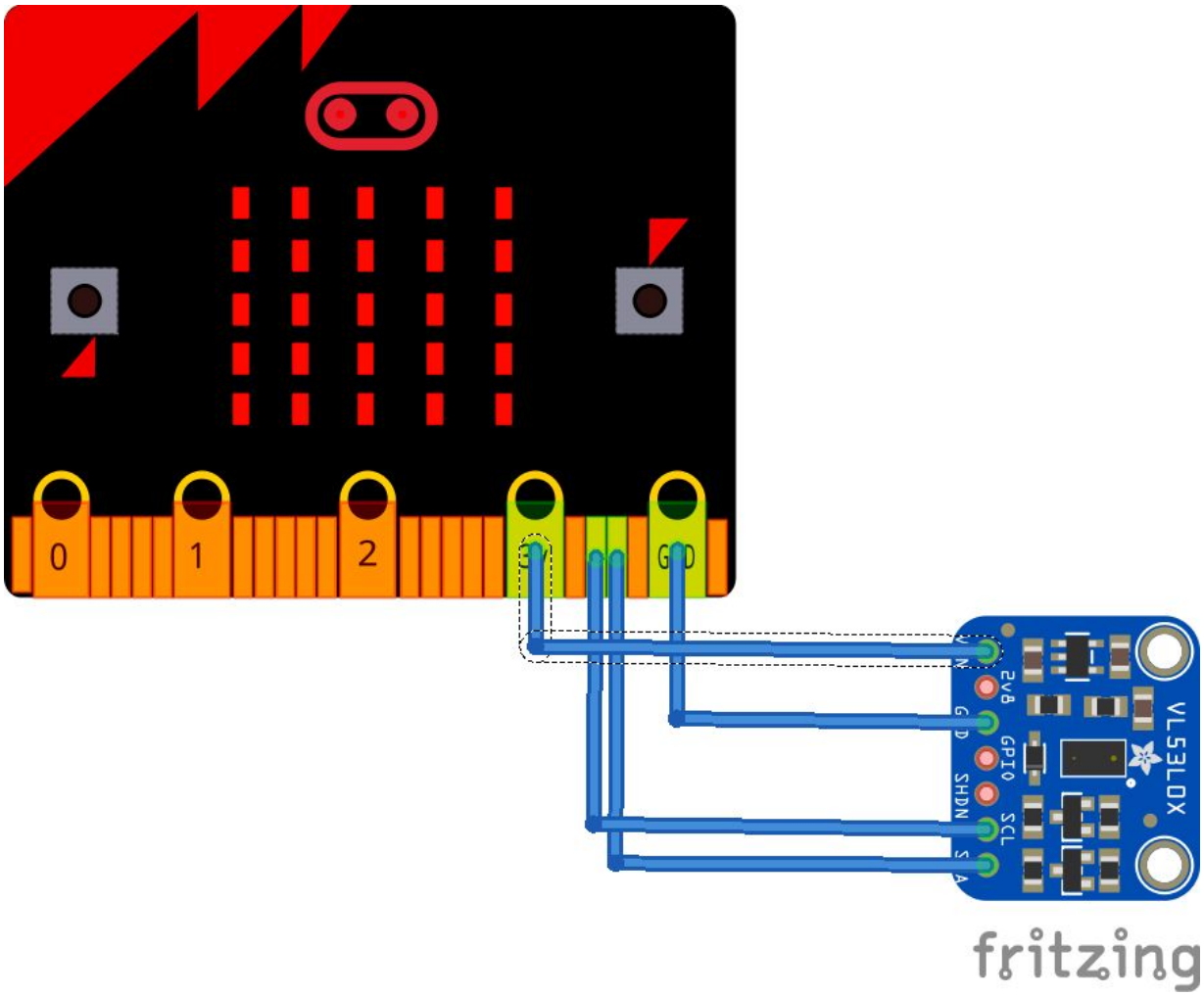
It can measure absolute distances up to 2m, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The VL53L0X integrates a leading-edge SPAD array (Single Photon Avalanche Diodes) and embeds ST's second generation FlightSenseTM patented technology.



The VL53L0X's 940 nm VCSEL emitter (Vertical Cavity Surface-Emitting Laser), is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distances, higher immunity to ambient light, and better robustness to cover glass optical crosstalk.

Schematics/Layout



Code

Again we use a library and again its an adafruit one - https://github.com/adafruit/Adafruit_VL53L0X The default example worked fine, so here it is

```
#include "Adafruit_VL53L0X.h"

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

void setup()
{
 Serial.begin(115200);

 // wait until serial port opens for native USB devices
 while (! Serial)
 {
  delay(1);
 }

 Serial.println("Adafruit VL53L0X test");
 if (!lox.begin())
 {
  Serial.println(F("Failed to boot VL53L0X"));
  while(1);
 }
 // power
 Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}

void loop()
{
 VL53L0X_RangingMeasurementData_t measure;

 Serial.print("Reading a measurement... ");
 lox.rangingTest(&measure, false); // pass in 'true' to get debug data printout!
```

```
  if (measure.RangeStatus != 4)
  { // phase failures have incorrect data
    Serial.print("Distance (mm): "); Serial.println(measure.RangeMilliMeter);
  }
  else
  {
    Serial.println(" out of range ");
  }

  delay(100);
}
```

Output

Open the serial monitor - this is what I saw. I moved an object closer to the sensor hence the readings decreasing

Reading a measurement... Distance (mm): 84
Reading a measurement... Distance (mm): 49
Reading a measurement... Distance (mm): 47
Reading a measurement... Distance (mm): 47
Reading a measurement... Distance (mm): 49
Reading a measurement... Distance (mm): 48
Reading a measurement... Distance (mm): 52
Reading a measurement... Distance (mm): 44

MS5611 barometric pressure sensor example

This barometric pressure sensor is optimized for altimeters and variometers with an altitude resolution of 10 cm. The sensor module includes a high linearity pressure sensor and an ultra-low power 24 bit ΔΣ ADC with internal factory calibrated coefficients. It provides a precise digital 24 Bit pressure and temperature value and different operation modes that allow the user to optimize for conversion speed and current consumption. A high resolution temperature output allows the implementation of an altimeter/thermometer function without any additional sensor. The MS5611-01BA can be interfaced to virtually any microcontroller. The communication protocol is simple, without the need of programming internal registers in the device.

Small dimensions of only 5.0 mm x 3.0 mm and a height of only 1.0 mm allow for integration in mobile devices. This new sensor module generation is based on leading MEMS technology and latest benefits from MEAS Switzerland proven experience and know-how in high volume manufacturing of altimeter modules, which have been widely used for over a decade. The sensing principle employed leads to very low hysteresis and high stability of both pressure and temperature signal.

- High resolution module, 10 cm
- Fast conversion down to 1 ms
- Low power, 1 µA (standby < 0.15 µA)
- QFN package 5.0 x 3.0 x 1.0 mm^3
- Supply voltage 1.8 to 3.6 V

- Integrated digital pressure sensor (24 bit ΔΣ ADC)
- Operating range: 10 to 1200 mbar, -40 to +85 °C
- I^2C and SPI interface up to 20 MHz
- No external components (Internal oscillator)
- Excellent long term stability

Connection

Microbit	**Module connection**
3.3v	Vcc
Gnd	Gnd
SCL - 19	SCL
SDA - 20	SDA

Code

This example comes from the https://github.com/jarzebski/Arduino-MS5611 library

```
#include <Wire.h>
#include <MS5611.h>

MS5611 ms5611;

double referencePressure;

void setup()
{
  Serial.begin(9600);

  // Initialize MS5611 sensor
  Serial.println("Initialize MS5611 Sensor");

  while(!ms5611.begin())
  {
    Serial.println("Could not find a valid MS5611 sensor, check wiring!");
    delay(500);
  }

  // Get reference pressure for relative altitude
  referencePressure = ms5611.readPressure();

  // Check settings
  checkSettings();
}

void checkSettings()
{
  Serial.print("Oversampling: ");
  Serial.println(ms5611.getOversampling());
```

```
}

void loop()
{
  // Read raw values
  uint32_t rawTemp = ms5611.readRawTemperature();
  uint32_t rawPressure = ms5611.readRawPressure();

  // Read true temperature & Pressure
  double realTemperature = ms5611.readTemperature();
  long realPressure = ms5611.readPressure();

  // Calculate altitude
  float absoluteAltitude = ms5611.getAltitude(realPressure);
  float relativeAltitude = ms5611.getAltitude(realPressure, referencePressure);

  Serial.println("--");

  Serial.print(" rawTemp = ");
  Serial.print(rawTemp);
  Serial.print(", realTemp = ");
  Serial.print(realTemperature);
  Serial.println(" *C");

  Serial.print(" rawPressure = ");
  Serial.print(rawPressure);
  Serial.print(", realPressure = ");
  Serial.print(realPressure);
  Serial.println(" Pa");

  Serial.print(" absoluteAltitude = ");
  Serial.print(absoluteAltitude);
  Serial.print(" m, relativeAltitude = ");
  Serial.print(relativeAltitude);
  Serial.println(" m");
```

```
  delay(1000);
}
```

Output

Open the serial monitor and you will see something like this

```
rawTemp = 8432348, realTemp = 24.08 *C
rawPressure = 8669780, realPressure = 101367 Pa
absoluteAltitude = -3.50 m, relativeAltitude = -0.17 m
--
rawTemp = 8476892, realTemp = 25.56 *C
rawPressure = 8654596, realPressure = 101383 Pa
absoluteAltitude = -4.83 m, relativeAltitude = -1.50 m
--
rawTemp = 8553596, realTemp = 28.06 *C
rawPressure = 8628308, realPressure = 101383 Pa
absoluteAltitude = -4.83 m, relativeAltitude = -1.50 m
--
rawTemp = 8595300, realTemp = 29.42 *C
rawPressure = 8613628, realPressure = 101374 Pa
absoluteAltitude = -4.08 m, relativeAltitude = -0.75 m
```

OPT3001 Digital Ambient Light Sensor

The OPT3001 is a sensor that measures the intensity of visible light. The spectral response of the sensor tightly matches the photopic response of the human eye and includes significant infrared rejection.

The OPT3001 is a single-chip lux meter, measuring the intensity of light as visible by the human eye. The precision spectral response and strong IR rejection of the device enables the OPT3001 to accurately meter the intensity of light as seen by the human eye regardless of light source. The strong IR rejection also aids in maintaining high accuracy when industrial design calls for mounting the sensor under dark glass for aesthetics. The OPT3001 is designed for systems that create light-based experiences for humans, and an ideal preferred replacement for photodiodes, photoresistors, or other ambient light sensors with less human eye matching and IR rejection.

Measurements can be made from 0.01 lux up to 83k lux without manually selecting full-scale ranges by using the built-in, full-scale setting feature. This capability allows light measurement over a 23-bit effective dynamic range.

The digital operation is flexible for system integration. Measurements can be either continuous or single-shot. The control and interrupt system features autonomous operation, allowing the processor to sleep while the sensor searches for appropriate wake-up events to report via the interrupt pin. The digital output is reported over an I2C- and SMBus-compatible, two-wire serial interface.

The low power consumption and low power-supply voltage capability of the OPT3001 enhance the battery life of battery-powered systems.

Features

- Precision Optical Filtering to Match Human Eye:
  - Rejects > 99% (typ) of IR
- Automatic Full-Scale Setting Feature Simplifies Software and Ensures Proper Configuration
- Measurements: 0.01 lux to 83 k lux
- 23-Bit Effective Dynamic Range With Automatic Gain Ranging
- 12 Binary-Weighted Full-Scale Range Settings: < 0.2% (typ) Matching Between Ranges
- Low Operating Current: 1.8 μA (typ)
- Operating Temperature Range: –40°C to +85°C
- Wide Power-Supply Range: 1.6 V to 3.6 V
- 5.5-V Tolerant I/O

Connection

Microbit	CJMCU-3001
3.3v	Vcc
Gnd	Gnd
SDA - 20	SDA
SCL - 19	SCL

Code

This example uses the following library
https://github.com/closedcube/ClosedCube_OPT3001_Arduino

```
/*

This is example for ClosedCube OPT3001 Digital Ambient Light Sensor breakout board

Initial Date: 02-Dec-2015

Hardware connections for Arduino Uno:
VDD to 3.3V DC
SDA to A4
SCL to A5
GND to common ground

Written by AA for ClosedCube

MIT License

*/

#include <Wire.h>
#include <ClosedCube_OPT3001.h>

ClosedCube_OPT3001 opt3001;
```

```
#define OPT3001_ADDRESS 0x44

void setup()
{
	Serial.begin(9600);
	Serial.println("ClosedCube OPT3001 Arduino Test");

	opt3001.begin(OPT3001_ADDRESS);
	Serial.print("OPT3001 Manufacturer ID");
	Serial.println(opt3001.readManufacturerID());
	Serial.print("OPT3001 Device ID");
	Serial.println(opt3001.readDeviceID());

	configureSensor();
	printResult("High-Limit", opt3001.readHighLimit());
	printResult("Low-Limit", opt3001.readLowLimit());
	Serial.println("----");
}

void loop()
{
	OPT3001 result = opt3001.readResult();
	printResult("OPT3001", result);
	delay(500);
}

void configureSensor() {
	OPT3001_Config newConfig;

	newConfig.RangeNumber = B1100;
	newConfig.ConvertionTime = B0;
	newConfig.Latch = B1;
	newConfig.ModeOfConversionOperation = B11;

	OPT3001_ErrorCode errorConfig = opt3001.writeConfig(newConfig);
	if (errorConfig != NO_ERROR)
```

```
        printError("OPT3001 configuration", errorConfig);
else {
        OPT3001_Config sensorConfig = opt3001.readConfig();
        Serial.println("OPT3001 Current Config:");
        Serial.println("------------------------------");

        Serial.print("Conversion ready (R):");
        Serial.println(sensorConfig.ConversionReady,HEX);

        Serial.print("Conversion time (R/W):");
        Serial.println(sensorConfig.ConvertionTime, HEX);

        Serial.print("Fault count field (R/W):");
        Serial.println(sensorConfig.FaultCount, HEX);

        Serial.print("Flag high field (R-only):");
        Serial.println(sensorConfig.FlagHigh, HEX);

        Serial.print("Flag low field (R-only):");
        Serial.println(sensorConfig.FlagLow, HEX);

        Serial.print("Latch field (R/W):");
        Serial.println(sensorConfig.Latch, HEX);

        Serial.print("Mask exponent field (R/W):");
        Serial.println(sensorConfig.MaskExponent, HEX);

        Serial.print("Mode of conversion operation (R/W):");
        Serial.println(sensorConfig.ModeOfConversionOperation, HEX);

        Serial.print("Polarity field (R/W):");
        Serial.println(sensorConfig.Polarity, HEX);

        Serial.print("Overflow flag (R-only):");
        Serial.println(sensorConfig.OverflowFlag, HEX);

        Serial.print("Range number (R/W):");
```

```
                Serial.println(sensorConfig.RangeNumber, HEX);

                Serial.println("------------------------------");
        }

}

void printResult(String text, OPT3001 result) {
        if (result.error == NO_ERROR) {
                Serial.print(text);
                Serial.print(": ");
                Serial.print(result.lux);
                Serial.println(" lux");
        }
        else {
                printError(text,result.error);
        }
}

void printError(String text, OPT3001_ErrorCode error) {
        Serial.print(text);
        Serial.print(": [ERROR] Code #");
        Serial.println(error);
}
```

Output

Here is a sample of the output from the Serial Monitor

```
OPT3001: 7910.40 lux
OPT3001: 8017.92 lux
OPT3001: 7969.28 lux
OPT3001: 454.40 lux
OPT3001: 125.28 lux
OPT3001: 192.88 lux
OPT3001: 247.76 lux
OPT3001: 252.56 lux
```

MAG3110 magnetic sensor example

The MAG3110 is a small, low-power digital 3D magnetic sensor with a wide dynamic range to allow operation in PCBs with high extraneous magnetic fields. The MAG3110:

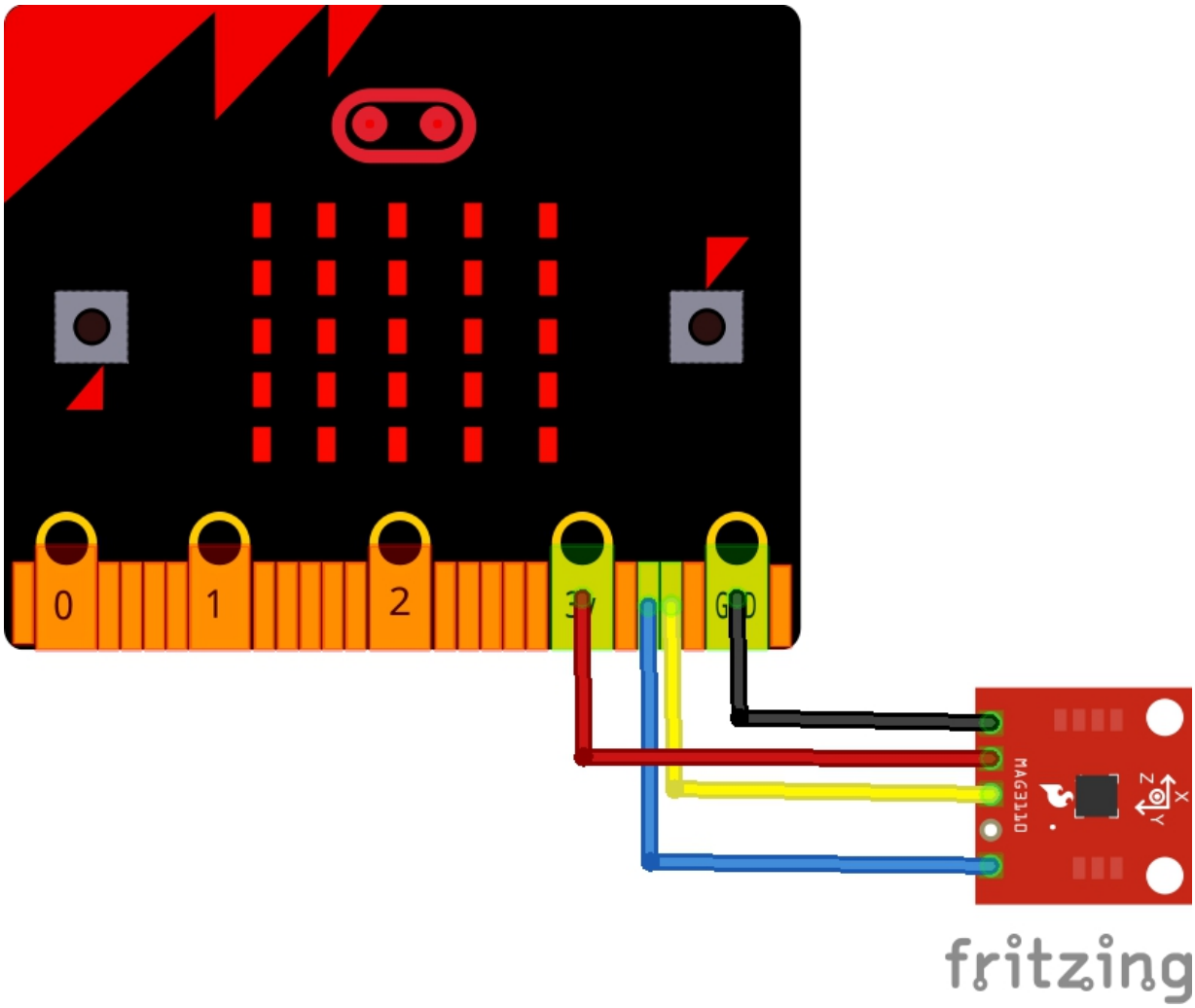
- Measures the components of the local magnetic field, the sum of the geomagnetic field and the magnetic field created by components on the circuit board
- Can be used in conjunction with a 3-axis accelerometer so that orientation-independent accurate compass heading information may be achieved
- Features a standard I²C serial interface and is capable of measuring local magnetic fields up to 10 Gauss with output data rates up to 80 Hz

Features

- 80 Hz maximum sampling rate
- I²C interface 400 kHz

- -40°C to +85°C operation
- 0.95 to 3.6-volt supply
- Full-scale range ±1000 μT
- Low-power, single-shot measurement mode
- Noise down to 0.25 μT rms
- Sensitivity of 0.10 μμT

Layout



Code

```
#include <Wire.h>
#define MAG_ADDR 0x0E //7-bit address for the MAG3110, doesn't change

void setup()
{
  Wire.begin(); // join i2c bus (address optional for master)
  Serial.begin(9600); // start serial for output
  config(); // turn the MAG3110 on
```

```
}

void loop()
{
  print_values();
  delay(500);
}

void config(void)
{
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x11); // cntrl register2
  Wire.write(0x80); // send 0x80, enable auto resets
  Wire.endTransmission(); // stop transmitting
  delay(15);
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x10); // cntrl register1
  Wire.write(1); // send 0x01, active mode
  Wire.endTransmission(); // stop transmitting
}

void print_values(void)
{
  Serial.print("x=");
  Serial.print(readx());
  Serial.print(",");
  Serial.print("y=");
  Serial.print(ready());
  Serial.print(",");
  Serial.print("z=");
  Serial.println(readz());
}

int readx(void)
{
  int xl, xh; //define the MSB and LSB
```

```
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x01); // x MSB reg
  Wire.endTransmission(); // stop transmitting
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
    xh = Wire.read(); // receive the byte
  }
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x02); // x LSB reg
  Wire.endTransmission(); // stop transmitting
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
    xl = Wire.read(); // receive the byte
  }
  int xout = (xl|(xh << 8)); //concatenate the MSB and LSB
  return xout;
}

int ready(void)
{
  int yl, yh; //define the MSB and LSB
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x03); // y MSB reg
  Wire.endTransmission(); // stop transmitting
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
    yh = Wire.read(); // receive the byte
```

```
  }
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x04); // y LSB reg
  Wire.endTransmission(); // stop transmitting
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
    yl = Wire.read(); // receive the byte
  }
  int yout = (yl|(yh << 8)); //concatenate the MSB and LSB
  return yout;
}

int readz(void)
{
  int zl, zh; //define the MSB and LSB

  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x05); // z MSB reg
  Wire.endTransmission(); // stop transmitting

  delayMicroseconds(2); //needs at least 1.3us free time between start and stop

  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
    zh = Wire.read(); // receive the byte
  }

  delayMicroseconds(2); //needs at least 1.3us free time between start and stop

  Wire.beginTransmission(MAG_ADDR); // transmit to device 0x0E
  Wire.write(0x06); // z LSB reg
```

```
  Wire.endTransmission(); // stop transmitting
  delayMicroseconds(2); //needs at least 1.3us free time between start and stop
  Wire.requestFrom(MAG_ADDR, 1); // request 1 byte

  while(Wire.available()) // slave may send less than requested
  {
   zl = Wire.read(); // receive the byte
  }
  int zout = (zl|(zh << 8)); //concatenate the MSB and LSB
  return zout;
}
```

Output

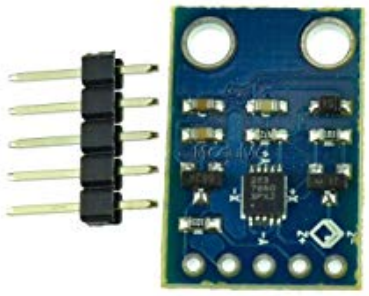
Open the serial monitor window

x=126,y=1552,z=72
x=256,y=1552,z=19
x=259,y=15,z=67
x=124,y=1544,z=67
x=128,y=1545,z=64
x=124,y=1544,z=68
x=0,y=1953,z=513
x=50,y=2082,z=88

MMA7660 accelerometer example

The MMA7660FC is a digital output I²C, very low-power, low-profile capacitive micro-machined accelerometer featuring a low pass filter, compensation for zero-g offset and gain errors and conversion to six-bit digital values at a user configurable output data rate. The device can be used for sensor data changes, product orientation and gesture detection through an interrupt pin (INT).

Communication is handled through a 2 pin I2C interface, available on a wide range of microcontrollers. The I2C address by default is 0x4c.



Features

- Digital output I²C
- 3 mm x 3 mm x 0.9 mm DFN package
- Low-power current consumption
  - Off mode: 0.4 µA
  - Standby mode: 2 µA
  - Active mode: Configurable down to 47 µA

- Low-voltage operation: 2.4 – 3.6-volts
- 3-axis ±1.5 g MEMS sensor and CMOS interface controller built into one package
- Configurable output data rate from one to 120 samples a second
- Auto wake/sleep feature for low-power consumption
- Tilt orientation detection for portrait/landscape capability
- Gesture detection including shake and pulse detection
- Robust design, high shock survivability (10,000g)

Connection

Module connection to Microbit

- GND = GND
- VCC = 3v3
- SDA = 20
- SCL = 21

Code

I used this library - https://github.com/mcauser/Grove-3Axis-Digital-Accelerometer-1.5g-MMA7660FC This is the default example

```
#include <Wire.h>
#include "MMA7660.h"
MMA7660 acc;
void setup()
{
acc.init();
Serial.begin(115200);
}
void loop()
{
static long cnt = 0;
```

```
static long cntout = 0;
float ax,ay,az;
int8_t x, y, z;
acc.getXYZ(&x,&y,&z);
if(acc.getAcceleration(&ax,&ay,&az))
{
Serial.print("got data ok: ");
}
else
{
Serial.print("timed out: ");
}
Serial.println("acceleration of X/Y/Z: ");
Serial.print(ax);
Serial.println(" g");
Serial.print(ay);
Serial.println(" g");
Serial.print(az);
Serial.println(" g");
Serial.println();
delay(500);
}
```

Output

Open the serial monitor

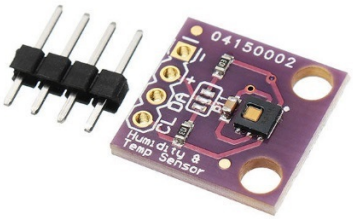
got data ok: acceleration of X/Y/Z:
2.76 g
0.19 g
2.33 g

got data ok: acceleration of X/Y/Z:
1.48 g
1.48 g
1.52 g

got data ok: acceleration of X/Y/Z:
2.62 g
0.00 g
0.48 g

HDC1080 humidity and temperature sensor example

The HDC1080 is a digital humidity sensor with integrated temperature sensor that provides excellent measurement accuracy at very low power. The HDC1080 operates over a wide supply range, and is a low cost, low power alternative to competitive solutions in a wide range of common applications. The humidity and temperature sensors are factory calibrated.



Features

Relative Humidity Accuracy ±2% (typical)
Temperature Accuracy ±0.2°C (typical)
Excellent Stability at High Humidity
14 Bit Measurement Resolution
100 nA Sleep Mode Current

Connection

Microbit connection	**Module connection**
3v3	3v3
GND	GND
SDA - 20	SDA
SCL - 21	SCL

Code

I needed to modify the https://github.com/closedcube/ClosedCube_HDC1080_Arduino library, otherwise there was compilation errors

You need to change Wire.write(0x00); to Wire.write((byte)0x00); - http://www.microbitlearning.com/wp-content/uploads/2018/05/ClosedCube_HDC1080_Arduino-master.zip

This is the default example

```
#include <Wire.h>
#include "ClosedCube_HDC1080.h"

ClosedCube_HDC1080 hdc1080;

void setup()
{
Serial.begin(9600);
Serial.println("ClosedCube HDC1080 Arduino Test");

// Default settings:
// - Heater off
// - 14 bit Temperature and Humidity Measurement Resolutions
hdc1080.begin(0x40);

Serial.print("Manufacturer ID=0x");
Serial.println(hdc1080.readManufacturerId(), HEX); // 0x5449 ID of Texas Instruments
Serial.print("Device ID=0x");
Serial.println(hdc1080.readDeviceId(), HEX); // 0x1050 ID of the device

printSerialNumber();

}
```

```
void loop()
{
Serial.print("T=");
Serial.print(hdc1080.readTemperature());
Serial.print("C, RH=");
Serial.print(hdc1080.readHumidity());
Serial.println("%");
delay(3000);
}

void printSerialNumber() {
Serial.print("Device Serial Number=");
HDC1080_SerialNumber sernum = hdc1080.readSerialNumber();
char format[12];
sprintf(format, "%02X-%04X-%04X", sernum.serialFirst, sernum.serialMid, sernum.serialLast);
Serial.println(format);
}
```

Output

Open the serial monitor window and you should expect to see something like this

ClosedCube HDC1080 Arduino Test
Manufacturer ID=0x54T=23.06C, RH=51.07%
T=23.06C, RH=51.07%
T=23.08C, RH=51.05%
T=23.08C, RH=51.15%
T=23.10C, RH=51.15%
T=23.11C, RH=51.15%
T=23.11C, RH=51.05%

http://www.microbitlearning.com/wp-content/uploads/2018/05/ClosedCube_HDC1080_Arduino-master.zip

micro:bit and OLED display example

This example uses an OLED display these typically come in a couple of different sizes 128×32 and 128×64, this particular example will use the I2C connection from the Micro:bit to the display. There are a couple of libraries that make life easier.

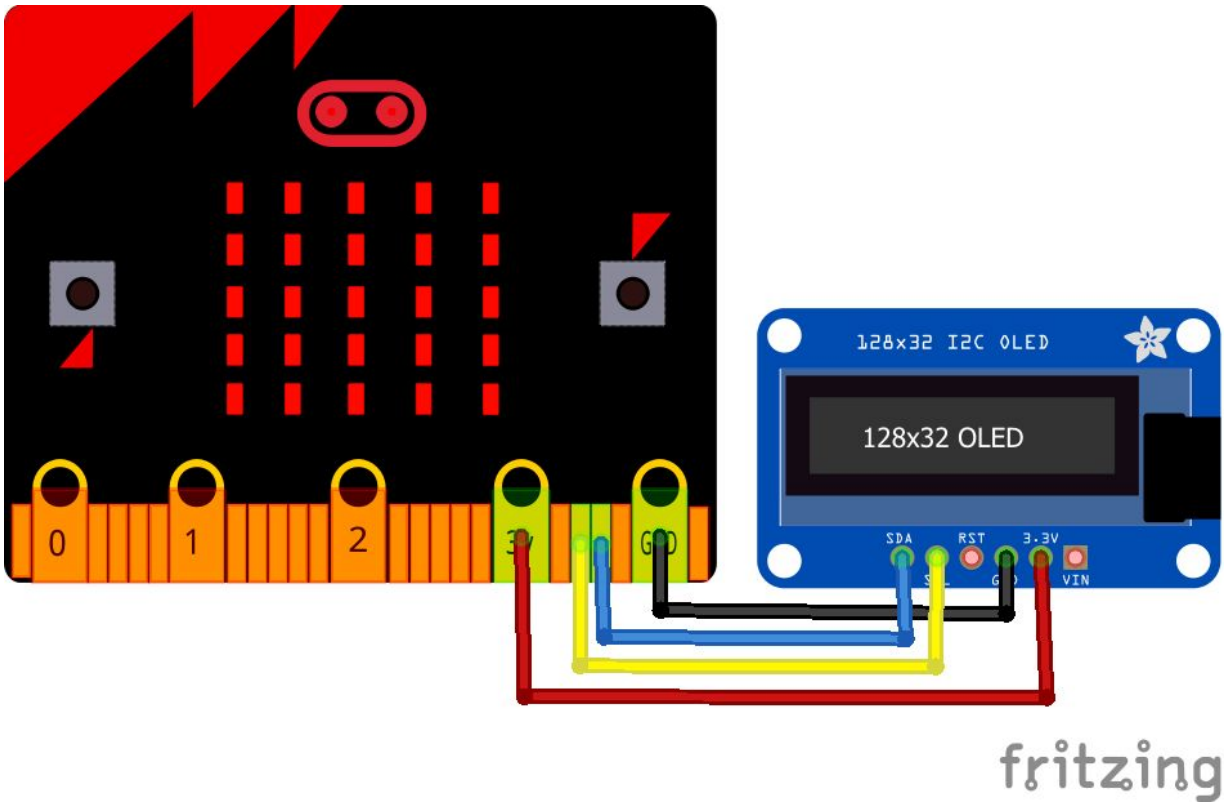
Lets look at a typical oled display

These will come in useful for various projects for example displaying the date and time or maybe temperature readings from a sensor

Connection

Pin Label	Microbit PIN	I²C Function	Notes
GND	Ground	Ground	0V
VCC	Power	Power	Regulated 5V supply.
SDA	SDA / P20	SDA	Serial data in
SCL	SCL / P19	SCL	I²C clock

This layout shows a 128×32 connected to the Micro:bit, 128×64 I2C devices would be the same



Code

This example uses the https://github.com/adafruit/Adafruit_SSD1306/archive/master.zip and https://github.com/adafruit/Adafruit-GFX-Library/archive/master.zip , there are several built in examples. I have modified one just to display text as further examples will write text to a display

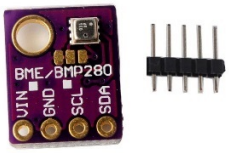
```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4

Adafruit_SSD1306 display(OLED_RESET);

void setup()
{
 Serial.begin(9600);
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C (for the 128x32)  // init done
 display.clearDisplay();  // text display tests
 display.setTextSize(1);
 display.setTextColor(WHITE);
 display.setCursor(0,0);
 display.println("Hello, world!");
 display.setTextColor(BLACK, WHITE); // &#039;inverted&#039; text
 display.println(3.141592);
 display.setTextSize(2);
 display.setTextColor(WHITE);
 display.print("0x");
 display.println(0xDEADBEEF, HEX);
 display.display();
 display.clearDisplay();
}
```

```
void loop()
{
}
```

bme280 environmental sensor example

The BME280 is an integrated environmental sensor developed specifically for mobile applications where size and low power consumption are key design constraints. The unit combines individual high linearity, high accuracy sensors for pressure, humidity and temperature in an 8-pin metal-lid 2.5 x 2.5 x 0.93 mm^3 LGA package, designed for low current consumption (3.6 µA @1Hz), long term stability and high EMC robustness.

The humidity sensor features an extremely fast response time which supports performance requirements for emerging applications such as context awareness, and high accuracy over a wide temperature range. The pressure sensor is an absolute barometric pressure sensor with features exceptionally high accuracy and resolution at very low noise. The integrated temperature sensor has been optimized for very low noise and high resolution. It is primarily used for temperature compensation of the pressure and humidity sensors, and can also be used for estimating ambient temperature.



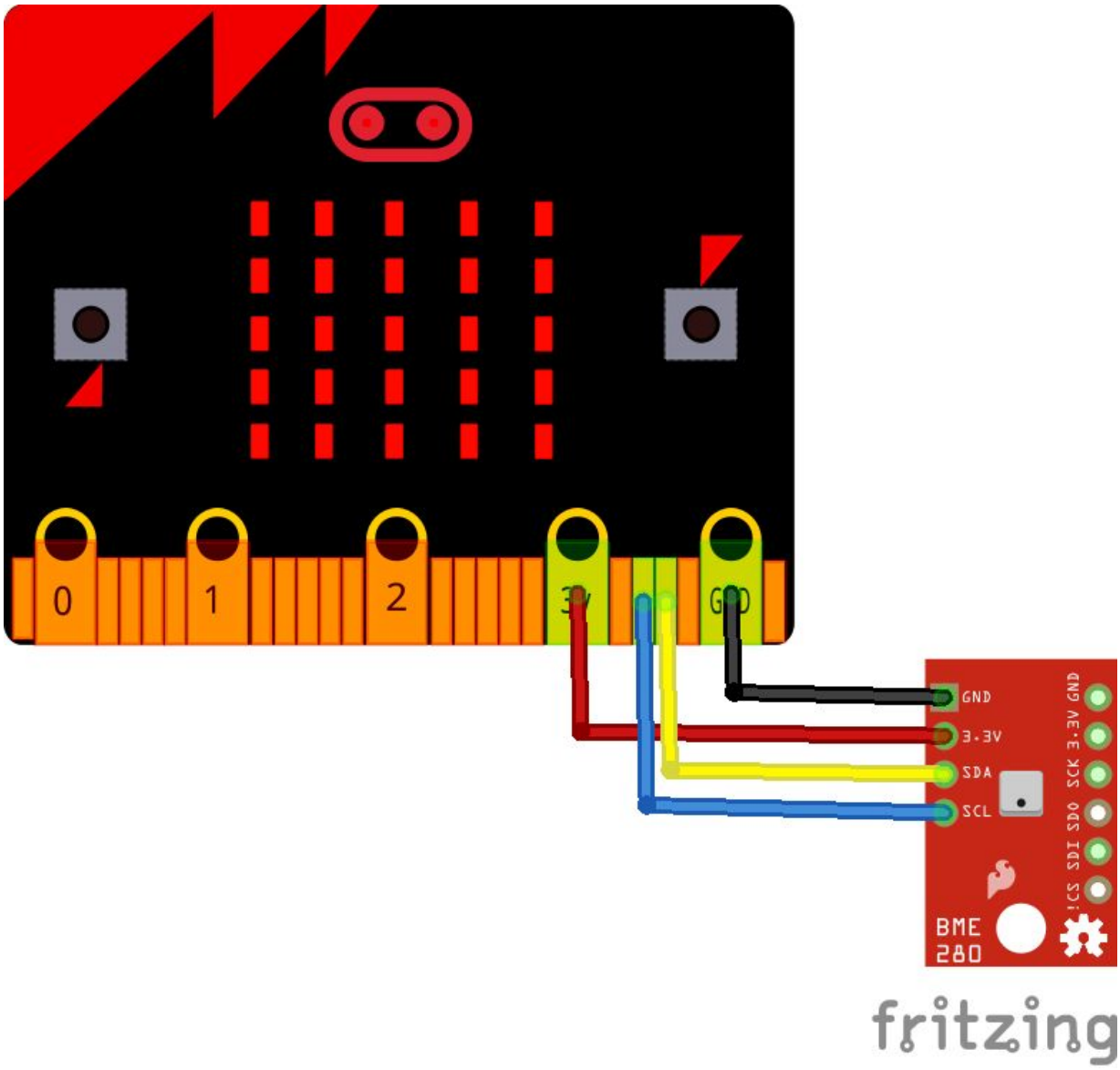
The BME280 supports a full suite of operating modes which provides the flexibility to optimize the device for power

consumption, resolution and filter performance."

Applications

- Context awareness, e.g. skin detection, room change detection
- Fitness monitoring / well-being
- Warning regarding dryness or high temperatures
- Measurement of volume and air flow
- Home automation control
- Control heating, ventilation, air conditioning (HVAC)
- Internet of things
- GPS enhancement (e.g. time-to-first-fix improvement, dead reckoning, slope detection)
- Indoor navigation (change of floor detection, elevator detection)
- Outdoor navigation, leisure and sports applications
- Weather forecast
- Vertical velocity indication (rise/sink speed)

Connection



Code:

No libraries required

```
// Distributed with a free-will license.
// Use it any way you want, profit or free, provided it fits in the licenses of its associated works.
// BME280
// This code is designed to work with the BME280_I2CS I2C Mini Module available from
```

ControlEverything.com.
// https://www.controleverything.com/content/Humidity?sku=BME280_I2CS#tabs-0-product_tabset-2

```
#include<Wire.h>

// BME280 I2C address is 0x76(108)
#define Addr 0x76

void setup()
{
// Initialise I2C communication as MASTER
Wire.begin();
// Initialise Serial communication, set baud rate = 9600
Serial.begin(9600);
}

void loop()
{
unsigned int b1[24];
unsigned int data[8];
unsigned int dig_H1 = 0;
for(int i = 0; i < 24; i++)
{
// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select data register
Wire.write((136+i));
// Stop I2C Transmission
Wire.endTransmission();

// Request 1 byte of data
Wire.requestFrom(Addr, 1);

// Read 24 bytes of data
if(Wire.available() == 1)
{
```

```
b1[i] = Wire.read();
}
}

// Convert the data
// temp coefficients
unsigned int dig_T1 = (b1[0] & 0xff) + ((b1[1] & 0xff) * 256);
int dig_T2 = b1[2] + (b1[3] * 256);
int dig_T3 = b1[4] + (b1[5] * 256);

// pressure coefficients
unsigned int dig_P1 = (b1[6] & 0xff) + ((b1[7] & 0xff ) * 256);
int dig_P2 = b1[8] + (b1[9] * 256);
int dig_P3 = b1[10] + (b1[11] * 256);
int dig_P4 = b1[12] + (b1[13] * 256);
int dig_P5 = b1[14] + (b1[15] * 256);
int dig_P6 = b1[16] + (b1[17] * 256);
int dig_P7 = b1[18] + (b1[19] * 256);
int dig_P8 = b1[20] + (b1[21] * 256);
int dig_P9 = b1[22] + (b1[23] * 256);

// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select data register
Wire.write(161);
// Stop I2C Transmission
Wire.endTransmission();

// Request 1 byte of data
Wire.requestFrom(Addr, 1);

// Read 1 byte of data
if(Wire.available() == 1)
{
dig_H1 = Wire.read();
}
```

```
for(int i = 0; i < 7; i++)
{
// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select data register
Wire.write((225+i));
// Stop I2C Transmission
Wire.endTransmission();

// Request 1 byte of data
Wire.requestFrom(Addr, 1);

// Read 7 bytes of data
if(Wire.available() == 1)
{
b1[i] = Wire.read();
}
}

// Convert the data
// humidity coefficients
int dig_H2 = b1[0] + (b1[1] * 256);
unsigned int dig_H3 = b1[2] & 0xFF ;
int dig_H4 = (b1[3] * 16) + (b1[4] & 0xF);
int dig_H5 = (b1[4] / 16) + (b1[5] * 16);
int dig_H6 = b1[6];

// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select control humidity register
Wire.write(0xF2);
// Humidity over sampling rate = 1
Wire.write(0x01);
// Stop I2C Transmission
Wire.endTransmission();

// Start I2C Transmission
```

```
Wire.beginTransmission(Addr);
// Select control measurement register
Wire.write(0xF4);
// Normal mode, temp and pressure over sampling rate = 1
Wire.write(0x27);
// Stop I2C Transmission
Wire.endTransmission();

// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select config register
Wire.write(0xF5);
// Stand_by time = 1000ms
Wire.write(0xA0);
// Stop I2C Transmission
Wire.endTransmission();

for(int i = 0; i < 8; i++)
{
// Start I2C Transmission
Wire.beginTransmission(Addr);
// Select data register
Wire.write((247+i));
// Stop I2C Transmission
Wire.endTransmission();

// Request 1 byte of data
Wire.requestFrom(Addr, 1);

// Read 8 bytes of data
if(Wire.available() == 1)
{
data[i] = Wire.read();
}
}

// Convert pressure and temperature data to 19-bits
```

```
long adc_p = (((long)(data[0] & 0xFF) * 65536) + ((long)(data[1] & 0xFF) * 256) + (long)(data[2] & 0xF0)) / 16;
long adc_t = (((long)(data[3] & 0xFF) * 65536) + ((long)(data[4] & 0xFF) * 256) + (long)(data[5] & 0xF0)) / 16;
// Convert the humidity data
long adc_h = ((long)(data[6] & 0xFF) * 256 + (long)(data[7] & 0xFF));

// Temperature offset calculations
double var1 = (((double)adc_t) / 16384.0 - ((double)dig_T1) / 1024.0) * ((double)dig_T2);
double var2 = ((((double)adc_t) / 131072.0 - ((double)dig_T1) / 8192.0) *
(((double)adc_t)/131072.0 - ((double)dig_T1)/8192.0)) * ((double)dig_T3);
double t_fine = (long)(var1 + var2);
double cTemp = (var1 + var2) / 5120.0;
double fTemp = cTemp * 1.8 + 32;

// Pressure offset calculations
var1 = ((double)t_fine / 2.0) - 64000.0;
var2 = var1 * var1 * ((double)dig_P6) / 32768.0;
var2 = var2 + var1 * ((double)dig_P5) * 2.0;
var2 = (var2 / 4.0) + (((double)dig_P4) * 65536.0);
var1 = (((double) dig_P3) * var1 * var1 / 524288.0 + ((double) dig_P2) * var1) / 524288.0;
var1 = (1.0 + var1 / 32768.0) * ((double)dig_P1);
double p = 1048576.0 - (double)adc_p;
p = (p - (var2 / 4096.0)) * 6250.0 / var1;
var1 = ((double) dig_P9) * p * p / 2147483648.0;
var2 = p * ((double) dig_P8) / 32768.0;
double pressure = (p + (var1 + var2 + ((double)dig_P7)) / 16.0) / 100;

// Humidity offset calculations
double var_H = (((double)t_fine) - 76800.0);
var_H = (adc_h - (dig_H4 * 64.0 + dig_H5 / 16384.0 * var_H)) * (dig_H2 / 65536.0 * (1.0 + dig_H6 / 67108864.0 * var_H * (1.0 + dig_H3 / 67108864.0 * var_H)));
double humidity = var_H * (1.0 - dig_H1 * var_H / 524288.0);
if(humidity > 100.0)
{
humidity = 100.0;
}
```

```
else if(humidity < 0.0)
{
humidity = 0.0;
}

// Output data to serial monitor
Serial.print("Temperature in Celsius : ");
Serial.print(cTemp);
Serial.println(" C");
Serial.print("Temperature in Fahrenheit : ");
Serial.print(fTemp);
Serial.println(" F");
Serial.print("Pressure : ");
Serial.print(pressure);
Serial.println(" hPa");
Serial.print("Relative Humidity : ");
Serial.print(humidity);
Serial.println(" RH");
delay(1000);
}
```

Output

In the serial monitor

```
Temperature in Celsius : 34.20 C
Temperature in Fahrenheit : 93.56 F
Pressure : 903.94 hPa
Relative Humidity : 0.00 RH
Temperature in Celsius : 34.51 C
Temperature in Fahrenheit : 94.11 F
Pressure : 893.48 hPa
```

MICRO:BIT and I2C LCD example

In this example we will interface to an I2C LCD using our MICRO:BIT. Now these I2C LCD's consist of 2 parts usually an HD44780 16×2 LCD and an I2C backpack which connects to the LCD exposing the standard power and I2C pins.

This is a typical module you can buy, you can see the backpack which is obviously on the back of the LCD

Layout

Connect the pins as follows:

MICRO:BIT	**LCD1602**
GND	GND
3v3	VCC

SDA/21	SDA
SCL/22	SCL

Code

You will need an updated I2C LCD library, the original one I couldn't get to work but this one does seem to work – http://www.esp32learning.com/wp-content/uploads/2017/12/LiquidCrystal_I2C-master.zip

You will need to import this into the IDE as usual

Now my example below required the I2C address to be changed to 0x3F, a lot of the example I have looked at are 0x27

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2); // set the LCD address to 0x3F for a 16 chars and 2 line display

void setup()
{
  lcd.init(); // initialize the lcd
// Print a message to the LCD.
lcd.backlight();
lcd.setCursor(0,0);
lcd.print("Hello world");
lcd.setCursor(1,1);
lcd.print("MICROBIT");
}

void loop()
{

}
```

All going well you should see the messages in the code on your LCD display

micro:bit and TM1637 7 segment display example

A common display module that you can buy on the internet contain the Tm1638 driver chip, I was interested in this one which is the TM1637 which appears to be a more basic version which can only control a display, the TM1638 can also control LED's, buttons and two displays at the same time.

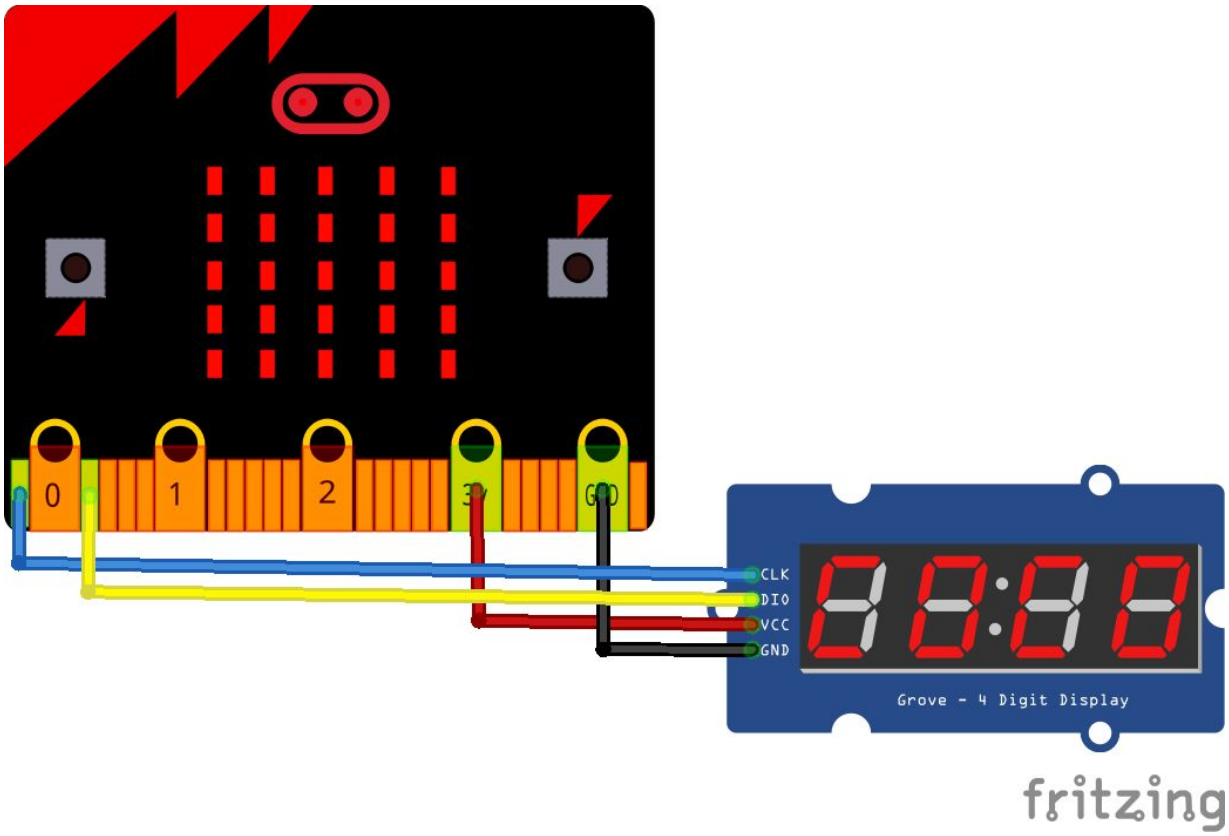
This is a common anode 4-digit tube display module which uses the TM1637 driver chip; Only 2 connections are required to control the 4-digit 8-segment displays

Here is the module

Features of the module

Display common anode for the four red LED
Powered supply by 3.3V/5V
Four common anode tube display module is driven by IC TM1637
Can be used for Arduino devices, two signal lines can make the MCU control 4 8 digital tube. Digital tube 8 segment is adjustable
Here is how to hook the module up, the good news is this worked with my Micro:bit and 3.3v

Schematic



Code

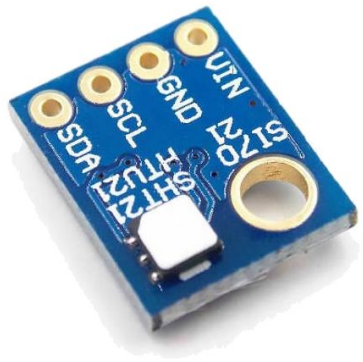
There is a library for this IC, you can get it from https://github.com/avishorp/TM1637 , as usual there is a built in example but here is a simple sketch

```
#include <TM1637Display.h>

const int CLK = 3; //Set the CLK pin connection to the display
const int DIO = 4; //Set the DIO pin connection to the display

int numCounter = 0;

TM1637Display display(CLK, DIO); //set up the 4-Digit Display.

void setup()
{
display.setBrightness(0x0a); //set the diplay to maximum brightness
}

void loop()
{
for(numCounter = 0; numCounter < 1000; numCounter++) //Iterate numCounter
{
display.showNumberDec(numCounter); //Display the numCounter value;
delay(1000);
}
}
```

Si7021 I2C Humidity and Temperature Sensor example

The Si7021 I2C Humidity and Temperature Sensor is a monolithic CMOS IC integrating humidity and temperature sensor elements, an analog-to-digital converter, signal processing, calibration data, and an I2C Interface.

The patented use of industry-standard, low-K polymeric dielectrics for sensing humidity enables the construction of low-power, monolithic CMOS Sensor ICs with low drift and hysteresis, and excellent long term stability, it would be a great sensor to have on the roller. It would be able to measure everything before I go out for a ride around town.

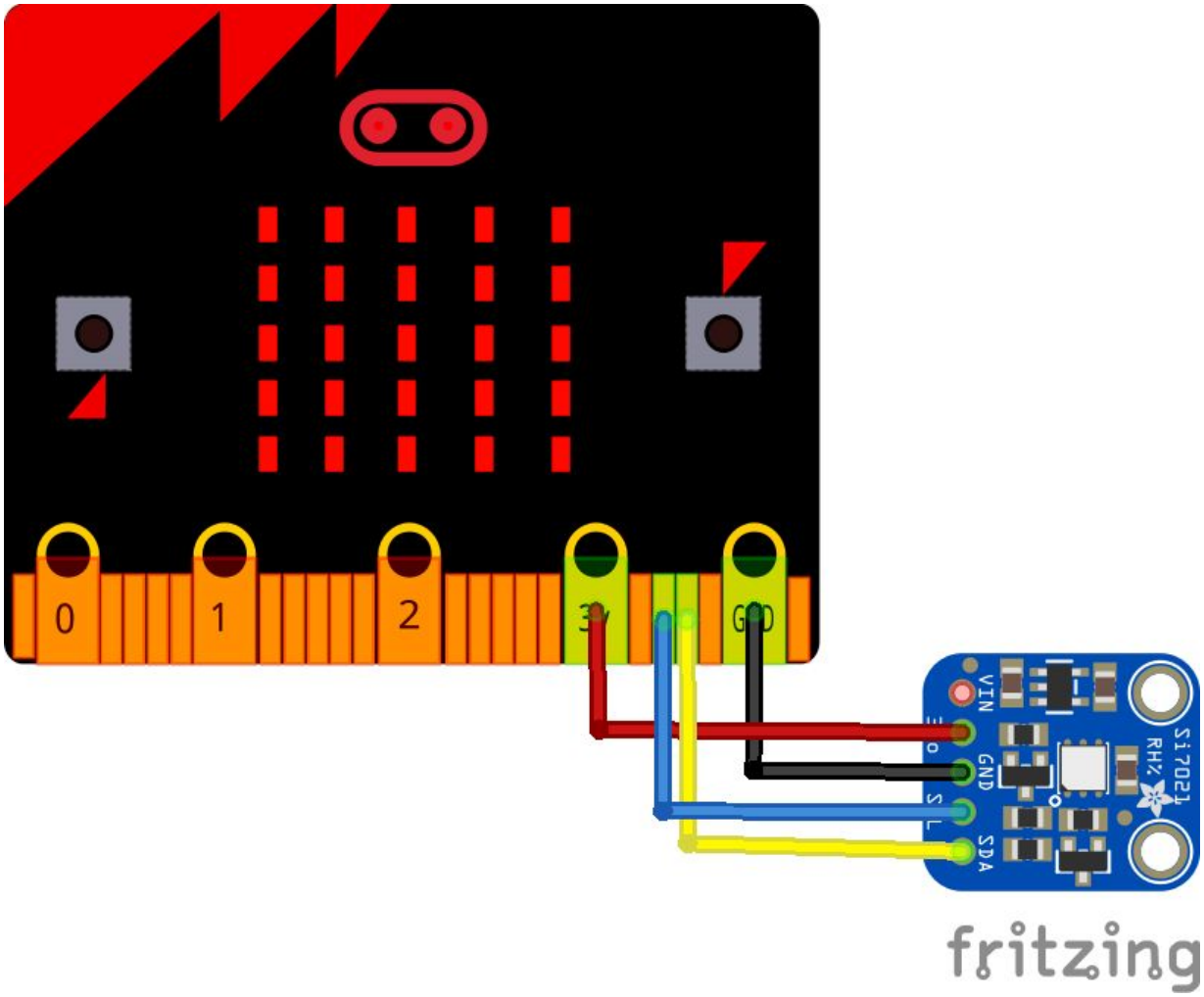


Features
Relative Humidity Sensor:
Si7013/21: ± 3% RH (maximum) @ 0-80% RH
Si7020: ± 4% RH (maximum) @ 0-80% RH
Si7006: ± 5% RH (maximum) @ 0-80% RH

Temperature Sensor:
Si7013/20/21: ±0.4°C accuracy (maximum) @ -10 to +85°C
Si7006: ±1.0°C accuracy (maximum) @ -10 to +85°C
0 to 100% RH operating range
Up to -40 to +125°C operating range
Wide operating voltage range (1.9 to 3.6V)
Low Power Consumption: 2.2µW average power at 3.3V and 1 sample per

second
I2C host interface
Integrated on-chip heater
3mm x 3mm QFN package
Excellent long term stability
Factory calibrated
Optional factory-installed filter/cover

Connection



Code

```
#include <Wire.h>
const int ADDR =0x40;
int X0,X1,Y0,Y1,Y2,Y3;
double X,Y,X_out,Y_out1,Y_out2;

void setup()
{
Serial.begin(9600);
Wire.begin();
delay(100);
Wire.beginTransmission(ADDR);
Wire.endTransmission();
}

void loop()
{
/**Send command of initiating temperature measurement**/
Wire.beginTransmission(ADDR);
Wire.write(0xE3);
Wire.endTransmission();
Serial.print("Temp");
Serial.print("\t");
Serial.println("Humidity");
/**Read data of temperature**/
Wire.requestFrom(ADDR,2);
if(Wire.available()<=2);
{
X0 = Wire.read();
X1 = Wire.read();
X0 = X0<<8;
X_out = X0+X1;
}
/**Calculate and display temperature**/
X=(175.72*X_out)/65536;
```

```
X=X-46.85;
Serial.print(X);
Serial.print("C");
Serial.print("\t");
/**Send command of initiating relative humidity measurement**/
Wire.beginTransmission(ADDR);
Wire.write(0xE5);
Wire.endTransmission();
/**Read data of relative humidity**/
Wire.requestFrom(ADDR,2);
if(Wire.available()<=2);
{
Y0 = Wire.read();
Y2=Y0/100;
Y0=Y0%100;
Y1 = Wire.read();
Y_out1 = Y2*25600;
Y_out2 = Y0*256+Y1;
}
/**Calculate and display relative humidity**/
Y_out1 = (125*Y_out1)/65536;
Y_out2 = (125*Y_out2)/65536;
Y = Y_out1+Y_out2;
Y=Y-6;
Serial.print(Y);
Serial.println("%");
delay(300);
Serial.println();
delay(1000);
}
```

Output

Open the serial monitor, you should see something like this

```
Temp    Humidity
23.12C    52.83%
```

Temp Humidity
24.04C 53.13%

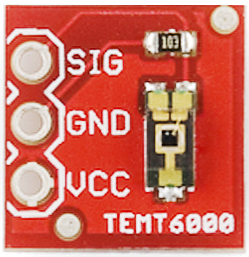
Temp Humidity
26.28C 53.83%

Temp Humidity
27.42C 54.57%

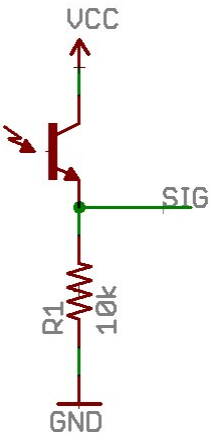
TEMT6000 ambient light sensor

TEMT6000X01 ambient light sensor is a silicon NPN epitaxial planar phototransistor in a miniature transparent 1206 package for surface mounting. It is sensitive to visible light much like the human eye and has peak sensitivity at 570 nm.

Here is a picture of a module



Here is a schematic of the module

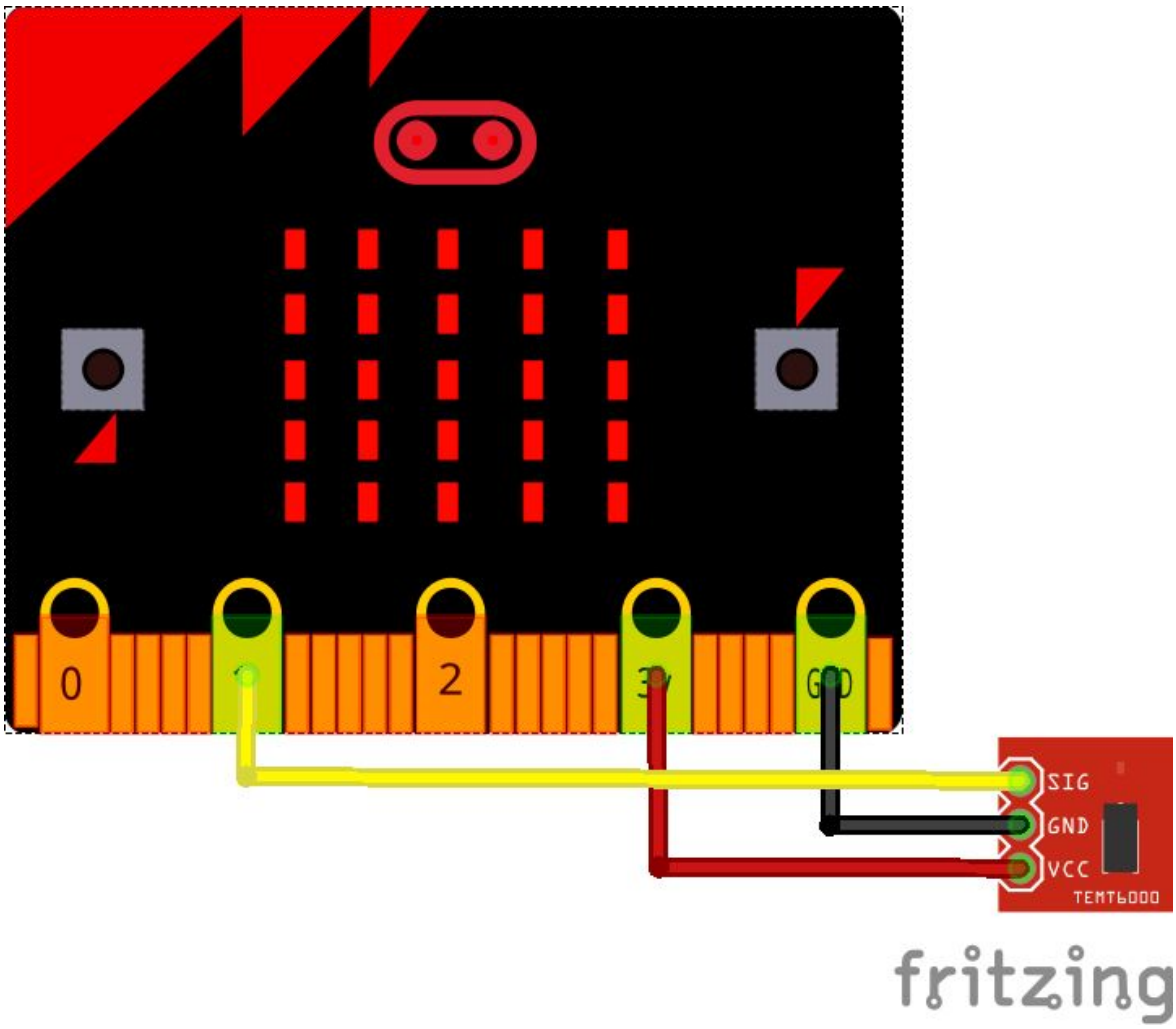


APPLICATIONS

Ambient light sensor for control of display backlight dimming in LCD displays and keypad backlighting of mobile devices and in industrial on/off-lighting operation.

• Automotive sensors
• Mobile phones
• Notebook computers
• PDA’s
• Cameras
• Dashboards

Connection



Code

```
#define LIGHTSENSORPIN A1 //Ambient light sensor reading

void setup()
```

```
{
pinMode(LIGHTSENSORPIN, INPUT);
Serial.begin(9600);
}

void loop()
{
float reading = analogRead(LIGHTSENSORPIN); //Read light level
float square_ratio = reading / 1023.0; //Get percent of maximum value (1023)
square_ratio = pow(square_ratio, 2.0);
Serial.println(reading);
delay(1000);
}
```

Output

Open the serial monitor and you should see something like this

```
41.00
42.00
4.00
1.00
21.00
38.00
41.00
41.00
```

Microbit and GUVA-S12SD UV Sensor

The GUVA-S12SD UV Sensor chip is suitable for detecting the UV radiation in sunlight. It can be used in any application where you want monitor for the amount of UV light and is simple to connect to any microcontroller. I recently noticed that some sellers had little modules for this sensor at a reasonable price so decided to purchase one

The module, with a typical UV detection wavelength of 200 – 370nm, outputs a calibrated analog voltage which varies with the UV light intensity so basically all you need to do is connect this to an ADC input and read in the value.

This value ties in with the UV index, this looks something like this

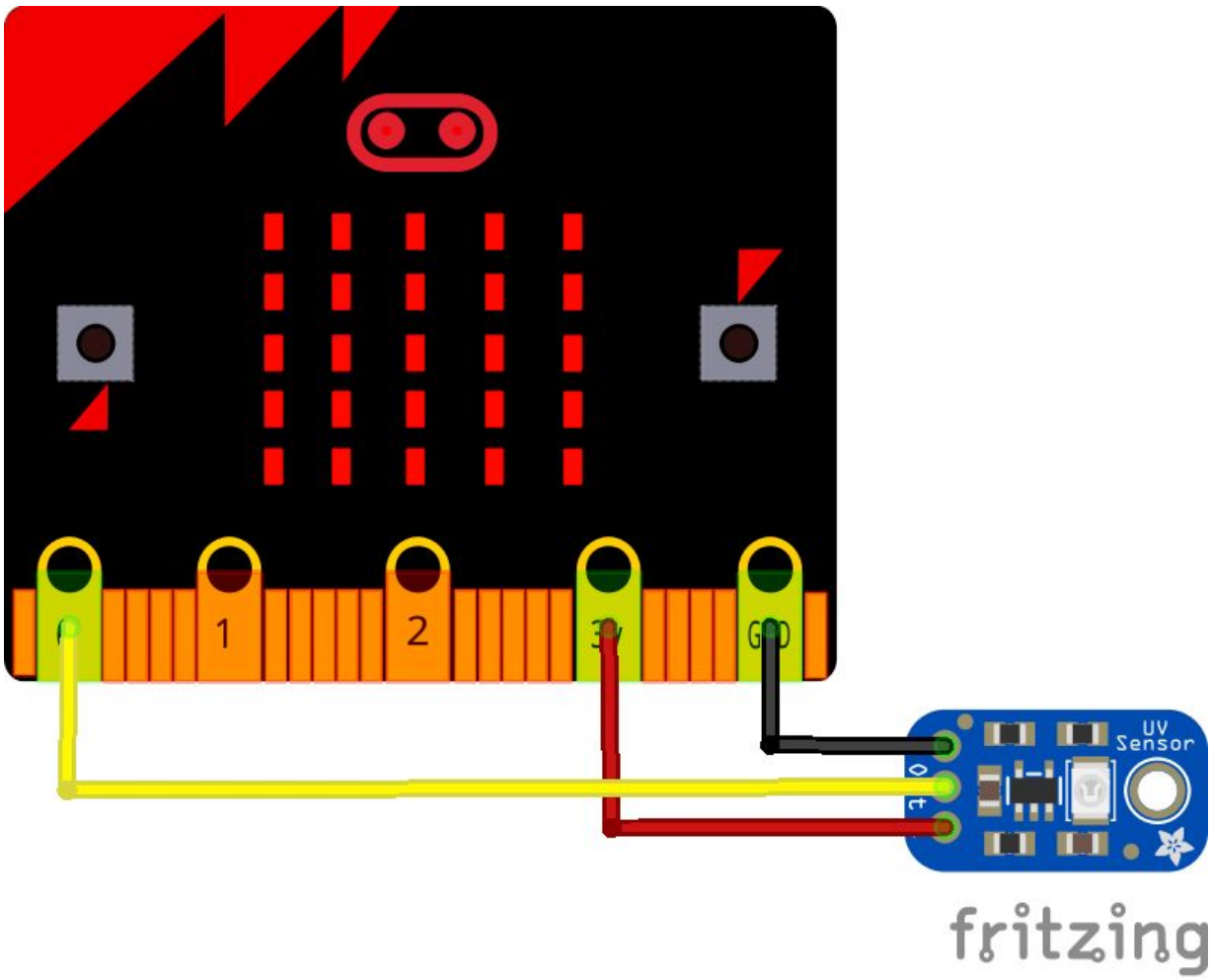
UV Index	0	1	2	3	4	5
Vout(mV)	<50	227	318	408	503	606
Analog Value	<10	46	65	83	103	124
UV Index	6	7	8	9	10	11+
Vout(mV)	696	795	881	976	1079	1170+
Analog Value	142	162	180	200	221	240

Connection

The connections are straightforward and described below, I used 3.3v from my Micro:bit. This was mainly for compatibility with other development boards but the module works with 5v.

1. GND: 0V (Ground)
2. VCC: 3.3V to 5.5V
3. OUT: 0V to 1V (0 to 10 UV Index)

Layout


0
1
2
3V
GND
UV
Sensor
fritzing

Code

Simple code example that reads the value at A0 and outputs the results via the serial monitor

```
sensorVoltage = sensorValue/1024*3.3;

void setup()
{
Serial.begin(9600);
}

void loop()
{
float sensorVoltage;
float sensorValue;
sensorValue = analogRead(A0);
sensorVoltage = sensorValue/1024*3.3;
Serial.print("sensor reading = ");
Serial.print(sensorValue);
Serial.println("");
Serial.print("sensor voltage = ");
Serial.print(sensorVoltage);
Serial.println(" V");
delay(1000);
}
```

Testing

Open the serial monitor and look at the readings

sensor reading = 46.00
sensor voltage = 0.15 V

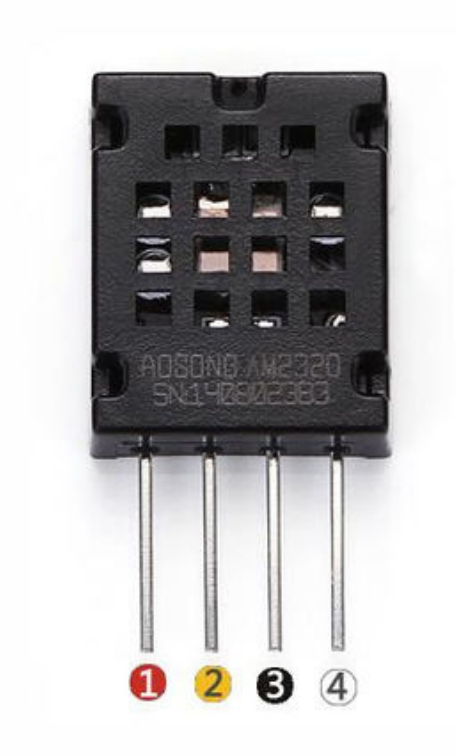
sensor reading = 46.00
sensor voltage = 0.15 V
sensor reading = 46.00
sensor voltage = 0.15 V
sensor reading = 46.00
sensor voltage = 0.15 V
sensor reading = 46.00
sensor voltage = 0.15 V

If you look at the image earlier that corresponds to UV index of 0

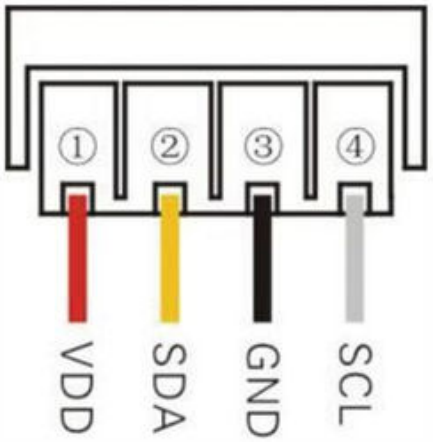
AM2320 temperature and humidity sensor

Temperature and humidity combined sensor AM2320 digital temperature and humidity sensor is a digital signal output has been calibrated. Using special temperature and humidity acquisition technology, ensure that the product has a very high reliability and excellent long-term stability. Sensor consists of a capacitive moisture element and an integrated high-precision temperature measurement devices, and connected with a high-performance microprocessor .

AM2320 communication using a single bus, two communication modes standard I2C. Standard single-bus interface, the system integration becomes easy and quick. Ultra-small size, low power consumption, signal transmission distance up to 20 meters, making all kinds of applications and even the most demanding applications the best choice. I2C communication using standard communication sequence, the user can directly linked to the I2C communication bus without additional wiring, simple to use. Two communication modes are used as humidity, temperature, and other digital information directly CRC checksum temperature-compensated output, users do not need to calculate the secondary digital output, and no need for temperature compensation of the humidity, temperature and humidity can be accurately information. Two communication modes are free to switch, the user can freely choose, easy to use, wide range of applications.

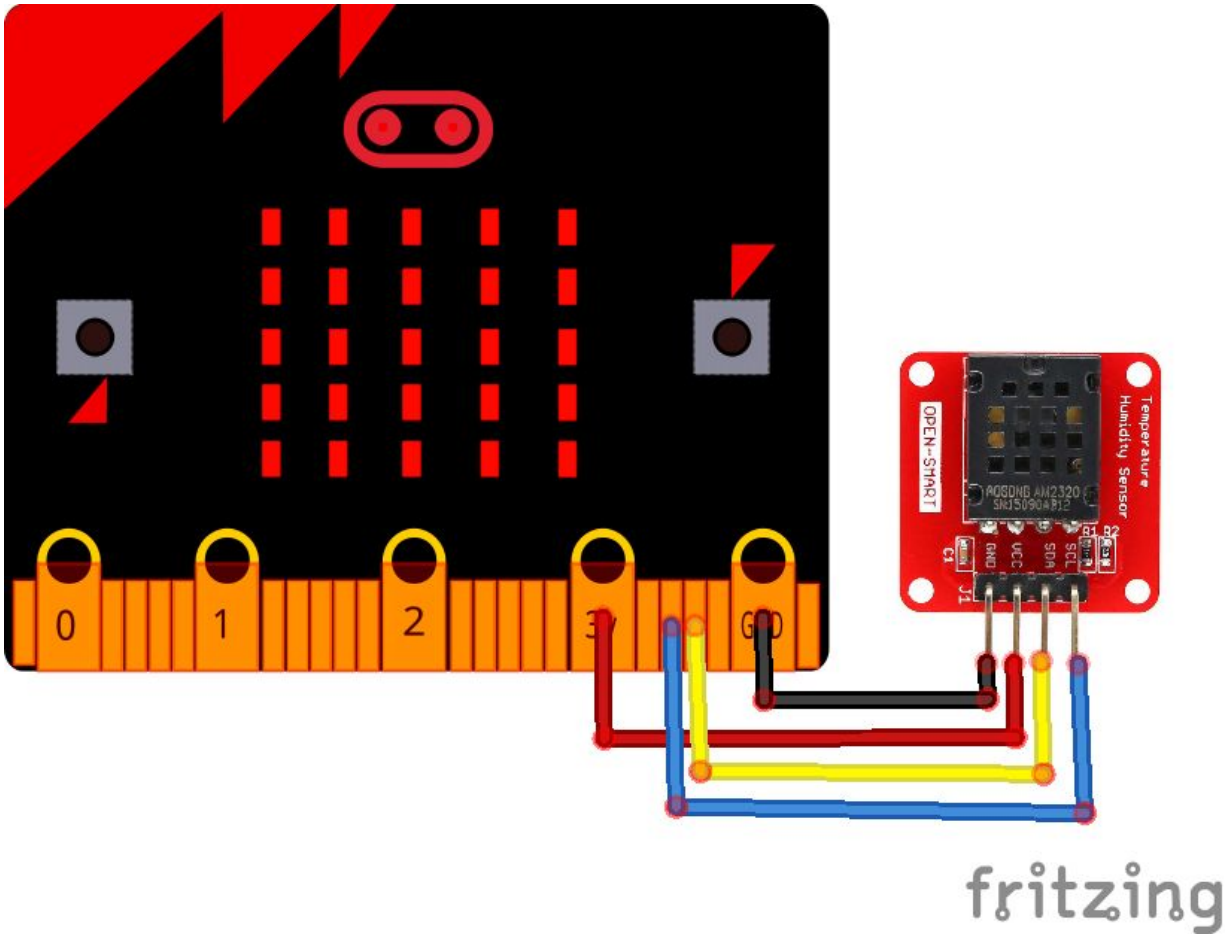


	Name	Description
①	VDD	Power supply (3.1V–5.5V)
②	SDA	Serial data, Di-directional port
③	GND	Ground
④	SCL	Serial clock access port



Layout

I couldn't find a fritzing part but as you can see being a simple I2C sensor with a 3.1 to 5.5v range its straightforward to connect this device to a Micro:bit



Code

You will need to install the folllowing library from https://github.com/EngDial/AM2320 This needs updated This is the default example

```
#include <Wire.h>
#include <AM2320.h>
AM2320 th;

void setup() {
Serial.begin(9600);
Wire.begin();
}

void loop() {
Serial.println("Chip = AM2320");
switch(th.Read()) {
case 2:
Serial.println(" CRC failed");
break;
case 1:
Serial.println(" Sensor offline");
break;
case 0:
Serial.print(" Humidity = ");
Serial.print(th.Humidity);
Serial.println("%");
Serial.print(" Temperature = ");
Serial.print(th.cTemp);
Serial.println("*C");
Serial.println();
break;
}
delay(2000);
}
```

Output

Open the serial monitor

Chip = AM2320
Humidity = 47.10%
Temperature = 24.80*C

Chip = AM2320
Humidity = 48.70%
Temperature = 25.10*C

Chip = AM2320
Humidity = 53.60%
Temperature = 25.40*C

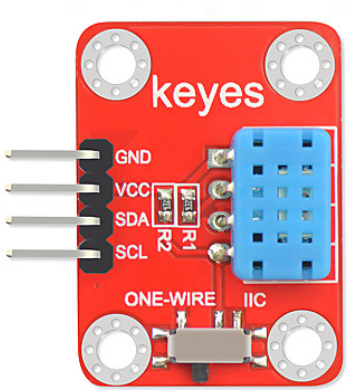
DHT12 temperature sensor

The DHt12 is an upgraded version of the classic DHT11 humidity temperature sensor, it is fully downward compatible, more precise and adds an I2C interface.

Features:
compact size
low power consumption
low voltage operation
Standard I2C and 1-wire interface.

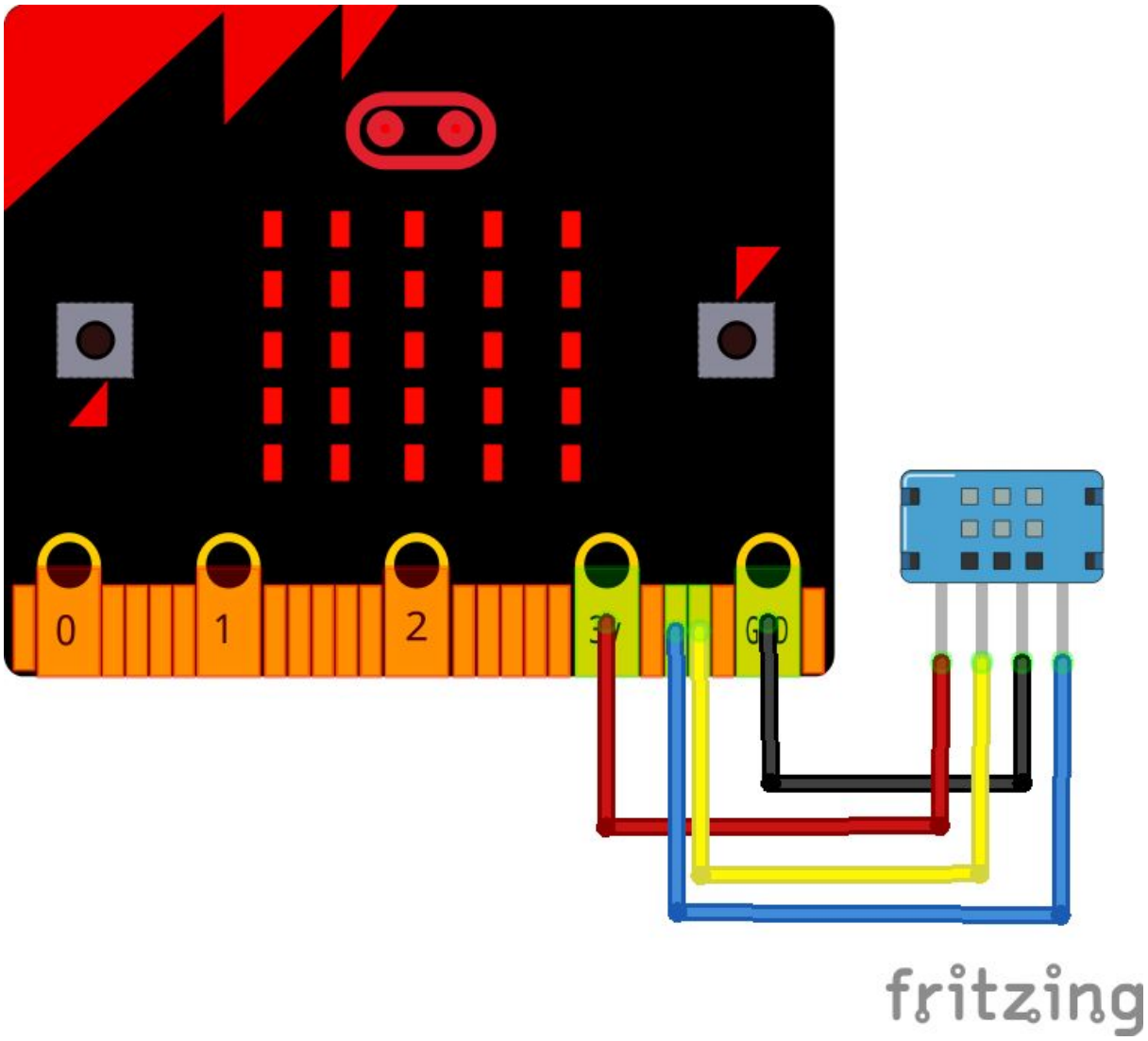
Sensing range
Temperature: -20 ~ +60 C
Humidity: 20-95 RH
Humidity:
Resolution: 0.1%RH
Repeat: -+ 1%RH
Precision 25C @ -+5RH
Temperature:
Resolution: 0.1C
Repeat: -+0.2C
Precision: 25C @ -+0.5C
Power: DC 2.7-5.5V
Normal current 1mA
Standby current 60uA
Sample cycle: > 2 seconds

Pin interface: 1. VDD 2. SDA 3. GND 4. SCL (connect to GND when use as 1-wire)


keyes
GND
VCC
SDA
SCL
R2
R1
ONE-WIRE
IIC

Layout

This shows how to connect the DHT12 to a Microbit



Code

This is from https://github.com/xreef/DHT12_sensor_library, I had to modify it to work correctly Here is the modified version - http://www.microbitlearning.com/wp-content/uploads/2018/02/DHT12_sensor_library-master.zip

```
#include "Arduino.h"

#include <DHT12.h>

// Set dht12 i2c comunication on default Wire pin
DHT12 dht12;

void setup()
{
Serial.begin(112560);
// Start sensor handshake
dht12.begin();
}
int timeSinceLastRead = 0;

void loop()
{
// Report every 2 seconds.
if(timeSinceLastRead > 2000) {
// Reading temperature or humidity takes about 250 milliseconds!
// Read temperature as Celsius (the default)
float t12 = dht12.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
float f12 = dht12.readTemperature(true);
// Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
float h12 = dht12.readHumidity();
```

```
bool dht12Read = true;
// Check if any reads failed and exit early (to try again).
if (isnan(h12) || isnan(t12) || isnan(f12)) {
Serial.println("Failed to read from DHT12 sensor!");

dht12Read = false;
}

if (dht12Read){
// Compute heat index in Fahrenheit (the default)
float hif12 = dht12.computeHeatIndex(f12, h12);
// Compute heat index in Celsius (isFahreheit = false)
float hic12 = dht12.computeHeatIndex(t12, h12, false);
// Compute dew point in Fahrenheit (the default)
float dpf12 = dht12.dewPoint(f12, h12);
// Compute dew point in Celsius (isFahreheit = false)
float dpc12 = dht12.dewPoint(t12, h12, false);

Serial.print("DHT12=> Humidity: ");
Serial.print(h12);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t12);
Serial.print(" *C ");
Serial.print(f12);
Serial.print(" *F\t");
Serial.print(" Heat index: ");
Serial.print(hic12);
Serial.print(" *C ");
Serial.print(hif12);
Serial.print(" *F");
Serial.print(" Dew point: ");
Serial.print(dpc12);
Serial.print(" *C ");
Serial.print(dpf12);
Serial.println(" *F");
}
```

```
timeSinceLastRead = 0;
}
delay(100);
timeSinceLastRead += 100;

}
```

Output

Open the serial monitor

DHT12=> Humidity: 33.70 % Temperature: 20.20 *C 68.36 *F Heat index: 19.16 *C 66.48 *F Dew point: 3.60 *C 38.48 *F
DHT12=> Humidity: 39.70 % Temperature: 20.60 *C 69.08 *F Heat index: 19.75 *C 67.55 *F Dew point: 6.34 *C 43.41 *F
DHT12=> Humidity: 44.60 % Temperature: 21.20 *C 70.16 *F Heat index: 20.54 *C 68.97 *F Dew point: 8.64 *C 47.55 *F
DHT12=> Humidity: 47.80 % Temperature: 21.80 *C 71.24 *F Heat index: 21.28 *C 70.31 *F Dew point: 10.25 *C 50.45 *F
DHT12=> Humidity: 48.90 % Temperature: 22.40 *C 72.32 *F Heat index: 21.97 *C 71.55 *F Dew point: 11.16 *C 52.08 *F
DHT12=> Humidity: 53.90 % Temperature: 22.90 *C 73.22 *F Heat index: 22.65 *C 72.78 *F Dew point: 13.14 *C 55.66 *F

micro:bit and TCS34725 Color Sensor

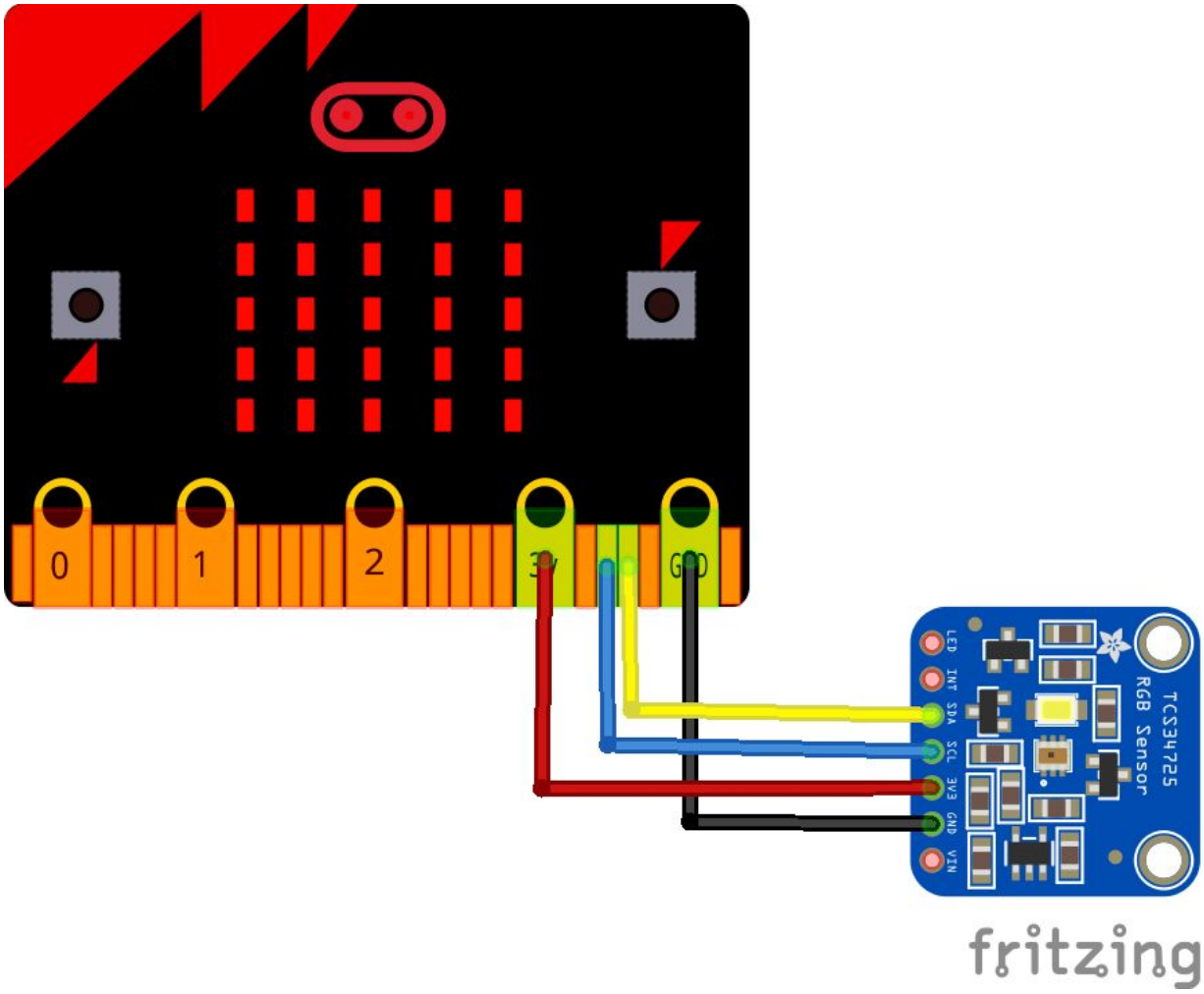
The TCS34725 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately.

The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials. This data is transferred via an I2C to the host.

Features
Integrated IR blocking filter
3.8M:1 dynamic range
Four independent analog-to-digital converters
A reference channel for color analysis (clear channel photodiode)

Connection and Layout

This example is for the micro:bit connected to the module



Code

We will use the adafruit library - https://github.com/adafruit/Adafruit_TCS34725
This is one of the default example

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

/* Example code for the Adafruit TCS34725 breakout library */

/* Initialise with default values (int time = 2.4ms, gain = 1x) */
// Adafruit_TCS34725 tcs = Adafruit_TCS34725();

/* Initialise with specific int time and gain values */
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS,
TCS34725_GAIN_1X);

void setup(void) {
Serial.begin(9600);

if (tcs.begin()) {
Serial.println("Found sensor");
} else {
Serial.println("No TCS34725 found ... check your connections");
while (1);
}

// Now we're ready to get readings!
}

void loop(void) {
uint16_t r, g, b, c, colorTemp, lux;

tcs.getRawData(&r, &g, &b, &c);
```

```
colorTemp = tcs.calculateColorTemperature(r, g, b);
lux = tcs.calculateLux(r, g, b);

Serial.print("Color Temp: "); Serial.print(colorTemp, DEC); Serial.print(" K - ");
Serial.print("Lux: "); Serial.print(lux, DEC); Serial.print(" - ");
Serial.print("R: "); Serial.print(r, DEC); Serial.print(" ");
Serial.print("G: "); Serial.print(g, DEC); Serial.print(" ");
Serial.print("B: "); Serial.print(b, DEC); Serial.print(" ");
Serial.print("C: "); Serial.print(c, DEC); Serial.print(" ");
Serial.println(" ");
}
```

Output

Open the serial monitor, this is what you should see

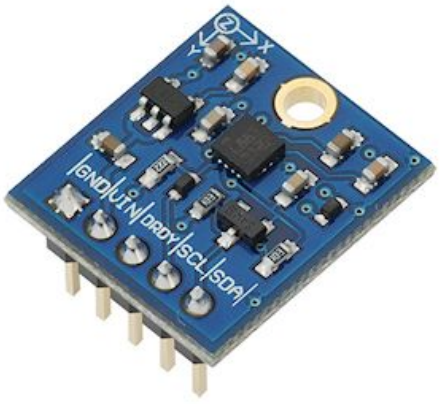
Color Temp: 4554 K - Lux: 379 - R: 1122 G: 831 B: 776 C: 1429
Color Temp: 3173 K - Lux: 181 - R: 475 G: 339 B: 272 C: 707
Color Temp: 3425 K - Lux: 224 - R: 604 G: 435 B: 364 C: 868
Color Temp: 2833 K - Lux: 1497 - R: 2983 G: 2240 B: 1461 C: 5723
Color Temp: 5847 K - Lux: 109 - R: 4109 G: 1327 B: 890 C: 5814
Color Temp: 2767 K - Lux: 460 - R: 4468 G: 1703 B: 1062 C: 6734
Color Temp: 4381 K - Lux: 463 - R: 1379 G: 1012 B: 938 C: 1789
Color Temp: 4276 K - Lux: 588 - R: 1464 G: 1136 B: 997 C: 2153

micro:bit and HMC5883L magnetometer example

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as lowcost compassing and magnetometry. The HMC5883L includes our state-of-theart, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy.

The I2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC).

Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.

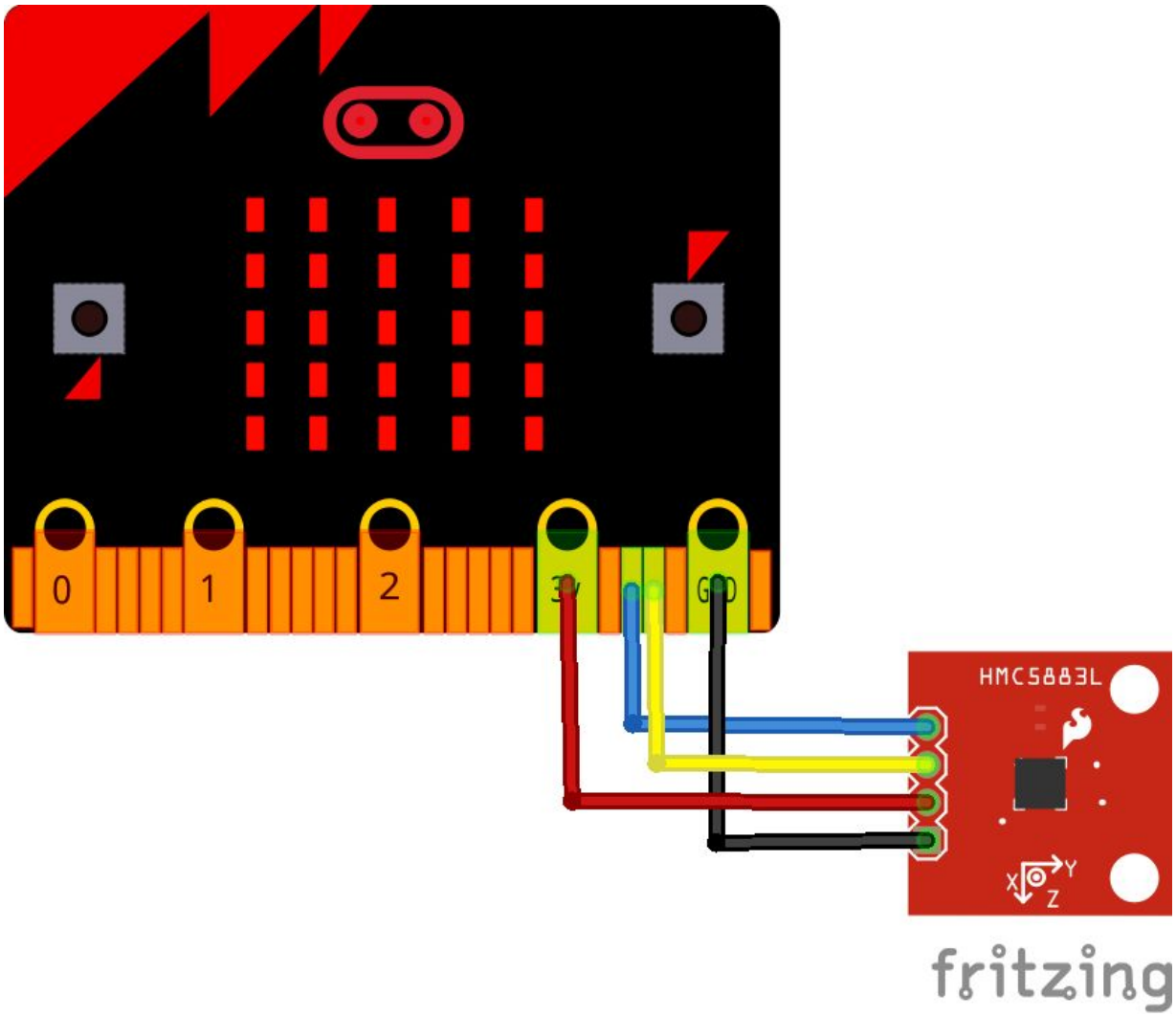


The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over

other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity.

These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry

Connection



Code

```
#include <Wire.h> //I2C Arduino Library

#define addr 0x1E //I2C Address for The HMC5883

void setup(){

  Serial.begin(9600);
  Wire.begin();


  Wire.beginTransmission(addr); //start talking
  Wire.write(0x02); // Set the Register
  Wire.write((byte)0x00); // Tell the HMC5883 to Continuously Measure
  Wire.endTransmission();
}


void loop(){

  int x,y,z; //triple axis data

  //Tell the HMC what regist to begin writing data into
  Wire.beginTransmission(addr);
  Wire.write(0x03); //start with register 3.
  Wire.endTransmission();


//Read the data.. 2 bytes for each axis.. 6 total bytes
  Wire.requestFrom(addr, 6);
  if(6<=Wire.available()){
    x = Wire.read()<<8; //MSB  x
    x |= Wire.read(); //LSB  x
    z = Wire.read()<<8; //MSB  z
```

```
  z |= Wire.read(); //LSB z
  y = Wire.read()<<8; //MSB y
  y |= Wire.read(); //LSB y
 }

 // Show Values
 Serial.print("X Value: ");
 Serial.println(x);
 Serial.print("Y Value: ");
 Serial.println(y);
 Serial.print("Z Value: ");
 Serial.println(z);
 Serial.println();

 delay(500);
}
```

MCP4725 DAC example

MCP4725 is a single channel, 12-bit, voltage output Digital-to-Analog Converter with integrated EEPROM and an I2C Compatible Serial Interface.

Features

12-Bit Resolution
On-Board Non-Volatile Memory (EEPROM)
±0.2 LSB DNL (typ)
External A0 Address Pin
Normal or Power-Down Mode
Fast Settling Time of 6µs (typ)
External Voltage Reference (VDD)
Low Power Consumption
Single-Supply Operation: 2.7V to 5.5V
Standard (100 kbps), Fast (400 kbps) andHigh Speed (3.4 Mbps) Modes
Extended Temperature Range: -40°C to +125°C

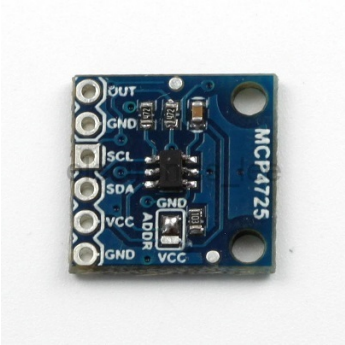
As this is a 12 bit DAC converter. What this means is that it will accept up to 4096 possible inputs to provide an analog output, where an output value of zero is zero and an output value of 4095 is full scale.

Full scale is determined by the reference voltage you supply to the VCC pin. Also you can see from above that the supply voltage can be anywhere from 2.7 volts to 5.5 volts. We will use 5v, or as close as what is supplied via the USB in. You may want to measure this voltage for accurate readings, I've seen this vary.

This means that to work out the value of the Least Significant Bit (LSB) is as follows:

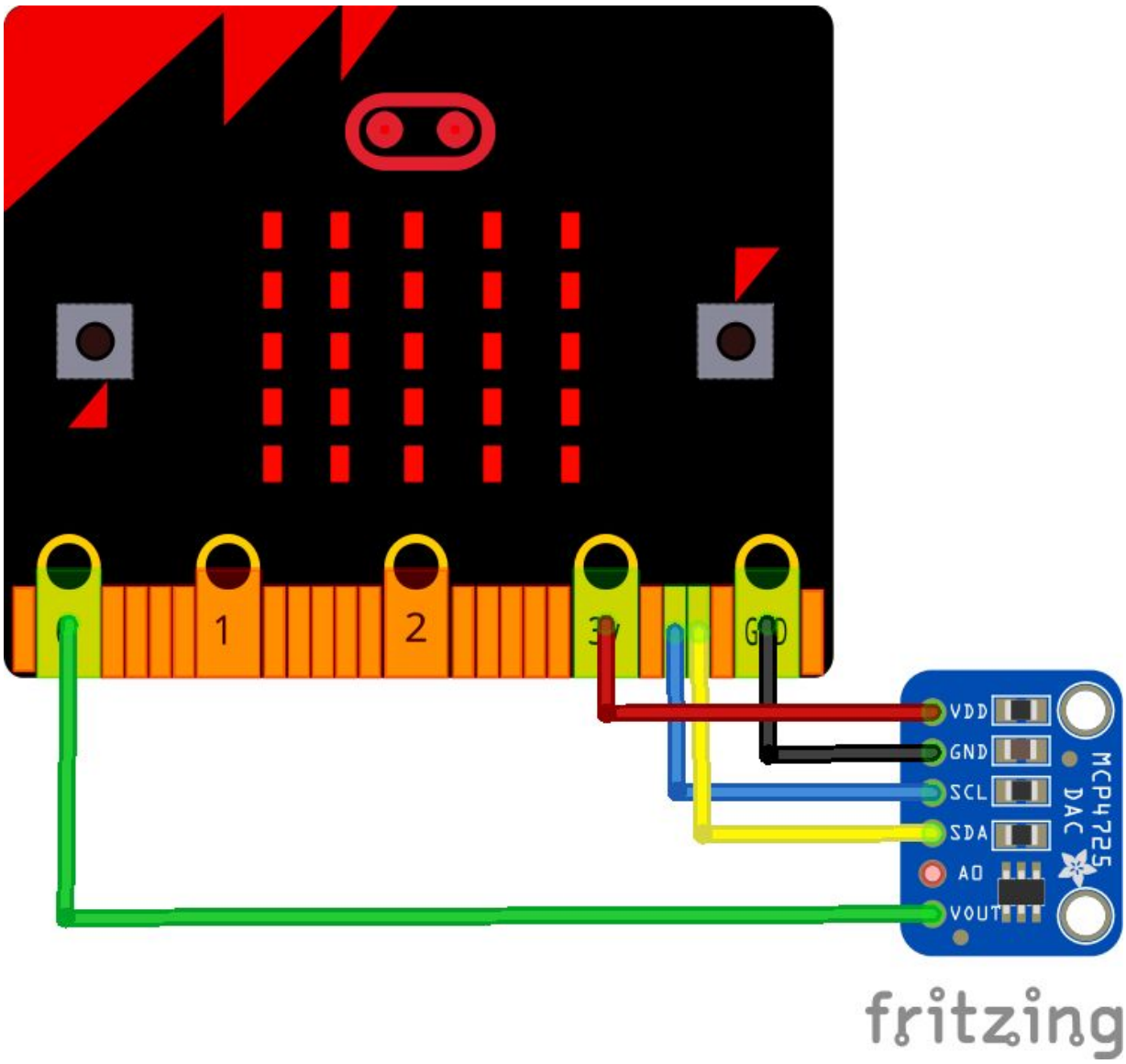
1 LSB = VCC Voltage / 4096

Again the easiest way to interface this to an Arduino is to purchase a module, these are available from many sources, here is what my one looked at.



Layout

Fairly straightforward as its an I2C device, I connected the Vout to 0.


1
2
3V
GND
VDD
GND
SCL
SDA
A0
VOUT
MCP4725
DAC
fritzing

Code

The module has the ability to use different I2C addresses My one was the default 0x60 address

This example uses the Adafruit MCP4725 library

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#define voltsIn PIN_A0

Adafruit_MCP4725 dac; // constructor

void setup(void) {
Serial.begin(9600);
dac.begin(0x60); // The I2C Address: Run the I2C Scanner if you're not sure

}

void loop(void) {

uint32_t dac_value;
int adcValueRead = 0;
float voltageRead = 0;

float dac_expected_output;


for (dac_value = 0; dac_value < 4096; dac_value = dac_value + 15)
{
dac_expected_output = (3.3/4096.0) * dac_value;
dac.setVoltage(dac_value, false);
delay(250);
adcValueRead = analogRead(voltsIn);
voltageRead = (adcValueRead * 3.3 )/ 1024.0;
```

```
Serial.print("DAC Value: ");
Serial.print(dac_value);

Serial.print("\tExpected Voltage: ");
Serial.print(dac_expected_output,3);

Serial.print("\tArduino ADC Value: ");
Serial.print(adcValueRead);

Serial.print("\tArduino Voltage: ");
Serial.println(voltageRead,3);
}
}
```

SHT31 temperature and humidity sensor

SHT31 is the next generation of Sensirion's temperature and humidity sensors. It builds on a new CMOSens® sensor chip that is at the heart of Sensirion's new humidity and temperatureplatform.

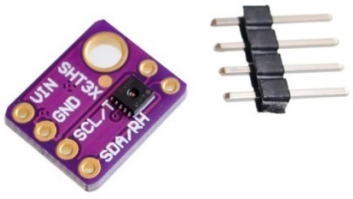
The SHT3x-DIS has increased intelligence, reliability and improved accuracy specifications compared to its predecessor. Its functionality includes enhanced signal processing, two distinctive and user selectable I2C addresses and communication speeds of up to 1 MHz. The DFN package has a footprint of 2.5 x 2.5 mm2 while keeping a height of 0.9 mm.

This allows for integration of the SHT3x-DIS into a great variety of applications.

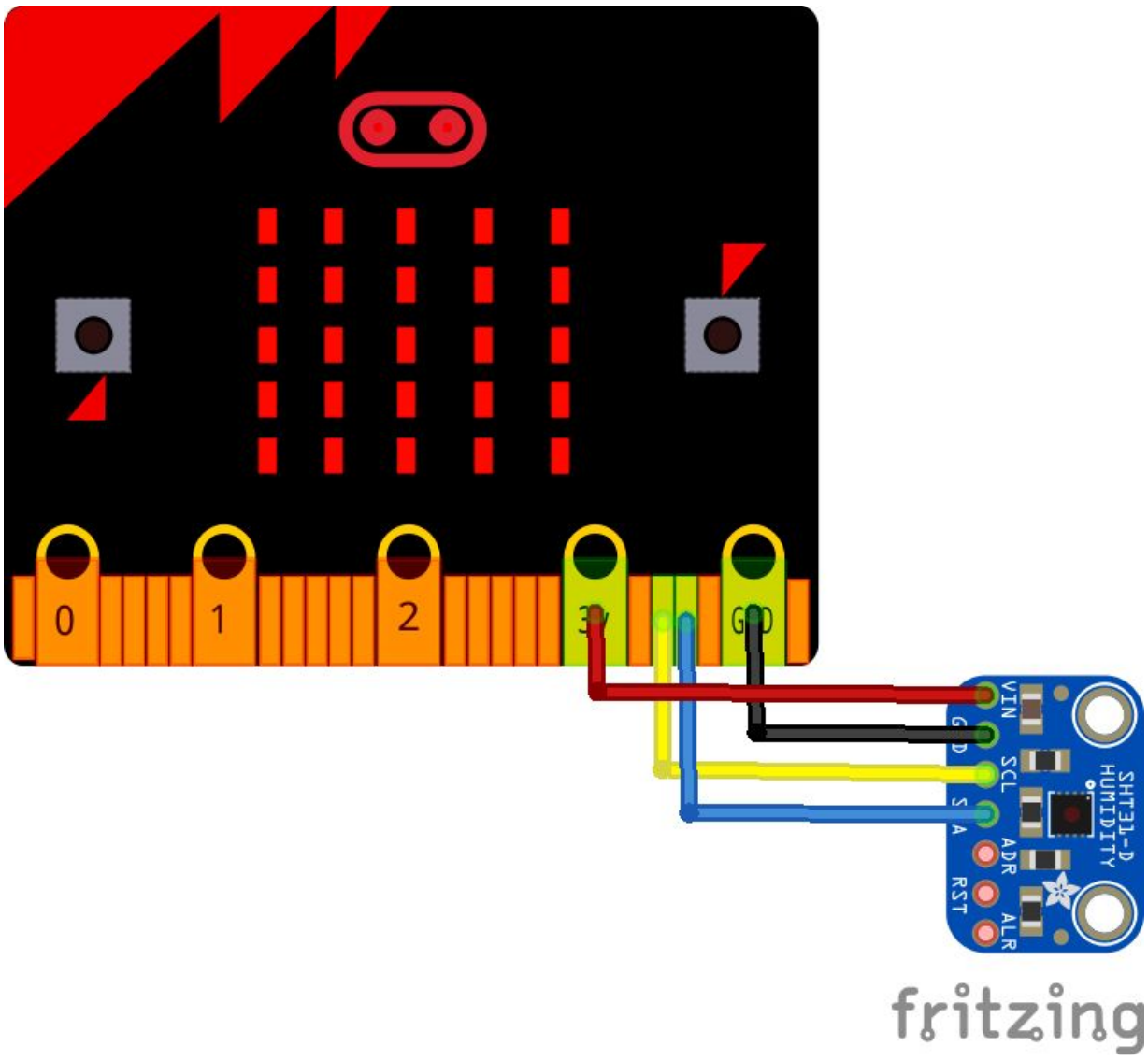
Features

Fully calibrated, linearized, and temperature compensated digital output
Wide supply voltage range, from 2.4 V to 5.5 V
I2C Interface with communication speeds up to 1 MHz and two user selectable addresses

I bought the following module



Layout


0
1
2
3V
GND
VIN
GND
SCL
SDA
ADR
RST
ALR
SHT31-D
HUMIDITY
fritzing

Code

This example uses the adafruit sht31 library - https://github.com/adafruit/Adafruit_SHT31

```
#include <Arduino.h>
#include <Wire.h>
#include "Adafruit_SHT31.h"

Adafruit_SHT31 sht31 = Adafruit_SHT31();


void setup()
{
 Serial.begin(9600);
 if (! sht31.begin(0x44))
 {
  Serial.println("Couldn't find SHT31");
  while (1) delay(1);
 }
}


void loop()
{
 float t = sht31.readTemperature();
 float h = sht31.readHumidity();

 if (! isnan(t))
 {
  Serial.print("Temp *C = "); Serial.println(t);
 }
 else
 {
  Serial.println("Failed to read temperature");
 }
```

```
  if (! isnan(h))
  {
    Serial.print("Hum. % = "); Serial.println(h);
  }
  else
  {
    Serial.println("Failed to read humidity");
  }
  Serial.println();
  delay(1000);
}
```

Output

Open the serial monitor window

Temp *C = 27.27
Hum. % = 32.90

Temp *C = 27.01
Hum. % = 32.20

Temp *C = 26.76
Hum. % = 31.83

Temp *C = 26.52
Hum. % = 31.79

micro:bit and bmp180 sensor example

This bmp180 from Bosch is the best low-cost sensing solution for measuring barometric pressure and temperature. The sensor is soldered onto a PCB with a 3.3V regulator, I2C level shifter and pull-up resistors on the I2C pins. The BMP180 replaces the BMP085.

Specification

- Pressure sensing range: 300-1100 hPa (9000m to -500m above sea level)
- Up to 0.03hPa / 0.25m resolution
- -40 to +85°C operational range, +-2°C temperature accuracy

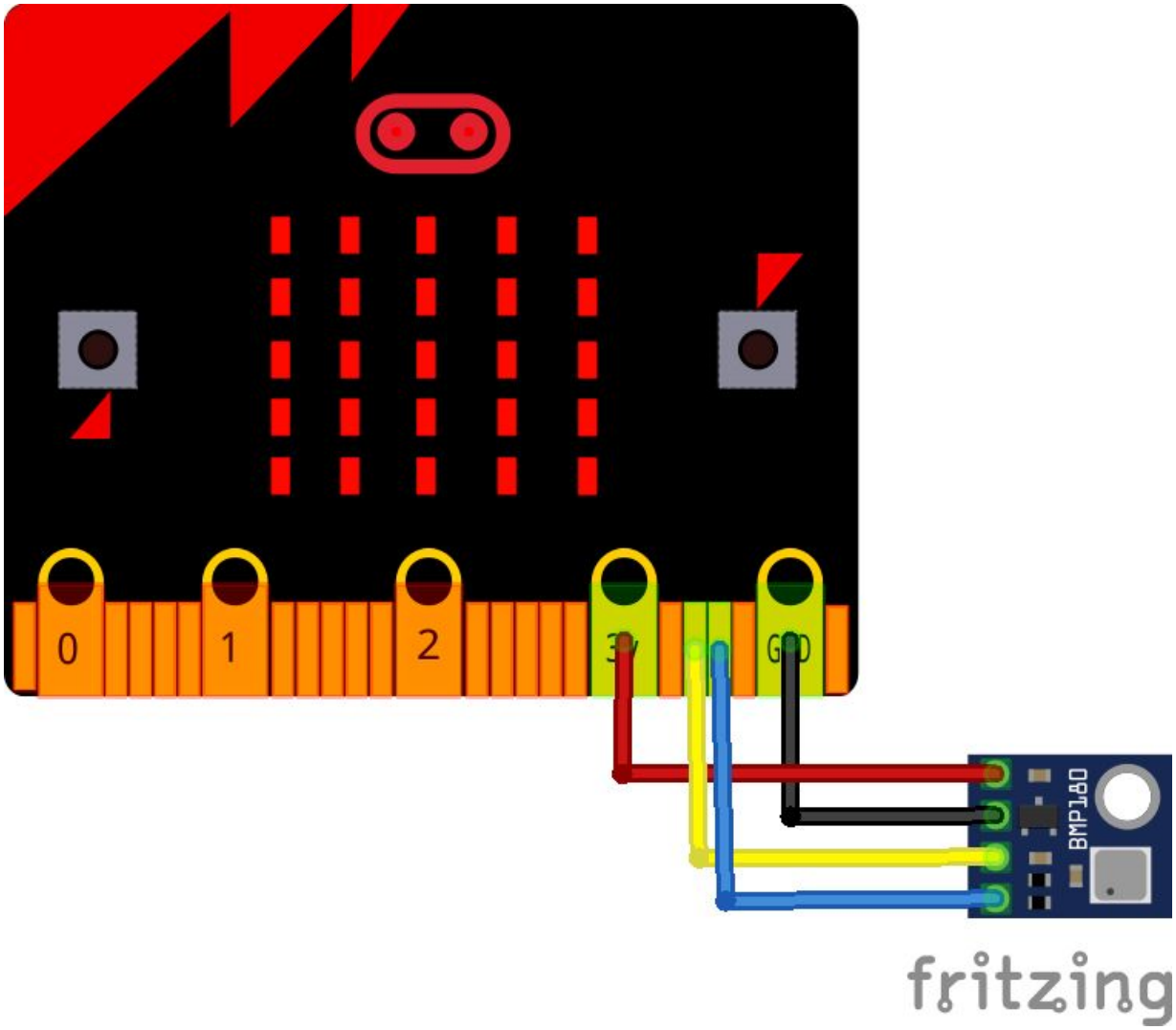
Here is a breakout which makes it easy to use the sensor, link at the bottom.

Wiring and layout

Make the following connections

GND <-> GND
3V3 <-> VIN (or 3Vo)
P19 <-> SCL
P20 <-> SDA

Here is a layout drawn in fritzing



Code

You will need the Adafruit BMP085 library for this example, you can either download it or use the library manager in newer Arduino IDEs.
https://github.com/adafruit/Adafruit-BMP085-Library

In this example I am only looking at the temperature and pressure but there are other functions in the library

```
#include <Wire.h>
#include <Adafruit_BMP085.h>

Adafruit_BMP085 bmp;

void setup()
{
  Serial.begin(9600);
  if (!bmp.begin())
  {
    Serial.println("Could not find BMP180 or BMP085 sensor at 0x77");
    while (1) {}
  }
}

void loop()
{
  Serial.print("Temperature = ");
  Serial.print(bmp.readTemperature());
  Serial.println(" Celsius");

  Serial.print("Pressure = ");
  Serial.print(bmp.readPressure());
  Serial.println(" Pascal");


  Serial.println();
```

```
  delay(5000);
}
```

Output

Open the Serial monitor and you should see something like this

Temperature = 20.10 Celsius
Pressure = 100775 Pascal

Temperature = 20.00 Celsius
Pressure = 100773 Pascal

Temperature = 19.90 Celsius
Pressure = 100778 Pascal

Temperature = 25.80 Celsius
Pressure = 100835 Pascal

In review

We hope you enjoyed this ebook , we have uploaded source code to

https://github.com/getelectronics/microbit-arduino-ebook