BME280 Environmental Sensor User Manual

OVERVIEW

The BME280 is as combined digital humidity, pressure and temperature sensor. Its small dimensions, low power consumption, high precision and stability allow the implementation in environmental monitor, whether forecast, altitude detection and IOT application.

FEATURES

- Support I2C interface, I2C device address could be set by changing I/O or welds
- Supports SPI interface. Default I2C, you can change to SPI by change I/O
- Integrated Level convert circuit, compatible with 3.3V/5V
- Provide examples and user guide (Raspberry Pi/Arduino/STM32)

SPECIFICATION

Operating voltage:	5V/3.3V
Interface:	I2C/SPI
Temperature range:	-40~85°C (Resolution 0.1°C, tolerance ±1°C)
Humidity range:	0~100%rh (Resolution 0.008%RH, tolerance ±1°C)
Pressure range:	300~1100hPa(Resolution 0.18Pa, tolerance ±1hPa)
Dimensions:	27mmx20mm
Holes size:	2.0mm

INTERFACES

I2C

PIN	Arduino	STM32	RASPBERRY	Description
VCC	3.3V/5V	3.3V /5V	3.3V /5V	Power
GND	GND	GND	GND	Ground
SDA	A4	PB7	SDA	I2C Data
SCL	A5	PB6	SCL	I2C Clock
ADDR	NC/GND	NC/GND	NC/GND	Address Select (High by default)： High：address is 0x77 Low：address is 0x76
CS	NC	NC	NC	NC

SPI:

PIN	Arduino	STM32	RASPBERRY	Description
VCC	3.3V /5V	3.3V /5V	3.3V /5V	Power
GND	GND	GND	GND	Ground
MOSI	D11	PA7	MOSI	SPI Data input
SCK	D13	PA5	SCK	SPI Clock input
MISO	D12	PA6	MISO	SPI Data output
CS	D10	PB6	27	SPI chip select, Low active

WORKING WITH EXAMPLES

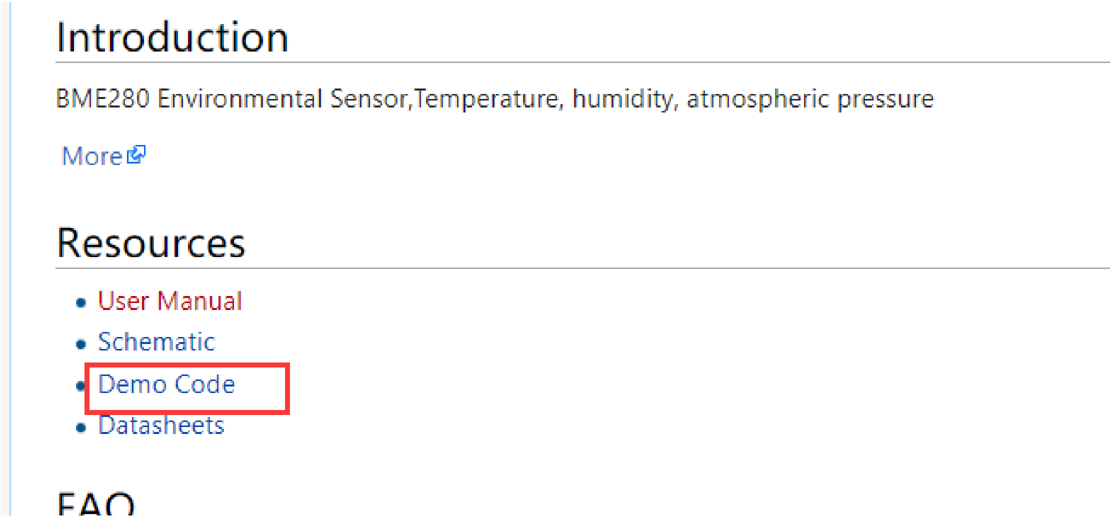
WORKING WITH RASPBERRY PI

INSTALL LIBRARIES

To use examples we provide, you need to first install WiringPi library, or it could be be used normally. About how to install WiringPi library, you can visit the page: Libraries installation for RPi for details.

EXAMPLE DOWNLOAD

Visit Our Wiki and find the page of BME280 Environmental Sensor, download the demo code.



Introduction

BME280 Environmental Sensor,Temperature, humidity, atmospheric pressure

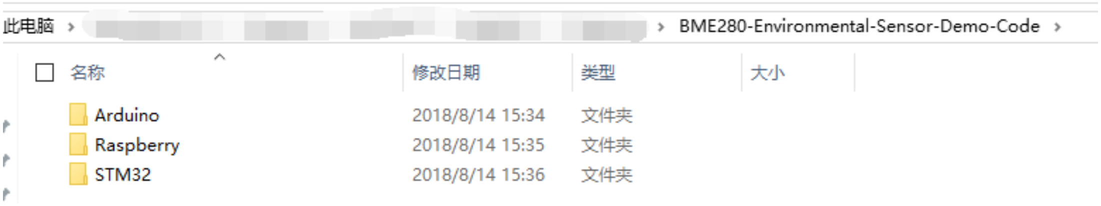
More

Resources

- User Manual
- Schematic
- Demo Code
- Datasheets

FAQ

Extract it to get the folders as below:



此电脑 › [redacted] › BME280-Environmental-Sensor-Demo-Code ›

□ 名称	修改日期	类型	大小
Arduino	2018/8/14 15:34	文件夹	
Raspberry	2018/8/14 15:35	文件夹	
STM32	2018/8/14 15:36	文件夹	

Copy Raspberry folder to your Raspberry Pi. You can copy it to the root of the TF card which you use for your Raspberry Pi.

CONNECTION

1. Enable I2C or SPI interface first， open configuration page with the command：

   sudo raspi-config

2. Enable the interface separately.

   - For SPI: Choose Interfacing Options-> SPI -> Yes

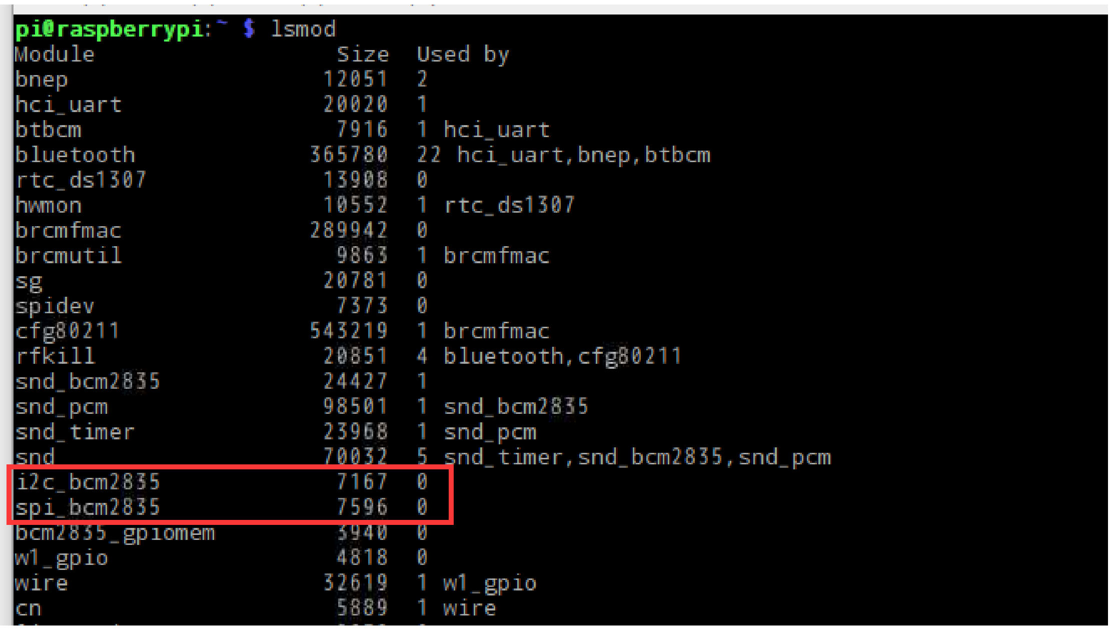
   - For I2C (default): Choose Interfacing Options->I2C->Yes

3. After configuring, reboot your Raspberry Pi

   sudo reboot

4. After rebooting, you can check if I2C or SPI module has been enabled with command (If there are I2C or SPI information printed, modules are started):

   lsmod



```
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
bnep                   12051  2
hci_uart               20020  1
btbcm                   7916  1 hci_uart
bluetooth             365780  22 hci_uart,bnep,btbcm
rtc_ds1307             13908  0
hwmon                  10552  1 rtc_ds1307
brcmfmac              289942  0
brcmutil                9863  1 brcmfmac
sg                     20781  0
spidev                  7373  0
cfg80211              543219  1 brcmfmac
rfkill                 20851  4 bluetooth,cfg80211
snd_bcm2835            24427  1
snd_pcm                98501  1 snd_bcm2835
snd_timer              23968  1 snd_pcm
snd                    70032  5 snd_timer,snd_bcm2835,snd_pcm
i2c_bcm2835             7167  0
spi_bcm2835             7596  0
bcm2835_gpiomem         3940  0
w1_gpio                 4818  0
wire                   32619  1 w1_gpio
cn                      5889  1 wire
```

5. Connet BME280 Environmental Sensor (called BME280) to your Raspberry Pi according to the I2C table above.

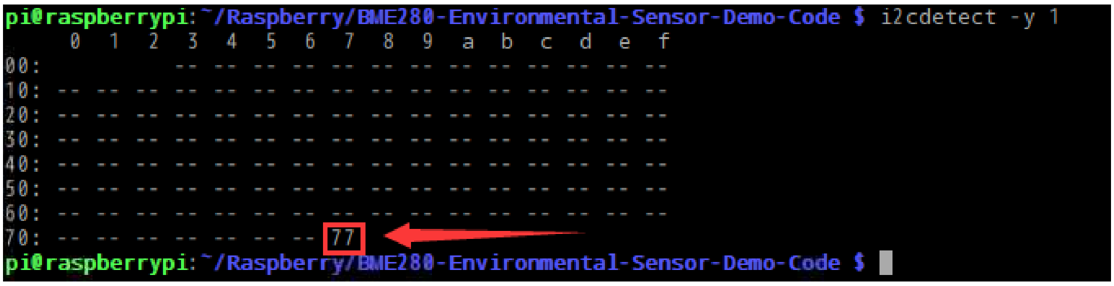
6. The address of I2C device is 0x77 by default. If you want to change it to 0x76, yu can connect ADDR to GND.

7. Install I2C-tools tool, for I2C devices detecting

   sudo apt-get install i2c-tools

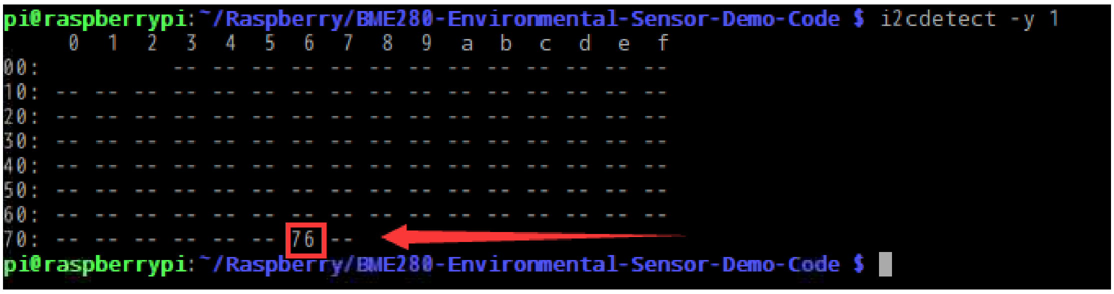
8. You can query I2C devices with command:

   i2cdetect -y 1



```
pi@raspberrypi:~/Raspberry/BME280-Environmental-Sensor-Demo-Code $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
pi@raspberrypi:~/Raspberry/BME280-Environmental-Sensor-Demo-Code $
```

   If there is 77 printed (76 if you change the address), it means that BME280 has connected to Raspberry Pi normally.



```
pi@raspberrypi:~/Raspberry/BME280-Environmental-Sensor-Demo-Code $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@raspberrypi:~/Raspberry/BME280-Environmental-Sensor-Demo-Code $
```

Note: Please make sure that there are not other I2C devices which has the same address as BME280. For SPI connection, you can refer to the SPI table above.

RUNNING DEMO CODE

1. Connecting BME280 as above

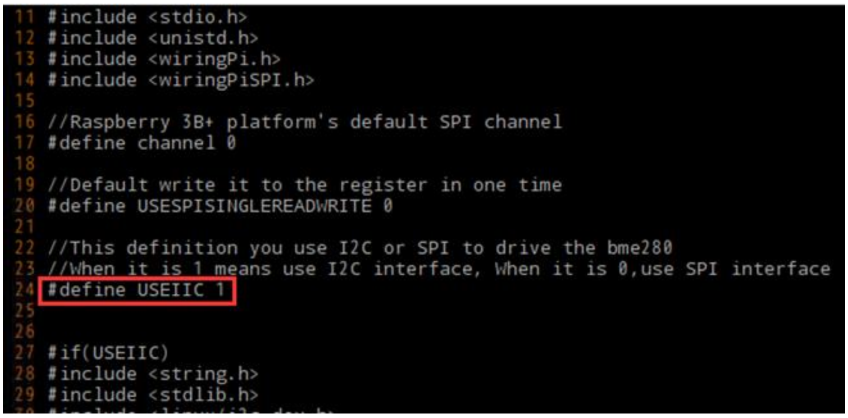
2. Enter directory of BME280-Environmental-Sensor-Demo-Code (which is included in the folder we copied before):

   cd BME280-Environmental-Sensor-Demo-Code

3. Open and edit file main.c

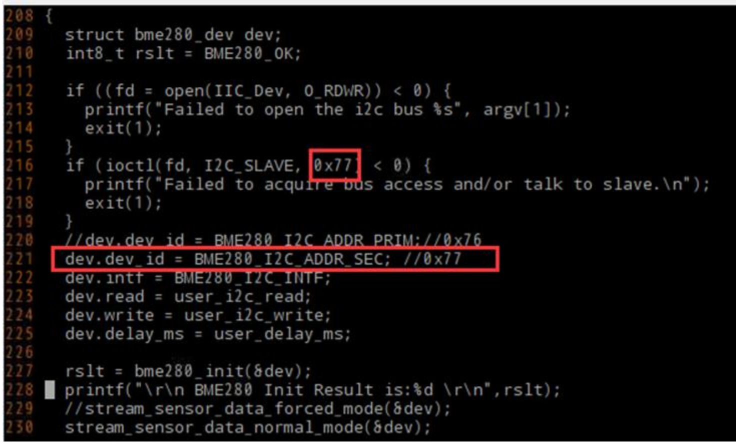
   vim main.c

   - If you use I2C connection which is default setting, you should set the USEIIC define to 1:



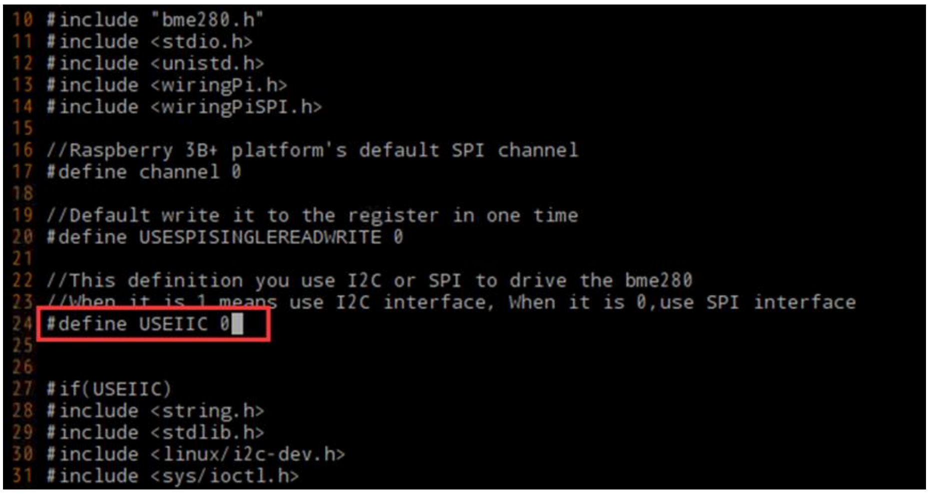
```
#include <stdio.h>
#include <unistd.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>

//Raspberry 3B+ platform's default SPI channel
#define channel 0

//Default write it to the register in one time
#define USESPISINGLEREADWRITE 0

//This definition you use I2C or SPI to drive the bme280
//When it is 1 means use I2C interface, When it is 0,use SPI interface
#define USEIIC 1


#if(USEIIC)
#include <string.h>
#include <stdlib.h>
```

Then check the address of I2C device on the code, the address used on code should be same as the one we detected before:



```
{
  struct bme280_dev dev;
  int8_t rslt = BME280_OK;

  if ((fd = open(IIC_Dev, O_RDWR)) < 0) {
    printf("Failed to open the i2c bus %s", argv[1]);
    exit(1);
  }
  if (ioctl(fd, I2C_SLAVE, 0x77) < 0) {
    printf("Failed to acquire bus access and/or talk to slave.\n");
    exit(1);
  }
  //dev.dev_id = BME280_I2C_ADDR_PRIM;//0x76
  dev.dev_id = BME280_I2C_ADDR_SEC; //0x77
  dev.intf = BME280_I2C_INTF;
  dev.read = user_i2c_read;
  dev.write = user_i2c_write;
  dev.delay_ms = user_delay_ms;

  rslt = bme280_init(&dev);
  printf("\r\n BME280 Init Result is:%d \r\n",rslt);
  //stream_sensor_data_forced_mode(&dev);
  stream_sensor_data_normal_mode(&dev);
```

- If you use SPI connection. You should change the USEIIC define to 0 on main.c:



```
#include "bme280.h"
#include <stdio.h>
#include <unistd.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>

//Raspberry 3B+ platform's default SPI channel
#define channel 0

//Default write it to the register in one time
#define USESPISINGLEREADWRITE 0

//This definition you use I2C or SPI to drive the bme280
//When it is 1 means use I2C interface, When it is 0,use SPI interface
#define USEIIC 0


#if(USEIIC)
#include <string.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
```

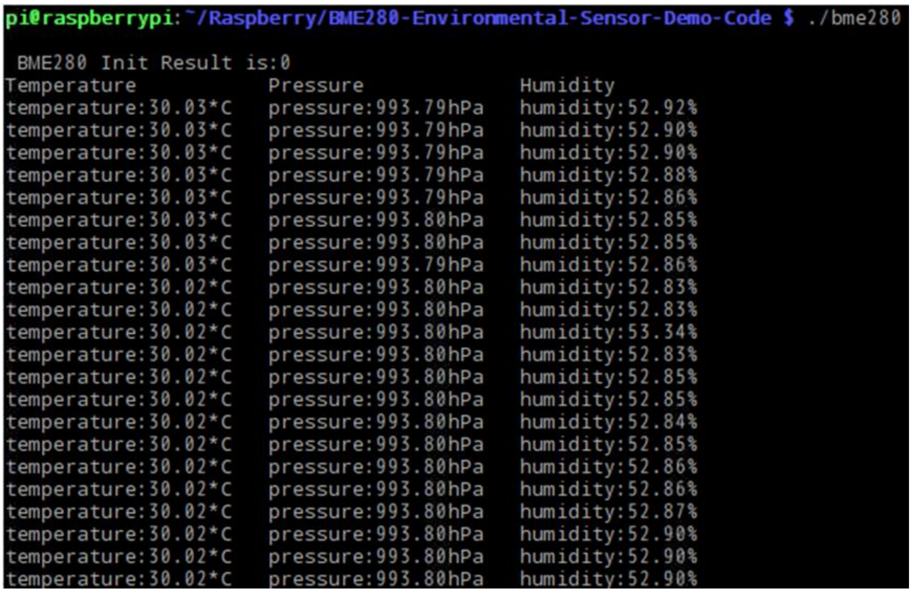
4. Save and exit. Then re-compile the demo code:

   sudo make clean

   sudo make

5. Running the demo code:

   sudo ./bme280



```
pi@raspberrypi:~/Raspberry/BME280-Environmental-Sensor-Demo-Code $ ./bme280

 BME280 Init Result is:0
Temperature                 Pressure                Humidity
temperature:30.03*C     pressure:993.79hPa     humidity:52.92%
temperature:30.03*C     pressure:993.79hPa     humidity:52.90%
temperature:30.03*C     pressure:993.79hPa     humidity:52.90%
temperature:30.03*C     pressure:993.79hPa     humidity:52.88%
temperature:30.03*C     pressure:993.79hPa     humidity:52.86%
temperature:30.03*C     pressure:993.80hPa     humidity:52.85%
temperature:30.03*C     pressure:993.80hPa     humidity:52.85%
temperature:30.03*C     pressure:993.79hPa     humidity:52.86%
temperature:30.02*C     pressure:993.80hPa     humidity:52.83%
temperature:30.02*C     pressure:993.80hPa     humidity:52.83%
temperature:30.02*C     pressure:993.80hPa     humidity:53.34%
temperature:30.02*C     pressure:993.80hPa     humidity:52.83%
temperature:30.02*C     pressure:993.80hPa     humidity:52.85%
temperature:30.02*C     pressure:993.80hPa     humidity:52.85%
temperature:30.02*C     pressure:993.80hPa     humidity:52.84%
temperature:30.02*C     pressure:993.80hPa     humidity:52.85%
temperature:30.02*C     pressure:993.80hPa     humidity:52.86%
temperature:30.02*C     pressure:993.80hPa     humidity:52.86%
temperature:30.02*C     pressure:993.80hPa     humidity:52.87%
temperature:30.02*C     pressure:993.80hPa     humidity:52.90%
temperature:30.02*C     pressure:993.80hPa     humidity:52.90%
temperature:30.02*C     pressure:993.80hPa     humidity:52.90%
```

Note: if there are not any data outputted after running the demo code or get wrong data, please check the hardware connection and address used.

WORKING WITH ARDUINO

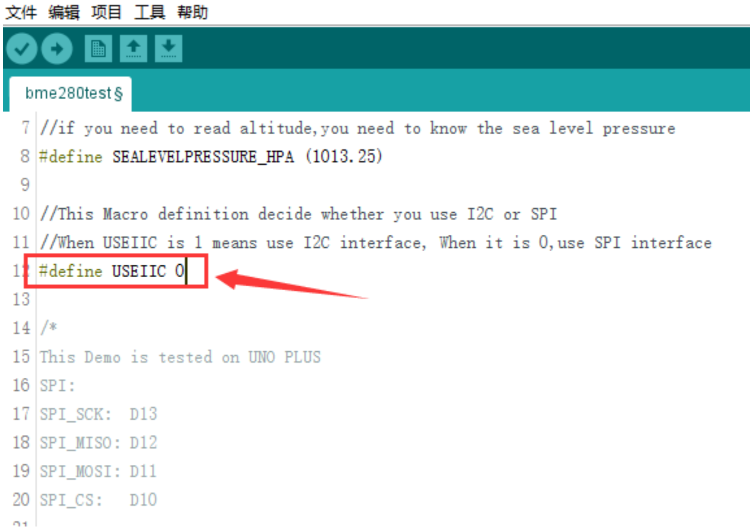
1. Open Arduino folder (from the one we download), Copy folder BME280-Arduino-Library to Libraries directory of IDE, which locates on installation directory of Arduino IDE.

2. Open Arduino IDE, Choose File->Examples-> BME280_Libreay->bme280test to open the demo code.

3. Connect BME280 to Arduino according to Interfaces

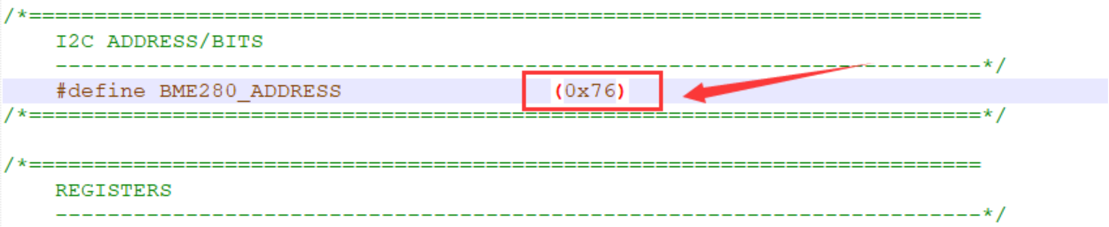
   - By default, communication interface is I2C with device address 0x77

   - If you want to use SPI interface, you need to change USEIIC to 0 on demo code



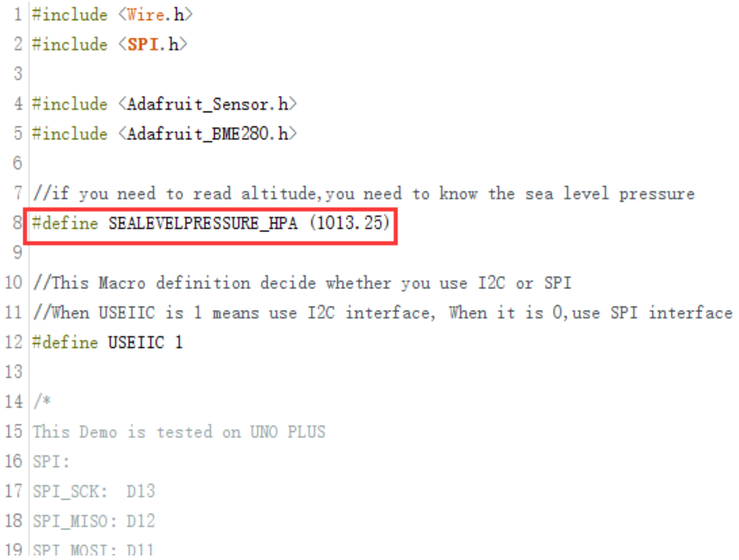
```
//if you need to read altitude,you need to know the sea level pressure
#define SEALEVELPRESSURE_HPA (1013.25)

//This Macro definition decide whether you use I2C or SPI
//When USEIIC is 1 means use I2C interface, When it is 0,use SPI interface
#define USEIIC 0

/*
This Demo is tested on UNO PLUS
SPI:
SPI_SCK:  D13
SPI_MISO: D12
SPI_MOSI: D11
SPI_CS:   D10
```

4. If you want to change the device address to 0x76, you could connect ADDR to ND, and change the BME280_ADDRES value on Adafruit.h of demo code:



```
/*=========================================================================
    I2C ADDRESS/BITS
    -----------------------------------------------------------------------*/
    #define BME280_ADDRESS                (0x76)
/*=========================================================================*/

/*=========================================================================
    REGISTERS
    -----------------------------------------------------------------------*/
```

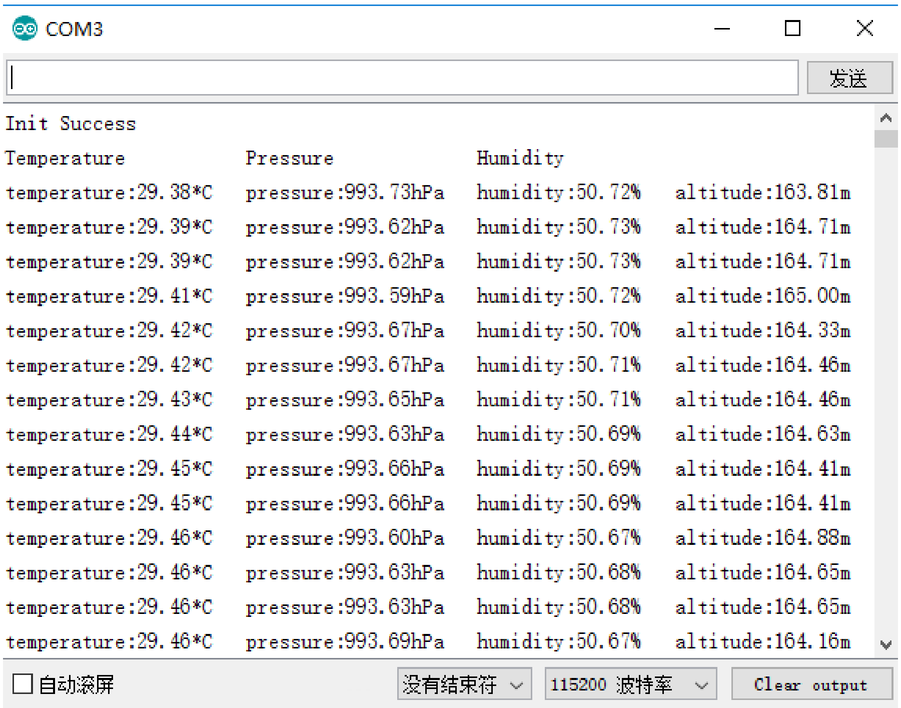
5. If you want to get correct altitude data, you should first measure the atmosphere pressure of local sea level and then change the define of SEALEVELPRESSURE_HPA



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

#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

//if you need to read altitude,you need to know the sea level pressure
#define SEALEVELPRESSURE_HPA (1013.25)

//This Macro definition decide whether you use I2C or SPI
//When USEIIC is 1 means use I2C interface, When it is 0,use SPI interface
#define USEIIC 1

/*
This Demo is tested on UNO PLUS
SPI:
SPI_SCK:  D13
SPI_MISO: D12
SPI_MOSI: D11
```

6. Then compile and download to your Arduino board.

7. Open Serial monitor and set the baud rate to 11520



```
Init Success
Temperature             Pressure            Humidity
temperature:29.38*C     pressure:993.73hPa  humidity:50.72%   altitude:163.81m
temperature:29.39*C     pressure:993.62hPa  humidity:50.73%   altitude:164.71m
temperature:29.39*C     pressure:993.62hPa  humidity:50.73%   altitude:164.71m
temperature:29.41*C     pressure:993.59hPa  humidity:50.72%   altitude:165.00m
temperature:29.42*C     pressure:993.67hPa  humidity:50.70%   altitude:164.33m
temperature:29.42*C     pressure:993.67hPa  humidity:50.71%   altitude:164.46m
temperature:29.43*C     pressure:993.65hPa  humidity:50.71%   altitude:164.46m
temperature:29.44*C     pressure:993.63hPa  humidity:50.69%   altitude:164.63m
temperature:29.45*C     pressure:993.66hPa  humidity:50.69%   altitude:164.41m
temperature:29.45*C     pressure:993.66hPa  humidity:50.69%   altitude:164.41m
temperature:29.46*C     pressure:993.60hPa  humidity:50.67%   altitude:164.88m
temperature:29.46*C     pressure:993.63hPa  humidity:50.68%   altitude:164.65m
temperature:29.46*C     pressure:993.63hPa  humidity:50.68%   altitude:164.65m
temperature:29.46*C     pressure:993.69hPa  humidity:50.67%   altitude:164.16m
```

☐ 自动滚屏　没有结束符　115200 波特率　Clear output

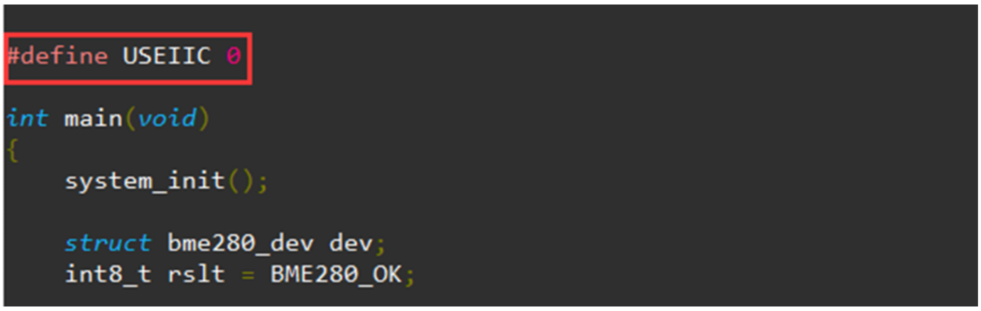
Note: if there are not any data outputted after running the demo code or get wrong data, please check the hardware connection and address used

WORKIGN WITH STM32

1. Open STM32 project which is under STM32 directory.

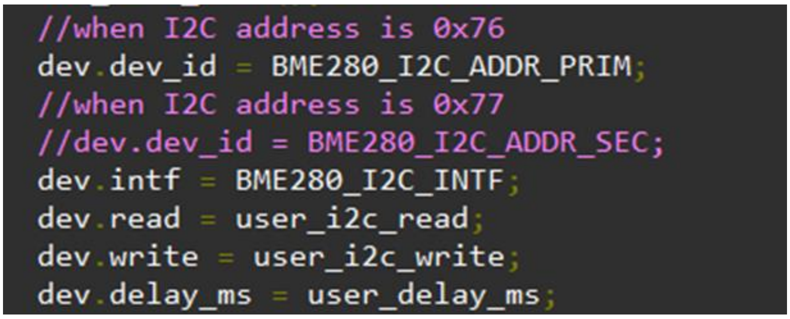
2. Connect BME280 to STM32 board according to Interfaces

3. Default interface is I2C with device address 0x77. If you change to SPI, you need to change the USEIIR define to 0 of main.c:



```
#define USEIIC 0

int main(void)
{
    system_init();

    struct bme280_dev dev;
    int8_t rslt = BME280_OK;
```

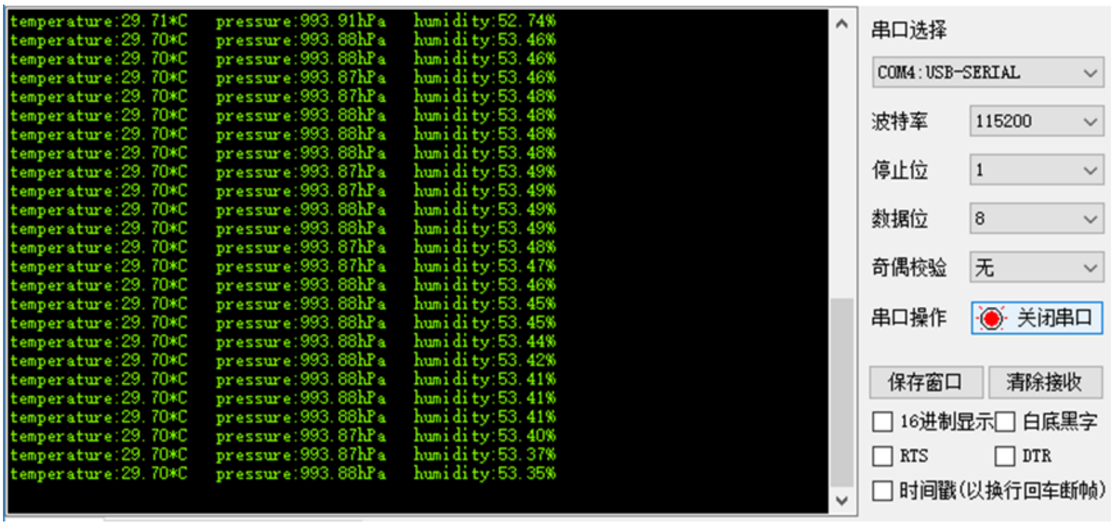
- If you want to use I2C and change its address, you can connect ADDR to GND and discommend statement dev.dev_id = BME280_I2C_ADDR_SEC:



```
//when I2C address is 0x76
dev.dev_id = BME280_I2C_ADDR_PRIM;
//when I2C address is 0x77
//dev.dev_id = BME280_I2C_ADDR_SEC;
dev.intf = BME280_I2C_INTF;
dev.read = user_i2c_read;
dev.write = user_i2c_write;
dev.delay_ms = user_delay_ms;
```

4. Compile and download the demo code to STM32 board, the examples we provide is based on STM32F103RBT6, Use UART2 to print sensor data. You can connect UART to PC by TTL to UART module, then open Serial assistance software on PC,

default 115200 8N1:



```
temperature:29.71*C   pressure:993.91hPa   humidity:52.74%
temperature:29.70*C   pressure:993.88hPa   humidity:53.46%
temperature:29.70*C   pressure:993.88hPa   humidity:53.46%
temperature:29.70*C   pressure:993.87hPa   humidity:53.46%
temperature:29.70*C   pressure:993.87hPa   humidity:53.48%
temperature:29.70*C   pressure:993.88hPa   humidity:53.48%
temperature:29.70*C   pressure:993.88hPa   humidity:53.48%
temperature:29.70*C   pressure:993.88hPa   humidity:53.48%
temperature:29.70*C   pressure:993.87hPa   humidity:53.49%
temperature:29.70*C   pressure:993.87hPa   humidity:53.49%
temperature:29.70*C   pressure:993.88hPa   humidity:53.49%
temperature:29.70*C   pressure:993.88hPa   humidity:53.49%
temperature:29.70*C   pressure:993.87hPa   humidity:53.48%
temperature:29.70*C   pressure:993.87hPa   humidity:53.47%
temperature:29.70*C   pressure:993.88hPa   humidity:53.46%
temperature:29.70*C   pressure:993.88hPa   humidity:53.45%
temperature:29.70*C   pressure:993.88hPa   humidity:53.45%
temperature:29.70*C   pressure:993.88hPa   humidity:53.44%
temperature:29.70*C   pressure:993.88hPa   humidity:53.42%
temperature:29.70*C   pressure:993.88hPa   humidity:53.41%
temperature:29.70*C   pressure:993.88hPa   humidity:53.41%
temperature:29.70*C   pressure:993.88hPa   humidity:53.41%
temperature:29.70*C   pressure:993.87hPa   humidity:53.40%
temperature:29.70*C   pressure:993.87hPa   humidity:53.37%
temperature:29.70*C   pressure:993.88hPa   humidity:53.35%
```

串口选择
COM4:USB-SERIAL
波特率 115200
停止位 1
数据位 8
奇偶校验 无
串口操作 关闭串口

Note: if there are not any data outputted after running the demo code or get wrong data, please check the hardware connection and address used.

CODE ANALYSIS

Libraries used In demo code is official library Bosch Sensortec:

https://github.com/BoschSensortec/BME280_driver

For SPI, its initial code:

```
struct bme280_dev dev;
int8_t rslt = BME280_OK;

/* Sensor_0 interface over SPI with native chip select line */
dev.dev_id = 0;
dev.intf = BME280_SPI_INTF;
dev.read = user_spi_read;
dev.write = user_spi_write;
dev.delay_ms = user_delay_ms;

rslt = bme280_init(&dev);
```

For I2C, its initial code:

```
struct bme280_dev dev;
int8_t rslt = BME280_OK;

dev.dev_id = BME280_I2C_ADDR_PRIM;
dev.intf = BME280_I2C_INTF;
dev.read = user_i2c_read;
dev.write = user_i2c_write;
dev.delay_ms = user_delay_ms;

rslt = bme280_init(&dev);
```

bme280_dev is BME280 device structure provided by official library, could be used to initialize and obtain data. For different platform we should realize the functions below:

```
user_i2c_read()
user_i2c_write()
user_spi_read()
user_spi_write()
user_delay_ms()
```

And then transfer function pointers of these functions to bem280_dev structure.

The function to read BME280 data is:

```
int8_t stream_sensor_data_forced_mode(struct bme280_dev *dev)
int8_t stream_sensor_data_normal_mode(struct bme280_dev *dev)
```

Print function:

```
void print_sensor_data(struct bme280_data *comp_data)
```

The read/write function of I2C and SPI:

```
void user_delay_ms(uint32_t period)
{
    /*
     * Return control or wait,
     * for a period amount of milliseconds
     */
}

int8_t user_spi_read(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data,
uint16_t len)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */

    /*
     * The parameter dev_id can be used as a variable to select which Chip
Select pin has
     * to be set low to activate the relevant device on the SPI bus
     */

    /*
     * Data on the bus should be like
     * |----------------+---------------------+-------------|
     * | MOSI           | MISO                | Chip Select |
     * |----------------+---------------------|-------------|
     * | (don't care)   | (don't care)        | HIGH        |
     * | (reg_addr)     | (don't care)        | LOW         |
     * | (don't care)   | (reg_data[0])       | LOW         |
     * | (....)         | (....)              | LOW         |
     * | (don't care)   | (reg_data[len - 1]) | LOW         |
     * | (don't care)   | (don't care)        | HIGH        |
     * |----------------+---------------------|-------------|
     */

    return rslt;
}

int8_t user_spi_write(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data,
uint16_t len)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */

    /*
     * The parameter dev_id can be used as a variable to select which Chip
Select pin has
     * to be set low to activate the relevant device on the SPI bus
     */

    /*
     * Data on the bus should be like
     * |---------------------+--------------+-------------|
     * | MOSI                | MISO         | Chip Select |
     * |---------------------+--------------|-------------|
     * | (don't care)        | (don't care) | HIGH        |
     * | (reg_addr)          | (don't care) | LOW         |
     * | (reg_data[0])       | (don't care) | LOW         |
     * | (....)              | (....)       | LOW         |
     * | (reg_data[len - 1]) | (don't care) | LOW         |
     * | (don't care)        | (don't care) | HIGH        |
     * |---------------------+--------------|-------------|
     */

    return rslt;
}

int8_t user_i2c_read(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data,
uint16_t len)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */

    /*
     * The parameter dev_id can be used as a variable to store the I2C
address of the device
     */
    /*
     * Data on the bus should be like
     * |------------+---------------------|
     * | I2C action | Data                |
     * |------------+---------------------|
     * | Start      | -                   |
     * | Write      | (reg_addr)          |
     * | Stop       | -                   |
     * | Start      | -                   |
     * | Read       | (reg_data[0])       |
     * | Read       | (....)              |
     * | Read       | (reg_data[len - 1]) |
     * | Stop       | -                   |
     * |------------+---------------------|
     */

    return rslt;
}

int8_t user_i2c_write(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data,
uint16_t len)
{
    int8_t rslt = 0; /* Return 0 for Success, non-zero for failure */
    /*
     * The parameter dev_id can be used as a variable to store the I2C
address of the device
     */
    /*
     * Data on the bus should be like
     * |------------+---------------------|
     * | I2C action | Data                |
     * |------------+---------------------|
     * | Start      | -                   |
     * | Write      | (reg_addr)          |
     * | Write      | (reg_data[0])       |
     * | Write      | (....)              |
     * | Write      | (reg_data[len - 1]) |
     * | Stop       | -                   |
     * |------------+---------------------|
     */
    return rslt;
}
```

The basic flows to read BME280 data are:

Step1: Initialize for OS and peripheral

Step2: Realize Read, Write and Delay functions (I2C and SPI), and assignment the functions pointer to *bme280_dev* structure as publics, then transfer the pointer of structure to initialization function *int8_t bme280_init(struct bme280_dev *dev).* Initialize BME280 device.

Step3: Calling *int8_t stream_sensor_data_forced_mode(struct bme280_dev *dev)* or *int8_t stream_sensor_data_normal_mode(struct bme280_dev *dev)* to get data of BME280 sensor and print them to console or PC.