BME280 Environmental Sensor
用户手册

产品概述

我是一款环境传感器，可感知环境温度、湿度和大气压强，支持 I2C 和 SPI 接口，兼容 3.3V/5V 电平，小尺寸，低功耗，高精度和稳定性，适用于环境监测、天气预测、海拔高度监测和物联网应用场景。

特点

● 支持 I2C 接口通信，可通过 I/O 口或焊点设置 I2C 从机地址

● 支持 SPI 接口通信，默认为 I2C 接口，可通过 I/O 切换成 SPI

● 板载电平转换电路，可兼容 3.3V/5V 的工作电平

● 提供完善的配套资料手册(Raspberry/Arduino/STM32 示例程序和用户手册等)

产品参数

工作电压：	5V/3.3V
通信接口：	I2C/SPI
温度范围：	-40~85°C (分辨率 0.01°C，误差±1°C)
湿度范围：	0~100%RH (分辨率 0.008%RH，±3% RH)
压力范围：	300~1100 hPa (分辨率 0.18Pa，误差±1 hPa)
产品尺寸：	27mmx20mm
过孔直径：	2.0mm

版本：V1.0.0，日期：2018 年 08 月 14 日

接口定义

I2C 接口

功能引脚	Arduino 接口	STM32 接口	RASPBERRY	描述
VCC	3.3V/5V	3.3V /5V	3.3V /5V	电源正
GND	GND	GND	GND	电源地
SDA	A4	PB7	SDA	I2C 数据线
SCL	A5	PB6	SCL	I2C 时钟线
ADDR	NC/GND	NC/GND	NC/GND	地址片选(默认为高电平)： 为高电平时，地址：0x77 为低电平时，地址为：0x76
CS	NC	NC	NC	NC

SPI 接口

功能引脚	Arduino 接口	STM32 接口	RASPBERRY	
VCC	3.3V /5V	3.3V /5V	3.3V /5V	3.3V 电源正
GND	GND	GND	GND	电源地
MOSI	D11	PA7	MOSI	SPI 数据输入
SCK	D13	PA5	SCK	SPI 时钟输入
MISO	D12	PA6	MISO	SPI 数据输出
CS	D10	PB6	27 (wPI)	SPI 片选，低电平有效

用于树莓派

安装必要的函数库

需要安装必要的 WiringPi 函数库，否则以下的示例程序可能无法正常工作。安装方法详见：

http://www.waveshare.net/wiki/Pioneer600_Datasheets

在官网上找到对应产品，在产品资料打开下载路径，在 wiki 中下载示例程序：

文档

- 用户手册
- 原理图

程序

- 示例程序

得到解压包，解压得到如下：

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	

此电脑 > ... > BME280-Environmental-Sensor-Demo-Code >

将 Raspberry 文件夹拷至树莓派。

前置工作及演示

前置工作

执行如下命令进行树莓派配置：

sudo raspi-config

选择 Interfacing Options -> I2C -> yes 启动 I2C 内核驱动

选择 Interfacing Options -> SPI -> yes 启动 SPI 内核驱动

保存退出后，重启树莓派：

sudo reboot

重启后，运行命令查看，I2C，SPI 模块是否已启动：

lsmod

将会有如下的打印信息：

版本：V1.0.0，日期：2018 年 08 月 14 日

如果显示 i2c_bcm2835 和 spi_bcm2835 则表示 I2C，SPI 模块已启动。

将 BME280 模块按照前述 I2C 总线接口说明连接至树莓派。

BME280 模块的默认 I2C 器件地址是 0x77，若将 ADDR 接地则器件地址更变为 0x76。

安装 i2c-tools 工具进行确认：

sudo apt-get install i2c-tools

查询已连接的 I2C 设备

i2cdetect -y 1

将会有如下打印信息：

若显示 77 则表示 BME280 模块成功连接至树莓派成功。

若将 ADDR 连接至 GND 则打印出 76：

版本：V1.0.0，日期：2018 年 08 月 14 日

注意：以上测试确保 I2C 总线上没有其它地址和该器件地址重合的设备。

如果以上测试成功则 I2C 模块加载成功，同时 BME280 模块成功连接至树莓派。

同时，BME280 模块支持 SPI 驱动，可参考 SPI 接口说明部分将 BME280 连接至树莓派。

演示

成功将 BME280 模块连接至树莓派后：

如果采用 I2C 驱动：则先确定 I2C 器件地址，BME280 模块默认 I2C 器件地址为 0X77，若将 ADDR 接地(或用 0 欧姆电阻将焊桥连接)，则其 I2C 器件地址变更为 0X76。

打开 main.c 文件：

进入到 BME280-Environmental-Sensor-Demo-Code 路径下：

cd BME280-Environmental-Sensor-Demo-Code

打开 main.c 文件：

vim main.c

确保 main.c 中的 USEIIC 的宏定义为 1，以采用 I2C 驱动。

版本：V1.0.0，日期：2018 年 08 月 14 日

同时检查 main.c 中的 I2C 器件地址，确保和当前 BME280 模块器件地址一致(默认 I2C 器件地址为 0x77，若将 ADDR 接地则其器件地址为 0x76)：

```
208 {
209    struct bme280_dev dev;
210    int8_t rslt = BME280_OK;
211
212    if ((fd = open(IIC_Dev, O_RDWR)) < 0) {
213       printf("Failed to open the i2c bus %s", argv[1]);
214       exit(1);
215    }
216    if (ioctl(fd, I2C_SLAVE, 0x77) < 0) {
217       printf("Failed to acquire bus access and/or talk to slave.\n");
218       exit(1);
219    }
220    //dev.dev_id = BME280_I2C_ADDR_PRIM;//0x76
221    dev.dev_id = BME280_I2C_ADDR_SEC; //0x77
222    dev.intf = BME280_I2C_INTF;
223    dev.read = user_i2c_read;
224    dev.write = user_i2c_write;
225    dev.delay_ms = user_delay_ms;
226
227    rslt = bme280_init(&dev);
228    printf("\r\n BME280 Init Result is:%d \r\n",rslt);
229    //stream_sensor_data_forced_mode(&dev);
230    stream_sensor_data_normal_mode(&dev);
```

如果采用 SPI 驱动：则将 BME280 模块按照接口说明中的 SPI 总线接线方式进行接线，并将 main.c 文件中的 USEIIC 宏定义改为 0。

```
10 #include "bme280.h"
11 #include <stdio.h>
12 #include <unistd.h>
13 #include <wiringPi.h>
14 #include <wiringPiSPI.h>
15
16 //Raspberry 3B+ platform's default SPI channel
17 #define channel 0
18
19 //Default write it to the register in one time
20 #define USESPISINGLEREADWRITE 0
21
22 //This definition you use I2C or SPI to drive the bme280
23 //When it is 1 means use I2C interface, When it is 0,use SPI interface
24 #define USEIIC 0
25
26
27 #if(USEIIC)
28 #include <string.h>
29 #include <stdlib.h>
30 #include <linux/i2c-dev.h>
31 #include <sys/ioctl.h>
```

保存并退出编辑，然后重新编译：

sudo make clean

sudo make

运行：

sudo ./bme280

将显示如下数据：

从左至右分别显示了 BME280 测得的温度(摄氏度)，大气压(百帕斯卡)，相对湿(%RH)。

若未成功显示数据，或数据显示不正常请检查连线，通信方式，以及器件地址是否有误。

用于 Arduino

将下载的示例程序的压缩包解压后，将 Arduino 文件夹下的 BME280-Arduino-Library 拷贝至 Arduino 第三方库目录下.

之后重启 ArduinoIDE，进入并打开文件->实例->BME280_Libreay->bme280test

按照接口说明中 Arduino 接口部分接线。

默认采用 I2C 驱动 BME280 模块，默认 I2C 器件地址为 0X77。

如需采用 SPI 驱动 BME280 模块，请将 bme280test.ino 中的宏定义 USEIIC 改为 0：

```
文件 编辑 项目 工具 帮助

bme280test§

 7 //if you need to read altitude,you need to know the sea level pressure
 8 #define SEALEVELPRESSURE_HPA (1013.25)
 9
10 //This Macro definition decide whether you use I2C or SPI
11 //When USEIIC is 1 means use I2C interface, When it is 0,use SPI interface
12 #define USEIIC 0
13
14 /*
15 This Demo is tested on UNO PLUS
16 SPI:
17 SPI_SCK:  D13
18 SPI_MISO: D12
19 SPI_MOSI: D11
20 SPI_CS:   D10
```

如需更改 I2C 器件的地址为 0X76，则将 ADDR 引脚接至 GND(或用 0 欧姆电阻将焊桥连接)，同
时将 Adafruit.h 中的 BME280_ADDRESS 器件地址改为 0X76：

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

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

如需获取准测量的准确海拔，还需测得当地海平面的大气压，并修改 SEALEVELPRESSURE_HPA
宏定义：

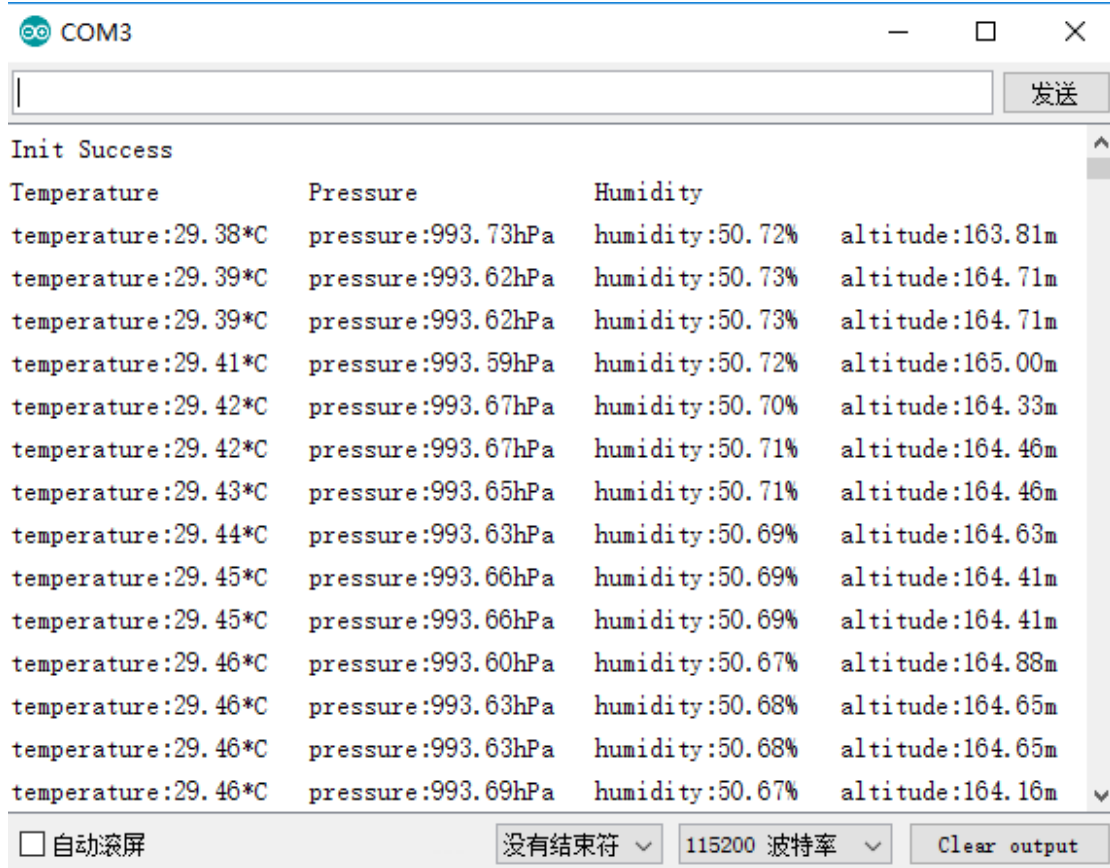```
 1 #include <Wire.h>
 2 #include <SPI.h>
 3
 4 #include <Adafruit_Sensor.h>
 5 #include <Adafruit_BME280.h>
 6
 7 //if you need to read altitude,you need to know the sea level pressure
 8 #define SEALEVELPRESSURE_HPA (1013.25)
 9
10 //This Macro definition decide whether you use I2C or SPI
11 //When USEIIC is 1 means use I2C interface, When it is 0,use SPI interface
12 #define USEIIC 1
13
14 /*
15 This Demo is tested on UNO PLUS
16 SPI:
17 SPI_SCK: D13
18 SPI_MISO: D12
19 SPI_MOSI: D11
```

版本：V1.0.0，日期：2018 年 08 月 14 日

在正确接线，确定通信方式以及器件地址之后，编译，下载到 Arduino。

打开：工具 -> 串口监视器，选择波特率为 115200，可得如下信息



其中从左至右分别显示了 BME280 传感器测得的温度(摄氏度)，大气压(百帕斯卡)，相对湿度 (%RH)，海拔(m)。

若未成功显示数据，或数据显示不正常请检查连线，通信方式，以及器件地址是否有误。

## 用于 STM32

将下载的示例程序的压缩包解压后，打开 STM32 文件夹下的 STM32-STM32_BME280->USR 路径 下的工程文件：

按照接口说明中 STM32 接口部分接线。

默认采用 I2C 驱动 BME280 模块，默认 I2C 器件地址为 0X77。

如需采用 SPI 驱动 BME280 模块，请将 main.c 中的宏定义 USEIIC 改为 0：

```
#define USEIIC 0

int main(void)
{
    system_init();

    struct bme280_dev dev;
    int8_t rslt = BME280_OK;
```
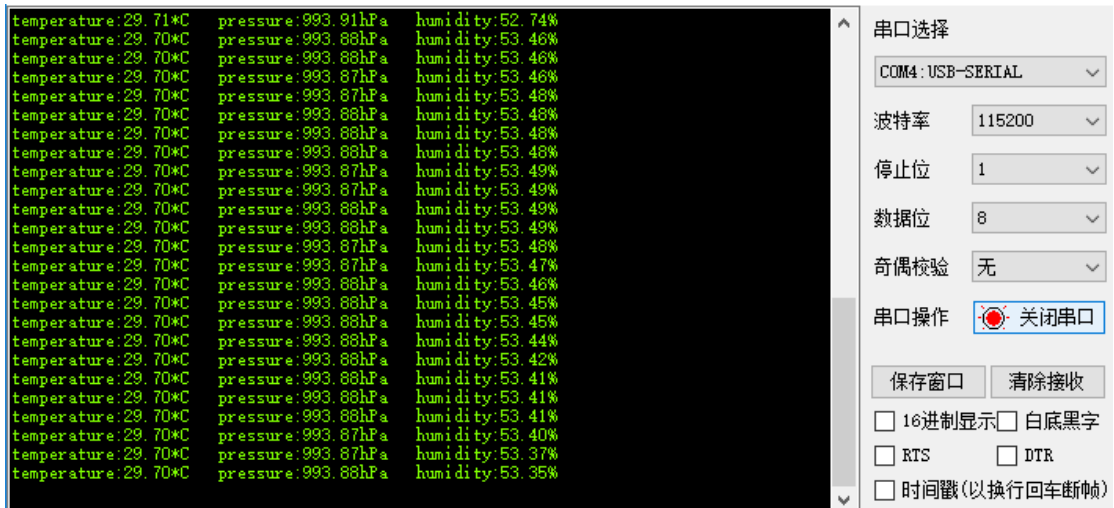
如需更改 I2C 器件的地址为 0X76，则将 ADDR 引脚接至 GND(或用 0 欧姆电阻将焊桥连接),并将 dev.dev_id = BME280_I2C_ADDR_SEC;注释，取消 dev.dev_id = BME280_I2C_ADDR_PRIM;注释：

```
//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;
```

编译，下载，本次采用的芯片是 STM32F103RBT6，采用 USART2 输出获得的传感器数据。

打开串口调试助手，选择对应的 COM 口，设置波特率为 115200，数据位 8 位，停止位 1 位，无奇偶校验位，可得如下数据：

其中从左至右分别显示了 BME280 传感器测得的温度(摄氏度)，大气压(百帕斯卡)，相对湿度(%RH)。

若未成功显示数据，或数据显示不正常请检查连线，通信方式，以及器件地址是否有误。

代码分析

例程主要采用了官方(Bosch Sensortec)提供的库:

https://github.com/BoschSensortec/BME280_driver

针对不同平台实现其底层函数，供上层调用。

采用 SPI 驱动 BME280 的初始化部分为：

```c
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);
```

采用 I2C 驱动 BME280 初始化部分：

```c
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);
```

其中 bme280_dev 为官方库中给定的 BME280 设备结构体，用于初始化以及获取数据用，需要针对不同的平台实现以下函数：

版本：V1.0.0，日期：2018 年 08 月 14 日

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

并将该函数的函数指针传递给结构体 bme280_dev。

读取 BME280 数据的函数为：

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

并且以上函数均调用了打印函数：

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

不同平台的延时函数，I2C 读，I2C 写，SPI 读，SPI 写的实现思路为：

```
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;
}
```

```c
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;
}
```

综上，基于官方库，针对不同平台，获取 BME280 数据的基本流程为：

第一步：不同平台的系统及外设初始化。

第二步：实现不同平台的 I2C 读，I2C 写，SPI 读，SPI 写，延时函数，并将函数指针赋值给 bme280_dev 结构体成员变量，将该结构体指针传递给初始化函数 int8_t bme280_init(struct bme280_dev *dev)，初始化 BME280 设备。

第三步：调用 int8_t stream_sensor_data_forced_mode(struct bme280_dev *dev)或

int8_t stream_sensor_data_normal_mode(struct bme280_dev *dev)函数获取 BME280 传感器数据并打印到上位机或控制台。

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Multiple Function Sensor Development Tools *category:*

Click to view products by Waveshare *manufacturer:*

Other Similar products are found below :

RD-KL25-AGMP01 PTC-04-DB-ACT FRDM-K22F-SA9500 EV_ICM-20649 MULTI-SENSE-GEVB FRDM-STBC-SA9500 BRKT-STBC-SA9500 STEVAL-MKIT01V1 EVAL-CN0411-ARDZ SEN0004 KT-TVOC-200-TB200A KT-NmHc-200-TB200A SEN0344 PIM520 PIM518 PIM519 PIM510 103030375 ZSSC4132KIT ADIS16505-3/PCBZ SEN-16794 PIM502 SEN0359 4829 EV26Q64A EVAL-AD7746RDZ EVAL-AD7746HDZ AS7022-EVALKIT RTK0ESXB10C00001BJ MAX30134EVSYS# EV-CBM-PIONEER1-1Z EVAL-ADPD188BIZ-S2 EVAL-ADCM-1 EVAL-CN0507-ARDZ SI118X-KIT ALTEHTG2SMIP EVAL-CN0533-EBZ MIKROE-4305 MAX30101WING# MIKROE-4192 MIKROE-4049 OB1203SD-U-EVK OB1203SD-BT-EVK MIKROE-4037 101990644 MIKROE-4267 MIKROE-4265 MIKROE-4330 ARG-LDKT EVAL-CN0503-ARDZ