TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA CƠ KHÍ CHẾ TẠO MÁY

HỆ THỐNG NHÚNG

HƯỚNG DẪN SỬ DỤNG BME280 – 3.3V

GVHD : TS. BÙI HÀ ĐỨC

SVTH : Lâm Gia Hào 21146222

Bùi Phước Huy 21146233

Phan Nhật Tân 21146310

MỤC LỤC

CHƯƠNG 1 : GIỚI THIỆU VỀ BME280:

1.1 KHÁI NIỆM

BME280 là cảm biến độ ẩm, áp suất và nhiệt độ kỹ thuật số kết hợp. Kích thước nhỏ, tiêu thụ điện năng thấp, độ chính xác và ổn định cao cho phép thực hiện giám sát môi trường, dự báo thời tiết và các ứng dụng IoT

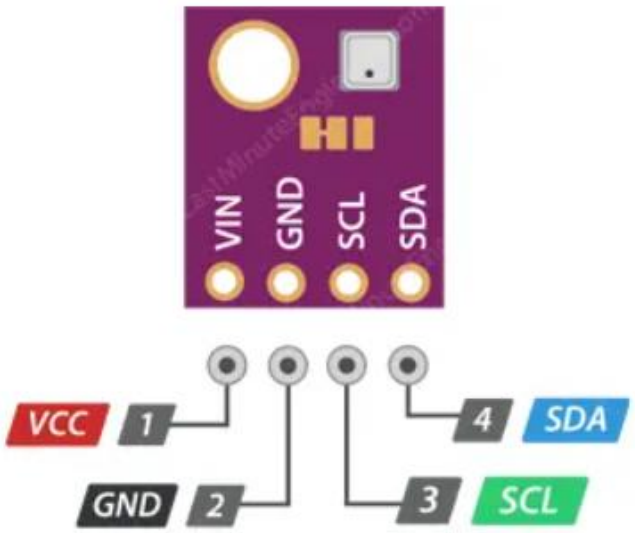
1.2 TÍNH NĂNG CỦA BME280:

- Hỗ trợ giao tiếp I2C, địa chỉ thiết bị I2C có thể được đặt bằng cách thay đổi I/O.
- Hỗ trợ giao diện SPI, I2C mặc định, bạn có thể thay đổi thành SPI bằng cách thay đổi I/O.
- Mạch chuyển đổi mức tích hợp, tương thích với 3.3V / 5V
- Hỗ trợ thư viện và hướng dẫn sử dụng cho một số dòng vi điều khiển phổ biến (Raspberry Pi / Arduino / STM32).

1.3 THÔNG SỐ KỸ THUẬT:

- Điện áp hoạt động: 5V / 3.3V
- Giao thức truyền: I2C/SPI
- Phạm vi nhiệt độ: -40 ~ 85 °C (Độ phân giải 0,1 °C, dung sai ±1 °C)
- Phạm vi độ ẩm: 0 ~ 100% rh (Độ phân giải 0,008% RH, dung sai ±1 °C)
- Phạm vi áp suất: 300 ~ 1100hPa (Độ phân giải 0,18Pa, dung sai ±1hPa)

1.4 SƠ ĐỒ CHÂN CỦA MODULE BME280:



- **VCC:** Chân nguồn (điện áp khuyến cáo từ 3.3 -5V)
- **GND:** Chân nối đất
- **SDA:** Chân dữ liệu
- **SCL:** Chân xung Clock

1.5 SƠ ĐỒ ĐẤU DÂY GIỮA MODULE VỚI RASPBERRY:

Module BME280	Raspberry
VCC	Pin 1
GND	Pin 6
SDA	Pin 3
SCL	Pin 5

1.6 SƠ LƯỢC VỀ GIAO THỨC I2C:

I2C (Inter-Integrated Circuit) là một giao thức truyền thông nối tiếp 2 dây được phát triển bởi Philips (nay là NXP Semiconductors) vào những năm 1980. Nó cho phép

truyền dữ liệu giữa các thiết bị điện tử như vi điều khiển, cảm biến, …thông qua hai dây **SDA** và **SCL**.

- SDA (Serial Data): đường truyền theo hai chiều giữa master và slave để gửi và nhận dữ liệu
- SCL (Serial Clock): đường truyền mang tín hiệu xung clock, chỉ do Master phát đi (thông thường ở 100kHz và 400kHz. Mức cao nhất là 1Mhz và 3.4MHz)

CHƯƠNG 2 : MÔ TẢ MỘT SỐ THANH GHI CẤU HÌNH:

2.1 THANH GHI 0XD0 “ID”

Là thanh ghi chứa số định danh của BME280 với giá trị định danh là 0x60, giá trị này có thể đọc ngay khi khởi động.

2.2 THANH GHI 0XE0 “RESET”

Là thanh ghi dùng để thực hiện khởi động lại phần mềm module, có kích thước 1 byte. Ghi giá trị 0xB6 vào thanh ghi ‘reset’ để thực hiện việc khởi động lại.

2.3 REGISTER 0XF2 “CTRL_HUM”

Là thanh ghi dùng để thiết lập các tham số cấu hình cho độ ẩm. Cụ thể là nó xác định chế độ oversampling cho độ ẩm. Oversampling là quá trình lấy mẫu nhiều lần và tính toán giá trị trung bình để tăng độ chính xác của dữ liệu đo được.

Thanh ghi gồm 8 bit nhưng chỉ cần quan tâm bits [2:0]

Bits[2:0]	**Oversampling (osrs_h)**	**Mô tả**
000	Skipped (osrs_h = 0)	Bỏ qua
001	×1 (osrs_h = 1)	Lấy mẫu một lần
010	×2 (osrs_h = 2)	Lấy mẫu hai lần
011	×4 (osrs_h = 3)	Lấy mẫu bốn lần
100	×8 (osrs_h = 4)	Lấy mẫu tám lần

101	×16 (osrs_h = 5)	Lấy mẫu mười sáu lần

2.4 REGISTER 0XF3 “STATUS”

Là thanh ghi dùng để thể hiện trạng thái của thiết bị.

Thanh ghi gồm 8 bit nhưng chỉ cần quan tâm bit 3 và bit 0.

Bit	Tên	Mô tả
3	measuring[0]	Luôn ở giá trị 1 và chỉ về giá trị 0 khi quá trình đo đã hoàn tất
0	im_update[0]	Luôn ở giá trị 1 và chỉ về giá trị 0 khi việc cập nhật thành công

2.5 REGISTER 0XF4 “CTRL_MEAS”

Là thanh ghi điều khiển chế độ đo và các thông số oversampling cho nhiệt độ và áp suất.

Đây là một thanh ghi 8 bit với các bit đều được sử dụng để điều khiển các chức năng cụ thể.

Bits	**Tên**	**Mô tả**
Bit 7, 6, 5	osrs_t[2:0]	Oversampling cho nhiệt độ
Bit 4, 3, 2	osrs_p[2:0]	Oversampling cho áp suất
Bit 1, 0	mode [1:0]	Chế độ đo

Các giá trị đo cho các bits [7:5] (osrs_t):

Bits[7:5]	**Oversampling (osrs_t)**	**Mô tả**
000	Skipped (osrs_t = 0)	Bỏ qua đo nhiệt độ
001	×1 (osrs_t = 1)	Lấy mẫu một lần
010	×2 (osrs_t = 2)	Lấy mẫu hai lần
011	×4 (osrs_t = 3)	Lấy mẫu bốn lần
100	×8 (osrs_t = 4)	Lấy mẫu tám lần
101	×16 (osrs_t = 5)	Lấy mẫu mười sáu lần

Các giá trị đo cho các bits [4:2] (osrs_p):

Bits[4:2]	**Oversampling (osrs_p)**	**Mô tả**
000	Skipped (osrs_p = 0)	Bỏ qua đo áp suất
001	×1 (osrs_p = 1)	Lấy mẫu một lần
010	×2 (osrs_p = 2)	Lấy mẫu hai lần
011	×4 (osrs_p = 3)	Lấy mẫu bốn lần
100	×8 (osrs_p = 4)	Lấy mẫu tám lần
101	×16 (osrs_p = 5)	Lấy mẫu mười sáu lần

Module BME280 có 3 chế độ đo là: Sleep, Forced và Normal

Bits[1:0]	**Chế độ đo**	**Mô tả**
00	Sleep mode	Cảm biến ở chế độ ngủ
01 and 10	Forced mode	Cảm biến đo 1 lần, sau đó về chế độ ngủ
11	Normal mode	Cảm biến đo liên tục

2.6 REGISTER 0XF5 "CONFIG"

Là thanh ghi dùng để thiết lập thời gian chờ của các phép đo trong chế độ Normal, hệ số bộ lọc IIR, kích hoạt giao thức SPI 3-wire.

Thanh ghi gồm 8 bits với cấu trúc như sau:

Register 0xF5 *"config"*

Thanh ghi 0xF5 *"config"*	**Tên**	**Miêu tả**
Bit [7:5]	t_sb[2:0]	Thời gian chờ các phép đo trong chế độ normal
Bit [4:2]	filter[2:0]	Hệ số bộ lọc
Bit 0	spi3w_en[0]	Kích hoạt hoặc vô hiệu giao thức SPI 3-wire

***t_sb* [2:0]**	**tstandby [ms]**
000	0.5
001	62.5
010	125
011	250
100	500
101	1000

110	10
111	20

***filter* [2:0]**	**Hệ số Filter**
000	Filter off
001	2
010	4
011	8
100, others	16

2.7 BẢNG TỔNG QUÁT CÁC THANH GHI:

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]				filter[2:0]			spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]				osrs_p[2:0]		mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1…0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88…0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

CHƯƠNG 3 : THƯ VIỆN NGƯỜI DÙNG

3.1 MỘT SỐ HÀM CÓ SẴN TRÊN DRIVER:

- ***BME280_IOCTL_READ_T***
  Hàm có chức năng đọc giá trị nhiệt độ của cảm biến ở đơn vị độ C (℃).
  Hàm sẽ trả về giá trị nhiệt độ ở kiểu dữ liệu int32_t và cần phải chia cho 100 để đạt được giá trị thực tế.
  Ví dụ:

```
if (ioctl(fd, BME280_IOCTL_READ_T, &temp_int) < 0) {
        perror("Failed to read temparature data");
        close(fd);
        return errno;
}
printf("Temparature: %.2f°C\n", temp_int/100.0);
```

- ***BME280_IOCTL_READ_P***
  Hàm có chức năng đọc giá trị áp suất của cảm biến ở đơn vị Pa.
  Hàm sẽ trả về giá trị nhiệt độ ở kiểu dữ liệu int32_t
  Ví dụ:

```
if (ioctl(fd, BME280_IOCTL_READ_P, &press_int) < 0) {
        perror("Failed to read pressure data");
        close(fd);
        return errno;
}
printf("Pressure: %dPa\n", press_int);
```

- ***BME280_IOCTL_READ_H***
  Hàm có chức năng đọc giá trị độ ẩm của cảm biến ở đơn vị %RH.
  Hàm sẽ trả về giá trị độ ẩm ở kiểu dữ liệu int32_t và cần phải chia cho 1024 để được giá trị độ ẩm chính xác.

Ví dụ:

```
if (ioctl(fd, BME280_IOCTL_READ_H, &hum_int) < 0) {
        perror("Failed to read humidity data");
        close(fd);
        return errno;
}
printf("Humidity: %dRH\n", hum_int/1024.0);
```

- ***BME280_IOCTL_INIT_FORCED_MODE***
  Hàm có chức năng thiết lập cảm biến đo ở chế độ Forced, với oversampling là x1. Trả về 0 khi việc khởi tạo thành công
  Ví dụ:

```
if (ioctl(fd, BME280_IOCTL_INIT_FORCED_MODE) < 0) {
        perror("Failed to initialize the sensor");
        close(fd);
        return errno;
}
printf("Sensor initialized successfully\n");
```

- ***BME280_IOCTL_INIT_NORMAL_MODE***
  Hàm có chức năng thiết lập cảm biến đo ở chế độ Normal, với oversampling là x1. Trả về 0 khi việc khởi tạo thành công
  Ví dụ:

```
if (ioctl(fd, BME280_IOCTL_INIT_NORMAL_MODE) < 0) {
        perror("Failed to initialize the sensor");
        close(fd);
        return errno;
}
printf("Sensor initialized successfully\n");
```

- ***BME280_IOCTL_CHECK_ADDR***
  Hàm có chức năng kiểm tra xem giá trị của thanh ghi 0xD0 “id” có đúng với tài liệu của nhà sản xuất hay không.

```
id = ioctl(fd, BME280_IOCTL_CHECK_ADDR);
    if (id < 0) {
        perror("Unable to configure i2c identification");
        close(fd);
}
```

- ***BME280_IOCTL_CHECK_STATUS***

  Hàm có chức năng kiểm tra xem dữ liệu đã sẵn sàng được đọc hay chưa. Hàm sẽ trả về kết quả 0 khi sẵn sàng.

  Ví dụ:

```
status = ioctl(fd, BME280_IOCTL_CHECK_STATUS);
        if (status < 0) {
            perror("Data is not ready");
            close(fd);
       }
```

3.2 HƯỚNG DẪN SỬ DỤNG:

Để sử dụng được driver của module BME280, người dùng cần dẫn đúng tên của driver từ thư mục /dev. Sau đây là các bước để dùng driver từ con số 0:
Bước 1: Định nghĩa đường dẫn vào driver BME280:

```
#define DEVICE_PATH "/dev/bme280"
```

Bước 2: Định nghĩa các lệnh ioctl của driver BME280:

```
#define BME280_IOCTL_MAGIC 'm'
#define BME280_IOCTL_READ_T _IOR(BME280_IOCTL_MAGIC, 1,
uint32_t)
#define BME280_IOCTL_READ_P _IOR(BME280_IOCTL_MAGIC, 2,
uint32_t)
#define BME280_IOCTL_READ_H _IOR(BME280_IOCTL_MAGIC, 3,
uint32_t)
#define BME280_IOCTL_INIT_FORCED_MODE
_IO(BME280_IOCTL_MAGIC, 4)
#define BME280_IOCTL_INIT_NORMAL_MODE
_IO(BME280_IOCTL_MAGIC, 5)
#define BME280_IOCTL_CHECK_ADDR _IOR(BME280_IOCTL_MAGIC, 6,
int)
#define BME280_IOCTL_CHECK_STATUS _IOR(BME280_IOCTL_MAGIC,
7, int)
```

Bước 3: Sau khi đã định nghĩa một số macro quan trọng của driver, trong hàm main(), người dùng thực hiện mở driver.

```
int fd;
fd = open(DEVICE_PATH, O_RDWR);
    if (fd < 0) {
        perror("Failed to open the device");
        return errno;
    }
printf("Successfully open device\n");
```

Bước 4: Người dùng thực hiện gọi các hàm mà mình mong muốn vào hàm main như: kiểm tra id, cấu hình chế độ, …
Bước 5: Sau khi đã thực hiện việc đọc giá trị nhiệt độ, áp suất, độ ẩm. Người dùng nhập vào hàm close() để đóng chương trình.

```
close(fd);
```

3.3 MỘT SỐ VÍ DỤ CƠ BẢN:

Dưới đây là một đoạn code hoàn chỉnh để đọc giá trị áp suất, nhiệt độ, độ ẩm ở chế độ Forced Mode.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <errno.h> // Include errno header
#include <i2c/smbus.h>
#include <math.h>
#include <stdint.h>
#include <linux/i2c-dev.h>

#define DEVICE_PATH "/dev/bme280"
#define DEV_ID 0x76
// IOCTL commands
#define BME280_IOCTL_MAGIC 'm'
#define BME280_IOCTL_READ_T _IOR(BME280_IOCTL_MAGIC, 1, uint32_t)
#define BME280_IOCTL_READ_P _IOR(BME280_IOCTL_MAGIC, 2, uint32_t)
#define BME280_IOCTL_READ_H _IOR(BME280_IOCTL_MAGIC, 3, uint32_t)
#define BME280_IOCTL_INIT_FORCED_MODE _IO(BME280_IOCTL_MAGIC, 4)
#define BME280_IOCTL_INIT_NORMAL_MODE _IO(BME280_IOCTL_MAGIC, 5)
#define BME280_IOCTL_CHECK_ADDR _IO(BME280_IOCTL_MAGIC, 6)
#define BME280_IOCTL_CHECK_STATUS _IO(BME280_IOCTL_MAGIC, 7)
```

```
int main() {
    int fd;
    uint8_t status;
    uint8_t id;
    int32_t temp_int = 0;
    int32_t press_int = 0;
    uint32_t hum_int = 0;
    // Open the device
    fd = open(DEVICE_PATH, O_RDWR);
    if (fd < 0) {
        perror("Failed to open the device");
        return errno;
    }
    printf("Successfully open device\n");
    // Check ID
    id = ioctl(fd, BME280_IOCTL_CHECK_ADDR);
    if (id < 0) {
        perror("Unable to configure i2c slave device");
        close(fd);
        return 2;
    }
    //Init sensor mode
    if (ioctl(fd, BME280_IOCTL_INIT_FORCED_MODE) < 0) {
        perror("Failed to initialize the sensor mode");
        close(fd);
        return errno;
    }
    printf("Sensor initialized successfully\n");
    //-------------------------------------------------------------------------
```

```
while(1){
        sleep(1);
        //Check STATUS
        status = ioctl(fd, BME280_IOCTL_CHECK_STATUS);
        if (status < 0) {
            perror("Data is not ready");
            close(fd);
        }
        // Read Temperature
        if (ioctl(fd, BME280_IOCTL_READ_T, &temp_int) < 0) {
            perror("Failed to read temparature data");
            close(fd);
            return errno;
        }
        printf("Temparature: %.2f°C\n", temp_int/100.0);
        // Read Pressure
        if (ioctl(fd, BME280_IOCTL_READ_P, &press_int) < 0) {
            perror("Failed to read pressure data");
            close(fd);
            return errno;
        }
        printf("Pressure: %.2f hPa\n", press_int/100.0);
        // Read Humidity
        if (ioctl(fd, BME280_IOCTL_READ_H, &hum_int) < 0) {
            perror("Failed to read humidity data");
            close(fd);
            return errno;
        }
        printf("Humidity: %.2f %%rH\n", hum_int/1024.0);
    }
    close(fd);
    return 0;
```