Đọc cảm biến ánh sáng BH1750

Note: đọc datasheet của cảm biến và tài liệu rm của stm32 trước khi code

I. Sơ lược về giao thức I2C

1. Khái niệm

- I2C (Inter-Integrated Circuit) là phương thức giao tiếp nối tiếp.2.Chức năng

- Giao tiếp giữa các vi điều khiển,..

2. Các kiến thức cần nắm

- Tốc độ truyền:

    Standard mode: 100 kbps

    Fast mode: 400 kbps

    High speed mode: 3.4 Mbps

    Ultra fast mode: 5 Mbps

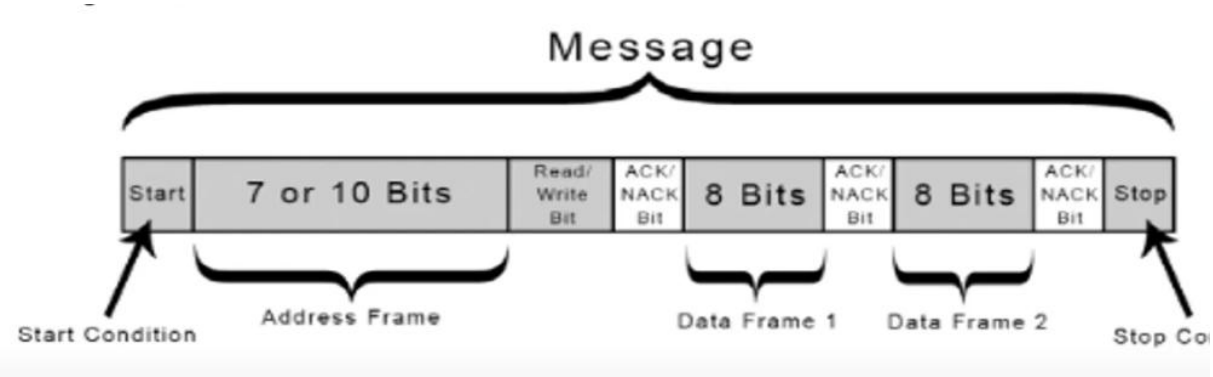
- SDA (Serial Data): Đường truyền và nhận dữ liệu.

- SCL (Serial Clock): Đường mang tín hiệu xung nhịp.

- Dữ liệu được truyền từng bit dọc theo một đường duy nhất

- Với I2C ta có thể kết nối nhiều Slave với một Master duy nhất (như SPI) và nhiều Master điều khiển một hoặc nhiều Slave.

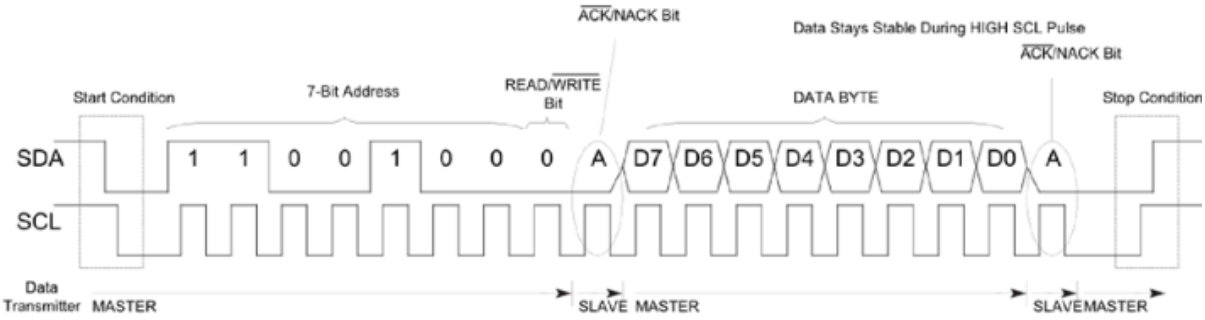
- Khi có nhiều master, các tín hiệu xung clock (SCL) của các Master có thể được đồng bộ hóa để đảm bảo chỉ có một tín hiệu clock được sử dụng tại một thời điểm.



- Bit ACK (Acknowledge): Xác nhận một byte dữ liệu đã được nhận thành công và có thể tiếp tục gửi byte tiếp theo. Sau khi gửi xong 8 bit dữ liệu, nó sẽ nhả bus để thiết bị nhận có thể phát hiện ACK. Thiết bị nhận (có thể là Master hoặc Slave) sẽ kéo đường SDA xuống mức logic thấp trong khoảng thời gian của xung clock thứ 9 (do master tạo ra), để cho biết rằng nó đã nhận thành công byte dữ liệu vừa được gửi.

- Bit NACK(Not Acknowledge): Báo hiệu dữ liệu không được gửi thành công hoặc không còn dữ liệu để nhận(kết thúc truyền). Thay vì kéo đường SDA xuống mức thấp trong xung clock thứ 9, thiết bị nhận để bus SDA ở mức cao. Điều này tạo ra tín hiệu NACK.

- Cách hoạt động I2C: dữ liệu được truyền qua các tin nhắn, tin nhắn được chia thành các khung dữ liệu, mỗi tin nhắn có một khung địa chỉ.



- Điều kiện khởi động: Đường SDA chuyển từ cao xuống thấp trước khi SCL chuyển từ cao xuống thấp.

- Điều kiện dừng: Đường SDA chuyển từ thấp lên cao sau khi đường SCL chuyển từ thấp lên cao(tức là SCL lên cao trước).

- Khung địa chỉ: Mỗi chuỗi 7 bit hoặc 10 bit duy nhất cho mỗi Slave để xác định Slave khi Master muốn giao tiếp với nó(tức là mỗi Slave sẽ có địa chỉ).

II. Đọc cảm biến ánh sáng BH1750

1. Đọc datasheet của cảm biến(tự đọc trên mạng có rất nhiều)
2. Code cấu hình I2C1

- File define.h : nơi mà include các thư viện cần (có thể có hoặc không)

```
#ifndef _DEFINES_H
#define _DEFINES_H

#include <stdio.h>
#include <stdlib.h>

#include "stm32f10x.h"
#include "stm32f10x_dma.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_exti.h"            // Keil::Device:StdPeriph Drivers:EXTI

#include "app_i2c.h"
#include "bh1750.h"
#include "tim2.h"
#include "uart.h"


#endif
```

- File app_i2c.h : khai báo các hàm của I2C

```
#ifndef _APP_I2C_H
#define _APP_I2C_H

#include "defines.h"

void I2C1_config (void);
void I2C1_write(unsigned char HW_address, unsigned char sub, unsigned char data);
char I2C1_read (unsigned char HW_address, unsigned char sub);
void I2C1_read_buf (unsigned char HW_address, unsigned char sub, unsigned char * p_buf, unsigned char buf_size);


#endif
```

- File bh1750.h : khai báo địa chỉ của cảm biến(đọc datasheet hoặc dùng code để dò ra địa chỉ I2C của cảm biến) và các hàm của cảm biến(Init , Read)

```h
#ifndef _BH1750_H
#define _BH1750_H

#include "defines.h"

#define BH1750_ADDR          0x46

#define BH1750_POWER_DOWN    0x00
#define BH1750_POWER_ON      0x01
#define BH1750_RESET         0x07
#define BH1750_CONT_H_MODE   0x10
#define BH1750_CONT_H_MODE2  0x11
#define BH1750_CONT_L_MODE   0x13
#define BH1750_ONE_H_MODE    0x20
#define BH1750_ONE_H_MODE2   0x21
#define BH1750_ONE_L_MODE    0x23
#define BH1750_CHG_MEAS_TIME_H 0x40
#define BH1750_CHG_MEAS_TIME_L 0x60

#define I2C_TIMEOUT          100000

void BH1750_Init(void);
float BH1750_ReadLux (void);

#endif
```

- File app_i2c.c : định nghĩa các hàm mà đã khai báo ở trong file.h (cần hiểu kĩ về giao thức I2C để code được)
    - Hàm I2C_Config():
        - Kích hoạt clock cho GPIOB, AFIO và I2C1

```
//Kích hoạt đồng hồ cho GPIOB và AFIO (chức năng luân phiên) trên bus APB2
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
//Kích hoạt đồng hồ cho I2C1 trên bus APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
```

        - Cấu hình PB6 (SCL) và PB7 (SDA) ở chế độ AF_OD với tốc độ 50MHz

```
    //Khai báo biến cấu trúc để cấu hình GPIO
    GPIO_InitTypeDef GPIO_InitStructure;
    //Khai báo biến cấu trúc để cấu hình I2C
    I2C_InitTypeDef I2C_InitStructure;

    //Cấu hình chân GPIO cho I2C
    //Chế độ Alternate Function Open Drain (bắt buộc cho I2C)
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    //Sử dụng chân PB6 (SCL) và PB7 (SDA)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;
    //Tốc độ truyền dữ liệu tối đa 50MHz
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    //Áp dụng cấu hình cho GPIOB
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

        - Thiết lập I2C với tốc độ 200kHz, địa chỉ 7-bit, tắt ACK

```
    // Cấu hình module I2C
    // Tắt chế độ tự động xác nhận (ACK)
    I2C_InitStructure.I2C_Ack = I2C_Ack_Disable;
    // Sử dụng địa chỉ 7-bit (chuẩn I2C thông thường)
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    // Thiết lập tốc độ clock 200kHz (giữa standard mode 100kHz và fast mode 400kHz)
    I2C_InitStructure.I2C_ClockSpeed = 200000;
    // Chu kỳ xung tỉ lệ 1:2 (tỉ lệ thời gian cao:thấp của xung clock)
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    // Hoạt động ở chế độ I2C tiêu chuẩn
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    // Địa chỉ của vi điều khiển khi hoạt động ở chế độ slave
    I2C_InitStructure.I2C_OwnAddress1 = 0x0B;
```

- Kích hoạt module I2C1

```
    // Áp dụng cấu hình cho I2C1
    I2C_Init(I2C1,&I2C_InitStructure);
    // Kích hoạt I2C1
    I2C_Cmd(I2C1,ENABLE);
```

- Hàm I2C1_Write(): ghi dữ liệu tới thiết bị I2C
  - Tham số: địa chỉ thiết bị, địa chỉ thanh ghi, dữ liệu

```
/**
 * Hàm ghi dữ liệu tới thiết bị I2C
 * @param HW_address: Địa chỉ thiết bị (7 bit)
 * @param sub: Địa chỉ thanh ghi con (0xFF nếu không sử dụng)
 * @param data: Dữ liệu cần ghi
 */
void I2C1_write(unsigned char HW_address, unsigned char sub, unsigned char data)
```

- Có timeout để tránh treo hệ thống

```
    {
        ticks--;
    }
    if (ticks == 0) return;
    ticks = I2C_TIMEOUT;
```

- Tạo điều kiện START → gửi địa chỉ → gửi dữ liệu → tạo điều kiện STOP

```
    // Tạo điều kiện START
    I2C_GenerateSTART(I2C1, ENABLE);
    // Đợi cho đến khi trạng thái MASTER được chọn
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT)&&ticks) {ticks--;}
    // Thoát nếu quá thời gian chờ
    if (ticks == 0) return;
    ticks = I2C_TIMEOUT;

    // Gửi địa chỉ thiết bị ở chế độ truyền (TRANSMITTER)
    I2C_Send7bitAddress(I2C1, HW_address, I2C_Direction_Transmitter);
    // Đợi cho đến khi chế độ MASTER TRANSMITTER được chọn
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)) && ticks)
    {
        ticks--;
    }
    if (ticks == 0) return;
    ticks = I2C_TIMEOUT;

    // Nếu sub != 0xFF thì gửi địa chỉ thanh ghi con
    if (sub != 0xFF)
    {
        // Gửi địa chỉ thanh ghi con
        I2C_SendData(I2C1, sub);
        // Đợi cho đến khi byte được truyền đi
        while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED))&&ticks) {ticks--;}
        if (ticks == 0) return;
        ticks = I2C_TIMEOUT;
    }

    // Gửi dữ liệu cần ghi
    I2C_SendData(I2C1, data);
    // Đợi cho đến khi byte được truyền đi
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED))&&ticks) {ticks--;}
    if (ticks == 0) return;
    ticks = I2C_TIMEOUT;

    // Tạo điều kiện STOP để kết thúc giao tiếp
    I2C_GenerateSTOP(I2C1, ENABLE);
    // Đợi cho đến khi bus I2C không còn bận
    while((I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY)) && ticks) {ticks--;}
    if (ticks == 0) return;
```

- Hàm I2C1_Read() : đọc 1 byte từ thiết bị I2C
  - Tham số: địa chỉ thiết bị, địa chỉ thanh ghi

```
/**
 * Hàm đọc 1 byte từ thiết bị I2C
 * @param HW_address: Địa chỉ thiết bị (7 bit)
 * @param sub: Địa chỉ thanh ghi con (0xFF nếu không sử dụng)
 * @return: Giá trị đọc được hoặc 0 nếu xảy ra lỗi
 */
char I2C1_read (unsigned char HW_address, unsigned char sub)
```

- Trả về: giá trị đọc được hoặc 0 nếu lỗi

```
char I2C1_read (unsigned char HW_address, unsigned char sub)
{
    char data;
    // Tạo điều kiện START
    I2C_GenerateSTART(I2C1, ENABLE);
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
    // Gửi địa chỉ thiết bị ở chế độ truyền để chọn thanh ghi
    I2C_Send7bitAddress(I2C1, HW_address, I2C_Direction_Transmitter);
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)));
    // Nếu sub != 0xFF thì gửi địa chỉ thanh ghi con
    if (sub != 0xFF)
    {
        I2C_SendData(I2C1, sub);
        while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED)));
    }
    // Tạo điều kiện START lặp lại (repeated START) để chuyển sang chế độ nhận
    I2C_GenerateSTART(I2C1, ENABLE);
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT)));
    // Gửi địa chỉ thiết bị ở chế độ nhận (RECEIVER)
    I2C_Send7bitAddress(I2C1, HW_address, I2C_Direction_Receiver);
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)));
    // Đợi cho đến khi nhận được byte dữ liệu
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)));
    // Đọc dữ liệu từ thanh ghi dữ liệu
    data = I2C1->DR;
    // Cấu hình vị trí NACK (không xác nhận)
    I2C_NACKPositionConfig(I2C1,I2C_NACKPosition_Current);
    // Tạo điều kiện STOP để kết thúc giao tiếp
    I2C_GenerateSTOP(I2C1, ENABLE);
    while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
    // Trả về dữ liệu đã đọc
    return data;
}
```

- Có cơ chế timeout để tránh treo (giống các hàm trên)

- I2C1_read_buf(): Đọc nhiều byte từ thiết bị I2C
  - Tham số: địa chỉ thiết bị, địa chỉ thanh ghi, con trỏ buffer, kích thước buffer

```
/**
 * Hàm đọc nhiều byte từ thiết bị I2C vào buffer
 * @param HW_address: Địa chỉ thiết bị (7 bit)
 * @param sub: Địa chỉ thanh ghi con (0xFF nếu không sử dụng)
 * @param p_buf: Con trỏ đến buffer lưu dữ liệu
 * @param buf_size: Kích thước buffer (số byte cần đọc)
 */
void I2C1_read_buf (unsigned char HW_address, unsigned char sub, unsigned char * p_buf, unsigned char buf_size)
```

  - Đọc dữ liệu vào buffer với xác nhận (ACK) cho mỗi byte
  - Tắt ACK và tạo STOP sau khi hoàn thành

```
    // Tạo điều kiện START
    I2C_GenerateSTART(I2C1, ENABLE);
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
    // Gửi địa chỉ thiết bị ở chế độ truyền để chọn thanh ghi
    I2C_Send7bitAddress(I2C1, HW_address, I2C_Direction_Transmitter);
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
    // Nếu sub != 0xFF thì gửi địa chỉ thanh ghi con
    if (sub != 0xFF)
    {
        I2C_SendData(I2C1, sub);
        while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED)));
    }
    // Tạo điều kiện START lặp lại (repeated START) để chuyển sang chế độ nhận
    I2C_GenerateSTART(I2C1, ENABLE);
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT)));
    // Bật chế độ ACK để xác nhận dữ liệu nhận được
    I2C_AcknowledgeConfig(I2C1,ENABLE);
    // Gửi địa chỉ thiết bị ở chế độ nhận (RECEIVER)
    I2C_Send7bitAddress(I2C1, HW_address, I2C_Direction_Receiver);
    while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)));
    // Đọc nhiều byte dữ liệu vvào buffer
    for (uint8_t i=0; i<buf_size; i++)
    {
        // Đợi cho đến khi nhận được byte dữ liệu
        while((!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)));
        // Lưu dữ liệu vào buffer
        p_buf[i] = I2C1->DR;
    }
    // Tắt chế độ ACK sau khi đọc xong tất cả các byte
    I2C_AcknowledgeConfig(I2C1,DISABLE);
    // Cấu hình vị trí NACK
    I2C_NACKPositionConfig(I2C1,I2C_NACKPosition_Current);
    // Tạo điều kiện STOP để kết thúc giao tiếp
    I2C_GenerateSTOP(I2C1, ENABLE);
    while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
```

3. Đọc cảm biến

- Phần Init

```
void BH1750_Init (void)
{
    // Khởi tạo và cấu hình giao tiếp I2C
    I2C1_config();
    // Gửi lệnh bật nguồn đến cảm biến BH1750 theo địa chỉ BH1750_ADDR
    I2C1_write(BH1750_ADDR, 0xFF, BH1750_POWER_ON);

}
```

- **Phần đọc dữ liệu từ cảm biến**

```
    // Đọc dữ liệu từ cảm biến ở chế độ đo một lần với độ phân giải cao
    I2C1_read_buf(BH1750_ADDR, BH1750_ONE_H_MODE, tmp8, 2);
```

Dòng này thực hiện các nhiệm vụ sau:

- BH1750_ADDR: Địa chỉ I2C của cảm biến (thường là 0x23 hoặc 0x5C tùy thuộc vào cách kết nối chân ADDR)
- BH1750_ONE_H_MODE: Đây là lệnh để cảm biến thực hiện đo cường độ ánh sáng một lần ở độ phân giải cao
- tmp8: Mảng đệm để lưu 2 byte dữ liệu đọc được
- 2: Số byte cần đọc từ cảm biến

Cảm biến BH1750 trả về giá trị cường độ ánh sáng dưới dạng 2 byte dữ liệu qua giao tiếp I2C.

Phần tính toán giá trị ánh sáng

```
    // Chuyển đổi 2 byte thành giá trị 16-bit và chia cho 1.2 để có giá trị lux
    // byte đầu tiên là byte cao (MSB), byte thứ hai là byte thấp (LSB)
    ret = (float)(tmp8[0]<<8 | tmp8[1])/1.2;
```

Dòng này thực hiện các bước chuyển đổi:

1. tmp8[0]<<8: Dịch byte đầu tiên (byte cao - MSB) sang trái 8 bit
2. | tmp8[1]: Kết hợp (phép OR) với byte thứ hai (byte thấp - LSB)
3. Kết quả là một số nguyên 16-bit biểu diễn giá trị đo thô

4. /1.2: Chia cho 1.2 để chuyển đổi thành đơn vị lux

Tại sao lại chia cho 1.2?

Đây là hệ số chuyển đổi được quy định trong tài liệu kỹ thuật (datasheet) của cảm biến BH1750. Theo thiết kế của cảm biến, để chuyển đổi giá trị số thô thành đơn vị lux chính xác, cần áp dụng công thức:

Lux = (MSB << 8 | LSB) / 1.2

Trong đó:

- MSB (Most Significant Byte): Byte cao (tmp8[0])
- LSB (Least Significant Byte): Byte thấp (tmp8[1])

Hệ số 1.2 này được nhà sản xuất cảm biến BH1750 xác định dựa trên đặc tính vật lý của cảm biến và bộ chuyển đổi tương tự - số (ADC) bên trong nó.