扩展点1——寄存器配置MyI2C

知识笔记

1. 直接配置寄存器与调用库函数

库函数的底层就是配置寄存器，只不过库函数经过了系统的封装。比起直接配置寄存器，有了直观易懂的命名，同时使用也更加方便。

在工程文件中，我们可以对右键库函数进行一层一层的“跳转到定义”，就可以逐步接近底层对寄存器的操作。

2. 直接配置寄存器的好处

- 更接近底层，利于搞明白各个寄存器的功能和配置细节
- 更方便对寄存器进行位操作，单独改变某一位的值

3. 如何直接配置寄存器

配置寄存器最最最重要的是STM32官方手册，上面有对外设的简介和寄存器详尽的描述

1. 先搞清楚你要操作的对象，要操作的内容
2. 在手册目录中找到需要的模块单元，浏览有关寄存器
3. 找到与要实现的功能相对应的寄存器，阅读寄存器描述
4. 写出目标寄存器的赋值，转化为16进制
5. 在工程中直接给寄存器赋值
6. 每个操作写好注释，不然之后很可能不知道在干什么

> 辅助手段：先找到相应的库函数，然后跳转分析库函数内部的代码，我们做的就是挖掘库函数的核心操作部分。

4. 与重写MyI2C文件有关的寄存器

- **My_I2C初始化**
  - *APB2外设时钟使能寄存器*

6.3.7 APB2 外设时钟使能寄存器(RCC_APB2ENR)

偏移地址：0x18

复位值：0x0000 0000

访问：字，半字和字节访问

通常无访问等待周期。但在APB2总线上的外设被访问时，将插入等待状态直到APB2的外设访问结束。

注：当外设时钟没有启用时，软件不能读出外设寄存器的数值，返回的数值始终是0x0。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	保留	AFIOEN
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

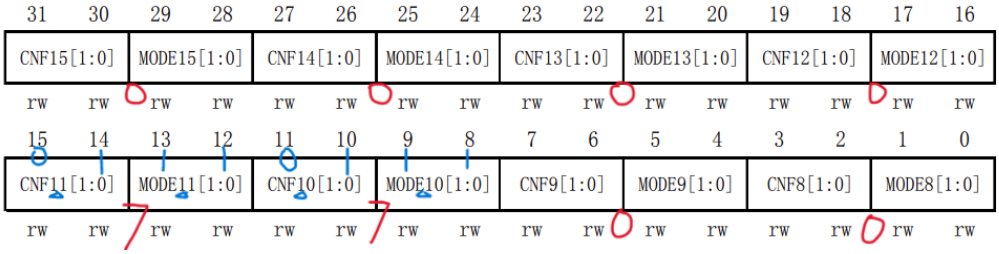
图示赋值代表开启APB2总线上GPIOB外设的时钟。

- *端口配置高寄存器*

8.2.2 端口配置高寄存器(GPIOx_CRH) (x=A..E)

偏移地址：0x04

复位值：0x4444 4444



CNF：00-推挽输出；01-开漏输出；10-复用推挽输出；11-复用开漏输出

MODE：00-输入模式；01-输出最大速率10MHz；10-2MHz；11-50MHz 图示赋值代表配置Pin10， Pin11 —— 开漏输出 ; 传输频率50MHz。

- *端口输入数据寄存器*

8.2.4 端口输出数据寄存器(GPIOx_ODR) (x=A..E)

地址偏移：0Ch

复位值：0x0000 0000

图示赋值代表置Pin10， Pin11高电平。

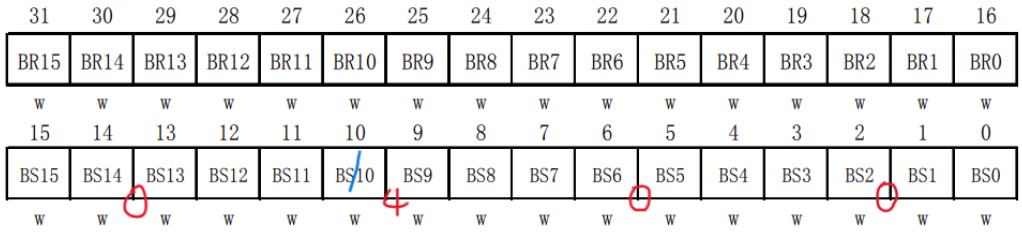
- **MyI2C六大基本时序**

- *端口位设置/清除寄存器*

8.2.5 端口位设置/清除寄存器(GPIOx_BSRR) (x=A..E)

地址偏移：0x10

复位值：0x0000 0000



低16位为设置，高16位是清除

图示赋值代表设置Pin_10为高电平。

- *端口位清除寄存器*

8.2.6 端口位清除寄存器(GPIOx_BRR) (x=A..E)

地址偏移：0x14

复位值：0x0000 0000

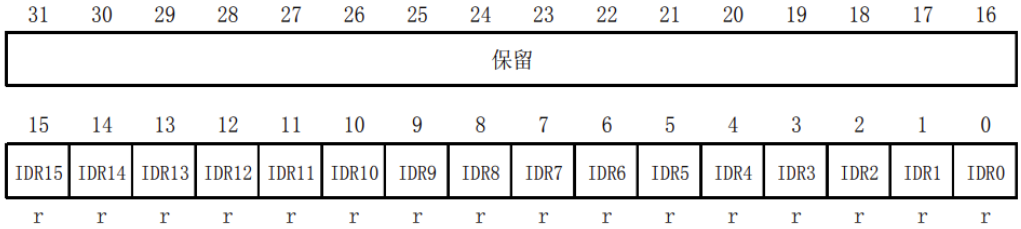
图示赋值代表清除Pin_10的高电平，也就是输出低电平。

- *端口输入数据寄存器*

8.2.3 端口输入数据寄存器(GPIOx_IDR) (x=A..E)

地址偏移：0x08

复位值：0x0000 XXXX



用于I2C中SDA读数据，所以赋值不确定。

实现步骤

下文中代码中注释掉的是原库函数代码，这样更有利于看懂直接配置寄存器的意图。

1. 初始化I2C：MyI2C_Init（）函数

```
//RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
RCC->APB2ENR = 0x00000008;    //RCC开启APB2时钟

/* GPIO输入输出模式配置
GPIO_InitTypeDef GPIO_InitStructrue;
GPIO_InitStructrue.GPIO_Mode = GPIO_Mode_Out_OD;
GPIO_InitStructrue.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
GPIO_InitStructrue.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB,&GPIO_InitStructrue);
*/
GPIOB->CRH = 0x00007700;     //Pin10 Pin11 ; 开漏输出 ; 50MHz

//GPIO_SetBits(GPIOB,GPIO_Pin_10 | GPIO_Pin_11);
GPIOB->ODR = 0x00000C00;    //Pin10 Pin11置高电平
```

2. I2C起始信号：MyI2C_Start（）函数

```
void MyI2C_Start(void)    //SCL高电平期间，SDA产生下降沿
{
```

```
    //MyI2C_W_SDA(1);
    //MyI2C_W_SCL(1);
    //MyI2C_W_SDA(0);
    //MyI2C_W_SCL(0);

    GPIOB->BSRR = 0x00000800;    //先释放SDA，避免终止条件提前出现
    Delay_us(10);
    GPIOB->BSRR = 0x00000400;    //SCL高电平
    Delay_us(10);
    GPIOB->BRR = 0x00000800;    //SDA产生下降沿
    Delay_us(10);
    GPIOB->BRR = 0x00000400;    //SCL低电平
    Delay_us(10);
}
```

3. I2C终止信号：MyI2C_Stop（）函数

```
void MyI2C_Stop(void)    //SCL高电平期间，SDA出现上升沿
{
    //MyI2C_W_SDA(0);
    //MyI2C_W_SCL(1);
    //MyI2C_W_SDA(1);

    GPIOB->BRR = 0x00000800;    //开始时先拉低SDA，确保其能产生上升沿
    Delay_us(10);
    GPIOB->BSRR = 0x00000400;    //SCL高电平
    Delay_us(10);
    GPIOB->BSRR = 0x00000800;    //SDA出现上升沿
    Delay_us(10);
}
```

4. I2C发送一个字节：MyI2C_SendByte（uint8_t Byte）函数

```
void MyI2C_SendByte(uint8_t Byte)
{
    uint8_t i;
    for (i = 0; i < 8; i ++)
    {
        //MyI2C_W_SDA(Byte & (0x80 >> i));
        //MyI2C_W_SCL(1);
        //MyI2C_W_SCL(0);

        if ((Byte & (0x80 >> i)) != Bit_RESET)    //如果Byte对应位为1
        {
            GPIOB->BSRR = 0x00000800;    //SDA置高电平
        }
        else
        {
            GPIOB->BRR = 0x00000800;    //否则SDA置低电平
        }
        Delay_us(10);
```

```
        GPIOB->BSRR = 0x00000400;    //SCL置高电平
        Delay_us(10);

        GPIOB->BRR = 0x00000400;    //SCL置低电平
        Delay_us(10);
    }
}
```

5. I2C接收一个字节：MyI2C_ReceiveByte（uint8_t Byte）函数

```
uint8_t MyI2C_ReceiveByte(void)
{
    uint8_t i, Byte = 0x00;
    //MyI2C_W_SDA(1);
    GPIOB->BSRR = 0x00000800;    //主机释放控制权，转换为输入模式
    for (i = 0; i < 8; i ++)
    {
        //MyI2C_W_SCL(1);
        //if (MyI2C_R_SDA() == 1){Byte |= (0x80 >> i);}
        //MyI2C_W_SCL(0);

        GPIOB->BSRR = 0x00000400;    //SCL置高电平
        Delay_us(10);

        if ((GPIOB->IDR & 0x00000800) != (uint32_t)Bit_RESET)
        {
            Delay_us(10);
            Byte |= (0x80 >> i);    //如果接收高电平，则对应位写1
        }

        GPIOB->BRR = 0x00000400;    //SCL置低电平
        Delay_us(10);
    }
    return Byte;    //返回接收数据
}
```

6. I2C发送应答：MyI2C_SendAck(uint8_t AckBit)函数

```
void MyI2C_SendAck(uint8_t AckBit)
{
    //MyI2C_W_SDA(AckBit);
    //MyI2C_W_SCL(1);
    //MyI2C_W_SCL(0);

    if (AckBit != Bit_RESET)    //如果应答为1
    {
        GPIOB->BSRR = 0x00000800;    //SDA置高电平
    }
    else
    {
        GPIOB->BRR = 0x00000800;    //否则SDA置低电平
    }
```

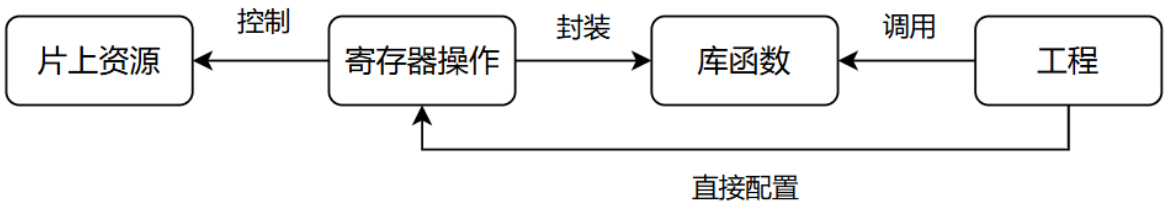
```
    Delay_us(10);

    GPIOB->BSRR = 0x00000400;    //SCL置高电平
    Delay_us(10);

    GPIOB->BRR = 0x00000400;    //SCL置低电平
    Delay_us(10);
}
```

7. I2C接受应答：MyI2C_ReceiveAck(void)函数

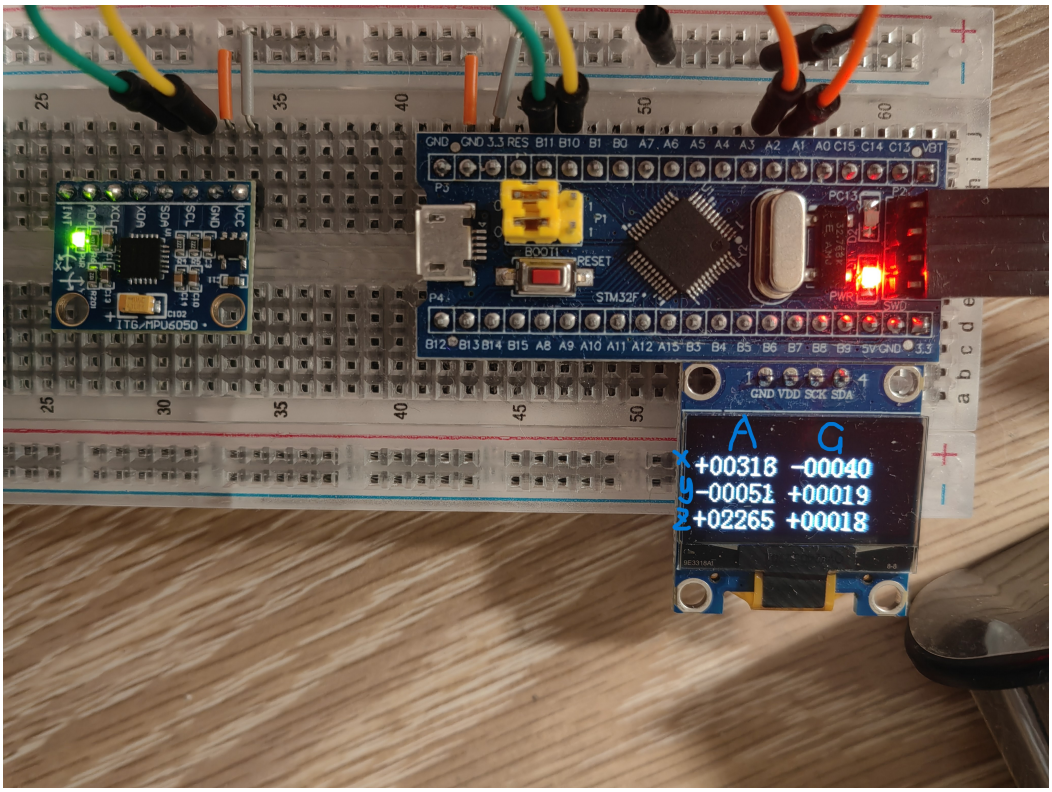
```
uint8_t MyI2C_ReceiveAck(void)
{
    uint8_t AckBit;
    //MyI2C_W_SDA(1);
    //MyI2C_W_SCL(1);
    //AckBit = MyI2C_R_SDA();
    //MyI2C_W_SCL(0);

    GPIOB->BSRR = 0x00000800;    //主机释放控制权，防止干扰从机
    Delay_us(10);

    GPIOB->BSRR = 0x00000400;    //SCL置高电平
    Delay_us(10);

    AckBit = GPIOB->IDR & 0x00000800;   //读取SDA电平

    GPIOB->BRR = 0x00000400;    //SCL置低电平
    Delay_us(10);

    return AckBit;    //返回SDA电平信号
}
```

8. 用重写的MyI2C读写MPU6050原始数据

框图



照片


A G
X +00318 -00040
Y -00051 +00019
Z +02265 +00018
GND VDD SCK SDA
BOOT
RESET
STM32F
ITG/MPU6050
GND 3.3 RES B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VBT
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V GND 3.3