

# 第十章 STM8S207 外部输入中断及其应用实例

这一章内容和第九章内容有很大关联,前面章节已经对 TLI 外部不可屏蔽中断有过详细说明,所以这一章相对来说比较简单

## 10.1 STM8S207 外部可屏蔽中断

这一节,我们将向大家介绍如何使用 STM8 的外部输入中断。通过前面的学习,我们掌握了 STM8 的 IO 口操作以及设计到的串口中断。这节课我们将介绍作为外部中断输入口,STM8 需要做的一些设置

## 10.2 STM8 外部中断简介

STM8 的 IO 口在之前已经有详细的介绍,而中断也在串口章节中做过介绍。我们这节课是将这两者结合起来,实现外部中断输入。

STM8 的大部分 IO 口可以作为中断输入,这点很好用。具体的 IO 口有 PA, PB, PC, PD, PE

要把 IO 口作为外部中断输入,有以下步骤:

- 1) 初始化 IO 口作为输入中断,其中有悬浮和上拉,设置 CR1 和 CR2 寄存器
- 2) 设置中断产生条件上升沿还是下降沿或其它类型,设置 EXTI\_CR1, EXTI\_CR2
- 3) 在 main 函数中开全局中断 `asm("rim")`
- 4) 编写中断服务程序

注意的是,外部中断是每个端口分配一个中断向量号,例如 PA 口只分配了 3 这个中断向量号,也就是说 8 个 IO 口共用一个中断向量号,我们可以通过 IDR 寄存器读出对应的 IO 口引脚值来判断到底是那个产生的中断,这在初始化了多个 IO 口的必须使用的。

更为要注意的是 PD7 是一个特别的外部中断输入,看数据手册可以知道 PD7 后缀有 TLI,这个是拥有独立的中断向量号,为 0

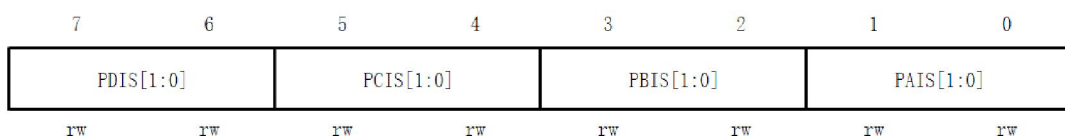
下面我们将逐步讲解与外部中断有关的寄存器

### 1) CR1, CR2

这两个寄存器相信大家都很熟悉了,我们前面与 IO 口有关的操作都要到这两个寄存器,之前我们设置的一般都是推挽式输出以及上拉式输入

在这里我们设置成中断上拉输入,因为我们的硬件上没有外接上拉,需要内部上拉,设置对应位的寄存器值是  $CR1 = 1$  以及  $CR2 = 0$

### 2) EXTI\_CR1

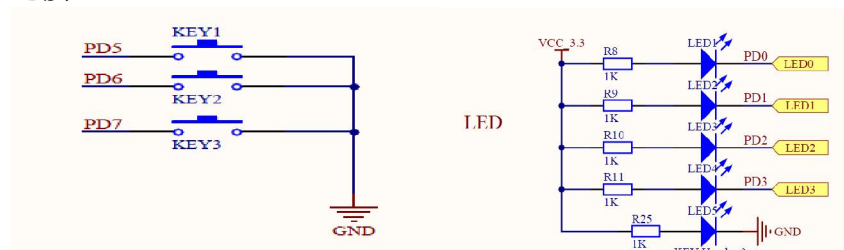


我们使用的是 D 口，所以只需要设置与 D 口有关的位，EXTI\_CR1 中的第 6、7 位是设置 PD 口的中断方式。如下图所示：

位7:0	<b>PDIS[1:0] : PORT D 的中断触发位</b> 这些位仅在CC寄存器的I1和I0位都为1(级别3)时才可以写入。这些位定义端口D 的中断触发位 00: 下降沿和低电平触发 01: 仅上升沿触发 10: 仅下降沿触发 11: 上升沿和下降沿触发
------	---------------------------------------------------------------------------------------------------------------------------------------------------

我们的实验是设置为仅下降沿触发，所以在我们只需要设置为 EXTI\_CR1 = 0x80 就可以了。

实验硬件连接：



下面是软件代码的编写

```
#include "iostm8s207rb.h"
#define LED1_FLASH PD_ODR_ODR0 = !PD_ODR_ODR0
#define LED2_FLASH PD_ODR_ODR1 = !PD_ODR_ODR1
#define LED3_FLASH PD_ODR_ODR2 = !PD_ODR_ODR2
#define LED4_FLASH PD_ODR_ODR3 = !PD_ODR_ODR3

void GPIO_init(void)
{
    PD_DDR = 0x0F; // 配置PD端口的方向寄存器
    PD_CR1 = 0xFF; // 设置推挽输出, 以及中断上拉输入
    PD_CR2 = 0xF0; // 使能PD5、6、7外部中断
    PD_ODR = 0xFF;
}

void EXTI_init(void)
{
    EXTI_CR1 = 0x80; // PD口下降沿触发中断
}

void init_devices(void)
```

```
{
    asm("sim"); // 关全局中断
    GPIO_init();
    EXTI_init();
    asm("rim"); // 开全局中断
}

void main( void )
{
    init_devices();
    // 主循环里没有程序需要执行
    while(1);
}

#pragma vector=0x02 // 这里很关键！看说明。
__interrupt void EXTI_PD7_TLI(void)
{
    LED4_FLASH;
}

#pragma vector=0x08
__interrupt void EXTI_PD(void)
{
    if(PD_IDR_IDR5 == 0) //key1按下
    {
        if(PD_IDR_IDR6 == 0) //key1 key2同时按下
            LED3_FLASH;
        else
            LED1_FLASH;
    }
    else
        LED2_FLASH;
}
```

编译下载后，通过不同的按键可以实现 LED 的开关，不过干扰很大。这是因为速度很快而且没有延时滤波而产生的，我们的实验只是说明外部中断的使用

**风驰电子祝您学习愉快!!! ~~~**