

按键中断（EXTI）例程实验

实验平台：奋斗版STM32开发板V2、V2.1、V3

实验内容：板子加电后，按动板子上K1-K3按键，可控制对应的LED1-LED3的亮灭，该实验学习了外部中断（EXTI）程序的编制及控制流程。

1 复用功能I/O和调试配置(AFIO)

为了优化外设数目，可以把一些复用功能重新映射到其他引脚上。设置复用重映射和调试I/O配置寄存器(AFIO_MAPR)(参见0节)实现引脚的重新映射。这时，复用功能不再映射到它们的原始分配上。

1.1 外部中断配置寄存器1(AFIO_EXTICR1)

地址偏移：08h

复位值：0000h

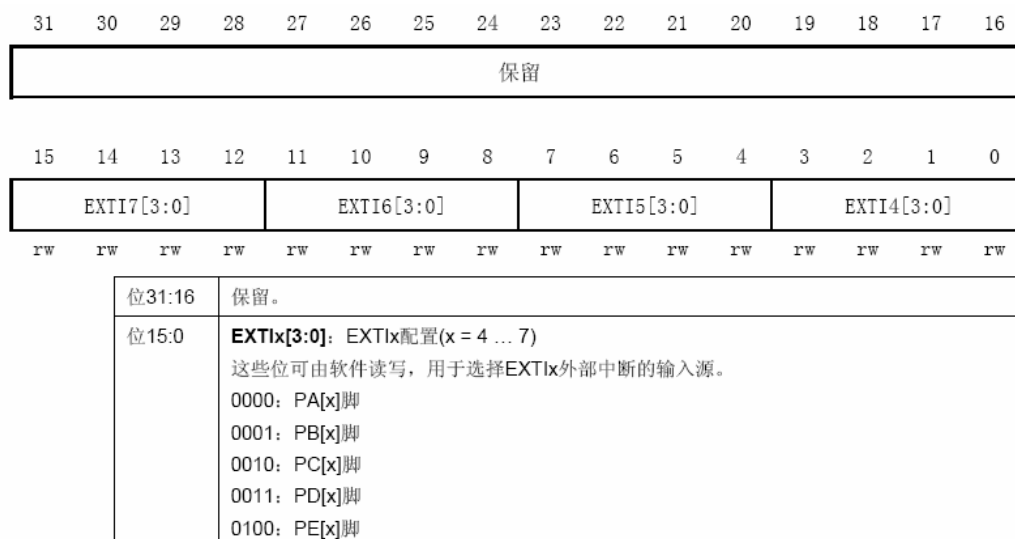
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W	I'W

位31:16	保留。
位15:0	EXTIx[3:0]: EXTIx配置(x = 0 ... 3) 这些位可由软件读写，用于选择EXTIx外部中断的输入源。参看6.2.5节外部中断/事件线映像。 0000: PA[x]脚 0001: PB[x]脚 0010: PC[x]脚 0011: PD[x]脚 0100: PE[x]脚

1.2 外部中断配置寄存器2(AFIO_EXTICR2)

地址偏移：0Ch

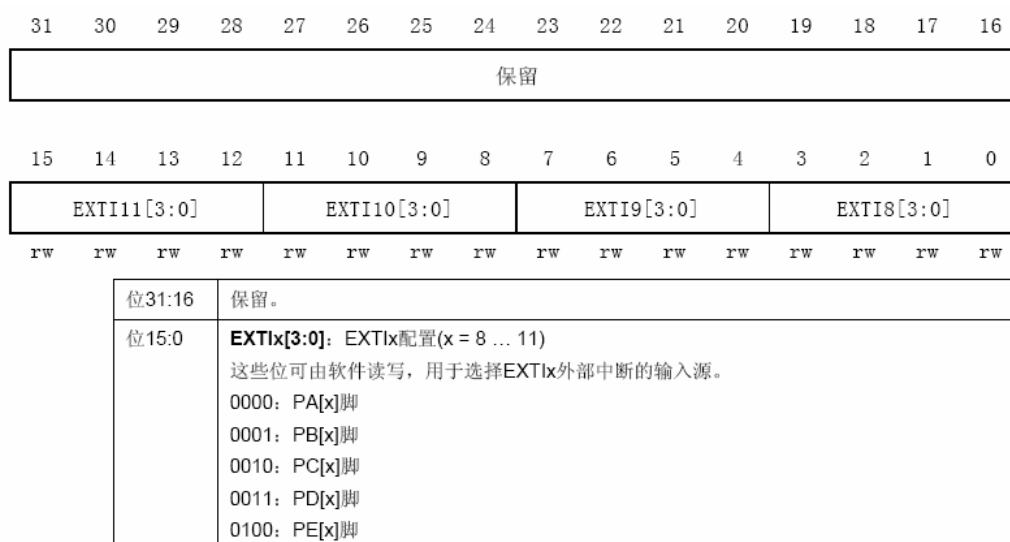
复位值：0000h



1.3 外部中断配置寄存器3(AFIO_EXTICR3)

地址偏移：10h

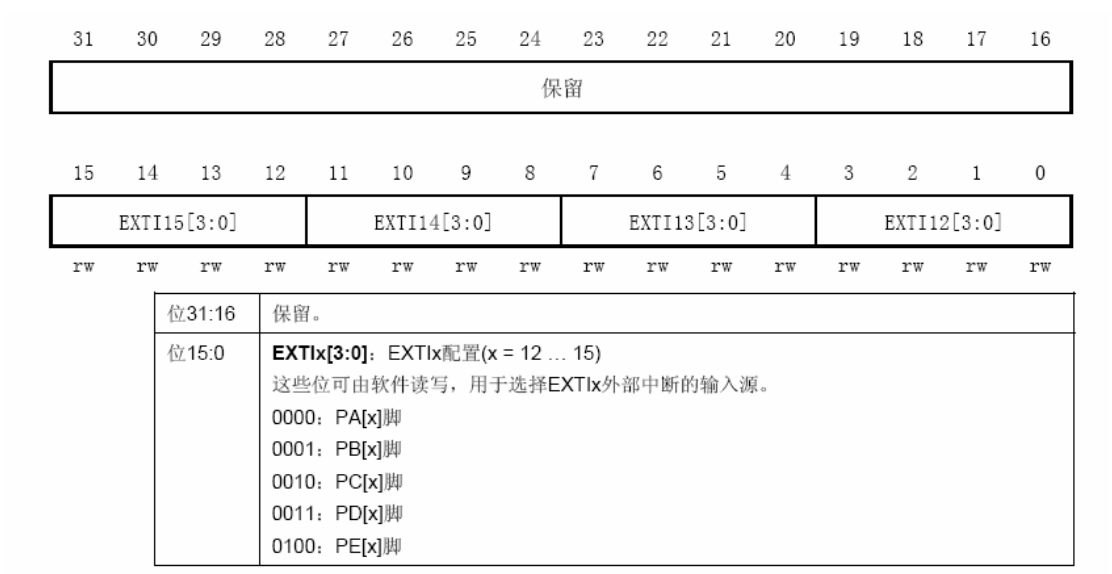
复位值：0000h



1.4 外部中断配置寄存器4(AFIO_EXTICR4)

地址偏移：14h

复位值：0000h



2.1 嵌套向量中断控制器（NVIC）

特性

- 43 个可屏蔽中断通道（不包含16 个Cortex-M3 的中断线）；
 - 16 个可编程的优先等级；
 - 低延迟的异常和中断处理；
 - 电源管理控制；
 - 系统控制寄存器的实现；
- 嵌套向量中断控制器(NVIC)和处理器核的接口紧密相连，可以实现低延迟的中断处理和有效处理地处理晚到的中断。
- 嵌套向量中断控制器管理着包括核异常等中断。关于更多的异常和NVIC编程的说明请参考ARM《Cortex-M3TM技术参考手册》的第5章的异常和第8章的嵌套向量中断控制器。

2.1.1 系统嘀嗒(SysTick)校准值寄存器

系统嘀嗒校准值固定到9000，当系统嘀嗒时钟设定为9兆赫，产生1ms时基。

2.1.2 中断和异常向量

表1 向量表

位置	优先级	优先级 类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC时钟安全系统(CSS)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效	所有类型的失效	0x0000_000C
	0	可设置	存储管理	存储器管理	0x0000_0010
	1	可设置	总线错误	预取指失败, 存储器访问失败	0x0000_0014
	2	可设置	错误应用	未定义的指令或非法状态	0x0000_0018
	-	-	-	保留	0x0000_001C ~0x0000_002B
	3	可设置	SVCall	通过SWI指令的系统服务调用	0x0000_002C
	4	可设置	调试监控	调试监控器	0x0000_0030
	-	-	-	保留	0x0000_0034
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C
0	7	可设置	WWDG	窗口定时器中断	0x0000_0040

1	8	可设置	PVD	联到EXTI的电源电压检测(PVD)中断	0x0000_0044
2	9	可设置	TAMPER	侵入检测中断	0x0000_0048
3	10	可设置	RTC	实时时钟(RTC)全局中断	0x0000_004C
4	11	可设置	FLASH	闪存全局中断	0x0000_0050
5	12	可设置	RCC	复位和时钟控制(RCC)中断	0x0000_0054
6	13	可设置	EXTI0	EXTI线0中断	0x0000_0058
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
11	18	可设置	DMA通道1	DMA通道1全局中断	0x0000_006C
12	19	可设置	DMA通道2	DMA通道2全局中断	0x0000_0070
13	20	可设置	DMA通道3	DMA通道3全局中断	0x0000_0074
14	21	可设置	DMA通道4	DMA通道4全局中断	0x0000_0078
15	22	可设置	DMA通道5	DMA通道5全局中断	0x0000_007C
16	23	可设置	DMA通道6	DMA通道6全局中断	0x0000_0080
17	24	可设置	DMA通道7	DMA通道7全局中断	0x0000_0084
18	25	可设置	ADC	ADC全局中断	0x0000_0088
19	26	可设置	USB_HP_CAN_TX	USB高优先级或CAN发送中断	0x0000_008C
20	27	可设置	USB_LP_CAN_RX0	USB低优先级或CAN接收0中断	0x0000_0090
21	28	可设置	CAN_RX1	CAN接收1中断	0x0000_0094
22	29	可设置	CAN_SCE	CAN SCE中断	0x0000_0098
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
24	31	可设置	TIM1_BRK	TIM1断开中断	0x0000_00A0
25	32	可设置	TIM1_UP	TIM1更新中断	0x0000_00A4
26	33	可设置	TIM1_TRG_COM	TIM1触发和通信中断	0x0000_00A8
27	34	可设置	TIM1_CC	TIM1捕获比较中断	0x0000_00AC
28	35	可设置	TIM2	TIM2全局中断	0x0000_00B0
29	36	可设置	TIM3	TIM3全局中断	0x0000_00B4
30	37	可设置	TIM4	TIM4全局中断	0x0000_00B8
31	38	可设置	I2C1_EV	I ² C1事件中断	0x0000_00BC
32	39	可设置	I2C1_ER	I ² C1错误中断	0x0000_00C0
33	40	可设置	I2C2_EV	I ² C2事件中断	0x0000_00C4
34	41	可设置	I2C2_ER	I ² C2错误中断	0x0000_00C8
35	42	可设置	SPI1	SPI1全局中断	0x0000_00CC
36	43	可设置	SPI2	SPI2全局中断	0x0000_00D0
37	44	可设置	USART1	USART1全局中断	0x0000_00D4

38	45	可设置	USART2	USART2全局中断	0x0000_00D8
39	46	可设置	USART3	USART3全局中断	0x0000_00DC
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0
41	48	可设置	RTCAlarm	联到EXTI的RTC闹钟中断	0x0000_00E4
42	49	可设置	USB唤醒	联到EXTI的从USB待机唤醒中断	0x0000_00E8

3.1 外部中断/事件控制器(EXTI)

外部中断/事件控制器由19个产生事件/中断要求的边沿检测器组成。每个输入线可以独立地配置输入类型（脉冲或挂起）和对应的触发事件（上升沿或下降沿或者双边沿都触发）。每个输入线都可以被独立的屏蔽。挂起寄存器保持着状态线的中断要求。

3.1.1 主要特性

EXTI控制器的主要特性如下：

- 每个中断/事件都有独立的触发和屏蔽

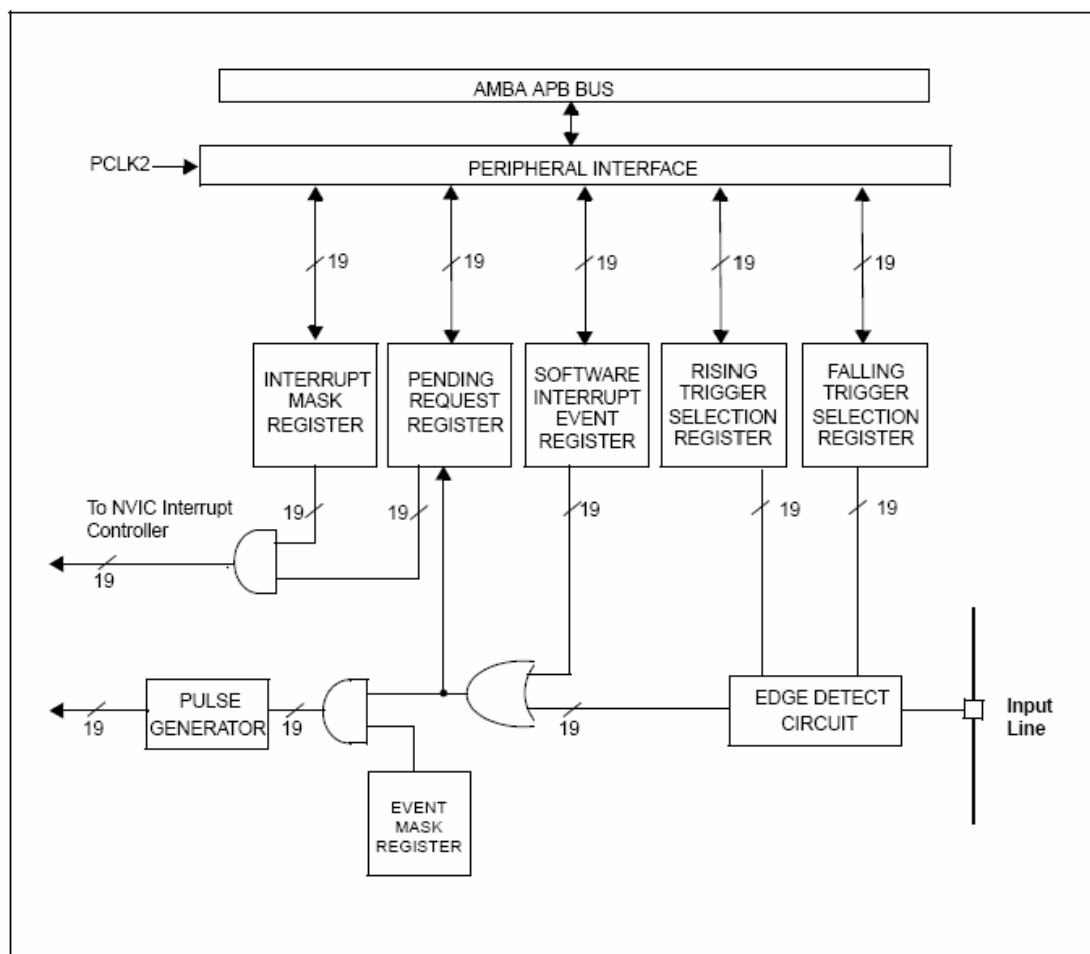
- 每个中断线都有专用的状态位

- 支持多达19 个中断/事件请求

- 检测脉冲宽度低于APB2 时钟宽度的外部信号。参见数据手册中电气特性部分的相关参数。

3.1.2 框图

图1 外部中断/事件控制器框图



3.1.3 唤醒事件管理

Cortex-M3 可以处理外部时间或内部中断来唤醒内核。通过配置任何外部I/O端口、RTC 闹钟和USB唤醒事件可以唤醒CPU（内核从WFE退出）。使用外部I/O端口作为唤醒事件，

3.1.4 功能说明

如要产生中断，中断线必须事先配置好并被激活。这是根据需要的边沿检测通过设置2个触发寄存器，和在中断屏蔽寄存器的相应位写“1”到来允许中断请求。当需要的边沿在外部中断线上发生时，将产生一个中断请求，对应的挂起位也随之被置1。通过写“1”到挂起寄存器，可以清除该中断请求。为产生事件触发，事件连接线必须事先配置好并被激活。这是根据需要的边沿检测通过设置2个触发寄存器，和在事件屏蔽寄存器的相应位写“1”到来允许事件请求。当需要的边沿在事件连线上发生时，将产生一个事件请求脉冲，对应的挂起位不被置1。通过在软件中断/事件寄存器写“1”，一个中断/事件请求也可以通过软件来产生。

通过下面的过程来配置19个线路做为中断源：

配置19 个中断线的屏蔽位（EXTI—IMR）

配置所选中断线的触发选择位（EXTI_RTISR 和EXTI_FTSR）；

配置那些控制映像到外部中断控制器(EXTI)的NVIC 中断通道的使能和屏蔽位，使得19个中断线中的请求可以被正确地响应。

硬件事件选择通过下面的过程，可以配置19个线路为事件源

配置19 个事件线的屏蔽位（EXTI_EMR）

配置事件线的触发选择位（EXTI_RTISR and EXTI_FTSR）

软件中断/事件的选择19个线路可以被配置成软件中断/事件线。下面是产生软件中断的过程：

配置19 个中断/事件线屏蔽位（EXTI_IMR, EXTI_EMR）

设置软件中断寄存器的请求位（EXTI_SWIER）

3.1.5 外部中断/事件线路映像

80通用I/O端口以下图的方式连接到19个外部中断/事件线上：

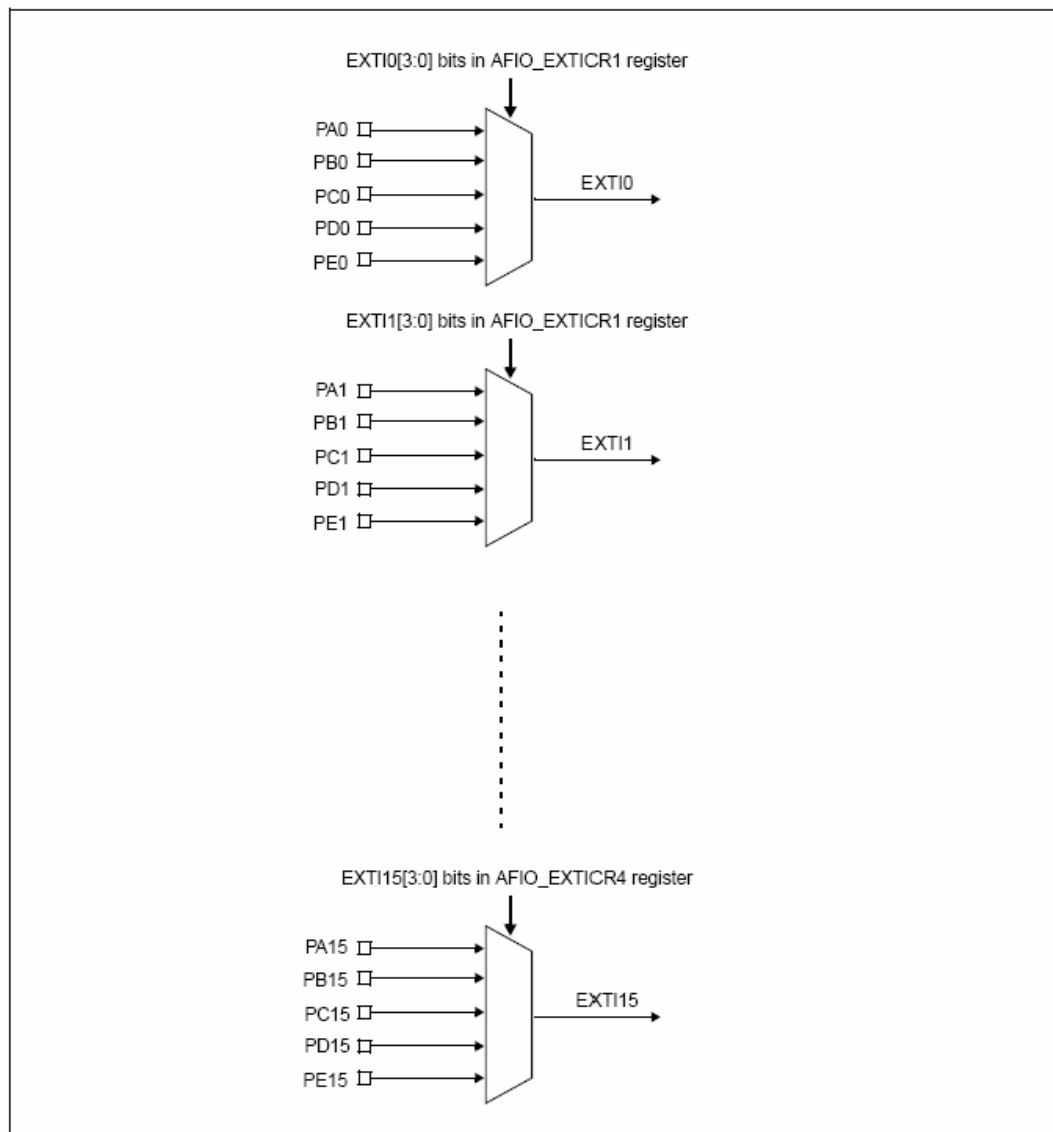


图2 外部中断通用I/O映像

另外三种其他的外部中断/事件控制器的连接如下：

EXTI 线16 连接到PVD 输出

EXTI 线17 连接到RTC 闹钟事件

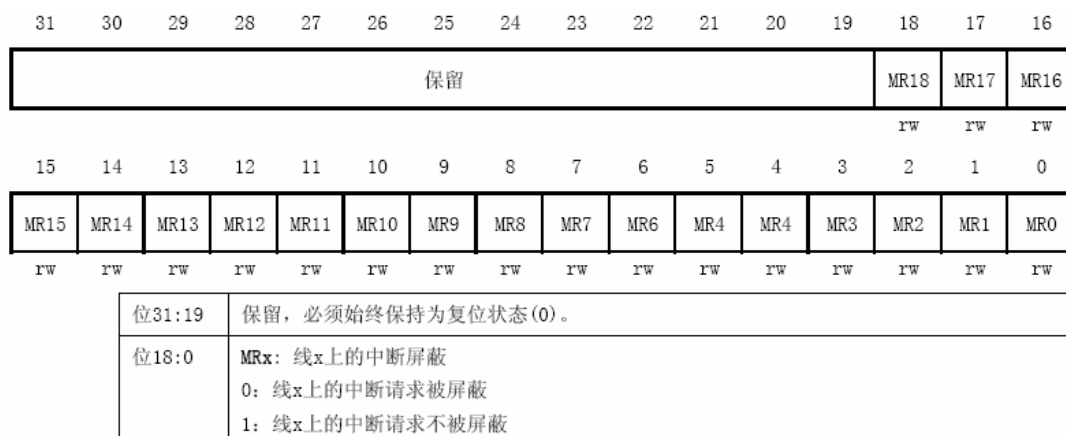
EXTI 线18 连接到USB 唤醒事件

3 EXTI 寄存器描述

中断屏蔽寄存器（EXTI_IMR）

偏移地址：00H

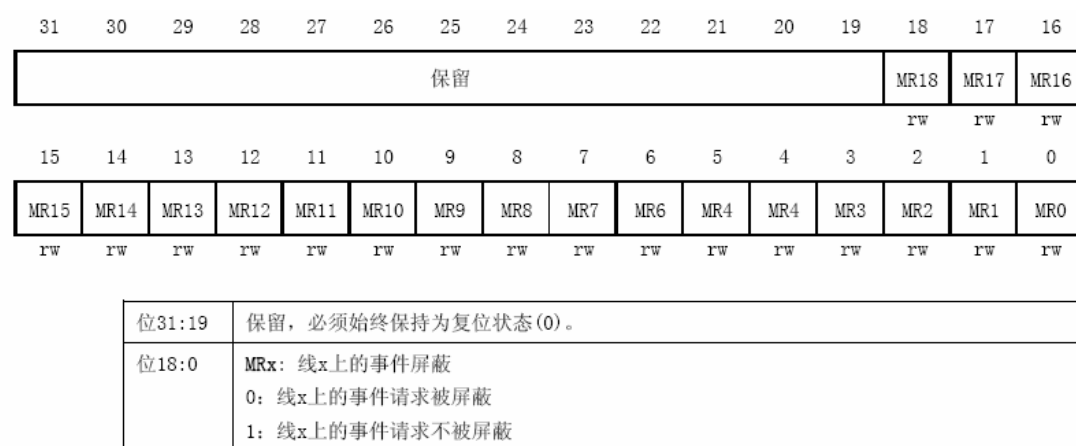
复位值：0000 0000h



事件屏蔽寄存器 (EXTI_EMR)

偏移地址：04H

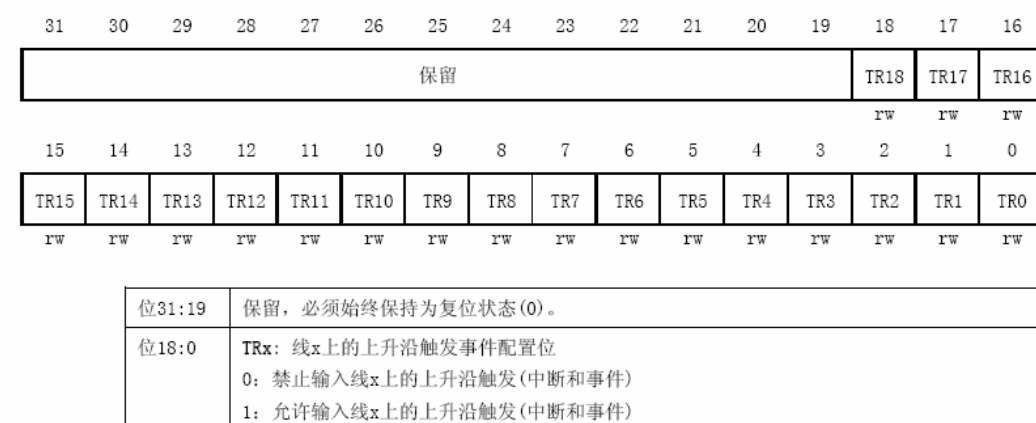
复位值：0000 0000h



上升沿触发选择寄存器 (EXTI_RTSTR)

偏移地址：08H

复位值：0000 0000h



注意: 外部唤醒线是边沿触发的，这些线上不能出现毛刺信号。

在写EXTI_RTSTR 寄存器时在外中断线上的上升沿信号不能被识别，挂起位不会被置位。在同一中断线上，可以同时设置上升沿和下降沿触发。即任一边沿都可触发中断。

下降沿触发选择寄存器 (EXTI_FTSR)

偏移地址：0CH

复位值：0000 0000h



位31:19	保留，必须始终保持为复位状态(0)。
位18:0	TRx : 线x上的下降沿触发事件配置位 0: 禁止输入线x上的下降沿触发(中断和事件) 1: 允许输入线x上的下降沿触发(中断和事件)

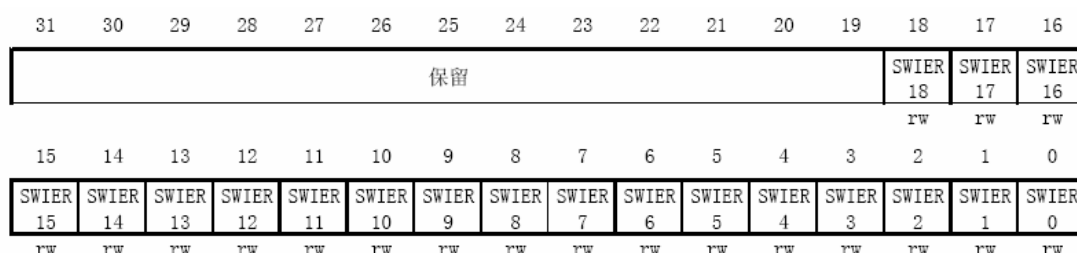
注意：外部唤醒线是边沿触发的，这些线上不能出现毛刺信号。

在写EXTI_RTSTR 寄存器时在外中断线上的下降沿信号不能被识别，挂起位不会被置位。在同一中断线上，可以同时设置上升沿和下降沿触发。即任一边沿都可触发中断。

软件中断事件寄存器 (EXTI_SWIER)

偏移地址：10h

复位值：0000 0000h

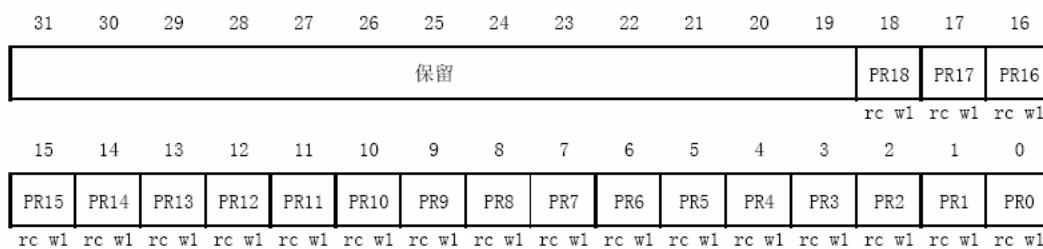


位31:19	保留，必须始终保持为复位状态(0)。
位18:0	SWIERx : 线x上的软件中断 当该位为0时，写1将设置EXTI_PR中相应的挂起位。如果在EXTI_IMR和EXTI_EMR中允许产生该中断，则此时将产生一个中断。 通过清除EXTI_PR的对应位(写入1)，可以清除该位为0。

挂起寄存器 (EXTI_PR)

偏移地址：14H

复位值：xxxx xxxh



位31:19	保留，必须始终保持为复位状态(0)。
位18:0	PRx: 挂起位 0: 没有发生触发请求 1: 发生了选择的触发请求 当在外部中断线上发生了选择的边沿事件，该位被置1。在该位中写入1可以清除它，也可以通过改变边沿检测的极性清除。 注：如果在进入停机模式前的一个周期发生了一个中断，则EXTI_PR寄存器将只在系统从停机模式退出后才被修改，并在EXTI_IMR寄存器中未屏蔽该中断时产生中断请求。

3.1 外部中断/事件寄存器映像

表2 外部中断/事件控制器寄存器映像和复位值

偏移	寄存器	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
000h	EXTI_IMR	保留													MR[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
004h	EXTI_EMR	保留													MR[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
008h	EXTI_RISR	保留													TR[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00Ch	EXTI_FTSR	保留													TR[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
010h	EXTI_SWIER	保留													SWIER[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
014h	EXTI_PR	保留													PR[18:0]																								
	复位值														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

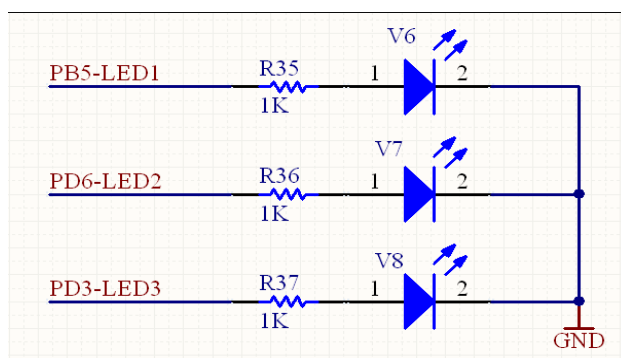
4. 应用实例

4.1. 设计要求

开发板上有3个蓝色状态指示灯V6(LED1),V7(LED2),V8(LED3),通过对应的按键K1-K3 ,控制LED的亮灭,将PE2引脚配置为外部中断,当其上出现下降沿时产生一个中断,根据扫描PC5 , PC2 , PC3来判别是哪个按键按下。

4.2 硬件电路设计

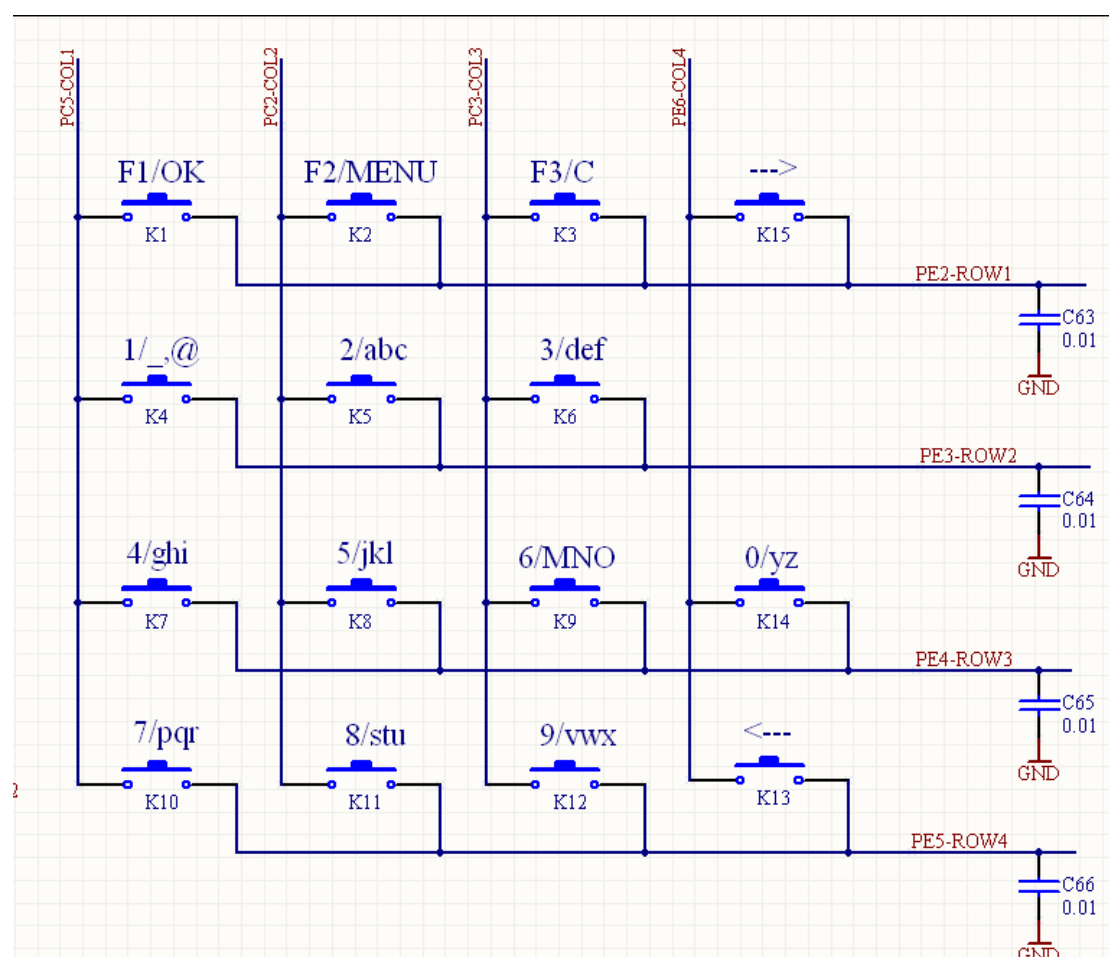
在开发板上V6、V7、V8分别与MCU的PB5、PD6、PD3相连,如下图所示



键盘部分如下图所示：

例程所用到的列扫描线：PC5,PC2,PC3。

例程所用到的行扫描线（EXTI中断线）：PE2。

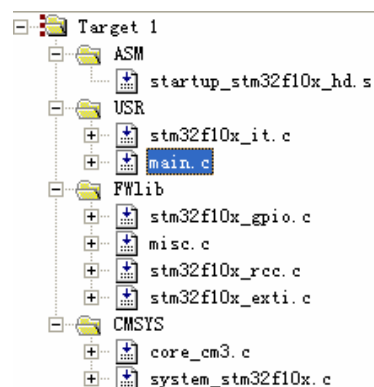


4.3 软件程序设计

根据任务要求，程序内容主要包括：

1. 配置NVIC控制器、EXTI控制器，将PE2引脚配置为EXTI2，下降沿触发
2. 键盘扫描

整个工程包含4类源文件：



ASM--startup_stm32f10x_hd.s 由于奋斗板采用的是STM32F103大存储器芯片，因此采用STM32标准库自带的大存储器芯片启动代码，这个文件已经配置好了初始状态，以及中断向量表。可以直接在工程里使用，如果你在以后的应用中采用了中存储器或者小存储器STM32芯片，可以将启动代码换为startup_stm32f10x_md.s 或者 startup_stm32f10x_ld.s。

FWLIB--stm32f10x_gpio.c ST公司的标准库，包含了关于对通用IO口设置的函数。

stm32f10x_rcc.c ST公司的标准库，包含了关于对系统时钟设置的函数。

CMSYS—是关于CORETEX-M3平台的系统函数及定义

USER—main.c 例程的主函数。

USER--stm32f10x_it.c 中断服务程序

```
// _____ 主程序 _____

int main(void)
{
    unsigned char a=0,b=0,c=0;

    /*完成对系统时钟的设置，例程中通过系统时钟设置函数，外接晶振采用8Mhz，经过片内频率合成，9倍频，设置为72Mhz的时钟。*/
    RCC_Configuration();

    /*嵌套向量中断控制器
       说明了EXTI2 抢占优先级0（最多1位），和子优先级0（最多7位）*/
    NVIC_Configuration();

    /*对控制3个LED指示灯的IO口进行了初始化，将3个端口配置为推挽上拉输出，口线速度为50Mhz。将中断线PE2配置为输入模式。将键盘扫描列
       线PC5, PC2, PC3设置为推挽上拉输出。在配置某个口线时，首先应对它所在的端口的时钟进行使能。否则无法配置成功，由于用到了端口
       B和端口D, C, D, 因此要对这4个端口的时钟进行使能，同时由于用到复用IO口功能用于配置外部中断。因此还要使能AFIO(复用功能IO)
       时钟。*/
    GPIO_Configuration();

    //用于配置AFIO外部中断配置寄存器AFIO_EXTICR1，用于选择EXTI2外部中断的输入源是PE2。
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);

    EXTI_InitStructure.EXTI_Line = EXTI_Line2;      //PE2 作为键盘的行线。检测状态
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //中断模式
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //下降沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
```

```

EXTI_Init(&EXTI_InitStructure);

/* 置3根键盘列扫描线为0，以便在按键按下时产生中断
GPIO_ResetBits(GPIOC, GPIO_Pin_2);
GPIO_ResetBits(GPIOC, GPIO_Pin_3);
GPIO_ResetBits(GPIOC, GPIO_Pin_5);
while (1)
{
    GPIO_ResetBits(GPIOC, GPIO_Pin_2);
    GPIO_ResetBits(GPIOC, GPIO_Pin_3);
    GPIO_ResetBits(GPIOC, GPIO_Pin_5);

    numm();                                //键盘扫描程序    判断是哪个键按下
    if(num==1&&a==0){GPIO_ResetBits(GPIOB, GPIO_Pin_5);a=1;}          //K1 按下作处理
    else if(num==1&&a==1){GPIO_SetBits(GPIOB, GPIO_Pin_5);a=0;}
    if(num==2&&b==0){GPIO_ResetBits(GPIOD, GPIO_Pin_6);b=1;}          //K2 按下作处理
    else if(num==2&&b==1){GPIO_SetBits(GPIOD, GPIO_Pin_6);b=0;}
    if(num==3&&c==0){GPIO_ResetBits(GPIOD, GPIO_Pin_3);c=1;}          //K3 按下作处理
    else if(num==3&&c==1){GPIO_SetBits(GPIOD, GPIO_Pin_3);c=0;}
}

// _____ 键盘扫描程序 _____
void numm(void){
    num=0;
    if(_it0==1){                                //按键按下标志
        GPIO_ResetBits(GPIOC, GPIO_Pin_5);          //置PC5为0。
        GPIO_SetBits(GPIOC, GPIO_Pin_2);
        GPIO_SetBits(GPIOC, GPIO_Pin_3);
        if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){          //K1
            Delay(0xff);
            if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){          //按键消抖动
                while(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0); //是否松开按键
                num=1;          //键值1 为K1按下
                goto n_exit;
            }
        }
    }

    GPIO_SetBits(GPIOC, GPIO_Pin_5);
    GPIO_ResetBits(GPIOC, GPIO_Pin_2);          //置PC2为0
    GPIO_SetBits(GPIOC, GPIO_Pin_3);
    Delay(0xff);
    if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){          //K2
        Delay(0xff);

```

```


        if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){           //按键消抖动
            while(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0);    //是否松开按键
            num=2;                                                  //键值2 为K2按下
            goto n_exit;
        }
    }

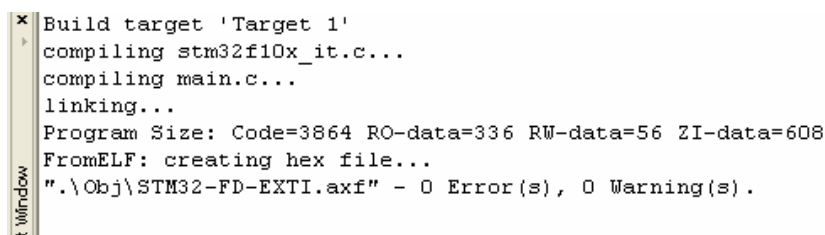
    GPIO_SetBits(GPIOC, GPIO_Pin_5);
    GPIO_SetBits(GPIOC, GPIO_Pin_2);
    GPIO_ResetBits(GPIOC, GPIO_Pin_3);                             //置PC3为0
    Delay(0xff);
    if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){               //K3
        Delay(0xff);
        if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0){           //按键消抖动
            while(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0);    //是否松开按键
            num=3;                                                  //键值3 为K3按下
            goto n_exit;
        }
    }
    n_exit::
    _it0=0;
}

// _____ 键盘中断服务程序 _____
/*键盘中断 (EXTI2) 服务程序*/
void EXTI2_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line2) != RESET)                      //判别是否有键按下
    {
        _it0=1;            //按键中断标志
        EXTI_ClearITPendingBit(EXTI_Line2);                       //清除中断请求标志
    }
}

```

5 运行过程

按  编译工程，完成后会提示如下。



```

x Build target 'Target 1'
  compiling stm32f10x_it.c...
  compiling main.c...
  linking...
  Program Size: Code=3864 RO-data=336 RW-data=56 ZI-data=608
  FromELF: creating hex file...
  ".\Obj\STM32-FD-EXTI.axf" - 0 Error(s), 0 Warning(s).

```

调试例程无问题后，可以通过JLINK V8或者串口将代码写入板子，具体的烧写步骤，参考奋斗板文档目录下的《奋斗版STM32开发板JTAG下载步骤》或者《奋斗版STM32开发板串口下载步骤》，板子上电复位后，按动K1-K3 可分别控制相应的LED1-LED3亮灭。