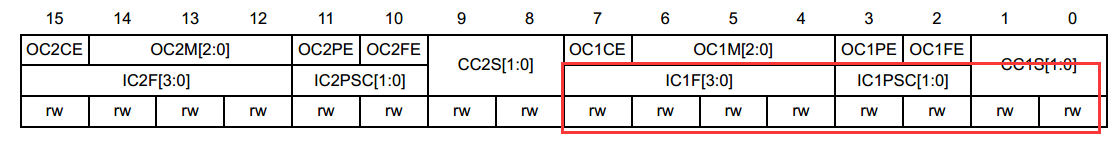
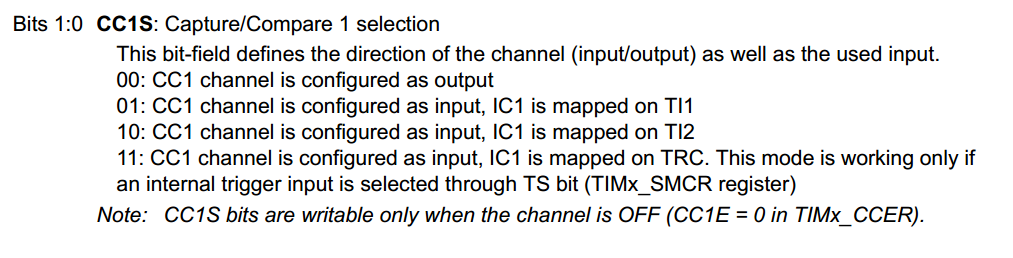
输入捕获模式可以用来测量脉冲宽度或者测量频率。STM32的定时器，除了TIM6和TIM7，其他定时器都有输入捕获功能。STM32 的输入捕获，简单地说就是通过检测 TIMx\_CHx 上的边沿信号，在边沿信号发生跳变（比如上升沿/下降沿）的时候，将当前定时器的值（TIMx\_CNT）存放到对应的通道的捕获/比较寄存器（TIMx\_CCRx）中。

1. 相关寄存器介绍

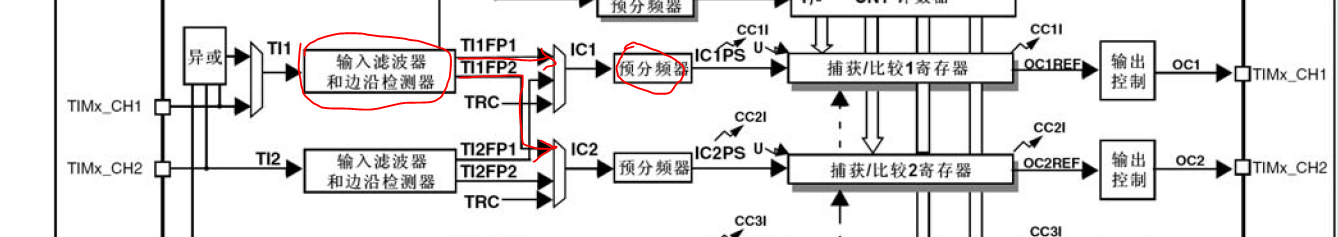
1） 捕获/比较模式寄存器 (TIMx\_CCMRx)



当在输入捕获模式下使用的时候，对应上图的第二行描述，从图中可以看出，TIMx\_CCMR1 明显是针对 第2 个通道的配置，低八位[7：0]用于捕获/比较通道 1 的控制，而高八位[15：8]则用于捕获/比较通道 2 的控制。同理，TIMx\_CCMR2 这个寄存器用来控制通道 3 和通道 4。今天的实验，我们用到的是 TIM5 的捕获/比较通道 1。



 （1）CC1S[1:0]，这两个位用于 CCR1 的通道配置，这里我们设置 CC1S[1:0]=01，也就是配置 IC1 映射在 TI1 上。如果不理解没有关系，看图就知道了。



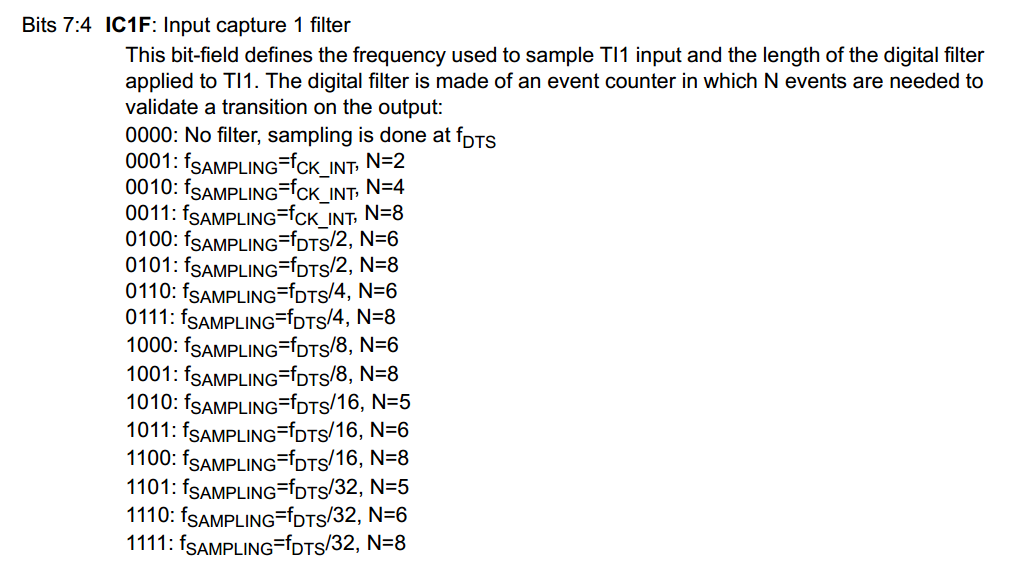
从图中可以知道，IC1有三个输入源，分别是TI1，TI2，TRC。

TIM5\_ICInitStructure.TIM\_ICSelection = TIM\_ICSelection\_DirectTI; 这是通过库函数配置，映射到TI1。

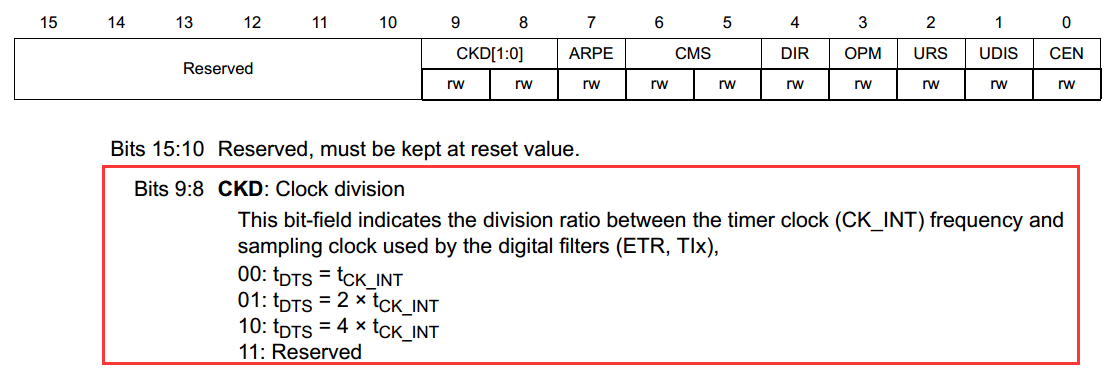
头文件（stm32f10x\_tim.h）定义如下：

#define TIM\_ICSelection\_DirectTI           ((uint16\_t)0x0001)   
#define TIM\_ICSelection\_IndirectTI         ((uint16\_t)0x0002)   
#define TIM\_ICSelection\_TRC                 ((uint16\_t)0x0003)

（2）IC1F[3:0]



表格中的fCK\_INT 是定时器的输入频率（TIMxCLK），fDTS则是根据 TIMx\_CR1 的 CKD[1:0]来确定。



上图就是TIMx\_CR1寄存器。

#define TIM\_CKD\_DIV1                       ((uint16\_t)0x0000)  
#define TIM\_CKD\_DIV2                       ((uint16\_t)0x0100)  
#define TIM\_CKD\_DIV4                       ((uint16\_t)0x0200)

假定选IC1F[3:0]=0101，即采样频率为fDTS/2, N=8；

选CKD[1:0]=01，即 tDTS = 2 × tCK\_INT；并假定 fCK\_INT=72MHz。

这时可以算出，采样频率为72MHz/2/2=18MHz.

1/(18MHz)\*8 = 0.4444us = 444.4ns.

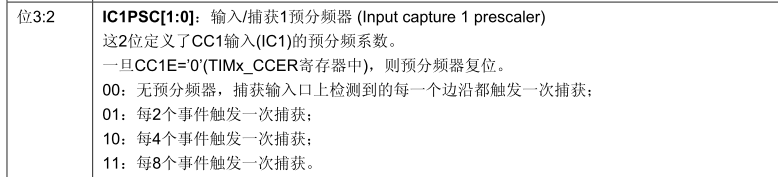
因为N=8，所以滤波器可以滤除掉输入端小于444.4ns的脉冲。

“数字滤波器由一个事件计数器组成，它记录到N个事件后会产生一个输出的跳变”，这句话可以这样理解：

当滤波器连续采样到N个有效电平时，就输出这个有效电平。   
当滤波器没有连续采样到N个有效电平时，再从0开始计数，输出一直保持上一次输出的有效电平。

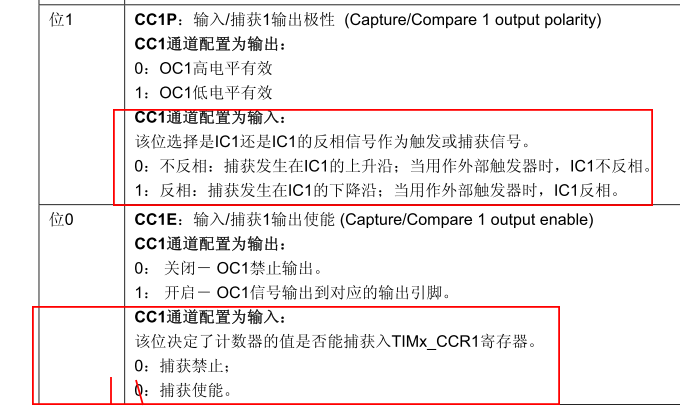
例如：滤波器上一次输出是高电平，本次连续采样到（N-1）个高电平，但第N个是个低电平，那么滤波器仍然保持上次输出的高电平，并重新开始计数，记录1次低电平，如果其后采样的（N-1）个也是低电平，此时滤波才输出低电平，于是一个下降沿才出现在IC1上。

（3）IC1PSC[1:0]



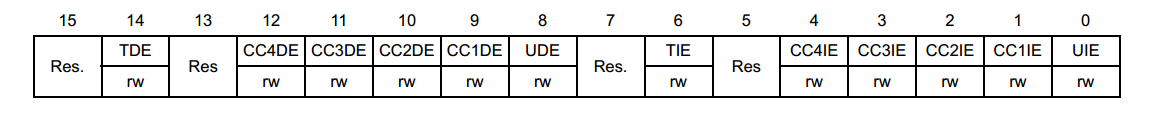
这里的“事件”指的是上升沿或者下降沿，具体是哪个由TIMx\_CCER寄存器的CCxP位决定。

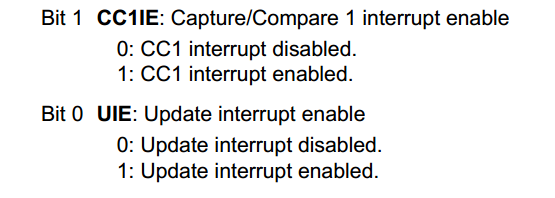
2）捕获/比较使能寄存器：TIMx\_CCER



所以，要使能输入捕获，必须设置 CC1E=1，而 CC1P 则根据自己的需要来配置（捕获发生在上升沿还是下降沿）。

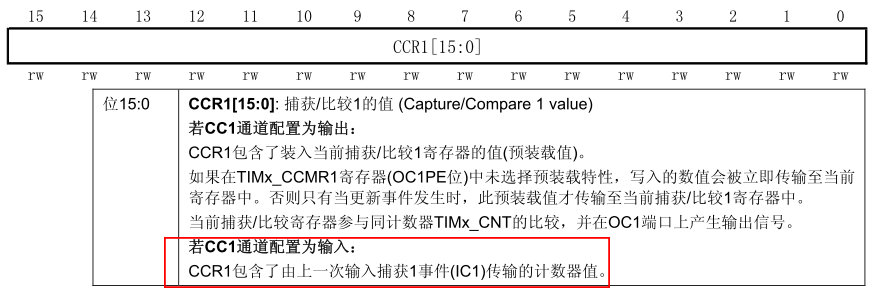
3） DMA/中断使能寄存器(TIMx\_DIER)





在实验中，我们需要允许位0和位1的中断。

4） 捕获/比较寄存器 (TIMx\_CCRx)



该寄存器用来存储捕获发生时，TIMx\_CNT的值，我们从 TIMx\_CCR1 就可以读出通道 1 捕获发生时刻的 TIMx\_CNT 值，通过两次捕获（一次上升沿捕获，一次下降沿捕获）的差值，就可以计算出高电平脉冲的宽度。

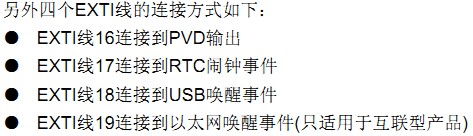
STM32的**EXTI控制器**支持**19 个外部中断/ 事件**请求。每个中断设有状态位，每个中断/ 事件都有独立的触发和屏蔽设置。  
STM32的19个外部中断对应着**19路中断线**，分别是**EXTI\_Line0-EXTI\_Line18**：

线0~15：对应外部 IO口的输入中断。  
线16：连接到 PVD 输出。  
线17：连接到 RTC 闹钟事件。  
线18：连接到 USB 唤醒事件。

**触发方式**：STM32 的外部中断是**通过边沿来触发**的，**不支持电平触发**。

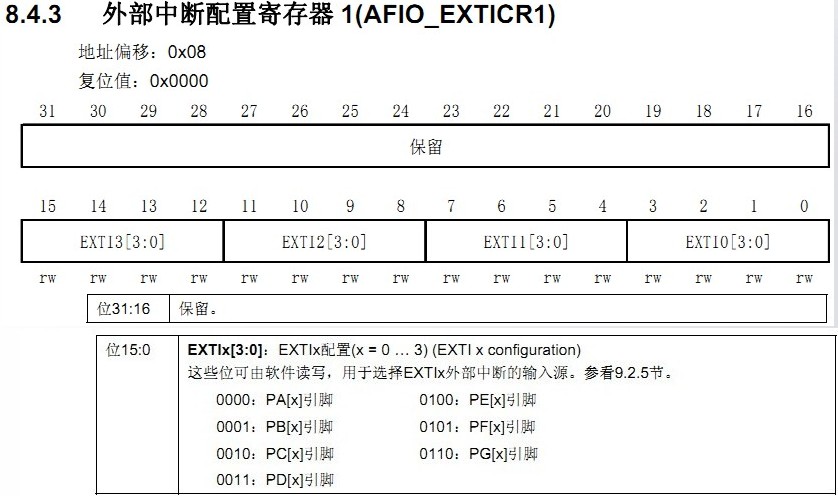
**外部中断分组：**STM32 的**每一个GPIO都能配置成一个外部中断触发源**，STM32 通过根据引脚的序号不同将众多中断触发源分成不同的组，比如：PA0，PB0，PC0，PD0，PE0，PF0，PG0为第一组，那么依此类推，我们能得出一共有16 组，STM32 规定，每一组中同时只能有一个中断触发源工作，那么，最多工作的也就是16个外部中断。

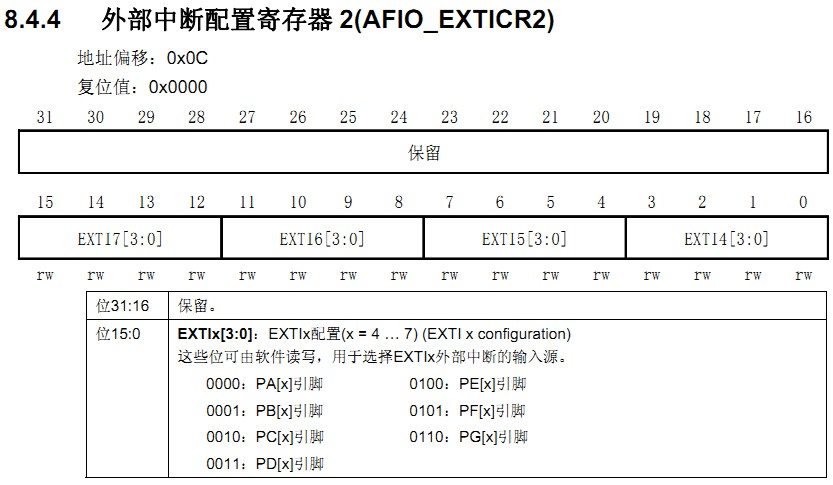


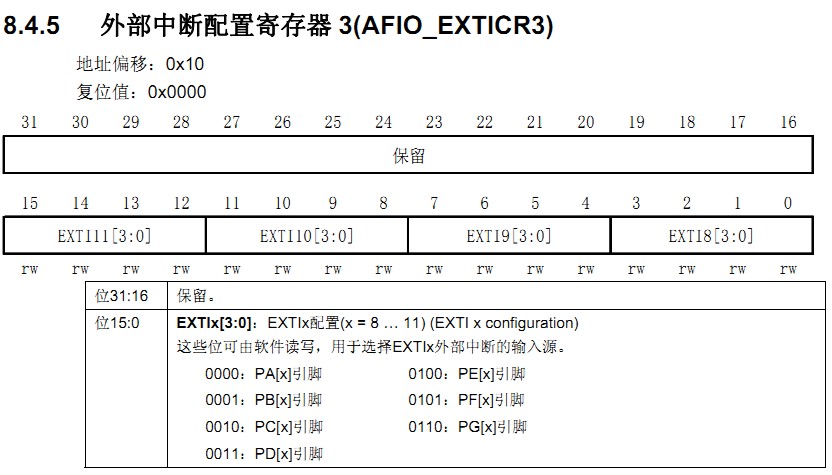


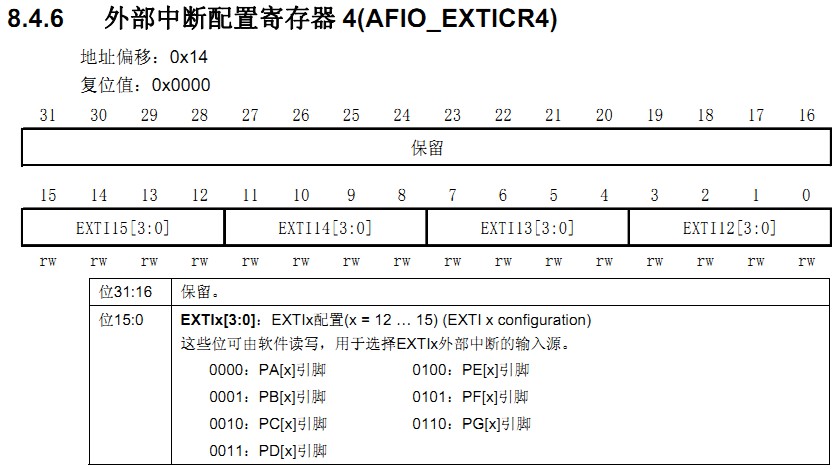
EXTICR寄存器组，总共有4 个，因为编译器的寄存器组都是从0 开始编号的，所以**EXTICR[0]~ EXTICR[3]，对应《STM32参考手册》里的 EXTICR1~ EXTICR 4（查了好久才搞明白这个数组的含义！！）**。每个 EXTICR只用了其低16 位。

EXTICR[0] ~EXTICR[3]的分配如下：









# EXTI寄存器的结构体：

typedef struct

{

vu32 IMR;

vu32 EMR;

vu32 RTSR;

vu32 FTSR;

vu32 SWIER;

vu32 PR;

} EXTI\_TypeDef;

**IMR：中断屏蔽寄存器**

这是一个 32 寄存器。但是只有前 19 位有效。当位 x 设置为1 时，则开启这个线上的中断，否则关闭该线上的中断。

**EMR：事件屏蔽寄存器**

同IMR ，只是该寄存器是针对事件的屏蔽和开启。

**RTSR：上升沿触发选择寄存器**

该寄存器同IMR ，也是一个32为的寄存器，只有前 19位有效。位 x 对应线x 上的上升沿触发，如果设置为 1 ，则是允许上升沿触发中断/ 事件。否则，不允许。

**FTSR：下降沿触发选择寄存器**

同 PTSR，不过这个寄存器是设置下降沿的。下降沿和上升沿可以被同时设置，这样就变成了任意电平触发了。

**SWIER：软件中断事件寄存器**

通过向该寄存器的位x 写入 1 ，在未设置 IMR 和EMR的时候，将设置PR中相应位挂起。如果设置了IMR 和EMR时将产生一次中断。被设置的SWIER位，将会在PR中的对应位清除后清除。

**PR：挂起寄存器**

0 ，表示对应线上没有发生触发请求。

1，表示外部中断线上发生了选择的边沿事件。通过向该寄存器的对应位写入 1 可以清除该位。

在中断服务函数里面经常会要向该寄存器的对应位写1 来清除中断请求。

# Ex\_NVIC\_Config基本是按照这个结构来编写的

# 中断配置步骤

STM32的每个IO口都可以作为中断输入，这点很好用。要把IO口作为外部中断输入，有以下几个步骤：

**1）初始化IO口为输入。**

这一步设置你要作为外部中断输入的IO口的状态，可以设置为上拉/下拉输入，也可以设置为浮空输入，但浮空的时候外部一定要带上拉，或者下拉电阻。否则可能导致中断不停的触发。在干扰较大的地方，就算使用了上拉/下拉，也建议使用外部上拉/下拉电阻，这样可以一定程度防止外部干扰带来的影响。

**2）开启IO口复用时钟，设置IO口与中断线的映射关系。**

STM32的IO口与中断线的对应关系需要配置外部中断配置寄存器EXTICR，这样我们要先开启复用时钟，然后配置IO口与中断线的对应关系。才能把外部中断与中断线连接起来。  
**3）开启与该IO口相对的线上中断/事件，设置触发条件。**  
这一步，我们要配置中断产生的条件，STM32可以配置成上升沿触发，下降沿触发，或者任意电平变化触发，但是不能配置成高电平触发和低电平触发。这里根据自己的实际情况来配置。同时要开启中断线上的中断，这里需要注意的是：如果使用外部中断，并设置该中断的EMR位的话，会引起软件仿真不能跳到中断，而硬件上是可以的。而不设置EMR，软件仿真就可以进入中断服务函数，并且硬件上也是可以的。建议不要配置EMR位。  
**4）配置中断分组（NVIC），并使能中断。**  
这一步，我们就是配置中断的分组，以及使能，对STM32的中断来说，只有配置了NVIC的设置，并开启才能被执行，否则是不会执行到中断服务函数里面去的。关于NVIC的详细介绍，请参考前面章节。  
**5）编写中断服务函数。**

这是中断设置的最后一步，中断服务函数，是必不可少的，如果在代码里面开启了中断，但是没编写中断服务函数，就可能引起硬件错误，从而导致程序崩溃！所以在开启了某个中断后，一定要记得为该中断编写服务函数。在中断服务函数里面编写你要执行的中断后的操作。

**实验4--外部中断实验exit.c函数如下：**

1. #include "exti.h"
2. #include "led.h"
3. #include "key.h"
4. #include "delay.h"
5. #include "usart.h"
7. //外部中断0服务程序
8. **void** EXTI0\_IRQHandler(**void**)
9. {
10. delay\_ms(10);//消抖
11. **if**(KEY2==1)  //按键2
12. {
13. LED0=!LED0;
14. LED1=!LED1;
15. }
16. EXTI->PR=1<<0;  //清除LINE0上的中断标志位
17. }
19. //外部中断15~10服务程序
20. **void** EXTI15\_10\_IRQHandler(**void**)
21. {
22. delay\_ms(10);    //消抖
23. **if**(KEY0==0)      //按键0
24. {
25. LED0=!LED0;
26. }**else** **if**(KEY1==0)//按键1
27. {
28. LED1=!LED1;
29. }
30. EXTI->PR=1<<13;     //清除LINE13上的中断标志位
31. EXTI->PR=1<<15;     //清除LINE15上的中断标志位
32. }
33. //外部中断初始化程序
34. //初始化PA0,PA13,PA15为中断输入.
35. **void** EXTIX\_Init(**void**)
36. {
37. RCC->APB2ENR|=1<<2;     //使能PORTA时钟
38. JTAG\_Set(JTAG\_SWD\_DISABLE);//关闭JTAG和SWD
40. GPIOA->CRL&=0XFFFFFFF0;//PA0设置成输入
41. GPIOA->CRL|=0X00000008;
42. GPIOA->CRH&=0X0F0FFFFF;//PA13,15设置成输入
43. GPIOA->CRH|=0X80800000;
44. GPIOA->ODR|=1<<13;    //PA13上拉,PA0默认下拉
45. GPIOA->ODR|=1<<15;    //PA15上拉
47. Ex\_NVIC\_Config(GPIO\_A,0,RTIR); //上升沿触发
48. Ex\_NVIC\_Config(GPIO\_A,13,FTIR);//下降沿触发
49. Ex\_NVIC\_Config(GPIO\_A,15,FTIR);//下降沿触发
51. MY\_NVIC\_Init(2,2,EXTI0\_IRQChannel,2);    //抢占2，子优先级2，组2
52. MY\_NVIC\_Init(2,1,EXTI15\_10\_IRQChannel,2);//抢占2，子优先级1，组2
53. }

其中的两个函数:**Ex\_NVIC\_Config(GPIO\_A,0,RTIR)**;和**MY\_NVIC\_Init(2,2,EXTI0\_IRQChannel,2)**;这两个函数都是在sys.c里定义，分别完成了步骤2、3、4.函数原型如下：

**[cpp]** [view plain](http://blog.csdn.net/zzwdkxx/article/details/9036679) [copy](http://blog.csdn.net/zzwdkxx/article/details/9036679)

1. //外部中断配置函数
2. //只针对GPIOA~G;不包括PVD,RTC和USB唤醒这三个
3. //参数:GPIOx:0~6,代表GPIOA~G;BITx:需要使能的位;TRIM:触发模式,1,下升沿;2,上降沿;3，任意电平触发
4. //该函数一次只能配置1个IO口,多个IO口,需多次调用
5. //该函数会自动开启对应中断,以及屏蔽线
6. //待测试...
7. **void** Ex\_NVIC\_Config(u8 GPIOx,u8 BITx,u8 TRIM)
8. {
9. u8 EXTADDR;
10. u8 EXTOFFSET;
11. EXTADDR=BITx/4;//得到中断寄存器组的编号
12. EXTOFFSET=(BITx%4)\*4;
14. RCC->APB2ENR|=0x01;//使能io复用时钟
16. AFIO->EXTICR[EXTADDR]&=~(0x000F<<EXTOFFSET);//清除原来设置！！！
17. AFIO->EXTICR[EXTADDR]|=GPIOx<<EXTOFFSET;//EXTI.BITx映射到GPIOx.BITx
19. //自动设置
20. EXTI->IMR|=1<<BITx;//  开启line BITx上的中断
21. //EXTI->EMR|=1<<BITx;//不屏蔽line BITx上的事件 (如果不屏蔽这句,在硬件上是可以的,但是在软件仿真的时候无法进入中断!)
22. **if**(TRIM&0x01)EXTI->FTSR|=1<<BITx;//line BITx上事件下降沿触发
23. **if**(TRIM&0x02)EXTI->RTSR|=1<<BITx;//line BITx上事件上升降沿触发
24. }

**这个函数完成了两个步骤：**

**2、开启IO口复用时钟，设置IO口与中断线的映射关系**

**3、开启与该IO口相对的线上的中断/时间，设置触发条件**

**[cpp]** [view plain](http://blog.csdn.net/zzwdkxx/article/details/9036679) [copy](http://blog.csdn.net/zzwdkxx/article/details/9036679)

1. //设置NVIC
2. //NVIC\_PreemptionPriority:抢占优先级
3. //NVIC\_SubPriority       :响应优先级
4. //NVIC\_Channel           :中断编号
5. //NVIC\_Group             :中断分组 0~4
6. //注意优先级不能超过设定的组的范围!否则会有意想不到的错误
7. //组划分:
8. //组0:0位抢占优先级,4位响应优先级
9. //组1:1位抢占优先级,3位响应优先级
10. //组2:2位抢占优先级,2位响应优先级
11. //组3:3位抢占优先级,1位响应优先级
12. //组4:4位抢占优先级,0位响应优先级
13. //NVIC\_SubPriority和NVIC\_PreemptionPriority的原则是,数值越小,越优先
14. //CHECK OK
15. //100329
16. **void** MY\_NVIC\_Init(u8 NVIC\_PreemptionPriority,u8 NVIC\_SubPriority,u8 NVIC\_Channel,u8 NVIC\_Group)
17. {
18. u32 temp;
19. u8 IPRADDR=NVIC\_Channel/4;  //每组只能存4个,得到组地址
20. u8 IPROFFSET=NVIC\_Channel%4;//在组内的偏移
21. IPROFFSET=IPROFFSET\*8+4;    //得到偏移的确切位置
22. MY\_NVIC\_PriorityGroupConfig(NVIC\_Group);//设置分组
23. temp=NVIC\_PreemptionPriority<<(4-NVIC\_Group);
24. temp|=NVIC\_SubPriority&(0x0f>>NVIC\_Group);
25. temp&=0xf;//取低四位
27. **if**(NVIC\_Channel<32)NVIC->ISER[0]|=1<<NVIC\_Channel;//使能中断位(要清除的话,相反操作就OK)
28. **else** NVIC->ISER[1]|=1<<(NVIC\_Channel-32);
29. NVIC->IPR[IPRADDR]|=temp<<IPROFFSET;//设置响应优先级和抢断优先级

**这个函数完成了：**

**4、配置中断分组（NVIC），并使能中断**

# 补充

## 在实验18--触摸屏实验中，中断初始化没有调用这个函数，它是这样配置的：

**[cpp]** [view plain](http://blog.csdn.net/zzwdkxx/article/details/9036679) [copy](http://blog.csdn.net/zzwdkxx/article/details/9036679)

1. MY\_NVIC\_Init(2,0,EXTI1\_IRQChannel,2);
2. RCC->APB2ENR|=0x01;    //使能io复用时钟
3. AFIO->EXTICR[0]|=0X0020; //EXTI1映射到PC1（这句原子的程序里注释错了搞成了EXTI13）
4. EXTI->IMR|=1<<1;        //开启line1上的中断
5. EXTI->EMR|=1<<1;        //不屏蔽line1上的事件
6. EXTI->FTSR|=1<<1;       //line1上事件下降沿触发

        RCC->APB2ENR|=0x01;  这一句是开启复用时钟，**什么时候需要开启复用时钟**？手册有这样一段：



       也就是说只要操作**EVCR、EXTICRX、MAPR**的时候,就必须开启复用功能时钟，即当你要配置stm32的事件输出、外部中断、重映射的时候.就必须开启复用时钟。

         AFIO->EXTICR[0]|=0X0020; //EXTI1映射到PC1

         这一句设置中断映射，如上文所说EXTICR[0]~ EXTICR[3] 对应 EXTICR1~ EXTICR4，举例：

         AFIO->EXTICR[3] &= 0xFFFFFF0F; 

         AFIO->EXTICR[3] |= 0xFFFFFF0F; //EXTI13映射到PA13，0（即0x00）代表A口,1（即0x01）代表B口,依次类推,6（即0x0110）代表G口.

         AFIO->EXTICR[3] &= 0xFFFFFF0F;

         AFIO->EXTICR[3] |= 0xFFFFFF2F; //EXIT13映射到PC13，2（0x0010）代表C口

**外部中断函数不能进入的原因分析分析，可能为以下几个方面：**

1）GPIO或者AFIO的时钟没有开启；

2）GPIO和配置的中断线路不匹配；

3）中断触发方式和实际不相符合；

4）中断处理函数用库函数时，写错，经常可能出现数字和字母之间没有下划线；

5）外部中断是沿触发，有可能检测不到沿，比如中断线是低电平（浮空输入），触发是下降沿触发，可能会出现一直是低电平,高电平的时候是一样的情况，电平持续         为高电平；

6）没有用软件中断来触发外部中断，调用函数EXTI\_GenerateSWInterrupt；，因为软件中断先于边沿中断处理。