目录

第4章 STM32—Interrupted

4.1 中断的概述

核心(FPU)

函数调用：当 CPU 执行到函数调用语句时，CPU 就会跑到相应函数定义的地方去执行代码，执行完后会回来接着执行下面的代码。

入栈：保存走的位置和中间变量的值

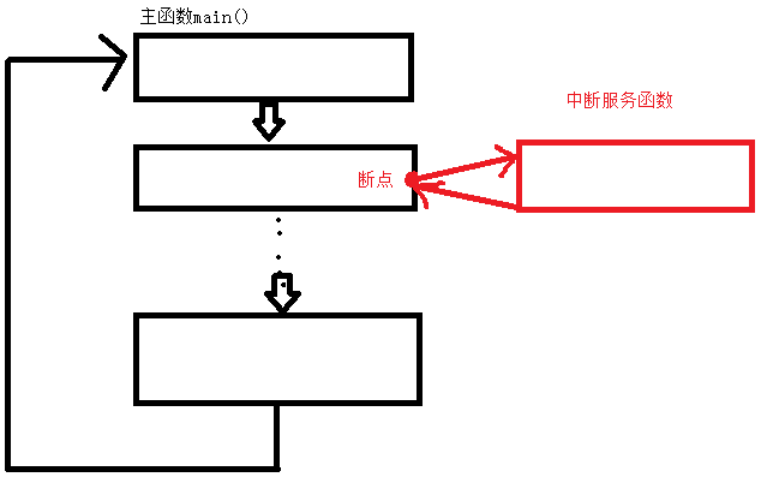
出栈：读取走的位置和变量的值

4.1.1 什么是中断？

中断就是程序在正常运行的过程中发生了不正常的事情，必须要暂停一下去处理这个不正常的事情，然后跑回来继续干正常的事情。

发生的条件：未知

正常运行的程序是主函数(main),代码是由 CPU 运行的。CPU 在主函数里运行是正常的执行过程，当在这个过程中突然发生了异常事件（中断），CPU 必须暂停当前的工作（设下断点），然后跑去能处理这个异常事件的函数中做异常处理（中断服务函数），处理完这个异常事件后（执行完中断服务函数），CPU 就会跑回刚才的断点处，继续正常运行下去。



函数调用和中断

不同点：发生的条件不一样(函数调用发生条件是已知，中断发生条件是未知)，函数调用可能有返回值。(中断没有返回值)

相同点：都需要 CPU 去执行，都需要入栈和出栈

4.1.2 中断入口

函数名就是函数的入口地址。

函数调用是通过函数名找到函数的入口地址

中断服务函数也是函数，中断服务函数名也就是该中断的入口地址。→ 51 单片机

51 单片机里中断服务函数名不是中断入口，中断服务函数名是可以用户自定义 void Timer(void) interrupt 0

在 stm32 里，中断服务函数名是不可自定义，所有的中断服务函数名都已经被写好了。(固定不变)

中断服务函数名在启动文件中” startup_stm32f40_41xxx.s”(407)不要手敲，直接复制

主函数和(自定义函数)子函数之间的关系？

调用关系。

主函数可以调用子函数

子函数可以调用子函数

子函数不可以调用主函数

主函数和中断服务函数的关系？平级

中断服务函数不是被主函数调用，不存在主函数调用中断服务函数的这个说法。当发生了异常事件，对应的中断服务函数直接从主函数手上抢走 CPU(NVIC)，执行完后再还回去。

所以，在裸机里面，中断服务函数的返回值类型和形参都是空的。

举例：usart1 的中断服务函数的写法

```
void USART1_IRQHandler (void)
{

}
```

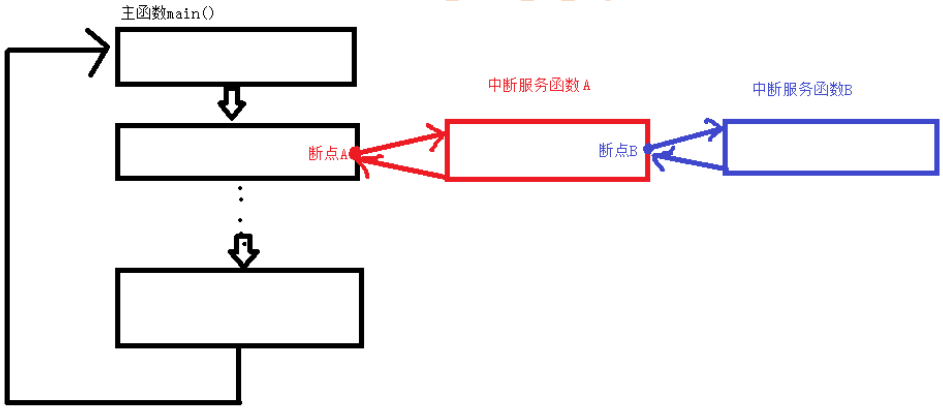
4.1.3 中断优先级

中断优先级就是处理当同时发生了多个异常事件时，CPU 该先处理谁的问题。

中断优先级以数字作为编号，数值越小优先级越高

0>1>2 以此类推

4.1.4 中断嵌套



如上图所示，在主函数执行过程中发生了异常事件 A，CPU 就会跑到中断服务函数 A 中执行，在执行中断服务函数 A 的过程中又发生了比异常事件 A 更加紧急的异常事件 B，那么 CPU 就会跑到中断服务函数 B 中执行，执行完中断服务函数 B 后才回去断点 B，执行完中断服务函数 A 后才回去断点 A

4.2 ARM 单片机中断体系

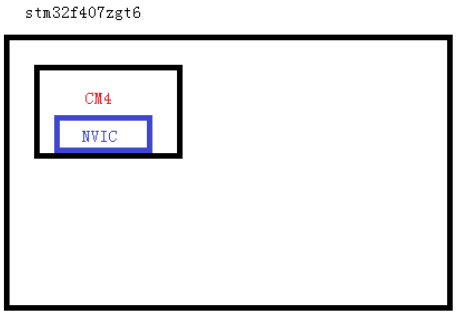
ARM:是一家公司名，也是一系列单片机的统称，在 ARM 架构下的单片机所采用的中断体系都是同一套。

M0 M3 M4 M7

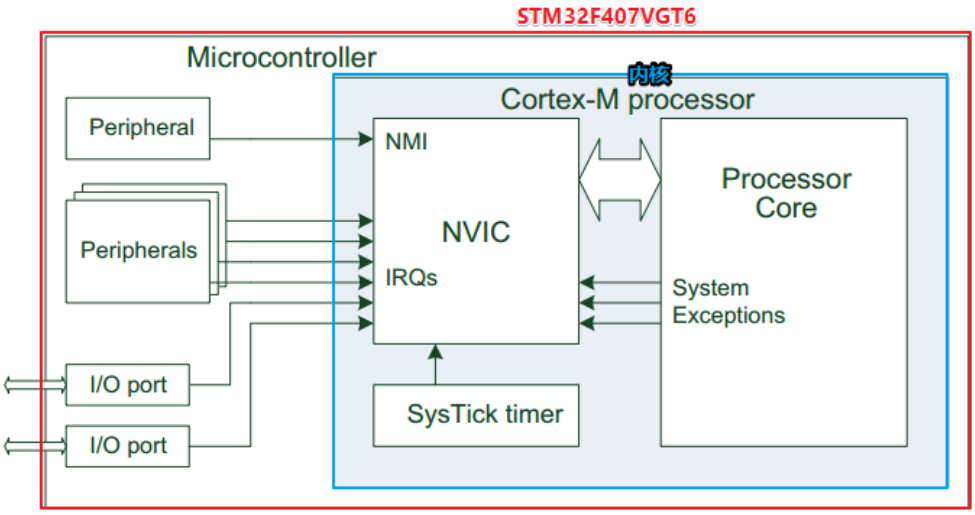
ST：意法半导体，芯片厂商，它做的芯片采用的就是 ARM 内核。

所以，当前我们使用的 stm32f407vgt6，使用的就是 ARM 的内核，使用的中断体系也是 ARM 的中断体系。

中断体系就是管理中断的一套机制。在内核里专门有一个管理中断机制的模块---NVIC 控制器



In a typical Cortex-M microcontroller, the NVIC receives interrupt requests from various sources, as shown in Figure 7.1.



NVIC 控制器属于内核级的模块，专门做中断管理，包括中断响应，优先级的设置，相关中断的使能....

4.2.1 NVIC 优先级说明

每一个中断源都有属于自己的三种优先级（抢占优先级、响应优先级、自然优先级）
每一个片上外设就是一个中断源
串口 1(中断源)：抢占优先级 2，响应优先级 4，自然优先级 44
定时器 2(中断源)：抢占优先级 2，响应优先级 4，自然优先级 35

对比规则

首先比较抢占优先级，如果抢占优先级一样，才会去比较响应优先级，如果响应优先级一样，才会去比较自然优先级。

抢占优先级：处理中断嵌套问题。当发送了异常事件 A 的时候，并且在处理事件 A 的过程中发生了比异常事件 A 的抢占优先级更高的异常事件 B，那么事件 B 抢占事件 A 的 CPU。

举例：
事件 A：抢占优先级 1
事件 B：抢占优先级 2
问：

1. 当同时发送异常事件时，CPU 先执行谁？
   CPU 先执行事件 A，执行完后接着执行事件 B
2. 当执行事件 A 的过程中发生了事件 B，CPU 怎么做？
   继续执行事件 A，执行完接着执行事件 B
3. 当执行事件 B 的过程中发生了事件 A，CPU 怎么做？
   事件 A 抢占 CPU，执行完后接着事件 B

响应优先级：处理同时发生多个异常事件时，并且这些事件的抢占优先级相同的情况下，CPU 使用权归谁的问题。

举例：
事件 A：抢占优先级 2 响应优先级 4
事件 B：抢占优先级 2 响应优先级 3 ---事件 B 的响应优先级更高并没有打断特性
问：
1. 当同时发送异常事件时，CPU 先执行谁？
   先执行事件 B，执行完接着事件 A
2. 当执行事件 A 的过程中发生了事件 B，CPU 怎么做？
   继续执行事件 A，执行完接着执行事件 B
3. 当执行事件 B 的过程中发生了事件 A，CPU 怎么做？
   继续执行事件 B，执行完接着执行事件 A

自然优先级：当同时发生多个异常事件时，并且这些事件的抢占优先级和响应优先级都相同的情况下，由自然优先级来决定 CPU 的使用权。

举例：
事件 A：抢占优先级 2 响应优先级 3 自然优先级 10
事件 B：抢占优先级 2 响应优先级 3 自然优先级 12
问：
1. 当同时发送异常事件时，CPU 先执行谁？
   先执行事件 A，执行完接着执行事件 B
2. 当执行事件 A 的过程中发生了事件 B，CPU 怎么做？
   继续执行事件 A，执行完接着执行事件 B
3. 当执行事件 B 的过程中发生了事件 A，CPU 怎么做？
   继续执行事件 B，执行完接着执行事件 A

总结：
1. 只要抢占优先级一样，就不会发生中断嵌套问题。
2. 抢占优先级和响应优先级都是用户自定义。自然优先级由厂家固定
3. 两个中断之间，抢占优先级和响应优先级可以设置成一样，但是自然优先级不可能一样。

4.2.2 NVIC 优先级分配方式

优先级分配方式 → 响应优先级和抢占优先级
抢占优先级和响应优先级是存放在哪里的？
在 CM4 里面，系统会给每一个中断源都分配一个 8 位寄存器来存放它的优先级（抢占优先级和响应优先级）。在这 8 位寄存器里面，一部分用于存放抢占优先级，另一部分存放响应优先级。
所以，就有以下分配方式的可能：优先级是以数字进行编号的

抢占优先级位数	响应优先级位数	抢占优先级范围	响应优先级范围
0	8	NONE	0~255
1	7	0~1	0~127
2	6	0~3	0~63
3	5	0~7	0~31
4	4	0~15	0~15

5	3	0~31	0~7
6	2	0~63	0~3
7	1	0~127	0~1
8	0	0~255	NONE

系统只允许一种分配方式。

在一个工程里面，只允许一种分配方式(就是抢占优先级的位数和响应优先级的位数)

选择好之后，这个工程里面的所有中断源的抢占优先级和响应优先级的分配方式都是一样

选择好了一种分配方式：抢占优先级位数 4　响应优先级位数 4

整个工程里面的所有中断源都是按照这种分配方式

如何决定分配方式？--优先级分组 → 确定响应优先级和抢占优先级的分配方式

一个系统在使用中断之前，必须确定优先级分组(抢占优先级的位数和响应优先级的位数)，也就是确定抢占优先级和响应优先级的可设置范围

优先级分组由 NVIC 控制器管理，设置优先级分组就是要找到 NVIC 相关寄存器。

NVIC 控制器是属于内核级模块，现在操作 NVIC 控制器就是对内核进行编程。之前的参考手册描述的是片上外设，所以 NVIC 控制器相关资料不在这个手册。需要查看内核相关的手册，比如《Cortex M3 与 M4 权威指南》或者《Cortex M3 权威指南(中文)》或者《STM32F3 与 F4 系列 Cortex M4 内核编程手册》。

表 D.13 应用程序中断及复位控制寄存器(AIRCR) 0xE000_ED0C

位段	名称	类型	复位值	描述
31:16	VECTKEY	RW	-	访问钥匙：任何对该寄存器的写操作，都必须同时把 0x05FA 写入此段，否则写操作被忽略。若读取此半字，则 0xFA05
15	ENDIANESS	R	-	指示端设置。1＝大端(BE8)，0＝小端。此值是在复位时确定的，不能更改。
10:8	PRIGROUP	R/W	0	优先级分组
2	SYSRESETREQ	W	-	请求芯片控制逻辑产生一次复位
1	VECTCLRACTIVE	W	-	清零所有异常的活动状态信息。通常只在调试时用，或者在 OS 从错误中恢复时用。
0	VECTRESET	W	-	复位 CM3 处理器内核（调试逻辑除外），但是此复位不影响芯片上在内核以外的电路

设置优先级分组

4.4.5 Application interrupt and reset control register (AIRCR)

Address offset: 0x0C

Reset value: 0xFA05 0000

Required privilege: Privileged

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANESS	Reserved				PRIGROUP			Reserved					SYS RESET REQ	VECT CLR ACTIVE	VECT RESET
r					rw	rw	rw						w	w	w

Bits 10:8 **PRIGROUP**: Interrupt priority grouping field

This field determines the split of group priority from subpriority, see *Binary point on page 213*.

Table 7.6 Definition of Pre-empt Priority Field and Sub-priority Field in a Priority-level Register in Different Priority Group Settings

PRIGROUP值 Priority Group	抢占优先级位数 Pre-empt Priority Field	响应优先级位数 Sub-priority Field
0 (default)	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

优先级分组	抢占优先级位数	响应优先级位数	抢占优先级设置范围	响应优先级设置范围
0	7	1	0~127	0~1
1	6	2	0~63	0~3
2	5	3	0~31	0~7
3	4	4	0~15	0~15
4	3	5	0~7	0~31
5	2	6	0~3	0~63
6	1	7	0~1	0~127
7	0	8	none	0~255

4.2.3 STM32FXX 中断体系

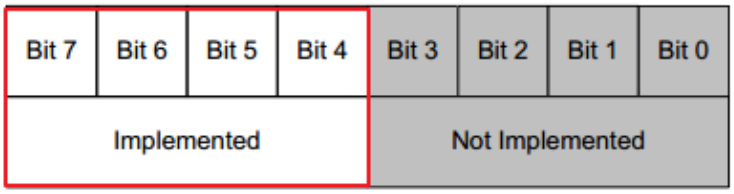
在 ARM 中断体系中，采用的是 8 位寄存器来存放优先级。但是，并不是所有采用 ARM 内核的芯片厂家都

把这 8 位用尽，一般都会使用 3~8 位。

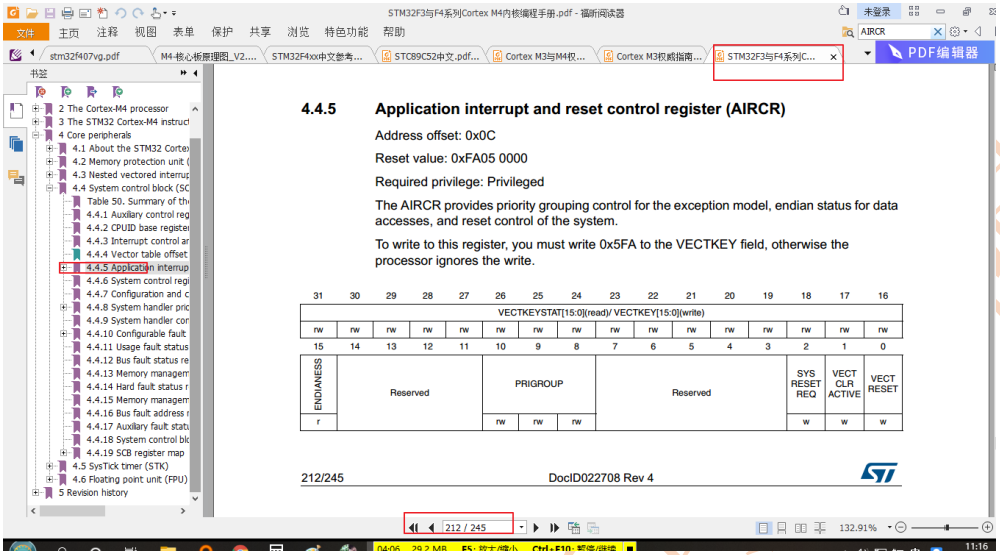
每一个中断源都有一个 8 位寄存器来存放响应优先级和抢占优先级

NXP 公司用 5 位(响应和抢占)，ST 公司用 4 位(响应和抢占)。

STM32 的抢占优先级位数和响应优先级位数一共 4 位



ST 公司用到了这 8 位寄存器的高四位来存放响应优先级和抢占优先级



4.4.5 **Application interrupt and reset control register (AIRCR)**

Address offset: 0x0C

Reset value: 0xFA05 0000

Required privilege: Privileged

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANESS	Reserved				PRIGROUP			Reserved					SYS RESET REQ	VECT CLR ACTIVE	VECT RESET
r					rw	rw	rw						w	w	w

212/245 DocID022708 Rev 4

Table 51. Priority grouping

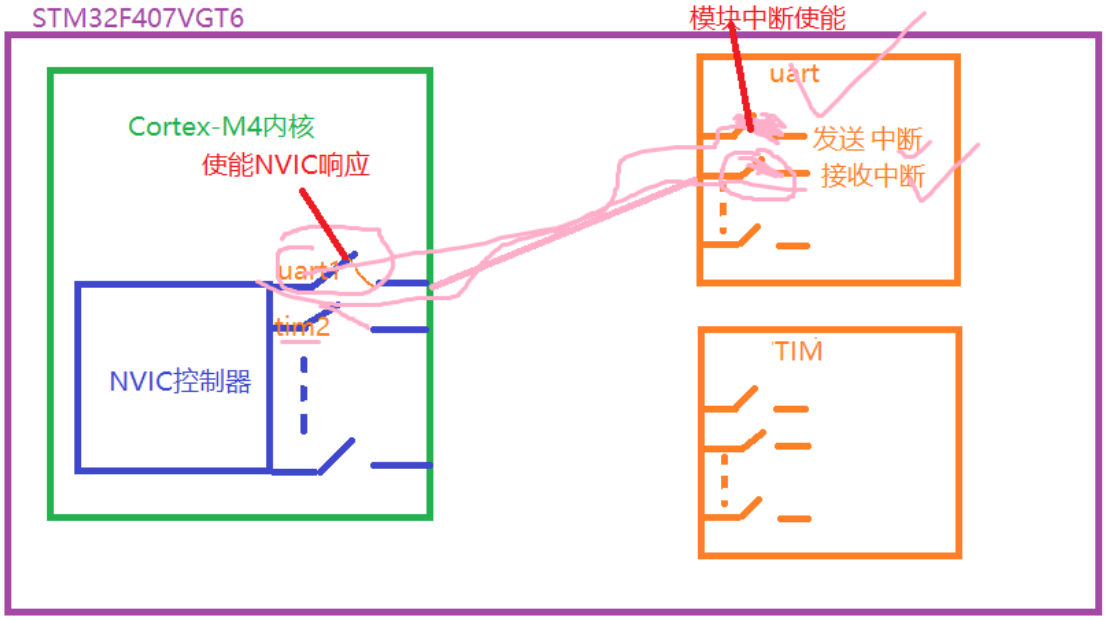
优先级分组值 PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point(1)	抢占优先级位数 Group priority bits	响应优先级位数 Subpriority bits	Group priorities	Sub priorities
0b011	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

这张表才是 STM32F407 需要看，所有 ST 公司生产的芯片，都看他

优先级分组	抢占优先级位数	响应优先级位数	抢占优先级设置范围	响应优先级设置范围
3	4	0	0~15	None
4	3	1	0~7	0~1
5	2	2	0~3	0~3
6	1	3	0~1	0~7
7	None	4	None	0~15

设置优先级分配方式的方法：

是把优先级分组值(3/4/5/6/7)，写到 AIRCR 寄存器[10:8]中，就可以确定抢占优先级位数和响应优先级位数，抢占优先级和响应优先级的设置范围也就确定了。



设置中断？uart1 接收中断为例

1. 设置优先级分组(确定抢占优先级和响应优先级的设置范围)
2. 设置 uart1 的中断优先级
3. 使能 NVIC 响应 uart1 的中断请求(因为 NVIC 管理单片机的所有外设，所以要先确定好 NVIC 响应谁)
4. 使能模块的接收中断(USART1->CR1 的第五位为 1)(一个中断源里面有很多中断)

4.2.4 NVIC 相关配置函数

ARM 公司提供了一套专门用于设置 NVIC 的通用函数。
操作库函数 → 操作寄存器
在这函数存在于“core_m4.h”里面

core_cm4.h

```
1493 /** \defgroup CMSIS_Core_FunctionInterface Functions and Instructions Reference
1494  */
1495
1496                                    NVIC相关功能函数
1497
1498 /* ##########################   NVIC functions  ##################################### */
1499 /** \ingroup  CMSIS_Core_FunctionInterface
1500     \defgroup CMSIS_Core_NVICFunctions NVIC Functions
1501     \brief      Functions that manage interrupts and exceptions via the NVIC.
1502     @{
1503  */
1504
1505 /** \brief  Set Priority Grouping
1506
1507   The function sets the priority grouping field using the required unlock sequence.
1508   The parameter PriorityGroup is assigned to the field SCB->AIRCR [10:8] PRIGROUP field.
1509   Only values from 0..7 are used.
1510   In case of a conflict between priority grouping and available
1511   priority bits (__NVIC_PRIO_BITS), the smallest possible priority group is set.
1512
1513     \param [in]      PriorityGroup  Priority grouping field.
```

4.2.4.1 设置优先级分组函数

函数原型：void NVIC_SetPriorityGrouping(uint32_t PriorityGroup)
函数功能：设置优先级分组
参数说明：PriorityGroup-写入 AIRCR 寄存器 PRIGROUP[10:8]位的值(3/4/5/6/7)---7-抢占优先级位数
NVIC_SetPriorityGrouping(7-3)；抢占优先级位数：就可以推断出响应优先级位数
NVIC_SetPriorityGrouping(4)；
效果是一样的

举例：NVIC_SetPriorityGrouping(7-2)。
NVIC_SetPriorityGrouping(5)。

4.2.4.2 设置中断优先级函数

函数原型;void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
函数功能：设置某个中断的抢占优先级优先级和响应优先级
参数说明：IRQn_Type-枚举类型
IRQn：中断源标号 USART1_IRQn 或者 37
priority：优先级分组值和抢占优先级级别值和响应优先级级别值的合成值

4.2.4.3 优先级合成函数

函数原型：uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)
函数功能：将优先级分组值和抢占优先级级别值和响应优先级级别值的合成一个 32 位值
参数说明：PriorityGroup 优先级分组值
PreemptPriority 抢占优先级级别值
SubPriority 响应优先级级别值
返回值：优先级分组值和抢占优先级级别值和响应优先级级别值的合成值

举例：以 usart1 的接收中断为例

1. 设置优先级分组
   NVIC_SetPriorityGrouping(7-2);//抢占优先级位数 2(0 - 3)， 响应优先级位数 2(0 - 3).
2. 设置 usart1 的中断优先级
   方法一：
   uint32_t priority;
   priority = NVIC_EncodePriority (7-2, 1, 1)； //得到一个合成值
   NVIC_SetPriority(USART1_IRQn, priority);//设置好 uart1 的中断优先级
   方法二：
   NVIC_SetPriority(USART1_IRQn, NVIC_EncodePriority (7-2, 1, 1));
   方法三：
   NVIC_SetPriority(37, NVIC_EncodePriority (7-2, 1, 1));
3. 使能 NVIC 响应
   NVIC_EnableIRQ(USART1_IRQn)；

4.2.4.4 NVIC 响应中断使能

函数原型：void NVIC_EnableIRQ(IRQn_Type IRQn)
函数功能：开启 NVIC 对某个中断的响应
参数说明：IRQn_Type-枚举类型
IRQn：中断源标号
EnableIRQ(USART1_IRQn);响应 uart1 的中断请求
EnableIRQ(37);响应 uart1 的中断请求

4.3 Stm32F4XX 异常向量表

异常向量表就是所有中断的一个列表。

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) 连接到 NMI 向量。	0x0000 0008
	-1	固定	HardFault	所有类型的错误	0x0000 000C
	0	可设置	MemManage	存储器管理	0x0000 0010
	1	可设置	BusFault	预取指失败，存储器访问失败	0x0000 0014
	2	可设置	UsageFault	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 指令调用的系统服务	0x0000 002C
	4	可设置	Debug Monitor	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

这是一条分界线，上面的异常事件是系统级的，下面的模块级的。NVIC 对系统级中断是必须响应的，而下模块级中断默认是不响应的。

0	7	可设置	WWDG	窗口看门狗中断	0x0000 0040
1	8	可设置	PVD	连接到 EXTI 线的可编程电压检测(PVD) 中断	0x0000 0044
2	9	可设置	TAMP_STAMP	连接到 EXTI 线的入侵和时间戳中断	0x0000 0048
3	10	可设置	RTC_WKUP	连接到 EXTI 线的 RTC 唤醒中断	0x0000 004C
4	11	可设置	FLASH	Flash 全局中断	0x0000 0050
5	12	可设置	RCC	RCC 全局中断	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	DMA1_Stream0	DMA1 流 0 全局中断	0x0000 006C
12	19	可设置	DMA1_Stream1	DMA1 流 1 全局中断	0x0000 0070
13	20	可设置	DMA1_Stream2	DMA1 流 2 全局中断	0x0000 0074
14	21	可设置	DMA1_Stream3	DMA1 流 3 全局中断	0x0000 0078
15	22	可设置	DMA1_Stream4	DMA1 流 4 全局中断	0x0000 007C
16	23	可设置	DMA1_Stream5	DMA1 流 5 全局中断	0x0000 0080
17	24	可设置	DMA1_Stream6	DMA1 流 6 全局中断	0x0000 0084
18	25	可设置	ADC	ADC1、ADC2 和 ADC3 全局中断	0x0000 0088

位置	优先级	优先级类型	名称	说明	地址
19	26	可设置	CAN1_TX	CAN1 TX 中断	0x0000 008C
20	27	可设置	CAN1_RX0	CAN1 RX0 中断	0x0000 0090
21	28	可设置	CAN1_RX1	CAN1 RX1 中断	0x0000 0094
22	29	可设置	CAN1_SCE	CAN1 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_TIM9	TIM1 刹车中断和 TIM9 全局中断	0x0000 00A0
25	32	可设置	TIM1_UP_TIM10	TIM1 更新中断和 TIM10 全局中断	0x0000 00A4
26	33	可设置	TIM1_TRG_COM_TIM11	TIM1 触发和换相中断与 TIM11 全局中断	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^2C1 事件中断	0x0000 00BC
32	39	可设置	I2C1_ER	I^2C1 错误中断	0x0000 00C0
33	40	可设置	I2C2_EV	I^2C2 事件中断	0x0000 00C4
34	41	可设置	I2C2_ER	I^2C2 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

4.4 串口接收中断实验

uart1 接收中断实验

1. 保证 uart1 正常工作(四要素)—波特率 数据位 停止位 校验位
2. 设置优先级分组(设置优先级分配方式)一个工程里面只有一种的
3. 设置 usart1 的中断优先级
4. 使能 NVIC 响应 usart1 中断(核心中断使能)
5. 使能接收中断(模块中断使能)
6. 编写中断服务函数

编写 usart1 中断服务函数。”LED_ON#”

当 PC 发送数据给单片机，接收缓冲区不为空，对应的标志位(中断请求)就会置起来，NVIC 接收到中断请求后就会将 CPU 拉倒中断服务函数中去执行代码，执行完后回到原来的地方继续往下执行。

NVIC 根据相应的标志位(中断请求)帮你抢 CPU

4.5 外部中断概述

芯片外部发生的异常事件(中断)

GPIO 发生电平变化产生的中断(外部中断)

外部中断控制器 → 检测芯片 GPIO 的电平变化(不需要开时钟)

外部中断/事件控制器包含多达 23 个用于产生事件/中断请求的边沿检测器（23 跟外部中断线）。每根输入线都可单独进行配置(每根线都是独立)，以选择类型（中断或事件）和相应的触发事件（上升沿触发、下降沿触发或边沿触发）。每根输入线还可单独屏蔽。挂起寄存器用于保持中断请求的状态线(状态寄存器)。

多达 140 个 GPIO（STM32F405xx/07xx 和 STM32F415xx/17xx）通过以下方式连接到 16 个

第几号管脚的 GPIO 口，那么就只能映射到外部中断线几

PA0 → exti0

PB2 → exti2

PC4 → exti4

PB4 → exti4

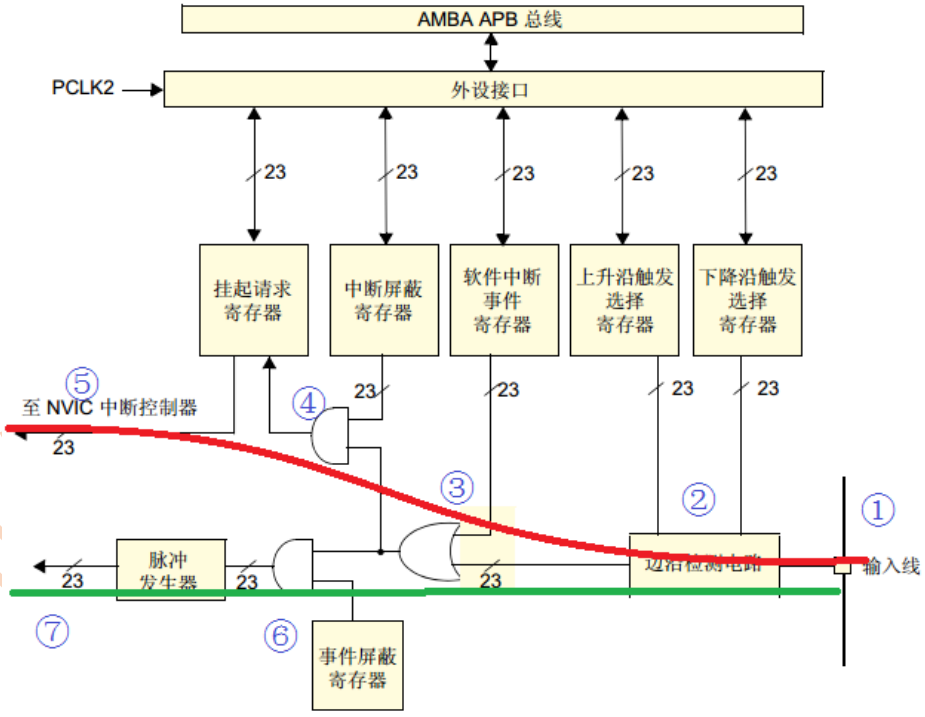
外部中断/事件线：

另外七根 EXTI 线连接方式如下：

- 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 OTG FS 唤醒事件
- EXTI 线 19 连接到以太网唤醒事件
- EXTI 线 20 连接到 USB OTG HS（在 FS 中配置）唤醒事件
- EXTI 线 21 连接到 RTC 入侵和时间戳事件
- EXTI 线 22 连接到 RTC 唤醒事件

4.6 外部中断/事件控制器框架

EXTI2



两个线：红色和绿色

红色的表示产生中断

绿色的表示产生事件

分析红色：

1. 输入线：一共 23 根， GPIO 占 16 根，另外七根是片上外设
2. 边沿检测器：对应着有两个寄存器分为上升沿触发选择寄存器和下降沿触发选择寄存器。如果输入线产生的边沿和上升沿触发寄存器或者下降沿触发寄存器匹配，就会输出一个有效信号”1”;
3. 或门：信号输入端有两个：边沿检测器输出端和软件中断事件寄存器，因为我们要做外部电平变化，所以要将软件中断事件寄存器输出”0”,此时或门输出端才受到边沿检测电路的输出端控制。

4. 与门：信号来源有两个：或门的输出端和中断屏蔽寄存器，如果想要与门输出端输出有效信号，必须中断屏蔽寄存器输出”1”,此时在产生一个有效电平触发。
5. 挂起寄存器：标志位

分析绿色部分

前面 1-3 都是与中断一样

6. 与门：信号来源有两个：或门的输出端和事件屏蔽寄存器，如果想要与门输出端输出有效信号，事件屏蔽寄存器输出”1”,此时产生一个有效电平触发。
7. 脉冲信号发生器：与门输出端输出有效信号“1”,就能让脉冲信号发生器产生一个脉冲去触发硬件工作。

中断：产生异常事件，CPU 就回去执行中断服务函数

事件：产生异常事件，不需要 CPU 参与，只能去触发其它硬件工作

4.7 外部中断相关寄存器

4.7.1 SYSCFG 外部中断配置寄存器 1 (SYSCFG_EXTICR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
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]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

4.7.2 SYSCFG 外部中断配置寄存器 2 (SYSCFG_EXTICR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

4.7.3 SYSCFG 外部中断配置寄存器 3 (SYSCFG_EXTICR3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

4.7.4 SYSCFG 外部中断配置寄存器 4 (SYSCFG_EXTICR4)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 15:0 EXTIx[3:0]： EXTI x 配置（x = 12 到 15） (EXTI x configuration (x = 12 to 15))

这些位通过软件写入，以选择 EXTIx 外部中断的源输入。

0000： PA[x] 引脚

0001： PB[x] 引脚

0010： PC[x] 引脚

0011： PD[x] 引脚
0100： PE[x] 引脚
0101： PF[x] 引脚
0110： PG[x] 引脚
0111： PH[x] 引脚
注意： PI[15:12] 未使用。

举例：需要将 PB7 作为外部中断的信号源。
将 SYSCFG_EXTICR2 的 EXTI7 位写入 0001

4.7.5 中断屏蔽寄存器 (EXTI_IMR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **MRx**：x 线上的中断屏蔽 (Interrupt mask on line x)
0：屏蔽来自 x 线的中断请求
1：开放来自 x 线的中断请求

4.7.6 事件屏蔽寄存器 (EXTI_EMR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **MRx**：x 线上的事件屏蔽 (Event mask on line x)
0：屏蔽来自 x 线的事件请求
1：开放来自 x 线的事件请求

4.7.7 上升沿触发选择寄存器 (EXTI_RTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **TRx**：线 x 的上升沿触发事件配置位 (Rising trigger event configuration bit of line x)

0：禁止输入线上升沿触发（事件和中断）

1：允许输入线上升沿触发（事件和中断）

4.7.8 下降沿触发选择寄存器 (EXTI_FTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **TRx**：线 x 的下降沿触发事件配置位 (Falling trigger event configuration bit of line x)

0：禁止输入线下降沿触发（事件和中断）

1：允许输入线下降沿触发（事件和中断）

4.7.9 软件中断事件寄存器 (EXTI_SWIER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **SWIERx**：线 x 上的软件中断 (Software Interrupt on line x)

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，则产生中断请求。

通过清除 EXTI_PR 的对应位（写入“1”），可以清除该位为“0”。

4.7.10 挂起寄存器 (EXTI_PR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	PR20	PR19	PR18	PR17	PR16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

位 31:23 保留，必须保持复位值。

位 22:0 **PRx**：挂起位 (Pending bit)

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”。在此位中写入“1”可以清除它，也可以通过改变边沿检测的极性清除。

4.8 外部中断实验

以 PA0 为例 EXTI0 (外部中断不需要开时钟(特殊))

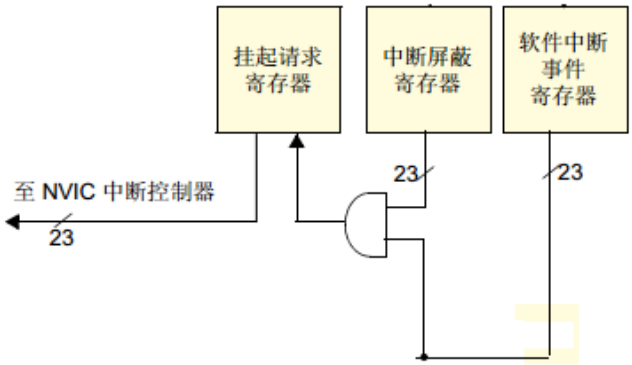
1. 将 PA0 映射到 EXTI0 上
2. 设置 EXTI0 的检测边沿(上升沿)
3. 软件中断事件寄存器输出 0
4. 事件屏蔽寄存器输出 0
5. 中断屏蔽寄存器输出 1
6. 设置 EXTI0 的中断优先级
7. 使能 NVIC 响应 EXTI0 的中断请求
8. 编写中断服务函数

4.9 软件中断作用

可以把比较重要或者紧急的代码放在软件中断服务函数里面，不需要等待外部硬件是否出现异常事件，直接通过软件产生中断。

保护比较重要的代码不被打断

4.10 软件中断框架（重点）



4.11 软件中断相关寄存器

4.11.1 中断屏蔽寄存器 (EXTI_IMR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **MRx**：x 线上的中断屏蔽 (Interrupt mask on line x)
0：屏蔽来自 x 线的中断请求
1：开放来自 x 线的中断请求

4.11.2 软件中断事件寄存器 (EXTI_SWIER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **SWIERx**：线 x 上的软件中断 (Software Interrupt on line x)
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，则产生中断请求。
通过清除 EXTI_PR 的对应位（写入“1”），可以清除该位为“0”。

4.12 软件中断实验

配置流程：(使用任意一条外部中断线都 OK)

作业：PC 发送 OPEN，触发产生软件中断(服务函数里面输出“hello world”);

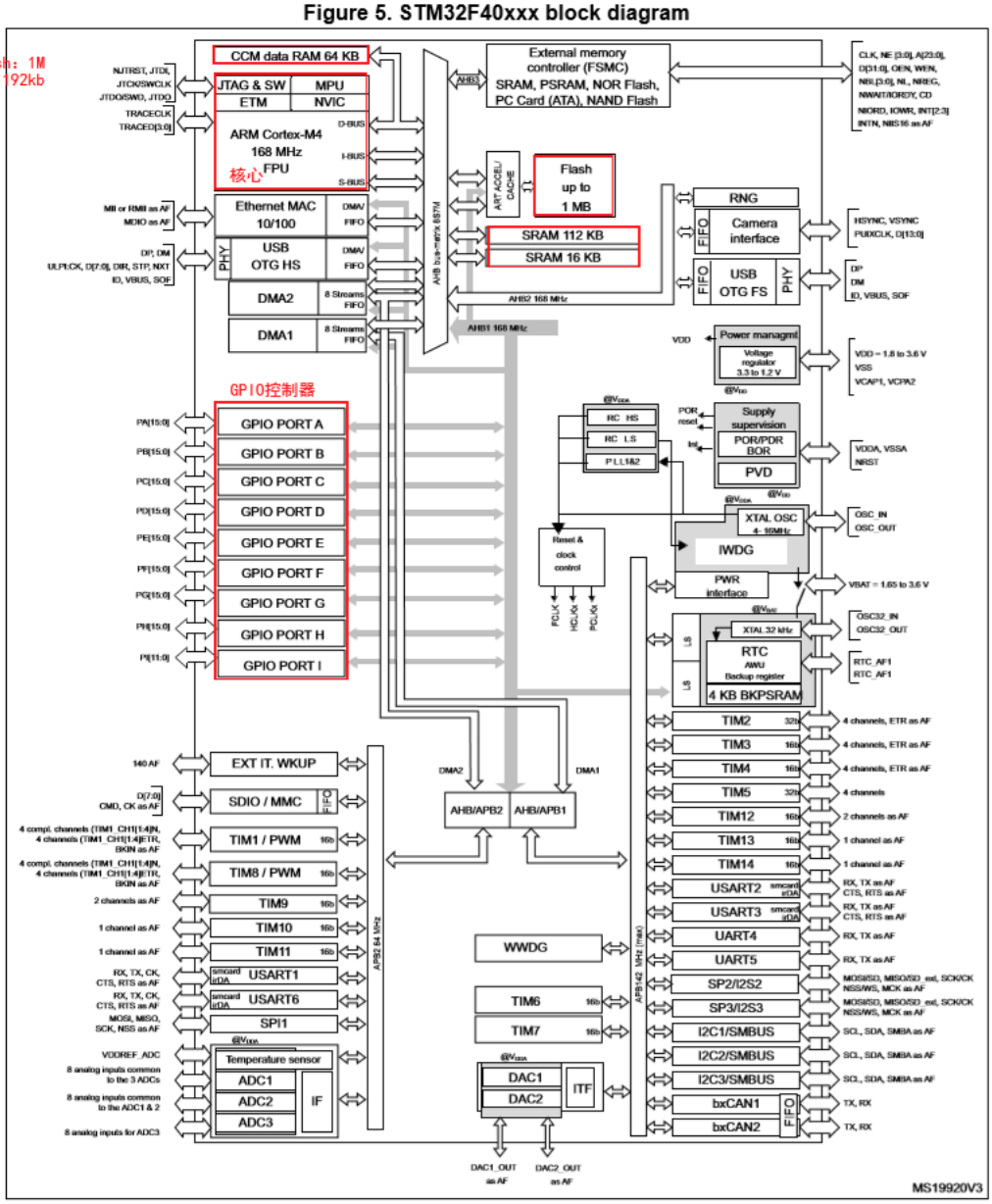
4.13 中断服务函数书写注意事项

中断服务函数名是固定的，在启动代码里面已经定下来了。

书写中断服务函数的时候注意的问题：

1. 中断服务函数名尽量用复制，不要自己写，因为只要你写错一个字母，这个函数就变成普通函数了。
2. （如果中断服务函数是公共入口）进入到中断服务函数后先要查询是哪种中断
3. 先清中断标志，然后再做中断处理，不要把清中断标志放在函数的最后。（如果把清除中断标志放在中断服务函数的最后，会出现当发出清中断标志指令后，硬件还没有把相关标志清除掉，程序就已经跳出了中断服务函数，这个时候 NVIC 又会识别到标志是 1，出现重复中断）。---可以清除中断标志命令发出后，等待清除成功再往下执行。
4. 中断服务函数应该尽量简短，一般是做一些标识，不要在中断中做延时之类的占用 CPU 很长时间的工作。----快进快出
5. 中断服务函数不会被任何一个函数调用，当中断条件满足后，NVIC 控制把 CPU 拉到中断服务函数中执行。

Figure 5. STM32F40xxx block diagram



1. The camera interface and ethernet are available only on STM32F407xx devices.

1、 关于 GPIO 的初始化

2、 关于 USART 的初始化

```
//初始化 UART1
RCC->APB2ENR |= (0x1 << 4);//打开 USART1 的时钟
USART1->CR1 = 0;//整体清零

USART1->CR1 |= (0x1 << 2);//接收器使能

USART1->CR1 |= (0x1 << 3);//发送器使能
```

位 2 **RE**：接收器使能 (Receiver enable)

该位使能接收器。该位由软件置 1 和清零。

0：禁止接收器

1：使能接收器并开始搜索起始位

位 3 **TE**：发送器使能 (Transmitter enable)

该位使能发送器。该位由软件置 1 和清零。

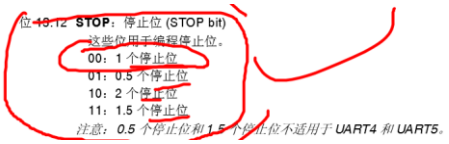
0：禁止发送器

1：使能发送器

注意：1：除了在智能卡模式下以外，传送期间 TE 位上的“0”脉冲（“0”后紧跟的是“1”）会在当前字的后面发送一个报头（空闲线路）。

2：当 TE 置 1 时，在发送下始前存在 1 位的时间延迟。

USART1->CR2 &= ~(0x3 << 12);//1 个停止位



//波特率计算，对应数据手册

USARTDIV = (float)84000000/(bound*16); 波特率表示 1S 钟发送的位数：9600 个位

频率是 1M----对应周期=1/1000 000 S=1us 频率 84M----对应周期=1/84000 000S 1S

MAN_DIV = USARTDIV;//取整数

FRA_DIV = (USARTDIV - MAN_DIV) * 16 + (float)0.5;//四舍五入

USART1->BRR = MAN_DIV << 4 | FRA_DIV; 整数部分为：1011 小数部分 1110

整数；如何变化 1011 << 4 ------1011 0000 | 小数部分：1110 等于 1011 1110 整数和小数组合完成

位 15:4 **DIV_Mantissa[11:0]**：USARTDIV 的尾数
这 12 个位用于定义 USART 除数 (USARTDIV) 的尾数

位 3:0 **DIV_Fraction[3:0]**：USARTDIV 的小数
这 4 个位用于定义 USART 除数 (USARTDIV) 的小数。当 OVER8 = 1 时，不考虑 DIV_Fraction3 位，且必须将该位保持清零。

//中断 内核提供直接进行中断优先级分组

NVIC_SetPriority(USART1_IRQn, NVIC_EncodePriority (7-2, 1, 1));//设置 uart1 的中断优先级

NVIC_EnableIRQ(USART1_IRQn);//使能 NVIC 响应 uart1 的中断请求

USART1->CR1 |= (0x1 << 5);//使能接收中断

//使能 UART1 一般来说模块使能都是放在最后

USART1->CR1 |= (0x1 << 13);

位 5 **RXNEIE**：RXNE 中断使能 (RXNE interrupt enable)
此位由软件置 1 和清零。
0：禁止中断
1：当 USART_SR 寄存器中 ORE=1 或 RXNE=1 时，生成 USART 中断

定义变量：u8 rev_str[50]; //不需要进行初始化 这个写应用程序的时候进行判断和调用

u8 rev_ok; //字符串接收完成标志

void USART1_IRQHandler() ---→ 中断向量表里面必须一模一样

{

u8 dat = 0;

static u8 cnt = 0;

if(USART1->SR & (0x1 << 5))//接收中断标志

位 5 **RXNE**：读取数据寄存器不为空 (Read data register not empty)
当 RDR 移位寄存器的内容已传输到 USART_DR 寄存器时，该位由硬件置 1。如果 USART_CR1 寄存器中 RXNEIE = 1，则会生成中断。通过对 USART_DR 寄存器执行读入操作将该位清零。RXNE 标志也可以通过向该位写入零来清零。建议仅在多缓冲区通信时使用此清零序列。
0：未接收到数据
1：已准备好读取接收到的数据

```
{
 USART1->SR &= ~(0x1 << 5);//清接收中断标志
 USART1->SR = USART1->SR & (~(0x1 << 5));   //清接收中断标志
 while( USART1->SR & (0x1 << 5) );//等待接收中断标志位为 0     while(1)—死等    while(0)—跳出
                  dat = USART1->DR;   //DR 寄存器的数值保存到 dat 里面
```

26.6.2 数据寄存器 (USART_DR)

Data register

偏移地址：0x04

复位值：0xXXXX XXXX

位 31:9 保留，必须保持复位值

位 8:0 **DR[8:0]**：数据值

包含接收到数据字符或已发送的数据字符，具体取决于所执行的操作是“读取”操作还是“写入”操作。

因为数据寄存器包含两个寄存器，一个用于发送 (TDR)，一个用于接收 (RDR)，因此它具有双重功能（读和写）。

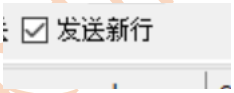
TDR 寄存器在内部总线和输出移位寄存器之间提供了并行接口（参见图 1）。

RDR 寄存器在输入移位寄存器和内部总线之间提供了并行接口。

在使能奇偶校验位的情况下（USART_CR1 寄存器中的 PCE 位被置 1）进行发送时，由于 MSB 的写入值（位 7 或位 8，具体取决于数据长度）会被奇偶校验位所取代，因此该值不起任何作用。

在使能奇偶校验位的情况下进行接收时，从 MSB 位中读取的值为接收到的奇偶校验位。

```
 if(dat == '#')//说明字符串接收完成
  {
       rev_str[cnt] = '\0';//字符串都是以\0 结尾的
       cnt = 0;//下一次字符也是从 0 号下标开始存
       rev_ok = 1;//表示接收完成
  }
  else if(dat == '\n') //表示有新行加入
  {
       rev_str[cnt-1] = '\0';//字符串都是以\0 结尾的
       cnt = 0;//下一次字符也是从 0 号下标开始存
       rev_ok = 1;//表示接收完成
  }
  else
  {
       rev_str[cnt] = dat;//0   1   2   3
       cnt++;
  }
```




```
 }
}

Uart1_Send_String(rev_str);   //把接收到的字符串重新发送出去在串口助手上显示
void Uart1_Send_String(u8 *p)//hello
{
    while(*p != '\0')
    {
        while( (USART1->SR & (0x1 << 7)) == 0 );//等待发送缓冲为空
```

注意：如果 LBDIE=1，则当 LBD=1 时生成中断

位 7 **TXE**：发送数据寄存器为空 (Transmit data register empty)

当 TDR 寄存器的内容已传输到移位寄存器时，该位由硬件置 1。如果 USART_CR1 寄存器中 TXEIE 位 = 1，则会生成中断。通过对 USART_DR 寄存器执行写入操作将该位清零。

0：数据未传输到移位寄存器

1：数据传输到移位寄存器

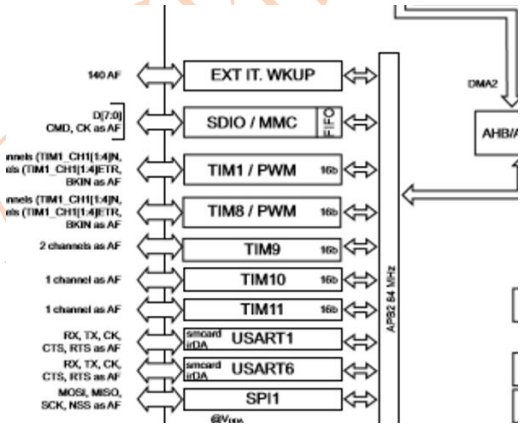
注意：单缓冲区发送期间使用该位。

```
        //如果位 7 为 0，核心卡在这个 while 里面
        //如果位 7 为 1，核心就出来
        USART1->DR = *p;//把 dat 里面的值发送出去   清除标志位
        p++;//指向下一个字符
    }
}

//字符比较函数 strcmp
if( strcmp("LED_ON", (char *)rev_str) == 0 )//字符串一样
            {
                LED1_ON;    //点亮所有 LED 灯
                LED2_ON;
                LED3_ON;
                LED4_ON;
            }else if( strcmp("LED_OFF", (char *)rev_str) == 0 )
            {
                LED1_OFF;
                LED2_OFF;
                LED3_OFF;
                LED4_OFF;
            }
```

3、 外部中断

```
//将 PA0 映射外部中断线 0
RCC->APB2ENR |= (0x1 << 14);//开启 SYSCFG 时钟
```




```
SYSCFG->EXTICR[0] &= ~(0xf << 0);//PA0 --> EXTI0
```

复位值：0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
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]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:16 保留，必须保持复位值。

位 15:0 **EXTI[3:0]**：EXTI x 配置（x = 0 到 3）（EXTI x configuration (x = 0 to 3)）
这些位通过软件写入，以选择 EXTIx 外部中断的源输入。
0000：PA[x] 引脚
0001：PB[x] 引脚
0010：PC[x] 引脚
0011：PD[x] 引脚
0100：PE[x] 引脚
0101：PF[C] 引脚
0110：PG[x] 引脚
0111：PH[x] 引脚
1000：PI[x] 引脚

//配置上升沿触发
EXTI->RTSR |= (0x1 << 0);//上升沿
EXTI->FTSR &= ~(0x1 << 0);//禁止下降沿触发

10.3.3 上升沿触发选择寄存器 (EXTI_RTSR)

Rising trigger selection register

偏移地址：0x08
复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **TRx**：线 x 的上升沿触发事件配置位 (Rising trigger event configuration bit of line x)
0：禁止输入线上升沿触发（事件和中断）
1：允许输入线上升沿触发（事件和中断）

10.3.4 下降沿触发选择寄存器 (EXTI_FTSR)

Falling trigger selection register

偏移地址：0x0C
复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **TRx**：线 x 的下降沿触发事件配置位 (Falling trigger event configuration bit of line x)
0：禁止输入线下降沿触发（事件和中断）
1：允许输入线下降沿触发（事件和中断）

//软件中断事件寄存器输出 0
EXTI->SWIER &= ~(0x1 << 0);

10.3.5 软件中断事件寄存器 (EXTI_SWIER)

Software interrupt event register

偏移地址：0x10
复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **SWIERx**：线 x 上的软件中断 (Software interrupt on line x)
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，则产生中断请求。
通过清除 EXTI_PR 的对应位（写入“1”），可以清除该位为“0”。

//中断屏蔽寄存器(中断使能位)
EXTI->IMR |= (0x1 << 0);

10.3.1 中断屏蔽寄存器 (EXTI_IMR)

Interrupt mask register

偏移地址：0x00

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **MRx**：x 线上的中断屏蔽 (Interrupt mask on line x)

0：屏蔽来自 x 线的中断请求

1：开放来自 x 线的中断请求

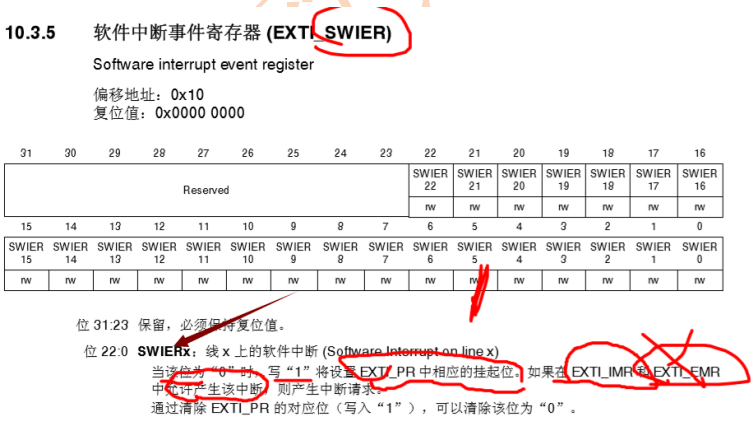
```
//设置 EXTI0 的中断优先级   --调用内核里面的函数
 NVIC_SetPriority(EXTI0_IRQn, NVIC_EncodePriority (7-2, 1, 3));

 //使能 NVIC 响应 EXTI0 的中断请求
 NVIC_EnableIRQ( EXTI0_IRQn );

//编写中断服务函数
void EXTI0_IRQHandler(void)
{
 EXTI->PR |= (0x1 << 0);//清 EXTI0 标志
 while( EXTI->PR & (0x1 << 0) );//等待 EXTI0 标志位清 0
 LED1_ON;
 printf("hello world\r\n");
}
```

4、 软件中断 --- 外部中断线 9

```
//软件中断事件寄存器输出 0(刚开始不进中断)
 EXTI->SWIER &= ~(0x1 << 9);
```



10.3.5 软件中断事件寄存器 (EXTI_SWIER)

Software interrupt event register

偏移地址：0x10

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **SWIERx**：线 x 上的软件中断 (Software Interrupt on line x)

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，则产生中断请求。

通过清除 EXTI_PR 的对应位（写入“1”），可以清除该位为“0”。

```
//中断屏蔽寄存器输出 1
 EXTI->IMR |= (0x1 << 9);
```

10.3.1 中断屏蔽寄存器 (EXTI_IMR)

Interrupt mask register

偏移地址：0x00

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:23 保留，必须保持复位值。

位 22:0 **MRx**：x 线上的中断屏蔽 (Interrupt mask on line x)

0：屏蔽来自 x 线的中断请求

1：开放来自 x 线的中断请求

```
//设置 EXTI9 的中断优先级 – 调用系统内部中断函数
 NVIC_SetPriority(EXTI9_5_IRQn, NVIC_EncodePriority (7-2, 0, 0));

 //使能 NVIC 响应 EXTI9 的中断请求
 NVIC_EnableIRQ( EXTI9_5_IRQn );

//软件中断服务函数
//外部中断线 9 的中断服务函数
void EXTI9_5_IRQHandler(void)
{
 if( EXTI->PR & (0x1 << 9) )//说明外部中断线 9

 {
      EXTI->PR |= (0x1 << 9);//清外部中断线 9 的标志位
      while( EXTI->PR & (0x1 << 9) );//等待外部中断线 9 清除完毕
```

10.3.6 挂起寄存器 (EXTI_PR)

Pending register

偏移地址：0x14

复位值：未定义

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	PR20	PR19	PR18	PR17	PR16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

位 31:23 保留，必须保持复位值。

位 22:0

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”。在此位中写入“1”可以清除它，也可以通过改变边沿检测的极性清除。

```
      LED1_ON;
      LED2_ON;
 }
}
```

主函数：

```
while(1)//死循环  让核心有地方去  防止核心跑飞
 {
      if( rev_ok == 1 )
      {
           rev_ok = 0;

           if(strcmp((char *)rev_str, "OPEN") == 0)
```

```
            {
EXTI->SWIER |= (0x1 << 9);//触发产生软件中断----------》给软件中断事件寄存器第九位赋值 1，触发中断进入中断服务函数
            }
            else
            {
               EXTI->SWIER |= (0x1 << 3);//触发产生软件中断
            }
            if(strcmp((char *)rev_str, "LED") == 0)
            {
                EXTI->SWIER |= (0x1 << 3);//触发产生软件中断
            }
        }
    }
```