第 5 章 中断控制器 NVIC 与 EXTI

CHAPTER 5

5.1 什么是中断

在嵌入式系统中，中断是一种机制，用于在系统执行正常程序流程的同时，及时响应特定事件或条件的发生。当某个特定的事件发生时（如外部硬件触发、定时器到达指定时间），中断会打断当前正在执行的程序流程，转而执行一个预先定义好的中断服务程序（Interrupt Service Routine，ISR），来处理该事件。而中断嵌套是指中断系统正在执行一个中断服务时，有另一个优先级更高的中断提出中断请求，这时会暂时终止当前正在执行的级别较低的中断服务程序，去执行级别更高的中断服务程序，待处理完毕，再返回被中断了的中断服务程序继续执行过程。中断的处理过程如图 5.1 所示。

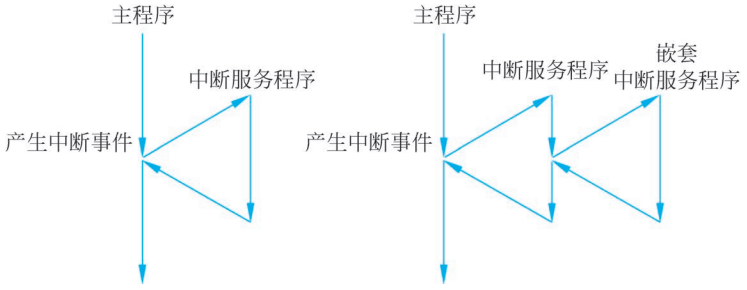

图 5.1 中断的处理过程示意图

中断的出现主要是为了提高系统的响应速度和效率。相比于轮询（Polling）方式，中断可以让系统在等待事件发生时不会浪费处理器资源，而是通过硬件或软件的方式，让事件一旦发生就立即得到响应。这样可以让系统在处理多个任务时更加灵活高效。

中断处理过程一般包括以下几个步骤。

（1）中断请求触发：某个事件或条件发生，触发了相应的中断请求。

（2）中断控制器响应：中断控制器检测到中断请求，根据优先级确定下一个要执行的中断服务程序。

（3）保存上下文：在转到中断服务程序之前，系统需要保存当前程序的执行状态（如寄存器值、程序计数器等），以便在中断服务程序执行完毕后能够恢复到中断之前的状态。

（4）执行中断服务程序：执行与中断相关的中断服务程序，处理中断事件。

（5）恢复现场：中断服务程序执行完毕后，系统需要恢复之前保存的程序执行状态，以

便回到中断发生之前的程序执行流程。

(6) 继续执行：恢复程序执行流程，继续执行中断发生时被打断的程序。

通过合理地使用中断机制，可以实现系统的多任务处理、实时响应外部事件等功能，提高系统的性能和可靠性。

在 STM32 中还需要区分异常这个概念。中断和异常都是用于处理系统中的特殊事件，但它们的触发原因和处理方式有所不同。中断是由 CPU 外部事件触发的，如外部设备的状态改变、定时器溢出、外部 I/O 引脚的状态变化等，系统可以设计中断服务程序来处理这些事件，并且可以被优先级化和屏蔽；而异常是由 CPU 内部的条件触发的，如除零、非法指令、访问未定义地址等，通常表示系统出现了错误或者特殊情况。


视频讲解

5.2 嵌套向量中断控制器 NVIC

5.2.1 NVIC 简介

STM32 系列单片机支持众多的系统异常和外部中断。如 STM32F10xxx 系列有多达 10 个系统内部异常和 60 个外部中断，STM32F405xx 系列有多达 10 个系统内部异常和 82 个外部中断，STM32U5 系列有多达 12 个系统内部异常和 141 个外部中断。中断向量表存放在内存中的特定位置，其中存储着各种中断服务程序的入口地址。当一个中断事件发生时，处理器会根据中断向量表中对应中断号的地址跳转到相应的中断服务程序，以执行相应的中断处理操作。每个中断号对应着一个特定的中断事件，比如外部中断、定时器中断等，也对应着一个中断服务程序入口地址。通过中断向量表，嵌入式系统可以实现有效的中断处理，提高系统的响应速度和可靠性。表 5.1 为 STM32U5 系列的部分中断向量表。

表 5.1 STM32U5 系列的部分中断向量表

自然优先级	优先级类型	名　　称	说　　明	地　　址
—	—	—	保留	0x0000 0000
−4	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
−3 或−1	固定	Secure HardFault	安全硬件错误	0x0000 000C
−1	固定	Nonsecure HardFault	非安全硬件故障，所有类别的故障	0x0000 000C
0	可设置	MemManage	内存管理	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
3	可设置	SecureFault	安全故障	0x0000 001C
—	—	—	保留	0x0000 0020 ～ 0x0000 002B
4	可设置	SVC	通过 SWI 指令调用的系统服务	0x0000 002C
5	可设置	Debug Monitor	调试监控器	0x0000 0030
—	—	—	保留	0x0000 0034
6	可设置	PendSV	可挂起的系统服务	0x0000 0038

续表

自然优先级	优先级类型	名　　称	说　　明	地　　址
7	可设置	SysTick	系统嘀嗒定时器	0x0000 003C
8	可设置	WWDG	窗口看门狗中断	0x0000 0040
9	可设置	PVD_PVM	可编程电压检测(PVD) /外设电压检测	0x0000 0044
10	可设置	RTC	RTC 全局非安全中断	0x0000 0048
…				
146	可设置	DCACHE2	DCACHE 2 全局中断	0x0000 0268
147	可设置	GFXTIM	GFXTIM 全局中断	0x0000 026C
148	可设置	JPEG	JPEG 同步中断	0x0000 0270

为了方便地管理这些中断和内部异常，STM32 使用了内嵌向量中断控制器(Nested Vectored Interrupt Controller，NVIC)。以下是 STM32 NVIC 的一些重要特点和功能。

(1) 中断向量表：NVIC 使用中断向量表来确定中断服务程序的位置。在 STM32 中，中断向量表通常位于内部 Flash 的起始地址处，包含每个中断的地址。

(2) 中断优先级：NVIC 允许每个中断有抢占优先级和子优先级。较低抢占优先级的中断可以被较高抢占优先级的中断打断，这样可以确保重要的中断能够及时得到处理。当多个中断或任务具有相同的抢占优先级时，子优先级决定了哪一个会首先得到执行。优先级的序号越小，优先级越高。

(3) 中断控制：NVIC 可以用于使能或禁用特定中断。

(4) 异常处理：除普通的外部中断请求外，NVIC 还负责处理内部异常，如硬件错误、系统异常等。

(5) 中断挂起：NVIC 允许中断被暂时挂起，直到某些条件满足后重新使能。

通过合理配置 NVIC，STM32 可以有效地管理各种中断，并确保系统的可靠性和响应性。

5.2.2　NVIC 的优先级

为了设置每个中断的优先级，便于高优先级的中断优先得到响应，STM32 具有灵活的优先级管理方式。

首先，每个中断可被设置为不同的抢占优先级和子优先级等级。抢占优先级高的中断可以打断抢占优先级低的中断从而优先执行。当抢占优先级相同时，子优先级较高的中断会优先执行，但不会打断子优先级较低的中断。若两个中断的抢占优先级和子优先级相同，则按照自然优先级排序。需要注意的是，优先级的序号越小，优先级越高。

在 STM32 中，还有优先级分组的概念，一般可选择 5 个组中的一个组。设置为组 0 时，无法设置中断的抢占优先级的等级，可以用 4 个位设置子优先级的等级。设置为组 1 时，可以用 1 个位设置中断的抢占优先级的等级，可以用 3 个位设置子优先级的等级。具体采用哪个组的方案需要通过设置应用中断和复位控制寄存器(Application Interrupt and Reset Control Register，AIRCR)的第[10：8]位指定，如表 5.2 所示。

表 5.2 NVIC 的优先级分组

优先级分组	AIRCR[10：8]	优先级划分
0	111	0 个位用于抢占优先级，4 个位用于子优先级
1	110	1 个位用于抢占优先级，3 个位用于子优先级
2	101	2 个位用于抢占优先级，2 个位用于子优先级
3	100	3 个位用于抢占优先级，1 个位用于子优先级
4	011	4 个位用于抢占优先级，0 个位用于子优先级

举例来说，如果在系统初始化时选择使用组 2 的划分方式，那么对于某个中断，我们可以将其抢占优先级设置为 0～3 中的一个数，将其子优先级设置为 0～3 中的一个数。如果选择使用组 4 的划分方式，那么对于某个中断，可以将其抢占优先级设置为 0～16 中的一个数，而不可设置其子优先级。序号越小，优先级越高。

5.2.3 NVIC 的 STM32CubeMX 配置

在 NVIC 设置页面中，Priority Group 即为设置优先级分组的位置，其中有 5 个分组，如图 5.2 所示。如果选择 4 bits for pre-emption priority，那么抢占优先级可以设置为 0～15 中的一个，而子优先级不可以设置，因为抢占优先级和子优先级所使用的位数一共是 4 位。

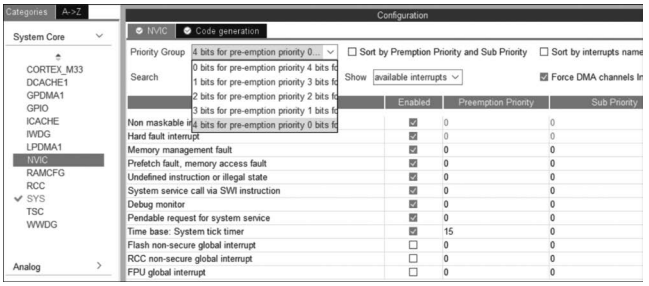

图 5.2 STM32CubeMX 中的 NVIC 优先级分组设置

在 Code generation 页面，如果想调用中断处理函数，则需要选中 Generate IRQ handler 和 Call HAL handler 复选框，这样在生成的代码中才会生成中断处理函数和 HAL 库中的对应外设的中断处理函数，如图 5.3 所示。

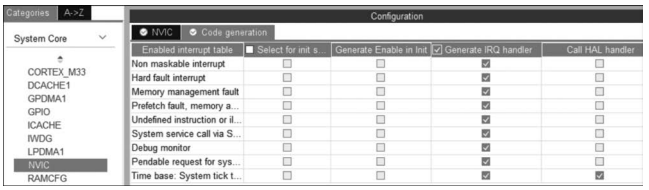

图 5.3 NVIC 的 Code generation 页面

这样，在所生成工程中的 stm32u5xx_it.c 文件中，会有如图 5.4 所示的两个中断处理函数。

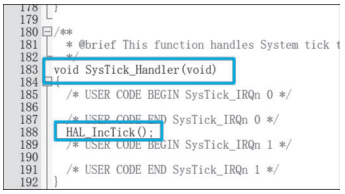

图 5.4 STM32CubeMX 生成的中断处理函数


视频讲解

5.3　EXTI

5.3.1　EXTI 简介

EXTI 的全称是 External Interrupt/Event Controller，即外部中断/事件控制器，它允许外部事件（如 GPIO 引脚状态变化、以太网唤醒、USB 唤醒等）触发中断或事件，并在需要时唤醒系统。EXTI 与其他外设中断的不同是，EXTI 专指来源于芯片外部的信号所触发的中断，而其他外设中断来自芯片内部。

这里还需要区分一下中断和事件的概念。在 STM32 的手册中会经常看到中断（Interrupt）和事件（Event）。中断是事件的一种，可以称为中断事件。中断发生时会立即中断当前的程序执行流程，转而执行与该中断相关联的中断服务程序（Interrupt Service Routine，ISR），处理完中断服务程序后返回源程序继续执行。非中断事件不会立即中断当前 CPU 程序的执行，而是会生成一个事件信号，可以通过软件轮询、触发 DMA 传输、使能其他外设等方式来处理。

5.3.2　EXTI 的内部架构

一个 EXTI 线上可以部署多个外部中断/事件来源。STM32F1、STM32F4 和 STM32U5 系列芯片都有多个 EXTI 线，前 16 个 EXTI 线的功能大致相同，都负责 GPIO 引脚的外部中断，其他的 EXTI 线不尽相同，不同型号 STM32 的 EXTI 线的总数也不太一样，如表 5.3 所示。

表 5.3　不同型号的 STM32 的 EXTI 线的功能说明

EXTI 线	STM32F103	STM32F405	STM32U575
0～15	GPIO	GPIO	GPIO
16	PVD 输出	PVD 输出	PVD 输出
17	RTC 闹钟事件	RTC 闹钟事件	COMP1 输出
18	USB 唤醒事件	USB OTG FS 唤醒事件	COMP2 输出
19	以太网唤醒事件	以太网唤醒事件	V_{DDUSB} 电压监测
20	—	USB OTG HS（在 FS 中配置）唤醒事件	V_{DDIO2} 电压监测
21	—	RTC 入侵和时间戳事件	V_{DDA} 电压监测 1
22	—	RTC 唤醒事件	V_{DDA} 电压监测 2

STM32F103、STM32F405 的 EXTI 内部架构如图 5.5 所示。

STM32U575 的 EXTI 内部架构如图 5.6 所示。

尽管两个结构框图细节不太一样，但是它们包含相似的结构和功能。图 5.6 的可配置事件输入对应图 5.5 的输入线，AHB 接口对应图 5.5 的 APB 总线，hclk 对应图 5.5 的 PCLK2，它们都受 EXTI 内部的寄存器和逻辑单元控制，从而向 CPU 或芯片内部其他单元输出中断或事件信息。

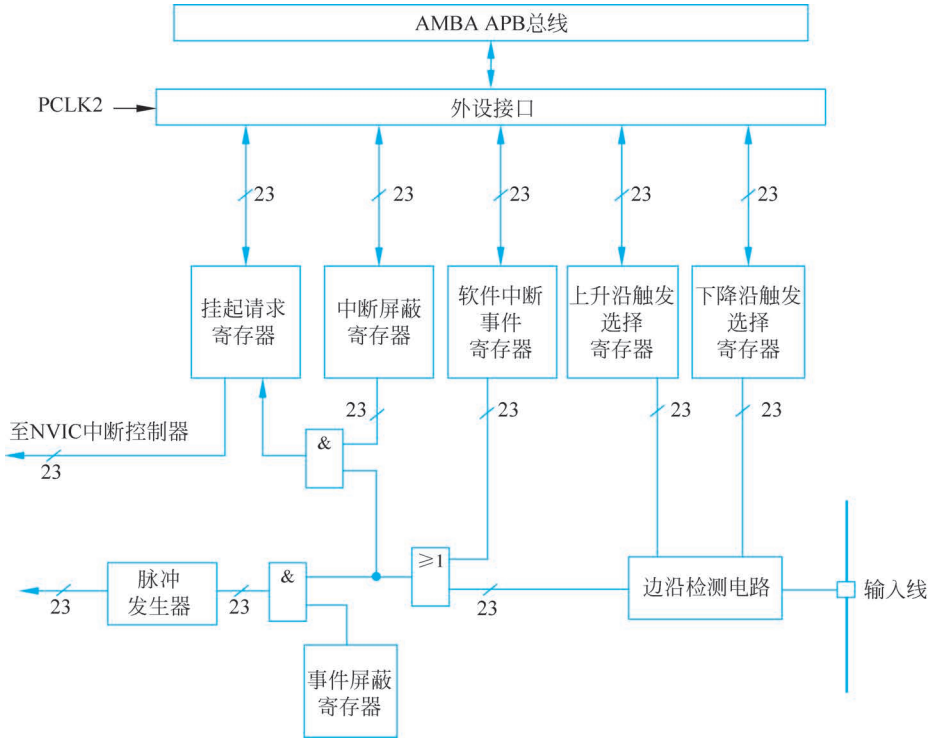

图 5.5　STM32F103、STM32F405 的 EXTI 内部架构图

图 5.5 的输入线连接着边沿检测电路，这对应图 5.6 的异步边沿探测电路。也就是说，需要通过外部信号电平跳变时产生的上升沿或下降沿来判断中断是否产生。

5.3.3　GPIO 的外部中断

由 GPIO 引脚触发的外部中断是经常被用到的，下面着重介绍 GPIO 外部中断的原理。如 5.3.2 节所述，EXTI0～EXTI15 连接的是 GPIO。由于 GPIO 引脚众多，远远超过 16 个，因此每根 EXTI 线需要连接多个 GPIO 引脚。因为一组 GPIO（如 GPIOA、GPIOB）有 16 个引脚，所以可以利用前 16 根 EXTI 线对这些引脚进行分组。分组方式如图 5.7 所示。

通过该结构图可以知道，不管同一组的哪个引脚触发了中断，该组的 EXTI 线上都会产生信号。但是，每个组只能设置一个引脚触发中断。举例来说，如果把 PA0 设置为外部中断模式，该组的其他引脚都不能被设置为外部中断。因此，实际上只能使用 16 个 GPIO 外部中断。每组使用的引脚编号由 EXTICRx（x＝1～4）寄存器设置。每个寄存器分为 4 个区域，每个区域的 8 位用于配置一个组内采用的中断引脚是哪个。

在 STM32F405 中，EXTI5～EXTI9 共用一个中断地址，EXTI10～EXTI15 共用一个中断地址。这意味着 EXTI5～EXTI9 会共用一个中断服务程序 EXTI9_5_IRQHandler()，如图 5.8 所示。

在 STM32U575 中，EXTI0～EXTI15 中每根中断线都有独立的中断地址，可以使用独立的中断服务程序。

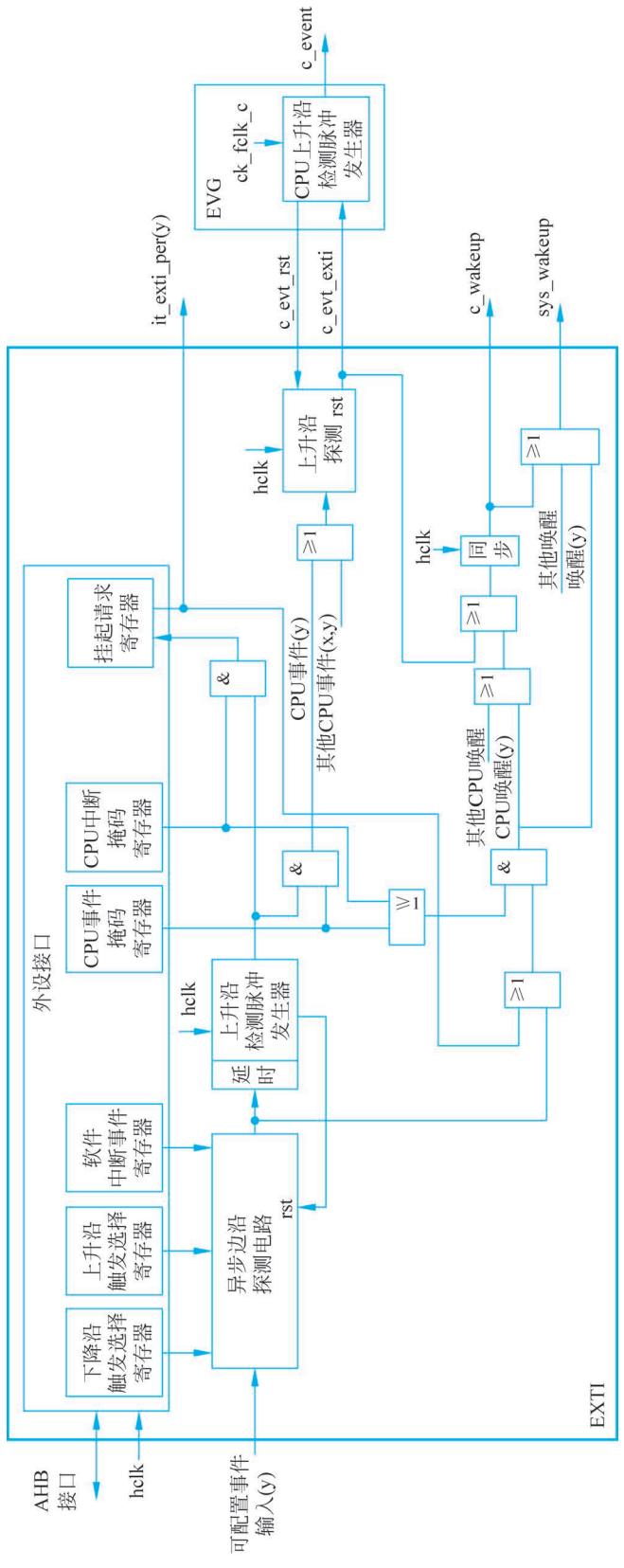

图 5.6　STM32U575 的 EXTI 内部架构

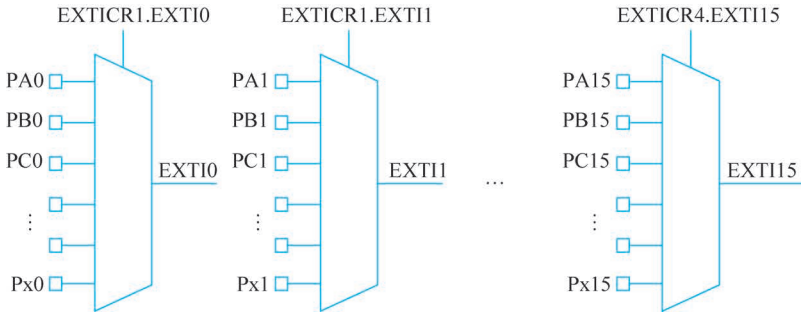

图 5.7　GPIO 与 EXTI 线的分组方式

表 45.　STM32F405xx/07xx 和 STM32F415xx/17xx 的向量表（续）

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

图 5.8　STM32F405 的 EXTI 中断地址分配

5.3.4　EXTI 的 STM32CubeMX 配置

在 STM32CubeMX 中随便找一个 GPIO 引脚设置为 EXTI 模式，即可在 GPIO 界面中看到相应的配置，如图 5.9 所示。

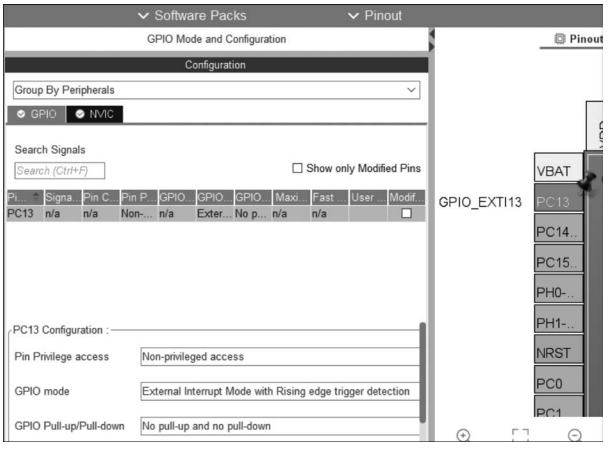

图 5.9　STM32CubeMX 中的 GPIO 的 EXTI 配置页面

Pin Privilege access 可以设置特权访问属性。Privileged-only access（仅特权访问）是一种访问权限，它意味着只有处于特权模式（Privileged Mode）的代码或者任务才能够配置外部中断线。这种权限设置可以确保只有系统的关键部分才能够对外部中断进行控制和操作，从而增强系统的安全性和稳定性。该配置是 STM32U5 中特有的，在 STM32F405、STM32F103 中没有。

GPIO mode 可以设置中断的触发条件，如上升沿触发、下降沿触发、上升沿或下降沿触发；也可以设置为外部中断模式或外部事件模式。在外部事件模式下，引脚上的电平或边

沿的变化并不会直接触发中断，而是通过外部事件触发器（External Event Trigger）来控制。这通常用于同步外部事件和微控制器内部的操作，可以通过配置的方式触发相应的外部事件，如定时器的开始和结束、某些数据的传输等。

GPIO Pull-up/Pull-down 用于配置引脚的上拉电阻或下拉电阻。

设置完 GPIO 的触发模式后，还需要在 NVIC 页面使能该中断，如图 5.10 所示。

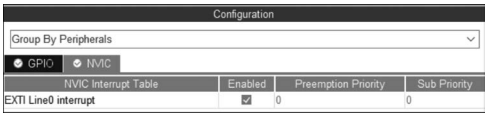

图 5.10　STM32CubeMX 的 GPIO 的 NVIC 使能页面

5.3.5　EXTI 的寄存器

下面介绍几个常用的寄存器。

1. EXTI 上升沿触发选择寄存器（EXTI_RTSR）和 EXTI 下降沿触发选择寄存器（EXTI_FTSR）

如图 5.11 和图 5.12 所示的两个寄存器用于配置某个中断线是否可被上升沿或下降沿触发。在 Cortex-M33 架构的单片机中，这两个寄存器各有 26 个有效位，对应 26 根 EXTI 线（特定型号的芯片不会全部用到），而在 Cortex-M3 架构的单片机中，只有 20 个。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	RT25	RT24	RT23	RT22	RT21	RT20	RT19	RT18	RT17	RT16
						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
RT15	RT14	RT13	RT12	RT11	RT10	RT9	RT8	RT7	RT6	RT5	RT4	RT3	RT2	RT1	RT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 5.11　EXTI 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
Res.	Res.	Res.	Res.	Res.	Res.	FT25	FT24	FT23	FT22	FT21	FT20	FT19	FT18	FT17	FT16
						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
FT15	FT14	FT13	FT12	FT11	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3	FT2	FT1	FT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 5.12　EXTI 下降沿触发选择寄存器 EXTI_FTSR

寄存器中的每个位对应一根 EXTI 线。当该位置 1 时，即可设置为上升沿或下降沿触发。如果两个寄存器对应位都被设置成了 1，则上升沿和下降沿都触发。

2. EXTI 软件中断事件寄存器（EXTI_SWIER）

EXTI_SWIER 寄存器用于软件触发事件，每个位对应每根 EXTI 线。当向对应位写 1 时，则可以软件触发该 EXTI 线上的事件，如图 5.13 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	SWI25	SWI24	SWI23	SWI22	SWI21	SWI20	SWI19	SWI18	SWI17	SWI16
						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
SWI15	SWI14	SWI13	SWI12	SWI11	SWI10	SWI9	SWI8	SWI7	SWI6	SWI5	SWI4	SWI3	SWI2	SWI1	SWI0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 5.13　EXTI 软件中断事件寄存器 EXTI_SWIER

3. EXTI 外部中断选择寄存器（EXTI_EXTICRm）

EXTI_EXTICRm 寄存器用于配置 EXTI0～EXTI15 这 16 根 EXTI 线连接到哪个引脚

上(m=1～4),每个寄存器可配置 4 根 EXTI 线,每根 EXTI 线使用 8 个位进行配置,如图 5.14 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EXTI{4*(m-1)+3}[7:0]								EXTI{4*(m-1)+2}[7:0]							
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
EXTI{4*(m-1)+1}[7:0]								EXTI{4*(m-1)}[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 5.14　EXTI 外部中断选择寄存器 EXTI_EXTICRm

PA～PJ 对应 0x00～0x09 的写入值。例如,如果想配置 EXTI0 由 PC0 引脚触发,那么 EXTICR1 的位[7：0]应该被写入 0x02。如果想配置 EXTI5 由 PE5 引脚触发,那么 EXTICR2 的位[15：8]应该写入 0x04。

5.3.6　EXTI 的 HAL 库配置流程

配置 EXTI 的中断并编写中断服务程序共涉及以下几个流程。

(1) 设置中断触发条件(在 STM32CubeMX 中配置)。如配置 GPIO 的某个引脚采用上升沿或下降沿触发。

(2) 设置中断优先级(在 STM32CubeMX 中配置)。涉及使用 HAL_NVIC_SetPriority()设置中断优先级,如 HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0)。

(3) 使能中断(在 STM32CubeMX 中配置)。涉及使用 HAL_NVIC_EnableIRQ()使能中断,如 HAL_NVIC_EnableIRQ(EXTI0_IRQn)。

(4) 在中断服务程序中判断中断外设来源。不同的中断会调用不同的中断服务程序,如 EXTI0_IRQHandler()、TIM2_IRQHandler()等。这些函数的名称通常以 PPP_IRQHandler()命名(PPP 为外设简称)。在这些中断服务程序中又会调用外设通用中断处理函数 HAL_PPP_IRQHandler()(PPP 为外设简称)来向通用函数中引入外设句柄或名称。在 HAL 库中,同一种类型的外设会调用同一个通用中断处理函数。

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
```

(5) 在外设通用中断处理函数中会依据引入的外设句柄或名称来访问对应的中断标志位,判断到底哪个外设发生了中断,并清除该标志位,然后访问外设通用中断回调函数 HAL_PPP_Callback()(PPP 为外设简称)。

```
/**
  * @brief  This function handles EXTI interrupt request.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
```

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}
```

(6) 执行中断回调函数。外设通用中断回调函数 HAL_PPP_Callback()是弱函数,可以在其他文件中重定义(一般写在 main.c 文件中),从而根据用户需求编写程序。

```
/**
  * @brief  EXTI line detection callbacks.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

在上面的步骤中,执行中断时主要涉及 3 类函数:外设中断服务程序 PPP_IRQHandler()、HAL 库外设通用中断处理函数 HAL_PPP_IRQHandler()和外设通用中断回调函数 HAL_PPP_Callback()。图 5.15 再次表明了它们之间的关系。

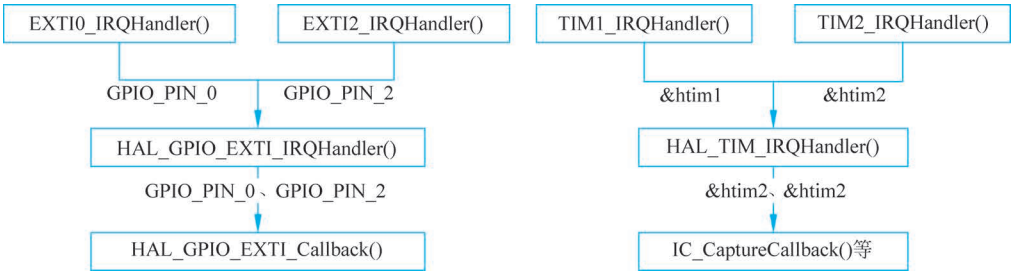

图 5.15　中断处理函数之间的关系

图 5.15 表明,不同中断来源所对应的中断服务程序(ISR)入口地址(启动文件中定义)不同,但是同一类外设会调用相同的 HAL 库外设通用中断处理函数 HAL_PPP_IRQHandler(),进而调用相同的外设通用中断回调函数 HAL_PPP_Callback()。这主要是因为 HAL 库的设计考虑到了可移植性和模块化。

5.4　实验:用外部中断进行按键上升沿/下降沿检测


视频讲解

5.4.1　应用场景及目的

第 4 章进行的按键实验是依靠轮询的方式判断状态的,这会大量占用 CPU 的资源,也

会导致程序在执行其他任务时检测不到突然的电平变化。在实际应用中，更多地会采用外部中断的方式来判断按键是否按下。当按键按下时，连接按键的引脚会触发一个电平变化。电平由高到低变化会产生下降沿，由低到高变化会产生上升沿。可以通过利用边沿的变化来触发中断，进而执行相应的动作。

本实验使用底板上的 USER 按键和五向按键来控制核心板上 D1 处的 LED 灯的亮灭。当按下 USER 按键时，点亮 LED 灯，当按下五向按键时，关闭 LED 灯。

本实验对应的例程为 04-1_STM32U575_GPIO_EXTI_Rise_Fall。

5.4.2 原理图

先来看 USER 按键的连接原理图，如图 5.16 所示。可以看到当按下 USER 按键后，USER 引脚会变为低电平，也就是产生下降沿。因此需要将与 USER 引脚相连的 GPIO 引脚 PA12 设置为下降沿触发中断。

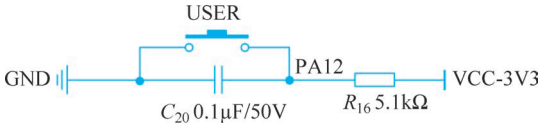

图 5.16 USER 按键原理图

在底板原理图中包含了五向按键的电路，如图 5.17 所示。U7B 处的运算放大器被连接为电压跟随器，U7A 处的运算放大器被连接为电压比较器。当按键抬起时，U7B 的 5 号引脚为 3.3V 电压，7 号引脚也输出 3.3V 电压。7 号引脚与 U7A 处的 3 号引脚相连。U7A 处的 2 号引脚通过 R_{30} 与 R_{31} 的分压会得到 2.5V 电压。由于 U7A 处的 3 号引脚电压大于 2 号引脚，因此会在 1 号引脚处输出 5V 电压。此时 Q3 处的场效应管导通，IO INT4 引脚为低电平。当五向按键按下时，IO INT4 引脚为高电平。因此按下五向按键会产生一个上升沿。

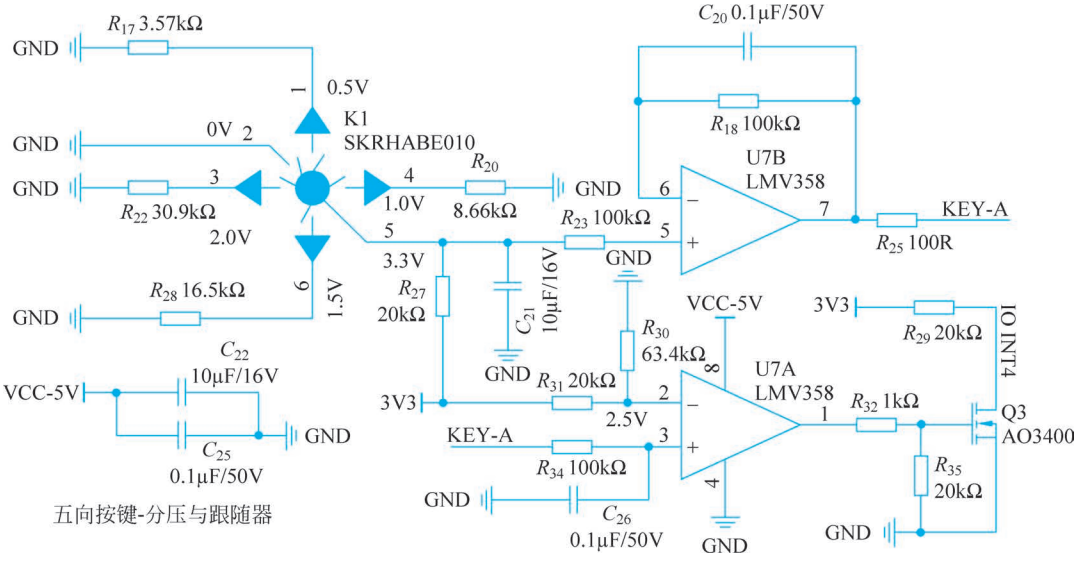

图 5.17 五向按键原理图

通过底板原理图的 J2（见图 5.18）可知，IO INT4 连接到了 PA0 上。PA0 需要被设置为上升沿触发才能在按下五向按键时触发中断。

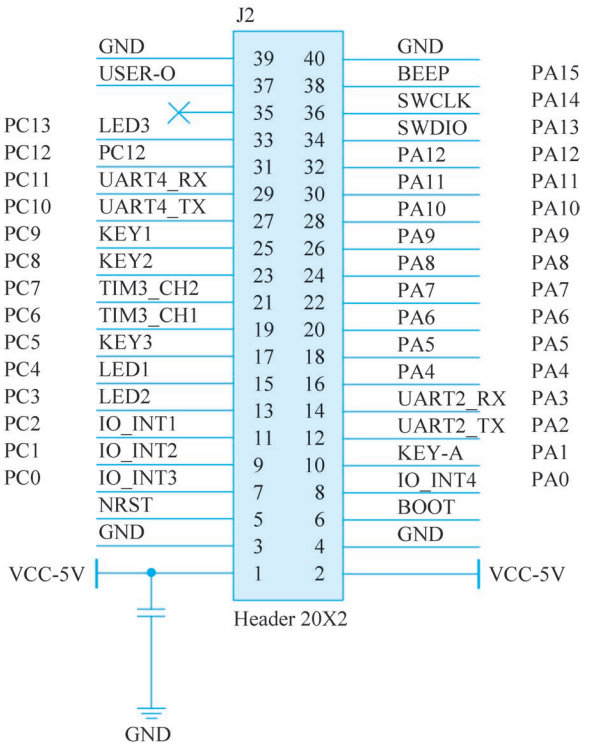

图 5.18　底板原理图的 J2 处插座

5.4.3　程序流程

本实验程序流程如图 5.19 所示。

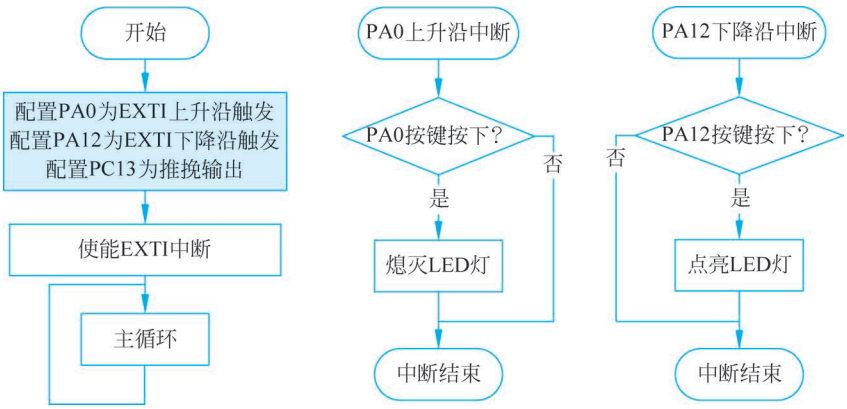

图 5.19　按键中断检测实验程序流程

通过流程图可知，PA0 需要配置为上升沿触发中断，PA12 需要配置为下降沿触发中断，然后在使能对应的 EXTI 中断后，进入空的主循环。后续任务完全由按键按下产生的中断所对应的中断服务程序执行。

5.4.4　程序配置

打开 STM32CubeMX，根据芯片型号新建一个工程。在打开的页面中，分别配置 PA0

和 PA12 为 EXTI0 和 EXTI12，设置为上拉输入模式，分别配置为上升沿触发和下降沿触发，如图 5.20 所示。

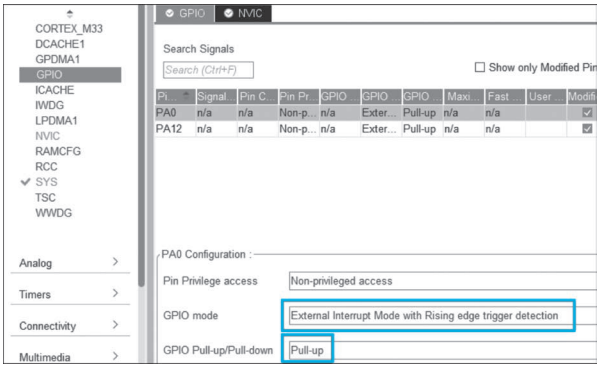

图 5.20　GPIO 配置页面

然后在 NVIC 标签页使能中断，如图 5.21 所示。

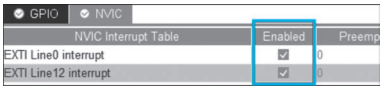

图 5.21　使能 GPIO 的中断

如图 5.22 所示的 NVIC 配置页面自动选中了两个中断的 Generate IRQ handler 和 Call HAL handler。这对应 stm32 ** xx_it.c 文件中的 EXTI0_IRQHandler()、EXTI12_IRQHandler()和 HAL_GPIO_EXTI_IRQHandler()。

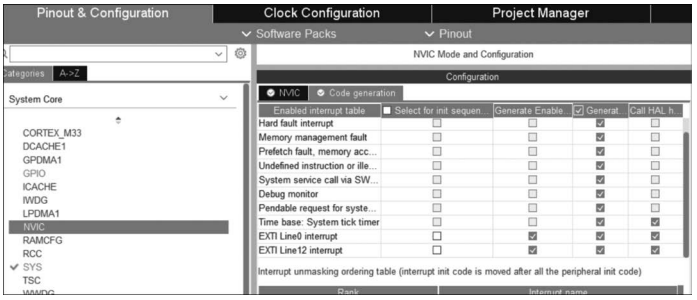

图 5.22　NVIC 配置页面

由于本实验要控制核心板上 D1 处的 LED 灯，该灯连接到了 PC13 引脚，因此 PC13 引脚需要被配置为推挽输出模式，默认为低电平，如图 5.23 所示。

图 5.23　PC13 引脚的配置

配置英文工程目录 Toolchain/MDK，设置为复制必要的文件，分别生成.c/.h 文件后，生成并打开工程。

main.c 文件中调用的 MX_GPIO_Init()函数中配置了对应的引脚模式，并使能了中断。

在 stm32u5xx_it.c 文件中，分别生成了 EXTI0 和 EXTI12 的中断处理函数。在对应的函数中又调用了 HAL_GPIO_EXTI_IRQHandler()函数并输入了对应的引脚编号。

```
/**
  * @brief This function handles EXTI Line0 interrupt.
  */
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
  /* USER CODE END EXTI0_IRQn 1 */
}
/**
  * @brief This function handles EXTI Line12 interrupt.
  */
void EXTI12_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI12_IRQn 0 */
  /* USER CODE END EXTI12_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_12);
  /* USER CODE BEGIN EXTI12_IRQn 1 */
  /* USER CODE END EXTI12_IRQn 1 */
}
```

在 HAL_GPIO_EXTI_IRQHandler()函数中，通过输入引脚编号来判断是否为该引脚产生的中断标志。若是对应引脚，则清除该中断标志，然后调用 HAL_GPIO_EXTI_Falling_Callback()或 HAL_GPIO_EXTI_Rising_Callback()函数进入回调函数。

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if (__HAL_GPIO_EXTI_GET_RISING_IT(GPIO_Pin) != 0U)
  {
    __HAL_GPIO_EXTI_CLEAR_RISING_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Rising_Callback(GPIO_Pin);
  }

  if (__HAL_GPIO_EXTI_GET_FALLING_IT(GPIO_Pin) != 0U)
  {
    __HAL_GPIO_EXTI_CLEAR_FALLING_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Falling_Callback(GPIO_Pin);
  }
}
```

在 main.c 文件的 USER CODE BEGIN 4 后面重新编写这两个函数：

```
void HAL_GPIO_EXTI_Falling_Callback(uint16_t GPIO_Pin)                    //下降沿触发的回调函数
{
    if(GPIO_Pin == GPIO_PIN_12)
    {
        if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_12) == GPIO_PIN_RESET) //读取按键是否按下
        {
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,GPIO_PIN_SET);    //点亮 LED 灯
        }
    }
}

void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)                     //上升沿触发的回调函数
{
    if(GPIO_Pin == GPIO_PIN_0)
    {
        if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) == GPIO_PIN_SET)     //读取按键是否按下
        {
            HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,GPIO_PIN_RESET); //熄灭 LED 灯
        }
    }
}
```

在下降沿触发的回调函数中，先判断触发下降沿中断的引脚是不是 GPIO_PIN_12。这里不用判断是 PA12 还是 PB12，因为 PA12、PB12、PC12 这些引脚都在 EXTI12 线上，只有其中一个能够被设置成 EXTI 模式。如果是 GPIO_PIN_12 产生的，则设置 PC13 为高电平，点亮 LED 灯。

另外，在 NVIC 页面中 HAL_Delay()函数所依赖的 System tick timer 默认的抢占优先级是最低的，如图 5.24 所示。因此，如果此时在 EXTI 中断回调函数中调用 HAL_Delay()进行延时用于消抖的话，会导致程序无法正常执行。因为在执行高抢占优先级的中断时，无法执行低抢占优先级的中断，HAL_Delay()无法获得计时值。

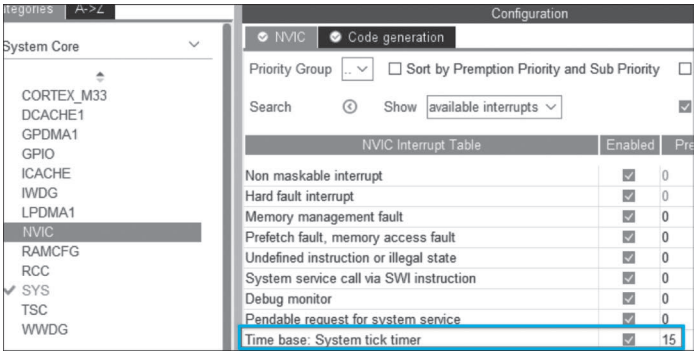

图 5.24　NVIC 中的 Systick 中断优先级配置

5.4.5　实验现象

编译工程，选择好调试下载器后，将程序下载到开发板上。复位后按下 USER 按键，即可看到核心板上 D1 处的 LED 灯点亮；按下五向键，即可看到该灯熄灭。

5.5　习题

简答题

1. 什么是中断？它在程序执行过程中有什么作用？

2. NVIC 的抢占优先级和子优先级有什么区别？

3. PA12 在 EXTI 的几号线上？能同时使能 PA1 和 PC1 为外部中断模式吗？尝试从芯片手册中找到依据。

4. 什么是中断向量表？发生对应中断时，程序是如何跳转到对应的中断服务程序中的？

5. 如果用户设置的两个中断的抢占优先级和子优先级相同，那么如何判断哪个优先级高？找到官方手册对应的说明位置。

思考题

设计一个方案，利用外部中断实现按下按键时打开某个外设，松开按键时关闭某个外设，主循环中不写任何程序。