第七章　中断系统

主讲人：漆强

ytqiqiang@163.com

本章内容

- 中断概述
- HAL库的中断处理流程
- 外部中断的数据类型及接口函数
- 任务实践

教学目标

- 掌握中断的基本概念和作用
- 了解HAL库的中断处理流程
- 熟练运用外部中断进行程序设计

7.1 中断概述

1 中断相关的基本概念

数据传输方式

处理器和外部设备的数据传输方式

无条件传输

处理器不必了解外部设备状态，直接进行数据传输，用于指示灯和按键等简单设备

查询方式

传输前，一方先查询另一方的状态，若已经准备好就传输，否则就继续查询

中断方式

一方通过申请中断的方式与另一方进行数据传输，收发双方可以并行工作

直接存储器访问

处理器内部建立片内外设和内存之间的数据传输通道，传输过程不需要处理器参与

中断概念

中断全过程

1 中断发生 当CPU在处理某一事件A时，发生了另一事件B，请求CPU迅速去处理

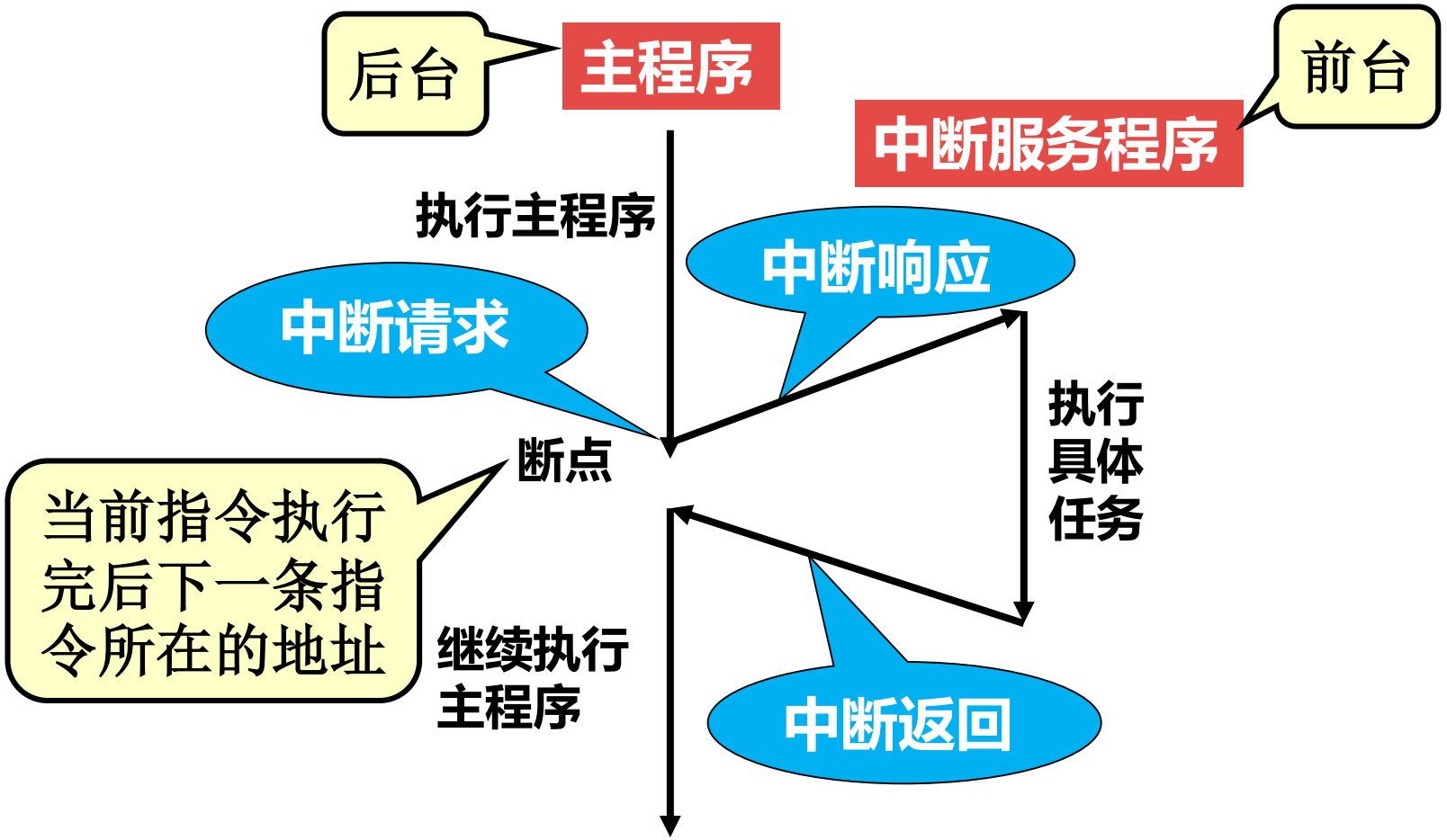
2 中断处理 CPU暂停当前的工作，转去处理事件B

3 中断返回 当CPU将事件B处理完毕后，再回到事件A中被暂停的地方继续处理事件A

整个过程称为中断

示意图

中断程序执行过程示意图



中断的作用

速度匹配 01

可以解决快速的CPU与慢速的外部设备之间传送数据的矛盾

分时操作 02

CPU可以分时为多个外部设备服务，提高计算机的利用率

实时响应 03

CPU能够及时处理应用系统的随机事件，增强系统的实时性

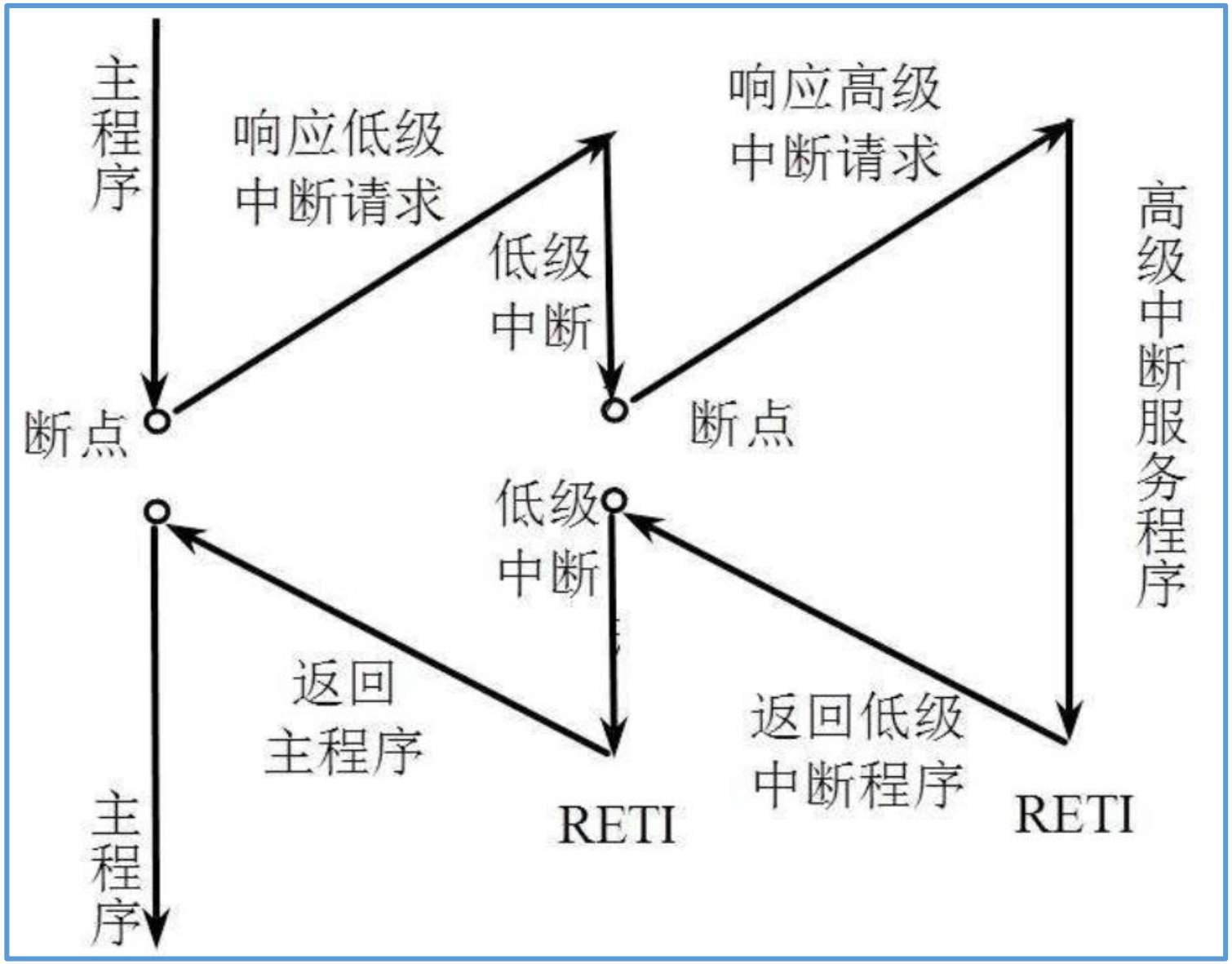
可靠性高 04

CPU可以处理设备故障及掉电等突发事件，提高系统可靠性

中断优先级

处理器根据不同中断的重要程序设置不同的优先等级。不同优先级中断的处理原则是：高级中断可以打断低级中断；低级中断不能打断高级中断

中断嵌套



中断向量

中断服务程序

在响应一个特定中断的时候，处理器会执行一个函数，该函数一般称为中断处理程序或者中断服务程序

中断向量和中断向量表

- **中断向量** ：中断服务程序在内存中的入口地址称为中断向量。
- **中断向量表**：把系统中所有的中断向量集中起来放到存储器的某一区域内，这个存放中断向量的存储区就叫中断向量表

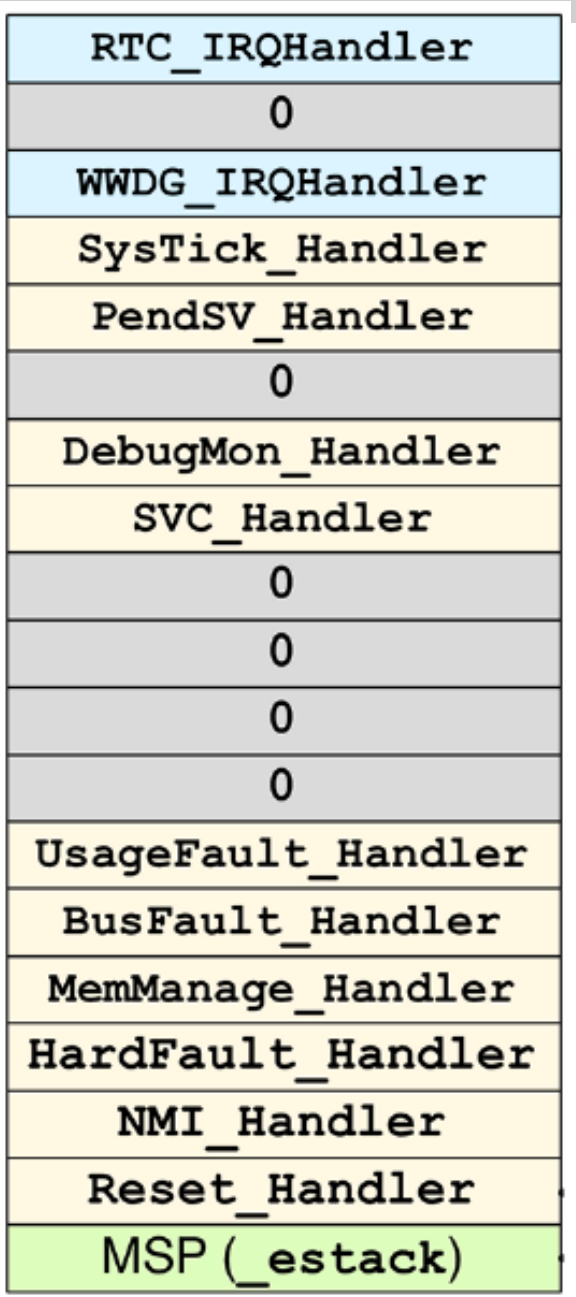
启动文件

startup_stm32f411xe.s

```
__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Hand
                DCD     MemManage_Handler         ; MPU Fault Handl
                DCD     BusFault_Handler          ; Bus Fault Handl
                DCD     UsageFault_Handler        ; Usage Fault Han
                . .
                External Interrupts
                . . .
                DCD     EXTI0_IRQHandler          ; EXTI Line0
                DCD     EXTI1_IRQHandler          ; EXTI Line1
                DCD     EXTI2_IRQHandler          ; EXTI Line2
                DCD     EXTI3_IRQHandler          ; EXTI Line3
                DCD     EXTI4_IRQHandler          ; EXTI Line4
```

类似于数组的定义

各个中断源对应的中断服务程序



查找中断向量

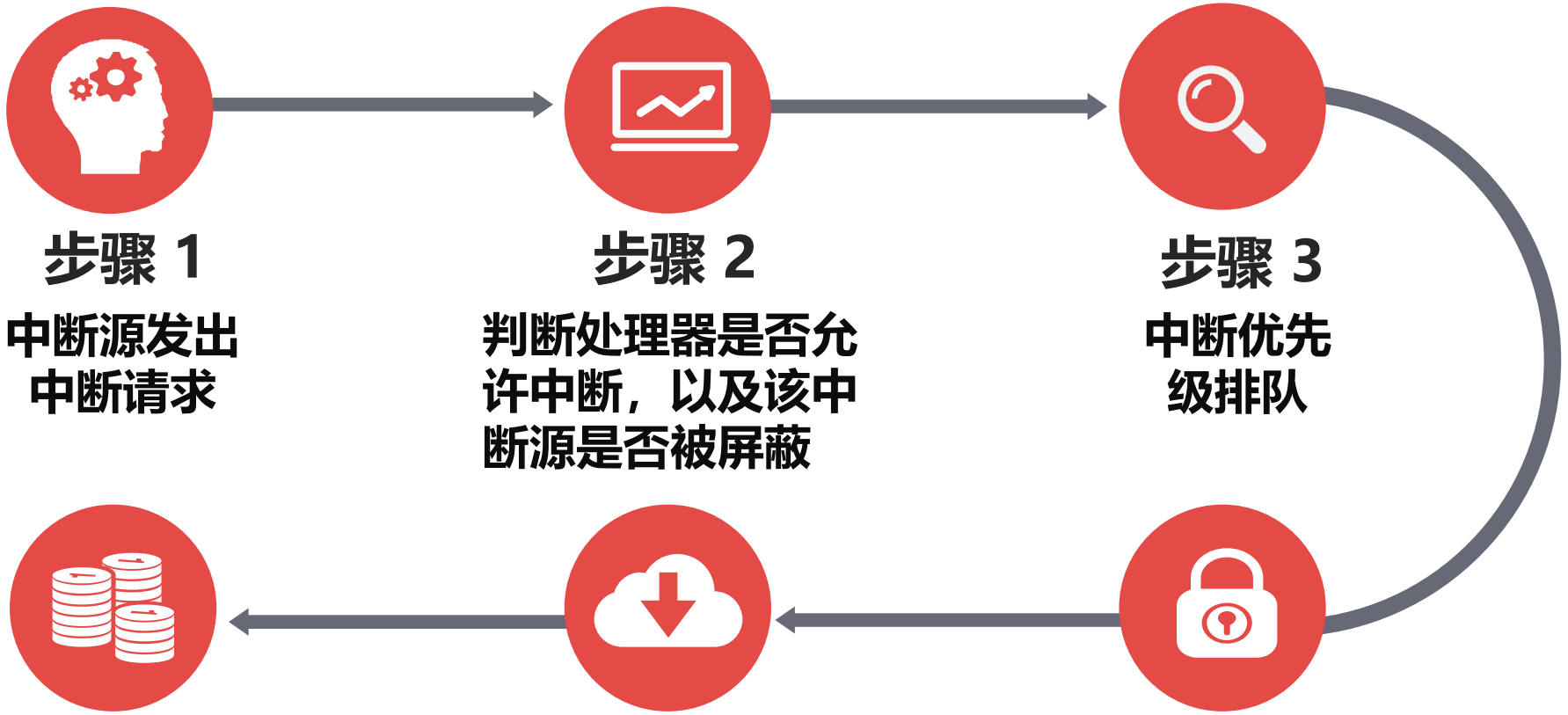
查找中断向量的过程

计算机系统对每一个中断源进行编号，这个号码称为中断类型号

根据中断类型号，到中断向量表中找到对应的表项

取出表项内容，即该中断源对应的中断服务程序地址，进入该程序执行相应操作

中断响应过程



步骤 1

中断源发出中断请求

步骤 2

判断处理器是否允许中断，以及该中断源是否被屏蔽

步骤 3

中断优先级排队

步骤 4

处理器暂停当前程序，保护断点地址和处理器的当前状态，根据中断类型号，查找中断向量表，转到对应的中断服务程序

步骤 5

执行中断服务程序

步骤 6

恢复被保护的状态，执行中断返回指令，回到被中断的程序

电子科技大学
University of Electronic Science and Technology of China

2 STM32微控制器中断系统

基本概念

中断和异常

中断

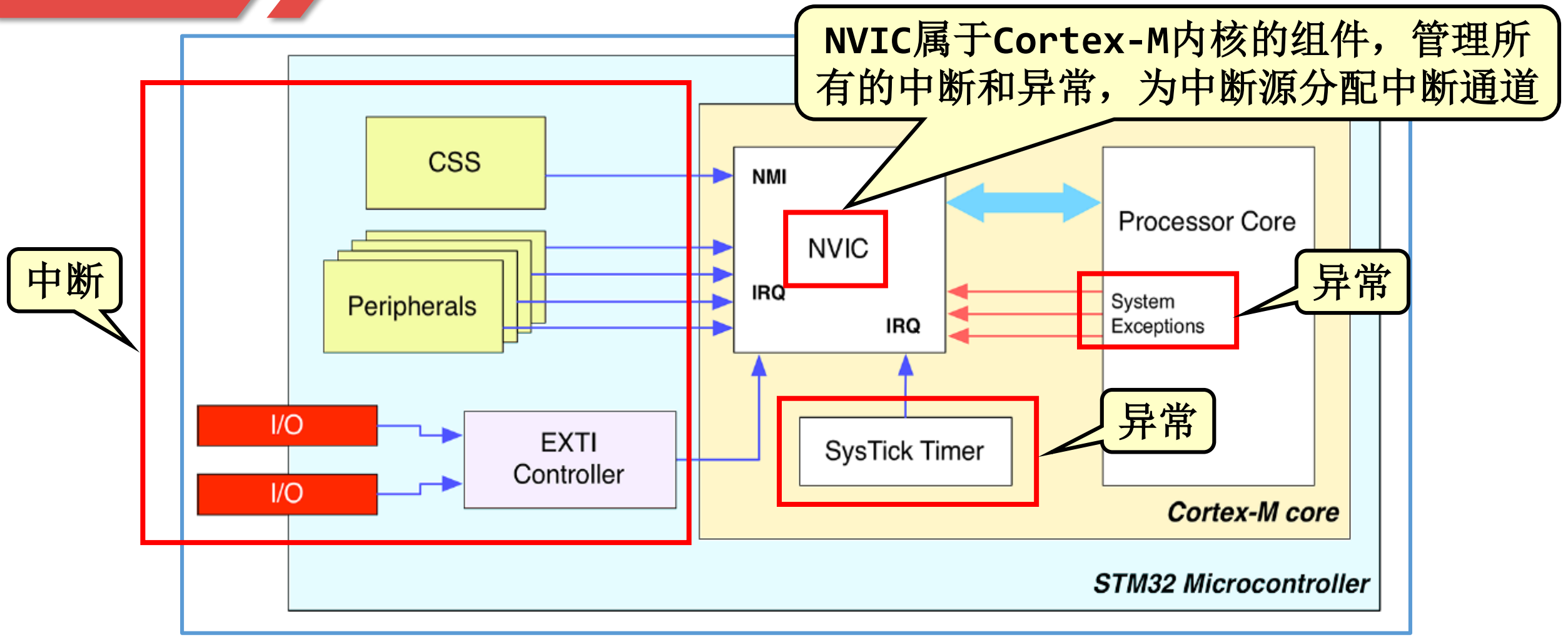
中断是由内核外部产生的，一般由硬件引起，比如外设中断和外部中断等

异常

异常通常是内核自身产生的，大多是软件引起的，比如除法出错异常、预取值失败等

NVIC

嵌套向量中断控制器



中断通道

中断通道

微控制器片内集成了很多外设，对于单个外设而言，它通常具备若干个可以引起中断的中断源，而该外设的所有中断源只能通过指定的中断通道向内核申请中断。

以STM32F411芯片为例，它支持68个中断通道，已经固定分配给相应的片内外设。由于中断源数量较多，而中断通道有限，会出现多个中断源共享同一个中断通道的情况。

中断优先级

- NVIC中有一个8位中断优先级寄存器NVIC_IPR，理论上可以配置0~255共256级中断
- STM32只使用了其中的高4位，并分成抢占优先级和子优先级两组

中断嵌套

- 多个中断同时提出中断申请时：先比较抢占优先级，抢占优先级高的中断先执行。
- 如果抢占优先级相同，则比较子优先级。
- 二者都相同时，比较中断编号。编号越小，优先级越高。

中断编号位于芯片头文件中

优先级分组

STM32中断优先级分组

优先级分组	抢占优先级	子优先级
第0组：NVIC_PriorityGroup_0	无	4位/16级（0~15）
第1组：NVIC_PriorityGroup_1	1位/2级（0~1）	3位/8级（0~7）
第2组：NVIC_PriorityGroup_2	2位/4级（0~3）	2位/4级（0~3）
第3组：NVIC_PriorityGroup_3	3位/8级（0~7）	1位/2级（0~1）
第4组：NVIC_PriorityGroup_4	4位/16级（0~15）	无

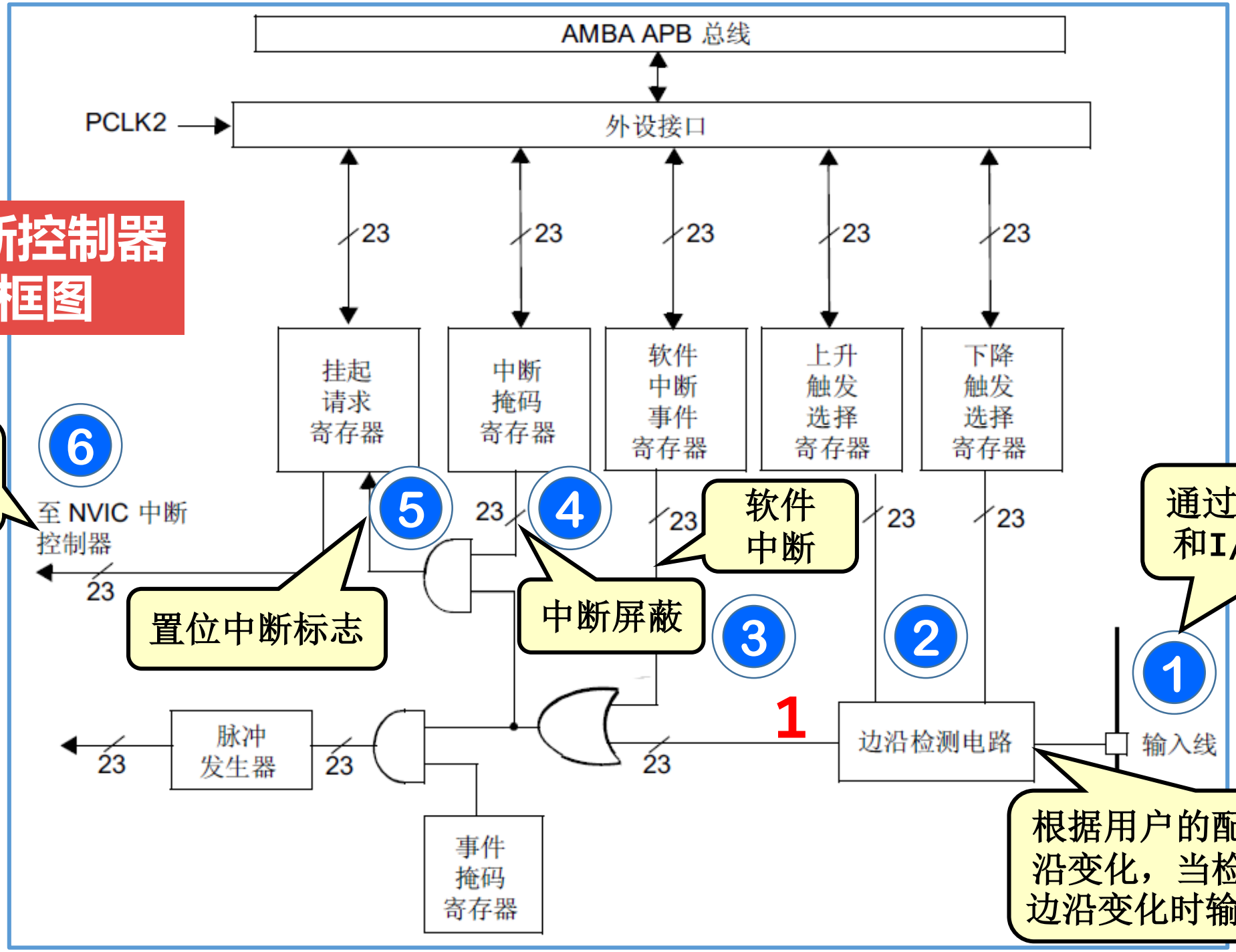
HAL库初始化函数HAL_Init将优先级分组设置为第4组，即有0～15，共16级抢占优先级，没有子优先级。编号越小的优先级越高：0号为最高，15号为最低。

EXTI

外部中断控制器

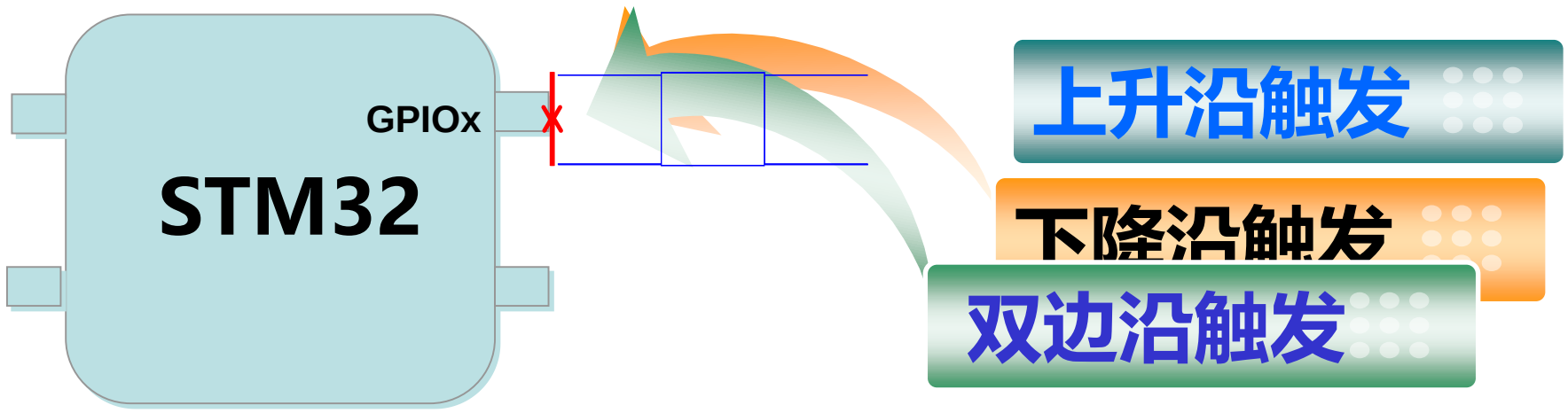
- 管理23个外部中断线（EXTI Line）
- 0～15号外部中断线用于由GPIO引脚触发的外部中断
- 16～22号外部中断线用于RTC闹钟事件、以太网唤醒事件和USB唤醒事件等
- 当对应GPIO引脚与外部中断线连接后，GPIO引脚才具备外部中断的功能，可以设置外部中断的触发方式

外部中断控制器结构框图



GPIO引脚

GPIO引脚的外部中断触发方式



引脚分组

引脚分组

① 尾号相同的引脚一组，接入1个外部中断线

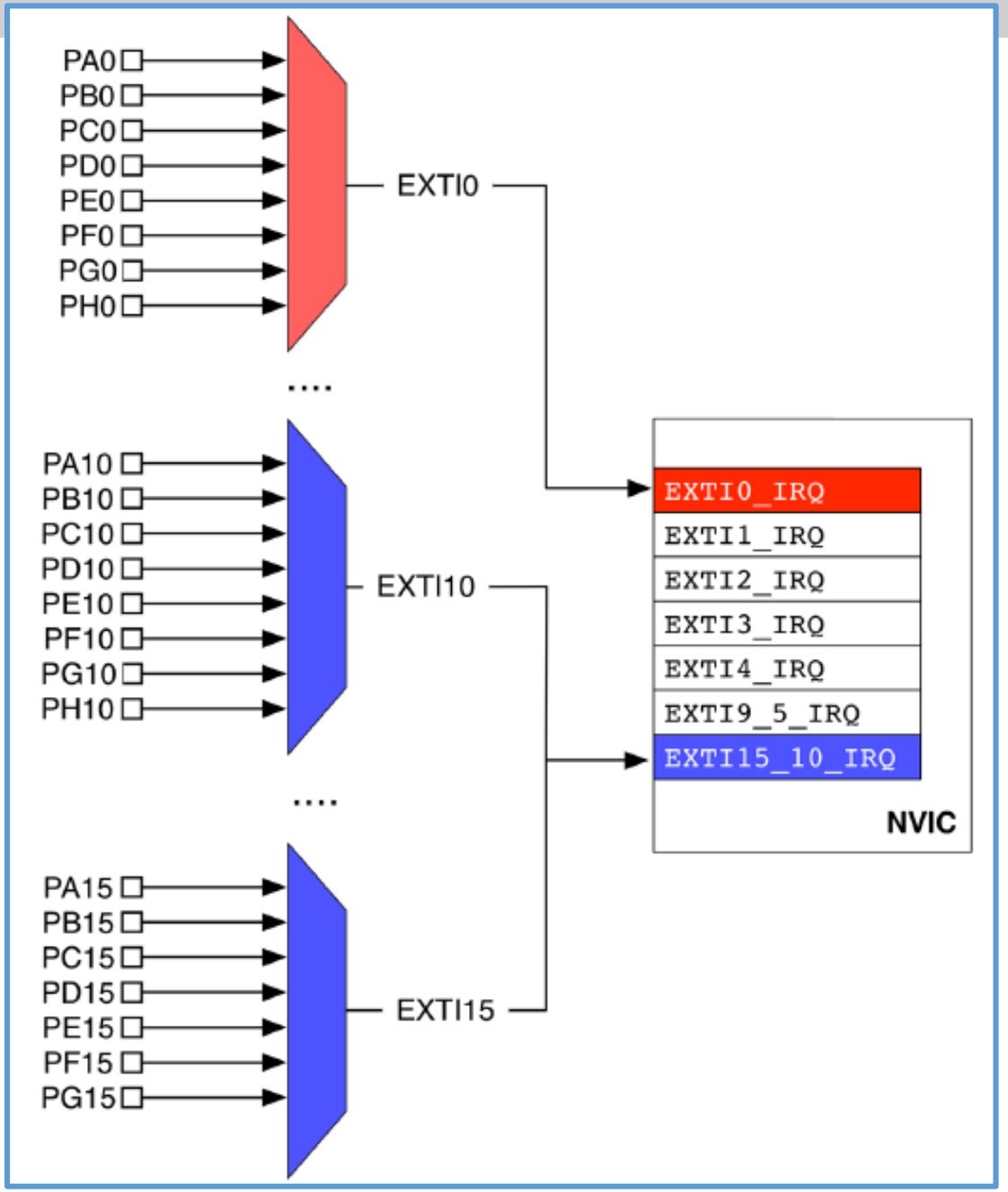
② 同组引脚只能有一个设置为外部中断功能

中断通道

① EXTI0~EXIT4分别具有独立的中断通道

② EXTI5~EXIT9共享同一个中断通道

③ EXTI10~EXIT15共享同一个中断通道

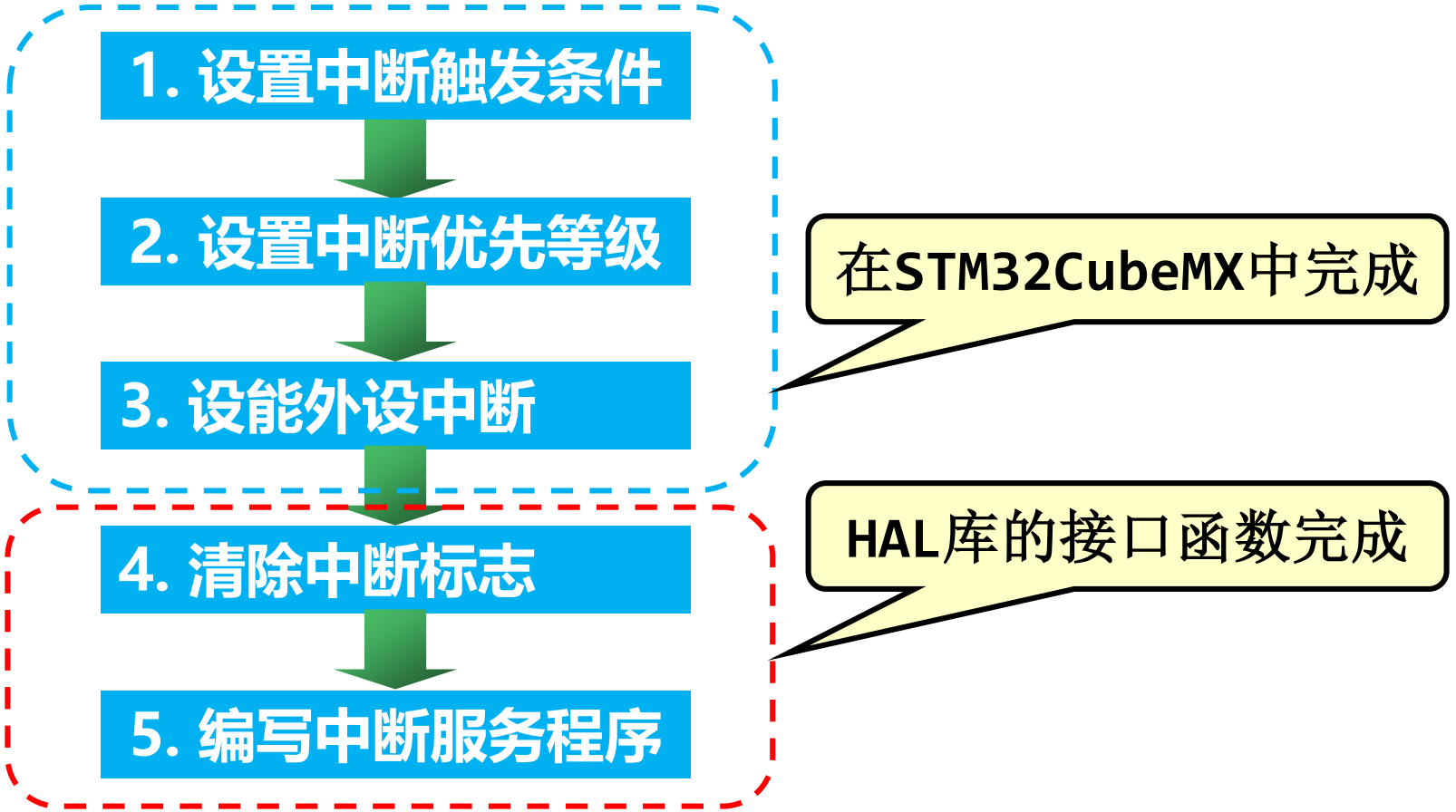


7.2 HAL库的中断处理流程

1 HAL库的中断封装

编程步骤

中断程序的编程步骤



HAL库封装

HAL库对中断的封装处理

PPP代表外设名称

一：统一规定处理各个外设的中断服务程序HAL_PPP_IRQHandler

二：在中断服务程序HAL_PPP_IRQHandler完成了中断标志的判断和清除

由外设初始化、中断、处理完成/出错触发的函数

三：将中断中需要执行的操作以回调函数的形式提供给用户

启动文件

由CubeMX生成的MDK工程中与中断相关的编程文件

启动文件：startup_stm32fxxx.s

① 该文件存放在MDK-ARM组中。在该文件中，预先为每个中断编写了一个中断服务程序，只是这些中断服务程序都是死循环，目的只是初始化中断向量表；

② 中断服务程序的属性定义为“weak”。weak属性的函数表示：如果该函数没有在其他文件中定义，则使用该函数；如果用户在其他地方定义了该函数，则使用用户定义的函数。

中断文件

由CubeMX生成的MDK工程中与中断相关的编程文件

中断服务程序文件：stm32fxxx_it.c

① 该文件存放在User组中，用于存放各个中断的中断服务程序；

② 在使用CubeMX软件进行初始化配置时，如果使能了某一个外设的中断功能，那么在生成代码时，相对应的外设中断服务程序HAL_PPP_IRQHandler就会自动添加到该文件中，用户只需要在该函数中添加相应的中断处理代码即可。

2 外部中断处理流程

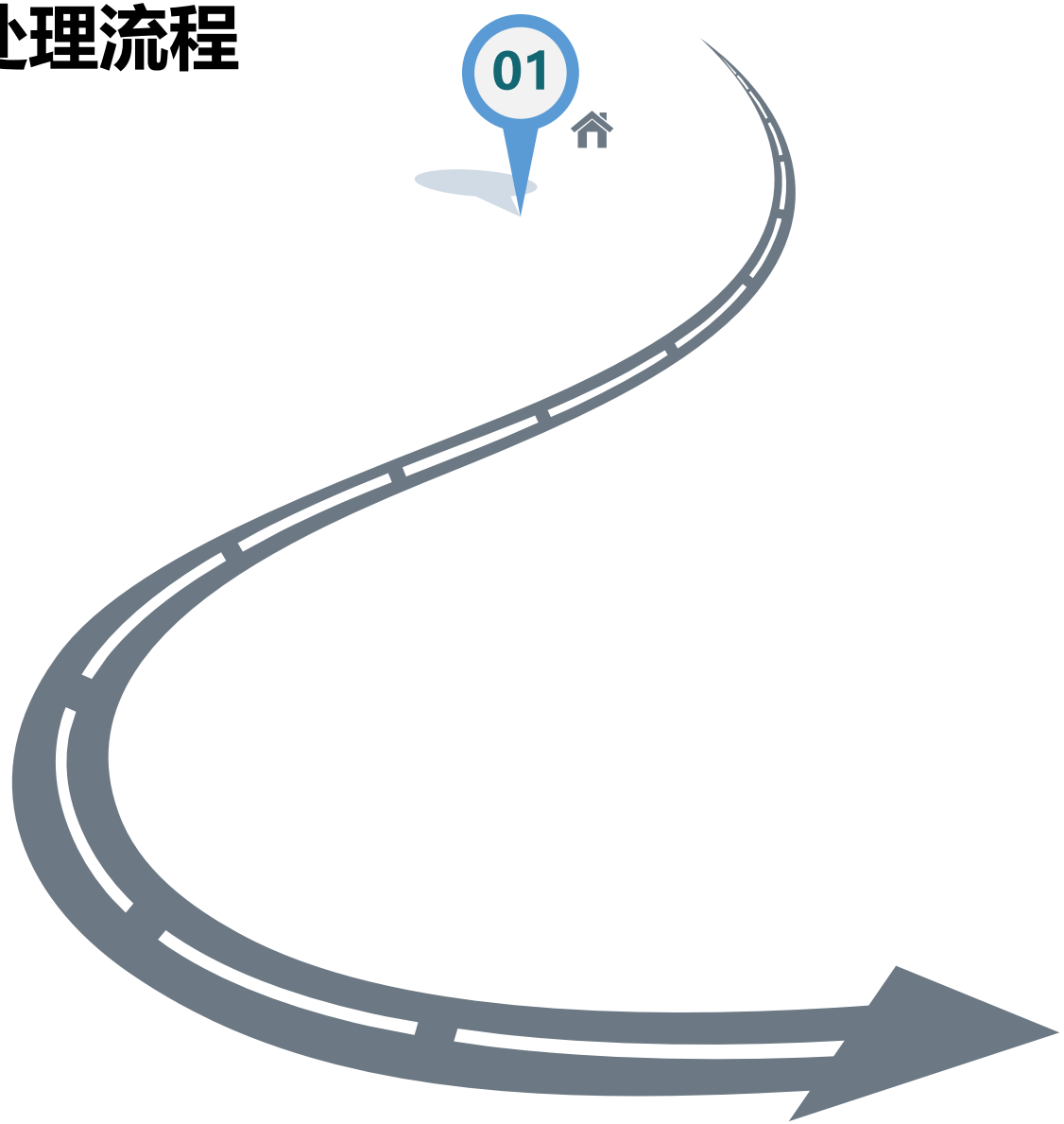
任务模拟

以外部中断为例分析HAL库的中断处理流程

- 假设微控制器芯片为STM32F411，设置引脚PC0和PC13为外部中断功能。
- 当引脚PC0或PC13出现脉冲边沿时，将触发外部中断。

处理流程

HAL库外部中断处理流程



中断跳转

跳转到该中断所对应的中断服务程序

ISR

外部中断所对应的中断服务程序

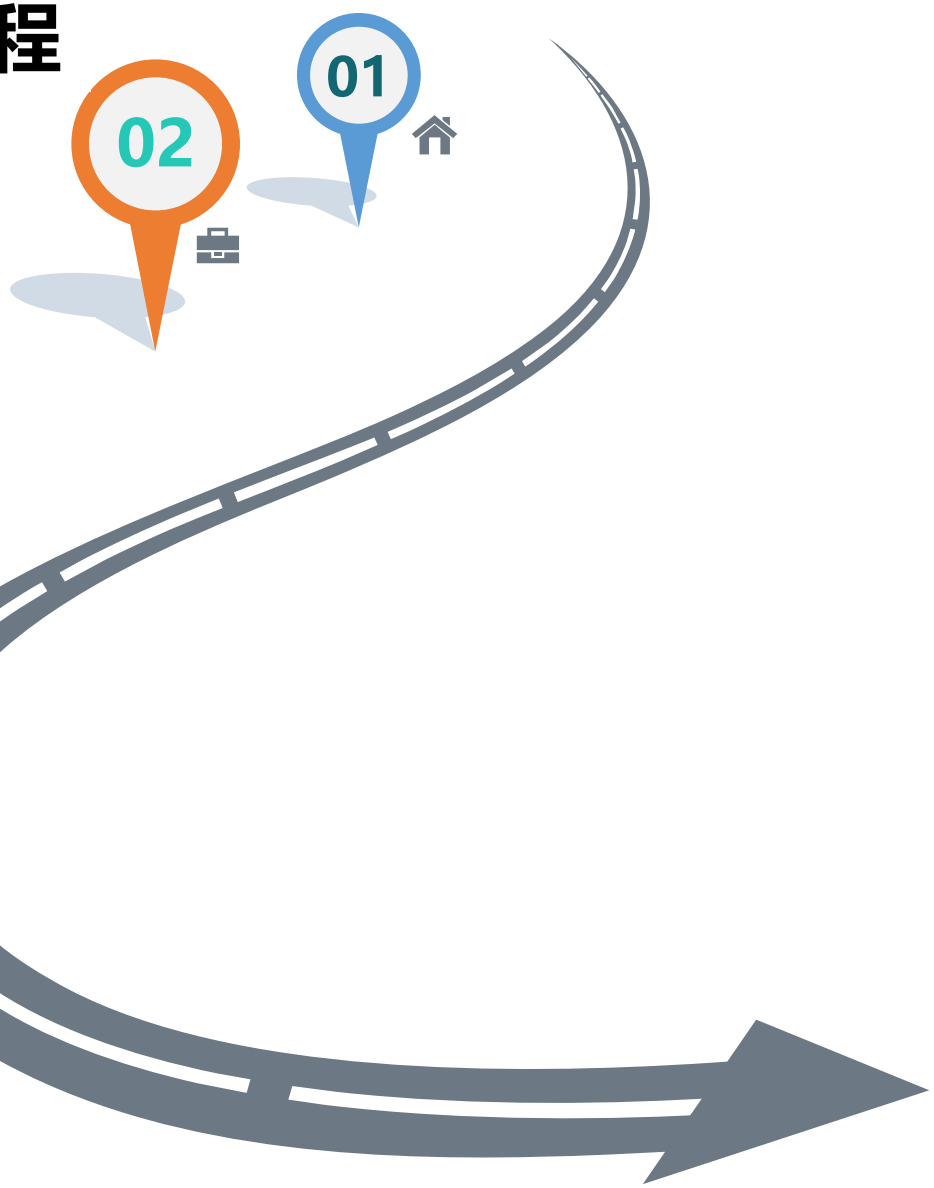
引脚PC0对应的
外部中断服务程序

外部中断线	中断服务程序的函数名称
外部中断线0(EXTI Line 0)	EXTI0_IRQHandler
外部中断线1(EXTI Line 1)	EXTI1_IRQHandler
外部中断线2(EXTI Line 2)	EXTI2_IRQHandler
外部中断线3(EXTI Line 3)	EXTI3_IRQHandler
外部中断线4(EXTI Line 4	EXTI4_IRQHandler
外部中断线5～9(EXTI Line[9:5])	EXTI9_5_IRQHandler
外部中断线10～15(EXTI Line[15:10])	EXTI15_10_IRQHandler

引脚PC13对应的
外部中断服务程序

处理流程

HAL库外部中断处理流程



中断跳转

跳转到该中断所对应的中断服务程序

执行中断服务程序

执行在stm32f4xx_it.c文件中对应的中断服务程序

stm32f4xx_it.c文件中的外部中断服务程序

```
1. /*
2.  * @brief This function handles EXTI line0 interrupt.
3.  */
4. void EXTI0_IRQHandler(void)
5. {
6.   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);   // 调用外部中断通用处理函数
7. }
8. /*
9.  * @brief This function handles EXTI line[15:10] interrupts.
10. */
11.void EXTI15_10_IRQHandler(void)
12.{
13.  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);  // 调用外部中断通用处理函数
14.}
```

外部中断通用处理函数

外部中断通用处理函数

处理流程

HAL库外部中断处理流程

中断跳转

跳转到该中断所对应的中断服务程序

执行中断服务程序

执行在stm32f4xx_it.c中对应的中断服务程序

执行外部中断通用处理函数

判断中断标志并清除，调用外部中断回调函数

stm32f4xx_hal_gpio.c文件中的外部中断通用处理函数

```
1. /*
2.  * @brief  This function handles EXTI interrupt request.
3.  * @param  GPIO_Pin Specifies the pins connected EXTI line
4.  * @retval None
5.  */
6. void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
7. {
8.   /* EXTI line interrupt detected */
9.   if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)  // 检测中断标志
10.    {
11.      __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);       // 清除中断标志
12.      HAL_GPIO_EXTI_Callback(GPIO_Pin);         // 调用回调函数
13.    }
14.  }
```

处理流程

HAL库外部中断处理流程

中断跳转

跳转到该中断所对应的中断服务程序

执行中断服务程序

执行在stm32f4xx_it.c中对应的中断服务程序

外部中断通用处理函数

判断中断标志并清除，调用外部中断回调函数

执行用户编写的回调函数

完成具体的中断任务处理

在main.c文件中编写外部中断的回调函数

```
1. /* USER CODE BEGIN 4 */
2. void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
3. {
4.   switch(GPIO_Pin)
5.   {
6.     case GPIO_PIN_0:                          // 引脚PC0对应的中断处理任务
7.     /* GPIO_PIN_0 EXTI handling */
8.     break;
9.     case GPIO_PIN_13:
10.    /*GPIO_PIN_13 EXTI handling */           // 引脚PC13对应的中断处理任务
11.    break;
12.    case GPIO_PIN_x:                          // 引脚PCx对应的中断任务
13.    /* GPIO_PIN_x EXTI handling */
14.    break;
15.    ......
16.    default:  break;
17.  }
18.}
19./* USER CODE END 4 */
```

所有的外部中断服务程序都会调用该回调函数。如果系统中存在多个外部中断时，需要判断是哪一个GPIO引脚触发的本次外部中断

stm32f4xx_hal_gpio.c文件中定义的默认回调函数

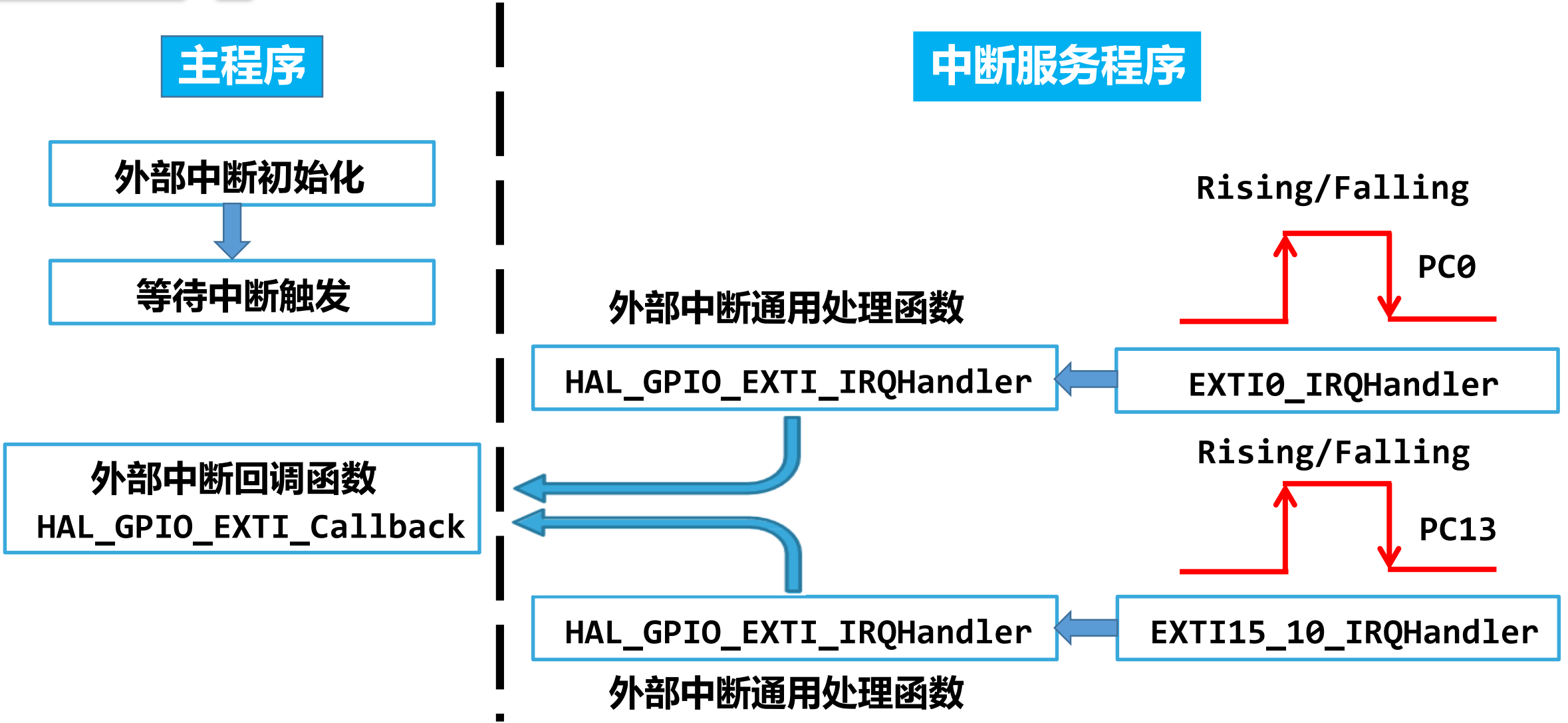
```
1. /*
2.  * @brief  EXTI line detection callbacks.
3.  * @par                                    ns connected EXTI line
4.  * @ret
5.  */
6. __weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
7. {
8.   /* Prevent unused ar                     tion warning */
9.   UNUSED(GPIO_Pin);
10.  /* NOTE:This function Should not be modified, when the
            callback is needed,the HAL_GPIO_EXTI_Callback
            could be implemented in the user file
12.  */
13.  }
```

默认的回调函数采用weak属性定义，用户需要编写同名的回调函数

避免编译器的警告

流程图

HAL库的外部中断处理流程



7.3 外部中断的数据类型及接口函数

所在文件

外部中断数据类型和接口函数所在文件

由于外部中断主要是利用GPIO引脚实现，因此外部中断数据类型的定义放在stm32f4xx_hal_gpio.h文件中，外部中断接口函数的实现放在stm32f4xx_hal_gpio.c文件中。

引脚初始化

引脚初始化数据类型

结构体类型，包括5个成员变量

```
1.  typedef struct
2.  {
3.    uint32_t Pin;    // 指定需要配置的 GPIO 引脚，该参数可以是 GPIO_pins 的值之一
4.    uint32_t Mode;   // 指定所选引脚的工作模式，该参数可以是 GPIO_mode 的值之一
5.    uint32_t Pull;   // 指定所选引脚的上/下拉电阻，该参数可以是 GPIO_pull 的值之一
6.    uint32_t Speed; // 指定所选引脚的速度，该参数可以是 GPIO_speed 的值之一
7.    // 将外设连接至所选择的引脚，该参数可以是 Alternate_function_selection 的值之一
8.    uint32_t Alternate;
9.  }GPIO_InitTypeDef;
```

Mode

成员变量Mode的取值范围

对于外部中断功能增加的取值范围

宏常量定义	含义
GPIO_MODE_IT_RISING	上升沿触发
GPIO_MODE_IT_FALLING	下降沿触发
GPIO_MODE_IT_RISING_FALLING	双边沿触发

1 外部中断通用处理函数

接口函数：HAL_GPIO_EXTI_IRQHandler	
函数原型	void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
功能描述	作为所有外部中断发生后的通用处理函数
入口参数	GPIO_Pin：连接到对应外部中断线的引脚，范围是 GPIO_PIN_0～GPIO_PIN_15
返回值	无
注意事项	1. 所有外部中断服务程序均调用该函数完成中断处理 2. 函数内部根据GPIO_Pin的取值判断中断源，并清除对应外部中断线的中断标志 3. 函数内部调用外部中断回调函数HAL_GPIO_EXTI_Callback完成实际的处理任务 4. 该函数由CubeMX自动生成

2 外部中断回调函数

接口函数：HAL_GPIO_EXTI_Callback	
函数原型	void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
功能描述	外部中断回调函数，用于处理具体的中断任务
入口参数	GPIO_Pin：连接到对应外部中断线的引脚，范围是 GPIO_PIN_0～GPIO_PIN_15
返回值	无
注意事项	1. 该函数由外部中断通用处理函数HAL_GPIO_EXTI_IRQHandler调用，完成所有外部中断的任务处理 2. 函数内部先根据GPIO_Pin的取值来判断中断源，然后执行对应的中断任务 3. 该函数由用户根据实际需求编写

电子科技大学
University of Electronic Science and Technology of China

7.4 任务实践

电子科技大学
University of Electronic Science and Technology of China

1 基础任务：中断方式读取按键

基础任务

基础任务：中断方式读取按键

1 任务目标

掌握CubeMX软件配置外部中断的方法

2 任务内容

采用中断方式检测按键状态，按键按下后执行操作：翻转指示灯LD2的状态。

按键电路

上拉式按键

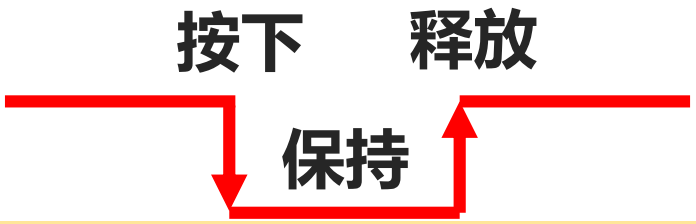
按键按下，引脚PC13读到低电平

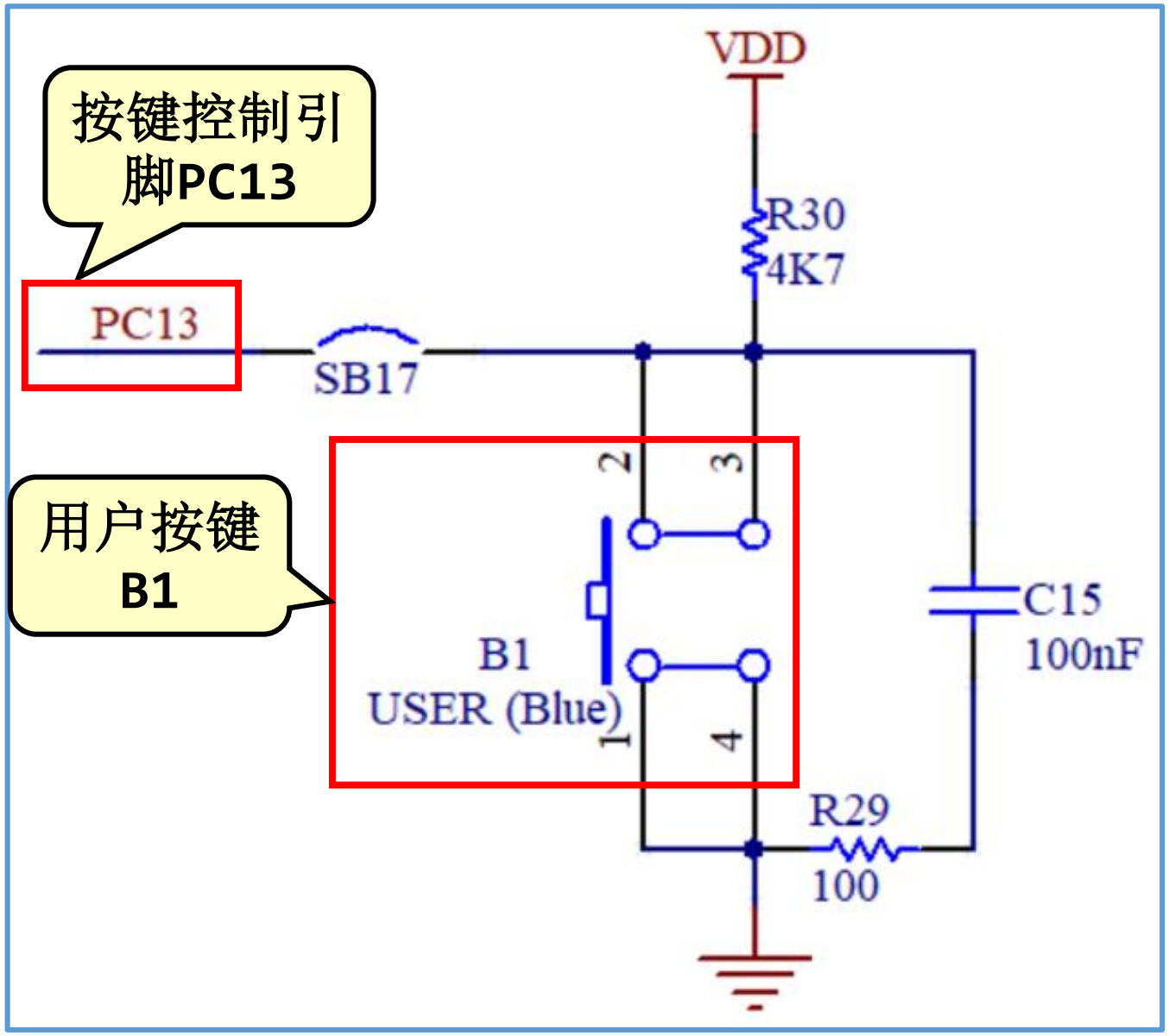
按键释放，引脚PC13读到高电平

触发方式

按键按下瞬间，形成下降沿

按键释放瞬间，形成上升沿

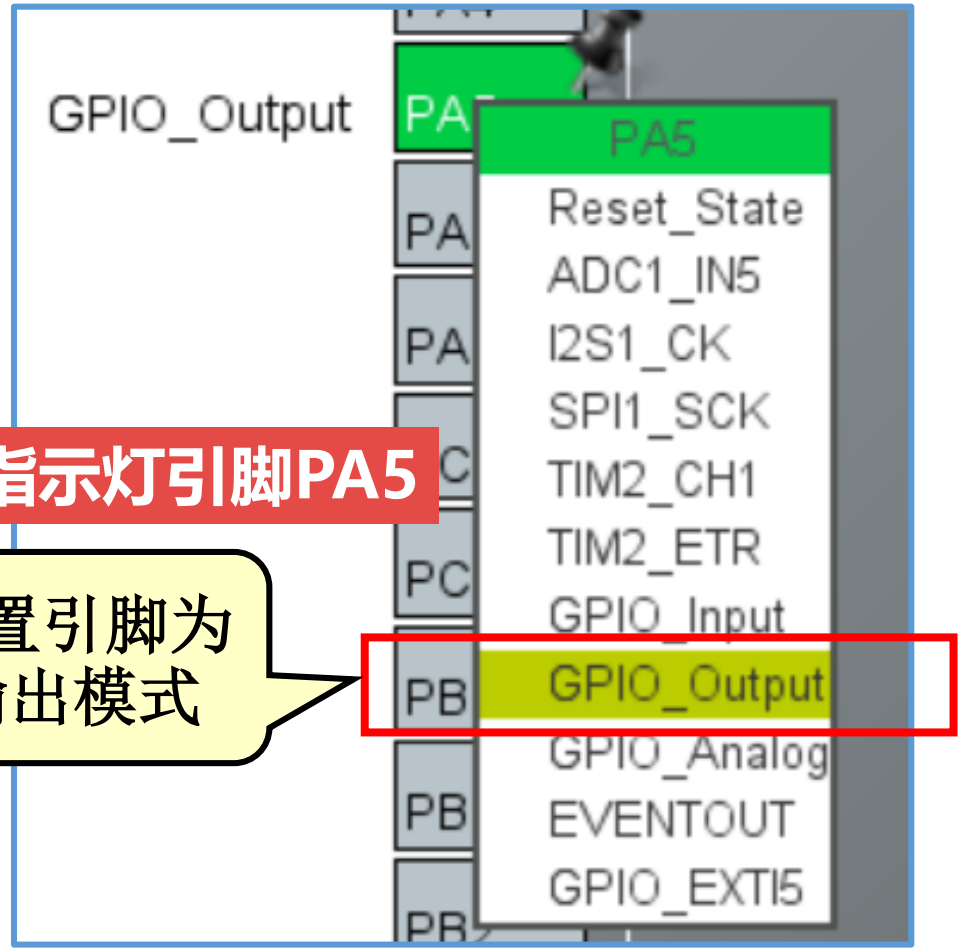




引脚分配

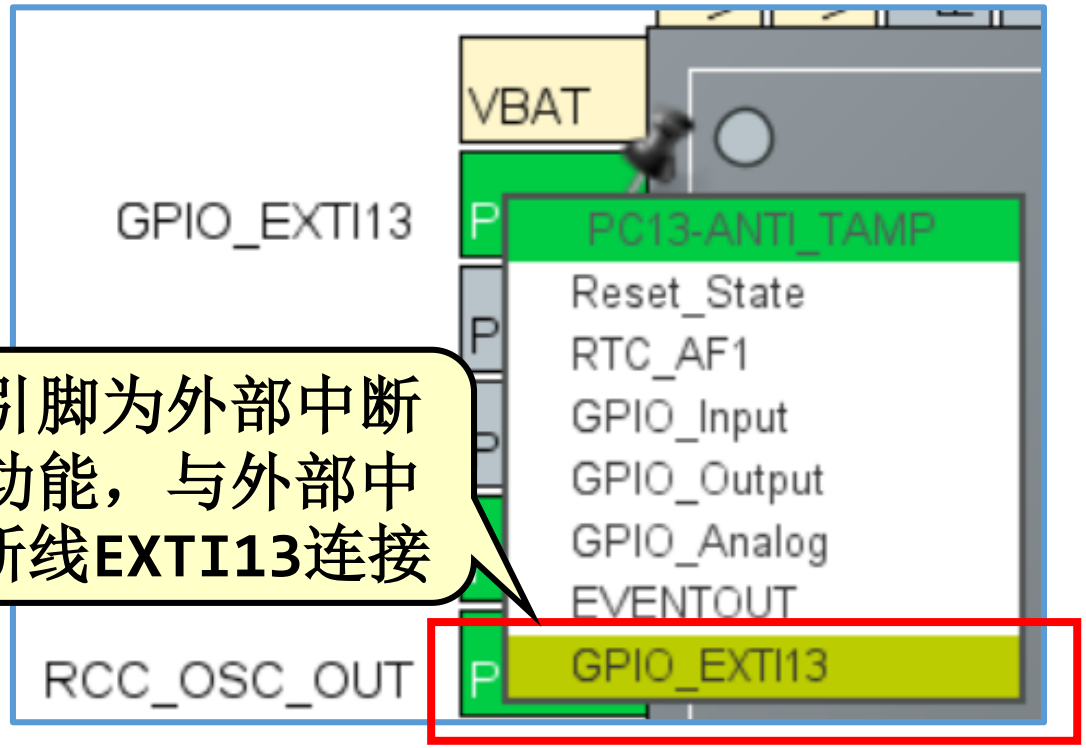
步骤二：引脚分配

设置指示灯引脚PA5



设置引脚为输出模式

设置按键引脚PC13



引脚为外部中断功能，与外部中断线EXTI13连接

外设配置

步骤三：外设配置

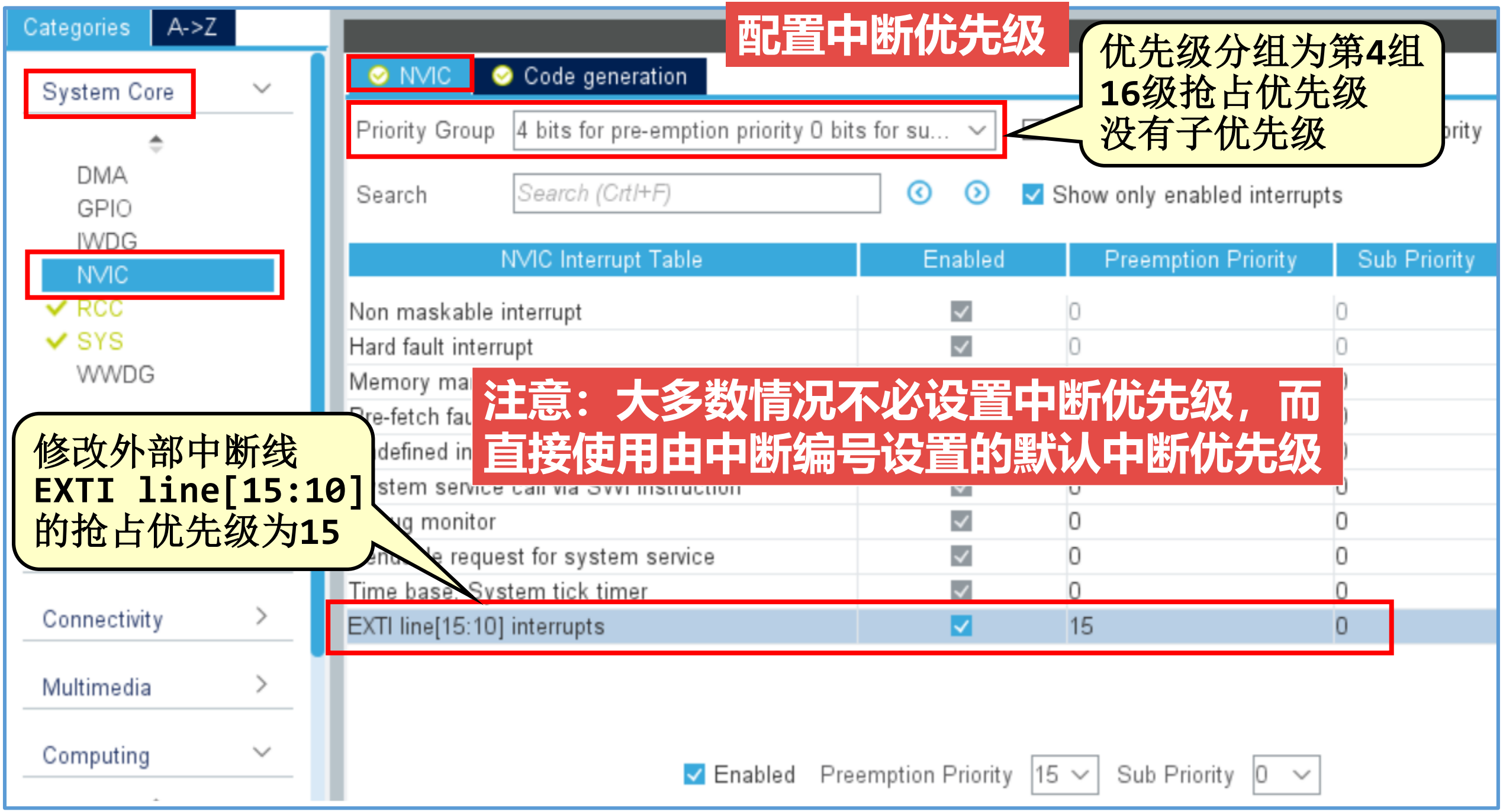
PA5 Configuration :

GPIO output level	初始电平	Low
GPIO mode	引脚模式	Output Push Pull
GPIO Pull-up/Pull-down	上/下拉电阻	No pull-up and no pull-down
Maximum output speed	引脚速度	Low
User Label	引脚名称	LD2


System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
Connectivity
Multimedia
Computing
Configuration
Group By Peripherals
引脚外部中断功能设置
GPIO
RCC
SYS
NVIC
Search Signals
Search (Crtl+F)
在出现的引脚列表中单击需要配置的引脚PC13
Pin...
Signal ...
GPIO ...
Modified
PA5
n/a
Low
No pull...
Low
LD2
PC13-...
n/a
n/a
Extern...
No pull...
n/a
B1_EXTI
PC13-ANTI_TAMP Configuration :
触发方式：下降沿触发
GPIO mode
External Interrupt Mode with Falling edge trigger det...
上/下拉电阻
GPIO Pull...
No pull-up and no pull-down
引脚名称
User Label
B1_EXTI

外设配置

使能对应的外部中断线


Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Connectivity
Multimedia
Computing
配置中断优先级
NVIC
Code generation
Priority Group
4 bits for pre-emption priority 0 bits for su...
优先级分组为第4组
16级抢占优先级
没有子优先级
Search
Search (Crtl+F)
Show only enabled interrupts
NVIC Interrupt Table
Enabled
Preemption Priority
Sub Priority
Non maskable interrupt
Hard fault interrupt
Memory ma
Pre-fetch fau
defined in
stem service call via SWI instruction
ug monitor
e request for system service
Time base: System tick timer
EXTI line[15:10] interrupts
注意：大多数情况不必设置中断优先级，而直接使用由中断编号设置的默认中断优先级
修改外部中断线
EXTI line[15:10]
的抢占优先级为15
Enabled
Preemption Priority
15
Sub Priority
0

步骤六：程序编写

```
1.  /* USER CODE BEGIN 4 */
2.  /*
3.   * @brief  EXTI line detection callbacks.
4.   * @param  GPIO_Pin: Specifies the pins connected EXTI line,
5.   * @retval None
6.   */
7.  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
8.  {
9.    if( GPIO_Pin == B1_EXTI_Pin )                    // 判断外部中断源
10.     {
11.       HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);  // 翻转LD2状态
12.     }
13. }
14.   /* USER CODE END 4 */
```

外部中断回调函数

注意：本任务只设置一个外部中断引脚，因此不需要判断多个引脚。如果系统中设置了多个外部中断，建议使用switch-case进行多分支判断。

引脚设置为外部中断的初始化函数

```
6.  static void MX_GPIO_Init(void)
7.  {
8.    GPIO_InitTypeDef GPIO_InitStruct = {0};
9.    /* GPIO Ports Clock Enable */
10.   __HAL_RCC_GPIOC_CLK_ENABLE();
15.   /*Configure GPIO pin : B1_EXTI_Pin */
16.   GPIO_InitStruct.Pin = B1_EXTI_Pin;
17.   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
18.   GPIO_InitStruct.Pull = GPIO_NOPULL;
19.   HAL_GPIO_Init(B1_EXTI_GPIO_Port, &GPIO_InitStruct);
26.   /* EXTI interrupt init*/
27.   HAL_NVIC_SetPriority(EXTI15_10_IRQn, 15, 0);
28.   HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
29. }
```

由CubeMX自动生成的引脚初始化函数

1

设置外部中断触发方式为下降沿触发

2

设置外部中断EXTI15_10的中断优先级为15，并使能这个外部中断

你的任务

1 任务一：改变中断触发方式

将本任务中外部中断的触发方式改为上升沿触发，并观察指示灯状态变化的时刻。

2 任务二：利用GPIO引脚模拟外部中断信号

使用一个GPIO引脚输出一个10Hz的方波来模拟外部中断信号，利用杜邦线将该引脚与PC13相连，设置PC13为双边沿触发，在中断中执行翻转指示灯LD2状态的操作，要求观察并记录指示灯的变化情况。

电子科技大学
University of Electronic Science and Technology of China

2 进阶任务：改变指示灯闪烁频率

进阶任务

进阶任务：改变指示灯闪烁频率

1 任务目标

掌握CubeMX软件配置外部中断的方法

2 任务内容

利用按键B1改变指示灯LD2的闪烁频率，闪烁频率设置为3档：初始状态时，LD2按照1Hz的频率闪烁；第一次按键后，LD2按照5Hz的频率闪烁；第二次按键后，LD2按照20Hz的频率闪烁，并重复上述过程。

设计思路

设计思路

定义一个全局变量Speed，该变量的取值在按键控制引脚PC13触发的外部中断服务程序中改变。主程序在while循环中不断判断Speed的值，根据取值的不同设置不同的指示灯闪烁频率。

注意：本任务的引脚分配及外设配置与基础任务相同

程序编写

步骤六：程序编写

在main.c文件中定义全局变量

```
1. /* USER CODE BEGIN PV */
2. // 指示灯闪烁频率: 0-->1Hz 1-->5Hz 2-->20Hz
3. volatile uint8_t Speed = 0;     // 避免编译器优化
4. /* USER CODE END PV */
```

在main.c文件中编写应用代码

```
10.   /* USER CODE BEGIN 3 */
11.   if( Speed == 0 )                  // 指示灯的闪烁频率为1Hz
12.   {
13.      HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
14.      HAL_Delay(1000);
15.    }
16.   else if( Speed == 1 )             // 指示灯的闪烁频率为5Hz
17.   {
18.      HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
19.      HAL_Delay(200);
20.    }
21.   else                              // 指示灯的闪烁频率为20Hz
22.   {
23.      HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
24.      HAL_Delay(50);
25.   }
26.}
27.   /* USER CODE END 3 */
```

代码解析

根据Speed的不同取值，设置不同的指示灯闪烁频率。

在main.c文件中编写外部中断的回调函数

```
1. /* USER CODE BEGIN 4 */
2. /*
3.  * @brief  EXTI line detection callbacks.
4.  * @param  GPIO_Pin: Specifies the pins connected EXTI line
5.  * @retval None
6.  */
7. void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
8. {
9.   if(GPIO_Pin == B1_EXTI_Pin)    // 判断外部中断源
10.   {
11.     Speed++;                     // 修改Speed变量的值
12.     if( Speed == 3)              // 限制Speed变量的取值范围: 0~2
13.     {
14.       Speed = 0;
15.     }
16.   }
17.}
18./* USER CODE END 4 */
```

本章结束