嵌入式系统原理及实验

顾　震

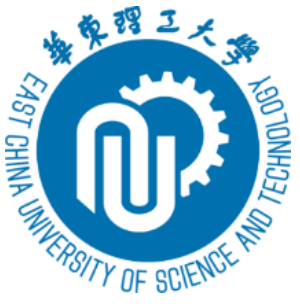
信息科学与工程学院自动化系

华东理工大学

Email: guzhen@ecust.edu.cn

课程大纲

1. 嵌入式系统导论
2. Cortex-M3微处理器
3. STM32最小系统及开发环境
4. 嵌入式C语言
5. 通用输入输出GPIO模块
6. 中断
7. 定时器原理与应用
8. USART通信原理及实现
9. DMA控制器
10. SPI与I2C通信原理及实现
11. 模数转换原理及实现
12. 人工智能辅助的嵌入式项目开发
13. 嵌入式应用前沿



6. 中断

本章知识与能力要求

- 理解和掌握中断的概念和中断处理过程；
- 理解和掌握STM32F103微控制器的中断类型、优先级概念和中断向量表；
- 了解STM32F103微控制器的NVIC中断结构和特点；
- 掌握STM32F103微控制器EXTI的内部结构、工作原理和特性；
- 掌握基于HAL库进行外部中断的开发过程。

6.1 中断的相关概念

6.1.1 什么是中断？

6.1.2 为什么使用中断？

6.1.3 中断处理流程

6.1.1 什么是中断和异常?

嵌入式中经常出现的中断

- 按钮按下
- 接收到数据
- 测量完成
- 时间到了

CPU在处理当前任务时，因为内部或外部事件的发生，暂停当前任务的执行，转而执行与事件相关的程序，处理完成后再放回原任务的过程。

6.1.1 什么是中断和异常?

- 在计算机系统中，**中断**和**异常**是两种重要的机制，它们本质上是**改变处理器执行指令的顺序**。

中断特点：

- 中断的主要作用是允许**处理器响应外部事件**，如用户输入或定时器信号。
- 当中断发生时，CPU保存当前执行状态，并跳转到处理该中断的特定代码执行。
- 处理完毕后，CPU会恢复之前的执行状态，继续执行被中断的程序。
- 中断可以被屏蔽或忽略。

异常特点：

- 异常由**内部**程序错误或CPU状态引起，如非法指令、地址越界或系统调用。
- 异常不可被屏蔽或忽略，通常需要立即处理。

6.1.2 为什么使用中断?

- **中断在嵌入式系统占有极其重要的地位，中断机制使得系统能更有效更合理地发挥效能和提高效率。**

CPU的速度快 等待 外设速度慢

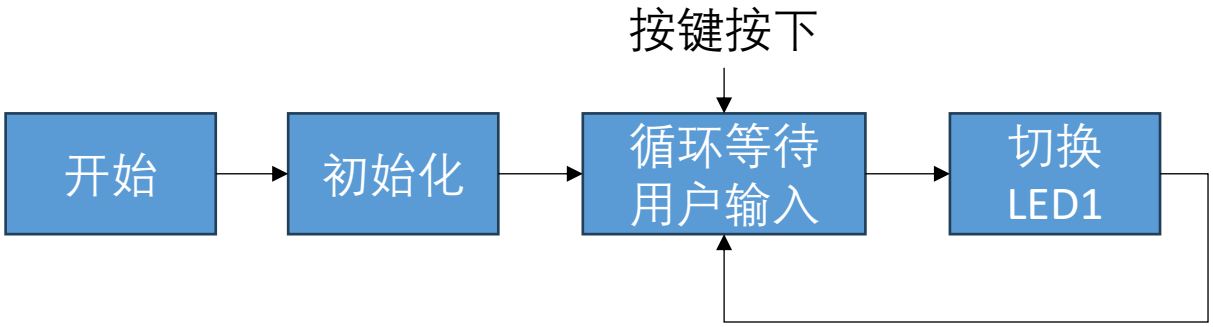
降低CPU效率

\+ 中断 →

- 提高CPU效率
- 实时事件的及时处理（实时处理）
- 突发故障及时处理（异常处理）

6.1.2 为什么使用中断?

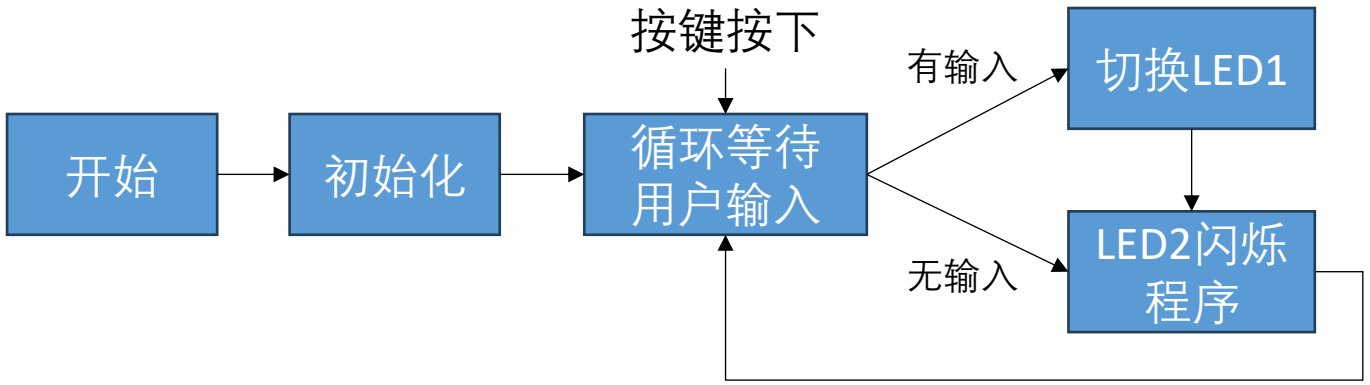
案例：如何设计一个等待用户按键按下，每次按下切换小灯LED1状态的程序?



如果在等待的同时，控制小灯LED2进行闪烁呢?

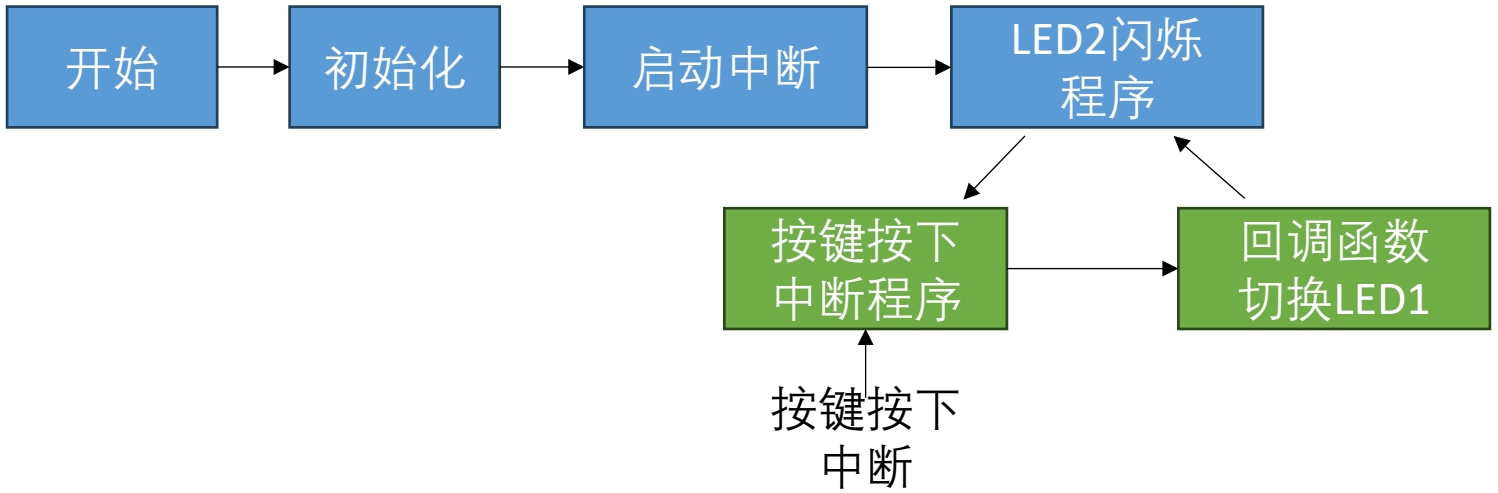
6.1.2 为什么使用中断?

方案1



如果LED2闪烁程序持续时间为1s?

方案2



6.1.2 为什么使用中断?

中断的使用意义

实时控制: 中断允许系统在确定的时间内对特定事件作出及时响应，这对于实时控制系统非常关键。

故障处理: 中断机制允许系统检测到故障并迅速作出响应。

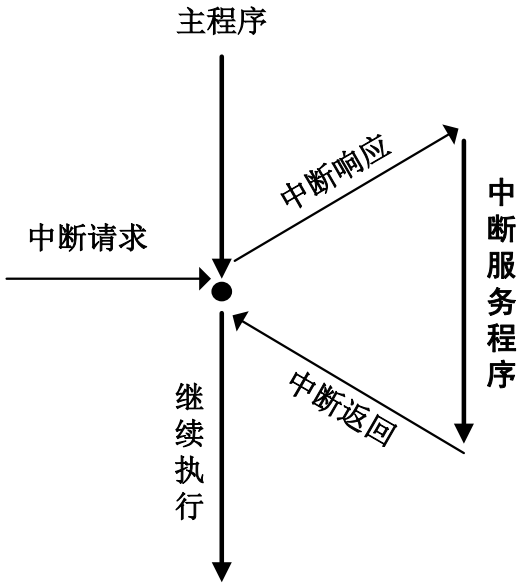
数据传输: 对于异步事件，如数据传输，中断可以在数据到达时通知系统，并立即处理接收到的数据。

高效处理紧急程序: 中断允许处理紧急程序而不需要等待当前任务的完成。

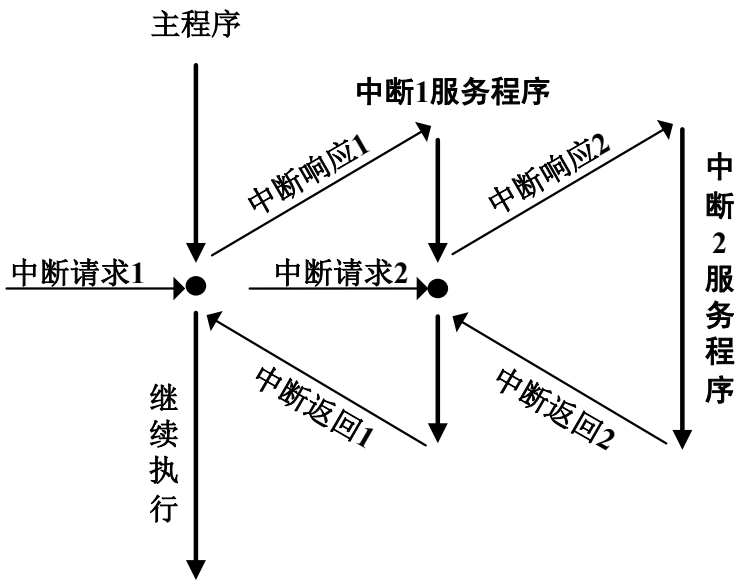
不占用CPU资源: 中断允许系统在等待事件发生的同时执行其他任务，而不会一直占用CPU资源。

6.1.3 中断处理流程

- 中断处理流程：中断请求、中断响应、中断服务和中断返回



单重中断的中断处理流程



多重中断的中断处理流程

6.1.3 中断处理流程

中断请求

中断请求是中断源向CPU发出中断请求信号，此时中断控制系统的**中断请求寄存器**被置位，向CPU请求中断。

中断响应

CPU的中断系统判断中断源的中断请求是否符合中断响应条件，如果符合条件，则暂时中断当前程序并控制程序跳转到中断服务程序

中断服务

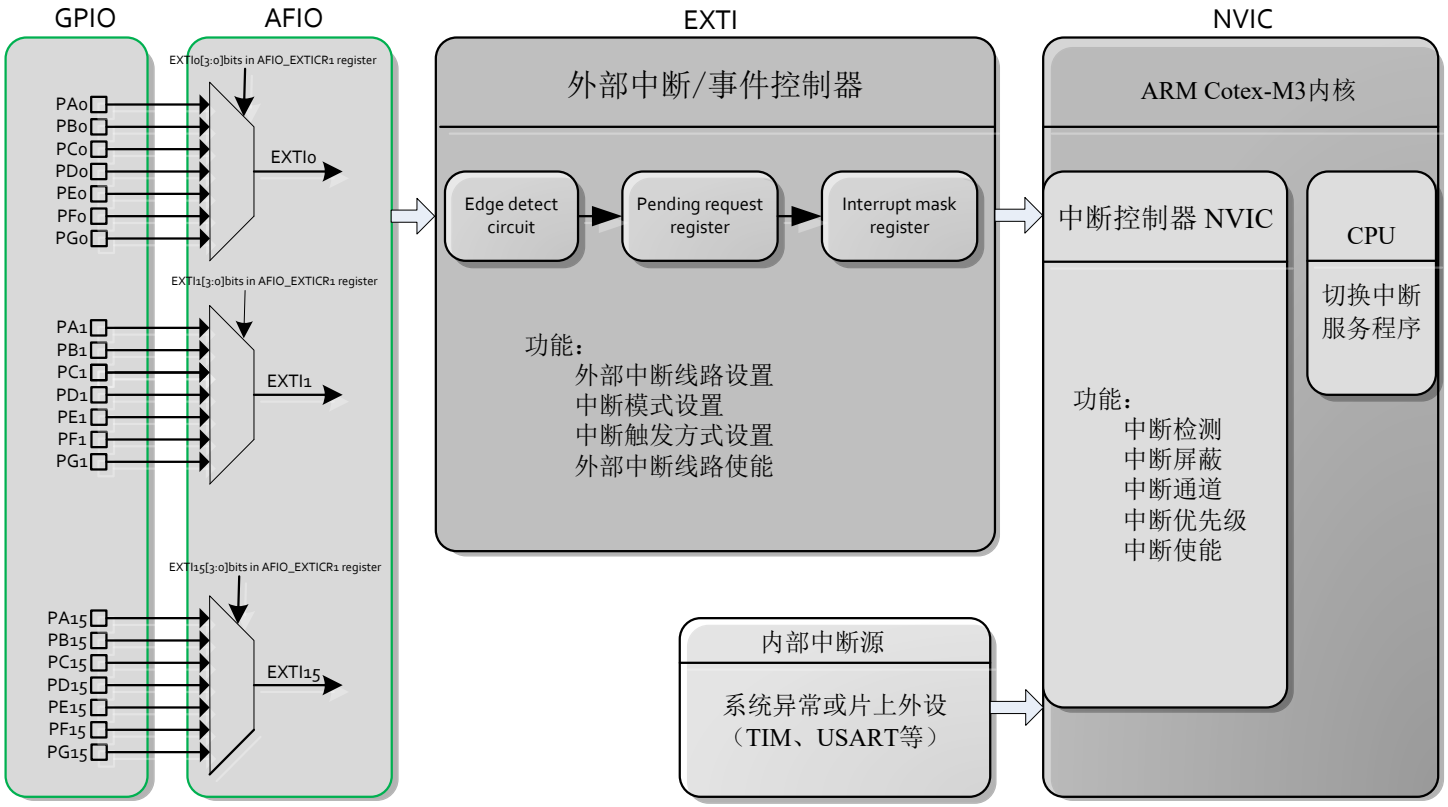
为处理中断而编写的程序称为中断服务程序，是由开发人员针对具体中断所要实现的功能进行设计和编写的，需要由开发人员来实现。

中断返回

CPU退出中断服务程序，返回到中断请求响应之前被中止的位置继续执行主程序。这部分操作同样由硬件来实现，不需要开发人员进行处理。

6.2 STM32中断和异常

STM32的内部中断处理机制



6.2 STM32中断和异常

- Cortex-M3内核集成了一个外设——**内嵌向量中断控制器**

NVIC (Nested Vectored Interrupt Controller)用于专门负责中断。

作用

- 统一管理和配置中断
- 通过优先级来控制中断的嵌套和调度。

在NVIC中，优先级的数值越小，则优先级越高。

NVIC具有以下特性

01 可嵌套中断支持，即高优先级的中断可以打断低优先级的中断。

02 向量中断支持，缩短中断延迟时间。

03 动态优先级调整支持。软件可以在运行期间更改中断的优先级。

04 引入新特长新技术，中断延迟大大缩短

05 中断可屏蔽

6.2 STM32中断和异常

6.2.1 STM32中断和异常向量表

6.2.2 STM32中断优先级

6.2.3 STM32中断服务程序

6.2.1 STM32中断和异常向量表

ARM公司设计的Cortex-M3内核可支持256个中断：

- **15个内核中断**
- **240个外部中断**

具有256级的可编程中断优先级设置，即除了3个固定的高优先级（Reset、NMI、硬件失效），其他中断和异常的优先级是可以由用户进行设置的

使用Cortex-M3内核的芯片制造商对中断进行精简

- STM32F10x系列产品有84个中断通道，包括16个内核中断和68个可屏蔽中断；
- STM32F103系列芯片只有60个可屏蔽中断
- STM32F107系列有68个可屏蔽中断。

6.2.1 STM32中断和异常向量表

中断向量表

当发生了异常或中断，内核要想响应这些异常或中断，就需要知道这些异常或中断的服务程序的**入口地址**，再由入口地址找到相应的**中断服务程序**，由中断入口地址组成的表称作中断向量表。

入口地址一般存放在程序存储器（ROM），默认情况下，Cortex-M3内核的中断向量表从零地址处开始，且每个向量占用4个字节。

向量编号	向量入口地址	说明
-	0x00000000	MSP的初始值
1	0x00000004	复位向量（PC初始值）
2	0x00000008	NMI异常服务程序的入口地址
3	0x0000000C	硬Fault异常服务程序的入口地址
…	…	其它中断服务程序的入口地址

6.2.2 STM32中断优先级

- STM32使用Cortex-M3的8位优先级寄存器中的**4位来配置中断优先级**，即STM32中的NVIC只支持16级中断优先级的管理。

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);
void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
void NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t Offset);
void NVIC_SystemLPConfig(uint8_t LowPowerMode, FunctionalState NewState);
void SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource);
```

6.2.2 STM32中断优先级

抢占优先级

- 抢占优先级决定了**一个中断是否可以打断另一个正在执行的中断**。
- 具有较高抢占优先级的中断可以打断具有较低抢占优先级的中断。
- 这种机制允许重要的任务能够迅速响应，即使系统正在处理另一个较不重要的中断。

响应优先级

- 响应优先级**在抢占优先级相同的情况下起作用**。
- 当两个具有相同抢占优先级的中断同时发生时，系统会先处理具有较高响应优先级的中断。

6.2.2 STM32中断优先级

- 高抢占优先级的中断可以打断低抢占优先级的中断服务，构成中断嵌套；

- 中断优先级的数值越小，优先级级别越高；

中断优先级判断原则

- 抢占优先级的优先级总是高于响应优先级；

中断优先级判断：先判断抢占优先级的大小，如果抢占优先级相同，则比较响应优先级的大小，若抢占优先级和响应优先级均相同，则根据中断向量表中的顺序来决定；

Reset，NMI（不可屏蔽中断），Hard Fault的优先级为负，且不可修改，高于普通的中断优先级。

6.2.2 STM32中断优先级

中断分组管理函数

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)

用于设置中断的优先级分组，此函数只有一个参数NVIC_PriorityGroup，其取值共有5组，每组的抢占优先级和响应优先级所占位数均不同，取值范围不同

NVIC_PriorityGroup	抢占优先级取值范围	响应优先级取值范围	描述
NVIC_PriorityGroup_0	0 (0位)	0 ~ 15 (4位)	抢占优先级占0位，响应优先级占4位
NVIC_PriorityGroup_1	0~1 (1位)	0~7 (3位)	抢占优先级占1位，响应优先级占3位
NVIC_PriorityGroup_2	0~3 (2位)	0~3 (2位)	抢占优先级占2位，响应优先级占2位
NVIC_PriorityGroup_3	0~7 (3位)	0~1 (1位)	抢占优先级占3位，响应优先级占1位
NVIC_PriorityGroup_4	0 ~ 15 (4位)	0 (0位)	抢占优先级占4位，响应优先级占0位

6.2.2 STM32中断优先级

结构体

中断初始化函数: void NVIC_Init(NVIC_InitTypeDef * NVIC_InitStruct)

作用：设置抢占优先级和响应优先级

指向NVIC_InitTypeDef结构体的指针

```
typedef struct
{
  uint8_t  NVIC_IRQChannel;//配置中断源，IRQ通道
  uint8_t  NVIC_IRQChannelPreemptionPriority; //配置抢占优先级
  uint8_t  NVIC_IRQChannelSubPriority; //配置响应优先级
  FunctionalState  NVIC_IRQChannelCmd; //使能中断通道
} NVIC_InitTypeDef;
```

6.2.3 STM32中断服务程序

STM32将中断服务程序统一放在标准外设库stm32f10x_it.c文件中，其中的每个中断服务函数都只有函数名，函数体都是空的，需要**用户自己编写相应的函数体**，但中断服务程序的函数名不能更改。

stm32f10x_it.c

```
24  /* Includes ------------------------------------------------------------------*/
25  #include "stm32f10x_it.h"
26
27  /** @addtogroup STM32F10x_StdPeriph_Template
28    * @{
29    */
30
31  /* Private typedef -----------------------------------------------------------*/
32  /* Private define ------------------------------------------------------------*/
33  /* Private macro -------------------------------------------------------------*/
34  /* Private variables ---------------------------------------------------------*/
35  /* Private function prototypes -----------------------------------------------*/
36  /* Private functions ---------------------------------------------------------*/
37
38  /******************************************************************************/
39  /*            Cortex-M3 Processor Exceptions Handlers                         */
40  /******************************************************************************/
41
42  /**
43    * @brief  This function handles NMI exception.
44    * @param  None
45    * @retval None
46    */
47  void NMI_Handler(void)
48  {
49  }
50
51  /**
52    * @brief  This function handles Hard Fault exception.
53    * @param  None
54    * @retval None
55    */
56  void HardFault_Handler(void)
57  {
58    /* Go to infinite loop when Hard Fault exception occurs */
59    while (1)
60    {
61    }
62  }
63
64  /**
65    * @brief  This function handles Memory Manage exception.
66    * @param  None
```

stm32f10x_it.c中断处理函数

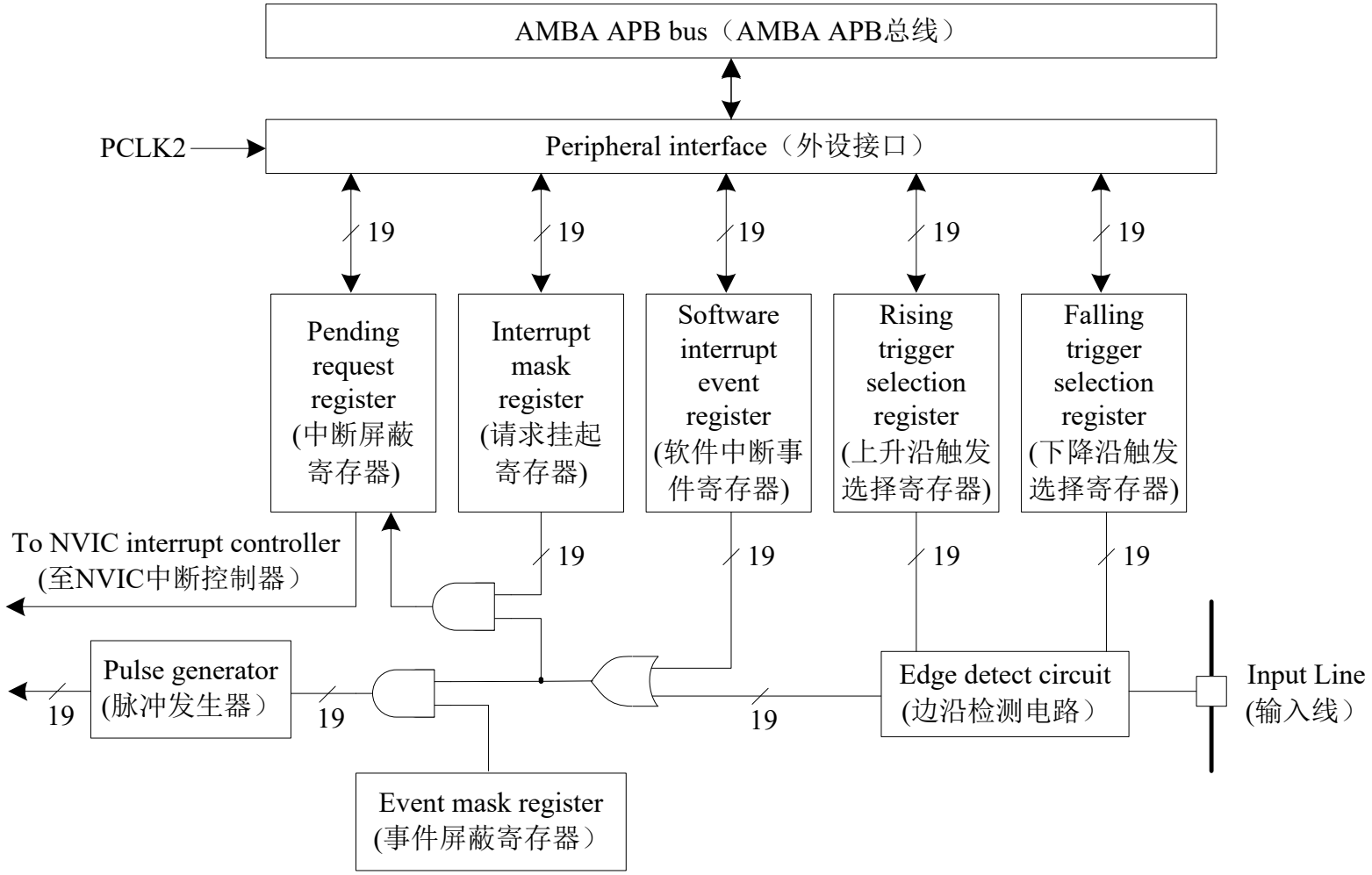
6.3 STM32外部中断EXTI

STM32芯片之外的外设的中断（I/O端口）由EXTI和NVIC共同负责，即STM32的每一个GPIO引脚都可以配置成一个**外部中断触发源**

EXTI（External interrupt/event controller，**外部中断/事件**控制器）支持19个外部中断/事件请求，每个中断/事件都有独立的触发和屏蔽设置，具有中断模式和事件模式两种设置模式。

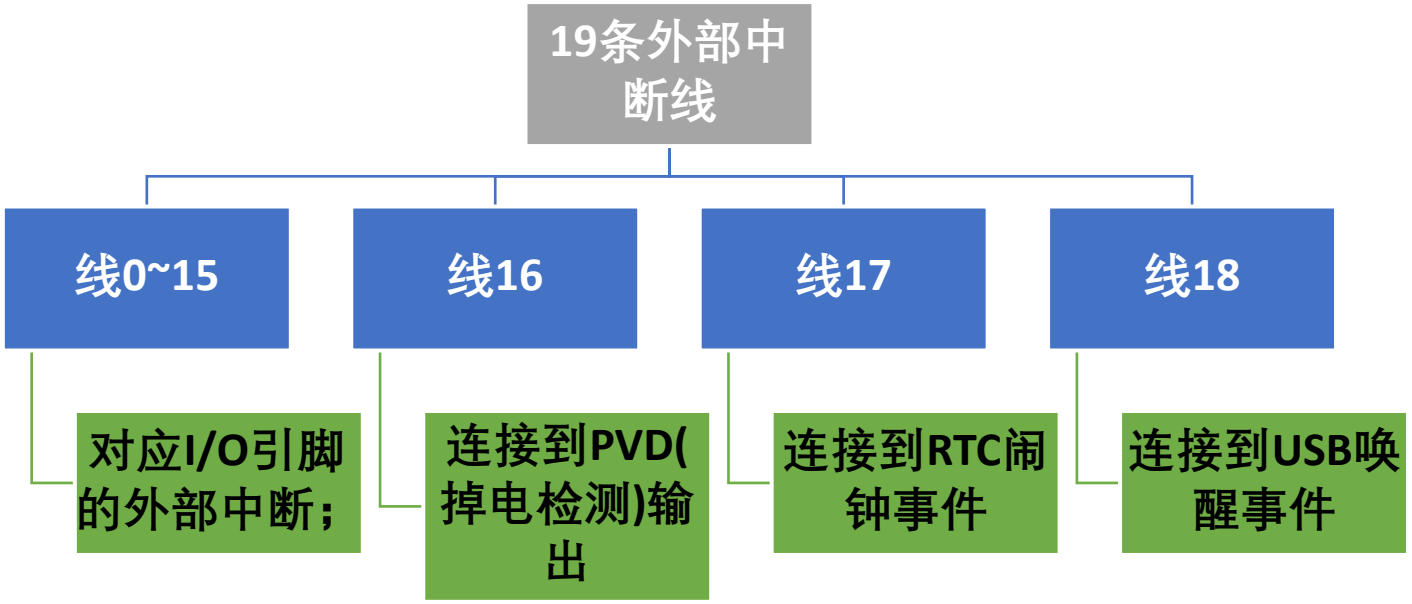
6.3 STM32外部中断EXTI

STM32 EXTI内部功能框图



6.3 STM32外部中断EXTI

STM32中，每一个GPIO都可以触发一个外部中断

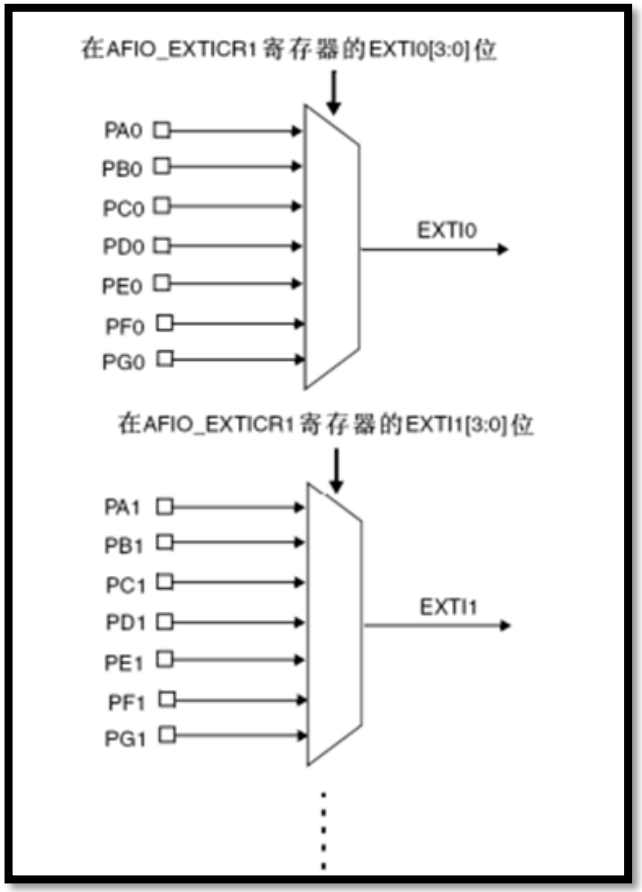


6.3 STM32外部中断EXTI

19条外部中断线如何实现对所有IO引脚的响应？

GPIO的中断是以组为单位的，同组的外部中断公用一条外部中断线。

例如：PA0、PB0、PC0、PD0、PE0、PF0、PG0这些为一组，如果使用PA0作为外部中断源，那么PB0、PC0、PD0、PE0、PF0、PG0就不能同时再作为外部中断使用了，在此情况下，只能使用类似于PB1、PC2这种末端序号不同的外部中断源。



GPIO引脚和外部中断线的映射关系图

6.3 STM32外部中断EXTI

中断服务函数的共用

中断标志	中断服务函数
EXTI0	void EXTI0_IRQHandler(void)
EXTI1	void EXTI1_IRQHandler(void)
EXTI2	void EXTI2_IRQHandler(void)
EXTI3	void EXTI3_IRQHandler(void)
EXTI4	void EXTI4_IRQHandler(void)
EXTI5-9	void EXTI9_5_IRQHandler(void)
EXTI10-15	void EXTI15_10_IRQHandler(void)

- EXTI0-EXTI4这5个外部中断有着各自独立的中断服务函数，如EXTI3的中断服务函数为 void EXTI3_IRQHandler(void);
- EXTI5-9共用一个中断服务函数 void EXTI9_5_IRQHandler(void);
- EXTI10-15共用一个中断服务函数 void EXTI15_10_IRQHandler(void);

6.4 EXTI模块的HAL库接口函数及应用

目 录

6.4.1 EXTI的HAL库接口函数

EXTI的HAL库接口函数的源码定义在stm32f1xx_hal_exti.c源文件中，其对应的头文件stm32f1xx_hal_exti.h声明了EXTI所有的库函数

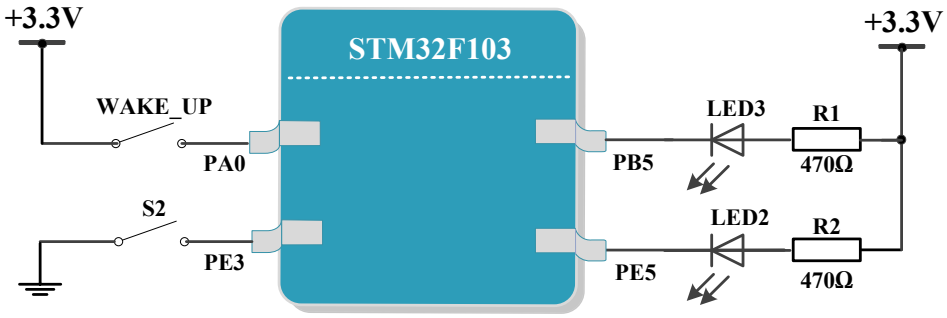
类型	函数及功能描述
配置函数	HAL_EXTI_SetConfigLine(EXTI_HandleTypeDef *hexti, EXTI_ConfigTypeDef *pExtiConfig); 功能描述：设置外部中断EXTI的外部中断线路
	HAL_EXTI_GetConfigLine(EXTI_HandleTypeDef *hexti, EXTI_ConfigTypeDef *pExtiConfig); 功能描述：获取已配置好的外部中断EXTI的外部中断线路
	HAL_StatusTypeDef HAL_EXTI_ClearConfigLine(EXTI_HandleTypeDef *hexti); 功能描述：将已设置好的外部中断线路清除
	HAL_EXTI_GetHandle(EXTI_HandleTypeDef *hexti, uint32_t ExtiLine); 功能描述：获取EXTI的句柄
引脚功能操作函数	void HAL_EXTI_IRQHandler(EXTI_HandleTypeDef *hexti); 功能描述：外部中断EXTI的中断处理函数
	uint32_t HAL_EXTI_GetPending(EXTI_HandleTypeDef *hexti, uint32_t Edge); 功能描述：获取被挂起的外部中断线/事件线
	void HAL_EXTI_ClearPending(EXTI_HandleTypeDef *hexti, uint32_t Edge); 功能描述：清除被挂起的外部中断/事件线
	void HAL_EXTI_GenerateSWI(EXTI_HandleTypeDef *hexti); 功能描述：产生一个软件中断

6.4.2 EXTI的HAL库应用实例

功能

采用基于HAL库设计方式，通过两个按键分别控制两个LED灯状态翻转。

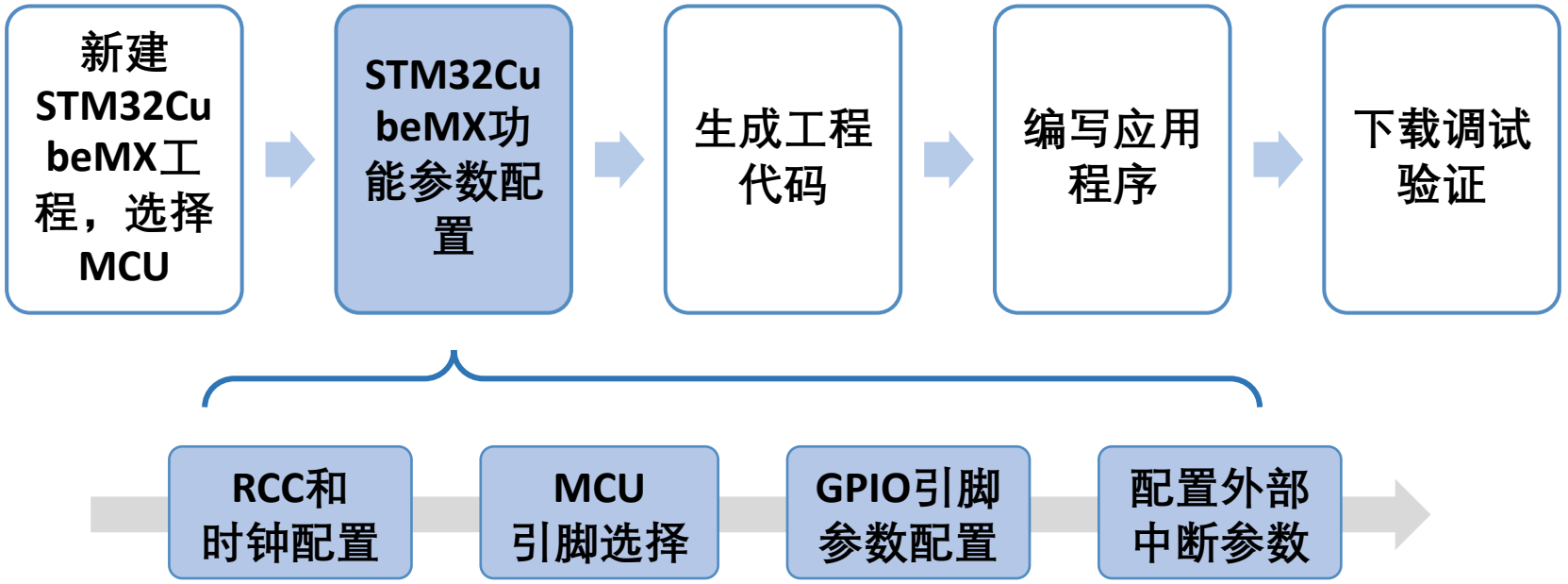
硬件设计



按键WAKE_UP连接在STM32F103的PA0引脚上，按下时PA0为高电平，按键S2连接在STM32F103的PE3引脚上。

6.4.2 EXTI的HAL库应用实例

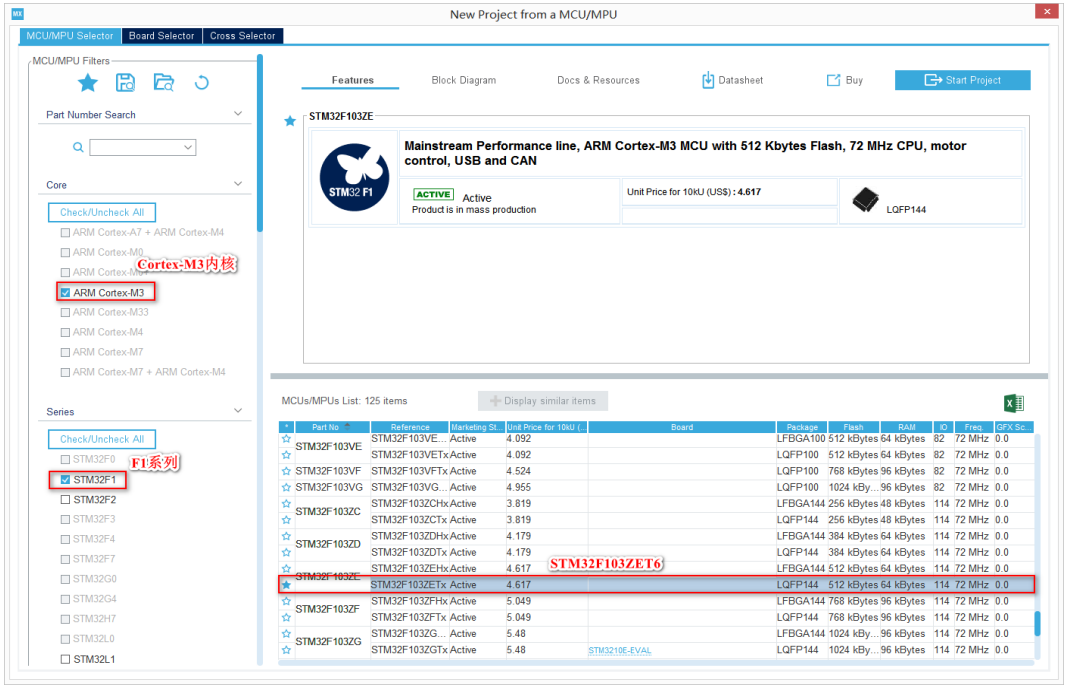
软件设计流程



6.4.2 EXTI的HAL库应用实例

软件设计——新建STM32CubeMX工程，选择MCU

在D盘或其他盘符目录下新建一个文件夹，用来存放后面建立的STM32CubeMX工程。需要注意的是：所建工程文件名最好是英文名称，且最好是英文路径。这里采用STM32F103ZET6芯片。

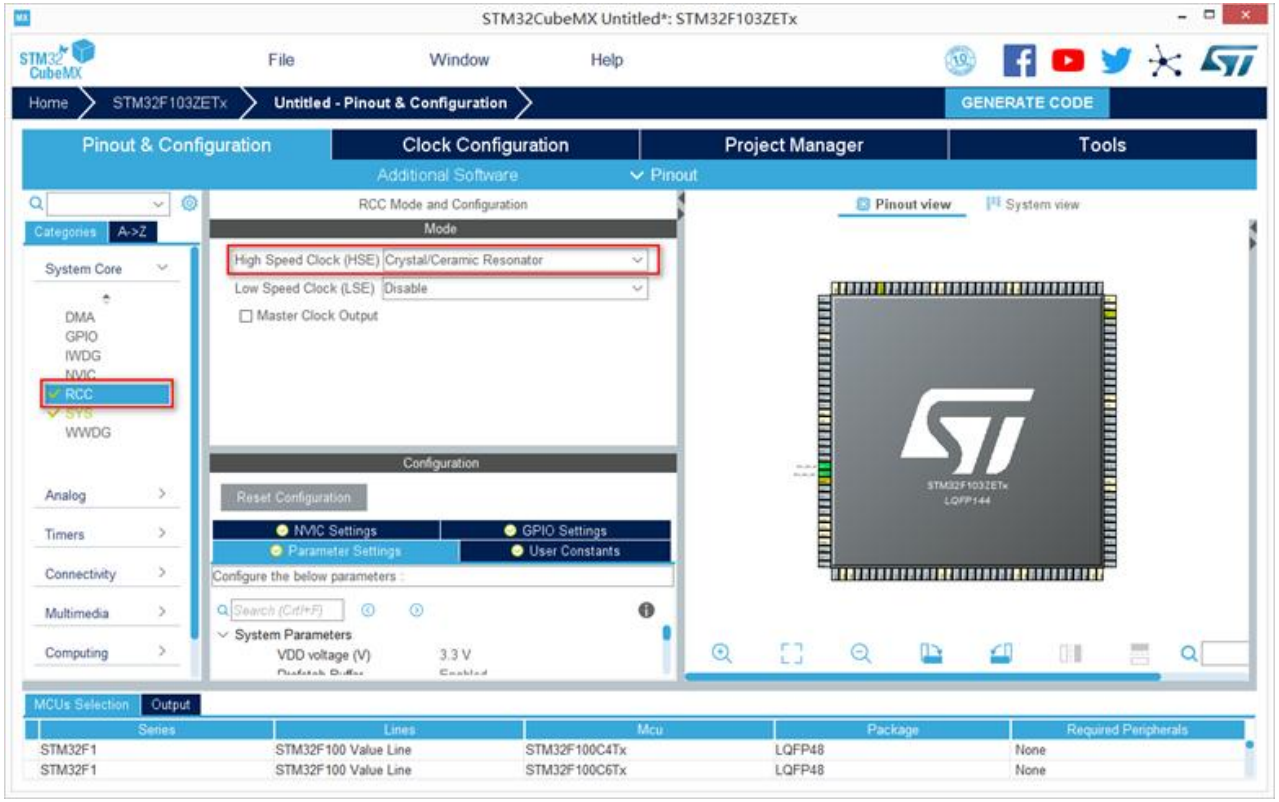


6.4.2 EXTI的HAL库应用实例

软件设计——STM32CubeMX功能参数配置

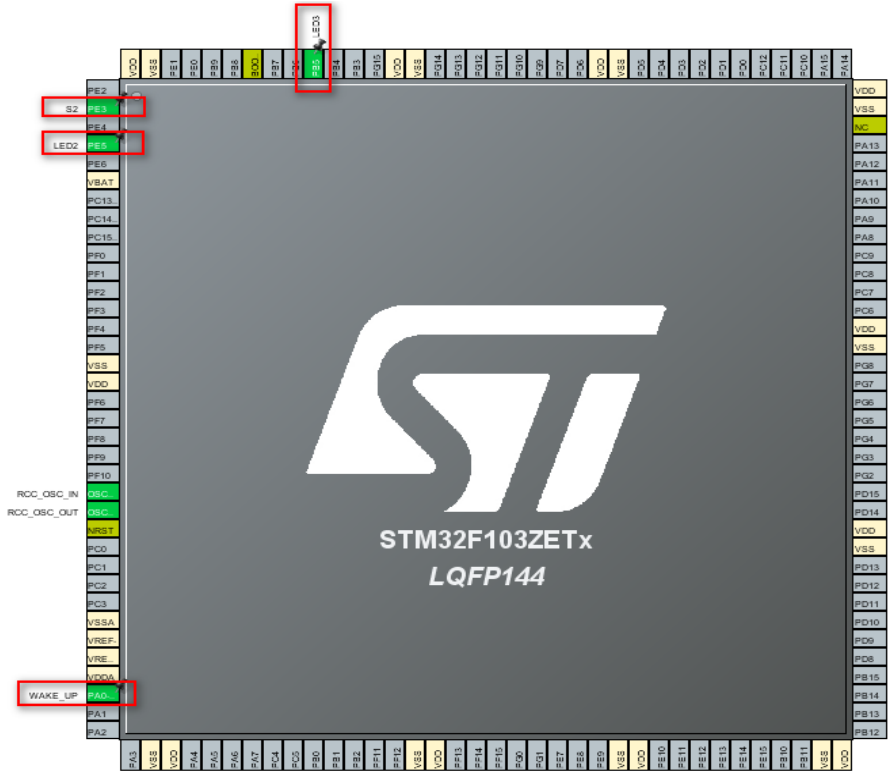
RCC配置

HSE选择“Crystal/Ceramic Resonator”（晶振/陶瓷谐振器），LSE选择“Disable”



6.4.2 EXTI的HAL库应用实例

软件设计——STM32CubeMX功能参数配置

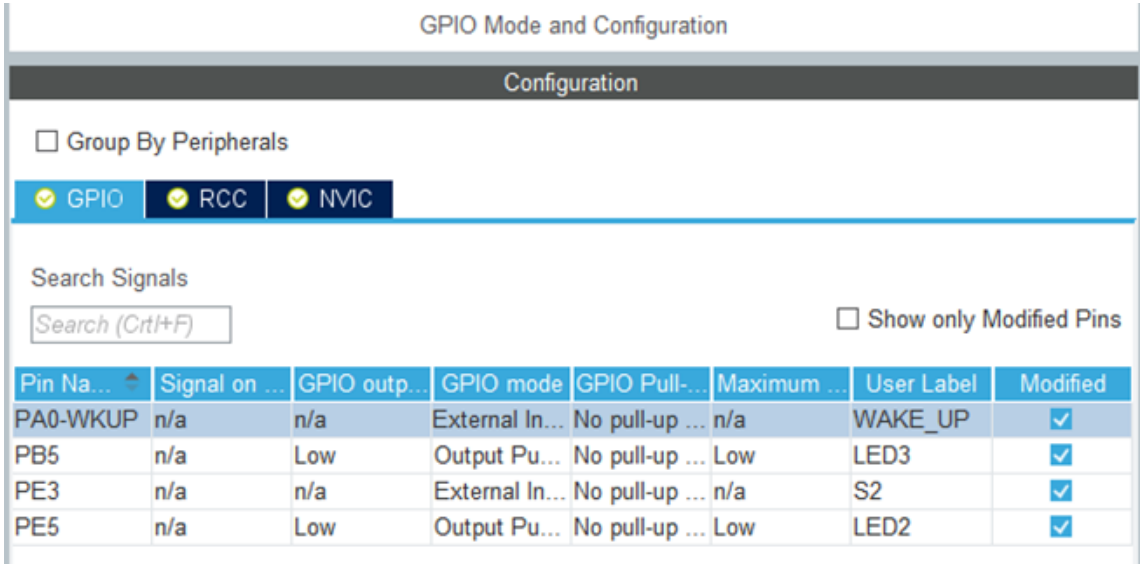


MCU引脚选择

- LED2连接在PE5引脚上，LED3连接在PB5引脚上，设置为GPIO_Output；
- 设置PA0为GPIO_EXTI0外部中断线作为WAKE_UP按键使用，用于控制LED3灯闪烁；
- 设置PE3为GPIO_EXTI3作为按键S2使用，用于控制LED2灯闪烁。

6.4.2 EXTI的HAL库应用实例

软件设计——STM32CubeMX功能参数配置



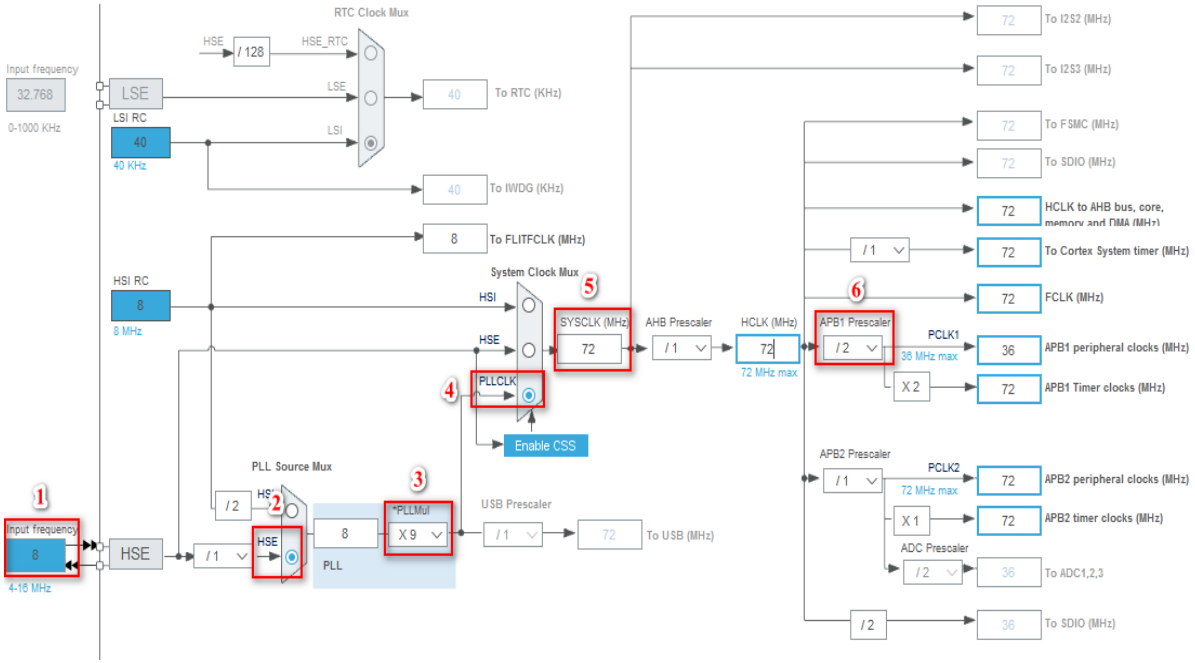
Pin Na...	Signal on ...	GPIO outp...	GPIO mode	GPIO Pull-...	Maximum ...	User Label	Modified
PA0-WKUP	n/a	n/a	External In...	No pull-up ...	n/a	WAKE_UP	☑
PB5	n/a	Low	Output Pu...	No pull-up ...	Low	LED3	☑
PE3	n/a	n/a	External In...	No pull-up ...	n/a	S2	☑
PE5	n/a	Low	Output Pu...	No pull-up ...	Low	LED2	☑

GPIO引脚参数配置

- 在“GPIO”参数配置中设置PA0为上升沿触发；
- PE3的“GPIO mode”设置为下降沿触发，添加用户标签S2。
- PB5和PE5两个LED灯设置为推挽输出，分别添加用户标签为LED2和LED3。

6.4.2 EXTI的HAL库应用实例

软件设计——STM32CubeMX功能参数配置

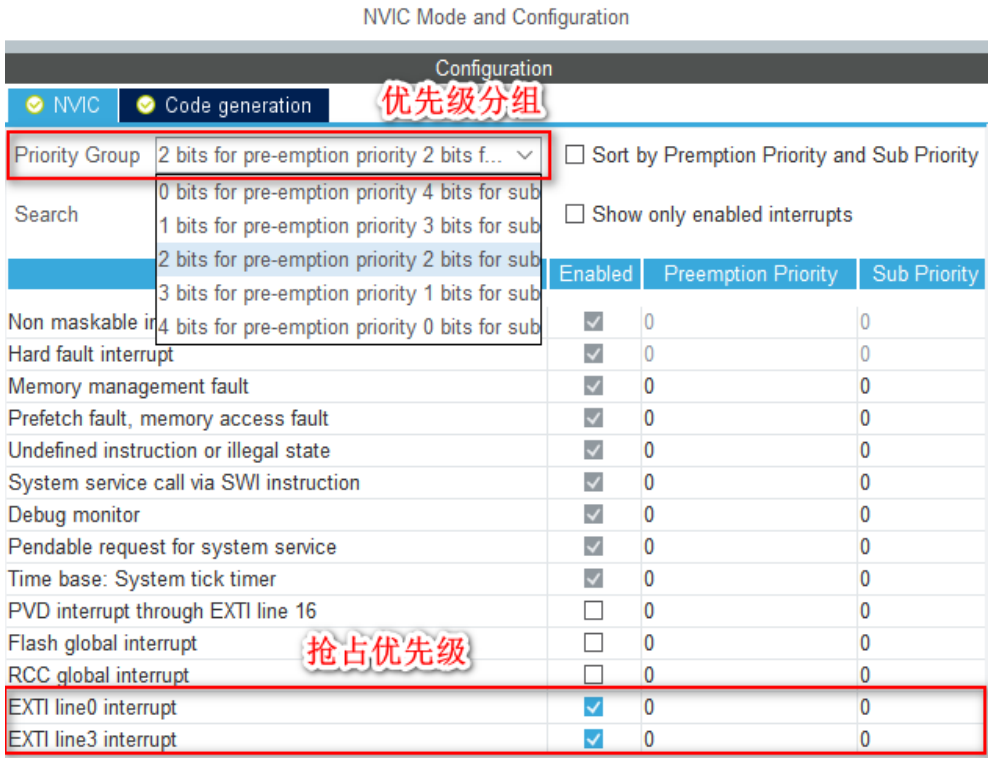


时钟配置

通过图形化方式直观对系统时钟进行配置，系统时钟采用外部高速时钟，配置STM32F103系列芯片最大时钟为72MHz，配置APB2为72MHz，配置APB1为36MHz。

6.4.2 EXTI的HAL库应用实例

软件设计——STM32CubeMX功能参数配置

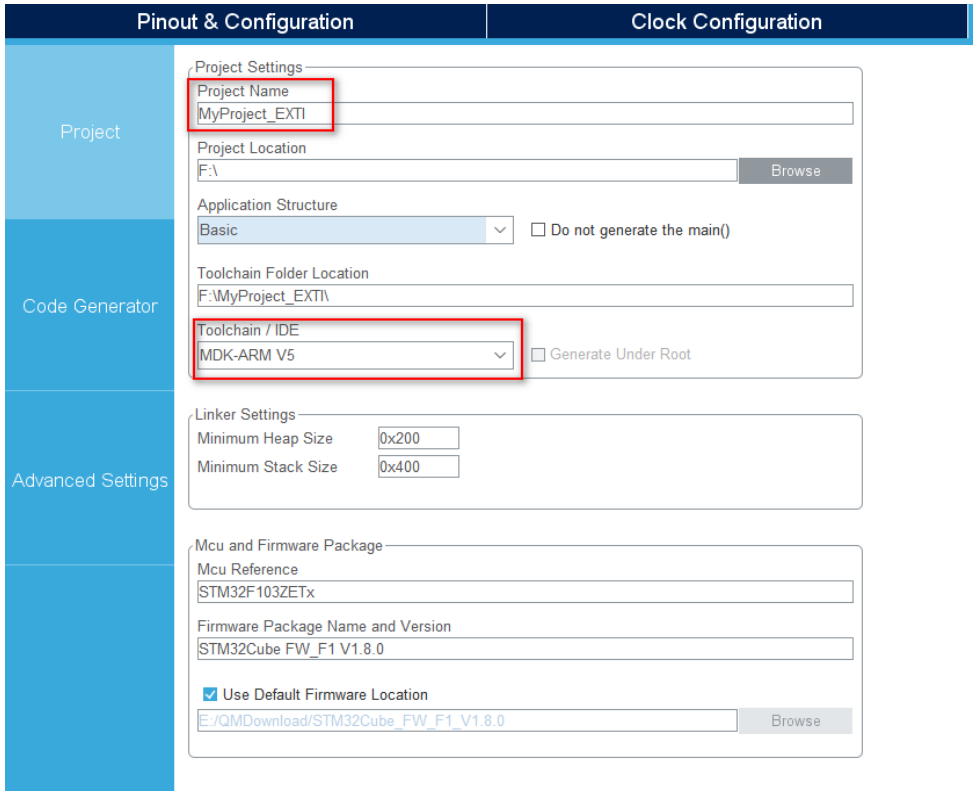


配置外部中断参数

- 在“NVIC”选项页中，设置“Priority Group”进行优先级分组；
- 勾选“EXIT_Line0 interrupt”使能PA0外部中断线；
- 类似地，勾选“EXTI_Line3 interrupt”使能PE3外部中断线。

6.4.2 EXTI的HAL库应用实例

软件设计——生成工程代码

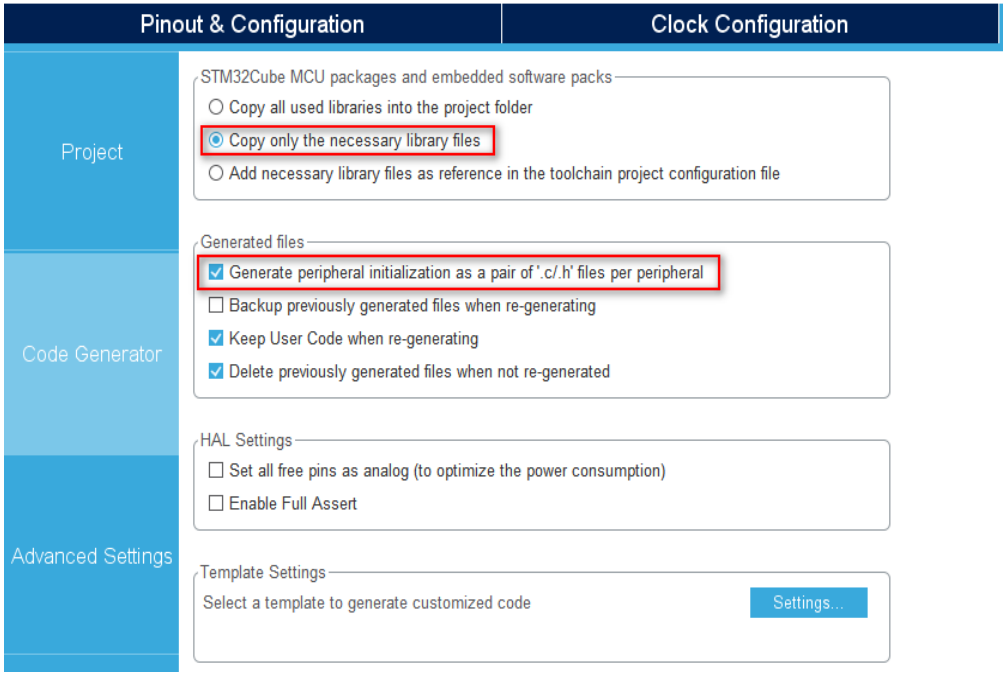


配置keil工程名称和存放位置

定义“Project Name”为“MyProject_EXTI”，“Toolchain/IDE”选择“MDK-ARM V5”；

6.4.2 EXTI的HAL库应用实例

软件设计——生成工程代码

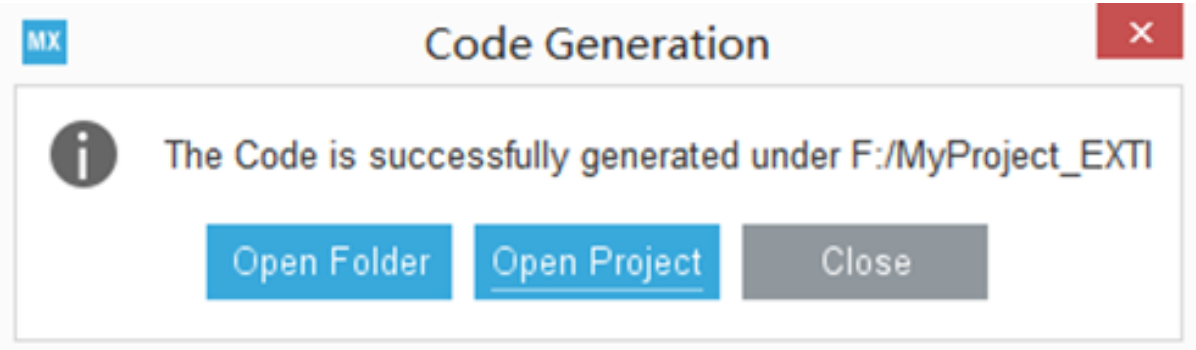


在“Code Generator”选项栏中找到“Generated files”框，勾选“Generate peripheral initialization as a pair of ’.c/.h’ files per IP”，将外设初始化的代码设置生成为独立的.c源文件和.h头文件。

6.4.2 EXTI的HAL库应用实例

软件设计——生成工程代码

通过STM32CubeMX的菜单栏中的“Generate Code”生成工程代码，生成代码后，会提示是否打开该工程窗口



6.4.2 EXTI的HAL库应用实例

软件设计——编写应用程序

- 在keil 中单击编译按钮，对生成的工程进行编译；
- 在stm32f1xx_it.c文件中可以找到所定义的中断函数；
- 外部的两个按键中断触发后，首先会调用GPIO相应引脚的外部中断处理函数。

main.c | gpio.c | stm32f1xx_it.h | **stm32f1xx_it.c** | stm32f1xx_hal_gpio.c

```
188 /* STM32F1xx Peripheral Interrupt Handlers
189 /* Add here the Interrupt Handlers for the used peripherals.
190 /* For the available peripheral interrupt handler names,
191 /* please refer to the startup file (startup_stm32f1xx.s).
192 /*****************************************************************
193
194 /**
195  * @brief This function handles EXTI line0 interrupt.
196  */
197 void EXTI0_IRQHandler(void)
198 {
199   /* USER CODE BEGIN EXTI0_IRQn 0 */
200
201   /* USER CODE END EXTI0_IRQn 0 */
202   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
203   /* USER CODE BEGIN EXTI0_IRQn 1 */
204
205   /* USER CODE END EXTI0_IRQn 1 */
206 }
207
208 /**
209  * @brief This function handles EXTI line3 interrupt.
210  */
211 void EXTI3_IRQHandler(void)
212 {
213   /* USER CODE BEGIN EXTI3_IRQn 0 */
214
215   /* USER CODE END EXTI3_IRQn 0 */
216   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3);
217   /* USER CODE BEGIN EXTI3_IRQn 1 */
218
219   /* USER CODE END EXTI3_IRQn 1 */
220 }
221
222 /* USER CODE BEGIN 1 */
223
```

外部中断线0的中断处理函数

外部中断线3的中断处理函数

6.4.2 EXTI的HAL库应用实例

软件设计——编写应用程序

```
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);
  }
}
```

HAL_GPIO_EXTI_IRQHandler()函数原型

- 比如用户按键S2连接在PE3引脚上会触发外部中断EXTI3的中断处理程序EXTI3_IRQHandler(void)，而这个函数又调用HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_3)函数，参数为GPIO_PIN_3，即EXTI3中断。单击右键跳转到这个函数，可以看到该函数的原型，

6.4.2 EXTI的HAL库应用实例

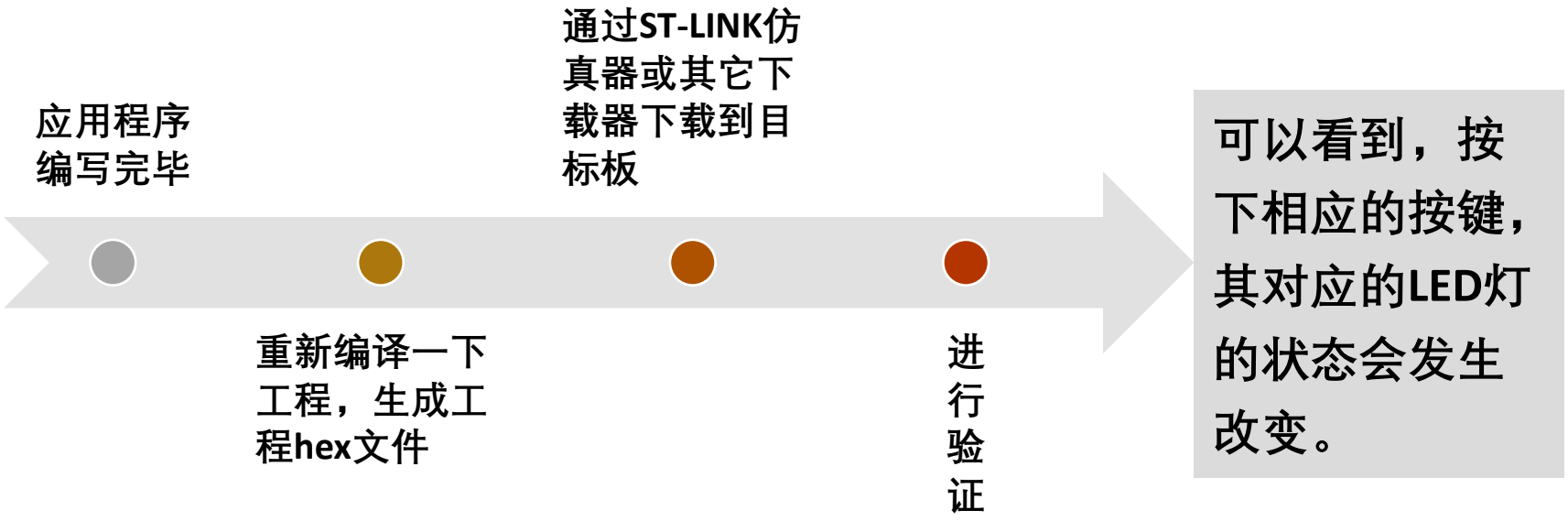
软件设计——编写应用程序

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == GPIO_PIN_3)
  {
    HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_5);
  }
 if(GPIO_Pin == GPIO_PIN_0)
 {
     HAL_GPIO_TogglePin(LED3_GPIO_Port,LED3_Pin);
 }
}
/* USER CODE END 4 */
```

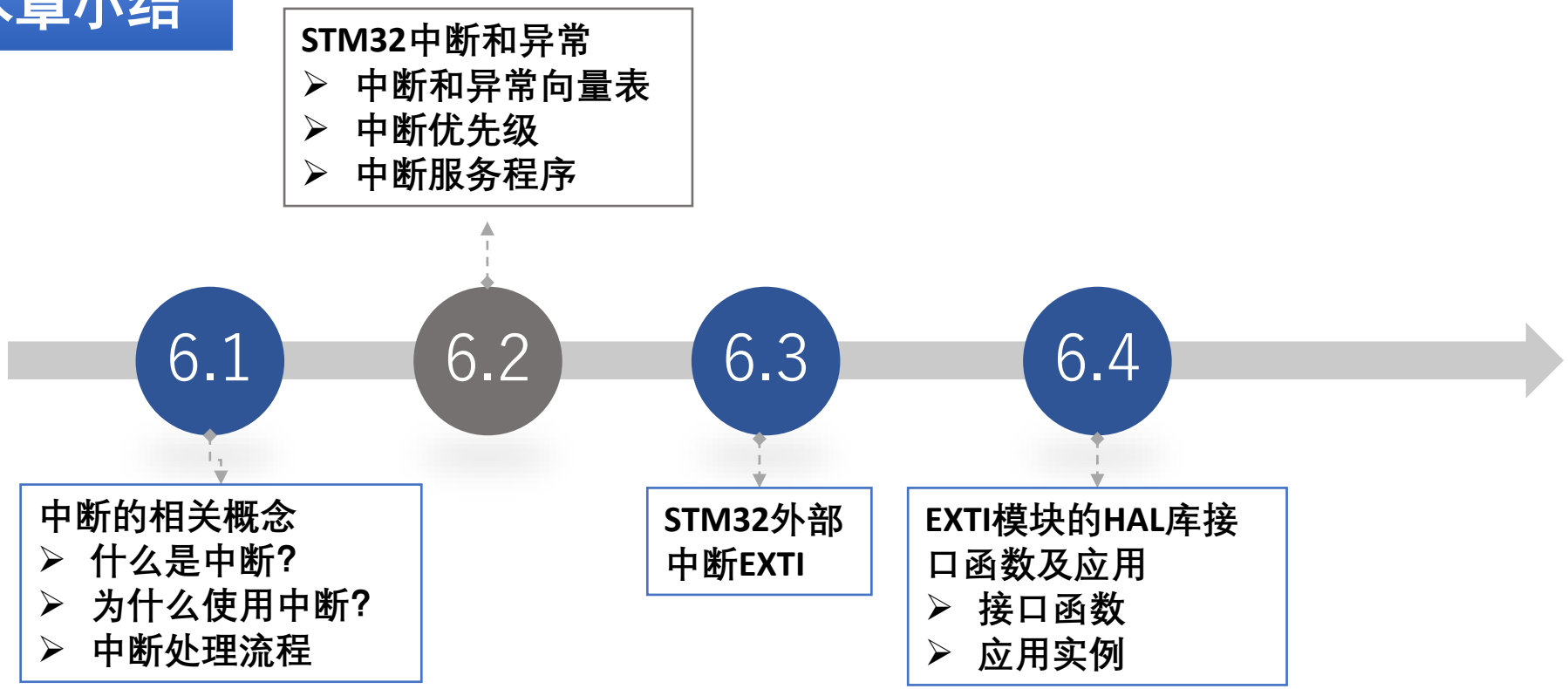
- 当中断发生时，会调用HAL_GPIO_EXTI_Callback()函数，但程序中只给出了一个虚函数，需要用户添加应用程序代码来实现相关功能。
- 在main.c文件中的/*USER CODE BEGIN 4*/和/*USER CODE END 4*/之间添加中断回调函数相应的程序代码。

6.4.2 EXTI的HAL库应用实例

软件设计——下载调试验证



本章小结


STM32中断和异常
➢ 中断和异常向量表
➢ 中断优先级
➢ 中断服务程序
6.1
6.2
6.3
6.4
中断的相关概念
➢ 什么是中断?
➢ 为什么使用中断?
➢ 中断处理流程
STM32外部中断EXTI
EXTI模块的HAL库接口函数及应用
➢ 接口函数
➢ 应用实例