电子科学与技术专业课

嵌入式系统

Embedded System

信息学院 光电子系

第4章 STM32外设进阶

- 4.1 外设学习概述
- 4.2 IO外部中断与编程
- 4.3 通用定时器原理与编程
- 4.4 。。。。。

ADC模数转换器

教学视频及慕课内容

☆ **1、ARM微控制器外设：IO的中断编程 - 上、下**（清华慕课 – 中）

—上篇19min。中断基本概念，轮询方式的缺陷，中断设置的步骤（Step1）涉及中断向量表及其中的外部中断，NVIC及其寄存器，开放中断总、子开关(NVIC和Port)的代码示例

—下篇17min。中断设置第2~4步，配置中断引脚，中断标志位作用，编写中断服务子程序ISR，ISR的被执行过程

☆ **2、IO中断编程实现按键控制LED**（教师演示视频 – 中）

—10min。展示了实现按键中断功能的STM32工程结构、头文件、源文件介绍

☆ **3、在线调试基本手段及中断程序调试**（教师演示视频 – 中）

—20min。介绍单步、断点等调试手段，各种调试窗口的用途，并对按键中断程序进行了调试

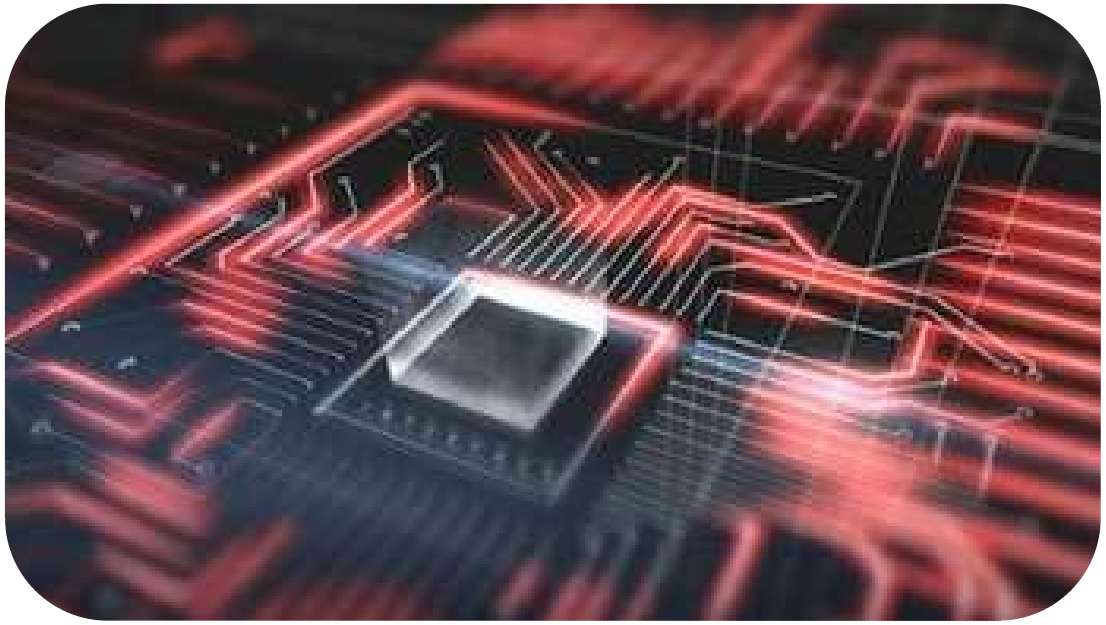
☆ **4、定时器的原理**（清华慕课 – 中）

—16min。从轮询点灯、IO中断点灯导入，引入定时器基本概念和用途，工作方式自由式(满模)和调制式，各种定时器：包括SysTick（24位）、Timer、PWM

☆ **5、脉冲宽度调制PWM的原理**（清华慕课 – 中）

—15min。占空比的概念，改变占空比达到调节电压的目的，用途驱动扬声器、电机调速等；定时器的PWM模块可以输出占空比可调信号；边沿对齐和中心对齐，后者可以避免死区出现。（忽略寄存器部分内容）

嵌入式系统(EMBEDDED SYSTEM)

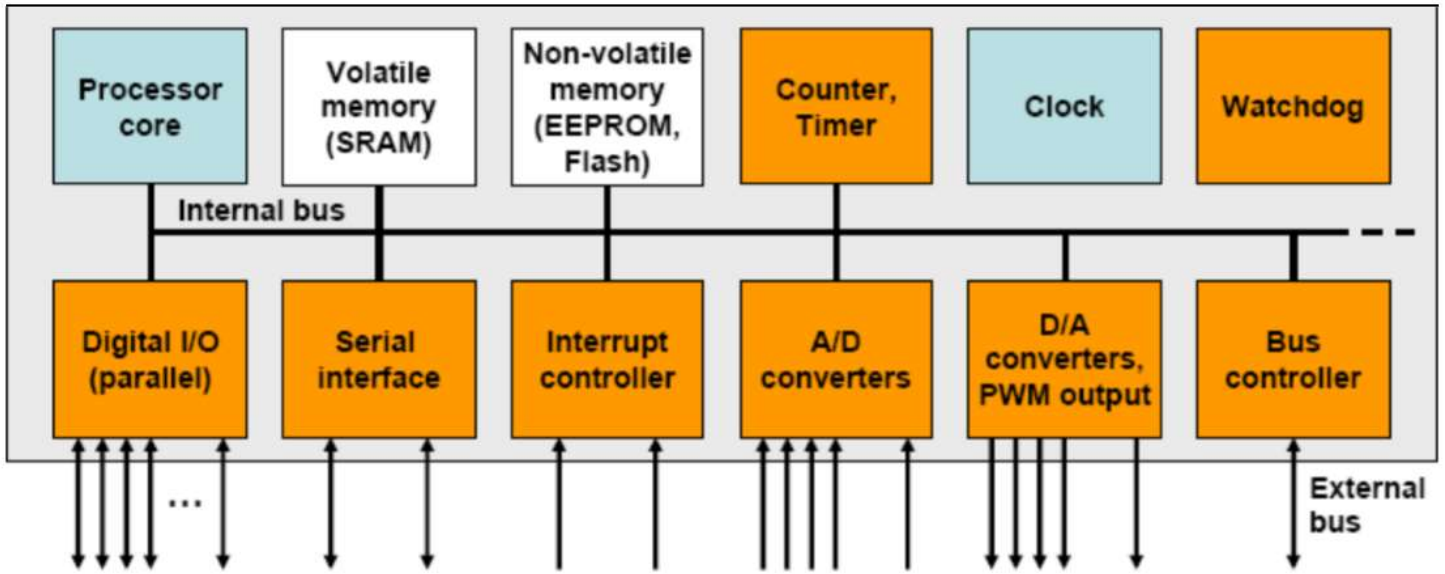


第4章 STM32外设进阶

- **4.1 外设学习概述**

MCU结构

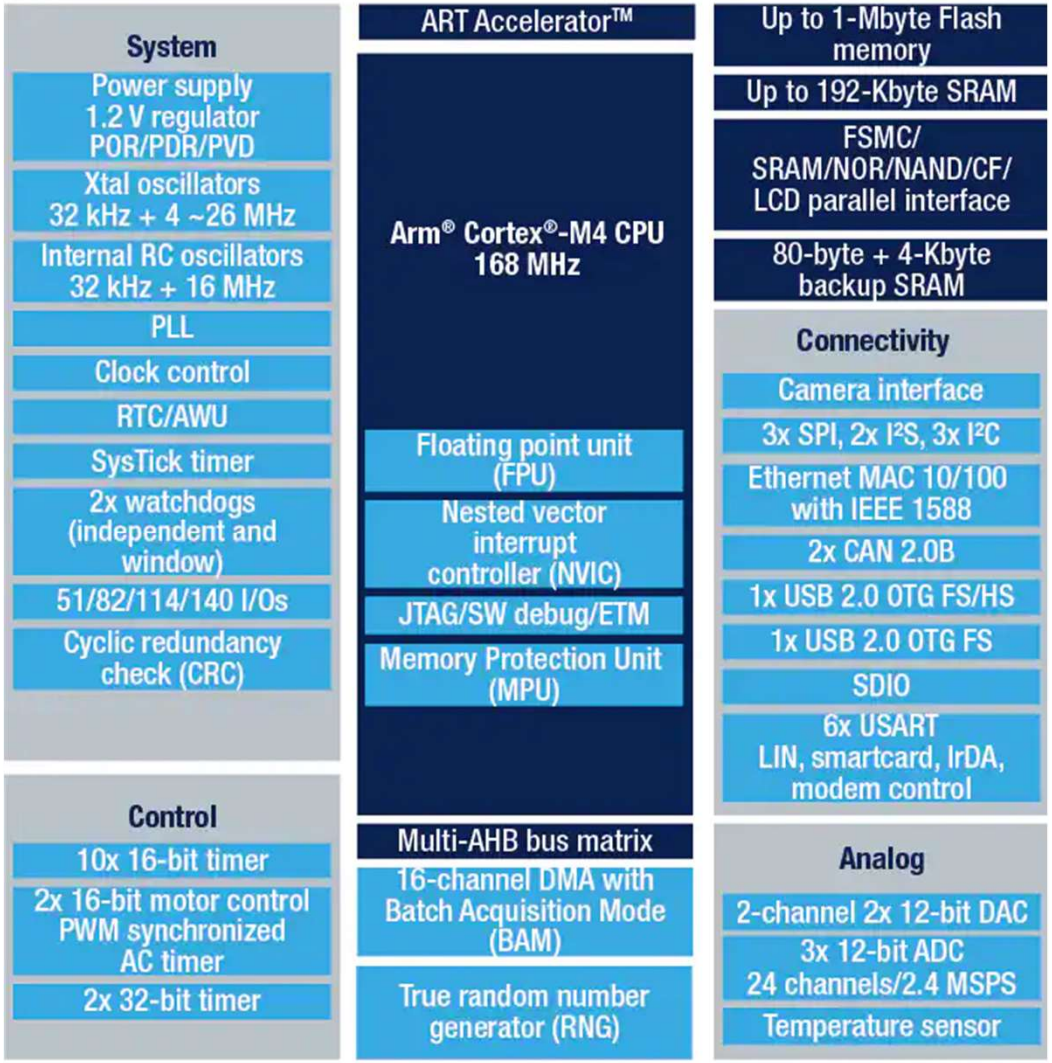
- CPU+存储器+外设



片上外部设备

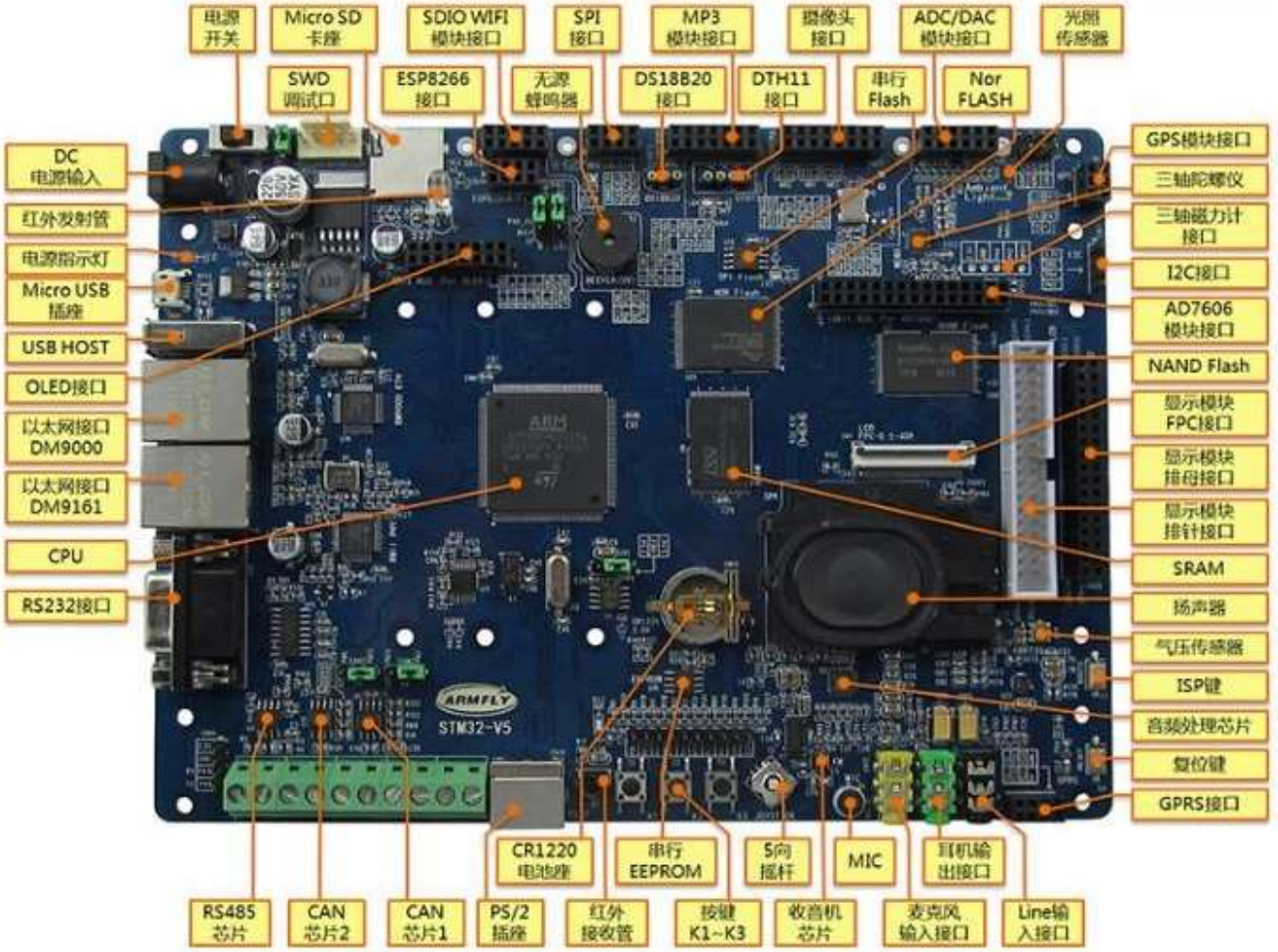
STM32F407IG 内部组成

- 丰富外设
  - GPIO，中断，定时器
  - UART，SPI，I²C
  - ……
- 学习外设
  - 原理
  - 寄存器/库函数
  - 编程

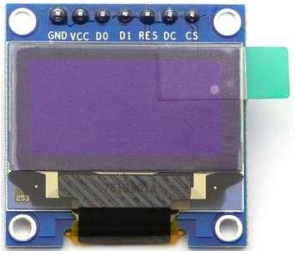


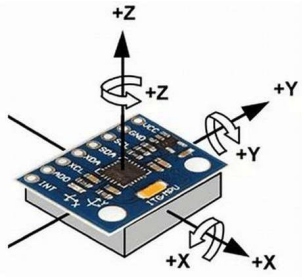
教学开发板

- 理论结合实践
  - GPIO点灯
  - 按键中断
  - 定时器中断
  - 液晶屏
  - OLED屏
  - 计算机通信
  - 加速度测量



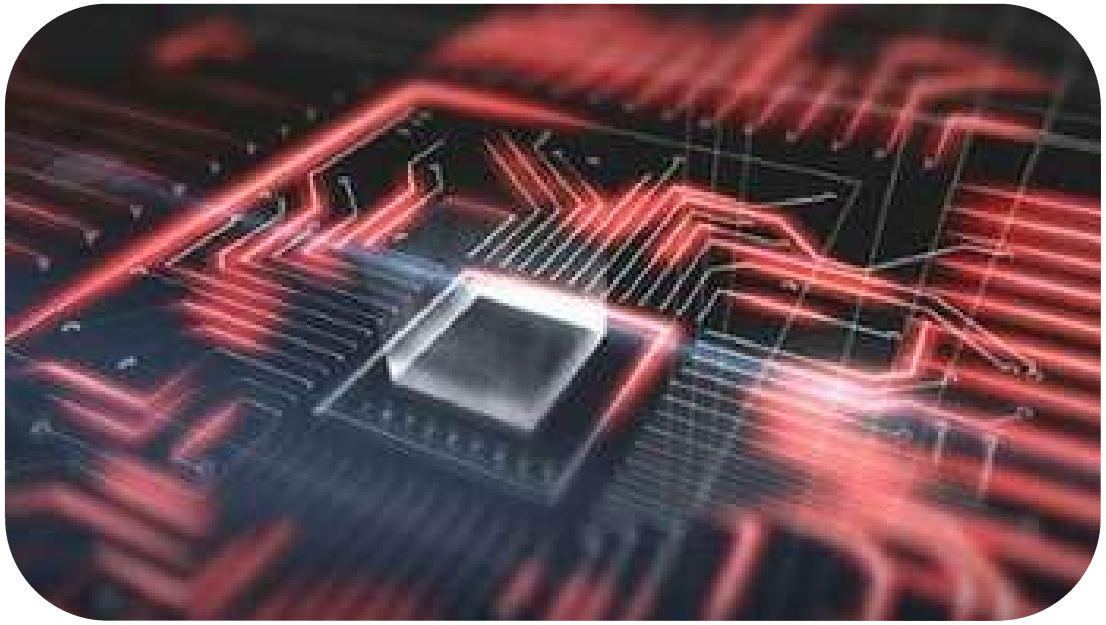
OLED屏幕





加速度传感器

嵌入式系统(EMBEDDED SYSTEM)



第4章 STM32外设进阶

❀ **4.2 IO外部中断及编程**

本节内容

- 中断基本概念
- STM32的中断系统
  - 中断源和中断向量表
  - EXTI中断控制器
  - NVIC中断控制器
- 常用库函数
- 中断服务程序规范
- IO中断编程实例 – 按键
- 课后编程练习E2.1：IO中断编程实现按键控制LED

中断 vs 轮询

- 比如，看看是否有来电。。。

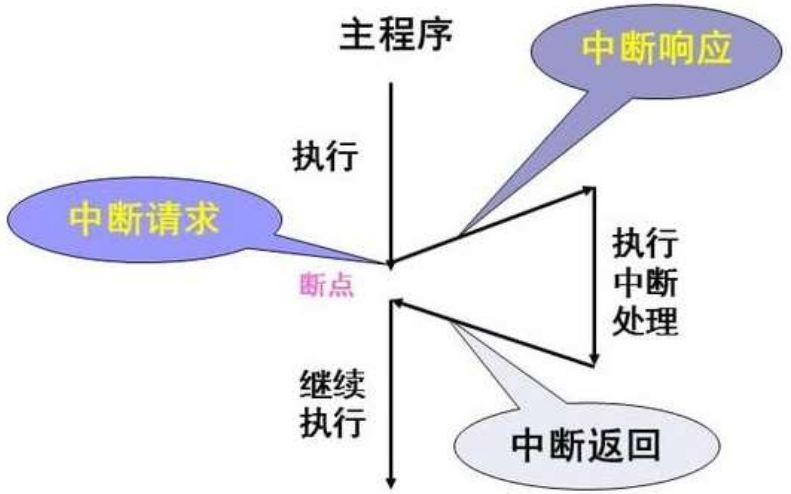
轮询：定期查询某一事件是否发生

主动方式为CPU

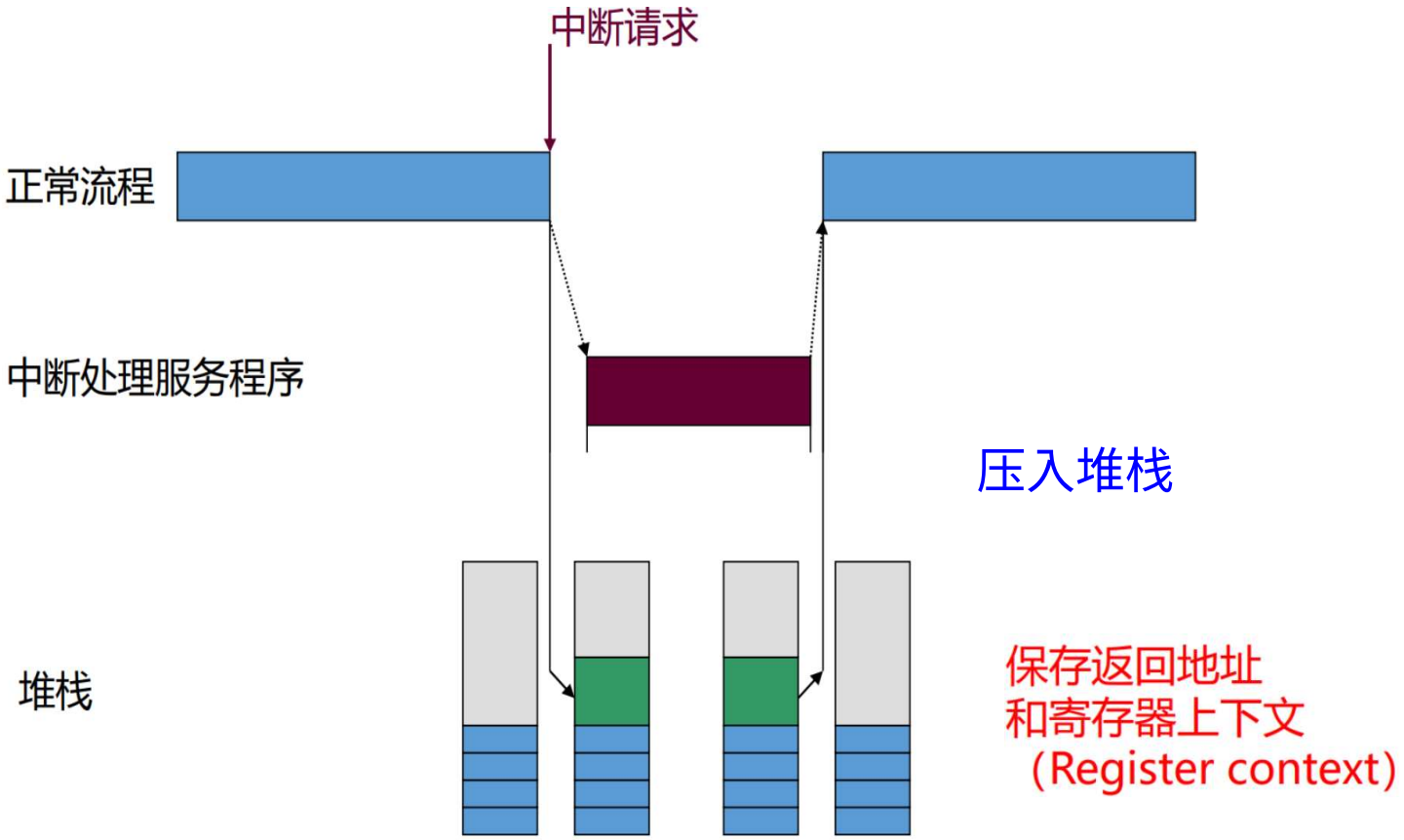
"Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring."

什么是中断

- 中断是一个需要CPU立即处理的内部/外部事件
  - 内部事件：定时器定时时间到
    AD变换结束
    ……
  - 外部事件：按键动作
    发生外部通信
    ……
- 内部/外部事件请求CPU处理
- CPU停止正常流程执行中断服务程序ISR
- ISR结束后，CPU返回正常流程



中断的堆栈占用


中断请求
正常流程
中断处理服务程序
堆栈
压入堆栈
保存返回地址
和寄存器上下文
(Register context)

实验一回顾

- E1.5 Button控制LED

切换闪烁模式

代码示例 – 轮询

```
// LED1、2 同时亮灭；LED3、4 交替亮灭
main()
{ ……
  LedOn(LED1);  //亮
  LedOn(LED2);  //亮
  LedOn(LED3);  //亮
  LedOff(LED4);   //灭
  while(1)
  {
   keytemp1 = GetKey();  /*第一次检测*/
   delayms(20);
   keytemp2 = GetKey();  /*第二次检测*/
if(keytemp1 == keytemp2) {
  switch(keytemp1)
  {
    case   KEY1:
      LedToggle(LED1);
      LedToggle(LED2);
      delayms(1000);
    break;
    case   KEY2:
      LedToggle(LED3);
      LedToggle(LED4);
     delayms(1000);
    break;
} } } }
```

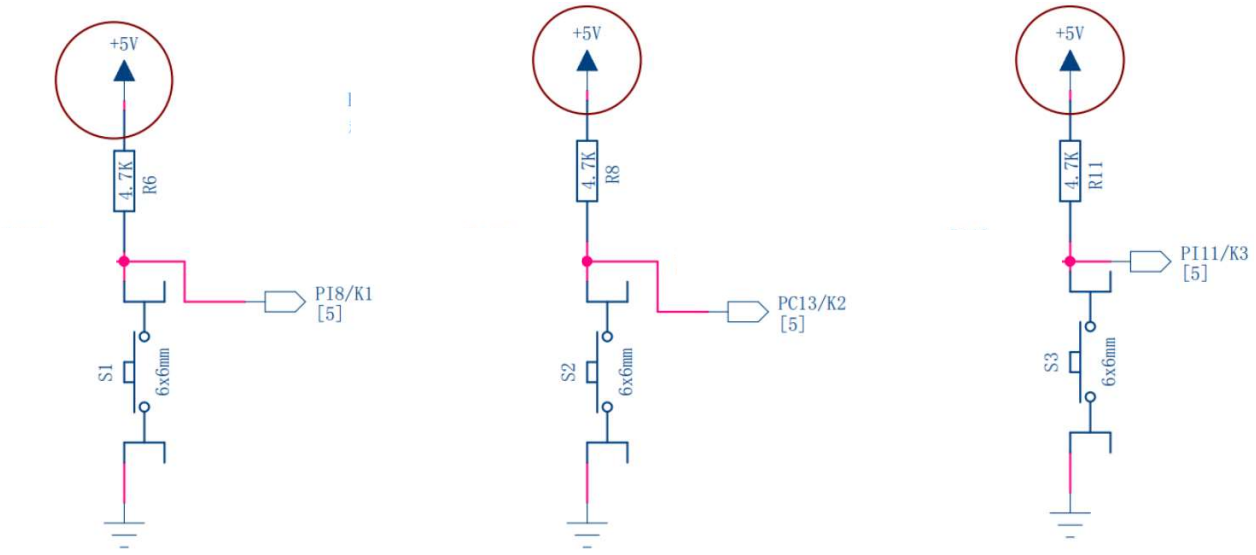
存在什么问题?

按键端口

- 按键连接的端口具有中断功能

F407开发板自定义按钮

都具备中断功能



不按为高电平，按下为低电平

中断使用的步骤

- 打开中断开关（Global & IRQx）
- 配置好对应的中断源（Interrupt Source）
- 编写对应的中断服务子程（ISR）

中断的允许/禁止控制

- **中断允许/禁止**
  - **全局中断控制（Global IE）** 一级开关

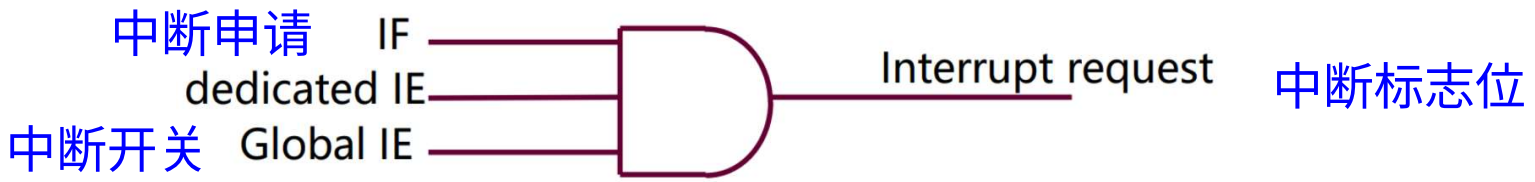
  -- ARM Cortex-M CPU寄存器中的一个特殊位（PM in PRIMASK）

  -- 上电复位后是Enable的

  - **指定中断控制（Dedicated IE）**

  -- 上电复位后所有中断都被Disable 二级开关

  -- ARM Cortex-M CPU的NVIC模块管理所有中断源



STM32的中断系统

- 中断来源
  - 系统异常
  - 可屏蔽中断

  (数量及优先级数取决于具体型号)
- 可屏蔽中断有
  - 外部中断（含GPIO）
  - 定时器中断
  - 串口通信中断

  ……
- 中断控制器
  - EXTI（仅管理外部中断）
  - NVIC（管理所有中断）

中断源和中断向量表

- 系统异常(F103)

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)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效(HardFault)	所有类型的失效	0x0000_000C
	0	可设置	存储管理(MemManage)	存储器管理	0x0000_0010
	1	可设置	总线错误(BusFault)	预取指失败，存储器访问失败	0x0000_0014
	2	可设置	错误应用(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指令的系统服务调用	0x0000_002C
	4	可设置	调试监控(DebugMonitor)	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

中断向量表是一个存储中断服务程序入口地址的特定内存区域，它可以让CPU在发生中断时快速找到对应的处理程序。

中断源和中断向量表

- 可屏蔽中断(F103)

外部中断

定时器中断

0	7	可设置	WWDG	窗口定时器中断	0x0000_0040
1	8	可设置	PVD	连到EXTI的电源电压检测(PVD)中断	0x0000_0044
2	9	可设置	TAMPER	侵入检测中断	0x0000_0048
3	10	可设置	RTC	实时时钟(RTC)全局中断	0x0000_004C
4	11	可设置	FLASH	闪存全局中断	0x0000_0050
5	12	可设置	RCC	复位和时钟控制(RCC)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通道1	DMA1通道1全局中断	0x0000_006C

外部中断

部分表格内容省略。。。

23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
24	31	可设置	TIM1_BRK	TIM1刹车中断	0x0000_00A0
25	32	可设置	TIM1_UP	TIM1更新中断	0x0000_00A4
26	33	可设置	TIM1_TRG_COM	TIM1触发和通信中断	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

定时器中断

部分表格内容省略。。。

58	65	可设置	DMA2通道3	DMA2通道3全局中断	0x0000_0128
59	66	可设置	DMA2通道4_5	DMA2通道4和DMA2通道5全局中断	0x0000_012C

STM32的中断系统

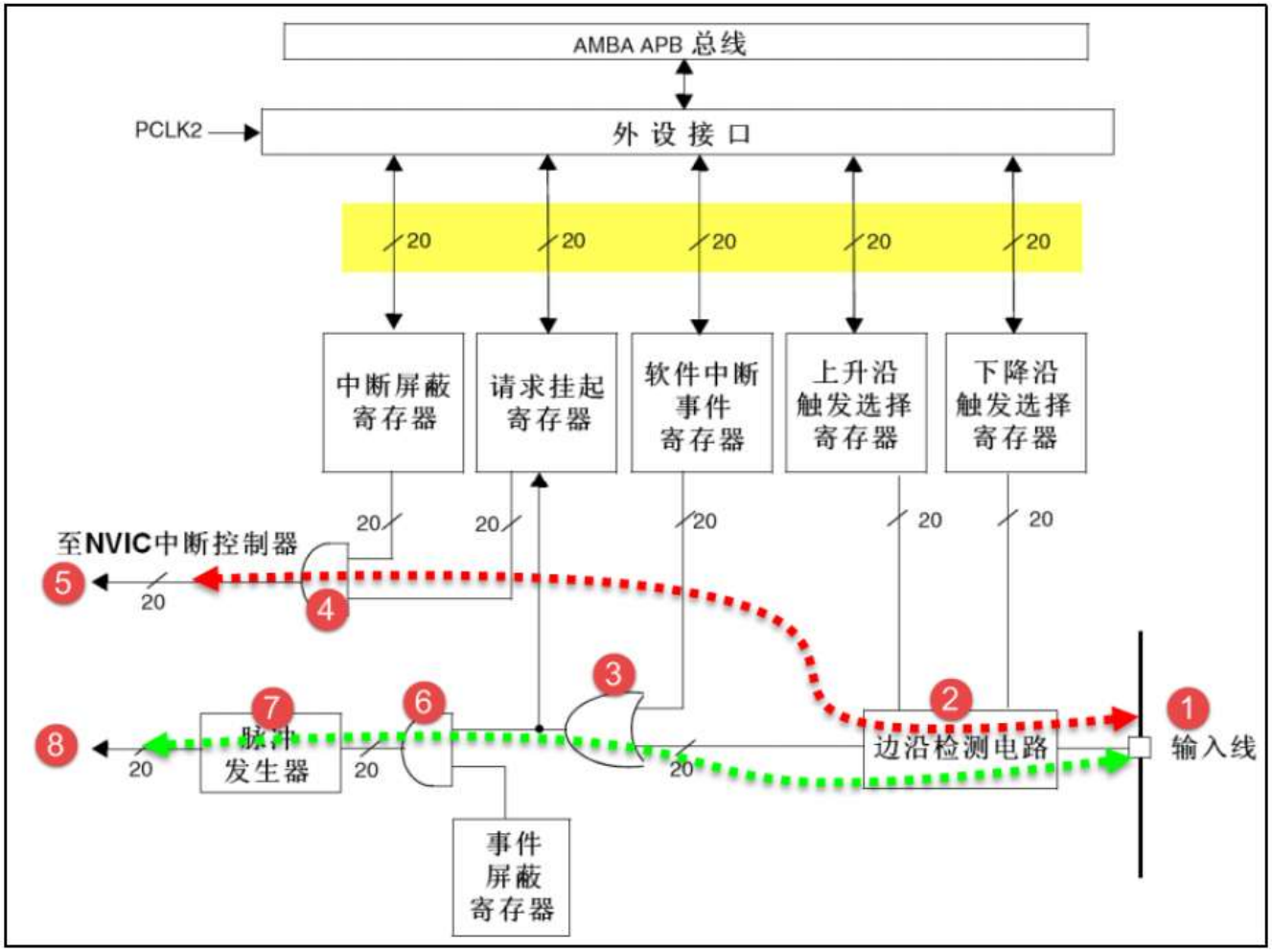
- 中断来源
  - 系统异常
  - 可屏蔽中断

  （数量及优先级数取决于具体型号）
- 可屏蔽中断有
  - 外部中断（含GPIO）
  - 定时器中断
  - 串口通信中断

  ……
- 中断控制器
  - EXTI（仅管理外部中断）
  - NVIC（管理所有中断）

EXTI 控制器

- **外部中断/事件控制器**
  - 可捕捉芯片引脚上请求
  - 中断请求信号路线（红线）
  - 事件请求信号路线（绿线）



EXTI 控制器

外部中断/事件线

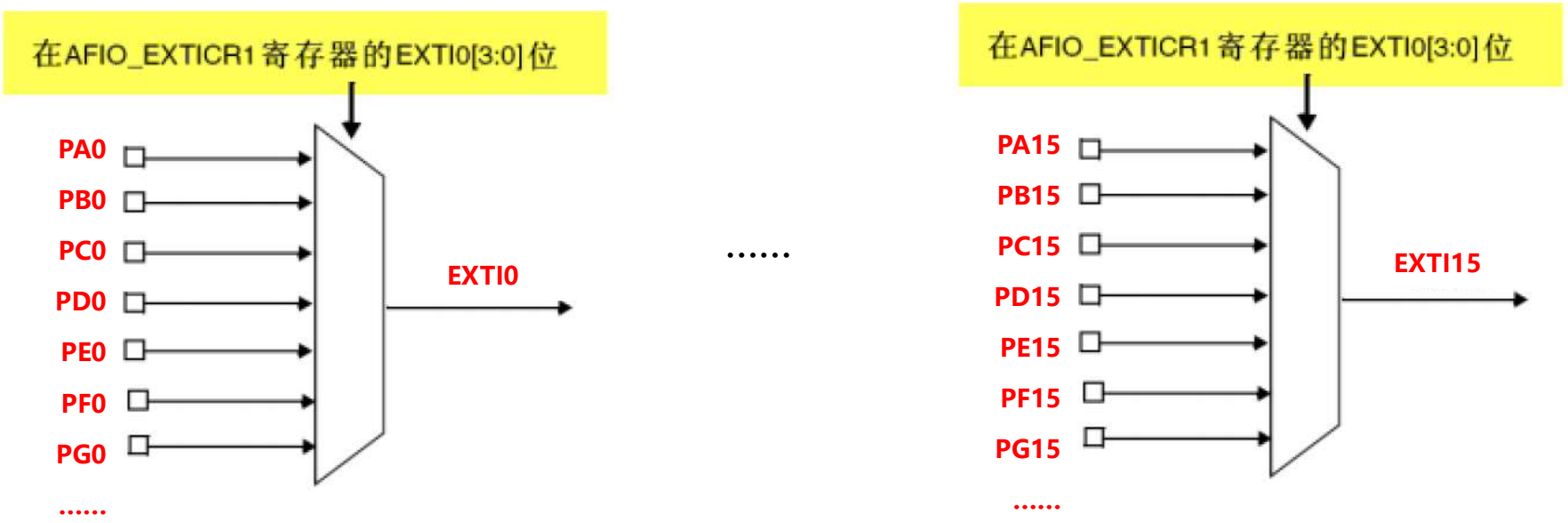
- GPIO引脚占16根 EXTI0~EXTI15
- 特定外设占4根 EXTI16~EXTI19

中断/事件线	输入源
EXTI0	PX0(X 可为 A, B, C, D, E, F, G, H, I)
EXTI1	PX1(X 可为 A, B, C, D, E, F, G, H, I)
EXTI2	PX2(X 可为 A, B, C, D, E, F, G, H, I)
EXTI3	PX3(X 可为 A, B, C, D, E, F, G, H, I)
EXTI4	PX4(X 可为 A, B, C, D, E, F, G, H, I)
EXTI5	PX5(X 可为 A, B, C, D, E, F, G, H, I)
EXTI6	PX6(X 可为 A, B, C, D, E, F, G, H, I)
EXTI7	PX7(X 可为 A, B, C, D, E, F, G, H, I)
EXTI8	PX8(X 可为 A, B, C, D, E, F, G, H, I)
EXTI9	PX9(X 可为 A, B, C, D, E, F, G, H, I)
EXTI10	PX10(X 可为 A, B, C, D, E, F, G, H, I)
EXTI11	PX11(X 可为 A, B, C, D, E, F, G, H, I)
EXTI12	PX12(X 可为 A, B, C, D, E, F, G, H, I)
EXTI13	PX13(X 可为 A, B, C, D, E, F, G, H, I)
EXTI14	PX14(X 可为 A, B, C, D, E, F, G, H, I)
EXTI15	PX15(X 可为 A, B, C, D, E, F, G, H, I)
EXTI16	PVD 输出
EXTI17	RTC 闹钟事件
EXTI18	USB 唤醒事件
EXTI19	以太网唤醒事件（只适用互联型）

EXTI 控制器

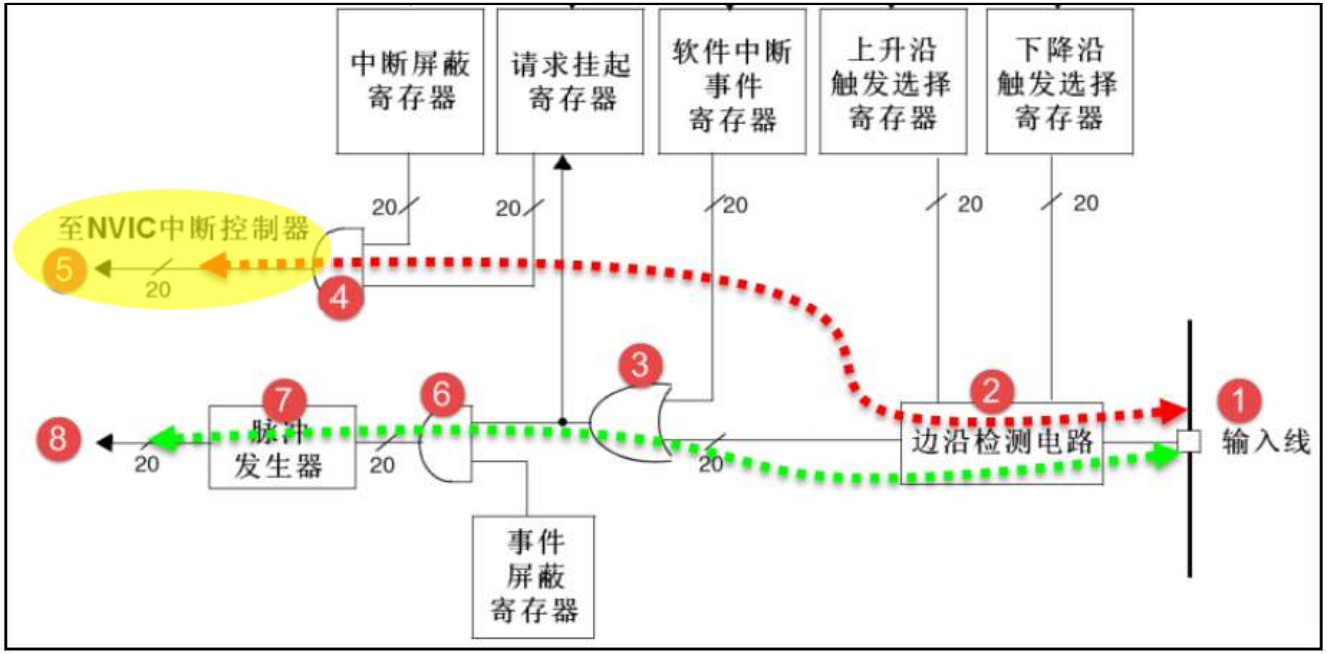
EXTI0~EXTI15 的输入源

- 每组Port的同编号引脚共用一个中断线路(EXTIx)



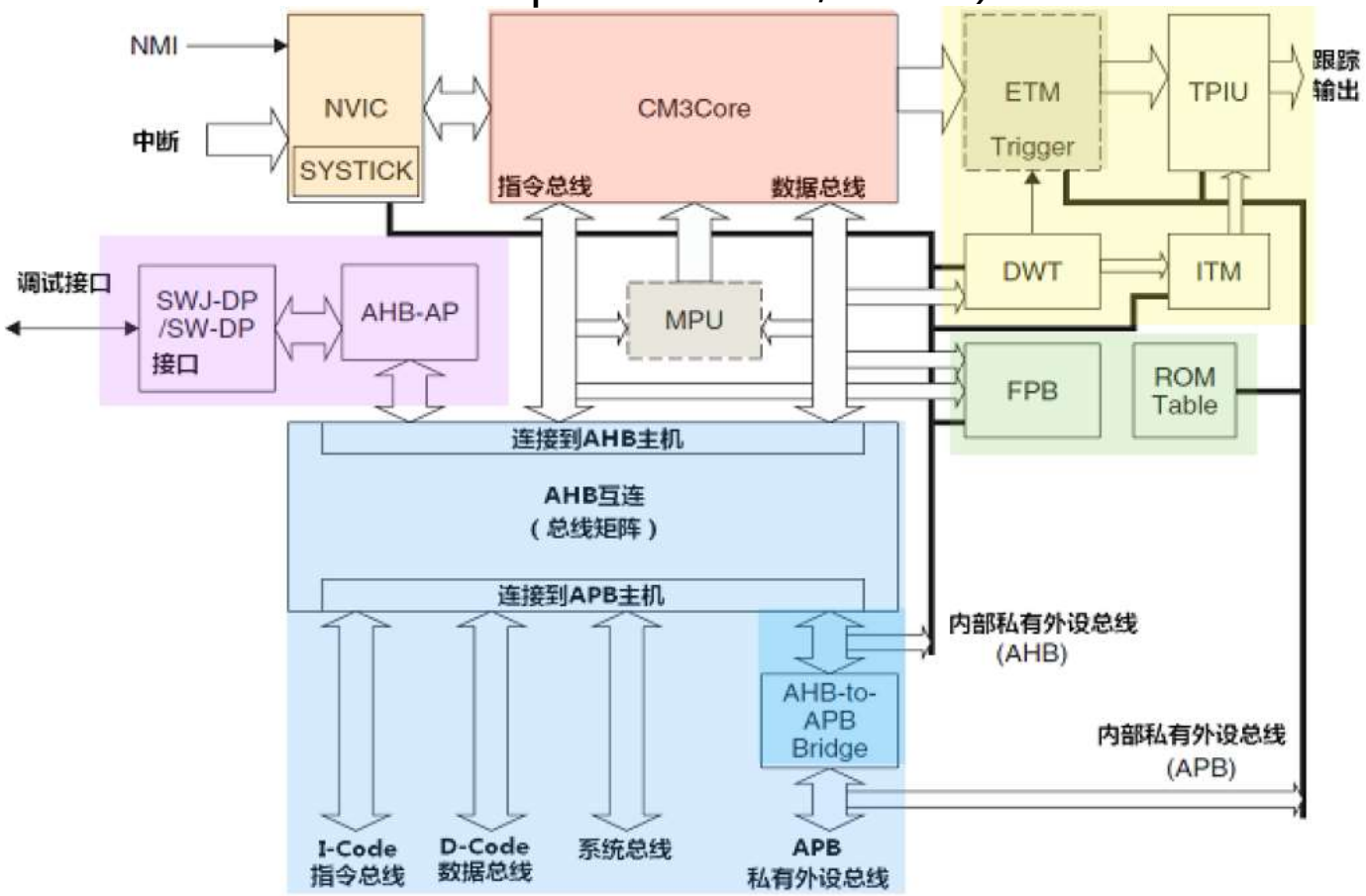
NVIC 控制器

- 接收来自EXTI控制器的请求信号



NVIC 控制器

- 嵌套向量中断控制器（Nested Vectored Interrupt Controller，NVIC）
  - 隶属于内核
  - 管理NMI
  - 管理所有可屏蔽中断
  - 优先级管理



中断优先级

- 多个中断同时出现时，高优先级中断先得到响应
- 中断优先级可以是固定的或者编程指定的
  - 固定优先级：根据中断向量表中约定顺序
  - 设定优先级：每个中断都有优先级设置位

    (比如ARM Cortex M4支持16个优先级)
- 相同优先级的中断，按先后顺序处理

优先级分组

- 优先级控制寄存器（NVIC IPRx）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
用于表达优先级				未使用，读回为 0			

  - 4bit表达，16个优先级
  - 分两组，抢占级(主)和响应级(子)，如下

优先级分组	主优先级	子优先级	描述
NVIC_PriorityGroup_0	0	0-15	主-0bit，子-4bit
NVIC_PriorityGroup_1	0-1	0-7	主-1bit，子-3bit
NVIC_PriorityGroup_2	0-3	0-3	主-2bit，子-2bit
NVIC_PriorityGroup_3	0-7	0-1	主-3bit，子-1bit
NVIC_PriorityGroup_4	0-15	0	主-4bit，子-0bit

-- 优先级：主级>子级；同级值小级高，如 0 >1

-- 中断嵌套：高级可中断低级，同级按时间先后

NVIC 常用库函数

- NVIC_PriorityGroupConfig (uint32_t　NVIC_PriorityGroup)

  功能：配置NVIC优先级分组

优先级分组	主优先级	子优先级	描述
NVIC_PriorityGroup_0	0	0-15	主-0bit，子-4bit
NVIC_PriorityGroup_1	0-1	0-7	主-1bit，子-3bit
NVIC_PriorityGroup_2	0-3	0-3	主-2bit，子-2bit
NVIC_PriorityGroup_3	0-7	0-1	主-3bit，子-1bit
NVIC_PriorityGroup_4	0-15	0	主-4bit，子-0bit

NVIC 常用库函数

(misc.h文件)

- NVIC_Init (NVIC_InitTypeDef* NVIC_InitStructure)

  功能：根据结构体NVIC_InitStructure完成对NVIC的参数初始化

- **NVIC初始化的结构体**

```
typedef struct
{
  uint8_t   NVIC_IRQChannel;                    /* 中断通道/中断源 */
  uint8_t   NVIC_IRQChannelPreemptionPriority;  /* 主优先级 */
  uint8_t   NVIC_IRQChannelSubPriority;         /* 子优先级 */
  FunctionalState   NVIC_IRQChannelCmd;         /*  使能或失能*/
} NVIC_InitTypeDef;
//定义初始化结构体变量
 NVIC_InitTypeDef   NVIC_InitStructure;
```

NVIC 常用库函数

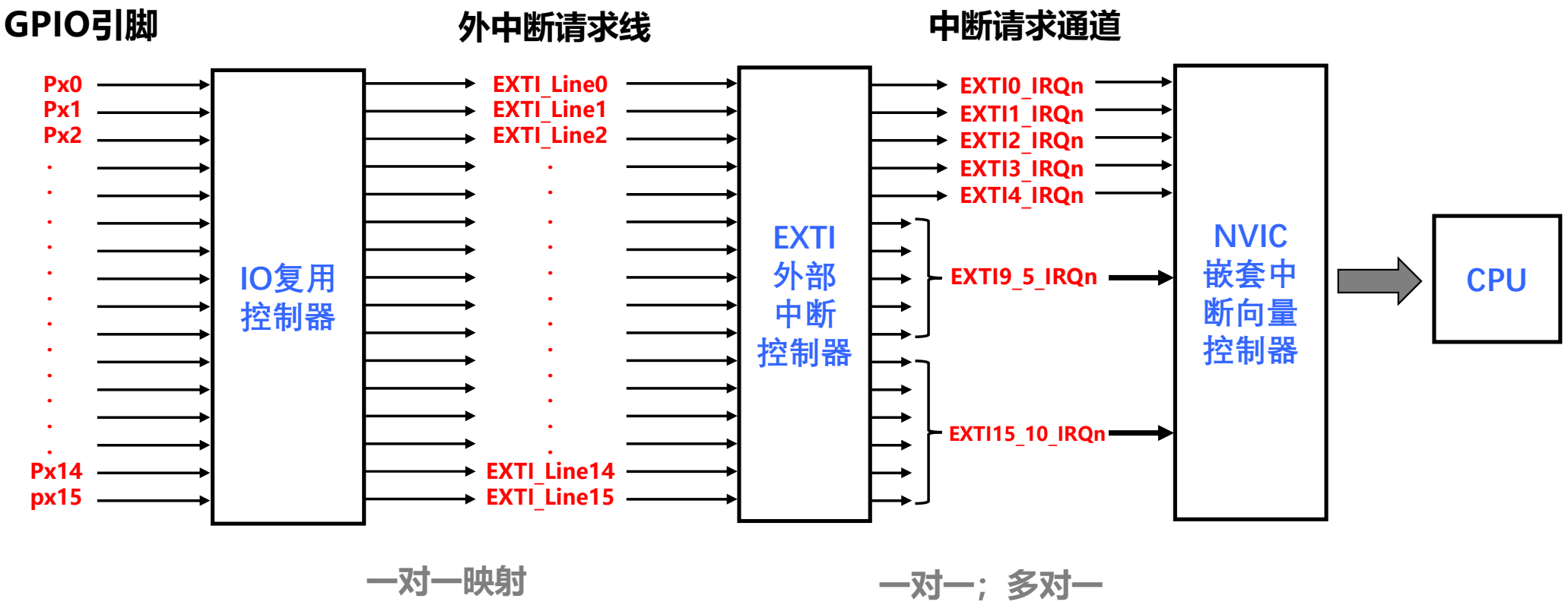
- 中断通道/中断源 NVIC_IRQChannel

NVIC_IRQChannel 取值表 (GPIO相关部分)

NVIC_IRQChannel 值	对应外中断线路	对应GPIO引脚
EXTI0_IRQn	EXTI_Line0	Px0
EXTI1_IRQn	EXTI_Line1	Px1
EXTI2_IRQn	EXTI_Line2	Px2
EXTI3_IRQn	EXTI_Line3	Px3
EXTI4_IRQn	EXTI_Line4	Px4
EXTI9_5_IRQn	EXTI_Line5~EXTI_Line9	Px5~Px9
EXTI15_10_IRQn	EXTI_Line10~EXTI_Line15	Px10~15

(更多值参照教材68页表)

EXTI外中断线与NVIC中断通道的映射关系



EXTI 常用库函数

- EXTI_Init (EXTI_InitTypeDef*　EXTI_InitStruct)

  功能：根据结构体EXTI_InitStructure完成对NVIC的参数初始化

- EXTI初始化结构体

```
typedef struct
{
  uint32_t                   EXTI_Line;      /*外中断线(号)*/
  EXTIMode_TypeDef           EXTI_Mode;      /*EXTI模式*/
  EXTITrigger_TypeDef        EXTI_Trigger;   /*触发类型*/
  FunctionalState            EXTI_LineCmd;   /*使能或失能*/
} EXTI_InitTypeDef;
  //定义初始化结构体变量
  EXTI_InitTypeDef   EXTI_InitStructure;
```

EXTI 常用库函数

EXTI初始化结构体参数选择

- EXTI_Line;　/*外中断线号*/

  -- EXTI0 ~ EXTI19　(其中0~15分配给GPIO引脚)

- EXTI_Mode;　　/*EXTI模式*/

  -- 枚举类型，中断模式 EXTI_Mode_Interrupt 或事件模式 EXTI_Mode_Event

- EXTI_Trigger;　　/*触发类型*/

  -- 枚举类型，上升沿↑ EXTI_Trigger_Rising 或下降沿↓ EXTI_Trigger_Falling 或二者 ↑↓

- EXTI_LineCmd;　/*使能或失能*/

  -- 使能ENABLE 或失能DISABLE

EXTI 常用库函数

- ITStatus EXTI_GetITStatus (uint32_t EXTI_Line)
  - 检测并返回中断线EXTI_Line上的中断标志位
  - 返回值类型 ITStatus 为枚举，定义如下

    typedef enum {RESET = 0, SET = 1} ITStatus; //0 – 无中断请求，1 – 有中断请求
- void EXTI_ClearFlag (uint32_t EXTI_Line)
  - 清除中断线EXTI_Line上的中断标志位IF
  - 写在中断服务子程函数最后，执行后IF被清0

中断服务程序规范

- 中断服务子程（ISR，Interrupt Service Routine）
  - 中断后执行的一个函数，实现某个用户任务（如点灯、蜂鸣、A/D转换……）
  - 在ARM Cortex-M平台上，中断服务子程与一般C函数写法没有区别
  - 在STM32工程中，中断服务子程可以统一写到stm32fxxx_it.c文件中
- 中断服务子程的共同特点
  - 被CPU自动调用的，而不是被其他程序调用
  - 每一个中断，必须调用对应的ISR
  - 在STM32工程中，ISR函数统一命名格式为 void PPP_IRQHandler(void){……}

    （PPP是中断源的缩写）

中断服务程序规范

- **PPP_IRQHandler 命名举例**
  - EXTI0_IRQHandler ; EXTI Line0 – Px0脚
  - EXTI1_IRQHandler ; EXTI Line1 – Px1脚
  - EXTI2_IRQHandler ; EXTI Line2 – Px2脚
  - EXTI3_IRQHandler ; EXTI Line3 – Px3脚
  - EXTI4_IRQHandler ; EXTI Line4 – Px4脚
  - EXTI9_5_IRQHandler ; EXTI Line[9:5]s – Px[9:5]脚
  - EXTI15_10_IRQHandler ; EXTI Line[15:10]s – Px[15:10]脚

  GPIO中断服务子程

  - TIM2_IRQHandler ; TIM2
  - TIM3_IRQHandler ; TIM3
  - TIM4_IRQHandler ; TIM4

  定时器中断服务子程

  ……

  (更多见STM32工程内的启动文件startup_...xxx.s第68~168行)

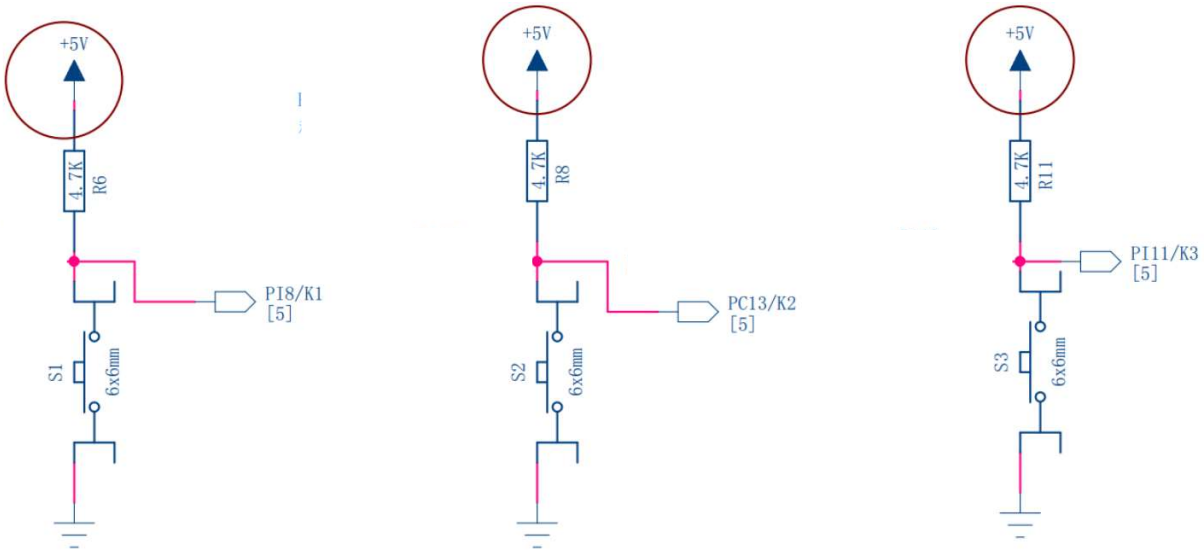
GPIO中断编程 - 按键

```
(void) PPP_IRQHandler(void)
{

    //Key1(PI8)控制LED
    ......

}
```

F407开发板自定义按钮



不按为高电平，按下为低电平

GPIO中断编程三步走

- Step 1：初始化NVIC – 配置分组、中断通道、优先级 ……
- Step 2：初始化EXTI
  - 开启EXTI时钟、开启GPIO复用功能、中断配置 ……
  - *开启中断引脚的GPIO时钟，设置IO输入模式 ……
- Step 3：编写中断服务子程

(* -- 之前已经讲过GPIO引脚初始化，故后面不再介绍)

初始化 NVIC

- **设置优先级分组方案，中断通道，优先级别，中断使能。**

```
void NvicCfg (void)
{   //定义初始化结构体变量
    NVIC_InitTypeDef  NVIC_InitStructure;
    //设置优先级分组方案
    NVIC_PriorityGroupConfig (NVIC_PriorityGroup_1);  /*方案1：主级1bit，子级3bit*/
    //NVIC配置 - 针对按键Key1(PI8)
    NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;            /*中断通道*/
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  /*主优先级 0~1级*/
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;            /*子优先级 0~7级*/
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;               /*使能*/
    NVIC_Init(&NVIC_InitStructure);                               /*完成初始化*/
}
```

初始化 EXTI

- **开启EXTI时钟，开启IO的EXTI复用功能，设置中断线号、中断方式、触发方式、中断使能**

```
void KeyExtiCfg (void)
{   //定义初始化结构体变量
    EXTI_InitTypeDef  EXTI_InitStructure;
    //开启EXTI的时钟
    RCC_APB2PeriphClockCmd (RCC_APB2Periph_SYSCFG, ENABLE);
    //开启GPIO的EXTI复用功能 – 针对Key1(PI8)
    SYSCFG_EXTILineConfig (EXTI_PortSourceGPIOI, EXTI_PinSource8);
    //EXTI配置 - 针对按键Key1(PI8)
   EXTI_InitStructure.EXTI_Line = EXTI_Line8;                  /*外部中断线路号*/
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;         /*中断方式*/
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;     /*下降沿触发*/
   EXTI_InitStructure.EXTI_LineCmd=  ENABLE;                   /*使能*/
   EXTI_Init(&EXTI_InitStructure);                             /*完成配置*/
}
```

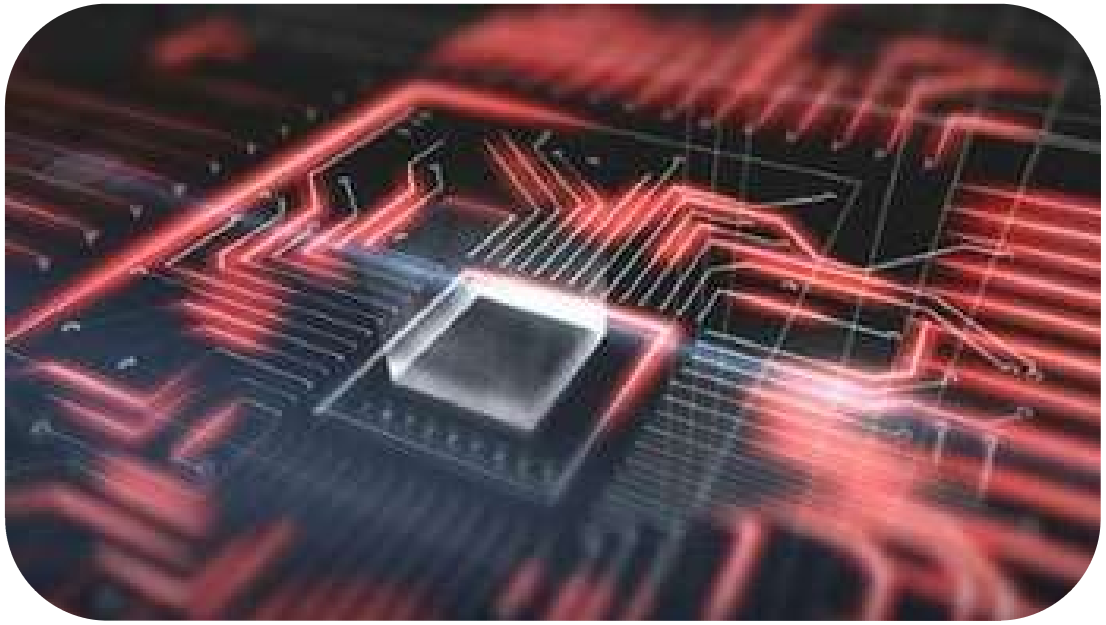
GPIO中断服务程序

- **流程：复核中断标志，复核按键状态，切换LED，清除中断标志**

```
void EXTI9_5_IRQHandler(void)
{
 if (EXTI_GetITStatus (EXTI_Line8) == 1)    /*检查中断标志是否为1*/
 {
    delayms(20);
    if (GPIO_ReadInputDataBit (KEY1_PORT, KEY1_Pin) == 0)    /*检测按键是否稳定按下*/
    {
      LedToggle(LED4);     /*4#灯亮灭切换*/
     }
      EXTI_ClearFlag(EXTI_Line8);     /*清除中断标志*/
 }
}
```

✓ 编程练习E2.1：中断编程实现按键控制LED (配套视频E2.1.1和E2.1.2)

嵌入式系统(EMBEDDED SYSTEM)



第4章 STM32外设进阶

❀ **4.3 通用定时器原理与编程**

本节内容

- 定时器基本概念
- STM32的通用定时器
  - 定时器结构
  - 定时器功能
- 常用库函数
- 定时中断服务程序规范
- 定时中断编程 – LED定时闪烁
- 课后编程练习E2.2：定时中断编程控制LED

实验回顾

- E1.5 & E2.1 Button控制LED

希望精确控制闪烁时间间隔?

定时器和计数器

- 定时器是MCU的重要特征
- 许多应用场合需要定时器
- 一块MCU提供多个定时器
- 定时器与计数器本质上相同

  -- 定时就是对固定的时间间隔进行计数

  -- 时钟可以是系统内部或外部时钟

- 计数器长度典型值为8位或16位

  -- 8位：0~255

  -- 16位：0~65535

Clock

位数决定最大脉冲段、时钟的个数

定时器的广泛用途

- 用途
  - 定时中断的产生
  - 产生特定的延时，或者测量延时
  - 周期与脉宽的测量
  - 频率测量
  - 外部事件计数　　本质是计数器
  - 波形调制输出（PWM）

- 定时器可以帮助编程者相对于主程序独立地、准确定时去执行一段程序
  - 主程序中的延时函数，往往会由于中断等事件的打断而不准确

定时器工作方式

- 自由运行方式
  - 正向、反向计数，计满上溢或到0下溢，然后开始新一轮
  - Quiz：1MHz，8bit定时器的定时时长，T = ？
- 可调制方式
  - 计数的上限可以预设，不一定计满为止，到预设后归零，然后开始新一轮
  - 可调计数器使用灵活，可以提供各级精度的定时，如1ms/1μs

STM32的定时器种类

- 系统滴答定时器SysTic （System Tick Timer）
  - 24位，递减
  - 通常用于实时操作系统
- 看门狗定时器（Watch Dog）
  - 7位和12位，递减
  - 系统出现问题，程序跑飞时复位系统
- 常规定时器
  - 16位，加或减或双向
  - 数量取决于型号
  - 定时、PWM调制、事件捕获、脉冲测量，……

STM32的常规定时器

- STM32F103系列
  - 基本定时器 TIM6、TIM7
  - 通用定时器 TIM2 ~ TIM5
  - 高级定时器 TIM1、TIM8

  (共8个)
- STM32F407系列
  - 基本定时器 TIM6、TIM7
  - 通用定时器 TIM2 ~ TIM5，TIM9 ~ TIM14
  - 高级定时器 TIM1、TIM8

  (共14个)

常规定时器比较

- 功能比较(F103)

主要功能	高级控制定时器	通用定时器	基本定时器
内部时钟源（8MHz）	●	●	●
带 16 位分频的计数单元	●	●	●
更新中断和 DMA	●	●	●
计数方向	向上、向下、双向	向上、向下、双向	向上
外部事件计数	●	●	○
其他定时器触发或级联	●	●	○
4 个独立输入捕获、输出比较通道	●	●	○
单脉冲输出方式	●	●	○
正交编码器输入	●	●	○
霍尔传感器输入	●	●	○
输出比较信号死区产生	●	○	○
制动信号输入	●	○	○

常规定时器引脚

- 通用定时器引脚分布

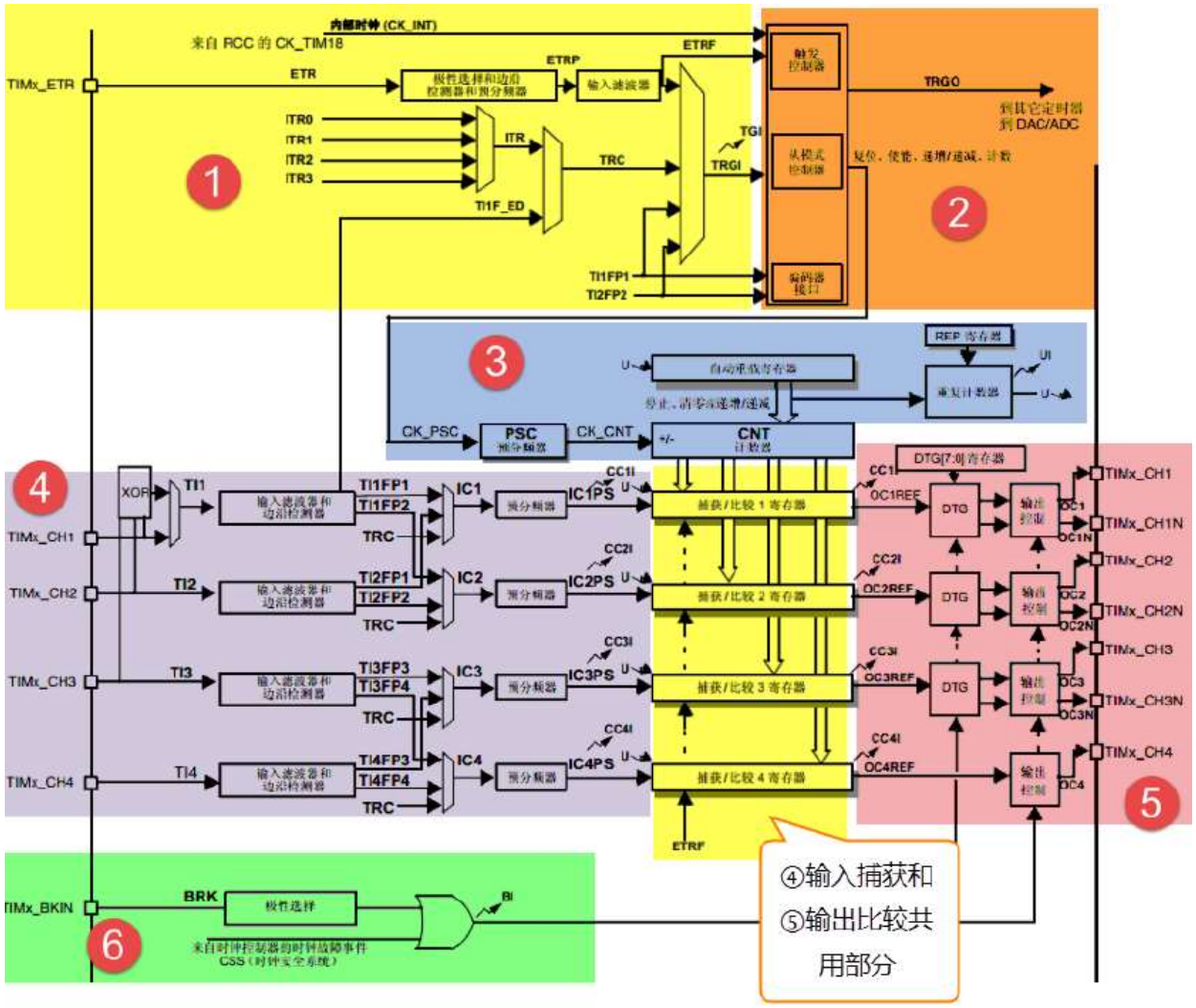
  (F103)

  - CHx 输入/输出通道
  - ETR 外部触发引脚

	高级定时器		通用定时器			
	TIM1	TIM8	TIM2	TIM5	TIM3	TIM4
CH1	PA8/PE9	PC6	PA0/PA15	PA0	PA6/PC6/PB4	PB6/PD12
CH1N	PB13/PA7/PE8	PA7				
CH2	PA9/PE11	PC7	PA1/PB3	PA1	PA7/PC7/PB5	PB7/PD13
CH2N	PB14/PB0/PE10	PB0				
CH3	PA10/PE13	PC8	PA2/PB10	PA2	PB0/PC8	PB8/PD14
CH3N	PB15/PB1/PE12	PB1				
CH4	PA11/PE14	PC9	PA3/PB11	PA3	PB1/PC9	PB9/PD15
ETR	PA12/PE7	PA0	PA0/PA15		PD2	PE0
BKIN	PB12/PA6/PE15	PA6				

通用定时器结构

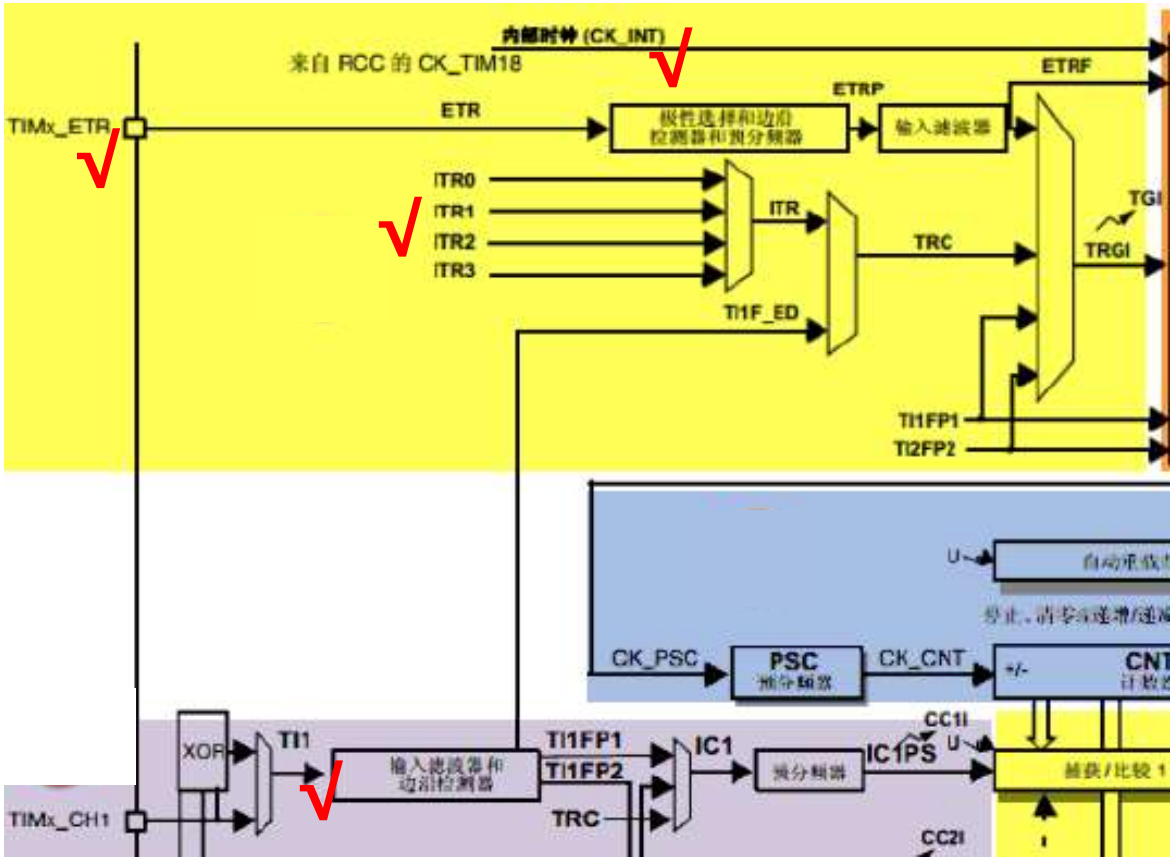
- 基本结构
  - ① 时钟源
  - ② 控制器
  - ③ 时基单元
  - ④ 输入捕获
  - ⑤ 输出比较
  - ⑥ 断路功能



通用定时器结构

时钟源

- 内部时钟 CK_INT
  - -- 来自系统时钟
  - -- 用于精确定时
- 外部时钟模式1：TIx输入
  - -- 外部信号计数
- 外部时钟模式2：ETR输入
  - -- 外部信号计数
- 内部触发输入：ITR(定时器提供)

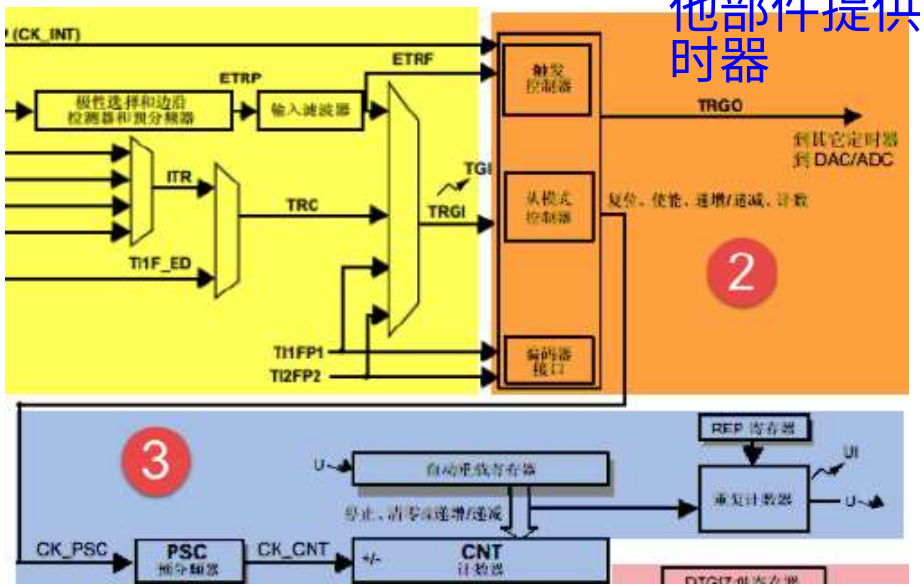


通用定时器结构

- **控制器(②)**
  - 触发控制器

    -- 为片上外设提供触发信号
  - 从模式控制器

    -- 控制计数器 复位、启动、增减
  - 编码器接口

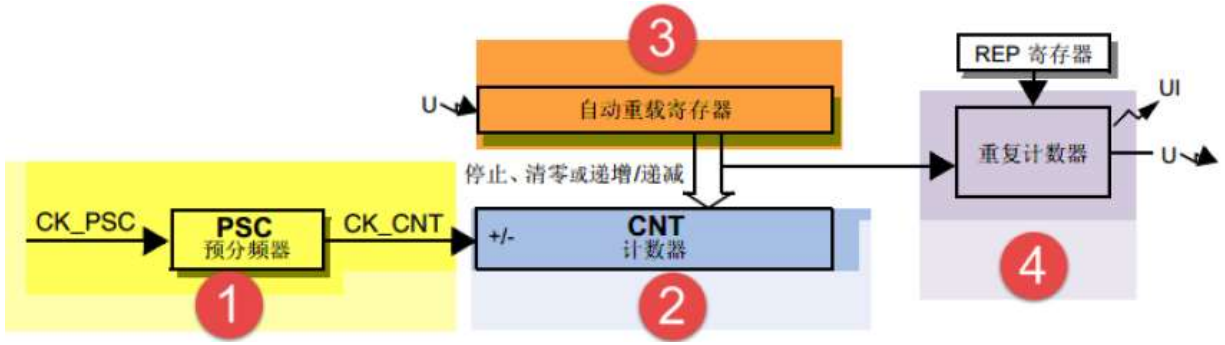


引出去，为其他部件提供计时器

通用定时器结构

时基单元

- 预分频器PSC
  - -- 对输入时钟进行分频
  - -- 驱动计数器
- 计数器CNT
  - -- 递增、递减、双向
  - -- 上溢或下溢后中断
- 自动重载寄存器ARR
  - -- 存放与CNT比较的值
- 重复计数器（高级定时器独有）



与其中的值进行对比，对比相同计时结束

通用定时器结构

输入捕获

- 信号边沿捕获（上升、下降、双边沿）
- 测量脉宽、频率、占空比
- 结构包括 （1-6）

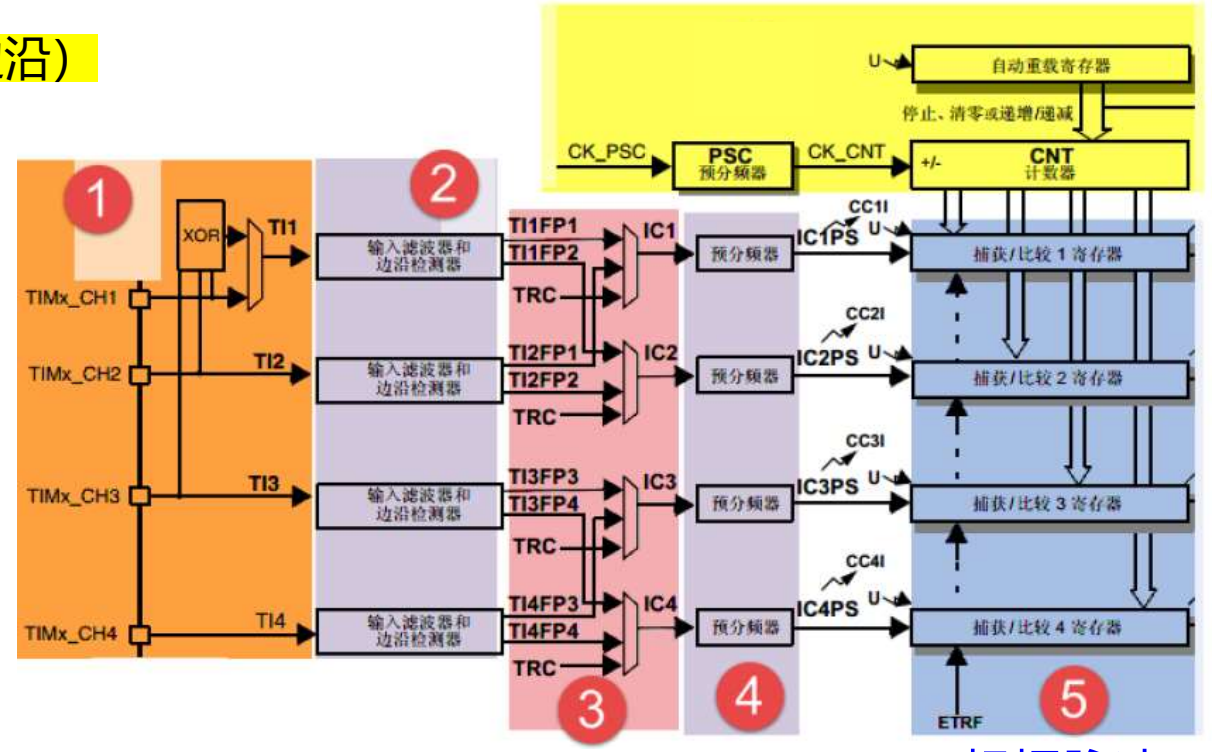
  -- 输入通道 TIx

  -- 滤波及边沿检测

  -- 捕获通道ICx

  -- 预分频器

  -- 捕获寄存器

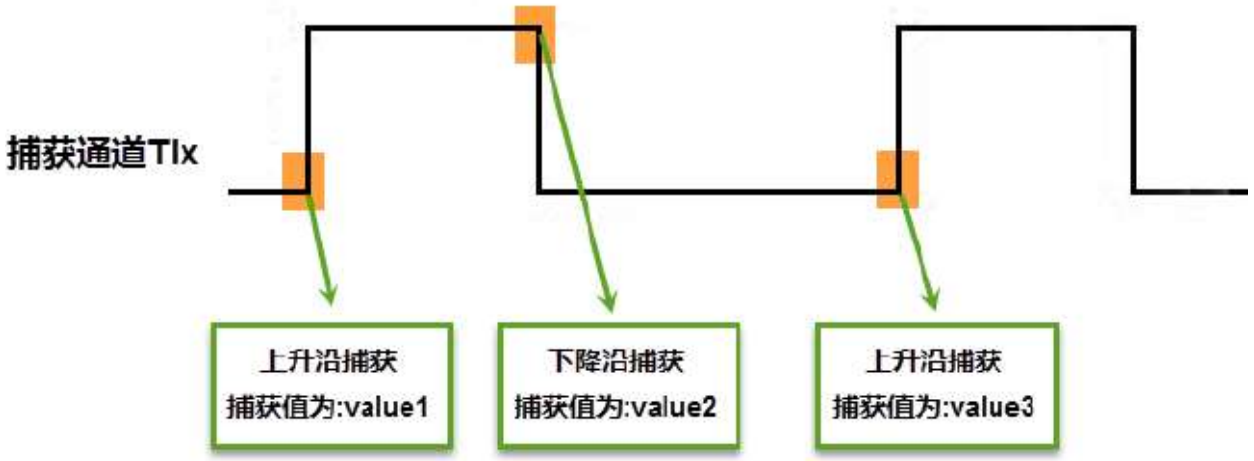


根据脉冲边沿做差值得到脉冲宽度

通用定时器功能 – 输入捕获

输入捕获应用

- 测量脉宽和频率



脉宽 tw = ?　　tw = value2 – value1

频率 f = ?　　f = 1/T = 1/(value3 – value1)

占空比 q = ?　　q = tw/T

通用定时器功能 – 精确定时

定时功能基本原理

- 依靠时基单元中的计数器(②)

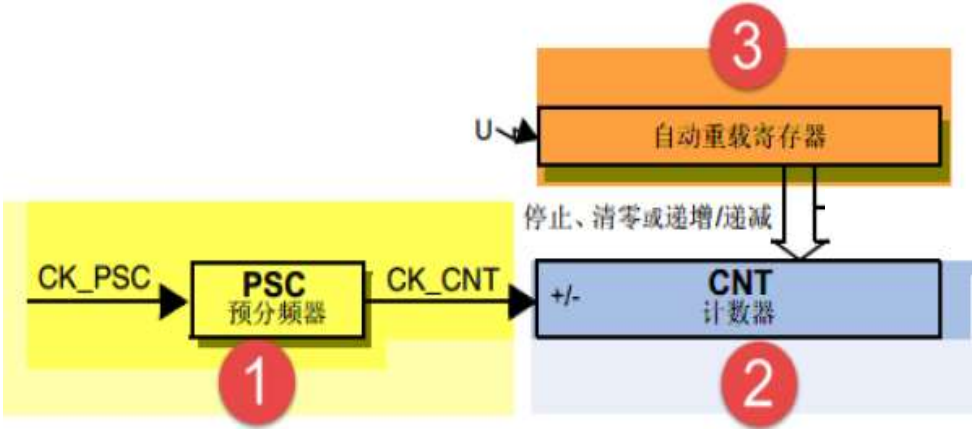
  -- 16位，0~65535

- 计数时钟为CK_CNT

  -- 对CK_PSC分频得到

- 分频由预分频器(①)完成

  -- 16位，0~65535



定时如何计算 t = ?

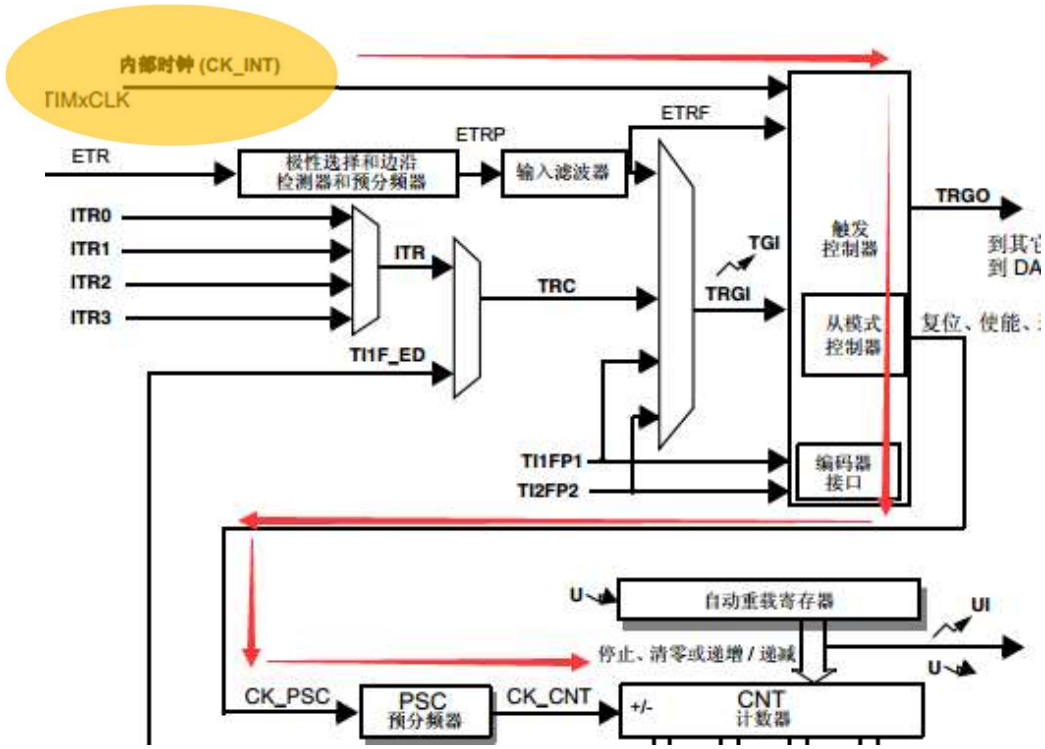
考试要考

Key： $t = 1/(CK\_PSC \div PSC) \times CNT$

通用定时器功能 – 精确定时

定时时钟来源

- 内部时钟(CK_INT)→控制器→CK_PSC
- CK_INT是从时钟树上获得的时钟TIMxCLK

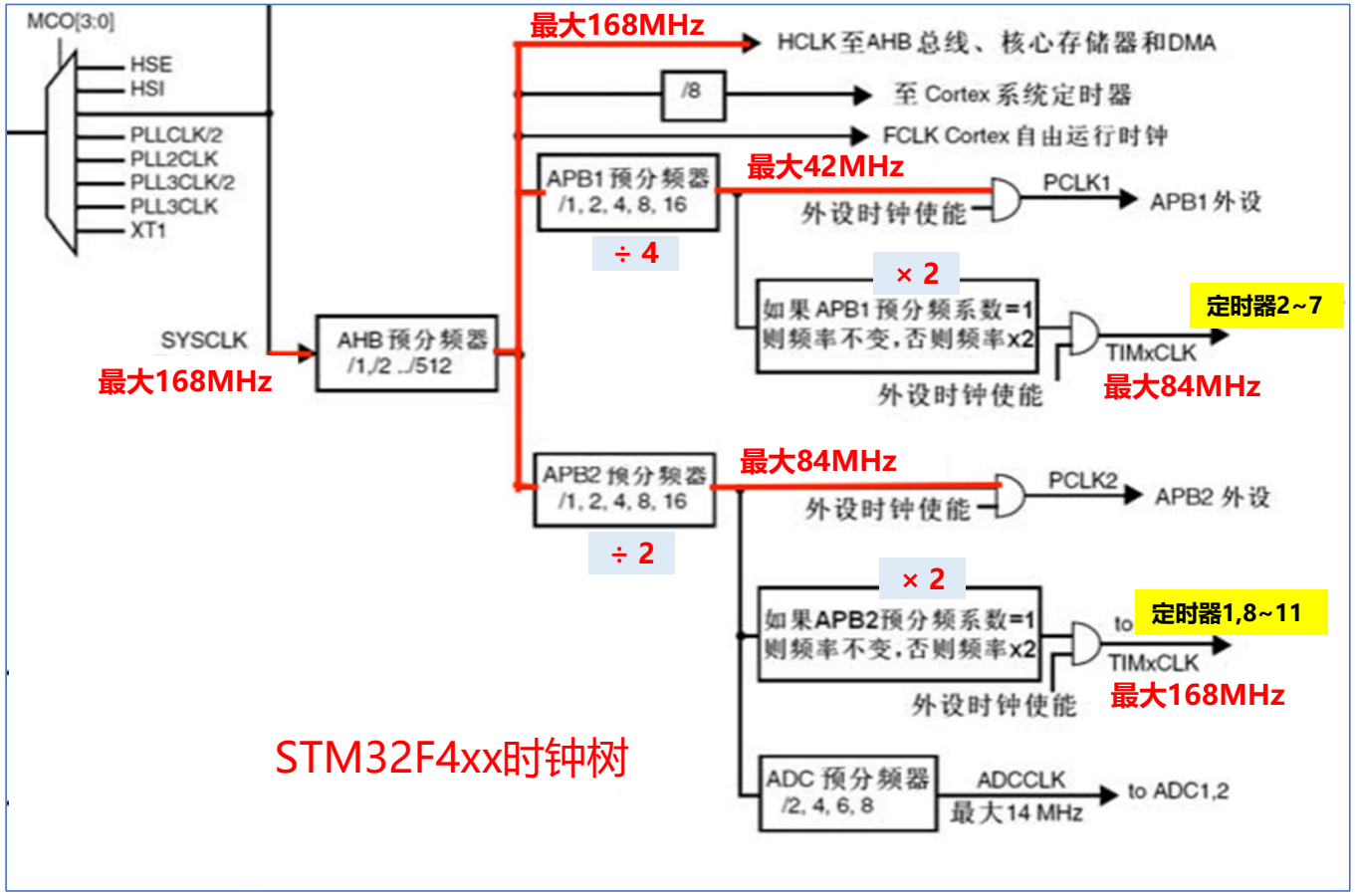


通用定时器功能 – 精确定时

定时器时钟TIMxCLK

- 取决于系统时钟SYSCLK频率和外设总线APBx的分频器比
- 开发板的F407IG芯片
  - -- 系统及AHB时钟最大168MHz
  - -- APB1总线最大42MHz
    - 所挂TIMxCLK最大84MHz
  - -- APB2总线最大84MHz
    - 所挂TIMxCLK最大168MHz

(数据手册 2.2.12-Clocks and startup)



通用定时器功能 – 精确定时

APBx所挂定时器确认

- stm32f4xx.h文件中宏定义

```
 /* APB1 总线外设 */
#define  TIM2_BASE       (APB1PERIPH_BASE + 0x0000)
#define  TIM3_BASE       (APB1PERIPH_BASE + 0x0400)
#define  TIM4_BASE       (APB1PERIPH_BASE + 0x0800)
#define  TIM5_BASE       (APB1PERIPH_BASE + 0x0C00)
 ……
/* APB2 总线外设 */
#define TIM1_BASE        (APB2PERIPH_BASE + 0x0000)
#define TIM8_BASE        (APB2PERIPH_BASE + 0x0400)
……
```

通用定时器功能 – 精确定时

- 定时器的设备时钟
  - stm32f4xx_rcc.h文件中宏定义

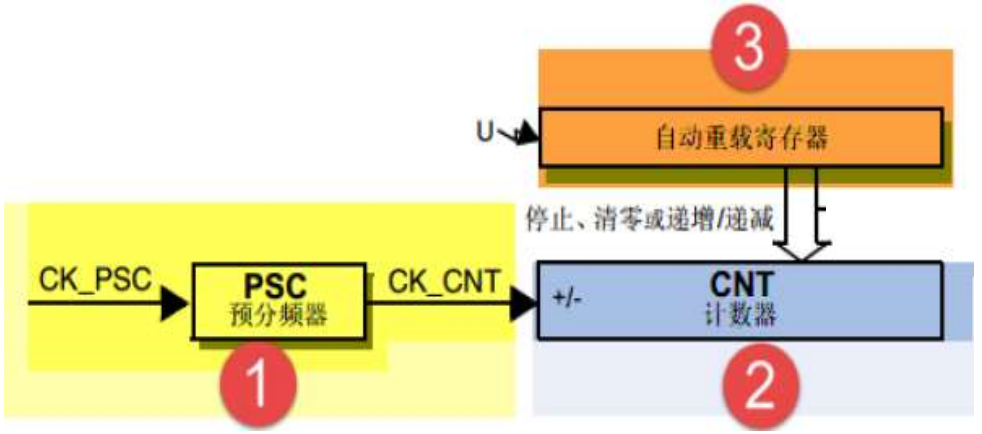
```
/* RCC_APB1_Peripherals */
#define  RCC_APB1Periph_TIM2          ((uint32_t)0x00000001)
#define  RCC_APB1Periph_TIM3          ((uint32_t)0x00000002)
#define  RCC_APB1Periph_TIM4          ((uint32_t)0x00000004)
#define  RCC_APB1Periph_TIM5          ((uint32_t)0x00000008)
……
```

通用定时器功能 – 精确定时

- 定时时间计算 - 举例
  - APB1总线频率42MHz
  - 使用定时器2 (TIM2)
  - 时基单元配置：

    -- 预分频PSC = 8400

    -- 计数值CNT= 10000



定时时长？ $t = 1/(CK\_PSC \div PSC) \times CNT$

分析：TIM2时钟 CK_PSC = APB1时钟×2 = 84MHz

$\therefore t = 1/(84\times10\text{^}6 \div 8400) \times 10000 = 1$秒

定时器 常用库函数

Base:基础

- TIM_TimeBaseInit (TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)

  功能：根据结构体 TIM_TimeBaseInitStruct 完成对定时器 TIMx 的时基单元初始化

- **时基初始化结构体**

```
typedef  struct
{
  uint16_t   TIM_Prescaler;          /* 预分频比 – PSC值 */
  uint16_t   TIM_Period;             /* 计数周期数 – CNT值 */
  uint16_t   TIM_CounterMode;        /* 计数模式 – 增、减或± */
  ……
} TIM_TimeBaseInitTypeDef;
  //定义初始化结构体变量
 TIM_TimeBaseInitTypeDef   TIM_TimeBaseInitStruct;
```

定时器 常用库函数

- TIM_ITConfig (TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)
  - 功能：使能/失能定时器 TIMx 的中断源 TIM_IT
  - 参数：TIMx - 定时器号

    TIM_IT - 中断源选择，如 TIM_IT_Update 定时中断源、TIM_IT_CCx 捕获/比较中断源

    NewState - ENABLE使能、或DISABLE失能
- TIM_Cmd (TIM_TypeDef* TIMx, FunctionalState NewState)
  - 功能：启动/停止定时器TIMx
  - 参数：NewState = ENABLE 启动，= DISABLE 停止

定时器中断服务程序规则

- 中断服务子程（ISR，Interrupt Service Routine）
  - 中断服务子程可以统一写到stm32f4xx_it.c文件中
  - 必须调用对应的ISR，函数命名格式 void TIMx_IRQHandler(void) {……}
  - 定时器中断的ISR具体命名如下

```
-- void  TIM2_IRQHandler (void)        ; 定时器2#
-- void  TIM3_IRQHandler (void)        ; 定时器3#
-- void  TIM4_IRQHandler (void)        ; 定时器4#
-- void  TIM5_IRQHandler (void)        ; 定时器5#
……
```

定时器中断编程 - 定时闪灯

```
/*定时器2的中断服务程序*/
(void) TIM2_IRQHandler(void)
{
   //每秒切换一次LED4状态
      ……


 }
```

定时器中断编程三步走

- Step 1：初始化NVIC – 配置分组、中断通道、优先级 ……
- Step 2：初始化定时器 – 开启TIM时钟、配置时基、中断方式 ……
- Step 3：编写定时器中断服务子程

初始化 NVIC

- **<NVIC.c>设置优先级分组方案，中断通道，优先级别，中断使能。**

```
void NvicCfg (void)
{  //定义初始化结构体变量
   NVIC_InitTypeDef  NVIC_InitStructure;
   //设置优先级分组方案
   NVIC_PriorityGroupConfig (NVIC_PriorityGroup_1);  /* 方案1: 主级1bit，子级3bit*/
   //NVIC配置 - 针对定时器2
   NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;                  /*中断通道*/
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  /*主优先级 0~1*/
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;               /*子优先级 0~7*/
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                  /*使能*/
   NVIC_Init(&NVIC_InitStructure);                                  /*完成初始化*/
}
```

中断总控器，包含所有可控式中断

数量越小级别越高

初始化 定时器

- **<timer.c> 开启外设时钟，设置预分频比、计数周期数、计数模式等**

```
void Tim2Init (void)
{  //定义初始化结构体变量
   TIM_TimeBaseInitTypeDef   TIM_TimeBaseStructure;
   //开启定时器2的外设时钟
   RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2 ,  ENABLE);
   //配置前先停止定时器
   TIM_Cmd(TIM2,  DISABLE);
   //定时器配置 – 时长为1秒
   TIM_TimeBaseStructure.TIM_Prescaler = 8400;                          /* 分频比*/
   TIM_TimeBaseStructure.TIM_Period = 10000;                            /* 计数周期数*/
   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;          /* 正向计数模式*/
   TIM_TimeBaseInit (TIM2, &TIM_TimeBaseStructure);                     /* 完成配置*/
   TIM_ITConfig (TIM2,  TIM_IT_Update, ENABLE);     /* 采用定时中断方式*/
   TIM_ClearFlag (TIM2,  TIM_FLAG_Update);          /* 清除中断标志*/
   TIM_Cmd (TIM2,  ENABLE);                         /* 启动定时器*/
}
```

什么是双向计数

到点响铃

定时器中断服务程序

力求稳妥

- **流程：复核中断标志，切换LED灯态，清除中断标志**

```
void  TIM2_IRQHandler (void)
{
  if (TIM_GetITStatus (TIM2,  TIM_IT_Update) == 1)    /*检查定时中断标志是否为1*/
  {
     LedToggle (LED4);     /*亮灭切换*/
     TIM_ClearFlag(TIM2, TIM_IT_Update);    /*清除中断标志*/
  }
 }
```

✓ 编程练习E2.2：定时中断编程实现控制LED (本次无演示视频)

思考题

- 中断请求和事件请求有何区别
- 解释什么是NVIC和EXTI
- 外中断请求线和中断请求通道的区别及映射关系（见PPT32页图）
- 中断嵌套的规则是怎样的
- 中断服务子程(ISR)最后一条语句为什么要清中断标志
- 描述一下通用定时器的时基单元构成及各部分功能
- 若APB1总线频率20MHz，TIM3的预分频为4000，计数上限为5000，列式计算定时时长
- 定时器中断编程分哪三步走

致谢

- 部分图片和文字来自网络、学堂在线慕课《ARM微控制器与嵌入式系统》

第4章 STM32外设进阶