第三讲　事出有因

世事无绝对，不可能总是一气呵成，做人也是一样，人生势必坎坷磨难，只要平常心即可。程序亦是如此，不可能总是 while(1); ，中断在所难免，本讲“事出有因”即是简述 STM32 的中断过程，M3 的 NVIC 中断也是一大亮点，希望经过本讲能知其“因”，得其“果”。

本讲学习目标：

1、了解中断体系。

2、完成外部中断与串口中断实验。

A：STM32 中断选讲

顾名思义，中断是因系统需要而打断原有的事件，执行较为优先的事件的过程。STM32 含有复杂的中断系统，但这也正是 STM32 的过人之处。

NVIC(Nested Vectored Interrupt Controller) -------嵌套向量中断控制器，是 M3 内核的亮点，“嵌套”是其核心！有了嵌套，那么我们就知道，势必会有一个概念应运而生即——**“优先级”**，现在来具体区分几个概念 ：

占先式优先级(pre-emption priority) ：高占先式优先级的中断事件会打断当前的主程序/中断程序运行——抢断式优先响应，俗称中断嵌套。

响应优先级（subpriority）：①在占先式优先级相同的情况下，高响应优先级的中断优先被响应；②在占先式优先级相同的情况下，如果有低响应优先级中断正在执行，高响应优先级的中断要等待已被响应的低响应优先级中断执行结束后才能得到响应——非抢断式响应(不能嵌套)。

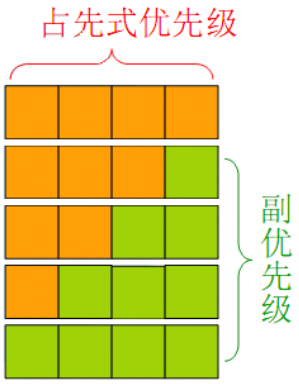
笔者的话：有人把响应优先级称作**'亚优先级'或'副优先级'**，每个中断源都需要被指定这两种优先级。

STM32 为了适应不同的优先级组合，设置了 GROUP 的概念，组是一个大的框架，在组下分别分配了占先优先级与副优先级。每一个中断都有一个专门的寄存器(Interrupt Priority Registers)来描述该中断的占先式优先级及副优先级。在这个寄存器中 STM32 使用 4 个二进制位描述优先级（Cortex-M3 定义了 8 位，但 STM32 只使用了 4 位）。严重要记住：四位！

笔者的话：四位，如果以二进制的思维来想就是 16 阶，也就是说，优先级被简单分为了十六阶的方式，但是从信息论与编码的角度来说，编码方式有了两种！有人会问，哪两种？答：上面已经说过了，占先式优先与副优先！十六阶被此二种编码分为了阵列，阵列何解？见下图：

4个优先级描述位可以有5种组合方式。

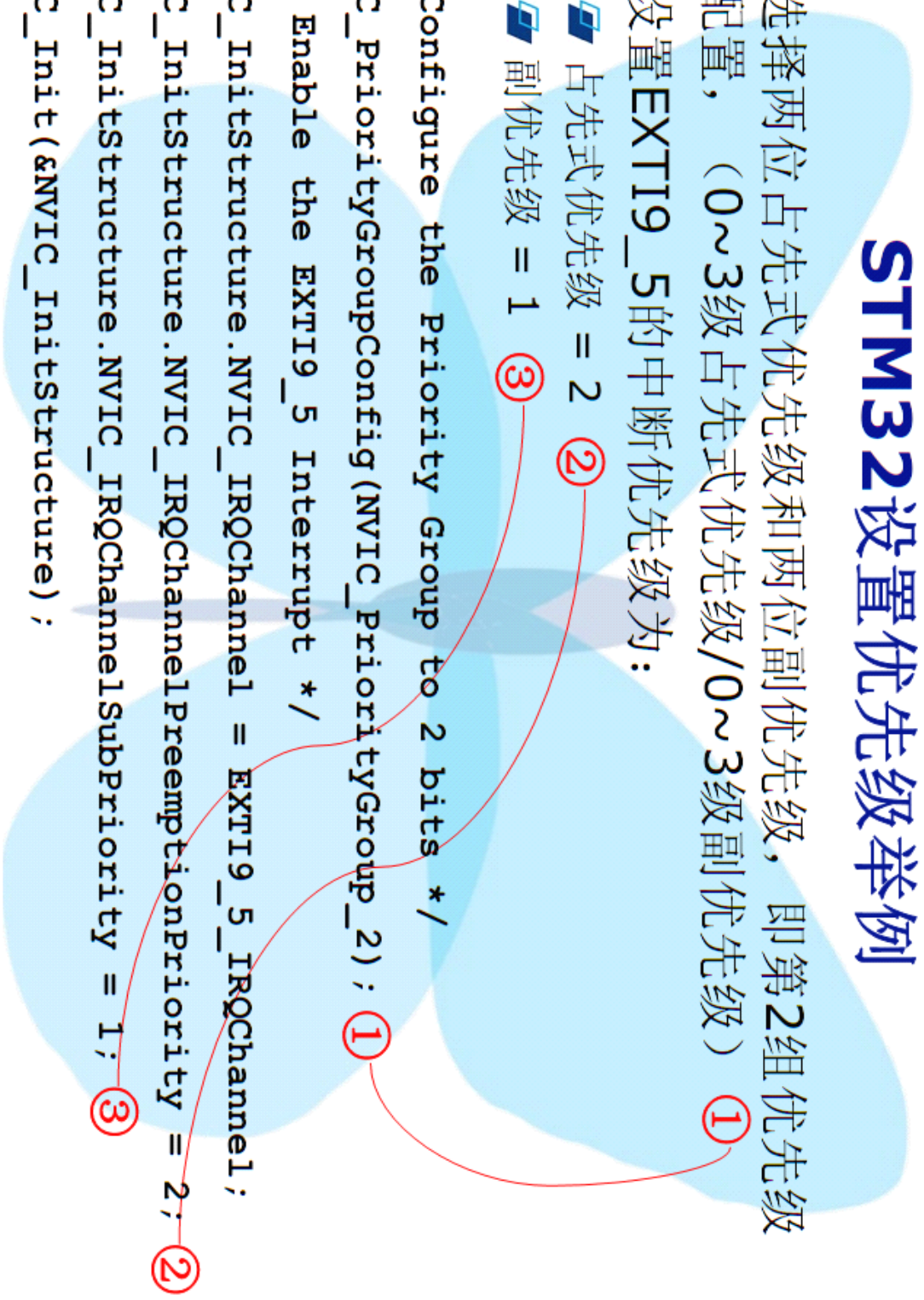
优先级组别	占先式优先级	副优先级
4	4位/16级	0位/0级
3	3位/8级	1位/2级
2	2位/4级	2位/4级
1	1位/2级	3位/8级
0	0位/0级	4位/16级



可以先用上电时候的 GROUP 来理解：

上电后，程序动作，自动设置为优先组 GROUP 0 ，则四位皆被分配至响应优先级，即响应式优先级，参看刚才的内容发现“在占先式优先级相同的情况下，高响应优先级的中断优先被响应；在占先式优先级相同的情况下，如果有低响应优先级中断正在执行，高响应优先级的中断要等待已被响应的低响应优先级中断执行结束后才能得到响应——非抢断式响应(不能嵌套)。”

在此我们看一段配置：（选自08年研讨会资料——常见应用解析 P45）



看到这里时候对 NVIC 的分组时候有概念了？接下来进入本章的另一个环节，外部中断。

B：外部中断

在上一讲中牵涉到了一些数字电路的问题，尤其着重的强调了 COMS 电路的“线与”是需要避免的，今天的外部中断，将有一些类似的性质牵涉其中，且听我细细说来：

笔者修改过后的网上资料：

STM32 中,所有 IO 脚都可以设置为 EINT 中断,对于外部中断线,有 16 路中断,即 EINT0-15 ，分别对应于管脚的 0-16 脚。例如，PA0、PA1 使用的 EINT 中断分别为 EXTI_Line0 ，和 EXTI_Line1。如果 PA0、PB0 呢?那就是共用 EXTI_Line0 咯,所以设计的时候要注意。

关于中断源，这里，外部中断的 0-4 使用独立的中断源，5-9 与 10-15 各自共用一个。所以你会在中断函数中看到有 EXTI9_5_IRQHandler() 和 EXTI15_10_IRQHandler() 这样的中断函数。那么你会问，如果进入了中断，我怎么知道是那路 EXINT 引起的呢？那就是通过读取中断标志位来判断的了，记住，16 路 EXINT 都是独立的，只是 5-9,10-15 共用了中断响应函数而已.程序中,可利用 **if(EXTI_GetITStatus(EXTI_Line5) != RESET)**语句判断是否是 LINE5 引起的中断。

使用库函数写中断，记得开放 **stm32f10x_conf.h**的 **#include "misc.h"**

关于软件设置中断：有时候在程序中我们会强制程序进入中断，这需要我们将中断标志位置位，而 STM32 提供了一个很方便的寄存器，即软中断事件寄存器，由以下库函数支持：

EXTI_GenerateSWInterrupt(EXTI_Line3);

这个函数运行的结果就是，如果 EXINT3 配置了必须的中断寄存器和响应函数，则会马上产生一个 EXINT3 中断。

E-mail：Poseidonstorm@126.com or 471661781@qq.com

判断中断是否会被响应的依据

1、首先是占先式优先级，其次是副优先级；

2、占先式优先级决定是否会有中断嵌套；

3、Reset、NMI、Hard Fault 优先级为负(高于普通中断优先级)且不可调整。

4、具体优先级是否可做调整请参看参考手册“9.1.2 中断和异常向量”。

软件配置：

1、NVIC_Configuration(void)——用以配置中断组与中断优先级；(参看第二版库中文说明)

设置优先级分组：先占优先级和从优先级	NVIC_PriorityGroupConfig
使能或者失能指定的 IRQ 通道	NVIC_InitStructure.NVIC_IRQChannel
设置了成员 NVIC_IRQChannel 中的先占优先级	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority
设置了成员 NVIC_IRQChannel 中的从优先级	NVIC_InitStructure.NVIC_IRQChannelSubPriority
设置了中断功能（失能/使能）	NVIC_InitStructure.NVIC_IRQChannelCmd

2、GPIO_Configuration(void)——用以配置中断功能；(此部分可参看第二版库的中文说明)

选择 GPIO 管脚用作外部中断线路	GPIO_EXTILineConfig
选择了待使能或者失能的外部线路	EXTI_InitStructure.EXTI_Line
设置了被使能线路的模式（事件或中断）	EXTI_InitStructure.EXTI_Mode
设置了被使能线路的触发边沿	EXTI_InitStructure.EXTI_Trigger
用来定义选中线路的新状态（使能或失能）	EXTI_InitStructure.EXTI_LineCmd

3、利用 stm32f10x_it.c 中实现中断之功能

C：串口

串口是有效的调试工具，在没有仿真器的年代，我们可以看到诸多的开发都利用串口的输出，来进行调试。本讲的主要内容是中断也就是 NVIC，串口在此即粗略带过。

1、配置串口（参看第二版库中文说明，注 3.0 以后的库结构体被分为了两个）

使能或者失能 AHB 外设时钟	RCC_APB2PeriphClockCmd
该成员设置了 USART 传输的波特率	USART_InitStructure.USART_BaudRate
帧传输或者接收到的数据位数	USART_InitStructure.USART_WordLength
定义了发送的停止位数目	USART_InitStructure.USART_StopBits
定义了奇偶模式	USART_InitStructure.USART_Parity
硬件流控制模式使能还是失能	USART_InitStructure.USART_HardwareFlowControl
指定了使能或者失能发送和接收模式	USART_InitStructure.USART_Mode

注意串口的 GPIO 复用设置：

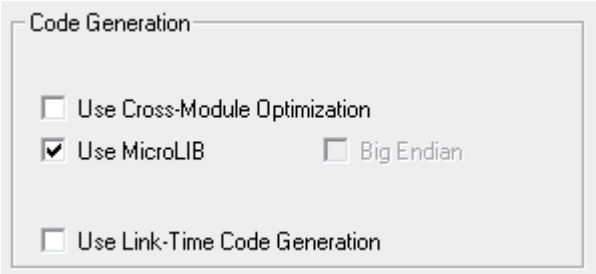
```
/* A9 USART1_Tx */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;          //推挽输出-TX
GPIO_Init(GPIOA, &GPIO_InitStructure);
/* A10 USART1_Rx   */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;//浮空输入-RX
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

2、同步串行接口设置

提示了 USART 时钟高低电平作用	USART_ClockInitStructure.USART_Clock
指定了下 SLCK 引脚上时钟输出的极性	USART_ClockInitStructure.USART_CPOL
指定了下 SLCK 引脚上时钟输出的相位	USART_ClockInitStructure.USART_CPHA
在 SCLK 引脚上输出最后发送的那个数据字	USART_ClockInitStructure.USART_LastBit

3、PUTC 相关(稍加改动即可以实现 Printf 功能)

第一步：使用内部库函数



第二步：定向 Putc

```
void USART1_Putc(unsigned char c)
{
    USART_SendData(USART1, c);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET );
}

void USART1_Puts(char * str)
{
    while(*str)
    {
        USART_SendData(USART1, *str++);
        /* Loop until the end of transmission */
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    }
}
```

第三步： USART1_Puts("\r\n 获取到串口测试");

串口方式举例：

串口 1 为查询方式、串口 2 为中断方式

Main 中的修改：

```
//串口2使用中断方式置位标志
if(Uart2_Get_Flag)
{
    Uart2_Get_Flag=0;
    USART2_Puts("\r\n获取到串口2数据:");
    USART2_Putc(Uart2_Get_Data);
    USART2_Puts("\r\n");
}
//串口1则使用查询方式
if(USART_GetFlagStatus(USART1,USART_FLAG_RXNE)==SET)
{
    uart1_get_data = USART_ReceiveData(USART1);
    USART1_Puts("\r\n获取到串口1数据:");
    USART1_Putc(uart1_get_data);
    USART1_Puts("\r\n");
```

It 中的更改：

```
extern u8 Uart2_Get_Flag;
extern u8 Uart2_Get_Data;

void USART2_IRQHandler(void)
{
    //接收中断
    if(USART_GetITStatus(USART2,USART_IT_RXNE)==SET)
    {
        USART_ClearITPendingBit(USART2,USART_IT_RXNE);
        Uart2_Get_Data=USART_ReceiveData(USART2);
        Uart2_Get_Flag=1;
    }

    //溢出-如果发生溢出需要先读SR,再读DR寄存器 则可清除不断入中断的问题
    if(USART_GetFlagStatus(USART2,USART_FLAG_ORE)==SET)
    {
        USART_ClearFlag(USART2,USART_FLAG_ORE); //读SR
        USART_ReceiveData(USART2);              //读DR
    }
}
```

C：课后练习

第三讲旨在初步学习中断系统，将 STM32 中断系统的层次深入的剖析，希望有所帮助。课后练习是根据自身理解，完成串口输入输出(PRINTF 方式)，外部中断输入两个程序，一定要学以致用啊，否则忘记的会很快。可以参考的手册有：

第二版库中文说明/M3 内核书/STM32 编程指南

D：听课笔记