STM32 中断优先级相关概念与使用笔记

上海 华东师范大学 通信工程系 ma-chao

一、基本概念

1．ARM cortex_m3 内核支持 256 个中断（16 个内核+240 外部）和可编程 256 级中断优先级的设置，与其相关的中断控制和中断优先级控制寄存器（NVIC、SYSTICK 等）也都属于 cortex_m3 内核的部分。STM32 采用了 cortex_m3 内核，所以这部分仍旧保留使用，但 STM32 并没有使用 cortex_m3 内核全部的东西（如内存保护单元 MPU 等），因此它的 NVIC 是 cortex_m3 内核的 NVIC 的子集。

2．STM32 目前支持的中断共为 84 个（16 个内核+68 个外部），和 16 级可编程中断优先级的设置（仅使用中断优先级设置 8bit 中的高 4 位，见后面解释）。《参考最新 101xx-107xx STM32 Reference manual, RM0008》。

3．以下主要对“外部中断通道”进行说明。

对于 cortex_m3 内核所支持的 240 个外部中断，我在这里使用了“中断通道”这个概念，因为尽管每个中断对应一个外围设备，但该外围设备通常具备若干个可以引起中断的中断源或中断事件。而该设备的所有的中断都只能通过该指定的“中断通道”向内核申请中断。因此，下面关于中断优先级的概念都是针对“中断通道”的。当该中断通道的优先级确定后，也就确定了该外围设备的中断优先级，并且该设备所能产生的所有类型的中断，都享有相同的通道中断优先级。至于该设备本身产生的多个中断的执行顺序，则取决于用户的中断服务程序。

4． STM32 可以支持的 68 个外部中断通道，已经固定的分配给相应的外部设备。每个中断通道都具备自己的中断优先级控制字节 PRI_n（8 位，但在 STM32 中只使用 4 位，高 4 位有效),每 4 个通道的 8 位中断优先级控制字(PRI_n)构成一个 32 位的优先级寄存器(Priority Register)。68 个通道的优先级控制字至少构成 17 个 32 位的优先级寄存器，它们是 NVIC 寄存器中的一个重要部分。

5．对于这 4bit 的中断优先级控制位还必须分成 2 组看：从高位开始，前面是定义抢先式优先级的位，后面用于定义子优先级。4bit 的分组组合可以有以下几种形式：

编 号	分配情况	
7	0:4	无抢先式优先级，16 个子优先级
6	1:3	2 个抢先式优先级，8 个子优先级
5	2:2	4 个抢先式优先级，4 个子优先级
4	3:1	8 个抢先式优先级，2 个子优先级
3/2/1/0	4:0	16 个抢先式优先级，无子优先级

6．在一个系统中，通常只使用上面 5 种分配情况的一种，具体采用哪一种，需要在初始化时写入到一个 32 位寄存器 AIRC（Application Interrupt and Reset Control Register）

的第[10：8]这 3 个位中。这 3 个 bit 位有专门的称呼：PRIGROUP（具体写操作后面介绍）。比如你将 0x05（即上表中的编号）写到 AIRC 的[10：8]中，那么也就规定了你的系统中只有 4 个抢先式优先级，相同的抢先式优先级下还可以有 4 个不同级别的子优先级。

7．AIRC 中 PRIGROUP 的值规定了设置和确定每个外部中断通道优先级的格式。例如，在上面将 0x05 写入了 AIRC 中 PRIGROUP，也就规定了当前系统中只能有 4 个抢先式优先级，相同的抢先式优先级下还可以有 4 个不同级别的子优先级，他们分别为：

位[7：6]		位[5：4]		位[3：0]
00	0 号抢先优先级	00	0 号子优先级	无效
01	1 号抢先优先级	01	1 号子优先级	无效
10	2 号抢先优先级	10	2 号子优先级	无效
11	3 号抢先优先级	11	3 号子优先级	无效

8．如果在你的系统中使用了 TIME2（中断通道 28）和 EXTI0（中断通道 6）两个中断，而 TIME2 中断必须优先响应，而且当系统在执行 EXIT0 中断服务时也必须打断（抢先、嵌套），就必须设置 TIME2 的抢先优先级比 EXTI0 的抢先优先级要高（数目小）。假定 EXTI0 为 2 号抢先优先级，那么 TIME2 就必须设置成 0 或 1 号抢先优先级。这些工作需要在 AIRC 中的 PRIGROUP 设置完成，确定了整个系统所具有的优先级个数后，再分别对每个中断通道（设备）进行设置。

9．具体优先级的确定和嵌套规则。ARM cortex_m3（STM32）规定

a/ 只能高抢先优先级的中断可以打断低抢先优先级的中断服务，构成中断嵌套。

b/ 当 2（n）个相同抢先优先级的中断出现，它们之间不能构成中断嵌套，但 STM32 首先响应子优先级高的中断。

c/ 当 2（n）个相同抢先优先级和相同子优先级的中断出现，STM32 首先响应中断通道所对应的中断向量地址低的那个中断（见 ROM0008，表 52）。

具体一点：

0 号抢先优先级的中断，可以打断任何中断抢先优先级为非 0 号的中断；1 号抢先优先级的中断，可以打断任何中断抢先优先级为 2、3、4 号的中断；……；构成中断嵌套。

如果两个中断的抢先优先级相同，谁先出现，就先响应谁，不构成嵌套。如果一起出现（或挂在那里等待），就看它们 2 个谁的子优先级高了，如果子优先级也相同，就看它们的中断向量位置了。

10．上电 Reset 后，寄存器 AIRC 中 PRIGROUP[10：8]的值为 0（编号 0），因此此时系统使用 16 个抢先优先级，无子优先级。另外由于所有外部中断通道的优先级控制字 PRI_n 也都是 0，所以根据上面的定义可以得出，此时 68 个外部中断通道的抢先优先级都是 0 号，没有子优先级的区分。故此时不会发生任何的中断嵌套行为，谁也不能打断当前正在执行的中断服务。当多个中断出现后，则看它们的中断向量地址：地址越低，中断级别越高，STM32 优先响应。

注意：此时内部中断的抢先优先级也都是 0 号，由于它们的中断向量地址比外部中断向量地址都低，所以它们的优先级比外部中断通道高，但如果此时正在执行一个外部中断服务，它们也必须排队等待，只是可以插队，当正在执行的中断完成后，它们可以优先得到执行。

了解以上基本概念还是不够的，还要了解具体中断的控制有那些途径，中断服务程序如何正确的编写。下面的描述主要以 TIME2 通道为例。

二、中断控制

1．对于 STM32 讲，外部中断通道位置 28（35 号优先级）是给外部设备 TIME2 的，但 TIME2 本身能够引起中断的中断源或事件有好多个，比如更新事件（上溢/下溢）、输入捕获、输出匹配、DMA 申请等。所有 TIME2 的中断事件都是通过一个 TIME2 的中断通道向 STM32 内核提出中断申请，那么 STM32 中如何处理和控制 TIME2 和它众多的、不同的、中断申请呢？

（题外话：STM32 中的一个通用定时计数器，就比 8 位控制器（如 AVR，MCS-51 就更不必说了）中 TIME 要复杂多了。学过 AVR 的，可能对输入捕获、输出匹配等还有概念，但如果你学的标准架构的 MCS-51，那么上手 32 位可能困难就更多了。所以我一直推荐学习 8 位机应该认真的从 AVR 开始。尽管 51 有很大的市场，价格也相对便宜，但从长远的眼光看问题，从后续掌握 32 位的使用，考虑到学生的可持续发展，AVR 应该是比较好的选择。）

2．cortex_m3 内核对于每一个外部中断通道都有相应的控制字和控制位，用于单独的和总的控制该中断通道。它们包括有：

- 中断优先级控制字：PRI_n（上面提到的）
- 中断允许设置位：在 ISER 寄存器中
- 中断允许清除位：在 ICER 寄存器中
- 中断悬挂 Pending（排队等待）位置位：在 ISPR 寄存器中（类似于置中断通道标志位）
- 中断悬挂 Pending（排队等待）位清除：在 ICPR 寄存器中（用于清除中断通道标志位）
- 正在被服务（活动）的中断（Active）标志位：在 IABR 寄存器中，（只读，可以知道当前内核正在处理哪个中断通道）

因此，与 TIME2 中断通道相关的，在 NVIC 中有 13 个 bits，它们是 PRI_28（IP[28]），的 8 个 bits(只用高 4 位)；加上中断通道允许，中断通道清除（相当禁止中断），中断通道 Pending 置位（我的理解是中断请求发生了，但当前有其它中断服务在执行，你的中断级别又不能打断别人，所以 Pending 等待，这个应该由硬件自动置位的），中断 Pending 位清除（可以通过软件将本次中断请求、且尚处在 Pending 状态，取消掉），正在被服务的中断（Active）标志位，各 1 个 bit。

上面的控制字和控制位都是分布在 NVIC 的寄存器组中的，可惜在 STM32 手册中竟然不给出任何的解释和说明。

3．作为外围设备 TIME2 本身也包括更具体的，管理自己不同中断的中断控制器（位），它们主要是自身各个不同类型中断的允许控制位，和各自相应的中断标志位（这个在 STM32 的手册中有详细的说明了）。

4．在弄清楚 2、3 两点的基础上，我们可以全程、全面和综合的来了解 TIME2 的中断过程，以及如何控制的。

a/ 初始化过程

首先要设置寄存器 AIRC 中 PRIGROUP 的值，规定系统中的抢先优先级和子优先级的个数

（在 4 个 bits 中占用的位数）；

设置 TIME2 本身的寄存器，允许相应的中断，如允许 UIE（TIME2_DIER 的第[0]位）

设置 TIME2 中断通道的抢先优先级和子优先级（IP[28]，在 NVIC 寄存器组中）

设置允许 TIME2 中断通道。在 NVIC 寄存器组的 ISER 寄存器中的一位。

b/ 中断响应过程

当 TIME2 的 UIE 条件成立（更新，上溢或下溢），硬件将 TIME2 本身寄存器中 UIE 中断标志置位，然后通过 TIME2 中断通道向内核申请中断服务。

此时内核硬件将 TIME2 中断通道的 Pending 标志置位（相当与中断通道标志置位），表示 TIME2 有中断申请。

如果当前有中断在处理，TIME2 的中断级别不够高，那么就保持 Pending 标志，当然用户可以在软件中通过写 ICPR 寄存器中相应的位把本次中断清除掉。

当内核有空，开始响应 TIME2 的中断，进入 TIME2 的中断服务。此时硬件将 IABR 寄存器中相应的标志位置位，表示 TIME2 中断正在被处理。同时硬件清除 TIME2 的 Pending 标志位。

c/ 执行 TIME2 的中断服务程序

所有 TIME2 的中断事件，都是在一个 TIME2 中断服务程序中完成的，所以进入中断程序后，中断程序需要首先判断是哪个 TIME2 的具体事件的中断，然后转移到相应的服务代码段去。

注意不要忘了把该具体中断事件的中断标志位清除掉，硬件是不会自动清除 TIME2 寄存器中具体的中断标志位的。

如果 TIME2 本身的中断事件多于 2 个，那么它们服务的先后次序就由用户编写的中断服务决定了。换句话说，对于 TIME2 本身的多个中断的优先级，系统是不能设置的。所以用户在编写服务程序时，应该根据实际的情况和要求，通过软件的方式，将重要的中断优先处理掉。

当然你也可以每次中断服务只处理其中的一个，然后再次进入中断，处理下一个。

d/ 中断返回

内核执行完中断服务后，便进入中断返回过程，在这个过程中需要：

硬件将 IABR 寄存器中相应的标志位清另，表示该中断处理完成

如果 TIME2 本身还有中断标志位置位，表示 TIME2 还有中断在申请，则重新将 TIME2 的 Pending 标志置为 1，等待再次进入 TIME2 的中断服务。

以上中断过程在《ARM Cortex-M3 权威指南》中有详细描述，并配合时序图说明，可以参考。

上述两点弄清楚后，就可以在 ST 提供的函数库的帮助下，正确的设置和使用 STM32 的中断系统了。

如果你要了解更深入的东西，或者直接对寄存器操作，还要继续望下看。

三、深入 NVIC

1. 看看 Cortex-M3 中与 NVIC 相关的寄存器有那些

SysTick	Control and Status Register	Read/write	0xE000E010
SysTick	Reload Value Register	Read/write	0xE000E014
SysTick	Current Value Register	Read/write clear	0xE000E018
SysTick	Calibration Value Register	Read-only	0xE000E01C
//==================			
Irq 0 to 31	Set Enable Register	Read/write	0xE000E100
. . . . .			
Irq 224 to 239	Set Enable Register	Read/write	0xE000E11C
//==================			
Irq 0 to 31	Clear Enable Register	Read/write	0xE000E180
. . . . .			
Irq 224 to 239	Clear Enable Register	Read/write	0xE000E19C
//==================			
Irq 0 to 31	Set Pending Register	Read/write	0xE000E200
. . . . .			
Irq 224 to 239	Set Pending Register	Read/write	0xE000E21C
//==================			
Irq 0 to 31	Clear Pending Register	Read/write	0xE000E280
. . . . .			
Irq 224 to 239	Clear Pending Register	Read/write	0xE000E29C
//==================			
Irq 0 to 31	Active Bit Register	Read-only	0xE000E300
. . . . .			
Irq 224 to 239	Active Bit Register	Read-only	0xE000E31C
//==================			
Irq 0 to 3	Priority Register	Read/write	0xE000E400
. . . . .			
Irq 224 to 239	Priority Register	Read/write	0xE000E4EC
//=======================			
CPUID Base Register		Read-only	0xE000ED00
Interrupt Control State Register		Read/write or read-only	0xE000ED04
Vector Table Offset Register		Read/write	0xE000ED08
Application Interrupt/Reset Control Register		Read/write	0xE000ED0C
System Control Register		Read/write	0xE000ED10
Configuration Control Register		Read/write	0xE000ED14
System Handlers 4-7 Priority Register		Read/write	0xE000ED18
System Handlers 8-11 Priority Register		Read/write	0xE000ED1C
System Handlers 12-15 Priority Register		Read/write	0xE000ED20
. . . . .			

2．STM32 中用了那些

下面是从 ST 公司提供的函数库的头文件得到的，库的版本是 v3.1.0

```
/* memory mapping struct for Nested Vectored Interrupt Controller (NVIC) */
typedef struct
{
  __IO uint32_t ISER[8];               /*!< Interrupt Set Enable Register */
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];               /*!< Interrupt Clear Enable Register */
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];               /*!< Interrupt Set Pending Register */
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];               /*!< Interrupt Clear Pending Register */
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];               /*!< Interrupt Active bit Register */
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];               /*!< Interrupt Priority Register, 8Bit wide */
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;                  /*!< Software Trigger Interrupt Register */
}  NVIC_Type;
```

a/ 寄存器 ISER、ICER、ISPR、ICPR、IABR 在 STM32 中都使用的 8 个（实际 3 个就够了，后面的留在后面扩充？）。这些 32 位的寄存器中每一位对应了一个中断通道相应的标志。

比如地址在 0xE000E100 的 ISER[0]这个 32 位的寄存器，第 0 位是中断通道 0 的允许位，第 1 位是中断通道 1 的允许标志……第 31 位是中断通道 31 的允许位；接下来地址在 0xE000E104 的 ISER[1]则是中断通道 32-63 的允许位。ICER、ISPR、ICPR、IABR 的结构相同，只是含义不同。

注意是对这些寄存器的操作：写 1 表示置位或清除，写 0 无任何影响。

例如：对地址在 0xE000E100 的 ISER[0]的第 0 位写 1，表示允许中断通道 0 中断；

但对 0xE000E100 的 ISER[0]的第 0 位写 0，则没有任何作用，该位保持不变。

如果要禁止中断通道 0 的中断响应，那么就必须：

对地址 0xE000E180 的 ICER[0]的第 0 位写 1，表示禁止中断通道 0 的中断；

对 0xE000E180 的 ICER[0]的第 0 位写 0，也是不起任何作用的。

b/ IP[240]用于定义 240 个外部中断通道的优先级，每 1 个字节对应一个中断通道。4 个中断通道的 IP[n]字构成一个 32 位的寄存器。在 STM32 中最多有 68 个外部中断通道，每个 IP[n]的 1 个字节中只使用高 4 位（见前面介绍）。IP[n]的结构如下：

	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	
E000E400	PIR_3		PIR_2		PIR_1		PIR_0		每 8 位的高 4 位有效，灰色位表示无效
E000E404	PIR_7		PIR_6		PIR_5		PIR_4		
……	……		……		……		……		

c/ 在 ST 公司提供的函数库的头文件中另一个数据结构中，还有一个重要的 32 位寄存器需要关注 ：AIRCR

```
/* memory mapping struct for System Control Block */
typedef struct
{
__I uint32_t CPUID;      /*!<CPU ID Base Register */
__IO uint32_t ICSR;      /*!< Interrupt Control State Register */
__IO uint32_t VTOR;      /*!< Vector Table Offset Register  */
__IO uint32_t AIRCR;     /*!< Application Interrupt / Reset Control Register */
__IO uint32_t SCR;       /*!< System Control Register */
__IO uint32_t CCR;       /*!< Configuration Control Register */
__IO uint8_t  SHP[12];   /*!<System Handlers Priority Registers(4-7,8-11,12-15) */
__IO uint32_t SHCSR;     /*!< System Handler Control and State Register */
__IO uint32_t CFSR;      /*!< Configurable Fault Status Register */
__IO uint32_t HFSR;      /*!< Hard Fault Status Register */
__IO uint32_t DFSR;      /*!< Debug Fault Status Register */
__IO uint32_t MMFAR;     /*!< Mem Manage Address Register */
__IO uint32_t BFAR;      /*!< Bus Fault Address Register  */
__IO uint32_t AFSR;      /*!< Auxiliary Fault Status Register */
__I  uint32_t PFR[2];    /*!< Processor Feature Register  */
__I  uint32_t DFR;       /*!< Debug Feature Register */
__I  uint32_t ADR;       /*!< Auxiliary Feature Register */
__I  uint32_t MMFR[4];   /*!< Memory Model Feature Register */
__I  uint32_t ISAR[5];   /*!< ISA Feature Register */
} SCB_Type;
```

它就是地址在 0xE000ED0C 的 32 位寄存器 AIRCR（Application Interrupt/Reset Control Register），该寄存器的[10:8]3 位就是 PRIGROUP 的定义位，它的值规定了系统中有多少个抢先级中断和子优先级中断。而 STM32 只使用高 4 位 bits，其可能的值如下（来自 ST 的函数库头文件中的定义）

```
#define NVIC_PriorityGroup_0         ((uint32_t)0x700) /*!< 0 bits for pre-emption priority
                                                            4 bits for subpriority */
#define NVIC_PriorityGroup_1         ((uint32_t)0x600) /*!< 1 bits for pre-emption priority
                                                            3 bits for subpriority */
#define NVIC_PriorityGroup_2         ((uint32_t)0x500) /*!< 2 bits for pre-emption priority
                                                            2 bits for subpriority */
#define NVIC_PriorityGroup_3         ((uint32_t)0x400) /*!< 3 bits for pre-emption priority
                                                            1 bits for subpriority */
#define NVIC_PriorityGroup_4         ((uint32_t)0x300) /*!< 4 bits for pre-emption priority
                                                            0 bits for subpriority */
```

由于这个寄存器相当重要，为了防止误操作（写），因此当改写这个寄存器的内容时，

必须要同时向这个寄存器的高 16 位[31：16]写验证字（Register key） 0x05FA。

例如：SBC->AIRCR |= （0x05FA0000 || 0x300）；　　// 设置系统中断有 16 个抢先优先
　　// 级，无子优先级

d/ 下面的定义与SYSTICK相关，有时也会用到的。

```
/* memory mapping struct for SysTick */
typedef struct
{
  __IO uint32_t CTRL;        /*!< SysTick Control and Status Register */
  __IO uint32_t LOAD;        /*!< SysTick Reload Value Register       */
  __IO uint32_t VAL;         /*!< SysTick Current Value Register      */
  __I  uint32_t CALIB;       /*!< SysTick Calibration Register        */
} SysTick_Type;
```

e/ 另外的几个寄存器，也是需要使用的（请具体参考相关的资料）

```
__IO uint8_t  SHP[12];  /*!<System Handlers Priority Registers(4-7,8-11,12-15) */
```

同每个外部中断通道优先级定义字相同,它们是内核中断通道4-15的优先级定义字所在的寄存器。用户可以通过设置SHP[n]，改变内部中断通道的优先级。

```
__IO uint32_t VTOR;     /*!< Vector Table Offset Register  */
```

如果你的代码要在RAM中启动执行，就需要对这个寄存器进行设置。