STM32 的 NVIC 优先级

xudian 0907

第一篇 优先级的表达方法

由于 STM32F10xxx Reference Munual 上面对 NVIC 的介绍并不多，所以最主要的参考资料就是 Cortex-M3 TRM 了。最近看了一篇网友写的 STM32 中断与嵌套 NVIC 快速入门，觉得有些地方的理解与作者有出入，因此也谈谈俺对 STM32 的 NVIC 的一点看法。
首先是 STM32F10xxx Reference Munual 手册上在第六章中断和事件开头说：

特性

- 43 个可屏蔽中断通道（不包含 16 个 Cortex™-M3 的中断线）；
- 16 个可编程的优先等级(使用了 4 位中断优先级)；

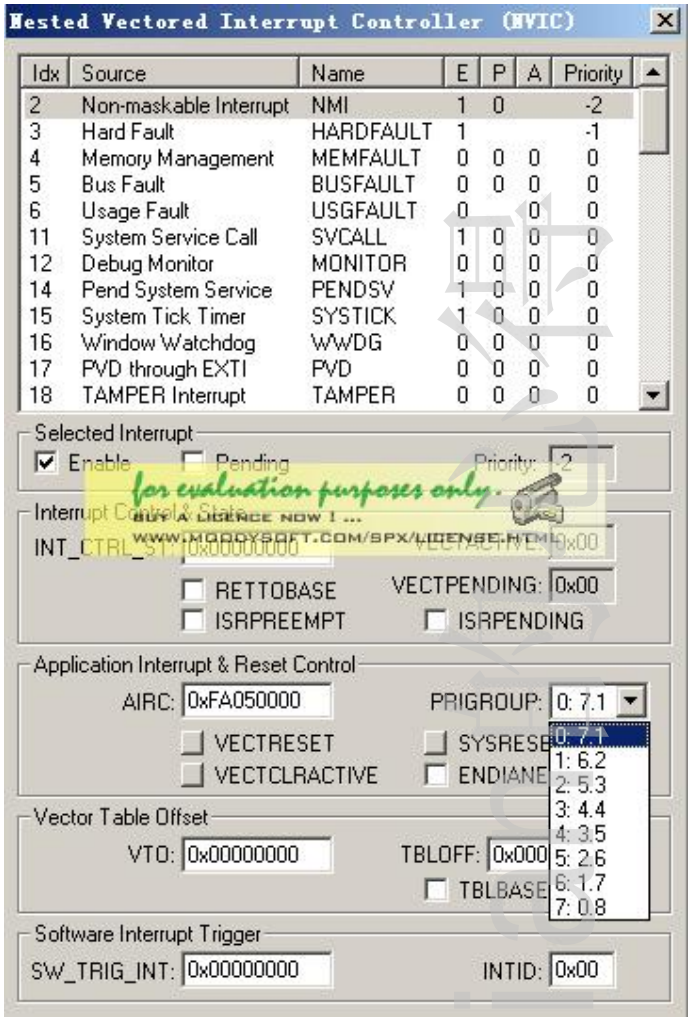
那么这 16 个可编程的优先等级、4 位的中断优先等级应该如何理解呢？
在 NVIC 快速入门里面，作者这么介绍：
说 STM32 里有 64 级可编程优先等级和 8 级中断嵌套。对于“64 级可编程优先等级”，我不知作者是如何获得这信息的，而参考 STM32F10xxx Reference Munual 上面就用文字表示有 16 个可编程的优先等级、4 位的中断优先等级。而 NVIC 快速入门里面，作者对这 64 级是这么理解的：
对 64 级中断就是说：（ INT0 到 INT63）这个大家比较好理解，其它的 64 ••• 239 就不用了。
IRQ CHANAEL 0
。。。
IRQ CHANAEL 63

其实上，STM32 里有 64 个中断（包含异常和外部中断 IRQ），除了 3 个 CM3 固定使用的异常外，每个中断（包含异常和外部中断 IRQ）都可以使用的优先级只有 16 级。

在 Cortex-M3 TRM 里有写道，原则上，CM3 支持 3 个固定的高优先级和多达 256 级的可编程优先级，并且支持 128 级抢占。在 NVIC 快速入门里面，作者却写：由于 CM3 支持硬件中断嵌套，所以可以有 256 级的可编程优先级和 256 级中断嵌套【 书上称：抢占（preempt）优先级】。
估计是作者笔误。因为 CM3 对 256 个优先级做了分组，分为抢占优先级和子优先级，并做了规定：子优先级至少占一个位！所以抢占优先级最多也就只有 7 个位即 128 级。

而 8 级中断嵌套这又是何解呢？
在 NVIC 快速入门里面，作者说道：
是这样的，上面说 一个 【中断】对应 一个【中断优先级寄存器】，而这个寄存器是 8 位的。当然就是 256 级了。而现在就用了 它其中的 BIT7,IT6,BIT5 三位来表示，而且是 MSB 对齐的。用了 3 个位来表达优先级(MSB 对齐的我们能够使用的 8 个优先级为：0x00（最高），0x20，0x40，0x60，0x80，0xA0，0xC0 以及 0xE0。)
这样我们在【中断优先级寄存器】就不能按理论的填 0 到 255 之间的数了，
而只能填 0x00（最高），0x20，0x40，0x60，0x80，0xA0，0xC0 以及 0xE0。)

其实作者这里弄错了，**8** 级中断嵌套应该为 **NVIC->AIRC** 里的 **BIT[10..8]**“优先级分组（**PRIGROUP**）”这三个位的值，也就是说优先级（包括抢占和子优先级）可以分 **8** 组（注意 **8** 组不等于只有 **8** 个优先级）。请看 KEIL 里对 NVIC->AIRC 的设置：



看到这里也许你还有点乱，再看下面对 PRIGROP 和优先级寄存器的配置你就会明白了：

表 1_0：

分组位置	表达抢占优先级的位段	表达子优先级的位段
0	[7:1]	[0:0]
1	[7:2]	[1:0]
2	[7:3]	[2:0]
3	[7:4]	[3:0]
4	[7:5]	[4:0]
5	[7:6]	[5:0]
6	[7:7]	[6:0]
7	--无--	[7:0]

例如：如果用 3 个位来表达优先级，则可以有 8 个优先级（包含抢占优先级和子优先级）按照 MSB 对齐的原则，[7：5]被用来表示这 8 个优先级。

如果用分组（7）来分配这 8 个优先级的话则[7：5]表示子优先级。

表 1_1：

用 3 位来表达优先级，采用分组 7 的情况	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	子优先级			未使用				

如果用分组（6）来分配这 8 个优先级的话则[7]表示抢占优先级，[6：5]表示子优先级。

表 1_2：

用 3 位来表达优先级，采用分组 6 的情况	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	抢占优先级	子优先级		未使用				

如果用分组（5）来分配这 8 个优先级的话则[7：6]表示抢占优先级，而[5]则表示子优先级

表 1_3：

用 3 位来表达优先级，采用分组 5 的情况	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	抢占优先级		子优先级	未使用				

如果用分组（4—0）来分配这 8 个优先级的话则[7：5]都表示抢占优先级

表 1_4：

用 3 位来表达优先级，采用分组 4 的情况	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	抢占优先级			未使用				

到这里已经很清晰了，其实 STM32 用 4 位来表示优先级，则有 16 个优先级。如下：

在 NVIC.H 文件中有

```
/* Preemption Priority Group ------------------------------------------------*/
#define NVIC_PriorityGroup_0         ((u32)0x700) /* 0 bits for pre-emption priority
                                                     4 bits for subpriority */
#define NVIC_PriorityGroup_1         ((u32)0x600) /* 1 bits for pre-emption priority
                                                     3 bits for subpriority */
#define NVIC_PriorityGroup_2         ((u32)0x500) /* 2 bits for pre-emption priority
                                                     2 bits for subpriority */
#define NVIC_PriorityGroup_3         ((u32)0x400) /* 3 bits for pre-emption priority
                                                     1 bits for subpriority */
#define NVIC_PriorityGroup_4         ((u32)0x300) /* 4 bits for pre-emption priority
                                                     0 bits for subpriority */
```

注意：NVIC.H 里这个定义的顺序是跟 CM3 里 PRIGROUP 的顺序刚好相反的，就是说

#define NVIC_PriorityGroup_0 ——对应 PRIGROUP 7

#define NVIC_PriorityGroup_1 ——对应 PRIGROUP 6

#define NVIC_PriorityGroup_2 ——对应 PRIGROUP 5

#define NVIC_PriorityGroup_3 ——对应 PRIGROUP 4

#define NVIC_PriorityGroup_4 ——对应 PRIGROUP 3

下面的分析将按照CM3实际的PRIGROUP进行，而不是NVIC.H里定义的**NVIC_PriorityGroup_x**。

分组（3）则16个优先级都表示抢占优先级

表1_5：

分组寄存器中[10:8]的配置情况：
采用分组 3 的情况

PRIGROUP	BIT10	BIT9	BIT8
3	0	1	1

优先级寄存器的配置情况：
采用分组 3 的情况

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
抢占优先级				未使用			

分组（4）则16个优先级表现为8个抢占优先级，每个抢占优先级里包含2个子优先级

表1_6：

分组寄存器中[10:8]的配置情况：
采用分组 4 的情况

PRIGROUP	BIT10	BIT9	BIT8
4	1	0	0

优先级寄存器的配置情况：
采用分组 4 的情况

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
抢占优先级			子优先级	未使用			

分组（5）则16个优先级表现为4个抢占优先级，每个抢占优先级里包含4个子优先级

表1_7：

分组寄存器中[10:8]的配置情况：
采用分组 5 的情况

PRIGROUP	BIT10	BIT9	BIT8
5	1	0	1

优先级寄存器的配置情况：
采用分组 5 的情况

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
抢占优先级		子优先级		未使用			

分组（6）则16个优先级表现为2个抢占优先级，每个抢占优先级里包含8个子优先级

表1_8：

分组寄存器中[10:8]的配置情况：
采用分组 6 的情况

PRIGROUP	BIT10	BIT9	BIT8
6	1	1	0

优先级寄存器的配置情况：
采用分组 6 的情况

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
抢优	子优先级			未使用			

分组（7）则16个优先级表示子优先级

表1_9：

分组寄存器中[10:8]的配置情况：
采用分组 7 的情况

PRIGROUP	BIT10	BIT9	BIT8
7	1	1	1

优先级寄存器的配置情况：
采用分组 7 的情况

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
子优先级				未使用			

STM32的64个中断，除了那3个CM3固定的异常外，每个中断都可以分配优先级，如果有N个中断的优先级相同则按中断号最低的那个最先等到响应。

为了判断你使用的芯片到底使用了多少位来表达优先级，可以使用下面的方法：往某个优先级寄存器里写入0XFF，再读回来。则从MSB开始，有多少位是1就有多少位表达优先级，最少要使用3个位。

第二篇 优先级的实现方法

以下红色字体为 NVIC_INIT 这个函数的内容以及一位网友对这些内容的解释。在此对这位网友表示感谢！

NVIC_INIT:

tmppriority = (0x700 - (SCB->AIRC & (u32)0x700))>> 0x08;

//算出来得到的就是抢占优先级的位数占四位中的几位

tmppre = (0x4 - tmppriority);

//算出来得到的就是子优先级的位数占四位中的几位

例如：结合上面对 STM32 的分组我们可以得到：

分组（3）**0x300**：tmppriority = (0x700 – 0x300)>> 0x08=0x4; //抢占优先级的位数占满四位

tmppre = 0x4 – 0x4=0; //子优先级占零位

分组（4）**0x400**：tmppriority = (0x700 – 0x400)>> 0x08=0x3; //抢占优先级的位数占 3 位

tmppre = 0x4 – 0x3=1; //子优先级占 1 位

分组（5）**0x500**：tmppriority = (0x700 – 0x500)>> 0x08=0x2; //抢占优先级的位数占 2 位

tmppre = 0x4 – 0x2=2; //子优先级占 2 位

其他几种分组情况也相同就不一一列出了。

tmpsub = tmpsub >> tmppriority;//然后将四位掩码通过抢占优先级位数得到子优先级的掩码

例如：采用分组 5 则得到抢占优先级为 2，子优先级为 2。Tmpsub 为 4 位掩码初始值为 0xF；

tmpsub=0xf>>0x2=0x3;// 子优先级的掩码为 3。

表 2_0：

分组 GROUP	抢占优先级 tmppriority	子优先级 tmppre	子优先级掩码（二进制）tmpsub = tmpsub >> tmppriority;
3	4	0	0000
4	3	1	0001
5	2	2	0011
6	1	3	0111
7	0	4	1111

子优先级掩码有什么用？

继续看下面的程序就知道了。

tmppriority = (u32)NVIC_InitStruct->NVIC_IRQChannelPreemptionPriority << tmppre;

//把用户输入的优先级组别参数 结合 前面计算出来的子优先级的位数 通过 移位放置到抢占优先级相应的 bit 上。

例如：在上面例子采用分组 5 的前提下，我们选择

抢占优先级 NVIC_IRQChannelPreemptionPriority=1；

通过前面计算出来的子优先级的位数 tmppre =2；（注意不是子优先级=2）
那么：tmppriority =1<<0x2=0x4;也就是说优先级寄存器（8 位）中的高 4 位为 0100；

具体情况如下表所示：

（以 STM32 为例并且假设用了分组 5，由分组 5 可以看到抢占优先级和子优先级分别占两位，因此抢占优先级的值有 4 个。子优先级暂时先不管）

表 2_1：

抢占优先级	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	0	—	—				
01	0	1	—	—				
10	1	0	—	—				
11	1	1	—	—				

tmppriority |= NVIC_InitStruct->NVIC_IRQChannelSubPriority & tmpsub;
//先 与 上子优先级掩码得到单纯的子优先级
//然后 或 上子优先级得到了该中断通道 8 位的优先级数

例如：在上面例子采用分组 5 的前提下，我们选择
子优先级 NVIC_IRQChannelSubPriority=1；
结果：tmppriority =0x4|(1 & 0x3)=0x5;

先 与 上子优先级掩码得到单纯的子优先级：

表 2_2：

子优先级掩码	子优先级	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0011	00	—	—	0	0				
0011	01	—	—	0	1				
0011	10	—	—	1	0				
0011	11	—	—	1	1				

然后 或 上子优先级得到了该中断通道 8 位的优先级数：

表 2_3：

抢占优先级	子优先级	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	00	0	0	0	0				
01	01	0	1	0	1				
10	10	1	0	1	0				
11	11	1	1	1	1				

以上就是掩码的作用了，接下来：

tmppriority = tmppriority << 0x04; //首先将这个优先级数左移四位放置到相应通道优先级寄存器 8 位的高四位（优先级 8 位从左边算起的）

例如：tmppriority=0x5<<0x4=0x50;

tmppriority = ((u32)tmppriority) << ((NVIC_InitStruct->NVIC_IRQChannel & (u8)0x03) * 0x08);
//因为一个 32 位中包含 4 个 8 位，而这些 32 位的优先级寄存器又排列在一块连

续的空间中组成优先级寄存器阵列，这样的话，需要有一个对准的问题，比如我现在是第六个通道，我就需要将其放置到第二个 32 位中的第二个八位（从左边数），具体算法是：

//首先将该通道在 32 位中的第几个八位算出来，所以同 0x03 相与，
//然后再乘以 8 得到的便是该通道在 32 位中的第几位，
//有 0*8=0，1*8=8，2*8=16，3*8=24，这四种情况
//再将该优先级左移这么多位就得到它在 32 位寄存器中的位置了。
//总之该语句将该通道优先级寄存在其32位当中从第几位开始的8位找出来
//放在 tmppriority 相应的地方中

例如 通道 6：tmppriority =0x50<<((0x6&0x3)*0x8)=0x500000;

表 2_4：优先级寄存器阵列

Bit [31..24]	Bit[23..16]	Bit[15..8]	Bit[7..0]	0xE000 E400
CHANNEL3	CHANNEL2	CHANNEL1	CHANNEL0	第 0 个 32 位
Bit [31..24]	Bit[23..16]	Bit[15..8]	Bit[7..0]	0xE000 E4004
CHANNEL7	CHANNEL6	CHANNEL5	CHANNEL4	第 1 个 32 位
Bit [31..24]	Bit[23..16]	Bit[15..8]	Bit[7..0]	0xE000 E4008
CHANNEL11	CHANNEL10	CHANNEL9	CHANNEL8	第 2 个 32 位

tmpreg = NVIC->Priority[(NVIC_InitStruct->NVIC_IRQChannel >> 0x02)];
//接下来将通道数除以四（这里通道移位来运算除法的），得到的其实就是该通道
//在第几个 32 位优先级寄存器上，把这个 32 位寄存结果读出来放在 tmpreg 中。
例如 通道 6：tmpreg=NVIC->Priority[1];//在第 1 个 32 位优先级寄存器上
tmpmask=(u32)0xFF<<((NVIC_InitStruct->NVIC_IRQChannel& (u8)0x03) * 0x08);
//该语句将该通道优先级寄存器在 32 位当中的第几位开始的八位加上了掩码，以
//便后面方便写入到该通道优先级寄存器的八位上
例如 tmpmask=0xff<<((0x6&0x03)*0x08)=0xff0000;
表 2_5：

0xE000 E4004	Bit [31..24]	Bit[23..16]	Bit[15..8]	Bit[7..0]
通道	CHANNEL7	CHANNEL6	CHANNEL5	CHANNEL4
掩码	0000	FF	00	00

tmpreg &= ~tmpmask;
//将该通道在 32 位的优先级寄存器组上的 8 位清零，其他位也就是其它通道的
//优先级寄存器保证同 1 相与，不会改变
例如 tmpreg&=0x00ffff;
//tmppriority &= tmpmask;
//只保留该通道的优先级寄存器在 32 位上的八位，其他的通道的优先级寄存
//器全部用 0 屏蔽
例如 tmppriority=0x500 00&0ff0000=0x500000;
tmpreg |= tmppriority;
//终于现在，tmpreg 中该通道优先级寄存器的 8 位全部为 0，其他通道优先级寄
//存器的 8 位（还有三组）全部为原来本身的数而没有改变
//tmppriority 中该通道优先级寄存器的 8 位数据完好，其他通道优先级寄存器的
//8 位（还有三组）全部为 0
//两者相"或"，该 32 位 tmpreg 便是已经包含了本次用户输入的通道中断优先
//级寄存器的 8 位信息

例如 tmpreg=0x500000|0x00fffff;
NVIC->Priority[(NVIC_InitStruct->NVIC_IRQChannel >> 0x02)] = tmpreg;
//输入到包含该通道中断子优先级寄存器的 32 位寄存器中，可以看出，本质上也是一个读--修改--写的过程