Appplication Note

Luminary 单片机

LM3S 系列微控制器中断优先级应用笔记

V1.00 Date: 2007/11/26 产品应用笔记

文件信息

类别	内容
关键词	中断、优先级、LM3S、Cortex-M3
摘要	对 LM3S 系列单片机的中断优先级的理解、编程及应用方法。

修订历史

版本	日期	原因
V1.00	2007/11/26	创建文档。

目　录

1. 中断优先级

● 中断概述

正常的程序被暂时中止，处理器便进入异常。所有异常可以通过 NVIC（嵌套向量中断控制器）进行控制，通过 NVIC 可以设置各个异常的优先等级并对异常进行处理。异常可分为系统异常和外部中断，它们通过不同的寄存器组进行控制（包括优先级的设置）。系统异常和外部中断及优先级关系如图 1 所示。

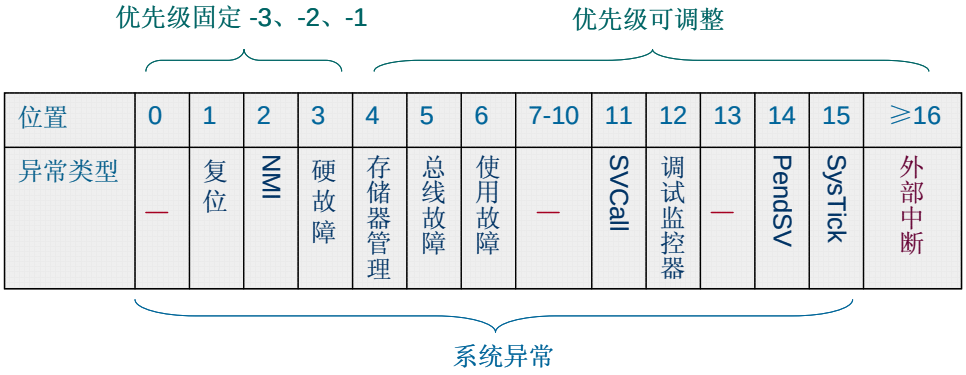

图 1 异常类型

图 1 中的位置号也即各个异常的中断号，处理器根据中断号从向量表中取出异常处理函数的入口（指针）。

● 中断方式

Cortex-M3 处理器以其灵活的异常机制在同类中脱颖而出，其异常可以通过占先、末尾连锁和迟来等处理来降低中断的延迟。

优点：ARM7TDMI-S 中断响应需要 24-42 个时钟，而 Cortex-M3 只需要 12 个时钟周期。

异常基于优先级而采取的动作主要有四种：占先、末尾连锁、返回和迟来，如表 1 所示。

表 1 异常基于优先级的动作

动作	描述
占先	产生条件：新的异常比当前的 ISR 或线程的优先级更高 发生时刻：ISR 或线程正在执行 中断结果：当前处于线程状态，则进入挂起中断；当前处于 ISR 状态，则产生中断嵌套 附加动作：处理器自动保存状态并压栈
末尾连锁	产生条件：新的异常优先级比当前正在返回的 ISR 的优先级更高 发生时刻：当前 ISR 结束时 中断结果：跳过出栈操作，将控制权转向新的 ISR
返回	产生条件：没有新的异常或没有比被压栈的 ISR 优先级更高的异常 发生时刻：当前 ISR 结束时 中断结果：执行出栈操作，并返回到被压栈的 ISR 或线程模式 附加动作：自动将处理器状态恢复为进入 ISR 之前的状态

迟来	产生条件：新的异常比正在保存状态的占先优先级更高 发生时刻：当前 ISR 开始时 中断结果：处理器转去处理优先级更高的中断

注：这四种方式最大的差别在于中断出现的时刻不同。

- 中断优先级

中断优先级使中断变得更加的灵活多变，它决定了处理器何时以及怎样处理异常，这样，对待中断就可以随心所欲了。

异常可分为系统异常和外部中断，那么异常优先级也可分为系统异常优先级和外部中断优先级。

所有的异常本身具有硬件优先级，其硬件优先级顺序决定于如图 1 中的位置号（中断号），中断号越低，硬件优先级越高。

也可以通过软件来设置异常的优先级，称为软件优先级。它只可以改变可调整优先级的异常，即除了复位、NMI 和硬件故障异常外，其它中断的优先级都可以通过寄存器配置。异常一旦指定软件优先级后，硬件优先级则无效。

注：软件优先级的设置对复位，NMI，和硬故障无效。它们的优先级始终比其他中断要高。复位（优先级-3），NMI（优先级-2），和硬故障（优先级-1）。

用户可设置的最高优先级为 0 号优先级，其仅次于复位，NMI 以及硬件故障的第 4 优先级。0 号优先级也是所有可调整优先级的默认优先级。如果有两个或更多的中断指定为相同的优先级（例如优先级全为 0），那么它们的硬件优先级（位置编号越低优先级越高）就决定了处理器激活这些中断的顺序。例如，如果 PendSV 和 SysTick 的优先级都为 0，那么 PendSV 的优先级更高。

系统异常与外部中断通过不同的寄存器组进行控制。外部中断的优先级可以通过外部中断优先级寄存器（NVIC_PRIn）进行配置，而系统异常的优先级可以通过系统处理器优先级寄存器（NVIC_SYS_PRIn）来配置，具体配置示例将在后续章节介绍。

2. 中断优先级分组

- 中断优先级分组描述

为了对具有大量中断的系统加强优先级控制，NVIC 支持优先级分组机制 。通过向应用中断和复位控制寄存器（NVIC_APINT）中的 PRIGROUP 区写不同的值来将异常分为占先优先级区和次优先级区，如表 2 所示。（寄存器地址为：0xE000ED0C）

表 2 应用中断与复位控制寄存器的位分配

域	名称	定义
[31:16]	VECTKEY	注册码（register key）。对寄存器进行写操作时要求在VECTKEY域中写入0x5FA。否则写入值被忽略。
[31:16]	VECTKEYSTAT	读取时为0xFA05
[15]	ENDIANESS	数据的字节顺序位： 1= 大端（高位在前） 0= 小端（低位在前）
[14:11]	-	保留
[10:8]	PRIGROUP	中断优先级分组域： PRIGROUP 从子优先级中拆分强占式优先级 0 7.1表示7位抢占式优先级，1位子优先级 1 6.2表示6位抢占式优先级，2位子优先级 2 5.3表示5位抢占式优先级，3位子优先级 3 4.4表示4位抢占式优先级，4位子优先级 4 3.5表示3位抢占式优先级，5位子优先级 5 2.6表示2位抢占式优先级，6位子优先级 6 1.7表示1位抢占式优先级，7位子优先级 7 0.8表示0位抢占式优先级，8位子优先级 PRIGROUP域是一个二进制小数点定位指示器，用于为共用同一抢占级别的异常创建优先级。它将中断优先级的PRI_n域分成抢占式优先级和子优先级。二进制小数点是一个偏左值。即PRIGROUP值代表一个从LSB左边开始的小数值。这是7:0的位0。 最低的值不能为0，这取决于为优先级分配的位数以及设备的选择
[7:3]	-	保留
[2]	SYSRESETREQ	让信号在外部系统有效，表示请求复位。
[1]	VECTCLRACTIVE	清除有效向量位： 1= 清除活动NMI、故障和中断的所有状态信息 0= 不清除
[0]	VECTRESET	系统复位位。将系统复位，调试元件除外： 1= 复位系统 0= 不复位系统

注：本表只详细介绍了该寄存器的 PRIGROUP 区[10:8]，其他位参见数据手册。

优先级包括组优先级和次优先级，组优先级又称占先优先级。如果有多个挂起异常共用

相同的组优先级，则需使用次优先级区来决定同组中的异常的优先级，这就是同组内的次优先级。如果两个挂起异常具有相同的组优先级和次优先级，则挂起异常的编号越低优先级越高。这与优先级机制是一致的。

注意：如果一个中断想抢占另一个正在处理的中断，则它的占先优先级（组优先级）必须比正在处理的中断的占先优先级要高。在组优先级相同，又存在多个次优先级都挂起时，可由次优先级来决定谁先执行，但只能产生末尾连锁。

- PRIGROUP 的优先级分组

向 PRIGROUP[2:0]写入不同值，可以到不同的分组结果，如表 3 所示。

表 3 PRIGROUP 的优先级分组

PRIGROUP[2:0]	中断优先级区，PRI_N[7:0]				
	二进制点的位置	占先区	次优先级区	占先优先级的数目	次优先级的数目
b000	bxxxxxxx.y	[7:1]	[0]	128	2
b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
b110	bx.yyyyyyy	[7]	[6:0]	2	128
b111	b.yyyyyyyy	无	[7:0]	0	256

注：Stellaris 系统处理器只使用 3 个位来配置优先级，即[7:5]。默认 PRIGROUP[2:0]为 0。

对于 Stellaris 系统处理器，使用[7:5]位来配置优先级，那么 PRIGROUP[2:0]的值 0~4 的效果是一样的，其占先优先级数最大为 8 个，所以该处理器的中断嵌入层数最大为 7 层。

PRIGROUP[2:0]被配置为 7

不产生异常占先，3 位次优先级，可设置次优先级为 000、001、…、111。在异常处理时，如果有更高优先级的异常产生，只会进入末尾连锁处理。分组结果如图 2 所示。

PRIGROUP[2:0] =7

组优先级	次优先级 位[7:5]	无关位 位[4:0]
-	xxx	00000

图 2 PRIGROUP[2:0]被配置为 7

Stellaris 系统处理器只使用 PRI_N [7:5]位来配置优先级，PRIGROUP[2:0]被配置为 7 时，优先级的高 3 位设置如表 4 所示。编号越低，优先级越高，如：次优先级 2 高于次优先级 4。

表 4 PRIGROUP[2:0]被配置为 7 的优先级设置

PRI_N [7:5]	PRI_N [4:0]	优先级描述
000	无效位	次优先级 0
001	无效位	次优先级 1
010	无效位	次优先级 2
011	无效位	次优先级 3
100	无效位	次优先级 4

101	无效位	次优先级 5
110	无效位	次优先级 6
111	无效位	次优先级 7

PRIGROUP[2:0]被配置为 6

1 位组优先级，2 位次优先级。可设置组优先级为 0、1，次优先级为 00、01、10、11。也即只有优先级被配置为 0xx0 0000 的异常，占先优先级被配置为 1xx0 0000 的中断。分组结果如图 3 所示。编号越低，优先级越高，如：组优先级 0 可占先组优先级 1；次优先级 1 高于次优先级 2。

PRIGROUP[2:0] =6

组优先级 位[7]	次优先级 位[6:5]	无关位 位[4:0]
x	xx	00000

图 3 PRIGROUP[2:0]被配置为 6

Stellaris 系统处理器只使用 PRI_N [7:5]位来配置优先级，PRIGROUP[2:0]被配置为 6 时，优先级的高 3 位设置如表 5 所示。

表 5 PRIGROUP[2:0]被配置为 6 的优先级设置

PRI_N [7]	PRI_N [6:5]	PRI_N [4:0]	优先级描述
0	00	无效位	组优先级 0，次优先级 0
0	01	无效位	组优先级 0，次优先级 1
0	10	无效位	组优先级 0，次优先级 2
0	11	无效位	组优先级 0，次优先级 3
1	00	无效位	组优先级 1，次优先级 0
1	01	无效位	组优先级 1，次优先级 1
1	10	无效位	组优先级 1，次优先级 2
1	11	无效位	组优先级 1，次优先级 3

PRIGROUP[2:0]被配置为 5

2 位组优先级，1 位次优先级，可设置组优先级为 00、01、10、11，次优先级为 0、1。通过优先级的高 2 位确定占先， PRI_N[7:6]的值越小其优先级越高。分组结果如图 4 所示。

PRIGROUP[2:0] =5

组优先级 位[7:6]	次优先级 位[5]	无关位 位[4:0]
xx	x	00000

图 4 PRIGROUP[2:0]被配置为 5

Stellaris 系统处理器只使用 PRI_N [7:5]位来配置优先级，PRIGROUP[2:0]被配置为 5 时，优先级的高 3 位设置如表 6 所示。如：组优先级 1 可占先组优先级 2。

表 6 PRIGROUP[2:0]被配置为 5 的优先级设置

PRI_N [7:6]	PRI_N [5]	PRI_N [4:0]	优先级描述

00	0	无效位	组优先级 0，次优先级 0
00	1	无效位	组优先级 0，次优先级 1
01	0	无效位	组优先级 1，次优先级 0
01	1	无效位	组优先级 1，次优先级 1
10	0	无效位	组优先级 2，次优先级 0
10	1	无效位	组优先级 2，次优先级 1
11	0	无效位	组优先级 3，次优先级 0
11	1	无效位	组优先级 3，次优先级 1

PRIGROUP[2:0]被配置为 0~4

优先级分组对于只使用 3 个位来配置优先级的处理器无效，全为组优先级。可设置组优先级为 000、001、…、111，分组结果如图 5 所示。

PRIGROUP[2:0] =0、1、2、3、4

组优先级　位[7:5]	次优先级	无关位　位[4:0]
xxx	-	00000

图 5　PRIGROUP[2:0]被配置为 0～4

Stellaris 系统处理器只使用 PRI_N [7:5]位来配置优先级，PRIGROUP[2:0]被配置为 0～4 时，优先级的高 3 位设置如表 7 所示。PRI_N[7:5]的值越小其优先级越高，如：组优先级 0 可占先组优先级 1～7，组优先级 6 可占先组优先级 7 等。

表 7　PRIGROUP[2:0]被配置为 0～4 的优先级设置

PRI_N [7:5]	PRI_N [4:0]	优先级描述
000	无效位	组优先级 0
001	无效位	组优先级 1
010	无效位	组优先级 2
011	无效位	组优先级 3
100	无效位	组优先级 4
101	无效位	组优先级 5
110	无效位	组优先级 6
111	无效位	组优先级 7

- 优先级分组设置示例

例如，设置成 2 位组优先级，1 位次优先级，则 PRIGROUP[2:0]被配置为 5，PRI_N[7:6]为组优先级，PRI_N[5] 为次优先级，对应优先级分组设置示例如程序清单 1 所示。对应用中断和复位控制寄存器执行写操作时，要求在 VECTKEY 域[31:16]中写入 0x5FA，否则写入值被忽略。

程序清单 1　优先级分组设置示例

```
#define NVIC_APINT              0xE000ED0C                  /*  应用中断与复位控制寄存器      */
#define HWREG(x)                (*((volatile unsigned long *)(x)))  /*  以字的方式访问寄存器          */
HWREG(0xE000ED0C) = (5 << 8) | (0x5fa << 16);               /*  2 位组优先级，1 位次优先级    */
```

3. 外部中断优先级

外部中断指的是除系统异常之外的异常，也即中断号等于和大于 16 的异常。外部中断的 0 号中断对应于 NVIC 的 16 号中断，依次类推，如图 6 所示。

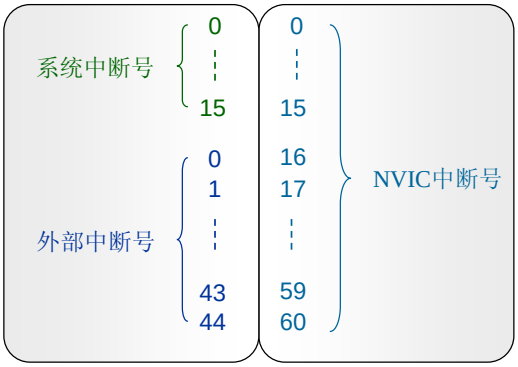

图 6 中断号对应关系

- 外部中断优先级描述

通过对中断优先级寄存器的 8 位 PRI_N 区执行写操作，来将中断的优先级指定为 0～255。各外部中断优先级寄存器 0-31 的位分配描述如图 7 所示。

注意：目前 Stellaris 系统处理器的外部中断只有 44 个，最新定义的外部中断、各外部中断号与 NVIC 中断号的详细对应关系请参见数据手册的中断部分，也可以关注流明诺瑞官方网站（www.luminarymicro.com）或周立功公司网站（www.zlgmcu.com）上的最新信息。

	31 24	23 16	15 8	7 0	
E000E400	PRI_3 GPIOD	PRI_2 GPIOC	PRI_1 GPIOB	PRI_0 GPIOA	外部中断0
E000E404	PRI_7 --	PRI_6 --	PRI_5 --	PRI_4 GPIOE	外部中断4
E000E408	PRI_11 --	PRI_10 --	PRI_9 --	PRI_8 --	
E000E40C	PRI_15 --	PRI_14 --	PRI_13 --	PRI_12 --	
E000E410	PRI_19 --	PRI_18 --	PRI_17 --	PRI_16 --	
E000E414	PRI_23 --	PRI_22 --	PRI_21 --	PRI_20 --	
E000E418	PRI_27 --	PRI_26 --	PRI_25 --	PRI_24 --	
E000E41C	PRI_31 --	PRI_30 --	PRI_29 --	PRI_28 --	
E000E420	PRI_35 --	PRI_34 --	PRI_33 --	PRI_32 --	
E000E424	PRI_39 --	PRI_38 --	PRI_37 --	PRI_36 --	外部中断36
E000E428	PRI_43 --	PRI_42 --	PRI_41 --	PRI_40 --	外部中断40

图 7 各外部中断优先级寄存器 0-31 的位分配

通过 11 个外部中断优先级寄存器来设置这 44 个外部中断的优先级，分别是 NVIC_PRI0（寄存器地址：0xE000E400）、NVIC_PRI1（寄存器地址：0xE000E404）、…、NVIC_PRI10（寄存器地址：0xE000E428），如图 7 所示。

每一个外部中断，都对应一个外部中断优先级寄存器中的 PRI_n，如：PRI_0 对应 GPIOA 中断（字节地址：E000E400），PRI_1 对应 GPIOB 中断（字节地址：E000E401），依此类推。

各外部中断的优先级设置位如表 8 所示。向相应的外部中断优先级寄存器写入优先级数值即可设置成相应的优先级。

表 8 外部中断的优先级设置位

域	名称	定义
[7:0]	PRI_n	外部中断n的优先级

注：其中，Stellaris 系列处理器就只有高三位（位[7:5]）有效。其占先优先级数最大为 8 个，所以该处理器的中断嵌入层数最大为 7 层。

- 外部中断优先级设置示例

例如，设置成 2 位组优先级，1 位次优先级；则 PRIGROUP[2:0]被配置为 5，PRI_N[7:6]为组优先级，PRI_N[5] 为次优先级。假设 GPIOA 组优先级为 2,次优先级为 0；GPIOB 组优先级为 2,次优先级为 1；GPIOC 组优先级为 1,次优先级为 1。外部中断优先级设置示例如程序清单 2 所示。

程序清单 2 外部中断优先级设置示例

```
#define HWREG(x)        (*((volatile unsigned long *)(x)))
#define HWREGB(x)       (*((volatile unsigned long *)(x)))

#define NVIC_APINT      0xE000ED0C      /*  应用中断和复位控制寄存器        */
#define PRI_0           0xE000E400      /*  GPIOA(外部 0 号)中断优先级地址   */
#define PRI_1           0xE000E401      /*  GPIOB(外部 1 号)中断优先级地址   */
#define PRI_2           0xE000E402      /*  GPIOC(外部 2 号)中断优先级地址   */

HWREG(NVIC_APINT) = (5 << 8) | (0x5FA << 16);   /*  2 位组优先级，1 位次优先级   */

HWREGB(PRI_0) |= 2 << 6 | 0 << 5;       /*  GPIOA 组优先级为 2,次优先级为 0  */
HWREGB(PRI_1) |= 2 << 6 | 1 << 5;       /*  GPIOB 组优先级为 2,次优先级为 1  */
HWREGB(PRI_2) |= 1 << 6 | 1 << 5;       /*  GPIOC 组优先级为 1,次优先级为 1  */
```

4. 系统处理器优先级

外部中断以外的异常称为系统异常，系统异常与外部中断通过不同的寄存器组进行控制，系统异常对应于 NVIC 的前 16 号中断。除了复位、NMI 和硬件故障异常外，其它中断的优先级都可以通过寄存器配置的，具体信息如图 1 和图 6 所示。

- 系统处理器优先级描述

可设置优先级的系统处理器异常有：存储器管理、总线故障、使用故障、调试监控、SVC（软件中断）、PendSV（系统服务请求）、SysTick(系统节拍定时器)。

通过 3 个处理器优先级寄存器设置以上系统异常的优先级，即 NVIC_SYS_PRI1（寄存器地址：0xE000ED18）、NVIC_SYS_PRI2（寄存器地址：0xE000E41C）、NVIC_SYS_PRI3（寄存器地址：0xE000E420）。

系统处理器优先级寄存器的位分配描述如图 8 所示。

注：Stellaris 系列处理器对系统处理器异常同样只有高三位（位[7:5]）有效。其占先优先级数最大为 8 个，中断嵌入层数最大为 7 层。

	31 24		23 16		15 8		7 0	
E000ED18	PRI_7	保留	PRI_6	使用故障	PRI_5	总线故障	PRI_4	存储器管理
E000ED1C	PRI_11	SVCall	PRI_10	保留	PRI_9	保留	PRI_8	保留
E000ED20	PRI_15	SysTick	PRI_14	PendSV	PRI_13	保留	PRI_12	调试监控

图 8 系统处理器优先级寄存器的位分配

每一个系统异常，都对应一个系统异常优先级寄存器中的 PRI_n，如：PRI_4 对应存储器管理（字节地址：E000ED18），PRI_5 对应总线故障（字节地址：E000ED19），其它的系统异常对应如图 8 所示。

其中 PRI_0 为保留，PRI_1、 PRI_2、 PRI_3 分别对应于优先级不可改变的复位、NMI 和硬件故障异常。

- 系统处理器异常设置示例

例如，设置成 2 位组优先级，1 位次优先级；则 PRIGROUP[2:0]被配置为 5，PRI_N[7:6]为组优先级，PRI_N[5] 为次优先级。假设 SysTick 系统中断优先级为 3，PendSV 系统中断优先级为 2，系统处理器异常设置示例如程序清单 3 所示。

程序清单 3 系统处理器异常设置示例

```
#define HWREG(x)        (*((volatile unsigned long *)(x)))      /*  以字的方式访问内存      */
#define HWREGB(x)       (*((volatile unsigned long *)(x)))      /*  以字节方式访问内存      */

#define NVIC_APINT      0xE000ED0C                              /*  应用中断和复位控制寄存器  */
#define PRI_15          0xE000ED23                              /*  PendSV 系统中断优先级地址 */
#define PRI_14          0xE000ED22                              /*  SysTick 系统中断优先级地址 */

HWREG(NVIC_APINT) = (5 << 8) | (0x5FA << 16);                   /*  2 位组优先级，1 位次优先级 */
```

```
HWREGB(PRI_15) |= 150;                  /*  SysTick 系统中断优先级为 2     */
HWREGB(PRI_14) |= 100;                  /*  PendSV  系统中断优先级为 1     */
```

该程序中，HWREGB(PRI_15) |= 150，也即 10.010110，组优先级[7:6]为 11，中断优先级为 2。HWREGB(PRI_14) |= 100，也即 01.100100，组优先级[7:6]为 10，中断优先级为 1。

- 触发系统处理器异常示例

系统处理器异常的触发由中断控制状态寄存器（NVIC_INT_CTRL）设置，向相应位设置为 1 即可触发该异常，要清楚各中断标志也向相应位写 1 即可。但不是所有系统异常都可以软件触发，这里只有 SysTick 系统中断和 PendSV 系统中断可以软件触发，如表 9 所示，该寄存器其它位这里不给出，该寄存器详细的描述参见各数据手册。

表 9　中断控制状态寄存器的位分配

域	名称	类型	定义
[31]	NMIPENDSET	读/写	设置挂起（pending）NMI位
[30:29]	-	-	保留
[28]	PENDSVSET	读/写	设置挂起pendSV位： 1= 设置挂起pendSV 0= 不设置挂起pendSV
[27]	PENDSVCLR	只写	清除挂起pendSV位 1= 清除挂起pendSV 0= 不清除挂起pendSV
[26]	PENDSTSET	读/写	设置挂起SysTick位 1= 设置挂起SysTick 0= 不设置挂起SysTick
[25]	PENDSTCLR	只写	清除挂起SysTick位 1= 清除挂起SysTick 0= 不清除挂起SysTick

触发系统处理器异常示例程序如程序清单 4 所示。

程序清单 4　触发系统处理器异常示例程序

```
#define HWREG(x)                (*((volatile unsigned long *)(x)))      /*  以字的方式访问内存     */

HWREG(NVIC_INT_CTRL) |= 1 << 26;        /*  触发 SysTick 系统中断          */
HWREG(NVIC_INT_CTRL) |= 1 << 28;        /*  触发 PendSV 系统中断           */
```

清除相应系统处理器异常标志位示例程序如程序清单 5 所示。

程序清单 5　触发系统处理器异常示例程序

```
#define HWREG(x)                (*((volatile unsigned long *)(x)))      /*  以字的方式访问内存     */

HWREG(NVIC_INT_CTRL) |= 1 << 25;        /*  触发 SysTick 系统中断          */
HWREG(NVIC_INT_CTRL) |= 1 << 27;        /*  触发 PendSV 系统中断           */
```

5. 中断优先级示例

- 示例程序描述

本示例程序基于 EasyARM8962 开发板，采用 CrossStudio 集成开放环境。采用 3 个按键 KEY1、KEY2、KEY3 控制 3 个 LED 灯来实现各个中断优先级的演示。

采用的中断资源和各中断的优先级设置如表 10 所示。

表 10 中断和各中断的优先级设置

中断	所用资源	中断来源	优先级设置情况
GPIOA 外部中断	KEY1、LED4	按键 KEY1	组优先级为 2,次优先级为 0
GPIOB 外部中断	KEY2、LED5	按键 KEY2	组优先级为 2,次优先级为 1
GPIOC 外部中断	KEY3、LED6	按键 KEY3	组优先级为 1,次优先级为 1
SysTick 系统异常	LED5、LED6	软件触发	中断优先级（组优先级）为 2
PendSV 系统异常	LED3、LED4	软件触发	中断优先级（组优先级）为 1

示例程序现象描述如表 11 所示。

表 11 示例程序现象及描述

动作	现象描述
对开发板复位	程序首先触发 SysTick 系统异常，LED5、LED6 同时闪烁一段时间，在 SysTick 系统异常中触发 PendSV 系统异常，由于 PendSV 系统异常优先级高于 SysTick 系统异常，所以 LED3、LED4 同时闪烁一段时间，然后返回到 SysTick 系统异常，LED5、LED6 同时继续闪烁一段时间，退出 SysTick 系统异常后，进入死循环，LED3 一直闪烁，等待外部中断。 注意：此过程外部中断也可以参与，GPIOC 外部中断可占先 SysTick 系统异常，与 PendSV 系统异常产生末尾连锁；GPIOA 和 GPIOB 外部中断不可占先 PendSV 系统异常，与 SysTick 系统异常只能产生末尾连锁。
按下 KEY1	进入 GPIOA 外部中断，LED4 闪烁一段时间，然后回到主程序 LED3 闪烁。 注意：此时可以被 GPIOC 外部中断占先，即按下 KEY3，LED6 闪烁一段时间，然后回到 GPIOA 外部中断。GPIOA 外部中断与 GPIOB 外部中断产生末尾连锁。
按下 KEY2	进入 GPIOB 外部中断，LED5 闪烁一段时间，然后回到主程序 LED3 闪烁。 注意：此时可以被 GPIOC 外部中断占先，即按下 KEY3，LED6 闪烁一段时间，然后回到 GPIOA 外部中断。GPIOB 外部中断与 GPIOA 外部中断产生末尾连锁。
按下 KEY3	进入 GPIOC 外部中断，LED6 闪烁一段时间，然后回到主程序 LED3 闪烁。 注意：此时不能被任何中断占先。与 GPIOA 外部中断、GPIOB 外部中断产生末尾连锁。如果在 GPIOC 外部中断时，GPIOA 外部中断和 GPIOB 外部中断都产生，则 GPIOA 外部中断先响应，因为 GPIOA 外部中断比 GPIOB 外部中断的次优先级要高。

- 示例程序清单

中断优先级示例程序如程序清单 6 所示。

程序清单 6 中断优先级示例程序

```
#define HWREG(x)          (*((volatile unsigned long *)(x)))
#define HWREGB(x)         (*((volatile unsigned long *)(x)))

#define SYSCTL_PERIPH_GPIOA   0x20000001    /*  GPIO A                      */
#define SYSCTL_PERIPH_GPIOB   0x20000002    /*  GPIO B                      */
#define SYSCTL_PERIPH_GPIOC   0x20000004    /*  GPIO C                      */
#define SYSCTL_RCGC2          0x400fe108    /*  运行模式时钟门控寄存器 2    */

#define GPIO_PORTA_BASE       0x40004000    /*  GPIO Port A                 */
#define GPIO_PORTB_BASE       0x40005000    /*  GPIO Port B                 */
#define GPIO_PORTC_BASE       0x40006000    /*  GPIO Port C                 */

#define GPIO_O_DIR            0x00000400    /*  数据方向寄存器              */
#define GPIO_O_AFSEL          0x00000420    /*  模式控制寄存器              */
#define GPIO_O_DATA           0x00000000    /*  数据寄存器                  */
#define GPIO_O_DEN            0x0000051C    /*  数字输入使能                */
#define GPIO_O_DR2R           0x00000500    /*  设置成 2mA 驱动             */

#define NVIC_EN0              0xE000E100    /*  中断使能设置寄存器          */
#define NVIC_PRI0             0xE000E400    /*  中断优先级寄存器            */
#define NVIC_APINT            0xE000ED0C    /*  应用中断和复位控制寄存器    */
#define GPIO_O_IS             0x00000404    /*  GPIO  中断检测寄存器        */
#define GPIO_O_RIS            0x00000414    /*  GPIO  中断事件寄存器        */
#define GPIO_O_IM             0x00000410    /*  GPIO  中断屏蔽寄存器        */
#define GPIO_O_ICR            0x0000041C    /*  GPIO  中断清除寄存器        */

#define NVIC_INT_CTRL         0xE000ED04    /*  中断控制状态寄存器          */
#define NVIC_SYS_PRI3         0xE000ED20    /*  系统优先级寄存器            */

#define KEY1                  0x00000010    /*  设置 PA4 口为 KEY1          */
#define KEY2                  0x00000010    /*  设置 PB4 口为 KEY2          */
#define KEY3                  0x00000010    /*  设置 PC4 口为 KEY3          */

#define LED3                  0x00000040    /*  设置 PB6 口为 LED3          */
#define LED4                  0x00000020    /*  设置 PA5 口为 LED4          */
#define LED5                  0x00000020    /*  设置 PB5 口为 LED5          */
#define LED6                  0x00000020    /*  设置 PC5 口为 LED6          */

#define PRI_0                 0xE000E400    /*  GPIOA(0 号)中断优先级地址   */
#define PRI_1                 0xE000E401    /*  GPIOB(0 号)中断优先级地址   */
#define PRI_2                 0xE000E402    /*  GPIOC(0 号)中断优先级地址   */
#define PRI_15                0xE000ED23    /*  PendSV 系统中断优先级地址   */
#define PRI_14                0xE000ED22    /*  SysTick 系统中断优先级地址  */
```

```
/*********************************************************************************************
** Function name:           CPUcpsie
** Descriptions:            使能全局中断
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    CPUcpsie(void)
{
    __asm (
        "cpsie i \n"
        "bx lr \n"
    );
}

/*********************************************************************************************
** Function name:           delay
** Descriptions:            延时函数
** input parameters:        t，延时时间常数
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void delay (int    t)
{
    for ( ; t; --t);
}

/*********************************************************************************************
** Function name:           GPIO_Port_A_ISR
** Descriptions:            GPIOA 中断函数
**                          用 KEIL 软件时，在 Startup.S 中添加该中断函数名
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    GPIO_Port_A_ISR (void)
{
    unsigned long i = 0;
    HWREG(GPIO_PORTA_BASE + GPIO_O_ICR) |= KEY1;      /*    清除中断标志                    */
    for ( i = 20; i > 0; i--) {
        HWREG(GPIO_PORTA_BASE + (GPIO_O_DATA + (LED4 << 2))) = ~LED4;
                                                      /*    点亮 LED4                       */
        delay(200000);
        HWREG(GPIO_PORTA_BASE + (GPIO_O_DATA + (LED4 << 2))) = LED4;
```

```
                                                    /*  熄灭 LED4                   */
            delay(200000);
        }
}

/*********************************************************************************************
** Function name:           GPIO_Port_B_ISR
** Descriptions:            GPIOB 中断函数
**                          用 KEIL 软件时，在 Startup.S 中添加该中断函数名
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    GPIO_Port_B_ISR (void)
{
        unsigned long i = 0;
        HWREG(GPIO_PORTB_BASE + GPIO_O_ICR) |= KEY2;     /*  清除中断标志                */
        for ( i = 20; i > 0; i--) {
            HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = ~LED5;
                                                    /*  点亮 LED5                   */
            delay(200000);
            HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = LED5;
                                                    /*  熄灭 LED5                   */
            delay(200000);
        }
}

/*********************************************************************************************
** Function name:           GPIO_Port_C_ISR
** Descriptions:            GPIOA 中断函数
**                          用 KEIL 软件时，在 Startup.S 中添加该中断函数名
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    GPIO_Port_C_ISR (void)
{
        unsigned long i = 0;
        HWREG(GPIO_PORTC_BASE + GPIO_O_ICR) |= KEY3;     /*  清除中断标志                */
        for ( i = 20; i > 0; i--) {
            HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = ~LED6;
                                                    /*  点亮 LED6                   */
            delay(200000);
            HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = LED6;
```

```
                                                    /*  熄灭 LED6                   */
            delay(200000);
        }
}

/*********************************************************************************************
** Function name:           SysTick_ISR
** Descriptions:            SysTick 中断函数
**                          用 KEIL 软件时，在 Startup.S 中添加该中断函数名
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    SysTick_ISR (void)
{
    unsigned long i = 0;
    HWREG(NVIC_INT_CTRL) |= 1 << 25;                /*  清除中断标志                */
    for ( i = 10; i > 0; i--) {
        HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = ~LED6;
                                                    /*  点亮 LED6                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = ~LED5;
                                                    /*  点亮 LED5                   */
        delay(200000);
        HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = LED6;
                                                    /*  熄灭 LED6                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = LED5;
                                                    /*  熄灭 LED5                   */
        delay(200000);
    }

    HWREG(NVIC_INT_CTRL) |= 1 << 28;                /*  触发 PendSV 系统中断        */

    for ( i = 10; i > 0; i--) {
        HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = ~LED6;
                                                    /*  点亮 LED6                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = ~LED5;
                                                    /*  点亮 LED5                   */
        delay(200000);
        HWREG(GPIO_PORTC_BASE + (GPIO_O_DATA + (LED6 << 2))) = LED6;
                                                    /*  熄灭 LED6                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED5 << 2))) = LED5;
                                                    /*  熄灭 LED5                   */
        delay(200000);
    }
```

```
}

/*********************************************************************************************
** Function name:           PendSV_ISR
** Descriptions:            PendSV 中断函数
**                          用 KEIL 软件时，在 Startup.S 中添加该中断函数名
** input parameters:        无
** output parameters:       无
** Returned value:          无
*********************************************************************************************/
void    PendSV_ISR (void)
{
    unsigned long i = 0;
    HWREG(NVIC_INT_CTRL) |= 1 << 27;                    /*  清除中断标志                */
    for ( i = 20; i > 0; i--) {
        HWREG(GPIO_PORTA_BASE + (GPIO_O_DATA + (LED4 << 2))) = ~LED4;
                                                        /*  点亮 LED4                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED3 << 2))) = ~LED3;
                                                        /*  点亮 LED3                   */
        delay(200000);
        HWREG(GPIO_PORTA_BASE + (GPIO_O_DATA + (LED4 << 2))) = LED4;
                                                        /*  熄灭 LED4                   */
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED3 << 2))) = LED3;
                                                        /*  熄灭 LED3                   */
        delay(200000);
    }
}

/*********************************************************************************************
** Function name:           main
** Descriptions:            主函数
**                          程序运行时，进入 SysTick 系统中断，LED3、LED4 闪烁，在 SysTick 系统中
**                          断中启动了优先级
**                          更高的 PendSV 系统中断，出现 LED5、LED6 闪烁，退出 PendSV 系统中断后
**                          回到 SysTick 系统中断，退出 SysTick 系统中断后回到主程序，LED3 一直闪烁。
**                          然后：
**                          KEY1--GPIOA 中断 ，对应于 LED4，设置成组优先级为 2，次优先级为 0
**                          KEY2--GPIOB 中断 ，对应于 LED5，设置成组优先级为 2，次优先级为 1
**                          KEY3--GPIOC 中断 ，对应于 LED6，设置成组优先级为 1，次优先级为 1
**                          按下 KEY1，LED4 闪烁；按下 KEY2，LED5 闪烁；按下 KEY3，LED6 闪烁；
**                          KEY1 与 KEY2 产生末尾连锁，KEY3 可抢断 KEY1、KEY2，通过 LED 可以
**                          观测结果
** input parameters:        无
** output parameters:       无
```

```
** Returned value:            无
*******************************************************************************************/
int main (void)
{
    unsigned char    i;
    HWREG(SYSCTL_RCGC2) |= SYSCTL_PERIPH_GPIOA & 0x0fffffff;
                                                        /*   使能 GPIO PA 口外设          */
    HWREG(SYSCTL_RCGC2) |= SYSCTL_PERIPH_GPIOB & 0x0fffffff;
                                                        /*   使能 GPIO PB 口外设          */
    HWREG(SYSCTL_RCGC2) |= SYSCTL_PERIPH_GPIOC & 0x0fffffff;
                                                        /*   使能 GPIO PC 口外设          */

    for (i = 0; i < 4; i++);

    HWREG(GPIO_PORTA_BASE + GPIO_O_DIR) &= ~KEY1;
                                                        /*   设置 KEY1 为输入             */
    HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) &= ~KEY2;
                                                        /*   设置 KEY2 为输入             */
    HWREG(GPIO_PORTC_BASE + GPIO_O_DIR) &= ~KEY3;
                                                        /*   设置 KEY3 为输入             */

    HWREG(GPIO_PORTA_BASE + GPIO_O_AFSEL) &= ~KEY1;
                                                        /*   配置 KEY1 为 GPIO 功能       */
    HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) &= ~KEY2;
                                                        /*   配置 KEY2 为 GPIO 功能       */
    HWREG(GPIO_PORTC_BASE + GPIO_O_AFSEL) &= ~KEY3;
                                                        /*   配置 KEY3 为 GPIO 功能       */

    HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) |= LED3;        /*   设置 LED3 为输出             */
    HWREG(GPIO_PORTA_BASE + GPIO_O_DIR) |= LED4;        /*   设置 LED4 为输出             */
    HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) |= LED5;        /*   设置 LED5 为输出             */
    HWREG(GPIO_PORTC_BASE + GPIO_O_DIR) |= LED6;        /*   设置 LED6 为输出             */

    HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) &= ~LED3;
                                                        /*   配置 LED3 为 GPIO 功能       */
    HWREG(GPIO_PORTA_BASE + GPIO_O_AFSEL) &= ~LED4;
                                                        /*   配置 LED4 为 GPIO 功能       */
    HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) &= ~LED5;
                                                        /*   配置 LED5 为 GPIO 功能       */
    HWREG(GPIO_PORTC_BASE + GPIO_O_AFSEL) &= ~LED6;
                                                        /*   配置 LED6 为 GPIO 功能       */

    /*
     *   设置所有的 LED 和 KEY 为 2mA 驱动,并设为推挽管脚
```

```
 */
HWREG(GPIO_PORTA_BASE + GPIO_O_DR2R) = (HWREG(GPIO_PORTA_BASE +
                                       GPIO_O_DR2R) | KEY1 | LED4);
HWREG(GPIO_PORTB_BASE + GPIO_O_DR2R) = (HWREG(GPIO_PORTB_BASE +
                                       GPIO_O_DR2R) | KEY2 | LED3 | LED5);
HWREG(GPIO_PORTC_BASE + GPIO_O_DR2R) = (HWREG(GPIO_PORTC_BASE +
                                       GPIO_O_DR2R) | KEY3 | LED6);

HWREG(GPIO_PORTA_BASE + GPIO_O_DEN)  = (HWREG(GPIO_PORTA_BASE +
                                       GPIO_O_DEN) | KEY1 | LED4);
HWREG(GPIO_PORTB_BASE + GPIO_O_DEN)  = (HWREG(GPIO_PORTB_BASE +
                                       GPIO_O_DEN) | KEY2 | LED3 | LED5);
HWREG(GPIO_PORTC_BASE + GPIO_O_DEN)  = (HWREG(GPIO_PORTC_BASE +
                                       GPIO_O_DEN) | KEY3 | LED6);

HWREG(GPIO_PORTA_BASE + GPIO_O_IS)  |= ~KEY1;   /*  中断为边沿触发          */
HWREG(GPIO_PORTA_BASE + GPIO_O_RIS) &= ~KEY1;   /*  上升沿触发              */
HWREG(GPIO_PORTB_BASE + GPIO_O_IS)  |= ~KEY2;   /*  中断为边沿触发          */
HWREG(GPIO_PORTB_BASE + GPIO_O_RIS) &= ~KEY2;   /*  上升沿触发              */
HWREG(GPIO_PORTC_BASE + GPIO_O_IS)  |= ~KEY3;   /*  中断为边沿触发          */
HWREG(GPIO_PORTC_BASE + GPIO_O_RIS) &= ~KEY3;   /*  上升沿触发              */

CPUcpsie();

HWREG(GPIO_PORTA_BASE + GPIO_O_IM) |= KEY1;     /*  使能 KEY1 中断          */
HWREG(GPIO_PORTB_BASE + GPIO_O_IM) |= KEY2;     /*  使能 KEY2 中断          */
HWREG(GPIO_PORTC_BASE + GPIO_O_IM) |= KEY3;     /*  使能 KEY3 中断          */

HWREG(NVIC_EN0) |= 1 << 0;                      /*  使能 GPIOA 口中断(中断号为 0)  */
HWREG(NVIC_EN0) |= 1 << 1;                      /*  使能 GPIOB 口中断(中断号为 1)  */
HWREG(NVIC_EN0) |= 1 << 2;                      /*  使能 GPIOC 口中断(中断号为 2)  */

HWREG(NVIC_APINT) = (5 << 8) | (0x5FA << 16);   /*  组优先级为 4 位，次优先级 2 位  */

/*
 *  GPIOA 组优先级为 2,次优先级为 0;GPIOB 组优先级为 2,次优先级为 1;GPIOC 组优先级为 1,
 *  次优先级为 1.
 */
HWREGB(PRI_0) |= 2 << 6 | 0 << 5;
HWREGB(PRI_1) |= 2 << 6 | 1 << 5;
HWREGB(PRI_2) |= 1 << 6 | 0 << 5;

HWREGB(PRI_15) |= 150;                          /*  SysTick 系统中断优先级为 2      */
HWREGB(PRI_14) |= 100;                          /*  PendSV  系统中断优先级为 1      */
```

```
    HWREG(NVIC_INT_CTRL) |= 1 << 26;                    /*  触发 SysTick 系统中断          */

    while (1) {
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED3 << 2))) = ~LED3;
                                                        /*  点亮 LED3                      */
        delay(200000);
        HWREG(GPIO_PORTB_BASE + (GPIO_O_DATA + (LED3 << 2))) = LED3;
                                                        /*  熄灭 LED3                      */
        delay(200000);
    }
}
```

6. 中断优先级问题解决实例

在我们的 Luminary 客户群中，有部分客户对中断优先级了解得不够深入，遇到了不少问题，这里便总结出客户所遇到的一个典型问题，并作简要分析。

- 中断优先级问题

LM3S 系列单片机的优先级只有 8 个，占用 PRI_N 的[7:5]位，那么 PRI_N 的优先级及对应的优先级值如表 12 所示。

表 12 优先级及优先级值

PRI_N 优先级	PRI_N 优先级值
优先级 0	00000000
优先级 1	00100000
优先级 2	01000000
优先级 3	01100000
优先级 4	10000000
优先级 5	10100000
优先级 6	11000000
优先级 7	11100000

在 Luminary 驱动库的 hw_nvic.h 中定义了优先级的分组，如程序清单 7 所示。

程序清单 7 优先级的分组定义

```
#define    NVIC_APINT_PRIGROUP_7_1    0x00000000    /*  Priority group 7.1 split  */
#define    NVIC_APINT_PRIGROUP_6_2    0x00000100    /*  Priority group 6.2 split  */
#define    NVIC_APINT_PRIGROUP_5_3    0x00000200    /*  Priority group 5.3 split  */
#define    NVIC_APINT_PRIGROUP_4_4    0x00000300    /*  Priority group 4.4 split  */
#define    NVIC_APINT_PRIGROUP_3_5    0x00000400    /*  Priority group 3.5 split  */
#define    NVIC_APINT_PRIGROUP_2_6    0x00000500    /*  Priority group 2.6 split  */
#define    NVIC_APINT_PRIGROUP_1_7    0x00000600    /*  Priority group 1.7 split  */
#define    NVIC_APINT_PRIGROUP_0_8    0x00000700    /*  Priority group 0.8 split  */
```

现在提出问题是，如果按 NVIC_APINT_PRIGROUP_2_6 分组，那么以下分组的描述正确吗？

表 13 优先级描述

PRI_N 优先级	PRI_N 优先级值	优先级描述
优先级 0	00000000	低组优先级
优先级 1	00100000	低组优先级
优先级 2	01000000	高组优先级
优先级 3	01100000	高组优先级
优先级 4	10000000	高组优先级

优先级 5	10100000	高组优先级
优先级 6	11000000	高组优先级
优先级 7	11100000	高组优先级

按照这种描述，优先级 7 还高于优先级 0，并且能嵌套优先级 0?

- 中断优先级问题的解答

Luminary 单片机采用 3 个位来配置优先级，即[7：5]，而[4：0]位无效。

它最多可以配置 8 个优先级，即组优先级＋次优先级最多为 8 级。

按照问题所说的，假设优先级是按 NVIC_APINT_PRIGROUP_2_6 分组，则：

分组结果为：组优先级为 2 位[7：6]，可以设四个不同的组优先级；次优先级为 1 位[5]，可以设置两个不同次优先级；而[4：0]位对 Luminary 单片机无效。

分组后，组优先级只看位[7：6]，次优先级看位[5]，描述如表 14 所示。

表 14 按 NVIC_APINT_PRIGROUP_2_6 分组结果

PRI_N 优先级	PRI_N 优先级值			描述
	组优先级位	次优先级位	无效位	
优先级 0	00	0	00000	组优先级为 0，次优先级为 0
优先级 1	00	1	00000	组优先级为 0，次优先级为 1
优先级 2	01	0	00000	组优先级为 1，次优先级为 0
优先级 3	01	1	00000	组优先级为 1，次优先级为 1
优先级 4	10	0	00000	组优先级为 2，次优先级为 0
优先级 5	10	1	00000	组优先级为 2，次优先级为 1
优先级 6	11	0	00000	组优先级为 3，次优先级为 0
优先级 7	11	1	00000	组优先级为 3，次优先级为 1

因此，前面问题中的分组描述是错误的，应该按照表 14 的描述。

通过上面的分组后，LM3S 单片机被分成了以上 8 种情况，可以将任何一种可改变优先级的中断设置成以上的其中一种。

其中，组优先级可设置成 0、1、2、3，次优先级可设成 0、1，而 0 为最高优先级，组优先级可占先；也可以将多个可改变优先级的中断设置成上面的同一种，在这种情况下，优先级按照其硬件优先级执行。

组优先级之间，高组优先级可占先低组优先级，组优先级依然是 0 为最高优先级。次优先级只能采用末尾连锁。

针对这个问题，首先高低组优先级分组理解错误了，正确的应当为前面介绍的。

按照上面描述，优先级 7 的组优先级为 3，而优先级 0 的组优先级为 0；组优先级 0 高于组优先级 3，所以优先级 0 高于优先级 7，优先级 0 可以占先优先级 7。

而优先级 0～优先级 7 可以简单的理解为一个代号，但也不违被优先级规则。

所以问题中说的“优先级 7 高于优先级 0，并且能嵌套优先级 0？”不成立。因为已经误解了优先级分组方法。

问题中的这些宏定义也是正确的，但对于 Luminary 单片机来说，程序清单 8 是无效的设置，将默认为 8 个优先级均为组优先级。

程序清单 8 在 LM3S 中无效的宏定义

```
#define   NVIC_APINT_PRIGROUP_7_1   0x00000000   /*  Priority group 7.1 split   */
#define   NVIC_APINT_PRIGROUP_6_2   0x00000100   /*  Priority group 6.2 split   */
#define   NVIC_APINT_PRIGROUP_5_3   0x00000200   /*  Priority group 5.3 split   */
#define   NVIC_APINT_PRIGROUP_4_4   0x00000300   /*  Priority group 4.4 split   */
```

NVIC_APINT_PRIGROUP_3_5：组优先级为 3 位，可设置成 8 个组优先级；无次优先级。

NVIC_APINT_PRIGROUP_2_6：组优先级为 2 位，可设置成 4 个组优先级；次优先级 1 位，可设置成 2 个次优先级。

NVIC_APINT_PRIGROUP_1_7：组优先级为 1 位，可设置成 2 个组优先级；次优先级为 2 位，可设置成 4 个次优先级；

NVIC_APINT_PRIGROUP_0_8：全部为次优先级，无组优先级，在异常处理时，如果有更高优先级的异常产生，所有中断采用末尾连锁。

如果多个中断组优先级或组次优先级都完全相同，则按照硬件优先级来执行末尾连锁。

占先、末尾连锁、返回、迟来等中断动作的详细资料请阅读《Cortex-M3 开发指南——基于 LM3S8000》或 ARM 公司的《Cortex-M3 技术参考手册》。

7. 声明

修改文档的权利

广州致远电子有限公司保留任何时候在不事先声明的情况下对本文档修改的权力。

8. 销售与服务网络

广州周立功单片机发展有限公司

地址：广州市天河北路 689 号光大银行大厦 15 楼 F1 邮编：510630
电话：(020)38730916 38730917 38730976 38730977
传真：(020)38730925
网址：http://www.zlgmcu.com

广州专卖店
地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634 87569917
传真：(020)87578842

南京周立功
地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221 83613271 83603500
传真：(025)83613271

北京周立功
地址：北京市海淀区知春路 113 号银网中心 712 室（中发电子市场斜对面）
电话：(010)62536178 62536179 82628073
传真：(010)82614433

重庆周立功
地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1611 室
电话：(023)68796438 68796439
传真：(023)68796439

杭州周立功
地址：杭州市登云路 428 号浙江时代电子市场 205 号
电话：(0571)88009205 88009932 88009933
传真：(0571)88009204

成都周立功
地址：成都市一环路南一段 57 号金城大厦 612 室
电话：(028)85499320 85437446
传真：(028)85439505

深圳周立功
地址：深圳市深南中路 2070 号电子科技大厦 A 座 24 楼 2403 室
电话：(0755)83781768 83781788 83782922
传真：(0755)83793285

武汉周立功
地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497 87168297 87168397
传真：(027)87163755

上海周立功
地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452 53083453 53083496
传真：(021)53083491

西安办事处
地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296 83063000 87881295
传真：(029)87880865