# 一起学习 CC3200 系列教程之中断优先级

阿汤哥

序：

能力有限，难免有错，有问题请联系我，

QQ1519256298　　hytga@163.com

Pdf 下载 http://pan.baidu.com/s/1hqiWB56

现在介绍 CC3200 的定时器中断优先级。

特性：

1、　可编程的中断优先级：0-7（共 3bit），0 是最高优先级，7 是最低优先级，值越高，优先级就越低。

2、　可编程的优先级分组：类似于 stm32 的优先级分组，分成抢占优先级和响应优先级，注意：中断优先级只有 3bit，如果设置分组，只有以下情况：

1：0bit 是抢占优先级　　3bit 是响应优先级

2：1bit 是抢占优先级　　2bit 是响应优先级

3：2bit 是抢占优先级　　1bit 是响应优先级

4：3bit 是抢占优先级　　0bit 是响应优先级

情况 1，　中断优先级值：yyy

情况 2，　中断优先级值：xyy

情况 3，　中断优先级值：xxy

情况 4，　中断优先级值：xxx

X 是抢占优先级，Y 是响应优先级。

3、 中断向量表也是可以设置，这里不讨论，等哪天有兴趣了在搞。在 sdk 需要例程的 BoardInit（）函数都会去设置中断向量表。

4、 软件触发中断，譬如定时器一般都是溢出了才发生中断的，但是也可以用软件直接触发中断，当然一般是不需要这个功能的，一般是操作系统才需要这个功能（触发任务切换）。

什么是抢占优先级？什么是响应优先级？

(1)具有高抢占式优先级的中断可以在具有低抢占式优先级的中断处理过程中被响应，即中断嵌套，或者说高抢占式优先级的中断可以嵌套低抢占式优先级的中断。

(2)当两个中断源的抢占式优先级相同时，这两个中断将没有嵌套关系，当一个中断到来后，如果正在处理另一个中断，这个后到来的中断就要等到前一个中断处理完之后才能被处理。

(3)如果这两个中断同时到达，则中断控制器根据他们的响应优先级高低来决定先处理哪一个；如果他们的抢占式优先级和响应优先级都相等，则根据他们在中断表中的排位顺序决定先处理哪一个。

寄存器：不讲解了，有兴趣的自己查 datasheet，可以直接跳过

中断分组设置寄存器

3.3.1.14 APINT Register (offset = D0Ch) [reset = FA050000h]

中断向量表设置寄存器

3.3.1.13 VTABLE Register (offset = D08h) [reset = 0h]

中断优先级设置寄存器

3.3.1.10 PRI_0 to PRI_49 Register (offset = 400h to 4C4h) [reset = 0h]

软件触发中断设置寄存器

3.3.1.7 PEND_0 to PEND_6 Register (offset = 200h to 218h) [reset = 0h]

中断还有其他的寄存器，，，

例程：配置了两个计数器（单次模式，16bit，），配置计数器的中断的优先级，首先使能第一个计数器 A0，然后在 A0 的中断函数使能计数器 A1，然后进行等待，如果 A1 的抢占优先级高于 A0，那么就会发生中断嵌套，如果 A1 的抢占优先级低于 A0 就不会发生中断嵌套。

主要函数讲解：

设置中断优先级分组，参数 1：抢占优先级的位数，

IntPriorityGroupingSet(1);

参数 1：输入范围 0 -8

因为优先级只有 3bit，所以实际起作用只有 0-3，4-8 的效果跟 3 一样，

中断优先级设置函数，参数 1：哪一个中断，参数 2：中断优先级

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_4);

参数 2：输入范围

#define INT_PRIORITY_LVL_0 0x00//中断优先值 0，优先级最高

#define INT_PRIORITY_LVL_1 0x20//中断优先值 1

#define INT_PRIORITY_LVL_2 0x40//中断优先值 2

#define INT_PRIORITY_LVL_3 0x60

#define INT_PRIORITY_LVL_4 0x80

#define INT_PRIORITY_LVL_5 0xA0

#define INT_PRIORITY_LVL_6 0xC0

#define INT_PRIORITY_LVL_7 0xE0//中断优先值 7，优先级最地

优先级设置例程：

示例 1：

IntPriorityGroupingSet(1);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_4);

抢占优先级有 1 位，INT_TIMERA0A 的抢占优先级是 1，相应优先级是 0；

示例 2：

IntPriorityGroupingSet(1);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_3);

抢占优先级有 1 位，INT_TIMERA0A 的抢占优先级是 0，相应优先级是 3；

示例 3：

IntPriorityGroupingSet(1);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_0);

抢占优先级有 1 位，INT_TIMERA0A 的抢占优先级是 0，相应优先级是 0；

示例 3 的中断可以抢占示例 1，但是不能抢占示例 2

示例 4：

IntPriorityGroupingSet(2);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_3);

抢占优先级有 2 位，INT_TIMERA0A 的抢占优先级是 1，相应优先级是 1；

示例 5：

IntPriorityGroupingSet(2);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_2);

抢占优先级有 2 位，INT_TIMERA0A 的抢占优先级是 1，相应优先级是 0；

示例 6：

IntPriorityGroupingSet(2);

IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_1);

抢占优先级有 2 位，INT_TIMERA0A 的抢占优先级是 0，相应优先级是 1；

示例 5 的中断不能抢占示例 4，但是示例 6 可以抢占示例 4

代码

```
void delay(int temp) {
    int i = 0;
    for (i = 0;i < temp;i++) {

    }

}
void
TimerA0Handle(void)
{
```

```
    int i;
    unsigned long ulInts;
    ulInts = TimerIntStatus(TIMERA0_BASE,true);
    TimerIntClear(TIMERA0_BASE,ulInts);
    my_printf("enter A0 Handle\r\n");
    //使能定时器A溢出中断
    TimerIntEnable(TIMERA1_BASE,TIMER_TIMA_TIMEOUT);

    ulInts = TimerIntStatus(TIMERA1_BASE,false);
    TimerIntClear(TIMERA1_BASE,ulInts);
    //使能定时器A溢出了才会发生中断
    TimerEnable(TIMERA1_BASE,TIMER_A);
    //软件触发中断
    //IntPendSet(INT_TIMERA1A);
    delay(4000);
    delay(4000);
    delay(4000);
    delay(4000);
    delay(4000);
    delay(4000);
    my_printf("exit A0 Handle\r\n");
}
void
TimerA1Handle(void)
{
    int i;
    unsigned long ulInts;
    ulInts = TimerIntStatus(TIMERA1_BASE,true);
    TimerIntClear(TIMERA1_BASE,ulInts);
    my_printf("enter A1 Handle\r\n");

    delay(4000);
    delay(4000);
    delay(4000);
    my_printf("exit A1 Handle\r\n");

}
void  int_test(void) {
     unsigned long ulInts;
#define shili 4
#if shili == 1
         //不能发生抢占
        IntPriorityGroupingSet(1);
        IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_3);
```

```
        IntPrioritySet(INT_TIMERA1A, INT_PRIORITY_LVL_0);
#endif
#if shili == 2
         //发生抢占
        IntPriorityGroupingSet(1);
        IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_4);
        IntPrioritySet(INT_TIMERA1A, INT_PRIORITY_LVL_0);
#endif


#if shili == 3
         //不能发生抢占
        IntPriorityGroupingSet(2);
        IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_3);
        IntPrioritySet(INT_TIMERA1A, INT_PRIORITY_LVL_2);
#endif
#if shili == 4
         //发生抢占
        IntPriorityGroupingSet(2);
        IntPrioritySet(INT_TIMERA0A, INT_PRIORITY_LVL_3);
        IntPrioritySet(INT_TIMERA1A, INT_PRIORITY_LVL_1);
#endif
    //使能定时器A溢出中断
    TimerIntEnable(TIMERA0_BASE,TIMER_TIMA_TIMEOUT);

    ulInts = TimerIntStatus(TIMERA0_BASE,false);
    TimerIntClear(TIMERA0_BASE,ulInts);
    //使能定时器A
    TimerEnable(TIMERA0_BASE,TIMER_A);
    delay(40);


}
void main(void) {

    BoardInit();
    uart0_Init();


    //使能timer的时钟
    PRCMPeripheralClkEnable(PRCM_TIMERA0, PRCM_RUN_MODE_CLK);
    //使能timer的时钟
    PRCMPeripheralClkEnable(PRCM_TIMERA1, PRCM_RUN_MODE_CLK);
    //软复位定时器
```

```
    PRCMPeripheralReset(PRCM_TIMERA1);

    //软复位定时器
    PRCMPeripheralReset(PRCM_TIMERA0);
    //设置定时器A的位数为16BIT，向下计数，周期性触发
    TimerConfigure(TIMERA0_BASE,TIMER_CFG_A_ONE_SHOT
|TIMER_CFG_SPLIT_PAIR);
    //设置分频系数：,80Mhz 时钟频率为80Mhz
    TimerPrescaleSet(TIMERA0_BASE,TIMER_A,0);
    //设置溢出值，10000.10ms触发一次中断
    //100o * 0.0125us = 12.5us
    TimerLoadSet(TIMERA0_BASE,TIMER_A, 999);

    //设置中断函数
    TimerIntRegister(TIMERA0_BASE,TIMER_A,TimerA0Handle);




    //设置定时器A的位数为16BIT，向下计数，周期性触发
    TimerConfigure(TIMERA1_BASE,TIMER_CFG_A_ONE_SHOT
|TIMER_CFG_SPLIT_PAIR);
    //设置分频系数：,80Mhz 时钟频率为80Mhz
    TimerPrescaleSet(TIMERA1_BASE,TIMER_A,0);
    //设置溢出值，10000.10ms触发一次中断
    //1000 * 0.0125us = 12.5us
    TimerLoadSet(TIMERA1_BASE,TIMER_A, 999);

    //设置中断函数
    TimerIntRegister(TIMERA1_BASE,TIMER_A,TimerA1Handle);
    int_test();
    while(1)
    {}

}
```