嵌入式中断：如何正确设置中断优先级 (万字总结) - 基于Cortex-M和FreeRTOS

嵌入式　**中断：深入探讨如何正确设置中断优先级 - 基于Cortex-M和FreeRTOS**

---

1. 如何正确设置中断优先级

之前有遇到过基于FreeRTOS　的中断优先级设置不对，导致系统有时随机产生hardfault，实际排查过程很费劲才找到root cause是中断优先级配置不对，这里将相关知识统一整理一下。

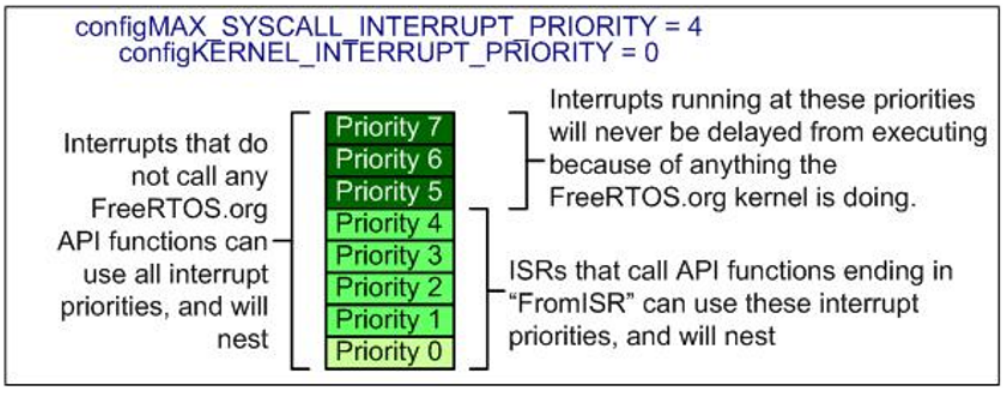
先上结论：

- **如果在中断服务例程中调用了RTOS API函数，其中断优先级的数值应该高于configMAX_SYSCALL_INTERRUPT_PRIORITY**
- 或者说其逻辑优先级必须低于或等于configMAX_SYSCALL_INTERRUPT_PRIORITY（低逻辑优先级意味着高优先级数值）

> FreeRTOS API calls from interrupts can only be called from interrupts *numerically equal or higher* (lower urgency) than configMAX_SYSCALL_INTERRUPT_PRIORITY

既：

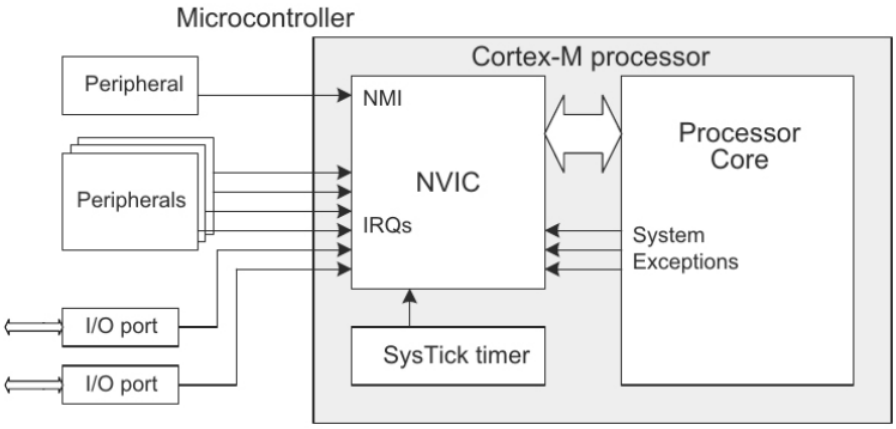
InterruptPriorityNum_ISRCalledRTOSAPI > configMAX_SYSCALL_INTERRUPT_PRIORITY



Example interrupt priority configuration　

再从原理、应用角度深入讨论下：

2. 从Cortex-M角度

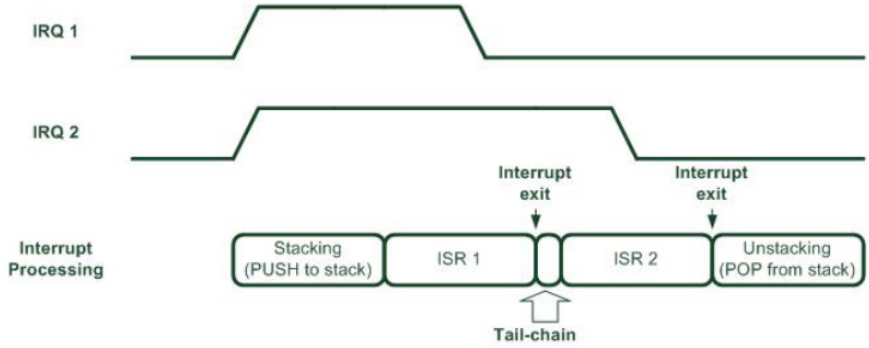


Interrupts can be triggered internally – from a timer, for example – or externally, from peripherals.

Cortex-M通过NVIC(Nested vector interrupt control)来控制管理所有内部和外部中断。

中断优先级，是指在某一个中断产生到处理完成过程中间，又有一个新的中断产生，这时就需要通过中断的优先级来决定接下来的操作：是否打断当前中断执行新中断，还是将新中断Pending。

如下图；



With tail-chaining, if an interrupt is pending as the ISR for another, higher-priority interrupt completes, the processor will immediately begin the ISR for the next interrupt, without restoring its previous state.

Image credit: Arm Limited

Cortex-M构架自身最多允许**256**级可编程优先级，既优先级数值只能在0-255（优先级配置寄存器最多8位，所以优先级范围从0x00~0xFF），而绝大多数微控制器制造商只是使用其中的一部分优先级。

因为我们的MCU用的NXP RT1062，这里就都以此为例
从Datasheet或者MIMXRT1062.h可看出，
1062只使用了其中的高4bits

NOTE

This chip supports up to 16 interrupt priority levels, i.e. it implements bits [7:4] of each NVIC Interrupt Priority Register.

因此针对所有的外部中断，**1062的中断优先级数值支持0(0x0)-15(0xf)，一共16个**, 并且都是可被抢占的（Preemptable）

> 根据Cortex-M内核定义，一个中断的优先级数值越低，逻辑优先级却越高。既中断优先级数值为 0 则代表是最高优先级（逻辑优先级最高），而16则为逻辑最低。

Cortex系列优先级数值和逻辑优先级是相反的：

- 优先级数值，既设置在优先级寄存器里的数值，也是程序中读取和设置的IRQ Priority
- 逻辑优先级，就是字面上说的 优先级高、优先级低

在程序中，NXP Lib通过 **NVIC_SetPriority(IRQ, PriorityNum)** 在0-15之间 设置中断优先级，通过NVIC_SetPriority(LCDIF_IRQn, 8)，最终设置到相应NVIC_IPR(NVIC Interrupt Priority Register)中：

Table 4.2. NVIC register summary

Address	Name	Type	Required privilege	Reset value	Description
0xE000E100 - 0xE000E11C	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000	Interrupt Set-enable Registers
0XE000E180 - 0xE000E19C	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000	Interrupt Clear-enable Registers
0XE000E200 - 0xE000E21C	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000	Interrupt Set-pending Registers
0XE000E280 - 0xE000E29C	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000	Interrupt Clear-pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0-NVIC_IABR7	RW	Privileged	0x00000000	Interrupt Active Bit Registers
0xE000E400 - 0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000	Interrupt Priority Registers
0xE000EF00	STIR	WO	Configurable [a]	0x00000000	Software Trigger Interrupt Register

部分NVIC寄存器Debug数值示例：

Nested Vectored Interrupt ...	Value	Access
NVIC_IABR15	0x00000000	ReadOnly
NVIC_IPR0	0xA0A0A0A0	ReadWrite
NVIC_IPR1	0xA0A0A0A0	ReadWrite
NVIC_IPR2	0xA0A0A0A0	ReadWrite
NVIC_IPR3	0xA0A0A0A0	ReadWrite
NVIC_IPR4	0xA0A0A0A0	ReadWrite
NVIC_IPR5	0xA0A0A0A0	ReadWrite
NVIC_IPR6	0xA0A0A0A0	ReadWrite
NVIC_IPR7	0xA0A0A0A0	ReadWrite
NVIC_IPR8	0xA0A0A0A0	ReadWrite
NVIC_IPR9	0xA0A0A0A0	ReadWrite
NVIC_IPR10	0xA0A0A0A0	ReadWrite
NVIC_IPR11	0xA0A0A030	ReadWrite
NVIC_IPR12	0xA0A0A0A0	ReadWrite
NVIC_IPR13	0xA0A0A0A0	ReadWrite
NVIC_IPR14	0xA0A0A0A0	ReadWrite
NVIC_IPR15	0xA0A0A0A0	ReadWrite

configPRIO_BITS

- configPRIO_BITS (the number of piority bits available)

在FreeRTOSConfig.h中**configPRIO_BITS**定义了MCU使用的优先级寄存器高位Bits，例如在 MIMXRT1062.h定义__NVIC_PRIO_BITS为4 （高4位）

```
4: //  __NVIC_PRIO_BITS == 4 for MIMXRT1062.h
5: /* Interrupt nesting behaviour configuration. Cortex-M specific. */
6: #ifdef __NVIC_PRIO_BITS
7: /* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
8: #define configPRIO_BITS __NVIC_PRIO_BITS
9: #else
0: #define configPRIO_BITS 4 /* 15 priority levels */
1: #endif
2:
```

而NVIC_SetPriority 就是将传入的PriorityNum**左移configPRIO_BITS** 再写入到 该中断寄存器中：NVIC Interrupt Priority Register，实现配置中断优先级。

3. 从RTOS角度

FreeRTOS通过 configPRIO_BITS 来定义所有的中断优先级level，

configLIBRARY_LOWEST_INTERRUPT_PRIORITY

这个宏定义了**逻辑最低优先级**，例如1062 1<<4 -1 最低优先级数值为 15
同时也是systick 和pendSV这两个系统中断的优先级数值

```
:
: /* The lowest interrupt priority that can be used in a call to a "set priority"
: function. */
: #define configLIBRARY_LOWEST_INTERRUPT_PRIORITY ((1U << (configPRIO_BITS)) - 1) //1<<4 -1 =15
:
```

configKERNEL_INTERRUPT_PRIORITY

configLIBRARY_LOWEST_INTERRUPT_PRIORITY 的值在0-15之间，
但是实际NVIC 优先级寄存器只有高4 bits有效，所以通过 宏 configKERNEL_INTERRUPT_PRIORITY 来转换：

一般这个宏设置为 RTOS内核最低的优先级（优先级数值最大）

例如1062 最低优先级 15<<4 = 240, 0xF0， 这个值写入寄存器，既配置为最低优先级

```
:
: /* Interrupt priorities used by the kernel port layer itself.  These are generic
: to all Cortex-M ports, and do not rely on any particular library functions. */
: #define configKERNEL_INTERRUPT_PRIORITY (configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8 - configPRIO_BITS)) //15<<4 = 240, 0xF0
: /* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
```

一般RTOS 内核系统中断都运行在最低优先级 如 SysTick/PendSV

configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY

```
/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS!  (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 2
```

这个宏定义了FreeRTOS Lib能Handle的最大中断优先级数值，

例如设置为2，则：

- 优先级为 0,1的中断，其ISR不能调用RTOS 的API，既不能调用FreeRTOS的任何 _FromISR结尾的API
- 优先级在 2- 15之间的中断，可以随便调用RTOS 的API

configMAX_SYSCALL_INTERRUPT_PRIORITY

```
: * !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
: ee http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
: define configMAX_SYSCALL_INTERRUPT_PRIORITY (configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS)) //2<<4
: * END:   Added by Howard Xue, 2021/4/16 */
:
```

同上面的configKERNEL_INTERRUPT_PRIORITY，因为NVIC 优先级寄存器只有高4 bits有效，如果写入到寄存器里面数值，也需要左移configPRIO_BITS

因此实际写入到NVIC Interrupt Priority Register的值实际为2<<4 = 0x20, 既configMAX_SYSCALL_INTERRUPT_PRIORITY宏的数值

同时，这个数值0x20 会写入到**BASEPRI** 寄存器，FreeRTOS就可以通过BASEPRI来实现屏蔽中断了

> 因此针对这类中断，我们要设置configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 的数值 高于 这类中断优先级的数值， 使得临界区能够生效，这样调用taskENTER_CRITICAL()/taskEXIT_CRITICAL() 才能正常实现临界区的意义

---

4. 中断屏蔽原理

为了保持中断数据不被打断，大部分ISR程序处理中首先会需要进入临界区（如通过调用FreeRTOS的taskENTER_CRITICAL()）

- 在上面栗子中，通过调用taskENTER_CRITICAL()可以屏蔽 中断优先级为 2-15的中断，（在此时所有大于BASEPRI 0x20的中断都会被屏蔽掉，直到退出临界区）
- 但是，如果优先级为0,1的中断产生了，还是会打断该中断的ISR （**既FreeRTOS 临界区也不能屏蔽0,1这两个中断**），导致异常情况发生（如数据丢失、Hardfault，程序跑飞等）

> 大部分在中断服务程序中调用的以“FromISR”结尾的FreeRTOS API，是需要具有**中断调用保护**的，在执行这些函数前会先进入**临界区**操作，但是如果该中断的优先级高于 configMAX_SYSCALL_INTERRUPT_PRIORITY，则临界区就不能生效实现中断保护了

因此这些FromISR函数，不可以被逻辑优先级高于（优先级数值低于）configMAX_SYSCALL_INTERRUPT_PRIORITY的中断服务函数调用。

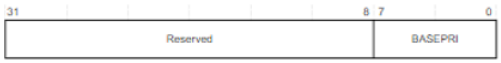
---

5. 临界区原理

BASEPRI寄存器

Base Priority Mask Register

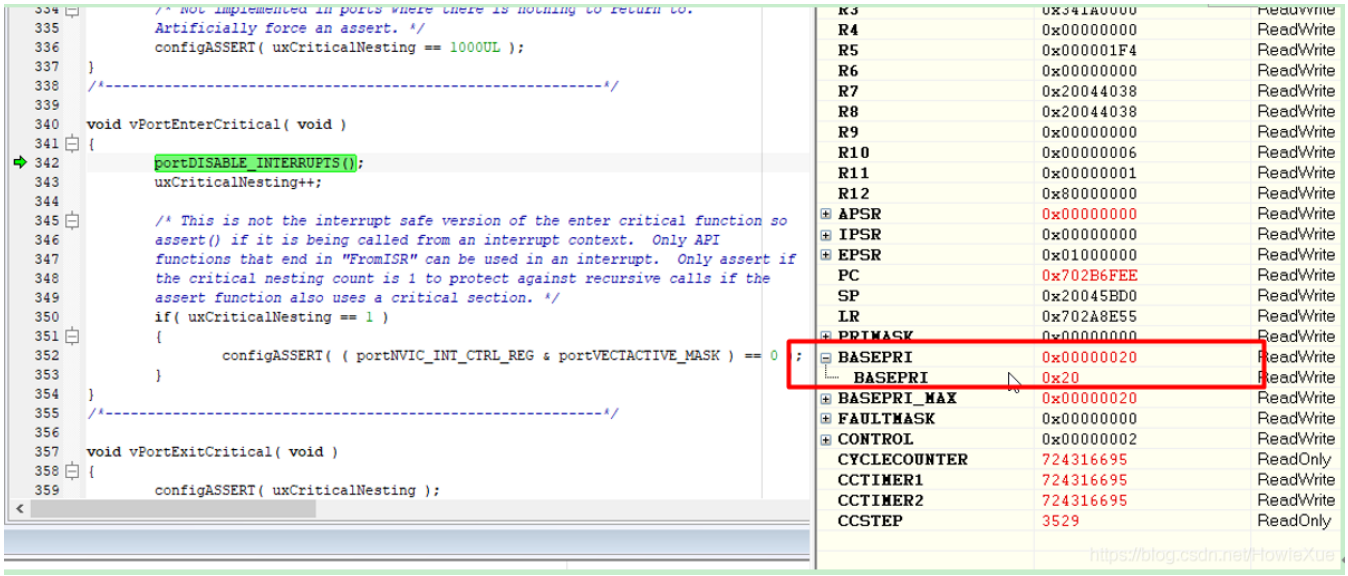
The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in Table 2.2 for its attributes. The bit assignments are:



BASEPRI为优先级屏蔽寄存器，优先级数值大于或等于该寄存器的中断都会被屏蔽，为零时不屏蔽任何中断

因为BASEPRI是寄存器，只有高4bit有作用，所以实际优先级是数值的高4bit：
如BASEPRI: 0x20 则优先级数数值是2， 0x50既为5 （对应NXP优先级数值 0-15）

下图为Debug进入临界区查看的BASEPRI数值示例：



FreeRTOS通过BASEPRI实现临界区

- 当进入临界区时，将寄存器BASEPRI的值设置成configMAX_SYSCALL_INTERRUPT_PRIORITY，如0x20，任何大于0x20的中断都会被屏蔽
- 当退出临界区时，将寄存器BASEPRI的值设置成0。

代码上通过portSET_INTERRUPT_MASK() / portCLEAR_INTERRUPT_MASK()：

It is used in the port to mask interrupts as below:

```
#define portSET_INTERRUPT_MASK() \
 __asm volatile \
 ( \
 " mov r0, %0 \n" \
 " msr basepri, r0 \n" \
 : /* no output operands */ \
 :"i"(configMAX_SYSCALL_INTERRUPT_PRIORITY) /* input */\
 :"r0" /* clobber */ \
 )
/*
* Set basepri back to 0 without effective other registers.
* r0 is clobbered.
*/
#define portCLEAR_INTERRUPT_MASK() \
 __asm volatile \
 ( \
 " mov r0, #0 \n" \
 " msr basepri, r0 \n" \
 : /* no output */ \
 : /* no input */ \
 :"r0" /* clobber */ \
 )
```


vPortEnterCritical() == vPortRaiseBASERPI()

FreeRTOS 调用taskENTER_CRITICAL()进入临界区，底层先调用vPortEnterCritical()，最终实际调用的是vPortRaiseBASERPI()

```
portFORCE_INLINE static void vPortRaiseBASEPRI( void )
{
uint32_t ulNewBASEPRI;

    __asm volatile
    (
        "   mov %0, %1                                              \n" \
        "   msr basepri, %0                                         \n" \
        "   isb                                                     \n" \
        "   dsb                                                     \n" \
        :"=r" (ulNewBASEPRI) : "i" ( configMAX_SYSCALL_INTERRUPT_PRIORITY ) : "memory"
    );
}
```


可以看到，代码实际实现的就是：
BASEPRI = configMAX_SYSCALL_INTERRUPT_PRIORITY；

官方描述有一句比较中肯：

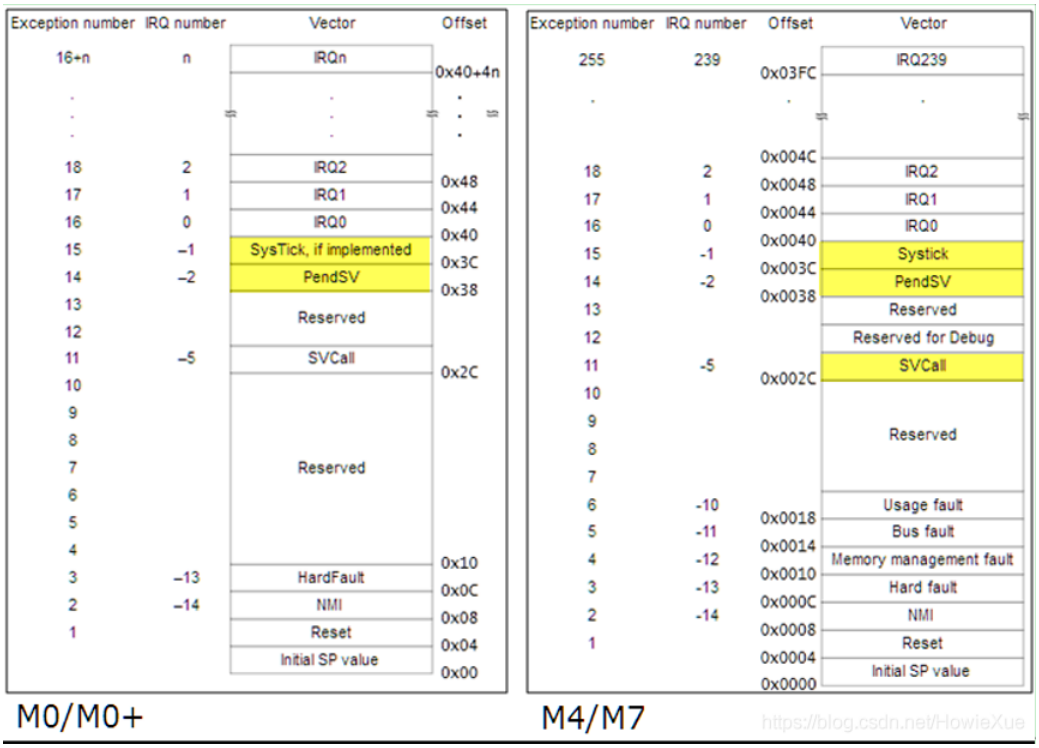
- CriticalSection does not disable all interrupts but allows some high priority interrupts to run.

一句话总结就是：

> **在Cotext-M0，临界区是可以屏蔽所有中断的；**
> **而在Cortex-M3/4/7，临界区只通过BASEPRI 来屏蔽大于或等于configMAX_SYSCALL_INTERRUPT_PRIORITY的中断**

---

6. 系统中断



SysTick

系统时钟中断，RTOS的Timebase，Timer 的interrupt. 频率经常设置在1kHz 或100Hz。

- 一般配置为最低优先级
- 优先级数值==configKERNEL_INTERRUPT_PRIORITY （如1062中为15）

PendSV (Pendable SerVice)

上下文切换中断，既OS上下文切换时，会强制产生一个PendSV中断，来进行task之间的context switch操作。

代码上通过：vPortPendSVHandler()

- 一般配置为最低优先级
- 优先级数值==configKERNEL_INTERRUPT_PRIORITY （如1062中为15）

t拓展一下：FreeRTOS 各Task使用的是PSP(Process Stack Pointer), 而中断使用的是MSP(Main Stack Pointer)， vPortPendSVHandler()则只能在PSP之间切换，不能在MSP和PSP之间切换:

Stack Pointer

The *Stack Pointer* (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = *Main Stack Pointer* (MSP). This is the reset value.
- 1 = *Process Stack Pointer* (PSP).

On reset, the processor loads the MSP with the value from address `0x00000000`.

SVCall (SuperVisor Call)

启动调度器中断，SVC指令会触发该中断，通过来调用FreeRTOS Scheduler调度器，该中断只在启动时产生一次

代码上通过：vPortSVCHandler()

- 配置优先级为最高 0
