

- 存储器系统
- 编程模式
- 中断及其处理
 - 中断及中断向量表
 - 向量表重定位
 - 中断优先级
 - 中断响应过程
 - 咬尾机制
 - 晚到机制
 - 中断延迟
- 低功耗模式
- 存储保护单元(MPU)

Cortex-M3异常与中断

49

- 支持10个系统异常和最多240个外部中断
- 支持3个固定的高优先级和多达256级的可编程优先级，支持128级抢占。
- #0~15在Cortex-M3中定义，IRQ#0~239中断由各个芯片商定义

编号	类型	优先级	描述
0	N/A	N/A	没有异常运行
1	复位	-3	复位
2	NMI	-2	不可屏蔽中断（来自外部NMI输入脚）
3	硬fault	-1	所有被除能的fault，都将“上访”成硬fault
4	存储器管理fault	可编程	MPU访问犯规以及访问非法位置，在“非执行区”取指均可引发
5	总线fault	可编程	总线系统收到错误响应
6	用法fault	可编程	程序错误导致异常：使用了一条无效指令，或者非法的状态转换。
7~10	保留	N/A	
11	SVCall	可编程	执行系统服务调用指令(SVC)引发的异常
12	调试监视器	可编程	调试监视器（断点，数据观察点，或者是外部调试请求）
13	保留	N/A	
14	PendSV	可编程	为系统设备而设的“可悬挂请求”
15	SysTick	可编程	系统滴答时钟
16~255	IRQ#0~IRQ#239	可编程	外部中断#0~#239

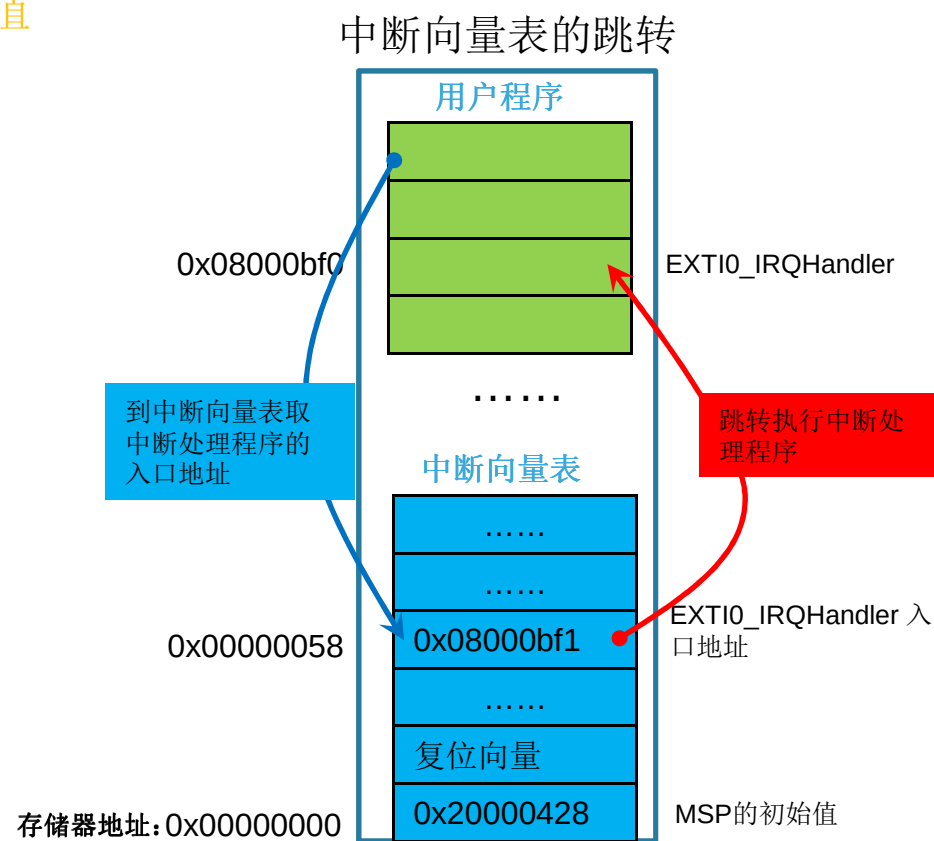
Cortex-M3向量表

50

- 向量表定义了中断的处理例程的入口地址。缺省情况下，CM3认为向量表位于零地址处
- 响应中断时，CM3会根据中断号从表中找出对应的中断处理程序的入口地址
- 每个表项占用4字节
- 位置0x00000000处保存的是MSP的初始值

中断向量表

地址	编号	值（4字节大小）
0x00000000	0	MSP初始值
0x00000004	1	复位向量地址
0x00000008	2	NMI异常处理程序起始地址
0x0000000C	3	硬fault异常处理程序起始地址
.....		
0x00000040	16	IRQ#0中断处理程序起始地址
.....	17~255	IRQ#1~IRQ#239中断处理程序起始地址



STM32F2支持10个
系统异常和81个中断

STM32F2向量表

51

地址	编号	值 (4字节大小)
0x00000000	0	MSP初始值
0x00000004	1	复位向量地址
0x00000008	2	NMI异常处理程序起始地址
0x0000000C	3	硬fault异常处理程序起始地址
.....		
0x00000040	16	WWDG 中断处理程序起始地址
0x00000044	17	PDV起始地址
0x00000048	18	RTC时间戳/RTC侵入中断起始地址
0x0000004C	19	RTC唤醒中断起始地址
0x00000050	20	Flash全局中断起始地址
0x00000054	21	RCC全局中断起始地址
0x00000058	22	EXTI Line0中断起始地址
.....		

由Cortex-M3定义

__vector_table		
DCD	sfe (CSTACK)	
DCD	Reset_Handler	; Reset Handler
DCD	NMI_Handler	; NMI Handler
DCD	HardFault_Handler	; Hard Fault Handler
DCD	MemManage_Handler	; MPU Fault Handler
DCD	BusFault_Handler	; Bus Fault Handler
DCD	UsageFault_Handler	; Usage Fault Handler
0		; Reserved
0		; Reserved
0		; Reserved
0		; Reserved
DCD	SVC_Handler	; SVCcall Handler
DCD	DebugMon_Handler	; Debug Monitor Handler
DCD	0	; Reserved
DCD	PendSV_Handler	; PendSV Handler
DCD	SysTick_Handler	; SysTick Handler
; External Interrupts		
DCD	WWDG_IRQHandler	; Window WatchDog
DCD	PVD_IRQHandler	; PVD through EXTI Line detection
DCD	TAMP_STAMP_IRQHandler	; Tamper and TimeStamps through the EXTI line
DCD	RTC_WKUP_IRQHandler	; RTC Wakeup through the EXTI line
DCD	FLASH_IRQHandler	; FLASH
DCD	RCC_IRQHandler	; RCC
DCD	EXTI0_IRQHandler	; EXTI Line0

startup_stm32f2xx.s

由STM32定义

```

* @brief This function handles SysTick Handler.
* @param None
* @retval None
*/
void SysTick_Handler(void)
{
    TimingDelay_Decrement();
}

```

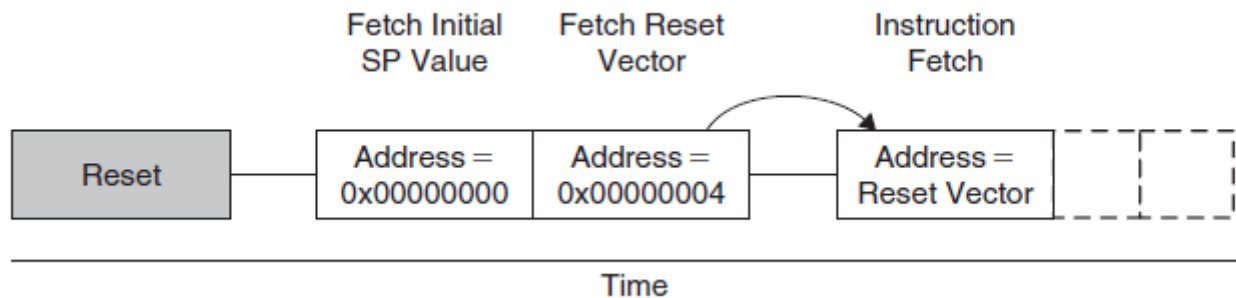
stm32f2xx_it.c

向量表中的MSP初始值和复位向量

52

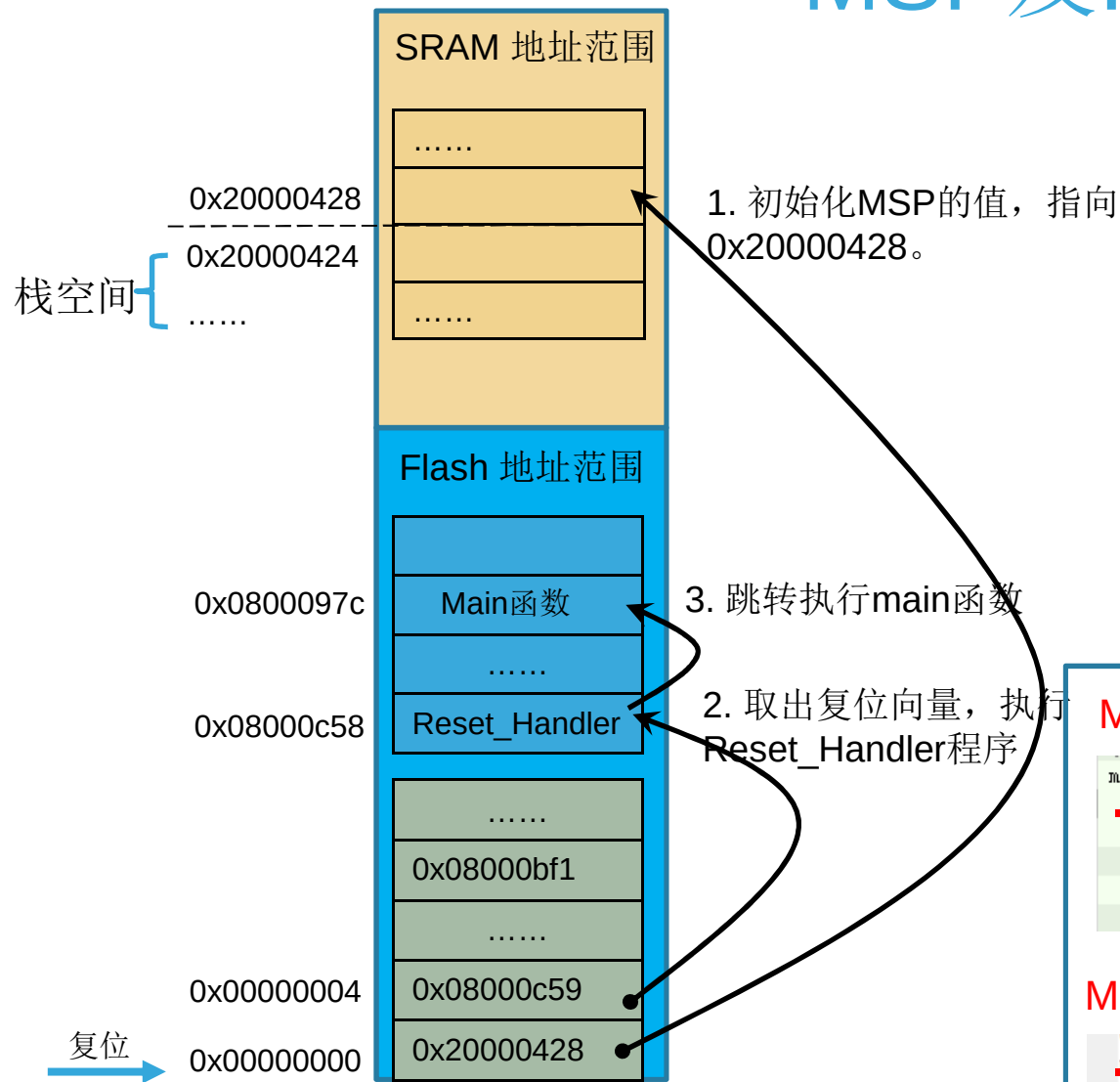
CM3离开复位状态时，首先要做的是读取下面两个值：

- 从地址0x0000 0000,取出MSP(主堆栈指针)的值
- 从地址0x0000 0004,取出复位向量(程序开始执行的地址，LSB必须是1)



MSP及PC初始化举例

53



必须保证加载到PC的数值是奇数（即LSB=1），用以表明这是在Thumb状态下执行。

Main函数实际开始地址

```
main:
0x800097c: 0xb510      PUSH    {R4, LR}
0x800097e: 0xb092      SUB     SP, SP, #0x48
HAL_Init();
0x8000980: 0xf7ff 0xfe5c BL      HAL_Init
BSP_LED_Init(LED3);
```

Map文件中，Main函数的地址

Symbol	Address	Size	Type
main	0x0800097d	0xbe	Code
__main	0x20000000	0x4	Data

- 存储器系统
- 编程模式
- 中断及其处理
 - 中断及中断向量表
 - 向量表重定位
 - 中断优先级
 - 中断响应过程
 - 咬尾机制
 - 晚到机制
 - 中断延迟
- 低功耗模式
- 存储保护单元(MPU)

向量表重定位

55

- 可以通过修改“向量表偏移量寄存器VTOR”来重定位向量表
- 向量表的起始地址的限制。必须先求出系统中共有多少个向量，再把该数字向上“圆整”到2的整次幂，而起始地址必须对齐到后者的边界上。例如：一共有32个中断，则共有 $32+16=48$ 个向量，向上圆整到2的整次幂后值为64，因此向量表重定位的地址必须能被 $64 \times 4 = 256$ 整除。合法的起始地址可以是：0x0, 0x100, 0x200等。

向量表偏移量寄存器VTOR(地址：0xE000_ED08)

只能在特权级下访问

位	名称	类型	复位值	描述
30:31	Reserved			
29	TBLBASE	R/W	0	向量表是在code区(0)，还是RAM区(1)
7:28	TBLOFF	R/W	0	向量表的起始地址
0:6	Reserved			

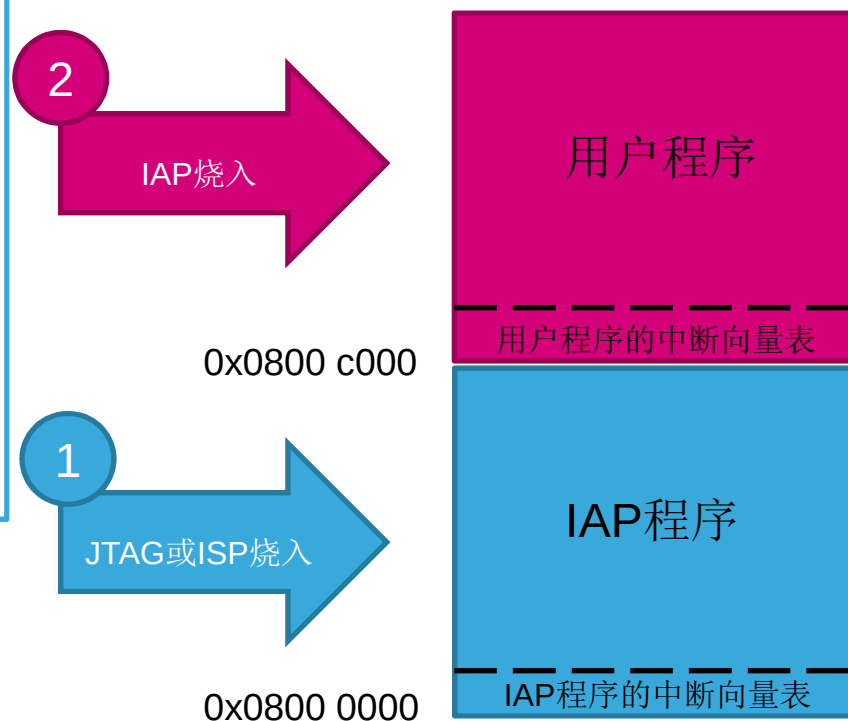
向量表只能放在内部Flash和RAM区

STM32向量表重定位在IAP中的使用(1/2)

56

什么是IAP?

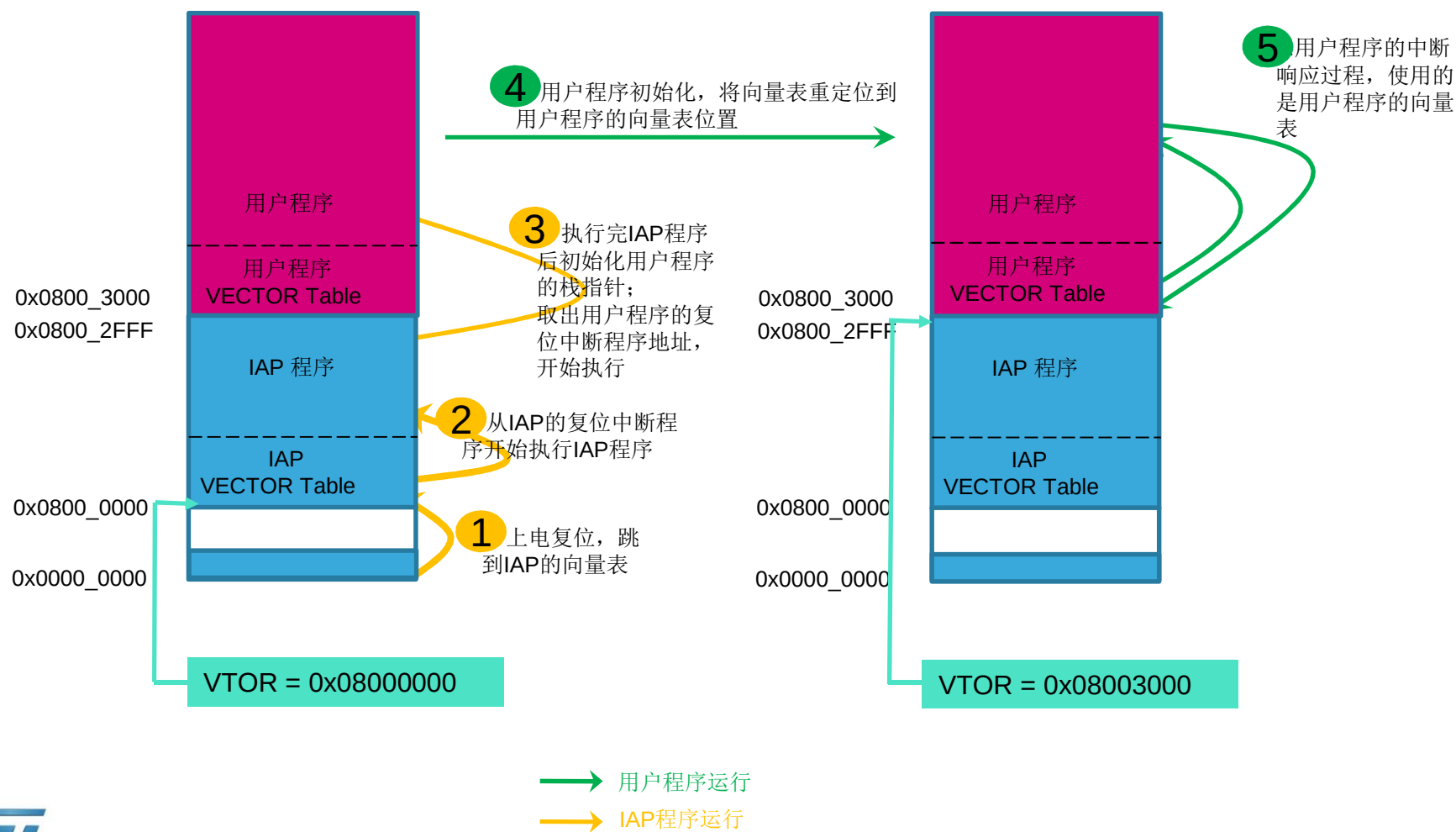
- IAP (In-Application Programming), 在应用编程
- 需要准备两个程序:
 - IAP程序: 不执行正常的功能操作, 只通过某种方式接受上位机的数据, 来对第二段程序进行更新;
 - 用户程序: 真正的功能代码
- IAP程序放在Flash的起始位置, 上电首先执行。用户程序代码放在IAP程序后面
- 用户程序开始执行前, 要把CPU的向量表映射到自己的向量表位置



STM32向量表重定位在IAP中的使用(2/2)

57

IAP启动流程:



STM32的IAP代码示例

58

Step 3 :

```
int main(void){
.....
/* Jump to user application */
```

IAP程序的main函数

```
    JumpAddress = *(__IO uint32_t*) (ApplicationAddress + 4); //取出复位中断向量的地址

    Jump_To_Application = (pFunction) JumpAddress;
    /* Initialize user application's Stack Pointer */
    __set_MSP(*(__IO uint32_t*) ApplicationAddress); //初始化用户程序的堆栈指针

    Jump_To_Application(); //跳转执行用户程序复位程序
}
```

Step 4 :

```
#define VECT_TAB_OFFSET 0x3000 /*!< Vector Table base offset field. */
```

```
.....
void SystemInit(void)
{
```

用户程序的systemInit函数
System_stm32fx.c

```
.....
/* Configure the Vector Table location add offset address -----*/
#ifdef VECT_TAB_SRAM
    SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal
SRAM */
#else
    SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal
FLASH */
#endif
}
```

- 存储器系统
- 编程模式
- 中断及其处理
 - 中断及中断向量表
 - 向量表重定位
 - 中断优先级
 - 中断响应过程
 - 咬尾机制
 - 晚到机制
 - 中断延迟
- 低功耗模式
- 存储保护单元(MPU)

Cortex-M3优先级设定

60

- CM3支持**3个固定最高优先级**和多达**256级**的可编程优先级，并且支持**128级抢占**
- 每个中断的优先级由一个**8位**的寄存器来设定，分为高低两个位段。高位段表示抢占优先级，低位段表示子优先级。CM3允许最少使用位数为**3个位**，即至少要支持**8级**优先级。
- 优先级以**MSB对齐**，简化程序的跨器件移植



优先级分组设置

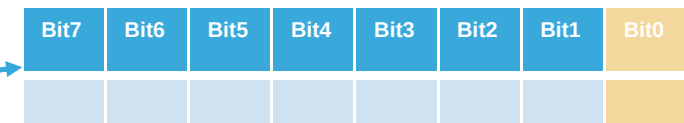
61

- 在应用程序中断及复位控制器AIRCRC中设置优先级分组
(地址: 0xE000_ED0C)

位	名称	类型	复位值	描述
10:8	PRIGROUP	R/W	0	优先级分组

设置优先级
分组

PRIGROUP P的值	抢占优先级的 位段	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]



优先级设置

62

- 中断优先级寄存器阵列IPRx(地址：0xE000_E400 ~0xE000_E4EF)

地址	名称	类型	复位值	描述
0xE000_E400	PRI_0	R/W	0	外部中断0的优先级
0xE000_E401	PRI_0	R/W	0	外部中断1的优先级
.....				
0xE000_E4EF	PRI_239	R/W	0	外部中断239的优先级

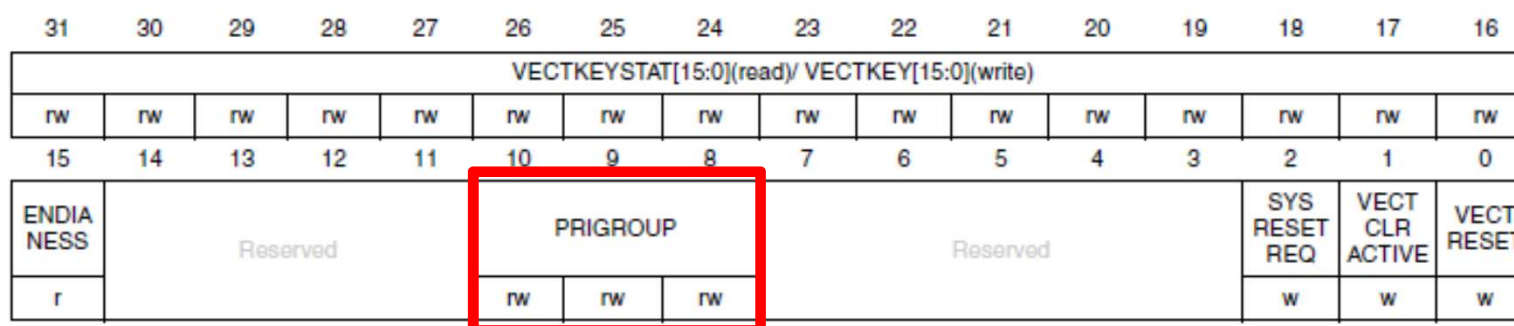
- 系统异常优先级寄存器阵列SHPRx(地址：0xE000_ED18 ~0xE000_ED23)

地址	名称	类型	复位值	描述
0xE000_ED18	PRI_4	R/W	0	存储器管理fault的优先级
0xE000_ED19	PRI_5	R/W	0	总线fault的优先级
0xE000_ED19	PRI_6	R/W		用法fault的优先级
.....				
0xE000_ED23	PRI_15	R/W	0	SysTick的优先级

STM32F2实现的中断优先级寄存器(1/2)

63

- 应用程序中断及复位控制器SCB_AIRCR(地址: 0xE000_ED0C)



PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point ⁽¹⁾	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b011	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

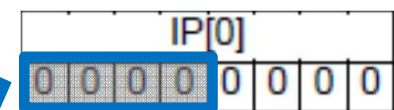
STM32只用了中断优先级寄存器的高4位

STM32F2实现的中断优先级寄存器(2/2)

64

- 中断优先级寄存器NVIC_IPRx (地址从0xE000_E400开始)

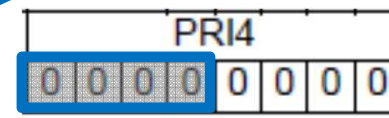
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x204	NVIC_IABR1	ACTIVE[63:32]																																
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x208	NVIC_IABR2	Reserved																ACTIVE [80:64]																
	Reset Value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x300	NVIC_IPR0	IP[3]					IP[2]					IP[1]					IP[0]																	
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
⋮	⋮	⋮																																
0x320	NVIC_IPR20	Reserved																IP[80]																
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
SCB registers																																		
Reserved																																		
0xE00	NVIC_STIR	Reserved																								INTID[8:0]								
	Reset Value																									0	0	0	0	0	0	0	0	0



- 只有bit[7:4]有效，bit[3:0]写无效，读出为0
- 最多支持16级优先级

- 系统异常优先级寄存器SCB_SHPRx (地址从0xE000_ED18开始)

0x18	SCB_SHPR1	Reserved	PRI6								PRI5								PRI4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Reset Value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



STM32设置中断优先级的函数

65

stm32fxxx_hal_cortex.c
/stm32fxxx_hal_cortex.h

```
/** @defgroup CORTEX_Preemption_Priority_Group
 * @{
 */

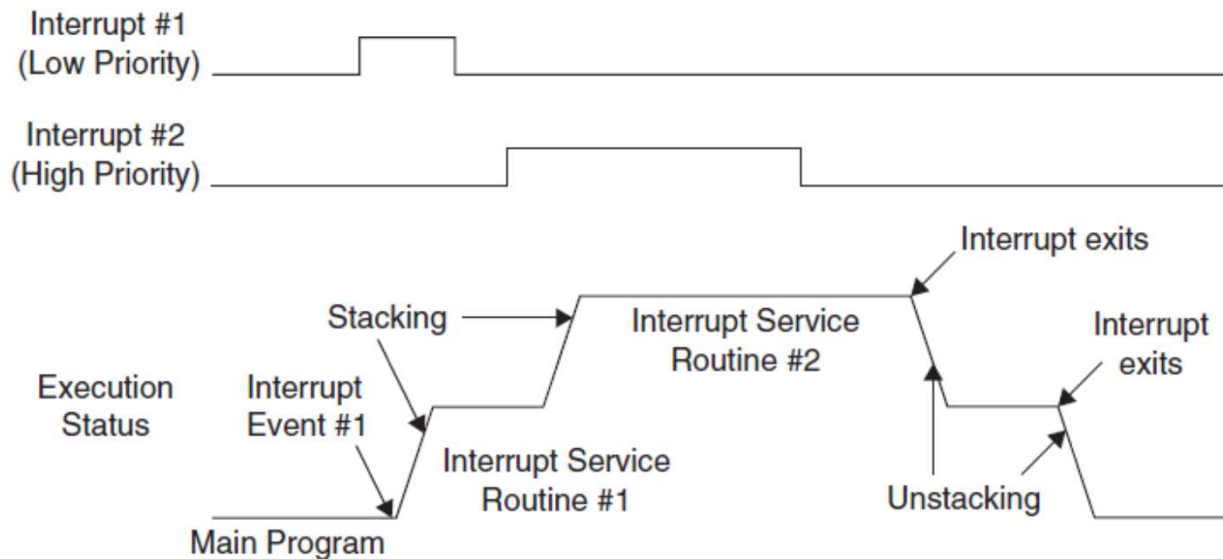
#define NVIC_PRIORITYGROUP_0      ((uint32_t)0x00000007) /*!< 0 bits for pre-emption priority
4 bits for subpriority */
#define NVIC_PRIORITYGROUP_1      ((uint32_t)0x00000006) /*!< 1 bits for pre-emption priority
3 bits for subpriority */
#define NVIC_PRIORITYGROUP_2      ((uint32_t)0x00000005) /*!< 2 bits for pre-emption priority
2 bits for subpriority */
#define NVIC_PRIORITYGROUP_3      ((uint32_t)0x00000004) /*!< 3 bits for pre-emption priority
1 bits for subpriority */
#define NVIC_PRIORITYGROUP_4      ((uint32_t)0x00000003) /*!< 4 bits for pre-emption priority
0 bits for subpriority */

void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup)
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
```

中断优先级响应规则

66

- 抢占行为由抢占优先级决定，只有抢占优先级更高的中断，才可以抢占当前正在响应的中断
- 当抢占优先级相同的中断有不只一个挂起时，最先响应子优先级最高的中断
- 如果优先级完全相同的多个中断同时挂起，则先响应中断编号最小的那一个



■ PRIMASK、FAULTMASK和BASEPRI寄存器（只能在特权级下访问）

名字	功能描述
PRIMASK	1bit的寄存器，置1后屏蔽除NMI和硬fault以外所有可屏蔽的异常。
FAULTMASK	1bit的寄存器，置1后，只有NMI才能响应。
BASEPRI	最多9bit,定义了被屏蔽优先级的阈值。所有优先级号大于等于此值的中断都被关。

还可以使用BASEPRI_MAX这名字来访问BASEPRI寄存器。

- 使用BASEPRI时，可以任意设置新的优先级阈值；
- 使用BASEPRI_MAX时，置允许新的优先级阈值比原来的那个在数值上更小。只能一次次扩大屏蔽范围。

PRIMASK、FAULTMASK的使用

68

PRIMASK置1

仅有NMI和硬fault异常能响应

编号	类型
2	NMI异常处理程序起始地址
3	硬fault异常处理程序起始地址
4	存储器管理fault异常处理程序起始地址
5	总线fault异常处理程序起始地址
6	用法fault异常处理程序起始地址
7~10	保留
11	SVCall异常处理程序起始地址
12	调试监视器异常处理程序起始地址
13	保留
14	PendSV异常处理程序起始地址
15	SysTick异常处理程序起始地址
16	IRQ#0中断处理程序起始地址
17~255	IRQ#1~IRQ#239中断处理程序起始地址

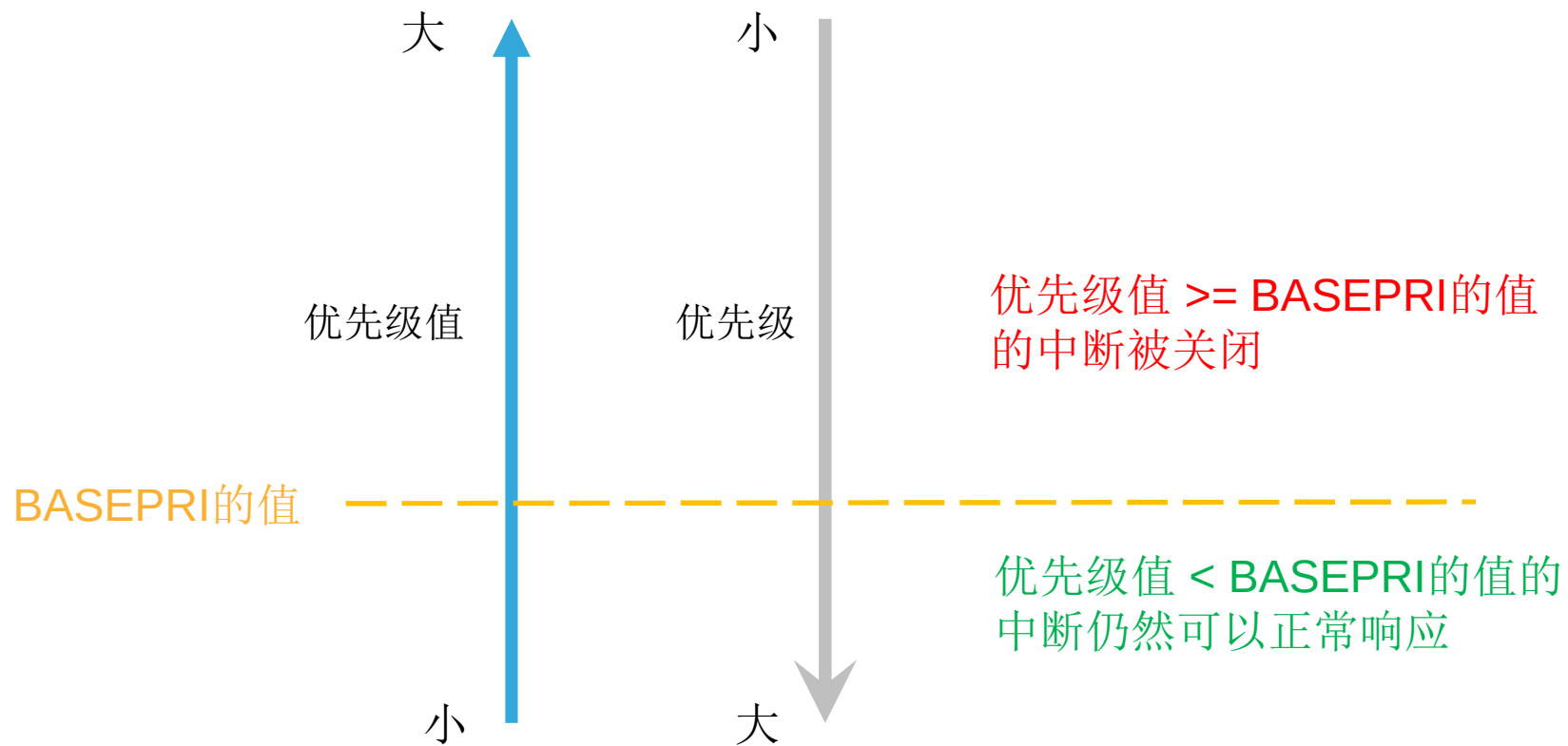
FAULTMASK置1

仅有NMI异常能响应

编号	类型
2	NMI异常处理程序起始地址
3	硬fault异常处理程序起始地址
4	存储器管理fault异常处理程序起始地址
5	总线fault异常处理程序起始地址
6	用法fault异常处理程序起始地址
7~10	保留
11	SVCall异常处理程序起始地址
12	调试监视器异常处理程序起始地址
13	保留
14	PendSV异常处理程序起始地址
15	SysTick异常处理程序起始地址
16	IRQ#0中断处理程序起始地址
17~255	IRQ#1~IRQ#239中断处理程序起始地址

BASEPRI的使用

69



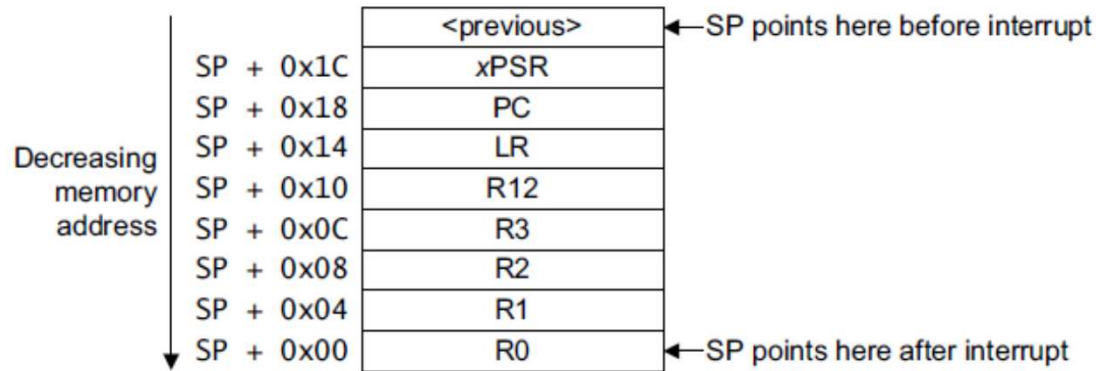
- 存储器系统
- 编程模式
- 中断及其处理
 - 中断及中断向量表
 - 向量表重定位
 - 中断优先级
 - 中断响应过程
 - 咬尾机制
 - 晚到机制
 - 中断延迟
- 低功耗模式
- 存储保护单元(MPU)

中断响应过程

71

CM3开始响应中断时，要做三件事情：

- **入栈：**把xPSP，PC，LR,R12以及R3~R0寄存器的值压入栈
 - 如果程序过大，需要用到R4~R11，由编译器负责生成代码来push他们
 - 当前代码正在使用PSP，则压入PSP，正在使用MSP则压入MSP。进入服务程序后，将一直使用MSP



- **取中断向量：**同时从中断向量表中找出对应的服务程序入口地址
- **更新寄存器：**
 - 更新SP（PSP或MSP）
 - PSR的IPSR位段（新响应的中断编号）
 - PC（指向中断服务程序的入口地址）
 - LR（EXC_RETURN）
 - NVIC相关寄存器

- 根据LR中的EXC_RETURN值决定返回的模式和使用堆栈指针

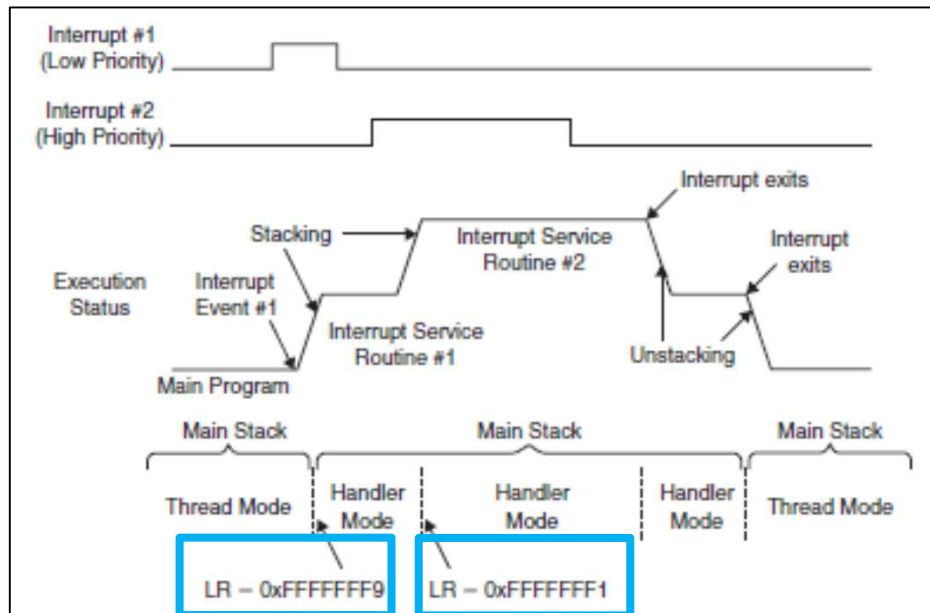
位段	含义
[31:4]	EXC_RETURN的标识，必须全为1
3	0 = 返回后进入Handler模式 1 = 返回后进入线程模式
2	0 = 从主堆栈中做出栈操作，返回后使用MSP， 1 = 从进程堆栈中做出栈操作，返回后使用PSP
1	保留，必须为0
0	0 = 返回ARM状态 1 = 返回Thumb状态，在CM3中必须为1

EXC_RETURN数值	功能
0xFFFF_FFF1	返回handler模式
0xFFFF_FFF9	返回线程模式，并使用主堆栈（SP=MSP）
0xFFFF_FFFD	返回线程模式，并使用线程堆栈（SP=PSP）

- 出栈：恢复先前压入栈的寄存器值，恢复堆栈指针值
- 更新NVIC相关寄存器

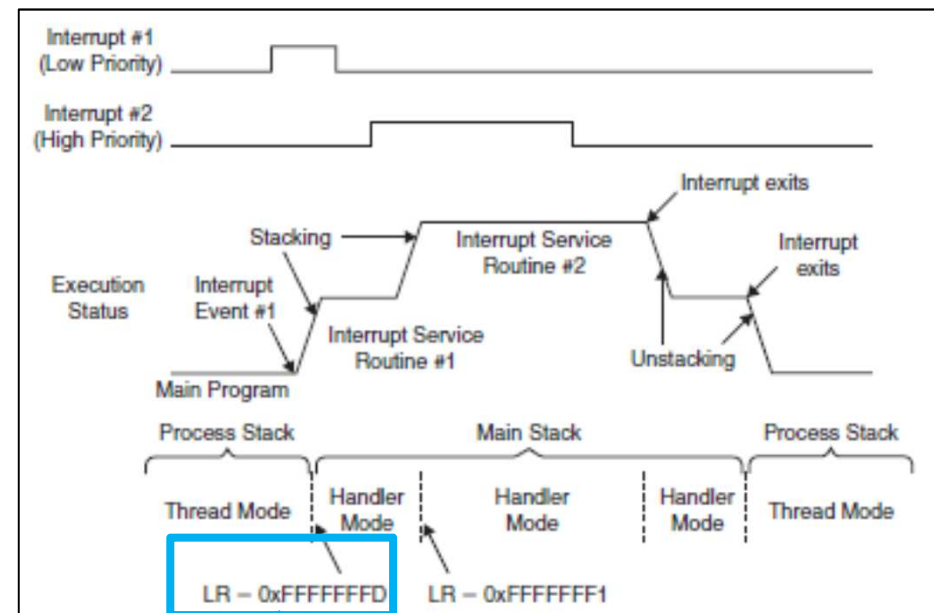
EXC_RETURN值

73



主程序在线程模式下运行，并在使用MSP时被打断

主程序在Handler模式下运行时被打断。

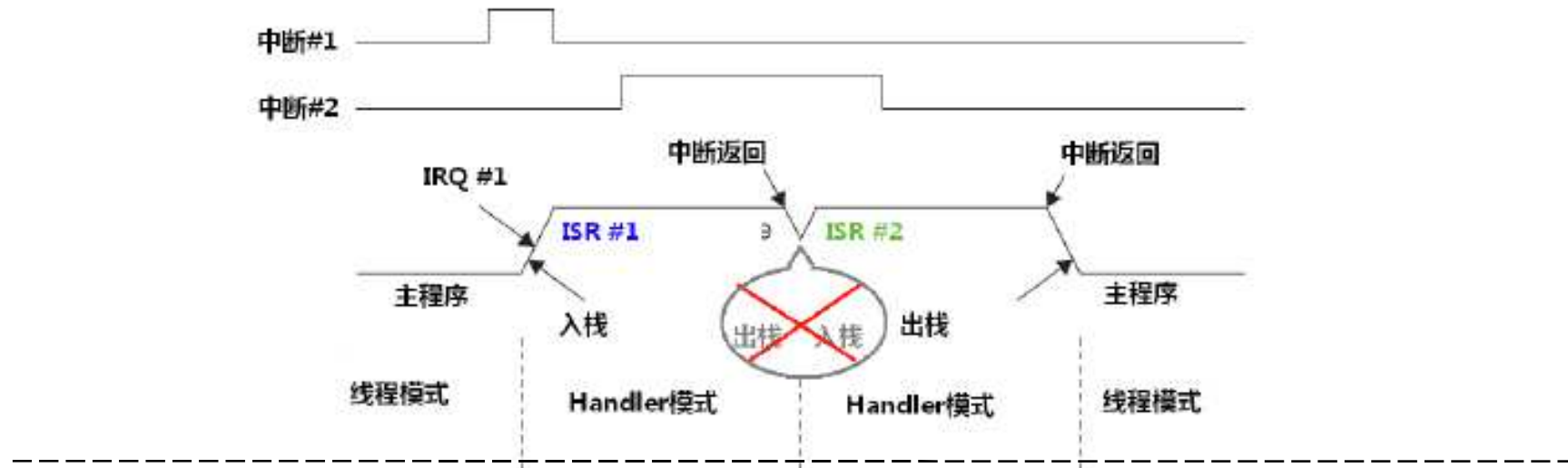


主程序在线程模式下运行，并在使用PSP时被打断

咬尾中断机制

74

- 当中断处理完后，还有其他中断处于挂起状态，处理器不会返回到中断前的程序，而是重新进入中断处理流程
- 降低中断处理开销



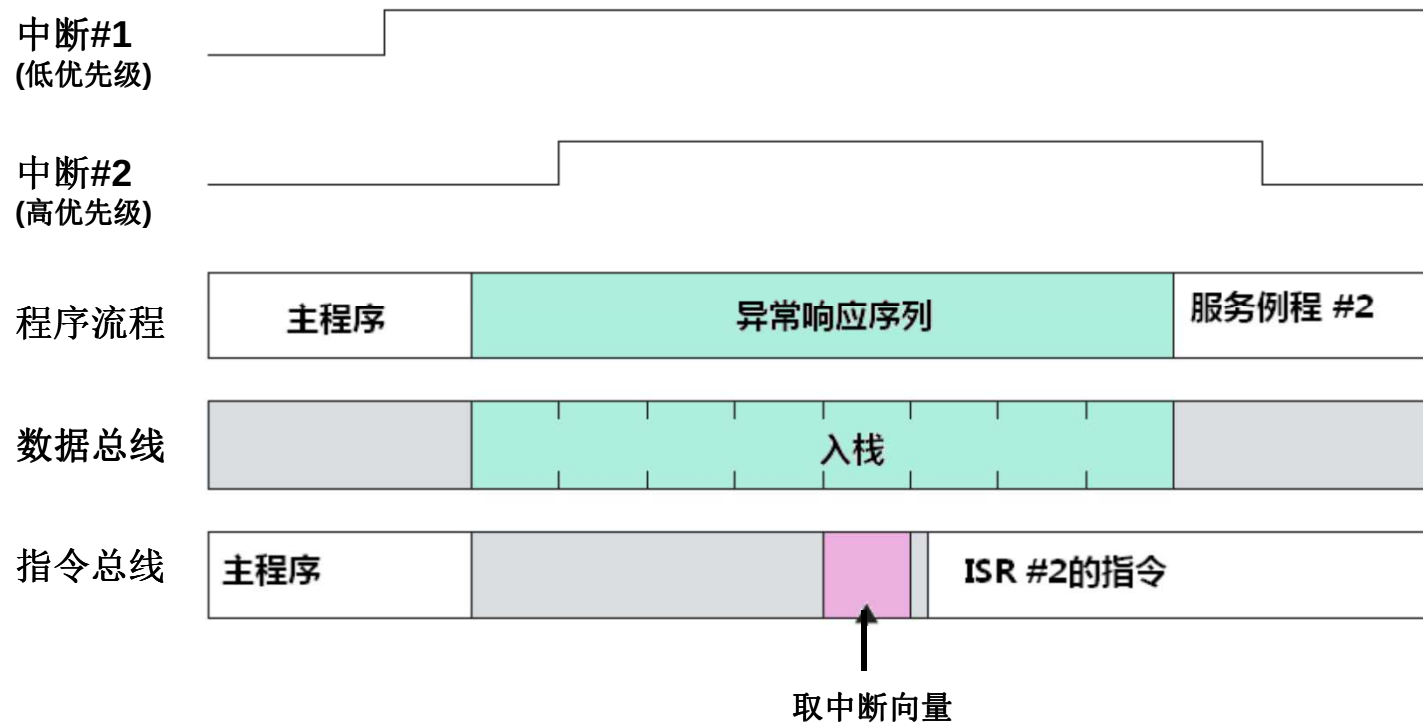
咬尾中断
VS
常规处理



晚到中断机制

75

- 如果在低优先级中断（ISR#1）压栈过程中发生了高优先级中断（ISR#2），处理器会首先处理高优先级中断
- 待ISR#2执行完后，以“咬尾中断”的方式，执行ISR#1
- 减少中断延迟



- 中断延迟：从检测到某中断请求，到执行了其服务例程的第1条指令时，所用的时间。
- 在CM3中，若存储器系统够快，总线系统允许入栈与取指同时进行，同时该中断可以立即响应，则中断延迟是12个周期。
- 处理咬尾中断时，省去了堆栈操作，切入新中断服务例程的时间可以缩短至6个周期。
- 对于接到中断请求时，正在执行需要较多周期才能完成的指令的处理。
 - LDRD/STRD，取消执行，中断返回后重新开始。
 - LDM/STM，在xPSR中记录LDM/STM的执行进度（ICI），待服务例程返回后，再从xPSR中获取当时的执行进度，继续传送。如果IF-THEN(IT)指令使用了ICI/IT位，则不再记录LDM/STM的进度，取消，待中断返回重新开始。

其他中断相关寄存器

77

所有NVIC的寄存器都只能在特权级下访问
(软件触发中断寄存器可配置成用户级访问权限)

名称	地址	描述
SETENA	0xE000_E100~ 0xE000_E11C	中断使能寄存器。一共8个32位寄存器，每个寄存器的每一位对应一个中断IRQ。
CLRENA	0xE000_E180~ 0xE000_E19C	中断除能寄存器。一共8个32位寄存器，每个寄存器的每一位对应一个中断IRQ。
SETPEND0~ SETPEND7	0xE000_E200~ 0xE000_E21C	中断挂起寄存器。一共8个32位寄存器，每个寄存器的每一位对应一个中断IRQ。
CLRPEND0~ CLRPEND7	0xE000_E280~ 0xE000_E29C	中断清除寄存器。一共8个32位寄存器，每个寄存器的每一位对应一个中断IRQ。
ACTIVE	0xE000_E300~ 0xE000_E31C	活动状态寄存器。每个中断对应一个比特位。处理器执行了其ISR的第1条指令后，它的活动位被置1，直到ISR返回才硬件清0。
SHCSR	0xE000_ED24	系统异常控制及状态寄存器
ICSR	0xE000_ED04	中断控制及状态寄存器
STIR	0xE000_EF00	软件触发中断寄存器

- 存储器系统
- 编程模式
- 中断及其处理
- 低功耗模式
 - 低功耗模式的定义
 - 低功耗模式的进入和退出
 - Sleep-on-exit
 - 唤醒中断控制器WIC
 - STM32的低功耗模式
- 存储保护单元(MPU)

低功耗模式

79

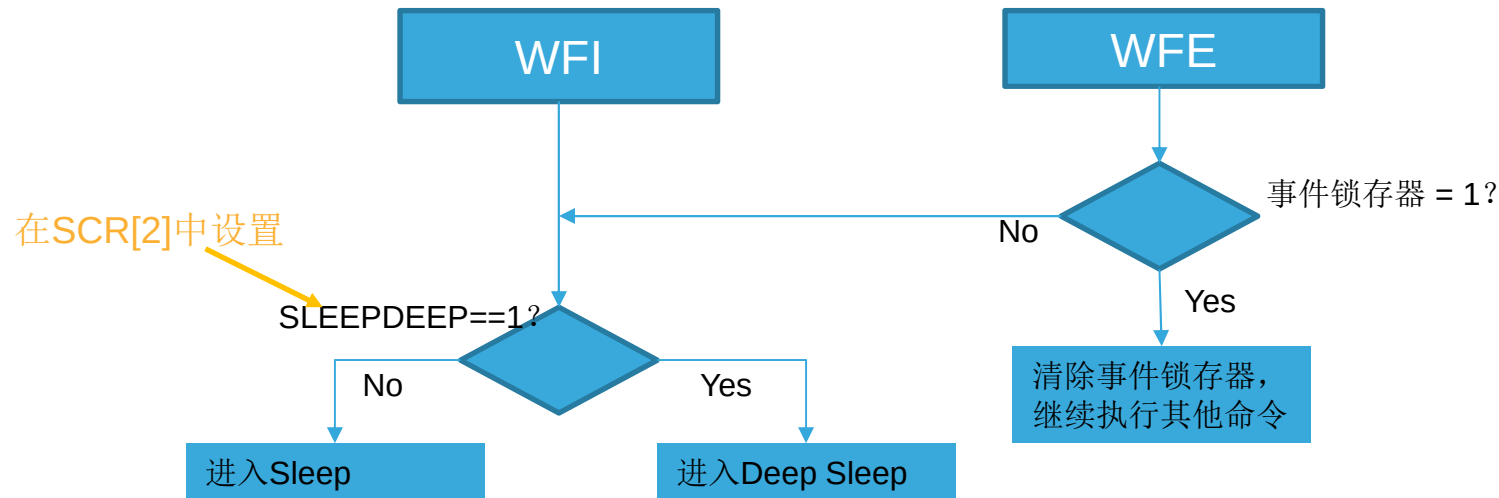
Cortex-M3 定义		STM32定义*	
低功耗模式	说明	低功耗模式	说明
Sleep mode	内核停止工作，外设继续工作	Sleep mode	内核停止工作，外设继续工作
Deep sleep mode	停止系统时钟，关闭PLL和Flash	Stop mode	所有时钟停止
		Standby mode	内核区域掉电

* STM32L1定义了更多的低功耗模式

低功耗模式的进入

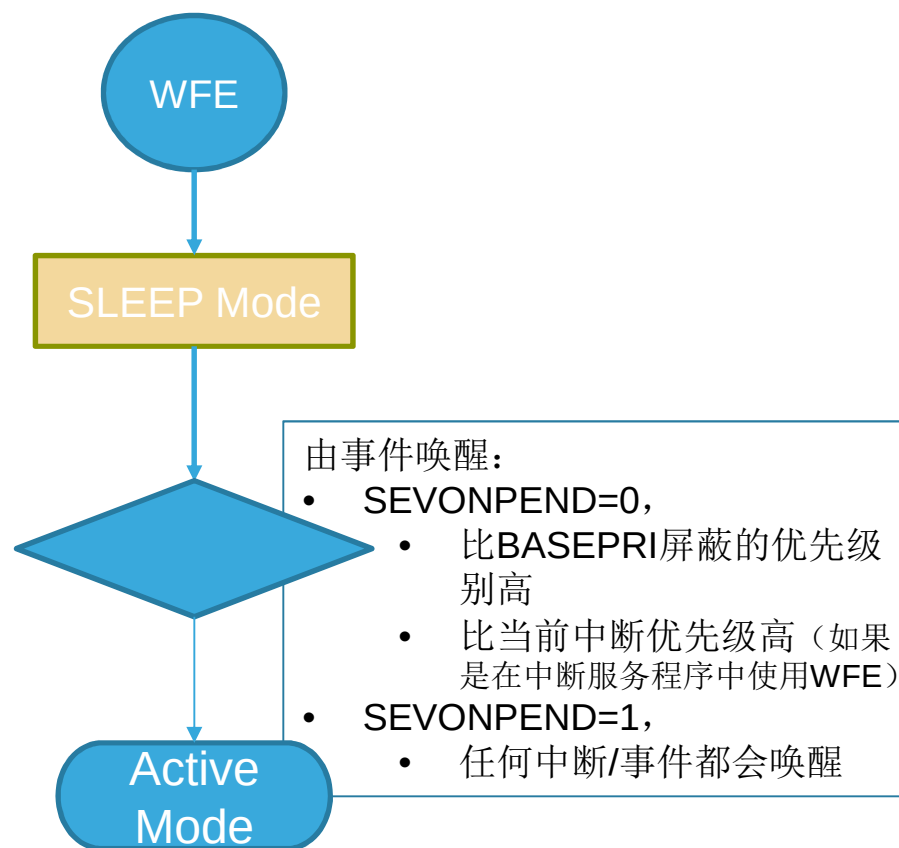
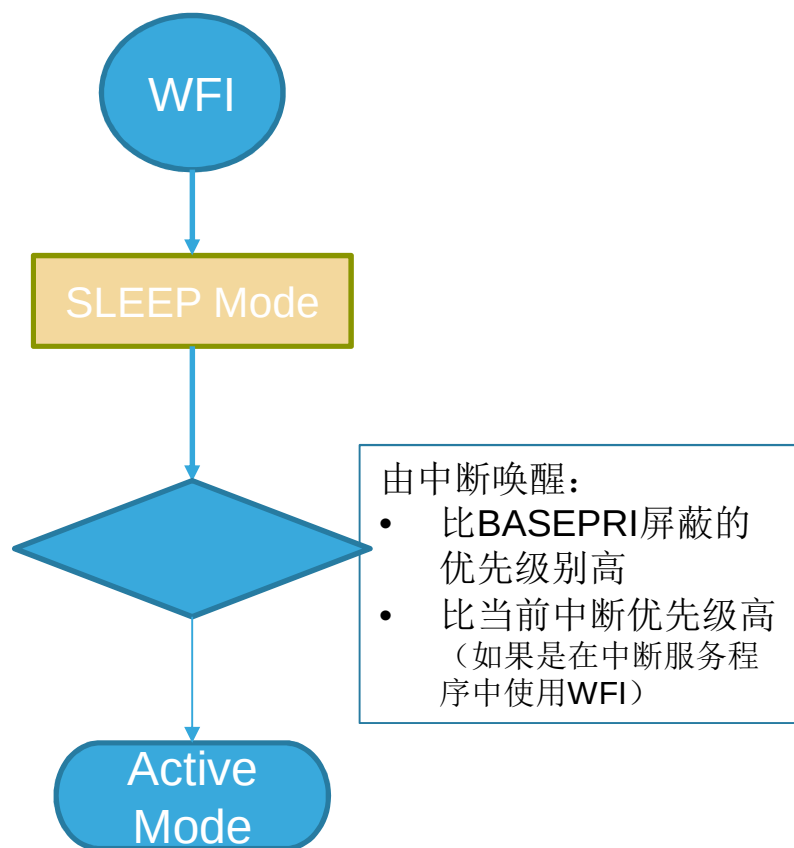
80

- WFI(Wait for interrupt): 执行到WFI指令后，处理器立刻进入睡眠状态
- WFE (Wait for event)：执行到WFE指令后，先检查事件锁存器。如果为0，则立刻进入睡眠状态；如果为1，则清除该锁存器，然后继续执行其他命令。



低功耗模式的退出

81



- PRIMASK位对是否能唤醒处理器没有影响，只决定中断是否执行

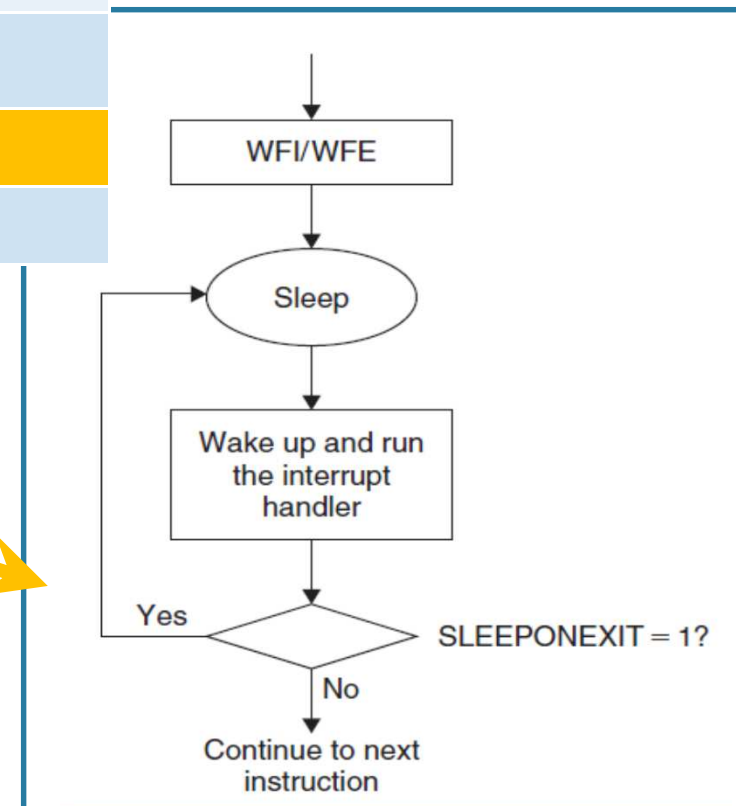
Sleep-on-exit

82

系统控制寄存器SCR（地址：0xE000_ED10）

位	名称	类型	描述
4	SEVONPEND	R/W	置1：任何中断挂起都把CM3从WFE指令处唤醒，不管这个中断的优先级是否比当前高。
3	保留	----	----
2	SLEEPDEEP	R/W	0：进入sleep模式 1：进入deep sleep模式
1	SLEEPONEXIT	R/W	激活sleep-on-exit功能
0	保留	----	----

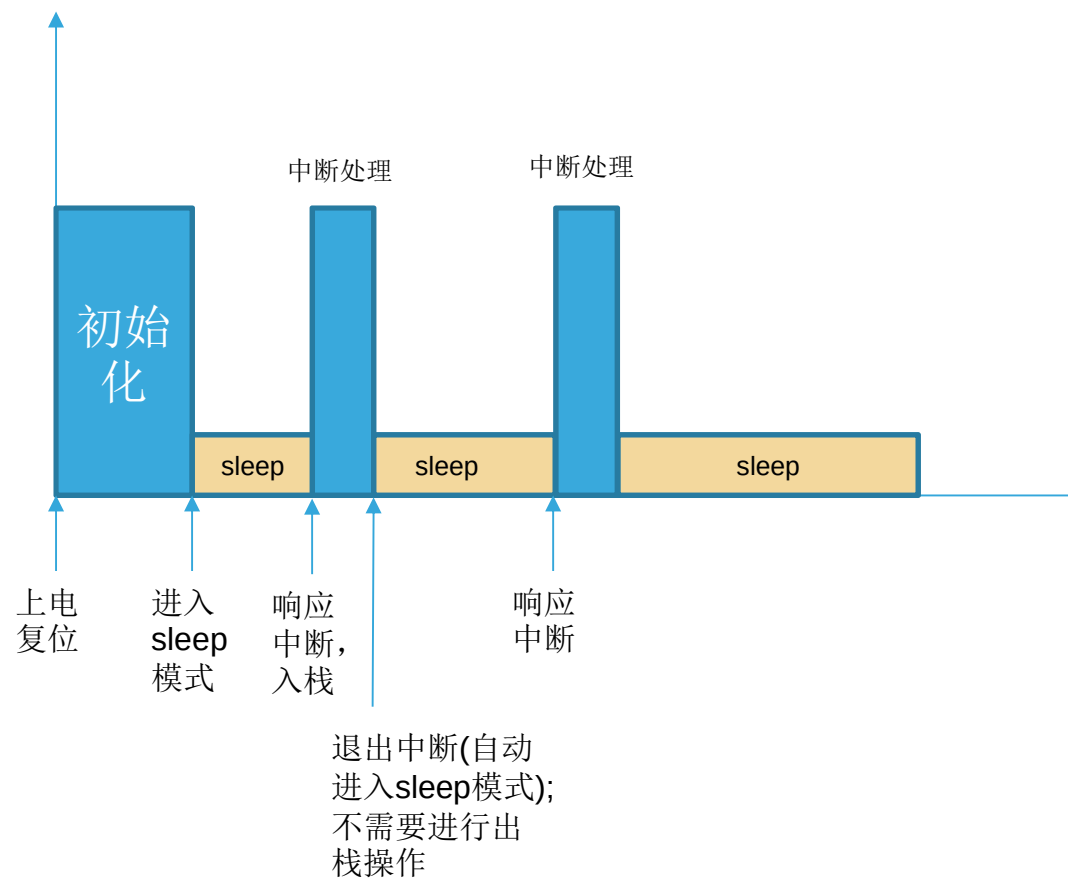
SLEEPONEXIT位置1时，
CPU执行完所有被挂起
的中断服务程序后立即
进入睡眠状态



Sleep-on-exit的应用

83

典型应用： 中断驱动的处理器程序。
主循环里不做任何事情，中断完成后
希望尽快进入低功耗模式。



- 节省中断响应时，出栈入栈消耗的时间。最大程度的让处理器处于低功耗状态

STM32F2运行和低功耗模式比较

84

	状态	进一步降低功耗的措施
运行模式 Run	上电/系统复位后的默认模式，HCLK驱动CPU运行代码	1) 降低系统时钟和所用外设的时钟 2) 关闭不用外设的时钟 RCC_AHB/APB1/APB2ENR
睡眠模式 Sleep	内核时钟停止；外设继续运行	进入之前 1) 降低所用外设时钟 2) 关闭不用外设的时钟 RCC_AHB/APB1/APB2LPENR
停止模式 Stop	1.2V电压域内的时钟全部停止 内部SRAM和寄存器内容仍保持 PLL/HSI/HSE关闭； IWDG/RTC/LSI/LSE都可由用户决定是否运行	进入之前 1) 把电压调节器（VR）配置到低功耗模式 LPDS@PWR_CR 2) 把Flash配置到关闭模式 FPDS @ PWR_CR 3) 关掉ADC/DAC如果不需要 ADON@ADC_CR2 Enx@DAC_CR
待机模式 Standby	电压调节器（VR）关掉→1.2V电压域失电：内部SRAM和寄存器内容丢失； PLL/HSI/HSE关闭； IWDG/RTC/LSI/LSE都可由用户决定是否运行	进入之前 关闭备份SRAM BRE@PWR_CSR

STM32F2低功耗模式实现

85

	进入	退出	唤醒延迟
睡眠模式	执行WFI指令, SLEEPDEEP = 0	NVIC向量表中的任意中断	无中断进出带来的延迟
	执行WFE指令 & SLEEPDEEP = 0 &事件锁存器被清除	唤醒事件*	
停止模式	PDDS=0, SLEEPDEEP=1; 清除所有EXTI线上的等待位以及 RTC中对应的标志; 执行1) WFI或者2) WFE指令	由外部中断/事件唤醒 1)WFI进入→任意EXTI线配置 成中断模式 2)WFE进入→任意EXTI线配置 成中断或事件模式	醒来后HSI作为系统时钟 HIS RC唤醒时间 (6us) + 主电压调节器从低功耗的唤醒 时间 (2.5us) + Flash恢复时间 (100us)
待机模式	PDDS=1, SLEEPDEEP=1 清除WUF@PWR_CSR 清除RTC中和唤醒对应的标志; 执行WFI或者WFE指令	RTC(报警/唤醒/时间戳/侵入事件) WKUP引脚上升沿 NRST引脚上的外部复位 IWDG复位	和复位一样: 采样启动引脚, 获取 向量表的复位矢量...

- [唤醒事件:](#)
 - 在外设寄存器中使能中断, 在NVIC不使能; 置位SEVONPEND@SCB->SCR
 - 唤醒后, 需分别清零外设中断和NVIC向量中的等待(pending)位
 - 把EXTI配置成事件模式
 - 唤醒后无需清除以上两个等待位, 因为它们并未置位
 - **NVIC向量表中的中断也可唤醒***



Sleep-on-exit (SLEEPONEXIT位置1): 只能和WFI一起用

- 存储器系统
- 编程模式
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

存储器保护单元（MPU）

87



提高系统的
可靠性！

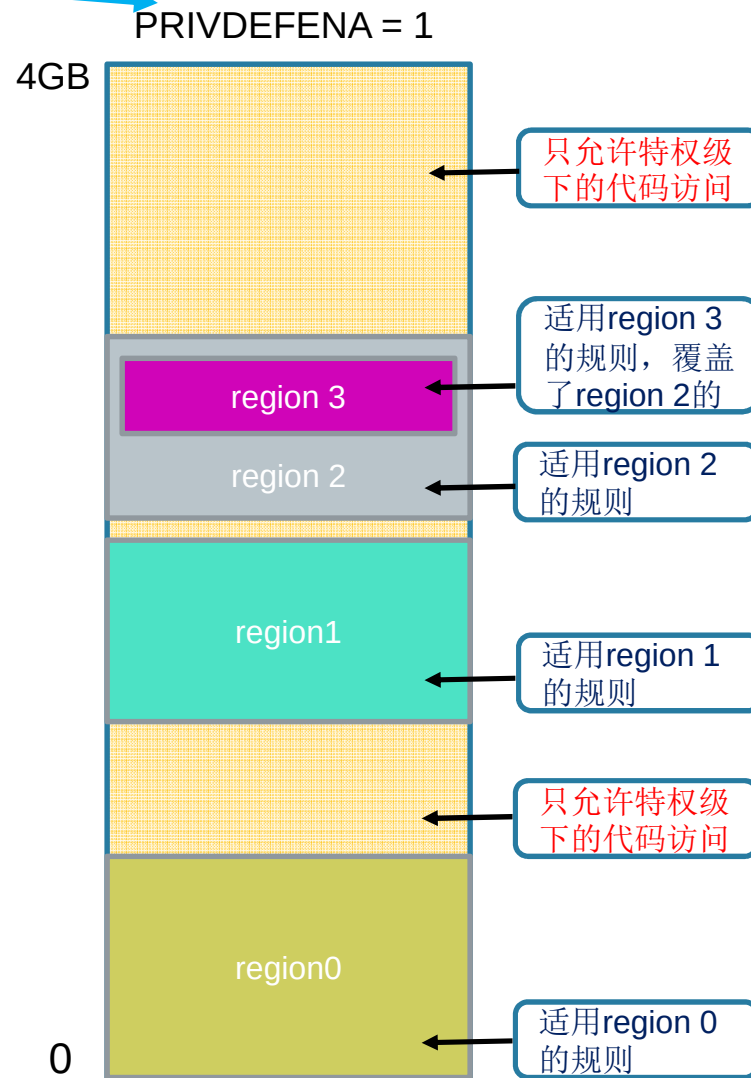
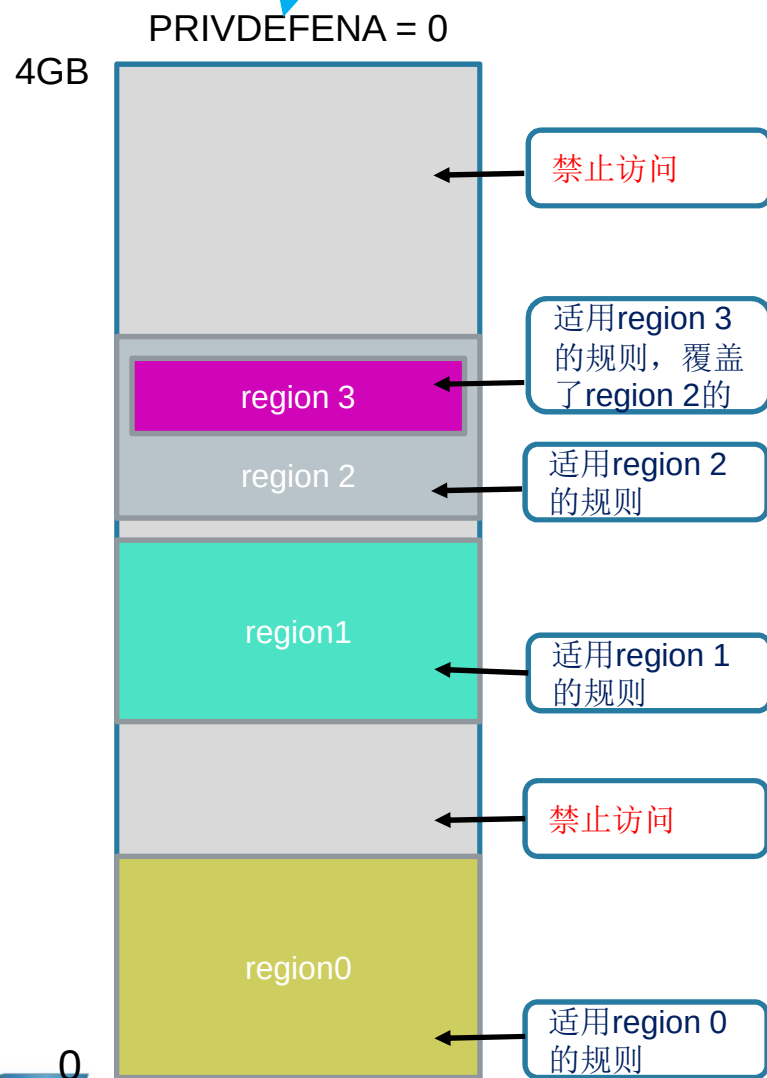
- 阻止用户应用程序破坏操作系统使用的数据
- 阻止一个任务访问其他任务的数据区，从而把任务隔开
- 可以把关键数据区设置为只读，从根本上消除了被破坏的可能
- 检测意外的存储访问，如堆栈溢出和数组越界
- 可以通过MPU设置存储器region的其他访问属性，比如是否缓冲，是否缓存等

- CM3的MPU最多支持8个region。每个region最小容量为32字节。
- 允许启用一个“背景region”（全部地址空间），只能特权级享用。
- 各个region可以相互重叠,重叠部分的属性由编号最大的region来决定。
- 每个region还可等分为8个子region。子region完全继承父region的属性。Region的容量必须大于等于256字节，才能划分子region。
- 每个子region可以独立的使能/除能。
- 若某个子region被除能，而这部分地址范围又没有落在其他region中，则对该子region覆盖的范围进行访问会引发fault。

PRIVDEFENA位在MPUCR寄存器中设置

Region的划分与关系

89



Region的属性

90

- 通过MPURASR寄存器可以为各个region配置不同的属性

位段	名称	类型	描述
31:29	保留	----	----
28	XN	R/W	1: 此区禁止取指 0: 此区允许取指
27	保留	----	----
26:24	AP	R/W	访问许可设置
23:22	保留	----	----
21:19	TEX	R/W	类型扩展
18	S	R/W	1: 可共享; 0: 不可共享
17	C	R/W	1: 可缓存; 0: 不可缓存
16	B	R/W	1: 可缓冲; 0: 不可缓冲
15:8	SRD	R/W	子region禁用设置位段。
7:6	保留	----	----
5:1	REGION SIZE	R/W	(SIZE+1) Region的容量 = 2
0	SZENABLE	R/W	1: 使能此region; 0: 禁止此region

AP 的值	特权级访问许可	非特权级访问许可	典型用法
000	不可访问	不可访问	该区没有存储器，是空地址
001	R/W	不可访问	OS及系统软件使用的数据
010	R/W	RO	禁止在用户级（非特权级）下更改的高危地带
011	R/W	R/W	共享内存，或彻底开放的设备
100	N/A	N/A	N/A
101	RO	不可访问	OS使用的常量数据
110	RO	RO	常量数据或只读存储器的地址区
111	RO	RO	常量数据或只读存储器的地址区

STM32MPU配置例程

91

```
void MPU_Config(void)
{
    /* Disable MPU */
    MPU->CTRL &= ~MPU_CTRL_ENABLE_Msk;

    /* Configure RAM region as Region N?, 8kB of size and R/W region */
    MPU->RNR = RAM_REGION_NUMBER;
    MPU->RBAR = RAM_ADDRESS_START;
    MPU->RASR = RAM_SIZE | portMPU_REGION_READ_WRITE;

    /* Configure FLASH region as REGION N?, 1MB of size and R/W region */
    MPU->RNR = FLASH_REGION_NUMBER;
    MPU->RBAR = FLASH_ADDRESS_START;
    MPU->RASR = FLASH_SIZE | portMPU_REGION_READ_WRITE;

    /* Configure Peripheral region as REGION N?, 0.5GB of size, R/W and
    Execute Never region */
    MPU->RNR = PERIPH_REGION_NUMBER;
    MPU->RBAR = PERIPH_ADDRESS_START;
    MPU->RASR = PERIPH_SIZE | portMPU_REGION_READ_WRITE |
    MPU_RASR_XN_Msk;

    /* Enable the memory fault exception */
    SCB->SHCSR |= SCB_SHCSR_MEMFAULTENA_Msk;

    /* Enable MPU */
    MPU->CTRL |= MPU_CTRL_PRIVDEFENA_Msk |
    MPU_CTRL_ENABLE_Msk;
}
```

1. 配置MPU之前，必须先关闭MPU功能。如果在程序运行中更新MPU的配置，还要关闭中断。
2. 设置各个region号，起始地址，大小和属性。允许重叠，但注意在重叠区域是region号大的那个属性起作用。
3. 开启存储器管理fault异常。如果对某个region执行了不被属性支持的操作，则会产生存储器管理fault异常
4. 重新使能MPU

子region的应用

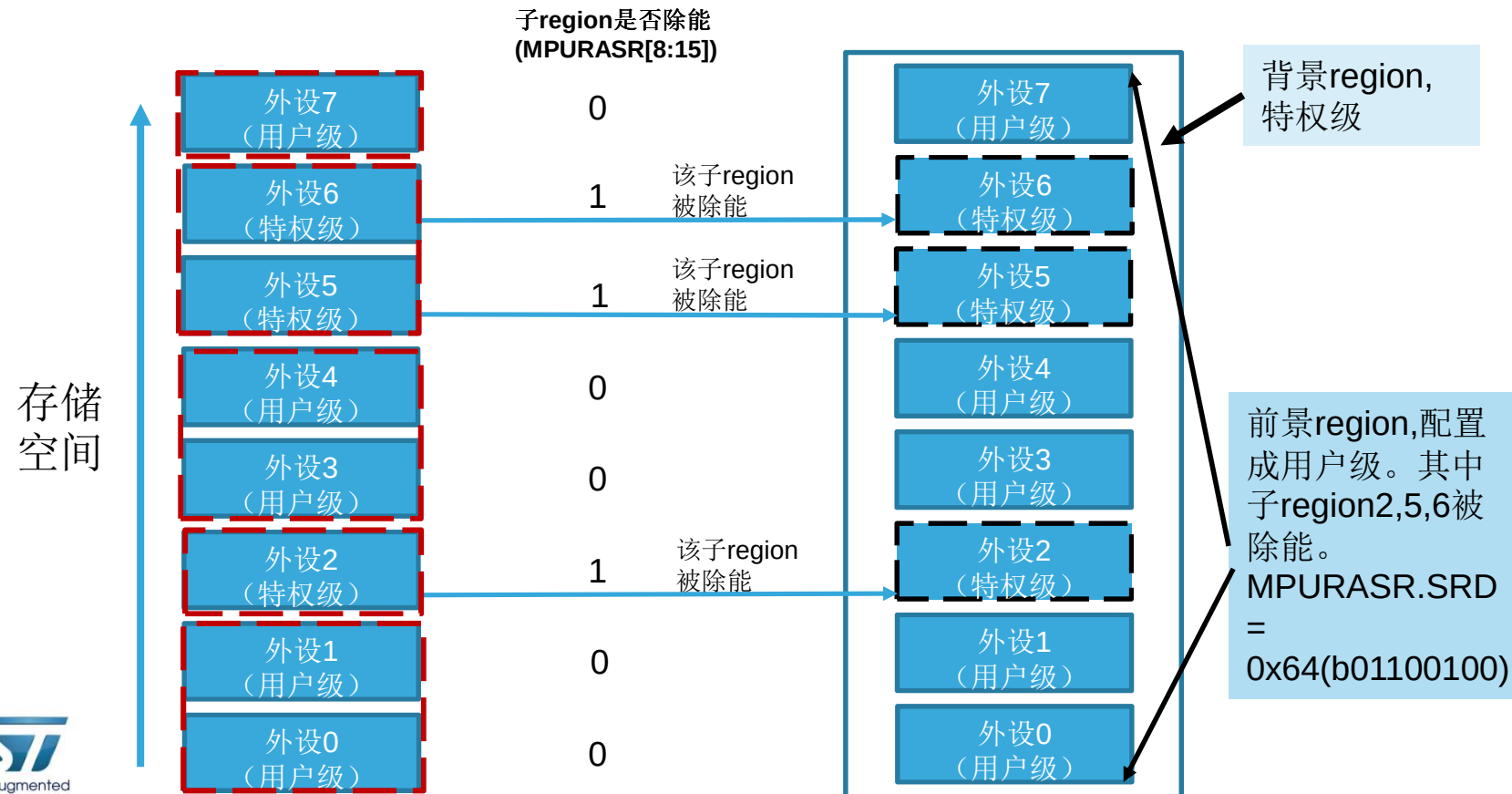
92

外设区中，有些外设是用户级程序可以访问，有些外设只有特权级的程序才能访问（误操作可能导致严重后果）。

实现方法：

- 定义多个用户级外设region
- 在用户级外设region中重叠地定义一个特权级的region
- 在用户级外设region中启用“子region除能机制”

很容易消耗掉8个
可用的region



寄存器缺省访问许可

93

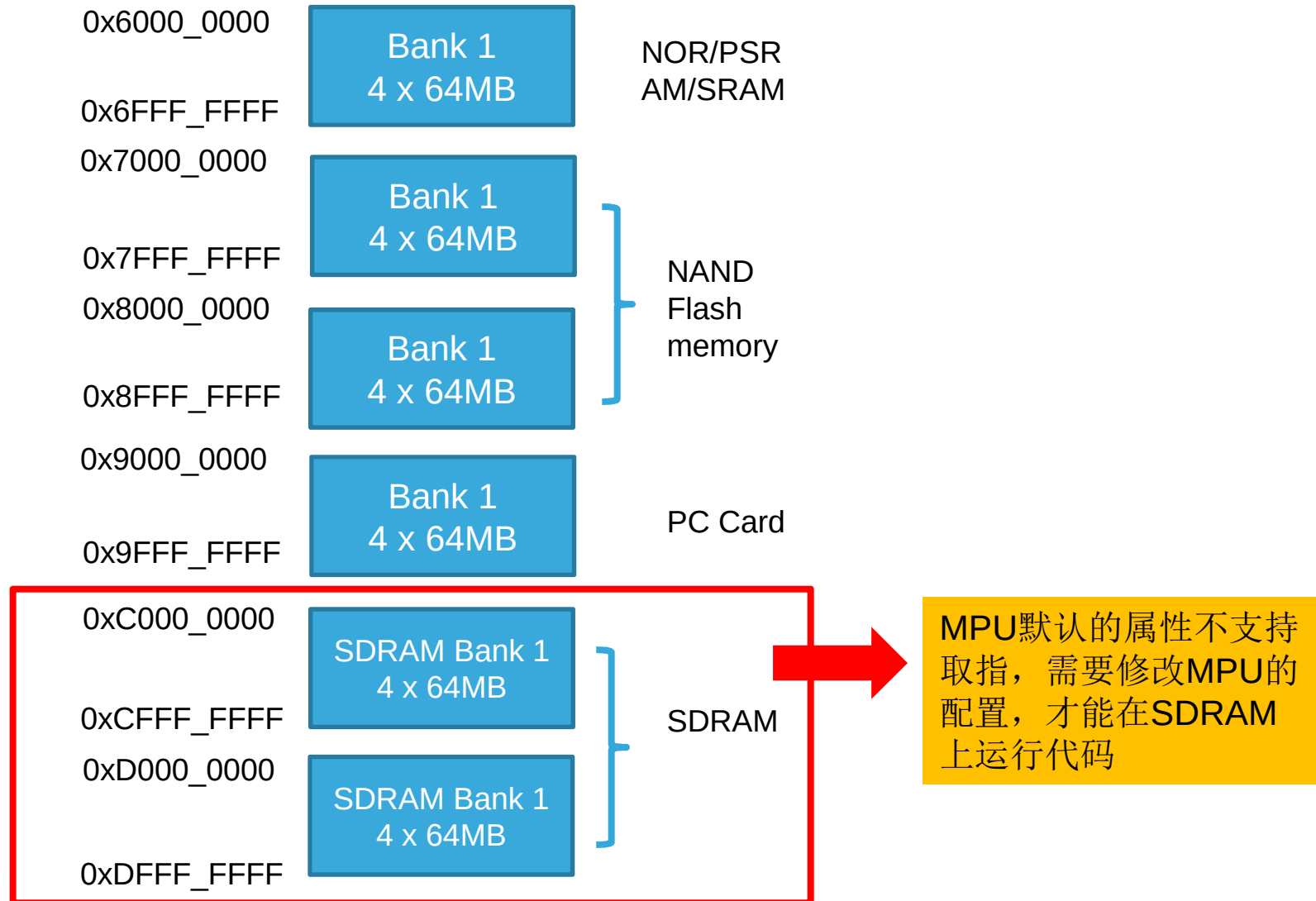
地址范围	存储器区域	可否执行指令 (XN)	描述
0x0000_0000~ 0x1FFF_FFFF	片上代码区	可执行指令	片上闪存
0x2000_0000~ 0x3FFF_FFFF	片上SRAM区	可执行指令	片上RAM
0x4000_0000~ 0x5FFF_FFFF	片上外设区	不可执行指令	用于片上外设的寄存器
0x6000_0000~ 0x9FFF_FFFF	外部RAM区	可执行指令	外部RAM
0xA000_0000~ 0xDFFF_FFFF	外部外设	不可执行指令	用于外部外设的寄存器
0xE000_0000~ 0xE00F_FFFF	私有外设总线	不可执行指令	NVIC, SysTick, MPU等都在这个区域
0xE010_0000~ 0xFFFF_FFFF	供应商指定功能区域	不可执行指令	

什么情况下使用默认设置

- MPU不存在
- 存在MPU但被禁用

STM32外扩SDRAM的默认MPU属性

94



CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED "MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:

- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A "NEED TO KNOW BASIS", PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITTER.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is registered trademark of STMicroelectronics
All other names are the property of their respective owners

© 2015 STMicroelectronics - All Rights Reserved

STMicroelectronics group of companies Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States.

www.st.com