上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

S800板第三次实验(下)

准备知识的学习

- UART
- NVIC

NVIC

Nested Vector Interrupt Controller

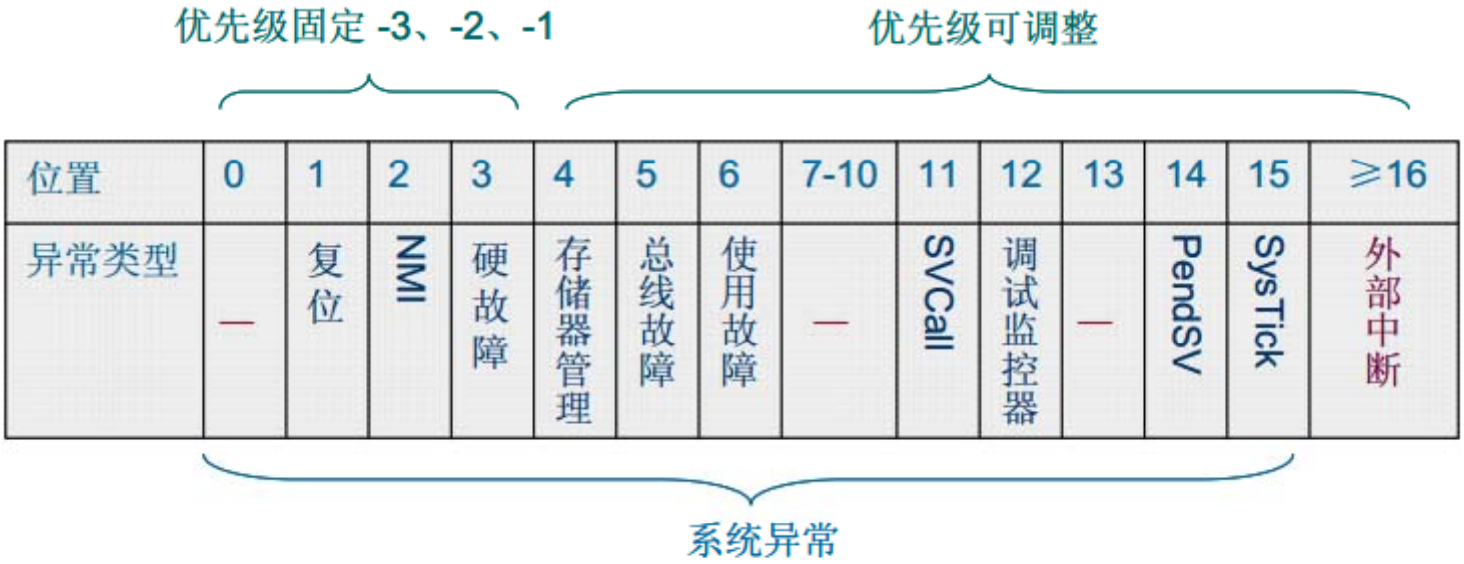
可嵌套的向量中断控制器

Cortex M4中断类型表

No.	Exception Type	Priority	Type of Priority	Descriptions
1	Reset	-3 (Highest)	fixed	Reset
2	NMI	-2	fixed	Non-Maskable Interrupt
3	Hard Fault	-1	fixed	Default fault if other hander not implemented
4	MemManage Fault	0	settable	MPU violation or access to illegal locations
5	Bus Fault	1	settable	Fault if AHB interface receives error
6	Usage Fault	2	settable	Exceptions due to program errors
7-10	Reserved	N.A.	N.A.	
11	SVCall	3	settable	System Service call
12	Debug Monitor	4	settable	Break points, watch points, external debug
13	Reserved	N.A.	N.A.	
14	PendSV	5	settable	Pendable request for System Device
15	SYSTICK	6	settable	System Tick Timer
16	Interrupt #0	7	settable	External Interrupt #0
.....	.....	.....	settable	.....
256	Interrupt#240	247	settable	External Interrupt #240

⌘ Tiva C系列最多支持129个中断类型

异常的优先级


优先级固定 -3、-2、-1
优先级可调整
位置
0
1
2
3
4
5
6
7-10
11
12
13
14
15
≥16
异常类型
—
复位
NMI
硬故障
存储器管理
总线故障
使用故障
—
SVCall
调试监控器
—
PendSV
SysTick
外部中断
系统异常

异常基于优先级而采取的动作

⌘ 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。

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

优先级的编程设置

- 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。

抢占优先级和子优先级

- 抢占式优先级（pre-emption priority)
  - 高抢占式优先级的中断事件可以打断当前的主程序/中断程序的运行——抢断式优先响应，俗称中断嵌套。
- 非抢占式优先级，子优先级（sub priority)
  - 在抢占式优先级相同的情况下，高子优先级的中断优先被响应；
  - 在抢占式优先级相同的情况下，如果有低子优先级中断正在执行，高子优先级的中断要等待已被响应的低子优先级中断执行结束后才能得到响应——非抢断式响应（不能嵌套）。

中断响应的依据

- 判断中断是否会被响应的依据：
  - 首先是抢占式优先级，其次是子优先级；
  - 抢占式优先级决定是否会有中断嵌套；
  - Reset、 NMI、 Hard Fault的优先级为负（高于普通中断优先级）且不可修改。

NVIC的优先级概念

- ARM设计优先级寄存器最多有8位。IC芯片生产商可以自行选择优先级位数。
- 8位分成抢占式优先级（组优先级）和非抢占优先级（子优先级）。
- 子优先级仅仅在抢占优先级相同时才有影响；
- 可编程设定优先级组寄存器中的抢占优先级的位数和非抢占优先级的位数；数字越低，优先级越高；硬件中断数用于区分低级别的优先级次序。

```
IntPriorityGroupingSet（7）        //分配7位给抢占式优先级，1位给子优先级
IntPriorityGroupingSet（0）        //分配0位给抢占式优先级，8位给子优先级
```

Tiva C系列的优先级

- Tiva C系列芯片中优先级有3位，有效位为BIT7、6、5，共8个优先级，即每个中断源有8位中断优先级，从0x00~0x0E0。
- 由于Tiva C系列ARM只实现了3个优先级位，因此实际有效的抢占式优先级位数只有0～3位。
- Tiva C系列，复位默认情况下，抢占式优先级为4位，子优先级为0位，允许中断嵌套。

PRIGROUP	二进制格式	抢占式优先级	子优先级	抢占式可选配置	子优先可选配置
0x0	B.yyy	无	3位	0	0~7
0x1	Bx.yy	1位	2位	0~1	0~3
0x2	Bxx.y	2位	1位	0~3	0~1
0x3-7	Bxxx.	3位	无	0~7	0

PRIGROUP表示分配给子优先级的位数

规则

- 规则1
  - 多个中断源在它们的抢占式优先级相同的情况下，子优先级不论是否相同，如果某个中断已经在服务当中，则其它中断源都不能打断它（可以末尾连锁）；只有抢占式优先级高的中断才可以打断其它抢占式优先级低的中断。
- 规则2
  - 多个中断源在它们的抢占式优先级相同的情况下，同时产生中断，则子优先级高的中断被响应。
- 规则3
  - 多个中断源在抢占式优先级与子优先级均相同的情况下，按中断矢量号的大小为优先级，即中断矢量号小的中断优先被响应，即SYSTICK优先于UART0中断。

举例

- 例：有两个中断源，A中断的中断优先级级置成INTA = b011，即0x60, B中断的中断优先级设置成INTB = b001,即0x20。单单依靠这两个设置我们是无法判断A，B是如何进行中断调度的，我们首先要看中断的组别管理设置。这里我们假设两种不同的组别管理方法，来说明如何分析中断的优先级管理
  - 1) 假设我们设置PRIGROUP = 0x02, 我们按下面来分析中断是如何调度的:

    通过查上面的表我们可以看出，INTn的优先级按照bxx.y来划分

    a. INTA的中断优先级就被划分为 INTA = b01.1

    组优先级 = 01；子优先级 = 1

    b. INTB的中断优先级也被分为 INTB = b00.1

    组优先级 = 00；子优先级 = 1

    由此可见，B的组优先级比A的优先级要高（注意，数字越小，级别越高），B的中断可以打断A的中断处理。

续上页

- 2) 假设我们设置PRIGROUP = 0x01, 我们按下面来分析中断是如何调度的:

  通过查上面的表我们可以看出，INTn的优先级按照bx.yy来划分

  a. INTA的中断优先级被划分成INTA = b0.11.

  组优先级 = 0； 子优先级 = 11

  b. INTB的优先级被划分为INTB = b0. 01

  组优先级 = 0； 子优先级 = 01

  由此可见，A和B处于同一个组优先级，他们两个互相不能打断对方的中断处理。B中断的子优先级高，当两个中断同时发生时，会先进B中断处理，但如果A先发生，在未处理结束前，B是不能打算A进行处理的。

实验例程

```
ui32IntPriorityMask  = IntPriorityMaskGet();
IntPriorityGroupingSet(3);                  //Set all priority to pre-emtption priority

IntPrioritySet(INT_UART0,0);                //Set INT_UART0 to highest priority
IntPrioritySet(FAULT_SYSTICK,0x0e0);        //Set INT_SYSTICK to lowest priority

ui32IntPriorityGroup= IntPriorityGroupingGet();

ui32IntPriorityUart0 = IntPriorityGet(INT_UART0);
ui32IntPrioritySystick= IntPriorityGet(FAULT_SYSTICK);
```

函数IntPriorityMaskGet

⌘ 驱动库手册：P356

功能	获得屏蔽优先级
原型	uint32_t **IntPriorityMaskGet**(void)
说明	该函数获得当前的屏蔽优先级别。返回值是优先级（阈值），因此所有该级别及低优先级将被屏蔽。值0意味不屏蔽。
返回值	返回中断屏蔽优先级的值。

函数IntPriorityGroupingSet

⌘ 驱动库手册：P356

功能	设置中断控制器中的优先级分组。
原型	Void **IntPriorityGroupingSet**(uint32_t **ui32Bits**)
参数	**ui32Bits**指出抢占优先级的位数。
说明	该函数给出在中断优先级寄存器中抢占优先级和子优先级之间的分割。组值的范围取决于硬件；Tiva C系列，硬件中断优先级有3位，因此优先级分组值从3到7的效果是一样的。

函数IntPriorityGroupingGet

⌘ 驱动库手册：P356

功能	获得中断控制器的优先组。
原型	uint32_t **IntPriorityGroupingGet**(void)
说明	该函数返回抢占优先级和子优先级在中断优先级寄存器中的分割。
返回值	返回抢占优先级的位数。

函数IntPrioritySet

⌘ 驱动库手册：P358

功能	设置一个中断的优先级。
原型	Void **IntPrioritySet**(uint32_t **ui32Interrupt**, uint8_t **ui8Priority**)
参数	**ui32Interrupt** 给出中断的名称。 **ui8Priority**给出中断的优先级。
说明	该函数用于设置一个中断的优先级。参数**ui32Interrupt**必须为Peripheral Driver Library User's Guide中列出的有效的INT_*值之一，并且在inc/hw_ints.h头文件中定义过的，如INT_UART0。参数**ui8Priority**给出在中断控制器中的中断的硬件优先级，实验中是从0x00~0x0E0，有效位为BIT7、6、5。0x0优先级最高。

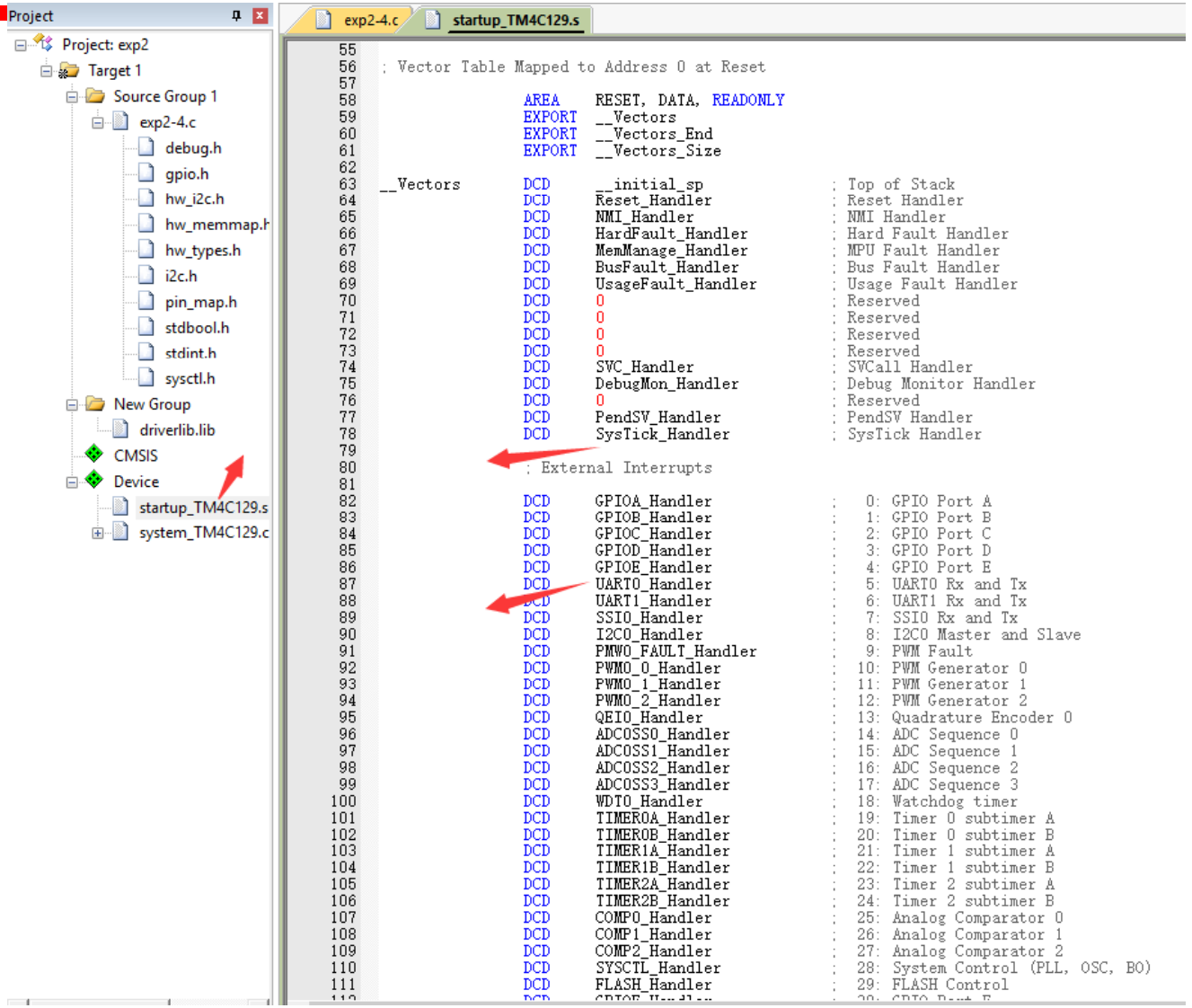
函数IntPriorityGet

⌘ 驱动库手册：P355

功能	获得一个中断的优先级。
原型	int32_t **IntPriorityGet**(uint32_t **ui32Interrupt**)
参数	**ui32Interrupt**给出询问的中断名称。
说明	该函数获得一个中断的优先级。参数**ui32Interrupt**必须是Peripheral Driver Library User's Guide中列出的有效的INT_*值之一，并且在inc/hw_ints.h头文件中定义过的。
返回值	返回给定中断的中断优先级。

中断程序的入口

- 在生成项目时，即自动生成了所有的中断函数的入口函数，在startup_tm4c129.s中。
- 根据所使用的中断类型，按startup_tm4c129.s 中的中断函数名称在主程序中引用此函数，即可实现中断函数。



中断的使能

- 0-15号系统中断是单独设定，16号及以后采用统一的中断使能

  IntEnable 。

- SYSTICK的中断
  - 中断使能

```
SysTickIntEnable();          //systick中断使能
IntMasterEnable();           //总中断允许
```

  - 中断函数，函数名称***SysTick_Handler***

```
void SysTick_Handler(void) {
//在此实现SYSTICK中断函数功能
}
```

中断的使能

- UART的中断
  - UART0中断使能

```
IntEnable(INT_UART0);
UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
                          //Enable UART0 RX,TX interrupt
IntMasterEnable();        //总中断允许
```

  - 中断函数，函数名称***UART0_Handler***

```
void UART0_Handler(void) {
//在此实现UART0中断函数功能
}
```

第三次实验（下）

实验内容

⌘ 实验3-4

- 请编程实现一个虚拟AT 指令集：

当PC 端发来AT+CLASS 后，实验板回以CLASS######，其中######为你的班级号

当PC 端发来AT+STUDENTCODE 后，实验板回以CODE######，其中######为你的学号

- 要求：阅读3-4.c 并理解。

⌘ 实验3-5

- 选做
- 将4 改为大小写均能适应。

实验内容

⌘ 实验3-6

- 选做
- 请编程实现以下三个命令：

底板运行后自动实现1S 计时。

a) PC 端发来绝对对时命令，如SET12:56:03 或12-56-03，自动将当前时间同步到12:56:03，并回之以当前时间

b) PC 端发来相对对时命令，如INC00:00:12，自动将当前时间加12 秒，并回之以当前时间

c) PC 端发来查询命令，GETTIME，自动回之以当前时间

d) 当前时间格式统一为TIME12:56:03，其中TIME 为字符，后续为时间值

实验内容

⌘ 实验3-7

- 优先级调整实验。
- 基于实验3，调整UART0 的优先级，使之高于SYSTICK的优先级，并处于抢占式优先，这样当按下USR_SW2 时，UART0 不退出，导致SYSTICK 中断不能进入，整个系统停滞。显示不再跳变。请阅读3-4.c 并理解。

  改写程序，将SYSTICK 的抢占式优先级高于UART0，从而达到即使USR_SW2 按下时，SYSTICK 中断仍然能进入。

第三次实验讨论题

4. 请根据上位机的命令，如“MAY+01”，格式为：
其中MAY 为月份，（JAN,FEB,...DEC）均为三位。
+表示加运算符,-表示减运算符，均为1 位。
01 表示增加或减少量，均为2 位。范围00-11
以上均为ASCII 码，
MAY+01 应该回之以JUNE
MAY-06 应该回之以NOV

5. 请根据上位机的命令，如“14:12+05:06”，格式为：
其中14:12 为分钟与秒，共5 位，包括一个“:”。
+表示加运算符,-表示减运算符，均为1 位。
05:06 为分钟与秒的变化量，共5 位。包括一个“：”，范围00:00~23:59
以上均为ASCII 码，
14:12+05:06 回之以19:18