Sugar的专栏

如果只得一星期

个人资料

Suagr

+ 加关注 发私信

访问： 94023次
积分： 2221
等级： BLOG 5
排名： 第6566名
原创： 125篇 转载： 3篇
译文： 8篇 评论： 47条

文章搜索

博客专栏

python学习——Learn Beautiful Soup
文章：9篇
阅读：1495

文章分类

ARM7 (17)
嵌入式积累 (11)
电脑鼠征程 (1)
Cortex-M3 (6)
Cortex-M0 (3)
51单片机 (3)
TCP/IP (2)
MSP430 (3)
Live (2)
Python (17)
uc/OS (2)
Linux (29)
eLua (10)
毕业设计之路 (9)
CC2530/Zigbee (5)
CodeCombat (12)
30天自制操作系统 (0)
射雕ZERO (0)

文章存档

2014年10月 (9)
2014年07月 (1)
2014年05月 (1)
2014年03月 (12)
2013年06月 (1)

原 从第二个函数NVIC_PriorityGroupConfig()中了解Cortex-M3的中断

技术问答 快速回复 返回顶部

分类： Cortex-M3 2012-12-06 19:55 2625人阅读 评论(0) 收藏

在第一个函数SystemInit()的下一行，会有常见的另一个函数**NVIC_PriorityGroupConfig（NVIC_PriorityGroup_X）**（X代表数字1，2，3...）。此函数跟中断配置相关，配置中断优先级，包括抢占优先级与子优先级。

关于NVIC（中断向量控制器）的介绍STM32参考手册中是找不到的。需要看**《Cortex-M3权威指南》**，这本书是着重描述此M3内核相关的东西。**NVIC也在其中，因为它是内核中很重要的一部分。**

看了下这本书，对于NVIC描述的很详细。我就直接截图咯。

跟NVIC打交道的寄存器截图说明。操作NVIC，这些寄存器都是需要掌握的，尤其是需要“挂号”的那四个寄存器。

中断配置基础

每个外部中断都在 NVIC 的下列寄存器中“挂号”：

- 使能与除能寄存器
- 悬起与“解悬”寄存器
- 优先级寄存器
- 活动状态寄存器

另外，下列寄存器也对中断处理有重大影响

- 异常掩蔽寄存器（PRIMASK, FAULTMASK 以及 BASEPRI）
- 向量表偏移量寄存器
- 软件触发中断寄存器
- 优先级分组位段

关于中断使能与禁能

这个的话Cortex-M3分别使用了**两个寄存器控制**，一个负责使能与禁能，不像一些简单的单片机一个BIT位就搞定这两个功能，此功能带来的好处就是安全了，不用害怕担心自己的误操作。**因为此寄存器都是写1有效，写0无效**。使能某寄存器则需向使能寄存器写1，禁能需向禁能寄存器写1。

下面的截图说的是关于悬起与解悬寄存器

中断的悬起与解悬

如果中断发生时，正在处理同级或高优先级异常，或者被掩蔽，则中断不能立即得到响应。此时中断被悬起。中断的悬起状态可以通过“中断设置悬起寄存器(SETPEND)”和“中断悬起清除寄存器(CLRPEND)”来读取，还可以写它们来手工悬起中断。

悬起寄存器和“解悬”寄存器也可以有 8 对，其用法和用量都与前面介绍的使能/除能寄存器完全相同，见表 8.2。

中断优先级

每个外部中断的优先级寄存器占8位，但是允许最少只使用最高3位。**STM32的话就是**

使用了最高4位。并且4个相邻的优先级寄存器拼接成一个32位寄存器。中断优先级又分为抢占优先级和子优先级。STM32中，优先级寄存器中高四位中的高两位说明抢占优先级，低两位说明子优先级。抢占优先级的话就是能打断低抢占优先级的中断，从而实现中断嵌套。

下面也是截图看看优先级有哪几种分配情况。

编　号	分配情况	
7	0:4	无抢先式优先级，16 个子优先级
6	1:3	2 个抢先式优先级，8 个子优先级
5	2:2	4 个抢先式优先级，4 个子优先级
4	3:1	8 个抢先式优先级，2 个子优先级
3/2/1/0	4:0	16 个抢先式优先级，无子优先级

下面的截图说明优先级如何确定和嵌套规则。

9．具体优先级的确定和嵌套规则。ARM cortex_m3（STM32）规定

a/ 只能高抢先优先级的中断可以打断低抢先优先级的中断服务，构成中断嵌套。

b/ 当 2（n）个相同抢先优先级的中断出现，它们之间不能构成中断嵌套，但 STM32 首先响应子优先级高的中断。

c/ 当 2（n）个相同抢先优先级和相同子优先级的中断出现，STM32 首先响应中断通道所对应的中断向量地址低的那个中断（见 ROM0008，表 52）。

具体一点：

0 号抢先优先级的中断，可以打断任何中断抢先优先级为非 0 号的中断；1 号抢先优先级的中断，可以打断任何中断抢先优先级为 2、3、4 号的中断；……；构成中断嵌套。

如果两个中断的抢先优先级相同，谁先出现，就先响应谁，不构成嵌套。如果一起出现（或挂在那里等待），就看它们 2 个谁的子优先级高了，如果子优先级也相同，就看它们的中断向量位置了。

关于那几个需要掌握的与中断寄存器，还有一个截图：

2．cortex_m3 内核对于每一个外部中断通道都有相应的控制字和控制位，用于单独的和总的控制该中断通道。它们包括有：

- 中断优先级控制字：PRI_n（上面提到的）
- 中断允许设置位：在 ISER 寄存器中
- 中断允许清除位：在 ICER 寄存器中
- 中断悬挂 Pending（排队等待）位置位：在 ISPR 寄存器中（类似于置中断通道标志位）
- 中断悬挂 Pending（排队等待）位清除：在 ICPR 寄存器中（用于清除中断通道标志位）
- 正在被服务（活动）的中断（Active）标志位：在 IABR 寄存器中，（只读，可以知道当前内核正在处理哪个中断通道）

下面的截图关于中断是如何建立的：

中断建立全过程的演示

下面给出一个简单的例子，以演示如何建立一个外部中断。

1. 当系统启动后，先设置优先级组寄存器。缺省情况下使用组0（7位抢占优先级，1位亚优先级）。
2. 如果需要重定位向量表，先把硬fault和NMI服务例程的入口地址写到新表项所在的地址中。
3. 配置向量表偏移量寄存器，使之指向新的向量表（如果有重定位的话）
4. 为该中断建立中断向量。因为向量表可能已经重定位了，保险起见需要先读取向量表偏移量寄存器的值，再根据该中断在表中的位置，计算出服务例程入口地址应写入的表项，再填写之。如果一直使用ROM中的向量表，则无需此步骤。
5. 为该中断设置优先级。
6. 使能该中断

如果应用程序储存在ROM中，并且不需要改变异常服务程序，则我们可以把整个向量表编码到ROM的起始区域（从0地址开始的那段）。在这种情况下，向量表的偏移量将一直为0,并且中断向量一直在ROM中，因此上例可以大大简化，只需3步：

1. 建立优先级组
2. 为该中断指定优先级
3. 使能该中断

当然还有一些其它跟中断相关的寄存器，不是很常用，就没写在这里了。

那么现在**具体说说这个优先级配置函数**，函数定义实现如下：

```
[cpp] view plain copy print ?
/**
  * @brief  Configures the priority grouping: pre-emption priority and subpriority.
  * @param  NVIC_PriorityGroup: specifies the priority grouping bits length.
  *   This parameter can be one of the following values:
  *     @arg NVIC_PriorityGroup_0: 0 bits for pre-emption priority
  *                                4 bits for subpriority
  *     @arg NVIC_PriorityGroup_1: 1 bits for pre-emption priority
  *                                3 bits for subpriority
  *     @arg NVIC_PriorityGroup_2: 2 bits for pre-emption priority
  *                                2 bits for subpriority
  *     @arg NVIC_PriorityGroup_3: 3 bits for pre-emption priority
  *                                1 bits for subpriority
  *     @arg NVIC_PriorityGroup_4: 4 bits for pre-emption priority
  *                                0 bits for subpriority
  * @retval None
  */
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
{
  /* Check the parameters */
  assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));

  /* Set the PRIGROUP[10:8] bits according to NVIC_PriorityGroup value */
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
}
```

此函数在misc.c原文件中，函数只有两句话。其中

```
[cpp] view plain copy print ?
assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
```

assert_param是定义的一个宏，用来检测表达式的正确性。如果表达式正确则什么也不做，继续执行下面的语句。如果参数有错，就会在当前行报错。这里主要检测我们输入的NVIC配置优先级是否有效。

```
[cpp] view plain copy print ?
/* Set the PRIGROUP[10:8] bits according to NVIC_PriorityGroup value */
SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
```

这句话才是关键，实现了优先级的配置。其中AIRCR_VECTKEY_MASK相当于个钥匙，用一个宏实现。在此源文件的开始处声明，其值为：

```
[cpp] view plain copy print ?
#define AIRCR_VECTKEY_MASK    ((uint32_t)0x05FA0000)
```

因为NVIC是个**很关键的寄存器，不能随便配置，于是需要一个输入标记才能进行正确配置，此标记就相当于一把钥匙**。其中NVIC_PriorityGroup值的选择就是函数上方中那些宏，一共有5种情况。

```
[cpp] view plain copy print ?
/**
  * @brief  Configures the priority grouping: pre-emption priority and subpriority.
  * @param  NVIC_PriorityGroup: specifies the priority grouping bits length.
  *   This parameter can be one of the following values:
  *     @arg NVIC_PriorityGroup_0: 0 bits for pre-emption priority
  *                                4 bits for subpriority
  *     @arg NVIC_PriorityGroup_1: 1 bits for pre-emption priority
  *                                3 bits for subpriority
  *     @arg NVIC_PriorityGroup_2: 2 bits for pre-emption priority
  *                                2 bits for subpriority
  *     @arg NVIC_PriorityGroup_3: 3 bits for pre-emption priority
  *                                1 bits for subpriority
  *     @arg NVIC_PriorityGroup_4: 4 bits for pre-emption priority
  *                                0 bits for subpriority
  * @retval None
  */
```

另外发现现在有很多实时系统都是根据Cortex-M3内核量身订造的，想必其强大的NVIC就是其中一个原因吧。

上一篇 第一个函数SystemInit()里面有些啥

下一篇 数据结构学习(十一)——二叉树的操作

主题推荐 32位 单片机 内核 安全 宏

猜你在找

- STM32F10x 学习笔记7独立看门狗IWDG 模块
- STM32F10x 学习笔记5USART实现串口通讯 1
- OSTimeTick函数解析
- uCOSII移植到友善之臂mini2440
- 网络通信基础
- STM32 开发点滴
- ARM中链接寄存器LR和指令寄存器IR的关系
- MFC判断某路径下的目标文件是否存在
- MFC文件夹打开和文件夹下文件遍历
- atoi函数实现

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap