基于 ARM 架构的可重入异常处理程序的设计与实现

Design and Implementation of repeated Exception Processing Mechanism Based On ARM Structure

秦振涛　杨　茹　（攀枝花学院　四川攀枝花 617000）

摘　要： 本文分析了 ARM 架构异常处理机制，提出了 ARM 可重入异常处理设计的方法和异常处理函数设计，并在 ARM 系统中实现。

关键词： ARM　可重入异常处理　设计与实现

ARM® 系列微处理器作为全球 16/32 位 RISC 处理器市场的领先者，在许多领域内得到了成功的应用。近年来，ARM 在国内的应用也得到了飞速的发展，越来越多的公司和工程师在基于 ARM 的平台上面开发自己的产品。与传统的 4/8 位单片机相比，ARM 的性能和处理能力当然是遥遥领先的，但与之相应，ARM 的系统设计复杂度和难度，较之传统的设计方法也大大提升了。其中异常或中断是用户程序中最基本的一种执行流程或形态，如何对 ARM 架构下可重入异常处理程序的编写是本文要解决的主要问题。

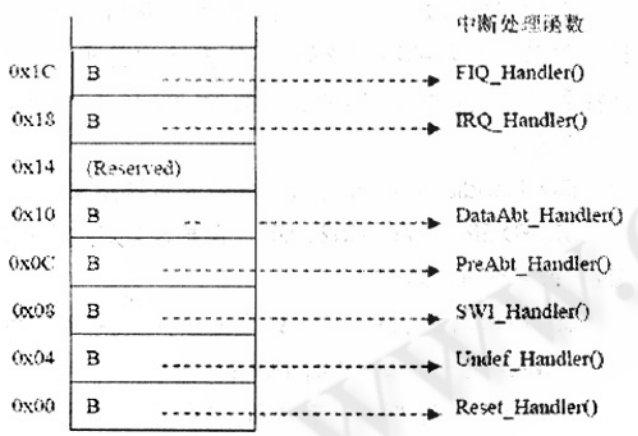

图 1　异常向量表

ARM 一共有 7 种类型的异常，按优先级从高到低排列如下：Reset 、Data Abort 、FIQ 、IRQ 、Prefetch Abort 、SWI 、Undefined instruction.

1　异常响应流程

如以前介绍异常向量表时所提到过的，每一个异常发生时，总是从异常向量表开始起跳的，最简单的一种情况是：

向量表里面的每一条指令直接跳向对应的异常处理函数。其中 FIQ_Handler( ) 可以直接从地址 0x1C 处开始，省下一条跳转指令。但是当执行跳转的时候有 2 个问题需要处理：跳转范围和异常分支。

1.1　跳转范围

我们知道 ARM 的跳转指令（B）是有范围限制的（±32MB），但很多情况下不能保证所有的异常处理函数都定位在向量表的 32MB 范围内，需要大于 32MB 的长跳转，而且因为向量表空间的限制只能由一条指令完成。这可以通过下面二种方法实现。

（1）MOV PC，#imme_value 把目标地址直接赋给 PC 寄存器。

（2）LDR PC，[PC + offset] 把目标地址先存储在某一个合适的地址空间，然后把这个存储器单元上的 32 位数据传送给 PC 来实现跳转。

1.2　异常分支

ARM 内核只有二个外部中断输入信号 nFIQ 和 nIRQ，但对于一个系统来说，中断源可能多达几十个。为此，在系统集成的时候，一般都会有一个异常控制器来处理异常信号。

这时候，用户程序可能存在多个 IRQ/FIQ 的中断处理函数，为了从向量表开始的跳转最终能找到正确的处理函数入口，需要设计一套处理机制和方法。

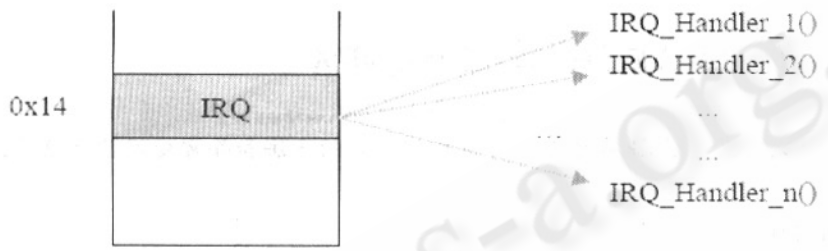

图 2　中断分支

2　异常处理函数的设计

2.1　异常发生时处理器的动作

当任何一个异常发生并得到响应时，ARM 内核自动完成以下动作：

(1) 拷贝 CPSR 到 SPSR_<mode>

(2) 设置适当的 CPSR 位：

① 改变处理器状态进入 ARM 状态

② 改变处理器模式进入相应的异常模式

③ 设置中断禁止位禁止相应中断

(3) 更新 LR_<mode>

(4) 设置 PC 到相应的异常向量

当响应异常后，不管异常发生在 ARM 还是 Thumb 状态下，处理器都将自动进入 ARM 状态。另一个需要注意的地方是中断使能被自动关闭，也就是说缺省情况下中断是不可重入的。单纯的把中断使能位打开接受重入的中断会带来新的问题，除这些自动完成的动作之外，如果在汇编级进行手动编程，还需要注意保存必要的通用寄存器。

2.2　进入异常处理循环后软件的任务

进入异常处理程序以后，用户可以完全按照自己的意愿来进行程序设计，包括调用 Thumb 状态的函数，等等。但是对于绝大多数的系统来说，有一个步骤必须处理，就是要把中断控制器中对应的中断状态标识清掉，表明该中断请求已经得到响应。否则等退出中断函数以后，又马上会被再一次触发，从而进入周而复始的死循环。

2.3　异常的返回

当一个异常处理返回时，一共有 3 件事情需要处理：通用寄存器的恢复、状态寄存器的恢复以及 PC 指针的恢复。

通用寄存器的恢复采用一般的堆栈操作指令，而 PC 和 CPSR 的恢复可以通过一条指令来实现，下面是 3 个例子：

MOVS pc，lr 或 SUBS pc，lr，#4 或 LDMFD sp!，{pc}^

这几条指令都是普通的数据处理指令，特殊之处就是把 PC 寄存器作为了目标寄存器，并且带了特殊的后缀“S”或“^”，在特权模式下，“S”或“^”的作用就是使指令在执行时，同时完成从 SPSR 到 CPSR 的拷贝，达到恢复状态寄存器的目的。

2.4　ARM 编译器对异常处理函数编写的扩展

考虑到异常处理函数在现场保护和返回地址的处理上与普通函数的不同之处，不能直接把普通函数体连接到异常向量表上，需要在上面加一层封装，下面是一个例子：

```
IRQ_Handler ；中断响应，从向量表直接跳来
STMFD SP!，{R0－R12，LR} ；保护现场，一般只需保护{r0－r3，lr} 即可
BL IrqHandler ；进入普通处理函数，C 或汇编均可
LDMFD SP!，{R0－R12，LR} ；恢复现场
SUBS PC，LR，#4 ；中断返回，注意返回地址
```

为了方便使用高级语言直接编写异常处理函数，ARM 编译器对此作了特定的扩展，可以使用函数声明关键字__irq，这样编译出来的函数就满足异常响应对现场保护和恢复的需要，并且自动加入对 LR 进行减 4 的处理，符合 IRQ 和 FIQ 中断处理的要求。

```
__irq void IRQ_Handler (void)
{…}
```

2.5　软件中断处理

软件中断由专门的软中断指令 SWI 触发，SWI 指

令后面跟一个中断编号，以标识可能共存的多个软件中断程序。

这时候 LR 寄存器（因在 IRQ 模式下，是 LR_irq）的值将调整为 BL 指令的下一条指令（ADD）地址，以期能

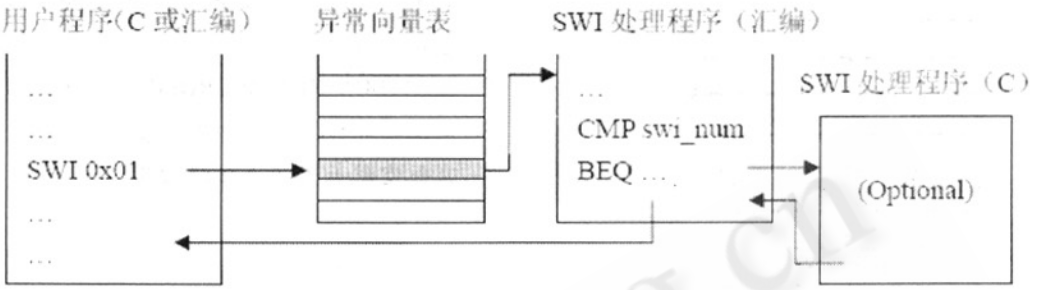

图 3　软件中断处理流程

3　可重入中断设计

如 2.1 节所述，缺省情况下 ARM 中断是不可重入的，因为一旦进入异常响应状态，ARM 自动关闭中断使能。如果在异常处理过程中简单地打开中断使能而发生中断嵌套，显然新的异常处理将破坏原来的中断现场而导致出错。但有时候中断的可重入又是需要的，因此要能够通过程序设计来解决这个问题。其中有二点是这个问题的关键：

（1）新中断使能之前必须要保护好前一个中断的现场信息，比如 LR_irq 和 SPSR_irq 等，这一点容易想到也容易做到。

（2）中断处理过程中对 BL 的保护在中断处理函数中发生函数调用（BL）是很常见的，假设有下面一种情况：

```
IRQ_Handler:
...
BL Foo - - - - - - > Foo:
ADD ··· STMFD SP!, {R0 - R3, LR}
... ...
LDMFD SP!, {R0 - R3, PC}
```

上述程序，在 IRQ 处理函数 IRQ_Handler( ) 中调用了函数 Foo( )，这是一个普通的异常处理函数。但若是在 IRQ_Handler( ) 里面中断可重入的话，则可能发生问题，考察下面的情况：

当新的中断请求恰好在“BL Foo”指令执行完成后发生。

从 Foo( ) 正确返回；但是因为这时候发生了中断请求，接下去要进行新中断的响应，处理器为了能使新中断处理完成后能正确返回，也将进行 LR_irq 保存。因为新中断是在指令流

BL Foo - - > STMFD SP!, {R0 - R3, LR}

执行过程中插入的，完成跳转操作后，进行流水线刷新，最后 LR_irq 保存的是 STMFD 后面一条指令的地址；这样当新中断利用（PC = LR - 4）操作返回时，正好可以继续原来的流程执行 STMFD 指令。这二次对 LR_irq 的操作发生了冲突，当新中断返回后往下执行 STMFD 指令，这时候压栈的 LR 已不是原来需要的 ADD 指令的地址，从而使子程序 Foo( ) 无法正确返回。

这个问题无法通过增加额外的现场保护指令来解决。一个巧妙的办法是在重新使能中断之前改变处理器的模式，也就是使上面程序中的“BL Foo ”指令不要运行在 IRQ 模式下。这样当新中断发生时就不会造成 LR 寄存器的冲突了。考虑 ARM 的所有运行模式，采用 System 模式是最恰当的，因为它既是特权模式，又与中断响应无关。

所以一个完整的可重入中断应该遵循以下流程：

4　结束语

可重入异常中断处理的设计非常重要，关系到设备驱动和实响应等方面的性能。开发嵌入式系统无法回避系统底层的编程设计，因此在异常分支、异常返回、软中断等方面都需要特别注意。下面是一个实现的例程。

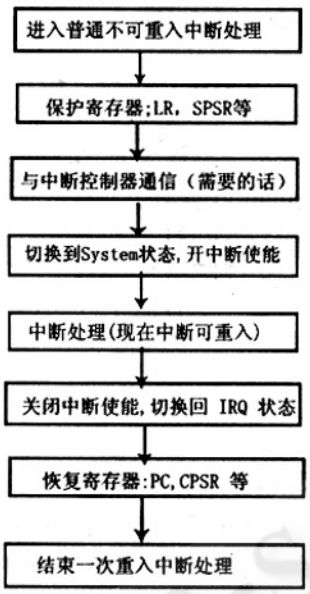

图 4　可重入中断处理流程

参考文献

1 ARM7TDM1 Technical Reference Manual 2000. http://www.arm.com/arm/documentation open documentation

2 S3C44BOX RISC MICROPROCESSOR SUMSUNGCO.

3 周立功等,ARM 微控制器基础与实践[M],北京:北京航空航天大学出版社,2003.


```
IRQHanlder
    SUB     lr, lr, #4
    STMFD   sp!, {lr}
    MRS     r14 , SPSR
    STMFD   SP!, { r12, r14 }

    MOV     r12, #IntBase
    LDR     r12 ,[r12,#IntSource]

    MRS     r14, CPSR
    BIC     r14, r14, #0x9F
    ORR     r14, r14 ,#0x1F
    MSR     CPSR_c, r14

    STMFD   sp!, {r0-r3, lr}
    MOV     r0,r12
    BL      C_irq_handler
    LDMFD   SP!, {r0-r3, lr}

    MRS     r12 , CPSR
    BIC     r12, r12 ,0XIF
    ORR     r12,r12, 0x92
    MSR     CPSR_c, r12

    LDMFD   SP!, {r12, r14}
    MSR     SPSR_csxf, r14
    LDMFD   SP!, {PC}^
```

读/清中断控制器中断源

切换到System模式同时使能IRQ

切换到IRQ模式同时禁止IRQ

LR irq:SPSR irq 和工作寄存器 (r12) 压栈保护来避免下一次中断发生使它们被冲掉

保存R0-3，LR_user到user 栈中，然后调用C子程序，中断源（R0）作为一个参数传入 C处理函数。

恢复LR irq;SPSR irq 和二作寄存器（r12),然后退出中断处理使用修正后的LR

图 5