中断处理设计在嵌入式系统开发中起着非常重要的作用。本文以ARM EP7312为对象，研究ARM体系的中断处理程序设计和调试技术，对程序设计涉及的系统引导和MMU使用方法也进行了详细描述。

设计ARM的中断处理

■ 李尚锋 高金山 朱韦伟 梁志毅

中断技术在对实时性要求较高的嵌入式系统中占有重要的地位。中断处理程序设计是应用程序、设备驱动程序设计的必要环节，对嵌入式系统开发起着重要作用。本文将以ARM EP7312为对象，重点说明ARM体系的中断处理程序设计和调试技术，同时对程序设计中涉及的系统引导和MMU使用方法进行了详细的描述。

ARM体系及中断处理

首先来了解ARM体系及中断处理的相关概念。

1. ARM处理器模式

ARM处理器包括用户(User)模式、快速中断模式、外部中断模式、特权模式、中止模式、未定义指令模式和系统模式等七种模式。除用户模式外，其他六种处理器模式被称为特权模式。大多数用户程序都运行在用户模式下，此时应用程序不能访问一些受系统保护的系统资源。应用程序不能直接进行处理器模式的切换，当需要进行切换时，应用程序可以产生中断异常，在中断异常处理过程中进行处理器模式的切换。

当应用程序发生中断异常时，处理器进入相应的异常模式。每一种异常模式都有一组寄存器，供相应的异常处理程序使用，可以保证在进入异常模式时用户模式下的寄存器不被破坏。在任意时刻，可见的寄存器包括15个通用寄存器，1～2个程序状态字寄存器及程序计数器寄存器等。

2. 中断向量表及中断优先级

中断向量表指定各中断及其相应处理程序的对应关系。ARM体系结构要求中断向量表放在0开始的32字节的连续地址空间中。每个中断向量占据4个字节，在这4个字节的空间中存放一个跳转指令。当中断发生时，PC通过跳转指令跳转到相应的异常中断处理程序处执行。当多个中断同时发生时，处理器必须按照一定的次序来处理这些中断。中断向量表中各中断地址及中断优先级如表1所示。

3. ARM处理器对异常中断的响应

ARM处理器对异常中断的响应过程如下：

(上接第90页)等功能。ETS API包括硬件模拟功能，可使应用软件开发人员针对某一接口编写的代码应用于其他接口。

Certicom Security Architecture的内置式硬件加密技术和密钥管理功能还可以大大方便这一可靠硬件平台的建立。安全启动技术及密码功能将共同搭建起一个可靠的运行环境。该软件通过一个普通的API，充分利用了Intel芯片及其他设计元素。只要开发人员熟悉某种API，他们便可将其应用于同芯片家族的所有模块中。

总的来说，那些受困于嵌入式设备安全问题的软件开发人员们将会发现，建立一个可靠平台是确保系统安全的前提条件。如果最低级别的代码容易受到攻击，那么整个设备便容易受到攻击。这种威胁不仅会造成经济损失，还可能威胁到公共信息安全。软件开发人员完全可以应用DRM建立自己的可定制可靠平台，这不仅可以节省大量资金，而且更加有针对性。

(1)保存处理器当前状态、中断屏蔽位及各条件标志位；

(2) 设置当前程序状态寄存器CPSR相应的位，使处理器进入相应的执行模式；

(3)将寄存器LR设置为返回地址；

(4) 将程序计数器的值设置成该异常中断的中断向量地址，从而跳转到相应的异常中断处理程序处执行。

4. ARM处理器异常中断返回过程

(1) 恢复被中断程序的处理器状态，即将spsr_mode寄存器的内容复制到当前程序状态寄存器中。

(2) 返回发生异常中断指令的下一条指令处执行，即将lr_mode寄存器的内容复制到程序计数器PC 中。

中断程序的设计方法

这里采用的开发平台是基于EP 7312的ARM 7，编程环境为ARM SDT 2.5。其中，平台支持MMU，有16MB的SDRAM和4MB的Flash ROM。由于Flash的擦除和烧写比较麻烦，所有程序均通过SDT 2.5加载到SDRAM中。考虑到中断向量表必须加载到零地址，这里通过设置MMU表使SDRAM的某段地址映射到零地址处，从而实现中断时从SDRAM访问中断向量表。

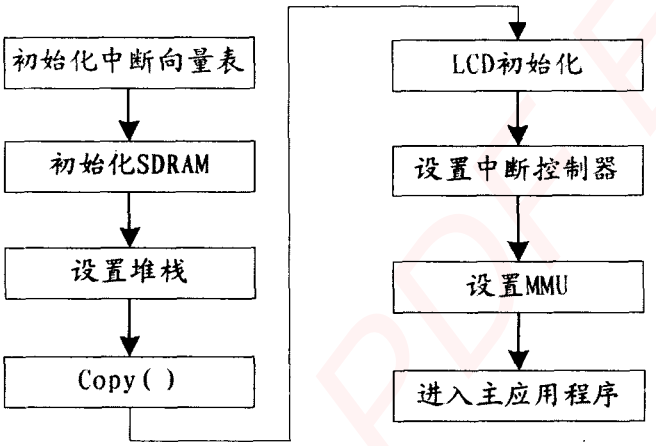

图1 程序设计流程

程序的设计流程如图1所示，分为初始化、主应用程序、普通中断处理程序和快速中断处理程序四个部分。

首先初始化C程序的运行环境，并设置好中断向量表，MMU使能后，进入主应用程序“跑马灯”。当键盘有键按下时产生普通中断，执行完普通中断后返回主应用程序继续执行。如果普通中断有fiq，则跳入fiq处理程序，执行完毕后跳转回普通中断程序，普通中断程序执行完毕后再返回主应用程序。对于硬件和中断向量表的初始化都采用ARM汇编实现，对于应用程序和中断处理程序则采用C语言实现。下面结合程序分析各个主要部分功能的实现。

1. 初始化部分

(1) 中断向量表的设置

```
B       Reset_Handler
B       Undef_Handler
B       Swi_Handler
```

表1 中断向量表中各中断地址及中断优先级

中断向量地址	异常中断类型	异常中断模式	优先级（6最低）
0x0	复位	特权模式	1
0x4	未定义的指令	未定义指令模式	6
0x8	软件中断	特权模式	6
0x0c	指令预取中止	中止模式	5
0x10	数据访问中止	中止模式	2
0x14	保留	未使用	未使用
0x18	外部中断请求	外部中断模式	4
0x1c	快速中断请求	快速中断模式	3

```
B       Pre_Handler
B       Dabort_Handler
NOP
B       IRQ_Handler   ;irq中断地址0x18
```

fiq中断的起始地址为0x1c，该地址不同于其他中断，不是跳换指令，而是将现场寄存器保存到相应模式的堆栈里面：

```
STMFD   SP!,{r14}
MSR     R14,SPSR
STMFD   SP!,{R12,R14}
STMFD   SP!,{R0-R11}
```

保存好现场后，跳转到相应的处理程序里面：

```
IMPORT  fiq_interrupt
BL  fiq_interrupt
```

执行完快速中断处理程序后返回，将寄存器的值一一恢复：

```
LDMFD   SP!,{R0-R11}
LDMFD   SP!,{R12,R14}
MSR     SPSR_CXSF,R14
LDMFD   SP!,{PC}^
```

对于irq_handler保存现场的操作同fiq相同，都是把通用寄存器的值保存到相应模式的堆栈里面。

（2）初始化SDRAM

设定SDRAM的数据宽度、CAS延迟和刷新率等。

（3）定义SP堆栈地址

对程序中需要用到的每一种模式给SP定义一个堆栈地址。通过改变状态寄存器的状态位，把处理器切换到不同的状态，然后给SP赋值。注意，不要切换到User模式进行User模式的堆栈设置。因为进入User模式后就不能再操作CPSR回到其他模式，可能会对接下去的程序执行造成影响。下面给出普通中断的堆栈设置：

```
MOV R0,#MODE_IRQ:OR:IRQ_DIS_BIT:OR:FIQ_DIS_BIT
MSR CPSR_c,R0
LDR R13,=IRQ_STACK
```

（4）初始化LCD

配置LCD使能寄存器、LCD控制寄存器和LCD调色板寄存器，并设置LCD显存在系统内存的起始地址。

（5）设置与键盘中断相关寄存器

设置键盘中断使能寄存器和屏蔽寄存器，并设置当前运行模式的I位和F位，允许发生快速中断和普通中断。

（6）设置MMU

copy函数的作用是把SDRAM的中断向量表拷贝到0xc0a0.0000处。MMU的作用就是进行地址重映射。由于ARM 7的处理器在中断发生时强制把PC指针置为向量表中对应中断类型的地址值，要求中断向量表必须放置在从0地址开始的连续8×4字节空间内，所以需要对地址空间进行重映射。

整个程序都是在SDRAM中完成，因此对ARM 7平台的SDRAM地址分配为0xc000.0000—0xc0ff.fffc，共16MB。其中0xc000.0000—0xc07f.fffc作为LCD的显存；0xc080.0000—0xc09f.fffc存放程序段；0xc0a0.0000—0xc0bf.fffc存放中断向量表；0xc0c0.0000—0xc0fb.fffc作为各个模式的堆栈；0xc0fc.0000—0xc0ff.fffc存储MMU表。图2为MMU使能前后SDRAM的地址分配图。当MMU使能后，虚拟地址0x0000.0000—0x001f.fffc将对应SDRAM的0xc0a0.0000—0xc0bf.fffc地址段。MMU使能后，PC指针首先通过MMU表把虚拟地址转换为实际地址，然后从实际地址处执行。

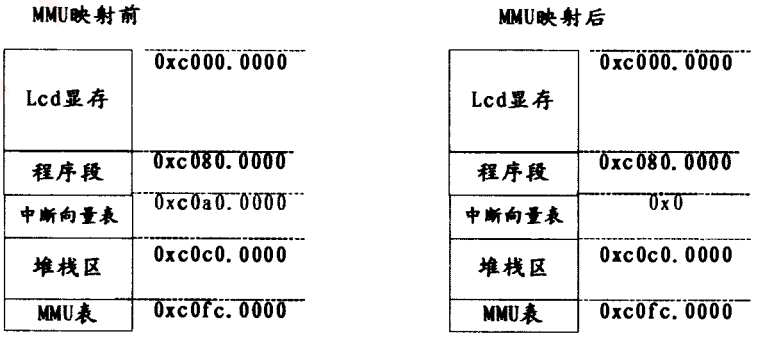

图2 SDRAM在MMU使能前后的地址分配图

实现程序如下：

```
low_20_bit=0xc02;
/* 设置MMU表的基址为0xc0fc.0000*/
base_addr=(unsigned char*)
```

```
0xc0fc0000;
    /*MMU物理地址和虚拟地址相同的操作*/
    for(i=0;i<0x1000;i++)
    {
        high_12_bit=i;
       high_12_bit=high_12_bit << 20;
            the_all_data=high_12_bit|low_20_bit;
        *(volatile unsigned int*)base_addr=the_all_data;
        base_addr+=4;
    }
    /*0x0000 - 0x001f.fffc地址空间与0xc0a0.0000—0xc0bf.fffc地址的互换*/
    *(volatile unsigned int*)0xc0fc3028=0x00000c0a;
    *(volatile unsigned int*)0xc0fc302c=0x00100c0a;
    *(volatile unsigned int*)0xc0fc0000=0xc0a00c02;
    *(volatile unsigned int*)0xc0fc0004=0xc0b00c0a;
```

2. 主程序

主程序主要对一个端口寄存器进行初始化，对其数据端口进行循环赋值，与端口寄存器相连的LED指示灯会循环点亮。

3. irq中断处理程序

键盘中有无键按下是由列线和行线的状态决定。其中键盘为4×4的键盘矩阵，行线通过上拉电阻接地。初始化时，行线端口设置为输入，输入全是0；列线全部设置为高。当按键按下时，该按键所对应的行即相应行线断口的电压变为高，表示有键按下。若此时使能键盘中断，则引发键盘中断。

在中断处理程序中，按键为哪个键的判定步骤是：把列线依次置高，然后读入行线状态，如果行线全为0，则无键按下；如果不为零，则必有键按下且所按键在高电平列中，根据行、列交叉点判断按下的具体键。键盘上的每一个键都有一个键值，键值由程序事先设定。

当键盘有按键按下时，处理器模式跳转到irq模式，保存好现场后，进入中断处理程序。中断处理程序在完成清除中断请求后，判断是哪个按键按下，并在LCD上显示相应按键。

```
    *(volatile unsigned int*)0x80001700=0xffffffff;
    /*清除中断请求标志*/
    keyvalue = keyscan(i);
    /*判断是哪个键按下，给出键值，然后根据键值在LCD上显示不同的图像*/
    switch(keyvalue)
    {
        case '0' :LCD_zero();
        break;
        case '1' : LCD_one();
        break;
        ……
        ……
        case 'e' : LCD_fourteen();
        break;
        case 'f' : LCD_fifteen();
        break;
    }
```

4. fiq中断处理程序

将平台上的一个开关做为控制快速中断的控制键。当开关由开变关时，输入引脚电平就会由高变低，从而引发快速中断。快速中断的引入只是为了说明在多个中断发生时，优先级高的先得到执行。

fiq的处理程序很简单，主要通过向超级终端发送字符串“this is fiq interrupt”，来验证进入快速中断。

程序编译前要先在“ARM Linker”→“Output”设置“ROBASE = 0Xc080.0000”，这样保证了链接时地址0xc080.0000是希望执行的第一条指令。编译生成.axf格式文件后就可以直接加载到SDRAM中，单击连续执行，就可以看到LED指示灯以计数器的形式点亮。单击任一键盘按键，LCD上就会有相应按键显示。

引入快速中断的目的是为了验证快速中断的优先级比普通中断的优先级高，所以在键盘中断处理程序中加上延迟10秒，然后在LCD上显示。如果在跳转到键盘中断但没有显示在LCD的时间段内按下fiq触发键，则会看到超级终端有字符串输出，然后才是LCD上有相应按键的显示。

本文主要讨论在EP 7312上的键盘中断过程，对于其他中断过程是一样的。在键盘中断的实现过程中引入MMU对地址重映射，地址重映射对固化在Flash里的程序非常重要。当程序从Flash拷贝到SDRAM后，设定MMU把Flash和SDRAM的地址空间交换后，可以实现程序自动跳转到SDRAM中执行。