实验 6 ARM 裸机中断程序

实验目的

- 学会 S3C6410 裸机状态下，中断程序的编写、编译、运行方法

实验设备

- 硬件：PC 机+tiny6410 开发板+SD 卡
- 软件：WINXP+VMWARE 虚拟机+UBUNTU1004-32bit+arm-linux-gcc。

实验内容

- 1.使用 S3C6410 的 VIC_port 中断模式，编写中断程序，实现如下功能：按下开发板上的 key1~key4 时，分别点亮 LED1~LED4

实验原理

参考课程第 6 章，中断部分

实验步骤

1.在虚拟机 LINUX 交叉开发环境下编写、编译源程序。

1) 分别编辑所需源程序 Start.S （注意：这里的 ARM 汇编程序采用的是 GNU 风格，使用的是 GNU 编译器）、irq.c、led.c、main.c、irq.h

2) 编写 Makefile 文件（注意：编辑 Makefile 文件时，不要把本实验指导书的代码复制过去，这涉及到 WINDOWS 和 LINUX 文本编码的问题，所以直接复制可能会出现错误。）

3) 执行 Make，对上述源程序编译，生成可执行的 irq.bin 文件

2.硬件开发板连线

1）把开发板的 USB 口和 PC 机的 USB 口用开发板附带的 USB 线缆连接起来。

2）把电源线连接到开发板（连接后先不要开机），把开发板的 S2 开关拨到 SDBOOT 一侧。

3.把 irq.bin 文件烧写到 tiny6410 开发板

1）安装 minitools 烧写工具（MiniToolsSetup-Windows-20140317.exe），安装完成 minitools 后，上电启动开发板，这时主机会提示安装驱动程序，按照提示安装好驱动即可。参考“03-Tiny6410 刷机指南.pdf”的 P28~P29 的 2.1.1 节。

2）把 irq.bin 文件从虚拟 LINUX 系统拷贝到 WINDWOS 系统。在 irq.bin 所在目录下执行如下指令：

cp irq.bin /mnt/hgfs/share

执行成功后，就可以在 WINDOWS 系统“D:\VM_OS\ub100432bit\share”看到 irq.bin 文件。

3）然后使用 minitools 工具烧写 irq.bin 到开发板，参考“Linux 平台下 Tiny6410 裸机程

序开发指南”的 P10，“烧写运行”的“方式一”。注意：绝对不要使用“方式二”!!!!!!!!

注意：有时候 minitools 会发现有卡死的现象。解决方法是:第一安装完 minitools 和驱动之后，先把 USB 口线拔下来重插一次，并且要先启动开发板，后启动 minitools。先启动 minitools，后启动开发板，容易造成 minitools 卡死。

4.irq.bin 运行后，分别按下 key1~key4，观察 LED1~LED4 的点亮情况。

参考源程序

1. start.S 源程序：

```
.global _start
.global do_irq
.extern key_isr

_start:
 //把外设基地址告诉 CPU
 ldr r0, =0x70000000
        orr r0, r0, #0x13
        mcr p15,0,r0,c15,c2,4

 //开启 S3C6410 的中断 VIC_port 功能
 //开启此功能，发生中断时，CPU 会自动
 //跳到 ISR，执行 ISR。这种跳转操作
 //是由硬件自动执行的
        mrc p15,0,r0,c1,c0,0
 orr r0,r0,#(1<<24)
 mcr p15,0,r0,c1,c0,0

 // 关看门狗
 ldr r0, =0x7E004000
 mov r1, #0
 str r1, [r0]
```

```
// 设置栈
ldr sp, =0x0c002000

//初始化 LED
bl led_init
// 初始化 irq
bl irq_init

//开启中断总开关，设置 CPSR 的 I 位为 0
mrs r0,cpsr
bic r0,r0,#0x80
msr cpsr_c,r0

bl main//跳转到主函数

//do_irq 函数是中断服务函数，首先保存现场
//然后跳转到 key_press 按键判断程序
//最后恢复现场
do_irq:
ldr sp, =0x54000000     //发生 IRQ 中断后，CPU 自动切换到 IRQ 模式下
                        //sp 是分组寄存器，因此关看门狗后设置的
                        //栈不能使用，这里要重新设置栈，供保存、
                        //恢复现场使用

sub lr, lr, #4          //lr 存放的是发生中断时那条指令的下一条指令
                        //恢复现场时，保证能正常返回到主程序，
                        //这里 lr 应减去 4

stmdb sp!, {r0-r12, lr}   //保存现场
bl key_isr                //跳到 key_isr 程序，
ldmia sp!, {r0-r12, pc}^ //恢复现场，^表示把 SPSR 恢复到 CPSR

halt:
b halt
```

2 irq.c 源程序

```
#include "irq.h"

extern unsigned long do_irq;

void key_isr(void)
{
 unsigned long key_press=0;
```

```
key_press=(EINT0PEND & 0xf);//读 EINT0PEND[3:0]，判断哪个键按下

switch(key_press)
{     //key1 按下
      case 1:
      {
            led_display(LED_ALL_OFF);
            led_display(LED1_ON);
            break;
      }
      //key2 按下
      case 2:
      {
            led_display(LED_ALL_OFF);
            led_display(LED2_ON);
            break;
      }
      //key3 按下
      case 4:
      {
            led_display(LED_ALL_OFF);
            led_display(LED3_ON);
            break;
      }
      //key4 按下
      case 8:
      {
            led_display(LED_ALL_OFF);
            led_display(LED4_ON);
            break;
      }

      default:
            break;

}

//清中断
EINT0PEND      = 0xf;
VIC0ADDRESS = 0;
}

void irq_init(void)
{
```

```
    /* 配置 GPN0~3 引脚为中断功能 */
    GPNCON &= ~(0xff);
    GPNCON |= 0xaa;

    /* 设置中断触发方式为: 下降沿触发 */
    EINT0CON0 &= ~(0xff);
    EINT0CON0 |= 0x22;

    /* 开启中断 */
    EINT0MASK &= ~(0xf);

    VIC0INTENCLEAR |= (0x1);
    VIC0INTSELECT &= (~(0x1));
    VIC0ADDRESS = 0;

    //中断服务函数入口地址赋值给 VIC0VECTADDR0
    //中断发生时，CPU 会自动到 VIC0VECTADDR0 读取
    //中断服务程序的入口地址，执行中断服务函数
    //配置 VIC_PORT 使能模式下，中断发生不会进入
    //异常向量，避免使用 sys_bus 模式处理中断
    VIC0VECTADDR0 = (unsigned long)&do_irq;

    /* 在中断控制器里使能这些中断 */
    VIC0INTENABLE |= (0x1); /* bit0: EINT0~3 */
    }


    /*void do_irq(void)
    {
    //此段代码是 sys_bus 模式下的中断处理方式
    //此种方式在 tiny6410 下貌似调试不通
    void (*the_isr)(void);

    the_isr = VIC0ADDRESS;

    the_isr();

    EINT0PEND     = 0xf;
    VIC0ADDRESS = 0;
    } */
```

3. led.c 源程序：

```
#include "irq.h"
```

```
void delay()
{
 volatile int i = 0x500000;
 while (i--);
}


//LED 初始化函数，配置 GPK4~7 为输出功能
void led_init()
{
 GPKCON0=(GPKCON0&~(0xffffU<<16))
              |GPK4_OUT
              |GPK5_OUT
              |GPK6_OUT
              |GPK7_OUT;
}

//LED 点亮函数，通过 state 变量，控制 LED 点亮或熄灭
void led_display(int state)
{
 GPKDAT=(GPKDAT&~(0xf<<4))|((state&0xf)<<4);
}
```

4. mian.c 源程序

```
#include "irq.h"

int main()
{
 led_display(LED_ALL_OFF);//关闭所有 LED 灯后进入死循环
 while(1);
 return 0;
}
```

5. irq.h 头文件：

```
#ifndef __IRQ_H__
#define __IRQ_H__

#define    GPKCON0                    (*(volatile unsigned long *)0x7F008800)
#define    GPKDAT                     (*(volatile unsigned long *)0x7F008808)
#define GPNCON                 (*((volatile unsigned long *)0x7F008830))
#define GPNDAT                 (*((volatile unsigned long *)0x7F008834))
```

```
#define EINT0CON0                   (*((volatile unsigned long *)0x7F008900))
#define EINT0MASK                   (*((volatile unsigned long *)0x7F008920))
#define EINT0PEND                   (*((volatile unsigned long *)0x7F008924))

#define VIC0INTENABLE     (*((volatile unsigned long *)0x71200010))
#define VIC0IRQSTATUS    (*((volatile unsigned long *)0x71200000))
#define VIC0FIQSTATUS      (*((volatile unsigned long *)0x71200004))
#define VIC0RAWINTR       (*((volatile unsigned long *)0x71200008))
#define VIC0INTSELECT       (*((volatile unsigned long *)0x7120000c))
#define VIC0INTENCLEAR      (*((volatile unsigned long *)0x71200014))
#define VIC0PROTECTION      (*((volatile unsigned long *)0x71200020))
#define VIC0SWPRIORITYMASK    (*((volatile unsigned long *)0x71200024))
#define VIC0PRIORITYDAISY      (*((volatile unsigned long *)0x71200028))

#define VIC0ADDRESS               (*((volatile unsigned long *)0x71200f00))
#define VIC0VECTADDR0          (*((volatile unsigned long *)0x71200100))



#define    GPK4_OUT         (0x1U<<(4*4))
#define    GPK5_OUT         (0x1U<<(5*4))
#define    GPK6_OUT         (0x1U<<(6*4))
#define    GPK7_OUT         (0x1U<<(7*4))

#define    LED_ALL_OFF          0xF
#define    LED_ALL_ON           0x0
#define    LED1_ON              (~(0X1U<<0))
#define    LED2_ON              (~(0X1U<<1))
#define    LED3_ON              (~(0X1U<<2))
#define    LED4_ON              (~(0X1U<<3))

#endif
```

Makefile 文件：

```
irq.bin: start.o main.o led.o irq.o
	arm-linux-ld -Ttext 0x50000000 -o irq.elf $^
	arm-linux-objcopy -O binary irq.elf irq.bin
	arm-linux-objdump -D irq.elf > irq_elf.dis
%.o : %.S
	arm-linux-gcc -o $@ $< -c

%.o : %.c
	arm-linux-gcc -o $@ $< -c
```

```
clean:
	rm *.o *.elf *.bin *.dis    -rf
```

练习题：

修改程序，实现如下功能：
程序运行时，4 个 LED 灯全灭，然后间隔一段时间后全亮，循环往复。
按下 key1 时，LED 点亮顺序为：LED1、LED2、LED3、LED4
按下 key2 时，LED 点亮顺序为：LED2、LED3、LED4、LED1
按下 key3 时，LED 点亮顺序为：LED3、LED4、LED1、LED2
按下 key4 时，LED 点亮顺序为：LED4、LED1、LED2、LED3