嵌入式系统与开发

实验报告

|  |  |
| --- | --- |
| 实验名称 | 实验 11 按键中断控制  LED 跑马灯实验 |
| 实验次数 | 实验十一 |
| 学生姓名 | 戴高一 |
| 所在专业 | 计算机科学与技术 |
| 所在班级 | 201班 |
| 指导教师 | 李剑 |
| 地点 | 学10-509 |
| 时间 | 2023年5月 |

实验 11 按键中断控制 LED 跑马灯实验

**一、 实验目的**

1. 熟悉中断控制主程序需要进行的步骤

2. 熟悉中断服务程序需要进行的步骤

**二、 实验内容**

利用按键SW4和SW5实现按键中断控制LED跑马灯。

实验11的示例源代码：按下按键SW4，LED从左向右跑马灯闪烁10次；按下按键

SW5，LED从右向左跑马灯闪烁10次。

实验11的实验内容：做一个跑马灯的密码锁，当按照以下顺序“SW4，SW4，SW5”按

下按键时，LED才从左向右跑马灯闪烁10次。

编写系统的启动代码、按键中断控制程序、头文件以及Makefile文件，编译得到可执

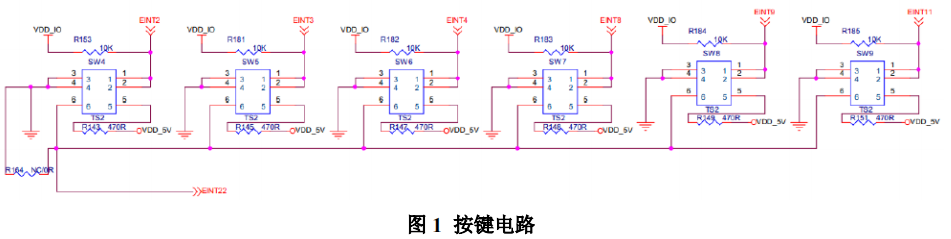
行文件，下载至开发板，实现在开发板上启动系统及按键控制功能。

1. **实验原理**

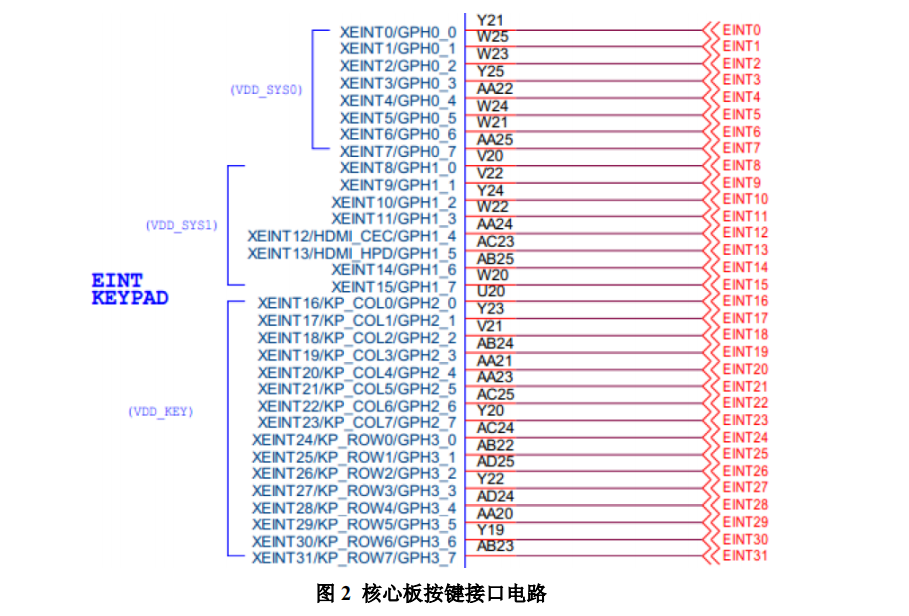
1．实验箱按键电路

按键使用GPIO接口，但按键本身需要外部的输入。按键硬件驱动原理图如图1所示。在下图的1×6矩阵按键（SW4~SW9）电路中，使用6个输入（EINT2、EINT3、EINT4、EINT8

、EINT9 和EINT11）。



按键入口对应的核心板接口电路如图2所示。



按键入口对应于核心板的GPH0接口，当其中一个SW按键被按下，通过查询方式就可以检测到是哪一个接口有输入信号，从而控制相应的操作。前述的按键工作原理都是在按键的理想状态下进行的，实际的按键动作会在短时间（几毫秒至几十毫秒）内产生信号抖动。

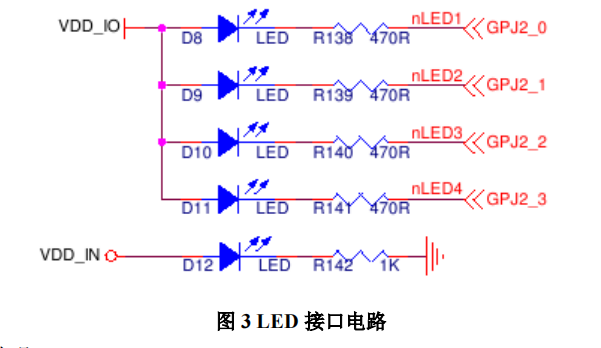
例如，当按键被按下时，其动作就像弹簧的若干次往复运动，将产生几个脉冲信号。一次按

键操作将会产生若干次按键中断，从而会产生抖动现象。因此驱动程序中必须要解决去除抖

动所产生的毛刺信号的问题。

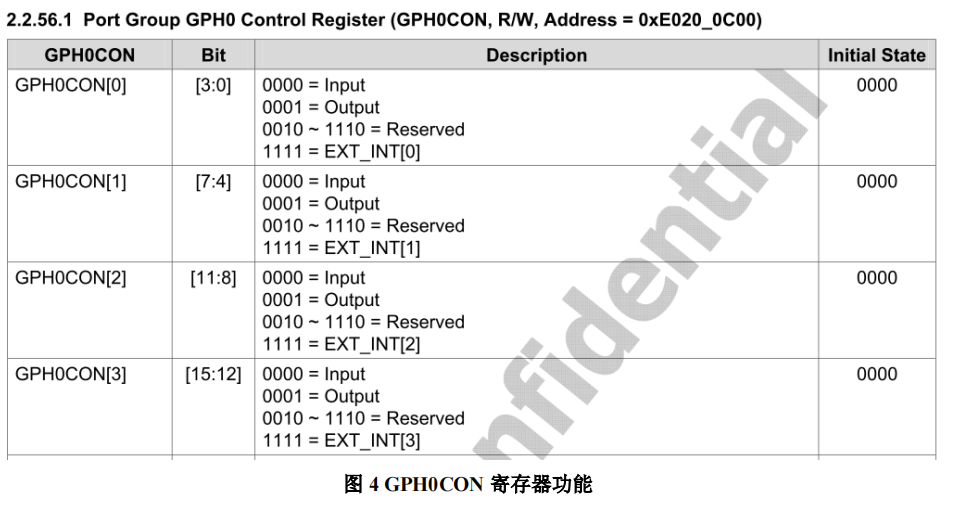
2．实验箱LED电路

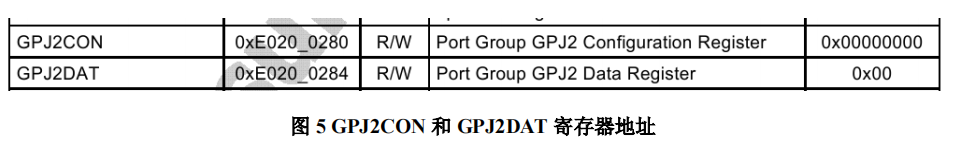
LED使用GPIO接口，对应于核心板的GPJ2接口，其接口电路如图3所示。当GPJ2接口输出低电平时，LED亮；输出高电平时，LED灭。

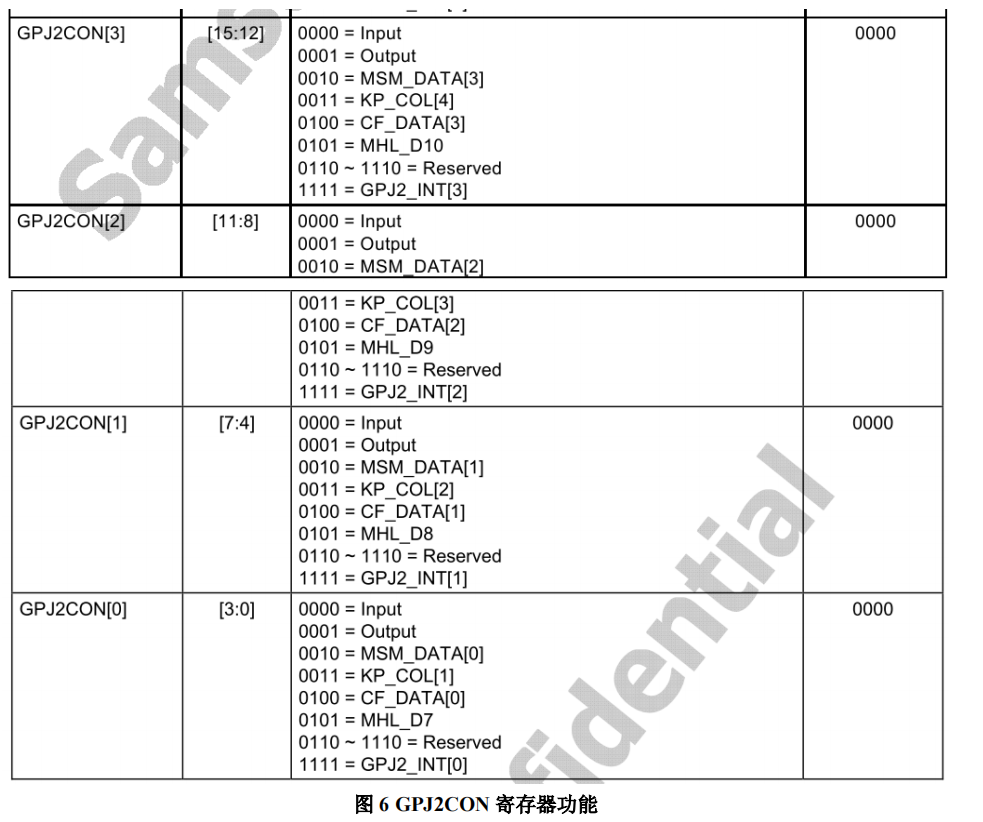


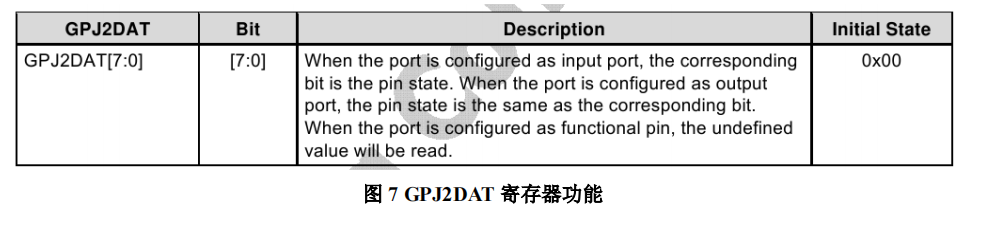
3．寄存器说明

按键对应的GPH0相关寄存器以及LED对应的GPJ2相关寄存器功能如下图所示。









**四、实验步骤**

1．编写键控制 LED 代码，将代码编译为二进制文件

1）在 ubuntu 系统中，进入共享文件夹 forlinux，新建 key 文件夹。

2）进入 key 文件夹，新建启动文件 start.S，并添加启动代码（参考附录 1）。

3）新建按键控制代码文件 key.c，自行编写代码。

4）新建 Makefile 文件，添加编译命令（参考附录 3），由以下命令编译生成二进制文件

key.bin。

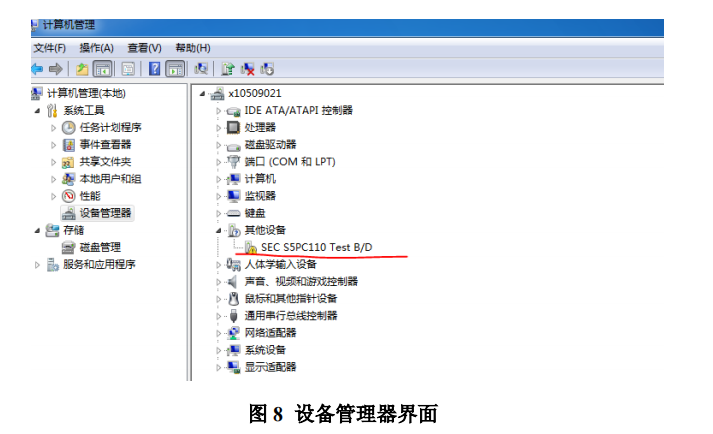
$ make clean

$ make

2．安装 USB 驱动

将实验箱中的拨码开关 2 拨到 on，长按 Power 键直至电脑提示安装驱动。打开计算机

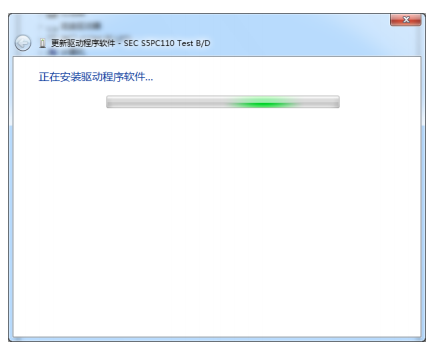
设备管理器，选择下图所示硬件安装驱动。



右键选择更新驱动程序，手动添加 USB 驱动程序路径“D:\新 509\04-常用工具\DNW”。

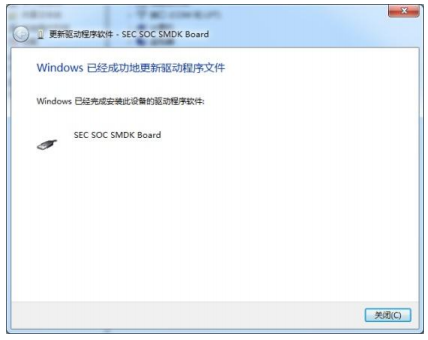
在图 4 所示驱动安装过程中，将拨码开关 2 重新置为 OFF 状态，然后关闭开发板的电源，

等待驱动安装完毕。



**图 9 驱动程序安装**

驱动安装完成后出现如图 5 所示提示。



**图 10 驱动安装成功**

3. 下载文件、启动系统

1）用 USB device 线连接电脑和开发板，设置开发板为 nandflash 启动(拨码开关全部拨

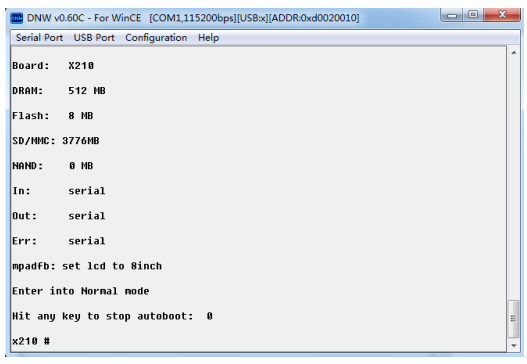
至OFF 状态)。

2）在目录“D:\新509\04-常用工具\DNW”中打开DNW.exe。设置串口：波特率为115200，

USB Port 为Download，Address 为0xd0020010。

3）在菜单栏开启DNW 串口连接（Serial Port ->Connect）。启动开发板后立即在 DNW

窗口迅速敲击空格键进入Uboot 状态，可见图11 启动界面



**图 11 开发板 Uboot 启动界面**

4）在 DNW 窗口中输入“dnw 0xd0020010”设置下载地址。如果DNW 驱动安装失败

或首次使用DNW，会提示安装驱动，请正确安装驱动，等到提示硬件可使用从进行下一步。

5）在DNW 菜单中，选择usbport->Transmit->Transmit 发送生成的 210.bin 文件，

DNW

自动下载210.bin 文件至开发板。

（用MV指令将linux下的210.bin，移动到/mnt/hgfs/forlinux，方便DNW传输，

/mnt/hgfs/forlinux 文件夹就是window下的forlinux文件夹）

6）在 DNW 窗口中输入“go 0xd0020010”，即可开始运行210.bin 程序。

注意，将二进制文件下载到SRAM 中不会破坏开发板中现有文件与程序，

但是掉电后所下载的文件将丢失。主要步骤中的命令及其结果截图。

**五、实验报告**

1. 附主要步骤中的命令及其结果截图。



1. 归纳总结。

本次实验主要帮助我们熟悉中断控制主程序、中断控制服务程序需要进行的步骤。主要学习配置引脚、中断寄存器等设置。流水灯操作与之前的实验类似，密码锁则通过在中断服务程序里判断第几位、是否吻合来实现。实验总体较为简单，主要帮助我们熟悉中断相关操作。

1. 实验代码

**addheader.c**

/\*

\*\* 在BL0阶段，iROM内固化的代码读取nandflash或SD卡前面最大16K的内容（即BL1）到iRAM，

\*\* 并比对前16字节中的校验和是否正确，正确则继续，错误则尝试启动下一个设备。

\*\* BL1的头信息规定如下

\*\* 0x0：BL1的大小（最大16K，包括BL1头信息的大小）

\*\* 0x4: 0（规定）

\*\* 0x8：校验和

\*\* 0xC：0（规定）

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define IMG\_SIZE (16\*1024)

#define HEADER\_SIZE 16

#define BLKSIZE 512

int main (int argc, char \*argv[])

{

FILE \*fp;

unsigned char \*Buf;

int BufLen;

int nbytes, fileLen;

unsigned int checksum, count;

int i;

if (argc != 3)

{

printf("Usage: %s <source file> <destination file>\n", argv[0]);

return -1;

}

/\* 分配16K的buffer \*/

BufLen = IMG\_SIZE;

Buf = malloc(BufLen);

if (!Buf)

{

perror("Alloc buffer failed!");

return -1;

}

memset(Buf, 0x00, BufLen);

/\* 读源bin到buffer \*/

fp = fopen(argv[1], "rb");

if( fp == NULL)

{

perror("source file open error");

free(Buf);

return -1;

}

/\* 获取源bin长度 \*/

fseek(fp, 0L, SEEK\_END);

fileLen = ftell(fp);

fseek(fp, 0L, SEEK\_SET);

/\* 源bin长度不得超过16K-16byte \*/

fileLen = (fileLen < (IMG\_SIZE - HEADER\_SIZE)) ? fileLen : (IMG\_SIZE - HEADER\_SIZE);

/\* 读源bin到buffer[16] \*/

nbytes = fread(Buf + HEADER\_SIZE, 1, fileLen, fp);

if (nbytes != fileLen)

{

perror("source file read error\n");

free(Buf);

fclose(fp);

return -1;

}

fclose(fp);

/\* 计算校验和 \*/

for(i = 0, checksum = 0; i < fileLen; i++)

checksum += Buf[HEADER\_SIZE + i];

/\* 计算BL1的大小:

\*\* BL1的大小包括BL1的头信息

\*\* 另外iROM从SD卡拷贝是按块拷贝的，因此这里需要调整大小为512字节的整数倍

\*/

fileLen += HEADER\_SIZE;

count = fileLen / BLKSIZE \* BLKSIZE;

if (count < fileLen)

count += BLKSIZE;

memcpy(Buf, &count, 4); // 保存BL1的大小到Buf[0-3]

// 将校验和保存在buffer[8~15]

memcpy(Buf + 8, &checksum, 4);

fp = fopen(argv[2], "wb");

if (fp == NULL)

{

perror("destination file open error");

free(Buf);

return -1;

}

// 将count + HEADER\_SIZE字节的buffer拷贝到目的bin中

nbytes = fwrite(Buf, 1, count, fp);

if (nbytes != count)

{

perror("destination file write error");

free(Buf);

fclose(fp);

return -1;

}

free(Buf);

fclose(fp);

return 0;

}

**key.c**

#define GPJ2CON (\*(volatile unsigned long \*) 0xE0200280)

#define GPJ2DAT (\*(volatile unsigned long \*) 0xE0200284)

#define GPH0CON (\*(volatile unsigned long \*) 0xE0200C00)

#define GPH0DAT (\*(volatile unsigned long \*) 0xE0200C04)

//外中断0-7的中断控制寄存器

#define EXT\_INT\_0\_CON \*((volatile unsigned int \*)0xE0200E00)

//外中断0-7的中断屏蔽寄存器

#define EXT\_INT\_0\_MASK \*((volatile unsigned int \*)0xE0200F00)

//外部中断0-7的中断挂起寄存器，记录是否有中断产生

#define EXT\_INT\_0\_PEND \*((volatile unsigned int \*)0xE0200F40)

//第一组矢量中断选择寄存器

#define VIC0INTSELECT \*((volatile unsigned int \*)0xF200000C)

//第一组矢量中断使能寄存器

#define VIC0IRQSTATUS \*((volatile unsigned int \*)0xF2000000)

#define VIC0INTENABLE \*((volatile unsigned int \*)0xF2000010)

#define VIC0VECTADDR2 \*((volatile unsigned int \*)0xF2000108)

#define VIC0VECTADDR3 \*((volatile unsigned int \*)0xF200010C)

#define VIC0ADDRESS \*((volatile unsigned int \*)0xF2000F00)

extern void key\_isr(void);

int count=0;

void delay(int t){

while(t--);

}

void led\_init(void)

{

/\* 配置GPC0\_1~GPC0\_4为输出：LED1~LED4 \*/

GPJ2CON &= ~(0xFF << 0);

GPJ2CON |= ((0x01 << 0) | (0x01 << 4) | (0x01 << 8) | (0x01 << 12));

GPJ2DAT |= (0xFF << 0);

}

void key\_init(void)

{

/\* 配置GPH0\_3为外部中断：SW5 \*/

GPH0CON |= (0xF << 4\*3) | (0xF << 4\*2);

/\* 清空相应位\*/

EXT\_INT\_0\_CON &= ~((0xF << 4\*3) | (0xF << 4\*2));

/\* 配置EXT\_INT[0]和EXT\_INT[1]为下降沿触发 0b0 010 0 010 \*/

EXT\_INT\_0\_CON |= (2 << 4\*2)|(2 << 4\*3);

/\* 取消屏蔽外部中断EXT\_INT[3] \*/

EXT\_INT\_0\_MASK &= ~0xC;

}

void int\_init(void)

{

/\* 选择外部中断EXT\_INT[3]为IRQ类型的中断 \*/

VIC0INTSELECT &= ~0xC;

/\* 使能外部中断EXT\_INT[3] \*/

VIC0INTENABLE |= 0xC;

/\* 清VIC0ADDRESS \*/

VIC0ADDRESS = 0X0;

/\* 当EXT\_INT[0]触发中断，即用户按下key1时，CPU就会自动的将VIC0VECTADDR0的值赋给VIC0ADDRESS并跳转到这个地址去执行\*/

VIC0VECTADDR2 = (int)key\_isr;

VIC0VECTADDR3 = (int)key\_isr;

}

void key\_handle(void)

{

/\* 获取键值 \*/

/\*volatile unsigned char key\_code = EXT\_INT\_0\_PEND & 0x8; \*/

volatile unsigned char key\_code = VIC0IRQSTATUS & 0xC;

/\* 清中断向量寄存器 \*/

VIC0ADDRESS = 0;

/\* 清中断挂起寄存器 \*/

EXT\_INT\_0\_PEND |= 0xC;

if (count == 0 && key\_code == 0x4){

GPJ2DAT |= (0xFF << 0);

GPJ2DAT ^= 0x1<<count;

count+=1;

}

else if(count == 1 && key\_code == 0x4){

GPJ2DAT |= (0xFF << 0);

GPJ2DAT ^= 0x1<<count;

count+=1;

}

else if(count == 2 && key\_code == 0x8){

GPJ2DAT |= (0xFF << 0);

int i=0;

for(i=0;i<10;i++){

GPJ2DAT ^= 0x1<<0;

delay(100000);

GPJ2DAT ^= 0x1<<0;

GPJ2DAT ^= 0x1<<1;

delay(100000);

GPJ2DAT ^= 0x1<<1;

GPJ2DAT ^= 0x1<<2;

delay(100000);

GPJ2DAT ^= 0x1<<2;

GPJ2DAT ^= 0x1<<3;

delay(100000);

GPJ2DAT ^= 0x1<<3;

}

GPJ2DAT |= (0xFF << 0);

count=0;

}

else {

GPJ2DAT |= (0xFF << 0);

count=0;

}

}

int main()

{

key\_init();

led\_init();

int\_init();

while (1);

return 0;

}

**start.S**

.global \_start

.global key\_isr

\_start:

@设置栈，以调用c函数

ldr sp, =0x40000000

@ 开总中断

mrs r0, cpsr

@ 读取cpsr寄存器中的值到r0

bic r0, r0, #0x00000080

@ 清除第7位，IRQ中断禁止位，写0使能IRQ

msr cpsr, r0

@把修改好的r0的值重新赋会cpsr

bl main

@ 跳转到C函数去执行

halt:

b halt

key\_isr:

@; 计算返回地址:PC的值等于当前执行的地址+8，当CPU正要执行某条指令时（还未执行），被中断， 这时这条刚要执行的指令的地址刚好=PC-4

sub lr, lr, #4

stmfd sp!, {r0-r12, lr}

@; 保护现场

bl key\_handle

@; 恢复现场

ldmfd sp!, {r0-r12, pc}^

@; ^表示把spsr恢复到cpsr

**Makefile**

key.bin: start.o key.o

arm-linux-ld -Ttext 0xd0020010 -o key.elf $^

arm-linux-objcopy -O binary key.elf key.bin

arm-linux-objdump -D key.elf > key.dis

key.o: key.c

arm-linux-gcc -nostdlib -c $< -o $@

start.o:start.S

arm-linux-gcc -nostdlib -c $< -o $@

clean:

rm \*.o \*.elf \*.bin \*.dis