基于 OMAPL138 的 Linux 设备驱动程序开发入门

1 led 驱动程序

1.1 led 设备驱动程序编写和解析

光盘中有驱动程序安装镜像和源码，路径为：

led.ko: demo\driver\led\led.ko //驱动安装镜像

led.c: demo\driver\led\led.c//驱动程序源码

以下为此驱动程序的解释：

```
#include<linux/init.h>
#include<linux/module.h>
#include<linux/kernel.h>
#include<linux/fs.h>
#include<linux/types.h>
#include<linux/ioctl.h>
#include<linux/cdev.h>
#include<linux/kdev_t.h>
#include<mach/hardware.h>
#include<linux/gpio.h>

#include <asm/mach-types.h>
#include <asm/mach/arch.h>
#include <mach/da8xx.h>
/*PINMUX1 寄存器的地址*/
#define SYSCFG_BASE 0x01c14000
#define PINMUX1_OFFSET 0x124
/*GPIO0 配置和数据寄存器地址*/
```

```
#define    GPIO0_DIRECT      0x01e26010
#define    GPIO0_DATA        0x01e26014
/*设置主从设备号*/
static int LED_MAJOR=239;
static int LED_MINOR=0;
static int count =1;

static int temp;
/*led 设备名字*/
#define LED_NAME    "led-device"

#define led_on 0
#define led_off 1

static struct cdev *led_cdev;
static struct class *led_class;
static dev_t led_dev;
/*文件打开函数*/
static int led_open(struct inode *inode,struct file *file)
{
    return 0;
}
/*文件释放函数*/
static int led_release(struct inode *inode,struct file *file)
{
```

```
        printk("release success!\n");
        return 0;
    }
    /*io control 函数*/
    static int led_ioctl(struct file *file,unsigned int cmd,unsigned long arg)
    {
    /*判断用户所输入的灯号是否正确：6 选中 D6，7 选中 D7，9 选中 D9，10 选中 D10，
all 选中 D6、D7、D9、D10。*/
        if(arg<6||arg>10)
        {
            return -1;
        }
    /*判断用户所输入的命令 cmd 是否正确，cmd 只能为 0 或 1*/
        if(cmd>led_off)
        {
            return   -1;
        }
        switch(cmd)
        {
          case led_on:
          {
            if(arg==6)
            {
                        temp=(temp&(~0x0020))|(1<<5); //GPIO0[5]置 1，灯 D6 亮
                __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
```

```
        }
        else    if(arg==7)
        {
            temp=(temp&(~0x0001))|(1<<0); //GPIO0[0]置 1，灯 D7 亮
            __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
        }
        else if(arg==9)
        {
        temp=(temp&(~0x0002))|(1<<1); //GPIO0[1]置 1，灯 D9 亮
            __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
        }
        else if(arg==10)//GPIO0[2]置 1，灯 D10 亮
        {
            temp=(temp&(~0x0004))|(1<<2);
            __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
        }
//GPIO0[0]、GPIO0[1] 、GPIO0[2]、GPIO0[5]同时置 1，灯 D6、D7、D9、D10 同时亮
        else
        {
            temp=(temp&(~0x0027))|(1<<0)|(1<<1)|(1<<2)|(1<<5);
            __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
        }
        return 0;
    }
```

```
case led_off:
{
    if(arg==6)
    {
        temp=(temp&(~0x0020)); //GPIO0[5]清 0，灯 D6 灭
        __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
    }
    else    if(arg==7)
    {
        temp=(temp&(~0x0001)); //GPIO0[0]清 0，灯 D7 灭
        __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
    }
    else if(arg==9)
    {
        temp=(temp&(~0x0002)); //GPIO0[1]清 0，灯 D9 灭
        __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
    }
    else if(arg==10)
    {
    temp=(temp&(~0x0004)); //GPIO0[2]清 0，灯 D10 灭
        __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
    }
//GPIO0[0]、GPIO0[1] 、GPIO0[2]、GPIO0[5]同时清 0，灯 D6、D7、D9、D10 同时灭
    else
    {
```

```
                temp=(temp&(~0x0027));
                __raw_writel(temp,IO_ADDRESS(GPIO0_DATA));
            }
            return 0;
        }
        default:
            return -1;
    }
}
```

/*led 驱动文件操作结构体，file_operations 结构体中的成员函数会在应用程序进行 Linux 的 open()、release()、ioctl()协同调用时最终被调用。*/

```
static struct    file_operations led_fops=
{
    .owner=THIS_MODULE,
    .open=led_open,
    .release=led_release,
    .unlocked_ioctl=led_ioctl,
};
```

/*当通过 insmod 命令加载内核模块时，模块的加载函数会自动被内核执行，完成一系列的初始化工作，初始化的内容包括：申请设备号、注册字符设备、创建节点、设置 OMAPL138 的 GPIO 工作方式。*/

/*LED 驱动模块加载函数*/

```
static int __init LED_init(void)
{
    int ret;
```

```
unsigned    int     PINMUX1_REG_old;

   if(LED_MAJOR) /*静态申请设备号*/
{
    led_dev=MKDEV(LED_MAJOR,LED_MINOR);
    ret=register_chrdev_region(led_dev,count,LED_NAME);
}
else/*动态申请设备号*/
{
    ret=alloc_chrdev_region(&led_dev,LED_MINOR,count,LED_NAME);
    LED_MAJOR=MAJOR(led_dev);
}

if(ret<0) /*判断设备号申请是否成功*/
{
    printk(KERN_ERR "register chrdev fail!");
    return -1;
}

led_cdev=cdev_alloc();/*分配 cdev*/

if(led_cdev!=NULL)
{
      /*初始化 cdev*/
    cdev_init(led_cdev,&led_fops);
```

```
        led_cdev->ops=&led_fops;
        led_cdev->owner=THIS_MODULE;
         /*注册 cdev*/
        if(cdev_add(led_cdev,led_dev,count))
              printk(KERN_ERR "register cdev fail!");
        else
              printk(KERN_ERR "register success!\n");
    }
    else
    {
        printk(KERN_ERR "register cdev err!");
        return -1;
    }
    /*创建 LED_NAME 这个类*/
  led_class=class_create(THIS_MODULE,LED_NAME);
    if(IS_ERR(led_class))
    {
        printk(KERN_ERR "register class err!");
        return -1;
    }
     /*创建 dev/LED_NAME 这个设备节点*/
    device_create(led_class,NULL,led_dev,NULL,LED_NAME);

    printk(LED_NAME " initialized\n");
     //配置 PINMUX1 寄存器，配置 GPIO0[0]、GPIO0[1]、GPIO0[2]、GPIO0[5]为 I/O 引脚
```

```
        PINMUX1_REG_old=__raw_readl(IO_ADDRESS(SYSCFG_BASE)+PINMUX1_OFFSET);
        PINMUX1_REG_old=(PINMUX1_REG_old&0x000ff0ff)|0x88800800;

        __raw_writel(PINMUX1_REG_old, IO_ADDRESS(SYSCFG_BASE)+PINMUX1_OFFSET);

    __raw_writel(0,IO_ADDRESS(GPIO0_DIRECT)); //配置 GPIO0 为输出

        __raw_writel(0x00000000,IO_ADDRESS(GPIO0_DATA));//初始化时将 GPIO0 的引脚拉低

        return 0;
    }
/*当通过 rmmod 命令卸载内核模块时，模块的卸载函数会自动被内核执行，完成一系列的卸载工作，包括：注销字符设备、注销设备号、取消设备节点、取消类。*/

    /*LED 驱动模块卸载函数*/
    static void __exit LED_exit(void)
    {
        printk("led chrdev exit!\n");
        cdev_del(led_cdev);//注销设备
        unregister_chrdev_region(led_dev,count);//注销设备号
        device_destroy(led_class,led_dev);//取消设备节点
        class_destroy(led_class);//取消类

    }
    module_init(LED_init);
```

```
module_exit(LED_exit);


MODULE_DESCRIPTION("led character");
MODULE_AUTHOR("Apple");
MODULE_LICENSE("GPL");
```

以上是 led 驱动的驱动程序，对于 GPIO 口的操作，有以下几点步骤：

1.查看开发板的原理图，查看与 led 连接的 GPIO 口号，广州创龙电子的 OMAPL138 开发板与 led 连接的 GPIO 分别是 GPIO0[5]、GPIO0[0]、GPIO0[1]、GPIO0[2]；

2.查看 OMAPL138 的数据手册，查找 PINMUX1 寄存器的地址，将对应的管脚的寄存器中相应位设置为 GPIO 的工作模式；

3.设置 GPIO 的方向寄存器，在本例程中将 GPIO 口配置为输出；

4.配置 GPIO 的数据寄存器，写“1”表示输出高电平，写“0”表示输出低电平。

1.2 使用 Makefile 文件编译驱动程序

1.2.1 编写驱动程序 Makefile 文件

一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，Makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 Makefile 就像一个 Shell 脚本一样，其中也可以执行操作系统的命令。

光盘中有 led 驱动 Makefile 文件，路径为：

Makefile: demo\driver\led\Makefile //测试程序镜像

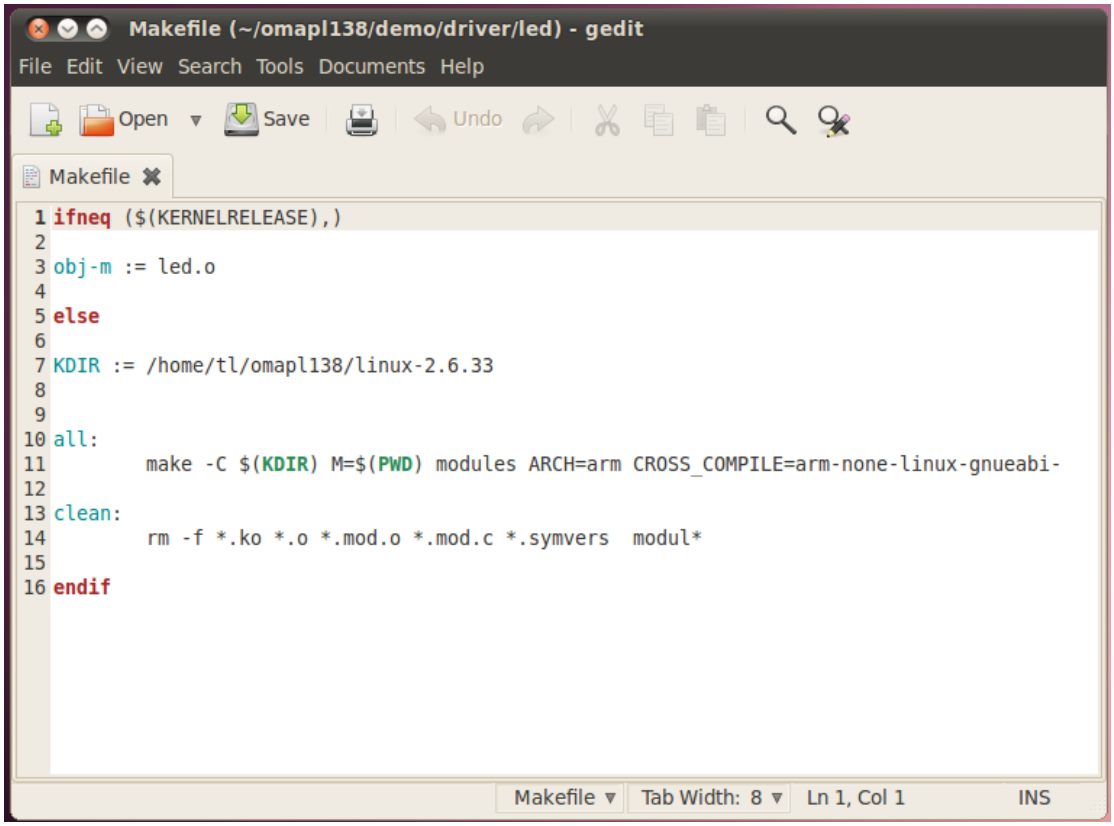
以下为 led 驱动的 Makefile 文件的注释：

```
ifneq ($(KERNELRELEASE),)
obj-m := led.o//定义了要编译的驱动文件为 led.c，生成的模块名字为 led.ko
else
KDIR := /home/tl/omapl138/linux-2.6.33//使用的内核源码路径，注意，内核源码一定要先编译
```

一次，否则编译驱动程序会出错。

//以下定义运行“make”编译命令时使用的内核源码、驱动源码路径、平台、使用的交叉编译工具链。

```
all:
	make -C $(KDIR) M=$(PWD) modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
//定义运行“make clean”时清除的文件。
clean:
	rm -f *.ko *.o *.mod.o *.mod.c *.symvers  modul*
endif
```



1.2.2 编译驱动程序

将光盘 demo\driver\led 的 led.c 和 Makefile 文件复制到开发系统 ubuntu 以下目录（若目录不存在请先建立）：/home/tl/omapl138/demo/driver/led，并在驱动程序目录的运行以下

命令编译驱动程序：

```
Host#    make
```

运行以上命令后，系统会根据 Makefile 文件的规则去编译整个驱动源码，产生了 led.ko 和其他中间文件，如下图所示：

```
tl@tl-desktop:~/omapl138/demo/driver/led$ ls
led.c  Makefile
tl@tl-desktop:~/omapl138/demo/driver/led$ make
make -C /home/tl/omapl138/linux-2.6.33 M=/home/tl/omapl138/demo/driver/led modu
les ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
make[1]: Entering directory `/home/tl/omapl138/linux-2.6.33'
  CC [M]  /home/tl/omapl138/demo/driver/led/led.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/tl/omapl138/demo/driver/led/led.mod.o
  LD [M]  /home/tl/omapl138/demo/driver/led/led.ko
make[1]: Leaving directory `/home/tl/omapl138/linux-2.6.33'
tl@tl-desktop:~/omapl138/demo/driver/led$ ls
led.c   led.mod.c  led.o     modules.order
led.ko  led.mod.o  Makefile  Module.symvers
```

得到的 led.ko 文件可以放到开发板上，具体作用下面章节会详细介绍。

1.3 将 led 设备驱动模块静态编译进内核或者内核模块

1.3.1 修改内核 Kconfig

假如用户需要将自己的驱动程序编译到内核，请将光盘 demo\driver\led 目录下 led 灯的驱动程序源代码 led.c 放到内核源码 drivers/char 目录下，修改内核源码 drivers/char 目录下的 Kconfig 菜单配置文件，在 menu "Character devices"行下面添加如下内容：

```
 5 menu "Character devices"
 6 config USER_LED
 7         tristate "user led"
 8         depends on ARM
 9         default y
10         ---help---
11           There are two leds on the OMAPL138 base board which name D6 and D7.
```

config USER_LED：USER_LED 是驱动程序的配置名称，可以任意修改，但字母务必大写。

tristate "user led"：在编译内核 make menuconfig 时菜单栏出现的驱动名字。

depends on ARM：注明是 ARM 平台下的驱动程序。

default y：默认是静态编译到内核镜像的。

---help---：驱动程序的补充信息，让用户进一步了解此驱动程序的作用。

1.3.2 修改内核 Makefile

修改内核源码 drivers/char 目录下的 Makefile 编译文件，在第 12 行添加如下内容：

obj-$(CONFIG_USER_LED) += led.o

```
 8 FONTMAPFILE = cp437.uni
 9
10 obj-y    += mem.o random.o tty_io.o n_tty.o tty_ioctl.o tty_ldisc.o tty_buffer.o tty_port.o
11
12 obj-$(CONFIG_USER_LED)          += led.o
13 obj-$(CONFIG_LEGACY_PTYS)       += pty.o
```

obj-$(CONFIG_USER_LED)：USER_LED 这个字段必须和前面提到的 Kconfig 的 config 后面的字段一致。

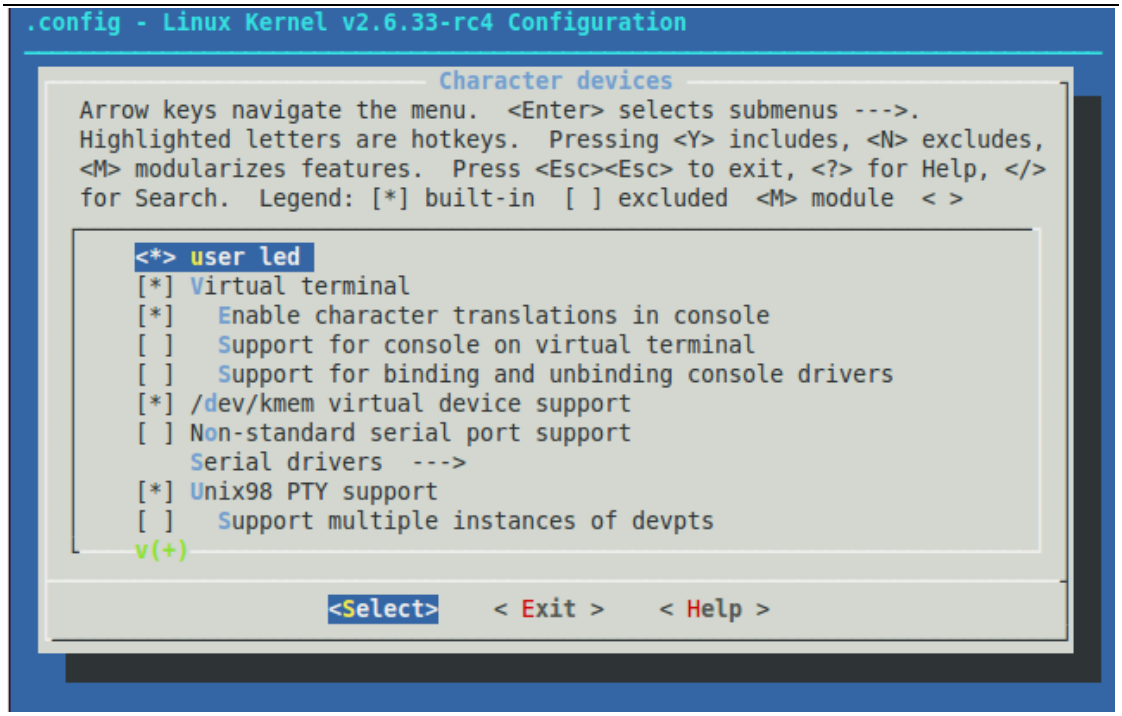
+= led.o：这个前缀必须是“led”，编译驱动程序时，系统会去找 driver/char 目录下的 led.c 文件。

1.3.3 设备驱动模块静态编译到内核

在内核源码顶层目录执行以下命令查看设备内核配置情况：

Host# make menuconfig ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

在“device drivers->character devices”下有“user led”的驱动配置选项，如下图：



前面的“*”符号代表将设备驱动程序模块静态编译到内核里面。保存退出，接着运行以下命令编译更新设备（led）驱动程序后的内核源码:

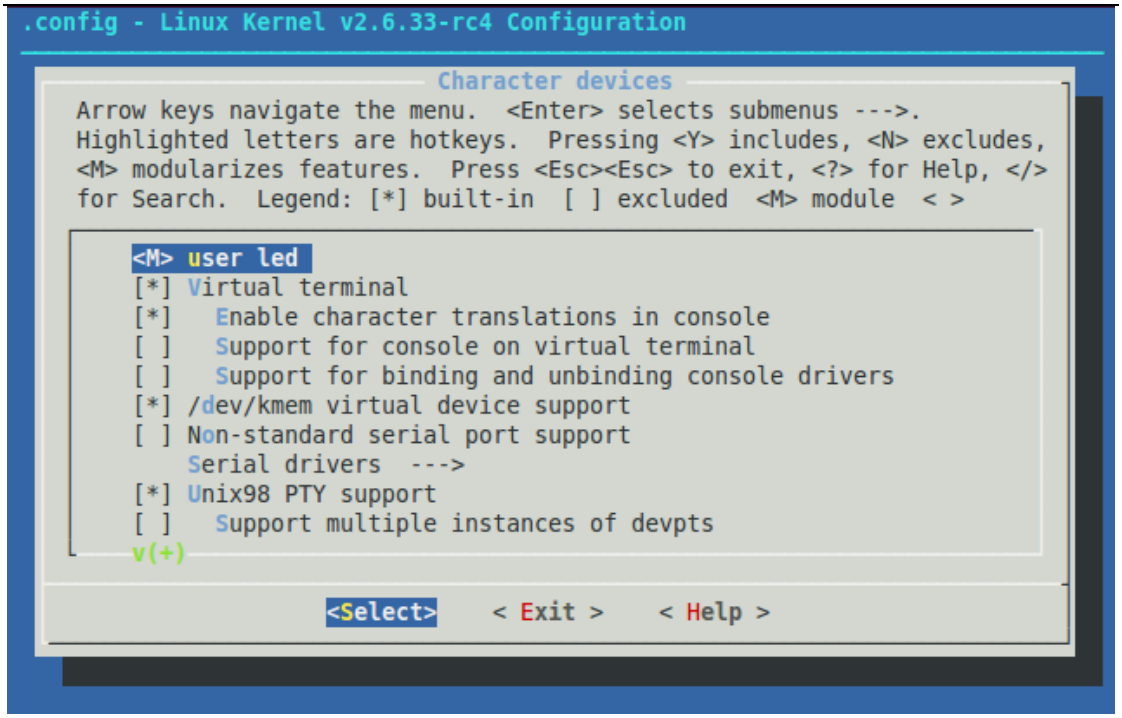
```
Host#    uImage ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

```
tl@tl-desktop:~/omapl138/linux-2.6.33$ make uImage ARCH=arm CROSS_COMPILE=arm-no
ne-linux-gnueabi-
scripts/kconfig/conf -s arch/arm/Kconfig
drivers/usb/musb/Kconfig:147:warning: defaults for choice values not supported
  CHK     include/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
  CALL    scripts/checksyscalls.sh
```

然后使用内核 arch/arm/boot 目录下的 uImage 更新开发板的内核。

1.3.4 设备驱动程序编译成内核模块

若需要将驱动程序编译成内核模块的形式，按空格键是其变为<M>（变为空代表不编译），如下图所示：



保存退出，接着运行以下命令产生 led.ko 文件，其路径是内核源码 drivers/char/led.ko：

Host# make SUBDIR=drivers/char modules ARCH=arm

CROSS_COMPILE=arm-none-linux-gnueabi-

```
tl@tl-desktop:~/omapl138/linux-2.6.33$ make SUBDIR=drivers/char modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
  CHK     include/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
  CALL    scripts/checksyscalls.sh
  CC [M]  fs/autofs4/init.o
  CC [M]  fs/autofs4/inode.o
  CC [M]  fs/autofs4/root.o
```

1.4 led 设备驱动测试程序编写和解析

光盘中有测试程序镜像和源码，路径为：

led_test: demo\app\led\led_test //测试程序镜像

led_test.c: demo\app\led\led_test.c//测试程序源码 s

以下为此测试程序的解释：

```
#include<stdio.h>

#include<stdlib.h>
```

```
#include<sys/ioctl.h>
#include<unistd.h>

int main(int argc,char **argv)//argc 表示输入参数的个数，*argv 表示输入的参数
{
    int fd; //定义 led 设备文件描述的变量
    int led_nr;//定义 led 编号
    int led_on;//定义 LED 状态变量，0 表示亮，1 表示灭
    char str[10];
    char str1[]={"all"};
    char str2[]={"run"};
    sscanf(argv[1],"%s",str);
    if(strcmp(str,str1)==0)//如果第二个输入参数为“all”指令，将 led_nr 赋值为 8
        led_nr=8;
    else if(strcmp(str,str2)==0)//如果第二个输入参数为“run”指令，将 led_nr 赋值
为 11
        led_nr=11;
    //将输入 led 编号赋给 led_nr，并检查参数是否符合要求
    else if(sscanf(argv[1],"%d",&led_nr)!=1||led_nr<6||led_nr>10||led_nr==8)
    {
        fprintf(stderr,"wrong input format!\n");
        exit(1);
    }
    /*判断所输入的参数是否符合要求*/
    if(argc!=3||sscanf(argv[2],"%d",&led_on)!=1||led_on<0||led_on>1)
```

```
{
    fprintf(stderr,"wrong input format!\n");
    exit(1);
}
if(led_nr==8||led_nr==11)
    printf("led number:all\n");
else
    printf("led number:%d\n",led_nr);
//打开设备驱动节点文件，在开发板运行“insmod led.ko”命令后，会生成/dev/led-device 文件
fd=open("/dev/led-device",0);
if(fd<0)
{
    fprintf(stderr,"User led: open failed!\n");
    exit(1);
}
else
    printf("open success!\n");
if(led_nr==11)
    printf("led state:run\n");
else if(!led_on)
    printf("led state:on\n");
else
    printf("led state:off\n");
if(led_nr==11)//如果第二个输入参数是“run”指令，则执行流水灯程序
```

```
        {
            while(1)
            {
                ioctl(fd,1,8);
                ioctl(fd,0,7);
                sleep(1);
                ioctl(fd,1,8);
                ioctl(fd,0,6);
                sleep(1);
                ioctl(fd,1,8);
                ioctl(fd,0,9);
                sleep(1);
                ioctl(fd,1,8);
                ioctl(fd,0,10);
                sleep(1);
            }
        }
        else
            ioctl(fd,led_on,led_nr);
        sleep(1);
        close(fd);
        return 0;
    }
```

1.5 led 驱动测试程序编译

将光盘 demo\app\led 目录下 led 灯的设备驱动测试程序源代码 led_test.c 拷贝到 Ubuntu 虚拟机的/home/tl/omapl138/demo/app/led 目录下（若目录不存在请先建立），进入此目录，确保已经安装交叉编译工具链后，执行命令 “arm-none-linux-gnueabi-gcc led_test.c -o led_test”，可以发现在当前目录生成了 ARM 端可执行文件 led_test，如下图：

```
tl@tl-desktop:~/omapl138/demo/app/led$ pwd
/home/tl/omapl138/demo/app/led
tl@tl-desktop:~/omapl138/demo/app/led$ ls
led_test.c
tl@tl-desktop:~/omapl138/demo/app/led$ arm-none-linux-gnueabi-gcc led_test.c -o led_test
tl@tl-desktop:~/omapl138/demo/app/led$ ls
led_test  led_test.c
tl@tl-desktop:~/omapl138/demo/app/led$ file led_test
led_test: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs)
, for GNU/Linux 2.6.14, not stripped
tl@tl-desktop:~/omapl138/demo/app/led$
```

从命令中可以看出需要编译的源文件是 led_test.c，“-o led_test”指明了输出的可执行文件是 led_test。

1.6 测试 led 设备驱动程序

对于将驱动程序编译成模块的用户，需要先将编译得到或者光盘中的 led.ko 拷贝到开发板的/home/tl/omapl138/demo/driver/led 目录下（若没有此目录请先建立）。

同时将编译得到或者光盘中 demo\app\led 目录下的 led_test 文件也拷贝到开发板的/home/tl/omapl138/demo/driver/led 目录下（若没有此目录请先建立）。

然后进入开发板的/home/tl/omapl138/demo/driver/led 目录，执行以下命令安装 led 驱动程序：

Target# insmod led.ko //编译进内核的省略此安装步骤

```
root@arago:~# cd /home/tl/omapl138/demo/driver/led/
root@arago:/home/tl/omapl138/demo/driver/led# ls
led.ko
root@arago:/home/tl/omapl138/demo/driver/led# pwd
/home/tl/omapl138/demo/driver/led
root@arago:/home/tl/omapl138/demo/driver/led# insmod led.ko
register success!
led-device initialized
root@arago:/home/tl/omapl138/demo/driver/led# _
```

然后进入开发板的/home/tl/omapl138/demo/app/led 目录，执行以下命令运行 led 驱动测试程序：

Target# ./led_test 6 0//控制 D6 亮

Target# ./led_test 6 1//控制 D6 灭

Target# ./led_test 7 0//控制 D7 亮

Target# ./led_test 7 1//控制 D7 灭

Target# ./led_test 9 0//控制 D9 亮

Target# ./led_test 9 1//控制 D9 灭

Target# ./led_test 10 0//控制 D10 亮

Target# ./led_test 10 1//控制 D10 灭

Target# ./led_test run 0//让 led 运行跑马灯，按 Ctrl+C 停止

Target# ./led_test all 0//控制所有 led 灯亮

Target# ./led_test all 1//控制所有 led 灯灭

运行结果如下图：

```
root@arago:/home/tl/omapl138/demo/app/led# pwd
/home/tl/omapl138/demo/app/led
root@arago:/home/tl/omapl138/demo/app/led# ls
led_test
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 6 0
led number:6
open success!
led state:on
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 6 1
led number:6
open success!
led state:off
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 7 0
led number:7
open success!
led state:on
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 7 1
led number:7
open success!
release success!
```

```
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 9 0
led number:9
open success!
led state:on
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 9 1
led number:9
open success!
led state:off
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 10 0
led number:10
open success!
led state:on
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test 10 1
led number:10
open success!
led state:off
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test run 0
led number:all
open success!
led state:run
release success!
```

```
root@arago:/home/tl/omapl138/demo/app/led# ./led_test all 0
led number:all
open success!
led state:on
release success!
root@arago:/home/tl/omapl138/demo/app/led# ./led_test all 1
led number:all
open success!
led state:off
release success!
```

卸载驱动程序的命令如下：

Target# rmmod led

```
root@arago:/home/tl/omapl138/demo/app/led# rmmod led
led chrdev exit!
root@arago:/home/tl/omapl138/demo/app/led#
```

led 是 led.ko 的前缀，其他名字的驱动程序一样，卸载只需要输入驱动程序的前缀即可。

2 按键设备驱动程序

2.1 按键设备驱动程序编写和解析

光盘中有驱动程序安装镜像和源码，路径为：

button.ko: demo\driver\button\button.ko //驱动安装镜像

button.c: demo\driver\button\button.c //驱动程序源码

/*头文件*/

```
#include<linux/init.h>          //init .exit   cinfig
#include<linux/module.h>        //module     config
#include<linux/kernel.h>        //kernel
#include<linux/fs.h>            //file
#include<linux/types.h>         //type config   like #define   __u64 uint64_t
#include<linux/cdev.h>          //cdev_register   and so on   cdev_t
#include<linux/kdev_t.h>        //mkdev (major ,minor )
#include<linux/device.h>        //dev_t
#include<linux/err.h>           //err

#include<linux/interrupt.h>     //request_irq   free_irq
#include<linux/irq.h>           //irq_set_irq_type IRQ_TYPE_RASING
#include <linux/gpio.h>         //gpio_to_irq   gpio_request   gpio_direction_input/gpiolib

#include<asm/uaccess.h>         //copy to user
#include<linux/ioctl.h>         //ioctl
#include<linux/delay.h>         //for delay
```

```
static int BUTTONS_MAJOR=235;
static int BUTTONS_MINOR=0;
static int count =1;                    //for    one cdev

#define DEVICE_NAME    "buttons-device"
#define button_gpio        GPIO_TO_PIN(6, 1)        //GPIO6[1]      MUX19_23_20
#define button2_gpio     GPIO_TO_PIN(0,6)          //GPIO0[6]      MUX1_7_4

static struct cdev *buttons_cdev;
static dev_t buttons_dev;
static struct class *buttons_class;

#define PINMUX1_         (unsigned long)ioremap(0x01c14124,4)
#define    PINMUX19_                  (unsigned long)ioremap(0x01c1416C,4)
#define    BINTEN                     (unsigned long)ioremap(0x01e26008,4)
#define IN_DATA01          (unsigned long)ioremap(0x01e26020,4)
#define IN_DATA67        (unsigned long)ioremap(0x01e26098,4)
#define    INTSTAT01                  (unsigned long)ioremap(0x01e260ac,4)
#define          INTSTAT67                (unsigned long)ioremap(0x01e26034,4)

/*中断的顶半部和底半部机制*/
void key_do_tasklet(unsigned long);
DECLARE_TASKLET(key_tasklet,key_do_tasklet,0);
void key_do_tasklet(unsigned long data)
{
```

```
mdelay(20);     //do    nothing    ,    just    for format .
}

/*sw6  中断函数*/
static    irqreturn_t    button_irq_han6(int    irq, void *id)
{
tasklet_schedule(&key_tasklet);
printk("sw6 down \n");                  //gpio6[1] = sw6

return IRQ_RETVAL(IRQ_HANDLED);
}
/*sw5  中断函数*/
static    irqreturn_t    button_irq_han5(int    irq, void *id)
{
tasklet_schedule(&key_tasklet);
printk("sw5 down \n");                  //gpio6[1] = sw5

return IRQ_RETVAL(IRQ_HANDLED);
}

/*用于中断申请的结构体*/
struct key_irq_desc {
        int gpio;
        int pin_setting;
                char *name;
```

```
};
/*定义 key_irqs 结构体*/
static struct key_irq_desc key_irqs [] = {
        { button_gpio,IRQ_TYPE_EDGE_FALLING, "sw6"},
        {button2_gpio,IRQ_TYPE_EDGE_FALLING,"sw5"},
};

static int buttons_open(struct inode *inode,struct file *file)
{
        unsigned int num;
        unsigned char err;
        unsigned char i;

        printk("open succeed \n");

        num=readl(PINMUX19_ );
        num=(num&(0xff0fffff))|(0x00800000);          //set funtion GPIO6[1]
        writel(num, PINMUX19_ );

        num=readl(PINMUX1_);
        num=(num&(0xffffff0f))|(0x00000080);
        writel(num,PINMUX1_);                                    //set funtion GPIO0[6]

        num=readl(BINTEN);          //INTERRUPT BINK    ENABLE
          num|=(1<<6);
```

```
        writel(num,BINTEN);


        for (i = 0; i < sizeof(key_irqs)/sizeof(key_irqs[0]); i++)
        {
            gpio_request(key_irqs[i].gpio,key_irqs[i].name);        /*gpio  申请*/
            gpio_direction_input(key_irqs[i].gpio);                 /*设置 gpio  方向*/
/*linux-2.6.33 内核需要将 irq_set_irq_type()改为 set_irq_type()*/
            irq_set_irq_type(gpio_to_irq(key_irqs[i].gpio),key_irqs[i].pin_setting);
/*设置中断触发边缘*/
             if(i==0)     /*sw6 的 gpio 中断申请 ， 其中 gpio_to_irq 用于获取中断号*/
                 {
        err=request_irq(gpio_to_irq(key_irqs[i].gpio),&button_irq_han6,0 ,key_irqs[i].name,(vo
id *)&key_irqs[i] );
            if (err)
            break;
                 }
          else    if(i==1)/*sw5 的 gpio 中断申请*/
            {
        err=request_irq(gpio_to_irq(key_irqs[i].gpio),&button_irq_han5,0 ,key_irqs[i].name,(vo
id *)&key_irqs[i] );
            if (err)
            break;
             }
        }
```

```
        if(err)
        {
                printk("button_irq fail!\n");
           return    -1;
         }
  return 0;
}

static int buttons_release(struct inode *inode,struct file *file)
 /*关闭释放 gpio，释放中断号*/
{
        int i;

        for (i = 0; i < sizeof(key_irqs)/sizeof(key_irqs[0]); i++) {
                gpio_free(key_irqs[i].gpio);
        disable_irq(gpio_to_irq(key_irqs[i].gpio));
                free_irq(gpio_to_irq(key_irqs[i].gpio), (void *)&key_irqs[i]);
        }

    printk("button release \n");
        return 0;
}
/*read 函数*/
static int buttons_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
{
```

```
    int err;
    unsigned    int    indata;
    unsigned    int     temp[2];

    indata=readl(IN_DATA01);
    indata=indata&(1<<6);
    if(indata>0)
        temp[0]=1;
    else
        temp[0]=0;

    indata=readl(IN_DATA67);
    indata=indata&(1<<1);
    if(indata>0)
        temp[1]=1;
    else
        temp[1]=0;

    err=copy_to_user(buff,(void *)temp,sizeof(temp));
    return err?-EFAULT:sizeof(temp);
}

/*文件操作函数映射*/
static struct    file_operations buttons_fops=
{
```

```
    .owner=THIS_MODULE,
    .open=buttons_open,
    .release=buttons_release,
    .read=buttons_read,
};

/*初始化，申请主次设备号，申请 cdev ,添加 cedv,申请类，创建设备文件节点*/
static int __init BUTTONS_init(void)
{
    int ret;

    if(BUTTONS_MAJOR)
    {
        buttons_dev=MKDEV(BUTTONS_MAJOR,BUTTONS_MINOR);
        ret=register_chrdev_region(buttons_dev,count,DEVICE_NAME);
    }
    else
    {
        ret=alloc_chrdev_region(&buttons_dev,BUTTONS_MINOR,count,DEVICE_NAME);
        BUTTONS_MAJOR=MAJOR(buttons_dev);
    }

    if(ret<0)
    {
        printk(KERN_ERR "register chrdev fail!");
```

```
        return -1;
    }

    buttons_cdev=cdev_alloc();

    if(buttons_cdev!=NULL)
    {
        cdev_init(buttons_cdev,&buttons_fops);
        buttons_cdev->ops=&buttons_fops;
        buttons_cdev->owner=THIS_MODULE;
        if(cdev_add(buttons_cdev,buttons_dev,count))
            printk(KERN_ERR "register cdev fail!");
        else
            printk(KERN_ERR "register success \n");
    }
    else
    {
        printk(KERN_ERR "register cdev err!");
        return -1;
    }


    buttons_class=class_create(THIS_MODULE,DEVICE_NAME);
    if(IS_ERR(buttons_class))
    {
```

```
        printk(KERN_ERR "register class err!");
        return -1;
    }

    device_create(buttons_class,NULL,buttons_dev,NULL,DEVICE_NAME);

    return 0;
}
/*模块卸载函数，释放 cdev,设备号，文件节点，类*/
static void __exit BUTTONS_exit(void)
{
    printk("chrdev exit!");
    cdev_del(buttons_cdev);
    unregister_chrdev_region(buttons_dev,count);
    device_destroy(buttons_class,buttons_dev);
    class_destroy(buttons_class);
}

module_init(BUTTONS_init);
module_exit(BUTTONS_exit);

MODULE_DESCRIPTION("buttons character");
MODULE_AUTHOR("Apple");
MODULE_LICENSE("GPL");
```

2.2 按键驱动编译

将光盘 demo\driver\button 中的 button.c 和 Makefile 文件复制到开发系统 ubuntu 以下目录（若目录不存在请先建立）：/home/tl/omapl138/demo/driver/button，并在进入目录的运行 Make 命令编译驱动程序：

```
Host#    /home/tl/omapl138/demo/driver/button
Host#    make
```

运行以上命令后，系统会根据 Makefile 文件的规则去编译整个驱动源码，产生了 button.ko 和其他中间文件，如下图所示：

```
tl@tl-desktop:~/omapl138/demo/driver/button$ make
make -C /home/tl/omapl138/linux-3.3 M=/home/tl/omapl138/demo/driver/button modul
es ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
make[1]: Entering directory `/home/tl/omapl138/linux-3.3'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/home/tl/omapl138/linux-3.3'
tl@tl-desktop:~/omapl138/demo/driver/button$ ls
button.c   button.mod.c  button.o  modules.order
button.ko  button.mod.o  Makefile  Module.symvers
```

Makefile 文件内容如下：

```
ifneq ($(KERNELRELEASE),)
obj-m := button.o
else
KDIR := /home/tl/omapl138/linux-3.3
all:
    make -C $(KDIR) M=$(PWD) modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
clean:
    rm -f *.ko *.o *.mod.o *.mod.c *.symvers    modul*
endif
```

```
ifneq ($(KERNELRELEASE),)

obj-m := button.o

else

KDIR := /home/tl/omapl138/linux-3.3

all:
	make -C $(KDIR) M=$(PWD) modules ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-

clean:
	rm -f *.ko *.o *.mod.o *.mod.c *.symvers  modul*

endif
```

2.3 按键驱动测试程序编写和解析

光盘中有测试程序镜像和源码，路径为：

button_test: demo\app\button\ button_test //测试程序镜像

button_test.c: demo\app\button\ button_test.c//测试程序源码

以下为测试程序的解释：

```
/*头文件*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>


unsigned char   quit_flag=0;
void *quit(void *arg);


int main(int argc,char** argv[])
{
    int button_fd;
```

```
    unsigned int button_values[2];
    unsigned char button_down_flag=0;//0:up    1:down

    pthread_t thread_id;    /*申明 thred_id*/

    button_fd=open("/dev/buttons-device",0); /*打开文件，驱动注册按键中断*/
    if(button_fd<0)
    {
        fprintf(stderr,"open failed!\n");
        exit(1);
    }
    pthread_create(&thread_id,NULL,(void *)*quit,NULL); /*创建线程 quit 函数*/
    while(quit_flag==0) ; /*等待用户按下 q 结束该循环*/
    sleep(1);
    close(button_fd); /*关闭文件*/
    return 0;
}

void *quit(void *arg) /*quit  函数*/
{
    while(getchar()!='q');
    quit_flag=1;
    return ((void *)0);
}
```

2.4 编译按键驱动测试程序

将光盘 demo\app\button 目录下的按键设备驱动测试程序源代码 button_test.c 拷贝到 Ubuntu 虚拟机的/home/tl/omapl138/demo/app/button 目录下（若目录不存在请先建立）.

测试程序使用了线程创建，所以在编译的时候需要加入静态库-lpthread。进入此目录执行“arm-none-linux-gnueabi-gcc button_test.c -o button_test -lpthread”后会在当前目录生成 button_test 文件，如下图所示：

```
tl@tl-desktop:~/omapl138/demo/app/button$ arm-none-linux-gnueabi-gcc button_test
.c -o button_test -lpthread
tl@tl-desktop:~/omapl138/demo/app/button$ ls
button_test  button_test.c
tl@tl-desktop:~/omapl138/demo/app/button$
```

2.5 测试 Linux 设备驱动程序

将 button.ko 和 button_test 文件都拷贝到开发板的如下路径：

button.ko：/home/tl/omapl138/demo/driver/button

button_test: /home/tl/omapl138/demo/app/button

首先加载 button.ko 文件，显示加载成功。然后执行 button_test 文件，然后按下按键。

Target: insmod /home/tl/omapl138/demo/driver/button/button.ko

Target: /home/tl/omapl138/demo/app/button/button_test

```
root@tl:~# insmod /home/tl/omapl138/demo/driver/button/button.ko
[  177.620768] register success
root@tl:~# /home/tl/omapl138/demo/app/button/button_test
[   38.138907] open succeed
[   40.487619] sw5 down
[   41.055365] sw6 down
[   41.848131] sw5 down
[   42.197924] sw6 down
```