实验 22 字符类 GPIOS

22.1 本章导读

本期实验是对前面知识的一个小结，将字符类驱动的框架和 GPIO 操作函数的结合，就可以完成一个字符类 GPIO 驱动。

22.1.1 工具

22.1.1.1 硬件工具

1）iTOP4412 开发板

2）U 盘或者 TF 卡

3）PC 机

4）串口

22.1.1.2 软件工具

1）虚拟机 Vmware

2）Ubuntu12.04.2

3）超级终端（串口助手）

4）源码文件夹“char_driver_leds”

22.1.2 预备课程

实验 21 字符驱动

22.1.3 视频资源

本节配套视频为“视频 22_字符类 GPIOS”

22.2 学习目标

本章需要学习以下内容：

将前面杂项设备中对 GPIO 操作的函数引入到字符设备中

22.3 实验操作

在视频教程中只做了简单的演示，因为所有的知识点在前面都已经介绍过了。

将“21_字符驱动”中的文件“char_driver.c”改为“char_driver_leds.c”，因为 char_driver_leds.c 文件中的代码有点长了，所以将宏定义和结构体定义放到头文件“char_driver_leds.h”中，如下图所示。

```
#ifndef _CHAR_DRIVER_LEDS_H_
#define _CHAR_DRIVER_LEDS_H_

#ifndef DEVICE_NAME
#define DEVICE_NAME "chardevnode"
#endif

#ifndef DEVICE_MINOR_NUM
#define DEVICE_MINOR_NUM 2
#endif

#ifndef DEV_MAJOR
#define DEV_MAJOR 0
#endif

#ifndef DEV_MINOR
#define DEV_MINOR 0
#endif

#ifndef REGDEV_SIZE
#define REGDEV_SIZE 3000
#endif

struct reg_dev
{
    char *data;
    unsigned long size;

    struct cdev cdev;
};
#endif
```

然后添加 GPIO 操作的头文件，以及自定义的头文件，如下图所示。

```
/*自定义头文件*/
#include "char_driver_leds.h"

/*Linux中申请GPIO的头文件*/
#include <linux/gpio.h>
/*三星平台的GPIO配置函数头文件*/
/*三星平台EXYNOS系列平台，GPIO配置参数宏定义头文件*/
#include <plat/gpio-cfg.h>
/*三星平台4412平台，GPIO宏定义头文件*/
#include <mach/gpio-exynos4.h>


MODULE_LICENSE("Dual BSD/GPL");
/*声明是开源的，没有内核版本限制*/
MODULE_AUTHOR("iTOPEET_dz");
/*声明作者*/
```

将两个 gpio 定义为数组，如下图所示。

```
static int led_gpios[] = {
    EXYNOS4_GPL2(0),EXYNOS4_GPK1(1),
};
#define LED_NUM     ARRAY_SIZE(led_gpios)
```

如下图所示，自定义一个 GPIO 初始化函数 gpio_init。

```
static int gpio_init(void){
    int i=0,ret;

    for(i=0;i<LED_NUM;i++){
        ret = gpio_request(led_gpios[i], "LED");
        if (ret) {
            printk("%s: request GPIO %d for LED failed, ret = %d\n", DEVICE_NAME,i,ret);
            return -1;
        }
        else{
            s3c_gpio_cfgpin(led_gpios[i], S3C_GPIO_OUTPUT);
            gpio_set_value(led_gpios[i], 1);
        }
    }

    return 0;
}
```

在驱动入口的函数中，创建设备节点成功之后再调用 GPIO 初始化函数，如下图所示。

```
    /*创建设备节点*/
    device_create(myclass,NULL,MKDEV(numdev_major,numdev_minor+i),NULL,DEVICE_NAME"%d",i);
}

ret = gpio_init();
if(ret){
    printk(KERN_EMERG "gpio_init failed!\n");
}

printk(KERN_EMERG "scdev_init!\n");
/*打印信息，KERN_EMERG表示紧急信息*/
return 0;
```

然后在驱动的出口函数中添加释放 GPIO 的代码，如下图所示。

```
static void scdev_exit(void)
{
    int i;
    printk(KERN_EMERG "scdev_exit!\n");

    /*除去字符设备*/
    for(i=0;i<DEVICE_MINOR_NUM;i++){
        cdev_del(&(my_devices[i].cdev));
        /*摧毁设备节点函数d*/
        device_destroy(myclass,MKDEV(numdev_major,numdev_minor+i));
    }
    /*释放设备class*/
    class_destroy(myclass);
    /*释放内存*/
    kfree(my_devices);

    /*释放GPIO*/
    for(i=0;i<LED_NUM;i++){
        gpio_free(led_gpios[i]);
    }

    unregister_chrdev_region(MKDEV(numdev_major,numdev_minor),DEVICE_MINOR_NUM);
}
```

然后在 ioctl 中添加 io 操作的代码，如下图所示。

```
/*IO操作*/
static long chardevnode_ioctl(struct file *file, unsigned int cmd, unsigned long arg){

    switch(cmd)
    {
        case 0:
        case 1:
            if (arg > LED_NUM) {
                return -EINVAL;
            }

            gpio_set_value(led_gpios[arg], cmd);
            break;

        default:
            return -EINVAL;
    }

    printk(KERN_EMERG "chardevnode_ioctl is success! cmd is %d,arg is %d \n",cmd,arg);

    return 0;
}
```

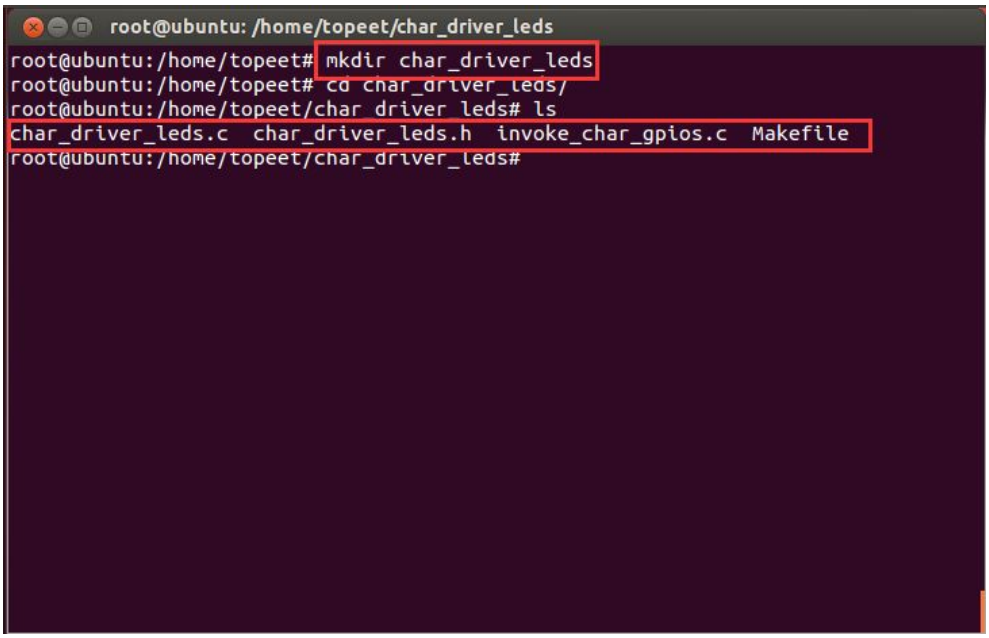
如下图所示，修改一下 Makefie 文件。

```
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#这里是编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o
obj-m += char_driver_leds.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -C就是指调用执行的路径
#$(KDIR)Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD)当前目录变量
#modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.mod.c *.o *.order *.ko *.mod.o *.symvers
```

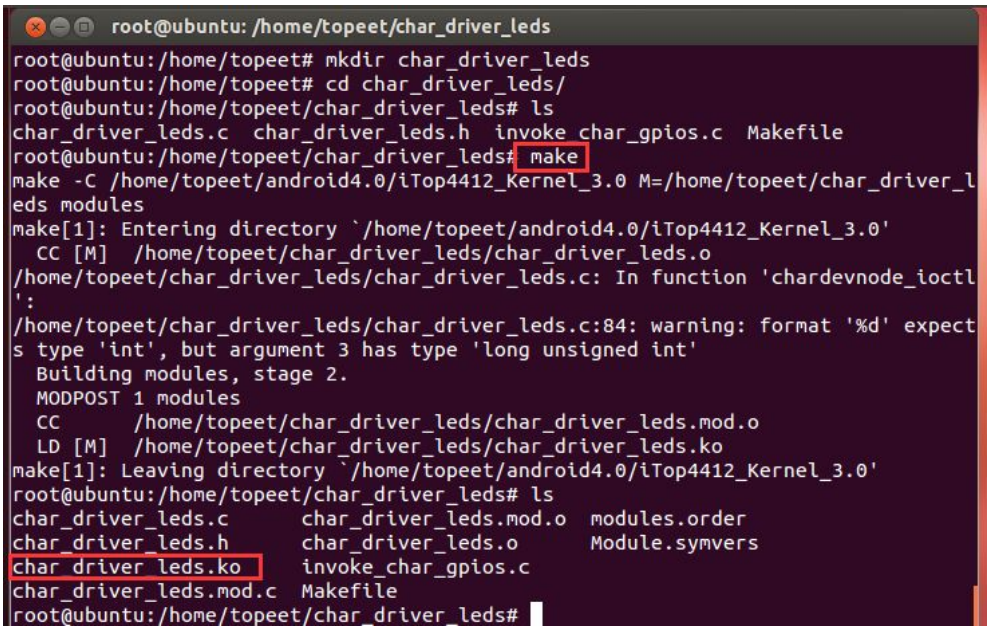
编写一个简单的应用“invoke_char_gpios.c”，如下图所示。

```c
#include <stdio.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

/*argv[1] is cmd , argv[2] is io_arg*/
int main(int argc , char **argv){
    int fd;
    char *lednode = "/dev/chardevnode0";

    /*O_RDWR只读打开,O_NDELAY非阻塞方式*/
    if((fd = open(lednode,O_RDWR|O_NDELAY))<0){
        printf("APP open %s failed!\n",lednode);
    }
    else{
        printf("APP open %s success!\n",lednode);
            ioctl(fd,atoi(argv[1]),atoi(argv[2]));
            printf("APP ioctl %s ,cmd is %s! io_arg is %s!\n",lednode,argv[1],argv[2]);
    }

    close(fd);
}
```

修改完成之后，在 Ubuntu 系统下使用命令“mkdir char_driver_leds”新建文件夹“char_driver_leds”，然后将修改好的驱动文件“char_driver_leds.c”、头文件“char_driver_leds.h”、Makefile 文件以及应用文件“invoke_char_gpios.c”拷贝到文件夹“char_driver_leds”中，如下图所示。



使用编译命令“make”编译驱动，如下图所示。



使用命令

“arm-none-linux-gnueabi-gcc -o invoke_char_gpios invoke_char_gpios.c -static”

编译应用程序“invoke_char_gpios”，如下图所示。

```
root@ubuntu: /home/topeet/char_driver_leds
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
  CC [M]  /home/topeet/char_driver_leds/char_driver_leds.o
/home/topeet/char_driver_leds/char_driver_leds.c: In function 'chardevnode_ioctl
':
/home/topeet/char_driver_leds/char_driver_leds.c:84: warning: format '%d' expect
s type 'int', but argument 3 has type 'long unsigned int'
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/topeet/char_driver_leds/char_driver_leds.mod.o
  LD [M]  /home/topeet/char_driver_leds/char_driver_leds.ko
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
root@ubuntu:/home/topeet/char_driver_leds# ls
char_driver_leds.c      char_driver_leds.mod.o  modules.order
char_driver_leds.h      char_driver_leds.o      Module.symvers
char_driver_leds.ko     invoke_char_gpios.c
char_driver_leds.mod.c  Makefile
root@ubuntu:/home/topeet/char_driver_leds# arm-none-linux-gnueabi-gcc -o invoke_
char_gpios invoke_char_gpios.c -static
root@ubuntu:/home/topeet/char_driver_leds# ls
char_driver_leds.c      char_driver_leds.mod.o  Makefile
char_driver_leds.h      char_driver_leds.o      modules.order
char_driver_leds.ko     invoke_char_gpios       Module.symvers
char_driver_leds.mod.c  invoke_char_gpios.c
root@ubuntu:/home/topeet/char_driver_leds#
```

将生成的驱动模块“char_driver_leds.ko”以及应用“invoke_char_gpios”拷贝到 U 盘。

启动开发板，将 U 盘插入开发板，使用命令“mount /dev/sda1 /mnt/udisk/”加载 U 盘，如下图所示。

```
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
```

使用命令“insmod /mnt/udisk/char_driver_leds.ko”加载驱动模块，如下图所示。

```
COM1
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]# insmod /mnt/udisk/char_driver_leds.ko
[   23.528827] numdev_major is 0!
[   23.530533] numdev_minor is 0!
[   23.533485] adev_region req 249 !
[   23.545821] cdev_add 0 is success!
[   23.555677] cdev_add 1 is success!
[   23.565667] scdev_init!
[root@iTOP-4412]#
```

使用命令“./mnt/udisk/invoke_char_gpios 0 1”，运行应用。参数 1 为命令，参数 2 为 GPIO 编号，如下图所示。

```
COM1
[root@iTOP-4412]# ./mnt/udisk/invoke_char_gpios 0 1
[  156.096445] chardevnode_open is success!
APP open /dev/cha[  156.104623] chardevnode_ioctl is success! cmd is 0,arg is 1
rdevnode0 success!
APP ioctl /dev/ch[  156.120194] chardevnode_release is success!
ardevnode0 ,cmd is 0! io_arg is 1!
[root@iTOP-4412]#
```

运行如上图所示命令之后，可以看到小灯会灭一个。

另外如果大家想将两个设备节点定义为不同的操作，那么就需要额外的定义“file_operations”。

然后，在如下图所示的代码处，分别对设备定义不同的“file_operations”即可。

```
static void reg_init_cdev(struct reg_dev *dev,int index){
    int err;
    int devno = MKDEV(numdev_major,numdev_minor+index);

    /*数据初始化*/
    cdev_init(&dev->cdev,&my_fops);
    dev->cdev.owner = THIS_MODULE;
    dev->cdev.ops = &my_fops;

    /*注册到系统*/
    err = cdev_add(&dev->cdev,devno,1);
    if(err){
        printk(KERN_EMERG "cdev_add %d is fail! %d\n",index,err);
    }
    else{
        printk(KERN_EMERG "cdev_add %d is success!\n",numdev_minor+index);
    }
}
```