实验 15 LEDS 驱动二

15.1 本章导读

考虑到用户在实际开发中可能需要更多的 GPIO，本实验给用户提供 32 个 GPIO，需要注意的是有一部分是复用的 GPIO，需要首先对内核进行配置和编译之后才能使用。例如 WIFI 模块和摄像头接口用到的 GPIO ,如果当做 GPIO 使用 ,那么就无法使用 WIFI 模块和摄像头模块。

15.1.1 工具

15.1.1.1 硬件工具

1）iTOP4412 开发板

2）U 盘或者 TF 卡

3）PC 机

4）串口

15.1.1.2 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“gpios”

15.1.2 预备课程

实验 14 LEDS 驱动一

15.1.3 视频资源

本节配套视频为“视频 15_LEDS 驱动二”

15.2 学习目标

本章需要学习以下内容：

如何将功能复用的 IO 配置为 GPIO

15.3 操作过程

因为前面关于 GPIO 的使用都已经介绍的差不多了，现在直接给大家介绍操作过程。

如下图所示，提供了 32 个 GPIO。

```
/*led的两个IO，网络是KP_COL0，VDD50_EN*/
/*蜂鸣器的1个IO，网络是MOTOR_PWM*/

/*矩阵键盘的8个IO，网络是CHG_FLT，HOOK_DET，CHG_UOK，XEINT14_BAK，
GM_INT1，6260_GPIO1，CHG_COK，XEINT29/KP_ROW13/ALV_DBG25*/

/*摄像头的14个IO，网络是CAM_MCLK，CAM2M_RST，CAM2M_PWDN，
CAM_D5，CAM_D7，CAM_D6，CAM_D4，CAM_D3，CAM_D2，CAM_D1，
CAM_PCLK，CAM_D0，CAM_VSYNC，CAM_HREF。
I2C_SDA7，I2C_SCL7也是可以设置为GPIO，不过总线一般不要去动它*/

/*WIFI模块的7个IO，WIFI_D3，WIFI_CMD，WIFI_D1，WIFI_CLK，WIFI_D0，WIFI_D2，GPC1_1*/
/*串口RX和TX等也是可以设置为GPIO，一般不要动它*/

/*数组中有32个引出到端子或者模块的IO，还有类似sd卡等也是可以作为GPIO，
其它引到连接器但是没有使用的GPIO等等*/
/*SCP管脚编号和POP的稍微有点不同，下面是SCP的*/
```

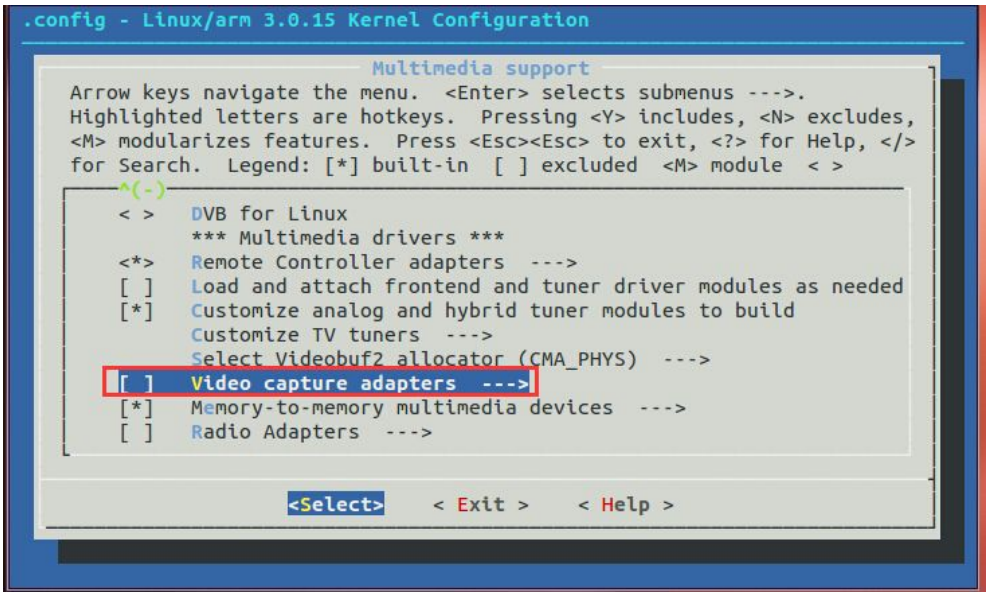
先需要去掉调用了这些 GPIO 的相关驱动。

1）去掉摄像头驱动 VIDEO_OV5640

Device Drivers --->

Multimedia support(MEDIA_SUPPORT [=y]) --->

Video capture adapters(VIDEO_CAPTURE_DRIVERS [=y])（去掉） --->



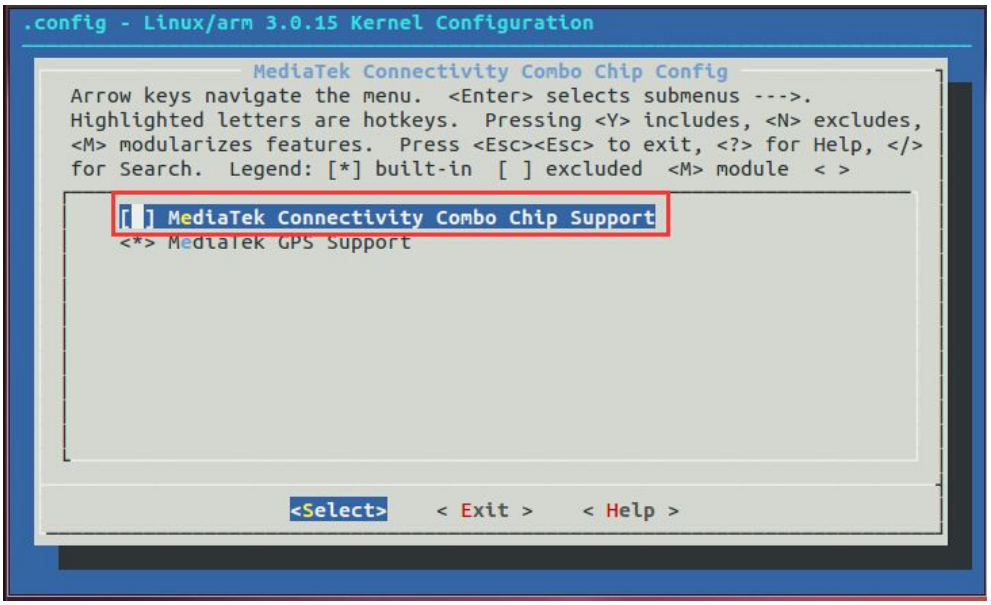
2）去掉 WIFI 驱动 MTK_COMBO_CHIP_MT662

Device Drivers --->

MediaTek Connectivity Combo Chip Config --->

MediaTek Connectivity Combo Chip Support (MTK_COMBO [=y])（去掉） --->

Select Chip (<choice> [=y]) --->

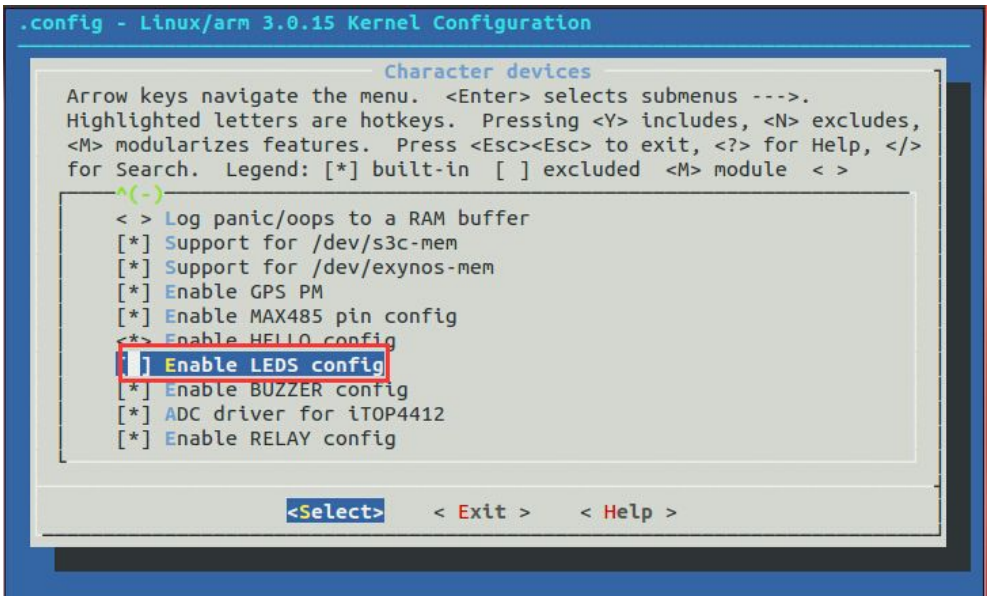


3）去掉 leds 的驱动

Device Drivers --->

Character devices --->

Enable LEDS config --->

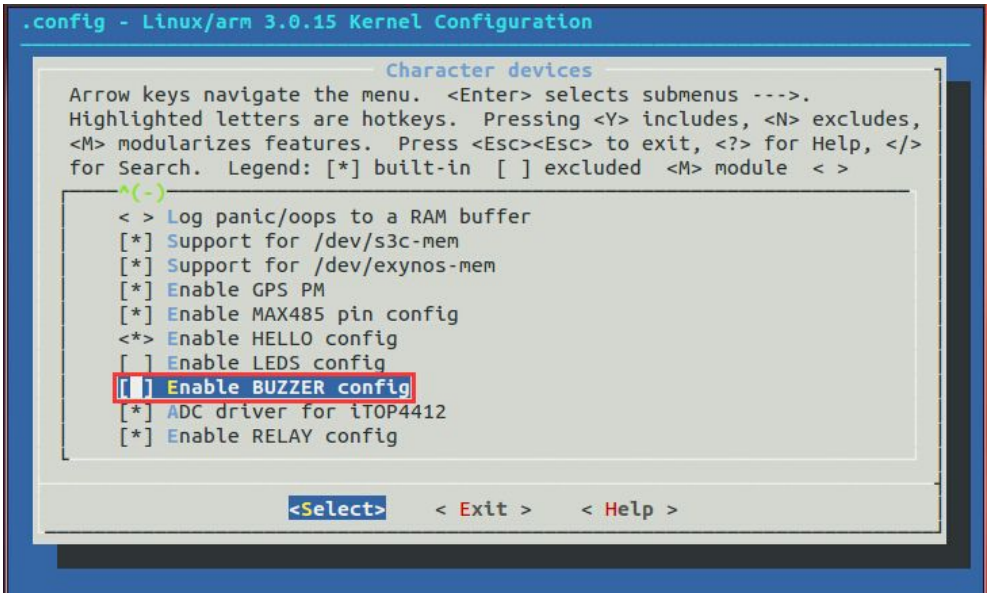


4）去掉 Buzzer 的驱动

Device Drivers --->

Character devices --->

Enable BUZZER config --->



修改完之后重新编译内核，将新生成的内核二进制文件 zImage 烧写到开发板。

接着将前一个实验的 leds.c 改为 gpios.c。

修改一下 Makefile 文件，如下图所示。

```
#!/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itop4412_hello.o
obj-m += gpios.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 *.o
```

然后将这些 GPIO 打包为一个数组，数组如下图所示，然后定义一下数组长度 LED_NUM。

```
static int led_gpios[] = {
    EXYNOS4_GPL2(0),EXYNOS4_GPK1(1),
    EXYNOS4_GPD0(0),

    EXYNOS4_GPX1(0),EXYNOS4_GPX1(3),EXYNOS4_GPX1(5),EXYNOS4_GPX1(6),
    EXYNOS4_GPX3(0),EXYNOS4_GPX2(6),EXYNOS4_GPX2(7),EXYNOS4_GPX3(5),

    EXYNOS4212_GPJ1(3),EXYNOS4_GPL0(1),EXYNOS4_GPL0(3),EXYNOS4212_GPJ1(0),
    EXYNOS4212_GPJ1(2),EXYNOS4212_GPJ1(1),EXYNOS4212_GPJ0(7),EXYNOS4212_GPJ0(6),
    EXYNOS4212_GPJ0(5),EXYNOS4212_GPJ0(4),EXYNOS4212_GPJ0(0),EXYNOS4212_GPJ0(3),
    EXYNOS4212_GPJ0(1),EXYNOS4212_GPJ0(2),

    EXYNOS4_GPK3(6),EXYNOS4_GPK3(1),EXYNOS4_GPK3(4),EXYNOS4_GPK3(0),
    EXYNOS4_GPK3(3),EXYNOS4_GPK3(5),EXYNOS4_GPC1(1),
};

#define LED_NUM     ARRAY_SIZE(led_gpios)
```

将设备节点的名称修改为 hello_gpio，如下图所示。

```
#define DRIVER_NAME "hello_ctl"
#define DEVICE_NAME "hello_gpio"


MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("TOPEET");
```

如下图所示，先在 probe 函数中初始化。

```
static int hello_probe(struct platform_device *pdv){
    int ret,i;

    printk(KERN_EMERG "\tinitialized\n");

    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", DRIVER_NAME,
                    i, ret);
            }
        else{
            s3c_gpio_cfgpin(led_gpios[i], S3C_GPIO_OUTPUT);
            gpio_set_value(led_gpios[i], 1);
        }
    }

    gpio_set_value(led_gpios[2], 0);

    misc_register(&hello_dev);
    if(ret<0)
    {
        printk("leds:register device failed!\n");
        goto exit;
    }
    return 0;

exit:
    misc_deregister(&hello_dev);
    return ret;
    return 0;
}
```

如下图所示，然后是 ioctl 函数中写一个简单的 switch 语句。

```
static long hello_ioctl( struct file *files, unsigned int cmd, unsigned long arg){
    printk("cmd is %d,arg is %d\n",cmd,arg);

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

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

        default:
            return -EINVAL;
    }

    gpio_set_value(led_gpios[2], 0);

    return 0;
}
```

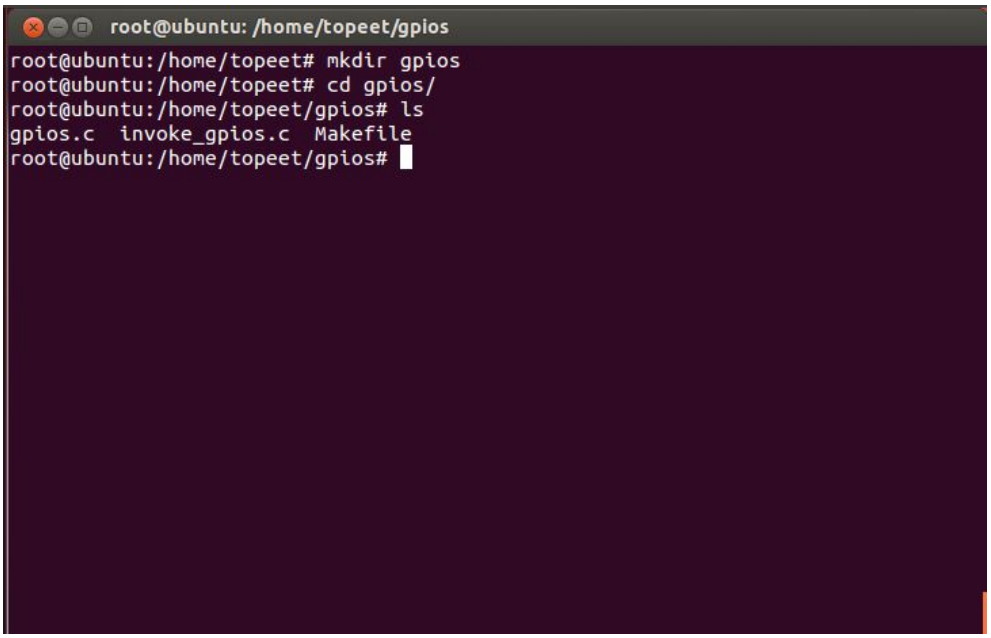
如下图所示，最后是在 remove 函数中添加 gpio_free 释放 GPIO。

```
static int hello_remove(struct platform_device *pdv){
    int i;

    printk(KERN_EMERG "\tremove\n");

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

    misc_deregister(&hello_dev);
    return 0;
}
```

接着简单的修改一下应用。

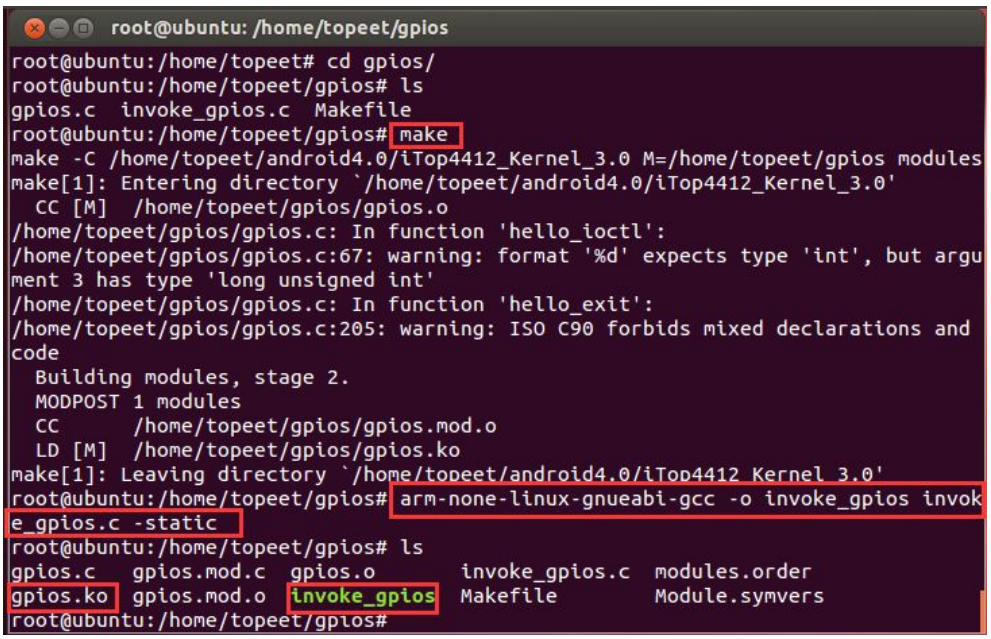
```
int main(int argc , char **argv){
    int fd,i,cmd=2;
    char *hello_node = "/dev/hello_gpio";
    char *cmd0 = "0";
    char *cmd1 = "1";

    printf("argv[0] is %s;argv[1] is %s;",argv[0],argv[1]);

    if(strcmp(argv[1], cmd0) == 0){
        cmd = 0;
        printf("cmd is 0!\n");
    }
    if(strcmp(argv[1], cmd1) == 0){
        cmd = 1;
        printf("cmd is 1!\n");
    }

/*O_RDWR只读打开,O_NDELAY非阻塞方式*/
    if((fd = open(hello_node,O_RDWR|O_NDELAY))<0){
        printf("APP open %s failed!\n",hello_node);
    }
    else{
        printf("APP open %s success!\n",hello_node);
        for(i=0;i<GPIOS;i++){
            ioctl(fd,cmd,i);
            printf("APP ioctl %s ,cmd is %d,i is %d!\n",hello_node,cmd,i);
        }
    }

    close(fd);
}
```

在 Ubuntu 系统下新建 gpios 文件夹，将写好的 gpios、编译脚本以及应用拷贝到 gpios 文件夹下，如下图所示。



使用 Makefile 命令编译驱动，然后使用

“arm-none-linux-gnueabi-gcc -o invoke_gpios invoke_gpios.c -static”命令编译应用，如下图所示。



将上图中的文件 invoke_gpios 和 gpios.ko 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 盘符。

使用命令“insmod /mnt/udisk/gpios.ko”加载驱动 gpios.ko，

使用命令“./mnt/udisk/invoke_gpios 0”或者“./mnt/udisk/invoke_gpios 1”运行小应用 invoke_gpios，如下图所示。

```
[root@iTOP-4412]# insmod /mnt/udisk/gpios.ko
[  313.913783] HELLO WORLD enter!
[  313.915457]  initialized
[  313.920964]  DriverState is 0
[root@iTOP-4412]# lsmod
gpios 2587 0 - Live 0xbf000000
[root@iTOP-4412]# ./mnt/udisk/invoke_gpios 0
argv[0] is ./mnt/[  329.128562] hello open
[  329.130901] cmd is 0,arg is 0
[  329.133844] cmd is 0,arg is 1
[  329.136811] cmd is 0,arg is 2
[  329.139739] cmd is 0,arg is 3
[  329.142724] cmd is 0,arg is 4
[  329.145660] cmd is 0,arg is 5
[  329.148592] cmd is 0,arg is 6
[  329.151553] cmd is 0,arg is 7
[  329.154492] cmd is 0,arg is 8
[  329.157455] cmd is 0,arg is 9
[  329.160431] cmd is 0,arg is 10
[  329.163432] cmd is 0,arg is 11
[  329.166482] cmd is 0,arg is 12
[  329.169508] cmd is 0,arg is 13
[  329.172558] cmd is 0,arg is 14
[  329.175595] cmd is 0,arg is 15
[  329.178621] cmd is 0,arg is 16
[  329.181671] cmd is 0,arg is 17
[  329.184697] cmd is 0,arg is 18
[  329.187747] cmd is 0,arg is 19
[  329.190786] cmd is 0,arg is 20
[  329.193854] cmd is 0,arg is 21
```

如上图所示，使用命令“./mnt/udisk/invoke_gpios 0”之后灯会灭，然后其它的 GPIO 也会都成为低电平。

使用命令“./mnt/udisk/invoke_gpios 0”之后灯会亮，然后其它的 GPIO 也会都成为高电平。

也可以检查一下运行应用之后有没有错误，如果有错误，多半是因为没有将调用对应 GPIO 的驱动去除，导致 GPIO 被占用了。