

## iTOP-4412 实现中断驱动例程

大家好，今天我们来学习一下 linux 中断处理驱动的编写，本节我们实现的功能是通过开发板上的按键来控制 led 发光二极管，在之前的章节我们学习了 led 驱动的编写，在掌握了 led 驱动的编写以后，如果要实现按键控制 led 的功能，大家可能会想到可以在 led 的驱动里面使用轮询的方式一直查询按键的状态，如果有按键按下就设置 led 的状态。通过这种方式可以实现按键控制 led 的功能，但是通过这样的方式有一个缺点就是 led 驱动会占用 cpu，这样 cpu 的利用率就大大降低了，所以我们可以通过中断的方式来实现。这样 cpu 就可以去做其他的事情了，当有按键中断触发的时候才会去设置 led。

ARM 架构 linux 内核中，有 5 种常见的异常，其中中断异常是其一，Linux 内核将所有中断统一编号，使用一个 irq\_desc 结构体来描述这些中断，里面记录了中断名称、中断状态、中断标记、并提供了中断的底层硬件访问函数（如：清除、屏蔽、使能中断），提供了这个中断的处理函数入口，通过它还可以调用用户注册的中断处理函数。linux 内核的中断体系已经很完善了，驱动工程师需要做的就是调用 request\_irq 函数向内核注册中断处理函数，下面我们来看看 request\_irq 函数的定义：

```
static inline int __must_check
```

```
request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
```

```
const char *name, void *dev)
```

第一个参数 irq：中断号，与平台架构相关；

第二个参数 handler：用户中断处理函数；

第三个参数 flags：中断标记

第四个参数 devname：中断名字，可以通过 cat /proc/interrupts 查看；

第五个参数 dev\_id：在 free\_irq 中有用，也用做区分中断处理函数；

有注册就得对应着有注销，驱动的注销函数是 `free_irq`，其定义如下：

```
void free_irq(unsigned int irq, void *dev_id)
```

第一个参数 `irq`：中断号，与 `request_irq` 中的 `irq` 一致，用于定位 `action` 链表；

第二个参数 `dev_id`：用于在 `action` 链表中找到要卸载的表项；同一个中断的不同中断处理函数必须使用不同的 `dev_id` 来区分，这就要求在注册中断共享时参数 `dev_id` 必须唯一。

下面我们来看一下中断按键的驱动，代码如下：

```
#include <linux/init.h>
```

```
#include <linux/module.h>
```

```
#include <linux/kernel.h>
```

```
#include <linux/fs.h>
```

```
#include <linux/interrupt.h>
```

```
#include <linux/irq.h>
```

```
#include <mach/gpio.h>
```

```
#include <plat/gpio-cfg.h>
```

```
#include <linux/miscdevice.h>
```

```
#include <linux/platform_device.h>
```

```
#include <mach/regs-gpio.h>
```

```
#include <asm/io.h>
```

```
#include <linux/regulator/consumer.h>

#include <linux/delay.h>

#define IRQ_DEBUG

#ifndef IRQ_DEBUG

#define DPRINTK(x...) printk("IRQ_CTL DEBUG:" x)

#else

#define DPRINTK(x...)

#endif

#define DRIVER_NAME "irq_test"

static int led_gpios[] = {

    EXYNOS4_GPL2(0),

    EXYNOS4_GPK1(1),

};

#define LED_NUM    ARRAY_SIZE(led_gpios)

#if 1

static irqreturn_t eint9_interrupt(int irq, void *dev_id) {
```

```
printk("%s(%d)\n", __FUNCTION__, __LINE__);
```

```
if(gpio_get_value(led_gpios[0]))
```

```
    gpio_set_value(led_gpios[0], 0);
```

```
else
```

```
    gpio_set_value(led_gpios[0], 1);
```

```
return IRQ_HANDLED;
```

```
}
```

```
static irqreturn_t eint10_interrupt(int irq, void *dev_id) {
```

```
printk("%s(%d)\n", __FUNCTION__, __LINE__);
```

```
if(gpio_get_value(led_gpios[1]))
```

```
    gpio_set_value(led_gpios[1], 0);
```

```
else
```

```
    gpio_set_value(led_gpios[1], 1);
```

```
return IRQ_HANDLED;
```

```
}

#endif

static int irq_probe(struct platform_device *pdev)

{

    int ret, i;

    char *banner = "irq_test Initialize\n";


    printk(banner);


    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,

                led_gpios[i], ret);

            return ret;

        }


        s3c_gpio_cfgpin(led_gpios[i], S3C_GPIO_OUTPUT);

        gpio_set_value(led_gpios[i], 0);

    }
```

```
ret = gpio_request(EXYNOS4_GPX1(1), "EINT9");
```

```
if (ret) {
```

```
    printk("%s: request GPIO %d for EINT9 failed, ret = %d\n", DRIVER_NAME,
```

```
           EXYNOS4_GPX1(1), ret);
```

```
    return ret;
```

```
}
```

```
s3c_gpio_cfgpin(EXYNOS4_GPX1(1), S3C_GPIO_SFN(0xF));
```

```
s3c_gpio_setpull(EXYNOS4_GPX1(1), S3C_GPIO_PULL_UP);
```

```
gpio_free(EXYNOS4_GPX1(1));
```

```
ret = gpio_request(EXYNOS4_GPX1(2), "EINT10");
```

```
if (ret) {
```

```
    printk("%s: request GPIO %d for EINT10 failed, ret = %d\n", DRIVER_NAME,
```

```
           EXYNOS4_GPX1(2), ret);
```

```
    return ret;
```

```
}
```

```
s3c_gpio_cfgpin(EXYNOS4_GPX1(2), S3C_GPIO_SFN(0xF));
```

```
s3c_gpio_setpull(EXYNOS4_GPX1(2), S3C_GPIO_PULL_UP);
```

```
gpio_free(EXYNOS4_GPX1(2));

#if 1

ret = request_irq(IRQ_EINT(9), eint9_interrupt,

IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint9", pdev);

if (ret < 0) {

printk("Request IRQ %d failed, %d\n", IRQ_EINT(9), ret);

goto exit;

}

ret = request_irq(IRQ_EINT(10), eint10_interrupt,

IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint10", pdev);

if (ret < 0) {

printk("Request IRQ %d failed, %d\n", IRQ_EINT(10), ret);

goto exit;

}

#endif

return 0;

exit:

return ret;

}
```

```
static int irq_remove (struct platform_device *pdev)
{
    return 0;
}
```

```
static int irq_suspend (struct platform_device *pdev, pm_message_t state)
{
    DPRINTK("irq suspend:power off!\n");
    return 0;
}
```

```
static int irq_resume (struct platform_device *pdev)
{
    DPRINTK("irq resume:power on!\n");
    return 0;
}
```

```
static struct platform_driver irq_driver = {
    .probe = irq_probe,
    .remove = irq_remove,
```



```
.suspend = irq_suspend,  
  
.resume = irq_resume,  
  
.driver = {  
  
    .name = DRIVER_NAME,  
  
    .owner = THIS_MODULE,  
  
},  
};  
  
static void __exit irq_test_exit(void)  
{  
  
    platform_driver_unregister(&irq_driver);  
}  
  
static int __init irq_test_init(void)  
{  
  
    return platform_driver_register(&irq_driver);  
}  
  
module_init(irq_test_init);  
  
module_exit(irq_test_exit);
```

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

首先我们来看下这个驱动的入口函数 `irq_test_init`，定义如下：

```
static int __init irq_test_init(void)
{
    return platform_driver_register(&irq_driver);
}
```

该函数想内核注册一个 `irq_driver` 类型的设备，`irq_driver` 定义如下：

```
static struct platform_driver irq_driver = {
    .probe = irq_probe,
    .remove = irq_remove,
    .suspend = irq_suspend,
    .resume = irq_resume,
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
    },
};
```

从这个结构里我们可以看到驱动的探测函数是 `irq_probe`，注销函数是 `irq_remove`，休眠调用的函数是 `irq_suspend`，唤醒调用的函数是 `irq_resume`，驱动的名字是 `DRIVER_NAME`，`DRIVER_NAME` 是个宏定义，如下：

```
#define DRIVER_NAME "irq_test"
```

接下来我们看看驱动的探测函数 `irq_probe`，这个函数也是这个驱动里面最主要的函数，这个函数一开始掉欧

勇 `printk` 打印一条信息，接着是一个 `for` 循环，初始化 `led` 的 `GPIO`，如下：

```
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,
            led_gpios[i], ret);

        return ret;
    }

    s3c_gpio_cfgpin(led_gpios[i], S3C_GPIO_OUTPUT);
    gpio_set_value(led_gpios[i], 0);
}
```

然后是初始化两个按键的 `gpio`，设置为中断模式，如下所示：

```
ret = gpio_request(EXYNOS4_GPX1(1), "EINT9");

if (ret) {
    printk("%s: request GPIO %d for EINT9 failed, ret = %d\n", DRIVER_NAME,
        EXYNOS4_GPX1(1), ret);

    return ret;
}
```

```
s3c_gpio_cfgpin(EXYNOS4_GPX1(1), S3C_GPIO_SFN(0xF));
```

```
s3c_gpio_setpull(EXYNOS4_GPX1(1), S3C_GPIO_PULL_UP);
```

```
gpio_free(EXYNOS4_GPX1(1));
```

```
ret = gpio_request(EXYNOS4_GPX1(2), "EINT10");
```

```
if (ret) {
```

```
    printk("%s: request GPIO %d for EINT10 failed, ret = %d\n", DRIVER_NAME,
```

```
    EXYNOS4_GPX1(2), ret);
```

```
    return ret;
```

```
}
```

```
s3c_gpio_cfgpin(EXYNOS4_GPX1(2), S3C_GPIO_SFN(0xF));
```

```
s3c_gpio_setpull(EXYNOS4_GPX1(2), S3C_GPIO_PULL_UP);
```

```
gpio_free(EXYNOS4_GPX1(2));
```

接着调用中断注册函数 request\_irq 向内核注册中断处理函数，如下：

```
ret = request_irq(IRQ_EINT(9), eint9_interrupt,
```

```
    IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint9", pdev);
```

```
if (ret < 0) {
```

```
    printk("Request IRQ %d failed, %d\n", IRQ_EINT(9), ret);
```

```
    goto exit;
```

```
}

ret = request_irq(IRQ_EINT(10), eint10_interrupt,

                IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint10", pdev);

if (ret < 0) {

    printk("Request IRQ %d failed, %d\n", IRQ_EINT(10), ret);

    goto exit;

}
```

上面的中断注册函数分别注册了两个中断处理函数 eint9\_interrupt 和 eint10\_interrupt。

eint9\_interrupt 的定义如下：

```
static irqreturn_t eint9_interrupt(int irq, void *dev_id) {

    printk("%s(%d)\n", __FUNCTION__, __LINE__);

    if(gpio_get_value(led_gpios[0]))

        gpio_set_value(led_gpios[0], 0);

    else

        gpio_set_value(led_gpios[0], 1);

    return IRQ_HANDLED;

}
```

这个函数首先会打印一句信息，然后是获取 led 的状态，把状态取反。

eint10\_interrupt 的定义如下：

```
static irqreturn_t eint10_interrupt(int irq, void *dev_id) {  
  
    printk("%s(%d)\n", __FUNCTION__, __LINE__);  
  
    if(gpio_get_value(led_gpios[1]))  
        gpio_set_value(led_gpios[1], 0);  
    else  
        gpio_set_value(led_gpios[1], 1);  
  
    return IRQ_HANDLED;  
}
```

这个函数和 eint9\_interrupt 的功能类似。

其他的函数就和前面讲的 led 驱动里面的函数基本一样了，唯一的区别是我们在注销函数 irq\_remove 里面使用了 free\_irq 函数来注销之前初测的中断处理函数。

把这个驱动放到内核的 driver/char 目录下面，如下图所示：

```

Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/drivers/char
ar#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/drivers/char
ar#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/drivers/char
ar# ls
agp                hangcheck-timer.c  modules.builtin    s3c_mem.c
apm-emulation.c    hpet.c             modules.order       s3c_mem.h
applicom.c          hw_random          msm_smd_pkt.c      scc.h
applicom.h          i8k.c              mspec.c            scx200_gpio.c
bfin-otp.c          ipmi                mwave              snsc.c
briq-panel.c        itop4412_adc.c     nsc_gpio.c         snsc_event.c
bsr.c               itop4412_buzzer.c  nvram.c            snsc.h
dcc_tty.c           itop4412_irq.c     nwbutton.c         sonypi.c
ds1302.c            itop4412_leds.c    nwbutton.h         tb0219.c
ds1620.c            Kconfig            nwflash.c          tlclk.c
dsp56k.c            lp.c               pc8736x_gpio.c     toshiba.c
dtlk.c              Makefile           pcmcia             tpm
efirtc.c            max485_ctl.c       ppdev.c            ttyprintk.c
exynos_mem.c        mbc.c              ps3flash.c         uv_mmtimer.c
generic_nvram.c     mbc.h              ramoops.c          viotape.c
genrtc.c            mem.c              random.c           virtio_console.c
gps.c               misc.c             raw.c              xilinx_hwicap
gps.h               mmtimer.c          rtc.c
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/drivers/char

```

然后打开 driver/char 目录下面的 Makefile，添加 “obj-y += itop4412\_irq.o”，如下图所示：

```

Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/drivers/char
obj-$(CONFIG_PCMCIA) += pcmcia/
obj-$(CONFIG_IPMI_HANDLER) += ipmi/

obj-$(CONFIG_HANGCHECK_TIMER) += hangcheck-timer.o
obj-$(CONFIG_TCG_TPM) += tpm/

obj-$(CONFIG_DCC_TTY) += dcc_tty.o
obj-$(CONFIG_PS3_FLASH) += ps3flash.o
obj-$(CONFIG_RAMOOOPS) += ramoops.o

obj-$(CONFIG_JS_RTC) += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_S3C_MEM) += s3c_mem.o
obj-y += gps.o

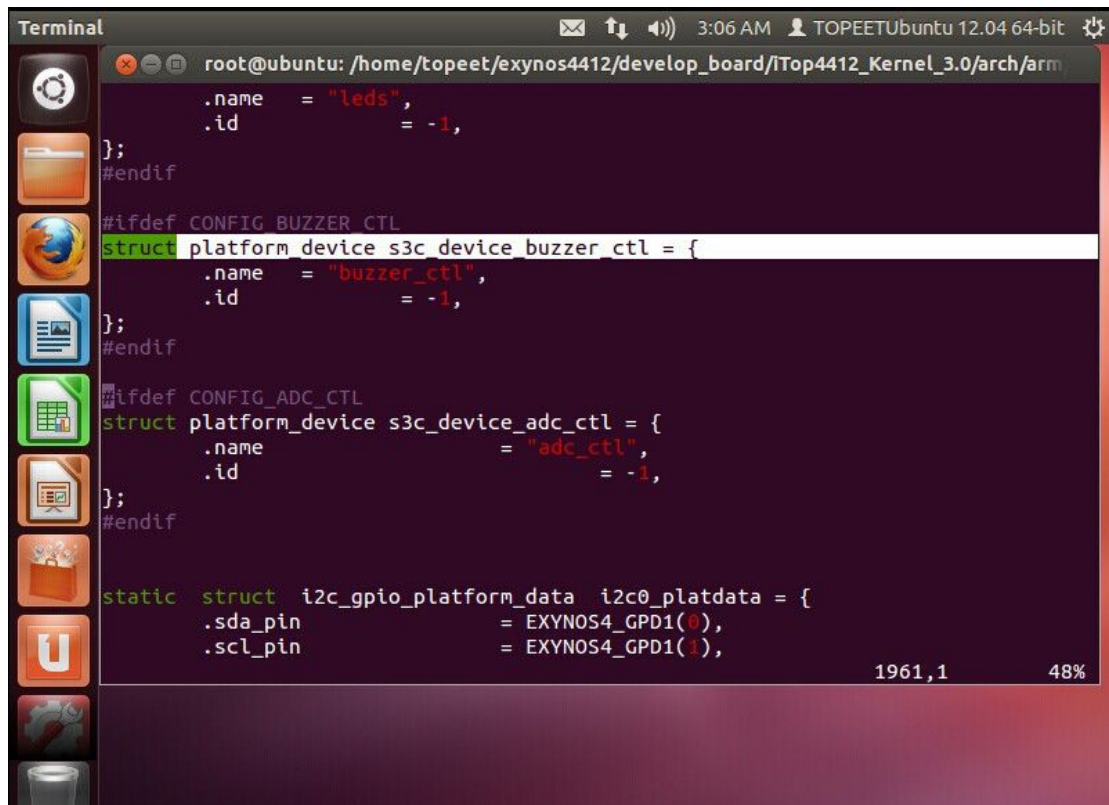
obj-$(CONFIG_MAX485_CTL) += max485_ctl.o
obj-$(CONFIG_LEDS_CTL) += itop4412_leds.o
obj-$(CONFIG_BUZZER_CTL) += itop4412_buzzer.o
obj-$(CONFIG_ADC_CTL) += itop4412_adc.o
obj-y += itop4412_irq.o

obj-$(CONFIG_EXYNOS_MEM) += exynos_mem.o
77,1 Bot

```

然后打开 arch/arm/mach-exynos/mach-itop4412.c 文件，找到 “struct platform\_device

s3c\_device\_buzzer\_ctl”，如下图所示：

A terminal window titled "Terminal" showing the contents of the file /home/topeet/exynos4412/develop\_board/iTop4412\_Kernel\_3.0/arch/arm/mach-itop4412.c. The code defines several platform devices. The line "struct platform\_device s3c\_device\_buzzer\_ctl = {" is highlighted. The code includes definitions for "leds", "buzzer\_ctl", "adc\_ctl", and "i2c\_gpio\_platform\_data". The terminal status bar at the bottom shows "1961,1" and "48%".

```
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/arch/arm
    .name  = "leds",
    .id    = -1,
};
#endif

#ifdef CONFIG_BUZZER_CTL
struct platform_device s3c_device_buzzer_ctl = {
    .name  = "buzzer_ctl",
    .id    = -1,
};
#endif

#ifdef CONFIG_ADC_CTL
struct platform_device s3c_device_adc_ctl = {
    .name  = "adc_ctl",
    .id    = -1,
};
#endif

static struct i2c_gpio_platform_data i2c0_platdata = {
    .sda_pin = EXYNOS4_GPD1(0),
    .scl_pin = EXYNOS4_GPD1(1),
};
```

然后在它的下面添加下面的信息：

```
struct platform_device s3c_device_irq_test = {
```

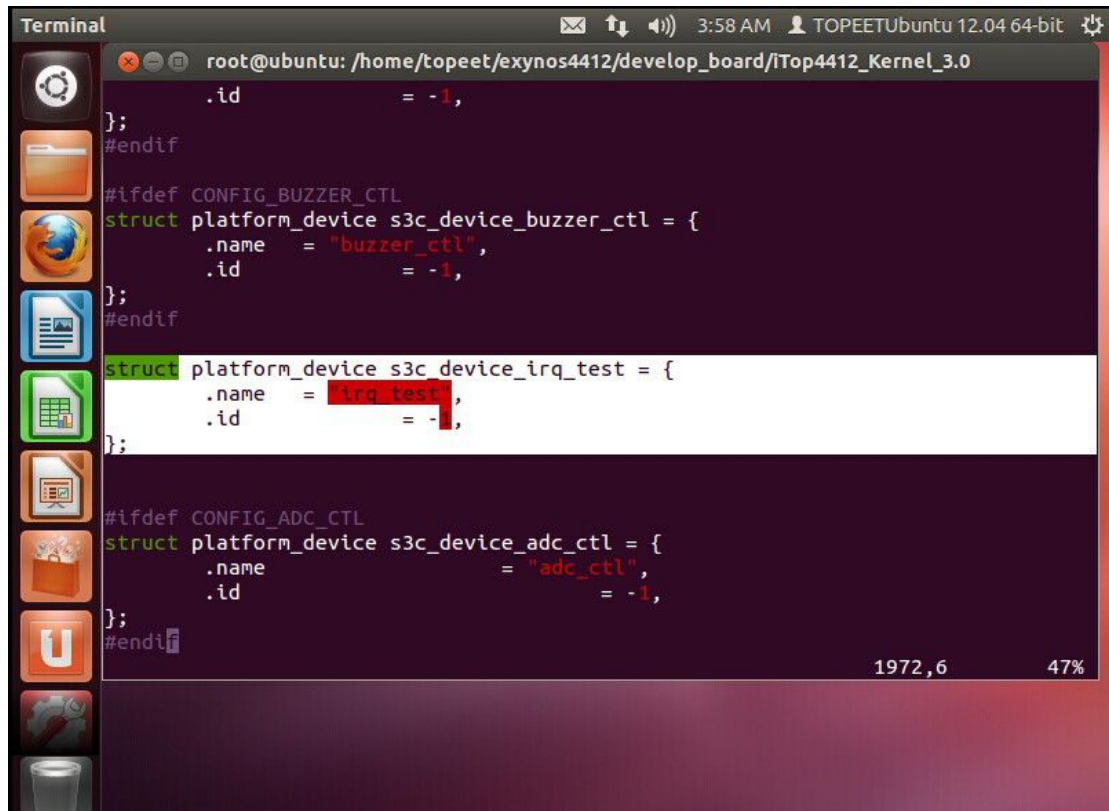
```
    .name  = "irq_test",
```

```
    .id    = -1,
```

```
};
```

如下图所示：





```
Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0

.id = -1,
};
#endif

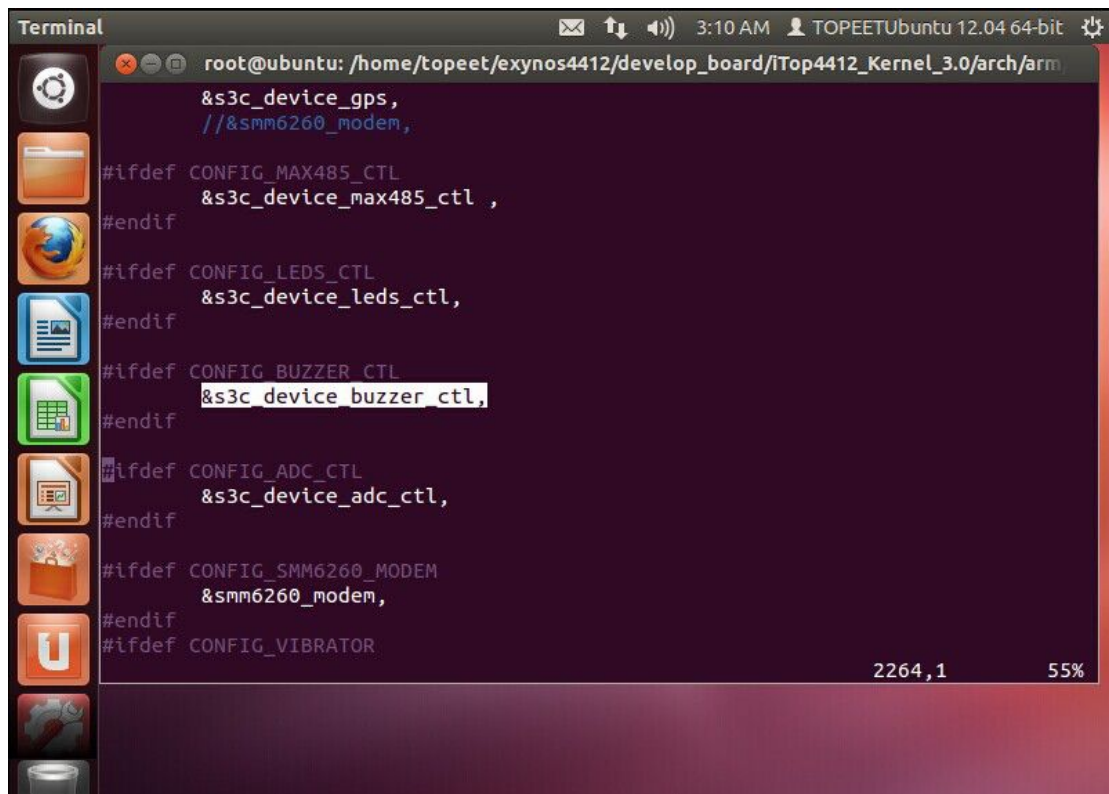
#ifdef CONFIG_BUZZER_CTL
struct platform_device s3c_device_buzzer_ctl = {
    .name = "buzzer_ctl",
    .id = -1,
};
#endif

struct platform_device s3c_device_irq_test = {
    .name = "irq_test",
    .id = -1,
};

#ifdef CONFIG_ADC_CTL
struct platform_device s3c_device_adc_ctl = {
    .name = "adc_ctl",
    .id = -1,
};
#endif

1972,6 47%
```

然后找到 “&s3c\_device\_buzzer\_ctl,” 这一行，如下图所示：



```
Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/arch/arm

&s3c_device_gps,
//&smm6260_modem,

#ifdef CONFIG_MAX485_CTL
&s3c_device_max485_ctl ,
#endif

#ifdef CONFIG_LEDS_CTL
&s3c_device_leds_ctl,
#endif

#ifdef CONFIG_BUZZER_CTL
&s3c_device_buzzer_ctl,
#endif

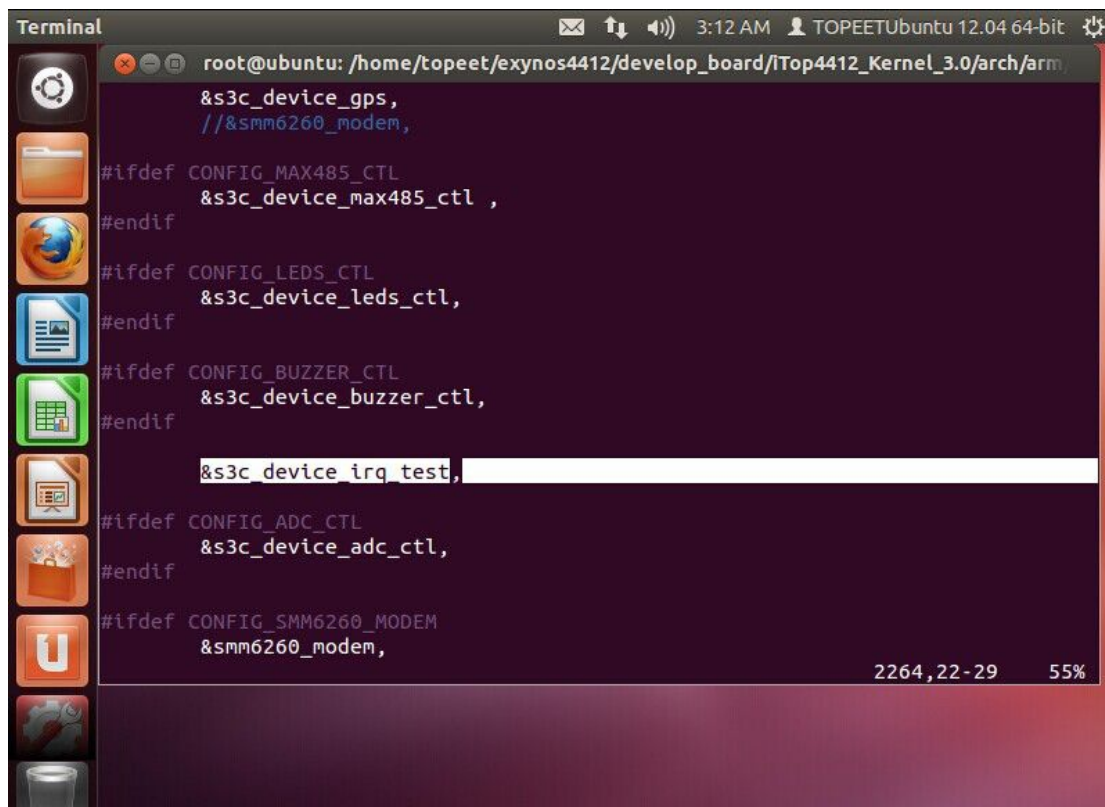
#ifdef CONFIG_ADC_CTL
&s3c_device_adc_ctl,
#endif

#ifdef CONFIG_SMM6260_MODEM
&smm6260_modem,
#endif

#ifdef CONFIG_VIBRATOR

2264,1 55%
```

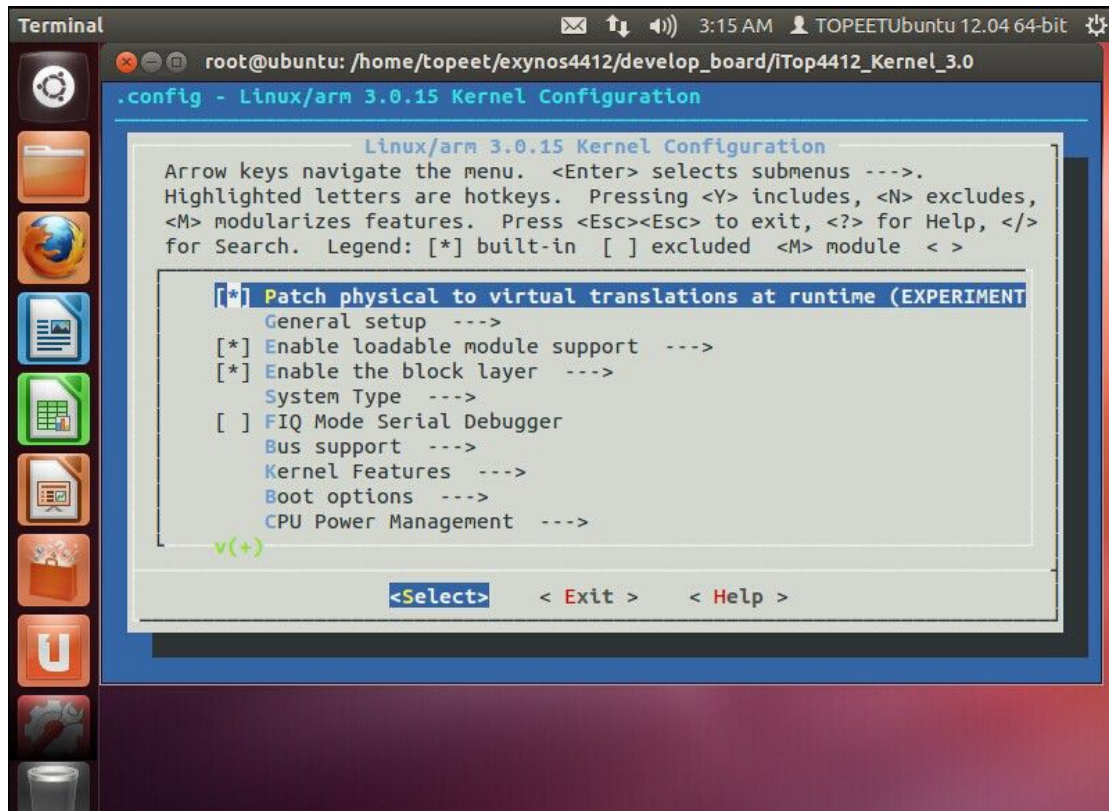
在这一行的下面添加 “&s3c\_device\_irq\_test,”，如下图所示：



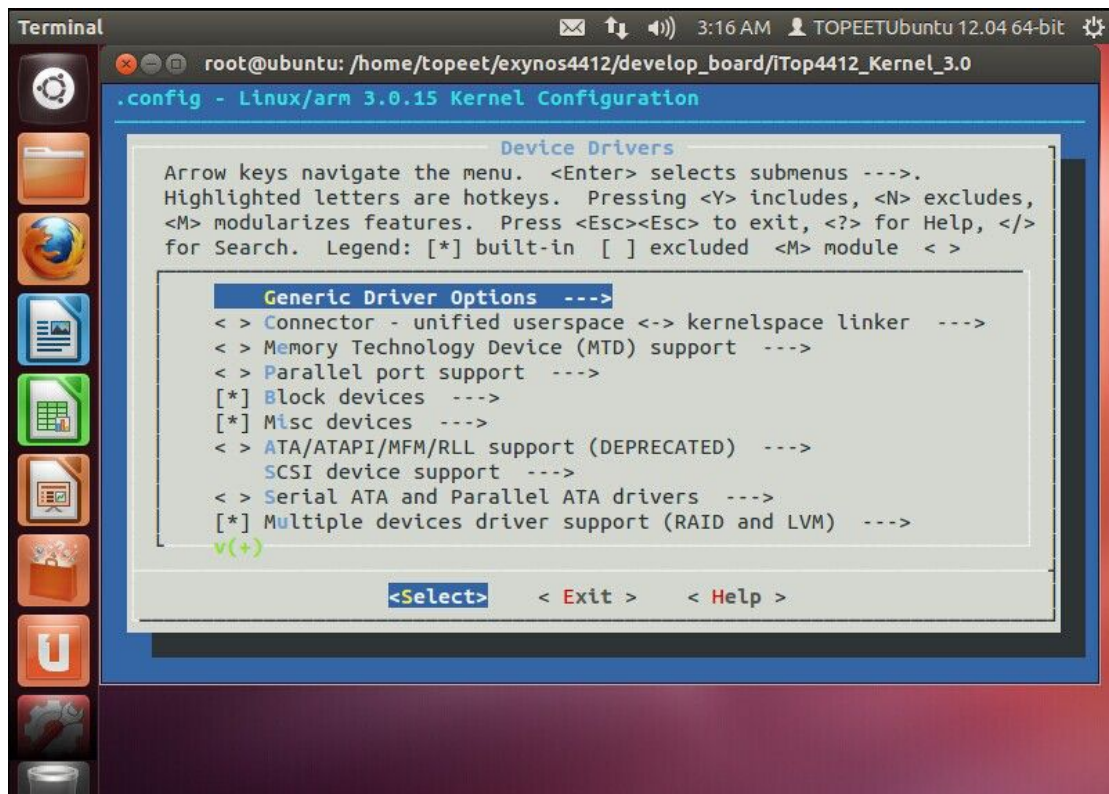
```
Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0/arch/arm
&s3c_device_gps,
//&smm6260_modem,
#ifdef CONFIG_MAX485_CTL
&s3c_device_max485_ctl ,
#endif
#ifdef CONFIG_LEDS_CTL
&s3c_device_leds_ctl,
#endif
#ifdef CONFIG_BUZZER_CTL
&s3c_device_buzzer_ctl,
#endif
&s3c_device_irq_test,
#ifdef CONFIG_ADC_CTL
&s3c_device_adc_ctl,
#endif
#ifdef CONFIG_SMM6260_MODEM
&smm6260_modem,
2264,22-29 55%
```

然后保存并退出。因为本章实验使用到了 led 和按键，所以我们要把内核里面的 led 驱动和按键的驱动去掉，

在内核目录下使用 make menuconfig 命令打开内核配置界面，如下图所示：

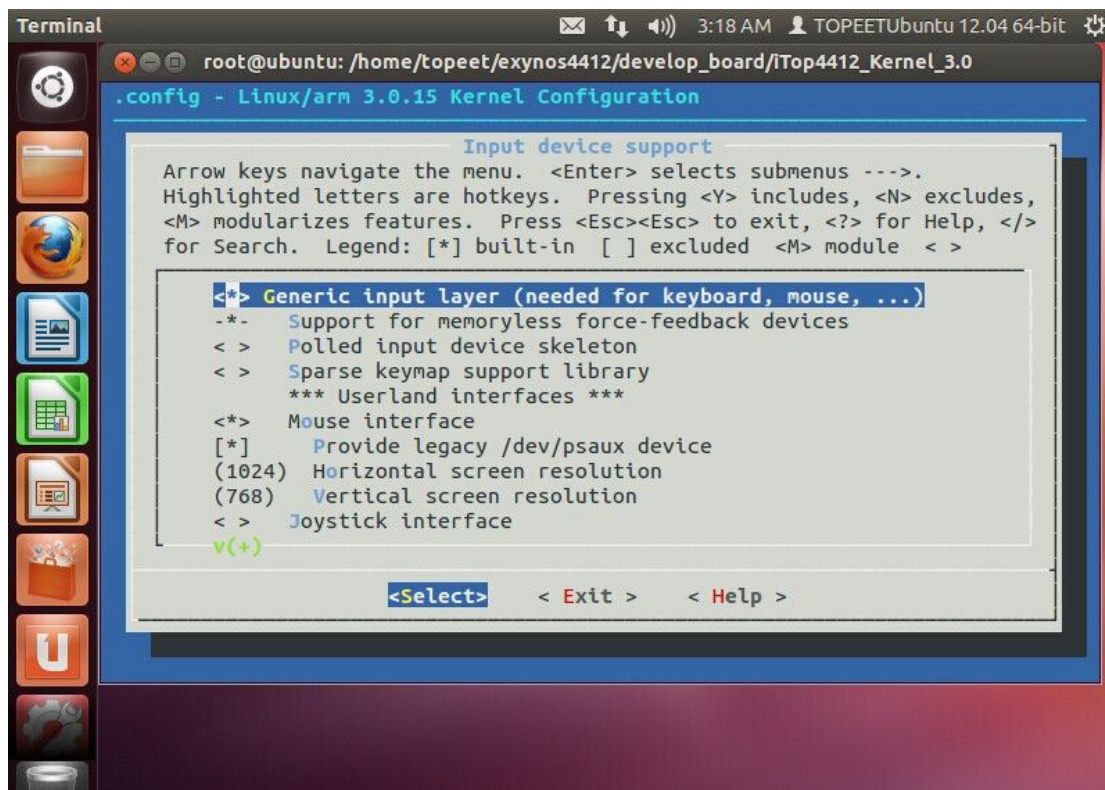


进入到“Device Drivers”，如下图所示：

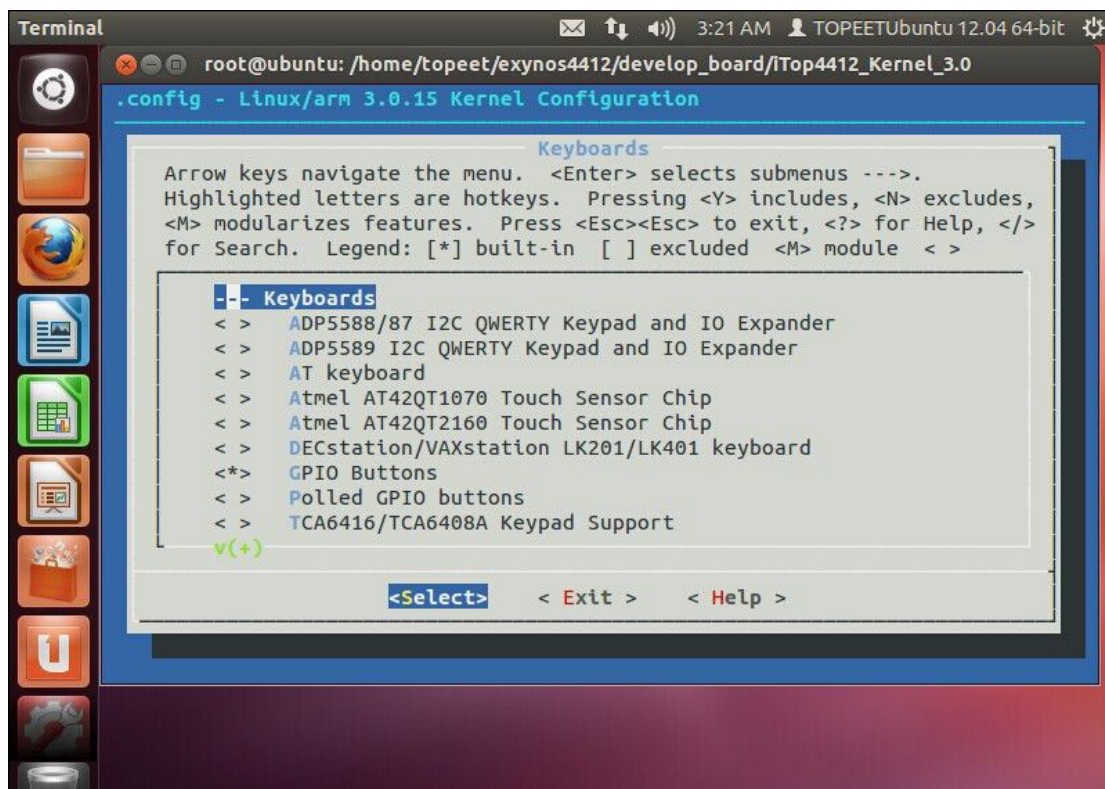




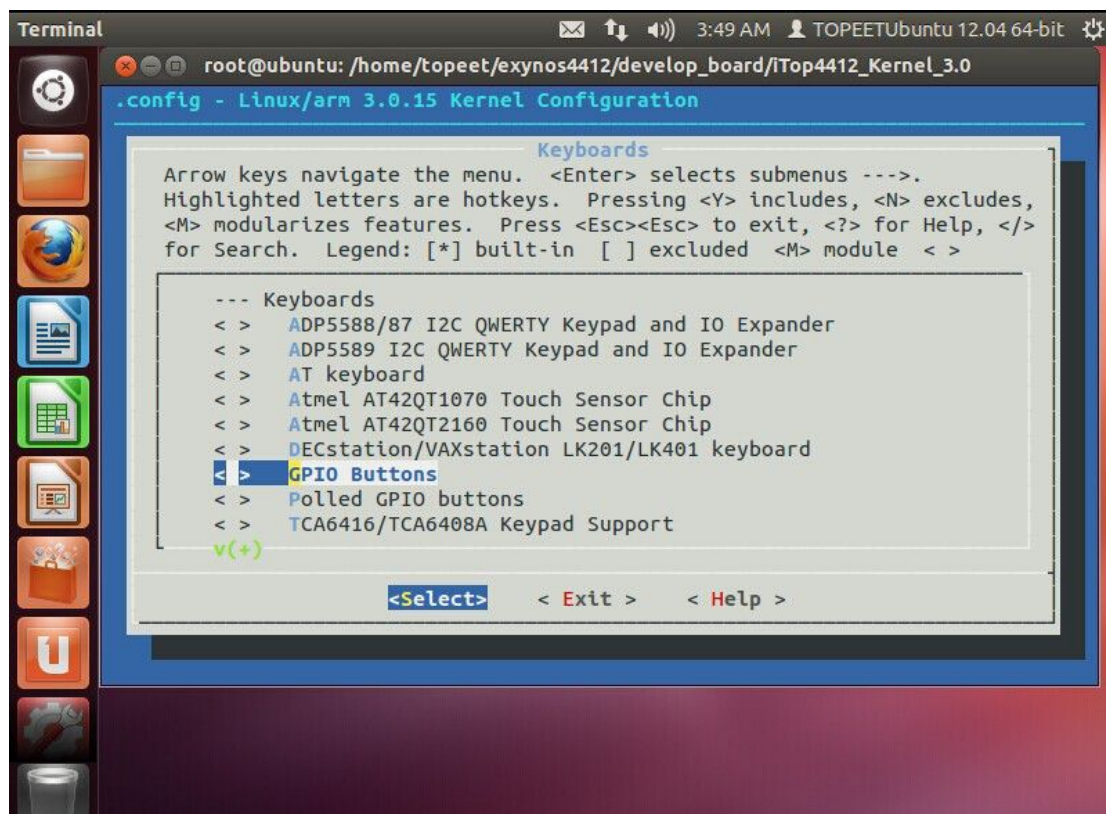
然后进入到 “Input device support” 界面，如下图所示：



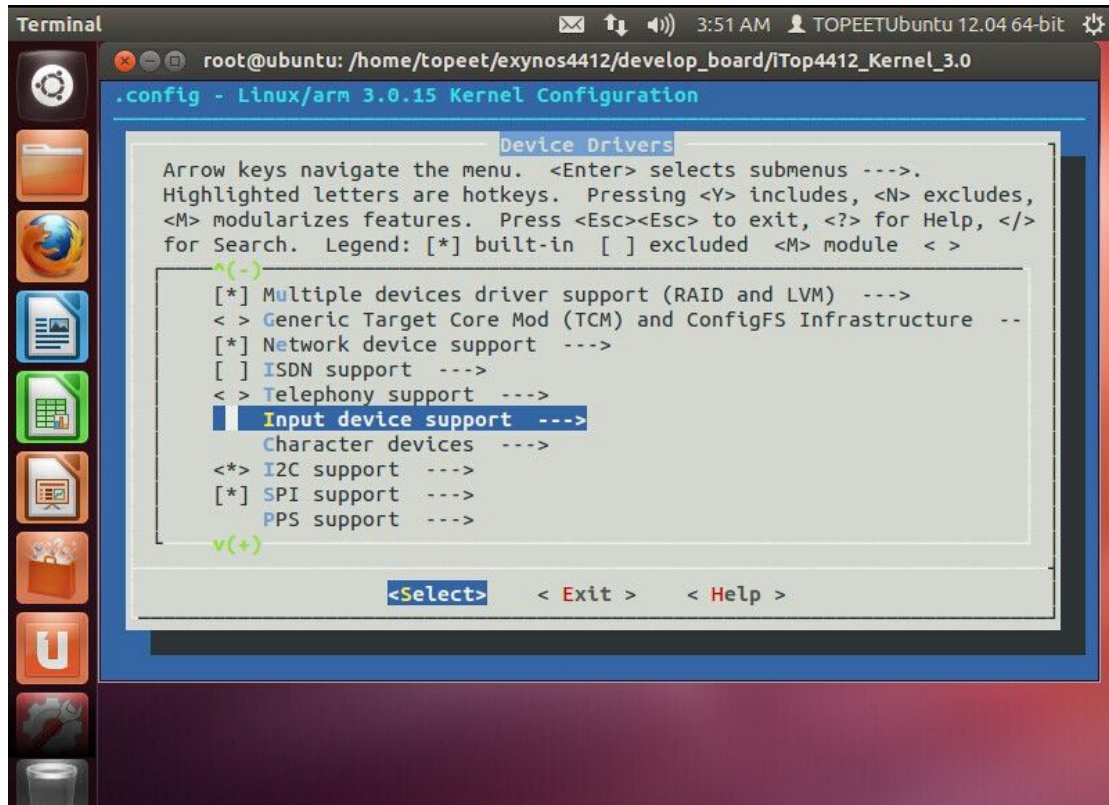
然后进入到 “Keyboards” 界面，如下图所示：



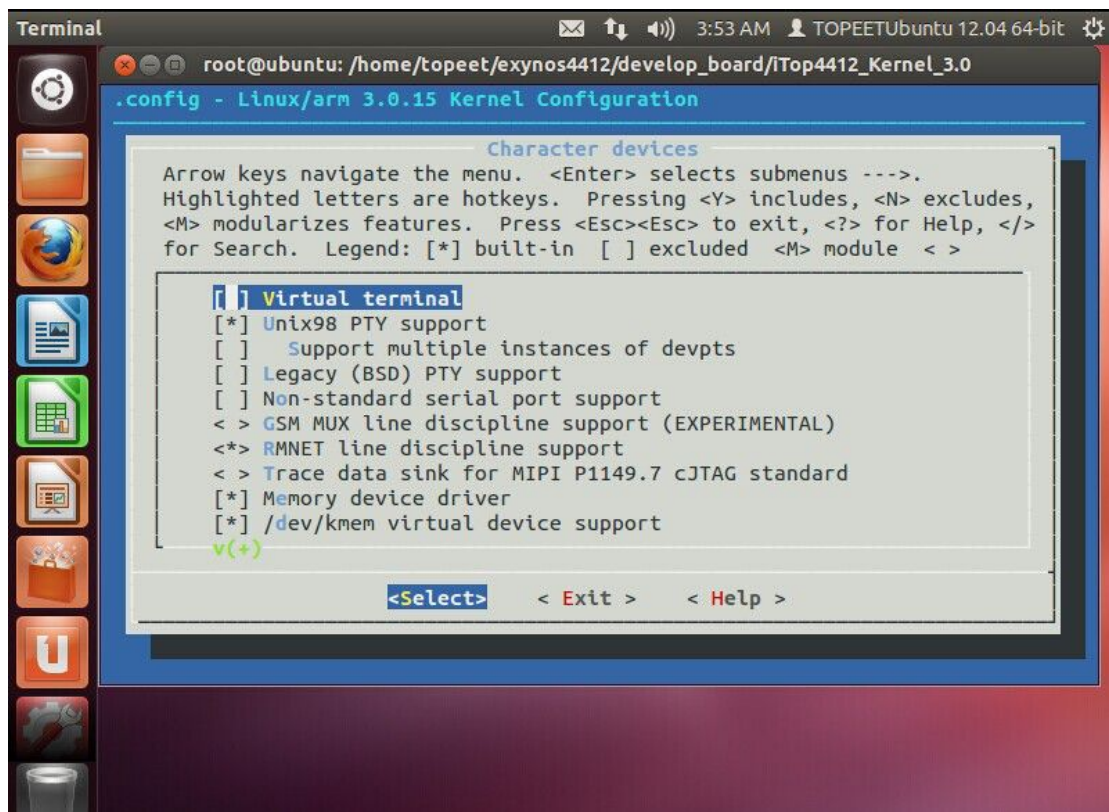
然后取消掉 “GPIO Buttons”，如下图所示：



然后返回到 “Device Drivers” 界面，如下图所示：

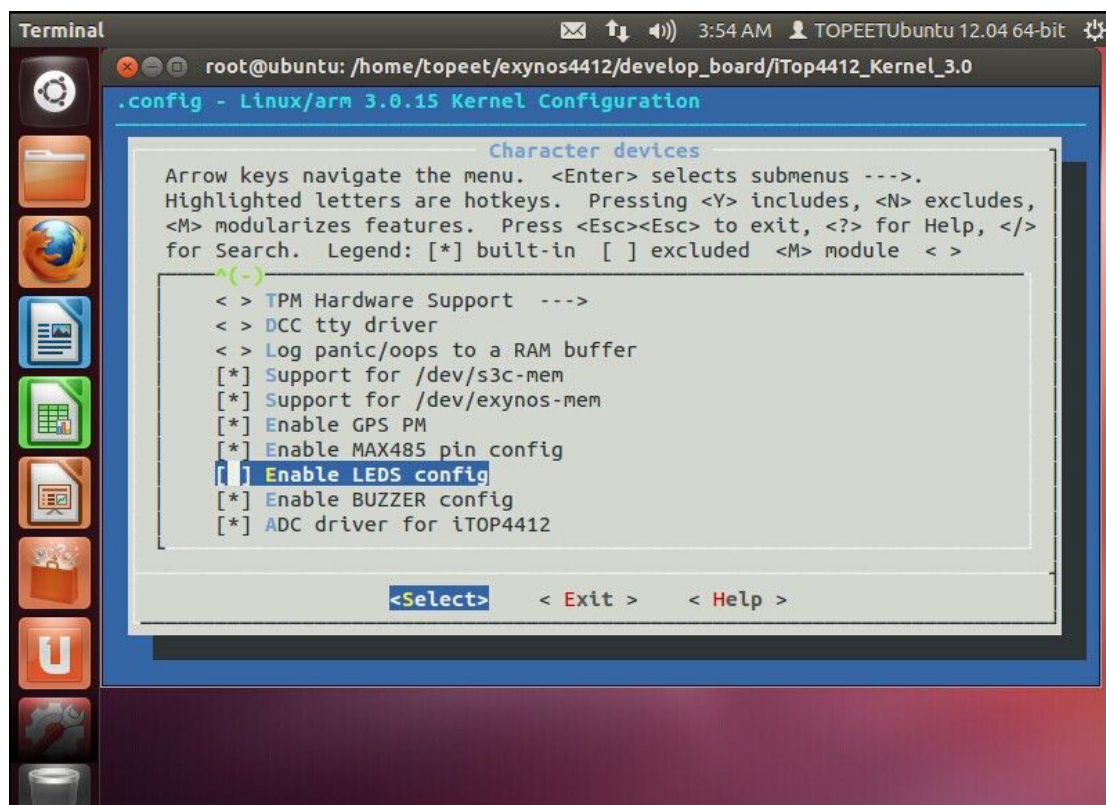


然后选择 “Character devices”，进入 Character devices 界面，如下图所示：

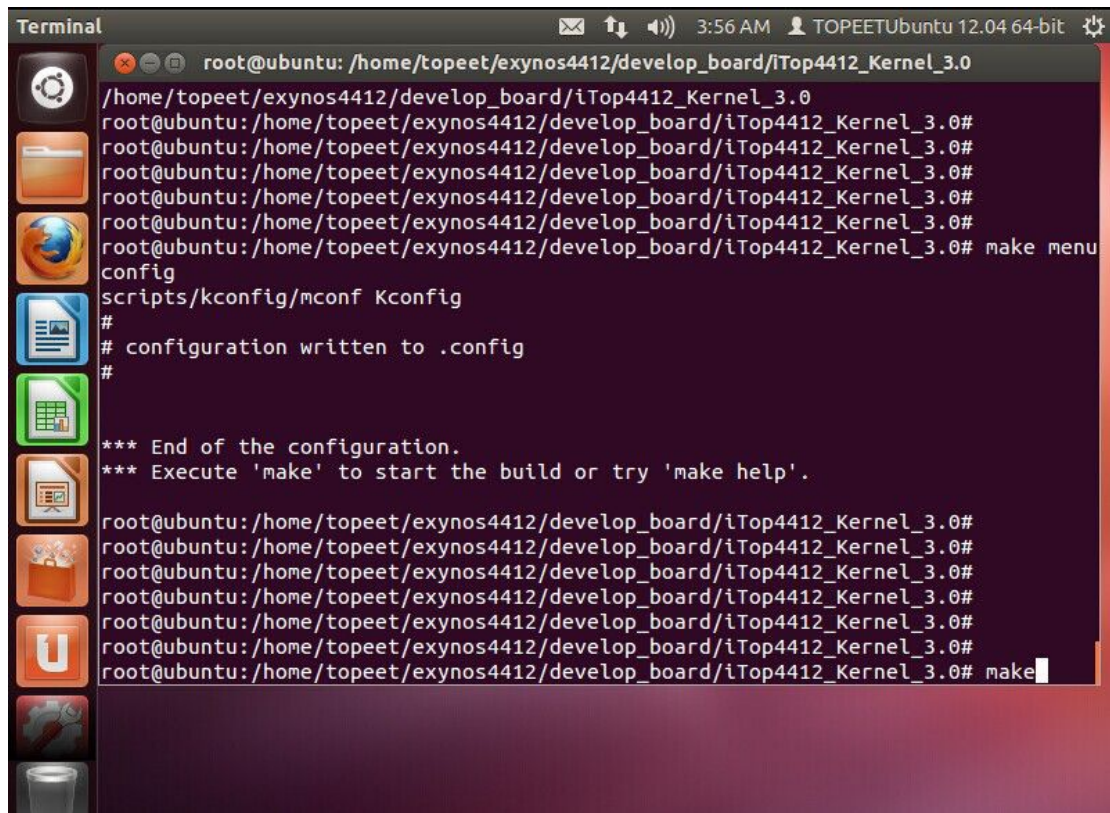




然后取消 “Enable LEDS config”，如下图所示：

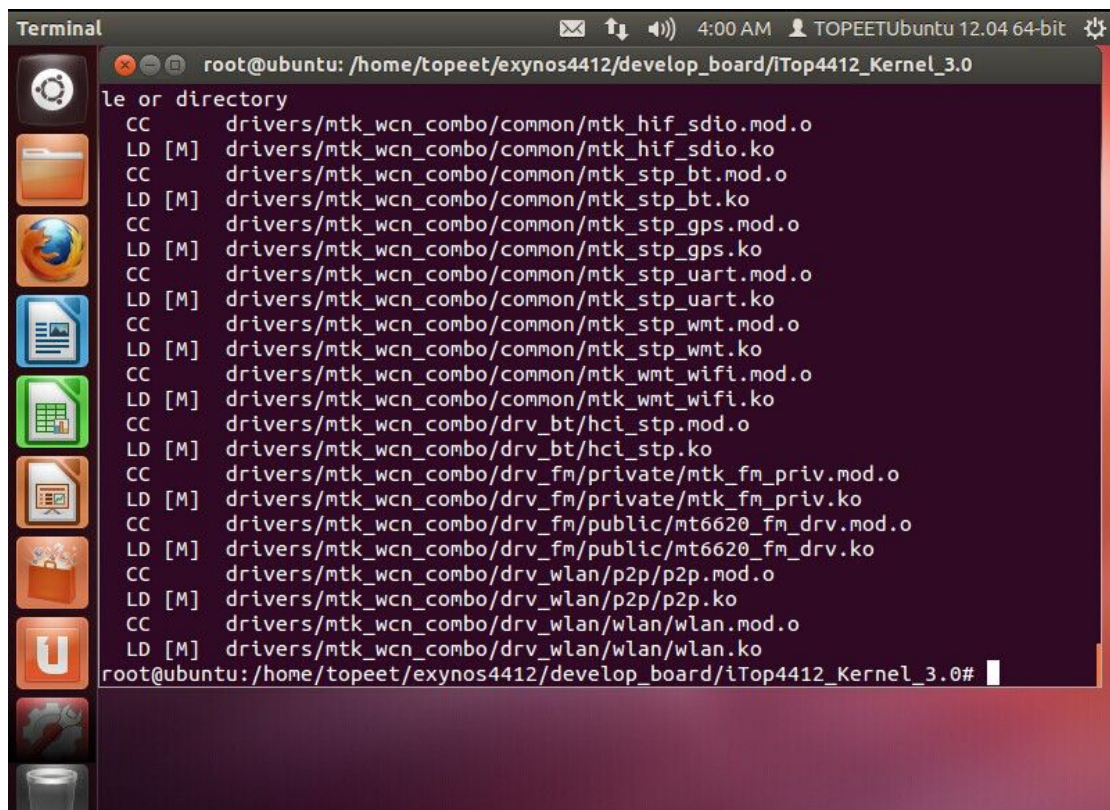


然后保存并退出内核配置界面，使用 make 命令编译内核，如下图所示：



```
Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0
/home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0# make menuconfig
scripts/kconfig/mconf Kconfig
#
# configuration written to .config
#
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0# make
```

编译完成后，如下图所示：



```
Terminal
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0
le or directory
CC      drivers/mtk_wcn_combo/common/mtk_hif_sdio.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_hif_sdio.ko
CC      drivers/mtk_wcn_combo/common/mtk_stp_bt.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_stp_bt.ko
CC      drivers/mtk_wcn_combo/common/mtk_stp_gps.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_stp_gps.ko
CC      drivers/mtk_wcn_combo/common/mtk_stp_uart.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_stp_uart.ko
CC      drivers/mtk_wcn_combo/common/mtk_stp_wmt.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_stp_wmt.ko
CC      drivers/mtk_wcn_combo/common/mtk_wmt_wifi.mod.o
LD [M]  drivers/mtk_wcn_combo/common/mtk_wmt_wifi.ko
CC      drivers/mtk_wcn_combo/drv_bt/hci_stp.mod.o
LD [M]  drivers/mtk_wcn_combo/drv_bt/hci_stp.ko
CC      drivers/mtk_wcn_combo/drv_fm/private/mtk_fm_priv.mod.o
LD [M]  drivers/mtk_wcn_combo/drv_fm/private/mtk_fm_priv.ko
CC      drivers/mtk_wcn_combo/drv_fm/public/mt6620_fm_drv.mod.o
LD [M]  drivers/mtk_wcn_combo/drv_fm/public/mt6620_fm_drv.ko
CC      drivers/mtk_wcn_combo/drv_wlan/p2p/p2p.mod.o
LD [M]  drivers/mtk_wcn_combo/drv_wlan/p2p/p2p.ko
CC      drivers/mtk_wcn_combo/drv_wlan/wlan/wlan.mod.o
LD [M]  drivers/mtk_wcn_combo/drv_wlan/wlan/wlan.ko
root@ubuntu: /home/topeet/exynos4412/develop_board/iTop4412_Kernel_3.0#
```



然后把编译生成的 zImage 烧写到 iTOP-4412 开发板上，烧写完成后启动开发板，系统起来以后，我们可以按开发板上的 BACK 或 HOME 按键，来看下运行结果。

当我们按下 BACK 按键时，可以看到 led 会点亮，在次按下，led 就会熄灭，同事串口会打印信息，如下图所示：

```
[root@iTOP-4412]#  
[root@iTOP-4412]#  
[root@iTOP-4412]# [ 98.575870] eint10_interrupt(50)  
[ 99.949779] eint10_interrupt(50)
```

我们按下 HOME 键，可以看到另外一个 led 会点亮，再次按下，led 就会熄灭，同事串口会打印信息，如下图所示：

```
[root@iTOP-4412]#  
[root@iTOP-4412]#  
[root@iTOP-4412]# [ 192.934924] eint9_interrupt(38)  
[ 195.210914] eint9_interrupt(38)
```

至此，linux 下中断驱动我们就已经完成了。