

## 7.4 Simple example of Linux drivers

In the previous section, we introduce a simple Hello module driver, it is just some information from the serial port output, the board did not correspond to the hardware to operate, and the following few examples are closely related to the actual hardware driver in embedded Linux systems, most of the hardware needed to operate a similar driver, such as touch screen, network card, audio, and etc., where we introduce some simple typical example, the driver actually has a in this section, we introduce driver is does not like the content of theoretical concepts, the Internet books that have a more systematic description.

### 7.4.1 LED driver

Source code description:

Driver source code directory	/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char
Driver Name	mini2440_leds.c
Equipment No.	LED is misc device, the device generated automatically
Equipment No.	/dev/leds
Test program source code directory	/opt/FriendlyARM/mini2440/examples/leds
Test program source code name	led.c
Test program executable file name	led
Note: LED driver has been compiled into the default kernel, so can't be loaded using insmod.	

Write the actual driver, you must understand the associated hardware resources, such as the use of registers, physical address, interrupt, etc., where LED is a very simple example, uses the following hardware resources.

The development board used in the hardware resources to 4 LED

LED	Name of the corresponding IO register	Corresponding CPU pin
LED1	GPB5	K2
LED2	GPB6	K5
LED3	GPB7	K7
LED4	GPB8	K8

Used to operate the IO port, it is necessary to set the registers they use, we need to call some of the existing function or macro, for example: [s3c2410\\_gpio\\_cfgpin](#).

Why S3C2410? Because the Samsung S3C2440 chips produced the register names used in the majority and the S3C2410, and resource allocation are the same, all in the current version of the Linux system, most are using the same function definition and macro definitions.

```
void s3c2410_gpio_cfgpin(unsigned int pin, unsigned int function)
{
    void __iomem *base = S3C24XX_GPIO_BASE(pin);
    unsigned long mask;
    unsigned long con;
    unsigned long flags;

    if (pin < S3C2410_GPIO_BANKB) {
        mask = 1 << S3C2410_GPIO_OFFSET(pin);
    }
    else {
        mask = 3 << S3C2410_GPIO_OFFSET(pin)*2;
    }

    switch (function) {
        case S3C2410_GPIO_LEAVE:
            mask = 0;
            function = 0;
            break;
        case S3C2410_GPIO_INPUT:
        case S3C2410_GPIO_OUTPUT:
        case S3C2410_GPIO_SFN2:
        case S3C2410_GPIO_SFN3:
            if (pin < S3C2410_GPIO_BANKB) {
                function -= 1;
                function &= 1;
                function <= S3C2410_GPIO_OFFSET(pin);
            }
            else {
                function &= 3;
                function <= S3C2410_GPIO_OFFSET(pin)*2;
            }
    }
}
```

```
    }  
  
/* modify the specified register wwith IRQs off */  
  
    local_irq_save(flags);  
    con = __raw_readl(base + 0x00);  
    con &= ~mask;  
    con |= function;  
    __raw_writel(con, base + 0x00);  
    local_irq_restore(flags);  
}
```

In fact, we do not need to care about drivers as long as they can be used, unless you are using a CPU system platform is can't supported by Linux, because most common embedded platforms have very similar definition, you do not need to write their own.

In the following driver list, you can see s3c2410\_gpio\_cfgpin called the situation. In addition, you also need to call a number of and device drivers are closely related to basic functions such as registration device misc\_register, fill out the driver function structure file\_operations, as well as Hello module in module\_init and module\_exit functions as such.

Some functions are not necessary, as you learn more about Linux driver development and reading more code, you naturally understand. Here is our list of LED driver code written,

Source code:

```
#include <linux/miscdevice.h>  
#include <linux/delay.h>  
#include <asm/irq.h>  
#include <mach/regs-gpio.h>  
#include <mach/hardware.h>  
#include <linux/kernel.h>  
#include <linux/module.h>  
#include <linux/init.h>  
#include <linux/mm.h>  
#include <linux/fs.h>  
#include <linux/types.h>  
#include <linux/delay.h>  
#include <linux/moduleparam.h>  
#include <linux/slab.h>  
#include <linux/errno.h>  
#include <linux/ioctl.h>  
#include <linux/cdev.h>  
#include <linux/string.h>  
#include <linux/list.h>  
#include <linux/pci.h>  
#include <asm/uaccess.h>  
#include <asm/atomic.h>
```

```

#include <asm/unistd.h>
#define DEVICE_NAME "leds"
static unsigned long led_table [] = {
    S3C2410_GPB5,
    S3C2410_GPB6,
    S3C2410_GPB7,
    S3C2410_GPB8,
};
static unsigned int led_cfg_table [] = {
    S3C2410_GPB5_OUTP,
    S3C2410_GPB6_OUTP,
    S3C2410_GPB7_OUTP,
    S3C2410_GPB8_OUTP,
};
static int sbc2440_leds_ioctl(
    struct inode *inode,
    struct file *file,
    unsigned int cmd,
    unsigned long arg    )
{
    switch(cmd) {
        case 0:
        case 1:
            if (arg > 4) {
                return -EINVAL;
            }
            s3c2410_gpio_setpin(led_table[arg], !cmd);
            return 0;
        default:
            return -EINVAL;
    }
}
static struct file_operations dev_fops = {
    .owner = THIS_MODULE,
    .ioctl = sbc2440_leds_ioctl,
};
static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};
static int __init dev_init(void)
{
    int ret;
    int i;

    for (i = 0; i < 4; i++) {
        s3c2410_gpio_cfgpin(led_table[i], led_cfg_table[i]);
        s3c2410_gpio_setpin(led_table[i], 0);
    }
    ret = misc_register(&misc);
    printk (DEVICE_NAME"\tinitialized\n");
    return ret;
}
static void __exit dev_exit(void)

```

```
{
    misc_deregister(&misc);
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

### 7.4.2 Button driver

Source code description:

Driver source code directory	/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char
Driver Name	mini2440_buttons.c
The driver major number	Misc equipment, device number will be generated automatically
Device Name	/dev/buttons
Test program source code directory	/opt/FriendlyARM/mini2440/examples/buttons
Test program source code name	buttons_test.c
Test program executable file name	buttons
Description: The key driver has been compiled into the default kernel, so cannot be loaded using insmod.	

Development board buttons are resources to used are as follows:

Button	Name of the corresponding IO register	The corresponding interrupt
K1	GPG0	EINT8
K2	GPG3	EINT11
K3	GPG5	EINT13
K4	GPG6	EINT14
K5	GPG7	EINT15
K6	GPG11	EINT19

Button driver source code:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <asm/irq.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>
#define DEVICE_NAME "buttons"
struct button_irq_desc {
    int irq;
    int pin;
    int pin_setting;
    int number;
    char *name;
};
static struct button_irq_desc button_irqs [] = {
    {IRQ_EINT8 , S3C2410_GPG0 , S3C2410_GPG0_EINT8 , 0, "KEY0"},
    {IRQ_EINT11, S3C2410_GPG3 , S3C2410_GPG3_EINT11 , 1, "KEY1"},
    {IRQ_EINT13, S3C2410_GPG5 , S3C2410_GPG5_EINT13 , 2, "KEY2"},
    {IRQ_EINT15, S3C2410_GPG7 , S3C2410_GPG7_EINT15 , 3, "KEY3"},
    {IRQ_EINT14, S3C2410_GPG6 , S3C2410_GPG6_EINT14 , 4, "KEY4"},
    {IRQ_EINT19, S3C2410_GPG11, S3C2410_GPG11_EINT19, 5, "KEY5"},
};
static volatile char key_values [] = {'0', '0', '0', '0', '0', '0'};
static DECLARE_WAIT_QUEUE_HEAD(button_waitq);
static volatile int ev_press = 0;
static irqreturn_t buttons_interrupt(int irq, void *dev_id)
{
```

```

    struct button_irq_desc *button_irqs = (struct button_irq_desc
*)dev_id;
    int down;
// udelay(0);
    down = !s3c2410_gpio_getpin(button_irqs->pin);
    if (down != (key_values[button_irqs->number] & 1)) { // Changed
        key_values[button_irqs->number] = '0' + down;
        ev_press = 1;
        wake_up_interruptible(&button_waitq);
    }
    return IRQ_RETVAL(IRQ_HANDLED);
}
static int s3c24xx_buttons_open(struct inode *inode, struct file
*file)
{
    int i;
    int err;

    for (i=0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++)
    {
        err = request_irq(button_irqs[i].irq, buttons_interrupt,
IRQ_TYPE_EDGE_BOTH,
        button_irqs[i].name, (void *)&button_irqs[i]);

        if (err)
            break;
    }
    if (err) {
        i--;
        for (; i >= 0; i--) {
            disable_irq(button_irqs[i].irq);
            free_irq(button_irqs[i].irq, (void
*)&button_irqs[i]);
        }
        return -EBUSY;
    }

    ev_press = 1;
    return 0;
}
static int s3c24xx_buttons_close(struct inode *inode, struct file
*file)
{
    int i;
    for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]);
i++) {
        free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
    }
    return 0;
}
static int s3c24xx_buttons_read(struct file *filp, char __user
*buff, size_t count, loff_t *offp)
{
    unsigned long err;
    if (!ev_press) {

```

```

        if (filp->f_flags & O_NONBLOCK)
            return -EAGAIN;
        else
            wait_event_interruptible(button_waitq, ev_press);
    }
    ev_press = 0;
    err = copy_to_user(buff, (const void *)key_values,
min(sizeof(key_values), count));
    return err ? -EFAULT : min(sizeof(key_values), count);
}
static unsigned int s3c24xx_buttons_poll( struct file *file, struct
poll_table_struct *wait)
{
    unsigned int mask = 0;
    poll_wait(file, &button_waitq, wait);
    if (ev_press)
        mask |= POLLIN | POLLRDNORM;
    return mask;
}
static struct file_operations dev_fops = {
    .owner = THIS_MODULE,
    .open = s3c24xx_buttons_open,
    .release = s3c24xx_buttons_close,
    .read = s3c24xx_buttons_read,
    .poll = s3c24xx_buttons_poll,
};
static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};
static int __init dev_init(void)
{
    int ret;
    ret = misc_register(&misc);
    printk (DEVICE_NAME"\tinitialized\n");
    return ret;
}
static void __exit dev_exit(void)
{
    misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");

```