字符设备驱动之中断按键

`driver`

说明

由于所用灯板上按键没有添加上拉电阻, 所以使用芯片内部有上拉功能的引脚.

灯板引脚

Key	Pin	GPIO	No.	set pullup
1	P7	B18	50	
2	P8	C27	91	PAD_PULLCTL1:1 = 1
3	P10	C26	90	PAD_PULLCTL1:2 = 1
4	P11	C0	64	
5	P13	C1	65	PAD_PULLCTL0:26 = 1
6	P15	C4	68	PAD_PULLCTL1:31 = 1
7	P16	A25	25	PAD_PULLCTL0:(13:12) = 01
8	P18	C6	70	
9	P19	C25	89	
10	P21	C24	88	
11	P22	C5	69	PAD_PULLCTL1:30 = 1
12	P23	C22	86	
13	P24	C23	87	
14	P26	B19	51	

Key	Pin	GPIO	No.	set pullup
15	P29	B15	47	

定义按键

```
static struct pin_desc{
    unsigned int pin;
    unsigned int key_val;
    char *name;
};

static struct pin_desc pins_desc[] = {
//  {OWL_GPIO_PORTB(18), 0x1,  "SW1"},
    {OWL_GPIO_PORTC(27), 0x2,  "SW2"},
    {OWL_GPIO_PORTC(26), 0x3,  "SW3"},
};
```

通过寄存器设置上拉

因为灯板上按键没有上拉电阻, 所以在这里通过寄存器设置相应的GPIO引脚为上拉模式. 根据上面表格,设置引脚GPIO C27, C26, C1, C4, A25, C5(SW2, SW3, SW5, SW6, SW7, SW11)为上拉:

```
    act_writel(act_readl(PAD_PULLCTL0) | (1 << 26) | (1 << 12) & 0xffffdfff, PAD_PULLCTL0);
    act_writel(act_readl(PAD_PULLCTL1) | (3 << 30) | 6 , PAD_PULLCTL1);
```

申请与释放gpio

在设置gpio引脚为上拉状态后, 申请相应的gpio并设置为输入模式:

```
gpio_request(pins_desc[i].pin, pins_desc[i].name);
gpio_direction_input(pins_desc[i].pin);
```

获取中断号

在申请与释放中断时会用到中断号,如:

```
request_irq(gpio_to_irq(pins_desc[i].pin), buttons_irq,
    IRQF_TRIGGER_FALLING, pins_desc[i].name, &pins_desc[i]);
```

这里使用了`gpio_to_irq(unsigned gpio)`函数, 将GPIO映射为IRQ中断, 参数为相应的引脚, 如**OWL_GPIO_PORTC(26)**.

源代码

驱动源码

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/interrupt.h>
#include <linux/input/mt.h>
#include <linux/sched.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <mach/gpio.h>
#include <mach/hardware.h>
#include <asm/uaccess.h>

static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

static struct class *key_led_class;
static struct device *key_led_device;

static struct pin_desc{
    unsigned int pin;
    unsigned int key_val;
    char *name;
};

static struct pin_desc pins_desc[] = {
//  {OWL_GPIO_PORTB(18), 0x1,  "SW1"},
    {OWL_GPIO_PORTC(27), 0x2,  "SW2"},
```

```
26.      {OWL_GPIO_PORTC(26), 0x3,  "SW3"},
27.  //  {OWL_GPIO_PORTC(0),  0x4,  "SW4"},
28.      {OWL_GPIO_PORTC(1),  0x5,  "SW5"},
29.      {OWL_GPIO_PORTC(4),  0x6,  "SW6"},
30.      {OWL_GPIO_PORTA(25), 0x7,  "SW7"},
31.  //  {OWL_GPIO_PORTC(6),  0x8,  "SW8"},
32.  //  {OWL_GPIO_PORTC(25), 0x9,  "SW9"},
33.  //  {OWL_GPIO_PORTC(24), 0x10, "SW10"},
34.      {OWL_GPIO_PORTC(5),  0x11, "SW11"},
35.  //  {OWL_GPIO_PORTC(22), 0x12, "SW12"},
36.  //  {OWL_GPIO_PORTC(23), 0x13, "SW13"},
37.  //  {OWL_GPIO_PORTB(19), 0x14, "SW14"},
38.  //  {OWL_GPIO_PORTB(15), 0x15, "SW15"},
39.  };
40.  struct pin_desc *irq_pindes;
41.
42.  static int ev_press = 0;
43.  static unsigned int key_val;
44.  int major;
45.  static struct timer_list key_led_timer;
46.
47.  static irqreturn_t buttons_irq(int irq, void *dev_id)
48.  {
49.      int ret;
50.      irq_pindes = (struct pin_desc *)dev_id;
51.      ret = mod_timer(&key_led_timer, jiffies + (HZ / 100));
52.      if (ret == 1)
53.          printk("mod timer success \n");
54.      return IRQ_HANDLED;
55.  }
56.
57.  static int key_led_open(struct inode * inode, struct file * filp)
58.  {
59.      int i;
60.      for (i = 0; i < sizeof(pins_desc)/sizeof(pins_desc[0]); i++) {
61.          request_irq(gpio_to_irq(pins_desc[i].pin), buttons_irq,
62.              IRQF_TRIGGER_FALLING, pins_desc[i].name, &pins_desc[i]);
63.      }
64.      return 0;
65.  }
66.
67.  static ssize_t key_led_read(struct file *file, char __user *user,
     size_t size,loff_t *ppos)
68.  {
69.      if (size != 1)
```

```
70.             return -EINVAL;
71.         wait_event_interruptible(button_waitq, ev_press);
72.         copy_to_user(user, &key_val, 1);
73.         ev_press = 0;
74.         return 1;
75.     }
76.
77.     static int key_led_close(struct inode *inode, struct file *file)
78.     {
79.         int i;
80.         for (i = 0; i < sizeof(pins_desc)/sizeof(pins_desc[0]); i++) {
81.             free_irq(gpio_to_irq(pins_desc[i].pin),&pins_desc[i]);
82.         }
83.         return 0;
84.     }
85.
86.     static const struct file_operations key_led_fops = {
87.         .owner      = THIS_MODULE,
88.         .open       = key_led_open,
89.         .read       = key_led_read,
90.         .release    = key_led_close,
91.     };
92.
93.     static void key_led_timer_function(unsigned long data)
94.     {
95.         struct pin_desc *pindesc = irq_pindes;
96.         key_val = pindesc->key_val;
97.         ev_press = 1;
98.         wake_up_interruptible(&button_waitq);
99.     }
100.
101.    static int key_led_init(void)
102.    {
103.        int i;
104.
105.        act_writel(act_readl(PAD_PULLCTL0) | (1 << 26) | (1 << 12) & 0xffff
    dfff, PAD_PULLCTL0);
106.        act_writel(act_readl(PAD_PULLCTL1) | (3 << 30) | 6 , PAD_PULLCTL1);
107.
108.        init_timer(&key_led_timer);
109.        key_led_timer.function = key_led_timer_function;
110.        add_timer(&key_led_timer);
111.
112.        major = register_chrdev(0, "key_led", &key_led_fops);
113.        key_led_class = class_create(THIS_MODULE, "key_led");
```

```
    key_led_device = device_create(key_led_class, NULL, MKDEV(major, 0), NULL, "buttons");

    for (i = 0; i < sizeof(pins_desc)/sizeof(pins_desc[0]); i++) {
        gpio_request(pins_desc[i].pin, pins_desc[i].name);
        gpio_direction_input(pins_desc[i].pin);
    }

    return 0;
}

static void key_led_exit(void)
{
    int i;
    for (i = 0; i < sizeof(pins_desc)/sizeof(pins_desc[0]); i++) {
        gpio_free(pins_desc[i].pin);
    }
    del_timer(&key_led_timer);
    unregister_chrdev(major, "key_led");
    device_unregister(key_led_device);
    class_destroy(key_led_class);
}

module_init(key_led_init);
module_exit(key_led_exit);

MODULE_AUTHOR("Tab Liu");
MODULE_DESCRIPTION("Just for Demon");
MODULE_LICENSE("GPL");
```

测试代码

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

int main (int argc, char *argc[])
{
    int fd;
    unsigned char key_val;

```

```
12.     fd = open("/dev/buttons", O_RDWR);
13.     if (fd < 0) {
14.         printf("open error\n");
15.     }
16.
17.     while(1) {
18.         read(fd, &key_val, 1);
19.         printf("key_val = 0X%x\n" key_val);
20.     }
21.     return 0;
22. }
```