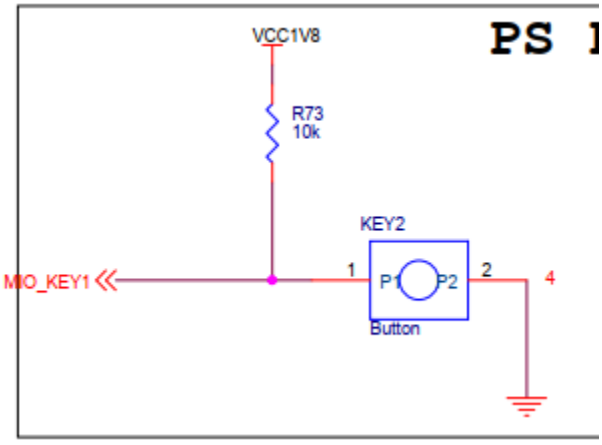
6 gpio 输入

本章我们做一个 gpio 输出的实验，巩固一下 pinctrl 和 gpio 子系统，同时也为后面的学习做铺垫。

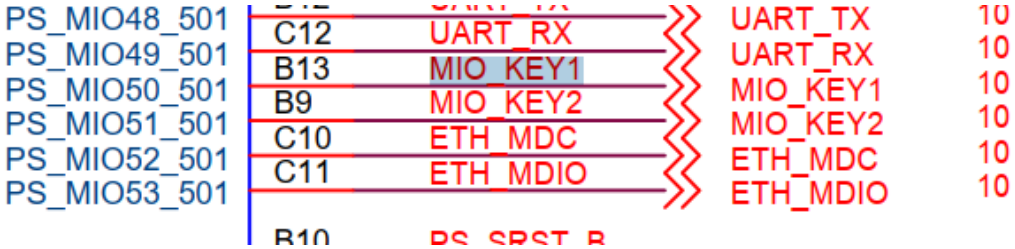
gpio 输出最经典的例子就是按键，我们就用按键来做个简单的 gpio 输入实验。实验的目标是应用程序通过 read 函数读取按键状态，如果按键被按下，就反转一次 led 的电平。用按键控制 led 的亮灭。

6.1 原理图

使用板子上的 ps_key1，查看原理图，对应到原理图上的 KEY2，key 的另一端是接地的。按下按键 MIO_KEY1 就会被拉低。



再找到 MIO_KEY1 连接的 IO，为 MIO50。



6.2 设备树

打开 system-user.dtsi 文件，在根节点中添加下面的节点：

```
amba {
    slcr@f8000000 {
        pinctrl_led_default: led-default {
            mux {
                groups = "gpio0_0_grp";
                function = "gpio0";
```

```
            };

            conf {
                pins = "MIO0";
                io-standard = <1>;
                bias-disable;
                slew-rate = <0>;
            };
        };
        pinctrl_key_default: key-default {
            mux {
                groups = "gpio0_50_grp";
                function = "gpio0";
            };

            conf {
                pins = "MIO50";
                io-standard = <1>;
                bias-high-impedance;
                slew-rate = <0>;
            };
        };
    };
};

alinxled {
    compatible = "alinx-led";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_led_default>;
    alinxled-gpios = <&gpio0 0 0>;
};

alinxkey {
    compatible = "alinx-key";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_key_default>;
    alinxkey-gpios = <&gpio0 50 0>;
};
```

其中 led 相关的部分和之前是一样的。

key 使用的 IO 是 MIO50，所以 groups = "gpio0_50_grp"、pins = "MIO50"、alinxkey-gpio = <&gpio0 50 0>。输入引脚电气特性配置成高阻 bias-high-impedance。

6.3 驱动代码

使用 petalinux 新建驱动名为”ax-key-dev”，在 ax-key-dev 中输入以下代码：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/ide.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/cdev.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/of_gpio.h>
#include <linux/device.h>
#include <linux/delay.h>
#include <linux/init.h>
#include <linux/gpio.h>
#include <linux/delay.h>
#include <asm/uaccess.h>
#include <asm/mach/map.h>
#include <asm/io.h>

/* 设备节点名称 */
#define DEVICE_NAME       "gpio_key"
/* 设备号个数 */
#define DEVID_COUNT       1
/* 驱动个数 */
#define DRIVE_COUNT       1
/* 主设备号 */
#define MAJOR1
/* 次设备号 */
#define MINOR1            0

/* 把驱动代码中会用到的数据打包进设备结构体 */
struct alinx_char_dev{
    dev_t               devid;          //设备号
    struct cdev         cdev;           //字符设备
    struct class        *class;         //类
    struct device       *device;        //设备
    struct device_node  *nd;            //设备树的设备节点
    int                 alinx_key_gpio; //gpio 号
};
```

```
/* 声明设备结构体 */
static struct alinx_char_dev alinx_char = {
    .cdev = {
        .owner = THIS_MODULE,
    },
};

/* open 函数实现，对应到 Linux 系统调用函数的 open 函数 */
static int gpio_key_open(struct inode *inode_p, struct file *file_p)
{
    /* 设置私有数据 */
    file_p->private_data = &alinx_char;
    printk("gpio_test module open\n");
    return 0;
}


/* write 函数实现，对应到 Linux 系统调用函数的 write 函数 */
static ssize_t gpio_key_read(struct file *file_p, char __user *buf, size_t len, loff_t *lof
f_t_p)
{
    int ret = 0;
    /* 返回按键的值 */
    unsigned int key_value = 0;
    /* 获取私有数据 */
    struct alinx_char_dev *dev = file_p->private_data;

    /* 检查按键是否被按下 */
    if(0 == gpio_get_value(dev->alinx_key_gpio))
    {
        /* 按键被按下 */
        /* 防抖 */
        mdelay(50);
        /* 等待按键抬起 */
        while(!gpio_get_value(dev->alinx_key_gpio));
        key_value = 1;
    }
    else
    {
        /* 按键未被按下 */
    }
    /* 返回按键状态 */
    ret = copy_to_user(buf, &key_value, sizeof(key_value));

```

```
84.     return ret;
85. }
86. 
87. /* release 函数实现，对应到 Linux 系统调用函数的 close 函数 */
88. static int gpio_key_release(struct inode *inode_p, struct file *file_p)
89. {
90.     printk("gpio_test module release\n");
91.     return 0;
92. }
93. 
94. /* file_operations 结构体声明，是上面 open、write 实现函数与系统调用函数对应的关键 */
95. static struct file_operations ax_char_fops = {
96.     .owner   = THIS_MODULE,
97.     .open    = gpio_key_open,
98.     .read    = gpio_key_read,
99.     .release = gpio_key_release,
100. };
101. 
102. /* 模块加载时会调用的函数 */
103. static int __init gpio_key_init(void)
104. {
105.     /* 用于接受返回值 */
106.     u32 ret = 0;
107. 
108.     /* 获取设备节点 */
109.     alinx_char.nd = of_find_node_by_path("/alinxkey");
110.     if(alinx_char.nd == NULL)
111.     {
112.         printk("alinx_char node not find\r\n");
113.         return -EINVAL;
114.     }
115.     else
116.     {
117.         printk("alinx_char node find\r\n");
118.     }
119. 
120.     /* 获取节点中 gpio 标号 */
121.     alinx_char.alinx_key_gpio = of_get_named_gpio(alinx_char.nd, "alinxkey-gpios", 0);
122.     if(alinx_char.alinx_key_gpio < 0)
123.     {
124.         printk("can not get alinxkey-gpios");
125.         return -EINVAL;
126.     }
127.     printk("alinxkey-gpio num = %d\r\n", alinx_char.alinx_key_gpio);
```

```
128.
129.    /* 申请 gpio 标号对应的引脚 */
130.    ret = gpio_request(alinx_char.alinx_key_gpio, "alinxkey");
131.    if(ret != 0)
132.    {
133.        printk("can not request gpio\r\n");
134.        return -EINVAL;
135.    }
136.
137.    /* 把这个 io 设置为输入 */
138.    ret = gpio_direction_input(alinx_char.alinx_key_gpio);
139.    if(ret < 0)
140.    {
141.        printk("can not set gpio\r\n");
142.        return -EINVAL;
143.    }
144.
145.    /* 注册设备号 */
146.    alloc_chrdev_region(&alinx_char.devid, MINOR1, DEVID_COUNT, DEVICE_NAME);
147.
148.    /* 初始化字符设备结构体 */
149.    cdev_init(&alinx_char.cdev, &ax_char_fops);
150.
151.    /* 注册字符设备 */
152.    cdev_add(&alinx_char.cdev, alinx_char.devid, DRIVE_COUNT);
153.
154.    /* 创建类 */
155.    alinx_char.class = class_create(THIS_MODULE, DEVICE_NAME);
156.    if(IS_ERR(alinx_char.class))
157.    {
158.        return PTR_ERR(alinx_char.class);
159.    }
160.
161.    /* 创建设备节点 */
162.    alinx_char.device = device_create(alinx_char.class, NULL,
163.                                      alinx_char.devid, NULL,
164.                                      DEVICE_NAME);
165.    if (IS_ERR(alinx_char.device))
166.    {
167.        return PTR_ERR(alinx_char.device);
168.    }
169.
170.    return 0;
171. }
```

```
172.
173. /* 卸载模块 */
174. static void __exit gpio_key_exit(void)
175. {
176.     /* 释放 gpio */
177.     gpio_free(alinx_char.alinx_key_gpio);
178.
179.     /* 注销字符设备 */
180.     cdev_del(&alinx_char.cdev);
181.
182.     /* 注销设备号 */
183.     unregister_chrdev_region(alinx_char.devid, DEVID_COUNT);
184.
185.     /* 删除设备节点 */
186.     device_destroy(alinx_char.class, alinx_char.devid);
187.
188.     /* 删除类 */
189.     class_destroy(alinx_char.class);
190.
191.     printk("gpio_key_dev_exit_ok\n");
192. }
193.
194. /* 标记加载、卸载函数 */
195. module_init(gpio_key_init);
196. module_exit(gpio_key_exit);
197.
198. /* 驱动描述信息 */
199. MODULE_AUTHOR("Alinx");
200. MODULE_ALIAS("gpio_key");
201. MODULE_DESCRIPTION("GPIO OUT driver");
202. MODULE_VERSION("v1.0");
203. MODULE_LICENSE("GPL");
```

和第 4 章的很相似，改动的部分已加粗。

138 原先是设置成输出，改成了输入。

59~82 行原先的 write 函数变成了 read。每次 read 就用 gpio_get_value 函数读取一下 IO 的电平，如果检测到低电平，则按键被按下，按键按下后则是常规的延时去抖，等待抬起。

read 函数最后要把读到的电平返回给用户 buf。

6.4 测试代码

新建 QT 工程名为”ax-key-test”，新建 main.c，输入下列代码：

```
1.  #include <stdio.h>
```

```c
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
    int fd, fd_l ,ret;
    char *filename, led_value = 0;
    unsigned int key_value;

    if(argc != 2)
    {
        printf("Error Usage\r\n");
        return -1;
    }

    filename = argv[1];
    fd = open(filename, O_RDWR);
    if(fd < 0)
    {
        printf("file %s open failed\r\n", argv[1]);
        return -1;
    }

    while(1)
    {
        ret = read(fd, &key_value, sizeof(key_value));
        if(ret < 0)
        {
            printf("read failed\r\n");
            break;
        }
        if(1 == key_value)
        {
            printf("ps_key1 press\r\n");
            led_value = !led_value;

            fd_l = open("/dev/gpio_leds", O_RDWR);
            if(fd_l < 0)
            {
                printf("file /dev/gpio_leds open failed\r\n");
                break;
            }

```

```
46.             ret = write(fd_l, &led_value, sizeof(led_value));
47.             if(ret < 0)
48.             {
49.                 printf("write failed\r\n");
50.                 break;
51.             }
52.
53.             ret = close(fd_l);
54.             if(ret < 0)
55.             {
56.                 printf("file /dev/gpio_leds close failed\r\n");
57.                 break;
58.             }
59.         }
60.     }
61.
62.     ret = close(fd);
63.     if(ret < 0)
64.     {
65.         printf("file %s close failed\r\n", argv[1]);
66.         return -1;
67.     }
68.
69.     return 0;
70. }
```

因为要点灯，所以还要用到前面 led 的设备节点，这边就直接用设备节点/dev/gpio_leds 了，在测试时记得加载 led 驱动就行了。

在 **26** 行的 while 循环里，不停的去调用 read 函数读取按键状态，一旦读取到按键被按下，就去对 led 的 io 进行写操作，每次写入相反的值以达到每一次按键 led 状态反转的效果。

6.5 运行测试

因为修改了设备树，所以要跟新 sd 卡中 image.ub 文件。因为同时用到了 led 和 key，所以要加载两个驱动，led 驱动用上面任意一章的都行。步骤如下：

```
mount -t nfs -o nolock 192.168.1.107:/home/alinx/work /mnt    //IP 和路径根据实际情况调整

cd /mnt

mkdir /tmp/qt

mount qt_lib.img /tmp/qt

cd /tmp/qt

source ./qt_env_set.sh
```

```
cd /mnt

./ax-concled-dev.ko

./ax-key-dev.ko

cd ./build-ax-key-test-ZYNQ-Debug/    //app 目录可能不一样，根据实际情况调整

./ax-key-test /dev/gpio_key
```

操作结果如下图：

```
root@ax_peta:~# mount -t nfs -o nolock 192.168.1.107:/home/alinx/work /mnt
root@ax_peta:~# cd /mnt
root@ax_peta:/mnt# mkdir /tmp/qt
root@ax_peta:/mnt# mount qt_lib.img /tmp/qt
random: fast init done
EXT4-fs (loop0): recovery complete
EXT4-fs (loop0): mounted filesystem with ordered data mode. Opts: (null)
root@ax_peta:/mnt# cd /tmp/qt
root@ax_peta:/tmp/qt# source ./qt_env_set.sh
/tmp/qt
root@ax_peta:/tmp/qt# cd /mnt
root@ax_peta:/mnt# insmod ./ax-concled-dev.ko
ax_concled_dev: loading out-of-tree module taints kernel.
alinx_char node find
alinxled-gpio num = 899
root@ax_peta:/mnt# insmod ./ax-key-
ax-key-dev.ko  ax-key-test/
root@ax_peta:/mnt# insmod ./ax-key-dev.ko
alinx_char node find
alinxkey-gpio num = 949
```

```
root@ax_peta:/mnt# cd ./build-ax-key-test-ZYNQ-Debug/
root@ax_peta:/mnt/build-ax-key-test-ZYNQ-Debug# ./ax-key-test /dev/gpio_key
gpio_test module open
ps_key1 press
gpio_test module open
gpio_test module release
```

执行 app 后，打印出的第一句” gpio_test module open”是 open 函数打开/dev/gpio_key 设备时打印出来的，之后按下按键，led 的状态就反转一次并打印 ps_key1 press、gpio_test module open、gpio_test module release 三行信息。