第13章 查询方式的按键驱动程序_编写框架

13.1 LED 驱动回顾

对于 LED，APP 调用 open 函数导致驱动程序的 led_open 函数被调用。在里面，把 GPIO 配置为输出引脚。安装驱动程序后并不意味着会使用对应的硬件，而 APP 要使用对应的硬件，必须先调用 open 函数。所以建议在驱动程序的 open 函数中去设置引脚。

APP 继续调用 write 函数传入数值，在驱动程序的 led_write 函数根据该数值去设置 GPIO 的数据寄存器，从而控制 GPIO 的输出电平。

怎么操作寄存器？从芯片手册得到对应寄存器的物理地址，在驱动程序中使用 ioremap 函数映射得到虚拟地址。驱动程序中使用虚拟地址去访问寄存器。

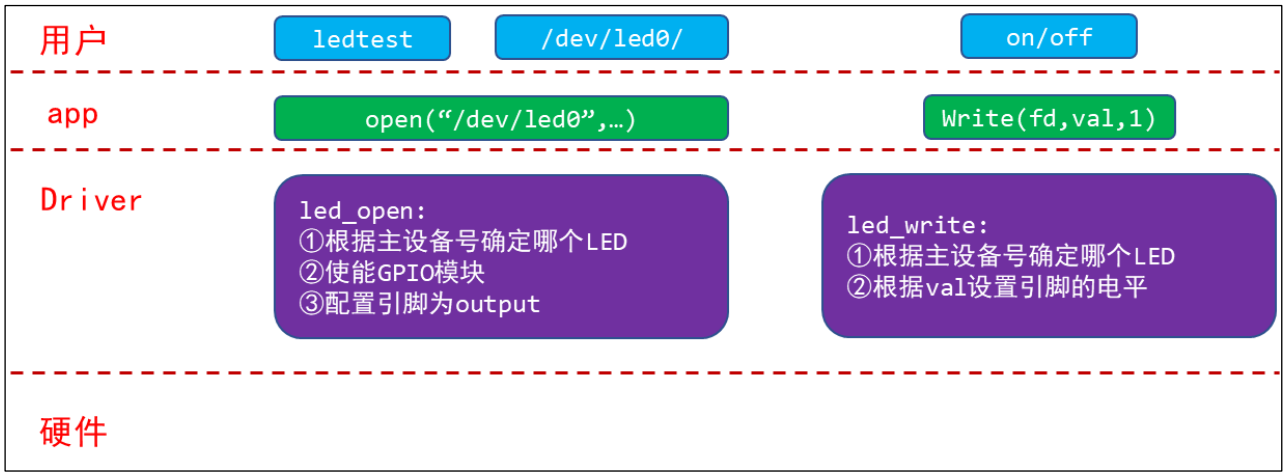

图 13.1 回顾 LED 驱动

13.2 按键驱动编写思路

GPIO 按键的原理图一般有如下 2 种：

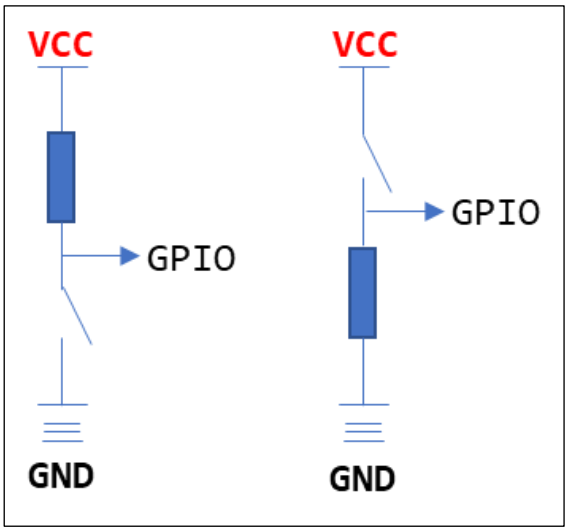

图 13.2 按键原理图示意

- 按键没被按下时，上图中左边的 GPIO 电平为高，右边的 GPIO 电平为低。
- 按键被按下后，上图中左边的 GPIO 电平为低，右边的 GPIO 电平为高。

编写按键驱动程序最简单的方法如图 13.3 所示：

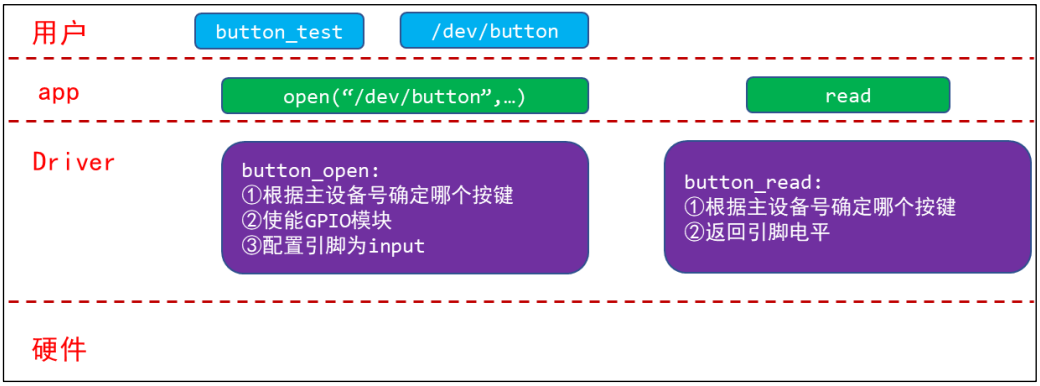

图 13.3 按键驱动编写

回顾一下编写驱动程序的套路：

图 13.4 驱动程序编写套路

对于使用查询方式的按键驱动程序，我们只需要实现 button_open、button_read。

13.3 编程：先写框架

我们的目的写出一个容易扩展到各种芯片、各种板子的按键驱动程序，所以驱动程序分为上下两层：

① button_drv.c 分配/设置/注册 file_operations 结构体

起承上启下的作用，向上提供 button_open,button_read 供 APP 调用。

而这 2 个函数又会调用底层硬件提供的 p_button_opr 中的 init、read 函数操作硬件。

② board_xxx.c 分配/设置/注册 button_operations 结构体

这个结构体是我们自己抽象出来的，里面定义单板 xxx 的按键操作函数。

这样的结构易于扩展，对于不同的单板，只需要替换 board_xxx.c 提供自己的 button_operations 结构体即可。

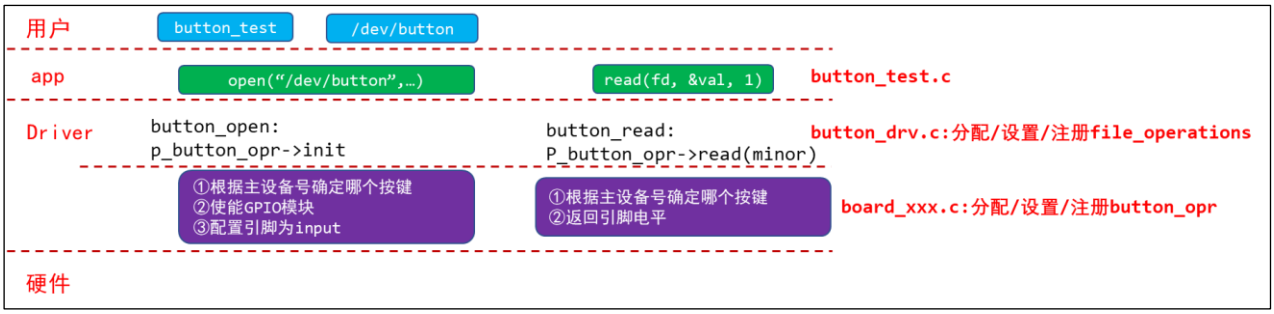

图 13.5 按键驱动框架

使用 GIT 下载所有源码后，本节源码位于如下目录：

```
01_all_series_quickstart\
05_嵌入式 Linux 驱动开发基础知识\
source\04_button_drv\01_button_drv_template
```

13.3.1 把按键的操作抽象出一个 button_operations 结构体

首先看看 button_drv.h，它定义了一个 button_operations 结构体，把按键的操作抽象为这个结构体：

```
04 struct button_operations {
05     int count;
06     void (*init) (int which);
07     int (*read) (int which);
08 };
09
10 void register_button_operations(struct button_operations *opr);
11 void unregister_button_operations(void);
```

再看看 board_xxx.c，它实现了一个 button_operations 结构体，代码如下。

第 45 行调用 register_button_operations 函数，把这个结构体注册到上层驱动中。

```
37 static struct button_operations my_buttons_ops ={
38     .count = 2,
39     .init  = board_xxx_button_init_gpio,
40     .read  = board_xxx_button_read_gpio,
41 };
42
43 int board_xxx_button_init(void)
44 {
45     register_button_operations(&my_buttons_ops);
46     return 0;
47 }
```

13.3.2 驱动程序的上层：file_operations 结构体

上层是 button_drv.c，它的核心是 file_operations 结构体，首先看看入口函数，代码如下。

第 83 行向内核注册一个 file_operations 结构体。

第 85 行创建一个 class，但是该 class 下还没有 device，在后面获得底层硬件的信息时再在 class 下创建 device：这只是用来创建设备节点，它不是驱动程序的核心。

```
81 int button_init(void)
82 {
83     major = register_chrdev(0, "100ask_button", &button_fops);
84
85     button_class = class_create(THIS_MODULE, "100ask_button");
86     if (IS_ERR(button_class))
87         return -1;
88
89     return 0;
90 }
```

再来看看 button_drv.c 中 file_operations 结构体的成员函数，代码如下。

第 34、44 行都用到一个 button_operations 指针，它是从何而来？

```
28 static struct button_operations *p_button_opr;
29 static struct class *button_class;
30
31 static int button_open (struct inode *inode, struct file *file)
32 {
33     int minor = iminor(inode);
34     p_button_opr->init(minor);
35     return 0;
36 }
37
38 static ssize_t button_read (struct file *file, char __user *buf, size_t size, lof
f_t *off)
39 {
40     unsigned int minor = iminor(file_inode(file));
41     char level;
42     int err;
43
44     level = p_button_opr->read(minor);
45     err = copy_to_user(buf, &level, 1);
46     return 1;
47 }
48
49
50 static struct file_operations button_fops = {
51     .open = button_open,
52     .read = button_read,
53 };
```

上面第 34、44 行都用到一个 button_operations 指针，来自于底层硬件相关的代码。

底层代码调用 register_button_operations 函数，向上提供这个结构体指针。

register_button_operations 函数代码如下，它还根据底层提供的 button_operations 调用 device_create，这是创建设备节点(第 62 行)。

```
55 void register_button_operations(struct button_operations *opr)
56 {
57     int i;
58
59     p_button_opr = opr;
60     for (i = 0; i < opr->count; i++)
61     {
62         device_create(button_class, NULL, MKDEV(major, i), NULL, "100ask_button%d", i);
63     }
64 }
```

13.4 测试

这只是一个示例程序，还没有真正操作硬件。测试程序操作驱动程序时，只会导致驱动程序中打印信息。

首先设置交叉工具链，修改驱动 Makefile 中内核的源码路径，编译驱动和测试程序。

启动开发板后，通过 NFS 访问编译好驱动程序、测试程序，就可以在开发板上如下操作了：

```
[root@100ask:~]# insmod button_drv.ko   // 装载驱动程序
[  435.276713] button_drv: loading out-of-tree module taints kernel.
[root@100ask:~]# insmod board_xxx.ko
[root@100ask:~]# ls /dev/100ask_button* -l     // 查看设备节点
crw-------    1 root     root      236,   0 Jan 18 08:57 /dev/100ask_button0
crw-------    1 root     root      236,   1 Jan 18 08:57 /dev/100ask_button1
[root@100ask:~]# ./button_test /dev/100ask_button0    // 读按键
[  450.886180] /home/book/source/04_button_drv/01_button_drv_template/board_xxx.c bo
ard_xxx_button_init_gpio 28, init gpio for button 0
[  450.910915] /home/book/source/04_button_drv/01_button_drv_template/board_xxx.c bo
ard_xxx_button_read_gpio 33, read gpio for button 0
get button : 1    // 得到数据
```

13.5 课后怎业

合并 LED、BUTTON 框架驱动程序：01_led_drv_template、01_button_drv_template，合并为：gpio_drv_template