《嵌入式系统》

（第十讲）

厦门大学信息学院软件工程系 曾文华

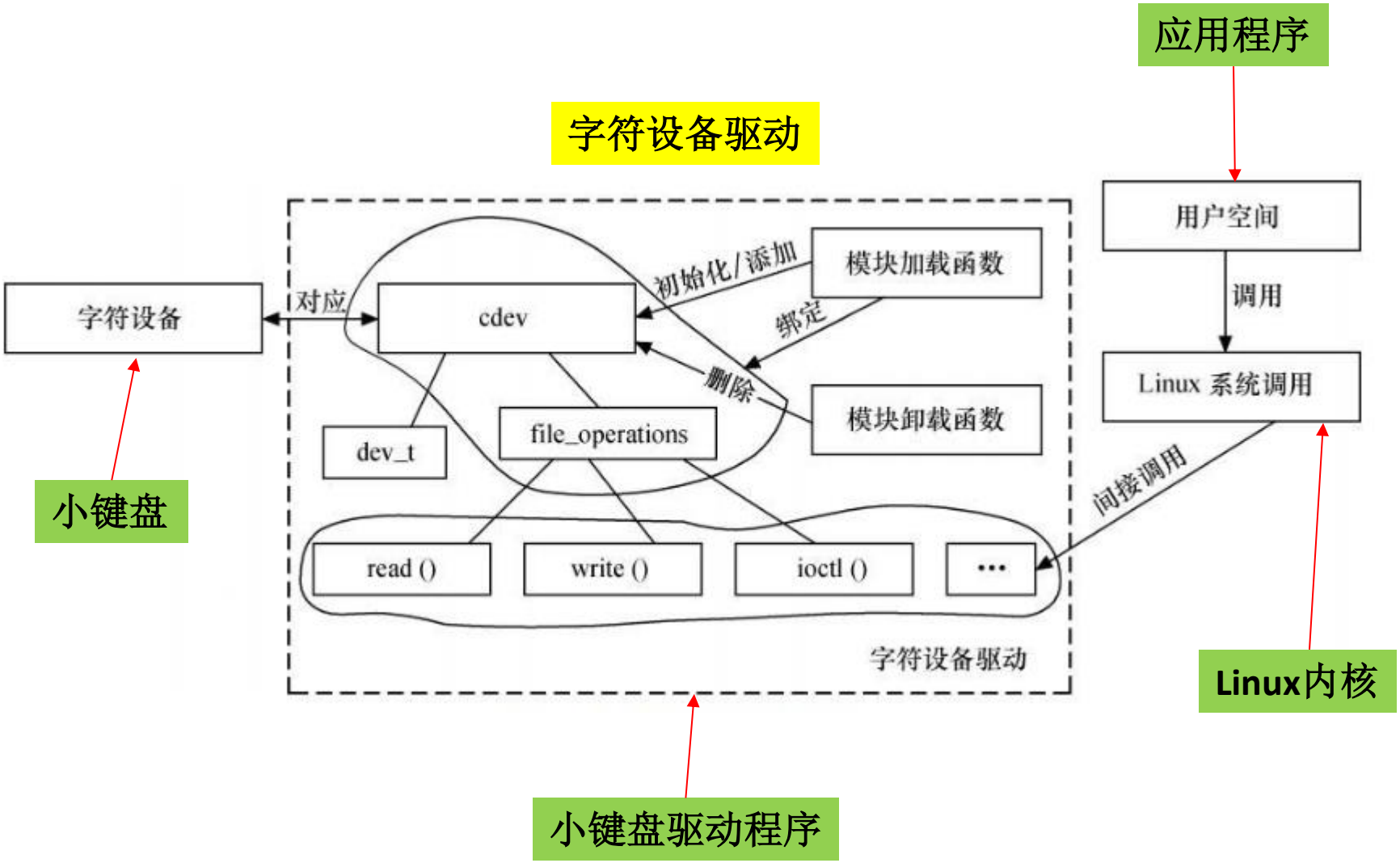
2024年11月5日

第10章　字符设备和驱动程序设计

- 10.1　字符设备驱动框架
- 10.2　字符设备驱动开发
- 10.3　GPIO驱动概述
- 10.4　串口总线概述
- 10.5　字符设备驱动程序示例

- 字符设备是Linux三大设备之一（另外两种是块设备，网络设备）。
- 字符设备就是采用字节流形式通讯的I/O设备，绝大部分设备都是字符设备。
- 常见的字符设备包括鼠标、键盘、显示器、串口等等。

10.1 字符设备驱动框架



- cdev结构（c：字符，dev：设备）
  - dev_t：设备号
  - file_operations：文件操作
    - read()
    - write()
    - ioctl()
    - 等等

字符设备结构体

```
struct cdev {
        struct kobject kobj;
        struct module *owner;
        const struct file_operations *ops;
        struct list_head list;
        dev_t dev;
        unsigned int count;
};
```

文件操作结构体

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **);
};
```

10.2 字符设备驱动开发

- **10.2.1 设备号**
  - 查看主设备号和次设备号：
    - **ls -l /dev**
  - 查看已经加载了驱动程序的主设备号：
    - **cat /proc/devices**
  - 创建指定类型的设备文件：
    - **mknod /dev/lp0 c 6 0**

设备名　字符设备　主设备号　次设备号

查看实验箱的主设备号和次设备号（ls -l /dev）

```
root@localhost:/home/linux# ls -l /dev
total 4
crw-------  1 root  root     10,  55 Feb 11 16:28 binder
drwxr-xr-x  2 root  root         420 Feb 11 16:28 block
drwxr-xr-x  3 root  root          60 Jan  1  1970 bus
crwxrwxrwx  1 root  root     10,  60 Feb 11 16:28 buzzer
crwxrwxrwx  1 root  root     10,  58 Feb 11 16:28 buzzer_ctl
crw-------  1 root  root    250,   0 Feb 11 16:28 cec0
drwxr-xr-x  2 root  root        3640 Feb 11 16:28 char
crw-------  1 root  root      5,   1 Feb 11 16:28 console
crw-------  1 root  root     10,  51 Feb 11 16:28 cpu_dma_latency
crwxrwxrwx  1 root  root     10,  56 Feb 11 16:28 dc_motor
drwxr-xr-x  7 root  root         140 Feb 11 16:28 disk
drwxr-xr-x  2 root  root         100 Jan  1  1970 dri
crwxrwxrwx  1 root  root     10,  57 Feb 11 16:28 farsight_keys
crw-rw----  1 root  video    29,   0 Feb 11 16:28 fb0
lrwxrwxrwx  1 root  root          13 Feb 11 16:28 fd -> /proc/self/fd
crw-rw-rw-  1 root  root      1,   7 Feb 11 16:28 full
crw-rw-rw-  1 root  root     10, 229 Feb 11 16:28 fuse
crw-rw-rw-  1 root  root     10,  46 Feb 11 16:28 hdmi_hdcplx
crw-------  1 root  root     10,  54 Feb 11 16:28 hwbinder
crw-------  1 root  root     10, 183 Feb 11 16:28 hwrng
```

实验箱字符设备的主设备号（cat /proc/devices）

```
root@localhost:/home/linux# cat   /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
 89 i2c
 90 stepmotor
108 ppp
116 alsa
128 ptm
136 pts
401 led
153 spi
166 ttyACM
180 usb
188 ttyUSB
189 usb_device
226 drm
239 hidraw
240 ttyGS
241 usbmon
242 nvme
243 leds_ctl
244 rkvdec
245 vpu_service
```

– 1、设备号类型

- dev_t类型表示设备号：
  - typedef __u32 __kernel_dev_t;
  - typedef __kernel_dev_t dev_t;

- 操作dev_t的函数：
  - #define MINORBITS 20
  - #define MINORMASK ((1U << MINORBITS) - 1)
  - #define MAJOR(dev) ((unsigned int) ((dev) >> MINORBITS))
  - #define MINOR(dev) ((unsigned int) ((dev) & MINORMASK))
  - #define MKDEV(ma,mi) (((ma) << MINORBITS) | (mi))

– 2、注册和注销设备号

- 动态申请设备号范围的函数：
  - extern int alloc_chrdev_region(dev_t *, unsigned, unsigned, const char *);
- 申请设备号的函数（注册）：
  - extern int register_chrdev_region(dev_t, unsigned, const char *);
- 释放设备号的函数（注销）：
  - extern void unregister_chrdev_region(dev_t, unsigned);

- **10.2.2 关键数据结构**
  - **1、file_operations（文件操作结构体）**
    - 改变文件中的读写位置
      - **loff_t (*llseek) (struct file *, loff_t, int);**
    - 从设备中读取数据
      - **ssize_t (*read) (struct file *, char *, size_t, loff_t *);**
    - 向设备写数据
      - **ssize_t (*write) (struct file *, const char *, size_t, loff_t *);**
    - 对设备进行控制
      - **int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);**
    - 将设备内存映射到进程的地址空间
      - **int (*mmap) (struct file *, struct vm_area_struct *);**
    - 打开设备和初始化
      - **int (*open) (struct inode *, struct file *);**
    - 释放设备占用的内存并关闭设备
      - **int (*release) (struct inode *, struct file *);**

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **);
};
```

file_operations结构体

– 2、file（文件结构体）

- 文件的读写模式
  - fmode_t f_mode;
- 文件的当前读写位置
  - loff_t f_pos;
- 文件标志
  - unsigned int f_flags;
- 指向和文件关联的操作
  - const struct file_operations *f_op;
- 指向已分配的数据
  - void *private_data;

file结构体

```
struct file {
        union {
                struct list_head        fu_list;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path                     f_path;
#define f_dentry                        f_path.dentry
#define f_vfsmnt                        f_path.mnt
        const struct file_operations    *f_op;
        spinlock_t                      f_lock;
        atomic_long_t                   f_count;
        unsigned int                    f_flags;
        fmode_t                         f_mode;
        loff_t                          f_pos;
        struct fown_struct              f_owner;
        const struct cred               *f_cred;
        struct file_ra_state            f_ra;
        u64                             f_version;
#ifdef CONFIG_SECURITY
        void                            *f_security;
#endif
        void                            *private_data;
#ifdef CONFIG_EPOLL
        struct list_head                f_ep_links;
#endif
        struct address_space            *f_mapping;
#ifdef CONFIG_DEBUG_WRITECOUNT
        unsigned long                   f_mnt_write_state;
#endif
};
```

文件操作

– 3、inode（索引节点对象结构体）

- 实际的设备号：
  - dev_t i_rdev;
- 指向cdev设备的指针：
  - struct cdev *i_cdev;

inode结构体

```
struct inode {
        struct hlist_node               i_hash;
        struct list_head                i_list;
        struct list_head                i_sb_list;
        struct list_head                i_dentry;
        unsigned long                   i_ino;
        atomic_t                        i_count;
        unsigned int                    i_nlink;
        uid_t                           i_uid;
        gid_t                           i_gid;
        dev_t                           i_rdev;
        u64                             i_version;
        loff_t                          i_size;
#ifdef __NEED_I_SIZE_ORDERED
        seqcount_t                      i_size_seqcount;
#endif
        struct timespec                 i_atime;
        struct timespec                 i_mtime;
        struct timespec                 i_ctime;
        unsigned int                    i_blkbits;
        blkcnt_t                        i_blocks;
        unsigned short                  i_bytes;
        umode_t                         i_mode;
        spinlock_t                      i_lock;
        struct mutex                    i_mutex;
        struct rw_semaphore             i_alloc_sem;
        const struct inode_operations   *i_op;
        const struct file_operations    *i_fop;
        struct super_block              *i_sb;
        struct file_lock                *i_flock;
        struct address_space            *i_mapping;
        struct address_space            i_data;
```

设备号

文件操作

```
#ifdef CONFIG_QUOTA
        struct dquot            *i_dquot[MAXQUOTAS];
#endif
        struct list_head i_devices;
        union {
                struct pipe_inode_info          *i_pipe;
                struct block_device             *i_bdev;
                struct cdev                     *i_cdev;
        };
        int                     i_cindex;
        __u32                   i_generation;
#ifdef CONFIG_DNOTIFY
        unsigned long           i_dnotify_mask;
        struct dnotify_struct   *i_dnotify;
#endif
#ifdef CONFIG_INOTIFY
        struct list_head inotify_watches;
        struct mutex            inotify_mutex;
#endif
        unsigned long           i_state;
        unsigned long           dirtied_when;
        unsigned int            i_flags;
        atomic_t                i_writecount;
#ifdef CONFIG_SECURITY
        void                    *i_security;
#endif
        void                    *i_private;
};
```

cdev结构

inode结构体

- **10.2.3 字符设备注册和注销**
  - 描述字符设备的结构体：cdev结构体

```
struct cdev {
        struct kobject kobj;
        struct module *owner;
        const struct file_operations *ops;
        struct list_head list;
        dev_t dev;
        unsigned int count;
};
```

文件操作

设备号

  - 操作cdev结构体的一组函数：
    - void cdev_init(struct cdev *, const struct file_operations *);
    - struct cdev *cdev_alloc(void);
    - void cdev_put(struct cdev *p);
    - int cdev_add(struct cdev *, dev_t, unsigned);
    - void cdev_del(struct cdev *);
    - void cd_forget(struct inode *);
    - extern struct backing_dev_info directly_mappable_cdev_bdi;

10.3 GPIO驱动概述

- GPIO：General Purpose Input/Output，通用输入输出，可以对GPIO进行编程，将GPIO的每一个引脚设为输入或输出，因此GPIO也称为通用可编程接口。

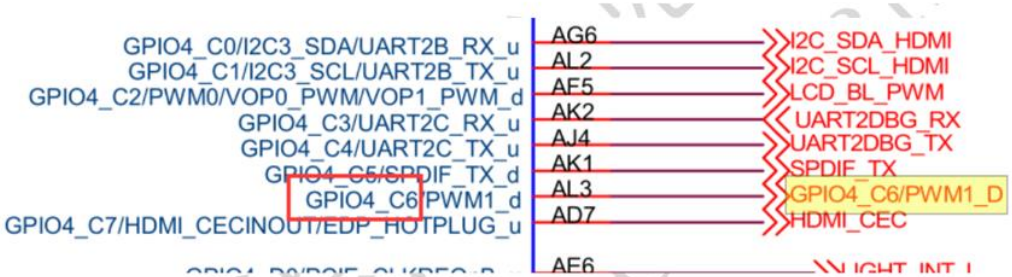
- GPIO接口至少要有两个寄存器：
  - 控制寄存器
  - 数据寄存器

- GPIO的寄存器可以使用内存映射（将I/O当作内存看待，I/O与存储器统一编址，访问I/O与访问存储器一样），或者端口映射（I/O单独编址）。

- 如果使用内存映射，要向GPIO的寄存器A写入数据0xff，设寄存器A的地址为0x36000000，则使用以下代码：
  - #define A (*(volatile unsigned long *)0x36000000)
  - A = 0xff

MPU开发板上的3个LED灯对应的GPIO

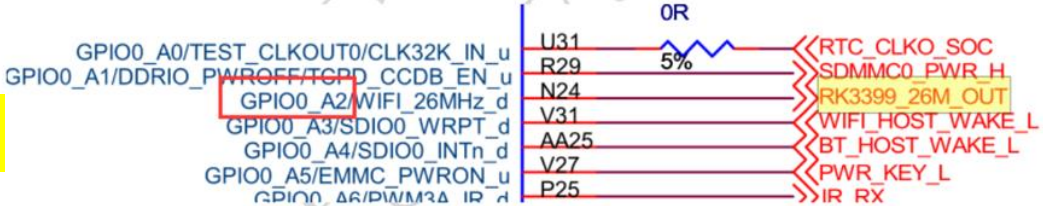
GPIO4_C6



LED1灯

图 6-2 GPIO4_C6/PWM1_D 与核心板的连接

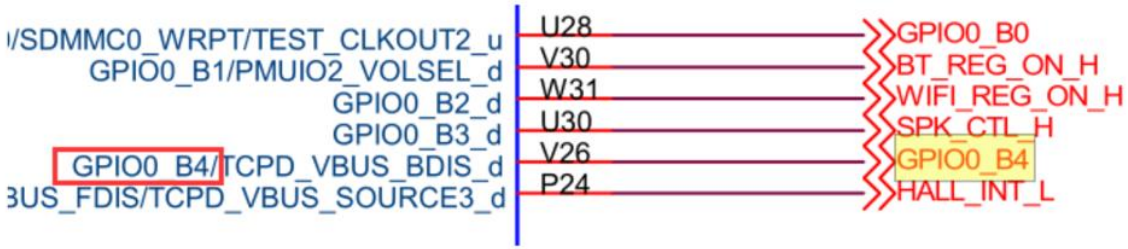
GPIO0_A2



LED2灯

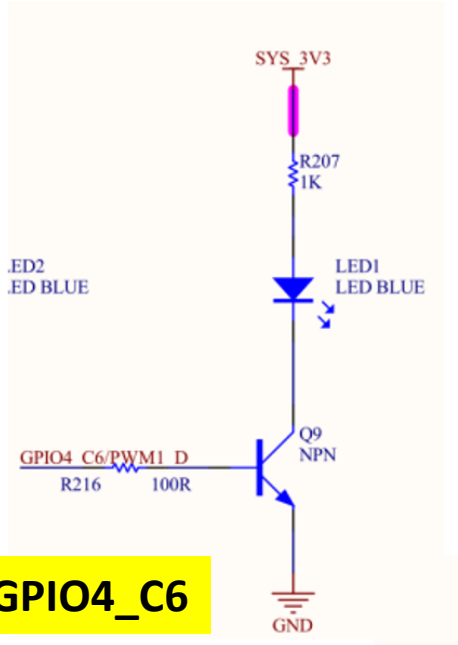
图 6-3 RK3399_26M_OUT 与核心板的连接

GPIO0_B4



LED3灯

MPU开发板上的呼吸灯（LED1灯）对应的GPIO

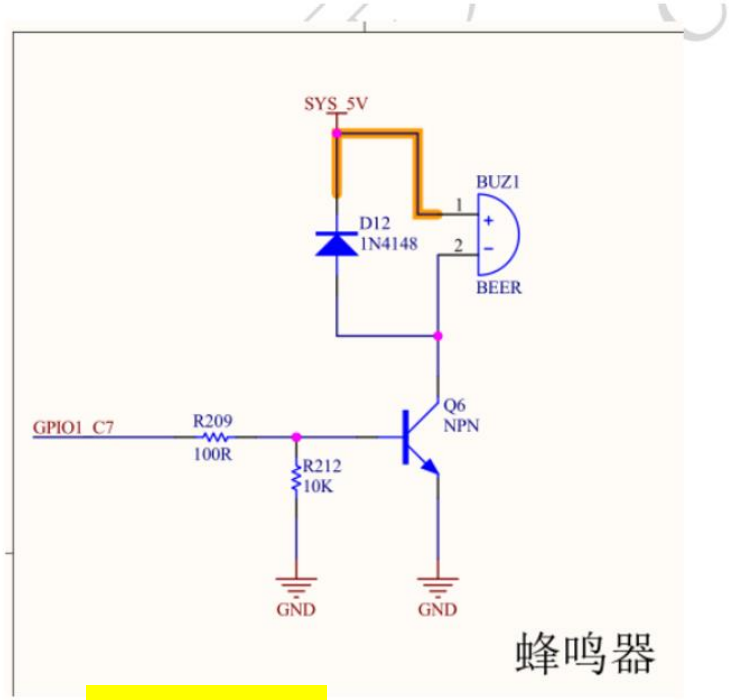


LED1灯

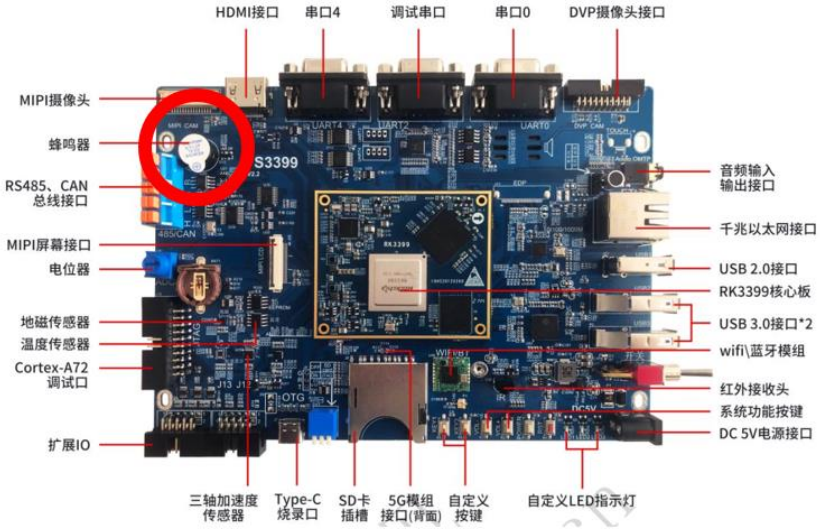
改变这个引脚的信号，使LED1灯由亮变暗、由暗变亮

GPIO4_C6

MPU开发板上的蜂鸣器对应的GPIO

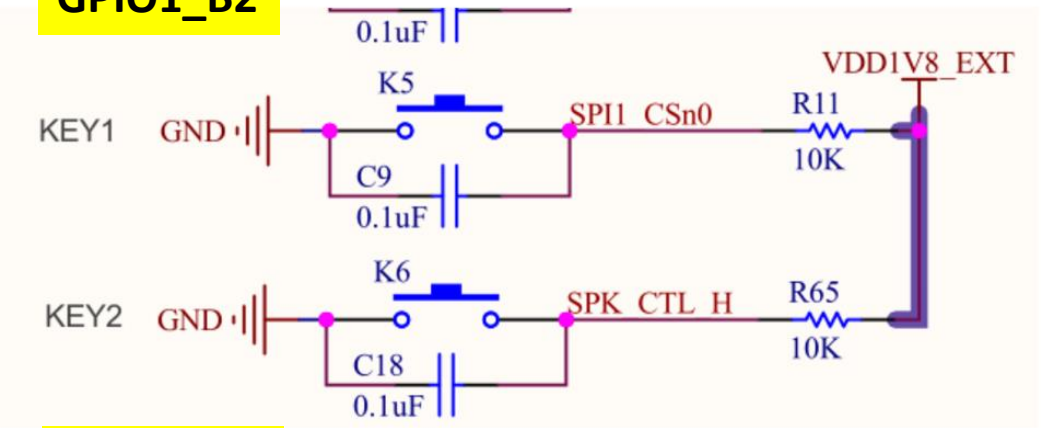


GPIO1_C7

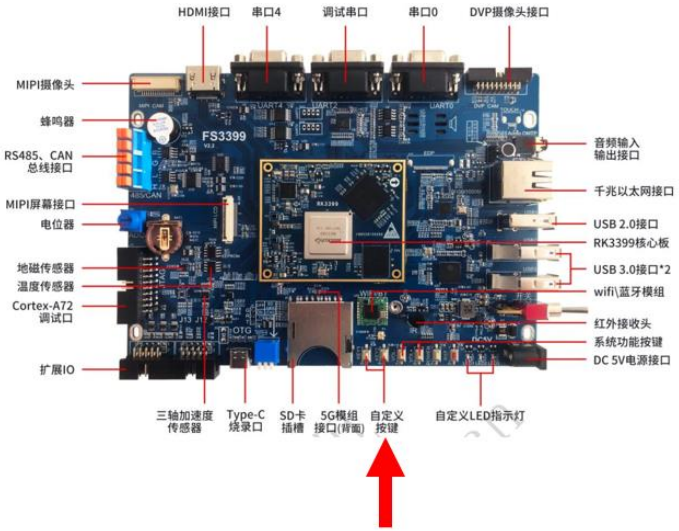


MPU开发板上的2个按键对应的GPIO

GPIO1_B2



GPIO0_B3



2个按键

10.4　串行总线概述

- **10.4.1 SPI总线**
  - SPI是串行外设接口（Serial Peripheral Interface）的缩写。是Motorola 公司推出的一 种同步串行接口技术，是一种高速的，全双工，同步的通信总线。主要应用于EEPROM、Flash、实时时钟、A/D转换以及数字信号处理器和数字信号解码器。SPI的传输速率可达3Mb/s。
  - SPI有两种工作模式：
    - 主模式
    - 从模式
  - SPI有4条接口线：
    - SDI（MISO）：Serial Data In，串行数据输入；
    - SDO（MOSI）：Serial Data Out，串行数据输出；
    - SCLK（SCK）：Serial Clock，时钟信号，由主设备产生；
    - CS（SS）：Chip Select，从设备使能信号，由主设备控制。

- **10.4.2　I^2C总线**

  - **I^2C（Inter Integrated-Circuit，IIC，I2C，内部集成电路）总线，是由PHILIPS公司在上世纪80年代发明的一种电路板级串行总线标准，最初应用于音频和视频领域的设备开发。**

  - **I^2C总线有两根接口线：**
    - **数据线：SDA**
    - **时钟线：SCK，或SCL**

  - **I^2C总线在传输过程中有三种不同类型的信号：**
    - **开始信号**
    - **结束信号**
    - **应答信号**

  - **I^2C总线在标准模式下传输速率可达100kb/s，在快速模式下传输速率可达400kb/s，在高速模式下传输速率可达3.4Mb/s。**

- **10.4.3 SMBus**

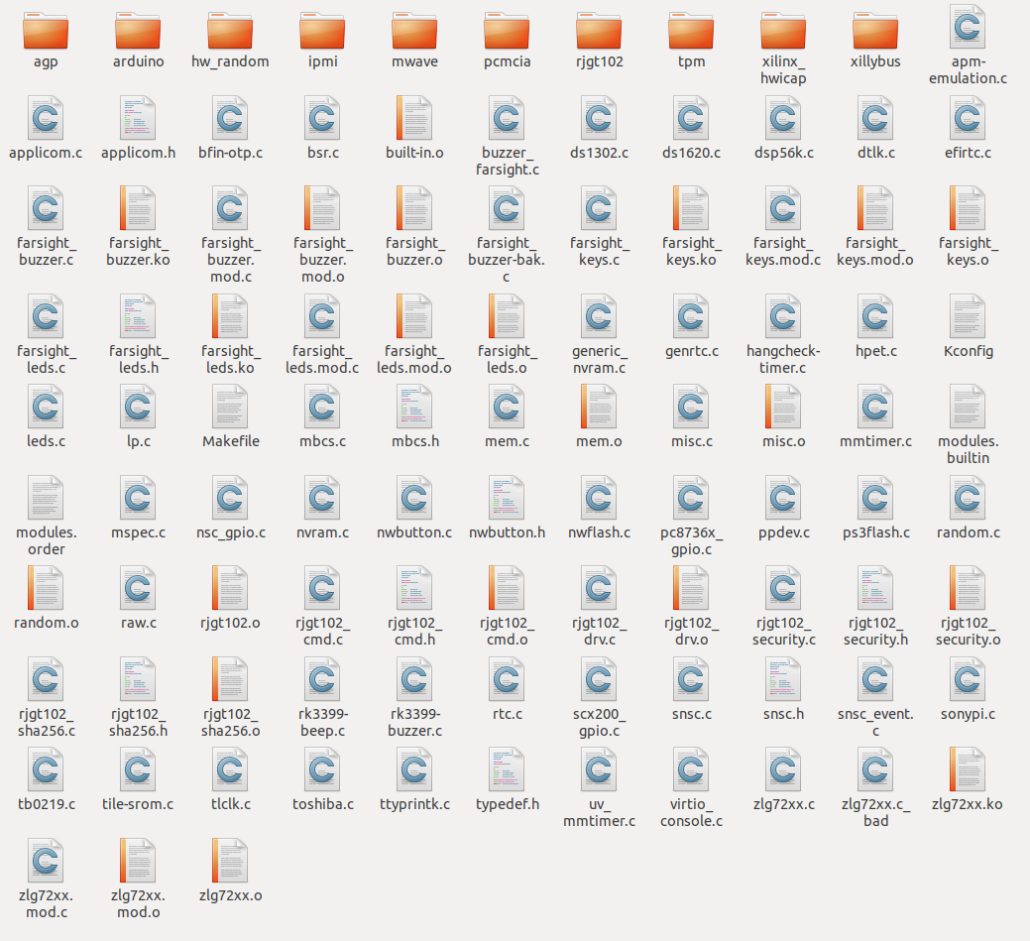
  - **SMBus（System Management Bus，系统管理总线）是1995年由Intel提出的，应用于移动PC和桌面PC系统中的低速率通讯。希望通过一条廉价并且功能强大的总线（由两条线组成），来控制主板上的设备并收集相应的信息。**

  - **SMBus有两根接口线：**
    - **数据线：SMBDAT**
    - **时钟线：SMBCLK**

  - **SMBus的传输率只有100kb/s，SMBus总线的特点是结构简单、造价低。**

10.5　字符设备驱动程序示例

- 位于/home/linux/workdir/fs3399/system/kernel/drivers/char目录



- 第4次实验的16个设备驱动程序

实验1：内核模块
实验2：字符设备驱动
实验3：3个LED灯（基于寄存器控制）
实验4：3个LED灯（基于GPIO子系统控制）
实验5：呼吸灯（LED1灯）
实验6：蜂鸣器
实验7：2个按键
实验8：温度传感器（I2C总线）
实验9：小键盘/数码管
实验10：ADC信号采集（电位器+4个传感器）
实验11：继电器
实验12：光电开关
实验13：蜂鸣器（实验箱底板）
实验14：陀机
实验15：步进电机
实验16：直流电机

module_test.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1、内核模块测试程序

```
//-----------------------------------内核模块测试程序-------------------------
#include <linux/kernel.h>
#include <linux/module.h>

static int __init hello_init(void)
{
        printk("hello init\n");
        return 0;
}

static void __exit hello_exit(void)
{
        printk("hello exit\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_ALIAS("hqyj:module");
MODULE_AUTHOR("HQYJ <yanfa@hqyj.com>");
MODULE_DESCRIPTION("A sample Hello World module");
```

初始化函数

退出函数

模块初始化

模块退出

character.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

2、字符设备驱动程序

```
//-----------------------------------字符设备驱动程序-------------------------
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/cdev.h>
#include <linux/of_gpio.h>
#include <asm/uaccess.h>

struct cdev* cdev;              //字符设备结构体，Linux管理字符设备
struct class* class;            //类结构体
struct device* dev;             //保存设备基本信息的结构体
dev_t devnum;                   //设备编号
```

open()函数

```
static int farsight_open(struct inode *inode, struct file *file)
{
        printk(KERN_NOTICE"This is driver : open function\n");
        return 0;
}
```

read()函数

```
static ssize_t farsight_read(struct file *file, char *buf, size_t count, loff_t* f_pos)
{
        char msg[20] = "abcdefg";
        int ret;
        printk(KERN_NOTICE"This is driver : read function\n");
        ret = copy_to_user(buf,msg,count);
        return count;
}
```

write()函数

```
static ssize_t farsight_write(struct file *filp, const char *buffer, size_t count, loff_t *ppos)
{
        char str[20];
        printk(KERN_NOTICE"This is driver : write function\n");
        memset(str,0,20);
        if(copy_from_user(str, buffer, count))
        {
                return -EINVAL;
        }
        printk(KERN_NOTICE"get APP buf is : %s\n",str);
        return count;
}
```

close()函数

```
static int farsight_close(struct inode *inode, struct file *file)
{
        printk(KERN_NOTICE"This is driver : close function\n");
        return 0;
}
```

ioctl()函数

```
static long farsight_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
        switch(cmd)
        {
                case 'a':
                        printk(KERN_NOTICE"This is driver : ioctl a function\n");
                        break;
                case 'b':
                        printk(KERN_NOTICE"This is driver : ioctl b function\n");
                        break;
                default:
                        return -EINVAL;
        }
        return 0;
}
```

file_operations结构体

```
static struct file_operations farsight_ops =
{
        .owner          = THIS_MODULE,
        .open           = farsight_open,
        .read           = farsight_read,
        .write          = farsight_write,
        .release        = farsight_close,
        .unlocked_ioctl = farsight_ioctl,
};
```

注册设备文件函数：register_char()

```
int register_char(void)
{
        int ret_1;
        cdev_init(cdev,&farsight_ops);                                  //初始化字符设备对象
        cdev->owner = THIS_MODULE;                                      //设置字符设备所属模块
        ret_1 = alloc_chrdev_region(&devnum,0,1,"character_device");    //申请设备号
        if(ret_1<0)
        {
                goto out_err_0;
        }
        ret_1 = cdev_add(cdev, devnum, 1);                              //添加字符设备
        if (ret_1 < 0)
        {
                goto out_err_1;
        }
        class = class_create(THIS_MODULE,"character_class");            //创建character类
        if (IS_ERR(class))
        {
                ret_1 = PTR_ERR(class);
                goto out_err_2;
        }
        dev = device_create(class,NULL,devnum,NULL,"character_device");  //创建字符设备节点
        if (IS_ERR(dev))
        {
                ret_1 = PTR_ERR(dev);
                goto out_err_3;
        }
        return 0;
out_err_3:
        device_del(dev);
        class_destroy(class);
out_err_2:
        cdev_del(cdev);
out_err_1:
        unregister_chrdev_region(devnum,1);
out_err_0:
        kfree(cdev);
        return ret_1;
}
```

```
static int __init farsight_dev_init(void)
{
        cdev = kmalloc(sizeof(struct cdev), GFP_KERNEL);        //注册字符设备
        return register_char();
}

static void __exit farsight_dev_exit(void)
{
        device_del(dev);
        class_destroy(class);
        cdev_del(cdev);
        kfree(cdev);
        unregister_chrdev_region(devnum,1);
}

module_init(farsight_dev_init);
module_exit(farsight_dev_exit);

MODULE_LICENSE("GPL");                  //许可证
MODULE_AUTHOR("FARSIGHT");              //作者
MODULE_DESCRIPTION("LED Driver");       //描述信息
```

初始化函数

退出函数

模块初始化

模块退出

reg_leds_driver.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

3、基于寄存器的LED灯设备驱动程序（不使用设备树文件）

```
//-----------------------------------
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/of_gpio.h>
#include <linux/of.h>
#include <linux/cdev.h>
#include <linux/of_address.h>
#include <linux/of_gpio.h>
#include <asm/uaccess.h>
#include <asm/io.h>

#define LED1_ON    _IO('L',11)
#define LED1_OFF   _IO('L',10)
#define LED2_ON    _IO('L',21)
#define LED2_OFF   _IO('L',20)
#define LED3_ON    _IO('L',31)
#define LED3_OFF   _IO('L',30)

#define DEVICE_NAME        "leds_ctl"
#define DRIVER_NAME        "leds_ctl"

#define GPIO_DR 0x00       //数据寄存器
#define GPIO_DDR 0x04      //方向寄存器

struct cdev* cdev;
struct class* class;
struct device* dev;
dev_t devnum;

void __iomem *gpio0_swporta_base = NULL;      //gpio0基地址
void __iomem *gpio4_swporta_base = NULL;      //gpio4基地址
void __iomem *pclk_gpio0_en = NULL;
void __iomem *pclk_gpio4_en = NULL;
```

open()函数

```
static int farsight_led_open(struct inode *inode, struct file *file)
{
	return 0;
}
```

close()函数

```
static int farsight_led_close(struct inode *inode, struct file *file)
{
	return 0;
}
```

ioctl ()函数

```
//ioctl函数
//功能：控制3个LED开关
//参数1：文件指针,应用层文件描述符
//参数2：应用层下发控制指令
//参数3：用户空间传递的数据
static long farsight_led_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
	switch(cmd)
	{
		case LED1_ON:
			writel((readl((gpio4_swporta_base + GPIO_DR)) | 0x1<<22) , (gpio4_swporta_base + GPIO_DR));
			break;
		case LED1_OFF:
			writel((readl((gpio4_swporta_base + GPIO_DR)) & ~(0x1<<22)) , (gpio4_swporta_base + GPIO_DR));
			break;
		case LED2_ON:
			writel((readl((gpio0_swporta_base + GPIO_DR)) | 0x1<<2) , (gpio0_swporta_base + GPIO_DR));
			break;
		case LED2_OFF:
			writel((readl((gpio0_swporta_base + GPIO_DR)) & ~(0x1<<2)) , (gpio0_swporta_base + GPIO_DR));
			break;
		case LED3_ON:
			writel((readl((gpio0_swporta_base + GPIO_DR)) | 0x1<<12) , (gpio0_swporta_base + GPIO_DR));
			break;
		case LED3_OFF:
			writel((readl((gpio0_swporta_base + GPIO_DR)) & ~(0x1<<12)) , (gpio0_swporta_base + GPIO_DR));
			break;
		default:
			return -EINVAL;
	}
	return 0;
}
```

file_operations 结构体

```
static struct file_operations farsight_led_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

初始化函数（设备初始化 + 设备文件注册）

```
static int __init farsight_led_dev_init(void)
{
        int ret_1;
        gpio0_swporta_base = ioremap(0xFF720000,8);                                  //映射GPIO0的基地址
        gpio4_swporta_base = ioremap(0xFF790000,8);                                  //映射GPIO4的基地址
        pclk_gpio0_en = ioremap((0xFF750000+0x104),4);                               //映射GPIO0的时钟寄存器地址
        pclk_gpio4_en = ioremap((0xFF760000+0x37c),4);                               //映射GPIO4的时钟寄存器地址
        writel(((readl(pclk_gpio0_en) & ~(0x1<<3)) | 0x1<<19) , pclk_gpio0_en);     //使能GPIO0的时钟
        writel(((readl(pclk_gpio4_en) & ~(0x1<<5)) | 0x1<<21) , pclk_gpio4_en);     //使能GPIO4的时钟
        writel(readl((gpio4_swporta_base + GPIO_DDR)) | 0x1<<22, gpio4_swporta_base + GPIO_DDR);   //设置GPIO4的DDR的相应位为输出模式（LED1）
        writel(readl((gpio0_swporta_base + GPIO_DDR)) | 0x1<<2, gpio0_swporta_base + GPIO_DDR);    //设置GPIO0的DDR的相应位为输出模式（LED2）
        writel(readl((gpio0_swporta_base + GPIO_DDR)) | 0x1<<12, gpio0_swporta_base + GPIO_DDR);   //设置GPIO0的DDR的相应位为输出模式（LED3）
        cdev = kmalloc(sizeof(struct cdev), GFP_KERNEL);
        cdev_init(cdev,&farsight_led_ops);                          //初始化字符设备对象
        cdev->owner = THIS_MODULE;                                  //设置字符设备所属模块
        ret_1 = alloc_chrdev_region(&devnum,0,1,"leds_reg_device");  //申请设备号
        if(ret_1<0)
        {
                goto out_err_0;
        }
        ret_1 = cdev_add(cdev, devnum, 1);                          //添加字符设备
        if (ret_1 < 0)
        {
                goto out_err_1;
        }
        class = class_create(THIS_MODULE,"leds_reg_class");         //创建character类
        if (IS_ERR(class))
        {
                ret_1 = PTR_ERR(class);
                goto out_err_2;
        }
        dev = device_create(class,NULL,devnum,NULL,"leds_reg_device");  //创建字符设备节点
        if (IS_ERR(dev))
        {
                ret_1 = PTR_ERR(dev);
                goto out_err_3;
        }
        return 0;
```

设备名称：/dev/leds_reg_device

```
out_err_3:
        device_del(dev);
        class_destroy(class);
out_err_2:
        cdev_del(cdev);
out_err_1:
        unregister_chrdev_region(devnum,1);
out_err_0:
        kfree(cdev);
        return ret_1;
}

static void __exit farsight_led_dev_exit(void)
{
        iounmap(gpio0_swporta_base);
        iounmap(gpio4_swporta_base);
        iounmap(pclk_gpio0_en);
        iounmap(pclk_gpio4_en);
        device_del(dev);
        class_destroy(class);
        cdev_del(cdev);
        kfree(cdev);
        unregister_chrdev_region(devnum,1);
}

module_init(farsight_led_dev_init);
module_exit(farsight_led_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("FARSIGHT");
MODULE_DESCRIPTION("LED reg Driver");
```

退出函数

模块初始化

模块退出

gpio_leds.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

4、基于GPIO的LED灯设备驱动程序（使用设备树文件）

```
//-----------------------------------3个LED灯设备驱动程序（使用设备树文件）
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/cdev.h>
#include <linux/of_gpio.h>
#include <asm/uaccess.h>

#define LED1_ON     _IO('G',1)
#define LED1_OFF    _IO('G',2)
#define LED2_ON     _IO('G',3)
#define LED2_OFF    _IO('G',4)
#define LED3_ON     _IO('G',5)
#define LED3_OFF    _IO('G',6)

#define DEVICE_NAME         "leds_ctl"
#define DRIVER_NAME         "leds_ctl"
```

定义设备名： **/dev/leds_ctl**

LED灯设备驱动结构体

```
struct led_driver_t{
        struct cdev *cdev_leds;          //字符设备结构体，Linux管理字符设备
        struct class* class_leds;        //类结构体
        struct device* dev_leds;         //保存设备基本信息的结构体
        struct device_node *np;          //保存设备树节点信息结构体
        dev_t devnum;
};

static uint32_t LED_GPIO1 = 0;
static uint32_t LED_GPIO2 = 0;
static uint32_t LED_GPIO3 = 0;

static struct led_driver_t *led;
```

open()函数

```
static int farsight_led_open(struct inode *inode, struct file *file)
{
        return 0;
}
```

close()函数

```
static int farsight_led_close(struct inode *inode, struct file *file)
{
        return 0;
}
```

ioctl()函数

```
//ioctl函数
//功能：控制3个LED开关
//参数1：文件指针,应用层文件描述符
//参数2：应用层下发控制指令
//参数3：用户空间传递的数据
static long farsight_led_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {
            case LED1_ON:
                    gpio_set_value(LED_GPIO1, 1);
                    printk(KERN_ERR " ### LED1 ON ###\n  ");
                    break;
            case LED1_OFF:
                    gpio_set_value(LED_GPIO1, 0);
                    printk(KERN_ERR " ### LED1 OFF ###\n  ");
                    break;
            case LED2_ON:
                    gpio_set_value(LED_GPIO2, 1);
                    printk(KERN_ERR " ### LED2 ON ###\n  ");
                    break;
            case LED2_OFF:
                    gpio_set_value(LED_GPIO2, 0);
                    printk(KERN_ERR " ### LED2 OFF ###\n  ");
                    break;
            case LED3_ON:
                    gpio_set_value(LED_GPIO3, 1);
                    printk(KERN_ERR " ### LED3 ON ###\n  ");
                    break;
            case LED3_OFF:
                    gpio_set_value(LED_GPIO3, 0);
                    printk(KERN_ERR " ### LED3 OFF ###\n  ");
                    break;
            default:
                    return -EINVAL;
    }
    return 0;
}
```

file_operations 结构体

```
static struct file_operations farsight_led_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

注册设备文件的函数

```
int register_leds(struct platform_device *pdev)
{
        int ret_1;
        cdev_init(led->cdev_leds,&farsight_led_ops);                    //初始化字符设备对象
        led->cdev_leds->owner = THIS_MODULE;                            //设置字符设备所属模块
        ret_1 = alloc_chrdev_region(&led->devnum,0,1,"leds_ctl");       //申请设备号
        if(ret_1<0){
                dev_err(&pdev->dev, "alloc_chrdev_region() failed ret: %d.\n", ret_1);
                goto out_err_0;
        }
        ret_1 = cdev_add(led->cdev_leds, led->devnum, 1);               //添加字符设备
        if (ret_1 < 0) {
                dev_err(&pdev->dev, "cdev_add() failed ret: %d.\n", ret_1);
                goto out_err_1;
        }
        led->class_leds = class_create(THIS_MODULE,"leds_ctl");         //创建LED类
        if (IS_ERR(led->class_leds)) {
                ret_1 = PTR_ERR(led->class_leds);
                dev_err(&pdev->dev, "class_create() failed ret: %d.\n", ret_1);
                goto out_err_2;
        }
        led->dev_leds = device_create(led->class_leds,NULL,led->devnum,NULL,"leds_ctl");   //创建字符设备节点
        if (IS_ERR(led->dev_leds)) {
                ret_1 = PTR_ERR(led->dev_leds);
                dev_err(&pdev->dev, "device_create() failed ret: %d.\n", ret_1);
                goto out_err_3;
        }
        return 0;
out_err_3:
        device_del(led->dev_leds);
        class_destroy(led->class_leds);
out_err_2:
        cdev_del(led->cdev_leds);
out_err_1:
        unregister_chrdev_region(led->devnum,1);
out_err_0:
        kfree(led);
        return ret_1;
}
```

probe（探查）函数（设备初始化）

设备树匹配成功后执行该函数

```
static int farsight_led_probe(struct platform_device *pdev)
{
        int ret;
        led = kmalloc(sizeof(struct led_driver_t), GFP_KERNEL);          //申请内存
        led->np = pdev->dev.of_node;                                     //获取设备树节点信息

        LED_GPIO1 = of_get_named_gpio(led->np,"led1", 0);                //获取LED1引脚的管脚号
        if (LED_GPIO1 == -EPROBE_DEFER)
                return LED_GPIO1;
        if (LED_GPIO1 < 0) {
                dev_err(&pdev->dev, "error acquiring led gpio: %d\n", LED_GPIO1);
                return LED_GPIO1;
        }
        ret = gpio_request(LED_GPIO1,NULL);                              //向内核注册该引脚，确保不被其他驱动占用
        if(ret) {
                dev_err(&pdev->dev, "error requesting led gpio: %d\n", ret);
                return ret;
        }
        gpio_direction_output(LED_GPIO1, 0);                             //设置LED1引脚为输出模式，初始为低电平

        LED_GPIO2 = of_get_named_gpio(led->np,"led2", 0);                //获取LED2引脚的管脚号
        if (LED_GPIO2 == -EPROBE_DEFER)
                return LED_GPIO2;
        if (LED_GPIO2 < 0) {
                dev_err(&pdev->dev, "error acquiring led gpio: %d\n", LED_GPIO2);
                return LED_GPIO2;
        }
        ret = gpio_request(LED_GPIO2,NULL);                              //向内核注册该引脚，确保不被其他驱动占用
        if(ret) {
                dev_err(&pdev->dev, "error requesting led gpio: %d\n", ret);
                return ret;
        }
        gpio_direction_output(LED_GPIO2, 0);                             //设置LED2引脚为输出模式，初始为低电平
```

```
    LED_GPIO3 = of_get_named_gpio(led->np,"led3", 0);                    //获取LED3引脚的管脚号
    if (LED_GPIO1 == -EPROBE_DEFER)
            return LED_GPIO3;
    if (LED_GPIO1 < 0) {
            dev_err(&pdev->dev, "error acquiring led gpio: %d\n", LED_GPIO3);
            return LED_GPIO3;
    }
    ret = gpio_request(LED_GPIO3,NULL);                          //向内核注册该引脚，确保不被其他驱动占用
    if(ret) {
            dev_err(&pdev->dev, "error requesting led gpio: %d\n", ret);
            return ret;
    }
    gpio_direction_output(LED_GPIO3, 0);                         //设置LED3引脚为输出模式，初始为低电平

    ret = register_leds(pdev);                                   //注册字符设备
    platform_set_drvdata(pdev,led);                              //存储在probe()中的指针以防止丢失
    return 0;
}
```

移除设备文件的函数

```
static int farsight_led_remove (struct platform_device *pdev)
{
        gpio_free(LED_GPIO1);
        gpio_free(LED_GPIO2);
        gpio_free(LED_GPIO3);
        device_del(led->dev_leds);
        class_destroy(led->class_leds);
        cdev_del(led->cdev_leds);
        unregister_chrdev_region(led->devnum,1);
        kfree(led);
        return 0;
}
```

暂停设备文件的函数

```
static int farsight_led_suspend (struct platform_device *pdev, pm_message_t state)
{
        return 0;
}
```

恢复设备文件的函数

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

设备树匹配表结构体

```
#ifdef CONFIG_OF
static const struct of_device_id led_of_match[] = {
        { .compatible = "farsight_led" },
        {}
};
MODULE_DEVICE_TABLE(of, led_of_match);                  //将设备加入到外设队列，告诉程序员读者，这是个热插拔设备
#endif
```

platform（平台总线）结构体

```
//内核中的platform结构体对象，定义了操作对象的方法
static struct platform_driver farsight_led_driver = {
        .probe = farsight_led_probe,                    //设备树和匹配表匹配成功后之后执行函数，在该函数中，一般初始化设备，申请驱动所需资源
        .remove = farsight_led_remove,                  //从内核删除设备，释放资源
        .suspend = farsight_led_suspend,                //挂起
        .resume = farsight_led_resume,                  //唤醒
        .driver = {                                     //设备驱动通用属性
                .name = DRIVER_NAME,                    //设备驱动的名字
                .owner = THIS_MODULE,                   //表示实现该驱动程序的模块
                .of_match_table = of_match_ptr(led_of_match),  //与设备树进行匹配，使用id_table
        },
};
```

初始化函数（注册platform）

```
//驱动模块入口
static int __init farsight_led_dev_init(void)
{
        return platform_driver_register(&farsight_led_driver);   //将驱动注册到platform总线上
}
```

退出函数（注销platform）

```
//驱动模块出口
static void __exit farsight_led_dev_exit(void)
{
        platform_driver_unregister(&farsight_led_driver);        //将驱动从platform总线删除
}
```

模块初始化　　模块退出

```
module_init(farsight_led_dev_init);
module_exit(farsight_led_dev_exit);

MODULE_LICENSE("GPL");//许可证
MODULE_AUTHOR("FARSIGHT");//作者
MODULE_DESCRIPTION("LED Driver");//描述信息
```

pwm_led_driver.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

5、呼吸灯设备驱动程序（不使用设备树文件）

```
//-----------------------------------
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/of_gpio.h>
#include <linux/of.h>
#include <linux/cdev.h>
#include <linux/of_address.h>
#include <linux/of_gpio.h>
#include <asm/uaccess.h>
#include <asm/io.h>

#define LED1_PERIOD  _IO('L',0)
#define LED1_DUTY    _IO('L',1)
#define LED1_ON      _IO('L',2)
#define LED1_OFF     _IO('L',3)

#define DEVICE_NAME        "leds_pwm_ctl"
#define DRIVER_NAME        "leds_pwm_ctl"

struct cdev* cdev;
struct class* class;
struct device* dev;

dev_t devnum;

#define PWM_CNT 0x00
#define PWM_PERIOD_HPR 0x04
#define PWM_DUTY_LPR 0x08
#define PWM_CTRL 0x0c

void __iomem *pclk_gpio4_en = NULL;
void __iomem *grf_gpio4c_iomux = NULL;
void __iomem *pwm1 = NULL;
```

open()函数

```
static int farsight_led_open(struct inode *inode, struct file *file)
{
        return 0;
}
```

close()函数

```
static int farsight_led_close(struct inode *inode, struct file *file)
{
        return 0;
}
```

ioctl ()函数

```
static long farsight_led_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
        unsigned int buf;
        switch(cmd)
        {
                case LED1_PERIOD:
                        if(copy_from_user((void *)&buf,(void *)arg,4))
                                return -EFAULT;
                        writel(buf,(pwm1+PWM_PERIOD_HPR));
                        break;
                case LED1_DUTY:
                        if(copy_from_user((void *)&buf,(char*)arg,4))
                                return -EFAULT;
                        writel(buf,(pwm1+PWM_DUTY_LPR));
                        break;
                case LED1_ON:
                        printk("4\n");
                        writel((readl(pwm1 + PWM_CTRL) | (0x1<<0)),(pwm1 + PWM_CTRL));
                        break;
                case LED1_OFF:
                        writel((readl(pwm1 + PWM_CTRL) | (0x0<<0)),(pwm1 + PWM_CTRL));
                        break;
                default:
                        return -EINVAL;
        }
        return 0;
}
```

file_operations 结构体

```
static struct file_operations farsight_pwm_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_led_open,
        .release        = farsight_led_close,
        .unlocked_ioctl = farsight_led_ioctl,
};
```

初始化函数（设备初始化 + 设备文件注册）

```
static int __init farsight_led_pwm_init(void)
{
        int ret_1;
        pclk_gpio4_en = ioremap((0xFF760000+0x37c),4);
        grf_gpio4c_iomux = ioremap(0xFF77E028,4);
        pwm1 = ioremap(0xFF420010,4);
//-------------------------------PWM配置---------------------------------------------
        writel(((readl(pclk_gpio4_en) | 0x1<<26) & (~(0x1<<10))), pclk_gpio4_en);          //使能GPIO4时钟
        writel(((readl(grf_gpio4c_iomux) | 0x3<<28) | 0x1<<12),grf_gpio4c_iomux);          //设置GPIO4_C6为PWM
        writel((readl(pwm1 + PWM_CTRL) & (~(0x1<<0))),(pwm1 + PWM_CTRL));                  //关闭PWM通道
        writel((readl(pwm1 + PWM_CTRL) | (0x0<<9)),(pwm1 + PWM_CTRL));                     //选择时钟源
        writel((readl(pwm1 + PWM_CTRL) | (0x1<<12)),(pwm1 + PWM_CTRL));                    //设置输入时钟的分频因子，2的1次方
        writel((readl(pwm1 + PWM_CTRL) | (0x1<<16)),(pwm1 + PWM_CTRL));                    //设置缩放比例因子,时钟除以2的N次方,这里设置为2的1次方
        writel((readl(pwm1 + PWM_CTRL) | (0x1<<1)),(pwm1 + PWM_CTRL));                     //设置连续模式，PWM产生连续波形
        writel((readl(pwm1 + PWM_CTRL) | (0x1<<3)),(pwm1 + PWM_CTRL));                     //设置PWM的极性为positive
        writel((readl(pwm1 + PWM_CTRL) & (~(0x1<<5))),(pwm1 + PWM_CTRL));                  //输出方式为左对齐
        writel((readl(pwm1 + PWM_CTRL) | (0x1<<0)),(pwm1 + PWM_CTRL));                     //使能PWM通道
//-----------------------------创建字符设备-----------------------------------------
        cdev = kmalloc(sizeof(struct cdev), GFP_KERNEL);
        cdev_init(cdev,&farsight_pwm_ops);                          //初始化字符设备对象
        cdev->owner = THIS_MODULE;                                  //设置字符设备所属模块
        ret_1 = alloc_chrdev_region(&devnum,0,1,"led_pwm_device");  //申请设备号
        if(ret_1<0)
        {
                goto out_err_0;
        }
        ret_1 = cdev_add(cdev, devnum, 1);                          //添加字符设备
        if (ret_1 < 0)
        {
                goto out_err_1;
        }
        class = class_create(THIS_MODULE,"leds_pwm_class");         //创建character类
        if (IS_ERR(class))
        {
                ret_1 = PTR_ERR(class);
                goto out_err_2;
        }
        dev = device_create(class,NULL,devnum,NULL,"leds_pwm_device");  //创建字符设备节点
        if (IS_ERR(dev))
        {
                ret_1 = PTR_ERR(dev);
                goto out_err_3;
        }
        return 0;
```

```
out_err_3:
        device_del(dev);
        class_destroy(class);
out_err_2:
        cdev_del(cdev);
out_err_1:
        unregister_chrdev_region(devnum,1);
out_err_0:
        kfree(cdev);
        return ret_1;
}

static void __exit farsight_led_pwm_exit(void)
{
        iounmap(pclk_gpio4_en);
        iounmap(grf_gpio4c_iomux);
        iounmap(pwm1);
        device_del(dev);
        class_destroy(class);
        cdev_del(cdev);
        kfree(cdev);
        unregister_chrdev_region(devnum,1);
}

module_init(farsight_led_pwm_init);
module_exit(farsight_led_pwm_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("FARSIGHT");
MODULE_DESCRIPTION("LED pwm Driver");
```

退出函数

模块初始化

模块退出

buzzer_driver.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

6、蜂鸣器设备驱动程序（使用设备树文件）

```
//-----------------------------------------------------------------------
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/of_gpio.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <asm/uaccess.h>
#include <asm/io.h>

#define DEVICE_NAME "buzzer_ctl"
#define DRIVER_NAME "buzzer_ctl"

#define Buzzer_on   1
#define Buzzer_off  0

#define GPIO_DR 0x00
#define GPIO_DDR 0x04

unsigned int addr_base[1] = {0};                    //基地址缓冲区
void __iomem *pclk_gpio1_en = NULL;                 //GPIO1使能时钟寄存器
void __iomem *gpio1_swporta_base = NULL;            //GPIO1虚拟映射地址

struct device_node *buzzer_node;
```

ioctl ()函数

```
static long farsight_buzzer_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
    printk(KERN_ERR "%s:\n", __func__);
    switch(cmd)
    {
        case Buzzer_off:
                        printk(KERN_ERR "buzzer off\n");
                        writel((readl((gpio1_swporta_base + GPIO_DR)) & ~(0x1<<23)) , (gpio1_swporta_base + GPIO_DR));        //置数据寄存器=0
                        break;
        case Buzzer_on:
                        printk(KERN_ERR "buzzer on\n");
                        writel((readl((gpio1_swporta_base + GPIO_DR)) | 0x1<<23) , (gpio1_swporta_base + GPIO_DR));        //置数据寄存器=1
                        break;
        default:
                        return -EINVAL;
    }
    return 0;
}
```

open ()函数

```
static int farsight_buzzer_open(struct inode *inode, struct file *file)
{
        printk(KERN_ERR "thisd *** %s  \n",__func__);
        return 0;
}
```

close ()函数

```
static int farsight_buzzer_close(struct inode *inode, struct file *file)
{
        printk(KERN_ERR "thisd *** %s  \n",__func__);
        return 0;
}
```

file_operations 结构体

```
static struct file_operations farsight_buzzer_ops = {
    .owner          = THIS_MODULE,
    .open           = farsight_buzzer_open,
    .release        = farsight_buzzer_close,
    .unlocked_ioctl = farsight_buzzer_ioctl,
};
```

miscdevice 杂项设备结构体

```
static struct miscdevice farsight_misc_dev = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &farsight_buzzer_ops,
};
```

probe（探查）函数（设备初始化）

```
static int farsight_buzzer_probe(struct platform_device *pdev)
{
        int ret = 0;
        ret = misc_register(&farsight_misc_dev);                                    //注册杂项设备
        if(ret<0)
            return ret;
        buzzer_node = pdev->dev.of_node;                                            //获取设备树节点信息
        ret = of_property_read_u32_array(buzzer_node, "gpio1_base", addr_base, 1);  //设备树获取基地址
        if(ret<0)
            return ret;
        gpio1_swporta_base = ioremap(addr_base[0],8);                               //映射GPIO1基地址
        ret = of_property_read_u32_array(buzzer_node, "pclk_gpio1", addr_base, 1);  //设备树获取基地址
        if(ret<0)
            return ret;
        pclk_gpio1_en = ioremap((addr_base[0]+0x104),8);                            //映射使能GPIO1时钟寄存器基地址
        writel(((readl(pclk_gpio1_en) & ~(0x1<<4)) | 0x1<<20) , pclk_gpio1_en);    //使能GPIO1时钟
        writel(readl((gpio1_swporta_base + GPIO_DDR)) | 0x1<<23, (gpio1_swporta_base + GPIO_DDR));  //设置蜂鸣器引脚为输出模式
        return 0;
};
```

移除设备文件的函数

```
static int farsight_buzzer_remove (struct platform_device *pdev)
{
        iounmap(gpio1_swporta_base);
        iounmap(pclk_gpio1_en);
        misc_deregister(&farsight_misc_dev);
        return 0;
}
```

设备树匹配表结构体

```
static const struct of_device_id buzzer_of_match[] = {
    { .compatible = "farsight_buzzer" },
    {}
};
```

platform（平台总线）结构体

```
static struct platform_driver farsight_buzzer_driver = {
   .probe = farsight_buzzer_probe,
   .remove = farsight_buzzer_remove,
   .driver = {
      .name = DRIVER_NAME,
      .owner = THIS_MODULE,
      .of_match_table = of_match_ptr(buzzer_of_match),
   },
};
```

```
static int __init farsight_buzzer_dev_init(void)
{
        return platform_driver_register(&farsight_buzzer_driver);
}

static void __exit farsight_buzzer_dev_exit(void)
{
        platform_driver_unregister(&farsight_buzzer_driver);
}

module_init(farsight_buzzer_dev_init);
module_exit(farsight_buzzer_dev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("FARSIGHT");
MODULE_DESCRIPTION("FARSIGHT BUZZER Driver");
```

初始化函数（注册platform）

退出函数（注销platform）

模块初始化

模块退出

farsight_keys.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

7、按键设备驱动程序（使用设备树文件）

```
//-----------------------------
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/err.h>
#include <linux/gpio.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_gpio.h>
#include <linux/interrupt.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <asm/uaccess.h>

struct keys_gpio_info {
        int KEY_GPIO[2];
        int KEY_IRQ[2];
        int KEY_MODE[2];
        int value[2];
        int state;
        int tmp_data[2];
        wait_queue_head_t wq_head;
};

static struct keys_gpio_info *mykeys;
```

```
static int farsight_keys_open(struct inode *inode, struct file *file)
{
        return 0;
}
```

open ()函数

```
static int farsight_keys_close(struct inode *inode, struct file *file)
{
        return 0;
}
```

close ()函数

```
static ssize_t farsight_keys_read(struct file *file, char __user * ubuf, size_t size, loff_t *loff_t)
{
        int ret;
        wait_event_interruptible(mykeys->wq_head, mykeys->tmp_data[0]!=0);
        ret = copy_to_user((void *)ubuf,mykeys->tmp_data,sizeof(mykeys->tmp_data));
        if(ret != 0)
        {
                printk("copy_to_user err\n");
                return ret;
        }
        mykeys->tmp_data[0] = 0;
        return 2;
}
```

read()函数

```
static struct file_operations farsight_keys_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_keys_open,
        .release        = farsight_keys_close,
        .read           = farsight_keys_read,
};
```

file_operations 结构体

```
static struct miscdevice farsight_keys_dev = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = "farsight_keys",
        .fops = &farsight_keys_ops,
};
```

miscdevice 杂项设备结构体

按键1中断回调函数

```
static irqreturn_t key1_irq_callback(int irq, void *dev_id)                    //按键KEY1
{
        mykeys->value[0] = gpio_get_value(mykeys->KEY_GPIO[0]);
        mykeys->tmp_data[0] = 0x01;
        mykeys->tmp_data[1] = mykeys->value[0];
        wake_up_interruptible(&mykeys->wq_head);
        return IRQ_HANDLED;
}
```

按键2中断回调函数

```
static irqreturn_t key2_irq_callback(int irq, void *dev_id)                    //按键KEY2
{
        mykeys->value[1] = gpio_get_value(mykeys->KEY_GPIO[1]);
        mykeys->tmp_data[0] = 0x02;
        mykeys->tmp_data[1] = mykeys->value[1];
        wake_up_interruptible(&mykeys->wq_head);
        return IRQ_HANDLED;
}
```

probe（探查）函数（设备初始化）

```
static int keys_gpio_probe(struct platform_device *pdev)
{
        int ret;
        enum of_gpio_flags flag;
        struct device_node *keys_gpio_node = pdev->dev.of_node;
        mykeys = devm_kzalloc(&pdev->dev,sizeof(struct keys_gpio_info *), GFP_KERNEL);
        if (!mykeys)
        {
                dev_err(&pdev->dev, "devm_kzalloc failed!\n");
                return -ENOMEM;
        }

/************************************** KEY1属性 ****************************************************/
        mykeys->KEY_GPIO[0] = of_get_named_gpio_flags(keys_gpio_node, "key1-gpio", 0, &flag);
        if (!gpio_is_valid(mykeys->KEY_GPIO[0]))
        {
                dev_err(&pdev->dev, "keys-gpio: %d is invalid\n", mykeys->KEY_GPIO[0]);
                return -ENODEV;
        }
        mykeys->KEY_IRQ[0] = gpio_to_irq(mykeys->KEY_GPIO[0]);
        mykeys->KEY_MODE[0] = flag;
        if (gpio_request(mykeys->KEY_GPIO[0], "key1-gpio"))
        {
                dev_err(&pdev->dev, "keys-gpio: %d request failed!\n", mykeys->KEY_GPIO[0]);
                gpio_free(mykeys->KEY_GPIO[0]);
                return -ENODEV;
        }
        ret = request_irq(mykeys->KEY_IRQ[0],key1_irq_callback,mykeys->KEY_MODE[0],"KEY1_IRQ",mykeys);
        if(ret != 0)
        {
                free_irq(mykeys->KEY_IRQ[0],mykeys);
                dev_err(&pdev->dev, "Failed to request key1 IRQ: %d\n", ret);
        }
        gpio_direction_input(mykeys->KEY_GPIO[0]);
```

```
/********************************** KEY2属性 ********************************************************/
        mykeys->KEY_GPIO[1] = of_get_named_gpio_flags(keys_gpio_node, "key2-gpio", 0, &flag);
        if (!gpio_is_valid(mykeys->KEY_GPIO[1]))
        {
                dev_err(&pdev->dev, "keys-gpio: %d is invalid\n", mykeys->KEY_GPIO[1]);
                return -ENODEV;
        }
        mykeys->KEY_IRQ[1] = gpio_to_irq(mykeys->KEY_GPIO[1]);
        mykeys->KEY_MODE[1] = flag;
         if (gpio_request(mykeys->KEY_GPIO[1], "key2-gpio"))
        {
                dev_err(&pdev->dev, "keys-gpio: %d request failed!\n", mykeys->KEY_GPIO[1]);
                gpio_free(mykeys->KEY_GPIO[1]);
                return -ENODEV;
        }
        ret = request_irq(mykeys->KEY_IRQ[1],key2_irq_callback,mykeys->KEY_MODE[1],"KEY2_IRQ",mykeys);
        if(ret != 0)
        {
                free_irq(mykeys->KEY_IRQ[1],mykeys);
                dev_err(&pdev->dev, "Failed to request key2 IRQ: %d\n", ret);
        }
        gpio_direction_input(mykeys->KEY_GPIO[1]);

        mykeys->state = 0;
        init_waitqueue_head(&mykeys->wq_head);          //初始化等待队列头
        ret = misc_register(&farsight_keys_dev);        //注册杂项驱动
        return 0;
}
```

```
static int keys_gpio_remove(struct platform_device *pdev)
{
        gpio_free(mykeys->KEY_GPIO[0]);
        free_irq(mykeys->KEY_IRQ[0],mykeys);
        gpio_free(mykeys->KEY_GPIO[1]);
        free_irq(mykeys->KEY_IRQ[1],mykeys);
        misc_deregister(&farsight_keys_dev);
        kfree(mykeys);
        return 0;
}
```

移除设备文件的函数

```
static struct of_device_id keys_match_table[] = {
    { .compatible = "farsight_keys",},
    {},
};
```

设备树匹配表结构体

```
static struct platform_driver keys_gpio_driver = {
    .driver = {
        .name = "farsight_keys",
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(keys_match_table),
    },
    .probe  = keys_gpio_probe,
    .remove = keys_gpio_remove,
};
```

platform（平台总线）结构体

```
static int keys_gpio_init(void)
{
    return platform_driver_register(&keys_gpio_driver);
}
module_init(keys_gpio_init);
```

初始化函数（注册**platform**）

```
static void keys_gpio_exit(void)
{
    platform_driver_unregister(&keys_gpio_driver);
}
module_exit(keys_gpio_exit);
```

退出函数（注销**platform**）

```
MODULE_AUTHOR("FARSIGHT");
MODULE_DESCRIPTION("FARSIGHT KEYS DRIVER");
MODULE_LICENSE("GPL");
```

farsight_lm75.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

8、（i2c）温度传感器设备驱动程序（使用设备树文件）

```
//-----------------------------------
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/i2c.h>
#include <linux/slab.h>
#include <asm/uaccess.h>

#define LM75_REG_CONF      0x01

static const u8 LM75_REG_TEMP[3] = {
        0x00,                           // input
        0x03,                           // max
        0x02,                           // hyst
};

struct lm75_data
{
        u16 temp[3];
};

static int lm75_major = 300;
static int lm75_minor = 0;
static int number_of_devices = 1;
static dev_t devno = 0;
static struct cdev cdev;
struct class* class;
struct device* dev;
static struct i2c_client *new_client;
struct lm75_data *data;
```

read_value ()函数

```
static int lm75_read_value(struct i2c_client *client)
{
        struct i2c_msg msgs[2];
        int status;
        char buf1[2];
        char buf2[2];
        msgs[0].len = 1;
        msgs[0].addr = client->addr;                    // lm75 设备地址
        msgs[0].flags = 0;                              //write
        msgs[0].buf = buf1;
        msgs[0].buf[0] = LM75_REG_TEMP[0];
        msgs[1].len = 2;                                //读出的数据
        msgs[1].addr = client->addr;                    // lm75 设备地址
        msgs[1].flags = I2C_M_RD;                       //read
        msgs[1].buf = buf2;                             //存放返回值的地址。
        status = i2c_transfer(client->adapter, msgs, 2);
        if(status < 0)
                return status;
        return (buf2[0] << 8) | buf2[1];
}
```

read()函数

```
static ssize_t lm75_read(struct file *file, char __user *buff, size_t count, loff_t *offset)
{
        int status;
        status = lm75_read_value(new_client);
        if(status < 0)
        {
                return status;
        }
        if(copy_to_user(buff, (char *)&status, sizeof(status)))
                return -EFAULT;
        return 0;
}
```

```
static int lm75_open(struct inode *inode, struct file *file)
{
        return 0;
}

static int lm75_release(struct inode *inode, struct file *file)
{
        return 0;
}

static struct file_operations lm75_fops = {
        .owner    = THIS_MODULE,
        .read     = lm75_read,
        .open     = lm75_open,
        .release  = lm75_release,
};
```

open ()函数

release ()函数

file_operations 结构体

probe（探查）函数（设备初始化）

```
static int lm75_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
        int ret = 0;
        new_client = client;
        if (!i2c_check_functionality(client->adapter, I2C_FUNC_SMBUS_BYTE_DATA | I2C_FUNC_SMBUS_WORD_DATA))
                return -EIO;
        data = kzalloc(sizeof(struct lm75_data), GFP_KERNEL);
        if (!data)
                return -ENOMEM;
        i2c_set_clientdata(client, data);
        devno = MKDEV(lm75_major, lm75_minor);
        ret = register_chrdev_region(devno, number_of_devices, "lm75");
        if(ret)
        {
                printk("failed to register device number\n");
                goto err_register_chrdev_region;
        }
        cdev_init(&cdev, &lm75_fops);
        cdev.owner = THIS_MODULE;
        ret = cdev_add(&cdev, devno, number_of_devices);
        if(ret)
        {
                printk("failed to add device\n");
                goto err_cdev_add;
        }
        class = class_create(THIS_MODULE,"lm75_class");
        if (IS_ERR(class))
        {
                ret = PTR_ERR(class);
                goto out_err_2;
        }
        dev = device_create(class,NULL,devno,NULL,"temp");
        if (IS_ERR(dev))
        {
                ret = PTR_ERR(dev);
                goto out_err_3;
        }
        return 0;
out_err_3:
        device_del(dev);
        class_destroy(class);
out_err_2:
        cdev_del(&cdev);
err_cdev_add:
        unregister_chrdev_region(devno, number_of_devices);
```

```
        return 0;
out_err_3:
        device_del(dev);
        class_destroy(class);
out_err_2:
        cdev_del(&cdev);
err_cdev_add:
        unregister_chrdev_region(devno, number_of_devices);
err_register_chrdev_region:
        kfree(data);
        return ret;
}

static int lm75_remove(struct i2c_client *client)
{
        cdev_del(&cdev);
        device_del(dev);
        class_destroy(class);
        unregister_chrdev_region(devno, number_of_devices);
        return 0;
}

enum lm75_type {
        lm75,
        lm75a,
};

static const struct i2c_device_id lm75_ids[] = {
        { "lm75", lm75, },
        { "lm75a", lm75a, },
        {}
};

static struct i2c_driver lm75_driver = {
        .driver = {
                .name = "lm75",
        },
        .probe     = lm75_probe,
        .remove    = lm75_remove,
        .id_table  = lm75_ids,
};
```

移除设备文件的函数

i2c总线设备结构体

i2c总线驱动结构体

```
static int __init s5pc100_lm75_init(void)
{
        return i2c_add_driver(&lm75_driver);
}

static void __exit s5pc100_lm75_exit(void)
{
        i2c_del_driver(&lm75_driver);
}

module_init(s5pc100_lm75_init);
module_exit(s5pc100_lm75_exit);

MODULE_LICENSE ("GPL");
```

初始化函数（增加i2c总线）

退出函数（删除i2c总线）

zlg72xx_driver.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

9、小键盘/数码管设备驱动程序（使用设备树文件）

```
//--------------------------------
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/irq.h>
#include <linux/gpio.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/platform_device.h>
#include <linux/module.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/poll.h>
#include <linux/sched.h>

#define ZLG72XX_NAME        "zlg72xx"
#define ZLG72128_NAME       "zlg72128"
#define ZLG7290_NAME        "zlg7290"

#define SET_VAL _IO('Z', 0)
#define GET_KEY _IO('Z', 1)

unsigned int ZLG72128_ID = 1;
unsigned int ZLG7290_ID = 0;

#define CONFIG_ZLG72XX_INPUT_DEVICE

unsigned int zlg72XX_major = 401;
unsigned int zlg72XX_minor = 1;
```

```
struct zlg72XX{
        struct i2c_client *client;
        struct delayed_work work;
#ifdef CONFIG_ZLG72XX_INPUT_DEVICE
        struct input_dev *key;
#endif
        struct cdev cdev;
        int current_key;
        wait_queue_head_t readq;
        struct class *class;
#ifndef CONFIG_OF
        const struct i2c_device_id *id;
#endif
#ifdef CONFIG_OF
        const struct of_device_id *id_tree;
#endif
};

#ifdef CONFIG_ZLG72XX_INPUT_DEVICE
unsigned int key_value[65] = {
        0,
        KEY_D, KEY_NUMERIC_POUND, KEY_0, KEY_NUMERIC_STAR,  5,  6,  7,  8,
        KEY_C, KEY_9, KEY_8, KEY_7, 13, 14, 15, 16,
        KEY_B, KEY_6, KEY_5, KEY_4, 21, 22, 23, 24,
        KEY_A, KEY_3, KEY_2, KEY_1, 29, 30, 31, 32,
        33, 34, 35, 36, 37, 38, 39, 40,
        41, 42, 43, 44, 45, 46, 47, 48,
        49, 50, 51, 52, 53, 54, 55, 56,
        57, 58, 59, 60, 61, 62, 63, 64,
};
#endif
```

```
static int zlg72XX_hw_write(struct  zlg72XX *ctr_zlg72XX,  int len, size_t *retlen, char *buf)
{
        struct i2c_client *client = ctr_zlg72XX->client;
        int ret;
        struct i2c_msg msg[] = {
                { client->addr, 0, len, buf},
        };
        ret =i2c_transfer(client->adapter, msg, 1);
        if (ret < 0)
        {
                dev_err(&client->dev, "i2c write error/n");
                return -EIO;
        }
        *retlen = len;
        return 0;
}
```

write ()函数

```
static int zlg72XX_hw_read(struct zlg72XX *ctr_zlg72XX , int len, size_t *retlen, char *buf)
{
        struct i2c_client *client = ctr_zlg72XX->client;
        int ret;
        struct i2c_msg msg[] = {
                { client->addr, 0, len, buf},
                { client->addr, I2C_M_RD, len, buf },
        };
        ret =i2c_transfer(client->adapter, msg, 2);
        if (ret < 0)
        {
                dev_err(&client->dev, "i2c read error/n");
                return ret;
        }
        *retlen = len;
        return 0;
}
```

read ()函数

open ()函数

```
static int zlg72XX_open(struct inode *inode, struct file *file)
{
        struct cdev* open_cdev = inode->i_cdev;
        struct zlg72XX* open_zlg72xx = container_of(open_cdev,struct zlg72XX,cdev);
        file->private_data = open_zlg72xx;
#ifndef CONFIG_ZLG72XX_INPUT_DEVICE
        schedule_delayed_work(&open_zlg72xx->work, HZ / 5);
#endif
        return 0;
}
```

release ()函数

```
static int zlg72XX_release(struct inode *inode, struct file *file)
{
#ifndef CONFIG_ZLG72XX_INPUT_DEVICE
        cancel_delayed_work_sync(&release_zlg72xx->work);
#endif
        return 0;
}
```

work ()函数

```
static void zlg72XX_work(struct work_struct *work)
{
        struct zlg72XX *ctr_zlg72xx = container_of(work, struct zlg72XX, work.work);
        unsigned char val = 0;
        size_t len;
        unsigned char status = 0;
        zlg72XX_hw_read(ctr_zlg72xx, 1, &len, &status);
        if(status & 0x1)
        {
                val = 0x1;
                zlg72XX_hw_read(ctr_zlg72xx, 1, &len, &val);
                if (val == 0)
                {
                        val = 3;
                        zlg72XX_hw_read(ctr_zlg72xx, 1, &len, &val);
                        if (val == 0 || val == 0xFF)
                                goto out;
                }
#ifndef CONFIG_OF
                if(ctr_zlg72xx->id->driver_data == ZLG7290_ID)
                {
#endif
#ifdef CONFIG_OF
                if((unsigned int *)(ctr_zlg72xx->id_tree->data) == &ZLG7290_ID)
                {
#endif
                        if (val > 56)
                        {
                                switch (val)
                                {
                                        case 0xFE: val = 57; break;
                                        case 0xFD: val = 58; break;
                                        case 0xFB: val = 59; break;
                                        case 0xF7: val = 60; break;
                                        case 0xEF: val = 61; break;
                                        case 0xDF: val = 62; break;
                                        case 0xBF: val = 63; break;
                                        case 0x80: val = 64; break;
                                        default : goto out;
                                }
                        }
                }
#ifndef CONFIG_OF
```

```
                                        default : goto out;
                                }
                        }
                }
#ifndef CONFIG_OF
                if(ctr_zlg72xx->id->driver_data == ZLG72128_ID)
                {
#endif
#ifdef CONFIG_OF
                if((unsigned int *)(ctr_zlg72xx->id_tree->data) == &ZLG72128_ID)
                {
#endif
                        switch(val)
                        {
                                case 0xF7: val = 28; break;                     //[3][3]
                                case 0xFB: val = 27; break;                     //[3][2]
                                case 0xFD: val = 26; break;                     //[3][1]
                                case 0xFE: val = 25; break;                     //[3][0]
                                case 0x14: val = 20; break;                     //[2][3]
                                case 0x13: val = 19; break;                     //[2][2]
                                case 0x12: val = 18; break;                     //[2][1]
                                case 0x11: val = 17; break;                     //[2][0]
                                case 0x0C: val = 12; break;                     //[1][3]
                                case 0x0B: val = 11; break;                     //[1][2]
                                case 0x0A: val = 10; break;                     //[1][1]
                                case 0x09: val =  9; break;                     //[1][0]
                                case 0x04: val =  4; break;                     //[0][3]
                                case 0x03: val =  3; break;                     //[0][2]
                                case 0x02: val =  2; break;                     //[0][1]
                                case 0x01: val =  1; break;                     //[0][0]
                                default : goto out;
                        }
                }
#ifdef CONFIG_ZLG72XX_INPUT_DEVICE
                input_report_key(ctr_zlg72xx->key, key_value[val], 0);
                input_sync(ctr_zlg72xx->key);
#endif
                ctr_zlg72xx->current_key = val;
                wake_up_interruptible(&ctr_zlg72xx->readq);
        }
out:
        schedule_delayed_work(&ctr_zlg72xx->work, HZ / 5);
}
```

ioctl ()函数

```
static long zlg72XX_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
        struct zlg72XX* ioctl_zlg72xx = file->private_data;
        unsigned char buf[9] = {0};
        ssize_t len = 0;
        unsigned short ioctl_len = 0;
        unsigned int ioctl_flag1 = 0;
        unsigned int ioctl_flag2 = 0;
        unsigned char val[2] = {0};
        unsigned char reg[8] = {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17};
        int i = 0;
        switch (cmd)
        {
            case SET_VAL:
                if(copy_from_user(buf, (char *)arg, 9))
                        ioctl_len = strlen(buf);
                for(i=0;i<ioctl_len;i++)
                {
                        if(buf[i] == '.')
                        ioctl_flag1 = 1;
                }
                if (ioctl_flag1 == 1)
                        ioctl_len = 9;
                else
                        ioctl_len = 8;
                for(i = 0; i < ioctl_len; i++)
                {
                        if(buf[i] == '.')
                        {
                                val[0] = reg[i-1];
                                ioctl_flag2 = 1;
                        }
                        if(ioctl_flag2 == 1)
                                val[0] = reg[i-1];
                        else
                                val[0] =reg[i];
#ifndef CONFIG_OF
                        if(ioctl_zlg72xx->id->driver_data == ZLG7290_ID)
                        {
#endif
#ifdef CONFIG_OF
                        if((unsigned int *)(ioctl_zlg72xx->id_tree->data) == &ZLG7290_ID)
                        {
#endif
                                switch(buf[i])
                                {
                                        case '0': val[1] = 0xFC; break;
```

```
                        switch(buf[i])
                        {
                                case '0': val[1] = 0xFC; break;
                                case '1': val[1] = 0x0C; break;
                                case '2': val[1] = 0xDA; break;
                                case '3': val[1] = 0xF2; break;
                                case '4': val[1] = 0x66; break;
                                case '5': val[1] = 0xB6; break;
                                case '6': val[1] = 0xBE; break;
                                case '7': val[1] = 0xE0; break;
                                case '8': val[1] = 0xFE; break;
                                case '9': val[1] = 0xF6; break;
                                case 'a':
                                case 'A': val[1] = 0xEE; break;
                                case 'b':
                                case 'B': val[1] = 0x3E; break;         //7F
                                case 'c':
                                case 'C': val[1] = 0x9C; break;
                                case 'd':
                                case 'D': val[1] = 0x7A; break;         //3F
                                case 'e':
                                case 'E': val[1] = 0x9E; break;
                                case 'f':
                                case 'F': val[1] = 0x8E; break;
                                case ' ': val[1] = 0x00; break;
                                case '.':
                                        if(val[1] != 0x00)
                                                val[1] |= 0x01;
                                        break;
                                default:
                                        val[1] = 0x00; break;
                        }
                }
#ifndef CONFIG_OF
                if(ioctl_zlg72xx->id->driver_data == ZLG7290_ID)
                {
#endif
```

```
#ifdef CONFIG_OF
                        if((unsigned int *)(ioctl_zlg72xx->id_tree->data) == &ZLG72128_ID)
                        {
#endif
                                switch(buf[i])
                                {
                                        case '0': val[1] = 0x3f; break;
                                        case '1': val[1] = 0x30; break;
                                        case '2': val[1] = 0x5B; break;
                                        case '3': val[1] = 0x4F; break;
                                        case '4': val[1] = 0x66; break;
                                        case '5': val[1] = 0x6D; break;
                                        case '6': val[1] = 0x7D; break;
                                        case '7': val[1] = 0x07; break;
                                        case '8': val[1] = 0x7F; break;
                                        case '9': val[1] = 0x6F; break;
                                        case 'a':
                                        case 'A': val[1] = 0x77; break;
                                        case 'b':
                                        case 'B': val[1] = 0x7C; break;         //7F
                                        case 'c':
                                        case 'C': val[1] = 0x39; break;
                                        case 'd':
                                        case 'D': val[1] = 0x5E; break;         //3F
                                        case 'e':
                                        case 'E': val[1] = 0x79; break;
                                        case 'f':
                                        case 'F': val[1] = 0x71; break;
                                        case ' ': val[1] = 0x00; break;
                                        case '.':
                                                if(val[1] != 0x00)
                                                        val[1] |= 0x80;
                                                break;
                                        default:
                                                val[1] = 0x00; break;
                                }
                        }
                        msleep(10);
                        zlg72XX_hw_write(ioctl_zlg72xx, 2, &len, val);
                }
                break;
```

```
                break;
        case GET_KEY:
                wait_event_interruptible(ioctl_zlg72xx->readq, ioctl_zlg72xx->current_key != 0);
                if (copy_to_user((void *)arg, &ioctl_zlg72xx->current_key, 4))
                        return -EFAULT;
                ioctl_zlg72xx->current_key = 0;
                break;
        }
        ioctl_flag1 = 0;
        ioctl_flag2 = 0;
        return 0;
}

static struct file_operations zlg72XX_fops = {
        .owner = THIS_MODULE,
        .open = zlg72XX_open,
        .release = zlg72XX_release,
        .unlocked_ioctl = zlg72XX_ioctl,
};
```

file_operations 结构体

注册设备的函数

```
static int register_zlg72XX_device(struct zlg72XX *zlg)
{
        int ret;
        struct zlg72XX* register_dev_zlg72xx = zlg;
        dev_t devno = MKDEV(zlg72XX_major, zlg72XX_minor);
        ret = register_chrdev_region(devno, 1, "led");
        if (ret < 0)
        {
                printk("Failed: register_chrdev_region : ret = %d\n",ret);
                return -1;
        }
        cdev_init(&register_dev_zlg72xx->cdev, &zlg72XX_fops);
        register_dev_zlg72xx->cdev.owner = THIS_MODULE;
        ret = cdev_add(&register_dev_zlg72xx->cdev, devno, 1);
        if (ret < 0)
        {
                unregister_chrdev_region(devno, 1);
                printk("Failed: cdev_add\n");
                return -1;
        }
        register_dev_zlg72xx->class = class_create(THIS_MODULE, "zlg72xx-old");
        if (register_dev_zlg72xx->class == NULL)
        {
                printk("failed: class_create\n");
                return -1;
        }
        device_create(register_dev_zlg72xx->class, NULL, devno, NULL, "zlg72xx");
        return 0;
}
```

注销设备的函数

```
static int unregister_zlg72XX_device(struct zlg72XX * zlg)
{
        struct zlg72XX* unregister_dev_zlg72xx = zlg;
        dev_t devno = MKDEV(zlg72XX_major, zlg72XX_minor);
        device_destroy(unregister_dev_zlg72xx->class, devno);
        class_destroy(unregister_dev_zlg72xx->class);
        cdev_del(&unregister_dev_zlg72xx->cdev);
        unregister_chrdev_region(devno, 1);
        return 0;
}
```

```
#ifdef CONFIG_OF
static const struct of_device_id zlg72XX_of[] = {
        {
                .compatible = ZLG7290_NAME,
                .data = &ZLG7290_ID
        },{
                .compatible = ZLG72128_NAME,
                .data = &ZLG72128_ID
        },{
        },
};
MODULE_DEVICE_TABLE(of,zlg72XX_of);
#endif

static int zlg72XX_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
#ifdef CONFIG_ZLG72XX_INPUT_DEVICE
        struct input_dev *key_dev;
        int i = 0;
#endif
        unsigned char probe_status = 3;
        size_t probe_len;
        struct zlg72XX* probe_zlg72xx;
        int ret = 0;
        if(!(probe_zlg72xx = kzalloc(sizeof(struct zlg72XX),GFP_KERNEL)))
                return -ENOMEM;
        memset(probe_zlg72xx,0,sizeof(struct zlg72XX));
        if (!i2c_check_functionality(client->adapter, I2C_FUNC_I2C))
        {
                return -ENODEV;
        }
        probe_zlg72xx->client = client;
        ret = zlg72XX_hw_read(probe_zlg72xx,0x1,&probe_len, &probe_status);
        if(ret < 0)
        {
                printk("driver_probe_72XX one of them doesn't %d\n",ret);
                goto err1;
        }
#ifdef CONFIG_OF
        probe_zlg72xx->id_tree = of_match_node(zlg72XX_of,probe_zlg72xx->client->dev.of_node);
        if(!probe_zlg72xx->id_tree)
                return -ENODEV;
#endif
#ifndef CONFIG_OF
        probe_zlg72xx->id = id;
```

probe（探查）函数（设备初始化）

移除设备文件的函数

```
static int zlg72XX_remove(struct i2c_client *client)
{
        struct zlg72XX *remove_zlg72xx;
        remove_zlg72xx = (struct zlg72XX*)i2c_get_clientdata(client);
        cancel_delayed_work_sync(&remove_zlg72xx->work);
        unregister_zlg72XX_device(remove_zlg72xx);
        i2c_set_clientdata(client, NULL);
#ifdef CONFIG_ZLG72XX_INPUT_DEVICE
        input_unregister_device(remove_zlg72xx->key);
        input_free_device(remove_zlg72xx->key);
#endif
        kfree(remove_zlg72xx);
        return 0;
}
```

I2c设备结构体

```
static const struct i2c_device_id zlg72XX_id[] = {
        {ZLG7290_NAME, 0},
        {ZLG72128_NAME, 1},
        {}
};

MODULE_DEVICE_TABLE(i2c, zlg72XX_id);
```

i2c驱动结构体

```
static struct i2c_driver  zlg72XX_driver= {
        .probe      = zlg72XX_probe,
        .remove     = zlg72XX_remove,
        .id_table   = zlg72XX_id,
        .driver     = {
                .name           = ZLG72XX_NAME,
                .owner          = THIS_MODULE,
                .of_match_table = of_match_ptr(zlg72XX_of),
        },
};
```

初始化函数（增加i2c总线）

```
static int __init zlg72XX_init(void)
{
        return i2c_add_driver(&zlg72XX_driver);
}
```

退出函数（删除i2c总线）

```
static void __exit zlg72XX_exit(void)
{
        i2c_del_driver(&zlg72XX_driver);
}
```

```
module_init(zlg72XX_init);
module_exit(zlg72XX_exit);

MODULE_AUTHOR("farsight");
MODULE_DESCRIPTION("zlg72XX driver");
MODULE_LICENSE("GPL");
```

farsight_adc.c - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

10、ADC信号采集设备驱动程序（使用设备树文件）

```
//-----------------------------------
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/interrupt.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/clk.h>
#include <linux/completion.h>
#include <linux/delay.h>
#include <linux/reset.h>
#include <linux/slab.h>
#include <linux/regulator/consumer.h>
#include <linux/iio/iio.h>
#include <asm/uaccess.h>

#define MQ3    _IO('A',0)        //酒精
#define MQ5    _IO('A',1)        //气体
#define FLAME _IO('A',2)        //火焰
#define LDR    _IO('A',3)        //光敏
#define RP     _IO('A',4)        //电位器

#define DEVICE_NAME        "adc_ctl"
#define DRIVER_NAME        "adc_ctl"

#define SARADC_DATA                0x00
#define SARADC_STAS                0x04
#define SARADC_CTRL                0x08
#define SARADC_DLY_PU_SOC          0x0C

#define CTRL_IRQ_ENABLE            BIT(5)
#define CTRL_POWER_UP              BIT(3)
#define CTRL_IRQ_DISABLE           ~BIT(5)
#define CTRL_POWER_DOWN            ~BIT(3)
```

```
struct farsight_saradc_data {
        int num_bits;
        unsigned long clk_rate;
};

struct farsight_saradc {
        void __iomem *regs;
        struct clk *pclk;
        struct clk *clk;
        const struct farsight_saradc_data *data;
        struct cdev cdev;
        struct class* class;
        struct device* dev;
        dev_t devnum;
};

struct farsight_saradc *info = NULL;

static const struct farsight_saradc_data rk3399_saradc_data = {
        .num_bits = 10,
        .clk_rate = 1000000,
};

static int farsight_saradc_open(struct inode *inode, struct file *file)
{
        return 0;
}

static int farsight_saradc_close(struct inode *inode, struct file *file)
{
        return 0;
}
```

open ()函数

close()函数

ioctl ()函数

```
/**********************************/
/*   选择ADC通道              */
/*   气体传感器使用SARADC通道0   */
/*   电位器使用SARADC通道2      */
/*   火焰传感器使用SARADC通道3   */
/*   光敏传感器使用SARADC通道3   */
/*   酒精传感器使用SARADC通道4   */
/**********************************/
static long farsight_saradc_ioctl(struct file *filep, unsigned int cmd, unsigned long arg)
{
        unsigned int adc_data = 0;
        writel((readl(info->regs + SARADC_CTRL) & CTRL_IRQ_DISABLE & CTRL_POWER_DOWN), (info->regs + SARADC_CTRL));        //停止ADC转换 - 禁用中断
        writel(0x8, (info->regs + SARADC_DLY_PU_SOC));
        switch(cmd)
        {
                case MQ5:
                        writel(((readl(info->regs + SARADC_CTRL) & ~(0x7<<0)) | CTRL_POWER_UP), (info->regs + SARADC_CTRL));                    //ADC采集通道选择0通道
                        break;
                case RP:
                        writel((((readl(info->regs + SARADC_CTRL) & ~(0x7<<0)) | 0x2) | CTRL_POWER_UP), (info->regs + SARADC_CTRL));            //ADC采集通道选择2通道
                        break;
                case FLAME:
                        writel((((readl(info->regs + SARADC_CTRL) & ~(0x7<<0)) | 0x3) | CTRL_POWER_UP), (info->regs + SARADC_CTRL));            //ADC采集通道选择3通道
                        break;
                case LDR:
                        writel((((readl(info->regs + SARADC_CTRL) & ~(0x7<<0)) | 0x3) | CTRL_POWER_UP), (info->regs + SARADC_CTRL));            //ADC采集通道选择3通道
                        break;
                case MQ3:
                        writel((((readl(info->regs + SARADC_CTRL) & ~(0x7<<0)) | 0x4) | CTRL_POWER_UP), (info->regs + SARADC_CTRL));            //ADC采集通道选择4通道
                        break;
                default:
                        return -EINVAL;
        }
        msleep(1);
        while((readl((info->regs + SARADC_STAS)) & 0x1) == 0x1);                //判断转换是否停止
        adc_data = readl((info->regs + SARADC_DATA));                           //获取数据
        if(copy_to_user((unsigned int*)arg,&adc_data,sizeof(adc_data)))
                return -EFAULT;
        return 0;
}
```

file_operations 结构体

```
static struct file_operations farsight_saradc_ops = {
        .owner          = THIS_MODULE,
        .open           = farsight_saradc_open,
        .release        = farsight_saradc_close,
        .unlocked_ioctl = farsight_saradc_ioctl,
};

static const struct of_device_id farsight_saradc_match[] = {
        {.compatible = "rockchip,rk3399-saradc",
        .data = &rk3399_saradc_data,},
        {},
};

MODULE_DEVICE_TABLE(of, farsight_saradc_match);
```

probe（探查）函数（设备初始化）

```
static int farsight_saradc_probe(struct platform_device *pdev)
{
        struct device_node *np = pdev->dev.of_node;
        struct resource    *mem;
        const struct of_device_id *match;
        int ret;
        if(!(info = kmalloc(sizeof(struct farsight_saradc),GFP_KERNEL)))
                return -ENOMEM;

/*----------------------------字符设备创建----------------------------------*/
        cdev_init(&info->cdev,&farsight_saradc_ops);                    //初始化字符设备对象
        info->cdev.owner = THIS_MODULE;                                 //设置字符设备所属模块
        ret = alloc_chrdev_region(&info->devnum,0,1,"saradcs_reg_device");  //申请设备号
        if(ret<0)
        {
                goto out_err_0;
        }
        ret = cdev_add(&info->cdev,info->devnum, 1);                    //添加字符设备
        if (ret < 0)
        {
                goto out_err_1;
        }
        info->class = class_create(THIS_MODULE,"saradcs_reg_class");    //创建character类
        if (IS_ERR(info->class))
        {
                ret = PTR_ERR(info->class);
                goto out_err_2;
        }
        info->dev = device_create(info->class,NULL,info->devnum,NULL,"adc_ctrl");  //创建字符设备节点
        if (IS_ERR(info->dev))
        {
                ret = PTR_ERR(info->dev);
```

移除设备文件的函数

```
static int farsight_saradc_remove(struct platform_device *pdev)
{
	clk_disable_unprepare(info->clk);
	clk_disable_unprepare(info->pclk);
	device_del(info->dev);
	class_destroy(info->class);
	cdev_del(&info->cdev);
	kfree(info);
	unregister_chrdev_region(info->devnum,1);
	return 0;
}
```

暂停设备文件的函数

```
#ifdef CONFIG_PM_SLEEP
static int farsight_saradc_suspend(struct device *dev)
{
	clk_disable_unprepare(info->clk);
	clk_disable_unprepare(info->pclk);
	return 0;
}
```

设备概要函数

```
static int farsight_saradc_resume(struct device *dev)
{
	int ret;
	ret = clk_prepare_enable(info->pclk);
	if (ret)
		return ret;
	ret = clk_prepare_enable(info->clk);
	if (ret)
		return ret;
	return ret;
}
#endif

static SIMPLE_DEV_PM_OPS(farsight_saradc_pm_ops, farsight_saradc_suspend, farsight_saradc_resume);

static struct platform_driver farsight_saradc_driver = {
	.probe		= farsight_saradc_probe,
	.remove		= farsight_saradc_remove,
	.driver		= {
		.name		= "farsight-saradc",
		.of_match_table = farsight_saradc_match,
		.pm		= &farsight_saradc_pm_ops,
	},
};
```

```
static int __init farsight_saradc_init(void)
{
        return platform_driver_register(&farsight_saradc_driver);
}

static void __exit farsight_saradc_exit(void)
{
        platform_driver_unregister(&farsight_saradc_driver);
}

module_init(farsight_saradc_init);
module_exit(farsight_saradc_exit);

MODULE_AUTHOR("Heiko Stuebner <heiko@sntech.de>");
MODULE_DESCRIPTION("farsight SARADC driver");
MODULE_LICENSE("GPL v2");
```

初始化函数（注册platform总线）

退出函数（注销platform总线）

小结

- 主要介绍嵌入式系统中字符设备驱动的开发。
- 字符设备驱动的基本框架和原理。
- 字符设备驱动程序的编写流程、关键的数据结构和驱动程序的主要组成部分。
- GPIO驱动。
- 串行总线驱动。
- I2C总线驱动。

进一步探索

- 阐述嵌入式系统中字符设备驱动的地位和主要作用。
- 驱动的加载使用主要有哪些方法？它们的差别是什么？

Thanks