《嵌入式系统》

（第十讲）

厦门大学信息学院软件工程系 曾文华

2023年11月7日

第10章　字符设备和驱动程序设计

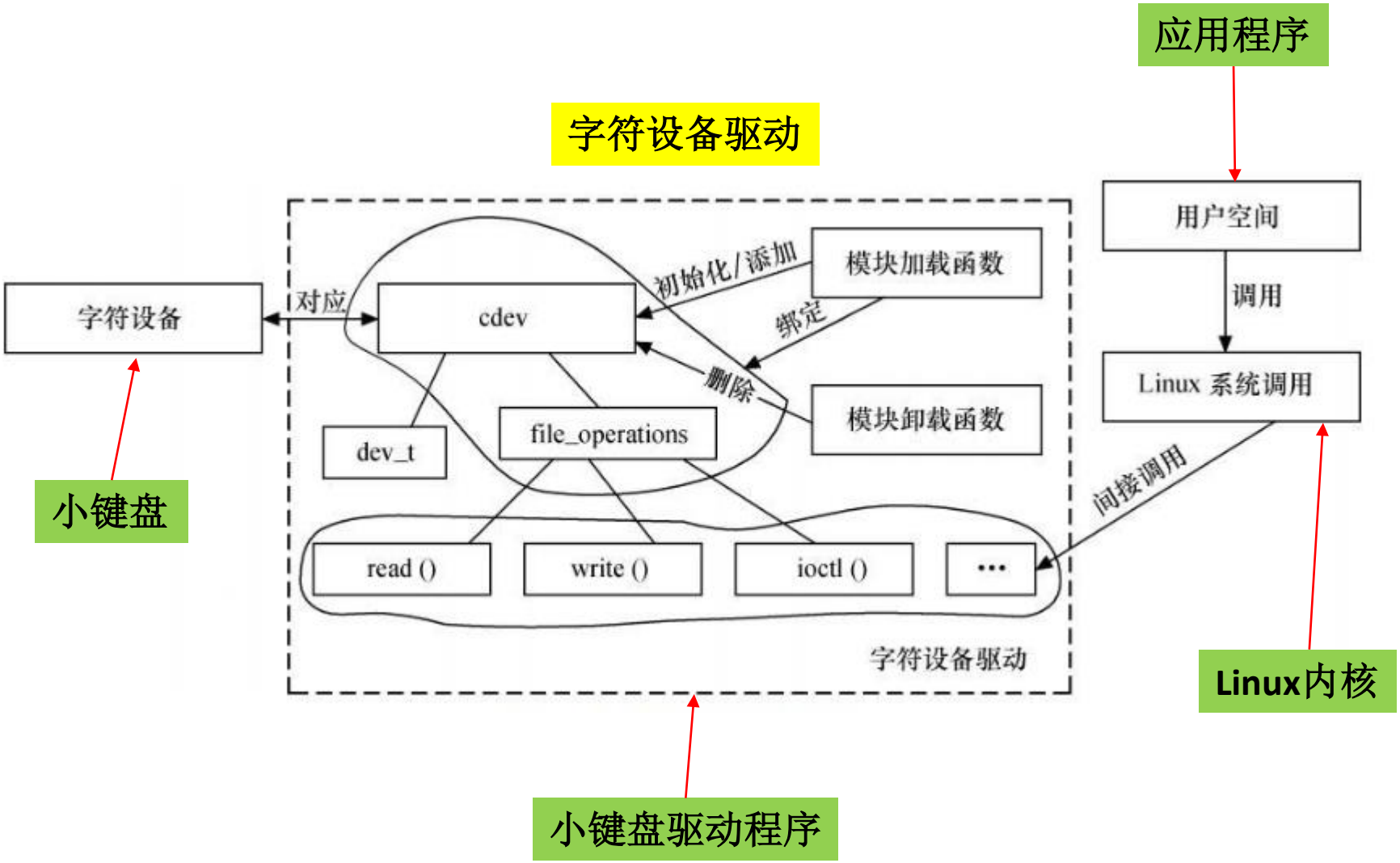
- 10.1　字符设备驱动框架
- 10.2　字符设备驱动开发
- 10.3　GPIO驱动概述
- 10.4　串口总线概述
- 10.5　字符设备驱动程序示例

- 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

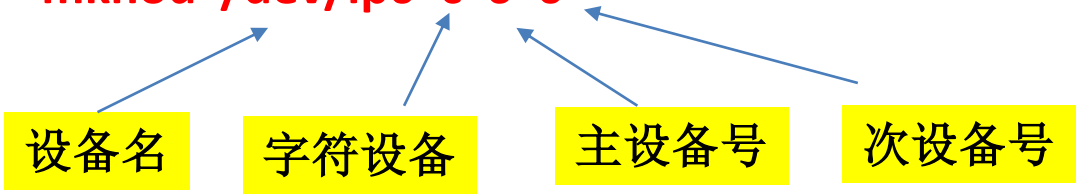
```
struct cdev {
        struct kobject kobj;
        struct module *owner;
        const struct file_operations *ops;
        struct list_head list;
        dev_t dev;
        unsigned int count;
};
```

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



查看实验箱的主设备号和次设备号（ls -l /dev）

```
root@imx6dlsabresd:~# ls  -l  /dev
total 0
crw------- 1 root root     30,   0 Jan  1  1970 UART485
crw------- 1 root root     10, 235 Jan  1  1970 autofs
drwxr-xr-x 2 root root       640 Jan  1  1970 block
drwxr-xr-x 3 root root        60 Jan  1  1970 bus
drwxr-xr-x 2 root root      2700 Sep 29 08:03 char
crw------- 1 root root      5,   1 Sep 29 08:03 console
crw------- 1 root root     10,  61 Jan  1  1970 cpu_dma_latency
drwxr-xr-x 5 root root       100 Jan  1  1970 disk
drwxr-xr-x 2 root root        60 Jan  1  1970 dri
crw-rw---- 1 root video    29,   0 Jan  1  1970 fb0
crw-rw---- 1 root video    29,   1 Jan  1  1970 fb1
lrwxrwxrwx 1 root root        13 Jan  1  1970 fd -> /proc/self/fd
crw-rw-rw- 1 root root      1,   7 Jan  1  1970 full
crw-rw-rw- 1 root root     10, 229 Jan  1  1970 fuse
crw-rw---- 1 root video   199,   0 Jan  1  1970 galcore
crw------- 1 root root     10, 183 Jan  1  1970 hwrng
crw------- 1 root root     89,   0 Jan  1  1970 i2c-0
crw------- 1 root root     89,   1 Jan  1  1970 i2c-1
crw------- 1 root root     89,   2 Jan  1  1970 i2c-2
prw------- 1 root root           0 Jan  1  1970 initctl
```

实验箱字符设备的主设备号（cat /proc/devices）

```
root@imx6dlsabresd:~# 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
 30 UART485
 81 video4linux
 89 i2c
 90 mtd
116 alsa
128 ptm
136 pts
180 usb
188 ttyUSB
```

– 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

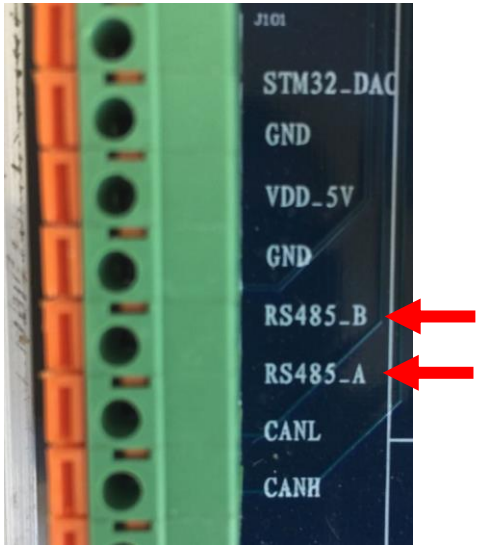
- GPIO：General Purpose Input/Output，通用输入输出，可以对GPIO进行编程，将GPIO的每一个引脚设为输入或输出，因此GPIO也称为通用可编程接口。

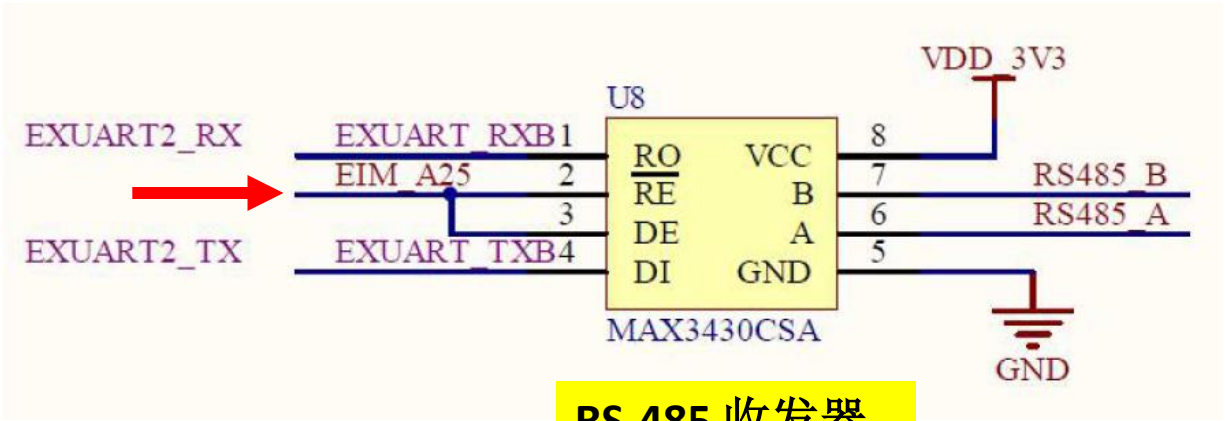
- GPIO接口至少要有两个寄存器：
  - 控制寄存器
  - 数据寄存器

- 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

RS-485实验（1个GPIO）





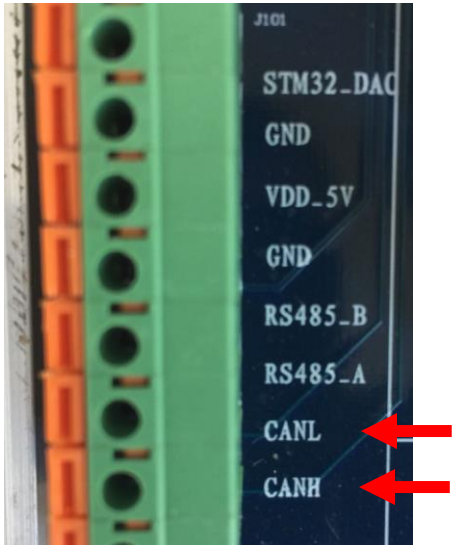
RS-485 收发器

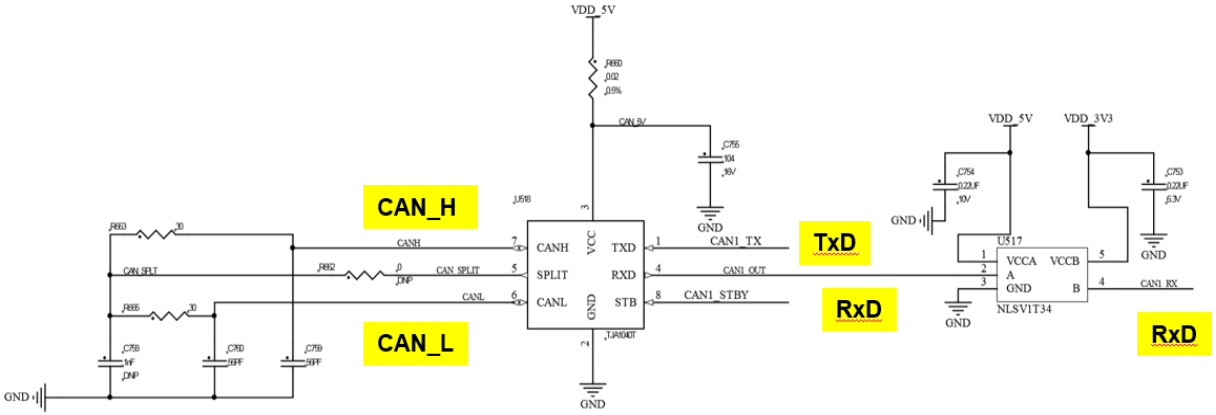
```
pinctrl_gpio_uart485_ctrl: gpio_uart485_ctrlgrp
{
        fsl,pins = <
            MX6QDL_PAD_EIM_A25__GPIO5_IO02 0x80000000
        >;
};
```

GPIO5_IO02

EIM_A25

CAN总线实验（3个GPIO）






```
pinctrl_flexcan1:flexcan1grp
{
        fsl,pins = <
                MX6QDL_PAD_GPIO_7__FLEXCAN1_TX  0x80000000
                MX6QDL_PAD_GPIO_8__FLEXCAN1_RX  0x80000000
                MX6QDL_PAD_GPIO_19__GPIO4_IO05  0x80000000
        >;
};
```

LED灯实验（4个GPIO）

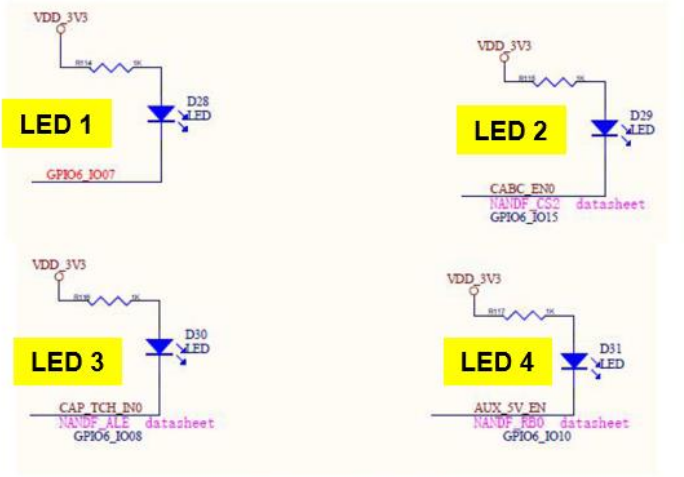


图 3.2.1 LED 灯电路原理图

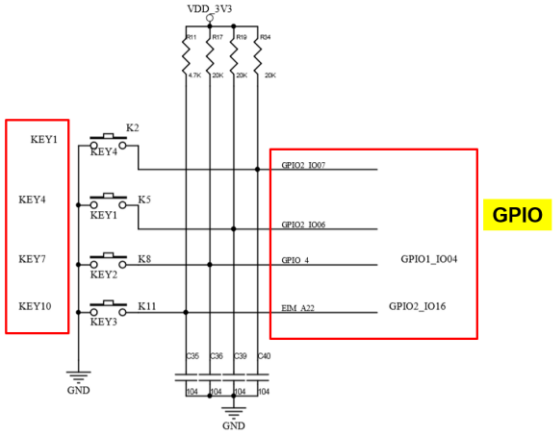
LED 1 — D28：GPIO6_IO07

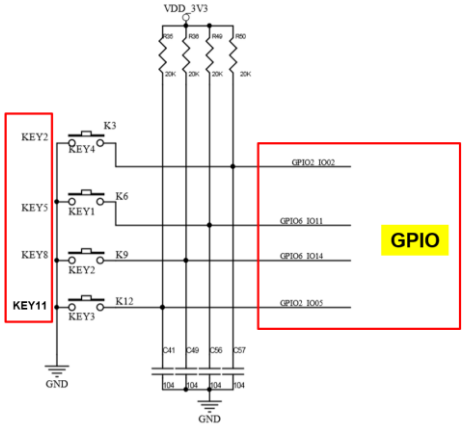
LED 2 — D29：GPIO6_IO15

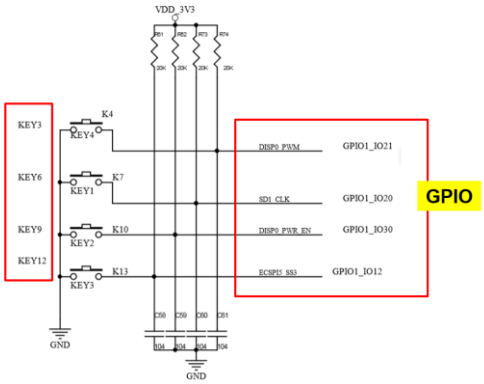
LED 3 — D30：GPIO6_IO08

LED 4 — D31：GPIO6_IO10

小键盘实验（12个GPIO）



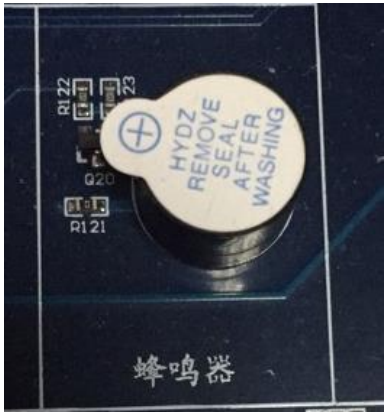


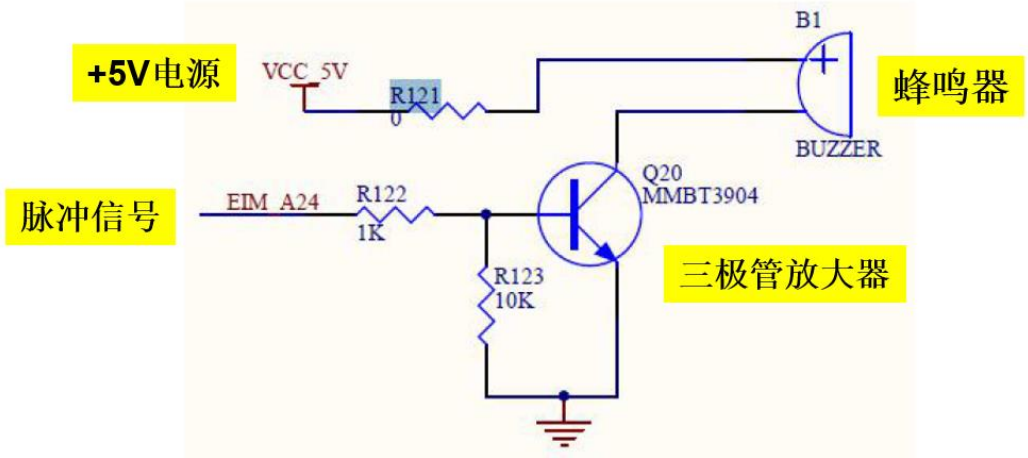


MX6QDL_PAD_NANDF_D7__GPIO2_IO07 0x80000000
MX6QDL_PAD_NANDF_D2__GPIO2_IO02 0x80000000
MX6QDL_PAD_SD1_DAT3__GPIO1_IO21 0x80000000
MX6QDL_PAD_NANDF_D6__GPIO2_IO06 0x80000000
MX6QDL_PAD_NANDF_CS0__GPIO6_IO11 0x80000000
MX6QDL_PAD_SD1_CLK__GPIO1_IO20 0x80000000
MX6QDL_PAD_GPIO_4__GPIO1_IO04 0x80000000
MX6QDL_PAD_NANDF_CS1__GPIO6_IO14 0x80000000
MX6QDL_PAD_ENET_TXD0__GPIO1_IO30 0x80000000
MX6QDL_PAD_EIM_A22__GPIO2_IO16 0x80000000
MX6QDL_PAD_NANDF_D5__GPIO2_IO05 0x80000000
MX6QDL_PAD_SD2_DAT3__GPIO1_IO12 0x80000000

IMX6处理器的GPIO

蜂鸣器实验（1个GPIO）





IMX6处理器的GPIO
（General Purpose Input Output，通用输入/输出）

Table 29-5. GPIO5 External Signals

Signal	Description	Pad	Mode	Direction
GPIO5_IO00	-	EIM_WAIT	ALT5	IO
GPIO5_IO02	-	EIM_A25	ALT5	IO
GPIO5_IO04 ←		EIM_A24 ←	ALT5	IO

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　I²C总线**

  - I²C（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。

  - I²C总线有两根接口线：
    - 数据线：SDA
    - 时钟线：SCK，或SCL

  - I²C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

  - I²C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/uptech/fsl-6dl-source/kernel-3.14.28/drivers/char目录
  - RS-485驱动程序： uptech485.c
  - LED灯驱动程序： imx6-leds.c

RS-485驱动程序

uptech485.c

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/device.h>
#include <asm/system.h>
#include <asm/uaccess.h>
#include <linux/platform_device.h>
#include <asm/irq.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_gpio.h>

#define DRVNAME “UART485“                    //设备名
#define UART485_MAJOR 30                     //主设备号
#define UART485_MINOR 0                      //次设备号
#define UART485_TX 1
#define UART485_RX 0
```

```
static unsigned int gpio_ctrl;
static struct class *uart485_class;
static struct cdev uart485cdev;

static int Uart485PowerOpen(struct inode *inode, struct file *filp)
{
        return 0;
}

static ssize_t Uart485PowerWrite(struct file *filp,char __user *buf,size_t count,loff_t *ppos)
{
        return 0;
}

static ssize_t Uart485PowerRead(struct file *filp,char __user *buf,size_t count,loff_t *ppos)
{
        return 0;
}
```

打开485设备

485设备写操作

485设备读操作

ioctl函数

```
static int Uart485PowerIoctl(struct file *filp,unsigned int cmd,unsigned long arg)
{
        gpio_request(gpio_ctrl,"uart485Ctrl");

        if(cmd == UART485_TX)                                   //RS485发送
        {
                gpio_direction_output(gpio_ctrl,1);

        }
        else if(cmd == UART485_RX)                              //RS485接收
        {
                gpio_direction_output(gpio_ctrl,0);
        }

        gpio_free(gpio_ctrl);
        return 0;
}
```

file_operations结构体

```
static const struct file_operations uart485_fops = {
        .owner = THIS_MODULE,
        .write = Uart485PowerWrite,
        .read = Uart485PowerRead,
        .open = Uart485PowerOpen,
        .unlocked_ioctl = Uart485PowerIoctl,
};
```

RS-485初始化

```
static int Uart485Init(void)
{
        dev_t devt;
        int retval;

        devt = MKDEV(UART485_MAJOR,UART485_MINOR);
        retval = register_chrdev_region(devt,1,DRVNAME);

        if(retval>0)
                return retval;

        cdev_init(&uart485cdev,&uart485_fops);
        retval = cdev_add(&uart485cdev,devt,1);

        if(retval)
                goto error;
```

注册字符设备

```
        uart485_class = class_create(THIS_MODULE,"UART485");
        if (IS_ERR(uart485_class)) {
                printk(KERN_ERR "Error creating raw class.\n");
                cdev_del(&uart485cdev);
                goto error;
        }

        device_create(uart485_class, NULL, MKDEV(UART485_MAJOR,UART485_MINOR), NULL,DRVNAME);
        gpio_request(gpio_ctrl,"uart485Ctrl");
        gpio_direction_output(gpio_ctrl,0);
        gpio_free(gpio_ctrl);
        return 0;

error:
        unregister_chrdev_region(devt, 1);
        return retval;

}
```

```
static void Uart485Exit(void)
{
        device_destroy(uart485_class, MKDEV(UART485_MAJOR,UART485_MINOR));
        class_destroy(uart485_class);
        cdev_del(&uart485cdev);
        unregister_chrdev_region(MKDEV(UART485_MAJOR,UART485_MINOR), 1);
}
```

RS-485退出

注销字符设备

探测函数

```
static int gpio_uart485_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct device_node *of_node;
        of_node = dev->of_node;

        if (!of_node) {
                return -ENODEV;
        }

        gpio_ctrl = of_get_named_gpio(of_node,"uartctrl",0);

        if(!gpio_is_valid(gpio_ctrl))
        {
                return -ENODEV;
        }

        printk("\n\n\nkzkuan___%s\n\n\n",__func__);
        Uart485Init();
        return 0;
}
```

RS-485初始化

移除设备

```
static int gpio_uart485_remove(struct platform_device *pdev)
{
        Uart485Exit();
        return 0;
}
```

RS-485退出

```
static struct of_device_id gpio_uart485_of_match[] =
{
        { .compatible = "fsl,gpio-uart485-ctrl", },
        { },
};

MODULE_DEVICE_TABLE(of, gpio_uart485_of_match);

static struct platform_driver gpio_uart485_device_driver =
{
        .probe          = gpio_uart485_probe,
        .remove         = gpio_uart485_remove,
        .driver         = {
                .name   = "gpio-uart485-ctrl",
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(gpio_uart485_of_match),
        }
};
```

设备ID结构体

平台驱动结构体

```
static int __init gpio_uart485_init(void)
{
        printk("\n\n\nkzkuan___%s\n\n\n",__func__);
        return platform_driver_register(&gpio_uart485_device_driver);
}

static void __exit gpio_uart485_exit(void)
{
        printk("\n\n\nkzkuan___%s\n\n\n",__func__);
        platform_driver_unregister(&gpio_uart485_device_driver);
}


module_init(gpio_uart485_init);
module_exit(gpio_uart485_init);


MODULE_AUTHOR("uptech kzkuan");
MODULE_DESCRIPTION("uart 485 control");
MODULE_LICENSE("GSL");
```

字符设备模块初始化

字符设备模块退出

LED灯驱动程序

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/cdev.h>
#include <linux/platform_device.h>
#include <asm/irq.h>
#include <linux/of.h>
#include <linux/of_device.h>
#include <linux/of_gpio.h>
```

imx6-leds.c

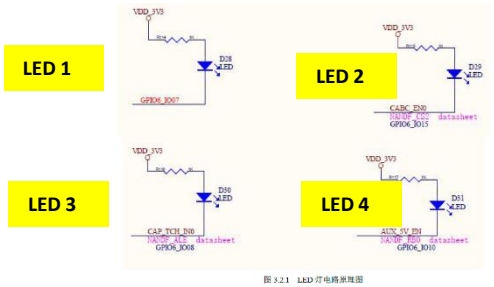
```
MODULE_LICENSE("GPL");

#define DEVICE_NAME     “ledtest”                    //设备名
#define DEVICE_MAJOR 231                             //主设备号
#define DEVICE_MINOR  0                              //次设备号

struct cdev *mycdev;
struct class *myclass;
dev_t devno;

static unsigned int led_table [4] = {};              //4个LED灯
```

ioctl函数

ioctl(fd,*argv[2]-'0',*argv[3]-'0');

```
static long uptech_leds_ioctl( struct file *file,  unsigned int cmd,
unsigned long arg)
{
      switch(cmd) {
          case 1:
                    if (arg < 0 || arg > 4) {
                              return -EINVAL;
                    }
                    gpio_request(led_table[arg],"ledCtrl");
                    gpio_direction_output(led_table[arg],0);
                    gpio_free(led_table[arg]);
                    break;
```

cmd=1，则LED灯亮



cmd=0，则LED灯灭

```
        case 0:
                if (arg < 0 || arg > 4) {
                        return -EINVAL;
                }
                gpio_request(led_table[arg],"ledCtrl");
                gpio_direction_output(led_table[arg],1);
                gpio_free(led_table[arg]);
                break;

        default:
                return -EINVAL;
    }

    return 0;
}
```

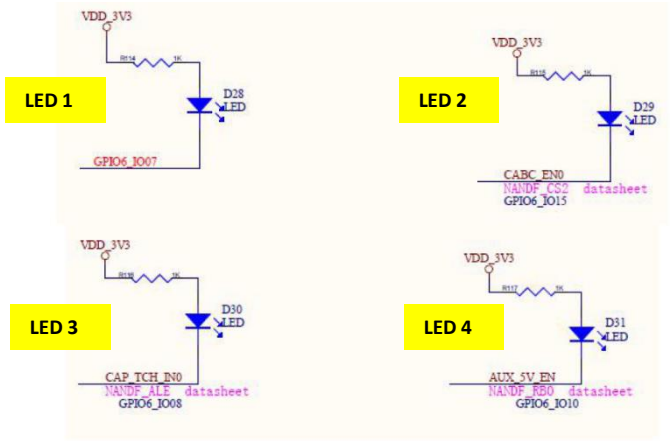


图 3.2.1 LED 灯电路原理图

file_operations结构体

```
static struct file_operations uptech_leds_fops = {
        .owner                  =   THIS_MODULE,
        .unlocked_ioctl         =   uptech_leds_ioctl,
};
```

LED设备初始化

```
static int uptech_leds_init(void)
{
    int err;

    devno = MKDEV(DEVICE_MAJOR, DEVICE_MINOR);
    mycdev = cdev_alloc();
    cdev_init(mycdev, &uptech_leds_fops);
    err = cdev_add(mycdev, devno, 1);

    if (err != 0)
        printk("Exynos4412 leds device register failed!\n");
```

```
        myclass = class_create(THIS_MODULE, "ledtest");

        if(IS_ERR(myclass)) {
                printk("Err: failed in creating class.\n");
                return -1;
        }

        device_create(myclass,NULL, MKDEV(DEVICE_MAJOR,DEVICE_MINOR), NULL,
DEVICE_NAME);

        printk(DEVICE_NAME "leds initialized\n");

        return 0;
}
```

探测函数

```
static int gpio_leds_probe(struct platform_device *pdev)
{
        unsigned int i;

        struct device *dev = &pdev->dev;
        struct device_node *of_node;
        of_node = dev->of_node;

        if (!of_node) {
                return -ENODEV;
        }
```

```
        led_table[0] = of_get_named_gpio(of_node,"gpio0",0);
        led_table[1] = of_get_named_gpio(of_node,"gpio1",0);
        led_table[2] = of_get_named_gpio(of_node,"gpio2",0);
        led_table[3] = of_get_named_gpio(of_node,"gpio3",0);

        if(!gpio_is_valid(led_table[0])||!gpio_is_valid(led_table[1])||!gpio_is_valid(led_tabl
e[2])||!gpio_is_valid(led_table[3]))
        {
                return -ENODEV;
        }

        for (i = 0; i < 5; i++) {
                gpio_request(led_table[i],"ledCtrl");
                gpio_direction_output(led_table[i],1);
                gpio_free(led_table[i]);
        }

        printk("\n\n\nkzkuan___%s\n\n\n",__func__);
        uptech_leds_init();
        return 0;
}
```

LED设备退出

```
static void uptech_leds_exit(void)
{
	cdev_del(mycdev);
	device_destroy(myclass,devno);
	class_destroy(myclass);
}
```

移除设备

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

设备ID结构体

```
static struct of_device_id gpio_leds_of_match[] = {
        { .compatible = "fsl,gpio-leds-test", },
        { },
};

MODULE_DEVICE_TABLE(of, gpio_leds_of_match);
```

设备驱动结构体

```
static struct platform_driver gpio_leds_device_driver = {
        .probe          = gpio_leds_probe,
        .remove         = gpio_leds_remove,
        .driver         = {
                .name   = "gpio-leds-test",
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(gpio_leds_of_match),
        }
};
```

字符设备模块初始化

```
static int __init gpio_leds_init(void)
{
    printk("\n\n\nkzkuan___%s\n\n\n",__func__);
    return platform_driver_register(&gpio_leds_device_driver);
}
```

字符设备模块退出

```
static void __exit gpio_leds_exit(void)
{
    printk("\n\n\nkzkuan___%s\n\n\n",__func__);
    platform_driver_unregister(&gpio_leds_device_driver);
}


module_init(gpio_leds_init);
module_exit(gpio_leds_exit);
```

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？

第5次作业

- 1、请分析“RS-485设备驱动程序”（位于Ubuntu的“/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/char/uptech485.c”）
- 2、请分析“LED灯设备驱动程序”（位于Ubuntu的“/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/char/imx6-leds.c”）
- 3、请分析“蜂鸣器设备驱动程序”（位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/misc/gpio-beeper.c）

- 要求：
  ① 请每个同学用PPT回答上述3个问题。

  ② 2023年11月14日上课时，会随机抽取3位同学，到讲台上进行汇报（用PPT汇报）；第一位同学汇报第一个问题，第二位同学汇报第二个问题，第三位同学汇报第三个问题；每个同学的汇报时间控制在5分钟左右。

  ③ 所有同学都要将汇报的PPT作为第5次作业上传到FTP上，截止日期：2023年11月13日晚上24点。

- 有兴趣的同学，请分析：
  - 1、CAN总线的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/net/can/flexcan.c
  - 2、小键盘的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/keyboard/gpio_keys.c
  - 3、LCD的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/video/console/fbcon.c、位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/include/linux/fb.h、位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/video/fbdev/core/fbmem.c
  - 4、摄像头的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/media/platform/mxc/capture目录下ov5640*.c文件
  - 5、陀螺仪的驱动程序：位于Ubuntu的/home/uptech/fsl-6dl-source/kernel-4.9.88/drivers/input/misc/mpu6880/mpu6880.c

Thanks