weiqing的专栏

- 目录视图
- 摘要视图
- 订阅

Linux下的Backlight子系统（一）

分类： linux驱动 子系统 Mini2440 2013-01-17 09:24 3032人阅读 评论(1) 收藏 举报

原创作者：南京邮电大学 通信与信息系统专业 研二 魏清

一．Backlight背光子系统概述

我们的LCD屏常常需要一个背光，调节LCD屏背光的亮度，这里所说的背光不是仅仅亮和不亮两种，而是根据用户的需求，背光亮度是可以任意调节。Linux内核中有一个backlight背光子系统，该系统就是为满足用户这种需求设计的，用户只要根据自己的LCD背光电路中PWM输出引脚，对内核backlight子系统代码进行相应的配置，就可以实现LCD的背光。

LCD的背光原理主要是由核心板的一根引脚控制背光电源，一根PWM引脚控制背光亮度组成，应用程序可以通过改变PWM的频率达到改变背光亮度的目的。

我们这里主要讲解基于backlight子系统的蜂鸣器驱动，其实简单的使得蜂蜜器发声的驱动很简单，这里只是把蜂鸣器作为一种设备，而且这种设备原理类似背光的原理，都是基于pwm的，而我们的终极目的是使用backlight背光子系统。综上所述，backlight子系统是基于pwm核心的一种驱动接口，如果你使用的一种设备也是基于pwm的，并且需要用户可以调节pwm的频率以达到诸如改变背光亮度，改变蜂鸣器频率的效果，那么你可以使用这个backlight背光子系统。

二．PWM核心驱动

我们先讲解下PWM核心

先熟悉下pwm核心代码在/arch/arm/plat-s3c/pwm.c

查看/arch/arm/plat-s3c/Makefile

obj-$(CONFIG_HAVE_PWM) += pwm.o

查看/arch/arm/plat-s3c/Konfig，发现同目录的Konfig中无对应HAVE_PWM选项

查看/arch/arm/plat-s3c24xx/Konfig

config S3C24XX_PWM

bool "PWM device support"

```
    select HAVE_PWM
    help
      Support for exporting the PWM timer blocks via the pwm device
      system.
```

所以配置内核make menuconfig时，需要选中这一项。

好了，我们看看pwm.c，它是pwm核心驱动，该驱动把设备和驱动没有分离开来，都写在了这个pwm.c中，我们先看看pwm.c中的驱动部分

```
static int __init pwm_init(void)
{
    int ret;
    clk_scaler[0] = clk_get(NULL, "pwm-scaler0");  //获取0号时钟
    clk_scaler[1] = clk_get(NULL, "pwm-scaler1");  //获取1号时钟
    if (IS_ERR(clk_scaler[0]) || IS_ERR(clk_scaler[1])) {
        printk(KERN_ERR "%s: failed to get scaler clocks\n", __func__);
        return -EINVAL;
    }
    ret = platform_driver_register(&s3c_pwm_driver);  //注册pwm驱动
    if (ret)
        printk(KERN_ERR "%s: failed to add pwm driver\n", __func__);
    return ret;
}
```

跟踪下s3c_pwm_driver的定义

```
static struct platform_driver s3c_pwm_driver = {
    .driver         = {
        .name     = "s3c24xx-pwm",  //驱动名
        .owner    = THIS_MODULE,
    },
    .probe          = s3c_pwm_probe,  //探测函数
    .remove         = __devexit_p(s3c_pwm_remove),
};
```

我们看看探测函数s3c_pwm_probe

```
static int s3c_pwm_probe(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct pwm_device *pwm;
    unsigned long flags;
    unsigned long tcon;
    unsigned int id = pdev->id;
    int ret;
    if (id == 4) {
        dev_err(dev, "TIMER4 is currently not supported\n");
        return -ENXIO;
    }
    pwm = kzalloc(sizeof(struct pwm_device), GFP_KERNEL); //分配pwm设备空间
    if (pwm == NULL) {
        dev_err(dev, "failed to allocate pwm_device\n");
        return -ENOMEM;
    }
    pwm->pdev = pdev;
    pwm->pwm_id = id;
    pwm->tcon_base = id == 0 ? 0 : (id * 4) + 4;  //计算TCON中控制哪个定时器
    pwm->clk = clk_get(dev, "pwm-tin");  //获取预分频后的时钟
    if (IS_ERR(pwm->clk)) {
        dev_err(dev, "failed to get pwm tin clk\n");
        ret = PTR_ERR(pwm->clk);
        goto err_alloc;
    }
    pwm->clk_div = clk_get(dev, "pwm-tdiv");
    if (IS_ERR(pwm->clk_div)) {    //获取二次分频后的时钟
        dev_err(dev, "failed to get pwm tdiv clk\n");
```

```
            ret = PTR_ERR(pwm->clk_div);
            goto err_clk_tin;
        }
        local_irq_save(flags);
        tcon = __raw_readl(S3C2410_TCON);
        tcon |= pwm_tcon_invert(pwm);  //信号反转输出
        __raw_writel(tcon, S3C2410_TCON);
        local_irq_restore(flags);
        ret = pwm_register(pwm);       //注册pwm设备
        if (ret) {
            dev_err(dev, "failed to register pwm\n");
            goto err_clk_tdiv;
        }
        pwm_dbg(pwm, "config bits %02x\n",
            (__raw_readl(S3C2410_TCON) >> pwm->tcon_base) & 0x0f);
        dev_info(dev, "tin at %lu, tdiv at %lu, tin=%sclk, base %d\n",
             clk_get_rate(pwm->clk),
             clk_get_rate(pwm->clk_div),
             pwm_is_tdiv(pwm) ? "div" : "ext", pwm->tcon_base);
        platform_set_drvdata(pdev, pwm);
        return 0;
    err_clk_tdiv:
        clk_put(pwm->clk_div);
    err_clk_tin:
        clk_put(pwm->clk);
    err_alloc:
        kfree(pwm);
        return ret;
    }
```

下面看看注册pwm设备的函数pwm_register

```
static LIST_HEAD(pwm_list);
static int pwm_register(struct pwm_device *pwm)
{
    pwm->duty_ns = -1;
    pwm->period_ns = -1;
    mutex_lock(&pwm_lock);
    list_add_tail(&pwm->list, &pwm_list); //把pwm设备挂到pwm_list链表上
    mutex_unlock(&pwm_lock);
    return 0;
}
```

剩下来，我们看看这个pwm.c给我们提供了哪些接口函数

```
struct pwm_device *pwm_request(int pwm_id, const char *label)
int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns)
int pwm_enable(struct pwm_device *pwm)
void pwm_free(struct pwm_device *pwm)
EXPORT_SYMBOL(pwm_request); //申请PWM设备
EXPORT_SYMBOL(pwm_config); //配置PWM设备，duty_ns为空占比，period_ns为周期
EXPORT_SYMBOL(pwm_enable); //启动Timer定时器
EXPORT_SYMBOL(pwm_disable); //关闭Timer定时器
```

上面这个函数，只要知道API，会调用就行了，在此，我分析下最难的一个配置PWM函数，这个函数主要是根据周期period_ns，计算TCNT，根据空占比duty_ns，计算TCMP，然后写入相应寄存器。

```
int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns)
{
    unsigned long tin_rate;
    unsigned long tin_ns;
    unsigned long period;
    unsigned long flags;
    unsigned long tcon;
    unsigned long tcnt;
    long tcmp;
    if (period_ns > NS_IN_HZ || duty_ns > NS_IN_HZ)
```

```
        return -ERANGE;
    if (duty_ns > period_ns)
        return -EINVAL;
    if (period_ns == pwm->period_ns &&
       duty_ns == pwm->duty_ns)
        return 0;
    tcmp = __raw_readl(S3C2410_TCMPB(pwm->pwm_id));
    tcnt = __raw_readl(S3C2410_TCNTB(pwm->pwm_id));
    period = NS_IN_HZ / period_ns; //计算周期
    pwm_dbg(pwm, "duty_ns=%d, period_ns=%d (%lu)\n",
        duty_ns, period_ns, period);
    if (pwm->period_ns != period_ns) {
        if (pwm_is_tdiv(pwm)) {
            tin_rate = pwm_calc_tin(pwm, period);
            clk_set_rate(pwm->clk_div, tin_rate);
        } else
            tin_rate = clk_get_rate(pwm->clk);
        pwm->period_ns = period_ns;
        pwm_dbg(pwm, "tin_rate=%lu\n", tin_rate);
        tin_ns = NS_IN_HZ / tin_rate;
        tcnt = period_ns / tin_ns;  //根据周期求TCNT，n=To/Ti
    } else
        tin_ns = NS_IN_HZ / clk_get_rate(pwm->clk);
    tcmp = duty_ns / tin_ns;   //根据空占比求TCMP
    tcmp = tcnt - tcmp;  //根据占空比求TCMP
    if (tcmp == tcnt)
        tcmp--;
    pwm_dbg(pwm, "tin_ns=%lu, tcmp=%ld/%lu\n", tin_ns, tcmp, tcnt);
    if (tcmp < 0)
        tcmp = 0;
```

```
    local_irq_save(flags);
    __raw_writel(tcmp, S3C2410_TCMPB(pwm->pwm_id)); //写入TCMP
    __raw_writel(tcnt, S3C2410_TCNTB(pwm->pwm_id)); //写入TCNT
    tcon = __raw_readl(S3C2410_TCON);
    tcon |= pwm_tcon_manulupdate(pwm);
    tcon |= pwm_tcon_autoreload(pwm); //自动加载
    __raw_writel(tcon, S3C2410_TCON);
    tcon &= ~pwm_tcon_manulupdate(pwm); //更新TCNT和TCMP
    __raw_writel(tcon, S3C2410_TCON);
    local_irq_restore(flags);
    return 0;
}
```

下面说说这个周期是怎么设计的

我们定时器的输出频率fi=PCLK/(prescaler value+1)/(divider value)，这个可以获得确定值

我们需要写入一个初值n给TCNT，这样就可以获得一个频率，为什么呢？

根据初值n=fi/fo，那么n=To/Ti

所以当用户给pwm_config函数传递一个周期period_ns，其实就是To=period_ns

这样根据前面公式n=To/Ti= period_ns/fi，然后将这个初值n写入TCNT就可以改变周期了

接着我再补充说明下pwm_config函数里代码注释关于自动加载怎么回事？

定时器工作原理其实是TCNT的值在时钟到来时，减一计数，每次减一完后，拿当前TCNT与TCMP比较，如果TCNT=TCMP，那么信号电平反向输出，然后TCNT继续减一计数，知道TCNT减到零后，如果有自动加载功能那么此时将由TCNTB把计数初值再次写给TCNTP，同时TCMPB把比较值给TCMP，这样就完成一次初值重装，然后继续进行计数。我们给这种加载模式起了个名字叫双缓冲机制，其中TCMPB和TCNTB就是Buffer缓存。

前面说pwm.c集驱动和设备于一体，那么下面我们看看设备相关的代码

```
#define TIMER_RESOURCE_SIZE (1)
#define TIMER_RESOURCE(_tmr, _irq)              \
    (struct resource [TIMER_RESOURCE_SIZE]) { \
        [0] = {                         \
```

```
                .start      = _irq,                 \
                .end = _irq,                    \
                .flags      = IORESOURCE_IRQ   \
            }                           \
        }
#define DEFINE_S3C_TIMER(_tmr_no, _irq)             \
        .name           = "s3c24xx-pwm",        \
        .id         = _tmr_no,                  \
        .num_resources      = TIMER_RESOURCE_SIZE,          \
        .resource = TIMER_RESOURCE(_tmr_no, _irq),      \
struct platform_device s3c_device_timer[] = {
        [0] = { DEFINE_S3C_TIMER(0, IRQ_TIMER0) },
        [1] = { DEFINE_S3C_TIMER(1, IRQ_TIMER1) },
        [2] = { DEFINE_S3C_TIMER(2, IRQ_TIMER2) },
        [3] = { DEFINE_S3C_TIMER(3, IRQ_TIMER3) },
        [4] = { DEFINE_S3C_TIMER(4, IRQ_TIMER4) },
};
```

上面的代码就是设备部分代码，其实就是五个定时器的资源，我们把目光放在DEFINE_S3C_TIMER宏上，你会发现其设备名是"s3c24xx-pwm"，而我们在pwm.c中定义的驱动名也是"s3c24xx-pwm"，这样如果我们把设备注册到内核，那么设备"s3c24xx-pwm"和驱动"s3c24xx-pwm"就会匹配成功。所以如果你用到定时器0，那么你只要在BSP中添加s3c_device_timer[0]就可以了。我们现在做的是蜂鸣器驱动，使用的是Timer0定时器，我们就在mini2440的BSP文件mach-mini2440.c中添加如下代码

```
static struct platform_device *mini2440_devices[] __initdata = {
        ……
        &s3c_device_timer[0],   //添加
};
```

这样我们就分析完pwm核心层的代码了。

顶

2

踩
0

猜你在找

查看评论
* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

个人资料

weiqing1981127
- 访问：110842次
- 积分：1645
- 等级：
- 排名：第13712名
- 原创：51篇
- 转载：14篇
- 译文：0篇
- 评论：33条

文章搜索

搜索

文章分类

- linux驱动(30)
- linux内核(16)
- 子系统(15)
- 总线(14)
- Mini2440(28)
- linux c(4)
- AT91SAM9G45(1)
- 其他(6)
- 硬件(1)
- 问题解决(7)
- 闲谈(1)

阅读排行

- Linux下的PCI总线驱动(5652)
- VFS: Cannot open root device "nfs" or unknown-block(0,255)错误解决(5431)

- Linux下的触摸屏驱动(5292)
- Linux下的LCD驱动(一)(4129)
- Linux下的Backlight子系统（二）(3575)
- Linux下的SPI总线驱动（三）(3481)
- Verifying Checksum ... Bad Data CRC 错误解决(3425)
- Linux下的Backlight子系统（一）(3028)
- Linux下的USB总线驱动（一）(2969)
- Determining IP information for eth0...failed 错误解决(2954)

评论排行

- Linux下的串口总线驱动（四）(6)
- Linux下的SPI总线驱动（三）(5)
- VFS: Cannot open root device "nfs" or unknown-block(0,255)错误解决(3)
- Linux下的USB总线驱动（二）(2)
- Linux下的I2C总线驱动(2)
- Linux下的SPI总线驱动（二）(2)
- 随笔————写于毕业前夕(2)
- Linux下的PCI总线驱动(2)
- Determining IP information for eth0...failed 错误解决(1)
- Linux内核中的系统调用(1)

推荐文章

- * 最老程序员创业开发实训4--IOS平台下MVC架构
- * Android基础入门教程--6.1 数据存储与访问之--文件存储读写
- * 聊天界面的制作（三）--表情列表发送功能
- * Linux下编程--文件与IO（三） 文件共享和fcntl函数
- * Windows 多进程通信API总结
- Redis学习总结和相关资料

最新评论

- Determining IP information for eth0...failed 错误解决
  weiqubo: 没有用，哎。
- Linux下的网络设备驱动(一)
  skyxiaoyan1: 简洁清晰
- error: variable '__this_module' has initializer but incomplete type错误解决
  hechengwang321: 这种问题，楼主你是怎么做出来的，谢谢楼主
- VFS: Cannot open root device "nfs" or unknown-block(0,255)错误解决
  helloskeety: IP_PNP这个选项在什么位置
- Linux内核中的系统调用
  yzh07137: 请问我尝试用read函数读取idt的内容，但是errno为1提示没权限，问了别人，说要用内核陷入的方...
- Linux下的SPI总线驱动（三）
  bjq1016: @lijing198997:谢谢 经过一段时间的学习，我明白了spi的底层，原来是设备驱动->平台...
- Linux下的SPI总线驱动（三）
  lijing198997: @wangdaobadao:你好，我的设备spidev0.0出来了，但是在执行调试spidev_te...
- Linux下的SPI总线驱动（三）
  lijing198997: @bjq1016:SPI_IOC_MESSAGE 是全双工的传输，你提的问题spi1到spi2,这个...

- Linux下的SPI总线驱动（三）
  wangdaobadao: 大哥啊，转载文章请实验之后再转行不行，真讨厌你们这种不实验直接转载的，还有文章中的大部分内容都没有交...
- Linux下的SPI总线驱动（二）
  bjq1016: 博主以后把代码单独放在代码块中吧。。。谢谢