努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处

http://blog.csdn.net/woshixingaaa/archive/2011/05/29/6452689.aspx

下面详细分析一下 framebuffer 的驱动源码,framebuffer 作为一个平台驱动注册进内核：

```
static struct platform_driver s3c2410fb_driver = {
    .probe      = s3c2410fb_probe,
    .remove     = s3c2410fb_remove,
    .suspend    = s3c2410fb_suspend,
    .resume     = s3c2410fb_resume,
    .driver     = {
        .name   = "s3c2410-lcd",
        .owner  = THIS_MODULE,
    },
};

int __init s3c2410fb_init(void)
{
    int ret = platform_driver_register(&s3c2410fb_driver);

    if (ret == 0)
        ret = platform_driver_register(&s3c2412fb_driver);;

    return ret;
}

static void __exit s3c2410fb_cleanup(void)
{
    platform_driver_unregister(&s3c2410fb_driver);
    platform_driver_unregister(&s3c2412fb_driver);
}

module_init(s3c2410fb_init);
module_exit(s3c2410fb_cleanup);
```

在 arch/arm/plat-s3c24xx/devs.c 中定义了framebuffer 的平台设备：

```
/* LCD Controller */
static struct resource s3c_lcd_resource[] = {
    [0] = {
        .start = S3C24XX_PA_LCD,                    //IO 内存的物理起始地址
        .end   = S3C24XX_PA_LCD + S3C24XX_SZ_LCD - 1,        //IO 内存的物理结束地址
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .start = IRQ_LCD,                             //LCD 的中断号
        .end   = IRQ_LCD,
        .flags = IORESOURCE_IRQ,
    }

};
static u64 s3c_device_lcd_dmamask = 0xffffffffUL;
struct platform_device s3c_device_lcd = {
    .name         = "s3c2410-lcd",
    .id       = -1,
    .num_resources    = ARRAY_SIZE(s3c_lcd_resource),
    .resource     = s3c_lcd_resource,
    .dev              = {
        .dma_mask       = &s3c_device_lcd_dmamask,
        .coherent_dma_mask  = 0xffffffffUL
    }
};

EXPORT_SYMBOL(s3c_device_lcd);
```

devs.c 中的这个函数把 s3c2410fb_mach_info 存放到 s3c_device_lcd.dev.platform_data，probe 函数中会用到的。

```
void __init s3c24xx_fb_set_platdata(struct s3c2410fb_mach_info *pd)
{
    struct s3c2410fb_mach_info *npd;

    npd = kmalloc(sizeof(*npd), GFP_KERNEL);
    if (npd) {
        memcpy(npd, pd, sizeof(*npd));
        s3c_device_lcd.dev.platform_data = npd;
    } else {
        printk(KERN_ERR "no memory for LCD platform data/n");
    }
}
```

这个函数是在 arch/arm/mach-s3c2440/mach-smdk2440.c 中的 smdk2440_machine_init 中调用的，所以在系统启动后会自动调用。

```
static void __init smdk2440_machine_init(void)
{
    s3c24xx_fb_set_platdata(&smdk2440_fb_info);
    s3c_i2c0_set_platdata(NULL);
    platform_add_devices(smdk2440_devices, ARRAY_SIZE(smdk2440_devices));
    smdk_machine_init();
}
```

s3c2410fb_display 表示屏的显示参数，这个结构体在我们移植 LCD 驱动的时候需要根据我们屏的参数重新设置。

```
/* LCD driver info */
static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {

    .lcdcon5    = S3C2410_LCDCON5_FRM565 |
              S3C2410_LCDCON5_INVVLINE |
              S3C2410_LCDCON5_INVVFRAME |
              S3C2410_LCDCON5_PWREN |
              S3C2410_LCDCON5_HWSWP,

    .type       = S3C2410_LCDCON1_TFT,

    .width      = 240,
    .height     = 320,

    .pixclock   = 270000,
    .xres       = 320,
    .yres       = 240,
    .bpp        = 16,
    .left_margin    = 8,
    .right_margin   = 5,
    .hsync_len  = 63,
    .upper_margin   = 15,
    .lower_margin   = 3,
    .vsync_len  = 5,
};
```

将 s3c2410fb_display 结构体存于 s3c2410fb_mach_info 的 displays 域。

```
static struct s3c2410fb_mach_info smdk2440_fb_info __initdata = {
    .displays   = &smdk2440_lcd_cfg,
    .num_displays   = 1,
    .default_display = 0,
    .lpcsel     = 0,
};
```

下面来看看当 lcd 驱动和设备匹配成功后会调用的探测函数：

```
static int __init s3c2410fb_probe(struct platform_device *pdev)
{
    return s3c24xxfb_probe(pdev, DRV_S3C2410);
}
```

这里调用了 s3c24xxfb_probe(pdev, DRV_S3C2410)，进行了一层封装，因为这样这部分代码可以与 s3c2412 进行复用。

```
static int __init s3c24xxfb_probe(struct platform_device *pdev,
                  enum s3c_drv_type drv_type)
{
    struct s3c2410fb_info *info;
    struct s3c2410fb_display *display;
    struct fb_info *fbinfo;
    struct s3c2410fb_mach_info *mach_info;
    struct resource *res;
    int ret;
    int irq;
    int i;
    int size;
    u32 lcdcon1;
    /*这就获得了刚才保存的 s3c2410fb_mach_info*/
    mach_info = pdev->dev.platform_data;
    if (mach_info == NULL) {
        dev_err(&pdev->dev,
            "no platform data for lcd, cannot attach/n");
        return -EINVAL;
    }
    if (mach_info->default_display >= mach_info->num_displays) {
        dev_err(&pdev->dev, "default is %d but only %d displays/n",
            mach_info->default_display, mach_info->num_displays);
        return -EINVAL;
    }
    /*获取显示屏的相关参数*/
    display = mach_info->displays + mach_info->default_display;
    /*获得中断号*/
    irq = platform_get_irq(pdev, 0);
    if (irq < 0) {
        dev_err(&pdev->dev, "no irq for device/n");
        return -ENOENT;
    }
    /*分配一个 fb_info 结构体*/
    fbinfo = framebuffer_alloc(sizeof(struct s3c2410fb_info), &pdev->dev);
    if (!fbinfo)
        return -ENOMEM;
    /*设置 pdev->dev->driver_data 保存 fbinfo 的地址*/
    platform_set_drvdata(pdev, fbinfo);
```

```
info = fbinfo->par;
info->dev = &pdev->dev;
info->drv_type = drv_type;
这 4 句构建的关系图如下:

/*获得 IO 内存*/
res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
if (res == NULL) {
    dev_err(&pdev->dev, "failed to get memory registers/n");
    ret = -ENXIO;
    goto dealloc_fb;
}

size = (res->end - res->start) + 1;
/*申请 IO 内存*/
info->mem = request_mem_region(res->start, size, pdev->name);
if (info->mem == NULL) {
    dev_err(&pdev->dev, "failed to get memory region/n");
    ret = -ENOENT;
    goto dealloc_fb;
}
/*映射 IO 内存*/
info->io = ioremap(res->start, size);
if (info->io == NULL) {
    dev_err(&pdev->dev, "ioremap() of registers failed/n");
    ret = -ENXIO;
    goto release_mem;
}
/*获得 LCD 中断挂起寄存器的基地址*/
info->irq_base = info->io + ((drv_type == DRV_S3C2412) ? S3C2412_LCDINTBASE : S3C2410_LCDINTBASE);

dprintk("devinit/n");

strcpy(fbinfo->fix.id, driver_name);

/*暂时关闭 LCD 控制器*/
lcdcon1 = readl(info->io + S3C2410_LCDCON1);
writel(lcdcon1 & ~S3C2410_LCDCON1_ENVID, info->io + S3C2410_LCDCON1);

fbinfo->fix.type        = FB_TYPE_PACKED_PIXELS;
fbinfo->fix.type_aux        = 0;
fbinfo->fix.xpanstep        = 0;
fbinfo->fix.ypanstep        = 0;
fbinfo->fix.ywrapstep          = 0;
```

```
fbinfo->fix.accel        = FB_ACCEL_NONE;

fbinfo->var.nonstd       = 0;
fbinfo->var.activate     = FB_ACTIVATE_NOW;
fbinfo->var.accel_flags     = 0;
fbinfo->var.vmode        = FB_VMODE_NONINTERLACED;
/*将底层操作函数与上层联系起来*/
fbinfo->fbops            = &s3c2410fb_ops;
fbinfo->flags            = FBINFO_FLAG_DEFAULT;
fbinfo->pseudo_palette      = &info->pseudo_pal;
/*用于填充调色板*/
for (i = 0; i < 256; i++)
    info->palette_buffer[i] = PALETTE_BUFF_CLEAR;
/*注册中断处理函数*/
ret = request_irq(irq, s3c2410fb_irq, IRQF_DISABLED, pdev->name, info);
if (ret) {
    dev_err(&pdev->dev, "cannot get irq %d - err %d/n", irq, ret);
    ret = -EBUSY;
    goto release_regs;
}
/*获得 LCD 时钟*/
info->clk = clk_get(NULL, "lcd");
if (!info->clk || IS_ERR(info->clk)) {
    printk(KERN_ERR "failed to get lcd clock source/n");
    ret = -ENOENT;
    goto release_irq;
}
/*使能 LCD 时钟*/
clk_enable(info->clk);
dprintk("got and enabled clock/n");
/*初始化 LCD 控制器之前要延时一段时间*/
msleep(1);

/*计算缓冲区需要的最大内存，就是缓冲区一共占多少字节，xres*yres*bpp/8 */
for (i = 0; i < mach_info->num_displays; i++) {
    unsigned long smem_len = mach_info->displays[i].xres;

    smem_len *= mach_info->displays[i].yres;
    smem_len *= mach_info->displays[i].bpp;
    smem_len >>= 3;
    if (fbinfo->fix.smem_len < smem_len)
        fbinfo->fix.smem_len = smem_len;
}
/*申请帧缓冲内存*/
ret = s3c2410fb_map_video_memory(fbinfo);
```

```
    if (ret) {
        printk(KERN_ERR "Failed to allocate video RAM: %d/n", ret);
        ret = -ENOMEM;
        goto release_clock;
    }

    dprintk("got video memory/n");

    fbinfo->var.xres = display->xres;
    fbinfo->var.yres = display->yres;
    fbinfo->var.bits_per_pixel = display->bpp;
    /*初始化相关寄存器*/
    s3c2410fb_init_registers(fbinfo);
    /*检查 fb_info 中的可变参数*/
    s3c2410fb_check_var(&fbinfo->var, fbinfo);

    ret = register_framebuffer(fbinfo);                //注册帧缓冲设备
    if (ret < 0) {
        printk(KERN_ERR "Failed to register framebuffer device: %d/n",
            ret);
        goto free_video_memory;
    }

    /* create device files */
    ret = device_create_file(&pdev->dev, &dev_attr_debug);  //创建设备文件
    if (ret) {
        printk(KERN_ERR "failed to add debug attribute/n");
    }

    printk(KERN_INFO "fb%d: %s frame buffer device/n",
        fbinfo->node, fbinfo->fix.id);

    return 0;

free_video_memory:
    s3c2410fb_unmap_video_memory(fbinfo);
release_clock:
    clk_disable(info->clk);
    clk_put(info->clk);
release_irq:
    free_irq(irq, info);
release_regs:
    iounmap(info->io);
release_mem:
    release_resource(info->mem);
    kfree(info->mem);
dealloc_fb:
    platform_set_drvdata(pdev, NULL);
    framebuffer_release(fbinfo);
    return ret;
}
```

总结一下探测函数完成的任务：

1）申请 fb_info 结构体的内存空间，初始化 fb_info 结构中固定和可变的内存参数，即填充 fb_info 中的 fb_var_screeninfo var 和 struct fb_fix_screeninfo fix 成员。

2）申请帧缓冲设备的显示缓冲区空间

3）注册帧缓冲设备

```
struct fb_info *framebuffer_alloc(size_t size, struct device *dev)
{
#define BYTES_PER_LONG (BITS_PER_LONG/8)
#define PADDING (BYTES_PER_LONG - (sizeof(struct fb_info) % BYTES_PER_LONG))
    int fb_info_size = sizeof(struct fb_info);
    struct fb_info *info;
    char *p;
    if (size)
        fb_info_size += PADDING;
    /*这里开辟的堆空间用来存储 struct fb_info 结构体和 struct s3c2410fb_info 结构体*/
    p = kzalloc(fb_info_size + size, GFP_KERNEL);
    if (!p)
        return NULL;
    info = (struct fb_info *) p;
    /*在这里将 par 成员赋值，以后用于存储 struct s3c2410fb_info 结构*/
    if (size)
        info->par = p + fb_info_size;
    info->device = dev;

#ifdef CONFIG_FB_BACKLIGHT
    mutex_init(&info->bl_curve_mutex);
#endif

    return info;
#undef PADDING
#undef BYTES_PER_LONG
}
```

中断处理函数：

```
static irqreturn_t s3c2410fb_irq(int irq, void *dev_id)
```

```
{
    struct s3c2410fb_info *fbi = dev_id;
    /*LCD 中断挂起寄存器基地址*/
    void __iomem *irq_base = fbi->irq_base;
    /*读取 LCD 中断挂起寄存器值*/
    unsigned long lcdirq = readl(irq_base + S3C24XX_LCDINTPND);
    /*如果 framebuffer 发出了中断请求*/
    if (lcdirq & S3C2410_LCDINT_FRSYNC) {
        /*填充调色板*/
        if (fbi->palette_ready)
            s3c2410fb_write_palette(fbi);
        /*设置帧已插入中断请求*/
        writel(S3C2410_LCDINT_FRSYNC, irq_base + S3C24XX_LCDINTPND);
        writel(S3C2410_LCDINT_FRSYNC, irq_base + S3C24XX_LCDSRCPND);
    }

    return IRQ_HANDLED;
}
```

填充调色板:

```
static void s3c2410fb_write_palette(struct s3c2410fb_info *fbi)
{
    unsigned int i;
    void __iomem *regs = fbi->io;

    fbi->palette_ready = 0;

    for (i = 0; i < 256; i++) {
        unsigned long ent = fbi->palette_buffer[i];
        if (ent == PALETTE_BUFF_CLEAR)
            continue;

        writel(ent, regs + S3C2410_TFTPAL(i));

        /* it seems the only way to know exactly
         * if the palette wrote ok, is to check
         * to see if the value verifies ok
         */

        if (readw(regs + S3C2410_TFTPAL(i)) == ent)
            fbi->palette_buffer[i] = PALETTE_BUFF_CLEAR;
        else
            fbi->palette_ready = 1;   /* retry */
    }
}
```

申请帧缓冲设备 fb_info 的缓冲区空间：

```
static int __init s3c2410fb_map_video_memory(struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    dma_addr_t map_dma;
    /*获得帧缓冲区的大小*/
    unsigned map_size = PAGE_ALIGN(info->fix.smem_len);

    dprintk("map_video_memory(fbi=%p) map_size %u/n", fbi, map_size);
    /*分配一个写合并缓冲区来设置帧缓冲的虚拟地址*/
    info->screen_base = dma_alloc_writecombine(fbi->dev, map_size,
                          &map_dma, GFP_KERNEL);
    if (info->screen_base) {
        /* prevent initial garbage on screen */
        dprintk("map_video_memory: clear %p:%08x/n",
            info->screen_base, map_size);
        /*初始化为 0*/
        memset(info->screen_base, 0x00, map_size);
        /*设置物理地址*/
        info->fix.smem_start = map_dma;
        dprintk("map_video_memory: dma=%08lx cpu=%p size=%08x/n",
            info->fix.smem_start, info->screen_base, map_size);
    }
    /*返回虚拟地址*/
    return info->screen_base ? 0 : -ENOMEM;
}
```

初始化相关寄存器：

```
static int s3c2410fb_init_registers(struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    struct s3c2410fb_mach_info *mach_info = fbi->dev->platform_data;
    unsigned long flags;
    /*获得 LCD 寄存器基地址，这个在 probe 中获得*/
    void __iomem *regs = fbi->io;
    void __iomem *tpal;
    void __iomem *lpcsel;

    if (is_s3c2412(fbi)) {
        tpal = regs + S3C2412_TPAL;
        lpcsel = regs + S3C2412_TCONSEL;
    } else {
        /*获得 LCD 调色板寄存器基地址，注意对于 lpcsel 这是一个针对三星 TFT 屏的一个专用寄存器，如果用的不是三星的屏就不用管它*/
        tpal = regs + S3C2410_TPAL;
        lpcsel = regs + S3C2410_LPCSEL;
    }

    /* Initialise LCD with values from haret */
    /*关中断*/
    local_irq_save(flags);

    /* modify the gpio(s) with interrupts set (bjd) */
    /*把 IO 端口 C 和 D 设置成 LCD 模式*/
    modify_gpio(S3C2410_GPCUP,  mach_info->gpcup,  mach_info->gpcup_mask);
    modify_gpio(S3C2410_GPCCON, mach_info->gpccon, mach_info->gpccon_mask);
    modify_gpio(S3C2410_GPDUP,  mach_info->gpdup,  mach_info->gpdup_mask);
    modify_gpio(S3C2410_GPDCON, mach_info->gpdcon, mach_info->gpdcon_mask);
    /*恢复被屏蔽的中断*/
    local_irq_restore(flags);

    dprintk("LPCSEL    = 0x%08lx/n", mach_info->lpcsel);
    writel(mach_info->lpcsel, lpcsel);

    dprintk("replacing TPAL %08x/n", readl(tpal));

    /*临时调色板使能禁止*/
    writel(0x00, tpal);

    return 0;
}
```

设置 fb_info 中的可变参数:

```
static int s3c2410fb_check_var(struct fb_var_screeninfo *var,
                struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    struct s3c2410fb_mach_info *mach_info = fbi->dev->platform_data;
    struct s3c2410fb_display *display = NULL;
    struct s3c2410fb_display *default_display = mach_info->displays +
                        mach_info->default_display;
    /*LCD 的类型，S3C2410_LCDCON1_TFT*/
    int type = default_display->type;
    unsigned i;

```

```
dprintk("check_var(var=%p, info=%p)/n", var, info);

/*获取与LCD屏有关的参数，封装在s3c2410fb_display中*/
if (var->yres == default_display->yres &&
    var->xres == default_display->xres &&
    var->bits_per_pixel == default_display->bpp)
    display = default_display;
else
    for (i = 0; i < mach_info->num_displays; i++)
        if (type == mach_info->displays[i].type &&
            var->yres == mach_info->displays[i].yres &&
            var->xres == mach_info->displays[i].xres &&
            var->bits_per_pixel == mach_info->displays[i].bpp) {
            display = mach_info->displays + i;
            break;
        }

if (!display) {
    dprintk("wrong resolution or depth %dx%d at %d bpp/n",
        var->xres, var->yres, var->bits_per_pixel);
    return -EINVAL;
}

/*配置屏的虚拟解析度和高度宽度*/
var->xres_virtual = display->xres;
var->yres_virtual = display->yres;
var->height = display->height;
var->width = display->width;

/*这里是时序了，设置时钟像素，行帧切换值，水平同步，垂直同步切换值*/
var->pixclock = display->pixclock;
var->left_margin = display->left_margin;
var->right_margin = display->right_margin;
var->upper_margin = display->upper_margin;
var->lower_margin = display->lower_margin;
var->vsync_len = display->vsync_len;
var->hsync_len = display->hsync_len;

/*配置LCD控制寄存器1中5-6位(配置成TFT类型)，配置寄存器5*/
fbi->regs.lcdcon5 = display->lcdcon5;
/* set display type */
fbi->regs.lcdcon1 = display->type;

/*设置透明度*/
var->transp.offset = 0;
```

```
var->transp.length = 0;
/*根据色位模式设置(BPP)来设置可变参数中 R,G,B 的颜色位域,显示缓冲区与显示点对应如下图：*/
```

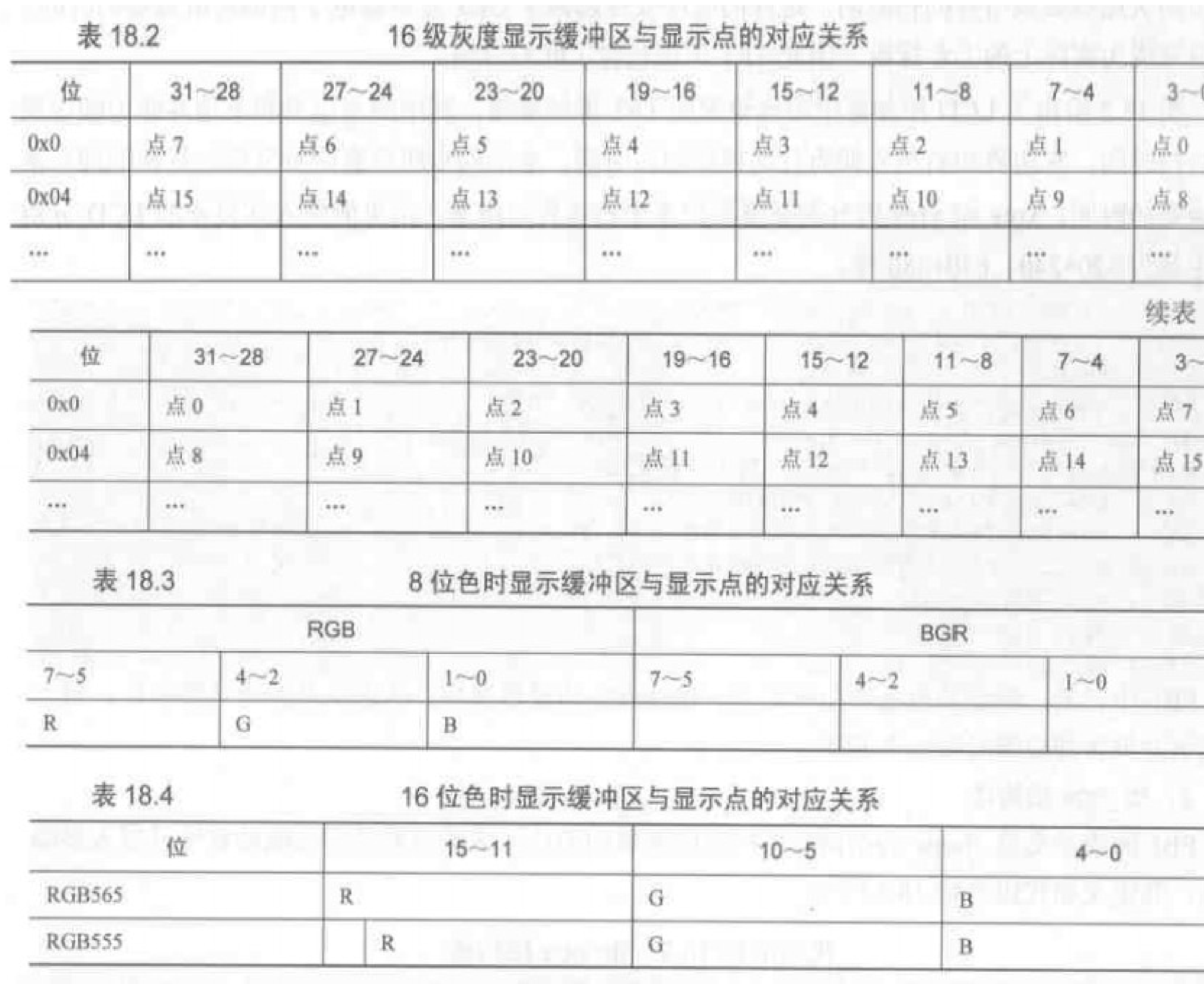

表 18.2　　16 级灰度显示缓冲区与显示点的对应关系

位	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~
0x0	点 7	点 6	点 5	点 4	点 3	点 2	点 1	点 0
0x04	点 15	点 14	点 13	点 12	点 11	点 10	点 9	点 8
…	…	…	…	…	…	…	…	…

续表

位	31~28	27~24	23~20	19~16	15~12	11~8	7~4	3~
0x0	点 0	点 1	点 2	点 3	点 4	点 5	点 6	点 7
0x04	点 8	点 9	点 10	点 11	点 12	点 13	点 14	点 15
…	…	…	…	…	…	…	…	…

表 18.3　　8 位色时显示缓冲区与显示点的对应关系

RGB			BGR		
7~5	4~2	1~0	7~5	4~2	1~0
R	G	B			

表 18.4　　16 位色时显示缓冲区与显示点的对应关系

位	15~11		10~5	4~0
RGB565	R		G	B
RGB555		R	G	B

```
switch (var->bits_per_pixel) {
  case 1:
  case 2:
  case 4:
    var->red.offset  = 0;
    var->red.length  = var->bits_per_pixel;
    var->green   = var->red;
    var->blue    = var->red;
    break;
  case 8:
    if (display->type != S3C2410_LCDCON1_TFT) {
      /* 8 bpp 332 */
      var->red.length      = 3;
```

```
            var->red.offset      = 5;
            var->green.length    = 3;
            var->green.offset    = 2;
            var->blue.length = 2;
            var->blue.offset = 0;
        } else {
            var->red.offset      = 0;
            var->red.length      = 8;
            var->green       = var->red;
            var->blue        = var->red;
        }
        break;
    case 12:
        /* 12 bpp 444 */
        var->red.length      = 4;
        var->red.offset      = 8;
        var->green.length    = 4;
        var->green.offset    = 4;
        var->blue.length = 4;
        var->blue.offset = 0;
        break;

    default:
    case 16:
        if (display->lcdcon5 & S3C2410_LCDCON5_FRM565) {
            /* 16 bpp, 565 format */
            var->red.offset      = 11;
            var->green.offset    = 5;
            var->blue.offset = 0;
            var->red.length      = 5;
            var->green.length    = 6;
            var->blue.length = 5;
        } else {
            /* 16 bpp, 5551 format */
            var->red.offset      = 11;
            var->green.offset    = 6;
            var->blue.offset = 1;
            var->red.length      = 5;
            var->green.length    = 5;
            var->blue.length = 5;
        }
        break;
    case 32:
        /* 24 bpp 888 and 8 dummy */
        var->red.length      = 8;
        var->red.offset       = 16;
        var->green.length     = 8;
        var->green.offset     = 8;
        var->blue.length = 8;
        var->blue.offset = 0;
        break;
    }
    return 0;
}
```

注册帧缓冲设备:

```
int
register_framebuffer(struct fb_info *fb_info)
{
    int i;
    struct fb_event event;
    struct fb_videomode mode;

    if (num_registered_fb == FB_MAX)
        return -ENXIO;

    if (fb_check_foreignness(fb_info))
        return -ENOSYS;
    /*
     *每一个注册的 fb_info,都会分配一个下标"i"，对应的就是 registered_fb[i]
     *最多能注册的 fb_info 个数为 FB_MAX,若新注册 FB 则 num_registered_fb++
     */
    num_registered_fb++;
    for (i = 0 ; i < FB_MAX; i++)
        if (!registered_fb[i])
            break;
    /*找到空闲的 i，赋值给 fb_info->node,这个 node 相当于次设备号了，以后通过这个 i 找到 fb_info*/
    fb_info->node = i;
    mutex_init(&fb_info->lock);
    /*创建设备文件*/
    fb_info->dev = device_create(fb_class, fb_info->device,
                 MKDEV(FB_MAJOR, i), NULL, "fb%d", i);
    if (IS_ERR(fb_info->dev)) {
        /* Not fatal */
        printk(KERN_WARNING "Unable to create device for framebuffer %d; errno = %ld/n", i, PTR_ERR(fb_info->dev));
        fb_info->dev = NULL;
    } else
        /*初始化 fb 的属性文件*/
        fb_init_device(fb_info);
    。。。。。。。。。。。。。。
    return 0;
}

static struct fb_ops s3c2410fb_ops = {
    .owner      = THIS_MODULE,
    .fb_check_var   = s3c2410fb_check_var,
    .fb_set_par = s3c2410fb_set_par,
    .fb_blank   = s3c2410fb_blank,
    .fb_setcolreg   = s3c2410fb_setcolreg,
    .fb_fillrect    = cfb_fillrect,
    .fb_copyarea    = cfb_copyarea,
    .fb_imageblit   = cfb_imageblit,
};
```

设置参数，根据可变参数设置固定参数：

```
static int s3c2410fb_set_par(struct fb_info *info)
{
    struct fb_var_screeninfo *var = &info->var;
    /*根据可变参数的位色模式*/
    switch (var->bits_per_pixel) {
    case 32:
    case 16:
    case 12://设置成真彩，分红，绿，蓝三基色
        info->fix.visual = FB_VISUAL_TRUECOLOR;
        break;
    case 1://设置为黑白，FB_VISUAL_MONO01 代表黑，FB_VISUAL_MONO10 代表白
        info->fix.visual = FB_VISUAL_MONO01;
        break;
    default://默认设置为伪彩色，采用索引颜色显示
        info->fix.visual = FB_VISUAL_PSEUDOCOLOR;
        break;
    }
    /*设置 fb_info 中固定参数一行的字节数*/
    info->fix.line_length = (var->xres_virtual * var->bits_per_pixel) / 8;

    /*激活新的参数配置，设置控制寄存器的值*/
    s3c2410fb_activate_var(info);
    return 0;
}
```

激活设置：

```
static void s3c2410fb_activate_var(struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
```

```
void __iomem *regs = fbi->io;
/*获得屏的类型*/
int type = fbi->regs.lcdcon1 & S3C2410_LCDCON1_TFT;
struct fb_var_screeninfo *var = &info->var;
/*获得 CLKVAL*/
int clkdiv = s3c2410fb_calc_pixclk(fbi, var->pixclock) / 2;

dprintk("%s: var->xres  = %d/n", __func__, var->xres);
dprintk("%s: var->yres  = %d/n", __func__, var->yres);
dprintk("%s: var->bpp   = %d/n", __func__, var->bits_per_pixel);

if (type == S3C2410_LCDCON1_TFT) {
    /*就是根据可变参数结构设置 lcdcon1~5*/
    s3c2410fb_calculate_tft_lcd_regs(info, &fbi->regs);
    --clkdiv;
    if (clkdiv < 0)
        clkdiv = 0;
} else {
    s3c2410fb_calculate_stn_lcd_regs(info, &fbi->regs);
    if (clkdiv < 2)
        clkdiv = 2;
}
/*设置分频值*/
fbi->regs.lcdcon1 |=  S3C2410_LCDCON1_CLKVAL(clkdiv);

/* write new registers */

dprintk("new register set:/n");
dprintk("lcdcon[1] = 0x%08lx/n", fbi->regs.lcdcon1);
dprintk("lcdcon[2] = 0x%08lx/n", fbi->regs.lcdcon2);
dprintk("lcdcon[3] = 0x%08lx/n", fbi->regs.lcdcon3);
dprintk("lcdcon[4] = 0x%08lx/n", fbi->regs.lcdcon4);
dprintk("lcdcon[5] = 0x%08lx/n", fbi->regs.lcdcon5);
/*设置寄存器前先把 LCD 使能关闭，然后将刚才设置的值写入真正的寄存器*/
writel(fbi->regs.lcdcon1 & ~S3C2410_LCDCON1_ENVID,
    regs + S3C2410_LCDCON1);
writel(fbi->regs.lcdcon2, regs + S3C2410_LCDCON2);
writel(fbi->regs.lcdcon3, regs + S3C2410_LCDCON3);
writel(fbi->regs.lcdcon4, regs + S3C2410_LCDCON4);
writel(fbi->regs.lcdcon5, regs + S3C2410_LCDCON5);

/*设置 LCDSADDR1 ~ 3*/
s3c2410fb_set_lcdaddr(info);
/*使能 LCD*/
fbi->regs.lcdcon1 |= S3C2410_LCDCON1_ENVID,
    writel(fbi->regs.lcdcon1, regs + S3C2410_LCDCON1);
}
```

显示空白：blank_mode 有 5 中模式，是一个枚举，定义在 include/linux/fb.h 中：

```
static int s3c2410fb_blank(int blank_mode, struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    void __iomem *tpal_reg = fbi->io;

    dprintk("blank(mode=%d, info=%p)/n", blank_mode, info);

    tpal_reg += is_s3c2412(fbi) ? S3C2412_TPAL : S3C2410_TPAL;

    if (blank_mode == FB_BLANK_POWERDOWN) {   //如果是空白模式，则关闭 LCD
        s3c2410fb_lcd_enable(fbi, 0);
    } else {
        s3c2410fb_lcd_enable(fbi, 1);
    }
    if (blank_mode == FB_BLANK_UNBLANK)
        /*临时调色板无效*/
        writel(0x0, tpal_reg);
    else {
        /*临时调色板有效*/
        dprintk("setting TPAL to output 0x000000/n");
        writel(S3C2410_TPAL_EN, tpal_reg);
    }
    return 0;
}
```

设置颜色表：

```
static int s3c2410fb_setcolreg(unsigned regno,
                  unsigned red, unsigned green, unsigned blue,
                  unsigned transp, struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    void __iomem *regs = fbi->io;
    unsigned int val;

    /* dprintk("setcol: regno=%d, rgb=%d,%d,%d/n",
           regno, red, green, blue); */

    switch (info->fix.visual) {
        /*真彩色*/
    case FB_VISUAL_TRUECOLOR:
        /* true-colour, use pseudo-palette */

        if (regno < 16) {
            u32 *pal = info->pseudo_palette;

            val  = chan_to_field(red,   &info->var.red);
            val |= chan_to_field(green, &info->var.green);
            val |= chan_to_field(blue,  &info->var.blue);

            pal[regno] = val;
        }
        break;
        /*伪彩色*/
    case FB_VISUAL_PSEUDOCOLOR:
        if (regno < 256) {
            /* currently assume RGB 5-6-5 mode */

            val  = (red   >>  0) & 0xf800;
            val |= (green >>  5) & 0x07e0;
            val |= (blue  >> 11) & 0x001f;

            writel(val, regs + S3C2410_TFTPAL(regno));
            /*修改调色板*/
            schedule_palette_update(fbi, regno, val);
        }
        break;
    default:
        return 1;   /* unknown type */
    }
    return 0;
}
static inline unsigned int chan_to_field(unsigned int chan,
                     struct fb_bitfield *bf)
{
    chan &= 0xffff;
    chan >>= 16 - bf->length;
    return chan << bf->offset;
}
```

修改调色板:

```
static void schedule_palette_update(struct s3c2410fb_info *fbi,
                    unsigned int regno, unsigned int val)
{
    unsigned long flags;
    unsigned long irqen;
    /*LCD 中断挂起寄存器基地址*/
    void __iomem *irq_base = fbi->irq_base;

    /*屏蔽中断，将中断状态保存在 flags 中*/
    local_irq_save(flags);

    fbi->palette_buffer[regno] = val;
    /*判断调色板是否准备就绪*/
    if (!fbi->palette_ready) {
        fbi->palette_ready = 1;
        /*使能中断屏蔽寄存器*/
        irqen = readl(irq_base + S3C24XX_LCDINTMSK);
        irqen &= ~S3C2410_LCDINT_FRSYNC;
        writel(irqen, irq_base + S3C24XX_LCDINTMSK);
    }
    /*回复被屏蔽的中断*/
    local_irq_restore(flags);
}
```