SC8810 LCD 驱动介绍

课程名称： SC8810 LCD 驱动介绍

课程类别： 客户培训课程

课程目标:

1. Lcd 接口
2. Lcdc刷屏原理
3. Framebuffer接口
4. 客户定制要点

对象： 客户　　课时数： 1H

讲师： 驱动软件工程师　　教学法： 面授

主要内容

- SC8810 LCD硬件接口介绍
- SC8810 LCDC刷屏过程
- SC8810 LCDC/LCD驱动函数接口介绍
- SC8810 Framebffer驱动及接口介绍
- SC8810 LCD 客户定制简介

1 SC8810 LCD硬件接口介绍

1. SC8810支持的硬件接口
2. SC8810 MCU接口
3. I80 模式时序分析

1.1 SC8810支持的接口

并行接口

SC8810 LCM支持的硬件接口是MCU接口，MCU接口根据时序不同分为两种类型

1、 Intel的80接口。

2、 Motorola的68接口。

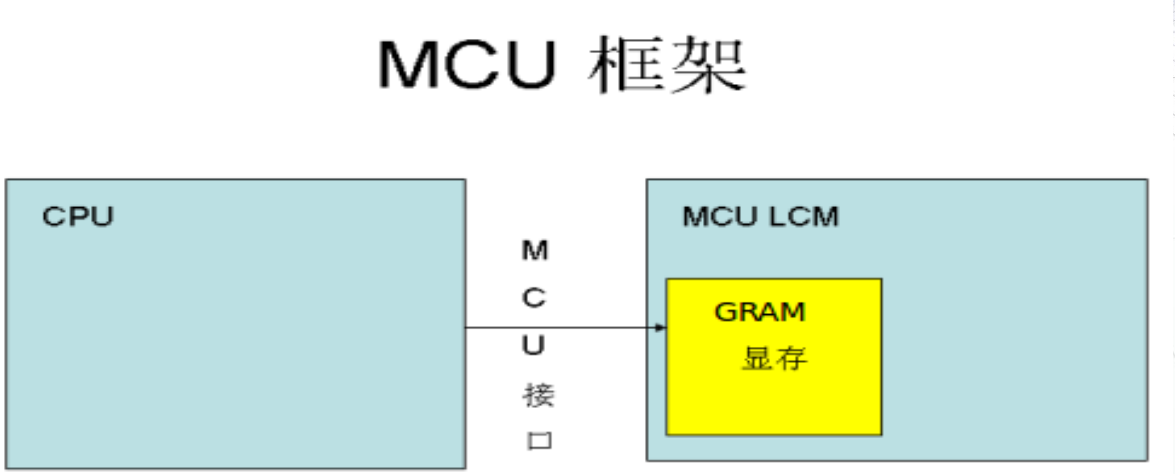
两种接口时序通过寄存器0x2070014C进行选择。

芯片默认的是80接口。

MCU接口控制信号线有：WR，RD，RS，RESET，CS

1.2 SC8810 MCU 接口

MCU接口特点

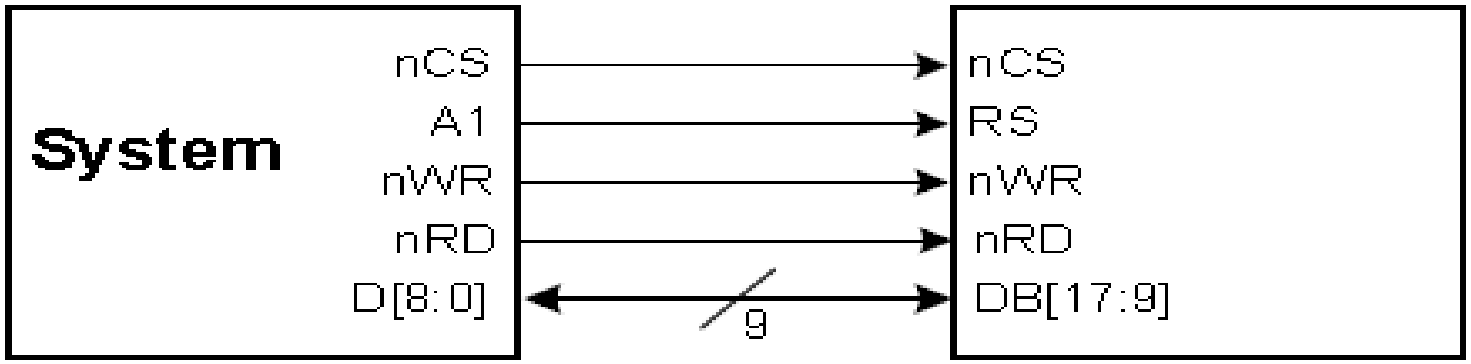


MCU接口的特点是控制简单不需要时钟和同步信号。刷屏的数据首先写到LCD屏的内部GRAM中，然后由LCD自动刷到屏上。由于GRAM大小的限制，这种接口的屏都难以做大。目前SC8810支持的LCD屏的大小为QCIF，QVGA，VGA，CIF，HVGA，WVGA。

MCU接口数据线是并行连接的，一般称为数据总线。目前SC8810支持的数据总线的宽度为8，9，16，18位。

1.2 SC8810 MCU 接口

以9位总线的MCU接口为例



	1st Transfer (Upper bits)									2nd Transfer (Lower bits)								
Input Data	DB 17	DB 16	DB 15	DB 14	DB 13	DB 12	DB 11	DB 10	DB 9	DB 17	DB 16	DB 15	DB 14	DB 13	DB 12	DB 11	DB 10	DB 9
Write Data Register	WD 17	WD 16	WD 15	WD 14	WD 13	WD 12	WD 11	WD 10	WD 9	WD 8	WD 7	WD 6	WD 5	WD 4	WD 3	WD 2	WD 1	WD 0
GRAM Data & RGB Mapping	R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0

1.2 SC8810 MCU 接口

CS: 片选线，表示是否对LCD屏操作低电平有效。
RS：命令/数据选择线，告诉LCD屏数据总线上传送的是命令还是数据。
WR：写入信号线，表示数据要写入LCD的寄存器或GRAM。
RD：读出信号线，表示要从LCD寄存器中读出数据。
DB[17:9]：数据总线，在SC8810侧连接到D[8:0]。

1.2 SC8810 MCU 接口

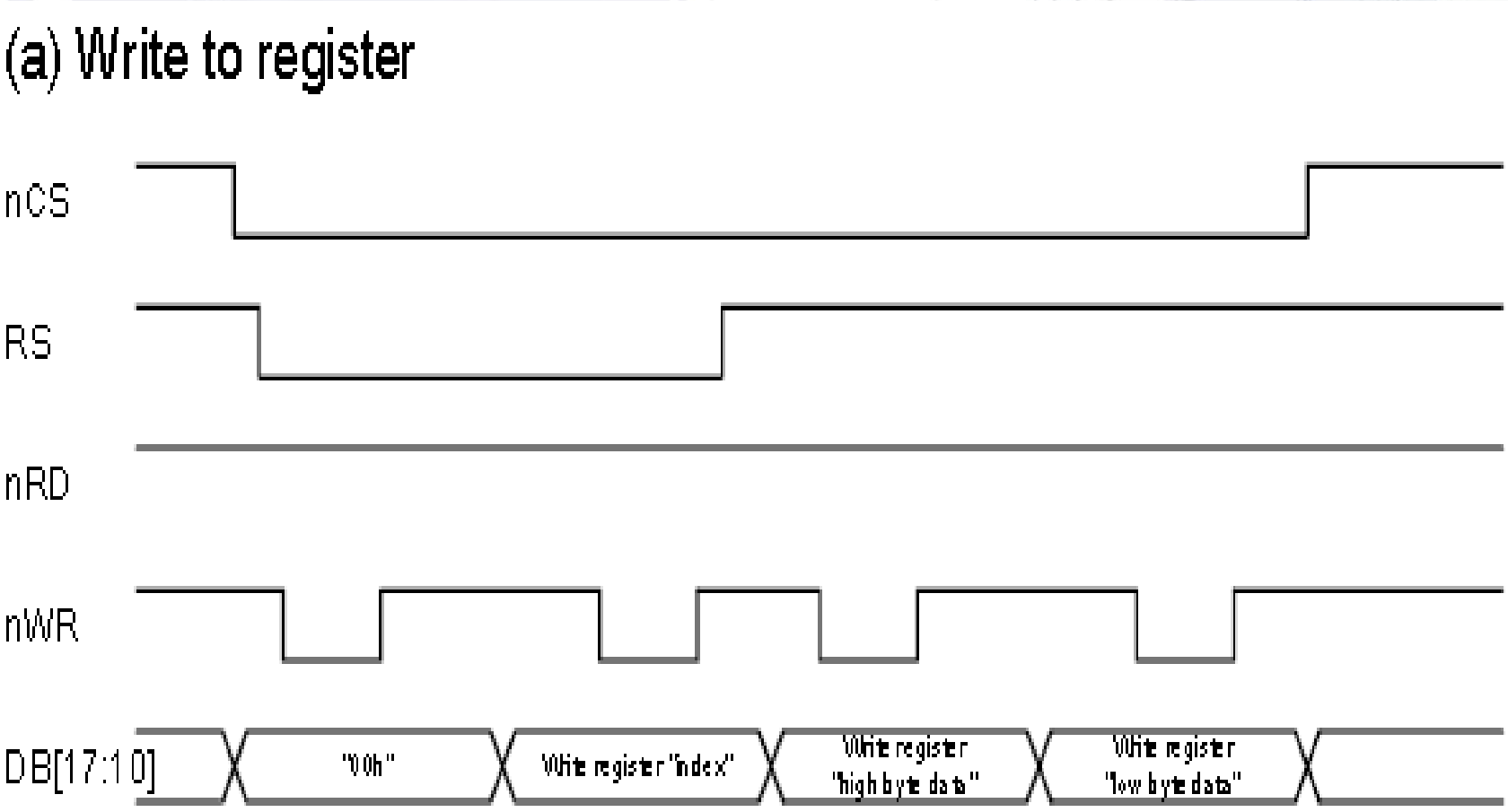
目前的SC8810平台的硬件参考设计中一般使用8位并行接口，16位并行接口。

1、使用8位并接口，BB端接16位总线的低8位[7~0]。LCD接18位的高8位[17~10]。

2、 使用16位并接口，BB端接16位总线[15~0]。LCD接18位中的[8~1]和[17~10]。

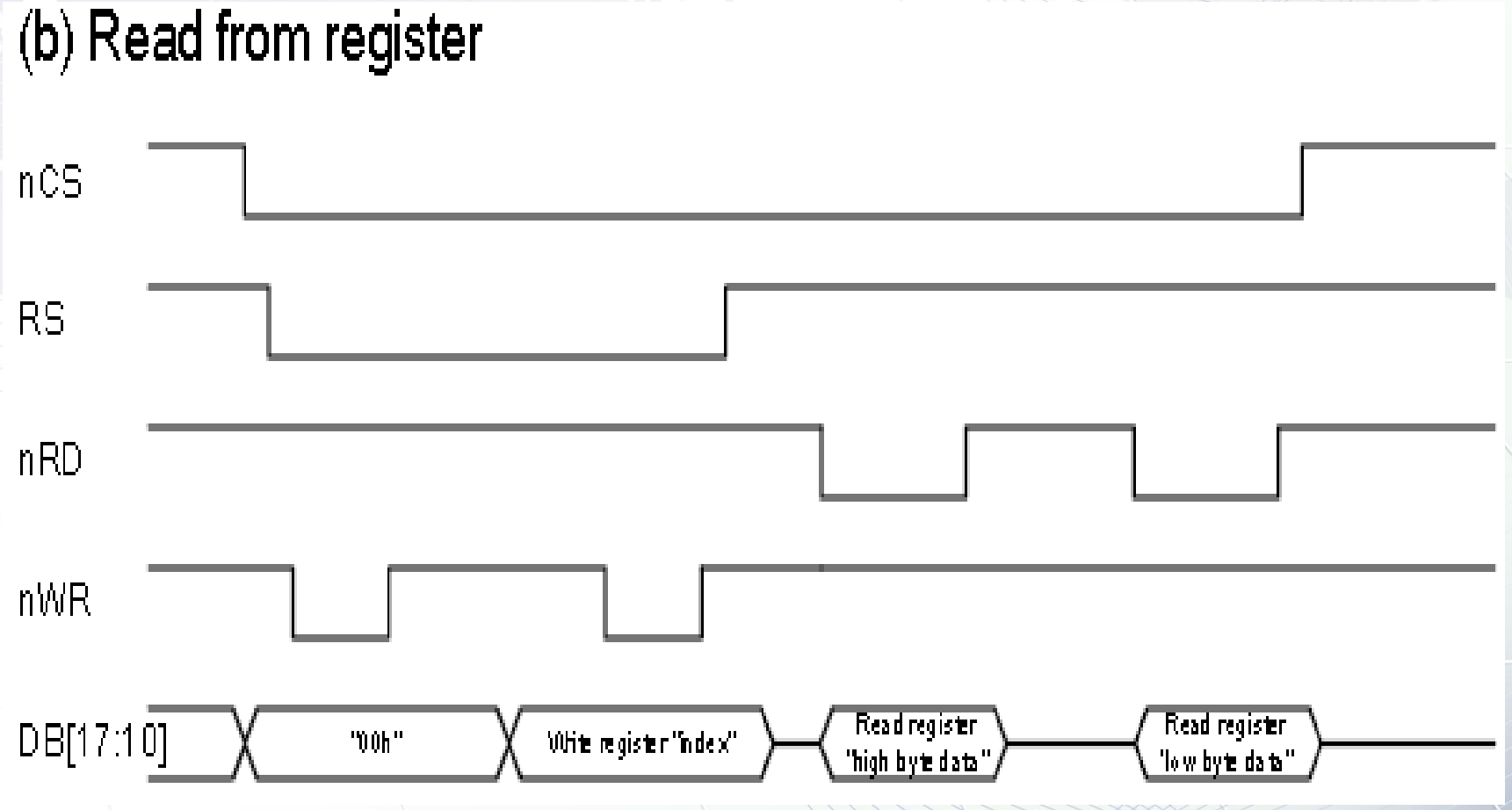
1.3 I80模式时序

I80模式写LCD寄存器时序图



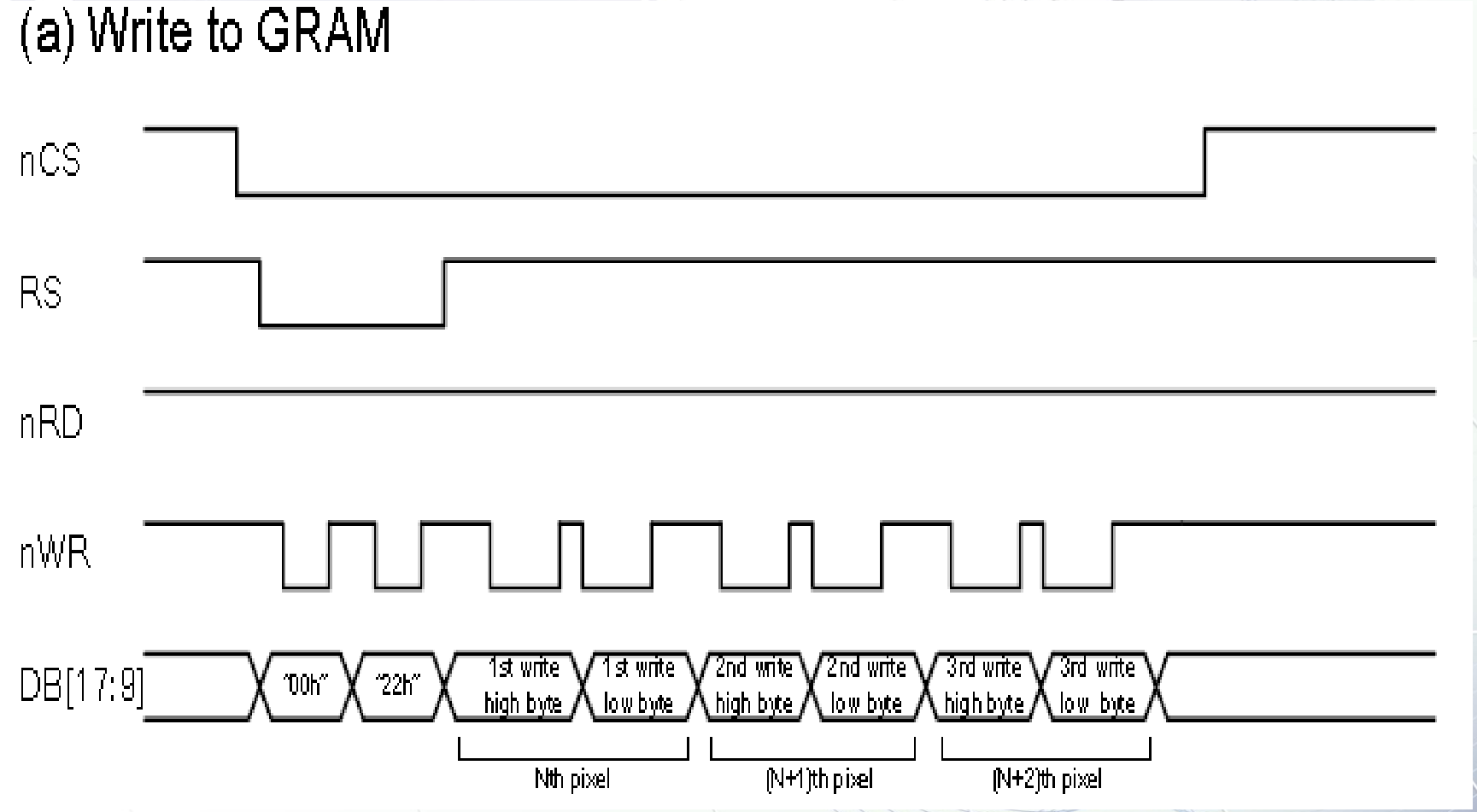
1.3 I80模式时序

I80模式读LCD寄存器时序图



1.3 I80模式时序

I80模式写LCD GRAM时序图



2 SC8810 LCDC硬件接口介绍

1. LCDC控制器简介
2. LCDC刷屏过程函数接口
3. LCD驱动函数接口

2.1 SC8810 LCDC介绍

LCDC是什么？

LCDC是LCD Controller的缩写，用于获取图像数据和OSD数据，并将这些数据混合在一起,输出到LCD屏或写回到内存中。

SC8810支持6个数据源，一个图像层数据源，5个OSD层数据源,这6个数据源可以同时进行alpha合成。

LCDC控制器内部包含了两个LCM单元。通过这两个LCM单元，LCDC可以同时控制两个屏。两个LCM的选择由寄存器控制，请参考 **<<SC8810 Device Specification>>**

正常刷屏情况下，LCDC将合成后的图像直接通过LCM刷到LCD屏上。

同时支持两个屏只有片选线是独立的，其他数据和信号线复用。

LCDC支持FMARK功能。

2.2 LCDC刷屏过程函数接口

Sc8810中底层刷屏接口函数是
/kernel/drivers/video/sc8810/fb_rrm.c

//刷新请求管理,初始化帧状态、OSD层次数据源以及分配初始化刷新请求队列

```
struct rrmanager* rrm_init(void (*hw_refresh)(void*para), void*
para);
//发送刷新请求函数
int rrm_refresh(int id, void (*callback)(void* data), void *data);
//初始化id对应的层
int rrm_layer_init(int id, int buf_num, void (*set_layer)(void *data));
//刷新层中断处理函数
void rrm_interrupt(struct rrmanager *rrm);
```

2.2 LCDC刷屏过程函数接口

lcdc刷屏请求函数rrm_refresh

```
int rrm_refresh(int id, void (*callback)(void* data), void *data)
{
unsigned long flags;
struct rr r = {callback, data}; //刷新请求数据

/* acquire lcdc first */    //向lcdc请求一个层
if(lm_acquire(id)) {
        printk(KERN_ERR "lm_acquire failed!\n");
        return -1;
}

spin_lock_irqsave(&rrm.lock, flags);
```

2.2 LCDC刷屏过程函数接口

```
//判断当前rrm是否处于busying状态，如果是则放入等候队列等候处理
if (rrm.frame_state == FS_BUSY) {
        int available;
        available = rrm.que[id]->enqueue(rrm.que[id], &r);
        if(available == 0) {
                RRM_PRINT("RRM[%s] layer[%d] is waiting\n",
                                __FUNCTION__, id);
                spin_unlock_irqrestore(&rrm.lock, flags);
                rrm.que[id]->wait(rrm.que[id]);
                spin_lock_irqsave(&rrm.lock, flags);
                RRM_PRINT("RRM[%s] layer[%d] has waked up\n",
                                __FUNCTION__, id);
        }
} else {    //否则调用refresh函数发送刷屏请求
        rrm.frame_state = FS_BUSY;
        rrm.exec->refresh(rrm.exec, id, &r);
}

spin_unlock_irqrestore(&rrm.lock, flags);
return 0;
```

2.2 LCDC刷屏过程函数接口

真正刷屏接口
/kernel/drivers/video/sc8810/fb_main.c

```
static void real_refresh(void *para)
 {
      struct sc8810fb_info *info = (struct sc8810fb_info *)para;

//设置全屏显示(根据具体的屏设置)
     info->panel->ops->lcd_invalidate(info->panel);

       //设置写gram时序
 lcdc_update_lcm_timing(info->write_gram_timing);

      //设置lcdc刷屏寄存器
   __raw_bits_or((1<<3), LCDC_CTRL); /* start refresh */
}
```

2.2 LCDC刷屏过程函数接口

真正设置OSD层接口

/kernel/drivers/video/sc8810/fb_main.c

```
static void real_set_layer(void *data)
{
struct fb_info *fb = (struct fb_info *)data;

uint32_t reg_val = (fb->var.yoffset == 0)?fb->fix.smem_start:
                   (fb->fix.smem_start + fb->fix.smem_len/2);
#ifdef LCD_UPDATE_PARTLY
if (fb->var.reserved[0] == 0x6f766572) {
      uint32_t x,y;
      x = fb->var.reserved[1] & 0xffff;
      y = fb->var.reserved[1] >> 16;

      reg_val += ((x + y * fb->var.xres) * fb->var.bits_per_pixel / 8);
}
#endif
__raw_writel(reg_val, LCDC_OSD1_BASE_ADDR);
}
```

2.3 LCD驱动函数接口

```
// 这个函数指针对应的函数用于初始化LCD屏。
int32_t (*lcd_init)(struct lcd_spec *self);
//这个函数使LCD进入睡眠。
int32_t (*lcd_enter_sleep)(struct lcd_spec *self, uint8_t is_sleep);
//设置LCD的对比度
int32_t (*lcd_set_contrast)(struct lcd_spec *self, uint16_t contrast);
//设置LCD的亮度
int32_t (*lcd_set_brightness)(struct lcd_spec *self,uint16_t brightness);
//设置显示窗口的大小。
int32_t (*lcd_set_window)(struct lcd_spec *self,
                uint16_t left, uint16_t top,
                    uint16_t right, uint16_t bottom);
//设置刷屏窗口的大小。
int32_t (*lcd_invalidate_rect)(struct lcd_spec *self,
                uint16_t left, uint16_t top,
                    uint16_t right, uint16_t bottom);
//设置刷屏窗口为全屏。
int32_t (*lcd_invalidate)(struct lcd_spec *self);
//lcd 读id
uint32_t (*lcd_readid)(struct lcd_spec *self);
```

2.3 LCD驱动函数接口

```
//设置坐标点旋转后的刷屏区域。
int32_t (*lcd_rotate_invalidate_rect)(struct lcd_spec *self,
                uint16_t left, uint16_t top,
                uint16_t right, uint16_t bottom,
                uint16_t angle);
//设置刷屏方向
int32_t (*lcd_set_direction)(struct lcd_spec *self, uint16_t direction);
//复位LCD。
int32_t (*lcd_reset)(struct lcd_spec *self);
```

2 Framebuffer驱动

1. FB驱动简介
2. SC8810 FB探测及电源管理函数接口
3. SC8810 FB与 lcd 驱动关系

2.1 Framebuffer 驱动简介

Framebuffer是内核当中提供的一种驱动程序接口，Linux是工作在保护模式下，所以用户态进程是无法直接写屏操作。因此Linux抽象出FrameBuffer这个设备来供用户态进程实现直接写屏。Framebuffer机制模仿显卡的功能，将显卡硬件结构抽象掉，可以通过Framebuffer的读写直接对显存进行操作。用户可以将Framebuffer看成是显示内存的一个映像，将其映射到进程地址空间之后，就可以直接进行读写操作，再通过LCDC将缓存中的数据反应在屏幕上。

这种操作是抽象的，统一的。用户不必关心物理显存的位置、换页机制等等具体细节，这些都是由Framebuffer设备驱动来完成的。但framebuffer本身不具备任何运算数据的能力,尽管Framebuffer需要真正的显卡驱动的支持，但所有显示任务都有CPU完成,因此CPU负担很重．

2.2 FB函数接口

1、Fb主处理文件fb_main.c,设置寄存器(命令/数据)接口

/kernel/drivers/video/sc8810/fb_main.c

```
static int32_t lcm_send_cmd (uint32_t cmd)
{
while(__raw_readl(LCM_CTRL) & BIT20);

__raw_writel(cmd, LCM_CD0);  //9bit

return 0;
}

static int32_t lcm_send_data (uint32_t data)
{
 while(__raw_readl(LCM_CTRL) & BIT20);

__raw_writel(data, LCM_DATA0);    //9bit

return 0;
}
```

2、HAL使用该设备
hardware\sprd\hsdroid\libgralloc\framebuffer.cpp
open(“/dev/graphics/fb0”,…)

2.2 FB函数接口(probe)

Framebuffer关联LCD接口

```
static int sc8810fb_probe(struct platform_device *pdev)
{
  ...
        lcd_adapt =  find_adapt_from_uboot(platform_data);  // 获取配置id号
if (lcd_adapt == -1) {
     info->need_reinit = 1;
     lcd_adapt = find_adapt_from_readid(info,platform_data);
}
if (lcd_adapt < 0) { // invalid index
     printk(KERN_ERR " can not read device id, and we will not refresh!\n");
     info->fb_state = FB_NO_REFRESH;
     lcd_adapt = 0;
}
   mount_panel(info, lcd_panel[lcd_adapt].panel); //根据读取的id关联lcd_xxxx.c的
}
```

2.2 FB函数接口(suspend)

/kernel/drivers/video/sc8810/fb_main.c

```
static int sc8810fb_suspend(struct platform_device *pdev,pm_message_t state)
{
        //调用lcd_enter_sleep指针函数进入睡眠
struct sc8810fb_info *info = platform_get_drvdata(pdev);
if (info->panel->ops->lcd_enter_sleep != NULL) {
     info->panel->ops->lcd_enter_sleep(info->panel,1);
}

      //判断并设置lcdc时钟disable,进入deepsleep状态
      if (info->clk_lcdc) {
     FB_PRINT("clk_disable(info->clk_lcdc)\n");
     clk_disable(info->clk_lcdc);
}
FB_PRINT("deep sleep: [%s]\n", __FUNCTION__);
return 0;
}
```

2.2 FB函数接口(resume)


```
static int sc8810fb_resume(struct platform_device *pdev)
{
struct sc8810fb_info *info = platform_get_drvdata(pdev);
if (info->clk_lcdc) {   //设置lcdc clock 进入enable状态
        FB_PRINT("clk_enable(info->clk_lcdc)\n");
        clk_enable(info->clk_lcdc);
}
    //重置lcdc、硬件初始化等
if (__raw_readl(LCDC_CTRL) == 0) { // resume from deep sleep
        info->need_reinit = 1;
        lcdc_reset();
        hw_early_init(info);
        hw_init(info);
        hw_later_init(info);
        info->need_reinit = 0;
} else {
        if(info->panel->ops->lcd_enter_sleep != NULL){
                info->panel->ops->lcd_enter_sleep(info->panel,0);
        }
}
return 0;
}
```

3 SC8810 LCD集成与客制化要点

SC8810 LCD驱动模块根据不同屏的厂家集成到spreadtrum平台

1. 客制化修改点
2. 编译与调试

谢谢