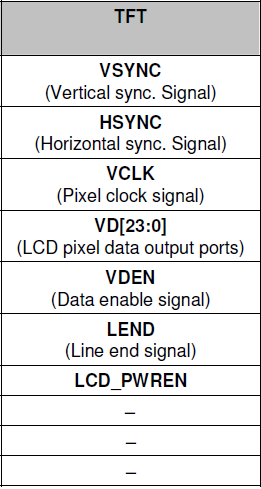
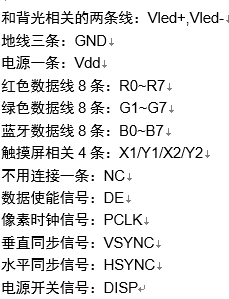
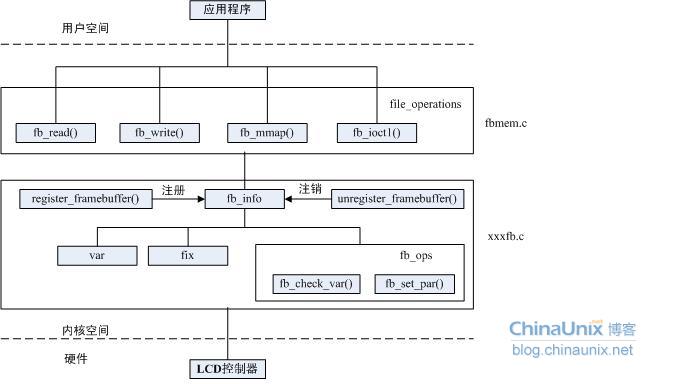
# Linux驱动\_LCD

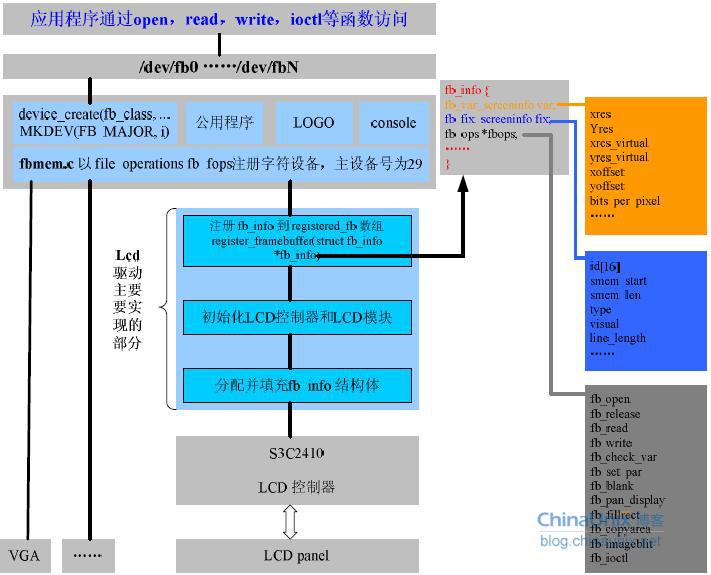
## 1. LCD物理连接

JZ2440上使用的LCD屏一共有40条线。其中：  
左边为LCD屏的pin脚，右边为CPU S3C2440的pin脚。  


其中LEDN信号不是必须的，这样两边的pin脚便可以对应起来。触摸屏另做处理。

## 2. LCD驱动架构图

[](http://blog.chinaunix.net/attachment/201209/3/27664726_1346683702gj8i.jpg)

[](http://blog.chinaunix.net/attachment/201209/4/27664726_1346764943DgTg.jpg)

## 3. LCD驱动代码分析

### 3.1 platform device注册

在和开发板相关的初始化函数smdk2440\_machine\_init中，会在总线上注册LCD platform device。

static struct platform\_device **\***smdk2440\_devices**[]** \_\_initdata **=** **{**

**&**s3c\_device\_usb**,**

**&**s3c\_device\_lcd**,**

**&**s3c\_device\_wdt**,**

**&**s3c\_device\_i2c**,**

**&**s3c\_device\_iis**,**

**&**s3c2440\_device\_sdi**,**

**};**

static void \_\_init smdk2440\_machine\_init**(**void**)**

**{**

s3c24xx\_fb\_set\_platdata**(&**smdk2440\_lcd\_cfg**);**

platform\_add\_devices**(**smdk2440\_devices**,** ARRAY\_SIZE**(**smdk2440\_devices**));**

smdk\_machine\_init**();**

**}**

其中lcd platform device相关代码如下：

#define IRQ\_LCD S3C2410\_IRQ(16) /\* 32 \*/

#define S3C24XX\_SZ\_LCD SZ\_1M

#define S3C24XX\_PA\_LCD S3C2410\_PA\_LCD

#define S3C2410\_PA\_LCD (0x4D000000)

static struct resource s3c\_lcd\_resource**[]** **=** **{**

**[**0**]** **=** **{**

**.**start **=** S3C24XX\_PA\_LCD**,**

**.**end **=** S3C24XX\_PA\_LCD **+** S3C24XX\_SZ\_LCD **-** 1**,**

**.**flags **=** IORESOURCE\_MEM**,**

**},**

**[**1**]** **=** **{**

**.**start **=** IRQ\_LCD**,**

**.**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

**}**

**};**

在上述代码中定义了lcd操作相关的寄存器地址和中断信息。

另外注意s3c24xx\_fb\_set\_platdata函数，该函数设置了platform device的platform\_data选项。

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"**);**

**}**

**}**

/\* 480x272 \*/

static struct s3c2410fb\_mach\_info smdk2440\_lcd\_cfg \_\_initdata **=** **{**

**.**regs **=** **{**

**.**lcdcon1 **=** S3C2410\_LCDCON1\_TFT16BPP **|** \

S3C2410\_LCDCON1\_TFT **|** \

S3C2410\_LCDCON1\_CLKVAL**(**0x04**),**

**.**lcdcon2 **=** S3C2410\_LCDCON2\_VBPD**(**1**)** **|** \

S3C2410\_LCDCON2\_LINEVAL**(**271**)** **|** \

S3C2410\_LCDCON2\_VFPD**(**1**)** **|** \

S3C2410\_LCDCON2\_VSPW**(**9**),**

**.**lcdcon3 **=** S3C2410\_LCDCON3\_HBPD**(**1**)** **|** \

S3C2410\_LCDCON3\_HOZVAL**(**479**)** **|** \

S3C2410\_LCDCON3\_HFPD**(**1**),**

**.**lcdcon4 **=** S3C2410\_LCDCON4\_HSPW**(**40**),**

**.**lcdcon5 **=** S3C2410\_LCDCON5\_FRM565 **|**

S3C2410\_LCDCON5\_INVVLINE **|**

S3C2410\_LCDCON5\_INVVFRAME **|**

S3C2410\_LCDCON5\_PWREN **|**

S3C2410\_LCDCON5\_HWSWP**,**

**},**

**.**gpccon **=** 0xaaaaaaaa**,**

**.**gpccon\_mask **=** 0xffffffff**,**

**.**gpcup **=** 0xffffffff**,**

**.**gpcup\_mask **=** 0xffffffff**,**

**.**gpdcon **=** 0xaaaaaaaa**,**

**.**gpdcon\_mask **=** 0xffffffff**,**

**.**gpdup **=** 0xffffffff**,**

**.**gpdup\_mask **=** 0xffffffff**,**

**.**fixed\_syncs **=** 1**,**

**.**type **=** S3C2410\_LCDCON1\_TFT**,**

**.**width **=** 480**,**

**.**height **=** 272**,**

**.**xres **=** **{**

**.**min **=** 480**,**

**.**max **=** 480**,**

**.**defval **=** 480**,**

**},**

**.**yres **=** **{**

**.**max **=** 272**,**

**.**min **=** 272**,**

**.**defval **=** 272**,**

**},**

**.**bpp **=** **{**

**.**min **=** 16**,**

**.**max **=** 16**,**

**.**defval **=** 16**,**

**},**

**};**

### 3.2 platform driver注册

int \_\_devinit s3c2410fb\_init**(**void**)**

**{**

**return** platform\_driver\_register**(&**s3c2410fb\_driver**);**

**}**

static void \_\_exit s3c2410fb\_cleanup**(**void**)**

**{**

platform\_driver\_unregister**(&**s3c2410fb\_driver**);**

**}**

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**,**

**},**

**};**

### 3.3 probe函数

当platform device和platform driver匹配上时，s3c2410fb\_probe函数被调用。

(1)从lcd注册的platform\_device中获取一些有用的信息，操作寄存器和中断号。

//获取s3c24xx\_fb\_set\_platdata设置的smdk2440\_lcd\_cfg结构体

mach\_info **=** pdev**->**dev**.**platform\_data**;**

**if** **(**mach\_info **==** **NULL)** **{**

dev\_err**(&**pdev**->**dev**,**"no platform data for lcd, cannot attach\n"**);**

**return** **-**EINVAL**;**

**}**

//获取控制lcd的5个寄存器lcdcon1~lcdcon5内容

mregs **=** **&**mach\_info**->**regs**;**

//获取中断号

irq **=** platform\_get\_irq**(**pdev**,** 0**);**

**if** **(**irq **<** 0**)** **{**

dev\_err**(&**pdev**->**dev**,** "no irq for device\n"**);**

**return** **-**ENOENT**;**

**}**

(2)为fd\_info结构体分配内存空间

//分配一个fb\_info结构体,第一参数不为0表示，额外多申请的空间

//用来存放额外的数据，这里用来存放s3c2410fb\_info额外的数据

//比如:clk，resource，io,irq\_base,drv\_type等额外信息

fbinfo **=** framebuffer\_alloc**(sizeof(**struct s3c2410fb\_info**),** **&**pdev**->**dev**);**

**if** **(!**fbinfo**)** **{**

**return** **-**ENOMEM**;**

**}**

framebuffer\_alloc源码如下：

struct fb\_info **\***framebuffer\_alloc**(**size\_t size**,** struct device **\***dev**)**

**{**

//计算一个long型多少个Byte

#define BYTES\_PER\_LONG (BITS\_PER\_LONG/8)

//计算fb\_info结构体按Long型对齐还差几个字节

#define PADDING (BYTES\_PER\_LONG - (sizeof(struct fb\_info) % BYTES\_PER\_LONG))

//计算fb\_info的大小

int fb\_info\_size **=** **sizeof(**struct fb\_info**);**

struct fb\_info **\***info**;**

char **\***p**;**

//假如size不为0，fb\_info的长度需要按四字节对齐

**if** **(**size**)**

fb\_info\_size **+=** PADDING**;**

//开辟fb\_info及其额外内容的空间

p **=** kzalloc**(**fb\_info\_size **+** size**,** GFP\_KERNEL**);**

**if** **(!**p**)**

**return** **NULL;**

info **=** **(**struct fb\_info **\*)** p**;**

//额外的开辟的空间指向fb\_info的par指针

**if** **(**size**)**

info**->**par **=** p **+** fb\_info\_size**;**

//platform\_device的device结构体赋给fb\_info的device指针

info**->**device **=** dev**;**

#ifdef CONFIG\_FB\_BACKLIGHT

mutex\_init**(&**info**->**bl\_curve\_mutex**);**

#endif

**return** info**;**

#undef PADDING

#undef BYTES\_PER\_LONG

**}**

(3)设置一些指针

//info指向了创建fb\_info时额外多申请内存空间的首地址

info **=** fbinfo**->**par**;**

//info的fb指针设置为fbinfo地址

info**->**fb **=** fbinfo**;**

//info的dev指向platform\_device的dev

info**->**dev **=** **&**pdev**->**dev**;**

//相当于pdev->dev->driver\_data = fbinfo

platform\_set\_drvdata**(**pdev**,** fbinfo**);**

(4)设置fb\_info的相关参数，关闭lcd信号输出

//设置identification string为s3c2410fb

strcpy**(**fbinfo**->**fix**.**id**,** driver\_name**);**

//获取lcdcon1~lcdcon5的内容

memcpy**(&**info**->**regs**,** **&**mach\_info**->**regs**,** **sizeof(**info**->**regs**));**

//清空lcdcon1的ENVID位，ENVID：LCD信号输出使能位

info**->**regs**.**lcdcon1 **&=** **~**S3C2410\_LCDCON1\_ENVID**;**

//读出目前lcdcon1寄存器内容

lcdcon1 **=** readl**(**S3C2410\_LCDCON1**);**

//LCD信号输出禁止

writel**(**lcdcon1 **&** **~**S3C2410\_LCDCON1\_ENVID**,** S3C2410\_LCDCON1**);**

//关闭背光

s3c2410\_gpio\_setpin**(**S3C2410\_GPB0**,** 0**);** // back light control

//s3c2410fb\_info的mach\_info设置为smdk2440\_lcd\_cfg

info**->**mach\_info **=** pdev**->**dev**.**platform\_data**;**

//设置framebuffer类型 Packed Pixels

fbinfo**->**fix**.**type **=** FB\_TYPE\_PACKED\_PIXELS**;**

fbinfo**->**fix**.**type\_aux **=** 0**;**

fbinfo**->**fix**.**xpanstep **=** 0**;**

fbinfo**->**fix**.**ypanstep **=** 0**;**

fbinfo**->**fix**.**ywrapstep **=** 0**;**

//无硬件加速

fbinfo**->**fix**.**accel **=** FB\_ACCEL\_NONE**;**

fbinfo**->**var**.**nonstd **=** 0**;**

//set values immediately

fbinfo**->**var**.**activate **=** FB\_ACTIVATE\_NOW**;**

//smdk2440\_lcd\_cfg height 272

fbinfo**->**var**.**height **=** mach\_info**->**height**;**

//smdk2440\_lcd\_cfg weight 480

fbinfo**->**var**.**width **=** mach\_info**->**width**;**

fbinfo**->**var**.**accel\_flags **=** 0**;**

fbinfo**->**var**.**vmode **=** FB\_VMODE\_NONINTERLACED**;**

//设置和framebuffer相关的操作函数

fbinfo**->**fbops **=** **&**s3c2410fb\_ops**;**

fbinfo**->**flags **=** FBINFO\_FLAG\_DEFAULT**;**

//设置假调色板

fbinfo**->**pseudo\_palette **=** **&**info**->**pseudo\_pal**;**

//480

fbinfo**->**var**.**xres **=** mach\_info**->**xres**.**defval**;**

//480

fbinfo**->**var**.**xres\_virtual **=** mach\_info**->**xres**.**defval**;**

//272

fbinfo**->**var**.**yres **=** mach\_info**->**yres**.**defval**;**

//272

fbinfo**->**var**.**yres\_virtual **=** mach\_info**->**yres**.**defval**;**

//16 每个像素16bit

fbinfo**->**var**.**bits\_per\_pixel **=** mach\_info**->**bpp**.**defval**;**

//根据VBPD等值设置对应的参数

//6

fbinfo**->**var**.**upper\_margin **=** S3C2410\_LCDCON2\_GET\_VBPD**(**mregs**->**lcdcon2**)** **+** 1**;**

//4

fbinfo**->**var**.**lower\_margin **=** S3C2410\_LCDCON2\_GET\_VFPD**(**mregs**->**lcdcon2**)** **+** 1**;**

//2

fbinfo**->**var**.**vsync\_len **=** S3C2410\_LCDCON2\_GET\_VSPW**(**mregs**->**lcdcon2**)** **+** 1**;**

//2

fbinfo**->**var**.**left\_margin **=** S3C2410\_LCDCON3\_GET\_HFPD**(**mregs**->**lcdcon3**)** **+** 1**;**

//11

fbinfo**->**var**.**right\_margin **=** S3C2410\_LCDCON3\_GET\_HBPD**(**mregs**->**lcdcon3**)** **+** 1**;**

//1

fbinfo**->**var**.**hsync\_len **=** S3C2410\_LCDCON4\_GET\_HSPW**(**mregs**->**lcdcon4**)** **+** 1**;**

//设置红绿蓝数据在16个bit中的偏移和大小

fbinfo**->**var**.**red**.**offset **=** 11**;**

fbinfo**->**var**.**green**.**offset **=** 5**;**

fbinfo**->**var**.**blue**.**offset **=** 0**;**

fbinfo**->**var**.**transp**.**offset **=** 0**;**

fbinfo**->**var**.**red**.**length **=** 5**;**

fbinfo**->**var**.**green**.**length **=** 6**;**

fbinfo**->**var**.**blue**.**length **=** 5**;**

fbinfo**->**var**.**transp**.**length **=** 0**;**

//480\*272所占用的内存大小480\*272\*2

fbinfo**->**fix**.**smem\_len **=** mach\_info**->**xres**.**max **\***

mach\_info**->**yres**.**max **\***

mach\_info**->**bpp**.**max **/** 8**;**

//palette\_buffer[i] = 0x80000000，清空调色板

**for** **(**i **=** 0**;** i **<** 256**;** i**++)**

info**->**palette\_buffer**[**i**]** **=** PALETTE\_BUFF\_CLEAR**;**

其中S3C2410\_LCDCON1定义如下,开机的时候会完成静态映射。

static struct map\_desc s3c244x\_iodesc**[]** \_\_initdata **=** **{**

IODESC\_ENT**(**CLKPWR**),**

IODESC\_ENT**(**TIMER**),**

IODESC\_ENT**(**WATCHDOG**),**

IODESC\_ENT**(**LCD**),**

**};**

#define S3C2410\_LCDCON1 S3C2410\_LCDREG(0x00)

#define S3C2410\_LCDREG(x) ((x) + S3C24XX\_VA\_LCD)

#define S3C24XX\_VA\_LCD S3C2410\_ADDR(0x00300000)

#define S3C2410\_ADDR(x) ((void \_\_iomem \_\_force \*)0xF0000000 + (x))

s3c2410fb\_ops定义如下：

static struct fb\_ops s3c2410fb\_ops **=** **{**

**.**owner          **=** THIS\_MODULE**,**

**.**fb\_check\_var   **=** s3c2410fb\_check\_var**,** //设置可变参数

**.**fb\_set\_par     **=** s3c2410fb\_set\_par**,** //设置固定参数及lcdcon寄存器

**.**fb\_blank       **=** s3c2410fb\_blank**,**   //设置是否使能LCD控制器

**.**fb\_setcolreg   **=** s3c2410fb\_setcolreg**,** //设置RGB颜色，实现伪颜色表

**.**fb\_fillrect    **=** cfb\_fillrect**,**       //画一个矩形

**.**fb\_copyarea    **=** cfb\_copyarea**,**//Copy data from area to another

**.**fb\_imageblit   **=** cfb\_imageblit**,** //Draws a image to the display

**};**

(5)标记LCD要使用的物理地址，并申请LCD中断

//标记S3C24XX\_VA\_LCD后的1M空间被使用了

**if** **(!**request\_mem\_region**((**unsigned long**)**S3C24XX\_VA\_LCD**,** SZ\_1M**,** "s3c2410-lcd"**))** **{**

ret **=** **-**EBUSY**;**

**goto** dealloc\_fb**;**

**}**

dprintk**(**"got LCD region\n"**);**

//申请lcd中断，处理函数为s3c2410fb\_irq，私有信息为s3c2410fb\_info

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\_mem**;**

**}**

(6)使能时钟

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**;**

**}**

clk\_enable**(**info**->**clk**);**

lcd的时钟会在开机的时候加入到一个链表中。使能时钟就是将S3C2410\_CLKCON寄存器LCD对应的bit位置1。

**.**name **=** "lcd"**,**

**.**id **=** **-**1**,**

**.**parent **=** **&**clk\_h**,**

**.**enable **=** s3c2410\_clkcon\_enable**,**

**.**ctrlbit **=** S3C2410\_CLKCON\_LCDC**,**

int clk\_enable**(**struct clk **\***clk**)**

**{**

**if** **(**IS\_ERR**(**clk**)** **||** clk **==** **NULL)**

**return** **-**EINVAL**;**

clk\_enable**(**clk**->**parent**);**

mutex\_lock**(&**clocks\_mutex**);**

**if** **((**clk**->**usage**++)** **==** 0**)**

**(**clk**->**enable**)(**clk**,** 1**);**

mutex\_unlock**(&**clocks\_mutex**);**

**return** 0**;**

**}**

int s3c2410\_clkcon\_enable**(**struct clk **\***clk**,** int enable**)**

**{**

unsigned int clocks **=** clk**->**ctrlbit**;**

unsigned long clkcon**;**

clkcon **=** \_\_raw\_readl**(**S3C2410\_CLKCON**);**

**if** **(**enable**)**

clkcon **|=** clocks**;**

**else**

clkcon **&=** **~**clocks**;**

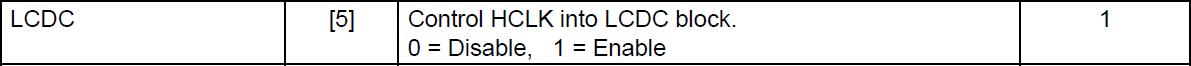
/\* ensure none of the special function bits set \*/

clkcon **&=** **~(**S3C2410\_CLKCON\_IDLE**|**S3C2410\_CLKCON\_POWER**);**

\_\_raw\_writel**(**clkcon**,** S3C2410\_CLKCON**);**

**return** 0**;**

**}**



(7)分配显存

/\* Initialize video memory \*/

ret **=** s3c2410fb\_map\_video\_memory**(**info**);**

**if** **(**ret**)** **{**

printk**(** KERN\_ERR "Failed to allocate video RAM: %d\n"**,** ret**);**

ret **=** **-**ENOMEM**;**

**goto** release\_clock**;**

**}**

dprintk**(**"got video memory\n"**);**

具体实现代码如下：

static int \_\_init s3c2410fb\_map\_video\_memory**(**struct s3c2410fb\_info **\***fbi**)**

**{**

dprintk**(**"map\_video\_memory(fbi=%p)\n"**,** fbi**);**

//framebuffer缓存大小，多分配PAGE\_SIZE

fbi**->**map\_size **=** PAGE\_ALIGN**(**fbi**->**fb**->**fix**.**smem\_len **+** PAGE\_SIZE**);**

//A = dma\_alloc\_writecombine(B,C,D,GFP\_KERNEL);

//A：内存的虚拟起始地址，在内核要用此地址来操作所分配的内存

//B: struct device指针，可以平台初始化里指定，主要是dma\_mask之类

//dma\_mask是设备DMA能访问的内存范围

//C: 实际分配大小，传入dma\_map\_size即可

//D: 返回的内存物理地址，dma就可以用

//A和D是一一对应的，只不过，A是虚拟地址，而D是物理地址。

//dma\_alloc\_writecombine分配的内存物理上是连续的并禁止C(Cacheable)域.

//dma\_alloc\_writecombine分配出来的内存不使用缓存，但是会使用写缓冲区。

fbi**->**map\_cpu **=** dma\_alloc\_writecombine**(**fbi**->**dev**,** fbi**->**map\_size**,**

**&**fbi**->**map\_dma**,** GFP\_KERNEL**);**

fbi**->**map\_size **=** fbi**->**fb**->**fix**.**smem\_len**;**

**if** **(**fbi**->**map\_cpu**)** **{**

/\* prevent initial garbage on screen \*/

dprintk**(**"map\_video\_memory: clear %p:%08x\n"**,**

fbi**->**map\_cpu**,** fbi**->**map\_size**);**

memset**(**fbi**->**map\_cpu**,** 0xf0**,** fbi**->**map\_size**);**

fbi**->**screen\_dma **=** fbi**->**map\_dma**;**

fbi**->**fb**->**screen\_base **=** fbi**->**map\_cpu**;**

fbi**->**fb**->**fix**.**smem\_start **=** fbi**->**screen\_dma**;**

dprintk**(**"map\_video\_memory: dma=%08x cpu=%p size=%08x\n"**,**

fbi**->**map\_dma**,** fbi**->**map\_cpu**,** fbi**->**fb**->**fix**.**smem\_len**);**

**}**

**return** fbi**->**map\_cpu **?** 0 **:** **-**ENOMEM**;**

**}**

(8)设置寄存器

ret **=** s3c2410fb\_init\_registers**(**info**);**

s3c2410fb\_init\_registers代码如下：

static int s3c2410fb\_init\_registers**(**struct s3c2410fb\_info **\***fbi**)**

**{**

unsigned long flags**;**

local\_irq\_save**(**flags**);**

//设置lcd对应的gpio口为lcd总线

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**);**

local\_irq\_restore**(**flags**);**

//设置lcd寄存器

writel**(**fbi**->**regs**.**lcdcon1**,** S3C2410\_LCDCON1**);**

writel**(**fbi**->**regs**.**lcdcon2**,** S3C2410\_LCDCON2**);**

writel**(**fbi**->**regs**.**lcdcon3**,** S3C2410\_LCDCON3**);**

writel**(**fbi**->**regs**.**lcdcon4**,** S3C2410\_LCDCON4**);**

writel**(**fbi**->**regs**.**lcdcon5**,** S3C2410\_LCDCON5**);**

//设置帧缓冲寄存器，把分配的显存物理地址告知

s3c2410fb\_set\_lcdaddr**(**fbi**);**

dprintk**(**"LPCSEL = 0x%08lx\n"**,** mach\_info**->**lpcsel**);**

writel**(**mach\_info**->**lpcsel**,** S3C2410\_LPCSEL**);**

//停止临时调色板

dprintk**(**"replacing TPAL %08x\n"**,** readl**(**S3C2410\_TPAL**));**

writel**(**0x00**,** S3C2410\_TPAL**);**

//打开背光控制

s3c2410\_gpio\_cfgpin**(**S3C2410\_GPB0**,** S3C2410\_GPB0\_OUTP**);** // back light control

s3c2410\_gpio\_pullup**(**S3C2410\_GPB0**,** 0**);**

s3c2410\_gpio\_setpin**(**S3C2410\_GPB0**,** 1**);** // back light control, enable

/\* probably not required \*/

msleep**(**10**);**

//LCD信号输出使能

fbi**->**regs**.**lcdcon1 **|=** S3C2410\_LCDCON1\_ENVID**;**

writel**(**fbi**->**regs**.**lcdcon1**,** S3C2410\_LCDCON1**);**

**return** 0**;**

**}**

s3c2410fb\_set\_lcdaddr代码如下：

static void s3c2410fb\_set\_lcdaddr**(**struct s3c2410fb\_info **\***fbi**)**

**{**

struct fb\_var\_screeninfo **\***var **=** **&**fbi**->**fb**->**var**;**

unsigned long saddr1**,** saddr2**,** saddr3**;**

//这里使用的都是分配出来的物理地址

saddr1 **=** fbi**->**fb**->**fix**.**smem\_start **>>** 1**;**

saddr2 **=** fbi**->**fb**->**fix**.**smem\_start**;**

saddr2 **+=** **(**var**->**xres **\*** var**->**yres **\*** var**->**bits\_per\_pixel**)/**8**;**

saddr2**>>=** 1**;**

saddr3 **=** S3C2410\_OFFSIZE**(**0**)** **|** S3C2410\_PAGEWIDTH**((**var**->**xres **\*** var**->**bits\_per\_pixel **/** 16**)** **&** 0x7ff**);**

dprintk**(**"LCDSADDR1 = 0x%08lx\n"**,** saddr1**);**

dprintk**(**"LCDSADDR2 = 0x%08lx\n"**,** saddr2**);**

dprintk**(**"LCDSADDR3 = 0x%08lx\n"**,** saddr3**);**

writel**(**saddr1**,** S3C2410\_LCDSADDR1**);**

writel**(**saddr2**,** S3C2410\_LCDSADDR2**);**

writel**(**saddr3**,** S3C2410\_LCDSADDR3**);**

**}**

(9)检查fb\_info中的var参数

ret **=** s3c2410fb\_check\_var**(&**fbinfo**->**var**,** fbinfo**);**

(10)向framebuffer子系统注册自己

ret **=** register\_framebuffer**(**fbinfo**);**

**if** **(**ret **<** 0**)** **{**

printk**(**KERN\_ERR "Failed to register framebuffer device: %d\n"**,** ret**);**

**goto** free\_video\_memory**;**

**}**

(11)创建/sys/devices/platform/s3c2410-lcd/debug文件，方便调试。

device\_create\_file**(&**pdev**->**dev**,** **&**dev\_attr\_debug**);**

相关代码如下：

static DEVICE\_ATTR**(**debug**,** 0666**,**

s3c2410fb\_debug\_show**,**

s3c2410fb\_debug\_store**);**

static int s3c2410fb\_debug\_show**(**struct device **\***dev**,** struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

**return** snprintf**(**buf**,** PAGE\_SIZE**,** "%s\n"**,** debug **?** "on" **:** "off"**);**

**}**

static int s3c2410fb\_debug\_store**(**struct device **\***dev**,** struct device\_attribute **\***attr**,**

const char **\***buf**,** size\_t len**)**

**{**

**if** **(**mach\_info **==** **NULL)**

**return** **-**EINVAL**;**

**if** **(**len **<** 1**)**

**return** **-**EINVAL**;**

**if** **(**strnicmp**(**buf**,** "on"**,** 2**)** **==** 0 **||**

strnicmp**(**buf**,** "1"**,** 1**)** **==** 0**)** **{**

debug **=** 1**;**

printk**(**KERN\_DEBUG "s3c2410fb: Debug On"**);**

**}** **else** **if** **(**strnicmp**(**buf**,** "off"**,** 3**)** **==** 0 **||**

strnicmp**(**buf**,** "0"**,** 1**)** **==** 0**)** **{**

debug **=** 0**;**

printk**(**KERN\_DEBUG "s3c2410fb: Debug Off"**);**

**}** **else** **{**

**return** **-**EINVAL**;**

**}**

**return** len**;**

**}**

可以通过操作/sys/devices/platform/s3c2410-lcd/debug打开和关闭debug功能。  
查询当前状态：

# cat /sys/devices/platform/s3c2410-lcd/debug

off

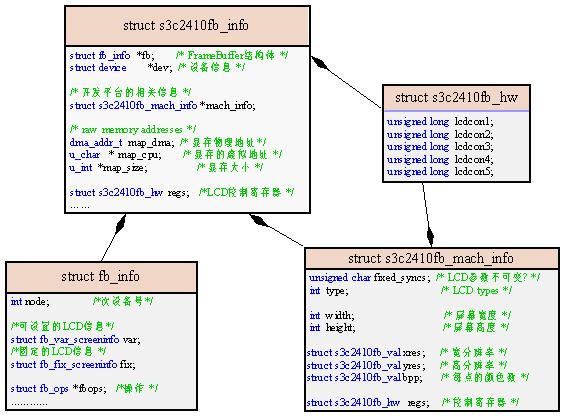
改变状态：

# echo 1 > /sys/devices/platform/s3c2410-lcd/debug

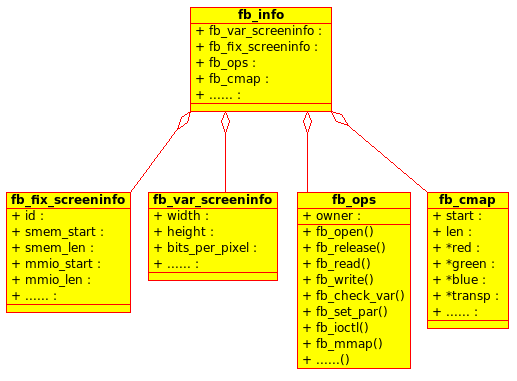
# cat /sys/devices/platform/s3c2410-lcd/debug

on

在probe中出现的几个lcd结构体关系如下：



fb\_info结构体内容：

****

fb\_var\_screeninfo：描述了一种显卡显示模式的所有信息，如宽、高、颜色深度等，不同的显示模式对应不同的信息；

fb\_fix\_screeninfo：定义了显卡信息，如 framebuffer 内存的起始地址，地址长度等；

fb\_cmap：设备独立的 colormap 信息，可以通过 ioctl 的 FBIOGETCMAP 和 FBIOPUTCMAP 命令设置 colormap；

fb\_info：包含当前 video card 的状态信息，只有 fb\_info 对内核可见；

fb\_ops ： 应用程序使用 ioctl 系统调用操作底层的 LCD 硬件，fb\_ops 结构中定义的方法用于支持这些操作；

## 4.framebuffer子系统

### 4.1 framebuffer字符设备注册

static int \_\_init fbmem\_init**(**void**)**

**{**

//建立proc只读项

// /proc/fb 帧缓冲设备列表，包括数量和控制它的驱动

create\_proc\_read\_entry**(**"fb"**,** 0**,** **NULL,** fbmem\_read\_proc**,** **NULL);**

//创建字符设备，主设备号29

**if** **(**register\_chrdev**(**FB\_MAJOR**,**"fb"**,&**fb\_fops**))**

printk**(**"unable to get major %d for fb devs\n"**,** FB\_MAJOR**);**

//在/sys/class目录下创建了一个graphics目录，主要是描述内核的图形系统

fb\_class **=** class\_create**(**THIS\_MODULE**,** "graphics"**);**

**if** **(**IS\_ERR**(**fb\_class**))** **{**

printk**(**KERN\_WARNING "Unable to create fb class; errno = %ld\n"**,** PTR\_ERR**(**fb\_class**));**

fb\_class **=** **NULL;**

**}**

**return** 0**;**

**}**

其中字符设备的的操作函数如下,后面会进行分析。

static const struct file\_operations fb\_fops **=** **{**

**.**owner **=** THIS\_MODULE**,**

**.**read **=** fb\_read**,**

**.**write **=** fb\_write**,**

**.**ioctl **=** fb\_ioctl**,**

#ifdef CONFIG\_COMPAT

**.**compat\_ioctl **=** fb\_compat\_ioctl**,**

#endif

**.**mmap **=** fb\_mmap**,**

**.**open **=** fb\_open**,**

**.**release **=** fb\_release**,**

#ifdef HAVE\_ARCH\_FB\_UNMAPPED\_AREA

**.**get\_unmapped\_area **=** get\_fb\_unmapped\_area**,**

#endif

#ifdef CONFIG\_FB\_DEFERRED\_IO

**.**fsync **=** fb\_deferred\_io\_fsync**,**

#endif

**};**

### 4.2 register\_framebuffer

lcd驱动最关键的操作就是将自己写的fb\_info注册到framebuffer子系统中，那么register\_framebuffer做了点什么呢？

int register\_framebuffer**(**struct fb\_info **\***fb\_info**)**

**{**

int i**;**

struct fb\_event event**;**

struct fb\_videomode mode**;**

//framebuffer最多支持32个从设备

**if** **(**num\_registered\_fb **==** FB\_MAX**)**

**return** **-**ENXIO**;**

num\_registered\_fb**++;**

**for** **(**i **=** 0 **;** i **<** FB\_MAX**;** i**++)**

**if** **(!**registered\_fb**[**i**])**

**break;**

//将registered\_fb节点号赋给fb\_info->node

fb\_info**->**node **=** i**;**

//自动创建/dev/fbx节点

//在/sys/class/graphics/设备类中创建一个设备fbx

fb\_info**->**dev **=** device\_create**(**fb\_class**,** fb\_info**->**device**,**

MKDEV**(**FB\_MAJOR**,** i**),** "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**

//在/sys/class/graphics/fbx下创建blank,virtual\_size等文件

fb\_init\_device**(**fb\_info**);**

//Image hardware mapper

//分配8k的空间

**if** **(**fb\_info**->**pixmap**.**addr **==** **NULL)** **{**

fb\_info**->**pixmap**.**addr **=** kmalloc**(**FBPIXMAPSIZE**,** GFP\_KERNEL**);**

**if** **(**fb\_info**->**pixmap**.**addr**)** **{**

fb\_info**->**pixmap**.**size **=** FBPIXMAPSIZE**;**

fb\_info**->**pixmap**.**buf\_align **=** 1**;**

fb\_info**->**pixmap**.**scan\_align **=** 1**;**

fb\_info**->**pixmap**.**access\_align **=** 32**;**

fb\_info**->**pixmap**.**flags **=** FB\_PIXMAP\_DEFAULT**;**

**}**

**}**

fb\_info**->**pixmap**.**offset **=** 0**;**

**if** **(!**fb\_info**->**pixmap**.**blit\_x**)**

fb\_info**->**pixmap**.**blit\_x **=** **~(**u32**)**0**;**

**if** **(!**fb\_info**->**pixmap**.**blit\_y**)**

fb\_info**->**pixmap**.**blit\_y **=** **~(**u32**)**0**;**

**if** **(!**fb\_info**->**modelist**.**prev **||** **!**fb\_info**->**modelist**.**next**)**

INIT\_LIST\_HEAD**(&**fb\_info**->**modelist**);**

//convert fb\_var\_screeninfo to fb\_videomode

fb\_var\_to\_videomode**(&**mode**,** **&**fb\_info**->**var**);**

//将fb\_videomode添加到modelist

fb\_add\_videomode**(&**mode**,** **&**fb\_info**->**modelist**);**

//将fb\_info添加到registered\_fb数组

registered\_fb**[**i**]** **=** fb\_info**;**

//使用Linux事件通知机制发送一个FrameBuffer注册事件FB\_EVENT\_FB\_REGISTERED

event**.**info **=** fb\_info**;**

fb\_notifier\_call\_chain**(**FB\_EVENT\_FB\_REGISTERED**,** **&**event**);**

**return** 0**;**

**}**

从源码分析，register\_framebuffer主要完成了/dev/fbx的添加，并将fb\_info添加到registered\_fb数组中。

fb\_init\_device代码如下，在/sys/class/graphics/fbx下创建blank,virtual\_size等文件,方便调试。

int fb\_init\_device**(**struct fb\_info **\***fb\_info**)**

**{**

int i**,** error **=** 0**;**

dev\_set\_drvdata**(**fb\_info**->**dev**,** fb\_info**);**

fb\_info**->**class\_flag **|=** FB\_SYSFS\_FLAG\_ATTR**;**

**for** **(**i **=** 0**;** i **<** ARRAY\_SIZE**(**device\_attrs**);** i**++)** **{**

error **=** device\_create\_file**(**fb\_info**->**dev**,** **&**device\_attrs**[**i**]);**

**if** **(**error**)**

**break;**

**}**

**if** **(**error**)** **{**

**while** **(--**i **>=** 0**)**

device\_remove\_file**(**fb\_info**->**dev**,** **&**device\_attrs**[**i**]);**

fb\_info**->**class\_flag **&=** **~**FB\_SYSFS\_FLAG\_ATTR**;**

**}**

**return** 0**;**

**}**

static struct device\_attribute device\_attrs**[]** **=** **{**

\_\_ATTR**(**bits\_per\_pixel**,** S\_IRUGO**|**S\_IWUSR**,** show\_bpp**,** store\_bpp**),**

\_\_ATTR**(**blank**,** S\_IRUGO**|**S\_IWUSR**,** show\_blank**,** store\_blank**),**

\_\_ATTR**(**console**,** S\_IRUGO**|**S\_IWUSR**,** show\_console**,** store\_console**),**

\_\_ATTR**(**cursor**,** S\_IRUGO**|**S\_IWUSR**,** show\_cursor**,** store\_cursor**),**

\_\_ATTR**(**mode**,** S\_IRUGO**|**S\_IWUSR**,** show\_mode**,** store\_mode**),**

\_\_ATTR**(**modes**,** S\_IRUGO**|**S\_IWUSR**,** show\_modes**,** store\_modes**),**

\_\_ATTR**(**pan**,** S\_IRUGO**|**S\_IWUSR**,** show\_pan**,** store\_pan**),**

\_\_ATTR**(**virtual\_size**,** S\_IRUGO**|**S\_IWUSR**,** show\_virtual**,** store\_virtual**),**

\_\_ATTR**(**name**,** S\_IRUGO**,** show\_name**,** **NULL),**

\_\_ATTR**(**stride**,** S\_IRUGO**,** show\_stride**,** **NULL),**

\_\_ATTR**(**rotate**,** S\_IRUGO**|**S\_IWUSR**,** show\_rotate**,** store\_rotate**),**

\_\_ATTR**(**state**,** S\_IRUGO**|**S\_IWUSR**,** show\_fbstate**,** store\_fbstate**),**

#ifdef CONFIG\_FB\_BACKLIGHT

\_\_ATTR**(**bl\_curve**,** S\_IRUGO**|**S\_IWUSR**,** show\_bl\_curve**,** store\_bl\_curve**),**

#endif

**};**

### 4.3 fb\_open

当执行完register\_framebuffer后，会产生/dev/fbx文件，当我们打开它时，执行的是fb\_open函数。

static int fb\_open**(**struct inode **\***inode**,** struct file **\***file**)**

**{**

//获取从设备号

int fbidx **=** iminor**(**inode**);**

struct fb\_info **\***info**;**

int res **=** 0**;**

**if** **(**fbidx **>=** FB\_MAX**)**

**return** **-**ENODEV**;**

#ifdef CONFIG\_KMOD

//获取对应的fb\_info

**if** **(!(**info **=** registered\_fb**[**fbidx**]))**

try\_to\_load**(**fbidx**);**

#endif /\* CONFIG\_KMOD \*/

**if** **(!(**info **=** registered\_fb**[**fbidx**]))**

**return** **-**ENODEV**;**

**if** **(!**try\_module\_get**(**info**->**fbops**->**owner**))**

**return** **-**ENODEV**;**

//将fb\_info设置为私有数据

file**->**private\_data **=** info**;**

//调用fb\_info的fb\_open函数

**if** **(**info**->**fbops**->**fb\_open**)** **{**

res **=** info**->**fbops**->**fb\_open**(**info**,**1**);**

**if** **(**res**)**

module\_put**(**info**->**fbops**->**owner**);**

**}**

**return** res**;**

**}**

实际上fb\_open调用的fb\_info中fbops的open函数，但是open函数在fbops未定义。

### 4.4 fb\_write

fb\_write实际上将用户层传来的数据写入到驱动分配的显存中。

static ssize\_t

fb\_write**(**struct file **\***file**,** const char \_\_user **\***buf**,** size\_t count**,** loff\_t **\***ppos**)**

**{**

unsigned long p **=** **\***ppos**;**

//获取子设备号

struct inode **\***inode **=** file**->**f\_path**.**dentry**->**d\_inode**;**

int fbidx **=** iminor**(**inode**);**

//通过fbidx索引registered\_fb得到设备的fb\_info结构体实例

struct fb\_info **\***info **=** registered\_fb**[**fbidx**];**

u32 **\***buffer**,** **\***src**;**

u32 \_\_iomem **\***dst**;**

int c**,** i**,** cnt **=** 0**,** err **=** 0**;**

unsigned long total\_size**;**

//若info不存在 或 DMA缓冲区的虚拟地址不存在，则出错返回

**if** **(!**info **||** **!**info**->**screen\_base**)**

**return** **-**ENODEV**;**

**if** **(**info**->**state **!=** FBINFO\_STATE\_RUNNING**)**

**return** **-**EPERM**;**

//若设备fb\_ops操作集中有fb\_write函数，则调用之，这里不存在

**if** **(**info**->**fbops**->**fb\_write**)**

**return** info**->**fbops**->**fb\_write**(**info**,** buf**,** count**,** ppos**);**

total\_size **=** info**->**screen\_size**;**

//total\_size为0的话，取 info->fix.smem\_len

**if** **(**total\_size **==** 0**)**

total\_size **=** info**->**fix**.**smem\_len**;**

//若 写入偏移量 > total\_size，则出错返回。

**if** **(**p **>** total\_size**)**

**return** **-**EFBIG**;**

//若 写入数目 > total\_size，则修正为total\_size

**if** **(**count **>** total\_size**)** **{**

err **=** **-**EFBIG**;**

count **=** total\_size**;**

**}**

//若 写入数目 + 偏移量 > total\_size，则 写入数目 修正为 total\_size - 偏移量

**if** **(**count **+** p **>** total\_size**)** **{**

**if** **(!**err**)**

err **=** **-**ENOSPC**;**

count **=** total\_size **-** p**;**

**}**

//动态申请内存，用于缓冲数据，最大申请PAGE\_SIZE大小，后面会分段传输

buffer **=** kmalloc**((**count **>** PAGE\_SIZE**)** **?** PAGE\_SIZE **:** count**,**

GFP\_KERNEL**);**

**if** **(!**buffer**)**

**return** **-**ENOMEM**;**

//DMA缓冲区的首地址 + 偏移量 设为目的地址

dst **=** **(**u32 \_\_iomem **\*)** **(**info**->**screen\_base **+** p**);**

//若设备fb\_ops操作集中有fb\_sync函数，则调用之

**if** **(**info**->**fbops**->**fb\_sync**)**

info**->**fbops**->**fb\_sync**(**info**);**

**while** **(**count**)** **{**

//缓冲大小为PAGE\_SIZE，若count较大，则分段。

c **=** **(**count **>** PAGE\_SIZE**)** **?** PAGE\_SIZE **:** count**;**

//源地址为缓冲区首地址

src **=** buffer**;**

//读取一个段数据

**if** **(**copy\_from\_user**(**src**,** buf**,** c**))** **{**

err **=** **-**EFAULT**;**

**break;**

**}**

//每次写4字节

**for** **(**i **=** c **>>** 2**;** i**--;** **)**

fb\_writel**(\***src**++,** dst**++);**

//对于最后小于4字节的数据，进行特别操作，每次传输1字节

**if** **(**c **&** 3**)** **{**

u8 **\***src8 **=** **(**u8 **\*)** src**;**

u8 \_\_iomem **\***dst8 **=** **(**u8 \_\_iomem **\*)** dst**;**

**for** **(**i **=** c **&** 3**;** i**--;** **)**

fb\_writeb**(\***src8**++,** dst8**++);**

dst **=** **(**u32 \_\_iomem **\*)** dst8**;**

**}**

**\***ppos **+=** c**;**

buf **+=** c**;**

cnt **+=** c**;**

count **-=** c**;**

**}**

kfree**(**buffer**);**

**return** **(**cnt**)** **?** cnt **:** err**;**

**}**

### 4.5 fb\_read

fb\_read用于读取显存的数据。

static ssize\_t

fb\_read**(**struct file **\***file**,** char \_\_user **\***buf**,** size\_t count**,** loff\_t **\***ppos**)**

**{**

unsigned long p **=** **\***ppos**;**

struct inode **\***inode **=** file**->**f\_path**.**dentry**->**d\_inode**;**

int fbidx **=** iminor**(**inode**);**

//通过fbidx索引registered\_fb得到设备的fb\_info结构体实例

struct fb\_info **\***info **=** registered\_fb**[**fbidx**];**

u32 **\***buffer**,** **\***dst**;**

u32 \_\_iomem **\***src**;**

int c**,** i**,** cnt **=** 0**,** err **=** 0**;**

unsigned long total\_size**;**

//若info不存在 或 DMA缓冲区的虚拟地址不存在，则出错返回

**if** **(!**info **||** **!** info**->**screen\_base**)**

**return** **-**ENODEV**;**

**if** **(**info**->**state **!=** FBINFO\_STATE\_RUNNING**)**

**return** **-**EPERM**;**

//若设备fb\_ops操作集中有fb\_read函数，则调用之

**if** **(**info**->**fbops**->**fb\_read**)**

**return** info**->**fbops**->**fb\_read**(**info**,** buf**,** count**,** ppos**);**

total\_size **=** info**->**screen\_size**;**

//total\_size为0的话，取 info->fix.smem\_len

**if** **(**total\_size **==** 0**)**

total\_size **=** info**->**fix**.**smem\_len**;**

//若 读取偏移量 > total\_size，则出错返回。

**if** **(**p **>=** total\_size**)**

**return** 0**;**

//若 读取数目 > total\_size，则修正为total\_size

**if** **(**count **>=** total\_size**)**

count **=** total\_size**;**

//若 读取数目 + 偏移量 > total\_size，则 读取数目 修正为total\_size - 偏移量

**if** **(**count **+** p **>** total\_size**)**

count **=** total\_size **-** p**;**

//动态申请内存，用于缓冲数据，最大申请PAGE\_SIZE大小，后面会分段传输

buffer **=** kmalloc**((**count **>** PAGE\_SIZE**)** **?** PAGE\_SIZE **:** count**,**

GFP\_KERNEL**);**

**if** **(!**buffer**)**

**return** **-**ENOMEM**;**

//DMA缓冲区的首地址 + 偏移量 设为源地址

src **=** **(**u32 \_\_iomem **\*)** **(**info**->**screen\_base **+** p**);**

**if** **(**info**->**fbops**->**fb\_sync**)**

info**->**fbops**->**fb\_sync**(**info**);**

**while** **(**count**)** **{**

//缓冲大小为PAGE\_SIZE，若count较大，则分段。

c **=** **(**count **>** PAGE\_SIZE**)** **?** PAGE\_SIZE **:** count**;**

//目的地址为缓冲区首地址

dst **=** buffer**;**

//每次读取4字节

**for** **(**i **=** c **>>** 2**;** i**--;** **)**

**\***dst**++** **=** fb\_readl**(**src**++);**

//对于最后小于4字节的数据，进行特别操作，每次传输1字节

**if** **(**c **&** 3**)** **{**

u8 **\***dst8 **=** **(**u8 **\*)** dst**;**

u8 \_\_iomem **\***src8 **=** **(**u8 \_\_iomem **\*)** src**;**

**for** **(**i **=** c **&** 3**;** i**--;)**

**\***dst8**++** **=** fb\_readb**(**src8**++);**

src **=** **(**u32 \_\_iomem **\*)** src8**;**

**}**

//将内核缓冲区中的数据传输给用户层

**if** **(**copy\_to\_user**(**buf**,** buffer**,** c**))** **{**

err **=** **-**EFAULT**;**

**break;**

**}**

**\***ppos **+=** c**;**

buf **+=** c**;**

cnt **+=** c**;**

count **-=** c**;**

**}**

kfree**(**buffer**);**

**return** **(**err**)** **?** err **:** cnt**;**

**}**

### 4.6 fb\_ioctl

fb\_ioctl提供了framebuffer子系统的一些操作。

static int

fb\_ioctl**(**struct inode **\***inode**,** struct file **\***file**,** unsigned int cmd**,**

unsigned long arg**)**

**{**

int fbidx **=** iminor**(**inode**);**

struct fb\_info **\***info **=** registered\_fb**[**fbidx**];**

struct fb\_ops **\***fb **=** info**->**fbops**;**

struct fb\_var\_screeninfo var**;**

struct fb\_fix\_screeninfo fix**;**

struct fb\_con2fbmap con2fb**;**

struct fb\_cmap\_user cmap**;**

struct fb\_event event**;**

void \_\_user **\***argp **=** **(**void \_\_user **\*)**arg**;**

int i**;**

**if** **(!**fb**)**

**return** **-**ENODEV**;**

**switch** **(**cmd**)** **{**

**case** FBIOGET\_VSCREENINFO**:**

//获取fb\_var\_screeninfo(可变参数)，直接将info->var传输给用户层

**return** copy\_to\_user**(**argp**,** **&**info**->**var**,**

**sizeof(**var**))** **?** **-**EFAULT **:** 0**;**

**case** FBIOPUT\_VSCREENINFO**:**

//通过argp，将用户层传递过来的参数保存在var变量中

**if** **(**copy\_from\_user**(&**var**,** argp**,** **sizeof(**var**)))**

**return** **-**EFAULT**;**

acquire\_console\_sem**();**

//调用fb\_set\_var，进行设定，FBINFO\_MISC\_USEREVENT用于指定需要设定参数

info**->**flags **|=** FBINFO\_MISC\_USEREVENT**;**

i **=** fb\_set\_var**(**info**,** **&**var**);**

info**->**flags **&=** **~**FBINFO\_MISC\_USEREVENT**;**

release\_console\_sem**();**

//若fb\_set\_var出错，则将错误码返回

**if** **(**i**)** **return** i**;**

//若没出错，则通过argp地址将var参数返回

**if** **(**copy\_to\_user**(**argp**,** **&**var**,** **sizeof(**var**)))**

**return** **-**EFAULT**;**

**return** 0**;**

**case** FBIOGET\_FSCREENINFO**:**

//获取fb\_fix\_screeninfo(可变参数)，直接将info->fix传输给用户层

**return** copy\_to\_user**(**argp**,** **&**info**->**fix**,**

**sizeof(**fix**))** **?** **-**EFAULT **:** 0**;**

**case** FBIOPUTCMAP**:**

//通过argp，将用户设定的调色板参数保存在cmap中

**if** **(**copy\_from\_user**(&**cmap**,** argp**,** **sizeof(**cmap**)))**

**return** **-**EFAULT**;**

//调用fb\_set\_user\_cmap，设置调色表

**return** **(**fb\_set\_user\_cmap**(&**cmap**,** info**));**

**case** FBIOGETCMAP**:**

//查询用户提供的颜色表

**if** **(**copy\_from\_user**(&**cmap**,** argp**,** **sizeof(**cmap**)))**

**return** **-**EFAULT**;**

**return** fb\_cmap\_to\_user**(&**info**->**cmap**,** **&**cmap**);**

**case** FBIOPAN\_DISPLAY**:**

**if** **(**copy\_from\_user**(&**var**,** argp**,** **sizeof(**var**)))**

**return** **-**EFAULT**;**

acquire\_console\_sem**();**

i **=** fb\_pan\_display**(**info**,** **&**var**);**

release\_console\_sem**();**

**if** **(**i**)**

**return** i**;**

**if** **(**copy\_to\_user**(**argp**,** **&**var**,** **sizeof(**var**)))**

**return** **-**EFAULT**;**

**return** 0**;**

**case** FBIO\_CURSOR**:**

**return** **-**EINVAL**;**

**case** FBIOGET\_CON2FBMAP**:**

**if** **(**copy\_from\_user**(&**con2fb**,** argp**,** **sizeof(**con2fb**)))**

**return** **-**EFAULT**;**

**if** **(**con2fb**.**console **<** 1 **||** con2fb**.**console **>** MAX\_NR\_CONSOLES**)**

**return** **-**EINVAL**;**

con2fb**.**framebuffer **=** **-**1**;**

event**.**info **=** info**;**

event**.**data **=** **&**con2fb**;**

fb\_notifier\_call\_chain**(**FB\_EVENT\_GET\_CONSOLE\_MAP**,** **&**event**);**

**return** copy\_to\_user**(**argp**,** **&**con2fb**,**

**sizeof(**con2fb**))** **?** **-**EFAULT **:** 0**;**

**case** FBIOPUT\_CON2FBMAP**:**

**if** **(**copy\_from\_user**(&**con2fb**,** argp**,** **sizeof(**con2fb**)))**

**return** **-** EFAULT**;**

**if** **(**con2fb**.**console **<** 0 **||** con2fb**.**console **>** MAX\_NR\_CONSOLES**)**

**return** **-**EINVAL**;**

**if** **(**con2fb**.**framebuffer **<** 0 **||** con2fb**.**framebuffer **>=** FB\_MAX**)**

**return** **-**EINVAL**;**

#ifdef CONFIG\_KMOD

**if** **(!**registered\_fb**[**con2fb**.**framebuffer**])**

try\_to\_load**(**con2fb**.**framebuffer**);**

#endif /\* CONFIG\_KMOD \*/

**if** **(!**registered\_fb**[**con2fb**.**framebuffer**])**

**return** **-**EINVAL**;**

event**.**info **=** info**;**

event**.**data **=** **&**con2fb**;**

**return** fb\_notifier\_call\_chain**(**FB\_EVENT\_SET\_CONSOLE\_MAP**,**

**&**event**);**

**case** FBIOBLANK**:**

acquire\_console\_sem**();**

info**->**flags **|=** FBINFO\_MISC\_USEREVENT**;**

i **=** fb\_blank**(**info**,** arg**);**

info**->**flags **&=** **~**FBINFO\_MISC\_USEREVENT**;**

release\_console\_sem**();**

**return** i**;**

**default:**

**if** **(**fb**->**fb\_ioctl **==** **NULL)**

**return** **-**EINVAL**;**

**return** fb**->**fb\_ioctl**(**info**,** cmd**,** arg**);**

**}**

**}**

### 4.7 fb\_mmap

Linux内核中，关于虚存管理的最基本的管理单元应该是struct vm\_area\_struct了，它描述的是一段连续的、具有相同访问属性的虚存空间，该虚存空间的大小为物理内存页面的整数倍。

vm\_area\_struct结构所描述的虚存空间以vm\_start、vm\_end成员表示，它们分别保存了该虚存空间的首地址和末地址后第一个字节的地址，以字节为单位，所以虚存空间范围可以用[vm\_start, vm\_end)表示。

假如该vm\_area\_struct描述的是一个文件映射的虚存空间，成员vm\_file便指向被映射的文件的file结构，vm\_pgoff是该虚存空间起始地址在vm\_file文件里面的文件偏移，单位为物理页面。

通过fb\_mmap可以将显存的物理地址映射到虚拟地址供用户使用。

static int

fb\_mmap**(**struct file **\***file**,** struct vm\_area\_struct **\*** vma**)**

**{**

int fbidx **=** iminor**(**file**->**f\_path**.**dentry**->**d\_inode**);**

struct fb\_info **\***info **=** registered\_fb**[**fbidx**];**

struct fb\_ops **\***fb **=** info**->**fbops**;**

unsigned long off**;**

unsigned long start**;**

u32 len**;**

**if** **(**vma**->**vm\_pgoff **>** **(~**0UL **>>** PAGE\_SHIFT**))**

**return** **-**EINVAL**;**

//计算出偏移地址

off **=** vma**->**vm\_pgoff **<<** PAGE\_SHIFT**;**

**if** **(!**fb**)**

**return** **-**ENODEV**;**

//如果有自定义的mmap，则调用

**if** **(**fb**->**fb\_mmap**)** **{**

int res**;**

lock\_kernel**();**

res **=** fb**->**fb\_mmap**(**info**,** vma**);**

unlock\_kernel**();**

**return** res**;**

**}**

lock\_kernel**();**

//start为显存的物理地址

start **=** info**->**fix**.**smem\_start**;**

//显存物理地址按页对齐的总长度

len **=** PAGE\_ALIGN**((**start **&** **~**PAGE\_MASK**)** **+** info**->**fix**.**smem\_len**);**

//如果偏移的长度大于总长度

**if** **(**off **>=** len**)** **{**

//对应于I/O端口统一编址的情况

off **-=** len**;**

**if** **(**info**->**var**.**accel\_flags**)** **{**

unlock\_kernel**();**

**return** **-**EINVAL**;**

**}**

start **=** info**->**fix**.**mmio\_start**;**

len **=** PAGE\_ALIGN**((**start **&** **~**PAGE\_MASK**)** **+** info**->**fix**.**mmio\_len**);**

**}**

unlock\_kernel**();**

//显存物理地址按页对齐

start **&=** PAGE\_MASK**;**

//判断虚拟内存长度是否超越实际长度

**if** **((**vma**->**vm\_end **-** vma**->**vm\_start **+** off**)** **>** len**)**

**return** **-**EINVAL**;**

off **+=** start**;**

vma**->**vm\_pgoff **=** off **>>** PAGE\_SHIFT**;**

//标记这段虚拟内存映射为IO区域，并阻止系统将该区域包含在进程的存放转存中

//标记这段区域不能被换出

vma**->**vm\_flags **|=** VM\_IO **|** VM\_RESERVED**;**

vma**->**vm\_page\_prot **=** pgprot\_writecombine**(**vma**->**vm\_page\_prot**);**

//创建物理内存到虚拟内存的映射，建立页表

//vma:虚拟内存区域

//vma->vm\_start:虚拟地址的起始地址

//off >> PAGE\_SHIFT：实际的物理地址偏移的page

//虚拟空间范围

//vm\_page\_prot对这个虚拟区间的存取权限

**if** **(**io\_remap\_pfn\_range**(**vma**,** vma**->**vm\_start**,** off **>>** PAGE\_SHIFT**,**

vma**->**vm\_end **-** vma**->**vm\_start**,** vma**->**vm\_page\_prot**))**

**return** **-**EAGAIN**;**

**return** 0**;**

**}**

## 4.测试

读取/proc/fb文件，可以获取fb设备信息。

# cat /proc/fb

0 s3c2410fb

查看/sys/class/graphics/fb0/内容如下：

# ls /sys/class/graphics/fb0/

bits\_per\_pixel device power uevent

blank mode rotate virtual\_size

console modes state

cursor name stride

dev pan subsystem

对/dev/fb0的简单操作：

# echo hello > /dev/tty1 显示hello

# echo 任意文件 > /dev/fb0 花屏

# dd if=/dev/zero of=/dev/fb0 bs=480 count=272 清空屏幕

一般如果要写应用程序对/dev/fb0进行操作，步骤如下：

(1)打开/dev/fb设备文件。

(2)用ioctl操作取得当前显示屏幕的参数，根据屏幕参数可计算屏幕缓冲区的大小。

(3)将屏幕缓冲区映射到用户空间。

(4)映射后即可直接读写屏幕缓冲区，进行绘图和图片显示。

示例代码如下：

int main **(** int argc**,** char **\***argv**[]** **)**

**{**

    int fbfd **=** 0**;**

    struct fb\_var\_screeninfo vinfo**;**

    struct fb\_fix\_screeninfo finfo**;**

    long int screensize **=** 0**;**

    struct fb\_bitfield red**;**

    struct fb\_bitfield green**;**

    struct fb\_bitfield blue**;**

    //1.打开显示设备

    fbfd **=** open**(**"/dev/fb0"**,** O\_RDWR**);**

    if **(!**fbfd**)**

**{**

        printf**(**"Error: cannot open framebuffer device.\n"**);**

        exit**(**1**);**

**}**

    if **(**ioctl**(**fbfd**,** FBIOGET\_FSCREENINFO**,** **&**finfo**))**

**{**

        printf**(**"Error：reading fixed information.\n"**);**

        exit**(**2**);**

**}**

    if **(**ioctl**(**fbfd**,** FBIOGET\_VSCREENINFO**,** **&**vinfo**))**

**{**

        printf**(**"Error: reading variable information.\n"**);**

        exit**(**3**);**

**}**

    printf**(**"R:%d,G:%d,B:%d \n"**,** vinfo**.**red**,** vinfo**.**green**,** vinfo**.**blue **);**

    printf**(**"%dx%d, %dbpp\n"**,** vinfo**.**xres**,** vinfo**.**yres**,** vinfo**.**bits\_per\_pixel **);**

    xres **=** vinfo**.**xres**;**

    yres **=** vinfo**.**yres**;**

    bits\_per\_pixel **=** vinfo**.**bits\_per\_pixel**;**

    //2.获取屏幕参数，计算屏幕的总大小（字节）

    screensize **=** vinfo**.**xres **\*** vinfo**.**yres **\*** vinfo**.**bits\_per\_pixel **/** 8**;**

    printf**(**"screensize=%d byte\n"**,**screensize**);**

    //3.对象映射

    fbp **=** **(**char **\*)**mmap**(**0**,** screensize**,** PROT\_READ **|** PROT\_WRITE**,** MAP\_SHARED**,** fbfd**,** 0**);**

    if **((**int**)**fbp **==** **-**1**)**

**{**

        printf**(**"Error: failed to map framebuffer device to memory.\n"**);**

        exit**(**4**);**

**}**

    printf**(**"sizeof file header=%d\n"**,** sizeof**(**BITMAPFILEHEADER**));**

    printf**(**"into show\_bmp function\n"**);**

    //4.操作，显示图像

    show\_bmp**();**

    //删除对象映射

    munmap**(**fbp**,** screensize**);**

    close**(**fbfd**);**

    return 0**;**

**}**  ·

## 5.sysfs快速调试功能

在sysfs下创建调试文件：

static DEVICE\_ATTR**(**rawdata**,** S\_IRUGO**|**S\_IWUSR**,** gtp\_sysfs\_rawdata\_sh

ow**,** gtp\_sysfs\_rawdata\_store**);**

s32 gtp\_sysfs\_init**(**void**)**

**{**

    s32 ret **;**

    debug\_kobj **=** kobject\_create\_and\_add**(**"gtp"**,** **NULL)** **;**

    //SET\_INFO\_LINE\_INFO("Starting initlizing gtp\_debug\_sysfs");

    if **(**debug\_kobj **==** **NULL)**

**{**

        GTP\_ERROR**(**"%s: subsystem\_register failed\n"**,** \_\_func\_\_**);**

        return **-**ENOMEM**;**

**}**

    ret **=** sysfs\_create\_file**(**debug\_kobj**,** **&**dev\_attr\_rawdata**.**attr**);**

    if **(**ret**)**

**{**

        GTP\_ERROR**(**"%s: sysfs\_create\_rawdata\_file failed\n"**,** \_\_func\_\_**);**

        return ret**;**

**}**

    return 0 **;**

**}**

其中读写函数为：

static ssize\_t gtp\_sysfs\_rawdata\_show**(**struct device **\***dev**,**struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

//当read /sys/gtp/rawdata，会打印on/off

**return** snprintf**(**buf**,** PAGE\_SIZE**,** "%s\n"**,** debug **?** "on" **:** "off"**);**

**}**

static ssize\_t gtp\_sysfs\_rawdata\_store**(**struct device **\***dev**,**struct device\_attribute **\***attr**,** char **\***buf**,** size\_t len**)**

**{**

//当write /sys/gtp/rawdata，数据会存放在buf中，长度为len

**if(**strnicmp**(**buf**,** "on"**,** 2**)** **==** 0 **||**

strnicmp**(**buf**,** "1"**,** 1**)** **==** 0**)**

**{**

debug **=** 1**;**

printk**(**KERN\_DEBUG "s3c2410fb: Debug On"**);**

**}** **else** **if** **(**strnicmp**(**buf**,** "off"**,** 3**)** **==** 0 **||**

strnicmp**(**buf**,** "0"**,** 1**)** **==** 0**)**

**{**

debug **=** 0**;**

printk**(**KERN\_DEBUG "s3c2410fb: Debug Off"**);**

**}** **else**

**{**

**return** **-**EINVAL**;**

**}**

**return** len**;**

**}**

方法2：

#在驱动程序中使用 device\_create\_file创建属性文件

static DEVICE\_ATTR**(**val**,** S\_IRUGO **|** S\_IWUSR**,** hello\_val\_show**,** hello\_val\_store**);**

/\*读取寄存器val的值到缓冲区buf中，内部使用\*/

static ssize\_t \_\_hello\_get\_val**(**struct xxx\_dev**\*** dev**,** char**\*** buf**)** **{**

int val **=** 0**;**

/\*同步访问\*/

**if(**down\_interruptible**(&(**dev**->**sem**)))** **{**

**return** **-**ERESTARTSYS**;**

**}**

val **=** dev**->**val**;**

up**(&(**dev**->**sem**));**

**return** snprintf**(**buf**,** PAGE\_SIZE**,** "%d/n"**,** val**);**

**}**

/\*把缓冲区buf的值写到设备寄存器val中去，内部使用\*/

static ssize\_t \_\_hello\_set\_val**(**struct xxx\_dev**\*** dev**,** const char**\*** buf**,** size\_t count**)** **{**

int val **=** 0**;**

/\*将字符串转换成数字\*/

val **=** simple\_strtol**(**buf**,** **NULL,** 10**);**

/\*同步访问\*/

**if(**down\_interruptible**(&(**dev**->**sem**)))** **{**

**return** **-**ERESTARTSYS**;**

**}**

dev**->**val **=** val**;**

up**(&(**dev**->**sem**));**

**return** count**;**

**}**

/\*读取设备属性val\*/

static ssize\_t hello\_val\_show**(**struct device**\*** dev**,** struct device\_attribute**\*** attr**,** char**\*** buf**)** **{**

struct xxx\_dev**\*** hdev **=** **(**struct xxx\_dev**\*)**dev\_get\_drvdata**(**dev**);**

**return** \_\_hello\_get\_val**(**hdev**,** buf**);**

**}**

/\*写设备属性val\*/

static ssize\_t hello\_val\_store**(**struct device**\*** dev**,** struct device\_attribute**\*** attr**,** const char**\*** buf**,** size\_t count**)** **{**

struct xxx\_dev**\*** hdev **=** **(**struct xxx\_dev**\*)**dev\_get\_drvdata**(**dev**);**

**return** \_\_hello\_set\_val**(**hdev**,** buf**,** count**);**

**}**

/\*模块加载方法\*/

static int \_\_init xxx\_init**(**void**){**

**...**

/\*在/sys/class/目录下创建设备类别目录xxx\*/

g\_vircdev\_class **=** class\_create**(**THIS\_MODULE**,** VIRCDEV\_CLASS\_NAME**);**

**if(**IS\_ERR**(**g\_vircdev\_class**))** **{**

err **=** PTR\_ERR**(**g\_vircdev\_class**);**

printk**(**KERN\_ALERT "Failed to create class.\n"**);**

**goto** CLASS\_CREATE\_ERR**;**

**}**

/\*在/dev/目录和/sys/class/xxx目录下分别创建设备文件xxx\*/

dev **=** device\_create**(**g\_vircdev\_class**,** **NULL,** devt**,** **NULL,** VIRCDEV\_DEVICE\_NAME**);**

**if(**IS\_ERR**(**dev**))** **{**

err **=** PTR\_ERR**(**dev**);**

printk**(**KERN\_ALERT "Failed to create device.\n"**);**

**goto** DEVICE\_CREATE\_ERR**;**

**}**

/\*在/sys/class/xxx/xxx目录下创建属性文件val\*/

err **=** device\_create\_file**(**dev**,** attr**);**

**if(**err **<** 0**)** **{**

printk**(**KERN\_ALERT"Failed to create attribute file."**);**

**goto** DEVICE\_CREATE\_FILE\_ERR**;**

**}**

**...**

**}**