FBDEV

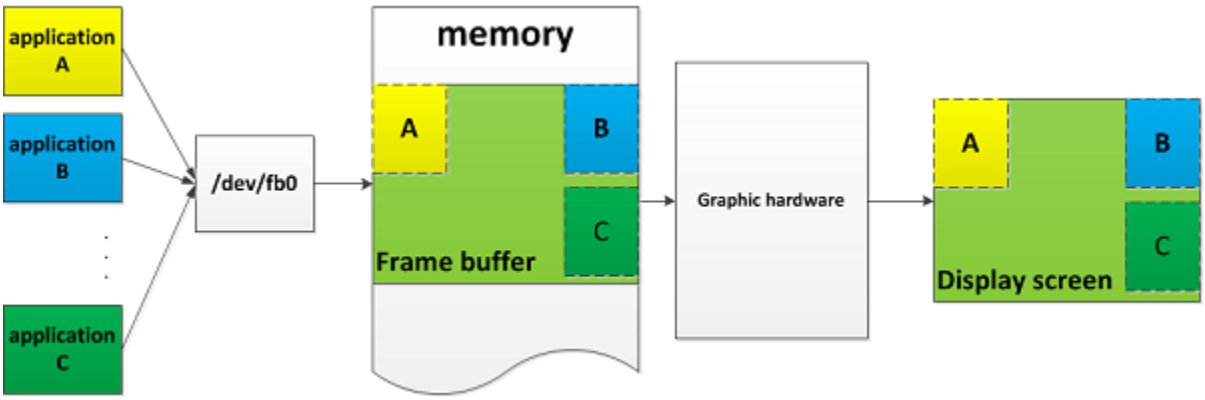
Overview

1. Introduction

> *The frame buffer device provides an abstraction for the graphics hardware. It represents the frame buffer of some video hardware and allows application software to access the graphics hardware through a well-defined interface, so the software doesn't need to know anything about the low-level (hardware register) stuff. fbdev用于为显示图形硬件提供一层软件抽象.它代理了显示图形硬件的帧内存,并且提供了一些良好定义的接口让应用软件去访问图形硬件，而不用去关心底层图形硬件的具体控制细节。*

> *The device is accessed through special device nodes, usually located in the /dev directory, i.e. /dev/fb*.*
>
> *访问fbdev通常通过一些特定的设备节点，例如位于/dev 目录下的 /dev/fb*.*

> *一个简单的framebuffer 使用场景：*
>
> *3个application 通过fb0这个fb device 节点对framebuffer进行修改，然后有图形硬件将修改之后的内容输出到显示端*



2. User's View of /dev/fb*

> *From the user's point of view, the frame buffer device looks just like any other device in /dev. It's a character device using major 29; the minor specifies the frame buffer number.*
>
> *By convention, the following device nodes are used (numbers indicate the device minor numbers): fbdev 目前是一个字符设备，首设备号29,尾备号标志frame buffer的编号*
>
> *方便起见，设备节点会如下生成(数字表示问设备号):*

```
0 = /dev/fb0      First frame buffer
1 = /dev/fb1      Second frame buffer
   ...
31 = /dev/fb31     32nd frame buffer
```

> *fbdev不单是一个字符设备,同时也是一个普通的内存设备.这就意味着,你可以读写它的内容.比如:*
>
> *1.你可以通过如下操作截屏:*

```
cp /dev/fb0 myfile
```

> *2.往屏幕的左上角画一个白色的像素点:*

```
echo -en '\xFF\xFF\xFF\x00' > /dev/fb0
```

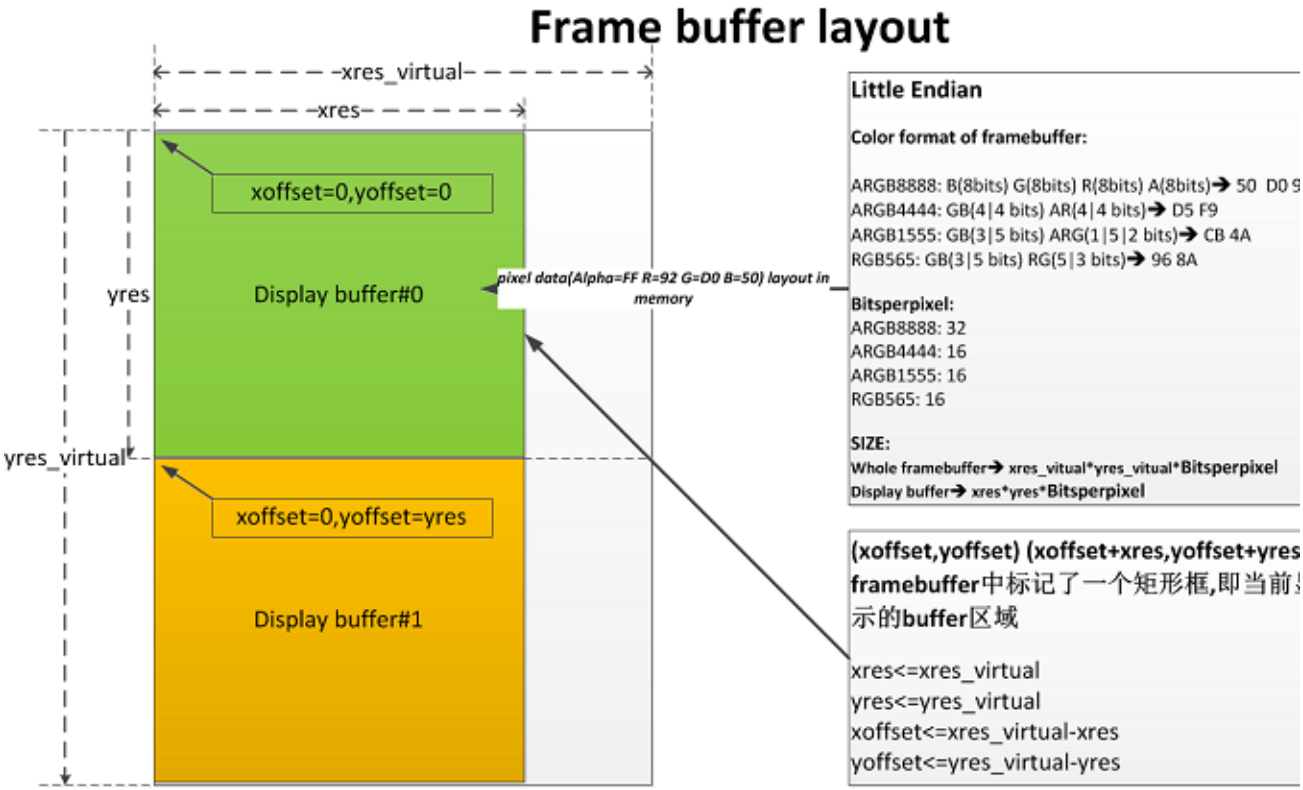
> *3.mmap /dev/fb0 用于更精细的绘图(这部分会在其它章节详述):*

```
int fb = open("/dev/fb0", O_RDWR);
assert(fb > 0);
struct fb_var_screeninfo info;
assert(0 == ioctl(fb, FBIOGET_VSCREENINFO, &info));
size_t len = 4 * info.xres * info.yres;
uint32_t *buf = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fb, 0);
assert(buf != MAP_FAILED);
```

通过给buf[y * info.xres + x] 赋值就可以轻松的修改 (x,y)这个点的像素.

3. Closer look to /dev/fb*

简介提到fbdev代理了图形硬件的显示帧内存，并且提供了一些接口控制这部分内存,对比上文的framebuffer简单应用场景。 下图是framebuffer的memory layout:



- 不同的fbdev的实现,让fbdev代理的图形硬件的显示帧内存（framebuffer）会具备不同的初始属性.有些是可配置的属性（通过`ioctl(fb, FBIOGET_VSCREENINFO, &info)`获取`struct fb_var_screeninfo defined in "linux/fb.h"`），有些则是固定属性（通过`ioctl(fb, FBIOGET_FSCREENINFO, &info)`获取`struct fb_fix_screeninfo defined in "linux/fb.h"`）.
- sstar 平台的初始设定方式会在后面讲述(fbdev.ini)

以上图为例我们需要关注几个主要属性：

- ***颜色格式(colorformat)***:framebuffer的颜色格式可以通过`ioctl(fb, FBIOPUT_VSCREENINFO, &info)` 设定,通常我们都是在初始化fbdev的时候就决定好. 对fbdev的用户来说，确定了颜色格式用户就知道如何修改对应pixel的内容.
- ***bitsperpixel***: 一个pixel的宽度,与颜色格式协调的变量,颜色格式确定了pixel的宽度也确定了.
- ***xres_virtual/yres_virtual***: 可通过ioctl FBIOGET_VSCREENINFO/FBIOPUT_VSCREENINFO获取.通常fbdev 代理的framebuffer总大小为:***xres_virtual*yres_virtual*bitsperpixel***
- ***xres/yres,xoffset/yoffset***: (xoffset,yoffset) (xoffset+xres,yoffset+yres) 在framebuffer中标记了一个矩形框,即当前显示的buffer区域.也就是说display buffer 用于显示的有效区域可以是从framebuffer开始地址开始。

为何 display buffer 在 framebuffer内

- 1. ***支持不同分辨率的内容***.

有时候framebuffer的内存大小在FBDEV初始化的时候已经申请好了. 我们可能需要申请一个能够容纳最大分辨的framebuffer，比如1080p,此时我们使用的display buffer 可能是720p.

-

2. ***double buffer***.使用单buffer无法避免画面撕裂的问题，因为你总是可能修改未显示的内容，造成已经显被graphic hardware显示的部分，和被修改将要显示的部分时空错位。使用双buffer,显示一张display buffer#0,修改另一张显示一张display buffer#1，修改完成之后交换.显示一张display buffer#1,修改另一张显示一张display buffer#0.

4.FBDEV in sstar platform

初始化

- 实现FBDEV的软件模块fbdev.ko.
- 以该方式加载FBDEV:insmod fbdev.ko [fbdev configure file(absolute path)],配置文件可选,系统默认的加载的配置文件路径 ***/config/fbdev.ini***

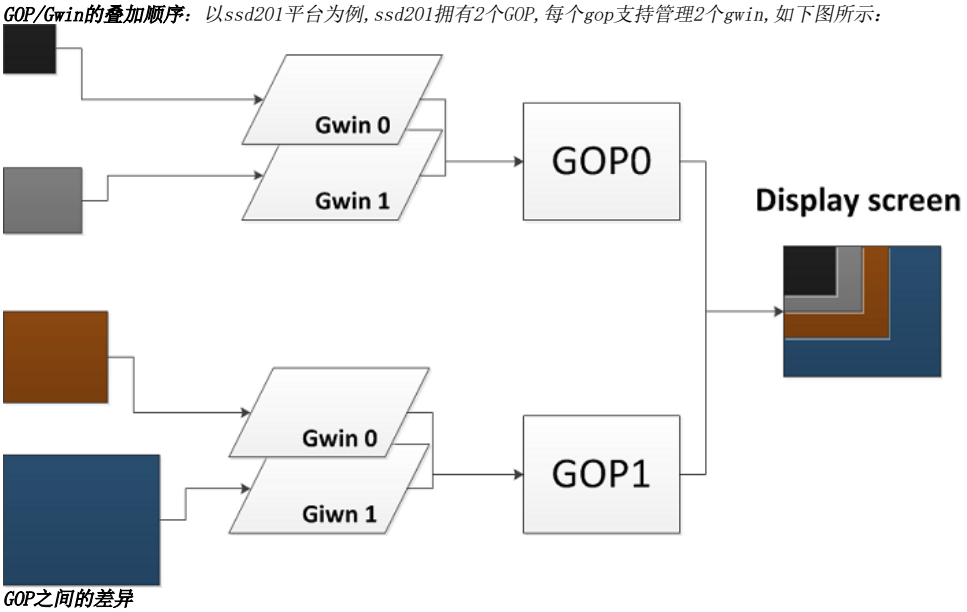
配置文件fbdev.ini

```
# FBDEV [FB_DEVICE]fbdev
# [FB_DEVICE],fbdev
[FB_DEVICE]
# fbdevgop(graphic hardware) ID
FB_HWLAYER_ID = 1
# fbdevframebuffergop graphic window ID
FB_HWWIN_ID = 0
# deprecated
FB_HWLAYER_DST = 3
# fbdevframebuffer
# RGB565 = 1
# ARGB4444 = 2
# ARGB8888 = 5
# ARGB1555 = 6
# YUV422 = 9
# I8 = 4
# I4 = 13
# I2 = 14
FB_HWWIN_FORMAT = 5
#deprecated
FB_HWLAYER_OUTPUTCOLOR = 1
# fbdevframebufferxres=xres_virtual=FB_WIDTH
FB_WIDTH = 1280
# fbdevframebufferyres=yres_virtual=FB_HEIGHT
FB_HEIGHT = 720
#timinggoptiming
FB_TIMMING_WIDTH = 1920
#timinggoptiming
FB_TIMMING_HEIGHT = 1080
# mmaplayoutE_MMAP_ID_FB
# FBDEVframebuffer
FB_MMAP_NAME = E_MMAP_ID_FB
# mmapFBDEV layout
# FBDEVframebufferframebuffer
FB_BUFFER_LEN = 8192
#unit:Kbyte,4096=4M, fbdev.ko alloc size = FB_BUFFER_LEN*1024

# FBDEV
[FB_CURSOR]
# gop ID
FB_HWLAYER_ID = 0
# gop graphic window ID
FB_HWWIN_ID = 0
# deprecated
FB_HWLAYER_DST = 3
#
# RGB565 = 1
# ARGB4444 = 2
# ARGB8888 = 5
# ARGB1555 = 6
# YUV422 = 9
# I8 = 4
# I4 = 13
# I2 = 14
FB_HWWIN_FORMAT = 6
# deprecated
FB_HWLAYER_OUTPUTCOLOR = 1
# mmaplayoutE_MMAP_ID_FB
# FBDEV
# mmapFBDEV layout
# FBDEV128K
FB_MMAP_NAME = E_MMAP_ID_HW_CURSOR

# deprecated,fbdev z order()
[LAYER_ZORDER]
LAYER_ZORDER0 = 0
LAYER_ZORDER1 = 1
LAYER_ZORDER2 = 2
LAYER_ZORDER3 = 3
LAYER_ZORDER4 = 4
```

graphic hardware in sstar(GOP)

观察fbdev.ini 的配置文件,看到每一个fbdev都会绑定 一个GOP ID,以及一个GWIN ID.在sstar的某个硬件平台中,可以有多个GOP,同时单个GOP可能可以控制多个graphic window（Gwin）,每一个Gwin对应一个framebuffer中的display buffer.具体的GOP/Gwin的数量不同的平台存在差异.

- ***GOP/Gwin的叠加顺序：*** *以ssd201平台为例,ssd201拥有2个GOP,每个gop支持管理2个gwin,如下图所示：*



- ***GOP之间的差异***

> *GOP之间可能存在设计差异，不用的GOP的用途存在倾向性。以ssd201 平台为例*
>
> - *支持的颜色格式*
> - *是否支持stretch*

5. Interfaces

简介提到fbdev定义了一些Interfaces给用户操作framebuffer,这里需要注意的是这些接口的实现程度取决于FBDEV的实现者对linux定义的这部分接口的具体实现程度.通常来讲,针对某些具体的平台不需要实现全部的接口定义.

比如:color map/write&read函数等。因为index color显示效果不如ARGB32，操作framebuff 使用mmap比wirte&read函数更方便。

除此之外,FBDEV的提供着还可以针对特定平台提供专有的控制接口

接口支持情况表:

User Space 操作接口	*描述*	*sstar支持平台*
open/fopen/close/fclose	*以文件方式打开/关闭fbdev*	*all*
write/fwrite/read/fread	*以读写文件的方式读写fbdev*	*none*
mmap	*将fbdev做内存映射*	*all*
iocontrol	*fbdev支持iocontrol 的方式放送控制命令*	*all*

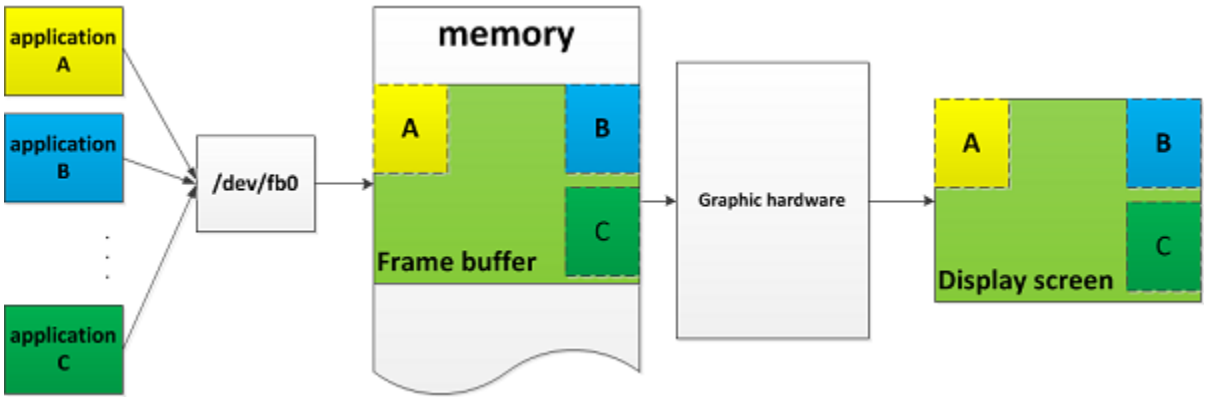
linux标准iocontrol命令表:

iocontrol命令	*描述*	*sstar支持平台*
FBIOGET_VSCREENINFO	*获取fbdev的可变信息*	*all*
FBIOPUT_VSCREENINFO	*设置fbdev的可变信息*	*all*
FBIOGET_FSCREENINFO	*获取fbdev的固定信息*	*all*
FBIOPAN_DISPLAY	*平移显示,在修改vinfo的可见区域之后*	*all*
FBIOPUTCMAP	*设置fbdev的colormap*	*none*
FBIOGETCMAP	*获取fbdev的colormap*	*none*
FBIOGET_CON2FBMAP	*获取console对应的framebuffer*	*all*
FBIOPUT_CON2FBMAP	*映射console对应的framebuffer*	*all*

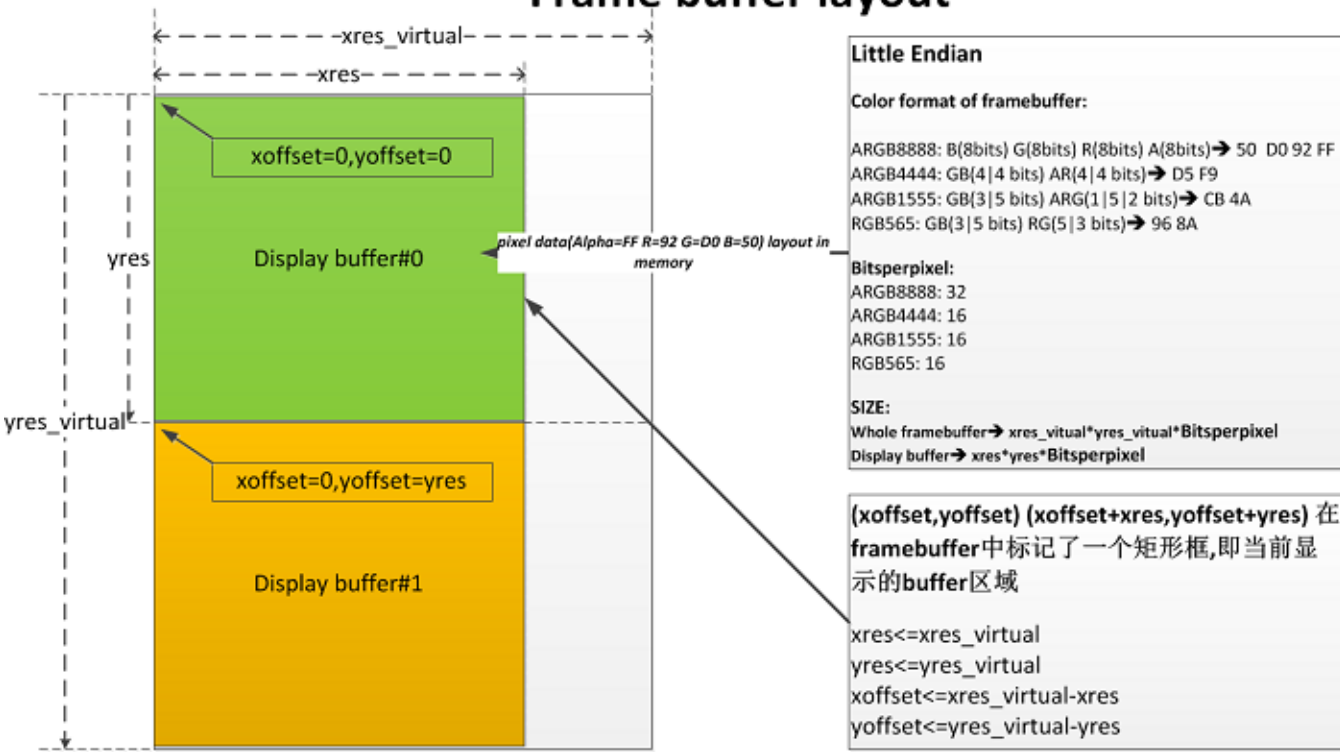
FBIOBLANK	清空framebuffer	none

sstar专有 iocontrol 命令表:

iocontrol命令	描述	sstar支持平台
FBIOGET_SHOW	获取fbdev的显示状态	all
FBIOSET_SHOW	设置fbdev的显示状态	all
FBIOGET_SCREEN_LOCATION	获取显示区域在framebuffer中的位置信息	all
FBIOSET_SCREEN_LOCATION	设置显示区域在framebuffer中的位置信息	all
FBIOGET_GLOBAL_ALPHA	获取fbdev的全局alpha和ARGB1555的index alpha信息	all
FBIOSET_GLOBAL_ALPHA	设置fbdev的全局alpha和ARGB1555的index alpha信息	all
FBIOGET_COLORKEY	获取fbdev的colorkey信息	all
FBIOSET_COLORKEY	设置fbdev的colorkey信息	all
FBIOGET_DISPLAYLAYER_ATTRIBUTRES	获取fbdev显示相关信息	all
FBIOSET_DISPLAYLAYER_ATTRIBUTRES	设置fbdev显示相关信息	all
FBIOGET_CURSOR_ATTRIBUTRE	设置fbdev的鼠标信息	all
FBIOSET_CURSOR_ATTRIBUTRE	获取fbdev的鼠标信息	all



Frame buffer layout



Little Endian

Color format of framebuffer:

ARGB8888: B(8bits) G(8bits) R(8bits) A(8bits)➔ 50 D0 92 FF
ARGB4444: GB{4|4 bits) AR{4|4 bits)➔ D5 F9
ARGB1555: GB{3|5 bits) ARG(1|5|2 bits)➔ CB 4A
RGB565: GB{3|5 bits) RG{5|3 bits)➔ 96 8A

Bitsperpixel:
ARGB8888: 32
ARGB4444: 16
ARGB1555: 16
RGB565: 16

SIZE:
Whole framebuffer➔ xres_vitual*yres_vitual*Bitsperpixel
Display buffer➔ xres*yres*Bitsperpixel

(xoffset,yoffset) (xoffset+xres,yoffset+yres) 在framebuffer中标记了一个矩形框,即当前显示的buffer区域

xres<=xres_virtual
yres<=yres_virtual
xoffset<=xres_virtual-xres
yoffset<=yres_virtual-yres