GFBG 开发指南

文档版本　V1.4

发布日期　2025-12-18

前 言

概述

Graphic Framebuffer Group（以下简称 GFBG）是 XM7606V13/XM7206V11A 数字媒体处理芯片提供的管理图形层的模块，它基于 Linux Framebuffer 实现，在提供 Linux Framebuffer 基本功能的基础上，扩展了一些图形层控制功能，如层间 Alpha、设置原点等。本文档主要介绍 GFBG 模块加载和如何快速开发应用。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠ 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
⚠ 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
⚠ 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2024-12-21	V1.0	DI 版本发布。
2025-03-04	V1.1	删除 proc 文字信息，更新 proc 截图。
		修改 insmod 参数，video->fb_vram。
2025-04-15	V1.2	添加 buildin 模式下 bootargs 传参说明。
2025-05-30	V1.3	添加 XM7206V11A 相关说明。
2025-12-18	V1.4	3.2 标准功能 GFBG_ACTIVATE_NOW 修正为 FB_ACTIVATE_NOW。

目 录

前 言 i

1 概述 1

1.1 GFBG 简介 1

1.1.1 体系结构 1

1.1.2 应用场景 2

1.2 GFBG 与 Linux Framebuffer 对比 2

2 模块加载 6

2.1 原理介绍 6

2.2 参数设置 6

2.3 配置举例 7

3 API 参考 8

3.1 API 类别 8

3.2 标准功能 8

3.3 扩展功能 16

3.4 数据类型 30

4 GFBG 应用开发流程 45

4.1 开发流程 45

4.2 实例介绍 47

5 图形界面开发 51

5.1 概述 51

5.1.1 模块概述 51
5.1.2 场景概述 51
5.2 实现用户界面方案 52
5.2.1 基本方案 52
5.2.2 同源方案 53
5.2.3 单缓存方案 54
5.2.4 局部刷新方案 55
5.2.5 开发流程 55
6 PROC 调试 57
6.1 图形层和 GFBG 设备号对应关系 57
6.2 单个图形层调试信息 57
6.3 图形层调试命令 61

插图目录

图 1-1 GFBG 体系结构 1
图 1-2 双 buffer 刷新时序图 4
图 1-3 单 buffer 刷新时序图 5
图 3-1 设置从虚拟分辨率中的不同偏移处开始显示 14
图 4-1 GFBG 的开发流程 45
图 5-1 基本方案结构图 53
图 5-2 同源方案结构图 54
图 5-3 单缓存方案结构图 54
图 5-4 局部刷新结构图 55

表格目录

表 1-1 GFBG 设备文件、图形层以及输出设备的对应关系 2

表 4-1 GFBG 的开发阶段任务表 46

1 概述

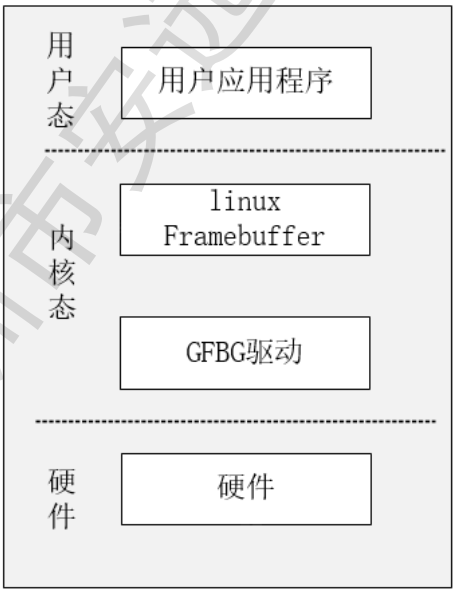
1.1 GFBG 简介

GFBG 是 XM7606V13/XM7206V11A 数字媒体处理平台提供的用于管理叠加图形层的模块，它不仅提供 Linux Framebuffer 的基本功能，还在 Linux Framebuffer 的基础上增加层间 colorkey、层间 Alpha、显示原点偏移等扩展功能。

1.1.1 体系结构

应用程序需基于 Linux 文件系统使用 GFBG，GFBG 的体系结构如图 1-1 所示。

图1-1 GFBG 体系结构



1.1.2 应用场景

GFBG 可应用于以下场景：

基于 Linux Framebuffer 的应用程序，可以不做或做少量改动即可移植到 XM7606V13/XM7206V11A，实现快速适配。

1.2 GFBG 与 Linux Framebuffer 对比

叠加图形层管理

Linux Framebuffer 是一个子设备号对应一个显卡，GFBG 则是一个子设备号对应一个叠加图形层，GFBG 可以管理多个叠加图形层，具体个数和芯片相关。

说明

- XM7606V13 芯片支持 2 个输出设备(高清输出设备，简称 DHD)上均可以叠加图形层。XM7206V11A 芯片支持 1 个输出设备(高清输出设备，简称 DHD)上可以叠加图形层。图形层与输出设备的关系如表 1-1 所示。
- 每个输出设备支持的接口类型和时序请参见《MPP 媒体处理软件开发参考》的视频输出章节。
- 为了显示图形层，用户必须先配置并启动输出设备（通过 VO 模块的接口），然后通过 GFBG 模块接口操作图形层使之显示。

表1-1 GFBG 设备文件、图形层以及输出设备的对应关系

芯片	设备文件	图形层	对应显示设备
XM7606V13	/dev/fb0	G0	只能在 DHD0 设备上显示。
	/dev/fb1	G1	只能在 DHD1 设备上显示。
XM7206V11A	/dev/fb0	G0	只能在 DHD0 设备上显示。

通过模块加载参数，可以控制 GFBG 管理其中的一个或多个叠加图形层，并像操作普通文件一样操作叠加图形层。

芯片差异

芯片	支持的图形层	是否支持压缩	是否支持colorkey	绑定关系	软鼠标
XM7606V13	G0	不支持压缩	支持	固定绑定到DHD0 上	不支持
	G1	不支持压缩	支持	固定绑定到DHD1 上	不支持
XM7206V11A	G0	不支持压缩	支持	固定绑定到DHD0 上	不支持

标准功能与扩展功能

GFBG 支持以下的 Linux Framebuffer 标准功能：

- 将物理显存映射（或解除映射）到虚拟内存空间。
- 像操作普通文件一样操作物理显存。
- 设置硬件显示分辨率和像素格式，每个叠加图形层的支持的最大分辨率和像素格式可以通过支持能力接口获取。
- 从物理显存的任何位置进行读、写、显示等操作。
- 在叠加图形层支持索引格式的情况下，支持设置和获取 256 色的调色板。

GFBG 增加以下的扩展功能：

- 设置和获取叠加图形层的 alpha 值。
- 设置和获取叠加图形层的 colorkey 值。
- 设置当前叠加图形层的起始位置（相对于屏幕原点的偏移）。
- 设置和获取当前叠加图形层的显示状态（显示/隐藏）。
- 通过模块加载参数配置 GFBG 的物理显存大小和管理叠加图形层的数目。
- 设置和获取预乘模式。

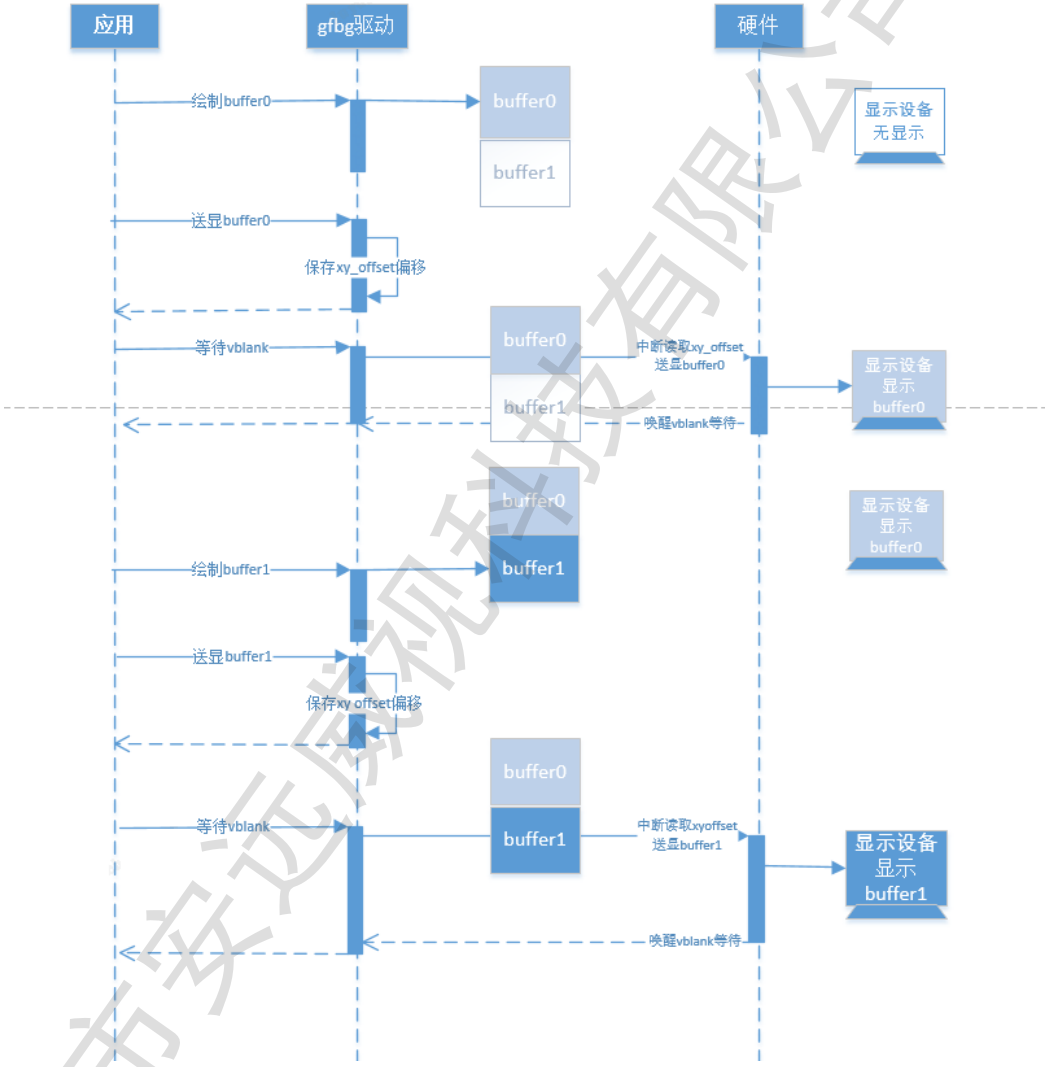
双 buffer 刷新模式流程

1、 用户获取 buffer0---绘制 buffer0----送显 buffer0----等待 vblank

2、 用户获取 buffer1---绘制 buffer1 ----送显 buffer1----等待 vblank

具体调用及中断送显时序如图 1-2 所示。

图1-2 双 buffer 刷新时序图

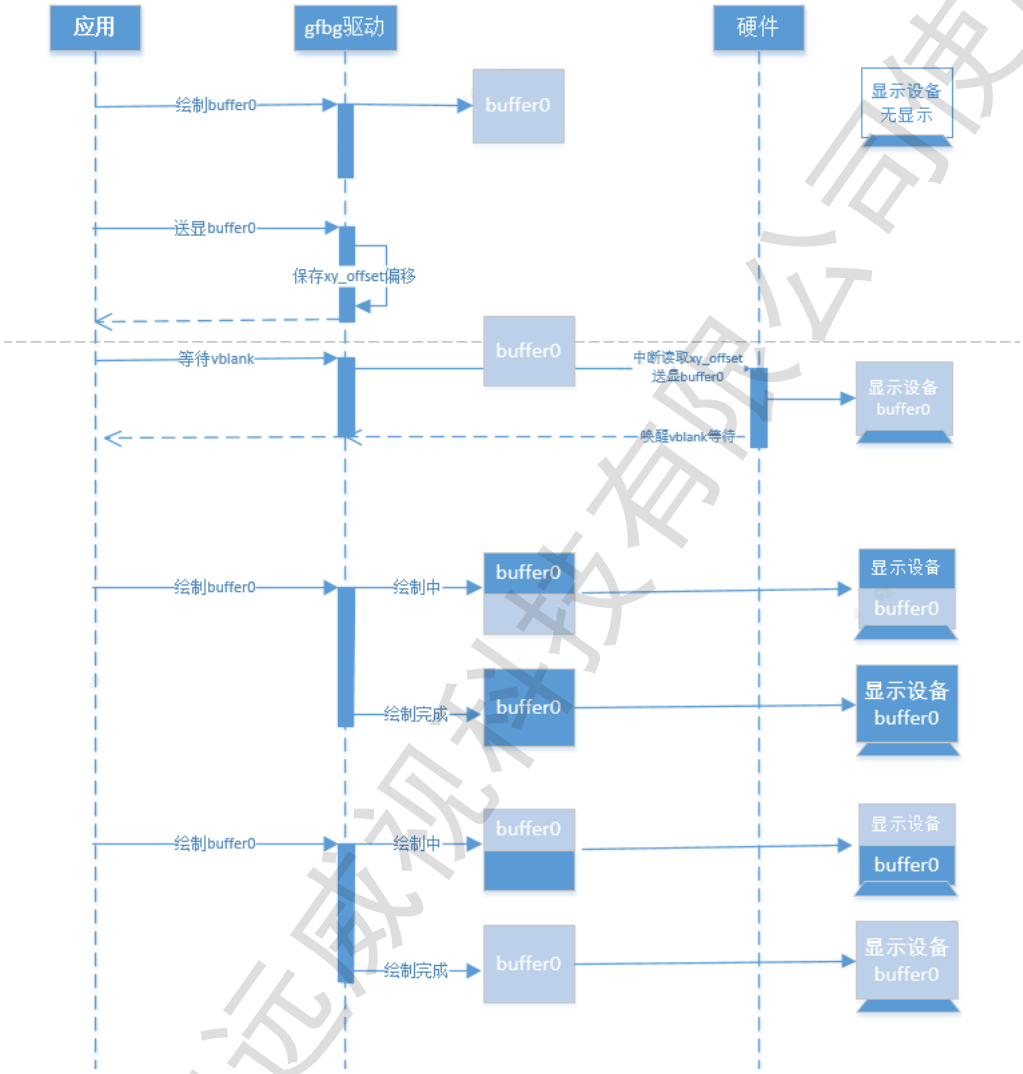


单 buffer 刷新模式流程

1、用户获取 buffer0---绘制 buffer0----送显 buffer0

2、 重新绘制 buffer0

图1-3 单 buffer 刷新时序图



2 模块加载

2.1 原理介绍

某些 Linux Framebuffer 驱动（如 versa）不支持在运行期间更改分辨率、颜色深度、时序等显示属性。对此，Linux 系统提供一种机制，允许在内核启动或模块加载时，通过参数将相应选项传递给 Linux Framebuffer。可以在内核加载器中配置内核启动参数。GFBG 驱动在加载时只能设置物理显存的大小，不允许设置其它选项。

加载 GFBG 驱动 xm_fb.ko 时必须保证内核中已经加载了标准的 Linux Framebuffer 框架，然后再加载 xm_fb.ko。

2.2 参数设置

GFBG 可配置其管理的叠加图形层物理显存的大小。物理显存大小决定了 GFBG 可使用的最大物理显存和系统的可设置虚拟分辨率。在加载 GFBG 驱动时通过参数传递设置物理显存大小，物理显存的大小一经设置就不会改变。

参数 fb_vram

```
XM7606V13：
insmod xm_fb.ko fb_vram="gfbg:vram0_size:xxx,vram1_size:xxx"
XM7206V11A：
insmod xm_fb.ko fb_vram="gfbg:vram0_size:xxx"
```

说明

- 选项之间用逗号“,”隔开。
- 选项和选项值之间用冒号“:”隔开。

- 如果 G0 不配置物理显存大小，XM7606V13 默认分配为 8100K，XM7206V11A 默认分配 3600K。如果 G1 不配置物理显存大小，则 G1 层不会加载，G1 层功能无法使用。
- vram0_size ~ vram1_size 分别对应于叠加图形层 0 ~ 叠加图形层 1。
- buildin 模式下，通过 bootargs 进行传参配置，相关文件路径：sdk/configs/芯片类型/prebuilts/xxx_bootargs_buildin.txt，在 bootargs 参数一行加入以下内容，注意不能换行：

fb_vram="gfbg:vram0_size:64800,vram1_size:16200"

其中，vramn_size:xxx 表示对叠加图形层 n 配置 xxx K 字节的物理显存。

对于 GFBG 标准模式，vramn_size 和虚拟分辨率的关系如下：

```
vramn_size * 1024 >= xres_virtual * yres_virtual * bpp;
```

其中：xres_virtual * yres_virtual 是虚拟分辨率，bpp 是每个像素所占字节数。

如：图形层 0 在 1280*720 分辨率、ARGB8888 格式，使用两块 buffer，需要的内存 vram0_size = 1280*720*4*2 = 7200 K。

参数默认值

如果加载 GFBG 驱动时不带任何参数或者传入的 size，或者超出能力集所能用到的 size，则系统默认配置 G0 的 size 8100K(XM7606V13 默认 8100K，XM7206V11A 默认 3600K)，G1 的 size 为 0，此时 G1 则不会初始化。

如用户需要使用 G1 图形层，首先需要卸载原有 GFBG 驱动，再通过修改 insmod 传参，设定 G1 层物理显存，才能正常使用。

用户需要从全局的角度出发配置 GFBG 需要管理的叠加图形层并且为每个叠加图形层分配适当的显存。

2.3 配置举例

配置 GFBG 管理叠加图形层的示例如下：

配置 GFBG 管理多个叠加图形层。

如果需要 GFBG 管理叠加图形层 0 和叠加图形层 1 两个图形层，且最大虚拟分辨率为 720 x 576，用到的像素格式为 ARGB1555，使用一块 buffer，则两个图形层需要的最小显存都为 720 x 576 x 2 = 829440 /1024= 810K（1024 向上对齐），配置参数如下：

insmod xm_fb.ko fb_vram="gfbg:vram0_size:810, vram1_size: 810"。

3 API 参考

3.1 API 类别

GFBG 的 API 分为以下几类：

- 文件操作类

  提供操作 GFBG 的接口。通过调用这些接口，可以像操作文件一样操作图形层。这些接口是 Linux 本身提供的标准接口，主要有 open、close 等。本文档不对这些标准接口进行描述。

- 显存映射类

  提供将物理显存映射到用户虚拟内存空间的接口。这些接口是 Linux 本身提供的标准接口，主要有 mmap、munmap 等。本文档不对这些标准接口进行描述。

- 显存控制和状态查询类

  允许设置像素格式和颜色深度等属性的接口。这些接口是 Linux 本身提供的标准接口，经常使用。本文档将对其进行简要描述。

- 层间效果控制和状态查询类

  GFBG 可以管理多个图形图形层，每层可以设置 Alpha 值和原点等。相对于 Linux Framebuffer，这些是 GFBG 的新增功能。本文档将重点描述该部分。

3.2 标准功能

FBIOGET_VSCREENINFO

【目的】

获取 Framebuffer 屏幕的可变信息。

【语法】

```
int ioctl (int fd, FBIOGET_VSCREENINFO, *var);
```

【描述】

使用此接口获取屏幕的可变信息，主要包括分辨率和像素格式。信息的详细描述请参见 struct fb_var_screeninfo。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
FBIOGET_VSCREENINFO	ioctl 号。	输入
var	可变信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_FAILURE	失败。

【需求】

头文件：fb.h

【注意】

- 高清设备的图形层默认分辨率为 1920x1080，像素格式默认为 ARGB1555。
- 对于 XM7606V13 芯片，两层高清图形层默认分辨率为 1920x1080。

【举例】

```
struct fb_var_screeninfo vinfo;
if (ioctl(fd, FBIOGET_VSCREENINFO, &vinfo) < 0)
{
    return -1;
}
```

【相关接口】

FBIOPUT_VSCREENINFO

FBIOPUT_VSCREENINFO

【目的】

设置 Framebuffer 屏幕分辨率和像素格式等。

【语法】

```
int ioctl (int fd,    FBIOPUT_VSCREENINFO, struct fb_var_screeninfo *var);
```

【描述】

使用此接口设置屏幕分辨率、像素格式。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
FBIOPUT_VSCREENINFO	ioctl 号。	输入
var	可变信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	GFBG 上下文环境为 NULL。
XMEDIA_ERRCODE_INVALID_PARAM	设置的格式不支持。
XMEDIA_ERRCODE_INVALID_DEV_ID	图形层 ID 不合法。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：fb.h

【注意】

- 分辨率的大小必须在各图形层支持的分辨率范围内，各图形层支持的最大分辨率和最小分辨率可通过 GFBG_GET_CAPABILITY 获取。
- 必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则系统会自动调整实际分辨率的大小让其在虚拟分辨率范围内。

【举例】

设置实际分辨率为 720x576，虚拟分辨率为 720x576，偏移为（0，0），像素格式为 ARGB1555 的示例代码如下：

```
struct fb_bitfield r16 = {10, 5, 0};
struct fb_bitfield g16 = {5, 5, 0};
struct fb_bitfield b16 = {0, 5, 0};
struct fb_bitfield a16 = {15, 1, 0};
struct fb_var_screeninfo vinfo;
if (ioctl(fd, FBIOGET_VSCREENINFO, &vinfo) < 0)
{
    return -1;
}
vinfo.xres_virtual = 720;
vinfo.yres_virtual = 576;
vinfo.xres = 720;
vinfo.yres = 576;
vinfo.activate = FB_ACTIVATE_NOW|FB_ACTIVATE_FORCE;
vinfo.bits_per_pixel = 16;
vinfo.xoffset = 0;
vinfo.yoffset = 0;
vinfo.red = r16;
vinfo.green = g16;
vinfo.blue = b16;
vinfo.transp= a16;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &vinfo) < 0)
{
    return -1;
}
```

【相关接口】

FBIOGET_VSCREENINFO

FBIOGET_FSCREENINFO

【目的】

获取 Framebuffer 的固定信息。

【语法】

```
int ioctl (int fd, FBIOGET_FSCREENINFO, struct fb_fix_screeninfo *fix);
```

【描述】

使用此接口获取 Framebuffer 固定信息，包括显存起始物理地址、显存大小和行间距等。信息的详细描述请参见 struct fb_fix_screeninfo。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
FBIOGET_FSCREENINFO	ioctl 号。	输入
fix	固定信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_FAILURE	失败。

【需求】

头文件：fb.h

【注意】

无。

【举例】

无。

【相关接口】

无。

FBIOPUTCMAP

【目的】

设置 cmap 表。

【语法】

```
int ioctl (int fd, FBIOPUTCMAP, struct fb_cmap *cmap);
```

【描述】

此接口用于配置 cmap 表，应用于 CLUT8 读表模式。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
FBIOPUTCMAP	ioctl 号。	输入
cmap	cmap 表结构体指针。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_ERRCODE_NOT_ENOUGH_MEMORY	保存 cmap 表的内存异常。
XMEDIA_ERRCODE_INVALID_PARAM	传入 cmap 参数不合法。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：fb.h

【注意】

无。

【举例】

无。

【相关接口】

无。

FBIOPAN_DISPLAY

【目的】

设置从虚拟分辨率中的不同偏移处开始显示。

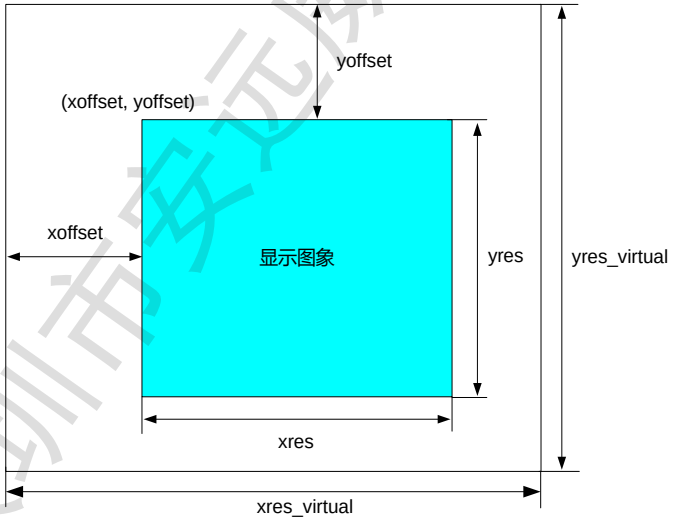
【语法】

```
int ioctl (int fd, FBIOPAN_DISPLAY, struct fb_var_screeninfo *var);
```

【描述】

使用此接口设置从虚拟分辨率中的不同偏移处开始显示，实际的分辨率不变。如图 3-1 所示：（xres_virtual, yres_virtual）是虚拟分辨率，（xres, yres）是实际显示的分辨率，(xoffset, yoffset)是显示的偏移。

图3-1 设置从虚拟分辨率中的不同偏移处开始显示



【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
FBIOPAN_DISPLAY	ioctl 号。	输入
var	可变信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_INVALID_PARAM	参数不合法。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：fb.h

【注意】

必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则设置不成功。

【举例】

设置实际分辨率为 300x300，虚拟分辨率为 720x576，起始偏移为（50，50），然后偏移到（300，0）处开始显示的 FBIOPAN_DISPLAY 设置代码如下：

```
struct fb_bitfield r32 = {16, 8, 0};
struct fb_bitfield g32 = {8, 8, 0};
struct fb_bitfield b32 = {0, 8, 0};
struct fb_bitfield a32 = {24, 8, 0};
struct fb_var_screeninfo vinfo;

vinfo.xres_virtual = 720;
vinfo.yres_virtual = 576;
vinfo.xres = 300;
vinfo.yres = 300;
vinfo.activate = FB_ACTIVATE_NOW| FB_ACTIVATE_FORCE;
vinfo.bits_per_pixel = 32;
vinfo.xoffset = 50;
```

```
vinfo.yoffset = 50;
vinfo.red = r32;
vinfo.green = g32;
vinfo.blue = b32;
vinfo.transp= a32;
if (ioctl(fd, FBIOPUT_VSCREENINFO, &vinfo) < 0)
{
      return -1;
}
vinfo.xoffset = 300;
vinfo.yoffset = 0;
if (ioctl(fd, FBIOPAN_DISPLAY, &vinfo) < 0)
{
return -1;
}
```

3.3 扩展功能

GFBG_GET_CAPABILITY

【目的】

获取图形层能力集。

【语法】

```
int ioctl (int fd, GFBG_GET_CAPABILITY, *cap);
```

【描述】

在使用其它功能接口前，用户可以通过调用此接口查询该图形层的能力。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_CAPABILITY	ioctl 号。	输入
cap	支持能力结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	失败。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

无。

GFBG_GET_SCREEN_POS

【目的】

获取图形层在屏幕上显示的起始点坐标。

【语法】

```
int ioctl (int fd, GFBG_GET_SCREEN_POS, gfbg_drv_point *point);
```

【描述】

使用此接口获取图形层在屏幕上显示的起始点坐标。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_SCREEN_POS	ioctl 号。	输入

参数名称	描述	输入/输出
point	坐标原点结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	失败。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

GFBG_PUT_SCREEN_POS

GFBG_PUT_SCREEN_POS

【目的】

设置图形层在屏幕上显示的起始点坐标。

【语法】

```
int ioctl (int fd, GFBG_PUT_SCREEN_POS, gfbg_drv_point *point);
```

【描述】

使用此接口设置图形层在屏幕上显示的起始点坐标，坐标范围：(0, 0）到 VO 显示分辨率减图形层支持的最小分辨率差值。比如 VO 显示分辨率为 1920x1080，图形层支持的最小分辨率为 32x32，则坐标范围是：[0，0]~[1887，1046]。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_PUT_SCREEN_POS	ioctl 号。	输入
point	坐标原点结构体指针 动态属性。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_INVALID_PARAM	参数不合法。
XMEDIA_ERRCODE_NULL_PTR	参数无效。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

如果图形层坐标原点超出了范围，默认将坐标原点设置为（screen_size.w-min_width，screen_size.h - min_height），其中 screen_size.w 和 screen_size.w 的值是 VO 设备当前的显示分辨率；min_width 和 min_height 分别表示图形层最小宽和高，可通过 GFBG_GET_CAPABILITY 接口中的 min_width 和 min_height t 成员获取。

【举例】

无。

【相关接口】

GFBG_PUT_SCREEN_POS

GFBG_GET_SHOW

【目的】

获取当前图形层的显示状态。

【语法】

```
int ioctl (int fd, GFBG_GET_SHOW, xmedia_bool *show);
```

【描述】

使用此接口获取当前图形层显示状态。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_SHOW	ioctl 号。	输入
show	指示当前图形层的状态： • * show = XMEDIA_TRUE：当前图形层处于显示状态。 • * show = XMEDIA_FALSE：当前图形层处于隐藏状态。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	失败。

【需求】

头文件：xmedia_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

GFBG_PUT_SHOW

GFBG_PUT_SHOW

【目的】

显示或隐藏该图形层。

【语法】

```
int ioctl (int fd, GFBG_PUT_SHOW, XMEDIA_BOOL *show);
```

【描述】

使用此接口设置图形层显示状态：显示或隐藏。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_PUT_SHOW	ioctl 号。	输入
show	该图形层的显示状态： • *show = XMEDIA _TRUE：显示当前图形层。 • *show = XMEDIA _FALSE：隐藏当前图形层。 动态属性。	输入

【返回值】

返回值	描述
0	成功。

XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_ERRCODE_INVALID_PARAM	参数无效。
XMEDIA_FAILURE	失败。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

- 图形层 open 之后默认是 XMEDIA _TRUE 状态。
- 显示时应保证图形层的分辨率不超出设备分辨率。

【举例】

无。

【相关接口】

GFBG_GET_SHOW

GFBG_GET_ALPHA

【目的】

获取图形层 alpha。

【语法】

```
int ioctl (int fd, GFBG_GET_ALPHA, gfbg_drv_alpha * alpha);
```

【描述】

使用此接口获取当前图形层的 alpha 设置。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_ALPHA	ioctl 号。	输入
alpha	alpha 结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

GFBG_PUT_ALPHA

GFBG_PUT_ALPHA

【目的】

设置图形层的 alpha。

【语法】

```
int ioctl (int fd, GFBG_PUT_ALPHA, gfbg_drv_alpha *alpha);
```

【描述】

使用此接口设置当前图形层的 alpha 功能，alpha 功能分 pix alpha 和 global alpha,可以单独使能也可以全部使能，详见 gfbg_drv_alpha。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入

参数名称	描述	输入/输出
GFBG_PUT_ALPHA	ioctl 号。	输入
alpha	alpha 结构体指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_ERRCODE_INVALID_PARAM	参数不合法。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

GFBG_GET_ALPHA

GFBG_GET_COLORKEY

【目的】

获取图形层的 colorkey。

【语法】

```
int ioctl (int fd, GFBG_GET_COLORKEY, gfbg_drv_colorkey * colorkey);
```

【参数】

参数名称	描述	输入/输出
Fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_COLORKEY	ioctl 号。	输入
colorkey	colorkey 结构体指针。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

无。

【相关接口】

GFBG_PUT_COLORKEY

GFBG_PUT_COLORKEY

【目的】

设置图形层的 colorkey。

【语法】

```
int ioctl (int fd, GFBG_PUT_COLORKEY,   gfbg_drv_colorkey *colorkey);
```

【描述】

使用此接口设置当前图形层的 colorkey 功能，和 colorkey 匹配的像素点将会透出背景色，colorkey 参数描述详见 gfbg_drv_colorkey。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_PUT_COLORKEY	ioctl 号。	输入
colorkey	colorkey 结构体指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_ERRCODE_INVALID_PARAM	参数无效。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h

【注意】

无。

【举例】

假设当前像素格式为 ARGB8888，则要过滤掉红色分量为 0x1F、绿色分量为 0x2F、蓝色分量为 0x3F 组合的颜色值颜色值 0x1F2F3F，具体设置如下：

```
colorkey colorkey;

colorkey.key_enable = XMEDIA_TRUE;
colorkey.key = 0x1F2F3F;
if (ioctl(fd, GFBG_PUT_COLORKEY, &colorkey) < 0)
```

```
{
    return -1;
}
```

【相关接口】

GFBG_GET_COLORKEY

GFBG_GET_VBLANK

【目的】

为了操作显存时不引起撕裂现象，建议在该图形层的垂直消隐区对显存进行操作，通过该接口可以等待该图形层垂直消隐区的到来。

【语法】

```
int ioctl (int fd, GFBG_GET_VBLANK, gfbg_drv_vblank_info *vblank);
```

【描述】

使用此接口获取当前图形层的消隐区。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_VBLANK	ioctl 号。	输入
vblank	用户获取当前真实刷新帧率及当前时间信息 此参数指针可以为 NULL。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_TIMEOUT	获取 vblank 超时。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h。

【注意】

当 struct fb_var_screeninfo 中的成员 activate 置位 FB_ACTIVATE_VBL，则表示阻塞送帧，就不需要调用此接口等待消隐区的到来。

【举例】

无。

【相关接口】

无。

GFBG_GET_PRE_MULTIPLY

【目的】

获取预乘状态。

【语法】

```
int ioctl (int fd, GFBG_GET_PRE_MULTIPLY, xmedia_bool *pre_multiply);
```

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_GET_PRE_MULTIPLY,	ioctl 号。	输入
pre_multiply	预乘使能状态。 pre_multiply = XMEDIA_TRUE：开启当前图形层预乘功能。 pre_multiply = XMEDIA_FALSE：关闭当前图形层预乘功能。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h。

【注意】

无。

【举例】

无。

【相关接口】

GFBG_PUT_PRE_MULTIPLY

GFBG_PUT_PRE_MULTIPLY

【目的】

设置预乘状态。

【语法】

```
int ioctl (int fd, GFBG_PUT_PRE_MULTIPLY, xmedia_bool *pre_multiply);
```

【描述】

设置预乘状态，状态为 TRUE 则会开启预乘功能，在和视频层叠加前，将会把 alpha 属性预先乘进像素位再做叠加。已经做过预乘的数据则不需要使能此状态位，未做过预乘的数据需要使能此状态位。

【参数】

参数名称	描述	输入/输出
fd	Framebuffer 设备文件描述符。	输入
GFBG_PUT_PRE_MULTIPLY,	ioctl 号。	输入

pre_multiply	预乘使能状态。 pre_multiply = XMEDIA_TRUE：开启当前图形层预乘功能。 pre_multiply = XMEDIA_FALSE：关闭当前图形层预乘功能。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
XMEDIA_ERRCODE_NULL_PTR	参数空指针。
XMEDIA_FAILURE	其他异常情况。

【需求】

头文件：drv_gfbg_ioctl.h。

【注意】

无。

【举例】

无。

【相关接口】

GFBG_GET_PRE_MULTIPLY

3.4 数据类型

struct fb_bitfield

【说明】

位域信息，用于设置像素格式。

【定义】

```
struct fb_bitfield
{
    __u32 offset;          /* beginning of bitfield */
    __u32 length;          /* length of bitfield */
    __u32 msb_right;       /* != 0: Most significant bit is right */
};
```

【成员】

成员名称	描述	支持情况
offset	颜色分量起始比特位。	支持。
length	颜色分量所占比特长度。	支持。
msb_right	右边的比特是否为最高有效位。	只支持该位为 0，即最左边的 bit 为最高有效位。

【注意事项】

例如 ARGB1555 格式，其位域信息的赋值如下：

```
struct fb_bitfield a16 = {15, 1, 0};
struct fb_bitfield r16 = {10, 5, 0};
struct fb_bitfield g16 = {5, 5, 0};
struct fb_bitfield b16 = {0, 5, 0};
```

【相关数据类型和接口】

无。

struct fb_var_screeninfo

【说明】

可变的屏幕信息。

【定义】

```
struct fb_var_screeninfo
{
    __u32 xres;                    /* visible resolution */
    __u32 yres;
    __u32 xres_virtual;            /* virtual resolution */
```

```
    __u32 yres_virtual;
    __u32 xoffset;              /* offset from virtual to visible */
    __u32 yoffset;              /* resolution */

    __u32 bits_per_pixel;       /* guess what */
    __u32 grayscale;            /* != 0 Graylevels instead of colors */

    struct fb_bitfield red;     /* bitfield in fb mem if true color, */
    struct fb_bitfield green;   /* else only length is significant */
    struct fb_bitfield blue;
    struct fb_bitfield transp;  /* transparency */

    __u32 nonstd;               /* != 0 Non standard pixel format */

    __u32 activate;             /* see FB_ACTIVATE_* */

    __u32 height;               /* height of picture in mm */
    __u32 width;                    /* width of picture in mm */

    __u32 accel_flags;          /* (OBSOLETE) see fb_info.flags */

    /* Timing: All values in pixclocks, except pixclock (of course) */
    __u32 pixclock;             /* pixel clock in ps (pico seconds) */
    __u32 left_margin;          /* time from sync to picture */
    __u32 right_margin;         /* time from picture to sync */
    __u32 upper_margin;         /* time from sync to picture */
    __u32 lower_margin;
    __u32 hsync_len;            /* length of horizontal sync */
    __u32 vsync_len;            /* length of vertical sync */
    __u32 sync;                     /* see FB_SYNC_* */
    __u32 vmode;                    /* see FB_VMODE_* */
    __u32 rotate;               /* angle we rotate counter clockwise */
    __u32 reserved[5];          /* Reserved for future compatibility */
};
```

【成员】

成员名称	描述	支持情况
xres	可见屏幕宽度（像素数）。	支持，fb0，fb1 默认值为 1920。
yres	可见屏幕高度（像素数）。	支持，fb0，fb1 默认为 1080。

成员名称	描述	支持情况
xres_virtual	虚拟屏幕宽度（显存中图像宽度）。	支持，fb0，fb1 默认值为 1920。
yres_virtual	虚拟屏幕高度（显存中图像高度）。结合 xres_virtual，可以用来快速水平或垂直平移图像。	支持，fb0，fb1 默认为 2160。
xoffset	在 x 方向上的偏移像素数。	支持，默认为 0。
yoffset	在 y 方向上的偏移像素数。	支持，默认为 0。
bits_per_pixel	每个像素所占的比特数。	支持，默认为 16。
grayscale	灰度级。	不支持，缺省值为 0，表示彩色。
red	颜色分量中红色的位域信息。	支持，默认为（10，5，0）。
green	颜色分量中绿色的位域信息。	支持，默认为（5，5，0）。
blue	颜色分量中蓝色的位域信息。	支持，默认为（0，5，0）。
transp	颜色分量中 alpha 分量的位域信息。	支持，默认为（15，1，0）。
nonstd	是否为标准像素格式。	不支持，缺省值为 0，表示支持标准像素格式。
activate	FB 状态标识。	FB_ACTIVATE_VBL 状态被赋值时 FBIOPAN_DISPLAY 送显时，等待一个 VBLANK 再返回。 FB_ACTIVATE_FORCE 状态被赋值时原生 fb 将会忽略 var_info 是否修改的判断，强制去调用 gfbg 驱动接口。
height	屏幕高，单位为 mm。	不支持，缺省值为–1。
width	屏幕宽，单位为 mm。	不支持，缺省值为–1。
accel_flags	加速标志。	不支持，缺省值为–1。

成员名称	描述	支持情况
pixclock	显示一个点需要的时间，单位为 ns。	不支持，缺省值为–1。
left_margin right_margin hsync_len	分别是左消隐信号、右消隐信号、水平同步时长，这三个值之和等于水平回扫时间，单位为点时钟。	不支持，缺省值为–1。
upper_margin lower_margin vsync_len	分别是上消隐信号、下消隐信号、垂直同步时长，这三个值之和等于垂直回扫时间，单位为点时钟。	不支持，缺省值为–1。
sync	同步信号方式。	不支持，缺省值为–1。
vmode	扫描模式。	不支持，缺省值为–1。
rotate	顺时针旋转的角度。	不支持，缺省值为 0，表示无旋转。

【注意】

XM7606V13 图形层默认分辨率为 1920x1080。像素格式为 ARGB1555。

XM7206V11A 图形层默认分辨率为 1280x720。像素格式为 ARGB1555。

【相关数据类型及接口】

- struct fb_bitfield
- FBIOGET_VSCREENINFO
- FBIOPUT_VSCREENINFO

struct fb_fix_screeninfo

【说明】

固定的屏幕信息。

【定义】

```
struct fb_fix_screeninfo
{
```

```
    char id[16];            /* identification string eg "TT Builtin" */
    unsigned long smem_start;       /* Start of frame buffer mem (physical
                                       address) */
    __u32 smem_len;                 /* Length of frame buffer mem */
    __u32 type;                     /* see FB_TYPE_* */
    __u32 type_aux;                 /* Interleave for interleaved Planes */
    __u32 visual;                   /* see FB_VISUAL_* */
    __u16 xpanstep;                 /* zero if no hardware panning */
    __u16 ypanstep;                 /* zero if no hardware panning */
    __u16 ywrapstep;                /* zero if no hardware ywrap */
    __u32 line_length;              /* length of a line in bytes */
    unsigned long mmio_start;       /* Start of Memory Mapped I/O (physical
                                       address) */
    __u32 mmio_len;                 /* Length of Memory Mapped I/O */
    __u32 accel; /* Indicate to driver which specific chip/card we have */
    __u16 reserved[3];              /* Reserved for future compatibility */
};
```

【成员】

成员名称	描述	支持情况
id	设备驱动名称。	支持。
smem_start	显存起始物理地址。	支持。
smem_len	显存大小。	支持。
type	显卡类型。	不支持。
type_aux	附加类型。	不支持。
visual	色彩模式。	不支持。
xpanstep	支持水平方向上的 PAN 显示： • 0：不支持。 • 非 0：支持，此时该值用于表示在水平方向上每步进的像素值。	固定为 1。

成员名称	描述	支持情况
ypanstep	支持垂直方向上的 PAN 显示： • 0：不支持。 • 非 0：支持，此时该值用于表示在垂直方向上每步进的像素值。	固定为 1。
ywrapstep	该方式类似于 ypanstep，不同之处在于：当其显示到底部时，能回到显存的开始处进行显示。	不支持。
line_length	每行字节数。	支持。
mmio_start	显存映射 I/O 首地址。	不支持。
mmio_len	显存映射 I/O 长度。	不支持。
accel	显示所支持的硬件加速设备。	不支持。
reserved	保留。	不支持。

【注意】

无。

【相关数据类型及接口】

FBIOGET_FSCREENINFO

gfbg_drv_fmt

【说明】

GFBG 格式。

【定义】

```
typedef enum {
    GFBG_DRV_FMT_RGB565 = 0,
    GFBG_DRV_FMT_RGB888,
    GFBG_DRV_FMT_KRGB444,
    GFBG_DRV_FMT_KRGB555,
```

```
    GFBG_DRV_FMT_KRGB888,
    GFBG_DRV_FMT_ARGB4444,
    GFBG_DRV_FMT_ARGB1555,
    GFBG_DRV_FMT_ARGB8888,

    GFBG_DRV_FMT_ARGB8565,
    GFBG_DRV_FMT_RGBA4444,
    GFBG_DRV_FMT_RGBA5551,
    GFBG_DRV_FMT_RGBA5658,

    GFBG_DRV_FMT_RGBA8888,
    GFBG_DRV_FMT_BGR565,
    GFBG_DRV_FMT_BGR888,
    GFBG_DRV_FMT_ABGR4444,

    GFBG_DRV_FMT_ABGR1555,
    GFBG_DRV_FMT_ABGR8888,
    GFBG_DRV_FMT_ABGR8565,
    GFBG_DRV_FMT_KBGR444,

    GFBG_DRV_FMT_KBGR555,
    GFBG_DRV_FMT_KBGR888,
    GFBG_DRV_FMT_1BPP,
    GFBG_DRV_FMT_2BPP,

    GFBG_DRV_FMT_4BPP,
    GFBG_DRV_FMT_8BPP,
    GFBG_DRV_FMT_ACLUT44,
    GFBG_DRV_FMT_ACLUT88,

    GFBG_DRV_FMT_PUYVY,
    GFBG_DRV_FMT_PYUYV,
    GFBG_DRV_FMT_PYVYU,
    GFBG_DRV_FMT_YUV888,

    GFBG_DRV_FMT_AYUV8888,
    GFBG_DRV_FMT_YUVA8888,

    GFBG_DRV_FMT_ARGB2101010,
    GFBG_DRV_FMT_ARGB10101010,
    GFBG_DRV_FMT_FP16,

    GFBG_DRV_FMT_MAX
```

```
} gfbg_drv_fmt;
```

【注意事项】

目前 XM7606V13 支持的格式如下，可以通过 GFBG_GET_CAPABILITY 获取到：

GFBG_DRV_FMT_ARGB1555,

GFBG_DRV_FMT_RGBA5551，

GFBG_DRV_FMT_ABGR1555，

GFBG_DRV_FMT_ARGB8888,

GFBG_DRV_FMT_RGBA8888,

GFBG_DRV_FMT_ABGR8888，

GFBG_DRV_FMT_ARGB4444，

GFBG_DRV_FMT_RGBA4444，

GFBG_DRV_FMT_ABGR4444，

GFBG_DRV_FMT_8BPP,

【相关数据类型及接口】

gfbg_drv_capability

GFBG_GET_CAPABILITY

gfbg_layer_id

【说明】

GFBG 图形层 ID 定义。

【定义】

```
typedef enum
{
    GFBG_LAYER_ID_G0 = 0x0,
    GFBG_LAYER_ID_G1,
    GFBG_LAYER_ID_MAX
} gfbg_layer_id;
```

【注意事项】

XM7606V13 支持，GFBG_LAYER_ID_G0，GFBG_LAYER_ID_G1。

XM7206V11A 支持，GFBG_LAYER_ID_G0。

目前 GFBG 设计为一个图层对应一个/dev/fb*节点，所以 api 不需要传入图形层 ID。

【相关数据类型及接口】

无。

gfbg_drv_colorkey

【说明】

GFBG colorkey 定义。

【定义】

```
typedef struct {
    xmedia_bool key_enable;
    xmedia_u32 key;
} gfbg_drv_colorkey;
```

【成员】

成员名称	描述
key_enable	colorkey 使能状态。
key	colorkey 值。

【注意事项】

- key 值是依据格式来定义的。
- 比如 ARGB1555 格式（ARGB8888 ARGB4444 类似），那么 key 值则是 R 占 5 位，G 占 5 位，B 占 5 位，A 位无效，可填可不填，假设需要 ARGB1555 格式的 colorkey 为红色，那么 colorkey 值为 0x7C00。
- 如果是 CLUT8 格式，那么 key 值则是 CMAP 表的索引号。

【相关数据类型及接口】

GFBG_GET_COLORKEY

GFBG_PUT_COLORKEY

gfbg_drv_point

【说明】

GFBG 显示原点。

【定义】

```
typedef struct {
    xmedia_s32 x;
    xmedia_s32 y;
} gfbg_drv_point;
```

【成员】

成员名称	描述
x	显示原点 x 坐标
y	显示原点 y 坐标

【注意事项】

无。

【相关数据类型及接口】

GFBG_GET_SCREEN_POS

GFBG_PUT_SCREEN_POS

gfbg_drv_alpha

【说明】

GFBG 图形层 alpha 结构体。

【定义】

```
typedef struct {
    xmedia_bool pixel_alpha;
    xmedia_bool global_alpha;
    xmedia_u8 alpha0;
    xmedia_u8 alpha1;
    xmedia_u8 global_alpha_value;
    xmedia_u8 reserved;
} gfbg_drv_alpha;
```

【成员】

成员名称	描述
pixel_alpha	像素 alpha 使能。
global_alpha	全局 alpha 使能。
alpha0	argb1555 格式 alpha 位为 0 像素 alpha 替换值设置。
alpha1	argb1555 格式 alpha 位为 1 像素 alpha 替换值设置。
global_alpha_value	全局 alpha 值。
reserved	保留。

【注意事项】

无。

【相关数据类型及接口】

GFBG_GET_ALPHA

GFBG_PUT_SHOW

gfbg_drv_time_val

【说明】

GFBG vblank 时间信息。

【定义】

```
typedef struct {
    xmedia_u64 tv_sec;
    xmedia_u64 tv_msec;
    xmedia_u64 tv_usec;
    xmedia_u64 tv_nsec;
} gfbg_drv_time_val;
```

【成员】

成员名称	描述
tv_sec	当前时间 秒计数。

成员名称	描述
tv_msec	当前时间 毫秒计数。
tv_usec	当前时间 微秒计数。
tv_nsec	当前时间 纳秒计数。

【注意事项】

当前只使用了秒和微秒计数。

【相关数据类型及接口】

gfbg_drv_vblank_info

GFBG_GET_VBLANK

gfbg_drv_vblank_info

【说明】

GFBG vblank 结构体。

【定义】

```
typedef struct {
    xmedia_u32 refresh_rate;
    gfbg_drv_time_val time_val;
} gfbg_drv_vblank_info;
```

【成员】

成员名称	描述
refresh_rate	当前实际的图形层刷新帧率。
time_val	当前时间信息。

【注意事项】

无。

【相关数据类型及接口】

GFBG_GET_VBLANK

gfbg_drv_capability

【说明】

GFBG 能力集结构体。

【定义】

```
typedef struct {
    xmedia_bool key_rgb;
    xmedia_bool key_alpha;
    xmedia_bool global_alpha;
    xmedia_bool cmap;
    xmedia_bool has_cmap_reg;
    xmedia_bool col_fmt[GFBG_DRV_FMT_MAX];
    xmedia_bool pre_mul;
    xmedia_u32   max_width;
    xmedia_u32   max_height;
    xmedia_u32   min_width;
    xmedia_u32   min_height;
} gfbg_drv_capability;
```

【成员】

成员名称	描述
key_rgb	是否支持 rgb colorkey。
key_alpha	是否支持 alpha colorkey。
global_alpha	是否支持全局 alpha。
cmap	是否支持 clut 读表模式。
has_cmap_reg	是否支持硬件 clut 读表模式。
col_fmt	格式支持种类。
pre_mul	是否支持预乘。
max_width	最大宽度。
max_height	最大高度。
min_width	最小宽度。

成员名称	描述
min_height	最小高度。

【注意事项】

无。

【相关数据类型及接口】

gfbg_drv_fmt

GFBG_GET_CAPABILITY

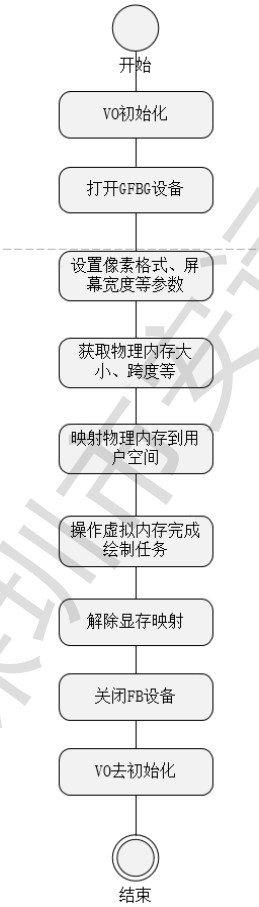
4 GFBG 应用开发流程

4.1 开发流程

GFBG 主要用于显示 2D 图形（以直接操作物理显存的方式）。

GFBG 的开发流程如图 4-1 所示。

图4-1 GFBG 的开发流程



GFBG 的开发步骤如下：

步骤 1 调用 VO 初始化接口。

步骤 2 调用 open 函数打开指定的 GFBG 设备。

步骤 3 调用 ioctl 函数设置 GFBG 的像素格式以及屏幕高宽等参数。

步骤 4 调用 ioctl 函数获取 GFBG 所分配的物理显存大小、跨度等固定信息。调用 ioctl 函数也可以使用 GFBG 提供的层间 colorkey、层间 alpha、原点偏移等功能。

步骤 5 调用 mmap 函数将物理显存映射到虚拟内存空间。

步骤 6 操作虚拟内存，完成具体的绘制任务。

步骤 7 调用 munmap 解除显存映射。

步骤 8 调用 close 函数关闭设备。

步骤 9 去初始化 VO。

----结束

说明

由于修改虚拟分辨率将改变 GFBG 的固定信息 fb_fix_screeninfo:line_length（跨度），为保证绘制程序能够正确执行，推荐先设置 GFBG 的可变信息 fb_var_screeninfo，再获取 GFBG 的固定信息 fb_fix_screeninfo:line_length。

GFBG 各个开发各阶段完成的任务如表 4-1 所示。

表4-1 GFBG 的开发阶段任务表

阶段	任务
初始化阶段	完成显示属性的设置和物理显存的映射。
绘制阶段	完成具体的绘制工作。
终止阶段	完成资源清理工作。

4.2 实例介绍

本实例利用 PAN_DISPLAY 连续显示 15 幅分辨率为 640×352 的图片，以达到动态显示的效果。

每个文件存储的都是像素格式为 ARGB1555 的纯数据（不包含附加信息的图像数据）。

【参考代码】

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/fb.h>
#include <assert.h>
#include <signal.h>
#include <time.h>
#include <sys/sysinfo.h>
#include <sys/time.h>

#include "xmedia_sys.h"

#include "drv_gfbg_ioctl.h"
#include "xmedia_sys.h"
#include "xmedia_vo.h"

#define IMAGE_WIDTH      640
#define IMAGE_HEIGHT     352
#define IMAGE_SIZE       (640*352*2)
#define IMAGE_NUM        14
#define IMAGE_PATH       "./res/%d.bits"

static struct fb_bitfield g_r16 = {10, 5, 0};
static struct fb_bitfield g_g16 = {5, 5, 0};
static struct fb_bitfield g_b16 = {0, 5, 0};
static struct fb_bitfield g_a16 = {15, 1, 0};

xmedia_s32 gfbg_demo_test(xmedia_void)
{
    int fd;
    int i;
```

```
    struct fb_fix_screeninfo fix;
    struct fb_var_screeninfo var;
    unsigned char *p_show_screen;
    unsigned char *p_hide_screen;

    gfbg_drv_point st_point = {40, 112};
    FILE *fp;
    char image_name[128];
    xmedia_s32 ret = XMEDIA_SUCCESS;

    xmedia_vo_dev dev = XMEDIA_VO_DEV_0;
    xmedia_vo_dev_config dev_config;

    dev_config.intf_sync = XMEDIA_VO_INTF_SYNC_1080I50;
    dev_config.bg_color = 0xff;
    dev_config.intf_config.intf_type = XMEDIA_INTF_TYPE_BT1120;
    dev_config.intf_config.bt1120_attr.data_seq = XMEDIA_INTF_BT1120_DATA_SEQ_UVUV;
    dev_config.intf_config.bt1120_attr.data_type =
XMEDIA_INTF_BT_DATA_TYPE_YUV422_8BIT;

    //enable vo dev
    xmedia_vo_set_dev_config(dev, &dev_config);
    xmedia_vo_enable_dev(dev);

    //open fb
    fd = open("/dev/fb0", O_RDWR);
    if(fd < 0)
    {
        printf("open fb0 failed!\n");
        ret = XMEDIA_FAILURE;
        goto error0;
    }

    //set fb position
    if (ioctl(fd, GFBG_PUT_SCREEN_POS, &st_point) < 0)
    {
        printf("set screen original show position failed!\n");
        ret = XMEDIA_FAILURE;
        goto error1;
    }

    //get fb var info
    if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0)
```

```
    {
        printf("Get variable screen info failed!\n");
        ret = XMEDIA_FAILURE;
        goto error1;
    }

    /* modify the variable screen info
     * the screen size: IMAGE_WIDTH*IMAGE_HEIGHT
     * the virtual screen size: IMAGE_WIDTH*(IMAGE_HEIGHT*2)
     * the pixel format: ARGB1555
     */
    var.xres = var.xres_virtual = IMAGE_WIDTH;
    var.yres = IMAGE_HEIGHT;
    var.yres_virtual = IMAGE_HEIGHT*2;

    var.transp= g_a16;
    var.red = g_r16;
    var.green = g_g16;
    var.blue = g_b16;
    var.bits_per_pixel = 16;

    //set fb var info
    if (ioctl(fd, FBIOPUT_VSCREENINFO, &var) < 0)
    {
        printf("Put variable screen info failed!\n");
        ret = XMEDIA_FAILURE;
        goto error1;
    }

    //get fb fix info
    if (ioctl(fd, FBIOGET_FSCREENINFO, &fix) < 0)
    {
        printf("Get fix screen info failed!\n");
        ret = XMEDIA_FAILURE;
        goto error1;
    }

    // map the physical video memory for user use
    p_show_screen = mmap(NULL, fix.smem_len, PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0);
    p_hide_screen = p_show_screen + IMAGE_SIZE;
    memset(p_show_screen, 0, IMAGE_SIZE);
```

```
    // load the bitmaps from file to hide screen and set pan display the hide screen
    for(i = 0; i < IMAGE_NUM; i++)
    {
        sprintf(image_name, IMAGE_PATH, i);
        fp = fopen(image_name, "rb");
        if(NULL == fp)
        {
            printf("Load %s failed!\n", image_name);
            ret = XMEDIA_FAILURE;
            goto error1;
        }

        fread(p_hide_screen, 1, IMAGE_SIZE, fp);
        fclose(fp);
        usleep(10);
        if(i%2)
        {
            var.yoffset = 0;
            p_hide_screen = p_show_screen + IMAGE_SIZE;
        }
        else
        {
            var.yoffset = IMAGE_HEIGHT;
            p_hide_screen = p_show_screen;
        }

        if (ioctl(fd, FBIOPAN_DISPLAY, &var) < 0)
        {
            printf("FBIOPAN_DISPLAY failed!\n");
            ret = XMEDIA_FAILURE;
            goto error1;
        }
    }

    printf("Enter to quit!\n");
    getchar();

error1:
    close(fd);
error0:
    xmedia_vo_enable_dev(dev);
    return ret;
}
```

5 图形界面开发

5.1 概述

5.1.1 模块概述

数字媒体处理平台提供一整套机制支持图形界面的开发，主要包括：

- 图形二维加速引擎（Two Dimensional Engine，简称 TDE），它利用硬件加速对图形进行处理。
- Graphic Framebuffer Group（以下简称 GFBG）用于管理叠加图形层，它不仅提供 Linux Framebuffer 的基本功能，还在 Linux Framebuffer 的基础上增加层间 colorkey、层间 Alpha 等扩展功能

说明

- TDE 相关使用方法请参见《TDE API 参考》。
- GFBG 相关使用方法请参见本文档。

5.1.2 场景概述

在监控领域中，一般输出设备的图形用户界面内容包括：

- 后端 OSD：显示画面分割线、通道号、时间等信息，用以界定多画面显示布局。
- GUI 界面：包括各种菜单、进度条等元素，用户通过操作 GUI 界面进行设备配置。
- 鼠标：提供更方便易用的界面菜单操作方式。

以上 3 类图形内容可以叠加到 1 个图形层实现。

5.2 实现用户界面方案

5.2.1 基本方案

该方案总体思路是：每个设备使用 1 个图形层来完成本设备的后端 OSD、UI 和鼠标的显示。

可具体描述为：每个输出设备使用一个图形层来完成本设备的后端 OSD、UI；UI 画在独立的 UI 画布上，后端 OSD 直接画在 FB 缓存上，再通过 TDE 进行 alpha 混合；鼠标可以画在 UI 画布上。同时图形层使用双缓存机制轮转，避免出现图形撕裂的情况如图 1-2。

该方案使用了以下机制：

- 每个设备的后端 OSD 直接绘制在各自的 FB 显存中。

  例如在每个图形层对应的 FB 显存中绘制分割布局、通道号或者时间。
- 每个设备使用一块独立的缓存绘制 UI（称该块缓存为 UI 画布）。
- 将绘制好的 UI 画布整体搬移到相应的 FB 缓冲中，在此过程中可利用 TDE 实现 UI 和 OSD 的叠加透明效果。
- FB 双缓冲

  为防止一块 FB 缓冲被边绘制边显示而导致绘制过程可见，推荐使用 FB 双缓冲机制。它们的原理都是为 FB 分配 2 块大小相同的缓冲作为显存交替绘制如图 1-2。

  1、假设 VO 正在显示缓冲 1，则本次绘制的对象为缓冲 0。

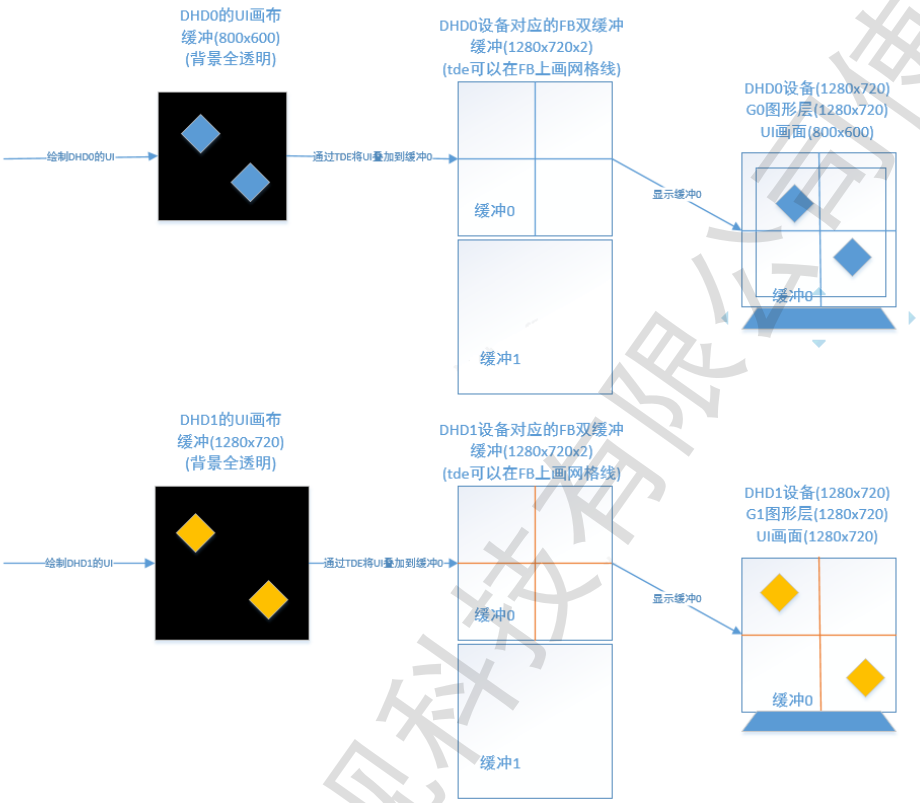
  2、缓冲 0 绘制完成后 FB 可通过 FB 的 FBIOPAN_DISPLAY 显示缓冲 0。

  3、可通过 GFBG_GET_VBLANK 获取 VSYNC 同步，以确保当前缓冲 0 送显已生效。

  4、继续绘制缓冲 1。

方案的结构如图 5-1 所示。

图5-1 基本方案结构图



该方案在后端 OSD 或者 UI 界面变动时，都需要重新绘制 FB 缓存：

- 本设备的后端 OSD 改变时，如 4 通道分割线切换到 9 通道分割线：先清空 FB 缓存，再绘制新的 OSD，再将 UI 界面整体搬移到 FB 缓存中。
- UI 界面每次变动时，都需要先清空 FB 缓存，再绘制 OSD，然后将新的 UI 界面整体搬移到 FB 缓存中。

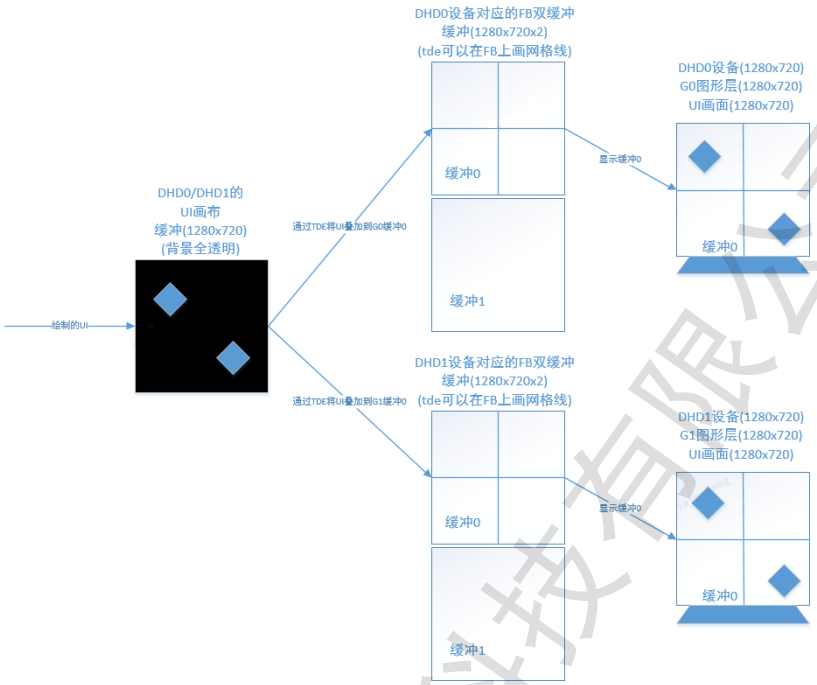
5.2.2 同源方案

当 DHD0 和 DHD1 设备上想同时显示同样的 UI 界面时，该方案可简化仅有一块 UI 画布缓存：

- 画布大小与 DHD0 的 UI 层大小相同（1280x720），用户可按照 DHD0 的 UI 规格（如 1280x720）准备一套图片。
- 每次更新画布后，利用 TDE 整体搬移至 G0 G1 层缓冲即可。

该同源方案的结构如图 5-2 所示

图5-2 同源方案结构图

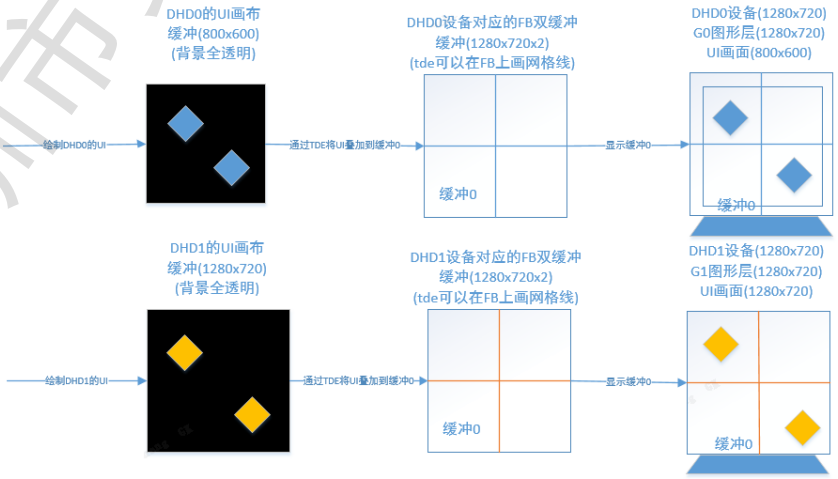


5.2.3 单缓存方案

本方案和基本方案类似，区别在于图形层只使用单个缓存来显示，优点可以节省 FB 内存，缺点是图形层会显示擦除及绘制的效果，屏幕出现撕裂等异常现象 如图 1-3。

结构图如下：

图5-3 单缓存方案结构图

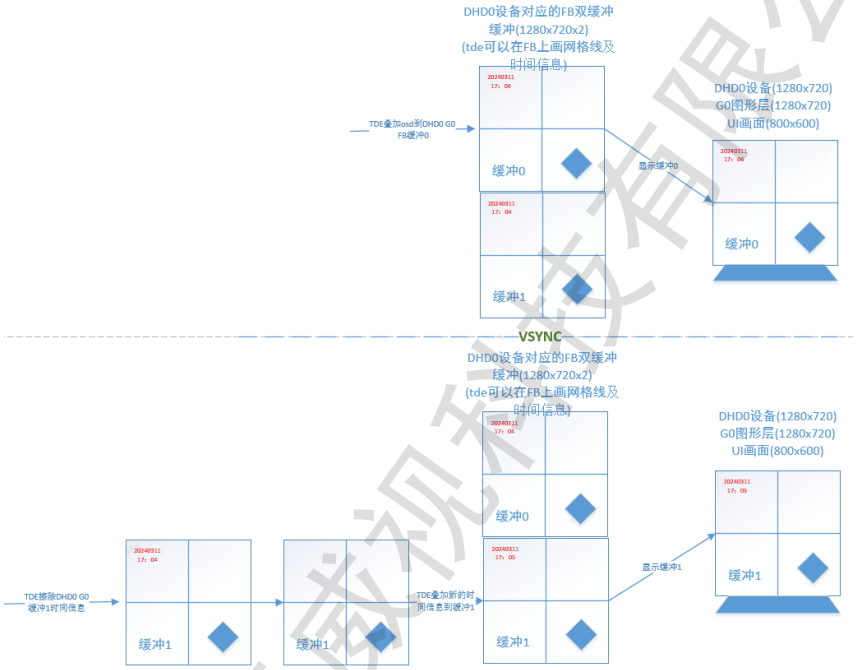


5.2.4 局部刷新方案

本方案主要通过调用 TDE 擦除以及叠加 FB 缓冲区局部区域，达到局部更新的效果。

通过 TDE 局部更新时间信息，结构图如下：

图5-4 局部刷新结构图



5.2.5 开发流程

以 5.2.2 同源方案为例：DHD0 DHD1 设备都做 4 画面等分分割线，且 DHD0 和 DHD1 同时显示同样的 UI。

若此时 UI 界面有变化，则该方案的实现过程为：

步骤 1 清空 DHD0 和 DHD1 对应图形层的 FB 的空闲缓冲（假设为缓冲 0，缓冲 1 正在被 VO 显示）。

步骤 2 在 DHD0 对应图形层的 FB 缓冲 0 中绘制 4 通道分割线。

步骤 3 在 DHD1 对应图形层的 FB 缓冲 0 中绘制 4 通道分割线。

步骤 4 更新 UI 画布。

步骤 5　用 TDE 将 UI 画布整体搬移到 DHD0 对应图形层的 FB 缓冲 0 的合适位置，此过程可以做 alpha 透明度叠加以实现 UI 半透明效果。

步骤 6　用 TDE 将 UI 画布整体搬移到 DHD1 对应图形层的 FB 缓冲 0 的合适位置，此过程可以做 alpha 透明度叠加以实现 UI 半透明效果。

步骤 7　通过 FB 接口调用 PAN_DISPLAY 通知 DHD0 显示本设备绑定图形层已准备好的 FB 的缓冲 0。

步骤 8　通过 FB 接口调用 PAN_DISPLAY 通知 DHD1 显示本设备绑定图形层已准备好的 FB 的缓冲 0。

步骤 9　通过 FB 接口调用 GFBG_GET_VBLANK DHD0 DHD1，获取到 VSYNC 同步状态，表示缓冲 0 已送显，接下来可操作缓冲 1。

----结束

6 PROC 调试

6.1 图形层和 GFBG 设备号对应关系

- 对于 XM7606V13 GFBG 最多可以管理 2 个叠加图形层：图形层 0，对应的设备文件是/dev/fb0，图形层 1，对应的设备文件是/dev/fb1。
- 对于 XM7206V11A GFBG 最多可以管理 1 个叠加图形层：图形层 0，对应的设备文件是/dev/fb0。
- 可通过命令 cat /proc/umap/gfbgn（n 为图形层号）查看各个图形层的状态。

6.2 单个图形层调试信息

【调试信息】

```
Version:[SPC020 B020], [GFBG V1.0 ], Release, Build Time[Feb 18 2025 11:30:18]
-----------------------------------GFBG STATUS-----------------------------------
FB_VERSION  V0.0.0.1
layer_id    |open_status |show_status |vo_dev      |ref_count   |cb_status
0           |1           |1           |0           |1           |1
-----------------------------------GFBG_VINFO-----------------------------------
xres        |yres        |xres_virtual |yres_virtual |bits_per_pix
800         |1280        |800         |2560        |16
-----------------------------------GFBG_FINFO-----------------------------------
smem_start  |smem_len    |xpanstep    |ypanstep    |line_length
50817000    |66355200    |1           |1           |1600
-----------------------------------GFBG_ATTR-----------------------------------
fmt                    |alpha0      |alpha1      |global_alpha |pix_alpha   |glo_alpha_val
GFBG_DRV_FMT_ARGB1555  |ff          |ff          |1           |1           |255
key_enable  |key         |pre_mul     |pos.x       |pos.y
0           |0           |0           |0           |0
luma        |contrast    |hue         |saturation  |csc_type    |bg_color
0           |0           |0           |0           |0           |0
-----------------------------------REFRESH-----------------------------------
phy_addr    |pre_phy_addr |phy_addr_off |canvas_size.w|canvas_size.h|disp_size.w |disp_size.h
50a0b000    |50817000    |1f4000      |800         |1280        |800         |1280
screen_size.w|screen_size.h|stride      |per_pix     |xoffset     |yoffset
800         |1280        |1600        |16          |0           |1280
-----------------------------------MEM_INFO-----------------------------------
buf_phy_addr |buf_vir_addr |buf_mem_len |cmap_phy_addr|cmap_vir_addr|cmap_mem_len
50817000    |ea0c014a    |66355200    |50029000    |8626875d    |1024
-----------------------------------DBG_INFO-----------------------------------
isr_all_cnt |isr_cb_cnt  |isr_cb_rate |isr_vo_cnt
136         |15          |60          |0
isr_max_time |isr_cur_time |isr_dbg_prt |ref_rate    |ref_cnt     |put_rate    |put_cnt
30          |14          |0           |0           |0           |0           |0
frame_time  |start_time  |delay_time
262392647   |264333      |2744
-----------------------------------CAPABILITY-----------------------------------
key_rgb     |key_alpha   |global_alpha |cmap        |has_cmap_reg |max_width   |max_height  |min_width
1           |0           |1           |1           |1           |3840        |2160        |32
min_height  |pre_mul
32          |1
-------------------------------------FB END-------------------------------------
```

【调试信息分析】

记录当前设备对应的图形层的内存配置信息及显示信息。

【参数说明】

类别	属性	描述
GFBG STATUS (状态信息)	GFBG_VERSION	软件版本
	open_status	图形层打开状态
	show_status	图层显示状态
	vo_dev	图形层绑定的 vo 设备号
	ref_count	图形层打开计数
	cb_status	GFBG 回调注册状态
GFBG_VINFO(可变信息)	xres	可见屏幕宽度
	yres	可见屏幕高度
	xres_virtual	虚拟屏幕宽度
	yres_virtual	虚拟屏幕高度
	bits_per_pix	每个像素所占的比特数
GFBG_FINFO(固定信息)	smem_start	显存起始物理地址
	smem_len	显存大小
	xpan_step	水平方向上每步进的像素值
	ypan_step	垂直方向上每步进的像素值
	line_length	每行字节数
GFBG_ATTR（属性）	fmt	图形层格式
	alpha0	argb1555 格式 alpha0 值
	alpha1	argb1555 格式 alpha1 值
	global_alpha	全局 alpha 使能状态
	pix_alpha	像素 alpha 使能状态
	glo_alpha_val	全局 alpha 值
	key_enable	colorkey 使能状态
	key	colorkey 值

类别	属性	描述
	pre_mul	预乘使能状态
	pos.x	显示原点 x 坐标值
	pos.y	显示原点 y 坐标值
	luma	图形层亮度
	contrast	图形层对比度
	hue	图形层色度
	saturation	图形层饱和度
	csc_type	图形层 csc 类型
	bg_color	图形层背景色
REFRESH (刷新状态信息)	phy_addr	当前送帧地址
	pre_phy_addr	上次送显地址
	phy_addr_off	当前显示偏移位置
	canvas_size.w	图形层画布宽
	canvas_size.h	图形层画布高
	disp_size.w	图形层宽
	disp_size.h	图形层高
	screen_size.w	屏幕显示宽
	screen_size.h	屏幕显示高
	stride	图形跨度
	per_pix	像素位宽
	xoffset	当前 x 偏移量
	xoffset	当前 y 偏移量
MEM_INFO (内存信息)	buf_phy_addr	内存起始物理地址
	buf_vir_addr	内存起始虚拟地址

类别	属性	描述
	buf_mem_len	内存总长度
	cmap_phy_addr	clut 表起始物理地址
	cmap_vir_addr	clut 表起始虚拟地址
	cmap_mem_len	clut 表总长度
DBG_INFO (调试相关)	force_pre	设定预乘状态
	force_pos	设定显示原点
	force_pat	设定 pattern
	isr_all_cnt	中断总次数
	isr_cb_cnt	当前中断计数，1s 清空
	isr_cb_rate	中断频率
	isr_vo_cnt	vo 中断计数
	isr_max_time	中断最大耗时
	isr_cur_time	最近一次中断耗时
	ref_rate	图新层刷新频率
	ref_cnt	刷新计数，1s 清空
	put_rate	送显频率
	put_cnt	送先计数，1s 清空
	frame_time	最近一次送显时间
	start_time	用于中断调用计时 1s 起始时间
	delay_time	送显到实际刷新的延时时间
CAPABILITY（能力集信息)	key_rgb	Colorkey rgb 支持能力
	key_alpha	Colorkey alpha 支持能力
	global_alpha	全局 alpha 支持能力
	cmap	Clut 读表模式支持能力

类别	属性	描述
	has_cmap_reg	Clut 读表模式硬件能力
	max_width	最大支持宽度
	max_height	最大支持高度
	min_width	最小支持宽度
	min_width	最小支持高度
	pre_mul	预乘支持能力

6.3 图形层调试命令

说明

以下调试命令设置的属性范围需满足“本文档 3.4 章节”各个数据类型要求。

获取 GFBG 调试命令说明

```
echo help > /proc/umap/gfbg*
```

设置 GFBG 为显示状态

```
echo show > /proc/umap/gfbg*
```

设置 GFBG 为隐藏状态

```
echo hide > /proc/umap/gfbg*
```

截取 GFBG 图片 path=*标识图片保存路径

```
echo capture=[0,1] path=* > /proc/umap/gfbg*
```

设置 colorbar 使能状态

```
echo colorbar=[0,1] > /proc/umap/gfbg*
```

设置像素 alpha 使能状态

```
echo alpha_pix_en=[0,1] > /proc/umap/gfbg*
```

设置全局 alpha 使能状态

```
echo alpha_glo_en=[0,1] > /proc/umap/gfbg*
```

设置全局 alpha 值

```
echo alpha_glo_val=[0,255] > /proc/umap/gfbg*
```

设置 argb1555 格式 alpha0 值

```
echo alpha_alpha0=[0,0xff] > /proc/umap/gfbg*
```

设置 argb1555 格式 alpha1 值

```
echo alpha_alpha1=[0,0xff] > /proc/umap/gfbg*
```

设置 colorkey 使能状态

```
echo colorkey_en=[0,1] > /proc/umap/gfbg*
```

设置 colorkey 值

```
echo colorkey_key=[0,0xffffff] > /proc/umap/gfbg*
```

设置预乘使能状态

```
echo premul=[0,1] > /proc/umap/gfbg*
```

设置显示原点 x 坐标，disp.w 为 VO 设备当前的显示分辨率的宽

```
echo position_x=[0, disp.w - 32] > /proc/umap/gfbg*
```

设置显示原点 y 坐标，disp.h 为 VO 设备当前的显示分辨率的高

```
echo position_y=[0, disp.h - 32] > /proc/umap/gfbg*
```