



Tina Linux Display 开发指南

版本号: 1.1
发布日期: 2021.04.02

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.07.05	AWA1422	1. 初始版本
1.1	2021.04.02	AWA1422	1. 更新框架图



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 模块介绍	2
2.1 模块功能介绍	2
2.2 相关术语介绍	3
2.2.1 硬件术语介绍	3
2.2.2 软件术语介绍	3
2.3 模块配置介绍	4
2.3.1 kernel_menuconfig 配置说明	4
2.4 源码结构介绍	5
2.5 驱动框架介绍	6
3 模块接口概述	7
4 显示输出设备操作说明	11
5 接口参数更改说明	12
6 输出设备介绍	13
6.1 屏	13
6.2 HDMI	13
6.3 同显	13
7 IOCTL 接口描述	14
7.1 Global Interface	14
7.1.1 DISP_SHADOW_PROTECT	14
7.1.2 DISP_SET_BKCOLOR	15
7.1.3 DISP_GET_BKCOLOR	16
7.1.4 DISP_GET_SCN_WIDTH	16
7.1.5 DISP_GET_SCN_HEIGHT	17
7.1.6 DISP_GET_OUTPUT_TYPE	18
7.1.7 DISP_GET_OUTPUT	19
7.1.8 DISP_VSYNC_EVENT_EN	20
7.1.9 DISP_DEVICE_SWITCH	20
7.1.10 DISP_DEVICE_SET_CONFIG	21
7.1.11 DISP_DEVICE_GET_CONFIG	22
7.2 Layer Interface	23
7.2.1 DISP_LAYER_SET_CONFIG	23
7.2.2 DISP_LAYER_GET_CONFIG	24
7.2.3 DISP_LAYER_SET_CONFIG2	25

7.2.4	DISP_LAYER_GET_CONFIG2	27
7.3	Capture interface	27
7.3.1	DISP_CAPTURE_START	27
7.3.2	DISP_CAPTURE_COMMIT	28
7.3.3	DISP_CAPTURE_STOP	29
7.3.4	DISP_CAPTURE_QUERY	30
7.4	LCD Interface	31
7.4.1	DISP_LCD_SET_BRIGHTNESS	31
7.4.2	DISP_LCD_GET_BRIGHTNESS	32
7.5	Enhance interface	32
7.5.1	DISP_ENHANCE_ENABLE	32
7.5.2	DISP_ENHANCE_DISABLE	33
7.5.3	DISP_ENHANCE_DEMO_ENABLE	34
7.5.4	DISP_ENHANCE_DEMO_DISABLE	35
7.6	Smart backlight	36
7.6.1	DISP_SMBL_ENABLE	36
7.6.2	DISP_SMBL_DISABLE	36
7.6.3	DISP_SMBL_SET_WINDOW	37
7.7	Hdmi interface	38
7.7.1	DISP_HDMI_SUPPORT_MODE	38
7.7.2	DISP_HDMI_GET_HPD_STATUS	39
8	sysfs 接口描述	41
8.1	enhance	41
8.1.1	enhance_mode	41
8.1.2	enhance_bright/contrast/saturation/edge/detail/denoise	42
8.2	hdmi edid	44
8.2.1	edid	44
8.2.2	hpd	44
8.2.3	hdcp_enable	45
9	Data Structure	47
9.1	disp_fb_info	47
9.2	disp_layer_info	48
9.3	disp_layer_config	49
9.4	disp_color_info	49
9.5	disp_rect	50
9.6	disp_rect64	51
9.7	disp_position	51
9.8	disp_rectsz	52
9.9	disp_pixel_format	52
9.10	disp_buffer_flags	53
9.11	disp_3d_out_mode	54

9.12 disp_color_space	55
9.13 disp_output_type	56
9.14 disp_tv_mode	57
9.15 disp_output	57
9.16 disp_layer_mode	58
9.17 disp_scan_flags	58
10 调试	60
10.1 查看显示模块的状态	60
10.2 截屏	61
10.3 colorbar	61
10.4 显示模块 debugfs 接口	62
10.4.1 总述	62
10.4.2 切换显示输出设备	62
10.4.3 开关显示输出设备	62
10.4.4 电源管理 (suspend/resume) 接口	63
10.4.5 调节 lcd 屏幕背光	63
10.4.6 vsync 消息开关	63
10.4.7 查看 enhance 的状态	64
10.4.8 查看智能背光的状态	64
10.5 常见问题	64
10.5.1 黑屏 (无背光)	64
10.5.2 黑屏 (有背光)	65
10.5.3 绿屏	65
10.5.4 界面卡住	66
10.5.5 局部界面花屏	66
10.5.6 快速切换界面花屏	66

插 图

2-1 模块框图	2
2-2 disp 配置	4
2-3 驱动框图	6
3-1 size 和 crop 示意图	8
3-2 crop 和 screen win 示意图	8
3-3 alpha 叠加模式	9



1 概述

1.1 编写目的

让显示应用开发人员了解显示驱动的接口及使用流程，快速上手，进行开发；让新人接手工作时能快速地了解驱动接口，进行调试排查问题。

1.2 适用范围

sunxi 平台 DE1.0/DE2.0。

1.3 相关人员

与显示相关的应用开发人员，及与显示相关的其他模块的开发人员，以及新人。

2 模块介绍

2.1 模块功能介绍

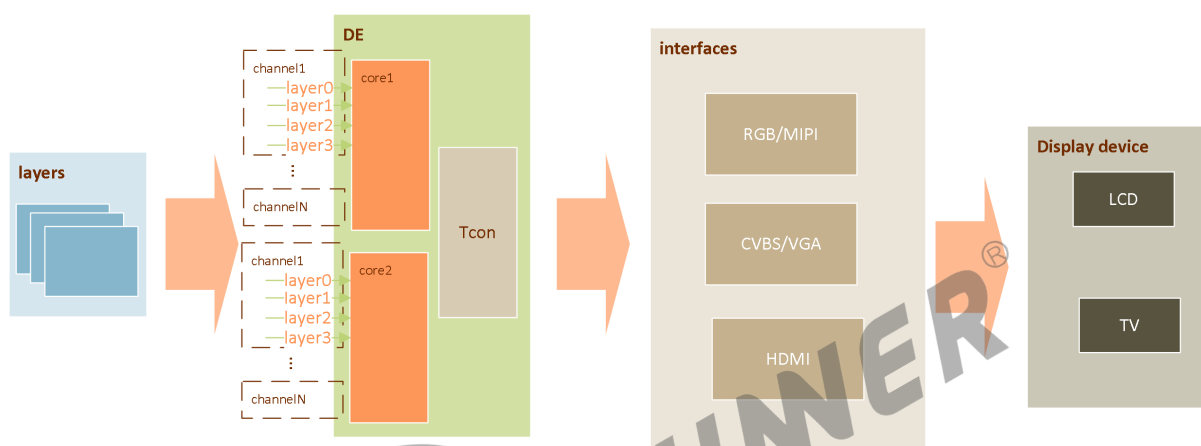


图 2-1: 模块框图

本模块框图如上，由显示引擎（DE）和各类型控制器（tcon）组成。输入图层（layers）在 DE 中进行显示相关处理后，通过一种或多种接口输出到显示设备上显示，以达到将众多应用渲染的图层合成后在显示器呈现给用户观看的作用。DE 有 2 个独立单元（可以简称 de0、de1），可以分别接受用户输入的图层进行合成，输出到不同的显示器，以实现双显。DE 的每个独立的单元有 1-4 个通道（典型地，de0 有 4 个，de1 有 2 个），每个通道可以同时处理接受 4 个格式相同的图层。sunxi 平台有视频通道和 UI 通道之分。视频通道功能强大，可以支持 YUV 格式和 RGB 图层。UI 通道只支持 RGB 图层。

简单来说，显示模块的主要功能如下：

- 支持 lcd(hv/lvds/cpu/dsi) 输出。
- 支持双显输出。
- 支持多图层叠加混合处理。
- 支持多种显示效果处理（alpha, colorkey, 图像增强，亮度/对比度/饱和度/色度调整）。
- 支持智能背光调节。
- 支持多种图像数据格式输入（arg,yuv）。
- 支持图像缩放处理。
- 支持截屏。
- 支持图像转换。

2.2 相关术语介绍

2.2.1 硬件术语介绍

表 2-1: 硬件术语介绍表

术语	解释
de	display engine, 显示引擎, 负责将输入的多图层进行叠加、混合、缩放等处理的硬件模块
channel	一个硬件通道, 包含若干图层处理单元, 可以同时处理若干 (典型 4 个) 格式相同的图层
layer	一个图层处理单元, 可以处理一张输入图像, 按支持的图像格式分 video 和 ui 类型
capture	截屏, 将 de 的输出保存到本地文件
alpha	透明度, 在混合时决定对应图像的透明度
transform	图像变换, 如平移、旋转等
overlay	图像叠加, 按顺序将图像叠加一起的效果。z 序大的靠近观察者, 会把 z 序小的挡住
blending	图像混合, 按 alpha 比例将图像合成一起的效果
enhance	图像增强, 有目的地处理图像数据以达到改善图像效果的过程或方法

2.2.2 软件术语介绍

表 2-2: 软件术语介绍表

术语	解释
fb	帧缓冲 (framebuffer), Linux 为显示设备提供的一个接口, 把显存抽象成的一种设备。有时也指一块显存
al	抽象层, 驱动中将底层硬件抽象成固定业务逻辑的软件层
lowlevel	底层, 直接操作硬件寄存器的软件层

2.3 模块配置介绍

2.3.1 kernel_menuconfig 配置说明

```
make kernel_menuconfig
```

具体配置目录为：

```
Device Drivers --->
Graphics support --->
  <*> Support for frame buffer devices --->
    Video support for sunxi --->
      <*> DISP Driver Support(sunxi-disp2)
```

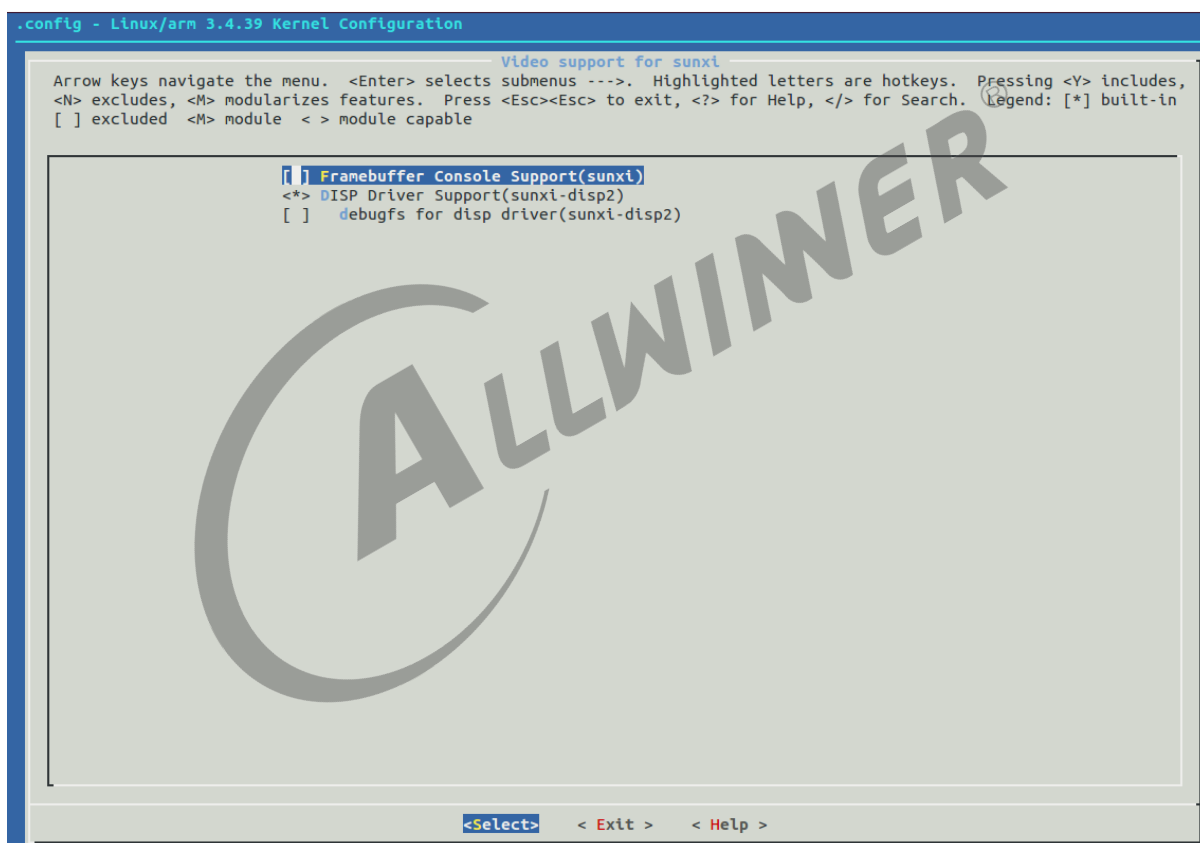


图 2-2: disp 配置

其中：

- DISP Driver Support(sunxi-disp2)

DE 驱动请选上。

- debugfs support for disp driver(sunxi-disp2)

调试节点，建议选上，方便调试。

- composer support for disp driver(sunxi-disp2)

disp2 的 fence 处理。linux 系统可以不选择。

2.4 源码结构介绍

源码结构如下：

```
├─drivers
│   └─video
│       └─fbdev
│           └─sunxi
│               └─disp2/
│                   └─disp
│                       ├──dev_disp.c      --display driver 层
│                       ├──dev_fb.c        --framebuffer driver 层
│                       ├──de
│                       │   ├──disp_lcd.c   --bsp层
│                       │   ├──disp_al.c    --disp_manager.c ..
│                       │   ├──disp_al.c    --al层
│                       │   └─lowlevel_sunxi/
│                       │       ├──de_lcd.c ...
│                       │       ├──disp_sys_int.c --OSAL 层,与操作系统相关层
│                       └─lcd/ lcd driver
│                           ├──lcd_src_interface.c --与display 驱动的接口
│                           └─default_panel.c... --平台已经支持的屏驱动
└─include
    ├──video video header dir
    └─sunxi_display2.h display header file
```

2.5 驱动框架介绍

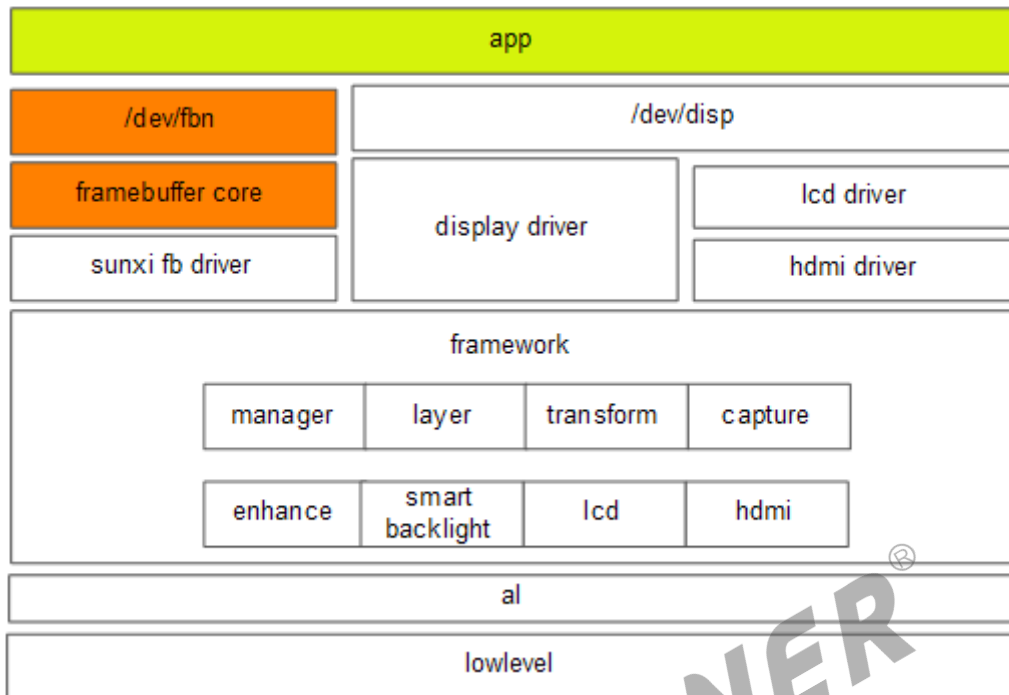


图 2-3: 驱动框图

显示驱动可划分为三个层面，驱动层，框架层及底层。底层与图形硬件相接，主要负责将上层配置的功能参数转换成硬件所需要的参数，并配置到相应寄存器中。

显示框架层对底层进行抽象封装成一个个的功能模块。驱动层对外封装功能接口，通过内核向用户空间提供相应的设备结点及统一的接口。

在驱动层，分为四个驱动，分别是 framebuffer 驱动，disp 驱动，lcd 驱动，hdmi 驱动。Framebuffer 驱动与 framebuffer core 对接，实现 linux 标准的 framebuffre 接口。

Disp 驱动是整个显示驱动中的核心驱动模块，所有的接口都由 disp 驱动来提供，包括 lcd 的接口。

3 模块接口概述

模块使用主要通过 ioctl 实现，对应的驱动节点是/dev/disp。

具体定义请仔细阅读头文件上面的注释，kernel/linux-4.9/include/video/sunxi_display2.h。

对于显示模块来说，把图层参数设置到驱动，让显示器显示为最重要。sunxi 平台的 DE 接受用户设置图层参数，通过 disp,channel,layer_id 三个索引确定需要设置的显示位置 (disp:0/1, channel: 0/1/2/3, layer_id:0/1/2/3)，其中 disp 表示显示器索引，channel 表示通道索引，layer_id 表示通道内的图层索引。

下面着重地把图层的参数从头文件中拿出来介绍。

```
struct disp_fb_info2 {
    int                fd;
    struct disp_rectsz size[3];
    unsigned int       align[3];
    enum disp_pixel_format format;
    enum disp_color_space color_space;
    int                trd_right_fd;
    bool               pre_multiply;
    struct disp_rect64 crop;
    enum disp_buffer_flags flags;
    enum disp_scan_flags scan;
    enum disp_eotf      eotf;
    int                 depth;
    unsigned int         fbd_en;
    int                  metadata_fd;
    unsigned int          metadata_size;
    unsigned int          metadata_flag;
};
```

- fd

显存的文件句柄。

- size 与 crop

Size 表示 buffer 的完整尺寸，crop 则表示 buffer 中需要显示裁减区。如下图所示，完整的图像以 size 标识，而矩形框住的部分为裁减区，以 crop 标识，在屏幕上只能看到 crop 标识的部分，其余部分是隐藏的，不能在屏幕上显示出来的。

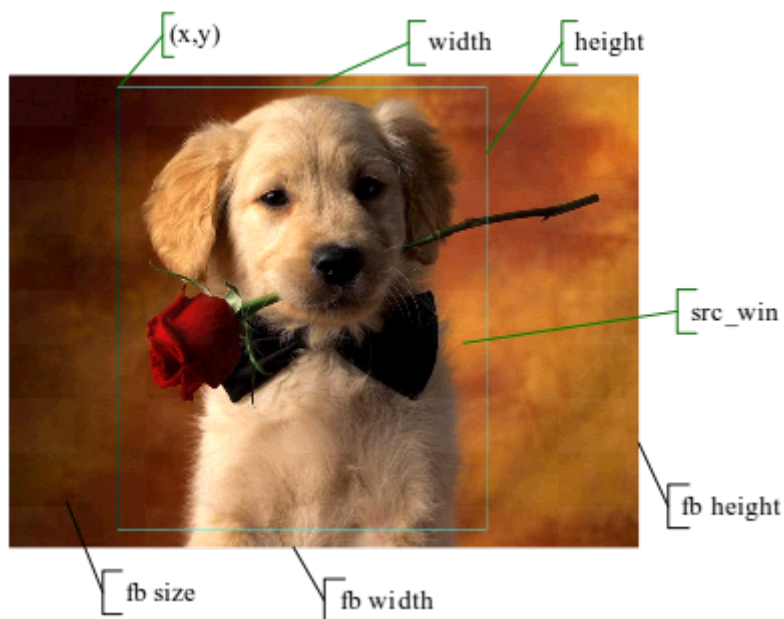


图 3-1: size 和 crop 示意图

- crop 和 screen_win

crop 上面已经介绍过, Screen_win 为 crop 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话, crop 和 screen_win 的 width,height 是相等的, 如果需要缩放, crop 和 screen_win 的 width,height 可以不相等。

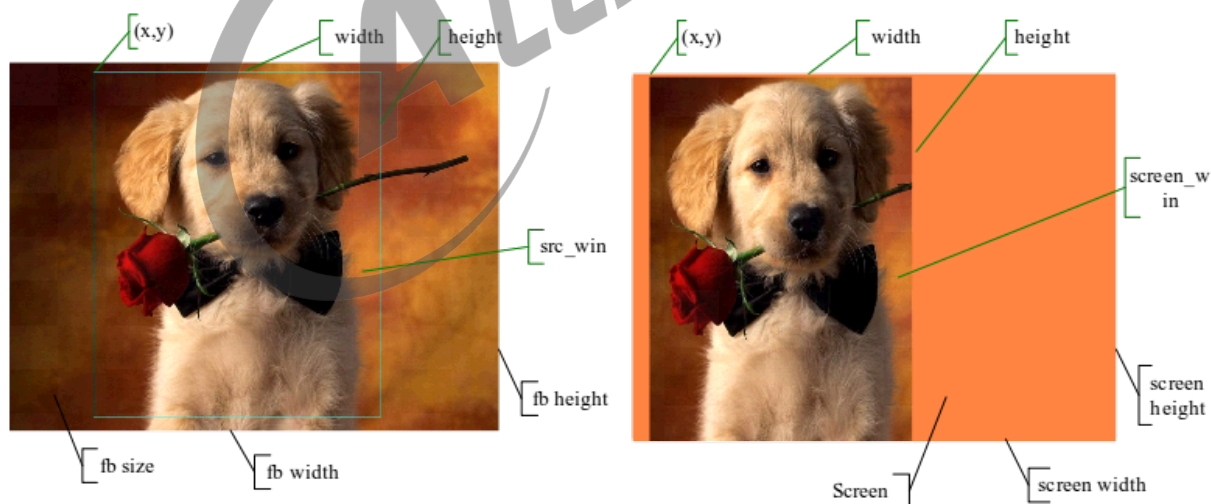


图 3-2: crop 和 screen win 示意图

- alpha

Alpha 模式有三种:

1. global alpha: 全局 alpha, 也叫面 alpha, 即整个图层共用一个 alpha, 统一的透明度。

2. pixel alpha: 点 alpha, 即每个像素都有自己单独的 alpha, 可以实现部分区域全透, 部分区域半透, 部分区域不透的效果。
3. global_pixel alpha: 可以说是以上两种效果的叠加, 在实现 pixel alpha 的效果的同时, 还可以做淡入淡出的效果。

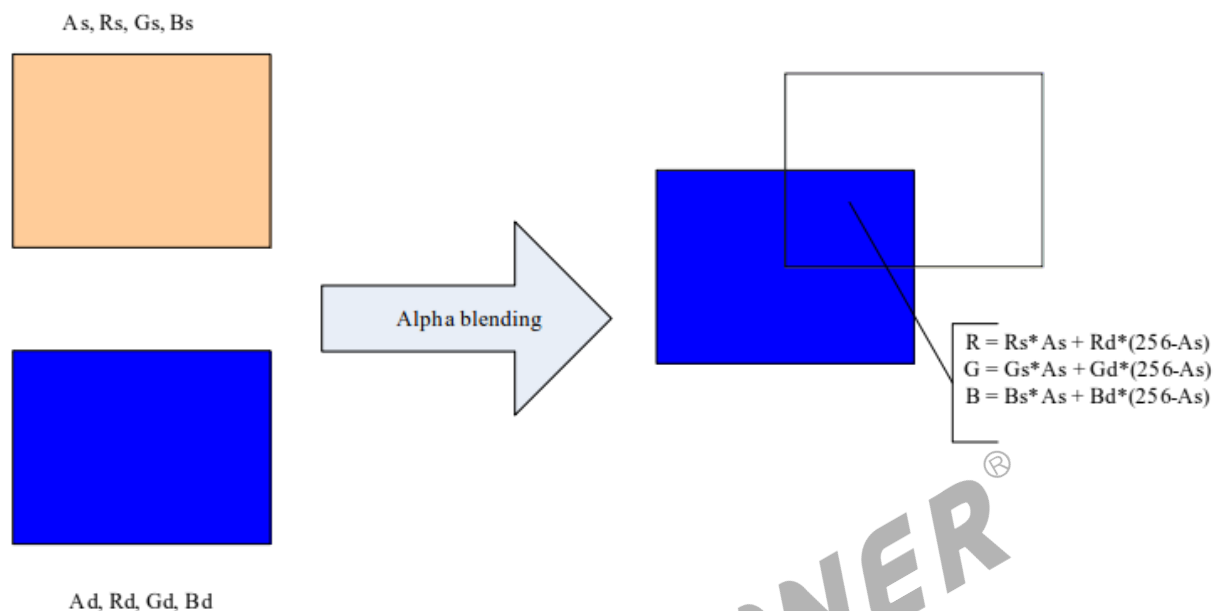


图 3-3: alpha 叠加模式

- align

显存的对齐字节数。

- format

输入图层的格式。Ui 通道支持的格式：

```
DISP_FORMAT_ARGB_8888
DISP_FORMAT_ABGR_8888
DISP_FORMAT_RGBA_8888
DISP_FORMAT_BGRA_8888
DISP_FORMAT_XRGB_8888
DISP_FORMAT_XBGR_8888
DISP_FORMAT_RGBX_8888
DISP_FORMAT_BGRX_8888
DISP_FORMAT_RGB_888
DISP_FORMAT_BGR_888
DISP_FORMAT_RGB_565
DISP_FORMAT_BGR_565
DISP_FORMAT_ARGB_4444
DISP_FORMAT_ABGR_4444
DISP_FORMAT_RGBA_4444
DISP_FORMAT_BGRA_4444
DISP_FORMAT_ARGB_1555
```

```
DISP_FORMAT_ABGR_1555
DISP_FORMAT_RGBA_5551
DISP_FORMAT_BGRA_5551
DISP_FORMAT_A2R10G10B10
DISP_FORMAT_A2B10G10R10
DISP_FORMAT_R10G10B10A2
DISP_FORMAT_B10G10R10A2
```

Video 通道支持的格式：

```
DISP_FORMAT_ARGB_8888
DISP_FORMAT_ABGR_8888
DISP_FORMAT_RGBA_8888
DISP_FORMAT_BGRA_8888
DISP_FORMAT_XRGB_8888
DISP_FORMAT_XBGR_8888
DISP_FORMAT_RGBX_8888
DISP_FORMAT_BGRX_8888
DISP_FORMAT_RGB_888
DISP_FORMAT_BGR_888
DISP_FORMAT_RGB_565
DISP_FORMAT_BGR_565
DISP_FORMAT_ARGB_4444
DISP_FORMAT_ABGR_4444
DISP_FORMAT_RGBA_4444
DISP_FORMAT_BGRA_4444
DISP_FORMAT_ARGB_1555
DISP_FORMAT_ABGR_1555
DISP_FORMAT_RGBA_5551
DISP_FORMAT_BGRA_5551
DISP_FORMAT_YUV444_I_AYUV
DISP_FORMAT_YUV444_I_VUYA
DISP_FORMAT_YUV422_I_YVYU
DISP_FORMAT_YUV422_I_YUYV
DISP_FORMAT_YUV422_I_UYVY
DISP_FORMAT_YUV422_I_VYUY
DISP_FORMAT_YUV444_P
DISP_FORMAT_YUV422_P
DISP_FORMAT_YUV420_P
DISP_FORMAT_YUV411_P
DISP_FORMAT_YUV422_SP_UVUV
DISP_FORMAT_YUV422_SP_VUVU
DISP_FORMAT_YUV420_SP_UVUV
DISP_FORMAT_YUV420_SP_VUVU
DISP_FORMAT_YUV411_SP_UVUV
DISP_FORMAT_YUV411_SP_VUVU
DISP_FORMAT_YUV444_I_AYUV_10BIT
DISP_FORMAT_YUV444_I_VUYA_10BIT
```

所有图层都支持缩放。对图层的操作如下所示：

1. 设置图层参数并使能，接口为 DISP_LAYER_SET_CONFIG，图像格式，buffer size，buffer 地址，alpha 模式，enable，图像帧 id 号等参数。
2. 关闭图层，依然通过 DISP_LAYER_SET_CONFIG，将 enable 参数设置为 0 关闭。

4 显示输出设备操作说明

Disp2 支持多种的显示输出设备，LCD、TV、HDMI。开启显示输出设备有几种方式，第一种是在 sys_config 或 dts 中配置 [disp] 的初始化参数，显示模块在加载时将会根据配置初始化选择的显示输出设备；第二种是在 kernel 启动后，调用驱动模块的 ioctl 接口去开启或关闭指定的输出设备，以下是操作的说明：

- 开启或切换到某个具体的显示输出设备，ioctl(DISP_DEVICE_SWITCH...)，参数设置为特定的输出设备类型，DISP_OUTPUT_TYPE_LCD/TV/HDMI。
- 关闭某个设备，ioctl(DISP_DEVICE_SWITCH...)，参数设置为 DISP_OUTPUT_TYPE_NONE。



5 接口参数更改说明

sunxi 平台支持 disp1 和 disp2。

表 5-1: disp1 与 disp2 区别

项目\平台	disp2	disp1
图层标识	以 disp, channel, layer_id 唯一标识	以 disp, layer_id 唯一标识
图层开关	将开关当成图层参数设置 DISP_LAYER_SET_CONFIG 中	独立图层开关接口
图层 size	每个分量都需要设置 1 个 size	一个 buffer 只有 1 个 size
图层 align	针对每个分量需要设置其 align, 单位为 byte。	无
图层 Crop	为 64 位定点小数, 高 32 位为整数, 低 32 位为小数	为 32 位参数, 不支持小数
YUV MB 格式支持	不再支持	支持
PALETTE 格式支持	不再支持	支持
单色模式 (无 buffer)	支持	不支持
Pipe 选择	Pipe 对用户透明, 用户无需选择, 只需要配置 channel	用户设置
zorder	用户设置, 保证 zorder 不重复, 从 0 到 N-1	用户不能设置
设置图层信息接口	一次可设置多个图层的信息, 增加一个图层信息数目参数	一次设置 1 个图层信息

6 输出设备介绍

平台支持屏以及 HDMI 输出，及二者同时显示。

6.1 屏

屏的接口很多，平台支持 RGB/CPU/LVDS/DSI 接口。

6.2 HDMI

HDMI 全名是：High-Definition Multimedia Interface。可以提供 DVD, audio device, set-top boxes, television sets, and other video displays 之间的高清互联。可以承载音，视频数据，以及其他的控制，数据信息。支持热插拔，内容保护，模式是否支持的查询。

6.3 同显

驱动支持双路显示。屏（主）+ HDMI（辅）。

同显或异显，差别只在于显示内容，如果显示内容一样，则为同显；反之，则为异显。

1. 如果是 android 系统，4.2 版本以上版本，原生框架已经支持多显（同显，异显，虚拟显示设备），实现同显则比较简单，在 android hal 与上层对接好即可。
2. 如果是 android 4.1 以下版本，同显需要自行实现，参考做法为主屏内容由 android 原生提供，辅屏需要 android hal 在合适的时机（比如 HDMI 插入时）打开辅屏，并且将主屏的内容（存放于 FB0 中），拷贝至辅屏的显示后端 buffer 中，然后将辅屏的后端 buffer 切换到前端 buffer。注意问题为，两路显示的显示 buffer 的同步，如果同步不好，会产生图像撕裂，错位的现象。
3. 如果是 Linux 系统，做法与上一个做法类似。

7 IOCTL 接口描述

sunxi 平台下显示驱动给用户提供了众多功能接口，可对图层、LCD、hdmi 等显示资源进行操作。

7.1 Global Interface

7.1.1 DISP_SHADOW_PROTECT

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_SHADOW_PROTECT
arg	arg[0] 为显示通道 0/1; arg[1] 为 protect 参数, 1 表示 protect, 0: 表示 not protect

- 返回值

如果成功，返回 DIS_SUCCESS，否则，返回失败号。

- 描述

DISP_SHADOW_PROTECT (1) 与 DISP_SHADOW_PROTECT (0) 配对使用，在 protect 期间，所有的请求当成一个命令序列缓冲起来，等到调用 DISP_SHADOW_PROTECT (0) 后将一起执行。

- 示例

```
//启动cache, disphd为显示驱动句柄
unsigned int arg[3];
arg[0] = 0;//disp0
arg[1] = 1;//protect
ioctl(disphd, DISP_SHADOW_PROTECT, (void*)arg);
//do something other
arg[1] = 0;//unprotect
ioctl(disphd, DISP_SHADOW_PROTECT, (void*)arg);
```

7.1.2 DISP_SET_BKCOLOR

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_SET_BKCOLOR
arg	arg[0] 为显示通道 0/1; arg[1] 为 backcolor 信息, 指向 disp_color 数据结构指针

- 返回值

如果成功, 返回 DIS_SUCCESS, 否则, 返回失败号。

- 描述

该函数用于设置显示背景色。

- 示例

```
//设置显示背景色, disphd为显示驱动句柄, sel为屏0/1
disp_color bk;
unsigned int arg[3];

bk.red      = 0xff;
bk.green    = 0x00;
bk.blue     = 0x00;
arg[0]      = 0;
arg[1]      = (unsigned int)&bk;
ioctl(disphd, DISP_SET_BKCOLOR, (void*)arg);
```

7.1.3 DISP_GET_BKCOLOR

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_GET_BKCOLOR
arg	arg[0] 为显示通道 0/1; arg[1] 为 backcolor 信息, 指向 disp_color 数据结构指针

- 返回值

如果成功, 返回 DIS_SUCCESS, 否则, 返回失败号。

- 描述

该函数用于获取显示背景色。

- 示例

```
//获取显示背景色, disphd为显示驱动句柄, sel为屏0/1
disp_color bk;
unsigned int arg[3];

arg[0]      = 0;
arg[1]      = (unsigned int)&bk;
ioctl(disphd, DISP_GET_BKCOLOR, (void*)arg);
```

7.1.4 DISP_GET_SCN_WIDTH

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_GET_SCN_WIDTH
arg	arg[0] 显示通道 0/1

- 返回值

如果成功，返回当前屏幕水平分辨率，否则，返回失败号。

- 描述

该函数用于获取当前屏幕水平分辨率。

- 示例

```
//获取屏幕水平分辨率
unsigned int screen_width;
unsigned int arg[3];

arg[0] = 0;
screen_width = ioctl(disphd, DISP_GET_SCN_WIDTH, (void*)arg);
```

7.1.5 DISP_GET_SCN_HEIGHT

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_GET_SCN_HEIGHT
arg	arg[0] 显示通道 0/1

- 返回值

如果成功，返回当前屏幕垂直分辨率，否则，返回失败号。

- 描述

该函数用于获取当前屏幕垂直分辨率。

- 示例

```
//获取屏幕垂直分辨率
unsigned int screen_height;
unsigned int arg[3];

arg[0] = 0;
screen_height = ioctl(disphd, DISP_GET_SCN_HEIGHT, (void*)arg);
```

7.1.6 DISP_GET_OUTPUT_TYPE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_GET_OUTPUT_TYPE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，返回当前显示输出类型，否则，返回失败号。

- 描述

该函数用于获取当前显示输出类型 (LCD,TV,HDMI,VGA,NONE)。

- 示例

```
//获取当前显示输出类型
disp_output_type output_type;
unsigned int arg[3];

arg[0] = 0;
output_type = (disp_output_type)ioctl(disphd, DISP_GET_OUTPUT_TYPE, (void*)arg);
```

7.1.7 DISP_GET_OUTPUT

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_GET_OUTPUT
arg	arg[0] 为显示通道 0/1；arg[1] 为指向 disp_output 结构体的指针，用于保存返回值

- 返回值

如果成功，返回 0，否则，返回失败号。

- 描述

该函数用于获取当前显示输出类型及模式 (LCD,TV,HDMI,VGA,NONE)。

- 示例

```
//获取当前显示输出类型
unsigned int arg[3];
disp_output output;
disp_output_type type;
disp_tv_mode mode;

arg[0] = 0;
arg[1] = (unsigned long)&output;
ioctl(disphd, DISP_GET_OUTPUT, (void*)arg);
type = (disp_output_type)output.type;
mode = (disp_tv_mode)output.mode;
```

7.1.8 DISP_VSYNC_EVENT_EN

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_VSYNC_EVENT_EN
arg	arg[0] 为显示通道 0/1; arg[1] 为 enable 参数, 0: disable, 1:enable

- 返回值

如果成功, 返回 DIS_SUCCESS.

否则, 返回失败号。

- 描述

该函数开启/关闭 vsync 消息发送功能。

- 示例

```
//开启/关闭vsync消息发送功能, disphd为显示驱动句柄, sel为屏0/1
unsigned int arg[3];

arg[0] = 0;
arg[1] = 1;
ioctl(disphd, DISP_VSYNC_EVENT_EN, (void*)arg);
```

7.1.9 DISP_DEVICE_SWITCH

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
----	----

hdle	显示驱动句柄
------	--------

cmd	DISP_DEVICE_SWITCH
-----	--------------------

arg	arg[0] 为显示通道 0/1；arg[1] 为输出类型；arg[2] 为输出模式，在输出类型不为 LCD 时有效
-----	--

- 返回值

如果成功，返回 DIS_SUCCESS，否则，返回失败号。

- 描述

该函数用于切换输出类型。

- 示例

```
//切换
unsigned int arg[3];
arg[0] = 0;
arg[1] = (unsigned long)DISP_OUTPUT_TYPE_HDMI;
arg[2] = (unsigned long)DISP_TV_MOD_1080P_60HZ;
ioctl(disphd, DISP_DEVICE_SWITCH, (void*)arg);
```

说明：如果传递的 type 是 DISP_OUTPUT_TYPE_NONE，将会关闭当前显示通道的输出。

7.1.10 DISP_DEVICE_SET_CONFIG

- 原型

```
int ioctl(int handle, unsigned int cmd,unsigned int *arg);
```

- 参数

参数	说明
----	----

hdle	显示驱动句柄
------	--------

cmd	DISP_DEVICE_SET_CONFIG
-----	------------------------

arg	arg[0] 为显示通道 0/1；arg[1] 为指向 disp_device_config 的指针
-----	--

- 返回值

如果成功，返回 DIS_SUCCESS，否则，返回失败号。

- 描述

该函数用于切换输出类型并设置输出设备的属性参数。

- 示例

```
//切换输出类型并设置输出设备的属性参数
unsigned long arg[3];
struct disp_device_config config;
config.type = DISP_OUTPUT_TYPE_LCD;
config.mode = 0;
config.format = DISP_CSC_TYPE_RGB;
config.bits = DISP_DATA_8BITS;
config.eotf = DISP_EOTF_GAMMA22;
config.cs = DISP_BT709;
arg[0] = 0;
arg[1] = (unsigned long)&config;
ioctl(dispfd, DISP_DEVICE_SET_CONFIG, (void*)arg);
//说明：如果传递的type是DISP_OUTPUT_TYPE_NONE，将会关闭当前显示通道的输出。
```

7.1.11 DISP_DEVICE_GET_CONFIG

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_DEVICE_GET_CONFIG
arg	arg[0] 为显示通道 0/1；arg[1] 为指向 disp_device_config 的指针

- 返回值

如果成功，返回 DIS_SUCCESS，否则，返回失败号。

- 描述

该函数用于获取当前输出类型及相关的属性参数。

- 示例

```
//获取当前输出类型及相关的属性参数
unsigned long arg[3];
struct disp_device_config config;
arg[0] = 0;
arg[1] = (unsigned long)&config;
ioctl(dispfd, DISP_DEVICE_GET_CONFIG, (void*)arg);
//说明：如果返回的type是DISP_OUTPUT_TYPE_NONE，表示当前输出显示通道为关闭状态
```

7.2 Layer Interface

7.2.1 DISP_LAYER_SET_CONFIG

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_CMD_SET_LAYER_CONFIG
arg	arg[0] 为显示通道 0/1；arg[1] 为图层配置参数指针；arg[2] 为需要配置的图层数目

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于设置多个图层信息。

- 示例

```

struct
{
    disp_layer_info info,
    bool enable;
    unsigned int channel,
    unsigned int layer_id,
} disp_layer_config;

//设置图层参数, disphd为显示驱动句柄
unsigned int arg[3];
disp_layer_config config;
unsigned int width = 1280;
unsigned int height = 800;
unsigned int ret = 0;

memset(&info, 0, sizeof(disp_layer_info));
config.channel = 0; //channel 0
config.layer_id = 0; //layer 0 at channel 0
config.info.enable = 1;
config.info.mode = LAYER_MODE_BUFFER;
config.info.fb.addr[0] = (__u32)mem_in; //FB地址
config.info.fb.size.width = width;
config.info.fb.format = DISP_FORMAT_ARGB_8888; //DISP_FORMAT_YUV420_P
config.info.fb.crop.x = 0;
config.info.fb.crop.y = 0;
config.info.fb.crop.width = ((unsigned long)width) << 32; //定点小数。 高32bit为整数, 低32bit为小数
config.info.fb.crop.height = ((unsigned long)height) << 32; //定点小数。 高32bit为整数, 低32bit为小数
config.info.fb.flags = DISP_BF_NORMAL;
config.info.fb.scan = DISP_SCAN_PROGRESSIVE;
config.info.alpha_mode = 1; //global alpha
config.info.alpha_value = 0xff;
config.info.screen_win.x = 0;
config.info.screen_win.y = 0;
config.info.screen_win.width = width;
config.info.screen_win.height = height;
config.info.id = 0;

arg[0] = 0; //screen 0
arg[1] = (unsigned int)&config;
arg[2] = 1; //one layer
ret = ioctl(disphd, DISP_CMD_LAYER_SET_CONFIG, (void*)arg);

```

7.2.2 DISP_LAYER_GET_CONFIG

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
----	----

hdle	显示驱动句柄
------	--------

cmd	DISP_LAYER_GET_CONFIG
-----	-----------------------

arg	arg[0] 为显示通道 0/1；arg[1] 为图层配置参数指针；arg[2] 为需要获取配置的图层数目
-----	---

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于获取图层参数。

- 示例

```
//获取图层参数，disphd为显示驱动句柄
unsigned int arg[3];
disp_layer_info info;

memset(&info, 0, sizeof(disp_layer_info));
config.channel = 0; //channel 0
config.layer_id = 0; //layer 0 at channel 0

arg[0] = 0; //显示通道0
arg[1] = 0; //图层0
arg[2] = (unsigned int)&info;
ret = ioctl(disphd, DISP_LAYER_GET_CONFIG, (void*)arg);
```

7.2.3 DISP_LAYER_SET_CONFIG2

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
----	----

hdle	显示驱动句柄
------	--------

cmd	DISP_SET_LAYER_CONFIG2
-----	------------------------

arg	arg[0] 为显示通道 0/1；arg[1] 为图层配置参数指针；arg[2] 为需要配置的图层数目
-----	---

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于设置多个图层信息，注意该接口只接受 disp_layer_config2 的信息。

- 示例

```
struct
{
    disp_layer_info info,
    bool enable;
    unsigned int channel,
    unsigned int layer_id,
}disp_layer_config2;
//设置图层参数，dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_layer_config2 config;
unsigned int width = 1280;
unsigned int height = 800;
unsigned int ret = 0;
memset(&config, 0, sizeof(struct disp_layer_config2));
config.channnel = 0;//blending channel
config.layer_id = 0;//layer index in the blending channel
config.info.enable = 1;
config.info.mode = LAYER_MODE_BUFFER;
config.info.fb.addr[0] = (unsigned long long)mem_in; //FB 地址
config.info.fb.size[0].width = width;
config.info.fb.align[0] = 4;//bytes
config.info.fb.format = DISP_FORMAT_ARGB_8888; //DISP_FORMAT_YUV420_P
config.info.fb.crop.x = 0;
config.info.fb.crop.y = 0;
config.info.fb.crop.width = ((unsigned long)width) << 32;//定点小数。高32bit 为整数，低32bit 为
小数
config.info.fb.crop.height= ((unsigned long)height)<<32;//定点小数。高32bit 为整数，低32bit 为
小数
config.info.fb.flags = DISP_BF_NORMAL;
config.info.fb.scan = DISP_SCAN_PROGRESSIVE;
config.info.fb.eotf = DISP_EOTF_SMPTE2084; //HDR
config.info.fb.metadata_buf = (unsigned long long)mem_in2;
config.info.alpha_mode = 2; //global pixel alpha
config.info.alpha_value = 0xff;//global alpha value
config.info.screen_win.x = 0;
config.info.screen_win.y = 0;
config.info.screen_win.width = width;
config.info.screen_win.height= height;
config.info.id = 0;
arg[0] = 0;//screen 0
arg[1] = (unsigned long)&config;
arg[2] = 1; //one layer
ret = ioctl(dispfd, DISP_LAYER_SET_CONFIG2, (void*)arg);
```

7.2.4 DISP_LAYER_GET_CONFIG2

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_LAYER_GET_CONFIG2
arg	arg[0] 为显示通道 0/1; arg[1] 为图层配置参数指针; arg[2] 为需要配置的图层数目

- 返回值

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

- 描述

该函数用于获取图层参数。

- 示例

```
//设置图层参数, dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_layer_config2 config;
memset(&config, 0, sizeof(struct disp_layer_config2));
arg[0] = 0; //disp
arg[1] = (unsigned long)&config;
arg[2] = 1; //layer number
ret = ioctl(dispfd, DISP_GET_LAYER_CONFIG2, (void*)arg);
```

7.3 Capture interface

7.3.1 DISP_CAPTURE_START

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_CAPTURE_START
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于开启截屏功能。

- 示例

```
//启动截屏功能，dispfd 为显示驱动句柄  
arg[0] = 0; //显示通道0  
ioctl(dispfd, DISP_CAPTURE_START, (void*)arg);
```

7.3.2 DISP_CAPTURE_COMMIT

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_CAPTURE_COMMIT
arg	arg[0] 为显示通道 0/1；arg[1] 为指向截屏的信息结构体，详见 disp_capture_info

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于提交截屏的任务，提交一次，则会启动一次截屏操作。

- 示例

```
//提交截屏功能, dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_capture_info info;
arg[0] = 0;
screen_width = ioctl(dispfd, DISP_GET_SCN_WIDTH, (void*)arg);
screen_height = ioctl(dispfd, DISP_GET_SCN_HEIGHT, (void*)arg);
info.window.x = 0;
info.window.y = 0;
info.window.width = screen_width;
info.window.y = screen_height;
info.out_frame.format = DISP_FORMAT_ARGB_8888;
info.out_frame.size[0].width = screen_width;
info.out_frame.size[0].height = screen_height;
info.out_frame.crop.x = 0;
info.out_frame.crop.y = 0;
info.out_frame.crop.width = screen_width;
info.out_frame.crop.height = screen_height;
info.out_frame.addr[0] = fb_address; //buffer address
arg[0] = 0; //显示通道0
arg[1] = (unsigned long)&info;
ioctl(dispfd, DISP_CAPTURE_COMMIT, (void*)arg);
```

7.3.3 DISP_CAPTURE_STOP

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_CAPTURE_STOP
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于关闭截屏功能。

- 示例

```
//停止截屏功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0;//显示通道0
ioctl(dispfd, DISP_CAPTURE_STOP, (void*)arg);
```

7.3.4 DISP_CAPTURE_QUERY

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

命令 DISP_CAPTURE_QUERY 是查询功能。

参数	说明
hdle	显示驱动句柄
cmd	DISP_CAPTURE_QUERY
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数查询刚结束的图像帧是否截屏成功。

- 示例

```
//查询截屏是否成功, dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; //显示通道0
ioctl(dispfd, DISP_CAPTURE_QUERY, (void*)arg);
```

7.4 LCD Interface

7.4.1 DISP_LCD_SET_BRIGHTNESS

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_LCD_SET_BRIGHTNESS
arg	arg[0] 为显示通道 0/1; arg[1] 为背光亮度值, (0~255)

- 返回值

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

- 描述

该函数用于设置 LCD 亮度。

- 示例

```
//设置LCD的背光亮度, dispfd为显示驱动句柄
unsigned int arg[3];
unsigned int bl = 197;

arg[0] = 0; //显示通道0
arg[1] = bl;
```

```
ioctl(disphd, DISP_LCD_SET_BRIGHTNESS, (void*)arg);
```

7.4.2 DISP_LCD_GET_BRIGHTNESS

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_LCD_GET_BRIGHTNESS
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于获取 LCD 亮度。

- 示例

```
//获取LCD的背光亮度的，disphd为显示驱动句柄
unsigned int arg[3];
unsigned int bl;

arg[0] = 0;//显示通道0
bl = ioctl(disphd, DISP_LCD_GET_BRIGHTNESS, (void*)arg);
```

7.5 Enhance interface

7.5.1 DISP_ENHANCE_ENABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_ENHANCE_ENABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于使能图像后处理功能。

- 示例

```
//开启图像后处理功能，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0;//显示通道0
ioctl(disphd, DISP_ENHANCE_ENABLE, (void*)arg);
```

7.5.2 DISP_ENHANCE_DISABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_ENHANCE_DISABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于关闭图像后处理功能。

- 示例

```
//关闭图像后处理功能，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0; //显示通道0
ioctl(disphd, DISP_ENHANCE_DISABLE, (void*)arg);
```

7.5.3 DISP_ENHANCE_DEMO_ENABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_ENHANCE_DEMO_ENABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于开启图像后处理演示模式，开启后，在屏幕会出现左边进行后处理，右边未处理的图像画面，方便对比效果。演示模式需要在后处理功能开启之后才有效。

- 示例

```
//开启图像后处理演示模式，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0;//显示通道0
ioctl(disphd, DISP_ENHANCE_DEMO_ENABLE, (void*)arg);
```

7.5.4 DISP_ENHANCE_DEMO_DISABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_ENHANCE_DEMO_DISABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于关闭图像后处理演示模式。

- 示例

```
//开启图像后处理演示模式，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0;//显示通道0
ioctl(disphd, DISP_ENHANCE_DEMO_ENABLE, (void*)arg);
```

7.6 Smart backlight

7.6.1 DISP_SMBL_ENABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_SMBL_ENABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于使能智能背光功能。

- 示例

```
//开启智能背光功能，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0;//显示通道0
ioctl(disphd, DISP_SMBL_ENABLE, (void*)arg);
```

7.6.2 DISP_SMBL_DISABLE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_SMBL_DISABLE
arg	arg[0] 为显示通道 0/1

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于关闭智能背光功能。

- 示例

```
//关闭智能背光功能，disphd为显示驱动句柄
unsigned int arg[3];

arg[0] = 0;//显示通道0
ioctl(disphd, DISP_SMBL_DISABLE, (void*)arg);
```

7.6.3 DISP_SMBL_SET_WINDOW

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_SMBL_SET_WINDOW
arg	arg[0] 为显示通道 0/1；arg[1] 为指向 struct disp_rect 的指针

- 返回值

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- 描述

该函数用于设置智能背光开启效果的窗口，智能背光在设置的窗口中有效。

- 示例

```
//设置智能背光窗口，disphd为显示驱动句柄
unsigned int arg[3];
unsigned int screen_width, screen_height;
struct disp_rect window;

screen_width = ioctl(disphd, DISP_GET_SCN_WIDTH, (void*)arg);
screen_height = ioctl(disphd, DISP_GET_SCN_HEIGHT, (void*)arg);
window.x = 0;
window.y = 0;
window.width = screen_width / 2;
window.height = screen_height;
arg[0] = 0; //显示通道0
arg[1] = (unsigned long)&window;
ioctl(disphd, DISP_SMBL_SET_WINDOW, (void*)arg);
```

7.7 Hdmi interface

7.7.1 DISP_HDMI_SUPPORT_MODE

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_HDMI_SUPPORT_MODE
arg	arg[0] 为显示通道 0/1；arg[1] 为需要查询的模式，详见 disp_tv_mode

- 返回值

如果支持，则返回 1；如果失败，则返回 0。

- 描述

该函数用于查询指定的 HDMI 模式是否支持。

- 示例

```
//查询指定的HDMI模式是否支持
unsigned int arg[3];

arg[0] = 0;//显示通道0
arg[1] = (unsigned long)DISP_TV_MOD_1080P_60HZ;
ioctl(disphd, DISP_HDMI_SUPPORT_MODE, (void*)arg);
```

7.7.2 DISP_HDMI_GET_HPD_STATUS

- 原型

```
int ioctl(int handle, unsigned int cmd, unsigned int *arg);
```

- 参数

参数	说明
hdle	显示驱动句柄
cmd	DISP_HDMI_GET_HPD_STATUS
arg	arg[0] 为显示通道 0/1

- 返回值

如果 HDMI 插入，则返回 1；如果未插入，则返回 0。

- 描述

该函数用于指定 HDMI 是否处于插入状态。

- 示例

```
//查询HDMI是否处于插入状态
unsigned int arg[3];

arg[0] = 0;//显示通道0
if (ioctl(disphd, DISP_HDMI_GET_HPD_STATUS, (void*)arg) == 1)
    printf("hdmi plug in\n");
else
    printf("hdmi plug out\n");
```



8 sysfs 接口描述

以下两个函数在下面接口的 demo 中会使用到。

```
const int MAX_LENGTH = 128;
const int MAX_DATA = 128;
static ssize_t read_data(const char *sysfs_path, char *data)
{
    ssize_t err = 0;
    FILE *fp = NULL;
    fp = fopen(sysfs_path, "r");
    if (fp) {
        err = fread(data, sizeof(char), MAX_DATA, fp);
        fclose(fp);
    }
    return err;
}

static ssize_t write_data(const char *sysfs_path, const char *data, size_t len)
{
    ssize_t err = 0;
    int fd = -1;
    fd = open(sysfs_path, O_WRONLY);
    if (fd) {
        errno = 0;
        err = write(fd, data, len);
        if (err < 0) {
            err = -errno;
        }
        close(fd);
    } else {
        ALOGE("%s: Failed to open file: %s error: %s", __FUNCTION__, sysfs_path,
            strerror(errno));
        err = -errno;
    }
    return err;
}
```

8.1 enhance

8.1.1 enhance_mode

- 系统节点

```
/sys/class/disp/disp/attr/disp
/sys/class/disp/disp/attr/enhance_mode
```

- 参数

参数	说明
disp display channel	比如 0: disp0, 1: disp1
enhance_mode	0: standard, 1: enhance, 2: soft, 3: enhance + demo

- 返回值

no。

- 描述

该接口用于设置色彩增强的模式。

- 示例

```
//设置disp0 的色彩增强的模式为增强模式
echo 0 > /sys/class/disp/disp/attr/disp;
echo 1 > /sys/class/disp/disp/attr/enhance_mode;
//设置disp1 的色彩增强的模式为柔和模式
echo 1 > /sys/class/disp/disp/attr/disp;
echo 2 > /sys/class/disp/disp/attr/enhance_mode;
//设置disp0 的色彩增强的模式为增加模式，并且开启演示模式
echo 0 > /sys/class/disp/disp/attr/disp;
echo 3 > /sys/class/disp/disp/attr/enhance_mode;
```

c/c++ 代码：

```
char sysfs_path[MAX_LENGTH];
char sysfs_data[MAX_DATA];
unsigned int disp = 0;
unsigned int enhance_mode = 1;
snprintf(sysfs_path, sizeof(sysfs_full_path), "sys/class/disp/disp/attr/disp");
snprintf(sysfs_data, sizeof(sysfs_data), "%d", disp);
write_data(sysfs_path, sys_data, strlen(sysfs_data));
snprintf(sysfs_path, sizeof(sysfs_full_path),
"/sys/class/disp/disp/attr/enhance_mode");
snprintf(sysfs_data, sizeof(sysfs_data), "%d", enhance_mode);
write_data(sysfs_path, sys_data, strlen(sysfs_data));
```

8.1.2 enhance_bright/contrast/saturation/edge/detail/denoise

- 系统节点

```
/sys/class/disp/disp/attr/disp  
/sys/class/disp/disp/attr/enhance_bright /* 亮度*/  
/sys/class/disp/disp/attr/enhance_contrast /* 对比度*/  
/sys/class/disp/disp/attr/enhance_saturation /* 饱和*/  
/sys/class/disp/disp/attr/enhance_edge /* 边缘锐度*/  
/sys/class/disp/disp/attr/enhance_detail /* 细节增强*/  
/sys/class/disp/disp/attr/enhance_denoise /* 降噪*/
```

- 参数

```
disp display channel, 比如0: disp0, 1: disp1.  
enhance_xxx: 范围: 0~100, 数据越大, 调节幅度越大。
```

- 返回值

no。

- 描述

该接口用于设置图像的亮度/对比度/饱和度/边缘锐度/细节增强/降噪的调节幅度。

- 示例

```
//设置disp0 的图像亮度为80  
echo 0 > /sys/class/disp/disp/attr/disp;  
echo 80 > /sys/class/disp/disp/attr/enhance_bright;  
//设置disp1 的饱和度为50  
echo 1 > /sys/class/disp/disp/attr/disp;  
echo 50 > /sys/class/disp/disp/attr/enhance_saturation;
```

c/c++ 代码:

```
char sysfs_path[MAX_LENGTH];  
char sysfs_data[MAX_DATA];  
unsigned int disp = 0  
unsigned int enhance_bright = 80;  
snprintf(sysfs_path, sizeof(sysfs_full_path), "sys/class/disp/disp/attr/disp");  
snprintf(sysfs_data, sizeof(sysfs_data), "%d", disp);  
write_data(sysfs_path, sys_data, strlen(sysfs_data));  
snprintf(sysfs_path, sizeof(sysfs_full_path),  
"/sys/class/disp/disp/attr/enhance_bright");  
snprintf(sysfs_data, sizeof(sysfs_data), "%d", enhance_bright);  
write_data(sysfs_path, sys_data, strlen(sysfs_data));
```

8.2 hdmi edid

8.2.1 edid

- 系统节点

```
/sys/class/hdmi/hdmi/attr/edid
```

- 参数

no。

- 返回值

Edid data(1024 bytes)。

- 描述

该接口用于读取 EDID 的裸数据。

- 示例

```
// 读取edid数据  
cat /sys/class/hdmi/hdmi/attr/edid
```

c/c++ 代码：

```
#define EDID_MAX_LENGTH 1024  
char sysfs_path[MAX_LENGTH];  
char sysfs_data[EDID_MAX_LENGTH];  
ssize_t edid_length;  
  
snprintf(sysfs_path, sizeof(sysfs_full_path), "/sys/class/hdmi/hdmi/attr/edid");  
edid_length = read_data(sysfs_path, sys_data);
```

8.2.2 hpd

- 系统节点

```
/sys/class/switch/hdmi/state
```

- 参数

no。

- 返回值

Hdmi hotplut state, 0: unplug; 1: plug in。

- 描述

该接口用于读取 HDMI 的热插拔状态。

- 示例

```
// 读取HDMI热插拔状态  
cat /sys/class/switch/hdmi/state
```

c/c++ 代码：

```
char sysfs_path[MAX_LENGTH];  
char sysfs_data[MAX_DATA];  
int hpd;  
  
snprintf(sysfs_path, sizeof(sysfs_full_path), "/sys/class/hdmi/hdmi/attr/edid");  
read_data(sysfs_path, sys_data);  
hpd = atoi(sys_data);  
If (hpd)  
    printf("hdmi plug in\n");  
else  
    printf("hdmi unplug \n");
```

8.2.3 hdcp_enable

- 系统节点

```
/sys/class/hdmi/hdmi/attr/hdcp_enable
```

- 参数

enable: 0: disable hdmi hdcp function; 1: enable hdmi hdcp function。

- 返回值

No returns。

- 描述

该接口用于使能、关闭 hdmi hdcp 功能。

- 示例

```
// 开启hdmi hdcp功能
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_enable

// 关闭hdmi hdcp功能
echo 0 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```

c/c++ 代码：

```
char sysfs_path[MAX_LENGTH];
char sysfs_data[MAX_DATA];

snprintf(sysfs_path, sizeof(sysfs_full_path), "/sys/class/hdmi/hdmi/attr/hdcp_enable");
snprintf(sysfs_data, sizeof(sysfs_data), "%d", 1);
write_data(sysfs_path, sys_data, strlen(sysfs_data));
```

9 Data Structure

9.1 disp_fb_info

- 原型

```
typedef struct
{
    unsigned long long    addr[3];          /* address of frame buffer,
                                             single addr for interleaved fomart,
                                             double addr for semi-planar fomart
                                             triple addr for planar format */
    disp_rectsz           size[3];          //size for 3 component,unit: pixels
    unsigned int          align[3];         //align for 3 comonent,unit: bytes(align=2^n,i.e
    .1/2/4/8/16/32..)
    disp_pixel_format     format;
    disp_color_space      color_space;      //color space
    unsigned int          trd_right_addr[3];/* right address of 3d fb,
                                             used when in frame packing 3d mode */
    bool                  pre_multiply;     //true: pre-multiply fb
    disp_rect64           crop;             //crop rectangle boundaries
    disp_buffer_flags     flags;            //indicate stereo or non-stereo buffer
    disp_scan_flags       scan;             //scan type & scan order
}disp_fb_info;
```

- 成员

变量	说明
addr	framebuffer 的内容地址，对于 interleaved 类型，只有 addr[0] 有效；planar 类型，三个都有；UV combined 的类型 addr[0]，addr[1] 有效
size	size of framebuffer, 单位为 pixel
align	对齐位宽，为 2 的指数
format	pixel format, 详见 disp_pixel_format
color_space	color space mode, 详见 disp_cs_mode
b_trd_src	1:3D source; 0: 2D source
trd_mode	source 3D mode, 详见 disp_3d_src_mode
trd_right_addr	used when in frame packing 3d mode
crop	用于显示的 buffer 裁减区
flags	标识 2D 或 3D 的 buffer
scan	标识描述类型，progress, interleaved

- 描述

disp_fb_info 用于描述一个 display framebuffer 的属性信息。

9.2 disp_layer_info

- 原型

```
typedef struct
{
    disp_layer_mode      mode;
    unsigned char        zorder; /*specifies the front-to-back ordering of the layers on
    the screen,
                                the top layer having the highest Z value
                                can't set zorder, but can get */
    unsigned char        alpha_mode; //0: pixel alpha; 1: global alpha; 2: global
    pixel alpha
    unsigned char        alpha_value; //global alpha value
    disp_rect            screen_win; //display window on the screen
    bool                 b_trd_out;  //3d display
    disp_3d_out_mode      out_trd_mode; //3d display mode
    union {
        unsigned int     color;      //valid when LAYER_MODE_COLOR
        disp_fb_info      fb;        //framebuffer, valid when LAYER_MODE_BUFFER
    };

    unsigned int          id; /* frame id, can get the id of frame display currently
    by DISP_LAYER_GET_FRAME_ID */
}disp_layer_info;
```

- 成员

变量	说明
mode	图层的模式, 详见 disp_layer_mode
zorder	layer zorder, 优先级高的图层可能会覆盖优先级低的图层
alpha_mode	0:pixel alpha, 1:global alpha, 2:global pixel alpha
alpha_value	layer global alpha value, valid while alpha_mode(1/2)
screenn_win	screen window, 图层在屏幕上显示的矩形窗口
fb	framebuffer 的属性, 详见 disp_fb_info,valid when BUFFER_MODE
color	display color, valid when COLOR_MODE
b_trd_out	if output in 3d mode,used for scaler layer
out_trd_mode	output 3d mode, 详见 disp_3d_out_mode
id	frame id, 设置给驱动的图像帧号, 可以通过 DISP_LAYER_GET_FRAME_ID 获取当前显示的帧号, 以做一下特定的处理, 比如释放掉已经显示完成的图像帧 buffer。

- 描述

disp_layer_info 用于描述一个图层的属性信息。

9.3 disp_layer_config

- 原型

```
typedef struct
{
    disp_layer_info info;
    bool enable;
    unsigned int channel;
    unsigned int layer_id;
}disp_layer_config;
```

- 成员

变量	说明
info	图像的信息属性
enable	使能标志
channel	图层所在的通道 id (0/1/2/3)
layer_id	图层的 id, 此 id 是在通道内的图层 id。即 (channel,layer_id)=(0,0) 表示通道 0 中的图层 0 之意

- 描述

disp_layer_config 用于描述一个图层配置的属性信息。

9.4 disp_color_info

- 原型

```
typedef struct
{
    u8 alpha;
    u8 red;
    u8 green;
    u8 blue;
}disp_color_info;
```

- 成员

变量	说明
alpha	颜色的透明度
red	红
green	绿
blue	蓝

- 描述

disp_color_info 用于描述一个颜色的信息。

9.5 disp_rect

- 原型

```
typedef struct
{
    s32  x;
    s32  y;
    u32  width;
    u32  height;
}disp_rect;
```

- 成员

变量	参数
x	起点 x 值
y	起点 y 值
width	宽
height	高

- 描述

disp_rect 用于描述一个矩形窗口的信息。

9.6 disp_rect64

- 原型

```
typedef struct
{
    long long x;
    long long y;
    long long width;
    long long height;
}disp_rect64;
```

- 成员

变量	说明
x	起点 x 值, 定点小数, 高 32bit 为整数, 低 32bit 为小数
y	起点 y 值, 定点小数, 高 32bit 为整数, 低 32bit 为小数
width	宽, 定点小数, 高 32bit 为整数, 低 32bit 为小数
height	高, 定点小数, 高 32bit 为整数, 低 32bit 为小数

- 描述

disp_rect64 用于描述一个矩形窗口的信息。

9.7 disp_position

- 原型

```
typedef struct
{
    s32 x;
    s32 y;
}disp_posistion;
```

- 成员

变量	说明
x	x
y	y

- 描述

disp_position 用于描述一个坐标的信息。

9.8 disp_rectsz

- 原型

```
typedef struct
{
    u32 width;
    u32 height;
}disp_rectsz;
```

- 成员

变量	说明
width	宽
height	高

- 描述

disp_rectsz 用于描述一个矩形尺寸的信息。

9.9 disp_pixel_format

- 原型

```
typedef enum
{
    DISP_FORMAT_ARGB_8888          = 0x00, //MSB  A-R-G-B  LSB
    DISP_FORMAT_ABGR_8888          = 0x01,
    DISP_FORMAT_RGBA_8888          = 0x02,
    DISP_FORMAT_BGRA_8888          = 0x03,
    DISP_FORMAT_XRGB_8888          = 0x04,
    DISP_FORMAT_XBGR_8888          = 0x05,
    DISP_FORMAT_RGBX_8888          = 0x06,
    DISP_FORMAT_BGRX_8888          = 0x07,
    DISP_FORMAT_RGB_888            = 0x08,
    DISP_FORMAT_BGR_888            = 0x09,
    DISP_FORMAT_RGB_565            = 0x0a,
```

```

DISP_FORMAT_BGR_565           = 0x0b,
DISP_FORMAT_ARGB_4444        = 0x0c,
DISP_FORMAT_ABGR_4444        = 0x0d,
DISP_FORMAT_RGBA_4444        = 0x0e,
DISP_FORMAT_BGRA_4444        = 0x0f,
DISP_FORMAT_ARGB_1555        = 0x10,
DISP_FORMAT_ABGR_1555        = 0x11,
DISP_FORMAT_RGBA_5551        = 0x12,
DISP_FORMAT_BGRA_5551        = 0x13,

/* SP: semi-planar, P:planar, I:interleaved
 * UVUV: U in the LSBs;   VUVU: V in the LSBs */
DISP_FORMAT_YUV444_I_AYUV      = 0x40, //MSB  A-Y-U-V  LSB
DISP_FORMAT_YUV444_I_VUYA      = 0x41, //MSB  V-U-Y-A  LSB
DISP_FORMAT_YUV422_I_YVYU      = 0x42, //MSB  Y-V-Y-U  LSB
DISP_FORMAT_YUV422_I_YUYV      = 0x43, //MSB  Y-U-Y-V  LSB
DISP_FORMAT_YUV422_I_UYVY      = 0x44, //MSB  U-Y-V-Y  LSB
DISP_FORMAT_YUV422_I_VYUY      = 0x45, //MSB  V-Y-U-Y  LSB
DISP_FORMAT_YUV444_P           = 0x46, //MSB  P3-2-1-0 LSB,  YYYY UUUU VVVV
DISP_FORMAT_YUV422_P           = 0x47, //MSB  P3-2-1-0 LSB  YYYY UU   VV
DISP_FORMAT_YUV420_P           = 0x48, //MSB  P3-2-1-0 LSB  YYYY U    V
DISP_FORMAT_YUV411_P           = 0x49, //MSB  P3-2-1-0 LSB  YYYY U     V
DISP_FORMAT_YUV422_SP_UVUV      = 0x4a, //MSB  V-U-V-U  LSB
DISP_FORMAT_YUV422_SP_VUVU      = 0x4b, //MSB  U-V-U-V  LSB
DISP_FORMAT_YUV420_SP_UVUV      = 0x4c,
DISP_FORMAT_YUV420_SP_VUVU      = 0x4d,
DISP_FORMAT_YUV411_SP_UVUV      = 0x4e,
DISP_FORMAT_YUV411_SP_VUVU      = 0x4f,
}disp_pixel_format;

```

• 成员

变量	说明
DISP_FORMAT_ARGB_8888	32bpp, A 在最高位, B 在最低位
DISP_FORMAT_YUV420_P	planar yuv 格式, 分三块存放, 需三个地址, P3 在最高位
DISP_FORMAT_YUV422_SP_UVUV	semi-planar yuv 格式, 分两块存放, 需两个地址, U 在低位
DISP_FORMAT_YUV422_SP_VUVU	semi-planar yuv 格式, 分两块存放, 需两个地址, V 在低位

• 描述

disp_pixel_format 用于描述像素格式。

9.10 disp_buffer_flags

• 原型

```
typedef enum
{
    DISP_BF_NORMAL          = 0, //non-stereo
    DISP_BF_STEREO_TB       = 1 << 0, //stereo top-bottom
    DISP_BF_STEREO_FP       = 1 << 1, //stereo frame packing
    DISP_BF_STEREO_SSH      = 1 << 2, //stereo side by side half
    DISP_BF_STEREO_SSF      = 1 << 3, //stereo side by side full
    DISP_BF_STEREO_LI       = 1 << 4, //stereo line interlace
}disp_buffer_flags;
```

- 成员

变量	说明
DISP_BF_NORMAL	2d
DISP_BF_STEREO_TB	top bottom 模式
DISP_BF_STEREO_FP	framepacking
DISP_BF_STEREO_SSF	side by side full, 左右全景
DISP_BF_STEREO_SSH	side by side half, 左右半景
DISP_BF_STEREO_LI	line interleaved, 行交错模式

- 描述

disp_buffer_flags 用于描述 3D 源模式。

9.11 disp_3d_out_mode

- 原型

```
typedef enum
{
    //for lcd
    DISP_3D_OUT_MODE_CI_1 = 0x5, //column interlaved 1
    DISP_3D_OUT_MODE_CI_2 = 0x6, //column interlaved 2
    DISP_3D_OUT_MODE_CI_3 = 0x7, //column interlaved 3
    DISP_3D_OUT_MODE_CI_4 = 0x8, //column interlaved 4
    DISP_3D_OUT_MODE_LIRGB = 0x9, //line interleaved rgb

    //for hdmi
    DISP_3D_OUT_MODE_TB = 0x0, //top bottom
    DISP_3D_OUT_MODE_FP = 0x1, //frame packing
    DISP_3D_OUT_MODE_SSF = 0x2, //side by side full
    DISP_3D_OUT_MODE_SSH = 0x3, //side by side half
    DISP_3D_OUT_MODE_LI = 0x4, //line interleaved
    DISP_3D_OUT_MODE_FA = 0xa, //field alternative
}disp_3d_out_mode;
```

- 成员

for lcd:

变量	说明
DISP_3D_OUT_MODE_CI_1	列交织
DISP_3D_OUT_MODE_CI_2	列交织
DISP_3D_OUT_MODE_CI_3	列交织
DISP_3D_OUT_MODE_CI_4	列交织
DISP_3D_OUT_MODE_LIRGB	行交织

for hdmi:

变量	说明
DISP_3D_OUT_MODE_TB	top bottom 上下模式
DISP_3D_OUT_MODE_FP	framepacking
DISP_3D_OUT_MODE_SSF	side by side full, 左右全景
DISP_3D_OUT_MODE_SSH	side by side half, 左右半景
DISP_3D_OUT_MODE_LI	line interleaved, 行交织
DISP_3D_OUT_MODE_FA	field alternate 场交错

- 描述

disp_3d_out_mode 用于描述 3D 输出模式。

9.12 disp_color_space

- 原型

```
typedef enum
{
    DISP_BT601 = 0,
    DISP_BT709 = 1,
    DISP_YCC   = 2,
}disp_color_mode;
```

- 成员

变量	说明
DISP_BT601	用于标清视频
DISP_BT709	用于高清视频
DISP_YCC	用于图片

- 描述

disp_color_space 用于描述颜色空间类型。

9.13 disp_output_type

- 原型

```
typedef enum
{
    DISP_OUTPUT_TYPE_NONE    = 0,
    DISP_OUTPUT_TYPE_LCD     = 1,
    DISP_OUTPUT_TYPE_TV      = 2,
    DISP_OUTPUT_TYPE_HDMI    = 4,
    DISP_OUTPUT_TYPE_VGA     = 8,
}disp_output_type;
```

- 成员

变量	说明
DISP_OUTPUT_TYPE_NONE	无显示输出
DISP_OUTPUT_TYPE_LCD	LCD 输出
DISP_OUTPUT_TYPE_TV	TV 输出
DISP_OUTPUT_TYPE_HDMI	HDMI 输出
DISP_OUTPUT_TYPE_VGA	VGA 输出

- 描述

disp_output_type 用于描述显示输出类型。

9.14 disp_tv_mode

- 原型

```
typedef enum
{
    DISP_TV_MOD_480I          = 0,
    DISP_TV_MOD_576I          = 1,
    DISP_TV_MOD_480P          = 2,
    DISP_TV_MOD_576P          = 3,
    DISP_TV_MOD_720P_50HZ     = 4,
    DISP_TV_MOD_720P_60HZ     = 5,
    DISP_TV_MOD_1080I_50HZ    = 6,
    DISP_TV_MOD_1080I_60HZ    = 7,
    DISP_TV_MOD_1080P_24HZ    = 8,
    DISP_TV_MOD_1080P_50HZ    = 9,
    DISP_TV_MOD_1080P_60HZ    = 0xa,
    DISP_TV_MOD_1080P_24HZ_3D_FP = 0x17,
    DISP_TV_MOD_720P_50HZ_3D_FP = 0x18,
    DISP_TV_MOD_720P_60HZ_3D_FP = 0x19,
    DISP_TV_MOD_1080P_25HZ     = 0x1a,
    DISP_TV_MOD_1080P_30HZ     = 0x1b,
    DISP_TV_MOD_PAL            = 0xb,
    DISP_TV_MOD_PAL_SVIDEO    = 0xc,
    DISP_TV_MOD_NTSC          = 0xe,
    DISP_TV_MOD_NTSC_SVIDEO   = 0xf,
    DISP_TV_MOD_PAL_M         = 0x11,
    DISP_TV_MOD_PAL_M_SVIDEO  = 0x12,
    DISP_TV_MOD_PAL_NC        = 0x14,
    DISP_TV_MOD_PAL_NC_SVIDEO = 0x15,
    DISP_TV_MOD_3840_2160P_30HZ = 0x1c,
    DISP_TV_MOD_3840_2160P_25HZ = 0x1d,
    DISP_TV_MOD_3840_2160P_24HZ = 0x1e,
    DISP_TV_MODE_NUM           = 0x1f,
}disp_tv_mode;
```

- 成员
- 描述

disp_tv_mode 用于描述 TV 输出模式。

9.15 disp_output

- 原型

```
typedef struct
{
    unsigned int type;
    unsigned int mode;
```

```
}disp_output;
```

- 成员

变量	说明
Type	输出类型
Mode	输出模式, 480P/576P, etc

- 描述

disp_output 用于描述显示输出类型, 模式。

9.16 disp_layer_mode

- 原型

```
typedef enum
{
    LAYER_MODE_BUFFER = 0,
    LAYER_MODE_COLOR = 1,
}disp_layer_mode;
```

- 成员

变量	说明
LAYER_MODE_BUFFER	buffer 模式, 带 buffer 的图层
LAYER_MODE_COLOR	单色模式, 无 buffer 的图层, 只需要一个颜色值表示图像内容

- 描述

disp_layer_mode 用于描述图层模式。

9.17 disp_scan_flags

- 原型

```
typedef enum
{
    DISP_SCAN_PROGRESSIVE          = 0, //non interlace
    DISP_SCAN_INTERLACED_ODD_FLD_FIRST  = 1 << 0, //interlace ,odd field first
    DISP_SCAN_INTERLACED_EVEN_FLD_FIRST = 1 << 1, //interlace,even field first
}disp_scan_flags;
```

- 成员

变量	说明
DISP_SCAN_PROGRESSIVE	逐行模式
DISP_SCAN_INTERLACED_ODD_FLD_FIRST	隔行模式，奇数行优先
DISP_SCAN_INTERLACED_EVEN_FLD_FIRST	隔行模式，偶数行优先

- 描述

disp_scan_flags 用于描述显示 Buffer 的扫描方式。

10 调试

10.1 查看显示模块的状态

```
cat /sys/class/disp/disp/attr/sys
```

示例如下：

```
# cat /sys/class/disp/disp/attr/sys
screen 0:
de_rate 432000000 Hz /* de 的时钟频率*/, ref_fps=50 /* 输出设备的参考刷新率*/
hdmi output mode(4) fps:50.5 1280x 720
err:0 skip:54 irq:21494 vsync:0
BUF enable ch[0] lyr[0] z[0] prem[N] a[global 255] fmt[ 1] fb
[1920,1080;1920,1080;1920,1080] crop[ 0, 0,1920,1080] frame[ 32, 18,1216, 684]
addr[716da000, 0, 0] flags[0x 0] trd[0,0]
screen 1:
de_rate 432000000 Hz /* de 的时钟频率*/, ref_fps=50 /* 输出设备的参考刷新率*/
tv output mode(11) fps:50.5 720x 576 /* TV 输出 | 模式为 (11: PAL) | 刷新率为: 50.5Hz | 分辨率: 720x576 */
err:0 skip:54 irq:8372 vsync:0
BUF enable ch[0] lyr[0] z[0] prem[Y] a[global 255] fmt[ 0] fb[ 720, 576; 720, 576; 720,
576] crop[ 0, 0, 720, 576] frame[ 18, 15, 684, 546]
addr[739a8000, 0, 0] flags[0x 0] trd[0,0]
acquire: 225, 2.6 fps
release: 224, 2.6 fps
display: 201, 2.5 fps
```

图层各信息描述如下：

BUF: 图层类型, BUF/COLOR, 一般为BUF, 即图层是带BUFFER 的。COLOR 意思是显示一个纯色的画面, 不带 BUFFER。

enable: 显示处于enable 状态。

ch[0]: 该图层处于blending 通道0。

lyr[0]: 该图层处于当前blending 通道中的图层0。

z[0]: 图层z 序, 越小越在底部, 可能会被z 序大的图层覆盖住。

prem[Y]: 是否预乘格式, Y 是, N 否。

a: alpha 参数, global/pixel/; alpha 值。

fmt: 图层格式, 值64 以下为RGB 格式; 以上为YUV 格式, 常见的72 为YV12, 76 为NV12。

fb: 图层buffer 的size, width,height, 三个分量。

crop: 图像buffer 中的裁减区域, [x,y,w,h]。

frame: 图层在屏幕上的显示区域, [x,y,w,h]。

addr: 三个分量的地址。

flags: 一般为0, 3D SS 时0x4, 3D TB 时为0x1, 3D FP 时为0x2。

trd: 是否3D 输出, 3D 输出的类型 (HDMI FP 输出时为1) 各counter 描述如下:

err: de 缺数的次数, de 缺数可能会出现屏幕抖动, 花屏的问题。de 缺数一般为带宽不足引起。

skip: 表示de 跳帧的次数, 跳帧会出现卡顿问题。跳帧是指本次中断响应较慢, de 模块判断在本次中断已经接近或者超过了消隐区, 将放弃本次更新图像的机会, 选择继续显示原有的图像。

irq: 表示该通路上垂直消隐区中断执行的次数, 一直增长表示该通道上的timing。

controller 正在运行当中。

vsync:表示显示模块往用户空间中发送的vsync 消息的数目，一直增长表示正在不断地发送中。

acquire/release/display 含义如下,只在android 方案中有效。

acquire: 是hw composer 传递给disp driver 的图像帧数以及帧率，帧率只要有在有图像更新时才有效，静止时的值是不准确的。

release: 是disp driver 显示完成之后，返还给android 的图像帧数以及帧率，帧率只要有在有图像更新时才有效，静止时的值是不准确的。

display: 是disp 显示到输出设备上的帧数以及帧率，帧率只要有在有图像更新时才有效，静止时的值是不准确的。如果acquire 与release 不一致，说明disp 有部分图像帧仍在使使用，未返还，差值在1~2 之间为正常值。二者不能相等，如果相等，说明图像帧全部返还，显示将会出。

现撕裂现象。如果display 与release 不一致，说明在disp 中存在丢帧情况，原因为在一个active 区内 hwcomposer 传递多于一帧的图像帧下来。

调试说明：

1. 对于android 系统，可以dumpsys SurfaceFlinger 打印surface 的信息，如果信息与disp 中sys 中的信息不一致，很大可能是hwc 的转换存在问题。
2. 如果发现图像刷新比较慢，存在卡顿问题，可以看一下输出设备的刷新率，对比一下ref_fps 与fps 是否一致，如果不一致，说明tcon 的时钟频率或timing 没配置正确。如果ref_fps 与屏的spec 不一致，则需要检查 sys_config 中的时钟频率和timing配置是否正确。屏一般为60Hz，而如果是TV 或HDMI，则跟模式有关，比较常见的为60/50/30/24Hz。
3. 如果是android 方案，还可以看一下display 与release 的counter 是否一致，如果相差太大，说明android 送帧不均匀，造成丢帧。
3. 如果发现图像刷新比较慢，存在卡顿问题，也需要看一下skip counter，如果skip counter 有增长，说明现在的系统负荷较重，对vblank 中断的响应较慢，出现跳帧，导致了图像卡顿问题。
4. 如果屏不亮，怀疑背光时，可以看一下屏的背光值是否为0。如果为0，说明上层传递下来的背光值不合理；如果不为0，背光还是不亮，则为驱动或硬件问题了。硬件上可以通过测量bl_en 以及pwm 的电压值来排查问题。
5. 如果花屏或图像抖动，可以查看err counter，如果err counter 有增长，则说明de缺数，有可能是带宽不足，或者瞬时带宽不足问题。

10.2 截屏

```
echo 0 > /sys/class/disp/disp/attr/disp
echo /data/filename.bmp > /sys/class/disp/disp/attr/capture_dump
```

该调试方法用于截取 DE 输出到 TCON 前的图像，用于显示通路上分段排查。如果截屏没有问题而界面异常，可以确定 TCON 到显示器间出错。

第一个路径接受显示器索引 0 或 1。

第二个路径接受文件路径。

10.3 colorbar

```
echo 0 > /sys/class/disp/disp/attr/disp
echo > /sys/class/disp/disp/attr/colorbar
```

第一个路径接受显示器索引 0 或 1。

第二个路径表示 TCON 选择的输入源。1, DE 输出; 2-7, TCON 自检用的 colorbar; 8, DE 自检用的 colorbar。

10.4 显示模块 debugfs 接口

10.4.1 总述

调试节点

```
mount -t debugfs none /sys/kerne/debug;  
/sys/kernel/debug/dispdbg;
```

```
name command param start info
```

```
//name: 表示操作的对象名字  
//command: 表示执行的命令  
//param: 表示该命令接收的参数  
//start: 输入1 开始执行命令  
//info: 保存命令执行的结果
```

```
//只读, 大小是1024 bytes。
```

10.4.2 切换显示输出设备

```
name: disp0/1/2 //表示显示通道0/1/2  
command: switch  
param: type mode  
  
//参数说明: type:0(none),1(lcd),2(tv),4(hdmi),8(vga)  
//mode 详见disp_tv_mode 定义
```

```
/* 例子 */
```

```
/* 显示通道0 输出LCD */
```

```
echo disp0 > name;echo switch > command;echo 1 0 > param;echo 1 > start;
```

```
/* 关闭显示通道0 的输出*/
```

```
echo disp0 > name;echo switch > command;echo 0 0 > param;echo 1 > start;
```

10.4.3 开关显示输出设备

```
name: disp0/1/2 //表示显示通道0/1/2  
command: blank  
param: 0/1
```

```
//参数说明: 1 表示blank, 即关闭显示输出; 0 表示unblank, 即开启显示输出
```

```
/* 例子 */
```

```
/* 关闭显示通道0 的显示输出*/  
echo disp0 > name;echo blank > command;echo 1 > param;echo 1 > start;  
/* 开启显示通道1 的显示输出*/  
echo disp1 > name;echo blank > command;echo 0 > param;echo 1 > start;
```

10.4.4 电源管理 (suspend/resume) 接口

```
name: disp0/1/2 //表示显示通道0/1/2  
command: suspend/resume //休眠, 唤醒命令  
param: 无  
  
/* 例子 */  
/* 让显示模块进入休眠状态*/  
echo disp0 > name;echo suspend > command;echo 1 > start;  
/* 让显示模块退出休眠状态*/  
echo disp1 > name;echo resume > command;echo 1 > start;
```

10.4.5 调节 lcd 屏幕背光

```
name: lcd0/1/2 //表示lcd0/1/2  
command: setbl //设置背光亮度的命令  
param: xx  
  
//参数说明: 背光亮度的值, 范围是0~255。  
  
/* 例子 */  
/* 设置背光亮度的值为100 */  
echo lcd0 > name;echo setbl > command;echo 100 > param;echo 1 > start;  
/* 设置背光亮度的值为0 */  
echo lcd0 > name;echo setbl > command;echo 0 > param;echo 1 > start;
```

10.4.6 vsync 消息开关

```
name: disp0/1/2 //表示显示通道0/1/2  
command: vsync_enable //开启/关闭vsync 消息  
param: 0/1  
  
//参数说明: 0: 表示关闭; 1: 表示开启  
  
/* 例子 */  
/* 关闭显示通道0 的vsync 消息*/  
echo disp0 > name;echo vsync_enable > command;echo 0 > param;echo 1 > start;  
/* 开启显示通道1 的vsync 消息*/  
echo disp1 > name;echo vsync_enable > command;echo 1 > param;echo 1 > start;
```

10.4.7 查看 enhance 的状态

```
name: enhance0/1/2 //表示enhance0/1/2
command: getinfo //获取enhance 的状态
param: 无

/* 例子 */
/* 获取显示通道0 的enhance 状态信息*/
# echo enhance0 > name;echo getinfo > command;echo 1 > start;cat info;
# enhance 0: enable, normal
```

10.4.8 查看智能背光的状态

```
name: smbl0/1/2 //表示显示通道0/1/2
command: getinfo //获取smart backlight 的状态
param: 无

/* 例子 */
/* 获取显示通道0 的smb1 状态信息*/
# echo smbl0 > name;echo getinfo > command;echo 1 > start;cat info;
# smbl 0: disable, window<0,0,0,0>, backlight=0, save_power=0 percent
//显示的是智能背光是否开启, 有效窗口大小, 当前背光值, 省电比例
```

10.5 常见问题

10.5.1 黑屏（无背光）

问题现象：机器接 LCD 输出，发现 LCD 没有任何显示，仔细查看背光也不亮。

问题分析：此现象说明 LCD 背光供电不正常，不排除还有其他问题，但没背光的问题必须先解决。

问题排查步骤：

- 步骤一

使用电压表量 LCD 屏的各路电压，如果背光灯脚电压不正常，确定是 PWM 问题。否则，尝试换个屏再试。

- 步骤二

先看看随 sdk 有没发布 PWM 模块使用指南，如果有按照里面步骤进行排查。

- 步骤三

如果 sdk 没有发布 PWM 模块使用指南。可以 `cat /sys/kernel/debug/pwm` 看看有没输出。如果没有就是 PWM 驱动没有加载，请检查一下 `menuconfig` 有没打开。

- 步骤四

如果步骤三未解决问题，请排查 `dtb` 或 `board.dts` 配置。如果还没有解决，可以寻求技术支持。

10.5.2 黑屏（有背光）

问题现象：机器接 LCD，发现有背光，界面输出黑屏。

问题分析：此现象说明没有内容输出，可能是 DE、TCON 出错或应用没有送帧。

问题排查步骤：

- 步骤一

根据 10.1 章节排查应用输入的图层信息是否正确。其中，宽高、显存的文件句柄出错问题最多。

- 步骤二

根据 10.2 章节截屏，看看 DE 输出是否正常。如果不正常，排查 DE 驱动配置是否正确；如果正常，接着下面步骤。

- 步骤三

根据 10.3 章节输出 `colorbar`，如果 TCON 自身的 `colorbar` 也没有显示，排查硬件通路；如果有显示，排查 TCON 输入源选择的寄存器。后者概率很低，此时可寻求技术支持。

10.5.3 绿屏

问题现象：显示器出现绿屏，切换界面可能有其他变化。

问题分析：此现象说明处理图层时 DE 出错。可能是应用送显的 `buffer` 内容或者格式有问题；也可能 DE 配置出错。

问题排查步骤：

- 步骤一

根据 10.1 章节排查应用输入的图层信息是否正确。其中，图层格式填错的问题最多。

- 步骤二

导出 DE 寄存器，排查异常。此步骤比较复杂，需要寻求技术支持。

10.5.4 界面卡住

问题现象：界面定在一个画面，不再改变。

问题分析：此现象说明显示通路一般是正常的，只是应用没有继续送帧。

问题排查步骤：

- 步骤一

根据 10.1 章节排查应用输入的图层信息有没改变，特别关注图层的地址。

- 步骤二

排查应用送帧逻辑，特别关注死锁，线程、进行异常退出，fence 处理异常。

10.5.5 局部界面花屏

问题现象：画面切到特定场景时候，出现局部花屏，并不断抖动。

问题分析：此现象是典型的 DE scaler 出错现象。

问题排查步骤：

根据 10.1 章节查看问题出现时带缩放图层的参数。如果屏幕输出的宽 x 高除以 crop 的宽 x 高小于 1/16 或者大于 32，那么该图层不能走 DE 缩放，改用 GPU，或应用修改图层宽高。

10.5.6 快速切换界面花屏

问题现象：快速切换界面花屏，变化不大的界面显示正常。

问题分析：此现象是典型的性能问题，与显示驱动关系不大。

问题排查步骤：

- 步骤一

排查 DRAM 带宽是否满足场景需求。

- 步骤二

若是安卓系统，排查 fence 处理流程；若是纯 linux 系统，排查送帧流程、swap buffer、pan-display 流程。



著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 全志科技、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。