

Tina

显示模块使用文档

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2017/11/11		统一显示模块文档

Yllwinnertec

目 录

1. 概述.....	5
1.1. 编写目的.....	5
1.2. 适用范围.....	5
1.3. 相关人员.....	5
2. 模块介绍.....	6
2.1. 模块功能介绍.....	6
2.2. 相关术语介绍.....	6
2.3. 模块配置介绍.....	6
2.3.1. menuconfig 配置说明.....	6
2.4. 源码结构介绍.....	6
3. 图层操作说明.....	8
4. 显示输出设备操作说明.....	9
5. 接口参数更改说明.....	10
6. 图层主要参数介绍.....	11
6.1. Size 与 crop.....	11
6.2. crop 和 screen_win.....	11
6.3. alpha.....	11
6.4. Format 支持.....	12
7. 输出设备介绍.....	14
7.1. 屏.....	14
7.2. HDMI.....	14
7.3. 同显.....	14
8. IOCTL 接口描述.....	15
8.1. Global Interface.....	15
DISP_SHADOW_PROTECT.....	15
DISP_SET_BKCOLOR.....	15
DISP_GET_BKCOLOR.....	15
DISP_GET_SCN_WIDTH.....	16
DISP_GET_SCN_HEIGHT.....	16
DISP_GET_OUTPUT_TYPE.....	17
DISP_GET_OUTPUT.....	17
DISP_VSYNC_EVENT_EN.....	17
DISP_DEVICE_SWITCH.....	18
8.2. Layer Interface.....	18
DISP_LAYER_SET_CONFIG.....	18
DISP_LAYER_GET_CONFIG.....	19
8.3. capture interface.....	20
DISP_CAPTURE_START.....	20
DISP_CAPTURE_STOP.....	20
DISP_CAPTURE_COMMIT.....	20
8.4. LCD Interface.....	21
DISP_LCD_SET_BRIGHTNESS.....	21
DISP_LCD_GET_BRIGHTNESS.....	21
8.5. enhance interface.....	21
DISP_ENHANCE_ENABLE.....	21
DISP_ENHANCE_DISABLE.....	22
DISP_ENHANCE_DEMO_ENABLE.....	22
DISP_ENHANCE_DEMO_DISABLE.....	22
8.6. smart backlight.....	23
DISP_SMBL_ENABLE.....	23

DISP_SMBL_DISABLE.....	23
DISP_SMBL_SET_WINDOW.....	23
8.7. hdmi interface.....	24
DISP_HDMI_SUPPORT_MODE.....	24
DISP_HDMI_GET_HPD_STATUS.....	24
9. sysfs 接口描述.....	26
9.1. enhance.....	26
enhance_mode.....	26
9.2. hdmi edid.....	27
edid.....	27
hpd.....	27
hdcp_enable.....	28
10. Data Structure.....	29
disp_fb_info.....	29
disp_layer_info.....	29
disp_layer_config.....	30
disp_color_info.....	30
disp_rect.....	31
disp_rect64.....	31
disp_position.....	31
disp_rectsz.....	32
disp_pixel_format.....	32
disp_buffer_flags.....	33
disp_3d_out_mode.....	33
disp_color_space.....	34
disp_output_type.....	34
disp_tv_mode.....	35
disp_output.....	36
disp_layer_mode.....	36
disp_scan_flags.....	36
11. Declaration.....	37

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

本模块设计适用于 F35 平台。
本模块设计适用于 aw1681

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。

Yllwinner

	—disp2/	disp2 的目录
	—disp	
	—dev_disp.c	display driver 层
	—dev_fb.c	framebuffer driver 层
	—de	bsp 层
	—disp_lcd.c disp_manager.c ..	
	—disp_al.c	al 层
	—lowlevel_sun*/	lowlevel 层
	—de_lcd.c de_rtmx.c...	
	—disp_sys_int.	OSAL 层,与操作系统相关层
	—lcd/	lcd driver
	—lcd_src_interface.c	与 display 驱动接口
	—default_panel.c ...	平台已经支持的屏驱动
include		
	—video	video header dir
	—sunxi_display2.c	display header file

3. 图层操作说明

显示驱动中最重要的显示资源为图层，sunxi 中支持 1 到 2 路显示通道，0 路显示一般支持 16 个图层（其中视频图层 4 个），3 个 blending 通道；1 路一般支持 8 个图层（其中视频图层 4 个），1 个 Blending 通道，所有图层都支持缩放。对图层的操作如下所示。图层以 disp, channel, layer_id 三个索引唯一确定（disp:0/1, channel: 0/1/2/3, layer_id:0/1/2/3）。

- 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。
- 关闭图层，依然通过 DISP_LAYER_SET_CONFIG，将 enable 参数设置为 0 关闭。

Yllwinner

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 `dtb` 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`。

Yllwinner

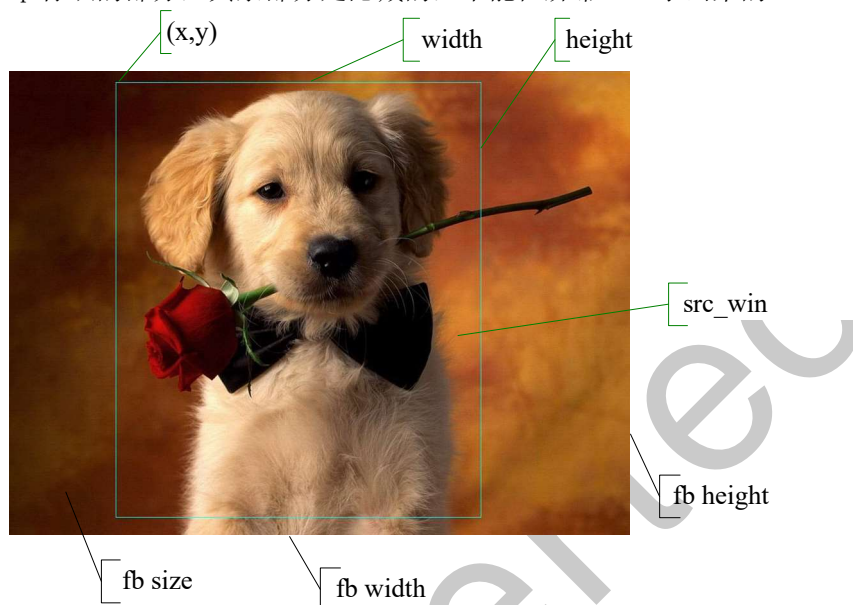
5. 接口参数更改说明

项目\平台	Disp2	Displ
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放置于 DISP_LAYER_SET_CONFIG 接口中	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, 用户保证 zorder 不重复, 从 0 到 N-1 (N 表示支持的最大图层数)	用户不能设置
设置图层信息接口	一次可设置多个图层的的信息, 增加一个图层信息数目的参数	一次设置 1 个图层信息

6. 图层主要参数介绍

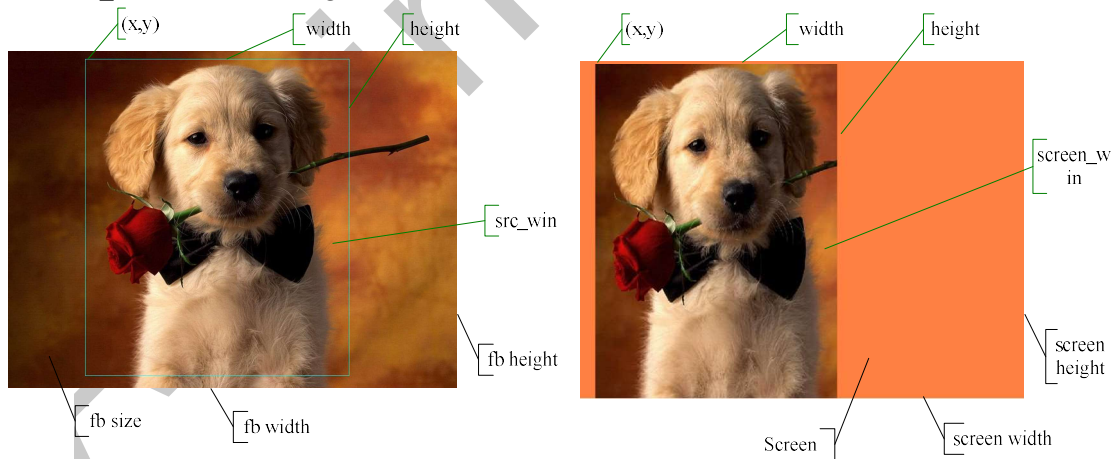
6.1.Size 与 crop

Fb 有两个与 size 有关的参数，分别是 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。



6.2.crop 和 screen_win

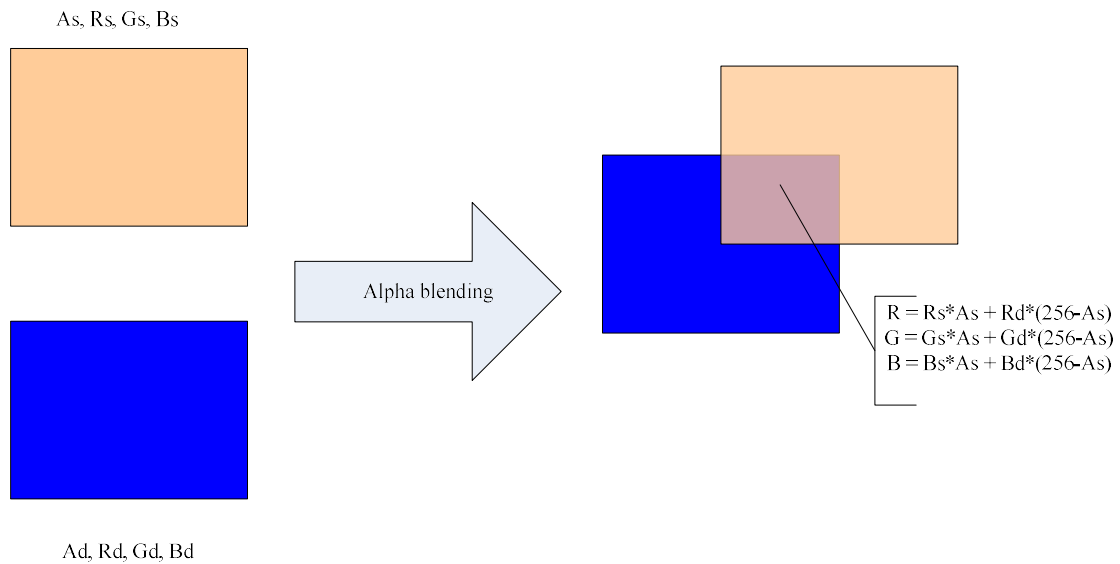
Src_win 上面已经介绍过了。Screen_win 为 crop 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话，crop 和 screen_win 的 width,height 是相等的，如果需要缩放，需要用 scaler_mode 的图层来显示，crop 和 screen_win 的 width,height 可以不等。



6.3.alpha

Alpha 模式有三种：

- 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
- 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
- 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。



6.4.Format 支持

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

```

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

AIwinne

7. 输出设备介绍

该平台支持屏以及 HDMI 输出，及二者同时显示。

7.1.屏

屏的接口很多，该平台支持 RGB/CPU/LVDS/DSI 接口。

7.2.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。

7.3.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 FBO 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。

8. 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。

8.1. Global Interface

DISP_SHADOW_PROTECT

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功，返回 DIS_SUCCESS，否则，返回失败号;

➤ **DESCRIPTION**

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。

➤ **DEMO**

```
//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);
```

DISP_SET_BKCOLOR

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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;

➤ **RETURNS**

如果成功，返回 DIS_SUCCESS，否则，返回失败号;

➤ **DESCRIPTION**

该函数用于设置显示背景色。

➤ **DEMO**

```
//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);
```

DISP_GET_BKCOLOR

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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;

➤ **RETURNS**

如果成功, 返回 DIS_SUCCESS, 否则, 返回失败号;

➤ **DESCRIPTION**

该函数用于获取显示背景色。

➤ **DEMO**

```
//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);
```

DISP_GET_SCN_WIDTH

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdle 显示驱动句柄;
cmd DISP_GET_SCN_WIDTH;
arg 显示通道 0/1;

➤ **RETURNS**

如果成功, 返回当前屏幕水平分辨率, 否则, 返回失败号;

➤ **DESCRIPTION**

该函数用于获取当前屏幕水平分辨率。

➤ **DEMO**

```
//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);
```

DISP_GET_SCN_HEIGHT

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功, 返回当前屏幕垂直分辨率, 否则, 返回失败号;

➤ **DESCRIPTION**

该函数用于获取当前屏幕垂直分辨率。

➤ **DEMO**

```
//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);
```


DISP_GET_OUTPUT_TYPE

➤ PROTOTYPE

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

➤ ARGUMENTS

hdle 显示驱动句柄;

cmd DISP_GET_OUTPUT_TYPE;

arg arg[0]为显示通道 0/1;

➤ RETURNS

如果成功, 返回当前显示输出类型, 否则, 返回失败号;

➤ DESCRIPTION

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

➤ DEMO

```
//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);
```

DISP_GET_OUTPUT

➤ PROTOTYPE

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

➤ ARGUMENTS

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

➤ RETURNS

如果成功, 返回 0, 否则, 返回失败号;

➤ DESCRIPTION

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

➤ DEMO

```
//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;
```

DISP_VSYNC_EVENT_EN

➤ PROTOTYPE

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

➤ ARGUMENTS

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

➤ RETURNS

如果成功, 返回 DIS_SUCCESS, 否则, 返回失败号;

➤ DESCRIPTION

该函数开启/关闭 vsync 消息发送功能。

➤ **DEMO**

```
//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);
```

DISP_DEVICE_SWITCH

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdlc 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

➤ **RETURNS**

如果成功, 返回 DIS_SUCCESS, 否则, 返回失败号;

➤ **DESCRIPTION**

该函数用于切换输出类型

➤ **DEMO**

```
//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。

8.2. Layer Interface

DISP_LAYER_SET_CONFIG

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdlc 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

➤ **RETURNS**

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ **DESCRIPTION**

该函数用于设置多个图层信息。

➤ **DEMO**

```
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);

```

DISP_LAYER_GET_CONFIG

➤ PROTOTYPE

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

➤ ARGUMENTS

hdle 显示驱动句柄;

cmd DISP_LAYER_GET_INFO

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;

➤ RETURNS

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ DESCRIPTION

该函数用于获取图层参数。

➤ DEMO

```

//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(dispatch_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(dispatch, DISP_LAYER_GET_CONFIG, (void*)arg);

```

8.3. capture interface

DISP_CAPTURE_START

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

➤ **DESCRIPTION**

该函数用于开启截屏功能。

DISP_CAPTURE_STOP

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

➤ **DESCRIPTION**

该函数用于关闭截屏功能。

DISP_CAPTURE_COMMIT

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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;

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

➤ **DESCRIPTION**

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。

8.4. LCD Interface

DISP_LCD_SET_BRIGHTNESS

➤ PROTOTYPE

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

➤ ARGUMENTS

hdlc 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)

➤ RETURNS

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ DESCRIPTION

该函数用于设置 LCD 亮度。

➤ DEMO

```
//设置 LCD 的背光亮度, disphd 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);
```

DISP_LCD_GET_BRIGHTNESS

➤ PROTOTYPE

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

➤ ARGUMENTS

hdlc 显示驱动句柄;
cmd DISP_LCD_GET_BRIGHTNESS
arg arg[0]为显示通道 0/1;

➤ RETURNS

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ DESCRIPTION

该函数用于获取 LCD 亮度。

➤ DEMO

```
//获取 LCD 的背光亮度, 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);
```

8.5. enhance interface

DISP_ENHANCE_ENABLE

➤ PROTOTYPE

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

➤ ARGUMENTS

hdlc 显示驱动句柄;
cmd DISP_ENHANCE_ENABLE
arg arg[0]为显示通道 0/1;

➤ RETURNS

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ **DESCRIPTION**

该函数用于使能图像后处理功能。

➤ **DEMO**

```
//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);
```

DISP_ENHANCE_DISABLE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ **DESCRIPTION**

该函数用于关闭图像后处理功能。

➤ **DEMO**

```
//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);
```

DISP_ENHANCE_DEMO_ENABLE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ **DESCRIPTION**

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。

➤ **DEMO**

```
//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);
```

DISP_ENHANCE_DEMO_DISABLE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果成功, 则返回 DIS_SUCCESS; 如果失败, 则返回失败号。

➤ **DESCRIPTION**

该函数用于关闭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。

➤ **DEMO**

```
//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);
```

8.6. smart backlight

DISP_SMBL_ENABLE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdlc 显示驱动句柄;

cmd DISP_SMBL_ENABLE

arg arg[0]为显示通道 0/1;

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

➤ **DESCRIPTION**

该函数用于使能智能背光功能。

➤ **DEMO**

```
//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);
```

DISP_SMBL_DISABLE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdlc 显示驱动句柄;

cmd DISP_SMBL_DISABLE

arg arg[0]为显示通道 0/1;

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS; 如果失败，则返回失败号。

➤ **DESCRIPTION**

该函数用于关闭智能背光功能。

➤ **DEMO**

```
//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);
```

DISP_SMBL_SET_WINDOW

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdlc 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

➤ **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

➤ **DESCRIPTION**

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。

➤ **DEMO**

```
//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);
```

8.7. hdmi interface

DISP_HDMI_SUPPORT_MODE

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

如果支持，则返回 1；如果失败，则返回 0。

➤ **DESCRIPTION**

该函数用于查询指定的 HDMI 模式是否支持。

➤ **DEMO**

```
//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);
```

DISP_HDMI_GET_HPD_STATUS

➤ **PROTOTYPE**

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

➤ **ARGUMENTS**

hdle 显示驱动句柄；

cmd DISP_HDMI_GET_HPD_STATUS

arg arg[0]为显示通道 0/1；

➤ **RETURNS**

如果 HDMI 插入，则返回 1；如果未插入，则返回 0。

➤ **DESCRIPTION**

该函数用于指定 HDMI 是否处于插入状态。

➤ **DEMO**

```
//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");
```

Yllwinnetec

9. 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;
}
```

9.1. enhance

enhance_mode

> SYSFS NODE

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

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

> ARGUMENTS

disp display channel, 比如 0: disp0, 1: disp1

enhance_mode: enhance mode, 0: standard, 1: enhance, 2: soft

> RETURNS

no

> DESCRIPTION

该接口用于设置色彩增强的模式

> DEMO

```
//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;
```

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));
```

9.2. hdmi edid

edid

➤ **SYSFS NODE**

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

➤ **ARGUMENTS**

no

➤ **RETURNS**

Edid data(1024 bytes)

➤ **DESCRIPTION**

该接口用于读取 EDID 的裸数据

➤ **DEMO**

```
//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);
```

hpd

➤ **SYSFS NODE**

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

➤ **ARGUMENTS**

no

➤ **RETURNS**

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

➤ **DESCRIPTION**

该接口用于读取 HDMI 的热插拔状态

➤ **DEMO**

```
//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");
```

hdcp_enable

➤ **SYSFS NODE**

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

➤ **ARGUMENTS**

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

➤ **RETURNS**

No returns.

➤ **DESCRIPTION**

该接口用于使能、关闭 hdmi hdcp 功能

➤ **DEMO**

```
//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));
```

10.Data Structure

disp_fb_info

➤ PROTOTYPE

```
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;
```

➤ MEMBERS

addr :frame buffer 的内容地址,对于 interleaved 类型,只有 addr[0]有效;planar 类型,三个都有效;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

➤ DESCRIPTION

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

disp_layer_info

➤ PROTOTYPE

```
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;

```

➤ MEMBERS

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

➤ DESCRIPTION

disp_layer_info 用于描述一个图层的属性信息。

disp_layer_config

➤ PROTOTYPE

```

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

```

➤ MEMBERS

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。

➤ DESCRIPTION

disp_layer_config 用于描述一个图层配置的属性信息。

disp_color_info

➤ PROTOTYPE

```

typedef struct
{
    u8 alpha;
    u8 red;
    u8 green;
}

```

```
    u8 blue;
}disp_color_info;
```

➤ **MEMBERS**

alpha :颜色的透明度
red :红
green :绿
blue :蓝

➤ **DESCRIPTION**

disp_color_info 用于描述一个颜色的信息。

disp_rect

➤ **PROTOTYPE**

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

➤ **MEMBERS**

x :起点 x 值
y :起点 y 值
width :宽
height :高

➤ **DESCRIPTION**

disp_rect 用于描述一个矩形窗口的信息。

disp_rect64

➤ **PROTOTYPE**

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

➤ **MEMBERS**

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

➤ **DESCRIPTION**

disp_rect64 用于描述一个矩形窗口的信息。

disp_position

➤ **PROTOTYPE**

```
typedef struct
{
    s32 x;
```

```
    s32    y;
}disp_posistion;
```

➤ **MEMBERS**

```
x        :x
y        :y
```

➤ **DESCRIPTION**

*disp_position*用于描述一个坐标的信息。

disp_rectsz

➤ **PROTOTYPE**

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

➤ **MEMBERS**

```
width    :宽
height   :高
```

➤ **DESCRIPTION**

*disp_rectsz*用于描述一个矩形尺寸的信息。

disp_pixel_format

➤ **PROTOTYPE**

```
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
```


	<i>DISP_FORMAT_YUV422_I_YUYV</i>	= 0x43, //MSB Y-U-Y-V LSB
	<i>DISP_FORMAT_YUV422_I_UYVY</i>	= 0x44, //MSB U-Y-V-Y LSB
	<i>DISP_FORMAT_YUV422_I_VYUY</i>	= 0x45, //MSB V-Y-U-Y LSB
VVVV	<i>DISP_FORMAT_YUV444_P</i>	= 0x46, //MSB P3-2-1-0 LSB, YYYY UUUU
VV	<i>DISP_FORMAT_YUV422_P</i>	= 0x47, //MSB P3-2-1-0 LSB YYYY UU
	<i>DISP_FORMAT_YUV420_P</i>	= 0x48, //MSB P3-2-1-0 LSB YYYY U V
	<i>DISP_FORMAT_YUV411_P</i>	= 0x49, //MSB P3-2-1-0 LSB YYYY U V
	<i>DISP_FORMAT_YUV422_SP_UVUV</i>	= 0x4a, //MSB V-U-V-U LSB
	<i>DISP_FORMAT_YUV422_SP_VUVU</i>	= 0x4b, //MSB U-V-U-V LSB
	<i>DISP_FORMAT_YUV420_SP_UVUV</i>	= 0x4c,
	<i>DISP_FORMAT_YUV420_SP_VUVU</i>	= 0x4d,
	<i>DISP_FORMAT_YUV411_SP_UVUV</i>	= 0x4e,
	<i>DISP_FORMAT_YUV411_SP_VUVU</i>	= 0x4f,

}disp_pixel_format;;

➤ MEMBERS

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, UV 的顺序为 U 在低位, *DISP_FORMAT_YUV420_SP_UVUV* 类似
DISP_FORMAT_YUV422_SP_VUVU: semi-planar yuv 格式, 分两块存放, 需两个地址, UV 的顺序为 V 在低位, *DISP_FORMAT_YUV420_SP_VUVU* 类似

➤ DESCRIPTION

disp_pixel_format 用于描述像素格式。

disp_buffer_flags

➤ PROTOTYPE

```
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;
```

➤ MEMBERS

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

➤ DESCRIPTION

disp_buffer_flags 用于描述 3D 源模式。

disp_3d_out_mode

➤ PROTOTYPE

```
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 interlaved 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 interlaved
DISP_3D_OUT_MODE_FA = 0xa, //field alternative
}disp_3d_out_mode;

```

➤ MEMBERS

```

//for lcd
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
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 interlaved, 行交织
DISP_3D_OUT_MODE_FA : field alternate 场交错

```

➤ DESCRIPTION

disp_3d_out_mode 用于描述 3D 输出模式。

disp_color_space

➤ PROTOTYPE

```

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

```

➤ MEMBERS

```

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

```

➤ DESCRIPTION

disp_color_space 用于描述颜色空间类型。

disp_output_type

➤ PROTOTYPE

```

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;

```

➤ **MEMBERS**

```

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

```

➤ **DESCRIPTION**

disp_output_type 用于描述显示输出类型。

disp_tv_mode

➤ **PROTOTYPE**

```

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;

```

➤ **MEMBERS**

➤ **DESCRIPTION**

disp_tv_mode 用于描述 TV 输出模式。

disp_output

➤ **PROTOTYPE**

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

➤ **MEMBERS**

Type: 输出类型
Mode: 输出模式, 480P/576P, etc.

➤ **DESCRIPTION**

disp_output 用于描述显示输出类型, 模式

disp_layer_mode

➤ **PROTOTYPE**

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

➤ **MEMBERS**

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

➤ **DESCRIPTION**

disp_layer_mode 用于描述图层模式。

disp_scan_flags

➤ **PROTOTYPE**

```
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;
```

➤ **MEMBERS**

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

➤ **DESCRIPTION**

disp_scan_flags 用于描述显示 Buffer 的扫描方式。

