全志科技
Allwinner Technology

H6

显示模块使用说明书

1.0

2017.07.17

文档履历

版本号	日期	制/修订人	内容描述
1.0	2017.07.17	AW	Initial version

目录

1. 前言

1.1 编写目的

1. 让显示应用开发人员了解显示驱动的接口及使用流程，能够快速上手功能开发；
2. 为客户提供显示差异化定制的指导方法；
3. 集合显示模块常见文件的分析、解决方法，帮助 FAE 快速定位问题。

1.2 适用范围

- Android 版本：Android N
- Linux 版本：linux-3.10

1.3 相关人员

- 显示应用开发工程师；
- AllwinnerTech 方案客户；
- FAE

1.4 相关术语

- SurfaceFlinger: 通过 OpenGL/Hwcomposer，负责向系统提供图层合成的服务；
- Hwcomposer: 是 Android 显示框架对显示子系统的抽象 HAL，主要负责图层合成；
- layer: 图层，按照一定数据格式存储的图像数据缓冲区；
- DE: Display Engine，显示引擎。

2. 显示模块配置说明

2.1 sys_config 配置说明

2.1.1 Boot 初始化配置

配置项	配置项含义
disp_init_enable	显示模块使能控制: 1: enable 0: disable
screen0_output_type	显示设备 0 的输出类型: 0: none 1: lcd 2: cvbs 3: hdmi
screen0_output_mode	显示设备 0 的输出模式: 0:480i 1:576i 2:480p 3:576p 4:720p50 5:720p60 6:1080i50 7:1080i60 8:1080p24 9:1080p50 10:1080p60
screen0_output_format	显示设备 0 的图像格式: 0:RGB 1:yuv444 2:yuv422 3:yuv420
screen0_output_bits	显示设备 0 的像素深度: 0:8bit 1:10bit 2:12bit 3:16bit
screen0_output_eotf	显示设备 0 的光电转换函数: 0:reserve 4:SDR 16:HDR10 18:HLG
screen0_output_cs	显示设备 0 的颜色空间: 0:undefined 257:BT709 260:BT601 263:BT2020
fb0_format	FrameBuffer 的数据格式: 0:ARGB 1:ABGR 2:RGBA 3:BGRA
fb0_width	FrameBuffer 的宽度 (pixels)
fb0_height	FrameBuffer 的高度 (pixels)
disp_para_zone	显示参数保存位置: 0: 分区文件形式保存 1:uboot private param zone

Table 1: display config

2.1.2 HDMI 模块配置

配置项	配置项含义
hdmi_used	HDMI 模块使能控制
hdmi_hdcp_enable	HDCP 使能控制
hdmi_cec_support	HDMI CEC 使能控制
hdmi_skip_bootedid	uboot 阶段是否读取 EDID: 0: 读取 1: 不读取

Table 2: hdmi config

2.1.3 CVBS 模块配置

配置项	配置项含义
tv_used	tv 模块使能控制
tv_module_name	驱动模块名字
tv_twi_used	AC200 是否使用 IIC: 0: 不使用 1: 使用
tv_twi_id	AC200 使用的 IIC master index
tv_twi_addr	AC200 的 IIC 设备地址
tv_pwm_ch	该模块使用的 pwm channel
tv_clk_dir	该模块时钟源分频因子
tv_regulator_name	该模块使用的 regulator 的名称

Table 3: cvbs config

2.2 方案差异化配置说明

2.2.1 多显策略配置说明

DisplayManagerPolicy2 类实现了显示的热插拔策略。DisplayManagerPolicy2 使用 HDMI 热插拔消息对 HDMI 和 CVBS 信号输出进行切换。如需实现自己的策略，可在 DisplayManagerPolicy2 修改对应的方法。

热插拔消息处理策略			
一. 显示通道选择（sys_config.fex可配置）：HDMI-disp0，CVBS-disp1。			
二. 优先级：HDMI-高，CVBS-低			
三. 默认输出：hdmi-720P50HZ，cvbs-PAL。			
四. 策略：			
序号	当前状态	具体操作	Android主辅显状态
1	只有主显设备	-	主显输出到此设备
2	只有主显设备	拔掉此设备	主显仍输出到此设备（Android认为设备还在）
3	只有主显设备	插入另一个设备	**打开辅显输出到第二个插入设备，如HDMI为辅显设备，则切换主辅显设备**
4	同时插着两个设备	-	HDMI的为主显设备，CVBS为辅显设备（参考3）
5	同时插着两个设备	拔掉主显设备（HDMI）	**主显无缝切换到辅显设备，再关掉前主显设备和辅显**
6	同时插着两个设备	拔掉辅显设备（CVBS）	关掉辅显设备和辅显
7	无设备	-	主显输出到最后拔出的设备（参考2）
8	无设备	插入主显设备	主显输出到此设备
9	无设备	插入非主显设备	主显输出到新插入设备
10	休眠唤醒	-	显示驱动：休眠时，不修改HPD状态属性节点，故Android无处理。唤醒后根据当前状态设置HPD状态属性节点。
11	休眠前，只有主显设备	休眠时换显示设备，唤醒	**参考（1和9）或（3和5）**
12	休眠前，只有主显设备	休眠时插辅显设备，唤醒	参考3
13	休眠前，只有主显设备	休眠时拔主显设备，唤醒	参考2
14	休眠前，插着两个设备	休眠时拔主显设备，唤醒	参考5
15	休眠前，插着两个设备	休眠时拔辅显设备，唤醒	参考6
16	休眠前，插着两个设备	休眠时拔两个设备，唤醒	**参考（5和2）或（6和2）**
17	休眠前，无设备	休眠时插一设备，唤醒	**参考8或9**
18	休眠前，无设备	休眠时插两设备，唤醒	**参考（8和3）或（9和3）**
19	正在显示	setting切换显示模式	支持切换到电视支持的显示模式

图 1: 双显显示的热插拔消息处理策略

2.2.2 横屏竖显配置方法

详见本文档的第 6 章。

2.2.3 默认切边配置

一般的电视显示存在画面溢出的现象，为了解决这种现象，显示系统支持切边功能(也称做 overscan)。配置默认切边值的方法如下:

```
# 在方案下的 mk 文件（例如 device\softwinner\petrel-p1\petrel_fvd_p1.mk）配置宏:
MARGIN_DEFAULT_PERCENT_WIDTH  := 95  //配置横向切边为 95%.
MARGIN_DEFAULT_PERCENT_HEIGHT := 95  //配置纵向切边为 95%.
```

2.2.4 显示系统 density 配置

显示系统的 density 值会影响系统 UI 画面的布局，其值越大，图片布局也越大。修改 density 的方法如下:

```
在方案下的 mk 文件（例如 device\softwinner\petrel-p1\petrel_fvd_p1.mk）配置属性:
persist.sys.disp_density=160 //配置 density 值为 160
```


3. Android 显示框架

3.1 Android 显示接口说明

Android/framework/base/core/java/android/os/DisplayOutputManager.java 类是 Android N 框架层的显示管理类，它向 APK 提供统一的接口对显示设备和显示方式进行操作。DisplayManager 提供的功能接口主要有：

- 设置和获取显示模式；
- 设置 3D 视频播放模式（视频格式包括：3D 左右，3D 上下，双流 3D）；
- 设置和获取横向缩放和纵向缩放；
- 获取电视支持的制式信息。

3.2 接口描述

3.2.1 Display Mode Interface

public int getDisplayOutput(int display)

- ARGUMENTS

  @display: 目标设备类型
  Display.TYPE_UNKNOWN = 0
  Display.TYPE_BUILT_IN = 1

- RETURNS

  返回 显示类型 + 显示模式。格式：bit15~8: disp_type， bit7~0: disp_mode.

- DESCRIPTION

获取当前的显示模式

- DEMO

```
DisplayOutputManager mDm;
int format;
mDm = (DisplayOutputManager)mCtx.getSystemService(
        Context.DISPLAYOUTPUT_SERVICE);
format = mDm.getDisplayOutput(Display.TYPE_BUILT_IN);
```

public int getDisplayOutputType(int display)

- ARGUMENTS

@display: 目标设备类型
Display.TYPE_BUILT_IN = 1

- RETURNS

返回 显示设备类型

- DESCRIPTION

获取当前的显示设备输出类型

- DEMO

```
DisplayOutputManager mDm;
int disp_type;
mDm = (DisplayOutputManager)mCtx.getSystemService(
        Context.DISPLAYOUTPUT_SERVICE);
disp_type = mDm.getDisplayOutputType(Display.TYPE_BUILT_IN);
```

public int getDisplayOutputMode(int display)

- ARGUMENTS

  @display: 目标显示设备类型
  Display.TYPE_BUILT_IN = 1

- RETURNS

  返回 当前显示输出分辨率模式

- DESCRIPTION

  获取当前的显示分辨率模式

- DEMO

```
DisplayOutputManager mDm;
int mode
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mode = mDm.getDisplayOutputMode(Display.TYPE_BUILT_IN);
```

public int setDisplayOutputMode(int display, int mode)

- ARGUMENTS

  @display: 目标显示设备
  Display.TYPE_BUILT_IN = 1
  @mode: 输出模式

- RETURNS

  成功返回 0，设备返回-1

- DESCRIPTION

设置显示模式

- DEMO

```
DisplayOutputManager mDm;
int type = DISPLAY_OUTPUT_TYPE_HDMI;
int mode = DISPLAY_TVFORMAT_1080P_60HZ;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayOutputMode(Display.TYPE_BUILT_IN, mode);
```

public int setDisplayOutput(int display, int format)

- ARGUMENTS

@display: 目标显示设备
Display.TYPE_BUILT_IN = 1
@format: bit15~8: disp_type， bit7~0: disp_mode.

- RETURNS

成功返回 0，设备返回-1

- DESCRIPTION

设置显示输出类型、模式

- DEMO

```
DisplayOutputManager mDm;
int format = (DISPLAY_OUTPUT_TYPE_HDMI << 8)
        | DISPLAY_TVFORMAT_1080P_60HZ;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayOutput(Display.TYPE_BUILT_IN, format);
```


public int getDisplayOutputPixelFormat(int display)

- ARGUMENTS

  @display: 目标显示设备类型
  Display.TYPE_BUILT_IN = 1

- RETURNS

  返回 当前显示输出的 video foramt 以及 色深 (depth)

- DESCRIPTION

  获取当前显示输出的 video foramt 以及 色深 (depth)

- DEMO

```
DisplayOutputManager mDm;
int pixelformat;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
pixelformat = mDm.getDisplayOutputPixelFormat(Display.TYPE_BUILT_IN);
```

public int setDisplayOutputPixelFormat(int display, int format)

- ARGUMENTS

  @display: 目标显示设备
  Display.TYPE_BUILT_IN = 1
  @format: video format | depth
  0 - DISPLAY_OUTPUT_FORMAT_AUTO
  1 - DISPLAY_OUTPUT_FORMAT_YUV422_10Bit
  2 - DISPLAY_OUTPUT_FORMAT_YUV420_10Bit

3 - DISPLAY_OUTPUT_FORMAT_YUV444_8Bit
4 - DISPLAY_OUTPUT_FORMAT_YUV888_8Bit

- RETURNS

成功返回 0，设备返回-1

- DESCRIPTION

设置显示输出的 video format 以及 depth

- DEMO

```
DisplayOutputManager mDm;
int format = DISPLAY_OUTPUT_FORMAT_YUV420_10Bit;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayOutputPixelFormat(Display.TYPE_BUILT_IN, format);
```

public int getDisplayOutputDataspaceMode(int display)

- ARGUMENTS

@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1

- RETURNS

返回 当前显示 dataspace 模式：SDR/HDR/AUTO

- DESCRIPTION

获取当前显示 dataspace 模式：SDR/HDR/AUTO

- DEMO


```
DisplayOutputManager mDm;
int dataspace;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
dataspace = mDm.getDisplayOutputDataspaceMode(Display.TYPE_BUILT_IN);
```

public int setDisplayOutputDataspaceMode(int display, int mode)

- ARGUMENTS

```
@display: 目标显示设备
Display.TYPE_BUILT_IN = 1
@mode: dataspace mode
    0 - DISPLAY_OUTPUT_DATASPACE_MODE_AUTO
    1 - DISPLAY_OUTPUT_DATASPACE_MODE_HDR
    2 - DISPLAY_OUTPUT_DATASPACE_MODE_WCG
    3 - DISPLAY_OUTPUT_DATASPACE_MODE_SDR
    4 - DISPLAY_OUTPUT_DATASPACE_MODE_OTHER
```

- RETURNS

```
成功返回 0，设备返回-1
```

- DESCRIPTION

```
设置显示输出的 dataspace mode
```

- DEMO

```
DisplayOutputManager mDm;
int mode = DISPLAY_OUTPUT_DATASPACE_MODE_SDR;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayOutputDataspaceMode(Display.TYPE_BUILT_IN, mode);
```


public boolean isSupportHdmiMode(int displaytype, int mode)

- ARGUMENTS

  @displaytype: 目标显示设备类型
  Display.TYPE_BUILT_IN = 1
  @mode: 显示模式

- RETURNS

  返回：当前电视支持该 mode，则返回 true；不支持则返回 false。

- DESCRIPTION

  设置显示模式。

- DEMO

```
DisplayOutputManager mDm;
int ret = 0;
int mode = DISPLAY_TVFORMAT_1080P_60HZ;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
ret  = mDm.isSupportHdmiMode(Display.TYPE_BUILT_IN,  mode);
```

public int[] getSupportModes(int display, int type)

- ARGUMENTS

  @display: 目标显示设备类型
  Display.TYPE_BUILT_IN = 1
  @type: 显示输出类型

- RETURNS

返回：当前电视所有支持的 mode。

- DESCRIPTION

返回当前所有支持的模式

- DEMO

None

3.2.2 Display Margin Interface

public int[] getDisplayMargin(int display)

- ARGUMENTS

@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1

- RETURNS

@ int[0]: Percent of horizen.
@ int[1]: Percent of vertical.

- DESCRIPTION

获取屏幕显示画面的纵向和横向的缩放比例。

- DEMO

```
DisplayOutputManager mDm;
int percents[2];
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
```



```
percents = mDm.getDisplayMargin(Display.TYPE_BUILT_IN);
```

public int setDisplayMargin(int display, int hpercent, int vpercent)

- ARGUMENTS

```
@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
@ hpercent: Percent of horizen.
@ vpercent: Percent of vertical.
```

- RETURNS

```
成功返回 0，失败返回-1
```

- DESCRIPTION

```
设置屏幕显示画面的纵向和横向的缩放比例
```

- DEMO

```
DisplayOutputManager mDm;
int hpercent = 95;
int vpercent = 96;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayMargin(Display.TYPE_BUILT_IN, hpercent, vpercent);
```


3.2.3 Display 3D Interface

public int getDisplaySupport3DMode(int display)

- ARGUMENTS


```
@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
```

- RETURNS

```
显示设备支持 3D，则返回 1；否则返回 0。
```

- DESCRIPTION

```
查询显示设备是否 3D 模式
```

- DEMO

```
DisplayOutputManager mDm;
int is3Dsupport;
mDm = (DisplayOutputManager)mCtx.getSystemService(
        Context.DISPLAYOUTPUT_SERVICE);
is3Dsupport = mDm.getDisplaySupport3DMode(Display.TYPE_BUILT_IN);
```

public int setDisplay3DMode(int display, int mode)

- ARGUMENTS

```
@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
@mode: (see DisplayManager.java)
DISPLAY_2D_ORIGINAL          = 0;
DISPLAY_2D_LEFT              = 1;
DISPLAY_2D_TOP               = 2;
DISPLAY_3D_LEFT_RIGHT_HDMI   = 3;
DISPLAY_3D_TOP_BOTTOM_HDMI   = 4;
DISPLAY_2D_DUAL_STREAM       = 5;
DISPLAY_3D_DUAL_STREAM       = 6;
```


- RETURNS

  成功返回 0，失败返回-1.

- DESCRIPTION

  设置视频的 3D 模式.

- DEMO

```
DisplayOutputManager mDm;
int trdmode = DISPLAY_3D_LEFT_RIGHT_HDMI;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplay3DMode(Display.TYPE_BUILT_IN, trdmode);
```

3.2.4 Display Color Interface

public int getDisplayBright(int display)

- ARGUMENTS

  @display: 目标显示设备类型
  Display.TYPE_BUILT_IN = 1

- RETURNS

  返回 显示画面的亮度值

- DESCRIPTION

  获取当前的显示画面的亮度

- DEMO


```
DisplayOutputManager mDm;
int bright;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
bright = mDm.getDisplayBright(Display.TYPE_BUILT_IN);
```

public int setDisplayBright(int display, int bright)

- ARGUMENTS

```
@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
@bright: the value of bright
```

- RETURNS

```
成功返回 0，失败返回-1.
```

- DESCRIPTION

```
设置显示画面的亮度
```

- DEMO

```
DisplayOutputManager mDm;
int bright = 50;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayBright(Display.TYPE_BUILT_IN, bright);
```

public int getDisplayContrast(int display)

- ARGUMENTS

@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1

- RETURNS

返回显示画面的对比度值

- DESCRIPTION

获取当前的显示画面的对比度

- DEMO

```
DisplayOutputManager mDm;
int contrast;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
contrast = mDm.getDisplayContrast(Display.TYPE_BUILT_IN);
```

public int setDisplayContrast(int display, int contrast)

- ARGUMENTS

@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
@contrast : the value of contrast

- RETURNS

成功返回 0，失败返回-1。

- DESCRIPTION

设置显示画面的对比度

- DEMO

```
DisplayOutputManager mDm;
int contrast = 50;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplayContrast(Display.TYPE_BUILT_IN, contrast);
```

public int getDisplaySaturation(int display)

- ARGUMENTS

```
@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
```

- RETURNS

```
返回 显示画面的饱和度值
```

- DESCRIPTION

```
获取当前的显示画面的饱和度
```

- DEMO

```
DisplayOutputManager mDm;
int saturation;
mDm = (DisplayOutputManager)mCtx.getSystemService(
            Context.DISPLAYOUTPUT_SERVICE);
saturation = mDm.getDisplaySaturation(Display.TYPE_BUILT_IN);
```

public int setDisplaySaturation(int display, int saturation)

- ARGUMENTS

@display: 目标显示设备类型
Display.TYPE_BUILT_IN = 1
@saturation: the value of saturation

- RETURNS

成功返回 0，失败返回-1。

- DESCRIPTION

设置显示画面的饱和度。

- DEMO

```
DisplayOutputManager mDm;
int saturation= 50;
mDm = (DisplayOutputManager)mCtx.getSystemService(
                Context.DISPLAYOUTPUT_SERVICE);
mDm.setDisplaySaturation(Display.TYPE_BUILT_IN, saturation);
```


4. display 驱动框架

4.1 模块功能介绍

显示驱动特性如下:

- 支持 linux 标准的 framebuffer 接口
- 支持多图层叠加混合处理
- 支持多种显示效果处理（alpha, colorkey, 图像细节增强）
- 支持丽色系统
- 支持多种图像数据格式输入 (ARGB/RGB/YUV)
- 支持图像缩放处理
- 支持多种制式的 HDMI 输出，包括: 480P@60Hz、576P@50Hz、720P@50Hz、720P@60Hz、1080I@50Hz、1080I@60Hz、1080P@50Hz、1080P@60Hz、4K@30Hz、4K@60Hz 等
- 支持 CVBS 信号输出（PAL 和 NTSC）

4.2 图层操作说明

显示驱动中最重要的显示资源为图层，sunxi 平台 soc 支持两路显示通道；0 路显示支持 16 个图层（其中视频图层 4 个），3 个 blending 通道；1 路支持 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。

对图层的操作按照以下步骤:

- 图层以 disp, channel, layer_id 三个索引唯一确定;
- 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;

4.3 接口参数说明

相关参数	备注说明
图层标识	以 disp, chennel, 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
图层 size	每个分量都需要设置 1 个 size
图层 align	针对每个分量需要设置其 align 位数，为 2 的倍数
图层 Crop	为 64 位参数，高 32 位为整数，低 32 位为小数
YUV MB 格式支持	不支持
PALETTE 格式支持	不支持
单色模式 (无 buffer)	支持
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的参数

4.4 图层主要参数

4.4.1 size 与 src_win

Fb 有两个与 size 有关的参数，分别是 size 与 src_win。Size 表示 buffer 的完整尺寸，src_win 则表示 buffer 中需要显示的一个矩形窗口。如下图所示，完整的图像以 size 标识，而矩形框住的部分为需要显示的部分，以 src_win 标识，在屏幕上只能看到 src_win 标识的部分，其余部分是隐藏的，不能在屏幕上显示出来的。

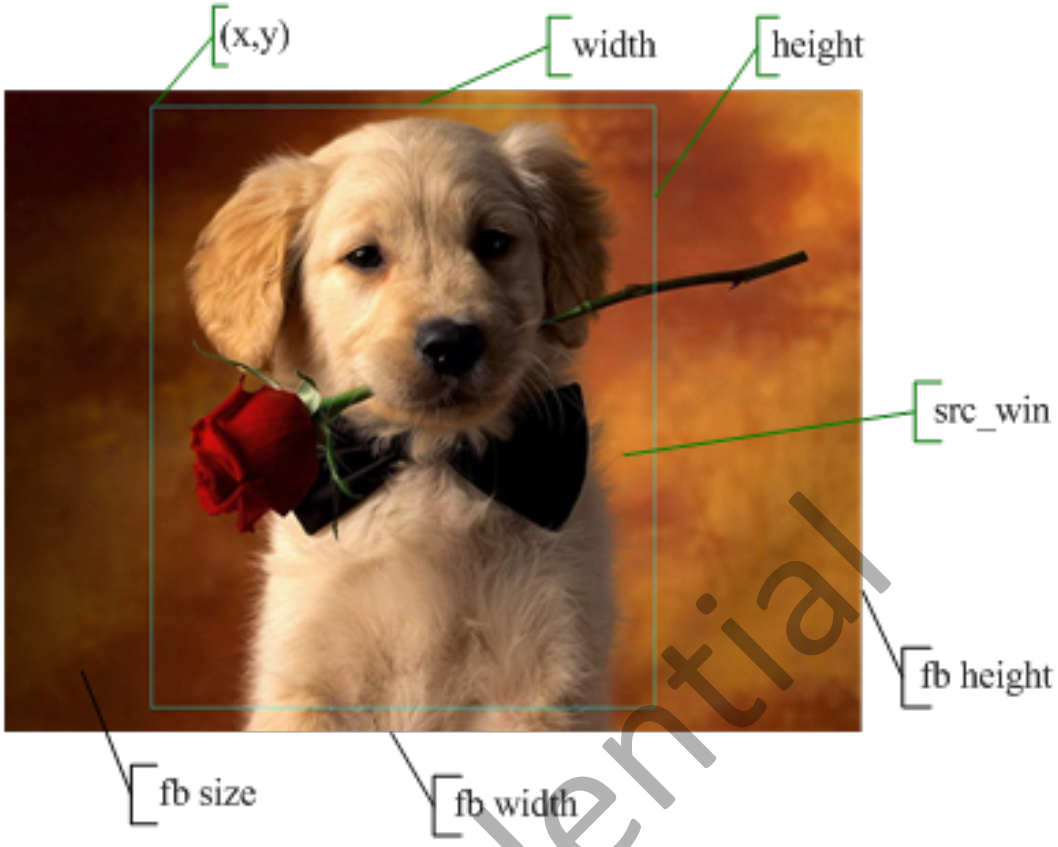

图 2: source window

4.4.2 src_win 和 screen_win

src_win 上面已经介绍过了。screen_win 为 src_win 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话，src_win 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，src_win 和 screen_win 的 width,height 可以不等。

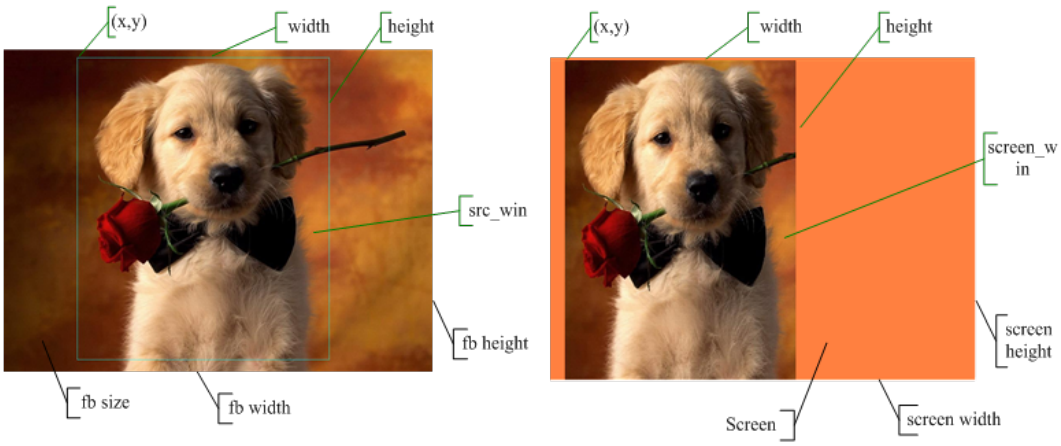

图 3: screen window

4.4.3 alpha

Alpha 模式有三种:

- Gloabal 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: 可以是说以上两种效果的叠加，在实现 pxiel alpha 的效果的同时，还可以做淡入浅出的效果。

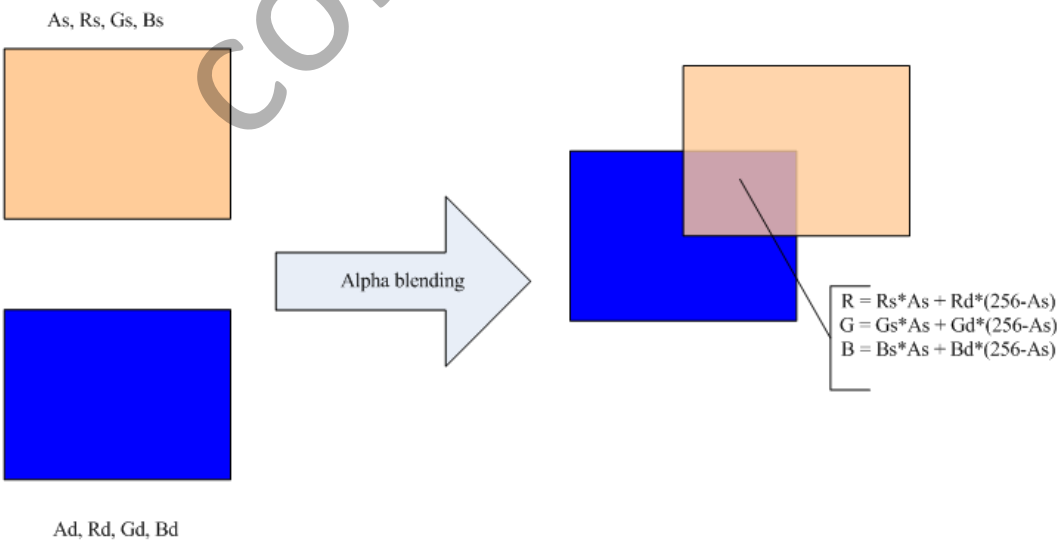

图 4: blending description

4.4.4 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
```


4.5 输出设备介绍

平台支持 HDMI 和 CVBS 两种输出接口。

4.5.1 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，模式查询。

4.5.2 CVBS

驱动支持 PAL 制式或 NTSC 制式信号输出。

4.6 驱动接口使用说明

sunxi 平台显示驱动向用户提供了众多功能接口，可对图层、HWC、hdmi、cvbs 等显示资源进行操作。

4.6.1 Global Interface

4.6.1.1 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保存起来, 等到调用 DISP_SHADOW_PROTECT（0）后将一起执行。

- DEMO

```
//启动 cache, disable 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 somthing other
arg[1] = 0;//unprotect
ioctl(disphd, DISP_SHADOW_PROTECT, (void*)arg);
```


4.6.1.2 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);
```


4.6.1.3 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);
```


4.6.1.4 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);
```

4.6.1.5 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);
```

4.6.1.6 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);
```

4.6.1.7 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;
```

4.6.1.8 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);
```

4.6.1.9 DISP_DEVICE_SWITCH

- PROTOTYPE

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

- ARGUMENTS

```
@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;
```

- 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。

4.6.1.10 DISP_DEVICE_SET_CONFIG

- PROTOTYPE

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

- ARGUMENTS

```
@hdle: 显示驱动句柄;
@cmd : DISP_DEVICE_SET_CONFIG;
@arg : arg[0] 为显示通道 0/1;
  arg[1] 设备输出属性配置;
```

- 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设备的 type/mode/color space/eotf 等信息
```

- DEMO

略。

4.6.1.11 DISP_DEVICE_GET_CONFIG

- PROTOTYPE


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

- ARGUMENTS

```
@hdle: 显示驱动句柄;
@cmd : DISP_DEVICE_SET_CONFIG;
@arg : arg[0] 为显示通道 0/1;
  arg[1] 设备输出属性配置;
```

- RETURNS

```
如果成功，返回 DIS_SUCCESS，否则，返回失败号;
```

- DESCRIPTION

```
该函数用于获取输出设备的 type/mode/color space/eotf 等信息
```

- DEMO

略。

4.6.2 Layer Interface

4.6.2.1 DISP_LAYER_SET_CONFIG

- PROTOTYPE

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

- ARGUMENTS

```
@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.

- 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;
  config.info.fb.crop.height= ((uunsigned long)height)<<32;
  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);
```


4.6.2.2 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(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);
```


4.6.2.3 DISP_LAYER_SET_CONFIG2

- PROTOTYPE

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

- ARGUMENTS

@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.

- 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

略。

4.6.2.4 DISP_LAYER_GET_CONFIG2

- 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

  略。

4.6.3 HDMI Interface

4.6.3.1 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);
```

4.6.4 enhance

4.6.4.1 DISP_ENHANCE_ENABLE

- PROTOTYPE

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

- ARGUMENTS

```
@hdle: 显示驱动句柄;
@cmd : DISP_ENHANCE_ENABLE;
@arg : arg[0] 为显示通道 0/1;
```

- RETURNS

```
如成功，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);
```

4.6.4.2 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);
```

4.6.4.3 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);
```

4.6.4.4 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);
```


4.7 Data Struct

4.7.1 disp_fb_info

功能：用于描述一个 display frambuffer 的属性信息

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

Table 5: struct disp_fb_info

4.7.2 disp_fb_info2

功能：用于描述一个 display frambuffer 的属性信息，相对 disp_fb_info 增加对 fbc/hdr 的支持。

成员	类型	描述
fd	int	dma_buf fd for frame buffer
size[3]	disp_rectsz	size for 3 component, unit: pixels
align[3]	unsigned int	align for 3 comonent, unit: bits(align=2n̂,i.e. 1/2/4/8/16/32..)
format	disp_pixel_format	pixel format
color_space	disp_color_space	color space
trd_right_fd	int	dma_buf fd for the right-eye frame buffer used when in frame packing 3d mode
pre_multiply	bool	true: pre-multiply fb
crop	disp_rect64	crop rectangle boundaries
flags	disp_buffer_flags	indicate stereo or non-stereo buffer
scan	disp_scan_flags	scan type & scan order
eotf	disp_eotf	depth perception for stereo image only valid when stereo image input
depth	int	electro-optical transfer function
metadata_fd	int	dma_buf fd for contained metadata for fbc/hdr buffer
metadata_size	unsigned int	size of metadata buffer
metadata_flag	unsigned int	type of metadata buffer

Table 6: struct disp_fb_info2

4.7.3 disp_layer_info

功能: 用于描述一个图层的属性信息

成员	类型	描述
mode	disp_layer_mode	图层的模式，详见 disp_layer_mode
zorder	unsigned char	layer zorder, 优先级高的图层可能会覆盖优先级低的图层
alpha_mode	unsigned char	0:pixel alpha, 1:global alpha, 2:global pixel alpha
alpha_value	unsigned char	layer global alpha value，valid while alpha_mode(1/2)
screen_win	disp_rect	screen window，图层在屏幕上显示的矩形窗口
b_trd_out	bool	if output in 3d mode,used for scaler layer
out_trd_mode	disp_3d_out_mode	output 3d mode, 详见 disp_3d_out_mode
color	unsigned int	display color, valid when COLOR_MODE
fb	disp_fb_info	framebuffer 的属性, 详见 disp_fb_info, valid when BUFFER_MODE
id	unsigned int	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 以做特定的处理，比如释放掉已经显示完成的图像帧 buffer

Table 7: struct disp_layer_info

4.7.4 disp_layer_info2

功能: 用于描述一个图层的属性信息

成员	类型	描述
mode	disp_layer_mode	图层的模式，详见 disp_layer_mode
zorder	unsigned char	layer zorder, 优先级高的图层可能会覆盖优先级低的图层
alpha_mode	unsigned char	0:pixel alpha, 1:global alpha, 2:global pixel alpha
alpha_value	unsigned char	layer global alpha value，valid while alpha_mode(1/2)
screen_win	disp_rect	screen window，图层在屏幕上显示的矩形窗口
b_trd_out	bool	if output in 3d mode,used for scaler layer
out_trd_mode	disp_3d_out_mode	output 3d mode, 详见 disp_3d_out_mode
color	unsigned int	display color, valid when COLOR_MODE
fb	disp_fb_info2	framebuffer 的属性, 详见 disp_fb_info2, valid when BUFFER_MODE
id	unsigned int	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 以做特定的处理，比如释放掉已经显示完成的图像帧 buffer
atw	disp_atw_info	asyncronous time wrap information

Table 8: struct disp_layer_info2

4.7.5 disp_layer_config

功能: 用于描述一个图层配置的属性信息

成员	类型	描述
info	disp_layer_info	图像的信息属性
enable	bool	使能标志
channel	unsigned int	图层所在的通道 id（0/1/2/3）
layer_id	unsigned int	图层的 id，此 id 是在通道内的图层 id (channel,layer_id)=(0,0) 表示通道 0 中的图层 0 之意

Table 9: struct disp_layer_config

4.7.6 disp_layer_config2

功能: 用于描述一个图层配置的属性信息

成员	类型	描述
info	disp_layer_info2	图像的信息属性
enable	bool	使能标志
channel	unsigned int	图层所在的通道 id（0/1/2/3）
layer_id	unsigned int	图层的 id，此 id 是在通道内的图层 id (channel,layer_id)=(0,0) 表示通道 0 中的图层 0 之意

Table 10: struct disp_layer_config2

4.7.7 disp_color_info

功能: 用于描述一个颜色的信息

成员	类型	描述
alpha	u8	颜色的透明度
red	u8	红色分量
green	u8	绿色分量
blue	u8	蓝色分量

Table 11: struct disp_color_info

4.7.8 disp_rect

功能: 用于描述一个矩形窗口的信息

成员	类型	描述
x	s32	起点 x 值
y	s32	起点 y 值
width	s32	宽
height	s32	高

Table 12: struct disp_rect

4.7.9 disp_position

功能: 用于描述一个坐标的信息

成员	类型	描述
x	s32	坐标 x 的值
y	s32	坐标 y 的值

Table 13: struct disp_position

4.7.10 disp_rectsz

功能: 用于描述一个矩形尺寸的信息

成员	类型	描述
width	s32	矩形的宽
height	s32	矩形的高

Table 14: struct disp_rectsz

4.7.11 disp_pixel_format

功能: 像素格式枚举值

成员	类型	描述
DISP_FORMAT_ARGB_8888	enum	像素格式枚举值
DISP_FORMAT_ABGR_8888	enum	像素格式枚举值
DISP_FORMAT_RGBA_8888	enum	像素格式枚举值
DISP_FORMAT_BGRA_8888	enum	像素格式枚举值
DISP_FORMAT_XRGB_8888	enum	像素格式枚举值
DISP_FORMAT_XBGR_8888	enum	像素格式枚举值
DISP_FORMAT_RGBX_8888	enum	像素格式枚举值
DISP_FORMAT_BGRX_8888	enum	像素格式枚举值
DISP_FORMAT_RGB_888	enum	像素格式枚举值
DISP_FORMAT_BGR_888	enum	像素格式枚举值
DISP_FORMAT_RGB_565	enum	像素格式枚举值
DISP_FORMAT_BGR_565	enum	像素格式枚举值
DISP_FORMAT_ARGB_4444	enum	像素格式枚举值
DISP_FORMAT_ABGR_4444	enum	像素格式枚举值
DISP_FORMAT_RGBA_4444	enum	像素格式枚举值
DISP_FORMAT_BGRA_4444	enum	像素格式枚举值
DISP_FORMAT_ARGB_1555	enum	像素格式枚举值
DISP_FORMAT_ABGR_1555	enum	像素格式枚举值
DISP_FORMAT_RGBA_5551	enum	像素格式枚举值
DISP_FORMAT_BGRA_5551	enum	像素格式枚举值
DISP_FORMAT_YUV444_I_AYUV	enum	像素格式枚举值
DISP_FORMAT_YUV444_I_VUYA	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_YVYU	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_YUYV	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_UYVY	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_VYUY	enum	像素格式枚举值
DISP_FORMAT_YUV444_P	enum	像素格式枚举值
DISP_FORMAT_YUV422_P	enum	像素格式枚举值
DISP_FORMAT_YUV420_P	enum	像素格式枚举值
DISP_FORMAT_YUV411_P	enum	像素格式枚举值
DISP_FORMAT_YUV422_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV422_SP_VUVU	enum	像素格式枚举值
DISP_FORMAT_YUV420_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV420_SP_VUVU	enum	像素格式枚举值
DISP_FORMAT_YUV411_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV411_SP_VUVU	enum	像素格式枚举值

Table 15: struct disp_pixel_format

4.7.12 disp_buffer_flags

功能: 用于描述 3D 源格式

成员	类型	描述
DISP_BF_NORMAL	enum	非 3D 格式
DISP_BF_STEREO_TB	enum	top bottom 模式
DISP_BF_STEREO_FP	enum	framepacking
DISP_BF_STEREO_SSH	enum	side by side full, 左右全景
DISP_BF_STEREO_SSF	enum	side by side half, 左右半景
DISP_BF_STEREO_LI	enum	line interleaved, 行交错模式

Table 16: struct disp_buffer_flags

4.7.13 disp_3d_out_mode

功能: 用于描述 3D 输出模式

成员	类型	描述
DISP_3D_OUT_MODE_CI_1	enum	列交织 1
DISP_3D_OUT_MODE_CI_2	enum	列交织 2
DISP_3D_OUT_MODE_CI_3	enum	列交织 3
DISP_3D_OUT_MODE_CI_4	enum	列交织 4
DISP_3D_OUT_MODE_LIRGB	enum	列交织 rgb
DISP_3D_OUT_MODE_TB	enum	top bottom 上下模式
DISP_3D_OUT_MODE_FP	enum	framepacking
DISP_3D_OUT_MODE_SSF	enum	side by side full, 左右全景
DISP_3D_OUT_MODE_SSH	enum	side by side half, 左右半景
DISP_3D_OUT_MODE_LI	enum	line interleaved, 行交织
DISP_3D_OUT_MODE_LA	enum	field alternate 场交错

Table 17: struct disp_3d_output_mode

4.7.14 disp_color_space

功能: 用于描述颜色空间类型

成员	类型	描述
DISP_BT601	enum	用于标清视频
DISP_BT709	enum	用于高清视频
DISP_YCC	enum	用于图片

Table 18: struct disp_color_space

4.7.15 disp_output_type

功能: 用于描述显示输出类型

成员	类型	描述
DISP_OUTPUT_TYPE_NONE	enum	无显示输出
DISP_OUTPUT_TYPE_LCD	enum	LCD 输出
DISP_OUTPUT_TYPE_TV	enum	TV 输出
DISP_OUTPUT_TYPE_HDMI	enum	HDMI 输出
DISP_OUTPUT_TYPE_VGA	enum	VGA 输出

Table 19: struct disp_output_type

4.7.16 disp_tv_mode

功能: 用于 TV 输出模式

成员	类型	描述
DISP_TV_MOD_480I	enum	480I 输出格式
DISP_TV_MOD_576I	enum	576I 输出格式
DISP_TV_MOD_480P	enum	480P 输出格式
DISP_TV_MOD_576P	enum	576P 输出格式
DISP_TV_MOD_720P_50HZ	enum	720P 50Hz 输出格式
DISP_TV_MOD_720P_60HZ	enum	720P 60Hz 输出格式
DISP_TV_MOD_1080I_50HZ	enum	1080I 50Hz 输出格式
DISP_TV_MOD_1080I_60HZ	enum	1080I 60Hz 输出格式
DISP_TV_MOD_1080P_24HZ	enum	1080P 24HZ 输出格式
DISP_TV_MOD_1080P_50HZ	enum	1080P 50HZ 输出格式
DISP_TV_MOD_1080P_60HZ	enum	1080P 60HZ 输出格式
DISP_TV_MOD_1080P_24HZ_3D_FP	enum	1080P 24HZ 3D FP 输出格式
DISP_TV_MOD_720P_50HZ_3D_FP	enum	720P 50HZ 3D FP 输出格式
DISP_TV_MOD_720P_60HZ_3D_FP	enum	720P 60HZ 3D FP 输出格式
DISP_TV_MOD_1080P_25HZ	enum	1080P 25HZ 输出格式
DISP_TV_MOD_1080P_30HZ	enum	1080P 30HZ 输出格式
DISP_TV_MOD_PAL	enum	PAL 输出格式
DISP_TV_MOD_PAL_SVIDEO	enum	PAL SVIDEO 输出格式
DISP_TV_MOD_NTSC	enum	NTSC 输出格式
DISP_TV_MOD_NTSC_SVIDEO	enum	SVIDEO 输出格式
DISP_TV_MOD_PAL_M	enum	PAL M 输出格式
DISP_TV_MOD_PAL_M_SVIDEO	enum	SVIDEO 输出格式
DISP_TV_MOD_PAL_NC	enum	PAL NC 输出格式
DISP_TV_MOD_PAL_NC_SVIDEO	enum	PAL NC SVIDEO 输出格式
DISP_TV_MOD_3840_2160P_30HZ	enum	3840 2160P 30HZ 输出格式
DISP_TV_MOD_3840_2160P_25HZ	enum	3840 2160P 25HZ 输出格式
DISP_TV_MOD_3840_2160P_24HZ	enum	3840 2160P 24HZ 输出格式
DISP_TV_MODE_NUM	enum	未定义输出

Table 20: struct disp_tv_mode

4.7.17 disp_output

功能: 用于描述显示输出类型，模式

成员	类型	描述
type	unsigned int	输出类型
mode	unsigned int	输出模式，480P/576P, etc

Table 21: struct disp_output

4.7.18 disp_layer_mode

功能: 用于描述图层模式

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

Table 22: struct disp_layer_mode

4.7.19 disp_scan_flags

功能: 用于描述图层模式

成员	类型	描述
DISP_SCAN_PROGRESSIVE	enum	逐行模式
DISP_SCAN_INTERLACED_ODD_FLD_FIRST	enum	隔行模式，奇数行优先
DISP_SCAN_INTERLACED_EVEN_FLD_FIRST	enum	隔行模式，偶数行优先

Table 23: struct disp_scan_flags

4.7.20 disp_device_config

成员	类型	描述
type	disp_output_type	输出模式
mode	disp_tv_mode	输出类型
format	disp_csc_type	YUV444/YUV422/YUV420
bits	disp_data_bits	色深：8bit/10bit/12bit
eotf	disp_eotf	光电转换
cs	disp_color_space	颜色空间

Table 24: struct disp_device_config

5. 调试说明

5.1 如何读取驱动图层信息

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

5.2 如何查看 Android 系统图层信息

```
# dumpsys SurfaceFlinger
```

```
# setprop debug.hwc.showfps 2
# logcat -s Hwcomposer
```

5.3 如何判断设备是否插入

```
#
# cat /sys/class/switch/hdmi/state
#
1: 表示 HDMI 插入
0: 表示 HDMI 未插入
```

5.4 如何读取 HDMI 设备 EDID 信息

EDID: Extended Display Identification Data, 是一种 VESA 标准数据格式，其中包含终端的性能参数（比如：分辨率，颜色设置，频率范围等）。通过 EDID 信息可以了解电视

的基本性能，对于分析兼容性问题有一定帮助。

EDID 读取方法如下：

```
# cat /sys/class/hdmi/hdmi/edid > /data/edid.data
```

5.5 如何打开 HDMI 驱动的调试信息

如需要分析 HDMI 驱动的配置流程，可以从其 DEBUG 打印入手；开启 DEBUG 打印后，驱动将各步骤的关键信息打印到串口终端。

```
# echo 1 > /sys/class/hdmi/hdmi/attr/debug
```


6. FAQ

6.1 如何配置系统（UI）分辨率大小

6.1.1 问题描述

系统分辨率决定了 UI 分辨率大小。不同 UI 大小的显示效果不一样，比如 1080P-UI 比 720P-UI 的显示效果更清晰美观。厂商可以根据自身产品方案的定位，配置系统分辨率。

6.1.2 问题分析

- 查看系统分辨率的方法

  串口输入： wm size
  输出如下打印： Physical size: 1280x720
  这表明当前系统分辨率为 720P。

- 不同系统分辨率对应用的 DPI 值有不同的要求。
- 不同系统分辨率对系统的内存、DDR 带宽等需求是不一样的。系统分辨率越大，内存需求越大，DDR 带宽需求也越高。

6.1.3 解决方法

步骤一: 配置系统分辨率

修改方案 mk 文件下的属性值 ro.hwc.sysrsl。属性值对应的系统分辨率如下表所示。目前只支持三种配置。

系统分辨率	ro.hwc.sysrsl
720P	5
1080P	9
4K(3840x2160)	28

例如配置 1080P UI, 以 petrel-p1 方案为例
修改 device/softwinner/petrel-p1/petrel_fvd_p1.mk，增添或修改以下语句：

```
PRODUCT_PROPERTY_OVERIDES += \
    ro.hwc.sysrsl=9
```

步骤二：配置 DPI

修改方案 mk 文件下的属性值 ro.sf.lcd_density。该属性值对应的系统分辨率如下表所示。

ro.hwc.sysrsl	ro.sf.lcd_density
5	160
9	320
28	320

例如配置 320 dpi, 以 petrel-p1 方案为例
修改 device/softwinner/petrel-p1/petrel_fvd_p1.mk，增添或修改以下语句：

```
PRODUCT_PROPERTY_OVERIDES += \
    ro.sf.lcd_density=320
```

步骤三：检查内存配置状况

- 1080P 系统分辨率需要至少 1G 的内存；
- 4K 系统分辨率需要大于 1G 的内存。

步骤四：修改所有的系统应用和自研应用的布局和资源图片，支持系统分辨率对应的 DPI 值。

6.2 如何配置显示设备的默认输出分辨率

6.2.1 问题描述

第一次烧写完固件启动时，系统以某个默认显示模式输出到显示设备，比如 HDMI 输出 720P_50Hz、CVBS 输出 PAL 等。
厂商可根据方案需求，配置显示设备的默认输出模式。

6.2.2 问题分析

- 判断当前显示设备的输出模式的方法。

```
串口输入命令：cat /sys/class/disp/disp/attr/sys
输出打印如下：
screen 0:
de_rate 432000000 hz, ref_fps:50
//注释：下一行说明 HDMI 设备输出刷新率是 50Hz, 分辨率是 720P。
hdmi output mode(4)     fps:50.5      1280x 720
err:1   skip:24 irq:4601      vsync:318
BUF   enable ch[0] lyr[0] z[0] prem[N] a[globl 255] fmt[ 1] fb[1280, 720;  0,  0;  0,  0]
crop[   0,   0,1280, 720] frame[   0,   0,1280, 720] addr[6daf1000,      0,      0]
flags[0x      0] trd[0,0]
screen 1:
de_rate 432000000 hz, ref_fps:50
//注释：下一行说明 CVBS 设备输出刷新率是 50Hz, 分辨率是 720x 576，即 PAL。
tv output mode(11)      fps:50.5        720x 576
err:0   skip:20 irq:4346        vsync:0
```

- 显示设备的默认输出模式在 boot/linux 和 android 阶段都有对应的配置。

6.2.3 解决方法

步骤一：配置 boot/linux 阶段的显示设备默认输出模式

修改方案的 sys_config.fex。举例：配置 HDMI 默认输出模式为 1080P_60Hz，CVBS 默认输出模式是 PAL。以 petrel_p1 为例修改 lichee/tools/pack/chips/sun50iw6p1/configs/petrel-p1/sys_config.fex

```
[disp]
dev0_output_type = 4  //注释：表示显示设备 0 的输出类型是 HDMI
dev0_output_mode = 10 //注释：表示输出模式值是 10(十进制).
...
dev1_output_type = 2  //注释：表示显示设备 1 的输出类型是 CVBS
dev1_output_mode = 11 //注释：表示输出模式值是 11(十进制).
...
注意：输出模式具体值所代表的输出模式可查询头文件 lichee/linux-3.10/include/video/sunxi_display2.h 的定义.
enum disp_tv_mode {
    ...
    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_PAL = 0xB, //即十进制 11
    DISP_TV_MOD_NTSC = 0xe, //即十进制 14
}
```

步骤二：配置 Android 阶段的显示设备默认输出模式

修改方案 mk 文件下的属性值举例：配置 HDMI 默认输出模式为 1080P_60Hz，CVBS 默认输出模式是 PAL。以 petrel_p1 为例修改 device/softwinner/petrel-p1/petrel_fvd_p1.mk，增添或修改以下语句

HDMI_CHANNEL := 0 //注释：HDMI 使用 channel 0
HDMI_DEFAULT_MODE := 10 //注释：设置 HDMI 默认输出模式是 10(十进制)。
CVBS_CHANNEL := 1 //注释：CVBS 使用 channel 1
CVBS_DEFAULT_MODE := 11 //注释：设置 CVBS 默认输出模式是 11(十进制)。
注意：输出模式具体值所代表的输出模式与上述 sys_config.fex 的配置意义是完全相同的，可查询头文件 lichee/linux-3.10/include/video/sunxi_display2.h 的定义。

步骤三：重新编译固件

（1）进入 Android 编译环境根目录，输入命令：make installclean
（2）按照正常编译流程重新编译生成固件。

6.3 如何修改为竖屏显示

6.3.1 问题描述

竖屏 (portrait) 显示，是指 Android UI 和视频播放支持竖屏显示。

6.3.2 问题分析

无。

6.3.3 解决方法

步骤一：修改 bootlogo 图片内容把 bootlogo 图片内容修改为旋转后的内容，如下图所示：

1）内容为顺时针旋转 90° 的图片　　2）内容为顺时针旋转 270° 的图片

图 5: bootlogo 内容旋转

步骤二：修改方案 mk 文件，配置属性 ro.sf.rotation 以 petrel-p1 为例，修改 device/softwinner/petrel-p1/petrel_fvd_p1.mk，增添或修改以下语句：

```
# 顺时针旋转 90°
PRODUCT_PROPERTY_OVERRIDES += ro.sf.rotation=90
```

```
# 顺时针旋转 270°
PRODUCT_PROPERTY_OVERRIDES += ro.sf.rotation=270
```

步骤三：修改文件 WindowManagerService.java

```
// path: frameworks/base/services/java/com/android/server/wm/
//       WindowManagerService.java
//

int computeForcedAppOrientationLocked() {
    int req = getOrientationFromWindowsLocked();
    if (req == ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED) {
        req = getOrientationFromAppTokensLocked();
    }
    req = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
    return req;
}
```


将 ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE 修改为 ActivityInfo.SCREEN_ORIENTATION_PORTRAIT。

7. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.

