DISP 模块说明书

文档版本号: SDK-V1.0

发布日期: 2019-03-30

商标声明

、全志和其他全志商标均为珠海全志科技股份有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受全志公司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的购买或使用范围之内。除非合同另有约定，全志公司对本文档内容不做任何明示或默示的声明或保证。
由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2019-03-30	Allwinner	V316 初始化版本

目 录

表目录

图目录

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 AW1816 平台

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 Linux 显示驱动篇

2.1 模块介绍

2.1.1 模块功能介绍

本模块主要处理显示相关功能，主要功能如下：

- 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 (arg,yuv)
- 支持图像缩放处理
- 支持带多种制式的 HDMI 输出，包括 480P@60Hz、576P@50Hz、720P@50Hz、720P@60Hz、1080I@50Hz、1080I@60Hz、1080P@50Hz、1080P@60Hz、4K@30Hz 等
- 支持 CVBS 信号输出（PAL 和 NTSC）

2.1.2 模块配置

linux 显示驱动和 boot 显示驱动是基本一致的。配置上的区别是初始化配置和电源配置。

(1). BOOT 初始化配置

```
; devx_aaa: 的值取x0，表示显示设备的/1/2...，例如：id
; dev0_output_type: 表示显示设备的输出类型，0none(0), lcd(1), cvbs(2), hdmi(4)
; dev0_output_mode: 表示显示设备的输出模式，详见0sunxi_display2.的定义hdisp_tv_mode
; devx_screen_id: 表示显示设备使用的通道xDE
; devx_do_hpd: 对于有热插拔的显示设备，表示不做热插拔检测，表示做热插拔检测。01
; dev_output_dev: 默认显示设备的。例如下述配置，配置了和的信息，当两者都没有检测到热插拔时，即使用所指示的显示设备（即）作为输出设备。iddev0dev1dev_output_devdev0
; hdmi_mode_check: 模式选择策略。表示不检测电视是否支持当前模式；表示检测电视是否支持；表示如果没有更换电视（HDMI012vendor 相同）就不检测，否则检测。id
```

```
[disp_init]
dev0_output_type = 4
dev0_output_mode = 4
dev0_screen_id = 0
dev0_do_hpd = 1
dev1_output_type = 2
dev1_output_mode = 11
dev1_screen_id = 1
dev1_do_hpd = 1

dev2_output_type = 0   //表示没有，即只有和dev2dev0dev1.
def_output_dev = 0
hdmi_mode_check = 1
```

(2). LINUX 初始化配置

```
‘‘‘
;-------------------------------------------------------------------------
;disp init configuration
;
;disp_mode          (0:<screen0,fb0；>1:<screen1,fb0>)
;screenx_output_type    (0:none; 1:lcd; 3:hdmi;)
;screenx_output_mode    (used for hdmi output, 0:480i 1:576i 2:480p 3:576p 4:720p50)
(5:720p60 6:1080i50 7:1080i60 8:1080p24 9:1080p50 10:1080p60)
;fbx format         (0:ARGB 1:ABGR 2:RGBA 3:BGRA)
;fbx_width,fbx_height
(framebuffer horizontal/vertical pixels, fix to output resolution while equal 0)
;-------------------------------------------------------------------------
[disp_init]
disp_init_enable          = 1
disp_mode             = 0

screen0_output_type     = 3
screen0_output_mode     = 4

screen1_output_type     = 2
screen1_output_mod      = 11

fb0_format            = 0
fb0_width             = 1280
fb0_height            = 720
‘‘‘
```

(3). HDMI 模块配置

```
;----------------------------------------------------------------------------
;hdmi configuration
;----------------------------------------------------------------------------
[hdmi_para]
hdmi_used          = 1
hdmi_power         = ”vcc-hdmi-18”
```

(4). CVBS 模块配置

```
tv_used           = 1
tv_dac_used       = 1
tv_dac_src0       = 0
```

2.1.3 源码结构介绍

```
drivers
 ├──video                          显示驱动目录
 │   ├──fbmem.c                      framebuffer core
 │   ├──sunxi                      display driver for sunxi
 │   │  ├──disp2/                    的目录disp2
 │   │  │   ├──disp
 │   │  │   │   ├──dev_disp.c            display 层driver
 │   │  │   │   ├──dev_fb.c             framebuffer 层driver
 │   │  │   │   ├──de                层bsp
 │   │  │   │   │   ├──disp_lcd.c disp_manager.c ..
 │   │  │   │   │   ├──disp_al.c          层al
 │   │  │   │   │   │   └──lowlevel_sun*i/       层lowlevel
 │   │  │   │   │   │   ├──de_lcd.c de_rtmx.c...
 │   │  │   │   │   └──disp_sys_int.       层OSAL与操作系统相关层,
 │   │  │   │   ├──lcd/                  lcd driver
 │   │  │   │   │   ├─lcd_src_interface.c       与驱动的接口display
 │   │  │   │   │   ├─default_panel.c ...       平台已经支持的屏驱动
 │   │  │   ├──hdmi
 │   │  │   │   ├──dev_hdm.c            hdmi 层driver
 │   │  │   │   ├──drv_hdm.c            hdmi 层driver
 │   │  │   │   ├──aw                hdmi 层bsp
 │   │  │   ├──tv
 │   │  │   │   ├──dev_tv.c            tv 层driver
```

```
|   |   |   ├──drv_tv.c              tv 层driver
|   |   |   ├──de_tve.c              tv 层bsp
include
├──video                          video header dir
|   ├──sunxi_display2.c                display header file
```

2.2 图层操作说明

显示驱动中最重要的显示资源为图层，H5 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。

2.3 接口参数说明

平台	H5
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

2.4 图层主要参数

2.4.1 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。

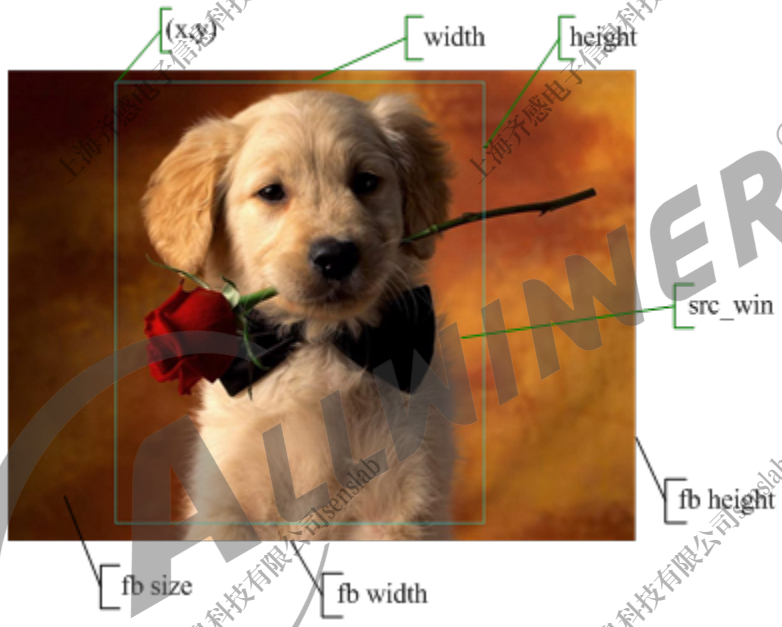

图 2-1: 图片 1.png

2.4.2 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。

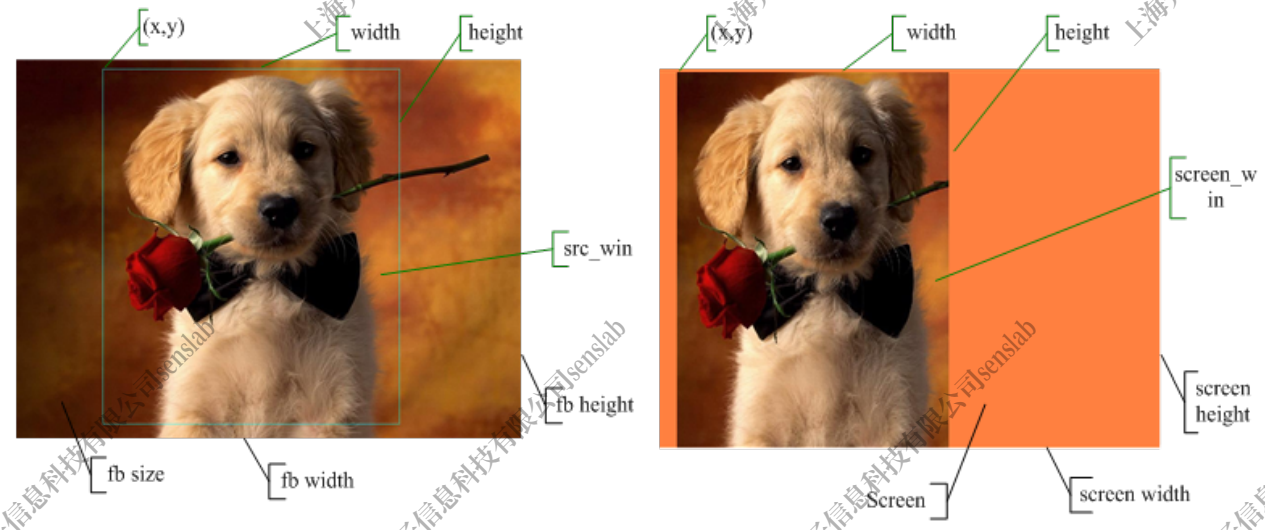

图 2-2: 图片 2.png

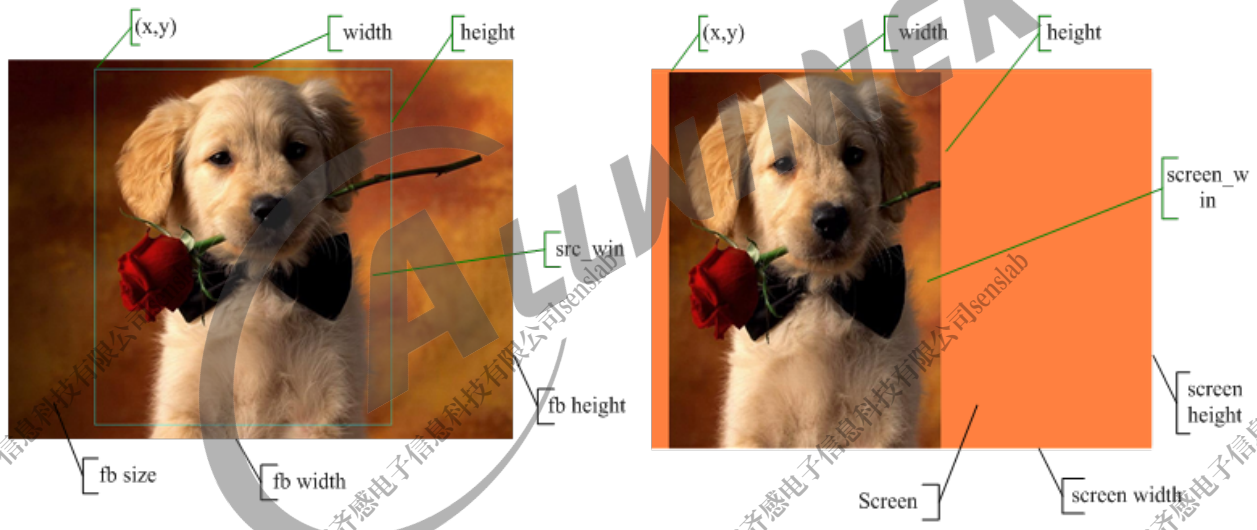

图 2-3: 图片 3.png

2.4.3 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。

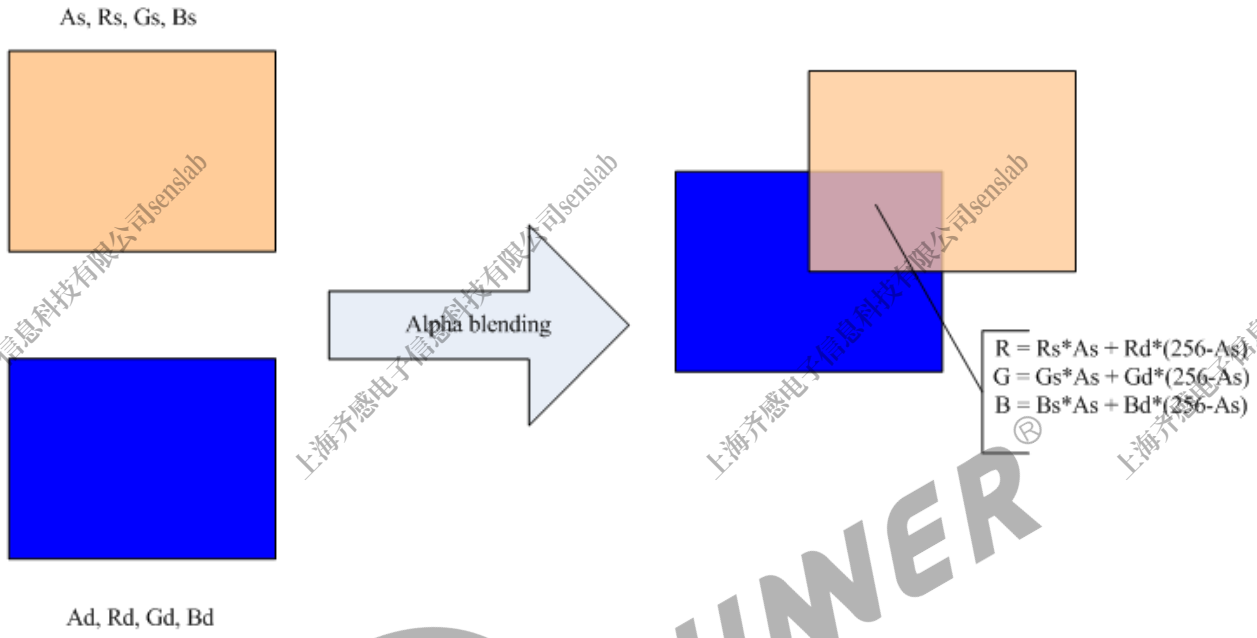

图 2-4: 图片 4.png

2.4.4 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
```

2.5 输出设备介绍

- H5 HDMI 输出和 CVBS 输出，或者二者同时输出。

2.5.1 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，模式是否支持的查询。

2.5.2 CVBS

H5 显示驱动支持 PAL 制式或 NTSC 制式信号输出。

2.6 接口描述

H5 平台下显示驱动给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。

2.6.1 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, 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);
```

- 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；

- 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，为显示驱动句柄，为屏disphdsel0/1
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，为显示驱动句柄，为屏disphdsel0/1
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/消息发送功能，为显示驱动句柄，为屏vsyncdisphdsel0/1
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

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：如果传递的是，将会关闭当前显示通道的输出。typeDISP_OUTPUT_TYPE_NONE
```

2.6.2 Layer Interface

- 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，为显示驱动句柄disphd
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; //地址FB
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);
```

- 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，为显示驱动句柄disphd
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);
```

2.6.3 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

```
//查询指定的模式是否支持HDMI
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);
```

2.6.4 enhance

- 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，为显示驱动句柄disphd
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);
```

- 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，为显示驱动句柄disphd
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);
```

- 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，为显示驱动句柄disphd
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，为显示驱动句柄disphd
unsigned int arg[3];

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

2.7 Data Structure

2.7.1 disp_fb_info

名称	disp_fb_info
功能描述	用于描述一个 display frambuffer 的属性信息
属性	类型
addr[3]	unsigned long long
size[3]	disp_rectsz
align[3]	unsigned int
format	disp_pixel_format
color_space	disp_color_space
trd_right_addr[3]	unsigned int
pre_multiply	bool
crop	disp_rect64
flags	disp_buffer_flags
scan	disp_scan_flags

2.7.2 disp_layer_info

名称	disp_layer_info
功能描述	用于描述一个图层的属性信息
属性	类型
mode	disp_layer_mode
zorder	unsigned char
alpha_mode	unsigned char
alpha_value	unsigned char
screen_win	disp_rect
b_trd_out	bool
out_trd_mode	disp_3d_out_mode
color unsigned	int
fb	disp_fb_info
id	unsigned int

2.7.3 disp_layer_config

名称	disp_layer_config
功能描述	用于描述一个图层配置的属性信息
属性	类型
info	disp_layer_info
enable	bool
channel	unsigned int
layer_id	unsigned int

2.7.4 disp_color_info

名称	disp_color_info
功能描述	用于描述一个颜色的信息
属性	类型

名称	disp_color_info
alpha	u8
red	u8
green	u8
blue	u8

2.7.5 disp_rect

名称 disp_rect	
功能描述	用于描述一个矩形窗口的信息
属性	类型
x	s32
y	s32
width	s32
height	s32

2.7.6 disp_position

名称	disp_position
功能描述	用于描述一个坐标的信息
属性	类型
x	s32
y	s32

2.7.7 disp_rectsz

名称	disp_rectsz
功能描述	用于描述一个矩形尺寸的信息
属性	类型
width	s32

名称	disp_rectsz
height	s32

2.7.8 disp_pixel_format

名称	disp_pixel_format
功能描述	像素格式枚举值
属性	类型
DISP_FORMAT_ARGB_8888	enum
DISP_FORMAT_ABGR_8888	enum
DISP_FORMAT_RGBA_8888	enum
DISP_FORMAT_BGRA_8888	enum
DISP_FORMAT_XRGB_8888	enum
DISP_FORMAT_XBGR_8888	enum
DISP_FORMAT_RGBX_8888	enum
DISP_FORMAT_BGRX_8888	enum
DISP_FORMAT_RGB_888	enum
DISP_FORMAT_BGR_888	enum
DISP_FORMAT_RGB_565	enum
DISP_FORMAT_BGR_565	enum
DISP_FORMAT_ARGB_4444	enum
DISP_FORMAT_ABGR_4444	enum
DISP_FORMAT_RGBA_4444	enum
DISP_FORMAT_BGRA_4444	enum
DISP_FORMAT_ARGB_1555	enum
DISP_FORMAT_ABGR_1555	enum
DISP_FORMAT_RGBA_5551	enum
DISP_FORMAT_BGRA_5551	enum
DISP_FORMAT_YUV444_I_AYUV	enum
DISP_FORMAT_YUV444_I_VUYA	enum
DISP_FORMAT_YUV422_I_YVYU	enum
DISP_FORMAT_YUV422_I_YUYV	enum
DISP_FORMAT_YUV422_I_UYVY	enum
DISP_FORMAT_YUV422_I_VYUY	enum

名称	disp_pixel_format
DISP_FORMAT_YUV444_P	enum
DISP_FORMAT_YUV422_P	enum
DISP_FORMAT_YUV420_P	enum
DISP_FORMAT_YUV411_P	enum
DISP_FORMAT_YUV422_SP_UVUV	enum
DISP_FORMAT_YUV422_SP_VUVU	enum
DISP_FORMAT_YUV420_SP_UVUV	enum
DISP_FORMAT_YUV420_SP_VUVU	enum
DISP_FORMAT_YUV411_SP_UVUV	enum
DISP_FORMAT_YUV411_SP_VUVU	enum

2.7.9 disp_buffer_flags

名称	disp_buffer _flags
功能描述	用于描述 3D 源格式
属性	类型
DISP_BF_NORMAL	enum
DISP_BF_STEREO_TB	enum
DISP_BF_STEREO_FP	enum
DISP_BF_STEREO_SSH	enum
DISP_BF_STEREO_SSF	enum
DISP_BF_STEREO_LI	enum

2.7.10 disp_3d_out_mode

名称 disp_3d_out_mode	
功能描述用于描述 3D 输出模式	
属性	类型
DISP_3D_OUT_MODE_CI_1	enum
DISP_3D_OUT_MODE_CI_2	enum
DISP_3D_OUT_MODE_CI_3	enum

名称 disp_3d_out_mode	
DISP_3D_OUT_MODE_CI_4	enum
DISP_3D_OUT_MODE_LIRGB	enum
DISP_3D_OUT_MODE_TB	enum
DISP_3D_OUT_MODE_FP	enum
DISP_3D_OUT_MODE_SSF	enum
DISP_3D_OUT_MODE_SSH	enum
DISP_3D_OUT_MODE_LI	enum
DISP_3D_OUT_MODE_LA	enum

2.7.11 disp_color_space

名称	disp_color_space
功能描述	用于描述颜色空间类型
属性	类型
DISP_BT601	enum
DISP_BT709	enum
DISP_YCC	enum

2.7.12 disp_output_type

名称	disp_output_type
功能描述	用于描述显示输出类型
属性	类型
DISP_OUTPUT_TYPE_NONE	enum
DISP_OUTPUT_TYPE_LCD	enum
DISP_OUTPUT_TYPE_TV	enum
DISP_OUTPUT_TYPE_HDMI	enum
DISP_OUTPUT_TYPE_VGA	enum

2.7.13 disp_tv_mode

名称	disp_tv_mode
功能描述	用于 TV 输出模式
属性	类型
DISP_TV_MOD_480I	enum
DISP_TV_MOD_576I	enum
DISP_TV_MOD_480P	enum
DISP_TV_MOD_576P	enum
DISP_TV_MOD_720P_50HZ	enum
DISP_TV_MOD_720P_60HZ	enum
DISP_TV_MOD_1080I_50HZ	enum
DISP_TV_MOD_1080I_60HZ	enum
DISP_TV_MOD_1080P_24HZ	enum
DISP_TV_MOD_1080P_50HZ	enum
DISP_TV_MOD_1080P_60HZ	enum
DISP_TV_MOD_1080P_24HZ_3D_FP	enum
DISP_TV_MOD_720P_50HZ_3D_FP	enum
DISP_TV_MOD_720P_60HZ_3D_FP	enum
DISP_TV_MOD_1080P_25HZ	enum
DISP_TV_MOD_1080P_30HZ	enum
DISP_TV_MOD_PAL	enum
DISP_TV_MOD_PAL_SVIDEO	enum
DISP_TV_MOD_NTSC	enum
DISP_TV_MOD_NTSC_SVIDEO	enum
DISP_TV_MOD_PAL_M	enum
DISP_TV_MOD_PAL_M_SVIDEO	enum
DISP_TV_MOD_PAL_NC	enum
DISP_TV_MOD_PAL_NC_SVIDEO	enum
DISP_TV_MOD_3840_2160P_30HZ	enum
DISP_TV_MOD_3840_2160P_25HZ	enum
DISP_TV_MOD_3840_2160P_24HZ	enum
DISP_TV_MODE_NUM	enum

2.7.14 disp_output

名称	disp_output
功能描述	用于描述显示输出类型，模式
属性	类型
type	unsigned int
mode	unsigned int

2.7.15 disp_layer_mode

名称	disp_output
功能描述	用于描述图层模式
属性	类型
LAYER_MODE_BUFFER	enum
LAYER_MODE_COLOR	enum

2.7.16 disp_scan_flags

名称	disp_output
功能描述	用于描述图层模式
属性	类型
DISP_SCAN_PROGRESSIVE	enum
DISP_SCAN_INTERLACED_ODD_FLD_FIRST	enum
DISP_SCAN_INTERLACED_EVEN_FLD_FIRST	enum

2.8 demo

2.8.1 显示一个图层

```
//demo1. 在屏幕上显示一个图层

#define writel(val, addr) (*(((unsigned int *)(addr))) = (val))
//only for DISP_FORMAT_ARGB_8888 format
int disp_draw_h_colorbar(unsigned int base, unsigned int width, unsigned int height)
{
    unsigned int i=0, j=0;

    for(i = 0; i<height; i++) {
        for(j = 0; j<width/4; j++) {
            unsigned int offset = 0;

            offset = width * i + j;
            writel((((1<<8)-1)<<24) | (((1<<8)-1)<<16), base + offset*4);

            offset = width * i + j + width/4;
            writel((((1<<8)-1)<<24) | (((1<<8)-1)<<8), base + offset*4);

            offset = width * i + j + width/4*2;
            writel((((1<<8)-1)<<24) | (((1<<8)-1)<<0), base + offset*4);

            offset = width * i + j + width/4*3;
            writel((((1<<8)-1)<<24) | (((1<<8)-1)<<16) | (((1<<8)-1)<<8), base + offset*4);
        }
    }

    return 0;
}

int main(int argc, char **argv)
{
    unsigned int arg[3];
    disp_layer_info info;
    unsigned int width = 1280;
    unsigned int height = 800;
    unsigned int ret = 0;
    unsigned int screen_id = 0;
    unsigned int layer_id = 0;
    unsigned int mem_id = 0;
    unsigned int buffer_num = 2;
    unsigned int dispfh;
    unsigned int fb_width,fb_height;
    unsigned int mem, mem_phy;
```

```
if((dispfh = open("/dev/disp",O_RDWR)) == -1) {
    printf("open display device fail!\n");
    return -1;
}

/* get screen size */
arg[0] = screen_id;
width = ioctl(dispfh,DISP_GET_SCN_WIDTH,(void*)arg);
height = ioctl(dispfh,DISP_GET_SCN_HEIGHT,(void*)arg);
printf("screen_size=%d x %d \n", width, height);

fb_width = width;
fb_height = height;

/* request memory for layer */
arg[0] = mem_id;
arg[1] = fb_width*fb_height*4*buffer_num;
arg[2] = 0;
arg[3] = 0;
if(ioctl(dispfh,DISP_MEM_REQUEST,(void*)arg) < 0) {
    printf("DISP_MEM_REQUEST 0\n");
    close(dispfh);
    return -1;
}

/* mmap memory requested */
arg[0] = mem_id;
arg[1] = 0;
arg[2] = 0;
arg[3] = 0;
ioctl(dispfh,DISP_MEM_SELIDX,(void*)arg);
mem = (int)mmap(NULL, fb_width*fb_height*4*buffer_num, PROT_READ |
PROT_WRITE, MAP_SHARED, dispfh, 0L);
if(mem == 0) {
    printf("DISP_MEM_MAP 0\n");
    arg[0] = mem_id;
    arg[1] = 0;
    arg[2] = 0;
    arg[3] = 0;
    ioctl(dispfh,DISP_MEM_RELEASE,(void*)arg);
    close(dispfh);
    return -1;
}

/* draw colorbar on the memory requested */
```

```
memset((void*)mem, 0x0, fb_width*fb_height*4*buffer_num);
disp_draw_h_colorbar(mem, fb_width, fb_height);
munmap((void*)mem, fb_width*fb_height*4*buffer_num);

/* get physics address */
arg[0] = mem_id;
mem_phy = ioctl(dispfh,DISP_MEM_GETADR,(void*)arg);

/* set layer info */
memset(&info, 0, sizeof(disp_layer_info));
info.mode           = DISP_LAYER_WORK_MODE_NORMAL;
info.fb.addr[0]     = (__u32)mem_phy; //地址FB
info.fb.size.width  = width;
info.fb.format      = DISP_FORMAT_ARGB_8888; //DISP_FORMAT_YUV420_P
info.fb.src_win.x   = 0;
info.fb.src_win.y   = 0;
info.fb.src_win.width = width;
info.fb.src_win.height= height;
info.ck_enable      = 0;
info.alpha_mode     = 1; //global alpha
info.alpha_value    = 0xff;
info.pipe           = 1;
info.screen_win.x   = 0;
info.screen_win.y   = 0;
info.screen_win.width = width;
info.screen_win.height= height;
info.id             = 0;

arg[0] = screen_id;//显示通道0
arg[1] = layer_id;//图层0
arg[2] = (unsigned int)&info;
ret = ioctl(dispfh, DISP_LAYER_SET_INFO, (void*)arg);
if(0 != ret)
   printf("fail to set layer info\n");

/* enable layer */
arg[0] = screen_id;
arg[1] = layer_id;
arg[2] = 0;
arg[3] = 0;
ret = ioctl(dispfh,DISP_LAYER_ENABLE,(void*)arg);
if(0 != ret)
   printf("fail to enable layer\n");

sleep(5);
```

```
    /* clear resource */
    arg[0] = screen_id;
    arg[1] = layer_id;
    arg[2] = 0;
    arg[3] = 0;
    ret = ioctl(dispfh,DISP_LAYER_DISABLE,(void*)arg);
    if(0 != ret)
        printf("fail to enable layer\n");

    memset(&info, 0, sizeof(disp_layer_info));
    arg[0] = screen_id;
    arg[1] = layer_id;
    arg[2] = (unsigned int)&info;
    ret = ioctl(dispfh, DISP_LAYER_SET_INFO, (void*)arg);
    arg[0] = mem_id;
    ioctl(dispfh,DISP_MEM_RELEASE,(void*)arg);

    close(dispfh);

    return 0;
}
```

3 Android 显示框架篇

3.1 模块介绍

3.1.1 模块功能介绍

（1）DisplayManager 类是 Android 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。

（2）DisplayManagerPolicy2 提供了热插拔时显示类型和显示模式的切换策略，即 hdmi 和 cvbs 之间的切换。

（3）实现了 CMCC 移动方案的显示接口。

3.1.2 相关术语介绍

（1）虚拟屏幕：即绝大多数 APK 绘制 UI 界面时的系统默认分辨率大小的 Surface。H5 平台，在无特殊配置的情况下，系统默认的虚拟屏幕的分辨率为 1920*1080。对于运行中的系统来说，虚拟屏幕的分辨率是不变的。

（2）实际屏幕：即具体的显示模式，如 HDMI 的 720P@50Hz、1080P@60Hz 等。各种显示模式可以在系统运行中进行切换。

（3）SCALE 显示方式：当虚拟屏幕和实际屏幕不相等时，系统必须使用 SCALE 显示模式来进行缩放显示。

（4）P2P 显示方式：即点对点显示模式。只有当虚拟屏幕与实际屏幕相等时，才能使用 P2P 显示模式。

3.1.3 模块配置介绍

（1）实际屏幕的配置为了实现开机画面平滑过渡，boot 阶段、linux 阶段和 android 阶段都为处于同一显示模式（即实际屏幕不变）。在烧写固件后，系统首次启动的实际屏幕由默认值（sys_config.fex 的配置）决定：

```
[disp_init]
dev0_output_type = 4    //是dev0HDMI
dev0_output_mode = 4    //默认输出模式为720P50Hz
dev0_screen_id = 0      //使用显示通道HDMI0
dev0_do_hpd = 1         //支持热插拔，并进行热插拔检测HDMI
def_output_dev = 0      //默认输出设备的为，上述即配置所描述的显示设备id0dev0_xxx
```

系统非首次启动的实际屏幕由上次关机前的实际屏幕决定。

（2）虚拟屏幕的配置在 SCALE 显示方式下，虚拟屏幕的分辨率由配置属性 ro.hwc.sysrsl 决定。配置在各个具体方案的 mk 文件，如device/softwinner/cheetah_fvd_p1/cheetah-fvd_p1.mk。配置值见表 3-1.

（3）显示方式的配置显示方式有两种：SCALE 显示方式和 P2P 显示方式，由属性 persist.sys.hwc_p2p 决定。配置在各个具体方案的 mk 文件，如device/softwinner/cheetah-fvd-p1/cheetah_fvd_p1.mk。配置值见表 3-1.

表 3-1 配置值与虚拟屏幕分辨率的对应关系为：

persist.sys.hwc_p2p	ro.hwc.sysrsl	虚拟屏幕的分辨率
1（P2P）	—	开机时由实际屏幕的分辨率决定
0 或不配置 (SCALE)	4/5	1280 * 720
0 或不配置 (SCALE)	6/7/8/9/10	1920 * 1080
0 或不配置 (SCALE)	28/29/30	3840 * 2160
0 或不配置 (SCALE)	其他值或不配置	1920 * 1080

（4）热插拔双显同显配置 H5 默认使用自适应双显策略（HDMI 和 CVBS 同时显示相同内容画面，HDMI 是主显，CVBS 是辅显）方案，必须配置如下：

A）sys_config.fex 配置 HDMI 和 CVBS 使用的显示通道。

B）修改 Android 热插拔

```
device/softwinner/cheetah-fvd-p1/cheetah_fvd_p1.mk
PRODUCT_PROPERTY_OVERRIDES += \
persist.sys.disp_policy=3 \
persist.sys.hdmi_hpd=1 \
persist.sys.hdmi_rvthpd=0 \
persist.sys.cvbs_hpd=1 \
persist.sys.cvbs_rvthpd=0
#DISPLAY_INIT_POLICY is used in init_disp.c to choose display policy.
DISPLAY_INIT_POLICY := 3
HDMI_CHANNEL := 0
HDMI_DEFAULT_MODE := 4
CVBS_CHANNEL := 1
CVBS_DEFAULT_MODE := 11
```

（5）横屏竖显配置方法 A) 修改文件frameworks/base/services/java/com/android/server/wm/WindowManagerService.java里的函数

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

把红色字体变量修改为：ActivityInfo.SCREEN_ORIENTATION_PORTRAIT ;

B）在方案下的 mk 文件（例如 device\softwinner\cheetah-fvd-p1\cheetah_fvd_p1.mk）配置属性 ro.sf.rotation。

```
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
```

C）修改 bootlogo 图片的内容把 bootlogo 图片内容修改为旋转后的内容，如

图 3-1: 1 内容为顺时针旋转 90° 的图片

图 3-2: 2 内容为顺时针旋转 270° 的图片

（6）切边设置方法一般的电视显示存在画面溢出的现象，为了解决这种现象，H5 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：

```
在方案下的文件（例如
mk ‘device\softwinner\cheetah-fvd-p1\cheetah_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%.
```

如果没有配置，默认切边值为 100%。

（7）显示系统 density 配置方法显示系统的 density 值会影响系统 UI 画面的布局，其值越大，图片布局也越大。如下图，左图的 density 是 160，右图的 density 是 200.

图 3-3: 3

图 3-4: 4

修改 density 的方法如下：

```
在方案下的文件（例如
mk ‘device\softwinner\cheetah-fvd-p1\cheetah_fvd_p1.mk）配置属性：‘
persist.sys.disp_density=160   //配置值为density160
```

3.1.4 源码结构介绍

DisplayManager 所在目录

android/frameworks/base/core/java/android/hardware/display

DisplayManagerPolicy2 所在目录

android/frameworks/base/core/java/android/hardware/display

CMCC 显示接口所在目录

android/vendor/cmccwasu/frameworks/display

init_disp 所在的目录

android/system/core/init/

3.2 接口描述

3.2.1 Display Mode Interface

public int getDisplayOutputMode(int displaytype)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
Display.TYPE_UNKNOWN = 0, //
Display.TYPE_BUILT_IN = 1, //主显
```

- RETURNS

返回 显示模式

- DESCRIPTION

获取当前的显示模式

- DEMO

```
DisplayManager mDm;
int disp_mode;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
disp_mode = mDm.getDisplayOutputMode(Display.TYPE_BUILT_IN);
```

public int getDisplayOutputType(int displaytype)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
   Display.TYPE_BUILT_IN = 1, //主显
```

- RETURNS

返回 显示模式

- DESCRIPTION

获取当前的显示模式

- DEMO

```
DisplayManager mDm;
int disp_type;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
disp_type = mDm.getDisplayOutputType(Display.TYPE_BUILT_IN);
```

public int getDisplayOutput(int displaytype)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
   Display.TYPE_BUILT_IN = 1, //主显
```

- RETURNS

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

- DESCRIPTION

```
获取当前的显示类型和显示模式。
```

- DEMO

```
DisplayManager mDm;
int disp_output
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
disp_output = mDm.getDisplayOutput(Display.TYPE_BUILT_IN);
```

public int setDisplayOutputMode(int displaytype, int type, int mode)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
 Display.TYPE_BUILT_IN = 1, //主显
@type:          显示类型
    @mode:
```

- RETURNS

```
返回0
```

- DESCRIPTION

```
设置显示模式。
```

- DEMO

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

public boolean isSupportHdmiMode(int displaytype, int mode)

- ARGUMENTS

```
   @Displaytype:  显示通路( see Display.java)
 Display.TYPE_BUILT_IN = 1, //主显
    @mode:        显示模式
```

- RETURNS

返回：当前电视支持该，则返回；不支持则。modetruefalse

- DESCRIPTION

设置显示模式。

- DEMO

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

public int saveDisplayResolution(int displaytype, int type, int mode)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
  Display.TYPE_BUILT_IN  =  1, //主显
    @tyep:          显示类型
    @mode:
```

- RETURNS

返回0

- DESCRIPTION

保存下次开机时所使用的显示模式。重点说明调用场景：需确认该显示模式是当前电视支持的。

- DEMO

```
DisplayManager mDm;
int type = DISPLAY_OUTPUT_TYPE_HDMI;
int mode = DISPLAY_TVFORMAT_1080P_60HZ;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
mDm.saveDisplayResolution(Display.TYPE_BUILT_IN, type, mode);
```

3.2.2 Display Margin Interface

public int getDisplayMargin(int displaytype)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
```

- 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

```
 DisplayManager mDm;
int percents[2];
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
percents = mDm.getDisplayMargin(Display.TYPE_BUILT_IN);
```

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

- ARGUMENTS

```
@Displaytype: ( see Display.java)
 Display.TYPE_BUILT_IN = 1, //主显
@ hpercent:
 Percent of horizen.
@ vpercent:
     Percent of vertical.
```

- RETURNS

```
 0
```

- DESCRIPTION

```
 设置屏幕显示画面的纵向和横向的缩放比例
```

- DEMO

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

3.2.3 Display 3D Interface

public int getDisplaySupport3DMode(int displaytype)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显.
```

- RETURNS

```
显示设备支持3，则返回；否则返回。D10
```

- DESCRIPTION

```
查询显示设备是否3模式D
```

- DEMO

```
DisplayManager mDm;
int is3Dsupport;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
is3Dsupport = mDm.getDisplaySupport3DMode(Display.TYPE_BUILT_IN);
```

public int setDisplay3DMode(int displaytype, int trdmode)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
 Display.TYPE_BUILT_IN  =  1, //主显
 @ trdmode: (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

```
0
```

- DESCRIPTION

```
设置视频的3模式D
```

- DEMO

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

3.2.4 Display Color Interface

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
```

- 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

```
DisplayManager mDm;
int bright;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
bright = mDm.getDisplayBright(Display.TYPE_BUILT_IN);
```

public int setDisplayBright(int displaytype, int bright)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
@bright: the value of bright
```

- RETURNS

```
0
```

- DESCRIPTION

```
设置显示画面的亮度
```

- DEMO

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

public int getDisplayContrast(int displaytype)

- ARGUMENTS

```
@Displaytype:   显示通路( see Display.java)
   Display.TYPE_BUILT_IN  = 1, //主显
```

- 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

```
DisplayManager mDm;
int contrast;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
contrast = mDm.getDisplayContrast(Display.TYPE_BUILT_IN);
```

public int setDisplayContrast(int displaytype, int contrast)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
@contrast : the value of contrast
```

- RETURNS

```
0
```

- DESCRIPTION

```
设置显示画面的对比度
```

- DEMO

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

public int getDisplaySaturation(int displaytype)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
 Display.TYPE_BUILT_IN = 1, //主显
```

- 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

```
DisplayManager mDm;
int saturation;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
saturation = mDm.getDisplaySaturation(Display.TYPE_BUILT_IN);
```

public int setDisplaySaturation(int displaytype, int saturation)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
 Display.TYPE_BUILT_IN = 1, //主显
@saturation: the value of saturation
```

- RETURNS

```
0
```

- DESCRIPTION

```
设置显示画面的饱和度
```

- DEMO

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

public int getDisplayhue(int displaytype)

- ARGUMENTS

```
@Displaytype:    显示通路( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
```

- RETURNS

```
返回 显示画面的色度值
```

- DESCRIPTION

获取当前的显示画面的色度

- DEMO

```
DisplayManager mDm;
int hue;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
Hue = mDm.getDisplayHue(Display.TYPE_BUILT_IN);
```

public int setDisplayHue(int displaytype, int hue)

- ARGUMENTS

```
@Displaytype: ( see Display.java)
Display.TYPE_BUILT_IN = 1, //主显
@hue: the value of hue
```

- RETURNS

```
0
```

- DESCRIPTION

设置显示画面的色度

- DEMO

```
DisplayManager mDm;
int hue = 50;
mDm = (DisplayManager)mCtx.getSystemService(Context.DISPLAY_SERVICE);
mDm.setDisplayHue(Display.TYPE_BUILT_IN, hue);
```

3.3 显示策略

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。

H5显示的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 3-5: 3-2 H5 双显显示的热插拔消息处理策略

3.4 CMCC 显示接口

摘录的内容:

图 3-6: 5

显示设置采用 android AIDL 系统服务方式来实现，类似 Android WIFI 设置框架，扩展显示相关设置，框架如下：

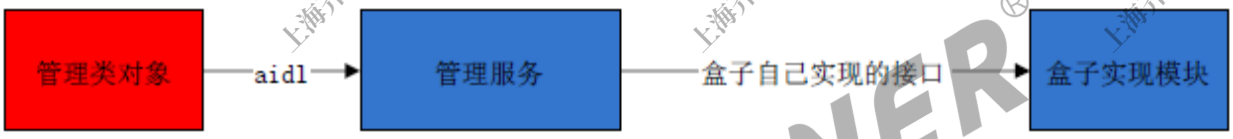

图 3-7: 6

需要扩展 framework 代码

1 增加 android.os.display. DisplayManager.java 类

2 修改 android.content.Context.java 类增加 public static final String DISPLAY_MANAGER_SERVICE = “displayManagerService”;

获取 DisplayManager 方法：

```
DisplayManager dm = (DisplayManager)context.getSystemService(Context.DISPLAY_MANAGER_SERVICE);
```

盒子底层需要实现一个显示管理的功能，显示管理所需要的接口见下文要求应管理类接口如下（具体方法需要机顶盒底层实现）：

```
package android.os.display;
/** * 输出相关控制的管理类 * * * @Date 2013-10-25 */
   public class DisplayManager {
      public final static int DISPLAY_STANDARD_1080P_60 = 0;
      public final static int DISPLAY_STANDARD_1080P_50 = 1;
      public final static int DISPLAY_STANDARD_1080P_30 = 2;
```

```
public final static int DISPLAY_STANDARD_1080P_25 = 3;
public final static int DISPLAY_STANDARD_1080P_24 = 4;
public final static int DISPLAY_STANDARD_1080I_60 = 5;
public final static int DISPLAY_STANDARD_1080I_50 = 6;
public final static int DISPLAY_STANDARD_720P_60 = 7;
public final static int DISPLAY_STANDARD_720P_50 = 8;
public final static int DISPLAY_STANDARD_576P_50 = 9;
public final static int DISPLAY_STANDARD_480P_60 = 10;
public final static int DISPLAY_STANDARD_PAL = 11;
public final static int DISPLAY_STANDARD_NTSC = 12;
/*** 判断是否支持该制式** @param standard
*    {@link #DISPLAY_STANDARD_1080P_60}*
*    {@link #DISPLAY_STANDARD_1080P_50}
*    {@link #DISPLAY_STANDARD_1080P_30}
*    {@link #DISPLAY_STANDARD_1080P_25}
*    {@link #DISPLAY_STANDARD_1080P_24}
*    {@link #DISPLAY_STANDARD_1080I_60}
*    {@link #DISPLAY_STANDARD_1080I_50}
*    {@link #DISPLAY_STANDARD_720P_60}
*    {@link #DISPLAY_STANDARD_720P_50}
*    {@link #DISPLAY_STANDARD_576P_50}*
*    {@link #DISPLAY_STANDARD_480P_60}*
*    {@link #DISPLAY_STANDARD_PAL}
*    {@link #DISPLAY_STANDARD_NTSC}
*     @return
*/
public boolean isSupportStandard(int standard)
{
//need stb implements    ……
}

/**
* 获取盒子所支持的所有制式
* @return
*/
public int[] getAllSupportStandards()  {
   //stb implements    ……
}

/**
* 设置制式* @param standard
* {@link #DISPLAY_STANDARD_1080P_60}
* {@link #DISPLAY_STANDARD_1080P_50}
*  {@link #DISPLAY_STANDARD_1080P_30}
*  {@link #DISPLAY_STANDARD_1080P_25}
```

```
     * {@link #DISPLAY_STANDARD_1080P_24}
     *  {@link #DISPLAY_STANDARD_1080I_60}
     *  {@link #DISPLAY_STANDARD_1080I_50}
     *  {@link #DISPLAY_STANDARD_720P_60}
     *  {@link #DISPLAY_STANDARD_720P_50}
     *  {@link #DISPLAY_STANDARD_576P_50}
     * {@link #DISPLAY_STANDARD_480P_60}
     * {@link #DISPLAY_STANDARD_PAL} {@link #DISPLAY_STANDARD_NTSC}
     */
    public void setDisplayStandard(int standard) {
        //stb implements   ……
    }

    /**
     * 获取当前制式* @return {@link #DISPLAY_STANDARD_1080P_60}
    * {@link #DISPLAY_STANDARD_1080P_50}
    * {@link #DISPLAY_STANDARD_1080P_30}
    * {@link #DISPLAY_STANDARD_1080P_25}
    * {@link #DISPLAY_STANDARD_1080P_24}
    * @link #DISPLAY_STANDARD_1080I_60}
    * {@link #DISPLAY_STANDARD_1080I_50}
    * {@link #DISPLAY_STANDARD_720P_60}
    * {@link #DISPLAY_STANDARD_720P_50}
    * {@link #DISPLAY_STANDARD_576P_50}
    * {@link #DISPLAY_STANDARD_480P_60}
    * @link #DISPLAY_STANDARD_PAL}
    * {@link #DISPLAY_STANDARD_NTSC}*/
    public int getCurrentStandard() {
     //need stb implements   ……
    }

    /**
    * 设置屏幕的边距
    * @param left  左边距
    * @param top  上边距
    * @param right  右边距
    * @param bottom  下边距
    */
    public void setScreenMargin(int left, int top, int right, int bottom) {
     //need stb implements   ……
     }

    /**
     * 获取屏幕的边距*
     *  @return int[]*
```

```
     * 数组内容顺序分别为：左边距，上边距，右边距，下边距
     */public int[] getScreenMargin() {
     //need stb implements   ……
     }

     /**
     * 保存参数
     */
     public void saveParams () {
        //need stb implements
     }
  }
```

```

# 调试说明

## 如何读取驱动图层信息
```

4 cat /sys/class/disp/disp/attr/sys

```
## 如何查看系统图层信息Android
```

5 dumpsys SurfaceFlinger

6 setprop debug.hwc.showfps 2

7 logcat -s Hwcomposer

```
##如何判断设备是否插入
```

8

9 cat /sys/class/switch/hdmi/state

10

1: 表示 HDMI 插入 0: 表示 HDMI 未插入

```
## 如何读取设备信息HDMIEDID

EDID: Extended Display Identification Data,是一种标准数据格式，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）。
VESA通过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：EDID**
```

11 cat /sys/class/hdmi/hdmi/edid > /data/edid.data

如何打开驱动的调试信息HDMI如需要分析驱动的配置流程，可以从其打印入手；

HDMIDEBUG开启打印后，驱动将各步骤的关键信息打印到串口终端。
DEBUG

12 echo 1 > /sys/class/hdmi/hdmi/attr/debug

```
# FAQ

## 如何配置系统（）分辨率大小UI

### 问题描述系统分辨率决定了分辨率大小。

UI不同大小的显示效果不一样，比如
UI1080P-比UI720P-的显示效果更清晰美观。UI厂商可以根据自身产品方案的定位，配置系统分辨率。


### 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值有不同的要求。DPI

+ 不同系统分辨率对系统的内存、带宽等需求是不一样的。系统分辨率越大，内存需求越大，带宽需求也越高。DDRDDR

### 解决方法步骤一

: 配置系统分辨率修改方案文件下的属性值

mkro.hwc.。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。sysrsl
\begin{tabular}{|l|l|}\hline系统分辨率
 & ro.hwc.sysrsl \\ \hline
720P  & 5 \\
1080P  & 9 \\
4K(3840x2160)  & 28 \\ \hline
\end{tabular}
```

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修改方案文件下的属性值

mkro.sf.。该属性值对应的系统分辨率如下表所示。lcd_density
\begin{tabular}{|l|l|}\hline
ro.hwc.sysrsl & ro.sf.lcd_density \\ \hline
5 & 160 \\
9 & 320 \\
28 & 320 \\ \hline
\end{tabular}

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

+ 1080系统分辨率需要至少的内存：P1G

+ 4系统分辨率需要大于的内存。K1G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值。

DPI

如何配置显示设备的默认输出分辨率

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，比如输出、输出等。HDMI720P_50HzCVBSPAL
厂商可根据方案需求，配置显示设备的默认输出模式。

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/和阶段都有对应的配置。linuxandroid

### 解决方法步骤一：

 配置boot/阶段的显示设备默认输出模式linux修改方案的

sys_config.。fex举例：配置默认输出模式为，默认输出模式是。以为例
HDMI1080P_60HzCVBSPALpetrel_p1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 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 0 是 HDMI 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：

 配置阶段的显示设备默认输出模式Android修改方案文件下的属性值

mk举例：配置默认输出模式为，默认输出模式是。以为例
HDMI1080P_60HzCVBSPALpetrel_p1修改
device/softwinner/petrel-p1/petrel_fvd_p1.，增添或修改以下语句mk
```

HDMI_CHANNEL := 0 //注释：HDMI 是显示设备 0 HDMI_DEFAULT_MODE := 10 //注释：HDMI 默认输出模式值是 10，十进制。CVBS_CHANNEL := 1 //注释：HDMI 是显示设备 0 CVBS_DEFAULT_MODE := 11 //注释：HDMI 默认输出模式值是 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。

如何修改为竖屏显示

问题描述竖屏

(portrait)显示，是指Android 和视频播放支持竖屏显示。UI

问题分析无。

解决方法步骤一：修改图片内容

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

![内容旋转bootlogo](figures/内容旋转bootlogo.jpg)步骤二：

修改方案文件，配置属性mkro.sf.rotation以
petrel-为例，修改p1device/softwinner/petrel-p1/petrel_fvd_p1.，增添或修改以下语句：mk

13 顺时针旋转 90°

PRODUCT_PROPERTY_OVERRIDES += ro.sf.rotation=90

14 顺时针旋转 270°

PRODUCT_PROPERTY_OVERRIDES += ro.sf.rotation=270

步骤三：

修改文件
frameworks/base/services/java/com/android/server/wm/WindowManagerService.的函数java

int computeForcedAppOrientationLocked() { int req = getOrientationFromWindowsLocked(); if (req == ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED) { req = getOrientationFromAppTo-kensLocked(); } req = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE; return req; } ‘‘‘

将 ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE 修改为 ActivityInfo.SCREEN_ORIENTATION_PO

15 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.

16 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.