全志科技
Allwinner Technology

H3 项目

显示模块说明书 V1.0

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2015-01-09		正式版本

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 H3 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, 图像细节增加）
- 支持丽色系统
- 支持智能背光调节
- 支持多种图像数据格式输入(arg,yuv)
- 支持图像缩放处理
- 支持带多种制式的 HDMI 输出（720P、1080P、1080I 等）
- 支持 CVBS 信号输出（PAL 和 NTSC）

2.1.2. 相关术语介绍

介绍一些本模块特有的术语，便于阅读文档。

2.1.3. 模块配置介绍

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.

```
;auto_hpd       -   1:need hotplud for hdmi/cvbs;    0:don't hotplud for lcd
;output_type    -   default config in homlet. 0:none; 1:lcd; 2:cvbs; 4:hdmi; 8:vga
;hdmi_channel-    default config of the display channel for hdmi
;cvbs_channel -   default config of the display channel for cvbs
;hdmi_mode     -   default config of output mode of hdmi
;cvbs_mode     -   default config of output mode of cvbs 11:PAL; 14:NTSC
;check the definition(of hdmi/cvbs_mode) of disp_tv_mode in drv_display.h
;output_full      -    config the bootlogo is display fully. This should be config as 1.
;hdmi_mode_check - disable/enable the function of checking hdmi mode, 0 is disable, 1 is enable
;-----------------------------------------------------------------
[boot_disp]
auto_hpd            = 1
output_type         = 4
hdmi_channe         = 0
hdmi_mode           = 4
cvbs_channel        = 1
cvbs_mode           = 11
```

```
output_full        = 1
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         = 3
screen1_output_mod          = 4

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

(5)menuconfig 配置说明

在命令行中进入内核根目录，执行 make ARCH=arm menuconfig 进入配置主界面。并按以下步骤操作：

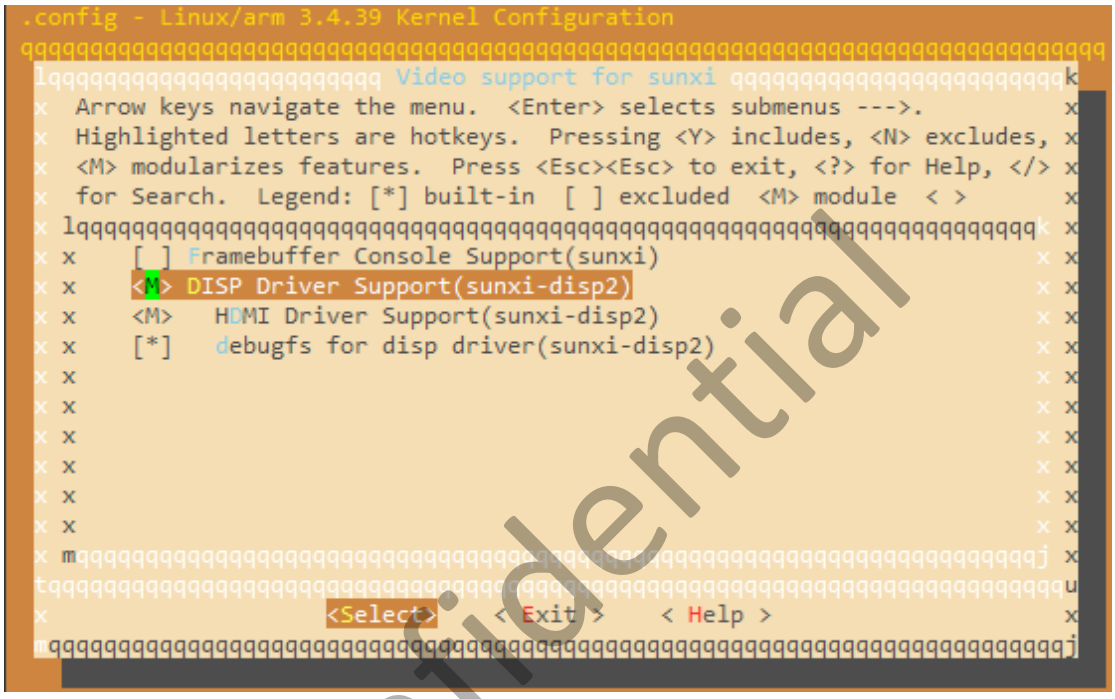


图 2.1 disp menuconfig 配置图

具体配置目录为：Device Drivers->Graphics support->Support for frame buffer devices->Video Support for sunxi -> DISP Driver Support(sunxi-disp2) / HDMI Drivers Support.

2.1.4. 源码结构介绍

```
drivers
├──video                                显示驱动目录
|    ├──fbmem.c                         framebuffer core
|    ├──sunxi                           display driver for sunxi
|    |    ├──disp2/                     disp2 的目录
|    |    |    ├──disp
|    |    |    |    ├──dev_disp.c              display driver 层
|    |    |    |    ├──dev_fb.c                framebuffer driver 层
|    |    |    |    ├──de                      bsp 层
```

```
|   |   |   |   |   ├──disp_lcd.c disp_manager.c ..
|   |   |   |   |   ├──disp_al.c                    al 层
|   |   |   |   |   |   └──lowlevel_sun*i/              lowlevel 层
|   |   |   |   |   |   ├──de_lcd.c de_rtmx.c...
|   |   |   |   |   └──disp_sys_int.              OSAL 层,与操作系统相关层
|   |   |   |   ├──lcd/                             lcd driver
|   |   |   |   |   |─lcd_src_interface.c             与 display 驱动的接口
|   |   |   |   |   |─default_panel.c ...             平台已经支持的屏驱动
|   |   |   ├──hdmi
|   |   |   |   ├──dev_hdm.c                    hdmi driver 层
|   |   |   |   ├──drv_hdm.c                    hdmi driver 层
|   |   |   |   ├──aw                           hdmi bsp 层
|   |   |   ├──tv
|   |   |   |   ├──dev_tv.c                     tv driver 层
|   |   |   |   ├──drv_tv.c                     tv driver 层
|   |   |   |   ├──de_tve.c                     tv bsp 层
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，H3 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

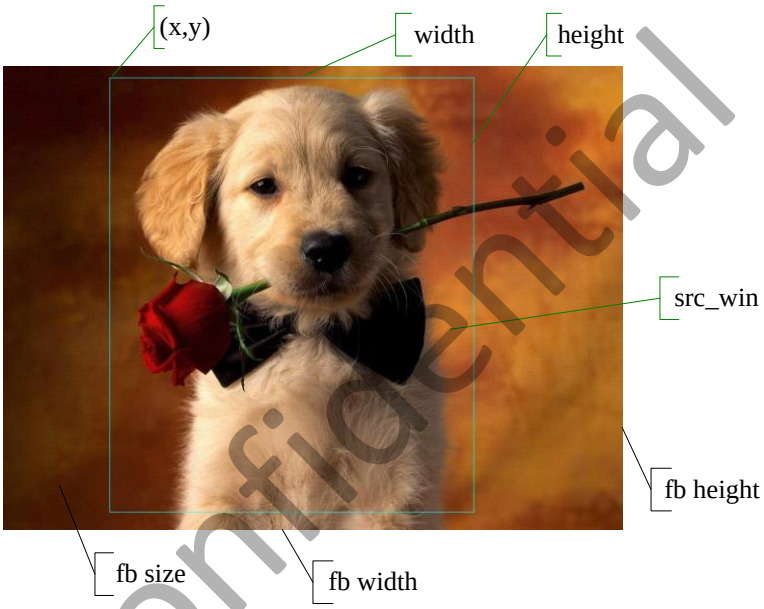
平台	H3
图层标识	以 disp, chennel, layer_id 唯一标识
图层开关	将开关当成图层参数放置于 DISP_LAYER_SET_CONFIG 接口中
图层 size	每个分量都需要设置 1 个 size
图层 align	针对每个分量需要设置其 align 位数，为 2 的倍数
图层 Crop	为 64 位参数，高 32 位为整数，低 32 位为小数
YUV MB 格式支持	不支持

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. 图层主要参数介绍

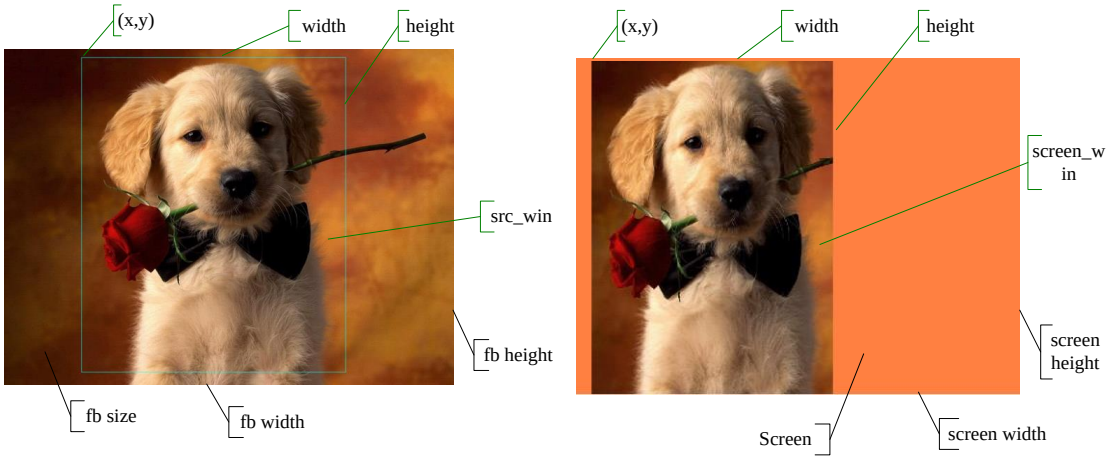
2.4.1. Size 与 src_win

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。



2.4.2. src_wind 和 screen_win

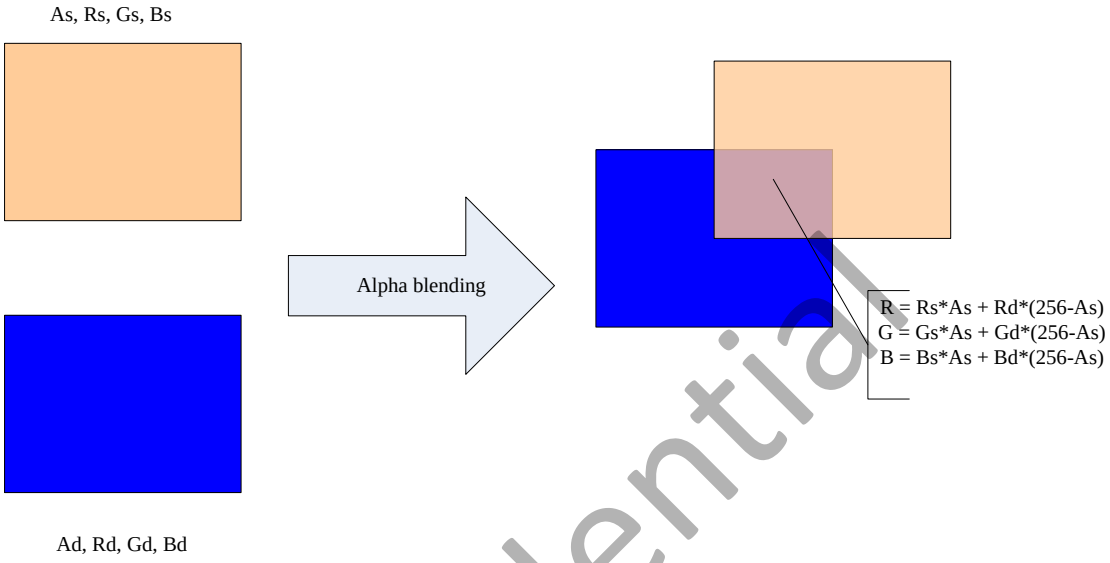
Src_win 上面已经介绍过了。Screen_win 为 src_win 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话，src_win 和 screen_win 的 width, height 是相等的，如果需要缩放，需要用 scaler_mode 的图层来显示，src_win 和 screen_win 的 width, height 可以不等。



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。



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

- H3 支持屏、HDMI 输出和 CVBS 输出，以及任意二者同时显示。

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

H3 显示驱动支持 PAL 制式或 NTSC 制式信号输出。

2.6. 接口描述

H3 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，disphd 为显示驱动句柄
unsigned int arg[3];
arg[0] = 0;//disp0
arg[1] = 1;//protect
ioctl(disphd, DISP_SHADOW_PROTECT, (void*)arg);

//do 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；
　　arg[1]为 backcolor 信息，指向 disp_color 数据结构指针；

- **RETURNS**

如果成功，返回 DIS_SUCCESS，否则，返回失败号；

- **DESCRIPTION**

该函数用于设置显示背景色。

- **DEMO**

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

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

DISP_GET_BKCOLOR

- **PROTOTYPE**

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

- **ARGUMENTS**

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

- **RETURNS**

如果成功，返回 DIS_SUCCESS，否则，返回失败号；

- **DESCRIPTION**

该函数用于获取显示背景色。

- **DEMO**

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

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

DISP_GET_SCN_WIDTH

- **PROTOTYPE**

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

- **ARGUMENTS**

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

- **RETURNS**

如果成功，返回当前屏幕水平分辨率，否则，返回失败号；

- **DESCRIPTION**

  该函数用于获取当前屏幕水平分辨率。

- **DEMO**

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

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

DISP_GET_SCN_HEIGHT

- **PROTOTYPE**

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

- **ARGUMENTS**

  hdle　　显示驱动句柄；

  cmd　　DISP_GET_SCN_HEIGHT；

  arg　　arg[0]为显示通道 0/1；

- **RETURNS**

  如果成功，返回当前屏幕垂直分辨率，否则，返回失败号；

- **DESCRIPTION**

  该函数用于获取当前屏幕垂直分辨率。

- **DEMO**

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

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

DISP_GET_OUTPUT_TYPE

- **PROTOTYPE**

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

- **ARGUMENTS**

  hdle　　显示驱动句柄；

  cmd　　DISP_GET_OUTPUT_TYPE；

  arg　　arg[0]为显示通道 0/1；

- **RETURNS**

  如果成功，返回当前显示输出类型，否则，返回失败号；

- **DESCRIPTION**

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

- **DEMO**

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

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

DISP_GET_OUTPUT

- **PROTOTYPE**

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

- **ARGUMENTS**

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

- **RETURNS**

如果成功，返回 0，否则，返回失败号；

- **DESCRIPTION**

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

- **DEMO**

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

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

DISP_VSYNC_EVENT_EN

- **PROTOTYPE**

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

- **ARGUMENTS**

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

- **RETURNS**

如果成功，返回 DIS_SUCCESS，否则，返回失败号；

- **DESCRIPTION**

该函数开启/关闭 vsync 消息发送功能。

- **DEMO**

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

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

DISP_DEVICE_SWITCH

- **PROTOTYPE**

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

- **ARGUMENTS**

  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。

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，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);
```

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);
```

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);
```

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**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于使能图像后处理功能。

- **DEMO**

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

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

DISP_ENHANCE_DISABLE

- **PROTOTYPE**

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

- **ARGUMENTS**

  hdle　　显示驱动句柄；

  cmd　　DISP_ENHANCE_DISABLE

  arg　　arg[0]为显示通道 0/1；

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于关闭图像后处理功能。

- **DEMO**

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

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

DISP_ENHANCE_DEMO_ENABLE

- **PROTOTYPE**

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

- **ARGUMENTS**

  hdle　　显示驱动句柄；

  cmd　　DISP_ENHANCE_DEMO_ENABLE

  arg　　arg[0]为显示通道 0/1；

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

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

- **DEMO**

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

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

DISP_ENHANCE_DEMO_DISABLE

- **PROTOTYPE**

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

- **ARGUMENTS**

hdle　　显示驱动句柄；

cmd　　DISP_ENHANCE_DEMO_DISABLE

arg　　arg[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于关闭图像后处理演示模式，开启后，在屏幕会出现左边进行后处理，右边未处理的图像画面，方便对比效果。

- **DEMO**

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

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

2.7.Data Structure

2.7.1. disp_fb_info

- **PROTOTYPE**

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

- **MEMBERS**

addr　　:frame buffer 的内容地址，对于 interleaved 类型，只有 addr[0]有效；planar 类型，三个都有效；UV combined 的类型 addr[0],addr[1]有效

```
size            :size of framebuffer,单位为 pixel
align           : 对齐位宽，为 2 的指数
format          :pixel format,详见 disp_pixel_format
color_space     :color space mode,详见 disp_cs_mode
b_trd_src:      1:3D source; 0：2D source
trd_mode        :source 3D mode, 详见 disp_3d_src_mode
trd_right_addr  :used when in frame packing 3d mode
crop            :用于显示的 buffer 裁减区
flags           : 标识 2D 或 3D 的 buffer
scan            :标识描述类型，progress, interleaved
```

- **DESCRIPTION**

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

2.7.2. disp_layer_info

- **PROTOTYPE**

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

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

- **MEMBERS**

```
mode         :图层的模式，详见 disp_layer_mode
zorder       :layer zorder,优先级高的图层可能会覆盖优先级低的图层;
alpha_mode   :0:pixel alpha, 1:global alpha, 2:global pixel alpha
alpha_value  :layer global alpha value，valid while alpha_mode(1/2)
screenn_win  :screen window，图层在屏幕上显示的矩形窗口
fb           :framebuffer 的属性，详见 disp_fb_info,valid when BUFFER_MODE
```

```
    color        :display color, valid when COLOR_MODE
    b_trd_out    :if output in 3d mode,used for scaler layer
    out_trd_mode:output 3d mode,详见 disp_3d_out_mode
    id      :frame id, 设置给驱动的图像帧号，可以通过 DISP_LAYER_GET_FRAME_ID 获取当前显示
的帧号，以做一下特定的处理，比如释放掉已经显示完成的图像帧 buffer
```

- **DESCRIPTION**

disp_layer_info 用于描述一个图层的属性信息。

2.7.3. disp_layer_config

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_layer_config 用于描述一个图层配置的属性信息。

2.7.4. disp_color_info

- **PROTOTYPE**

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

- **MEMBERS**

```
    alpha        :颜色的透明度
    red          :红
    green        :绿
    blue         :蓝
```

- **DESCRIPTION**

disp_color_info 用于描述一个颜色的信息。

2.7.5. disp_rect

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_rect 用于描述一个矩形窗口的信息。

2.7.6. disp_position

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_position 用于描述一个坐标的信息。

2.7.7. disp_rectsz

- **PROTOTYPE**

```
typedef struct
{
 u32 width;
```

```
        u32 height;
        }disp_rectsz;
```

- **MEMBERS**

```
   width        :宽
   height       :高
```

- **DESCRIPTION**

*disp_rectsz*用于描述一个矩形尺寸的信息。

2.7.8. disp_pixel_format

- **PROTOTYPE**

```
        typedef enum
        {
        DISP_FORMAT_ARGB_8888                   = 0x00,//MSB  A-R-G-B  LSB
        DISP_FORMAT_ABGR_8888                   = 0x01,
        DISP_FORMAT_RGBA_8888                   = 0x02,
        DISP_FORMAT_BGRA_8888                   = 0x03,
        DISP_FORMAT_XRGB_8888                   = 0x04,
        DISP_FORMAT_XBGR_8888                   = 0x05,
        DISP_FORMAT_RGBX_8888                   = 0x06,
        DISP_FORMAT_BGRX_8888                   = 0x07,
        DISP_FORMAT_RGB_888                     = 0x08,
        DISP_FORMAT_BGR_888                     = 0x09,
        DISP_FORMAT_RGB_565                     = 0x0a,
        DISP_FORMAT_BGR_565                     = 0x0b,
        DISP_FORMAT_ARGB_4444                   = 0x0c,
        DISP_FORMAT_ABGR_4444                   = 0x0d,
        DISP_FORMAT_RGBA_4444                   = 0x0e,
        DISP_FORMAT_BGRA_4444                   = 0x0f,
        DISP_FORMAT_ARGB_1555                   = 0x10,
        DISP_FORMAT_ABGR_1555                   = 0x11,
        DISP_FORMAT_RGBA_5551                   = 0x12,
        DISP_FORMAT_BGRA_5551                   = 0x13,

        /* SP: semi-planar, P:planar, I:interleaved
         * UVUV: U in the LSBs;     VUVU: V in the LSBs */
        DISP_FORMAT_YUV444_I_AYUV               = 0x40,//MSB  A-Y-U-V  LSB
        DISP_FORMAT_YUV444_I_VUYA               = 0x41,//MSB  V-U-Y-A  LSB
        DISP_FORMAT_YUV422_I_YVYU               = 0x42,//MSB  Y-V-Y-U  LSB
        DISP_FORMAT_YUV422_I_YUYV               = 0x43,//MSB  Y-U-Y-V  LSB
        DISP_FORMAT_YUV422_I_UYVY               = 0x44,//MSB  U-Y-V-Y  LSB
        DISP_FORMAT_YUV422_I_VYUY               = 0x45,//MSB  V-Y-U-Y  LSB
        DISP_FORMAT_YUV444_P                    = 0x46,//MSB  P3-2-1-0 LSB,   YYYY UUUU
VVVV
```

```
        DISP_FORMAT_YUV422_P                    = 0x47,//MSB  P3-2-1-0 LSB   YYYY UU
VV
        DISP_FORMAT_YUV420_P                    = 0x48,//MSB  P3-2-1-0 LSB   YYYY U
V
        DISP_FORMAT_YUV411_P                    = 0x49,//MSB  P3-2-1-0 LSB   YYYY U
V
        DISP_FORMAT_YUV422_SP_UVUV              = 0x4a,//MSB  V-U-V-U  LSB
        DISP_FORMAT_YUV422_SP_VUVU              = 0x4b,//MSB  U-V-U-V  LSB
        DISP_FORMAT_YUV420_SP_UVUV              = 0x4c,
        DISP_FORMAT_YUV420_SP_VUVU              = 0x4d,
        DISP_FORMAT_YUV411_SP_UVUV              = 0x4e,
        DISP_FORMAT_YUV411_SP_VUVU              = 0x4f,
        }disp_pixel_format;;
```

- **MEMBERS**

```
    DISP_FORMAT_ARGB_8888:  32bpp, A 在最高位，B 在最低位
    DISP_FORMAT_YUV420_P:   planar yuv 格式，分三块存放，需三个地址，P3 在最高位。
    DISP_FORMAT_YUV422_SP_UVUV：semi-planar yuv 格式，分两块存放，需两个地址，UV 的顺序为
U 在低位，  DISP_FORMAT_YUV420_SP_UVUV 类似
    DISP_FORMAT_YUV422_SP_VUVU：semi-planar yuv 格式，分两块存放，需两个地址，UV 的顺序为
V 在低位，  DISP_FORMAT_YUV420_SP_VUVU 类似
```

- **DESCRIPTION**

disp_pixel_format 用于描述像素格式。

2.7.9. disp_buffer_flags

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_buffer_flags 用于描述 3D 源模式。

2.7.10. disp_3d_out_mode

- **PROTOTYPE**

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

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

- **MEMBERS**

```
//for lcd
DISP_3D_OUT_MODE_CI_1  : 列交织
DISP_3D_OUT_MODE_CI_2  : 列交织
DISP_3D_OUT_MODE_CI_3  : 列交织
DISP_3D_OUT_MODE_CI_4  : 列交织
DISP_3D_OUT_MODE_LIRGB : 行交织

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

- **DESCRIPTION**

disp_3d_out_mode 用于描述 3D 输出模式。

2.7.11. disp_color_space

- **PROTOTYPE**

```
typedef enum
```

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_color_space 用于描述颜色空间类型。

2.7.12. disp_output_type

- **PROTOTYPE**

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

- **MEMBERS**

```
    DISP_OUTPUT_TYPE_NONE   ：无显示输出
    DISP_OUTPUT_TYPE_LCD    ：LCD 输出
    DISP_OUTPUT_TYPE_TV     ：TV 输出
    DISP_OUTPUT_TYPE_HDMI   ：HDMI 输出
    DISP_OUTPUT_TYPE_VGA    ：VGA 输出
```

- **DESCRIPTION**

disp_output_type 用于描述显示输出类型。

2.7.13. disp_tv_mode

- **PROTOTYPE**

```
typedef enum
{
DISP_TV_MOD_480I                = 0,
DISP_TV_MOD_576I                = 1,
DISP_TV_MOD_480P                = 2,
DISP_TV_MOD_576P                = 3,
DISP_TV_MOD_720P_50HZ           = 4,
DISP_TV_MOD_720P_60HZ           = 5,
```

```
    DISP_TV_MOD_1080I_50HZ          = 6,
    DISP_TV_MOD_1080I_60HZ          = 7,
    DISP_TV_MOD_1080P_24HZ          = 8,
    DISP_TV_MOD_1080P_50HZ          = 9,
    DISP_TV_MOD_1080P_60HZ          = 0xa,
    DISP_TV_MOD_1080P_24HZ_3D_FP    = 0x17,
    DISP_TV_MOD_720P_50HZ_3D_FP     = 0x18,
    DISP_TV_MOD_720P_60HZ_3D_FP     = 0x19,
    DISP_TV_MOD_1080P_25HZ          = 0x1a,
    DISP_TV_MOD_1080P_30HZ          = 0x1b,
    DISP_TV_MOD_PAL                 = 0xb,
    DISP_TV_MOD_PAL_SVIDEO          = 0xc,
    DISP_TV_MOD_NTSC                = 0xe,
    DISP_TV_MOD_NTSC_SVIDEO         = 0xf,
    DISP_TV_MOD_PAL_M               = 0x11,
    DISP_TV_MOD_PAL_M_SVIDEO        = 0x12,
    DISP_TV_MOD_PAL_NC              = 0x14,
    DISP_TV_MOD_PAL_NC_SVIDEO       = 0x15,
    DISP_TV_MOD_3840_2160P_30HZ     = 0x1c,
    DISP_TV_MOD_3840_2160P_25HZ     = 0x1d,
    DISP_TV_MOD_3840_2160P_24HZ     = 0x1e,
    DISP_TV_MODE_NUM                = 0x1f,
    }disp_tv_mode;
```

- **MEMBERS**

- **DESCRIPTION**

  disp_tv_mode 用于描述 TV 输出模式。

2.7.14. disp_output

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_output 用于描述显示输出类型，模式

2.7.15. disp_layer_mode

- **PROTOTYPE**

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

- **MEMBERS**

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

- **DESCRIPTION**

disp_layer_mode 用于描述图层模式。

2.7.16. disp_scan_flags

- **PROTOTYPE**

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

- **MEMBERS**

```
DISP_SCAN_PROGRESSIVE:  逐行模式
DISP_SCAN_INTERLACED_ODD_FLD_FIRST :   隔行模式，奇数行优先
DISP_SCAN_INTERLACED_EVEN_FLD_FIRST :   隔行模式，偶数行优先
```

- **DESCRIPTION**

disp_scan_flags 用于描述显示 Buffer 的扫描方式。

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; //FB 地址
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。H3 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：

```
[boot_disp]
auto_hpd        = 1
output_type     = 4     //默认 HDMI 输出
hdmi_channel    = 0     //H3 的 HDMI 输出通道是 0
hdmi_mode       = 4     //默认 HDMI 输出 720P@50Hz
```

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/dolphin-optimus/dolphin-optimus.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/dolphin-fvd-p1/dolphin_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 (只有 A80 支持此配置)	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

H3 默认使用热插拔双显同显（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 热插拔配置文件 android/device/softwinner/dolphin-common/overlay/frameworks/base/core/res/res/values/display_configs.xml：

```
<string-array translatable="false" name="default_display_format">
    <item>404</item> <!-- HDMI 720P 50Hz -->
    <item>20b</item> <!-- PAL -->
</string-array>
    <integer-array translatable="false" name="hotplug_support">
        <item>1</item> <!-- HDMI  -->
        <item>1</item> <!-- CVBS -->
    </integer-array>
    <integer-array translatable="false" name="hotplug_revert_regist">
        <item>0</item> <!-- none  -->
        <item>0</item> <!-- none -->
    </integer-array>
```

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

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.getDisplayOutputType(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 显示类型

- **DESCRIPTION**

获取当前的显示类型

- **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.getDisplayOutputMode(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
@tyep: 显示类型
@mode: 显示模式

- **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 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: 显示模式

- **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 3D，则返回 1；否则返回 0。

- **DESCRIPTION**

查询显示设备是否 3D 模式。

- **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 3D 模式

- **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

public int getDisplayBright(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。

表 3-2 H3（双显）显示的热插拔消息处理策略

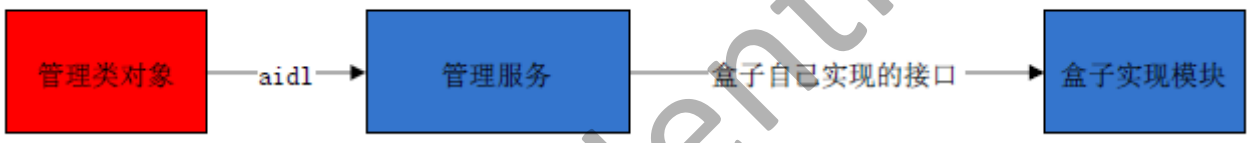
H3显示的热插拔消息处理策略			
一.显示通道选择（sys_config.fex 可配置）：HDMI-disp0，CVBS-disp1。			
二.优先级：HDMI-高，CVBS-低			
三.默认输出：hdmi-720P50HZ，cvbs-PAL。			
四.策略：			
序号	当前状态	具体操作	Android 主辅显状态
1	只有主显设备	-	主显输出到此设备
2	只有主显设备	拔掉此设备	主显仍输出到此设备（Android 认为设备还在）
3	只有主显设备	插入另一个设备	**打开辅显输出到第二个插入设备，如 HDMI 为辅显设备，则切换主辅显设备**
4	同时插着两个设备	-	HDMI 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

3.4. CMCC 显示接口

摘录 中国移动视频基地机顶盒应用服务框架对机顶盒接口要求2.1.7.1SP01.pdf 2014-07-01 10:32:27, 711 KB 的内容:

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：



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

        ……
}
}
```

4. Declaration