全志科技
Allwinner Technology

V40 项目

V40 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	2016-06-06		正式版本

目　　录

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 V40 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

2.1.2. 模块配置介绍

linux 显示驱动和 boot 显示驱动配置都包括在[disp]主键下：

```
;------------------------------------------------------------------------
;disp init configuration
;
;disp_mode            (0:screen0<screen0,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           (4:RGB655 5:RGB565 6:RGB556 7:ARGB1555
;                     8:RGBA5551 9:RGB888 10:ARGB8888 12:ARGB4444)
;fbx pixel sequence    (0:ARGB 1:BGRA 2:ABGR 3:RGBA)
;fb0_scaler_mode_enable(scaler mode enable, used FE)
;fbx_width,fbx_height  (framebuffer horizontal/vertical pixels,
;                       fix to output resolution while equal 0)
;lcdx_backlight        (lcd init backlight,the range:[0,256],default:197
;lcdx_yy               (lcd init screen bright/contrast/saturation/hue,
;                       value:0~100, default:50/50/57/50)
;lcd0_contrast        (LCD contrast, 0~100)
;lcd0_saturation      (LCD saturation, 0~100)
;lcd0_hue              (LCD hue, 0~100)
;------------------------------------------------------------------------
[disp]
disp_init_enable       = 1
disp_mode              = 0
```

```
screen0_output_type     = 1
screen0_output_mode     = 4

screen1_output_type     = 1
screen1_output_mode     = 4

fb0_format             = 0
fb0_width              = 400
fb0_height             = 1280

fb1_format             = 0
fb1_width              = 0
fb1_height             = 0

lcd0_backlight          = 50
lcd1_backlight          = 50

lcd0_bright            = 50
lcd0_contrast          = 50
lcd0_saturation         = 57
lcd0_hue               = 50

lcd1_bright           = 50
lcd1_contrast          = 50
lcd1_saturation       = 57
lcd1_hue               = 50
```

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/                           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，V40 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

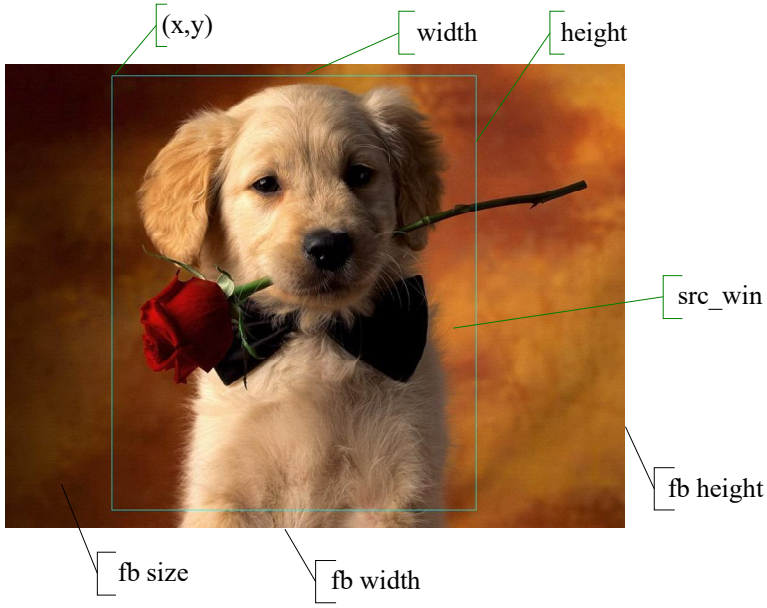
平台	V40
图层标识	以 disp, chennel, layer_id 唯一标识
图层开关	将开关当成图层参数放置于 DISP_LAYER_SET_CONFIG 接口中
图层 size	每个分量都需要设置 1 个 size
图层 align	针对每个分量需要设置其 align 位数，为 2 的倍数
图层 Crop	为 64 位参数，高 32 位为整数，低 32 位为小数
YUV MB 格式支持	不支持
PALETTE 格式支持	不支持
单色模式(无 buffer)	支持

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

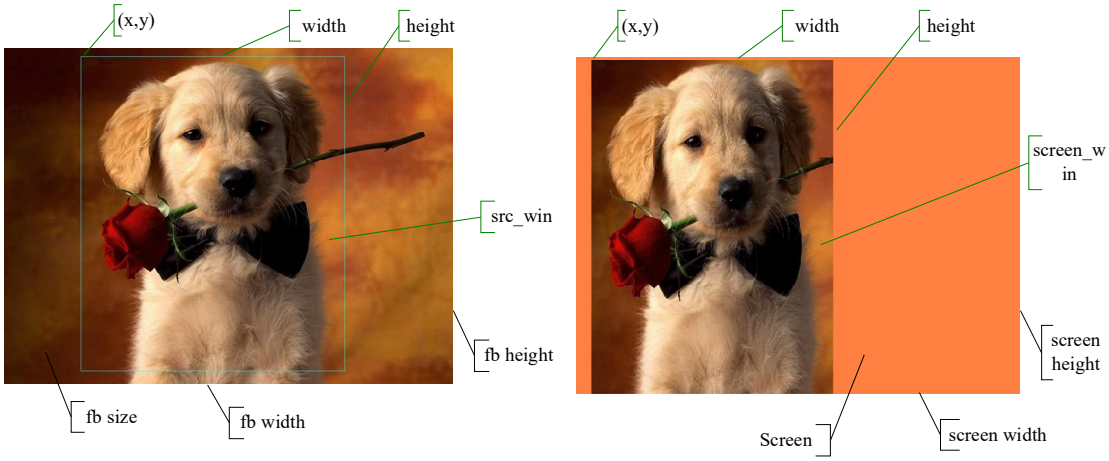
2.4.1. Size 与 src_win

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_win 和 screen_win

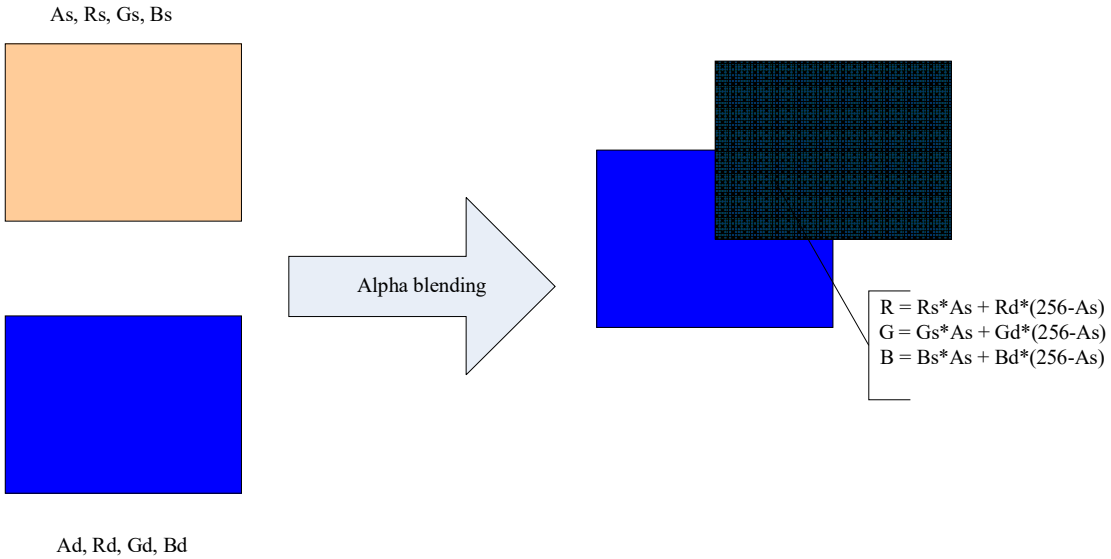
Src_win 上面已经介绍过了。Screen_win 为 src_win 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话，src_win 和 screen_win 的 width, height 是相等的，如果需要缩放，需要用 scaler_mode 的图层来显示，src_win 和 screen_win 的 width, height 可以不等。



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

- V40 默认使用 LCD 输出，支持：mipi dsi、LVDS、RGB 等接口。

2.6.接口描述

V40 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);
disphd, DISP_HDMI_SUPPORT_MODE, (void*)arg);
```

2.6.3. 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

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	描述
addr[3]	unsigned long long	address of frame buffer, single addr for interleaved fomart double addr for semi-planar fomart triple addr for planar format
size[3]	disp_rectsz	size for 3 component,unit: pixels
align[3]	unsigned int	align for 3 comonent, unit: bits(align=2^n,i.e. 1/2/4/8/16/32..)
format	disp_pixel_format	pixel format
color_space	disp_color_space	color space

trd_right_addr[3]	unsigned int	right address of 3d fb, used when in frame packing 3d mode
pre_multiply	bool	true: pre-multiply fb
crop	disp_rect64	crop rectangle boundaries
flags	disp_buffer_flags	indicate stereo or non-stereo buffer
scan	disp_scan_flags	scan type & scan order

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

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

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	描述

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

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

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	描述
x	s32	坐标 x 的值
y	s32	坐标 y 的值

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	描述
width	s32	矩形的宽
height	s32	矩形的高

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	描述
DISP_FORMAT_ARGB_8888	enum	像素格式枚举值
DISP_FORMAT_ABGR_8888	enum	像素格式枚举值
DISP_FORMAT_RGBA_8888	enum	像素格式枚举值
DISP_FORMAT_BGRA_8888	enum	像素格式枚举值
DISP_FORMAT_XRGB_8888	enum	像素格式枚举值
DISP_FORMAT_XBGR_8888	enum	像素格式枚举值
DISP_FORMAT_RGBX_8888	enum	像素格式枚举值

DISP_FORMAT_BGRX_8888	enum	像素格式枚举值
DISP_FORMAT_RGB_888	enum	像素格式枚举值
DISP_FORMAT_BGR_888	enum	像素格式枚举值
DISP_FORMAT_RGB_565	enum	像素格式枚举值
DISP_FORMAT_BGR_565	enum	像素格式枚举值
DISP_FORMAT_ARGB_4444	enum	像素格式枚举值
DISP_FORMAT_ABGR_4444	enum	像素格式枚举值
DISP_FORMAT_RGBA_4444	enum	像素格式枚举值
DISP_FORMAT_BGRA_4444	enum	像素格式枚举值
DISP_FORMAT_ARGB_1555	enum	像素格式枚举值
DISP_FORMAT_ABGR_1555	enum	像素格式枚举值
DISP_FORMAT_RGBA_5551	enum	像素格式枚举值
DISP_FORMAT_BGRA_5551	enum	像素格式枚举值
DISP_FORMAT_YUV444_I_AYUV	enum	像素格式枚举值
DISP_FORMAT_YUV444_I_VUYA	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_YVYU	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_YUYV	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_UYVY	enum	像素格式枚举值
DISP_FORMAT_YUV422_I_VYUY	enum	像素格式枚举值
DISP_FORMAT_YUV444_P	enum	像素格式枚举值
DISP_FORMAT_YUV422_P	enum	像素格式枚举值
DISP_FORMAT_YUV420_P	enum	像素格式枚举值
DISP_FORMAT_YUV411_P	enum	像素格式枚举值
DISP_FORMAT_YUV422_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV422_SP_VUVU	enum	像素格式枚举值
DISP_FORMAT_YUV420_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV420_SP_VUVU	enum	像素格式枚举值
DISP_FORMAT_YUV411_SP_UVUV	enum	像素格式枚举值
DISP_FORMAT_YUV411_SP_VUVU	enum	像素格式枚举值

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

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

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

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

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	描述
DISP_TV_MOD_480I	enum	480I 输出格式

DISP_TV_MOD_576I	enum	576I 输出格式
DISP_TV_MOD_480P	enum	480P 输出格式
DISP_TV_MOD_576P	enum	576P 输出格式
DISP_TV_MOD_720P_50HZ	enum	720P_50Hz 输出格式
DISP_TV_MOD_720P_60HZ	enum	720P_60Hz 输出格式
DISP_TV_MOD_1080I_50HZ	enum	1080I_50Hz 输出格式
DISP_TV_MOD_1080I_60HZ	enum	1080I_60Hz 输出格式
DISP_TV_MOD_1080P_24HZ	enum	1080P_24HZ 输出格式
DISP_TV_MOD_1080P_50HZ	enum	1080P_50HZ 输出格式
DISP_TV_MOD_1080P_60HZ	enum	1080P_60HZ 输出格式
DISP_TV_MOD_1080P_24HZ_3D_FP	enum	1080P_24HZ_3D_FP 输出格式
DISP_TV_MOD_720P_50HZ_3D_FP	enum	720P_50HZ_3D_FP 输出格式
DISP_TV_MOD_720P_60HZ_3D_FP	enum	720P_60HZ_3D_FP 输出格式
DISP_TV_MOD_1080P_25HZ	enum	1080P_25HZ 输出格式
DISP_TV_MOD_1080P_30HZ	enum	1080P_30HZ 输出格式
DISP_TV_MOD_PAL	enum	PAL 输出格式
DISP_TV_MOD_PAL_SVIDEO	enum	PAL_SVIDEO 输出格式
DISP_TV_MOD_NTSC	enum	NTSC 输出格式
DISP_TV_MOD_NTSC_SVIDEO	enum	SVIDEO 输出格式
DISP_TV_MOD_PAL_M	enum	PAL_M 输出格式
DISP_TV_MOD_PAL_M_SVIDEO	enum	SVIDEO 输出格式
DISP_TV_MOD_PAL_NC	enum	PAL_NC 输出格式
DISP_TV_MOD_PAL_NC_SVIDEO	enum	PAL_NC_SVIDEO 输出格式
DISP_TV_MOD_3840_2160P_30HZ	enum	3840_2160P_30HZ 输出格式
DISP_TV_MOD_3840_2160P_25HZ	enum	3840_2160P_25HZ 输出格式
DISP_TV_MOD_3840_2160P_24HZ	enum	3840_2160P_24HZ 输出格式
DISP_TV_MODE_NUM	enum	

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

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	描述
LAYER_MODE_BUFFER	enum	buffer 模式，带 buffer 的图层
LAYER_MODE_COLOR	enum	单色模式，无 buffer 的图层，

		只需要一个颜色值表示图像内容

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

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

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；
- 设置和获取横向缩放和纵向缩放；
- 显示设备管理。

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
（2）实际屏幕：即具体的显示模式
（3）SCALE 显示方式：当虚拟屏幕和实际屏幕不相等时，系统使用 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）显示方式的配置
略，V40 采用自适应 LCD 分辨率，无需额外设置。
（2）热插拔双显同显配置
略，V40 默认 LCD 输出，无需配置双显策略。
（3）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

1）内容为顺时针旋转 90°的图片

2）内容为顺时针旋转 270°的图片

（4）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，V40 显示系统支持切边功能（也称做 overscan）。配置默认切边值的方法如下：

在方案下的 mk 文件（例如 device\softwinner\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%。

（5）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。

修改 density 的方法如下：

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

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
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.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: 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 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 mode，则返回 true；不支持则 false。

- **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: 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);
```

4. Declaration