



AW Display 开发指南

版本号: 1.0

发布日期: 2024.3.13

版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.3.13	AWA02075	创建初始版本



目 录

1	前言	1
1.1	文档简介	1
1.2	目标读者	1
1.3	适用范围	1
1.4	相关术语介绍	1
1.4.1	硬件术语	1
1.4.2	软件术语	2
2	模块介绍	3
2.1	显示通路	3
2.2	显示驱动与用户空间交互	4
3	模块配置介绍	5
3.1	kernel menuconfig 配置说明	5
3.2	uboot-board.dts 配置说明	6
3.3	board.dts 配置说明	7
4	显示规格	9
4.1	各平台支持显示接口及其特性	9
4.2	最大输出分辨率和协议标准	9
4.2.1	T3 平台	9
4.2.2	V853 平台	10
4.2.3	A100/A133 平台	10
4.2.4	R528/T113 平台	10
4.2.5	H618/H616/T507 平台	10
4.2.6	A523 平台	11
4.2.7	A527/T527 平台	11
4.2.8	V821 平台	12
4.3	多显支持接口组合	12
4.3.1	A527/T527	12
5	显示驱动解析	14
5.1	源码结构介绍	14
5.2	驱动框架介绍	15
5.3	图像 Buffer 参数说明	16
5.4	显示驱动 Buffer 数据结构	16
5.5	约束条件	18
6	模块使用接口说明	19
6.1	global Interface	19

6.1.1	DISP_SHADOW_PROTECT	19
6.1.2	DISP_SET_BKCOLOR	19
6.1.3	DISP_GET_SCN_WIDTH	20
6.1.4	DISP_GET_SCN_HEIGHT	20
6.1.5	DISP_GET_OUTPUT_TYPE	20
6.1.6	DISP_GET_OUTPUT	21
6.1.7	DISP_VSYNC_EVENT_EN	21
6.1.8	DISP_DEVICE_SWITCH	21
6.1.9	DISP_DEVICE_SET_CONFIG	22
6.1.10	DISP_DEVICE_GET_CONFIG	22
6.2	layer Interface	23
6.2.1	DISP_LAYER_SET_CONFIG	23
6.2.2	DISP_LAYER_GET_CONFIG	24
6.2.3	DISP_LAYER_SET_CONFIG2	24
6.2.4	DISP_LAYER_GET_CONFIG2	25
6.3	capture interface	26
6.3.1	DISP_CAPTURE_START	26
6.3.2	DISP_CAPTURE_COMMIT	26
6.3.3	DISP_CAPTURE_STOP	27
6.3.4	DISP_CAPTURE_QUERY	27
6.4	lcd device Interface	27
6.4.1	DISP_LCD_SET_BRIGHTNESS	27
6.4.2	DISP_LCD_GET_BRIGHTNESS	28
6.4.3	DISP_LCD_SET_GAMMA_TABLE	28
6.4.4	DISP_LCD_GAMMA_CORRECTION_ENABLE	28
6.4.5	DISP_LCD_GAMMA_CORRECTION_DISABLE	29
6.5	smart backlight	29
6.5.1	DISP_SMBL_ENABLE	29
6.5.2	DISP_SMBL_DISABLE	30
6.5.3	DISP_SMBL_SET_WINDOW	30
7	调试方法	31
7.1	sysfs 节点介绍	31
7.1.1	查看显示驱动状态	31
7.1.2	使用 colorbar 测试	33
7.1.3	回写输出数据	33
7.1.4	调节画质 enhance	34
7.1.5	色温	34
7.2	debugfs 接口	34
7.2.1	配置输出设备参数	35
7.2.2	开关屏幕	35
7.2.3	开关显示	36
7.2.4	显示的休眠和唤醒	36

8	FAQ	37
8.1	黑屏（无背光）	37
8.2	黑屏（有背光）	37
8.3	绿屏	38
8.4	缺数报错	38
8.5	界面卡住	39
8.6	局部界面花屏	40
8.7	快速切换界面花屏	40
8.8	显示偏色	40
8.9	调节亮度对比度饱和度	41
8.10	如何实现双显 logo 启动	41
8.11	如何替换开机 logo	43



插 图

图 2-1	模块框图	3
图 2-2	显示驱动与用户态交互	4
图 3-1	disp2 配置	5
图 3-2	disp2 配置 2	6
图 3-3	uboot dts 参数	7
图 3-4	kernel dts 参数	8
图 4-1	显示规格	9
图 4-2	显示连接框架图	12
图 4-3	单显接口	12
图 4-4	双显接口组合	13
图 5-1	驱动框图	15
图 5-2	图像基础参数	16
图 7-1	disp2_sysfs 调试节点	32
图 8-1	显示调试节点	41
图 8-2	bmp 属性	43

表 格

表 1-1	适用产品	1
表 1-2	硬件术语	1
表 1-3	软件术语	2
表 3-1	menuconfig 配置参数	6
表 4-1	T3 最大显示输出与协议	9
表 4-2	V853 最大显示输出与协议	10
表 4-3	A100/A133 最大显示输出与协议	10
表 4-4	R528/T113 最大显示输出与协议	10
表 4-5	H618/H616/T507 最大显示输出与协议	10
表 4-6	A523 最大显示输出与协议	11
表 4-7	A527/T527 最大显示输出与协议	11
表 4-8	V821 最大显示输出与协议	12
表 5-1	disp2 驱动文件结构	14
表 7-1	display sysfs 节点属性	31

1 前言

1.1 文档简介

介绍 Sunxi 平台上 Display 驱动模块的一般使用方法及调试接口，为开发与调试提供参考。

1.2 目标读者

- Display 驱动开发人员/维护人员
- Display 模块的应用层使用者

1.3 适用范围

表 1-1: 适用产品

内核版本	驱动文件路径
Linux-4.9/Linux-5.4	lichee/{KERNEL_VER}/driver/video/fbdev/sunxi/disp2/*
Linux-5.4-ansc	bsp/drivers/video/sunxi/disp2/*
Linux-5.10 及以上	bsp/drivers/video/sunxi/disp2/*

1.4 相关术语介绍

1.4.1 硬件术语

表 1-2: 硬件术语

术语	解释
de	display engine，显示引擎，负责将输入的多图层进行叠加、混合、缩放等处理的硬件模块
tcon	timing controller，时序控制器，负责接收 de 数据信号，并将其转换为对应控制信号时序
channel	一个硬件通道，包含若干图层处理单元，可以同时处理若干（典型 4 个）格式相同的图层
layer	一个图层处理单元，可以处理一张输入图像，按支持的图像格式分 video 和 ui 类型

术语	解释
capture	截屏，将 de 的输出保存到本地文件
alpha	透明度，在混合时决定对应图像的透明度
transform	图像变换，如平移、旋转等
overlay	图像叠加，按顺序将图像叠加一起的效果。z 序大的靠近观察者，会把 z 序小的挡住
blending	图像混合，按 alpha 比例将图像合成一起的效果
enhance	图像增强，有目的地处理图像数据以达到改善图像效果的过程或方法

1.4.2 软件术语

表 1-3: 软件术语

术语	解释
fb	帧缓冲 (framebuffer) , Linux 为显示设备提供的一个接口，把显存抽象成的一种设备
al	抽象层，驱动中将底层硬件抽象成固定业务逻辑的软件层
lowlevel	底层，直接操作硬件寄存器的软件层

2 模块介绍

Sunxi 平台上 Display 驱动，对底层硬件进行抽象，为用户提供统一的接口，来更新画面和设置显示参数等。Display 驱动是一个字符设备，主要通过 file_operations 提供接口与用户态通信，设备节点为” /dev/disp “，同时对接到原生的 fb 框架，实现平滑启动，用户也可以调用 fb 框架接口来显示图像，设备节点为 “/dev/fb0” 。

2.1 显示通路

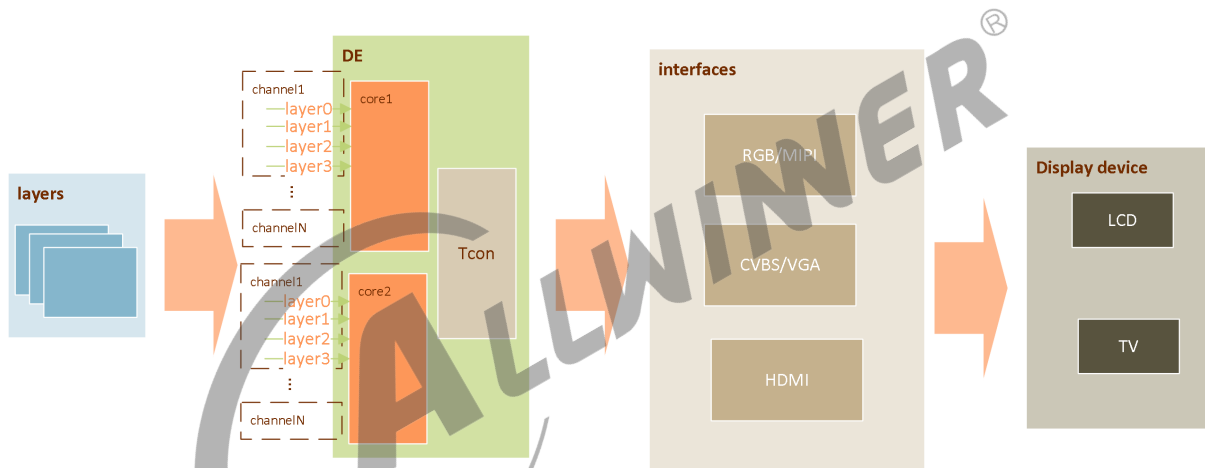


图 2-1: 模块框图

正常显示流程如下：

首先有一些已经填充内容的 buf (layers)，依据 index 会与硬件的图层处理单元一一对应，数据被读出到硬件内部后，根据配置参数做对应处理，例如：叠加 (overlay)、缩放 (scaler)、混合 (blending)、格式转换和画质处理等，送入时序控制器 (Tcon)，产生对应时序，然后送到接口 (interfaces)，根据不同协议要求处理数据，最终送入显示设备显示图像。

2.2 显示驱动与用户空间交互

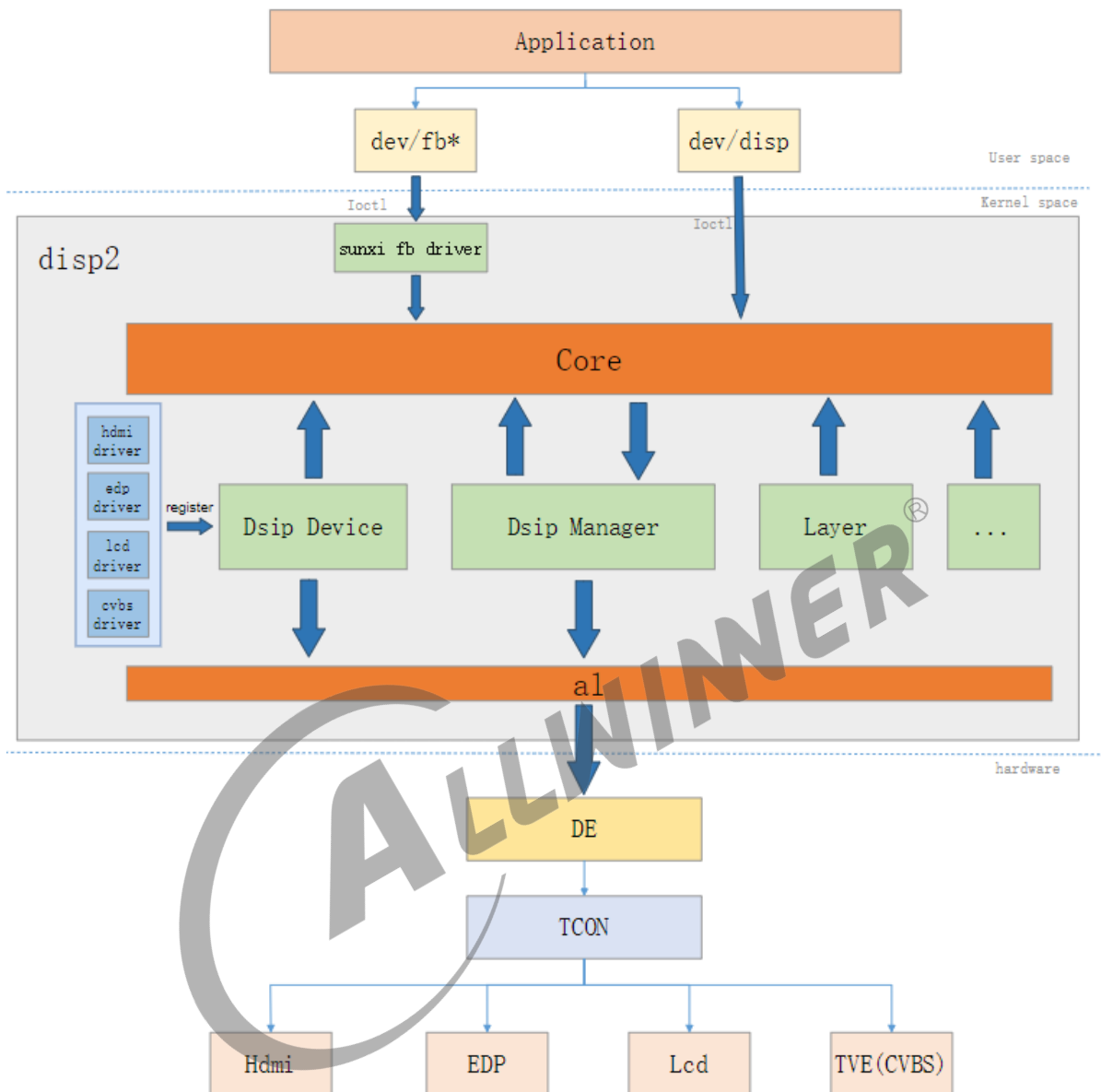


图 2-2: 显示驱动与用户态交互

3 模块配置介绍

3.1 kernel menuconfig 配置说明

在命令行中切换到 longan/tina 根目录，执行./build.sh menuconfig 进入配置主界面。

- Linux-4.9/Linux-5.4

Device Drivers > Graphics support > Frame buffer Devices > Video support for sunxi

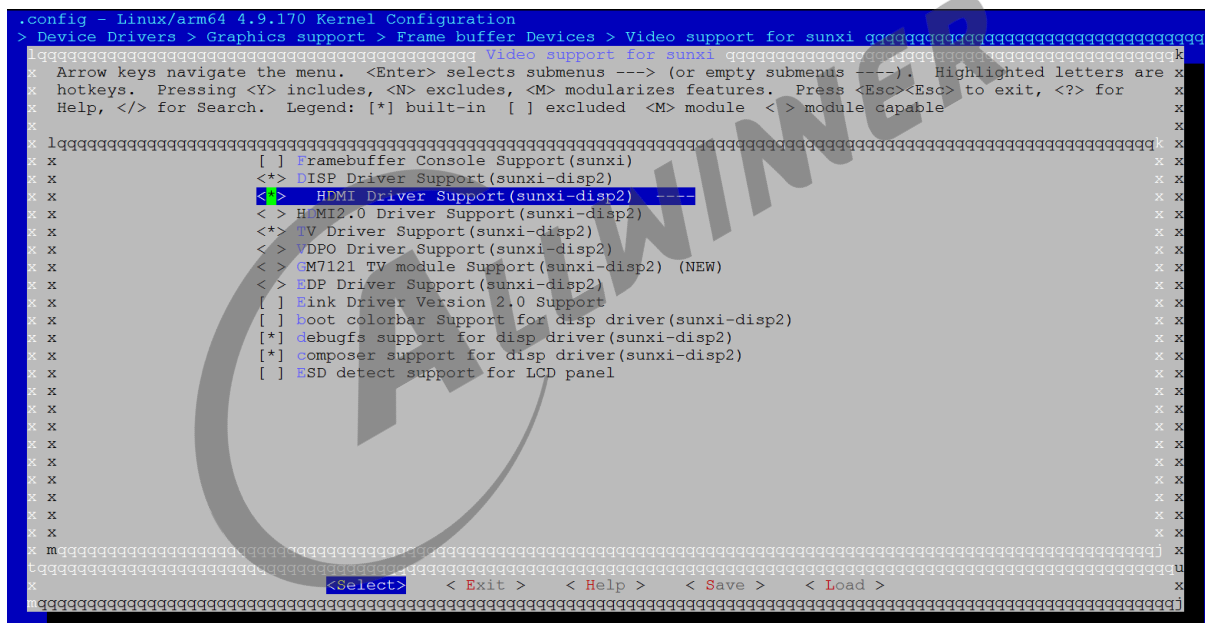


图 3-1: disp2 配置

- Linux-5.10 及以上

显示配置路径：Allwinner BSP > Device Drivers > Video Drivers

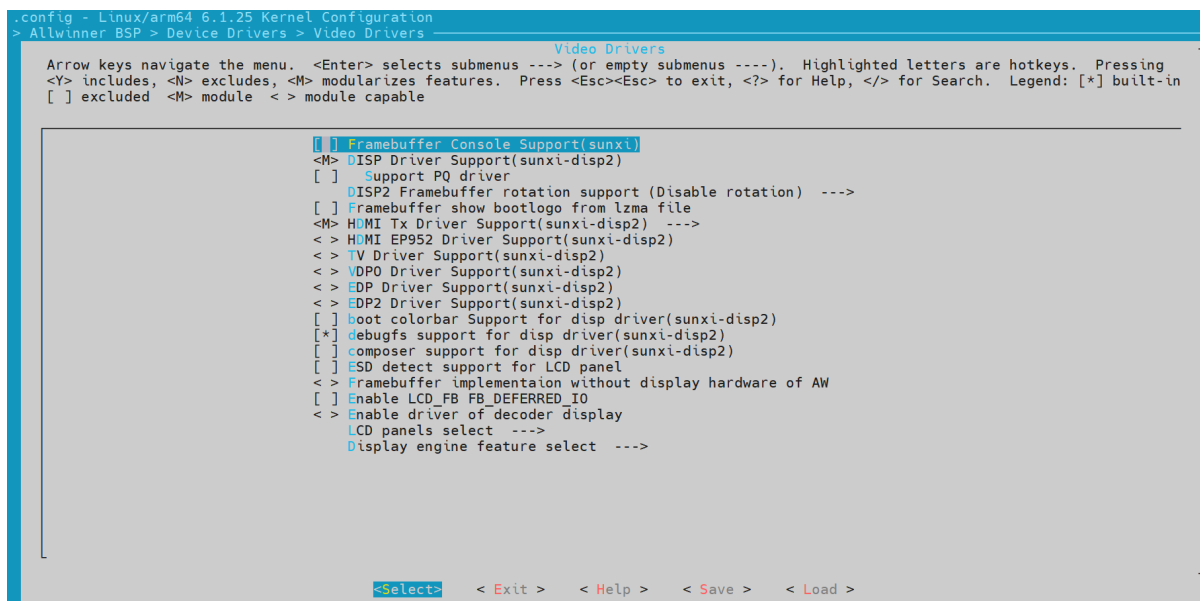


图 3-2: disp2 配置 2

配置参数配置说明：

表 3-1: menuconfig 配置参数

名字	值	说明
DISP Driver Support(sunxi-disp2)	M / * /	display 驱动开关
debugfs support for disp driver(sunxi-disp2)	* /	对接 debugfs 调试开关
composer support for disp driver(sunxi-disp2)	* /	disp2 的 fence 处理，安卓方案必须选上
interface driver	M / * /	接口驱动的开关，根据方案接口选择配置

3.2 uboot-board.dts 配置说明

配置 uboot 阶段显示节点属性，路径：device/config/chips/{IC}/configs/{BOARD}/uboot-board.dts。

节点参数配置说明：

属性	值	说明
dev_num	1/2	uboot 设备数量, AW1890 支持两个显示接口输出设备
dev(n)_screen_id	0/1	device n 连接到哪个 DE 核心, 0表示连接DE0, 1表示连接DE1
dev(n)_output_type	0: None 1: LCD 2: TV(cvbs) 4: HDMI 8: VGA 32: EDP	device n 输出类型
screen(n)_to_lcd_index	0 ~ 2	表示 DE n 连接到 lcd (0~2) , 使用 lcd (0~2) 的屏参, devn_output_type = 1 (LCD) 时有效
dev(n)_output_mode	0 ~ 92	device n 分辨率@帧率配置, 参考sunxi_display2.h: disp_tv_mode配置
dev(n)_do_hpd	0/1	device n 是否开启设备的热插拔检查
fb(n)_format	10: rgb565 bpp=16 8: rgb888 bpp=24 0: argb8888 bpp=32	fb n 存储图像 (bootlogo) 的像素格式, 与分配buf的大小相关
fb(n)_width	fb的宽度	fb n 的宽度, 单位为像素
fb(n)_height	fb的高度	fb n 的高度, 单位为像素

图 3-3: uboot dts 参数

3.3 board.dts 配置说明

配置 kernel 阶段显示节点属性, 路径: device/config/chips/{IC}/configs/{BOARD}/board.dts。

节点参数配置说明:

属性	值	说明
screen (n)_output_type	0: None 1: LCD 2: TV(CVBS) 3: HDMI 4: VGA 6: EDP	screen n 输出类型
screen(n)_to_lcd_index	0 ~ 2	表示 DE n 连接到 lcd (0~2) , 使用 lcd (0~2) 的屏参, , devn_output_type = 1 (LCD) 时有效
screen(n)_output_mode	0 ~ 92	screen n 分辨率@帧率配置, 参考sunxi_display2.h: disp_tv_mode配置
fb_num	fb的数量	fb 的数量
fb_format	8: rgb888 bpp=24 0: argb8888 bpp=32	fb 存储图像 (bootlogo) 的像素格式, 与分配buf的大小相关
fb(n)_width	fb的宽度	fb n 的宽度, 单位为像素
fb(n)_height	fb的高度	fb n 的高度, 单位为像素

图 3-4: kernel dts 参数

说明

- 1.screen (n)_{xxx} 属性在开启平滑显示功能 (带 uboot 方案) 时无效, 主要用来在不带 uboot 的方案来配置默认启动参数, 如: fast boot 方案;
- 2.fb(n)_{xxx} 这个属性在 uboot 和 kernel 阶段均存在, uboot fb 实现为精简版, kernel fb 对接到原生 fb 框架, 所以需要两套参数来灵活配置参数, 一般这两部分参数是同步的;

4 显示规格

4.1 各平台支持显示接口及其特性

	输出接口									显示特性									
SOC	RGB	LVDS	DUAL LVDS	MIPI	DUAL MIPI	EDP	DP	HDMI	CVBS	4K video	AFBD (AFBC解码)	TFBD (TFBC解码)	HDR	KSC (梯形校正)	VEP (视频增强)	DEP (画面增强)	CRC	通道数量	RCQ
T3	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗	(1v & 3u)+ (1v & 1u)	✗
V853	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	2v & 1u	✗
A100/A133	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	(2v & 2u)+ (1v & 2u)	✗
R528/T113	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	⊗	✗	(1v & 1u)+ (1v)	✗
H616/H618/T507	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	(3v + 3u) 软件分配	✓
A523	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✗	(3v + 3u) 软件分配	✓
A527/T527	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✗	(3v + 3u) 软件分配	✓

图 4-1: 显示规格

说明

1. 通道数量中 (1v & 3u) + (1v & 1u)，表示主显支持一个 video 通道，三个 ui 通道，副显支持一个 video 通道，一个 ui 通道，软件分配表示，通道类型可以由软件配置分配到两个 DE。2.RCQ, Register Command Queue，寄存器命令队列，主要用来大批量的拷贝寄存器，可以提升显示系统稳定性，与流畅度。

4.2 最大输出分辨率和协议标准

4.2.1 T3 平台

表 4-1: T3 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道：1280x800@60fps 双通道：1920x1080@60fps	支持 VESA 和 JEIDA 两种数据格式
MIPI	1920x1200@60fps	支持 MIPI DSI v1.01 和 D-PHY v1.00 协议
HDMI	1920x1080@60fps	支持 HDMI 1.4 协议
CVBS	1920x1080@60fps	支持 PAL 和 NTSC 协议

4.2.2 V853 平台

表 4-2: V853 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
MIPI	1920x1200@60fps	支持 MIPI DSI v1.02、支持 MIPI DCS v1.01 和 D-PHY v1.2 协议

4.2.3 A100/A133 平台

表 4-3: A100/A133 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1280x720@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道：1280x800@60fps 双通道：1920x1080@60fps	支持 VESA 和 JEIDA 两种数据格式
MIPI	1920x1200@60fps	支持 MIPI DSI v1.01、支持 MIPI DCS v1.01 和 D-PHY v1.00 协议

4.2.4 R528/T113 平台

表 4-4: R528/T113 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道：1280x800@60fps 双通道：1920x1080@60fps	支持 VESA 和 JEIDA 两种数据格式
MIPI	1920x1200@60fps	支持 MIPI DSI v1.01、支持 MIPI DCS v1.01 和 D-PHY v1.00 协议
HDMI	1920x1080@60fps	支持 HDMI 1.4 协议
CVBS	1920x1080@60fps	支持 PAL 和 NTSC 协议

4.2.5 H618/H616/T507 平台

表 4-5: H618/H616/T507 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道：1280x800@60fps	支持 VESA 和 JEIDA 两种数据格式

显示接口	最大输出	协议标准
	双通道：1920x1080@60fps	
HDMI	4096x2160@60fps	支持 HDMI 1.4 和 HDMI 2.0 协议
CVBS	1920x1080@60fps	支持 PAL 和 NTSC 协议

4.2.6 A523 平台

表 4-6: A523 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道： 1280x800@60fps 双通道： 1920x1080@60fps	支持 VESA 和 JEIDA 两种数据格式
MIPI	单通道： 1920x1200@60fps 双通道： 4096x2160@45fps	支持 MIPI DSI v1.02、支持 MIPI DCS v1.01 和 D-PHY v1.2 协议

4.2.7 A527/T527 平台

表 4-7: A527/T527 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1920x1080@60fps	支持 RGB888/RGB666/RGB565
LVDS	单通道：1280x800@60fps 双通道：1920x1080@60fps	支持 VESA 和 JEIDA 两种数据格式
MIPI	单通道：1920x1200@60fps 双通道：4096x2160@45fps	支持 MIPI DSI v1.02、支持 MIPI DCS v1.01 和 D-PHY v1.2 协议
HDMI	1920x1080@60fps	支持 HDMI 1.4 和 HDMI 2.0 协议
EDP	4090x2160@30fps/ 2560x1600@60fps	支持 edp 1.3 协议

4.2.8 V821 平台

表 4-8: V821 最大显示输出与协议

显示接口	最大输出	协议标准
RGB	1280x800@60fps	支持 RGB888/RGB666/RGB565
SRGB/I8080	800x480@60fps	支持 serial RGB & i8080 接口

4.3 多显支持接口组合

4.3.1 A527/T527

- 显示接口内部连接关系

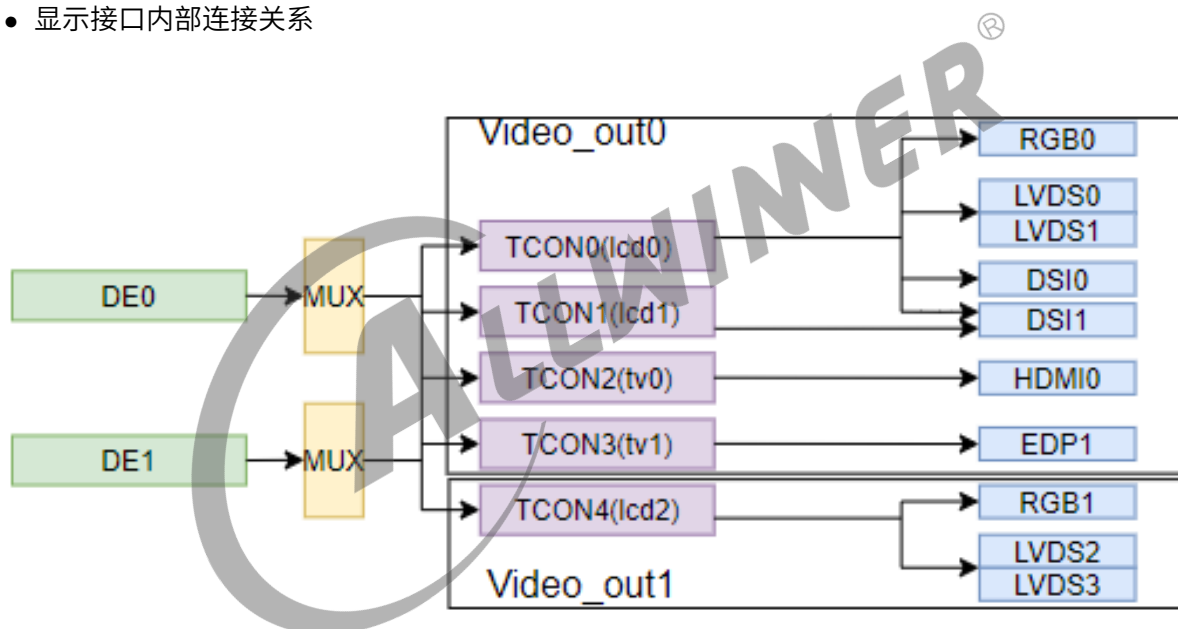


图 4-2: 显示连接框架图

- 单显接口

单显														
显示接口	RGB0	VGA0 (RGB0挂VGA转接IC)	RGB1	VGA1 (RGB1挂VGA转接IC)	MIPI-DSI0	MIPI-DSI1	LVDS0	LVDS1	LVDS2	LVDS3	Dual-LVDS0	Dual-LVDS1	Dual-MIPI-DSI	HDMI

图 4-3: 单显接口

- 双显接口组合

双显																
显示接口	RGB0	VGA0 (RGB0转VGA转 接IC)	RGB1	VGA1 (RGB1转VGA转 接IC)	MIPI-DSIO	MIPI-DSI1	LVDS0	LVDS1	LVDS2	LVDS3	Dual-LVDS0	Dual-LVDS1	Dual-MIPI- DSI	HDMI	eDP	typec-DP out (eDP转 typec-DP)
RGB0	/	/	RGB0 & RGB1	RGB0 & VGA1	/	/	/	/	RGB0 & LVDS2	RGB0 & LVDS3	/	RGB0 & Dual-LVDS1	/	RGB0 & HDMI	RGB0 & eDP	RGB0 & typec-DP
VGA0 (RGB0转VGA转 接IC)	/	/	VGA0 & RGB1	VGA0 & VGA1	/	/	/	/	VGA0 & LVDS2	VGA0 & LVDS3	/	VGA0 & Dual-LVDS1	/	VGA0 & HDMI	VGA0 & eDP	VGA0 & typec-DP
RGB1			/	/	RGB1 & MIPI-DSIO	RGB1 & MIPI-DSI1	RGB1 & LVDS0	RGB1 & LVDS1	/	/	RGB1 & Dual-LVDS0	/	RGB1 & Dual-MIPI-DSI	RGB1 & HDMI	RGB1 & eDP	RGB1 & typec-DP
VGA1 (RGB1转VGA转 接IC)			/	/	VGA1 & MIPI-DSIO	VGA1 & MIPI-DSI1	VGA1 & LVDS0	VGA1 & LVDS1	/	/	VGA1 & Dual-LVDS0	/	VGA1 & Dual-MIPI-DSI	VGA1 & HDMI	VGA1 & eDP	VGA1 & typec-DP
MIPI-DSIO					/	MIPI-DSIO & MIPI-DSI1	/	/	MIPI-DSIO & LVDS2	MIPI-DSIO & LVDS3	/	/	/	MIPI-DSIO & HDMI	MIPI-DSIO & eDP	MIPI-DSIO & typec-DP
MIPI-DSI1						/	/	/	MIPI-DSI1 & LVDS2	MIPI-DSI1 & LVDS3	/	MIPI-DSI1 & Dual-LVDS1	/	MIPI-DSI1 & HDMI	MIPI-DSI1 & eDP	MIPI-DSI1 & typec-DP
LVDS0							/	LVDS0 & LVDS1 (同)	LVDS0 & LVDS2	LVDS0 & LVDS3	/	LVDS0 & Dual-LVDS1	/	LVDS0 & HDMI	LVDS0 & eDP	LVDS0 & typec-DP
LVDS1								/	LVDS1 & LVDS2	LVDS1 & LVDS3	/	LVDS1 & Dual-LVDS1	/	LVDS1 & HDMI	LVDS1 & eDP	LVDS1 & typec-DP
LVDS2									/	LVDS2 & LVDS3 (同)	LVDS2 & Dual-LVDS0	/	LVDS2 & Dual-MIPI-DSI	LVDS2 & HDMI	LVDS2 & eDP	LVDS2 & typec-DP
LVDS3										/	LVDS3 & Dual-LVDS0	/	LVDS3 & Dual-MIPI-DSI	LVDS3 & HDMI	LVDS3 & eDP	LVDS3 & typec-DP
Dual-LVDS0											/	Dual-LVDS0 & Dual-LVDS1	/	Dual-LVDS0 & HDMI	Dual-LVDS0 & eDP	Dual-LVDS0 & typec-DP
Dual-LVDS1												/	Dual-LVDS1 & Dual-MIPI-DSI	Dual-LVDS1 & HDMI	Dual-LVDS1 & eDP	Dual-LVDS1 & typec-DP
Dual-MIPI-DSI													/	Dual-MIPI-DSI & HDMI	Dual-MIPI-DSI & eDP	Dual-MIPI-DSI & typec-DP
HDMI														/	HDMI & eDP	HDMI & typec-DP
eDP															/	/
typec-DP out (eDP转typec- DP转接IC)															/	/

NOTE:

1、Dual-LVDS0、Dual-LVDS1：物理上连接一个dual link lane lvds屏（Rlane 屏幕），（LVDS0 + LVDS1、LVDS2 + LVDS3 组合的 8 lane lvds 单个屏幕）
（相关配置见《Linux_LCD_开发指南》-> LVDS dual link 典型配置->场景1）；

2、LVDS0 & LVDS1（同）和 LVDS2 & LVDS3（同）这两个组合只用到单con，硬件上LVDS0和LVDS1使用同一个con输出，LVDS2和LVDS3使用同一个con输出。当使用这种组合时，相当于将LVDS0和LVDS2组成一个dual LVDS，物理上连接两个屏，每个屏各自 4 条 lane，两个屏是一样型号，分辨率和 timing 一样，最终一个dual LVDS接两个相同型号屏幕（4lane 屏）实现双显，若此时另一个con也用起来，可以组成三显，详见《三显》组合表格。

3、查找方法，横着找两个需要的显示接口，竖着找相同的两个显示接口，若有相交的点且有列举组合，则这两个接口可以组合为双显，反之亦然。

图 4-4: 双显接口组合

5 显示驱动解析

5.1 源码结构介绍

linux4.9/5.4 内核路径：/kernel/{KERNEL_VER}/drivers/video/fbdev/sunxi/*

linux5.10 及以上内核路径：/bsp/drivers/video/sunxi/*

文件结构如下：

表 5-1: disp2 驱动文件结构

文件目录	分层	说明
disp2/disp/	display driver	显示驱动框架对接，fb、sysfs、debugfs、ftrace 等
disp2/disp/de/	framework	显示的核心层，实现基础组件功能，与组件之间的通信机制
disp2/disp/de/lowlevel_XXX/	al/lowlevel	底层，对接到芯片寄存器操作，并通过 al 层透出统一接口
disp2/disp/lcd/	device driver	lcd 设备屏驱动
disp2/edp*/	device driver	edp 设备驱动
disp2/hdmi*/	device driver	hdmi 设备驱动
disp2/tv/	device driver	tv 设备驱动

5.2 驱动框架介绍

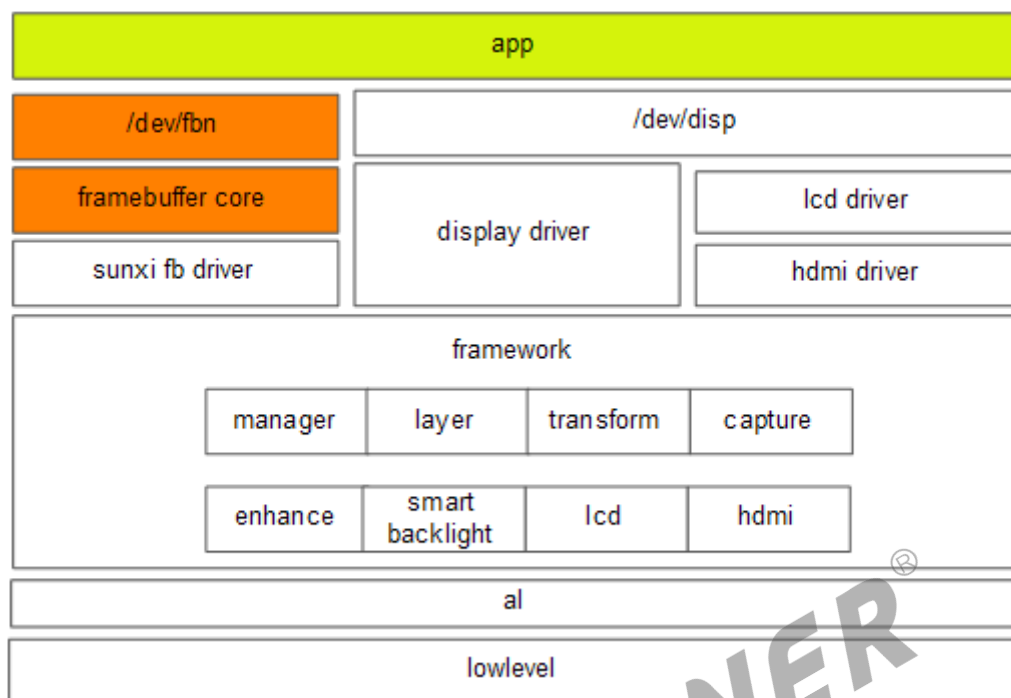


图 5-1: 驱动框图

显示驱动可划分为三个层面：驱动层，框架层及底层。底层与图形硬件相接，主要负责将上层配置的功能参数转换成硬件所需要的参数，并配置到相应寄存器中。显示框架层对底层进行抽象封装成一个个的功能模块。驱动层对外封装功能接口，通过内核向用户空间提供相应的设备结点及统一的接口。在驱动层，分为三个驱动：framebuffer 驱动，disp 驱动，lcd 驱动。framebuffer 驱动与 framebuffer core 对接，实现 linux 标准的 framebuffer 接口。disp 驱动是整个显示驱动中的核心驱动模块，所有的接口都由 disp 驱动来提供，包括 lcd 的接口。

5.3 图像 Buffer 参数说明

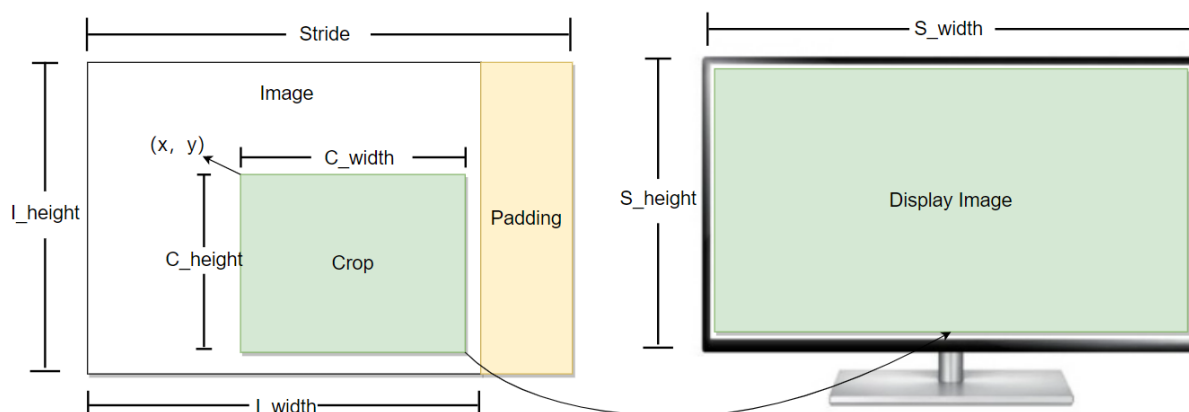


图 5-2: 图像基础参数

- Image

表示一块图像 buf，有宽高 I_width , I_height ，实际的图像一般会有对齐要求，提高处理效率，实际的宽度会对齐为 Stride，对齐多余出来的部分为 Padding。

- Crop

用做图像裁剪功能，有 $x&y$ 参数以图像 $(0, 0)$ 像素为基点的偏移，有 C_width , C_height 宽高参数。

- Screen_win

显示区域，表示最终显示图像的大小，有 $x&y$ 参数以屏幕设备 $(0, 0)$ 像素为基点的偏移，有 S_width , S_height 宽高参数。

说明

当 $C_width \neq S_width$ ，或者 $C_height \neq S_height$ ，图像会进行缩放处理。

5.4 显示驱动 Buffer 数据结构

显示驱动主要通过 ioctl 实现与应用层交互，常用数据结构定义在 `bsp/include/video/sunxi_display2.h` 中，应用层在程序中引用这个文件后，通过填充 `sunxi_display2.h` 中的数据结构后，通过 ioctl 接口传递到驱动，实现各种功能，这里讲解最常用的送显功能中 buf（图层）数据结构的含义。

```
// 抽象一个硬件 layer
struct disp_layer_config2 {
    struct disp_layer_info2 info;
    bool enable;
    unsigned int channel;
    unsigned int layer_id;
};

// 硬件 layer 的信息
struct disp_layer_info2 {
    unsigned char    zorder;
    unsigned char    alpha_mode;
    unsigned char    alpha_value;
    struct disp_rect  screen_win;
    union {
        unsigned int    color;
        struct disp_fb_info2 fb;
    };
    ...
};

// 抽象一块 framebuffer buffer
struct disp_fb_info2 {
    int            fd;
    struct disp_rectsz  size[3];
    unsigned int    align[3];
    enum disp_pixel_format format;
    struct disp_rect64  crop;
    ...
};
```

• struct disp_fb_info2

描述一块图像 buf 信息，参数：

@fd：图像文件句柄；

@size：图像大小，对应到**图像参数**的 l_width & l_height，[3] 用来存储 yuv panel 格式的 uv 分量；

@align：图像对齐信息，以 byte 为单位，对齐后 size = stride；

@format：图像像素格式；

@crop：图像 crop 信息，对应到**图像参数**的 Crop；

• struct disp_layer_info2

描述一块硬件 layer 信息，参数：

@zorder：图像叠加 z 序，大的覆盖小的；

@alpha_mode：图像混合模式，通道间混合；

@alpha_value: 混合全局 alpha 值;

@screen_win: 图像显示区域, 对应到图像参数的 S_width & S_height;

• struct disp_layer_config2

抽象一个硬件 layer, 参数:

@enable: 表示图层使能状态, 1: enable, 0: disable;

@channel: 通道 id, 用来索引硬件的通道;

@layer_id: 图层 id, 用来索引硬件的图层;

5.5 约束条件

1. RGB 格式没有透明度 alpha 分量的 layer 可以放在一个 channel, 优先从 channel0 开始分配;
2. 不交叠的 layer 可以放在一个 channel;
3. 每个 layer 分配到同一 channel 必须是同一缩放比例的;
4. 分配 layer 优先从 channel0 开始, video 优先分配 video channel;
5. 缩放都需要满足 scaler 缩放性能要求, 缩放比例从 1/16x 到 32x;
6. 3D layer 需占 2 个 layer;
7. 格式说明:

1. UI channel 格式请对照 Ui 通道支持格式, 同一 channel 可以有不同 RGB 格式;
2. VI channel 格式请对照 Video 通道支持格式, 同一 channel, 图层输入格式要么是 RGB, 要么是 YUV, 当输入为 RGB 时, 4 个 layer 可以是不同的 RGB 分量格式; 当输入为 YUV 时, 4 个 layer 只能是同一 YUV 分量格式;

6 模块使用接口说明

Sunxi 平台下显示驱动给用户提供了众多功能接口，主要通过 ioctl 与用户层交互，可对图层、画质、接口配置等显示资源进行操作。

6.1 global Interface

6.1.1 DISP_SHADOW_PROTECT

- 作用：DISP_SHADOW_PROTECT (1) 与 DISP_SHADOW_PROTECT (0) 配对使用，在两个接口调用之间的接口调用将不马上执行，而等到调用 DISP_SHADOW_PROTECT (0) 后才一并执行。
- 参数：handle：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。
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 启动 cache, dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 屏0
arg[1] = 1; // protect
ioctl(dispfd, DISP_SHADOW_PROTECT, (void*)arg);
// do something other
arg[1] = 0;
ioctl(dispfd, DISP_SHADOW_PROTECT, (void*)arg);
```

6.1.2 DISP_SET_BKCOLOR

- 作用：该函数用于设置显示背景色
- 参数：handle：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。
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 设置显示背景色, dispfd 为显示驱动句柄, sel 为屏0/1
disp_color bk;
unsigned long arg[3];
bk.red = 0xff;
bk.green = 0x00;
bk.blue = 0x00;
arg[0] = 0;
arg[1] = (unsigned long)&bk;
ioctl(dispfd, DISP_SET_BKCOLOR, (void*)arg);
```

6.1.3 DISP_GET_SCN_WIDTH

- 作用：该函数用于获取当前屏幕水平分辨率。
- 参数：handle：显示驱动句柄 cmd：DISP_GET_SCN_WIDTH arg：arg[0] 为显示通道 0/1
- 返回：正值：成功并返回当前屏幕水平分辨率其他：失败号
- 示例

```
// 获取屏幕水平分辨率
unsigned int screen_width;
unsigned long arg[3];
arg[0] = 0;
screen_width = ioctl(dispfd, DISP_GET_SCN_WIDTH, (void*)arg);
```

6.1.4 DISP_GET_SCN_HEIGHT

- 作用：该函数用于获取当前屏幕垂直分辨率
- 参数：handle：显示驱动句柄 cmd：DISP_GET_SCN_HEIGHT arg：arg[0] 为显示通道 0/1
- 返回：正值：成功并返回当前屏幕垂直分辨率其他：失败号
- 示例

```
// 获取屏幕垂直分辨率
unsigned int screen_height;
unsigned long arg[3];
arg[0] = 0;
screen_height = ioctl(dispfd, DISP_GET_SCN_HEIGHT, (void*)arg);
```

6.1.5 DISP_GET_OUTPUT_TYPE

- 作用：该函数用于获取当前显示输出类型（LCD，TV，HDMI，VGA，EDP，NONE）
- 参数：handle：显示驱动句柄 cmd：DISP_GET_OUTPUT_TYPE arg：arg[0] 为显示通道 0/1
- 返回：正值：成功，返回当前显示输出类型其他：失败号
- 示例

```
// 获取当前显示输出类型
disp_output_type output_type;
unsigned long arg[3];
arg[0] = 0;
output_type = (disp_output_type)ioctl(dispfd, DISP_GET_OUTPUT_TYPE, (void*)arg);
```

6.1.6 DISP_GET_OUTPUT

- 作用：该函数用于获取当前显示输出类型及模式（LCD，TV，HDMI，VGA，EDP，NONE）
- 参数：handle：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
- 返回：0：成功其他：失败号
- 示例

```
//获取当前显示输出类型
unsigned long arg[3];
struct disp_output output;
enum disp_output_type type;
enum disp_tv_mode mode;
arg[0] = 0;
arg[1] = (unsigned long)&output;
ioctl(dispfd, DISP_GET_OUTPUT, (void*)arg);
type = (enum disp_output_type)output.type;
mode = (enum disp_tv_mode)output.mode;
```

6.1.7 DISP_VSYNC_EVENT_EN

- 作用：该函数开启/关闭 vsync 消息发送功能
- 参数：handle：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
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
//开启/关闭vsync 消息发送功能，dispfd 为显示驱动句柄，sel 为屏0/1
unsigned long arg[3];
arg[0] = 0;
arg[1] = 1;
ioctl(dispfd, DISP_VSYNC_EVENT_EN, (void*)arg);
```

6.1.8 DISP_DEVICE_SWITCH

- 作用：该函数用于切换输出类型

- 参数：handle：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
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 切换
unsigned long arg[3];
arg[0] = 0;
arg[1] = (unsigned long)DISP_OUTPUT_TYPE_HDMI;
arg[2] = (unsigned long)DISP_TV_MOD_1080P_60HZ;
ioctl(dispfd, 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。

6.1.9 DISP_DEVICE_SET_CONFIG

- 作用：该函数用于切换输出类型并设置输出设备的属性参数
- 参数：handle：显示驱动句柄 cmd：DISP_DEVICE_SET_CONFIG arg：arg[0] 为显示通道 0/1；arg[1] 为指向 disp_device_config 的指针
- 返回：DIS_SUCCESS：成功其他：失败
- 示例

```
// 切换输出类型并设置输出设备的属性参数
unsigned long arg[3];
struct disp_device_config config;
config.type = DISP_OUTPUT_TYPE_HDMI;
config.mode = DISP_TV_MOD_1080P_60HZ;
config.format = DISP_CSC_TYPE_YUV420;
config.bits = DISP_DATA_10BITS;
config.eotf = DISP_EOTF_SMPTE2084;
config.cs = DISP_BT2020NC;
arg[0] = 0;
arg[1] = (unsigned long)&config;
ioctl(dispfd, DISP_DEVICE_SET_CONFIG, (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。

6.1.10 DISP_DEVICE_GET_CONFIG

- 作用：该函数用于获取当前输出类型及相关的属性参数
- 参数：handle：显示驱动句柄 cmd：DISP_DEVICE_GET_CONFIG arg：arg[0] 为显示通道 0/1；arg[1] 为指向 disp_device_config 的指针

- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 获取当前输出类型及相关的属性参数
unsigned long arg[3];
struct disp_device_config config;
arg[0] = 0;
arg[1] = (unsigned long)&config;
ioctl(dispfd, DISP_DEVICE_GET_CONFIG, (void*)arg);
```

📖 说明

说明：如果返回的 type 是 DISP_OUTPUT_TYPE_NONE，表示当前输出显示通道为关闭状态。

6.2 layer Interface

6.2.1 DISP_LAYER_SET_CONFIG

- 作用：该函数用于设置多个图层信息
- 参数：handle：显示驱动句柄 cmd：DISP_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
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
struct
{
    disp_layer_info info,
    bool enable;
    unsigned int channel,
    unsigned int layer_id,
}disp_layer_config;
// 设置图层参数，dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_layer_config config;
unsigned int width = 1280;
unsigned int height = 800;
unsigned int ret = 0;
memset(&config, 0, sizeof(struct disp_layer_config));
config.channnel = 0; // blending channel
config.layer_id = 0; // layer index in the blending channel
config.info.enable = 1;
config.info.mode = LAYER_MODE_BUFFER;
config.info.fb.addr[0] = (unsigned long long)mem_in; // FB 地址
config.info.fb.size[0].width = width;
config.info.fb.align[0] = 4; // bytes
config.info.fb.format = DISP_FORMAT_ARGB_8888; //DISP_FORMAT_YUV420_P
config.info.fb.crop.x = 0;
config.info.fb.crop.y = 0;
config.info.fb.crop.width = ((unsigned long)width) << 32; // 定点小数。高32bit 为整数，低32bit 为小数
config.info.fb.crop.height = ((unsigned long)height) << 32; // 定点小数。高32bit 为整数，低32bit 为小数
config.info.fb.flags = DISP_BF_NORMAL;
```

```
config.info.fb.scan = DISP_SCAN_PROGRESSIVE;
config.info.alpha_mode = 2; // global pixel alpha
config.info.alpha_value = 0xff; // global alpha value
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 long)&config;
arg[2] = 1; // one layer
ret = ioctl(dispfd, DISP_LAYER_SET_CONFIG, (void*)arg);
```

6.2.2 DISP_LAYER_GET_CONFIG

- 作用：该函数用于获取图层参数
- 参数：handle：显示驱动句柄 cmd：DISP_LAYER_GET_CONFIG arg：arg[0] 为显示通道 0/1；arg[1] 为图层配置参数指针；arg[2] 为需要获取配置的图层数目
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 设置图层参数，dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_layer_config config;
memset(&config, 0, sizeof(struct disp_layer_config));
arg[0] = 0; // disp
arg[1] = (unsigned long)&config;
arg[2] = 1; // layer number
ret = ioctl(dispfd, DISP_GET_LAYER_CONFIG, (void*)arg);
```

6.2.3 DISP_LAYER_SET_CONFIG2

- 作用：该函数用于设置多个图层信息，注意该接口只接受 disp_layer_config2 的信息
- 参数：handle：显示驱动句柄 cmd：DISP_SET_LAYER_CONFIG2 arg：arg[0] 为显示通道 0/1；arg[1] 为图层配置参数（disp_layer_config2）的指针；arg[2] 为需要配置的图层数目
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
struct
{
    disp_layer_info info,
    bool enable;
    unsigned int channel,
    unsigned int layer_id,
}disp_layer_config2;
// 设置图层参数，dispfd 为显示驱动句柄
unsigned long arg[3];
```

```
struct disp_layer_config2 config;
unsigned int width = 1280;
unsigned int height = 800;
unsigned int ret = 0;
memset(&config, 0, sizeof(struct disp_layer_config2));
config.channnel = 0; // blending channel
config.layer_id = 0; // layer index in the blending channel
config.info.enable = 1;
config.info.mode = LAYER_MODE_BUFFER;
config.info.fb.addr[0] = (unsigned long long)mem_in; // FB 地址
config.info.fb.size[0].width = width;
config.info.fb.align[0] = 4; // bytes
config.info.fb.format = DISP_FORMAT_ARGB_8888; // DISP_FORMAT_YUV420_P
config.info.fb.crop.x = 0;
config.info.fb.crop.y = 0;
config.info.fb.crop.width = ((unsigned long)width) << 32; // 定点小数。高32bit 为整数，低32bit 为小数
config.info.fb.crop.height = ((unsigned long)height) << 32; // 定点小数。高32bit 为整数，低32bit 为小数
config.info.fb.flags = DISP_BF_NORMAL;
config.info.fb.scan = DISP_SCAN_PROGRESSIVE;
config.info.fb.eotf = DISP_EOTF_SMPTE2084; // HDR
config.info.fb.metadata_buf = (unsigned long long)mem_in2;
config.info.alpha_mode = 2; // global pixel alpha
config.info.alpha_value = 0xff; // global alpha value
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 long)&config;
arg[2] = 1; // one layer
ret = ioctl(dispfd, DISP_LAYER_SET_CONFIG2, (void*)arg);
```

6.2.4 DISP_LAYER_GET_CONFIG2

- 作用：该函数用于获取图层参数
- 参数：handle：显示驱动句柄 cmd：DISP_LAYER_GET_CONFIG2 arg：arg[0] 为显示通道 0/1；arg[1] 为图层配置参数（disp_layer_config2）的指针；arg[2] 为需要获取配置的图层数目
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 设置图层参数，dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_layer_config2 config;
memset(&config, 0, sizeof(struct disp_layer_config2));
arg[0] = 0; // disp
arg[1] = (unsigned long)&config;
arg[2] = 1; // layer number
ret = ioctl(dispfd, DISP_GET_LAYER_CONFIG2, (void*)arg);
```


6.3 capture interface

6.3.1 DISP_CAPTURE_START

- 作用：该函数启动截屏功能
- 参数：handle：显示驱动句柄 cmd：DISP_CAPTURE_START arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 启动截屏功能，dispfd 为显示驱动句柄
arg[0] = 0; // 显示通道0
ioctl(dispfd, DISP_CAPTURE_START, (void*)arg);
```

6.3.2 DISP_CAPTURE_COMMIT

- 作用：该函数提交截屏信息，提交后才走在启动截屏功能
- 参数：handle：显示驱动句柄 cmd：DISP_CAPTURE_COMMIT arg：arg[0] 为显示通道 0/1；arg[1] 为 struct disp_capture_info 参数，用以设置截取的窗口信息和保存图片的信息。
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 提交截屏功能，dispfd 为显示驱动句柄
unsigned long arg[3];
struct disp_capture_info info;
arg[0] = 0;
screen_width = ioctl(dispfd, DISP_GET_SCN_WIDTH, (void*)arg);
screen_height = ioctl(dispfd, DISP_GET_SCN_HEIGHT, (void*)arg);
info.window.x = 0;
info.window.y = 0;
info.window.width = screen_width;
info.window.y = screen_height;
info.out_frame.format = DISP_FORMAT_ARGB_8888;
info.out_frame.size[0].width = screen_width;
info.out_frame.size[0].height = screen_height;
info.out_frame.crop.x = 0;
info.out_frame.crop.y = 0;
info.out_frame.crop.width = screen_width;
info.out_frame.crop.height = screen_height;
info.out_frame.addr[0] = fb_address; // buffer address
arg[0] = 0; // 显示通道0
arg[1] = (unsigned long)&info;
ioctl(dispfd, DISP_CAPTURE_COMMIT, (void*)arg);
```

6.3.3 DISP_CAPTURE_STOP

- 作用：该函数停止截屏功能
- 参数：handle：显示驱动句柄 cmd：DISP_CAPTURE_STOP arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 停止截屏功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
ioctl(dispfd, DISP_CAPTURE_STOP, (void*)arg);
```

6.3.4 DISP_CAPTURE_QUERY

- 作用：该函数查询刚结束的图像帧是否截屏成功
- 参数：handle：显示驱动句柄 cmd：DISP_CAPTURE_QUERY arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 查询截屏是否成功，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
ioctl(dispfd, DISP_CAPTURE_QUERY, (void*)arg);
```

6.4 lcd device Interface

6.4.1 DISP_LCD_SET_BRIGHTNESS

- 作用：该函数用于设置 LCD 的亮度
- 参数：handle：显示驱动句柄 cmd：DISP_LCD_SET_BRIGHTNESS arg：arg[0] 为 DE0/1 对应的显示通道；arg[1] 为背光亮度值（0~255）
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 设置LCD 的背光亮度，dispfd 为显示驱动句柄
unsigned long arg[3];
unsigned int bl = 197;
arg[0] = 0; // DE0输出的屏
arg[1] = bl;
ioctl(dispfd, DISP_LCD_SET_BRIGHTNESS, (void*)arg);
```

6.4.2 DISP_LCD_GET_BRIGHTNESS

- 作用：该函数用于获取 LCD 的亮度
- 参数：handle：显示驱动句柄 cmd：DISP_LCD_GET_BRIGHTNESS arg：arg[0] 为 DE0/1 对应的显示通道
- 返回：DIS_SUCCESS：成功其他：失败
- 示例

```
// 获取LCD的背光亮度，dispfd 为显示驱动句柄
unsigned long arg[3];
unsigned int bl;
arg[0] = 0; // 显示通道0
bl = ioctl(dispfd, DISP_LCD_GET_BRIGHTNESS, (void*)arg);
```

6.4.3 DISP_LCD_SET_GAMMA_TABLE

- 作用：该函数用于获取显示背景色。
- 参数：handle：显示驱动句柄 cmd：DISP_LCD_SET_GAMMA_TABLE arg：arg[0] 为显示通道 0/1；arg[1] 为 gamma table 的首地址；arg[2] 为 gamma table 的 size，字节为单位，建议为 1024，不能超过这个值
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 设置lcd的gamma table，dispfd 为显示驱动句柄
unsigned long arg[3];
unsigned int gamma_tbl[1024];
unsigned int size = 1024;
/* init gamma table */
/* gamma_tbl[nn] = xx; */
arg[0] = 0; // 显示通道0
arg[1] = gamma_tbl;
arg[2] = size;
if (ioctl(dispfd, DISP_LCD_SET_GAMMA_TABLE, (void*)arg))
    printf("set gamma table fail!\n");
else
    printf("set gamma table success\n");
```

6.4.4 DISP_LCD_GAMMA_CORRECTION_ENABLE

- 作用：该函数用于使能 lcd 的 gamma 校正功能
- 参数：handle：显示驱动句柄 cmd：DISP_LCD_GAMMA_CORRECTION_ENABLE arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号

- 示例

```
// 使能lcd 的gamma 校正功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
if (ioctl(dispfd, DISP_LCD_GAMMA_CORRECTION_ENABLE, (void*)arg))
    printf( "enable gamma correction fail!\n" );
else
    printf( "enable gamma correction success\n" );
```

6.4.5 DISP_LCD_GAMMA_CORRECTION_DISABLE

- 作用：该函数用于关闭 lcd 的 gamma 校正功能。
- 参数：handle：显示驱动句柄 cmd：DISP_LCD_GAMMA_CORRECTION_DISABLE arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 关闭lcd 的gamma 校正功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
if (ioctl(dispfd, DISP_LCD_GAMMA_CORRECTION_DISABLE, (void*)arg))
    printf( "disable gamma correction fail!\n" );
else
    printf( "disable gamma correction success\n" );
```

6.5 smart backlight

6.5.1 DISP_SMBL_ENABLE

- 作用：该函数用于使能智能背光功能
- 参数：handle：显示驱动句柄 cmd：DISP_SMBL_ENABLE arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS：成功其他：失败号
- 示例

```
// 开启智能背光功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
ioctl(dispfd, DISP_SMBL_ENABLE, (void*)arg);
```

6.5.2 DISP_SMBL_DISABLE

- 作用：该函数用于关闭智能背光功能
- 参数：handle：显示驱动句柄 cmd：DISP_SMBL_DISABLE arg：arg[0] 为显示通道 0/1
- 返回：DIS_SUCCESS: 成功其他: 失败号
- 示例

```
// 关闭智能背光功能，dispfd 为显示驱动句柄
unsigned long arg[3];
arg[0] = 0; // 显示通道0
ioctl(dispfd, DISP_SMBL_DISABLE, (void*)arg);
```

6.5.3 DISP_SMBL_SET_WINDOW

- 作用：该函数用于设置智能背光开启效果的窗口，智能背光在设置的窗口中有效
- 参数：handle：显示驱动句柄 cmd：DISP_SMBL_SET_WINDOW arg：arg[0] 为显示通道 0/1，arg[1] 为指向 struct disp_rect 的指针
- 返回：DIS_SUCCESS: 成功其他: 失败号
- 示例

```
// 设置智能背光窗口，dispfd 为显示驱动句柄
unsigned long arg[3];
unsigned int screen_width, screen_height;
struct disp_rect window;
screen_width = ioctl(dispfd, DISP_GET_SCN_WIDTH, (void*)arg);
screen_height = ioctl(dispfd, DISP_GET_SCN_HEIGHT, (void*)arg);
window.x = 0;
window.y = 0;
window.width = screen_width / 2;
window.height = screen_height;
arg[0] = 0; // 显示通道0
arg[1] = (unsigned long)&window;
ioctl(dispfd, DISP_SMBL_SET_WINDOW, (void*)arg);
```

7 调试方法

7.1 sysfs 节点介绍

```
diana-p1:/sys/class/disp/disp/attr # ls
boot_para    colorbar     enhance_bright enhance_detail enhance_saturation xres
capture_dump  cvbs_enhacne_mode enhance_contrast enhance_edge runtime_enable yres
color_temperature disp         enhance_denoise enhance_mode sys
```

disp2 显示驱动通过 sysfs 框架透出的调节点，各节点属性及功能如下：

表 7-1: display sysfs 节点属性

节点名字	说明	文件权限	取值范围
boot_para	查看 uboot -> kernel 传递的关键参数	可读/不可写	N/A
capture_dump	设置硬件截取输出后端数据	不可读/可写	N/A
color_temperature	查看/设置设备的色温	可读/可写	-50 ~ 150
colorbar	设置硬件输出内建 colorbar 数据	不可读/可写	0 ~ 8
cvbs_enhacne_mode	查看/设置 cvbs 设备的色彩增强模式	可读/可写	0 ~ 2
disp	查看/设置访问节点时的操作显示硬件索引	可读/可写	0/1
enhance_bright	查看/设置设备的亮度	可读/可写	0 ~ 100
enhance_contrast	查看/设置设备的对比度	可读/可写	0 ~ 100
enhance_denoise	查看/设置设备的去噪	可读/可写	0 ~ 10
enhance_detail	查看/设置设备的细节	可读/可写	0 ~ 10
enhance_edge	查看/设置设备的边缘	可读/可写	0 ~ 10
enhance_mode	查看/设置设备的色彩增强模式	可读/可写	0 ~ 3
enhance_saturation	查看/设置设备的饱和度	可读/可写	0 ~ 100
runtime_enable	查看/设置显示驱动的运行时休眠唤醒使能状态	可读/可写	0/1
sys	查看显示驱动的运行时当前时刻的状态	可读/不可写	N/A
xres	查看设备的 width	可读/不可写	N/A
yres	查看设备的 height	可读/不可写	N/A

7.1.1 查看显示驱动状态

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

```
diana-pi:/ # cat /sys/class/disp/disp/attr/sys
screen[0] -> dev[2]
de_rate 600000000 hz, ref_fps:60
mgr0: 1920x1080 fmt[yuv444] cs[0x101] range[limit] eotf[0x4] bits[8bits] err[0] force_sync[22306] unblank_direct_show[false] iommu[1] rcq_en[1]
rcq info: rcq_irq[89229] rcq_update_request[89228] rcq_update_req_irq[89516] rcq_finish_irq[89517]
dmabuf: cache[8] cache_max[12] umap_skip[0] umap_skip_max[0]
hdm1_output mode[10] fps:60.6 1920x1080
err:0 skip:0 irq:89517 vsync:89288 vsync_skip:0
BUF enable ch[0] lyr[0] z[0] prem[N] a[globl 255] fmt[ 72] fbd_type[none] fb[1920,1088; 960, 544; 960, 544] crop[ 0, 0,1920,1080] frame[ 0, 0
,1920,1080] addr[fea00000,fc7d800,febfe000] flags[0x 0] metadata_flag[0x 0] trd[0,0] depth[ 0] transf[0] sampling[1920:1920,1080:1080]
BUF enable ch[2] lyr[0] z[1] prem[Y] a[globl 255] fmt[ 1] fbd_type[afbd] fb[1920,1088; 0, 0; 0, 0] crop[ 0, 0,1920,1080] frame[ 0, 0
,1920,1080] addr[fc400000,fc5fe000,fc7fc000] flags[0x 0] metadata_flag[0x 10] trd[0,0] depth[ 0] transf[0] sampling[1920:1920,1080:1080]
disp[0] all:89224, sub:89224, cur:89224, free:89206, skip:0
```

图 7-1: disp2_sysfs 调试节点

整体可以分为四个部分介绍：

- (1) 红色部分表示 DE 硬件相关属性，以及一些核心机制的状态，包括 vsync，iommu，rcq 等；
- (2) 绿色部分表示输出设备的信息，当前输出为 HDMI，分辨率为 1080p@60 等；
- (3) 黄色部分表示显示驱动的 buf 相关信息，包括 dmabuf 缓存状态，以及 fence 相关信息；
- (4) 蓝色部分表示显示驱动图层信息，包括占用图层 index，图像格式，大小等；

调试说明：

可以重点关注这几个信息来判断系统是否正常工作：

1. 连续 cat sys 信息，对比 “err:xxx”、“vsync:xxx” 信息，如果 err 数有连续增加，大概率会有闪烁/花屏现象。vsync 计数是否一直递增，若 vsync 不递增，代表 vsync 中断刷新异常；
2. 查看 “ref_fps:xxx”、“fps:xxx” 是否与输出设备刷新率相同，如果 ref_fps 与输出设备刷新率不同时，需要检测 DTS 的 timing 配置是否合理；
3. 查看 “dmabuf:xxx” 的数值是否很大，当 cache 超过 50 左右时，系统可能出现内存不够，导致 dmabuf map 失败的问题；

各节点信息解析：

```
screen[0] -> dev[2]: 硬件底层连接关系，DE0 -> Tcon2;
de_rate: DE 的工作频率;
ref_fps: 参考 fps, 单位 hz;
mgr0: 逻辑显示实例id index;
1920x1080: 设备输出分辨率;
fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits]: 设备属性，依次表示：输出格式，颜色范围，eotf，输出 bit;
unblank: 设备是否 blank, unblank 开启显示，blank 关闭显示;
direct_show: cvbs 的直显模式;
iommu: iommu 是否使能;
rcq_en: rcq 更新 regs 是否使能

rcq info:
rcq_irq[2517]: rcq 完成中断次数;
rcq_update_request[2431]: rcq 更新请求次数;
rcq_update_req_irq[3933]: rcq 更新请求时的 vsync 计数;
rcq_finish_irq[3934]: rcq 完成中断时的 vsync 计数;

dmabuf:
cache[16]: 驱动内部缓存的 dma_buf 数量，若这个数超过50，则可能导致系统 dmabuf 内存被耗尽，从而无法 map;
cache_max[24]: 驱动内部缓存的 dma_buf 某个时刻出现的最大值，也就是 cache 的峰值;
```

umap skip[0]: 跳过 umap dma_buf 的次数，由于某些特殊情况，驱动或者无法判断是否有新帧被显示，送新帧时无法释放前面的显示帧情况发生，表示一次 umap skip；
umap skip max[0]: 跳过 umap dma_buf 的次数的最大值，也就是 umap skip 的峰值；

err: de 缺数的次数，de 缺数可能会出现屏幕抖动，花屏的问题。de 缺数一般为带宽不足引起；
skip: 表示 de 跳帧的次数，跳帧会出现卡顿问题。跳帧是指本次中断响应较慢，de 模块判断在本次中断已经接近或者超过了消隐区，将放弃本次更新图像的机会，选择继续显示原有的图像；
irq: 表示该通路上垂直消隐区中断执行的次数，一直增长表示该通道上的 timing controller 正在运行当中；
vsync: 表示显示模块往用户空间中发送的 vsync 消息的数目，一直增长表示正在不断地发送中；

BUF: 图层类型，BUF/COLOR，一般为 BUF，即图层是带 BUFFER 的。COLOR 意思是显示一个纯色的画面，不带 BUFFER。

enable: 显示处于 enable 状态

ch[0]: 该图层处于 blending 通道0

lyr[0]: 该图层处于当前 blending 通道中的图层0

z[0]: 图层z 序，越小越在底部，可能会被z 序大的图层覆盖住

prem[Y]: 是否预乘格式，Y 是，N 否

a: alpha 参数， globl/pixel/alpha 值

fmt: 图层格式，值64 以下为 RGB 格式；以上为YUV 格式，常见的72 为YV12,76 为NV12

fb: 图层buffer 的 size, width,height, 三个分量

crop: 图像buffer 中的裁减区域， [x,y,w,h]

frame: 图层在屏幕上的显示区域， [x,y,w,h]

addr: 三个分量的地址

flags: 一般为0， 3D SS 时为0x4, 3D TB 时为0x1, 3D FP 时为0x2;

trd: 是否 3D 输出， 3D 输出的类型（HDMI FP 输出时为1）各counter 描述如下：

sampling: 对输入图层采样过滤信息[in_w:out_w,in_h:out_h]

disp[0]all:2463, sub:2463, cur:2463, free:2436, skip:0

all: fence timeline 总数

submmit_count: 最后一次送显的 fence 序号

free_count: 已经释放的 fence 的序号

current_count: 当前正在显示的 fence 序号

skip_count: 跳帧总数，说明在 disp 中存在丢帧情况，原因为在一个 active 区内 hwcomposer 传递多于一帧的图像帧下来

7.1.2 使用 colorbar 测试

```
echo num > /sys/class/disp/disp/attr/colorbar
```

使用内建数据测试，num 可选参数及含义：

- 0: DE 直接输出（默认模式）
- 1 ~ 7: TCON 内建数据输出
- 8: DE 内建 colorbar 输出

7.1.3 回写输出数据

```
echo /data/filename.bmp > /sys/class/disp/disp/attr/capture_dump
```

截取 DE 输出图像，并保存到文件，支持 bmp 和 raw 格式，以文件名结尾区分：

- ”.bmp “截取并保存为 bmp 格式图片；

- “.yuv420_p “、”.yuv420_sp_uvuv “、”.yuv420_sp_vuvu “、”.argb8888 “、”.abgr8888”、”.bgr888”、“.rgba8888”、“.bgra8888” 截图并保存为对应 raw 格式；

注意：Linux5.15 及以上 aops 内核不在支持，可以编写应用端的 capture 程序实现，android 参考实现：HWComposer: hwcdebug -capture

7.1.4 调节画质 enhance

```
echo num > /sys/class/disp/disp/attr/enhance_xxx
```

enhance_mode、enhance_bright、enhance_contrast、enhance_saturation 为一组，用来调整画面整体的基础色彩参数

enhance_mode 可以取 0~3，分别表示：0 - 标准、1 - 生动、2 - 用户可配置、3 - demo 模式。

enhance_bright、enhance_contrast、enhance_saturation，用来调节画面整体的亮度、对比度、色度，可调范围为 0~100。

enhance_denoise、enhance_detail、enhance_edge 为一组，用来调节视频画面的细节。可调范围为 0~10。

注意：enhance_denoise、enhance_detail、enhance_edge，部分平台 vep 可能不支持

7.1.5 色温

```
echo num > /sys/class/disp/disp/attr/color_temperature
```

调节画面整体色温参数，可取 -50~150，负数为冷色调，正数为暖色调。

注意：AW1890 色温可调范围为 [-50, 150]，其它型号可调范围为 [-255, 255]

7.2 debugfs 接口

显示驱动使用 debugfs 框架，实现一些常用的功能接口组合，通过文件节点的形式透给用户使用。

运行前需要先挂载：

```
mount -t debugfs none /sys/kernel/debug  
cd /sys/kernel/debug/dispdbg
```

目录结构：

```
diana-p1:/sys/kernel/debug/dispdbg # ls
command dbgvl info name param start
```

- name：表示操作的对象名字
- command：表示执行的命令
- param：表示该命令接收的参数
- start：输入 1 开始执行命令
- info：保存命令执行的结果

7.2.1 配置输出设备参数

```
echo disp0 > name
echo switch1 > command
echo 4 10 1 1 0x4 0x104 0 0 8 > param
echo 1 > start
```

其中，param 参数含义为：

- Arg1：显示接口代号，1：LCD 显示接口；2：tv 显示接口；4：HDMI 显示接口；8：VGA 显示接口；32：edp 显示接口
- Arg2：tv mode，显示分辨率代号，可以在 include/video/sunxi_display2.h 中的 enum disp_tv_mode 中查到
- Arg3：color format，0：RGB；1：YUV444；2：YUV422；3：YUV420
- Arg4：color depth，0：8Bit；1：10Bit；3：12Bit；4：16Bit
- Arg5：eotf
- Arg6：color space；0x104：BT601；0x101：BT709；0x107：BT2020
- Arg7：mode，1：DVI Mode；0：HDMI Mode

通过 disp0/1，控制切换主显/副显连接的输出设备参数。

7.2.2 开关屏幕

开启 LCD 屏幕设备：

```
echo lcd0 > name
echo enable > command
echo 1 > start
```

关闭 LCD 屏幕设备：

```
echo lcd0 > name
echo disable > command
echo 1 > start
```

lcd (n) ,n 与 dts 配置屏参的 lcd 节点索引相同。

7.2.3 开关显示

开启显示：

```
echo blank > command
echo 1 > param
echo disp0 > name
echo 1 > start
```

关闭显示：

```
echo blank > command
echo 0 > param
echo disp0 > name
echo 1 > start
```

7.2.4 显示的休眠和唤醒

显示模块可以独立于其它模块进行休眠唤醒测试。

休眠：

```
echo suspend > command
echo disp0 > name
echo 1 > start
```

唤醒：

```
echo resume > command
echo disp0 > name
echo 1 > start
```

8 FAQ

8.1 黑屏（无背光）

问题现象：机器接 LCD 输出，发现 LCD 没有任何显示，仔细查看背光也不亮。

问题分析：此现象说明 LCD 背光供电不正常，不排除还有其他问题，但没背光的问题必须先解决。

问题排查步骤：

- 步骤一

使用电压表量 LCD 屏的各路电压，如果背光灯脚电压不正常，请对照原理图确定背光电压对应 GPIO、电源或者 PWM 有没使能。否则，尝试换个屏再试。

- 步骤二

如果怀疑是 GPIO、电源没有使能，那需要对照原理图，在 board.dts 配置上

- 步骤三

如果怀疑是 PWM 没有使能，先看看随 sdk 有没发布 PWM 模块使用指南，如果有按照里面步骤进行排查。如果 sdk 没有发布 PWM 模块使用指南。可以 `cat /sys/kernel/debug/pwm` 看看有没输出。如果没有就是 PWM 驱动没有加载，请检查一下 menuconfig 有没打开。

- 步骤四

如果步骤三未解决问题，请排查 dts 或 board.dts 配置。如果还没有解决，可以寻求技术支持。

8.2 黑屏（有背光）

问题现象：机器接 LCD，发现有背光，界面输出黑屏。

问题分析：此现象说明没有内容输出，可能是 DE、TCON 出错或应用没有送帧。

问题排查步骤：

- 步骤一

根据“查看显示模块的状态”章节排查应用输入的图层信息是否正确。其中，宽高、显存的文件句柄出错问题最多。

- 步骤二

根据“截屏”章节截屏，看看 DE 输出是否正常。如果不正常，排查 DE 驱动配置是否正确；如果正常，接着下面步骤。

- 步骤三

根据colorbar 输出 colorbar，如果 TCON 自身的 colorbar 也没有显示，排查硬件通路；如果有显示，排查 TCON 输入源选择的寄存器。后者概率很低，此时可寻求技术支持。

8.3 绿屏

问题现象：显示器出现绿屏，切换界面可能有其他变化。

问题分析：此现象说明处理图层时 DE 出错。可能是应用送显的 buffer 内容或者格式有问题；也可能 DE 配置出错。

问题排查步骤：

- 步骤一

根据查看显示模块的状态排查应用输入的图层信息是否正确。其中，图层格式填错的问题最多。

- 步骤二

导出 DE 寄存器，排查异常。此步骤比较复杂，需要寻求技术支持。

8.4 缺数报错

问题现象：连续查看显示模块的状态“err:”一直增加，同时屏幕出现闪屏/花屏现象。

问题分析：缺数是硬件提供了一种自检机制，触发原因是 TCON 往 DE 取数据时发现“buf”是空的，无数据可取，显示过程中，DE 会向 DDR 取数据来填充这个“buf”，主要有两种原因会触发 DE 未及时填充这个“buf”。

问题排查步骤：

- 步骤一

DE 性能瓶颈导致缺数，出现原因是，系统带宽是充足的，但是在单位时间内 DE 处理不过来这些数据，填充 “buf” 不及时，导致缺数。

1. 时序性能指标：DE 的性能指标为 1pix/cyc，基本判断标准是 $de_clk \geq pclk$ ，[查看显示模块的状态](#) 的 “de_rate:” 可以查看 de 的工作时钟，pclk 为设备输出的像素时钟，这两者必须满足这个公式 $de_clk \geq pclk$ 。
2. scaler 性能指标：当 crop 大小和显示大小不一致时，会使用 scaler 进行缩放，目前 DE 最大支持 4k video format 输入缩放，只有 channel0 支持，若某个通道开启了缩放，输入大小比较大（crop 的大小），DE 处理不过来，导致缺数

- 步骤二

大多数缺数都是由于 DDR 带宽导致，原因是 DE 往 DDR 取数时，由于带宽限制，要不到数据，导致缺数。

这种情况需要分析问题场景 DE 的带宽，并对比公版 DDR 参数与性能，来定位问题，可以通过下面方法调试：

1. [查看显示模块的状态](#)，计算问题场景图层需要的带宽
2. 调整 DDR 频率
3. 关闭 DVFS 功能

具体解决方法，可以寻求技术支持，需要根据 DDR 物料性能，出现场景等来分析提供解决方法。

8.5 界面卡住

问题现象：界面定在一个画面，不再改变。

问题分析：此现象说明显示通路一般是正常的，只是应用没有继续送帧。

问题排查步骤：

- 步骤一

根据[查看显示模块的状态](#)排查应用输入的图层信息有没改变，特别关注图层的地址。

- 步骤二

排查应用送帧逻辑，特别关注死锁，线程、进行异常退出，fence 处理异常。

8.6 局部界面花屏

问题现象：画面切到特定场景时候，出现局部花屏，并不断抖动。

问题分析：此现象是典型的 DE scaler 出错现象。

问题排查步骤：

根据查看显示模块的状态查看问题出现时带缩放图层的参数。如果屏幕输出的 width x height 除以 crop 的 width x height 小于 1/16 或者大于 32，那么该图层不能走 DE 缩放，改用 GPU，或应用修改图层宽高。

8.7 快速切换界面花屏

问题现象：快速切换界面花屏，变化不大的界面显示正常。

问题分析：此现象是典型的性能问题，与显示驱动关系不大。

问题排查步骤：

- 步骤一

排查 DRAM 带宽是否满足场景需求。

- 步骤二

若是安卓系统，排查 fence 处理流程；若是纯 linux 系统，排查送帧流程、swap buffer、pandisplay 流程。

8.8 显示偏色

问题现象：图像显示颜色异常偏色，其他显示现象正常。问题分析：此现象是显示设备偏色问题，可能与颜色空间的配置不对相关。

问题排查步骤：

- 步骤一

查看显示节点颜色空间值是否异常

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

```
# cat /sys/class/disp/disp/attr/sys
screen 0:
de_rate 696000000 hz, ref_fps:61
mgr0: 1280x800 fmt[rgb] cs[0x204] range[full] eof[0x4] bits[8bits] err[0] force_sync[0] unblank direct_show[false]
dmabuf: cache[0] cache_max[0] umap skip[0] overflow[0]
    lcd output backlight( 50) fps:61.4 1280x 800
    err:0 skip:18 irq:127547 vsync:0 vsync_skip:0
    BUF enable ch[1] lyr[0] z[0] prem[N] a[global 255] fmt[ 0] fb[1280, 800;1280, 800;1280, 800] crop[ 0, 0,1280, 800] frame[
    0, 0,1280, 800] addr[ff800000, 0, 0] flags[0x 0] trd[0,0]
depth[ 0] transf[0]
```

图 8-1: 显示调试节点

● 步骤二

一般的颜色空间配置原则是

```
DISP_BT601 = 0x104 (<720P)
DISP_BT709 = 0x101 (>=720p)
DISP_BT2020NC_F = 0x207 (4k)
```

如果颜色空间设置不对，排查应用层对颜色空间的改动。

8.9 调节亮度对比度饱和度

问题描述：T5 平台如何调节色温, 亮度, 对比度, 饱和度的值

问题解决：可通过 /sys/class/disp/disp/attr/ 内节点调节相关参数，参考 [enhance](#) 小节。

8.10 如何实现双显 logo 启动

问题描述：目前已支持双显的驱动版本，如何实现双显 logo 启动？

问题解决：

以 LVDS 1280x800 + HDMI 1920x1080 为例：

● 步骤一、修改 uboot-board.dts

```
&disp {
    ...
    dev_num          = <2>; # 关键配置：指示有 2 个启动显示设备，不定义则默认 1 个
    dev0_output_type  = <1>;
    dev0_output_mode  = <4>;
    dev0_screen_id    = <0>;
    dev0_do_hpd       = <0>;

    dev1_output_type  = <4>; # 配置副屏设备的信息
    dev1_output_mode  = <10>;
```



```

dev1_screen_id    = <1>;
dev1_do_hpd       = <1>;

...

fb0_format        = <0>;
fb0_width         = <1280>;
fb0_height        = <800>;
fb0_rot_used      = <0>;
fb0_rot_degree    = <0>;

fb1_format        = <0>; # 配置副屏 fb 的信息
fb1_width         = <1920>;
fb1_height        = <1080>;
fb1_rot_used      = <0>;
fb1_rot_degree    = <0>;

...
};

```

- 步骤二、修改 board.dts

```

&disp {
    ...
    fb_format      = <0>;
    fb_num         = <2>;    # 关键配置：定义 2 个 framebuffer 承接 uboot logo
    fb_debug       = <1>;
    /*<disp channel layer zorder>*/
    fb0_map        = <0 1 0 16>; # 主显 framebuffer 配置
    fb0_width      = <1280>;    # 宽高与 uboot 阶段保持一致
    fb0_height     = <800>;
    /*<disp channel layer zorder>*/
    fb1_map        = <1 1 0 16>; # 副显 framebuffer 配置
    fb1_width      = <1920>;    # 宽高与 uboot 阶段保持一致
    fb1_height     = <1080>;
    ...
};

```

- 步骤三、接线上电启动

例如：LCD + HDMI：连接好 LCD 排线和 HDMI 线缆，上电启动，即可看到 2 个屏幕都显示出 logo；

- 步骤四、单自定义 logo 图片

目前驱动的逻辑仅验证双显 logo 图片的可行性，所以 2 个屏幕都显示同一张 logo 图片（即：bootlogo.bmp）。如果想每个屏幕独立显示不同的启动 logo，只需改动以下驱动代码。

⚠ 注意

图片大小不能大于屏幕分辨率！

路径：<sdk 路径>/brandy/brandy-2.0/u-boot-2018/drivers/video/sunxi/logo_display/cmd_sunxi_bmp.c

```
int sunxi_bmp_display(char *name)
{
    ...
    /* 默认将 boot_logo.bmp 显示到每个屏幕的 framebuffer 中，每个屏幕 1 个 framebuffer。
       可自定义为：按不同名字显示到不同的 framebuffer 中。*/
    for (i = 0; i < get_framebuffer_num(); i++) {
        ret = show_bmp_on_fb(bmp_head_addr, i);
        if (ret != 0)
            pr_error("show bmp on fb failed !%d\n", ret);
    }
    ...
}
```

8.11 如何替换开机 logo

问题描述：如何替换开机 logo，实现开机自定义开机 logo 图片

问题解决：

● 步骤一

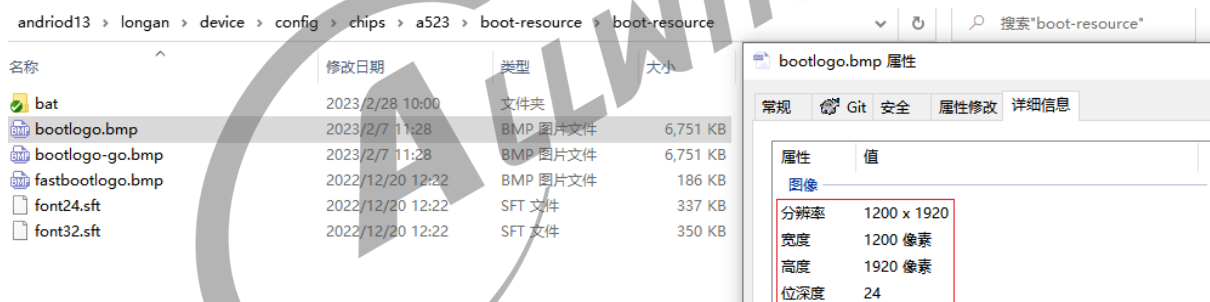


图 8-2: bmp 属性

修改分区 bootlogo.bmp 图片，并获取修改后的分辨率和位深度信息。路径：/device/config/chips/{IC}/boot-resource/boot-resource/

● 步骤二

参考uboot，配置 uboot fb 信息：

一般使用 ARGB8888/RGB888 bmp 图片，对应 bpp 为 32/24，配置 fb0_format 为 0/8

根据分辨率配置 fb0_width 和 fb0 _ width 信息

● 步骤三

参考kernel，与 uboot fb 同步配置 kernel fb 信息：

一般使用 ARGB8888/RGB888 bmp 图片，对应 bpp 为 32/24，配置 fb0_format 为 0/8

根据分辨率配置 fb0_width 和 fb0 _ width 信息






著作权声明

版权所有 © 2024 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 全志科技、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的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所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。