Rockchip DRM Display Driver 开发指南

文件标识：RK-YH-YF-455

发布版本：V3.5.1

日期：2022-07-06

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

本文主要介绍 Rockchip 平台处理器基于 DRM 显示框架 VOP 以及相关显示接口的基本特性、工作流程和常见问题分析。目的是为了相关工程师能对 DRM 显示驱动框架和硬件接口有更好的理解，并通过常见问题的分析能快速定位问题、解决问题。

产品版本

芯片名称	内核版本
RK3036	Linux kernel 4.4 及以上内核
RK312X/PX3SE	Linux kernel 4.4 及以上内核
RK3288	Linux kernel 4.4 及以上内核
RK322X/RK312XH	Linux kernel 4.4 及以上内核
RK3308	Linux kernel 4.4 及以上内核
RK322XH/RK332X	Linux kernel 4.4 及以上内核
RK3326/PX30	Linux kernel 4.4 及以上内核
RK3368/PX5	Linux kernel 4.4 及以上内核
RK3399	Linux kernel 4.4 及以上内核
RK1808	Linux kernel 4.4 及以上内核
RV1109/RV1126	Linux kernel 4.19 及以上内核
RK356X	Linux kernel 4.19 及以上内核
RK3588	Linux kernel 5.10 及以上内核
RV1103/RV1106	Linux kernel 5.10 及以上内核

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

硬件开发工程师

修订记录

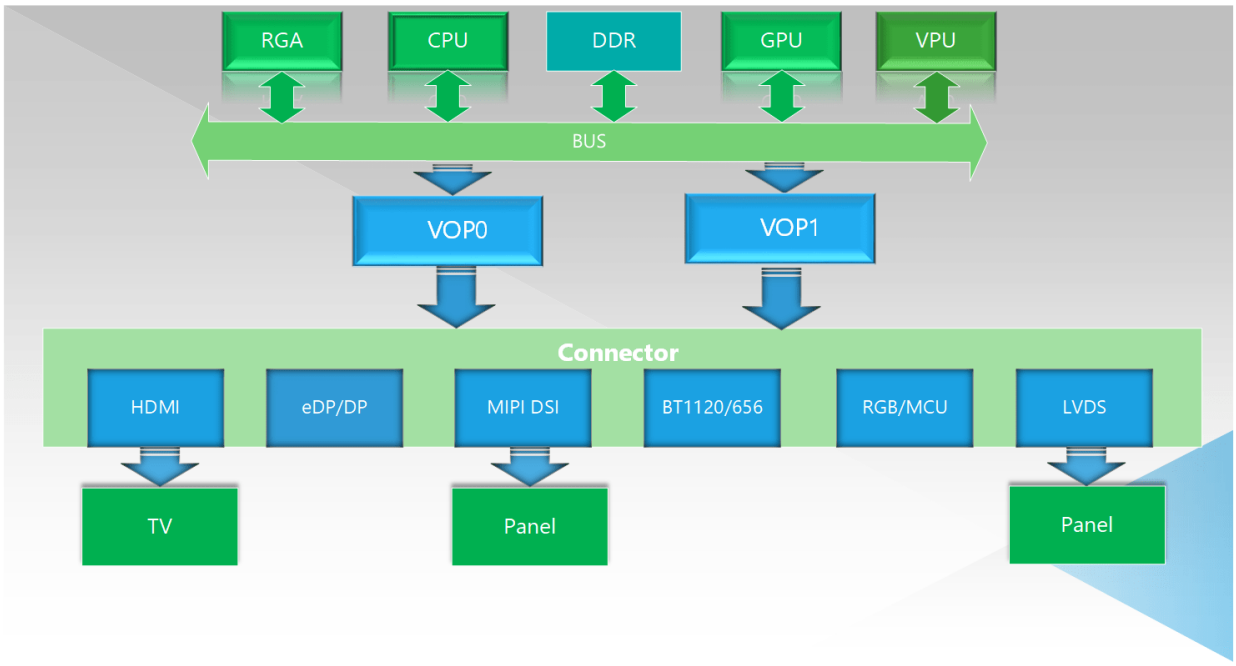
版本号	作者	修改日期	修改说明	
V1.0.0	黄家钗	2019-1-5	初始版本	
V2.0.0	黄家钗	2019-11-15	针对 Linux 4.19 更新	
V3.0.0	黄家钗	2022-03-17	针对 RK356X/RK3588/RV1103/RV1106 以及 Linux kernel 5.10 更新	
V3.1.0	闫孝军	2022-04-24	加入更多内容	
V3.1.2	闫孝军	2022-05-09	修正一些描述错误	
V3.1.3	闫孝军	2022-05-30	补充相关引用参考文档	
V3.2.0	黄家钗	2022-06-12	更新 VOP 和接口 feature	
V3.3.0	张玉炳	2022-06-23	添加 RK3588 VOP DCLK 分配策略	
V3.4.0	黄家钗	2022-06-29	加入 RK3566 图层分配策略说明	
V3.4.1	闫孝军	2022-07-06	修正 connector-split 相关描述	
V3.5.0	闫孝军	2022-07-08	加入一些定位问题的方法和建议	
V3.5.1	张玉炳	2022-07-13	修正 dclk 相关描述	

目录

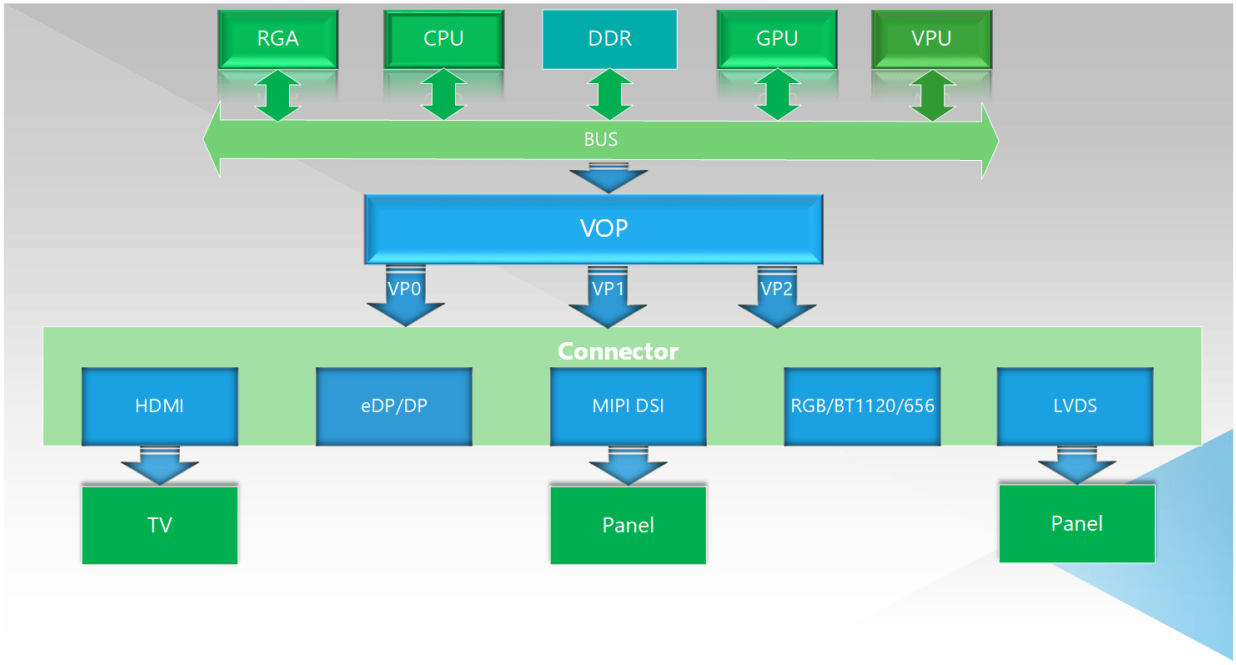
Rockchip 平台显示子系统（DSS） 概述

显示子系统是 Rockchip 平台显示输出相关软硬件系统的统称，它包括 VOP（比较老的平台叫 LCDC，比如 RK3188、RK3066）和 RGB、BT1120、BT656、I8080（MCU 显示接口），LVDS、MIPI DSI、EDP、DP、HDMI 等显示信号输出模块以及与之对应的软件驱动。

整个显示系统的硬件框架如下图所示：



VOP 1.0 显示子系统架构



VOP 2.0 显示子系统架构

从上面的 DSS 框图可以看到，在整个显示通路的最后端，是由 RGA，GPU、VPU 组成的显示图形加速模块，他们是专门针对图像处理优化设计的硬件 IP，能够高效的进行图像的生成和进一步处理（比如 GPU 通过 opengl 功能提供图像渲染功能，RGA 可以对图像数据进行缩放，旋转，合成等 2D 处理，VPU 可以高效的进行视频解码），从而减轻 CPU 负担。

经过这些图像加速模块处理后的数据会存放在 DDR 中，然后由 VOP 读取，根据应用需求进行 Alpha 叠加，颜色空间转换，gamma 矫正，HDR 转换 等处理后，再发送到对应的显示接口模块（HDMI/DP/DSI/RGB/LVDS），这些接口模块会把接收到的数据转换成符合各自协议的数据流，发送到显示器或者屏幕上，呈现在最终用户眼前。

目前 Rockchip 平台上存在两种 VOP 架构—— VOP 1.0 和 VOP 2.0，他们的主要的区别是对多显的支持方式不同，VOP 1.0 是用多 VOP 的方式来实现多屏幕显示，即正常情况下，一个 VOP 在同一时刻只能输出一路独立的显示时序，驱动一个屏幕显示独立的内容。如果需要实现双屏显示，则需要有两个 VOP 来实现，所以在 RK3288，RK3399，PX30 等支持双显的平台上，都有两个独立的 VOP。

VOP 2.0 采用了统一显示架构，即整个 SOC 上只存在一个 VOP，但是在 VOP 的后端设计了多路独立的 Video Port(简称 VP) 输出接口，这些 VP 能够同时独立工作，并且输出相互独立的显示时序。比如在上面的 VOP 2.0 框图中，有三个 VP，就能同时实现三屏异显。

如果想了解哪些芯片采用的是 VOP 1.0 架构，哪些芯片采用的是 VOP 2.0 架构，请参考后面的 **Display Feature** 章节内容。

DRM 概述

DRM 全称是 Direct Rendering Manager，进行显示输出管理、buffer 分配、帧缓冲。对应的 userspace 库为 libdrm，libdrm 库提供了一系列友好的控制封装，使用户可以方便的进行显示的控制和 buffer 申请。DRM 的设备节点为 "/dev/dri/cardX"， X 为 0-15 的数值，默认使用的是 /dev/dri/card0。

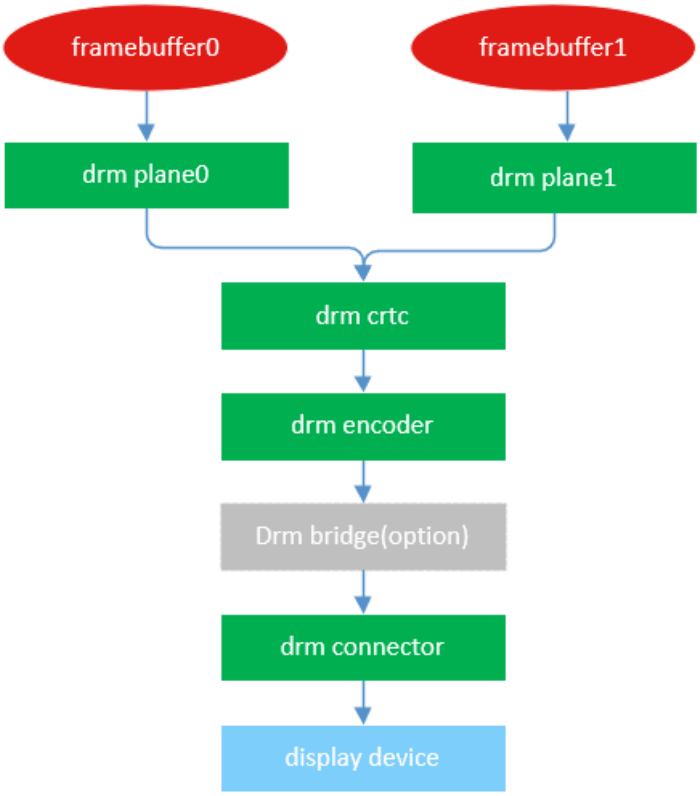
Rockchip 平台从 Linux 4.4 内核开始，显示驱动全部切到 DRM 显示框架。

基本概念

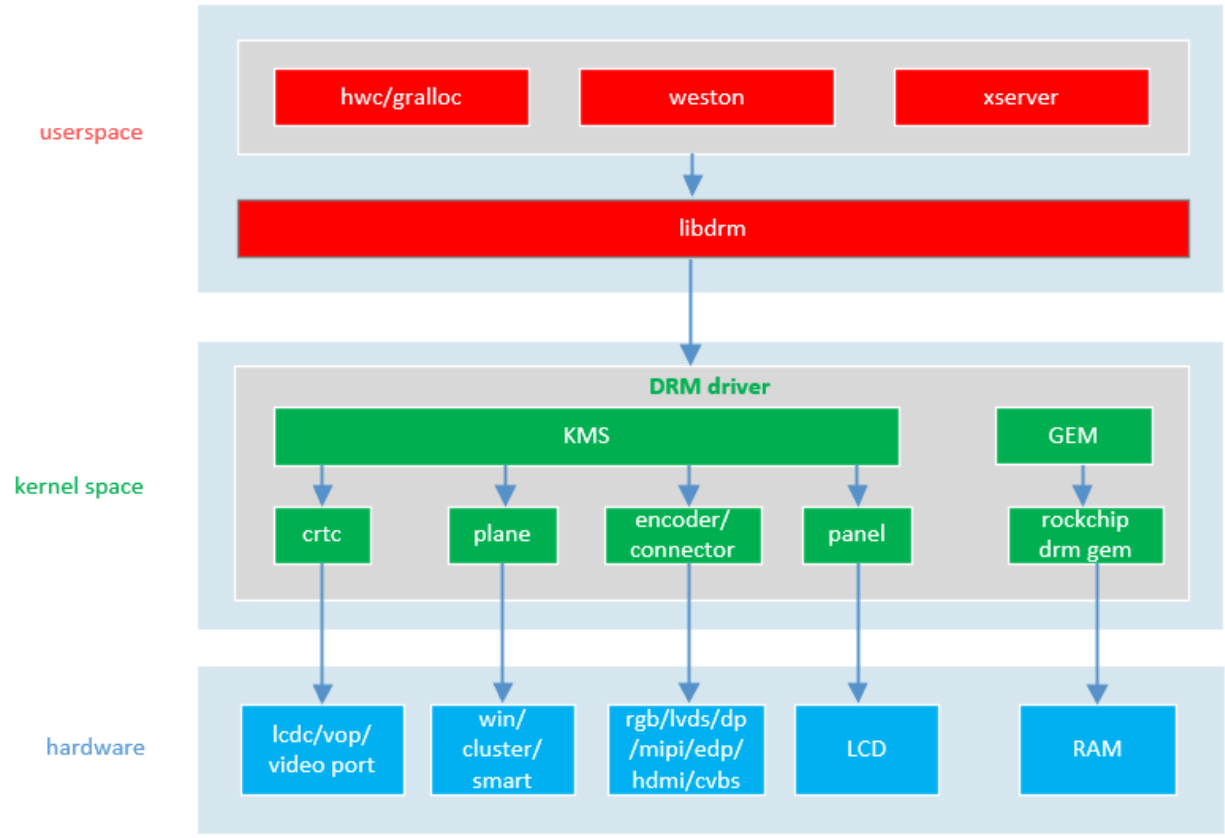
为了方便管理显示通路上的各种硬件模块，DRM 定义了以下几个概念：

基本概念	说明
CRTC	显示控制器，在 rockchip 平台是 SOC 内部 VOP（部分文档也称为 LCDC）模块或者 VOP2 中 Video Port 的抽象
Plane	图层，在 rockchip 平台是 SOC 内部 VOP（LCDC）模块 win 图层的抽象
Encoder	输出转换器，指 RGB、LVDS、DSI、eDP、DP、HDMI、CVBS、VGA 等显示接口
Connector	连接器，指 encoder 和 panel 之间交互的接口部分
Bridge	桥接设备，一般用于注册 encoder 后面另外再接的转换芯片，如 DSI2HDMI 转换芯片
Panel	泛指屏，各种 LCD 显示设备的抽象
GEM	DRM 下 buffer 管理和分配，类似 ION、DMA BUFFER

显示通路



DRM 驱动和 libdrm 的交互过程



软件驱动

U-Boot 驱动

显示驱动在 U-Boot 中主要提供开机 logo 显示和充电界面显示这两个功能。

在 Rockchip 平台上，开机 logo 一般分为两个阶段：显示 U-Boot logo 和 显示 Kernel logo。

默认 U-Boot logo

默认 Kernel logo

这两个 LOGO 图片默认放在 Linux kernel 根目录下（logo.bmp 和 logo_kernel.bmp），Linux Kernel 在编译的时候会把他们打包到 resource.img 中，再打包进入 Boot.img。

U-Boot 启动的时候会把这两个文件加载到内存中，U-Boot LOGO 在 U-Boot 阶段就开始显示，Kernel LOGO 在内存中的地址会被 U-Boot 传递给 Linux kernel，在 Linux Kernel 的 drm 驱动初始化阶段显示。

驱动目录

```
drivers/video/drm/
```

驱动文件

Driver	File
Core	rockchip_display.c rockchip_crtc.c rockchip_connector.c rockchip_phy.c rockchip_panel.c rockchip_bridge.c
VOP	rockchip_vop.c rockchip_vop_reg.c rockchip_vop2.c rockchip_vop2_reg.c
RGB	rockchip_rgb.c inno_video_combo_phy.c
LVDS	rockchip_lvds.c inno_video_combo_phy.c
MIPI-DSI	drm_mipi_dsi.c dw_mipi_dsi2.c inno_mipi_phy.c inno_video_combo_phy.c samsung_mipi_dcphy.c
eDP	rockchip_analogix_dp.c rockchip_analogix_dp_reg.c
HDMI	dw_hdmi.c rockchip_dw_hdmi.c rockchip-inno-hdmi-phy.c inno_hdmi.c dw_hdmi_qp.c rockchip_dw_hdmi_qp.c phy-rockchip-samsung-hdptx-hdmi.c
TVE /CVBS	rockchip_drm_tve.c
DP	dw-dp.c drm_dp_helper.c phy-rockchip-usbdp.c

以上驱动文件在不同的 U-Boot 版本可能会做些小的调整，但多数驱动文件和框架部分基本不会变化，查阅的时候可以根据实际情况调整。

接口说明

1. 显示 U-Boot logo

```
void rockchip_show_logo(void)
```

2. 显示指定的 bmp 图片，目前主要用于充电 logo 的显示

```
void rockchip_show_bmp(const char *bmp)
```

3. 将 U-Boot 中确定的一些变量通过修改 dtb 文件传递给内核，包括 kernel logo 的大小、地址、格式、输出扫描时序以及过扫描的配置等信息

```
void rockchip_display_fixup(void *blob)
```

应用说明

1. 开启 U-Boot logo

   logo 通过 Linux kernel dts(U-Boot 显示模块和 Linux kernel 复用同一个 dtb) 中对应显示接口的 route_xxx 节点控制，这里的 xxx 可以是 dsi，edp，hdmi，lvds，具体请搜索 dts 中的 route 关键字。

   以 MIPI DSI0 为例，在对应的板级 dts 文件里找到 route_dsi0 节点，把 status 设置为 “okay”:

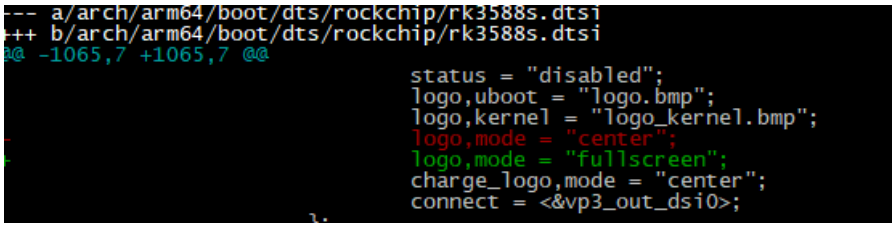
```
&route_dsi0 {
        status = "okay";
        connect = <&vp3_out_dsi0>;
};
```

`connect` 属性参考后面 **DTS 配置** 章节的详细说明。

2. 配置 U-Boot logo 全屏显示

   logo 默认是居中（center）显示，如果需要全屏显示，修改对应接口的 route 节点。

   比如需要把 DSI0 上输出的 logo 全屏显示，则修改 route_dis0 的 logo,mode 属性为 “fullscreen”。



3. logo 图片要求

   U-Boot logo 和 Linux Kernel logo 分辨率相同，而且分辨率必须是偶数；

   只支持 8bit，16bit，24bit、32bit 的 bmp 图片；

   U-Boot logo 和 Linux Kernel logo 必须同时开启，即 logo.bmp 和 logo_kernel.bmp 必须同时提供，不能只提供其中一个。

4. 启动 log 确认

   如果 logo 功能正常启用，在 U-Boot 阶段会看到类似如下的 log：

```
Rockchip UBOOT DRM driver version: v1.0.1
vp0 have layer nr:2[0 2 ], primary plane: 2
vp1 have layer nr:2[1 3 ], primary plane: 3
vp2 have layer nr:2[6 8 ], primary plane: 8
vp3 have layer nr:2[7 9 ], primary plane: 9
Using display timing dts
dsi@fde20000:  detailed mode clock 132000 kHz, flags[a]
    H: 1080 1095 1099 1129
    V: 1920 1935 1937 1952
bus_format: 100e
VOP update mode to: 1080x1920p0, type: MIPI0 for VP3
VOP VP3 enable Esmart3[654x270->654x270@213x825] fmt[2] addr[0xedf04000]
final DSI-Link bandwidth: 880000 Kbps x 4
......
hdmi_select_link_config use tmds mode
mode:1920x1080 bus_format:0x100a
hdmi@fde80000:  detailed mode clock 148500 kHz, flags[5]
    H: 1920 2008 2052 2200
    V: 1080 1084 1089 1125
bus_format: 100a
VOP update mode to: 1920x1080p0, type: HDMI0 for VP0
......
VOP VP0 enable Esmart0[654x270->654x270@633x405] fmt[2] addr[0xedf04000]
......
CLK: (uboot. arm: enter 1200000 KHz, init 1200000 KHz, kernel 0N/A)
```

   可以看到在 VP0 和 VP3 上都启动了 logo 显示。

   VP0 的显示分辨率为 1920 x 1080，显示接口是 HDMI。

   VP3 的显示分辨率为 1080 x 1920，显示接口是 MIPI DSI。

   显示的 logo 大小为 654 x 270。

分析 U-Boot logo 过渡到 Linux Kernel logo 过程中出现的闪屏或者无法显示的问题

1. 确认 DRM 驱动是否有正常加载

   DRM 驱动加载过程中可能会出现一些资源没有准备好，导致 DRM 驱动 bind 失败，可能会出现类似以下 log：

```
[1.792387] dw-mipi-dsi2 fde20000.dsi: [drm:dw_mipi_dsi2_bind] *ERROR* Failed to find panel or bridge: -517
```

这个 log 说明此时 panel 或者 bridge 没准备好，正常驱动框架会在一段时间后重新开始 bind，但如果最后出现以下 log 说明此时 drm 驱动已经加载成功了，这种情况我们就不需要太在意前面一两次 bind 失败的 log：

```
[2.566831] rockchip-drm display-subsystem: [drm] fb0: rockchipdrmfb frame buffer device
```

如果开机 log 一直不停的刷 bind 失败的 log，那可能要检查的你 dts 配置，产品中最经常遇到的是 GPIO 口被其他设备先注册、panel 的 compatible 未正确配置导致 panel 注册失败、backlight 驱动注册失败等。

如果 Linux kernel log 中出现了如下 logo 失败相关的信息，则需要结合代码认真分析：

```
rockchip-drm display-subsystem: failed to parse resources for logo display
rockchip-drm display-subsystem: connector[HDMI-A-1] can't found any modes
rockchip-drm display-subsystem: can't not find any logo display
rockchip-drm display-subsystem: failed to show loader logo
```

2. DDR 变频导致闪屏或者显示错位

(1) 尝试关闭 dts 文件中 DDR 变频节点，保证内核阶段不做 DDR 变频，修改方法如下：

```
&dmc {
    status = “disabled”;
};
```

(2) 尝试关闭 DDR 变频时对 DCLK 频率的调整，修改方法如下：

```
&dmc {
    vop-dclk-mode = <1>;
};
```

3. clk tree 变化导致

部分平台 U-Boot 中的 clk tree 配置和内核的 clk tree 配置是独立的，如果两个驱动的 clk 策略不一致，就有可能在 Linux kernel clk 重新初始化的时候出现闪屏问题，以 RK3399 为例，可以按如下方法确认：

(1) 在 rk3399_clk_init()@kernel/drivers/clk/rockchip/clk-rk3399.c 函数入口处加上 while(1)；确认是否会有闪屏问题；

(2) 在 rk3399_clk_init()@kernel/drivers/clk/rockchip/clk-rk3399.c 函数结束处加上 while(1)，确认是否会有闪屏问题；

(3) 如果步骤 (1) 中无闪屏现象步骤 (2) 中有闪屏现象，那基本可以确认是 clk tree 变化导致闪屏问题，可以直接找对应平台 cru 负责人或者提交 redmine 并说明转给 pll 相关负责人。

4. 时钟被关闭导致

有一些流程上的问题或者软件上的 bug 可能存在一些必要的时钟在驱动注册的时候没有被使能，导致这些时钟在内核跑完后被框架自动关闭从而导致显示异常，可以尝试按下面的修改默认不关闭时钟做测试：

在 dts 文件中，找到 chosen 节点，在 bootargs 末尾加上 clk_ignore_unused：如 bootargs = "xxxx clk_ignore_unused”;

```
--- a/arch/arm64/boot/dts/rockchip/rk3399-linux.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3399-linux.dtsi
@@ -47,7 +47,7 @@
        compatible = "rockchip,linux", "rockchip,rk3399";

        chosen {
-               bootargs = "earlycon=uart8250,mmio32,0xff1a0000";
+               bootargs = "earlycon=uart8250,mmio32,0xff1a0000 clk_ignore_unused";
        };
```

5. U-Boot logo 图片和 kernel logo 图片大小不一致

rockchip 平台有要求 U-Boot logo 和 kernel logo 的图片分辨率大小一样，如果出现 U-Boot logo 显示正常，到内核阶段显示异常，可以确认下 kernel 目录下 logo.bmp 和 logo_kernel.bmp 分辨率是否一致；

```
$file logo.bmp  logo_kernel.bmp
logo.bmp:        PC bitmap, Windows 3.x format, 654 x 258 x 8
logo_kernel.bmp: PC bitmap, Windows 3.x format, 654 x 258 x 8
```

6. 内核初始化过程一些电源 GPIO 被重新初始化

该问题涉及的可能性很多，总之在新项目 porting 过程中要及时确认显示相关的 GPIO 电源是否和其他模块有冲突；

如果 DTS 确认无误，可以在内核代码搜索串口中的关键字，通过二分法在各个模块加载的位置 while 住，逐步确认导致闪屏问题的点；

7. VOP 优先级配置问题

如果 VOP 优先级没有被配置最高，有可能在内核加载阶段被其他 IP 抢占总线导致闪屏问题，该问题一般会在 SDK 发布前 Fix。

8. 测试相关电源和信号

如果以上还未找到闪屏问题，请使用示波器抓取 CLK、DATA 和电源等相关信号从 U-Boot 到内核阶段的波形图并提交 redmine。

kernel 驱动

驱动目录

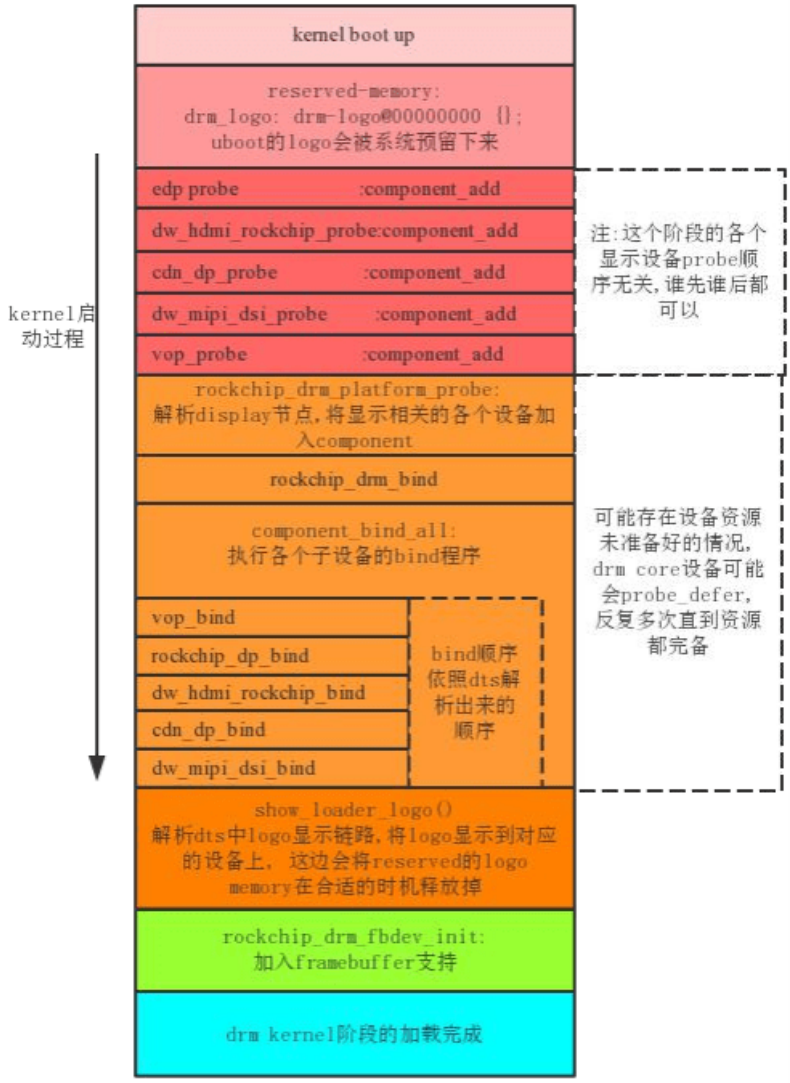
```
drivers/gpu/drm/rockchip/
drivers/gpu/drm/bridge/analogix/
drivers/gpu/drm/bridge/synopsys/
drivers/phy/rockchip/
```

驱动文件

Driver	File	Doc
Core	rockchip_drm_drv.c rockchip_drm_fb.c rockchip_drm_fbdev.c rockchip_drm_gem.c rockchip_drm_logo.c rockchip_drm_direct_show.c panel-simple.c	rockchip-drm.txt or rockchip-drm.yaml
VOP	rockchip_drm_vop.c rockchip_vop_reg.c rockchip_drm_vop2.c rockchip_vop2_reg.c	rockchip-vop.txt or rockchip-vop.yaml
RGB	rocckhip_rgb.c phy-rockchip-inno-video-combo-phy.c	rockchip-rgb.txt
LVDS	rockchip_lvds.c phy-rockchip-inno-video-combo-phy.c	rockchip-lvds.txt
MIPI-DSI	dw-mipi-dsi.c phy-rockchip-inno-dsidphy.c phy-rockchip-inno-video-combo-phy.c dw-mipi-dsi2-rockchip.c phy-rockchip-samsung-dcphy.c	dw_mipi_dsi_rockchip.txt phy-rockchip-inno-mipi-dphy.txt
eDP	analogix_dp_core.c analogix_dp-rockchip.c analogix_dp_reg.c phy-rockchip-dp.c	analogix_dp-rockchip.txt analogix_dp.txt rockchip-dp-phy.txt
DP	cdn-dp-core.c cdn-dp-reg.c dw-dp.c phy-rockchip-usbdp.c phy-rockchip-samsung-hdptx-hdmi.c	cdn-dp-rockchip.txt
HDMI	inno_hdmi.c dw-hdmi.c dw_hdmi-rockchip.c phy-rockchip-inno-hdmi-phy.c dw-hdmi-qp.c phy-rockchip-samsung-hdptx-hdmi.c	inno_hdmi-rockchip.txt dw_hdmi-rockchip.txt phy-rockchip-inno-hdmi-phy.txt
TVE/CVBS	rockchip_drm_tve.c	rockchip_drm_tve.txt

以上驱动文件在不同的 kernel 版本可能会做些小的调整，但多数驱动文件和框架部分基本不会变化，查阅的时候可以根据实际情况调整。

驱动加载流程



需要注意的是，DRM 驱动是一系列显示相关模块的驱动的结合，他包含了 backlight、panel、rgb、lvds、dsi、edp、lvds、hdmi、vop 等等显示通路上的依赖模块。只有这些相互依赖的模块都加载起来，整个 drm 系统才能启动成功。

因为这些复杂的依赖关系，在 drm 系统初始化的过程中，可能会出现某个资源暂时未就绪，而导致某个模块暂时无法顺利加载的情况，为了解决这种问题，drm 驱动利用了 Linux 驱动 中的 `deferred probe` 机制，当发现某个依赖的资源未就绪的时候，驱动返回 `-EPROBE_DEFER(-517)`，然后退出。Linux kernel 会在稍后再次尝试加载这个驱动，直到依赖的资源就绪，驱动顺利加载为止。

```
[1.747190] rockchip-drm display-subsystem: bound fdd90000.vop (ops vop2_component_ops)
[1.747877] dwhdmi-rockchip fde80000.hdmi: registered ddc I2C bus driver
[1.748022] rockchip-drm display-subsystem: bound fde80000.hdmi (ops dw_hdmi_rockchip_ops)
[1.748676] dwhdmi-rockchip fdea0000.hdmi: registered ddc I2C bus driver
[1.748807] rockchip-drm display-subsystem: bound fdea0000.hdmi (ops dw_hdmi_rockchip_ops)
[1.748840] dw-mipi-dsi2 : [drm:dw_mipi_dsi2_bind] *ERROR* Failed to find panel or bridge: -517
[1.755174] panel-simple-dsi fde20000.dsi.0: failed to get power regulator: -517
[1.759296] brd: module loaded
[1.764374] loop: module loaded
[1.764528] zram: Added device: zram0
[1.764698] system_heap: orders[0] = 6
[2.248871] imx415 5-001a: supply dovdd not found, using dummy regulator
[2.416673] rockchip-drm display-subsystem: bound fdd90000.vop (ops vop2_component_ops)
[2.418711] dwhdmi-rockchip fde80000.hdmi: registered ddc I2C bus driver
[2.420336] rockchip-drm display-subsystem: bound fde80000.hdmi (ops dw_hdmi_rockchip_ops)
[2.421725] dwhdmi-rockchip fdea0000.hdmi: registered ddc I2C bus driver
[2.422291] rockchip-drm display-subsystem: bound fdea0000.hdmi (ops dw_hdmi_rockchip_ops)
[2.422318] dw-mipi-dsi2 : [drm:dw_mipi_dsi2_bind] *ERROR* Failed to find panel or bridge: -517
[2.433888] input: adc-keys as /devices/platform/adc-keys/input/input3

[2.466237] rockchip-drm display-subsystem: bound fdd90000.vop (ops vop2_component_ops)
[2.468705] dwhdmi-rockchip fde80000.hdmi: registered ddc I2C bus driver
[2.469751] rockchip-drm display-subsystem: bound fde80000.hnd fde20000.dsi (ops dw_mipi_dsi2_ops)
[2.472282] rockchip-drm display-subsystem: bound fde50000.dp (ops dw_dp_component_ops)
[2.472319] rockchip-drm display-subsystem: bound fde60000.dp (ops dw_dp_component_ops)
[2.531892] rockchip-drm display-subsystem: [drm] fb0: rockchipdrmfb frame buffer device
```

```
[2.532850] [drm] Initialized rockchip 3.0.0 20140818 for display-subsystem on minor 0
```

从上面的 log 我们可以看到，在第 6 行和第 18 行，mipi dsi 驱动因为找不到 panel 或者 bridge 这个依赖资源，而返回 `-EPROBE_DEFER` 退出，一直到第 23 行，dsi 驱动获取到依赖的资源 bind 成功，最终看到整个 drm 驱动完成加载的标准 log。


```
[2.532850] [drm] Initialized rockchip 3.0.0 20140818 for display-subsystem on minor 0
```


DTS 配置

基础配置

在一颗 SOC 上，可能有多个 VOP，HDMI，eDP，DP，MIPI，Panel 模块，根据具体产品定义，一款产品可能只需要使用到其中一部分模块来组成显示通路。具体使用哪些模块，以及这些模块之间如何衔接则通过 dts 配置。

在每一个支持 drm 显示功能的 soc 的核心 dtsi 里面，都会有如下 `display_subsystem` 节点：

```
display_subsystem: display-subsystem {
                compatible = "rockchip,display-subsystem";
                memory-region = <&drm_logo>, <&drm_cubic_lut>;
                memory-region-names = "drm-logo", "drm-cubic-lut";
                ports = <&vop_out>;
                devfreq = <&dmc>;

                route {
                        route_dsi0: route-dsi0 {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp0_out_dsi0>;
                        };
                        route_dsi1: route-dsi1 {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp0_out_dsi1>;
                        };
                        route_edp: route-edp {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp0_out_edp>;
                        };
                        route_hdmi: route-hdmi {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp1_out_hdmi>;
                        };
                        route_lvds: route-lvds {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp1_out_lvds>;
                        };
                        route_rgb: route-rgb {
                                status = "disabled";
                                logo,uboot = "logo.bmp";
                                logo,kernel = "logo_kernel.bmp";
                                logo,mode = "center";
                                charge_logo,mode = "center";
                                connect = <&vp2_out_rgb>;
```

```
                        };
                };
        };
```

该节点控制 `rockchip_drm_drv.c` 驱动的加载。

memory-region 控制 logo 显示 buffer 的分配和传递，该属性在系统启动时会在 u-boot 中被动态修改，然后传递给内核。

route_xxx 节点控制着各个接口开机 logo 功能是否开启，默认处于关闭状态。

以 route_rgb 为例，解释相关属性的功能：

```
route_rgb: route-rgb {
    status = "disabled";
        logo,uboot = "logo.bmp";
        logo,kernel = "logo_kernel.bmp";
        logo,mode = "center";
        charge_logo,mode = "center";
        connect = <&vp2_out_rgb>;
};
```

`logo,uboot`、 `logo,kernel` 描述 U-Boot 阶段和 Kernel 阶段 logo 图片的名字，需要和真实的图片资源匹配。

`logo,mode` `charege_logo,mode` 控制 logo 图片的显示模式，有 `center` 和 `fullscreen` 两种模式。

`connect` 描述该显示接口和 VOP 的哪个 Video Port(VOP2.0 架构) 或者哪个 VOP（VOP1.0 架构)连接 。一般根据实际使用情况配置。

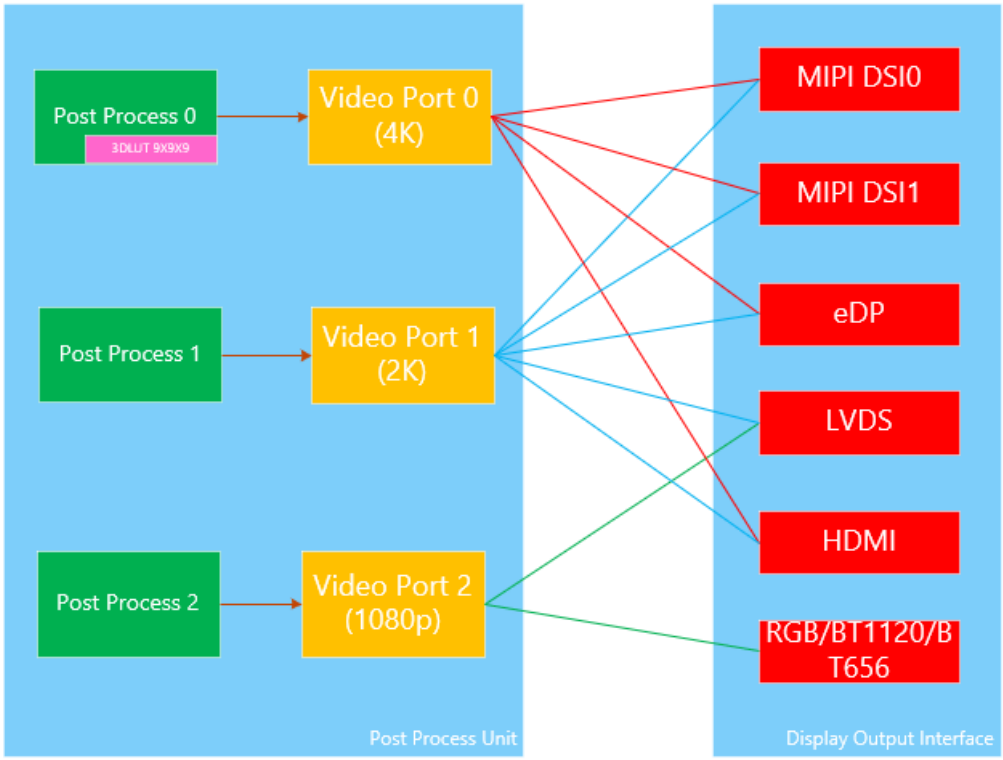
`display_subsystem` 节点的大部分属性不需要修改，一般只需要根据实际产品需求，开关对应的 logo 节点即可。

`vop` 节点：

```
vop: vop@fe040000 {
    compatible = "rockchip,rk3568-vop";
    reg = <0x0 0xfe040000 0x0 0x3000>, <0x0 0xfe044000 0x0 0x1000>;
    reg-names = "regs", "gamma_lut";
    rockchip,grf = <&grf>;
    interrupts = <GIC_SPI 148 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&cru ACLK_VOP>, <&cru HCLK_VOP>, <&cru DCLK_VOP0>, <&cru DCLK_VOP1>, <&cru DCLK_VOP2>;
    clock-names = "aclk_vop", "hclk_vop", "dclk_vp0", "dclk_vp1", "dclk_vp2";
    iommus = <&vop_mmu>;
    power-domains = <&power RK3568_PD_VO>;
    status = "disabled";

    vop_out: ports {
        ......
        vp0: port@0 {
            ......
            vp0_out_dsi0: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&dsi0_in_vp0>;
            };

            vp0_out_dsi1: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&dsi1_in_vp0>;
            };

            vp0_out_edp: endpoint@2 {
                reg = <2>;
                remote-endpoint = <&edp_in_vp0>;
            };

            vp0_out_hdmi: endpoint@3 {
                reg = <3>;
                remote-endpoint = <&hdmi_in_vp0>;
            };
        };

        vp1: port@1 {
            ......
            vp1_out_dsi0: endpoint@0 {
```

```
                reg = <0>;
                remote-endpoint = <&dsi0_in_vp1>;
            };

            vp1_out_dsi1: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&dsi1_in_vp1>;
            };

            vp1_out_edp: endpoint@2 {
                reg = <2>;
                remote-endpoint = <&edp_in_vp1>;
            };

            vp1_out_hdmi: endpoint@3 {
                reg = <3>;
                remote-endpoint = <&hdmi_in_vp1>;
            };

            vp1_out_lvds: endpoint@4 {
                reg = <4>;
                remote-endpoint = <&lvds_in_vp1>;
            };
        };

        vp2: port@2 {
            ......
            vp2_out_lvds: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&lvds_in_vp2>;
            };

            vp2_out_rgb: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&rgb_in_vp2>;
            };
        };
    };
};
```

该节点描述 VOP 硬件资源，控制着 vop 驱动的加载 `rockchip_drm_vop.c/rockchip_drm_vop2.c`， 它描述了如下的显示通路连接关系：



`vop_out: ports` 节点描述 VOP 的输出通道，vp0/1/2 对应 VOP 上 Video Port0/1/2 三个独立的输出通路。

`vp0/1/2 : port` 下的 endpoint 节点描述 VP 和显示接口的连接关系，以上面的 dts 描述为例：vp0 节点下有 `vp0_out_dsi0`，`vp0_out_dsi1`， `vp0_out_edp`， `vp0_out_hdmi` 四个节点，说明 vp0 可以和 dsi0、dsi1、edp、hdmi 四个显示接口连接。

每个 endpoint 通过 `remote-endpoint` 属性和对应的显示接口组成一个连接通路，比如和 hdmi 显示接口的连接：

```
hdmi: hdmi@fe0a0000 {
    compatible = "rockchip,rk3568-dw-hdmi";
    ......
    status = "disabled";

    ports {
        ......
        port@0 {
            reg = <0>;
            ......
            hdmi_in_vp0: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&vp0_out_hdmi>;
                status = "disabled";
            };

            hdmi_in_vp1: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&vp1_out_hdmi>;
                status = "disabled";
            };
        };
    };
};
```

结合上面的 dts 描述我们可以知道，在 rk3568 上，hdmi 可以和 vop 的 vp0，vp1 连接。

需要注意的是，一个显示接口在同一个时刻只能和一个 vp 连接，所以在具体的板级配置中，需要在 dts 中把要使用的通路打开，把不使用的通路设置为 `disabled` 状态。

比如在某款产品上，希望 HDMI 连接在 vp0 上，则 dts 中需要做如下设置：

```
&hdmi {
        status = "okay";
};

&hdmi_in_vp0 {
        status = "okay";
};

&hdmi_in_vp1 {
        status = "disabled";
};

&route_hdmi {
        status = "okay";
        connect = <&vp0_out_hdmi>;
};
```

指定图层分配策略

VOP2 采用统一显示架构，各个独立的 Video Port 共享 VOP 内部的所有图层资源，而且这些图层需要排他性的使用，即某个图层在同一时刻只能为其中一个 Video Port 所独占。

为了充分合理的使用所有图层资源，我们会根据当前产品 dts 配置的接口类型和数量在 U-Boot 中生成了一种默认的图层分配策略，如果有些产品没有开 U-Boot logo 显示或者对图层使用有特殊的需求，可以参考下面的写法在 dts 根据需求自行指定图层分配策略：

rockchip,plane-mask：指定分配给该 VP 的图层 ID 掩码集合，图层 ID 定义在 `dt-bindings/display/rockchip_vop.h` 中。

rockchip,primary-plane：指定 primary 图层，当前 VP 的 primary 图层一定是 rockchip,plane-mask 中的一个，我们一般选用 Smart 或者 Esmart 图层。

图层分配的基本原则是：把所有图层（rk3568 有 6 个图层，rk3588 有 8 个图层）平均分配给各个使用的 VP，不使用的 VP 一般不分配图层。由于 VOP 内部不同类型（Cluster，Esmart，Smart）的图层，性能，限制不同，一般推荐各种类型的图层平均搭配分配。

如下是一个 RK3568 上的典型图层分配参考， RK3568 VOP 一共有 6 个图层（2 Cluster + 2 Esmart + 2 Smart），该配置支持三屏异显。一般，我们尽量给使用场景最多的屏幕(主屏) 对应的 VP 分配三个以上的图层（在该应用案例下是 VP1），其他接口尽量分配不少于两个图层。

```
#include <dt-bindings/display/rockchip_vop.h>        // 图层 ID 定义头文件

&vp0 {
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_CLUSTER1 | 1 << ROCKCHIP_VOP2_SMART1)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_SMART1>;
};

&vp1 {
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_CLUSTER0 | 1 << ROCKCHIP_VOP2_ESMART0 | 1 <<
ROCKCHIP_VOP2_SMART0)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_SMART0>;
};

&vp2 {
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_ESMART1)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_ESMART1>;
};
```

配置生效后，系统启动的时候可以从 U-Boot 和 Linux kernel 的启动 log 中看到对应的 plane mask 解析信息。

```
Rockchip UBOOT DRM driver version: v1.0.1
VOP have 3 active VP
vp0 have layer nr:2[1 5 ], primary plane: 5
vp1 have layer nr:3[0 2 4 ], primary plane: 4
vp2 have layer nr:1[3 ], primary plane: 3
Using display timing dts
dsi@fe060000:  detailed mode clock 132000 kHz, flags[8000000a]
    H: 1080 1095 1097 1127
    V: 1920 1935 1937 1952
bus_format: 100e
VOP update mode to: 1080x1920p0, type: MIPI0 for VP1
VOP VP1 enable Smart0[654x270->654x270@213x825] fmt[2] addr[0x7df04000]
final DSI-Link bandwidth: 876 Mbps x 4
disp info 0, type:11, id:0
xfer: num: 2, addr: 0x50
xfer: num: 2, addr: 0x50
```

```
[2.314574] panel-simple-dsi fe060000.dsi.0: Specify missing connector_type
[2.315764] rockchip-vop2 : [drm:vop2_bind] vp0 assign plane mask: 0x22, primary plane phy id: 5
[2.315807] rockchip-vop2 : [drm:vop2_bind] vp1 assign plane mask: 0x15, primary plane phy id: 4
[2.315828] rockchip-vop2 : [drm:vop2_bind] vp2 assign plane mask: 0x8, primary plane phy id: 3
[2.316713] rockchip-drm display-subsystem: bound fe040000.vop (ops vop2_component_ops)
[2.317966] rockchip-drm display-subsystem: bound fe0c0000.edp (ops rockchip_dp_component_ops)
[2.318378] dwhdmi-rockchip : Detected HDMI TX controller v2.11a with HDCP (DWC HDMI 2.0 TX PHY)
[2.319625] dwhdmi-rockchip : registered DesignWare HDMI I2C bus driver
[2.321857] rockchip-drm display-subsystem: bound fe0a0000.hdmi (ops dw_hdmi_rockchip_ops)
[2.322069] rockchip-drm display-subsystem: bound fe060000.dsi (ops dw_mipi_dsi_rockchip_ops)
```

RK3566 图层分配策略

RK3566 IC 实现上有主图层和镜像图层的区别，即镜像图层 Cluster1 只能从主图层 Cluster0 对应的地址取数，同理 Esmart1/Smart1 只能从 Esmart0/Smart0 对应的地址取数，所以我们需要保证主图层被优先使用。正常产品的 U-Boot 显示驱动中会根据显示接口的类型设置好 plane-mask 属性，如果有些产品没有开 U-Boot logo 显示，可以在 dts 中按以下规则配置：

1. 只有一个屏显示，可以使用如下配置：

```
#include <dt-bindings/display/rockchip_vop.h>   //图层 ID 定义头文件

&vpx {  //x 取决于使用的 vp id，如 vp0
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_CLUSTER0 | 1 << ROCKCHIP_VOP2_ESMART0 | 1 <<
ROCKCHIP_VOP2_SMART0 | 1 << ROCKCHIP_VOP2_CLUSTER1 | 1 << ROCKCHIP_VOP2_ESMART1 | 1 <<
ROCKCHIP_VOP2_SMART1)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_SMART0>;
};
```

2. 有两个屏显示【RK3566 目前只有 android 产品支持双显，且要求两个屏刷新帧率一致】，那我们让不支持热插拔设备(即始终连接显示的通路)使用主图层，另一个通路使用镜像图层：

```
#include <dt-bindings/display/rockchip_vop.h>   //图层 ID 定义头文件

&vpx {  //x 取决于使用的 vp id，如 vp0
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_CLUSTER0 | 1 << ROCKCHIP_VOP2_ESMART0 | 1 <<
ROCKCHIP_VOP2_SMART0)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_SMART0>;
};  //单显或者双显时不支持热插拔的主显示设备使用主图层

&vpx {  //x 取决于使用的 vp id，如 vp1
    rockchip,plane-mask = <(1 << ROCKCHIP_VOP2_CLUSTER1 | 1 << ROCKCHIP_VOP2_ESMART1 | 1 <<
ROCKCHIP_VOP2_SMART1)>;
    rockchip,primary-plane = <ROCKCHIP_VOP2_SMART1>;
};
```

把某个图层设置为鼠标层

应用如果使用 Atomic 显示接口，可以不区分图层的类型(primary, overlay, cursor)，只需要根据图层支持的格式，使用任意可以使用的图层做各种类型的显示。

但是在 Linux 系统(非 Android) 下，还有一些应用使用 legacy 的 API，这些 API 对图层类型比较在意，比如使用 primary 图层显示桌面背景，使用 overlay 图层播放视频，使用 cursor 图层显示鼠标。

Rockchip drm 驱动默认只注册 primary 图层和 overlay 图层，不注册 cursor 图层，如果一些特殊的 Linux 系统希望使用 cursor 图层，可以在 dts 中对应的 vp 节点下设置 `cursor-win-id` 属性，为该 VP 对应的 crtc 分配一个 cursor 图层。

```
&vp0 {
    cursor-win-id = <ROCKCHIP_VOP2_CLUSTER0>;
};
```

如果是 Linux 系统（buildroot，Debian 等非 Android 系统），指定 Cluster 图层为鼠标层的话，要配合 Linux SDK 提供的 `libdrm-cursor` 库。具体细节可参考《Rockchip_Developer_Guide_Debian_CN.pdf》。

禁止图层迁移

某些 Linux 系统可能希望每个 crtc 上的图层都是唯一独占的，不在 crtc 之间做图层迁移，可以在 vop 节点下设置 `disable-win-move` 打开该功能。

```
&vop {
        disable-win-move;
}
```

Display feature

各平台 VOP 基础特性

SOC	VOP-version	VOP base feature									
		8K	4K	MMU	i-MODE	A/I FBDC	MULTI AREA	BCSH	gamma	3D-LUT	POST-SCALE
RK3066/PX2	V1.0	×	×	×	×	×	×	×	√	×	×
RK3188/PX3	V1.0	×	×	×	×	×	×	×	√	×	×
RK3126/RK3126C	V1.0	×	×	√	×	×	×	√	√	×	×
RK3128/PX3SE	V1.0	×	×	√	√	×	×	√	√	×	×
RK3036	V1.0	×	×	√	√	×	×	√	√	×	×
RK322X/RK312XH	V1.0	×	√	√	√	×	×	√	×	×	√
RK322XH/RK332X	V1.0	×	√	√	√	×	×	√	×	×	√
SOFIA 3GR	V1.0	×	×	√	×	×	×	√	√	×	×
RV1108	V1.0	×	×	×	√	×	×	√	√	×	×
RK3288	V1.0	×	√	√	×	×	√	√	√	×	√
RK3368/PX5	V1.0	×	√	√	×	√	√	√	√	×	√
RK3399	V1.0	×	√	√	√	√	√	√	√	×	√
RK3326/PX30	V1.0	×	×	√	√	√	√	√	√	×	×
RK3308	V1.0	×	×	×	×	×	×	√	√	×	×
RK1808	V1.0	×	×	√	×	×	×	√	√	×	×
RV1109/RV1126	V1.0	×	×	√	×	×	×	√	√	×	×
RK356X	V2.0	×	√	√	√	√	√	√	√	√	√
RK3588	V2.0	√	√	√	√	√	√	√	√	√	×
RV1103/RV1106	V1.0	×	×	×	×	×	×	√	√	×	×

各平台显示接口最大输出分辨率和协议标准

平台	显示接口	最大输出分辨率	协议标准
RK3036	HDMI	1920x1080@60hz	支持 HDMI 1.4a 协议标准
	CVBS	720x480i@60hz/720x576i@50hz	支持 NTSC/PAL 标准输出
RK312X/PX3SE	RGB	1280x800@60hz	支持 RGB666/sRGB888
	LVDS	1280x800@60hz	支持 VESA 和 JEIDA LVDS 数据格式
	MIPI	1920x1080@60hz	支持 DSI v1.0，DCS v1.0，DPHY v1.0 协议标准
	HDMI	1920x1080@60hz	支持 HDMI 1.4a 协议标准
	CVBS	720x480i@60hz/720x576i@50hz	支持 NTSC/PAL 标准输出
RK322X/RK312XH	HDMI	4096x2160@60hz	支持 HDMI 1.4a 和 2.0 协议标准
	CVBS	720x480i@60hz/720x576i@50hz	支持 NTSC/PAL 标准输出
RK3288	RGB	1920x1080@60hz	支持 GB888/RGB666/sRGB888
	LVDS	单通道：1280x800@60hz 双通道：1920x1080@60hz	支持 VESA 和 JEIDA LVDS 数据格式
	eDP	2560x1600@60hz	支持 DP1.2a 和 eDP1.3 协议标准
	MIPI	单通道：1920x1080@60hz 双通道：2560x1600@60hz	支持 DSI v1.1，DCS v1.1，DPHY v1.1 协议标准
	HDMI	VOP BIG:：3840x2160@60hz VOP LIT： 1920x1080@60hz	支持 HDMI 1.4a 和 2.0 协议标准
RK3308	RGB	1280x800@60hz	支持 RGB888/RGB666/sRGB888/MCU
RK1808	RGB	1280x800@60hz	支持 RGB888/RGB666/sRGB888/MCU
	MIPI	1280X800@60hz	支持 DSI v1.0，DCS v1.0，DPHY v1.0 协议标准
RK322XH/RK332X	HDMI	3840x2160@60hz	支持 HDMI 1.4a 和 2.0 协议标准
	CVBS	720x480i@60hz/720x576i@50hz	支持 NTSC/PAL 标准输出
RK3326/px30	RGB	1280x800@60hz	支持 RGB888/RGB666/sRGB888/MCU
	LVDS	1280x800@60hz	支持 VESA 和 JEIDA LVDS 数据格式
	MIPI	1920x1080@60hz	支持 DSI v1.0，DCS v1.0，DPHY v1.0 协议标准
RK3368/PX5	RGB	1280x800@60hz	支持 RGB888/RGB666/sRGB888/MCU
	LVDS	1280x800@60hz	支持 VESA 和 JEIDA LVDS 数据格式
	MIPI	1920x1080@60hz	支持 DSI v1.0，DCS v1.0， DPHY v1.0 协议标准
	eDP	2560x1600@60hz	支持 DP1.2a 和 eDP1.3 协议标准
	HDMI	4096x2160@60hz	支持 HDMI 1.4a 和 2.0 协议标准
RK3399	MIPI	单通道：1280x800@60hz 双通道：2560x1600@60hz	支持 DSI v1.1，DCS v1.1，DPHY v1.1 协议标准准
	eDP	2560x1600@60hz	支持 DP1.2a 和 eDP1.3 协议标准
	HDMI	VOP BIG: 4096x2160@60hz VOP LIT: 2560x1600@60hz	支持 HDMI 1.4a 和 2.0a 协议标准
	DP	VOP BIG: 3840x2160@60hz VOP LIT: 2560x1600@60hz	支持 DP 1.2 协议标准
RV1109/RV1126	RGB	1920x1080@60hz	支持 RGB888/RGB666/sRGB888/MCU/BT.1120
	MIPI	1920x1080@60hz	支持 DSI v1.1，DCS v1.1， DPHY v1.2 协议标准
RK356X	RGB	1920x1080@60hz	支持 RGB888/RGB666/BT.656/BT.1120

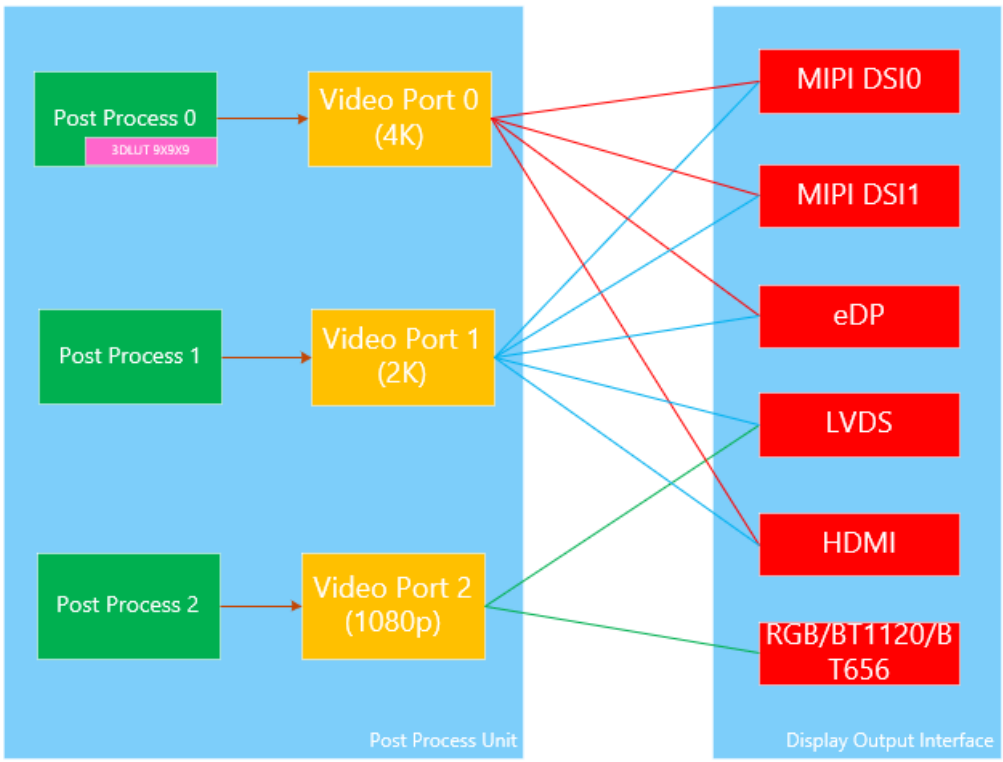
	MIPI	单通道：1920x1080@60hz 双通道：2560x1600@60hz	支持 DSI v1.1，DCS v1.1，DPHY v1.1 协议标准准
	eDP	2560x1600@60hz	支持 DP 1.2a 和 eDP 1.3 协议标准
	HDMI	4096x2160@60hz	支持 HDMI 1.4a 和 2.0a 协议标准
RK3588	RGB	1920x1080@60hz	支持 BT.656/BT.1120
	MIPI	3840x2160@60hz	双 MIPI，支持 DSI v1.1，DCS v1.1， DPHY v2.0，CPHY V1.1 协议标准
	eDP[0]	3840x2160@60hz	双 eDP，支持 DP1.2a 和 eDP1.3 协议标准
	HDMI	7680x4320@60hz	双 HDMI，支持 HDMI 2.1 协议标准
	DP	7680x4320@30hz	双 DP，支持 DP1.4 协议标准
RV1103/RV1106	RGB	1280x720@60hz	支持 RGB666/sRGB888/MCU/BT.656/BT.1120

Note:

1. 在 RK3588 上，eDP 和 HDMI 的 PHY 是 combo 的，即在同一个产品上，使用了 HDMI0 就不能使用 eDP0，HDMI1 和 eDP1 同理。

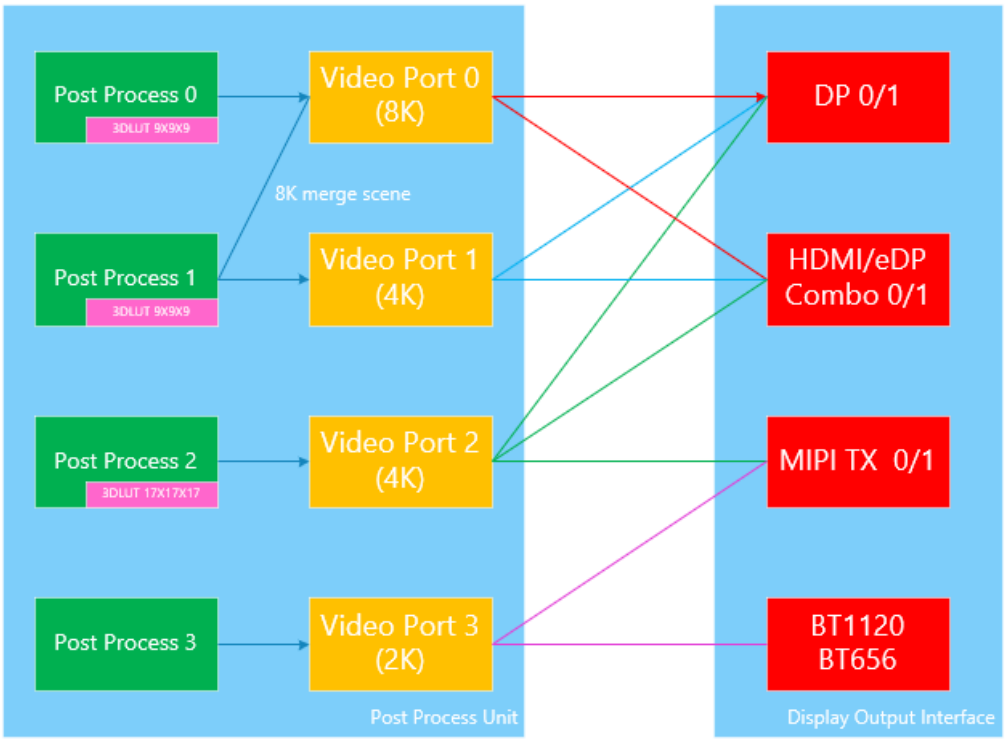
VOP2 平台显示通路

RK3568 VP 和各显示接口的连接关系



更详细的内容请参考《Rockchip RK3568 datasheet》 **Display Interface** 和 **Video Output Processor** 章节。

RK3588 VP 和各显示接口的连接关系



需要注意的是，RK3588 的 HDMI 和 DP 支持 8K 输出，但是在 8K 输出模式下，一个显示接口需要同时占用 VP0 和 VP1 。所以如果产品上需要支持 8K 显示输出，VP1 上要注意不要连接其他显示接口。

更详细的内容请参考 《Rockchip RK3588 datasheet》 **Display Interface** 和 **Video Output Processor** 章节。

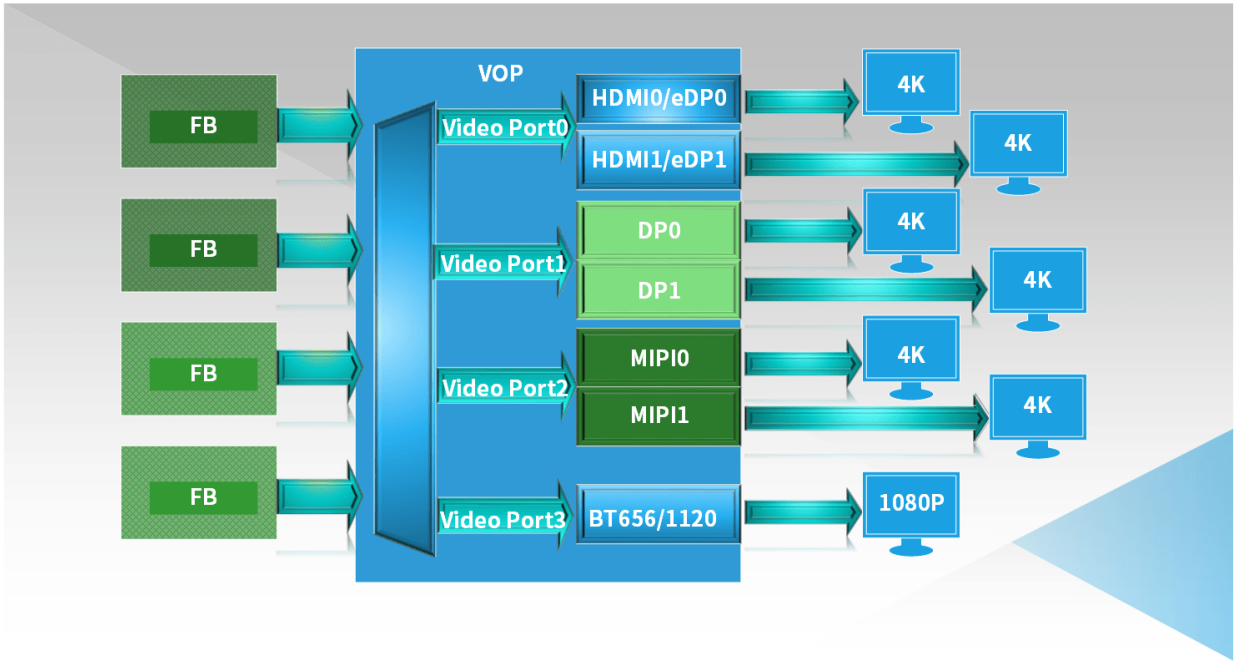
VOP 2.0 架构下的多屏显示

根据前面的介绍，对于采用 VOP 1.0 架构的芯片，如果要实现多屏异显，一般需要多个 VOP，所以对于 VOP 1.0 架构的平台，只有 RK3288，RK3399，PX30 这些包含双 VOP 的芯片才支持双屏异显。

对于 VOP 2.0 架构的芯片，由于采用了统一显示架构，可以更高效的利用 IP 资源，在最基础的条件下，VOP 内部包含几个 Video Port 就能实现几路独立的显示输出。根据 **VOP2 平台显示通路** 章节的内容可以知道，RK3568 有三路独立的 Video Port，RK3588 平台有四路独立的 Video Port, 所以在最基础的条件下，RK3568 和 RK3588 可以分别实现 3 路和 4 路独立的显示输出。

Connector-mirror

VOP2 的 Connector-mirror 技术支持一个 Video Port 同时驱动多路显示接口，输出相同的显示时序并显示相同的内容。



如图所示，在 RK3588 上，通过 connector-mirror 技术，把两路 HDMI/eDP 连接在 VP0 上，把两路 DP 连接在 VP1 上，把两路 MIPI DSI 连接在 VP2 上，VP3 通过 BT656，BT1120 可以同时输出 7 路，四组独立的显示输出，其中每一组（同一个 VIdeo Port 上的两个显示接口）输出的显示时序相同，且显示内容相同。

在这种应用模式下，每一组显示通路输出的最大分辨率受对应的 Video Port 和显示接口的最大分辨率限制。

这种显示特性可以通过 dts 配置开启，在 dts 里面只要把两个显示接口挂接在同一个 VP 上即可：

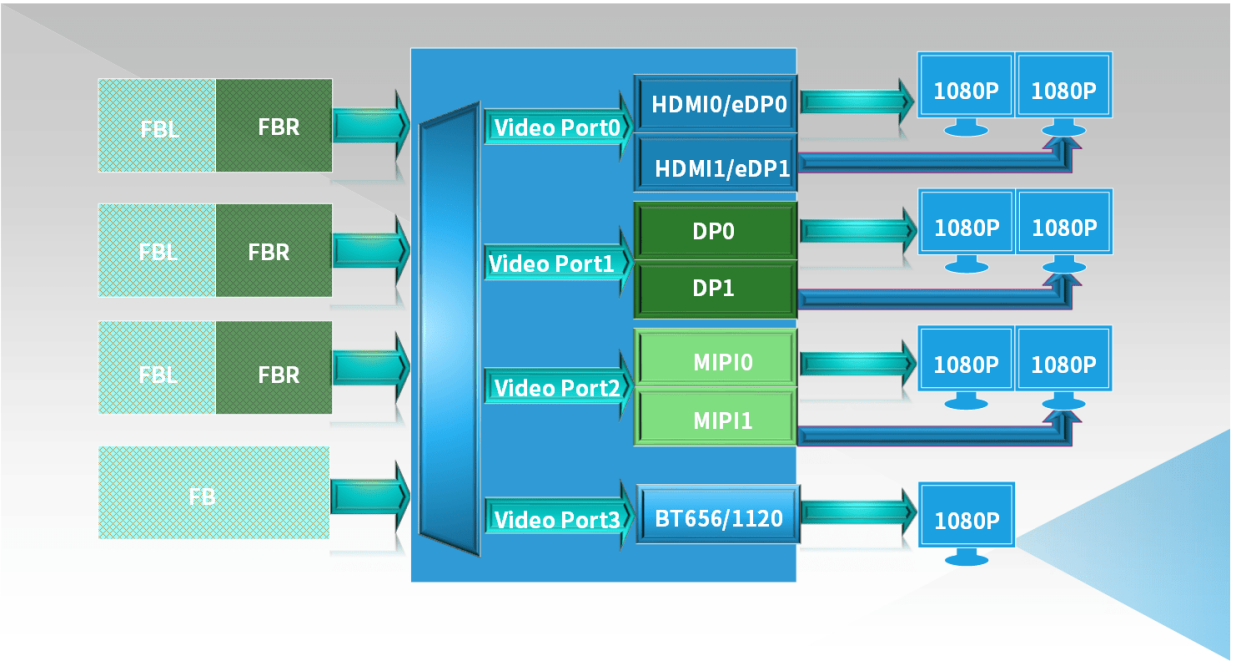
```
&hdmi0 {
        status = "okay";
};

&hdmi1 {
        status = "okay";
};

&hdmi0_in_vp0 {
        status = "okay";
};

&hdmi1_in_vp0 {
        status = "okay";
};

&hdmi0_in_vp1 {
        status = "disabled";
};

&hdmi0_in_vp2 {
        status = "disabled";
};

&hdmi1_in_vp1 {
        status = "disabled";
};

&hdmi1_in_vp2 {
        status = "disabled";
};
```

该配置把 HDMI0 和 HDMI1 挂接在 VP0 上，开启 connector-mirror 功能。

目前 NVR SDK 支持这种功能，Android hwc 暂时不支持这种特性，所以在 Android 系统上一定不能打开这种功能。

Connector-split

VOP2 提供的 Connector-split 功能是一种类似 mipi 双通道模式的技术，可以让一路 Video Port 输出按照水平方向平分成左右两路，同时驱动两个显示接口，显示时序相同，内容独立的画面。



如上图所示，如果在 VP0 上开启 split 模式，则 VP0 的输出可以同时驱动两个显示接口(HDMI0/1、eDP0/1 或者其他能与 VP0 连接的显示接口)，两个显示接口上显示的内容为 VP0 输出的内容水平方向左右平分，比如 VP0 以 3840x1080 的分辨率输出，则两个显示接口各显示 1920x1080 的输出。

通过这种技术，可以在 RK3588 上扩展出 7 路独立的显示输出。

需要注意的是，每一个 VP 上参与 split 输出的两个显示接口，输出的时序，帧率必须相同。

目前只有 RK3588 支持该功能。

这种显示特性可以通过 dts 开关，dts 只要把参与 split 的两个显示接口挂接在同一个 VP 上，且打开左边显示接口的 `split-mode` 属性。

比如，按照如下配置，打开 hdmi0 和 hdmi1 在 VP0 上的 split 功能：

```
&hdmi0 {
        status = "okay";
};

&hdmi1 {
        status = "okay";
        split-mode;
};

&hdmi0_in_vp0 {
        status = "okay";
};

&hdmi1_in_vp0 {
        status = "okay";
};

&hdmi0_in_vp1 {
        status = "disabled";
};

&hdmi0_in_vp2 {
        status = "disabled";
};

&hdmi1_in_vp1 {
        status = "disabled";
};

&hdmi1_in_vp2 {
        status = "disabled";
};
```

通过 Connector-split 功能，可以扩展出更多的多屏异显功能。

在驱动实现上，为了方便上层应用适配，尽量和 MIPI 双通道技术接近，屏蔽底层实现差异，每个 Video Port 上进行 split 的两个显示接口只会向 drm 系统注册一个 encoder 和 connector，所以在用户空间，每一 CRTC(Video Port) 上只会看到一个 connector 设备，这个信息可以通过 modetest 的输出确认。

对于 Android 应用，希望每一个屏幕都对应一个独立的显示设备，针对这种需求，Rockchip 平台的 Android hwc 有做针对性的优化，具体请参考《DrmHwc2 多屏拼接异显功能说明》文档。

硬件相关

RGB 输出/TTL 模式硬件连接

VOP 1.0 RGB 接口硬件连接方式

1. 判断是 VOP 1.0 还是 VOP 2.0 的设计，可以从 3.1 章节的 VOP version 中查询；
2. 对于 SOC 支持 24bit RGB 输出的硬件连接方式

interface	RGB parallel						
display mode index	mode0	mode1	mode2	mode3	mode4	mode5	
display mode	RGB parallel 24 bit	RGB parallel 18 bit	RGB parallel 18 bit	RGB parallel 16 bit	RGB parallel 16 bit	serial 3x8/4x8	serial 3x6/4x6
dclk	dclk						
vsync	vsync						
hsync	hsync						
den	den						
data	data[23:0]	data[23:18] data[15:10] data[7:2]	data[17:0]	data[23:19] data[15:10] data[7:3]	data[15:0]	data[7:0]	data[7:2]
D23	R7	R5	—	R4	—	—	—
D22	R6	R4	—	R3	—	—	—
D21	R5	R3	—	R2	—	—	—
D20	R4	R2	—	R1	—	—	—
D19	R3	R1	—	R0	—	—	—
D18	R2	R0	—	—	—	—	—
D17	R1	—	R5	—	—	—	—
D16	R0	—	R4	—	—	—	—
D15	G7	G5	R3	G5	R4	—	—
D14	G6	G4	R2	G4	R3	—	—
D13	G5	G3	R1	G3	R2	—	—
D12	G4	G2	R0	G2	R1	—	—
D11	G3	G1	G5	G1	R0	—	—
D10	G2	G0	G4	G0	G5	—	—
D9	G1	—	G3	—	G4	—	—
D8	G0	—	G2	—	G3	—	—
D7	B7	B5	G1	B4	G2	D7	D5
D6	B6	B4	G0	B3	G1	D6	D4
D5	B5	B3	B5	B2	G0	D5	D3
D4	B4	B2	B4	B1	B4	D4	D2
D3	B3	B1	B3	B0	B3	D3	D1
D2	B2	B0	B2	—	B2	D2	D0
D1	B1	—	B1	—	B1	D1	—
D0	B0	—	B0	—	B0	D0	—

3. 对于 SOC 支持 18bit RGB 输出的硬件连接方式

interface	RGB parallel			
display mode index	mode2	mode4	mode5	
display mode	RGB parallel 18 bit	RGB parallel 16 bit	serial 3x8/4x8	serial 3x6/4x6
dclk	dclk			
vsync	vsync			
hsync	hsync			
den	den			
data	data[17:0]	data[15:0]	data[7:0]	data[7:2]
D17	R5	—	—	—
D16	R4	—	—	—
D15	R3	R4	—	—
D14	R2	R3	—	—
D13	R1	R2	—	—
D12	R0	R1	—	—
D11	G5	R0	—	—
D10	G4	G5	—	—
D9	G3	G4	—	—
D8	G2	G3	—	—
D7	G1	G2	D7	D5
D6	G0	G1	D6	D4
D5	B5	G0	D5	D3
D4	B4	B4	D4	D2
D3	B3	B3	D3	D1
D2	B2	B2	D2	D0
D1	B1	B1	D1	—
D0	B0	B0	D0	—

4. 对于 VOP 1.0 中 MCU 接口 DATA 线连接方式和 RGB parallel 的连接方式一致，需要注意的是，RGB parallel 中的 4 个 时钟信号复用成 MCU 接口的控制信号，以下是具体的对应关系：

dclk	mcu_rs	1表示发送的是数据，0表示发送的是命令
vsync	mcu_csn	片选信号，低有效
hsync	mcu_wrn	写使能信号，上升沿有效
den	mcu_rdn	1：表示发数据到屏，0：表示从屏读数据

VOP 2.0 及之后的版本 RGB 接口硬件连接方式

interface	RGB parallel		
display mode index	mode0	mode1	mode3
display mode	RGB parallel 24 bit	RGB parallel 18 bit	RGB parallel 16 bit
dclk	dclk		
vsync	vsync		
hsync	hsync		
den	den		
data	data[23:0]	data[23:18] data[15:10] data[7:2]	data[23:19] data[15:10] data[7:3]
D23	R7	R5	R4
D22	R6	R4	R3
D21	R5	R3	R2
D20	R4	R2	R1
D19	R3	R1	R0
D18	R2	R0	—
D17	R1	—	—
D16	R0	—	—
D15	G7	G5	G5
D14	G6	G4	G4
D13	G5	G3	G3
D12	G4	G2	G2
D11	G3	G1	G1
D10	G2	G0	G0
D9	G1	—	—
D8	G0	—	—
D7	B7	B5	B4
D6	B6	B4	B3
D5	B5	B3	B2
D4	B4	B2	B1
D3	B3	B1	B0
D2	B2	B0	—
D1	B1	—	—
D0	B0	—	—

不同 display mode index 对应的软件配置

display mode index	bus format
mode0	MEDIA_BUS_FMT_RGB888_1X24
mode1	MEDIA_BUS_FMT_RGB666_1X24_CPADHI
mode2	MEDIA_BUS_FMT_RGB666_1X18
mode3	MEDIA_BUS_FMT_RGB565_1X24_CPADLO
mode4	MEDIA_BUS_FMT_RGB565_1X16
mode5	MEDIA_BUS_FMT_RGB888_3X8

BT.656 和 BT.1120 的硬件连接方式

SOC TX 引脚	Clock	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	软件 bus_format 配置
BT.656 接法	Clock	-	-	-	-	-	-	-	-	D7	D6	D5	D4	D3	D2	D1	D0	MEDIA_BUS_FMT_UYVY8_2X8
BT.1120 接法1	Clock	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	C7	C6	C5	C4	C3	C2	C1	C0	MEDIA_BUS_FMT_YUYV8_1X16
BT.1120 接法2	Clock	C7	C6	C5	C4	C3	C2	C1	C0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	MEDIA_BUS_FMT_UYVY8_1X16

根据接收端的 Y/U/V 处理顺序不同，如果出现显示颜色不对，还可以在 DTS 中调整不同的 bus_format 来适配，BT.656 和 BT.1120 分别有以下四种配置：

BT.656：

```
#define MEDIA_BUS_FMT_UYVY8_2X8         0x2006
#define MEDIA_BUS_FMT_VYUY8_2X8         0x2007
#define MEDIA_BUS_FMT_YUYV8_2X8         0x2008
#define MEDIA_BUS_FMT_YVYU8_2X8         0x2009
```

BT.1120：

```
#define MEDIA_BUS_FMT_UYVY8_1X16        0x200f
#define MEDIA_BUS_FMT_VYUY8_1X16        0x2010
#define MEDIA_BUS_FMT_YUYV8_1X16        0x2011
#define MEDIA_BUS_FMT_YVYU8_1X16        0x2012
```

MEDIA_BUS_FMT 的定义可以参考：

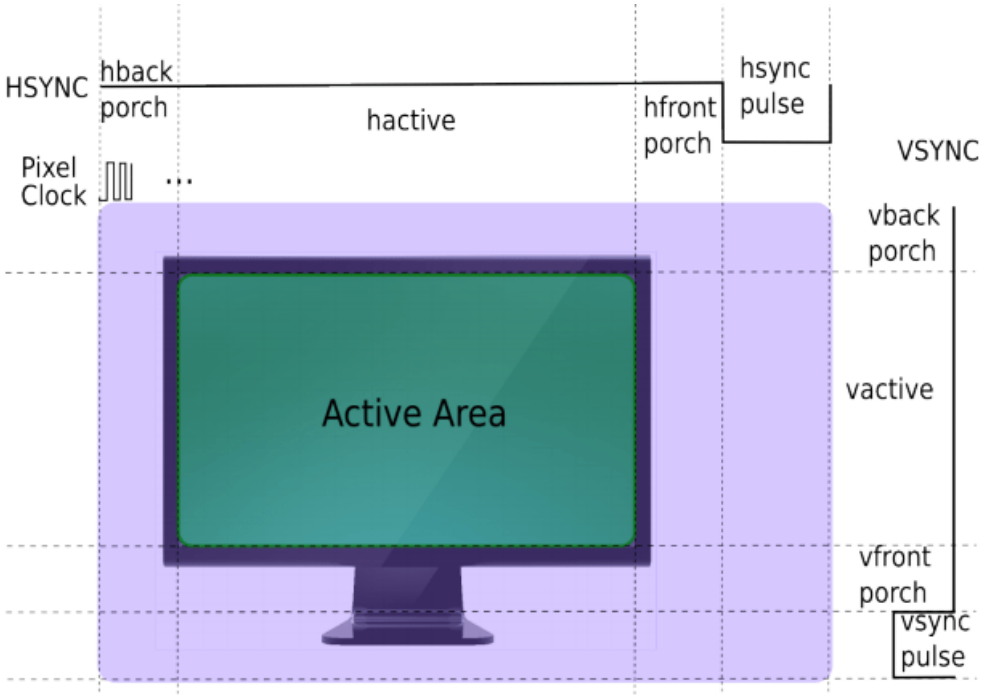
```
./include/uapi/linux/media-bus-format.h
```

LVDS Data Mapping

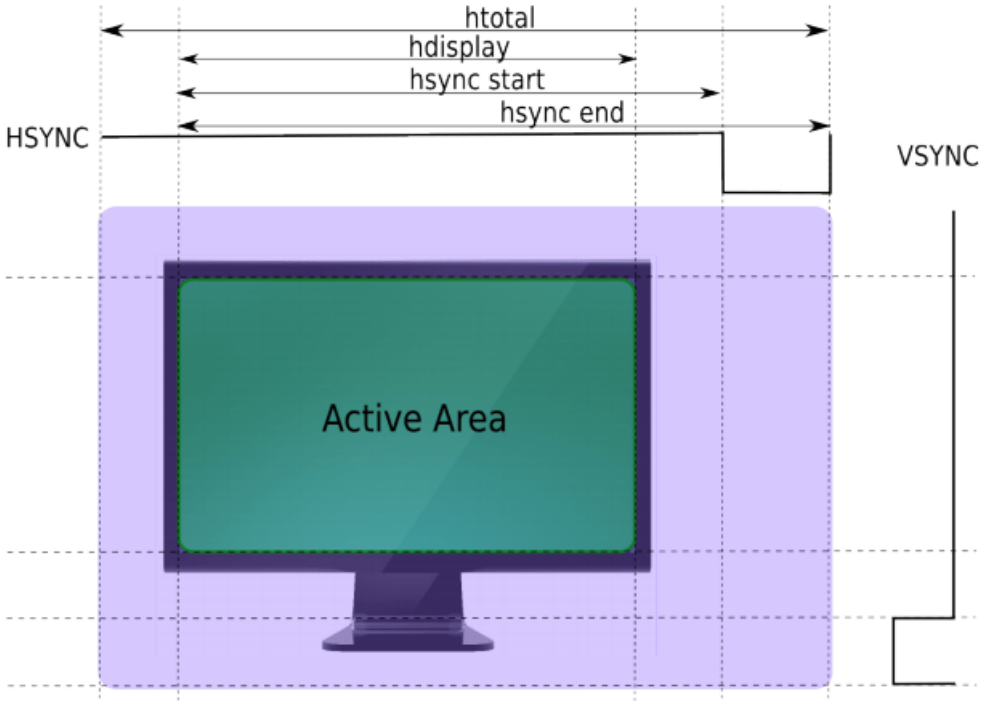
bus-format	Timeslot	Data organization			
		Lane3	Lane2	Lane1	Lane0
MEDIA_BUS_FMT_RGB666_1X7X3_SPWG	0	–	DEN	B1	G0
	1	–	VSYNC	B0	R5
	2	–	HSYNC	G5	R4
	3	–	B5	G4	R3
	4	–	B4	G3	R2
	5	–	B3	G2	R1
	6	–	B2	G1	R0
MEDIA_BUS_FMT_RGB888_1X7X4_SPWG	0	GND	DEN	B1	G0
	1	B7	VSYNC	B0	R5
	2	B6	HSYNC	G5	R4
	3	G7	B5	G4	R3
	4	G6	B4	G3	R2
	5	R7	B3	G2	R1
	6	R6	B2	G1	R0
MEDIA_BUS_FMT_RGB888_1X7X4_JEIDA	0	GND	DEN	B3	G2
	1	B1	VSYNC	B2	R7
	2	B0	HSYNC	G7	R6
	3	G1	B7	G6	R5
	4	G0	B6	G5	R4
	5	R1	B5	G4	R3
	6	R0	B4	G3	R2

扫描时序说明

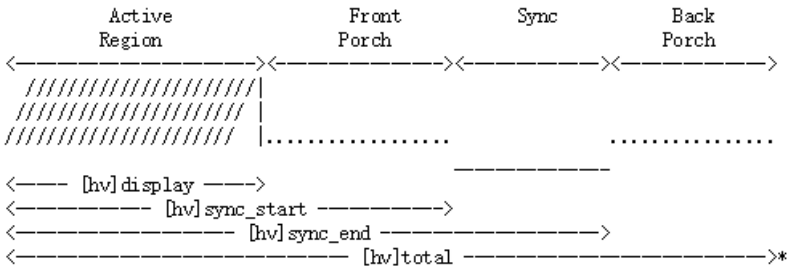
常见的扫描时序图



DRM 对扫描时序的定义



软件配置的对应关系和转换




```
void drm_display_mode_from_videomode(const struct videomode *vm,
                                     struct drm_display_mode *dmode)
{
        dmode->hdisplay = vm->hactive;
        dmode->hsync_start = dmode->hdisplay + vm->hfront_porch;
        dmode->hsync_end = dmode->hsync_start + vm->hsync_len;
        dmode->htotal = dmode->hsync_end + vm->hback_porch;

        dmode->vdisplay = vm->vactive;
        dmode->vsync_start = dmode->vdisplay + vm->vfront_porch;
        dmode->vsync_end = dmode->vsync_start + vm->vsync_len;
        dmode->vtotal = dmode->vsync_end + vm->vback_porch;

        dmode->clock = vm->pixelclock / 1000;

        if (vm->flags & DISPLAY_FLAGS_HSYNC_HIGH)
                dmode->flags |= DRM_MODE_FLAG_PHSYNC;
        ...

        drm_mode_set_name(dmode);
}

void drm_display_mode_to_videomode(const struct drm_display_mode *dmode,
                                   struct videomode *vm)
{
        vm->hactive = dmode->hdisplay;
        vm->hfront_porch = dmode->hsync_start - dmode->hdisplay;
        vm->hsync_len = dmode->hsync_end - dmode->hsync_start;
        vm->hback_porch = dmode->htotal - dmode->hsync_end;

        vm->vactive = dmode->vdisplay;
        vm->vfront_porch = dmode->vsync_start - dmode->vdisplay;
        vm->vsync_len = dmode->vsync_end - dmode->vsync_start;
        vm->vback_porch = dmode->vtotal - dmode->vsync_end;

        vm->pixelclock = dmode->clock * 1000;

        if (dmode->flags & DRM_MODE_FLAG_PHSYNC)
                vm->flags |= DISPLAY_FLAGS_HSYNC_HIGH;
        ...
}
```

查看当前配置的时序

```
rk3568_r:/ # cat /d/dri/0/summary
Video Port0: DISABLED
Video Port1: ACTIVE
    Connector: DSI-1
        bus_format[100a]: RGB888_1X24
        overlay_mode[0] output_mode[0] color_space[0]
    Display mode: 1080x1920p60
        clk[132000] real_clk[132000] type[48] flag[a]
        H: 1080 1095 1097 1127
        V: 1920 1935 1937 1952
    Cluster0-win0: ACTIVE
        win_id: 4
        format: AB24 little-endian (0x34324241)[AFBC] SDR[0] color_space[0]
        rotate: xmirror: 0 ymirror: 0 rotate_90: 0 rotate_270: 0
        csc: y2r[0] r2y[0] csc mode[0]
        zpos: 0
        src: pos[0, 0] rect[1080 x 1920]
        dst: pos[0, 0] rect[1080 x 1920]
        buf[0]: addr: 0x0000000000edb000 pitch: 4352 offset: 0
Video Port2: DISABLED
```

带宽的计算方法

图像的带宽

以 1080P ARGB 格式的图像数据为例：

p0	p0	p0	p0	p1	p1	p1	p1	p2	p2	p2	p2	p3	p3	p3	p3
p4	p4	p4	p4	p5	p5	p5	p5	p6	p6	p6	p6	p7	p7	p7	p7
p8	p8	p8	p8	p9	p9	p9	p9	p10	p10	p10	p10	p11	p11	p11	p11
p12	p12	p12	p12	p13	p13	p13	p13	p14	p14	p14	p14	p15	p15	p15	p15

ARGB 格式一个像素占用的内存大小：4 Byte

1080P ARGB 格式的数据占用内存：1920 x 1080 x 4Byte/pixel = 8,100 Kbyte

如果按 60fps 刷新，占用的带宽是： 8,100 x 60fps = 474.6 Mbyte/s

如果数据格式改成 YUV420SP(NV12):

Single Frame YUV420: NV12

Y1	Y2	Y3	Y4	Y5	Y6
Y7	Y8	Y9	Y10	Y11	Y12
Y13	Y14	Y15	Y16	Y17	Y18
Y19	Y20	Y21	Y22	Y23	Y24
U1	V1	U2	V2	U3	V3
U4	V4	U5	V5	U6	V6

Position in byte stream:

NV12 格式一个像素占用的内存大小：1.5 Byte

1080P NV12 格式的数据占用内存：1920 x 1080 x 1.5Byte/pixel = 3,037.5 Kbyte

如果按 60fps 刷新，占用的带宽是： 3,037.5 x 60fps = 178 Mbyte/s

显示接口的带宽

```
disp_timings0: display-timings {
        native-mode = <&dsi0_timing0>;
        dsi0_timing0: timing0 {
                clock-frequency = <132000000>;
                hactive = <1080>;
                vactive = <1920>;
                hfront-porch = <15>;
                hsync-len = <2>;
                hback-porch = <30>;
                vfront-porch = <15>;
                vsync-len = <2>;
                vback-porch = <15>;
                hsync-active = <0>;
                vsync-active = <0>;
                de-active = <0>;
                pixelclk-active = <1>;
        };
};
```

以上面这张图配置的时序为例，当前这个时序下，按 60 帧刷新需要的 dclk 是：131994240 hz，dts 实际按取整 132000000 hz 配置：

```
htotal = hfp + hsync + hbp + hactive = 15 + 2 + 30 + 1080 = 1,127
vtotal = vfp + vsync + vbp + vactive = 15 + 2 + 15 + 1920 = 1,952
dclk = htotal x vtotal x fps = 1127 x 1952 x 60fps = 131,994,240
```

MIPI 接口上传输的频率是：

```
132M x 3(RGB) x 8(bpc) / 4(lane) / 0.9 = 880 Mbps
```

其中：

x3(RGB)：是每一个 pixel 有 RGB 3 个分量；

x8(bpc)：是每一个分量的位深是 8bit；

/4(lane)：是这么多数据量在 4 lane 上传输，/4 是计算每 lane 的数据量；

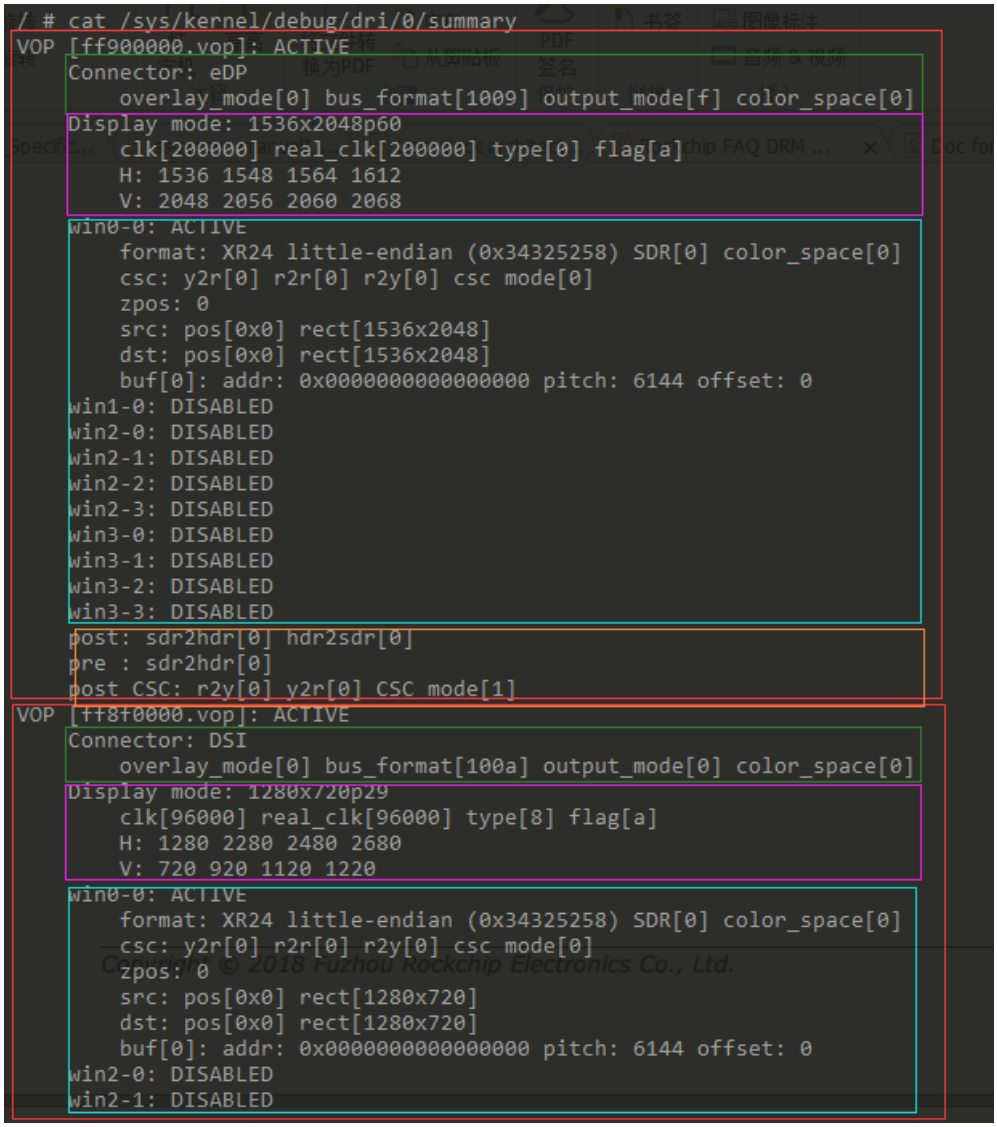
/0.9：是考虑 mipi 时序的传输效率；

常用的 debug 手段

dump 当前的显示状态

使用命令

```
cat /sys/kernel/debug/dri/0/summary
```

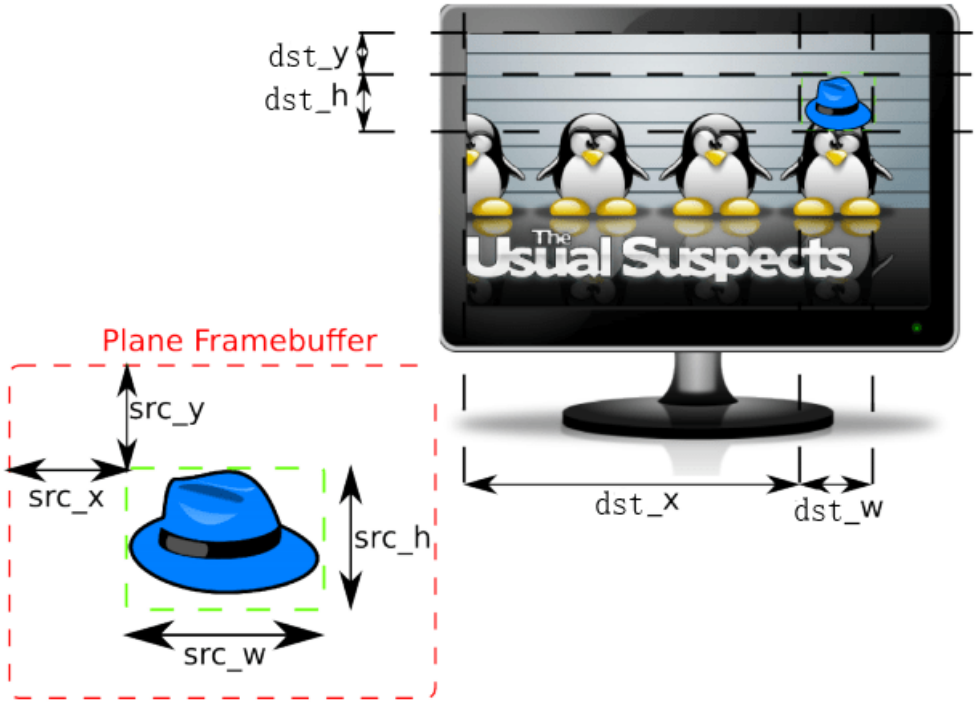


需要注意的是该命令依赖内核的 debugfs 功能，如果内核没有开启 debugfs 功能或者系统没有挂载 debugfs 节点，需要确保内核打开 CONFIG_DEBUG_FS 选项，并通过如下命令手动挂载：

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

参数说明

1. 两个红色方框表示两个显示设备使用的 vop 分别是 ff900000.vop 和 ff8f0000.vop；
2. 绿色部分表示 connector 信息，两个显示设备分别为 eDP 屏和 MIPI 屏；
3. 粉色部分为显示模式，可以知道具体的时序、DCLK 以及帧率，上图中两个设备分别为分辨率为 1536x2048p60 的 eDP 屏和分辨率 1280x720p29 的 MIPI 屏；
4. 蓝色部分是 VOP 图层信息，第一个显示设备打开 win0 图层，大小为 1536x2048 格式为 XRGB 第二个显示设备打开 win0 图层，大小 1280x720 格式为 XRGB， src 和 dst 表示源数据和显示的大小和位置，如果 src 和 dst 的大小不一致，VOP 会进行缩放处理，如下图所示:



5. 橙色部分为 VOP HDR、CSC 的一些状态信息；SDR 表示应用送下来的是 SDR 格式的数据，如果是 HDR 格式的数据，会看到 HDR 的关键字。
6. VOP2 平台的 summary 信息基本和之前平台的一致，只是把之前 VOP 改成了 Video Port，图层名字从原来的 winx， 变成了 Cluster-winx 或者 Esmart-winx:

```
[root@RK3588:/]#
[root@RK3588:/]#
[root@RK3588:/]#
[root@RK3588:/]# cat /sys/kernel/debug/dri/0/summary
Video Port0: DISABLED
Video Port1: DISABLED
Video Port2: DISABLED
Video Port3: ACTIVE
    Connector: DSI-1
        bus_format[100a]: RGB888_1X24
        overlay_mode[0] output_mode[0] color_space[0], eotf:0
    Display mode: 1080x1920p60
        clk[132000] real_clk[132000] type[48] flag[a]
        H: 1080 1095 1099 1129
        V: 1920 1935 1937 1952
    Esmart3-win0: ACTIVE
        win_id: 11
        format: XR24 little-endian (0x34325258) SDR[0] color_space[0] glb_alpha[0xff]
        rotate: xmirror: 0 ymirror: 0 rotate_90: 0 rotate_270: 0
        csc: y2r[0] r2y[0] csc mode[0]
        zpos: 3
        src: pos[0, 0] rect[1080 x 1920]
        dst: pos[0, 0] rect[1080 x 1920]
        buf[0]: addr: 0x0000000000000000 pitch: 4320 offset: 0
```

7. 每个图层下边有一个 format 属性，他可能是 AB24、XR24、NV12 之类的标准 fourcc 值，表示的是这个图层所显示的 framebuffer

   中数据的格式，具体的含义可以参考 drm fourcc。

使用 vop2_dump.sh 脚本 dump 显示信息

由于 VOP2 架构的复杂性，寄存器相关配置量非常大，我们提供了一个脚本可以更方便的 dump vop 相关的显示状态信息。大部分 SDK 里面默认有带这个脚本，如果没有，可以通过如下地址获取：vop2_dump。

需要注意的是，该脚本也依赖 `debugfs`，并且需要内核打开 `CONFIG_DEVMEM` 选项，以方便该脚本利用 `io` 命令读取寄存器配置。

dump 当前显示的 buffer

有时候我们发现屏幕上显示的内容异常，这种异常有可能是 VOP ——> 显示接口 ——> 屏幕/显示器，这条链路上有异常，也有可能是上面的应用绘制的显示数据就是异常的。这时候我们可以把 VOP 图层显示的 buffer 的数据 dump 出来，然后用 7yuv 之类的软件查看，这些数据是否正常。如果这些数据本身就是异常的，那就说明是 userspace 送下来的数据就是有问题的，反之则要排查 VOP 到屏幕之间的这条显示通路。

使用说明

首选需要确保 Linux kernel make menuconfig 打开 CONFIG_ROCKCHIP_DRM_DEBUG 选项。

```
/sys/kernel/debug/dri/0/ff900000.vop/vop_dump # cat dump
  echo dump     > dump to dump one frame
  echo dumpon   > dump to start vop keep dumping
  echo dumpoff > dump to stop keep dumping
  echo dumpn    > dump n is the number of dump times
  dump path is /data/vop_buf
  if fd err = -3 try rm -r /data/vopbuf echo dump1 > dump can fix it
  if fd err = -28 save needed data try rm -r /data/vopbuf
```

1. dump 一帧当前显示的 buffer

   echo dump > /sys/kernel/debug/dri/0/**ff900000.vop**/vop_dump/dump

2. 连续 dump n 帧显示的 buffer

   echo dumpn > /sys/kernel/debug/dri/0/**ff900000.vop**/vop_dump/dump

3. dump 出来的文件保存在 /data/vop_buf/，可以使用 7yuv 软件查看

4. 以上路径加粗部分 [**ff900000.vop**] 在不同平台不一样，具体路径可以和 cat /d/dri/0/summary 节点的 VOP/Video Port 对应，比如对于 RK3588 路径会变成：/sys/kernel/debug/dri/0/**video_port0**/dump

调整 DRM 打印 Log 等级抓 Log

DRM 根据不同的接口设定以下几个打印等级，可以通过 DRM 的 debug 节点决定开关哪个接口的打印，比如我要查看 ATOMIC 的打印，就你可以输入命令：

```
echo 0x10 > /sys/module/drm/parameters/debug
```

如果想看到所有的调试信息， 可以输入命令：

```
echo 0xff > /sys/module/drm/parameters/debug
```

其中 0x20 Bit5 对应的是 vsync 相关的信息，这个打印的内容非常多，如果调试的时候不关注这个信息，一般不要设置这个 bit。

其他更多的打印等级可以查看下面的定义：

```
Enable debug output, where each bit enables a debug category.
Bit 0 (0x01)  will enable CORE messages (drm core code)
Bit 1 (0x02)  will enable DRIVER messages (drm controller code)
Bit 2 (0x04)  will enable KMS messages (modesetting code)
Bit 3 (0x08)  will enable PRIME messages (prime code)
Bit 4 (0x10)  will enable ATOMIC messages (atomic code)
Bit 5 (0x20)  will enable VBL messages (vblank code)
Bit 7 (0x80)  will enable LEASE messages (leasing code)
Bit 8 (0x100) will enable DP messages (displayport code)
```

这里面有很多打印，默认是不打印到串口的，所以打开了对应的调试开关后，默认在串口终端只会看到部分调试信息，要获取完整 log 信息需要通过 `dmesg` 命令查看。

查看当前显示时钟

时钟是影响显示模块的关键因素之一，所以我们经常要获取当前系统提供的时钟是多少，可以通过以下命令获取：

1. 获取整个时钟树

   ```
   cat /sys/kernel/debug/clk/clk_summary
   ```

2. 如果只关注 VOP 的时钟

```
cat /sys/kernel/debug/clk/clk_summary | grep vop
```

对于 HDMI/DP/VGA/CVBS 等显示接口，每个分辨率对应的 DCLK 是有严格的标准要求，如果 DCLK 时钟不对可能会出现无法显示或者出现显示兼容性问题，而对于 eDP/LVDS/MIPI/RGB 等显示接口一般有一定的余量。

在 VOP2 驱动使能对应的显示接口的时候，会有如下的打印，显示接口需要的时钟频率以及从系统中获取到的时钟频率，如果二者不匹配的时候，就要特别注意。

```
rockchip-vop2: [vop2_crtc_atomic_enable] Update mode to 3840x2160p60, type: 10(if:200) for vp2 dclk:
533280000
fdd90000.vop: [drm:vop2_crtc_atomic_enable] dclk_out2 div: 2 dclk_core2 div: 2
fdd90000.vop: [drm:vop2_crtc_atomic_enable] set dclk_vop2 to 533280000, get 533279987
```

这个 log 显示的是，VP2 上整在使能一个 DP 接口(通过 type 判断)，显示 4K 的分辨率，需要的 dclk 为 533280000 HZ，但是系统分配到的是 533279987 HZ，因为 DP 对 dclk 的精度要求非常高，这种情况下，大概率屏幕是点不亮的。

强行开关显示设备

以 LVDS 为例：

关 LVDS: echo off > /sys/class/drm/card0-LVDS-1/status

开 LVDS: echo on > /sys/class/drm/card0-LVDS-1/status

查看 DRM buffer 使用情况

通过下面的命令可以知道通过 DRM GEM 申请的 buffer 使用情况：

```
cat /sys/kernel/debug/dri/0/mm_dump
```

```
/ # cat /sys/kernel/debug/dri/0/mm_dump
0x0000000000000000-0x0000000000c00000: 12582912: used
0x0000000000c00000-0x0000000100000000: 4282384384: free
total: 4294967296, used 12582912 free 4282384384
```

查看 GPIO 状态

在项目刚开始 bring up 点屏阶段，经常需要通过控制 GPIO 的状态来控制屏或者背光的电源，我们可以通过以下命令确认软件配置的 GPIO 状态是否正确：

```
cat /sys/kernel/debug/gpio
```

```
GPIOs 0-31, platform/pinctrl, gpio0:
 gpio-1   (                    |vcc_sd             ) out lo
 gpio-4   (                    |bt_default_wake_host) in  lo
 gpio-5   (                    |GPIO Key Power     ) in  hi
 gpio-9   (                    |bt_default_reset   ) out lo
 gpio-10  (                    |reset              ) out hi

GPIOs 32-63, platform/pinctrl, gpio1:
 gpio-34  (                    |int-n              ) in  hi
 gpio-45  (                    |enable             ) out hi
 gpio-46  (                    |vsel               ) out lo
 gpio-49  (                    |vsel               ) out lo
```

modetest 的使用

modetest 是集成在 libdrm 中用来测试 drm 系统的命令行工具，它可以提供如下功能：

1. Dump drm 系统的基础状态

```
[root@RK356X:/]# modetest -M rockchip
Encoders:
id      crtc    type    possible crtcs  possible clones
151     0       Virtual 0x00000007      0x00000000
153     0       TMDS    0x00000002      0x00000000
155     71      TMDS    0x00000001      0x00000000
166     0       DPI     0x00000004      0x00000000
```

```
Connectors:
id      encoder status          name            size (mm)       modes   encoders
154     0       disconnected    eDP-1           0x0             0       153
  props:
        1 EDID:
                flags: immutable blob
                blobs:
                value:
        2 DPMS:
                flags: enum
                enums: On=0 Standby=1 Suspend=2 Off=3
                value: 0
156     155     connected       HDMI-A-1        530x300         31      155
  modes:
        name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: preferred,
driver
  1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148352 flags: phsync, pvsync; type: driver
  1920x1080 50 1920 2448 2492 2640 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
  1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: driver
  720x576 50 720 732 796 864 576 581 586 625 27000 flags: nhsync, nvsync; type: driver
  720x480 60 720 736 798 858 480 489 495 525 27027 flags: nhsync, nvsync; type: driver
  props:
167     0       disconnected    HDMI-A-2        0x0             0       166
  props:

CRTCs:
id      fb      pos     size
71      0       (0,0)   (1920x1080)
   60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: userdef
  props:
        72 PLANE_MASK:
                flags: immutable bitmask
                values: Cluster0=0x1 Cluster1=0x2 Esmart0=0x4 Esmart1=0x8 Smart0=0x10 Smart1=0x20
Cluster2=0x40 Cluster3=0x80 Esmart2=0x100 Esmart3=0x200
                value: 21
        26 GAMMA_LUT:
                flags: blob
                blobs:
                value:
        27 GAMMA_LUT_SIZE:
                flags: immutable range
                values: 0 4294967295
                value: 1024
87      0       (0,0)   (0x0)
   0 0 0 0 0 0 0 0 0 0 flags: ; type:
  props:
        88 PLANE_MASK:
                flags: immutable bitmask
                values: Cluster0=0x1 Cluster1=0x2 Esmart0=0x4 Esmart1=0x8 Smart0=0x10 Smart1=0x20
Cluster2=0x40 Cluster3=0x80 Esmart2=0x100 Esmart3=0x200
                value: 34
        26 GAMMA_LUT:
                flags: blob
                blobs:
                value:
        27 GAMMA_LUT_SIZE:
                flags: immutable range
                values: 0 4294967295
                value: 1024
103     0       (0,0)   (0x0)
   0 0 0 0 0 0 0 0 0 0 flags: ; type:
  props:
        104 PLANE_MASK:
                flags: immutable bitmask
                values: Cluster0=0x1 Cluster1=0x2 Esmart0=0x4 Esmart1=0x8 Smart0=0x10 Smart1=0x20
Cluster2=0x40 Cluster3=0x80 Esmart2=0x100 Esmart3=0x200
                value: 8
        26 GAMMA_LUT:
                flags: blob
                blobs:
                value:
```

```
        27 GAMMA_LUT_SIZE:
                flags: immutable range
                values: 0 4294967295
                value: 1024

Planes:
id      crtc    fb      CRTC x,y        x,y     gamma size      possible crtcs
57      0       0       0,0             0,0     0               0x00000007
  formats: XR24 AR24 XB24 AB24 RG24 BG24 RG16 BG16
  props:
        8 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 1
        60 alpha:
                flags: range
                values: 0 65535
                value: 65535
        61 pixel blend mode:
                flags: enum
                enums: None=2 Pre-multiplied=0 Coverage=1
                value: 0
        62 zpos:
                flags: range
                values: 0 7
                value: 0
        63 NAME:
                flags: immutable bitmask
                values: Smart0-win0=0x1
                value: 1
        70 colorkey:
                flags: range
                values: 0 2164260863
                value: 0
73      0       0       0,0             0,0     0               0x00000007
  formats: XR24 AR24 XB24 AB24 RG24 BG24 RG16 BG16
  props:
        8 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 1
        76 alpha:
                flags: range
                values: 0 65535
                value: 65535
        77 pixel blend mode:
                flags: enum
                enums: None=2 Pre-multiplied=0 Coverage=1
                value: 0
        78 zpos:
                flags: range
                values: 0 7
                value: 1
        79 NAME:
                flags: immutable bitmask
                values: Smart1-win0=0x2
                value: 2
        86 colorkey:
                flags: range
                values: 0 2164260863
                value: 0
89      0       0       0,0             0,0     0               0x00000007
  formats: XR24 AR24 XB24 AB24 RG24 BG24 RG16 BG16 NV12 NV16 NV24 NA12 NA16 NA24 YVYU VYUY
  props:
        8 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 1
        92 alpha:
                flags: range
                values: 0 65535
                value: 65535
```

```
        93 pixel blend mode:
                flags: enum
                enums: None=2 Pre-multiplied=0 Coverage=1
                value: 0
        94 zpos:
                flags: range
                values: 0 7
                value: 2
        95 NAME:
                flags: immutable bitmask
                values: Esmart1-win0=0x4
                value: 4
105     0       0       0,0             0,0     0               0x00000007
  formats: XR24 AR24 XB24 AB24 RG24 BG24 RG16 BG16 NV12 NV16 NV24 NA12 NA16 NA24 YVYU VYUY
  props:
        8 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 0
        108 alpha:
                flags: range
                values: 0 65535
                value: 65535
        109 pixel blend mode:
                flags: enum
                enums: None=2 Pre-multiplied=0 Coverage=1
                value: 0
        110 zpos:
                flags: range
                values: 0 7
                value: 3
        111 NAME:
                flags: immutable bitmask
                values: Esmart0-win0=0x8
                value: 8
```

第一部分的 Encoders 输出和第二部分的 Connectors 对应，从 Dump 输出我们可以看到：

Connectore eDP-1 的 id 为 154，它对应的 Encoder id 为 153， 且它处于 `disconnected` 状态， 说明底层驱动未检测到该 eDP 设备的连接，Encoder 153 的 `possible crtcs` 的值为 2 即 BIT(1)， 这是一个按位与的掩码，说明该 eDP 只能和 VP1 连接。

Connector HDMI-A-1 的 id 为 156，它对应的 Encoder id 为 155，且它处于 `connected` 的状态，说明底层驱动已经检测到了该 HDMI 接口上已经有显示设备连接，modes 是驱动上报的对应显示设备支持的分辨率。

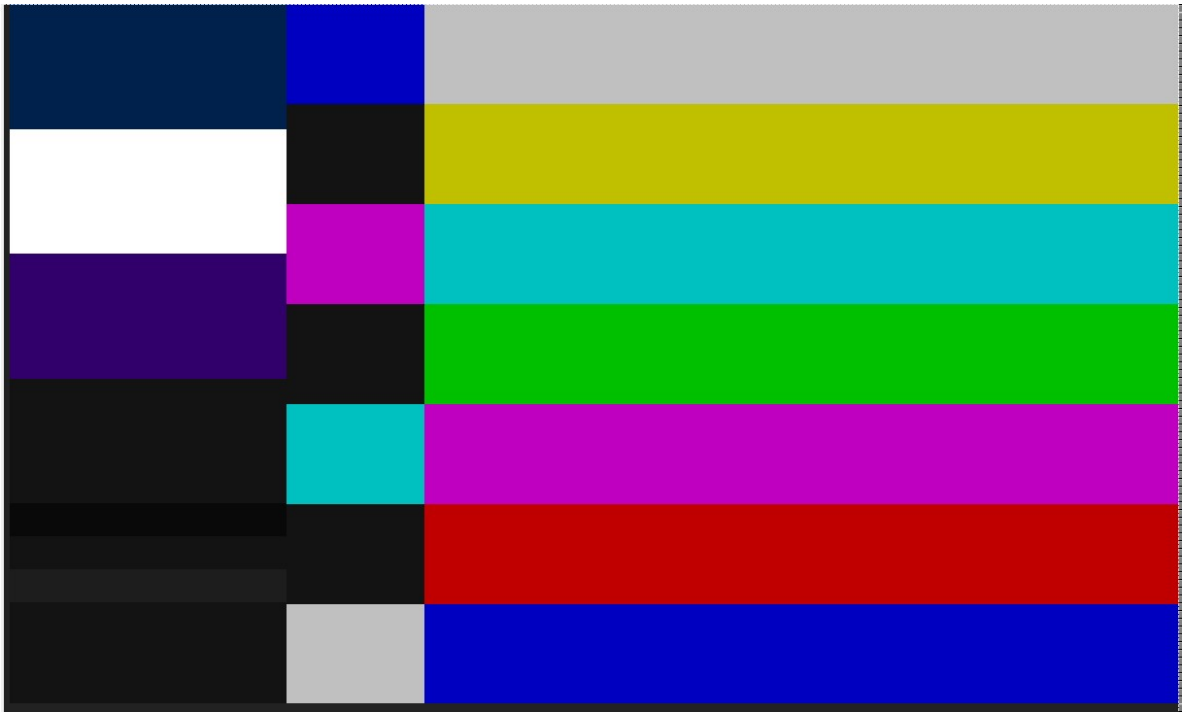
CRTC 对应 VOP 2.0 中的 Video Port 或者 VOP 1.0 中的 vop。

Planes 对应图层，列出的信息包含该图层可以在哪几个 VP 之间切换（`possible crtcs`）以及所支持的格式(`formats`)。

2. 在对应的显示接口上输出各种颜色的彩条，可以利用该功能对 Drm 驱动进行简单的验证。

```
modetest -M rockchip -s 156@71:1920x1080
```

通过上述命令，可以在 HDMI-A-1 上显示类似下边的 smpte 彩条，其中 156 是 HDMI-A-1 的 id，71 是 VP0 的 id。



3. 需要临时手动修改 drm 一些属性的值

更多的使用方式，可以参考以下使用说明：

```
[root@RK3588:/]# modetest -h
usage: modetest [-acDdefMPpsCvrw]

 Query options:

        -c      list connectors
        -e      list encoders
        -f      list framebuffers
        -p      list CRTCs and planes (pipes)

 Test options:

        -P <plane_id>@<crtc_id>:<w>x<h>[+<x>+<y>][*<scale>][@<format>]  set a plane
        -s <connector_id>[,<connector_id>][@<crtc_id>]:[#<mode index>]<mode>[-<vrefresh>][@<format>]    set a mode
        -C      test hw cursor
        -v      test vsynced page flipping
        -r      set the preferred mode for all connectors
        -w <obj_id>:<prop_name>:<value> set property
        -a      use atomic API
        -F pattern1,pattern2    specify fill patterns

 Generic options:

        -d      drop master after mode set
        -M module       use the given driver
        -D device       use the given device

        Default is to dump all info.
```

这里需要说明的是，modetest 是通过 id 来指定 plane，crtc，connector 等 Drm 资源的，这些 id 可以通过 modetest 查询，也可以通过如下命令来查询：

```
cat /sys/kernel/debug/dri/0/state
```

xrandr 的使用

xrandr 是使用 Xserver 的图形系统上的一个显示配置工具，它可以用来 dump 系统的显示信息，也可以用来配置系统的显示输出。

```
root@linaro-alip:/# xrandr
Screen 0: minimum 320 x 200, current 1920 x 2048, maximum 8192 x 8192
eDP-1 connected primary 1536x2048+0+0 (normal left inverted right x axis y axis) 0mm x 0mm
   1536x2048     59.99*+
HDMI-1 connected 1920x1080+0+0 (normal left inverted right x axis y axis) 708mm x 398mm
   1920x1080     60.00*+  50.00    59.94    30.00    24.00    29.97    23.98
   1920x1080i    60.00    50.00    59.94
   1280x1024     60.02
   1440x900      59.90
   1360x768      60.02
   1280x720      60.00    50.00    59.94
   1024x768      60.00
```

```
   800x600       60.32
   720x576       50.00
   720x576i      50.00
   720x480       60.00    59.94
   720x480i      60.00    59.94
   640x480       60.00    59.94
DP-1 disconnected (normal left inverted right x axis y axis)
```

该工具是 Xserver 环境下的通用工具，详细的使用介绍可参考 Xrandr

显示进程的暂停、启动

有时候为了调试方便，希望应用停在某一个固定的场景，我们可以使用下面的命令暂停和恢复显示进程。

- Android Surfaceflinger 进程：

1. 暂停进程

```
kill -STOP `pgrep surfaceflinger`
```

2. 恢复进程

```
kill -CONT `pgrep surfaceflinger`
```

- Linux weston 进程：

1. 暂停进程

```
Kill -STOP  `pgrep weston`
```

2. 恢复进程

```
kill -CONT `pgrep weston`
```

- Linux Xserver 进行

1. 暂停进程

```
Kill -STOP  `pgrep Xorg`
```

2. 恢复进程

```
kill -CONT `pgrep Xorg`
```

在日常的开发过程中，当我们希望临时通过 io 命令修改 VOP 的一些寄存器配置来查看某种显示效果的时候，特别注意提前使用类似的命令，停掉 usespace 中对应的显示主进程，否则显示进程的更新会随时覆盖本地的寄存器修改。

获取 EDID 信息

以 HDMI 为例：

```
cat /sys/class/drm/card0-HDMI-A-1/edid > /data/edid.bin
```

查看 HDMI 状态

```
cat /sys/kernel/debug/dw-hdmi/status
```

```
/ # cat /sys/kernel/debug/dw-hdmi/status
PHY: enabled                    Mode: HDMI
Pixel Clk: 148500000Hz          TMDS Clk: 148500000Hz
Color Format: RGB               Color Depth: 8 bit
Colorimetry: ITU.BT709          EOTF: Off
```

FAQ

VOP POST_BUF_EMPTY

可能会出现类似以下 log：

```
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
```

可能原因有：

1. 带宽不够

   如果系统带宽不够会导致 VOP 不能及时取到数据，从而报 post buf empty，可以做如下尝试：

   (1) 尝试将 ddr 固定最高频率;

   (2) 将屏的消隐期加长，提高一行的取数时间;

2. iommu 出错

   确认是否如下图所示的 iommu pagefault 错误，如果有，请先尝试更新到最新代码测试，如果最新代码还存在该问题，请提交 remine 系统，并附上相关 log;

```
rockchip-vop ff8f0000.vop: [drm:vop_isr] *ERROR* POST_BUF_EMPTY irq err
rk_iommu ff8f3f00.iommu: Page fault at 0x00000000f0000400 of type read
rk_iommu ff8f3f00.iommu: iova = 0x00000000f0000400: dte_index: 0x3c0 pte_index: 0x0
rk_iommu ff8f3f00.iommu: mmu_dte_addr: 0x00000000f1692000 dte@0x00000000f1692f00: 0x
l: 0 page@0x0000000000000000 flags: 0x0

0x00000000: 00000000 03058896 20805800 0003d000
0x00000010: 0000000f 0800780c 00000000 00711c08
0x00000020: ed000000 00000000 00000000 00000000
0x00000030: 3a001085 00400000 00000000 01400147
0x00000040: 00082000 00000000 0101028d 0101028d
```

3. Logic 电压太低

   Logic 电压太低会导致 VOP 异常，可以尝试提高 100mv 测试；

4. AFBDC/IFBDC 对齐要求

   对于 PX30/RK3326、RK3368、RK3399、RK356X、RK3588 平台如果屏的分辨率非 16pixel 对齐可以尝试关闭 afbdc/ifbdc 功能，修改方法参考文档《FAQ-DRM-HWC》 1.3.1 章节。

显示效果调节

VOP 内部的 BCSH 模块支持 亮度、对比度、饱和度、色度的调节，可以通过 modetest 配置 connector 下的对应属性调节，对于 android 系统，有实现了基于安卓系统的属性配置默认值是 50，N 每次+1。

android 9.0 之前使用命令：

```
setprop persist.sys.brightness.main val
setprop persist.sys.contrast.main val
setprop persist.sys.saturation.main val
setprop persist.sys.hue.main val
setprop sys.display.timeline N+1
```

android 9.0 及之后使用命令：

```
setprop persist.vendor.brightness.main val
setprop persist.vendor.contrast.main val
setprop persist.vendor.saturation.main val
setprop persist.vendor.hue.main val
setprop vendor.display.timeline N+1
```

从 Android 11 开始 SDK 有集成了整套显示效果调节工具和文档，可以参考 Android SDK 中的以下文档说明：

RKDocs/common/display/Rockchip_Introduction_DisplayAdjust_APK_CN.pdf

屏无法点亮/休眠唤醒显示异常/不显示问题

按以下几个方面做进一步确认：

1. 确认是否有背光；
2. 确认屏的相关电源及复位控制是否正常；
3. 确认上下电时序是否满足屏的 spec 要求；

RK3308 如何打开显示功能

RK3308 默认不支持显示功能，如果需要开显示，需要做以下配置：

1. U-Boot 修改

```
--- a/configs/evb-rk3308_defconfig
+++ b/configs/evb-rk3308_defconfig
@@ -4,7 +4,6 @@ CONFIG_SYS_MALLOC_F_LEN=0x2000
CONFIG_ROCKCHIP_RK3308=y
CONFIG_ROCKCHIP_SPL_RESERVE_IRAM=0x0
CONFIG_RKIMG_BOOTLOADER=y
-# CONFIG_USING_KERNEL_DTB is not set
CONFIG_TARGET_EVB_RK3308=y
CONFIG_DEFAULT_DEVICE_TREE="rk3308-evb"
CONFIG_DEBUG_UART=y
@@ -55,6 +54,11 @@ CONFIG_USB_GADGET_DOWNLOAD=y
CONFIG_G_DNL_MANUFACTURER="Rockchip"
CONFIG_G_DNL_VENDOR_NUM=0x2207
CONFIG_G_DNL_PRODUCT_NUM=0x330d
+CONFIG_DM_VIDEO=y
+CONFIG_DISPLAY=y
+CONFIG_DRM_ROCKCHIP=y
+CONFIG_DRM_ROCKCHIP_RGB=y
+CONFIG_LCD=y
CONFIG_USE_TINY_PRINTF=y
CONFIG_SPL_TINY_MEMSET=y
CONFIG_ERRNO_STR=y
```

2. kernel 修改

   RK3308 VOP 不支持 IOMMU，所以分配内存需要从预留的 cma buffer 分配，默认 CMA_SIZE 为 16M，如果出现分配内存失败，可以参考如下方法修改 CMA_SIZE：

```
--- a/arch/arm64/configs/rk3308_linux_defconfig
+++ b/arch/arm64/configs/rk3308_linux_defconfig
@@ -44,6 +44,7 @@ CONFIG_HZ_1000=y
 # CONFIG_COMPACTION is not set
 # CONFIG_BOUNCE is not set
 CONFIG_DEFAULT_MMAP_MIN_ADDR=32768
+CONFIG_CMA=y
 # CONFIG_UNMAP_KERNEL_AT_EL0 is not set
 # CONFIG_ARM64_HW_AFDBM is not set
 # CONFIG_ARM64_PAN is not set
@@ -89,6 +90,8 @@ CONFIG_RFKILL=y
 CONFIG_DEVTMPFS=y
 CONFIG_DEVTMPFS_MOUNT=y
 # CONFIG_ALLOW_DEV_COREDUMP is not set
+CONFIG_DMA_CMA=y
+CONFIG_CMA_SIZE_MBYTES=32
 # CONFIG_BLK_DEV is not set
 CONFIG_NETDEVICES=y
 # CONFIG_NET_CORE is not set
```

关闭 iommu 的方法

关闭 vop iommu 后通过 DRM 申请的内存会从 CMA 内存分配，系统默认 CMA 内存大小 16M，需要根据场景需求调整到对应的大小，否则会出现分配内存失败。

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3588s.dtsi b/ar
index 915034fc3d0a..26625ff51219 100644
--- a/arch/arm64/boot/dts/rockchip/rk3588s.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3588s.dtsi
@@ -3423,7 +3423,7 @@ vop: vop@fdd90000 {
                                     "dclk_vp1",
                                     "dclk_vp2",
                                     "dclk_vp3";
-               iommus = <&vop_mmu>;
+               //iommus = <&vop_mmu>;
                power-domains = <&power RK3588_PD_VOP>;
                rockchip,grf = <&sys_grf>;
                rockchip,vop-grf = <&vop_grf>;
```

各种接口应用参考文档

《Rockchip_DRM_Panel_Porting_Guide.pdf》

《RK3588_MIPI_DSI2_Developer_Guide_CN.pdf》。

RGB/MCU 屏帧率计算问题

已知：

```
htotal = hactive + hback-porch + hfront-porch + hsync-len
vtotal = vactive + vback-porch + vfront-porch + vsync-len
```

N 是每一个像素需要 N 个 cycly 发送完：

bus_format	一个 Pixel 要 N 个 Cycle 发送
MEDIA_BUS_FMT_RGB888_1X24 MEDIA_BUS_FMT_RGB666_1X18 MEDIA_BUS_FMT_RGB565_1X16	1
MEDIA_BUS_FMT_RGB888_3X8	3
MEDIA_BUS_FMT_RGB888_DUMMY_4X8	4

根据以上信息，可以计算帧率：

接口类型	帧率计算
RGB	fps = dclk / (htotal x vtotal x N)
MCU	fps = dclk / (htotal x vtotal x （mcu-pix-total + 1) x N)

如何编写第三方转换芯片驱动

有些第三方转换芯片需要软件驱动的，这个时候我们需要借助 DRM 框架 bridge 接口向框架注册 connector，比如 RGB2HDMI 的 SII902X 为例，此时 rockchip_rgb.c 充当 encoder 角色，sii902x.c 充当 connector 角色，具体可以参考 SII902X 驱动的实现：

```
drivers/gpu/drm/bridge/sii902x.c
```

RK3588 DSC 支持几个 slice，slice width 最大支持多少

RK3588 有 2 个 DSC，HDMI0 和 DSI0 使用 DSC0(DSC_8K)，最大支持 8 slice， 最大的 slice width 是 960； HDMI1 和 DSI1 使用 DSC1(DSC_4K)，最大支持 2 slice，最大的 slice width 是 2048。

超过 4kP60 对 aclk 的要求

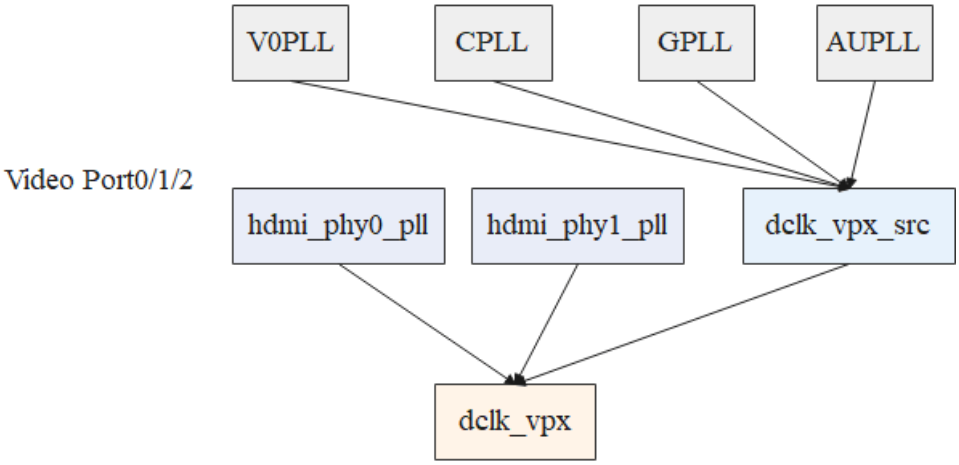
对于类似 RK3588 这种支持的分辨率超过 4KP60 的【如 4KP120, 8KP30, 8KP60 等分辨率】，默认的 500M aclk 无法满足这些分辨率的需求，所以需要在 DTS 中设置 aclk 频率为 800M，否则会出现 VOP 性能不够导致的显示横条纹问题，修改方法如下：

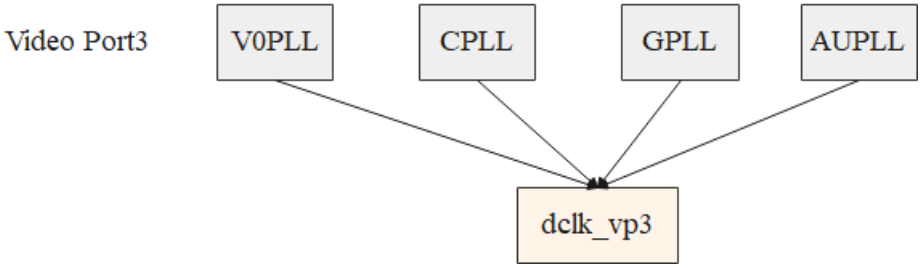
```
&vop {
    assigned-clocks = <&cru ACLK_VOP>;
    assigned-clock-rates = <800000000>;
};
```

RK3588 VOP DCLK 分配策略

RK3588 DCLK 概述

RK3588 有 4 个 Video Port, 每个 Video Port 的 dclk tree 如下图所示：





对于 dclk_vp0/1/2， 可以指定 hdmi_phy0_pll, hdmi_phy1_pll，dclk_vpx_src0/1/2 中的一个作为其 parent clock, 对 dclk_vpx_src0/1/2，可以指定

V0PLL, CPLL, GPLL, AUPLL 中的一个作为其 parent clock。

对于 dclk_vp3，可以指定 V0PLL, CPLL, GPLL, AUPLL 中的一个作为其 parent clock。

V0PLL, CPLL, GPLL, AUPLL 为系统 CRU 上的 PLL，hdmi_phy0_pll 和 hdmi_phy1_pll 为 HDMI PHY 上的 PLL

VOPLL 的特点如下：

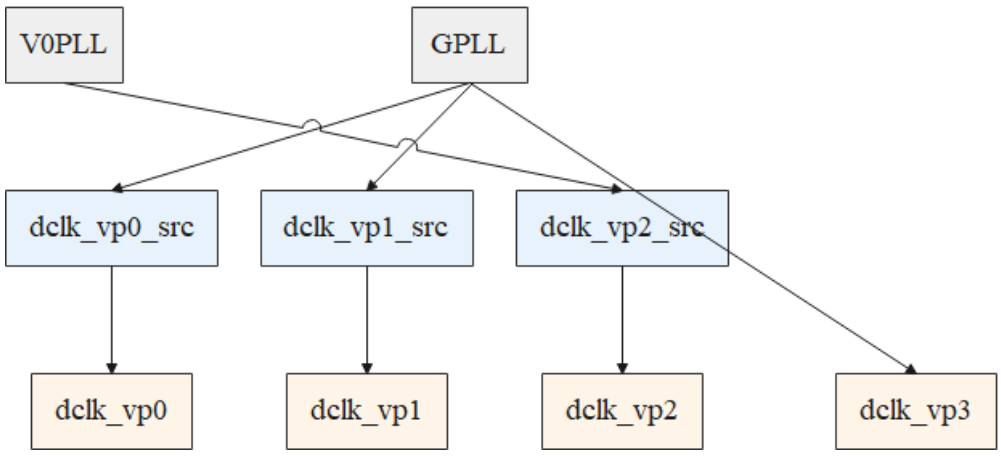
1. VOP 独占 PLL
2. 支持任意频率
3. dts 中默认与 dclk_vp2 绑定，代码如：

```
vp2: port@2 {
    assigned-clocks = <&cru DCLK_VOP2_SRC>;
    assigned-clock-parents = <&cru PLL_V0PLL>;
}
```

CPLL, GPLL, AUPLL 的特点如下：

1. 与其它 IP 模块共享 PLL
2. 不支持任意分频，输出频率为 PLL 频率的整数分频
3. dclk_vp0/1/3 默认与 GPLL 绑定

各个 Video Port 的默认 parent clock 的绑定关系如下：



hdmi_phy0_pll, hdmi_phy1_pll 的特点如下：

1. 支持任意分频
2. HDMI 不工作时 VOP 独占 PLL，HDMI 工作时与 HDMI PHY 共享 PLL
3. HDMI 和 EDP 共用 PHY，EDP 工作时，PHY 上的 PLL 无法被 VOP 使用

hdmi_phy1_pll, hdmi_phy1_pll 的第 2, 3 个特点导致使用时有一些限制，举例说明如下：

1. 当 HDMI0 工作时，且 Video Port0 绑定到 HDMI0，各 Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0_DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	DCLK <= 600MHz, 可用; DCLK > 600MHz, 不可用	不可用	不可用
hdmi_phy1_pll	可用	可用	可用

2. 当 HDMI1 工作时，且 Video Port0 绑定到 HDMI1，各 Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0 DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	可用	可用	可用
hdmi_phy1_pll	DCLK <= 600MHz, 可用 DCLK > 600MHz, 不可用	不可用	不可用

3. 当 HDMI0 和 HDMI1 同时工作时，且都绑定到 Video Port0 时， Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0_DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	DCLK <= 600MHz, 可用; DCLK > 600MHz, 不可用	不可用	不可用
hdmi_phy1_pll	DCLK <= 600MHz, 可用; DCLK > 600MHz, 不可用	不可用	不可用

4. 当 EDP0 工作时，且 Video Port0 绑定到 EDP0，各 Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0_DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	不可用	不可用	不可用
hdmi_phy1_pll	可用	可用	可用

5. 当 EDP1 工作时，且 Video Port0 绑定到 EDP1，各 Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0 DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	可用	可用	可用
hdmi_phy1_pll	不可用	不可用	不可用

6. 当 EDP0 和 EDP1 同时工作时，且都绑定到 Video Port0 时， Video Port 可以使用 hdmi_phy0_pll 和 hdmi_phy1_pll 的场景(绑定到 Video Port1/2 时类似)

HDMI PHY PLL	VP0_DCLK	VP1_DCLK	VP2_DCLK
hdmi_phy0_pll	不可用	不可用	不可用
hdmi_phy1_pll	不可用	不可用	不可用

由于 CPLL，GPLL，AUPLL 作为 Video Port dclk 源时，无法实现任意分频。当需要准确分频时，且满足使用的限制条件时，可以使用 hdmi_phy0_pll 或 hdmi_phy1_pll 作为 Video Port dclk 的 parent clock。分配 Video Port dclk parent 有两种方式，分别是静态分配和动态分配。

RK3588 VOP DCLK 静态分配策略

静态分配即通过在 dtsi 中绑定 Video Port dclk parent 的方式, 在 dtsi 中配置好绑定关系后，就不再发生变更。

```
&hdptxphy_hdmi0 {
    status = "okay";
};

&hdptxphy_hdmi1 {
    status = "okay";
};

&hdptxphy_hdmi_clk0 {
    status = "okay";
};

&hdptxphy_hdmi_clk1 {
    status = "okay";
};

&vp0 {
    assigned-clocks = <&cru DCLK_VOP0>;
    assigned-clock-parents = <&hdptxphy_hdmi_clk0>;
};

&vp1 {
    assigned-clocks = <&cru DCLK_VOP1>;
    assigned-clock-parents = <&hdptxphy_hdmi_clk1>;
};
```

如上，即绑定 dclk_vp0 的 parent clock 为 hdmi_phy0_pll, dclk_vp1 的 parent clock 为 hdmi_phy1_pll 。

通过 clock tree，可以确认配置是否生效，如下：

```
rk3588_s:/ # cat /sys/kernel/debug/clk/clk_summary | grep "hdmiphy" -A 1
       clk_hdmiphy_pixel1             0        1        0           0          0     0  50000
          dclk_vop1                   0        3        0           0          0     0  50000
       clk_hdmiphy_pixel0             1        1        0   148500000          0     0  50000
          dclk_vop0                   2        5        0   148500000          0     0  50000
```

如上，parent clock 绑定生效，并且当前 dclk_vp0 设置的 clock rate 为 148.5MHz，这里的 clock rate 与 log 打印中获取的 clock rate 是一样的。

如下 log 中的最后一行`set dclk_vop0 to 148500000, get 148500000`即设置 148.5MHz, 实际输出的也是 148.5MHz, 与从 clk tree 上获取的相同。

```
[  270.974623][  T381] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] Update mode to
1920x1080p60, type: 11(if:800) for vp0 dclk: 148500000
[  270.974837][  T381] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] dclk_out0 div: 0
dclk_core0 div: 2
[  270.974891][  T381] rockchip-hdptx-phy-hdmi fed60000.hdmiphy: hdptx_ropll_cmn_config bus_width:16a8c8
rate:1485000
[  270.975264][  T381] rockchip-hdptx-phy-hdmi fed60000.hdmiphy: hdptx phy pll locked!
[  270.975304][  T381] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] set dclk_vop0 to
148500000, get 148500000
```

结合之前的 hdmi_phy0_pll 和 hdmi_phy1_pll 的使用限制描述，静态分配时建议如下：

1. 连接到 HDMI 的 Video Port 的 parent clock 优先指定为 hdmi_phy0_pll 或 hdmi_phy1_pll；
2. HDMI 不工作时，HDMI PHY 上的 PLL 可以任意配置给 dclk_vp0/1/2 作为 parent clock。

RK3588 VOP DCLK 动态分配策略

动态绑定的方式如下：

```
&hdptxphy_hdmi0 {
    status = "okay";
};

&hdptxphy_hdmi1 {
    status = "okay";
};

&hdptxphy_hdmi_clk0 {
```

```
    status = "okay";
};

&hdptxphy_hdmi_clk1 {
    status = "okay";
};

&display_subsystem {
    clocks = <&hdptxphy_hdmi_clk0>, <&hdptxphy_hdmi_clk1>;
    clock-names = "hdmi0_phy_pll", "hdmi1_phy_pll";
};
```

由于可任意分频的 PLL 源有限，并且 HDMI PHY 上 的 PLL 使用还有一定的限制，因此动态分配的策略比较复杂，同时，动态分配的策略对业务场景有一定的限制。

每次使能一个 Video Port 时，都会按照如下的策略进行动态的 Video Port dclk 的绑定：

1. HDMI0 和 HDMI1 (可能也包含 DP 或 MIPI 挂在同一个 Video Port 下) 接到同一个 Video Port(0/1/2)并且 dclk 在 600MHz 以下:

   1.1 hdmi_phy0_pll 和 hdmi_phy1_pll 均未被其他 Video Port 使用时，使用 hdmi_phy0_pll 为当前 Video Port dclk 的 parent clock , 并且 hdmi_phy0_pll 和 hdmi_phy1_pll 不可被其他的 Video Port 使用;

   1.2 hdmi_phy0_pll 或 hdmi_phy0_pll 被其他 Video Port 使用时，则报错；

2. HDMI0 (可能也包含 DP 或 MIPI 挂在同一个 Video Port 下) 接到一个 Video Port(0/1/2)，下面称 VP_A，并且 dclk 在 600MHz 以下:

   2.1 hdmi_phy0_pll 和 hdmi_phy1_pll 均存在时，且 hdmi_phy0_pll 未被其他 Video Port 使用时，VP_A 使用 hdmi_phy0_pll ，同时 hdmi_phy0_pll 不可被其他的 Video Port 使用;

   2.2 hdmi_phy0_pll 和 hdmi_phy1_pll 均存在时，且 hdmi_phy0_pll 正在被其他的 Video Port(VP_B) 使用时, hdmi_phy1_pll 空闲时，VP_B 要把 hdmi_phy0_pll 的 pll 让出来，去使用 hdmi_phy1_pll ，VP_A 使用 hdmi_phy0_pll ;

   2.3 hdmi_phy0_pll 和 hdmi_phy1_pll 均存在且均被其他的 Video Port 使用时, 报错;

   2.4 只有配置 hdmi_phy0_pll 且 未被其他 Video Port 使用时， VP_A 使用 hdmi_phy0_pll;

   2.5 只有配置 hdmi_phy0_pll 且 被其他 Video Port 使用时，报错;

3. HDMI1 (可能也包含 DP 或 MIPI 挂在同一个 Video Port 下) 接到一个 Video Port(0/1/2) 并且 dclk 在 600MHz 以下时，策略和 2 类似，不再详细描述;

4. 只有 DP接到一个 Video Port(0/1/2) 时:

   4.1 Video Port 为 Video Port2 时, 使用默认配置的 dclk parent clock;

   4.2 hdmi_phy0_pll 或 hdmi_phy1_pll 未被其他 Video Port 使用时，优先使用 hdmi_phy0_pll 或 hdmi phy1_pll，否则使用默认配置的 dclk parent clock;

如果要求的 dclk 大于 4k@60Hz, 使用默认的 pll。

根据动态配置策略，为尽可能多的利用 hdmi_phy0_pll 和 hdmi_phy1_pll , 实际使用时有如下建议：

1. HDMI0 和 HDMI1 尽量不要挂在 Video Port2 上;
2. HDMI0 和 HDMI1 尽量不要挂在同一个Video Port上。

对于一些应用场景，可以给出如下的建议 Video Port 和 显示接口的配置：

2 个 HDMI 接口和 1 个 DP 接口输出时：

Video Port	**接口配置1**	**接口配置2**
Video Port0	HDMI0	HDMI1
Video Port1	HDMI1	HDMI0
Video Port2	DP0/DP1	DP0/DP1

2 个 DP 接口和 1 个 HDMI 接口输出时:

Video Port	**接口配置1**	**搭配方案2**	**接口配置1**	**接口配置2**
Video Port0	HDMI0/1	HDMI0/1	DP0	DP1
Video Port1	DP0	DP1	HDMI0/1	HDMI0/1
Video Port2	DP1	DP0	DP1	DP0

更多的应用场景，需要参考动态分配策略进行合理的配置。

动态分配策略可能会出现 HDMI PHY PLL 都被占用的情况，这个时候 log 中会打印 HDMI PHY PLL 的占用信息：

```
[  220.692900][  T383] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] Update mode to
1920x1080p60, type: 11(if:1000) for vp1 dclk: 148500000
[  220.693048][  T383] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] dclk_out1 div: 0
dclk_core1 div: 2
[  220.693094][  T383] [drm:vop2_clk_set_parent_extend.isra.61] *ERROR* hdmi1 phy pll is used by vp0:vp2
```

如上，即 hdmi_phy0_pll 已经被 Video Port0 占用，hdmi_phy1_pll 已经被 Video Port2 占用。

当只配置一个 HDMI PHY PLL 时，例如

```
&display_subsystem {
        clocks = <&hdptxphy_hdmi_clk0>;
        clock-names = "hdmi0_phy_pll";
};
```

当出现如下 log 时，即说明 hdmi_phy0_pll 已经被 Video Port2 占用。

```
[   35.059306][  T386] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] Update mode to
1920x1080p60, type: 11(if:800) for vp0 dclk: 148500000
[   35.059416][  T386] rockchip-vop2 fdd90000.vop: [drm:vop2_crtc_atomic_enable] dclk_out0 div: 0
dclk_core0 div: 2
[   35.059447][  T386] [drm:vop2_clk_set_parent_extend.isra.61] *ERROR* hdmi0: phy pll is used by vp2
```

如何更高效的定位问题

在日常开发中，如果发现了 Bug 或者 issue，科学的分析和沟通有助于问题的更快解决和收敛。对于显示相关的问题，我们建议发现问题后，开发者能做如下确认。

1. 看内核 log，或者应用层 log，是否能看到明显的异常提示
2. 如果能看到内核异常的 log 提示，根据 log 去搜索内核驱动，找到相关代码，看是根据什么判断逻辑输出这些异常 log 的。
3. 确认问题是概率的，还是必现的。
4. 如果是 HDMI、DP 这些支持多分辨率的设备，确认异常是在某个特定分辨率才会出现，还是所有分辨率都会出现，确认问题是特定显示器才会出现，还是所有显示器都会出现。
5. 如果是 HDMI，DP 兼容性相关的问题，请在控制变量的前提下找第三方设备做对比，看第三方设备在同样条件下是什么表现。
6. 如果是播放视频场景下出现的异常，确认问题是特定视频，APP 才会出现，还是任何视频，APP 都会出现。
7. 如果找不到任何方向，请确认这个异常是一直都存在还是更新特定代码或者版本后才存在，如果是后者，用二分法、控制变量法，找到更新哪个版本或者模块后，异常才出现。

参考文档

1. Rockchip_drm_integration_helper-zh.pdf
2. Rockchip_DRM_Panel_Porting_Guide.pdf
3. Rockchip_Developer_Guide_HDMI_CN.pdf
4. RK3588_MIPI_DSI2_Developer_Guide_CN.pdf
5. Rockchip_BT656_TX_AND_BT1120_TX_Developer_Guide_CN.pdf
6. Rockchip RK3568 Datasheet.pdf
7. Rockchip rk fb development guide.pdf
8. Wikipedia for Direct_Rendering_Manager
9. Linux DRM Developer's Guide