VOP2 Plane Assign

文件标识：RK-KF-YF-201

发布版本：V1.1.1

日期：2021-03-11

文件密级：□绝密 □秘密 ■内部资料 □公开

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

概述

本文的主要用来记录 VOP2 的图层切换和分配限制及其方案

读者对象

本文档（本指南）主要适用于以下工程师：

显示系统开发工程师

修订记录

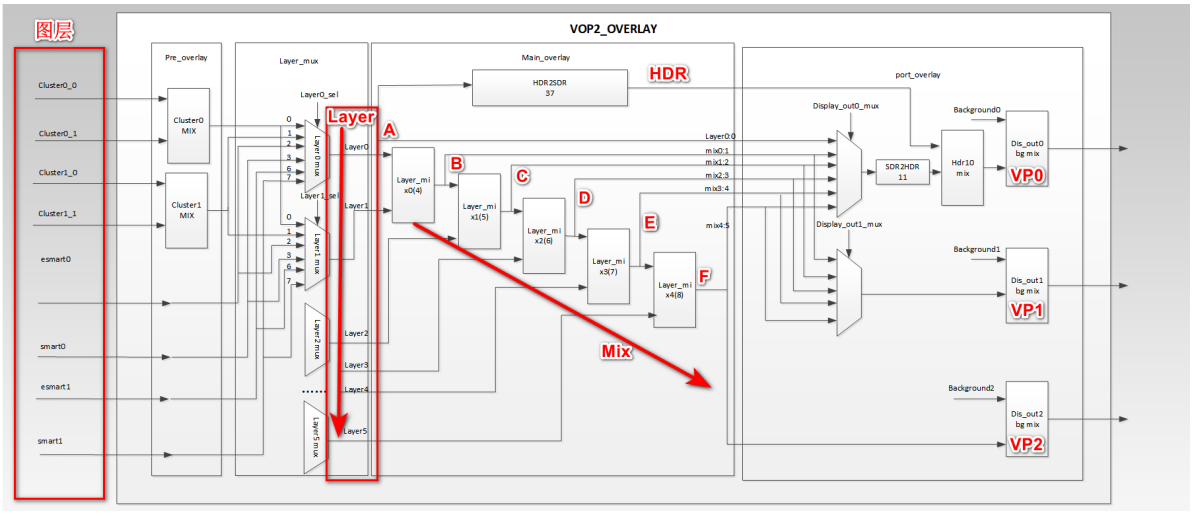
版本号	作者	修改日期	修改说明
V1.0.0	闫孝军	2021-04-06	初始版本
V1.1.0	黄家钗	2021-04-28	加入三屏显示调试问题记录和当前显示策略

目录

VOP2 Plane Assign

1. VOP2 图层和显示端口互联框架
2. 图层、Layer、VP 映射寄存器控制
3. 三屏显示调试遇到的问题
4. 当前显示策略

1. VOP2 图层和显示端口互联框架



VOP2 的图层通过上图 Layer-》MIX 的逐级映射最终分配到显示端口(VP0/1/2)上。

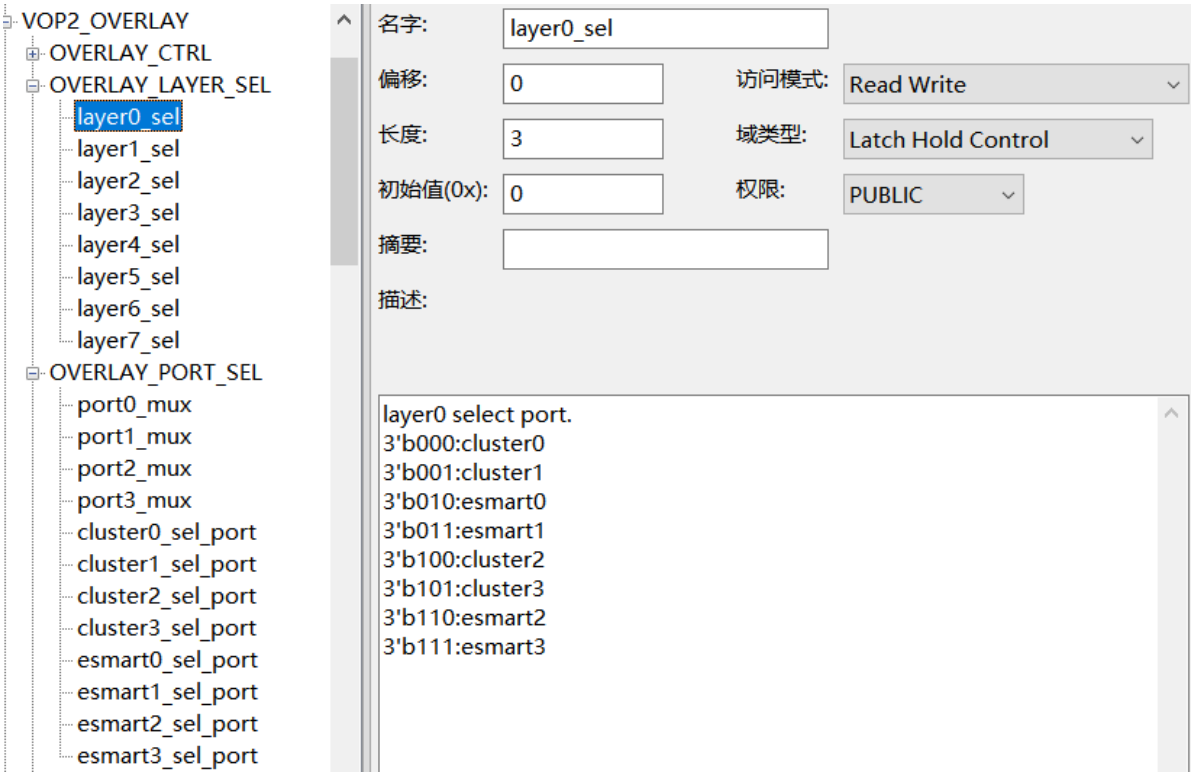
对于同一给定时刻，六个图层被三个 VP 排他性的瓜分，即每个图层同一时刻只能分配到三个 VP 中的一个，不能出现某个图层同时被多个 VP 共享的情况。

VP0、VP1，VP2 按照从 Layer0——》Layer5，Mix0~Mix5 的顺序依次分配到属于自己的图层：比如给 VP0 分配 3 个图层，VP1 分配 2 个图层，VP2 分配一个图层，则 Layer0/1/2 合成后从 **C** 路径给 VP0，Layer3/4 合成后从 **E** 路径给 VP1，剩下的图层合成后从 **F** 路径给 VP2。这些图层的 Alpha 信息需要软件配置到对应的 Mix 上。

排在前面的任何一个 VP 上分配的图层数发送变化，会自动影响后面的 VP 在 Mix 通路上的位置：比如下一帧分配给 VP0 的图层从 3 个减少为 2 个，则图层(Layer0/1)合成后由 `C` 路径变为 `B` 路径给 VP0，而且分配给 VP1 的图层自动由 Layer3/4 切换成 Layer2/3 ，并且从 **E** 路径变成 **D** 路径给 VP1， 依次类推。这些图层、Layer、Mix 对应关系变化后，每个图层的 Alpha 信息需要重新设置到变化后对应的 Mix 上。

2. 图层、Layer、VP 映射寄存器控制

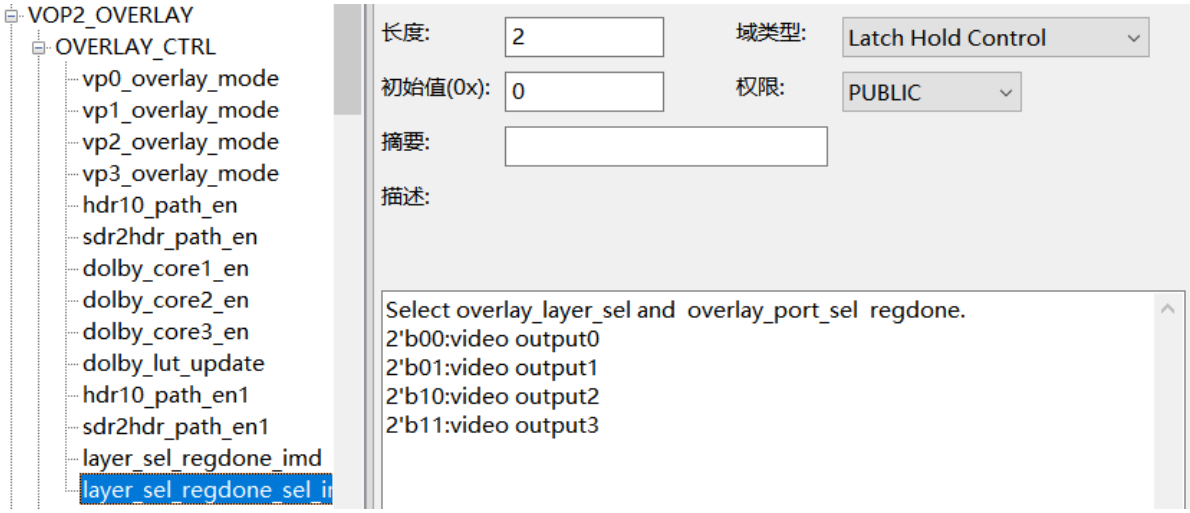
图层和 Layer 的映射关系通过 **OVERLAY_LAYER_SEL** 寄存器设置，图层和 Layer 需要排他性的一一映射，即同一时刻，一个图层只能映射到唯一的一个 Layer 上，并且按照 Layer0、Layer1、Layer2、Layer3、Layer4、Layer5 的顺序从底层到顶层做 zorder 排序进行 alpha 合成。



每个 VP 分配的图层数目通过 **OVERLAY_PORT_SEL** 寄存器中对应的 port_mux 寄存器设置，前一级图层数目的增减会影响到后面的 VP。

Bit	Attr	Reset Value	Description
7:4	RW	0x4	port1_mux port1 output mux is selected according to the layer number. Note that port1_mux >= port0_mux. 4'b0001: 2 - port0_layer_number 4'b0010: 3 - port0_layer_number 4'b0011: 4 - port0_layer_number 4'b0100: 5 - port0_layer_number 4'b0101: 6 - port0_layer_number 4'b1000: 0 layers
3:0	RW	0x3	port0_mux port0 output mux is selected according to the layer number . 4'b0000: 1 layer 4'b0001: 2 layers 4'b0010: 3 layers 4'b0011: 4 layers 4'b0100: 5 layers 4'b0101: 6 layers 4'b1000: 0 layers

OVERLAY_LAYER_SEL 和 **OVERLAY_PORT_SEL** 寄存器的生效时机是可选的：立即生效或者根据某个 VP 的 VSYNC 生效。



立即生效会存在的问题：

1. 某一帧改变了一个或者几个图层的 zpos(对应 $layer_n$ _sel)，这种一般需要 vsync 到来后下一帧生效。

   所以能使用的基本是根据某一个 **VP** 的 **VSYNC** 来生效，但是在多屏显示的模式下，各个 VP 驱动的屏幕帧率不同，即使帧率相同，他们的相位也各不相同。

这就对应用(composer) 提出了要求：

1. 一个 commit 只能配置一个 CRTC(VP)， 然后这两个寄存器根据这个 commit 配置的 VP 来生效。如果一个 commit 同时为两个 CRTC(VP) 分配图层，一定会有一个 VP 上的寄存器配置无法帧同步(包括port_mux, zorder)，屏幕上显示可能会看到异常。
2. 把一个图层从一个 VP_a 迁移到另外一个 VP_b 上，需要分两个 commit，第一个 commit 在 VP_a 上 disable 这个图层，第二个 commit 再把这个图层迁移到 VP_b 。 而且迁移到 VP_b 后，要等到 port_mux 的配置生效即该图层已经真正切换到 VP_b 上后，，才能对这个图层进行配置，否则对这个图层的配置有可能会在原来的 VP_a 上生效(针对 VP_b 的 port_mux 还未生效，VP_a 的 config done 先生效了，等到 VP_b 的 vsync 到来，其 port_mux 生效，又会立即把这个已经在 VP_a 上生效的图层强制从 Actived 的状态切到 VP_b，这个时刻也有可能会发生异常，比如 产生 POST_BUF_EMPTY 中断)，这个瞬间我们可能会看到 VP_a 上有个异常的内容闪了以下，或者 VP_b 上闪一下绿屏(这是该图层在 Actived 状态被强制从 VP_a 切换到 VP_b 导致的)。
3. composer 在向一个 VP 上迁移图层的时候，必须保证其他的 VP 只使用一个图层，等迁移完成后才能使用多图层合成功能，原因是因为，一个 VP 上的图层数目发生变化后，其他 VP 上的图层数目会跟着变化，进而导致所有 layer 和 MIX 的对应关系变化了，如果这时候使用多图层合成，图层的 alpha 信息和 mix 上的 alpha 信息无法对应上，屏幕会显示异常。
4. 基于这些限制，在实际的应用场景下，composer 应该只在热插拔的时候做图层的分配迁移，热插拔之后，不应该再做图层的迁移切换，否则无法保证屏幕上显示内容的平滑过渡。

1、3，4只要 composer 配合都比较好实现，难以处理的是限制 2。

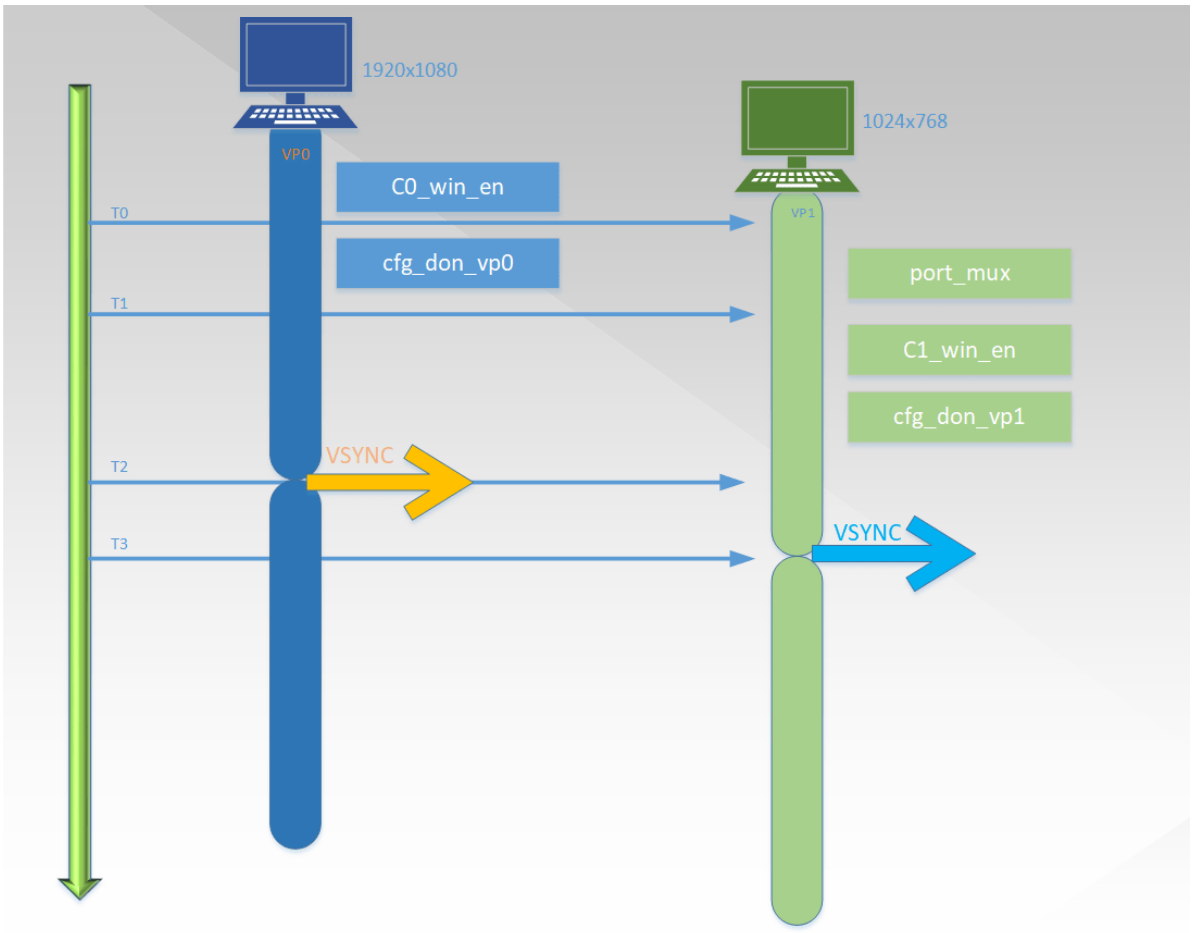
对于 DRM 框架来说，一个 commit 里面，迁移图层和 enable 图层是打包在一起的，要在一个 vsync 内完成，无法做到一个 commit 只迁移图层，而不去开图层，DRM 框架会做检测，不带 fb 的 plane 不让设置：

```
static int drm_atomic_plane_check(const struct drm_plane_state *old_plane_state,
                                  const struct drm_plane_state *new_plane_state)
{
        struct drm_plane *plane = new_plane_state->plane;
        struct drm_crtc *crtc = new_plane_state->crtc;
        const struct drm_framebuffer *fb = new_plane_state->fb;
        unsigned int fb_width, fb_height;
```

```
        struct drm_mode_rect *clips;
        uint32_t num_clips;
        int ret;

        /* either *both* CRTC and FB must be set, or neither */
        if (crtc && !fb) {
                DRM_DEBUG_ATOMIC("[PLANE:%d:%s] CRTC set but no FB\n",
                                 plane->base.id, plane->name);
                return -EINVAL;
        } else if (fb && !crtc) {
                DRM_DEBUG_ATOMIC("[PLANE:%d:%s] FB set but no CRTC\n",
                                 plane->base.id, plane->name);
                return -EINVAL;
        }
```

所以 composer 必定会在一个 commit 里面同时包含迁移图层和使能图层的要求，对于多显来说，多个 VP 由不同的线程独立配置，他们按照各自独立的刷新率产生 VSYNC，会存在如下图所示的问题：



该系统有两个屏幕，分别通过 VP0(1920x1080)、VP1(1024x768) 驱动，在 T0 之前，由于 VP1 上的屏幕还未接入，所有的图层都分配给 VP0 显示。T0 时刻，VP1 上线，composer 会启动一个单独的线程，该线程发起一个 commit 把图层 Cluster1 从 VP0 迁移到 VP1，这时候 VOP 驱动会依次设置 port_mux，enable Cluster1， cfg_done_vp1，(port_mux 在 VP1 的下一个 VSYNC 到来的时刻 T3 生效)，port_mux 生效后，Cluster1 才能被真正迁移到 VP1 上。

由于各个 VP 相互独立，在 Cluster1 图层被配置后，在 VP1 的 VSYNC 到来之前(port_mux 未生效)，有可能 VP0 的 VSYNC 在 T2 时刻先产生了，而且在这之前 VP0 的 commit 已经写了对应的 vp0_cfg_done 寄存器。这时候配置在 Cluster1 上的内容将会跟着 VP0 的 VSYNC 生效，显示在 VP0 上。

图中右边 由 VP0 驱动的屏幕左上角 1024x768 的图象即为 Cluster1 显示出来的，本来是要显示在左边的由 VP1 驱动的屏幕上。

对应的内核 Log 如下：

```
[   34.455312] vsync-vp0
[   34.455523] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Cluster0-win0 active zpos:0 for vp0 from vp0
[   34.455544] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp0: 6 windows, active layers 1
[   34.455576] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] Cluster0-win0 1920 x 1080 --> 1920 x 1080 for vp0 at 0x2f1
2000 by vo_render_threa
[   34.455610] pending_done: 0x0
[   34.455616] cfg done: 0x8001 done_bits: 0x0
[   34.455616]
[   34.467759] vsync-vp1
[   34.467979] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Cluster1-win0 active zpos:2 for vp1 from vp0
[   34.468000] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp1: 1 windows, active layers 1
[   34.468027] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] Cluster1-win0 1024 x 768 --> 1024 x 768 for vp1 at 0x48bd0
00 by vo_render_threa
[   34.475290] vsync-vp0
[   34.475451] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Cluster0-win0 active zpos:0 for vp0 from vp0
[   34.475472] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp0: 5 windows, active layers 1
[   34.475501] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] Cluster0-win0 1920 x 1080 --> 1920 x 1080 for vp0 at 0x229
c000 by vo_render_threa
[   34.484441] vsync-vp1
[   34.495309] vsync-vp0
```

VP0_cfg_done

VP0_VSYNC

针对这种情况，比较可行的处理方案是，**VOP** 驱动发现这个 **commit** 里面包含了图层迁移请求的时候，只修改 **port_mux,** 不使能新迁移过来的图层，即放弃该图层显示当前这一帧，等下一帧 **port_mux** 切换生效后再开启这个图层显示内容。

3. 三屏显示调试遇到的问题

应用场景：

VP0--》HDMI 可插拔

VP1--》MIPI 主屏幕

VP2--》DP 转 VGA， 可插拔

VOP 图层按 3+2+1 策略分配，即 VP0 两个图层（Cluster1、Smart1），VP1 三个图层（Cluster0、Esmart0、Smart0）， VP2 一个图层（Esmart1），设想在当前 VP 对应的接口关闭时可以将当前 VP 的图层挪给其他 VP 使用，这样可以更好的发挥 VOP 的性能，但实际调试过程中，我们遇到了以下问题：

问题一：通过内核节点关闭 VP1->DSI 通路，发现 VP2 -> eDP -> VGA 通路闪黑屏：

```
echo off > /sys/class/drm/card0-DSI-1/status
```

分析过程：

1. 将 VP2 背景色设置为蓝色， 发现黑屏现象变成蓝屏，设定判断条件停在出错帧，确认此时图层配置未及时生效，导致黑色/蓝色背景层显示出来；
2. 场景分解：

(1) 开机默认三屏都显示

Display0： [Cluster1、Smart1] -> VP0 -> HDMI -> TV

Display1： [Cluster0、Esmart0、Smart0] -> VP1 -> MIPI DSI -> LCD

Display2： [Esmart1] -> VP2 -> eDP -> eDP2VGA -> TV

(2) 应用关闭 Display1 通路，应用将 VP1 上的 3 个图层挪到 VP2 使用， 此时应用会刷一帧将 Cluster0 配置给 VP2，在应用看来这一帧数据要下一个 VSYNC 来的时候正常显示在屏上，实际测试看，Cluster0 的寄存器没有生效；

(3) 结合本文第二点的描述以及 IC 的说明，确认导致该问题的原因是：

> 硬件设计上第一个config done + fs 只能让 Port mux 相关寄存器生效，Cluster0 的寄存器需要等到 Port mux寄存器生效后再来一个 config done + fs 才能生效。

3. 软件上的处理

   软件在遇到这种场景，在修改 Port mux 的时候，加一个 config done，等到 Port mux 生效了， 再走图层配置的流程；

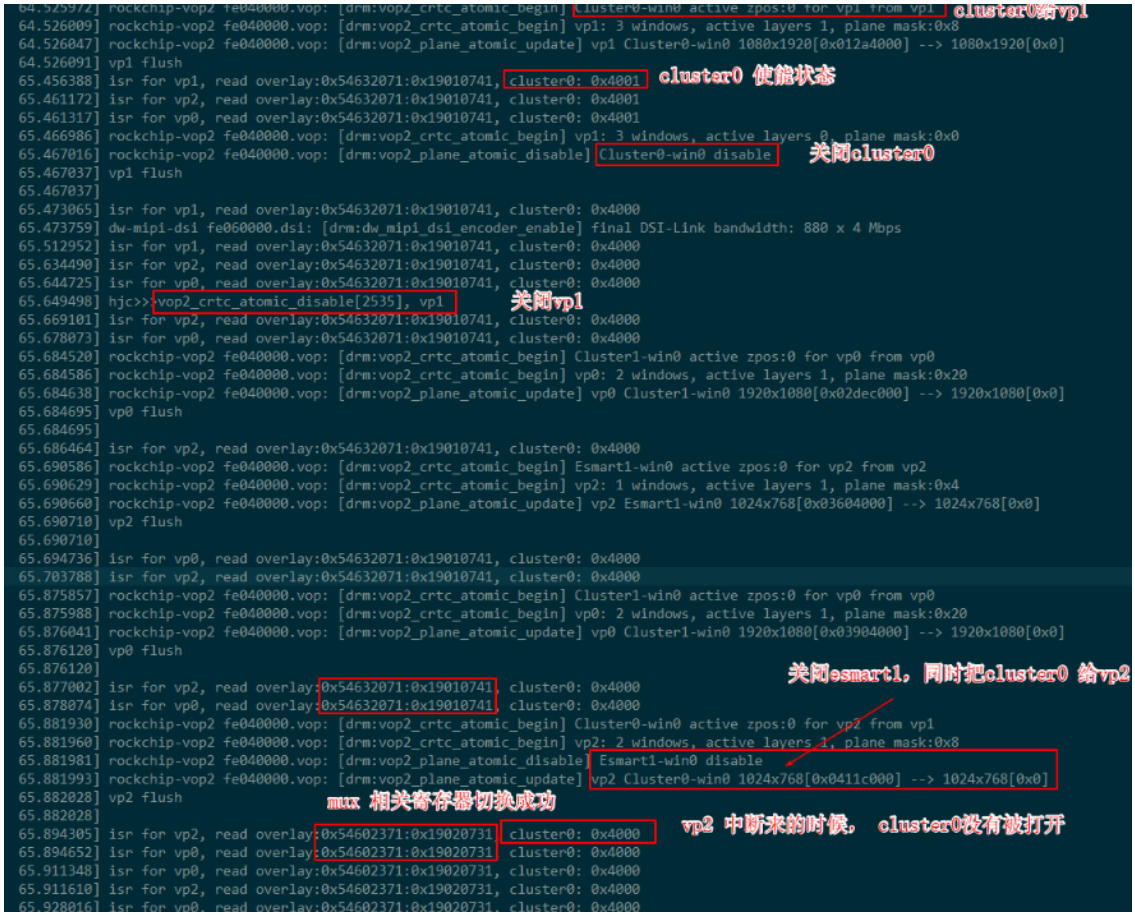
4. 加上以上修改后可以让关闭 VP1 通路时， VP2 通路不闪屏，但是还存在以下问题

   (1) 在一帧配置的中间过程强制插入一个的 config done 让 Port mux 生效，可能会导致 delay num/alpha 等不到预期效果；

   (2) Port mux 需要等一个 VSYNC 才能生效，所以从应用上看此时有一帧数据就会被刷两帧的时间，出现卡顿现象;

   (3）强制多等一个 config done，相当于多等了一个 vsync，应用层的 composer 监控到 vsync 时间后，一般都会认为上一帧的 buffer 已经显示完毕，会把这个 buffer 释放掉或者继续绘制下一帧内容，实际上 VOP 还在显示这个 buffer 的内容。如果释放掉，立刻就会引起 VOP 内存访问异常，如果不释放，继续绘制下一帧内容，当这个 buffer 的内容是 AFBC 格式的时候，VOP 正在访问的 AFBC header 可能就被破坏了，也会引起异常的内存访问。

5. 出错流程记录



问题二：通过内核节点打开 VP1->DSI 通路，发现 VP2 -> eDP -> VGA 通路闪黑屏：

```
echo on > /sys/class/drm/card0-DSI-1/status
```

分析过程

1. DSI 拔出后，图层策略从原来的 VP0[Cluster1+Smart1]，VP1[Cluster0+Smart0+Esmart0]，VP2[Esmart1] 变成了 VP0[Cluster1+Smart1]，VP1[Smart0+Esmart0]，VP2[Cluster0+Esmart1]，VP2 使用 Cluster0 显示；此时 OVERLAY 配置为：0x54602371 0x19020731
2. DSI 插入后，VP2 关闭 Cluster0，使用 Esmart1，图层策略保持：VP0[Cluster1+Smart1]，VP1[S0+ES0]，VP2[Cluster0+Esmart1],此时对于当前 VP 来说，并不知道合适 Cluster1 会被挪回 VP1，所以此时 Smart1被放到了 layer4，此时 OVERLAY 配置为：0x54062371 0x19020731
3. 应用提交一帧给 VP1 显示，使用的图层为 Cluster0，此时 Cluster0 被迁到 VP1， VP1 的 Port mux 从3变成4，同时 Cluster0 会被移动到 layer2，变成 Cluster0->layer2，Smart0->layer5，变成未使能的 Smart0 挪到给了 VP2; 导致 VP2 显示黑屏，此时 OVERLAY 配置为：0x54362071 0x19010741
4. 出错流程记录

```
[  969.894032] isr for vp0, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  969.895793] isr for vp2, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  969.910704] isr for vp0, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
---------------
insert
---------------
[  969.913054] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_enable] Update mode to 1080x1920p60, type: 16 for vp1
[  969.913097] isr for vp2, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  969.913198] dw-mipi-dsi fe060000.dsi: [drm:dw_mipi_dsi_encoder_enable] final DSI-Link bandwidth: 880 x 4 Mbps
[  969.927372] isr for vp0, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.294230] isr for vp2, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.296388] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp1: 2 windows, active layers 0, plane mask:0x0
[  970.296455] vp1 flush

[  970.326729]
[  970.327399] isr for vp0, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.328913] isr for vp2, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.329804] isr for vp1, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.343998] isr for vp0, read overlay:0x54602371:0x19020731, cluster0: 0x4001, esmart1:0x4000
[  970.344617] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Esmart1-win0 active zpos:0 for vp2 from vp2
[  970.344687] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp2: 2 windows, active layers 1, plane mask:0x4
[  970.344737] vp2, nr layer:1, used_layers:4, i:0, layer id:4, cur win:Esmart1-win0, cur win_old layer id:5
[  970.344767] vp2, old layer id:5, layer id, old layer store win:Cluster0-win0
[  970.344826] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] vp2 Esmart1-win0 1024x768[0x03604000] --> 1024x768[0x0]
[  970.345184] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_disable] vp2 Cluster0-win0 disable, phys_id:0
[  970.345240] vp2 flush
================================================================================
[  970.363530] isr for vp2, read overlay:0x54062371:0x19020731, cluster0: 0x4000, esmart1:0x4001
[  970.364313] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Cluster1-win0 active zpos:0 for vp0 from vp0
[  970.364378] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp0: 2 windows, active layers 1, plane mask:0x20
[  970.364404] vp0, nr layer:1, used_layers:0, i:0, layer id:0, cur win:Cluster1-win0, cur win_old layer id:0
[  970.364415] vp0, old layer id:0, layer_id, old layer store win:Cluster1-win0
[  970.364452] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] vp0 Cluster1-win0 1920x1080[0x03904000] --> 1920x1080[0x0]
[  970.364511] vp0 flush
....
[  970.679772] isr for vp1, read overlay:0x54062371:0x19020731, cluster0: 0x4000, esmart1:0x4001
[  970.681488] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] Cluster0-win0 active zpos:0 for vp1 from vp2
[  970.681522] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_begin] vp1: 3 windows, active layers 1, plane mask:0x8
[  970.681544] vp1, nr layer:1, used_layers:2, i:0, layer id:2, cur win:Cluster0-win0, cur win_old layer id:5
[  970.681557] vp1, old layer id:5, layer_id, old layer store win:Smart0-win0
[  970.692707] isr for vp2, read overlay:0x54062371:0x19020731, cluster0: 0x4000, esmart1:0x4001
[  970.693956] isr for vp0, read overlay:0x54062371:0x19020731, cluster0: 0x4000, esmart1:0x4001
================================================================================
[  970.696422] isr for vp1, read overlay:0x54362071:0x19010741, cluster0: 0x4000, esmart1:0x4001
[  970.696447] rockchip-vop2 fe040000.vop: [drm:vop2_plane_atomic_update] vp1 Cluster0-win0 1080x1920[0x012a4000] --> 1080x1920[0x0]
[  970.696484] vp1 flush
[  970.710013] isr for vp2, read overlay:0x54362071:0x19010741, cluster0: 0x4000, esmart1:0x4001
```

默认策略：VP0[C1+S1]，VP1[C0+S0+ES0]，VP2[ES1]
关闭DSI后，VP1的C0 挪到V2使用，变成
VP0[C1+S1]，VP1[S0+ES0]，VP2[ES1+C0]

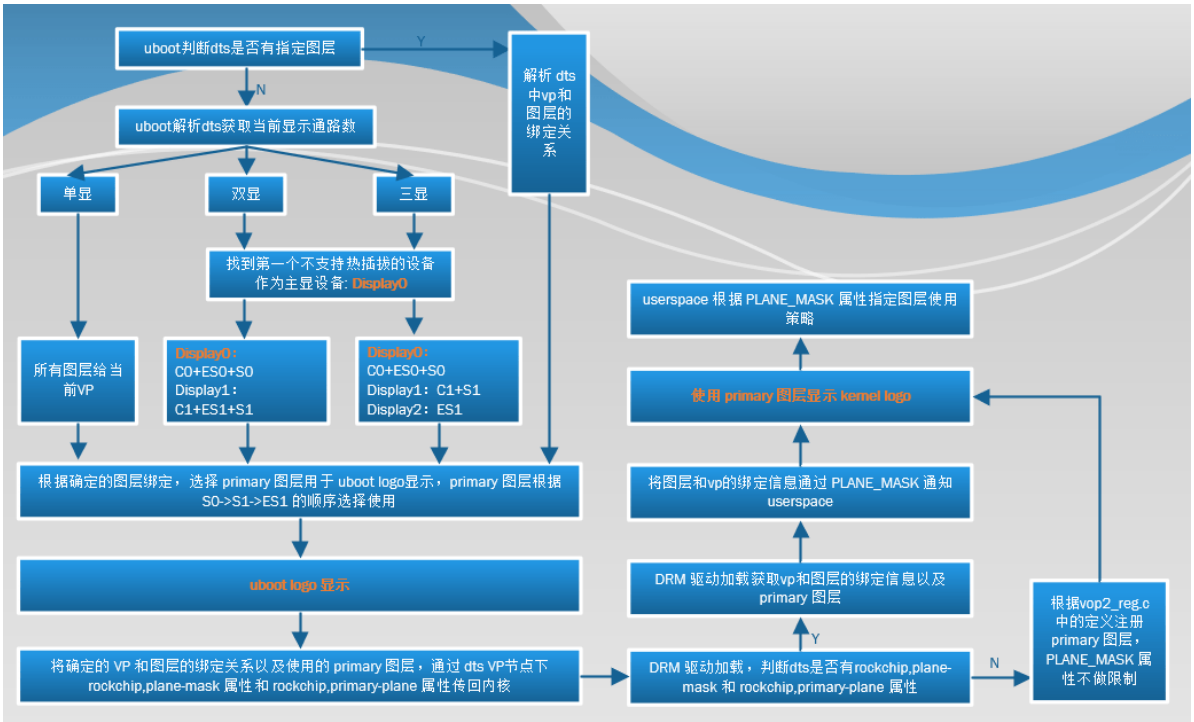
使能VP1->DSI

VP2刷一帧，由于此时C0还没被挪回VP1,所以看到的used layer=4，VP2的esmart1图层被放到layer4，layer5 还是C0

VP1 开始刷第一帧配置到C0，会让layer5的c0放到layer2，layer2和layer5 的win交换，导致smart0被放到layer5，
同时此时vp1的图层从2个变成3个，vp1的port mux从3 改到4，导致vp2 使用未使能的smart0，不显示；
vp1 也出现显示异常；

4. 当前显示策略

基于以上分析结果，我们不做图层动态切换，制定以下显示策略：



如果产品有需求需要指定图层策略的，可以参考下面的 demo 在所使用的dts文件中指定每个 VP 所使用的图层：

```
#define ROCKCHIP_VOP2_CLUSTER0      0
#define ROCKCHIP_VOP2_CLUSTER1      1
#define ROCKCHIP_VOP2_ESMART0       2
#define ROCKCHIP_VOP2_ESMART1       3
#define ROCKCHIP_VOP2_SMART0        4
```

```
#define ROCKCHIP_VOP2_SMART1        5

&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>;
};
```

注意：由于驱动中我们只设定了SMART0/SMART1/ESMART1可以做为 primary 图层，所以对于 rockchip,primary-plane 只能从以上三个图层中选出；