Rockchip
瑞芯微电子

Rockchip Graphics

FAQ

DRM Hardware Composer

发布版本:1.00

日期:2018.12

免责声明

本文档按“现状”提供，福州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。

版权所有 © 2018 福州瑞芯微电子股份有限公司

福州瑞芯微电子股份有限公司
Fuzhou 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： +86-591-83991906
客户服务传真： +86-591-83951833
客户服务邮箱： service@rock-chips.com

前言

概述

本文档主要介绍 Hardware Composer 及 SurfaceFlinger Services 产品使用过程中常见的问题与调试手段。

相关工程师遇到有关本文档涉及的问题，可尝试通过提供方法进行调试，收敛问题，提高解决问题的效率，进一步解决问题。如果遇到还无法解决的问题，可将初步的调试结果及对应 log 提供，方便显示相关工程师定位问题。

产品版本

芯片名称	**Android** 版本
RK3399	Android 7.1 / 8.1 / 9.0
RK3328	Android 8.1 / 9.0
RK3326 / PX30	Android 8.1 / 9.0
RK3288	Android 7.1 / 8.1 / 9.0
RK312x	Android 8.1 / 9.0

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

修订记录

日期	版本	作者	审核	修改说明
2018-12-13	V1.00	李斌	黄德胜	正式发布

目录

1 图形通用调试手段介绍

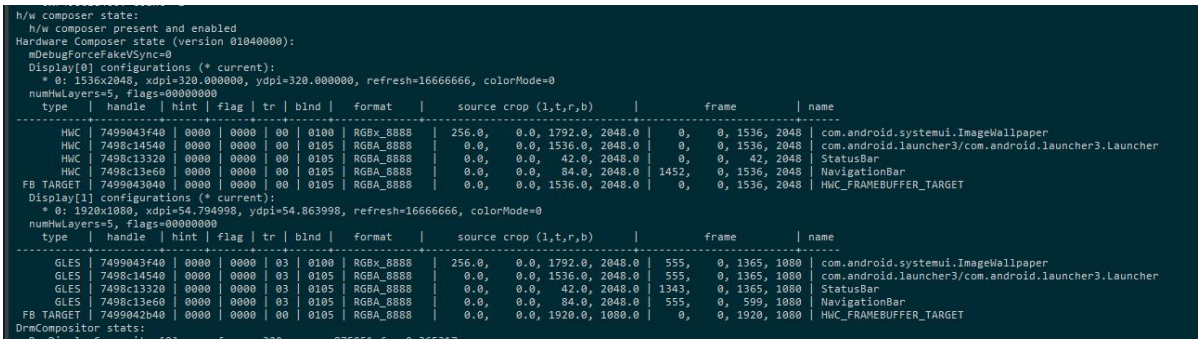
1.1 SurfaceFlinger

1.1.1 Dumpsys SurfaceFlinger

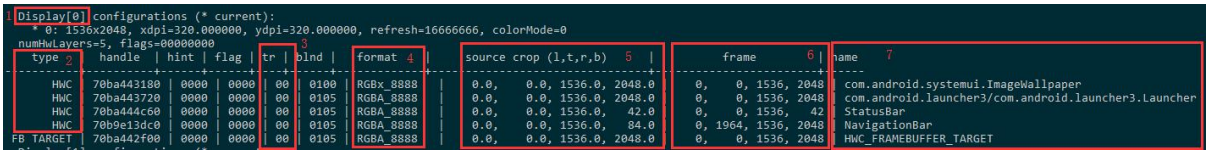
```
//command line
dumpsys SurfaceFlinger
```

该命令可查询打印当前系统注册显示设备的情况以及提交显示的 Layer 情况：

Android 7.1 及以下版本：



```
h/w composer state:
  h/w composer present and enabled
Hardware Composer state (version 01040000):
  mDebugForceFakeVSync=0
  Display[0] configurations (* current):
    * 0: 1536x2048, xdpi=320.000000, ydpi=320.000000, refresh=16666666, colorMode=0
  numHwLayers=5, flags=00000000
    type   |  handle  | hint | flag | tr | blnd |   format    |     source crop (l,t,r,b)      |          frame         | name
-----------+----------+------+------+----+------+-------------+--------------------------------+------------------------+------
       HWC | 7499043f40 | 0000 | 0000 | 00 | 0100 | RGBx_8888   |  256.0,    0.0, 1792.0, 2048.0 |    0,    0, 1536, 2048 | com.android.systemui.ImageWallpaper
       HWC | 7498c14540 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0, 2048.0 |    0,    0, 1536, 2048 | com.android.launcher3/com.android.launcher3.Launcher
       HWC | 7498c13320 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0,   42.0, 2048.0 |    0,    0,   42, 2048 | StatusBar
       HWC | 7498c13e60 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0,   84.0, 2048.0 | 1452,    0, 1536, 2048 | NavigationBar
 FB TARGET | 7499043040 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0, 2048.0 |    0,    0, 1536, 2048 | HWC_FRAMEBUFFER_TARGET
  Display[1] configurations (* current):
    * 0: 1920x1080, xdpi=54.794998, ydpi=54.863998, refresh=16666666, colorMode=0
  numHwLayers=5, flags=00000000
    type   |  handle  | hint | flag | tr | blnd |   format    |     source crop (l,t,r,b)      |          frame         | name
-----------+----------+------+------+----+------+-------------+--------------------------------+------------------------+------
       GLES | 7499043f40 | 0000 | 0000 | 03 | 0100 | RGBx_8888   |  256.0,    0.0, 1792.0, 2048.0 |  555,    0, 1365, 1080 | com.android.systemui.ImageWallpaper
       GLES | 7498c14540 | 0000 | 0000 | 03 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0, 2048.0 |  555,    0, 1365, 1080 | com.android.launcher3/com.android.launcher3.Launcher
       GLES | 7498c13320 | 0000 | 0000 | 03 | 0105 | RGBA_8888   |    0.0,    0.0,   42.0, 2048.0 | 1343,    0, 1365, 1080 | StatusBar
       GLES | 7498c13e60 | 0000 | 0000 | 03 | 0105 | RGBA_8888   |    0.0,    0.0,   84.0, 2048.0 |  555,    0,  599, 1080 | NavigationBar
 FB TARGET | 7499042b40 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1920.0, 1080.0 |    0,    0, 1920, 1080 | HWC_FRAMEBUFFER_TARGET
DrmCompositor stats:
```



```
Display[0] configurations (* current):
  * 0: 1536x2048, xdpi=320.000000, ydpi=320.000000, refresh=16666666, colorMode=0
 numHwLayers=5, flags=00000000
   type 2  |  handle  | hint | flag | tr 3 | blnd |  format 4   |     source crop (l,t,r,b) 5    |        frame 6         | name 7
-----------+----------+------+------+------+------+-------------+--------------------------------+------------------------+------
       HWC | 70ba443180 | 0000 | 0000 | 00 | 0100 | RGBx_8888   |    0.0,    0.0, 1536.0, 2048.0 |    0,    0, 1536, 2048 | com.android.systemui.ImageWallpaper
       HWC | 70ba443720 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0, 2048.0 |    0,    0, 1536, 2048 | com.android.launcher3/com.android.launcher3.Launcher
       HWC | 70ba444c60 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0,   42.0 |    0,    0, 1536,   42 | StatusBar
       HWC | 70b9e13dc0 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0,   84.0 |    0, 1964, 1536, 2048 | NavigationBar
 FB TARGET | 70ba442f00 | 0000 | 0000 | 00 | 0105 | RGBA_8888   |    0.0,    0.0, 1536.0, 2048.0 |    0,    0, 1536, 2048 | HWC_FRAMEBUFFER_TARGET
```

红框依次信息如下：

1 设备类型：

Display[0] 主屏
Display[1] 副屏
Display[2] 虚拟屏

2 合成方式 type ：

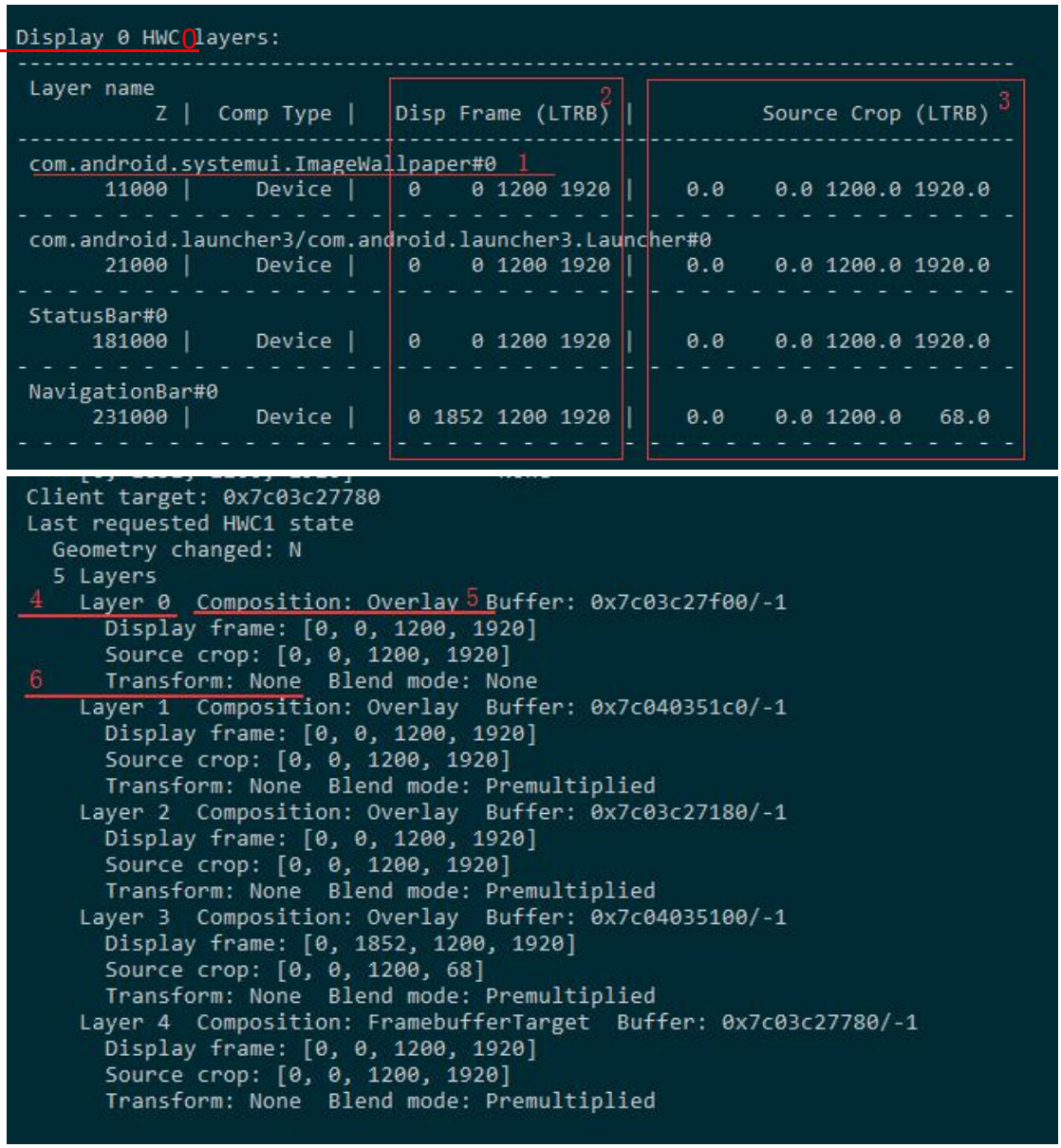
HWC / HWC_TOWIN0 overlay 硬件合成
HWC_MIX / HWC_MIX_V2 overlay 硬件合成
GLES GPU 合成
blit RGA 合成
HWC_NODRAW 回退 GPU 合成，但 GPU 不处理这层
FB TARGET GPU 合成输出 buffer 类型，可以不关注

3 变换矩阵 tr:

0x01 垂直镜像 HAL_TRANSFORM_FLIP_H
0x02 水平镜像 HAL_TRANSFORM_FLIP_V
0x04 旋转 90 度 HAL_TRANSFORM_ROT_90
0x03 旋转 180 度 HAL_TRANSFORM_ROT_180
0x07 旋转 270 度 HAL_TRANSFORM_ROT_270

4 格式 format 对应 Layer 的格式
5 源数据纹理坐标 source crop (l,t,r,b) 对应 left,top,right,bottom
6 目标显示区域顶点坐标 frame 对应 left,top,right,bottom
7 Name 对应 APK LayerName

Android 8.1 及以上版本



标注信息依次为：

0 设备类型：

Display 0 主屏
Display 1 副屏
Display 2 虚拟屏

1 Layer name 对应 APK LayerName

2 目标显示区域顶点坐标 frame 对应 left,top,right,bottom

3 源数据纹理坐标 source crop (l,t,r,b) 对应 left,top,right,bottom

4.Layer 0 对应 Wallpaper 也就是最底层的 Layer,依次往上对应

5 合成类型 Composition ：

Overlay overlay 硬件合成
Framebuffer GPU 合成
Unknow 基本上为 HWC_NODRAW 策略
FramebufferTarget GPU 合成输出 buffer 类型，不关注

6 变换矩阵 Transform：

None 无角度
Rotate90 旋转 90 度
Rotate180 旋转 180 度
Rotate270 旋转 270 度

1.1.2 Printf mFps

```
/*
 * 该命令可输出系统帧率统计：
 * 统计方式为通过计算大于 500ms 的时间间隔内送显的帧数，
 * 来得出系统帧率，可作为系统帧率参考。
 */

setprop debug.sf.fps 1
logcat -c ;logcat | grep mFps
```

敲入命令后输出如下，说明系统刷新帧率稳定在 60 帧：

```
rk3399_mid:/ # setprop debug.sf.fps 1
rk3399_mid:/ # logcat -c ;logcat | grep mFps
01-18 08:58:06.012   229   229 D SurfaceFlinger: mFps = 0.002
01-18 08:58:06.513   229   229 D SurfaceFlinger: mFps = 59.930
01-18 08:58:07.032   229   229 D SurfaceFlinger: mFps = 59.696
01-18 08:58:07.546   229   229 D SurfaceFlinger: mFps = 60.324
01-18 08:58:08.063   229   229 D SurfaceFlinger: mFps = 59.973
01-18 08:58:08.578   229   229 D SurfaceFlinger: mFps = 60.123
01-18 08:58:09.078   229   229 D SurfaceFlinger: mFps = 59.982
01-18 08:58:09.582   229   229 D SurfaceFlinger: mFps = 59.616
01-18 08:58:10.096   229   229 D SurfaceFlinger: mFps = 60.325
01-18 08:58:10.596   229   229 D SurfaceFlinger: mFps = 59.989
01-18 08:58:11.096   229   229 D SurfaceFlinger: mFps = 60.000
```

1.2 Hardware Composer

1.2.1 HWC version

```
/*
 * 该命令可输出 HWC 版本信息：
 *      sys.ghwc.commit      HWC 最新的 commit-ID
 *      sys.ghwc.version     HWC 版本信息
 */
adb shell getprop | grep ghwc
```

如下表示，最新 commit-id 为 d107032，HWC version 为 0.53

该信息有助于代码维护者同步现场代码状态以及定位问题，所以每次定位 HWC 问题首先应该要确认代码 version..

```
$ adb shell getprop | grep ghwc
[sys.ghwc.commit]: [commit-id:d107032]
[sys.ghwc.version]: [0.53-rk3399-MID]
```

1.2.2 Enable/disable HWC

```
/*
 * sys.hwc.compose_policy
 *    6 :      Enable HWC, HWC 策略匹配正常运行
 *    0 :      Disable HWC, 关闭 HWC 策略匹配，所有合成工作由 GPU 完成
 * Tips:
 *    要确定是否是 HWC 匹配合成策略导致的问题，可直接通过设置该属性，开关 HWC,来初步
 *    定位问题
 *    1) 若 HWC 打开有问题，关闭问题消失，则问题基本确定为 HWC 逻辑导致
 *    2) 若 HWC 打开与关闭均存在问题，则基本排除 HWC 策略匹配导致问题
 */
setprop sys.hwc.compose_policy 6      //HWC enable
setprop sys.hwc.compose_policy 0      //HWC disable
```

设置完成后，可通过 1.1.1 Dumpsys SurfaceFlinger 命令来验证是否修改成功。

1.2.3 DumpSurface

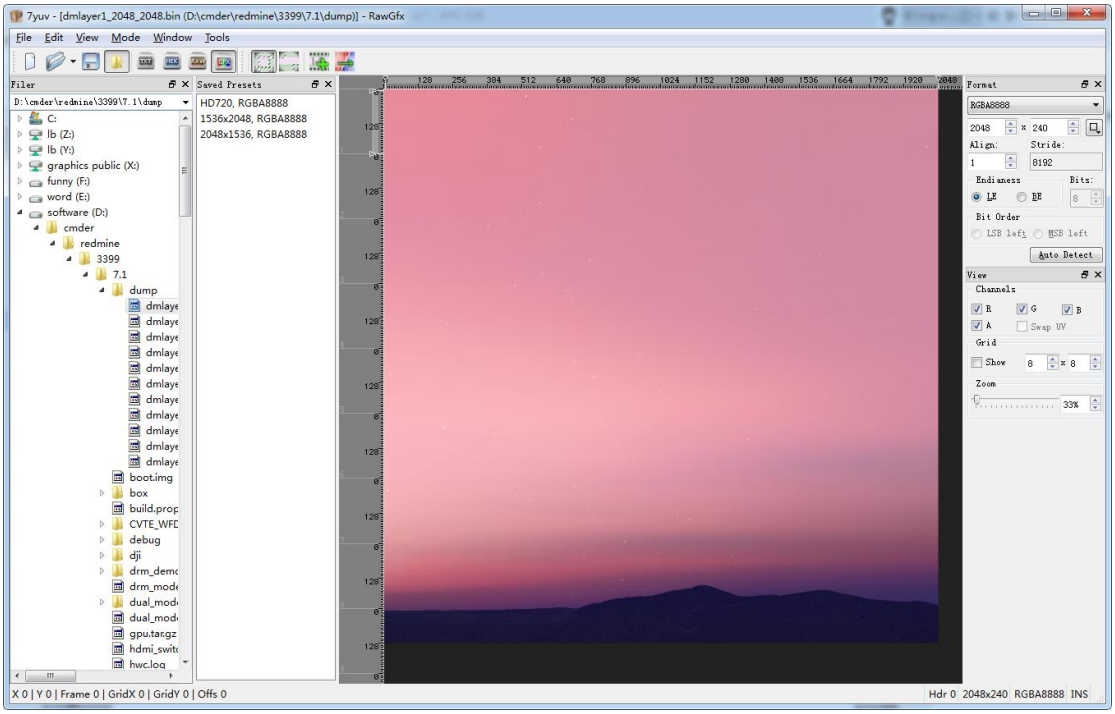
```
/*
 * 该命令可将所有 apk 提交 layer 的内容以 bin 文件的形式写出
 * 输出路径为：                /data/dump/dmlayer%d_%d_%d.bin
 * 文件命名方式为：            dmlayer%d_%d_%d.bin   DumpSurfaceCount,stride,height
 * 该文件可确认 apk 提交数据是否异常：
 *      1) 若 apk 提交数据正常，显示异常，那么问题很有可能出现在后端合成流程。
 *      2) 若 apk 提交数据异常，那问题很有可能出现在从前端渲染。
 *
 * 注意事项：
 *      1)部分平台可能需要预先创建好/data/dump 目录，因为 HWC 没有创建文件夹权限
 *      2)9.0 版本 data 分区无权限问题，可通过 setenforce 0 后再进行操作
 */

adb root
adb remount
adb shell "setprop sys.dump true"
adb pull /data/dump/
```

操作后在对应/data/dump/ 存在如下输出：

```
rk3399_mid:/ # setprop sys.dump true
rk3399_mid:/ # ls data/dump/
dmlayer10_1536_2048.bin dmlayer1_2048_2048.bin dmlayer3_48_2048.bin dmlayer5_2048_2048.bin dmlayer7_48_2048.bin dmlayer9_2048_2048.bin
dmlayer11_48_2048.bin   dmlayer2_1536_2048.bin dmlayer4_96_2048.bin dmlayer6_1536_2048.bin dmlayer8_96_2048.bin
```

该 bin 文件可通过 7yuv 打开，查看：

以下为 com.android.systemui.ImageWallpaper apk 提交显示的源数据：



1.2.4 Hwc log

```
/*
 * 该命令可打开 HWC 所有等级 log 输出，通过 logcat 抓打印
 */
adb shell "setprop sys.hwc.log 511"
adb shell "logcat -c ;logcat" > hwc.log
```

可将该 Log 输出上传至 Redmine 或发送给 HWC 维护者分析。

1.2.5 Close releaseFence

将 HWC 目录的 Android.mk `ENABLE_RELEASE_FENCE` 设置为 `0`

HWC 目录：`hardware/rockchip/hwcomposer`

```
diff --git a/Android.mk b/Android.mk
index de3f78f..18fabda 100755
--- a/Android.mk
+++ b/Android.mk
@@ -388,7 +388,7 @@ LOCAL_CPPFLAGS += -DUSE_DRM_GENERIC_IMPORTER \
                  -DRK_INVALID_REFRESH=$(RK_INVALID_REFRESH) -DRK_HDR_PERF_MODE=0 \
                   -DRK_3D_VIDEO=$(RK_3D_VIDEO) -DRK_PRINT_LAYER_NAME=$(RK_PRINT_LAYER_NAME) \
                  -DRK_SORT_AREA_BY_XPOS=$(RK_SORT_AREA_BY_XPOS) -DRK_HOR_INTERSECT_LIMIT=$(RK_HOR_INTERSECT_LIMIT) \
-                 -DENABLE_RELEASE_FENCE=1 -DFORCE_WAIT_ACQUIRE_FENCE=0 -DRK_RGA_SCALE_AND_ROTATE=$(RK_RGA_SCALE_AND_ROTATE) \
+                 -DENABLE_RELEASE_FENCE=0 -DFORCE_WAIT_ACQUIRE_FENCE=0 -DRK_RGA_SCALE_AND_ROTATE=$(RK_RGA_SCALE_AND_ROTATE) \
                   -DRGA_VER=$(RGA_VER) -DRK_PER_MODE=$(RK_PER_MODE) -DRK_ROTATE_VIDEO_MODE=$(RK_ROTATE_VIDEO_MODE) \
                   -DRK_CTS_WORKROUND=$(RK_CTS_WORKROUND)
 MAJOR_VERSION := "RK_GRAPHICS_VER=commit-id:$(shell cd $(LOCAL_PATH) && git log  -1 --oneline | awk '{print $$1}')"
```

关闭 ENABLE_RELEASE_FENCE 即可关闭 releaseFence，此时后端显示系统不再使用 Fence,画面可能会出现撕裂的现象，是正常的。

1.2.6 Modetest

编译方法：

```
mmm external/libdrm/tests/modetest/ -j8
```

使用方法：

1. 将编译输出文件更新到设备，并修改执行权限
2. 输出底层 DRM 显示设备信息，直接执行 modetest demo

```
adb shell modetest > modetest.log
```

3. 利用 modetest demo 点亮屏幕
   a) 确认底层 connector ID 与 mode:通过以下命令输出底层 connector 信息

```
adb shell modetest -c
```

```
trying to open device 'rockchip'...done
Connectors:
id      encoder status          name            size (mm)       modes   encoders
87      0       connected       eDP-1           0x0             1       86
  modes:
        name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1536x2048 60 1536 1548 1564 1612 2048 2056 2060 2068 flags: nhsync, nvsync; type: preferred
  props:
        1 EDID:
                flags: immutable blob
                blobs:
```

   b) 通过 modetest 点亮屏幕：

   比如我们现在需要测试挖掘机 eDP 屏幕的底层驱动情况，获取到上述信息后，通过一下命令点亮屏幕，画面为彩条，如下图：

```
adb shell modetest -s 87:1536x2048
```

若此时显示正常，则表示底层显示驱动正常，若此命令显示异常，则因先从底层驱动检查是否有异常。

1.3 Gralloc

1.3.1 Disable afbdc

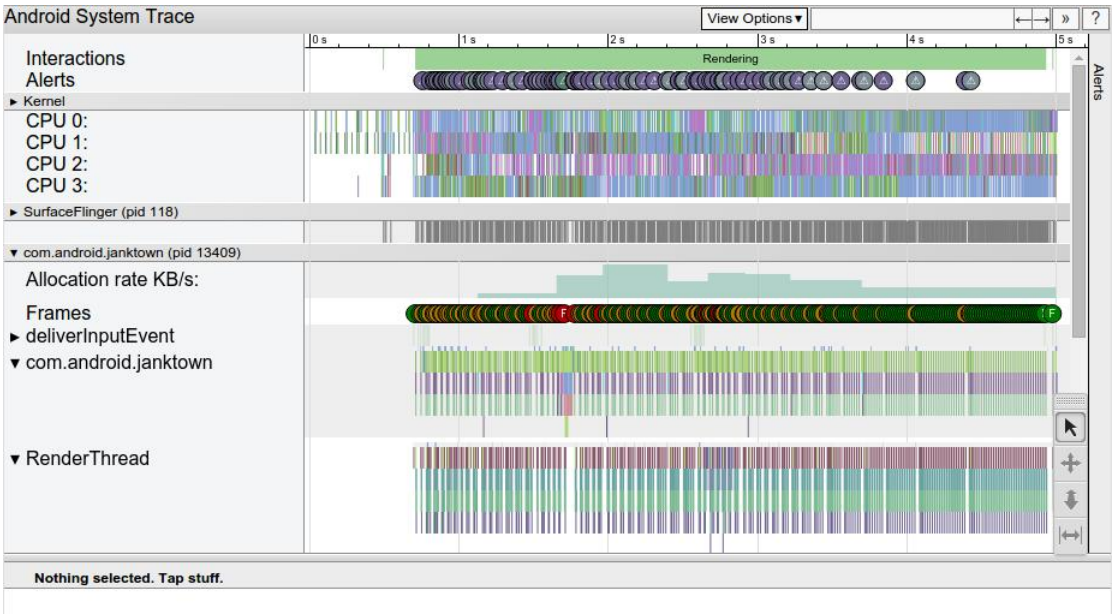
AFBDC 编码是 GPU 与 VOP 之前以降低带宽为目的的一种图像压缩格式，只有 VOP BIG 支持 AFBDC 编码，VOP LITTLE 不支持该编码格式，故产品应用过程中有可能出现将 AFBDC 数据配置到 VOP LITTLE 的情况，这种情况需要关闭 AFBDC 编码。

关闭 AFBDC 编码，需要修改 hardware/rockchip/libgralloc/Android.mk 文件，将其中的 USE_AFBC_LAYER 设置为 0，即可，如下补丁：

```
/*
 * 关闭 GPU AFBC 功能：
 * 补丁修改路径：              hardware/rockchip/libgralloc
 * 更新动态链接库 so：
 *
 *    Android 7.1：
 *              32 位           system/lib/libgralloc_drm.so
 *                              system/lib/hw/gralloc.rk30board.so
 *              64 位           system/lib64/libgralloc_drm.so
 *                              system/lib64/hw/gralloc.rk30board.so
 *    Android 8.1 及以上：
 *              32 位           vendor/lib/libgralloc_drm.so
 *                              vendor/lib/hw/gralloc.rk30board.so
 *              64 位           vendor/lib64/libgralloc_drm.so
 *                              vendor/lib64/hw/gralloc.rk30board.so
 */

diff --git a/Android.mk b/Android.mk
index dd3d518..f99dd33 100644
--- a/Android.mk
+++ b/Android.mk
@@ -128,7 +128,7 @@ MALI_USE_YUV_AFBC_WIDEBLK?=0
 GRALLOC_INIT_AFBC?=1

 # for bifrost GPU, use afbc layer by default.
-USE_AFBC_LAYER = 1
+USE_AFBC_LAYER = 0

 ifeq ($(strip $(TARGET_BOARD_PLATFORM)),rk3399)
 USE_AFBC_LAYER = 1
```

1.4 Systrace

该 systrace 命令允许您在系统级别的设备上运行的所有进程中收集和检查时序信息。它结合了来自 Android 内核的数据，例如 CPU 调度程序，磁盘活动和应用程序线程，以生成 HTML 报告，类似于下图中所示：



https://developer.android.com/studio/command-line/systrace

1.5 调频调压

参见 rockchip-产品频率查询与修改.pdf 文档

1.6 通用异常日志分析

1.6.1 ANR 日志

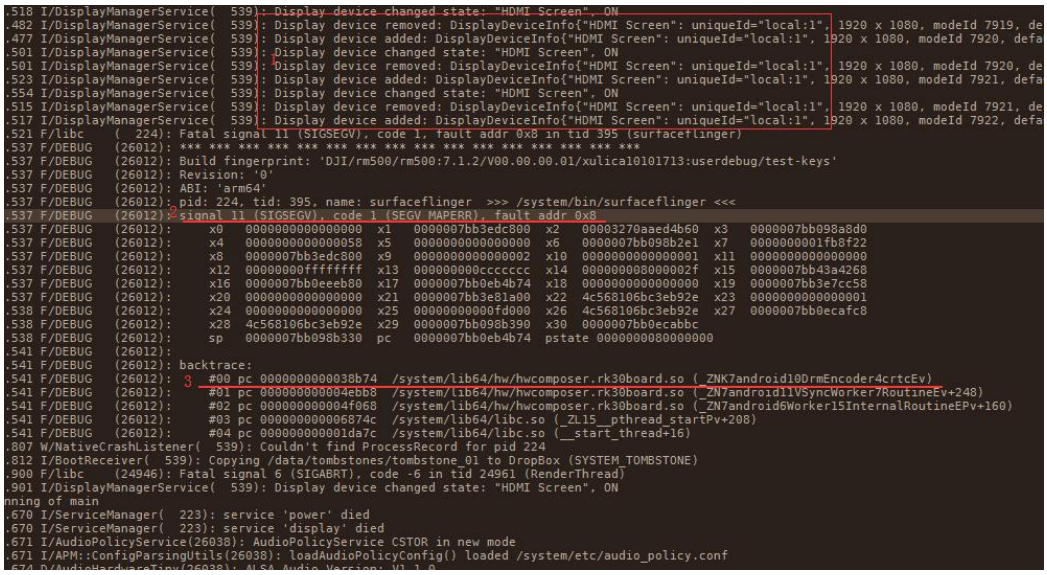
应用程序未响应（ANR: Application Not Responding）发生通常会输出/data/anr/traces.txt 文件，我们只要分析 traces.txt 文件就可以大致知道问题现场位置。

http://haiolv.github.io/2016/06/13/android-anr%E5%88%86%E6%9E%90/
Android ANR 分析

2.2.1.3 画面卡住，提供 ANR 案例分析

1.6.2 Crash 日志

我们以 2.2.1.4 Crash 重启问题为例，分析 Crash 日志，现场日志如下：



我们通过构造该错误，在 HWC 代码适当位置加上如下代码：

```
struct test_t{
    int a = 0;
    int b = 0;
    int c = 0;
    void add(){return;};
};
struct test_t *test_a;    //构造 test_a
test_a = NULL;            //设置为 NULL
test_a->c = 1;            //访问 NULL 指针的成员
```

更新入设备后即可重新问题打印，具体打印如下：

```
--------- beginning of crash
01-18 08:57:38.507  1510  1510 F libc    : Fatal signal 11 (SIGSEGV), code 1, fault addr 0x8 in tid 1510 (surfaceflinger)
--------- beginning of main
01-18 08:57:38.508   216   216 W         : debuggerd: handling request: pid=1510 uid=1000 gid=1003 tid=1510
01-18 08:57:38.523  2630  2630 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
01-18 08:57:38.523  2630  2630 F DEBUG   : Build fingerprint: 'Android/rk3399_all/rk3399_all:7.1.2/NHG47K/zwp06120914:userdebug/test-keys'
01-18 08:57:38.523  2630  2630 F DEBUG   : Revision: '0'
01-18 08:57:38.523  2630  2630 F DEBUG   : ABI: 'arm64'
01-18 08:57:38.524  2630  2630 F DEBUG   : pid: 1510, tid: 1510, name: surfaceflinger  >>> /system/bin/surfaceflinger <<<
01-18 08:57:38.524  2630  2630 F DEBUG   : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x8
01-18 08:57:38.524  2630  2630 F DEBUG   :     x0   0000000000000001  x1   0000000000000000  x2   000000000000000a  x3   0000000000000031
01-18 08:57:38.524  2630  2630 F DEBUG   :     x4   0000000000000062  x5   0000000000000000  x6   0000007fe01806ee  x7   6bfefefefefefefe
01-18 08:57:38.524  2630  2630 F DEBUG   :     x8   0000000000000008  x9   0000000000000001  x10  0000000000000000  x11  0101010101010101
01-18 08:57:38.524  2630  2630 F DEBUG   :     x12  0000000000000008  x13  ffffffffffffffff  x14  ffffffff000000  x15  ffffffffffffffff
01-18 08:57:38.524  2630  2630 F DEBUG   :     x16  0000070c5c3d3e0  x17  0000070c5bd004c  x18  0000070c16c3480  x19  0000070c5667418
01-18 08:57:38.524  2630  2630 F DEBUG   :     x20  0000000000000001  x21  0000000000000003  x22  00000000ffffffff  x23  0000070c1b975a0
01-18 08:57:38.524  2630  2630 F DEBUG   :     x24  000000000066666f  x25  0000070c5a8d000  x26  0000070c567c920  x27  000000000100ffff
01-18 08:57:38.524  2630  2630 F DEBUG   :     x28  0000070c567c800  x29  0000007fe0180ca0  x30  0000070c5a5994c
01-18 08:57:38.524  2630  2630 F DEBUG   :     sp   0000007fe0180390  pc   0000070c5a5995c  pstate 0000000060000000
01-18 08:57:38.532  2630  2630 F DEBUG   :
01-18 08:57:38.532  2630  2630 F DEBUG   : backtrace:
01-18 08:57:38.532  2630  2630 F DEBUG   :     #00 pc 000000000004195c  /system/lib64/hw/hwcomposer.rk30board.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #01 pc 000000000005216c  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #02 pc 00000000044db0  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #03 pc 00000000004424c  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #04 pc 0000000000043fb4  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.533  2630  2630 F DEBUG   :     #05 pc 0000000000018270  /system/lib64/libutils.so (_ZN7android6Looper9pollInnerEi+764)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #06 pc 000000000017eb4  /system/lib64/libutils.so (_ZN7android6Looper8pollOnceEiPiS1_PPv+60)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #07 pc 0000000000034400  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.533  2630  2630 F DEBUG   :     #08 pc 00000000000439a0  /system/lib64/libsurfaceflinger.so (_ZN7android14SurfaceFlinger3runEv+20)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #09 pc 0000000000014bc  /system/bin/surfaceflinger
01-18 08:57:38.533  2630  2630 F DEBUG   :     #10 pc 00000000001a594  /system/lib64/libc.so (__libc_init+88)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #11 pc 0000000000011e8  /system/bin/surfaceflinger
```

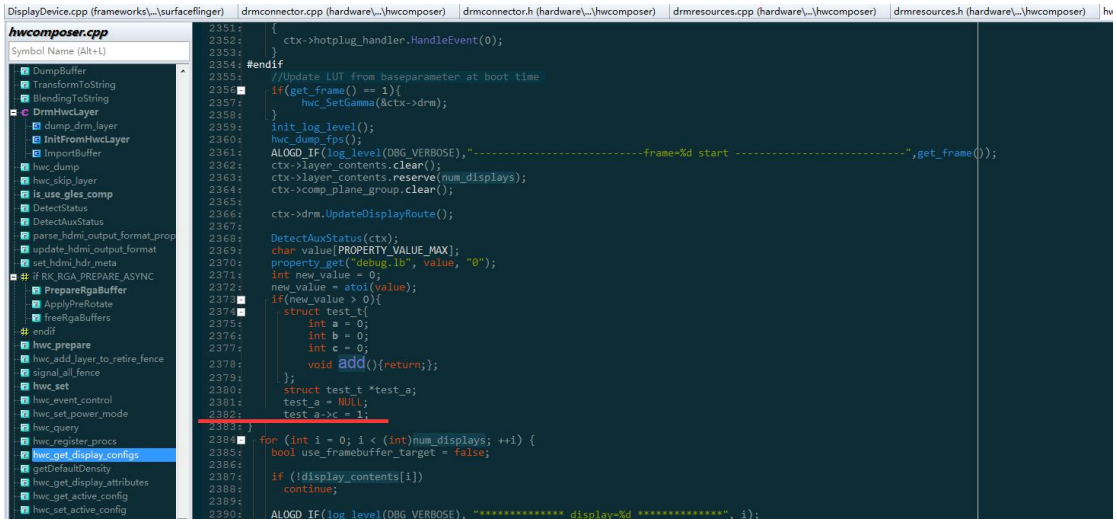
通过 addr2line 命令可反编译到现场出错位置：

```
addr2line -e $OUT/symbols/system/lib64/hw/hwcomposer.rk30board.so 000000000004195c
```

命令输出如下：

```
lb@gpusvr:~/3399/7.1$ addr2line -e /home/lb/3399/7.1/out/target/product/rk3399_mid/symbols/system/lib64/hw/hwcomposer.rk30board.so 000000000004195c
/proc/self/cwd/hardware/rockchip/hwcomposer/hwcomposer.cpp:2382
```

对应代码行如下：

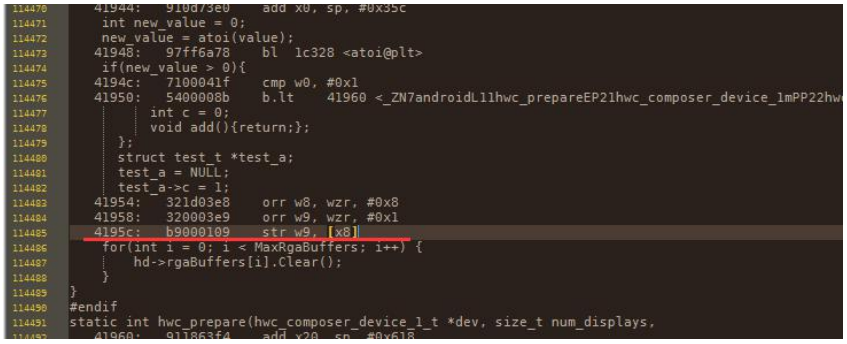


即可定位问题代码行，通过构造的错误复现现场问题，从而定位问题。

我们进一步分析问题，通过以下命令可以反汇编，将对应汇编源码输出：

```
prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin/aarch64-linux-android-objdu
mp -S -D $OUTsymbols/system/lib64/hw/hwcomposer.rk30board.so > hwcomposer.dump
```

在输出文件 hwcomposer.dump 查询 4195c 堆栈打印地址位置，如下图：



即可得对应汇编代码，出现了问题：

```
41958:  320003e9    orr w9, wzr, #0x1  //w9 写入 0x1 立即数
4195c:  b9000109    str w9, [x8]     //将 w9 内容存储到 x8 存储地址
```

通过查询 ARM 汇编命令手册：

```
STR Wt, addr
```

Store Register: stores word from Wt to memory addressed by addr.

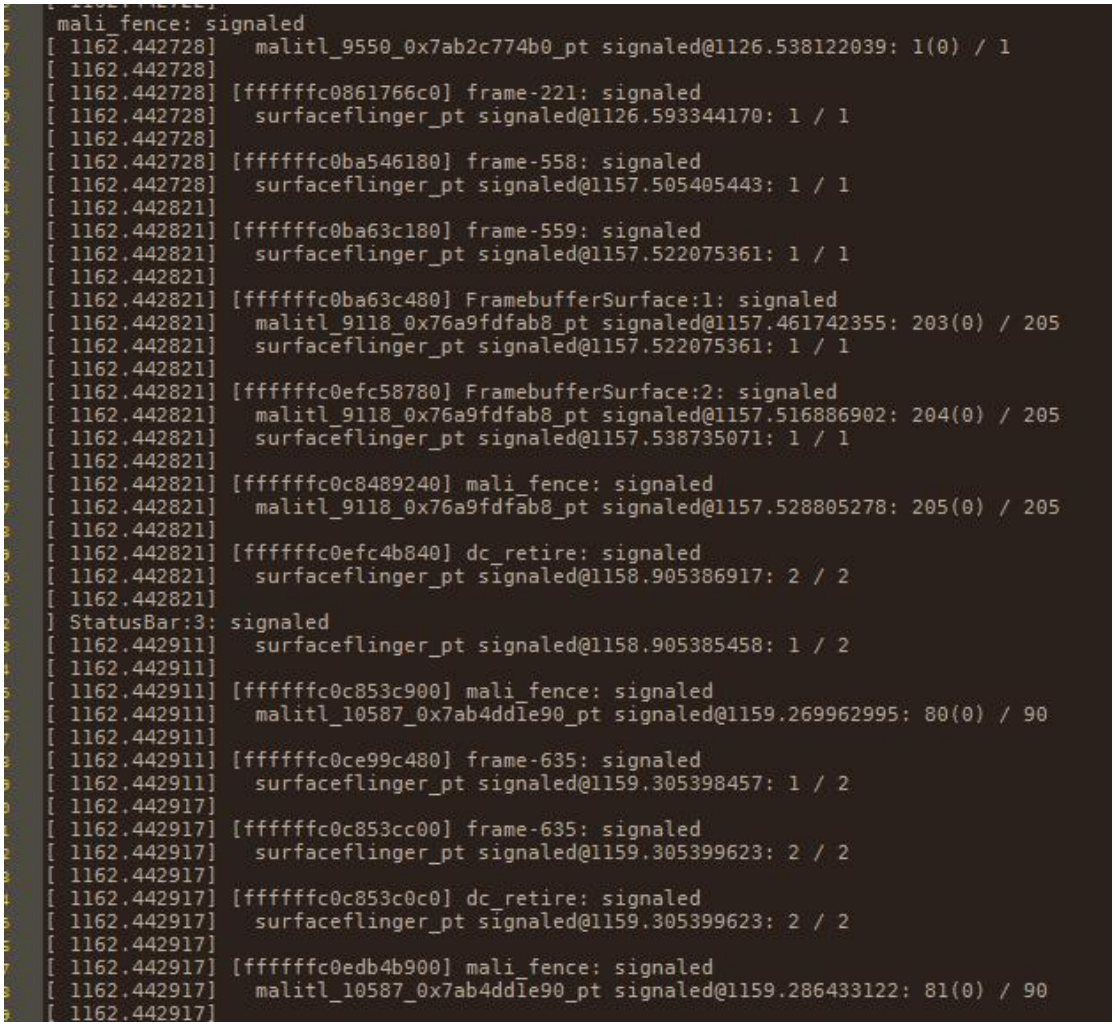
即 str 汇编命令：将来自 Wt 的单词存储到 addr 寻址的内存中，即将 w9 单词存储到 x8 寄存器所指向的内存中，我们需要查看 x8 寄存器所存储内容：

```
--------- beginning of crash
01-18 08:57:38.507  1510  1510 F libc    : Fatal signal 11 (SIGSEGV), code 1, fault addr 0x8 in tid 1510 (surfaceflinger)
--------- beginning of main
01-18 08:57:38.508   216   216 W         : debuggerd: handling request: pid=1510 uid=1000 gid=1003 tid=1510
01-18 08:57:38.523  2630  2630 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
01-18 08:57:38.523  2630  2630 F DEBUG   : Build fingerprint: 'Android/rk3399_all/rk3399_all:7.1.2/NHG47K/zwp06120914:userdebug/test-keys'
01-18 08:57:38.523  2630  2630 F DEBUG   : Revision: '0'
01-18 08:57:38.523  2630  2630 F DEBUG   : ABI: 'arm64'
01-18 08:57:38.524  2630  2630 F DEBUG   : pid: 1510, tid: 1510, name: surfaceflinger  >>> /system/bin/surfaceflinger <<<
01-18 08:57:38.524  2630  2630 F DEBUG   : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x8
01-18 08:57:38.524  2630  2630 F DEBUG   :     x0   0000000000000001  x1   0000000000000000  x2   000000000000000a  x3   0000000000000031
01-18 08:57:38.524  2630  2630 F DEBUG   :     x4   0000000000000062  x5   0000000000000000  x6   0000007fe01806ee  x7   6bfefefefefefefe
01-18 08:57:38.524  2630  2630 F DEBUG   :     x8   0000000000000008  x9   0000000000000001  x10  0000000000000000  x11  0101010101010101
01-18 08:57:38.524  2630  2630 F DEBUG   :     x12  0000000000000008  x13  ffffffffffffffff  x14  ffffffffff000000  x15  ffffffffffffffff
01-18 08:57:38.524  2630  2630 F DEBUG   :     x16  00000070c5c3d3e0  x17  00000070c5bd004c  x18  00000070c16c3480  x19  00000070c5667418
01-18 08:57:38.524  2630  2630 F DEBUG   :     x20  0000000000000001  x21  0000000000000003  x22  00000000ffffffff  x23  00000070c1b975a0
01-18 08:57:38.524  2630  2630 F DEBUG   :     x24  000000000066666f  x25  00000070c5a8d000  x26  00000070c567c920  x27  000000000100ffff
01-18 08:57:38.524  2630  2630 F DEBUG   :     x28  00000070c567c800  x29  0000007fe0180ca0  x30  00000070c5a5994c
01-18 08:57:38.524  2630  2630 F DEBUG   :     sp   0000007fe0180390  pc   00000070c5a5995c  pstate 0000000060000000
01-18 08:57:38.532  2630  2630 F DEBUG   :
01-18 08:57:38.532  2630  2630 F DEBUG   : backtrace:
01-18 08:57:38.532  2630  2630 F DEBUG   :     #00 pc 000000000004195c  /system/lib64/hw/hwcomposer.rk30board.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #01 pc 000000000005216c  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #02 pc 0000000000044db0  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #03 pc 000000000004424c  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.532  2630  2630 F DEBUG   :     #04 pc 0000000000043fb4  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.533  2630  2630 F DEBUG   :     #05 pc 0000000000018270  /system/lib64/libutils.so (_ZN7android6Looper9pollInnerEi+764)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #06 pc 0000000000017eb4  /system/lib64/libutils.so (_ZN7android6Looper8pollOnceEiPiS1_PPv+60)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #07 pc 0000000000034400  /system/lib64/libsurfaceflinger.so
01-18 08:57:38.533  2630  2630 F DEBUG   :     #08 pc 00000000000439a0  /system/lib64/libsurfaceflinger.so (_ZN7android14SurfaceFlinger3runEv+20)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #09 pc 00000000000014bc  /system/bin/surfaceflinger
01-18 08:57:38.533  2630  2630 F DEBUG   :     #10 pc 000000000001a594  /system/lib64/libc.so (__libc_init+88)
01-18 08:57:38.533  2630  2630 F DEBUG   :     #11 pc 00000000000011e8  /system/bin/surfaceflinger
--------- beginning of system
```

x8 寄存器所存储的地址就是 0x8，也就是最终引用地址出错的地址.

1.6.3 Fence Timeout

Fence Timeout 问题 log 如下图：



```
mali_fence: signaled
[ 1162.442728]   malitl_9550_0x7ab2c774b0_pt signaled@1126.538122039: 1(0) / 1
[ 1162.442728]
[ 1162.442728] [ffffffc0861766c0] frame-221: signaled
[ 1162.442728]   surfaceflinger_pt signaled@1126.593344170: 1 / 1
[ 1162.442728]
[ 1162.442728] [ffffffc0ba546180] frame-558: signaled
[ 1162.442728]   surfaceflinger_pt signaled@1157.505405443: 1 / 1
[ 1162.442821]
[ 1162.442821] [ffffffc0ba63c180] frame-559: signaled
[ 1162.442821]   surfaceflinger_pt signaled@1157.522075361: 1 / 1
[ 1162.442821]
[ 1162.442821] [ffffffc0ba63c480] FramebufferSurface:1: signaled
[ 1162.442821]   malitl_9118_0x76a9fdfab8_pt signaled@1157.461742355: 203(0) / 205
[ 1162.442821]   surfaceflinger_pt signaled@1157.522075361: 1 / 1
[ 1162.442821]
[ 1162.442821] [ffffffc0efc58780] FramebufferSurface:2: signaled
[ 1162.442821]   malitl_9118_0x76a9fdfab8_pt signaled@1157.516886902: 204(0) / 205
[ 1162.442821]   surfaceflinger_pt signaled@1157.538735071: 1 / 1
[ 1162.442821]
[ 1162.442821] [ffffffc0c8489240] mali_fence: signaled
[ 1162.442821]   malitl_9118_0x76a9fdfab8_pt signaled@1157.528805278: 205(0) / 205
[ 1162.442821]
[ 1162.442821] [ffffffc0efc4b840] dc_retire: signaled
[ 1162.442821]   surfaceflinger_pt signaled@1158.905386917: 2 / 2
[ 1162.442821]
] StatusBar:3: signaled
[ 1162.442911]   surfaceflinger_pt signaled@1158.905385458: 1 / 2
[ 1162.442911]
[ 1162.442911] [ffffffc0c853c900] mali_fence: signaled
[ 1162.442911]   malitl_10587_0x7ab4ddle90_pt signaled@1159.269962995: 80(0) / 90
[ 1162.442911]
[ 1162.442911] [ffffffc0ce99c480] frame-635: signaled
[ 1162.442911]   surfaceflinger_pt signaled@1159.305398457: 1 / 2
[ 1162.442917]
[ 1162.442917] [ffffffc0c853cc00] frame-635: signaled
[ 1162.442917]   surfaceflinger_pt signaled@1159.305399623: 2 / 2
[ 1162.442917]
[ 1162.442917] [ffffffc0c853c0c0] dc_retire: signaled
[ 1162.442917]   surfaceflinger_pt signaled@1159.305399623: 2 / 2
[ 1162.442917]
[ 1162.442917] [ffffffc0edb4b900] mali_fence: signaled
[ 1162.442917]   malitl_10587_0x7ab4ddle90_pt signaled@1159.286433122: 81(0) / 90
[ 1162.442917]
```

Fence Timeout 问题比较复杂，这边提供检测步骤：

1）确定复现场景

2）按 1.3.2.5 关闭 releaseFence 后，问题是否还存在，若存在则问题不是 HWC 导致，若问题消失，问题与 HWC 创建 releaseFence 相关。

3）将上述复现场景及关闭 releaseFence 实验结果提供到 redmine,指派给显示部门同事继续确认问题。

2 FAQ

2.1 流畅性问题

流畅性可用 FPS 量化，相关的命令可参考 1.3.1.2 命令输出系统当前 FPS 值；

对于帧率不够的问题主要有以下几个原因：

1) 显示策略不合理匹配（后端显示问题，HWC 问题）
2) 场景存在不合理图层（应用端问题）
3) 系统性能不够（提高系统性能，降低场景负载，不合理变频策略）等

对于显示策略而言我们需要将刷新率高，数据量大的图层通过 overlay 输出，overlay 输出具有省带宽，合成耗时短，效率高的优点。这也是 HWC 主要功能之一，所以策略不合理问题均可归结为 HWC 代码逻辑问题，如果有遇到相关问题，可指派 HWC 模块维护者。

2.1.1 视频卡顿

视频卡顿问题归咎问题原因基本就有一个，就是视频层没有通过 Overlay 输出，Overlay 是指通过 Vop 硬件直接输出到 DisplayDevice 上,无需通过预合成处理，也就是 GPU 或 RGA 做预合成后再输出到 Vop，Overlay 输出的优点，省带宽，合成耗时短，效率高。而视频层无法 Overlay 输出的原因有如下几点：

1) 视频输出存在角度，也就是旋转输出；
2) 视频层数过多
3) 视频格式不支持
4) 播放视频场景不支持等

Debug 流程：

```
//通过打印当前显示帧率，判断播放是否异常
setprop debug.sf.fps 1
logcat -c ;logcat | grep mFps

//通过 SurfaceFlinger Services 检查合成策略是否正常
dumpsys SurfaceFlinger

//若不正常，通过打印 HWC log 查看不正常原因
adb shell "setprop sys.hwc.log 511"
adb shell "logcat -c ;logcat" > hwc.log

//尝试添加 HWC patch 修复卡顿问题
```

2.1.1.1 视频卡顿 - 4K hdr or 10bit

Defect #179655：https://redmine.rockchip.com.cn/issues/179655#change-1585243

1. 通过 dumpsys SurfaceFlinger 命令输出，我们获得以下信息：

```
dumpsys SurfaceFlinger
```

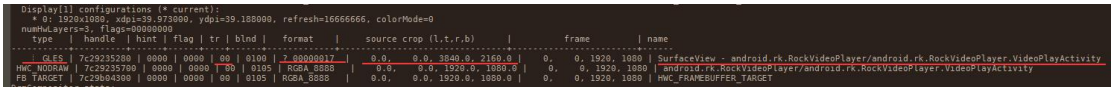
a) SurfaceView 为视频层，数据原大小为 3840x2160；

b) 视频层格式为 0x17，查询后确定为 NV12_10，为 10bit YUV 视频；

c) 合成方式为 GLES，为 GPU 合成输出；

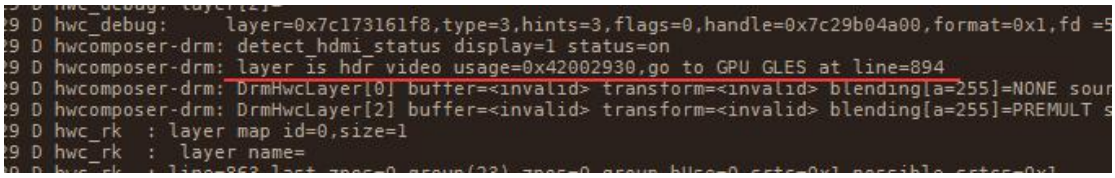
d) TR 为 00，表明视频无旋转；

结论：4k 10bit 视频无旋转播放，采用 GPU 合成输出，存在异常，正常应该是 Overlay 输出，也就是 HWC，所以应该去检查 HWC 代码逻辑回退 GPU 合成原因。



2. 通过检查 HWC log，发现如下打印：

```
adb shell "setprop sys.hwc.log 511"
adb shell "logcat -c ;logcat" > hwc.log
```



发现 HWC 相关逻辑导致该视频层回退 GPU 处理，所以需要移交 HWC 模块负责人处理。

回退 GPU 原因：RK3399 Vop 不支持 HDR 视频 Overlay，所以 HWC 需要回退 GPU，GPU 有 hdr2sdr 实现逻辑，这样才能保证 hdr 视频的效果。

3. 联系客户，客户希望保证帧率达到视频默认帧率，hdr2sdr 效果优先级较低，故需要和客户确认代码版本。

```
adb shell getprop | grep ghwc
```

确认代码版本如下：

[sys.ghwc.commit]: [commit-id:cc0f345]
[sys.ghwc.version]: [0.28-rk3399-MID]

4. HWC 修改相关逻辑，采用直接 Overlay 方式送显视频层。视频卡顿问题解决。

2.1.1.2 视频卡顿 - 4K

Defect #181433: https://redmine.rockchip.com.cn/issues/181433#change-1623747

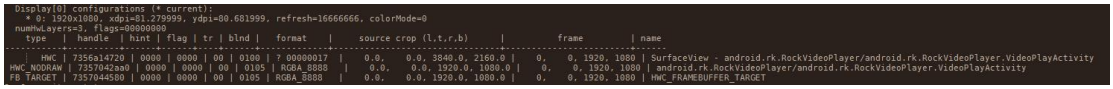
问题描述：

客户播放 4K 视频时，下拉状态栏，发现视频卡顿

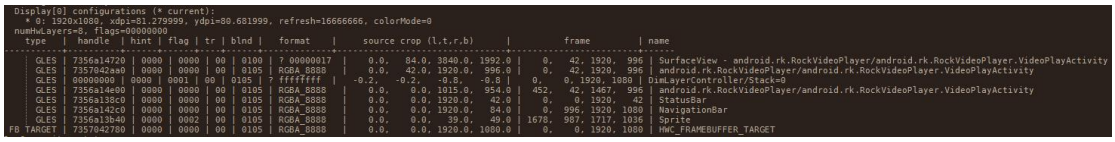
1. 通过对比 dumpsys SurfaceFlinger 命令输出，我们获得以下信息：

```
dumpsys SurfaceFlinger
```

正常播放输出：



卡顿时输出：

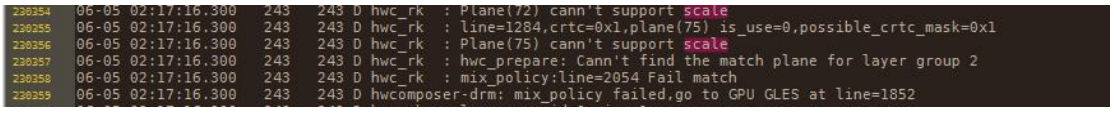


1) 正常播放时系统送显 Layer 有 2 层，且视频层是通过 HWC overlay 送显；
2) 异常播放时系统送显 Layer 有 7 层，并且全部 Layer 均是通过 GPU 合成；
3) 异常播放时考虑到当时的场景，7 层状态为正常；
4) 视频无旋转，格式为可支持 overlay 格式。

结论：首先视频层没有 Overlay，就存在异常，且异常播放场景为 Layer 层数很多的场景，怀疑可能是 Layer 层数过多，导致视频层无法 Overlay。

2. 通过检查 HWC log，发现如下打印：

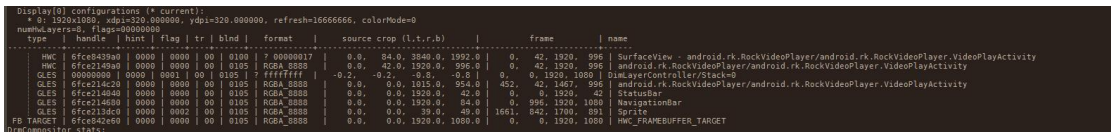
```
adb shell "setprop sys.hwc.log 511"

adb shell "logcat -c ;logcat" > hwc.log
```



发现 HWC 匹配不到合适的策略，导致回退 GPU。这时候就要检查 HWC 匹配策略的逻辑，需要适配该场景。联系 HWC 模块开发者，将上述 log 提供 redmine 以及 HWC version 信息提供至 redmine，供开发者定位问题。

```
adb shell getprop | grep ghwc
```

3. 修改 HWC 策略，使视频层 Overlay，对应 dumpsys SurfaceFlinger 信息如下：



可以看到视频层及一层 UI 合成策略为 vop，则卡顿问题解决。

2.1.2 操作卡顿、延时

首先操作的流畅性体现在两个方面：

1) 帧率平滑；
2) 操作延时；

帧率平滑：要求每帧之间时间间隔保持一致，若不一致则会导致不平滑问题，肉眼可以明显感受到。通常是通过 1.3.4 systrace 分析，也可通过 1.3.1.2 mFps 粗略估计。

操作延时：由于 Android Vsync 工作原理，实际操作一般需要在 3-5 个 Vsync 时间才会输出到屏幕上，如果是 60 帧的画面（对应 Vsync 时间为 16ms），则操作延时在 48 - 80 ms 之间，肉眼很难察觉，但如果是 30 帧的画面（对应 Vsync 时间为 32ms），那么将近 96 - 160ms 的延时，反馈到屏幕上就是严重的滞后现象。

综上，要保证操作流畅，理想帧率需要保持在 60 帧。

影响帧率的原因：

1) 显示策略不合理匹配（后端显示问题，HWC 问题）
2) 场景存在不合理图层（应用端问题）
3) 系统性能不够（提高系统性能，降低场景负载，不合理变频策略）等

2.1.2.1 界面操作卡顿

rk3326_8.1 主界面操作卡顿问题：

1) 卡顿场景打印 SurfaceFlinger 信息，查看当前系统提交 Layer 情况：

```
dumpsys SurfaceFlinger
```

```
Display 0 HWC layers:
----------------------------------------------------------------------------------
 Layer name
           Z |  Comp Type |   Disp Frame (LTRB) |          Source Crop (LTRB)
----------------------------------------------------------------------------------
 com.android.systemui.ImageWallpaper#0
       11000 |      Client |    0    0 1920 1080 |    0.0  420.0 1920.0 1500.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 com.android.launcher3/com.android.launcher3.Launcher#0
       21005 |      Client |    0    0 1920 1080 |    0.0    0.0 1920.0 1080.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 StatusBar#0
      181000 |      Client |    0    0 1920   39 |    0.0    0.0 1920.0   39.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 NavigationBar#0
      231000 |      Client |    0 1002 1920 1080 |    0.0    0.0 1920.0   78.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Display 1 HWC layers:
----------------------------------------------------------------------------------
 Layer name
           Z |  Comp Type |   Disp Frame (LTRB) |          Source Crop (LTRB)
----------------------------------------------------------------------------------
 com.android.systemui.ImageWallpaper#0
       11000 |      Client |    0    0  800 1280 |    0.0  420.0 1920.0 1500.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 com.android.launcher3/com.android.launcher3.Launcher#0
       21005 |      Client |    0    0  800 1280 |    0.0    0.0 1920.0 1080.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 StatusBar#0
      181000 |      Client |  771    0  800 1280 |    0.0    0.0 1920.0   39.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 NavigationBar#0
      231000 |      Client |    0    0   58 1280 |    0.0    0.0 1920.0   78.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

主屏

```
5 Layers
  Layer 0  Composition: Framebuffer  Buffer: 0x6fdf01d080/-1
    Display frame: [0, 0, 1920, 1080]
    Source crop: [0, 420, 1920, 1500]
    Transform: None  Blend mode: None
  Layer 1  Composition: Framebuffer  Buffer: 0x6fdf01cf00/-1
    Display frame: [0, 0, 1920, 1080]
    Source crop: [0, 0, 1920, 1080]
    Transform: None  Blend mode: Premultiplied
  Layer 2  Composition: Framebuffer  Buffer: 0x6fdf01d140/-1
    Display frame: [0, 0, 1920, 39]
    Source crop: [0, 0, 1920, 39]
    Transform: None  Blend mode: Premultiplied
  Layer 3  Composition: Framebuffer  Buffer: 0x6fdf01e1c0/-1
    Display frame: [0, 1002, 1920, 1080]
    Source crop: [0, 0, 1920, 78]
    Transform: None  Blend mode: Premultiplied
  Layer 4  Composition: FramebufferTarget  Buffer: 0x6fdf01c900/-1
    Display frame: [0, 0, 1920, 1080]
    Source crop: [0, 0, 1920, 1080]
    Transform: None  Blend mode: Premultiplied
```

副屏

```
Last requested HWC1 state
  Geometry changed: Y
  5 Layers
    Layer 0  Composition: Framebuffer  Buffer: 0x6fdf01d980/-1
      Display frame: [0, 0, 800, 1280]
      Source crop: [0, 420, 1920, 1500]
      Transform: Rotate90  Blend mode: None
    Layer 1  Composition: Framebuffer  Buffer: 0x6fdf01d380/-1
      Display frame: [0, 0, 800, 1280]
      Source crop: [0, 0, 1920, 1080]
      Transform: Rotate90  Blend mode: Premultiplied
    Layer 2  Composition: Framebuffer  Buffer: 0x6fdf01d200/-1
      Display frame: [771, 0, 800, 1280]
      Source crop: [0, 0, 1920, 39]
      Transform: Rotate90  Blend mode: Premultiplied
    Layer 3  Composition: Framebuffer  Buffer: 0x6fdf01e340/-1
      Display frame: [0, 0, 58, 1280]
      Source crop: [0, 0, 1920, 78]
      Transform: Rotate90  Blend mode: Premultiplied
    Layer 4  Composition: FramebufferTarget  Buffer: 0x6fdf01cc00/-1
      Display frame: [0, 0, 800, 1280]
      Source crop: [0, 0, 800, 1280]
      Transform: None  Blend mode: Premultiplied
```

可知信息如下：

1) 此时显示场景为双屏同显，layer 层级为 4 层，且合成方式为 GPU

2) 副屏存在旋转 90 度行为

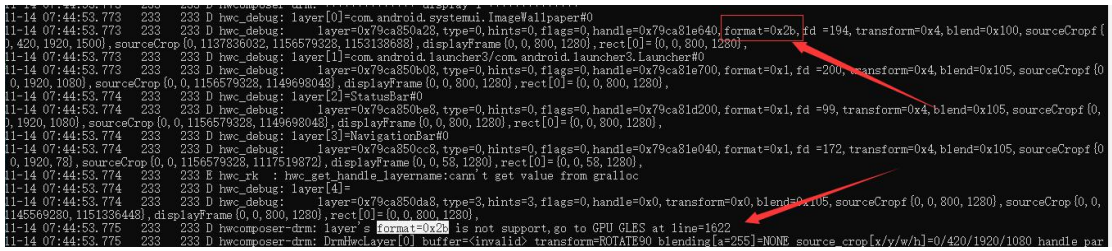
总结如下：

1) 主屏未存在缩放，旋转行为，合成方式为 GPU 存在异常，必然有原因导致无法 overlay，可打印 HWC log 查看。

2) 副屏存在 90 度旋转行为，并且 3326 vop_l 只能输出一层 Layer，故副屏 GPU 合成正常。

2) 打印 HWC log：

```
adb shell "setprop sys.hwc.log 511"

adb shell "logcat -c ;logcat" > hwc.log
```



可以观察到因为 wallpaper 的格式为 2B ，导致主屏所有 Layer 回退 GPU。

经查询 2B 格式为 HAL_PIXEL_FORMAT_RGBA_1010102

故可以确认，导致主界面操作卡顿因为主屏合成方式为 GPU，而导致合成方式为 GPU 的原因是因为 wallpaper 格式为 2B，故只要解决 wallpaper 格式问题即可解决卡顿问题。

3) wallpaper 格式为 2B 问题后续定位为 GOOGLE 的相关提交导致，修复后则帧率稳定，解决卡顿问题。

```
Display 0 HWC layers:
------------------------------------------------------------------------------
 Layer name
           Z |  Comp Type |   Disp Frame (LTRB) |          Source Crop (LTRB)
------------------------------------------------------------------------------
 com.android.systemui.ImageWallpaper#0
       11000 |     Device |    0    0 1200 1920 |    0.0    0.0 1200.0 1920.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 com.android.launcher3/com.android.launcher3.Launcher#0
       21000 |     Device |    0    0 1200 1920 |    0.0    0.0 1200.0 1920.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 StatusBar#0
      181000 |     Device |    0    0 1200   34 |    0.0    0.0 1200.0   34.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 NavigationBar#0
      231000 |     Device |    0 1852 1200 1920 |    0.0    0.0 1200.0   68.0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
  5 Layers
    Layer 0  Composition: Overlay  Buffer: 0x717ce34cc0/-1
      Display frame: [0, 0, 1200, 1920]
      Source crop: [0, 0, 1200, 1920]
      Transform: None  Blend mode: None
    Layer 1  Composition: Overlay  Buffer: 0x717d434c80/-1
      Display frame: [0, 0, 1200, 1920]
      Source crop: [0, 0, 1200, 1920]
      Transform: None  Blend mode: Premultiplied
    Layer 2  Composition: Overlay  Buffer: 0x717d435040/-1
      Display frame: [0, 0, 1200, 34]
      Source crop: [0, 0, 1200, 34]
      Transform: None  Blend mode: Premultiplied
    Layer 3  Composition: Overlay  Buffer: 0x717d433cc0/-1
      Display frame: [0, 1852, 1200, 1920]
      Source crop: [0, 0, 1200, 68]
      Transform: None  Blend mode: Premultiplied
    Layer 4  Composition: FramebufferTarget  Buffer: 0x717d433f00/-1
      Display frame: [0, 0, 1200, 1920]
      Source crop: [0, 0, 1200, 1920]
      Transform: None  Blend mode: Premultiplied
```

可以看到修复格式为 2B 的问题后，HWC 显示策略为 Overlay，则帧率提高。

2.1.2.2 手写痕迹不跟手

2.2 显示问题

2.2.1 显示错误

Android 显示问题涉及组件众多，主要有 GPU，VOP，RGA，DDR 等，所以造成画面显示错误的原因也很多，以下整理出显示错误的大致调试思路：

1）观察出错现象：许多显示错误均可通过现象大致估计出错的代码流程位置，比如显示格式出错（afbc 显示出错），宽高配置错，图像层级结构错等

2）关闭 HWC 查看错误还存在：通过 1.3.2.2 章节提供方法关闭 HWC，所有的合成方式通过 GPU 合成，若关闭 HWC 问题消失则说明问题很有可能出现在 HWC 代码流程中。关闭 HWC 主要是跳过 HWC 策略匹配逻辑，所有 Layer 通过 GPU 合成输出，因为 GPU 合成极少场景出现错误，所以可以通过此方法初略估计错误位置。

3）打印图像源数据：1.3.2.2 章节提供方法可以将源数据以 bin 文件形式 dump 出来，可通过 7yuv 工具查看源数据是否有错误。此方法可以初略收敛出错的位置，若打印出来的源数据本身就是出错的，那么出错的位置应该在打印流程之前（也就是前端渲染出错）；若打印出的源数据是对的，显示却错了，那么出错的位置就在后端显示，有可能是 HWC 的策略匹配，也有可能是 VOP 驱动出错。

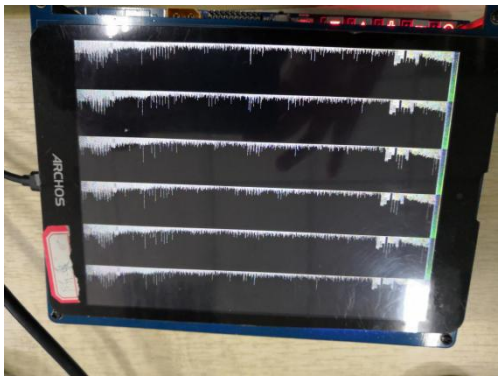
4）打印 HWC log 查看：通过 1.3.2.3 命令可以打印 HWC log,通过 HWC log 可以查看到每一层 Layer 的匹配规则，查看有可能出错的匹配逻辑。

5）modetest 配置：若上述情况均正常，那么有可能就是驱动显示的问题，可以通过 modetest 直接配置驱动输出，若输出异常则就是驱动有问题，若正常，还需要继续收敛问题。

2.2.1.1 花屏

AFBC 显示花屏：

现象：



这种现象就是 AFBC 格式的图像通过 RGBA 格式显示出来的效果，造成这种显示效果原因有：

1. VOP 不支持 AFBC 解析
2. HWC 版本不支持配置 AFBC 格式

调试与解决方法：

1. 关闭 AFBC 编码：采用 1.3.3.1 Disable afbdc 方法关闭 AFBC 编码
2. 改用 VOP_B 显示：VOP 分为两种类型，VOP_B 与 VOP_L,通常 VOP_B 支持 AFBC 编码，VOP_L 不支持。若用 VOP_L 输出 AFBC 图像，通常会在串口打印

```
[drm:vop_afbdc_atomic_check] *ERROR* not support afbdc
```

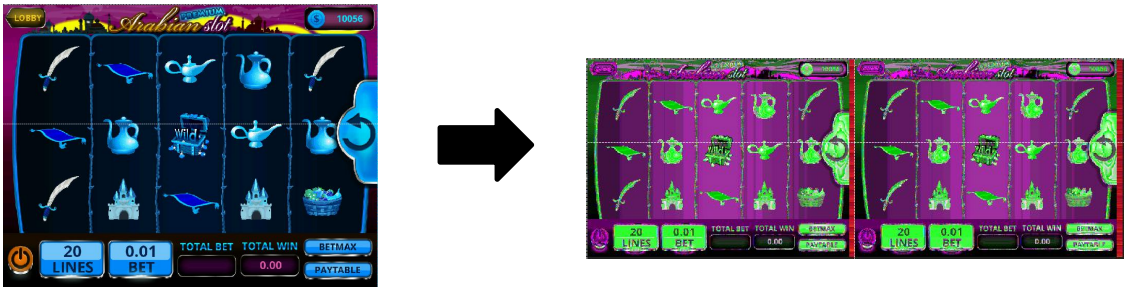
可查看串口信息确认问题。

3. 更新 HWC 版本

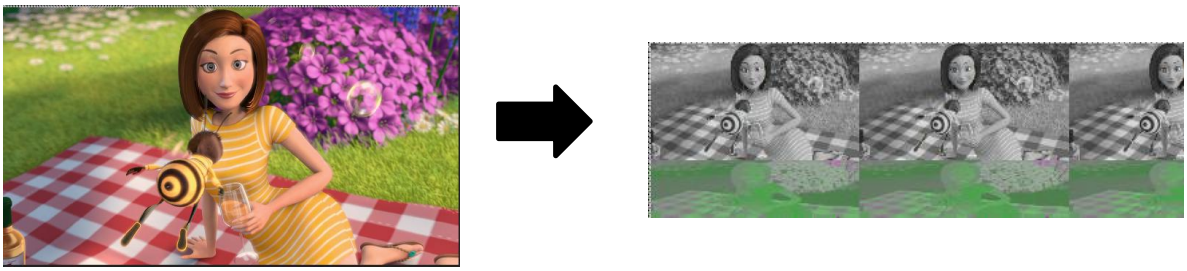
格式出错导致花屏：

现象可用 7yuv 软件模拟，从现象上就可定位为格式出错，比如：

RGB565 通过 RGBA 8888 显示：



NV12 通过 RGBA8888 显示：



宽高配置导致出错：

现象可用 7yuv 软件模拟，从现象上就可定位为宽高配置出错，比如：

1280x720 NV12

1282x720 NV12

1278x720 NV12

如果是格式或者宽高出错，可按显示错误调试流程确定问题：

1） 关闭 HWC 查看错误还存在

2） 打印图像源数据查看是否存在异常

3） 打印 HWC log 查看显示是否存在异常

4） modetest 配置屏幕输出查看显示是否正确

2.2.1.2 闪屏、黑屏

闪屏问题检查思路：

1）**确认闪屏的内容：**可通过手机慢镜头模式录制闪屏现象，后通过 qq 影音的单帧播放模式，确定闪屏内容。

a) 如果闪屏内容为花屏，可根据 2.2.1.1 花屏的调试步骤继续确定问题。

b) 如果闪屏内容不为花屏：图层层级关系错（底层内容被显示出来）、黑屏、绿屏等，就需要查看具体场景。

2）**关闭 HWC 查看闪屏还存在：**利用 1.3.2.2 命令关闭 HWC

3）**打印 HWC log 查看：**利用 1.3.2.3 命令抓打印 HWC log

2.2.1.3 画面卡住

画面卡住问题通常伴随着 Fence Timeout、ANR 等问题，画面卡住最关键的问题是需要找到复现的场景。

调试步骤：

1） 确定复现场景：场景的确定有助于问题的分析与解决，并且再提供补丁后也可验证是否解决问题，所以确定复现的场景非常重要。

2） 分析相关日志：日志保存着第一现场，故需要具备分析相关日志的能力。画面卡住涉及日志有：traces.txt 文件、Fence Timeout 日志等。

3） 定位问题，解决问题。

范例说明：https://redmine.rockchip.com.cn/issues/177719#change-1654879 monkey + HDMI 插拔拷机测试 小概率引起系统卡死

1）**确定复现场景：**客户发现 monkey + HDMI 插拔拷机场景小概率出现系统卡死，无响应问题。**问题场景已确定。**但由于复现概率小，log 分析尤其重要。

2）**分析相关日志：**

logcat 日志：

```
2256 32256 I DJILocationManager: Attempting to get last known location from default providers...
2253  2253 I DualScreen: onKeyUp->keyCode:82
2256 32364 E ContextUtil: dji.go.v4 32256
2256 32364 E DJIStreamLocalSocket: ExitSocket =  ContextUtil.isAppBroughtToBackground()
2256 32262 I art     : Wrote stack traces to '/data/anr/traces.txt'
 547   585 E ActivityManager: ANR in com.android.wallpapercropper (com.android.wallpapercropper/.WallpaperCropActivity)
 547   585 E ActivityManager: PID: 2253
 547   585 E ActivityManager: Reason: Input dispatching timed out (Waiting to send key event because the focused window has not fi
 547   585 E ActivityManager: Load: 4.0 / 5.44 / 5.0
 547   585 E ActivityManager: CPU usage from 81995ms to 0ms ago (2018-06-02 00:29:13.524 to 2018-06-02 00:30:35.520):
 547   585 E ActivityManager:   33% 225/surfaceflinger: 14% user + 19% kernel / faults: 4446 minor
 547   585 E ActivityManager:   21% 547/system_server: 15% user + 5.3% kernel / faults: 24529 minor
 547   585 E ActivityManager:   7.6% 32587/acr.browser.barebones: 5.6% user + 1.9% kernel / faults: 17133 minor
 547   585 E ActivityManager:   5.3% 32256/dji.go.v4: 4.3% user + 1% kernel / faults: 10955 minor
 547   585 E ActivityManager:   4.9% 404/com.lenovo.anyshare: 3.4% user + 1.4% kernel / faults: 12713 minor
 547   585 E ActivityManager:   4.8% 721/com.android.gallery3d: 3.3% user + 1.5% kernel / faults: 8395 minor
 547   585 E ActivityManager:   4.1% 1523/adbd: 1% user + 3% kernel / faults: 2369 minor
 547   585 E ActivityManager:   3.7% 32691/com.android.camera2: 2.5% user + 1.1% kernel / faults: 5253 minor
 547   585 E ActivityManager:   3.3% 200/logd: 1.4% user + 1.8% kernel / faults: 29 minor
 547   585 E ActivityManager:   3.1% 734/com.android.systemui: 2.1% user + 1% kernel / faults: 5580 minor
 547   585 E ActivityManager:   2.1% 29422/com.android.settings: 1.5% user + 0.5% kernel / faults: 9055 minor
 547   585 E ActivityManager:   1.7% 296/djilink: 0.4% user + 1.2% kernel / faults: 152 minor
 547   585 E ActivityManager:   1.7% 1598/com.android.commands.monkey: 1.3% user + 0.3% kernel / faults: 466 minor
 547   585 E ActivityManager:   1.6% 32297/dji.go.v4: 0.4% user + 1.2% kernel
 547   585 E ActivityManager:   1.2% 368/logcatext: 0.6% user + 0.5% kernel
 547   585 E ActivityManager:   1% 1527/irq/232-dwc3: 0% user + 1% kernel
 547   585 E ActivityManager:   0.8% 32544/com.android.launcher3: 0.6% user + 0.2% kernel / faults: 1355 minor
 547   585 E ActivityManager:   0.7% 327/logcat: 0.4% user + 0.3% kernel
```

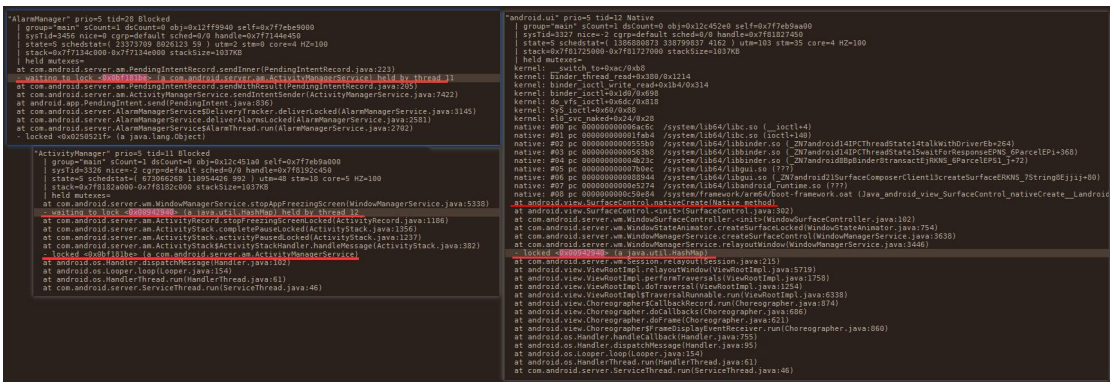
根据 logcat 日志内容可得信息：

a) com.android.wallpapercropper 输入事件超时，引发 ANR 事件

b) ANR 发生前 82s 内 surfaceflinger CPU 占用率为 33%，system_server 为 21%

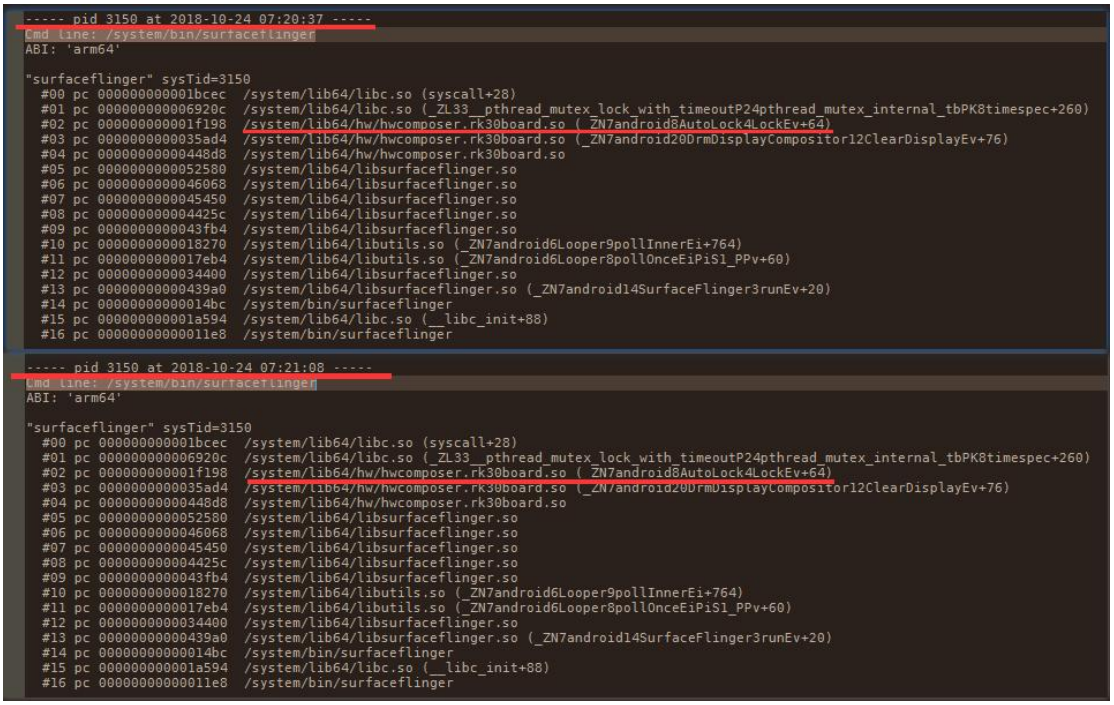
c) 输出 traces.txt 文件到 /data/anr/traces.txt

traces.txt 日志：



由该部分 log 可知：android.ui 在等待 Native method 方法的返回，而 android.view.SurfaceControl.nativeCreate 方法可知 surfaceflinger serverces 未在指定时间内处理该请求，引起 ANR。因此，需要去查看 surfaceflinger 堆栈信息。

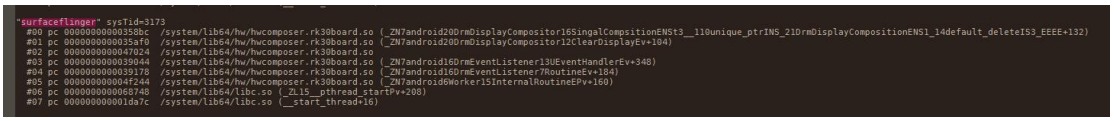
surfaceflinger 堆栈信息：



```
----- pid 3150 at 2018-10-24 07:20:37 -----
Cmd line: /system/bin/surfaceflinger
ABI: 'arm64'

"surfaceflinger" sysTid=3150
  #00 pc 00000000001bcec  /system/lib64/libc.so (syscall+28)
  #01 pc 000000000006920c  /system/lib64/libc.so (_ZL33__pthread_mutex_lock_with_timeoutP24pthread_mutex_internal_tbPK8timespec+260)
  #02 pc 000000000001f198  /system/lib64/hw/hwcomposer.rk30board.so ( ZN7android8AutoLock4LockEv+64)
  #03 pc 0000000000035ad4  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android20DrmDisplayCompositor12ClearDisplayEv+76)
  #04 pc 00000000000448d8  /system/lib64/hw/hwcomposer.rk30board.so
  #05 pc 0000000000052580  /system/lib64/libsurfaceflinger.so
  #06 pc 0000000000046068  /system/lib64/libsurfaceflinger.so
  #07 pc 0000000000045450  /system/lib64/libsurfaceflinger.so
  #08 pc 000000000004425c  /system/lib64/libsurfaceflinger.so
  #09 pc 0000000000043fb4  /system/lib64/libsurfaceflinger.so
  #10 pc 0000000000018270  /system/lib64/libutils.so (_ZN7android6Looper9pollInnerEi+764)
  #11 pc 0000000000017eb4  /system/lib64/libutils.so (_ZN7android6Looper8pollOnceEiPiS1_PPv+60)
  #12 pc 0000000000034400  /system/lib64/libsurfaceflinger.so
  #13 pc 00000000000439a0  /system/lib64/libsurfaceflinger.so (_ZN7android14SurfaceFlinger3runEv+20)
  #14 pc 000000000000014bc  /system/bin/surfaceflinger
  #15 pc 000000000001a594  /system/lib64/libc.so (__libc_init+88)
  #16 pc 00000000000011e8  /system/bin/surfaceflinger

----- pid 3150 at 2018-10-24 07:21:08 -----
Cmd line: /system/bin/surfaceflinger
ABI: 'arm64'

"surfaceflinger" sysTid=3150
  #00 pc 00000000001bcec  /system/lib64/libc.so (syscall+28)
  #01 pc 000000000006920c  /system/lib64/libc.so (_ZL33__pthread_mutex_lock_with_timeoutP24pthread_mutex_internal_tbPK8timespec+260)
  #02 pc 000000000001f198  /system/lib64/hw/hwcomposer.rk30board.so ( ZN7android8AutoLock4LockEv+64)
  #03 pc 0000000000035ad4  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android20DrmDisplayCompositor12ClearDisplayEv+76)
  #04 pc 00000000000448d8  /system/lib64/hw/hwcomposer.rk30board.so
  #05 pc 0000000000052580  /system/lib64/libsurfaceflinger.so
  #06 pc 0000000000046068  /system/lib64/libsurfaceflinger.so
  #07 pc 0000000000045450  /system/lib64/libsurfaceflinger.so
  #08 pc 000000000004425c  /system/lib64/libsurfaceflinger.so
  #09 pc 0000000000043fb4  /system/lib64/libsurfaceflinger.so
  #10 pc 0000000000018270  /system/lib64/libutils.so (_ZN7android6Looper9pollInnerEi+764)
  #11 pc 0000000000017eb4  /system/lib64/libutils.so (_ZN7android6Looper8pollOnceEiPiS1_PPv+60)
  #12 pc 0000000000034400  /system/lib64/libsurfaceflinger.so
  #13 pc 00000000000439a0  /system/lib64/libsurfaceflinger.so (_ZN7android14SurfaceFlinger3runEv+20)
  #14 pc 000000000000014bc  /system/bin/surfaceflinger
  #15 pc 000000000001a594  /system/lib64/libc.so (__libc_init+88)
  #16 pc 00000000000011e8  /system/bin/surfaceflinger
```

由该 log 可知，30s 内 surfaceflinger 堆栈信息均为改变，并且堆栈反编译发现 hwcomposer 在等待获取一个 Auto lock，进入死锁状态。

在观察 surfaceflinger 其他线程堆栈信息，发现如下堆栈：



```
"surfaceflinger" sysTid=3173
  #00 pc 00000000000358bc  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android20DrmDisplayCompositor16SingalCompsitionENSt3__110unique_ptrINS_21DrmDisplayCompositionENS1_14default_deleteIS3_EEEE+132)
  #01 pc 0000000000035af0  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android20DrmDisplayCompositor12ClearDisplayEv+104)
  #02 pc 0000000000047024  /system/lib64/hw/hwcomposer.rk30board.so
  #03 pc 0000000000039044  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android16DrmEventListener13UEventHandlerEv+348)
  #04 pc 0000000000039178  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android16DrmEventListener7RoutineEv+184)
  #05 pc 000000000004f244  /system/lib64/hw/hwcomposer.rk30board.so (_ZN7android6Worker15InternalRoutineEPv+160)
  #06 pc 0000000000068748  /system/lib64/libc.so (_ZL15__pthread_startPv+200)
  #07 pc 000000000001da7c  /system/lib64/libc.so (__start_thread+16)
```

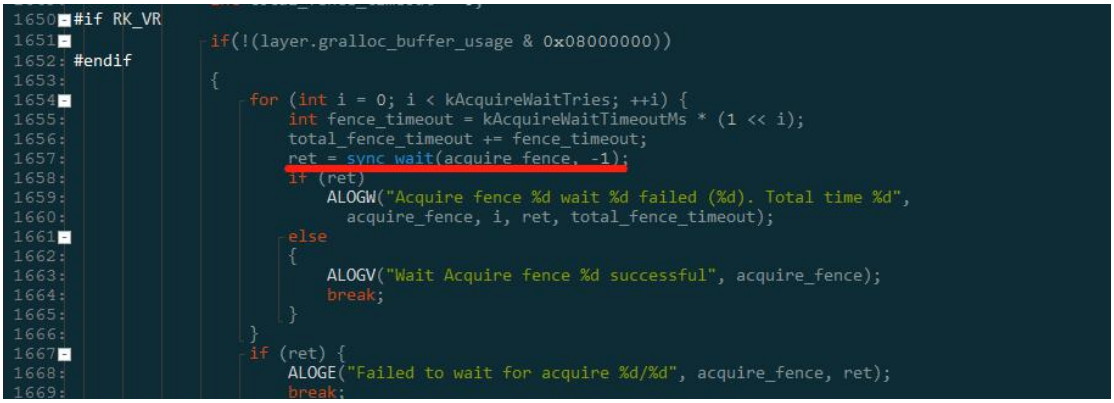
对应反编译源代码如下：

```
683: void DrmDisplayCompositor::ClearDisplay() {
684:   AutoLock lock(&lock_, "compositor");
685:   int ret = lock.Lock();
686:   if (ret)
687:     return;
688:
689:   SingalCompsition(std::move(active_composition_));
690:
691:   //Singal the remainder fences in composite queue.
692:   while(!composite_queue_.empty())
693:   {
694:     std::unique_ptr<DrmDisplayComposition> remain_composition(
695:       std::move(composite_queue_.front()));
696:
697:     if(remain_composition)
698:       ALOGD_IF(log_level(DBG_DEBUG),"ClearDisplay: composite_queue_ size=%zu frame_no=%" PRIu64 "",composite_queue_.size(), remain_composition->frame_no());
699:
700:     SingalCompsition(std::move(remain_composition));
701:     composite_queue_.pop();
702:     pthread_cond_signal(&composite_queue_cond_);
703:   }
704: } « end ClearDisplay »
705:
```

自动锁获取

堆栈打印点

存在死锁现象，继而跟进代码，阻塞调用如下：

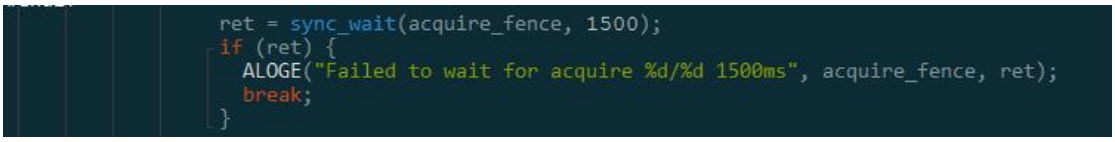


```
#if RK_VR
if(!(layer.gralloc_buffer_usage & 0x08000000))
#endif
{
    for (int i = 0; i < kAcquireWaitTries; ++i) {
        int fence_timeout = kAcquireWaitTimeoutMs * (1 << i);
        total_fence_timeout += fence_timeout;
        ret = sync_wait(acquire_fence, -1);
        if (ret)
            ALOGW("Acquire fence %d wait %d failed (%d). Total time %d",
              acquire_fence, i, ret, total_fence_timeout);
        else
        {
            ALOGV("Wait Acquire fence %d successful", acquire_fence);
            break;
        }
    }
    if (ret) {
        ALOGE("Failed to wait for acquire %d/%d", acquire_fence, ret);
        break;
```

这是一个 wait Fence 函数，并且是无限制等待，只要 acquireFence 出现异常无法返回，该线程阻塞，锁无法释放，导致 surfaceflinger 进程阻塞，导致 ANR，故问题定位为 hwc 线程死锁。

3）定位问题，解决问题。

死锁问题可通过流程优化与超时机制解决，故此处问题采用超时机制解决该问题。



```
ret = sync_wait(acquire_fence, 1500);
if (ret) {
  ALOGE("Failed to wait for acquire %d/%d 1500ms", acquire_fence, ret);
  break;
}
```

设置超时时间 1.5s，若达到 1.5s 释放该锁，则问题解决。

2.2.1.4 Crash 重启问题

显示框架 crash 重启问题常见原因：空指针异常，野指针异常等，通常就是分析 crash 日志，加上一些分析手段即可完成问题收敛。

调试步骤：

1） 确定复现场景：场景的确定有助于问题的分析与解决，并且再提供补丁后也可验证是否解决问题，所以确定复现的场景非常重要。

2） 分析相关日志：日志保存着第一现场，故需要具备分析相关日志的能力。画面卡住涉及日志有：crash 日志等。

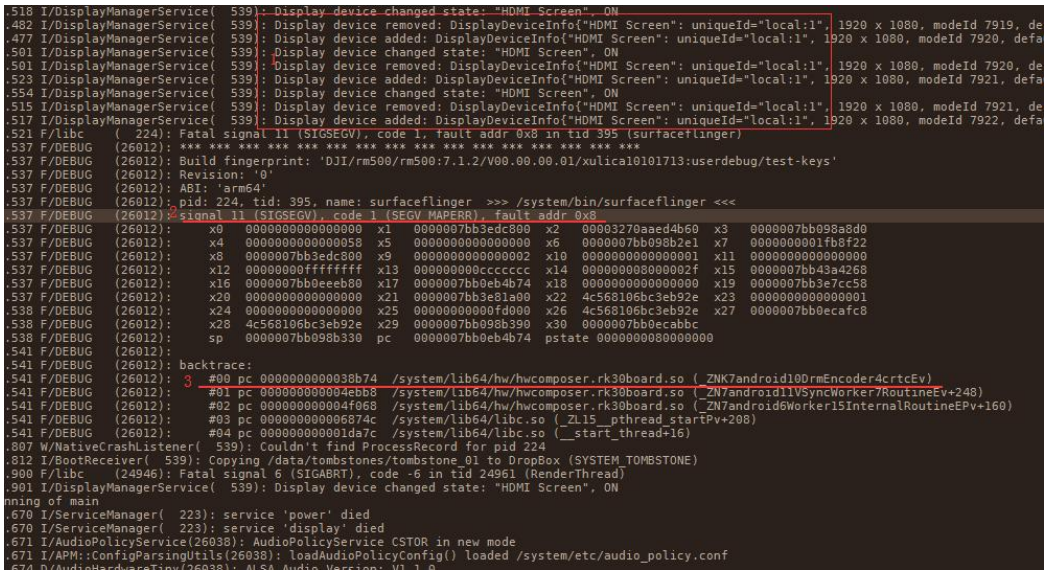
3） 定位问题，解决问题。

范例说明：https://redmine.rockchip.com.cn/issues/177719#change-1654879 monkey + HDMI 插拔拷机测试 大概率引起 framework crash 问题。

3）**确定复现场景：**客户发现 monkey + HDMI 插拔拷机场景会出现 framework crash 重启问题，并且复现概率为 80%以上。**问题场景已确定。**

4）**分析相关日志：**

logcat 日志：



根据日志内容可得信息：

a) 方框 1：问题场景确实存在 Display Devices 添加与删除工作，与 HDMI 插拔拷机场景相符。

b) 划线 2：SIGSEGV surfaceflinger 进程执行了一次无效引用，引用地址为 0x8。

c) 划线 3：现场堆栈打印，可通过反编译定位出错代码行。

重点分析 b,c 两点：

b) 引用的错误地址为 0x8，在一个对象为空指针的情况下，指向这个空对象的成员或成员函数则会引用诸如 0x8 的空指针异常，故 0x8 等同于 0x0，也是一个空指针异常。

可通过如下代码验证，也会导致 fault addr 0x8 错误：

```
struct test_t{
    int a = 0;
    int b = 0;
    int c = 0;
    void add(){return;};
};
struct test_t *test_a;
test_a = NULL;
test_a->c = 1;
```

c) 可通过 addr2line 命令反编译找到源代码代码行：

```
addr2line -e $OUT/symbols/system/lib64/hw/hwcomposer.rk30board.so 38b74
```

可定位代码行为如下图：

```
659: DrmCrtc *DrmResources::GetCrtcFromConnector(DrmConnector *conn) const {
660:     if (conn->encoder())
661:       return conn->encoder()->crtc();
662:     return NULL;
663: }
```

分析代码逻辑，可知 if 空指针判断后，存在 conn->encoder() 返回值被改写的场景，根据拷机场景为 HDMI 插拔，故有可能在空指针判断后，因为热插拔事件导致 conn->encoder()被设置为 NULL，继而去引用其成员函数 crtc()导致空指针异常。

a. **定位问题，解决问题：**将该行逻辑修改为如下所示，最终解决该问题：

```
DrmCrtc *DrmResources::GetCrtcFromConnector(DrmConnector *conn) const {
  DrmEncoder *encoder = conn->encoder();
  if (NULL != encoder)
    return encoder->crtc();
  else
    return NULL;
}
```

2.3 主副屏相关

主副屏相关属性介绍：

```
persist.sys.framebuffer.main    //主屏 UI 画布尺寸
persist.sys.framebuffer.aux     //副屏 UI 画布尺寸
persist.sys.resolution.main     //主屏分辨率
persist.sys.resolution.aux      //副屏分辨率
sys.hwc.device.primary          //主屏设备类型
sys.hwc.device.extend           //副屏设备类型
sys.hwc.device.main             //主屏当前设置设备
sys.hwc.device.aux              //副屏当前设置设备
sys.display.timeline            //修改生效标志
```

2.3.1.1 主副屏设置

涉及属性：

```
sys.hwc.device.primary          //主屏设备类型，用户设置
sys.hwc.device.extend           //副屏设备类型，用户设置
sys.hwc.device.main             //主屏当前设置设备，系统设置
sys.hwc.device.aux              //副屏当前设置设备，系统设置
```

其中，用户设置属性类型设置格式为：

```
//在 system/build.prop 加入格式字段，主副屏设备可根据产品自行设置，即可配置主
//副屏，对于 HDMI-A-1,HDMI-A-2 问题，目前 HWC 最新代码支持对应判断，如果不支持
//请升级 hwc 代码
//  sys.hwc.device.xxx=xx,xx
sys.hwc.device.primary=eDP,LVDS,VGA
sys.hwc.device.extend=HDMI-A-1,HDMI-A-2
```

当前系统作为主副屏显示设备可查询的属性如下：

```
sys.hwc.device.main             //主屏当前设置设备，系统设置
sys.hwc.device.aux              //副屏当前设置设备，系统设置

//可通过如下命令查询
getprop sys.hwc.device.main     //查询主屏当前设置显示设备
getprop sys.hwc.device.aux      //查询副屏当前设置显示设备
```

以下 Log 输出表明当前系统连接主屏为 eDP，副屏为 HDMI-A-1

```
130|rk3399_all:/ # getprop | grep sys.hwc.device
[sys.hwc.device.aux]: [HDMI-A-1]
[sys.hwc.device.main]: [eDP]
```

如果我们需要修改主副屏逻辑关系，那么以下命令就可以设置：

```
setprop sys.hwc.device.primary HDMI-A-1   //设置 HDMI-A-1 为主屏
setprop sys.hwc.device.extend eDP         //设置 eDP 为副屏
stop;start   //设置完需要 Android 重启生效,需要 reboot 生效需要写入 build.prop 中
```

设置完后，可通过查看相关属性确认修改，命令与输出如下：

```
getprop | grep sys.hwc.device
```

```
rk3399_all:/ # getprop | grep sys.hwc.device
[sys.hwc.device.aux]: [eDP]
[sys.hwc.device.extend]: [eDP]
[sys.hwc.device.main]: [HDMI-A-1]
[sys.hwc.device.primary]: [HDMI-A-1]
```

可确认，主副屏关系已经按预设值进行设置。

对于用户没有指定主屏的设备状态，系统有默认设置，即：

```
//默认主屏设备类型
DRM_MODE_CONNECTOR_LVDS      LVDS
DRM_MODE_CONNECTOR_eDP       eDP
DRM_MODE_CONNECTOR_DSI       DSI
DRM_MODE_CONNECTOR_VIRTUAL   Virtual
DRM_MODE_CONNECTOR_TV        TV
DRM_MODE_CONNECTOR_DPI       DPI
```

若主屏设置属性未指定，则默认在以上设备类型内的设备为主屏，不在该类型范围内的为副屏。

建议：若产品只有一个显示设备，建议将主屏属性设置为该显示设备类型，副屏属性设置为 NULL，以 HDMI 设备为例

```
sys.hwc.device.primary=HDMI-A-1   //设置 HDMI-A-1 为主屏
sys.hwc.device.extend=NULL        //副屏设备为 NULL
```

2.3.1.2 主副屏开机无显示

主副屏开机无显示问题，一般原因如下：

1. **显示设备物理未正确连接**
2. **显示设备 driver 存在问题**
3. **显示设备框架注册流程存在问题**
4. **显示设备显示配置存在问题**

1.物理未正确连接：

即设备未上电，连接线损坏，接口接触不良等硬件问题，不讨论。

2.显示设备 driver 存在问题：

可通过 1.2.6 modetest 章节配置屏显，若有显示正常，则 driver 正常；若无显示，应先从 driver 查询。屏参数配置，屏初始化等方面进行排查。

3.显示设备框架注册流程存在问题：

a) 可通过 2.3.1.1 主副屏设置 章节介绍命令进行查询，查询当前显示设备是否正确注册。

b) 可通过 1.1.1 Dumpsys SurfaceFlinger 章节命令查询上层 surfaceflinger 是否有对应设备存在。并且相关参数是否正确。若均正常，则有可能是显示配置存在问题。

4.显示设备显示配置存在问题:

可通过 2.2.1 显示错误章节排查问题，因为若无显示，且设备注册连接均正常，就可以优先检查是否显示错误的问题。

2.3.1.3 主副屏分辨率设置问题

DRM_HWC 分辨率设置相关联的属性如下：

```
persist.sys.resolution.main     //主屏分辨率
persist.sys.resolution.aux      //副屏分辨率
sys.display.timeline            //修改生效标志
```

主屏分辨率设置流程：

设置 persist.sys.resolution.main 属性 -> 设置 sys.display.timeline

sys.display.timeline 作为属性生效的标志，每次修改属性之后都需要将该属性值+1 确保更新

检查问题步骤：1.设置分辨率属性；2.将 sys.display.timeline 属性值+1

范例：

Defect #183018：AM40 調節分辨率沒有反應(Main Board/Debug Board)

https://redmine.rockchip.com.cn/issues/183018#change-1674777

该问题为更新 hwc 版本后出现，最终定位问题为 setting apk 在修改分辨率属性后，没有将 sys.display.timeline 值+1 导致。

sys.display.timeline 属性是后期版本加入，可能早期的 apk 没有设置该属性，故需要联系 apk 负责人进行升级。