RK Rockit MPI FAQ

文件标识：RK-SYS1-MPI-FAQ

发布版本：V0.5.0

日　　期：2021.10

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

本文档主要描述多媒体处理应用开发过程中遇到的常见问题及解决方法。

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

版本号	作者	修改日期	修改说明
V0.1.0	方兴文	2021-09-08	初始版本
V0.2.0	李耀辉	2021-10-09	添加内存优化
V0.3.0	李鑫煌	2021-10-11	增加VI/VENC FAQ
V0.4.0	黄晓明	2021-10-11	添加VGS/TDE FAQ
V0.5.0	周弟东	2021-10-11	添加AUDIO FAQ

1. 目录

2. 1. SYS

2.1 日志信息

2.1.1 查看MPI日志信息

【现象描述】

需要查看日志和调整 log 日志的等级。

【分析解决】

目前日志分为 6 个等级，默认设置为等级 5。等级设置的越高，表示记录到日志中的信息量就越多，当等级为 6 时，此时的信息量非常庞大，会降低系统的整体性能。因此，通常情况下，推荐设置为等级 4 - 5。

目前仅支持设置所有模块日志，可以通过以下几种方式修改日志等级：

- export rt_log_level=x (x为日志等级，值范围1～6），需设置之后启动用户进程方可生效。
- 通过 **RK_MPI_LOG_SetLevelConf** 接口将 **s32Level** 设置为指定等级。

2.2 内存使用

2.2.1 打开CMA配置

【现象描述】

默认情况下，我们使用 IOMMU 非连续物理内存，但在某些特定场景下，需要使用 CMA 物理连续内存时，以能够获取物理地址进行操作。

【分析解决】

- 内核驱动修改，配置 CMA 预留内存大小

  以RK356X驱动为例，在DTS中预留CMA内存，大小根据实际需要调整，单位Byte。如果没有添加inactive，将不是媒体模块独占内存，当系统内存不足时，将从这段预留内存申请，修改补丁如下：

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
b/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
index a60fa8f0c9d4..c9ddf42a55ee 100644
--- a/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi
@@ -65,9 +65,16 @@
compatible = "shared-dma-pool";
inactive;
reusable;
-           size = <0x0 0x20000000>;
+           size = <0x0 0x00000000>;
alignment = <0x0 0x1000>;
};
```

```
+        linux_cma: linux-cma {
+            compatible = "shared-dma-pool";
+            reusable;
+            size = <0x0 0x30000000>;
+            alignment = <0x0 0x1000>;
+                        linux,cma-default;
+        };
};
```

- 用户态配置，目前可以通过以下几种方式使能 CMA 内存
  - export rt_mem_heap 0x1200。 输入该命令后， 将会使得各模块（除VO模块输出 Buffer）默认使用CMA内存。
  - 调用 **RK_MPI_MMZ_Alloc** 申请内存时，**u32Flags |= RK_MMZ_ALLOC_TYPE_CMA**。
  - 如果需要将内存池的内存使用为CMA内存，可以在调用 **RK_MPI_MB_CreatePool** 创建内存池时，将传参中的 **enDmaType** 设置为**MB_DMA_TYPE_CMA**。

2.2.2 优化媒体内存方法

以下内存优化方法，需要结合具体产品/场景确认对应的参数配置合理值。

- **VO模块配置**

措施	相关接口	收益	影响	注意
像素格式改为NV12	RK_MPI_VO_SetLayerAttr: enPixFormat = RK_FMT_YUV420SP	相比RGBA8888一个素格节省2.5 Byte	GPU合成性能	需根据不同平台的GPU合成效率，配置相应的合成策略。如在RK356X 双屏4K+1080P场景时，4K屏配置为NV12，1080P屏配置为RGB888
显示缓存队列buffer数设置为3	RK_MPI_VO_SetLayerDispBufLen	默认4个buffer	VO显示的流畅性	-
wbc像素格式配置为NV12	RK_MPI_VO_SetWbcAttr: enPixFormat = RK_FMT_YUV420SP	节省像素占用的内存	-	-
wbc输出缓存buffer数配置为3	RK_MPI_VO_SetWbcDepth	默认4个buffer	-	-

- **VDEC模块配置**

措施	相关接口	收益	影响	注意
配置解码输出帧缓存数	RK_MPI_VDEC_CreateChn: u32FrameBufCnt = 3	默认为6帧	过少可能导致解码卡住	H.264/H.265每个解码通道输出帧缓存至少为参考帧+显示帧+1 JPEG/MJPEG每个解码通道输出帧缓存至少为显示帧+1
配置解码输入码流缓存数	RK_MPI_VDEC_CreateChn: u32StreamBufCnt = 3	默认为8包	过少会增加解码器等待数据输入耗时	-
H.264/H.265不申请MV buffer	RK_MPI_VDEC_SetChnParam: bTemporalMvpEnable = RK_FALSE	节省MV buffer	需要MV却没创建会导致画面频繁跳转不连续，内存访问越界，内核日志频繁打印iommu: Page fault	1.如果 H.264 解码不需要解码 B 帧，或者 H.265 解码不需要解码支持时域运动矢量预测（sps_temporal_mvp_enabled_flag = 1）的码流，则配置 bTemporalMvpEnable 为 FALSE，否则配置为TRUE（若H.265 解码配置为 FALSE, 但码流中 sps_temporal_mvp_enabled_flag为1，仍会去创建MV内存。） 2.目前限定需配置为解码序输出，以上才可生效。 RK_MPI_VDEC_SetChnParam：enOutputOrder = VIDEO_OUTPUT_ORDER_DEC
限制解码总/驻留buffer	配置通道内存分配总大小: export rt_vdec_mb_max_alloc=209715200 //200M 配置通道内存驻留总大小: export rt_vdec_mb_max_retain=104857600 //100M	默认不限制总大小，解码完buffer不驻留	-	如果限制总大小，超出的解码通道将会创建失败。配置通道驻留内存目的是为了加快通道创建销毁速度。
关闭编解码器packet/frame buffer状态记录	export buf_slot_debug=0x0 (默认值为0x10000000)	一路会开辟16k*2(一个存packet，一个存frame)	-	-

- **VENC模块配置**

措施	相关接口	收益	影响	注意
配置编码输出包缓存数	RK_MPI_VDEC_CreateChn: u32StreamBufCnt = 3	默认配置为4包	-	-

- **ZRAM内存压缩**

  ZRAM 是 Linux 的一种内存优化技术，可通过开启ZRAM，当系统内存紧张时释放出更多的可用内存。以下脚本示例为开启512MB大小ZRAM内存空间。

```
#!/bin/sh
echo $((512*1024*1024)) > /sys/block/zram0/disksize
mkswap /dev/zram0
swapon  /dev/zram0
sysctl -w vm.swappiness=100
```

2.3 性能优化

2.3.1 CPU性能

- 对精度要求不高的场景下，可调整调度时间片、关闭高精度定时器，来提升CPU Idle。

```
// 修改调度时间片为100Hz
CONFIG_HZ_100=y
// 关闭高精度定时器
#CONFIG_HIGH_RES_TIMERS=y
```

- 分散中断，比如将以太网中断eth0中断迁移出CPU0。
- CPU频率定频、提升运行频率。

2.3.2 解码性能

- H264、H265解码器通道配置为帧压缩模式，以降低 DDR 带宽占用率，提升解码性能。

```
VDEC_CHN_PARAM_S stVdecParam;
stVdecParam.stVdecVideoParam.enCompressMode = COMPRESS_AFBC_16x16;
RK_MPI_VDEC_SetChnParam(VdecChn, &stVdecParam);
```

- MJPEG 不支持帧压缩，不能开启。

2.3.3 VPSS性能

- 通道模式配置策略详见6.2.1 通道模式最佳配置。
- 在RK356X平台，VPSS采用GPU模块实现，需要关注GPU频率配置。

2.3.4 VO性能

以RK356X平台为例：

- 建议HDMI最大支持4KP30，VGA最大支持1080P30。4K/双屏场景下，主副屏输出帧率应配置为30fps，否则性能会不足。
- 屏幕输出格式PixelFormat建议配置RGB888。

```
VO_VIDEO_LAYER_ATTR_S stLayerAttr;
stLayerAttr.enPixFormat = RK_FMT_BGR888;
stLayerAttr.u32DispFrmRt = 30;
RK_MPI_VO_SetLayerAttr(VoLayer, &stLayerAttr);
```

2.3.5 应用性能优化建议

- 减少线程数量以减少调度开销。
- 减少sleep，poll，timer定时器使用或适当调大调度间隔时长。
- 开启 O3/Os 编译优化。

2.4 系统相关

2.4.1 跨进程说明

【现象描述】

目前除 VO 模块外，其他 MPI 模块不支持跨进程操作，但根据项目需求在持续研发。目前 VO 模块视频层是通过 DRM 框架接口操作，如果 UI 在另一个进程操作也是通过DRM接口操作，两个进程可能会出现竞争问题，目前Rockit接口没有直接支持跨进程操作，但可以通过共享 Buffer 方式实现跨进程。

【分析解决】

- UI 初始化为单 Buffer 模式，把 VO 申请的 UI Buffer 转成唯一 ID，在通过 pipe/socket 等跨进程方式将此唯一 ID 传递给 UI 进程，UI 进程收到后再通过查询接口获取Fd，然后 map 出虚拟地址，写入 UI 数据。
- 鼠标跨进程支持也可以用此方法。
- 我们有提供 Sample 示例代码，可以通过我们支持人员获取。

2.4.2 1.4.2 其他系统相关

- NVR 相关产品
  详见文档 NVR SDK 文档《Rockchip_RK356X_NVR_SDK_Release_xxx_CN.pdf》，xxx为版本信息，不断迭代更新。

3. VDEC

VDEC 模块常见问题可借助 dumpsys vdec 工具进行分析排查。

3.1 接口调用错误

3.2 送流接口返回错误

【现象描述】

调用送流接口 **RK_MPI_VDEC_SendStream** 返回 **RK_ERR_VDEC_BUF_FULL** 错误。

【分析解决】

- max_input_cnt 被设置太小，导致 Buffer 很容易溢出，可尝试修改 u32StreamBufCnt 来增加输入缓冲的个数，比如设置8块。
- max_output_cnt 被设置太小，导致解码 Buffer 轮转不够，可尝试修改 u32FrameBufCnt 来增加输出轮转的个数，比如设置8块。

```
id    send_rate   max_input_cnt  left_input_cnt  left_input_size  max_output_cnt
err_status
0     19.35       8              7               41540            8
0
id    input_strm_cnt   output_frm_cnt   error_frm_cnt   unused_buf_num
0     77               70               0               0
```

- 若可用解码 Buffer(unused_buf_num) 个数为零，则可能有如下原因:
  - 确认 VDEC 后级模块是否长时间占用解码 Buffer。
    - dumpsys vpss，确认 VPSS 是否所有通道开启 USER 模式，全部开启 USER 模式会导致解码 Buffer 占用时间长。把不需要缩放的通道修改成 PassThrough 模式以降低模块处理开销，加快 Buffer 轮转速度。

```
---------------------- vpss channel attr -----------------------------
grp_id    chn_id    mode      width     height    pixel_format
is_compress    src_rate  dst_rate  depth     align     mirror    flip
0         0         USER      1920      1080      image:yuv420p  N
       -1        -1        1         16        N         N
1         0         USER      1920      1080      image:yuv420p  N
       -1        -1        1         16        N         N
2         0         USER      1920      1080      image:yuv420p  N
       -1        -1        1         16        N         N
```

    - dumpsys vo，确认 VO 的 Cache 缓存帧数大于1，说明 VO 占用解码 Buffer 时间长，需要确认 GPU 的负载和 CPU 的负载，若 GPU 负载超过90%，确认是否有关键日志出现，可以查看2.4 如何分析VDEC常见异常日志。

```
LAYER clu0 CHANNEL STATUS:
ChnId   Prio    X       Y       W       H       ChnFrt  FgAlpha BgAlpha
Show    Pause   Step    Cache   RevCnt
0       2       265     79      1650    888     13      0       128
Y       N       N       8       93
```

```
cat /sys/devices/platform/fde60000.gpu/utilisation  // 查看gpu负载
```

  - 应用业务是否超出该芯片对应的最大解码能力，可查看 VDEC 模块文档概述章节。
  - 系统带宽不足，需结合业务整体分析优化 DDR 带宽。
  - 可通过以下命令确认硬件解码能力耗时，内核日志会输出硬件解码耗时。

```
echo 0x100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
```

3.3 播放卡顿

【现象描述】

解码输出帧率不足或显示输出时出现播放卡顿现象。

【分析解决】

- **送数据帧率不足**
  dumpsys vdec，若 send_rate 的帧率小于预期，且 left_input_size 和 left_input_cnt 为零，表示输入数据不够导致播放卡顿，需要检查输入数据的速率。

```
id      send    send_ok    send_rate   max_input_cnt  left_input_cnt
left_input_size
0       386     385        19.35       8              0               0
```

- **解码速率不够**
  解码速率不够，会导致解码器报错 **RK_ERR_VDEC_BUF_FULL**，可参考2.1 如何分析送流接口返回错误。

3.4 播放无输出

【现象描述】

解码无输出帧或显示画面黑屏（无解码图像输出）。

【分析解决】

- **VDEC未接收到数据**
  dumpsys vdec，send 值为0，说明 **RK_MPI_VDEC_SendStream** 收到的数据为零，需要检查输入情况。

```
id      send    send_ok    send_rate   max_input_cnt  left_input_cnt
left_input_size
0       0       0           0          8              0               0
```

- **VDEC未解码**
  - 若 input_strm_cnt 大于零，output_frm_cnt 等于零，说明解码没有工作，检查log，看解码器是否有错误产生。
  - 若 input_strm_cnt 大于零，output_frm_cnt 大于零，且不断增长，说明解码器正常，需要检查后级模块是否丢失解码数据。
  - 若 input_strm_cnt 大于零，output_frm_cnt 大于零，且没有不断增长，说明解码器只解码部分，需要检查数据停止输入导致。
  - 若 input_strm_cnt 和 error_frm_cnt 个数接近，且不断增长，说明解码的数据有问题，需要检查输入数据的完整性，比如是否是完整的一帧。

```
id        input_strm_cnt   output_frm_cnt   error_frm_cnt
unused_buf_num
0         4                0                0              0
```

3.5 分析VDEC常见异常日志

- 当日志出现类似错误信息， 说明进程的fd已耗尽，可以使用命令ulimit -n 4096 来调整进程的fd个数。

```
fail to drm_ioctl(fd = 17, req =-1072929747), error: Too many open files
```

- 当日志出现类似错误信息， 说明 GPU 负载较重，合成变慢。

```
mpi_vo_composer  03:29:33-736 {vo_composer_thread:968} Layer 0 compose cost
61607 us, frame interval 40000 us
```

4. VENC

4.1 编码支持类型及规格参数

查看《Rockchip_Developer_Guide_MPI_VENC.md》的“概述”章节部分。

4.2 3.2 如何排查没有数据输出

- 首先使用rk_mpi_venc_test测试，若测试正常，检查下与代码示例的差异。
- 检查s32RecvPicNum是否设置为0，或者通过dumpsys venc查看snap_set是否为非-1并且seq已经等于snap_set数值，此时编码会停止输出。
- 检测输出buffer是否设置太小，导致编码输出异常报错。
- 查看串口日志是否有mpp或者其他相关报错。

4.3 销毁通道后为何内存没有释放

- 为避免频繁创建销毁编码通道导致额外开销，目前默认编码通道为常驻状态。通过查看dumpsys mb可以看到通道销毁后的常驻编码内存。可以减小编码输出buffer个数和大小来限制常驻内存。建议输出buf个数设置4个，大小为图像宽*高大小即可。

4.4 编码输出帧率不足

- 确认帧率是否超过实际规格书定义。

- 确认单独使用rk_mpi_venc_test是否帧率正常。如果test测试也异常，查看芯片温度是否过高导致机器进入温控，查看编码频率是否有降低。如果都正常则串口输入以下命令查看内核编码单帧时间是否过长。

```
echo 0x100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
```

- 确认是否开启帧率控制导致，可以通过dumpsys venc查看。
- 确认是否是VENC通道后级绑定模块归还Buffer数据慢，或用户获取/释放Buffer的速度慢。
- 若单独编码正常，加上其他业务出现帧率低，此时需分析DDR带宽是否充足，可以尝试提高DDR频率，并结合整体业务分析优化带宽占用。

4.5 输出图像马赛克

- 确认输入图像是否正常。
- 确认单独使用rk_mpi_venc_test是否正常。
- 使用rk_mpi_vdec_test解码查看是否正常。
- 检查是否超频使用编码频率，降低对应clk查看是否正常。
- 尝试使用其他编码器是否正常。如果是mjpeg/h264/h265单独一个编码器异常，查看对应编码器电压供电是否满足需求，尝试抬升25-50mv查看是否有改善。

4.6 其他注意事项

- 动态参数设置建议在创建通道后、启动编码前设置。避免在编码过程中频繁设置。

5. VI

5.1 如何获取支持的分辨率

- 分辨率支持情况可以查看《Rockchip_Developer_Guide_MPI_VI.md》的功能描述章节介绍。
- 是否支持某个分辨率可以通过v4l2-ctl命令去抓流验证，目前VI支持的分辨率跟v4l2-ctl支持的一致。命令示例如下：

```
v4l2-ctl -d /dev/video19 --set-fmt-
video=width=1920,height=1080,pixelformat=NV12 --stream-mmap=3 --stream-skip=3
--stream-count=10 --stream-poll  --stream-to=/tmp/12M.yuv
```

5.2 输出帧率不足

- 确认输入帧率是否正常。ISP输入可通过cat /proc/rkisp*查看rate信息。
- 确认是否开启帧率控制。可以通过dumpsys vi查看。
- 确认是否是VI通道后级绑定模块归还Buffer数据慢，或用户获取/释放Buffer的速度慢。
- 若VI单独输出帧率正常，而绑定VO后输出帧率不足，可提高VI的输出buffer个数，建议buffer个数设置不小于4个。

5.3 VI初始化失败

日志中有提示VI报错信息: “failed to ispStreamOn()”, 分析步骤如下:

- 确认硬件及驱动正常注册，通过使用rk_mpi_vi_test或者v4l2-ctl抓流来确认是否正常。
- 确认打开的分辨率是否是当前通道支持的分辨率。
- 查看对应输出log，找到第一个对应的报错是否有明显提示，如格式不支持、设备busy等。
- 确认配置的memoryType参数是否正常，若输入源为BT1120或者ISP驱动配置使用的是CMA Buffer，则memoryType要配置为1。

5.4 如何区分多个设备

- RV1126平台带Sensor可以通过dev号进行区分。默认dev 0为isp0，dev 1为isp1。
- 全平台都可以通过设置aEntityName来进去区分，如/dev/video0 、/dev/video19等。

5.5 其他注意事项

不支持多个进程/线程同时打开一路数据流，支持一个进程/线程通过bind多个后级或者绑定vpss处理同一路vi数据。同理，vi或者v4l2-ctl同时只能有一个应用能使用同一路数据流，比如/dev/video0。

6. VO

6.1 配置VO底色

```
stVoPubAttr.u32BgColor = 0xFF0000;
RK_MPI_VO_SetPubAttr(VoDev, &stVoPubAttr)
```

6.2 显示输出配置

- 目前的HDMI强制输出方案，强制输出后，应用还是能检测到HDMI热拔插事件，可以获取到HDMI的状态。默认配置，uboot，kernel阶段HDMI & VGA强制输出，默认输出分辨率为1024x768。详细配置在arch/arm64/boot/dts/rockchip/rk3568-nvr.dtsi的&route_hdmi 以及&route_edp 节点中。
- 应用通过drm接口可以获取到HDMI接口的状态，上层可以通过RK_S32 RK_MPI_VO_RegCallbackFunc(RK_U32 enIntfType, RK_U32 u32Id, RK_VO_CALLBACK_FUNC_S *pstCallbackFunc) 这个接口注册对应的回调。
- 应用起来后，使能HDMI&VGA对应的VoDev之后对应的设备就是常输出的，建议（HDMI用RK356X_VO_DEV_HD0，VGA用RK356X_VO_DEV_HD1）。

6.3 同显异显输出

以 RK356X 平台为例：

- 同显的模式下，HDMI&VGA 只能输出同一种分辨率。
- 异显的模式下，建议HDMI最大支持4kP30，VGA最大支持1080P30。
- 同显/异显切换，是通过应用层去操作VoDev来实现的。
    - 同显：dts中把edp_in_vp0使能起来 ;应用使能VP0，VoPubAttr.enIntfType = VO_INTF_HDMI | VO_INTF_EDP。
    - 异显：应用使能两个VoDev。

6.4 VO图层使用

以RK356X平台为例，对应的默认绑定配置如下：

```
dev 0:   0-video,   4-ui,  6-cursor
dev 1:   2-video,   5-ui,  7-cursor
-------LAYER BIND CONFIG--------
DevId   Video   Gfx     Cursor
0       clu0    esm0    sm0
1       clu1    esm1    sm1
2       unkown  unkown  unkown
```

使用建议：

- 视频层和 UI 层均使用 Cluster 图层， UI 图层数据送往视频层的末尾通道(如VO_MAX_CHN_NUM-1，VO_MAX_CHN_NUM-2通道)。
- 鼠标单独初始化一层32x32大小。

7. VPSS

7.1 常见问题分析

7.1.1 无法获取Group帧

【现象描述】

调用 **RK_MPI_VPSS_GetGrpFrame** 无法获取到帧。

【原因分析】

RK MPI获取Group帧条件较多，需要满足多种条件：

- 该Group存在Passthrough模式的通道。
- 该Group存在与VO绑定的通道。
- backup帧开启。

【解决方法】

解决该问题需要有以下操作：

- 调用 **RK_MPI_VPSS_EnableChn** 开启通道，并且该通道设置为 **VPSS_CHN_MODE_PASSTHROUGH**。
- 调用 **RK_MPI_SYS_Bind** 将条件1的 VPSS 通道同 VO 通道进行绑定。
- 调用 **RK_MPI_VPSS_EnableBackupFrame** 使能backup帧。

7.1.2 无法从通道获取帧

【现象描述】

调用 **RK_MPI_VPSS_GetChnFrame** 无法获取到帧。

【原因分析】

通道无法获取帧的情况有以下几种需要确认：

- 确认该通道是否开启。（可以通过dumpsys vpss确认是否可以查看到该通道的信息，如果查看不到，则是未开启）
- 确认该通道属性 u32Depth 是否不为0。（为0时表示通道不保留帧，全部丢弃）。
- 确认该通道是否绑定了下级。目前不支持绑定下级的同时手动取帧的情况。
- 确认该通道的输出均被 **RK_MPI_VPSS_GetChnFrame** 全部取出后而没有调用 **RK_MPI_VPSS_ReleaseChnFrame** 释放。

【解决方法】

解决该问题需要有以下操作：

- 调用 **RK_MPI_VPSS_EnableChn** 开启通道。
- 调用 **RK_MPI_VPSS_SetChnAttr** 将 **VPSS_CHN_ATTR_S** 中的 **u32Depth** 设置为大于0的值。
- 如果有绑定下级，请调用 **RK_MPI_SYS_UnBind** 进行解绑。
- **RK_MPI_VPSS_GetChnFrame** 和 **RK_MPI_VPSS_ReleaseChnFrame** 必须成对使用。

7.1.3 解决获取的图像花屏的问题

【现象描述】

通过 **RK_MPI_VPSS_GetChnFrame** 或 **RK_MPI_VPSS_GetGrpFrame** 获取到的帧图像花屏。

【原因分析】

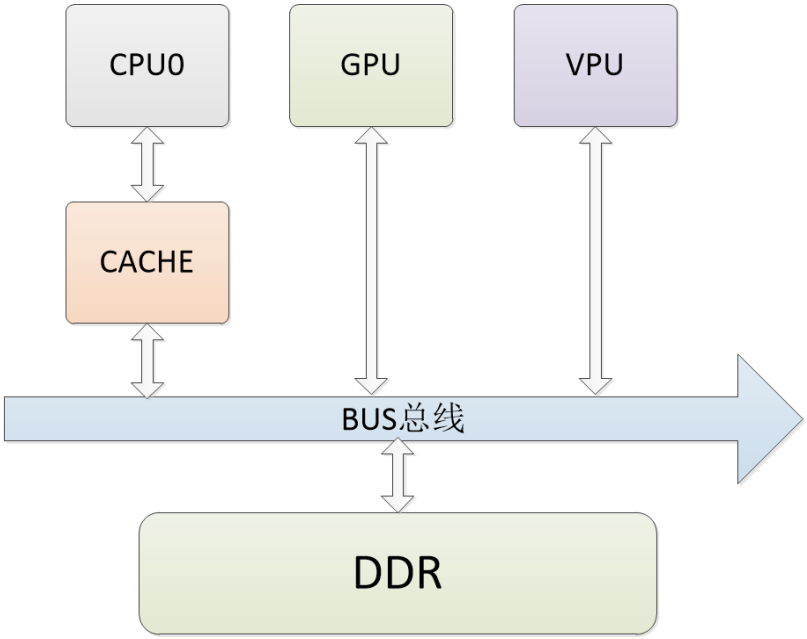
可能有以下几种原因：

- 图像虚宽高不对齐导致图像数据读取错误。
- CPU cache一致性问题。
- 图像源数据为花屏。
- 调用 **RK_MPI_VPSS_ReleaseChnFrame** 后仍然在使用。

【解决方法】

- 检查图像对齐情况
    - 检查输出图像是否按照 **VIDEO_FRAME_INFO_S** 结构体中的实际宽高、虚宽高（像素对齐宽高）、图像格式、压缩方法使用。各种对齐方式遵循 VPSS 模块开发文档中的对齐方式列表要求。
    - 检查是否将虚宽高当成实宽高使用，如果误设，将会在图像右侧或下方出现黑色条纹（也可能是随机的脏数据）。

- 检查CPU读写图像的cache一致性



这种情况下的典型画面是出现**细条的错误数据**（由于cache整块未被同步，会出现连续的脏数据），原理如上图。解决办法为从图像中读取数据到CPU后调用 **RK_MPI_SYS_MmzFlushCache** 清读写方向的cache（即参数设置为RK_FALSE）。从图像写数据到CPU前调用 **RK_MPI_SYS_MmzFlushCache** 清读方向的cache（即参数设置为RK_TRUE）。

- 检查数据源是否已经花屏

  此时可以将输入数据写下来，检查输入数据是否花屏。

```
// 录制VPSS 组 0 通道 0 输入数据至/data/out1.nv12，录制10帧
dumpsys record -m vpss -d 0 -c 0 -i 1 -o /data/out1.nv12 -n 10
```

  然后将录制的数据通过7YUV等工具查看。

- 检查是否出现 **RK_MPI_VPSS_ReleaseChnFrame** 后仍然使用的情况

  如果在图像被释放仍然读取图像数据，那么这帧图像可能在被重新使用时读取，此时也会表现为花屏，甚至应用直接崩溃。

7.1.4 送帧超时

【现象描述】

调用 **RK_MPI_VPSS_SendFrame** 返回 **RK_ERR_VPSS_BUF_FULL**。

【原因分析】

开启了帧率控制，导致内部送帧阻塞。

【解决方法】

帧率控制开启后的阻塞时间小于送帧超时时间。例如输入帧率设置为30，内部送入阻塞时间将会变为33ms左右，此时送帧超时时间必须大于33ms，否则将可能出现送帧超时的情况。

7.1.5 取帧超时

【现象描述】

调用 **RK_MPI_VPSS_GetChnFrame** 返回 **RK_ERR_VPSS_BUF_EMPTY**。

【原因分析】

取帧超时一般有以下几种可能性：

- 开启了帧率控制。
- 达到性能边界。

【解决方法】

- 开启帧率控制时输出帧率时间小于取帧超时时间。例如设定输出帧率为5帧，那么在没有性能问题情况下取帧间隔为200ms，如果设定取帧超时时间小于200ms，也会出现取帧超时。
- 调用接口超时时间必须大于 VPSS 处理帧的时间，但在系统压力很大的情况下这个时间可能被拉长，需要结合其他模块综合处理性能问题。

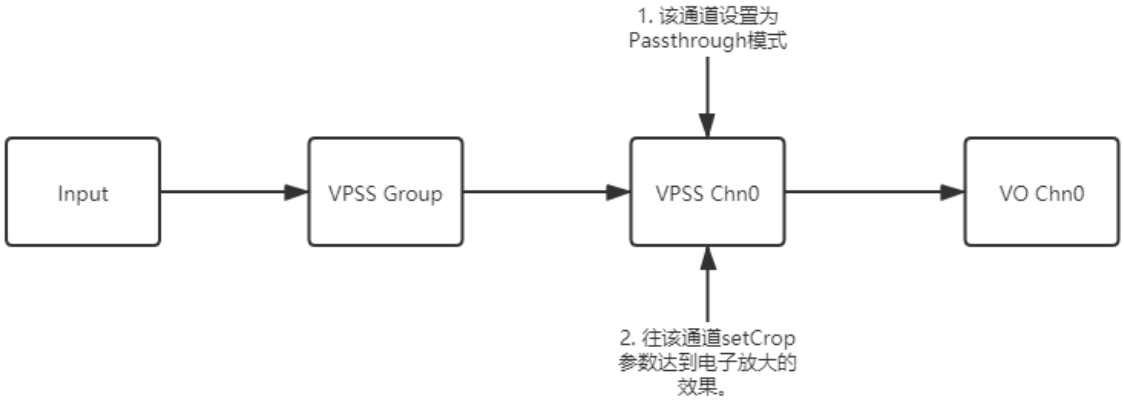
7.2 最优性能实践

7.2.1 通道模式最佳配置

VPSS 模块硬件资源性能受限，在不需要从 VPSS 通道手动获取输出帧（如用于AI算法、用户数据分析处理）情况下，需要将 VPSS 通道的工作模式 **enChnMode** 配置为 **VPSS_CHN_MODE_PASSTHROUGH**。在该模式下，VPSS 模块不做实际数据处理，以透传的方式传递给后级模块，以达到最高效率。

7.2.2 电子放大

电子放大通常使用 VPSS 设置组或通道裁剪，来达到放大缩小的功能。目前RK平台的最优解如下图所示：



如上图所需注意点：

- 通道设置为 **VPSS_CHN_MODE_PASSTHROUGH** 模式。
- 电子放大参数往通道中设，而不是 Group。
- 需要绑定 VO 通道。

8. VGS

8.1 版本和调试信息

8.1.1 版本信息

与 VGS 功能相关的的库包括librockit.so 和 libgraphic_lsf.so。可以通过 dumpsys --version命令一键获取相关版本号。

log相关的版本信息如下：

```
rockit version: git-9625ad8 Tue Sep 28 19:16:44 2021 +0800
rockit building: built-daw 2021-09-29 11:31:52

rk-debug init version=2.93,args[16,16,0], threadId=547547050416
```

8.1.2 调试信息获取

通过dumpsys vgs 获取调试信息， 调试信息包含输入和输出图像的宽高 format、当前和历史处理的job和task的信息等。具体的调试信息参考《Rockchip_Developer_Guide_MPI_DUMP》文档 VGS 章节。

8.1.3 VGS支持的规格

VGS支持常见的图像格式如yuv420、rgb888、rgba8888等, 更多支持规格参考《Rockchip_Developer_Guide_MPI_VGS》文档。

8.2 问题调试

8.2.1 分析Crop处理图像绿屏

- 使用VGS的crop接口进行抠图，抠出来的图像是绿的。出现如下的错误log:

```
rk-debug eglCreateImageKHR NULL dpy=0x7fa8000f00
rk-debug create_egl_img [14,736,480] afbc=0 ,format=35314152,stride=736
,color_space=327f,sample_range=3283
rk-debug eglCreateImageKHR out fail
```

- VGS硬件输入和输出的图像的宽高、虚宽高根据不同的格式有不同的对齐要求。 比如BGRA1555 格式的宽需要32像素对齐，如果输入的图像宽度不是32像素对齐，那么就会输出上述log，抠图失败。同样的输出图像的buffer也需要根据不同格式进行对齐, 否则会出现内存操作越界。

8.2.2 分析抠图边缘出现绿边或者花边

目前VGS硬件处理时，不会对输出buffer进行清空操作。如果输出Buffer是复用的，那么就存在脏数据，在VGS操作区域范围外的数据就可能是绿或者花的。建议输出Buffer在使用前进行清空操作。

9. TDE

9.1 版本和调试信息

9.1.1 版本信息

与TDE功能相关的的库 包括librockit.so 和 librga.so。可以通过 dumpsys --version命令一键获取相关版本号。

log中相关版本信息如下：

```
rockit version: git-9625ad8 Tue Sep 28 19:16:44 2021 +0800
rockit building: built-daw 2021-09-29 11:31:52
Rga built version:1.04 9e331f5+2021-04-12 14:19:55
```

9.1.2 调试信息获取

通过dumpsys tde 获取调试信息， 调试信息包含输入和输出图像的宽高 format、当前和历史处理的job和task的信息、task处理耗时等信息。具体的调试信息参考《Rockchip_Developer_Guide_MPI_DUMP》文档TDE章节。

9.1.3 TDE支持的规格

TDE支持常见的图像格式如yuv420、rgb888、rgba8888等, 更多支持规格参考《Rockchip_Developer_Guide_MPI_TDE》文档。

9.2 问题调试

9.2.1 多次调用TDE失败

- TDE 多次调用begin 和end job，只有第一次调用成功，后面几次都会失败，出现如下报错。

  ```
  Try to use uninit rgaCtx=(nil)
  ```

- TDE调用RGA 相关接口没有初始化导致的。 问题在新版本的rga库上面已经解决，需要更新librga.so。

9.2.2 外部源数据使用TDE处理出现绿边

Buffer的数据存在虚宽高，如果没有设置虚宽高的话，使用的是图像实际的宽高作为虚宽高，导致硬件取数出现偏差。可以通过RK_MPI_CAL_XXX_GetPicBufferSize（XXX为具体模块名）传入实际的图像宽高获取到具体模块对齐后的虚宽高，再将虚宽高转换为字节为单位的Stride，通过RK_MPI_MB_SetBufferStride 设置到MB_BLK中。目前MPI模块(如VI/VDEC/VPSS)产生的图像数据送入到TDE，TDE模块可以获取到虚宽高，不需要设置Stride，外部图像数据源根据需要设置对应的Stride。

```
PIC_BUF_ATTR_S stPicBufAttr;
MB_PIC_CAL_S stMbPicCalResult;
TDE_SURFACE_S stSrcSurface, stDstSurface;

stPicBufAttr.u32Width = stSrcSurface.u32Width;
stPicBufAttr.u32Height = stSrcSurface.u32Height;
stPicBufAttr.enPixelFormat = stSrcSurface.enColorFmt;
stPicBufAttr.enCompMode = COMPRESS_MODE_NONE;
RK_MPI_CAL_TDE_GetPicBufferSize(&stPicBufAttr, &stMbPicCalResult);
RK_U32 u32HorStride = RK_MPI_CAL_COMM_GetHorStride(stMbPicCalResult.u32VirWidth,
stSrcSurface.enColorFmt);
RK_U32 u32VerStride = stMbPicCalResult.u32VirHeight;
RK_MPI_MB_SetBufferStride(stDstSurface.pMbBlk, u32HorStride, u32VerStride);
```

9.2.3 使用TDE进行缩放或者格式转换出现绿屏

- 确认需要处理的格式是不是在模块支持的列表中。
- 确认输入和输出图像的实宽高和虚宽高是否符合对齐的要求。

9.2.4 使用TDE进行压缩图像拷贝出现绿屏

压缩图像的拷贝需要调用者知道拷贝的大小，然后根据大小转换成非压缩图像的拷贝，实际使用的是TDE单纯的内存拷贝功能。设置的参数为拷贝的大小转换后的非压缩格式宽高，建议图像格式设置为RGBA8888。

9.2.5 TDE yuv400 格式支持

目前TDE支持yuv400数据的输入和输出，但由于硬件的限制，yuv400作为输入格式时，建议Buffer大小按YUV420大小分配，否则内部处理前会进行一次输入buffer的扩大和拷贝。

9.3 性能相关

9.3.1 TDE处理耗时分析

TDE处理耗时跟图像大小、硬件模块的频率、DDR带宽频率和使用率有关系。

- 确认TDE处理耗时的时间

通过dumpsys tde 中cost_time 来查看最近几次调用TDE所使用的时间。
打印如下：

```
seq_no    job_hdl   state     task_num  in_size   out_size  cost_time hw_time
0         0         proced    1         345600    921600    2813      0
```

通过如下命令 来确认底层硬件实际使用的时间

```
echo time > /sys/kernel/debug/rga2_debug/rga2
dmesg
```

log如下：

```
[668612.190859] rga2: sync one cmd end time 709
```

- 确认RGA硬件模块的频率

通过如下命令可以 确认RGA运行的频率 和 修改RGA运行的频率。

```
cat /sys/kernel/debug/clk/clk_summary | grep rga  //查看rga频率
echo 400000000 > /sys/kernel/debug/clk/aclk_rga/clk_rate // 修改rga频率
```

- 确认DDR的频率

通过如下命令可以确认和修改DDR的频率

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

查看DDR频率的可设置范围
cat /sys/class/devfreq/dmc/available_frequencies
设置DDR governors为userspace模式
echo userspace > /sys/class/devfreq/dmc/governor
设置DDR频率
echo 1560000000 > /sys/class/devfreq/dmc/userspace/set_freq
```

- 确认DDR的带宽使用率

通过DDR带宽使用来确认当前的带宽资源是否比较紧张。当带宽资源紧张时，RGA硬件模块访问DDR效率降低，从而导致TDE处理的性能下降。

正常情况下TDE处理一张1080p的图像耗时在4-5ms之间，如果耗时明显多于4-5ms。 首先确认下RGA硬件的频率。RGA正常运行的频率在300M左右，如果明显低于这个频率，可以通过手动修改RGA频率来确认问题。RGA运行频率正常的情况下，确认DDR的运行频率，如果频率明显低于预期频率，可以通过手动修改DDR频率来确认。如果频率都正常的话，需要跑DDR带宽使用率的脚本，确认各个模块的DDR使用率，结合应用业务优化DDR带宽使用率。

10. AUDIO

10.1 AO/AI对接调试步骤

在对接AO/AI之前，先确定SDK默认挂载的声卡驱动是否满足产品场景需求，根据对接客户的经验，一般SDK默认挂载的声卡是不满足的，需要根据产品硬件图调试对接音频驱动。
以具体SDK注册的声卡为例，按以下步骤进行声卡调试。

1.首先通过命令查看音频驱动注册的声卡

查看注册声卡命令	描述
cat proc/asound/cards	[root@RK356X:/]# cat proc/asound/cards 0 [rockchiphdmi]: rockchip_hdmi - rockchip,hdmi rockchip,hdmi 1 [rockchiprk809co]: rockchip_rk809- - rockchip,rk809-codec rockchip,rk809-codec 2 [ROCKCHIPSPDIF]: ROCKCHIP_SPDIF - ROCKCHIP,SPDIF ROCKCHIP,SPDIF

如果声卡初始化成功，那么通过如上命令，可以看到注册成功后的声卡信息。反之，如果通过如上命令看不到声卡，则说明声卡驱动注册失败，需要检查声卡驱动。

2.用命令行测试声卡播放和录音功能

1）播放测试

使用aplay命令播放wav文件，测试下声卡的播放功能。
注意：某些codec，比如rk809需要先使用amixer配置Playback Path，以配置codec内部的播放通路。

测试hdmi播放：这里是播放声卡0（rockchiphdmi）：

命令行播放	描述	参数描述
aplay -Dhw:0,0 /userdata/t48.wav	[root@RK356X:/]# aplay -Dhw:0,0 /userdata/t48.wav Playing WAVE '/userdata/t48.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Stereo	-Dhw:**x**,0 : **x**代表的声卡序号

测试rk809播放：需要先设置amixer，再使用aplay播放。这里rk809是声卡1（rockchiprk809co）：

命令行播放	描述	参数描述
amixer cset -c 1 name="Playback Path" "SPK_HP" (非rk809codec不需要设置)	[root@RK356X:/]# amixer cset -c 1 name="Playback Path" "SPK_HP" numid=1,iface=MIXER,name='Playback Path' ;type=ENUMERATED,access=rw----- -,values=1,items=11 ; Item #0 'OFF' ; Item #1 'RCV' ; Item #2 'SPK' ; Item #3 'HP' ; Item #4 'HP_NO_MIC' ; Item #5 'BT' ; Item #6 'SPK_HP' ; Item #7 'RING_SPK' ; Item #8 'RING_HP' ; Item #9 'RING_HP_NO_MIC' ; Item #10 'RING_SPK_HP' : values=6	-c **x** : **x**代表的声卡序号; values=6 : 表示Playback Path设置为SPK_HP
aplay -Dhw:1,0 /userdata/t48.wav	[root@RK356X:/]# aplay -Dhw:1,0 /userdata/t48.wav Playing WAVE '/userdata/t48.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Stereo	-Dhw:**x**,0 : **x**代表的声卡序号

2）录音测试

录音：使用arecord命令录制wav文件，测试声卡的录音功能。
注意：某些codec，比如rk809需要先使用amixer配置Capture MIC Path，以配置codec内部的录音通路。
测试rk809录音，需要先设置amixer，再使用arecord播放。这里是录制声卡1（rockchiprk809co）：

命令行播放	描述	参数描述
amixer cset -c 1 name="Capture MIC Path" "Main Mic" (非rk809codec不需要设置)	[root@RK356X:/]# amixer cset -c 1 name="Capture MIC Path" "Main Mic" numid=2,iface=MIXER,name='Capture MIC Path' ; type=ENUMERATED,access=rw----- -,values=1,items=4 ; Item #0 'MIC OFF' ; Item #1 'Main Mic' ; Item #2 'Hands Free Mic' ; Item #2 'Hands Free Mic' ; Item #3 'BT Sco Mic' : values=1	-c **x** : **x**代表的声卡序号; values=1 : 表示Capture MIC Path设置为Main Mic
arecord -Dhw:1,0 -r 16000 -c 2 -f s16_le /userdata/test.wav	[root@RK356X:/]# arecord -Dhw:1,0 -r 16000 -c 2 -f s16_le /userdata/test.wav Recording WAVE '/userdata/test.wav' : Signed 16 bit Little Endian, Rate 16000 Hz, Stereo	-Dhw:**x**,0 : **x**代表的声卡序号 -r sample rate（采样率） -c channels（声道数） -f format（格式）

3.AO/AI软件接口对接

声卡驱动播放和录音功能调试正常后，再对接AO/AI软件接口。可以先用公版的测试sample编译出来的bin文件（sample源码：test_mpi_ao.cpp/test_mpi_ai.cpp）进行验证测试。下面是常用参数测试用例：

AO/AI	测试参数	参数描述
AO	./rk_mpi_ao_test -i 44100_2.pcm --device_rate 48000 --device_ch 2 --input_ch 2 --input_rate 44100 --sound_card_name hw:1,0	i : 输入文件 device_rate ： 播放声卡采样率 device_ch ： 播放声卡声道数 input_rate ： 播放数据的采样率 input_ch ： 播放数据的声道数 sound_card_name ： 声卡名称（可以是实际物理声卡或者asound.conf的虚拟声卡，不配置默认打开pcm.card0）
AI	./rk_mpi_ai_test --device_rate 16000 --device_ch 2 --out_rate 16000 --out_ch 2 --sound_card_name hw:1,0 -o ./ai	o : 输出文件夹 device_rate ： 打开录制声卡采样率 device_ch ： 打开录制声卡声道数 out_rate ： 录制数据的采样率 out_ch ： 录制数据的声道数 sound_card_name ： 打开声卡（可以是实际物理声卡或者asound.conf的虚拟声卡，不配置默认打开pcm.record0）

Linux版本sdk，音频MPI（AO/AI）基于alsa框架，可以通过下面官方网站了解更多。

描述	网址
Alsa项目的官方网址	http://www.alsa-project.org/
Alsa LIB API Reference	http://www.alsa-project.org/alsa-doc/alsa-lib/
配置文件的语法	http://www.alsa-project.org/alsa-doc/alsa-lib/conf.html
Asoundrc的官方说明文档	http://www.alsa-project.org/main/index.PHP/Asoundrc

AO/AI中 AUDIO_DEV与声卡的映射：

AO/AI使用的虚拟声卡名	描述
pcm.cardx	x表示AoDevId，其取值范围为0~AO_DEV_MAX_NUM
pcm.recordx	x表示AiDevId，其取值范围为0~AI_DEV_MAX_NUM

以AUDIO_DEV AoDevId = 0为例，asound.conf的定义如下：

```
pcm.card0 {
    type asym
    playback.pcm "softvol_play0"
}

pcm.softvol_play0 {
    type softvol
    slave.pcm "dmixer0"
    control {
```

```
        name "MasterP Volume"
        card 0
        device 0
    }
    min_dB -40.0
    max_dB -1.8
    resolution 100
}

pcm.dmixer0 {
    type dmix
    ipc_key 5978293 # must be unique for all dmix plugins!!!!
    ipc_key_add_uid yes
    slave {
        pcm "hw:0,0"
    }
}
```

当应用使用AO AoDevId = 0作为输出时，AO会打开虚拟声卡pcm.card0，并通过aound.conf中的定义，逐级打开下一级节点，直至打开真正的物理声卡hw:0,0。如上，pcm.card0总的输出通路：

```
pcm.card0-->softvol_play0-->dmixer0-->hw:0,0
```

声卡节点	描述
pcm.card0	AO 使用的虚拟声卡名。AO AoDevId=0 时打开声卡
pcm.softvol_play0	alsa标准的softvol插件，用于音量流的音量控制。
pcm.dmixer0	alsa标准的混音插件。用于一个物理声卡上，不同音频流(AoChn)的混音成一路音频输出
"hw:**c**,**d**"	为实际的物理声卡。 **c**：card（可以通过查看声卡确认：cat proc/asound/cards） **d**：device（一般为0） (比如上面的"hw:0,0"表示实际的声卡0（rockchiphdmi）)

- 客户可按照自己的需求，在asound.conf中增加或者减少alsa的插件，以满足自己特定要求。
- 如客户可修改映射关系，比如修改最后一级虚拟声卡"pcm.dmixer0"中的"hw:0,0"修改为"hw:1,0"，那么AO(AoDevId=0)映射的是codec(hw:1,0)。

AI对接情况同理，这里不再赘述。

4.AO/AI打开声卡的两种方式

方式一：

MPI接口中不配置声卡名，使用AUDIO_DEV映射。如上AUDIO_DEV=0的AO, 会固定打开asound.conf中pcm.card0这个虚拟声卡（AI固定是pcm.record0），然后数据传递softvol_play0，再传递给dmixer0，最后写入hw:0,0。同理AUDIO_DEV=1，就是打开pcm.card1。
AUDIO_DEV与asound.conf中映射函数如下：

AO/AI	devId	固定打开虚拟声卡
AO	AUDIO_DEV=**x**	pcm.card**x**
AI	AUDIO_DEV=**x**	pcm.record**x**

方式二：

MPI接口中配置声卡名。以AO为例，参考测试sample中接口：

```
if (ctx->chCardName) {
    snprintf(reinterpret_cast<char *>(aoAttr.u8CardName),
                sizeof(aoAttr.u8CardName), "%s", ctx->chCardName);
}
```

这种方式打开声卡比较灵活，不受AUDIO_DEV和asound.conf中映射的限制。这里配置的声卡名，可以是任意在asound.conf中定义的声卡名，也可以直接是物理声卡"hw:**c**,**d**"。
比如AUDIO_DEV=0的AO，打开asound.conf中定义的虚拟声卡：pcm.card0，

```
snprintf(reinterpret_cast<char *>(aoAttr.u8CardName), sizeof(aoAttr.u8CardName),
"%s", "pcm.card0");
```

也可以直接打开物理声卡："hw:**c**,**d**"，比如打开物理声卡1："hw:1,0"，

```
snprintf(reinterpret_cast<char *>(aoAttr.u8CardName), sizeof(aoAttr.u8CardName),
"%s", "hw:1,0");
```

AI对接情况同理。

10.2 AO无声或者断音卡顿问题

AO主要有无声和卡顿断音问题。首先确认音频驱动是否正常：使用aplay播放wav是否正常，如果播放无声，需要找硬件驱动工程师确认。
1.AO无声

- 确认音频驱动是否正常，可以使用aplay确认能否正常播放wav音频。
- AO日志是否正常，播放过程中声卡是否正常打开：

```
// 查看挂载声卡
cat proc/asound/cards
// 查看场景声卡是否打开
cat proc/asound/cardX/pcm0p/sub0/hw_params /* cardX : X 表示声卡序号 */
```

如果声卡正常打开：确认送的pcm数据是否正常，以及音量是否静音。
如果声卡close：确认AO接口是否正常对接，可以对比AO测试sample情况。

2.AO断音卡顿
一般是送数据不及时造成的，可以看到是否有underrun的日志，类似如下：

```
card:hw:1,0: underrun
```

这时需要确认上层送数据的及时性。