

Rockchip Android Multimedia FAQ

文件标识: RK-PC-YF-310

发布版本: V1.0.0

日期: 2022-03-03

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

免责声明

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

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

商标声明

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

本文档可能提及的其他所有注册商标或商标, 由其各自拥有者所有。

版权所有 © 2022 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

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

本文档主要介绍 Rockchip 多媒体平台常见的调试手段以及开发过程中常见的问题清单。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。

芯片名称	内核版本
适配所有芯片	Linux-4.19、Linux-5.10

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
V1.0.0	陈锦森	2022-03-03	初始版本

目录

Rockchip Android Multimedia FAQ

1. 多媒体常见调试手段
 - 1.1 视频处理硬件基础
 - 1.1.1 平台常见编解码 IP
 - 1.1.2 平台芯片 IP 携带表
 - 1.1.3 芯片编解码能力规格表
 - 1.2 硬件单帧编解码处理时间
 - 1.3 VPU 频率查询与修改
 - 1.4 显示帧率
 - 1.5 抓取编解码输入输出
 - 1.6 抓取 USB Camera JPEG 解码输入输出
 2. 编解码支持类
 - 2.1 片源是否支持?
 - 2.2 硬编解码最大支持路数?
 - 2.2.1 多路编解码支持路数
 - 2.2.2 多路同编同解支持路数
 3. 编解码应用设计参考
 - 3.1 Android Media API
 - 3.2 编解码器设计范例
 - 3.2.1 native-codec
 - 3.2.2 rkvpv-codec
 - 3.2.2 mpp-codec
 - 3.3 JPEG 硬编解码参考范例
 4. 播放流畅性问题
 5. 其他常见问题
 - 5.1 编解码实例数量限制
 - 5.2 指定 SurfaceTexture 解码输出绿边问题
 - 5.3 编码模糊、马赛克问题
 - 5.4 视频播放内存泄漏问题
-

1. 多媒体常见调试手段

1.1 视频处理硬件基础

本文档问题的讨论可能涉及芯片硬编解码能力，因此首先简要介绍 Rockchip 平台芯片常见的编解码 IP，并记录了常见芯片的编解码 IP 携带情况及编解码能力最大规格。

Note: DateSheet 手册或 Codec BenchMark 标定芯片的编解码能力，是本章节格式支持类问题分析的必要条件。如需获取，请邮件联系方晗 (helen.fang@rock-chips.com)。

1.1.1 平台常见编解码 IP

1) VPU (VPU = VDPU + VEPU)

- VPU 是 1080P 多格式编解码器，现有两个版本 VPU1\VPU2，标称能力为 1080P@30fps
- VDPU 解码器，代号为 VDPU12X，支持 H.263/H.264、MPEG1/2/4、VP8、JPEG
- VEPU 编码器，代号为 VEPU12X，支持 H.264、VP8、JPEG
- 多见于早期芯片，如 RK312X、RK3326 等

2) VDPU34X

- RK 自研，解码 IP，代号 VDPU34X
- VDPU34X 是多格式高性能 4K 解码器，标称能力是 4K@30fps 以上
- 支持格式 H.264/H.265/VP9 解码，均支持 10bit
- 多见于现在的中高端芯片，如 RK356X、RK3399、RK3328 等

3) VDPU38X

- RK 自研，解码 IP，代号 VDPU38X
- VDPU38X 是多格式高性能 8K 解码器，标称能力是 8K@30fps 以上
- 支持格式 H.264/H.265/VP9/AVS2 解码，均支持 10bit
- 在目前的高端芯片 RK3588 中搭载

4) VEPU54X

- RK 自研，编码 IP，代号 VEPU54X
- VEPU54X 是高性能 4K 编码器，标称能力是 4K@30fps 以上
- 支持格式 H.264/H.265 编码
- 多见于现在的中高端芯片，如 RK356X

5) VEPU58X

- RK 自研，编码 IP，代号 VEPU58X
- VEPU58X 是高性能 8K 编码器，标称能力是 8K@30fps 以上
- 支持格式 H.264/H.265 编码
- 在目前的高端芯片 RK3588 中搭载

6) rkjpegd

- RK 自研，解码 IP，代号 VDPU720
- rkjpegd 是 jpeg 专用解码器，最大可支持 65536x65536，标称能力是 1080@60fps
- 多见于现在的中高端芯片，如 RK356X、RK3588 等
- RK3588 支持更高的主频，测试能力可达 1080P@200fps

后面所列 IP 是基于以上的改版或升级版本：

7) vdp1_2160p

VDPU1 的升级IP，在原先的 1080P 解码器基础上增加了 H264 4K 能力，因此除了常规的 H.263、MPEG1/2/4、VP8、JPEG 1080P 解码支持，还支持 ==4K 8bit H264== 解码。

目前 RK3288\RK3368 携带该版本解码 IP。

8) rk_hevc_1080p & rk_hevc

VDPU34X 的前身，用于支持 H265 解码，其中 rk_hevc_1080p 为 H265 1080P 解码支持能力，rk_hevc 支持 ==H265 4K 10bit== 解码。

目前 RK312X\RK3326 携带 rk_hevc_1080p IP，RK3288\RK3368 携带 rk_hevc IP。

9) vepu22

vepu22 是 vepu2 的升级 IP，是 H265 专用编码器，能力规格为 1080P@30fps。

目前 RK3328 携带该版本编码 IP。

NOTE：我们通常把高性能解码器 vdpu34x\vdpu38x 以 rkvddec 代称，高性能编码器以 vepu54x\vepu58x 以 rkvinc 代称。

1.1.2 平台芯片 IP 携带表

以下列举常见芯片的编解码 IP 携带情况，根据芯片持有 IP 的情况很容易推出芯片的编解码能力。

	dec_cap[1]	dec_cap[2]	dec_cap[3]	enc_cap[1]	enc_cap[2]
RK3588	vdpu38x	vdpu	rkjpegd	vepu58x	vepu
RK356X	vdpu34x	vdpu	rkjpegd	vepu54x	vepu
RK3399	vdpu34x	vdpu	N/A	vepu	N/A
RK3328	vdpu34x	vdpu	N/A	vepu	vepu22
RK3288	vdpu1_2160p	rk_hevc	N/A	vepu	N/A
RK3368	vdpu1_2160p	rk_hevc	N/A	vepu	N/A
PX5	vdpu1_2160p	rk_hevc	N/A	vepu	N/A
RK3326	vpdu	hevc_1080p	N/A	vepu	N/A
PX30	vpdu	hevc_1080p	N/A	vepu	N/A
RK312X	vpdu	hevc_1080p	N/A	vepu	N/A

1) RK3588 编解码能力推算

RK3588 带vdpu38x + vepu58x + rkjpegd + vdpu + vepu，因此支持：

- H264\H265\AVS2 8K 10bit 解码 [vdpu38x]
- H264\H265 8K 解码 [vepu58x]
- H.263、MPEG1/2/4、VP8、JPEG 1080P 解码 [vdpu + rkjpegd]
- VP8、JPEG 1080P 编码 [vepu]

2) RK3288 编解码能力推算

RK3288 带 vdpu1_2160p + vepu + rk_hevc，因此支持：

- H264 4K 8bit 解码 [vdpu1_2160p]
- H265 4K 10bit 解码 [rk_hevc]
- H.263、MPEG1/2/4、VP8、JPEG 1080P 解码 [vdpu1_2160p]
- H264、VP8、JPEG 1080P 编码 [vepu]

3) RK3399 编解码能力推算

RK3399 带 vdpu34x + vdpu + vepu，因此支持:

- H264\H265\VP9 4K 10bit 解码 [vdpu34x]
- H.263、MPEG1/2/4、VP8、JPEG 1080P 解码 [vdpu]
- H264、VP8、JPEG 1080P 编码 [vepu]

1.1.3 芯片编解码能力规格表

以下列举平台常见芯片编解码能力的标定规格。

测试最大规格与众多因素相关，因此可能出现不同芯片相同解码 IP 规格能力不同。

解码能力规格表:

	H264	H265	VP9	JPEG
RK3588	7680X4320@30f	7680X4320@60f	7680X4320@60f	1920x1088@200f
RK356X	4096x2304@60f	4096x2304@60f	4096x2304@60f	1920x1080@60f
RK3399	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1088@30f
RK3328	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1088@30f
RK3288	3840x2160@30f	4096x2304@60f	N/A	1920x1080@30f
RK3368	4096x2160@25f	4096x2304@60f	N/A	1920x1080@30f
PX5	4096x2160@25f	4096x2304@60f	N/A	1920x1080@30f
RK3326	1920x1088@60f	1920x1088@60f	N/A	1920x1080@30f
PX30	1920x1088@60f	1920x1088@60f	N/A	1920x1080@30f
RK312X	1920x1088@30f	1920x1088@60f	N/A	1920x1080@30f

编码能力规格表:

	H264	H265	VP8
RK3588	7680x4320@30f	7680x4320@30f	1920x1088@30f
RK356X	1920x1088@60f	1920x1088@60f	1920x1088@30f
RK3399	1920x1088@30f	N/A	1920x1088@30f
RK3328	1920x1088@30f	1920x1088@30f	1920x1088@30f
RK3288	1920x1088@30f	N/A	1920x1088@30f
RK3368	1920x1088@30f	N/A	1920x1088@30f
PX5	1920x1088@30f	N/A	1920x1088@30f
RK3326	1920x1088@30f	N/A	1920x1088@30f
PX30	1920x1088@30f	N/A	1920x1088@30f
RK312X	1920x1088@30f	N/A	1920x1088@30f

1.2 硬件单帧编解码处理时间

各个内核版本的调试日志开关:

```
4.19/5.10 内核 (Android 10.0 及以上版本)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
$ cat /proc/kmsg

4.4 内核 (Android 7.1 到 9.0)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/debug
$ cat /proc/kmsg
```

该命令输出内核单帧编解码的执行时间，常用于性能评估或卡顿、流畅性问题的分析。

1.3 VPU 频率查询与修改

评估编解码硬件的性能问题，常常需要操作 VPU 频率。通常认为，==当性能达到瓶颈时，提高 VPU 和 DDR 频率对硬件的编解码能力有一定提升。==

提高 DDR 频率请参考SDK RKDocs/common/DDR/Rockchip-Developer-Guide-DDR/ 目录下文档说明。本章主要介绍平台 VPU 频率操作。

[VPU 频率查询]

通过 cat /d/clk/clk_summary 查看系统时钟树，内核约定的各编解码 IP 频率节点名如下：

vdpu34x\vdpu38x 以 rkvddec 统称, vepu54x\vepu58x 以 rkvinc 统称

	vdpu	vepu	vepu22	rv_1080p	rk_hevc	rkvddec	rkvinc	rkjpegd
RK3588	vdpu	vdpu	×	×	×	rkvddec	rkvinc	jpeg_decoder
RK356X	vpu	jenc	×	×	×	rkvddec	rkvinc	jdec
RK3399	vcodec	vcodec	×	×	×	vdu	×	×
RK3328	vpu	h264	h265	×	×	rkvddec	×	×
RK3288	vcodec	vcodec	×	×	hevc	×	×	×
RK3368	video	video	×	×	video	×	×	×
PX5	video	video	×	×	video	×	×	×
RK3326	vpu	vpu	×	vpu	×	×	×	×
PX30	vpu	vpu	×	vpu	×	×	×	×
RK312X	vdpu	vdpu	×	vdpu	×	×	×	×

举例说明 - 以下场景所需频率获取:

1) RK3399 H264 1080P 解码

平台硬编解码通常选择更高等级的 IP，如 RK3399 H264 1080P 视频播放，有 vdpu 和 rkvddec 两个 IP 支持，选取 rkvddec IP，因此查询上面 rkvddec 频率节点名:

```
$ cat /d/clk/clk_summary | grep vdu      // <ac1k_vdu>
```

2) RK356X H264 编码

RK356X vepu 和 rkvinc 两个 IP 支持 H264 编码，选择更高等级 rkvinc，因此:

```
$ cat /d/c1k/c1k_summary | grep rkvinc // <ac1k_rkvinc>
```

3) RK3288 h264 解码

```
$ cat /d/c1k/c1k_summary | grep vcodec // <ac1k_vcodec>
```

4) RK3328 H265编码

```
$ cat /d/c1k/c1k_summary | grep h265 // <ac1k_h265>
```

[VPU频率修改]

4.4 内核 (Android 7.1 ~ 9.0) 频率修改参考如下方式, 配置 vpu 频率跑 500M 测试, 4.4 内核驱动版本不支持单独配置 IP 的频率.

```
--- a/drivers/video/rockchip/vcodec/vcodec_service.c
+++ b/drivers/video/rockchip/vcodec/vcodec_service.c
@@ -2307,6 +2307,7 @@ static void vcodec_set_freq_default(struct vpu_service_info
 *pservice,
 {
     enum VPU_FREQ curr = atomic_read(&pservice->freq_status);

+    reg->freq = VPU_FREQ_500M;
     if (curr == reg->freq)
         return;
```

4.19/5.10 内核 (>=Android 10.0) 频率配置参考如下方式, 配置 rkvidec 频率为 500M 测试

1) mpp_service 驱动使用 dtsi 方式配置频率信息

```
<RK3568.dtsi 配置说明>
// rockchip,normal-rates 为分辨率小于 1920x1088 时设置的 c1ock
// rockchip,advanced-rates 为分辨率大于 1920x1088 时设置的 c1ock

rkvidec: rkvidec@fdf80200 {
    clock-names = "ac1k_vcodec", "hc1k_vcodec", "c1k_cabac",
                  "c1k_core", "c1k_hevc_cabac";
    rockchip,normal-rates = <297000000>, <0>, <297000000>,
                          <297000000>, <400000000>;
    rockchip,advanced-rates = <400000000>, <0>, <400000000>,
                          <400000000>, <500000000>;
    rockchip,default-max-load = <2088960>; // 1920x1088
};
```

因此, 对于 RK3399 配置高性能解码器 rkvidec 500M 频率修改如下 (未配置默认 300M) :


```

--- a/arch/arm64/boot/dts/rockchip/rk3399.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3399.dtsi
@@ -1402,6 +1402,8 @@
                <&cru SCLK_VDU_CA>, <&cru SCLK_VDU_CORE>;
        clock-names = "ac1k_vcodec", "hc1k_vcodec",
                        "clk_cabac", "clk_core";
+
        rockchip,normal-rates = <500000000>, <0>,
+
        "500000000", <500000000>;
        resets = <&cru SRST_H_VDU>, <&cru SRST_A_VDU>,
                <&cru SRST_H_VDU_NOC>, <&cru SRST_A_VDU_NOC>,
                <&cru SRST_VDU_CA>, <&cru SRST_VDU_CORE>;

```

2) 测试性能 debug 时, 可以采用 debugfs 节点

```

$ echo 500000000 > /proc/mpp_service/rkvdec/ac1k
$ echo 500000000 > /proc/mpp_service/rkvdec/clk_core
$ echo 500000000 > /proc/mpp_service/rkvdec/clk_cabac

```

1.4 显示帧率

```

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

```

该命令可输出系统显示帧率统计, 统计方式为计算大于 500ms 时间间隔内送显的帧率。视频播放过程中卡顿、流畅性问题通常使用 FPS 来量化。

1.5 抓取编解码输入输出

对视频花屏、绿屏类问题, 抓取编解码的输入和输出有利于快速定位问题, 缩小问题范围。下面介绍平台框架从应用上层往下的一些抓取方式。

[应用上层]

1. 使用 MediaCodec API

- 编码情况:
 - 保存输入: queueInputBuffer 输入 buffer 前保存文件
 - 保存输出: dequeueOutputBuffer 后可以对编码输出 buffer 进行读写
- 解码情况:
 - 保存输入: queueInputBuffer 输入 buffer 前保存文件
 - 保存输出: ==未配置 configure surface 情况==可以在 dequeueOutputBuffer 后对解码输出 buffer 进行读写

[NuPlayer & MediaCodec & omx_il]

使用 MediaPlayer (Nuplayer 分支) 或 MediaCodec API 的情况可以用如下调试开关抓取:

```

#define DBG_RECORD_IN      0x01000000
#define DBG_RECORD_OUT    0x02000000

$ setenforce 0
$ mkdir /data/video/

// 解码 dec_in*.bin
$ setprop vendor.omx.vdec.debug 0x01000000

```

```
$ setprop record_omx_dec_in 1

// 编码enc_in*.bin enc_out*.bin
$ setprop vendor.omx.venc.debug 0x03000000
$ setprop record_omx_enc_in 1
$ setprop record_omx_enc_out 1
```

[MPP]

libmpp 是平台的通用编解码处理库，可以使用下面调试开关抓取：

```
#define MPP_DBG_DUMP_IN      0x00000200
#define MPP_DBG_DUMP_OUT    0x00000400

$ setenforce 0
$ mkdir /data/video/

$ setprop mpp_dump_in /data/video/mpp_dec_in.bin
$ setprop mpp_dump_out /data/video/mpp_dec_out.bin
$ setprop vendor.mpp_dump_in /data/video/mpp_dec_in.bin
$ setprop vendor.mpp_dump_out /data/video/mpp_dec_out.bin
$ setprop mpp_debug 0x600 && setprop vendor.mpp_debug 0x600

$ 开始测试然后将 /data/video/ 目录 pull 上传
```

1.6 抓取 USB Camera JPEG 解码输入输出

Android 10.0 及 10.0 之上的版本：

```
#define DEBUG_RECORD_IN      0x00000002
#define DEBUG_RECORD_OUT    0x00000004

$ setenforce 0
$ chmod 777 /data/video

$ setprop hwjpeg_dec_debug 0x00000006
$ setprop hwjpeg_enc_debug 0x00000006

$ 开始测试然后将 /data/video/ 目录 pull 上传
```

其他更小的 Android 版本请参考下面仓库的补丁添加：

```
https://github.com/ChenJinsen1/rk-media-patch
patch_file/cameraHal_dump_jpeg.patch
```

2. 编解码支持类

编解码支持类问题的讨论可能涉及芯片硬编解码能力，需要依赖章节 1.1<视频处理硬件基础> 的内容。

2.1 片源是否支持？

使用 Windows 和 Linux 提供的 MediaInfo 工具可以查询到音视频的媒体参数信息，包含编码格式、分辨率、码率、扫描方式、位深等基本信息。对比平台提供的芯片 DataSheet 手册或者 Codec BenchMark 可初步判断片源是否支持。

目前平台由于版权等原因明确已知不支持的音视频格式列举如下：

- 音频：mlp、ac3、eac3、dts、dsp、heaac、sbraac、杜比相关
- 视频：divx、rmvb、vp6、svq、iso蓝光



2.2 硬编解码最大支持路数？

根据实际情况，本章节问题的讨论分为多路编解码支持和多路同编同解支持。

2.2.1 多路编解码支持路数

芯片硬编码/解码最大支持路数换算涉及硬件 pixel 算力，以一个具体例子说明：

-> RK3399 最大能支持多少路H264 1080P@30fps规格解码？

查询 RK3399 规格表可知 H264 的解码能力为: 4096x2304@30fps

1. 硬件 pixel 算力：一秒钟 4096x2304x30
2. 换算 1080P@30fps，计算方式为：(4096x2304x30) / (1920x1088x30) = 4.5
3. 路数越多，折算损耗也越大，一般最后值计算为向下取整，故支持 4 路 1080P@30fps

其他 codec 可参考类似计算，需要注意的是以上计算依据为高码率极限片源，因此可保证任何情况下都支持 H264 4 路 1080P@30fps 解码。

RK3588、RK356X、RK3399、RK3328 搭建高性能解码器 rkvddec (H264、H265、VP9)，对 H264\H265\VP9 解码，极限情况下普通的测试片源有机会突破计算极限。

如 RK3399 有机会能达到 1080P@30fps 8 路解码，具体评估方式如下：

前提: 非高码率片源

- 1) 解码性能指标: VPU 驱动内核单帧解码时间

```
4.19/5.10 内核 (Android 10.0 及以上版本)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
$ cat /proc/kmsg
```

```
4.4 内核 (Android 7.1 到 9.0)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/debug
$ cat /proc/kmsg
```

要支持 8 路 1080P@30fps 需要的单帧解码时间为 (时间 / 总帧数) :

```
-> (1 x 1000) / (8 * 30) ≈ 4.16 ms
```

抛开解码通路上的时间损耗, 一帧的解码时间在 4ms 以内可以符合要求, 如果目前测试单帧解码时间达不到所需, 按下面方式评估改善频率信息, 通常提高 VPU 频率或 DDR 频率对硬编解码性能会有一定提升。

2) 频率信息

查看测试过程中的 VPU 频率及 DDR 频率, 通常认为硬编解码的性能模式为 DDR performance & VPU 频率 500M (最高 600M, 长时间可能会出现不稳定), 如果查看频率尚有改善空间, 可以尝试提高频率然后继续按步骤 2 查看内核解码时间是否能达到需求。

```
/* VPU 频率 */
$ cat /d/clk/clk_summary | grep vdu      <ac1k_vdu>      rk3399
$ cat /d/clk/clk_summary | grep rkvddec  <ac1k_rkvdec> rk3328\rk356x

/* DDR 频率 */
$ cat /sys/class/devfreq/dmc/cur_freq
```

3) 提高 VPU 频率

请参考章节 1.3 节的介绍提高 VPU 频率。

4) 提高 DDR 频率

```
echo performance > /sys/class/devfreq/dmc/governor // 将DDR频率设为performance
```

具体提高 DDR 频率请参考 RKDocs/common/DDR/Rockchip-Developer-Guide-DDR/ 文档说明

2.2.2 多路同编同解支持路数

监控或视频会议类产品经常需要评估 "同时跑 H264 1080P@30f 一路编码, 四路解码" 类似需求, 我们需要知道 VPU 编码和解码 IP 组合运行时, 存在两种情况:

1. 相互独立, IP 间硬件独立, 可以同时运行
2. vpu_combo, IP 间共享资源 (常见共享 ram), 需要分时运行

IP 相互独立时, 编解码硬件并行运行, 编解码的支持情况分别按上一节提到的多路支持场景分析即可。vpu_combo 的情况, 编解码 task 分时间片运行, 分析情况以内核解码时间为参考依据。

下面以具体的例子说明该问题: 同时跑 "同时跑 H264 1080P@30f 一路编码, 四路解码" 需求, 分别从 RK3288 和 RK3399 的角度分析能否支持。

1) 确认涉及格式同编同解使用的编解码 IP

默认会选择更高等级的 IP，如对 RK3399 H264 解码来说，根据章节 1.1 的介绍，rkvddec 和 vdpu 都能支持 H264 解码，但 rkvddec 是 4K 高性能解码器等级更高，优先选择 rkvddec 用于 H264 解码。

- 对 RK3288，选择 vdpu1_2160p 和 vepu 分别用于 H264 解码和编码
- 对 RK3399，选择 rkvddec 和 vepu 分别用于 H264 解码和编码

2) 确认编解码 IP 能否同时运行

查看 IP 是否存在 combo，可以通过查询 IP 对应的 dtsti 节点 taskqueue-node 是否是同一个。

```
vdpu: vdpu@fffb90400 {                                rkvddec: rkvddec@fffb80000 {
    rockchip,taskqueue-node = <1>;                      rockchip,taskqueue-node =
<0>;
};                                                         };

vepu: vepu@fffb90000 {                                rkvdenc: rkvdenc@fffb00000 {
    rockchip,taskqueue-node = <1>;                      rockchip,taskqueue-node =
<2>;
};                                                         };
```

taskqueue-node 是同一个，则说明存在 combo，即 IP 间存在资源共享，需要分时运行。

taskqueue-node 不是同一个，则说明编解码 task 可以并行，互不影响。

以下为常用芯片的编解码 IP combo 情况表（供查询）：

	vdpu	vepu	vepu22	rv_1080p	rk_hevc	rkvddec	rkvdenc	rkjpegd
RK3588	0	2	×	×	×	9	7	1
RK356X	2	2	×	×	×	4	3	1
RK3399	0	0	×	×	×	1	×	×
RK3328	0	0	2	×	×	1	×	×
RK3288	0	0	×	×	1	×	×	×
RK3368	0	0	×	×	0	×	×	×
PX5	0	0	×	×	0	×	×	×
RK3326	0	0	×	0	×	×	×	×
PX30	0	0	×	0	×	×	×	×
RK312X	0	0	×	0	×	×	×	×

- 对 RK3288，vdpu1_2160p 和 vepu node 为同一个值 0，所以 H264 同编同解分时运行
- 对 RK3399，rkvddec 和 vepu node 值分别为 1 和 0，所以 H264 同编同解可以并行，互不影响

所以对于 "RK3399 H264 1080P@30f 一路编码，四路解码" 需求，由于编解码可以并行，因此根据 vepu H264 编码 1080P@30fps 和 rkvddec 4096x2160@30fps 的最大规格，可以判断支持该需求。

对于 "RK3288 H264 1080P@30f 一路编码，四路解码" 需求，由于编解码分时运行，需要评估内核单帧处理时间能否满足要求 <1.2 章节查看硬件单帧处理时间>。对该问题由于 vepu 1080@30f 已经达到最大规格，再增加解码时间片消耗，故判断无法支持该需求。

3. 编解码应用设计参考

3.1 Android Media API

Google 原生 Android 为多媒体应用开发设计了两套接口，函数调用流程简单明了，能满足大部分编解码应用的设计要求，章节开始首先对这些原生接口做简要介绍。

[MediaPlayer]

MediaPlayer 自 API Level 1 以来就一直存在，它是一套纯粹的播放器接口，提供了一种简单的方式来播放音频和视频。它只注重 "play" 播放的过程，完全屏蔽掉解码通路过程中的细节。因此如果只是需要播放一个本地或服务器上的音视频文件，可以毫不犹豫的选择 MediaPlayer。

作为早期的设计接口，MediaPlayer 媒体格式支持受限的局限性也同样明显。原生 MediaPlayer 由于直接使用 Android MediaExtractor 作为媒体格式解析器，只支持 mp4\mkv\mpeg2 等视频封装格式解析。

Rockchip Media 平台实现了一套自己的播放器库代码，使用 FFmpeg AVFormat 作为媒体格式解析器，因此不会有该局限性（Rockchip 播放器库在 Android 10.0 之前为 Nuplayer，10.0 及之后为 RockitPlayer，相对应的 Google 原生播放器库实现为 Nuplayer）。

MediaPlayer 的使用方式简单，网上参考资料丰富，请自行查阅开发。

[MediaCodec]

相对于 MediaPlayer 接口 "播放器" 的特点，MediaCodec 是多媒体 "编解码器"，它是 Android 底层多媒体支持基础框架的一部分。与 MediaPlayer prepare\start\pause\stop 的使用方式相比较，它的使用方式更为繁琐，但同时为开发者带来了更多的创造性。

1. 作为编解码器组件，使用并遵循 OpenMax 框架，我们可以从 codec-name 区分软硬解/软硬编，"OMX.rk" 打头为 Rockchip 硬编解码器组件，"OMX.google" 或 "c2.android" 打头为 Google 的软编解码器组件。
2. MediaCodec 可以直接与音视频裸流打交道，可以自己控制部分编解码细节。
3. MediaCodec 作为编解码器组件，通常与 MediaExtractor\MediaSync\MediaCrypto\Surface\AudioTrack 等模块配合使用。除了与以上原生的组件，也可以使用 FFmpeg 的媒体格式解封器解决 MediaPlayer 媒体支持格式受限的问题，配合 OpenGL*\SurfaceTexture API 做一些纹理渲染、视频后处理等。

MediaCodec 的 API 及使用细节请参考 Google Developer 官网介绍：

<https://developer.android.com/reference/android/media/MediaCodec>

MediaCodec 设计有 Java \ Native(C++) 两层的用户接口，API 及使用方法都类似，Java 层应用设计可以参考 Google ExoPlayer，Native 层 Demo 查阅下一章节的介绍。

3.2 编解码器设计范例

github MediaExample 仓库提供了系统几个不同层级的编解码使用接口及范例，工程师可以根据自身需求选取做设计参考。源码地址：

<https://github.com/ChenJinsen1/mediaExample>

不同芯片硬编解码能力不同，使用前请先查阅确认芯片 DataSheet 手册确认是否支持。下面对该仓库做简单介绍，具体使用方法请参阅仓库 readme 说明。

MediaExample 为 Rockchip Andorid 平台硬编解码 demo 仓库，提供了系统几个不同层级的硬编解码使用接口及范例，包括：

1. native-codec - Android native 层 MediaCodec 使用范例
2. rkvpv-codec - 基于 libvpv 直接调用硬件编解码库使用范例
3. mpp-codec - 基于 Media Process Platform 使用范例及文档

3.2.1 native-codec

native-codec 为 Android native 层 MediaCodec 编解码接口使用范例，仓库目录中包含 native_dec_test\native_enc_test 测试程序。

[解码器 Demo]

native_dec_test 为解码器测试程序，目前设计支持解码带封装格式 (mp4\mkv) 或 raw 数据 (h264\h265) 输入，使用方式：

```
Usage: native_dec_test [options]
Android native mediacodec decoder demo.
- 1) native_dec_test --i input.264 --o out.yuv --w 1280 --h 720 --t 1
- 2) native_dec_test --i input.mp4 --t 2

Options:
--u
    Show this message.
--i
    input file
--o
    output bitstream files, if output file not specified,
    will render surface on the screen
--w
    the width of input picture
--h
    the height of input picture
--t
    input type:
        1: raw 264 or 265 format, default
        2: with container, use extrator to get decode_input
```

1. 默认为 raw 数据，如果要指定封装格式输入请带参数 (-t 2)
2. 默认将解码输出渲染到屏幕，可以直接通过 --o 参数指定输出文件将解码输出保存到文件
3. 如果指定 raw 数据输入并且不能保证 queueInputBuffer 送入完整的一帧数据，需要打开 vpu Split-Mode，使用内部的分帧处理，否则可能出现解码错误。修改方式参考如下：

hardware/rockchip/omx_il/component/video/dec/Rkvpu_OMX_vdec.c

Rkvpu_SendInputData -> p_vpu_ctx->init 初始化前加入下面的代码：

```
RK_U32 need_split = 1;
p_vpu_ctx->control(p_vpu_ctx,VPU_API_SET_PARSER_SPLIT_MODE, (void*)&need_split);
```

4. 目前该目录下的 demo 程序基于 Android 7.1 平台开发，如果要在高于 7.1 平台上移植，使用需要更新 MediaCodec 使用头文件，建议参考 SDK 目录 <frameworks/av/cmds> 下的程式头文件使用。

[编码器 Demo]

native_enc_test 为编码器测试程序，目前支持文件输入和输出。使用方式：


```
Usage: native_enc_test [options]
Android native mediacodec encoder demo.
- native_enc_test --i input.yuv --o out.h264 --w 1280 --h 720
Options:
--u
    Show this message.
--i
    input bitstream file
--o
    output bitstream files
--w
    the width of input picture
--h
    the height of input picture
--b
    the bitrate of encoder, default 3Mbps
--f
    the framerate of encoder, default 30fps
```

3.2.2 rkvpv-codec

Rockchip libvpu 硬编解码库中设计了 VpuApiLegacy 用户接口，使用 VpuCodecContext 对象可以方便的进行硬编解码接口调用，native-codec 目录为该用户接口的范例使用。

该硬编解码调用方式与目前 Android MediaCodec -> OMX -> libvpu 的硬解调用方式一致，omx_il 的设计实现代码中同样使用了该用户接口，相较于 native MediaCodec 接口，VpuApiLegacy 直接与底层编解码库交互，省去了通路中的时间消耗。

[VpuApiLegacy 一些重要概念说明]

- 1) vpu_open_context
获取 VpuCodecContext 对象，并完成一些回调函数的映射
- 2) vpu_close_context
关闭 VpuCodecContext，释放资源
- 3) VpuCodecContext->init
VpuCodecContext 初始化，宽、高、codingType 类型设置等
- 4) VpuCodecContext->flush
清空缓存中的数据
- 5) VpuCodecContext->control
配置一些模式，例如快速出帧模式、分帧处理模式等
- 6) VpuCodecContext->decode_sendstream | VpuCodecContext->decode_getframe
往解码器中异步存取数据
- 7) VpuCodecContext->encoder_sendframe | VpuCodecContext->encoder_getstream
往编码器中异步存取数据

[RKHWDecApi]

rkvpv_dec_api-RKHWDecApi 为可参考的 VpuApiLegacy 接口 decoder 设计，rkvpv_dec_test 为 RKHWDecApi 使用范例，可参考这两个文件进行硬解码器设计。使用方式：


```
Usage: rkvpv_dec_test [options]
Rockchip VpuApiLegacy decoder demo.
  - rkvpv_dec_test --i input.h264 --o out.yuv --w 1280 --h 720 --t 1

Options:
--u
    Show this message.
--i
    input file
--o
    output bitstream files
--w
    the width of input picture
--h
    the height of input picture
--t
    input picture type(h264 default):
        1: h264
        2: h265
```

1. 解码器输出 NV12 格式
2. 平台硬解码器只处理对齐过的 buffer，因此 RKHWDecApi 输出的 YUV buffer 也是经过对齐的，返回的 VPU_FRAME 句柄中，FrameWidth 为对齐过的 buffer 宽度 (horizontal stride)，DisplayWidth 为实际图像的大小，如果这两者不匹配也就是解码输入为非对齐的分辨率，直接显示可能出现绿边的情况，需要先经过外部裁剪才能正常显示。
3. VPU_FRAME 在解码库内部循环使用，在解码显示完成之后记得使用 deinitOutFrame 解除使用状态。

[RKHWEncApi]

rkvpv_enc_api-RKHWEncApi 为可参考的 VpuApiLegacy 接口 encoder 设计，rkvpv_enc_test 为 RKHWEncApi 使用范例，可参考这两个文件进行硬编码器设计。使用方式：

```
Usage: rkvpv_enc_test [options]
Rockchip VpuApiLegacy encoder demo(h264 default).
  - rkvpv_enc_test --i input.yuv --o out.h264 --w 1280 --h 720

Options:
--u
    Show this message.
--i
    input yuv file
--o
    output bitstream files
--w
    the width of input yuv
--h
    the height of input yuv
--f
    the framerate of encoder, default 30fps
--b
    the bitrate of encoder, default 3Mbps
```

相较于 native MediaCodec 接口，RKHWEncApi 直接与底层编码库交互(省去通路上的时间消耗)，并支持更多编码细节的控制。如 gop 长度、cabac 模式、profile level、RateControl 码率控制等。

3.2.2 mpp-codec

Rockchip 提供的媒体处理软件平台(Media Process Platform，简称 MPP)，是适用于所有芯片系列的通用媒体处理软件平台。MPP 是最底层的媒体中间件，直接与 vpu 内核驱动交互，无论是 native-codec 还是 rkvpv-codec 最后都需要与 mpp 交互。

MPP 为用户提供的统一媒体接口叫 MPI (Media Process Interface)，可以用于编解码设计。具体的代码设计请参考 mpp-codec/＜MPP 开发参考_v0.5.pdf＞

外部官方地址: <https://github.com/rockchip-linux/mpp>

外部个人地址: <https://github.com/HermanChen/mpp/>

MPI 设计参考: mpp_repository/test/mpi_dec_test.c (mpi_enc_test.c)

3.3 JPEG 硬编解码参考范例

由于 Google 原生 MediaCodec 通路不支持 JPEG 编解码, 同时 JPEG 编解码可能存在 EXIF 头信息的操作, 需要在 Rockchip 系统多媒体应用集成 JPEG 硬编解码功能可以使用 MPP JPEG 封装库 libhwjpeg 或直接参考使用 MPP MPI 接口。Hwjpeg 源码仓库如下, 具体使用方法请查阅仓库 readme 说明。

```
https://github.com/ChenJinsen1/libhwjpeg
```

[Hwjpeg概述]

Hwjpeg 库用于支持 Rockchip 平台 JPEG 硬编解码, 是平台 MPP 库 JPEG 编解码逻辑的封装。其中 MpijpegEncoder 类封装了硬编码相关操作, MpijpegDecoder 类封装了硬解码相关操作, 用于支持图片或 MJPEG 码流解码。工程代码使用 mk 文件组织, 在 Android SDK 环境下直接编译使用即可。

工程包含主要目录

- inc : 头文件
- src : Hwjpeg 库实现代码
- test: Hwjpeg 测试实例

[MpijpegDecoder]

MpijpegDecoder 类是平台 JPEG 硬解码的封装, 支持输入 JPG 图片及 MJPEG 码流, 同时支持同步及异步解码方式。

- decodePacket(char* data, size_t size, OutputFrame_t *aframeOut);
- decodeFile(const char *input_file, const char *output_file);

decodePacket & decodeFile 为同步解码方式, 同步解码方式使用简单, 阻塞等待解码输出, OutputFrame_t 为解码输出封装, 包含输出帧宽、高、物理地址、虚拟地址等信息。

- decode_sendpacket(char* input_buf, size_t buf_len);
- decode_getoutframe(OutputFrame_t *aframeOut);

decode_sendpacket & decode_getoutframe 用于配合实现异步解码输出, 应用端处理开启两个线程, 一个线程送输入 decode_sendpacket, 另一个线程异步取输出 decode_getoutframe。

1. HwJpeg 解码默认输出 RAW NV12 数据
2. 平台硬解码器只处理对齐过的 buffer, 因此 MpijpegDecoder 输出的 YUV buffer 经过 16 位对齐。
如果原始宽高非 16 位对齐, 直接显示可能出现底部绿边等问题, MpijpegDecoder 实现代码中 OUTPUT_CROP
宏用于实现 OutFrame 的裁剪操作, 可手动开启, 也可以外部获取 OutFrame 句柄再进行相应的裁剪。
3. OutFrame buffer 在解码库内部循环使用, 在解码显示完成之后使用 deinitOutputFrame 释放内存。

[MpijpegEncoder]

MpijpegEncoder 类是平台 JPEG 硬编码的封装, 目前主要有几个接口提供:

- encodeFrame(char *data, OutputPacket_t *aPktOut);

- `encodeFile(const char *input_file, const char *output_file);`

`encodeFrame` & `encodeFile` 为同步阻塞编码方式，`OutputPacket_t` 为编码输出封装，包含输出数据的内存地址信息。

- `encode(EncInInfo *aInInfo, EncOutInfo *aOutInfo);`

`encode` 输入数据类型为文件 `fd`，是为 `cameraHal` 设计的一套编码方式，输出 JPEG 图片含编码缩略图、APP1 EXIF 头信息等。

`OutputPacket_t buffer` 在编码库内部循环使用，在编码处理完成之后使用 `deinitOutputPacket` 释放内存

4. 播放流畅性问题

视频播放卡顿、声音卡顿、音视频不同步等可以归为流畅性问题。流畅性问题的分析需要依据显示帧率 FPS 及内核单帧解码时间，相关命令在 1 章节调试手段中已经有介绍。

音频解码输出通常时间很快，由于媒体框架需要对音视频做时间戳同步，音视频不同步或音频卡顿类问题通常是由于视频帧解码速度不够。

```
// 显示帧率FPS
$ setprop debug.sf.fps 1
$ logcat -c ;logcat | grep mFps

// 内核单帧解码时间
4.19/5.10 内核（Android 10.0 及以上版本）
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
$ cat /proc/kmsg

4.4 内核（Android 7.1 到 9.0）
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/debug
$ cat /proc/kmsg
```

对于视频播放不流畅，主要原因可以归结为以下几点，工程师遇到相关问题时可以参考以下方式进行排查：

1) 检查是否使用平台硬解

一些视频网站或应用程序，由于代码及使用 API 未知，可能使用软解来进行视频格式解码。因此这类问题卡顿发生可以先在 Logcat 日志中查找相应的视频格式，如果确定视频格式及规格都在芯片解码能力范围之内而未使用平台硬解，请先检查程序或网站是否存在配置用来控制使用硬解与否。

判断是否使用平台硬解可以键入查询内核单帧解码时间，经过内核硬件处理则会有值打印，说明当前走硬件解码，否则说明走软解。

2) 显示合成效率不够

视频播放存在两个动作：解码与显示，解码输出提交给显示做渲染，因此请先查询 1.1.3 节芯片编解码能力规格表，确定在解码能力范围内的片源用上面的命令查询下内核单帧解码时间及显示帧率。

如果单帧解码时间足够（30 帧源 33 ms 之内/60 帧源 16 ms 之内），而显示帧率不够，可先归纳为显示合成效率不够，此类问题原因通常有以下几点：

- 视频显示输出存在角度，也就是屏幕旋转输出
- 场景 Surface 层数过多，存在多个图层
- 视频格式不支持

如果确定是显示效率不够导致的不流畅，抓取以下日志并提交 redmine 指派对应的工程师处理。

```
// 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 51"
adb shell "logcat -c ;logcat" > hwc
```

3) 屏幕刷新率未正确配置

使用下面的命令检查屏幕刷新率是否正确配置，按屏幕参数一般需要配置为 60fps

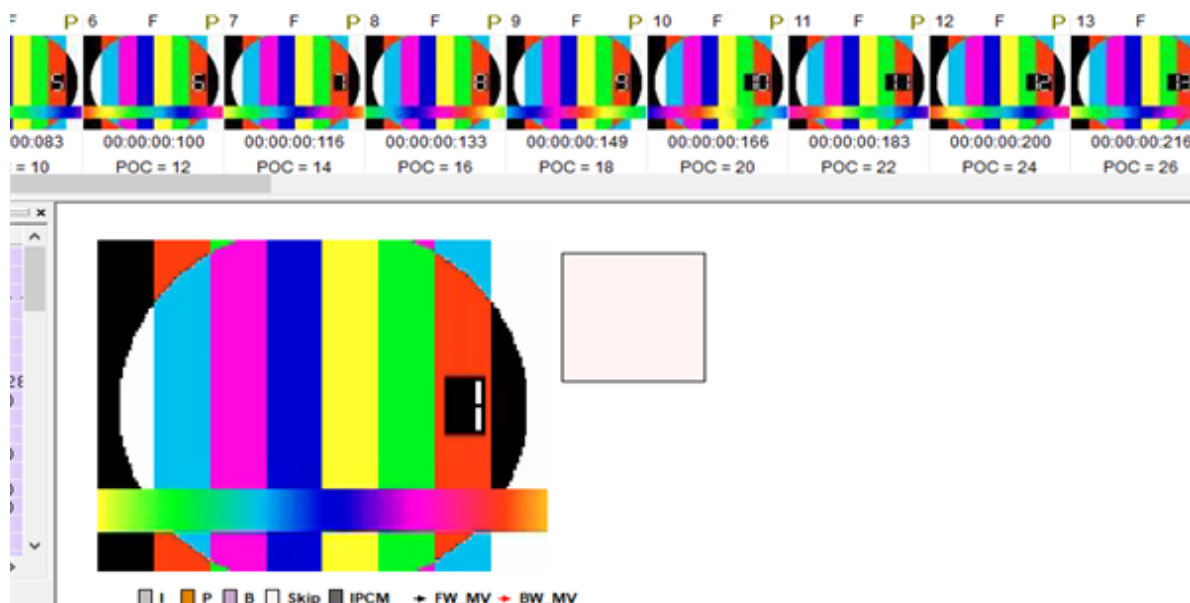
```
cat /d/dri/0/summary

rk3566_r:/ # cat /d/dri/0/summary
Video Port0: DISABLED
Video Port1: ACTIVE
Connector: DSI-1
    bus_format[100a]: RGB888_1x24
    overlay_mode[0] output_mode[0] color_space[0]
Display mode: 1080x1920p60
    clk[132000] real_clk[132000] type[48] flag[a]
H: 1080 1095 1097 1127
V: 1920 1935 1937 1952
```

4) 视频解码过程存在解码错误

日志中出现 "error frame" 相关的打印，则说明硬解过程出错，出现了丢帧。此类问题请先确认送给解码器的码流无误且无丢帧，媒体框架中提供了几种 Dump 解码输入的方式，请参考 1.5 命令进行操作。

抓取的输入为纯视频轨，可以使用 PC 工具查看分析，如 H264 格式可使用 eseye 或 Vega H264 Analyzer 分析 H265 格式使用 HEVCAnalyzer，通常这些工具可以判断出码流本身是否存在问题或丢帧。如下面的 vega 分析工具预览，POC值连续且预览无误则说明码流正常且无丢帧。



确认解码输入正确，则说明目前的解码框架对该码流适配出现问题，可提交对应的码流并附上 Logcat 日志提交 redmine 指派相应工程师处理。

5) 解码性能瓶颈

此类问题发生的场景通常在芯片解码能力规格边界，如 RK356X 4K@60F H264/RK3399 4K@60F H265 的高码率片源等，分析此类问题依据内核单帧解码时间，如果确认解码过程无异常，可适当抬频来满足片源需求。如果场景超过芯片解码能力规格，则判断为不支持不做分析，如 RK3399 4K@60F H264 片源。

-> a) 解码性能指标: VPU驱动内核单帧解码时间

参考所列调试命令查看硬件解码时间，片源流畅播放需保证解码时间保证足够(60 帧片源 16ms 内\30 帧片源 33ms 内)。

抛开解码通路上的时间损耗，一帧解码时间留 2ms 左右的余量，如 60 帧解码时间 14ms 内。如果目前测试单帧解码时间达不到所需，通常提高 VPU 频率或 DDR 频率对硬解码性能会有一定提升。

-> b) 频率信息

查看测试过程中的 VPU 频率及 DDR 频率，通常认为硬编解码的性能模式为 DDR performance 与 VPU 频率 500M（最高 600M，长时间可能会出现不稳定），如果查看频率尚有改善空间，可以尝试提高频率然后继续按步骤 1 查看内核解码时间是否能达到需求。

```
/* VPU 频率 */
$ cat /d/clk/clk_summary | grep vdu      <aclk_vdu>      rk3399
$ cat /d/clk/clk_summary | grep rkvddec  <aclk_rkvdec> rk3328\rk356x

/* DDR 频率 */
$ cat /sys/class/devfreq/dmc/cur_freq
```

-> c) 提高 VPU 频率

请参考章节 1.3 节的介绍提高 VPU 频率。

-> d) 提高 DDR 频率

```
echo performance > /sys/class/devfreq/dmc/governor // 将DDR频率设为performance
```

具体提高 DDR 频率请参考 RKDocs/common/DDR/Rockchip-Developer-Guide-DDR/ 文档说明。

5. 其他常见问题

5.1 编解码实例数量限制

场景：多路解码，使用 Android MediaCodec API

描述：目前 OMX 框架中由于稳定性需求限制了 MediaCodec 实例的数量，但是对于不同的平台理论上多路编解码如果硬件 pixer 算力足够，编解码的实例数量也无限制。参考下面的方式放开限制即可。

```
hardware/rockchip/omx_il$ git diff
diff --git a/component/common/Rockchip_OMX_Resourceanager.c
b/component/common/Rockchip_OMX_Resourceanager.c
index d33fe7a..efbb336 100755
--- a/component/common/Rockchip_OMX_Resourceanager.c
+++ b/component/common/Rockchip_OMX_Resourceanager.c
@@ -38,8 +38,8 @@
#include "git_info.h"

-#define MAX_RESOURCE_VIDEO_DEC 16 /* for Android */
-#define MAX_RESOURCE_VIDEO_ENC 8 /* for Android */
```

```

#define MAX_RESOURCE_VIDEO_DEC 20 /* for Android */
#define MAX_RESOURCE_VIDEO_ENC 20 /* for Android */

/* Max allowable video scheduler component instance */
static ROCKCHIP_OMX_RM_COMPONENT_LIST *gpVideoDecRMComponentList = NULL;

```

5.2 指定 SurfaceTexture 解码输出绿边问题

场景：使用 MediaCodec，configure 配置使用 SurfaceTexture，出现解码输出绿边

描述：VPU 只能处理对齐过的 buffer，因此框架申请的解码输出 buffer 都是经过对齐的，对齐方式可以参考下图OMX中的定义。

宽按如下方式对齐：

```

OMX_U32 Get_Video_HorAlign(OMX_VIDEO_CODINGTYPE codecId, OMX_U32 width, OMX_U32 height)
{
    OMX_U32 stride = 0;;
    if (codecId == OMX_VIDEO_CodingHEVC) {
        stride = ((width + 255) & (~255)) | (256);
    } else if (codecId == OMX_VIDEO_CodingVP9) {
        stride = (width + 127) & (~127);
    } else {
        stride = ((width + 15) & (~15));
    }
    if (access("/dev/rkvdec", 06) == 0) {
        if (width > 1920 || height > 1088) {
            if (codecId == OMX_VIDEO_CodingAVC) {
                stride = ((width + 255) & (~255)) | (256);
            }
        }
    }
    return stride;
}

```

高度按以下方式对齐：

```

OMX_U32 Get_Video_VerAlign(OMX_VIDEO_CODINGTYPE codecId, OMX_U32 height)
{
    OMX_U32 stride = 0;;
    if (codecId == OMX_VIDEO_CodingHEVC) {
        stride = (height + 7) & (~7);
    } else if (codecId == OMX_VIDEO_CodingVP9) {
        stride = (height + 63) & (~63);
    } else {
        stride = ((height + 15) & (~15));
    }
    return stride;
}

```

即解码输出存在如上所示的 stride，如 H264 1920x1080 格式申请 buffer 规格为 1920x1088，送显前需要先经过裁剪，否则高度会存在 8 个像素的绿边。

SurfaceView 与 SurfaceTexture 的区别在于 SurfaceView 走 Android nativeWindow -> surfaceFlinger 的限制框架，事先在框架中已经配置了裁剪参数 native_window_set_crop，因此框架显示时会做裁剪，所以显示是正常的。

SurfaceTexture 走 GL 纹理渲染，框架不会有裁剪的处理，需要应用上拿到解码输出后进行裁剪预处理。

设置方式请参考：

<https://github.com/ChenJinsen1/rk-media-patch>
patch_file/SurfaceTexture裁剪预处理

说明：

```
mSurfaceTexture.getTransformMatrix(this.transformMatrix);
```

Android的SurfaceTexture 类有标准的这个接口。

这个接口是根据 media 框架层给出的视频的虚宽和实宽等，计算出来的变换矩阵。GPU 拿到这个变换矩阵，实现裁剪的效果。

应用层需要做如下修改：

- 1、`mSurfaceTexture.getTransformMatrix(this.transformMatrix);` // 获取这个矩阵
- 2、将这个矩阵传递到应用层的 Shader 当中，将这个矩阵乘到所有的片段上。

5.3 编码模糊、马赛克问题

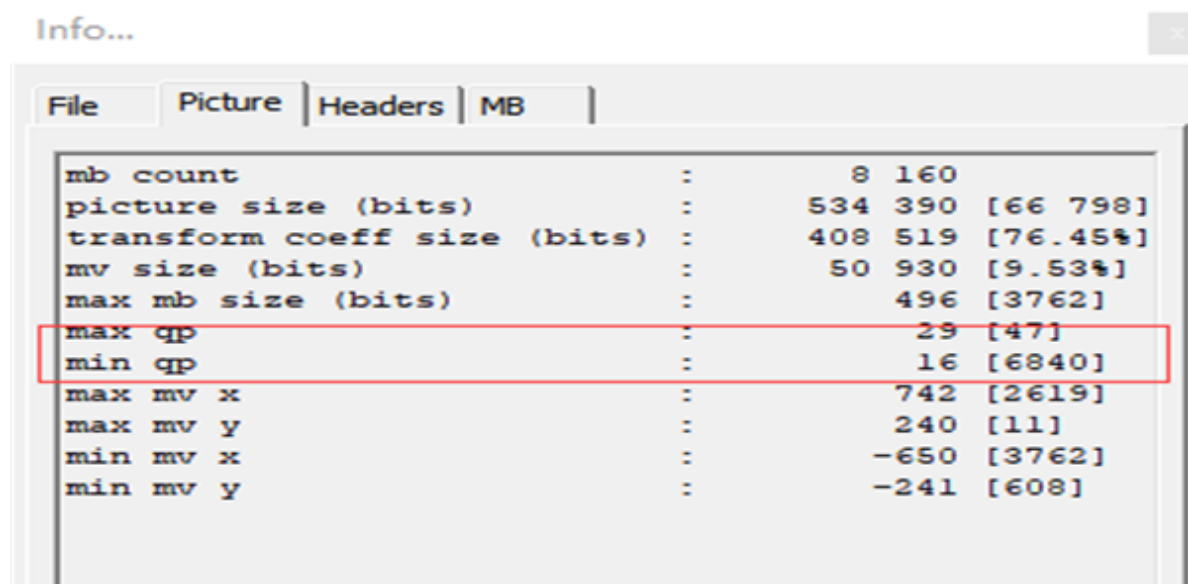
场景：录像或应用编码出现类似马赛克的模糊输出

描述：平台编码属于有损编码，模糊、马赛克通常是因为编码质量过低，而 QP 是编码量化参数，范围在 1~51，QP 值越小编码质量越高。

目前用户通常不会使用 codec 接口对 QP 进行限定，这种情况下 QP 范围由用户参数码率决定。码率越大，编码质量越高，单帧 QP 值就越小。所以编码马赛克问题通常排查步骤如下：

1) 检查编码输出的 QP 范围

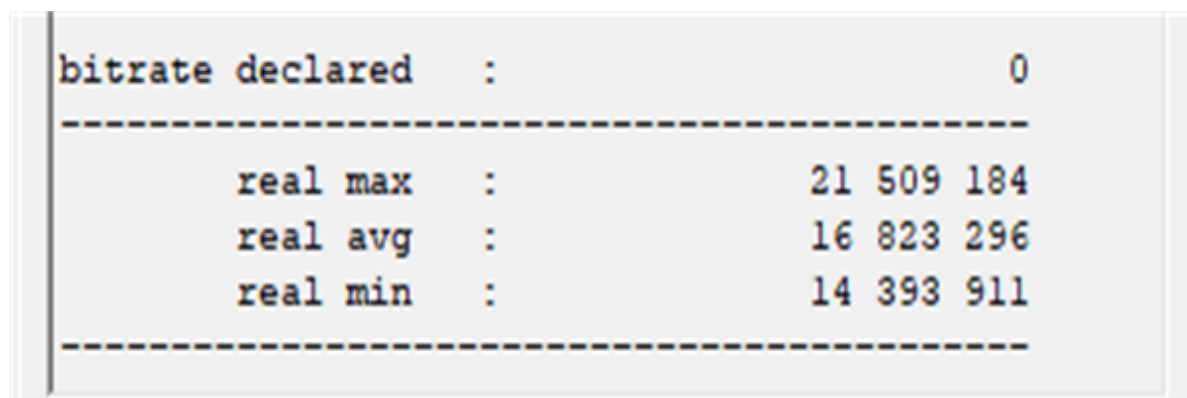
eseye 或 Vega H264 Analyzer 提供了单帧 QP 范围查询，如下图所示该帧的 QP 范围为 16~29，根据经验大于 40 QP 值的宏块可能会引起模糊、马赛克的出现，如果出现了 40 几的 QP 质量值，需要继续往下排查码率和编码质量策略等，否则就需要检查输入 yuv 是否本身就是带马赛克。



2) 检查码率的合理性

eseye 或 Vega H264 Analyzer 工具同样提供了文件输出码率的查询，如下图 eseye info 截图所示，该片源为动态码率模式，平均码率为 16M。

如果码率与应用设定的码率出入不大，且 QP 值出现过大的情况，则判定为用户设定码率过小，建议用户调大码率测试，否则可能需要调整 MPP 的 rc 控制策略。



bitrate declared	:	0

real max	:	21 509 184
real avg	:	16 823 296
real min	:	14 393 911

3) 调整 MPP QP 控制策略

MPI 提供了一系列编码参数配置接口，常用的就是对 QP 范围进行限定，使编码过程不至于出现 QP 质量过低的 case。如下所示：

```
diff --git a/mpp/legacy/vpu_api_legacy.cpp b/mpp/legacy/vpu_api_legacy.cpp
index 9a4f4067..1982af83 100644
--- a/mpp/legacy/vpu_api_legacy.cpp
+++ b/mpp/legacy/vpu_api_legacy.cpp
@@ -146,9 +146,9 @@ static MPP_RET vpu_api_set_enc_cfg(MppCtx mpp_ctx, M
     mpp_enc_cfg_set_s32(enc_cfg, "h264:cabac_idc", 0);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_init", -1);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_min", 10);
-    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max", 51);
+    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max", 40);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_min_i", 10);
-    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max_i", 51);
+    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max_i", 40);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_step", 4);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_delta_ip", 3);
 } break;
```

需要注意的是，调整 QP 范围可能会出现码率失控的情况，需要额外留意编码输出码率。

5.4 视频播放内存泄漏问题

视频播放内存泄漏问题的通用排查思路：

1) 确认是哪个进程内存泄漏

首先确认是定制应用还是系统媒体进程存在的泄漏，可以通过确认视频播放拷机前后 meminfo 信息，对比各个进程 RSS/PSS 内存占用，确认是哪个进程存在泄漏的情况。以下的排查步骤为系统进程内存泄漏，应用进程内存泄漏不在本节讨论范围。

```
dumpsys meminfo
```

2) 确认内存泄漏类型

明确泄漏类型有助于我们更好的定位问题点。视频播放类型主要可能存在的泄漏类型为 malloc 或者 dmabuf 类型，dmabuf 用于解码输出及显示，是零拷贝视频播放的基础，通过 dma buffer 的接口 (ION\DRM\dmaBufferHeap) 申请。malloc 就是 alloc、malloc 函数申请的内存。

Linux showmap 命令用于定位进程申请的内存 map 表。

```
showmap $(pidof xxx)
```

其中 RSS\PSS 为进程占用内存，单位为K，找出使用比较大的内存块，明确泄漏类型，如果是 dmaBuf 类型泄漏就主要检查视频播放初始化、销毁以及 info-change 时的 buffer 申请释放。如果是 malloc 内存泄漏，就需要 heapsnap backtrace 功能定位泄漏堆栈。

virtual size	RSS	PSS	shared clean	shared dirty	private clean	private dirty	swap	swa
113328	10628	6005	5352	228	1960	3088	0	

Map 内存 object 类型：

- /dmabuf: dma buffer 类型内存
- malloc 或 anon:scudo malloc 类型内存

3) Heapsnap 定位内存泄漏增长点堆栈

- setprop libc.debug.malloc.options backtrace
- kill <PID>
- mkdir /data/local/tmp/heap_snap
- /data/local/tmp/heapsnap -p <PID> -l /vendor/lib/libheapsnap.so
- kill -21 <PID>

拷机一段时间后抓取 heapsnap 快照，生成路径为 /data/local/tmp/。如果所示：定位到该位置存在 192 次调用，共 202669 内存未释放。

```
0 sz      192 num 202669 bt e88b39e2 e82b3a04 e82b2f3c e82af4dc e820fba4 e8209f50 e8245a38 e8241c38 e
#00 pc 000289e2 /apex/com.android.runtime/lib/bionic/libc.so (malloc+33)
#01 pc 000f1a04 /vendor/lib/libmmp.so (os_malloc+12)
#02 pc 000f0f3c /vendor/lib/libmmp.so (mmp_osal_malloc+76)
#03 pc 000ed4dc /vendor/lib/libmmp.so (mmp_mem_pool_get_f+208)
#04 pc 0004dba4 /vendor/lib/libmmp.so (mmp_frame_init+68)
#05 pc 00047f50 /vendor/lib/libmmp.so (mmp_buf_slot_set_prop+440)
#06 pc 00083a38 /vendor/lib/libmmp.so (vp9_parser_frame+11232)
#07 pc 0007fc38 /vendor/lib/libmmp.so (vp9d_parse+4)
#08 pc 0002c298 /vendor/lib/libmmp.so (mmp_dec_parser_thread(void*)+3464)
#09 pc 0007f7c2 /apex/com.android.runtime/lib/bionic/libc.so
#10 pc 000388b4 /apex/com.android.runtime/lib/bionic/libc.so
```

明确泄漏的代码堆栈，参考查找修复即可。