

Rockchip Linux5.10 Camera Trouble Shooting

文件标识: RK-PC-YF-A21

发布版本: V1.0.1

日期: 2024-05-14

文件密级: 公开资料

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 © 2024 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

本文记录**RKISP** 及 **Camera** 在调试过程中常见的一些问题与排查思路。

产品版本

芯片名称	内核版本
RV1106 / RV1103	Linux 5.10及以上

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	作者	版本	主要内容
2024-04-16	Ma Longchang	V1.0.0	初始版本
2024-05-14	Ruby Zhang	V1.0.1	部分语句表达修正

目录

Rockchip Linux5.10 Camera Trouble Shooting

1. Sensor点亮相关
 - 1.1 Sensor ID识别不到, I2C通讯失败
 - 1.1.1 什么是**7-bits**地址
 - 1.1.2 开机后, 测量不到 24M mclk 和 VDD 电源
 - 1.1.3 仍然测量不到 24M mclk
 - 1.1.4 检查 Sensor 的上电时序是否满足要求
 - 1.2 Sensor 驱动中的 `exp_def`、`hts_def`、`vts_def` 默认值是多少
 - 1.3 `link_freq` 与 `pixel_rate` 值应该是多少
 - 1.4 怎么才算点亮 Sensor
 - 1.5 i2ctransfer 工具的使用
 - 1.6 Sensor AVL 列表
 - 1.7 Sensor 驱动调试参考文档
2. MIPI / ISP 异常相关
 - 2.1 MIPI 需要设置哪些参数
 - 2.2 没有收到帧数据, 也没有看到 ISP/MIPI 有报错
 - 2.3 命令正确, select timeout 报错
 - 2.4 MIPI报错
 - 2.4.1 MIPI错误信息详细表
 - 2.4.2 如何处理 SOT/SOT_SYNC 错误
 - 2.4.3 如何处理 CRC/Checksum(CS)、ECC/ECC1/ECC2 错误
 - 2.4.4 如何处理 ERR_PROTOCOL/ERR_F_BNDRY 错误
 - 2.4.5 能正常收帧, 但偶现 MIPI 错误
 - 2.4.6 报很多 MIPI 错误甚至死机
 - 2.4.7 如何处理 ISP PIC_SIZE_ERROR
3. 获取图像相关
 - 3.1 有哪些方式可以抓图
 - 3.2 抓到的图颜色不对, 亮度也明显偏暗或偏亮
 - 3.3 什么是 ISP 的拓扑结构(topology, 链路结构), 如何使用 media-ctl 命令
 - 3.3.1 一个 ISP 怎样接多个 Sensor
 - 3.4 抓取 RAW 图是否与原图完全一致
 - 3.5 ISP 怎样双路(MP, SP)同时输出
 - 3.6 ISP 是否具有放大功能
 - 3.7 ISP 是否具有旋转功能
 - 3.8 怎样抓灰度 (GREY) 图
 - 3.9 如何区分MP、SP、BP
 - 3.10 图像分屏问题
 - 3.11 如何提高ISP频率
4. 3A相关
 - 4.1 如何确认 camera_engine_rkaiq 的版本
 - 4.1.1 如何确认 camera_engine_rkaiq 所需要的 rkisp kernel 驱动的版本号
 - 4.2 如何升级 camera_engine_rkaiq
 - 4.3 如何确认 3A 是否正在工作
 - 4.3.1 没有看到 rkisp_3A_server 进程
 - 4.3.2 rkisp_3A_server 是如何启动的
 - 4.3.3 如何确定 Sensor IQ 配置文件文件名及路径
 - 4.4 怎样手动曝光
 - 4.5 如何打开 librkaiq 的 log
5. 应用开发相关
6. 快速启动相关
 - 6.1 dts 修改
 - 6.2 kernel Sensor 驱动
 - 6.3 mcu rtt Sensor 驱动相关
 - 6.4 Sensor iq 文件
 - 6.5 问题处理

- 6.5.1 编译提示 “Not found main camera sensor config, ...”
 - 6.5.2 kernel 崩溃
 - 6.5.3 如何确认不带Sensor 镜头的出流效果
 - 6.5.4 kernel启动后一直打印 MIPI SIZE ERROR 错误
 - 6.5.5 Failed to stop decompress: decompress@ff520000
 - 6.5.6 Failed to stop decompress: decompress@ff520000, ret=-110
 - 6.5.7 如何设置快启应用不自启动
 - 6.5.8 如何确认 rtt 切到大图配置以后，是否提前出流
 - 6.5.9 如何快速定位快启阶段离线帧的问题
 - 6.5.10 如何抓取rtt阶段的小图，查看效果
 - 6.5.11 首帧图像颜色不正常
7. AOV 相关
- 7.1 如何支持 Sensor 硬件 standby 模式
 - 7.2 如何对补光灯进行控制
 - 7.3 AOV 开发参考文档

1. Sensor 点亮相关

1.1 Sensor ID 识别不到, I2C 通讯失败

Sensor ID 如果未识别到, 这与 RKISP 或 RKCIF 没有任何关系, 仅仅是 Sensor 上电时序未满足要求。

请按以下顺序排查:

1. Sensor 的 **7-bits** i2c slave id 是否正确, 是否误写成8-bits。
2. mclk 是否有输出, 电压幅度是否正确。mclk 一般是 24Mhz, 也有27Mhz。
3. Sensor 的上电时序是否满足要求, 主要包括 avdd, dovdd, dvdd, power down, reset 等。

1.1.1 什么是7-bits地址

8bits 中的最低位 (LSB) 表示 R/W, 高 7bits 即是我们需要的 i2c slave id。

1.1.2 开机后, 测量不到 24M mclk 和 VDD 电源

在 Sensor 驱动的实现中, 一般是只有在需要时才开启 mclk 及电源, 因此开机后 mclk 及电源默认是关闭的。

调试时, 可以将驱动中的 `power_off()` 函数的实现注释掉, 这样不会下电, 方便测量。

1.1.3 仍然测量不到 24M mclk

使用示波器时, 检查示波器的带宽是否足够, 建议至少 48M 以上的带宽。

1. Sensor 没有正确打开 mclk, 请参考如 `drivers/media/i2c/ov5695.c` 中对 mclk 的操作。
2. 该 gpio 被其它模块占用了, 这种情况时, 一般 kernel log 会有相应的提示。还可以通过 io 命令去查看 pin-ctrl 寄存器设置是否正确。

1.1.4 检查 Sensor 的上电时序是否满足要求

Sensor 的 Datasheet 中一般会详细描述每路电源的上电顺序及间隔要求, 请通过示波器检查是否满足。

有一些 Sensor 的电源 vdd 在上电时是没有时间先后要求的, 如 `ov5695`, 它的驱动中可能是用 `regulator_bulk` 来管理电源; 但有一些是有先后要求的, 如 `ov2685.c`, 它在驱动中是用多个 regulator 去分别控制, 具体如 `avdd_regulator`, `dovdd_regulator`。请根据实际情况选择。

1.2 Sensor 驱动中的 `exp_def`、`hts_def`、`vts_def` 默认值是多少

如果有 Sensor 原厂的联系方式, 请联系原厂获取。否则, 请从 Datasheet 中查找到对应的寄存器, 并从寄存器列表中找到初始化时配置的值即可。以 `ov2685.c` 为例:

```

#define OV2685_REG_VTS                                0x380e

...

    {0x380e, 0x05},
    {0x380f, 0x0e},

...

    .vts_def = 0x050e,

```

0x380e 与 0x380f 是 VTS 对应的寄存器，在初始化时配置的值是 0x050e，那么 `vts_def` 就是 0x050e。`exp` 与 `hts` 采用默认值，可直接从 Datasheet 中查找。

如果不期望应用程序去调节曝光、帧率时，可以不必要用到 `exp`, `hts`, `vts`。一般 RAW 格式的 Sensor 需要这三个参数。

1.3 `link_freq` 与 `pixel_rate` 值应该是多少

`link_freq` 指的是 MIPI clk 的实际频率。注意不是 24M 的 `mclk`，而是 **MIPI dn/dp clk**。优先通过原厂窗口查问，或查找 Datasheet 是否有相关的参数。

一般情况下，`link_freq` 实际值不会小于如下公式的计算结果，单位是(Hz)

```
link_freq = width * height * fps * bits_per_pixel / lanes / 2
```

如果实在不知道 `link_freq` 的实际值，可以用示波器测量。

`pixel_rate` 指的是每秒传输的像素个数，在 `link_freq` 确定下之后，可用以下公式计算：

```
pclk = link_freq * 2 * lanes / bits_per_pixel
```

1.4 怎么才算点亮 Sensor

首先需要能认到 Sensor id，即 i2c 的读写不能有异常。这时用 `media-ctl -p -d /dev/media0` 应该能够看到 Sensor 的具体信息，如名称、分辨率等。如下所示：

```

# media-ctl -p -d /dev/media0 | tail -n 30
    -> "rkcif_tools_id2":0 []
pad11: Source
    -> "stream_cif_mipi_id0":0 []
    -> "stream_cif_mipi_id1":0 []
    -> "stream_cif_mipi_id2":0 []
    -> "stream_cif_mipi_id3":0 []
    -> "rkcif_scale_ch0":0 []
    -> "rkcif_scale_ch1":0 []
    -> "rkcif_scale_ch2":0 []
    -> "rkcif_scale_ch3":0 []
    -> "rkcif_tools_id0":0 []
    -> "rkcif_tools_id1":0 []
    -> "rkcif_tools_id2":0 [ENABLED]

```

```

- entity 58: rockchip-csi2-dphy0 (2 pads, 2 links)
    type V4L2 subdev subtype Unknown flags 0
    device node name /dev/v4l-subdev1
    pad0: Sink
        [fmt:SBGGR10_1X10/2688x1520@10000/300000 field:none]
        <- "m00_b_sc450ai 4-0030":0 [ENABLED]
    pad1: Source
        -> "rockchip-mipi-csi2":0 [ENABLED]

- entity 63: m00_b_sc450ai 4-0030 (1 pad, 1 link)
    type V4L2 subdev subtype Sensor flags 0
    device node name /dev/v4l-subdev2
    pad0: Source
        [fmt:SBGGR10_1X10/2688x1520@10000/300000 field:none]
        -> "rockchip-csi2-dphy0":0 [ENABLED]

```

其次，上层抓图时，MIPI 要能输出数据，且不报 MIPI/ISP 相关错误，应用层能接收到帧。

1.5 i2ctransfer 工具的使用

Sensor 驱动作为 I2C 的设备，在调试过程中，难免会读取或者写入 Sensor 的寄存器值，这里介绍 i2ctransfer 工具的常用方法。

```
# i2ctransfer -f -y 4 w3@0x32 0x43 0x24 0x18
```

4: 代表 IIC 的总线号(对应的可为0,1,2,3...)

w: 代表写

3: 代表写 3 Byte

0x32: 代表 IIC 设备地址

后边三个数据代表要写的数据,这里0x4324假如为16bit的寄存器地址, 0x18为要写入的值。

```
# i2ctransfer -f -y 4 w1@0x30 0x08 r3
```

4: 代表 IIC 的总线号(对应的可为0,1,2,3...)

w: 表示写

1: 表示写 1 Byte

0x30: 代表 IIC 设备地址

r: 代表读取

3: 代表读取 3 Byte

这句代表的意思是从0x30地址上偏移0x08之后读取3个Byte, r后数字表示读取的个数。

其他 i2c-tools 的使用可参考博客文章: https://blog.csdn.net/qg_42952079/article/details/125217208。

注意: 不同平台Linux 的位数不同(32位、64位), 使用的 i2ctransfer 的版本不同。若没有此工具可自行下载或找开发人员获取。

1.6 Sensor AVL 列表

RGB Sensor AVL 位于https://redmine.rockchip.com.cn/projects/rockchip_camera_module_support_list/camera支持列表中显示了 Sensor 模组的详细信息。

如果是其它非 RGB Sensor，如 YUV Sensor，可以直接查看 kernel 源码的 drivers/media/i2c/ 目录，其中驱动作者是Rockchip 的，驱动都是有调试过的。

1.7 Sensor 驱动调试参考文档

RV1106 / RV1103 SDK 中调试新的 Sensor，可参考如下 驱动开发文档：/docs/zh/isp/<Rockchip_Driver_Guide_VI_CN_v1.1.5.pdf>。

2. MIPI / ISP 异常相关

Sensor 调试初期，比较经常碰到的几类问题是：

1. 没有收到帧数据，也没看到 ISP/MIPI 有报错
2. 看到 log 不停打印 MIPI 错误
3. ISP 报 PIC_SIZE_ERROR
4. 偶现 MIPI 错误
5. MIPI 不停报错，直至死机

2.1 MIPI 需要设置哪些参数

在 Sensor 与 ISP 之间 MIPI 通讯需要设置4个参数，请**务必**确认4个 MIPI 参数的正确性。

- Sensor 输出的分辨率大小
- Sensor 输出的图像格式，是 YUV 或 RGB RAW，8-bits、10-bits、或12-bits
- Sensor 的 MIPI 实际输出 link_freq
- Sensor 使用了几个 MIPI lane，这需要在 dts 中2个位置都配置正确

2.2 没有收到帧数据，也没有看到 ISP/MIPI 有报错

1. 确认 kernel log 中有没有关于 MIPI 的报错，比如用 `dmesg | grep MIPI` 看看有没有出错信息。
 2. 确认 kernel log 中有没有出现 Sensor 的 I2C 读写失败，如果 Sensor 在配置寄存器时失败了，Sensor 也可能没有正确初始化并使能输出。
 3. 实际量测下 MIPI 的 clk 及 data 线上有没有信息输出。如果没有，建议从 Sensor 初始化及硬件方面分析。
 4. 实际量测有 MIPI 信号输出，但没报错也收不到数据
- 请再次检查[2.1 MIPI需要设置哪些参数](#)，
 - 请确认 I2C 通讯没有错，Sensor 的寄存器初始化列表有全部写到 Sensor 中，
 - 在 Sensor 驱动中，最后使能 MIPI 输出的是 `s_stream()`，请确认在这个函数前，特别是 `s_power()`，不要让 MIPI 信号输出。这是因为在 `s_stream()` 前，MIPI 控制器还未实际准备好接收数据，如果在 `s_stream()` 前输出数据，可能导致 MIPI 协议头 SOT 信号丢失，
 - 也可以将 Camera Sensor 端 clock lane 由 continue 模式切换到 no continues。

2.3 命令正确，select timeout 报错

经常会出现抓 Raw 数据没数据返回，串口也没报错，抓图的时候报 select timeout 的错误。

此现象可以按照如下排查：

1. 查看DPHY的状态

根据TRM手册，查看dphy的stopstate 来判断是否有收到数，例如查看RK3588 CSI0：

```
io -4 -l 0x100 0xfdd30000
```

需要连续读取10次上述寄存器，正常有识别到 MIPI 信号的话，对应的stopstate会在0/1之间变化。

2. 测量 MIPI 信号
- 使用示波器测量是否有 MIPI 通道信号的输出。
3. 确认 sensor 寄存器
- 确认 sensor mipi ou t的寄存器是正常的，可以使用i2c工具读取，推荐使用 i2ctransfer。
4. 确定芯片的睡眠和复位引脚电平是否正常。

2.4 MIPI报错

2.4.1 MIPI错误信息详细表

针对RK3288/RK3399/RK3368， 错误信息表如下：

错误位（Bit）	简称	描述
25	ADD_DATA_OVFLW	additional data fifo overflow occurred
24	FRAME_END	正常收到一帧，不是错误
23	ERR_CS	checksum error
22	ERR_ECC1	1-bit ecc error
21	ERR_ECC2	2-bit ecc error
20	ERR_PROTOCOL	packet start detected within current packet
19:16	ERR_CONTROL	PPI interface control error occured, one bit per lane
15:12	ERR_EOT_SYNC	MIPI EOT(End Of Transmission) sync, one bit per lane
11:8	ERR_SOT_SYNC	MIPI SOT(Start Of Transmission) sync, one bit per lane
7:4	ERR_SOT	MIPI SOT(Start Of Transmission), one bit per lane
3:0	SYNC_FIFO_OVFLW	synchronization fifo overflow occurred, one bit per lane

针对RK3326/PX30/RK1808， 3个错误信息表如下：

ERR1 错误位（Bit）	简称	描述
28	ERR_ECC	ECC ERROR
27:24	ERR_CRC	CRC ERROR
23:20	ERR_FRAME_DATA	Frame 传输完毕，但至少包含一个CRC错误
19:16	ERR_F_SEQ	Frame Number 不连续不符合预期
15:12	ERR_F_BNDRY	Frame start与Frame end没有匹配
11:8	ERR_SOT_SYNC	MIPI PHY SOT(Start Of Transmission) sync error
7:4	ERR_EOT_SYNC	MIPI PHY EOT(End Of Transmission) sync error

ERR2 错误位 (Bit)	简称	描述
19:16	ERR_CONTROL	
15:12	ERR_ID	
11:8	ERR_ECC_CORRECTED	
7:4	ERR_SOTHS	PHY SOTHS error
3:0	ERR_ESC	PHY ESC error

常见的错误分析如下小章节。

2.4.2 如何处理 SOT/SOT_SYNC 错误

SOT (Start of Transmission) 和 SOT_SYNC (Start of Transmission Sync) 是在 MIPI (Mobile Industry Processor Interface) 接口中可能出现的错误类型之一。

SOT 信号需要符合 **MIPI_D-PHY_Specification**。如果需要深入分析，请直接从网上搜索该 pdf 文档，并建议重要参考：

- High-Speed Data Transmission
- Start-of-Transmission Sequence
- HS Data Transmission Burst
- High-Speed Clock Transmission
- Global Operation Timing Parameters

但一般来讲，Sensor 如果有在其它平台调通过，那么不符合 MIPI 协议的可能性比较小，建议客户：

- 首先向 Sensor 厂家确认该 Sensor 是否有实际成功使用过 MIPI 接口传输数据，
- 再次确认 **link_freq** 是否正确。因为 SOT 时序中的 Ths-settle 需要在 MIPI 接收端配置正确，所以 link_freq 很关键，
- 如果使用了多 lane，看 Sensor 原厂有没有办法修改成 1 lane 传输。
- 检查物理连接：确保 MIPI 接口的物理连接良好。检查线缆、接头和连接器是否松动、损坏或有不良接触。物理连接问题可能导致数据传输错误和通信中断。
- 验证电源供应：检查 MIPI 接口的供电稳定性。确保电源线路连接正常，供电电压水平符合规范要求。电源问题可能导致通信错误和协议异常。
- 调整时序参数：MIPI 接口的时序参数对通信稳定性至关重要。尝试调整时钟频率、数据线延迟等参数，以获得更稳定的通信。这可能需要参考设备规格和厂商建议，进行适当的优化和调整。
- 检查协议设置：确保 MIPI 接口的协议设置正确，并与设备之间的通信协议匹配。这包括时钟频率、数据线延迟、通信模式等方面的设置。参考 MIPI 接口的规范和相关文档，确保协议配置符合要求。
- 分析错误日志：查看系统或设备的错误日志，以了解更多关于 SOT 和 SOT_SYNC 错误的详细信息。错误日志可能包含有关错误类型、位置和时间戳等信息，有助于定位问题。
- 调试工具和设备：使用 MIPI 接口调试工具和设备，如逻辑分析仪、协议分析仪或信号发生器，来监测和分析 MIPI 接口的信号和通信过程。这些工具可以提供更深入的调试能力，帮助定位和解决 SOT 和 SOT_SYNC 错误。

需要注意的是，SOT 和 SOT_SYNC 错误可能由多种原因引起，包括物理连接问题、电源供应问题、协议设置不正确等。因此，解决问题的方法可能因具体情况而异。在排查过程中，综合考虑硬件、软件和通信方面的因素，并进行逐步排查和验证，有助于定位和解决这些错误。

2.4.3 如何处理 CRC/Checksum(CS)、ECC/ECC1/ECC2 错误

出现了 ECC 错误，CS 检验错误，说明数据在传输时不完整。建议：

- 优先排查硬件信号。
- 如果使用了多lane，看 Sensor 原厂有没有办法修改成 1 lane 传输。因为多 lane 之间没有同步好，也有可能出现ECC 错误。
- 检查物理连接：确保 MIPI 接口的物理连接正常，包括线缆、接头和连接器。检查是否存在松动、损坏或不良接触等问题。
- 验证电源供应：确保 MIPI 接口的供电稳定。检查电源线路是否正常连接，电压水平是否符合规范要求。
- 检查时序配置：MIPI 接口的正确操作需要正确配置时序参数，如时钟频率、数据线延迟等。请确保时序配置与设备要求一致，并且在正常范围内。
- 检查协议设置：MIPI 接口使用不同的协议，如 MIPI D-PHY 或 MIPI C-PHY。确保协议设置正确，并与设备之间的通信协议匹配。
- 分析错误日志：查看系统或设备的错误日志，以了解更多关于 ECC 错误的详细信息。错误日志可能包含有关错误类型、位置和时间戳等信息，有助于定位问题。
- 调试工具和设备：使用 MIPI 接口调试工具和设备，如逻辑分析仪、协议分析仪或信号发生器，来监测和分析 MIPI 接口的信号和通信过程。这些工具可以提供更深入的调试能力，帮助定位 ECC 错误的原因。
- 咨询设备厂商或技术支持团队：如果以上步骤无法解决问题，可以咨询 MIPI 接口相关的设备厂商或技术支持团队，寻求他们的帮助和建议。他们通常具有更深入的了解和专业知識，可以提供针对特定设备和应用的解决方案。

需要注意的是，ECC 错误可能由多种原因引起，包括硬件故障、信号干扰、配置错误等。因此，解决问题的方法可能因具体情况而异。在排查过程中，综合考虑硬件、软件和通信方面的因素，并进行逐步排查和验证，有助于定位和解决 ECC 错误。

2.4.4 如何处理 ERR_PROTOCOL/ERR_F_BNDRY 错误

该错误说明没有收到预期的 EOT/SOT。SOT,EOT 应该成对匹配出现。建议实测波形检查。

- 检查协议设置：确保 MIPI 接口的协议设置正确，并与设备之间的通信协议匹配。这包括时钟频率、数据线延迟、通信模式等方面的设置。参考 MIPI 接口的规范和相关文档，确保协议配置符合要求。
- 验证电源供应：检查 MIPI 接口的供电稳定性。确保电源线路连接正常，供电电压水平符合规范要求。电源问题可能导致通信错误和协议异常。
- 检查物理连接：确保 MIPI 接口的物理连接良好。检查线缆、接头和连接器是否松动、损坏或有不良接触。物理连接问题可能导致数据传输错误和通信中断。
- 调整时序参数：MIPI 接口的时序参数对通信稳定性至关重要。尝试调整时钟频率、数据线延迟等参数，以获得更稳定的通信。这可能需要参考设备规格和厂商建议，进行适当的优化和调整。
- 分析错误日志：查看系统或设备的错误日志，以了解更多关于 ERR_PROTOCOL 和 ERR_F_BNDRY 错误的详细信息。错误日志可能包含有关错误类型、位置和时间戳等信息，有助于定位问题。
- 调试工具和设备：使用 MIPI 接口调试工具和设备，如逻辑分析仪、协议分析仪或信号发生器，来监测和分析 MIPI 接口的信号和通信过程。这些工具可以提供更深入的调试能力，帮助定位和解决 ERR_PROTOCOL 和 ERR_F_BNDRY 错误。

需要注意的是，ERR_PROTOCOL 和 ERR_F_BNDRY 错误可能由多种原因引起，包括协议不匹配、物理连接问题、时序参数设置不正确等。因此，解决问题的方法可能因具体情况而异。在排查过程中，综合考虑硬件、软件和通信方面的因素，并进行逐步排查和验证，有助于定位和解决这些错误。

2.4.5 能正常收帧，但偶现 MIPI 错误

如果是 MIPI 错误，参考前面的错误描述。与信号相关建议从硬件信号上分析。

特别地，如果 MIPI 错误只在刚开始抓图时有，有可能是 Sensor 在上电的过程中 MIPI 信号有输出但并不符合协议，从而报错。

这种情况下，可以尝试按如下流程修改：

- 将完整的 Sensor 寄存器的初始化放到 `s_power()` 中。
因为此时 MIPI 接收端尚未开始接收数据，会忽略所有数据。
- 在 `s_power()` 函数的最后，关闭sensor的输出，即相当于调用了 `stop_stream()`
- 在 `start_stream()` 与 `stop_stream()` 中，仅打开或关闭 MIPI 的输出。

2.4.6 报很多 MIPI 错误甚至死机

这可能是[2.4.5 能正常收帧，但偶现MIPI错误](#)的更坏的情况。

碰到过这样的现象，其原因是 MIPI 信号不符合要求，而且 MIPI 接收端某些错误是电平中断，导致中断风暴并最终死机。

可以尝试按[2.4.5 能正常收帧，但偶现MIPI错误](#)的方法看是否有效。

2.4.7 如何处理 ISP PIC_SIZE_ERROR

Picture size error 是 ISP 级的错误，它提示未接收到预期的行数，列数。因此从各级的分辨率大小检查。

如果前级（即MIPI）有报错，应该先解决 MIPI 错误。

请从如下几点检查：

- DDR 频率是否太小。当 DDR 频率太低时，响应速度不够时，也会出现该错误。尝试将 DDR 定频到最高频率看还会不会出错：

```
echo performance > /sys/class/devfreq/dmc/governor
```
- 整个 ISP 链路中，有没有出现后级比前级的分辨率还大的情况。可以用 `media-ctl -p -d /dev/media0` 去查看拓扑结构。
分辨率应该要满足 `Sensor == MIPI_DPHY >= isp_sd input >= isp_sd output`。如果您没有手动修改过，默认应该是满足这个条件的。
- Sensor 的输出分辨率大小是否正确。尝试在驱动代码中将分辨率强制改小。比如ov7251.c中默认分辨率是640x480，

```
static const struct ov7251_mode supported_modes[] = {
    {
        .width = 640,
        .height = 480,
```

将 width, height 都改小些，比如 320x240，寄存器的配置不用改。这是为了确认 Sensor 的配置大小会不会超过实际输出的大小。

```
static const struct ov7251_mode supported_modes[] = {
    {
        .width = 320,
        .height = 240,
```

3. 获取图像相关

这部分主要涉及与抓图相关的常见问题。

3.1 有哪些方式可以抓图

RKISP 及 RKCIF 驱动支持 v4l2 接口，获取图像可以使用：

- v4l-utils 包中的 v4l2-ctl 工具获取图像。在调试过程中，建议首先使用该工具检验能否成功出图。
v4l2-ctl 抓图保存成文件，它不能解析图像并显示出来。如需要解析，Ubuntu/Debian环境下可以使用 mplayer，Windows 下可以使用如 7yuv 等工具。
对 v4l2-ctl, mplayer 工具的详细说明，请参考《Rockchip_Developer_Guide_Linux_Camera_CN.pdf》。v4l2-ctl也自带有详细的 `v4l2-ctl --help` 文档。
- 使用 rockit 多媒体库中提供的 demo bin 二进制程序取图保存。

常见命令有：

取 raw 图

```
rk_mpi_vi_test -w 1920 -h 1080 -d 0 -c 0 -m 0 -l 10 -n /dev/video0 -f 131076
```

从 mainpath 通道取 yuv 图，并保存在文件中。文件目录 /data/test_0_0_0.bin, 可使用 7yuv等工具打开。文件命名根据 dev id, pipe id, chn id相关。

```
rk_mpi_vi_test -w 1920 -h 1080 -d 0 -c 0 -m 0 -l 10 -o 1
```

从 mainpath 通道取 yuv 图，并并编码，写文件。文件保存在/data/venc_0.bin, 文件命名跟 chn id 有关。

```
rk_mpi_vi_test -w 1920 -h 1080 -d 0 -c 0 -m 1 -l 10 -o 1
```

...

其他 demo 可参考 rockit 多媒体开发文档，查看demo help帮助命令来使用。

- 实时预览。开发中，需要实时预览取流效果时，可使用 SDK中提供的 simple demo。

使用 simple_vi_bind_venc_rtsp demo, 通过rtsp 实时预览(使用vlc 或 potplayer工具配置板端IP)。

```
simple_vi_bind_venc_rtsp -I 0 -w 1920 -h 1080 (rtsp://ip/live/0)
```

3.2 抓到的图颜色不对，亮度也明显偏暗或偏亮

需要根据 Sensor 分情况：

1. Sensor 是 RAW RGB 的输出，如 RGGB、BGGR 等，需要 3A 正常跑起来。可以参考[4.3A相关](#) 3A 确认正常在跑时，请再次检查解析/显示图像时使用的格式是否正确，uv分量有没有弄反。
 2. Sensor 是 yuv 输出，或 RGB如RGB565、RGB888，此时 ISP 处于bypass 状态，
- 如果颜色不对，请确认sensor的输出格式有没有配置错误，uv分量有没有弄反。确认无误时，建议联系Sensor原厂

- 如果亮度明显不对，请联系 Sensor 原厂

3.3 什么是 ISP 的拓扑结构(topology, 链路结构)，如何使用 media-ctl 命令

RKISP 或 RKCIF 可以接多个的 Sensor，分时复用；同时 RKISP 还有多级的裁剪功能。因此用链接的方式将各个节点连接，并可通过 media-ctl 分别配置参数。关于 media-ctl 的使用，在《Rockchip_Developer_Guide_Linux_Camera_CN.pdf》文档中有较完整的描述。

3.3.1 一个 ISP 怎样接多个 Sensor

可以接多个 Sensor，但只能分时复用。通过配置 dts，将多个 Sensor 链接到 MIPI DPHY 后，可通过 media-ctl 切换 Sensor。

3.4 抓取 RAW 图是否与原图完全一致

当 ISP 以 bypass 模式获取 Sensor RAW 图（如RGGB, BGGR）时，需要 8 bit 对齐，不足8 bit 会低位填充 0，即

- 如果是 8bit, 16bit 的原图，应用获取到的是原图，没有填充
- 如果是 10bit, 12bit 的原图，会每个像素低位补0到16bit

只有 MP 对应的 video 设备可以出 RAW 图，SP 不支持 RAW 图输出的。

3.5 ISP 怎样双路(MP, SP)同时输出

RKISP 有 SP, MP 两路输出，即 Sensor 出来一张图像，SP, MP 可以分别对该图像做裁剪、格式转换，并可同时输出。

SP, MP 具有不同的视频处理能力，详细请参考《Rockchip_Developer_Guide_Linux_Camera_CN.pdf》。

只有当 SP, MP 都输出 RGB 或 YUV 时才可以同时输出。如果 MP 输出 RAW 图，那么 SP 不可以出图。

3.6 ISP 是否具有放大功能

硬件上有该功能，但不建议使用，驱动中也是默认关闭该功能。

3.7 ISP 是否具有旋转功能

没有。如果需要使用旋转功能，建议：

- 如果是 flip, mirror，首先查看 Sensor 是不是有该功能，如果有，直接使用。这样效率最高
- 如果无法使用 Sensor flip, mirror，考虑使用 RGA 模块，它的代码及 demo 位于external/linux-rga/目录，且有相关文档位于 docs/目录下

3.8 怎样抓灰度 (GREY) 图

只要 ISP 可以输出 YUV、或者 Sensor输出是 Y8灰度图时，应用程序总是可以使用 V4L2_PIX_FMT_GREY(FourCC为GREY) 格式直接获取图像。

3.9 如何区分MP、 SP、 BP

可通过 `media-ctl -p -d /dev/media0` (如有多个 media 设备，也尝试下 `/dev/media1`, `/dev/media2`) 去查看拓扑结构，如下截取部分输出：

```
# media-ctl -p -d /dev/media0
...
- entity 2: rkisp1_mainpath (1 pad, 1 link)           //表示该entity是
MP(MainPath)
    type Node subtype V4L flags 0
    device node name /dev/video1                     //对应的设备节点
是/dev/video1
    pad0: Sink
        <- "rkisp1-isp-subdev":2 [ENABLED]

- entity 3: rkisp1_selfpath (1 pad, 1 link)           //表示该entity是
SP(SelfPath)
    type Node subtype V4L flags 0
    device node name /dev/video2                     //对应的设备节点
是/dev/video2
    pad0: Sink
        <- "rkisp1-isp-subdev":2 [ENABLED]
...
```

少数情况下如果没有media-ctl命令，可以通过/sys/节点查找，如：

```
# grep '' /sys/class/video4linux/video*/name
/sys/class/video4linux/video0/name:stream_cif
/sys/class/video4linux/video1/name:rkisp1_mainpath    # MP节点对应/dev/video1
/sys/class/video4linux/video2/name:rkisp1_selfpath    # SP节点对应/dev/video2
/sys/class/video4linux/video3/name:rkisp1_rawpath
/sys/class/video4linux/video4/name:rkisp1_dmapath
/sys/class/video4linux/video5/name:rkisp1-statistics
/sys/class/video4linux/video6/name:rkisp1-input-params
```

RV1106 / RV1103 平台下, 还有BP、MP的下采样通道、BP的下采样通道。

```
# media-ctl -p -d /dev/media1
...
- entity 6: rkisp_mainpath (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video11
    pad0: Sink
        <- "rkisp-isp-subdev":2 [ENABLED]

- entity 12: rkisp_selfpath (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video12
```



```

    pad0: Sink
        <- "rkisp-isp-subdev":2 [ENABLED]

- entity 18: rkisp_bypasspath (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video13
    pad0: Sink
        <- "rkisp-isp-subdev":2 [ENABLED]

- entity 24: rkisp_mainpath_4x4sampling (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video14
    pad0: Sink
        <- "rkisp-isp-subdev":2 [ENABLED]

- entity 30: rkisp_bypasspath_4x4sampling (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video15
    pad0: Sink
        <- "rkisp-isp-subdev":2 [ENABLED]

...

```

3.10 图像分屏问题

现象：当 MIPI 通道受到干扰时，便会出现分屏问题，复现也很简单，只需要对 MIPI 通道的数据线或时钟线进行干扰即可分屏，并且每次分屏的位置都不同。

解决办法：这里存在两种情况

- 启动时分屏，当设备启动时分屏，这样的情况是因为获取图片之前没有对设备进行复位（可能软复位和硬复位都要执行），RN6725V1 的复位如下图所示：

```

ret = rn6752_write(client, 0x80, 0x31);
usleep_range(200, 500);
ret |= rn6752_write(client, 0x80, 0x30);
if (ret)
{
    dev_err(&client->dev, "rn6752 soft reset failed\n");
    return ret;
}

```

- 运行时分屏，当图像正常运行后，受到硬件部分的干扰也会导致分屏，例如：接在 CSI0 会出现分屏，而接在 CSI1 不会分屏，是因为他的 CSI0 通道经过了 VICP 模块，CSI1 直接接到了 ISP 模块上。

这是因为从 VICP 没有开启图像异常检测功能导致的，这样的现象只需要增加 CIF 通道的异常检测功能即可，具体的操作见 VICAP 异常复位的设置。

3.11 如何提高ISP频率

RV1106/RV1103 平台上，为了解决某些场景下出流慢，ISP 处理速度慢，系统卡顿等情况时，需要提高 ISP 频率，可使用如下操作：

```
# cat /proc/clock/summary | grep isp
      clk_core_isp3p2      1      1      0  339428572      0      0
50000
      aclk_isp3p2      1      1      0  339428572      0      0
50000
      hclk_isp3p2      1      1      0  148500000      0      0
50000
      isp0clk_vicap      2      2      0      0      0      0
50000
```

set clk rate:

```
echo [clk_name] [rate(Hz)] > /proc/clock/rate
```

如下:

```
# echo clk_core_isp3p2 420000000 > /proc/clock/rate

# cat /proc/clock/summary | grep isp
      aclk_isp3p2      1      1      0  339428572      0
0  50000
      hclk_isp3p2      1      1      0  148500000      0
0  50000
      clk_core_isp3p2      1      1      0  420000000      0
0  50000
      isp0clk_vicap      2      2      0      0      0
0  50000
```

4.3A相关

如果 Sensor 需要 3A tuning，如 Sensor 输出格式 RGGB, BGGR 等这样的 RAW BAYER RGB 格式，那么需要 RKISP 提供图像处理。

根据 camera_engine_rkaiq 版本的不同，3A 处理方式有差别。建议尽量将 camera_engine_rkaiq 升级到最新的版本。

请首先确认该模组是否在支持列表中，

- 已经在支持列表中的，media/isp/camera_engine_rkaiq/rkaiq/iqfiles/ 目录下会有一份对应的 json 文件
- 否则请向业务窗口发起模组调试申请

4.1 如何确认 camera_engine_rkaiq 的版本

从源码中查看

```
#grep RK_AIQ_VERSION_REAL media/isp/camera_engine_rkaiq/rkaiq/RkAiqVersion.h

#define RK_AIQ_VERSION_REAL_V "v5.0x5.0"
```

4.1.1 如何确认 camera_engine_rkaiq 所需要的 rkisp kernel 驱动的版本号

camera_engine_rkisp 对 kernel 驱动版本有要求，需要保证 rkisp 驱动足够新。

- 从 kernel 源码中查看 ISP 驱动版本

```
# grep RKISP_DRIVER_VERSION drivers/media/platform/rockchip/isp/version.h
```

- 从 kernel log 中查看 ISP 驱动版本

```
# dmesg | grep "version"

dmesg | grep "version"
[ 0.848252] udevd[65]: starting version 3.2.7
[ 3.889404] imx415 4-001a: driver version: 00.01.08
[ 3.967388] os04a10 4-0036: driver version: 00.01.05
[ 4.084418] sc4336 4-0030-3: driver version: 00.01.01
[ 4.114867] sc3336 4-0030-1: driver version: 00.01.01
[ 4.152066] sc530ai 4-0030: driver version: 00.01.01
[ 4.180572] sc200ai 4-0030-6: driver version: 00.01.09
[ 4.237776] rkCIF rkCIF-mipi-lvds: rkCIF driver version: v00.02.00
[ 4.260419] rkisp rkisp-vir0: rkisp driver version: v02.05.00
```

4.2 如何升级 camera_engine_rkaiq

包含有三部分

1. camera_engine_rkaiq 本身

位于SDK 的 media/isp/camera_engine_rkaiq 目录，直接通过 git 或 repo 工具可以更新。可以仅更新该目录而不影响其它 SDK 中的目录。

2. kernel 根据 camera_engine_rkaiq 的需要相应升级

在 media/isp/camera_engine_rkaiq 目录下通过查看 `git log`，可以找到它所需要的kernel rkisp驱动的版本号。例如：

```
# git log
commit 3d71d22e1e1cc080cd299b914e4e8daac2a58329
Author: ZhongYichong <zyc@rock-chips.com>
Date:   Sun Feb 18 10:17:18 2024 +0800

    release v5.0x5.0

cherry-pick:
6227d46 Revert "fastboot: remove rk_aiq_uapi2_sysctl_preInit_tb_info"
f8efdd6 Revert "fastboot: _first_awb_cfg use pointer replace struct"
```

4.3 如何确认 3A 是否正在工作

通过抓取图像，查看图像的色彩及曝光是否正常。

同时，通过查看后台是否有rkisp_3A_server进程在执行，如下：

```
# ps -ef | grep rkisp_3A_server
706 root      9176 S      /usr/bin/rkisp_3A_server --mmedia=/dev/media1
746 root      2408 S      grep rkisp_3A_server
# pidof rkisp_3A_server
706
```

可以看到进程号 706 即是 rkisp_3A_server。

4.3.1 没有看到 rkisp_3A_server 进程

- 首先确认/usr/bin/rkisp_3A_server可执行文件是否存在，如不存在，请检查camera_engine_rkaiq 版本及编译。
- 查看 /var/log/syslog 中是否有 rkisp_3A 相关的错误，如有，看具体错误是什么，是否 Sensor 模组对应的iq 文件（xxx.json）没有找到，或不匹配。
- 在 shell 中执行 `rkisp_3A_server --mmedia=/dev/media0`（如有多个/dev/media设备，选择/dev/video对应的那一个），从另一个shell中抓图。获取rkisp_3A_server对应的错误信息

4.3.2 rkisp_3A_server 是如何启动的

Linux SDK中, rkisp_3A_server由脚本/etc/init.d/S40rkisp_3A 启动并在后台执行。

如果/etc/init.d/S40rkisp_3A文件未找到, 检查camera_engine_rkisp的版本及buildroot package编译脚本。

4.3.3 如何确定 Sensor IQ 配置文件文件名及路径

Sensor IQ文件由三部分组成,

- Sensor Type, 比如sc200ai。
- Module Name, 在dts中定义, 比如rv1106g2 rk evb板上, 该名称为"CMK-OT2115-PC1"

```
rockchip,camera-module-name = "CMK-OT2115-PC1";
```

- Module Lens Name, 在dts中定义, 比如以下的"30IRC-F16":

```
rockchip,camera-module-lens-name = "30IRC-F16";
```

那么上例中的iq文件名为: sc200ai_CMK-OT2115-PC1_30IRC-F16.json, 存放在/etc/iqfiles/目录下。注意大小写有区分。

4.4 怎样手动曝光

需要手动曝光的情况下, rkisp_3A_server 进程必须先退出。然后可参考 rkisp_demo.cpp 程序或 librkisp_api.so 的源码。

4.5 如何打开 librkaiq 的 log

通过设置环境变量 persist_camera_engine_log, 其对应的位表示如下:

```
bits:      23-20    19-16 15-12  11-8   7-4    3-0
module: [xcore]   [ISP]  [AF]    [AWB]  [AEC]  [NO]

0: error
1: warning
2: info
3: verbose
4: debug
```

例如, 打开 ISP 及 AWB 的 debug log:

```
# /etc/init.d/S40rkisp_3A stop
# export persist_camera_engine_log=0x040400
# /usr/bin/rkisp_3A_server &
```

5. 应用开发相关

C 语言参考 demo

- RK 提供的 Linux SDK 中包含 rkisp_demo 工具及源码

rkisp_demo 是一个简单的工具，可以用于获取图像。类似于 v4l2-ctl 工具，rkisp_demo 也不能显示图像，它主要是提供源码供参考。

源码位于 /media/isp/camera_engine_rkaiq/rkisp_demo 目录下。

- RK 提供的 IPC SDK 中包含 sample demo 源码，simple demo 源码

sample demo 是 RK 基于 rockit 多媒体库以及 rkaiq 库基础上开发的样例程序。根据不同模块或功能分别提供有相关应用程序。

源码位于 /media/samples/example 目录下：

```
.
├── audio
│   ├── Makefile
│   ├── sample_ai_aenc.c
│   └── sample_ai.c
├── avs
│   ├── Makefile
│   ├── sample_avs.c
│   └── sample_multi_vi_avs.c
├── common
│   ├── fillimage.c
│   ├── isp2.x
│   ├── isp3.x
│   ├── lib
│   ├── loadbmp.c
│   ├── loadbmp.h
│   ├── Makefile
│   ├── sample_comm_aenc.c
│   ├── sample_comm_ai.c
│   ├── sample_comm_ao.c
│   ├── sample_comm_avs.c
│   ├── sample_comm.c
│   ├── sample_comm.h
│   ├── sample_comm_iva.c
│   ├── sample_comm_ivs.c
│   ├── sample_comm.o
│   ├── sample_comm_rgn.c
│   ├── sample_comm_tde.c
│   ├── sample_comm_venc.c
│   ├── sample_comm_vi.c
│   ├── sample_comm_vo.c
│   └── sample_comm_vpss.c
├── demo
│   ├── Makefile
│   ├── sample_demo_aiisp.c
│   ├── sample_demo_dual_aiisp.c
│   ├── sample_demo_dual_camera.c
│   └── sample_demo_dual_camera_wrap.c
```

```

|   └─ sample_demo_multi_camera_eptz.c
|   └─ sample_demo_vi_avs_venc.c
|   └─ sample_demo_vi_venc.c
|   └─ sample_rv1103_dual_memory_opt.c
└─ Makefile
└─ out
|   └─ bin
|   └─ install_to_userdata
└─ test
|   └─ Makefile
|   └─ sample_ai_aenc_adec_ao_stresstest.c
|   └─ sample_avs_stresstest.c
|   └─ sample_demo_aiisp_stresstest.c
|   └─ sample_demo_dual_aiisp_stresstest.c
|   └─ sample_demo_vi_avs_venc_stresstest.c
|   └─ sample_demo_vi_venc_stresstest.c
|   └─ sample_isp_stresstest.c
|   └─ sample_muilt_isp_stresstest.c
|   └─ sample_rgn_stresstest.c
|   └─ sample_venc_stresstest.c
|   └─ sample_vpss_stresstest.c
|   └─ source
└─ venc
|   └─ Makefile
|   └─ sample_multi_vi_avs_osd_venc.c
|   └─ sample_vi_vpss_osd_venc.c
└─ vi
|   └─ Makefile
|   └─ sample_multi_vi.c
|   └─ sample_vi.c
|   └─ sample_vi_eis.c
└─ vo
|   └─ Makefile
|   └─ sample_vi_vo.c

```

其中 **demo** 目录提供应用程序样例，**test** 目录提供压力测试应用程序样例。其他目录根据功能模块分为 **common**、**VI**、**VO**、**VENC**、**AUDIO**、**AVS**等。

6. 快速启动相关

6.1 dts 修改

1. 根据具体设备板子硬件原路图，正确配置 Sensor 相关的链接关系、上下电引脚、电源域配置。
2. 正确配置 rkisp_thunderboot 快启内存分配，根据 Sensor 实际分辨率与 VICAP 离线帧Buf 个数配置 ramdisk_r ramdisk_c 大小与偏移。

```
&rkisp_thunderboot {
    /* reg's offset MUST match with RTOS */
    /*
     * vicap, capture raw10, ceil(w*10/8/256)*256*h *4 (buf num)
     * e.g. 2304x1296: 0xf30000
     */
    // 2560 x 1440: 0x1248000
    reg = <0x00860000 0x1248000>;

    &ramdisk_r {
        reg = <0x1aa8000 (10 * 0x00100000)>;
    };

    &ramdisk_c {
        reg = <0x24a8000 (5 * 0x00100000)>;
    };
};
```

6.2 kernel Sensor 驱动

1. 按照电池 IPC 门铃的

《Rockchip_RV1106_RV1103_Quick_Start_Linux_Battery_IPC_Doorbell_CN.md》指导文档，修改当前 Sensor 驱动，添加支持快启方案的代码。

注意：快启中起流接口中不配置 Sensor 寄存器序列，仅写入起流寄存器。

```
static int __sc401ai_start_stream(struct sc401ai *sc401ai)
{
    int ret;

    if (!sc401ai->is_thunderboot) {
        ret = sc401ai_write_array(sc401ai->client, sc401ai->cur_mode-
>reg_list);
        if (ret)
            return ret;

        /* In case these controls are set before streaming */
        ret = __v4l2_ctrl_handler_setup(&sc401ai->ctrl_handler);
        if (ret)
            return ret;
    }
}
```



```

// 仅写入起流寄存器。
return sc401ai_write_reg(sc401ai->client,
    SC401AI_REG_CTRL_MODE,
    SC401AI_REG_VALUE_08BIT,
    SC401AI_MODE_STREAMING);
}

```

2. Sensor 驱动中需要正常配置曝光、增益、VBLANK 接口，否则会出现画面异常的情况。

```

static int sc401ai_set_ctrl(struct v4l2_ctrl *ctrl)
{
    .....

    switch (ctrl->id) {
    case V4L2_CID_EXPOSURE: // 曝光
        if (sc401ai->cur_mode->hdr_mode == NO_HDR) {
            val = ctrl->val << 1;
            /* 4 least significant bits of exposure are fractional part */
            ret = sc401ai_write_reg(sc401ai->client,
                SC401AI_REG_EXPOSURE_H,
                SC401AI_REG_VALUE_08BIT,
                SC401AI_FETCH_EXP_H(val));
            ret |= sc401ai_write_reg(sc401ai->client,
                SC401AI_REG_EXPOSURE_M,
                SC401AI_REG_VALUE_08BIT,
                SC401AI_FETCH_EXP_M(val));
            ret |= sc401ai_write_reg(sc401ai->client,
                SC401AI_REG_EXPOSURE_L,
                SC401AI_REG_VALUE_08BIT,
                SC401AI_FETCH_EXP_L(val));
        }
        break;
    case V4L2_CID_ANALOGUE_GAIN: // 模拟Gain
        if (sc401ai->cur_mode->hdr_mode == NO_HDR)
            ret = sc401ai_set_gain_reg(sc401ai, ctrl->val);
        break;
    case V4L2_CID_VBLANK: // VBLANK, 影响帧率
        ret = sc401ai_write_reg(sc401ai->client,
            SC401AI_REG_VTS_H,
            SC401AI_REG_VALUE_08BIT,
            (ctrl->val + sc401ai->cur_mode->height)
            >> 8);
        ret |= sc401ai_write_reg(sc401ai->client,
            SC401AI_REG_VTS_L,
            SC401AI_REG_VALUE_08BIT,
            (ctrl->val + sc401ai->cur_mode->height)
            & 0xff);
        if (!ret)
            sc401ai->cur_vts = ctrl->val + sc401ai->cur_mode->height;
        sc401ai_modify_fps_info(sc401ai);
        break;
    case V4L2_CID_TEST_PATTERN: // pattern 测试模式
        ret = sc401ai_enable_test_pattern(sc401ai, ctrl->val);
        break;
    case V4L2_CID_HFLIP: // 水平镜像
        ret = sc401ai_read_reg(sc401ai->client, SC401AI_FLIP_MIRROR_REG,
            SC401AI_REG_VALUE_08BIT, &val);
    }
}

```

```

        ret |= sc401ai_write_reg(sc401ai->client,
                                SC401AI_FLIP_MIRROR_REG,
                                SC401AI_REG_VALUE_08BIT,
                                SC401AI_FETCH_MIRROR(val, ctrl->val));

        break;
    case V4L2_CID_VFLIP:        // 垂直翻转
        ret = sc401ai_read_reg(sc401ai->client, SC401AI_FLIP_MIRROR_REG,
                                SC401AI_REG_VALUE_08BIT, &val);
        ret |= sc401ai_write_reg(sc401ai->client,
                                SC401AI_FLIP_MIRROR_REG,
                                SC401AI_REG_VALUE_08BIT,
                                SC401AI_FETCH_FLIP(val, ctrl->val));

        break;

    .....
}

pm_runtime_put(&client->dev);
return ret;
}

```

6.3 mcu rtt Sensor 驱动相关

rtt Sensor 驱动开发过程中需要注意以下几点：

1. rtt Sensor 驱动中小分辨率、大分辨率的寄存器配置不能配置 mipi enable 或控制 Sensor 起流的寄存器。若有配置会导致 rtt 阶段 AE 收敛异常。切到 kernel 按大图分辨率取流时无法取流。

```

static const uint8_t g_sc401ai_2560x1440_30fps_reg_table[] = {
    .....
    0x3,  0x36,  0xf9,  0x14,
    // 0x3,  0x01,  0x00,  0x01,      // 0x0100 寄存器起流控制寄存器不能配置
    0x3a,
}

```

2. rtt Sensor 驱动中的曝光、gain 值 计算按照 kernel Sensor 驱动中的方式。

6.4 Sensor iq 文件

- 使用正确的 iq 文件，iq 文件 json 格式转 bin 格式的方式如下：

```

./media/isp/release_camera_engine_rkaiq_rv1106_arm-rockchip830-linux-
uclibcgnueabihf/host/j2s4b    json文件    bin文件

```

修改 iq 文件后快速编译方式：

iq bin 文件可直接放在 output/out/media_out/isp_iqfiles 下，

```
cd output/out/media_out/isp_iqfiles
```

j2s4b 工具转换：

```
../host/j2s4b mis2032_CMK-OT2115-PC1_30IRC-F16.json mis2032_CMK-OT2115-PC1_30IRC-F16.bin
```

编译 meta 分区、烧录 meta 分区固件：

```
cd -  
./build.sh meta
```

或将 iq bin文件推到板端，可以参考 sdk 的文档，更新meta分区参数 sensor_iq_bin 参数重启生效。

- 快启取流前面帧图像绿色问题

考虑 awb 或曝光异常、可开启 json 文件中的 earlierAwbAct 功能；

```
"earlierAwbAct ":{  
    "enable":    1,  
    .....  
}
```

同时关闭CAC 模块

```
"cac_v11":  {  
    "SettingPara":  {  
        "enable":    0,  
    }  
}
```

6.5 问题处理

6.5.1 编译提示 “Not found main camera sensor config, ...”

编译固件时，遇到找不到主摄配置的问题，如下：

```
[build_meta.sh:error] Not found main camera sensor config, please add  
[support_sensors] in build_meta.sh
```

解决方法：

- 检查 iq 文件是否存在，build_meta.sh 脚本中已添加支持的 Sensor
- 板级配置中已修改 eport RK_CAMERA_SENSOR_IQFILES为当前Sensor的iq文件。
- 将对应 Sensor 的 iq bin 文件直接放在 output/out/media_out/isp_iqfiles下，然后编译 kernel、编译固件。重新烧录boot分区

6.5.2 kernel 崩溃

rtt 启动后，切换到kernel时崩溃，报rk_csirx_irq1_handler 异常，原因是host驱动注册好了，rtt运行阶段产生 mipi报错，这个时候会进err1的中断，但此时差不多 rtt 结束，所以 rtt 把 host 时钟关掉，导致中断中访问寄存器报错。

```
[    0.274655] [<b037486c>] (rk_csirx_irq1_handler) from [<b022e2a7>]
(__handle_irq_event_percpu+0x25/0x7e)
[    0.275490] [<b022e2a7>] (__handle_irq_event_percpu) from [<b022e30f>]
(handle_irq_event_percpu+0xf/0x30)
[    0.276334] [<b022e30f>] (handle_irq_event_percpu) from [<b022e34b>]
(handle_irq_event+0x1b/0x28)
[    0.277121] [<b022e34b>] (handle_irq_event) from [<b02300d9>]
(handle_fasteoi_irq+0x57/0x90)
[    0.277864] [<b02300d9>] (handle_fasteoi_irq) from [<b022df6f>]
(__handle_domain_irq+0x4b/0x64)
[    0.278630] [<b022df6f>] (__handle_domain_irq) from [<b02ee983>]
(gic_handle_irq+0x41/0x4e)
[    0.279375] [<b02ee983>] (gic_handle_irq) from [<b0208d13>]
(__irq_svc+0x53/0x7c)
```

解决方案:

更新rkcif驱动，添加中断异常处理。

6.5.3 如何确认不带Sensor 镜头的出流效果

如果不带镜头的sensor效果确认时，可以打开 pattern 测试模式，出彩条或灰阶图像，查看出大图效果。

pattern 模式的代码添加在起流接口中，设置起流寄存器之前。

6.5.4 kernel启动后一直打印 MIPI SIZE ERROR 错误

需要确认是否是 rtt 阶段寄存器配置的实际出图宽、高大小有问题。

仔细检查setting与手册说明。

6.5.5 Failed to stop decompress: decompress@ff520000

若遇到“Failed to stop decompress: decompress@ff520000, ret=-119”， kernel 解压缩失败情况时，可尝试修改 sysdrv/source/uboot/u-boot/arch/arm/dts/rv1106-evb2.dts 中 spi_nor 节点的最大频率，降低到 100Mhz 或更低。

```
&spi_nor {
    spi-max-frequency = <125000000> 为 spi-max-frequency = <100000000>
```

6.5.6 Failed to stop decompress: decompress@ff520000, ret=-110

若遇到“Failed to stop decompress: decompress@ff520000, ret=-110”， kernel 解压缩失败情况时，尝试修改 ramdisk 与 CMA 大小，二者不要设置太大。

6.5.7 如何设置快启应用不自启动

若 kernel 开机脚本中配置了开机自动快启应用，有时会引起开发调试不便，可使用下面 meta 命令设置快启应用不自启动。

```
make_meta --update --meta_path /dev/block/by-name/meta --cmdline NoAuto=1
```

使能 AE log 打印，同时设置不自启动快启应用。

```
make_meta --update --meta_path /dev/block/by-name/meta --cmdline  
"persist_camera_engine_log=0x1fff4 NoAuto=1"
```

注意：

目前仅 RV1106/RV1103 SDK 支持该命令。

6.5.8 如何确认 rtt 切到大图配置以后，是否提前出流

rtt 切到大图序列的时候，要确认 sensor 是否有提前出流，可以在 spl while 1 住，不进内核去然后使用示波器测量 Sensor 相关 MIPI data 波形。

6.5.9 如何快速定位快启阶段离线帧的问题

vicap 预留了一个 debug 开关，启动后，可以打印前 15 帧的 FS/FE 中断，以及 buffer 的轮转等 debug 信息。当出现拼帧/错帧等异常时，通过相关 log 来分析，可以较快定位出问题。

开启方式：

```
diff --git a/drivers/media/platform/rockchip/cif/dev.c  
b/drivers/media/platform/rockchip/cif/dev.c  
index c3b59414eede3..bb616b1897b58 100644  
--- a/drivers/media/platform/rockchip/cif/dev.c  
+++ b/drivers/media/platform/rockchip/cif/dev.c  
@@ -1957,7 +1957,7 @@ int rkCIF_plat_init(struct rkCIF_device *cif_dev, struct  
device_node *node, int  
    cif_dev->sensor_linetime = 0;  
    cif_dev->early_line = 0;  
    cif_dev->is_thunderboot = false;  
-    cif_dev->rdbk_debug = 0;  
+    cif_dev->rdbk_debug = 3;  
  
    cif_dev->resume_mode = 0;  
    memset(&cif_dev->channels[0].capture_info, 0, sizeof(cif_dev->  
channels[0].capture_info));
```

6.5.10 如何抓取 rtt 阶段的小图，查看效果

- dts 中将所有 rkisp、rkCIF、mipi 注释掉: disabled
csi2_dphy_hw:disable
csi2_dphy0:disable

```
mipi0_csi2:disabled
rkCIF:disabled
rkCIF_mipi_lvds:disabled
rkCIF_mipi_lvds_sdtf:disabled
rkisp:disabled
rkisp_vir0:disabled
```

- `io -rf /tmp/1.yuv -l 393216 0x866000` 保存小图数据，只能保存最后5帧数据，可强制修改 `fastae_max_run_frame=5`

```
[STREAM]: L 0, 1, 0, 0x866000, 783360, 0x45d, 0x8000, tick:59
[STREAM]: L 1, 2, 0, 0x926000, 783360, 0x45d, 0x8000, tick:76
[STREAM]: L 2, 3, 0, 0x9e6000, 783360, 0x45d, 0x8000, tick:87
[STREAM]: L 3, 4, 0, 0xaa6000, 783360, 0x45d, 0x1828, tick:98
[STREAM]: L 4, 5, 0, 0xb66000, 783360, 0x45d, 0x48d, tick:109
```

0x866000、0x926000 ...等是rtt阶段保存的最后几帧数据。

6.5.11 首帧图像颜色不正常

首帧图像出现偏黑、偏绿时，一般是iq文件不正确，导致awb白平衡不正常导致异常。

如下 rtt log 中 luma 亮度为很小的值，导致首帧偏黑：

```
[FASTTAE]: >>> als type: 1, als value: 0xa0000, night mode: 0
start_stream 278 tick 23
[isp_0]:rk_isr_hw_set_vicap_clk enable is 1
[FASTTAE]: set_firstae 1733 tick 23
[STREAM]: L 0, 1, 0, 0x866000, 178r400, 0x2, 0x40, tick:43
[AELIB]: 1,luma=1.0000,setpoint=60.0000,curexp=0.0000,newexp=0.0000,T?
8.0000,G=1.0000,ispG=1.0000,regT=2,regG=64

[STREAM]: L 1, 2, 0, 0x9b80000 1382400, 0x2, 0x40, tick:62
[AELIB]:2,luma=1.0000,setpoint=60.0000,curexp=0.0000,newexp=0.0000,T=0.0000,G=1.0
000,ispG=1.0000 regT=2,regG=64

[STREAM]: L 2, 3, 0, 0xb0a000, 138r400, 0x2, 0x40, tick:80
[AELIB]: 3,luma=1.0000,setpoint=60.0000,curexp=0.0000 =====fastae is
match=====
```

解决方案：

- 检查 iq文件中 `earlierAwbAct` 参数是否使能，参数是否配置正确。如下

```
"earlierAwbAct": {
    "enable": 1,
    "mode": "CALIB_AWB_EARLACT_XYREG_FIXED",
    "xyRegion": [{
        "normal": [-161, 286, 81, -95],
        "big": [-161, 286, 112, -125]
    }, {
        "normal": [-803, -161, 135, -199],
        "big": [-803, -161, 185, -224]
    }, {
        "normal": [-1467, -796, 121, -78],
```

```
    "big":  [-1559, -796, 136, -93]
  }, {
    "normal":  [-2062, -1535, 65, -130],
    "big":  [-2017, -1535, 130, -130]
  }]
}
```

说明:

针对快启应用, 需要在json上打开 `earlierAwbAct` 功能, 尽早进行awb的调整。 awb 正常是 2 帧才会算出一个结果, 不开快速 awb, 那就是用的初始值 awb, wb gain 都是1, 就是绿的。

针对快启, 我们是要求 tuning 的时候, `earlierawb` 开启, 选择 AUTO 模式, 然后填上对应的参数。

7. AOV 相关

7.1 如何支持 Sensor 硬件 standby 模式

standby 模式是指 Sensor 为了节省功能处在睡眠模式或者复位模式。

- 睡眠模式

Sensor 停止图像数据流，工作在低功耗状态，保持当前寄存器值。

以 SC200AI 为例，有两种方式进入睡眠模式：

1. 将 PWDN 拉低，此时不支持 I2C 的读写。
2. 将出流控制寄存器写入0，此时支持 I2C 读写。

- 复位模式

Sensor 停止图像数据流，工作在低功耗状态，重置所有寄存器。

以 SC200AI 为例，有两种方式进入复位模式：

1. 将 XSHUTDN 拉低，此时不支持 I2C 读写。
2. 将软使能复位寄存器写入1，此复位模式持续 150ns。

硬件 standby 模式指通过拉低 PWDN 管脚，使 Sensor 进入睡眠模式，从而使其工作在低功耗状态，此时 I2C 不可读写。

以 SC200AI 为例，dts 中 Sensor 节点中添加支持硬件 standby 模式属性 `rockchip,camera-module-stb` = <1>，其属性值用来表示是否支持硬件 standby 方式，1 表示指支持，0 表示不支持。

```
sc200ai: sc200ai@30 {
    compatible = "smartsens,sc200ai";
    status = "okay";
    reg = <0x30>;
    ...
    rockchip,camera-module-stb = <1>;
    port {
        sc200ai_out: endpoint {
            remote-endpoint = <&csi_dphy_input1>;
            data-lanes = <1 2>;
        };
    };
};
```

Sensor 驱动 probe 接口中需添加解析该属性值，获取到硬件 standby 模式是否支持，并保存在 `sc200ai->standby_hw` 成员变量中。

若支持硬件 standby 模式，`quitck_stream` 中会根据起流状态上、下拉相关 PWDN GPIO。休眠唤醒中会下发 v4l2 ioctl 命令来重新设置曝光、增益寄存器等。

```
case RKMODULE_SET_QUICK_STREAM:
    stream = *((u32 *)arg);
    if (sc200ai->standby_hw) { // hardware standby
        if (stream) {
            if (!IS_ERR(sc200ai->pwdn_gpio))
                gpiod_set_value_cansleep(sc200ai->pwdn_gpio, 1);
        }
    }
}
```



```

        ret = sc200ai_write_reg(sc200ai->client, SC200AI_REG_MIPI_CTRL,
                                SC200AI_REG_VALUE_08BIT, SC200AI_MIPI_CTRL_ON);

        ret |= sc200ai_write_reg(sc200ai->client, SC200AI_REG_CTRL_MODE,
                                SC200AI_REG_VALUE_08BIT, SC200AI_MODE_STREAMING);

        dev_info(&sc200ai->client->dev, "quickstream, streaming on: exit
standby mode\n");
        sc200ai->is_standby = false;
    } else {
        ret = sc200ai_write_reg(sc200ai->client, SC200AI_REG_CTRL_MODE,
                                SC200AI_REG_VALUE_08BIT, SC200AI_MODE_SW_STANDBY);

        ret |= sc200ai_write_reg(sc200ai->client, SC200AI_REG_MIPI_CTRL,
                                SC200AI_REG_VALUE_08BIT, SC200AI_MIPI_CTRL_OFF);

        if (!IS_ERR(sc200ai->pwdn_gpio))
            gpiod_set_value_cansleep(sc200ai->pwdn_gpio, 0);

        dev_info(&sc200ai->client->dev, "quickstream, streaming off:
enter standby mode\n");
        sc200ai->is_standby = true;
    }
} else {    // software standby
    ...
}

```

其他可参考 SC200AI 驱动。

7.2 如何对补光灯进行控制

在 AOV 模式下，支持对补光灯进行控制。支持每个 Sensor 控制一个补光灯，支持 PWM 或 GPIO 类型的补光灯。

控制的设置如下：

1. 板级配置 DTS 中添加 light_ctl 节点，节点属性中根据硬件实际连接添加补光灯的 PWM、GPIO 对应的控制设备或管脚。如下示例：这里的补光灯有 pwm7 设备控制，index 为 0，gpio 未使用。

```

/ {
    model = "Rockchip RV1106G EVB2 V10 Board";
    compatible = "rockchip,rv1106g-evb2-v10", "rockchip,rv1106";

    chosen {
        bootargs = "loglevel=0 rootfstype=erofs rootflags=dax console=ttyFIQ0
root=/dev/rd0 snd_soc_core.prealloc_buffer_size_kbytes=16 coherent_pool=0
driver_async_probe=dwmmc_rockchip";
    };
    ...
    light_ctl: light-ctl {
        compatible = "rockchip,light-ctl";
        pwms=<&pwm7 0 25000 0>;
        light-gpios = <&gpio3 RK_PD2 GPIO_ACTIVE_HIGH>;
        rockchip,module-index = <0>;
        status = "okay";
    };
}

```

```
};

};
```

2. kernel 配置中打开 `CONFIG_LIGHT_CTL` 配置。

3. Sensor 驱动中添加相关控制接口。

```
#include <linux/clock.h>
@@ -40,8 +41,9 @@
#include "../platform/rockchip/isp/rkisp_tb_helper.h"
#include "cam-tb-setup.h"
#include "cam-sleep-wakeup.h"
+include "light_ctl.h"

-define DRIVER_VERSION          KERNEL_VERSION(0, 0x01, 0x09)
+define DRIVER_VERSION          KERNEL_VERSION(0, 0x01, 0x10)

#ifndef V4L2_CID_DIGITAL_GAIN
#define V4L2_CID_DIGITAL_GAIN      V4L2_CID_GAIN
@@ -192,6 +194,7 @@ struct sc200ai {
    bool          is_standby;
    struct preisp_hdrae_exp_s init_hdrae_exp;
    struct cam_sw_info  *cam_sw_inf;
+    struct rk_light_param  light_ctl_param;
};

#define to_sc200ai(sd) container_of(sd, struct sc200ai, subdev)
@@ -1191,6 +1194,8 @@ static long sc200ai_ioctl(struct v4l2_subdev *sd,
unsigned int cmd, void *arg)
    u32 i, h, w;
    long ret = 0;
    u32 stream = 0;
+    int rt = 0;
+    struct rk_light_param *light_param;

    switch (cmd) {
        case RKMODULE_GET_MODULE_INFO:
@@ -1238,6 +1243,22 @@ static long sc200ai_ioctl(struct v4l2_subdev *sd,
unsigned int cmd, void *arg)

        stream = *((u32 *)arg);

+        dev_err(&sc200ai->client->dev, "%s: quick_stream = %d\n",
+            __func__, stream);
+        // light control
+        if (stream) {
+            sc200ai->light_ctl_param.light_enable = true;
+            rt = light_ctl_write(sc200ai->module_index,
+                &sc200ai->light_ctl_param);
+        } else {
+            sc200ai->light_ctl_param.light_enable = false;
+            rt = light_ctl_write(sc200ai->module_index,
+                &sc200ai->light_ctl_param);
+        }
+        dev_err(&sc200ai->client->dev, "%s: light_ctl_write ret:%d\n",
+            __func__, rt);
```

```

+
+         if (sc200ai->standby_hw) { // hardware standby
+             if (stream) {
+                 if (!IS_ERR(sc200ai->pwdn_gpio))
@@ -1284,6 +1305,18 @@ static long sc200ai_ioctl(struct v4l2_subdev *sd,
unsigned int cmd, void *arg)
+                 ch_info = (struct rkmodule_channel_info *)arg;
+                 ret = sc200ai_get_channel_info(sc200ai, ch_info);
+                 break;
+
+
+ case RKCIS_CMD_FLASH_LIGHT_CTRL:
+     light_param = (struct rk_light_param *)arg;
+     dev_err(&sc200ai->client->dev,
+         "%s: RKCIS_CMD_FLASH_LIGHT_CTRL type: %s enable: %s\n",
+         __func__,
+         light_param->light_type == LIGHT_PWM ? "pwm" : "gpio",
+         light_param->light_enable ? "enable" : "disable");
+
+     memcpy(&sc200ai->light_ctl_param, light_param, sizeof(*light_param));
+
+     break;
+
 default:
     ret = -ENOIOCTLCMD;
     break;
@@ -1304,6 +1337,7 @@ static long sc200ai_compat_ioctl32(struct v4l2_subdev
*sd,
+     struct rkmodule_channel_info *ch_info;
+     long ret;
+     u32 stream = 0;
+     struct rk_light_param *light_param = NULL;
+
+     switch (cmd) {
+         case RKMODULE_GET_MODULE_INFO:
@@ -1400,6 +1434,19 @@ static long sc200ai_compat_ioctl32(struct v4l2_subdev
*sd,
+         }
+         kfree(ch_info);
+         break;
+
+ case RKCIS_CMD_FLASH_LIGHT_CTRL:
+     light_param = kzalloc(sizeof(*light_param), GFP_KERNEL);
+     if (!light_param) {
+         ret = -ENOMEM;
+         return ret;
+     }
+     ret = copy_from_user(light_param, up, sizeof(*light_param));
+     if (!ret)
+         ret = sc200ai_ioctl(sd, cmd, light_param);
+     else
+         ret = -EFAULT;
+     kfree(light_param);
+     break;
+
 default:
     ret = -ENOIOCTLCMD;
     break;
@@ -1451,6 +1498,10 @@ static int __sc200ai_stop_stream(struct sc200ai
*sc200ai)
+     sc200ai->is_first_streamoff = true;
+     pm_runtime_put(&sc200ai->client->dev);

```

```

    }
+   sc200ai->light_ctl_param.duty_cycle = 0;
+   sc200ai->light_ctl_param.light_enable = false;
+   light_ctl_write(sc200ai->module_index,
+                   &sc200ai->light_ctl_param);
    return sc200ai_write_reg(sc200ai->client, SC200AI_REG_CTRL_MODE,
                             SC200AI_REG_VALUE_08BIT, SC200AI_MODE_SW_STANDBY);
}
@@ -1973,6 +2024,7 @@ static int sc200ai_initialize_controls(struct sc200ai
*sc200ai)
    sc200ai->subdev.ctrl_handler = handler;
    sc200ai->has_init_exp = false;
    sc200ai->is_standby = false;
+   sc200ai->light_ctl_param.duty_cycle = 0;

    return 0;

```

4. 应用代码示例:

```

--- a/sample_aov_vi.c
+++ b/sample_aov_vi.c
@@ -572,6 +572,19 @@ int main(int argc, char *argv[]) {
    if (!ctx->vi.bIfQuickStart) {
        RK_MPI_VI_StartPipe(ctx->vi.u32PipeId);
    }

+   // Enable VI light
+   VI_LIGHT_CTL_PARAM_S tLightCtlParam;
+   printf("%s - VI_LIGHT_CTL_PARAM_S size:%d\n", __func__,
+         sizeof(VI_LIGHT_CTL_PARAM_S));
+   memset(&tLightCtlParam, 0, sizeof(tLightCtlParam));
+   tLightCtlParam.light_enable = RK_TRUE;
+   tLightCtlParam.light_type = LIGHT_TYPE_PWM;
+   tLightCtlParam.duty_cycle = 25000;
+   tLightCtlParam.period = 25000;
+   tLightCtlParam.polarity = 0;
+
+   RK_MPI_VI_DevEnableLight(ctx->vi.s32DevId, ctx->vi.s32DevId,
+ &tLightCtlParam);
+
    if (s32ViFrameMode == 0)
        pthread_create(&vi_thread_id, 0, vi_get_stream, (void *)
(&ctx->vi));

```

7.3 AOV 开发参考文档

AOV 相关的详细开发文档，可参考 《Rockchip_RV1106_Developer_Guide_Linux_AOV_CN.md》。