

密级状态：绝密() 秘密() 内部() 公开(✓)

Camera_Hal3_User_Manual

(ISP 部)

文件状态： [✓] 正在修改 [] 正式发布	当前版本：	V1.0
	作 者：	付祥
	完成日期：	2018-11-12
	审 核：	
	完成日期：	

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有, 翻版必究)

版本历史

版本号	作者	修改日期	修改说明	审核	备注
V1.0	付祥	2018-11-12	发布初版		
V1.0.1	付祥	2018-11-13			
V2.0	付祥	2018-11-16	增加初版 hal3 框架 说明		

目录

目录.....	3
ROCKCHIP CAMERAHAL3 框架与新增 CAMERA 配置及调试说明	5
1. CAMERA HAL3 框架	5
1.1 CAMERA HAL3 基本框架:	5
1.2 代码目录简要说明.....	6
1.3 CAMERA HAL3 基本组件:	6
1.4 CAMERA HAL3 与 FRAME WORK 交互时序:	7
1.5 CAMERA HAL3 实现详细时序:	8
1.6 GRAPH 与 MEDIACTL PIPELINE:	8
1.7 CAMERA BUFFER 与 METADATA 管理:	8
2. SENSOR 适配简要步骤说明:	9
2.1 生成 GRAPH_SETTINGS_<SENSOR NAME>.XML.....	9
2.1.1 脚本使用:	9
2.1.2 脚本参数说明如下:	9
2.2 获取 TUNNING XML	10
2.3 配置 CAMERA3_PROFILES.XML	10
2.3.1 camera3_profiles.xml 说明:	10
2.3.2 客户所需修改:	11
2.3.3 xml 运行生效:	13
3. 编译运行调试:	13
3.1 编译:	13
3.2 生成库:	14
3.3 运行:	14
4. DUMP 说明.....	15
4.1 属性说明:	15
4.2 未生成 DUMP 文件问题:	16
5. 版本说明:	16

5.1 HAL3 版本获取:16

Rockchip CameraHal3 框架与新增 camera 配置及调试说明

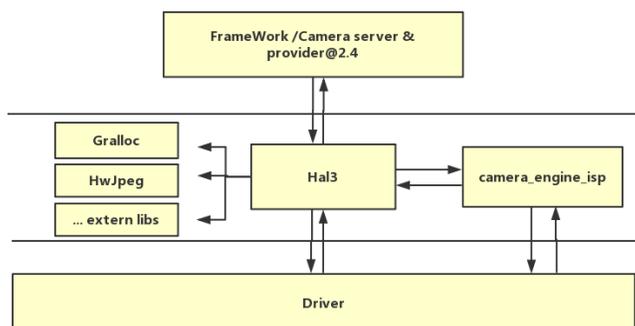
Hal3 基于新框架的 rkisp1 及 cif 驱动，新框架驱动介绍可参考文档《Rockchip Linux Camera 开发指南.pdf》。
(Hal3 代码目录位于 <工程根目录>/hardware/rockchip/camera，以下使用<hal3_camera>来代替)

文档适用平台

芯片平台	驱动	操作系统	支持情况
RV3326	Linux (Kernel-4.4):rkisp1 driver	Android9.0	Y

1. Camera Hal3 框架

1.1 Camera Hal3 基本框架:



Camera hal3 在 android 框架中所处的位置如上图，对上，主要实现 Framework 一整套 API 接口，响应其控制命令，返回数据与控制参数结果。对下，主要是通 V4l2 框架实现与 kernel 的交互。3a 控制则是通 control loop 接口与 camera_engine_isp 交互。另外，其中一些组件或功能的实现也会调用到其他一些第三方库，如 cameraBuffer 相关，会调用到 Galloc 相关库，jpeg 编码则会调用到 Hwjpeg 相关库。

驱动框架文档参考：《RKISP_Driver_User_Manual_v1.0》

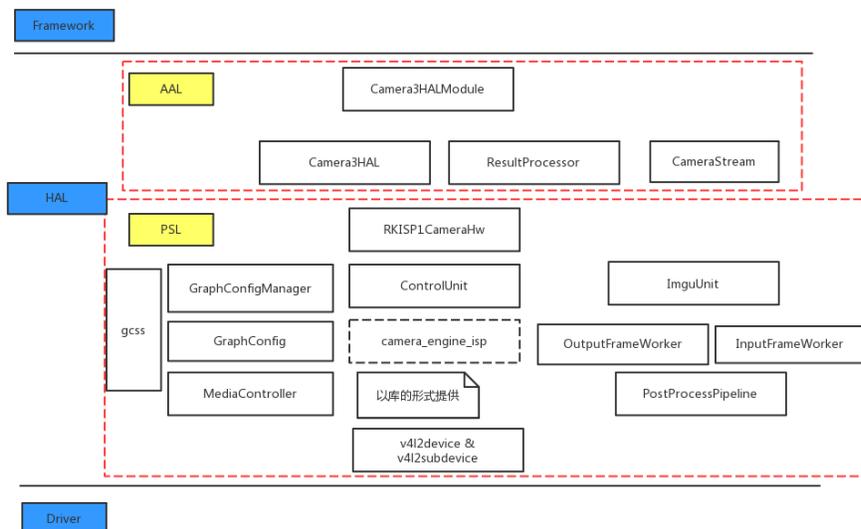
Camera_engine_isp 参考：《Camera_Engine_Rkisp_User_Manual》

1.2 代码目录简要说明

hal3_camera :

—— AAL	Android Abstraction Layer, 负责与 framework 交互
—— common	公用文件, 如线程, 消息处理, Log 打印等实现
—— gcss	xml 解析相关
—— imageProcess	图像处理相关, 如 scale
—— jpeg	jpeg 编码相关
—— mediacontroller	media pipeline 相关
—— platformdata	hal3 能力支持的属性, 主要是管理从 xml 获取到的属性
—— utils	目前只有一个 Error.h, 定义了一些返回值
—— v4l2dev	封装了一些与 v4l2 驱动交互的具体 io
—— etc	配置文件目录
—— include	Control loop 的头文件, buffer_manager 相关头文件
—— lib	3a engine 相关库
—— psl	Physical Layer, 物理实现层, 所有的实现逻辑基本都在这里
—— rkisp1	目前只有 Rkisp1 一套实现方案
—— tasks	基本只用到了里面的几个 Notify 的接口类和 JpegEncodeTask
—— workers	数据的获取处理都在这里
—— tools	包含了一个自动生成 graph setting xml 的 Python 脚本

1.3 Camera Hal3 基本组件:

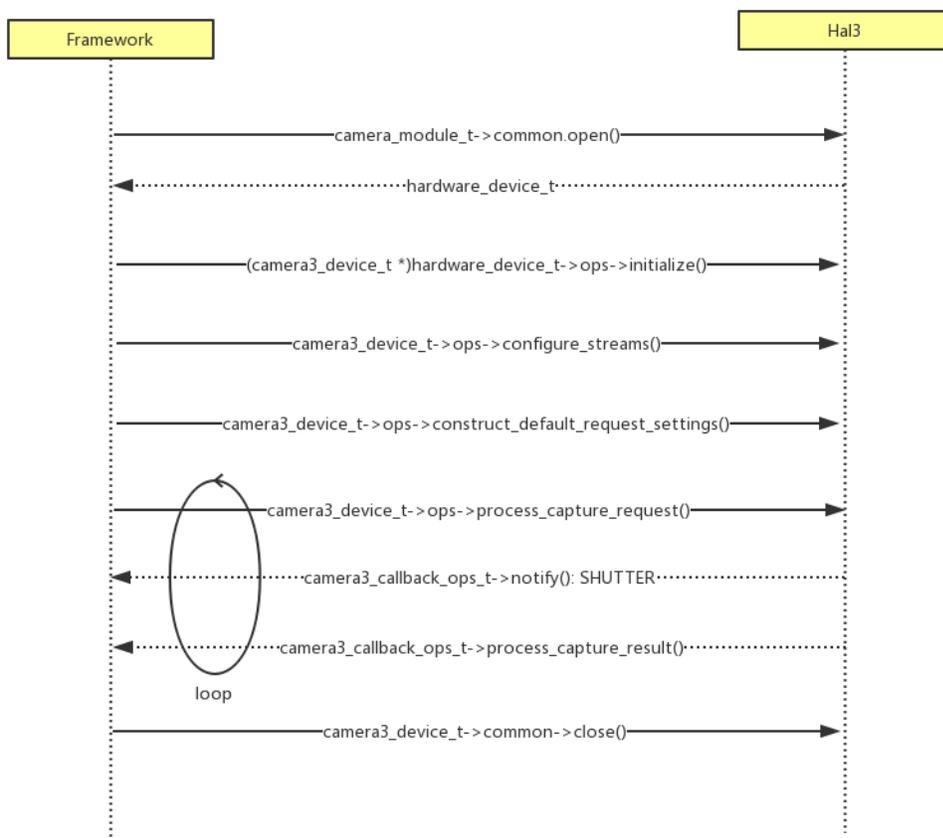


Camera hal3 中的模块主要包括 AAL 与 PSL。

AAL: 主要负责与 framework 交互, camera_module 与 API 接口实例 camera3_device_ops 在此模块定义。该模块对此 API 加以封装, 并将请求发往 PSL, 并等待接收 PSL 返回相应数据流与控制参数。

PSL: 则是物理层的具体实现, 其中 gcsc、GraphConfig、MediaController 主要负责配置文件 xml 的解析, 底层 pipeline 的配置, ControlUnit 主要负责与 camera_engine_isp 的交互, 以实现 3a 的控制, 并中转一些请求以及 Metadata 的处理收集上报。ImgUnit、OutputFrameWork、postProcessPipeline 则主要负责获取数据帧并做相应处理以及上报。V4l2device、V4l2Subdevice 则是负责与 v4l2 驱动交互, 实现具体的 io 操作。

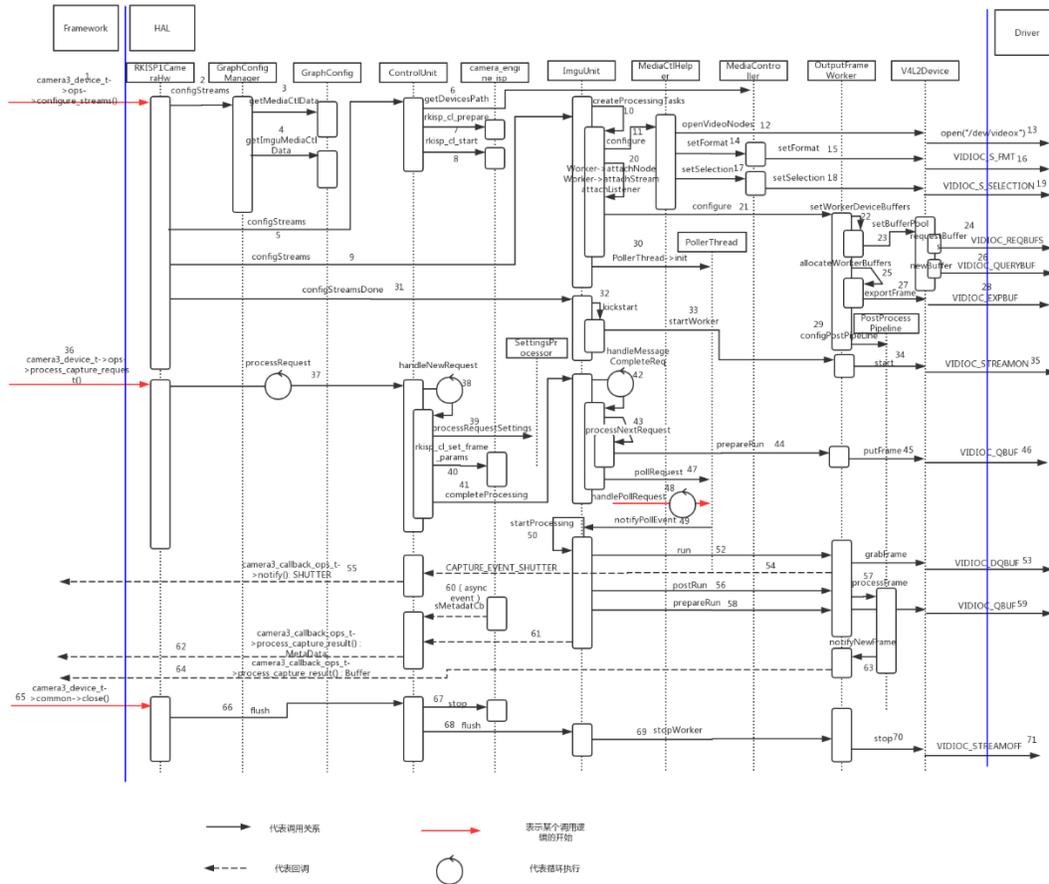
1.4 Camera hal3 与 Frame work 交互时序:



关于 framework 与 hal 交互的 API 详细说明文档可以参考:

<android_root>/hardware/libhardware/include/hardware/camera3.h

1.5 Camera Hal3 实现详细时序:



此图主要描绘了 `configure_streams` 流程, `process_capture_request` 流程在 Hal3 中的具体实现逻辑, 时序图以这两个 API 接口为起始点, 直到 hal3 下发 v4l2 相关 ioctl 并返回相关数据结果为止。该流程图基本涵盖了 Hal3 中的主要模块

上图中  符号代表循环执行, 也表示此处有线程正在等待事件的到来。Hal 层的运行也正是由这些事件驱动(即上面的红色键头)。

1.6 Graph 与 mediactl pipeline:

TODO

1.7 Camera buffer 与 MetaData 管理:

TODO

2. Sensor 适配简要步骤说明:

在 sensor 驱动已经调通的基础上, HAL 中添加新 sensor 支持需要配置如下文件: (Hal3 代码目录位于 <工程根目录>/hardware/rockchip/camera, 以下使用<hal3_camera>来代替)

- 1) 生成与 sensor 相匹配的 graph setting 文件: graph_settings_<sensor name>.xml
- 2) 获取 tuning 文件, SOC sensor 可略过此步骤
- 3) 配置 camera3_profiles.xml
- 4) 将配置文件 push 到板子, 并重新启动 camera 进程

以下章节是各个步骤详细说明。

2.1 生成 graph_settings_<sensor name>.xml

graph_settings_<sensor name>.xml 主要 hal3 用来是配置 pipeline 各级输出格式的。可以使用 <hal3_camera>/tools/gen_sensor_graph_setting.py 脚本自动生成。

2.1.1 脚本使用:

RAW sensor 以 ov5695 为例: (SOC sensor 配置方法一样, sensor_fmt 不同而已)

```
./gen_sensor_graph_setting.py --name=ov5695 --sensor_fmt=BG10 --width=2592 --height=1944 --binner_width=1296 --binner_height=972
```

执行该脚本后, 会生成 graph_setting_ov5695.xml 文件, 需要将之拷贝至<hal3_camera>/etc/camera 目录

```
cp graph_setting_ov5695.xml <hal3_camera>/etc/camera/
```

SOC sensor 以 gc0132 为例:

```
./gen_sensor_graph_setting.py --name=gc0132 --sensor_fmt=YUYV --width=640 --height=480
```

如果客户暂不清楚各项参数如何配置, 可先只配置 name、width、height, 即如下:

```
./gen_sensor_graph_setting.py --name=ov5695 width=2592 --height=1944
```

2.1.2 脚本参数说明如下:

--name: sensor 名称, 一般与 sensor 驱动文件名相同, 具体可用

```
adb shell cat /sys/class/video4linux/v4l-subdev*/name 列出所有的 subdev name, 然后找到对应的 sensor subdev 的 name。
```

--sensor_fmt: sensor_fmt 可查询 sensor 驱动, 或者使用 `adb shell media-ctl -p` (需先执行 `adb root`, 后续该命令不做说明) 列出 media 设备的 pipeline 信息, 然后找到对应 sensor 的 entity 部分, entity 信息里有包含 sensor_fmt 信息。(该项设置错误将会导致预览颜色显示不对)

--bayer_order: 与 sensor_fmt 中的的信息相对应。目前暂时未用到，可不配。

--width: sensor 输出 full width, 参见下图

--height: sensor 输出 full height , 参见下图

--binner_width: sensor 输出 binner width, 此项需要驱动支持输出 binner 尺寸, 如不支持, 可不配

--binner_height: sensor 输出 binner height, 此项需要驱动支持输出 binner 尺寸, 如不支持, 可不配

```
- entity 7: 1ov5695 2-0036 (1 pad, 1 link) 1: name
  type V4L2 subdev subtype Sensor flags 0 2: sensor_fmt
  device node name /dev/v4l-subdev2 3: width
  pad0: Source 2 3 4 4: height
        [fmt:SBGGR10/2592x1944@10000/300000 field:none]
        -> "rockchip-sy-mipi-dphy":0 [ENABLED]
```

上图是从 \$ adb shell media-ctl -p 列出的信息中截取的信息

其中 fmt 为 fmt:SBGGR10 , 则设置项对应为 --sensor_fmt=BG10, 对应关系可以参见 videodev2.h (下图截取该文件中一小段)

```
3 /* WARNING: DO NOT EDIT, AUTO-GENERATED CODE - SEE TOP FOR INSTRUCTIONS */
4 #define V4L2_PIX_FMT_SGRB8 v4l2_fourcc('G', 'R', 'B', 'G')
5 #define V4L2_PIX_FMT_SRGB8 v4l2_fourcc('R', 'G', 'B', 'B')
6 #define V4L2_PIX_FMT_SBGGR10 v4l2_fourcc('B', 'G', '1', '0')
7 #define V4L2_PIX_FMT_SGBRG10 v4l2_fourcc('G', 'B', '1', '0')
8 /* WARNING: DO NOT EDIT, AUTO-GENERATED CODE - SEE TOP FOR INSTRUCTIONS */
9 #define V4L2_PIX_FMT_SRGBG10 v4l2_fourcc('B', 'A', '1', '0')
0 #define V4L2_PIX_FMT_SRGBB10 v4l2_fourcc('R', 'G', '1', '0')
1 #define V4L2_PIX_FMT_SBGGR10P v4l2_fourcc('P', 'B', 'A', 'A')
2 #define V4L2_PIX_FMT_SGBRG10P v4l2_fourcc('P', 'G', 'A', 'A')
```

2.2 获取 tuning xml

tunning 文件是效果参数文件, 只有 Raw sensor 才需该此文件。该文件如何获取可以联系 FAE。一般以如下方式命名 senso_name+module_name+lens_name, 然后将该文件放入 <hal3_camera>/etc/camera/rkisp/ 目录下, 并参照下一章节 iqTuningFile 配置, 配置该文件路径即可。另外, 调试 Raw sensor 数据通路时, 也可先 bypass isp。只需要将 sensor 类型设置为 SOC 即可 (参见 3.2 节中 sensor 类型的设置。), 此时, tuning 文件可暂不配置。

2.3 配置 camera3_profiles.xml

2.3.1 camera3_profiles.xml 说明:

在 <hal3_camera>/etc/camera 目录下有多个 camera3_profiles_<platform>.xml, 最终会有一个文件 push 到 /vendor/etc/camera/camera3_profiles.xml. 选择一个适用的 camera3_profiles_<platform>.xml 文件, 参照前面 sensor 的配置添加新 sensor。

camera3_profiles.xml 中包含了多个 Profiles 节点, Profiles 节点包含一个 camera 完整属性列表。开发

板上接了几个 sensor，即需要配置几个 Profiles 节点。

Profiles 节点下又包含了如下五个子节点。

```
<Profiles cameraId="0" name="ov5695">
  <Supported_hardware>
  </Supported_hardware>

  <Android_metadata>
  </Android_metadata>

  <!-- **** PSL specific section start ****-->
  <Hal_tuning_RKISP1>
  </Hal_tuning_RKISP1>

  <Sensor_info_RKISP1>
  </Sensor_info_RKISP1>

  <MediaCtl_elements_RKISP1>
  </MediaCtl_elements_RKISP1>
  <!-- **** PSL specific section end ****-->
</Profiles>
```

`<Android_metadata>` 节点包含的信息主要是 camera 的能力支持，该字段的信息上层将通过 camera_module 的 API: `get_camera_info()` 获取到。Camera 运行时也可以通过如下命令获取到相关的信息。

```
$ adb shell dumpsys media.camera
```

该节点中详细字段的定义可以参见 android 开发者网站：（CTS 中的一些问题需要详细查看该网站中字段的定义）

<https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics>

其他的几个子节点 主要是平台实现所需要的一些信息， 这些对上层是透明的。

2.3.2 客户所需修改:

如下属性需要客户修改以适应不同 sensor:

```
<Android_metadata>
```

- cameraId="0" // cameraId 后置为 0，前置为 1， 只有一个 camera 时， 为 0

- name="ov5695" // sensor 的名字, 参见第一步所述方法获取
- control.aeAvailableTargetFpsRanges // 该设置项有多个限制需要注意
 - 1) 录像必需有一组恒定帧率, 假如帧率为 x, 那就要包含 (x, x)
 - 2) 录像帧率必需至少要一组大于 24 帧
 - 3) 第一组的 Min <= 15. 所以第一组一般为 (15, x)
 如不确定, 先设置为 control.aeAvailableTargetFpsRanges value="15, 30, 30, 30"

```
<control.aeAvailableTargetFpsRanges value="15, 30, 30, 30"/>
```

- scaler.availableStreamConfigurations // hal 层支持的分辨率列表, 需要按照 sensor 的最大输出尺寸依次降序排列。Sensor 尺寸下面列出的都是一些通用尺寸, 如 1080P, 720P, VGA 等。如不确定, 可从前面例子中 copy 一组, 然后修改最大分辨率

```
<scaler.availableStreamConfigurations value="BLOB, 1600x1200, OUTPUT,
BLOB, 1280x720, OUTPUT,
BLOB, 800x600, OUTPUT,
BLOB, 640x480, OUTPUT,
BLOB, 320x240, OUTPUT,
BLOB, 176x144, OUTPUT,
YCbCr_420_888, 1600x1200, OUTPUT,
YCbCr_420_888, 1280x720, OUTPUT,
YCbCr_420_888, 800x600, OUTPUT,
YCbCr_420_888, 640x480, OUTPUT,
YCbCr_420_888, 320x240, OUTPUT,
YCbCr_420_888, 176x144, OUTPUT,
IMPLEMENTATION_DEFINED, 1600x1200, OUTPUT,
IMPLEMENTATION_DEFINED, 1280x720, OUTPUT,
IMPLEMENTATION_DEFINED, 800x600, OUTPUT,
IMPLEMENTATION_DEFINED, 640x480, OUTPUT,
IMPLEMENTATION_DEFINED, 320x240, OUTPUT,
IMPLEMENTATION_DEFINED, 176x144, OUTPUT,
IMPLEMENTATION_DEFINED, 1600x1200, INPUT,
IMPLEMENTATION_DEFINED, 1280x720, INPUT,
IMPLEMENTATION_DEFINED, 800x600, INPUT,
IMPLEMENTATION_DEFINED, 640x480, INPUT,
IMPLEMENTATION_DEFINED, 320x240, INPUT,
IMPLEMENTATION_DEFINED, 176x144, INPUT,
YCbCr_420_888, 1600x1200, INPUT,
YCbCr_420_888, 1280x720, INPUT,
YCbCr_420_888, 800x600, INPUT,
YCbCr_420_888, 640x480, INPUT,
YCbCr_420_888, 320x240, INPUT,
YCbCr_420_888, 176x144, INPUT" />
```

如不确定, 可先将最大分辨率修改为 sensor 输出尺寸, 其他不做修改

- scaler.availableMinFrameDurations // 各分辨率下最小时延, 需要按照 sensor 的最大输出尺寸依次降序排列。如不确定, 可先将各个分辨率的最小时延设置为 33333333

```
<scaler.availableMinFrameDurations value="BLOB, 1600x1200, 50125313,
BLOB, 1280x720, 50125313,
BLOB, 800x600, 33333333,
BLOB, 640x480, 33333333,
BLOB, 320x240, 33333333,
BLOB, 176x144, 33333333,
YCbCr_420_888, 1600x1200, 50125313,
YCbCr_420_888, 1280x720, 50125313,
YCbCr_420_888, 800x600, 33333333,
YCbCr_420_888, 640x480, 33333333,
YCbCr_420_888, 320x240, 33333333,
YCbCr_420_888, 176x144, 33333333,
IMPLEMENTATION_DEFINED, 1600x1200, 50125313,
IMPLEMENTATION_DEFINED, 1280x720, 50125313,
IMPLEMENTATION_DEFINED, 800x600, 33333333,
IMPLEMENTATION_DEFINED, 640x480, 33333333,
IMPLEMENTATION_DEFINED, 320x240, 33333333,
IMPLEMENTATION_DEFINED, 176x144, 33333333" />
```

如果不确定, 先设置成 33333333

- scaler.availableStallDurations // 各分辨率下拍照最小时延, 需要按照 sensor 的最大输出尺寸依次降序排列。如不确定, 请参照 scaler.availableMinFrameDurations
- sensor.info // sensor 相关属性的配置

```

<!-- Sensor Info -->
<sensor.info.activeArraySize value="0,0,1600,1200"/>
<sensor.info.sensitivityRange value="32,2400"/>
<sensor.info.colorFilterArrangement value="GRBG"/> <!-- HAL may override this value from CMC for RAW sensors -->
<sensor.info.exposureTimeRange value="100000,53000000"/>
<sensor.info.maxFrameDuration value="66666666"/>
<sensor.info.physicalSize value="5.5,4.5"/> <!-- 224x1.12um 3136x1.12um -->
<sensor.info.pixelArraySize value="1600x1200"/>
<sensor.info.whiteLevel value="0"/> <!-- HAL may override this value from CMC for RAW sensors -->
<sensor.info.timestampSource value="UNKNOWN"/>

```

<Hal_tuning_RKISP1>

- graphSettingsFile // 配置文件的名称, 本目录下 graph_settings_xx.xml
- iqTuningFile // tuning file 的文件路径, tuning file 存放在 rkisp 目录下, SOC sensor 无需配置

```

<Hal_tuning_RKISP1> <!-- Parameters to tune the HAL and hacks for the HAL that are camera dependent -->
  <flipping value="" value_v=""/> <!-- value: SENSOR_FLIP_H or "", value_v: SENSOR_FLIP_V or "" -->
  <supportIsoMap value="false"/>
  <graphSettingsFile value="graph_settings_ov5695.xml"/>
  <iqTuningFile value="OV5695_lens_CHT-842B-MD.xml"/>
</Hal_tuning_RKISP1>

```

<Sensor_info_RKISP1>

- sensorType 修改成对应 sensor 的类型, (RAW or SOC)

```

<Sensor_info_RKISP1> <!-- Information that parametrizes the behavior or qualities of the physical sensor -->
<sensorType value="SENSOR_TYPE_RAW"/> <!-- SENSOR_TYPE_SOC or SENSOR_TYPE_RAW -->
<exposure.sync value="true"/> <!-- compensate exposure sync -->
<sensor.digitalGain value="true"/> <!-- digital gain support on sensor -->
<gain.lag value="2"/> <!-- camera3 HAL CPF parameters moved here start-->
<exposure.lag value="2"/>
<fov value="54.8" value_v="42.5"/>
<statistics.initialSkip value="1"/> <!-- camera3 HAL CPF parameters moved here end-->
<frame.initialSkip value="3"/> <!-- camera3 HAL CPF parameters moved here end-->
<isoAnalogGain1 value="75"/> <!-- Pseudo ISO corresponding analog gain value 1.0. -->
<ITMaxMargin value="10"/> <!-- coarse integration time max margin -->
</Sensor_info_RKISP1>

```

<MediaCtl_elements_RKISP1>

- 修改 type="pixel_array" 中的 element name, // 该 name 为 sensor 在 driver 中的设备名称, 可以通过如下命令获取 adb shell cat /sys/class/video4linux/v4l-subdev*/name 查看

```

<MediaCtl_elements_RKISP1>
  <element name="ov5695_2-0036" type="pixel_array"/>
  <element name="rockchip-sy-mipi-dphy" type="csi_receiver"/>
  <element name="rkisp1-isp-subdev" type="isys_backend"/>
</MediaCtl_elements_RKISP1>

```

2.3.3 xml 运行生效:

参照下一章节

3. 编译运行调试:

3.1 编译:

- 1) 确认 <android_root>/device/rockchip/common/BoardConfig.mk 文件中, 是否有定义宏 BOARD_DEFAULT_CAMERA_HAL_VERSION, 如无定义, 请在文件末添加如下:
BOARD_DEFAULT_CAMERA_HAL_VERSION := 3.3
(Sdk 发部应该带有 hal1 和 hal3 两套源码, 该两套源码编译目标是相同的, 所以同时编译会产生冲突,

因此在编译时加一个宏来判断编译哪一套 hal 源码，如下：`ifeq (1,$(strip $(shell expr $(BOARD_DEFAULT_CAMERA_HAL_VERSION) \>= 3.0)))`，只有当该宏 ≥ 3.0 时才会编译 hal3.)

- 2) 在<android_root> 目录

```
$ source build/envsetup.sh && lunch
```

- 3) 进入 hal3 源码目录

```
$ mma -j8
```

3.2 生成库：

- 1) Hal3 库： /vendor/lib<64>/hw/camera.rk30board.so
- 2) librkisp： /vendor/lib<64>/librkisp.so
- 3) 3a lib： /vendor/lib<64>/rkisp/<ae/awb/af/>
- 4) 配置文件： /vendor/etc/camera/

上述 librkisp、3a lib、配置文件都是通过预编译将<hal3>/lib、<hal3>/etc/camera 中的文件 copy 到 android out 目录。每一项都可以单独更新。当修改源码编译后，只需 push camera.rk30board.so 即可，如修改配置文件，也只需要 push 相应配置文件。

3.3 运行：

1. 将需要更新的库或者 xm 配置文件 push 到板子相应的目录。

```
$ adb root && adb remount
```

```
$ adb push <hal3_camera>/etc/camera /vendor/etc/ (android version >= 8.0)
```

```
$ adb push <hal3_camera>/etc/camera /system/etc/ (android version < 8.0)
```

2. 重新启动 camera 服务进程

```
$ adb shell pkill camera && adb shell pkill provider
```

3. 通过如下命令查看 camera 是否加载成功。

```
$ adb shell dumpsys media.camera
```

4. 如果没有打印出 camera 相关信息(camera 正常信息有好几百行)，则加载失败。

此时：

5. 再次确认配置文件是否有 Push 到板子（普遍是这个问题，请再三确认）：

```
$ adb shell
```

```
$ cat /vendor/camera/camera3_profiles.xml //查看该文件是否是修改过后的文件，
```

```
$ adb logcat|grep "E RkCamera" 查看是否是致命错误，定位分析。
```

6. 如果前三步都没有问题，底层驱动正常，可用 v4l2-ctl 抓到数据，此时 camera 应该可以打开了。

Camera 如果打不开，可以打开相关 camera log 的开关来定位问题。

```
$ adb shell setprop persist.vendor.camera.hal.debug 5
```

4. Dump 说明

为了方便调试，hal3 增加了几个属性值，可以将预览，录像，拍照等数据流直接 dump 到文件。以下是详细说明：

4.1 属性说明：

`persist.vendor.camera.dump`：表示相关数据的 dump 开关，属性值对应不同数据流
例：

```
adb shell setprop persist.vendor.camera.dump 1 #dump preview
adb shell setprop persist.vendor.camera.dump 2 # dump video
adb shell setprop persist.vendor.camera.dump 4 # dump zsl
adb shell setprop persist.vendor.camera.dump 8 # dump jpeg
adb shell setprop persist.vendor.camera.dump 16 # dump raw
// 上述 raw 并非从 sensor 直接出来的数据，而是从 isp 拿到的数据，还未在 Hal 层处理过
adb shell setprop persist.vendor.camera.dump 11 # dump preview + video + jpeg
```

`persist.vendor.camera.dump.skip` 属性表示跳过前面 n 帧

例：\$ adb shell setprop persist.vendor.camera.dump.skip 10
表示前面 10 帧不 dump

`persist.vendor.camera.dump.interval` 属性是表示 dump 帧的间隔

例：\$ adb shell setprop persist.vendor.camera.dump.interval 10
表示隔 10 帧 dump 一帧

`persist.vendor.camera.dump.count` 属性表示 dump 帧的总帧数

例：\$ adb shell setprop persist.vendor.camera.dump.count 100
表示总共只 dump 100 帧

`persist.vendor.camera.dump.path` 属性表示 dump 帧的路径

例：\$ adb shell setprop persist.vendor.camera.dump.path /data/dump/
表示 dump 的路径为 /data/dump/（最后的” /” 不能省）

以下是一个完整例子，表示 dump 预览帧，前面 10 帧不 dump，每隔 10 帧 dump 一次，总共 dump 100 帧，路径为/data/dump/

```
adb shell setprop persist.vendor.camera.dump 1
adb shell setprop persist.vendor.camera.dump.skip 10
adb shell setprop persist.vendor.camera.dump.interval 10
adb shell setprop persist.vendor.camera.dump.count 100
adb shell setprop persist.vendor.camera.dump.path /data/dump/
```

4.2 未生成 dump 文件问题:

Dump 属性设置完成，打开相机，预览后，板子 /data/dump/目录下应该会有如下 dump 文件生成。

```
rk3399_all:/data/dump # ls
dump_1280x960_00000160_PREVIEW_0 dump_1280x960_00000274_PREVIEW_0 dump_1280x960_00000388_PREVIEW_0 dump_1280x960_00000502_PREVIEW_0 dump_1280x960_00000616_PREVIEW_0
dump_1280x960_00000161_PREVIEW_0 dump_1280x960_00000275_PREVIEW_0 dump_1280x960_00000389_PREVIEW_0 dump_1280x960_00000503_PREVIEW_0 dump_1280x960_00000617_PREVIEW_0
dump_1280x960_00000162_PREVIEW_0 dump_1280x960_00000276_PREVIEW_0 dump_1280x960_00000390_PREVIEW_0 dump_1280x960_00000504_PREVIEW_0 dump_1280x960_00000618_PREVIEW_0
dump_1280x960_00000163_PREVIEW_0 dump_1280x960_00000277_PREVIEW_0 dump_1280x960_00000391_PREVIEW_0 dump_1280x960_00000505_PREVIEW_0 dump_1280x960_00000619_PREVIEW_0
dump_1280x960_00000164_PREVIEW_0 dump_1280x960_00000278_PREVIEW_0 dump_1280x960_00000392_PREVIEW_0 dump_1280x960_00000506_PREVIEW_0 dump_1280x960_00000620_PREVIEW_0
```

如果未生成 dump 文件，请参照前面调试说明章节，打开 log 开关。并查看 log 是否有以下错误

```
$ adb logcat |grep "open file failed"
```

```
I RkCamera: <HAL> CameraBuffer: dumpImage filename is /data/dump_1280x960_00000015_PREVIEW_0
E RkCamera: <HAL> CameraBuffer: open file failed
```

该错误是由于 dump 路径无权限访问，或者不存在导致的。可以尝试以下步骤解决。

确认 dump 路径是否存在，不存在请更改目录，或创建目录

目录存在但依然无权限访问，可以使用如下命令暂时关闭 selinux

```
$ adb root && adb shell setenforce 0
```

5. 版本说明:

5.1 Hal3 版本获取:

1. 通过读取属性值获取

```
$ adb shell getprop |grep cam.hal3.ver
```

2 也可以通过查看 logcat 获取

```
$ adb logcat |grep "Hal3 Release version"
```