

密级状态：绝密() 秘密() 内部() 公开(√)

Camera_Engine_Rkisp_User_Manual
(ISP 部)

文件状态： <input type="checkbox"/> 正在修改 <input checked="" type="checkbox"/> 正式发布	当前版本：	v2.0
	作 者：	钟以崇
	完成日期：	2019-5-27
	审 核：	邓达龙
	完成日期：	2019-5-27

福州瑞芯微电子股份有限公司
Fuzhou Rockchips Electronics 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	钟以崇	2018-11-08	发布初版		
V1.1	温暖	2018-12-25	增加 linux 平台 使用说明		
V1.9	钟以崇	2019-03-07	对应 camera engine v1.9.0, 增加 metadata 接 口使用说明		
V2.0	钟以崇	2019-05-27	对应 camera engine v2.0.0 增加 rk1608 调试 说明 增加 iq xml 强校 验说明		

目 录

1. 文档适用说明	4
1.1 适用软件版本	4
2. CAMERA ENGINE 基本框架	4
2.1 driver layer	4
2.2 Engine layer	4
2.3 Interface layer	5
2.4 Application layer	5
3. 源码目录结构	5
4. API 简要说明	6
4.1 Control loop API	6
4.1.1 rkisp_cl_init	6
4.1.2 rkisp_cl_prepare	7
4.1.3 rkisp_cl_start	7
4.1.4 rkisp_cl_stop	7
4.1.5 rkisp_cl_deinit	8
4.1.6 rkisp_cl_set_frame_params	8
4.1.7 设置 metadata 基本步骤	8
4.2 支持的 metadata 列表	9
5. IQ 效果文件相关	11
5.1 IQ 文件名定义规则	11
5.2 IQ 版本校验机制	11
5.3 calibdb 及 IQ xml 文件版本号	11
6. CAMERA_ENGINE 使用与调试	12
6.1 Android 平台使用	12
6.1.1 编译	12
6.2 Android 平台调试	12
6.2.1 log 开关	12
6.2.2 更新库	12
6.3 LINUX 平台使用	13
6.3.1 编译	13
6.3.2 log 开关	13
6.3.3 库及 IQ 文件	13
6.4 Linux 平台集成 camera engine 方法	15

6.4.1 通过 <i>gstreamer</i>	15
6.4.2 通过 <i>v4l2</i> 应用编程.....	16
7. RK1608 适配调试.....	16
7.1 设备驱动调试.....	16
7.2 HAL 层数据流调试.....	17
7.3 <i>camera_engine</i> 3A 调试.....	17
8. FAQ.....	18
8.1 共性 FAQ.....	18
8.1.1 如何获取版本号.....	18
8.1.2 集成 <i>camera_engine</i> 后,3A 并没有自动调整.....	19
8.1.3 <i>rkisp_cl_prepare</i> 未执行完成程序就 <i>crash</i>	19
8.2 Android FAQ.....	19
8.2.1 Android 系统中 LOGV 打印不出来.....	19
8.3 Linux FAQ.....	19
8.3.1 修改部分源码后, 直接编译 <i>camera_engine</i> 代码, 生成的 <i>librkisp.so</i> 中未生效.....	19
8.3.2 <i>media get entity by name: rkisp1_xxx is null</i>	20
8.3.3 Failed to load plugin ‘*** <i>libgstrkisp.so</i> ’: <i>libgstvideo4linux2.so</i>	20

Camera engine 主要实现的是 Raw sensor 的 3A 控制，对于 Linux 系统来说，还可通过在它基础上实现的 libgstrikisp 插件来实现数据流获取等。除了 3A 库源码不开放外，其他部分的代码都是开源的。该文档主要描述了 camera engine 的模块组成，简要 API 说明，编译步骤，及调试方面的注意事项。

1. 文档适用说明

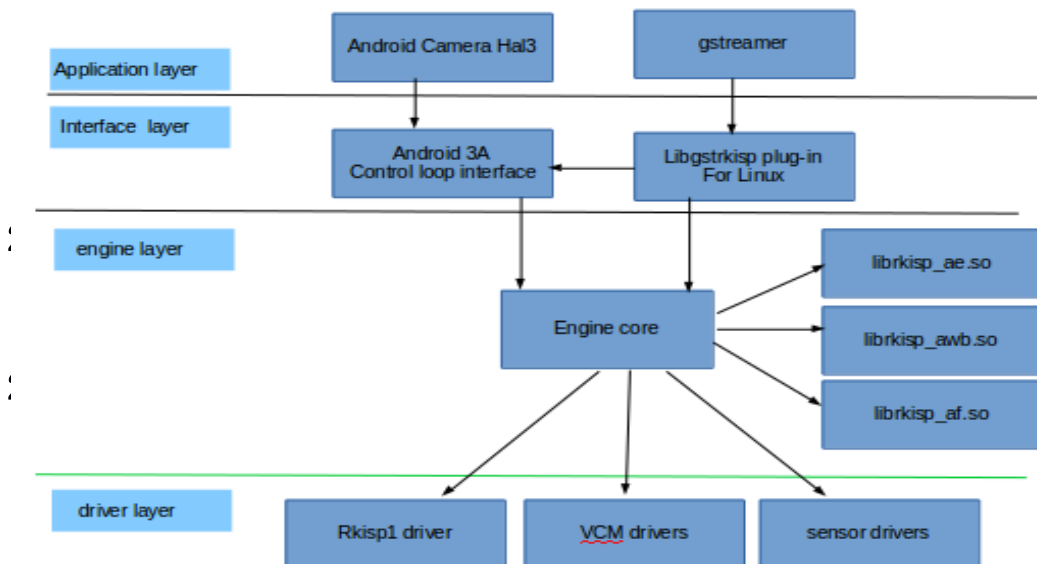
1.1 适用平台及系统

芯片平台	操作系统	支持情况
RK3399/RK3326/RK3288/RK3368	android-9.0	Y
RK3399/RK3326/RK3288/RK1808	Linux(Kernel-4.4)	Y
RV1108	Linux(Kernel-3.10)	N

1.2 适用软件版本

软件类型	版本号
camera_engine_rkisp	v2.0.0

2. Camera engine 基本框架



包括 core engine 库 (librkisp.so) 及 3A 库。Core engine 主体功能为获取驱动数据流, 实现上层帧参数控制, 如 3A 模式等, 从 ISP 驱动获取 3A 统计, 调用 3A 库实现 3A 调整。为上层主要提供的类接口为 DeviceManager。librkisp_ae.so, librkisp_awb.so 及 librkisp_af.so 为 RK 实现的 3A 库, 实现为动态加载库, 且有标准接口, 用户如有需求, 可实现自己的 3A 库进行替换。

2.3 Interface layer

在 engine 层基础上为 Android 及 Linux 封装了不同接口。Android 层不需要数据流部分, 只需要 3A 控制部分, 控制接口及说明请参考头文件 rkisp_control_loop.h, 该文件中对实现的接口以及基本调用流程都有详细说明及注释。libgstrkisp 是为 gstreamer 实现的插件, 通过该插件, 用户可通过 gstreamer 获取数据流以及控制 3A。如用户有其他需求, 可封装满足自己需求的接口层。

2.4 Application layer

应用层, 目前有适配 Android 的 Camera Hal3 及 Linux 平台的 gstreamer。

3. 源码目录结构

```
|—— Android.mk*    // Android 编译 mk
|—— build_system/  // 移植的简易编译系统
|—— config.h*
|—— ext/           // 引用的外部库, 文件等
|—— gstreamer/     // 基于 camera engine 实现的 gstreamer 插件 demo
|—— install*
|—— interface/     // camera engine 提供给外部的接口实现
|—— iqfiles/       // 已调试过的模组 iq 文件
|—— Makefile       // Linux 编译文件
|—— metadata/      // 从 Android 移植, 控制 3A 参数等
|—— modules/       // 适配于 xcore 框架的具体实现
|—— plugins/       // 3A 库及头文件
|—— productConfigs.mk // 编译配置文件
```

```

├── rkisp/           // 3A 库接口层，连接 xcore 框架及 3A 库
├── tests/           // demo 程序
├── update*
├── update_header*  // 更新 Linux 版本 3A 库
├── update_header_android* // 更新 Android 版本 3A 库
└── xcore/          // camera engine 框架，移植自 intel 开源项目

```

4. API 简要说明

Camera engine 主要提供 3A 功能，3A 功能主要由 `interace/rkisp_control_loop.h` 文件提供，以下主要介绍该文件相关接口。

4.1 Control loop API

接口在 `rkisp_control_loop.h` 中已有详细说明，此外，在 `tests/rkisp_demo.cpp` 中有 3A 接口的使用示例，这里简要说明如下：

4.1.1 rkisp_cl_init

[描述]

初始化 control loop。

[语法]

```
int rkisp_cl_init(void** cl_ctx, const char* tuning_file_path,
                  const cl_result_callback_ops_t *callback_ops);
```

[参数]

参数名称	描述	输入输出
<code>cl_ctx</code>	成功返回 control loop context	输出
<code>tuning_file_path</code>	RAW sensor 使用的 tuning xml 文件，engine v2.0.0 开始已不需要提供该文件，engine 中自动选择	输入
<code>callback_ops</code>	接收 result metadata 的回调，提供该回调后，该回调函数每一帧都会被执行一次，返回帧对应的统计信息、所应用的参数及 3A 状态等。	

[返回值]

返回值	描述
0	成功
非 0	失败

4.1.2 rkisp_cl_prepare

[描述]

prepare control loop。

[语法]

```
int rkisp_cl_prepare(void* cl_ctx,
                    const struct rkisp_cl_prepare_params_s*
                    prepare_params);
```

[参数]

参数名称	描述	输入输出
cl_ctx	control loop context	输入
prepare_params	所需控制的设备路径集	输入

[返回值]

返回值	描述
0	成功
非 0	失败

4.1.3 rkisp_cl_start

[描述]

start control loop，调用成功后 control loop 开始运行，3A 开始工作。

[语法]

```
int rkisp_cl_start(void* cl_ctx)
```

[参数]

参数名称	描述	输入输出
cl_ctx	control loop context	输入

[返回值]

返回值	描述
0	成功
非 0	失败

4.1.4 rkisp_cl_stop

[描述]

stop control loop

[语法]

```
int rkisp_cl_stop(void* cl_ctx)
```

[参数]

参数名称	描述	输入输出
cl_ctx	control loop context	输入

[返回值]

返回值	描述
0	成功
非 0	失败

4.1.5 rkisp_cl_deinit

[描述]

反初始化 control loop

[语法]

```
void rkisp_cl_deinit(void* cl_ctx)
```

[参数]

参数名称	描述	输入输出
cl_ctx	control loop context	输入

4.1.6 rkisp_cl_set_frame_params

[描述]

设置新的帧参数，主要包括 3A 模式等

[语法]

```
int rkisp_cl_set_frame_params(const void* cl_ctx,  
                             const struct rkisp_cl_frame_metadata_s* frame_params);
```

[参数]

参数名称	描述	输入输出
cl_ctx	control loop context	输入
frame_params	新的帧参数	输入

[返回值]

返回值	描述
0	成功
非 0	失败

[注意]

参数结构体直接使用 Android 的 camera_metadata_t 结构，可设置的参数请参考 camera_metadata_doc.html，用户可根据该文档描述进行参数设置。tests/rkisp_demo.cpp 提供了一些基本参数的设置及获取示例。

4.1.7 设置 metadata 基本步骤

1) 包含如下头文件：

CameraMetadata.h: 包含了 CameraMetadata 类, 该类封装了 camera_metadata_t 结构体, 使得 metadata 管理更加方便

rkcamera_vendor_tags.h: 包含了 RK 的自定义 metadata。

2) 初始化 metadata

提供一些 camera 的基础能力信息, 如支持的 3A 模式, 最大曝光时间, 支持的帧率范围, 支持的分辨率等等。初始化 metadata 信息需要在 rkisp_cl_prepare 时传入。初始化 metadata 示例代码如下:

```
static void construct_default metas(CameraMetadata* metas)
{
    int64_t exptime_range_ns[2] = {0, 30*1000*1000};
    int32_t sensitivity_range[2] = {0, 3200};
    uint8_t ae_mode = ANDROID_CONTROL_AE_MODE_ON;
    uint8_t control_mode = ANDROID_CONTROL_MODE_AUTO;
    uint8_t ae_lock = ANDROID_CONTROL_AE_LOCK_OFF;
    int64_t exptime_ns = 10*1000*1000;
    int32_t sensitivity = 1600;

    metas->update(ANDROID_SENSOR_INFO_EXPOSURE_TIME_RANGE, exptime_range_ns, 2);
    metas->update(ANDROID_SENSOR_INFO_SENSITIVITY_RANGE, sensitivity_range, 2);
    metas->update(ANDROID_CONTROL_AE_MODE, &ae_mode, 1);
    metas->update(ANDROID_SENSOR_EXPOSURE_TIME, &exptime_ns, 1);
    metas->update(ANDROID_CONTROL_MODE, &control_mode, 1);
    metas->update(ANDROID_SENSOR_SENSITIVITY, &sensitivity, 1);
    metas->update(ANDROID_CONTROL_AE_LOCK, &ae_lock, 1);
}

static void init_3A_control_params()
{
    camera_metadata_t* meta;

    meta = allocate_camera_metadata(DEFAULT_ENTRY_CAP, DEFAULT_DATA_CAP);
    assert(meta);
    g_3A_control_params = new control_params_3A();
    assert(g_3A_control_params);
    g_3A_control_params->result_cb_ops.metadata_result_callback = metadata_result_callback;
    g_3A_control_params->_settings_metadata = meta;
    construct_default_metas(&g_3A_control_params->_settings_metadata);
    g_3A_control_params->_frame_metas.id = 0;
    g_3A_control_params->_frame_metas.metas =
        g_3A_control_params->_settings_metadata.getAndLock();
    g_3A_control_params->_settings_metadata.unlock(g_3A_control_params->_frame_metas.metas);
}
```

数状态。下面以获取当前曝光参数为例:

```
static
void metadata_result_callback(const struct cl_result_callback_ops *ops,
                             struct rkisp_cl_frame_metadata_s *result)
{
    static int rkisp_getAeTime(void* &engine, float &time)
    {
        struct control_params_3A* ctl_params =
            (struct control_params_3A*)engine;
        camera_metadata_entry entry;

        SmartLock lock(ctl_params->_meta_mutex);

        entry = ctl_params->_result_metadata.find(ANDROID_SENSOR_EXPOSURE_TIME);
        if (!entry.count)
            return -1;

        time = entry.data.i64[0] / (1000.0 * 1000.0 * 1000.0);
        printf("expousre time is %f secs\n", time);

        return 0;
    }
}
```

TAG 名称	描述
ANDROID_CONTROL_AE_LOCK	Lock 住 ae
ANDROID_CONTROL_AE_MODE	支持 on/off, off 时为 manual 模式, 可设置手动曝光参数
ANDROID_CONTROL_AE_REGIONS	ae 的测光区域
ANDROID_CONTROL_AE_TARGET_FPS_RANGE	帧率范围, 上限值等于下限值时代表固定帧率
ANDROID_SENSOR_INFO_EXPOSURE_TIME_RANGE	定义曝光时间范围
ANDROID_SENSOR_INFO_SENSITIVITY_RANGE	定义曝光增益范围
ANDROID_CONTROL_AE_STATE	获取当前 ae 状态
ANDROID_CONTROL_AWB_MODE	支持 off/auto/INCANDESCENT/FLUORESCENT/DAYLIGHT/CLOUDY_DAYLIGHT
ANDROID_CONTROL_AWB_REGIONS	Awb 统计窗口
ANDROID_CONTROL_AWB_STATE	获取当前 awb 状态
ANDROID_CONTROL_AF_MODE	支持 OFF/AUTO/CONTINUOUS_PICTURE
ANDROID_CONTROL_AF_REGIONS	af 统计窗口
ANDROID_CONTROL_AF_TRIGGER	主动触发 af 对焦
ANDROID_CONTROL_AF_STATE	获取 af 状态
ANDROID_SENSOR_SENSITIVITY	设置手动曝光时的增益及反馈当前曝光增益
ANDROID_SENSOR_EXPOSURE_TIME	设置手动曝光时的时长及反馈当前曝光时长
RKCAMERA3_PRIVATEDATA_EFFECTIVE_DRIVER_FRAME_ID	反馈的当前帧 metada 对应的帧 id, 与数据帧 id 做对应后, 可做到帧与生效参数的对应
RKCAMERA3_PRIVATEDATA_FRAME_SOF_TIME_STAMP	当前帧的开始传输时刻, 减去曝光时间可知当前帧的起始曝光时刻

5. IQ 效果文件相关

5.1 IQ 文件名定义规则

IQ 文件放置于 iqfiles 文件夹，文件名定义需要遵循以下规则：

<sensor 名称>_<模组名称>_<lens 名称>.xml

上述信息需要与内核中 dts 文件里定义的相一致。否则，camera engine 将找不到对应的 iq 文件。

注：如果 sensor 连接到 preisp(即 RK1608)，再连接到 AP ISP，那么<sensor 名称> 后面需要加上后缀 “-preisp”，即：

<sensor 名称>-preisp_<模组名称>_<lens 名称>.xml

5.2 IQ 版本校验机制

Engine v2.0.0 引入 IQ xml 强校验机制，校验使用的 IQ xml 是否与 engine 库版本相匹配。校验机制会检测 xml 文件中每个 tag 的定义是否与当前 engine 库版本匹配。因此，只修改 xml 中版本号等非正常方式升级 xml 是可能会出现错误的。

如果使用的 iq xml 版本错误，将导致校验失败，camera 应用会退出，搜索 log 会有类似 如下的 calibtags 的 assert 错误：

```
05-26 21:41:45.337 15485 15559 E rkisp : XCAM ERROR calibtags.cpp:3155: calib_check_tag_attrs(3153): parent_tag_id:2 parent_tag_name:header
tag_id:7 tag_name:generator_version --- tag_info type not match (1) != (2)
05-26 21:41:45.337 15485 15559 E rkisp : 
05-26 21:41:45.337 15485 15559 E rkisp : XCAM ERROR calibtags.cpp:3175: calib_check_tag_attrs(3175): parent_tag_id:2 parent_tag_name:header
tag_id:7 tag_name:generator_version --- ASSERT!!!
```

5.3 calibdb 及 IQ xml 文件版本号

calibdb 为 iq xml 解析器，解析器中包含了对应 iq xml 版本的模板定义。每个版本的模板都会定义一个版本号（version number）及版本特征码（magic version code），版本号为字符串，格式如“v1.0.0”，iq xml 中定义的解析器版本号需要与之一致；版本特征码根据具体 iq 版本模板生成，用于标识 iq 版本唯一性，用一个 32 位数据表示，后续 iq tuning tool 可根据该特征码为具体 sensor 生成对应版本的 iq 文件。

版本号及特帧码可通过如下方式获取：

(1) 通过 log 确认，有类似如下信息：

```
I rkisp : *****
I rkisp : Calibdb Version IS:v1.1.0 Magic Version Code IS 677941
I rkisp : *****
I rkisp :
```

(2) 通过源码确认：

rkisp/ia-engine/calib_xml/calibdb.cpp 中有版本信息

```
//v1.0.0: add xml check & magic version code 635075
// start from this version, like va.b.c
// a: show the big change version
// b: show the little change version
// c: show that the content of xml is not changed, but fix some parse bugs.
//v1.1.0: add xml check & magic version code 675496
// add flash ctrl parameters in iq xml file.

/*****
*****/

#define CODE_XML_PARSE_VERSION "v1.1.0"
```

6. camera_engine 使用与调试

6.1 Android 平台使用

6.1.1 编译

1. 将 camera engine 源码放至 <android 工程根目录>/hardware/rockchip/
2. 工程编译环境设置好后, camera engine 源码目录执行 mm 编译
编译后生成 librkisp.so, 3A 库不提供源码, 随工程提供编好的库在
plugins/rkiq/<aec/af/awb>/<lib32/lib64>

6.2 Android 平台调试

6.2.1 log 开关

```
setprop persist.vendor.rkisp.log <level>
level:
    0: error level, defalut level
    1: warning level
    2: info level
    3: verbose level
```

6.2.2 更新库

android 8.x 及以上库路径:

```
librkisp : /vendor/lib<64>
3a: /vendor/lib<64>/rkisp/<ae/awb/af>/
iq: /vendor/etc/camera/rkisp/
```

更新库后重启 camera 服务:

```
pkill provider && pkill camera
```

android 7.x 及以下库路径:

```
librkisp: /system/lib<64>/
3a: /system/lib<64>/rkisp/<ae/awb/af>/
iq: /system/etc/camera/rkisp/
更新库后重启 camera 服务:
pkill camera*
```

6.3 Linux 平台使用

6.3.1 编译

(1) 配置 productConfigs.mk

配置编译工具链路径: CROSS_COMPILE, 如果使用的是 linux sdk 工程则不需要该步骤。

(2) 编译

可通过 ARCH=arm 或者 aarch64 来指定编译 32 位或者 64 位库

32 bit 编译:

```
make ARCH= arm
```

64 bit 编译:

```
make ARCH=aarch64
```

编译成功后库文件生成在 camera engine 工程目录 build 文件夹下。3A 库不提供源码, 随工程提供编好的库在 plugins/rkiq/<aec/af/awb>/<lib32/lib64>

6.3.2 log 开关

```
export persist_camera_engine_log=<level>
```

level:

0: error level, defalut level

1: warning level

2: info level

3: verbose level

6.3.3 库及 IQ 文件

(1) 库文件

camera_engine_rkisp 需要将 5 个库文件 push 到板子里。

- 1) librkisp.so push 到板子的/usr/lib/
- 2) librkisp_aec.so push 到板子的/usr/lib/rkisp/ae/
- 3) librkisp_awb.so push 到板子的/usr/lib/rkisp/awb/
- 4) librkisp_af.so push 到板子的/usr/lib/rkisp/af/
- 5) libgstarkisp.so push 到板子的/usr/lib/gstreamer-1.0/

注：若不使用 gstreamer 可以不用 push libgstrkisp.so)
 在 buildroot 系统中，已自动将全部的库拷贝到系统中，
 buildroot/package/rockchip/camera_engine_rkisp/camera_engine_rkisp.mk
 如下图：

```

RKgstDir = $(TARGET_DIR)/usr/lib/gstreamer-1.0
RKafDir = $(TARGET_DIR)/usr/lib/rkisp/af
RKaeDir = $(TARGET_DIR)/usr/lib/rkisp/ae
RKawbDir = $(TARGET_DIR)/usr/lib/rkisp/awb

define CAMERA_ENGINE_RKISP_INSTALL_TARGET_CMDS
    mkdir -p $(RKgstDir)
    mkdir -p $(RKafDir)
    mkdir -p $(RKaeDir)
    mkdir -p $(RKawbDir)
    mkdir -p $(TARGET_DIR)/etc/iqfiles
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/set_pipeline.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/camera_rkisp.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/S50set_pipeline $(TARGET_DIR)/etc/init.d/
    $(INSTALL) -D -m 755 $(@D)/build/bin/rkisp_demo $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 644 $(@D)/iqfiles/*.xml $(TARGET_DIR)/etc/iqfiles/
    $(INSTALL) -D -m 644 $(@D)/build/lib/librkisp.so $(TARGET_DIR)/usr/lib/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/af/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_af.so $(RKafDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/aec/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_aec.so $(RKaeDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/awb/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_awb.so $(RKawbDir)/
    $(INSTALL) -D -m 644 $(@D)/build/lib/libgstrkisp.so $(RKgstDir)/
endef

```

图 6.3.3-1

(2) IQ 文件

在 SDK 工程目录中，在 etc/external/camera_engine_rkisp/iqfiles 目录下统一存放 IQ 文件。如果需要加入新的 IQ 文件，就放在此目录下，并且 IQ 名字规范参照前述 iq 文件定义 章节，然后删除以下目录 buildroot/output/rockchip_rkxxxx_xx/build/camera_engine_rkisp-1.0，最后重新编译 buildroot。

在 buildroot 系统中，IQ 文件会统一拷贝到板子的/etc/iqfiles/目录下，如图 6.3.3-2。(buildroot/package/rockchip/camera_engine_rkisp/camera_engine_rkisp.mk)

```

RKgstDir = $(TARGET_DIR)/usr/lib/gstreamer-1.0
RKafDir = $(TARGET_DIR)/usr/lib/rkisp/af
RKaeDir = $(TARGET_DIR)/usr/lib/rkisp/ae
RKawbDir = $(TARGET_DIR)/usr/lib/rkisp/awb

define CAMERA_ENGINE_RKISP_INSTALL_TARGET_CMDS
    mkdir -p $(RKgstDir)
    mkdir -p $(RKafDir)
    mkdir -p $(RKaeDir)
    mkdir -p $(RKawbDir)
    mkdir -p $(TARGET_DIR)/etc/iqfiles
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/set_pipeline.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/camera_rkisp.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/S50set_pipeline $(TARGET_DIR)/etc/init.d/
    $(INSTALL) -D -m 755 $(@D)/build/bin/rkisp_demo $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 644 $(@D)/iqfiles/*.xml $(TARGET_DIR)/etc/iqfiles/
    $(INSTALL) -D -m 644 $(@D)/build/lib/librkisp.so $(TARGET_DIR)/usr/lib/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/af/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_af.so $(RKafDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/aec/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_aec.so $(RKaeDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/awb/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_awb.so $(RKawbDir)/
    $(INSTALL) -D -m 644 $(@D)/build/lib/libgstrkisp.so $(RKgstDir)/
endef

```

图 6.3.3-2

当系统启动后，会运行/etc/init.d/S50set_pipeline start，这里会匹配当前连接的 sensor，如图 6.3.3-3 所示，


```

if [[ $MP_NODE =~ "/dev/video" ]]
then
    set_pipeline.sh --sensorbayer $BAYER --sensorname "$NAME"
    if [[ $SENSOR ]]
    then
        ln -fs /etc/iqfiles/$SENSOR*.xml /etc/cam_iq.xml
    fi
fi
done

```

图 6.3.3-3

通过名字找到/etc/iqfiles/目录下匹配的 xml 文件，链接成/etc/cam_iq.xml，如图 6.3.3-4 所示，当前 cam_iq.xml 链接的是 OV5695.xml。

```

- entity 7: ov5695 2-0036 (1 pad, 1 link)
  type V4L2 subdev subtype Sensor flags 0
  device node name /dev/v4l-subdev2
  pad0: Source
    [fmt:SBGGR10_1X10/2592x1944@10000/300000 field:none]
    -> "rockchip-mipi-dphy-rx":0 [ENABLED]

/ # ls -l /etc/cam_iq.xml
lrwxrwxrwx 1 root root 23 Aug 5 09:12 /etc/cam_iq.xml -> /etc/iqfiles/OV5695.xml

```

图 6.3.3-4

注意：camera engine v1.9.0 版本后，iq 文件已不可由外部传入，camera engine 中根据从 sensor 驱动查询到的信息自动进行 iq 文件匹配。

6.4 Linux 平台集成 camera engine 方法

Camera_engine_rkisp 使用方式有两种：1、通过 gstreamer ，2、V4L2 应用编程。

6.4.1 通过 gstreamer

Camera_engine_rkisp 的使用通过以 plugin 的形式通过 gst-launch-1.0 实现。测试前我们需要指明动态库的路径：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/gstreamer-1.0
```

通过以下命令可以测试

```
gst-launch-1.0 rkisp device=/dev/video1 io-mode=1 analyzer=1 enable-3a=1 path-iqf=/etc/cam_iq.xml ! video/x-raw,format=NV12,width=640,height=480, framerate=30/1 ! videoconvert ! autovideosink
```

若没有显示设备，需要 dump 图像，可以将以上命令 ‘autovideosink’ 修改为 ‘filesink location=/tmp/streamer.yuv’，最后通过 yuv 工具预览。

Buildroot 中可以直接使用 camera_rkisp.sh 测试。

6.4.2 通过 v4l2 应用编程

我们提供了 rkisp_demo 供客户参考测试。如图 6.4.2-1 代码在工程的 tests/下 rkisp_dmeo 随工程生成在目录 build/bin/

```
leo@leo:~/rk3326/external/camera_engine_rkisp$ ls tests/
Android.mk  build  rkisp_demo.cpp  rkisp_demo.o  test_camcl.cpp  test_camcl.o
leo@leo:~/rk3326/external/camera_engine_rkisp$ ls build/bin/
rkisp_demo  test_ispcl
```

图 6.4.2-1

Buildroot 系统中，已经将 rkisp_dmeo 拷贝到/usr/bin/下 (buildroot/package/rockchip/camera_engine_rkisp/camera_engine_rkisp.mk)

```
RKgstDir = $(TARGET_DIR)/usr/lib/gstreamer-1.0
RKafDir = $(TARGET_DIR)/usr/lib/rkisp/af
RKaeDir = $(TARGET_DIR)/usr/lib/rkisp/ae
RKawbDir = $(TARGET_DIR)/usr/lib/rkisp/awb

define CAMERA_ENGINE_RKISP_INSTALL_TARGET_CMDS
    mkdir -p $(RKgstDir)
    mkdir -p $(RKafDir)
    mkdir -p $(RKaeDir)
    mkdir -p $(RKawbDir)
    mkdir -p $(TARGET_DIR)/etc/iqfiles
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/set_pipeline.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/camera_rkisp.sh $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 755 $(TOPDIR)/package/rockchip/camera_engine_rkisp/S50set_pipeline $(TARGET_DIR)/etc/init.d/
    $(INSTALL) -D -m 755 $(@D)/build/bin/rkisp_demo $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 644 $(@D)/iqfiles/*.xml $(TARGET_DIR)/etc/iqfiles/
    $(INSTALL) -D -m 644 $(@D)/build/lib/librkisp.so $(TARGET_DIR)/usr/lib/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/af/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_af.so $(RKafDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/aec/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_aec.so $(RKaeDir)/
    $(INSTALL) -D -m 644 $(@D)/plugins/3a/rkiq/awb/$(CAMERA_ENGINE_RKISP_LIB)/librkisp_awb.so $(RKawbDir)/
    $(INSTALL) -D -m 644 $(@D)/build/lib/libgstrkisp.so $(RKgstDir)/
endef
```

图 6.4.2-2

使用方法：如图 6.4.2-3，可以通过 rkisp_demo -h 查看，最后会在指定的 ouput 目录下生成图像数据，再通过 yuv 工具预览。

```
rkisp_demo: Add some control parameters(device, output...) for the rkisp_demo

--width, default 640, optional, width of image
--height, default 480, optional, height of image
--format, default NV12, optional, fourcc of format
--count, default 5, optional, how many frames to capture
--iqfile, default /etc/cam_iq.xml, optional, camera IQ file
--device, required, path of video device
--output, required, output file path
--verbose, optional, print more log

Example:
rkisp_demo --device=/dev/video1 --output=/tmp/test.yuv \
--width=1920 --height=1080 --count=10
```

图 6.4.2-3

7. RK1608 适配调试

7.1 设备驱动调试

参考《RKISP_Driver_User_Manual_v1.2》驱动调试文档bringup rk1608，与普通 sensor 调试类似。

RK1608 可以实现不同的算法，不同的算法对应不同的 RK1608 固件，固件的在工程中的存储路径如下：

hardware/rockchip/camera/etc/firmware/

目前支持的固件列表如下：

固件名称	描述
fw_rk1608_bypass.rkl	Rk1608 bypass 固件，RK1608 输入数据不做任何处理，bypass 直接输出。这边的 bypass 指的是 mipi csi rx 直接 bypass 到 mipi csi tx，数据都未进入 RK1608 端 ddr；
fw_rk1608_ov2718_2frame.rkl	RK1608 集成 ov2718 DCG HDR 算法；

生效固件的配置方式，详见《RKISP_Driver_User_Manual_v1.2》文档中第 6.3 章节：Rk1608 AP 设备注册 (DTS) ， firmware-names。

7.2 HAL 层数据流调试

设备驱动调试成功后可进行 HAL 调试，可参考 HAL3 调试文档。

1. 配置 camera3_profiles.xml 文件

hardware/rockchip/camera/camera_etc.mk：配置 HAL3 加载哪个 profile；

hardware/rockchip/camera /camera3_profiles_rk3399_1608.xml：RK1608 配置 camera3_profiles.xml 参考配置文件；具体参见 HAL 配置文档《camera_hal3_user_manual_v2.1》。

2. 配置为 SOC 类型调试数据流

camera3_profiles_xxx.xml 文件中<sensorType value=""/>配置成：
<sensorType value="SENSOR_TYPE_SOC"/>。

该配置将决定不启动 camera_engine，即 3A 控制 bypass；

3. 配置 RK1608 采用 bypass 固件

7.3 camera_engine 3A 调试

HAL 层预览调试成功后可进行 3A 效果调试，建议步骤如下：

1. 配置为 RAW 类型使能 camera_engine

camera3_profiles_xxx.xml 文件中<sensorType value=""/>配置成：
<sensorType value="SENSOR_TYPE_RAW"/>。

2. 如果 RK1608 固件集成的算法会影响 3A，建议配置该模组对应的 IQ 文件，关闭其相应功能来调试基础 3A

模组对应的 IQ 文件，规则参考：5.1 IQ 文件名定义规则

举例：

RK1608 实现 HDR 算法，该算法会影响 AP 端 ISP AE，建议将 IQ 文件中的 HDR AE 功能功能关闭

IQ 文件：HdrCtrl 项中：

enable = 0: 关闭 HDR AE，采用线性(Linear)AE；

enable = 1: 使能 HDR AE；

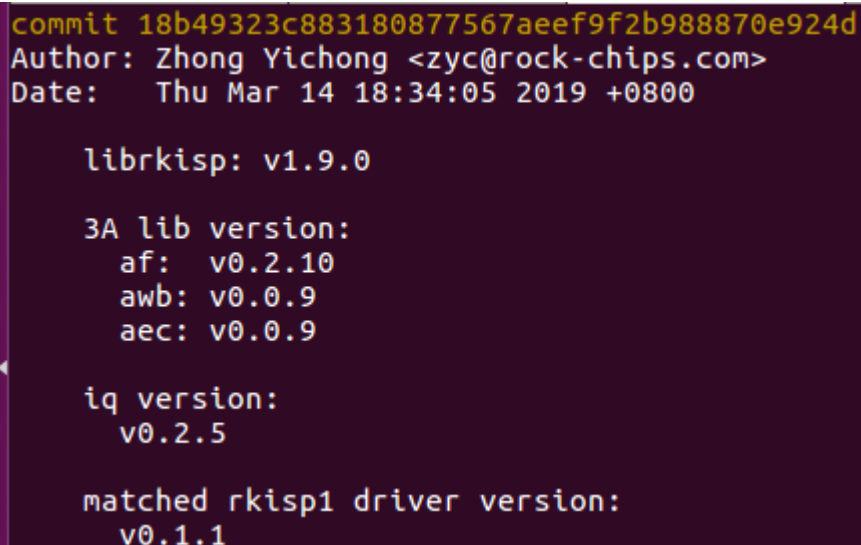
8. FAQ

8.1 共性 FAQ

8.1.1 如何获取版本号

方式一：

查看 engine 工程 git log，有类似如下信息：



```
commit 18b49323c883180877567aeef9f2b988870e924d
Author: Zhong Yichong <zyc@rock-chips.com>
Date: Thu Mar 14 18:34:05 2019 +0800

    librkisp: v1.9.0

    3A lib version:
    af: v0.2.10
    awb: v0.0.9
    aec: v0.0.9

    iq version:
    v0.2.5

    matched rkisp1 driver version:
    v0.1.1
```

方式二：

查看 engine 源码，各库版本信息文件路径如下：

librkisp.so: <engine project>/interface/rkisp_dev_manager.h

librkisp_af.so: <engine project>/plugins/3a/rkiq/af/af.h

librkisp_aec.so: <engine project>/plugins/3a/rkiq/ae/aec.h

librkisp_awb.so: <engine project>/plugins/3a/rkiq/awb/awb.h

方式三：

查看 log。打开 log 开关后，执行以下命令：

(1) pkill provider && pkill camera

(2) 然后打开 camera 应用

(3) logcat | grep version -i，输出如下：

```
04-07 16:33:33.713 2107 2219 I rkisp : AE LIB VERSION IS v0.0.9
04-07 16:33:33.713 2107 2219 I rkisp : AWB LIB VERSION IS v0.0.9
04-07 16:33:33.714 2107 2219 I rkisp : AF LIB VERSION IS v0.2.10
04-07 16:33:33.736 2107 2219 I rkisp : YUV420P TO RGB565 (3107) 714400 : 1.000000
```

8.1.2 集成 camera_engine 后, 3A 并没有自动调整

A: 打开 log 开关, 查找进入应用时的 log,

"failed to get iq file name"

"load tuning file failed"

以上错误信息代表 IQ 文件未找到或者解析 IQ 文件时出错, 需要检查 IQ 文件名是否与内核 dts 中定义的信息一致;

8.1.3 rkisp_cl_prepare 未执行完成程序就 crash

A: 1. camera_engine_rkisp v2.0.0 之前的版本, IQ 文件与 camera_engine 集成的 calidb 的匹配失败会出现该问题, v2.0.0 之后 IQ 文件增加了版本校验机制, 详见: 5.2 IQ 版本校验机制

2. 其他因素

8.2 Android FAQ

8.2.1 Android 系统中 LOGV 打印不出来

A: 建议更新至 camera_engine_rkisp: v1.9.0 及其以上版本

8.3 Linux FAQ

8.3.1 修改部分源码后, 直接编译 camera_engine 代码, 生成的 librkisp.so 中未生效

A: 修改非 interface 文件夹中代码时, 建议执行以下步骤:

#make clean && make

8.3.2 media get entity by name: rkisp1_xxx is null

```
[root@rk3326_64:/etc]# camera_rkisp.sh
Setting pipeline to PAUSED ...
media get entity by name: rkisp1_mainpath is null
media get entity by name: rkisp1_selfpath is null
media get entity by name: rkisp1_isp_subdev is null
media get entity by name: rkisp1_input_params is null
media get entity by name: rkisp1_statistics is null
media get entity by name: rockchip_sy_mipi_dphy is null
media get entity by name: lens is null
Caught SIGSEGV
exec gdb failed: No such file or directory
Spinning. Please run 'gdb gst-launch-1.0 578' to continue debugging, Ctrl-C to quit, or Ctrl-\ to dump core.
```

A: /dev/videoX 设备节点没有指定正确

8.3.3 Failed to load plugin ‘**libgstarkisp.so’:libgstvideo4linux2.so

```
/ # gst-launch-1.0 rkisp device=/dev/video1 io-mode=1 analyzer=1 enable-3a=1 pat
h=igf=/etc/cam iq.xml ! video/x-raw,format=NV12,width=640,height=480,framerate=
30/1 ! videoconvert ! autovideosink
(gst-launch-1.0:583): GStreamer-WARNING **: Failed to load plugin '/usr/lib/gstreamer-1.0/libgstarkisp.so': libgstvideo4linux2.so: cannot open shared object file: No such file or directory
WARNING: erroneous pipeline: no element "rkisp"
```

A: 动态链接库路径没有配置，例如：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/gstreamer-1.0
```