

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:	V1.0
	作 者:	钟以崇
	完成日期:	2018-11-08
	审 核:	邓达龙
	完成日期:	2018-11-14

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		
V2.0	钟以崇	2019-03-07	对应 camera engine v1.9.0		

目 录

文档适用平台.....	3
CAMERA ENGINE 基本框架.....	3
driver layer.....	4
Engine layer.....	4
Interface layer.....	4
Application layer.....	4
源码目录结构.....	4
API 简要说明.....	5
Android CL API.....	5
rkisp_cl_init.....	5
rkisp_cl_prepare.....	6
rkisp_cl_start.....	7
rkisp_cl_stop.....	8
rkisp_cl_deinit.....	8
rkisp_cl_set_frame_params.....	9
设置 metadata 基本步骤.....	9
支持的 metadata 列表.....	12
IQ 文件名定义.....	14
编译.....	14
Android 平台编译.....	14
Linux 平台编译.....	14
调试.....	15
Android 平台调试.....	15
log 开关.....	15
更新库.....	15
Linux 平台调试.....	16
log 开关.....	16
更新库.....	16
FAQ.....	16

LINUX 平台使用.....	17
1、库及 IQ 文件.....	17
1.1 库文件.....	17
1.2 IQ 文件.....	18
2、使用方法.....	20
2.1 通过 gstreamer.....	20
2.2 通过 v4l2 应用编程.....	20
3、常见问题.....	21

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。

文档适用平台

芯片平台	驱动	操作系统	支持情况
RK3288	Linux(Kernel-4.4):rkispl driver	Android 9.0 Linux	Y
RK3399	Linux(Kernel-4.4):rkispl driver	Android 9.0 Linux	Y
RK3326	Linux(Kernel-4.4):rkispl driver	Android-9.0 Linux	Y
RK3368	Linux(Kernel-4.4):rkispl driver	Android-9.0 Linux	Y

Camera engine 基本框架

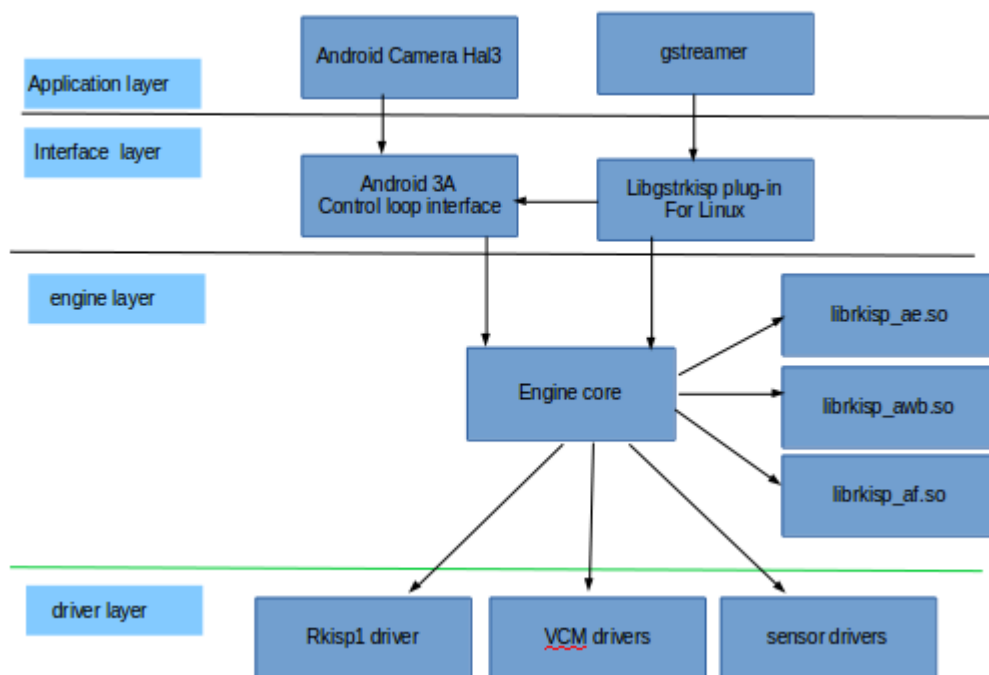


图 1 Camera engine 模块结构

上图各模块简要说明如下：

driver layer

为驱动层，不在本文描述范围内，具体参考《RKISP_Driver_User_Manual_v1.0》。

Engine layer

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。

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。

Application layer

应用层, 目前有适配 Android 的 Camera Ha13 及 Linux 平台的 gstreamer。

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
```

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。

Android CL 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：

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

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

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

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

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

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。

设置 **metadata** 基本步骤

1) 包含头文件 `CameraMetadata.h`, `rkcamera_vendor_tags.h`

`CameraMetadata.h` 中包含了 `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：

3) 参数设置

```

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);
    params.staticMeta =
        g_3A_control_params->_settings_metadata.getAndLock();
    RKISPFunc.prepare_func(_rkisp_engine, &params);
    g_3A_control_params->_settings_metadata.unlock(params.staticMeta);
}

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);
}

```

以设置固定曝光参数为例，其他参数可使用类似流程就行设置。

```

static int rkisp_setManualGainAndTime(void* &engine, float hal_gain, float hal_time)
{
    struct control_params_3A* ctl_params =
        (struct control_params_3A*)engine;

    int64_t exptime_ns = hal_time * 1000 * 1000 * 1000;
    // set to manual mode
    uint8_t ae_mode = ANDROID_CONTROL_AE_MODE_OFF;
    // convert to ISO100 unit
    int32_t sensitivity = hal_gain * 100;

    ctl_params->_settings_metadata.update(ANDROID_SENSOR_SENSITIVITY, &sensitivity, 1);
    ctl_params->_settings_metadata.update(ANDROID_CONTROL_AE_MODE, &ae_mode, 1);
    ctl_params->_settings_metadata.update(ANDROID_SENSOR_EXPOSURE_TIME, &exptime_ns, 1);
    ctl_params->_frame_metas.id++;
    ctl_params->_frame_metas.metas =
        ctl_params->_settings_metadata.getAndLock();
    ctl_params->_settings_metadata.unlock(ctl_params->_frame_metas.metas);

    _RKISPFunc.set_frame_params_func(_rkisp_engine,
        &ctl_params->_frame_metas);

    return 0;
}

```

需要说明的是，设置参数时所传入的 metadata 可重新构造，也可以与初始化时传入的复用；但是，如果使用不同的 metadata 变量，则一些初始化时的参数不可再通过 rkisp_cl_set_frame_params 进行更改，如

ANDROID_SENSOR_INFO_EXPOSURE_TIME_RANGE,
ANDROID_SENSOR_INFO_SENSITIVITY_RANGE 等定义能力信息（这些通常被认为是静态信息，不可更改）的。

4) 获取参数

rkisp_cl_init 时需传入 metadata callback 函数，engine 从驱动中得到统计数据，然后根据当前设置，计算出新的参数后，会将当前统计帧对应的参数及 3A 状态等通过该 callback 返回。demo 中的示例只保存了最新的参数状态。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)
{
    camera_metadata_entry entry;
    struct control_params_3A* ctl_params =
        (struct control_params_3A*)ops;

    SmartLock lock(ctl_params->_meta_mutex);
    /* this will clone results to _result_metadata */
    ctl_params->_result_metadata = result->metas;
    printf("meta callback!\n");
}
```

```
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;
}
```

支持的 metadata 列表

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, 范围为固定值时则为固定帧率
ANDROID_SENSOR_INFO_EXPOSURE_TIME_RANGE	定义曝光时长范围
ANDROID_SENSOR_INFO_SENSITIVITY_RANGE	定义曝光增益范围
ANDROID_CONTROL_AE_STATE	反馈当前 ae 状态
ANDROID_CONTROL_AWB_MODE	支持 off/auto/INCANDESCENT/FLUORESCENT/DAYLIGHT/CLOUDY_DAYLIGHT
ANDROID_CONTROL_AWB_REGIONS	Awb 统计区域
ANDROID_CONTROL_AWB_STATE	反馈当前 awb 状态
ANDROID_CONTROL_AF_MODE	支持 OFF/AUTO/CONTINUOUS_PICTURE
ANDROID_CONTROL_AF_REGIONS	af 统计区域
ANDROID_CONTROL_AF_TRIGGER	主动触发 af 对焦
ANDROID_CONTROL_AF_STATE	反馈 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_TIMESTAMP	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

iq 文件名定义

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。

需要注意的是，如果 sensor 连接到 preisp（如 rk1608），再连接到 isp，那么 <sensor 名称> 后面需要加上后缀 “-preisp”。

编译

Android 平台编译

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>

Linux 平台编译

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>

调试

Android 平台调试

log 开关

```
setprop persist.vendor.rkisp.log <level>
```

level:

- 0: error level, default level
- 1: warning level
- 2: info level
- 3: verbose level

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*
```

Linux 平台调试

log 开关

`export persist_camera_engine_log=<level>`

level:

- 0: error level, default level
- 1: warning level
- 2: info level
- 3: verbose level

更新库

库路径:

librkisp: /usr/lib/

3a: /usr/lib/rkisp/<ae/awb/af>/

FAQ

(1) Q: Android 平台中 LOGV 打印不出来

A: bug, 还未找到原因, 可暂时先将 LOGV 改成其他如 LOGD。

(2) Q: 修改 engine 中代码后 make, 但是生成的 librkisp.so 中却未生效

A: 修改非 interface 文件夹中代码时, 需要先 make clean 再 make。

(3) Q: 预览时曝光, 白平衡没有自动调整

A: 打开 log 开关, 查找进入应用时的 log, 看是否有 "failed to get iq file name" 及 "load tuning file failed" 等 error, fail 信息, 如有则表明 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, 以及 iq 文件内容是否有问题, 比如 iq 版本是否匹配等。

(4) Q: rkisp_cl_prepare 未执行完成程序就 crash

A: 如果没有什么出错信息输出，一般是在解析 iq xml 时出错，因为目前代码 xml 解析部分的 log 未开启（如果是 android 平台，即使开启也打印不出来，主要是由于 android 平台将控制台 log 重定向了），需要检查所使用的 iq 文件是否与代码相匹配。

Linux 平台使用

1、库及 IQ 文件

1.1 库文件

参考《编译》的章节，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. libgstrkisp.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，如图 1.1-1。

(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 1.1-1

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 1.2-1。
(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 1.2-1

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 1.2-2 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 1.2-2

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 1.2-3 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 1.2-3

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。

2、使用方法

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

2.1 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
```

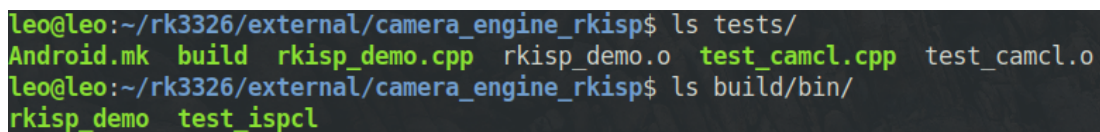
若没有显示设备，需要 dump 图像，可以将以上命令‘autovideosink’修改为‘filesink
location=/tmp/streamer.yuv’，最后通过 yuv 工具预览。

Buildroot 中可以直接使用 camera_rkisp.sh 测试。

2.2 通过 v4l2 应用编程

我们提供了 rkisp_demo 供客户参考测试。如图 2.2-1 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 2.2-1

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 2.2-2

使用方法：如图 2.2-3，可以通过 rkisp_demo -h 查看，最后会在指定的 output 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 2.2-3

3、常见问题

1、如图 3.1，若遇到以下错误提示，是 /dev/video 没有指定正确。

```

[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.

```


图 3.1

2、如图 3.2，若遇到以下错误提示，是动态库的路径没有指定。

```
// # gst-launch-1.0 rkisp device=/dev/video1 io-mode=1 analyzer=1 enable-3a=1 pat
h=igfss/etc/cam_ig.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/libgstvkisp.so': libgstvideo4linux2.so: cannot open shared object file: No such file or directory
WARNING: erroneous pipeline: no element "rkisp"
```

图 3.2