Rockchip

Linux Camera 开发指南

发布版本:V1.1

日期:2019.03

免责声明

本文档按“现状”提供，福州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 © 2019 福州瑞芯微电子股份有限公司

福州瑞芯微电子股份有限公司
Fuzhou 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： +86-591-83991906
客户服务传真： +86-591-83951833
客户服务邮箱： service@rock-chips.com

前言

概述

本文档主要介绍 Rockchip 系列芯片的 CIF， ISP 的新驱动结构， 以及在此基础上， 如何编写/移植 Sensor 驱动，上层如何应用 demo 及工具测试。

为了适应用户的定制需求，本文所描述的 CIF，ISP 及 Camera 驱动，都尽可能满足 V4L2 标准接口，提供更为丰富的的配置。同时，也对客户编写及移植驱动有更高的要求，用户需要了解更多底层的拓扑结构，V4L2 相关的概念等等。

产品版本

芯片名称	内核版本	CIF	ISP
PX3SE	4.4	有	无
RK312x	4.4	有	无
RK3288	4.4	有	有
RK332x	4.4	有	有
RK3399	4.4	无	无

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	版本	作者	修改说明
2019.7.10	V1.0	ZSQ	添加初始版本
2019.3.12	V1.1	Leo Wen	更新 isp 驱动描述 添加 gstreamer 应用编程 更新 3A 功能说明等

约定

- 代码及命令，本文中代码及命令均用灰色背景填充
- 命令格式，本文中示例的操作命令， 如遇多行， 则使用转义字符 ‘\’ 将一行命令拆成多行，用户在使用时可以直接复制粘贴，但如果用户是将命令放在一行中， 请去掉转义字符 ‘\

目录

目录

插图目录

插图目录

表格目录

表格目录

1 Camera 及驱动介绍

1.1 驱动版本

Camera 在本文泛指 Rockchip 芯片中的 CIF 或 ISP 及其连接的 Sensor。并且本文描述 CIF 及 ISP 驱动指的是基于 Media Controller, Async sub device 和 vb2、以符合 V4L2 框架标准为目标的新版本驱动。

各驱动版本及支持情况如下表格。

驱动名称	类型	Kernel 版本	平台	是否在本文范畴
Rkisp1	ISP	4.4	Linux	是
Rkcif	CIF	4.4	Linu	是
Oneframe	CIF	3.10 或 4.4	Linu	否
Rk-isp10	ISP	4.4	Linu	是，详见注释[1]

表 1 驱动版本

1.2 相关概念

- 3A，指自动聚焦（AF），自动曝光（AE）和自动白平衡（AWB）算法，或者算法.so 库
- Async Sub Device，特指在 Media Controller 结构下的 V4L2 子设备
- Bayer Raw，或者 Raw Bayer，表示设备（Sensor 或 ISP）输出的 RGGB，BGGR，GBRG，GRBG 等格式，或该格式的数据帧
- Buildroot, 特指 Rockchip 基于 Buildroot[2] 发布的一系列 Linux SDK
- CIF，Camera Interface， 即 Camera 接口， 用以接收 Sensor 数据并保存到 Memory 中
- DVP，一种并行接口，即 Digital Video Port
- Entity, 本文指 Media Controller 框架下的各节点
- FCC 或 FourCC，即 Four Character(FCC) codes，指 Linux Kernel 中用 4 个字符表示的图像格式，详见 FourCC
- HSYNC，行同步信号
- ISP，Image Signal Processing，用以接收并处理图像

1[] Rk-isp10 仍然在支持中， 但它并没有采用 media controller, async sub device 的结构， 并不是本文重点。
2[] Buildroot 详见其官方网站，https://buildroot.org/

- IOMMU，Input-Output Memory Management Unit，本文指 Rockchip 系列芯片中的 iommu 模块，用于将物理上分散的内存页映射成 cif、isp 可见的连续内存
- IQ，Image Quality，本文指为 bayer raw camera 调试的 IQ 或 IQ 对应的 xml。用于 3A tunning
- Media Controller，Linux kernel 的一种媒体框架
- MIPI，本文指 MIPI 协议
- MIPI-DPHY，本文指 MIPI-DPHY 协议，或 Rockchip 芯片中符合 MIPI-DPHY 协议的控制器
- MP，即 Main Path，指 Rockchip ISP 的一个输出节点，可输出全分辨率图像，一般用来拍照，抓取 Raw 图
- PCLK，即 Pixel clock
- Pipeline，本文指 media controller 下各个 entity 相互连接形成的链路
- SP，即 Self Path，指 Rockchip ISP 的一个输出节点，最高只能输出 1080p 分辨率，一般用作预览
- Userspace，即 Linux 用户空间（相对于 Linux 内核空间），本文特指用户层或在用户层执行的程序
- V4L2，即 Video4Linux2，Linux kernel 的视频处理模块
- VIP，在 Rockchip 芯片中，即 Video Input Processor， 曾作为 CIF 的别名，已不再使用
- VSYNC，场同步信号

2 CIF 驱动及 dts 配置

本章主要介绍 CIF 驱动及 dts 的配置。并详细描述 CIF 驱动在各芯片上的测试验证情况。

CIF 即 Camera Interface，在 Rockchip 系列芯片中曾经用 VIP(Video Input Processor)一词。二者都是指 Camera Interface。

CIF 在各芯片 Linux SDK 上的支持情况如下表。

平台	DVP 接口	MIPI CSI	裁剪(Crop)	放大 (Scale)	模式	最大分辨率	iommu
PX3SE	支持	不支持	支持	不支持	单帧		无
RK3288	支持	待验证	支持	不支持	单帧		有
RK3326	支持	待验证	支持	不支持	单帧		有

表 2 各芯片的 CIF 验证情况

注意：

- 单帧模式下， 如果图像分辨率较高， 比如 1080p，那么 fps 可能只能达到 15fps
- MIPI CSI，在某些平台下， CIF 能够接收从 MIPI CSI 传输过来的图像
- 如果无 iommu 功能并在 reqbuf 时采用 mmap 方式分配 buffer，要求 kernel 保留较大的 CMA size。可以在 kernel defconfig 中修改，如下。

```
CONFIG_CMA_SIZE_MBYTES=64
```

2.1 CIF 驱动代码简介

CIF 内核驱动代码位于 drivers/media/platform/rockchip/cif/目录，其对应的 device tree binding 文档位于 Documentation/devicetree/bindings/media/rockchip-cif.txt。通过编译开关 CONFIG_VIDEO_ROCKCHIP_CIF 控制是否编译。

CIF 驱动根据 media controller、v4l2、 vb2 框架，完成硬件配置，帧中断处理，buffer 轮转

等功能。各文件的功能内容如下。

```
$ tree drivers/media/platform/rockchip/cif/
drivers/media/platform/rockchip/cif/
├── capture.c  #主要完成硬件配置， v4l2、vb2 框架下的相关回调，帧中断处理
├── dev.c  #主要完成 probe，sub-device 异步(Async)注册，iommu 及 clk 管理
├── dev.h  #驱动相关结构体定义
├── regs.h  #寄存器宏定义
```

- CIF oneframe(单帧)模式。单帧模式下，驱动每收到一个帧中断，在中断处理函数中设置下一帧 buffer 地址，然后再开始采集下一帧数据。
- CIF pingpong(双 buffers)模式。驱动尚未实现。

2.2 CIF dts 配置

请首先参考 Documentation/devicetree/bindings/media/rockchip-cif.txt。该文档会随着驱动代码及时更新。在芯片的 dtsi 中，一般配置好了 cif 的基本信息，包括但不限于：

- Reg，寄存器偏移地址
- Clocks，所需要的 clocks。Clock-names 需要和驱动中定义的相同。
- Reset，可用 CRU 软件复位 CIF
- Interrupts
- Iommu，如果 cif 有 iommu 的情况下一般都会启用 iommu。

2.2.1 板级配置

首先确认对应的芯片级 dtsi 中，是否存在新的 cif 驱动的节点定义。请用 compatible 区分新旧 cif 驱动。

- 旧的 cif 驱动

```
compatible = "rockchip,cif";
```

- 新的 cif 驱动

```
compatible = "rockchip,rk3xxx-cif";
```

然后确认该 cif 是否有 iommu 功能， 如果有 iommu 节点也需要设置为”okay”状态。

最后还需要定义 Remote Port[1]。Remote Port 将 Sensor 与 CIF 连接起来，在 kernel 初始化过程中，Sensor 与 CIF 异步注册， 二者最终根据 dts 中 Remote Port 信息绑定起来。

示例一，px3se evb 板子上 cif dts 配置。

CIF 节点定义在 arch/arm/boot/dts/rk312x.dtsi，

```
cif_new: cif-new@1010a000 {
    compatible = "rockchip,rk3128-cif";
    reg = <0x1010a000 0x200>;
    clocks = <&cru ACLK_CIF>, <&cru HCLK_CIF>,
        <&cru SCLK_CIF_OUT>;
    clock-names = "aclk_cif", "hclk_cif",
        "sclk_cif_out";
    resets = <&cru SRST_CIF0>;
    reset-names = "rst_cif";
    interrupts = <GIC_SPI 8 IRQ_TYPE_LEVEL_HIGH>;
    /* rk312x has not iommu attached */
    /* iommus = <&cif_mmu>; */
    power-domains = <&power RK3128_PD_VIO>;
    status = "disabled";
};
```

板级配置在 arch/arm/boot/dts/px3se-evb.dts， 需要引用 cif_new，并修改 status 状态为 okay，最后加上 port 子节点。

```
&cif_new {
    status = "okay";
    port {
        cif_in: endpoint {
            remote-endpoint = <&adv7181_out>;
            vsync-active = <0>;
            hsync-active = <1>;
```

1[] Remote Port 更多信息请参考 Documentation/devicetree/bindings/media/video-interfaces.txt

```
            };
        };
    };
```

Port 子节点定义了 cif_in 节点，并声明与它链接的远端节点为 adv7191_out。因为 adv7181 采用 dvp 接口，这里也同时指定 vsync, hsync 的有效状态，其中 0 表示低电平有效， 1 表示高电平有效。

2.3 CIF 驱动调试及常见问题

本小节介绍如何判断 CIF 设备的状态，如何打开 debug 开关，以及利用 v4l2-ctl 抓帧，利用 mplayer 回放图像，gstreamer 预览，及常见问题。

本小节中的命令是基于 px3se-evb 板，其它板子可能各不相同，特别是/dev/media0 及/dev/video0 设备节点的序号可能不同。甚至 video0 设备的序号在 px3se-evb 板子上也有可能变更，请参考 判断 cif 是否 probe 成功 这一小节中关于如何获取 video 设备编号的方法。

2.3.1 判断 cif 是否 probe 成功

CIF 如果 probe 成功，会有 video 及 media 设备存在于/dev/目录下。例如/dev/media0[1]设备。

系统中可能存在多个/dev/video 设备，可以通过/sys 下的节点查询到 cif 对应的 video 节点。

```
[root@px3se:/]# grep -H '' /sys/class/video4linux/video*/name
/sys/class/video4linux/video0/name:stream_cif
```

可以看出，cif 设备对应到 video0 节点，即/dev/video0 是 cif 设备。

另外，还可以通过 media-ctl，打印拓扑结构查看 pipeline 是否正常。请参考 media-ctl 及拓扑结构。

如果有错误， 请从 kernel log 中查找是否有 cif 相关的错误 log。例如，

```
[root@px3se:/]# dmesg | grep -i cif
```

注意：

- 用户如果需要向 Rockchip 报告 cif 的 bug，issue，请提供完整的 kernel log。Log 越完整，越有助于分析问题。

1[] 如果存在多个 media 设备，编号不一定是 0。例如 3288 上 CIF 和 ISP 同时启用， 会有两个 media 设备：/dev/media0 及/dev/media1。

2.3.2 判断 Sensor 与 CIF 是否已经绑定

如前文所述，CIF 与 Sensor 分别异步加载(probe)，如果二者驱动都加载成功，最后会绑定在一起。此时，kernel log 会有相应提示。

```
[root@px3se:/]# dmesg | grep Async
[    2.681364] rkcif: Async subdev notifier completed
```

看到”Async subdev notifier completed”，即说明 Sensor 与 CIF 成功绑定。

同时，用户仍然可以通过查看 media 拓扑结构，应有 cif 及 sensor 两个 entity 存在。请参考 media-ctl 及拓扑结构。

如果发现异步注册没有成功，即没有“Async subdev notifier completed”这行 log，请分别检查 cif 及 sensor probe 是否出错。比较经常碰到的是 Sensor 驱动上电时序不对，Sensor I2C 通讯失败等。

2.3.3 打开 debug 开关

CIF 驱动中包含一些 v4l2_dbg() log。通过命令可以打开 log 开关，如下。

```
echo 1 > /sys/module/video_rkcif/parameters/debug
```

打开 vb2 相关的 log。这部分 log 主要包括 buffer 的轮转，如 reqbuf，qbuf, dqbuf 及 buffer 状态变化等。如下。需要注意 vb2 模块开关是通用的开关，其它使用了 vb2（如 VPU/ISP 等）的相关 log 也会使能输出。

```
echo 7 > /sys/module/videobuf2_core/parameters/debug
```

打开 v4l2 相关的 log， 比如 ioctl 调用。如下命令将 v4l2 相关 log 全部打开。

```
echo 0x1f > /sys/class/video4linux/video0/dev_debug
```

也可以分别只开一小部分的 log。如下宏[1]定义了各个 bit 会 enable 哪些 log。将所需要的 log 对应的 bit 打开即可。

```
/* Just log the ioctl name + error code */
#define V4L2_DEV_DEBUG_IOCTL        0x01
/* Log the ioctl name arguments + error code */
#define V4L2_DEV_DEBUG_IOCTL_ARG    0x02
/* Log the file operations open, release, mmap and get_unmapped_area */
```

1[] 这些宏定义在 kernel 头文件 include/media/v4l2-ioctl.h 中。

```
#define V4L2_DEV_DEBUG_FOP          0x04
/* Log the read and write file operations and the VIDIOC_(D)QBUF ioctls */
#define V4L2_DEV_DEBUG_STREAMING    0x08
/* Log poll() */
#define V4L2_DEV_DEBUG_POLL         0x10
```

2.3.4 常见问题 QA

3 RKISP1 驱动及 dts 配置

ISP 较 CIF 更为复杂，功能也丰富很多，本章主要介绍 RKISP1 的驱动代码结构、dts 配置、debug 方法等。与 CIF 一样，ISP 驱动符合 media controller，v4l2，vb2 框架，与 mipi-dphy，Sensor 相互独立且通过异步注册。

3.1 RKISP1 驱动代码结构及其框图

本章节描述 RKISP1 驱动代码位于 kernel 的 drivers/media/platform/rockchip/isp1/目录。它主要是依据 v4l2 / media framework 实现硬件的配置，中断处理，控制 buffer 轮转，控制 subdevice(如 mipi dphy 及 sensor)的上下电等功能。简单介绍驱动中各个文件的内容如下。

```
$ tree drivers/media/platform/rockchip/isp1/
drivers/media/platform/rockchip/isp1/
├── capture.c  #包含 mp/sp 的配置及 vb2，帧中断处理
├── dev.c #包含 probe，异步注册，clock，pipeline，iommu 及 media/v4l2 framework
├── isp_params.c #3A 相关参数设置
├── isp_stats.c  # 3A 相关统计
├── regs.c  #寄存器相关的读写操作
├── rkisp1.c  # 对应 isp_sd entity 节点，包含从 mipi 接收数据，并有 crop 功能

$ ls drivers/phy/rockchip/phy-rockchip-mipi-rx.c
drivers/phy/rockchip/phy-rockchip-mipi-rx.c # mipi dphy 驱动
```

如下框图简单描述 ISP 内部的结构。

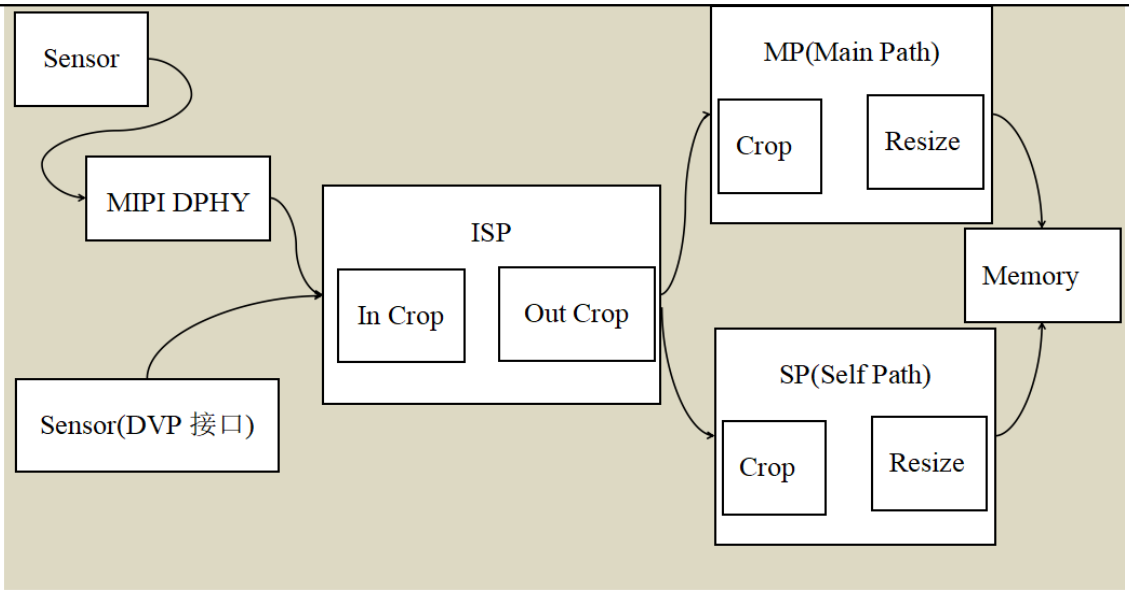

图 1 RKISP1 简单框图

以上框图主要用以体现 ISP 输入与输出端的对于用户使用需要了解的功能特性

- ISP 可以适配 DVP 接口
- ISP 可以适配 MIPI 接口，但需要有 mipi dphy
- 图像输入到 ISP 后，可以分成两路 MP 和 SP 输出。MP 和 SP 是同一张图像，但 resize, crop 可以不同，输出格式也可以不同
- MP，即 Main Path。可以输出全分辨率的图像，最大到 4416x3312。
- MP 可以输出 yuv 或 raw 图，且仅 MP 可以输出 raw 图
- SP，即 Self Path。最高支持 1920x1080 分辨率
- SP 可以输出 yuv 或 rgb 图像，但不能输出 raw 图
- MP 和 SP 都有 crop 和 resize 功能，且相互不影响

输出设备	最大分辨率	支持格式	Crop/Resize
SP	1920 x 1080	YUV, RGB	支持
MP	4416 x 3312	YUV, RAW	支持

表 3 MP 与 SP 输出功能比较

3.2 RKISP1 dts 配置

与 CIF 相同， RKISP1 的 DTS binding 也有完整的 Document 位

于 Documentation/devicetree/bindings/media/rockchip-isp1.txt。请首先参考该文档，它会跟驱动保持同步更新。在 RK Linux SDK 发布时， 若芯片支持 ISP，其 dtsi 中已经有定义好 rkisp1 节点，如 rk3288-rkisp1.dtsi 中的 isp 节点，rk3399.dtsi 中的 rkisp1_0，rkisp1_1 节点。下表描述各芯片 ISP 的信息。

	ISP 个数	dts 节点名称	对应的 mipi dphy	Iommu 节点
RK3288	1	isp	RX0 或 RX1	支持，isp_mmu
RK3399	2	rkisp1_0	RX0	支持，isp0_mmu
		rkisp1_1	TX1RX1	支持，isp1_mmu

表 4 各芯片的 ISP dts 节点信息

注意：

- RK3288.dtsi 也注册了 isp 节点， 但那是对应旧的驱动。新的驱动注册在 rk3288-rkisp1.dtsi 文件中。请在板级配置中， 包含该文件
- RK3399.dtsi 中也注册了多个 isp 节点，请注意本章所描述的是 rkisp1_0 及 rkisp1_1
- Iommu 需要和 isp 一同 enable 或 disable
- 如有多个 isp（比如 rk3399），请注意 iommu 节点要选对

3.2.1 Rkisp1 dts 的板级配置

板级配置的方式与 CIF 类似，isp、mipi-dphy、sensor 是单独定义的节点， 三者之间通过 remote endpoint 相互建立连接。在结构框图上，mipi-dphy 连接了 isp 及 sensor。以下示例是基

于 RK3399 挖掘机 上的 dts 板级配置，可以参考文件 rk3399-sapphire-excavator-edp.dts。在该板子上， RK3399 的两个 ISP 都启用了，这里仅举例 rkisp1_0。

首先， 将 rkisp1_0 节点设置为”okay”状态。

```
&isp0 {
    status = "disabled";
};

&rkisp1_0 {
    status = "okay";
    port {
        #address-cells = <1>;
        #size-cells = <0>;
        isp0_mipi_in: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&dphy_rx0_out>;
        };
    };
};
```

注意，

- 其它版本的 isp 驱动的节点状态需要 disabled。否则两套不同的驱动会冲突
- Port 节点中定义了该 rkisp1_0 是与 dphy_rx0_out 节点相互连接的

其次，rkisp1_0 对应的 iommu 也需要是”okay”状态。

```
&isp0_mmu {
    status = "okay";
};
```

第三，该 sensor 是 mipi 接口，因此需要把 mipi dphy 也启用。

```
&mipi_dphy_rx0 {
    status = "okay";
```

```
        ports {
            #address-cells = <1>;
            #size-cells = <0>;
            port@0 {
                reg = <0>;
                #address-cells = <1>;
                #size-cells = <0>;
                mipi_in_ucam0: endpoint@1 {
                    reg = <1>;
                    remote-endpoint = <&ucam_out0>;
                    data-lanes = <1 2>;
                };
            };
            port@1 {
                reg = <1>;
                #address-cells = <1>;
                #size-cells = <0>;
                dphy_rx0_out: endpoint@0 {
                    reg = <0>;
                    remote-endpoint = <&isp0_mipi_in>;
                };
            };
        };
    };
```

注意：

- Dphy 既连接 isp，也连接 sensor。因此它有两个 endpoint。mipi_in_ucam0 是与 Sensor 相连接，dphy_rx0_out 与 isp 相连接
- mipi_in_ucam0 中声明了 data-lanes，指明了 sensor 使用了两个 lane。如果有 4 个 lanes，这里就应该定义成<1 2 3 4>。以此类推。

最后板级的 sensor 节点定义好，并且也需要声明 port 子节点，与 mipi dphy 的 dphy_rx0_out 相连接。

3.3 RKISP1 调试及常见的问题

本小节主要介绍如何调试 ISP，判断驱动是否加载成功及常见问题 QA 等。因为 ISP 在结构上与 CIF 类似，也使用相同的框架， 用户也可以参考 CIF 驱动调试及常见问题。

3.3.1 判断驱动 probe 状态

RKISP1 如果 probe 成功，会有 video 及 media 设备存在于/dev/目录下。系统中可能存在多个/dev/video 设备，通过/sys 可以查询到 RKISP1 注册的 video 节点。 如下是 RK3399 Dru Chrome Table 上的执行命令及结果[1]（有删剪）。

```
localhost ~ # grep '' /sys/class/video4linux/video*/name
/sys/class/video4linux/video3/name:rkisp1_selfpath
/sys/class/video4linux/video4/name:rkisp1_mainpath
/sys/class/video4linux/video5/name:rkisp1-statistics
/sys/class/video4linux/video6/name:rkisp1-input-params
```

RKISP1 驱动会注册 4 个设备，分别为 selfpath，mainpath，statistics，input-params。其中前两个是用于帧输出， 后两个是用于 3A 的参数设置与统计。通过查找 video4linux 下的 sys 节点，我们得到 RK3399 Dru 的 RKISP1 信息如下表。

设备节点	名称	功能
/dev/video3	SP(Self Path)	视频输出
/dev/video4	MP(Main Path)	视频输出
/dev/video5	Statistics	3A 统计
/dev/video6	Input-params	3A 参数设置

1[] 不同的板子返回会有差异，特别是当 VPU/CIF 等设备也有启用时。他们都是基于 v4l2 框架。

表 5 rkisp1 注册的各 video 设备

注意：

- Rkisp1 驱动会注册 4 个/dev/video 设备，编号连续。
- 如 RK3399 平台的两个 isp 同时启用，共会注册 8 个/dev/video 设备。前 4 个属于一个 isp，后 4 个属于另外一个 isp。软件上尚无法区分两个 isp 注册的先后顺序。
- 该例子中， video 编号由 3 开始。因为还有 vpu 设备占用了 0, 1, 2 三个 video 节点。

用户还可以通过 media-ctl，打印拓扑结构查看 pipeline 是否正常。请参考 media-ctl 及拓扑结构。

如果上述两种方法返回结果有错误， 请从 kernel log 中查找是否有 cif 相关的错误 log。例如，

```
[root@px3se:/]# dmesg | grep -i isp
```

注意：

- 用户如果需要向 Rockchip 报告相关的 bug，issue，请提供完整的 kernel log。Log 越完整，越有助于分析问题。

3.3.2 判断 sub-device 是否绑定成功

请通过 media-ctl 打印拓扑结构查看 dts 所注册的 Sensor 是否都异步注册成为一个 Entity。并且当所有的 Sensor 都注册完毕，kernel 会打印出如下的 log。

```
localhost ~ # dmesg | grep Async
[    0.682982] rkisp1: Async subdev notifier completed
```

如发现 kernel 没有 Async subdev notifier completed 这行 log， 或者拓扑结构与预期不符，那么请首先查看 sensor 是否有相关的报错，I2C 通讯是否成功。

3.3.3 打开调试开关

这里介绍三个调试开关，rkisp1 驱动的 v4l2_dbg 开关，vb2 模块的 log 开关，v4l2 框架层的 log 开关。

Rkisp1 驱动包含一些 v4l2_dbg，如 format, size 参数设置，中断中丢帧等信息。如下。

```
echo 1 > /sys/module/video_rkisp1/parameters/debug
```

Vb2 模块及 v4l2 框架层的 log 开关， 与 CIF 一样。 请参考 打开 debug 开关。需要注意的两点如下。

- 所有使用了 vb2 的模块如 isp, vpu, cif 模块都会被 vb2 log 开关使能并输出
- V4l2 框架层的 log 是针对某个特定的/dev/video 节点，因此只会使能特定的 video 设备。关于如何查找用户所需的 video 设备，请参考本章的 判断驱动 probe 状态 小节

3.3.4 常见问题 QA

4 Sensor 驱动开发移植

Sensor 驱动位于 drivers/media/i2c 目录下，注意到本章节所描述的是具有 media controller 属性的 sensor 驱动， 故 drivers/media/i2c/soc_camera 目录下的驱动并不适用。

Sensor 驱动与 CIF 或者 RKISP1 驱动完全独立，二者异步注册，最后通过 remote-endpoint 建立连接。因此本章所描述的 Sensor 驱动同时适用于 CIF 和 RKISP1。

在 Media Controller 结构下， Sensor 一般作为 sub-device 并通过 pad 与 cif、isp 或者 mipi phy 链接在一起。本章主要介绍 Sensor 驱动的代码[1]， dts 配置，及如何验证 sensor 驱动的正确性。

本章将 Sensor 驱动的开发移植概括为 5 个部分，

- 按照 datasheet 编写上电时序， 主要包括 vdd，reset，powerdown，clk 等。
- 配置 sensor 的寄存器以输出所需的分辨率、格式。
- 编写 struct v4l2_subdev_ops 所需要的回调函数，一般包括 set_fmt，get_fmt，ov5695_s_stream
- 增加 v4l2 controller 用来设置如 fps，exposure，gain，test pattern
- 编写.probe()函数，并添加 media control 及 v4l2 sub device 初始化代码

作为良好的习惯，完成驱动编码后，也需要增加相应的 Documentation，可以参考 Documentation/devicetree/bindings/media/i2c/。这样板级 dts 可以根据该文档快速配置。

在板级 dts 中，引用 Sensor 驱动，一般需要

- 配置正确的 clk，io mux
- 根据原理图设置上电时序所需要的 regulator 及 gpio
- 增加 port 子节点，与 cif 或者 isp 建立连接

本章以 ov5695 及 ov2685 为例，简单分析 Sensor 驱动。

4.1 上电时序

不同 Sensor 对上电时序要求不同，例如 OV Camera。可能很大部分的 OV Camera 对时序要求不严格，只要 mclk，vdd，reset 或 powerdown 状态是对的，就能正确进行 I2C 通讯并输出图

1[] Sensor 驱动开发要求用户了解 v4l2 框架的一些基本知识，本章只基于已有代码进行简要分析，梳理结构。实际开发过程中可能会碰到各种各样的问题，大部分在这里无法解释到。

片。但还是有小部分 Sensor 对上电要求非常严格，例如 OV2685 必须严格按时序上电。

在 Sensor 厂家提供的 DataSheet 中，一般会有上电时序图，只需要按顺序配置即可。以 OV5695 为例，其中__ov5695_power_on()即是用来给 Sensor 上电。如下（有删减）。

```
static int __ov5695_power_on(struct ov5695 *ov5695)
{
    ret = clk_prepare_enable(ov5695->xvclk);
    if (!IS_ERR(ov5695->reset_gpio))
        gpiod_set_value_cansleep(ov5695->reset_gpio, 1);
    ret = regulator_bulk_enable(OV5695_NUM_SUPPLIES, ov5695->supplies);
    if (!IS_ERR(ov5695->reset_gpio))
        gpiod_set_value_cansleep(ov5695->reset_gpio, 0);
    if (!IS_ERR(ov5695->pwdn_gpio))
        gpiod_set_value_cansleep(ov5695->pwdn_gpio, 1);
    /* 8192 cycles prior to first SCCB transaction */
    delay_us = ov5695_cal_delay(8192);
    usleep_range(delay_us, delay_us * 2);
    return 0;
}
```

OV5695 的上电时序简要说明如下，

- 首先提供 xvclk(即 mclk)
- 紧接着 reset pin 使能
- 各路的 vdd 上电。这里使用了 regulator_bulk，因为 vdd, vodd, avdd 三者无严格顺序。如果 vdd 之间有严格的要求，需要分开处理，可参考 OV2685 驱动代码
- Vdd 上电后， 取消 Sensor Reset，powerdown 状态。Reset, powerdown 可能只需要一个，Sensor 封装不同， 可能有差异
- 最后按时序要求，需要 8192 个 clk cycle 之后，上电才算完成。

注意，虽然不按 datasheet 要求上电许多 Sensor 也能正常工作，但按原厂建议的时序操作，无疑是最可靠的。

同样，datasheet 中还会有下电时序(Power Down Sequence)，也同样按要求即可。

4.1.1 判断上电时序是否正确

在.probe()阶段会去尝试读取 chip id，如 ov5695 的 ov5695_check_sensor_id()，如果能够正确读取到 chip id，一般就认为上电时序正确，Sensor 能够正常进行 i2c 通信。

4.2 Sensor 初始化寄存器列表

在 OV5695 及 OV2685 中，各定义了 struct ov5695_mode 及 struct ov2685_mode，用来表示 sensor 不同的初始化 mode。Mode 可以包括如分辨率，mbus code，寄存器初始化列表等。

寄存器初始化列表，请按厂家提供的直接填入即可。需要注意的是， 列表最后用了 REG_NULL 表示结束。

4.3 V4l2_subdev_ops 回调函数

V4l2_subdev_ops 回调函数是 Sensor 驱动中逻辑控制的核心。回调函数包括非常多的功能，具体可以查看 kernel 代码 include/media/v4l2-subdev.h。建议 Sensor 驱动至少包括如下回调函数。

- .open，这样上层才可以打开/dev/v4l-subdev?节点。在上层需要单独对 sensor 设置 v4l control 时，.open()是必须实现的
- .s_stream，即 set stream，包括 stream on 和 stream off，一般在这里配置寄存器，使其输出图像
- .enum_mbus_code，枚举驱动支持的 mbus_code
- .enum_frame_size，枚举驱动支持的分辨率
- .get_fmt，返回当前 Sensor 选中的 format/size。如果.get_fmt 缺失，media-ctl 工具无法查看 sensor entity 当前配置的 fmt
- .set_fmt，设置 Sensor 的 format/size

以上回调中， .s_stream stream_on 会比较复杂些。在 ov5695 驱动代码中，它包括 pm_runtime 使能（即唤醒并上电），配置 control 信息（v4l2 control 可能会在 sensor 下电时配置）即 v4l2_ctrl_handler_setup()，并最终写入寄存器 stream on。

4.4 V4l2 controller

对于需要配置 fps，exposure, gain, blanking 的场景，v4l2 controller 部分是必要的。OV5695 驱动代码中，

- ov5695_initialize_controls()，用来声明支持哪些 control，并设置最大最小值等信息
- Struct v4l2_ctrl_ops，包含了 ov5695_set_ctrl()回调函数，用以响应上层的设置。

4.5 Probe 函数及注册 media entity, v4l2 subdev

Probe 函数中， 首先对 dts 进行解析，获取 regulator, gpio, clk 等信息用以对 sensor 上下电。其次注册 media entity, v4l2 subdev，及 v4l2 controller 信息。注意到 v4l2 subdev 的注册是异步。如下几个关键的函数调用。

- v4l2_i2c_subdev_init(), 注册为一个 v4l2 subdev，参数中提供回调函数
- ov5695_initialize_controls()，初始化 v4l2 controls
- media_entity_init()，注册成为一个 media entity，OV5695 仅有一个输出，即 Source Pad
- v4l2_async_register_subdev()，声明 sensor 需要异步注册。因为 RKISP1 及 CIF 都采用异步注册 sub device，所以这个调用是必须的。

4.6 dts 示例

Sensor 的 dts 配置大同小异，根据硬件的设计，主要是 pinctl(iomux)，clk，gpio，以及 remote port。

以下示例是在 RK3399 Dru Chrome Tablet 上的 OV5695 dts 节点。

```
wcam: camera@36 {
    compatible = "ovti,ov5695";
    reg = <0x36>;
    pinctrl-names = "default";
    pinctrl-0 = <&wcam_rst &test_clkout1>;
    clocks = <&cru SCLK_TESTCLKOUT1>;
    clock-names = "xvclk";
    avdd-supply = <&pp2800_cam>;
    dvdd-supply = <&pp1250_cam>;
    dovdd-supply = <&pp1800_s0>;
    reset-gpios = <&gpio2 5 GPIO_ACTIVE_LOW>;
```

```
            port {
                wcam_out: endpoint {
                    remote-endpoint = <&mipi_in_wcam>;
                    data-lanes = <1 2>;
                };
            };
        };
```

注意：

- Pinctrl，声明了必要的 pinctrl， 该例子中包括了 reset pin 初始化和 clk iomux
- Clock，指定名称为 xvclk（驱动会讯取名为 xvclk 的 clock），即 24M 时钟
- Vdd supply，OV5695 需要的三路供电
- Port 子节点，定义了一个 endpoint，声明需要与 mipi_in_wcam 建立连接。同样地 mipi dphy 会引用 wcam_out
- Data-lanes 指定了 OV5695 使用两个 lane

4.7 Sensor 调试及常见 QA

4.7.1 Sensor 是否注册成功

Sensor 调试的第一个关键节点是 i2c 能否通讯成功，chip id 检查是否正确。如果是，一般能说明上电时序没有问题。驱动中， 一般也会打印出相关的 log，不同 Sensor 的 log 都不太一样，不再举例。第二，检查 media-ctl 的拓扑结构，查看 Sensor 是否已经注册成一个 entity。如果是，说明 Sensor 已经注册成功。

4.7.2 Sensor 是否有数据输出

条件允许的情况下，直接测量硬件 MIPI 信号 CLK 及 Data Lane 是最准确的手段。是否有信号输出，以及信号幅度是否正确等等。

软件上可以通过判断 CIF 或者 ISP 是否有中断。比如 kernel log 是否有 CIF 或 ISP 报告错误。

4.7.3 检查 control 是否生效

可以利用 v4l2-ctl 设置相关的参数，如 gain, exposure, blanking 并生成图片，查看 sensor 的

controls 是否有生效。例如增加 gain 或 exposure，图片亮度是否增加；加大 blanking，帧率是否下降。

4.7.4 常见 QA

Q：已经重复检查了上电时序满足 datasheet 要求，但 I2c 通讯不成功

A：之前碰到过这样几个原因，如下。

- Io domain 设置不对导致 i2c clk、data 输出幅度不对（虽然有波形）。比如供电是 1.8v，io domain 却设置成 3.3v
- Regulator start up delay 太大，导致虽然代码中配置的上电时序是对，但实际测量仍不满足要求。比如某个板子上 OV2685 pp1250_dvdd 所用的 regulator 从输入有效到输出有效，有 740us 的 delay，导致它上电太慢，时序出错
- I2C slave address 配置错误

5 v4l-utils 工具及应用

v4l-utils 工具是由 Linuxtv[1] 维护的一个 V4L2 开发套件， 它提供一系列 V4L2 及 media framework 相关的工具，用来配置 V4L2 子设备的属性，测试 V4L2 设备，并提供如 libv4l2.so 开发库等等。

这一章主要介绍 v4l-utils 中的两个命令行工具：media-ctl 以及 v4l2-ctl。

- media-ctl， 用以查看、配置拓扑结构
- v4l2-ctl， 用以配置 v4l2 controls，可抓帧，设置 cif，isp，sensor 参数

不同版本的 v4l-utils 所对应的参数，format 代码会有些不同，特别是 mbus-fmt 部分。本文所采用的版本是集成在 Linux SDK 中的 v4l-utils-1.14.1。

5.1 获取并编译 v4l-utils

在 Rockchip 发布的 Linux SDK 中， 默认已集成了 v4l-utils 包。用户可以通过 buildroot 的编译开关开启或关闭 v4l-utils 包。如 SDK 目录下，buildroot/configs/rockchip_px3se_defconfig 文件中， 如下这一行将 v4l-utils 编译选项打开：

```
BR2_PACKAGE_LIBV4L_UTILS=y
```

用户也可以在 Linuxtv 的官网获取源码编译，编译请参考 wiki。

V4l-utils 包在 ubuntu 系统下， 也可通过 apt 工具直接安装，如下。

```
$ sudo apt-get install v4l-utils
```

5.2 FourCC

FourCC，全称 Four Character Codes，它用 4 个字符（即 32bit）来命名图像格式。在 Linux Kernel 中，它是一个宏，定义如下：

```
#define v4l2_fourcc(a,b,c,d) \
(((__u32)(a)<<0)|((__u32)(b)<<8)|((__u32)(c)<<16)|((__u32)(d)<<24))
```

FourCC 所定义的格式， 是图像视频在内存中存储的格式。这点要注意和下文的 mbus-fmt 区

1[] Linuxtv 是一个开源组织， 维护 Linux 多媒体模块。详见 https://www.linuxtv.org/

分。

以下列出本文中常用到的几个格式。更详细的定义请参阅 kernel 代码之 videodev2.h。

图像格式	FourCC
V4L2_PIX_FMT_NV12	NV12
V4L2_PIX_FMT_NV21	NV21
V4L2_PIX_FMT_NV16	NV16
V4L2_PIX_FMT_NV61	NV61
V4L2_PIX_FMT_NV12M	NM12
V4L2_PIX_FMT_YUYV	YUYV
V4L2_PIX_FMT_YUV420	YU12
V4L2_PIX_FMT_SBGGR10	BG10
V4L2_PIX_FMT_SGBRG10	GB10
V4L2_PIX_FMT_SGRBG10	BA10
V4L2_PIX_FMT_SRGGB10	RG10

表 6 本文常用的 FourCC 代码

5.3 mbus-fmt

Mbus-fmt，全称是 Media Bus Pixel Codes，它描述的是用于在物理总线上传输的格式， 比如 sensor 通过 mipi dphy 向 isp 传输的图像格式， 或者在 ISP 内部各子模块间传输的格式。特别需要将 Mbus-fmt 与上一小节的 FourCC 区分， 后者是专指存储在 Memory 中的图像格式。

下表列出本文中常用到的几种 Mbus-fmt。

名称	类型	Bpp	Bus width	Sampes per Pixel
MEDIA_BUS_FMT_SBGGR10_1X10	Bayer Raw	10	10	1
MEDIA_BUS_FMT_SGBRG10_1X10	Bayer Raw	10	10	1
MEDIA_BUS_FMT_YUYV8_2X8	YUV:422	16	8	2
MEDIA_BUS_FMT_YUYV8_1_5X8	YUV:420	12	8	1.5

表 7 本文中常用的几种 mbus-fmt

最新的 media-ctl 可以列举出所支持的 mbus code。如下

```
media-ctl --known-mbus-fmts
```

5.4 media-ctl 及拓扑结构

media-ctl 是 v4l-utils 包中的一个工具，主要用来配置 media framework 的各节点 fmt，size，link。用户可以通过如下 help 来查询使用方法。本小节也将介绍几种常用的例子。

```
[root@px3se:/]# media-ctl --help
media-ctl [options]
-d, --device dev       Media device name (default: /dev/media0)
-e, --entity name      Print the device name associated with the given entity
-V, --set-v4l2 v4l2    Comma-separated list of formats to setup
  --get-v4l2 pad     Print the active format on a given pad
  --set-dv pad       Configure DV timings on a given pad
-h, --help           Show verbose help and exit
-i, --interactive      Modify links interactively
-l, --links links      Comma-separated list of link descriptors to setup
  --known-mbus-fmts   List known media bus formats and their numeric values
-p, --print-topology    Print the device topology. If an entity
                is specified through the -e option, print
                information for that entity only.
  --print-dot       Print the device topology as a dot graph
-r, --reset          Reset all links to inactive
-v, --verbose        Be verbose
.....................
```

5.4.1 Media-ctl: 拓扑结构(topology)

驱动如果有支持 Media Controller， 在 CIF 或 ISP 加载成功后会创建 media 设备，如/dev/media0。利用 media-ctl 可以打印出当前的 pipeline 的链接情况。

接下来几个小节介绍 CIF 和 ISP 拓扑结构的几种 Cases。

5.4.2 Case 1: CIF topology

图 2 CIF topology 是 px3se evb 板子上 cif 的拓扑结构。ADV7181 作为 sub-device Source 端链接到 CIF Sink 端。

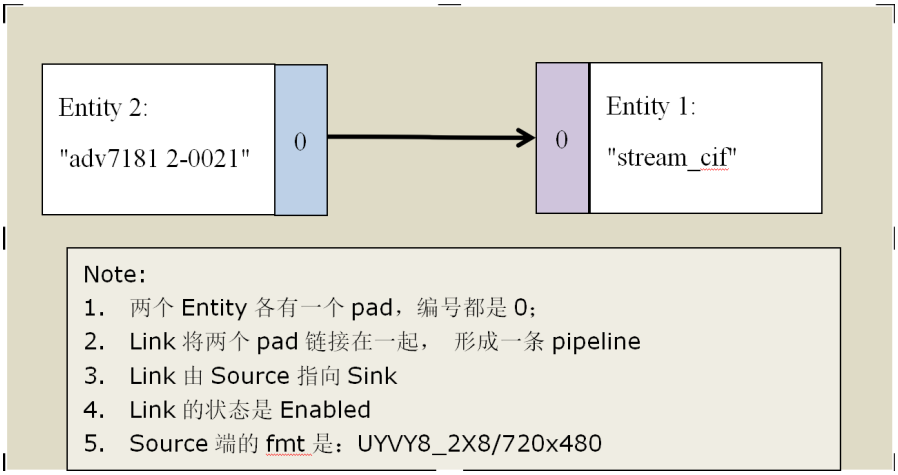

图 2 CIF topology

Topology 可通过 media-ctl 打印出来，如下(有删减)。

```
[root@px3se:/]# media-ctl -p /dev/media0
Media controller API version 0.1.0
------------------------
driver        rkcif
Device topology
- entity 1: stream_cif (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video0
        pad0: Sink
                <- "adv7181 2-0021":0 [ENABLED]
- entity 2: adv7181 2-0021 (1 pad, 1 link)
            type V4L2 subdev subtype Sensor flags 0
            device node name /dev/v4l-subdev0
        pad0: Source
                [fmt:UYVY8_2X8/720x480 field:none colorspace:smpte170m]
                -> "stream_cif":0 [ENABLED]
```

5.4.3 Case 2: ISP camera topology

以 RK3326 EVB 为例， Sensor Ov5695 通过 mipi 总线连接到 mipi dphy；ISP 采集到 mipi 数据后，由 isp entity 做相应的图像处理，如去马赛克等；mp、sp 可以配置各自输出帧数据的格式与分辨率。

通过 media-ctl 打印出来更多信息，比如当前 entity 的输入输出 format、size，是否具有 crop、resize 能力。 以下是 RK3326 EVB 的 media topology。

```
[root@rk3326_64:/]# media-ctl -d /dev/media1 -p
Media controller API version 0.1.0

Media device information
------------------------
driver       rkisp1
model        rkisp1
serial
bus info
hw revision    0x0
driver version  0.0.0

Device topology
- entity 1: rkisp1-isp-subdev (4 pads, 5 links)
          type V4L2 subdev subtype Unknown flags 0
          device node name /dev/v4l-subdev0
        pad0: Sink
             [fmt:SBGGR10_1X10/2592x1944 field:none
              crop.bounds:(0,0)/2592x1944
              crop:(0,0)/2592x1944]
             <- "rockchip-mipi-dphy-rx":1 [ENABLED]
        pad1: Sink
             <- "rkisp1-input-params":0 [ENABLED]
```

```
        pad2: Source
                [fmt:YUYV8_2X8/2592x1944 field:none
                 crop.bounds:(0,0)/2592x1944
                 crop:(0,0)/2592x1944]
                -> "rkisp1_selfpath":0 [ENABLED]
                -> "rkisp1_mainpath":0 [ENABLED]
        pad3: Source
                -> "rkisp1-statistics":0 [ENABLED]

- entity 2: rkisp1_mainpath (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video1
        pad0: Sink
                <- "rkisp1-isp-subdev":2 [ENABLED]

- entity 3: rkisp1_selfpath (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video2
        pad0: Sink
                <- "rkisp1-isp-subdev":2 [ENABLED]

- entity 4: rkisp1-statistics (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video3
        pad0: Sink
                <- "rkisp1-isp-subdev":3 [ENABLED]

- entity 5: rkisp1-input-params (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video4
```

```
        pad0: Source
            -> "rkisp1-isp-subdev":1 [ENABLED]

    - entity 6: rockchip-mipi-dphy-rx (2 pads, 2 links)
          type V4L2 subdev subtype Unknown flags 0
          device node name /dev/v4l-subdev1
        pad0: Sink
            [fmt:SBGGR10_1X10/2592x1944 field:none]
            <- "ov5695 2-0036":0 [ENABLED]
        pad1: Source
            [fmt:SBGGR10_1X10/2592x1944 field:none]
            -> "rkisp1-isp-subdev":0 [ENABLED]

    - 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]
```

5.4.4 Media-ctl: entity、pad 及 link

Entity 在 Media Controller 中，表示一个节点。它包含一个或多个的输入输出 pads。Link 表示一条链接，它连接多个不同的 pad。多个 Link 组成了一条完整的 pipeline。

Entity 的名称可以从拓扑结构中查看，比如以下都是 entity 的名称

- ov5695 2-0036
- rkisp1-isp-subdev
- rkisp1_mainpath

Pad 由数字表示，一个 Entity 中可以包含多个 pad，既可以有 Source， 也可以有 Sink。

Link 连接两个”entity”:pad，比如以下表示的链接，

- "rkisp1-isp-subdev":2->"rkisp1_mainpath":0

- "rkisp1-isp-subdev":3->"rkisp1-statistics":0

链接的状态可以是 Active， 也可以是 In-Active。

Media Controller 提供了灵活配置 pipeline 的功能， 在 CIF、ISP 驱动初始化过程中，会根据配置，将 Link 完整建立起来，如果有多个 Sensor 会激活其中的一个。

用户可以使用 media-ctl 命令修改 Link 的 Active 状态，也可以修改 Pad 的 format、size。

5.4.5 Media-ctl：In-/Active Link

如下错误：引用源未找到中，有两个 Sensor 接到同一个 mipi dphy，可以由 userspace 启用或关闭 Sensor。Sensor Ov5695 及 Ov2685 通过 mipi 总线连接到 mipi dphy，这个 Case 中两个 Sensor 同时只能有一个 Enabled。

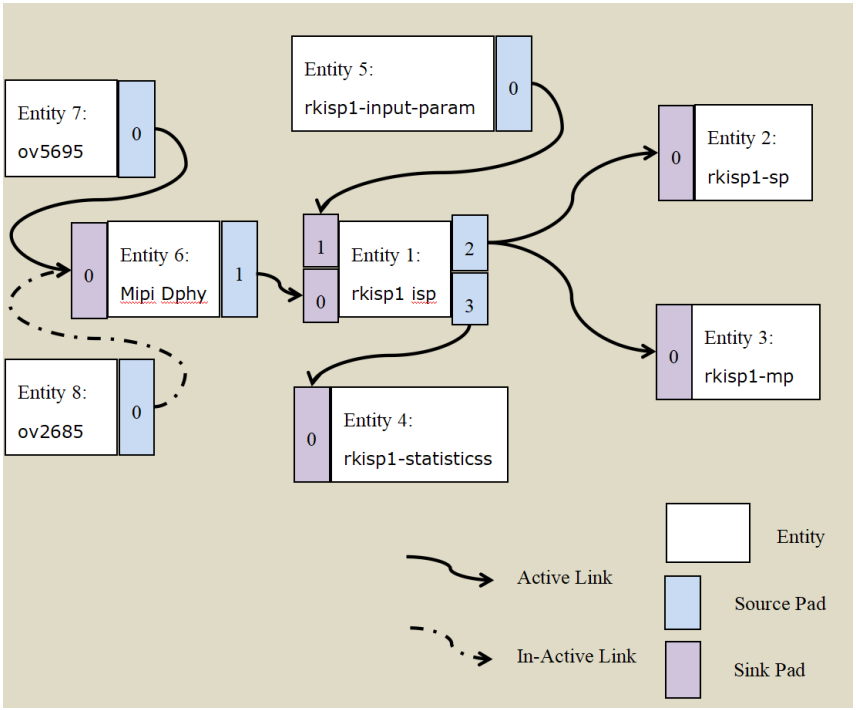

图 3 ISP 双 Camera 拓扑结构

错误：引用源未找到，有多个 Sensor 连接到同一个 Mipi D-Phy， 同时只能有一个是 Active 状态。以下示例设置 ov2659 为 Active。

```
#首先将 ov5695 设置为 In-Active
media-ctl -d /dev/media1 -l '"ov5695 2-0036":0->"rockchip-mipi-dphy-rx":0[0]'
#再将 ov2685 设置为 Active
media-ctl -d /dev/media1 -l '"ov2685 2-003c":0->"rockchip-mipi-dphy-rx":0[1]'
```

注意：

- 格式为：media-ctl -l '"entity name":pad->"entity name":pad[Status]'
- 整个 link 需要用单引号，因为有特殊字符如 > [ ]
- Entity name 需要用双引号，因为中间有空格
- Status 用 0 或 1 表示 Active 或 In-Active，需要用中括号

5.4.6 Media-ctl：修改 fmt/size

如错误：引用源未找到，以下示例如何修改 fmt/size。

示例一，修改 ov5695 输出的 size 为 640x480（前提是 ov5695 驱动中已经有支持 640x480 输出，否则设置不会成功），并设置整个 pipeline 的输入输出分辨率都为 640x480，同时 isp entity 的 format 为 YUYV。

设置 Sensor 输出 640x480

```
media-ctl -d /dev/media0 \
    --set-v4l2 '"ov5695 7-0036":0[fmt:SBGGR10_1X10/640x480]'
```

设置 isp 的接收（从 mipi dphy）格式及大小，格式要求要与 Sensor 输出相同

```
media-ctl -d /dev/media0 \
    --set-v4l2 '"rkisp1-isp-subdev":0[fmt:SBGGR10_1X10/640x480]'
```

isp 接收的大小可以 crop（裁剪），但这里还保持 640x480

```
media-ctl -d /dev/media0 \
    --set-v4l2 '"rkisp1-isp-subdev":0[crop:(0,0)/640x480]'
```

设置 isp 输出的，YUV 格式只能是 YUYV2X8

```
media-ctl -d /dev/media0 \
    --set-v4l2 '"rkisp1-isp-subdev":2[fmt:YUYV8_2X8/640x480]'
```

isp 输出也有 crop 功能，但也保持 640x480

```
media-ctl -d /dev/media0 \
    --set-v4l2 '"rkisp1-isp-subdev":2[crop:(0,0)/640x480]'
```

注意：

- 不同版本的 v4l-utils 其对应的 fmt 代码可能不同，最好用 media-ctl 查看

```
media-ctl --known-mbus-fmts
```

- 注意特殊字符，需要使用单引号或双引号
- 注意引号中不要少掉空格，也不要多出空格

- 请使用 media-ctl --help，查看更详细的使用帮助

5.5 v4l2-ctl 设置 fmt、controls 及抓帧

Media-ctl 工具的操作是通过/dev/medio0 等 media 设备，它所管理是 Media 的拓扑结构中各个节点的 format，大小，链接。V4l2-ctl 工具则是针对/dev/video0，/dev/video1 等 video 设备，它在 video 设备上进行 set_fmt，reqbuf，qbuf，dqbuf，stream_on，stream_off 等一系列操作。

本文中主要用 v4l2-ctl 进行采集帧数据，设置曝光、gain、VTS 等 v4l2_control。

首先还是要建议先查看 v4l2-ctl 的帮助文档。帮助文档内容比较多，分成很多个部分，我们比较关心，streaming，vidcap。

查看帮助文档梗概如下。

```
$ v4l2-ctl --help
```

查看完整的帮助文档如下，内容很多。

```
$ v4l2-ctl --help-all
```

查看与 streaming 相关的参数如下。

```
$ v4l2-ctl --help-streaming
```

查看与 vidcap 相关的参数如下。它主要包括 get-fmt、set-fmt 等。

```
$ v4l2-ctl --help-vidcap
```

5.5.1 设置 fmt 并抓帧

V4l2-ctl 的具体使用， 请参考 v4l-utils 工具及应用 。这里仅以 px3se-evb 板子为例。CIF 的 media 拓扑结构一般都很简单， 只有一个 sensor 和 cif 两个 entity。如果 sensor 的输出 format 和 size 不需要修改而使用默认值的话， 并不需要 media-ctl 配置即可直接使用 v4l2-ctl 抓取一帧。如下只需要指定 fmt，count 等相关参数即可。

```
v4l2-ctl -d /dev/video0 \
        --set-fmt-video=width=720,height=480,pixelformat=NV12 \
    --stream-mmap=3 \
    --stream-skip=3 \
    --stream-to=/tmp/cif.out \
    --stream-count=1 \
    --stream-poll
```

参数的说明：

- -d，指定操作对象为/dev/video0 设备。
- --set-fmt-video，指定了宽高及 pxielformat。
- NV12，即用 FourCC 表示的 pixelformat。FourCC 编码详见上文 FourCC。
- --stream-mmap，指定 buffer 的类型为 mmap，即由 kernel 分配的物理连续的或经过 iommu 映射的 buffer。
- --stream-skip，指定丢弃（不保存到文件）前 3 帧
- --stream-to，指定帧数据保存的文件路径
- --stream-count，指定抓取的帧数， 不包括--stream-skip 丢弃的数量
- --stream-poll，该选项指示 v4l2-ctl 采用异步 IO，即在 dqbuf 前先用 select 等等帧数据完成，从而保证 dqbuf 不阻塞。否则 dqbuf 将会阻塞直到有数据帧到来。

5.5.2 设置曝光、gain 等 control

如果 sensor 驱动有实现 v4l2 control，在采集图像前， 可以通过 v4l2-ctl 设置如曝光、gain 等 v4l2 control。

CIF 或 ISP 会继承 sub device 的 control，因此这里通过/dev/video3 可以看到 Sensor 的 v4l2 control。如下是 Dru 机子上查看到的 OV5695 的相关设置，包括 exposure，gain，blanking，test_pattern 等。

```
localhost /tmp # v4l2-ctl -d /dev/video3 -l
User Controls
   exposure (int)    : min=4 max=2020 step=1 default=1104 value=1104
   gain (int)    : min=0 max=16383 step=1 default=1024 value=1024
Image Source Controls
   vertical_blanking (int)    : min=40 max=31795 step=1 default=40 value=80
   horizontal_blanking (int)    : min=1664 max=1664 step=1 default=1664
analogue_gain (int)    : min=16 max=248 step=1 default=248 value=248
Image Processing Controls
   link_frequency (intmenu): min=0 max=0 default=0 value=0 flags=read-only
  pixel_rate (int64) : min=0 max=180000000 step=1 default=180000000 value=180000000
flags=read-only
```

```
test_pattern (menu)   : min=0 max=4 default=0 value=0
```

用 v4l2-ctl 可以修改这些 control。如修改 exposure 及 analogue_gain 如下。

```
v4l2-ctl -d /dev/video3 --set-ctrl 'exposure=1216,analogue_gain=10'
```

注意：

- 有特殊字符需要用单引号
- 如多个 Camera 连到同一个 ISP，只有查看、修改第一个 sensor 的 control。是 V4l2-ctl 存在 bug，还是因为驱动没有实现 G/S_INPUT

5.5.3 Ubuntu 上利用 mplayer 回放帧

在抓取到了帧数据保存到文件中后， 可以在 PC 机上回放图像。例如在 ubuntu 上可以用 mplayer 播放。

```
W=720; H=480; mplayer /tmp/cif.out -loop 0 -demuxer rawvideo -fps 25 \
    -rawvideo w=${W}:h=${H}:size=$((${W}*${H}*3/2)):format=nv12
```

关于 mplayer 更详细的使用，请参考 mplayer 的 manual[1]，或者网上教程。

5.5.4 设置 fmt 并抓 Raw Bayer 原始数据

rkisp 驱动会同步当前所连接 sensor 的 format/size 等信息，并设置各个 pipeline 的节点上，不需要再通过上层设置，如果需要设置自定义 format/size 到各个 pipeline 上，可参考以下示例。

示例，抓取 Sensor OV5695 输出的 Raw Bayer 原始数据。格式为 SBGGR10_1X10 大小为 2592x1944。

```
media-ctl -d /dev/media1 --set-v4l2 '"ov5695 2-0036":0[fmt:SBGGR10_1X10/2592x1944]'
media-ctl -d /dev/media1 --set-v4l2 '"rkisp1-isp-subdev":0[fmt:SBGGR10_1X10/2592x1944]'
media-ctl -d /dev/media1 --set-v4l2 '"rkisp1-isp-subdev":0[crop:(0,0)/2592x1944]'
media-ctl -d /dev/media1 --set-v4l2 '"rkisp1-isp-subdev":2[fmt:SBGGR10_1X10/2592x1944]'
media-ctl -d /dev/media1 --set-v4l2 '"rkisp1-isp-subdev":2[crop:(0,0)/2592x1944]'
v4l2-ctl -d /dev/video1 --set-ctrl 'exposure=1216,analogue_gain=10' \
 --set-selection=target=crop,top=0,left=0,width=2592,height=1944 \
 --set-fmt-video=width=2592,height=1944,pixelformat=BG10 \
```

1[] 如果 Ubuntu 上装有 mplayer， 用命令 man mplayer 可以查看 mplayer 的使用手册。

```
--stream-mmap=3 --stream-to=/tmp/mp.raw.out --stream-count=1 --stream-poll
```

注意：

- 第 4 行 media-ctl 设置了 isp-subdev 输出格式与 sensor 一致
- 第 3，第 5 行设置了 crop 与 Sensor 大小一致， 即不裁剪
- 第 6 行，如果图片太暗，可以调节曝光，gain 以增加亮度。可选，且 Sensor 驱动需要有实现该 v4l2 control
- 第 7 行，v4l2-ctl 设置了 selection 不裁剪，且输出 pixelformat FourCC 为 BG10
- 特别要注意的是， ISP 虽然不对 Raw 图处理，但它仍然会将 10bit 的数据低位补 0 成 16bit。不管 Sensor 输入的是 10bit，12bit，最终上层拿的都是 16bit 每像素。

5.5.5 Bayer Raw 图转成 PGM

在 Bayer Raw 文件头部加上 PGM 标识，即可其转成 Ubuntu[1]可直接打开查看的 pgm 图片。添加三行 PGM 头即可。用户常常会在添加 pgm header 时回车符错误，多一个空行导致无法打开 pgm 图片。以下命令可直接用于生成 pgm header，并将 raw 产追加到 pgm header 尾部。将 bayer raw 数据追加到该 raw.pgm 后面，如下。这样/tmp/raw.pgm 即可直接查看。注意文件追加需要使用两个’>’。

可以编写 raw2pgm.sh 脚本如下。

```
#!/bin/bash
cat > /tmp/raw.pgm << EOF
P5
2592 1944
65535
cat $1 >> /tmp/raw.pgm
```

其中，

- 行 1， P5 为固定标识符
- 行 2， 表示 Raw 图的分辨率，即长宽，中间用一个空格符分隔
- 行 3， 表示深度，65535 即 16bit。如果是 8bit，相应地改成 255

只要在脚本后面添加待转换的一帧 RAW 文件即可。

1[] Windows 下似乎需要安装其它工具才能找开 pgm 文件。也可以用 photoshop 打开 bayer raw 图，但同样需要设置宽高，bpp。

```
./raw2gpm.sh mp.raw
```

6 GStreamer 预览

6.1 gst-launch-1.0 命令

在 Rockchip 最新发布的 Linux SDK 下，GStreamer 的命令是一样的，不论是 CIF/ISP 还是 UVC 的 Camera，对于 V4L2 设备都可用 GStreamer 捕获 Camera 图像。对于没有 3A 效果的预览测试，可以使用如下命令利用 GStreamer 预览 Camera 的图像，前提要保证用 v4l2-ctl 可以抓图。对于 RKISP1 驱动层会根据所连接 sensor 的属性配置 pipeline，设置 isp-subdev 输入输出大小等操作，若需要自定义分辨率格式等属性，请阅读前面的小节。

6.1.1 gst-launch-1.0 命令预览

```
export XDG_RUNTIME_DIR=/tmp/.xdg
gst-launch-1.0 --gst-debug=3 v4l2src device=/dev/video1 ! videoconvert ! \
    video/x-raw,format=NV12,width=640,height=480 ! kmssink
```

关于 GStreamer 的使用请参考随 SDK 发布的 docs 文档。

--gst-debug=3：打印等级,数值越大等级越高打印信息越多

device=/dev/video1：指定打开的 camera 设备节点，默认 video0

videoconvert：将 src 数据格式转换为 sink 可以显示的数据格式

video/x-raw,format=NV12,width=640,height=480：sink 显示数据格式

kmssink：显示插件

6.1.2 gst-launch-1.0 命令保存文件

想保存本地文件，只需要更换 sink 插件，取 10 帧数据为例：

```
gst-launch-1.0 v4l2src  device=/dev/video1 num-buffers=10 ! \
video/x-raw,format=NV12,width=640,height=480, framerate=30/1 ! \
videoconvert !  filesink location=/tmp/test.yuv
```

6.2 GStreamer 应用编程

Rockchip 发布的 Linux SDK 都集成有 GStreamer，可以用来预览图像。本节介绍如何使用 GStreamer 编写一个简单的 camera 预览程序。

首先必须在主函数中调用 gst_init()来完成相应的初始化工作，以便将用户从命令行输入的参数传递给 GStreamer 函数库。一个典型的 GStreamer 应用程序的初始化如下所示：

```
#include <gst/gst.h>
int main(int argc, char *argv[]) {
        /* Initialize GStreamer */
        gst_init (&argc, &argv);
}
```

通过 **g_main_loop_new()**来创建 **GMainLoop** 对象。

```
/* create main loop, start to loop after running g_main_loop_run */
loop = g_main_loop_new(NULL, -1);
```

管道在 GStreamer 框架中是用来容纳和管理元件的，下面的代码将创建一条名为 uvc-camera 的新管道：

```
pipeline = gst_pipeline_new("uvc-camera");
```

创建数据源元件,使用 v4l2src，元件负责从 camera 中取数据

```
source = gst_element_factory_make("v4l2src", "camera-input");
```

设置 v4l2src 的一些属性

```
/* set source parameters */
g_object_set(G_OBJECT(source), "device", argv[1], NULL);


source_capsfilter = gst_element_factory_make("capsfilter", "source_capsfilter");
        source_caps = gst_caps_new_simple ("video/x-raw",
            "format", G_TYPE_STRING, "YUY2",
            "width", G_TYPE_INT, 640,
            "height", G_TYPE_INT, 480,
            NULL);
g_object_set(G_OBJECT(source_capsfilter),"caps", source_caps, NULL);
```

创建过滤器元件，接收器元件。

```
/* create converter element */
converter = gst_element_factory_make("videoconvert", "video-converter");
/* create sink element */
 sink = gst_element_factory_make("autovideosink", "camera-output");
```

加入一个消息处理函数 bus_call 来监视产生的消息。

```
/* Wait until error or EOS */
bus = gst_element_get_bus (pipeline);
gst_bus_add_watch(bus, bus_call, loop);
gst_object_unref(bus);
```

已经创建好的三个元件需要全部添加到管道中，并按顺序连接起来。

```
/* add elements to pipeline */
gst_bin_add_many(GST_BIN(pipeline), source, source_capsfilter, converter, sink, NULL);
/* connect elements sequentially */
gst_element_link_many(source, source_capsfilter, converter, sink, NULL);
```

所有准备工作都做好之后，就可以通过将管道的状态切换到 PLAYING 状态，来启动整个管道的数据处理流程。

```
gst_element_set_state(pipeline, GST_STATE_PLAYING);
```

进入主循环，它会一直阻塞在这里，直到让它退出为止。有事件时，它就处理事件，没事件时就睡眠。

```
/* start to loop */
g_main_loop_run(loop);
```

终止管道，释放资源。

```
/* quit loop and return */
g_print("Returned,stopping playback\n");
gst_element_set_state(pipeline, GST_STATE_NULL);
gst_object_unref(GST_OBJECT(pipeline));
```

可以直接用 PC 测试该程序，编译命令如下，开发板测试则使用交叉编译工具链编译。

```
gcc gst-video.c -o gst-video `pkg-config --cflags --libs gstreamer-1.0`
```

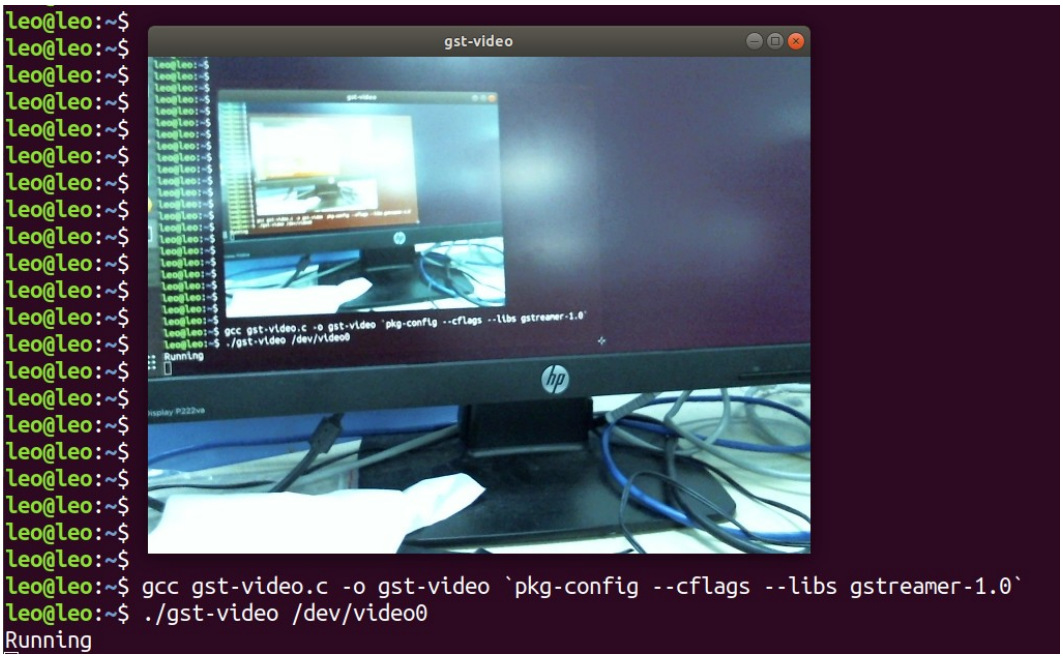

图 4 gst-video 应用预览

完整代码如下所示：

```
#include <gst/gst.h>
#include <glib.h>
#include <stdio.h>
#include <stdbool.h>

/* message processing */
static gboolean bus_call(GstBus * bus, GstMessage * msg, gpointer data)
{
  GMainLoop *loop = (GMainLoop *) data;

  switch (GST_MESSAGE_TYPE(msg))
  {

    case GST_MESSAGE_EOS:
      g_print("End of stream\n");
      g_main_loop_quit(loop);
      break;
```

```
        case GST_MESSAGE_ERROR:
          {
            gchar *debug;
            GError *error;
            gst_message_parse_error(msg, &error, &debug);
            g_free(debug);
            g_printerr("ERROR:%s\n", error->message);
            g_error_free(error);
            g_main_loop_quit(loop);
            break;
          }
        default:
          break;
      }

    return 1;
}

int main(int argc, char *argv[]) {

        GMainLoop *loop;
        GstElement *pipeline, *source, *converter, *sink, *vqueue;
        GstElement *source_capsfilter;
        GstCaps *source_caps;
        GstBus *bus;

        /* Initialize GStreamer */
        gst_init (&argc, &argv);
        if (argc < 2)
        {
          g_printerr("Usage:%s [camera device name] for camera capture. eg(./gst-video /dev/video0)\n",
```

```
argv[0]);
        return -1;
    }
    /* create main loop, start to loop after running g_main_loop_run */
    loop = g_main_loop_new(NULL, -1);
    /* create pipeline and element */
    pipeline = gst_pipeline_new("uvc-camera");
    source = gst_element_factory_make("v4l2src", "camera-input");
    /* set source parameters */
    g_object_set(G_OBJECT(source), "device", argv[1], NULL);

    source_capsfilter = gst_element_factory_make("capsfilter", "source_capsfilter");
    source_caps = gst_caps_new_simple ("video/x-raw",
       "format", G_TYPE_STRING, "YUY2",
       "width", G_TYPE_INT, 640,
       "height", G_TYPE_INT, 480,
       NULL);
    g_object_set(G_OBJECT(source_capsfilter),"caps", source_caps, NULL);

    /* create converter element */
    converter = gst_element_factory_make("videoconvert", "video-converter");
    /* create sink element */
    sink = gst_element_factory_make("autovideosink", "camera-output");

    if (!pipeline || !source || !source_capsfilter || !converter || !sink)
    {
        g_printerr("One element could not be created.Exiting.\n");
        return -1;
    }
    /* Wait until error or EOS */
    bus = gst_element_get_bus (pipeline);
```

```
    gst_bus_add_watch(bus, bus_call, loop);
    gst_object_unref(bus);
    /* add elements to pipeline */
    gst_bin_add_many(GST_BIN(pipeline), source, source_capsfilter, converter, sink, NULL);
    /* connect elements sequentially */
    gst_element_link_many(source, source_capsfilter, converter, sink, NULL);

    gst_element_set_state(pipeline, GST_STATE_PLAYING);
    g_print("Running\n");

    /* start to loop */
    g_main_loop_run(loop);

    /* quit loop and return */
    g_print("Returned,stopping playback\n");
    gst_element_set_state(pipeline, GST_STATE_NULL);
    gst_object_unref(GST_OBJECT(pipeline));
    return 0;
```

7 3A 功能集成

对于 Sensor 的输出是 Bayer Raw，经过 ISP 处理后转换成 YUV 格式的应用场景，后期需要使用 ISP 的 3A 功能 tuning IQ。IQ tuning 需要客户经由业务窗口向 Rockchip 提交申请，提供典型模组及样机由 Rockchip 调试 IQ。详情请向业务咨询。

这一章节，主要的 3A 库的编译，安装以及使用，在 Linux SDK 一同发布的 docs/Develop reference documents/CAMERA/《camera_engine_rkisp_user_manual_v1.0.pdf》文档中有详细的描述，请先阅读该文档，这里不再赘述。

最终，针对该 Sensor（包括镜头）会生成一份 xml 文件（假设以 OV5695.xml 命名）。同时还会提供四个 so 库，由 camera_engine_rkisp 工程生成， 位于 external 目录下[1]，Buildroot 会自动将四个库拷贝到相应目录下，Debian 若不存在 3A 库请自行安装。

- OV5695.xml：3A 的 tuning IQ 文件。请放到/etc/files/目录下，buildroot 中 etc/init.d/S50link_iq 脚本会在开机后，通过 ISP 连接的 sensor 的名字，匹配/etc/files/目录下同名的 xml，并创建链接文件 cam_iq.xml，对于 squash 只读文件系统，脚本无效。
- librkisp.so：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。存在开发板的/usr/lib/目录下。
- librkisp_aec.so：为 RK 实现的自动曝光库，实现为动态加载库，且有标准接口。存在开发板的/usr/lib/rkisp/ae/目录下。
- librkisp_af.so：为 RK 实现的自动对焦库，实现为动态加载库，且有标准接口。存在开发板的/usr/lib/rkisp/af/目录下。
- librkisp_awb.so：为 RK 实现的白平衡库，实现为动态加载库，且有标准接口。存在开发板的/usr/lib/rkisp/awb/目录下。

应用程序如何调用 3A 相关的函数，也请参考随 Linux SDK 一同发布的 docs/Develop reference documents/CAMERA/《camera_engine_rkisp_user_manual_v1.0.pdf》。简单的 demo，也可以参考应用程序开发及 demo。

1[] 如果用户的 Linux SDK，并没有这个 rkisp_demo，请及时更新或向 FAE 索要源码

7.1 应用程序开发及 demo

随 Linux SDK 发布的有一个 external/camera_engine_rkisp 仓库，该 camera_engine_rkisp/tests/rkisp_demo.c 简单地使用了 v4l2 接口配置设备，并简单调用了 3A 的接口。请直接参考源码即可。

应用程序开发，还可以直接参考 v4l2-utils 源码包(libv4l-1.14.1/utils/v4l2-ctl)，其提供的 so 加也可以直接使用。

或者更简单的是参考官网的 https://linuxtv.org/downloads/v4l-dvb-apis-new/uapi/v4l/v4l2.html。

注意：上述官网应用的 buff 类型仅仅支持 V4L2_BUF_TYPE_VIDEO_CAPTURE，而 RKisp 驱动层使用的 buff 类型是 V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE，此类型用于多平面存储格式的视频捕获设备，所以应用层的 buff 也要匹配此类型，rkisp_demo.c 以及 v4l2-ctl 均添加了此类型。

7.2 Gstreamer-1.0 3A 插件

Rockchip 已经将 3A 功能集成在插件中，上述《camera_engine_rkisp_user_manual_v1.0》文档详细说明了 3A 库以及包含的 gstreamer 插件（libgstrkisp.so）的使用。

Rockchip 也基于 gstreamer-1.14 编写了一套新的插件 rkv4l2src，工程位于 external/gst-plugins-rockchip，rkv4l2src 功能比较简单的 elements 插件，基于 v4l2 协议进行数据采集，并加入 camera_engine_rkisp 提供的 3A 功能，该插件可以在 Buildroot 以及 Debian（gstreamer 升级 1.14）下运行。

1、无论是 Buildroot 以及 Debian，要先确认 3A 库是否存在。对于 rkisp1 驱动来说，3A 库一共有 4 个，librkisp.so，librkisp_aec.so，librkisp_af.so，librkisp_awb.so。

2、Buildroot 以及 Debian，gstreamer 插件存放的路径有所区别，Buildroot 存放在/usr/lib/gstreamer-1.0/libgstrkv4l2.so，Debian 存放在/usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstrkv4l2.so，确定 libgstrkv4l2.so 存在后，通过 gst-inspect-1.0 rkv4l2src 可以查看插件详细信息

```
gst-inspect-1.0 rkv4l2src
```

```
[root@rk3326_64:/]# gst-inspect-1.0 rkv4l2src
Factory Details:
  Rank                     primary (256)
  Long-name                Rockchip Video (video4linux2) Source
  Klass                    Source/Video
  Description              Reads frames from a Rockchip Video4Linux2 device
  Author                   Leo Wen <leo.wen@rock-chips.com>

Plugin Details:
  Name                     rkv4l2
  Description              Rockchip elements for Video 4 Linux
  Filename                 /usr/lib/gstreamer-1.0/libgstrkv4l2.so
  Version                  1.1.0
  License                  GPL
  Source module            gst-plugins-rockchip
  Binary package           GStreamer Plug-ins source release
  Origin URL               Unknown package origin

GObject
 +----GInitiallyUnowned
       +----GstObject
             +----GstElement
                   +----GstBaseSrc
                         +----GstPushSrc
                               +----GstRKV4l2Src
```

图 6 rkv4l2src 插件信息

3、在保证用 v4l2src 可以预览后，可直接用 gst-launch-1.0 测试 3A 插件预览：

gst--launch-1.0 rkv4l2src device=/dev/video1 xml-path=/etc/cam_iq.xml ! video/x-raw,format=NV12,width=640,height=480, framerate=30/1 ! videoconvert ! kmssink

xml-path=/etc/cam_iq.xml：指定 3A 的 xml 文件，默认路径/etc/cam_iq.xml

7.3 Media 设备的开发 demo

7.4 Video 设备的开发 demo

7.5 V4l-subdev 设备的开发 demo

8 rk-isp10 驱动简介

Rk-isp10 没有使用 media-controller 及 async sub device，但支持 vb2 及 v4l2 接口。并且 rk-isp10 驱动集成并最简化了 sensor 的开发。关于 rk-isp10（又名 cif-isp10）驱动，请参考随 Linux SDK 一同发布的《CIF_ISP10_Driver_User_Manual_v1.0.pdf》及《RK_ISP10_Camera_User_Manual_v2.2.pdf》。

这里主要介绍几点 rkisp1 与 rk-isp10 区别。

首先，他们是同一硬件 IP 的不同驱动实现。

RKISP1 是基于 media controller，async sub device，vb2 及 v4l2 接口。RK-ISP10 则没有 media controller 和 async sub device。因此 RKISP1 可以认为是提供更为丰富自由的 pipeline 设置，而 RK-ISP10 则提供给用户最简单快速的使用。

RK-ISP10 将 CIF 与 ISP 合并成一个驱动。而本文介绍的 CIF 和 ISP 是相互独立的。

RK-ISP10 自定义了一套 sensor 驱动的框架，代码位于 drivers/media/i2c/soc_camera/rockchip/。而本文介绍的 CIF、ISP 则使用 drivers/media/i2c/目录下符合 async sub device 的驱动。

Rkisp1 驱动编码风格更简洁，代码行数仅为 rk-isp10 的一半。