Rockchip_Driver_Guide_ISP2x_CN

文件标识：RK-YH-GX-602

发布版本：V1.0.3

日期：2021-02-23

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文旨在描述RKISP（Rockchip Image Signal Processing）模块的作用，整体工作流程，及相关的API接口。主要给
驱动工程师调试Camera提供帮助。

产品版本``

芯片名称	内核版本
RV1126/RV1109	Linux 4.19

读者对象

本文档（本指南）主要适用于以下工程师：

驱动开发工程师

系统集成软件开发工程师

适用平台及系统

芯片名称	软件系统	支持情况
RV1126	Linux(Kernel-4.19)	Y
RV1109	Linux(Kernel-4.19)	Y
RK3566	Linux(Kernel-4.19)	Y
RK3568	Linux(Kernel-4.19)	Y

修订记录

版本号	作者	修改日期	修改说明
v0.1.0	蔡艺伟	2020-06-11	初始版本
v1.0.0	陈泽发	2020-10-30	新增focus、zoom、iris、ircut说明
v1.0.1	陈泽发	2021-01-04	修改格式错误
v1.0.2	蔡艺伟	2021-01-21	rv1109/rv1126内存优化指南
v1.0.3	黄江龙	2021-02-04	新增rkvicap驱动说明

目录

Rockchip_Driver_Guide_ISP2x_CN
- Camera 软件驱动目录说明
- ISP和VICAP的链接关系
- RKISP 驱动
  - 框架简要说明
  - ISP HDR 模式说明
- RKVICAP 驱动
  - 框架说明
- CIS(cmos image sensor)驱动
  - CIS 设备注册(DTS)
    - 单注册
      - MIPI接口
        - 链接ISP
        - 链接VICAP
      - LVDS接口
        - 链接VICAP
      - DVP接口
        - 链接VICAP
    - 多sensor 注册
  - CIS驱动说明
    - 数据类型简要说明
      - struct i2c_driver
      - struct v4l2_subdev_ops

struct v4l2_subdev_core_ops
struct v4l2_subdev_video_ops
struct v4l2_subdev_pad_ops
struct v4l2_ctrl_ops
struct xxxx_mode
struct v4l2_mbus_framefmt
struct rkmodule_base_inf
struct rkmodule_fac_inf
struct rkmodule_awb_inf
struct rkmodule_lsc_inf
struct rkmodule_af_inf
struct rkmodule_inf
struct rkmodule_awb_cfg
struct rkmodule_lsc_cfg
struct rkmodule_hdr_cfg
struct preisp_hdrae_exp_s
API简要说明
xxxx_set_fmt
xxxx_get_fmt
xxxx_enum_mbus_code
xxxx_enum_frame_sizes
xxxx_g_frame_interval
xxxx_s_stream
xxxx_runtime_resume
xxxx_runtime_suspend
xxxx_set_ctrl
xxx_enum_frame_interval
xxxx_g_mbus_config
xxxx_get_selection
驱动移植步骤
VCM驱动
VCM设备注册(DTS)
VCM驱动说明
数据类型简要说明
struct i2c_driver
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
API简要说明
xxxx_get_ctrl
xxxx_set_ctrl
xxxx_ioctl xxxx_compat_ioctl
驱动移植步骤
FlashLight驱动
FLASHLight设备注册(DTS)
FLASHLight驱动说明
数据类型简要说明
struct i2c_driver
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
API简要说明
xxxx_set_ctrl
xxxx_get_ctrl
xxxx_ioctl xxxx_compat_ioctl
驱动移植步骤
FOCUS ZOOM P-IRIS驱动
FOCUS ZOOM P-IRIS设备注册(DTS)
数据类型简要说明
struct platform_driver

struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
API简要说明
xxxx_set_ctrl
xxxx_get_ctrl
xxxx_ioctl xxxx_compat_ioctl
驱动移植步骤
DC-IRIS驱动
DC-IRIS设备注册(DTS)
数据类型简要说明
struct platform_driver
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
API简要说明
xxxx_set_ctrl
xxxx_ioctl xxxx_compat_ioctl
驱动移植步骤
RK-IRCUT驱动
RK-IRCUT设备注册(DTS)
数据类型简要说明
struct platform_driver
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
API简要说明
xxxx_set_ctrl
xxxx_ioctl xxxx_compat_ioctl
驱动移植步骤
media-ctl v4l2-ctl工具
rv1109/rv1126内存优化指南
FAQ
如何获取驱动版本号
如何判断RKISP驱动加载状态
如何抓取ispp输出的yuv数据
如何抓取Sensor输出的Bayer Raw数据
如何支持黑白摄像头
如何支持奇偶场合成
如何查看debug信息
附录A CIS驱动V4L2-controls列表
附录B MEDIA_BUS_FMT表
附录C CIS参考驱动列表
附录D VCM driver ic参考驱动列表
附录E Flash light driver ic参考驱动列表

Camera 软件驱动目录说明

Linux Kernel-4.19

|-- arch/arm/boot/dts DTS配置文件

|-- drivers/phy/rockchip

|-- phy-rockchip-mipi-rx.c mipi dphy驱动
|-- phy-rockchip-csi2-dphy-common.h
|-- phy-rockchip-csi2-dphy-hw.c
|-- phy-rockchip-csi2-dphy.c

|-- drivers/media

|-- platform/rockchip/cif RKCIF驱动

|-- platform/rockchip/isp RKISP驱动

|-- dev 包含 probe、异步注册、clock、pipeline、 iommu及media/v4l2 framework

|-- capture 包含 mp/sp/rawwr的配置及 vb2，帧中断处理

|-- dmarx 包含 rawrd的配置及 vb2，帧中断处理

|-- isp_params 3A相关参数设置

|-- isp_stats 3A相关统计

|-- isp_mipi_luma mipi数据亮度统计

|-- regs 寄存器相关的读写操作

|-- rkisp isp subdev和entity注册

|-- csi csi subdev和mipi配置

|-- bridge bridge subdev，isp和ispp交互桥梁

|-- platform/rockchip/ispp rkispp驱动

|-- dev 包含 probe、异步注册、clock、pipeline、 iommu及media/v4l2 framework

|-- stream 包含 4路video输出的配置及 vb2，帧中断处理

|-- rkispp ispp subdev和entity注册

|-- params TNR/NR/SHP/FEC/ORB参数设置

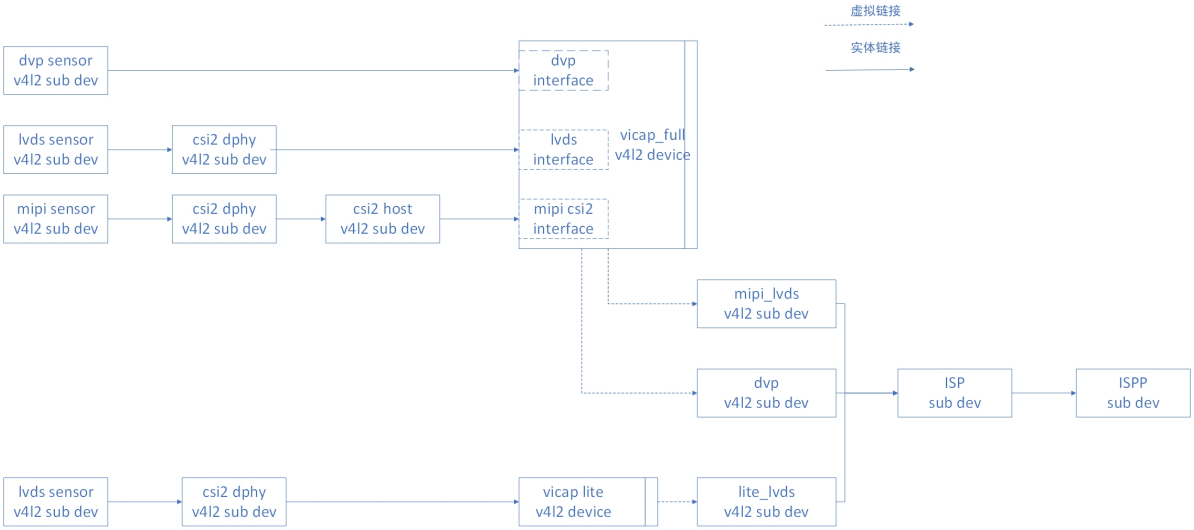
|-- stats ORB统计信息

|-- i2c

 |-- os04a10.c CIS(cmos image sensor)驱动

ISP和VICAP的链接关系

对于RV1126/RV1109及RK356X平台而言，VICAP和ISP是独立的两个图像处理IP, VICAP所采集图像若要通过ISP处理，在驱动层面需要生成VICAP对应接口的v4l2 sub device链接到ISP对应的节点，以提供参数给ISP驱动使用。ISP 驱动说明参见RKISP驱动，VICAP驱动说明参见RKVICAP驱动。具体的VICAP各接口与ISP链接的整体框图如下所示：

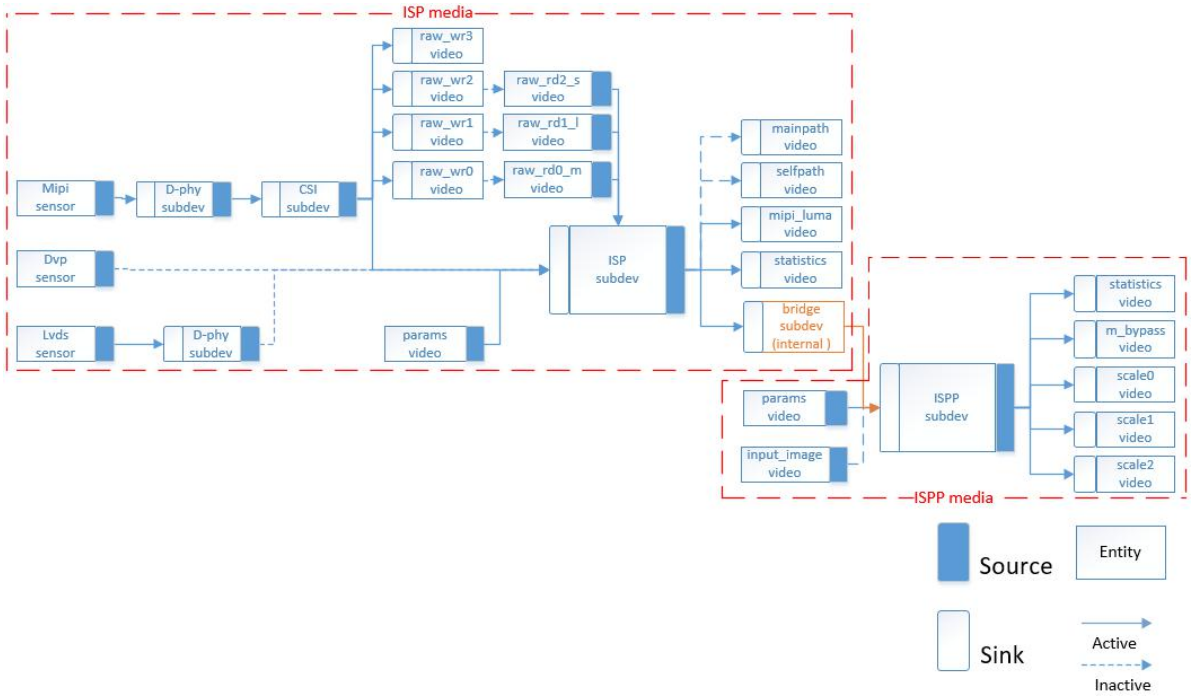


RKISP 驱动

框架简要说明

RKISP驱动主要是依据v4l2 / media framework实现硬件的配置、中断处理、控制 buffer轮转，以及控制subdevice(如 mipi dphy及sensor)的上下电等功能。

下面的框图描述了RKISP驱动的拓扑结构：

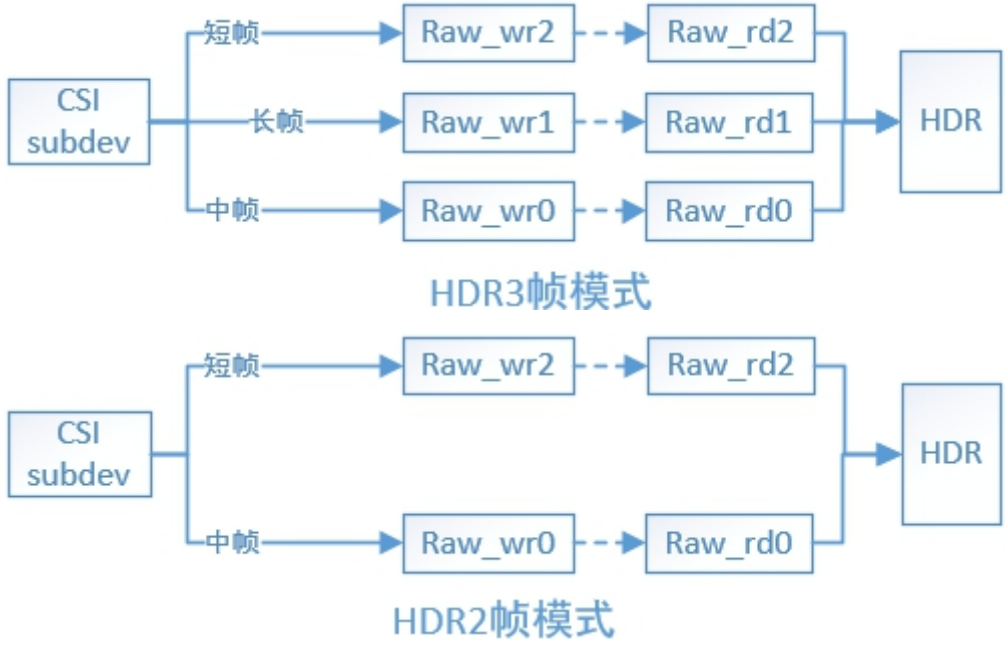


名称	类型	描述
rkisp_mainpath	v4l2_vdevcapture	Format: YUV, RAW Bayer; Support: Crop
rkisp_selfpath	v4l2_vdevcapture	Format: YUV, RGB; Support: Crop
rkisp-isp-subdev	v4l2_subdev	Internal isp blocks; Support: source/sink pad crop. The format on sink pad equal to sensor input format, the size equal to sensor input size. The format on source pad should be equal to vdev output format if output format is raw bayer, otherwise it should be YUYV2X8. The size should be equal/less than sink pad size.
rkisp-mipi-luma	v42_vdevcapture	Provice raw image luma
rkisp-statistics	v4l2_vdevcapture	Provide Image color Statistics information.
rkisp-input-params	v4l2_vdevoutput	Accept params for AWB, BLC..... Image enhancement blocks.
rkisp_rawrd0_m	v4l2_vdevoutput	Raw image read from ddr to isp,usually using for the hdr middle frame
rkisp_rawrd1_l	v4l2_vdevoutput	Raw image read from ddr to isp,usually using for the hdr long frame
rkisp_rawrd2_s	v4l2_vdevoutput	Raw image read from ddr to isp,usually using for the hdr short frame
rkisp-csi-subdev	v4l2_subdev	Mipi csi configure
rkisp_rawwr0	v4l2_vdevcapture	Raw image write to ddr from sensor,usually using for the hdr middle frame
rkisp_rawwr1	v4l2_vdevcapture	Raw image write to ddr from sensor,usually using for the hdr long frame
rkisp_rawwr2	v4l2_vdevcapture	Raw image write to ddr from sensor,usually using for the hdr short frame
rkisp_rawwr3	v4l2_vdevcapture	Raw image write to ddr from sensor
rockchip-mipi-dphy-rx	v4l2_subdev	MIPI-DPHY Configure.
rkisp-bridge-ispp	v4l2_subdev	Isp output yuv image to ispp
rkispp_input_image	v4l2_vdevoutput	Yuv image read from ddr to ispp
rkisp-isp-subdev	v4l2_subdev	The format and size on sink pad equal to isp outputThe support max size is 4416x3312, mix size is 66x258
rkispp_m_bypass	v4l2_vdev capture	Full resolution and yuv format

名称	类型	描述
rkispp_scale0	v4l2_vdev capture	Full or scale resolution and yuv formatScale range:[1 8] ratio, 3264 max width
rkispp_scale1	v4l2_vdev capture	Full or scale resolution and yuv formatScale range:[2 8] ratio, 1280 max width
rkispp_scale2	v4l2_vdev capture	Full or scale resolution and yuv formatScale range:[2 8] ratio, 1280 max width

ISP HDR 模式说明

RKISP2支持接收mipi sensor输出hdr 3帧或2帧模式，硬件通过3路或2路dmatx采集数据到ddr，再通过3路或2路dmarx读到isp，isp做3帧或2帧合成，驱动链路如下所示：



csi subdev通过get_fmt获取sensor驱动多个pad格式的输出信息，对应csi的source pad。

Mipi sensor驱动具体配置请参考驱动移植步骤。

名称	名称	描述
rkisp-isp-subdev	Sensor pad0	Isp采集Sensor vc0(默认) 宽高格式输出，常用线性模式
rkisp_rawwr0	Sensor pad1	Rawwr0采集sensor vcX宽高格式输出
rkisp_rawwr1	Sensor pad2	Rawwr1采集sensor vcX宽高格式输出
rkisp_rawwr2	Sensor pad3	Rawwr2采集sensor vcX宽高格式输出
rkisp_rawwr3	Sensor pad4	Rawwr3采集sensor vcX宽高格式输出

RKVICAP 驱动

框架说明

RKVICAP驱动主要是基于v4l2 / media框架实现硬件的配置、中断处理、控制 buffer轮转，以及控制subdevice(如 mipi dphy及sensor)的上下电等功能。

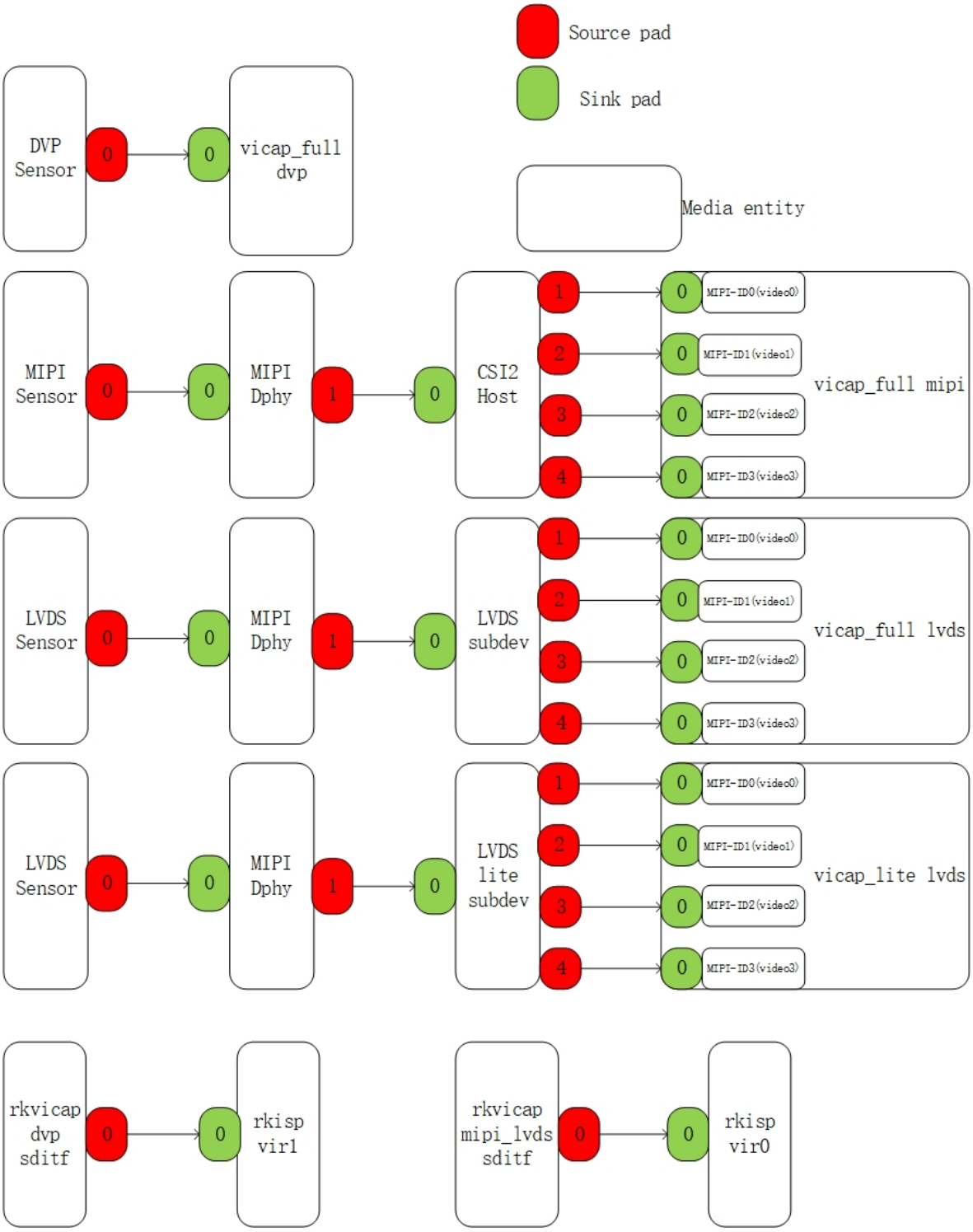
对于RV1126/RV1109而言，VICAP存在两个IP核，其中一个称之VICAP FULL,一个称之VICAP LITE。VICAP FULL拥有dvp/mipi/lvds三种接口，dvp可与mipi或者lvds接口同时工作，而mipi和lvds则不能同时工作，VICAP LITE 仅拥有lvds接口，可与VICAP FULL的接口同时工作；VICAP FULL dvp接口对应一个rkvicap_dvp节点，VICAP FULL mipi/lvds接口对应一个rkvicap_mipi_lvds节点，VICAP LITE 对应一个rkvicap_lite_mipi_lvds节点。各节点可独立采集。

对于RK356X芯片而言，VICAP只有单核，同时拥有dvp/mipi两种接口，dvp接口对应一个rkvicap_dvp节点，mipi接口对应一个rkvicap_mipi_lvds节点（与RV1126/RV1109的VICAP FULL同名），各节点可独立采集。

为了将VICAP采集数据信息同步给isp驱动，需要将VICAP驱动生成的逻辑sditf节点链接到isp所生成的虚拟节点设备。DVP接口对应rkvicap_dvp_sditf节点，VICAP FULL的mipi/lvds接口对应rkvicap_mipi_lvds_sditf节点，VICAP LITE对应rkvicap_lite_sditf。

具体各接口的dts链接方式参见[CIS 设备注册(DTS)][_CIS 设备注册(DTS)]

下图描述了RKVICAP驱动的设备拓扑结构：



CIS(cmos image sensor)驱动

CIS 设备注册(DTS)

单注册

MIPI接口

对于RV1126和RV1106平台而言，存在两个独立而完备的标准物理mipi csi2 dphy，对应于dts上的csi_dphy0和csi_dphy1（参见rv1126.dtsi），特性如下：

- data lane最大4 lanes；
- 最大速率2.5Gbps/lane；

对于RK356X平台而言，仅有一个标准物理mipi csi2 dphy，可以工作在两个模式：full mode 和split mode，拆分为csi2_dphy0/csi2_dphy1/csi2_dphy2三个逻辑dphy（参见rk3568.dtsi），特性如下：

Full mode

- 仅使用csi2_dphy0，csi2_dphy0与csi2_dphy1/csi2_dphy2互斥，不可同时使用；
- data lane最大4 lanes；
- 最大速率2.5Gbps/lane;

Split mode

- 仅使用csi2_dphy1和csi2_dphy2, 与csi2_dphy0互斥，不可同时使用；
- csi2_dphy1和csi2_dphy2可同时使用；
- csi2_dphy1和csi2_dphy2各自的data lane最大是2 lanes；
- csi2_dphy1对应物理dphy的lane0/lane1；
- csi2_dphy2对应物理dphy的lane2/lane3;
- 最大速率2.5Gbps/lane

具体dts用例，参见以下各示例。

链接ISP

RV1126/RV1106平台
下面以rv1126 isp和os04a10为例进行说明。

链接关系：sensor->csi_dphy->isp->ispp

arch/arm/boot/dts/rv1126-evb-v10.dtsi

配置要点

- data-lanes必须指明具体使用的lane数，否则无法识别为mipi 类型;

```
cam_ircut0: cam_ircut {
    status = "okay";
    compatible = "rockchip,ircut";
    ircut-open-gpios = <&gpio2 RK_PA7 GPIO_ACTIVE_HIGH>;
    ircut-close-gpios = <&gpio2 RK_PA6 GPIO_ACTIVE_HIGH>;
    rockchip,camera-module-index = <1>;
    rockchip,camera-module-facing = "front";
};

os04a10: os04a10@36 {
    compatible = "ovti,os04a10";// 需要与驱动中的匹配字符串一致
    reg = <0x36>;// sensor I2C设备地址，7位
    clocks = <&cru CLK_MIPICSI_OUT>;// sensor clickin配置
    clock-names = "xvclk";
    power-domains = <&power RV1126_PD_VI>;
    pinctrl-names = "rockchip,camera_default";
    pinctrl-0 = <&mipi_csi_clk0>;// pinctl设置
    //电源
    avdd-supply = <&vcc_avdd>;
    dovdd-supply = <&vcc_dovdd>;
    dvdd-supply = <&vcc_dvdd>;
    // power管脚分配及有效电平
    pwdn-gpios = <&gpio1 RK_PD4 GPIO_ACTIVE_HIGH>;
    // 模组编号，该编号不要重复
```

```
    rockchip,camera-module-index = <1>;
    // 模组朝向，有"back"和"front"
    rockchip,camera-module-facing = "front";
    // 模组名
    rockchip,camera-module-name = "CMK-OT1607-FV1";
    // lens名
    rockchip,camera-module-lens-name = "M12-4IR-4MP-F16";
    //ir cut设备
    ir-cut = <&cam_ircut0>;
    port {
        ucam_out0: endpoint {
            // mipi dphy端的port名
            remote-endpoint = <&mipi_in_ucam0>;
            // mipi lane数，1lane为 <1>, 4lane为 <1 2 3 4>
            data-lanes = <1 2 3 4>;
        };
    };
 };

&csi_dphy0 {
    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>;
                // sensor端的 port名
                remote-endpoint = <&ucam_out0>;
                // mipi lane数，1lane为 <1>, 4lane为 <1 2 3 4>
                data-lanes = <1 2 3 4>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;
            csidphy0_out: endpoint@0 {
                reg = <0>;
                // isp端的port名
                remote-endpoint = <&isp_in>;
            };
        };
    };
};

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";
    ports {
        #address-cells = <1>;
        #size-cells = <0>;
```

```
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;
            isp_in: endpoint@0 {
                reg = <0>;
                // mipi dphy端的 port名
                remote-endpoint = <&csidphy0_out>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;
            isp0_out: endpoint@1 {
                reg = <1>;
                // ispp 端port名，isp输出给ispp
                remote-endpoint = <&ispp0_in>;
            };
        };
    };
};

&rkispp {
    status = "okay";
};

&rkispp_vir0 {
    status = "okay";
    port {
        #address-cells = <1>;
        #size-cells = <0>;
        Ispp0_in: endpoint@0 {
            reg = <0>;
            // isp端port名，ispp输入
            remote-endpoint = <&isp0_out>;
        };
    };
};
```

RK356X平台

下面以rk3566 isp和gc8034 4lane为例进行说明：

链接关系：sensor->csi2_dphy0->isp

配置要点

- 需要配置data-lanes
- 需要使能csi2_dphy_hw节点

```
    /* full mode: lane0-3 */
    gc8034: gc8034@37 {
        // 需要与驱动中的匹配字符串一致
        compatible = "galaxycore,gc8034";
        status = "okay";
        // sensor I2C设备地址，7位
        reg = <0x37>;
        // sensor mclk源配置
```

```
        clocks = <&cru CLK_CIF_OUT>;
        clock-names = "xvclk";
        //sensor 相关电源域使能
        power-domains = <&power RK3568_PD_VI>;
        //sensor mclk pinctl设置
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clk>;
        // reset管脚分配及有效电平
        reset-gpios = <&gpio3 RK_PA6 GPIO_ACTIVE_LOW>;
        // powerdown管脚分配及有效电平
        pwdn-gpios = <&gpio4 RK_PB2 GPIO_ACTIVE_LOW>;
        // 模组编号，该编号不要重复
        rockchip,camera-module-index = <0>;
        // 模组朝向，有"back"和"front"
        rockchip,camera-module-facing = "back";
        // 模组名
        rockchip,camera-module-name = "RK-CMK-8M-2-v1";
        // lens名
        rockchip,camera-module-lens-name = "CK8401";
        port {
            gc8034_out: endpoint {
                // csi2 dphy端的port名
                remote-endpoint = <&dphy0_in>;
                // csi2 dphy lane数，1lane为 <1>, 4lane为 <1 2 3 4>
                data-lanes = <1 2 3 4>;
            };
        };
    };

    &csi2_dphy_hw {
       status = "okay";
    };

    &csi2_dphy0 {
        //csi2_dphy0不与csi2_dphy1/csi2_dphy2同时使用，互斥
        status = "okay";
        /*
         * dphy0 only used for full mode,
         * full mode and split mode are mutually exclusive
         */
        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            port@0 {
                reg = <0>;
                #address-cells = <1>;
                #size-cells = <0>;

                dphy0_in: endpoint@1 {
                    reg = <1>;
                    // sensor端的 port名
                    remote-endpoint = <&gc8034_out>;
                    // csi2 dphy lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor
端一致
                    data-lanes = <1 2 3 4>;
                };
            };
```

```
            port@1 {
                reg = <1>;
                #address-cells = <1>;
                #size-cells = <0>;

                dphy0_out: endpoint@1 {
                    reg = <1>;
                    // isp端的port名
                    remote-endpoint = <&isp0_in>;
                };
            };
        };
    };

&rkisp {
    status = "okay";
};

&rkisp_mmu {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

    port {
        #address-cells = <1>;
        #size-cells = <0>;

        isp0_in: endpoint@0 {
            reg = <0>;
            // csi2 dphy端的 port名
            remote-endpoint = <&dphy0_out>;
        };
    };
};
```

链接VICAP

RV1126/RV1109平台

以mipi os04a10 4lane链接vicap为例：

链接关系：sensor->csi dphy->mipi csi host->vicap

配置要点:

- data-lanes必须指明具体使用的lane数，否则无法识别为mipi 类型;
- dphy需要链接到csi host节点。

```
os04a10: os04a10@36 {
        // 需要与驱动中的匹配字符串一致
        compatible = "ovti,os04a10";
        // sensor I2C设备地址，7位
        reg = <0x36>;
        // sensor mclk源配置
        clocks = <&cru CLK_MIPICSI_OUT>;
```

```
        clock-names = "xvclk";
        //sensor 相关电源域使能
        power-domains = <&power RV1126_PD_VI>;
        avdd-supply = <&vcc_avdd>;
        dovdd-supply = <&vcc_dovdd>;
        dvdd-supply = <&vcc_dvdd>;
        //sensor mclk pinctl设置
        pinctrl-names = "rockchip,camera_default";
        pinctrl-0 = <&mipicsi_clk0>;
        // powerdown管脚分配及有效电平
        pwdn-gpios = <&gpio1 RK_PD4 GPIO_ACTIVE_HIGH>;
        // 模组编号，该编号不要重复
        rockchip,camera-module-index = <1>;
        // 模组朝向，有"back"和"front"
        rockchip,camera-module-facing = "front";
        // 模组名
        rockchip,camera-module-name = "CMK-OT1607-FV1";
        // lens名
        rockchip,camera-module-lens-name = "M12-40IRC-4MP-F16";
        // ircut名
        ir-cut = <&cam_ircut0>;
        port {
            ucam_out0: endpoint {
                // csi2 dphy端的port名
                remote-endpoint = <&mipi_in_ucam0>;
                // csi2 dphy lane数，1lane为 <1>, 4lane为 <1 2 3 4>
                data-lanes = <1 2 3 4>;
            };
        };
    };

&csi_dphy0 {
    //csi2_dphy0不与csi2_dphy1/csi2_dphy2同时使用，互斥
    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>;
                // sensor端的 port名
                remote-endpoint = <&ucam_out0>;
                // csi2 dphy lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor端一致
                data-lanes = <1 2 3 4>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            csidphy0_out: endpoint@0 {
                reg = <0>;
```

```
                // csi2 host端的port名
                remote-endpoint = <&mipi_csi2_input>;
            };
        };
    };
};

&mipi_csi2 {
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_input: endpoint@1 {
                reg = <1>;
                // csi2 dphy 端的port名
                remote-endpoint = <&csidphy0_out>;
                // csi2 host lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor端一致
                data-lanes = <1 2 3 4>;
            };
        };

        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_output: endpoint@0 {
                reg = <0>;
                // vicap端的port名
                remote-endpoint = <&cif_mipi_in>;
                // csi2 host lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor端一致
                data-lanes = <1 2 3 4>;
            };
        };
    };
};

&rkvicap_mipi_lvds {
    status = "okay";

    port {
        /* MIPI CSI-2 endpoint */
        cif_mipi_in: endpoint {
            // csi2 host端的port名
            remote-endpoint = <&mipi_csi2_output>;
            // vicap 端 lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor端一致
            data-lanes = <1 2 3 4>;
        };
    };
};
```

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

    port {
        /* sditf endpoint */
        mipi_lvds_sditf: endpoint {
            //isp 虚拟设备端port名
            remote-endpoint = <&isp_in>;
            //mipi csi2 dphy的lane数，与sensor一致
            data-lanes = <1 2 3 4>;
        };
    };
};

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

    ports {
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            isp_in: endpoint@0 {
                reg = <0>;
                //vicap sditf的端点名
                remote-endpoint = <&mipi_lvds_sditf>;
            };
        };
    };
};
```

RK356X平台

以gc5025 2lane链接rk3566 evb2 mipi csi2 dphy的lane2/lane3为例：

链接关系：sensor->csi2 dphy->mipi csi host->vicap

配置要点

- data-lanes必须指明具体使用的lane数，否则无法识别为mipi 类型;
- dphy需要链接到csi host节点;
- 需要使能csi2 dphy hw节点。

```
/* split mode: lane:2/3 */
gc5025: gc5025@37 {
        status = "okay";
        // 需要与驱动中的匹配字符串一致
        compatible = "galaxycore,gc5025";
        // sensor I2C设备地址，7位
        reg = <0x37>;
        // sensor mclk源配置
        clocks = <&pmucru CLK_WIFI>;
        clock-names = "xvclk";
```

```
        //sensor mclk pinctl设置
        pinctrl-names = "default";
        pinctrl-0 = <&refclk_pins>;
        // reset管脚分配及有效电平
        reset-gpios = <&gpio3 RK_PA5 GPIO_ACTIVE_LOW>;
        // powerdown管脚分配及有效电平
        pwdn-gpios = <&gpio3 RK_PB0 GPIO_ACTIVE_LOW>;
        //sensor 相关电源域使能
        power-domains = <&power RK3568_PD_VI>;
        /*power-gpios = <&gpio0 RK_PC1 GPIO_ACTIVE_HIGH>;*/
        // 模组编号，该编号不要重复
        rockchip,camera-module-index = <1>;
        // 模组朝向，有"back"和"front"
        rockchip,camera-module-facing = "front";
        // 模组名
        rockchip,camera-module-name = "TongJu";
        // lens名
        rockchip,camera-module-lens-name = "CHT842-MD";
        port {
            gc5025_out: endpoint {
                // csi2 dphy端的port名
                remote-endpoint = <&dphy2_in>;
                // csi2 dphy lane数，2lane为 <1 2>, 4lane为 <1 2 3 4>
                data-lanes = <1 2>;
            };
        };
};

 &csi2_dphy_hw {
        status = "okay";
 };

&csi2_dphy2 {
    //csi2_dphy2不与csi2_dphy0同时使用，互斥;可与csi2_dphy1并行使用
    status = "okay";

    /*
     * dphy2 only used for split mode,
     * can be used  concurrently  with dphy1
     * full mode and split mode are mutually exclusive
     */
    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            dphy2_in: endpoint@1 {
                reg = <1>;
                // sensor端的 port名
                remote-endpoint = <&gc5025_out>;
                // csi2 dphy lane数，2lane为 <1 2>, 4lane为 <1 2 3 4>,需与sensor端
一致
                data-lanes = <1 2>;
            };
```

```
        };

        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            dphy2_out: endpoint@1 {
                reg = <1>;
                // csi2 host端的port名
                remote-endpoint = <&mipi_csi2_input>;
            };
        };
    };
};

&mipi_csi2 {
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_input: endpoint@1 {
                reg = <1>;
                // csi2 dphy 端的port名
                remote-endpoint = <&dphy2_out>;
                // csi2 host lane数，2lane为 <1 2>, 4lane为 <1 2 3 4>,需与sensor端
一致
                data-lanes = <1 2>;
            };
        };

        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_output: endpoint@0 {
                reg = <0>;
                // vicap端的port名
                remote-endpoint = <&cif_mipi_in>;
                // csi2 host lane数，1lane为 <1>, 4lane为 <1 2 3 4>,需与sensor端一致
                data-lanes = <1 2>;
            };
        };
    };
};

&rkvicap_mipi_lvds {
    status = "okay";

    port {
```

```
        cif_mipi_in: endpoint {
            // csi2 host端的port名
            remote-endpoint = <&mipi_csi2_output>;
            // vicap 端 lane数，2lane为 <1 2>, 4lane为 <1 2 3 4>,需与sensor端一致
            data-lanes = <1 2>;
        };
    };
};

&rkvicap_mipi_lvds_sditf {
    status = "okay";

    port {
        /* MIPI CSI-2 endpoint */
        mipi_lvds_sditf: endpoint {
            //isp 虚拟设备端port名
            remote-endpoint = <&isp_in>;
            //mipi csi2 dphy的lane数，与sensor一致
            data-lanes = <1 2>;
        };
    };
};

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

    ports {
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            isp_in: endpoint@0 {
                reg = <0>;
                //vicap mipi sditf的端点名
                remote-endpoint = <&mipi_lvds_sditf>;
            };
        };
    };
};
```

LVDS接口

链接VICAP

RV1126/RV1109平台

以imx327 4lane为例, 链接关系如下：

链接关系：sensor->csi dphy->vicap

配置要点

- dphy不需要链接csi host节点，否则会导致收不到数据；

- data-lanes必须指明具体使用的lane数，否则会导致收不到数据；
- bus-type 必须配置为 3，否则无法识别为lvds接口，导致链路建立失败;

```
imx327: imx327@1a {
        // 需要与驱动中的匹配字符串一致
        compatible = "sony,imx327";
        // sensor I2C设备地址，7位
        reg = <0x1a>;
        // sensor mclk源配置
        clocks = <&cru CLK_MIPICSI_OUT>;
        clock-names = "xvclk";
        //sensor 相关电源域使能
        power-domains = <&power RV1126_PD_VI>;
        avdd-supply = <&vcc_avdd>;
        dovdd-supply = <&vcc_dovdd>;
        dvdd-supply = <&vcc_dvdd>;
        //sensor mclk pinctl设置
        pinctrl-names = "default";
        pinctrl-0 = <&mipicsi_clk0>;
        // powerdown管脚分配及有效电平
        pwdn-gpios = <&gpio3 RK_PA6 GPIO_ACTIVE_HIGH>;
        // reset管脚分配及有效电平
        reset-gpios = <&gpio1 RK_PD5 GPIO_ACTIVE_HIGH>;
        // 模组编号，该编号不要重复
        rockchip,camera-module-index = <1>;
        // 模组朝向，有"back"和"front"
        rockchip,camera-module-facing = "front";
        // 模组名
        rockchip,camera-module-name = "CMK-OT1607-FV1";
        // lens名
        rockchip,camera-module-lens-name = "M12-4IR-4MP-F16";
        // ircut名
        ir-cut = <&cam_ircut0>;
        port {
            ucam_out0: endpoint {
                // csi2 dphy端的port名
                remote-endpoint = <&mipi_in_ucam0>;
                //csi2 dphy lvds lane数，1lane为 <1>, 4lane为 <4>，必须指定
                data-lanes = <4>;
                //lvds接口的类型，必须指定
                bus-type = <3>;
            };
        };
};

&csi_dphy0 {
    //csi2_dphy0不与csi2_dphy1/csi2_dphy2同时使用，互斥
    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>;
                // sensor端的 port名
                remote-endpoint = <&ucam_out0>;
                //csi2 dphy lvds lane数，1lane为 <1>, 4lane为 <4>，必须指定
                data-lanes = <4>;
                //lvds接口的类型，必须指定
                bus-type = <3>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            csidphy0_out: endpoint@0 {
                reg = <0>;
                // vicap lite端的port名
                remote-endpoint = <&cif_lite_lvds_in>;
                //csi2 dphy lvds lane数，1lane为 <1>, 4lane为 <4>，必须指定
                data-lanes = <4>;
                //lvds接口的类型，必须指定
                bus-type = <3>;
            };
        };
    };
};

&rkvicap_lite_mipi_lvds {
    status = "okay";

    port {
        /* lvds endpoint */
        cif_lite_lvds_in: endpoint {
            // csi2 dphy端的port名
            remote-endpoint = <&csidphy0_out>;
            //csi2 dphy lvds lane数，1lane为 <1>, 4lane为 <4>，必须指定
            data-lanes = <4>;
            //lvds接口的类型，必须指定
            bus-type = <3>;
        };
    };
};

&rkvicap_lite_sditf {
    status = "okay";

    port {
        /* lvds endpoint */
        lite_sditf: endpoint {
            //isp 虚拟设备端port名
            remote-endpoint = <&isp_in>;
            //csi2 dphy的lane数，与sensor一致
            data-lanes = <4>;
        };
    };
};
```

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

&rkisp_vir0 {
    status = "okay";

    ports {
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            isp_in: endpoint@0 {
                reg = <0>;
                //lite vicap lvds sditf的端点名
                remote-endpoint = <&lite_sditf>;
            };
        };
    };
};
```

DVP接口

链接VICAP

在RV1126/RV1106/RK356X平台上，dvp各相关接口的dts配置是一样的。

BT601

以ar0230 bt601为例，链接关系如下：

链接关系：sensor->vicap

配置要点

- hsync-active/vsync-active必须配置，用于v4l2框架异步注册识别BT601接口,若不配置会识别为BT656接口；
- pclk-sample/bus-width可选；
- 必须在sensor驱动的g_mbus_config接口中，通过flag指明当前sensor的hsync-acitve/vsync-active/pclk-ative的有效极性，否则会导致无法收到数据；
- pinctrl需要引用对，以对bt601相关gpio做相应iomux，否则会导致无法收到数据；

g_mbus_config接口示例代码如下：

```
static int ar0230_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *config)
{
    config->type = V4L2_MBUS_PARALLEL;
    config->flags = V4L2_MBUS_HSYNC_ACTIVE_HIGH |
            V4L2_MBUS_VSYNC_ACTIVE_HIGH |
            V4L2_MBUS_PCLK_SAMPLE_FALLING;
    return 0;
}
```

dts配置示例如下：

```
ar0230: ar0230@10 {
```

```
        // 需要与驱动中的匹配字符串一致
        compatible = "aptina,ar0230";
        // sensor I2C设备地址，7位
        reg = <0x10>;
        // sensor mclk源配置
        clocks = <&cru CLK_CIF_OUT>;
        clock-names = "xvclk";
        //sensor 相关电源域使能
        avdd-supply = <&vcc_avdd>;
        dovdd-supply = <&vcc_dovdd>;
        dvdd-supply = <&vcc_dvdd>;
        power-domains = <&power RV1126_PD_VI>;
        // powerdown管脚分配及有效电平
        pwdn-gpios = <&gpio2 RK_PA6 GPIO_ACTIVE_HIGH>;
        /*reset-gpios = <&gpio2 RK_PC5 GPIO_ACTIVE_HIGH>;*/
        //配置dvp相关数据管脚和时钟管脚
        pinctrl-names = "default";
        pinctrl-0 = <&cifm0_dvp_ctl>;
        // 模组编号，该编号不要重复
        rockchip,camera-module-index = <0>;
        // 模组朝向，有"back"和"front"
        rockchip,camera-module-facing = "back";
        // 模组名
        rockchip,camera-module-name = "CMK-OT0836-PT2";
        // lens名
        rockchip,camera-module-lens-name = "YT-2929";
        port {
            cam_para_out1: endpoint {
                remote-endpoint = <&cif_para_in>;
            };
        };
};

&rkvicap_dvp {
    status = "okay";

    port {
        /* Parallel bus endpoint */
        cif_para_in: endpoint {
            //sensor端endpoint名
            remote-endpoint = <&cam_para_out1>;
            //sensor端相关配置参数
            bus-width = <12>;
            hsync-active = <1>;
            vsync-active = <1>;
            pclk-sample = <0>;
        };
    };
};

&rkvicap_dvp_sditf {
    status = "okay";

    port {
        /* parallel endpoint */
        dvp_sditf: endpoint {
            //isp 虚拟设备端port名
            remote-endpoint = <&isp_in>;
```

```
            //sensor端相关配置参数
            bus-width = <12>;
            hsync-active = <1>;
            vsync-active = <1>;
            pclk-sample = <0>;
        };
    };
};

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

    ports {
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            isp_in: endpoint@0 {
                reg = <0>;
                //dvp sditf的端点名
                remote-endpoint = <&dvp_sditf>;
            };
        };
    };
};
```

BT656/BT1120

BT656/BT1120的dts用法一致。

以ava fpga bt1120为例，链接关系如下：

链接关系： sensor->vicap

配置要点

- hsync-active/vsync-active不要配置，否则v4l2框架异步注册时会识别为BT601；
- pclk-sample/bus-width可选；
- 必须在sensor驱动的g_mbus_config接口中，通过flag指明当前sensor的pclk-ative的有效极性，否则会导致无法收到数据；
- 必须实现v4l2_subdev_video_ops中的querystd接口，指明当前接口为ATSC接口,否则会导致无法收到数据；
- pinctrl需要引用对，以对bt656/bt1120相关gpio做相应iomux，否则会导致无法收到数据。

g_mbus_config接口示例代码如下：

```
static int avafpga_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *config)
{
    config->type = V4L2_MBUS_BT656;
    config->flags = V4L2_MBUS_PCLK_SAMPLE_RISING;

    return 0;
}
```

querystd接口示例如下：

```
static int avafpga_querystd(struct v4l2_subdev *sd, v4l2_std_id *std)
{
    *std = V4L2_STD_ATSC;

    return 0;
}
```

dts配置示例如下：

```
avafpga: avafpga@70 {
    // 需要与驱动中的匹配字符串一致
    compatible = "ava,fpga";
    // sensor I2C设备地址，7位
    reg = <0x10>;
    // sensor mclk源配置
    clocks = <&cru CLK_CIF_OUT>;
    clock-names = "xvclk";
    //sensor 相关电源域使能
    avdd-supply = <&vcc_avdd>;
    dovdd-supply = <&vcc_dovdd>;
    dvdd-supply = <&vcc_dvdd>;
    // powerdown管脚分配及有效电平
    power-domains = <&power RV1126_PD_VI>;
    pwdn-gpios = <&gpio2 RK_PA6 GPIO_ACTIVE_HIGH>;
    /*reset-gpios = <&gpio2 RK_PC5 GPIO_ACTIVE_HIGH>;*/
    //配置dvp相关数据管脚和时钟管脚
    pinctrl-names = "default";
    pinctrl-0 = <&cifm0_dvp_ctl>;
    // 模组编号，该编号不要重复
    rockchip,camera-module-index = <0>;
    // 模组朝向，有"back"和"front"
    rockchip,camera-module-facing = "back";
    // 模组名
    rockchip,camera-module-name = "CMK-OT0836-PT2";
    // lens名
    rockchip,camera-module-lens-name = "YT-2929";
    port {
        cam_para_out2: endpoint {
            remote-endpoint = <&cif_para_in>;
        };
    };
};

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

```
    port {
        /* Parallel bus endpoint */
        cif_para_in: endpoint {
            //sensor端endpoint名
            remote-endpoint = <&cam_para_out2>;
            //sensor端相关配置参数，可选
            bus-width = <16>;
            pclk-sample = <1>;
        };
    };
};

&rkvicap_dvp_sditf {
    status = "okay";

    port {
        /* parallel endpoint */
        dvp_sditf: endpoint {
            //isp 虚拟设备端port名
            remote-endpoint = <&isp_in>;
            bus-width = <16>;
            pclk-sample = <1>;
        };
    };
};

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

    ports {
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            isp_in: endpoint@0 {
                reg = <0>;
                //dvp sditf的端点名
                remote-endpoint = <&dvp_sditf>;
            };
        };
    };
};
```

多sensor 注册

单个硬件isp/ispp通过虚拟多个设备，分时复用处理多路sensor数据。
链接关系，isp0->ispp0和isp1->ispp1是固定配置rv1126.dtsi
mipi进isp或cif进isp可选。

例如：
sensor0->csi_dphy0->csi2->vicap->isp0->ispp0
sensor1->csi_dphy1->isp1->ispp1

实例参考arch/arm/boot/dts/rv1109-evb-ddr3-v12-facial-gate.dts
gc2053->csi_dphy0->csi2->cif->isp1->ispp1
ov2718->csi_dphy1->isp0->ispp0
对于不同分辨率如下配置很重要

```
&rkispp {
   status = "okay";
   /* the max input w h and fps of mulit sensor */
   max-input = <2688 1520 30>;//取不同sensor的最大宽和高及帧率
};
```

CIS驱动说明

Camera Sensor采用I2C与主控进行交互，目前sensor driver按照I2C设备驱动方式实现，sensor driver同时采用v4l2 subdev的方式实现与host driver之间的交互。

数据类型简要说明

struct i2c_driver

[说明]

定义i2c 设备驱动信息

[定义]

```
struct i2c_driver {
    ……
    /* Standard driver model interfaces */
    int (*probe)(struct i2c_client *, const struct i2c_device_id *);
    int (*remove)(struct i2c_client *);
    ……
    struct device_driver driver;
    const struct i2c_device_id *id_table;
    ……
};
```

[关键成员]

成员名称	描述
@driver	Device driver model driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	List of I2C devices supported by this driver如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
#if IS_ENABLED(CONFIG_OF)
static const struct of_device_id os04a10_of_match[] = {
    { .compatible = "ovti,os04a10" },
    {},
};
MODULE_DEVICE_TABLE(of, os04a10_of_match);
#endif

static const struct i2c_device_id os04a10_match_id[] = {
    { "ovti,os04a10", 0 },
    { },
};

static struct i2c_driver os04a10_i2c_driver = {
    .driver = {
        .name = OS04A10_NAME,
        .pm = &os04a10_pm_ops,
        .of_match_table = of_match_ptr(os04a10_of_match),
    },
    .probe      = &os04a10_probe,
    .remove     = &os04a10_remove,
    .id_table   = os04a10_match_id,
};

static int __init sensor_mod_init(void)
{
    return i2c_add_driver(&os04a10_i2c_driver);
}

static void __exit sensor_mod_exit(void)
{
    i2c_del_driver(&os04a10_i2c_driver);
}

device_initcall_sync(sensor_mod_init);
module_exit(sensor_mod_exit);
```

struct v4l2_subdev_ops

[说明]

Define ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_ops {
    const struct v4l2_subdev_core_ops   *core;
    ……
    const struct v4l2_subdev_video_ops  *video;
    ……
    const struct v4l2_subdev_pad_ops    *pad;
};
```

[关键成员]

成员名称	描述
.core	Define core ops callbacks for subdevs
.video	Callbacks used when v4l device was opened in video mode.
.pad	v4l2-subdev pad level operations

[示例]

```
static const struct v4l2_subdev_ops os04a10_subdev_ops = {
    .core   = &os04a10_core_ops,
    .video  = &os04a10_video_ops,
    .pad    = &os04a10_pad_ops,
};
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ……
    int (*s_power)(struct v4l2_subdev *sd, int on);
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
        unsigned long arg);
#endif
    ……
};
```

[关键成员]

成员名称	描述
.s_power	puts subdevice in power saving mode (on == 0) or normal operation mode (on == 1).
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops os04a10_core_ops = {
    .s_power = os04a10_s_power,
    .ioctl = os04a10_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl32 = os04a10_compat_ioctl32,
#endif
};
```

目前使用了如下的私有ioctl实现模组信息的查询和OTP信息的查询设置。

私有ioctl	描述
RKMODULE_GET_MODULE_INFO	获取模组信息，详细参考struct rkmodule_inf;
RKMODULE_AWB_CFG	开关sensor对awb的补偿功能;若模组没有烧录golden awb值，可以在此设置;详细参考struct rkmodule_awb_cfg;
RKMODULE_LSC_CFG	开关sensor对lsc的补偿功能;详细参考struct rkmodule_lsc_cfg;
PREISP_CMD_SET_HDRAE_EXP	Hdr曝光设置详细参考struct preisp_hdrae_exp_s
RKMODULE_SET_HDR_CFG	设置Hdr模式，可实现normal和hdr模式切换，需要驱动适配normal和hdr 2组配置信息详细参考struct rkmodule_hdr_cfg
RKMODULE_GET_HDR_CFG	获取当前hdr模式详细参考struct rkmodule_hdr_cfg
RKMODULE_SET_CONVERSION_GAIN	设置线性模式的conversion gain，如imx347、os04a10 sensor带有conversion gain的功能，如sensor不支持conversion gain，可不实现

struct v4l2_subdev_video_ops

[说明]

Callbacks used when v4l device was opened in video mode.

[定义]

```
struct v4l2_subdev_video_ops {
    ……
    int (*s_stream)(struct v4l2_subdev *sd, int enable);
    ……
    int (*g_frame_interval)(struct v4l2_subdev *sd,
                struct v4l2_subdev_frame_interval *interval);
    int (*g_mbus_config)(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *cfg);
 ……
};
```

[关键成员]

成员名称	描述
.g_frame_interval	callback for VIDIOC_SUBDEV_G_FRAME_INTERVAL ioctl handler code
.s_stream	used to notify the driver that a video stream will start or has stopped
.g_mbus_config	get supported mediabus configurations

[示例]

```
static const struct v4l2_subdev_video_ops os04a10_video_ops = {
    .s_stream = os04a10_s_stream,
    .g_frame_interval = os04a10_g_frame_interval,
    .g_mbus_config = os04a10_g_mbus_config,
};
```

struct v4l2_subdev_pad_ops

[说明]

v4l2-subdev pad level operations

[定义]

```
struct v4l2_subdev_pad_ops {
    ......
    int (*enum_mbus_code)(struct v4l2_subdev *sd,
                struct v4l2_subdev_pad_config *cfg,
                struct v4l2_subdev_mbus_code_enum *code);
    int (*enum_frame_size)(struct v4l2_subdev *sd,
                 struct v4l2_subdev_pad_config *cfg,
                 struct v4l2_subdev_frame_size_enum *fse);
    int (*get_fmt)(struct v4l2_subdev *sd,
                 struct v4l2_subdev_pad_config *cfg,
                 struct v4l2_subdev_format *format);
    int (*set_fmt)(struct v4l2_subdev *sd,
                 struct v4l2_subdev_pad_config *cfg,
                 struct v4l2_subdev_format *format);
    int (*enum_frame_interval)(struct v4l2_subdev *sd,
                 struct v4l2_subdev_pad_config *cfg,
                 struct v4l2_subdev_frame_interval_enum *fie);
    int (*get_selection)(struct v4l2_subdev *sd,
                 struct v4l2_subdev_pad_config *cfg,
                 struct v4l2_subdev_selection *sel);
    ......
};
```

[关键成员]

成员名称	描述
. enum_mbus_code	callback for VIDIOC_SUBDEV_ENUM_MBUS_CODE ioctl handler code.
. enum_frame_size	callback for VIDIOC_SUBDEV_ENUM_FRAME_SIZE ioctl handler code.
.s_fmt	callback for VIDIOC_SUBDEV_S_FMT ioctl handler code.
.g_fmt	callback for VIDIOC_SUBDEV_G_FMT ioctl handler code
.enum_frame_interval	callback for VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL() ioctl handler code.
.get_selection	callback for VIDIOC_SUBDEV_G_SELECTION() ioctl handler code.

[示例]

```
static const struct v4l2_subdev_pad_ops os04a10_pad_ops = {
    .enum_mbus_code = os04a10_enum_mbus_code,
    .enum_frame_size = os04a10_enum_frame_sizes,
    .enum_frame_interval = os04a10_enum_frame_interval,
    .get_fmt = os04a10_get_fmt,
    .set_fmt = os04a10_set_fmt,
};
```

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.s_ctrl	actually set the new control value.

[示例]

```
static const struct v4l2_ctrl_ops os04a10_ctrl_ops = {
    .s_ctrl = os04a10_set_ctrl,
};
```

RKISP驱动要求使用框架提供的user controls功能， cameras sensor驱动必须实现如下control功能，参考CIS驱动V4L2-controls列表1

struct xxxx_mode

[说明]

Sensor能支持各个模式的信息。

这个结构体在sensor驱动中常常可以见到，虽然它不是v4l2标准要求的。

[定义]

```
struct xxxx_mode {
    u32 bus_fmt;
    u32 width;
    u32 height;
    struct v4l2_fract max_fps;
    u32 hts_def;
    u32 vts_def;
    u32 exp_def;
    const struct regval *reg_list;
    u32 hdr_mode;
    u32 vc[PAD_MAX];
};
```

[关键成员]

成员名称	描述
.bus_fmt	Sensor输出格式，参考MEDIA_BUS_FMT 表
.width	有效图像宽度，需要和sensor当前配置的width输出一致
.height	有效图像高度，需要和sensor当前配置的height输出一致
.max_fps	图像FPS，denominator/numerator为fps
hts_def	默认HTS，为有效图像宽度 + HBLANK
vts_def	默认VTS，为有效图像高度 + VBLANK
exp_def	默认曝光时间
*reg_list	寄存器列表
.hdr_mode	Sensor工作模式，支持线性模式，两帧合成HDR,三帧合成HDR
.vc[PAD_MAX]	配置MIPI VC通道

[示例]

```
 enum os04a10_max_pad {
    PAD0, /* link to isp */
    PAD1, /* link to csi rawwr0 | hdr x2:L x3:M */
    PAD2, /* link to csi rawwr1 | hdr    x3:L */
    PAD3, /* link to csi rawwr2 | hdr x2:M x3:S */
    PAD_MAX,
};

static const struct os04a10_mode supported_modes[] = {
    {
```

```
        .bus_fmt = MEDIA_BUS_FMT_SBGGR12_1X12,
        .width = 2688,
        .height = 1520,
        .max_fps = {
            .numerator = 10000,
            .denominator = 300372,
        },
        .exp_def = 0x0240,
        .hts_def = 0x05c4 * 2,
        .vts_def = 0x0984,
        .reg_list = os04a10_linear12bit_2688x1520_regs,
        .hdr_mode = NO_HDR,
        .vc[PAD0] = V4L2_MBUS_CSI2_CHANNEL_0,
    }, {
        .bus_fmt = MEDIA_BUS_FMT_SBGGR12_1X12,
        .width = 2688,
        .height = 1520,
        .max_fps = {
            .numerator = 10000,
            .denominator = 225000,
        },
        .exp_def = 0x0240,
        .hts_def = 0x05c4 * 2,
        .vts_def = 0x0658,
        .reg_list = os04a10_hdr12bit_2688x1520_regs,
        .hdr_mode = HDR_X2,
        .vc[PAD0] = V4L2_MBUS_CSI2_CHANNEL_1,
        .vc[PAD1] = V4L2_MBUS_CSI2_CHANNEL_0,//L->csi wr0
        .vc[PAD2] = V4L2_MBUS_CSI2_CHANNEL_1,
        .vc[PAD3] = V4L2_MBUS_CSI2_CHANNEL_1,//M->csi wr2
    },
};
```

struct v4l2_mbus_framefmt

[说明]

frame format on the media bus

[定义]

```
struct v4l2_mbus_framefmt {
    __u32           width;
    __u32           height;
    __u32           code;
    __u32           field;
    __u32           colorspace;
    __u16           ycbcr_enc;
    __u16           quantization;
    __u16           xfer_func;
    __u16           reserved[11];
};
```

[关键成员]

成员名称	描述
width	Frame width
height	Frame height
code	参考MEDIA_BUS_FMT 表
field	V4L2_FIELD_NONE：帧输出方式V4L2_FIELD_INTERLACED：场输出方式

[示例]

struct rkmodule_base_inf

[说明]

模组基本信息，上层用此信息和IQ进行匹配

[定义]

```
struct rkmodule_base_inf {
    char sensor[RKMODULE_NAME_LEN];
    char module[RKMODULE_NAME_LEN];
    char lens[RKMODULE_NAME_LEN];
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
sensor	sensor名，从sensor驱动中获取
module	模组名，从DTS配置中获取，以模组资料为准
lens	镜头名，从DTS配置中获取，以模组资料为准

[示例]

struct rkmodule_fac_inf

[说明]

模组OTP 工厂信息

[定义]

```
struct rkmodule_fac_inf {
    __u32 flag;
    char module[RKMODULE_NAME_LEN];
    char lens[RKMODULE_NAME_LEN];
    __u32 year;
    __u32 month;
    __u32 day;
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
flag	该组信息是否有效的标识
module	模组名，从OTP中获取编号，由编号得到模组名
lens	镜头名，从OTP中获取编号，由编号得到镜头名
year	生产年份,如12代表2012年
month	生产月份
day	生产日期

[示例]

struct rkmodule_awb_inf

[说明]

模组OTP awb测定信息

[定义]

```
struct rkmodule_awb_inf {
    __u32 flag;
    __u32 r_value;
    __u32 b_value;
    __u32 gr_value;
    __u32 gb_value;
    __u32 golden_r_value;
    __u32 golden_b_value;
    __u32 golden_gr_value;
    __u32 golden_gb_value;
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
flag	该组信息是否有效的标识
r_value	当前模组的AWB R测定信息
b_value	当前模组的AWB B测定信息
gr_value	当前模组的AWB GR测定信息
gb_value	当前模组的AWB GB测定信息
golden_r_value	典型模组的AWB R测定信息，如没有烧录，设为0
golden_b_value	典型模组的AWB B测定信息，如没有烧录，设为0
golden_gr_value	典型模组的AWB GR测定信息，如没有烧录，设为0
golden_gb_value	典型模组的AWB GB测定信息，如没有烧录，设为0

[示例]

struct rkmodule_lsc_inf

[说明]

模组OTP lsc测定信息

[定义]

```
struct rkmodule_lsc_inf {
    __u32 flag;
    __u16 lsc_w;
    __u16 lsc_h;
    __u16 decimal_bits;
    __u16 lsc_r[RKMODULE_LSCDATA_LEN];
    __u16 lsc_b[RKMODULE_LSCDATA_LEN];
    __u16 lsc_gr[RKMODULE_LSCDATA_LEN];
    __u16 lsc_gb[RKMODULE_LSCDATA_LEN];
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
flag	该组信息是否有效的标识
lsc_w	lsc表实际宽度
lsc_h	lsc表实际高度
decimal_bits	lsc 测定信息的小数位数，无法获取的话，设为0
lsc_r	lsc r测定信息
lsc_b	lsc b测定信息
lsc_gr	lsc gr测定信息
lsc_gb	lsc gb测定信息

[示例]

struct rkmodule_af_inf

[说明]

模组OTP af测定信息

[定义]

```
struct rkmodule_af_inf {
    __u32 flag; // 该组信息是否有效的标识
    __u32 vcm_start; // vcm启动电流
    __u32 vcm_end; // vcm终止电流
    __u32 vcm_dir; // vcm测定方向
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
flag	该组信息是否有效的标识
vcm_start	vcm启动电流
vcm_end	vcm终止电流
vcm_dir	vcm测定方向

[示例]

struct rkmodule_inf

[说明]

模组信息

[定义]

```
struct rkmodule_inf {
    struct rkmodule_base_inf base;
    struct rkmodule_fac_inf fac;
    struct rkmodule_awb_inf awb;
    struct rkmodule_lsc_inf lsc;
    struct rkmodule_af_inf af;
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
base	模组基本信息
fac	模组OTP 工厂信息
awb	模组OTP awb测定信息
lsc	模组OTP lsc测定信息
af	模组OTP af测定信息

[示例]

struct rkmodule_awb_cfg

[说明]

模组OTP awb配置信息

[定义]

```
struct rkmodule_awb_cfg {
    __u32 enable;
    __u32 golden_r_value;
    __u32 golden_b_value;
    __u32 golden_gr_value;
    __u32 golden_gb_value;
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
enable	标识awb校正是否启用
golden_r_value	典型模组的AWB R测定信息
golden_b_value	典型模组的AWB B测定信息
golden_gr_value	典型模组的AWB GR测定信息
golden_gb_value	典型模组的AWB GB测定信息

[示例]

struct rkmodule_lsc_cfg

[说明]

模组OTP lsc配置信息

[定义]

```
struct rkmodule_lsc_cfg {
    __u32 enable;
} __attribute__ ((packed));
```

[关键成员]

成员名称	描述
enable	标识lsc校正是否启用

[示例]

struct rkmodule_hdr_cfg

[说明]

hdr配置信息

[定义]

```
struct rkmodule_hdr_cfg {
    __u32 hdr_mode;
    struct rkmodule_hdr_esp esp;
} __attribute__ ((packed));
struct rkmodule_hdr_esp {
```

```
    enum hdr_esp_mode mode;
    union {
        struct {
            __u32 padnum;
            __u32 padpix;
        } lcnt;
        struct {
            __u32 efpix;
            __u32 obpix;
        } idcd;
    } val;
};
```

[关键成员]

成员名称	描述

hdr_mode	NO_HDR=0 //normal模式
	HDR_X2=5 //hdr 2帧模式
	HDR_X3=6 //hdr 3帧模式

struct rkmodule_hdr_esp	hdr especial mode

enum hdr_esp_mode	HDR_NORMAL_VC=0 //Normal virtual channel mode
	HDR_LINE_CNT=1 //Line counter mode (AR0239)
	HDR_ID_CODE=2 //Identification code mode(IMX327)

[示例]

struct preisp_hdrae_exp_s

[说明]

HDR曝光参数

[定义]

```
struct preisp_hdrae_exp_s {
    unsigned int long_exp_reg;
    unsigned int long_gain_reg;
    unsigned int middle_exp_reg;
    unsigned int middle_gain_reg;
    unsigned int short_exp_reg;
    unsigned int short_gain_reg;
    unsigned int long_exp_val;
    unsigned int long_gain_val;
    unsigned int middle_exp_val;
    unsigned int middle_gain_val;
    unsigned int short_exp_val;
    unsigned int short_gain_val;
    unsigned char long_cg_mode;
    unsigned char middle_cg_mode;
    unsigned char short_cg_mode;
};
```

[关键成员]

成员名称	描述
long_exp_reg	长帧曝光寄存器值
long_gain_reg	长帧增益寄存器值
middle_exp_reg	中帧曝光寄存器值
middle_gain_reg;	中帧增益寄存器值
short_exp_reg	短帧曝光寄存器值
short_gain_reg	短帧增益寄存器值
long_cg_mode	长帧conversion gain，0 LCG，1 HCG
middle_cg_mode	中帧conversion gain，0 LCG，1 HCG
short_cg_mode	短帧conversion gain，0 LCG，1 HCG

[说明]

preisp_hdrae_exp_s结构体内只需关注[关键成员]描述的几个参数，曝光、增益值转换成寄存器的公式在iq xml，具体如何转换请查阅iq xml格式说明，conversion gain需要Sensor本身支持这个功能，不支持的话，不需要关注conversion 参数，**HDR2X时，应将传下来的中帧、短帧参数设置进sensor的输出两帧对应的曝光参数寄存器。**

[示例]

API简要说明

xxxx_set_fmt

[描述]

设置sensor输出格式。

[语法]

```
static int xxxx_set_fmt(struct v4l2_subdev *sd,
                        struct v4l2_subdev_pad_config *cfg,
                        struct v4l2_subdev_format *fmt)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
*cfg	subdev pad information结构体指针	输入
*fmt	Pad-level media bus format结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_get_fmt

[描述]

获取sensor输出格式。

[语法]

```
static int xxxx_get_fmt(struct v4l2_subdev *sd,
                        struct v4l2_subdev_pad_config *cfg,
                        struct v4l2_subdev_format *fmt)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
*cfg	subdev pad information结构体指针	输入
*fmt	Pad-level media bus format结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

参考MEDIA_BUS_FMT 表

xxxx_enum_mbus_code

[描述]

枚举sensor输出bus format。

[语法]

```
static int xxxx_enum_mbus_code(struct v4l2_subdev *sd,
                               struct v4l2_subdev_pad_config *cfg,
                               struct v4l2_subdev_mbus_code_enum *code)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
*cfg	subdev pad information结构体指针	输入
*code	media bus format enumeration结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

下表总结了各种图像类型对应的format，参考MEDIA_BUS_FMT 表

xxxx_enum_frame_sizes

[描述]

枚举sensor输出大小。

[语法]

```
static int xxxx_enum_frame_sizes(struct v4l2_subdev *sd,
                                 struct v4l2_subdev_pad_config *cfg,
                                 struct v4l2_subdev_frame_size_enum *fse)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
*cfg	subdev pad information结构体指针	输入
*fse	media bus frame size结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_g_frame_interval

[描述]

获取sensor输出fps。

[语法]

```
static int xxxx_g_frame_interval(struct v4l2_subdev *sd,
                                 struct v4l2_subdev_frame_interval *fi)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
*fi	pad-level frame rate结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_s_stream

[描述]

设置stream输入输出。

[语法]

```
static int xxxx_s_stream(struct v4l2_subdev *sd, int on)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
on	1: 启动stream输出; 0: 停止stream输出	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_runtime_resume

[描述]

sensor上电时的回调函数。

[语法]

```
static int xxxx_runtime_resume(struct device *dev)
```

[参数]

参数名称	描述	输入输出
*dev	device结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_runtime_suspend

[描述]

sensor下电时的回调函数。

[语法]

```
static int xxxx_runtime_suspend(struct device *dev)
```

[参数]

参数名称	描述	输入输出
*dev	device结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_set_ctrl

[描述]

设置各个control的值。

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2_ctrl结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxx_enum_frame_interval

[描述]

枚举sensor支持的帧间隔参数。

[语法]

```
static int xxxx_enum_frame_interval(struct v4l2_subdev *sd,
                    struct v4l2_subdev_pad_config *cfg,
                    struct v4l2_subdev_frame_interval_enum *fie)
```

[参数]

参数名称	描述	输入输出
*sd	子设备实例	输入
*cfg	pad配置参数	输入
*fie	帧间隔参数	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_g_mbus_config

[描述]

获取支持的总线配置，比如使用mipi时，当Sensor支持多种MIPI传输模式时，可以根据Sensor当前使用的MIPI模式上传参数。

[语法]

```
static int xxxx_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *config)
```

[参数]

参数名称	描述	输入输出
*config	总线配置参数	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_get_selection

[描述]

配置裁剪参数，isp输入的宽度要求16对齐，高度8对齐，对于sensor输出的分辨率不符合对齐或sensor输出分辨率不是标准分辨率，可实现这个函数对输入isp的分辨率做裁剪。

[语法]

```
static int xxxx_get_selection(struct v4l2_subdev *sd,
                struct v4l2_subdev_pad_config *cfg,
                struct v4l2_subdev_selection *sel)
```

[参数]

参数名称	描述	输入输出
*sd	子设备实例	输入
*cfg	pad配置参数	输入
*sel	裁剪参数	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

1.实现标准I2C子设备驱动部分.

1.1 根据**struct i2c_driver**说明实现以下成员:

struct driver.name

struct driver.pm

struct driver. of_match_table

probe函数

remove函数

1.2 probe函数实现细节描述:
1). CIS 设备资源的获取,主要是解析DTS文件中定义资源, 参考Camera设备注册(DTS);

1.1) RK私有资源定义,命名方式如下rockchip,camera-module-xxx, 该部分资源会由驱动上传给用户态的camera_engine来决定IQ效果参数的匹配;

1.2) CIS设备资源定义,RK相关参考驱动一般包含以下几项:

成员名称	描述
CIS设备工作参考时钟	采用外部独立晶振方案无需获取,RK参考设计一般采用AP输出时钟，该方案需要获取，一般名称为xvclk
CIS设备控制GPIO	例如：Resst引脚,Powerdown引脚
CIS设备控制电源	根据实际硬件设计,获取匹配的软件电源控制资源,例如gpio,regulator

1.3) CIS设备ID号检查, 通过以上步骤获取必要资源后,建议驱动读取设备ID号以便检查硬件的准确性,当然该步骤非必要步骤.

1.4) CIS v4l2设备以及media实体的初始化;

v4l2子设备：v4l2_i2c_subdev_init，RK CIS驱动要求subdev拥有自己的设备节点供用户态rk_aiq访问，通过该设备节点实现曝光控制；

media实体：media_entity_init

2. 参考**struct v4l2_subdev_ops**说明实现v4l2子设备驱动，主要实现以下3个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_subdev_video_ops
struct v4l2_subdev_pad_ops
```

2.1 参考**struct v4l2_subdev_core_ops**说明实现其回调函数，主要实现以下回调：

.s_power.ioctl

.compat_ioctl32

ioctl主要实现的RK私有控制命令，涉及：

成员名称	描述
RKMODULE_GET_MODULE_INFO	DTS文件定义的模组信息(模组名称等)，通过该命令上传camera_engine
RKMODULE_AWB_CFG	模组OTP信息使能情况下，camera_engine通过该命令传递典型模组AWB标定值，CIS驱动负责与当前模组AWB标定值比较后，生成R/B Gain值设置到CIS MWB模块中；
RKMODULE_LSC_CFG	模组OTP信息使能情况下，camera_engine通过该命令控制LSC标定值生效使能；
PREISP_CMD_SET_HDRAE_EXP	HDR曝光设置详细参考struct preisp_hdrae_exp_s
RKMODULE_SET_HDR_CFG	设置HDR模式，可实现normal和hdr切换，需要驱动适配hdr和normal 2组配置信息详细参考struct rkmodule_hdr_cfg
RKMODULE_GET_HDR_CFG	获取当前HDR模式详细参考struct rkmodule_hdr_cfg
RKMODULE_SET_CONVERSION_GAIN	设置线性模式的conversion gain，如imx347、os04a10 sensor带有conversion gain的功能，高转换的conversion gain可以在低照度下获得更好的信噪比，如sensor不支持conversion gain，可不实现

2.2 参考**struct v4l2_subdev_video_ops**说明实现其回调函数，主要实现以下回调函数：

成员名称	描述
.s_stream	开关数据流的函数，对于mipi clk是continuous的模式，必须在这个回调函数内开启数据流，若提前开数据流，会识别不到MIPI LP状态
.g_frame_interval	获取帧间隔参数（帧率）
.g_mbus_config	获取总线配置，对于MIPI接口，sensor驱动内若支持不同lane数配置或者支持HDR,通过这个接口返回当前sensor工作模式下的MIPI配置

2.3 参考**struct v4l2_subdev_pad_ops**说明实现其回调函数，主要实现以下回调函数：

成员名称	描述
.enum_mbus_code	枚举当前CIS驱动支持数据格式
.enum_frame_size	枚举当前CIS驱动支持分辨率
.get_fmt	RKISP driver通过该回调获取CIS输出的数据格式，务必实现；针对Bayer raw sensor、SOC yuv sensor、BW raw sensor输出的数据类型定义参考MEDIA_BUS_FMT 表针对field 输出方式的支持，参考struct v4l2_mbus_framefmt定义；
.set_fmt	设置CIS驱动输出数据格式以及分辨率，务必实现
.enum_frame_interval	枚举sensor支持的帧间隔，包含分辨率
.get_selection	配置裁剪参数，isp输入的宽度要求16对齐，高度8对齐

2.4 参考**struct v4l2_ctrl_ops**说明实现，主要实现以下回调

成员名称	描述
.s_ctrl	RKISP driver、camera_engine通过设置不同的命令来实现CIS 曝光控制；

参考CIS驱动V4L2-controls列表1实现各控制ID，其中以下ID属于信息获取类，这部分实现按照standard integer menu controls方式实现；

成员名称	描述
V4L2_CID_LINK_FREQ	参考CIS驱动V4L2-controls列表1中标准定义，目前RKISP driver根据该命令获取MIPI总线频率；
V4L2_CID_PIXEL_RATE	针对MIPI总线：pixel_rate = link_freq * 2 * nr_of_lanes / bits_per_sample
V4L2_CID_HBLANK	参考CIS驱动V4L2-controls列表1中标准定义
V4L2_CID_VBLANK	参考CIS驱动V4L2-controls列表1中标准定义

RK camera_engine会通过以上命令获取必要信息来计算曝光，其中涉及的公式如下：

公式
line_time = HTS / PIXEL_RATE;
PIXEL_RATE = HTS * VTS * FPS
HTS = sensor_width_out + HBLANK;
VTS = sensor_height_out + VBLANK;

其中以下ID属于控制类，RK camera_engine通过该类命令控制CIS

成员名称	描述
V4L2_CID_VBLANK	调整VBLANK，进而调整frame rate、Exposure time max；
V4L2_CID_EXPOSURE	设置曝光时间，单位：曝光行数
V4L2_CID_ANALOGUE_GAIN	设置曝光增益，实际为total gain = analog gain*digital gain; 单位：增益寄存器值

3. CIS 驱动不涉及硬件数据接口信息定义，CIS设备与AP的接口连接关系由DTS设备节点的Port来体现其连接关系，参考 CIS 设备注册(DTS)中关于Port信息的描述。

4. CIS 参考驱动列表

VCM驱动

VCM设备注册(DTS)

RK VCM驱动私有参数说明：

名称	描述
启动电流	VCM 刚好能够推动模组镜头从模组镜头可移动行程最近端(模组远焦)移动，此时VCM driver ic的输出电流值定义为启动电流
额定电流	VCM刚好推动模组镜头至模组镜头可移动行程的最远端(模组近焦)，此时VCM driver ic的输出电流值定义为额定电流
VCM电流输出模式	VCM移动过程中会产生振荡,VCM driver ic电流输出变化需要考虑vcm的振荡周期，以便最大程度减小振荡，输出模式决定了输出电流改变至目标值的时间；

```
vm149c: vm149c@0c { // vcm驱动配置，支持AF时需要有这个设置
    compatible = "silicon touch,vm149c";
    status = "okay";
    reg = <0x0c>;
    rockchip,vcm-start-current = <0>; // 马达的启动电流
    rockchip,vcm-rated-current = <100>; // 马达的额定电流
    rockchip,vcm-step-mode = <4>; // 马达驱动ic的电流输出模式
    rockchip,camera-module-index = <0>; // 模组编号
    rockchip,camera-module-facing = "back"; // 模组朝向，有"back"和"front"
};

ov13850: ov13850@10 {
    ......
    lens-focus = <&vm149c>; // vcm驱动设置，支持AF时需要有这个设置
    ......
};
```

VCM驱动说明

数据类型简要说明

struct i2c_driver

[说明]

定义i2c 设备驱动信息

[定义]

```
struct i2c_driver {
    ……
    /* Standard driver model interfaces */
    int (*probe)(struct i2c_client *, const struct i2c_device_id *);
    int (*remove)(struct i2c_client *);
    ……
    struct device_driver driver;
    const struct i2c_device_id *id_table;
    ……
};
```

[关键成员]

成员名称	描述
@driver	Device driver model driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	List of I2C devices supported by this driver如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
static const struct i2c_device_id vm149c_id_table[] = {
    { VM149C_NAME, 0 },
    { { 0 } }
};
MODULE_DEVICE_TABLE(i2c, vm149c_id_table);
static const struct of_device_id vm149c_of_table[] = {
    { .compatible = "silicon touch,vm149c" },
    { { 0 } }
};
MODULE_DEVICE_TABLE(of, vm149c_of_table);
static const struct dev_pm_ops vm149c_pm_ops = {
    SET_SYSTEM_SLEEP_PM_OPS(vm149c_vcm_suspend, vm149c_vcm_resume)
    SET_RUNTIME_PM_OPS(vm149c_vcm_suspend, vm149c_vcm_resume, NULL)
};
static struct i2c_driver vm149c_i2c_driver = {
    .driver = {
        .name = VM149C_NAME,
        .pm = &vm149c_pm_ops,
        .of_match_table = vm149c_of_table,
    },
    .probe = &vm149c_probe,
    .remove = &vm149c_remove,
    .id_table = vm149c_id_table,
};
module_i2c_driver(vm149c_i2c_driver);
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ……
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
                           unsigned long arg);
#endif
    ……
};
```

[关键成员]

成员名称	描述
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops vm149c_core_ops = {
    .ioctl = vm149c_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl32 = vm149c_compat_ioctl32
#endif
};
```

目前使用了如下的私有ioctl实现马达移动时间信息的查询。

RK_VIDIOC_VCM_TIMEINFO

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*g_volatile_ctrl)(struct v4l2_ctrl *ctrl);
    int (*try_ctrl)(struct v4l2_ctrl *ctrl);
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.g_volatile_ctrl	Get a new value for this control. Generally only relevantfor volatile (and usually read-only) controls such as a control that returns the current signal strength which changes continuously.
.s_ctrl	Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

[示例]

```
static const struct v4l2_ctrl_ops vm149c_vcm_ctrl_ops = {
    .g_volatile_ctrl = vm149c_get_ctrl,
    .s_ctrl = vm149c_set_ctrl,
};
```

vm149c_get_ctrl和vm149c_set_ctrl对下面的control进行了支持

V4L2_CID_FOCUS_ABSOLUTE

API简要说明

xxxx_get_ctrl

[描述]

获取马达的移动位置。

[语法]

```
static int xxxx_get_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_set_ctrl

[描述]

设置马达的移动位置。

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_ioctl xxxx_compat_ioctl

[描述]

自定义ioctl的实现函数，主要包含获取马达移动的时间信息，

实现了自定义RK_VIDIOC_COMPAT_VCM_TIMEINFO。

[语法]

```
static int xxxx_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)
static long xxxx_compat_ioctl32(struct v4l2_subdev *sd, unsigned int cmd,
unsigned long arg)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
cmd	ioctl命令	输入
*arg/arg	参数指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

1.实现标准的i2c子设备驱动部分.

1.1 根据**struct i2c_driver**描述，主要实现以下几部分：

struct driver.name

struct driver.pm

struct driver. of_match_table

probe函数

remove函数

1.2 probe函数实现细节描述：

1）VCM设备资源获取，主要获取DTS资源，参考VCM设备注册（DTS）

1.1) RK私有资源定义，命名方式如rockchip,camera-module-xxx，主要是提供设备参数和Camera设备进行匹配。

1.2）VCM参数定义，命名方式如rockchip,vcm-xxx，主要涉及硬件参数启动电流、额定电流、移动模式，参数跟马达移动的范围和速度相关。

2） VCM v4l2设备以及media实体的初始化.

v4l2子设备：v4l2_i2c_subdev_init，RK VCM驱动要求subdev拥有自己的设备节点供用户态camera_engine访问，通过该设备节点实现调焦控制；

media实体：media_entity_init；

3）RK AF算法将模组镜头整个可移动行程的位置参数定义为[0,64]，模组镜头整个可移动行程在VCM驱动电流上对应的变化范围为[启动电流，额定电流]，该函数中建议实现这2者间的映射换算关系；

2.实现v4l2子设备驱动，主要实现以下2个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
```

2.1 参考**v4l2_subdev_core_ops**说明实现回调函数，主要实现以下回调函数：

.ioctl.compat_ioctl32

该回调主要实现RK私有控制命令，涉及：

成员名称	描述
RK_VIDIOC_VCM_TIMEINFO	camera_engine通过该命令获取此次镜头移动所需时间，据此来判断镜头何时停止以及CIS帧曝光时间段是否与镜头移动时间段有重叠；镜头移动时间与镜头移动距离、VCM driver ic电流输出模式相关。

2.2 参考**v4l2_ctrl_ops**说明实现回调函数，主要实现以下回调函数：

.g_volatile_ctrl.s_ctrl

.g_volatile_ctrl和.s_ctrl以标准的v4l2 control实现了以下命令：

成员名称	描述
V4L2_CID_FOCUS_ABSOLUTE	camera_engine通过该命令来设置和获取镜头的绝对位置， RK AF算法中将镜头整个可移动行程的位置参数定义为[0,64]。

FlashLight驱动

FLASHLight设备注册(DTS)

SGM378 DTS 参考：

```
&i2c1 {
    ...
    sgm3784: sgm3784@30 {//闪光灯设备
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "sgmicro,gsm3784";
        reg = <0x30>;
        rockchip,camera-module-index = <0>;//闪光灯对应camera模组编号
        rockchip,camera-module-facing = "back";//闪光灯对应camera模组朝向
        enable-gpio = <&gpio2 RK_PB4 GPIO_ACTIVE_HIGH>;//enable gpio
        strobe-gpio = <&gpio1 RK_PA3 GPIO_ACTIVE_HIGH>;//flash触发gpio
        status = "okay";
        sgm3784_led0: led@0 {//led0设备信息
            reg = <0x0>;//index
            led-max-microamp = <299200>;//torch模式最大电流
            flash-max-microamp = <1122000>;//flash模式最大电流
            flash-max-timeout-us = <1600000>;//falsh最大时间
        };
        sgm3784_led1: led@1 {//led1设备信息
            reg = <0x1>;//index
            led-max-microamp = <299200>;//torch模式最大电流
            flash-max-microamp = <1122000>;//flash模式最大电流
            flash-max-timeout-us = <1600000>;//falsh最大时间
        };
    };
    ...
    ov13850: ov13850@10 {
        ...
        flash-leds = <&sgm3784_led0 &sgm3784_led1>;//闪光灯设备挂接到camera
        ...
    };
    ...
}
```

GPIO、PWM控制 dts 参考：

```
flash_ir: flash-ir {
        status = "okay";
        compatible = "led,rgb13h";
        label = "pwm-flash-ir";
        led-max-microamp = <20000>;
        flash-max-microamp = <20000>;
        flash-max-timeout-us = <1000000>;
        pwms=<&pwm3 0 25000 0>;
```

```
        //enable-gpio = <&gpio0 RK_PA1 GPIO_ACTIVE_HIGH>;
        rockchip,camera-module-index = <1>;
        rockchip,camera-module-facing = "front";
};
&i2c1 {
    imx415: imx415@1a {
        ...
        flash-leds = <&flash_ir>;
        ...
    }
}
```

注意点：

1、软件上需要根据补光灯类型区分处理流程，如果是红外补光灯，dts 补光灯节点 label需要有ir字样用来识别硬件类型，led补光灯把ir字段去掉即可。

2、对于这种单个引脚控制的硬件电路，有两种情况，一种是固定亮度，直接使用gpio控制。另外一种是亮度可控，使用pwm，通过调节占空比设置亮度，dts pwms 或 enable-gpio，二选一配置。

FLASHLight驱动说明

数据类型简要说明

struct i2c_driver

[说明]

定义i2c 设备驱动信息

[定义]

```
struct i2c_driver {
    ……
    /* Standard driver model interfaces */
    int (*probe)(struct i2c_client *, const struct i2c_device_id *);
    int (*remove)(struct i2c_client *);
    ……
    struct device_driver driver;
    const struct i2c_device_id *id_table;
    ……
};
```

[关键成员]

成员名称	描述
@driver	Device driver model driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	List of I2C devices supported by this driver如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
static const struct i2c_device_id sgm3784_id_table[] = {
    { SGM3784_NAME, 0 },
    { { 0 } }
};
MODULE_DEVICE_TABLE(i2c, sgm3784_id_table);
static const struct of_device_id sgm3784_of_table[] = {
    { .compatible = "sgmicro,sgm3784" },
    { { 0 } }
};
MODULE_DEVICE_TABLE(of, sgm3784_of_table);
static const struct dev_pm_ops sgm3784_pm_ops = {
    SET_RUNTIME_PM_OPS(sgm3784_runtime_suspend, sgm3784_runtime_resume, NULL)
};
static struct i2c_driver sgm3784_i2c_driver = {
    .driver = {
        .name = sgm3784_NAME,
        .pm = &sgm3784_pm_ops,
        .of_match_table = sgm3784_of_table,
    },
    .probe = &sgm3784_probe,
    .remove = &sgm3784_remove,
    .id_table = sgm3784_id_table,
};
module_i2c_driver(vm149c_i2c_driver);
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ……
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
              unsigned long arg);
#endif
    ……
};
```

[关键成员]

成员名称	描述
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops sgm3784_core_ops = {
    .ioctl = sgm3784_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl32 = sgm3784_compat_ioctl32
#endif
};
```

目前使用了如下的私有ioctl实现闪光灯点亮时间信息的查询。

RK_VIDIOC_FLASH_TIMEINFO

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*g_volatile_ctrl)(struct v4l2_ctrl *ctrl);
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.g_volatile_ctrl	Get a new value for this control. Generally only relevantfor volatile (and usually read-only) controls such as a control that returns the current signal strength which changes continuously.
.s_ctrl	Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

[示例]

```
static const struct v4l2_ctrl_ops sgm3784_ctrl_ops[LED_MAX] = {
    [LED0] = {
        .g_volatile_ctrl = sgm3784_led0_get_ctrl,
        .s_ctrl = sgm3784_led0_set_ctrl,
    },
    [LED1] = {
        .g_volatile_ctrl = sgm3784_led1_get_ctrl,
        .s_ctrl = sgm3784_led1_set_ctrl,
    }
};
```

API简要说明

xxxx_set_ctrl

[描述]

设置闪光灯模式、电流和flash timeout时间。

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_get_ctrl

[描述]

获取闪光灯故障状态。

[语法]

```
static int xxxx_get_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_ioctl xxxx_compat_ioctl

[描述]

自定义ioctl的实现函数，主要包含获取闪光灯亮的时间信息，

实现了自定义RK_VIDIOC_COMPAT_FLASH_TIMEINFO。

[语法]

```
static int xxxx_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)

static long xxxx_compat_ioctl32(struct v4l2_subdev *sd, unsigned int cmd,
unsigned long arg)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
cmd	ioctl命令	输入
*arg/arg	参数指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

对于普通gpio直接控制led可参考使用kernel/drivers/leds/leds-rgb13h.c和

kernel/Documentation/devicetree/bindings/leds/leds-rgb13h.txt

对于flashlight driver IC可按如下步骤移植

1.实现标准的i2c子设备驱动部分.

1.1 根据**struct i2c_driver**描述，主要实现以下几部分：

struct driver.name

struct driver.pm

struct driver. of_match_table

probe函数

remove函数

1.2 probe函数实现细节描述：

1）flashlight设备资源获取，主要获取DTS资源，参考FLASHLIGHT设备注册(DTS);

1.1) RK私有资源定义，命名方式如rockchip,camera-module-xxx，主要是提供设备参数和Camera设备进行匹配。

2)flash设备名:

对于双led闪光灯，使用led0、led1设备名进行区分。

```
/* NOTE: to distinguish between two led
 * name: led0 meet the main led
 * name: led1 meet the secondary led
 */
snprintf(sd->name, sizeof(sd->name),
         "m%02d_%s_%s_led%d %s",
         flash->module_index, facing,
         SGM3784_NAME, i, dev_name(sd->dev));
```

3)FLASH v4l2设备以及media实体的初始化.

v4l2子设备：v4l2_i2c_subdev_init，RK flashlight驱动要求subdev拥有自己的设备节点供用户态camera_engine访问，通过该设备节点实现led控制；

media实体：media_entity_init；

2.实现v4l2子设备驱动，主要实现以下2个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
```

2.1 参考**v4l2_subdev_core_ops**说明实现回调函数，主要实现以下回调函数：

.ioctl.compat_ioctl32

该回调主要实现RK私有控制命令，涉及：

成员名称	描述
RK_VIDIOC_FLASH_TIMEINFO	camera_engine通过该命令获取此次led亮的时间，据此来判断CIS帧曝光时间是否在闪光灯亮之后。

2.2 参考**v4l2_ctrl_ops**说明实现回调函数，主要实现以下回调函数：

.g_volatile_ctrl.s_ctrl

.g_volatile_ctrl和.s_ctrl以标准的v4l2 control实现了以下命令：

成员名称	描述
V4L2_CID_FLASH_FAULT	获取闪光灯故障信息
V4L2_CID_FLASH_LED_MODE	设置Led模式 V4L2_FLASH_LED_MODE_NONE V4L2_FLASH_LED_MODE_TORCH V4L2_FLASH_LED_MODE_FLASH
V4L2_CID_FLASH_STROBE	控制闪光灯开
V4L2_CID_FLASH_STROBE_STOP	控制闪光灯关
V4L2_CID_FLASH_TIMEOUT	设置闪光灯模式最大持续亮时间
V4L2_CID_FLASH_INTENSITY	设置闪光灯模式电流
V4L2_CID_FLASH_TORCH_INTENSITY	设置火炬模式电流

FOCUS ZOOM P-IRIS驱动

这里驱动指的是由步进电机控制自动对焦(FOCUS)、变焦(ZOOM)、自动光圈(P-IRIS)。由于使用的步进电机控制方式一样及硬件设计的因素，将三个功能的驱动集成在一个驱动内。根据使用的驱动芯片，如SPI控制的芯片，可以将驱动封装成SPI框架子设备，本章节围绕MP6507驱动芯片描述驱动需要实现的数据结构、框架及注意事项。

FOCUS ZOOM P-IRIS设备注册(DTS)

```
mp6507: mp6507 {
    status = "okay";
    compatible = "monolithicpower,mp6507";
    #pwm-cells = <3>;
    pwms = <&pwm6 0 25000 0>,
           <&pwm10 0 25000 0>,
           <&pwm9 0 25000 0>,
           <&pwm8 0 25000 0>;
    pwm-names = "ain1","ain2","bin1","bin2";
    rockchip,camera-module-index = <1>;
    rockchip,camera-module-facing = "front";
    iris_en-gpios  = <&gpio0 RK_PC2 GPIO_ACTIVE_HIGH>;
    focus_en-gpios  = <&gpio0 RK_PC3 GPIO_ACTIVE_HIGH>;
    zoom_en-gpios  = <&gpio0 RK_PC0 GPIO_ACTIVE_HIGH>;
    iris-step-max = <80>;
    focus-step-max = <7500>;
    zoom-step-max = <7500>;
    iris-start-up-speed = <1200>;
    focus-start-up-speed = <1200>;
    focus-max-speed = <2500>;
    zoom-start-up-speed = <1200>;
    zoom-max-speed = <2500>;
    focus-first-speed-step = <8>;
    zoom-first-speed-step = <8>;
    focus-speed-up-table = < 1176 1181 1188 1196
                             1206 1217 1231 1246
                             1265 1286 1309 1336
                             1365 1396 1429 1464
                             1500 1535 1570 1603
                             1634 1663 1690 1713
                             1734 1753 1768 1782
                             1793 1803 1811 1818>;
    focus-speed-down-table = < 1796 1788 1779 1768
                               1756 1743 1728 1712
                               1694 1674 1653 1630
                               1605 1580 1554 1527
                               1500 1472 1445 1419
                               1394 1369 1346 1325
                               1305 1287 1271 1256
                               1243 1231 1220 1211
                               1203 1195 1189 1184
                               1179 1175>;
    zoom-speed-up-table = < 1198 1205 1212 1220
                            1228 1238 1249 1260
                            1272 1285 1299 1313
                            1328 1343 1359 1375
                            1390 1406 1421 1436
```

```
                            1450 1464 1477 1489
                            1500 1511 1521 1529
                            1537 1544 1551>;
    zoom-speed-down-table = < 1547 1540 1531 1522
                              1511 1499 1487 1473
                              1458 1443 1426 1409
                              1392 1375 1357 1340
                              1323 1306 1291 1276
                              1262 1250 1238 1227
                              1218 1209 1202 1195
                              1189 1184 1179 1175
                              1171 1168>;
};

&i2c1 {
    imx334: imx334@1a {
        ...
        lens-focus = <&mp6507>;
        ...
    }
}

&pwm6 {
    status = "okay";
    pinctrl-names = "active";
    pinctrl-0 = <&pwm6m1_pins_pull_up>;
};

&pwm8 {
    status = "okay";
    pinctrl-names = "active";
    pinctrl-0 = <&pwm8m1_pins_pull_down>;
    center-aligned;
};

&pwm9 {
    status = "okay";
    pinctrl-names = "active";
    pinctrl-0 = <&pwm9m1_pins_pull_down>;
    center-aligned;
};

&pwm10 {
    status = "okay";
    pinctrl-names = "active";
    pinctrl-0 = <&pwm10m1_pins_pull_down>;
};
```

RK私有定义说明：

成员名称	描述
rockchip,camera-module-index	camera序号，和camera匹配的字段
rockchip,camera-module-facing	camera朝向，和camera匹配的字段
iris_en-gpios	IRIS使能GPIO
focus_en-gpios	focus使能GPIO
zoom_en-gpios	zoom使能GPIO
rockchip,iris-step-max	P-IRIS步进电机移动的最大步数
rockchip,focus-step-max	对焦步进电机移动的最大步数
zoom-step-max	变焦步进电机移动的最大步数
iris-start-up-speed	IRIS使用的步进电机的启动速度
focus-start-up-speed	focus使用的步进电机的启动速度
focus-max-speed	focus使用的步进电机的最大运行速度
zoom-start-up-speed	zoom使用的步进电机的启动速度
zoom-max-speed	zoom使用的步进电机的最大运行速度
focus-first-speed-step	focus启动速度运行的步数，后续加速区间等比例增加步数，使各个速度段运行的时间尽量接近一致
zoom-first-speed-step	zoom启动速度运行的步数，后续加速区间等比例增加步数，使各个速度段运行的时间尽量接近一致
focus-speed-up-table	focus加速曲线采用查表方式，调整参数生成加速曲线，将生成的梯形加速曲线或 S 型加速曲线的数据表配置进来，不配置或配置单个数据，则直接按启动速度匀速运行；加速曲线最小值不超过马达最大启动速度，最大值不超过步进马达最大运行速度。
focus-speed-down-table	focus减速曲线，减速曲线最大值需小于加速曲线最大值；若加速曲线无效，则减速曲线一样无效，全程按启动速度匀速运行；若没有配置减速曲线，则减速曲线由加速曲线对称得到。
zoom-speed-up-table	zoom加速曲线采用查表方式，调整参数生成加速曲线，将生成的梯形加速曲线或 S 型加速曲线的数据表配置进来，不配置或配置单个数据，则直接按启动速度匀速运行；加速曲线最小值不超过马达最大启动速度，最大值不超过步进马达最大运行速度。

成员名称	描述
zoom-speed-down-table	zoom减速曲线，减速曲线最大值需小于加速曲线最大值；若加速曲线无效，则减速曲线一样无效，全程按启动速度匀速运行；若没有配置减速曲线，则减速曲线由加速曲线对称得到。

数据类型简要说明

struct platform_driver

[说明]

定义平台设备驱动信息

[定义]

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
        const struct platform_device_id *id_table;
        bool prevent_deferred_probe;
};
```

[关键成员]

成员名称	描述
@driver	struct device_driver driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
#if defined(CONFIG_OF)
static const struct of_device_id motor_dev_of_match[] = {
    { .compatible = "monolithicpower,mp6507", },
    {},
};
#endif

static struct platform_driver motor_dev_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
```

```
        .of_match_table = of_match_ptr(motor_dev_of_match),
    },
    .probe = motor_dev_probe,
    .remove = motor_dev_remove,
};
module_platform_driver(motor_dev_driver);
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ……
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
              unsigned long arg);
#endif
    ……
};
```

[关键成员]

成员名称	描述
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops motor_core_ops = {
    .ioctl = motor_ioctl,
};
static const struct v4l2_subdev_ops motor_subdev_ops = {
    .core   = &motor_core_ops,
};
```

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*g_volatile_ctrl)(struct v4l2_ctrl *ctrl);
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.g_volatile_ctrl	Get a new value for this control. Generally only relevantfor volatile (and usually read-only) controls such as a control that returns the current signal strength which changes continuously.
.s_ctrl	Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

[示例]

```
static const struct v4l2_ctrl_ops motor_ctrl_ops = {
    .s_ctrl = motor_s_ctrl,
};
```

API简要说明

xxxx_set_ctrl

[描述]

调用标准v4l2_control设置对焦、变焦、P光圈位置。

实现了以下v4l2标准命令：

成员名称	描述
V4L2_CID_FOCUS_ABSOLUTE	控制对焦，0表示焦距最小，近处清晰
V4L2_CID_ZOOM_ABSOLUTE	控制变焦倍数，0表示放大倍数最小，视场角最大
V4L2_CID_IRIS_ABSOLUTE	控制P光圈开口的大小，0表示光圈关闭

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_get_ctrl

[描述]

调用标准v4l2_control获取对焦、变焦、P光圈当前的位置。

实现了以下v4l2标准命令：

成员名称	描述
V4L2_CID_FOCUS_ABSOLUTE	控制对焦，0表示焦距最小，近处清晰
V4L2_CID_ZOOM_ABSOLUTE	控制变焦倍数，0表示放大倍数最小，视场角最大
V4L2_CID_IRIS_ABSOLUTE	控制P光圈开口的大小，0表示光圈关闭

[语法]

```
static int xxxx_get_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_ioctl xxxx_compat_ioctl

[描述]

自定义ioctl的实现函数，主要包含获取对焦、变焦、P光圈的时间信息（开始移动及结束移动的时间戳），由于使用的镜头没有定位装置，在必要的时候，需要对镜头马达位置进行复位。

实现了自定义：

成员名称	描述
RK_VIDIOC_VCM_TIMEINFO	对焦的时间信息，用来确认当前帧是否为对焦完成后的生效帧
RK_VIDIOC_ZOOM_TIMEINFO	变焦的时间信息，用来确认当前帧是否为变焦完成后的生效帧
RK_VIDIOC_IRIS_TIMEINFO	光圈的时间信息，用来确认当前帧是否为光圈调整后的生效帧
RK_VIDIOC_FOCUS_CORRECTION	对焦位置校正（复位）
RK_VIDIOC_ZOOM_CORRECTION	变焦位置校正（复位）
RK_VIDIOC_IRIS_CORRECTION	光圈位置校正（复位）

[语法]

```
static int xxxx_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)

static long xxxx_compat_ioctl32(struct v4l2_subdev *sd, unsigned int cmd,
unsigned long arg)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
cmd	ioctl命令	输入
*arg/arg	参数指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

对于SPI控制的驱动芯片，可以使用SPI框架进行设备驱动移植，RK参考驱动使用MP6507，直接使用pwm输出控制波形，通过MP6507进行功率放大，所以直接platform框架移植。
驱动参考：/kernel/drivers/media/i2c/mp6507.c

移植步骤如下：

1.实现标准的platform子设备驱动部分.

1.1 根据**struct platform_driver**描述，主要实现以下几部分：

struct driver.name

struct driver. of_match_table

probe函数

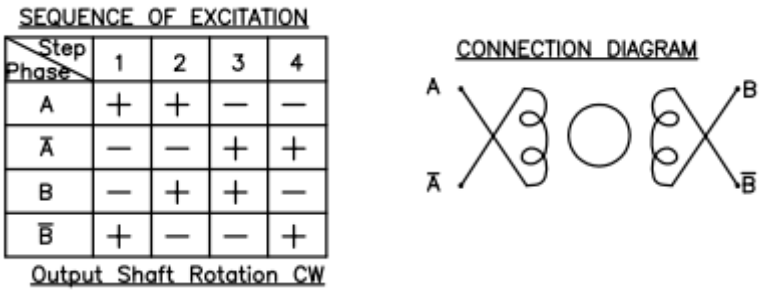
remove函数

1.2 probe函数实现细节描述：

1） 设备资源获取，主要获取DTS资源，参考FOCUS ZOOM P-IRIS设备注册(DTS)；

1.1) RK私有资源定义，命名方式如rockchip,camera-module-xxx，主要是提供设备参数和Camera设备进行匹配。

1.2) 获取pwm配置，根据马达的控制方式，AB相相差90度，可通过将B相的PWM设置中心对齐实现，在dts pwm节点配置center-aligned，详情见FOCUS ZOOM P-IRIS设备注册(DTS)；



1.3) 获取使能引脚，MP6507需要使用4个pwm产生步进电机控制波形，由于硬件pwm有限，而对焦、变焦、P光圈三个步进电机各使用一个MP6507驱动器驱动，所以通过gpio来使能对应的MP6507驱动器，从而实现pwm分时复用，当然这也有个弊端，同一时刻只能驱动一个步进电机，其他两个步进电机需等待上一个操作结束才能继续操作；
1.4) 获取各个电机的最大步程、最大启动速度、最大运行速度、加速曲线数据等硬件相关限制条件及资源；

2) hrtimer_init，定时器初始化，pwm使用的是continuous模式，需要定时器定时，达到指定输出pwm波形个数后，进定时器中断关闭pwm，加速过程也需要在运行到指定波形个数后进入定时器中断修改pwm频率，从而实现步进电机的加速;

3) init_completion，通过completion实现同步机制，只有前面一个马达移动操作结束，下一个马达操作才能进行；

4) v4l2设备以及media实体的初始化.

v4l2子设备：v4l2_i2c_subdev_init，驱动要求subdev拥有自己的设备节点供用户态rkaiq访问，通过该设备节点实现对马达的控制；

media实体：media_entity_init；

5) flash设备名:

```
snprintf(sd->name, sizeof(sd->name), "m%02d_%s_%s",
         motor->module_index, facing,
         DRIVER_NAME);
```

2.实现v4l2子设备驱动，主要实现以下2个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
```

2.1 参考**v4l2_subdev_core_ops**说明实现回调函数，主要实现以下回调函数：

```
.ioctl
.compat_ioctl32
```

该回调主要实现RK私有控制命令，涉及：

成员名称	描述
RK_VIDIOC_VCM_TIMEINFO	对焦的时间信息，用来确认当前帧是否为对焦完成后的生效帧
RK_VIDIOC_ZOOM_TIMEINFO	变焦的时间信息，用来确认当前帧是否为变焦完成后的生效帧
RK_VIDIOC_IRIS_TIMEINFO	光圈的时间信息，用来确认当前帧是否为光圈调整后的生效帧
RK_VIDIOC_FOCUS_CORRECTION	对焦位置校正（复位）
RK_VIDIOC_ZOOM_CORRECTION	变焦位置校正（复位）
RK_VIDIOC_IRIS_CORRECTION	光圈位置校正（复位）

2.2 参考**v4l2_ctrl_ops**说明实现回调函数，主要实现以下回调函数：

.g_volatile_ctrl

.s_ctrl

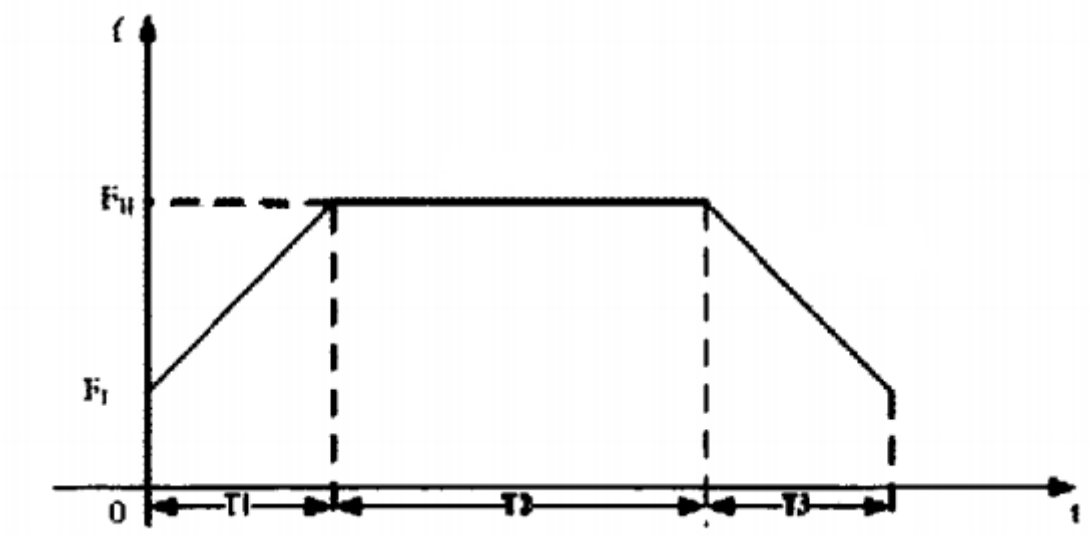
.g_volatile_ctrl和.s_ctrl以标准的v4l2 control实现了以下命令：

参数名称	描述
V4L2_CID_FOCUS_ABSOLUTE	控制对焦，0表示焦距最小，近处清晰
V4L2_CID_ZOOM_ABSOLUTE	控制变焦倍数，0表示放大倍数最小，视场角最大
V4L2_CID_IRIS_ABSOLUTE	控制P光圈开口的大小，0表示光圈关闭**

3. 步进电机加速曲线参考:

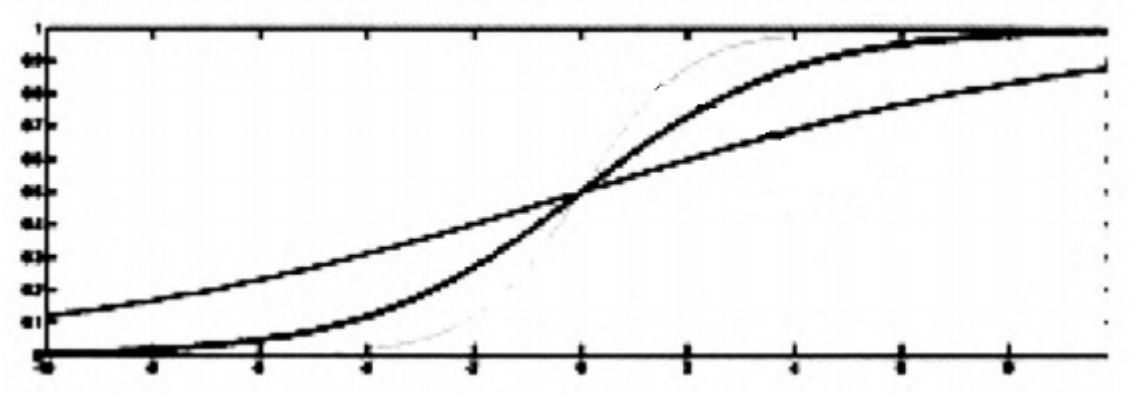
3.1 梯形曲线

可以简单按图示等间隔等速度进行加速、减速操作。



3.2 S型曲线

梯形加速如果不理想，可以考虑S型加速，可参考如下公式：
Speed = Vmin + ((Vmax - Vmin) / (1 + exp(-fac * (i - Num) / Num)));
其中，
Vmin指马达启动速度
Vmax指马达目标速度
fac是曲线系数，一般范围在4~6，值越大曲线中间越陡
i是速度分段序号，如分成32段加速，取值0~31
Num是速度分段数的一半，如分成32段，则num为16



DC-IRIS驱动

DC-IRIS相对于P-IRIS，无法准确知道光圈开口的大小，一般使用场景是默认全开，当曝光调节到最小时，图像还是过曝，则进入光圈调整，当曝光设置到最大，图像还是欠曝，进入光圈调整。DC-IRIS电机是直流电机，通过霍尔器件负反馈缓冲电机的调节速度。对于驱动而言，只要通过一个pwm控制电机转动，当pwm占空比小于20%，光圈会慢慢关闭，直到完全关闭，占空比越小，光圈关闭的速度越快；当占空比大于40%光圈会慢慢打开，占空比越大，打开速度越快；20%~40%区间光圈处于hold住的状态。这边的20%及40%不是定值，与pwm的频率及实际的硬件器件精度有关。
参考驱动： /kernel/drivers/media/i2c/hall-dc-motor.c

DC-IRIS设备注册(DTS)

```
hal_dc_motor: hal_dc_motor{
        status = "okay";
        compatible = "rockchip,hall-dc";
        pwms = <&pwm6 0 2500 0>;
        rockchip,camera-module-index = <1>;
        rockchip,camera-module-facing = "front";
};
&pwm6 {
        status = "okay";
        pinctrl-names = "active";
        pinctrl-0 = <&pwm6m0_pins_pull_down>;
};
&i2c1 {
    imx334: imx334@1a {
        ...
        lens-focus = <&hal_dc_motor>;
        ...
    }
}
```

数据类型简要说明

struct platform_driver

[说明]

定义平台设备驱动信息

[定义]

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
        const struct platform_device_id *id_table;
        bool prevent_deferred_probe;
};
```

[关键成员]

成员名称	描述
@driver	struct device_driver driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
#if defined(CONFIG_OF)
static const struct of_device_id motor_dev_of_match[] = {
    { .compatible = "rockchip,hall-dc", },
    {},
};
#endif

static struct platform_driver motor_dev_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(motor_dev_of_match),
    },
    .probe = motor_dev_probe,
    .remove = motor_dev_remove,
};
module_platform_driver(motor_dev_driver);
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ……
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
              unsigned long arg);
#endif
    ……
};
```

[关键成员]

成员名称	描述
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops motor_core_ops = {
    .ioctl = motor_ioctl,
};
static const struct v4l2_subdev_ops motor_subdev_ops = {
    .core   = &motor_core_ops,
};
```

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.s_ctrl	Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

[示例]

```
static const struct v4l2_ctrl_ops motor_ctrl_ops = {
    .s_ctrl = motor_s_ctrl,
};
```

API简要说明

xxxx_set_ctrl

[描述]

调用标准v4l2_control光圈位置，DC光圈实际上无法知道光圈的具体位置，这边设置的值是pwm的占空比。

实现了以下v4l2标准命令：

参数名称	描述
V4L2_CID_IRIS_ABSOLUTE	设置控制光圈的pwm的占空比，范围（0~100）

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_ioctl xxxx_compat_ioctl

[描述]

目前无私有定义需要实现，v4l2框架注册需要，实现空函数。

[语法]

```
static int xxxx_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)

static long xxxx_compat_ioctl32(struct v4l2_subdev *sd, unsigned int cmd,
unsigned long arg)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
cmd	ioctl命令	输入
*arg/arg	参数指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

驱动参考：/kernel/drivers/media/i2c/hall-dc-motor.c

移植步骤如下：

1.实现标准的platform子设备驱动部分.

1.1 根据**struct platform_driver**描述，主要实现以下几部分：

struct driver.name

struct driver. of_match_table

probe函数

remove函数

1.2 probe函数实现细节描述：

1） 设备资源获取，主要获取DTS资源，参考DC-IRIS设备注册(DTS);

```
1.1) RK私有资源定义，命名方式如rockchip,camera-module-xxx，主要是提供设备参数和Camera设备进行匹配。

1.2) 获取pwm资源，要注意pwm节点是否有使能。
```

2) v4l2设备以及media实体的初始化.

v4l2子设备：v4l2_i2c_subdev_init，驱动要求subdev拥有自己的设备节点供用户态rkaiq访问，通过该设备节点实现对马达的控制；

media实体：media_entity_init；

3) flash设备名:

```
snprintf(sd->name, sizeof(sd->name), "m%02d_%s_%s",
         motor->module_index, facing,
         DRIVER_NAME);
```

2.实现v4l2子设备驱动，主要实现以下2个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
```

2.1 参考**v4l2_subdev_core_ops**说明实现回调函数，主要实现以下回调函数：

```
ioctl
.compat_ioctl32
```

该回调目前不需要实现具体命令，但是作为v4l2子设备必须实现该操作函数，所以这边实现了一个空函数。

2.2 参考**v4l2_ctrl_ops**说明实现回调函数，主要实现以下回调函数：

.g_volatile_ctrl.s_ctrl

.g_volatile_ctrl和.s_ctrl以标准的v4l2 control实现了以下命令：

成员名称	描述
V4L2_CID_IRIS_ABSOLUTE	设置控制光圈的pwm的占空比，范围（0~100）

RK-IRCUT驱动

IRCUT由两根线控制，对这两根线施加3.5v~6v的电源，通过对IRCUT供电电源的正负极对调，且满足通电时间100ms±10%，能够实现IRCUT的切换。驱动通过两个gpio控制电机驱动器的电流输出方向，gpio命令为open（红线）、close（黑线）。电流由open流向close，为红外截止滤光片，白天工作状态；电流由close流向open，为白玻璃片，夜晚工作状态。

RK-IRCUT设备注册(DTS)

```
cam_ircut0: cam_ircut {
        status = "okay";
        compatible = "rockchip,ircut";
        ircut-open-gpios = <&gpio2 RK_PA7 GPIO_ACTIVE_HIGH>;
        ircut-close-gpios  = <&gpio2 RK_PA6 GPIO_ACTIVE_HIGH>;
        rockchip,camera-module-index = <1>;
        rockchip,camera-module-facing = "front";
};

&i2c1 {
    imx334: imx334@1a {
        ...
        ir-cut = <&cam_ircut0>;
        ...
    }
```

```
}
```

数据类型简要说明

struct platform_driver

[说明]

定义平台设备驱动信息

[定义]

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
        const struct platform_device_id *id_table;
        bool prevent_deferred_probe;
};
```

[关键成员]

成员名称	描述
@driver	struct device_driver driver主要包含驱动名称和与DTS注册设备进行匹配的of_match_table。当of_match_table中的compatible域和dts文件的compatible域匹配时，.probe函数才会被调用
@id_table	如果kernel没有使用of_match_table和dts注册设备进行进行匹配，则kernel使用该table进行匹配
@probe	Callback for device binding
@remove	Callback for device unbinding

[示例]

```
#if defined(CONFIG_OF)
static const struct of_device_id ircut_of_match[] = {
    { .compatible = "rockchip,ircut", },
    {},
};
#endif

static struct platform_driver ircut_driver = {
    .driver = {
        .name = RK_IRCUT_NAME,
        .of_match_table = of_match_ptr(ircut_of_match),
    },
    .probe = ircut_probe,
    .remove = ircut_drv_remove,
};
```

```
module_platform_driver(ircut_driver);
```

struct v4l2_subdev_core_ops

[说明]

Define core ops callbacks for subdevs.

[定义]

```
struct v4l2_subdev_core_ops {
    ......
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd,
              unsigned long arg);
#endif
    ......
};
```

[关键成员]

成员名称	描述
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core.used to provide support for private ioctls used on the driver.
.compat_ioctl32	called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.in order to fix data passed from/to userspace.

[示例]

```
static const struct v4l2_subdev_core_ops ircut_core_ops = {
    .ioctl = ircut_ioctl,
};

static const struct v4l2_subdev_ops ircut_subdev_ops = {
    .core   = &ircut_core_ops,
};
```

struct v4l2_ctrl_ops

[说明]

The control operations that the driver has to provide.

[定义]

```
struct v4l2_ctrl_ops {
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

[关键成员]

成员名称	描述
.s_ctrl	Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

[示例]

```
static const struct v4l2_ctrl_ops ircut_ctrl_ops = {
    .s_ctrl = ircut_s_ctrl,
};
```

API简要说明

xxxx_set_ctrl

[描述]

调用标准v4l2_control切换IRCUT。

实现了以下v4l2标准命令：

参数名称	描述
V4L2_CID_BAND_STOP_FILTER	0是CLOSE状态，红外光可进入； 3是OPEN状态，红外光不可进入；

[语法]

```
static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)
```

[参数]

参数名称	描述	输入输出
*ctrl	v4l2 control结构体指针	输入

[返回值]

返回值	描述
0	成功
非0	失败

xxxx_ioctl xxxx_compat_ioctl

[描述]

目前无私有定义需要实现，v4l2框架注册需要，实现空函数。

[语法]

```
static int xxxx_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)

static long xxxx_compat_ioctl32(struct v4l2_subdev *sd, unsigned int cmd,
unsigned long arg)
```

[参数]

参数名称	描述	输入输出
*sd	v4l2 subdev结构体指针	输入
cmd	ioctl命令	输入
*arg/arg	参数指针	输出

[返回值]

返回值	描述
0	成功
非0	失败

驱动移植步骤

驱动参考：/kernel/drivers/media/i2c/rk_ircut.c

移植步骤如下：

1.实现标准的platform子设备驱动部分.

1.1 根据**struct platform_driver**描述，主要实现以下几部分：

struct driver.name

struct driver. of_match_table

probe函数

remove函数

1.2 probe函数实现细节描述：

1） 设备资源获取，主要获取DTS资源，参考RK-IRCUT设备注册(DTS)；

1.1) RK私有资源定义，命名方式如rockchip,camera-module-xxx，主要是提供设备参数和Camera设备进行匹配。

1.2) 获取open、close gpio资源；

2) init_completion，通过completion实现同步机制，由于切换IRCUT需要约100ms，需要completion同步机制来确保上一次IRCUT切换已经完成，才能再次进行操作；

3） 创建工作队列，将切换操作放在work queue，避免长时间阻塞；

4) v4l2设备以及media实体的初始化.

v4l2子设备：v4l2_i2c_subdev_init，驱动要求subdev拥有自己的设备节点供用户态rkaiq访问，通过该设备节点实现对IRCUT的控制；

media实体：media_entity_init；

```
sd->entity.function = MEDIA_ENT_F_LENS;
sd->entity.flags = 1;//flag固定为1，用于区分其他MEDIA_ENT_F_LENS类型的子设备
```

5) 设备名:

```
snprintf(sd->name, sizeof(sd->name), "m%02d_%s_%s",
         ircut->module_index, facing,
         RK_IRCUT_NAME);
```

2.实现v4l2子设备驱动，主要实现以下2个成员：

```
struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops
```

2.1 参考**v4l2_subdev_core_ops**说明实现回调函数，主要实现以下回调函数：

```
.ioctl
.compat_ioctl32
```

该回调目前不需要实现私有命令，但是v4l2框架注册有要求，故实现空函数，后续可根据需求补充函数内容。

2.2 参考**v4l2_ctrl_ops**说明实现回调函数，主要实现以下回调函数：

.s_ctrl

.s_ctrl以标准的v4l2 control实现了以下命令：

成员名称	描述
V4L2_CID_BAND_STOP_FILTER	0是CLOSE状态，红外光可进入； 3是OPEN状态，红外光不可进入；

media-ctl v4l2-ctl工具

media-ctl工具的操作是通过/dev/medio0等media 设备，它管理的是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。

具体用法可以参考命令的帮助信息，下面是常见的几个使用。

1、打印拓扑结构

```
media-ctl -p -d /dev/media0
```

注: isp2的设备节点较多，可能存在media0/media1/media2节点 ，需要逐个枚举查看设备信息。

2、链接

```
media-ctl -l '"rkisp-isp-subdev":2->"rkisp-bridge-ispp":0[0]'
media-ctl -l '"rkisp-isp-subdev":2->"rkisp_mainpath":0[1]'
```

注： 把ispp的通路断开，链接到main_path，从main_path抓取raw图，media-ctl 没加-d指定设备，默认是/dev/media0设备，需要确认rkisp-isp-subdev挂在在哪个设备节点上，一般是/dev/media1。

3、修改fmt/size

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

注：需要确认camera设备节点（ov5695 7-0036）挂载在哪个media设备。

4、设置fmt并抓帧

```
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
```

5、设置曝光、gain等control

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

注： isp驱动会调用camera子设备的control命令，所以指定设备为video3（main_path or self_path）可以设置到曝光，vicap不会去调用camera子设备的control命令，对采集节点直接设置control命令会失败。正确做法是找到camera设备节点是/dev/v4l-subdevX，对终端节点直接配置。

rv1109/rv1126内存优化指南

MIPI -> DDR_1 -> ISP -> DDR_2 -> ISPP(TNR) -> DDR_3 -> ISPP(NR&Sharp) -> DDR_4 -> ISPP(FEC) -> DDR_5

1、DDR_1：vicap raw数据写到ddr，或者isp mipi raw数据写到ddr，isp再从ddr读取raw数据处理

占用内存：buf_cnt * buf_size * N，(N = 1:线性模式, 2:hdr2帧模式 3: hdr3帧模式)。

buf_size： ALIGN(width * bpp / 8, 256) * height; //bpp为位宽，raw8 raw10或raw12

buf_cnt： 默认4个，定义aiq库代码hwi/isp20/CamHwIsp20.h，最小需要3个。

#define ISP_TX_BUF_NUM 4

#define VIPCAP_TX_BUF_NUM 4

2、DDR_2：isp fbc yuv420和gain数据写到ddr，ispp再从ddr读取处理

占用内存：buf_size * buf_cnt

buf_size：ALIGN(width, 64) * ALIGN(height, 128) / 16 + ALIGN(width, 16) * ALIGN(hieght, 16) * 1.5625

buf_cnt：tnr 3to1模式4个buf，2to1模式3个buf，模式在iq xml中配置

3、DDR_3： ispp tnr fbc yuv420和gain数据写到的ddr，ispp NR&Sharp再从ddr读取处理

占用内存：buf_size * buf_cnt

buf_size：ALIGN(width, 64) * ALIGN(height, 128) / 16 + ALIGN(width, 16) * ALIGN(hieght, 16) * 1.5625

buf_cnt：2个，已最小

4、DDR_4：ispp NR&Sharp yuyv数据写到ddr，ispp fec再从ddr读取处理

占用内存：buf_size * buf_cnt （fec功能不开不占用内存）

buf_size：width * height * 2

buf_cnt：2个，已最小

5、DDR_5：ispp 4路输出图像buffer，根据用户设置分辨率、格式和**buf_cnt**计算buffer大小

上述**buf_cnt**为内存可优化配置的地方

FAQ

如何获取驱动版本号

从kernel启动log中获取

```
rkisp ffb50000.rkisp: rkisp driver version: v00.01.00
rkispp ffb60000.rkispp: rkispp driver version: v00.01.00
```

由以下命令获取

```
cat /sys/module/video_rkisp/parameters/version
cat /sys/module/video_rkispp/parameters/version
```

如何判断RKISP驱动加载状态

RKISP驱动如果加载成功，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 RKISP注册的video节点。

```
localhost ~ # grep '' /sys/class/video4linux/video*/name
```

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

判断camera驱动是否加载成功，当所有的camera都注册完毕，kernel会打印出如下的log。

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

如发现kernel没有Async subdev notifier completed这行log，那么请首先查看sensor是否有相关的报错，I2C通讯是否成功。

如何抓取ispp输出的yuv数据

ispp输入数据来源rkisp_mainpath、rkisp_selfpath和rkispp_input_image link关闭，rkisp-bridge-ispp link开启，rkisp-isp-subdev pad2: Source 格式必须是fmt:YUYV8_2X8，默认状态不需要配置link，参考命令如下，

```
media-ctl -l '"rkisp-isp-subdev":2->"rkisp_mainpath":0[0]'

media-ctl -l '"rkisp-isp-subdev":2->"rkisp_selfpath":0[0]'

media-ctl -l '"rkisp-isp-subdev":2->"rkisp-bridge-ispp":0[1]'

media-ctl -d /dev/media1 -l '"rkispp_input_image":0->"rkispp-subdev":0[1]'

v4l2-ctl -d /dev/video13 \

--set-fmt-video=width=2688,height=1520,pixelformat=NV12 \

--stream-mmap=3 --stream-to=/tmp/nv12.out --stream-count=20 --stram-poll
```

如何抓取Sensor输出的Bayer Raw数据

参考命令如下，

```
media-ctl -d /dev/media0 --set-v4l2 '"m01_f_os04a10 1-0036-
1":0[fmt:SBGGR12_1X12/2688x1520]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp-isp-
subdev":0[fmt:SBGGR12_1X12/2688x1520]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp-isp-subdev":0[crop:(0,0)/2688x1520]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp-isp-
subdev":2[fmt:SBGGR12_1X12/2688x1520]'

media-ctl -l '"rkisp-isp-subdev":2->"rkisp-bridge-ispp":0[0]'

media-ctl -l '"rkisp-isp-subdev":2->"rkisp_mainpath":0[1]'
v4l2-ctl -d /dev/video0 --set-ctrl 'exposure=1216,analogue_gain=10' \
--set-selection=target=crop,top=0,left=0,width=2688,height=1520 \
--set-fmt-video=width=2688,height=1520,pixelformat=BG12 \
--stream-mmap=3 --stream-to=/tmp/mp.raw.out --stream-count=1 --stream-poll
```

注：

1、指定media节点： -d /dev/media0 要看后面的节点实际挂载的media节点配置。

2、 "m01_f_os04a10 1-0036-1"是设置sensor的分辨率，sensor驱动若支持多个分辨率，可以通过这个命令去切分辨率。

3、"rkisp-isp-subdev":0 ，0是指pad0，isp的输入端口；"rkisp-isp-subdev":2，2指pad2，isp的输出端口，根据实际需要的format配置isp的输入输出端口。

4、需要注意的是， ISP 虽然不对 Raw 图处理， 但它仍然会将 12bit 的数据低位补 0 成16bit。 不管 Sensor 输入的是 10bit/12bit， 最终上层得到的都是16bit每像素。

如何支持黑白摄像头

CIS驱动需要将黑白sensor的输出format改为如下三种format之一，

```
MEDIA_BUS_FMT_Y8_1X8 （sensor 8bit输出）

MEDIA_BUS_FMT_Y10_1X10 （sensor 10bit输出）

MEDIA_BUS_FMT_Y12_1X12 （sensor 12bit输出）
```

即在函数xxxx_get_fmt和xxxx_enum_mbus_code返回上述format。

RKISP驱动会对这三种format进行特别设置，以支持获取黑白图像。

另外，如应用层需要获取Y8格式的图像，则只能使用SP Path，因为只有SP Path可以支持Y8格式输出。

如何支持奇偶场合成

RKISP 驱动支持奇偶场合成功能，限制要求：

1. MIPI接口： 支持输出frame count number (from frame start and frame end short packets)，RKISP驱动以此来判断当前场的奇偶；
2. BT656接口：支持输出标准SAV/EAV，即bit6为有奇场偶场标记信息，RKISP驱动以此来判断当前场的奇偶；
3. RKISP驱动中RKISP1_selfpath video设备节点具备该功能，其他video设备节点不具备该功能，app层误调用其他设备节点的话，驱动提示以下错误信息：

“only selfpath support interlaced”

RKISP_selfpath信息可以media-ctl -p查看:

```
entity 3: rkisp_selfpath (1 pad, 1 link)
     type Node subtype V4L flags 0
      device node name /dev/video1
    pad0: Sink
       <- "rkisp-isp-subdev":2 [ENABLED]
```

设备驱动实现方式如下：

设备驱动format.field需要设置为V4L2_FIELD_INTERLACED，表示此当前设备输出格式为奇偶场，即在函数xxxx_get_fmt返回format.field格式。可参考driver/media/i2c/tc35874x.c驱动；

如何查看debug信息

1、查看media pipeline信息，此对应dts camera配置

```
media-ctl -p -d /dev/mediaX  (X = 0, 1, 2 ..)
```

2、查看proc信息，此为isp/ispp单前状态和帧输入输出信息，可以多cat几次

```
cat /proc/rkisp*
```

3、查看驱动debug信息，设置debug level到isp和ispp节点，level数值越大信息越多

```
echo n > /sys/module/video_rkisp/parameters/debug (n = 0, 1, 2, 3; 0为关)
echo n > /sys/module/video_rkispp/parameters/debug
```

4、查看寄存器信息，把isp.reg和ispp.reg pull出来

```
io -4 -l 0x10000 0xffb50000 > /tmp/isp.reg
io -4 -l 0x1000 0xffb60000 > /tmp/ispp.reg
```

5、提供debug信息步骤

1) 有问题现场 1->2->4->3

2) 复现问题 3->启动->复现->1->2->4

6、proc信息说明

```
[root@RV1126_RV1109:/]# cat /proc/rkisp0
rkisp0      Version:v00.01.07
Input       rkcif_mipi_lvds Format:SGBRG10_1X10 Size:3840x2160@20fps Offset(0,0)
| RDBK_X2(frame:1378 rate:49ms)
Output      rkispp0 Format:FBC420 Size:3840x2160 (frame:1377 rate:51ms)
Interrupt   Cnt:6550 ErrCnt:0
clk_isp     594000000
aclk_isp    500000000
hclk_isp    250000000
DPCC0       ON(0x40000005)
DPCC1       ON(0x40000005)
DPCC2       ON(0x40000005)
BLS         ON(0x40000001)
SDG         OFF(0x80446197)
LSC         ON(0x1)
AWBGAIN     ON(0x80446197) (gain: 0x010d010d, 0x01f20218)
DEBAYER     ON(0xf000111)
CCM         ON(0xc0000001)
GAMMA_OUT   ON(0xc0000001)
CPROC       ON(0xf)
IE          OFF(0x0) (effect: BLACKWHITE)
WDR         OFF(0x30cf0)
HDRTMO      ON(0xc7705a23)
HDRMGE      ON(0xc0000005)
RAWNR       ON(0xc0100001)
GIC         OFF(0x0)
DHAZ        ON(0xc0101019)
3DLUT       OFF(0x2)
GAIN        ON(0xc0010111)
LDCH        OFF(0x0)
CSM         FULL(0x80446197)
SIAF        OFF(0x0)
SIAWB       OFF(0x0)
YUVAE       ON(0x400100f3)
SIHST       ON(0x38000107)
RAWAF       ON(0x7)
RAWAWB      ON(0x4037e887)
RAWAE0      ON(0x40000003)
RAWAE1      ON(0x400000f5)
```

```
RAWAE2      ON(0x400000f5)
RAWAE3      ON(0x400000f5)
RAWHIST0    ON(0x40000501)
RAWHIST1    ON(0x60000501)
RAWHIST2    ON(0x60000501)
RAWHIST3    ON(0x60000501)
```

Input: 输入源、输入格式、分辨率、DDR回读次数、当前帧号、实际帧间隔

Output: 输出对象、输出格式、分辨率、当前帧号、实际帧间隔

Interrupt: 包含mipi中断、isp内各模块的中断，数据有递增，说明有数据进isp，ErrCnt错误中断统计信息

clk_isp: isp 时钟频率

其他: isp各个模块的开关状态

```
[root@RV1126_RV1109:/]# cat /proc/rkispp0
rkispp0     Version:v00.01.07
Input       rkisp0 Format:FBC420 Size:3840x2160 (frame:1656 rate:51ms delay:85ms)
Output      rkispp_scale0 Format:NV12 Size:1920x1080 (frame:1655 rate:51ms
delay:108ms)
TNR         ON(0xd00000d) (mode: 2to1) (global gain: disable) (frame:1656
time:13ms) CNT:0x0 STATE:0x1e000000
NR          ON(0x47) (external gain: enable) (frame:1656 time:9ms) 0x5f0:0x0
0x5f4:0x0
SHARP       ON(0x1d) (YNR input filter: ON) (local ratio: OFF) 0x630:0x0
FEC         OFF(0x2) (frame:0 time:0ms) 0xc90:0x0
ORB         OFF(0x0)
Interrupt   Cnt:5300 ErrCnt:0
clk_ispp    500000000
aclk_ispp   500000000
hclk_ispp   250000000
```

Input: 输入源、输入格式、分辨率、当前帧号、实际帧间隔

Output: 输出对象、输出格式、分辨率、当前帧号、实际帧间隔

Interrupt: ispp内的处理中断，数据递增说明有数据进ispp，ErrCnt错误中断统计信息

clk_ispp: ispp 时钟频率

其他: ispp各个模块的开关状态

附录A CIS驱动V4L2-controls列表

CID	描述
V4L2_CID_VBLANK	Vertical blanking. The idle period after every frame during which no image data is produced. The unit of vertical blanking is a line. Every line has length of the image width plus horizontal blanking at the pixel rate defined by V4L2_CID_PIXEL_RATE control in the same sub-device.
V4L2_CID_HBLANK	Horizontal blanking. The idle period after every line of image data during which no image data is produced. The unit of horizontal blanking is pixels.
V4L2_CID_EXPOSURE	Determines the exposure time of the camera sensor. The exposure time is limited by the frame interval.
V4L2_CID_ANALOGUE_GAIN	Analogue gain is gain affecting all colour components in the pixel matrix. The gain operation is performed in the analogue domain before A/D conversion.
V4L2_CID_PIXEL_RATE	Pixel rate in the source pads of the subdev. This control is read-only and its unit is pixels / second. Ex mipi bus：pixel_rate = link_freq * 2 * nr_of_lanes / bits_per_sample
V4L2_CID_LINK_FREQ	Data bus frequency. Together with the media bus pixel code, bus type (clock cycles per sample), the data bus frequency defines the pixel rate (V4L2_CID_PIXEL_RATE) in the pixel array (or possibly elsewhere, if the device is not an image sensor). The frame rate can be calculated from the pixel clock, image width and height and horizontal and vertical blanking. While the pixel rate control may be defined elsewhere than in the subdev containing the pixel array, the frame rate cannot be obtained from that information. This is because only on the pixel array it can be assumed that the vertical and horizontal blanking information is exact: no other blanking is allowed in the pixel array. The selection of frame rate is performed by selecting the desired horizontal and vertical blanking. The unit of this control is Hz.

附录B MEDIA_BUS_FMT表

CIS sensor类型	Sensor输出format
Bayer RAW	MEDIA_BUS_FMT_SBGGR10_1X10 MEDIA_BUS_FMT_SRGGB10_1X10 MEDIA_BUS_FMT_SGBRG10_1X10 MEDIA_BUS_FMT_SGRBG10_1X10 MEDIA_BUS_FMT_SRGGB12_1X12 MEDIA_BUS_FMT_SBGGR12_1X12 MEDIA_BUS_FMT_SGBRG12_1X12 MEDIA_BUS_FMT_SGRBG12_1X12 MEDIA_BUS_FMT_SRGGB8_1X8 MEDIA_BUS_FMT_SBGGR8_1X8 MEDIA_BUS_FMT_SGBRG8_1X8 MEDIA_BUS_FMT_SGRBG8_1X8
YUV	MEDIA_BUS_FMT_YUYV8_2X8 MEDIA_BUS_FMT_YVYU8_2X8 MEDIA_BUS_FMT_UYVY8_2X8 MEDIA_BUS_FMT_VYUY8_2X8 MEDIA_BUS_FMT_YUYV10_2X10 MEDIA_BUS_FMT_YVYU10_2X10 MEDIA_BUS_FMT_UYVY10_2X10 MEDIA_BUS_FMT_VYUY10_2X10 MEDIA_BUS_FMT_YUYV12_2X12 MEDIA_BUS_FMT_YVYU12_2X12 MEDIA_BUS_FMT_UYVY12_2X12 MEDIA_BUS_FMT_VYUY12_2X12
Only Y(黑白)即raw bw sensor	MEDIA_BUS_FMT_Y8_1X8 MEDIA_BUS_FMT_Y10_1X10 MEDIA_BUS_FMT_Y12_1X12

附录C CIS参考驱动列表

CIS 数据接口	CIS 输出数据类型	Frame/Field	参考驱动
MIPI	Bayer RAW	frame	0.3M ov7750.c gc0403.c 1.2M ov9750.c jx-h65.c 2M ov2685.c ov2680.c ov2735.c gc2385.c gc2355.c gc2053.c sc2239.c sc210iot.c 4M gc4c33.c 5M ov5695.c ov5648.c ov5670.c gc5024.c gc5025.c gc5035.c 8M ov8858.c imx378.c imx317.c imx219.c gc8034.c 13M ov13850.c imx258.c

CIS 数据接口	CIS 输出数据类型	Frame/Field	参考驱动
MIPI	Bayer raw hdr	frame	2M imx307.c imx327.c gc2093.c ov02k10 ov2718.c sc200ai.c sc2310.c jx-f37.c 4M ov4689.c os04a10.c imx347.c sc4238.c 5M imx335.c 8M imx334.c imx415.c
MIPI	YUV	frame	2M gc2145.c
MIPI	RAW BW	frame	0.3M ov7251.c 1M ov9281.c 1.3M sc132gs.c
MIPI	YUV	field	tc35874x.c
ITU.BT601	Bayer RAW		2M imx323.c ar0230.c

CIS 数据接口	CIS 输出数据类型	Frame/Field	参考驱动
ITU.BT601	YUV		0.3M gc0329.c gc0312.c gc032a.c 2M gc2145.c gc2155.c gc2035.c bf3925.c
ITU.BT601	RAW BW		
ITU.BT656	Bayer RAW		2M imx323(可支持)

附录D VCM driver ic参考驱动列表

参考驱动
vm149c.c
dw9714.c
fp5510.c

附录E Flash light driver ic参考驱动列表

参考驱动
sgm3784.c
leds-rgb13h.c(GPIO控制)