密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKISP_Driver_User_Manual

（ISP 部）

文件状态： [] 正在修改 [√] 正式发布	当前版本：	v1.2
	作　　者：	ISP 部
	完成日期：	2019-03-18
	审　　核：	邓达龙
	完成日期：	2019-5-8

福州瑞芯微电子股份有限公司

Fuzhou Rockchips Electronics Co . , Ltd

版 本 历 史

版本号	作者	修改日期	修改说明	审核	备注
V1.0	胡克俊	2018-11-13	发布初版		
V1.1	胡克俊	2019-03-18	增加模组/OTP 信息说明；增加 VCM 驱动的说明		
V1.2	胡克俊 邓达龙 陈泽发 蔡艺伟	2019-5-8	1.RK1608 适配说明 2.增加 CIS、VCM 驱动移植步骤说明； 3.奇偶场输出支持说明		
	蔡艺伟	2019-5-30	增加闪光灯说明		

目 录

1 文档适用说明

1.1 适用平台及系统

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

1.2 适用驱动版本

驱动类型	版本号
rkisp driver	v0.1.2
RK1608 AP driver	v0.1.1

2 Camera 软件驱动目录说明

```
Linux Kernel-4.4：|
   |-- arch/arm64/boot/dts/rockchip       DTS 配置文件
   |-- drivers/phy/rockchip/
      |-- phy-rockchip-mipi-rx.c          mipi dphy 驱动
   |-- drivers/media|
      |-- platform/rockchip/isp1          rkisp1 isp 驱动
         |-- capture.c                    包含 mp/sp 的配置及 vb2，帧中断处理
         |-- dev.c                        包含 probe、异步注册、clock、pipeline、
                                          iommu 及 media/v4l2 framework
         |-- isp_params.c                 3A 相关参数设置
```

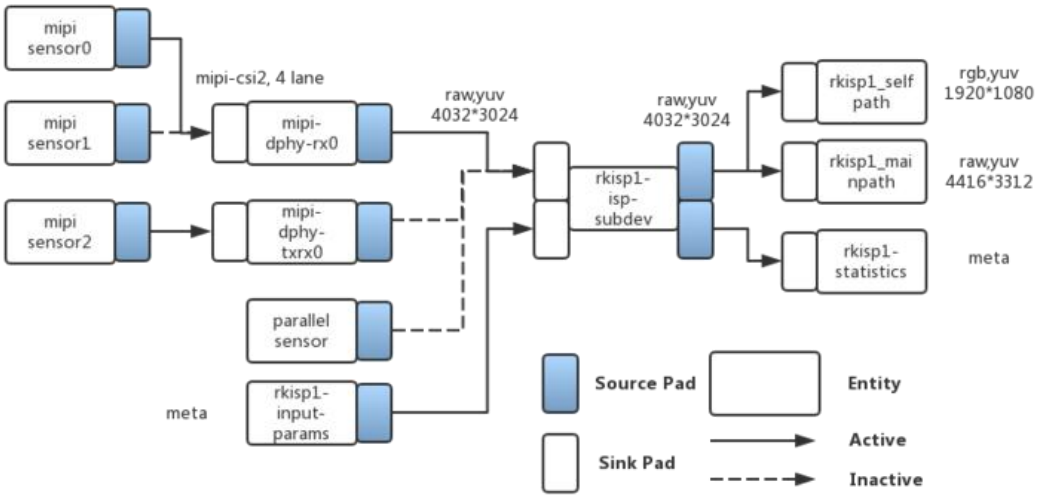
|-- isp_stats.c	3A 相关统计
|-- regs.c	寄存器相关的读写操作
|-- rkisp1.c	对应 isp_sd entity 节点，包含从 mipi 接收数据，并有 crop 功能
|-- i2c/	
|-- ov13850.c	CIS(cmos image sensor)驱动
|-- vm149c.c	VCM driver ic 驱动
|-- spi/	rk1608 ap driver 驱动
|-- rk1608.c	注册 rk1608 spi 设备
|-- rk1608_dev.c	注册/dev/rk_preisp misc 设备
|-- rk1608_dphy.c	注册 v412 media 节点，与 rk1608 和 AP 端交互

3 rkisp1 isp 驱动

3.1 框架简要说明

RKISP 驱动主要是依据 v412 / 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 RKISP1 驱动的拓扑结构。



名称	类型	描述
rkisp1_mainpath	v4l2_vdev, capture	Format: YUV, RAW Bayer; Support: Crop
rkisp1_selfpath	v4l2_vdev, capture	Format: YUV, RGB; Support: Crop
rkisp1-isp-subdev	v4l2_subdev	Internal isp blocks; Support: source/sink pad crop. The format on sink pad should be equal to sensor input format, the size should be equal/less than 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.
rockchip-sy-mipi	v4l2_subdev	MIPI-DPHY Configure.

-dphy		
rkisp1-statistics	v4l2_vdev, capture	Provide Image color Statistics information.
rkisp1-input-params	v4l2_vdev, output	Accept params for AWB, BLC..... Image enhancement blocks.

4 CIS(cmos image sensor)驱动

4.1 驱动版本号获取方式

- 从 kernel 启动 log 中获取

  rkisp1 ff910000.rkisp1: rkisp1 driver version: v00.01.02

- 由以下命令获取

cat /sys/module/video_rkisp1/parameters/version

4.2 CIS 设备注册(DTS)

Rkisp1 的 DTS 节点在 kernel 源码中有文档说明，路径如下：Documentation/devicetree/bindings/media/rockchip-isp1.txt。

mipi dphy 驱动节点 kernel 源码中有文档说明，路径如下：Documentation/devicetree/bindings/media/rockchip-mipi-dphy.txt

4.2.1 MIPI CIS 注册

下面以 rk3399 isp0 和 ov13850 为例进行说明。

```
ov13850: ov13850@10 {
        compatible = "ovti,ov13850"; // 需要与驱动中的匹配字符串一致
        status = "okay";
        reg = <0x10>; // sensor I2C 设备地址
        clocks = <&cru SCLK_CIF_OUT>; // sensor clickin 配置
        clock-names = "xvclk";
        reset-gpios = <&gpio2 10 GPIO_ACTIVE_HIGH>;
        // reset 管脚分配及有效电平
        pwdn-gpios = <&gpio1 4 GPIO_ACTIVE_HIGH>;
        // power 管脚分配及有效电平
        pinctrl-names = "rockchip,camera_default";
        pinctrl-0 = <&cif_clkout>; // pinctl 设置
        rockchip,camera-module-index = <0>; // 模组编号，该编号不要重复
        rockchip,camera-module-facing = "back"; // 模组朝向，有"back"和"front"
        rockchip,camera-module-name = "CMK-CT0116"; // 模组名
```

```
        rockchip,camera-module-lens-name = "Largan-50013A1"; // lens 名
        // 模组名和 lens 名被用来和 IQ xml 文件做匹配
        lens-focus = <&vm149c>; // vcm 驱动设置，支持 AF 时需要有这个设置
        port {
            ucam_out0: endpoint {
                remote-endpoint = <&mipi_in_ucam0>;
                // mipi dphy 端的 port 名
                data-lanes = <1 2>;
                // mipi lane 数，1lane 为 <1>, 4lane 为 <1 2 3 4>
            };
        };
    };
&mipi_dphy_rx0 {
    status = "okay";
    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;
            mipi_in_ucam0: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&ucam_out0>;
                // sensor 端的 port 名
                data-lanes = <1 2>;
```

```
                // mipi lane 数，1lane 为 <1>, 4lane 为 <1 2 3 4>
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;
            dphy_rx0_out: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&isp0_mipi_in>;
                // isp 端的 port 名
            };
        };
    };
};
&rkisp1_0 {
    status = "okay";
    port {
        #address-cells = <1>;
        #size-cells = <0>;
        isp0_mipi_in: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&dphy_rx0_out>;
            // mipi dphy 端的 port 名
        };
    };
```

```
};
&isp0_mmu {
    status = "okay"; // isp 驱动使用了 iommu，所以 isp iommu 也需要打开
};
```

4.2.2 DVP CIS 注册

以 rk3326 isp 和 gc0312/gc2145 为例进行说明。

```
&i2c2 {
    status = "okay";
    gc0312@21 {
        status = "okay";
        compatible = "galaxycore,gc0312"; // 需要与驱动中的匹配字符串一致
        reg = <0x21>; // sensor I2C 设备地址
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clkout_m0>; // pinctl 设置
        clocks = <&cru SCLK_CIF_OUT>; // sensor clickin 配置
        clock-names = "xvclk";
        avdd-supply = <&vcc2v8_dvp>; // sensor 电源配置
        dovdd-supply = <&vcc1v8_dvp>;
        dvdd-supply = <&vcc1v8_dvp>;
        pwdn-gpios = <&gpio2 14 GPIO_ACTIVE_HIGH>;
        // power 管脚分配及有效电平
        rockchip,camera-module-index = <1>; // 模组编号，该编号不要重复
        rockchip,camera-module-facing = "front"; // 模组朝向，有"back"和"front"
        rockchip,camera-module-name = "CameraKing"; // 模组名
        rockchip,camera-module-lens-name = "Largan"; // lens 名
```

```
        port {
            gc0312_out: endpoint {
                remote-endpoint = <&dvp_in_fcam>;// isp 端的 port 名
            };
        };
    };
    gc2145@3c {
        status = "okay";
        compatible = "galaxycore,gc2145"; // 需要与驱动中的匹配字符串一致
        reg = <0x3c>; // sensor I2C 设备地址
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clkout_m0>; // pinctl 设置
        clocks = <&cru SCLK_CIF_OUT>; // sensor clickin 配置
        clock-names = "xvclk";
        avdd-supply = <&vcc2v8_dvp>; // sensor 电源配置
        dovdd-supply = <&vcc1v8_dvp>;
        dvdd-supply = <&vcc1v8_dvp>;
        pwdn-gpios = <&gpio2 13 GPIO_ACTIVE_HIGH>;
        // power 管脚分配及有效电平
        rockchip,camera-module-index = <0>; // 模组编号，该编号不要重复
        rockchip,camera-module-facing = "back"; // 模组朝向，有"back"和"front"
        rockchip,camera-module-name = "CameraKing"; // 模组名
        rockchip,camera-module-lens-name = "Largan"; // lens 名
        port {
            gc2145_out: endpoint {
                remote-endpoint = <&dvp_in_bcam>;// isp 端的 port 名
```

```
            };
        };
    };
};
&isp_mmu {
    status = "okay";
};
&rkisp1 {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&cif_clkout_m0 &dvp_d0d1_m0 &dvp_d2d9_m0 &dvp_d10d11_m0>;
    // pinctl 设置，增加 dvp pin 脚相关配置
    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;
            dvp_in_fcam: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&gc0312_out>; // sensor 端的 port 名
            };
            dvp_in_bcam: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&gc2145_out>; // sensor 端的 port 名
```

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

4.3 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 v412 subdev 的方式实现与 host driver 之间的交互。

4.3.1 数据类型简要说明

4.3.1.1 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 ov13850_of_match[] = {
  { .compatible = "ovti,ov13850" },
  {},
};
MODULE_DEVICE_TABLE(of, ov13850_of_match);
#endif

static const struct i2c_device_id ov13850_match_id[] = {
  { "ovti,ov13850", 0 },
  { },
};

static struct i2c_driver ov13850_i2c_driver = {
```

```
    .driver = {
        .name = "ov13850",
        .pm = &ov13850_pm_ops,
        .of_match_table = of_match_ptr(ov13850_of_match),
    },
    .probe      = &ov13850_probe,
    .remove     = &ov13850_remove,
    .id_table   = ov13850_match_id,
};

static int __init sensor_mod_init(void)
{
    return i2c_add_driver(&ov13850_i2c_driver);
}

static void __exit sensor_mod_exit(void)
{
    i2c_del_driver(&ov13850_i2c_driver);
}

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

4.3.1.2 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 ov5695_subdev_ops = {
  .core   = &ov5695_core_ops,
  .video  = &ov5695_video_ops,
  .pad    = &ov5695_pad_ops,
};
```

4.3.1.3 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 ov13850_core_ops = {
  .ioctl = ov13850_ioctl,
#ifdef CONFIG_COMPAT
  .compat_ioctl32 = ov13850_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;

4.3.1.4 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 (*s_frame_interval)(struct v4l2_subdev *sd,
              struct v4l2_subdev_frame_interval *interval);
  ......
};
```

[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

[示例]

```
static const struct v4l2_subdev_video_ops ov13850_video_ops = {
  .s_stream = ov13850_s_stream,
  .g_frame_interval = ov13850_g_frame_interval,
};
```

4.3.1.5 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);
    ......

};
```

[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

[示例]

```
static const struct v4l2_subdev_pad_ops ov13850_pad_ops = {
  .enum_mbus_code = ov13850_enum_mbus_code,
  .enum_frame_size = ov13850_enum_frame_sizes,
  .get_fmt = ov13850_get_fmt,
  .set_fmt = ov13850_set_fmt,
};
```

4.3.1.6 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 relevant for volatile (and usually read-only) controls .
.try_ctrl	test whether the control's value is valid.
.s_ctrl	actually set the new control value.

[示例]

```
static const struct v4l2_ctrl_ops ov13850_ctrl_ops = {
    .s_ctrl = ov13850_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

4.3.1.7 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 width;
    u32 height;
    struct v4l2_fract max_fps;
    u32 hts_def;
```

```
    u32 vts_def;
    u32 exp_def;
    const struct regval *reg_list;
};
```

[关键成员]

成员名称	描述
.width	有效图像宽度
.height	有效图像高度
.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

[示例]

```
static const struct ov13850_mode supported_modes[] = {
    {
        .width = 2112,
        .height = 1568,
        .max_fps = {
            .numerator = 10000,
            .denominator = 300000,
        },
        .exp_def = 0x0600,
        .hts_def = 0x12c0,
        .vts_def = 0x0680,
        .reg_list = ov13850_2112x1568_regs,
```

```
    },{
        .width = 4224,
        .height = 3136,
        .max_fps = {
            .numerator = 20000,
            .denominator = 150000,
        },
        .exp_def = 0x0600,
        .hts_def = 0x12c0,
        .vts_def = 0x0d00,
        .reg_list = ov13850_4224x3136_regs,
    },
};
```

4.3.1.8 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]

4.3.1.9 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]

4.3.1.10 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]

4.3.1.11 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]

4.3.1.12 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]

4.3.1.13 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]

4.3.1.14 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]

4.3.1.15 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]

4.3.1.16 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]

4.3.2 API 简要说明

4.3.2.1 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

4.3.2.2xxxx_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

4.3.2.3xxxx_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

4.3.2.4xxxx_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

4.3.2.5xxxx_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

4.3.2.6xxxx_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

4.3.2.7xxxx_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

4.3.2.8xxxx_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

4.3.2.9xxxx_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

4.3.3 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:

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 camera_engine 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：

.ioctl .compat_ioctl32

该回调主要实现 RK 私有控制命令，涉及：

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；

2.2 参考 struct v4l2_subdev_video_ops 说明实现其回调函数，主要实现以下 2 个回调函数：

int (*s_stream)(struct v4l2_subdev *sd, int enable);
int (*g_frame_interval)(struct v4l2_subdev *sd, struct v4l2_subdev_frame_interval *interval);

2.3 参考 struct v4l2_subdev_pad_ops 说明实现其回调函数，主要实现以下 4 个回调

函数：

.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

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

.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；

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：

line_time = HTS * PIXEL_RATE;
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

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 4.1 MIPI Sensor 注册与 4.2 DVP Sensor 注册中关于 Port 信息的描述。

4. CIS 参考驱动列表

5 VCM 驱动

5.1 VCM 设备注册(DTS)

RK VCM 驱动私有参数说明：

名称	定义
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
        ......
};
```

5.2 VCM 驱动说明

5.2.1 数据类型简要说明

5.2.1.1 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);
```

5.2.1.2 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

5.2.1.3 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 relevant for 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

5.2.2 API 简要说明

5.2.2.1xxxx_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

5.2.2.2xxxx_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

5.2.2.3xxxx_ioctl/xxxx_compat_ioctl32

[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

5.2.3 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：

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：

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]。

6 Rk1608 AP 驱动

6.1 驱动版本号获取方式

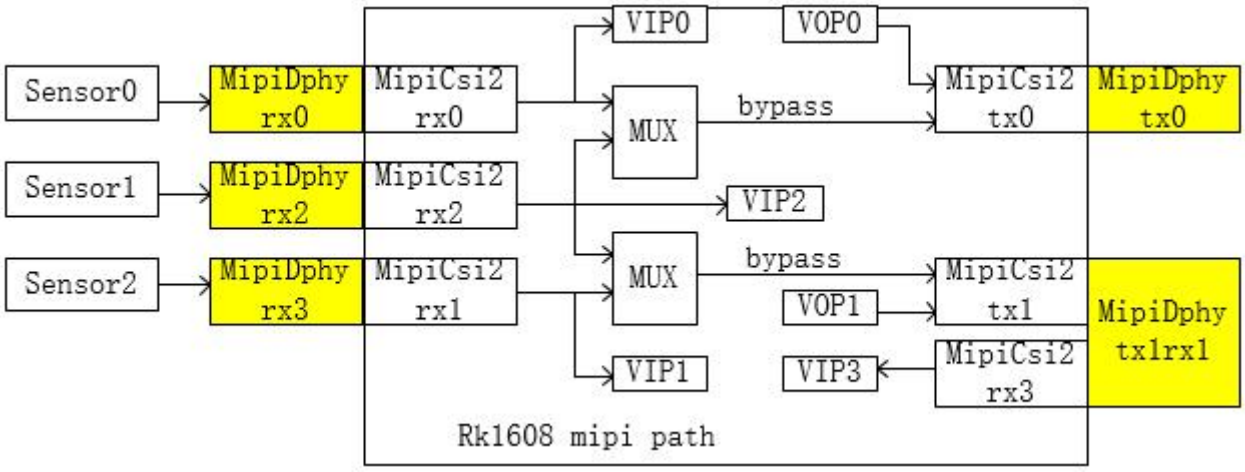
通过以下命令可以查询：

echo v > /dev/rk_preisp

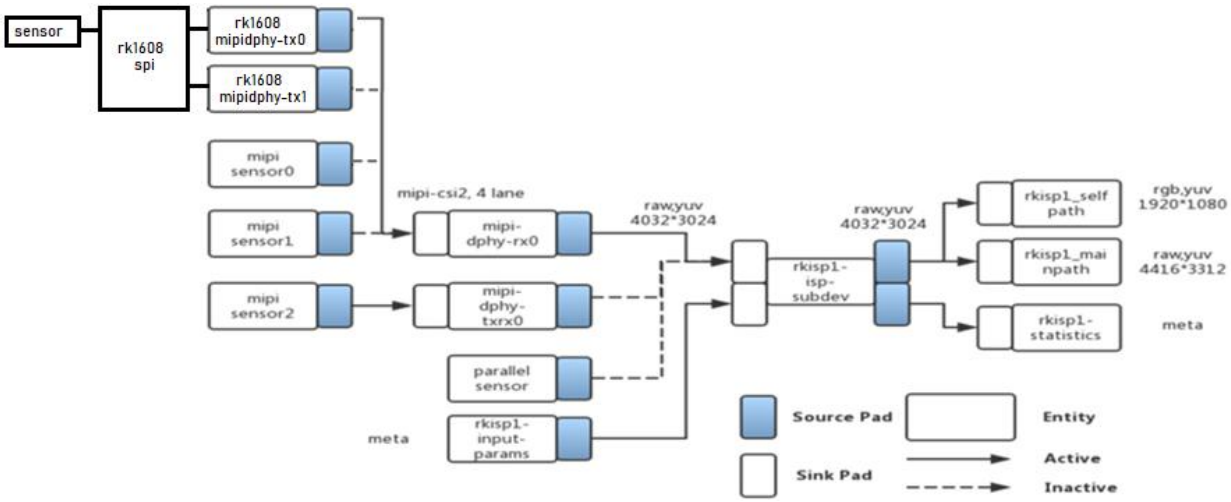
6.2 框架简要说明

Rk1608 AP driver 主要控制 rk1608 上下电，加载 rk1608 firmware，通过 spi 总线与 rk1608 通信，控制 rk1608 采集 sensor 数据、算法运算、数据输出等。

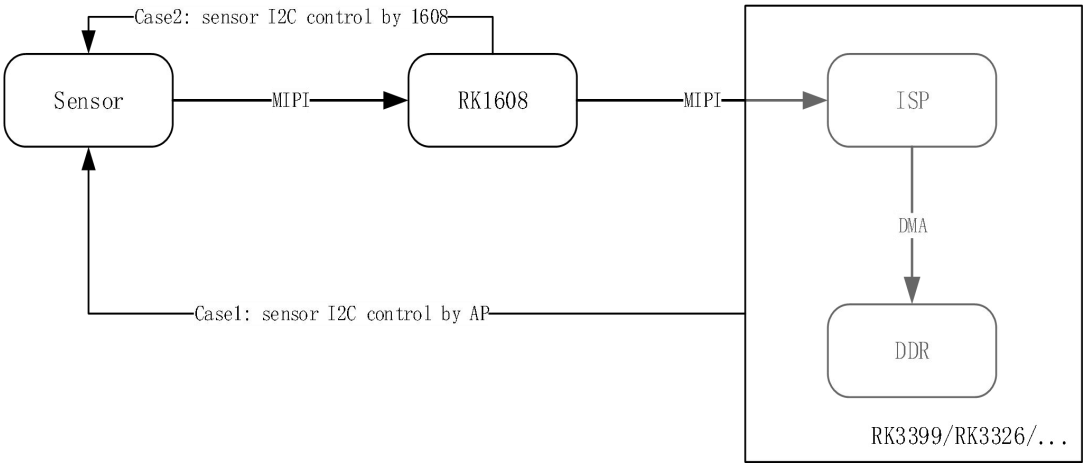
Rk1608 内部 mipi 通路连接图如下：



下面描述了 rk1608 AP driver 与 rkisp1 的拓扑结构，rk1608 AP driver 整体可看作一个 soc sensor。



sensor i2c 可以被 AP 控制，也可以被 1608 直接控制。

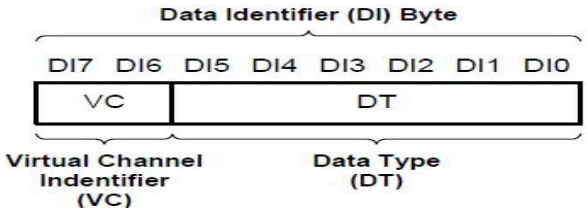


6.3 Rk1608 AP 设备注册(DTS)

以 rk3326-evb-lp3-v10-rk1608-linux.dts 为例

```
#define LINK_FREQ        400000000

    mipidphy0: mipidphy0 {//rk1608 mipi dphy

        compatible =  "rockchip,rk1608-dphy";

        status = "okay";

        rockchip,grf = <&grf>;

        id = <0>;//camera id

        cam_nums = <1>;//connect to rk1608’  camera nums

        data_type = <0x2c>;//mipi data type

        /*in_mipi: rk1608 in mipi phy index

         *out_mipi: rk1608 out mipi phy index

         *注意：此处走 bypass mode，vip/vop mode 需要更改 rk1608 固件

         *如果硬件连接 rk1608 mipi dphy rx3，软件此处需要配 in_mipi=<1>

         *bypass mode:

         *rx0/rx2 进，tx0 出

         *rx2/rx3 进，tx1 出*/

        in_mipi = <2>;

        out_mipi = <1>;

        mipi_lane = <2>;//mipi lane num

        //rk1608’  i2c bus index, use for rk1608<->i2c<->sensor

        sensor_i2c_bus = <1>;

        sensor_i2c_addr = <0x78>;

        sensor-name = "OPN8008";//sensor name

        field = <1>;//used interlacing type (from enum v4l2_field)

        colorspace = <8>;//colorspace of the data (from enum v4l2_colorspace)
```

```
//data format code (from enum v4l2_mbus_pixelcode)
code = <MEDIA_BUS_FMT_SRGGB12_1X12>;
width = <328>;//image width
height= <744>;//image height
htotal = <650>;//horizontal total
vtotal = <900>;//vertical total
/* rk1608 mipi out freqs
 * mipi clk: htotal * vtotal * max_fps * data_bits / lane */
link-freqs = /bits/ 64 <LINK_FREQ>;
/*input ch0 info:<width height data_id decode_format flag>
```




```
 * data_id
 * decode_format 0x2c(raw12)
 * flag 1(picture channel) 0(normal channel)*/
inch0-info = <328 744 0x2c 0x2c 1>;
outch0-info = <328 744 0x2c 0x2c 1>;//out ch0 info
rockchip,camera-module-index = <0>;//模组编号，该编号不要重复
rockchip,camera-module-facing = "back";//模组朝向，有"back"和"front"
rockchip,camera-module-name = "TongJu";//模组名
rockchip,camera-module-lens-name = "CHT842-MD";//lens 名
ports {
    #address-cells = <1>;
    #size-cells = <0>;
    port@0 {
        rk1608_dphy0_in: endpoint {
```

```
                    remote-endpoint = <&rk1608_out0>;//rk1608 mipidphy 输入端
                };
            };
            port@1 {
                rk1608_dphy_out: endpoint {
                    remote-endpoint = <&mipi_in_ucam>;//rk1608 mipidphy 输出端
                    clock-lanes = <0>;
                    data-lanes = <1 2>;
                    clock-noncontinuous;
                    link-frequencies =
                        /bits/ 64 <LINK_FREQ>;
                };
            };
        };
    };
&i2c2 {
…
    pisp_dmy: pisp_dmy@1 {
        /*dummy sensor for preisp, sensor 直接与 rk1608 通信使用此驱动
         *sensor 与 AP 端通信使用对应 sensor 驱动*/
        compatible = "pisp_dmy";
        status = "okay";
        reg = <0x1>;
        clocks = <&cru SCLK_CIF_OUT>;
        clock-names = "xvclk";
        pwdn-gpios = <&gpio2 14 GPIO_ACTIVE_HIGH>;
```

```
        rockchip,camera-module-index = <0>;
        rockchip,camera-module-facing = "back";
        rockchip,camera-module-name = "TongJu";
        rockchip,camera-module-lens-name = "CHT842-MD";
        port {
            cam_out: endpoint {
                remote-endpoint = <&rk1608_in0>;//sensor 输出端口
                data-lanes = <1 2>;
            };
        };
    };
};
&spi1 {
…
    spi_rk1608@00 {//rk1608 spi 设备
        compatible =  "rockchip,rk1608";
        status = "okay";
        reg = <0>;
        spi-max-frequency = <16000000>;
        spi-min-frequency = <16000000>;
        clocks = <&cru SCLK_CIF_OUT>;
        clock-names = "mclk";
        firmware-names = "rk1608.rkl";//rk1608 固件名
        reset-gpios = <&gpio3 RK_PC4 GPIO_ACTIVE_HIGH>;
        irq-gpios = <&gpio3 RK_PC5 GPIO_ACTIVE_HIGH>;
        pinctrl-names = "default";
```

```
pinctrl-0 = <&preisp_irq_gpios &preisp_sleep_gpios
    &preisp_reset_gpios>;
/* regulator config */
vdd-core-regulator = "vdd_preisp";
vdd-core-microvolt = <1150000>;
ports {
    #address-cells = <1>;
    #size-cells = <0>;
    port@0 {
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0>;
        rk1608_out0: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&rk1608_dphy0_in>;//rk1608 输出端口
        };
    };
    port@1 {
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <1>;
        rk1608_in0: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&cam_out>;//rk1608 输入端口
        };
    };
```

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

有些算法不需要 1608 直接接 sensor,此时可以只配置 spi_rk1608 节点， spi_rk1608 节点中的 port 信息不需要配置。应用通过 rk1608_dev.c 中提供 ioctl 接口控制 1608 完成算法应用。

另外，上述 mipidphy dts 仅能支持一个输出格式到 AP，在下面的这个提交之后，可以支持多个输出格式到 AP，这个提交目前暂时还未合并。

media: spi: rk1608: support multiple output format to isp

Change-Id: Icc9c14891d6f7494a6d6cc4752dabcf07278d708

Signed-off-by: Hu Kejun <william.hu@rock-chips.com>

这个提交对应的 dts 需要做一些变动。

```
        mipidphy0: mipidphy0 {
                compatible =  "rockchip,rk1608-dphy";
                status = "okay";
                rockchip,grf = <&grf>;
                id = <0>;
                cam_nums = <1>;
                in_mipi = <0>;
                out_mipi = <0>;
                link-freqs = /bits/ 64 <LINK_FREQ>;
                sensor_i2c_bus = <6>;
                sensor_i2c_addr = <0x1A>;
                sensor-name = "IMX317";
                rockchip,camera-module-index = <0>;
                rockchip,camera-module-facing = "back";
                rockchip,camera-module-name = "PREISP";
```

```
rockchip,camera-module-lens-name = "PREISP";
format-config-0 { // 1608 到 AP 的输出格式一
        data_type = <0x2b>;
        mipi_lane = <4>;
        field = <1>;
        colorspace = <8>;
        code = <MEDIA_BUS_FMT_SRGGB10_1X10>;
        width = <1932>;
        height= <1094>;
        htotal = <2500>;
        vtotal = <1500>;
        inch0-info = <1932 1094 0x2b 0x2b 1>;
        outch0-info = <1932 1094 0x2b 0x2b 1>;
};

format-config-1 { // 1608 到 AP 的输出格式二
        data_type = <0x2b>;
        mipi_lane = <4>;
        field = <1>;
        colorspace = <8>;
        code = <MEDIA_BUS_FMT_SRGGB10_1X10>;
        width = <3864>;
        height= <2174>;
        htotal = <4200>;
        vtotal = <2400>;
        inch0-info = <3864 2174 0x2b 0x2b 1>;
```

```
                outch0-info = <3864 2174 0x2b 0x2b 1>;
        };
        ports {
                #address-cells = <1>;
                #size-cells = <0>;
                port@0 {
                        rk1608_dphy0_in: endpoint {
                                remote-endpoint = <&rk1608_out0>;
                        };
                };
                port@1 {
                        rk1608_dphy_out: endpoint {
                                remote-endpoint = <&mipi_in_cam>;
                                clock-lanes = <0>;
                                data-lanes = <1 2 3 4>;
                                clock-noncontinuous;
                                link-frequencies =
                                        /bits/ 64 <LINK_FREQ>;
                        };
                };
        };
    };
```

对于 rk1608 hdr 模式，dts 需要增加 2 个 AE 同步引脚配置：

rk1608(gpio1_A2)->AP gpio　　rk1608 帧开始信号通知 AP 端

AP gpio-> rk1608(gpio1_A3)　　AP 端配置完曝光后通知 rk1608

```
&spi1 {
```

```
…
    spi_rk1608@00 {//rk1608 spi 设备
    ...
        //AP gpio-> rk1608(gpio1_A3)
        aesync-gpio = <&gpio2 RK_PA1 GPIO_ACTIVE_LOW>//AP 端 gpio
    ...
    };
};
&rkisp1 {
...
    //rk1608(gpio1_A2)->AP gpio
    Vsirq-gpios = <&gpio RK_PA0 GPIO_ACTIVE_LOW>;//AP 端 gpio
...
}
```

6.4 Rk1608 AP 驱动说明

Rk1608 驱动采用 spi 实现 AP 端与 rk1608 端消息通信，mipi sensor 可以通过 i2c 挂接到 rk1608 或挂接到 AP 端进行通信管理。

6.4.1 数据类型简要说明

6.4.1.1 struct spi_driver

[说明]

定义 spi 设备驱动信息

[定义]

```
struct spi_driver {
  const struct spi_device_id *id_table;
  int (*probe)(struct spi_device *);
  int (*remove)(struct spi_device *);
  void (*shutdown)(struct spi_device *)
  struct device_driver driver;
};
```

[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 SPI 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 spi_device_id rk1608_id[] = {
  { "rk1608", 0 },
  {  }
};
MODULE_DEVICE_TABLE(spi, rk1608_id);
static const struct of_device_id rk1608_of_match[] = {
```

```
    { .compatible = "rockchip,rk1608" },
    { }
};
MODULE_DEVICE_TABLE(of,rk1608_of_match);
static struct spi_driver rk1608_driver = {
  .driver = {
      .name = "rk1608",
      .of_match_table = of_match_ptr(rk1608_of_match),
  },
  .probe = &rk1608_probe,
  .remove = &rk1608_remove,
  .id_table = rk1608_id,
};
module_i2c_driver(vm149c_i2c_driver);
static int __init preisp_mod_init(void){
  return spi_register_driver(&rk1608_driver);
}
static int __exit preisp_mod_exit(void){
  return spi_unregister_driver(&rk1608_driver);
}
late_initcall(preisp_mod_init);
module_exit(preisp_mod_exit);
```

6.4.1.2 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 rk1608_core_ops = {
  .s_power = rk1608_sensor_power,
  .ioctl = rk1608_ioctl,
};
```

目前使用了如下的私有 ioctl 获取 sensor 信息、曝光控制。

私有 ioctl	描述

PREISP_CMD_SAVE_HDRAE_PARAM	向 1608 传递当前的 awb gain 信息和 lsc 补偿信息； 由 isp 驱动直接调用； 详细参考 struct preisp_hdrae_para_s；
PREISP_CMD_SET_HDRAE_EXP	进行 sensor HDR 模式的曝光设置； 详细参考 struct preisp_hdrae_exp_s；
RKMODULE_GET_MODULE_INFO	获取模组信息，实际返回对应 sensor 的模组信息； 详细参考 struct rkmodule_inf；

6.4.1.3 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 (*s_frame_interval)(struct v4l2_subdev *sd,
              struct v4l2_subdev_frame_interval *interval);
  ......
};
```

[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

[示例]

```
static const struct v4l2_subdev_video_ops rk1608_video_ops = {
  .s_stream = rk1608_s_stream,
  .g_frame_interval = rk1608_g_frame_interval,
};
```

6.4.1.4 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);
    ......

};
```

[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

[示例]

```
static const struct v4l2_subdev_pad_ops rk1608_subdev_pad_ops = {
  .enum_mbus_code = rk1608_enum_mbus_code,
  .enum_frame_size = rk1608_enum_frame_sizes,
  .get_fmt = rk1608_get_fmt,
  .set_fmt = rk1608_set_fmt,
};
```

6.4.1.5 struct file_operations

[说明]

File operations

[定义]

```
struct file_operations {
  ......
```

```
  struct module *owner;
  int (*open)(struct inode *, struct file *);
  int (*release)(struct inode *, struct file *);
  ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *);
  unsigned int (*poll)(struct file *, struct poll_table_struct *);
  long (*unlocked_ioctl)(struct file *, unsigned int, unsigned long);
  long (*conpat_ioctl)(struct file *, unsigned int, unsigned long);
    ……


  };
```

[示例]

```
static const struct file_operations rk1608_fops = {
    .owner = THIS_MODULE,
    .open = rk1608_dev_open,
    .release = rk1608_dev_release,
    .write = rk1608_dev_write,
    .poll = rk1608_dev_poll,
    .unlocked_ioctl = rk1608_dev_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = rk1608_compat_ioctl,
#endif
};
```

Rk1608 spi 设备文件节点操作，上层可直接访问此节点控制 rk1608。

6.4.1.6 struct preisp_hdrae_para_s

[说明]

Awb and lsc parameter for preisp.

[定义]

```
struct preisp_hdrae_para_s {
        unsigned short r_gain;
        unsigned short b_gain;
        unsigned short gr_gain;
        unsigned short gb_gain;
        int lsc_table[PREISP_LSCTBL_SIZE];
};
```

[关键成员]

成员名称	描述
r_gain	awb r gain
b_gain	awb b gain
gr_gain	awb gr gain
gb_gain	awb gb gain
lsc_table	lsc table

[示例]

6.4.1.7 struct preisp_hdrae_exp_s

[说明]

Hdr ae 曝光设置.

[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;
};
```

[关键成员]

成员名称	描述
long_exp_reg	HDR 长曝光时间寄存器值;
long_gain_reg	HDR 长曝光 gain 寄存器值;
middle_exp_reg	HDR 中曝光时间寄存器值;
middle_gain_reg	HDR 中曝光 gain 寄存器值;
short_exp_reg	HDR 短曝光时间寄存器值;
short_gain_reg	HDR 短曝光 gain 寄存器值;
long_exp_val	HDR 长曝光时间实际值，将上层传递的 float 值传给 1608;
long_gain_val	HDR 长曝光 gain 实际值，将上层传递的 float 值传给 1608;

middle_exp_val	HDR 中曝光时间实际值，将上层传递的 float 值传给 1608;
middle_gain_val	HDR 中曝光 gain 实际值，将上层传递的 float 值传给 1608;
short_exp_val	HDR 短曝光时间实际值，将上层传递的 float 值传给 1608;
short_gain_val	HDR 短曝光 gain 实际值，将上层传递的 float 值传给 1608;

[示例]

6.4.2 API 简要说明

如下 V4L2 API 用法与 sensor 一样，不再重复说明。

xxxx_set_fmt

xxxx_get_fmt

xxxx_enum_mbus_code

xxxx_enum_frame_sizes

xxxx_g_frame_interval

xxxx_s_stream

xxxx_set_ctrl

6.4.2.1 rk1608_dev_write

[描述]

通过/dev/rk_preisp 传输命令，可用于调试

[用法]

echo c > /dev/rk_preisp

receive a message from rk1608 -> AP message queue

echo f [fw_name] > /dev/rk_preisp

download firmware, there is no parameter and download preisp.rkl default

```
echo fw reg1 > /dev/rk_preisp

fast write reg1, make a interrupt of rk1608

echo fr > /dev/rk_preisp

fast read, read the value of reg2

echo log level > /dev/rk_preisp

set the rk1608 print level, the smaller of the value of number, the fewer of log

echo on > /dev/rk_preisp

power on, increase 1 on the count, and only execute when the count is 1.

echo off > /dev/rk_preisp

power off, decrease 1 from the count, execute only when the count is 0.

echo q > /dev/rk_preisp

rK1608 last operation state query

echo r addr [length] > /dev/rk_preisp

read data, output the data by kmsg

echo rate max [min] > /dev/rk_preisp

set the maximum speed and minimum speed of spi

echo s type,... > /dev/rk_preisp

send message to AP -> rk1608 message queue

echo w addr value,... > /dev/rk_preisp

write data

echo v > /dev/rk_preisp

inquire the version of driver
```

6.4.3 Bringup 步骤

1) 调试 1608 所接 sensor 时，可以先使用 1608 bypass 固件进行调试。1608 bypass 固件一般集成在 SDK 中。

使用 1608 bypass 固件调试时的大概步骤：

• 确认所接 sensor 驱动为 liner mode，1608 bypass 固件不支持 hdr mode;

• 确认 dts 文件中配置的 1608 固件为 1608 bypass 固件;

spi_rk1608 节点中的 firmware-names 可以指定固件名称

firmware-names = "fw_rk1608_bypass.rkl";

• 修改 dts 文件中 mipidphy 的配置;

code = <MEDIA_BUS_FMT_SRGGB10_1X10>; // sensor 输出图像格式

width = <1932>; // sensor 输出图像宽度

height= <1094>; // sensor 输出图像高度

htotal = <2500>; // 比 sensor 输出图像宽度大 400 左右

vtotal = <1500>; // 比 sensor 输出图像高度大 400 左右

inch0-info = <1932 1094 0x2b 0x2b 1>; // 按照 sensor 输出图像格式填写

outch0-info = <1932 1094 0x2b 0x2b 1>; // 按照 sensor 输出图像格式填写

link-freqs = /bits/ 64 <LINK_FREQ>; // 这里设置的 link-freqs 要比 sensor->1608 的 link-freqs 高才行，一般大 30%，也可以尝试最高的 link-freqs 750MHz。

2) 调试 1608 时，需要先确认 1608 固件加载成功。1608 固件是否加载成功，可确认 kernel log 中是否有下面的 log 来判断。

rk1608 spi32766.0: Download firmware success!

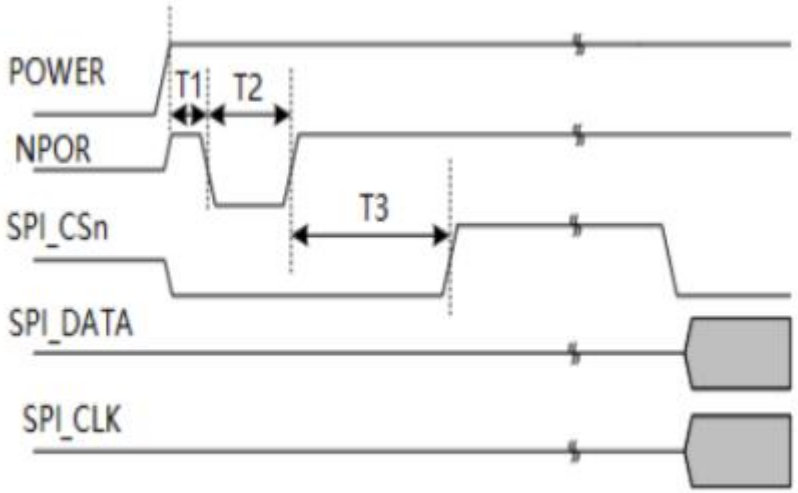
如果固件加载有问题，一般需要检查供给 1608 的 clock、电压、spi 信号还有 reset/spi-cs 引脚的时序。

1608 上电的时序要求如下：

1. Power up timing:

T1≥0, T2≥0, T3≥0, T4≥0

2. SPI timing after power up



T1>0, T2≥1ms, T3≥3ms

其中的 NPOR 即为 1608 reset 引脚。

2）调试 1608 hdr mode 时，dts 需要增加 2 个 AE 同步引脚配置，请参考 rk1608 dts 配置说明。Sensor 驱动需要实现 PREISP_CMD_SET_HDRAE_EXP ioctl，对 sensor hdr mode 配置曝光和 gain。1608 固件需要使用 hdr mode 固件。

7 FlashLight 驱动

7.1 FLASHLight 设备注册(DTS)

```
&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
        ...
    };
    ...
}
```

7.2 FLASHLight 驱动说明

7.2.1 数据类型简要说明

7.2.1.1 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);
```

7.2.1.2 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

7.2.1.3 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 relevant for 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,
    }
};
```

7.2.2 API 简要说明

7.2.2.1 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

7.2.2.2xxxx_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

7.2.2.3xxxx_ioctl/xxxx_compat_ioctl32

[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

7.2.3 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：

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：

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

8 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 /dev/media0
```

2） 修改 fmt/size

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

3） 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
```

4） 设置曝光、gain 等 control

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

9 FAQ

9.1 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
```

/sys/class/video4linux/video3/name:rkisp1_selfpath

/sys/class/video4linux/video4/name:rkisp1_mainpath

/sys/class/video4linux/video5/name:rkisp1-statistics

/sys/class/video4linux/video6/name:rkisp1-input-params

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

1） 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] rkisp1: 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。

9.2 如何抓取 isp 输出的 yuv 数据

参考命令如下，

```
media-ctl -d /dev/media0 --set-v4l2 '"ov5695 7-0036":0[fmt:SBGGR10_1X10/2592x1944]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":0[fmt:SBGGR10_1X10/2592x1944]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":0[crop:(0,0)/2592x1944]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":2[fmt:YUYV8_2X8/2592x1944]'
media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":2[crop:(0,0)/2592x1944]'
v4l2-ctl -d /dev/video4 \
--set-selection=target=crop,top=336,left=432,width=1920,height=1080 \
--set-fmt-video=width=1280,height=720,pixelformat=NV21 \
--stream-mmap=3 --stream-to=/tmp/mp.out --stream-count=20 --stream-poll
```

9.3 如何抓取 Sensor 输出的 Raw Bayer 原始数据

参考命令如下，

```
media-ctl -d /dev/media0 --set-v4l2 '"ov5695 7-0036":0[fmt:SBGGR10_1X10/2592x1944]'

media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":0[fmt:SBGGR10_1X10/2592x1944]'

media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":0[crop:(0,0)/2592x1944]'

media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":2[fmt:SBGGR10_1X10/2592x1944]'

media-ctl -d /dev/media0 --set-v4l2 '"rkisp1-isp-subdev":2[crop:(0,0)/2592x1944]'

v4l2-ctl -d /dev/video4 --set-ctrl 'exposure=1216,analogue_gain=10' \

--set-selection=target=crop,top=0,left=0,width=2592,height=1944 \

--set-fmt-video=width=2592,height=1944,pixelformat=SBGGR10 \

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

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

9.4 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。

9.5 如何支持奇偶场合成

Rkisp1 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)，Rkisp1 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，rkisp1 驱动以此来判断当前场的奇偶；

3. rkisp1 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”

rkisp1_selfpath 信息可以 media-ctl -p 查看:

```
- entity 3: rkisp1_selfpath (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video1
          pad0: Sink
                <- "rkisp1-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；

附录 A CIS 驱动 V4L2-controls 列表 1

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	ov13850.c ov8858.c ov7750.c ov5695.c ov5648.c ov4689.c ov2735.c ov2718.c ov2685.c ov2680.c imx327.c imx317.c imx258.c imx219.c gc8034.c gc5025.c gc2385.c gc2355.c jx-h65
MIPI	YUV	frame	gc2145.c
MIPI	RAW BW	frame	ov9281.c ov7251.c

			sc132gs.c
MIPI	YUV	field	tc35874x.c
ITU.BT601	Bayer RAW		imx323 ar0230.c
ITU.BT601	YUV		gc2145.c gc2155.c gc2035.c gc0329.c gc0312.c bf3925.c
ITU.BT601	RAW BW		
ITU.BT656	Bayer RAW		imx323(可支持)

附录 D VCM 参考驱动列表

参考驱动
vm149c.c
dw9714.c
fp5510.c

附录 E FLASH 参考驱动列表

参考驱动
sgm3784.c