RKISP_Driver_User_Manual

(ISP Department)

Status: [ ] Modifying [√] Released	Version:	V1.2
	Author:	ISP Department
	Date:	2019-03-18
	Auditor:	Dalong Deng
	Date:	2019-05-08

Fuzhou Rockchips Electronics Co . , Ltd

Revision History

Version No.	Author	Revision Date	Revision Description	Auditor	Remark
V1.0	Kejun Hu	2018-11-13	Initial Release		
V1.1	Kejun Hu	2019-03-18	1.Add module/OTP information instructions; 2.Add VCM driver instructions		
V1.2	Kejun Hu Dalong Deng Zefa Chen Yiwei Cai	2019-05-08	1.RK1608 adaptation instructions; 2.Add CIS and VCM driver porting steps instructions; 3. Parity field output support instructions;		

Table of content

1 Application platform

1.1 Applicable platforms and systems

Chipset platform	Kernel version	Support or not
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 Applicable driver version

Driver type	Version No.
rkisp driver	v0.1.2
RK1608 AP driver	v0.1.1

2 Camera software driver directory instructions

Linux Kernel-4.4：

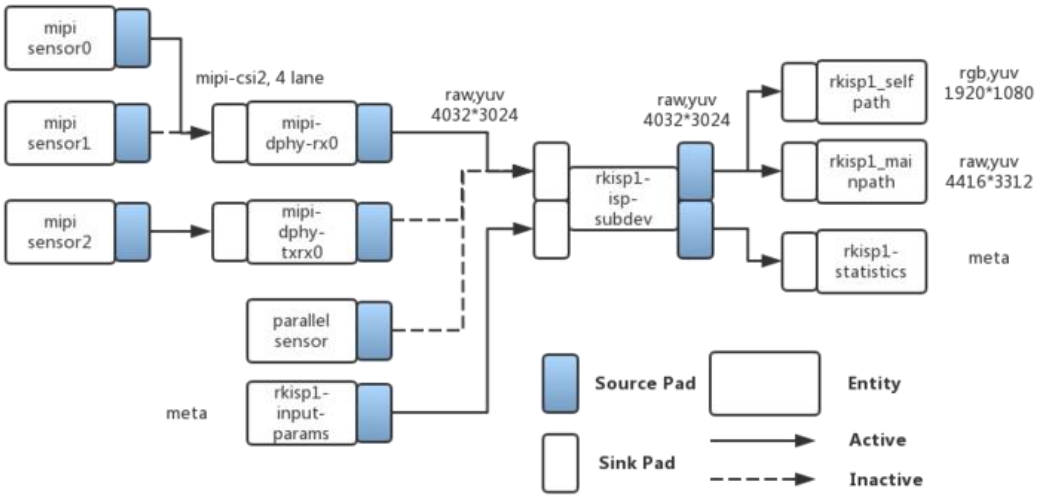
```
|-- arch/arm64/boot/dts/rockchip        //DTS configuration file
|-- drivers/phy/rockchip/
   |-- phy-rockchip-mipi-rx.c          //mipi dphy driver
|-- drivers/media|
  |-- platform/rockchip/isp1           // rkisp1 isp driver
     |-- capture.c                     //including mp/sp configurations and vb2, frame interrupt handling
     |-- dev.c                         //including probe, asynchronous registration, clock, pipeline, iommu and media/v4l2 framework
     |-- isp_params.c                  //3A related parameters setting
     |-- isp_stats.c                   //3A related statistics
     |-- regs.c                        //registers related read and write operations
     |-- rkisp1.c                      //corresponding isp_sd entity node, including data received from mipi and with crop function
    |-- i2c/
      |-- ov13850.c                    //CIS(cmos image sensor) driver
      |-- vm149c.c                     //VCM driver ic driver
    |-- spi/                           // rk1608 ap driver
      |-- rk1608.c                     //register rk1608 spi device
      |-- rk1608_dev.c                 //register /dev/rk_preisp misc device
      |-- rk1608_dphy.c                //register v4l2 media node, interact with rk1608 and AP
```

3 The rkisp1 isp driver

3.1 Brief description of the framework

RKISP driver implements hardware configurations, interrupt processing, control buffer rotation, and control subdevice(such as mipi dphy and sensor) power-on and power off and other functions mainly base on v4l2/media framework.

The block diagram below describes the topology of RKISP1 driver.



Name	Type	Description
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-dphy	v4l2_subdev	MIPI-DPHY configure.
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) driver

4.1 Methods to obtain driver version

- Obtain from kernel startup log

  Rkisp1 ff910000.rkisp1: rkisp1 driver version: v00.01.02

- Obtain by the following command

  cat /sys/module/video_rkisp1/parameters/version

4.2 CIS device registration (DTS)

There is an introduction document of RKISP DTS node in kernel source code, which is located in Documentation/devicetree/bindings/media/rockchip-isp1.txt.

There is also a mipi dphy driver node introduction document in kernel source code, which is located in Documentation/devicetree/bindings/media/rockchip-mipi-dphy.txt.

4.2.1 MIPI CIS registration

Take rk3399 isp0 and ov13850 as examples below:

```
ov13850: ov13850@10 {
        compatible = "ovti,ov13850"; // need to consistent with the matching
                                        string in driver
        status = "okay";
        reg = <0x10>; // sensor I2C device address
        clocks = <&cru SCLK_CIF_OUT>; // sensor clockin (MCLK)
        clock-names = "xvclk";
        reset-gpios = <&gpio2 10 GPIO_ACTIVE_HIGH>;
```

```
            // reset GPIO and active level
            pwdn-gpios = <&gpio1 4 GPIO_ACTIVE_HIGH>;
            // power GPIO and active level
            pinctrl-names = "rockchip,camera_default";
            pinctrl-0 = <&cif_clkout>; // pinctrl configuration
            rockchip,camera-module-index = <0>; // module number which should
                                                not be repeated
            rockchip,camera-module-facing = "back"; // module facing, can be
                                                    "back" or "front"
            rockchip,camera-module-name = "CMK-CT0116"; // module name
            rockchip,camera-module-lens-name = "Largan-50013A1"; // lens name
            // module name and lens name are used to match IQ xml file.
            lens-focus = <&vm149c>; // vcm driver settings which is required when
support AF
            port {
                ucam_out0: endpoint {
                    remote-endpoint = <&mipi_in_ucam0>;
                    // port name of mipi dphy
                    data-lanes = <1 2>;
                    // mipi lanes, 1lane is <1>, 4lane is <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>;
                  // port name of sensor
                data-lanes = <1 2>;
                  // mipi lanes，1lane is <1>, 4lane is <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>;
                  // port name of isp
            };
        };
    };
};
```

```
&rkisp1_0 {
  status = "okay";
  port {
      #address-cells = <1>;
      #size-cells = <0>;
      isp0_mipi_in: endpoint@0 {
          reg = <0>;
          remote-endpoint = <&dphy_rx0_out>;
            // port name of mipi dphy
      };
  };
};
&isp0_mmu {
status = "okay"; // iommu is used in isp driver, so isp iommu also needs to be
                    opened
};
```

4.2.2 DVP CIS registration

Take rk3326 isp and gc0312/gc2145 as examples:

```
&i2c2 {
  status = "okay";
  gc0312@21 {
      status = "okay";
      compatible = "galaxycore,gc0312"; // need to consistent with the
                                                matching string in driver
      reg = <0x21>; // sensor I2C device address
```

```
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clkout_m0>; // pinctrl configuration
        clocks = <&cru SCLK_CIF_OUT>; // sensor clockin (MCLK)
        clock-names = "xvclk";
        avdd-supply = <&vcc2v8_dvp>; // configure sensor power
        dovdd-supply = <&vcc1v8_dvp>;
        dvdd-supply = <&vcc1v8_dvp>;
        pwdn-gpios = <&gpio2 14 GPIO_ACTIVE_HIGH>;
          // power GPIO and active level
        rockchip,camera-module-index = <1>; // module number which should
                                                    not be repeated
        rockchip,camera-module-facing = "front"; // module facing, can be
                                                        "back" or "front"
        rockchip,camera-module-name = "CameraKing"; // module name
        rockchip,camera-module-lens-name = "Largan"; // lens name
        port {
            gc0312_out: endpoint {
                remote-endpoint = <&dvp_in_fcam>;
                  // port name of isp
            };
        };
    };
    gc2145@3c {
        status = "okay";
        compatible = "galaxycore,gc2145"; // need to consistent with the
                                                matching string in driver
```

```
        reg = <0x3c>; // sensor I2C device address
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clkout_m0>; // pinctrl configuration
        clocks = <&cru SCLK_CIF_OUT>; // sensor clockin (MCLK)
        clock-names = "xvclk";
        avdd-supply = <&vcc2v8_dvp>; // configure sensor power
        dovdd-supply = <&vcc1v8_dvp>;
        dvdd-supply = <&vcc1v8_dvp>;
        pwdn-gpios = <&gpio2 13 GPIO_ACTIVE_HIGH>;
          // power pin assignment and active level
      rockchip,camera-module-index = <0>; // module number which should
                                                    not be repeated
      rockchip,camera-module-facing = "back"; // module facing can be "back"
                                                        or "front"
        rockchip,camera-module-name = "CameraKing"; // module name
        rockchip,camera-module-lens-name = "Largan"; // lens name
        port {
            gc2145_out: endpoint {
                remote-endpoint = <&dvp_in_bcam>;
                  // port name of isp
            };
        };
    };
};
&isp_mmu {
    status = "okay";
```

```
    };
    &rkisp1 {
      status = "okay";
      pinctrl-names = "default";
      pinctrl-0 = <&cif_clkout_m0 &dvp_d0d1_m0 &dvp_d2d9_m0
&dvp_d10d11_m0>;
      // pinctrl setting, add dvp pin related configurations
      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>; // port name of sensor
          };
              dvp_in_bcam: endpoint@1 {
                  reg = <1>;
                  remote-endpoint = <&gc2145_out>; // port name of sensor
              };
          };
      };
    };
```

4.3 CIS driver instruction

Camera sensor interact with controller by I2C. At present, sensor driver is implemented according to I2C device driver mode, and uses v4l2 subdev method to realize interaction with host driver.

4.3.1 Brief introduction of data type

4.3.1.1 struct i2c_driver

[Note]

Define i2c device driver information

[Definition]

```
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;
    ……
};
```

[Key members]

Member name	Description
@driver	Device driver model driver It mainly contains driver name and of_match_table that matches with DTS registered device. The .probe function will

	be called when the compatible domain in of_match_table matches the compatible domain of dts file.
@id_table	List of I2C devices supported by this driver If kernel does not use of_match_table and dts registration devices for matching, kernel will use table to match
@probe	Callback for device binding
@remove	Callback for device unbinding

[Example]

```
#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

[Note]

Define ops callbacks for subdevs.

[Definition]

```
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;
};
```

[Key members]

Member name	Description
.core	define core ops callbacks for subdevs
.video	callbacks used when v4l device was opened in video mode.
.pad	v4l2-subdev pad level operations

[Example]

```
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

[Note]

Define core ops callbacks for subdevs.

[Definition]

```
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
    ……
};
```

[Key members]

Member name	Description
.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.

[Example]

```
static const struct v4l2_subdev_core_ops ov13850_core_ops = {
    .ioctl = ov13850_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl32 = ov13850_compat_ioctl32,
#endif
};
```

At present, the following private ioctls are used to implement module information query and OTP information query settings.

Private ioctl	Description
RKMODULE_GET_MODULE_INFO	obtain module information, please refer to struct rkmodule_inf for details

RKMODULE_AWB_CFG	switch sensor compensation function for awb; if the module does not download golden awb value, you can set it here. please refer to struct rkmodule_awb_cfg for details
RKMODULE_LSC_CFG	switch sensor compensation function for lsc; please refer to struct rkmodule_lsc_cfg for details

4.3.1.4 struct v4l2_subdev_video_ops

[Note]

Callbacks used when v4l device was opened in video mode.

[Definition]

```
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);
  ……
};
```

[Key members]

Members name	Description
.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

[Example]

```
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

[Note]

v4l2-subdev pad level operations

[Definition]

```
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);
    ……

};
```

[Key members]

Member name	Description
. 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

[Example]

```
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

[Note]

The control operations that the driver has to provide.

[Definition]

```
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);
};
```

[Key members]

Member name	Description
.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.

[Example]

```
static const struct v4l2_ctrl_ops ov13850_ctrl_ops = {
    .s_ctrl = ov13850_set_ctrl,
};
```

Rkisp driver requires the users controls function provided by framework, cameras sensor driver must implement the following control functions. Please refer to CIS driver V4L2-controls list 1.

4.3.1.7 struct xxxx_mode

[Note]

Sensor can support information of each mode.

This structure is commonly seen in sensor drivers, although it is not required by the v4l2 standard.

[Definition]

```
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;
};
```

[Key members]

Member name	Description
.width	Effective image width
.height	Effective image height
.max_fps	Image FPS, denominator/numerator is fps
hts_def	HTS by default, effective image width + HBLANK
vts_def	VTS by default, effective image height + VBLANK
exp_def	Default exposure time
*reg_list	Registers list

[Example]

```
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

[Note]

Frame format on the media bus

[Definition]

```
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];
};
```

[Key members]

Member name	Description
width	frame width
height	frame height
code	refer to MEDIA_BUS_FMT table
field	V4L2_FIELD_NONE：frame output mode V4L2_FIELD_INTERLACED：field output mode

[Example]

4.3.1.9 struct rkmodule_base_inf

[Note]

Basic information of modules, application use this information to match with IQ

[Definition]

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

[Key members]

Member name	Description
sensor	sensor name, obtained from sensor driver
module	module name, obtained from DTS configuration, according to module specification
lens	lens name, obtained from DTS configuration, according to modulespecification

[Example]

4.3.1.10 struct rkmodule_fac_inf

[Note]

Module OTP factory information

[Definition]

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

[Key members]

Member name	Description
flag	whether the group information is valid or not
module	module name, obtain the number from OTP, then get the module name by the number
lens	lens name, obtain the number from OTP, then get the

	lens name by the number
year	year of production, such as 12 standing for 2012
month	production month
day	production date

[Example]

4.3.1.11 struct rkmodule_awb_inf

[Note]

Module OTP awb measurement information

[Definition]

```
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));
```

[Key members]

Member name	Description
flag	whether the group information is valid or not
r_value	AWB R measurement information of current module
b_value	AWB B measurement information of current module

gr_value	AWB GR measurement information of current module
gb_value	AWB GB measurement information of current module
golden_r_value	AWB R measurement information of a typical module, if not downloaded, set to 0
golden_b_value	AWB B measurement information of a typical module, if not downloaded, set to 0
golden_gr_value	AWB GR measurement information of a typical module, if not downloaded, set to 0
golden_gb_value	AWB GB measurement information of a typical module, if not downloaded, set to 0

[Example]

4.3.1.12 struct rkmodule_lsc_inf

[Note]

Module OTP lsc measurement information

[Definition]

```
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));
```

[Key members]

Member name	Description
flag	whether the group information is valid or not
lsc_w	lsc actual width
lsc_h	lsc actual height
decimal_bits	the number of decimal bits of lsc measurement information. If it cannot be obtained, set it to 0.
lsc_r	lsc r measurement information
lsc_b	lsc b measurement information
lsc_gr	lsc gr measurement information
lsc_gb	lsc gb measurement information

[Example]

4.3.1.13 struct rkmodule_af_inf

[Note]

Module OTP af measurement information

[Definition]

```
struct rkmodule_af_inf {
  __u32 flag; // whether the group information is valid or not
  __u32 vcm_start; // vcm starting current
  __u32 vcm_end; // vcm ending current
  __u32 vcm_dir; // vcm measurement direction
} __attribute__ ((packed));
```

[Key members]

Member name	Description
flag	whether the group information is valid or not

vcm_start	vcm starting current
vcm_end	vcm ending current
vcm_dir	vcm measurement direction

[Example]

4.3.1.14 struct rkmodule_inf

[Note]

Module information

[Definition]

```
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));
```

[Key members]

Member name	Description
base	modules basic information
fac	module OTP factory information
awb	module OTP awb measurement information
lsc	module OTP lsc measurement information
af	module OTP af measurement information

[Example]

4.3.1.15 struct rkmodule_awb_cfg

[Note]

Module OTP awb configuration information

[Definition]

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

[Key members]

Member name	Description
enable	indicates whether awb correction is enabled
golden_r_value	AWB R measurement information of a typical module
golden_b_value	AWB B measurement information of a typical module
golden_gr_value	AWB GR measurement information of a typical module
golden_gb_value	AWB GB measurement information of a typical module

[Example]

4.3.1.16 struct rkmodule_lsc_cfg

[Note]

Module OTP lsc configuration information

[Definition]

```
struct rkmodule_lsc_cfg {
  __u32 enable;
```

} __attribute__ ((packed));

[Key members]

Member name	Description
enable	indicate whether lsc correction is enabled

[Example]

4.3.2 API brief instruction

4.3.2.1 xxxx_set_fmt

[Description]

Set sensor output format

[Syntax]

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

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structure pointer	input
*cfg	subdev pad information structure pointer	input
*fmt	Pad-level media bus format structure pointer	input

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.2.2 xxxx_get_fmt

[Description]

Obtain sensor output format.

[Syntax]

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

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structure pointer	input
*cfg	subdev pad information structure pointer	input
*fmt	Pad-level media bus format structure pointer	output

[Return value]

Return value	Description
0	successful
non 0	failed

Refer to MEDIA_BUS_FMT table.

4.3.2.3 xxxx_enum_mbus_code

[Description]

Enumerate sensor output bus format

[Syntax]

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

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structure pointer	input
*cfg	subdev pad information structure pointer	input
*code	media bus format enumeration structure pointer	output

[Return value]

Return value	Description
0	successful
non 0	failed

The following table summarizes formats corresponding to various image types. Refer to MEDIA_BUS_FMT table.

4.3.2.4 xxxx_enum_frame_sizes

[Description]

Enumerate sensor output size

[Syntax]

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

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structural pointer	input
*cfg	subdev pad information structural pointer	input
*fse	media bus frame size structural pointer	output

[Return value]

Return value	Description
0	successful

non 0	failed

4.3.2.5 xxxx_g_frame_interval

[Description]

Obtain sensor output fps。

[Syntax]

static int xxxx_g_frame_interval(struct v4l2_subdev *sd,

struct v4l2_subdev_frame_interval *fi)

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structure pointer	input
*fi	pad-level frame rate structure pointer	output

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.2.6 xxxx_s_stream

[Description]

Set stream input and output。

[Syntax]

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

[Parameters]

Parameter name	Description	Input/ Output
*sd	v4l2 subdev structure pointer	input
on	1: start stream output; 0: stop stream output	input

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.2.7 xxxx_runtime_resume

[Description]

The callback function when sensor is powered on.

[Syntax]

static int xxxx_runtime_resume(struct device *dev)

[Parameters]

Parameter name	Description	Input/ Output
*dev	device structure pointer	input

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.2.8 xxxx_runtime_suspend

[Description]

Callback function when sensor is powered off

[Syntax]

static int xxxx_runtime_suspend(struct device *dev)

[Parameters]

Parameter name	Description	Input/ Output
*dev	device structure pointer	input

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.2.9 xxxx_set_ctrl

[Description]

Set the value of each control

[Syntax]

static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)

[Parameters]

Parameter name	Description	Input/ Output
*ctrl	v4l2_ctrl structure pointer	input

[Return value]

Return value	Description
0	successful
non 0	failed

4.3.3 Driver porting steps

1. Implement the part of standard I2C sub-device driver.

1.1 Implement the following members according to struct i2c_driver instruction

struct driver.name

struct driver.pm

struct driver. of_match_table

probe function

remove function

1.2 Detailed description of probe function implementation

1).Obtaining CIS device resources which mainly used to analyze the resources defined in DTS file. Please refer to Camera device registration(DTS);

1.1) RK private resource definition, the naming method is as follows: rockchip,camera-module-xxx, this part of resource will be uploaded to the user state camera_engine to determine IQ effect parameter matching;

1.2) CIS device resource definition, RK related reference driver generally contains the following items:

CIS device working reference clock	The external independent crystal oscillator solution does not need to be obtained. RK reference design generally adopts AP output clock. This solution needs to be obtained, and usually names xvclk.
CIS device control GPIO	For example: Resst pin, Powerdown pin
CIS device control power supply	Obtain matching software power control resources based on actual hardware design, such as gpio, regulator

1.3) Check CIS device ID number, after obtaining the necessary resources through the above steps, it is recommended that driver read device ID number to check the accuracy of hardware, and this step is not necessary.

1.4) CIS v4l2 device and initialization of media entity;

V4l2 sub-device: v4l2_i2c_subdev_init, RK CIS driver requires subdev to have its own device node for user state camera_engine access, through device node achieves exposure control;

media entity：media_entity_init

2.Refer to struct v4l2_subdev_ops instruction to implement v4l2 sub device driver，

it mainly include the following 3 members:

struct v4l2_subdev_core_ops
struct v4l2_subdev_video_ops
struct v4l2_subdev_pad_ops

2.1 Refer to struct v4l2_subdev_core_ops instruction to implement its callback function, it mainly include the following callback:

.ioctl .compat_ioctl32

The callback mainly implements RK private control commands, including:

RKMODULE_GET_MODULE_INFO	The module information (module name, etc.) defined in DTS file, by which camera_engine is uploaded.
RKMODULE_AWB_CFG	When module OTP information is enabled, camera_engine passes the typical module AWB calibration value through this command. CIS driver is responsible for comparing with the current module AWB calibration value and generating R/B gain value to CIS MWB module.
RKMODULE_LSC_CFG	When module OTP information is enabled, camera_engine control LSC calibration value to be enabled by this command.

2.2 Refer to struct v4l2_subdev_video_ops instruction to implement its callback function, it mainly includes the following two callback functions:

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 Refer to struct v4l2_subdev_pad_ops instruction to implement its callback function, it mainly includes the following four callback functions:

.enum_mbus_code	Enumerate current CIS driver supporting data formats
.enum_frame_size	Enumerate current CIS driver supporting resolution
.get_fmt	Rkisp driver obtains the data format of CIS output through this callback, which must be implemented; Refer to MEDIA_BUS_FMT table for data type definitions for Bayer raw sensor, SOC yuv sensor, and BW raw sensor output. For support information of field output mode, refer to struct v4l2_mbus_framefmt definition;
.set_fmt	Set CIS driver output data format and resolution, which must be implemented;

2.4 Refer to struct v4l2_ctrl_ops instruction to implement, it mainly includes the following callbacks:

.s_ctrl	Rkisp driver and camera_engine implement CIS exposure control by setting different commands;

Refer to CIS driver V4L2-controls list 1 to implement each control ID, where the following ID belongs to information acquisition class, this part are implemented according to standard integer menu controls;

V4L2_CID_LINK_FREQ	Refer to the standard definition in CIS driver V4L2-controls list 1. Currently, the rkisp driver obtains MIPI bus frequency according to this command.
V4L2_CID_PIXEL_RATE	For MIPI bus: pixel_rate = link_freq * 2 * nr_of_lanes / bits_per_sample
V4L2_CID_HBLANK	Refer to the standard definition in CIS driver

	V4L2-controls list 1
V4L2_CID_VBLANK	Refer to the standard definition in CIS driver V4L2-controls list 1

RK camera_engine will use the above commands to get necessary information to calculate exposure, including the following formulas:

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

The following ID belongs to control class, and RK camera_engine controls CIS through this type of commands.

V4L2_CID_VBLANK	Adjust VBLANK to adjust frame rate and Exposure time max;
V4L2_CID_EXPOSURE	Set exposure time, unit: the number of exposure lines
V4L2_CID_ANALOGUE_GAIN	Set exposure gain, actually total gain = analog gain*digital gain in fact; unit: Gain register value

3. CIS driver does not involve the definition of hardware data interface. The interface connection between CIS device and AP is represented by the port of DTS device node. Refer to chapter 4.1 MIPI Sensor Registration and chapter 4.2 DVP Sensor Registration for the port description.

4. Please refer to the driver list for CIS

5 VCM driver

5.1 VCM device registration (DTS)

- **RK VCM driver private parameters:**

Name	Definition
Starting current	VCM can just push module lens to the nearest end of module lens's movable distance (module far-focus). At this time, the output current value of VCM driver ic is defined as starting current.
Rated current	VCM can just push module lens to the farthest end of module lens's movable distance (module near-focus). At this time, the output current value of VCM driver ic is defined as rated current.
VCM current output mode	Oscillation occurs during VCM movement. VCM driver ic current output change needs to consider vcm oscillation circle to minimize oscillation. The output mode determines the time when output current changes to the target value.

```
vm149c: vm149c@0c { // vcm driver configuration which is required when supporting AF
        compatible = "silicon touch,vm149c";
        status = "okay";
        reg = <0x0c>;
        rockchip,vcm-start-current = <0>; // Motor starting current
        rockchip,vcm-rated-current = <100>; // Motor rated current
```

```
        rockchip,vcm-step-mode = <4>; // Motor driver ic current output mode
        rockchip,camera-module-index = <0>; // Module number
        rockchip,camera-module-facing = "back"; // Module facing, include
                                                "back" and "front"
};
ov13850: ov13850@10 {
        ......
        lens-focus = <&vm149c>; // Vcm driver settings which is required when
                                supporting AF
        ......
};
```

5.2 VCM driver instructions

5.2.1 Data type brief instruction

5.2.1.1 struct i2c_driver

[Note]

Define i2c device driver information

[Definition]

```
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;
  ……
};
```

[Key members]

Member name	Description
@driver	Device driver model driver It mainly includes driver name and of_match_table that match DTS registered device. The .probe function will be called when the compatible domain in of_match_table matches the compatible domain of dts file.
@id_table	List of I2C devices supported by this driver If kernel does not use of_match_table and dts registration devices for matching, kernel will use table to match
@probe	Callback for device binding
@remove	Callback for device unbinding

[example]

```
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

[Note]

Define core ops callbacks for subdevs.

[Definition]

```
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

……

};

[Key members]

Member name	Description
.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.

[Example]

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

Currently using the following private ioctl to achieve motor movement time information query

RK_VIDIOC_VCM_TIMEINFO

5.2.1.3 struct v4l2_ctrl_ops

[Note]

The control operations that the driver has to provide.

[Definition]

```
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);
};
```

[Key members]

Member name	Description
.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.

[Example]

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

vm149c_get_ctrl and vm149c_set_ctrl support the following control：

V4L2_CID_FOCUS_ABSOLUTE

5.2.2 API brief instruction

5.2.2.1 xxxx_get_ctrl

[Description]

Obtain the moving position of motor

[Syntax]

static int xxxx_get_ctrl(struct v4l2_ctrl *ctrl)

[Parameter]

Parameter name	Description	Input /Output
*ctrl	v4l2 control structure pointer	output

[Return value]

Return value	Description
0	successful
non 0	failed

5.2.2.2 xxxx_set_ctrl

[Description]

Set the moving position of motor

[syntax]

static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)

[Parameter]

Parameter name	Description	Input /Output
*ctrl	v4l2 control structure pointer	input

[Return value]

Return value	Description
0	successful

non 0	failed

5.2.2.3 xxxx_ioctl/xxxx_compat_ioctl32

[Description]

The implementation function of customized ioctl mainly includes obtaining the time information of motor movement.

Implement customized RK_VIDIOC_COMPAT_VCM_TIMEINFO

[syntax]

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)

[Parameter]

Parameter name	Description	Input /Output
*sd	v4l2 subdev structure pointer	input
cmd	Ioctl command	input
*arg/arg	parameter pointer	input

[Return value]

Return value	Description
0	successful
non 0	failed

5.2.3 Driver porting steps

1. Implement the part of standard i2c sub-device driver.

1.1 According to struct i2c_driver description, it mainly implement the following items：

struct driver.name

struct driver.pm

struct driver. of_match_table

probe function

remove function

1.2 Details of probe function implementation：

1）VCM device resource acquisition is mainly to obtain DTS resources, refer to VCM device registration (DTS)

1.1) RK private resource definition, naming method is like rockchip, camera-module-xxx, mainly to provide device parameters and camera devices to match.

1.2) VCM parameter definition, naming method is like rockchip, vcm-xxx, mainly related to hardware parameter starting current, rated current, moving mode, and parameters are related to motor range and speed

2）VCM v4l2 device and initialization of media entity.

v4l2 sub device：v4l2_i2c_subdev_init, RK VCM driver requires subdev to have its own device node for user state camera_engine access, through which focus control is implemented.

Media entity：media_entity_init；

3) RK AF algorithm defines the positional parameter of the entire movable distance of module lens as [0, 64], and the corresponding range of the entire movable distance of module lens on VCM driver current is [starting current, rated current], in this function, it is recommended to implement the mapping conversion relationship between this two current.

2. The v4l2 sub-device driver implementation mainly includes following two members:

struct v4l2_subdev_core_ops
struct v4l2_ctrl_ops

2.1 Refer to v4l2_subdev_core_ops instruction to implement callback function, mainly includes the following callback function:

.ioctl .compat_ioctl32

The callback mainly implements RK private control commands, including:

RK_VIDIOC_VCM_TIMEINFO	The camera_engine obtains the time required for lens movement by this command, and accordingly determines when lens stops and whether CIS frame exposure time overlaps with lens movement time; Lens movement time is related to lens movement distance and VCM driver ic current output mode.

2.2 Refer to v4l2_ctrl_ops instruction to implement callback function, mainly includes the following callback function:

.g_volatile_ctrl .s_ctrl

.g_volatile_ctrl and.s_ctrl implement the following command according to the standard v4l2 control：

V4L2_CID_FOCUS_ABSOLUTE	The camera_engine uses this command to set and get the absolute position of lens. In RK AF algorithm, the position parameter of the entire movable distance of lens is defined as [0, 64].

6 Rk1608 AP driver

6.1 Driver version obtain method

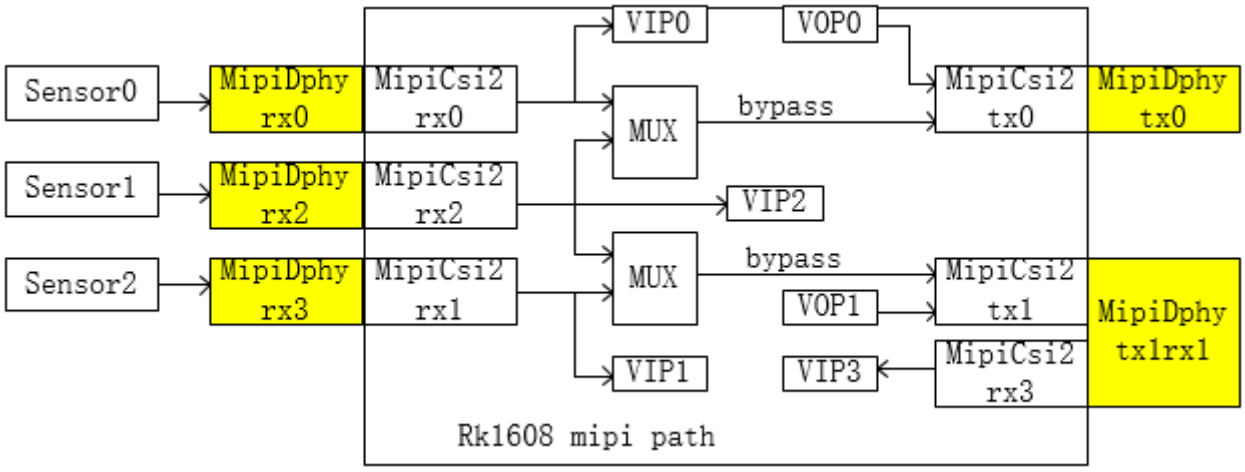
You can query by the following command:

echo v > /dev/rk_preisp

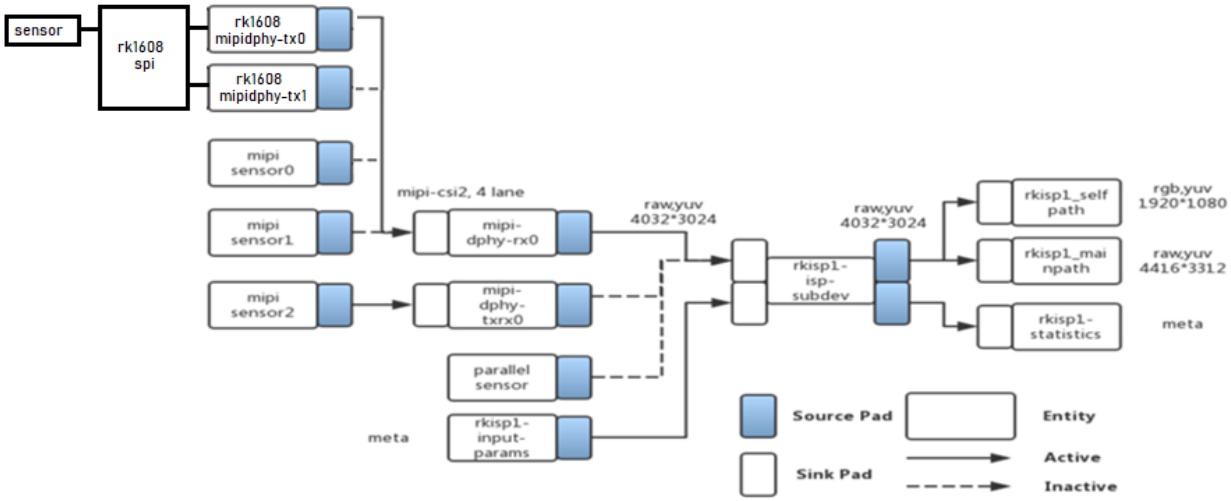
6.2 Brief description of the framework

Rk1608 AP driver mainly controls rk1608 to power on and off, load rk1608 firmware, communicate with rk1608 through spi bus, control rk1608 to collect sensor data, algorithm operation, data output and so on.

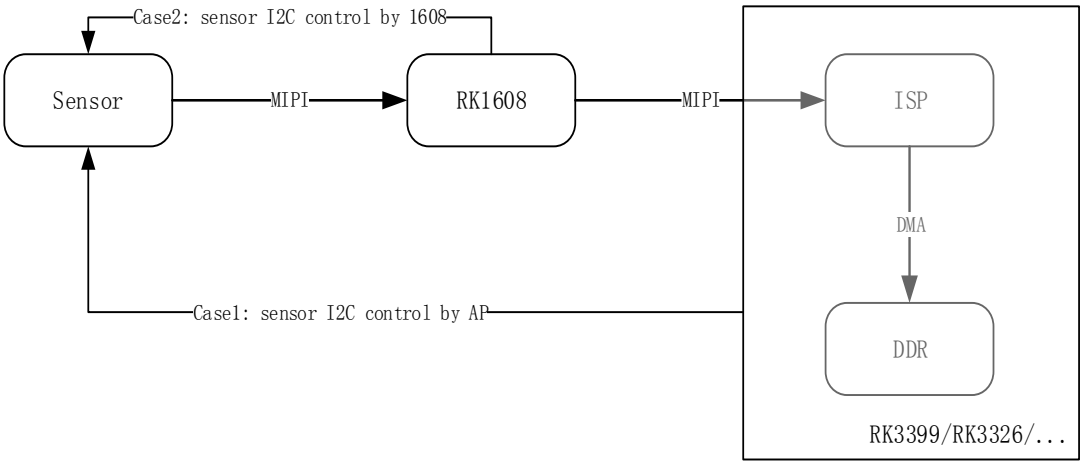
RK1608 internal mipi connection path diagram is as follows:



The following describes the topology of rk1608 AP driver and rkisp1. The whole rk1608 AP driver can be regarded as a soc sensor.



Sensor i2c can be controlled by AP or directly controlled by 1608



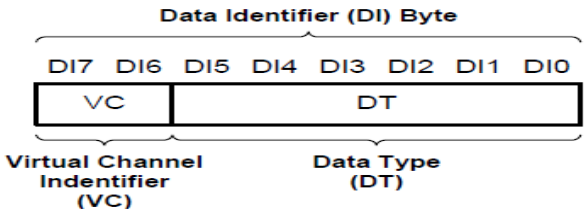
6.3 Rk1608 AP device registration (DTS)

Take rk3326-evb-lp3-v10-rk1608-linux.dts as an example:

```
#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
*Note: it is in bypass mode here, vip/vop mode needs to change rk1608 firmware
*If the hardware is connected to rk1608 mipi dphy rx3, it needs to be set in_mipi=<1> here
 *bypass mode:
 *rx0/rx2 in，tx0 out
 *rx2/rx3 in，tx1 out*/
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>;// Module number, which should not
                                                    be repeated
 rockchip,camera-module-facing = "back";//module facing, can be "back" or
                                                        "front"
rockchip,camera-module-name = "TongJu";//module name
rockchip,camera-module-lens-name = "CHT842-MD";//lens name
ports {
    #address-cells = <1>;
    #size-cells = <0>;
    port@0 {
        rk1608_dphy0_in: endpoint {
            remote-endpoint = <&rk1608_out0>;//rk1608 mipidphy input
port
        };
    };
    port@1 {
        rk1608_dphy_out: endpoint {
```

```
                    remote-endpoint = <&mipi_in_ucam>;//rk1608 mipidphy
output port
                    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 use this driver when communicating
directly with rk1608
         * sensor and AP communication use the corresponding sensor driver */
        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 output port
                data-lanes = <1 2>;
            };
        };
    };
};
&spi1 {
…
    spi_rk1608@00 {//rk1608 spi device
        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 firmware names
        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 output port
                };
            };
            port@1 {
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <1>;
                rk1608_in0: endpoint@0 {
                    reg = <0>;
                    remote-endpoint = <&cam_out>;//rk1608 input port
                };
            };
        };
    };
};
```

Some algorithms do not require 1608 to directly connect to sensor. In this case, only spi_rk1608 node can be configured. The port information in spi_rk1608 node does not need to be configured. Applications complete algorithm by providing ioctl interface control 1608 in rk1608_dev.c. In addition, the above mipidphy dts can only support one output format to AP. After the following commit, multiple output formats can be supported to AP, but this commit is not yet merged.

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

Change-Id: Icc9c14891d6f7494a6d6cc4752dabcf07278d708

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

The dts corresponding to this commit need to be changed.

```
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 { // One of the output formats of 1608 to AP
        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 { // Second output format of 1608 to AP
        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>;
                        };
                };
        };
};
```

6.4 Rk1608 AP driver instruction

Rk1608 driver uses spi to complete message communication between AP and rk1608. Mipi sensor can be connected to rk1608 through i2c or connected to AP for communication management.

6.4.1 Brief instruction of data types

6.4.1.1 struct spi_driver

[Note]

Define spi device driver information

[definition]

```
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;
};
```

[Key members]

Member name	Description
@driver	Device driver model driver It mainly contains driver name and of_match_table that match DTS registered device. The .probe function will be called when the compatible domain in of_match_table matches the compatible domain in dts file.
@id_table	List of SPI devices supported by this driver If kernel does not use of_match_table and dts registration devices for matching, kernel will use table to match
@probe	Callback for device binding
@remove	Callback for device unbinding

[Example]

```
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

[Note]

Define core ops callbacks for subdevs.

[Definition]

```
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
  ……
};
```

[Key members]

Member name	Description
.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.

[Example]

```
static const struct v4l2_subdev_core_ops rk1608_core_ops = {
  .s_power = rk1608_sensor_power,
  .ioctl = rk1608_ioctl,
};
```

Currently the following private ioctls are used to get sensor information and exposure control:

Private ioctl	Description
PREISP_CMD_SAVE_HDRAE_PARAM	Passing the current awb gain information and lsc compensation information to 1608; Directly called by isp driver; Refer to struct preisp_hdrae_para_s for details;
PREISP_CMD_SET_HDRAE_EXP	Make exposure setting in sensor HDR mode; Refer to struct preisp_hdrae_para_s for details
RKMODULE_GET_MODULE_INFO	Obtaining module information, and actually returning module information corresponding to the sensor; Refer to struct rkmodule_inf for details

6.4.1.3 struct v4l2_subdev_video_ops

[Note]

Callbacks used when v4l device was opened in video mode.

[Definition]

```
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);
    ……
};
```

[Key members]

Member name	Description
.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

[Example]

```
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

[Note]

v4l2-subdev pad level operations

[Definition]

```
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);
    ……

};
```

[Key members]

Member name	Description
. 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

[Example]

```
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

[Note]

File operations

[Definition]

```
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);
    ……

};
```

[Example]

```
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 device file node operation, upper layer can directly access this node to control rk1608.

6.4.1.6 struct preisp_hdrae_para_s

[Note]

Awb and lsc parameter for preisp.

[Definition]

```
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];
};
```

[Key members]

Member name	Description
r_gain	awb r gain
b_gain	awb b gain

gr_gain	awb gr gain
gb_gain	awb gb gain
lsc_table	lsc table

[Example]

6.4.1.7 struct preisp_hdrae_exp_s

[Note]

Hdr ae exposure setting.

[Definition]

```
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;
};
```

[Key members]

Member name	Description
long_exp_reg	HDR long exposure time register value;
long_gain_reg	HDR long exposure gain register value;
middle_exp_reg	HDR middle exposure time register value;
middle_gain_reg	HDR middle exposure gain register value;
short_exp_reg	HDR short exposure time register value;
short_gain_reg	HDR short exposure gain register value;
long_exp_val	HDR long exposure time actual value, pass the float value passed by the upper layer to 1608;
long_gain_val	HDR long exposure gain actual value，pass the float value passed by the upper layer to 1608;
middle_exp_val	HDR middle exposure time actual value，pass the float value passed by the upper layer to 1608;
middle_gain_val	HDR middle exposure gain actual value，pass the float value passed by the upper layer to 1608;
short_exp_val	HDR short exposure time actual value，pass the float value passed by the upper layer to 1608;
short_gain_val	HDR short exposure gain actual value，pass the float value passed by the upper layer to 1608;

[Example]

6.4.2 Brief instruction of API

The following V4L2 API usage is the same as sensor, it won't describe in

details here:

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

[Description]

Transfer commands via /dev/rk_preisp, can be used to debug

[usage]

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 steps

1) When porting the sensor connected to 1608, you can use 1608 bypass firmware for debugging. 1608 bypass firmware is generally integrated in SDK

Key steps when debugging with 1608 bypass firmware:

·Confirm that the driver connected to sensor is in liner mode, 1608 bypass firmware does not support hdr mode;

· Confirm that the 1608 firmware configured in dts file is 1608 bypass firmware;

Firmware-names in spi_rk1608 node can specify firmware name

firmware-names = "fw_rk1608_bypass.rkl";

·Modify mipidphy configuration in dts file

code = <MEDIA_BUS_FMT_SRGGB10_1X10>; // sensor output image format

width = <1932>; // sensor output image width

height= <1094>; // sensor output image height

htotal = <2500>; // about 400 bigger than the width of sensor output image

vtotal = <1500>; // about 400 bigger than the height of sensor output image

inch0-info = <1932 1094 0x2b 0x2b 1>; // Fill in according to sensor output image format

outch0-info = <1932 1094 0x2b 0x2b 1>; // Fill in according to sensor output image format

link-freqs = /bits/ 64 <LINK_FREQ>; // The link-freqs set here is higher than the link-freqs of sensor->1608, which is generally 30% larger. You can also try the highest link-freqs 750MHz.

2) When debugging 1608, you need to confirm that the 1608 firmware is successfully loaded. 1608 firmware is loaded successfully or not, you can confirm whether there is the following log in kernel log.

rk1608 spi32766.0: Download firmware success!

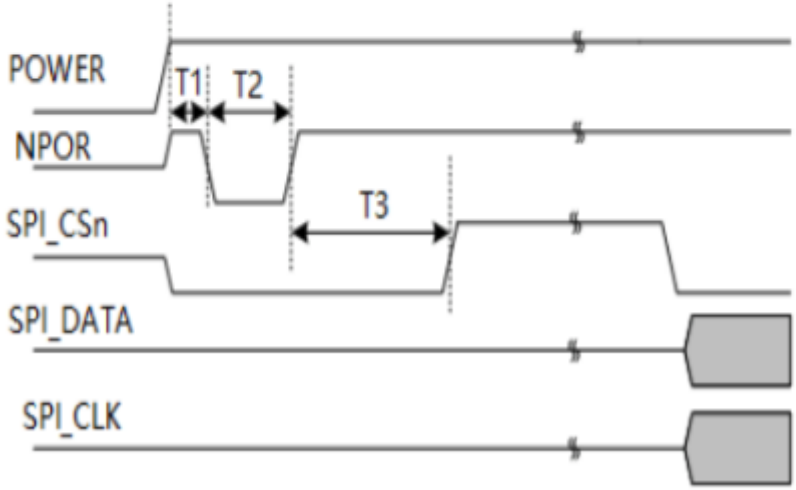
If there is a problem with firmware loading, it is generally necessary to check the timing of clock, voltage, spi signal, and reset/spi-cs pins supplied to 1608.

1608 timing sequence requirements for powering up are as follows:

1. Power up timing:

T1≥0, T2≥0, T3≥0, T4≥0

2. SPI timing after power up



T1>0, T2≥1ms, T3≥3ms

The NPOR is 1608 reset pin.

7 media-ctl / v4l2-ctl tools

The media-ctl tool operates through media devices such as /dev/medio0, which manage format, size, and links of various nodes in media topology.

The v4l2-ctl tool is for /dev/video0, /dev/video1 and other video devices. It performs set_fmt, reqbuf, qbuf, dqbuf, stream_on, stream_off and other operations on video device.

For specific usage, please refer to help information by command. The following are some common usage.

1）Print topology

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

2）Change fmt/size

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

3）set fmt and capture frame

```
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）Set exposure and gain control etc.

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

8 FAQ

8.1 How to judge rkisp driver loading status

If RKISP driver is successfully loaded, video and media devices will be saved in /dev/ directory. There may be multiple /dev/video devices in a system. You can query video node registered by RKISP through /sys.

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

You can also use media-ctl command to print topology to see if pipeline is working.

1）Judge whether camera driver is loaded successfully

When all cameras are registered, kernel will print the following log.

localhost ~ # dmesg | grep Async

[0.682982] rkisp1: Async subdev notifier completed

If you find that there is no "Async subdev notifier completed" this line log in kernel, firstly to check if sensor has an error, or I2C communication is successful.

8.2 How to capture yuv data of isp output

Reference commands are as follows：

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
```

8.3 How to capture original Raw Bayer data from sensor output

Reference commands are as follows:

```
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
```

It should be noted that although ISP does not process raw graph, it will still fill low bits of the 10-bit data to 16 bits. Whether sensor inputs is10bit/12bit, the application gets 16bits per pixel.

8.4 How to support black and white cameras

CIS driver needs to change output format of black and white sensor to one of the following three formats.

MEDIA_BUS_FMT_Y8_1X8 (sensor 8bit output)

MEDIA_BUS_FMT_Y10_1X10 (sensor 10bit output)

MEDIA_BUS_FMT_Y12_1X12 (sensor 12bit output)

The above format is returned by the functions xxxx_get_fmt and xxxx_enum_mbus_code.

Rkisp driver will specifically set these three formats to support obtaining black and white images.

In addition, if application layer needs to obtain images in Y8 format, only SP Path can be used, because only SP Path can support Y8 format output.

8.5 How to support Parity field synthesis

Rkisp1 driver supports Parity field synthesis, **limiting requirements are as follows:**

1.MIPI interface: support output frame count number (from frame start and frame end short packets), that Rkisp1 driver is used to judge parity of current field;

2. BT656 interface: support output standard SAV/EAV, that is, bit6 is parity field flag information, that rkisp1 driver is used to judge parity of current field;

3. This function is included in rkisp1_selfpath video device node in rkisp1 driver. Other video device nodes do not include this function. If app layer incorrectly calls other device nodes, driver will prompt the following error message:

"only selfpath support interlaced"

rkisp1_selfpath information can be viewed by media-ctl –p:

```
- 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]
```

Device driver implementation method is as follows:

Device driver format.field needs to be set to V4L2_FIELD_INTERLACED, it means that current device output format is a parity field, that is, format.field is returned in xxxx_get_fmt function. Refer to driver/media/i2c/tc35874x.c driver;

Appendix A CIS Driver V4L2-controls List 1

CID	Description
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 color 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.

Appendix B MEDIA_BUS_FMT table

CIS sensor type	Sensor output 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(black and white), that is raw bw sensor	MEDIA_BUS_FMT_Y8_1X8 MEDIA_BUS_FMT_Y10_1X10 MEDIA_BUS_FMT_Y12_1X12

Appendix C CIS reference driver list

CIS data interface	CIS output data type	Frame/Field	Reference drive
MIPI	Bayer RAW	frame	ov8858.c
MIPI	YUV	frame	Gc2145.c
MIPI	RAW BW	frame	Ov7251
MIPI	YUV	field	tc35874x.c
ITU.BT601	Bayer RAW		
ITU.BT601	YUV		
ITU.BT601	RAW BW		
ITU.BT656	Bayer RAW		