密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKCIF_Driver_User_Manual

（ISP 部）

文件状态： [] 正在修改 [√] 正式发布	当前版本：	V1.0
	作　　者：	徐鸿飞
	完成日期：	2018-12-7
	审　　核：	邓达龙
	完成日期：	2018-12-11

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	徐鸿飞	2018-12-7	发布初版		

目录

1. 文档适用平台

芯片平台	软件系统	支持情况
RK3288	Linux(Kernel-4.4)	Y
RK3126C	Android-9.0	Y

2. Camera 文件目录说明

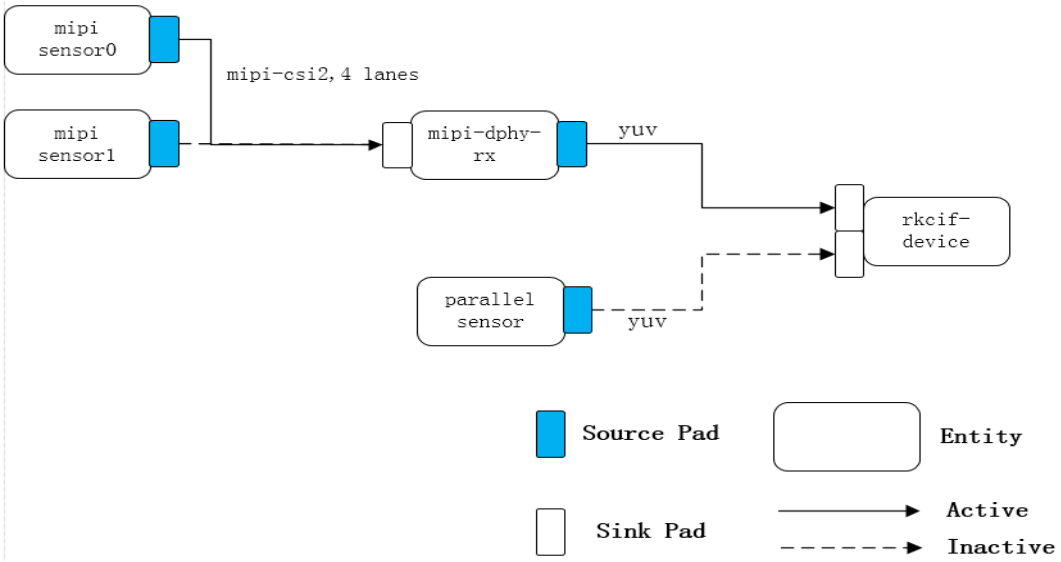
```
Linux Kernel-4.4：
  |
  |arch/arm64/boot/dts/rockchip        DTS 配置文件
  |phy/rockchip/phy-rockchip-mipi-rx.c    mipi dphy 驱动，独立于 cif 驱动
  |drivers/media
    |
    |platform/rockchip/cif          RKCIF 驱动
      |—— capture.c                 主要完成硬件配置，v4l2、vb2 框架下的相关回调，
                                    帧中断处理
      |—— dev.c                     主要完成 probe，sub-device 异步(Async)注册，
                                    iommu 及 clk 管理
      |—— dev.h                     驱动相关结构体定义
      |—— regs.h                    寄存器宏定义
      |i2c/                         Camera Sensor 驱动
```

3. RKCIF 驱动说明

3.1 CIF 驱动代码简介

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

下面的框图描述了 RKCIF1 驱动的拓扑结构。



名称	类型	描述
rkcif-device	v4l2_vdev, capture	Format: YUV; Support: Crop
rockchip-sy-mipi-dphy	v4l2_subdev	MIPI-DPHY Configure.

- CIF oneframe(单帧)模式. 单帧模式下,驱动每收到一个帧中断,在中断处理函数中设置下一帧 buffer 地址,然后再开始采集下一帧数据.
- CIF pingpong(双 buffers)模式. pingpong 帧模式下,cif 在采集一帧后(f1)，会自动采集下一帧(f2),cif 驱动在收到 f1 中断后,要更新数据地址指针并清除 frame1 status,以让 cif 继续采集新的数据帧.
- 单帧模式下， 如果图像分辨率较高， 比如 1080p， 那么 fps 可能只能达到 15fps
- 如果无 iommu 功能并在 reqbuf 时采用 mmap 方式分配 buffer， 要求 kernel 保留较大的 CMA size。 可以在 kernel defconfig 中修改， 如下。

```
CONFIG_CMA_SIZE_MBYTES=64
```

3.2 CIF dts 配置

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

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

3.2.1 板级配置

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

- 旧的 cif 驱动

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

- 新的 cif 驱动

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

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

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

示例一， px3se evb 板子上 cif dts 配置。

```
CIF 节点定义在 arch/arm/boot/dts/rk312x.dtsi，
cif_new: cif-new@1010a000 {
    compatible = "rockchip,rk3128-cif";
```

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

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

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

Port 子节点定义了 cif_in 节点， 并声明与它链接的远端节点为 adv7191_out。 因为

adv7181 采用 dvp 接口， 这里也同时指定 vsync, hsync 的有效状态， 其中 0 表示低电平有效， 1 表示高电平有效。

3.3 CIF 驱动调试及常见问题

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

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

3.3.1 判断 cif 是否 probe 成功

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

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

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

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

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

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

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

注意：

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

3.3.2 判断 Sensor 与 CIF 是否已经绑定

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

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

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

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

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

3.3.3 打开 debug 开关

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

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

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

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

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

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

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

```
/* Just log the ioctl name + error code */
#define V4L2_DEV_DEBUG_IOCTL 0x01
/* Log the ioctl name arguments + error code */
```

```
#define V4L2_DEV_DEBUG_IOCTL_ARG 0x02

/* Log the file operations open, release, mmap and get_unmapped_area */

#define V4L2_DEV_DEBUG_FOP 0x04

/* Log the read and write file operations and the VIDIOC_(D)QBUF ioctls */

#define V4L2_DEV_DEBUG_STREAMING 0x08

/* Log poll() */

#define V4L2_DEV_DEBUG_POLL 0x10
```

3.3.4 利用 v4l2-ctl 抓帧

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

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

其中，width,height 可以小于 sensor 的输出大小，驱动会从左上角开始裁剪(crop)。CIF 硬件上也可以支持从任意一位置开始裁剪， 但目前驱动尚未支持。

4. Camera 设备注册(DTS)

RKCIF 的 DTS 节点在 kernel 源码中有文档说明，

它位于 Documentation/devicetree/bindings/media/ rockchip-cif.txt。

mipi dphy 驱动节点也在 kernel 源码中有文档说明，

它位于 Documentation/devicetree/bindings/media/rockchip-mipi-dphy.txt

4.1 MIPI Sensor 注册

示例：

```
ov2685: ov2685@36 {
        compatible = "ovti,ov2685"; // 需要与驱动中的匹配字符串一致
        status = "okay";
        reg = <0x36>; // sensor I2C 7 位设备地址
        clocks = <&ext_cam_clk>; // sensor clickin 配置
        clock-names = "refclk";
        reset-gpios = <&gpio8 8 GPIO_ACTIVE_LOW>;
        // reset 管脚分配及有效电平
        port {
            ov2685_out: endpoint {
                remote-endpoint = <&dphy_rx_in>;
                // mipi dphy 端的 port 名
                clock-lanes = <0>;
                // mipi clock lane 信息
                data-lanes = <1 2>;
                // mipi data lane 信息，1lane 为 <1>, 4lane 为 <1 2 3 4>
                link-frequencies = /bits/ 64 <297000000>;
                //括号内为 mipi 频率
            };
        };
    };
```

```
&mipi_dphy_rx {
    status = "okay";
    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;
            dphy_rx_in: endpoint@1 {
                reg = <1>;
                /* TODO add sensor */
                remote-endpoint = <&ov2685_out>;
                data-lanes = <1 2 3 4>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;
            dphy_rx_out: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&cif_in>;
            };
        };
```

```
    };
};
```

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

    port {
        cif_in: endpoint@0 {
            remote-endpoint = <&dphy_rx_out>;
            data-lanes = <1 2 3 4>;
        };
    };
};
```

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

4.2 DVP Sensor 注册

以 px3se cif 和 adv7181 为例进行说明：

```
&i2c2 {
        status = "okay";
        camera: adv7181@21 {
            compatible = "adi,adv7181"; // 需要与驱动中的匹配字符串一致
            reg = <0x21>; // sensor I2C 7 位设备地址
            pinctrl-names = "default";
```

```
            pinctrl-0 = <&cif_rst>; // pinctl 设置
            dvdd-supply = <&dvdd_1v8>;
            dvddio-supply = <&dvdd_3v3>; // sensor 电源配置
            reset-gpios = <&gpio3 11 GPIO_ACTIVE_LOW>;
            port {
                adv7181_out: endpoint {
                    remote-endpoint = <&cif_in>; // cif 端的 port 名
                };
            };
    };
}
```

```
&cif_new {
    status = "okay";
    port {
        cif_in: endpoint {
            remote-endpoint = <&adv7181_out>; // sensor 端的 port 名
            vsync-active = <0>; //vsync input polarity
            hsync-active = <1>; //href  input polarity
        };
    };
};
```

5. Camera 设备驱动

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。

5.1 数据类型简要说明

struct i2c_driver

[说明]

定义 i2c 设备驱动信息

[定义]

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

[关键成员]

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

@remove	Callback for device unbinding

[示例]

```
#if IS_ENABLED(CONFIG_OF)
static const struct of_device_id 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);
```

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,
};
```

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,
};
```

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。

成员名称	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.
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.

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 v412 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,
  },
};
```

5.2 API 简要说明

xxxx_set_fmt

[描述]

设置 sensor 输出格式。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_get_fmt

[描述]

获取 sensor 输出格式。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_enum_mbus_code

[描述]

枚举 sensor 输出 bus format。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

下表总结了各种图像类型对应的 format

图像类型	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)	MEDIA_BUS_FMT_Y8_1X8 MEDIA_BUS_FMT_Y10_1X10 MEDIA_BUS_FMT_Y12_1X12

xxxx_enum_frame_sizes

[描述]

枚举 sensor 输出大小。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_g_frame_interval

[描述]

获取 sensor 输出 fps。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_s_stream

[描述]

设置 stream 输入输出。

[语法]

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

[参数]

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

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_runtime_resume

[描述]

sensor 上电时的回调函数。

[语法]

static int xxxx_runtime_resume(struct device *dev)

[参数]

参数名称	描述	输入输出
*dev	device 结构体指针	输入

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_runtime_suspend

[描述]

sensor 下电时的回调函数。

[语法]

static int xxxx_runtime_suspend(struct device *dev)

[参数]

参数名称	描述	输入输出
*dev	device 结构体指针	输入

[返回值]

返回值	描述
0	成功
非 0	失败

xxxx_set_ctrl

[描述]

设置各个 control 的值。

[语法]

static int xxxx_set_ctrl(struct v4l2_ctrl *ctrl)

[参数]

参数名称	描述	输入输出
*ctrl	v4l2_ctrl 结构体指针	输入

[返回值]

返回值	描述
0	成功
非 0	失败

6. 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'
```