密级状态：绝密(　　)　秘密(　　)　内部资料(　　)　公开(√)

CIF_ISP10_Driver_User_Manual

文件状态： ［　］草稿 ［　］正式发布 ［√］正在修改	文件标识：	
	当前版本：	1.0
	作　　者：	张云龙、黄春成、胡克俊、邓达龙
	完成日期：	2017-11-23

历 史 版 本

版本	日 期	描 述	作 者	审核
V1.0	2017-11-23	建立文档	邓达龙	邓达龙

目 录

1. 文档适用平台 4
2. Camera 文件目录说明 4
3. Camera 设备注册(DTS) 5
3.1 MIPI Sensor 注册 5
3.2 DVP Sensor 注册 6
4. Camera 设备驱动 8
4.1 数据类型简要说明 8
struct i2c_driver 8
struct v4l2_subdev_core_ops 9
struct v4l2_subdev_video_ops 10
struct v4l2_subdev_pad_ops 10
struct ov_camera_module_custom_config 11
struct ov_camera_module_config 13
PLTFRM_CAM_ITF_MIPI_CFG 14
PLTFRM_CAM_ITF_DVP_CFG 14
4.2 API 简要说明 15
sensorxxx_g_VTS 15
sensorxxx_auto_adjust_fps 16
sensorxxx_write_aec 16
sensorxxx_ filltimings 16
sensorxxx_ g_timings 17
sensorxxx_set_flip 17
sensorxxx_start_streaming 18
sensorxxx_ stop _streaming 18
sensorxxx_check_camera_id 19
5. 驱动移植简单步骤说明 20

1. 文档适用平台

芯片平台	软件系统	支持情况
RK3399	Linux(Kernel-4.4)	Y
RK3288	Linux(Kernel-4.4)	Y

此类平台的 isp driver 按照 isp 硬件版本来区分，具体命名如下：
RK3288/RK3399 平台 ISP Driver 名称：cif_isp10

2. Camera 文件目录说明

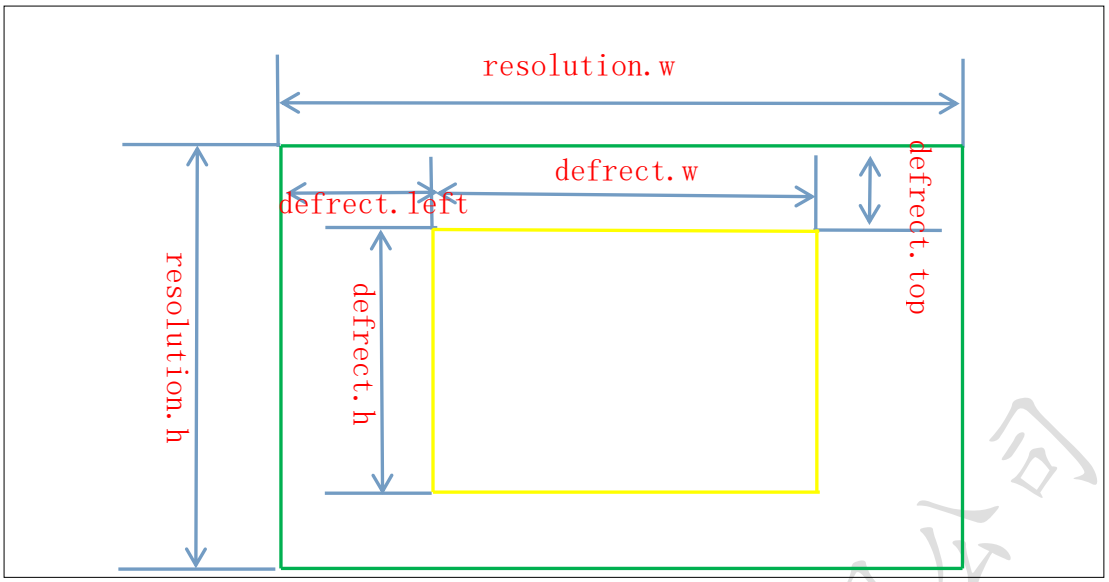
RK3288/RK3399 Linux kernel：

```
|
|arch/arm/boot/dts                  DTS 配置文件
|drivers/media
   |
   |platform/rk-isp10               ISP Host 驱动
   |i2c/soc_camera/rockchip/        Camera Sensor 驱动
```

3. Camera 设备注册(DTS)

3.1 MIPI Sensor 注册

```
camera0: camera-module@36 {
        status = "okay";//是否加载模块，默认开启
        compatible = "omnivision,ov2710-v4l2-i2c-subdev";
        //omnivision sensor 类型
        //ov2710-v4l2-i2c-subdev  中 ov2710 为 sensor 型号
        //需要与驱动名字一致
        reg = <0x36>;// Sensor I2C 设备地址
        device_type = "v4l2-i2c-subdev";//设备类型
        clocks = <&clk_cif_out>;//sensor clickin 配置
        clock-names = "clk_cif_out";
        pinctrl-names = "rockchip,camera_default", "rockchip,camera_sleep";
        pinctrl-0 = <&cif_dvp_clk_out>;
        pinctrl-1 = <&cif_dvp_clk_out_sleep>;
        rockchip,pd-gpio = <&gpio3 GPIO_B0 GPIO_ACTIVE_HIGH>;
        //powerdown 管脚分配及有效电平
        rockchip,pwr-gpio = <&gpio3 GPIO_B5 GPIO_ACTIVE_HIGH>;
        //power 管脚分配及有效电平
        rockchip,rst-gpio = <&gpio3 GPIO_D1 GPIO_ACTIVE_LOW>;
        //reset 管脚分配及有效电平
        rockchip,camera-module-mclk-name = "clk_cif_out";//mclk 时钟源配置
        rockchip,camera-module-facing = "back";//前后置配置
        rockchip,camera-module-name = "LA6110PA";//Camera 模组名称
        rockchip,camera-module-len-name = "YM6011P";//Camera 模组镜头
        rockchip,camera-module-fov-h = "128";//模组水平可视角度配置
        rockchip,camera-module-fov-v = "55.7";//模组垂直可视角度配置
        rockchip,camera-module-orientation = <0>;//模组角度设置
        rockchip,camera-module-flip = <0>;
        rockchip,camera-module-mirror = <0>;
//以上 2 个属性控制摄像头驱动中的镜像配置，如果图像旋转 180 度，可以将这 2 个属性修改成
相反的值即可旋转 180；
        /* resolution.w, resolution.h, defrect.left, defrect.top, defrect.w, defrect.h */
        rockchip,camera-module-defrect0 = <1920 1080 0 0 1920 1080>;
        // resolution.w: sensor 输出列数,
        //resolution.h: sensor 输出行数,
        // defrect.left:输出偏移列数,
        // defrect.top:输出偏移行数,
        // defrect.w:输出列数, defrect.left+defrect.w<=resolution.w,
        //defrect.h:输出行数,defrect.h+defrect.top<=resolution.h,
        //具体如下图所示：
```




```
        rockchip,camera-module-flash-support = <0>;//flash 控制开关
        rockchip,camera-module-mipi-dphy-index = <0>;
        //sensor 实际使用的 phy,要与硬件实际连接对应
};


&i2c1 { //配置 Camera 设备连接到哪个 I2C 模块上，一般为 I2C1
        status = "okay";//是否加载 i2c 模块，默认开启
        #include "rv1108-camb-xx.dtsi"
};
&cif_isp0 {
        rockchip,camera-modules-attached = <&camera0 &camera1 &camera2>;
        //配置需要使用的 camera 列表,连接到 ISP 设备节点
        status = "okay";
};
```

3.2 DVP Sensor 注册

```
camera2: camera-module@1a {
        status = "okay";
        compatible = "sony,imx323-v4l2-i2c-subdev";
        reg = <0x1a>;
        device_type = "v4l2-i2c-subdev";
        clocks = <&clk_cif_out>;
        clock-names = "clk_cif_out";
        pinctrl-names = "rockchip,camera_default", "rockchip,camera_sleep";
        pinctrl-0 = <&cif_dvp_d0d1 &cif_dvp_d2d9 &cif_dvp_d10d11
                &cif_dvp_clk_in &cif_dvp_clk_out &cif_dvp_sync>;
        pinctrl-1 = <&cif_dvp_d0d1_sleep &cif_dvp_d2d9_sleep
                &cif_dvp_d10d11_sleep &cif_dvp_clk_in_sleep
                &cif_dvp_clk_out_sleep &cif_dvp_sync_sleep>;
//DVP pin 引脚配置，具体定义在文件 rv1108.dtsi 中
其它配置和 MIPI Sensor 相同
        rockchip,pd-gpio = <&gpio3 GPIO_D1 GPIO_ACTIVE_LOW>;
```

```
        rockchip,pwr-gpio = <&gpio3 GPIO_B5 GPIO_ACTIVE_HIGH>;
        rockchip,rst-gpio = <&gpio3 GPIO_B0 GPIO_ACTIVE_LOW>;
        rockchip,camera-module-mclk-name = "clk_cif_out";
        rockchip,camera-module-facing = "back";
        rockchip,camera-module-name = "LA6114PA";
        rockchip,camera-module-len-name = "YM6011P";
        rockchip,camera-module-fov-h = "122";
        rockchip,camera-module-fov-v = "63";
        rockchip,camera-module-orientation = <0>;
        rockchip,camera-module-iq-flip = <0>;
        rockchip,camera-module-iq-mirror = <0>;
        rockchip,camera-module-flip = <0>;
        rockchip,camera-module-mirror = <0>;
        /* resolution.w, resolution.h, defrect.left, defrect.top, defrect.w, defrect.h */
        rockchip,camera-module-defrect0 = <2200 1125 48 13 1920 1080>;
        rockchip,camera-module-flash-support = <0>;
};
```

4. 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 v4l2 subdev 的方式实现与 host driver 之间的交互。

文件列表如下：

文件名称	描述
ov_camera_module.c	OV Sensor 驱动公共函数文件
ov4689_v4l2-i2c-subdev.c	OV4689 Sensor 驱动
aptina_camera_module.c	Aptina Sensor 驱动公共函数文件
ar0330cs_v4l2-i2c-subdev.c	AR0330 Sensor 驱动
imx_camera_module.c	Sony Sensor 驱动公共函数文件
imx323_v4l2-i2c-subdev.c	IMX323 Sensor 驱动文件
rk_camera_module.c	平台 Sensor 驱动公共函数实现

4.1 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
@probe	Callback for device unbinding
@remove	Callback for device unbinding

[示例]:

```
static const struct i2c_device_id ov4689_id[] = {
        { ov4689_DRIVER_NAME, 0 },
```

```
        { }
};
static struct of_device_id ov4689_of_match[] = {
        {.compatible = "omnivision,ov4689-v4l2-i2c-subdev"},
        {},
};
static struct i2c_driver ov4689_i2c_driver = {
        .driver = {
                .name = ov4689_DRIVER_NAME,
                .owner = THIS_MODULE,
                .of_match_table = ov4689_of_match
        },
        .probe = ov4689_probe,
        .remove = ov4689_remove,
        .id_table = ov4689_id,
};
```

struct v4l2_subdev_core_ops

[说明]:

Define core ops callbacks for subdevs

[定义]:

```
struct v4l2_subdev_core_ops {
        ……
        int (*g_ctrl)(struct v4l2_subdev *sd, struct v4l2_control *ctrl);
        int (*s_ctrl)(struct v4l2_subdev *sd, struct v4l2_control *ctrl);
        int (*s_ext_ctrls)(struct v4l2_subdev *sd, struct v4l2_ext_controls *ctrls);
        long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
……

        int (*s_power)(struct v4l2_subdev *sd, int on);
        ……
};
```

[关键成员]:

成员名称	描述
.g_ctrl	callback for VIDIOC_G_CTRL ioctl handler code
.s_ctrl	callback for VIDIOC_S_CTRL ioctl handler code
.s_ext_ctrls	callback for VIDIOC_S_EXT_CTRLS ioctl handler code
.s_power	puts subdevice in power saving mode (on == 0) or normal operation mode (on == 1).
.ioctl	called at the end of ioctl() syscall handler at the V4L2 core. used to provide support for private ioctls used on the driver.

[示例]:

```
static struct v4l2_subdev_core_ops ov4689_camera_module_core_ops = {
        .g_ctrl = ov_camera_module_g_ctrl,
        .s_ctrl = ov_camera_module_s_ctrl,
        .s_ext_ctrls = ov_camera_module_s_ext_ctrls,
```

```
    .s_power = ov_camera_module_s_power,
    .ioctl = ov_camera_module_ioctl
};
```

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
.s_frame_interval	callback for VIDIOC_S_FRAMEINTERVAL ioctl handler code
.s_stream	used to notify the driver that a video stream will start or has stopped

[示例]:

```
static struct v4l2_subdev_video_ops ov4689_camera_module_video_ops = {
    .s_frame_interval = ov_camera_module_s_frame_interval,
    .s_stream = ov_camera_module_s_stream
};
```

struct v4l2_subdev_pad_ops

[说明]:

v4l2-subdev pad level operations

[定义]:

```
struct v4l2_subdev_pad_ops {
    ……

    int (*enum_frame_interval)(struct v4l2_subdev *sd,
                struct v4l2_subdev_pad_config *cfg,
                struct v4l2_subdev_frame_interval_enum *fie);
    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_frameintervals	callback for VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL ioctl handler code.
.s_fmt	callback for VIDIOC_SUBDEV_G_FMT ioctl handler code.
.g_fmt	callback for VIDIOC_SUBDEV_S_FMT ioctl handler code

[示例]:

```
static struct v4l2_subdev_pad_ops ov4689_camera_module_pad_ops = {
    .enum_frame_interval = ov_camera_module_enum_frameintervals,
    .get_fmt = ov_camera_module_g_fmt,
    .set_fmt = ov_camera_module_s_fmt,
};

static struct v4l2_subdev_ops ov4689_camera_module_ops = {
    .core = &ov4689_camera_module_core_ops,
    .video = &ov4689_camera_module_video_ops,
    .pad = &ov4689_camera_module_pad_ops
};
```

struct ov_camera_module_custom_config

[说明]:

定义 ov camera sensor 驱动函数操作集

[定义]:

```
struct ov_camera_module_custom_config {
    int (*start_streaming)(struct ov_camera_module *cam_mod);
    int (*stop_streaming)(struct ov_camera_module *cam_mod);
    int (*check_camera_id)(struct ov_camera_module *cam_mod);
    int (*s_ctrl)(struct ov_camera_module *cam_mod, u32 ctrl_id);
    int (*g_ctrl)(struct ov_camera_module *cam_mod, u32 ctrl_id);
    int (*g_timings)(struct ov_camera_module *cam_mod,
        struct ov_camera_module_timings *timings);
    int (*g_exposure_valid_frame)(struct ov_camera_module *cam_mod);
    int (*s_ext_ctrls)(struct ov_camera_module *cam_mod,
        struct ov_camera_module_ext_ctrls *ctrls);
    int (*set_flip)(
        struct ov_camera_module *cam_mod,
        struct pltfrm_camera_module_reg reglist[],
        int len);
    int (*init_common)(struct ov_camera_module *cam_mod);
```

```
        int (*read_otp)(struct ov_camera_module *cam_mod);
            struct ov_camera_module_config *configs;
        u32 num_configs;
        u32 power_up_delays_ms[3];
        void *priv;
};
```

[关键成员]:

成员名称	描述
.start_streaming	start_streaming: (mandatory) will be called when sensor should be put into streaming mode right after the base config has been written to the sensor. After a successful call of this function the sensor should start delivering frame data. 调用该函数前，Sensor 不允许输出数据流
.stop_streaming	stop_streaming: (mandatory) will be called when sensor should stop delivering data. After a successful call of this function the sensor should not deliver any more frame data. 调用该函数后，Sensor 不允许输出数据流
.s_ext_ctrls	Sensor 控制函数， callback for VIDIOC_S_EXT_CTRLS ioctl handler code （V4L2_CID_GAIN、V4L2_CID_EXPOSURE）
.g_timings	获取 Sensor timing 相关参数，例如：PCLK,HTS,VTS
.check_camera_id	check_camera_id: (optional) will be called when the sensor is owered on. If provided should check the sensor ID/version required by the custom driver. Register access should be possible when this function is invoked.
.set_flip	设置 Sensor 镜像
.g_exposure_valid_frame	获取 Sensor 曝光生效场数（1 帧 = 2 场） 返回值： 0 ---- 当前帧及时生效 2 ---- 后一帧生效 4 ---- 后 2 帧生效
.power_up_delays_ms[3]	Sensor 操作时延： power_up_delays_ms [0]：上电后时延 power_up_delays_ms [1]：硬件 power down 唤醒后时延 power_up_delays_ms [2]：Stream on 后时延

[示例]:

```
static struct ov_camera_module_custom_config ov4689_custom_config = {
        .start_streaming = ov4689_start_streaming,
        .stop_streaming = ov4689_stop_streaming,
        .s_ctrl = ov4689_s_ctrl,
        .s_ext_ctrls = ov4689_s_ext_ctrls,
        .g_ctrl = ov4689_g_ctrl,
        .g_timings = ov4689_g_timings,
        .check_camera_id = ov4689_check_camera_id,
        .set_flip = ov4689_set_flip,
        .configs = ov4689_configs,
        .num_configs = ARRAY_SIZE(ov4689_configs),
        .power_up_delays_ms = {5, 20, 0}
};
```

struct ov_camera_module_config

[说明]:
定义 ov camera sensor 配置属性

[定义]:

```
struct ov_camera_module_config {
        const char *name;
        struct v4l2_mbus_framefmt frm_fmt;
        struct v4l2_subdev_frame_interval frm_intrvl;
        bool auto_exp_enabled;
        bool auto_gain_enabled;
        bool auto_wb_enabled;
        struct ov_camera_module_reg *reg_table;
        u32 reg_table_num_entries;
        struct ov_camera_module_reg *reg_diff_table;
        u32 reg_diff_table_num_entries;
        u32 v_blanking_time_us;
        u32 line_length_pck;
        u32 frame_length_lines;
        struct ov_camera_module_timings timings;
        bool soft_reset;
        bool ignore_measurement_check;
        struct pltfrm_cam_itf itf_cfg;
};
```

[关键成员]:

成员名称	描述
.name	当前配置名称
.frm_fmt	当前配置数据格式，参考 struct v4l2_mbus_framefmt 定义 .width： frame width .height：frame height .code：data format code (from enum v4l2_mbus_pixelcode)
.frm_intrvl	Pad-level frame rate (from struct v4l2_subdev_frame_interval)
.reg_table	当前配置寄存器列表
.v_blanking_time_us	当前配置场消隐时间
PLTFRM_CAM_ITF_MIPI_CFG	硬件接口属性定义，参考 struct pltfrm_cam_itf

[示例]:

```
static struct ov_camera_module_config ov4689_configs[] = {
        .name = "2688x1520_30fps",
        .frm_fmt = {
                .width = 2688,
                .height = 1520,
                .code = V4L2_MBUS_FMT_SBGGR10_1X10
        },
        .frm_intrvl = {
```

```
            .interval = {
                .numerator = 1,
                .denominator = 30
            }
        },
        .auto_exp_enabled = false,
        .auto_gain_enabled = false,
        .auto_wb_enabled = false,
        .reg_table = (void *)ov4689_init_tab_2688_1520_30fps,
        .reg_table_num_entries = ARRAY_SIZE(ov4689_init_tab_2688_1520_30fps),
        .v_blanking_time_us = 6100,
        PLTFRM_CAM_ITF_MIPI_CFG(0, 2, 999, ov4689_EXT_CLK)
};
```

PLTFRM_CAM_ITF_MIPI_CFG

[说明]:
定义 MIPI 硬件接口属性

[定义]:
#define PLTFRM_CAM_ITF_MIPI_CFG(v, nb, br, mk)

[关键成员]:

成员名称	描述
v	mipi visual channel number index value：0,1,2,3
nb	mipi lanes number value：1,2,4
br	mipi bit rate(单位: Mbps)
mk	Sensor 工作参考时钟频率(单位:Hz)

[示例]:
PLTFRM_CAM_ITF_MIPI_CFG(0, 2, 999, ov4689_EXT_CLK)

PLTFRM_CAM_ITF_DVP_CFG

[说明]:
定义 DVP 并口硬件接口属性

[定义]:
#define PLTFRM_CAM_ITF_DVP_CFG(ty, vs, hs, ck, ck_hz, mk)

[关键成员]:

成员名称	描述
ty	并口类型 value： PLTFRM_CAM_ITF_BT601_8 = 0x20000071, //8bit 位宽 BT601 PLTFRM_CAM_ITF_BT656_8 = 0x20000072,// 8bit 位宽 BT656

	PLTFRM_CAM_ITF_BT601_10 = 0x20000091, //10bit位宽BT601 PLTFRM_CAM_ITF_BT656_10 = 0x20000092, PLTFRM_CAM_ITF_BT601_12 = 0x200000B1, PLTFRM_CAM_ITF_BT656_12 = 0x200000B2, PLTFRM_CAM_ITF_BT601_16 = 0x200000F1, PLTFRM_CAM_ITF_BT656_16 = 0x200000F2, PLTFRM_CAM_ITF_BT656_8I = 0x20000172 //8bit BT656 Interlace 格式
vs	硬件场同步信号有效电平（BT601 有效）
hs	硬件行同步信号有效电平（BT601 有效）
ck:	Pclk 采集有效边沿
mk	Sensor 工作参考时钟频率(单位:Hz)

[示例]:

```
PLTFRM_CAM_ITF_DVP_CFG(
        PLTFRM_CAM_ITF_BT601_12,
        PLTFRM_CAM_SIGNAL_HIGH_LEVEL,
        PLTFRM_CAM_SIGNAL_HIGH_LEVEL,
        PLTFRM_CAM_SDR_NEG_EDG,
        IMX323_EXT_CLK)
```

4.2 API 简要说明

sensorxxx_g_VTS

[描述]：

获取当前 VTS 信息

[语法]：

static int sensorxxx_g_VTS(struct ov_camera_module *cam_mod, u32 *vts)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入
*vts	sensor vts 指针	输出

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_auto_adjust_fps

[描述]：

根据设置曝光时间调整帧率

[语法]：

static int sensorxxx_auto_adjust_fps(struct ov_camera_module *cam_mod, u32 exp_time)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入
exp_time	sensor 曝光行数	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_write_aec

[描述]：

设置 sensor 曝光时间、增益。曝光时间以及增益值的寄存器换算由上层算法实现，驱动设置的为寄存器值，无需换算。

[语法]：

static int sensorxxx_write_aec(struct ov_camera_module *cam_mod)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_ filltimings

[描述]：

根据 sensor 的配置寄存器表，按照寄存器地址匹配方式查询时序参数，填写入相应的结构体中，避免实时读取 I2C，

[语法]：

static int sensorxxx_filltimings(struct ov_camera_module_custom_config *custom)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_ g_timings

[描述]：

获取当前 Sensor 时序参数

[语法]：

static int sensorxxx_g_timings(struct ov_camera_module *cam_mod, strucov_camera_module_timings *timings)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入
*timings	时序信息结构体指针	输出

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_set_flip

[描述]：

根据当前的 flip/mirror 设置，修改 sensor 寄存器列表中相应寄存器设置，寄存器列表输入 sensor 后镜像生效

[语法]：

static int sensorxxx_set_flip(struct ov_camera_module *cam_mod, struct pltfrm_camera_module_reg reglist[], int len)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_start_streaming

[描述]：

打开 sensor 数据流输出

[语法]：

static int sensorxxx_start_streaming(struct ov_camera_module *cam_mod)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_ stop _streaming

[描述]：

关闭 sensor 数据流输出

[语法]：

static int sensorxxx_stop_streaming(struct ov_camera_module *cam_mod)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	成功
非 0	失败

sensorxxx_check_camera_id

[描述]：

Sensor 硬件 ID 号校验

[语法]：

static int sensorxxx_check_camera_id(struct ov_camera_module *cam_mod)

[参数]：

参数名称	描述	输入输出
cam_mod	struct ov_camera_module 结构体指针	输入

[返回值]：

返回值	描述
0	设备硬件 ID 匹配成功
非 0	失败

5. 驱动移植简单步骤说明

1） sensor 驱动的加载与 DTS 设备匹配；
static struct i2c_driver ov4689_i2c_driver
2）填写 senosr 寄存器设置列表以及相关的结构体信息；
static struct ov_camera_module_config ov4689_configs[]
3) RAW Sensor 实现 AEC 控制相关函数接口,YUV Sensor 忽略该步骤;
sensorxxx_write_aec
sensorxxx_auto_adjust_fps
sensorxxx_g_VTS
sensorxxx_filltimings
sensorxxx_g_timings
4）实现 Sensor 数据流控制函数接口；
sensorxxx_start_streaming
sensorxxx_stop_streaming
5) DTS 文件设备硬件相关配置；
设备挂载的 I2C 通道是否正确？
I2C 设备地址是否配置正确？
电源控制引脚以及电源设置是否正确？
Power down、Reset 引脚以及电平是否正确?
设备数据接口配置是否正确？