Camera驱动代码详解

WWW.UNISOC.COM
紫光展锐科技

修改历史 Revision History

版本号 Version	日期 Date	注释 Notes
V1.0	2020/8/4	初稿

Contents

1 Camera驱动的基本配置及文件路径说明

2 Camera驱动.h文件详解

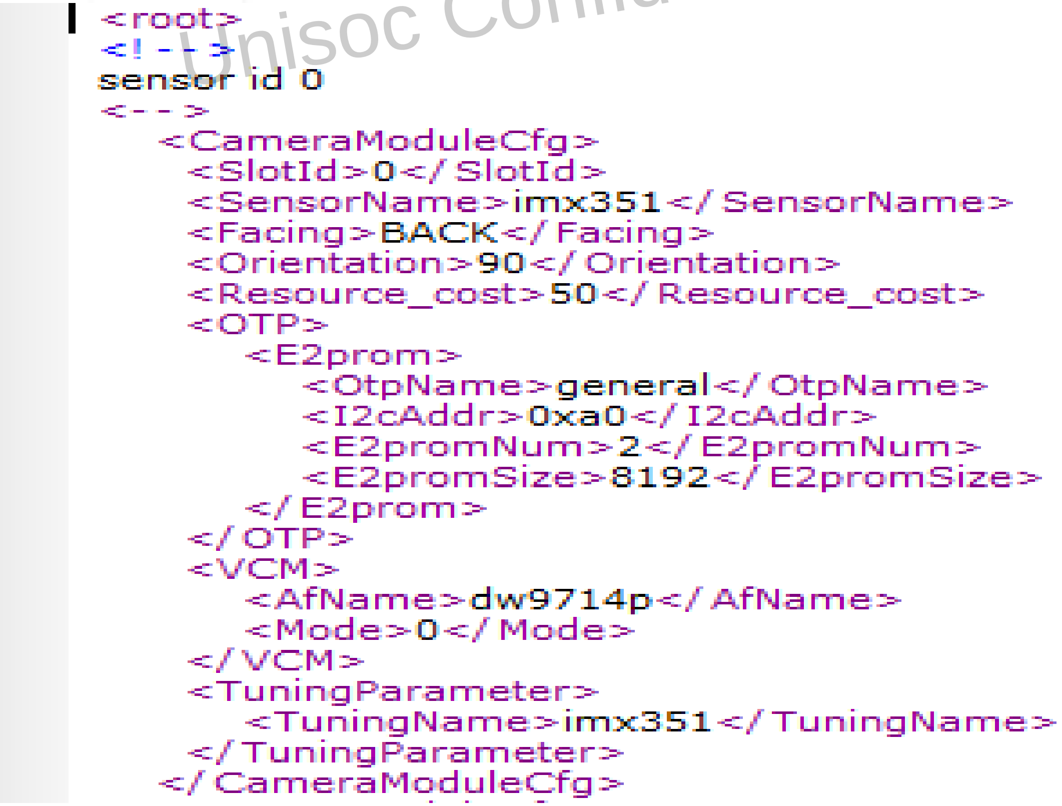
3 Camera驱动.c文件详解

4 Camera驱动总结

Camera驱动的基本配置及文件路径说明

这篇文章主要讲解展锐平台的camera sensor驱动代码设计详解。

首先，驱动的配置在路径：\device\sprd\xxxx\xxxx\camera\sensor_config.xml文件中如下视图

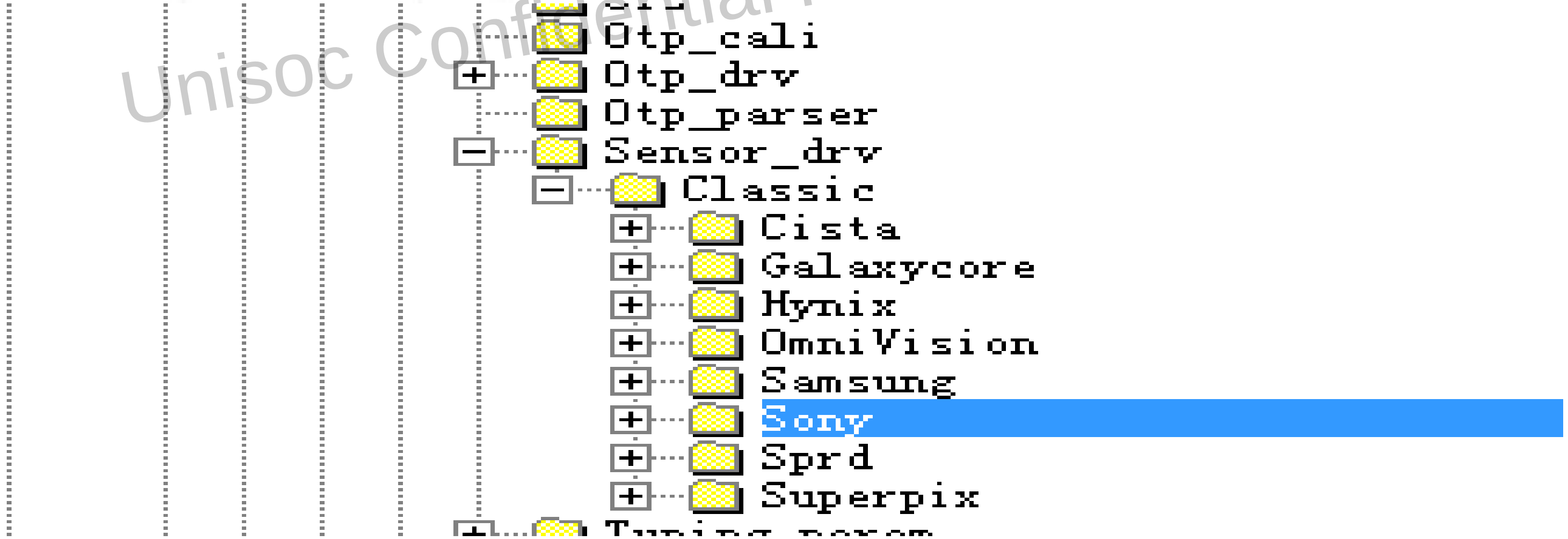


```
<root>
<!--
sensor id 0
-->
    <CameraModuleCfg>
      <SlotId>0</SlotId>
      <SensorName>imx351</SensorName>
      <Facing>BACK</Facing>
      <Orientation>90</Orientation>
      <Resource_cost>50</Resource_cost>
      <OTP>
          <E2prom>
              <OtpName>general</OtpName>
              <I2cAddr>0xa0</I2cAddr>
              <E2promNum>2</E2promNum>
              <E2promSize>8192</E2promSize>
          </E2prom>
      </OTP>
      <VCM>
          <AfName>dw9714p</AfName>
          <Mode>0</Mode>
      </VCM>
      <TuningParameter>
          <TuningName>imx351</TuningName>
      </TuningParameter>
    </CameraModuleCfg>
```

配置了sensor imx351，camera id 0，BACK表示朝后，90为竖屏的基础旋转角度，50为打开所占资源，general是平台端otp的驱动，0xa0表示i2c地址，2表示几个eeprom，8192表示eeprom的大小，vcm马达是dw9714，马达模式是0,参数文件夹是imx351.

Camera驱动的基本配置及文件路径说明

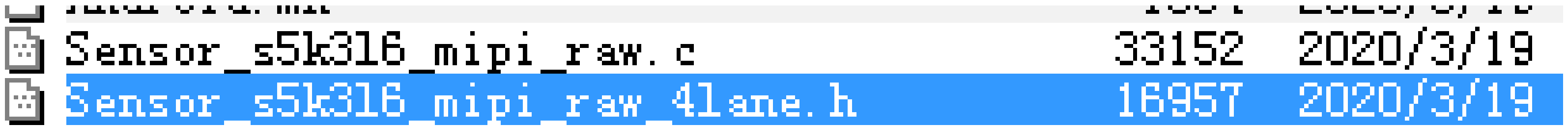
驱动代码的路径在：\vendor\sprd\modules\libcamera\sensor\sensor_drv\classic下，里面有各个sensor厂商的驱动文件夹，比如Galaxycore，Superpix，Samsung等



```
    Otp_cali
[+] Otp_drv
    Otp_parser
[-] Sensor_drv
    [-] Classic
        [+] Cista
        [+] Galaxycore
        [+] Hynix
        [+] OmniVision
        [+] Samsung
        [+] Sony
        [+] Sprd
        [+] Superpix
[+] Tuning_param
```

这份文档以ums512的后摄驱动s5k3l6为例子，简单介绍驱动中的各个配置及函数的作用

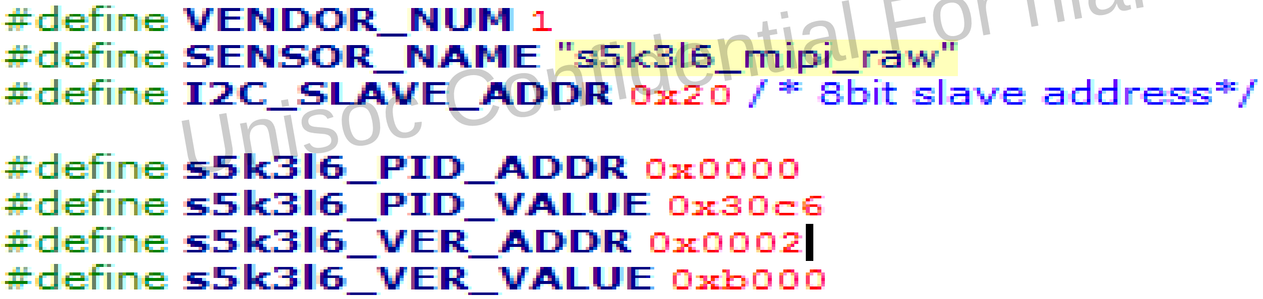
s5k3l6的驱动文件如下图所示，这个只是单纯的驱动，对于参数，将不会简介参数架构，因为参数修改不能使用文件修改，都需要使用工具修改。



```
Sensor_s5k3l6_mipi_raw.c          33152   2020/3/19
Sensor_s5k3l6_mipi_raw_4lane.h    16957   2020/3/19
```

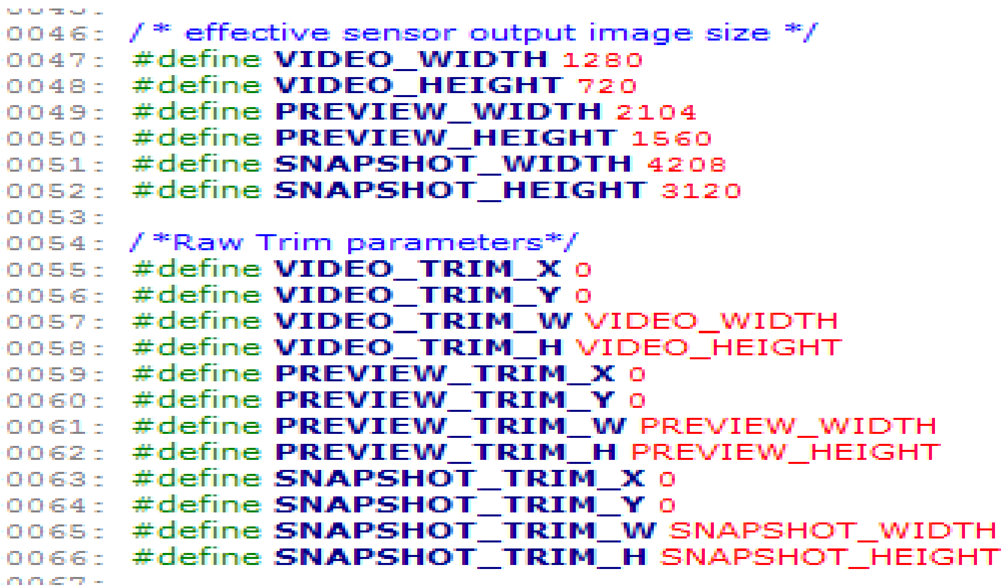
Camera驱动.h文件详解

驱动.h文件的配置讲解,直接对代码一一讲解：



```
#define VENDOR_NUM 1
#define SENSOR_NAME "s5k3l6_mipi_raw"
#define I2C_SLAVE_ADDR 0x20 /* 8bit slave address*/

#define s5k3l6_PID_ADDR 0x0000
#define s5k3l6_PID_VALUE 0x30c6
#define s5k3l6_VER_ADDR 0x0002
#define s5k3l6_VER_VALUE 0xb000
```

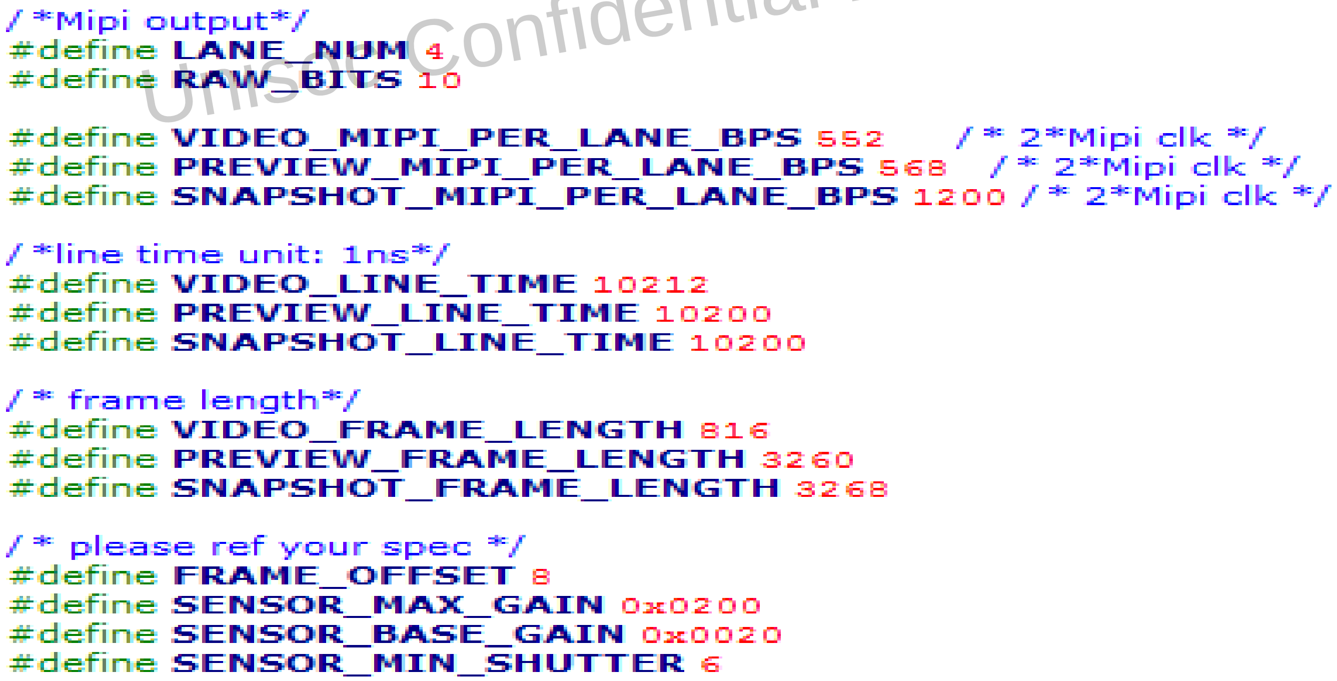
这个是VENDOR_NUM表示一个模组厂的配置，SENSOR_NAME表示打印语句的配置，I2C_SLAVE_ADDR表示sensor的i2c地址，记住，这里一般是8bit地址。



```
/* effective sensor output image size */
#define VIDEO_WIDTH 1280
#define VIDEO_HEIGHT 720
#define PREVIEW_WIDTH 2104
#define PREVIEW_HEIGHT 1560
#define SNAPSHOT_WIDTH 4208
#define SNAPSHOT_HEIGHT 3120

/*Raw Trim parameters*/
#define VIDEO_TRIM_X 0
#define VIDEO_TRIM_Y 0
#define VIDEO_TRIM_W VIDEO_WIDTH
#define VIDEO_TRIM_H VIDEO_HEIGHT
#define PREVIEW_TRIM_X 0
#define PREVIEW_TRIM_Y 0
#define PREVIEW_TRIM_W PREVIEW_WIDTH
#define PREVIEW_TRIM_H PREVIEW_HEIGHT
#define SNAPSHOT_TRIM_X 0
#define SNAPSHOT_TRIM_Y 0
#define SNAPSHOT_TRIM_W SNAPSHOT_WIDTH
#define SNAPSHOT_TRIM_H SNAPSHOT_HEIGHT
```

Camera驱动.h文件详解

这个是sensor的配置的尺寸，最大的是full size，preview使用的是binning size，1280X720是给slow motion使用

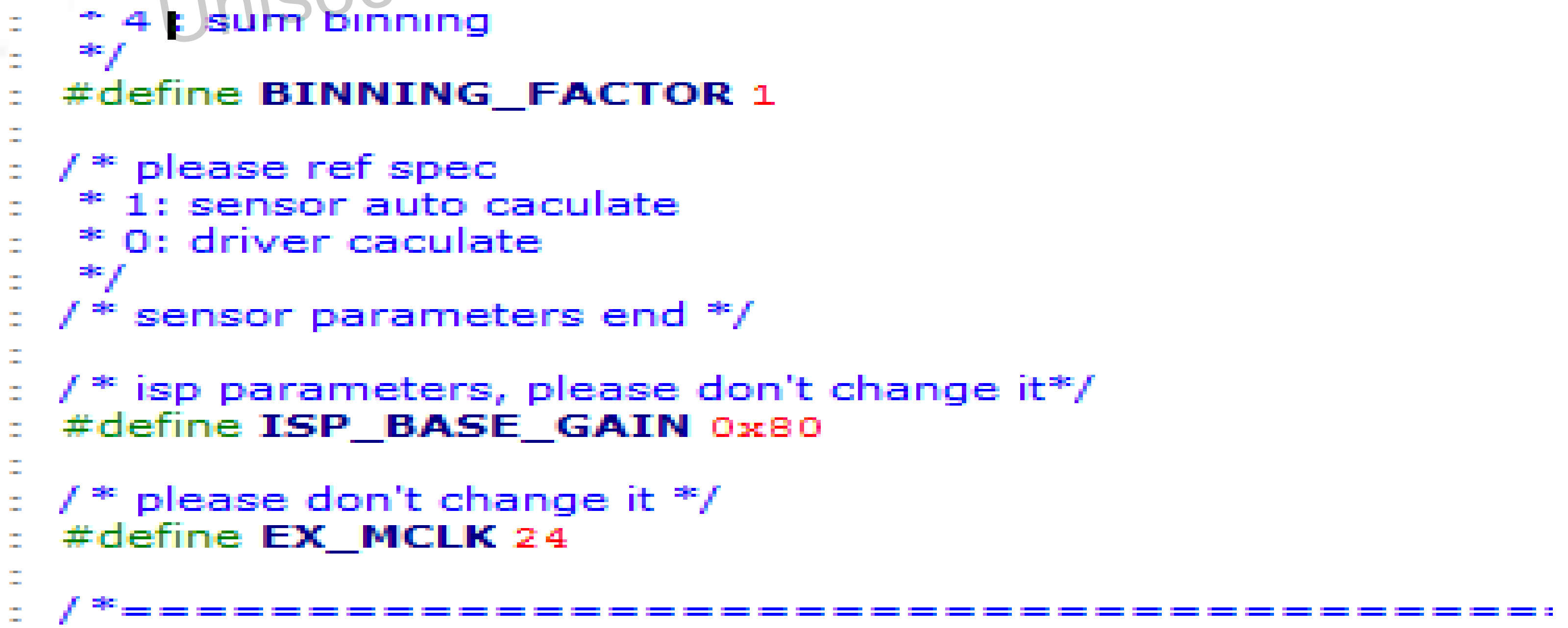


```
/*Mipi output*/
#define LANE_NUM 4
#define RAW_BITS 10

#define VIDEO_MIPI_PER_LANE_BPS 552    /* 2*Mipi clk */
#define PREVIEW_MIPI_PER_LANE_BPS 568  /* 2*Mipi clk */
#define SNAPSHOT_MIPI_PER_LANE_BPS 1200 /* 2*Mipi clk */

/*line time unit: 1ns*/
#define VIDEO_LINE_TIME 10212
#define PREVIEW_LINE_TIME 10200
#define SNAPSHOT_LINE_TIME 10200

/* frame length*/
#define VIDEO_FRAME_LENGTH 816
#define PREVIEW_FRAME_LENGTH 3260
#define SNAPSHOT_FRAME_LENGTH 3268

/* please ref your spec */
#define FRAME_OFFSET 8
#define SENSOR_MAX_GAIN 0x0200
#define SENSOR_BASE_GAIN 0x0020
#define SENSOR_MIN_SHUTTER 6
```

如上配置对sensor原厂的reg setting非常重要，一定要确认正确，Lane_num表示4组data lane，RAW_BITS表示sensor输出10bit的raw数据。接下来的三个bps是三个尺寸的bps，单位是M，接下来的三个line time是三个尺寸的linetime，这个需要原厂给出，要确保正确，单位是ns。在接下来的三个length是三个尺寸的frame length，这三个帧长也需要确认正确，正常情况下帧长乘linetime应该是那个尺寸reg setting的最大帧率的一帧的时间。

Camera驱动.h文件详解

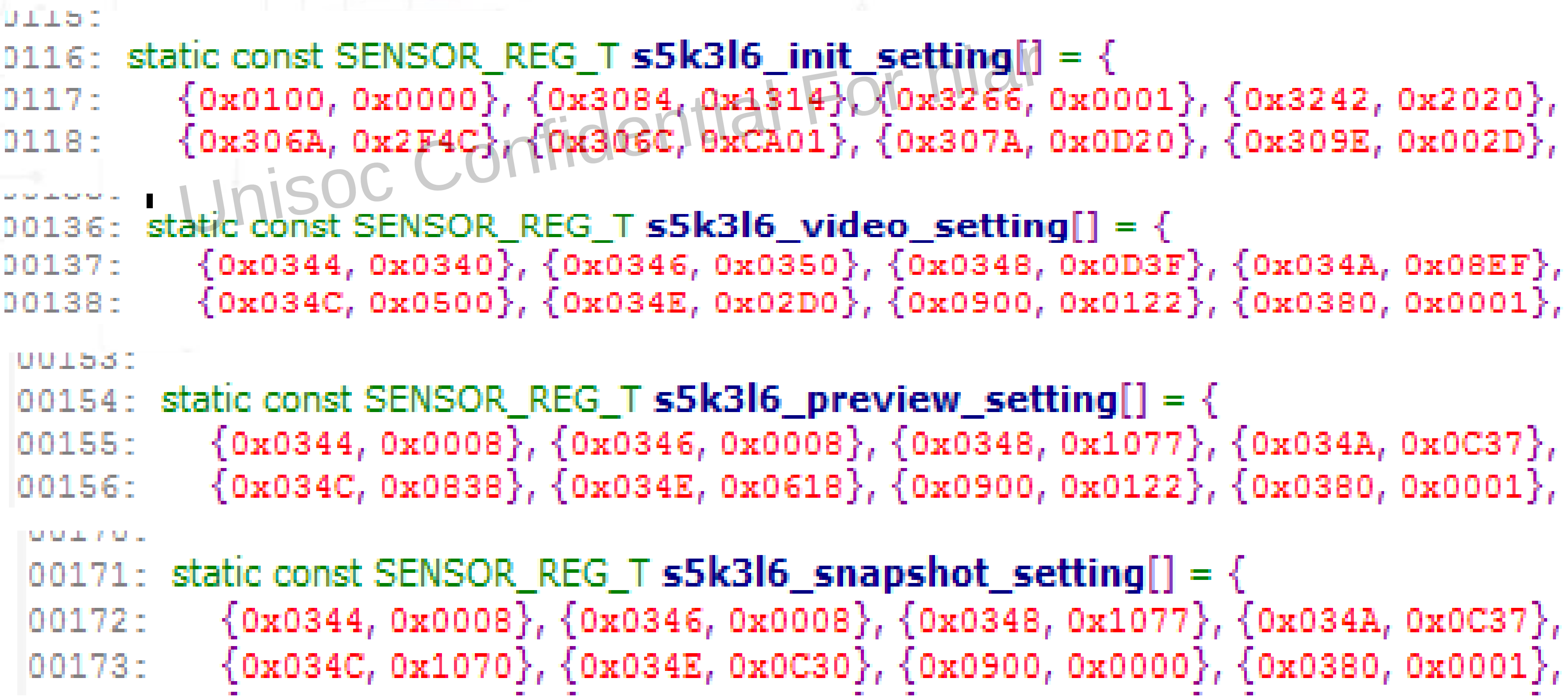
FRAME_OFFSET是帧偏，这个表示帧长和shutter的最小差值，需要原厂给出，属于sensor的特性。Base gain和max gain这个是sensor的gain可放大倍数，这个需要sensor原厂确认，后面进行写gain的函数需要用到。最后的最小曝光行也是需要sensor原厂给出。



```
  * 4 : sum binning
  */
#define BINNING_FACTOR 1

/* please ref spec
 * 1: sensor auto caculate
 * 0: driver caculate
 */
/* sensor parameters end */

/* isp parameters, please don't change it*/
#define ISP_BASE_GAIN 0x80

/* please don't change it */
#define EX_MCLK 24

/*======================================:
```

这个三个配置的的binning factor是为了shutter值的计算，主要是为了曝光时间full size和binning size的计算，如果是真实的binning，此值配置为1.另外，isp base gain为平台的128，EX_MCLK为mclk的配置，24的单位为M，但有个问题是，这里只有24是准的，其他的配置可能不准确，必须的使用示波器测试。

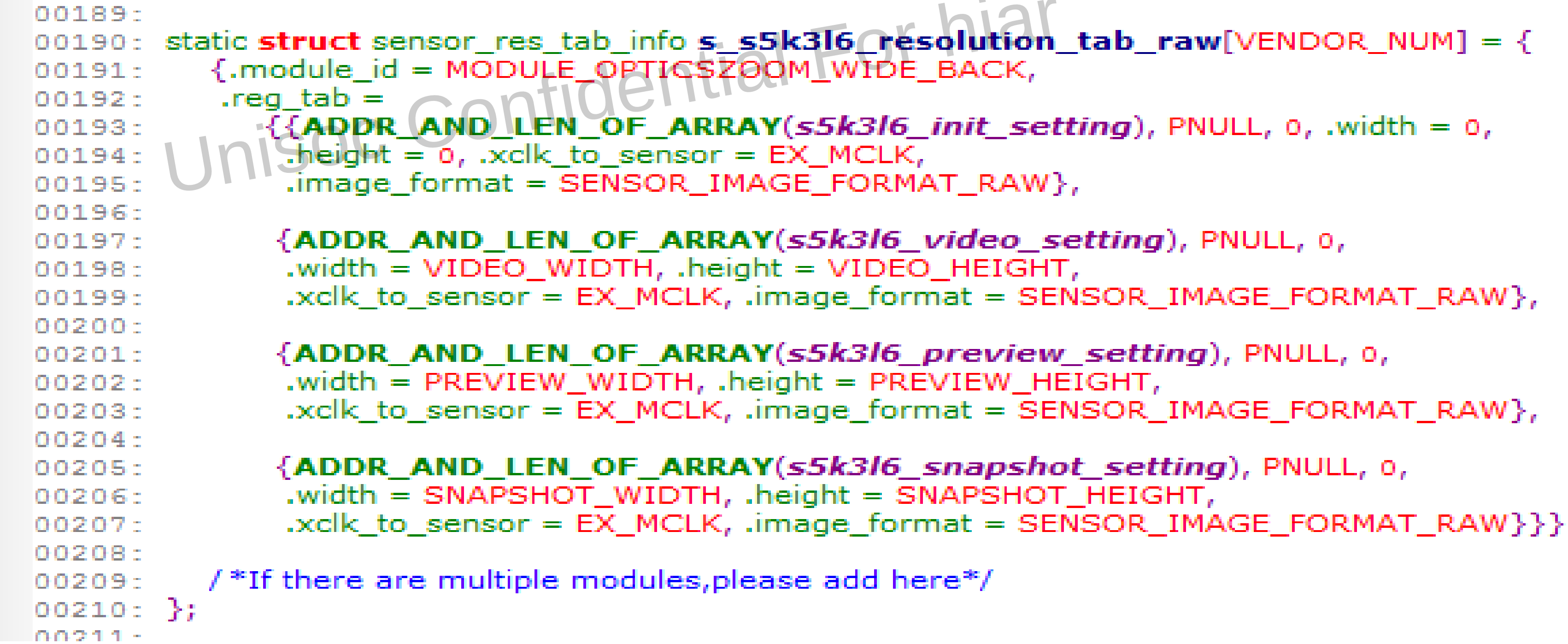
Camera驱动.h文件详解



```
static const SENSOR_REG_T s5k3l6_init_setting[] = {
    {0x0100, 0x0000}, {0x3084, 0x1314}, {0x3266, 0x0001}, {0x3242, 0x2020},
    {0x306A, 0x2F4C}, {0x306C, 0xCA01}, {0x307A, 0x0D20}, {0x309E, 0x002D},

static const SENSOR_REG_T s5k3l6_video_setting[] = {
    {0x0344, 0x0340}, {0x0346, 0x0350}, {0x0348, 0x0D3F}, {0x034A, 0x08EF},
    {0x034C, 0x0500}, {0x034E, 0x02D0}, {0x0900, 0x0122}, {0x0380, 0x0001},

static const SENSOR_REG_T s5k3l6_preview_setting[] = {
    {0x0344, 0x0008}, {0x0346, 0x0008}, {0x0348, 0x1077}, {0x034A, 0x0C37},
    {0x034C, 0x0838}, {0x034E, 0x0618}, {0x0900, 0x0122}, {0x0380, 0x0001},

static const SENSOR_REG_T s5k3l6_snapshot_setting[] = {
    {0x0344, 0x0008}, {0x0346, 0x0008}, {0x0348, 0x1077}, {0x034A, 0x0C37},
    {0x034C, 0x1070}, {0x034E, 0x0C30}, {0x0900, 0x0000}, {0x0380, 0x0001},
```

这四个寄存器数组分别代表为sensor的基本初始化配置，video尺寸模式的寄存器配置，preview binning尺寸模式的寄存器配置，snapshot full 尺寸模式的寄存器配置。

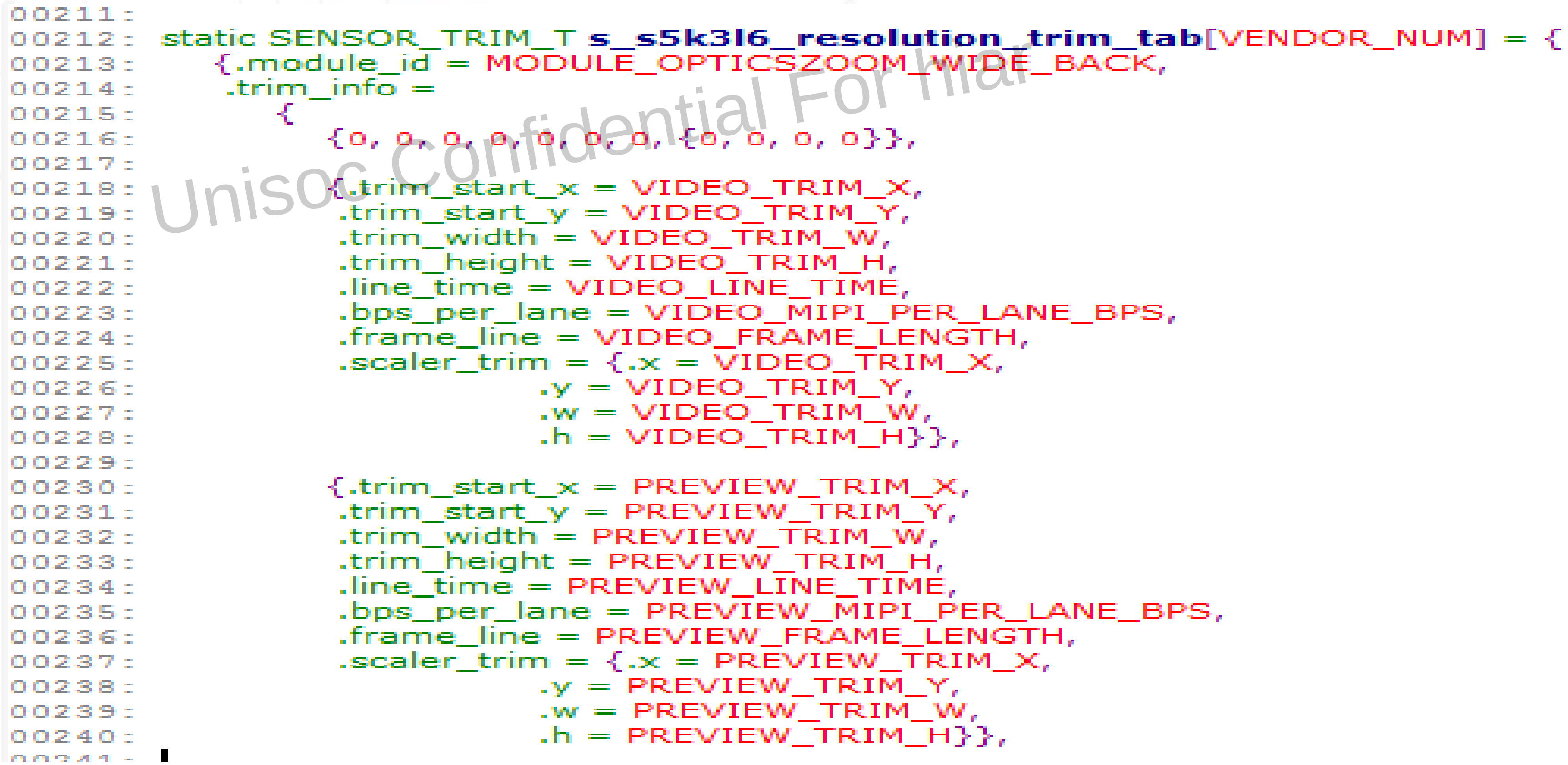
Camera驱动.h文件详解



```
static struct sensor_res_tab_info s_s5k3l6_resolution_tab_raw[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     .reg_tab =
        {{ADDR_AND_LEN_OF_ARRAY(s5k3l6_init_setting), PNULL, 0, .width = 0,
          .height = 0, .xclk_to_sensor = EX_MCLK,
          .image_format = SENSOR_IMAGE_FORMAT_RAW},

         {ADDR_AND_LEN_OF_ARRAY(s5k3l6_video_setting), PNULL, 0,
          .width = VIDEO_WIDTH, .height = VIDEO_HEIGHT,
          .xclk_to_sensor = EX_MCLK, .image_format = SENSOR_IMAGE_FORMAT_RAW},

         {ADDR_AND_LEN_OF_ARRAY(s5k3l6_preview_setting), PNULL, 0,
          .width = PREVIEW_WIDTH, .height = PREVIEW_HEIGHT,
          .xclk_to_sensor = EX_MCLK, .image_format = SENSOR_IMAGE_FORMAT_RAW},

         {ADDR_AND_LEN_OF_ARRAY(s5k3l6_snapshot_setting), PNULL, 0,
          .width = SNAPSHOT_WIDTH, .height = SNAPSHOT_HEIGHT,
          .xclk_to_sensor = EX_MCLK, .image_format = SENSOR_IMAGE_FORMAT_RAW}}}

    /*If there are multiple modules,please add here*/
};
```

这个在打开的时候，会将这个sensor相关的resolution配置获得给到oem层，当需要哪个尺寸的时候，就会通过尺寸相对应的进行模式的配置，需要说明一下的是，在open的时候，初始化会最先设置init setting,到了上层下发config尺寸匹配才会选择性的选择下面三个resolution配置。

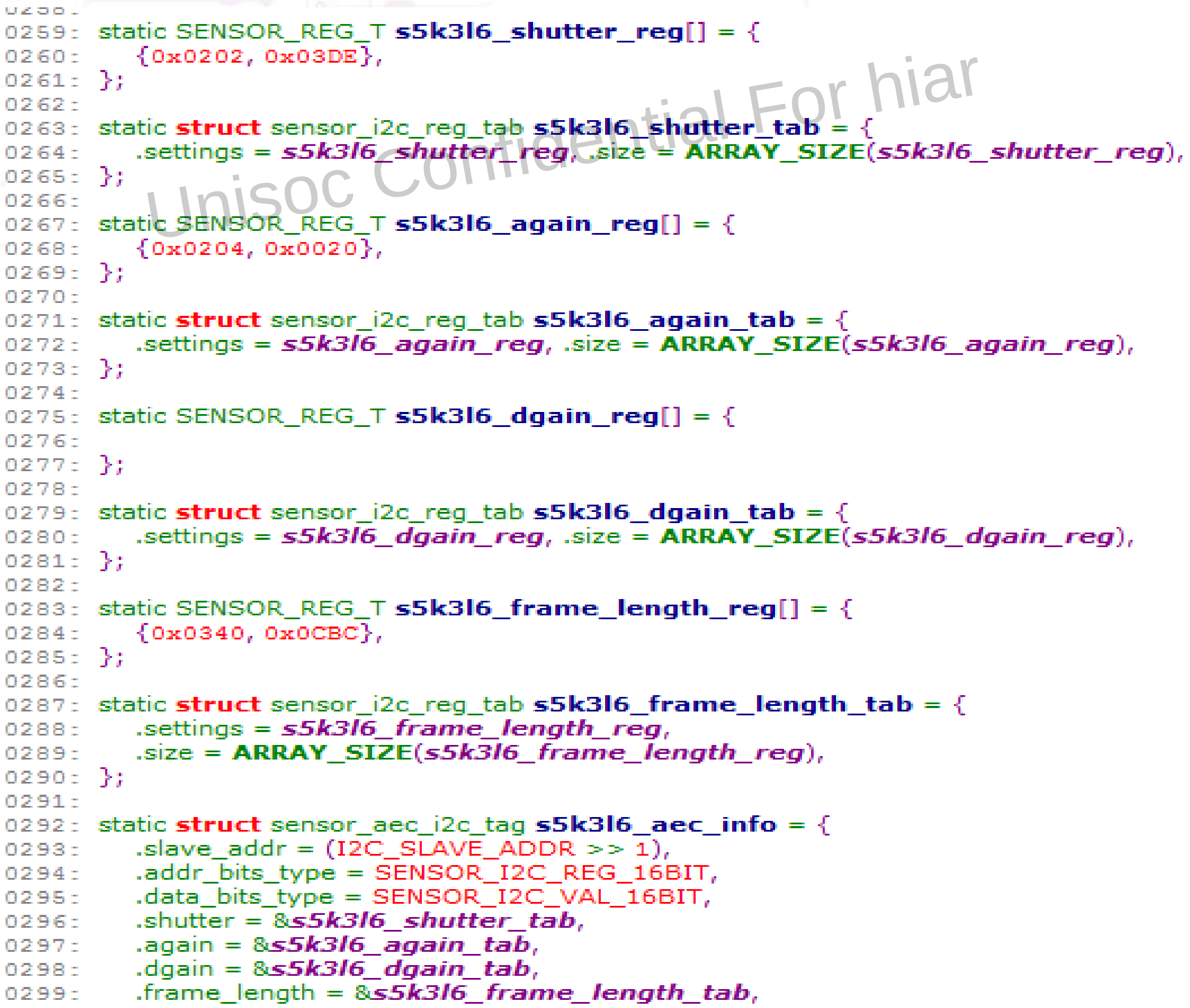
Camera驱动.h文件详解



```
static SENSOR_TRIM_T s_s5k3l6_resolution_trim_tab[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     .trim_info =
         {
             {0, 0, 0, 0, 0, 0, 0, {0, 0, 0, 0}},
             {.trim_start_x = VIDEO_TRIM_X,
              .trim_start_y = VIDEO_TRIM_Y,
              .trim_width = VIDEO_TRIM_W,
              .trim_height = VIDEO_TRIM_H,
              .line_time = VIDEO_LINE_TIME,
              .bps_per_lane = VIDEO_MIPI_PER_LANE_BPS,
              .frame_line = VIDEO_FRAME_LENGTH,
              .scaler_trim = {.x = VIDEO_TRIM_X,
                              .y = VIDEO_TRIM_Y,
                              .w = VIDEO_TRIM_W,
                              .h = VIDEO_TRIM_H}},

             {.trim_start_x = PREVIEW_TRIM_X,
              .trim_start_y = PREVIEW_TRIM_Y,
              .trim_width = PREVIEW_TRIM_W,
              .trim_height = PREVIEW_TRIM_H,
              .line_time = PREVIEW_LINE_TIME,
              .bps_per_lane = PREVIEW_MIPI_PER_LANE_BPS,
              .frame_line = PREVIEW_FRAME_LENGTH,
              .scaler_trim = {.x = PREVIEW_TRIM_X,
                              .y = PREVIEW_TRIM_Y,
                              .w = PREVIEW_TRIM_W,
                              .h = PREVIEW_TRIM_H}},
```

这个是各个模式下sensor这边对dcam的接口配置，需要配置到平台dcam的寄存器，Trim跟跟上面的resolution最好是需要一一对应，第一组为什么会是全0,那是因为init setting不是真正出数据，所以不需要配置DCAM，其中前面trim_start_x等为dcam从sensor crop的起始点和尺寸，Line_time也需要对应尺寸的line time，bps也是对应尺寸的bps，这些前面已经讲过，为原厂提供，frame length也是需相对应的帧长。最后的scaler为从trim那边进行crop的尺寸
。

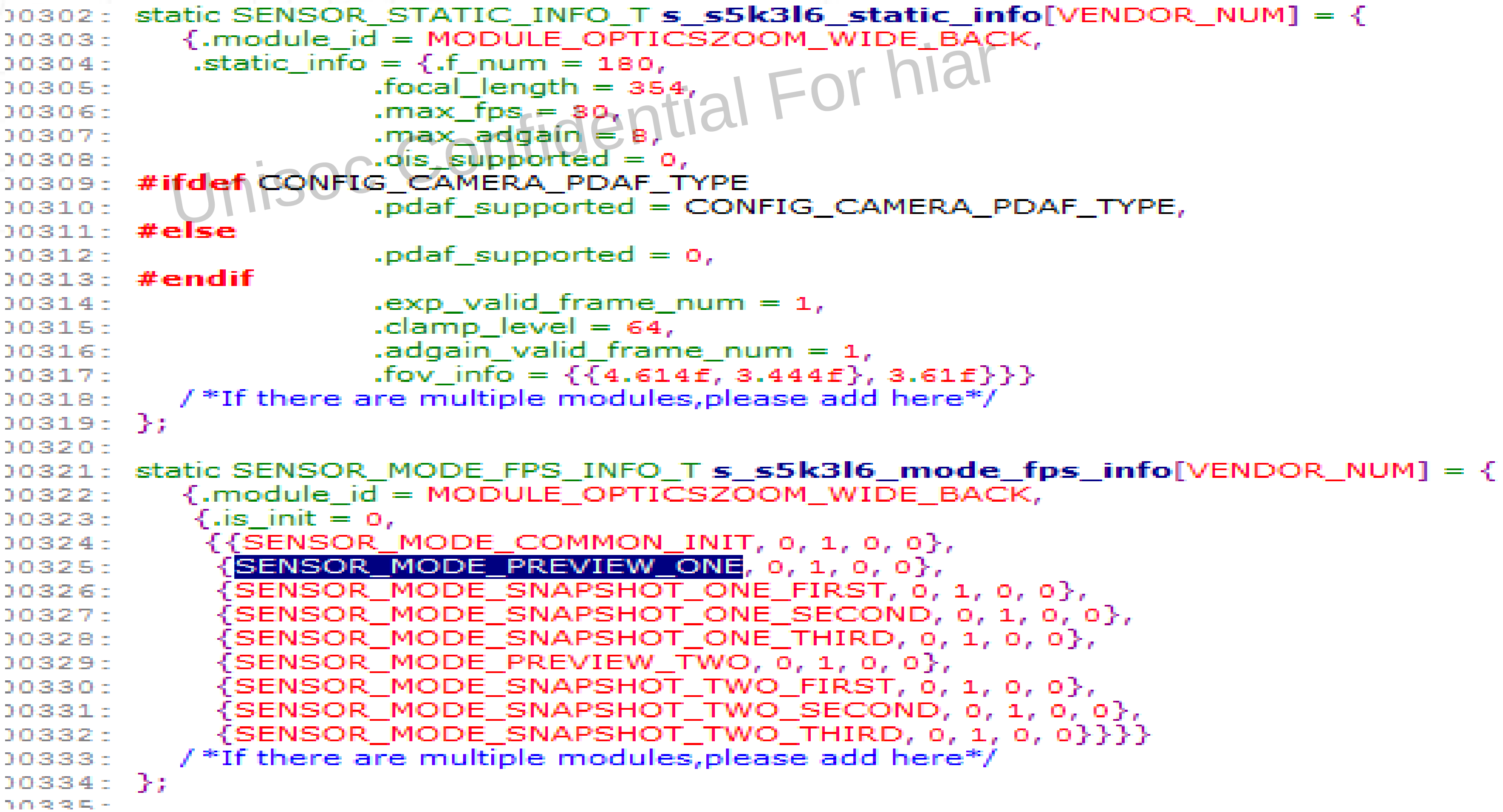
Camera驱动.h文件详解



```
static SENSOR_REG_T s5k3l6_shutter_reg[] = {
    {0x0202, 0x03DE},
};

static struct sensor_i2c_reg_tab s5k3l6_shutter_tab = {
    .settings = s5k3l6_shutter_reg, .size = ARRAY_SIZE(s5k3l6_shutter_reg),
};

static SENSOR_REG_T s5k3l6_again_reg[] = {
    {0x0204, 0x0020},
};

static struct sensor_i2c_reg_tab s5k3l6_again_tab = {
    .settings = s5k3l6_again_reg, .size = ARRAY_SIZE(s5k3l6_again_reg),
};

static SENSOR_REG_T s5k3l6_dgain_reg[] = {

};

static struct sensor_i2c_reg_tab s5k3l6_dgain_tab = {
    .settings = s5k3l6_dgain_reg, .size = ARRAY_SIZE(s5k3l6_dgain_reg),
};

static SENSOR_REG_T s5k3l6_frame_length_reg[] = {
    {0x0340, 0x0CBC},
};

static struct sensor_i2c_reg_tab s5k3l6_frame_length_tab = {
    .settings = s5k3l6_frame_length_reg,
    .size = ARRAY_SIZE(s5k3l6_frame_length_reg),
};

static struct sensor_aec_i2c_tag s5k3l6_aec_info = {
    .slave_addr = (I2C_SLAVE_ADDR >> 1),
    .addr_bits_type = SENSOR_I2C_REG_16BIT,
    .data_bits_type = SENSOR_I2C_VAL_16BIT,
    .shutter = &s5k3l6_shutter_tab,
    .again = &s5k3l6_again_tab,
    .dgain = &s5k3l6_dgain_tab,
    .frame_length = &s5k3l6_frame_length_tab,
```

这个是sensor的again，dgain，shutter值寄存器及framelength寄存器配置，在后面的.c文件中的函数需要用到，这里特别需要注意的是这个里面的初始值一定要是正确的，可以前面的宏定义及跟setting中的对应，如果setting中没有配置，这里的配置需要确认好正确。。

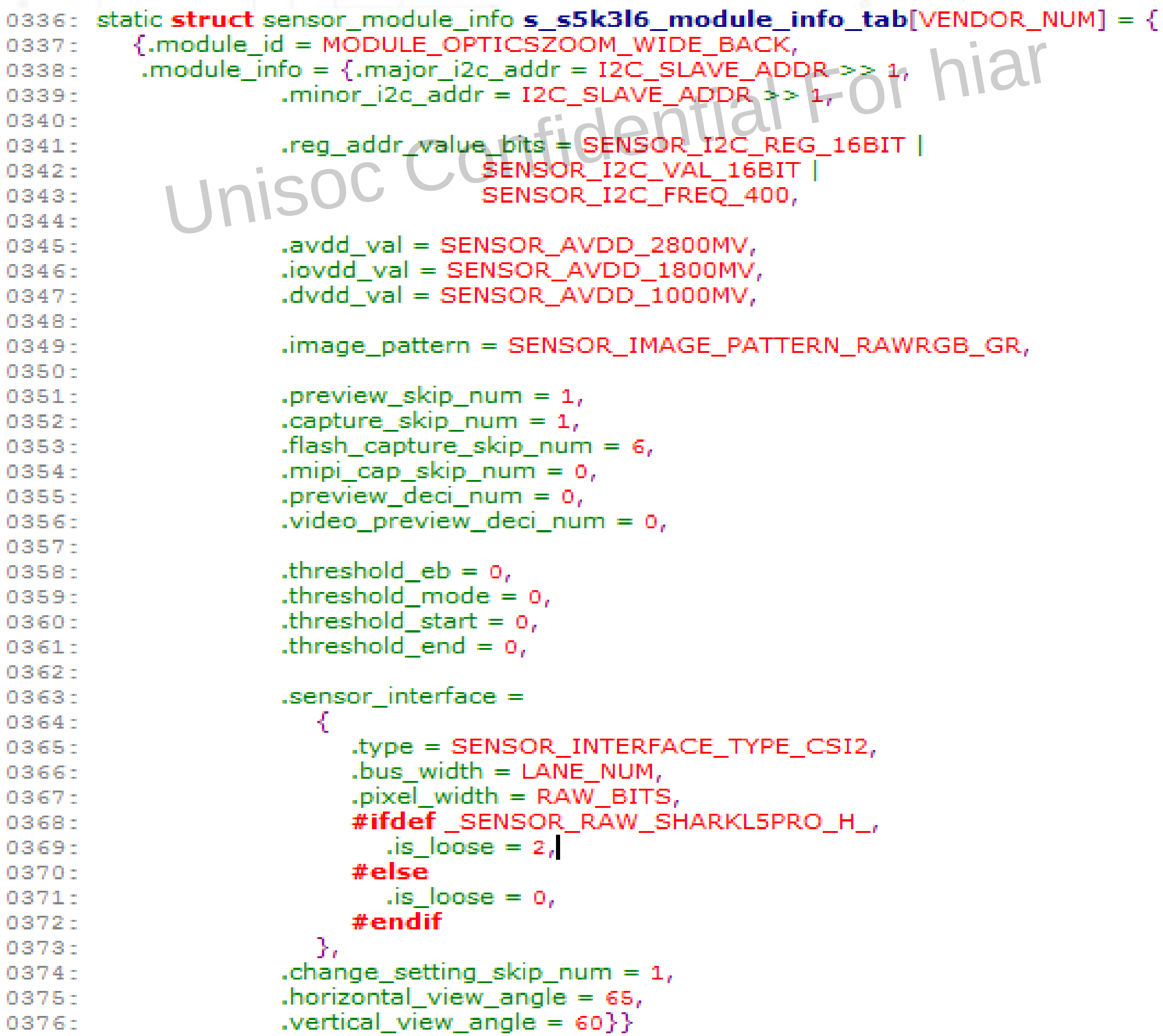
Camera驱动.h文件详解



```
static SENSOR_STATIC_INFO_T s_s5k3l6_static_info[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     .static_info = {.f_num = 180,
                     .focal_length = 354,
                     .max_fps = 30,
                     .max_adgain = 8,
                     .ois_supported = 0,
#ifdef CONFIG_CAMERA_PDAF_TYPE
                     .pdaf_supported = CONFIG_CAMERA_PDAF_TYPE,
#else
                     .pdaf_supported = 0,
#endif
                     .exp_valid_frame_num = 1,
                     .clamp_level = 64,
                     .adgain_valid_frame_num = 1,
                     .fov_info = {{4.614f, 3.444f}, 3.61f}}}
    /*If there are multiple modules,please add here*/
};

static SENSOR_MODE_FPS_INFO_T s_s5k3l6_mode_fps_info[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     {.is_init = 0,
      {{SENSOR_MODE_COMMON_INIT, 0, 1, 0, 0},
       {SENSOR_MODE_PREVIEW_ONE, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_ONE_FIRST, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_ONE_SECOND, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_ONE_THIRD, 0, 1, 0, 0},
       {SENSOR_MODE_PREVIEW_TWO, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_TWO_FIRST, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_TWO_SECOND, 0, 1, 0, 0},
       {SENSOR_MODE_SNAPSHOT_TWO_THIRD, 0, 1, 0, 0}}}}
    /*If there are multiple modules,please add here*/
};
```

第一个结构体主要是一些sensor模组的静态信息，比如光圈大小，焦距长度，fov，如果有pdaf需要在这里配置类型等，fov也是在这里配置，但帧率因为还有其他函数获得，这里理论上是没有效的。第二个结构体主要是配置高帧率相关，设置标志量，主要是给到算法使用

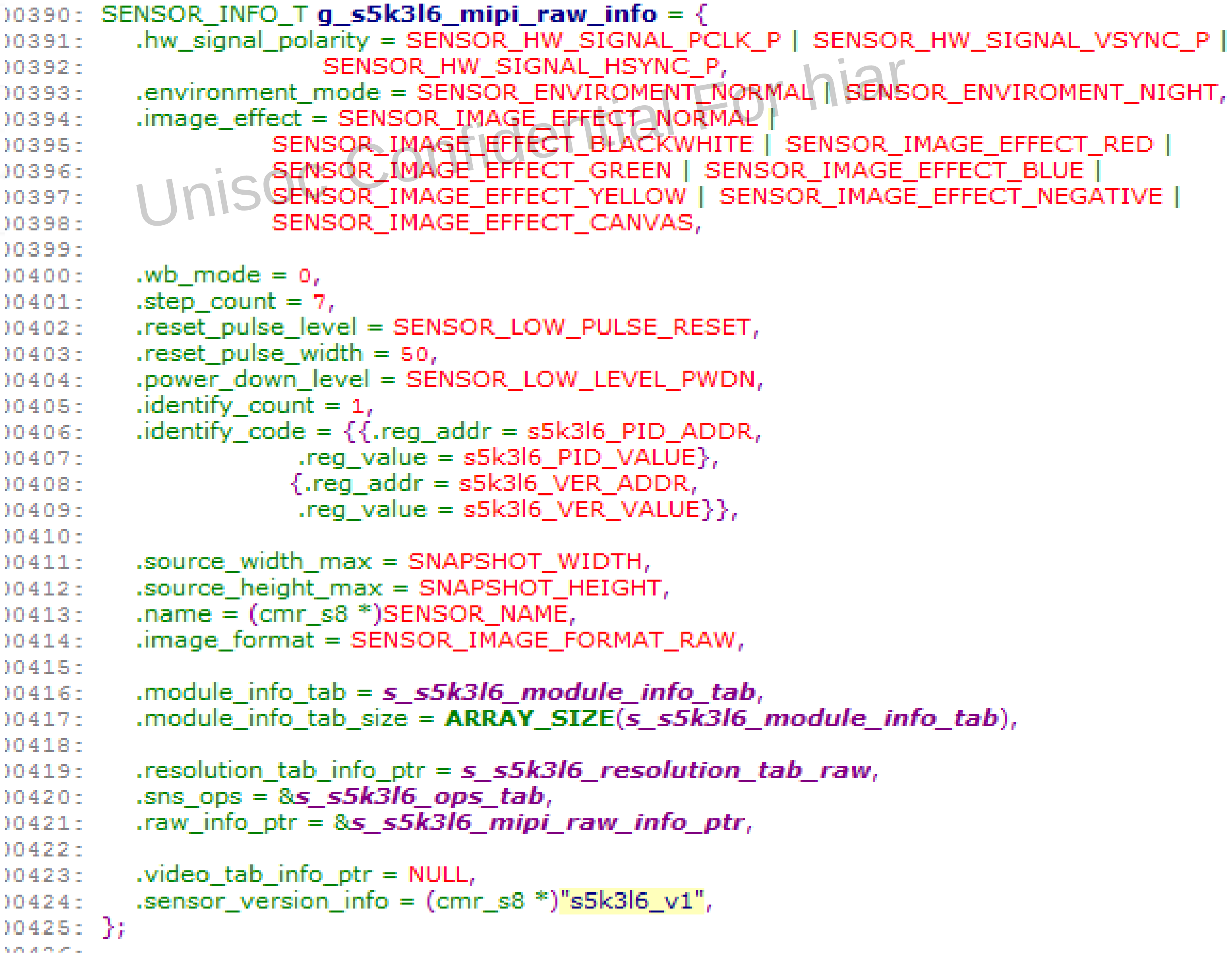
Camera驱动.h文件详解



```
static struct sensor_module_info s_s5k3l6_module_info_tab[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     .module_info = {.major_i2c_addr = I2C_SLAVE_ADDR >> 1,
                     .minor_i2c_addr = I2C_SLAVE_ADDR >> 1,

                     .reg_addr_value_bits = SENSOR_I2C_REG_16BIT |
                                            SENSOR_I2C_VAL_16BIT |
                                            SENSOR_I2C_FREQ_400,

                     .avdd_val = SENSOR_AVDD_2800MV,
                     .iovdd_val = SENSOR_AVDD_1800MV,
                     .dvdd_val = SENSOR_AVDD_1000MV,

                     .image_pattern = SENSOR_IMAGE_PATTERN_RAWRGB_GR,

                     .preview_skip_num = 1,
                     .capture_skip_num = 1,
                     .flash_capture_skip_num = 6,
                     .mipi_cap_skip_num = 0,
                     .preview_deci_num = 0,
                     .video_preview_deci_num = 0,

                     .threshold_eb = 0,
                     .threshold_mode = 0,
                     .threshold_start = 0,
                     .threshold_end = 0,

                     .sensor_interface =
                        {
                            .type = SENSOR_INTERFACE_TYPE_CSI2,
                            .bus_width = LANE_NUM,
                            .pixel_width = RAW_BITS,
                            #ifdef _SENSOR_RAW_SHARKL5PRO_H_,
                                .is_loose = 2,
                            #else
                                .is_loose = 0,
                            #endif
                        },
                     .change_setting_skip_num = 1,
                     .horizontal_view_angle = 65,
                     .vertical_view_angle = 60}}
```

这个结构体是sensor的电气相关的主要结构体，针对平台进行配置，首先是module_id，这个是需要模组的区分，也是广角等逻辑id的驱动配置，另外的话是i2c地址配置，这里需要注意的是这个地址是7bit地址，另外就是三路电源，及rgb顺序。下面的其他配置基本不能修改，包括预览拍照丢帧，下面的配置可以参考使用默认值，不需要修改，因为修改可能导致一些意想不到的错误，从而使问题查找困难。另外需要注意的是如果是raw14的raw图抓取，需要配置is_loose =2.

Camera驱动.h文件详解



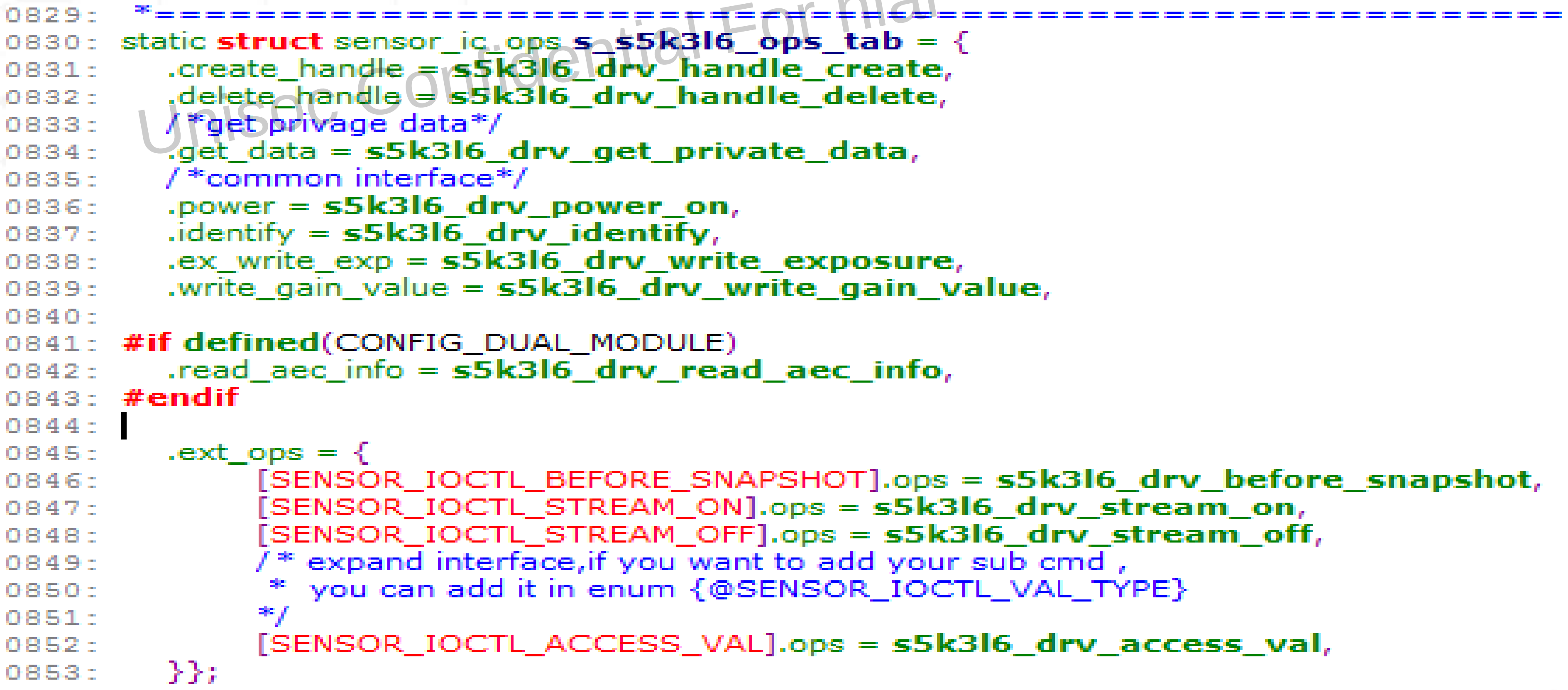
```
SENSOR_INFO_T g_s5k3l6_mipi_raw_info = {
    .hw_signal_polarity = SENSOR_HW_SIGNAL_PCLK_P | SENSOR_HW_SIGNAL_VSYNC_P |
                          SENSOR_HW_SIGNAL_HSYNC_P,
    .environment_mode = SENSOR_ENVIROMENT_NORMAL | SENSOR_ENVIROMENT_NIGHT,
    .image_effect = SENSOR_IMAGE_EFFECT_NORMAL |
                    SENSOR_IMAGE_EFFECT_BLACKWHITE | SENSOR_IMAGE_EFFECT_RED |
                    SENSOR_IMAGE_EFFECT_GREEN | SENSOR_IMAGE_EFFECT_BLUE |
                    SENSOR_IMAGE_EFFECT_YELLOW | SENSOR_IMAGE_EFFECT_NEGATIVE |
                    SENSOR_IMAGE_EFFECT_CANVAS,

    .wb_mode = 0,
    .step_count = 7,
    .reset_pulse_level = SENSOR_LOW_PULSE_RESET,
    .reset_pulse_width = 50,
    .power_down_level = SENSOR_LOW_LEVEL_PWDN,
    .identify_count = 1,
    .identify_code = {{.reg_addr = s5k3l6_PID_ADDR,
                       .reg_value = s5k3l6_PID_VALUE},
                      {.reg_addr = s5k3l6_VER_ADDR,
                       .reg_value = s5k3l6_VER_VALUE}},

    .source_width_max = SNAPSHOT_WIDTH,
    .source_height_max = SNAPSHOT_HEIGHT,
    .name = (cmr_s8 *)SENSOR_NAME,
    .image_format = SENSOR_IMAGE_FORMAT_RAW,

    .module_info_tab = s_s5k3l6_module_info_tab,
    .module_info_tab_size = ARRAY_SIZE(s_s5k3l6_module_info_tab),

    .resolution_tab_info_ptr = s_s5k3l6_resolution_tab_raw,
    .sns_ops = &s_s5k3l6_ops_tab,
    .raw_info_ptr = &s_s5k3l6_mipi_raw_info_ptr,

    .video_tab_info_ptr = NULL,
    .sensor_version_info = (cmr_s8 *)"s5k3l6_v1",
};
```

Camera驱动.h文件详解

这个结构体是驱动的灵魂，是对sensor的一些特性配置，包括帧行pclk的极性，环境模式，图像的影响等，这些都是默认配置，不建议做任何修改。需要注意的是reset和power down的配置，需要和power_on函数中对应起来，拉高拉低需要对应，还有就是full size，这个也需要配置正确，因为这个位置是auto detect的尺寸大小。后面的就是对其结构体和函数接口的指针，所有的接口都在这个结构体中。

Camera驱动.c文件详解

对于.c中的函数接口，里面的所有函数都是如下结构体函数指针对应的函数接口

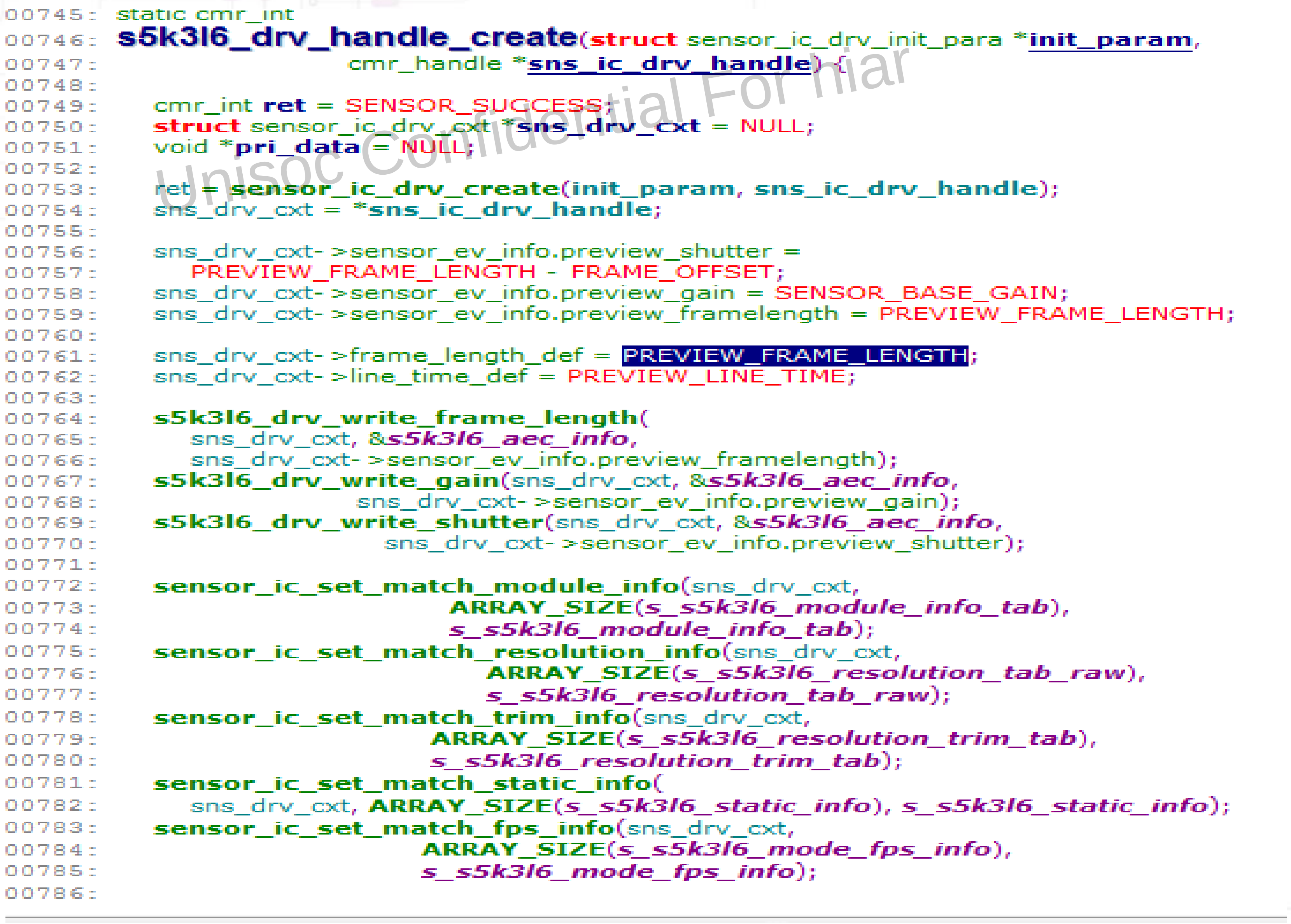


```
 *==================================================================================
static struct sensor_ic_ops s_s5k3l6_ops_tab = {
    .create_handle = s5k3l6_drv_handle_create,
    .delete_handle = s5k3l6_drv_handle_delete,
    /*get privage data*/
    .get_data = s5k3l6_drv_get_private_data,
    /*common interface*/
    .power = s5k3l6_drv_power_on,
    .identify = s5k3l6_drv_identify,
    .ex_write_exp = s5k3l6_drv_write_exposure,
    .write_gain_value = s5k3l6_drv_write_gain_value,

#if defined(CONFIG_DUAL_MODULE)
    .read_aec_info = s5k3l6_drv_read_aec_info,
#endif

    .ext_ops = {
            [SENSOR_IOCTL_BEFORE_SNAPSHOT].ops = s5k3l6_drv_before_snapshot,
            [SENSOR_IOCTL_STREAM_ON].ops = s5k3l6_drv_stream_on,
            [SENSOR_IOCTL_STREAM_OFF].ops = s5k3l6_drv_stream_off,
            /* expand interface,if you want to add your sub cmd ,
             *  you can add it in enum {@SENSOR_IOCTL_VAL_TYPE}
             */
            [SENSOR_IOCTL_ACCESS_VAL].ops = s5k3l6_drv_access_val,
    }};
```

从结构体的前三个函数说起：

对于create这个函数基本会最开始调用到，比如开机进行identify的时候就会调用到，对整个驱动进行初始化配置，分配一个驱动所需要的包括各种结构体的内存空间等。

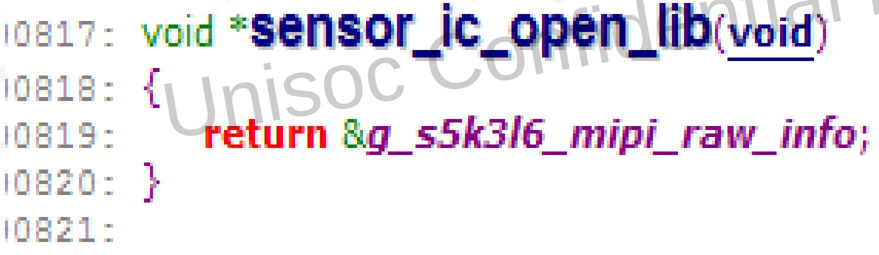
Camera驱动.c文件详解



```
static cmr_int
s5k3l6_drv_handle_create(struct sensor_ic_drv_init_para *init_param,
                         cmr_handle *sns_ic_drv_handle) {

    cmr_int ret = SENSOR_SUCCESS;
    struct sensor_ic_drv_cxt *sns_drv_cxt = NULL;
    void *pri_data = NULL;

    ret = sensor_ic_drv_create(init_param, sns_ic_drv_handle);
    sns_drv_cxt = *sns_ic_drv_handle;

    sns_drv_cxt->sensor_ev_info.preview_shutter =
        PREVIEW_FRAME_LENGTH - FRAME_OFFSET;
    sns_drv_cxt->sensor_ev_info.preview_gain = SENSOR_BASE_GAIN;
    sns_drv_cxt->sensor_ev_info.preview_framelength = PREVIEW_FRAME_LENGTH;

    sns_drv_cxt->frame_length_def = PREVIEW_FRAME_LENGTH;
    sns_drv_cxt->line_time_def = PREVIEW_LINE_TIME;

    s5k3l6_drv_write_frame_length(
        sns_drv_cxt, &s5k3l6_aec_info,
        sns_drv_cxt->sensor_ev_info.preview_framelength);
    s5k3l6_drv_write_gain(sns_drv_cxt, &s5k3l6_aec_info,
                          sns_drv_cxt->sensor_ev_info.preview_gain);
    s5k3l6_drv_write_shutter(sns_drv_cxt, &s5k3l6_aec_info,
                             sns_drv_cxt->sensor_ev_info.preview_shutter);

    sensor_ic_set_match_module_info(sns_drv_cxt,
                                    ARRAY_SIZE(s_s5k3l6_module_info_tab),
                                    s_s5k3l6_module_info_tab);
    sensor_ic_set_match_resolution_info(sns_drv_cxt,
                                        ARRAY_SIZE(s_s5k3l6_resolution_tab_raw),
                                        s_s5k3l6_resolution_tab_raw);
    sensor_ic_set_match_trim_info(sns_drv_cxt,
                                  ARRAY_SIZE(s_s5k3l6_resolution_trim_tab),
                                  s_s5k3l6_resolution_trim_tab);
    sensor_ic_set_match_static_info(
        sns_drv_cxt, ARRAY_SIZE(s_s5k3l6_static_info), s_s5k3l6_static_info);
    sensor_ic_set_match_fps_info(sns_drv_cxt,
                                 ARRAY_SIZE(s_s5k3l6_mode_fps_info),
                                 s_s5k3l6_mode_fps_info);

```

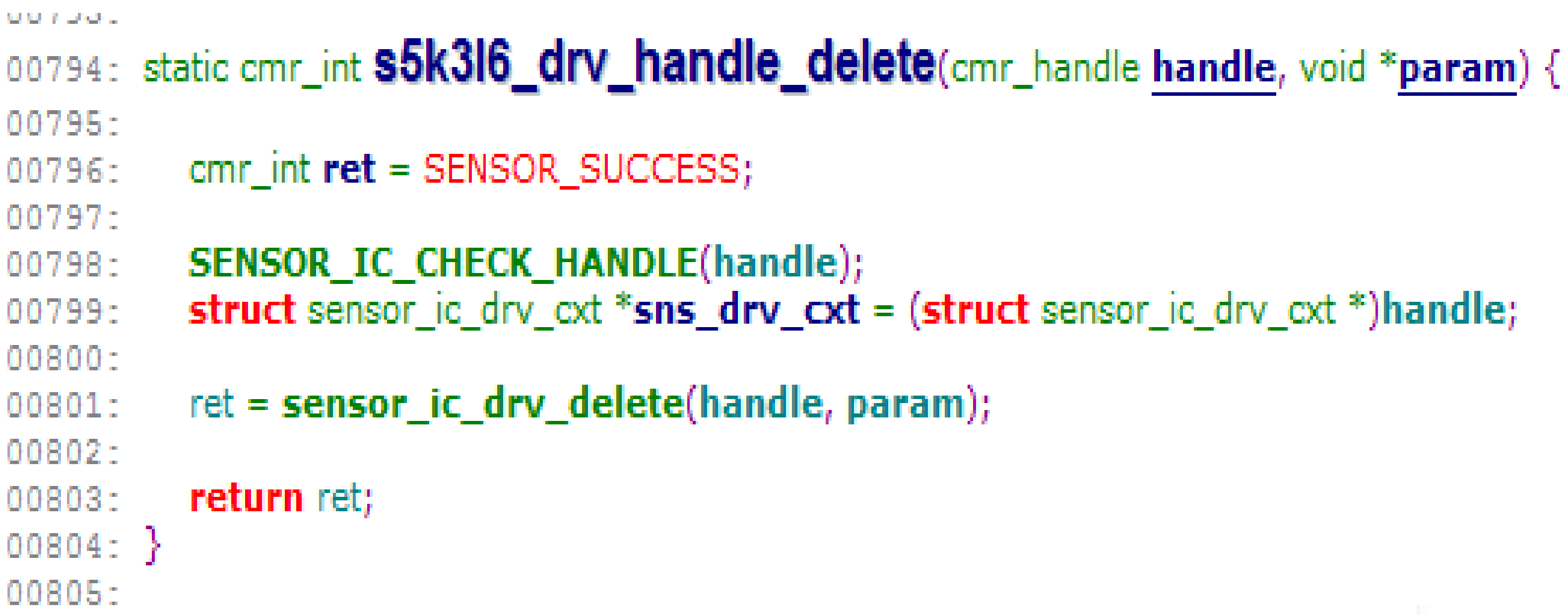
Camera驱动.c文件详解

当然实际上还有如下函数会在调用库的时候调用到：



```
void *sensor_ic_open_lib(void)
{
    return &g_s5k3l6_mipi_raw_info;
}

```

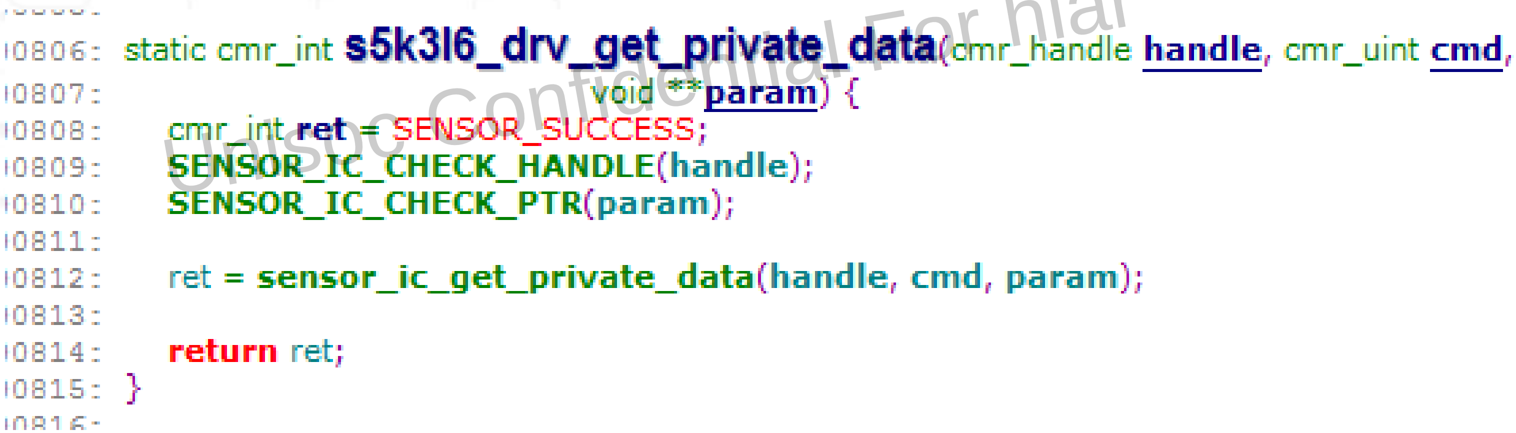
如下函数主要是在退出的时候，将前面分配的内存等进行销毁释放。



```
static cmr_int s5k3l6_drv_handle_delete(cmr_handle handle, void *param) {

    cmr_int ret = SENSOR_SUCCESS;

    SENSOR_IC_CHECK_HANDLE(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    ret = sensor_ic_drv_delete(handle, param);

    return ret;
}

```

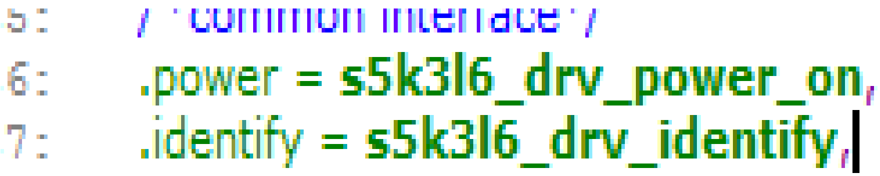
Camera驱动.c文件详解

如下函数是对sensor在.h文件中配置的一些私有信息传递给oem，在需要的使用，包括模式信息等：



```
static cmr_int s5k3l6_drv_get_private_data(cmr_handle handle, cmr_uint cmd,
                                           void **param) {
    cmr_int ret = SENSOR_SUCCESS;
    SENSOR_IC_CHECK_HANDLE(handle);
    SENSOR_IC_CHECK_PTR(param);

    ret = sensor_ic_get_private_data(handle, cmd, param);

    return ret;
}
```

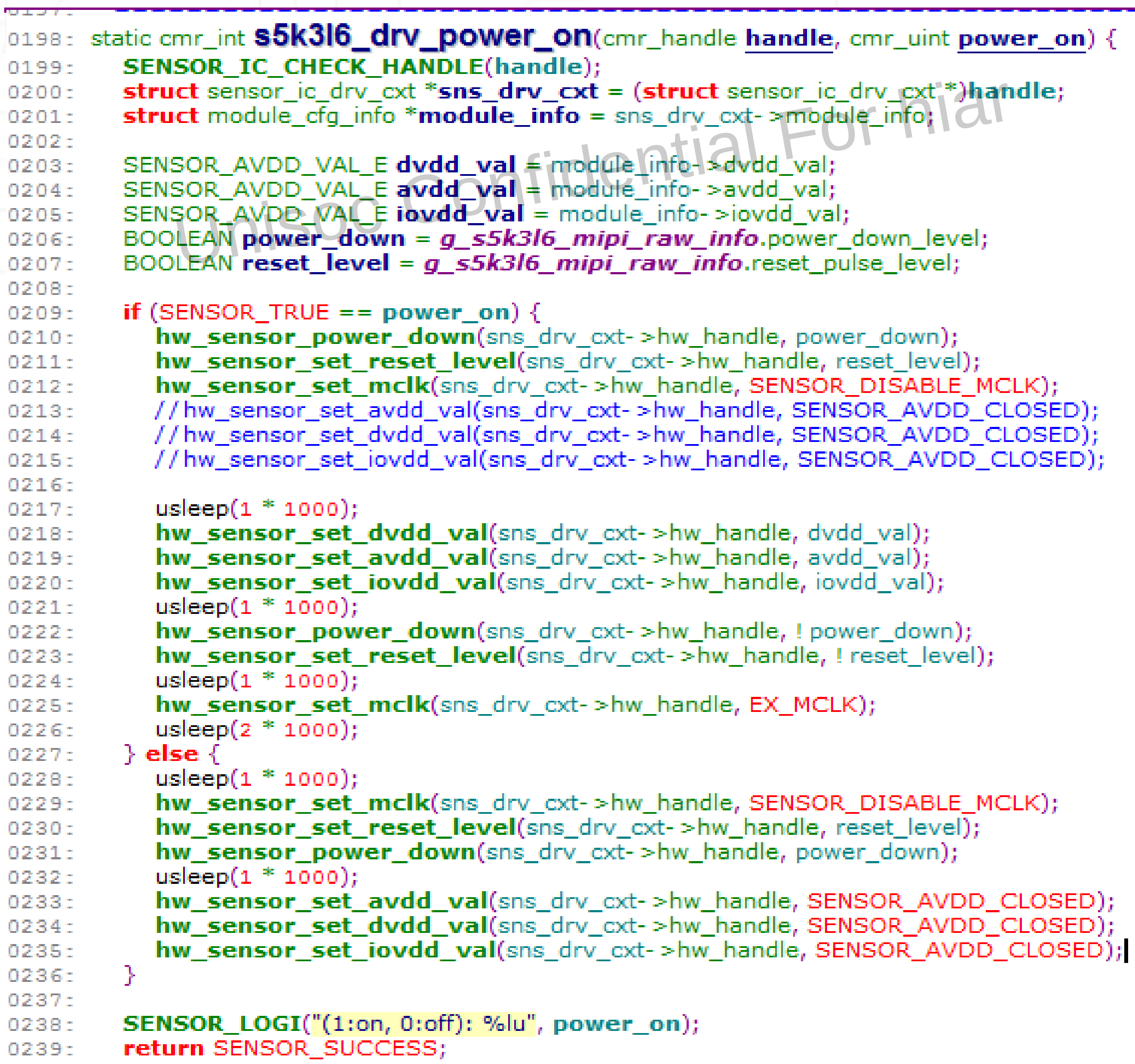
对于如下两个函数，使用的频率很高，最近才需要调试的，power on是上电和下电时序，这个应该要使用示波器测量让其跟datasheet的上下电时序对应。Identify是上电后对sensor的pid vid的识别。



```
    /* common interface */
    .power = s5k3l6_drv_power_on,
    .identify = s5k3l6_drv_identify,
```

这两个函数需要详细说明一下，首先看一下power_on:

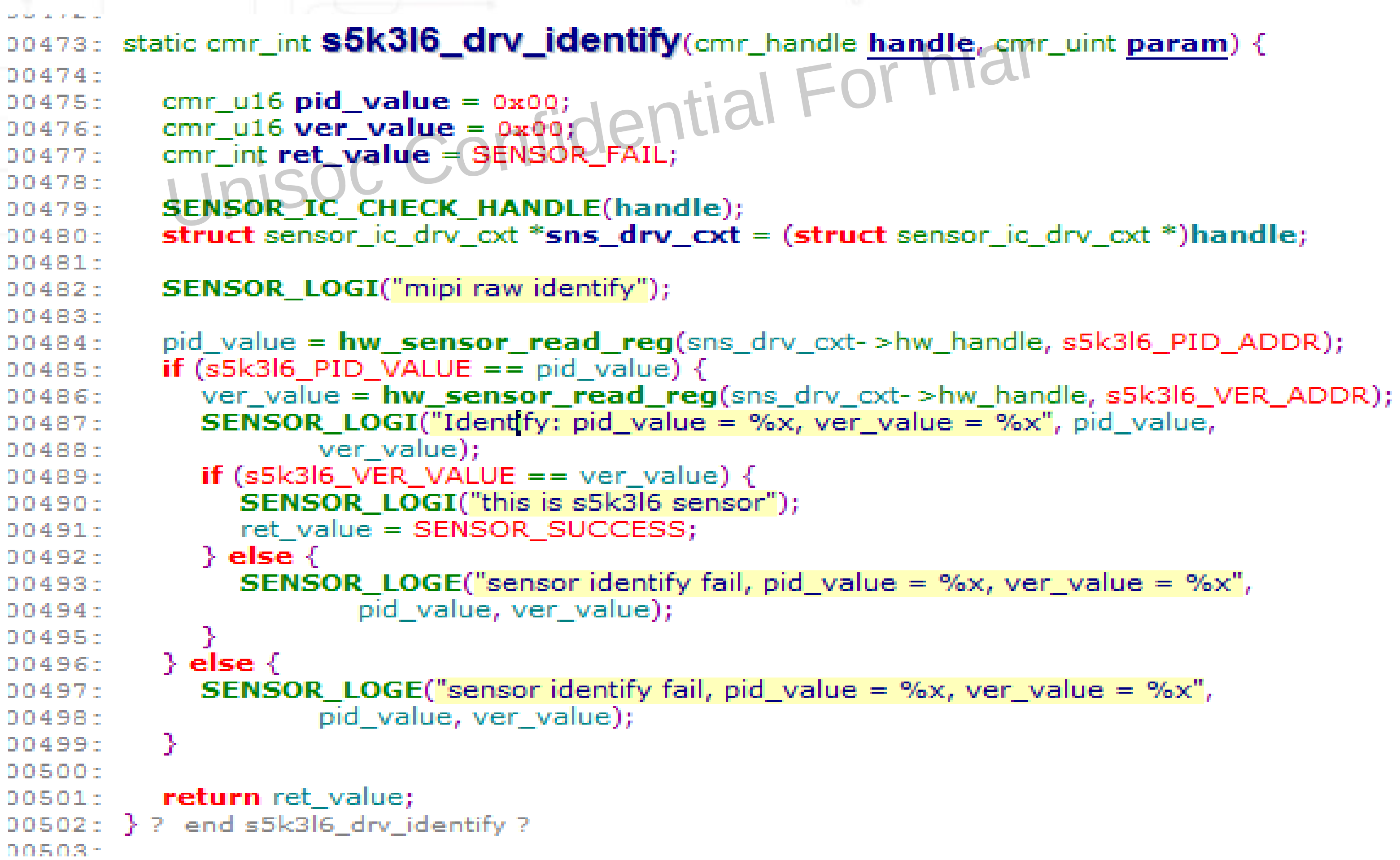
Camera驱动.c文件详解



```
static cmr_int s5k3l6_drv_power_on(cmr_handle handle, cmr_uint power_on) {
    SENSOR_IC_CHECK_HANDLE(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;
    struct module_cfg_info *module_info = sns_drv_cxt->module_info;

    SENSOR_AVDD_VAL_E dvdd_val = module_info->dvdd_val;
    SENSOR_AVDD_VAL_E avdd_val = module_info->avdd_val;
    SENSOR_AVDD_VAL_E iovdd_val = module_info->iovdd_val;
    BOOLEAN power_down = g_s5k3l6_mipi_raw_info.power_down_level;
    BOOLEAN reset_level = g_s5k3l6_mipi_raw_info.reset_pulse_level;

    if (SENSOR_TRUE == power_on) {
        hw_sensor_power_down(sns_drv_cxt->hw_handle, power_down);
        hw_sensor_set_reset_level(sns_drv_cxt->hw_handle, reset_level);
        hw_sensor_set_mclk(sns_drv_cxt->hw_handle, SENSOR_DISABLE_MCLK);
        //hw_sensor_set_avdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);
        //hw_sensor_set_dvdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);
        //hw_sensor_set_iovdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);

        usleep(1 * 1000);
        hw_sensor_set_dvdd_val(sns_drv_cxt->hw_handle, dvdd_val);
        hw_sensor_set_avdd_val(sns_drv_cxt->hw_handle, avdd_val);
        hw_sensor_set_iovdd_val(sns_drv_cxt->hw_handle, iovdd_val);
        usleep(1 * 1000);
        hw_sensor_power_down(sns_drv_cxt->hw_handle, !power_down);
        hw_sensor_set_reset_level(sns_drv_cxt->hw_handle, !reset_level);
        usleep(1 * 1000);
        hw_sensor_set_mclk(sns_drv_cxt->hw_handle, EX_MCLK);
        usleep(2 * 1000);
    } else {
        usleep(1 * 1000);
        hw_sensor_set_mclk(sns_drv_cxt->hw_handle, SENSOR_DISABLE_MCLK);
        hw_sensor_set_reset_level(sns_drv_cxt->hw_handle, reset_level);
        hw_sensor_power_down(sns_drv_cxt->hw_handle, power_down);
        usleep(1 * 1000);
        hw_sensor_set_avdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);
        hw_sensor_set_dvdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);
        hw_sensor_set_iovdd_val(sns_drv_cxt->hw_handle, SENSOR_AVDD_CLOSED);
    }

    SENSOR_LOGI("(1:on, 0:off): %lu", power_on);
    return SENSOR_SUCCESS;
```

Power on函数里面首先是对三路电源的控制和对power down及reset的初始值获取，另外两个if语句当为true的时候是上电，当为false的时候是下电，其中里面对上电下电的操作一定要严格按照sensor spec里面的时序图进行操作。

Camera驱动.c文件详解

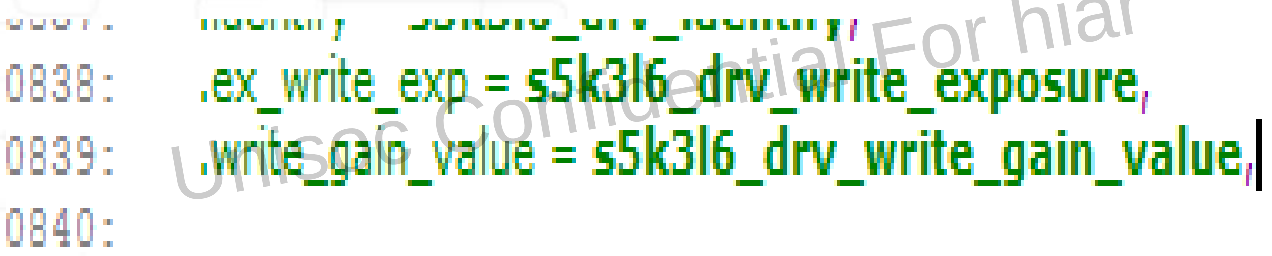


```
static cmr_int s5k3l6_drv_identify(cmr_handle handle, cmr_uint param) {

    cmr_u16 pid_value = 0x00;
    cmr_u16 ver_value = 0x00;
    cmr_int ret_value = SENSOR_FAIL;

    SENSOR_IC_CHECK_HANDLE(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    SENSOR_LOGI("mipi raw identify");

    pid_value = hw_sensor_read_reg(sns_drv_cxt->hw_handle, s5k3l6_PID_ADDR);
    if (s5k3l6_PID_VALUE == pid_value) {
        ver_value = hw_sensor_read_reg(sns_drv_cxt->hw_handle, s5k3l6_VER_ADDR);
        SENSOR_LOGI("Identify: pid_value = %x, ver_value = %x", pid_value,
                    ver_value);
        if (s5k3l6_VER_VALUE == ver_value) {
            SENSOR_LOGI("this is s5k3l6 sensor");
            ret_value = SENSOR_SUCCESS;
        } else {
            SENSOR_LOGE("sensor identify fail, pid_value = %x, ver_value = %x",
                        pid_value, ver_value);
        }
    } else {
        SENSOR_LOGE("sensor identify fail, pid_value = %x, ver_value = %x",
                    pid_value, ver_value);
    }

    return ret_value;
} ? end s5k3l6_drv_identify ?
```

Identify函数主要检查sensor的pid,vid，间接也可以检查一下sensor的上电和mclk给时钟等，这个函数在开机的时候会调用到，每一次打开都会调用到，很多时候log通过这个函数了解sensor的情况。

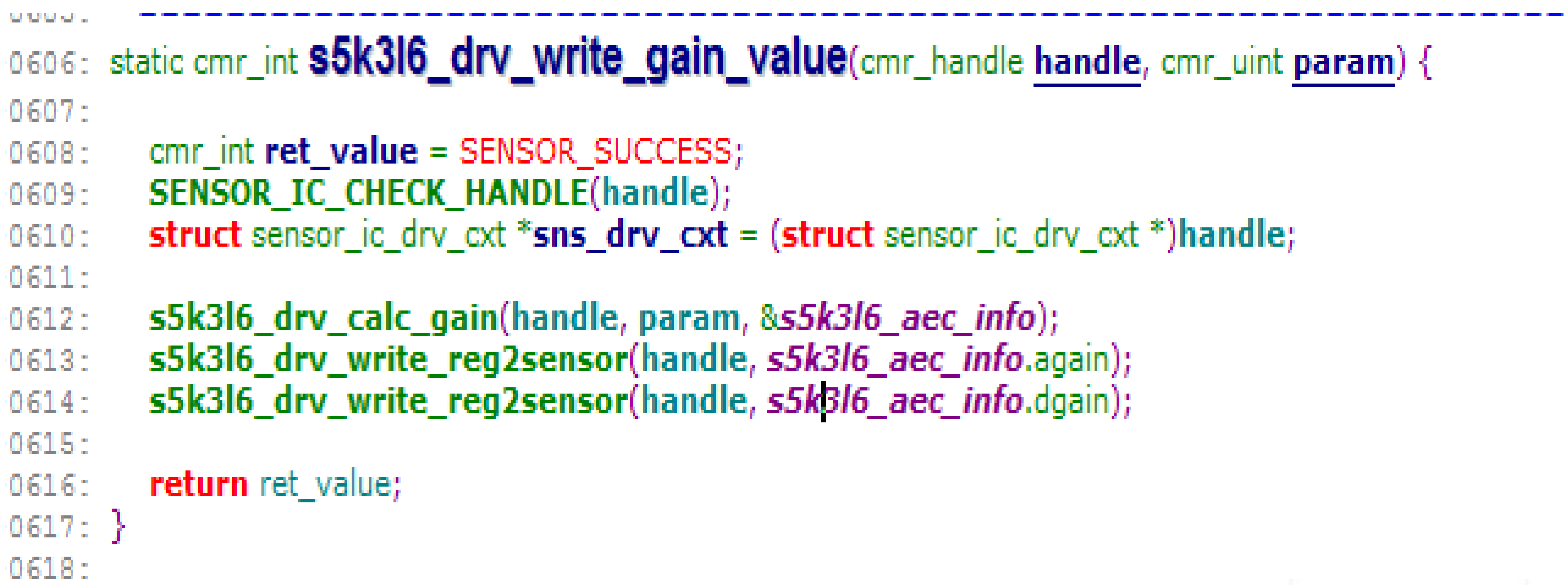
Camera驱动.c文件详解

如下这两个函数是sensor的灵魂，对sensor的控制都是通过这两个函数：



```
    .ex_write_exp = s5k3l6_drv_write_exposure,
    .write_gain_value = s5k3l6_drv_write_gain_value,
```

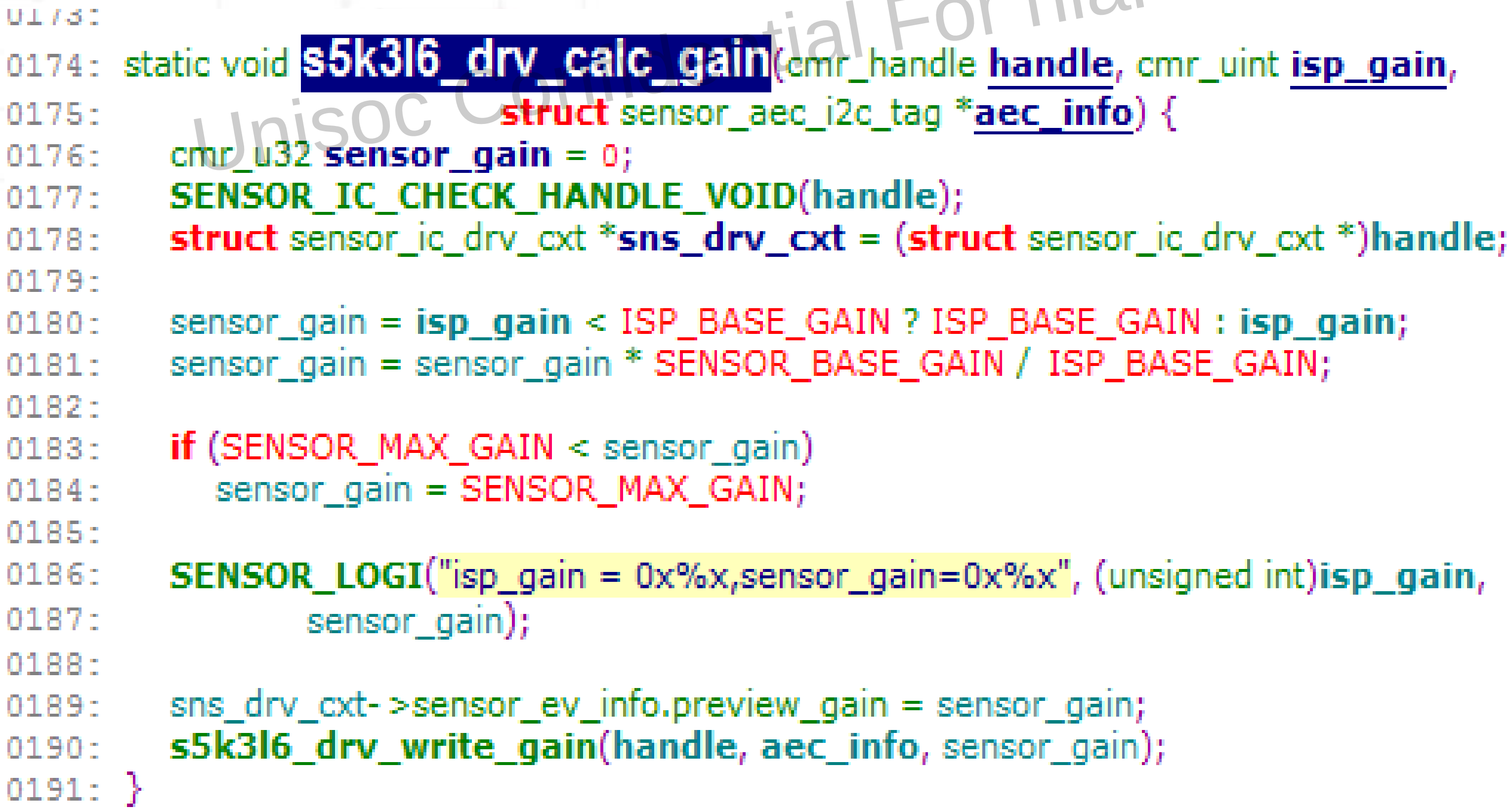
先说一下写gain相关函数：



```
static cmr_int s5k3l6_drv_write_gain_value(cmr_handle handle, cmr_uint param) {

    cmr_int ret_value = SENSOR_SUCCESS;
    SENSOR_IC_CHECK_HANDLE(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    s5k3l6_drv_calc_gain(handle, param, &s5k3l6_aec_info);
    s5k3l6_drv_write_reg2sensor(handle, s5k3l6_aec_info.again);
    s5k3l6_drv_write_reg2sensor(handle, s5k3l6_aec_info.dgain);

    return ret_value;
}
```

Camera驱动.c文件详解

这个函数里面最重要的是如下：

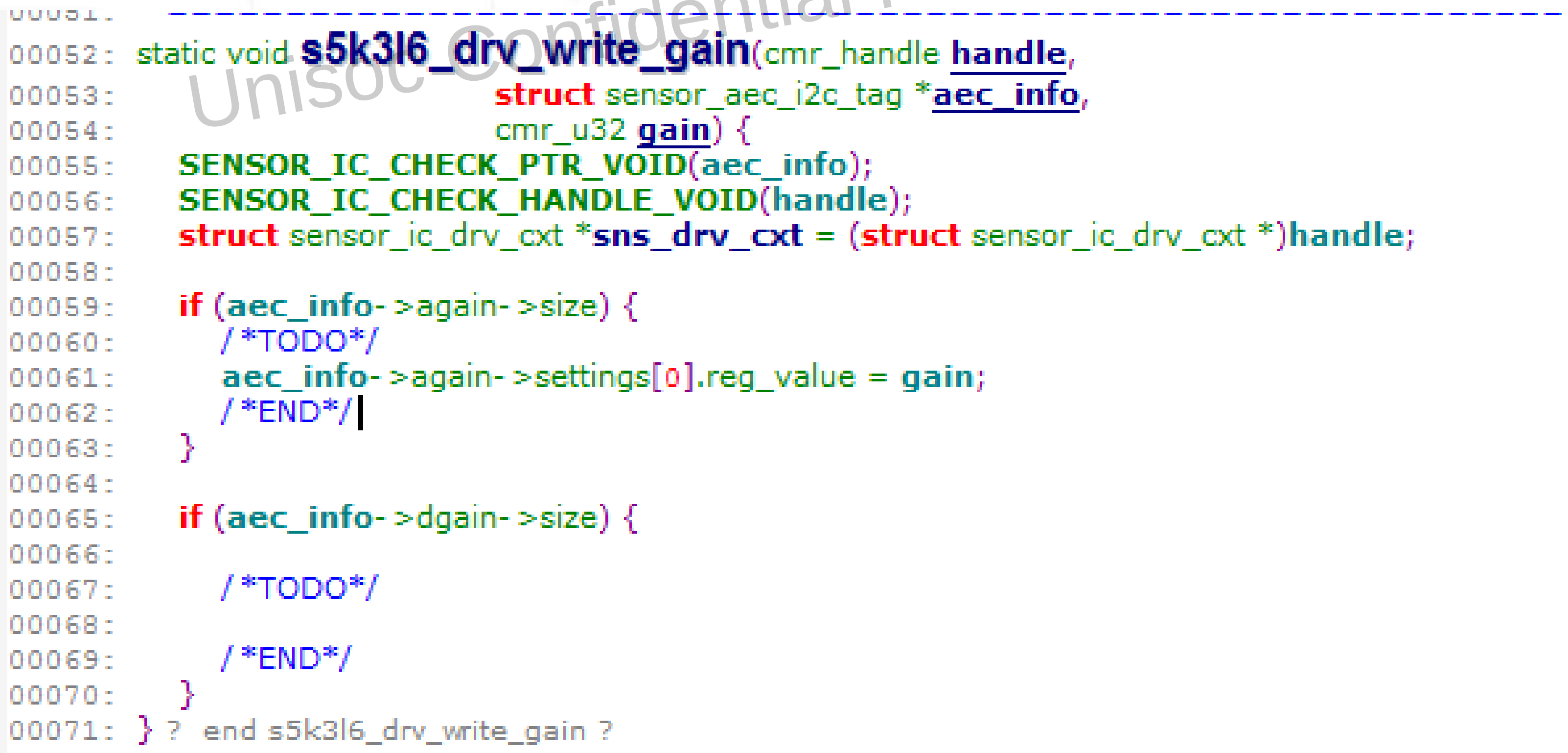


```
static void s5k3l6_drv_calc_gain(cmr_handle handle, cmr_uint isp_gain,
                                 struct sensor_aec_i2c_tag *aec_info) {
    cmr_u32 sensor_gain = 0;
    SENSOR_IC_CHECK_HANDLE_VOID(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    sensor_gain = isp_gain < ISP_BASE_GAIN ? ISP_BASE_GAIN : isp_gain;
    sensor_gain = sensor_gain * SENSOR_BASE_GAIN / ISP_BASE_GAIN;

    if (SENSOR_MAX_GAIN < sensor_gain)
        sensor_gain = SENSOR_MAX_GAIN;

    SENSOR_LOGI("isp_gain = 0x%x,sensor_gain=0x%x", (unsigned int)isp_gain,
                sensor_gain);

    sns_drv_cxt- >sensor_ev_info.preview_gain = sensor_gain;
    s5k3l6_drv_write_gain(handle, aec_info, sensor_gain);
}
```

写gain，最需要注意的就是这个函数，这里有一个按照算法给出的gain进行放大倍数的计算，.h中有对base gain等的定义，这里的需要注意的是ISP_BASE_GAIN是isp下发值下来的放大倍数的基准，然后sensor的base gain是sensor这边寄存器内部需要写进去的放大倍数的值的基准，这个跟sensor相关，需要原厂在写gain函数中将这个值定义好，

Camera驱动.c文件详解

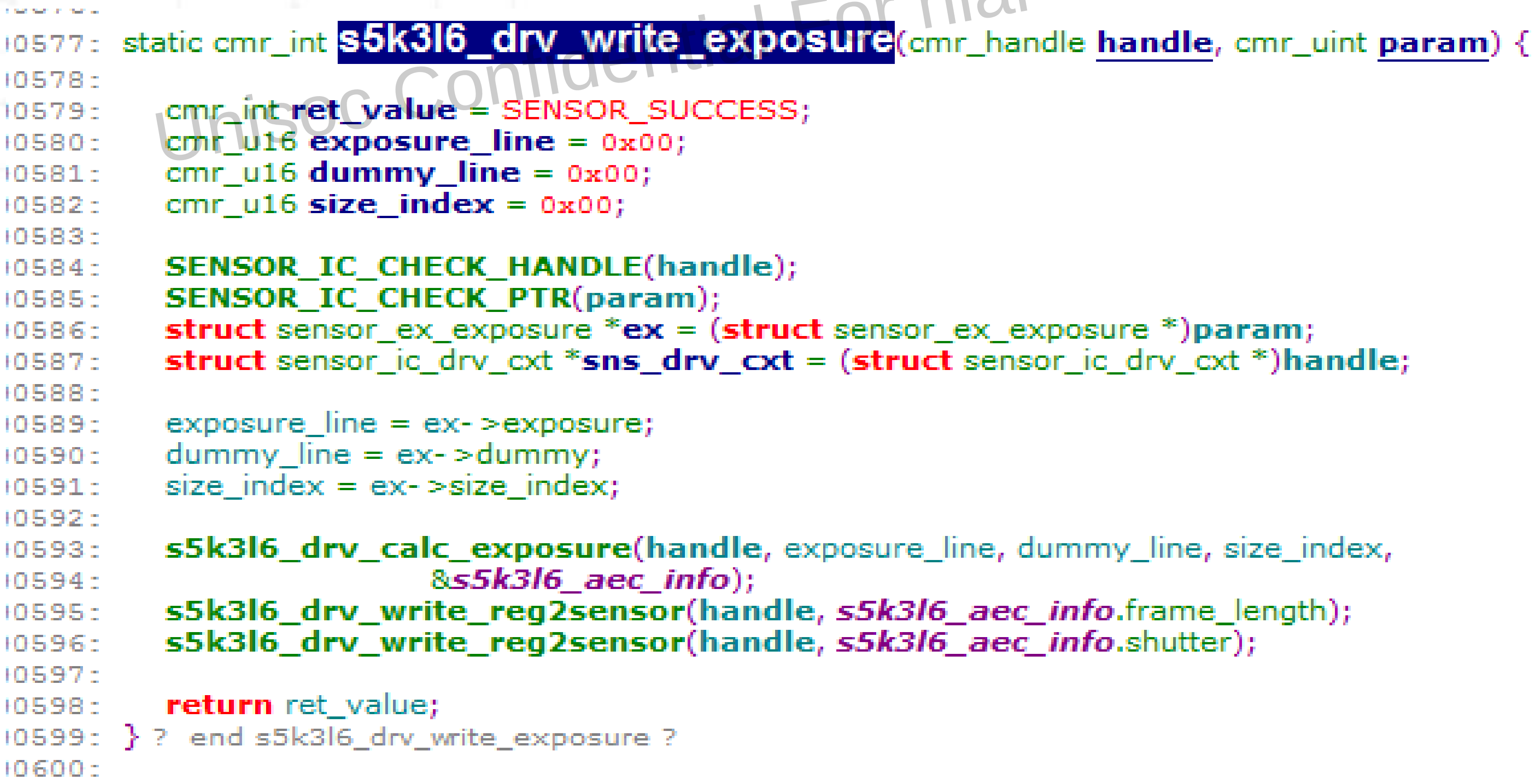
如下函数正常情况需要原厂来写，因为有些sensor的ae是分段的，写的方式不一样。
一定要重视这个函数的写的情况



```
        -----------------------------------------------------------------------------------------
static void s5k3l6_drv_write_gain(cmr_handle handle,
                                  struct sensor_aec_i2c_tag *aec_info,
                                  cmr_u32 gain) {
    SENSOR_IC_CHECK_PTR_VOID(aec_info);
    SENSOR_IC_CHECK_HANDLE_VOID(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    if (aec_info->again->size) {
        /*TODO*/
        aec_info->again->settings[0].reg_value = gain;
        /*END*/
    }

    if (aec_info->dgain->size) {

        /*TODO*/

        /*END*/
    }
} ? end s5k3l6_drv_write_gain ?
```

Camera驱动.c文件详解

下面来说一下写曝光值exposure的函数，如下是函数接口，

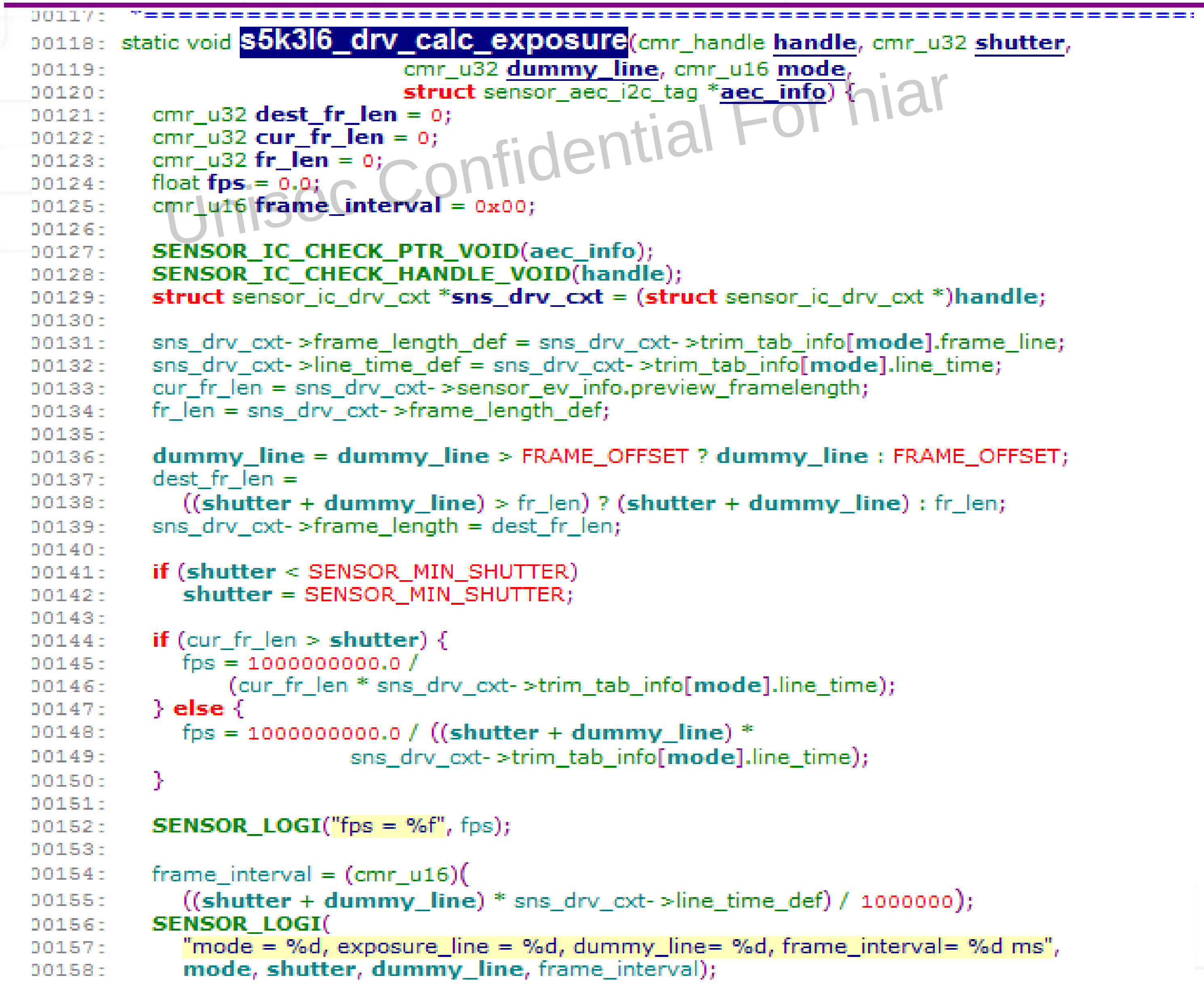


```
static cmr_int s5k3l6_drv_write_exposure(cmr_handle handle, cmr_uint param) {

    cmr_int ret_value = SENSOR_SUCCESS;
    cmr_u16 exposure_line = 0x00;
    cmr_u16 dummy_line = 0x00;
    cmr_u16 size_index = 0x00;

    SENSOR_IC_CHECK_HANDLE(handle);
    SENSOR_IC_CHECK_PTR(param);
    struct sensor_ex_exposure *ex = (struct sensor_ex_exposure *)param;
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    exposure_line = ex->exposure;
    dummy_line = ex->dummy;
    size_index = ex->size_index;

    s5k3l6_drv_calc_exposure(handle, exposure_line, dummy_line, size_index,
                             &s5k3l6_aec_info);
    s5k3l6_drv_write_reg2sensor(handle, s5k3l6_aec_info.frame_length);
    s5k3l6_drv_write_reg2sensor(handle, s5k3l6_aec_info.shutter);

    return ret_value;
} ? end s5k3l6_drv_write_exposure ?
```

从算法那边会给出三个值，分别是曝光值，dummy line，及size index为模式

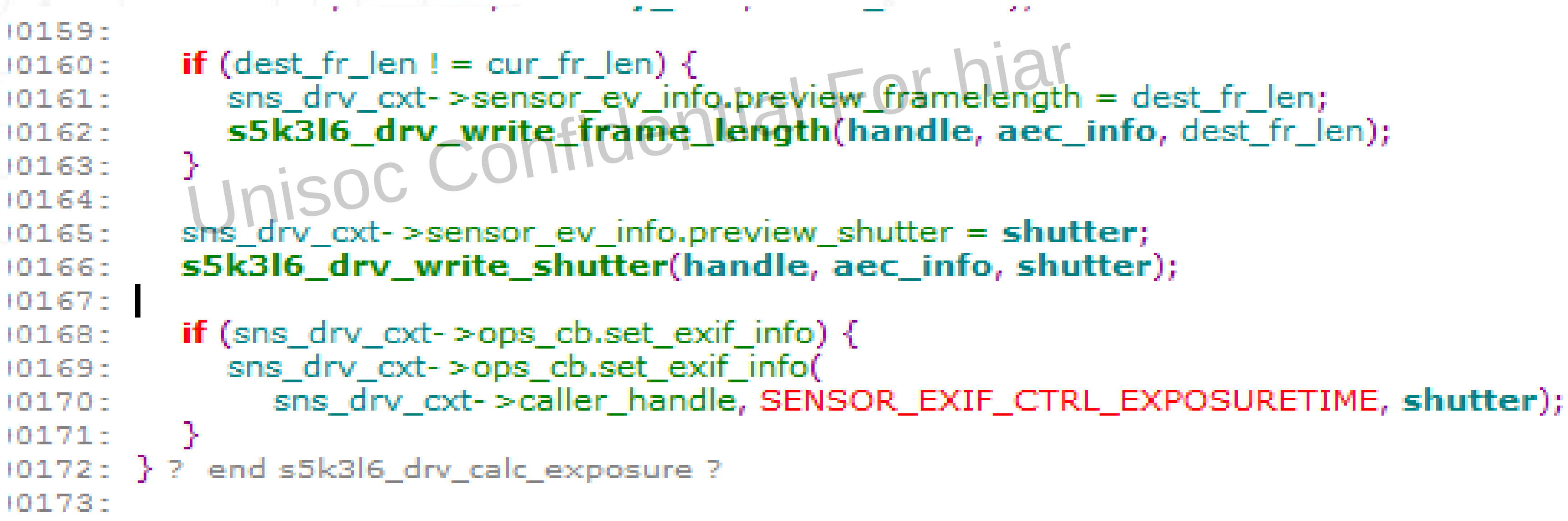
```
exposure_line = ex->exposure;
dummy_line = ex->dummy;
size_index = ex->size_index;
```

Camera驱动.c文件详解



```
 *=====================================================================*/
static void s5k3l6_drv_calc_exposure(cmr_handle handle, cmr_u32 shutter,
                                     cmr_u32 dummy_line, cmr_u16 mode,
                                     struct sensor_aec_i2c_tag *aec_info) {
    cmr_u32 dest_fr_len = 0;
    cmr_u32 cur_fr_len = 0;
    cmr_u32 fr_len = 0;
    float fps = 0.0;
    cmr_u16 frame_interval = 0x00;

    SENSOR_IC_CHECK_PTR_VOID(aec_info);
    SENSOR_IC_CHECK_HANDLE_VOID(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    sns_drv_cxt- >frame_length_def = sns_drv_cxt- >trim_tab_info[mode].frame_line;
    sns_drv_cxt- >line_time_def = sns_drv_cxt- >trim_tab_info[mode].line_time;
    cur_fr_len = sns_drv_cxt- >sensor_ev_info.preview_framelength;
    fr_len = sns_drv_cxt- >frame_length_def;

    dummy_line = dummy_line > FRAME_OFFSET ? dummy_line : FRAME_OFFSET;
    dest_fr_len =
        ((shutter + dummy_line) > fr_len) ? (shutter + dummy_line) : fr_len;
    sns_drv_cxt- >frame_length = dest_fr_len;

    if (shutter < SENSOR_MIN_SHUTTER)
        shutter = SENSOR_MIN_SHUTTER;

    if (cur_fr_len > shutter) {
        fps = 1000000000.0 /
              (cur_fr_len * sns_drv_cxt- >trim_tab_info[mode].line_time);
    } else {
        fps = 1000000000.0 / ((shutter + dummy_line) *
                              sns_drv_cxt- >trim_tab_info[mode].line_time);
    }

    SENSOR_LOGI("fps = %f", fps);

    frame_interval = (cmr_u16)(
        ((shutter + dummy_line) * sns_drv_cxt- >line_time_def) / 1000000);
    SENSOR_LOGI(
        "mode = %d, exposure_line = %d, dummy_line= %d, frame_interval= %d ms",
        mode, shutter, dummy_line, frame_interval);
```

Camera驱动.c文件详解

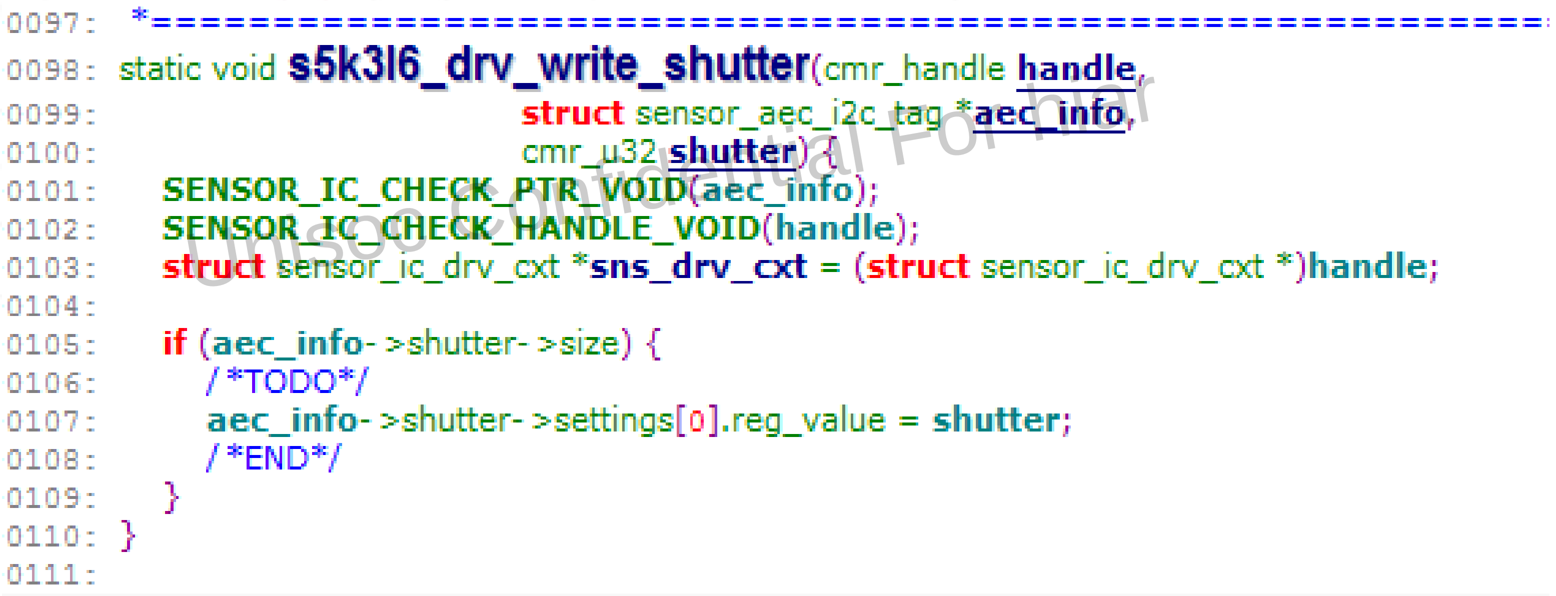


```
    if (dest_fr_len != cur_fr_len) {
        sns_drv_cxt->sensor_ev_info.preview_framelength = dest_fr_len;
        s5k3l6_drv_write_frame_length(handle, aec_info, dest_fr_len);
    }

    sns_drv_cxt->sensor_ev_info.preview_shutter = shutter;
    s5k3l6_drv_write_shutter(handle, aec_info, shutter);

    if (sns_drv_cxt->ops_cb.set_exif_info) {
        sns_drv_cxt->ops_cb.set_exif_info(
            sns_drv_cxt->caller_handle, SENSOR_EXIF_CTRL_EXPOSURETIME, shutter);
    }
} ? end s5k3l6_drv_calc_exposure ?
```

这个函数是计算曝光行写到sensor中的主要函数，需要这个函数来进行计算包括帧长和dummy line及shutter值。这里需要注意的是理论上帧长是决定帧率的，但很多sensor的帧率并不由帧长决定，可能为vb决定，需要的是dummy line，这个配置需要sensor原厂对vb的控制写在s5k3l6_drv_write_frame_length中来实现对帧率的控制。

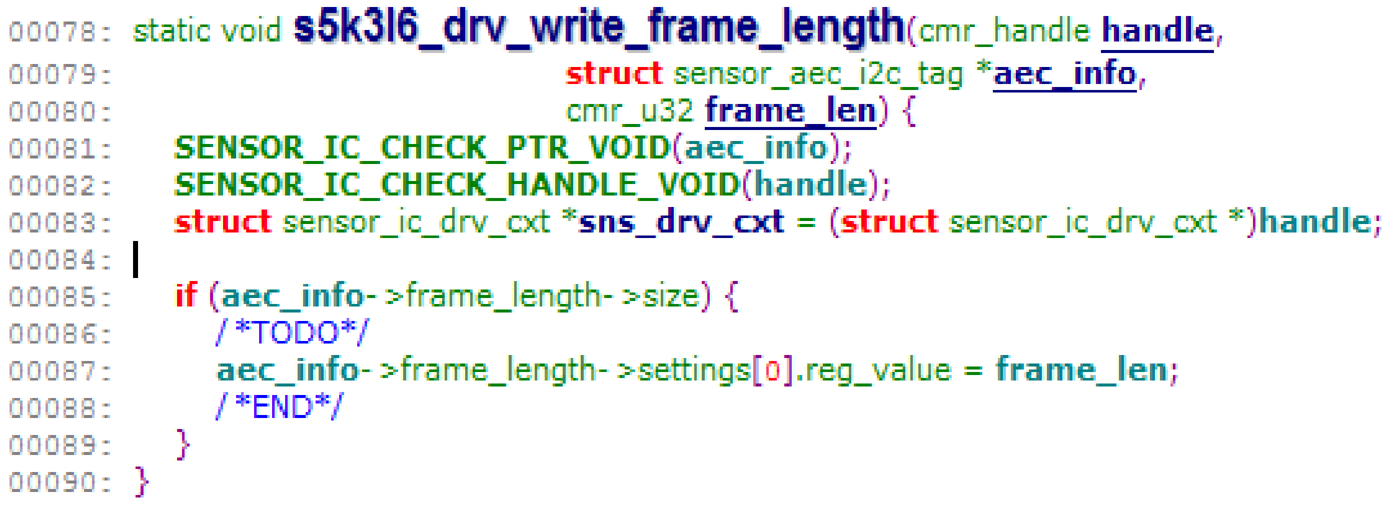
最后就是将shutter值真正写进寄存器：

Camera驱动.c文件详解



```
 *=====================================================================
static void s5k3l6_drv_write_shutter(cmr_handle handle,
                                     struct sensor_aec_i2c_tag *aec_info,
                                     cmr_u32 shutter) {
    SENSOR_IC_CHECK_PTR_VOID(aec_info);
    SENSOR_IC_CHECK_HANDLE_VOID(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    if (aec_info- >shutter- >size) {
        / *TODO*/
        aec_info- >shutter- >settings[0].reg_value = shutter;
        / *END*/
    }
}

```

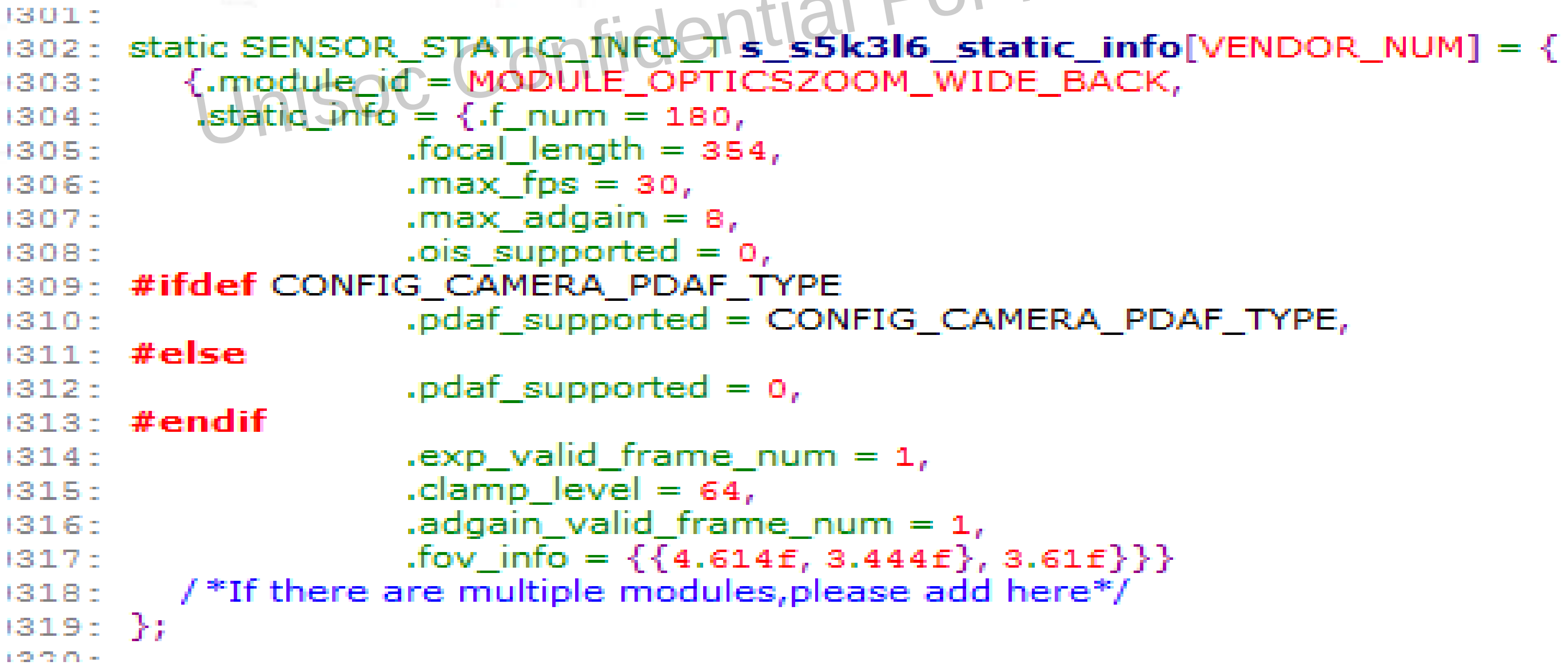
如下函数是对帧率的控制，帧长决定帧率，但部分sensor没有帧长控制，那需要在这个函数中控制VB



```
static void s5k3l6_drv_write_frame_length(cmr_handle handle,
                                          struct sensor_aec_i2c_tag *aec_info,
                                          cmr_u32 frame_len) {
    SENSOR_IC_CHECK_PTR_VOID(aec_info);
    SENSOR_IC_CHECK_HANDLE_VOID(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    if (aec_info- >frame_length- >size) {
        / *TODO*/
        aec_info- >frame_length- >settings[0].reg_value = frame_len;
        / *END*/
    }
}
```

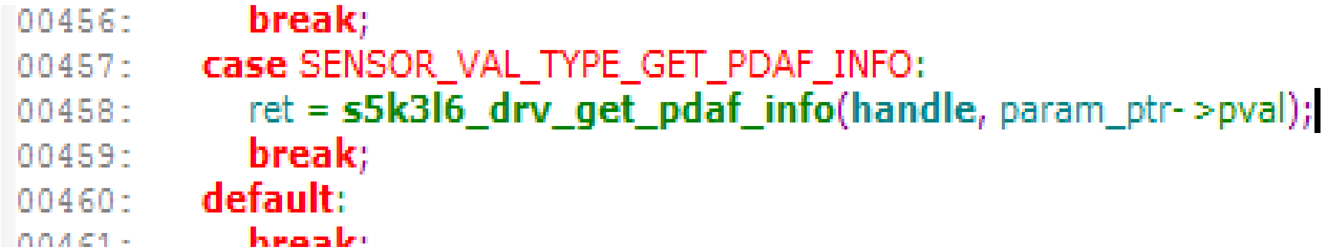
Camera驱动.c文件详解

如果sensor是带pdaf的话，还可以添加pdaf配置，平台支持PDAF，包括Type1/2/3及Dual PD等,如下图所示，这个pdaf的配置表示是否支持pdaf，配置的type种类：



```
static SENSOR_STATIC_INFO_T s_s5k3l6_static_info[VENDOR_NUM] = {
    {.module_id = MODULE_OPTICSZOOM_WIDE_BACK,
     .static_info = {.f_num = 180,
                     .focal_length = 354,
                     .max_fps = 30,
                     .max_adgain = 8,
                     .ois_supported = 0,
#ifdef CONFIG_CAMERA_PDAF_TYPE
                     .pdaf_supported = CONFIG_CAMERA_PDAF_TYPE,
#else
                     .pdaf_supported = 0,
#endif
                     .exp_valid_frame_num = 1,
                     .clamp_level = 64,
                     .adgain_valid_frame_num = 1,
                     .fov_info = {{4.614f, 3.444f}, 3.61f}}}
    /*If there are multiple modules,please add here*/
};
```

如下是oem层从驱动获得pdaf信息的接口函数：

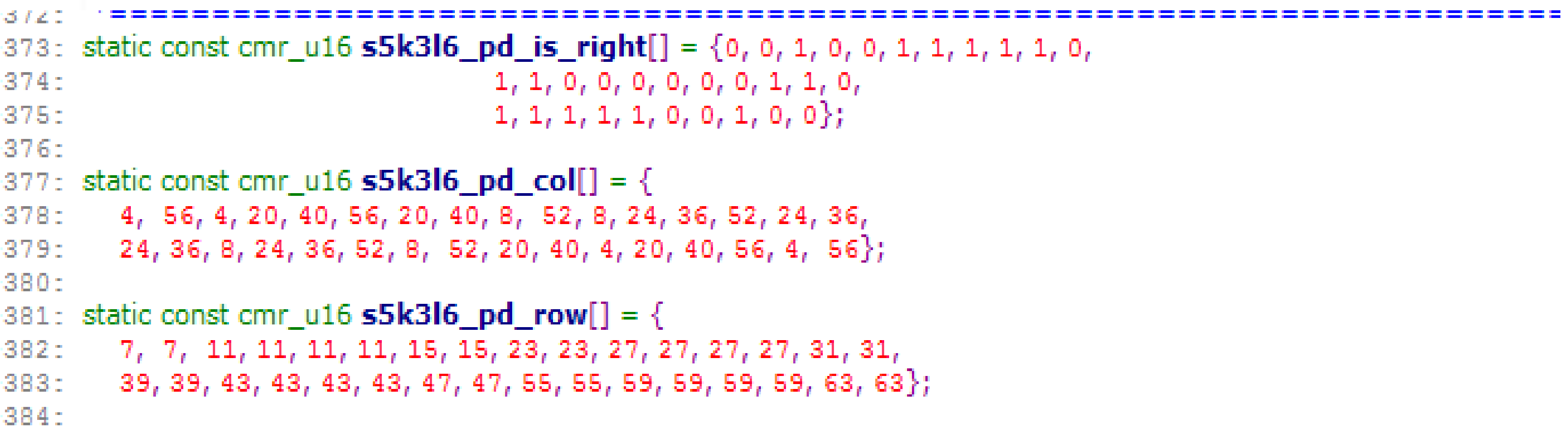


```
        break;
    case SENSOR_VAL_TYPE_GET_PDAF_INFO:
        ret = s5k3l6_drv_get_pdaf_info(handle, param_ptr- >pval);
        break;
    default:
        break;
```

Camera驱动.c文件详解

先了解一下pdaf的静态信息：

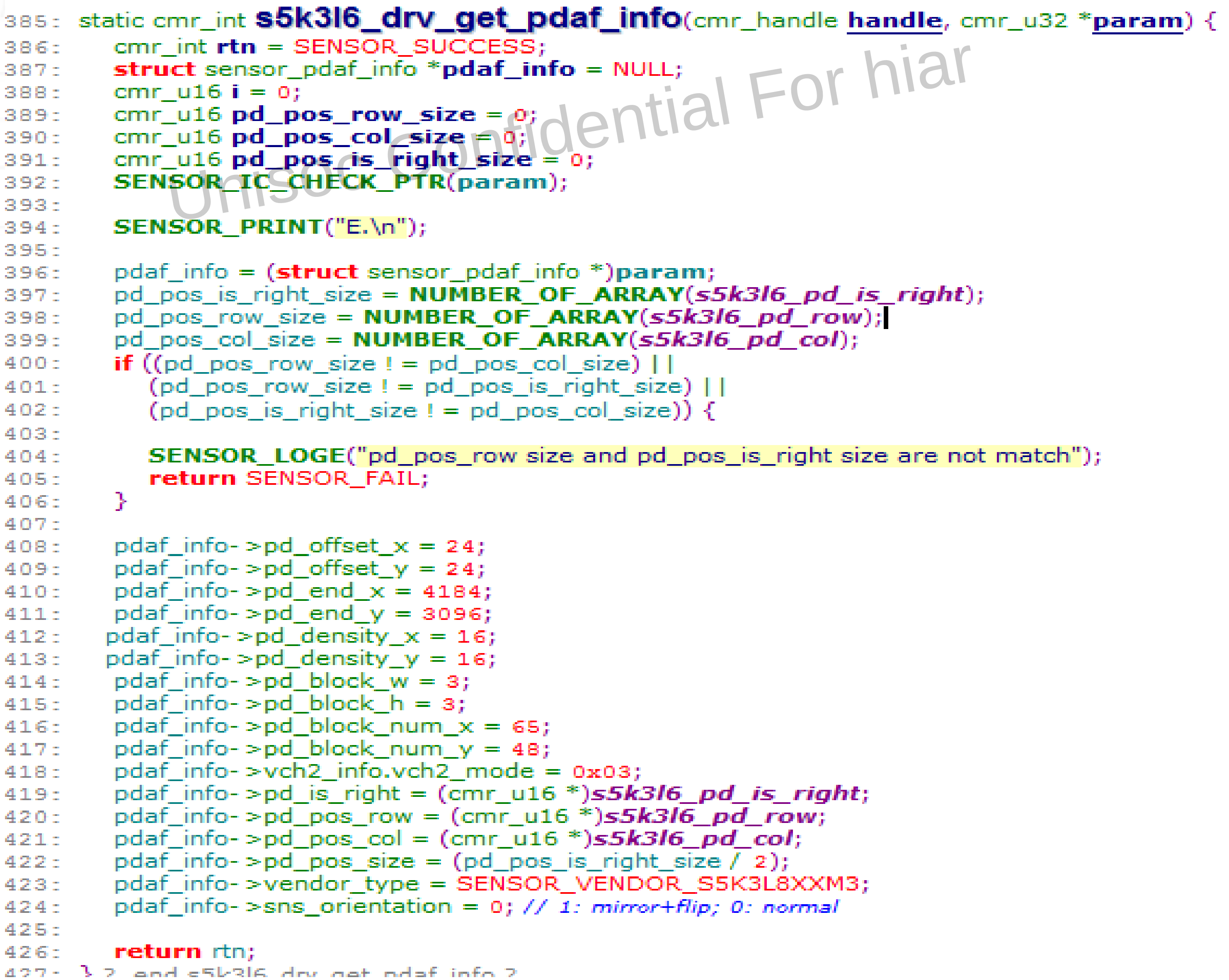
如下是pdaf的静态信息，s5k3l6_pd_is_right这个表示一个block里面有多少pd点，从这个里面看总共有32个pd点，另外，这个结构体名字看，叫做is_right，那0的话表示遮住了左边，1的话，遮住的是右边。下面的s5k3l6_pd_col和s5k3l6_pd_row是对应的pd点，在block中的位置，比如一个block是64X64的grid。



```
=====================================================================
static const cmr_u16 s5k3l6_pd_is_right[] = {0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
                                             1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                                             1, 1, 1, 1, 1, 0, 0, 1, 0, 0};

static const cmr_u16 s5k3l6_pd_col[] = {
    4,  56, 4, 20, 40, 56, 20, 40, 8,  52, 8, 24, 36, 52, 24, 36,
    24, 36, 8, 24, 36, 52, 8,  52, 20, 40, 4, 20, 40, 56, 4,  56};

static const cmr_u16 s5k3l6_pd_row[] = {
    7,  7,  11, 11, 11, 11, 15, 15, 23, 23, 27, 27, 27, 27, 31, 31,
    39, 39, 43, 43, 43, 43, 47, 47, 55, 55, 59, 59, 59, 59, 63, 63};

```

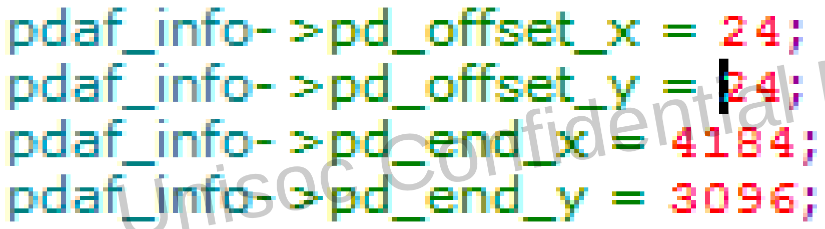
如下函数是获得pdaf接口函数的：

Camera驱动.c文件详解



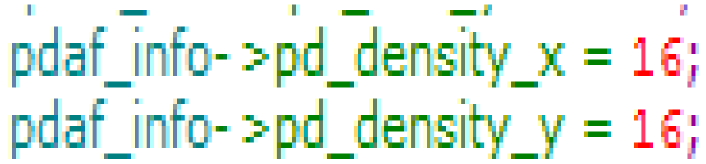
```
static cmr_int s5k3l6_drv_get_pdaf_info(cmr_handle handle, cmr_u32 *param) {
    cmr_int rtn = SENSOR_SUCCESS;
    struct sensor_pdaf_info *pdaf_info = NULL;
    cmr_u16 i = 0;
    cmr_u16 pd_pos_row_size = 0;
    cmr_u16 pd_pos_col_size = 0;
    cmr_u16 pd_pos_is_right_size = 0;
    SENSOR_IC_CHECK_PTR(param);

    SENSOR_PRINT("E.\n");

    pdaf_info = (struct sensor_pdaf_info *)param;
    pd_pos_is_right_size = NUMBER_OF_ARRAY(s5k3l6_pd_is_right);
    pd_pos_row_size = NUMBER_OF_ARRAY(s5k3l6_pd_row);
    pd_pos_col_size = NUMBER_OF_ARRAY(s5k3l6_pd_col);
    if ((pd_pos_row_size != pd_pos_col_size) ||
        (pd_pos_row_size != pd_pos_is_right_size) ||
        (pd_pos_is_right_size != pd_pos_col_size)) {

        SENSOR_LOGE("pd_pos_row size and pd_pos_is_right size are not match");
        return SENSOR_FAIL;
    }

    pdaf_info->pd_offset_x = 24;
    pdaf_info->pd_offset_y = 24;
    pdaf_info->pd_end_x = 4184;
    pdaf_info->pd_end_y = 3096;
   pdaf_info->pd_density_x = 16;
   pdaf_info->pd_density_y = 16;
    pdaf_info->pd_block_w = 3;
    pdaf_info->pd_block_h = 3;
    pdaf_info->pd_block_num_x = 65;
    pdaf_info->pd_block_num_y = 48;
    pdaf_info->vch2_info.vch2_mode = 0x03;
    pdaf_info->pd_is_right = (cmr_u16 *)s5k3l6_pd_is_right;
    pdaf_info->pd_pos_row = (cmr_u16 *)s5k3l6_pd_row;
    pdaf_info->pd_pos_col = (cmr_u16 *)s5k3l6_pd_col;
    pdaf_info->pd_pos_size = (pd_pos_is_right_size / 2);
    pdaf_info->vendor_type = SENSOR_VENDOR_S5K3L8XXM3;
    pdaf_info->sns_orientation = 0; // 1: mirror+flip; 0: normal

    return rtn;
} ? end s5k3l6_drv_get_pdaf_info ?
```

Camera驱动.c文件详解



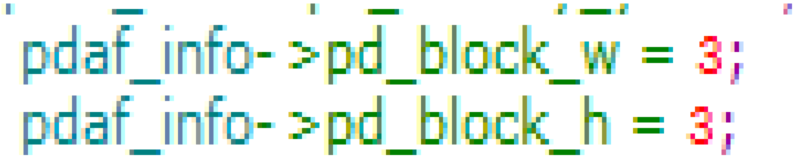
```
pdaf_info->pd_offset_x = 24;
pdaf_info->pd_offset_y = 24;
pdaf_info->pd_end_x = 4184;
pdaf_info->pd_end_y = 3096;
```

这四个值是sensor有效出pdaf图片的起始点和结束点
起始点是(24,24),结束点是(4184,3096),



```
pdaf_info->pd_density_x = 16;
pdaf_info->pd_density_y = 16;
```

如上两个表示pd点密度信息，目前大多数都没有用到。



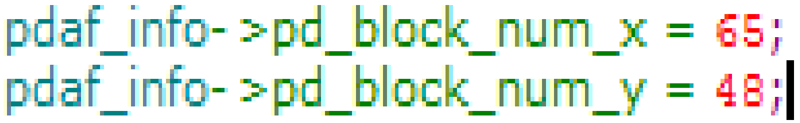
```
pdaf_info->pd_block_w = 3;
pdaf_info->pd_block_h = 3;
```

Camera驱动.c文件详解

对应值如下：

pd_block_w =0　像素宽度为8
pd_block_w =1　像素宽度为16
pd_block_w =2　像素宽度为32
pd_block_w =3　像素宽度为64
最多只支持64

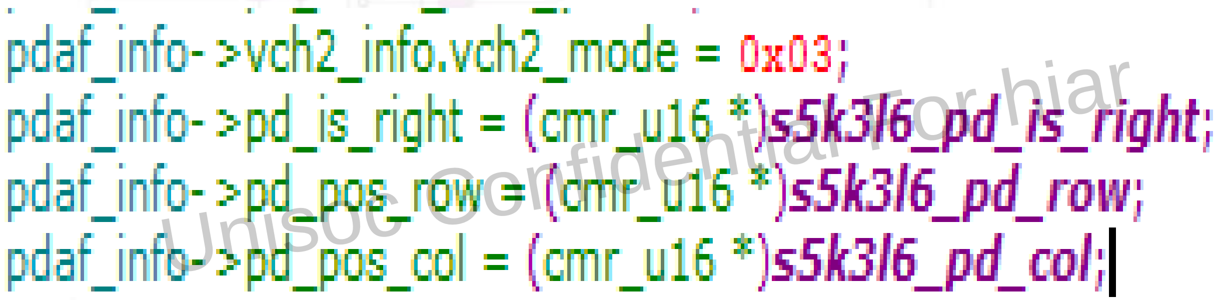
pd_block_h =0　像素高度为8
pd_block_h =1　像素高度为16
pd_block_h =2　像素高度为32
pd_block_h =3　像素高度为64
最多只支持64



```
pdaf_info->pd_block_num_x = 65;
pdaf_info->pd_block_num_y = 48;
```

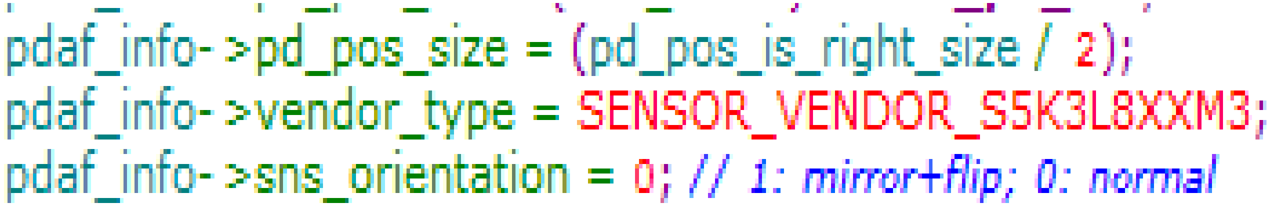
这个表示总共有多少个block，如65x64=4160,48x64=3072.这个值跟上面的是对应的，4184-24=4160，3096-24=3072.

Camera驱动.c文件详解



```
pdaf_info->vch2_info.vch2_mode = 0x03;
pdaf_info->pd_is_right = (cmr_u16 *)s5k3l6_pd_is_right;
pdaf_info->pd_pos_row = (cmr_u16 *)s5k3l6_pd_row;
pdaf_info->pd_pos_col = (cmr_u16 *)s5k3l6_pd_col;
```

这里的0x03是给到平台的，固定值。其它三个是sensor PD点静态信息。

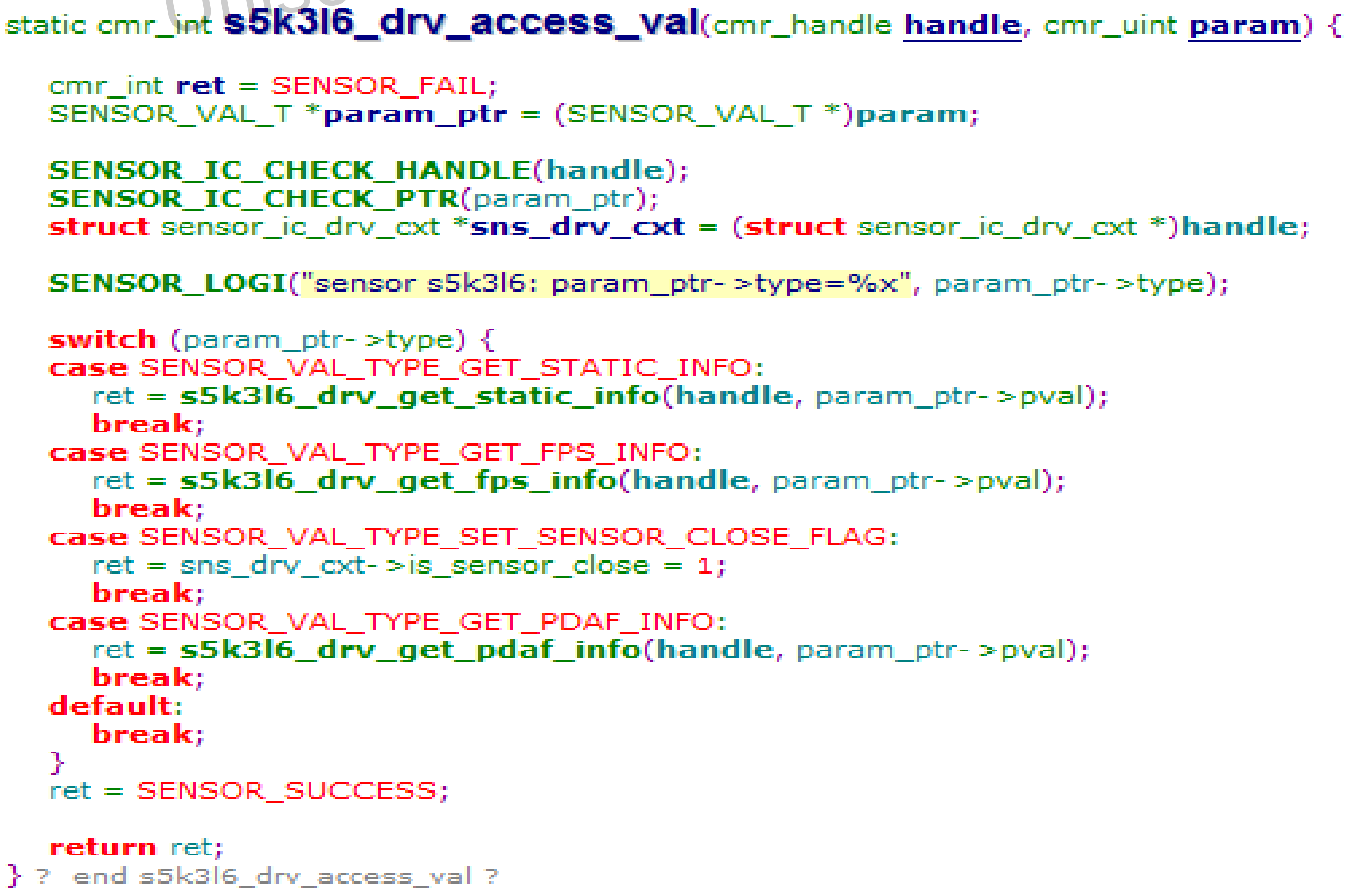


```
pdaf_info->pd_pos_size = (pd_pos_is_right_size / 2);
pdaf_info->vendor_type = SENSOR_VENDOR_S5K3L8XXM3;
pdaf_info->sns_orientation = 0; // 1: mirror+flip; 0: normal
```

上面的表示PD点的对数，SENSOR_VENDOR_S5K3L8XXM3表示平台对每一个不同sensor原厂及不同大小的sensor pdaf种类的定义，这个需要适配厂商和尺寸大小，sns_orientation表示是normal还是mirror+flip。

Camera驱动.c文件详解

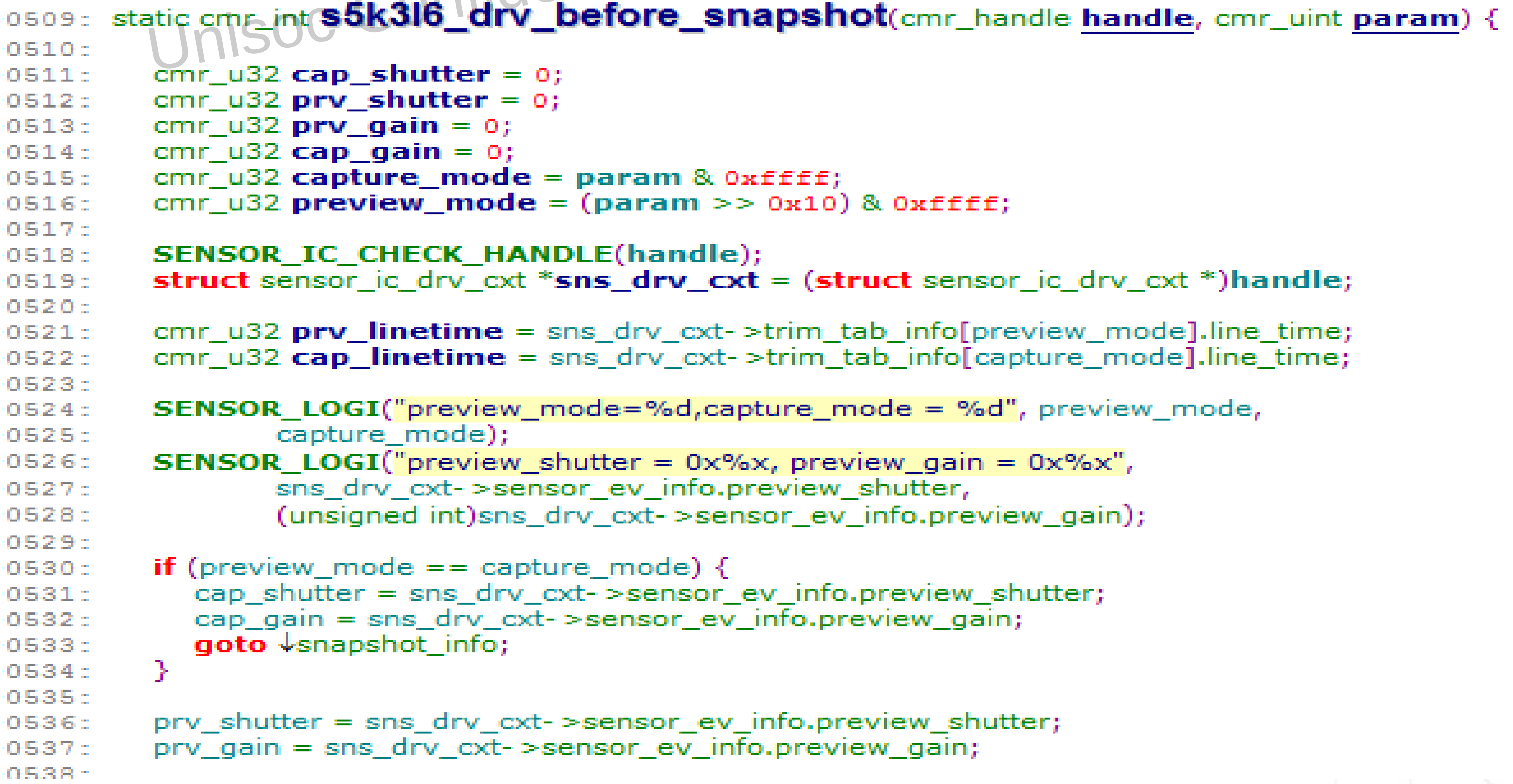
另外，讲解一下函数s5k3l6_drv_access_val，每一个驱动中都有这个access，这个函数主要是oem层对驱动中一些特性相关函数的调用，包括一些需要私自添加的接口，可以在这个函数里面实现，由oem调用到上层去，如sensor的otp驱动，可以在如下函数中添加case语句SENSOR_VAL_TYPE_READ_OTP，来实现sensor自带otp的校准，其他的接口添加也可以参考。



```
static cmr_int s5k3l6_drv_access_val(cmr_handle handle, cmr_uint param) {

    cmr_int ret = SENSOR_FAIL;
    SENSOR_VAL_T *param_ptr = (SENSOR_VAL_T *)param;

    SENSOR_IC_CHECK_HANDLE(handle);
    SENSOR_IC_CHECK_PTR(param_ptr);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    SENSOR_LOGI("sensor s5k3l6: param_ptr->type=%x", param_ptr->type);

    switch (param_ptr->type) {
    case SENSOR_VAL_TYPE_GET_STATIC_INFO:
        ret = s5k3l6_drv_get_static_info(handle, param_ptr->pval);
        break;
    case SENSOR_VAL_TYPE_GET_FPS_INFO:
        ret = s5k3l6_drv_get_fps_info(handle, param_ptr->pval);
        break;
    case SENSOR_VAL_TYPE_SET_SENSOR_CLOSE_FLAG:
        ret = sns_drv_cxt->is_sensor_close = 1;
        break;
    case SENSOR_VAL_TYPE_GET_PDAF_INFO:
        ret = s5k3l6_drv_get_pdaf_info(handle, param_ptr->pval);
        break;
    default:
        break;
    }
    ret = SENSOR_SUCCESS;

    return ret;
} ? end s5k3l6_drv_access_val ?
```

Camera驱动.c文件详解

最后，讲解一下函数s5k3l6_drv_before_snapshot，这个函数是在no-zsl拍照时使用，用于尺寸模式的切换，当预览是在小尺寸的时候，拍照需要大尺寸，那就会调用到这个函数，实现模式的切换，包括对shutter值，framelength，gain等的修改及模式的setting的切换。



```
static cmr_int s5k3l6_drv_before_snapshot(cmr_handle handle, cmr_uint param) {

    cmr_u32 cap_shutter = 0;
    cmr_u32 prv_shutter = 0;
    cmr_u32 prv_gain = 0;
    cmr_u32 cap_gain = 0;
    cmr_u32 capture_mode = param & 0xffff;
    cmr_u32 preview_mode = (param >> 0x10) & 0xffff;

    SENSOR_IC_CHECK_HANDLE(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    cmr_u32 prv_linetime = sns_drv_cxt->trim_tab_info[preview_mode].line_time;
    cmr_u32 cap_linetime = sns_drv_cxt->trim_tab_info[capture_mode].line_time;

    SENSOR_LOGI("preview_mode=%d,capture_mode = %d", preview_mode,
            capture_mode);
    SENSOR_LOGI("preview_shutter = 0x%x, preview_gain = 0x%x",
            sns_drv_cxt->sensor_ev_info.preview_shutter,
            (unsigned int)sns_drv_cxt->sensor_ev_info.preview_gain);

    if (preview_mode == capture_mode) {
        cap_shutter = sns_drv_cxt->sensor_ev_info.preview_shutter;
        cap_gain = sns_drv_cxt->sensor_ev_info.preview_gain;
        goto ↓snapshot_info;
    }

    prv_shutter = sns_drv_cxt->sensor_ev_info.preview_shutter;
    prv_gain = sns_drv_cxt->sensor_ev_info.preview_gain;

```

Camera驱动总结

总结：

到目前为止，camera驱动的讲解基本完成，在开发或者bring up camera驱动的时候，需要细致的驱动进行检查，并完成每一个配置和函数。所有这些都是需要sensor原厂和驱动工程师一起协同完成。任何一个配置不正确都可能导致很多问题，比如bps不正确可能导致接收数据错误，linetime不对可能导致水波纹等。对于上电时序，最好通过严格的示波器检查，VB也最好在一帧的占比为不超过3%，mipi的时序也是需要通过波形测试确认等。

THANKS

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。

WWW.UNISOC.COM
紫光展锐科技