SIM8500 Android10 Camera 移植文档

智能模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	SIM8500_android10_camera移植文档
版本：	1.00
日期：	2022.03.09
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

关于文档

版本历史

版本	日期	作者	备注
1.00	2022.03.09	李乐宁	第一版

适用范围

本文档适用于 SIMCom SIM850X 系列的 Android10 版本。

目录

1 介绍

1.1 本文目的

本文档提供 SIM8500(展锐 SL8541E) Android10 平台的 camera 驱动的移植方法。
展锐 SL8541E 平台 camera 架构包含下列组件：

- **Sensor**
- **PHY**
- **DCAM**
- **Flash**
- **Actuator**
- **Eeprom**
- **Chromatix**

1.2 参考文档

[1] SL8541EDeviceSpecificationV1.6

1.3 术语和缩写

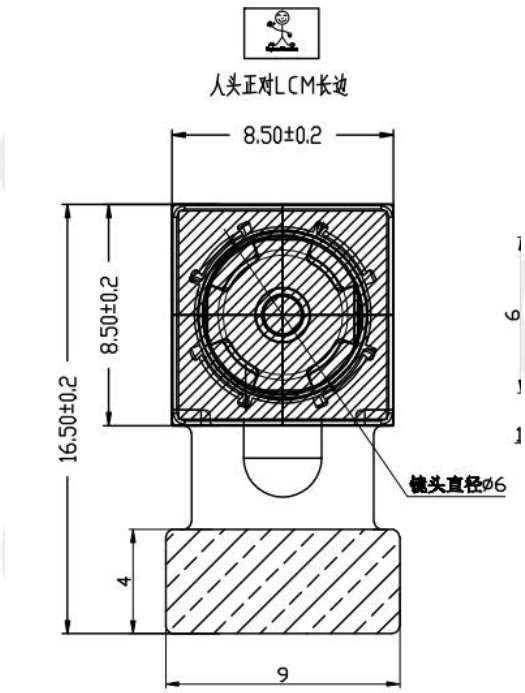
DCAM	Digital camera

2 模组的安装方向

为了确保 camera preview 的图像在 LCD 上显示是正确的也就是我们经常说的所见即所得，我们必须要保证模组正确的安装在我们的设备上。

2.1 Camera 和 lcd 的扫描方向

- **Camera 的扫描方向；**
  - **Camera 模组的扫描方式**



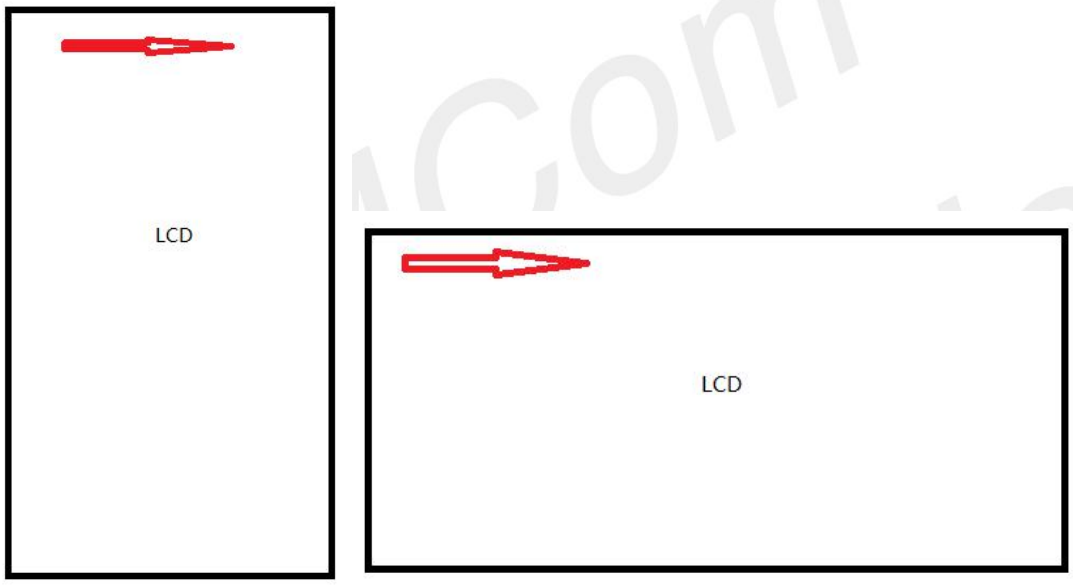
上面的图片在我们的模组 spec 里都可以找到，图片里的小人是我们关注的重点，我们常用 camera 的分辨率一般都是 4:3 的，后面介绍 4 我们说成长边，3 说成短边。上面图片里小人人头正对 camera 的长边，camera 的扫描方向是沿长边扫描的。

- **Camera 模组在不同方向时拍摄同一张图片的效果（红色箭头表示 camera 的扫描方向）**

原始图片	模组摆放	输出照片
		

我们看一下上面两个模组的按照不同的方向摆设拍摄同一张照片的输出效果。从输出看，第一组扫描的原点在 F 的左上角，第二组扫描原点在 F 的左下角。同样还有其他一些摆放方式我们就不介绍，只要确认了 camera 模组的扫描方向，那么就可以确认出不同摆设方向拍摄同一个物体的输出效果。

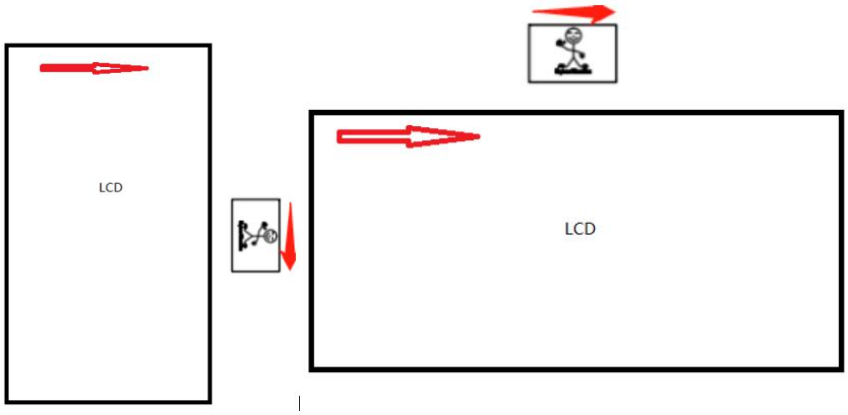
- **LCD 的刷屏方式；**
  - **竖屏和横屏**



从上面的图片可以看出 LCD 的刷屏方向一般都是从左上角的原点向由，这一点和 camera 不一样，camera 是固定的沿长边扫描。

2.2 Camera 模组的摆放

根据 camera 的扫描和 LCD 的刷屏方向介绍，如果我们要做到拍摄的图片是所见即所得并且尽量在 camera preview 的时候显示区域在 LCD 上呈现的区域做到最大，那么正常的摆放方式应该是 camera 的长边平行于 LCD 的长边。即小人脚踩 LCD 的长边，对于竖屏 LCD 同样在代码中会设置渲染到 LCD 上的图片是 camera 输出原始图片顺时针选择 90 度，同样如果小人头顶 LCD 的长边也可以，不过这时如果不修改默认设置旋转 90 度，那么可以修改 camera 模组的 Flip 和 Mirror 寄存器。此方式适合前后摄像图，不过按照默认配置一般后摄像顺时针旋转 90 度，前摄像头顺时针选择 270 度。上面是所提到的角度是针对竖屏手机的摆放，根据长边平行的原则，横屏手机如下图。



➢ **代码中方向怎么修改；**

代码位置：

device\sprd\sharkle\sl8541e_1h10_32b\camera\sensor_config.xml

```
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

Orientation 表示你需要旋转的度数，可以设置 0,90,180,270，请根据实际情况设置。

➢ **修改 camera 模组的 flip 和 mirror 寄存器；**

可以查看 sensor 位于 vendor 下的驱动文件

vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore\gc8024\sensor_gc8024_mipi_raw.h

```
#ifdef IMAGE_NORMAL_MIRROR
#define MIRROR 0xd4
#define PRE_STARTY 0x03
#define PRE_STARTX 0x07
#define CAP_STARTY 0x03
#define CAP_STARTX 0x09
#endif
```

寄存器描述可以参考 spec

Readout direction can be set by the registers.

Function	Register Address	Register Value	First Pixel
Normal	P0:0x17[1:0]	00	R
Horizontal mirror	P0:0x17[1:0]	01	Gr
Vertical Flip	P0:0x17[1:0]	10	Gb
Horizontal Mirror and Vertical Flip	P0:0x17[1:0]	11	B

上面两部分文档中的描述和代码中的位置不是绝对的，要根据你所配置的 sensor 的驱动和 spec 来查找。这两部分的修改最好咨询 sensor 的 fae。

3 模组厂需要提供的资料

3.1 下列是模组厂需要提供的资料

- ✧ **Sensor datasheet，AF datasheet(如果存在 AF)**
- ✧ **Sensor 的驱动**
- ✧ **Sensor Chromatix Code**
- ✧ **AF Actuator Code(如果 AF 的 IC 代码列表里已经存在可以不用提供)**
- ✧ **Eeprom Code(如果存在 Eeprom)**

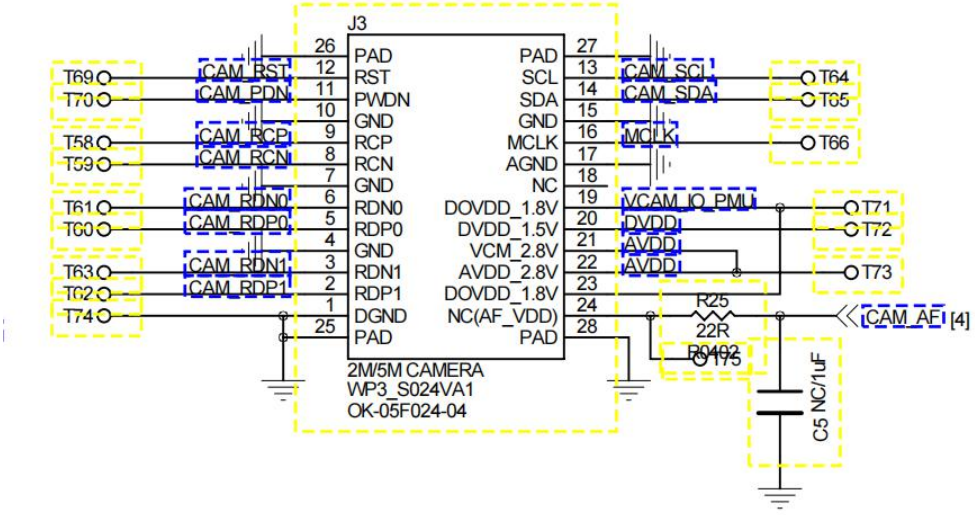
3.2 代码目录

✧ 下面已 **GC8024** 介绍厂家提供的 **code** 需要放置的位置

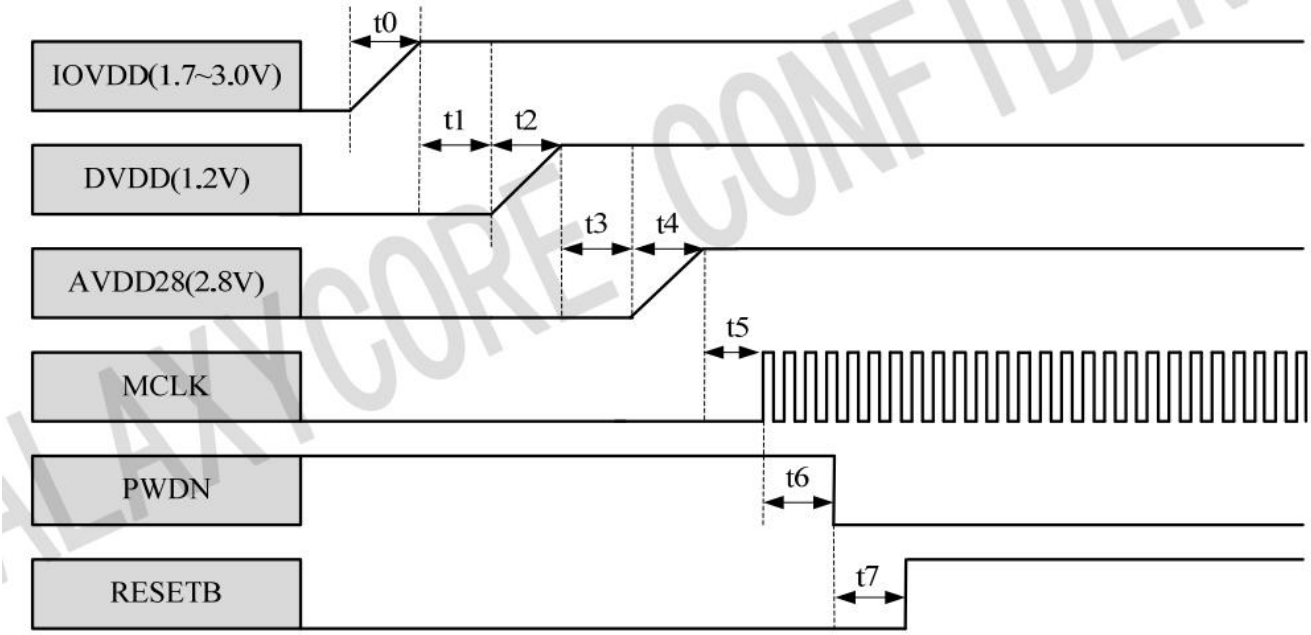
```
vendor\sprd\modules\libcamera\sensor:
classic\Galaxycore\gc8024:
├─── Android.mk
├─── sensor_gc8024_mipi_raw.c
└─── sensor_gc8024_mipi_raw.h
tuning_param\sharkle\Galaxycore\gc8024:
├─── Android.mk
├─── param_manager.c
├─── parameters
|   ├─── NR
|   ├─── .
|   ├─── .
|   ├─── .
|   └─── sensor_gc8024_raw_param_video_2.c
af_drv\dw9714:
├─── Android.mk
├─── af_dw9714.c
└─── af_dw9714.h
otp_drv\driver\gc8024:
├─── Android.mk
├─── gc8024_otp_drv.c
├─── gc8024_otp_drv.h
└─── gc8024_truly_otp.h
```

4 添加 Sensor 驱动

4.1 Sensor 的电源和时钟



电源:DVDD(1.2V), AVDD(2.8V), DOVDD(1.8V) (IOVDD), AF_VDD(2.8V)
Reset: CAM_RST
挂起: CAM_PDN
时钟:MCLK
MIPI :CLK，DATA
I2C:SCL，SDA
注意当我们根据 datasheet 里的上电时序配置完成电源，时钟，reset，powerdown 信号之后，我们可以通过 i2c 去读取 sensor 的 id 了。如果 id 读取不对请检查电路图和上电时序，i2c 地址。上电时序见下图：



4.2 添加 Kernel 驱动

本节介绍的重点是根据原理图的设置如何配置电源，reset，时钟等信号。
Devices 的配置文件
bsp\kernel\kernel4.14\arch\arm\boot\dts\sl8541e-1h10_32b.dts
先把整个 sensor 的 devices 贴在下面，后面会分别介绍

```
sensor_main: sensor-main@6e {
    compatible = "sprd,sensor-main";
    reg = <0x6e>;
    clock-names = "clk_src","sensor_eb",
        "clk_96m","clk_76m8",
        "clk_48m","clk_26m";
    clocks = <&mm_clk CLK_SENSOR0>, <&mm_gate CLK_SENSOR0_EB>,
        <&pll CLK_TWPLL_96M>,<&pll CLK_TWPLL_76M8>,
        <&pll CLK_TWPLL_48M>,<&ext_26m>;
    vddio-supply = <&vddcamio>;
    vddcama-supply = <&vddcama>;
    vddcamd-supply = <&vddcamd>;
    vddcammot-supply = <&vddcammot>;
    dvdd-gpios = <&ap_gpio 32 0>;
    mot-gpios = <&ap_gpio 129 0>;
    sprd,phyid = <1>;
    csi = <&csi0>;
    reset-gpios = <&ap_gpio 44 0>;
    power-down-gpios = <&ap_gpio 46 0>;
};
```

➢ GPIO 配置:

首先介绍一下 gpio 配置，展锐平台比较特殊，gpio function 的选择是在 uboot 阶段配置好的，在 kernel 中是无法配置的，关于 gpio function 的配置这里就不介绍了。请参考 gpio 的配置文档。
DTS 里可以配置哪些 gpio 呢？下面先列举一下
reset-gpios:控制 reset 的 gpio
power-down-gpios:控制 power down 的 gpio
下面四个 gpio 是控制外部 ldo，这四路电源我们可以使用内部的 ldo 也可以用 gpio 控制外部的 dcdc。如果你的原理图上使用 gpio 控制外部 dcdc 那么你需要根据原理图配置下面四个属性。
iovdd-gpios:
avdd-gpios:
dvdd-gpios:
mot-gpios:
下面把我拿一个实例的配置介绍一下：

```
        dvdd-gpios = <&ap_gpio 32 0>;
        mot-gpios = <&ap_gpio 129 0>;
        sprd,phyid = <1>;
        csi = <&csi0>;
        reset-gpios = <&ap_gpio 44 0>;
        power-down-gpios = <&ap_gpio 46 0>;
    };
```

从上面的截图可以看出我们定义了四个 gpio 分别是 dvdd，mot，reset，power。从这里可以看出 dvdd 和马达的电源我们使用了 gpio 来控制外部 dcdc 实现。这些 gpio 的配置根据原理图和我们硬件指导手册来填写就可以了，那关于 iovdd 和 avdd 下面接着介绍。

➢ 内部 LDO 配置:

SIM8500 模块同样引出了一些 ldo 可以使用，上面 gpio 中我们发现缺少了两路电的配置，通过原理图发现 iovdd 和 avdd 用了内部 ldo。我们可以根据原理图把这两路电配置好。

```
        vddio-supply = <&vddcamio>;
        vddcama-supply = <&vddcama>;
```

上面截图中 vddcamio 和 vddcama 这两个是怎么确认的呢？可以通过我们模块的 hd 文档来确认，见下图

LDO2_1V8	125	PO	摄像头 IOVDD 供电	Vnorm = 1.80 V I O max = 200 mA
LDO3_2V8	129	PO	摄像头 AVDD/AFVDD 2.8V 供电	Vnorm = 2.80 V I O max = 150 mA
CAM_DVDD	192	PO	摄像头 DVDD 供电	Vnorm = 1.20 V I O max = 400 mA

模组的 129 引脚对应 vddcama，125 引脚对应 vddcamio，192 引脚对应 vddcamcore。

➢ **时钟配置:**

SIM8500 有两个 camera 的 mclk 时钟，分别对应模块的 74(MAIN_MCLK)和 75(SCAM_MCLK)引脚.

时钟选择，如果你选择 74 引脚作为你的 camera 的 mclk，那么在下图中 clocks 的配置中 mm-clk 配置成 CLK_SENSOR0,mm_gate 配置成 CLK_SENSOR0_EB。如果选择了 75 引脚作为你的 camera 的 mclk 那么 clocks 的配置中 mm-clk 配置成 CLK_SENSOR1,mm_gate 配置成 CLK_SENSOR1_EB。

```
clock-names = "clk_src","sensor_eb",
    "clk_96m","clk_76m8",
    "clk_48m","clk_26m";
clocks = <&mm_clk CLK_SENSOR0>, <&mm_gate CLK_SENSOR0_EB>,
    <&pll CLK_TWPLL_96M>,<&pll CLK_TWPLL_76M8>,
    <&pll CLK_TWPLL_48M>,<&ext_26m>;
```

MCAM_MCLK	74	DO	后摄像头主时钟
SCAM_MCLK	75	DO	前摄像头主时钟

上图是硬件手册中对两组 mclk 的描述。

➢ **I2C 配置:**

默认情况下有两组 i2c 可以给 camera 使用，i2c0(模组的 83,84 引脚)和 i2c1(模组的 166，205 引脚)。I2c 请根据原理图配置。

```
&i2c0 {
    status = "okay";
    clock-frequency = <400000>;

    sensor_main: sensor-main@6e {
        compatible = "sprd,sensor-main";
        reg = <0x6e>;
```

上图中 reg 属性的值是 camera sensor 的 i2c 地址。此地址是 8bit 地址。

CAM_I2C_SCL	83	DO	前后摄像头 I2C 时钟
CAM_I2C_SDA	84	DI/DO	前后摄像头 I2C 数据
I2C1_SCL	166	DO	后摄辅摄像头 I2C 时钟
I2C1_SDA	205	DI/DO	后摄辅摄像头 I2C 数据

上图是硬件手册中两组 i2c 的描述。

- **DCAM 和 PHYID 配置:**

```
        sprd,phyid = <1>;
        csi = <&csi0>;
```

PHYID 的设置可以根据下面的介绍来选择，如果你的硬件连接 csi0，并且是 4lane 的那么 phyid 设置 0。
csi0@4lane: phyid = 0
csi1@4lane: phyid = 1
csi0_s@2lane: phyid = 3
csi0_m@2lane: phyid = 4
DCAM 对应的是 csi 的选择，这个 csi 是内部的数字处理器，SIM8500 有两个 DCAM0(dts 里叫做 csi0 注意这个不是硬件接口上的 csi0)，DCAM1(csi1),这两个 DCAM 是并列的，也就是说 SIM8500 可以同时打开两路 camera。另外只要不同时打开这个配置随便选择，如果需要同时打开那么两个 camera 的 csi 不能一样。

- **Kernel 驱动代码的位置:**

上面介绍了 camera 的 device 的定义，下面把 camera kernel driver 的代码路径，
bsp\modules\camera:
如果你需要看 gpio，ldo 等是怎么被调用的请查看
bsp\modules\camera\sensor\sprd_sensor_drv.c,sprd_sensor_core.c
这里面定义了一系列的节点给 vendor 下用户控件的 camera 驱动调用。下面列举一个 af_vdd 的定义:

```
static int sprd_sensor_io_set_cammot(struct sprd_sensor_file_tag *p_file,
                    unsigned long arg)
{
    int ret = 0;
    unsigned int vdd_val;

    ret = copy_from_user(&vdd_val, (unsigned int *)arg,
                sizeof(unsigned int));
    if (ret == 0) {
        vdd_val = sprd_sensor_get_voltage_value(vdd_val);
        ret = sprd_sensor_set_voltage_by_gpio(p_file->sensor_id,
            vdd_val,
            SPRD_SENSOR_MOT_GPIO_TAG_E);
        //if (ret)
            ret = sprd_sensor_set_voltage(p_file->sensor_id,
                        vdd_val,
                        SENSOR_REGULATOR_CAMMOT_ID_E);
    }

    return ret;
}
```

- **Camera 的三个设备和 sensor_config.xml 中 Slotid 的对应关系:**

sprd,sensor-main，sprd,sensor-main2，sprd,sensor-sub 这个设备可以都可以配置成前摄，后摄，后辅摄，但是要注意一点 main 对应 Slotid0，sub 对应 Slotid1，main2 对应 Slotid2，如果只有一个摄像头 Slotid 必须定义成 0.

```
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

5 VENDOR 下用户空间代码移植

先介绍一下 device\sprd\sharkle\sl8541e_1h10_32b\BoardConfig.mk 文件中需要配置的变量

```
CAMERA_SUPPORT_SIZE := 8M
FRONT_CAMERA_SUPPORT_SIZE :=
BACK_EXT_CAMERA_SUPPORT_SIZE :=
```

CAMERA_SUPPORT_SIZE：后摄分辨率
FRONT_CAMERA_SUPPORT_SIZE：前摄分辨率
BACK_EXT_CAMERA_SUPPORT_SIZE：后辅摄分辨率

```
CAMERA_SENSOR_TYPE_BACK := "gc8024"
CAMERA_SENSOR_TYPE_FRONT :=
CAMERA_SENSOR_TYPE_BACK_EXT :=
```

CAMERA_SENSOR_TYPE_BACK：后摄 sensor 型号
CAMERA_SENSOR_TYPE_FRONT：前摄 sensor 型号
CAMERA_SENSOR_TYPE_BACK_EXT：后辅摄 sensor 型号
Note：sensor 型号需与存放 driver 文件夹同名

```
#tuning param support list
TUNING_PARAM_LIST := "gc8024,s5k3l6xx03"
```

sensor 参数名称需与存放 tuning
Note：parameter 中 sensor 名相同
以上参数在 vendor 的编译过程中使用，会根据定义启动相应的编译规则和宏定义的添加，有兴趣可以查看 vendor\sprd\modules\libcamera\Android.mk，SprdCtrl.mk 等文件

5.1 VENDOR 下的驱动移植

➢ **下面以 gc8024 为例介绍:**
添加模组厂提供的驱动文件到 vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore 目录，注意 Galaxycore 这个是 sensor 厂家的名称，不同厂家的驱动放置到对应目录下。

箭头指向的驱动目录的名称必须与前面 BoardConfig.mk 定义的名字一样。
添加编译规则
首先在 vendor\sprd\modules\libcamera\sensor\sensor_drv\sensor_lib_cfg.mk 中添加 PRODUCT_PACKAGES

```
PRODUCT_PACKAGES += libsensor_gc8024
```

名字就是 vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore\gc8024\Android.mk 中生成

的 so 库的名字。

```
LOCAL_MODULE := libsensor_gc8024
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)

#include $(call first-makefiles-under,$(LOCAL_PATH))
```

- **Sensor 驱动中必须要检查的部分:**

打开下面的文件检查 i2c 地址，必须与前面 dts 中设置的一致

vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore\gc8024\sensor_gc8024_mipi_raw.h

```
#define I2C_SLAVE_ADDR 0x6e /* 8bit slave address*/
```

同样检查.h 文件中 lane number 设置是否正确，我使用的 gc8024 模组是 2lane 此处设置为 2.

```
#define LANE_NUM 2
```

打开下面的文件与 datasheet 对比检查上电时序

vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore\gc8024\sensor_gc8024_mipi_raw.c

```
/*==============================================================================
 * Description:
 * sensor power on
 * please modify this function acording your spec
 *============================================================================*/
static cmr_int gc8024_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 = MIPI_RAW_INFO.power_down_level;
    BOOLEAN reset_level = MIPI_RAW_INFO.reset_pulse_level;

    if (SENSOR_TRUE == power_on) {
        hw_sensor_power_down(sns_drv_cxt->hw_handle, power_down);
        usleep(12 * 1000);
        hw_sensor_set_iovdd_val(sns_drv_cxt->hw_handle, iovdd_val);
        usleep(1 * 1000);
        hw_sensor_set_dvdd_val(sns_drv_cxt->hw_handle, dvdd_val);
        usleep(1 * 1000);

        hw_sensor_set_avdd_val(sns_drv_cxt->hw_handle, avdd_val);
        usleep(1 * 1000);
        hw_sensor_set_mclk(sns_drv_cxt->hw_handle, EX_MCLK);
        usleep(1 * 1000);
        hw_sensor_power_down(sns_drv_cxt->hw_handle, !power_down);
        usleep(1 * 1000);
        hw_sensor_set_reset_level(sns_drv_cxt->hw_handle, !reset_level);
    } else {
        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);
        hw_sensor_power_down(sns_drv_cxt->hw_handle, !power_down);
    }
    SENSOR_LOGI("(1:on, 0:off): %ld", power_on);
    return SENSOR_SUCCESS;
}
```

5.2 VENDOR 下的效果代码移植

➢ **拿到模组厂提供的效果文件:**

拷贝模组厂提供的效果文件到
vendor\sprd\modules\libcamera\sensor\tuning_param\sharkle\Galaxycore\gc8024 目录下。
在下面的文件中添加 gc8024 的效果库
vendor\sprd\modules\libcamera\sensor\tuning_param\tunning_lib_cfg.mk

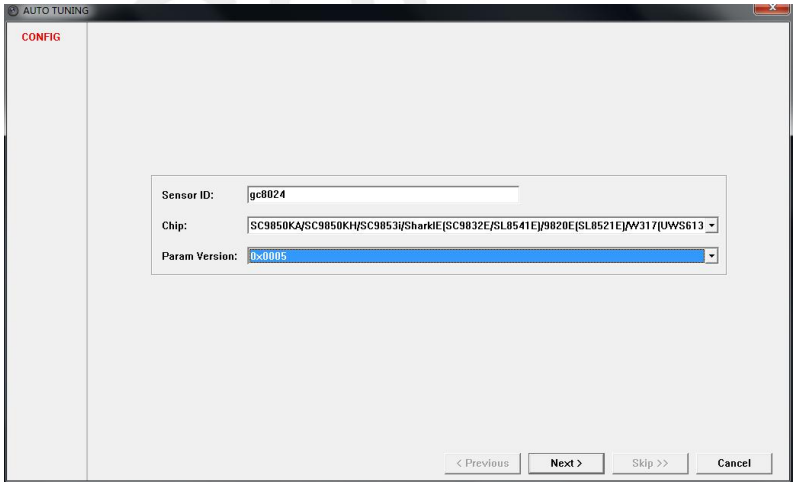
```
#ifeq ($(strip $(CHIP_NAME)),sharkle)
#sp9832e_1h10_go
PRODUCT_PACKAGES += libparam_ov13855_back_main
PRODUCT_PACKAGES += libparam_ov8856_shine_back
PRODUCT_PACKAGES += libparam_ov5675_front_main
PRODUCT_PACKAGES += libparam_ov5675_dual
PRODUCT_PACKAGES += libparam_s5k4h7_front_main_wifi5g
PRODUCT_PACKAGES += libparam_gc2385_wifi5g
PRODUCT_PACKAGES += libparam_gc8024
#endif
```

同样上面的名字是下面文件中生成库的名字
vendor\sprd\modules\libcamera\sensor\tuning_param\sharkle\Galaxycore\gc8024\Android.mk

```
LOCAL_MODULE            := libparam_$(PARAM_NAME)
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)
```

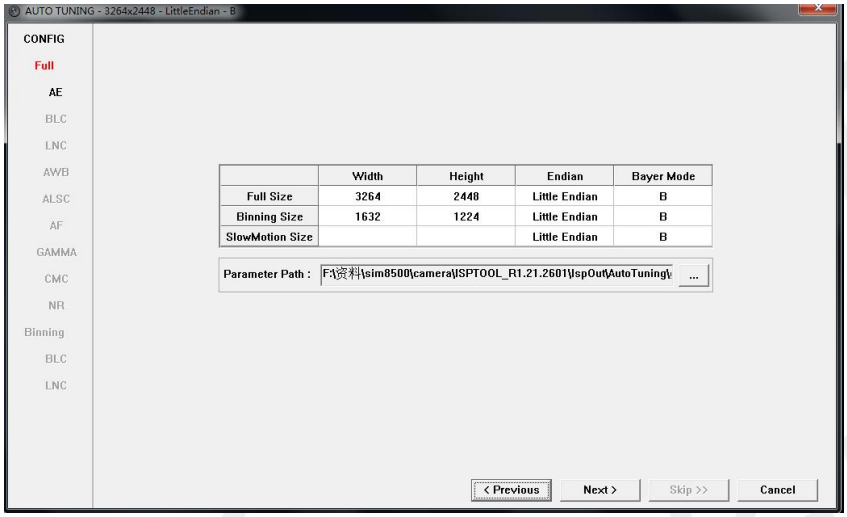
➢ **自己生成效果文件:**

需要用到 IspTool，此工具展锐提供，如果你现在需要自己生成默认的效果文件你可以找我们要此工具。下面简单介绍一下怎么通过此工具生成效果文件
1 在菜单 file 中打开 auto tuning



Sensor ID 填写前面你在效果驱动目录下定义的文件夹的名字，其他两项按照图片中填写

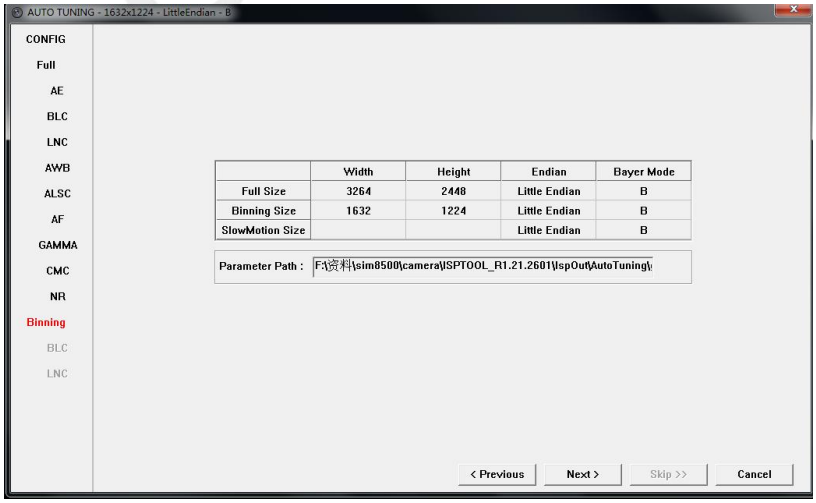
填写完成执行 next



上图中两个分辨率需要你到 vendor 的 sensor 驱动文件找到下面的文件 vendor\sprd\modules\libcamera\sensor\sensor_drv\classic\Galaxycore\gc8024\sensor_gc8024_mipi_raw.h 里定义 preview 和 snapshot 两个分辨率

```
#define SNAPSHOT_WIDTH 3264
#define SNAPSHOT_HEIGHT 2448
#define PREVIEW_WIDTH 1632
#define PREVIEW_HEIGHT 1224
```

之后你需要执行 next，然后从下个界面开始一直执行 skip，直到下面的界面



在此界面执行 next，之后再次一直执行 skip，直到最后提升效果文件生成成功。

生成的效果文件存放在 isptool 工具所在目录下的 IspOut\AutoTuning\gc8024 中。拷贝 gc8024 下的内容到 vendor\sprd\modules\libcamera\sensor\tuning_param\sharkle\Galaxycore\gc8024\parameters 中，其中 gc8024\parameters 是我们之前创建好的。
经过上面的流程我们自己创建了默认的效果文件。

下面来看一个文件(这部分的介绍如果你直接拿到模组厂提供的效果文件可以不用管，如果你需要自己生成默认的效果文件那你需要自己添加 param_manager.c 这个文件。因为参数工具生成的效果文件只有 parameters 这个文件夹)
vendor\sprd\modules\libcamera\sensor\tuning_param\sharkle\Galaxycore\gc8024\param_manager.c

```
void *tuning_param_get_ptr(void) {
    cmr_int rtn = 0;
#include "parameters/sensor_gc8024_raw_param_main.c"
    return &s_gc8024_mipi_raw_info;
}
```

上图中 include 包含的文件是我们效果文件夹下的一个.c 文件。
第二个箭头指向的是这个.c 文件里的一个结构体，

```
static struct sensor_raw_info s_gc8024_mipi_raw_info=
{
    &s_gc8024_version_info,
    {
        {s_gc8024_tune_info_common, sizeof(s_gc8024_tune_info_common)},
        {s_gc8024_tune_info_prv_0, sizeof(s_gc8024_tune_info_prv_0)},
        {s_gc8024_tune_info_prv_1, sizeof(s_gc8024_tune_info_prv_1)},
        {NULL, 0},
        {NULL, 0},
        {s_gc8024_tune_info_cap_0, sizeof(s_gc8024_tune_info_cap_0)},
        {s_gc8024_tune_info_cap_1, sizeof(s_gc8024_tune_info_cap_1)},
        {NULL, 0},
        {NULL, 0},
        {s_gc8024_tune_info_video_0, sizeof(s_gc8024_tune_info_video_0)},
        {s_gc8024_tune_info_video_1, sizeof(s_gc8024_tune_info_video_1)},
        {s_gc8024_tune_info_video_2, sizeof(s_gc8024_tune_info_video_2)},
        {NULL, 0},
    },
```

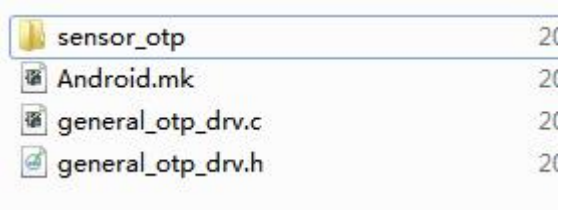
➢ 添加我们的效果文件到 **xml** 中:
device\sprd\sharkle\sl8541e_1h10_32b\camera\sensor_config.xml

```
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

5.3 VENDOR 下 OTP 驱动移植

➢ **移植 eeprom 的驱动:**

如果你的模组支持 OTP，那现在你需要把模组厂提供给你 eeprom 的驱动移植到平台上，下面我已系统默认的 eeprom 驱动来介绍。
下面下看一下 general eeprom 的驱动目录和内容
vendor\sprd\modules\libcamera\sensor\otp_drv\driver\general



同样如果你新加入自己的 eeprom 驱动需要确认两点：
1、Android.mk 中生成 so 库名称

```
LOCAL_MODULE := libotp_general
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)
```

2、在 vendor\sprd\modules\libcamera\sensor\otp_drv\otp_lib_cfg.mk 中添加 PRODUCT_PACKAGES

```
PRODUCT_PACKAGES += libotp_general \
                    libotp_s5k5e8yx_jd \
                    libotp_ov13855 \
                    libotp_gc5035_common \
                    libotp_s5k4h7_tsp
```

➢ **添加 eeprom 到 xml 中:**

device\sprd\sharkle\sl8541e_1h10_32b\camera\sensor_config.xml

```
<cameramodulecfg>
  <slotid>0</slotid>
  <sensorname>s5k3l6xx03</sensorname>
  <facing>back</facing>
  <orientation>180</orientation>
  <resource_cost>50</resource_cost>
  <otp>
      <e2prom>
          <otpname>general</otpname>
          <I2cAddr>0xa0</I2cAddr>
          <E2promNum>2</E2promNum>
          <E2promSize>8192</E2promSize>
      </E2prom>
  </OTP>
  <TuningParameter>
      <TuningName>s5k3l6xx03</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

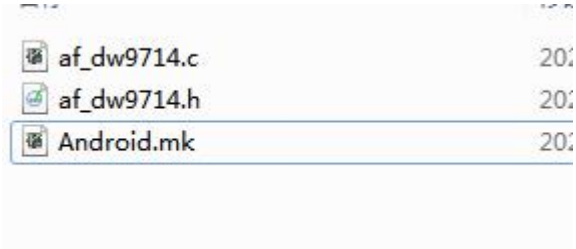
5.4 VENDOR 下 AF 驱动移植

➢ **移植 AF 的驱动:**

正常情况下在 vendor\sprd\modules\libcamera\sensor\af_drv 目录已经包含了大部分的 AF 驱动，如果此目录下没有你的 camera 模组的 AF 驱动你需要找模组厂提供。

下面以 DW9714 来介绍：

vendor\sprd\modules\libcamera\sensor\af_drv\dw9714



从厂家拿到驱动文件放入目录 vendor\sprd\modules\libcamera\sensor\af_drv\dw9714 目录，首先我们需要确认一下 i2c 地址配置是否正确

```
#define DW9714_VCM_SLAVE_ADDR (0x18 >> 1)
#define MOVE_CODE_STEP_MAX 20
#define WAIT_STABLE_TIME 10     // ms
#define DW9714_POWERON_DELAY 12 // ms
```

把 android.mk 中定义的 so 库名字添加到文件

vendor\sprd\modules\libcamera\sensor\af_drv\vcm_lib_cfg.mk 里的 PRODUCT_PACKAGES 中

```
endif

LOCAL_MODULE := libvcm_dw9714
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)

#include $(call first-makefiles-under,$(LOCAL_PATH))
```

```
PRODUCT_PACKAGES += libvcm_dw9714p \
                    libvcm_lc898213 \
                    libvcm_dw9768v \
                    libvcm_dw9800 \
                    libvcm_dw9714 \
                    libvcm_dw9718s \
                    libvcm_zc524
```

➢ **移植 AF 到 xml:**

device\sprd\sharkle\sl8541e_1h10_32b\camera\sensor_config.xml

```
    <CameraModuleCfg>
      <SlotId>0</SlotId>
      <SensorName>gc8024</SensorName>
      <Facing>BACK</Facing>
      <Orientation>270</Orientation>
      <Resource_cost>100</Resource_cost>
      <VCM>
          <AfName>dw9714</AfName>
          <Mode>0</Mode>
      </VCM>
      <TuningParameter>
          <TuningName>gc8024</TuningName>
      </TuningParameter>
    </CameraModuleCfg>
```

5.5 SENSOR 驱动兼容：

针对不同的器件我们经常需要调试一供，二供，下面我们简单介绍一下我们如果调试多个 sensor 应该怎们兼容，下面以连接到 SlotID 为 0 的摄像头为例介绍。

正常情况下我们 device\sprd\sharkle\sl8541e_1h10_32b\camera\sensor_config.xml 中一个 SlotID 只有一个见下图

```
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

如果你需要兼容多个 sensor，很简单，根据你调试好的 sensor，添加多个 SlotID 为 0 的配置即可。

```
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024</TuningName>
  </TuningParameter>
</CameraModuleCfg>
<CameraModuleCfg>
  <SlotId>0</SlotId>
  <SensorName>gc8024_bak</SensorName>
  <Facing>BACK</Facing>
  <Orientation>270</Orientation>
  <Resource_cost>100</Resource_cost>
  <VCM>
      <AfName>dw9714</AfName>
      <Mode>0</Mode>
  </VCM>
  <TuningParameter>
      <TuningName>gc8024_bak</TuningName>
  </TuningParameter>
</CameraModuleCfg>
```

6 编译

- 编译 **kernel:**

修改完 **kernel** 下的驱动需要 **make bootimage，make dtboimage。**
下载通过 **fastboot flash boot boot.img**
Fastboot flash dtbo dtbo.img

- 编译 **vendor** 下的 **camera** 库**:**

进入 **endor\sprd\modules\libcamera** 目录执行 **mm。关于 gc8024** 的库文件 **libsensor_gc8024.so** 和 **libparam_gc8024.so** 执行下列指令 **push** 到 **vendor/lib/**目录下
adb root && adb remount && adb push libsensor_gc8024.so /vendor/lib/ && adb push libparam_gc8024.so /vendor/lib/

7 DEBUG

- 读取 ID 成功的 LOG:

```
14042 03-04 09:05:37.025   366  2320 I sensor_gc8024: 507, gc8024_drv_power_on: (1:on, 0:off): 1
14043 03-04 09:05:37.025   366  2320 I sensor_gc8024: 593, gc8024_drv_identify: mipi raw identify
14044 03-04 09:05:37.025   374   374 I gralloc : gralloc_unregister hnd=0xb2441cc0, share_fd=31, format=1, internal_format=0x1, byte_stride=5760, flags=4,
      ight=1280, stride=1440, base=0xaf544000, writeOwner=0, min_pgsz=4096
14045 03-04 09:05:37.027   374   374 I gralloc : gralloc_register hnd=0xb24419c0, share_fd=28, format=1, internal_format=0x1, byte_stride=2816, flags=4, us
      928, stride=704, base=0xaf9ce000, writeOwner=0, min_pgsz=4096
14046 03-04 09:05:37.027   374   374 W DPUModule: Warning: Dispc cannot support 6 layer blending(0)
14047 03-04 09:05:37.028   425   425 E          : Shared attribute region not available to be mapped
14048 03-04 09:05:37.029  2281  2281 I CAM_PhotoUI: initUI cost: 0
14049 03-04 09:05:37.029   366  2320 I sensor_gc8024: 601, gc8024_drv_identify: Identify: PID = 80, VER = 24
14050 03-04 09:05:37.030   366  2320 I sensor_gc8024: 603, gc8024_drv_identify: this is gc8024 sensor
14051 03-04 09:05:37.030   366  2320 I sensor_gc8024: 629, gc8024_drv_identify: this is gc8024 sensor
14052 03-04 09:05:37.030   366  2320 I sns_drv_u: 3216, sensor_drv_ic_identify: sensor ic identify ok
```

- 读取 ID 失败的 LOG:

```
03-04 08:50:58.771   367   367 E hw_sensor: 360, hw_sensor_set_reset_level: hal_err failed,  level = 1, ret=-1
03-04 08:50:58.771   367   367 I sensor_gc8024: 507, gc8024_drv_power_on: (1:on, 0:off): 1
03-04 08:50:58.771   367   367 I sensor_gc8024: 593, gc8024_drv_identify: mipi raw identify
03-04 08:50:58.814   491   491 D APR     : SSRListener GetSystemServerPid = 2 times
03-04 08:50:58.832   367   367 E hw_sensor: 621, hw_sensor_read_reg: hal_err failed,  addr = 0xf0, value=0xad44, bit=128, ret=-1
03-04 08:50:58.832   367   367 E sensor_gc8024: 634, gc8024_drv_identify: hal_err sensor identify fail, pid value = ffff
03-04 08:50:58.832   367   367 I sns_drv_u: 3226, sensor_drv_ic_identify: sensor drv ic identify power off
03-04 08:50:58.832   367   367 I sns_drv_u: 311, sensor_power_on: E:power_on = 0
03-04 08:50:58.874   407   407 D APM::AudioPolicyEngine/PFWWrapper: start: in
03-04 08:50:58.875   367   367 E hw_sensor: 320, hw_sensor_power_down: hal_err failed,  power_level = 1, ret=-1
03-04 08:50:58.875   367   367 E hw_sensor: 360, hw_sensor_set_reset_level: hal_err failed,  level = 0, ret=-1
03-04 08:50:58.875   367   367 E hw_sensor: 392, hw_sensor_set_mclk: hal_err failed,  mclk = 0, ret = -1
03-04 08:50:58.875   367   367 I hw_sensor: 395, hw_sensor_set_mclk: mclk 0, ret 1
03-04 08:50:58.875   367   367 E hw_sensor: 177, _hw_sensor_set_avdd: hal_err failed,  vdd_value = 11, ret=-1
03-04 08:50:58.875   367   367 I hw_sensor: 250, hw_sensor_set_avdd_val: vdd_val is 11, set result is =-1
03-04 08:50:58.875   367   367 E hw_sensor: 191, _hw_sensor_set_dvdd: hal_err failed,  vdd_value = 11, ret=-1
03-04 08:50:58.875   367   367 I hw_sensor: 261, hw_sensor_set_dvdd_val: vdd_val is 11, set result is =1
03-04 08:50:58.875   367   367 E hw_sensor: 205, _hw_sensor_set_iovdd: hal_err failed,  vdd_value = 11, ret=-1
```

以上 log 可以到 vendor 下的 sensor driver 里查看相关的代码

```
static cmr_int gc8024_drv_identify(cmr_handle handle, cmr_uint param) {
    UNUSED(param);
    cmr_u16 pid_value = 0x00;
    cmr_u16 ver_value = 0x00;
    cmr_u16 ret_value = SENSOR_FAIL;

    SENSOR_LOGI("mipi raw identify");
    SENSOR_IC_CHECK_PTR(handle);
    struct sensor_ic_drv_cxt *sns_drv_cxt = (struct sensor_ic_drv_cxt *)handle;

    pid_value = hw_sensor_read_reg(sns_drv_cxt->hw_handle, GC8024_PID_ADDR);

    if (GC8024_PID_VALUE == pid_value) {
        ver_value = hw_sensor_read_reg(sns_drv_cxt->hw_handle, GC8024_VER_ADDR);
        SENSOR_LOGI("Identify: PID = %x, VER = %x", pid_value, ver_value);
        if (GC8024_VER_VALUE == ver_value) {
            SENSOR_LOGI("this is gc8024 sensor");

#ifdef FEATURE_OTP
            /*if read otp info failed or module id mismatched ,identify failed
             * ,return SENSOR_FAIL ,exit identify*/
            if (PNULL != s_gc8024_raw_param_tab_ptr->identify_otp) {
                SENSOR_LOGI("identify module_id=0x%x",
                            s_gc8024_raw_param_tab_ptr->param_id);
                // set default value
                memset(s_gc8024_otp_info_ptr, 0x00, sizeof(struct otp_info_t));
                ret_value = s_gc8024_raw_param_tab_ptr->identify_otp(
                    s_gc8024_otp_info_ptr);

                if (SENSOR_SUCCESS == ret_value) {
                    SENSOR_LOGI(
                        "identify otp sucess! module_id=0x%x, module_name=%s",
                        s_gc8024_raw_param_tab_ptr->param_id, MODULE_NAME);
                } else {
                    SENSOR_LOGI("identify otp fail! exit identify");
                    return ret_value;
                }
            } else {
                SENSOR_LOGI("no identify_otp function!");
            }
#endif
            ret_value = SENSOR_SUCCESS;
            SENSOR_LOGI("this is gc8024 sensor");
        } else {
            SENSOR_LOGI("Identify this is %x%x sensor", pid_value, ver_value);
        }
    } else {
        SENSOR_LOGE("sensor identify fail, pid_value = %x", pid_value);
    }

    return ret_value;
}
```

- 获取数据流失败的 log:

当出现下面的 log 时说明获取数据流出错了，请检查 mipi 总线的状态。