全志科技
Allwinner Technology

Allwinner

Camera 模块开发说明文档

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2013-03-17	曾令莹	建立初始版本
V1.1	2013-04-28	曾令莹	补充 device 调试的说明
V1.2	2013-05-31	曾令莹	补充 sensor 器件驱动的格式说明和编写要求，补充 cam_detect 模块，补光灯部分代码的说明，硬件原理图注意事项
V1.3	2013-07-28	曾令莹	补充 device 驱动调试说明
V1.4	2014-03-10	杨峰	增加 hawkview ISP 配置说明
V1.5	2014-04-18	杨峰	增加 camera 自动检测 sensor_list_cfg.ini 配置说明
V1.6	2014-06-03	杨峰	增加一些 camera 调试时常见问题说明
V1.7	2014-06-03	杨峰	增加一些 CCI_Client 调试节点使用方法
V1.8	2015-06-12	赵威	更新代码结构、更新 sys_config.fex、更新驱动加载、更新 flash_led 驱动部分、删掉 raw sensor 相关、添加一个 CCI 不通的案例

目 录

1. 前言

1.1. 编写目的

介绍本文档的编写目的。
了解 camera 模块在 SUNXI 平台上的硬件和驱动开发。

1.2. 适用范围

介绍本模块设计适用的平台：SUNXI 平台，VFE 模块。

1.3. 相关人员

SW/TS/客户开发人员。

2. 模块介绍

2.1. 模块功能介绍

用于接收并行接口的 sensor 信号或者是 bt656 格式的信号。

2.2. 硬件介绍

目前 A64 上支持一组最高 8bit 的并口 CSI 接口，内部的带有一个简单（只支持 scale 和 rotate）的 ISP，A64 只支持 YUV sensor。

如果使用 BT656 格式的器件输入信号给 CSI，其输入接口是并口的 D11~D3 这 4 根数据线（高位对齐）。

2.3. 源码结构介绍

驱动路径位于 linux-3.10\drivers\media\platform\sunxi-vfe

```
sunxi-vfe:.
|   bsp_common.c            ;底层 bsp 共用的函数
|   bsp_common.h            ;底层 bsp 共用函数头文件
|   config.c                  ;读取 sysconfig 的参数配置和 isp 参数
|   config.h                  ;读取 sysconfig 和 isp 参数函数的头文件
|   Kconfig
|   Makefile
|   platform_cfg.h          ;区分各个平台的头文件
|   vfe.c                   ;v4l2 驱动实现主体（包含视频接口和 ISP 部分）
|   vfe.h                   ;v4l2 驱动头文件
|   vfe_os.c                ;系统资源函数实现（pin，clock，memory）
|   vfe_os.h                ;系统资源函数头文件
|   vfe_subdev.c            ; sensor 调用 vfe 资源函数
|   vfe_subdev.h            ; sensor  调用 vfe 资源函数头文件
|
+---actuator
|       actuator.c          ; vcm driver 的一般行为
|       actuator.h          ; vcm driver 的头文件
|       ad5820_act.c        ;  具体 vcm driver 型号实现
|       dw9714_act.c        ;  具体 vcm driver 型号实现
|       Makefile
|       ov8825_act.c        ;  具体 vcm driver 型号实现
|
+---csi
|       bsp_csi.c           ;底层 csi bsp 函数
|       bsp_csi.h           ;底层 csi bsp 函数头文件
|       csi_reg.c           ; csi 硬件底层实现
|       csi_reg.h           ; csi 硬件底层实现头文件
```

```
│       csi_reg_i.h            ;csi  寄存器资源头文件
│       sunxi_csi.c             ;csi  子模块驱动原文件
│       sunxi_csi.h             ;csi  子模块驱动头文件
├──csi_cci
│       bsp_cci.c                  ;底层 cci bsp 函数
│       bsp_cci.h                  ;底层 cci bsp 函数头文件
│       cci_helper.c               ;cci  帮助函数，供 sensor 驱动调用
│       cci_helper.h               ;cci  帮助函数头文件
│       cci_platform_drv.c          ;cci  平台驱动源文件
│       cci_platform_drv.h          ;cci  平台驱动头文件
│       csi_cci_reg.c              ; cci 硬件底层实现
│       csi_cci_reg.h              ; cci 硬件底层实现头文件
│       csi_cci_reg_i.h            ;cci  寄存器资源头文件
│
├──device
│       camera.h                ; camera 公用结构体头文件
│       camera_cfg.h             ;camera ioctl 扩展命令头文件
│       gc0307.c                 ;具体的 sensor 驱动
│       gc0308.c                 ;具体的 sensor 驱动
│       gc2035.c                 ;具体的 sensor 驱动
│       gt2005.c                 ;具体的 sensor 驱动
│       hi253.c                  ;具体的 sensor 驱动
│       Makefile
│       ov5640.c                  ;具体的 sensor 驱动
│       ov5647.c                  ;具体的 sensor 驱动
│       ov5650.c                  ;具体的 sensor 驱动
│       s5k4e1.c                   ;具体的 sensor 驱动
│       s5k4e1_mipi.c              ;具体的 sensor 驱动
│       s5k4ec.c                  ;具体的 sensor 驱动
│       s5k4ec_mipi.c              ;具体的 sensor 驱动
│       t4k05.c                    ;具体的 sensor 驱动
│       t8et5.c                    ;具体的 sensor 驱动
│
├──flash_light
│       flash.h                 ;led 补光灯驱动头文件
│       flash_io.c               ;led 补光灯 io 控制实现
│       Makefile
│
├──lib
│       bsp_isp.h               ;底层 isp bsp 函数头文件
│       bsp_isp_algo.h           ;底层 isp  算法 bsp 函数头文件
│       bsp_isp_comm.h          ;底层 isp 共用函数头文件
│       isp_module_cfg.h         ;isp 里面各模块功能配置的头文件
│       libisp                  ;isp 的函数库
```

```
|
├──mipi_csi
|       bsp_mipi_csi.c                ;底层 mipi bsp 函数
|       bsp_mipi_csi.h                ;底层 mipi bsp 函数头文件
|   ├──dphy
|   |       dphy.h              ;mipi dphy 头文件
|   |       dphy_reg.c          ;mipi dphy 底层实现函数
|   |       dphy_reg.h          ;mipi dphy 底层实现函数头文件
|   |       dphy_reg_i.h        ;mipi dphy  寄存器资源头文件
|   |
|   └──protocol
|           protocol.h          ;mipi  协议层头文件
|           protocol_reg.c      ;mipi  协议层底层实现
|           protocol_reg.h      ;mipi  协议层底层实现头文件
|           protocol_reg_i.h    ;mipi  协议层寄存器资源头文件
└──test
|     csi_test                ;测试用例
|     csi_test.c              ;测试用例源码
|     Makefile
|     sunxi_camera.h          ;测试用例使用到的头文件
|
└──utility
    cfg_op.c                  ;读取 ini 文件的实现函数
    cfg_op.h                  ;读取 ini 文件函数对应的头文件
    sensor_info.c             ;sensor 信息文件
    sensor_info.h             ;sensor 信息头文件
    vfe_io.h                  ;vfe 模块寄存器操作头文件
```

2.4. 模块配置介绍

2.4.1. 软件配置

如果是从旧的内核更新的最新的代码，需要注意：

1、 同步更新 android 和 linux 代码；

2、 编译内核时，**先删除.config 并 make clean 后再编译内核**；

3、 建 议 删 掉 Android device 目 录 下 面 的 modules 目 录 ， 如 ： device\softwinner\tulip-t1\modules\modules，清除之前旧版驱动编译生成的旧版本 ko 文件。

2.4.1.1. menuconfig 配置说明

1. Device Drivers -> <*>I2C support

2. Device Drivers -> <*>Multimedia support -> [*] Cameras/video grabbers support

3. Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> sunxi video fornt end (camera and etc)driver

```
4. Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> v4l2 driver for SUNXI
```

2.4.1.2. Android 配置

Camera 相关模块加载有所调整，使用的时候请按顺序加载如下若干 ko 文件。例如某方案使用 gc2155 + gc0328，配置如下：

```
#csi module
    insmod /system/vendor/modules/videobuf2-core.ko
    insmod /system/vendor/modules/videobuf2-memops.ko
    insmod /system/vendor/modules/videobuf2-dma-contig.ko
    insmod /system/vendor/modules/io.ko
    insmod /system/vendor/modules/gc2155.ko
    insmod /system/vendor/modules/gc0328c.ko
    insmod /system/vendor/modules/vfe_v4l2.ko
```

vfe_v4l2.ko 必须在最后加载，其它 ko 可以按照上面的相对顺序加载。

2.4.1.3. camera.cfg 配置

在 device\softwinner\tulip-xxx\configs 中，Camera.cfg 中增加配置项 use_builtin_isp，用于配置是否启用 IC 的自带的 ISP 模块。use_builtin_isp = 1 表示启用 ISP；use_builtin_isp = 0 表示不启用。

```
;------------------------------------------------------------------------
; 1 for camera without isp(using built-in isp of Axx)
; 0 for camera with isp
;------------------------------------------------------------------------
use_builtin_isp = 1
```

2.4.2. 硬件配置

2.4.2.1. sys_config.fex 配置

A64 CSI 在 sys_config.fex 中对应的字段为：[csi0]、[csi0/csi0_dev0]、[csi0/csi0_dev1]。下面举例说明在实际使用中应该如何配置：

1. **使用一个并口 camera 模组：**需要配置[csi0]的公用部分和[csi0/csi0_dev0]的 csi0_dev0_(x)部分，[csi0]中 csi0_used 设置为 1。

2. **使用两个并口的 camera 模组：**同样需要配置[csi0]公用部分，另外需要配置[csi0/csi0_dev0]的 csi0_dev0_(x)部分和[csi0/csi0_dev1]的 csi0_dev1_(x)部分，[csi0]中 csi0_used 设置为 1。

下面给出一个参考配置：具体填写方法请参照以下说明：

```
;------------------------------------------------------------------------------
;csi (COMS Sensor Interface) configuration
;csi(x)_dev(x)_used: 0:disable 1:enable
;csi(x)_dev(x)_isp_used 0:not use isp 1:use isp
;csi(x)_dev(x)_fmt: 0:yuv 1:bayer raw rgb
;csi(x)_dev(x)_stby_mode: 0:not shut down power at standby 1:shut down power at standby
;csi(x)_dev(x)_vflip: flip in vertical direction 0:disable 1:enable
;csi(x)_dev(x)_hflip: flip in horizontal direction 0:disable 1:enable
;csi(x)_dev(x)_iovdd: camera module io power handle string, pmu power supply
;csi(x)_dev(x)_iovdd_vol: camera module io power voltage, pmu power supply
;csi(x)_dev(x)_avdd:      camera module analog power handle string, pmu power supply
;csi(x)_dev(x)_avdd_vol:      camera module analog power voltage, pmu power supply
;csi(x)_dev(x)_dvdd:      camera module core power handle string, pmu power supply
;csi(x)_dev(x)_dvdd_vol:      camera module core power voltage, pmu power supply
;csi(x)_dev(x)_afvdd:     camera module vcm power handle string, pmu power supply
;csi(x)_dev(x)_afvdd_vol:     camera module vcm power voltage, pmu power supply
;fill voltage in uV, e.g. iovdd = 2.8V, csix_iovdd_vol = 2800000
;fill handle string as below:
;axp22_eldo3
;axp22_dldo4
;axp22_eldo2
;fill handle string "" when not using any pmu power supply
;------------------------------------------------------------------------------

[csi0]
csi0_used               = 1
csi0_sensor_list        = 0
csi0_pck                    = port:PE00<2><default><default><default>
csi0_mck                    = port:PE01<1><0><1><0>
csi0_hsync              = port:PE02<2><default><default><default>
csi0_vsync              = port:PE03<2><default><default><default>
csi0_d0                     = port:PE04<2><default><default><default>
csi0_d1                     = port:PE05<2><default><default><default>
csi0_d2                     = port:PE06<2><default><default><default>
csi0_d3                     = port:PE07<2><default><default><default>
csi0_d4                     = port:PE08<2><default><default><default>
csi0_d5                     = port:PE09<2><default><default><default>
csi0_d6                     = port:PE10<2><default><default><default>
csi0_d7                     = port:PE11<2><default><default><default>
```

```
csi0_sck                    = port:PE12<2><default><default><default>
csi0_sda                    = port:PE13<2><default><default><default>

[csi0/csi0_dev0]
csi0_dev0_used              = 1
csi0_dev0_mname             = "ov5640"
csi0_dev0_twi_addr          = 0x78
csi0_dev0_pos           = "rear"
csi0_dev0_isp_used          = 1
csi0_dev0_fmt               = 0
csi0_dev0_stby_mode         = 0
csi0_dev0_vflip             = 0
csi0_dev0_hflip             = 0
csi0_dev0_iovdd             = "iovdd-csi"
csi0_dev0_iovdd_vol         = 2800000
csi0_dev0_avdd              = "avdd-csi"
csi0_dev0_avdd_vol          = 2800000
csi0_dev0_dvdd              = "dvdd-csi-18"
csi0_dev0_dvdd_vol          = 1500000
csi0_dev0_afvdd             = "afvcc-csi"
csi0_dev0_afvdd_vol         = 2800000
;csi0_dev0_power_en         = port:PB04<1><0><1><0>
csi0_dev0_power_en          =
csi0_dev0_reset             = port:PE14<1><0><1><0>
csi0_dev0_pwdn              = port:PE15<1><0><1><0>
csi0_dev0_flash_used        = 0
csi0_dev0_flash_type        = 2
csi0_dev0_flash_en          =
csi0_dev0_flash_mode        =
csi0_dev0_flvdd             = ""
csi0_dev0_flvdd_vol         =
csi0_dev0_af_pwdn           =
csi0_dev0_act_used          = 0
csi0_dev0_act_name          = "ad5820_act"
csi0_dev0_act_slave         = 0x18

[csi0/csi0_dev1]
csi0_dev1_used              = 0
csi0_dev1_mname             = ""
```

```
csi0_dev1_twi_addr            = 0x78
csi0_dev1_pos              = "rear"
csi0_dev1_isp_used          = 1
csi0_dev1_fmt                = 0
csi0_dev1_stby_mode         = 0
csi0_dev1_vflip             = 0
csi0_dev1_hflip             = 0
csi0_dev1_iovdd              = "iovdd-csi"
csi0_dev1_iovdd_vol         = 2800000
csi0_dev1_avdd               = "avdd-csi"
csi0_dev1_avdd_vol          = 2800000
csi0_dev1_dvdd               = "dvdd-csi-18"
csi0_dev1_dvdd_vol          = 1500000
csi0_dev1_afvdd              = "afvcc-csi"
csi0_dev1_afvdd_vol         = 2800000
csi0_dev1_power_en           =
csi0_dev1_reset             =
csi0_dev1_pwdn                =
csi0_dev1_flash_used        = 0
csi0_dev1_flash_type        = 2
csi0_dev1_flash_en          =
csi0_dev1_flash_mode         =
csi0_dev1_flvdd                = ""
csi0_dev1_flvdd_vol             =
csi0_dev1_af_pwdn             =
csi0_dev1_act_used           = 0
csi0_dev1_act_name            = "ad5820_act"
csi0_dev1_act_slave         = 0x18
```


注意：A64 平板上添加了对 device_tree 的支持，sys_config.fex 在打包 pack 的时候会被转换生 device_tree，驱动获取配置也是从 device_tree 获取的，所以不支持以前在线改 sysconfig 的配置然后重启生效的功能。

2.4.2.2. sensor_list_cfg.ini 配置

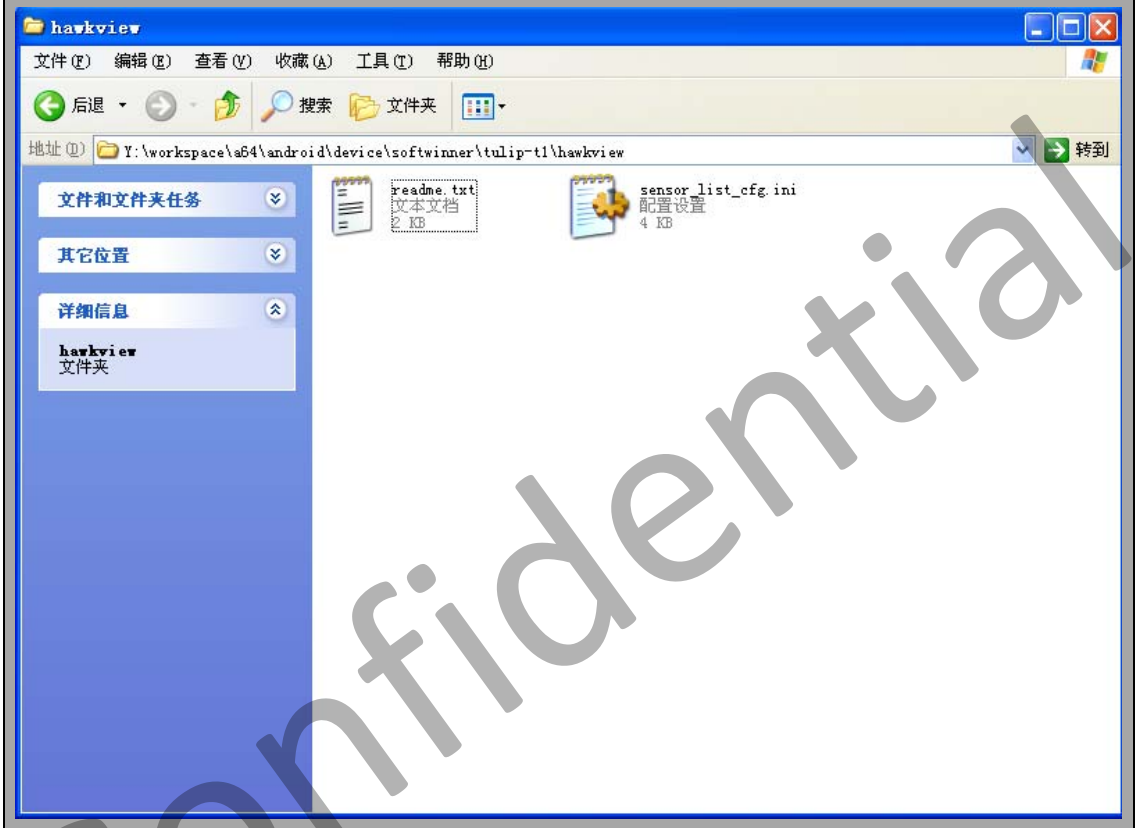
A64 方案继承了之前 A80 方案上自动检测 camera 的功能，该功能支持在同一个方案上采用不同的模组组合。如果需要使用该方案，需要在 sys_config.fex 中作出相应的配置：

1. 设置相应 csi 上的相关选项 csi0_sensor_list = 1。
2. 明确定义出前后摄像头，例如 csi0_dev0_pos = "rear" ，csi0_dev1_pos = "front"；

如果定义了 csi0_sensor_list = 1，则驱动就会去试图读取/system/etc/hawkview/sensor_list_cfg.ini，如果读取成功，则驱动会用 sensor_list_cfg.ini 中的相应信息替换掉原来从 sys_config.fex 中读取的信息，如果读取失败，则驱动会继续使用 sys_config.fex 中的配置。

与 hawkview 的配置相似， sensor_list_cfg.ini 配置文件也放在如下目录中：

android\device\softwinner\tulip-t1\hawkview



在 Android\device\softwinner\tulip-xxx\tulip_xxx.mk 文件中按如下方法增加配置:

```
# camera config for camera detector
PRODUCT_COPY_FILES += \
    device/softwinner/tulip-t1/hawkview/sensor_list_cfg.ini:system/etc/hawkview/sensor_list_cfg.ini
```

下面结合 sensor_list_cfg.ini 配置文件说明 camera detector 功能该如何使用：

1. sensor_list_cfg.ini 中整体上分为前置和后置两套 camera 配置。
2. 每套 camera 的配置分为 bus configs, power configs 和 sensor configs：
   a) **Bus configs：**考虑到客户已经习惯在 sysconfig 中配置相关的 bus，在这里暂不配置。
   b) **Power configs：**该部分可以根据客户或者开发人员需要，通过 power_settings_enable 来选择使用 sysconfig 中配置还是 sensor_list_cfg.ini 中的配置，例如 power_settings_enable = 0：代表使用 sysconfig 中配置，power_settings_enable = 1

代表使用 sensor_list_cfg.ini 中配置。

c) **Sensor configs：**考虑到检测速度等方面原因，对前置和后置最大检测数量做出了限制，最大都为 3。

d) 各个 sensor 实体配置比较灵活，可以 YUV sensor 也可以是 RAW sensor，也可以独立配置各自的 hflip 和 vflip。对于 RAW sensor 也可以独立配置 VCM。

3. 目前驱动不支持对供电电压要求不同的 sensor 列表做自动检测

4. 驱动也不能检测出相同的 sensor 使用不同的 VCM 的情况。

下面给出一个具体的使用例子，该例子后置使用 gc0325，gc2155，ov5640；前置使用 gc0328，gc0308，gc0328c。

```
#A80 sensor list configs
#
####################bus configs######################
#
#used: 0: not used, 1: used;                                    //暂时无需配置，使用 sysconfig 的配置
#csi_sel: 0: mipi, 1:    parallel ;                             //暂时无需配置，使用 sysconfig 的配置
#device_sel: 0: dev0, 1: dev1;                                  //暂时无需配置，使用 sysconfig 的配置
#sensor_twi_id: twi id, for example: sensor_twi_id = 0 //暂时无需配置，使用 sysconfig 的配置
#
####################power configs####################
#power_settings_enable: 0: enable the power settings in    sysconfig.fex;    1: enable the power
settings in this file.
#
#iovdd: The name of iovdd for this camera;
#iovdd_vol: The voltage value of iovdd in uV;
#

################detect sensor configs#################
#
#detect_sensor_num: The number of sensors need be detected in this bus.
#sensor_name[x]: The sensor name in sensor driver.
#sensor_twi_addr[x]: The i2c address of this sensor.
#sensor_type[x]: The sensor type, 0: YUV, 1: RAW;
#sensor_stby_mode[x]: Not used;
#sensor_hflip[x]: Horizontal flip;
#sensor_vflip[x]: Vertical    flip;
#act_name[x]: The VCM name in vcm driver, only RAW sensor need be configured;
#act_twi_addr[x]: The VCM i2c address;
#
###################################################
[rear_camera_cfg]
```

```
#bus configs
used                    = 1
csi_sel                 = 1
device_sel              = 0
sensor_twi_id           = 2

#power configs
power_settings_enable   = 1
iovdd                   = "iovdd-csi"
iovdd_vol               = 2800000
avdd                    = "avdd-csi"
avdd_vol                = 2800000
dvdd                    = "dvdd-csi-18"
dvdd_vol                = 1800000
afvdd                   = ""
afvdd_vol               =

#detect sensor configs
detect_sensor_num       = 3

sensor_name0            = "gc2035"
sensor_twi_addr0        = 0x78
sensor_type0            = 0
sensor_stby_mode0       = 0
sensor_hflip0           = 0
sensor_vflip0           = 0
act_name0               =
act_twi_addr0           =

sensor_name1            = "gc2155"
sensor_twi_addr1        = 0x78
sensor_type1            = 0
sensor_stby_mode1       = 0
sensor_hflip1           = 0
sensor_vflip1           = 0
act_name1               =
act_twi_addr1           =

sensor_name2            = "ov5640"
sensor_twi_addr2        = 0x78
sensor_type2            = 0
sensor_stby_mode2       = 0
sensor_hflip2           = 0
```

```
sensor_vflip2                = 0
act_name2                     =
act_twi_addr2                 =

[front_camera_cfg]

#bus configs
used                          = 1
csi_sel                      = 1
device_sel                   = 0
sensor_twi_id                = 2

#power configs

power_settings_enable      = 1
iovdd                         = "iovdd-csi"
iovdd_vol                    = 2800000
avdd                          = "avdd-csi"
avdd_vol                     = 2800000
dvdd                          = "dvdd-csi-18"
dvdd_vol                     = 1800000
afvdd                         = ""
afvdd_vol                    =

#detect sensor configs
detect_sensor_num          = 3

sensor_name0                 = "gc0328"
sensor_twi_addr0           = 0x42
sensor_type0                = 0
sensor_stby_mode0          = 0
sensor_hflip0               = 0
sensor_vflip0               = 0
act_name0                     =
act_twi_addr0                 =

sensor_name1                 = "gc0308"
sensor_twi_addr1           = 0x42
sensor_type1                = 0
sensor_stby_mode1          = 0
sensor_hflip1               = 0
sensor_vflip1               = 0
act_name1                     =
act_twi_addr1                 =
```

```
sensor_name2                  = "gc0328c"
sensor_twi_addr2              = 0x42
sensor_type2                  = 0
sensor_stby_mode2             = 0
sensor_hflip2                 = 0
sensor_vflip2                 = 0
act_name2                     =
act_twi_addr2                 =
```

2.4.2.3. CCI_Client 调试节点使用方法

VFE 驱动加载成功后，会为每个 sensor 或者 vcm 设置注册个设备节点，以 ov5640 为例会出现如下目录：/sys/devices/ov5640

该目录下有 5 个节点属性，其名字和代表意义如下：

a. addr_width：

地址位宽，16 进制，例如 echo 10 > addr_width，即设置 addr_width 为 0x10（也就是 16）

b. data_width

数据位宽，16 进制，例如 echo 10 > data_width，即设置 addr_width 为 0x10（也就是 16）

c. read_flag

读写标志，16 进制，例如 echo 1 > read_flag，即设置为读模式；

echo 0 > read_flag，设置为写模式。

d. read_value

读取的 sensor 寄存器值，16 进制，其为只读属性。通过命令 cat read_value 获取当前值。

e. cci_client

CCI 客户端，16 进制，cat cci_client 可以看实例，且会打印出当前 cci 读写状态。

例如执行 echo 30350031 > cci_client 即，向 0x3035 地址写入 0x0031，无论具体设备地址和数据位宽是多少，此命令一律格式化为 16bits。

例如要改变 ov5640 帧率可以执行如下命令：

cd /sys/devices/ov5640

echo 10 > addr_width //地址位宽为 16

echo 8 > data_width //数据位宽为 8

echo 0 > read_flag //设置为写状态

echo 30350021 > cci_client　　//向 0x3035 写入 0x21

注意：每个 sensor 的 addr_width 以及 data_width 只需设置一次（以后会在设备注册时候得到此信息，就不用写这两项了）。

3. 模块体系结构描述

这里仅对新版本驱动作介绍：

1、 使用过程中可简单的看成是 vfe 模块+ device 模块 +af driver + flash 控制模块的方式；

2、 Vfe.c 是驱动的主要功能实现，包括注册/注销、参数读取、与 v4l2 上层接口、与各 device 的下层接口、中断处理、buffer 申请切换等；

3、 Device 文件夹里面是各个 sensor 的器件层实现，一般包括上下电、初始化，各分辨率切换，YUV sensor 包括绝大部分的 v4l2 定义的 io-ctrl 命令的实现；而 RAW sensor 的话大部分 io-ctrl 命令在 vfe 层调用 isp 的库实现，少数如曝光/增益调节会透过 vfe 层到实际器件层；

4、 Actuator 文件夹内是各种 vcm 的驱动；

5、 Flash_light 文件夹内是闪光灯控制接口实现；

6、 Csi 和 mipi_csi 为对 csi 接口和 mipi 接口的控制文件；

7、 lib 文件夹为 isp 的库文件；

对于 YUV sensor 的调试，驱动结构与旧版本的差别不大，主要是一些参数和命令较旧版本有修改，带 AF 的模组调试可参考 ov5640.c；不带 AF 功能的模组调试参考 gc0308.c 等，替换各个接口和参数配置。详细过程参考下面。

4. 模块开发 demo

这里以 YUV sensor ov5640.c 为例，介绍 sensor 驱动 device 模块的实现，抽象控制的 vfe.c 一般不能修改，也不做介绍。

4.1. 硬件部分

检查硬件电源，io 是否在原理图正确连接，并且在 sys_config.fex 中正确配置，包括使用的电源名称和电压，sys_config.fex 配置详见前面说明；如果是电源选择有多个源头的请确认板子上的连接正确，比如 0ohm 电阻是否正确的焊接为 0ohm，NC 的电阻是否有正确断开等等。带补光灯的也需要检查灯和 driver IC 和控制 io 是否连好。

4.2. 内核 device 模块驱动

一般调试新模组的话建议以 sdk 中的某个现成的驱动为基础修改：

YUV 的带 AF 功能并口模组以 ov5640.c 为参考；

YUV 的无 AF 功能并口模组以 gc0308.c 为参考；

4.2.1. 添加器件驱动范例

4.2.1.1. 添加 makefile

\linux-3.10\drivers\media\platform\sunxi-vfe \device\Makefile 添加：obj-m += ov5640.o

4.2.1.2. 配置模组的参数

```
#define MCLK                    (24*1000*1000)
#define VREF_POL                V4L2_MBUS_VSYNC_ACTIVE_HIGH
#define HREF_POL                V4L2_MBUS_HSYNC_ACTIVE_HIGH
#define CLK_POL                 V4L2_MBUS_PCLK_SAMPLE_RISING
#define V4L2_IDENT_SENSOR 0x5640

#define CSI_STBY_ON         1
#define CSI_STBY_OFF        0
#define CSI_RST_ON          0
#define CSI_RST_OFF         1
#define CSI_PWR_ON          1
#define CSI_PWR_OFF         0
#define CSI_AF_PWR_ON       1
#define CSI_AF_PWR_OFF      0
…
#define regval_list reg_list_a16_d8     //注意不同 sensor 的数据位宽
/*
```

如果是用到美光的一些 sensor 需要混合写 16 和 8bit 的数据，可使用
struct regval_list_w_a16_d16 {
  unsigned short width;
  unsigned short addr;
  unsigned short data;
};
或者其他自定义的结构体，用于区分不同位宽的 i2c 读写
*/
…
#define I2C_ADDR 0x78 //填写对应 ID

4.2.1.3. 填写 sensor 的配置

sensor_default_regs[]
sensor_qsxga_regs[]
…..

sensor_sxga_regs[]等填写对应分辨率的配置，要求每个模式之间抽离干净，公共部分都放到 default_regs 里面，模式切换一般先是 software standby on，然后配置 clock，size，binning，yuv sensor 的话可能还要填写 banding step，如果跑不同 pclk 时候对信号质量有要求也需要加上调节驱动能力的参数，设置完之后再 software stanby off；

关于 size 的配置要求如下：

1、5M 的 sensor 要求 vga@30fps/720p@30fps/1080p@15/30fps 的配置供录像使用，5M 的 fullsize 为 2592x1936 的分辨率供拍照（yuv 的 sensor 一般要求 5~10fps，拍照固定或可变帧率均可）。素质较好的 5M 应该补充 SXGA(1280x960)的分辨率用来保证拍照预览的清晰度。

2、1M/2M/3M 一般素质相差较大，分辨率需要 fullsize，以及 720p/VGA，帧率越高越好。

3、VGA，只要 VGA，帧率越高越好。

sensor_af_fw_regs[]为不同模组厂使用不同 driver 时的 af 固件，数据量一般较大
/*如 OV5640 的 af 固件是独立于其他配置数组存放的，在其 PWDN IO 控制其进入 standby 之后需要重新下载 af 固件，时间也消耗比较多；也有些 sensor 的 af 固件是固化在 sensor 内部的，外部只需要设置少量参数即可，如 s5k4ec/mt9p111 等*/
sensor_wb_xxx
sensor_colorfx_xxx 等效果的配置
sensor_fmt_yuv422_yuyv[]等 yuv sequence 的配置，必须实现。

4.2.1.4. i2c 接口的读写函数

注意位宽，如果有混合不同位宽的读写还要再加函数区分。

```
static int sensor_read(struct v4l2_subdev *sd, unsigned short reg,
        unsigned char *value)
{
        …
        …cci_read_a16_d8(sd,reg,value);
        …
}
static int sensor_write(struct v4l2_subdev *sd, unsigned short reg,
        unsigned char value)
{
        …
        …cci_write_a16_d8(sd,reg,value);
        …
}
```

4.2.1.5. v4l2 命令接口

sensor_queryctrl / sensor_g_ctrl / sensor_s_ctrl 是实现 sensor 的 v4l2 命令的部分：

i. queryctrl 用于控制 g/s_ctrl 里面各个命令的参数范围；
ii. 2. g_ctrl 是获取 sensor 参数的接口，一般是后面会调用更细致的函数；
iii. 3. s_ctrl 是设置 sensor 参数的接口，一般是后面会调用更细致的函数；

对于 YUV 的 sensor 来说，以 ov5640 这样的带 af 功能的模组来说需要实现以下接口（这些命令一般是配合 HAL 调用的，所以这部分接口对于所有 YUV sensor 都是同样的，需要修改的仅仅是里面调用的具体函数，黄色部分的在无 af 功能的 sensor 驱动中是不需要实现的，必须删除， 如果是 5M 的 sensor 大多数是可能同时存在带 AF 和不带 AF 功能两种模组，这样的情况下，如果使用不带 AF 的模组，则需要在 camera.cfg 中关闭 AF 功能，驱动中仍然保留 AF 功能的相关接口）。

```
static int sensor_queryctrl(struct v4l2_subdev *sd,
        struct v4l2_queryctrl *qc)
{
    /* Fill in min, max, step and default value for these controls. */
    /* see include/linux/videodev2.h for details */
//    vfe_dev_dbg("queryctrl qc->id=0x%8x\n", qc->id);
    switch (qc->id) {
//    case V4L2_CID_BRIGHTNESS:
//        return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
//    case V4L2_CID_CONTRAST:
//        return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
//    case V4L2_CID_SATURATION:
```

```
//        return v4l2_ctrl_query_fill(qc, -4, 4, 1, 1);
//    case V4L2_CID_HUE:
//        return v4l2_ctrl_query_fill(qc, -180, 180, 5, 0);
    case V4L2_CID_VFLIP:
    case V4L2_CID_HFLIP:
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
//    case V4L2_CID_GAIN:
//        return v4l2_ctrl_query_fill(qc, 0, 255, 1, 128);
//    case V4L2_CID_AUTOGAIN:
//        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 1);
    case V4L2_CID_EXPOSURE:
    case V4L2_CID_AUTO_EXPOSURE_BIAS:
        return v4l2_ctrl_query_fill(qc, -4, 4, 1, 0);
    case V4L2_CID_EXPOSURE_AUTO:
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
    case V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE:
        return v4l2_ctrl_query_fill(qc, 0, 9, 1, 1);
    case V4L2_CID_AUTO_WHITE_BALANCE:
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 1);
    case V4L2_CID_COLORFX:
        return v4l2_ctrl_query_fill(qc, 0, 15, 1, 0);
    case V4L2_CID_FLASH_LED_MODE:
        return v4l2_ctrl_query_fill(qc, 0, 4, 1, 0);

    case V4L2_CID_3A_LOCK:
        return v4l2_ctrl_query_fill(qc, 0, V4L2_LOCK_FOCUS, 1, 0);
//    case V4L2_CID_AUTO_FOCUS_RANGE:
//        return v4l2_ctrl_query_fill(qc, 0, 0, 0, 0);//only auto
    case V4L2_CID_AUTO_FOCUS_INIT:
    case V4L2_CID_AUTO_FOCUS_RELEASE:
    case V4L2_CID_AUTO_FOCUS_START:
    case V4L2_CID_AUTO_FOCUS_STOP:
    case V4L2_CID_AUTO_FOCUS_STATUS:
        return v4l2_ctrl_query_fill(qc, 0, 0, 0, 0);
    case V4L2_CID_FOCUS_AUTO:
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
    case V4L2_CID_AUTO_EXPOSURE_WIN_NUM:
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
    case V4L2_CID_AUTO_FOCUS_WIN_NUM:
```

```
        return v4l2_ctrl_query_fill(qc, 0, 1, 1, 0);
    }
    return -EINVAL;
}

static int sensor_g_ctrl(struct v4l2_subdev *sd, struct v4l2_control *ctrl)
{
    //vfe_dev_dbg("sensor_g_ctrl ctrl->id=0x%8x\n", ctrl->id);
    switch (ctrl->id) {
    case V4L2_CID_BRIGHTNESS:
        return sensor_g_brightness(sd, &ctrl->value);
    case V4L2_CID_CONTRAST:
        return sensor_g_contrast(sd, &ctrl->value);
    case V4L2_CID_SATURATION:
        return sensor_g_saturation(sd, &ctrl->value);
    case V4L2_CID_HUE:
        return sensor_g_hue(sd, &ctrl->value);
    case V4L2_CID_VFLIP:
        return sensor_g_vflip(sd, &ctrl->value);
    case V4L2_CID_HFLIP:
        return sensor_g_hflip(sd, &ctrl->value);
    case V4L2_CID_GAIN:
        return sensor_g_gain(sd, &ctrl->value);
    case V4L2_CID_AUTOGAIN:
        return sensor_g_autogain(sd, &ctrl->value);
    case V4L2_CID_EXPOSURE:
    case V4L2_CID_AUTO_EXPOSURE_BIAS:
        return sensor_g_exp_bias(sd, &ctrl->value);
    case V4L2_CID_EXPOSURE_AUTO:
        return sensor_g_autoexp(sd, &ctrl->value);
    case V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE:
        return sensor_g_wb(sd, &ctrl->value);
    case V4L2_CID_AUTO_WHITE_BALANCE:
        return sensor_g_autowb(sd, &ctrl->value);
    case V4L2_CID_COLORFX:
        return sensor_g_colorfx(sd, &ctrl->value);
    case V4L2_CID_FLASH_LED_MODE:
        return sensor_g_flash_mode(sd, &ctrl->value);
    case V4L2_CID_POWER_LINE_FREQUENCY:
```

```
        return sensor_g_band_filter(sd, &ctrl->value);

    case V4L2_CID_3A_LOCK:
        return sensor_g_3a_lock(sd);
//    case V4L2_CID_AUTO_FOCUS_RANGE:
//        ctrl->value=0;//only auto
//        return 0;
//    case V4L2_CID_AUTO_FOCUS_INIT:
//    case V4L2_CID_AUTO_FOCUS_RELEASE:
//    case V4L2_CID_AUTO_FOCUS_START:
//    case V4L2_CID_AUTO_FOCUS_STOP:
    case V4L2_CID_AUTO_FOCUS_STATUS:
        return sensor_g_af_status(sd);
//    case V4L2_CID_FOCUS_AUTO:
    case V4L2_CID_AUTO_FOCUS_WIN_NUM:
        ctrl->value=1;
        return 0;
    case V4L2_CID_AUTO_EXPOSURE_WIN_NUM:
        ctrl->value=1;
        return 0;
    }
    return -EINVAL;
}

static int sensor_s_ctrl(struct v4l2_subdev *sd, struct v4l2_control *ctrl)
{
    struct v4l2_queryctrl qc;
    int ret;

//    vfe_dev_dbg("sensor_s_ctrl ctrl->id=0x%8x\n", ctrl->id);
    qc.id = ctrl->id;
    ret = sensor_queryctrl(sd, &qc);
    if (ret < 0) {
        return ret;
    }

        if (qc.type == V4L2_CTRL_TYPE_MENU ||
            qc.type == V4L2_CTRL_TYPE_INTEGER ||
            qc.type == V4L2_CTRL_TYPE_BOOLEAN)
```

```
    {
        if (ctrl->value < qc.minimum || ctrl->value > qc.maximum) {
            return -ERANGE;
        }
    }

switch (ctrl->id) {
    case V4L2_CID_BRIGHTNESS:
        return sensor_s_brightness(sd, ctrl->value);
    case V4L2_CID_CONTRAST:
        return sensor_s_contrast(sd, ctrl->value);
    case V4L2_CID_SATURATION:
        return sensor_s_saturation(sd, ctrl->value);
    case V4L2_CID_HUE:
        return sensor_s_hue(sd, ctrl->value);
    case V4L2_CID_VFLIP:
        return sensor_s_vflip(sd, ctrl->value);
    case V4L2_CID_HFLIP:
        return sensor_s_hflip(sd, ctrl->value);
    case V4L2_CID_GAIN:
        return sensor_s_gain(sd, ctrl->value);
    case V4L2_CID_AUTOGAIN:
        return sensor_s_autogain(sd, ctrl->value);
    case V4L2_CID_EXPOSURE:
    case V4L2_CID_AUTO_EXPOSURE_BIAS:
        return sensor_s_exp_bias(sd, ctrl->value);
    case V4L2_CID_EXPOSURE_AUTO:
        return sensor_s_autoexp(sd,
                (enum v4l2_exposure_auto_type) ctrl->value);
    case V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE:
            return sensor_s_wb(sd,
                (enum v4l2_auto_n_preset_white_balance) ctrl->value);
    case V4L2_CID_AUTO_WHITE_BALANCE:
        return sensor_s_autowb(sd, ctrl->value);
    case V4L2_CID_COLORFX:
        return sensor_s_colorfx(sd,
                (enum v4l2_colorfx) ctrl->value);
    case V4L2_CID_FLASH_LED_MODE:
        return sensor_s_flash_mode(sd,
```

```
                (enum v4l2_flash_led_mode) ctrl->value);
        case V4L2_CID_POWER_LINE_FREQUENCY:
            return sensor_s_band_filter(sd,
                (enum v4l2_power_line_frequency) ctrl->value);

        case V4L2_CID_3A_LOCK:
            return sensor_s_3a_lock(sd, ctrl->value);
//        case V4L2_CID_AUTO_FOCUS_RANGE:
//          return 0;
            case V4L2_CID_AUTO_FOCUS_INIT:
              return sensor_s_init_af(sd);
            case V4L2_CID_AUTO_FOCUS_RELEASE:
              return sensor_s_release_af(sd);
            case V4L2_CID_AUTO_FOCUS_START:
              return sensor_s_single_af(sd);
            case V4L2_CID_AUTO_FOCUS_STOP:
              return sensor_s_pause_af(sd);
        //    case V4L2_CID_AUTO_FOCUS_STATUS:
            case V4L2_CID_FOCUS_AUTO:
              return sensor_s_continueous_af(sd, ctrl->value);
            case V4L2_CID_AUTO_FOCUS_WIN_NUM:
              vfe_dev_dbg("s_ctrl win value=%d\n",ctrl->value);
              return sensor_s_af_zone(sd, (struct v4l2_win_coordinate *)(ctrl->user_pt));
            case V4L2_CID_AUTO_EXPOSURE_WIN_NUM:
              return 0;
    }
    return -EINVAL;
}
```

4.2.1.6. AF 控制

不同 YUV sensor 的 af 部分是大同小异的，需要对照 sensor 的 datasheet 和 application note 来实现以下函数（RAW sensor 的话由主控 ISP 的算法控制，命令在 vfe.c 接收，无需 device 文件控制）：

sensor_s_3a_lock（yuv sensor 需要实现至少 af lock，使能 HAL 能在连续自动对焦时候 halt 住，停在当前位置，awb lock 和 ae lock 可以不实现，由驱动自行控制）

sensor_s_init_af

sensor_s_release_af

sensor_s_single_af

sensor_s_pause_af（一般是 af lock 的具体实现，驱动内部会调用）
sensor_s_continueous_af
sensor_s_af_zone
sensor_g_af_status

在 YUV 模组的驱动中，对焦过程看到 HAL 发过来的命令是这样的：

i. HAL 在进入 camera 应用之后先发送 V4L2_CID_AUTO_FOCUS_INIT，让 sensor 下载对应 af 固件才能；

ii. 如果 sensor 自带连续自动对焦功能，需要在 sensor_s_fmt 中区分 VIDEO 和 CAPTURE 模式，在 VIDEO 模式时候判断上层是否有使能该功能（会通过发 V4L2_CID_FOCUS_AUTO（参数 0/1）命令来通知驱动使用该功能与否，这个状态会保存在 info->auto_focus 参数中），有的话保证在切换至 VIDEO 模式时候开启此功能；这样即使不点触屏幕，在预览画面中也是有自动对焦功能的。

iii. 如果 sensor 支持点触对焦功能，那么 HAL 会通过 V4L2_CID_AUTO_FOCUS_WIN_NUM 发送坐标给驱动，注意传下来的坐标是以当前画面的中心为(0,0)，然后屏幕的上下左右为-1000 至+1000 的范围，需要将此坐标系在驱动层 sensor_s_af_zone 里面转换成 sensor 需要的坐标系，需要查看对应 application note。

iv. 设置坐标之后，HAL 立刻发送 V4L2_CID_AUTO_FOCUS_START 命令下来，这时需要调用 sensor_s_single_af 实现具体寄存器的操作，同时注意 info->auto_focus 和 info->focus_status 的状态变化；

v. 单次对焦命令发送之后，HAL 会反复查询对焦状态，通过 sensor_g_ctrl 里面的 V4L2_CID_AUTO_FOCUS_STATUS 命令来获取，对应 sensor_g_af_status 实现寄存器的读取，里面如果支持连续自动对焦下的状态查询和单次对焦状态查询的话需要分别实现不同函数。单次对焦必须实现状态查询，连续自动对焦如果不支持查询的话可以直接返回 V4L2_AUTO_FOCUS_STATUS_REACHED；

vi. 在拍照之前，或者是录像的时候，HAL 根据不同情况可能会发送 V4L2_CID_3A_LOCK 命令下来，一般这里只去实现 af lock 的功能，awb 和 ae 的话不通过这个命令来锁，而是根据实际 sensor 的 preview 和 capture 切换的具体需要（并不是所有 sensor 都适合在切换时候锁定 ae 和 awb），由驱动在 sensor_s_fmt 中来控制是否锁定，所以在 queryctrl 中 case V4L2_CID_3A_LOCK: 这一项，写的是 return v4l2_ctrl_query_fill(qc, 0, V4L2_LOCK_FOCUS, 1, 0);

4.2.1.7. Sensor_s_fmt 模式切换

不同分辨率，以及预览和拍照切换。YUV sensor 一般情况下，5M 的 sensor 都是通过获取帧率，曝光行数，计算特定帧率下拍照达到和预览相同亮度时候的曝光行，在手动曝光模式下切换至 fullsize 进行拍照，再切换回预览（这个地方必须严格参考 datasheet 和 application note）， 2M 的 sensor 需要视具体 sensor 而定，有部分帧率高的 sensor（如 ov2643 等）可以直接用 2M 预览，使用其 AE 即可拍照，无需切换。0.3M 的一般不需要切换。这个过程在 A10/A13/A20/A31/A31s/A80/A83/A64 上面对于 YUV sensor 都是一样的。

4.2.1.8. 其他特效相关寄存器配置

其他 awb/exp/colorfx/brightness/bandfilter 等功能的实现，参考 datasheet 和 application note。

4.2.1.9. Sensor_power 上电下电过程的控制

这部分参考 datasheet 和 application note 给出的时序图

static int sensor_power(struct v4l2_subdev *sd, int on)。

CSI_SUBDEV_STBY_ON 时候：有软件 standby 时候去设置寄存器，有 stanby_io 控制的也控制 io，都没有的关闭 sensor 的 io（设置成高阻态，一般 standby 时候不要用 io reset sensor，这样很有可能在两个模组的 reset 脚共用一个 io 时候影响到另外一个 sensor，可能引起 i2c 访问失败，无法使用 camera 功能），最后关闭 MCLK。

CSI_SUBDEV_STBY_OFF 时候：设置 MCLK 频率，使能 MCLK 输出（因为同一个口上可能接两个不同的模组，两个模组需要的 MCLK 频率是有可能不同的）；io 设置成工作模式，有 sensor io 设置成高阻的重新使能，稍微延时后返回；

CSI_SUBDEV_PWR_ON 时候：是一个从完全没有电开始的状态，首先是将各个控制 io 设置成为输出模式，然后在设置其为合适的输出状态（查看 datasheet），打开电源，再控制 io 成工作状态，有 io reset 的话也复位一次 sensor；

CSI_SUBDEV_PWR_OFF 时候：stanby io 拉成 stanby 状态，reset io 拉成复位状态，关闭 MCLK，关闭电源，控制 io 回到高阻态。

```
switch(on)
{
  case CSI_SUBDEV_STBY_ON:
    vfe_dev_dbg("CSI_SUBDEV_STBY_ON!\n");
    #ifdef _FLASH_FUNC_
    io_set_flash_ctrl(sd, SW_CTRL_FLASH_OFF, to_state(sd)->fl_dev_info);
```

```
        #endif
        //software standby
        ret = sensor_write_array(sd, sensor_sw_stby_on_regs, ARRAY_SIZE(sensor_sw_stby_on_regs));
        if(ret < 0)
            vfe_dev_err("soft stby falied!\n");
        usleep_range(10000, 12000);
        //disable io oe
        vfe_dev_print("disalbe oe!\n");
        ret = sensor_write_array(sd, sensor_oe_disable_regs, ARRAY_SIZE(sensor_oe_disable_regs));
        if(ret < 0)
            vfe_dev_err("disalbe oe falied!\n");
        //make sure that no device can access i2c bus during sensor initial or power down
        //when using i2c_lock_adpater function, the following codes must not access i2c bus before calling i2c_unlock_adapter
        i2c_lock_adapter(client->adapter);
        //standby on io
        vfe_gpio_write(sd,PWDN,CSI_STBY_ON);
        //remember to unlock i2c adapter, so the device can access the i2c bus again
        i2c_unlock_adapter(client->adapter);
        //inactive mclk after stadby in
        vfe_set_mclk(sd,OFF);
        break;
    case CSI_SUBDEV_STBY_OFF:
        vfe_dev_dbg("CSI_SUBDEV_STBY_OFF!\n");
        //make sure that no device can access i2c bus during sensor initial or power down
        //when using i2c_lock_adpater function, the following codes must not access i2c bus before calling i2c_unlock_adapter
        i2c_lock_adapter(client->adapter);
        //active mclk before stadby out
        vfe_set_mclk_freq(sd,MCLK);
        vfe_set_mclk(sd,ON);
        usleep_range(10000, 12000);
        //standby off io
        vfe_gpio_write(sd,PWDN,CSI_STBY_OFF);
        usleep_range(10000, 12000);
        //remember to unlock i2c adapter, so the device can access the i2c bus again
        i2c_unlock_adapter(client->adapter);
        vfe_dev_print("enable oe!\n");
        ret = sensor_write_array(sd, sensor_oe_enable_regs,  ARRAY_SIZE(sensor_oe_enable_regs));
        if(ret < 0)
```

```
            vfe_dev_err("enable oe falied!\n");
        //software standby
        ret = sensor_write_array(sd, sensor_sw_stby_off_regs ,ARRAY_SIZE(sensor_sw_stby_off_regs));
        if(ret < 0)
            vfe_dev_err("soft stby off falied!\n");
        usleep_range(10000, 12000);
        break;
    case CSI_SUBDEV_PWR_ON:
        vfe_dev_dbg("CSI_SUBDEV_PWR_ON!\n");
        //make sure that no device can access i2c bus during sensor initial or power down
        //when using i2c_lock_adpater function, the following codes must not access i2c bus before calling
i2c_unlock_adapter
        i2c_lock_adapter(client->adapter);
        //power on reset
        vfe_gpio_set_status(sd,PWDN,1);//set the gpio to output
        vfe_gpio_set_status(sd,RESET,1);//set the gpio to output
        //power down io
        vfe_gpio_write(sd,PWDN,CSI_STBY_ON);
        //reset on io
        vfe_gpio_write(sd,RESET,CSI_RST_ON);
        usleep_range(1000, 2000);
        //active mclk before power on
        vfe_set_mclk_freq(sd,MCLK);
        vfe_set_mclk(sd,ON);
        usleep_range(10000, 12000);
        //power supply
        vfe_gpio_write(sd,POWER_EN,CSI_PWR_ON);
        vfe_set_pmu_channel(sd,IOVDD,ON);
        vfe_set_pmu_channel(sd,AVDD,ON);
        vfe_set_pmu_channel(sd,DVDD,ON);
        vfe_set_pmu_channel(sd,AFVDD,ON);
        //standby off io
        vfe_gpio_write(sd,PWDN,CSI_STBY_OFF);
        usleep_range(10000, 12000);
        //reset after power on
        vfe_gpio_write(sd,RESET,CSI_RST_OFF);
        usleep_range(30000, 35000);
        //remember to unlock i2c adapter, so the device can access the i2c bus again
        i2c_unlock_adapter(client->adapter);
```

```
        break;
    case CSI_SUBDEV_PWR_OFF:
        vfe_dev_dbg("CSI_SUBDEV_PWR_OFF!\n");
        //make sure that no device can access i2c bus during sensor initial or power down
        //when using i2c_lock_adpater function, the following codes must not access i2c bus before calling i2c_unlock_adapter
        i2c_lock_adapter(client->adapter);
        //inactive mclk before power off
        vfe_set_mclk(sd,OFF);
        //power supply off
        vfe_gpio_write(sd,POWER_EN,CSI_PWR_OFF);
        vfe_set_pmu_channel(sd,AFVDD,OFF);
        vfe_set_pmu_channel(sd,DVDD,OFF);
        vfe_set_pmu_channel(sd,AVDD,OFF);
        vfe_set_pmu_channel(sd,IOVDD,OFF);
        //standby and reset io
        usleep_range(10000, 12000);
        vfe_gpio_write(sd,POWER_EN,CSI_STBY_OFF);
        vfe_gpio_write(sd,RESET,CSI_RST_ON);
        //set the io to hi-z
        vfe_gpio_set_status(sd,RESET,0);//set the gpio to input
        vfe_gpio_set_status(sd,PWDN,0);//set the gpio to input
        //remember to unlock i2c adapter, so the device can access the i2c bus again
        i2c_unlock_adapter(client->adapter);
        break;
    default:
        return -EINVAL;
    }
```

4.2.1.10. 填写 sensor 检测 ID

static int sensor_detect(struct v4l2_subdev *sd)

填入恰当的寄存器和数值。

4.2.1.11. 填写不同分辨率寄存器配置和对应数组结构

static struct sensor_win_size sensor_win_sizes[]

对于 sensor 输出信号非实际需要的 size 时候，hoffset 和 voffset 可自行调整，达到 crop 的效果；**请软件保证对应配置下 sensor 输出的 hsize 和 vsize 在减去 hoffset 和 voffset 之后**

大于等于实际接收的 size。

4.2.1.12. 切换不同 size

static int sensor_s_fmt(struct v4l2_subdev *sd, struct v4l2_mbus_framefmt *fmt)

过程一般是 try_fmt 后写对应的 sensor_win_sizes 配置，YUV sensor 的话还需要根据 VIDEO_MODE 和 CAPTURE_MODE 来区分 3A 控制等，如果不介意切换时间的话可以直接用延时 cover 掉模式切换时候 3A 的变化过程。

RAW sensor 不需要考虑这个过程。直接切换即可。

4.2.1.13. LED 补光灯功能

如果 YUV 的 sensor 需要添加 LED 补光灯的功能时候，可自行参考 gc2155.c 添加补光灯需要的代码。驱动中已将 flash 的 io 和 pmu 控制部分的代码抽出放在 flash_light 目录中。v4l2 定义的几种 flash 模式，这里仅能支持 NONE/AUTO/FLASH_ON/TORCH_ON 这几种。

sys_config.fex 中 csi0_dev0(/1)_flash_used 要配为 1；csi0_dev0(/1)_flash_type 用来区分 io 控制还是 pmu 控制，为 2 时表示使用 pmu 控制；根据 flash 类型还需要配置控制 LED 的 io 或者 pmu；如果是仅有 ON/OFF 控制的补光灯 driver，请将该 IO 配置到 csi0_dev0(/1)_flash_en 项上。

camera.cfg 中也需要将 used_flash_mode 配置成 1，否则 HAL 层不会发命令下来。

A64 上 led 补光灯驱动已经摒弃了之前 ov5640.c 的做法，ov5640 中 led 补光灯功能如下：

i. 定义 flash_dev_info 的结构体和相关变量，详细控制在 sunxi-vfe/vfe_subdev.c 中定义，下面是 device 文件中需要添加的代码：

```
#define FLASH_EN_POL 1
#define FLASH_MODE_POL 1
#define _FLASH_FUNC_

#ifdef _FLASH_FUNC_
static unsigned int to_flash=0;
static unsigned int flash_auto_level=0x1c; //由具体 sensor 决定
#endif
```

ii. 如果是需要做 auto flash 功能，需要在预览模式下去获取当前画面的亮度，然后自行设定一个阈值作为触发闪光灯的标准，下面是 ov5640 部分的代码，通过读取 0x56a1

寄存器获得亮度值。

```
#ifdef _FLASH_FUNC_
void check_to_flash(struct v4l2_subdev *sd)
{
   struct sensor_info *info = to_state(sd);
   if(info->flash_mode==V4L2_FLASH_LED_MODE_FLASH)
   {
      to_flash=1;
   }
   else if(info->flash_mode==V4L2_FLASH_LED_MODE_AUTO)
   {
      unsigned char lum;
      sensor_read(sd, 0x56a1, &lum);
      //printk("check luminance=0x%x\n",lum);
      if( lum<flash_auto_level )
         to_flash=1;
      else
         to_flash=0;
   }
   else
   {
      to_flash=0;
   }

   //printk("to_flash=%d\n",to_flash);
}
#endif
```

iii. 在 sensor_s_single_af 函数中，通过获取当前画面亮度后，在设置单次对焦命令后增加开启闪光灯的命令。

```
   #ifdef _FLASH_FUNC_
   if(info->flash_mode!=V4L2_FLASH_LED_MODE_NONE)
   {
      check_to_flash(sd);
      if(to_flash==1)
      {
         //vfe_dev_print("open torch when start single af\n");
```

```
        io_set_flash_ctrl(sd, SW_CTRL_TORCH_ON, info->fl_dev_info);
    }
  }
  #endif
```

iv. 在 sensor_g_single_af 函数中，分别在获取单次 af 动作成功或者失败或其他状态后将 LED 关闭，在 busy 状态下不去开关 LED（前提是 YUV 的 sensor 控制 single af 返回状态不能一直是 busy，其内部必须能够在一段时间内超时后返回失败或其他状态）。

```
    #ifdef _FLASH_FUNC_
    if(info->flash_mode!=V4L2_FLASH_LED_MODE_NONE)
    {
      vfe_dev_print("shut flash when af fail/ok\n");
      io_set_flash_ctrl(sd, SW_CTRL_FLASH_OFF, info->fl_dev_info);
    }
    #endif

    #ifdef _FLASH_FUNC_
    if(info->flash_mode!=V4L2_FLASH_LED_MODE_NONE)
    {
      vfe_dev_print("shut flash when af idle 2\n");
      io_set_flash_ctrl(sd, SW_CTRL_FLASH_OFF, info->fl_dev_info);
    }
    #endif
```

v. 在进入 stanby 前，再将 flash 关闭一次，避免异常状态时候无法关闭补光灯

```
case CSI_SUBDEV_STBY_ON:
      vfe_dev_dbg("CSI_SUBDEV_STBY_ON!\n");
      #ifdef _FLASH_FUNC_
      io_set_flash_ctrl(sd, SW_CTRL_FLASH_OFF, to_state(sd)->fl_dev_info);
      #endif
```

vi. Sensor_init 函数里面，在返回前添加初始化 flash 相关参数

```
#ifdef _FLASH_FUNC_
  if(dev->flash_used==1)
  {
    info->fl_dev_info=&fl_info;
    info->fl_dev_info->dev_if=0;
    info->fl_dev_info->en_pol=FLASH_EN_POL;
```

```
        info->fl_dev_info->fl_mode_pol=FLASH_MODE_POL;
        info->fl_dev_info->light_src=0x01;
        info->fl_dev_info->flash_intensity=400;//如果要区分预览和拍照时候光强的话可使用
        info->fl_dev_info->flash_level=0x01;
        info->fl_dev_info->torch_intensity=200;
        info->fl_dev_info->torch_level=0x01;
        info->fl_dev_info->timeout_counter=300*1000;
        vfe_dev_print("init flash mode[V4L2_FLASH_LED_MODE_NONE]\n");
        config_flash_mode(sd, V4L2_FLASH_LED_MODE_NONE,
                                    info->fl_dev_info);
        io_set_flash_ctrl(sd, SW_CTRL_FLASH_OFF, info->fl_dev_info);
    }
    #endif
```

更新后的 led 补光灯驱动可以参考 gc2155.c，新的驱动将 led 的控制部分交给 vfe.c 去处理，sensor 中只需要提供读取当前帧亮度以及判断该亮度是否需要补光的接口即可，该接口会被上层驱动在适当的时机调用，用来判断是否需要补光。

```
static unsigned int flash_auto_level=0x13c;//由具体 sensor 决定
static void sensor_get_lum(struct v4l2_subdev *sd, unsigned int *lum)
{
        data_type temp = 0;
        sensor_read(sd, 0x03, &temp);
        *lum = temp << 8;
        sensor_read(sd, 0x04, &temp);
        *lum |= temp;
}
static void sensor_g_flash_flag(struct v4l2_subdev *sd, unsigned int *flash_flag)
{
        unsigned int current_lum = 0;
        sensor_get_lum(sd, &current_lum);
        if(current_lum > flash_auto_level)
                *flash_flag = 1;
        else
                *flash_flag = 0;
}

static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)
{
        int ret=0;
```

```
    switch(cmd) {
        ......
        case GET_FLASH_FLAG:
            sensor_g_flash_flag(sd,(unsigned int *)arg);
        default:
            return -EINVAL;
    }
    return ret;
}
```

4.2.1.14. 定义 sensor 的输出格式和配置

包括定义 i. sensor 输出的格式、ii. 主控端在 sensor 不同输出需要配置成的相应参数、iii. sensor 在不同分辨率时候的配置：

i. sensor 输出的格式

(1)使用并口的 YUV sensor 时候填写如下，一般不需要修改，只是需要实现 yuv 的 4 种顺序

```
static struct sensor_format_struct {
  __u8 *desc;
  //__u32 pixelformat;
  enum v4l2_mbus_pixelcode mbus_code;
  struct regval_list *regs;
  int regs_size;
  int bpp;    /* Bytes per pixel */
} sensor_formats[] = {
  {
    .desc     = "YUYV 4:2:2",
    .mbus_code    = V4L2_MBUS_FMT_YUYV8_2X8,
    .regs         = sensor_fmt_yuv422_yuyv,
    .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_yuyv),
    .bpp      = 2,
  },
  {
    .desc     = "YVYU 4:2:2",
    .mbus_code    = V4L2_MBUS_FMT_YVYU8_2X8,
    .regs         = sensor_fmt_yuv422_yvyu,
    .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_yvyu),
    .bpp      = 2,
  },
  {
```

```
        .desc      = "UYVY 4:2:2",
        .mbus_code    = V4L2_MBUS_FMT_UYVY8_2X8,
        .regs         = sensor_fmt_yuv422_uyvy,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_uyvy),
        .bpp       = 2,
    },
    {
        .desc      = "VYUY 4:2:2",
        .mbus_code    = V4L2_MBUS_FMT_VYUY8_2X8,
        .regs         = sensor_fmt_yuv422_vyuy,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_vyuy),
        .bpp       = 2,
    },
//  {
//      .desc      = "Raw RGB Bayer",
//      .mbus_code    = V4L2_MBUS_FMT_SBGGR8_1X8,
//      .regs         = sensor_fmt_raw,
//      .regs_size = ARRAY_SIZE(sensor_fmt_raw),
//      .bpp       = 1
//  },
};
```

(2)使用 BT656 接口的器件时，如 TV decoder，一般是输入 PAL/NTSC 两种格式的视频信号，其信号一般是固定为 UYVY 的，只需要填写一组即可。

```
static struct sensor_format_struct {
    __u8 *desc;
    //__u32 pixelformat;
    enum v4l2_mbus_pixelcode mbus_code;
    struct regval_list *regs;
    int regs_size;
    int bpp;      /* Bytes per pixel */
} sensor_formats[] = {
    {
        .desc      = "UYVY 4:2:2",
        .mbus_code    = V4L2_MBUS_FMT_UYVY8_2X8,
        .regs         = sensor_fmt_yuv422_uyvy,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuv422_uyvy),
        .bpp       = 2,
    },
```

```
};
```

ii. 主控端需要配置成的相应参数：

并口的 sensor 填写接口参数（YUV/RAW）

```
static int sensor_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *cfg)
{
    cfg->type = V4L2_MBUS_PARALLEL;
    cfg->flags = V4L2_MBUS_MASTER | VREF_POL | HREF_POL | CLK_POL ;
    return 0;
}
```

BT656 信号输入时候填写接口参数（YUV）

```
static int sensor_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *cfg)
{
    cfg->type = V4L2_MBUS_BT656;
    cfg->flags = 0 ;//no use here
    return 0;
}
```

iii. sensor 在不同分辨率时候的配置：

就是填写 sensor 不同分辨率配置的结构体。

```
struct sensor_win_size {
    int width;
    int height;
    unsigned int hoffset;         //receive hoffset from sensor output
    unsigned int voffset;         //receive voffset from sensor output
    unsigned int hts;              //h size of timing, unit: pclk
    unsigned int vts;              //v size of timing, unit: line
    unsigned int pclk;             //pixel clock in Hz
    unsigned int mipi_bps;             //mipi clock in bps, fill this if config for mipi,
    unsigned int fps_fixed;      //fps mode 1=fixed fps
                                       //N=varied fps to 1/N of org fps
    unsigned int bin_factor;    //binning factor
    unsigned int intg_min;       //integration min, unit: line, Q4
    unsigned int intg_max;       //integration max, unit: line, Q4
    unsigned int gain_min;       //sensor gain min, Q4
    unsigned int gain_max;       //sensor gain max, Q4
    void * regs; /* Regs to tweak */
```

```
    int regs_size;
    int (*set_size) (struct v4l2_subdev *sd);
/* h/vref stuff */
};
```

YUV sensor 一般只需要填写 size 和对应配置的数组，需要从某个 size 中 crop 的时候填写 hoffset 和 voffset，其他参数不必填。

```
static struct sensor_win_size sensor_win_sizes[] = {
    /* qsxga: 2592*1936 */
    {
        .width          = QSXGA_WIDTH,
        .height         = QSXGA_HEIGHT,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_qsxga_regs,
        .regs_size      = ARRAY_SIZE(sensor_qsxga_regs),
        .set_size       = NULL,
    },
    /* qxga: 2048*1536 */
    {
        .width          = QXGA_WIDTH,
        .height         = QXGA_HEIGHT,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_qxga_regs,
        .regs_size      = ARRAY_SIZE(sensor_qxga_regs),
        .set_size       = NULL,
    },
    /* 1080P */
    {
        .width          = HD1080_WIDTH,
        .height         = HD1080_HEIGHT,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_1080p_regs,
        .regs_size      = ARRAY_SIZE(sensor_1080p_regs),
        .set_size       = NULL,
    },
    /* UXGA */
```

```
{
    .width          = UXGA_WIDTH,
    .height         = UXGA_HEIGHT,
    .hoffset        = 0,
    .voffset        = 0,
    .regs           = sensor_uxga_regs,
    .regs_size      = ARRAY_SIZE(sensor_uxga_regs),
    .set_size       = NULL,
},
/* SXGA */
{
    .width          = SXGA_WIDTH,
    .height         = SXGA_HEIGHT,
    .hoffset        = 0,
    .voffset        = 0,
    .regs           = sensor_sxga_regs,
    .regs_size      = ARRAY_SIZE(sensor_sxga_regs),
    .set_size       = NULL,
},
/* 720p */
{
    .width          = HD720_WIDTH,
    .height         = HD720_HEIGHT,
    .hoffset        = 0,
    .voffset        = 0,
    .regs           = sensor_720p_regs,
    .regs_size      = ARRAY_SIZE(sensor_720p_regs),
    .set_size       = NULL,
},
/* XGA */
{
    .width          = XGA_WIDTH,
    .height         = XGA_HEIGHT,
    .hoffset        = 0,
    .voffset        = 0,
    .regs           = sensor_xga_regs,
    .regs_size      = ARRAY_SIZE(sensor_xga_regs),
    .set_size       = NULL,
},
```

```
    /* SVGA */
    {
        .width          = SVGA_WIDTH,
        .height         = SVGA_HEIGHT,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_svga_regs,
        .regs_size      = ARRAY_SIZE(sensor_svga_regs),
        .set_size       = NULL,
    },
    /* VGA */
    {
        .width          = VGA_WIDTH,
        .height         = VGA_HEIGHT,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_vga_regs,
        .regs_size      = ARRAY_SIZE(sensor_vga_regs),
        .set_size       = NULL,
    },
};
```

BT656 的设备可以仅仅实现 PAL/NTSC 两种 size，其中 PAL 的是每秒 50 场/25fps，NTSC 是每秒 60 场/30fps，一般是将这种信号以 frame 的形式接受下来（与主控后面显示和编码有关，一般是接收成 frame 的形式）。如果该器件有其他格式的话逐个实现即可。

```
static struct sensor_win_size sensor_win_sizes[] = {
    /* PAL */
    {
        .width          = 704,
        .height         = 480,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_pal_regs,
        .regs_size      = ARRAY_SIZE(sensor_pal_regs),
        .set_size       = NULL,
    },
    /* NTSC */
    {
        .width          = 704,
```

```
        .height         = 576,
        .hoffset        = 0,
        .voffset        = 0,
        .regs           = sensor_ntsc_regs,
        .regs_size      = ARRAY_SIZE(sensor_ntsc_regs),
        .set_size       = NULL,
    },
};
```

4.2.2. 内核代码注意事项

1） 驱动中的延时语句一般禁止使用 mdelay()，msleep 的话特别是较短 10~20ms 的时候常常会因为系统调度变成更长的时间，精度较差，需要较为精确的 ms 级别延时请使用 usleep_range(a, b)，比如原来 mdelay(1)、mdelay(10)可改为 usleep_range(1000, 2000)、usleep_range(10000, 12000)，如果是长达 30ms 或以上的延时使用 msleep()；

2） 中断过程中不能使用 msleep 和 usleep_range，除了特殊情况必须加延时之外，mdelay 一般也不可使用。

4.3. Android 部分

init.sun50iw1p1.rc 的 camera 部分添加必要的 ko。亦可手动写个 bat 脚本方便加载和卸载 ko。命令参考如下：

```
# csi module
    insmod /system/vendor/modules/videobuf2-core.ko
    insmod /system/vendor/modules/videobuf2-memops.ko
    insmod /system/vendor/modules/videobuf2-dma-contig.ko
    insmod /system/vendor/modules/vfe_io.ko
    insmod /system/vendor/modules/gc0328c.ko
    insmod /system/vendor/modules/gc2155.ko
    insmod /system/vendor/modules/vfe_v4l2.ko
```

Camera.cfg 中注意配置黄色部分，sensor 驱动支持什么样的分辨率，就填写相应的分辨率。修改 camera.cfg 之后可执行 adb push camera.cfg system/etc/。然后杀掉 media 进程或者 reboot 进行更新。

```
camera_id = 0
;如果是单摄像头，只填写 camera_id = 0 部分的配置
;如果是双摄像头，这个部分的配置是对应后置摄像头，
;前置的还需要去配置 camera_id = 1 的部分
;----------------------------------------------------------------------------
```

```
; 1 for CAMERA_FACING_FRONT
; 0 for CAMERA_FACING_BACK
;-------------------------------------------------------------------------
camera_facing = 0

;-------------------------------------------------------------------------
; 1 for camera without isp(using built-in isp of Axx)
; 0 for camera with isp
;-------------------------------------------------------------------------
use_builtin_isp = 0

;-------------------------------------------------------------------------
; camera orientation (0, 90, 180, 270)
;-------------------------------------------------------------------------
camera_orientation = 0

;-------------------------------------------------------------------------
; driver device name
;-------------------------------------------------------------------------
camera_device = /dev/video1
;如果 sysconfig 配置了 mipi 接口这个地方要写 video0，并口的填 video0
;-------------------------------------------------------------------------
; device id
; for two camera devices with one CSI
;-------------------------------------------------------------------------
device_id = 0

used_preview_size = 1
key_support_preview_size = 1920x1080,1280x720,640x480
key_default_preview_size = 1280x720

used_picture_size = 1
key_support_picture_size = 2592x1936,1600x1200,1280x1024
key_default_picture_size = 2592x1936

used_flash_mode = 1
key_support_flash_mode = on,off,auto
key_default_flash_mode = on
```

```
used_color_effect=1
key_support_color_effect = none,mono,negative,sepia,aqua
key_default_color_effect = none

used_frame_rate = 1
key_support_frame_rate = 25
key_default_frame_rate = 25

used_focus_mode = 1
key_support_focus_mode = auto,infinity,macro,fixed,continuous-video,continuous-picture
key_default_focus_mode = auto

used_scene_mode = 0
key_support_scene_mode =
auto,portrait,landscape,night,night-portrait,theatre,beach,snow,sunset,steadyphoto,fireworks,sports,
party,candlelight,barcode
key_default_scene_mode = auto

used_white_balance = 1
key_support_white_balance =
auto,incandescent,fluorescent,warm-fluorescent,daylight,cloudy-daylight
key_default_white_balance = auto

used_exposure_compensation = 1
key_max_exposure_compensation = 4
key_min_exposure_compensation = -4
key_step_exposure_compensation = 1
key_default_exposure_compensation = 0
```

media_profiles.xml 的配置，在 <CamcorderProfiles cameraId="0">或者对应 camera 的配置中，如果需要 1080p 录像（前提是驱动支持）需要添加 1080p 的配置。

```
        <EncoderProfile quality="1080p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="10000000"
                   width="1920"
                   height="1080"
                   frameRate="30" />

            <Audio codec="aac"
                   bitRate="128000"
```

```
                    sampleRate="44100"
                    channels="1" />
        </EncoderProfile>
```

修改 media_profiles.xml 之后可 adb push media_profiles.xml system/etc/。之后 reboot 进行更新。

5. 模块调试常见问题

初次调试建议打开 device 中的 DEV_DBG_EN 为 1，方便调试。

1)、使用 lsmod 命令查看驱动是否加载

2)、使用 ls /dev/v*查看是否有 video0/1节点生成

3)、在 adb shell 中使用 cat /proc/kmsg 命令，或者是使用串口查看内核的打印信息，查看不能正常加载的原因，一般情况下驱动加载不成功的原因有：一是读取的 sys_config.fex 文件中的配置信息与加载的驱动不匹配，二是 probe 函数遇到某些错误没能正确的完成 probe 的时候返回。

5.1. 调试 camera 常见现象和功能检查

1) Insmod 之后首先看内核打印，看加载有无错误打印，部分驱动在加载驱动进行上下电时候会进行 i2c 操作，如果此时报错的话就不需要再进入 camera 了，先检查是否 io 或电源配置不对。或者是在复用模组时候有可能是另外一个模组将 i2c 拉住了。

2) 如果 i2c 读写没有问题的话，一般就可以认为 sensor 控制是 ok 的，只需要根据 sensor 的配置填好 H/VREF、PCLK 的极性就能正常接收图像了。这个时候可以在进入 camera 应用之后用示波器测量 sensor 的各个信号，看 h/vref、pclk 极性、幅度是否正常（2.8V 的 vpp）。

3) 如果看到画面了，但是看起来是绿色和粉红色的，但是有轮廓，一般是 YUYV 的顺序设置反了，可检查 yuyv 那几个寄存器是否填写正确配置，其次，看是否是在配置的其他地方有填写同一个寄存器的地方导致将 yuyv fmt 的寄存器被改写。

4) 如果画面颜色正常，但是看到有一些行是粉红或者绿色的，往往是 sensor 信号质量不好所致，通常在比较长的排线中出现这个情况。在信号质量不好并且 yuyv 顺序不对的时候也会看见整个画面的是绿色的花屏。

5) 当驱动能力不足的时候增强 sensor 的 io 驱动能力有可能解决这个问题。此时用示波器观察 pclk 和数据线可能会发现：pclk 波形摆幅不够 IOVDD 的幅度，或者是 data 输出波形摆幅有时候能高电平达到 IOVDD 的幅度，有时候可能连一半都不够

6) 如果是两个模组复用数据线的话，不排除是另外一个 sensor 在进入 standby 时候没有将其数据线设置成高阻，也会影响到当前模组的信号摆幅，允许的话可以剪断另一个模组来证实。

7) 当画面都正常之后检查前置摄像头垂直方向是否正确，水平方向是否是镜像，后置水平

垂直是否正确，不对的话可以调节 sys_config.fex 中的 hflip 和 vflip 参数来解决，但如果屏幕上看到的画面与人眼看到的画面是成 90 度的话，只能是通过修改模组的方向来解决。

8) 之后可以检查不同分辨率之间的切换是否 ok，是否有切换不成功的问题；以及拍照时候是否图形正常，亮度颜色是否和预览一致；双摄像头的话需要检查前后切换是否正常。

9) 如果上述都没有问题的话，可认为驱动无大问题，接下来可以进行其他功能（awb/exp bias/color effect 等其他功能的测试）。

10) 测试对焦功能，单次点触屏幕，可正确对上不同距离的物体；不点屏幕时候可以自动对焦对上画面中心物体，点下拍照后拍出来的画面能清晰。

11) 打开闪光灯功能，检查在单次对焦时候能打开灯，对完之后无论成功失败或者超时能够关闭，在点下拍照之后能打开，拍完之后能关闭。

如果加载模块后，发现 dev/videoX 节点没有生成，请检查下面几点。

a. 模块加载的顺序

一定要按照以下顺序加载模块

```
insmod videobuf2-core.ko
insmod videobuf2-memops.ko
insmod videobuf2-dma-contig.ko
;如果有对应的 vcm driver，在这里加载，如果没有，请省略。
insmod actuator.ko
insmod ad5820_act.ko
;以下是 camera 驱动和 vfe 驱动的加载，先安装一些公共资源。
insmod vfe_io.ko
insmod ov5640.ko
insmod gc0308.ko
;如果一个 csi 接两个 camera，所有 camera 对应的 ko 都要在 vfe_v4l2.ko 之前加载。
insmod vfe_v4l2.ko
```

b. sys_config.fex 配置

```
csi0_used               = 1       ;确保 used 为 1
csi0_dev0_mname     = "ov5640"    ;确保 camera 型号与 ko 加载情况相同
```

5.2. I2C 通信出现问题

内核一般会伴随打印" cci_write_aX_dX error! slave = 0xXX, addr = 0xXX, value = 0xXX"。如果与此同时，内核出现打印"chip found is not an target chip."，则说明在初始化 camera 前，读取 camera 的 ID 已经失败。此时，一般是如下几点出现问题：

a. 最先考虑应该是更换一个 camera 模组试试。

b. 检查 i2c 地址填写是否正确

csi0_dev0_mname = "gc2155"

csi0_dev0_twi_addr = 0x78

c. 电源

检查 sysconfig.fex

csi0_dev0_iovdd = "axp22_eldo3"

csi0_dev0_iovdd_vol = 2800000

csi0_dev0_avdd = "axp22_dldo4"

csi0_dev0_avdd_vol = 2800000

csi0_dev0_dvdd = "axp22_eldo2"

vip_dev0_dvdd_vol = 1500000

一定要与原理图设计保持一致。必要时，需要用万用表测量 camera 模组的各路电压是否正常。

d. reset 和 power down 脚

检查 sysconfig 配置

csi0_dev0_reset = port:PH2<1><default><default><default>

csi0_dev0_pwdn = port:PH1<1><default><default><default>

是否与原理图设计保持一致。必要时，需要用示波器测量 reset，pwdn 脚，在 camera 加载时，是否有动作。

e. mclk

检查 sysconfig 配置

csi0_csi_mck = port:PE01<3><default><default><default>

pin 脚是否与原理图设计保持一致。必要时，在加载 camera 时，测量 mclk，看是否有正确输出（一般是 24MHz 或 27MHz）。

如果已经能够正确通过 camera 的 id 读取，只是在使用过程当中，偶尔出现 I2C 的读写

错误，此时需要从打印里面，将报错的地址和读写值，结合 camera 具体的 spec 来分析，到底是操作了 camera 哪些寄存器带来的问题。

5.3. 画面大体轮廓正常，颜色出现大片绿色和紫红色

一般可能是 csi 采样到的 yuyv 顺序出现错位。

a. 确认 camera 输出的 yuyv 顺序的设置与 camera 的 spec 一致；

b. 若 camera 输出的 yuyv 顺序没有问题，则可能是由于走线问题，导致 pclk 采样 data 时发生错位，此时可以调整 pclk 的采样沿。具体做法是：在对应的 camara 驱动源码，如 ov5640.c 里面，找到宏定义#define CLK_POL。此宏定义可以有两个值 V4L2_MBUS_PCLK_SAMPLE_RISING 和 V4L2_MBUS_PCLK_SAMPLE_FALLING。若原来是其中一个值，则修改成另外一个值，便可将 PCLK 的采样沿做反相。

5.4. 画面大体轮廓正常，但出现不规则的绿色紫色条纹

一般可能是 pclk 驱动能力不足，导致某个时刻采样 data 时发生错位。解决办法：

a. 若 pclk 走线上有串联电阻，尝试将电阻阻值减小。

b. 增强 pclk 的驱动能力，需要设置 camera 的内部寄存器。

5.5. 画面看起来像油画效果，过渡渐变的地方有一圈一圈

一般是 CSI 的 data 线没有接好，或短路，或断路。

5.6. camera 使用长时间后画面变色

有可能是 sensor 模组的封装散热不够好。出现过 ov5642 的模组使用塑料外壳加金属背板的封装形式，工作时散热不够，视环境温度不同使用几分钟到十几分钟后画面出现粉红色的噪点，画面变得模糊；但是只需将背板贴紧 LCD 的外壳辅助散热就没有问题。

5.7. 使用长时间后画面变色

有可能是 sensor 模组的封装散热不够好。出现过 ov5642 的模组使用塑料外壳加金属背板的封装形式，工作时散热不够，视环境温度不同使用几分钟到十几分钟后画面出现粉红色的噪点，画面变得模糊；但是只需将背板贴紧 LCD 的外壳辅助散热就没有问题。

5.8. 串口没有打印就死机

如果遇到串口没有打印就死机的问题，一般都是 CLK 没有设置正确。

5.9. 出现[VFE_WARN] Nobody is waiting on this video buffer

Hal 层还回来所有的 buffer，但是没有再来取 buffer。

5.10. 出现[VFE_WARN] Only three buffer left for csi

Hal 层占用了大部分 buffer，没有还回，驱动部分只有三个 buffer 此时驱动不再进行 buffer 切换，直到有 buffer 还回为止。

6. Sensor 的硬件接口注意事项

1、 如果是使用并口的 sensor 模组，会使用到 720p@30fps 或更高速度的，必须在 mclk/pclk/data/vsync/hsync 上面串 33ohm 电阻，5M 的 sensor 一律串电阻；
2、 使用 Mipi 模组时候 PCB layout 需要尽量保证 clk/data 的差分对等长，过孔数相等，特征阻抗 100ohm；
3、 如果使用并口复用 pin 的模组时候，不建议 reset 脚的复用；
4、 并口模组的排线长度加上 pcb 板上走线长度不超过 10cm， mipi 模组排线长度加上 pcb 板上走线长度不超过 20cm，超过此距离不保证能正常使用。
5、 主控并口数据线有 D7~D0 共 8bit，并口的 sensor 输出一般为 8/10bit，原理图连接需要做高位对齐。

7. 一些方案中出现的 camera 相关疑难杂症

下面是一些方案上面调试 camera 发现的比较难察觉到的问题：

案例 1：

某客户方案，使用 gc2035+gc0308，复用数据线、电源、信号线，gc2035 在拍照预览 VGA 下画面出现花屏，反复进出 camera 应用都一样，但是切换到 720p 时候有时候能正常，拍照 2M 的话输出图像也是正常的，切换至前置 gc0308 是正常的。检查配置 ok，驱动 ok，查看原理图连接也无问题。

只能用示波器观察信号，hvref、幅度极性 ok，pclk 幅度稍低，仅仅 30 多 MHz 的 pclk 摆幅却只有 2V 多一些，但还属正常，但是数据线上看起来高电平时候不够摆幅。虽然是和另外一个 sensor 复用数据线，但驱动上没有问题，理论上不会导致数据线拉住主 sensor。再查看电源部分，IOVDD、AVDD 均无问题，但是 DVDD 电压明显不对，且看起来随着 VREF 的有效和无效区变化，默认应该是 1.8V 的 DVDD，在 pin 脚上只有 1.3~1.5 左右变化，明显低于工作电压，但是，只要一拍照，电压就恢复到 1.6~1.8V。

最后检查出原理图上 DVDD 的地方是连接了一个 0ohm 电阻的，但实际上再板子上焊接的是一个 10 ohm 的电阻，直接导致了 DVDD 电源的波动到无法正常工作的电压，改为 0ohm 之后正常。

gc0308 的 DVDD（默认 1.8V）工作时候电流可能较小，或者是本身能比 gc2035 在更低一点的电压下工作，导致即使共用了 DVDD 的 pin 也没有发现问题。否则，应该会直接去测量电源，而 gc0308 的正常工作反而给人错觉：供电和 pin 脚连接都是 ok 的，问题可能在复用上面。

案例 2：

某客户方案，加载驱动之后 i2c 通讯失败，检查 io 正常，但是 MCLK 无信号，是一个 0.7V 的直流电平。检查 sysconfig 配置没有问题，检查原理图和 pcb，确认 MCLK 连接没有问题，该 io’直接改写寄存器也能够正常输出高低电平。检查驱动发现没有配置 MCLK 的频率（默认输出的频率可能是一个很高频的 clock，示波器可能检测不到），加上之后仍然没有 clock 信号，但是直流电平变为 1V 左右。这个改变推断出有可能在 MCLK 信号上面存在一个很大的电容，把 MHz 级别的信号都滤成了直流。再次检查原理图才发现原理图在 MCLK 上面接了个 10nF 的电容，且默认不是 NC。去掉之后 MCLK 正常。

通常情况下 MCLK、PCLK 的信号是不需要添加滤波电容的，即使添加的话也是 pF 级

别且默认 NC 的，这个地方原理图没有遵循这个规则，且没有检查出这个问题。

案例 3：

某客户自行打板做的 sensor 子板，sensor 的 die 是直接焊接在子板上面的，原理图 pcb 都无问题，电源、io 控制也正常，MCLK 也有，但是 i2c 读写就是失败。

因为是专门做的子板接口也没办法替换 sensor 上去测试。只能先检查其子板。但是从外围器件来看也没有发现连接问题。

最后查看排线接口，用万用表测量了多个数据管脚对地和对电源的二极管特性，发现都不正常。判断使用的 die 有问题，或者是 die 被焊反了。让客户工程师检查后确认为 die 焊反。

案例 4：

A64 原型机采用 gc2155 和 gc0328c 二合一模组，单独加载某个 sensor 驱动时，CCI 通信正常，且能够正常出图；两个 sensor 一起加载时第一个 sensor 能够成功加载，第二个 sensor 的 CCI 通信失败，调换顺序也是第一个 sensor 成功第二个失败。

开始怀疑是二合一模组第一个 sensor 对第二个有影响，但是测量各路电压以及 IO 引脚的电平都正常。重新对照原理图，发现二合一模组只有 PWDN 引脚未复用其他引脚都有复用，最后尝试在 sys_config.fex 上将 PWDN 引脚调换顺序，重新打包、烧写，两个 sensor 的 CCI 通信都正常了，且出图正常。因而确认是原理图上将两个 sensor 的 pwdn 引脚标反了。

分析此现象的原因是：默认情况下两个 PWDN 引脚的电平都是低电平，根据数据手册 PWDN 低电平时 sensor 在 power on 上电后能够正常工作。由于 PWDN 连反的缘故，加载第一个 sensor 的时候实际操作的是第二个 sensor 的 PWDN 引脚，且在 power off 的时候将第二个 sensor 的 PWDN 引脚置高，从而导致第二个 sensor 在加载时 CCI 通信失败。而且单独加载某个 sensor 由于 PWDN 一直未被操作处于低电平状态，所以 CCI 通信能够成功且能正常出图。