Tina Linux Camera
开发指南

版本号: 1.7
发布日期: 2022.02.15

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.07.11	AWA1450	初始版本
1.1	2020.02.24	AWA1450	补充 sensor 驱动部分
1.2	2020.05.25	AWA1450	补充 sensor 上电参数解析和 ISP 配置解析
1.3	2020.06.29	AWA1450	补充 RAW 格式支持以及曝光时间计算
1.4	2021.01.09	AWA1450	增加介绍如何修改 isp 亮度，增加图像异常排除方式
1.5	2021.04.22	AWA1450	增加 R528 平台配置介绍
1.6	2021.07.01	AWA1681	增加 AF 模块配置
1.7	2022.02.15	AWA1606	增加 V 系列平台配置

目　录

插　图

1 概述

1.1 编写目的

介绍 camera 模块在 sunxi 平台上的开发流程。

1.2 适用范围

本文档目前适用于 tina3.0 以上具备 camera 的硬件平台。

1.3 相关人员

公司开发人员、客户。

2 模块介绍

2.1 模块功能介绍

用于接收并行或者 mipi 接口的 sensor 信号或者是 bt656 格式的信号。

2.2 硬件介绍

目前 Tina 系统的各平台 camera 硬件接口、linux 内核版本以及 camera 驱动框架如下表所示：

表 2-1：平台 CSI 框架

平台	支持接口	是否具备 ISP 模块	linux 内核版本	camera 驱动框架
F35	并口 csi、mipi	否	3.4	VFE
R16	并口 csi	否	3.4	VFE
R18	并口 csi	否	4.4	VFE
R30	并口 csi	否	4.4	VFE
R40	并口 csi	否	3.10	VFE
R311	mipi csi	是	4.9	VIN
MR133	mipi csi	是	4.9	VIN
R818	mipi csi	是	4.9	VIN
MR813	mipi csi	是	4.9	VIN
R528	并口 csi	否	5.4	VIN
V536	并口 csi、mipi	是	4.9	VIN
V533	并口 csi、mipi	是	4.9	VIN
V831	并口 csi、mipi	是	4.9	VIN
V833	并口 csi、mipi	是	4.9	VIN
V851	并口 csi、mipi	是	4.9	VIN
V853	并口 csi、mipi	是	4.9	VIN

注意：

1. 如果平台没有 ISP 模块，那么将不支持 RAW sensor（即 sensor 只输出采集到的原始数据），文档中提到的 RAW 等相关信息不用理会；
2. 如果平台没有支持 mipi 接口，文档中提到的 mipi 相关信息忽略；

3. 不同平台可能将使用不同的 camera 驱动框架，这点注意区分；

2.3 源码结构介绍

2.3.1 linux3.4 VFE 框架

驱动路径位于 linux-3.4/drivers/media/video/sunxi-vfe 下。

```
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sys_config.fex的参数配置和isp参数
    |  config.h                     ;读取sys_config.fex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
    |      dw9714_act.c             ;具体vcm driver型号实现
    |      Makefile
    |
    ├─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
    |
    ├─device
    |      camera.h                 ;camera公用结构体头文件
    |      camera_cfg.h             ;camera ioctl扩展命令头文件
    |      Makefile
    |      ov5640.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
    |      lib_mipicsi2_v1          ;A31mipi库
    |      lib_mipicsi2_v2          ;A80/A83mipi库
```

```
    ├─csi_cci
    │       cci_helper.c              ;cci 与设备相关初始化、注册以及通信等相关函数集实现
    │       cci_helper.h              ;cci 与设备相关初始化、注册以及通信等相关函数集实现头文件
    │       bsp_cci.c                 ;cci 操作函数集实现
    │       bsp_cci.h                 ;cci 操作函数集头文件
    │       csi_cci_reg.c             ;cci 底层实现
    │       csi_cci_reg.h             ;cci 底层实现头文件
    │       csi_cci_reg_i.h           ;cci寄存器资源头文件
    │
    ├─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
    │
    └─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
```

2.3.2 linux3.10 VFE 框架

驱动路径位于 linux-3.10/drivers/media/platform/sunxi-vfe 下。

```
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sys_config.fex的参数配置和isp参数
    │  config.h                      ;读取sys_config.fex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
    │       dw9714_act.c              ;具体vcm driver型号实现
    │       Makefile
    │
    ├─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
    │
    ├─device
    │       camera.h                  ;camera公用结构体头文件
    │       camera_cfg.h              ;camera ioctl扩展命令头文件
    │       Makefile
    │       ov5640.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
    |       lib_mipicsi2_v1           ;A31mipi库
    |       lib_mipicsi2_v2           ;A80/A83mipi库
    ├─csi_cci
    |       cci_helper.c              ;cci 与设备相关初始化、注册以及通信等相关函数集实现
    |       cci_helper.h              ;cci 与设备相关初始化、注册以及通信等相关函数集实现头文件
    |       bsp_cci.c                 ;cci 操作函数集实现
    |       bsp_cci.h                 ;cci 操作函数集头文件
    |       csi_cci_reg.c             ;cci 底层实现
    |       csi_cci_reg.h             ;cci 底层实现头文件
    |       csi_cci_reg_i.h           ;cci寄存器资源头文件
    |
    ├─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
    |
    └─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
```

2.3.3 linux4.4 VFE 框架

驱动路径位于 linux-4.4/drivers/media/platform/sunxi-vfe 下。

```
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sys_config.fex的参数配置和isp参数
    |  config.h                     ;读取sys_config.fex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
    |       dw9714_act.c            ;具体vcm driver型号实现
    |       Makefile
    |
    ├─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
    |
    ├─device
    |       camera.h                 ;camera公用结构体头文件
    |       camera_cfg.h             ;camera ioctl扩展命令头文件
    |       Makefile
    |       ov5640.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
    |       lib_mipicsi2_v1          ;A31mipi库
    |       lib_mipicsi2_v2          ;A80/A83mipi库
    ├─csi_cci
    |       cci_helper.c             ;cci 与设备相关初始化、注册以及通信等相关函数集实现
    |       cci_helper.h             ;cci 与设备相关初始化、注册以及通信等相关函数集实现头文件
    |       bsp_cci.c                ;cci 操作函数集实现
    |       bsp_cci.h                ;cci 操作函数集头文件
    |       csi_cci_reg.c            ;cci 底层实现
    |       csi_cci_reg.h            ;cci 底层实现头文件
    |       csi_cci_reg_i.h          ;cci寄存器资源头文件
    |
    ├─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
    |
    └─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
```

2.3.4 linux4.4 VIN 框架

驱动路径位于 linux-4.4/drivers/media/platform/sunxi-vin 下。

```
sunxi-vin:
    |  vin.c                         ;v4l2驱动实现主体（包含视频接口和ISP部分）
    |  vin.h                         ;v4l2驱动头文件
    |  top_reg.c                     ;vin对各v4l2 subdev管理接口实现主体
    |  top_reg.h                     ;管理接口头文件
    |  top_reg_i.h                   ;vin模块接口层部分结构体
    |
    ├─modules
    |   ├─actuator
    |     |     actuator.c           ;vcm driver的一般行为
    |     |     actuator.h           ;vcm driver的头文件
    |     |     dw9714_act.c         ;具体vcm driver型号实现
    |     |     Makefile
    |   ├─flash
```

```
|       |       flash.h                 ;led补光灯驱动头文件
|       |       flash_io.c              ;led补光灯io控制实现
|   ├─sensor
|       |       camera.h                ;camera公用结构体头文件
|       |       camera_cfg.h            ;camera ioctl扩展命令头文件
|       |       sensor_helper.c         ;sensor公用操作接口函数文件
|       |       sensor_helper.h         ;sensor公用操作接口函数头文件
|       |       Makefile
|       |       ov5640.c                ;具体的sensor驱动
|
├─platform
|       platform_cfg.h                  ;平台相关的配置接口
|
├─utility
|       bsp_common.h                    ;底层公用的格式配置函数头文件
|       bsp_common.c                    ;底层公用的格式配置函数文件
|       cfg_op.h                        ;解析配置文件接口头文件
|       cfg_op.c                        ;解析配置文件接口函数实现主体
|       config.h                        ;解析设备树的函数头文件
|       config.c                        ;解析设备树的接口函数主体
|       sensor_info.h                   ;sensor列表信息头文件
|       sensor_info.c                   ;获取sensor列表信息函数主体
|       vin_io.h                        ;vin框架io操作接口头文件
|       vin_io.c                        ;vin框架io操作接口文件
|       vin_os.h                        ;vin框架系统操作接口头文件
|       vin_os.c                        ;vin框架系统操作接口文件
|       vin_supply.h                    ;vin框架设置时钟频率等接口头文件
|       vin_supply.c                    ;vin框架设置时钟频率等接口函数主体
|
├─vin-cci
|       cci_helper.c                ;cci 与设备相关初始化、注册以及通信等相关函数集实现
|       cci_helper.h                ;cci 与设备相关初始化、注册以及通信等相关函数集实现头文件
|       bsp_cci.c                   ;cci 操作函数集实现
|       bsp_cci.h                   ;cci 操作函数集头文件
|       csi_cci_reg.c               ;cci 底层实现
|       csi_cci_reg.h               ;cci 底层实现头文件
|       csi_cci_reg_i.h             ;cci寄存器资源头文件
|       sunxi_cci.c                 ;cci 接口封装实现
|       sunxi_cci.h                 ;cci 接口封装头文件
|
├─vin-csi
|       bsp_csi.c                   ;csi 操作函数集实现
|       bsp_csi.h                   ;csi 操作函数集头文件
|       csi_reg.c                   ;csi 底层实现
|       csi_reg.h                   ;csi 底层实现头文件
|       csi_reg_i.h                 ;csi寄存器资源头文件
|       parser_reg.c                ;csi 底层实现
|       parser_reg.h                ;csi 底层实现头文件
|       parser_reg_i.h              ;csi寄存器资源头文件
|       sunxi_csi.c                 ;csi 接口封装实现
|       sunxi_csi.h                 ;csi 接口封装头文件
|
├─vin-isp
|       bsp_isp.c                   ;isp操作函数集实现
|       bsp_isp.h                   ;isp操作函数集头文件
|       bsp_isp_comm.h              ;isp结构体定义
|       isp_default_tbl.h           ;isp默认配置列表
|       isp_platform_drv.h          ;isp平台操作集头文件
|       isp_platform_drv.c          ;isp平台操作集实现
|       sunxi_isp.h                 ;sunxi平台isp操作集头文件
```

```
    |       sunxi_isp.c              ;sunxi平台isp操作集实现
    |
    ├─vin-mipi
    |       bsp_mipi_csi.c           ;底层mipi bsp函数
    |       bsp_mipi_csi.h           ;底层mipi bsp函数头文件
    |       bsp_mipi_csi_v1.c        ;sunxi平台底层mipi bsp接口函数
    |       sunxi_mipi.c             ;sunxi平台mipi实现
    |       bsp_mipi_csi.h           ;sunxi平台mipi实现头文件
    |       combo_common.c           ;combo common头文件
    |       protocol.h               ;protocol头文件
    ├─vin-stat
    |       vin_h3a.c                ;vin 3a操作函数
    |       vin_h3a.h                ;vin 3a操作函数头文件
    |       vin_ispstat.c            ;sunxi isp stat操作函数
    |       vin_ispstat.h            ;sunxi isp stat操作函数头文件
    |
    ├─vin-video
    |       dma_reg.c                ;csi模块dma操作函数
    |       dma_reg.h                ;csi模块dma操作函数头文件
    |       dma_reg_i.h              ;csi dma寄存器资源头文件
    |       vin_core.c               ;vin video核心函数
    |       vin_core.h               ;vin video核心函数头文件
    |       vin_video.c              ;vin video设备接口函数
    |       vin_video.h              ;vin video设备接口函数头文件
    ├─vin-vipp
    |       sunxi_scaler.c           ;scaler 子设备操作函数集
    |       sunxi_scaler.h           ;caler 子设备操作函数头文件
    |       vipp_reg.c               ;vipp操作函数集
    |       vipp_reg.h               ;vipp操作函数集头文件
    |       vipp_reg_i.h             ;vipp操作函数集资源头文件
```

2.3.5 linux4.9 VIN 框架

驱动路径位于 linux-4.9/drivers/media/platform/sunxi-vin 下。

```
sunxi-vin:
    |  vin.c                         ;v4l2驱动实现主体（包含视频接口和ISP部分）
    |  vin.h                         ;v4l2驱动头文件
    |  top_reg.c                     ;vin对各v4l2 subdev管理接口实现主体
    |  top_reg.h                     ;管理接口头文件
    |  top_reg_i.h                   ;vin模块接口层部分结构体
    ├── modules
    |   ├── actuator                 ;vcm driver
    |   |   ├── actuator.c
    |   |   ├── actuator.h
    |   |   ├── dw9714_act.c
    |   |   ├── Makefile
    |   ├── flash                    ;闪光灯 driver
    |   |   ├── flash.c
    |   |   └── flash.h
    |   └── sensor                   ;sensor driver
    |       ├── ar0144_mipi.c
    |       ├── camera_cfg.h         ;camera ioctl扩展命令头文件
    |       ├── camera.h             ;camera公用结构体头文件
    |       ├── Makefile
    |       ├── ov2775_mipi.c
    |       ├── ov5640.c
```

```
    │       ├── sensor-compat-ioctl32.c
    │       ├── sensor_helper.c          ;sensor公用操作接口函数文件
    │       ├── sensor_helper.h
    ├── platform                         ;平台相关的配置接口
    ├── utility
    │   ├── bsp_common.c
    │   ├── bsp_common.h
    │   ├── cfg_op.c
    │   ├── cfg_op.h
    │   ├── config.c
    │   ├── config.h
    │   ├── sensor_info.c
    │   ├── sensor_info.h
    │   ├── vin_io.h
    │   ├── vin_os.c
    │   ├── vin_os.h
    │   ├── vin_supply.c
    │   └── vin_supply.h
    ├── vin-cci
    │   ├── sunxi_cci.c
    │   └── sunxi_cci.h
    ├── vin-csi
    │   ├── parser_reg.c
    │   ├── parser_reg.h
    │   ├── parser_reg_i.h
    │   ├── sunxi_csi.c
    │   └── sunxi_csi.h
    ├── vin-isp
    │   ├── sunxi_isp.c
    │   └── sunxi_isp.h
    ├── vin-mipi
    │   ├── sunxi_mipi.c
    │   └── sunxi_mipi.h
    ├── vin-stat
    │   ├── vin_h3a.c
    │   ├── vin_h3a.h
    │   ├── vin_ispstat.c
    │   └── vin_ispstat.h
    ├── vin_test
    ├── vin-video
    │   ├── vin_core.c
    │   ├── vin_core.h
    │   ├── vin_video.c
    │   └── vin_video.h
    └── vin-vipp
        ├── sunxi_scaler.c
        ├── sunxi_scaler.h
        ├── vipp_reg.c
        ├── vipp_reg.h
        └── vipp_reg_i.h
```

2.3.6 linux5.4 VIN 框架

驱动路径位于 linux-5.4/drivers/media/platform/sunxi-vin 下。

```
sunxi-vin:
    ├── Kconfig
    ├── Makefile
    ├── modules
    │   ├── actuator                    ;vcm driver
    │   │   ├── actuator.c
    │   │   ├── actuator.h
    │   │   ├── dw9714_act.c
    │   │   ├── Makefile
    │   ├── flash                       ;flash driver
    │   │   ├── flash.c
    │   │   └── flash.h
    │   ├── sensor                      ;cmos sensor driver
    │   │   ├── camera_cfg.h
    │   │   ├── camera.h
    │   │   ├── gc0310_mipi.c
    │   │   ├── Makefile
    │   │   ├── ov2710_mipi.c
    │   │   ├── ov5640.c
    │   │   ├── sensor-compat-ioctl32.c
    │   │   ├── sensor_helper.c
    │   │   ├── sensor_helper.h
    │   ├── sensor-list
    │   │   ├── sensor_list.c
    │   │   └── sensor_list.h
    │   └── sensor_power                ;sensor上下电接口函数
    │       ├── Makefile
    │       ├── sensor_power.c
    │       └── sensor_power.h
    ├── platform
    ├── top_reg.c
    ├── top_reg.h
    ├── top_reg_i.h
    ├── utility                         ;驱动通用接口
    │   ├── bsp_common.c
    │   ├── bsp_common.h
    │   ├── cfg_op.c
    │   ├── cfg_op.h
    │   ├── config.c
    │   ├── config.h
    │   ├── vin_io.h
    │   ├── vin_os.c
    │   ├── vin_os.h
    │   ├── vin_supply.c
    │   └── vin_supply.h
    ├── vin.c                           ;sunxi-vin驱动注册入口
    ├── vin-cci                         ;i2c操作相关接口
    │   ├── bsp_cci.c
    │   ├── bsp_cci.h
    │   ├── cci_helper.c
    │   ├── cci_helper.h
    │   ├── Kconfig
    │   ├── sunxi_cci.c
    │   └── sunxi_cci.h
    ├── vin-csi                         ;csi操作相关接口
    │   ├── sunxi_csi.c
    │   └── sunxi_csi.h
    ├── vin.h
    ├── vin-isp                         ;isp驱动
    │   ├── sunxi_isp.c
```

```
    │   └── sunxi_isp.h
    ├── vin-mipi                          ;mipi驱动
    │   ├── sunxi_mipi.c
    │   └── sunxi_mipi.h
    ├── vin-stat
    │   ├── vin_h3a.c
    │   └── vin_h3a.h
    ├── vin-tdm
    │   ├── vin_tdm.c
    │   └── vin_tdm.h
    ├── vin-video                         ;video节点相关的接口定义
    │   ├── vin_core.c
    │   ├── vin_core.h
    │   ├── vin_video.c
    │   └── vin_video.h
    └── vin-vipp
        ├── sunxi_scaler.c
        ├── sunxi_scaler.h
```

3 模块开发

3.1 模块体系结构描述

3.1.1 VFE 框架

- 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；
- 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；
- 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 ioctrl 命令的实现；而 raw sensor 的话大部分 ioctrl 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；
- actuator 文件夹内是各种 vcm 的驱动；
- flash_light 文件夹内是闪光灯控制接口实现；
- csi 和 mipi_csi 为对 csi 接口和 mipi 接口的控制文件；
- lib 文件夹为 isp 的库文件；
- linux-3.0 前的版本相当于 vivi.c+csi bsp 层
- linux-3.4 版本支持 isp 驱动和双 CSI
- linux-3.10 版本将 mipi/csi/isp 模块化（由 vfe.c 直接调用 =>v4l2_subdev_ops），支持 device tree

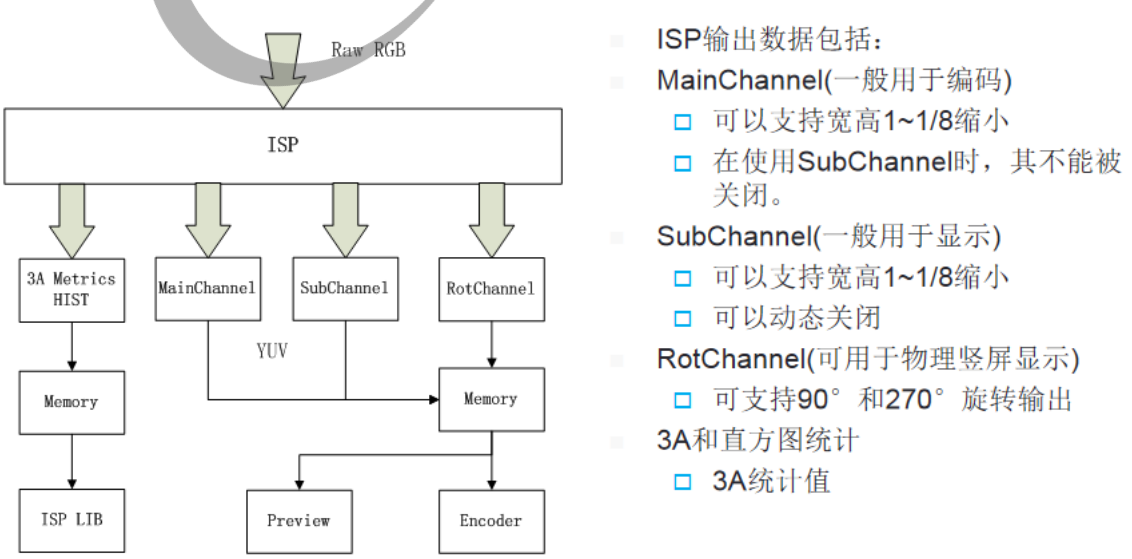

图 3-1: VFE

3.1.2 VIN 框架

- 使用过程中可简单的看成是 vin 模块 + device 模块 +af driver + flash 控制模块的方式；
- vin.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；
- modules/sensor 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 ioctrl 命令的实现；而 raw sensor 的话大部分 ioctrl 命令在 vin 层调用 isp 库实现，少数如曝光/增益调节会透过 vin 层到实际器件层；
- modules/actuator 文件夹内是各种 vcm 的驱动；
- modules/flash 文件夹内是闪光灯控制接口实现；
- vin-csi 和 vin-mipi 为对 csi 接口和 mipi 接口的控制文件；
- vin-isp 文件夹为 isp 的库操作文件；
- vin-video 文件夹内主要是 video 设备操作文件；

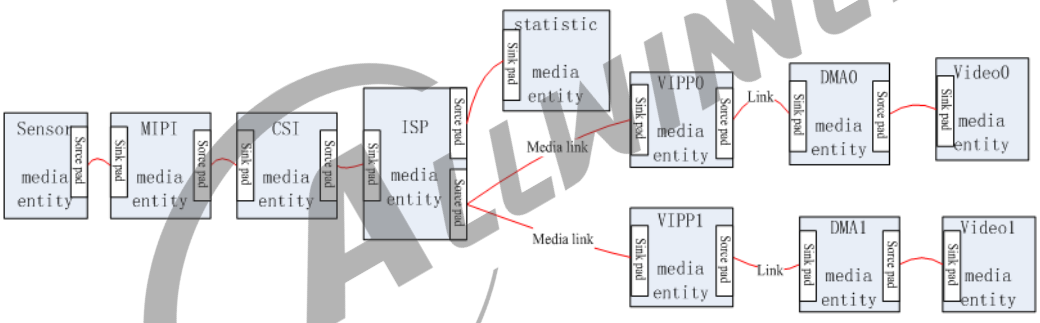

图 3-2: VIN

3.1.3 Camera 通路框架

- VIN 支持灵活配置单/双路输入双 ISP 多通路输出的规格
- 引入 media 框架实现 pipeline 管理
- 将 libisp 移植到用户空间解决 GPL 问题
- 将统计 buffer 独立为 v4l2 subdev
- 将的 scaler（vipp）模块独立为 v4l2 subdev
- 将 video buffer 修改为 mplane 方式，使用户层取图更方便
- 采用 v4l2-event 实现事件管理
- 采用 v4l2-controls 新特性

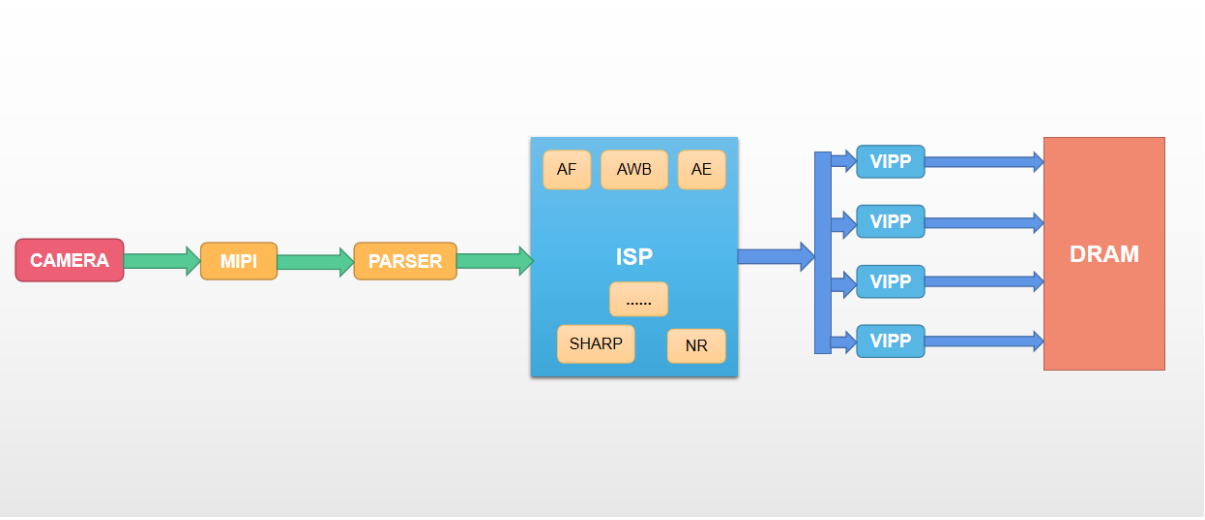

图 3-3: camera Input

3.2 驱动模块实现

3.2.1 硬件部分

检查硬件电源，gpio 是否和原理图一致并且正确连接；检查 sys_config.fex 或者 board.dts 是否正确配置，包括使用的电源名称和电压，详见 **CSI 板级配置章节**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。

3.2.2 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 的并口模组以 R40 平台 (linux3.10) 的 ov5640.c 为参考。

下面以 ov5640.c 为例说明调试新模组需要注意的两点：

1. 添加 Makefile

```
[linux-3.10/drivers/media/platform/sunxi-vfe/device/Makefile]
添加
    obj-m       + = ov5640.o    (详见1)

详注:
    1.具体取决于使用的模组，如果是新模组则将驱动代码放置在该device目录下。
```

2. 配置模组参数

配置参数在 linux-3.10/drivers/media/platform/sunxi-vfe/device/ov5640.c 中，只需注意下面两个参数。

```
#define SENSOR_NAME "ov5640"        (详见1)
#define I2C_ADDR        0x78            (详见2)

详注:
    1.该参数为模组名，必须和sys_config.fex的csi0_dev0_mname或者board.dts的sensor0_mname保持一致。
    2.I2C_ADDR可参考相应模组的datasheet，sys_config.fex的csi0_dev0_twi_addr与此值保持一致。
```

3.2.2.1 驱动宏定义

```
#define MCLK        (24*1000*1000)
```

sensor 输入时钟频率，可查看模组厂提供的 sensor datasheet，datasheet 中会有类似 `input clock frequency: 6~27 MHz` 信息，这个信息说明可提供给 sensor 的 MCLK 可以在 6 M 到 27 M 之间。其中 MCLK 和使用的寄存器配置强相关，在模组厂提供寄存器配置时，可直接询问当前配置使用的 MCLK 频率是多少。

```
#define VREF_POL    V4L2_MBUS_VSYNC_ACTIVE_LOW
#define HREF_POL    V4L2_MBUS_HSYNC_ACTIVE_HIGH
#define CLK_POL     V4L2_MBUS_PCLK_SAMPLE_RISING
```

并口 sensor 必须填写，MIPI sensor 无需填写，可在 sensor 规格书找到，如下

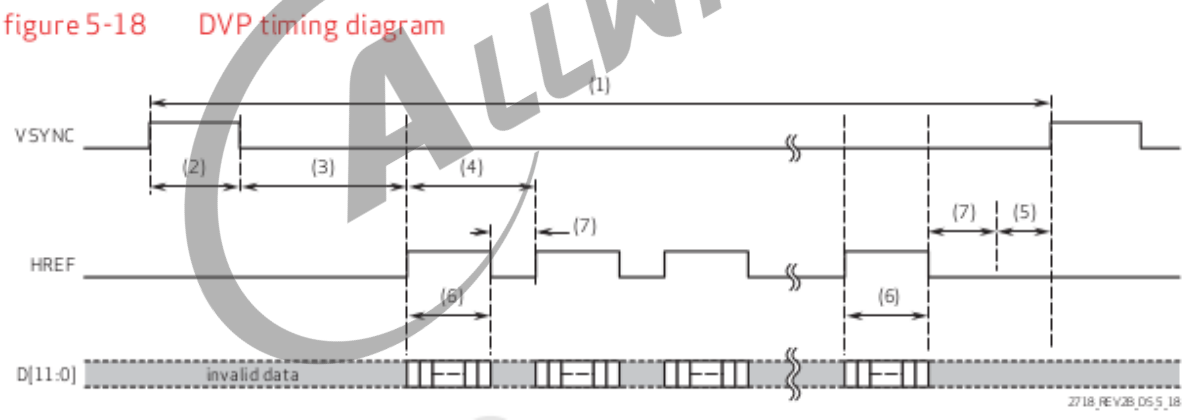

图 3-4: timing

从上述的图像可得到以下信息：

1. VSYNC 在低电平的时候，data pin 输出有效数据，所以 VREF_POL 设置为 V4L2_MBUS_VSYNC_ACTIV 即低电平有效；
2. HREF 在高电平的时候，data pin 输出有效数据，所以 HREF_POL 设置为 V4L2_MBUS_HSYNC_ACTIV 即高电平有效；
3. CLK_POL 则是表明 SOC 是在 sensor 输出的 pclk 上升沿采集 data pin 的数据还是下降沿采集数据，如果 sensor 在 pclk 上升沿改变 data pin 的数据，那么 SOC 应该在

下降沿采集，CLK_POL 设置为 V4L2_MBUS_PCLK_SAMPLE_FALLING；如果 sensor 在 pclk 下降沿改变 data pin 的数据，那么 SOC 应该在上降沿采集，CLK_POL 设置为 V4L2_MBUS_PCLK_SAMPLE_RISING。

```
#define V4L2_IDENT_SENSOR 0x2770
```

一般填写 sensor ID，用于 sensor 检测。sensor ID 可在 sensor 规格书的找到，如下

0x300A	CHIP_ID_H	0x27	R	Chip ID
0x300B	CHIP_ID_L	0x70	R	Chip ID

图 3-5: sensorid

```
#define I2C_ADDR 0x6c
```

sensor I2C 通讯地址，可在 sensor 规格书找到，如下

table A-2 sensor control registers (sheet 2 of 42)

address	register name	default value	R/W	description
0x300C	SCCB_ID	0x6C	RW	Bit[7:1]: sccb_id_n Bit[0]: sccb_id_sel

图 3-6: sccbid

```
#define SENSOR_NAME OV5640
```

定义驱动名字，与系统其他文件填写的名字要一致，比如需要和 sys_config.fex 中的 sensor name 一致。

3.2.2.2 初始化代码

```
static struct regval_list sensor_default_regs[] = {}; /* 填写寄存器代码的公共部分 */

static struct regval_list sensor_XXX_regs[] = {}; /* 填写各模式的寄存器代码，不同的模式可以是分辨
    率、帧率等 */
```

上述部分的寄存器配置，公共部分可以忽略，直接在模式代码中配置 sensor 即可，相应的寄存器配置，可让模组厂提供。

3.2.2.3 曝光增益接口函数

```
static int sensor_s_exp(struct v4l2_subdev *sd, unsigned int exp_val)   /* 曝光函数 */
static int sensor_s_gain(struct v4l2_subdev *sd, unsigned int gain_val) /* 增益函数 */
```

AE 是同时控制曝光时间和增益的，所以需要在上面的函数中分别同时 sensor 曝光和增益的寄存器。

table 5-3 manual exposure and gain control registers

address	register name	default value	R/W	description
0x3500	EXPOSURE	0x00	RW	Bit[3:0]: Exposure[19:16]
0x3501	EXPOSURE	0x02	RW	Bit[7:0]: Exposure[15:8]
0x3502	EXPOSURE	0x00	RW	Bit[7:0]: Exposure[7:0]
0x3503	MANUAL CONTROL	0x00	RW	Bit[1]: Gain manual enable Bit[0]: Exposure manual enable
0x350B	GAIN	0x10	RW	Bit[7:0]: Gain[7:0]

图 3-7: expgain

根据规格书中的寄存器说明，在相应的函数配置即可。若设置 exp/gain 无效，可能的原因有：

- sensor 寄存器打开了 AE；
- 设置值超出了有效范围

具体可根据模组厂提供的配置设置，如若检查之后设置仍失效，可与模组厂沟通，确认配置是否正确。

3.2.2.4 上下电控制函数

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

控制 sensor 上电、下电及进出待机状态，操作步骤须与规格书描述相同，注意 power down 和 reset pin 的电平变化。

2.5.1 power up sequence

figure 2-6 power up sequence diagram

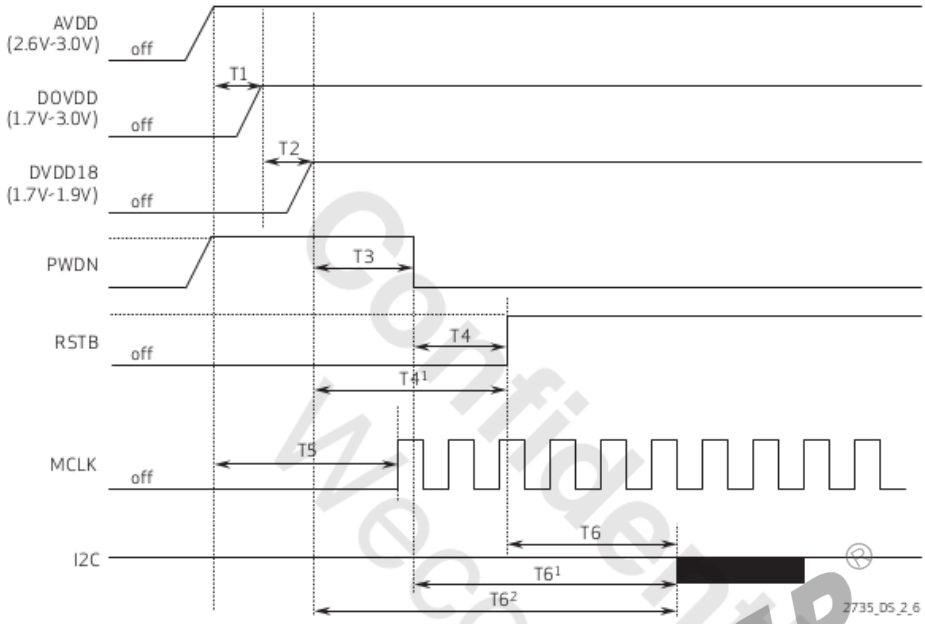

table 2-4 power up sequence parameters

symbol	description	min	unit
T1	delay from AVDD to DOVDD	0	ms
T2	delay from DOVDD to DVDD18	0	ms
T3	delay from DVDD18 stable to sensor power up stable	5	ms
T4	delay from PWDN pulling low to RSTB pulling high	4	ms
$T4^1$	delay from DVDD18 power up stable to RSTB pulling high when PWDN signal remains low during power up	9	ms
T5	delay from AVDD stable to MCLK on	0	ms
T6	delay from RSTB pulling high to first I2C command	5	ms
$T6^1$	delay from PWDN pulling low to first I2C command when RSTB signal remains high during power up	9	ms
$T6^2$	delay from DVDD18 power up stable to first I2C command when PWDN signal remains low and RSTB signal remains high during power up	14	ms

图 3-8: powerup

驱动中，按照规格书的上电时序进行配置，而如果上电之后测量硬件并没有相应的电压，这时候检查硬件和软件配置是否一致。关于 csi 电源的配置，操作流程可如下：

1. 先通过原理图确认 sensor 模组的各路电源是连接到 axp 的哪个 ldo;
2. 查看 sys_config.fex 的 regulator 配置，在相应的 ldo 后增加相应的字段，比如 “csi-vdd”

等；

3. 在 sys_config.fex 的 csi 部分，sensor 部分的电源后的字段再填写与上述一样的字段即可；
4. 根据 sensor 规格书的要求，填写相应的电压即可；

以上图为例，确认 sensor 驱动中的上电时序。

```
static int sensor_power(struct v4l2_subdev *sd, int on)
{
    int ret;

    ret = 0;

    switch (on) {

    /* STBY_ON 和 STBY_OFF 基本不使用，可忽略这两个选项的配置 */
    case STBY_ON:
        ...
        break;

    case STBY_OFF:
        ...
        break;

    /* 上电操作 */
    case PWR_ON:
        sensor_print("PWR_ON!\n");

        cci_lock(sd);

        /* 将 PWDN、RESET 引脚设置为输出 */
        vin_gpio_set_status(sd, PWDN, 1);
        vin_gpio_set_status(sd, RESET, 1);

        /* 按照上图知道，上电前 PWDN、RESET 信号为低，所以将其设置为低电平 */
        vin_gpio_write(sd, PWDN, CSI_GPIO_LOW);
        vin_gpio_write(sd, RESET, CSI_GPIO_LOW);

        /* 延时 */
        usleep_range(1000, 1200);

        /* CAMERAVDD 为 SOC 中的供电电源，部分板子可以忽略该电源，
         * 因为有些板子会通过一个 vcc-pe 给上拉电阻等供电，所以需要
         * 使能该路电，有些是直接和 iovdd 共用了，所以有部分会忽略该
         * 路电源配置.
         */
        vin_set_pmu_channel(sd, CAMERAVDD, ON);

        /* 将 PWDN 设置为高电平 */
        vin_gpio_write(sd, PWDN, CSI_GPIO_HIGH);

        /*AF上电*/
        vin_set_pmu_channel(sd, AFVDD, ON);

        /* AVDD 上电 */
        vin_set_pmu_channel(sd, AVDD, ON);
        /* 延时，延时时长为 T1，T1 的大小在 datasheet 的上电时序图下面有标注 */
        usleep_range(1000, 1200);

        /* DOVDD 上电 */
```

```
        vin_set_pmu_channel(sd, IOVDD, ON);
        /* 延时，按照上电时序中的标注的 T2 时间延时 */
        usleep_range(1000, 1200);

        /* DVDD 上电 */
        vin_set_pmu_channel(sd, DVDD, ON);
        /* 延时，按照上电时序中的标注的 T3 时间延时 */
        usleep_range(1000, 1200);

        /* 将 PWDN 设置为低电平 */
        vin_gpio_write(sd, PWDN, CSI_GPIO_LOW);
        /* 设置 MCLK 频率并使能 */
        vin_set_mclk_freq(sd, MCLK);
        vin_set_mclk(sd, ON);
        /* 延时，按照上电时序中的标注的 T4 时间延时 */
        usleep_range(1000, 1200);

        /* 将 RESET 设置为高电平 */
        vin_gpio_write(sd, RESET, CSI_GPIO_HIGH);
        /* 延时，按照上电时序中的标注的 T6 时间延时 */
        usleep_range(10000, 12000);

        cci_unlock(sd);
        break;

    /* 掉电操作 */
    case PWR_OFF:
        sensor_print("PWR_OFF!\n");
        cci_lock(sd);

        /* 具体的掉电操作同样的按照 datasheet 的 power off 操作即可 */
        vin_gpio_write(sd, PWDN, CSI_GPIO_HIGH);
        vin_gpio_write(sd, RESET, CSI_GPIO_LOW);

        vin_set_mclk(sd, OFF);
        vin_set_pmu_channel(sd, AFVDD, OFF);
        vin_set_pmu_channel(sd, AVDD, OFF);
        vin_set_pmu_channel(sd, DVDD, OFF);
        vin_set_pmu_channel(sd, IOVDD, OFF);
        vin_set_pmu_channel(sd, CAMERAVDD, OFF);

        vin_gpio_set_status(sd, PWDN, 0);

        cci_unlock(sd);
        break;
    default:
        return -EINVAL;
    }

    return 0;
}
```

3.2.2.5 检测函数

```
static int sensor_detect(struct v4l2_subdev *sd)
```

在开机加载驱动的时候，将会检测 sensor ID，用于测试 I2C 通讯是否正常和 sensor 识别。

```
#define V4L2_IDENT_SENSOR 0x7750

    sensor_read(sd, 0x300A, &rdval);
    if (rdval != (V4L2_IDENT_SENSOR >> 8))
        return -ENODEV;

    sensor_read(sd, 0x300B, &rdval);
    if (rdval != (V4L2_IDENT_SENSOR & 0xff))
        return -ENODEV;
```

0x300A	SC_CHIP_ID_HIGH	0x77	R	Chip ID High Byte
0x300B	SC_CHIP_ID_LOW	0x50	R	Chip ID Low Byte

图 3-9: sensordetect

3.2.2.6 SENSOR 相关的 IOCTL

```
static long sensor_ioctl(struct v4l2_subdev *sd, unsigned int cmd, void *arg)
```

用于应用层获取曝光增益，及进行与 sensor 相关模块的驱动控制，如对焦，闪光等

```
case VIDIOC_VIN_SENSOR_EXP_GAIN:/*设置sensor的曝光增益*/
                ret = sensor_s_exp_gain(sd, (struct sensor_exp_gain *)arg);
                break;
case VIDIOC_VIN_SENSOR_CFG_REQ:/*获取sensor驱动的基础配置信息*/
                sensor_cfg_req(sd, (struct sensor_config *)arg);
                break;
case VIDIOC_VIN_ACT_SET_CODE:/*设置对焦马达配置参数，在配置AF模块时，需要此ioctl*/
                actuator_set_code(sd, (struct actuator_ctrl *)arg);
                break;
```

3.2.2.7 与 CSI 的接口

```
static struct sensor_format_struct sensor_formats[] = {};

RAW sensor:
        .desc = "Raw RGB Bayer",
        .mbus_code = MEDIA_BUS_FMT_SGRBG10_1X10,
        .regs = sensor_fmt_raw,
        .regs_size = ARRAY_SIZE(sensor_fmt_raw),
        .bpp = 1

YUV sensor:
        .desc = "YUYV 4:2:2",
        .mbus_code = MEDIA_BUS_FMT_YUYV8_2X8,
        .regs = sensor_fmt_yuyv422_yuyv,
        .regs_size = ARRAY_SIZE(sensor_fmt_yuyv422_yuyv),
        .bpp = 2
```

其中，mbus_code 中 BGGR 可以根据 sensor raw data 输出顺序修改为 GBRG/RGGB/-GRBG。若填错，会导致色彩偏紫红和出现网格状纹理。10_1X10 表示 10 bit 并口输出，若是

12 bit MIPI 输出, 则改为 12_12X1。其他情况类推。对于 DVP YUV sensor, 需根据 yuv 输出顺序选择 yuyv/vyuy/uyvy/yvyu 其中一种。

```
static int sensor_g_mbus_config(struct v4l2_subdev *sd,
                struct v4l2_mbus_config *cfg)

DVP sensor:
    cfg->type = V4L2_MBUS_PARALLEL;
    cfg->flags = V4L2_MBUS_MASTER | VREF_POL | HREF_POL | CLK_POL;

MIPI sensor:
    cfg->type = V4L2_MBUS_CSI2;
    cfg->flags = 0 | V4L2_MBUS_CSI2_1_LANE | V4L2_MBUS_CSI2_CHANNEL_0;
```

其中，MIPI sensor 须根据实际使用的 lane 数，修改 V4L2_MBUS_CSI2_X_LANE 中的 X 值。如果使用 LVDS 接口，需要将 cfg->type 配置为 V4L2_MBUS_SUBLVDS。

3.2.2.8 分辨率配置

```
static struct sensor_win_size sensor_win_sizes[] = {
    {
        .width      = VGA_WIDTH,
        .height     = VGA_HEIGHT,
        .hoffset    = 0,
        .voffset    = 0,
        .hts        = 928,
        .vts        = 1720,
        .pclk       = 48 * 1000 * 1000,
        .mipi_bps   = 480 * 1000 * 1000,
        .fps_fixed  = 30,
        .bin_factor = 1,
        .intg_min   = 1 << 4,
        .intg_max   = (1720) << 4,
        .gain_min   = 1 << 4,
        .gain_max   = 16 << 4,
        .regs       = sensor_VGA_regs,
        .regs_size  = ARRAY_SIZE(sensor_VGA_regs),
        .set_size   = NULL,
    },
    {
        /* 定义图像输出的大小 */
        .width      = VGA_WIDTH,
        .height     = VGA_HEIGHT,

        /* 定义输入 ISP 的偏移量，用于截取所需的Size，丢弃不需要的部分图像 */
        .hoffset    = 0,
        .voffset    = 0,

        /*
          定义行长(以 pclk 为单位)、帧长(以 hts 为单位) 和像素时钟频率。hts 又称 line_length_pck,
          vts 又称 frame_length_lines，与寄存器的值要一致。pclk(pixel clock)的值由 PLL 寄存器计算得
    出。
        */
        .hts        = 928,
        .vts        = 1720,
        .pclk       = 48 * 1000 * 1000,
```

```
        /* 定义 MIPI 数据速率,MIPI sensor 必需，其他 sensor 忽略 */
        /* mipi_bps = hts * vts * fps * raw bit / lane num */
        .mipi_bps   = 480 * 1000 * 1000,

        /* 定义帧率，fps * hts * vts = pclk */
        .fps_fixed  = 30,

        /*
          定义曝光行数最小值和最大值,增益最小值和最大值,以 16 为 1 倍。最值的设置应在 sensor 规格和
          曝光函数限定的范围内,若超出会导致画面异常。此外,若 AE table 中的最值超出这里的限制,会使得
          AE table 失效。
        */
        .intg_min   = 1 << 4,
        .intg_max   = (1720) << 4,
        .gain_min   = 1 << 4,
        .gain_max   = 16 << 4,

        /* (必需)说明这部分的配置对应哪个寄存器初始化代码 */
        .regs       = sensor_VGA_regs,
        .regs_size  = ARRAY_SIZE(sensor_VGA_regs),
    },
};
```

根据应用的需求，在这里配置驱动能输出的不同尺寸帧率组合，注意，一种分辨率、帧率配置为一个数组成员，不要混淆。

3.2.3 LVDS 接口须知

除了完成以上函数的实现，LVDS Sensor 驱动还需要完成 combo 同步校验函数和 combo 数据线映射函数。combo 校验码可以在 sensor 规格书获取，combo 数据线映射关系需要查看原理图设计进行配对，可参考 imx274_slvds.c 完成开发。

Sync code details

LVDS output bit No.		Sync code (4 words)				
12-bit output	10-bit output	1st word	2nd word	3rd word	4th word	
11	9	1	0	0	1	
10	8	1	0	0	0	
9	7	1	0	0	V	1: Blanking line 0: Except blanking line
8	6	1	0	0	H	1: End sync code 0: Start sync code
7	5	1	0	0	P3	Protection bits
6	4	1	0	0	P2	
5	3	1	0	0	P1	
4	2	1	0	0	P0	
3	1	1	0	0	0	
2	0	1	0	0	0	
1	—	1	0	0	0	
0	—	1	0	0	0	

		Protection bits			
V	H	P3	P2	P1	P0
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

Sync code details (hexadecimal notation) 12-bit output

		1st word	2nd word	3rd word	4th word
Blanking line	Start sync code (SAV)	FFFh	000h	000h	AB0h
	End sync code (EAV)				B60h
Except blanking line	Start sync code (SAV)				800h
	End sync code (EAV)				9D0h

Sync code details (hexadecimal notation) 10-bit output

		1st word	2nd word	3rd word	4th word
Blanking line	Start sync code (SAV)	3FFh	000h	000h	2ACh
	End sync code (EAV)				2D8h
Except blanking line	Start sync code (SAV)				200h
	End sync code (EAV)				274h

图 3-10: SYNC_CODE

```
static void sensor_g_combo_sync_code(struct v4l2_subdev *sd,
                struct combo_sync_code *sync)
{
    int i;

    for (i = 0; i < 12; i++) {
        sync->lane_sof[i].low_bit = 0x0000ab00;
        sync->lane_sof[i].high_bit = 0xFFFF0000;
        sync->lane_sol[i].low_bit = 0x00008000;
        sync->lane_sol[i].high_bit = 0xFFFF0000;
        sync->lane_eol[i].low_bit = 0x00009d00;
        sync->lane_eol[i].high_bit = 0xFFFF0000;
        sync->lane_eof[i].low_bit = 0x0000b600;
        sync->lane_eof[i].high_bit = 0xFFFF0000;
    }
}

static void sensor_g_combo_lane_map(struct v4l2_subdev *sd,
                struct combo_lane_map *map)
```

```
{
    struct sensor_info *info = to_state(sd);

    if (info->isp_wdr_mode == ISP_DOL_WDR_MODE) {
        map->lvds_lane0 = LVDS_MAPPING_A_D0_TO_LANE0;
        map->lvds_lane1 = LVDS_MAPPING_A_D1_TO_LANE1;
        map->lvds_lane2 = LVDS_MAPPING_B_D2_TO_LANE2;
        map->lvds_lane3 = LVDS_MAPPING_B_D0_TO_LANE3;
        map->lvds_lane4 = LVDS_MAPPING_B_D3_TO_LANE4;
        map->lvds_lane5 = LVDS_MAPPING_C_D2_TO_LANE5;
        map->lvds_lane6 = LVDS_LANE6_NO_USE;
        map->lvds_lane7 = LVDS_LANE7_NO_USE;
        map->lvds_lane8 = LVDS_LANE8_NO_USE;
        map->lvds_lane9 = LVDS_LANE9_NO_USE;
        map->lvds_lane10 = LVDS_LANE10_NO_USE;
        map->lvds_lane11 = LVDS_LANE11_NO_USE;
    } else {
        map->lvds_lane0 = LVDS_MAPPING_A_D1_TO_LANE0;
        map->lvds_lane1 = LVDS_MAPPING_B_D2_TO_LANE1;
        map->lvds_lane2 = LVDS_MAPPING_B_D0_TO_LANE2;
        map->lvds_lane3 = LVDS_MAPPING_B_D3_TO_LANE3;
        map->lvds_lane4 = LVDS_LANE4_NO_USE;
        map->lvds_lane5 = LVDS_LANE5_NO_USE;
        map->lvds_lane6 = LVDS_LANE6_NO_USE;
        map->lvds_lane7 = LVDS_LANE7_NO_USE;
        map->lvds_lane8 = LVDS_LANE8_NO_USE;
        map->lvds_lane9 = LVDS_LANE9_NO_USE;
        map->lvds_lane10 = LVDS_LANE10_NO_USE;
        map->lvds_lane11 = LVDS_LANE11_NO_USE;

    }
}
```

3.2.4 内核代码注意事项

驱动中一般禁止使用 mdelay 或者 msleep 实现延时，例如使用 msleep 实现 10~20ms 的延时，通常会因为系统调度而变成延时更长的时间，这种做法精度较差。所以如果需要使用 ms 级别延时，则使用 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 或以上的延时可选择使用 msleep()；

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 模块配置

4.1 Tina 配置

Tina 中主要是修改平台的 modules.mk 配置，modules.mk 主要完成两个方面：

1. 拷贝相关的 ko 模块到小机 rootfs 中
2. rootfs 启动时，按顺序自动加载相关的 ko 模块。

由于内核框架的不一样，需要区分 vfe 和 vin 进行相应的配置。

4.1.1 vfe 框架

modules.mk 配置路径 (以 R40 平台的为例)：

```
target/allwinner/r40-common/modules.mk
```

其中的 r40-common 为 R40 平台共有的配置文件目录，相应的修改对应平台的 modules.mk 即可。

```
  define KernelPackage/sunxi-vfe
    SUBMENU:=$(VIDEO_MENU)
    TITLE:=sunxi-vfe support
    FILES:=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-core.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-memops.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-dma-contig.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-v4l2.ko
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vfe/vfe_io.ko
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vfe/device/ov5640.ko    (详见1)
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vfe/vfe_v4l2.ko
    AUTOLOAD:=$(call AutoLoad,90,videobuf2-core videobuf2-memops videobuf2-dma-contig
    videobuf2-v4l2 vfe_io ov5640 vfe_v4l2)        (详见2)
  endef

  define KernelPackage/sunxi-vfe/description
    Kernel modules for sunxi-vfe support
  endef

详注:
    1.由具体使用的模组确定，需要确定内核路径中这个驱动是否被编译出来。
    2.AUTOLOAD为小机rootfs挂载后自动加载的机制，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。必须修改相应的sensor ko才会开启自加载。
```

4.1.2 vin 框架

modules.mk 配置路径 (以 R30 平台的为例)：

```
target/allwinner/r30-common/modules.mk
```

其中的 r30-common 为 R30 平台共有的配置文件目录，相应的修改对应平台的 modules.mk 即可。

```
define KernelPackage/sunxi-vin
    SUBMENU:=$(VIDEO_MENU)
    TITLE:=sunxi-vin support
    FILES:=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-core.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-memops.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-dma-contig.ko
    FILES+=$(LINUX_DIR)/drivers/media/v4l2-core/videobuf2-v4l2.ko
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vin/vin_io.ko
    /*对焦马达驱动加载*/
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vin/modules/actuator/actuator.ko
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vin/modules/actuator/dw9714_act.ko(详见
    3)
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vin/modules/sensor/ov5640.ko  (详见1)
    FILES+=$(LINUX_DIR)/drivers/media/platform/sunxi-vin/vin_v4l2.ko
    AUTOLOAD:=$(call AutoLoad,90,videobuf2-core videobuf2-memops videobuf2-dma-contig
    videobuf2-v4l2 vin_io actuator dw9714_act ov5640 vin_v4l2)                    (详见2)
  endef

  define KernelPackage/sunxi-vin/description
    Kernel modules for sunxi-vin support
  endef

详注:
    1.由具体使用的模组确定，需要确定内核路径中这个驱动是否被编译出来。
    2.AUTOLOAD为小机rootfs挂载后自动加载的机制，vin_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。
    3.对焦马达驱动加载顺序必须在sensor驱动加载之前，具体驱动型号根据模组规格书进行确认。
```

V 系列平台在完成 modules.mk 配置后，还需要完成.ko 挂载脚本 S00mpp 的配置，S00mpp 配置路径（以 V853 平台为例）：

```
target/allwinner/v853-perf1/busybox-init-base-files/etc/init.d
```

其中的 v853-perf1 为 V 系列平台共有的配置文件目录，相应的修改对应平台的 S00mpp 即可。

```
#!/bin/sh
#
# Load mpp modules....
#

MODULES_DIR="/lib/modules/`uname -r`"

start() {
    printf "Load mpp modules\n"
    insmod $MODULES_DIR/videobuf2-core.ko
    insmod $MODULES_DIR/videobuf2-memops.ko
    insmod $MODULES_DIR/videobuf2-dma-contig.ko
    insmod $MODULES_DIR/videobuf2-v4l2.ko
    insmod $MODULES_DIR/vin_io.ko
```

```
#    insmod $MODULES_DIR/sensor_power.ko
     insmod $MODULES_DIR/gc4663_mipi.ko
     insmod $MODULES_DIR/vin_v4l2.ko
     insmod $MODULES_DIR/sunxi_aio.ko
     insmod $MODULES_DIR/sunxi_eise.ko
#    insmod $MODULES_DIR/vipcore.ko
}

stop() {
     printf "Unload mpp modules\n"
#    rmmod $MODULES_DIR/vipcore.ko
     rmmod $MODULES_DIR/sunxi_eise.ko
     rmmod $MODULES_DIR/sunxi_aio.ko
     rmmod $MODULES_DIR/vin_v4l2.ko
     rmmod $MODULES_DIR/gc4663_mipi.ko
#    rmmod $MODULES_DIR/sensor_power.ko
     rmmod $MODULES_DIR/vin_io.ko
     rmmod $MODULES_DIR/videobuf2-v4l2.ko
     rmmod $MODULES_DIR/videobuf2-dma-contig.ko
     rmmod $MODULES_DIR/videobuf2-memops.ko
     rmmod $MODULES_DIR/videobuf2-core.ko
}

case "$1" in
     start)
     start
     ;;
     stop)
     stop
     ;;
     restart|reload)
     stop
     start
     ;;
  *)
     echo "Usage: $0 {start|stop|restart}"
     exit 1
esac

exit $?
```

4.2 CSI 板级配置

Tina 平台根据不同的平台差异分别使用 sys_config.fex 或 board.dst 配置 camera CSI，具体的对应关系如下表，下面将分别介绍 sys_config.fex 和 board.dts 中关于 camera CSI 配置。

表 4-1: 平台配置方式对应表

平台	CSI 使用的配置方式
F35	sys_config.fex
R16	sys_config.fex
R18	sys_config.fex
R30	sys_config.fex

平台	CSI 使用的配置方式
R40	sys_config.fex
R311	sys_config.fex
MR133	sys_config.fex
R818	board.dts
MR813	board.dts
R528	board.dts
V536	sys_config.fex
V533	board.dts
V831	board.dts
V833	board.dts
V851	board.dts
V853	board.dts

4.2.1 sys_config.fex 平台配置

sys_config.fex 配置 camera CSI，CSI sys_config.fex 部分对应的字段为：[csi0]。

通过举例 R40 平台说明在实际使用中应该如何配置：假如使用一个并口 camera 模组需要配置 [csi0] 的公用部分和 [csi0] 的 vip_dev0_(x) 部分，另外 [csi0] 中 vip_used 设置为 1，[csi1] 中 vip_used 设置为 0。

下面给出一个 ov5640 模组的参考配置：其中 [csi0] 为并口的配置。具体填写方法请参照以下说明：

```
/* 下面部分的CSI配置适用4.9内核之前的平台 */
;----------------------------------------------------------------------------
;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<2><default><default><default>
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"          ;必须和sensor驱动中的SENSOR_NAME一致
csi0_dev0_twi_addr          = 0x78              ; 请参考实际模组的8bit ID填写
csi0_dev0_twi_id            = 2
csi0_dev0_pos               = "rear"
csi0_dev0_isp_used          = 0                 ; YUV格式 填0，RAW格式 填1
csi0_dev0_fmt               = 0                 ; YUV格式 填0，RAW格式 填1
csi0_dev0_stby_mode         = 0
csi0_dev0_vflip             = 0
csi0_dev0_hflip             = 0
csi0_dev0_iovdd             = "csi-iovcc"               ; 电源请参考实际原理图填写，同时参考
    sys_config.fex 的 regulator 配置，确认该字段有效
csi0_dev0_iovdd_vol         = 2800000                   ; 电压值参考datasheet
csi0_dev0_avdd              = "csi-avdd"                ; 电源请参考实际原理图填写，同时参考
    sys_config.fex 的 regulator 配置，确认该字段有效
csi0_dev0_avdd_vol          = 2800000                   ; 电压值参考datasheet
csi0_dev0_dvdd              = "csi-dvdd"                ; 电源请参考实际原理图填写，同时参考
    sys_config.fex 的 regulator 配置，确认该字段有效
csi0_dev0_dvdd_vol          = 1500000                   ; 电压值参考datasheet
csi0_dev0_afvdd             = "csi-afvcc"               ; 电源请参考实际原理图填写，同时参考
    sys_config.fex 的 regulator 配置，确认该字段有效
csi0_dev0_afvdd_vol         = 2800000                   ; 电压值参考datasheet
csi0_dev0_power_en          =
csi0_dev0_reset             = port:PE14<1><0><1><0> ; io选取参照实际原理图
csi0_dev0_pwdn              = port:PE15<1><0><1><0> ; io选取参照实际原理图
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

/* 下面部分的CSI配置适用4.9内核平台 */
;------------------------------------------------------------------------------
;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
;------------------------------------------------------------------------------
[vind0]
vind0_used      = 1

[vind0/csi_cci0]
csi_cci0_used           = 1 ;配置是否使用CCI，如果使用CCI，需要使能该配置并配置下面的CCI引脚
csi_cci0_sck            = port:PE01<2><default><default><default>
csi_cci0_sda            = port:PE02<2><default><default><default>

[vind0/flash0]
flash0_used     = 0
flash0_type     = 2
flash0_en       =
flash0_mode     =
flash0_flvdd        = ""
flash0_flvdd_vol    =

[vind0/actuator0]
actuator0_used      = 0
actuator0_name      = "ad5820_act"
actuator0_slave     = 0x18
actuator0_af_pwdn       =
actuator0_afvdd         = "afvcc-csi"
actuator0_afvdd_vol     = 2800000

[vind0/sensor0]
sensor0_used          = 0
sensor0_mname         = "gc8034_mipi"
sensor0_twi_cci_id    = 0
sensor0_twi_addr      = 0x6e
sensor0_pos           = "rear"
sensor0_isp_used      = 1
sensor0_fmt           = 1
sensor0_stby_mode     = 0
sensor0_vflip         = 0
sensor0_hflip         = 0
sensor0_cameravdd     = ""
sensor0_cameravdd_vol = 3300000
sensor0_iovdd         = "iovdd-csi"
sensor0_iovdd_vol     = 1800000
```

```
sensor0_avdd          = "avdd-csi-f"
sensor0_avdd_vol      = 2800000
sensor0_dvdd          = "dvdd-csi"
sensor0_dvdd_vol      = 1200000
sensor0_power_en      =
sensor0_reset         = port:PE06<0><0><1><0>
sensor0_pwdn          = port:PE05<0><0><1><0>

[vind0/sensor1]
sensor1_used          = 1
sensor1_mname         = "gc8034_mipi" ;必须要和驱动的 SENSOR_NAME 一致
sensor1_twi_cci_id    = 0             ;配置使用的TWI id，如果使用TWI，则不使用CCI
sensor1_twi_addr      = 0x6e          ;配置sensor的i2c地址
sensor1_pos           = "front"
sensor1_isp_used      = 1             ;配置是否使用isp
sensor1_fmt           = 1
sensor1_stby_mode     = 0
sensor1_vflip         = 0
sensor1_hflip         = 0
sensor1_cameravdd     = ""
sensor1_cameravdd_vol = 3300000
sensor1_iovdd         = "iovdd-csi"
sensor1_iovdd_vol     = 1800000
sensor1_avdd          = "avdd-csi-f"
sensor1_avdd_vol      = 2800000
sensor1_dvdd          = "dvdd-csi"
sensor1_dvdd_vol      = 1200000
sensor1_power_en      =
sensor1_reset         = port:PE06<0><0><1><0>
sensor1_pwdn          = port:PE05<0><0><1><0>

[vind0/vinc0]                     ;配置 video0 的数据链路
vinc0_used      = 1
vinc0_csi_sel       = 0
vinc0_mipi_sel      = 0
vinc0_isp_sel       = 0
vinc0_rear_sensor_sel   = 1       ;配置使用 sensor1 输出图像数据到 video0
vinc0_front_sensor_sel  = 1       ;配置使用 sensor1 输出图像数据到 video0
vinc0_sensor_list   = 0

[vind0/vinc1]
vinc1_used      = 0
vinc1_csi_sel       = 0
vinc1_mipi_sel      = 0
vinc1_isp_sel       = 0
vinc1_rear_sensor_sel   = 1
vinc1_front_sensor_sel  = 1
vinc1_sensor_list   = 0
```

关于电源的配置，根据板子的原理图，了解需要 sensor 驱动配置哪几路电，然后在 sys_config.fex 中进行配置：比如说 sensor0 有个“CSI-IOVCC”连接到 AXP 的“LDO4”，那么，在 sys_config.fex 中搜索 LDO4，然后在其后面增加“csi-iovcc”，这样，在 sensor 端就可以使用该标号配置 sensor0_iovdd。

```
regulator14     = "pmu1736_bldo2 none csi-iovdd"

sensor0_iovdd         = "csi-iovdd"
```

同时关于 mr133/R311 平台，sys_config.fex 中的 vinc0_rear_sensor_sel 和 vinc0_front_sensor_sel

配置决定着使用哪路 sensor 输入数据，该配置与硬件连接相关，可参考本文档最后的**其他注意事项**章节。

4.2.2 board.dts 平台配置

当前 MR813/R818/R528 平台的摄像头配置不再使用 sys_config.fex 而使用 board.dts，文件存放在 tina/device/config/chips/mr813(R818、R528)/configs/< 方案 > 目录下，摄像头相关的配置如下：

```
        vind0:vind@0 {
            vind0_clk = <336000000>;
            vind0_isp = <327000000>;
            status = "okay";

            actuator0:actuator@0 {
                device_type = "actuator0";
                actuator0_name = "ad5820_act";/*必须要和驱动的SUNXI_ACT_NAME一致*/
                actuator0_slave = <0x18>;/*必须和驱动的SUNXI_ACT_ID一致*/
                actuator0_af_pwdn = <>;
                actuator0_afvdd = "afvcc-csi";
                actuator0_afvdd_vol = <2800000>;/*af模块的配电不在此处，在sensor配置中*/
                status = "disabled";/*使能开关，当使用AF功能时，status = "okay"*/
            };
            flash0:flash@0 {
                device_type = "flash0";
                flash0_type = <2>;
                flash0_en = <>;
                flash0_mode = <>;
                flash0_flvdd = "";
                flash0_flvdd_vol = <>;
                device_id = <0>;
                status = "disabled";
            };
            sensor0:sensor@0 {
                device_type = "sensor0";
                sensor0_mname = "imx278_mipi"; /* 必须要和驱动的 SENSOR_NAME 一致 */
                sensor0_twi_cci_id = <2>;
                sensor0_twi_addr = <0x20>;
                sensor0_mclk_id = <0>;
                sensor0_pos = "rear";
                sensor0_isp_used = <1>; /* R528 没有isp，该项需要配置为0 */
                sensor0_fmt = <1>;
                sensor0_stby_mode = <0>;
                sensor0_vflip = <0>;
                sensor0_hflip = <0>;
                /* sensor iovdd 连接的 ldo，根据硬件原理图的连接，
                 * 确认是连接到 axp 哪个 ldo，假设 iovdd 连接到 aldo3，
                 * 则 sensor0_iovdd-supply = <&reg_aldo3>，其他同理。
                 */
                sensor0_iovdd-supply = <&reg_dldo2>;
                sensor0_iovdd_vol = <1800000>;
                sensor0_avdd-supply = <&reg_dldo3>;
                sensor0_avdd_vol = <2800000>;
                sensor0_dvdd-supply = <&reg_eldo2>;
                sensor0_afvdd-supply = <&reg_aldo3>;/*根据硬件原理图，确定配的哪路电*/
                sensor0_afvdd_vol = <2800000>;/*根据硬件原理图，确认工作电压*/
```

```
            sensor0_dvdd_vol = <1200000>;
            sensor0_power_en = <>;
            /* 根据板子实际连接，修改 reset、pwdn 的引脚即可 */
            /* GPIO 信息配置：pio 端口 组内序号 功能分配 内部电阻状态 驱动能力 输出电平状态 */
            sensor0_reset = <&pio PE 9 1 0 1 0>;
            sensor0_pwdn = <&pio PE 8 1 0 1 0>;
            status  = "okay";
        };
        sensor1:sensor@1 {
            device_type = "sensor1";
            sensor1_mname = "imx386_mipi";
            sensor1_twi_cci_id = <3>;
            sensor1_twi_addr = <0x20>;
            sensor1_mclk_id = <1>;
            sensor1_pos = "front";
            sensor1_isp_used = <1>;
            sensor1_fmt = <1>;
            sensor1_stby_mode = <0>;
            sensor1_vflip = <0>;
            sensor1_hflip = <0>;
            sensor1_iovdd-supply = <&reg_dldo2>;
            sensor1_iovdd_vol = <1800000>;
            sensor1_avdd-supply = <&reg_dldo3>;
            sensor1_avdd_vol = <2800000>;
            sensor1_dvdd-supply = <&reg_eldo2>;
            sensor1_dvdd_vol = <1200000>;
            sensor0_power_en = <>;
            sensor1_reset = <&pio PE 7 1 0 1 0>;
            sensor1_pwdn = <&pio PE 6 1 0 1 0>;
            status  = "okay";
        };
        /* 一个 vinc 代表一个 /dev/video 设备 */
        vinc0:vinc@0 {
            vinc0_csi_sel = <0>;            /* 代表选择的 csi，MR813/R818 有两个 csi 接口
*/
            vinc0_mipi_sel = <0>;           /* 代表选择的 mipi 接口，MR813/R818 有两个 mipi
 接口 */
            vinc0_isp_sel = <0>;
            vinc0_isp_tx_ch = <0>;          /* 表示 ISP 的通道数，一般配置为 0 */
            vinc0_tdm_rx_sel = <0>;         /* 与 isp_sel 保持一致即可 */
            vinc0_rear_sensor_sel = <0>;    /* 该 video 可以选择从哪个 sensor 输入图像数据
*/
            vinc0_front_sensor_sel = <1>;
            vinc0_sensor_list = <0>;
            status = "okay";
        };
        vinc1:vinc@1 {
            vinc1_csi_sel = <0>;
            vinc1_mipi_sel = <0>; /* R528没有mipi，该项配置为0xff */
            vinc1_isp_sel = <0>; /* R528没有isp，该项配置为0 */
            vinc1_isp_tx_ch = <0>; /* R528没有isp，该项配置为0 */
            vinc1_tdm_rx_sel = <0>; /* R528没有isp，该项配置为0xff */
            vinc1_rear_sensor_sel = <0>;
            vinc1_front_sensor_sel = <1>;
            vinc1_sensor_list = <0>;
            status = "okay";
        };
        vinc2:vinc@2 {
            vinc2_csi_sel = <1>;
            vinc2_mipi_sel = <1>;
```

```
                vinc2_isp_sel = <1>;
                vinc2_isp_tx_ch = <0>;
                vinc2_tdm_rx_sel = <1>;
                vinc2_rear_sensor_sel = <1>;
                vinc2_front_sensor_sel = <1>;
                vinc2_sensor_list = <0>;
                status = "okay";
            };
            vinc3:vinc@3 {
                vinc3_csi_sel = <1>;
                vinc3_mipi_sel = <1>;
                vinc3_isp_sel = <1>;
                vinc3_isp_tx_ch = <0>;
                vinc3_tdm_rx_sel = <1>;
                vinc3_rear_sensor_sel = <1>;
                vinc3_front_sensor_sel = <1>;
                vinc3_sensor_list = <0>;
                status = "okay";
            };
        };

    /* 以下将配置两路 sensor 输入，产生 4 个 video 节点，内核配置 CONFIG_SUPPORT_ISP_TDM=n,此时
     * 不同 sensor 输出的节点不能同时使用,比如以下配置的 video0 不可以和 video2 video3 同时使用
     */
    vinc0:vinc@0 {
        vinc0_csi_sel = <0>;
        vinc0_mipi_sel = <0>;
        vinc0_isp_sel = <0>;
        vinc0_isp_tx_ch = <0>;
        vinc0_tdm_rx_sel = <0>;
        vinc0_rear_sensor_sel = <0>;
        vinc0_front_sensor_sel = <0>;
        vinc0_sensor_list = <0>;
        status = "okay";
    };
    vinc1:vinc@1 {
        vinc1_csi_sel = <0>;
        vinc1_mipi_sel = <0>;
        vinc1_isp_sel = <0>;
        vinc1_isp_tx_ch = <0>;
        vinc1_tdm_rx_sel = <0>;
        vinc1_rear_sensor_sel = <0>;
        vinc1_front_sensor_sel = <1>;
        vinc1_sensor_list = <0>;
        status = "okay";
    };
    vinc2:vinc@2 {
        vinc2_csi_sel = <1>;
        vinc2_mipi_sel = <1>;
        vinc2_isp_sel = <0>;
        vinc2_isp_tx_ch = <0>;
        vinc2_tdm_rx_sel = <0>;
        vinc2_rear_sensor_sel = <1>;
        vinc2_front_sensor_sel = <1>;
        vinc2_sensor_list = <0>;
        status = "okay";
    };
    vinc3:vinc@3 {
        vinc3_csi_sel = <1>;
        vinc3_mipi_sel = <1>;
```

```
        vinc3_isp_sel = <0>;
        vinc3_isp_tx_ch = <0>;
        vinc3_tdm_rx_sel = <0>;
        vinc3_rear_sensor_sel = <1>;
        vinc3_front_sensor_sel = <1>;
        vinc3_sensor_list = <0>;
        status = "okay";
    };

    /* 以下配置将可以从两路 sensor 同时输入，内核配置 CONFIG_SUPPORT_ISP_TDM=y,但是有个限制，
     * 只能先运行 video0，然后才可以运行 video2，关闭的时候也是如此，先关 video2，再关 video0
     */
    vinc0:vinc@0 {
        vinc0_csi_sel = <0>;
        vinc0_mipi_sel = <0>;
        vinc0_isp_sel = <0>;
        vinc0_isp_tx_ch = <0>;
        vinc0_tdm_rx_sel = <0>;
        vinc0_rear_sensor_sel = <0>;
        vinc0_front_sensor_sel = <0>;
        vinc0_sensor_list = <0>;
        status = "okay";
    };
    vinc1:vinc@1 {
        vinc1_csi_sel = <0>;
        vinc1_mipi_sel = <0>;
        vinc1_isp_sel = <0>;
        vinc1_isp_tx_ch = <0>;
        vinc1_tdm_rx_sel = <0>;
        vinc1_rear_sensor_sel = <0>;
        vinc1_front_sensor_sel = <1>;
        vinc1_sensor_list = <0>;
        status = "okay";
    };
    vinc2:vinc@2 {
        vinc2_csi_sel = <1>;
        vinc2_mipi_sel = <1>;
        vinc2_isp_sel = <1>;
        vinc2_isp_tx_ch = <0>;
        vinc2_tdm_rx_sel = <1>;
        vinc2_rear_sensor_sel = <1>;
        vinc2_front_sensor_sel = <1>;
        vinc2_sensor_list = <0>;
        status = "okay";
    };
    vinc3:vinc@3 {
        vinc3_csi_sel = <1>;
        vinc3_mipi_sel = <1>;
        vinc3_isp_sel = <1>;
        vinc3_isp_tx_ch = <0>;
        vinc3_tdm_rx_sel = <1>;
        vinc3_rear_sensor_sel = <1>;
        vinc3_front_sensor_sel = <1>;
        vinc3_sensor_list = <0>;
        status = "okay";
    };
```

修改该文件之后，需要重新编译固件再打包，才会更新到 dts。同时，如果需要使用双摄，双摄分别使用到两个 ISP，那么内核需要选上 SUPPORT_ISP_TDM 配置。

4.3 menuconfig 配置说明

在命令行进入 Tina 根目录，执行命令进入配置主界面：

```
source build/envsetup.sh    (详见1)
lunch 方案编号               (详见2)
make menuconfig             (详见3)

详注:
    1.加载环境变量及tina提供的命令;
    2.输入编号，选择方案;
    3.进入配置主界面(对一个shell而言，前两个命令只需要执行一次)
```

make menuconfig 配置路径：

```
Kernel modules
    └->Video Support
                └->kmod-sunxi-vfe(vfe框架的csi camera)              (详见1)
                └->kmod-sunxi-vin(vin框架的csi camera)              (详见2)
                └->kmod-sunxi-uvc(uvc camera)                       (详见3)

详注:
    1.平台使用vfe框架的csi camera选择该驱动;
    2.平台使用vin框架的csi camera选择该驱动;(该项与vfe框架，在同一个平台只会出现其中一个)
    3.usb camera选择该驱动;
```

在完成 sensor 驱动编写，modules.mk 和板级配置后，通过 make menuconfig 选上相应的驱动，camera 即可正常使用。下面以 R40 平台介绍。首先，选择 Kernel modules 选项进入下一级配置，如下图所示：

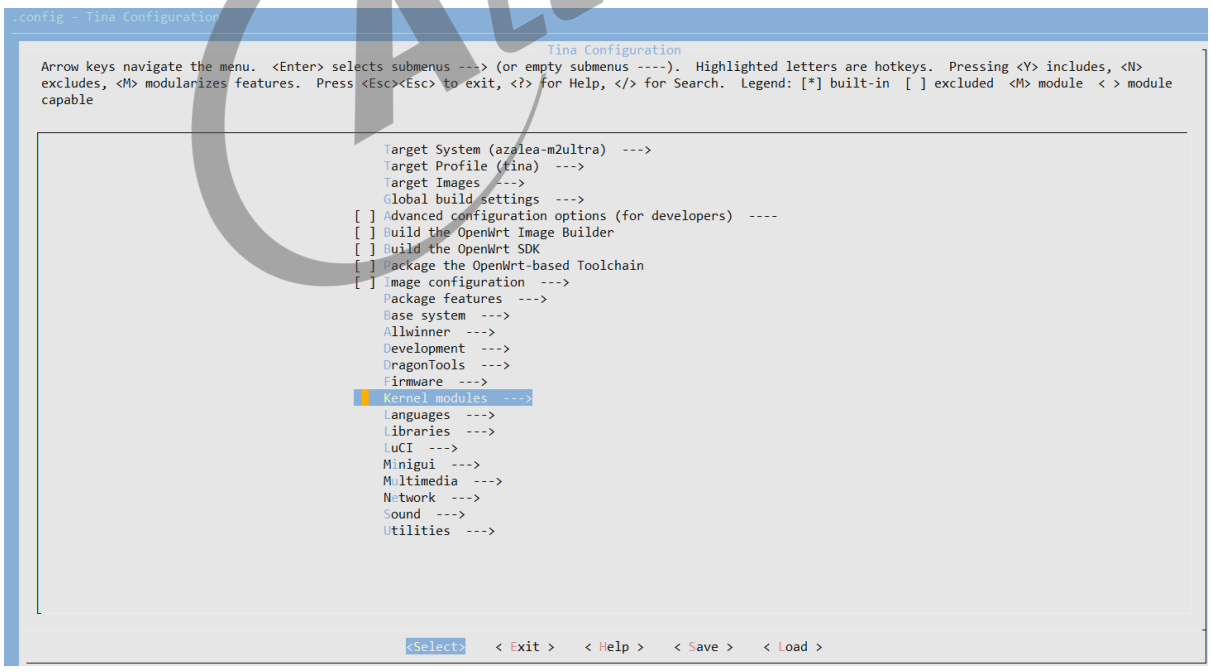

图 4-1: menuconfig

然后，选择 Video Support 选项，进入下一级配置，如下图所示：

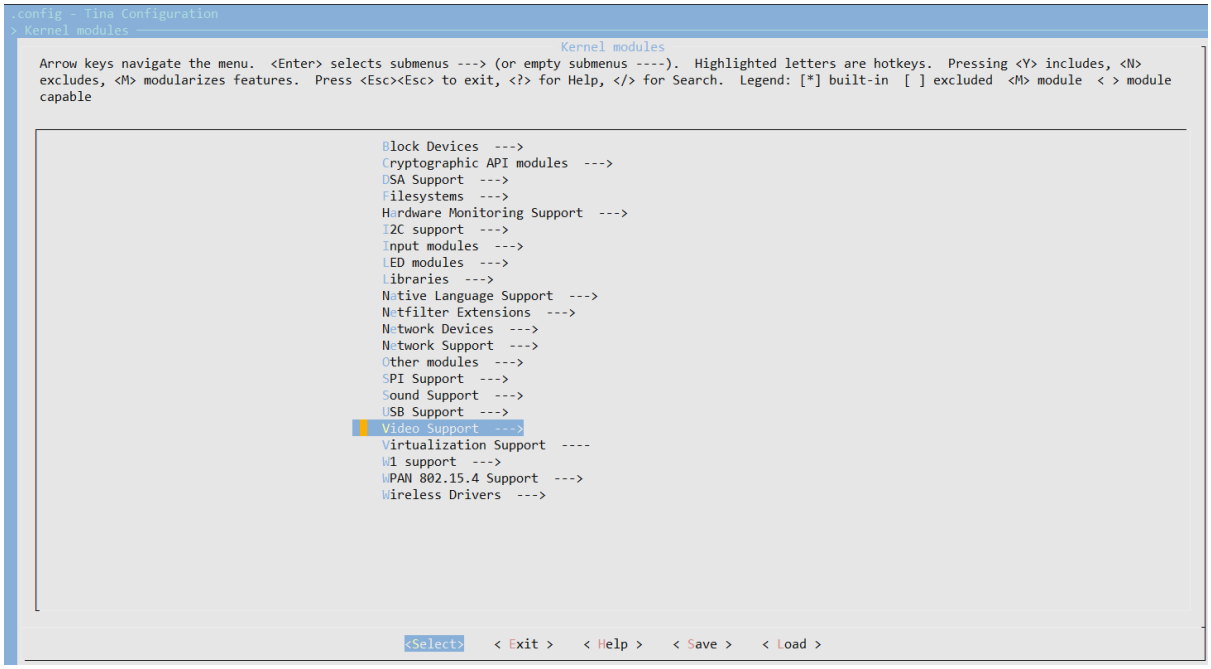

图 4-2: video

最后，选择 kmod-sunxi-vfe 选项，可选择 <*> 表示编译包含到固件，也可以选择表示仅编译不包含在固件。如下图所示：

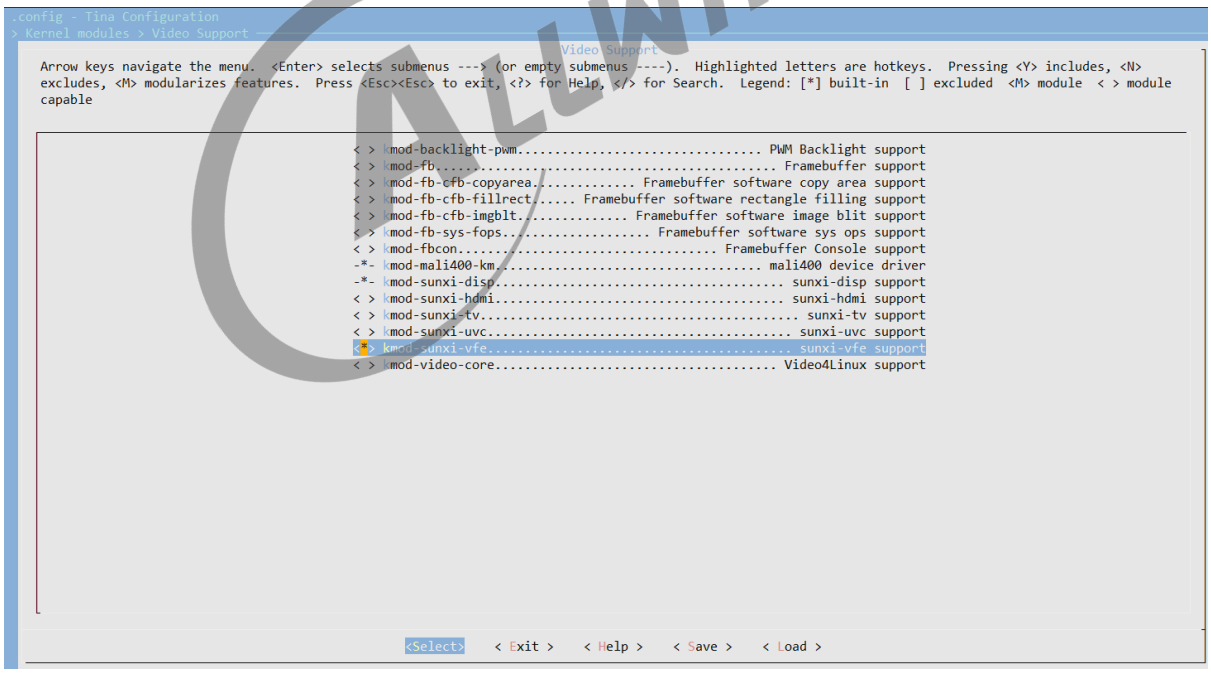

图 4-3: sunxi

4.4 如何增加 ISP 效果配置

在完成 ISP 调试之后，将会从 ISP 调试工程师中得到相应的头文件配置，添加操作如下：

4.4.1 VFE 框架

1. 将头文件添加到驱动的 sunxi-vfe/isp_cfg/SENSOR_H 目录下；
2. 在驱动 sunxi-vfe/isp_cfg 目录下，有个 isp_cfg.c 文件，这文件中有个 isp_cfg_array 数组，在 sensor 的 ISP 配置文件最下面也有个相应的结构体，在 isp_cfg_array 数组中按照数组的结构，增加 sensor 的 name 和结构体即可，这样将会在 ISP 匹配的时候，将会根据 name 匹配到相应的配置；

4.4.2 VIN 框架

4.4.2.1 R 系列

1. vin 框架的操作也是类似的，只是更换了位置。vin 的 ISP 配置在 tina/package/allwinner/libAWIspApi 目录下，其中 R311、MR133 在 src/isp520，而 R818、MR813 在 src/isp522。在 libisp/isp_cfg/SENSOR 目录下增加相应的头文件，然后在上一层目录的 isp_ini_parse.c 文件增加头文件以及修改相应的 isp_cfg_array cfg_arr 数组匹配即可。
2. VIN 使用 ISP，需要在 camerademo 中 make menuconfig 的时候，选择上 Choose whether to use VIN ISP (YES)。同时 VIN 的需要注意，当自己开发 camera HAL 层时，需要自己运行 camera ISP service，具体实现可参考 camerademo 的实现。添加正确时，在运行 camerademo 将会输出相应的 sensor 配置信息，比如：

```
[ISP]find imx278_mipi_2048_1152_60_0 [imx278_mipi_default_ini_mr813] isp config
```

上述表示正确查找到 imx278_mipi 这个 sensor 2048*1152 60fps 的 ISP 配置，其他的 sensor ISP 配置移植正确也将会有类似的打印，输出信息分别是 sensor name 、分辨率、帧率，确认这些信息一致即可。

4.4.2.2 V 系列

1. V 系列 ISP 库目录，V533、V83x 平台位于：softwinner/eyesee-mpp/middleware/sun8iw19p1/media
   V536 平台位于：softwinner/eyesee-mpp/middleware/v316/media/LIBRARY/libisp/，
   V85x 平台位于：external/eyesee-mpp/middleware/sun8iw21/media/LIBRARY/libisp/
2. 修改 libisp/isp_cfg/isp_ini_parse.c，将 ISP 效果.h 包含进来，并修改 struct isp_cfg_array cfg_arr[] 结构体；其中参数定义：

（1）Sensor 模块名称：sensor_mipi

（2）ISP 效果头文件名称

（3）分辨率宽

（4）分辨率高

（5）帧率

（6）红外 IR 模式标志位

（7）WDR 模式标志位

（8）ISP 参数结构体

```
struct isp_cfg_array cfg_arr[] = {

#ifdef SENSOR_GC2053
»       {"gc2053_mipi",  "gc2053_mipi_isp600_20220126_192349_day", 1920, 1080, 20, 0, 0, &gc2053_mipi_v853_isp_cfg},
»       {"gc2053_mipi",  "gc2053_mipi_isp600_20220211_152938_ir", 1920, 1080, 20, 0, 1, &gc2053_mipi_ir_isp_cfg},
#endif
          1                          2                      3     4    5  6  7            8
#ifdef SENSOR_GC4663
»       {"gc4663_mipi",  "gc4663_mipi_linear_isp600_20220106_173014", 2560, 1440, 20, 0, 0, &gc4663_mipi_linear_isp_cfg},
»       {"gc4663_mipi",  "gc4663_mipi_isp600_20220118_171111_ir", 2560, 1440, 20, 0, 1, &gc4663_mipi_ir_isp_cfg},
»       {"gc4663_mipi",  "gc4663_mipi_wdr_default_v853", 2560, 1440, 20, 1, 0, &gc4663_mipi_wdr_v853_isp_cfg},
#endif

};
```

图 4-4: sunxi

4.5 如何输出 RAW 数据

在 ioctl 的 VIDIOC_S_FMT 命令，将其参数 pixelformat 设置为 RAW 格式即可，RAW 格式如下：

```
    V4L2_PIX_FMT_SBGGR8
    V4L2_PIX_FMT_SGBRG8
    V4L2_PIX_FMT_SGRBG8
    V4L2_PIX_FMT_SRGGB8

    V4L2_PIX_FMT_SBGGR10
    V4L2_PIX_FMT_SGBRG10
    V4L2_PIX_FMT_SGRBG10
    V4L2_PIX_FMT_SRGGB10

    V4L2_PIX_FMT_SBGGR12
    V4L2_PIX_FMT_SGBRG12
    V4L2_PIX_FMT_SGRBG12
    V4L2_PIX_FMT_SRGGB12
```

将 pixelformat 设置为上述的其一即可输出 RAW 数据，而如何选择上述的操作，这个根据 sensor 驱动选择，如果驱动中 sensor_formats 的 mbus_code 设置为 MEDIA_BUS_FMT_SBGGR10_1X10，则在输出 RAW 数据时将 pixelformat 设置为 V4L2_PIX_FMT_SBGGR

当前 cameraemo 已经支持输出 RAW 数据，可参照本文档《cameraemo 输出 RAW 数据》章节。

4.6 如何计算实际曝光时间

该部分为使用到 ISP 的 RAW sensor 配置信息。曝光时间的计算和曝光控制寄存器、hts、pclk 这些配置相关，这些配置都在 sensor 驱动，ISP 将会根据 sensor 驱动中的设置计算相应的曝光时间，所以驱动中的配置必须正确，否则在调试 ISP 效果可能会遇到其他的一些问题。

```
static struct sensor_win_size sensor_win_sizes[] = {
    ...
    .hts = 928,
    .vts = 1720,
    .pclk = 48 * 1000 * 1000,
    ...
}
```

在 sensor 的驱动中有以上的一些配置，曝光时间在驱动中是以曝光行为计算单位的，即在 sensor_s_exp() 函数中设置的参数为曝光行，部分 sensor 是以 16 为一倍的，所以在计算实际的曝光行时，需要将上述函数参数除以 16。

曝光时间 = 曝光行 × hts / pclk

一般的 pclk 都是 M 级别的，所以时间单位为 us，部分 sensor 的曝光行为参数的十六分之一，需要除以十六，同时，曝光行不能大于 vts 的值，否则将会出现降帧、没有正常输出图像等问题。

4.7 如何脱离 isp tuning 工具微调图像亮度

在 isp 配置文件中，有类似以下的信息：

```
    .ae_cfg = {
        256, 555, 256, 555, 31, 22, 22, 25, 3, 130, 16, 60, 1, 2
    },
```

上述 ae_cfg 参数的倒数第 4 个数值（130）即是控制图像亮度的阀门（期望亮度），该值越大，图像亮度越高。ae_cfg 一共 14 个，分别对应着不同的环境亮度（Lux）。如何确定 AE 当前处于哪组 ae_cfg 参数呢？修改 isp 配置文件中的 isp_log_param = 0x1，然后重新编译运行相机应用，留意应用中关于 isp 的打印信息：

```
[ISP_DEBUG]: isp0 ae_target 92, pic_lum 0, weight_lum 0, delta_exp_idx 138, ae_delay 0, AE_TOLERANCE
 5
```

从上述信息可以看到当前的目标亮度是 92，这时可以查看 isp 配置文件 ae_cfg 中哪组的阀门处于 92 这个范围，如果需要增加亮度，则提高相应的阀门；降低亮度则降低阀门。相应的根据实际调试情况修改即可。调试之后，记得将 isp_log_param 参数还原为 0。

4.8 VIN 如何设置裁剪和缩放

裁剪修改 sensor 驱动：在驱动有类似以下的配置

```
static struct sensor_win_size sensor_win_sizes[] = {
    {
        .width      = VGA_WIDTH,
        .height     = VGA_HEIGHT,
        .hoffset    = 0,
        .voffset    = 0,
        .hts        = 878,
        .vts        = 683,
        .pclk       = 72 * 1000 * 1000,
        .mipi_bps   = 720 * 1000 * 1000,
        .fps_fixed  = 120,
        .bin_factor = 1,
        .intg_min   = 1 << 4,
        .intg_max   = (683) << 4,
        .gain_min   = 1 << 4,
        .gain_max   = 16 << 4,
        .regs       = sensor_VGA_120fps_regs,
        .regs_size  = ARRAY_SIZE(sensor_VGA_120fps_regs),
        .set_size   = NULL,
    },
};
```

上述的 width height 表示经过 isp 输出之后的数据，如果需要裁剪，修改 width、height、hoffset 和 voffset。裁剪之后的输出 width = sensor_output_src - 2*hoffset height = sensor_height_src - 2*voffset 注意，上述的 hoffset voffset 必须为双数。

所以，假设 sensor 输出的是 640 × 480，我们想裁剪为 320 × 240 的，则上述配置修改为：

```
static struct sensor_win_size sensor_win_sizes[] = {
    {
        .width      = 320,
        .height     = 240,
        .hoffset    = 160,
        .voffset    = 120,
        .hts        = 878,
        .vts        = 683,
        .pclk       = 72 * 1000 * 1000,
        .mipi_bps   = 720 * 1000 * 1000,
        .fps_fixed  = 120,
        .bin_factor = 1,
        .intg_min   = 1 << 4,
        .intg_max   = (683) << 4,
        .gain_min   = 1 << 4,
        .gain_max   = 16 << 4,
        .regs       = sensor_VGA_120fps_regs,
        .regs_size  = ARRAY_SIZE(sensor_VGA_120fps_regs),
        .set_size   = NULL,
    },
};
```

缩放配置：使用硬件缩放，可以在应用层通过 VIDIOC_S_FMT 设置分辨率的时候，直接设置分辨率的大小为缩放的分辨率即可。

```
        fmt.fmt.pix_mp.width = 320;
        fmt.fmt.pix_mp.height = 240;
```

上述的操作，将会使用硬件完成相应的缩放输出。

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。

Camera 模块调试一般可以分为三步：

1. 使用 lsmod 命令查看驱动是否加载，查看 /lib/modules/内核版本号目录下是否存在相应的 ko，如果没有，确认 modules.mk 是否修改正确，配置了开机自动加载。如果存在相应的 ko，可手动加载测试确认 ko 是否正常，手动加载成功，则确认内核的版本是否一致，导致开机时没有找到相应的 ko 从而没有加载。
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 移植一款 sensor 需要进行哪些操作

移植 camera sensor，主要进行以下操作：

1. 根据主板的原理图，确认与 sensor 模组的接口是否一致，一致才可以保证配置和数据的正常接收。
2. 根据产品的需求，让 sensor 模组厂提供产品所需的分辨率、帧率的寄存器配置，这一步需要注意，提供的配置需要是和模组匹配的。比如模组的 mipi 接口只引出 2lane，而提供的寄存器配置却是配置为 4lane 输出的，那么该配置在该模组无法正常使用，让模组厂提供该模组可以正常使用的正确配置。注意，该寄存器配置 SOC 原厂没有，需要 sensor 厂提供。
3. 拿到寄存器配置之后，按照本文档《驱动模块实现》章节完成 sensor 驱动的编写。
4. 在完成驱动的编写之后，按照本文档《Tina 配置》章节完成 modules.mk 的修改。
5. 根据板子的原理图与模组的硬件连接，参照本文档《sys_config.fex 配置》或者《MR813/R818 平台配置》章节完成 sys_config.fex 或者 board.dts 的修改。
6. 完成上述操作之后，按照本文档《menuconfig 配置说明》章节，选上 camera 驱动模块，按照《camera 功能测试》章节选上 camera 的测试程序，测试驱动移植是否正常。

5.2 I2C 通信出现问题

5.2.1 R16 R11 R40 等

I2C 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. 电源
    检查sys_config.fex
    vip_dev0_iovdd              = "axp22_eldo3"
    vip_dev0_iovdd_vol          = 2800000
    vip_dev0_avdd               = "axp22_dldo4"
    vip_dev0_avdd_vol           = 2800000
    vip_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。
c. reset和power down脚
    检查sys_config.fex配置
    vip_dev0_reset              = port:PH2<1><default><default><default>
    vip_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。
d. mclk
    检查sys_config.fex配置
    vip_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。

5.2.2 其他平台

出错时一般出现以下信息：

```
[    5.556579] sunxi_i2c_do_xfer()1942 - [i2c1] incomplete xfer (status: 0x20, dev addr: 0
    x30)
[    5.566234] sunxi_i2c_do_xfer()1942 - [i2c1] incomplete xfer (status: 0x20, dev addr: 0
    x30)
[    5.575963] sunxi_i2c_do_xfer()1942 - [i2c1] incomplete xfer (status: 0x20, dev addr: 0
    x30)
[    5.585375] [VIN_DEV_I2C]sc031gs_mipi sensor read retry = 2
[    5.591666] [sensorname_mipi] error, chip found is not an target chip.
```

出现上述错误打印时，可按以下操作逐步 debug。

1. 确认 sys_config.fex 中配置的 sensor I2C 地址是否正确（sensor datasheet 中标注，读地址为 0x6d，写地址为 0x6c，那么 sys_config.fex 配置 sensor I2C 地址为 0x6c）；
2. 在完成以上操作之后，在 senor 上电函数中，将掉电操作屏蔽，保持 sensor 一直上电状态，方便 debug；
3. 确认 I2C 地址正确之后，测量 sensor 的各路电源电压是否正确且电压幅值达到 datasheet 标注的电压要求；
4. 测量 MCLK 的电压幅值与频率，是否正常；
5. 测量 senso r 的 reset、pown 引脚电平配置是否正确，I2C 引脚 SCK、SDA 是否已经硬件上拉；
6. 确认 I2C 接口使用正确并使能（CCI / TWI）；
7. 如果还是 I2C 出错，协调硬件同事使用逻辑分析仪等仪器进行 debug；

5.2.3 经典错误

5.2.3.1 I2C 没有硬件上拉

```
twi_start()450 - [i2c2] START can't sendout!
twi_start()450 - [i2c2] START can't sendout!
twi_start()450 - [i2c2] START can't sendout!
[VFE_DEV_I2C_ERR]cci_write_a16_d16 error! slave = 0x1e, addr = 0xa03e, value = 0x1
```

出现上述的问题是因为 SDA、SCK 没有拉上，导致在进行 I2C 通信时，发送开始信号失败，SDA、SCK 添加上拉即可。

5.2.3.2 没有使能 I2C

```
[VFE]Sub device register "ov2775_mipi" i2c_addr = 0x6c start!
[VFE_ERR]request i2c adapter failed!
[VFE_ERR]vfe sensor register check error at input_num = 0
```

出现上述的错误，是因为使用 twi 进行 I2C 通信但没有使能 twi 导致的错误，此时需要确认 sys_config.fex 中，[twiX] 中的 twiX_used 是否已经设置为 1。

5.3 图像异常

5.3.1 运行 cameraderno 可以成功采集图像，但图像全黑 (RAW sensor)

当 cameraderno 成功采集到图像时，最起码整条数据通路已经正常，而发现图像时全黑的，注意以下几点：

1. 在编译 camerademo 之前，是根据平台 (MR813/R818/MR133/R311) 正确的选上了 “Enable vin isp support”，选上之后，重新编译 camerademo(建议 cd package/allwinner/-camerademo 目录后执行 mm -B 编译)；
2. 通过上述操作之后，执行新编译的 camerademo 可执行程序，运行过程应可看到类似”[ISP]create isp0 server thread‘信息，则正确运行 isp，这时再查看新抓取的图像数据；
3. 执行运行 camerademo 只会抓取 5 张图像数据，由于 isp 计算合适的图像曝光需要一定的帧数，所以可能存在前面几张图像黑的情况，修改 camerademo 运行参数，抓取多几张图像数据查看（20 张）；
4. 如果是没有移植 isp 的环境，则可修改 sensor 驱动中寄存器组中的曝光参数配置，增加初始化时曝光时间，从而使初始输出的图像亮度较合适；

5.3.2 camerademo 采集的图像颜色异常

运行 camerademo 采集图像之后，发现拍摄得到的轮廓正确但颜色不对，比如红蓝互换、画面整体偏红或偏蓝等颜色异常的情况，出现这样的问题，首先考虑是 sensor 驱动中配置的 RAW 数据 RGB 顺序错误导致的。在 sensor 驱动中有类似以下的配置：

```
static struct sensor_format_struct sensor_formats[] = {
    {
        .desc = "Raw RGB Bayer",
        .mbus_code = MEDIA_BUS_FMT_SBGGR10_1X10,
        .regs = sensor_fmt_raw,
        .regs_size = ARRAY_SIZE(sensor_fmt_raw),
        .bpp = 1
    },
};
```

以上配置表明 sensor 输出的图像数据是 RAW10，RGB 排列顺序是 BGGR，出现颜色异常时，一般就是 RGB 的排列顺序配置错误导致的，RGB 排列顺序一共有 4 种 (MEDIA_BUS_FMT_SBGGR10_1X10/MEDIA_BUS_FMT_SGBRG10_1X10/MEDIA_BUS_FMT_SGRBG10_
修改驱动中的 mbus_code 为上述的 4 种之一，确认哪一种颜色比较正常，则驱动配置正确。如果颜色还有细微的不够艳丽、准确等问题，需要进行 isp 效果调试，改善图像色彩。上述是以 10bit sensor 为例进行介绍，其他的 8bit、12bit、14bit 类似，参考上述即可。

5.4 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。
12. 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 videobuf-core.ko
    insmod videobuf-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_os.ko
    insmod vfe_subdev.ko
    insmod cci.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
    vip_used             = 1          ;确保used为1
    vip_dev_qty          = 2          ;确保csi接口上接的camera数量与ko加载情况相同
    vip_dev0_mname       = "ov5640"   ;确保camera型号与ko加载情况相同
    vip_dev0_twi_id      = 1          ;确保camera使用的i2c总线id与配置一样
```

```
    vip_dev1_mname      = "gc0308"   ;确保camera型号与ko加载情况相同
    vip_dev1_twi_id     = 1          ;确保camera使用的i2c总线id与配置一样
```

5.5 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**。

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

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。具体做法如下：

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。

5.6 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：

- 若 pclk 走线上有串联电阻，尝试将电阻阻值减小。
- 增强 pclk 的驱动能力，需要设置 camera 的内部寄存器。

5.7 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。

5.8 出现 [VFE_WARN] Nobody is waiting on this video buffer

上层还回来所有的 buffer，但是没有再来取 buffer。

5.9 出现 [VFE_WARN] Only three buffer left for csi

上层占用了大部分 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。

5.10 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 D11~D0 共 12bit，并口的 sensor 输出一般为 8/10bit，原理图连接需要做高位对齐。

6 camera 功能测试

Tina 系统可以通过 SDK 中的 camerademo 包来验证 camera sensor（usb camera）是否移植成功，如果可以正常捕获保存图像数据，则底层驱动、板子硬件正常。

6.1 camerademo 配置

在命令行中进入 Tina 根目录，执行 make menuconfig 进入配置主界面，并按以下配置路径操作：

```
Allwinner
    └->camerademo
```

首先，选择 Allwinner 选项进入下一级配置，如下图所示：

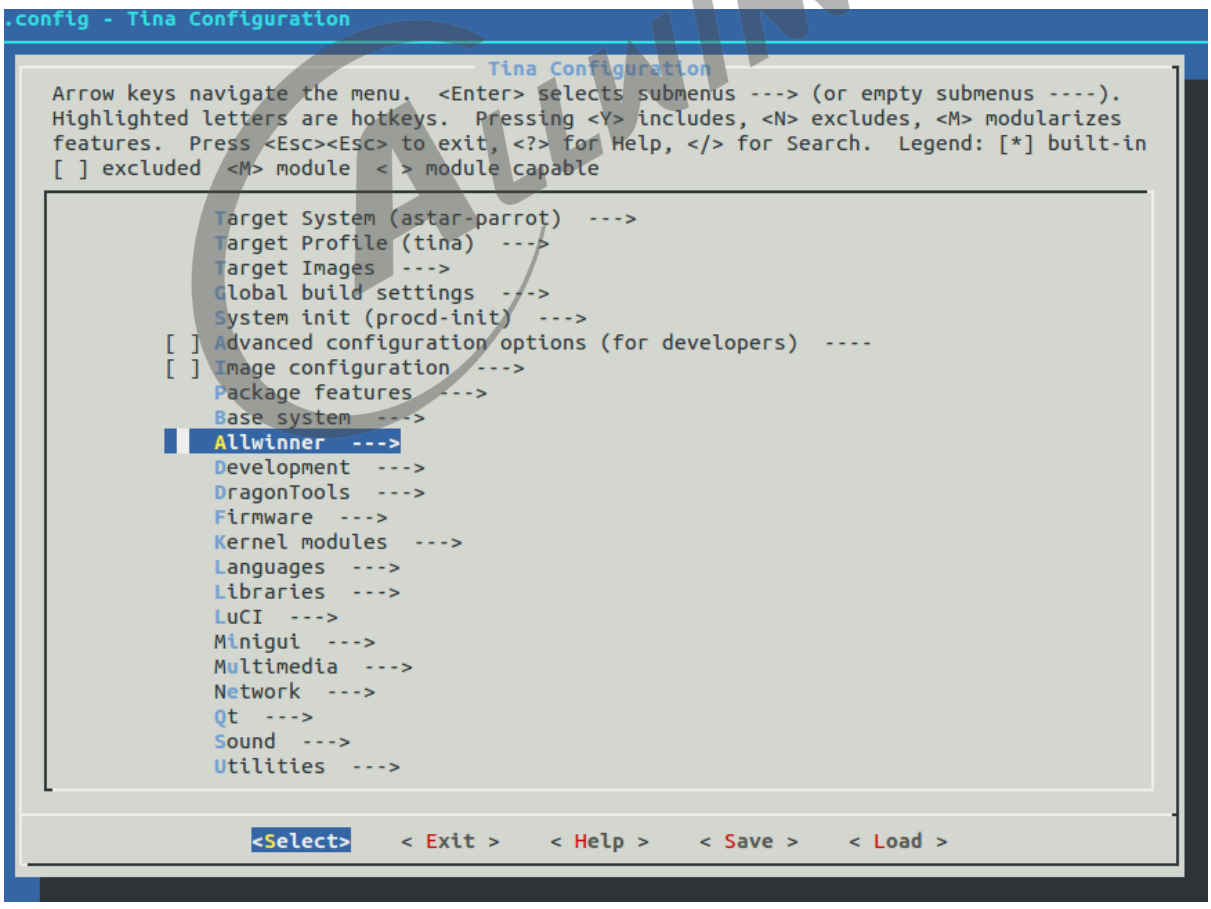

图 6-1: allwinner

然后，选择 cameraemo 选项，可选择 <*> 表示直接编译包含在固件，也可以选择表示仅编译不包含在固件。当平台的 camera 框架是 VIN 且需要使用 ISP 时，将需要在 cameraemo 的选项处点击回车进行以下界面选择使能 ISP。（该选项只能在 VIN 框架中，使用 RAW sensor 时使用，在修改该选项之后，需要先单独 mm -B 编译该 package）。

```
< > benchmarks.............................................. Benchmark program
< > boot-play.................................................... boot play
<*> cameraemo.......................................... cameraemo test sensor  --->
< > crash-worker.................................................. crash report  ----
< > crash-worker-test.......................................... crash report test
```

图 6-2: cameraemo

```
--- cameraemo.......................................... cameraemo test sensor
[*]   Enabel vin isp support
```

图 6-3: vinisp

6.2 源码结构

cameraemo 的源代码位于 package/allwinner/cameraemo/目录下：

```
|---src
|   cameraemo.c                  //camea测试的主流程代码
|   cameraemo.h                  //camera demo相关数据结构
|   common.c                  //实现共用的函数，转换时间、保存文件、测试帧率等
|   common.h                  //共用函数头文件
|   convert.c                 //实现图像格式转换函数
|   convert.h                     //图像格式转换函数头文件
```

6.3 cameraemo 使用方法

在小机端加载成功后输入 cameraemo help，假如驱动产生的节点 video0（测试默认以/dev/video0 作为设备对象）可以打开则会出现下面提示：

通过提示我们可以得到一些提示信息，了解到该程序的运行方式、功能，可以查询 sensor 支持的分辨率、sensor 支持的格式以及设置获取照片的数量、数据保存的格式、路径、添加水印、测试数据输出的帧率、从 open 节点到数据流打通需要的时间等，help 打印信息如下图：

```
root@TinaLinux:/# camerademo help
[CAMERA]*********************************************************
[CAMERA]*                                                       *
[CAMERA]*              this is camera test.                     *
[CAMERA]*                                                       *
[CAMERA]*********************************************************
[CAMERA]******************* camerademo help *********************
[CAMERA] This program is a test camera.
[CAMERA] It will query the sensor to support the resolution, output format and test frame rate.
[CAMERA] At the same time you can modify the data to save the path and get the number of photos.
[CAMERA] When the last parameter is debug, the output will be more detailed information
[CAMERA] There are eight ways to run:
[CAMERA]    1.camerademo --- use the default parameters.
[CAMERA]    2.camerademo debug --- use the default parameters and output debug information.
[CAMERA]    3.camerademo setting --- can choose the resolution and data format.
[CAMERA]    4.camerademo setting debug --- setting and output debug information.
[CAMERA]    5.camerademo NV21 640 480 30 bmp /tmp 5 --- param input mode,can save bmp or yuv.
[CAMERA]    6.camerademo NV21 640 480 30 bmp /tmp 5 debug --- output debug information.
[CAMERA]    7.camerademo NV21 640 480 30 bmp /tmp 5 Num --- /dev/videoNum param input mode,can save bmp or yuv.
[CAMERA]    8.camerademo NV21 640 480 30 bmp /tmp 5 Num debug --- /dev/videoNum output debug information.
[CAMERA]*********************************************************
root@TinaLinux:/#
```

图 6-4: help

Camerademo 共有 4 种运行模式：

1. 默认方式：直接输入 camerademo 即可，在这种运行模式下，将设置摄像头为 640*480 的 NV21 格式输出图像数据，并以 BMP 和 YUV 的格式保存在/tmp 目录下，而当输入 camerademo debug 将会输出更详细的 debug 信息；
2. 探测设置 camerademo setting：将会在运行过程中根据具体 camera 要求输入设置参数，当输入 camerademo setting debug 的时候，将会输出详细的 debug 信息；
3. 快速设置：camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7]，将会按照输入参数设置图像输出，同样，当输入 camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] debug 时将会输出更详细的 debug 信息。
4. 选择 camera 设置：camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] argv[8]，将会按照输入参数设置图像输出，同样，当输入 camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] argv[8] debug 时将会输出更详细的 debug 信息。

6.3.1 默认方式

当输入 camerademo 之后，使用默认的参数运行，则会打印一下信息，如下图：

```
root@TinaLinux:/# camerademo
[CAMERA]************************************************************
[CAMERA]*                                                          *
[CAMERA]*              this is camera test.                        *
[CAMERA]*                                                          *
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] open /dev/video0!
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] The path to data saving is /tmp.
[CAMERA] The number of captured photos is 5.
[CAMERA] save bmp and yuv format
[CAMERA] do not use watermarks
[CAMERA]************************************************************
[CAMERA] Using format parameters NV21.
[CAMERA] camera pixelformat: NV21
[CAMERA] Resolution size : 640 * 480
[CAMERA] The photo save path is /tmp.
[CAMERA] The number of photos taken is 5.
[CAMERA] Camera capture framerate is 30/1
[CAMERA] VIDIOC_S_FMT succeed
[CAMERA] fmt.type = 1
[CAMERA] fmt.fmt.pix.width = 640
[CAMERA] fmt.fmt.pix.height = 480
[CAMERA] fmt.fmt.pix.pixelformat = NV21
[CAMERA] fmt.fmt.pix.field = 1
[CAMERA] stream on succeed
[CAMERA] capture num is [0]
[CAMERA_PROMPT] the time interval from the start to the first frame is 87 ms
[CAMERA] capture num is [1]
[CAMERA] capture num is [2]
[CAMERA] capture num is [3]
[CAMERA] capture num is [4]
[CAMERA] Capture thread finish
[CAMERA] close /dev/video0
root@TinaLinux:/#
```

图 6-5: camerademouser

首先可以清楚的看到成功 open video0 节点，并且知道照片数据的保存路径、捕获照片的数量以及当前设置：是否添加水印、输出格式、分辨率和从开启流传输到第一帧数据达到时间间隔等信息。如果需要了解更多的详细信息，可以在运行程序的时候输入参数 debug 即运行 camerademo debug，将会打开 demo 的 debug 模式，输出更详细的信息，包括 camera 的驱动类型，支持的输出格式以及对应的分辨率，申请 buf 的信息，实际输出帧率等。

6.3.2 选择方式

在选择模式下有两种运行方式，一种是逐步选择，在 camera 的探测过程，知道其支持的输出格式以及分辨率之后再设置 camera 的相关参数；另一种是直接在运行程序的时候带上相应参数，程序按照输入参数运行（其中还可以选择 camera 索引，从而测试不同的 camera）。

1. 输入 cameraдemo setting，则按照程序的打印提示输入相应选择信息即可。

- 输入保存路径、照片数量、保存的格式等。

```
[CAMERA]************************************************************
[CAMERA] Please enter the data save path:
/tmp
[CAMERA] Please enter the number of captured photos:
5
[CAMERA] Please enter the data save type:
[CAMERA] 0:save BMP and YUV formats
[CAMERA] 1:save BMP format
[CAMERA] 2:save YUV format
0
[CAMERA]************************************************************
[CAMERA] The path to data saving is /tmp.
[CAMERA] The number of captured photos is 5.
[CAMERA] save bmp and yuv format
```

图 6-6: info

- 选择输出格式。

```
[CAMERA]************************************************************
[CAMERA] The sensor supports the following formats :
[CAMERA] index 0 : YUV422P
[CAMERA] index 1 : NV16
[CAMERA] index 2 : NV61
[CAMERA] index 3 : YUV420
[CAMERA] index 4 : YVU420
[CAMERA] index 5 : NV12
[CAMERA] index 6 : NV21
[CAMERA] index 7 : BGGR8
[CAMERA] index 8 : GBRG8
[CAMERA] index 9 : GRBG8
[CAMERA] index 10 : RGGB8
[CAMERA] index 11 : BGGR10
[CAMERA] index 12 : GBRG10
[CAMERA] index 13 : GRBG10
[CAMERA] index 14 : RGGB10
[CAMERA] index 15 : BGGR12
[CAMERA] index 16 : GBRG12
[CAMERA] index 17 : GRBG12
[CAMERA] index 18 : RGGB12
[CAMERA] index 19 : YUYV
[CAMERA] index 20 : UYVY
[CAMERA] index 21 : VYUY
[CAMERA] index 22 : YVYU
[CAMERA] index 23 : YUYV
[CAMERA] index 24 : UYVY
[CAMERA] index 25 : VYUY
[CAMERA] index 26 : YVYU
[CAMERA] index 27 : UYVY
[CAMERA] index 28 : VYUY
[CAMERA] index 29 : YVYU
[CAMERA] index 30 : YUYV
[CAMERA] Please enter the serial number you need for pixelformat:
6
[CAMERA] The input value is 6.
[CAMERA] camera pixelformat: NV21
[CAMERA]************************************************************
```

图 6-7: format

- 选择输出图像分辨率。

```
[CAMERA]************************************************************
[CAMERA] The NV12 supports the following resolutions:
[CAMERA] Index 0 : 2592 × 1936
[CAMERA] Index 1 : 2048 × 1536
[CAMERA] Index 2 : 1920 × 1080
[CAMERA] Index 3 : 1600 × 1200
[CAMERA] Index 4 : 1280 × 960
[CAMERA] Index 5 : 1280 × 720
[CAMERA] Index 6 : 1024 × 768
[CAMERA] Index 7 : 800 × 600
[CAMERA] Index 8 : 640 × 480
[CAMERA] Please enter the serial number you need for windows size:
0
[CAMERA] The input value is 0.
[CAMERA] Resolution size : 2592 × 1936
```

图 6-8: size

其它信息与默认设置一致，如需打印详细的信息，运行 cameraдemo setting debug 即可。

2. 第二种是设置参数：

- 默认的 video 0 节点：cameraдemo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7]。

输入参数代表意义如下：

```
argv[1]: camera输出格式---NV21 YUYV MJPEG等;

argv[2]: camera分辨率width;

argv[3]: camera分辨率height;

argv[4]: sensor输出帧率;

argv[5]: 保存照片的格式: all---bmp和yuv格式都保存、bmp---仅以bmp格式保存、yuv---仅以yuv格式保存;

argv[6]: 捕获照片的保存路径;

argv[7]: 捕获照片的数量;
```

例如：cameraдemo NV21 640 480 30 yuv /tmp 2，将会输出 640*480@30fps 的 NV21 格式照片以 yuv 格式、不添加水印保存在/tmp 路径下，照片共 2 张。

其它信息与默认设置一致，如需打印详细的信息，运行 cameraдemo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] debug 即可。

```
root@TinaLinux:/# camerademo NV21 640 480 0 yuv /tmp 2
[CAMERA]************************************************************
[CAMERA]*                                                            *
[CAMERA]*              this is camera test.                          *
[CAMERA]*                                                            *
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] open /dev/video0!
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] The path to data saving is /tmp.
[CAMERA] The number of captured photos is 2.
[CAMERA] save yuv format
[CAMERA] do not use watermarks
[CAMERA]************************************************************
[CAMERA] Using format parameters NV21.
[CAMERA] camera pixelformat: NV21
[CAMERA] Resolution size : 640 * 480
[CAMERA] The photo save path is /tmp.
[CAMERA] The number of photos taken is 2.
[CAMERA] Camera capture framerate is 30/1
[CAMERA] VIDIOC_S_FMT succeed
[CAMERA] fmt.type = 1
[CAMERA] fmt.fmt.pix.width = 640
[CAMERA] fmt.fmt.pix.height = 480
[CAMERA] fmt.fmt.pix.pixelformat = NV21
[CAMERA] fmt.fmt.pix.field = 1
[CAMERA] stream on succeed
[CAMERA] capture num is [0]
[CAMERA_PROMPT] the time interval from the start to the first frame is 90 ms
[CAMERA] capture num is [1]
[CAMERA] Capture thread finish
[CAMERA] close /dev/video0
root@TinaLinux:/#
```

图 6-9: run1

- 选择其他的 video 节点：camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] argv[8]。

输入参数代表意义如下：

```
argv[1]: camera输出格式---NV21 YUYV MJPEG等;
argv[2]: camera分辨率width;
argv[3]: camera分辨率height;
argv[4]: sensor输出帧率;
argv[5]: 保存照片的格式: all---bmp和yuv格式都保存、bmp---仅以bmp格式保存、yuv---仅以yuv格式保存;
argv[6]: 捕获照片的保存路径;
argv[7]: 捕获照片的数量;
argv[8]: video节点索引;
```

例如：camerademo YUYV 640 480 30 yuv /tmp 1 1，将会打开/dev/video1 节点并输出 640*480@30fps 的以 yuv 格式、不添加水印保存在/tmp 路径下，照片共 1 张。

其它信息与默认设置一致，如需打印详细的信息，运行 camerademo argv[1] argv[2] argv[3] argv[4] argv[5] argv[6] argv[7] argv[8] debug 即可。

```
root@TinaLinux:/# camerademo YUYV 640 480 0 yuv /tmp 1 1
[CAMERA]************************************************************
[CAMERA]*                                                            *
[CAMERA]*              this is camera test.                          *
[CAMERA]*                                                            *
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] open /dev/video1!
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] The path to data saving is /tmp.
[CAMERA] The number of captured photos is 1.
[CAMERA] save yuv format
[CAMERA] do not use watermarks
[CAMERA]************************************************************
[CAMERA] Using format parameters YUYV.
[CAMERA] camera pixelformat: YUYV
[CAMERA] Resolution size : 640 * 480
[CAMERA] The photo save path is /tmp.
[CAMERA] The number of photos taken is 1.
[CAMERA] Camera capture framerate is 30/1
[CAMERA] VIDIOC_S_FMT succeed
[CAMERA] fmt.type = 1
[CAMERA] fmt.fmt.pix.width = 640
[CAMERA] fmt.fmt.pix.height = 480
[CAMERA] fmt.fmt.pix.pixelformat = YUYV
[CAMERA] fmt.fmt.pix.field = 1
[CAMERA] stream on succeed
[CAMERA] capture num is [0]
[CAMERA_PROMPT] the time interval from the start to the first frame is 65 ms
[CAMERA] Capture thread finish
[CAMERA] close /dev/video1
```

图 6-10: run2

6.3.3 camerademo 保存 RAW 数据

当需要使用 camerademo 保存 RAW 数据时，只需要将输出格式设置为 RAW 格式即可。先确认 sensor 驱动中的 mbus_code 设置为多少位，假设驱动中，配置为 `mbus_code = MEDIA_BUS_FMT_SGRBG10_1X10`，那么可以确认 sensor 输出是 RAW10，camerademo 的输出格式设置为 RAW10 即可。比如输入 `camerademo RGGB10 1920 1080 30 bmp /tmp 5`，以上命令输出配置 sensor 输出 RAW 数据并保存在 /tmp 目录，命令的含义参考本章节的《选择方式》。

注意：RAW 数据文件的保存后缀是.raw 。

6.3.4 debug 信息解析

以下 debug 信息将说明 sensor 驱动的相关信息，拍摄到的照片保存位置、数量、保存的格式以及水印使用情况等：

```
root@TinaLinux:/# cameradeмo debug
[CAMERA]************************************************************
[CAMERA]*                                                            *
[CAMERA]*              this is camera test.                          *
[CAMERA]*                                                            *
[CAMERA]************************************************************
[CAMERA]************************************************************
[CAMERA] open /dev/video0!
[CAMERA]************************************************************
[CAMERA_DEBUG] Querey device capabilities succeed
[CAMERA_DEBUG] cap.driver=sunxi-vin
[CAMERA_DEBUG] cap.card=sunxi-vin
[CAMERA_DEBUG] cap.bus_info=
[CAMERA_DEBUG] cap.version=65536
[CAMERA_DEBUG] cap.capabilities=-2061496320
[CAMERA]************************************************************
[CAMERA] The path to data saving is /tmp.
[CAMERA] The number of captured photos is 5.
[CAMERA] save bmp and yuv format
[CAMERA] do not use watermarks
```

图 6-11: debug1

以下 debug 信息将说明驱动框架支持的格式以及 sensor 支持的输出格式：

```
[CAMERA_DEBUG]*************************************************************
[CAMERA_DEBUG] enumerate image formats
[CAMERA_DEBUG] format index = 0, name = YUV422P
[CAMERA_DEBUG] format index = 1, name = NV16
[CAMERA_DEBUG] format index = 2, name = NV61
[CAMERA_DEBUG] format index = 3, name = YUV420
[CAMERA_DEBUG] format index = 4, name = YVU420
[CAMERA_DEBUG] format index = 5, name = NV12
[CAMERA_DEBUG] format index = 6, name = NV21
[CAMERA_DEBUG] format index = 7, name = YUYV
[CAMERA_DEBUG] format index = 8, name = UYVY
[CAMERA_DEBUG] format index = 9, name = VYUY
[CAMERA_DEBUG] format index = 10, name = YVYU
[CAMERA_DEBUG] format index = 11, name = YUYV
[CAMERA_DEBUG] format index = 12, name = UYVY
[CAMERA_DEBUG] format index = 13, name = VYUY
[CAMERA_DEBUG] format index = 14, name = YVYU
[CAMERA_DEBUG]*************************************************************
[CAMERA_DEBUG] The sensor supports the following formats :
[CAMERA_DEBUG] Index 0 : YUV422P.
[CAMERA_DEBUG] Index 1 : NV16.
[CAMERA_DEBUG] Index 2 : NV61.
[CAMERA_DEBUG] Index 3 : YUV420.
[CAMERA_DEBUG] Index 4 : YVU420.
[CAMERA_DEBUG] Index 5 : NV12.
[CAMERA_DEBUG] Index 6 : NV21.
[CAMERA_DEBUG] Index 7 : YUYV.
[CAMERA_DEBUG] Index 8 : UYVY.
[CAMERA_DEBUG] Index 9 : VYUY.
[CAMERA_DEBUG] Index 10 : YVYU.
[CAMERA_DEBUG] Index 11 : YUYV.
[CAMERA_DEBUG] Index 12 : UYVY.
[CAMERA_DEBUG] Index 13 : VYUY.
[CAMERA_DEBUG] Index 14 : YVYU.
```

图 6-12: debug2

类似以下的信息代表这相应格式支持的分辨率信息：

```
[CAMERA_DEBUG]*************************************************************
[CAMERA_DEBUG] The YUV422P supports the following resolutions:
[CAMERA_DEBUG] Index 0 : 2592 * 1936
[CAMERA_DEBUG]*************************************************************
[CAMERA_DEBUG] The NV16 supports the following resolutions:
[CAMERA_DEBUG] Index 0 : 2592 * 1936
[CAMERA_DEBUG]*************************************************************
[CAMERA_DEBUG] The NV61 supports the following resolutions:
[CAMERA_DEBUG] Index 0 : 2592 * 1936
[CAMERA_DEBUG]*************************************************************
```

图 6-13: debug3

以下信息将会提示将要设置到 sensor 的格式和分辨率等信息：

```
[CAMERA] camera pixelformat: NV21
[CAMERA] Resolution size : 2592 * 1936
[CAMERA] The photo save path is /tmp.
[CAMERA] The number of photos taken is 5.
```

图 6-14: debug4

以下信息将会提示设置格式的情况，buf 的相应信息等：

```
[CAMERA] Camera capture framerate is 1/1
[CAMERA] VIDIOC_S_FMT succeed
[CAMERA] fmt.type = 9
[CAMERA] fmt.fmt.pix.width = 2592
[CAMERA] fmt.fmt.pix.height = 1936
[CAMERA] fmt.fmt.pix.pixelformat = NV21
[CAMERA] fmt.fmt.pix.field = 1
[CAMERA_DEBUG] reqbuf number is 3
[CAMERA_DEBUG] map buffer index: 0, mem: 0xb679e000, len: 72db00, offset: 0
[CAMERA_DEBUG] map buffer index: 1, mem: 0xb6070000, len: 72db00, offset: 72e000
[CAMERA_DEBUG] map buffer index: 2, mem: 0xb5942000, len: 72db00, offset: e5c000
[CAMERA] stream on succeed
```

图 6-15: debug5

以下信息将提示当前拍照的照片索引以及从开启流传输到 dqbuf 成功的时间间隔：

```
[CAMERA] capture num is [0]
[CAMERA_DEBUG]*****DQBUF[0] FINISH*****
[CAMERA_PROMPT] the time interval from the start to the first frame is 196 ms
[CAMERA_DEBUG] the interval of two frames is 0 ms
[CAMERA_DEBUG]************QBUF[0] FINISH*************
```

图 6-16: debug6

以下信息提示该 sensor 的实际测量帧率信息：

```
[CAMERA_DEBUG]*****************************************************************
[CAMERA_DEBUG] Query the actual frame rate.
[CAMERA_DEBUG] camera fps = 22.
[CAMERA_DEBUG]*****************************************************************
```

图 6-17: debug7

以下信息提示从 open 节点到可以得到第一帧数据的时间间隔，默认设置为测试拍照的相应设置：

```
[CAMERA_DEBUG]*****************************************************************
[CAMERA_DEBUG] Performance Testing---format:NV21 size:2592 * 1936
[CAMERA_DEBUG] The interval from open to streaming is 345 ms.
[CAMERA_DEBUG]*****************************************************************
```

图 6-18: debug8

6.3.5 文件保存格式

设置完毕之后，将会在所设路径（默认 /tmp）下面保存图像数据，数据分别有两种格式，一种是 YUV 格式，以 source_ 格式.yuv 名称保存；一种是 BMP 格式，以 bmp_ 格式.bmp 格式保存，如下图所示。

查看图像数据时，需要通过 adb pull 命令将相应路径下的图像数据 pull 到 PC 端查看。

```
root@TinaLinux:/tmp# ls
TZ                  booting_state      run                 source_NV21_5.yuv
bmp_NV21_1.bmp      lib                shm                 state
bmp_NV21_2.bmp      lock               source_NV21_1.yuv   tmp
bmp_NV21_3.bmp      log                source_NV21_2.yuv
bmp_NV21_4.bmp      resolv.conf        source_NV21_3.yuv
bmp_NV21_5.bmp      resolv.conf.auto   source_NV21_4.yuv
root@TinaLinux:/tmp#
```

图 6-19: save

6.4 select timeout 了，如何操作?

在完成 sensor 驱动的移植，驱动模块正常加载，I2C 正常通信，将会在 /dev 目录下创建相应的 video 节点，之后可以使用 camerademo 进行捕获测试，如果出现`select timeout,end capture thread!`，这个情况可按照以下操作进行 debug。

1. 先和模组厂确认，当前提供的寄存器配置是否可以正常输出图像数据。有些模组厂提供的寄存器配置还需要增加一个使能寄存器，这些可以在 sensor datasheet 上查询得到或者与模组厂沟通；
2. 通过 dmesg 命令，查看在运行 camerademo 的过程中内核是否有异常的打印。在 MR813/R818 平台，内核出现 tdm 相关字段的连续打印，则需要确认，board.dts 中的 isp 配置是否正确，单摄的配置，isp_sel 和 tdm_rx_sel 都需要配置为 0；双摄的则需要先运行配置为 isp0 的 video 节点才能再运行 isp1 的 video 节点；
3. 其他的按照是并口还是 mipi 接口进行相应的 debug；

6.4.1 DVP sensor

1. 确定 sensor 的出图 data 配置正确，是 8 位的、10 位的、12 位的? 确认之后，检查驱动中的 sensor_formats 和 sys_config.fex 中的 csi data pin 设置是否正确；
2. MCLK 的频率配置是否正确；
3. sensor 驱动的 sensor_g_mbus_config() 函数配置为 DVP sensor，type 需要设置为 V4L2_MBUS_PARALLEL；
4. 确定输出的 data 是高 8 位、高 10 位，确定硬件引脚配置没有问题；

5. 示波器测量 VSYNC、HSYNC 有没有波形输出，这两个标记着有一场数据、一行数据信号产生；
6. 测量 data 脚有没有波形，电压幅值是否正常；
7. 如果没有波形，检查一下 sensor 的寄存器配置，看看有没有软件复位的操作，如果有，在该寄存器配置后面加上”{REG_DLY，0xff}“进行相应的延时，防止在软件复位的时候，sensor 还没有准备好就 I2C 配置寄存器；
8. 如果上面都还是没有接收到数据，那么在 sensor 的驱动文件，有以下配置，这三个宏定义的具体值。每个都有两种配置，将这三个宏的配置两两组合，共 8 种配置，都尝试一下；

```
#define VREF_POL          V4L2_MBUS_VSYNC_ACTIVE_HIGH
#define HREF_POL          V4L2_MBUS_HSYNC_ACTIVE_HIGH
#define CLK_POL           V4L2_MBUS_PCLK_SAMPLE_RISING
```

6.4.2 mipi sensor

如果 mipi sensor 没有正常出图，做以下 debug 操作：

1. mipi 接口和主控板子连接不要飞线，mipi 信号本身就是高频差分信号，布线时都要求高，飞线更会影响其信号质量，导致无法正常接收数据；
2. 确认 sensor 驱动设置的 mipi 格式，同样是查看 sensor_g_mbus_config() 函数（lane 和通道数）；
3. 示波器测量 mipi 接口的 data 线、时钟线，看看有没有数据输出；
4. 检查一下寄存器配置方面有没有软件复位的，增加相应的延时；
5. 和模组厂商确认 sensor 驱动中对应分辨率的 sensor_win_sizes 以下参数配置是否与寄存器组配合的，因为这些参数将会影响 mipi 接收数据；

```
.hts = 3550,        .vts = 1126,        .pclk = 120 * 1000 * 1000,        .mipi_bps = 480 * 1000
 * 1000,
```

上面的 hts，又称 line_length_pck，VTS 又称 frame_length_lines，与寄存器的值要一致，Pclk(Pixel clock) 的值由 PLL 寄存器计算得出，可简单计算，pclk = hts × vts × fps；而 mipi_bps 为 mipi 数据速率，mipi_bps = hts × vts × fps ×（12bit/10bit/8bit）/ lane。

有些 sensor 的 datasheet 没有标注 hts 和 vts 的，但是他们有 H Blanking 和 Vertical blanking，他们的转换公式是：

hts = H Blanking + output_width

vts = Vertical blanking + output_height

Output_width 就是输出的一行的大小，output_height 就是输出的一列的大小。

gc 厂的 sensor，vts = VB + win_height + 16；VB 和 win_height 都是可以从寄存器中获取得到的，注意，win_height 是寄存器值，而不是输出的高。

6.4.3 其他注意事项

6.4.3.1 R311、MR133

sensor_sel

在 vin 框架中，sys_config.fex 有以下配置：

```
[vind0/sensor0]
...

[vind0/sensor1]
...

[vind0/vinc0]
vinc0_used              = 1
vinc0_csi_sel           = 0
vinc0_mipi_sel          = 0
vinc0_isp_sel           = 0
vinc0_rear_sensor_sel   = 0
vinc0_front_sensor_sel  = 0
vinc0_sensor_list       = 0
```

在这里主要是需要注意 vinc0_rear_sensor_sel 和 vinc0_front_sensor_sel 的配置，当它们都配置为 0，表明 vind0/vinc0 配置的 video0 节点，使用的是 vind0/sensor0 节点中配置的 sensor 输出图像数据；当它们配置为 0、1，表明 vind0/vinc0 配置的 video0 节点，可以使用 vind0/sensor0 和 vind0/sensor1 两个 sensor 输出图像数据，可以通过 ioctl 的 VIDIOC_S_INPUT 的 index 选择使用哪个 sensor 的输出；当它们都配置为 1 的时候，表明 vind0/vinc0 配置的 video0 节点，使用的是 vind0/sensor1 的输出。

mipi AB 配置

mipi 配置方面，还有一个需要注意的，该部分在 R311、MR133 平台才有这种情况。一般情况，我们使用的是 MCSIB 组的 mipi 接口，这个按照一般配置使用即可，MCSIA、MCSIB，这两个会在原理图上表明使用的是哪一组接口，如果单独使用 MCSIA 组的 mipi 接口，在 sys_config.fex 中配置如下：

```
由于只是使用MCSIA，所以应该配置的是[vind0/sensor1] 组sensor，vinc0_rear_sensor_sel 和
    vinc0_front_sensor_sel 都配置为 1.

[vind0/vinc0]
vinc0_used              = 1
vinc0_csi_sel           = 0
vinc0_mipi_sel          = 0
vinc0_isp_sel           = 0
vinc0_rear_sensor_sel   = 1
vinc0_front_sensor_sel  = 1
vinc0_sensor_list       = 0
```

著作权声明

商标声明

免责声明