



# Linux MIPI CSI 开发指南

版本号: 2.0  
发布日期: 2024.3.14

## 版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.8.2	AWA1689	初始版本
1.1	2022.8.3	AWA1689	更新调试方法/常见问题
1.2	2023.02.24	AWA2073	更新 sensor 不出图解决方法
1.3	2023.07.29	AWA2073	修改 csi_top 计算公式
1.4	2023.11.20	AWA2073	修改文档格式
2.0	2024.3.14	AWA2153	1. 增加了常见芯片系列的 CSI 模块主要规格参数的描述； 2. 修改了 kernel menuconfig 配置说明，分别对基本配置及其他常见配置进行详细说明； 3. 修改了 vind 节点配置说明，将节点属性说明放到设备树代码的对应行中； 4. 增加了在线和离线 Pipeline 典型场景的配图及相关说明； 5. 删除了源码模块结构相关内容； 6. 删除了 V4L2 标准 IOCTL 接口说明，增加了 VIN 框架中的非标准 IOCTL 接口说明； 7. 修改了 csi_test 测试用例的使用方法、参数说明及注意事项。

# 目 录

<b>1 前言</b>	<b>1</b>
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
<b>2 模块介绍</b>	<b>2</b>
2.1 模块功能介绍	2
2.2 规格说明	2
2.3 相关术语介绍	3
2.4 驱动框架介绍	4
2.4.1 Kernel 层	4
2.4.2 Video Input Framework 层	4
2.4.3 Device Driver 层	5
2.5 模块配置介绍	5
2.5.1 kernel menuconfig 配置说明	5
2.5.1.1 基本配置	5
2.5.1.2 可选配置	6
2.5.2 Device Tree 配置说明	7
2.5.3 Pipeline 典型场景	9
2.5.3.1 在线模式	10
2.5.3.2 离线模式	11
<b>3 非标准 IOCTL 接口</b>	<b>12</b>
<b>4 模块使用范例</b>	<b>13</b>
4.1 csi_test 测试用例	13
4.2 调用流程	14
<b>5 FAQ</b>	<b>15</b>
5.1 调试方法	15
5.1.1 调试节点	15
5.1.2 settle time	16
5.1.3 deskew	16
5.1.4 信号状态	17
5.1.4.1 MIPI	17
5.1.4.2 并口	17
5.2 常见问题	18
5.2.1 twi 不通	18
5.2.2 sensor 不出图	19

5.2.2.1 案例：imx317 wdr mode 无法出图 . . . . .	20
5.2.3 已出图但画面是绿色或者粉红色 . . . . .	20
5.2.4 twi 已通，但是读所有 sensor 寄存器值都为 0 . . . . .	21
5.2.5 画面旋转 180 度 . . . . .	21
5.2.6 没有 video 节点 . . . . .	21

## 插图

图 2-1	驱动框图 . . . . .	4
图 2-2	VIN_menuconfig 配置 . . . . .	6
图 2-3	在线模式_单路输入 4 路输出 . . . . .	10
图 2-4	离线模式_2 路 linear 输入 8 路输出 . . . . .	11
图 3-1	非标准 IOCTL 接口 . . . . .	12
图 4-1	CSI 调用流程 . . . . .	14
图 5-1	vi 节点 . . . . .	15
图 5-2	info->time_hs . . . . .	16
图 5-3	settle time 节点 . . . . .	16
图 5-4	settle time 节点读写 . . . . .	16
图 5-5	deskew . . . . .	17
图 5-6	twi 不通 . . . . .	18

# 1 前言

## 1.1 文档简介

介绍 VIN (video input) 驱动配置，API 接口和上层使用方法。

## 1.2 目标读者

camera 驱动开发、维护人员和应用开发人员。

## 1.3 适用范围

表 1-1: 适用产品列表

模块版本	驱动文件
Linux-5.10 以前版本	drivers/media/platform/sunxi_vin/*.c
Linux-5.10 及以后版本	longan/bsp/drivers/vin

## 2 模块介绍

### 2.1 模块功能介绍

1. Video input 主要由接口协议部分（MIPI, parser）、图像处理单元（ISP/VIPP）以及数据存储部分（bk）组成；
2. MIPI 与 parser（csi）模块主要实现视频图像数据的接收和解析；
3. ISP 模块实现 sensor raw data 数据的处理，包括 lens 补偿、去坏点、gain、gamma、de-mosaic、de-noise、color matrix 等以及一些 3A 的统计；
4. VIPP（scaler）模块能对图像进行缩小或添加水印的处理。VIPP 支持 bayer raw data 经过 ISP 处理后再缩小，也支持对一般的 YUV 格式的 sensor 图像直接缩小。
5. bk 模块通过直接内存访问技术（DMA）将采集到的图像数据搬运到内存中。

### 2.2 规格说明

下列表示列举了三种常见芯片系列的 CSI 模块主要规格参数：

芯片系列	sun50iw10p1	sun8iw21p1	sun55iw3p1
芯片型号	A100/A133/A133P 等	V853/R853/V853s/ R853S/V851s/V851SE 等	A523/A527/T527/AI985 等
miPI PHY 个数	2 个（PHYA/B），其 中 PHYA 支持 4Lane	2 个（PHYA/B）	4 个（PHYA/B/C/D）
支持的 miPI lane 数及组合	最大支持 4Lane+2Lane	最大支持 2Lane+2Lane、4Lane	最大支持 4Lane+4Lane、 4Lane+2Lane+2Lane、 2Lane+2Lane+2Lane+2Lane
并口支持情 况	BT656	BT656/BT1120	BT656/BT1120
parser(csi) 个数	2 个（parser0/1）， 其中 parser0 可配置 为并口输入	3 个（parser0/1/2）， 其中 parser2 可配置为 并口输入	4 个（parser0/1/2/3），其 中 parser3 可配置为并口输入
ISP 个数	1 个（可由 TDM 时分 复用为 4 个 ISP 虚体）	1 个（可由 TDM 时分复 用为 4 个 ISP 虚体）	1 个（可由 TDM 时分复用为 4 个 ISP 虚体）

芯片系列	sun50iw10p1	sun8iw21p1	sun55iw3p1
VIPP(scaler) 个数	4 个 (vipp0/1/2/3)	4 个 (vipp0/1/2/3)	4 个 (vipp0/1/2/3)
bk(DMA) 个数	4 个 (bk0/1/2/3)	4 个 (bk0/1/2/3)	6 个 (bk0/1/2/3/4/5)，其中 bk4/5 无对应的 vipp 相连
支持内核版本	linux5.4、linux5.15	linux4.9	linux5.15

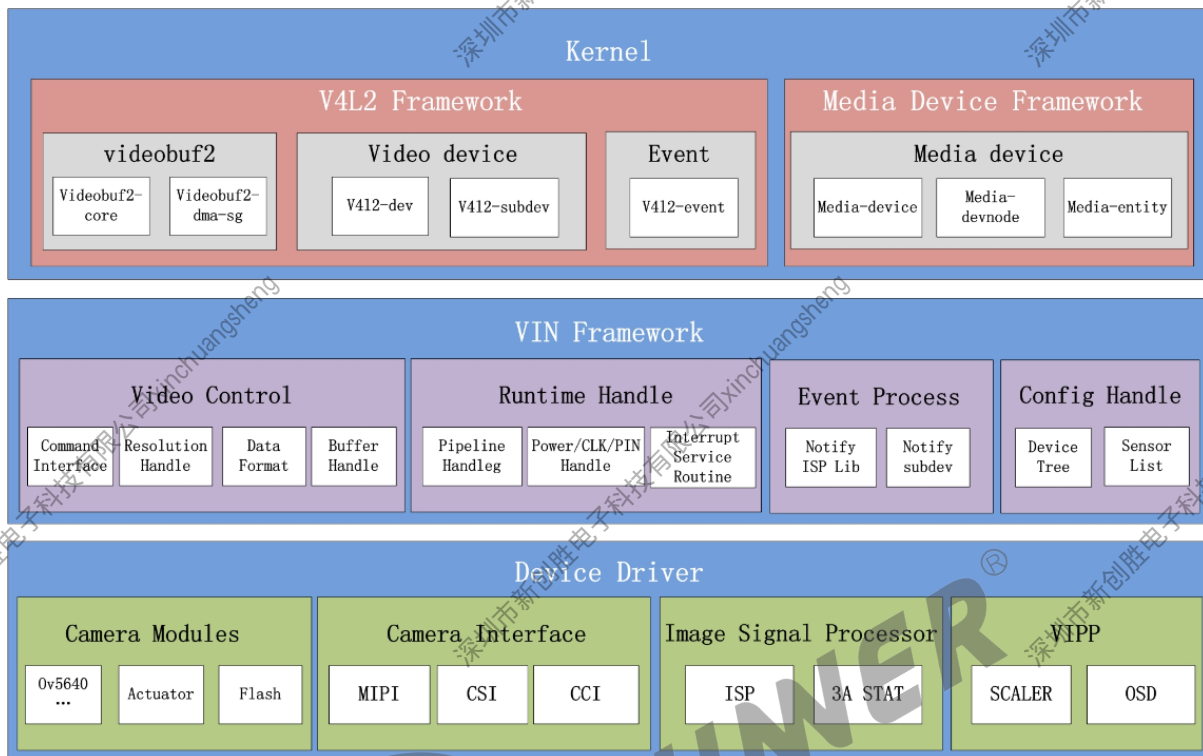
## 2.3 相关术语介绍

表 2-2: 软件术语

相关术语	解释说明
ISP	Image Signal Processor 图像信号处理
VIPP	Video Input Post Processor 图像输入后处理
MIPI	Mobile Industry Processor Interface 移动工业处理接口
CCI	Camera Control Interface 摄像头控制接口
TDM	Time division multiplexing ISP 时分复用
MCLK	Master clock (From AP to camera) 摄像头主时钟
PCLK	Pixel clock (From camera to AP, Sampling clock for data-bus) 像素时钟
YUV	Color Presentation (Y for luminance, U&V for Chrominance) 图像数据格式



## 2.4 驱动框架介绍



VIN 驱动可以分为 Kernel 层、Video Input Framework、Device Driver 层。

### 2.4.1 Kernel 层

- 1) Linux 内核视频驱动第二版 (Video for Linux Two);
- 2) 适用于收音机、视频编解码、视频捕获以及视频输出设备驱动;
- 3) 提供/dev/videoX 节点, 应用通过该节点进行相应视频流和控制操作;
- 4) Media Device Framework;
- 5) Linux 多媒体设备框架;
- 6) 适用于管理设备拓扑结构;
- 7) 提供/dev/mediaX 节点, 通过该节点应用可以获取媒体设备拓扑结构, 并能够通过 API 控制子设备间数据流向。

### 2.4.2 Video Input Framework 层

- 1) Video Control : 视频命令处理 (分辨率协商, 数据格式处理, Buffer 管理等) ;
- 2) Runtime Handle : 运行时管理 (Pipeline 管理, 系统资源管理, 中断调度等) ;

- 3) Event Process : 事件管理 (如上层调用, 中断等事件的接收与分发) ;
- 4) Config Handle : 配置管理 (如硬件拓扑结构, 模组自适应列表等) 。

## 2.4.3 Device Driver 层

- 1) Camera Modules : 模组驱动 (图像传感器, 对焦电机, 闪光灯等驱动) ;
- 2) Camera Interface : 接口驱动 (MIPI、Sub-Lvds、HiSpi、Bt656、Bt601、Bt1120、DC 等) ;
- 3) Image Signal Processor : 图像处理器驱动 (基本处理模块驱动, 3A 统计驱动) ;
- 4) Video Input Post Processor : 视频输入后处理 (Scaler, OSD 等) 。

## 2.5 模块配置介绍

### 2.5.1 kernel menuconfig 配置说明

#### 2.5.1.1 基本配置

以下为 VIN 模块的基本配置, 必须打开才能正常使用 Video Input 功能。

1. 确保 **V4L2 sub-device userspace API (CONFIG\_VIDEO\_V4L2\_SUBDEV\_API)** 配置已打开 (一般已默认打开)

- linux4.9 路径

- Device Drivers > Multimedia support[\*] > V4L2 sub-device userspace API[\*]

- linux5.15 路径

- Device Drivers > Multimedia support[\*] > Video4Linux options > V4L2 sub-device userspace API[\*]

2. 确保 **sunxi video input(AW\_VIDEO\_SUNXI\_VIN)** 和 **v4l2 new driver for SUNXI (CSI\_VIN)** 配置已打开 (一般已默认打开)

- linux4.9 路径

- Device Drivers > Multimedia support[\*] > V4L platform devices[\*] > sunxi video input (camera csi/mipi isp vipp)driver[\*]

- Device Drivers > Multimedia support[\*] > V4L platform devices[\*] > v4l2 new driver for SUNXI[\*]

- linux5.15 路径

- Allwinner BSP > Device Drivers > VIN (camera) Drivers > sunxi video input (camera csi/mipi isp vipp)driver[M]
- Allwinner BSP > Device Drivers > VIN (camera) Drivers > v4l2 new driver for SUNXI[M]

### 2.5.1.2 可选配置

以下配置则需要根据实际产品需求进行打开或关闭，如：使用闪光灯、对焦马达、使用 sensor\_list、打开 vin log、选择 sensor 驱动等。

配置路径：Allwinner BSP > Device Drivers > VIN (camera) Drivers，如下图所示：

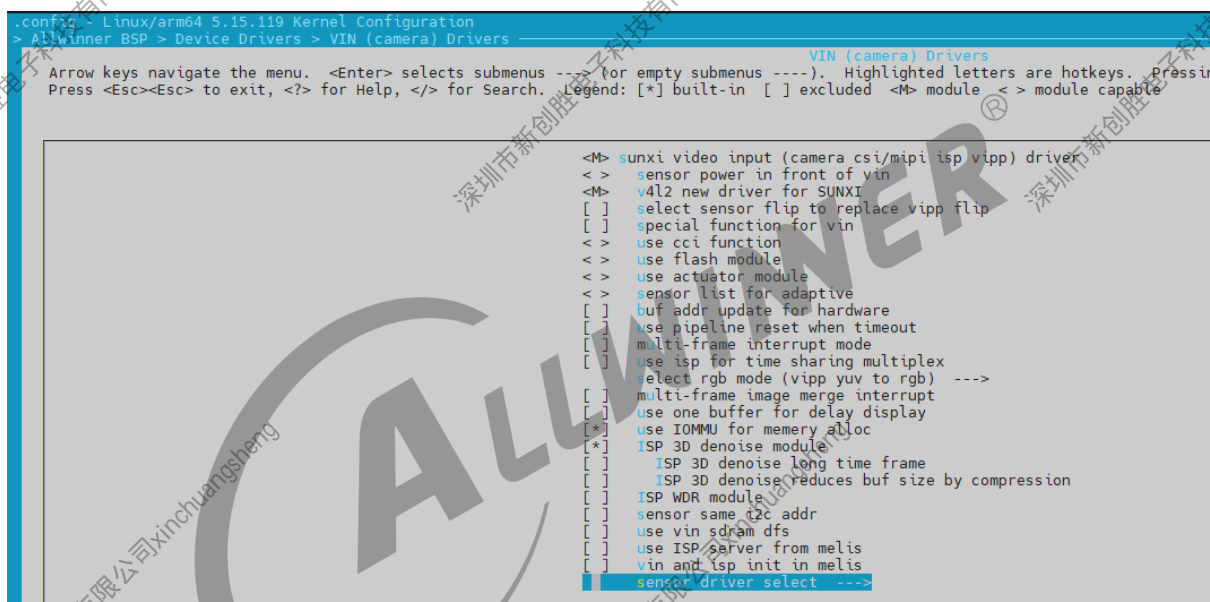


图 2-2: VIN\_menuconfig 配置

下面对几个配置做相关解释：

- special function for vin：特殊场景功能选项（如快速倒车场景）；
- use cci function：使用 CCI（Camera Control Interface）通信接口；
- sensor list for adaptive：使用 sensor\_list 功能，用于同一硬件接口位置适配多种 sensor 模组的情况；
- use isp for time sharing multiplex：使用 ISP 时分复用功能，开启离线模式时，需打开此配置开关；
- sensor same i2c addr：用于多个 i2c 设备地址相同的设备挂载在同一 I2C 总线的场景；
- use ISP server from melis：开启 melis 端的 ISP 服务；
- vin and isp init in melis：在 melis 端的进行 VIN 和 ISP 的初始化；
- sensor driver select：选择需要编译的 sensor 驱动。

## 2.5.2 Device Tree 配置说明

- sun\*.dtsi 为 SoC 所有方案的通用配置，其路径为：/bsp/config/{KERNEL\_VERSION}/sun\*.dtsi。
- 板级设备树 (board.dts) 路径：/device/config/chips/{IC}/configs/{BOARD}/{KERNEL\_VERSION}/board.dts。

在 dtsi 文件中，配置了该 SoC 的 CSI 控制器的通用配置信息，一般不建议修改，由 CSI 驱动维护者维护，如果需要修改配置请修改板级设备树 board.dts，板级设备树里面的内容会覆盖 dtsi 对应的信息。

- vind 节点配置

```
&vind0 {
    /* CSI模块时钟频率，计算公式：帧率 x (vts) x (hts) x 位宽 x 1(wdr则为2) / 8 / 1(双pixel则为2) / 1000000，向上取整，单位为MH;*/
    csi_top = <336000000>;
    /* ISP时钟频率，计算公式：帧率 x (vts) x (hts) / 1000000，向上取整，单位为MH;*/
    csi_isp = <300000000>;

    vind_mclkpin-supply = <&reg_aldo2>; /* 对应原理图中vcc-pe所采用的LDO */
    vind_mclkpin_vol = <1800000>;
    vind_mcsipin-supply = <&reg_cldo1>; /* 对应原理图中vcc-mcsi所采用的LDO */
    vind_mcsipin_vol = <1800000>;
    vind_mipipin-supply = <&reg_cldo3>; /* 对应原理图中vcc-pk所采用的LDO */
    vind_mipipin_vol = <3300000>;
    status = "okay";

    /* csi3常用于并口sensor，并口sensor不经过SOC MIPI模块，直接与csi3模块连接 */
    /* sun8iw21p1(V853系列)则csi2可用于并口 */
    csi3:csi@5823000 {
        pinctrl-names = "default","sleep";
        /* 可根据实际情况选择BT656/BT1120协议，若不使用并口sensor，下面两个pinctrl属性应置为空 */
        pinctrl-0 = <&ncsi_bt1120_pins_a>;
        pinctrl-1 = <&ncsi_bt1120_pins_b>;
        status = "okay";
    };

    /* ISP时分复用功能（TDM），若需使用ISP时分复用功能，则需开启TDM离线模式 */
    tdm0:tdm@0 {
        work_mode = <0>; /* 0:在线模式，1:离线模式，根据使用需求进行配置 */
    };

    /* 由于只有一个ISP实体，isp00(isp0)为普通在线模式下的唯一实体ISP，
     * isp01/02/03为ISP时分复用（离线模式下）的ISP虚体，
     * 而isp10(isp4)和isp(isp5)则为纯虚拟出来的ISP虚体，不具备ISP功能，通常用于不使用ISP功能时做路由，如YUV sensor
     * 。
     */
    isp00:isp@0 {
        work_mode = <0>; /* 0:在线模式，1:离线模式，根据使用需求进行配置 */
    };
    .....
    /* scaler00/10/20/30为真实scaler(vipp)，其他均为虚拟出来的vipp，与所用isp相对应*/
```

```

scaler0:scaler@0 {
    work_mode = <0>; /* 0:在线模式, 1:离线模式, 根据使用需求进行配置 */
};
.....

/* 对焦马达 */
actuator0:actuator@0 {
    device_type = "actuator0";
    actuator0_name = "ad5820_act";
    actuator0_slave = <0x18>;
    actuator0_af_pwdn = <0>;
    actuator0_afvdd = "afvcc-csi";
    actuator0_afvdd_vol = <2800000>;
    status = "disabled";
};

/* 闪光灯 */
flash0:flash@0 {
    device_type = "flash0";
    flash0_type = <2>; /* 0:FLASH_RELATING, 1:FLASH_EN_INDEPEND, 2:FLASH_POWER */
    flash0_en = <&r_pio PL 11 GPIO_ACTIVE_LOW>;
    flash0_mode = <0>;
    flash0_flvdd = "";
    flash0_flvdd_vol = <0>;
    status = "disabled";
};

sensor0:sensor@0 {
    device_type = "sensor0";
    sensor0_mname = "gc2053_mipi"; /* 需和sensor驱动中的SENSOR_NAME宏保持一致, 否则将无法成功加载驱动 */
    sensor0_twi_cci_id = <1>; /* sensor所使用的TWI总线号或者CCI的id. */
    sensor0_twi_addr = <0x6e>; /* sensor的twi地址 */
    sensor0_mclk_id = <0>; /* sensor所使用的mclk的id */
    sensor0_pos = "rear"; /* sensor的位置, 前置还是后置, 主要用在平板上, 其他场景一般无需配置该项 */
    sensor0_isp_used = <1>; /* 该sensor是否需要使用ISP */
    sensor0_fmt = <1>; /* sensor输出图像格式类型: 0-yuv, 1-bayer raw rgb */
    sensor0_stby_mode = <0>; /* 0-系统standby时sensor模组不掉电, 1-系统standby时sensor模组掉电 */
    sensor0_vflip = <0>; /* sensor画面垂直翻转使能开关 */
    sensor0_hflip = <0>; /* sensor画面水平翻转使能开关 */
    sensor0_iovdd_supply = <&reg_aldo2>; /* sensor IOVDD所用LDO编号, 具体参考原理图 */
    sensor0_iovdd_vol = <1800000>; /* sensor IOVDD所用LDO电压, 单位: 微伏 */
    sensor0_avdd_supply = <&reg_bldo2>; /* sensor AVDD所用LDO编号, 具体参考原理图 */
    sensor0_avdd_vol = <2800000>; /* sensor AVDD所用LDO电压, 单位: 微伏 */
    sensor0_dvdd_supply = <&reg_dldo2>; /* sensor DVDD所用LDO编号, 具体参考原理图 */
    sensor0_dvdd_vol = <1200000>; /* sensor DVDD所用LDO电压, 单位: 微伏 */
    sensor0_power_en = <0>; /* 模组POWER_EN电源使能GPIO配置, 根据实际情况进行配置 */
    sensor0_reset = <&pio PA 18 1 0 1 0>; /* 模组RESET复位GPIO配置, 根据实际情况进行配置 */
    sensor0_pwdn = <&pio PA 19 1 0 1 0>; /* 模组PWDN GPIO配置, 根据实际情况进行配置 */
    flash_handle = <&flash0>; /* 配置当前sensor所绑定的闪光灯 */
    act_handle = <&actuator0>; /* 配置当前sensor所绑定的对焦马达 */
    status = "okay";
};

sensor1:sensor@1 {
    .....
};

/* vinci节点说明:
* 1. vinci@X对应/dev/videoX节点
* 2. vinci00/10/20/30/40/50 (video0/4/8/12/16/17) 对应真实的bk0/1/2/3/4/5, 故在线模式下最多可支持6路视频输出

```

\*3. vinc01/02/03/11/12/13/21/22/23/31/32/33只在ISP时分复用（离线模式）的场景下才会使用。

```
*/
vinc00:vinc@0{
    vinc0_csi_sel = <0>; /* 该pipeline上parser的id, 必须配置, 且为有效id. */
    vinc0_mipi_sel = <0>; /* 该pipeline上mipi的id, 使用并口sensor时不经过mipi, 此项配置为0xff. */
    vinc0_isp_sel = <0>; /* 该pipeline上isp的id, 必须配置;
        * 若isp只是表示路由而不做效果处理, 则须配置为虚拟ISP(isp4/5)
        */
    vinc0_isp_tx_ch = <0>; /* 该pipeline上isp tx的通道id, 必须配置, 默认为0。
        * 当sensor配置为多通道输出时, 该ch可以配置0~3的值。
        */
    vinc0_tdm_rx_sel = <0>; /* 该pipeline上tdm rx的通道id, 必须配置, 默认为0;
        * 当不使用tdm功能时, 配置为0xff
        */
    vinc0_rear_sensor_sel = <0>; /* 该pipeline上使用的后置sensor的id;
        * 通过该vincX节点 (videoX) 来抓取指定sensor的数据
        */
    vinc0_front_sensor_sel = <0>; /* 该pipeline上使用的前置sensor的id;
        * 而一般情况下, 会配置成和上面rear_sensor_sel相同的id
        * 即仅通过不同的vinc (video) 节点来区分不同摄像头
        */
    vinc0_sensor_list = <0>; /* 是否使用sensor_list功能来自适应不同的模组, 1-使用, 0-不使用 */
    work_mode = <0x0>; /* 0-在线模式, 1-离线模式, 根据实际情况进行配置
        * 注: 只有vinc0/4/8/12可配置。
        */
    status = "okay"; /* vinc(video)节点使能开关, 根据实际情况配置为okay/disable */
};

vinc01:vinc@1{
    vinc1_csi_sel = <2>;
    vinc1_mipi_sel = <0xff>;
    vinc1_isp_sel = <1>;
    vinc1_isp_tx_ch = <1>;
    vinc1_tdm_rx_sel = <1>;
    vinc1_rear_sensor_sel = <0>;
    vinc1_front_sensor_sel = <0>;
    vinc1_sensor_list = <0>;
    status = "disabled";
};
.....
};
```

### 2.5.3 Pipeline 典型场景

下面例举两种典型的 Pipeline 场景，一种为常见的在线模式 pipeline，另一种为开启 ISP 时分复用功能（离线模式）下的 pipeline。



### 2.5.3.1 在线模式

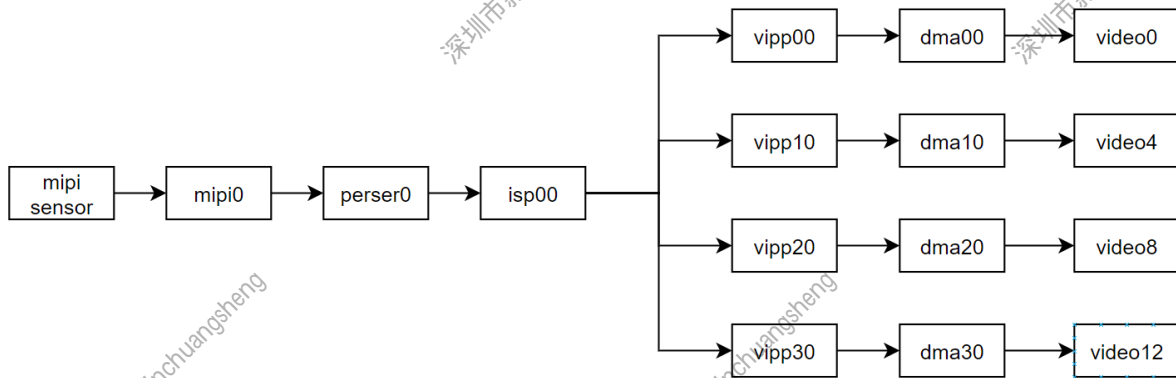


图 2-3: 在线模式 \_ 单路输入 4 路输出

上图为一种在线模式下单路输入 4 路输出的典型 pipeline 场景，下面简单描述一下数据从 sensor 到各 video 节点的过程：

- mipi sensor 数据首先经过 MIPI CSI 协议传输到 SOC 端的 mipi0 中，主要完成数据接收和 MIPI 协议解包等工作；
- mipiX 一般与 parserX 相连，数据经 parser 后进一步传输给 ISP；
- 然后经过 isp00（即 isp0，为真实存在的 ISP 实体）的处理后，分 4 路输入到 vipp00/10/20/30 中，必要时进行图像缩小；
- vipp、dma、video 为强关联（绑定），数据经相应的 vipp 处理后，由 dma 将图像数据搬运到内存中；
- 应用层通过/dev/video 节点来读取图像数据，每个 video 节点的图像数据完全相同，应用层可用于显示、编码等。

### 2.5.3.2 离线模式

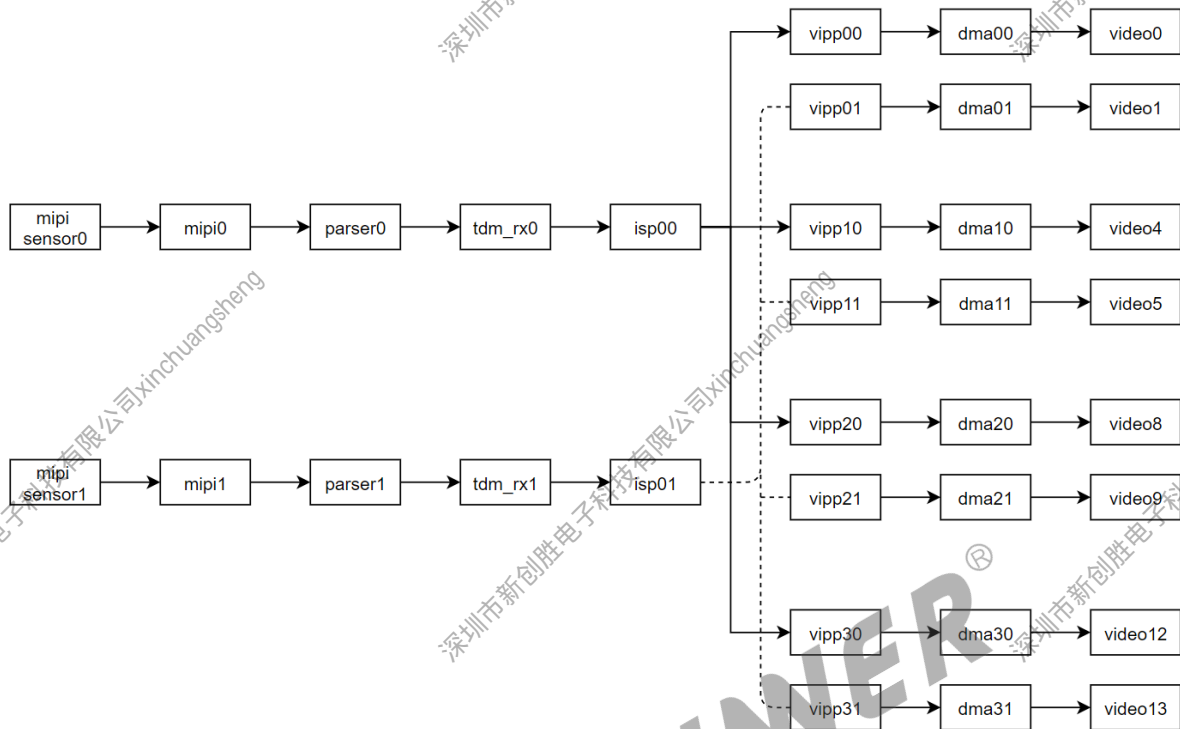


图 2-4: 离线模式\_2 路 linear 输入 8 路输出

上图作为一种离线模式下 2 路线性 sensor 输入 8 路输出的典型 pipeline 场景，两路 sensor 同时输入，但 ISP 只有一个，此时需要开启 ISP 时分复用功能（离线模式），下面简单描述一下数据从 sensor 到 video 节点的过程：

- sensor0/1 的两路数据分别经过 mipi0/1 后到达 parser0/1；
- tdm 为 isp 数据输入端，数据经过 tdm\_rx 控制同一个 isp 实现时分复用处理数据，isp00 和 isp01 为时分复用出来的 2 个 ISP（实际仍为同一个 ISP 实体）；
- 和在线模式一样，某一路 sensor 数据经过 ISP 处理后，分为 4 路输入到相应的 vipp、dma 和 video 中，2 路 sensor 数据所以共有 8 路，video0/4/8/12 为 sensor0 数据，video1/5/9/13 为 sensor1 数据；
- vipp 和 dma 在离线模式下也可时分复用成多个虚体，如在本例中，vipp(dma)00 时分复用成 vipp(dma)00 和 vipp(dma)01，vipp(dma)10 时分复用成 vipp(dma)10 和 vipp(dma)11，.....，故在本例的 dts 中，tdm0、isp00、scaler00/10/20/30、vinc00/10/20/30 的 work\_mode 均需配置成 <1>，即离线模式。



## 3 非标准 IOCTL 接口

下列表格中列举一些 VIN 框架中支持的非标准 IOCTL 接口，底层接口实现位于：longan/bsp/drivers/vin/vin-video/vin\_video.c 的 vin\_param\_handler() 函数中，各 IOCTL 的函数参数参考具体代码中的定义。

IOCTL	接口描述	备注
VIDIOC_SYNC_CTRL	parser SYNC控制	
VIDIOC_SET_TOP_CLK	设置CSI模块时钟频率	
VIDIOC_SET_FPS_DS	设置DMA FPS down sample功能	
VIDIOC_ISP_DEBUG	设置ISP debug模式	
VIDIOC_VIN_PTN_CFG	设置parser数据来源于ddr而不是sensor	
VIDIOC_VIN_RESET_TIME	设置parser的复位时间	该接口暂时不支持使用
VIDIOC_SET_PARSER_FPS	设置parser FPS down sample功能	
VIDIOC_SET_SENSOR_ISP_CFG	设置sensor的WDR模式或ISP的large image模式	必须在VIDIOC_S_PARM之后，VIDIOC_S_FMT之前调用
VIDIOC_SET_STANDBY	sensor standby控制	
VIDIOC_SET_VE_ONLINE	设置VE在线编码功能	必须在VIDIOC_S_INPUT之后，VIDIOC_S_PARM之前调用
VIDIOC_SET_VIPP_SHRINK	设置VIPP图像缩小设置	必须在VIDIOC_S_FMT之后，VIDIOC_STREAMON之前调用
VIDIOC_SET_TDM_SPEEDDN_CFG	设置TDM speed down	必须在VIDIOC_S_FMT之后，VIDIOC_STREAMON之前调用
VIDIOC_SET_ISP_CFG_ATTR	设置isp配置	仅在异构系统使用
VIDIOC_GET_ISP_CFG_ATTR	获取isp配置	仅在异构系统使用
VIDIOC_GET_ISP_ENCPP_CFG_ATTR	获取encpp参数	仅在异构系统使用
VIDIOC_SET_PHY2VIR	预留的物理地址转为虚拟地址，取melis保存的图像	仅在异构系统使用
VIDIOC_MERGE_INT_CH_CFG	开启混合中断配置	必须在VIDIOC_S_INPUT之前调用
VIDIOC_TVIN_INIT	开启混合制式配置	
VIDIOC_SET_DMA_MERGE	针对高分辨率sensor时，开启大图拆小图功能	

图 3-1: 非标准 IOCTL 接口

## 4 模块使用范例

### 4.1 csi\_test 测试用例

- 模块使用的 demo 的代码位于 longan/bsp/drivers/vin/vin\_test/mplane\_image，此目录下可以直接 make 生成 demo；
- 把 demo 推到机器里面执行便可以获取指定 video 节点的图像。
- 推荐在 pc 上创建 bat 批处理文件，使用 adb 命令完成一系列抓图的动作，bat 内容参考如下，不同机器请注意修改 push 进去的路径：

```
adb root
adb remount
adb shell "mkdir -p /vendor/tmp/csi_test/"
adb shell rm /vendor/tmp/csi_test/*.bin
adb push csi_test_mplane /vendor/tmp/csi_test/
adb shell chmod 777 /vendor/tmp/csi_test/csi_test_mplane
adb shell "cd /vendor/tmp/csi_test/ && ./csi_test_mplane 0 0 1920 1080 ./ 1 20"
adb shell ls /vendor/tmp/csi_test/
adb pull /vendor/tmp/csi_test/
pause
```

- ./csi\_test\_mplane 0 0 1920 1080 ./ 1 20 参数依次为：
- 【参数 0】./csi\_test\_mplane：测试用例可执行文件
- 【参数 1】0：/dev/video0
- 【参数 2】0：set\_input 时传入的 index 参数（默认设置为 0 即可）
- 【参数 3、4】1920 1080：抓图目标图像分辨率
- 【参数 5】./：抓取到的图像 bin 文件所保存路径，可自行修改
- 【参数 6】1：图像格式，1 表示抓取 YUV420M，其他分辨率可根据测试用例 csi\_test\_mplane.c 代码中 camera\_fmt\_set 函数来修改相应的图像格式。
- 【参数 7】20：采集帧数（若想要持续不间断抓图，将此帧数参数设置为 10000 及以上即可，常用于调试阶段使用）
- 【参数 8，可选】目标帧率（需根据实际所接模组或驱动配置进行合理选择）
- 【参数 9，可选】是否开启 WDR 功能
- 最后会在 bat 指令的文件夹生成 result 文件夹里面保存二进制的图像数据 \*.bin 文件
- 可用 RawViewer 等软件查看图像数据，注意看图时，看图软件中所设置的分辨率需注意按 16 字节进行对齐（向上取整），如使用 RawViewer 查看 1920x1080 图像时，需将看图尺寸设置为 1920x1088，否则可能会看到花屏或抖动的图像。

## 4.2 调用流程

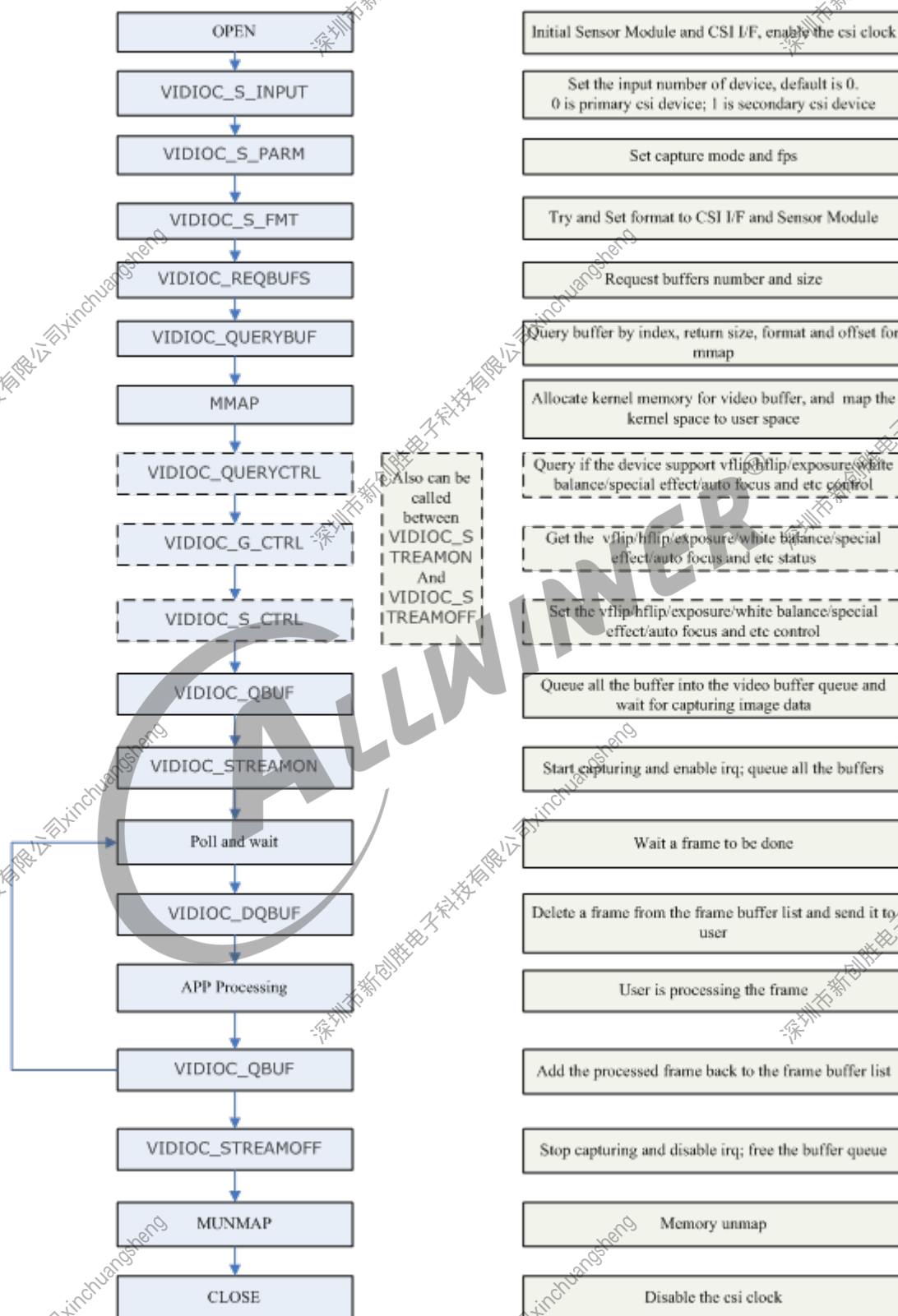


图 4-1: CSI 调用流程

## 5 FAQ

### 5.1 调试方法

#### 5.1.1 调试节点

```
*****
VIN hardware feature list:
mcsi 2, ncsi 1, parser 2, isp 1, vipp 4, dma 4
CSI_VERSION: CSI300_100, ISP_VERSION: ISP522_100
CSI_TOP: 336000000, CSI_ISP: 300000000
*****
vi0:
gc2385_mipi => mipi0 => csi0 => isp0 => vipp0
input => hoff: 0, voff: 0, w: 1600, h: 1200, fmt: GRBG10
output => width: 1600, height: 1080, fmt: YUV420M
interface: MIPI, isp_mode: NORMAL, hflip: 0, vflip: 0
prs_in => x: 1600, y: 1200, hb: 660, hs: 8181
buf => cnt: 5 size: 2617344 rest: 5, mode: software_update
frame => cnt: 2256, lost_cnt: 1, error_cnt: 0
internal => avg: 32(ms), max: 43(ms), min: 21(ms)
*****
```

图 5-1: vi 节点

当系统打开 DEBUG\_FS 编译宏时，可以 `cat /sys/kernel/debug/mpp/vi` 查看，否则可以 `cat /sys/devices/platform/soc@2900000/2000800.vind/vi`。

vi 节点保存的是当前或上一次工作（当前没有工作）的状态，下面对 vi 节点的关键信息进行说明。

- CSI\_TOP、CSI\_ISP 分别是对应 CSI、和 ISP 的工作频率；
- input 一行表示 CSI 接收到的图片尺寸，fmt 表示输入数据的格式；
- output 一行表示 CSI 输出尺寸，如果使用了缩放或者裁剪，那么输入输出尺寸会不一致，fmt 表示数据的输出格式；
- 最后一行分别表示平均帧间隔、最大帧间隔、最小帧间隔，一般可根据平均帧间隔计算得出帧率（如上图中 32ms 约为 31fps），调试帧率时可以参考。

## 5.1.2 settle time

方式一：修改对应 sensor 驱动中的 sensor\_probe 函数，可以添加或修改 info->time\_hs 的值即可。

```
info->win_size_num = N_WIN_SIZES;  
info->sensor_field = V4L2_FIELD_NONE;  
info->stream_seq = MIPI_BEFORE_SENSOR;  
info->time_hs = 0x30;  
info->af_first_flag = 1;  
info->exp = 0;  
info->gain = 0;
```

图 5-2: info->time\_hs

方式二：通过 mipi 子设备的 settle\_time 节点在线进行修改，settle\_time 节点路径：/sys/devices/platform/soc/5800800.vind/5810100.mipi。

进入节点路径后，可以看到当前目录下存在 settle\_time 节点：

```
root@TinaLinux:/sys/devices/platform/soc/5800800.vind/5810100.mipi# ls  
driver          modalias        power            subsystem  
driver override of node      settle_time      uevent
```

图 5-3: settle\_time 节点

可以通过 cat、echo 命令，对 settle\_time 节点进行读写操作：

```
root@TinaLinux:/sys/devices/platform/soc/5800800.vind/5810100.mipi# cat settle_time  
mipi0 settle time = 0x0  
mipi1 settle time = 0x0  
root@TinaLinux:/sys/devices/platform/soc/5800800.vind/5810100.mipi#  
root@TinaLinux:/sys/devices/platform/soc/5800800.vind/5810100.mipi# echo 0x20 >  
settle_time  
[ 146.574958] [VIN]Set mipi0 settle time as 0x20
```

图 5-4: settle\_time 节点读写

调整策略：settle time 的值慢慢增大调整，调大直到不能出图，再取一个略低于最大值的数值即可。调整范围：0x00-0xff。

## 5.1.3 deskew

方法一：修改 vin-mipi/combo\_csi/combo\_csi\_reg.c 中对应通道的 deskew 的默认值。

```
245 void cmb_phy_deskew1_cfg(unsigned int sel)
246 {
247     #if defined CONFIG_ARCH_SUN55IW3
248         if (sel) {
249             cmb_phy_set_deskew_laneck0(sel, 0x7);
250             //cmb_phy_set_deskew_laned0(sel, 0x5);
251             //cmb_phy_set_deskew_laned1(sel, 0x5);
252         } else {
253             cmb_phy_set_deskew_laneck0(sel, 0x2);
254             //cmb_phy_set_deskew_laned0(sel, 0x0);
255             //cmb_phy_set_deskew_laned1(sel, 0x0);
256         }
257     #endif
}
```

图 5-5: deskew

方法二：通过直接设置寄存器进行修改, 通过 echo 命令将修改后的值写入寄存器中。

```
echo [寄存器地址] [寄存器的值] > /sys/class/sunxi_dump/write
```

调整策略：建议修改值为 0、2、5、7。调整范围：0x0-0xf。

## 5.1.4 信号状态

介绍如何观测 SOC 主控的接收数据的信号状态，分别对 MIPI 和并口做出说明。

### 5.1.4.1 MIPI

MIPI 传输模式有两种：

LP (Low-Power) 模式：用于传输控制信号，最高速率 10 MHz。

HS (High-Speed) 模式：用于高速传输数据，以 MIPI DPHY V1.1 版本为例，速率范围 [80 Mbps - 1.5 Gbps] per Lane。

可以通过查看 user manual MIPI PHY 部分寄存器，观测 SOC 识别到的 clock lane 和 data lane 的 LP、HS 状态。

### 5.1.4.2 并口

对于并口接口的 sensor，可以查看 user manual CSI PARSE 部分的 parser signal 寄存器，观测 sensor 端 PCLK、DATA 的信号状态。以此判断 parser 是否有识别到 sensor 端发送的数据。



## 5.2 常见问题

### 5.2.1 twi 不通

如下图打印：

```
19.705480] sunxi-vin-core 2009400.vinc: Adding to iommu group 0
19.705978] sunxi-vin-core 2009600.vinc: Adding to iommu group 0
19.716659] [VIN_WARN]get csi mipi clk fail
19.721418] [VIN_WARN]get csi mipi src clk fail
19.735845] [gc2385_mipi] sd gc2385_mipi,PWR_ON!
19.746538] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.756148] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.765762] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.775179] [VIN_DEV_I2C]gc2385_mipi sensor read retry = 2
19.781552] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.791151] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.800764] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
19.810184] [VIN_DEV_I2C]gc2385_mipi sensor read retry = 2
19.816372] [gc2385_mipi]V4L2_IDENT_SENSOR = 0x0
19.821762] sunxi_i2c_do_xfer() 1828 - [i2c2] incomplete xfer (status: 0x20, dev addr: 0x37)
```

图 5-6: twi 不通

- 检查 dts 配置：

1. 检查引脚配置是否正确以及引脚是否有复用：MCLK 引脚以及引脚功能配置；TWI 引脚以及引脚功能配置；MIPI 引脚以及引脚功能配置；RESET/PWDN 引脚以及引脚功能配置；
2. sensor AVDD、DVDD、IOVDD 电压配置是否与 sensor datasheet 要求范围内
3. twi 地址配置是否与 sensor datasheet 一致，存在多个 twi 地址时可以尝试更换 twi 地址确认
4. 确认 sensor 节点中 twi\_cci\_id 属性配置与硬件实际使用一致

- sensor 驱动配置：

1. 检查 sensor 上电时序与 sensor datasheet 要求是否一致，主要检查三路电上电顺序，以及 pwn、reset 上电时序
2. 确认 chip id 与 sensor datasheet 上一致以及读取的寄存器地址正确
3. cci\_driver 结构体中地址位宽与数据位宽配置

- 硬件检查

1. 检查 sensor 子板原理图，并与实物进行对照，查看是否焊接正确
2. 确认 sensor 与开发板接触良好
3. 测量 sensor AVDD、DVDD、IOVDD 供电电压是否与配置一样
4. 测量 PWDN/RESET 引脚上电时序符合 datasheet 要求，电压变化符合要求以及引脚是否有上拉电阻
5. 确认 twi 引脚是否有上拉电阻

6. 测量 twi 波形是否正确，是否有回应，没有回应则按照排查流程重新仔细的进行排查一次。

- 其他

1. 考虑 sensor 是否损坏
2. 考虑开发板接口是否正常以及该 csi 通道是否正常
3. sensor 存在多个 twi 地址时确认 twi 地址方法：

方法一：在 sensor datasheet 中会给出不同 twi 地址时的硬件连接方式，并结合 sensor 子板原理图，即可确定 sensor 的 twi 地址

方法二：询问模组厂，确定 twi 地址

## 5.2.2 sensor 不出图

- dts 配置检查 video 节点配置与实际硬件连接的通道相同。video 节点属性含义参考 Device Tree 配置说明章节注：csi\_sel 与 mipi\_sel 应配置为相同 id，硬件已一一连接。

- sensor 驱动检查

1. 确认寄存器配置正确并且已经配置到 sensor 里：检查驱动中寄存器配置与原厂给的寄存器配置是否一致；将写进去的寄存器读出来和写入值对比是否一致
2. 检查 mclk 频率配置正确，并且 power on 时使能 mclk
3. 使用示波器测量 mclk 输出波形和频率是否正确
4. 确定 mbus\_code 位宽与寄存器配置支持的位宽相同

- mipi 检查

1. mipi 的 clock lane 存在两种工作模式，一种是连续时钟模式，传输过程不会切换 LP 状态；另一种是非连续时钟信号模式，每传输完一帧图像数据，帧 blanking 时将会切换为 LP 状态。目前大部分 MIPI sensor 一般都是非连续时钟模式。
2. 如果 sensor 是连续时钟模式，要保证 MIPI 在 sensor 之前初始化，需要在 sensor 驱动 sensor\_probe() 中配置 info->stream\_seq = MIPI\_BEFORE\_SENSOR;
3. 如果 sensor 是非连续时钟模式，可以通过判断 SOC 识别到的 LP、HS 模式状态是否在不断切换，来间接判断 SOC 的 MIPI 的接收状态。
4. 查看 user manual MIPI PHY 部分寄存器 (0xf0 寄存器)，观测 clock lane 和 data lane 的 LP、HS 状态是否有在不断切换，有则说明 MIPI 已经接收到了 sensor 发送的数据。如果没有切换则说明 MIPI 没有正确接收 sensor 数据，此时首先检查 sensor 与开发板接触是否良好，其次用示波器测量 sensor 端 mipi 数据 lane 和时钟 lane 波形，分析是否正在发送数据，如果没有可以尝试更换 sensor。



5. 检查 mipi payload 寄存器，查看 port mipi channel0 interrupt Pending(0x118) 寄存器，观测该寄存器第 7 bit~ 第 12 bit 是否有报错，如果有检查 mipi 寄存器配置是否正确，之后尝试修改 deskew 解决报错。
6. mipi 寄存器配置检查：检查 user manual MIPI PHY 部分寄存器 (0xf0 寄存器)，观测 clock lane 和 data lane 的 LP、HS 状态是否有在不断切换检查 port control(0x000) 寄存器配置是否正确检查 port lane mapp(0x004) 寄存器，mipi lane 使用是否正确检查 port mipi data identity(0x108) 寄存器，数据位宽设置是否正确检查 port mipi channel0 interrupt Pending(0x118) 寄存器，是否有报错，最低四位正常时都处于置起状态

- parser 检查

1. mipi 检查正确后，查看 parser 寄存器，查看 csic parser channel\_0 input parameter1(0x34) 寄存器，查看该寄存器值是否与设置的图像宽、高一致，数据是否稳定。如果数据有缺失且不稳定，可以尝试修改 settle time 解决（参考调试方法章节）。
2. parser 寄存器检查：检查使能位是否打开以及 parser mode(0x00) 设置是否正确检查 csic parser channel\_0 output horizontal size(0x28) 寄存器和 csic parser channel\_0 output vertical size(0x2c) 设置的图像分辨率与 sensor 驱动中 win\_size 中设置的是否一致检查 csic parser channel\_0 input parameter1(0x34) 寄存器，接收数据是否一致检查 csic parser channel\_0 interrupt status(0x44) 寄存器，若该寄存器有值，表示接收数据不稳定

- 其他

1. 向原厂确认寄存器配置并检查 sensor 驱动中寄存器配置
2. 更换 sensor 进行尝试
3. 更换开发板进行尝试

注：port control(0x000) 寄存器：括号内为寄存器地址，其他同理。

### 5.2.2.1 案例：imx317 wdr mode 无法出图

1. 按照上述对于“sensor 不出图”的排查步骤，检查到第四步时，发现 mipi 寄存器中 port mipi channel0 interrupt Pending 寄存器的 1bit 未置起，表示 mipi 接收数据不对或者 mipi 没有接收到数据；
2. 解决经过：再次向原厂确认寄存器配置，核对 mipi 寄存器配置；
3. 原因：imx317 只支持 10bit wdr mode，不支持 12bit wdr mode；
4. 解决方式：修改 sensor 驱动中 mbus\_code 位宽即可。

### 5.2.3 已出图但画面是绿色或者粉红色

一般是 YUYV 顺序反了，可以修改 sensor 驱动中 sensor\_formats 结构体的 mbus\_code 参数，修改 YUV 顺序即可。

## 5.2.4 twi 已通，但是读所有 sensor 寄存器值都为 0

【分析步骤一】检查 twi 通讯 addr 和 data 的位宽。

检查 sensor 驱动中 cci\_drv 结构体中定义的值是否符合 datasheet 要求。

【分析步骤二】检查 twi 通讯数据大小端是否不一致。

可以在读 sensor id 时把地址高低位相反来快速验证一下。

## 5.2.5 画面旋转 180 度

可以修改 board.dts 里面的 hflip 和 vflip 来解决，如果画面和人眼成 90 度的话，只能通过修改 sensor 配置来解决（只有部分 sensor 支持）。

## 5.2.6 没有 video 节点

【问题解析】没有加载 ko 或者 ko 加载失败。

【分析步骤一】检查模块加载顺序是否正确。

lsmod 看一下模块是否加载正确，如果报的错误是 [VIN\_ERR]registering gc2355\_mipi, No such device! 则表明 sensor 模块 gc2355\_mipi 没有加载。

【分析步骤二】检查 board.dts 文件配置是否配置了 vind0，且 status 为 okay。

【分析步骤三】如果是加载失败检查加载失败的原因是 twi 不通还是没有 ko。twi 不通参考前面的分析，没有 ko 请检查是否有对应的驱动并且在 Makefile 中使能了编译。




## 著作权声明

版权所有 © 2024 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标、产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。