



CV184x 屏幕对接使用手册

Version: 1.2.2

Release date: 2025-03-31

©2022 北京晶视智能科技有限公司
本文件所含信息归北京晶视智能科技有限公司所有。
未经授权，严禁全部或部分复制或披露该等信息。

目录

1	声明	2
2	MIPI DSI	3
2.1	环境准备	3
2.1.1	MIPI DSI 屏幕接口介绍	3
2.1.2	硬件连线确认	4
2.2	配置 MIPI 屏	4
2.2.1	在 u-boot 中配置 MIPI 屏	4
2.2.1.1	配置 MIPI Tx 设备属性	5
2.2.1.2	配置屏幕初始化序列	9
2.2.1.3	添加头文件的引用	10
2.2.1.4	配置 MIPI 屏 RESET 管脚	10
2.2.1.5	配置 MIPI 屏 POWER 管脚	11
2.2.1.6	配置 MIPI 屏 BACKLIGHT 管脚	12
2.2.1.7	配置 u-boot 环境变量	13
2.2.1.8	更换 logo 图片	13
2.2.1.9	编译烧写验证	13
2.2.2	在 kernel 中配置 MIPI 屏	14
2.2.2.1	配置 MIPI Tx 设备属性	15
2.2.2.2	配置屏幕初始化序列	15
2.2.2.3	添加头文件的引用	15
2.2.2.4	配置 MIPI 屏 RESET、POWER、BACKLIGHT 管脚	15
2.2.2.5	编译验证	17
2.2.3	在双系统中配置 MIPI 屏	18
2.2.3.1	配置 MIPI Tx 设备属性	18
2.2.3.2	打开 cvi_alios 中的配置开关	18
2.2.3.3	配置屏幕初始化序列	19
2.2.3.4	添加头文件的引用	19
2.2.3.5	配置 MIPI 屏 RESET、POWER、BACKLIGHT 管脚	19
2.2.3.6	编译验证	20
3	LVDS	21
3.1	环境准备	21
3.1.1	LVDS 屏幕接口介绍	21
3.1.2	硬件连线确认	22
3.2	配置 LVDS 屏	22
3.2.1	在 u-boot 中配置 LVDS 屏	22
3.2.1.1	配置 LVDS 设备属性	23
3.2.1.2	添加头文件的引用	25
3.2.1.3	配置 LVDS 屏 BACKLIGHT 管脚	26
3.2.1.4	配置 u-boot 环境变量	26

3.2.1.5	更换 logo 图片	26
3.2.1.6	编译烧写验证	26
3.2.2	在 kernel 中配置 LVDS	27
3.2.2.1	配置 LVDS 设备属性	27
3.2.2.2	添加头文件的引用	27
3.2.2.3	配置 LVDS 屏 BACKLIGHT 管脚	28
3.2.2.4	编译验证	28
3.2.3	在双系统中配置 LVDS	28
3.2.3.1	配置 LVDS 设备属性	28
3.2.3.2	打开 cvi_alios 中的配置开关	29
3.2.3.3	添加头文件的引用	29
3.2.3.4	配置 LVDS 屏 BACKLIGHT 管脚	29
3.2.3.5	编译验证	29
4	MIPI DSI(SDK V6.3.4 开始)	30
4.1	环境准备	30
4.1.1	MIPI DSI 屏幕接口介绍	30
4.1.2	硬件连线确认	31
4.2	PanelSupportList 框架概览	31
4.3	配置 MIPI 屏 (统一流程)	32
4.3.1	第 1 步: 编写屏参头文件	32
4.3.1.1	命名规范	32
4.3.1.2	头文件内容模板	32
4.3.1.3	关键数据结构的说明	33
4.3.1.4	初始化命令	36
4.3.2	第 2 步: 注册屏参头文件	37
4.3.2.1	u-boot / cvi_alios 注册方式 (cvi_panels.h)	37
4.3.2.2	cvi_mpi 注册方式 (sample_panel.h + sample_panel.c)	37
4.3.3	第 3 步: 配置使能并构建	38
4.3.3.1	u-boot 配置	38
4.3.3.2	cvi_alios 配置	38
4.3.3.3	cvi_mpi 配置	38
4.3.3.4	构建	38
4.4	RESET / POWER / BACKLIGHT 管脚配置	39
4.4.1	u-boot 配置方式 (DTS)	39
4.4.2	cvi_mpi 配置方式 (手动 GPIO)	39
4.4.2.1	方式一: sample_panel 命令行参数 (仅调试用, 代码中已注释)	39
4.4.2.2	方式二: shell 手动操作 sysfs	40
4.4.3	cvi_alios 配置方式 (PLATFORM_PanelInit)	40
4.5	更换 Logo 图片	41
4.6	编译烧写验证	41

修订记录

Revision	Date	Description
1.0	2025/03/31	Initial version

1 声明



法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

联系我们

地址 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

电话 +86-10-57590723 +86-10-57590724

邮编 100094（北京）518100（深圳）

官方网站 <https://www.sophgo.com/>

技术论坛 <https://developer.sophgo.com/forum/index.html>

2 MIPI DSI

概述

The Display Serial Interface (DSI) 接口是移动行业处理器接口联盟 (Mobile Industry Processor Interface alliance, MIPI 联盟) 定义的一种高速串行接口, 主要用于处理器和显示模块之间的连接。

本章介绍如何在 CVITEK 处理器解决方案上开发调试 MIPI LCD 屏, 以帮助客户有序快速开发 MIPI LCD 业务。

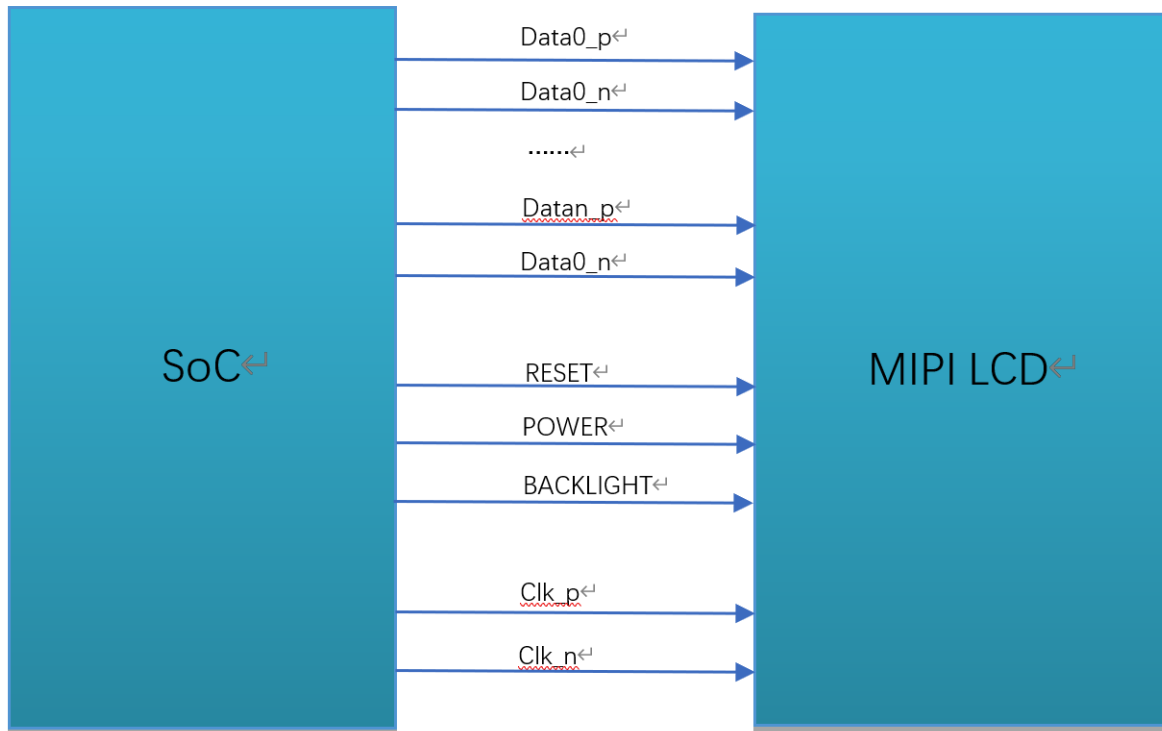
2.1 环境准备

2.1.1 MIPI DSI 屏幕接口介绍

MIPI DSI 屏幕一般有以下几种信号, 如图所示。

- mipi 时钟线 (CLK)
- mipi 数据线 (DATA), 最大为 4Lane (仅可以为 1/2/4Lane)
- 背光控制信号 (BACKLIGHT)
- 复位引脚 (RESET)
- Panel 电源供电 (POWER)

MIPI DSI 接口连线示意图



2.1.2 硬件连线确认

检查硬件连线，确认无异常。具体有些引脚差异，需对照屏幕厂商提供的规格书及电路原理图确认。

2.2 配置 MIPI 屏

根据上节环境准备的内容，在接口和连线上了解了屏幕对接的配置，在这一章节中将对屏幕对接时在软件方面需要进行的配置进行说明。

CVITEK 有两种方案进行 MIPI 屏幕的对接，分别是在 u-boot 及 kernel 中进行屏的初始化，区别在于 u-boot 中进行初始化后，开机可以显示客户的 logo 图片，而带屏的产品基本都会有显示 logo 的需求。实际应用中根据需求二者选其一。

2.2.1 在 u-boot 中配置 MIPI 屏

u-boot 中配置 MIPI 屏是通过 CVITEK 开发的 showlogo 命令，设备上电后，敲回车进入 u-boot 命令行，printenv 可以看到 showlogo 命令（不同板卡命令会有区别），bootcmd 在引导内核之前会执行该命令进行屏的初始化并显示 logo。

示例：

```
showlogo=mmc dev 0;mmc read 0x84080000 0xA000 0x400; cvi_jpeg 0x84080000 0x81800000 0x80000; startvo 0 8192 0;startvl 0 0x84080000 0x81800000 0x80000 32;setvobg 0 0xffffffff
```

本文档重点讲解屏的初始化部分，显示 logo 具体请参考《CV184X 开机画面使用指南》。其中，屏的初始化部分在 “startvo 0 8192 0” 中实现。

2.2.1.1 配置 MIPI Tx 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在 u-boot-2021.10/include/cvitek/cvi_panels/下，客户可以参照其余的头文件模板新增自己的 panel 头文件。

combo_dev_cfg_s 结构体定义

```
struct combo_dev_cfg_s {  
    unsigned int devno;  
  
    enum mipi_tx_lane_id lane_id[LANE_MAX_NUM];  
  
    enum output_mode_e output_mode;  
  
    enum video_mode_e video_mode;  
  
    enum output_format_e output_format;  
  
    struct sync_info_s sync_info;  
  
    unsigned int pixel_clk;  
  
    bool lane_pn_swap[LANE_MAX_NUM];  
};
```


成员名称	描述
devno	MIPI Tx 设备号，默认 0
lane_id	<p>主控端和屏端 Lane 号的对应关系，未使用的 Lane 填-1 即可。</p> <p>共 5 个成员，依序分别代表主控端的 MIPI_TX_0~MIPI_TX_4，实际填写的内容需要根据对应到屏端的 MIPI lane 号。</p> <p>例如，第一个成员是主控 MIPI_TX_0，查电路原理图，对应到屏端的 MIPI lane3，就填写为 MIPI_TX_LANE_3。</p> <p>对应关系不正确，将导致屏幕无法点亮。</p>
output_mode	MIPI Tx 输出模式，默认 OUTPUT_MODE_DSI_VIDEO
video_mode	MIPI Tx 视频模式，默认 BURST_MODE
output_format	MIPI Tx 输出格式，默认 OUT_FORMAT_RGB_24_BIT
sync_info	MIPI Tx 设备的同步信息
pixel_clk	<p>像素时钟，单位为 KHz。</p> <p>计算公式： $\text{pixel_clk} = (\text{htotal} * \text{vtotal}) * \text{fps} / 1000$ 其中： $\text{htotal} = \text{vid_hsa_pixels} + \text{vid_hbp_pixels} + \text{vid_hfp_pixels} + \text{vid_hline_pixels}$ $\text{vtotal} = \text{vid_vsa_lines} + \text{vid_vbp_lines} + \text{vid_vfp_lines} + \text{vid_active_lines}$ fps: 帧率，默认 60 lane_clk 根据 pixel_clk 反推，换算公式： $\text{lane_clk} = \text{pixel_clk} * 24 / 4 / 2$ (24 表示 RGB888 每个 pixel 占 24bits, 4 表示使用了 4 条 Data Lane, 2 表示 mipi clk 是双边沿触发) </p>
lane_pn_swap	<p>MIPI Tx 的 Lane P/N 极是否交换</p> <p>true: 交换</p> <p>false: 不交换</p>

combo_dev_cfg_s 中 sync_info (MIPI Tx 设备的同步信息) 比较难配置，下面详细介绍它的配置方法。一般开始会根据屏厂提供的规格书填写参考值，还有问题再根据现象调整。

sync_info_s 结构体定义

```
struct sync_info_s {
    unsigned short vid_hsa_pixels;
    unsigned short vid_hbp_pixels;
    unsigned short vid_hfp_pixels;
    unsigned short vid_hline_pixels;
    unsigned short vid_vsa_lines;
    unsigned short vid_vbp_lines;
    unsigned short vid_vfp_lines;
```

(下页继续)

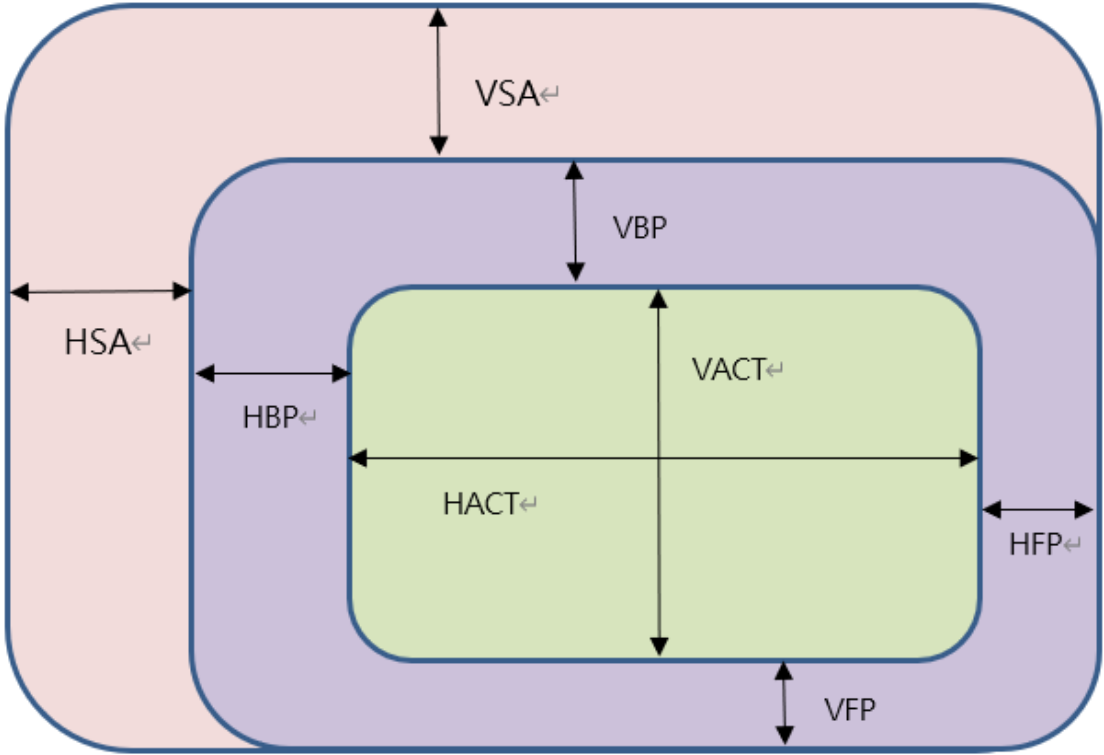
(续上页)

```
unsigned short vid_active_lines;
bool vid_vsa_pos_polarity;
bool vid_hsa_pos_polarity;

};
```

成员名称	描述
vid_hsa_pixels	水平同步脉冲 (HSA), 单位为像素
vid_hbp_pixels	水平消隐后肩 (HBP), 单位为像素
vid_hfp_pixels	水平消隐前肩 (HFP), 单位为像素
vid_hline_pixels	水平有效区 (HACT), 单位为像素
vid_vsa_lines	垂直同步脉冲 (VSA), 单位为行
vid_vbp_lines	垂直消隐后肩 (VBP), 单位为行
vid_vfp_lines	垂直消隐前肩 (VFP), 单位为行
vid_active_lines	垂直有效区 (VACT), 单位为行
vid_vsa_pos_polarity	垂直有效信号的极性, 0 为高有效, 1 为低有效
vid_hsa_pos_polarity	水平有效信号的极性, 0 为高有效, 1 为低有效

MIPI DSI 协议下 MIPI 像素区域示意图



hs_settle_s 结构体定义

```
struct hs_settle_s {
    unsigned char prepare;
```

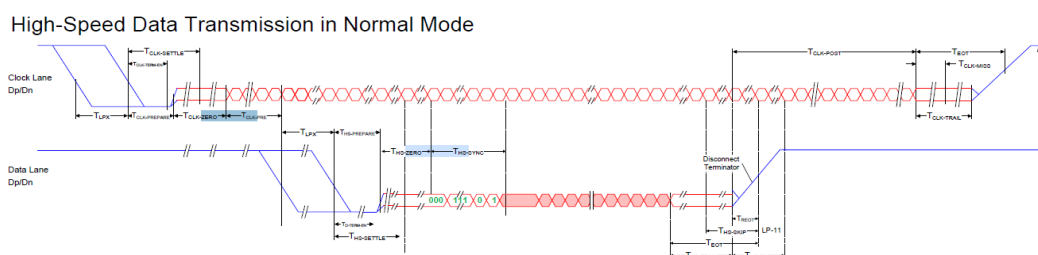
(下页继续)

(续上页)

```
    unsigned char zero;  
    unsigned char trail;  
};
```

成员名称	描述
prepare	MIPI Tx prepare 信号，默认值 6
zero	MIPI Tx zero 信号，默认值 32
trail	MIPI Tx trail 信号，默认值 1

MIPI Tx 时序图



示例：

```
const struct combo_dev_cfg_s dev_cfg = {
    .devno = 0,
    .lane_id = {MIPI_TX_LANE_3, MIPI_TX_LANE_0, MIPI_TX_LANE_CLK, MIPI_TX_LANE_2, MIPI_TX_LANE_1},
    .lane_pn_swap = {false, false, false, false, false},
    .output_mode = OUTPUT_MODE_DSI_VIDEO,
    .video_mode = BURST_MODE,
    .output_format = OUT_FORMAT_RGB_24_BIT,
    .sync_info = {
        .vid_hsa_pixels = 30,
        .vid_hbp_pixels = 100,
        .vid_hfp_pixels = 100,
        .vid_hline_pixels = 800,
        .vid_vsa_lines = 4,
        .vid_vbp_lines = 16,
        .vid_vfp_lines = 10,
        .vid_active_lines = 1280,
        .vid_vsa_pos_polarity = false,
        .vid_hsa_pos_polarity = true,
    },
    .pixel_clk = 80958,
};

const struct hs_settle_s hs_timing_cfg = { .prepare = 6, .zero = 32, .trail = 1 };
```

2.2.1.2 配置屏幕初始化序列

屏幕一般都有初始化的过程，MIPI LCD 屏是通过 MIPI Tx D-PHY 接口来发送指定类型的数据包。初始化序列由屏厂商提供。

屏的初始化序列一般包括像素格式、数据刷新方向、Gamma 配置等，初始化序列中每一个指令具体含义，请在屏厂提供的规格书或者 Driver IC Datasheet 中查找。初始化序列是通过 MIPI Tx 的 Data Lane0 在 LP 模式下发送，发送结束后会切换到 HS 模式。

dsc_instr 结构体定义

```
struct dsc_instr {
    u8 delay;
    u8 data_type;
    u8 size;
    u8 *data;
};
```

屏幕厂商提供的初始化序列一般有寄存器地址和对应的数据，需要根据屏幕厂商给的序列，填充数据类型、数据地址及数据。

成员名称	描述
delay	发送完此命令后，延时的毫秒数
data_type	写命令数据类型，即 DCS(DisplayCommandSet)(指令集) 中的 Data Type。根据数据个数选择数据类型。 类型 1、当只有寄存器地址没有数据时，数据类型选择 0x05； 类型 2、有寄存器地址和一个数据时，数据类型选择 0x15 或者 0x23； 类型 3、有寄存器地址且数据个数大于等于两个，数据类型一般用 0x29 或者 0x39。 一般情况下通用，具体使用请咨询屏幕厂商。
size	寄存器地址和数据个数之和。 例如当只有一个寄存器地址，填 1； 当有一个寄存器地址和 1 个数据填 2；一个寄存器地址和 2 个数据填 3，依此类推。
data	命令数据指针。 寄存器地址和数据。第一个一定是寄存器地址，接在后面的是数据，数据可以没有或者有多个。

注：对于命令数据类型参数的配置的选择，需要咨询厂商。如果没有得到厂商支持，建议没有数据时选择 0x05，有一个数据时选择 0x15，多个数据时选择 0x29。

示例：

```
static u8 data_0[] = { 0xFF, 0x98, 0x81, 0x03 };
static u8 data_1[] = { 0x01, 0x00 };
static u8 data_2[] = { 0x02, 0x00 };
.....
static u8 data_n[] = { 0x11 };
```

(下页继续)

(续上页)

```
static u8 data_XXXX_n+1[] = { 0x29 };

const struct dsc_instr dsi_init_cmds[] = {
    { .delay = 0, .data_type = 0x29, .size = 4, .data = data_XXXX_0 },
    { .delay = 0, .data_type = 0x15, .size = 2, .data = data_XXXX_1 },
    { .delay = 0, .data_type = 0x15, .size = 2, .data = data_XXXX_2 },
    .....
    { .delay = 120, .data_type = 0x05, .size = 1, .data = data_XXXX_n },
    { .delay = 20, .data_type = 0x05, .size = 1, .data = data_XXXX_n+1 },
}
```

2.2.1.3 添加头文件的引用

添加对该新增的头文件的引用，在 u-boot-2021.10/include/cvitek/cvi_panels/cvi_panels.h 中增加对上两节中新增头文件的引用。

示例：

```
#ifdef MIPI_PANEL_HX8394
#include "dsi_hx8394_evb.h"
static struct panel_desc s panel_desc = {
    .panel_name = "HX8394-720x1280",
    .dev_cfg = &dev_cfg_hx8394_720x1280,
    .hs_timing_cfg = &hs_timing_cfg_hx8394_720x1280,
    .dsi_init_cmds = dsi_init_cmds_hx8394_720x1280,
    .dsi_init_cmds_size = ARRAY_SIZE(dsi_init_cmds_hx8394_720x1280)
};
#endif
```

2.2.1.4 配置 MIPI 屏 RESET 管脚

在 u-boot-2021.10/drivers/video/cvitek/cvi_mipi.c 的 mipi_tx_set_combo_dev_cfg 函数中增加 RESET/POWER/BACKLIGHT 的控制。

MIPI 屏一般 RESET 管脚用的是 GPIO 口。所以需要对 GPIO 口进行配置，同时进行屏的复位操作。

- 查询硬件原理图，获取 RESET 管脚对应的管脚名。
- 对照《CV184X_PINOUT_CN》找到管脚对应的 GPIO 组号及序号。
- 修改 build/boards/default/dts/cv184x/cv184x_base.dtsi 中 vo 节点 reset 为对应的值。
- 配置 RESET 所用的 GPIO 的复位操作时序。

屏的复位操作需要参考屏幕的说明书，若无复位操作或者复位的时序与屏幕要求的不匹配，或者电平不匹配，屏幕可能会无法点亮或者工作异常。一般而言会是 high-low-high 的电平变化，具体请参照屏的规格书。

示例：

假设屏幕的 RESET 管脚是 GPIOE 2，复位电压为低，在 build/boards/default/dts/cv184x/cv184x_base.dtsi 修改如下：

```
reset-gpio = <&porte 2 GPIO_ACTIVE_LOW>;
```

配置如下：

```
gpio_request_by_name(dev, "reset-gpio", 0, &priv->ctrl_gpios.disp_reset_gpio,  
GPIO_IS_OUT | GPIO_IS_OUT_ACTIVE);
```

操作如下：

```
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIO_  
→ACTIVE_LOW ? 0 : 1);  
mdelay(10);  
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIO_  
→ACTIVE_LOW ? 1 : 0);  
mdelay(10);  
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIO_  
→ACTIVE_LOW ? 0 : 1);  
mdelay(100);
```

这几句的效果将会是 RESET 管脚产生一个 high-low-high 的电平变化

2.2.1.5 配置 MIPI 屏 POWER 管脚

MIPI 屏的 POWER 控制一般也是用 GPIO。通常只需拉高或拉低管脚电平即可控制 MIPI 屏的供断电。有的屏也可能直接供电，这样的话软件就无需控制。

- 配置方法与上节 RESET 管脚的方法一致

示例：

假设屏幕的 POWER 管脚是 GPIOE 1，工作电压为高，在 build/boards/default/dts/cv184x/cv184x_base.dtsi 修改如下：

```
power-ct-gpio = <&porte 1 GPIO_ACTIVE_HIGH>;
```

如不需要配置 POWER 直接删掉即可。

配置如下：

```
gpio_request_by_name(dev, "power-ct-gpio", 0, &priv->ctrl_gpios.disp_power_ct_gpio,  
GPIO_IS_OUT | GPIO_IS_OUT_ACTIVE);
```

操作如下：

```
dm_gpio_set_value(&ctrl_gpios.disp_power_ct_gpio, ctrl_gpios.disp_power_ct_gpio.flags  
& GPIO_ACTIVE_LOW ? 0 : 1);
```

2.2.1.6 配置 MIPI 屏 BACKLIGHT 管脚

MIPI 屏 BACKLIGHT 可以配置为 GPIO 或者 PWM。

配置为 GPIO

- 配置方法与上节 RESET 管脚的方法一致

示例：

假设屏幕的 PWM 管脚是 GPIOE 0，工作电压为高，在 build/boards/default/dts/cv184x/cv184x_base.dtsi 修改如下：

```
pwm-gpio = <&porte 0 GPIO_ACTIVE_HIGH>;
```

如不需要配置 POWER 直接删掉即可。

配置如下：

```
gpio_request_by_name(dev, "pwm-gpio", 0, &priv->ctrl_gpios, disp_pwm_gpio,  
GPIOD_IS_OUT | GPIOD_IS_OUT_ACTIVE);
```

操作如下：

```
dm_gpio_set_value(&ctrl_gpios.disp_pwm_gpio, ctrl_gpios.disp_pwm_gpio.flags &  
GPIOD_ACTIVE_LOW ? 0 : 1);
```

配置为 PWM

MIPI 屏的 BACKLIGHT 一般通过 PWM，这样可实现亮度调节。

- 查询硬件原理图，获取 BACKLIGHT 管脚对应的管脚名。
- 在 u-boot-2021.10/board/cvitek/cv181x/board.c 的 board_init 函数中，配置 BACKLIGHT 管脚复用功能为 PWM 功能。
- 对照《CV184X Preliminary Datasheet》外围设备 PWM 章节的寄存器信息，配置 PWM 输出的周期、占空比、使能。

PWM 基地址信息，其余寄存器信息具体请参考《CV184X Preliminary Datasheet》，CV184X 有 3 组 PWM，每组 6 个通道

pwm0	0x030010a4
pwm1:	0x030010d0
pwm2:	0x03001058

注意：这里的 PWM0~2 是 PWM 组号，而原理图或者 pinlist 中写的是 PWM0~PWM17，如当看到 PWM1，对应的是上述表格第 0 组的第一个通道 PWM0_1。

示例：

假设屏幕的 BACKLIGHT 管脚是 PWM8

```
devmem 0x030010A4 32 0x00000004 //复用相应的引脚为PWM

echo 2 > /sys/class/pwm/pwmchip6/export //配置待操作的PWM编号

echo 1000000 > /sys/class/pwm/pwmchip6/pwm2/period //
↪设置PWM一个周期的持续时间 (单位ns)

echo 500000 > /sys/class/pwm/pwmchip6/pwm2/duty_cycle //
↪设置一个周期中的" ON" 时间 (单位ns), 即占空比

echo 1 > /sys/class/pwm/pwmchip6/pwm2/enable // 使能 PWM 输出
```

2.2.1.7 配置 u-boot 环境变量

修改 u-boot-2021.10/include/configs/cv184x-asic.h 中的 u-boot 环境变量参数

示例:

```
#define SHOWLOGOCMD LOAD_LOGO CVI_JPEG START_VO START_VL SET_VO_BG
```

LOAD_LOGO 将图片由 MISC 分区读到 DRAM, CVI_JPEG 将图片解析到指定位置, START_VO 和 START_VL 开启 VO 并将 logo 显示在居中位置, SET_VO_BG 设置 VO 背景色, 屏幕除 logo 之外的其他区域由此颜色填充。

2.2.1.8 更换 logo 图片

将客户的 logo 图片放置在该路径下 build/tools/common/bootlogo/, 编译执行 build_all 后会拷贝至 image 生成路径下。

注意: I80 屏幕需要 24bit BMP 图片, 其他需要 YUV420 格式 jpg。

2.2.1.9 编译烧写验证

在上述步骤均完成以后, 重新编译烧写新的 u-boot。上电, 敲回车进入 u-boot 命令行。执行 run showlogo, 顺利的话就可以看到屏幕显示出 logo 图片。如果未显示出 logo, 请确认以下。

- 确认背光点亮
- 确认 RESET 管脚电平状态有达到预期
- 确认屏幕供电正常
- 执行 mw 0x0a094094 0x0701000a, 输出 VO 的 test pattern, 假如屏幕初始化成功, 此时会看到 colorbar

test pattern 寄存器如下图

h94	REG_37			rstn	reg_gra_inv	0	0	1	h0	rw
h94				rstn	reg_pat_en	1	1	1	h0	rw
h94				rstn	reg_auto_en	2	2	1	h0	rw
h94				rstn	reg_dith_en	3	3	1	h1	rw
h94				rstn	reg_snow_en	4	4	1	h0	rw
h94				rstn	reg_fix_mc	5	5	1	h0	rw
h94				rstn	reg_dith_md	10	8	3	h0	rw
h94				rstn	reg_pat_prd	23	16	8	h1	rw
h94				rstn	reg_pat_idx	28	24	5	h0	rw

发现有以上任何异常请回头检查此前的流程是否设置正确及达到预期。

假如以上均未发现异常，建议查看 Driver IC datasheet 或直接咨询屏幕厂商，如何开启屏的 BIST mode，通常是调整初始化序列中的某个寄存器值，会显示 colorbar 等。

假如 BIST mode 不正常，则需要再检查 MIPI Lane 顺序、RESET、POWER、PWM 等是否配置正确，并使用万用表/示波器等确认电路电平状态符合预期，假如均符合预期，则可能是屏幕本身的问题，请咨询屏幕厂商。

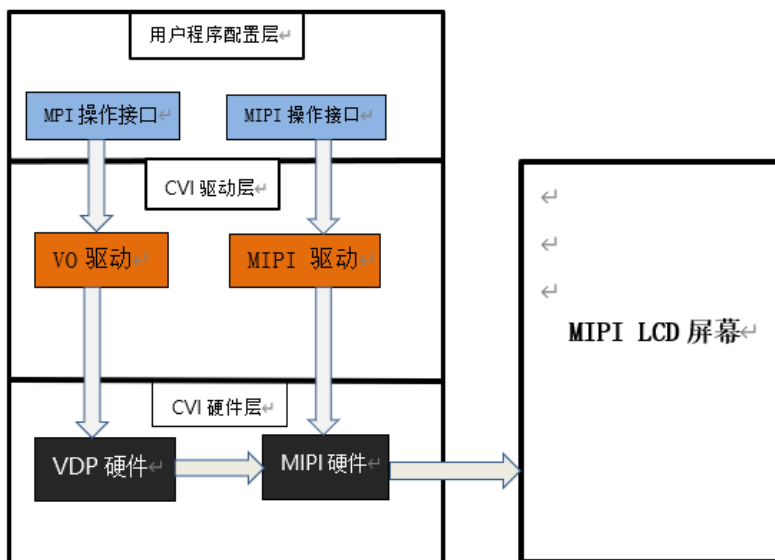
假如 BIST 正常，则说明以上配置正确，硬件电路没有异常，这时通常需要调整 sync_info_s 中的各参数。

2.2.2 在 kernel 中配置 MIPI 屏

在 kernel 中配置 MIPI 屏的方法跟在 u-boot 中几乎是一样的，只是实现流程不一样。当无需显示 logo 的时候，可选择此种方式。

另外，也可以先用 kernel 方式调通，再移植到 u-boot，避免频繁烧写 u-boot。

kernel 中对接 MIPI 屏幕基本框图



2.2.2.1 配置 MIPI Tx 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在路径 `cvi_mpi/component/panel/` 下，客户可以参照其余的头文件模板新增自己的 `panel` 头文件。

参见 2.2.1.1 节

2.2.2.2 配置屏幕初始化序列

参见 2.2.1.2 节

2.2.2.3 添加头文件的引用

添加对该新增的头文件的引用，在 `cvi_mpi/sample_app/panel/sample_panel.c` 中增加对上两节中新增头文件的引用。

示例：

如要点屏 HX8394_EVB，首先在 `cvi_mpi/sample_app/panel/sample_panel.h` 中 include 该屏幕的头文件 `dsi_hx8394_evb.h`，然后确保 `cvi_mpi/sample_app/panel/sample_panel.c` 文件中的字符数组 `static char *s_panel_model_type_arr[]` 里有“HX8394_EVB”字符，没有则自己添加，接着在该文件屏幕枚举类型 `PANEL_MODEL` 中添加与该字符索引值相等的枚举量 `HX8394_EVB`，最后在 `SAMPLE_SET_PANEL_DESC` 函数中添加对该屏幕相关参数进行调用的 `case`，如下所示：

```
case DSI_PANEL_HX8394_EVB:
    g_panel_desc.panel_type = PANEL_MODE_DSI;
    g_panel_desc.stdsicfg.dev_cfg = &dev_cfg_hx8394_720x1280;
    g_panel_desc.stdsicfg.hs_timing_cfg = &hs_timing_cfg_hx8394_720x1280;
    g_panel_desc.stdsicfg.dsi_init_cmds = dsi_init_cmds_hx8394_720x1280;
    g_panel_desc.stdsicfg.dsi_init_cmds_size = ARRAY_SIZE(dsi_init_cmds_hx8394_720x1280);
    break;
```

2.2.2.4 配置 MIPI 屏 RESET、POWER、BACKLIGHT 管脚

方法 1：

在路径 `build/boards/default/dts/cv184x/` 下找到对应的 `dtb` 文件，配置 MIPI Tx 的 `gpio` 信息，如果没有该管脚，则直接不写即可。

示例：

```
mipi_tx: mipi_tx {
    compatible = "cvitek,mipi_tx";
    reset-gpio = <&porte 2 GPIO_ACTIVE_LOW>;
    pwm-gpio = <&porte 0 GPIO_ACTIVE_HIGH>;
    power-ct-gpio = <&porte 1 GPIO_ACTIVE_HIGH>;
    clocks = <&clk CV181X_CLK_DISP_VIP>, <&clk CV181X_CLK_DSI_MAC_VIP>;
    clock-names = "clk_disp", "clk_dsi";
};
```

说明：

```
pwm-gpio = <&porte 0 GPIO_ACTIVE_HIGH>;
```

为调试方便，背光可先用 GPIO 控制，切记先不要在 u-boot 中配置 pinmux 为 PWM 功能，否则可能无法控制。

后续根据需求，如果需要调节亮度，再在 u-boot 中配置 pinmux 功能为 PWM，删除 dts 中的此行，同时 app 中用 PWM 方式控制。

系统启动后加载 MIPI Tx 驱动方式（一般会自动加载，可先用 lsmod 查看是否已加载）：

```
insmod /mnt/system/ko/cvi_mipi_tx.ko
```

这样当驱动加载后，会根据 dts 中的 GPIO 信息，自动申请这些 GPIO，并初始化成对应的电平状态。

方法 2：

无需修改内核 dts 文件。

系统启动加载 MIPI Tx 驱动方式：

```
insmod /mnt/system/ko/cvi_mipi_tx.ko gpio=354,0,353,1,352,1
```

这三个 GPIO 依序分别为 RESET、POWER、PWM

当驱动加载后，驱动会优先使用 gpio 参数中的信息自动申请 GPIO 号对应的 GPIO，并初始化成其后的电平状态。如果没有写 gpio 参数，驱动会根据 dts 中的 GPIO 信息申请 GPIO。如果没有该管脚，则 GPIO 号和电平状态均写-1 即可。

同样，为调试方便，背光可先用 GPIO 控制，先不要在 u-boot 中配置 pinmux 为 PWM 功能。后续需要调节亮度，再在 u-boot 中配置 pinmux 功能为 PWM，同时 app 中用 PWM 方式控制，并将第三个 GPIO 号和电平状态写成-1 即可。

方法 3：

直接应用层直接控制 GPIO。

示例：假设 reset: GPIOE2, pwm: GPIOE0, power: GPIOE1, 可以用 cat /sys/kernel/debug/gpio 指令查看芯片的 gpio 配置，可依次得到 gpiochip0 至 gpiochip4 的编号范围，依次对应着 GPIOA 至 GPIOE 的编号范围，若得出 gpiochip4 的范围是 352 至 383，则对于所要拉的 GPIOE2 来说，下面 echo 操作的号码可由 352+2=354 得到。

因此需要以下操作去拉对应引脚：

```
1. echo 352 > /sys/class/gpio/export
   echo 353 > /sys/class/gpio/export
   echo 354 > /sys/class/gpio/export
2. echo out > /sys/class/gpio/gpio352/direction
   echo out > /sys/class/gpio/gpio353/direction
   echo out > /sys/class/gpio/gpio354/direction
3. echo 1 > /sys/class/gpio/gpio352/value
   echo 1 > /sys/class/gpio/gpio353/value
   echo 1 > /sys/class/gpio/gpio354/value
   echo 0 > /sys/class/gpio/gpio354/value
   echo 1 > /sys/class/gpio/gpio354/value
```

说明：

为调试方便，背光可先用 GPIO 控制，切记先不要在 u-boot 中配置 pinmux 为 PWM 功能，否则可能无法控制。

后续根据需求，如果需要调节亮度，在 u-boot 中配置 pinmux 功能为 PWM，同时 app 中用 PWM 方式控制。

2.2.2.5 编译验证

执行 build_middleware 编译 middleware，在路径 cvi_mpi/sample_app/panel/下会生成 sample_panel 可执行文件。该程序和 u-boot 中 “startvo 0 65536 0” 做的事情是一样的，切换到 LP 模式，设置 MIPI Tx 设备属性并通过 Data Lane0 向屏幕发送初始化序列，然后切回 HS 模式。

将 sample_panel 拷贝至设备，执行命令 ./sample_panel 后会弹出该命令的执行方式，按提示运行即可。

示例：./sample_panel -panel=HX8394_EVB

说明：

RESET 初始电平设置为 low，将需要 high-low-high 时序变化。

RESET 初始电平设置为 high，将需要 low-high-low 时序变化。

使能 VO 的 test pattern，寄存器如下图。执行 devmem 0x0a094094 32 0x0701000a 将会看到 colorbar。

h94	REG_37			rstn	reg_gra_inv	0	0	1	h0	rw
h94				rstn	reg_pat_en	1	1	1	h0	rw
h94				rstn	reg_auto_en	2	2	1	h0	rw
h94				rstn	reg_dith_en	3	3	1	h1	rw
h94				rstn	reg_snow_en	4	4	1	h0	rw
h94				rstn	reg_fix_mc	5	5	1	h0	rw
h94				rstn	reg_dith_md	10	8	3	h0	rw
h94				rstn	reg_pat_prd	23	16	8	h1	rw
h94				rstn	reg_pat_idx	28	24	5	h0	rw

若 colorbar 未正常显示，请回头检查此前的流程是否设置正确及达到预期。

假如此前流程均未发现异常，建议查看 Driver IC datasheet 或直接咨询屏幕厂商，如何开启屏的 BIST mode，通常是调整初始化序列中的某个寄存器值，会显示 colorbar 等。

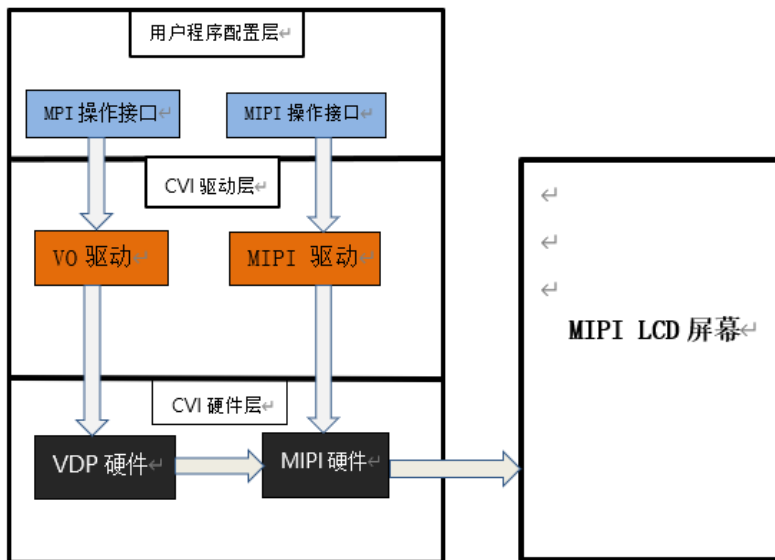
假如 BIST mode 不正常，则需要再检查 MIPI Lane 顺序、RESET、POWER、PWM 等是否配置正确，并使用万用表/示波器等确认电路电平状态符合预期，假如均符合预期，则可能是屏幕本身的问题，请咨询屏幕厂商。

假如 BIST 正常，则说明以上配置正确，硬件电路没有异常，这时通常需要调整 sync_info_s 中的各参数。

2.2.3 在双系统中配置 MIPI 屏

在双系统中配置 MIPI 屏的方法和在上述单系统中几乎是一样的，只是有一些小区别。下面说明一下其中的异同。

对接 MIPI 屏幕基本框图



其中，上图的 VO 与 MIPI 驱动均在小核 alios 中完成。

2.2.3.1 配置 MIPI Tx 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在路径 `cvi_mpi/component/panel/` 和 `cvi_alios/components/cvi_mmf_sdk/cvi_panel/` 下，客户可以参照其余的头文件模板新增自己的 panel 头文件。

参见 2.2.1.1 节

2.2.3.2 打开 cvi_alios 中的配置开关

在 `cvi_alios/solutions/{normboot/fastboot}/customization/{project_config}/package.yaml` 中打开对应芯片和屏幕类型的开关，

示例，若选择 `CONFIG_CV1842HP_GC8613` 芯片，`CONFIG_PANEL_HX8394` 屏幕，则打开如下开关：

```
CONFIG_CV1842HP_GC8613: 1
```

```
CONFIG_PANEL_HX8394: 1
```

注意 `CONFIG_CHIP_cv1842hp` 芯片若需同时支持 sensor 和 panel，需改装硬件，并按需修改 `cvi_alios/solutions/normboot/customization/cv1842hp_gc8613/src/custom_platform.c` 中 `_MipiTxPinmux()` 接口里的引脚复用。

2.2.3.3 配置屏幕初始化序列

参见 2.2.1.2 节

2.2.3.4 添加头文件的引用

参见 2.2.2.3 节

2.2.3.5 配置 MIPI 屏 RESET、POWER、BACKLIGHT 管脚

在 `cvi_alios/solutions/normboot/customization/cv1842hp_gc8613/src/custom_platform.c` 中 `_MipiTxPinmux` 接口里添加对 RESET、POWER、BACKLIGHT 管脚的复用，然后直接应用层直接控制 GPIO。

示例：

对于 `CONFIG_CHIP_cv1842hp` 芯片，一般要拉引脚为：reset: GPIOE2, pwm: GPIOE0, power: GPIOE01。

因此需要以下操作去拉对应引脚：

```
devmem 0x03022004 32 0x0
devmem 0x03022000 32 0x0
echo 352 > /sys/class/gpio/export
echo 353 > /sys/class/gpio/export
echo 354 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio352/direction
echo out > /sys/class/gpio/gpio353/direction
echo out > /sys/class/gpio/gpio354/direction
echo 1 > /sys/class/gpio/gpio354/value
echo 1 > /sys/class/gpio/gpio352/value
echo 1 > /sys/class/gpio/gpio353/value
echo 0 > /sys/class/gpio/gpio354/value
echo 1 > /sys/class/gpio/gpio354/value
```

对于 `CONFIG_CHIP_cv1842cp` 芯片，一般要拉引脚为：reset: GPIOA15, pwm: GPIOA18, power: GPIOA19。

因此需要以下操作去拉对应引脚：

```
devmem 0x0300103c 32 0x3
devmem 0x03001068 32 0x3
devmem 0x03001064 32 0x3
echo 495 > /sys/class/gpio/export
echo 498 > /sys/class/gpio/export
echo 499 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio495/direction
echo out > /sys/class/gpio/gpio498/direction
echo out > /sys/class/gpio/gpio499/direction
echo 1 > /sys/class/gpio/gpio495/value
echo 1 > /sys/class/gpio/gpio498/value
echo 1 > /sys/class/gpio/gpio499/value
echo 0 > /sys/class/gpio/gpio495/value
echo 1 > /sys/class/gpio/gpio495/value
```

上述代码中 `devmem` 写寄存器指令是为了复用相应的引脚，若已完成相关复用，可不执行该指令。

2.2.3.6 编译验证

参见 2.2.2.5 节

3 LVDS

概述

Low Voltage Differential Signal(LVDS)，即低电压差分信号。LVDS 接口又称 RS644 总线接口，1994 年由美国国家半导体公司 (NS) 提出的为克服以 TTL 电平方式传输宽带高码率数据时功耗大、EMI 电磁干扰大等缺点而研制的一种视频信号传输模式，是一种电平标准，广泛应用于液晶屏接口。LVDS 屏总体和 MIPI 类似，但是还有一些区别。本章节介绍如何在 CVITEK 处理器解决方案上开发调试 LVDS LCD 屏。

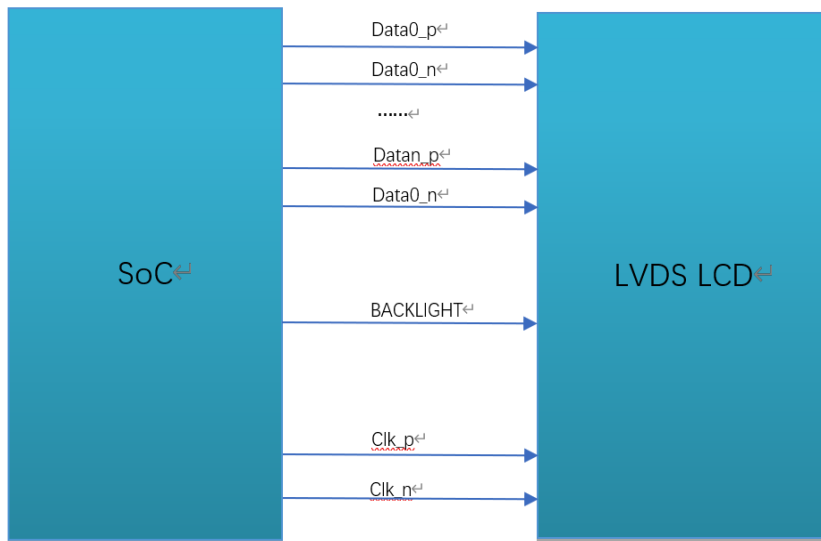
3.1 环境准备

3.1.1 LVDS 屏幕接口介绍

LVDS 屏幕一般有以下几种信号，如图所示。

- LVDS 时钟线 (CLK)
- LVDS 数据线 (DATA) (单路 6bit: 3 lane, 单路 8bit: 4 lane, 单路 10bit: 5 lane, 双路 6bit: 6 lane, 双路 6bit: 8 lane, 双路 6bit: 10 lane, 目前仅支持单路 6bit 和单路 8bit)
- 背光控制信号 (BACKLIGHT)

LVDS 接口连线示意图



3.1.2 硬件连线确认

检查硬件连线，确认无异常。具体有些引脚差异，需对照屏幕厂商提供的规格书及电路原理图确认。

3.2 配置 LVDS 屏

根据上节环境准备的内容，在接口和连线上了解了屏幕对接的配置，在这一节中将对屏幕对接时在软件方面需要进行的配置进行说明。

CVITEK 有两种方案进行 LVDS 屏幕的对接，和 MIPI 屏类似，分别是在 u-boot 及 kernel 中进行屏的初始化。实际应用中根据需求二者选其一。

3.2.1 在 u-boot 中配置 LVDS 屏

u-boot 中配置 MIPI 屏是通过 CVITEK 开发的 showlogo 命令，设备上电后，敲回车进入 u-boot 命令行，printenv 可以看到 showlogo 命令，bootcmd 在引导内核之前会执行该命令进行屏的初始化并显示 logo。

示例：

```
showlogo=mmc dev 0;mmc read 0x84080000 0xA000 0x400; cvi_jpeg 0x84080000 0x81800000 0x80000; ↵  
↵startvo 0 2048 0;startvl 0 0x84080000 0x81800000 0x80000 16;setvobg 0 0xffffffff
```

注：单路 6bit 为 1024，单路 8bit 为 2048，单路 10bit 为 4096。

本文档重点讲解屏的初始化部分，显示 logo 具体请参考《CVITEK 开机画面使用指南》。其中，屏的初始化部分在“startvo 0 2048 0”中实现。

3.2.1.1 配置 LVDS 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在路径

u-boot-2021.10/include/cvitek/cvi_panels/下，客户可以参照其余的头文件模板新增自己的 panel 头文件。

cvi_lvds_cfg_s 结构体定义

```
struct cvi_lvds_cfg_s {  
    enum LVDS_OUT_BIT out_bits;  
    enum LVDS_MODE mode;  
    unsigned char chn_num;  
    bool data_big_endian;  
    enum lvds_lane_id lane_id[LANE_MAX_NUM];  
    bool lane_pn_swap[LANE_MAX_NUM];  
    struct sync_info_s sync_info;  
    unsigned short u16FrameRate;  
    unsigned int pixelclock;  
};
```

成员名称	描述
out_bits	LVDS_OUT_6BIT、LVDS_OUT_8BIT、LVDS_OUT_10BIT
mode	LVDS_MODE_JEI DA、LVDS_MODE_VESA，一般设置为 LVDS_MODE_VESA
chn_num	通道数 1、2，现在处理器仅支持 1 通道
data_big_endian	发送数据的大小端顺序，一般设置 false
Lane_id	<p>主控端和屏端 Lane 号的对应关系，未使用的 Lane 填-1 即可。</p> <p>共 5 个成员，依序分别代表主控端的 VO_LVDS_LANE_0 ~ VO_LVDS_LANE_4，实际填写的内容需要根据对应到屏端的 LVDS lane 号。</p> <p>例如，第一个成员是主控 LANE 0，查电路原理图，对应到屏端 lane 3，就填写为 VO_LVDS_LANE_3。</p> <p>对应关系不正确，将导致屏幕无法点亮。</p>
lane_pn_swap	<p>LVDS 的 Lane P/N 极是否交换</p> <p>true: 交换</p> <p>false: 不交换</p>
sync_info	LVDS 设备的同步信息
pixel_clk	<p>像素时钟，单位为 KHz。</p> <p>计算公式：</p> $\text{pixel_clk} = (\text{htotal} * \text{vtotal}) * \text{fps} / 1000$ <p>其中：</p> $\text{htotal} = \text{vid_hsa_pixels} + \text{vid_hbp_pixels} + \text{vid_hfp_pixels} + \text{vid_hline_pixels}$ $\text{vtotal} = \text{vid_vsa_lines} + \text{vid_vbp_lines} + \text{vid_vfp_lines} + \text{vid_active_lines}$ <p>fps: 帧率，默认 60</p> <p>lane_clk 根据 pixel_clk 反推，换算公式：</p> $\text{lane_clk} = \text{pixel_clk} * 24 / 4 / 2$ <p>(24 表示 RGB888、单路 8bit，每个 pixel 占 24bits，4 表示使用了 4 条 Data Lane，2 表示 lvds clk 是双边沿触发)</p>

示例：

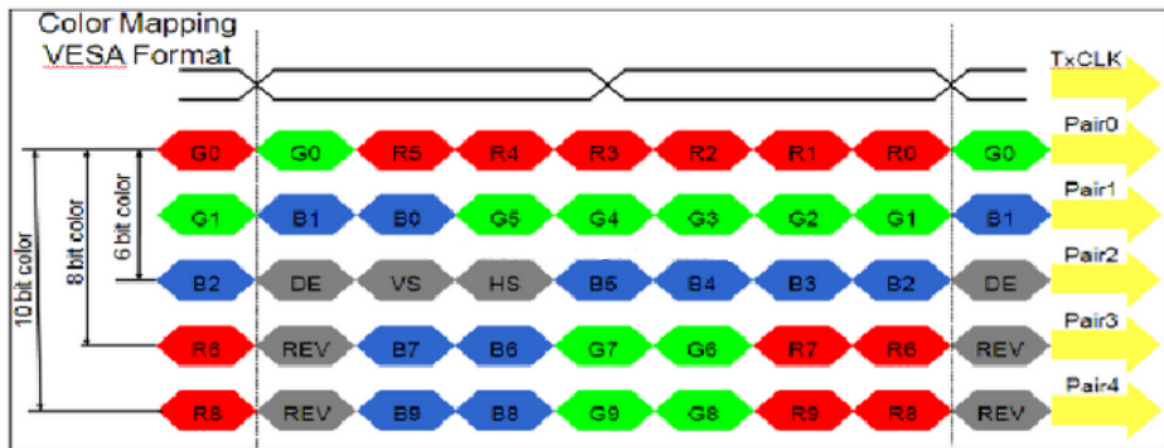
```

struct cvi_lvds_cfg_s lvds_ek79202_cfg = {
    .mode = LVDS_MODE_VESA,
    .out_bits = LVDS_OUT_8BIT,
    .chn_num = 1,
    .lane_id = {VO_LVDS_LANE_0, VO_LVDS_LANE_1, VO_LVDS_LANE_2, VO_LVDS_LANE_3, VO_LVDS_LANE_CLK},
    .lane_pn_swap = {false, false, false, false, false},
    .sync_info = {
        .vid_hsa_pixels = 10,
        .vid_hbp_pixels = 88,
        .vid_hfp_pixels = 62,
        .vid_hline_pixels = 1280,
        .vid_vsa_lines = 4,
        .vid_vbp_lines = 23,
        .vid_vfp_lines = 11,
        .vid_active_lines = 800,
        .vid_vsa_pos_polarity = 0,
        .vid_hsa_pos_polarity = 0,
    },
    .u16FrameRate = 60,
    .pixelclock = 72403,
};

```

sync_info_s 结构体定义

与 MIPI 类似，参见 2.2.1.1。



3.2.1.2 添加头文件的引用

添加对该新增的头文件的引用，在 u-boot-2021.10/include/cvitek/cvi_panels/cvi_panels.h 中增加对上一节中新增头文件的引用。

示例：

```

#ifdef LVDS_PANEL_EK79202
#include "lvds_ek79202.h"
static struct panel_desc_s panel_desc = {

```

(下页继续)

(续上页)

```
.lvds_cfg = &lvds_ek79202_cfg
};
#endif
```

3.2.1.3 配置 LVDS 屏 BACKLIGHT 管脚

LVDS 屏的 BACKLIGHT 可以配置为 GPIO 或者 PWM。

配置为 GPIO

可通过修改 build/boards/cv184x/cv184xxx/u-boot/cvitek.h 中 VO_GPIO_PWM_PORT、VO_GPIO_PWM_INDEX、VO_GPIO_PWM_ACTIVE 实现。

配置为 PWM

一般通过 PWM，这样可实现亮度调节。实现与 MIPI 屏类似，参见 2.2.1.6 小节。

3.2.1.4 配置 u-boot 环境变量

实现与 MIPI 屏类似，参见 2.2.1.7 小节。

3.2.1.5 更换 logo 图片

实现与 MIPI 屏类似，参见 2.2.1.8 小节。

3.2.1.6 编译烧写验证

在上述步骤均完成以后，重新编译烧写新的 u-boot。上电，敲回车进入 u-boot 命令行。执行 run showlogo，顺利的话就可以看到屏幕显示出 logo 图片。如果未显示出 logo，请确认以下。

- 确认背光点亮
- 确认屏幕供电正常
- 执行 mw 0x0a088094 0x0701000a，输出 VO 的 test pattern，假如屏幕初始化成功，此时会看到 colorbar

test pattern 寄存器如下图

h94	REG_37			rstn	reg_gra_inv	0	0	1	h0	rw
h94				rstn	reg_pat_en	1	1	1	h0	rw
h94				rstn	reg_auto_en	2	2	1	h0	rw
h94				rstn	reg_dith_en	3	3	1	h1	rw
h94				rstn	reg_snow_en	4	4	1	h0	rw
h94				rstn	reg_fix_mc	5	5	1	h0	rw
h94				rstn	reg_dith_md	10	8	3	h0	rw
h94				rstn	reg_pat_prd	23	16	8	h1	rw
h94				rstn	reg_pat_idx	28	24	5	h0	rw

发现有以上任何异常请回头检查此前的流程是否设置正确及达到预期。

假如以上均未发现异常，则需要再检查 LVDS Lane 顺序、PWM 等是否配置正确，并使用万用表/示波器等确认电路电平状态符合预期，假如均符合预期，则可能是屏幕本身的问题，请咨询屏幕厂商。

如以上配置正确，硬件电路没有异常，这时通常需要调整 `sync_info_s` 中的各参数。

3.2.2 在 kernel 中配置 LVDS

在 kernel 中配置 LVDS 屏的方法跟在 u-boot 中几乎是一样的，只是实现流程不一样。当无需显示 logo 的时候，可选择此种方式。

另外，也可以先用 kernel 方式调通，再移植到 u-boot，避免频繁烧写 u-boot。

3.2.2.1 配置 LVDS 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在路径 `cvi_mpi/component/panel/` 下，客户可以参照其余的头文件模板新增自己的 panel 头文件。

参见 3.2.1.1 节。

3.2.2.2 添加头文件的引用

添加对该新增的头文件的引用，在 `cvi_mpi/sample_app/sample_panel/sample_panel.c` 中增加对上两节中新增头文件的引用。

示例：

如要点亮屏幕 LCM185X56，首先在 `cvi_mpi/sample_app/sample_panel/sample_panel.h` 中 `include` 该屏幕的头文件 `lvds_lcm185x56.h`，然后确保 `cvi_mpi/sample_app/sample_panel/sample_panel.c` 文件中的字符数组 `static char *s_panel_model_type_arr[]` 里有“LCM185X56”字符，没有则自己添加。接着在该文件屏幕枚举类型 `PANEL_MODEL` 中添加与该字符索引值相等的枚举量 `LVDS_PANEL_LCM185X56`，最后在 `SAMPLE_SET_PANEL_DESC` 函数中添加对该屏幕相关参数进行调用的 `case`，如下所示：

```
case LVDS_PANEL_LCM185X56:
    g_panel_desc.panel_type = PANEL_MODE_LVDS;
    g_panel_desc.stVoPubAttr.enIntfType = VO_INTF_LCD_24BIT;
    g_panel_desc.stVoPubAttr.enIntfSync = VO_OUTPUT_USER;
    VO_SYNC_INFO_S stLcm185x56_SyncInfo = {.bSynm = 1, .blop = 1, .u16FrameRate = 60
    , .u16Vact = 768, .u16Vbb = 20, .u16Vfb = 10
    , .u16Hact = 1366, .u16Hbb = 100, .u16Hfb = 88
    , .u16Vpw = 2, .u16Hpw = 20, .bIdv = 0, .bIhs = 0, .bIvs = 0};
    g_panel_desc.stVoPubAttr.stSyncInfo = stLcm185x56_SyncInfo;
    g_panel_desc.stVoPubAttr.stLvdsAttr = lvds_lcm185x56_cfg;
    break;
```

其中，`enIntfType` 可以根据实际需求选择 `VO_INTF_LCD_18BIT`、`VO_INTF_LCD_24BIT` 或 `VO_INTF_LCD_30BIT`，`enIntfSync` 可以参考《CV184x 媒体软件开发指南》查看支持的

其他输出时序类型，也可以选择自定义时序模式 `VO_OUTPUT_USER`，自定义时序时需填写 `stSyncInfo`，类似本示例中的 `stLcm185x56_SyncInfo`。

3.2.2.3 配置 LVDS 屏 BACKLIGHT 管脚

在路径 `cvi_mpi/component/panel/` 下找到对应的头文件，配置 LVDS 的 `gpio` 信息，如果没有该管脚或者由 APP 控制，则直接不写或者 `gpio_num` 赋值为 -1 即可。

示例：

```
.backlight_pin = {  
    .gpio_num = GPIOE_02,  
    .active = GPIO_ACTIVE_HIGH,  
},
```

说明：

为调试方便，背光可先用 GPIO 控制，切记先不要在 u-boot 中配置 `pinmux` 为 PWM 功能，否则可能无法控制。

后续根据需求，如果需要调节亮度，再在 u-boot 中配置 `pinmux` 功能为 PWM，删除头文件中的此配置或者 `gpio_num` 赋值为 -1，同时 APP 中用 PWM 方式控制。

3.2.2.4 编译验证

执行 `build_middleware` 编译 `middleware`，在路径 `cvi_mpi/component/panel/` 下会生成 `sample_panel` 可执行文件。该程序和和在 u-boot 中 “`startvo 0 2048 0`” 做的事情是一样的，切换到 LP 模式向屏幕发送初始化序列，然后切回 HS 模式。

将 `sample_panel` 拷贝至设备，执行命令 `./sample_panel` 后会弹出该命令的执行方式，按提示运行即可。

示例：`./sample_panel -panel=LCM185X56`

如未能正常显示，可参见 3.2.1.6.

3.2.3 在双系统中配置 LVDS

在双系统中配置 LVDS 屏的方法跟在 kernel 中几乎是一样的，下面简要说明一下异同。

3.2.3.1 配置 LVDS 设备属性

根据屏的规格书，实现每个屏的配置头文件，并放置在路径 `cvi_mpi/component/panel/` 下，客户可以参照其余的头文件模板新增自己的 `panel` 头文件。

参见 3.2.1.1 节。

3.2.3.2 打开 cvi_alios 中的配置开关

在 `cvi_alios/solutions/normboot/package_yamls/package.yaml.turnkey` 中打开对应芯片和屏幕类型的开关，

示例，选择 `CONFIG_BOARD_CV181XH` 芯片，打开如下开关：

```
CONFIG_BOARD_CV181XC: 0
CONFIG_BOARD_CV181XH: 1
```

3.2.3.3 添加头文件的引用

请参考 3.2.2.2 节。

3.2.3.4 配置 LVDS 屏 BACKLIGHT 管脚

在 `cvi_alios/solutions/normboot/customization/cv1811c_cv2003_1l_triple/src/custom_platform.c` 中 `_MipiTxPinmux` 接口里添加对 RESET、POWER、BACKLIGHT 管脚的复用，然后直接应用层直接控制 GPIO。

示例：

对于 `CONFIG_BOARD_CV181XH` 芯片，一般要拉引脚为：reset: GPIOE2, pwm: GPIOE0, power: GPIOE01。

因此需要以下操作去拉对应引脚：

```
devmem 0x03022004 32 0x0
devmem 0x03022000 32 0x0
echo 352 > /sys/class/gpio/export
echo 353 > /sys/class/gpio/export
echo 354 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio352/direction
echo out > /sys/class/gpio/gpio353/direction
echo out > /sys/class/gpio/gpio354/direction
echo 1 > /sys/class/gpio/gpio354/value
echo 1 > /sys/class/gpio/gpio352/value
echo 1 > /sys/class/gpio/gpio353/value
```

3.2.3.5 编译验证

请参考 3.2.2.4 节。

4 MIPI DSI(SDK V6.3.4 开始)

概述

Display Serial Interface (DSI) 接口是移动行业处理器接口联盟 (Mobile Industry Processor Interface alliance, MIPI 联盟) 定义的一种高速串行接口, 主要用于处理器和显示模块之间的连接。

本章介绍如何在 CVITEK 处理器解决方案上开发调试 MIPI LCD 屏, 以帮助客户有序快速开发 MIPI LCD 业务。

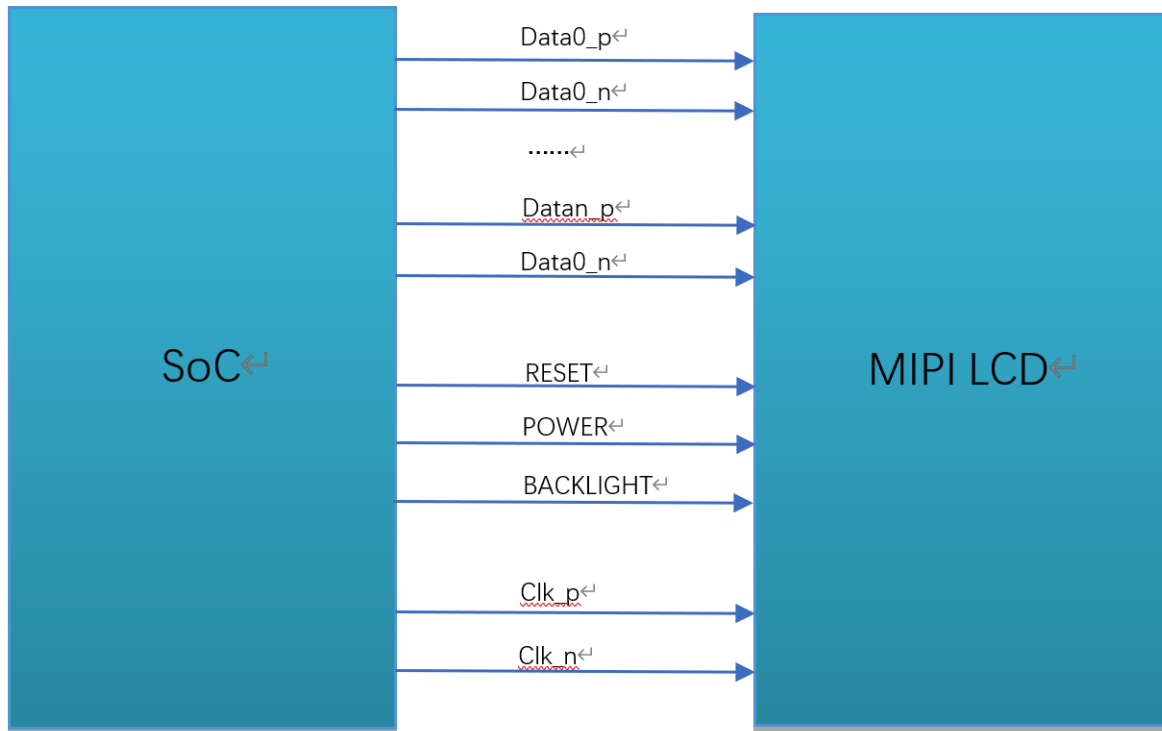
4.1 环境准备

4.1.1 MIPI DSI 屏幕接口介绍

MIPI DSI 屏幕一般有以下几种信号, 如图所示。

- MIPI 时钟线 (CLK)
- MIPI 数据线 (DATA), 最大为 4 Lane (仅可以为 1/2/4 Lane)
- 背光控制信号 (BACKLIGHT)
- 复位引脚 (RESET)
- Panel 电源供电 (POWER)

MIPI DSI 接口连线示意图



4.1.2 硬件连线确认

检查硬件连线，确认无异常。具体有些引脚差异，需对照屏幕厂商提供的规格书及电路原理图确认。

4.2 PanelSupportList 框架概览

SDK 通过 PanelSupportList 仓库统一管理所有屏参头文件，u-boot、cvi_mpi 和 AliOS 共用同一套屏参数据，但配置方式各有不同。

框架结构如下：

- **屏参头文件**：全部放在 PanelSupportList/panels/ 目录下，按规范命名。

三种系统的屏参引用方式：

- **u-boot / cvi_alios**：通过 PanelSupportList/cvi_panels.h 汇总入口，使用 CONFIG_ 宏在编译期选定唯一一块屏。cvi_panels.h 只支持单选，不能同时使能多块屏。
- **cvi_mpi**：在 sample_panel.h 中直接 #include 所有屏参头文件，运行时通过 --panel= 命令行参数动态选择，可同时编译进所有屏参。

对接一块新 MIPI 屏的基本步骤：

1. 在 PanelSupportList/panels/ 下编写规范命名的屏参头文件。
2. 根据目标系统选择注册方式（见下文）。

3. 配置使能并构建。

4.3 配置 MIPI 屏（统一流程）

4.3.1 第 1 步：编写屏参头文件

4.3.1.1 命名规范

头文件统一放在 PanelSupportList/panels/ 下，命名格式：

```
dsi_{driveric}_{wxh}_{moduleid}_{lanenum}lane_{fps}fps[_vN].h
```

- driveric: 驱动 IC，小写，例如 hx8394。
- wxh: 有效分辨率，例如 720x1280。
- moduleid: 屏幕模组 ID；不确定时用 NULL。
- lanenum: 物理 DSI lane 数，例如 4lane。
- fps: 目标刷新率，例如 60fps。
- _vN (可选): 当同一 IC + 分辨率 + lane + fps 仍需区分不同硬件版本时使用。

示例：

```
dsi_hx8394_720x1280_NULL_4lane_60fps.h
dsi_st7701_480x640_NULL_2lane_60fps.h
```

4.3.1.2 头文件内容模板

每个头文件必须包含 panel_platform.h（提供平台适配层和 DSI_CMD 等宏），然后按顺序定义 timing 宏、设备配置、HS timing 和初始化命令。

以下是一个完整的范例：

```
#include "panel_platform.h"

/* ---- Timing 宏 ---- */
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_VACT 1280
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_VSA 16
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_VBP 4
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_VFP 6
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_HACT 720
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_HSA 64
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_HBP 36
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_HFP 128
#define DSI_HX8394_720X1280_NULL_4LANE_60FPS_FPS 60

/* ---- MIPI Tx 设备配置 ---- */
struct combo_dev_cfg_s dev_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps = {
    .devno = 0,
```

(下页继续)

(续上页)

```

.lane_id = {MIPI_TX_LANE_0, MIPI_TX_LANE_1, MIPI_TX_LANE_CLK,
            MIPI_TX_LANE_2, MIPI_TX_LANE_3},
.lane_pn_swap = {true, true, true, true, true},
.output_mode = OUTPUT_MODE_DSI_VIDEO,
.video_mode = BURST_MODE,
.output_format = OUT_FORMAT_RGB_24_BIT,
.sync_info = {
    .vid_hsa_pixels = DSI_HX8394_720X1280_NULL_4LANE_60FPS_HSA,
    .vid_hbp_pixels = DSI_HX8394_720X1280_NULL_4LANE_60FPS_HBP,
    .vid_hfp_pixels = DSI_HX8394_720X1280_NULL_4LANE_60FPS_HFP,
    .vid_hline_pixels = DSI_HX8394_720X1280_NULL_4LANE_60FPS_HACT,
    .vid_vsa_lines = DSI_HX8394_720X1280_NULL_4LANE_60FPS_VSA,
    .vid_vbp_lines = DSI_HX8394_720X1280_NULL_4LANE_60FPS_VBP,
    .vid_vfp_lines = DSI_HX8394_720X1280_NULL_4LANE_60FPS_VFP,
    .vid_active_lines = DSI_HX8394_720X1280_NULL_4LANE_60FPS_VACT,
    .vid_vsa_pos_polarity = false,
    .vid_hsa_pos_polarity = true,
},
.pixel_clk = PIXEL_CLK(DSI_HX8394_720X1280_NULL_4LANE_60FPS),
};

/* ---- HS timing 配置 ---- */
struct hs_settle_s hs_timing_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps = {
    .prepare = 6, .zero = 32, .trail = 1
};

/* ---- 初始化命令 ---- */
struct dsc_instr dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_60fps[] = {
    DSI_CMD(0, 0xb9, 0xff, 0x83, 0x94),
    DSI_CMD(0, 0xb1, 0x50, 0x15, 0x75, 0x09, 0x32, 0x44, 0x71, 0x31,
            0x4d, 0x2f, 0x56, 0x73, 0x02, 0x02),
    /* ... 更多初始化命令 ... */
    DSI_CMD(120, 0x11), /* Sleep Out, delay 120ms */
    DSI_CMD(20, 0x29), /* Display On, delay 20ms */
};

```

4.3.1.3 关键数据结构的说明

combo_dev_cfg_s (MIPI Tx 设备属性)

```

struct combo_dev_cfg_s {
    unsigned int devno;
    enum mipi_tx_lane_id lane_id[LANE_MAX_NUM];
    enum output_mode_e output_mode;
    enum video_mode_e video_mode;
    enum output_format_e output_format;
    struct sync_info_s sync_info;
    unsigned int pixel_clk;
    bool lane_pn_swap[LANE_MAX_NUM];
};

```

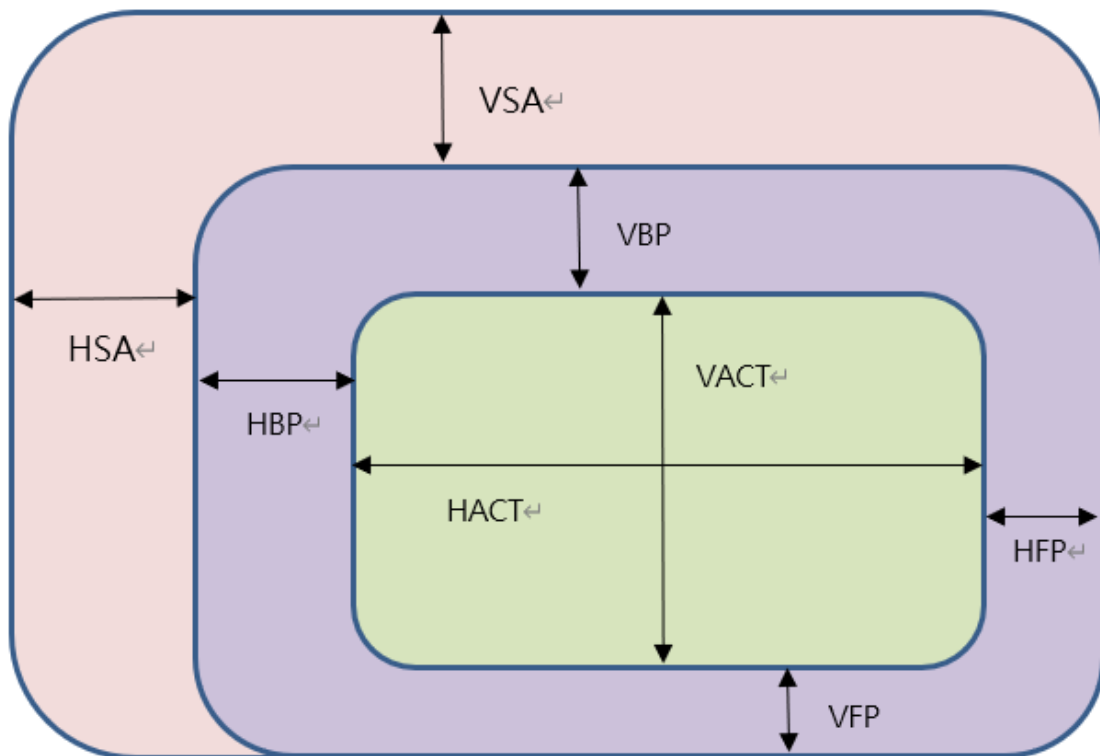
成员名称	描述
devno	MIPI Tx 设备号，默认 0
lane_id	<p>主控端和屏端 Lane 号的对应关系，未使用的 Lane 填 -1 即可。</p> <p>共 5 个成员，依序分别代表主控端的 MIPI_TX_0 ~ MIPI_TX_4，实际填写的内容需要根据对应到屏端的 MIPI lane 号。</p> <p>例如，第一个成员是主控 MIPI_TX_0，查电路原理图对应到屏端的 MIPI lane3，就填写为 MIPI_TX_LANE_3。</p> <p>对应关系不正确，将导致屏幕无法点亮。</p>
output_mode	MIPI Tx 输出模式，默认 OUTPUT_MODE_DSI_VIDEO
video_mode	MIPI Tx 视频模式，默认 BURST_MODE
output_format	MIPI Tx 输出格式，默认 OUTPUT_FORMAT_RGB_24_BIT
sync_info	MIPI Tx 设备的同步信息（详见下方）
pixel_clk	<p>像素时钟，单位为 KHz。</p> <p>使用 PIXEL_CLK(x) 宏自动计算：</p> <pre>#define PIXEL_CLK(x) \ ((x##_VACT + x##_VSA + x##_VBP \ ↪ + x##_VFP) \ * (x##_HACT + x##_HSA + x##_ \ ↪ HBP + x##_HFP) * x##_FPS / 1000)</pre> <p>lane_clk 根据 pixel_clk 反推，换算公式： $\text{lane_clk} = \text{pixel_clk} \times 24 / 4 / 2$ (24 表示 RGB888 每个 pixel 占 24 bits, 4 表示使用了 4 条 Data Lane, 2 表示 MIPI clk 是双边沿触发)</p>
lane_pn_swap	<p>MIPI Tx 的 Lane P/N 极是否交换</p> <p>true: 交换 false: 不交换</p>

sync_info_s (MIPI Tx 设备同步信息)

```
struct sync_info_s {
    unsigned short vid_hsa_pixels;
    unsigned short vid_hbp_pixels;
    unsigned short vid_hfp_pixels;
    unsigned short vid_hline_pixels;
    unsigned short vid_vsa_lines;
    unsigned short vid_vbp_lines;
    unsigned short vid_vfp_lines;
    unsigned short vid_active_lines;
    bool vid_vsa_pos_polarity;
    bool vid_hsa_pos_polarity;
};
```

成员名称	描述
vid_hsa_pixels	水平同步脉冲 (HSA), 单位为像素
vid_hbp_pixels	水平消隐后肩 (HBP), 单位为像素
vid_hfp_pixels	水平消隐前肩 (HFP), 单位为像素
vid_hline_pixels	水平有效区 (HACT), 单位为像素
vid_vsa_lines	垂直同步脉冲 (VSA), 单位为行
vid_vbp_lines	垂直消隐后肩 (VBP), 单位为行
vid_vfp_lines	垂直消隐前肩 (VFP), 单位为行
vid_active_lines	垂直有效区 (VACT), 单位为行
vid_vsa_pos_polarity	垂直有效信号的极性, false 为高有效, true 为低有效
vid_hsa_pos_polarity	水平有效信号的极性, false 为高有效, true 为低有效

MIPI DSI 协议下 MIPI 像素区域示意图

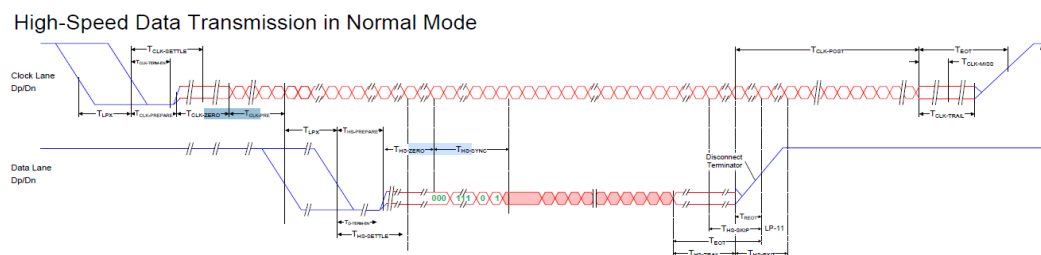


建议先将屏厂规格书中的 timing 参数定义为宏（使用头文件 stem 的大写形式作为前缀），再在 sync_info 中引用，避免裸常量分散在各处难以维护。

hs_settle_s (MIPI Tx HS timing)

```
struct hs_settle_s {
    unsigned char prepare;
    unsigned char zero;
    unsigned char trail;
};
```

MIPI Tx 时序图



屏幕初始化命令通过 `DSI_CMD()` 宏定义，不需要手动写 `data_type` 和 `size`。宏会自动按数据字节数选择 `data type`：

- 1 字节 → 0x05 (只有寄存器地址, 无数据)
- 2 字节 → 0x15 (寄存器地址 + 1 个数据)
- 3 字节及以上 → 0x29 (寄存器地址 + 2 个及以上数据)

```
/* 第一个参数始终是 delay (发送后延时, 单位 ms), 后面是寄存器地址 + 数据 */
#define DSI_CMD(delay_ms, ...) \
    { .delay = (delay_ms),      \
      .data_type = ...,         \
      .size = sizeof(...),     \
      .data = (CVI_U8[]){ VA_ARGS } }
```

使用示例：

```
struct dsc_instr dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_60fps[] = {
    DSI_CMD(0, 0xb9, 0xff, 0x83, 0x94), /* 多字节, 自动选 0x29 */
    DSI_CMD(0, 0x36, 0x02),             /* 2 字节, 自动选 0x15 */
    DSI_CMD(120, 0x11),                  /* Sleep Out, 1 字节 → 0x05 */
    DSI_CMD(20, 0x29),                   /* Display On */
};
```

如果屏厂指定了不同的 data type (如 0x23、0x39), 使用 DSI CMD TYPE 显式指定:

DSI CMD TYPE(0, 0x39, 0xff, 0x77, 0x01, 0x00),

初始化序列由屏厂商提供，具体指令含义请在屏厂规格书或 Driver IC Datasheet 中查找。

4.3.2 第 2 步：注册屏参头文件

根据目标系统选择对应的注册方式。

4.3.2.1 u-boot / cvi_alios 注册方式 (cvi_panels.h)

在 PanelSupportList/cvi_panels.h 中增加一个 #elif 块。cvi_panels.h 是 u-boot 和 cvi_alios 共用的编译期单选入口，通过 CONFIG_ 宏决定使用哪一块屏，**不能同时选择多块屏**。

```
#elif CONFIG_DSI_HX8394_720X1280_NULL_4LANE_60FPS
#include "panels/dsi_hx8394_720x1280_NULL_4lane_60fps.h"
static struct panel_desc s panel_desc = {
    .panel_name = "HX8394-720x1280-NULL-4lane-60fps",
    .dev_cfg = &dev_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps,
    .hs_timing_cfg = &hs_timing_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps,
    .dsi_init_cmds = dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_60fps,
    .dsi_init_cmds_size = ARRAY_SIZE(dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_
→60fps),
};
```

说明：

- CONFIG_ 宏由头文件 stem 大写派生 (dsi_hx8394_720x1280_NULL_4lane_60fps → CONFIG_DSI_HX8394_720X1280_NULL_4LANE_60FPS)。
- 不再使用旧式 MIPI_PANEL_HX8394 等别名，统一使用 CONFIG_ 宏。

4.3.2.2 cvi_mpi 注册方式 (sample_panel.h + sample_panel.c)

cvi_mpi 不使用 cvi_panels.h，而是在 sample_panel.h 中直接 include 所有屏参头文件，运行时通过命令行参数选择。

1. 在 sample_panel.h 中添加 #include：

```
#include "dsi_hx8394_720x1280_NULL_4lane_60fps.h"
```

2. 在 sample_panel.c 的 SAMPLE_SET_PANEL_DESC 函数中增加 case：

```
case DSI_PANEL_HX8394_EVB:
    g_panel_desc.panel_type = PANEL_MODE_DSI;
    g_panel_desc.stdsicfg.dev_cfg =
        (struct combo_dev_cfg_s *)&dev_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps;
    g_panel_desc.stdsicfg.hs_timing_cfg =
        &hs_timing_cfg_dsi_hx8394_720x1280_NULL_4lane_60fps;
    g_panel_desc.stdsicfg.dsi_init_cmds =
        dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_60fps;
    g_panel_desc.stdsicfg.dsi_init_cmds_size =
        ARRAY_SIZE(dsi_init_cmds_dsi_hx8394_720x1280_NULL_4lane_60fps);
    break;
```

cvi_mpi 方式支持编译进所有屏参，运行时用 ./sample_panel --panel=xxx 切换，适合调试阶段快速验证。

4.3.3 第 3 步：配置使能并构建

4.3.3.1 u-boot 配置

首先运行脚本生成 Kconfig（扫描 panels/ 目录下所有 .h 文件）：

```
cd PanelSupportList
python3 gen_panel_config.py
```

生成的 Kconfig.panels 供 u-boot 使用，包含类似以下内容：

```
config DSI_HX8394_720X1280_NULL_4LANE_60FPS
    bool "dsi_hx8394_720x1280_NULL_4lane_60fps"
    help
        "y" Config dsi_hx8394_720x1280_NULL_4lane_60fps.
```

然后通过以下任一方式使能：

- make u-boot-menuconfig 进入图形界面选择面板。
- 或在 build/boards/<board>/u-boot/<board>_defconfig 中添加 CONFIG_DSI_HX8394_720X1280_NULL_4LANE_60FPS=y。

4.3.3.2 cvi_alios 配置

在 cvi_alios/solutions/normboot/package_yamls/package.yaml.turnkey 中设置对应的宏为 1，其余置 0：

```
CONFIG_DSI_HX8394_720X1280_NULL_4LANE_60FPS: 1
CONFIG_PANEL_HX8394: 0
```

注：旧的 CONFIG_PANEL_XXX 别名保留但置 0，新屏统一用 CONFIG_DSI_XXX。

4.3.3.3 cvi_mpi 配置

cvi_mpi 无需编译期配置，所有屏参已在 sample_panel.h 中 include。编译后通过命令行选择：

```
./sample_panel --panel=HX8394_EVB
```

4.3.3.4 构建

完成上述配置后，执行 build_all 即可完成全系统构建。

4.4 RESET / POWER / BACKLIGHT 管脚配置

三种系统中仅 u-boot 支持通过设备树自动控制 GPIO；cvi_mpi 和 cvi_alios 需要在代码中手动操作 GPIO。

4.4.1 u-boot 配置方式 (DTS)

u-boot 通过设备树 (DTS) 配置 GPIO。在 build/boards/default/dts/cv184x/cv184x_base.dtsi 的 mipi_tx 节点中配置：

```
mipi_tx: mipi_tx {
    compatible = "cvitek,mipi_tx";
    reset-gpio = <&porte 2 GPIO_ACTIVE_LOW>;
    pwm-gpio = <&porte 0 GPIO_ACTIVE_HIGH>;
    power-ct-gpio = <&porte 1 GPIO_ACTIVE_HIGH>;
    clocks = <&clk CV181X_CLK_DISP_VIP>, <&clk CV181X_CLK_DSI_MAC_VIP>;
    clock-names = "clk_disp", "clk_dsi";
};
```

说明：

- reset-gpio：复位引脚，按屏规格书填写 GPIO 组、序号和有效电平。驱动根据 GPIO_ACTIVE_LOW / GPIO_ACTIVE_HIGH 自动产生 high-low-high（或 low-high-low）复位时序。
- pwm-gpio：背光控制。调试阶段可先用 GPIO 控制点亮背光；后续如需调节亮度，配置 pinmux 为 PWM 功能后将此行删除，改用 PWM 方式。
- power-ct-gpio：Panel 电源控制。如屏幕直接供电无需软件控制则删除。
- 管脚对应的 GPIO 组号和序号，对照《CV184X_PINOUT_CN》确认。
- 不需要的管脚直接不写对应 DTS 属性，驱动加载时自动跳过。

4.4.2 cvi_mpi 配置方式 (手动 GPIO)

cvi_mpi 不解析 DTS 中的 GPIO 属性，需在应用层通过 sysfs 或 sample_panel 命令行参数手动控制。

4.4.2.1 方式一：sample_panel 命令行参数 (仅调试用，代码中已注释)

运行时传入 GPIO 编号：

```
./sample_panel --panel=HX8394_EVB --gpio=354,0,353,1,352,1
```

三个 GPIO 依次为 RESET、POWER、PWM，每个 GPIO 两个参数（编号 + 电平）。

4.4.2.2 方式二：shell 手动操作 sysfs

先通过 `cat /sys/kernel/debug/gpio` 查看 GPIO 编号范围，确认各 GPIO 组对应的起始编号后，计算目标管脚的全局编号：

```
# 假设 RESET=GPIOE2(354), POWER=GPIOE1(353), PWM=GPIOE0(352)
echo 352 > /sys/class/gpio/export
echo 353 > /sys/class/gpio/export
echo 354 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio352/direction
echo out > /sys/class/gpio/gpio353/direction
echo out > /sys/class/gpio/gpio354/direction
echo 1 > /sys/class/gpio/gpio352/value # 背光点亮
echo 1 > /sys/class/gpio/gpio353/value # 电源使能
echo 1 > /sys/class/gpio/gpio354/value # RESET 拉高
sleep 0.02
echo 0 > /sys/class/gpio/gpio354/value # RESET 拉低
sleep 0.02
echo 1 > /sys/class/gpio/gpio354/value # RESET 拉高
```

调试阶段背光可先用 GPIO 控制；后续需调节亮度时，在 u-boot 中配置 pinmux 为 PWM 功能，则在命令中省略背光 GPIO。

4.4.3 cvi_alios 配置方式 (PLATFORM_PanelInit)

cvi_alios 不解析设备树，GPIO 控制在 PLATFORM_PanelInit 接口中手动完成。该函数位于板级定制目录下，例如：

cvi_alios/solutions/normboot/customization/<board>/src/custom_platform.c

在 PLATFORM_PanelInit 中通过 _GPIOSetValue 手动控制各管脚：

```
int PLATFORM_PanelInit(void)
{
    #if CONFIG_PANEL_HX8394
        u8 pw_port, pw_pin, bl_port, bl_pin, rst_port, rst_pin;
        pw_port = 4; pw_pin = 1; /* POWER → GPIOE1 */
        bl_port = 4; bl_pin = 0; /* BACKLIGHT → GPIOE0 */
        rst_port = 4; rst_pin = 2; /* RESET → GPIOE2 */
        _GPIOSetValue(pw_port, pw_pin, 1);
        _GPIOSetValue(bl_port, bl_pin, 1);
        _GPIOSetValue(rst_port, rst_pin, 1);
        udelay(20 * 1000);
        _GPIOSetValue(rst_port, rst_pin, 0);
        udelay(20 * 1000);
        _GPIOSetValue(rst_port, rst_pin, 1);
        udelay(20 * 1000);
    #endif
    return CVI_SUCCESS;
}
```

说明：

- port 为 GPIO 组号：0 = GPIOA, 1 = GPIOB, ..., 4 = GPIOE。

- pin 为组内序号，对照《CV184X_PINOUT_CN》确认。
- 复位时序需参考屏规格书，通常为 high-low-high（或 low-high-low）。
- 新增屏参时需在 PLATFORM_PanelInit 中添加对应的 #if CONFIG_XXX 分支。
- 如需 PWM 调节背光亮度，另外配置 pinmux 和 PWM 寄存器，此处仅做 GPIO 控制。

4.5 更换 Logo 图片

将客户的 logo 图片放置在对应板卡配置目录下：

build/boards/<board>/bootlogo/logo.jpg

例如：

build/boards/cv184x/cv1842cp_wevb_0015a_spinand/bootlogo/logo.jpg

如果板卡目录下没有 bootlogo/，新建该文件夹再放入 logo.jpg。编译时若板卡目录下不存在 logo.jpg，会自动从 build/tools/common/bootlogo/logo.jpg 拷贝。

注意：

- 不能直接将原始 logo.jpg 烧录，必须通过 copy_tools 脚本处理。
- I80 屏幕需要 24bit BMP 图片，其他类型屏幕需要 YUV420 格式 jpg。
- Logo 显示的具体细节请参考《CV184X 开机画面使用指南》。

4.6 编译烧写验证

在上述步骤均完成后，重新编译烧写。上电后可验证 MIPI 屏是否正常点亮。

如果未正常显示，按以下步骤排查：

1. 确认背光点亮。
2. 确认 RESET 管脚电平状态达到预期（如 high-low-high）。
3. 确认屏幕供电正常。
4. 执行 devmem 0x0a094094 32 0x0701000a 使能 VO test pattern，如果屏幕初始化成功，会看到 colorbar。

test pattern 寄存器如下图：

h94	REG_37			rstn	reg_gra_inv	0	0	1	h0	rw
h94				rstn	reg_pat_en	1	1	1	h0	rw
h94				rstn	reg_auto_en	2	2	1	h0	rw
h94				rstn	reg_dith_en	3	3	1	h1	rw
h94				rstn	reg_snow_en	4	4	1	h0	rw
h94				rstn	reg_fix_mc	5	5	1	h0	rw
h94				rstn	reg_dith_md	10	8	3	h0	rw
h94				rstn	reg_pat_prd	23	16	8	h1	rw
h94				rstn	reg_pat_idx	28	24	5	h0	rw

如果 test pattern 正常但无显示画面，问题通常在初始化序列或 timing 参数；如果 test pattern 也不正常，检查 MIPI Lane 顺序、RESET/POWER/PWM 配置，并使用万用表/示波器确认电路电平。

建议查看 Driver IC datasheet 或咨询屏幕厂商，开启屏的 BIST mode（通常是调整初始化序列中的某个寄存器值），可快速判断是主控端配置问题还是屏幕本身问题。

- BIST 不正常 → 检查 Lane 顺序 / RESET / POWER / PWM 及硬件电路，或咨询屏厂。
- BIST 正常 → 配置与硬件无异常，重点调整 sync_info_s 中的 timing 参数。

调试技巧：可先在 cvi_mpi 侧用 sample_panel 工具快速验证屏参，命令格式为 ./sample_panel --panel=<PANEL_NAME>。调通后再全系统构建验证，避免反复烧写 u-boot。