



CV181x 屏幕对接使用手册

Version: 1.2.2

Release date: 2022-06-13

©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
2.2.1.6.1	配置为 GPIO	12
2.2.1.6.2	配置为 PWM	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	14
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	17
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
3.2.1.3.1	配置为 GPIO	26
3.2.1.3.2	配置为 PWM	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	BT	30
4.1	环境准备	30
4.1.1	BT 接口介绍	30
4.1.2	硬件连线确认	31
4.2	配置步骤	31
4.2.1	配置管脚复用和驱动能力	33
4.2.2	配置复位	33
4.2.3	用 I2C 配置转换芯片	33
4.2.4	配置 BT 设备属性	33
4.3	问题 debug	35
4.3.1	管脚复用与驱动能力确认	35
4.3.2	确认硬件正常	36
4.3.3	波形检查	37
4.3.4	确认 display tgen 和 timing	39
4.3.5	转换 IC 相关 debug	39
4.3.6	抖动问题	40
4.3.7	图像格式问题	40
5	sRGB	41
5.1	环境准备	41
5.1.1	sRGB 接口介绍	41
5.1.2	CLK 设置	43
5.1.3	VO MUX	44
5.1.4	如何配置 VO MUX	45
5.2	配置步骤	47
5.2.1	背光和电源	48
5.2.2	spi 问题	49
5.3	问题 DEBUG	49
5.3.1	线序检查	49
5.3.2	示波器检查 CLK	49
5.3.3	波形电压	49
6	sRGB_bring_up_case	50

6.1	环境准备	50
6.1.1	JKC147H002 屏幕接口准备	50
6.2	配置步骤	54
6.2.1	CLK 设置	54
6.2.2	线序配置	55
6.2.3	驱动注册	56
6.2.4	模拟 spi	56
6.2.5	烧录测试点屏	57
6.2.6	注意事项	58
6.3	Checklist	58
7	I80	59
7.1	环境准备	59
7.1.1	I80 屏幕接口介绍	59
7.1.2	硬件连线确认	60
7.2	配置 I80 屏	60
7.2.1	在 u-boot 中配置 I80 屏	60
7.2.1.1	配置 I80 设备属性	61
7.2.1.2	添加头文件的引用	64
7.2.1.3	配置 RESET 管脚	65
7.2.1.4	配置 BACKLIGHT 管脚	65
7.2.1.5	配置 POWER 管脚	65
7.2.1.6	配置 u-boot 环境变量, 更换 logo 图片, 编译烧写验证	65
7.2.2	在 kernel 中配置 I80 屏	65
7.2.2.1	配置 I80 设备属性	65
7.2.2.2	添加头文件的引用	66
7.2.2.3	配置 I80 屏 RESET、POWER、BACKLIGHT 管脚	66
7.2.2.4	编译验证	67
7.3	问题 DEBUG	68
7.3.1	屏幕点不亮问题排查	68
7.3.2	常见问题	68

修订记录

Revision	Date	Description
0.1	2021/04/20	Initial version
1.1.1	2021/06/11	Modify some typo and description
1.2.0	2021/10/26	Revision update
1.2.1	2022/02/07	Revision update
1.2.1.0	2022/06/15	Update for CV181x
1.2.2	2022/06/23	Revision update
1.2.3	2026/04/29	Add BT/sRGB/I80 bring up guide

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。

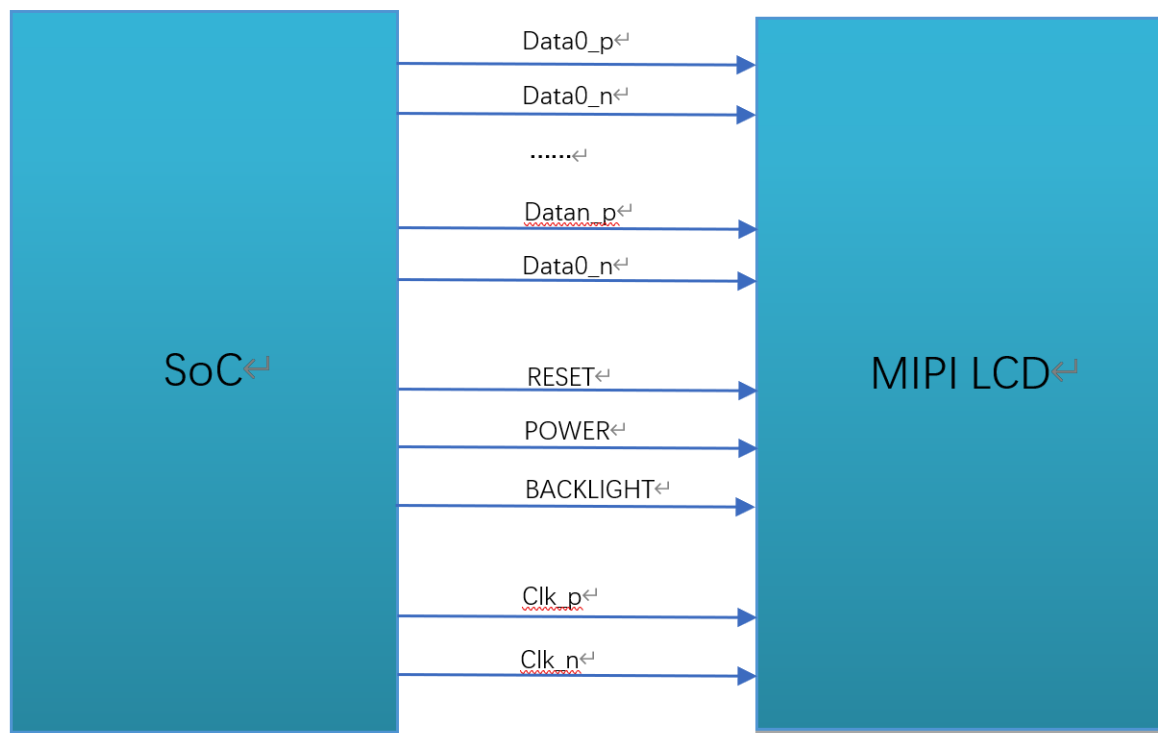
2.1 环境准备

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《CVITEK 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)

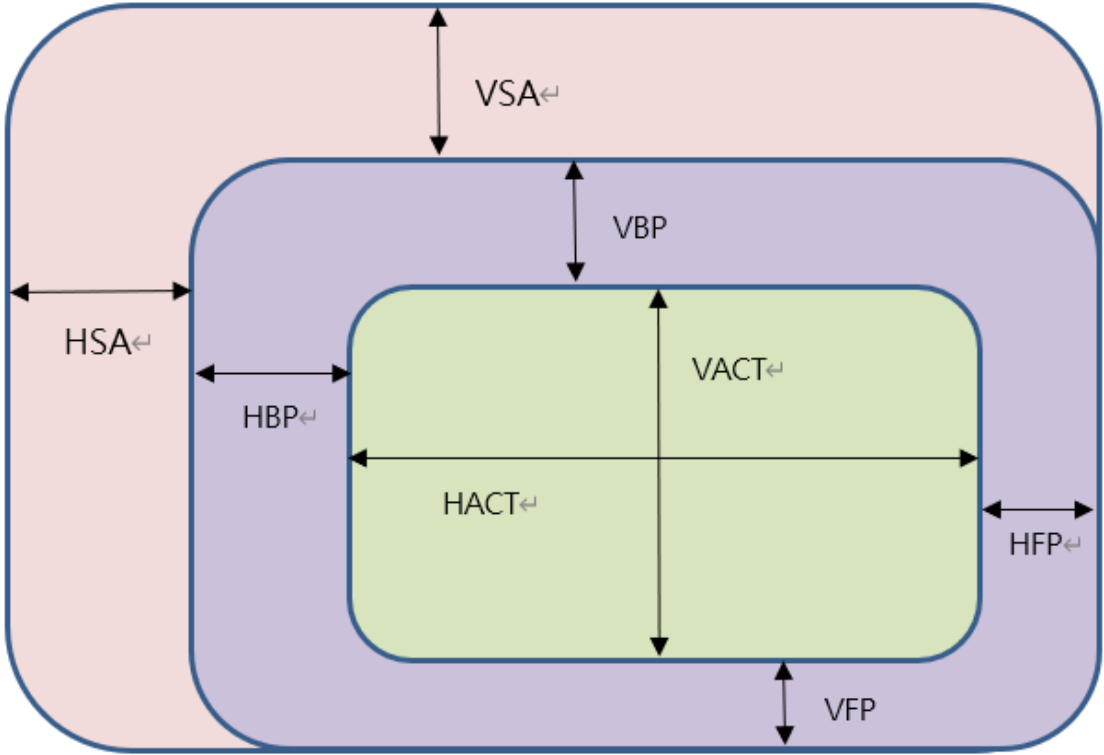
(续上页)

```
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;
```

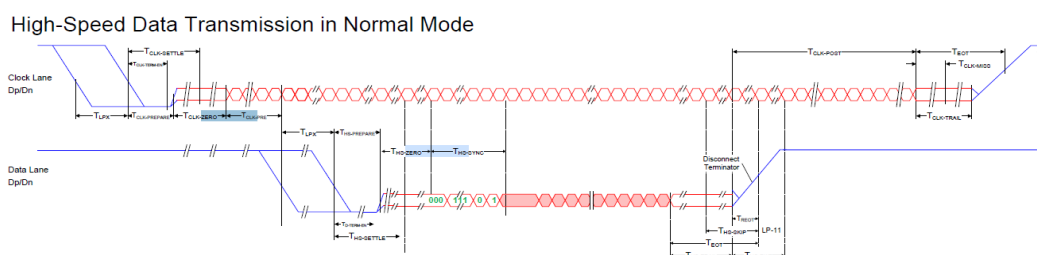
(下页继续)

(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《CV181X_PINOUT_CN》找到管脚对应的 GPIO 组号及序号。
- 修改 build/default/dts/cv181x/cv181x_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/default/dts/cv181x/cv181x_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 0，工作电压为高，在 build/default/dts/cv181x/cv181x_base.dtsi 修改如下：

```
power-ct-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, "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。

2.2.1.6.1 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 1，工作电压为高，在 build/default/dts/cv181x/cv181x_base.dtsi 修改如下：

```
pwm-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, "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);
```

2.2.1.6.2 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《CV181X 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《CV181X Preliminary Datasheet》，CV181X 有 4 组 PWM，每组 4 个通道

pwm0	0x03060000
pwm1:	0x03061000
pwm2:	0x03062000
pwm3:	0x03063000

注意：这里的 PWM0~3 是 PWM 组号，而原理图或者 pinlist 中写的是 PWM0~PWM15，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 PWM1

```
_reg_write(0x03060008, 0x3E8); // PWM1 低电平拍数 (单位ns)

_reg_write(0x0306000C, 0xF4240); // PWM1 方波周期拍数 (单位ns)

_reg_write(0x03060044, 0x02); // 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/cv181x-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_SW 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 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，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。

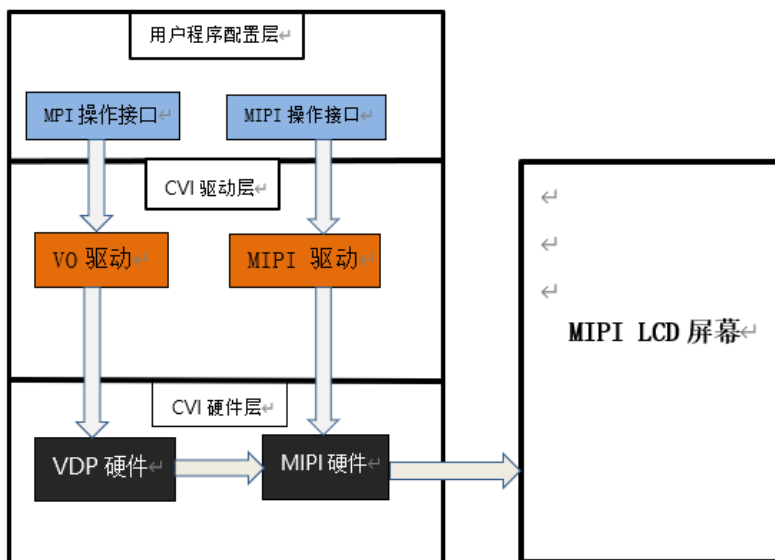
假如 BIST 正常，则说明以上配置正确，硬件电路没有异常，这时通常需要调整 sync_info_s 中的各参数。

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 `middleware/v2/component/panel/cv181x/` 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 `middleware/v2/sample/sample_panel/sample_panel.c` 中增加对上两节中新增头文件的引用。

示例：

如要点屏 HX8394_EVB，首先在 `middleware/v2/sample/sample_panel/sample_panel.h` 中 `include` 该屏幕的头文件 `dsi_hx8394_evb.h`，然后确保 `middleware/v2/sample/sample_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/cv181x/` 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 `dtb` 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=424,0,425,1,452,1
```

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：

直接在应用层直接控制 GPIO。

示例：假设 reset: GPIOB5, pwm: GPIOB3, power: GPIOB4, 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 GPIOB 的编号范围，若得出 gpiochip1 的范围是 448 至 479，则对于所要拉的 GPIOB5 来说，下面 echo 操作的号码可由 $448+5=453$ 得到。

因此需要以下操作去拉对应引脚：

```
1. echo 453 > /sys/class/gpio/export
   echo 451 > /sys/class/gpio/export
   echo 452 > /sys/class/gpio/export
2. echo out > /sys/class/gpio/gpio453/direction
   echo out > /sys/class/gpio/gpio451/direction
   echo out > /sys/class/gpio/gpio452/direction
3. echo 1 > /sys/class/gpio/gpio453/value
   echo 1 > /sys/class/gpio/gpio451/value
   echo 1 > /sys/class/gpio/gpio452/value
   echo 0 > /sys/class/gpio/gpio453/value
   echo 1 > /sys/class/gpio/gpio453/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 `middleware/v2/sample/sample_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 0x0a088094 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。

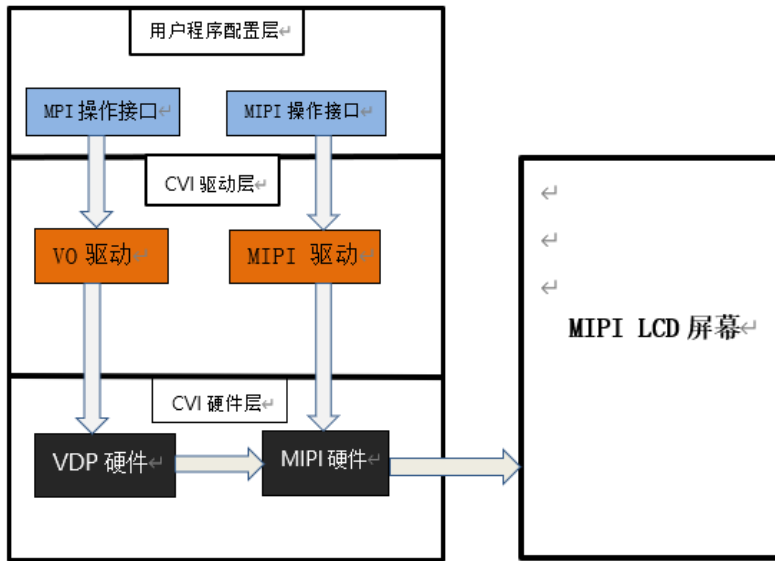
假如 `BIST mode` 不正常，则需要再检查 `MIPI Lane` 顺序、`RESET`、`POWER`、`PWM` 等是否配置正确，并使用万用表/示波器等确认电路电平状态符合预期，假如均符合预期，则可能是屏幕本身的问题，请咨询屏幕厂商。

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 `middleware/v2/component/panel/cv181x/` 和 `cvi_alios/components/cvi_mmf_sdk/cvi_middleware/include/panel/` 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 cv_i_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，若选择 `CONFIG_BOARD_CV181XH` 芯片，`CONFIG_PANEL_HX8394` 屏幕，则打开如下开关：

```
CONFIG_BOARD_CV181XC: 0
CONFIG_BOARD_CV181XH: 1

CONFIG_PANEL_HX8394: 1
```

注意 `CONFIG_BOARD_CV181XC` 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/cv1811c_cv2003_1l_triple/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/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
```

对于 `CONFIG_BOARD_CV181XC` 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。

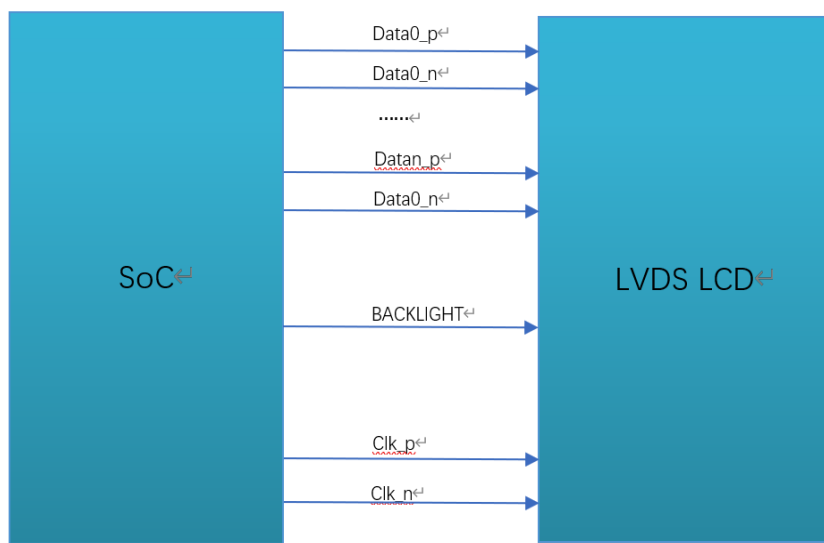
3.1 环境准备

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 8bit: 8 lane, 双路 10bit: 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：

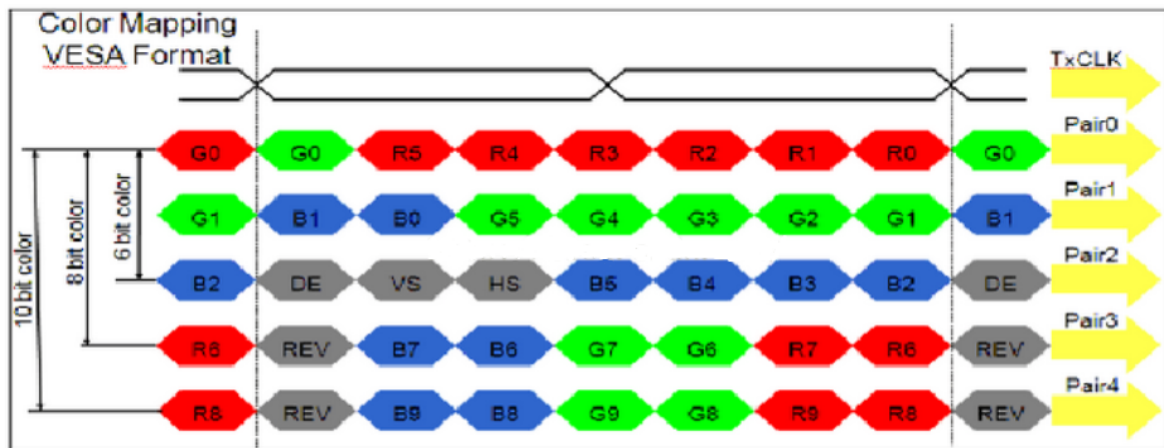

```

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。

3.2.1.3.1 配置为 GPIO

可通过修改 build/boards/cv181x/cv181xxx/u-boot/cvitek.h 中 VO_GPIO_PWM_PORT、VO_GPIO_PWM_INDEX、VO_GPIO_PWM_ACTIVE 实现。

3.2.1.3.2 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 `middleware/v2/component/panel/cv181x/` 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 `middleware/v2/sample/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 `middleware/v2/sample/sample_panel/sample_panel.c` 中 `include` 该屏幕的头文件 `lvds_lcm185x56.h`，然后确保 `middleware/v2/sample/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《CV180x/CV181x 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 `middleware/v2/component/panel/cv181x` 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 `middleware/v2/sample/sample_panel/` 下会生成 `sample_panel` 可执行文件。该程序和 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 `middleware/v2/component/panel/cv181x/` 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 BT

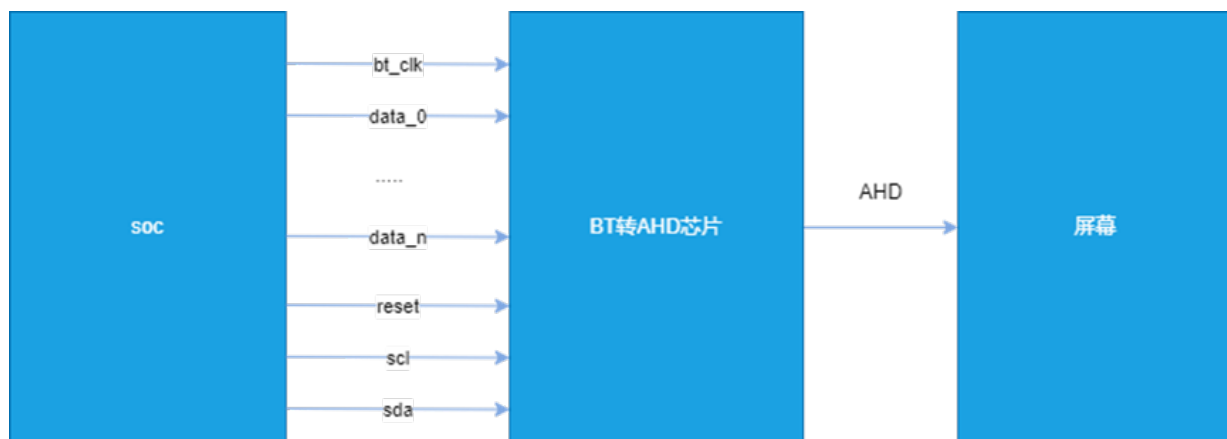
4.1 环境准备

4.1.1 BT 接口介绍

BT 接口现阶段最多用法是对接转换 IC，通过转换 IC，将 BT656/BT1120 信号转换为 AHD 信号或 CVBS 信号，再接入支持 AHD 或 CVBS 输入的显示设备，一般有以下几种信号，如图所示。

- BT 时钟线 (CLK)
- BT 数据线 (DATA) (BT656 8lane、BT1120 16lane)
- I2C SDA/SCL (用于配置转换芯片)
- 转换 IC 复位信号 (RESET)

BT 接口连线示意图



4.1.2 硬件连线确认

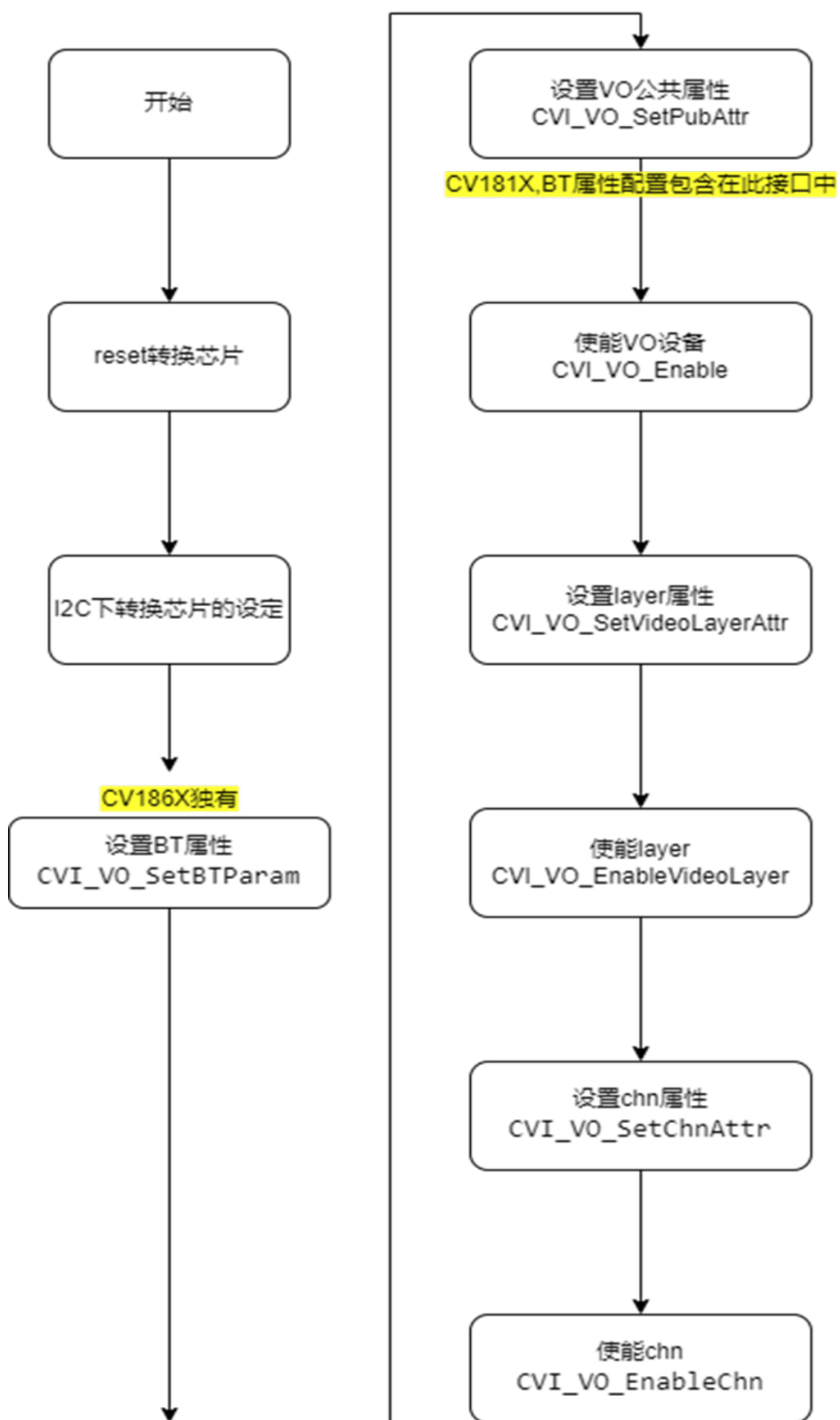
CV181X/CV184X: Data lane 一定要连接芯片端可以复用为 VO_D[X] 或者 VO_CLK0/1 信号的引脚否则无法输出数据。内部可以自由控制每个 VO_D 信号作为 BT 接口的具体哪一个引脚。VO_CLK0/1 也可作为数据输出。Clk lane 必须连接 PAD_VIVO_CLK 引脚。

CV186X: Dev0: Data lane 一定要连接芯片端可以复用为 VO0_D[X] 信号的引脚否则无法输出数据。内部可以自由控制每个 VO0_D 信号作为 BT 接口的具体哪一个引脚。Clk lane 必须连接可以复用为 VO0_CLK 的引脚。

Dev1: Data lane 一定要连接芯片端可以复用为 VO1_D[X] 信号的引脚否则无法输出数据。内部可以自由控制每个 VO1_D 信号作为 BT 接口的具体哪一个引脚。Clk lane 必须连接 PAD_VIVO_CLK 引脚。

4.2 配置步骤

BT 屏幕配置流程图



4.2.1 配置管脚复用和驱动能力

BT clk/data 驱动内会根据用户配置的 pin 映射表自动配置复用，并将对应信号设置为用户配置的功能。reset、I2C 需要用户自行配置复用。

4.2.2 配置复位

转换芯片一般不需要软件控制供电，一般上电就开始工作。reset 需要用户自行控制。方法参考前文。

4.2.3 用 I2C 配置转换芯片

如果前面的一切顺利，I2C 管脚复用配置正确，这时可以先用 i2cdetect 扫描对应总线上的 slave 设备。

查转换芯片的规格书，如果能找到对应地址的设备，说明 I2C 正常。这时可用 I2C 去下发原厂给的对应 resolution 下的初始化序列。

可参考 sample_panel 目录下的 panel_i2c.c，该目录下有 cvitek 对接 pt1000k，bt656/1120 转 AHD 转换芯片的示例代码。如果有 I2C 报错信息，请检查 I2C 管脚复用，I2C 总线 bus ID 和设备 I2C 地址是否符合规格书，和原理图。

4.2.4 配置 BT 设备属性

BT 暂时不支持 u-boot 配置，这里只讨论 kernel 中配置方式。

VO_BT_ATTR_S 结构体定义

CV186X:

```
typedef struct _VO_BT_ATTR_S {
    CVI_U8 pin_num;
    CVI_BOOL bt_clk_inv;
    CVI_BOOL bt_vs_inv;
    CVI_BOOL bt_hs_inv;
    VO_BT_DATA_SEQ_E data_seq;
    struct VO_D_REMAP d_pins[MAX_VO_PINS];
} VO_BT_ATTR_S;
```

CV181X:

```
struct VO_D_REMAP {
    enum VO_TOP_D_SEL sel;
    CVI_U32 mux;
};

struct VO_PINMUX {
    unsigned char pin_num;
    struct VO_D_REMAP d_pins[MAX_VO_PINS];
};
```

(下页继续)

(续上页)

```
};

typedef struct _VO_BT_ATTR_S {
    struct VO_PINMUX pins;
} VO_BT_ATTR_S;
```

成员名称	描述
pin_num	BT data + clk pin 总数
d_pins	PIN 脚映射关系
bt_clk_inv	bt clk 信号反向
bt_hs_inv	bt vs 信号反向, 仅 bt601 支持。
bt_hs_inv	bt hs 信号反向, 仅 bt601 支持。
data_seq	YUV 数据顺序, BT656: 00: Cb0Y0Cr0Y1 01: Cr0Y0Cb0Y1 10: Y0Cb0Y1Cr0 11: Y0Cr0Y1Cb0 BT601/1120: 0 : Cb0Cr0 1 : Cr0Cb0

示例:

```
#ifndef __BT656_PT1000K_H__
#define __BT656_PT1000K_H__

#include <cvi_comm_vo.h>

const VO_BT_ATTR_S s_pt1000kbt656cfg = {
    .pin_num = 9,
    .bt_clk_inv = 0,
    .bt_vs_inv = 0,
    .bt_hs_inv = 0,
    .data_seq = VO_BT_DATA_SEQ0,
    .d_pins = {
        {VO_MIPIO_TXP0, VO_MUX_BT_DATA0},
        {VO_MIPIO_TXN0, VO_MUX_BT_DATA1},
        {VO_MIPIO_TXP1, VO_MUX_BT_DATA2},
        {VO_MIPIO_TXN1, VO_MUX_BT_DATA3},
        {VO_MIPIO_TXP2, VO_MUX_BT_DATA4},
        {VO_MIPIO_TXN2, VO_MUX_BT_DATA5},
        {VO_MIPIO_TXP3, VO_MUX_BT_DATA6},
        {VO_MIPIO_TXN3, VO_MUX_BT_DATA7},
        {VO_VIVO_CLK, VO_MUX_BT_CLK},
    },
};
```

(下页继续)

(续上页)

```
#endif // __BT656_PT1000K_H__
```

Pin 脚映射关系根据原理图得到。关系不正确不能出图。

CV181X:

BT 属性嵌入 CVI_VO_SetPubAttr 中一起下发。

CV184X/CV186X:

BT 属性通过单独接口 CVI_VO_SetBTPParam 下发。

VO 会用 CVI_VO_SetPubAttr 中的同步信息和 framerate。pixclk 直接由内部拿到的同步信息和帧率计算得到。

示例：

```
case BT_PANEL_PT1000K_BT656_1280x720_25FPS_74M:
    g_panel_desc.panel_type = PANEL_MODE_BT;
    g_panel_desc.stbtcfg.stVoPubAttr.enIntfType = VO_INTF_BT656;
    g_panel_desc.stbtcfg.stVoPubAttr.enIntfSync = VO_OUTPUT_USER;
    VO_SYNC_INFO_S stPt1000kbt656_SyncInfo = {
        .bSym = 1, .bIop = 1, .u16FrameRate = 25,
        .u16Vact = 720, .u16Vbb = 20, .u16Vfb = 5,
        .u16Hact = 1280, .u16Hbb = 220, .u16Hfb = 440,
        .u16Vpw = 5, .u16Hp = 40, .bIdv = 0, .bIhs = 0, .bIvs = 0
    };
    g_panel_desc.stbtcfg.stVoPubAttr.stSyncInfo = stPt1000kbt656_SyncInfo;
    g_panel_desc.stbtcfg.BTAttr = s_pt1000kbt656cfg;
    break;
```

快速验证

同前文，在 sample_panel 中增加相关代码。需要额外增拉 reset 脚，和 I2C 下寄存器的操作。如果顺利转换芯片开始工作，将 SOC 输出的 BT 信号转换为 AHD 信号，再通过模拟高清线接入对应显示设备，输出 color bar。

4.3 问题 debug

4.3.1 管脚复用与驱动能力确认

驱动能力在调试初期可以先配置较高档位，待调亮屏幕后，再调优，以免损坏器件。可以参照《CV18XX_PINOUT_CN》和硬件设计图，通过 devmem + 寄存器地址，来读出管脚复用状态。如果状态不符合，请检查 BT 属性中的 pin 映射表是否按原理图配置，修改不正确的映射关系。以 CV181X 芯片的 PAD_MIPI_TXP0 引脚为例，在《CV181X_PINOUT_CN》中找到。

E1	PAD_MIPI_TXP0	1.8V GPIO	G12	VD018A_MIPI	IOBLK_G12_REG_PAD_MIPI_TXP0 0x0300_1C84	FHUX_GPIO_REG_IOCTL_PAD_MIPI_TXP0 0x0300_11B8	0x3	IO PAD_MIPI_TXP0 function select : 1 : VIO_D[10] 2 : VO_D[3] 3 : XGPIOC[13] (default) 4 : CAM_MCLK0 5 : P0H[15] 6 : CAM_HSD 7 : DBG[13] Others : Reserved
----	---------------	-----------	-----	-------------	--	--	-----	---

读寄存器 0x030011B8 的值确认复用关系。

PAD_MIPI1_TXP0	IOBLK_G12_REG_PAD_MIPI_TXP0	0x0300_1C84	IOBLK_G12_REG_PAD_MIPI_TXP0_PU	2	0x0	上拉电阻使能, 0=无效; 1=有效
			IOBLK_G12_REG_PAD_MIPI_TXP0_PD	3	0x1	下拉电阻使能, 0=无效; 1=有效
			IOBLK_G12_REG_PAD_MIPI_TXP0_DS0	5	0x0	输出驱动能力档位 bit 0
			IOBLK_G12_REG_PAD_MIPI_TXP0_DS1	6	0x1	输出驱动能力档位 bit 1
			IOBLK_G12_REG_PAD_MIPI_TXP0_ST0	8	0x0	输入施密特触发器强度控制, bit 0
			IOBLK_G12_REG_PAD_MIPI_TXP0_ST1	9	0x0	输入施密特触发器强度控制, bit 1
			IOBLK_G12_REG_PAD_MIPI_TXP0_HE	10	0x0	弱电平维持器(Bus holder) 使能, 0=无效; 1=有效
			IOBLK_G12_REG_PAD_MIPI_TXP0_SL	11	0x0	输出电平转换速率限制, 0=无效(较快); 1=有效(较慢)

读寄存器 0x03001C84 的值确认和修改驱动能力。也可使用 sdk 提供的 cvi_pinmux 工具, 使用方式参考 help 信息。

```
[root@cvitek]/mnt/nfs/A2_release/new# cvi_pinmux
cvi_pinmux for Athena2
./cvi_pinmux -p          <=> List all pins
./cvi_pinmux -l          <=> List all pins and its func
./cvi_pinmux -r pin      <=> Get func from pin
./cvi_pinmux -w pin/func <=> Set func to pin
./cvi_pinmux -c <pin name>,<0 or 1 or 2> <=> Set pin pull up/down (0:pull down; 1:pull up; 2:pull off)
./cvi_pinmux -d <pin name>,<0 ~ 15> <=> Set pin driving
./cvi_pinmux -D all <=> List all power domains' voltage
./cvi_pinmux -D <power domain>,<0 or 1> <=> Set the <power domain> voltage to 1(1.8V)/0(3.3V)
[root@cvitek]/mnt/nfs/A2_release/new# cvi_pinmux -r PAD_MIPI1_TX2P
pinctrl reg: 0x28104EC8
PAD_MIPI1_TX2P function:
[ ] PAD_MIPI1_TX2P
[ ] V00_D1
[ ] I2S1_MCLK
[v] GPIO189
[ ] UART2_TX
[ ] SPI2_SDI
[ ] IIC1_SDA
[ ] V01_D1
value: 3
```

逐一检查 BT 使用到的 pin 脚复用关系是否正确配置。

4.3.2 确认硬件正常

reset 因为为 GPIO, 可通过拉高拉低 GPIO 排查硬件异常:

```
echo 399 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio399/direction
echo 1 > /sys/class/gpio/gpio399/value
echo 0 > /sys/class/gpio/gpio399/value
```

I2C 参考上文对 I2C 的描述。

BT clk 引脚只能通过量测波形来检查,

- BT656 下: 量测频率 = bt clk = 2 * pix_clk
- BT1120 下: 量测频率 = bt clk = pix_clk

$\text{pix_clk} = \text{htt} * \text{vtt} * \text{fps}$

对应到 vo pubattr 中的 sync info 为 $\text{pix_clk} = (\text{u16Hact} + \text{u16Hbb} + \text{u16Hfb} + \text{u16Hpw}) * (\text{u16Vact} + \text{u16Vbb} + \text{u16Vfb} + \text{u16Vpw}) * \text{u16FrameRate}$

上例中 pt1000k 为 $\text{pix_clk} = (1280 + 220 + 440 + 40) * (720 + 20 + 5 + 5) * 25 = 37.125\text{M}$

详细的 timing 信息和 clk 需要, 不同的转换芯片不同的 mode 有不同的要求, 一般通过转换芯片的规格书和原厂获取。主要注意的是有的转换芯片, 在没有独立晶振提供 mclk(转换芯片工作 clk) 时, 需要内部分频电路通过, bt clk 分频得到 mclk, 因此如果 bt clk 管脚如果没有信号或者频率不符合要求, 芯片可能无法工作。

V: 标记消隐信息, 传输消隐数据时为 1, 传输有效视频数据时为 0

H: 标记 EAV 还是 SAV, SAV 为 0, EAV 为 1

而 P0~P3 为保护比特, 其值取决于 F、H、V, 起到校验的作用目前只支持渐进式逐行扫描格 (Progressive), 故 bit 6 值为 0。SAV_VLD: 有效行区, 行同步信号结束, 有效画素开始。EVA_VLD: 有效行区, 行同步信号开始, 有效画素结束。SAV_BLK: 消隐行区, 行同步信号结束。EAV_BLK: 消隐行区, 行同步信号开始。

SAV/EAV bit				Protection Bit				注解
7 (Fixed)	6 (F)	5 (V)	4 (H)	3 (P3)	2 (P2)	1 (P1)	0 (P0)	
1	0	0	0	0	0	0	0	SAV_VLD
1	0	0	1	1	1	0	1	EAV_VLD
1	0	1	0	1	0	1	1	SAV_BLK
1	0	1	1	0	1	1	0	EAV_BLK

BT.656: BT.656 和 BT.1120 差别只在影像传输用 16 bit (BT.1120) 和 8bit(BT.656), 其余垂直时序和同步码格式都一致。因为才用 8bit, 所以 $\text{clk} * 2$ 。



正常情况下, 配置正确了 BT 的 pin 映射表, 并且按流程开启了 VO 的业务之后, BT 的波形就会开始输出。如果量测到某些 pin 脚无波形输出, 请按前两部排查硬件。

因为 BT656/1120 的 sav, eav 是嵌入在 data lane 内部的, 当存在不出图的情况, 可以检查转换 IC 相关的的输入状态寄存器 (通过 spec 查找), 判断转换 IC input 的状态, 如果存在收到行数 and 实际设置不匹配的问题, 或者 sav eav err 这类错误, 可以量测波形查看 clk 是否能采样到 data, 如果不能, 可以设置 clk 反向尝试。

4.3.4 确认 display tgen 和 timing

```
[root@cvitek]/mnt/nfs/A2_release/new# cat /proc/soph/vo_disp
-----DISP(0)-----
disp_from_sc(0)      sync_ext(0)      tgen_en(0)      fmt(RGB888_PLANAR)
in_csc(      Disable) out_csc(      Disable)      burst(0)      y_thresh(0)      c_thresh(0)
start_x(  0)      start_y(  0)      width(  80)      height(  80)
pitch_y(  0)      pitch_c(  0)
err_fwr_yuv(000)      err_erd_yuv(000)      lb_full_yuv(000)      lb_empty_yuv(111)
bw fail(0)
-----DISP-TIMING(0)-----
total( 479 * 359)      hsync_pol(  0) vsync_pol(  0)
hsync_start( 479)      hsync_end( 359) vsync_start(  1)      vsync_end(  3)
hde-start(  20)      hde-end( 339) vde-start( 110)      vde-end( 349)
-----DISP(1)-----
disp_from_sc(0)      sync_ext(0)      tgen_en(1)      fmt(RGB888_PLANAR)
in_csc(      Disable) out_csc(      Disable)      burst(0)      y_thresh(0)      c_thresh(0)
start_x(  0)      start_y(  0)      width(  80)      height(  80)
pitch_y(  0)      pitch_c(  0)
err_fwr_yuv(000)      err_erd_yuv(000)      lb_full_yuv(000)      lb_empty_yuv(111)
bw fail(1)
-----DISP-TIMING(1)-----
total( 947 * 1305)      hsync_pol(  1) vsync_pol(  0)
hsync_start( 947)      hsync_end(1305) vsync_start(  0)      vsync_end( 15)
hde-start( 100)      hde-end( 819) vde-start(  20)      vde-end(1299)
```

如果 tgen 未开说明 VO 硬件未使能，是不会出图的。对于 BT 在 CVI_VO_ENABLE 接口中打开，如未打开请检查业务流程。在 sample_panel 中显示 pattern 时 CVI_VO_ShowPattern 会自动打开。如果遇到图像位置不对等问题确认 timing 信息，确认实际生效和屏幕厂家提供是否符合，如不符合则需做出调整。

4.3.5 转换 IC 相关 debug

当确认前几步都准确无误之后，SOC 正常输出信号并且波形质量都很好，仍然不能出图，此时就需要排查转换 IC 了。咨询原厂，开启转换 IC 的 test pattern，即不依赖外部 bt 输入信号，产生测试图例，查看是否能正常显示，一般情况下，此时主控 SOC 端，只控制 reset，I2C 下寄存器，提供 mclk。部分 IC，如 TP2803，需要 mclk 和 bt clk 同时提供，才能出 test pattern。如果连转换 IC 自带的 test pattern 都不出图，那么需要检查，soc 供电，I2C 寄存器是否能写入。独立晶振提供 mclk 的话需要检查晶振是否起振，频率是否准确。mclk 由 bt clk 分频得到的话需要 bt clk 是否准确，转换芯片相关 pll 分频是否配置正确。

4.3.6 抖动问题

当 mclk 和 bt clk 不同源的时候，往往会遇到图像抖动问题。需要咨询原厂当不同源时，是否有相关配置寄存器可以优化这类问题，如果转换芯片不支持，那么只能换用同源时钟了。

4.3.7 图像格式问题

这类问题一般都是转换 IC 端收的 YUV 顺序和主控 SOC 端出的 YUV 顺序不一致导致的，主控和转换 IC 端都有相应寄存器可以调整顺序，SOC 端可以直接通过设置 data_seq，转换 IC 端需要查看规格书或咨询原厂。

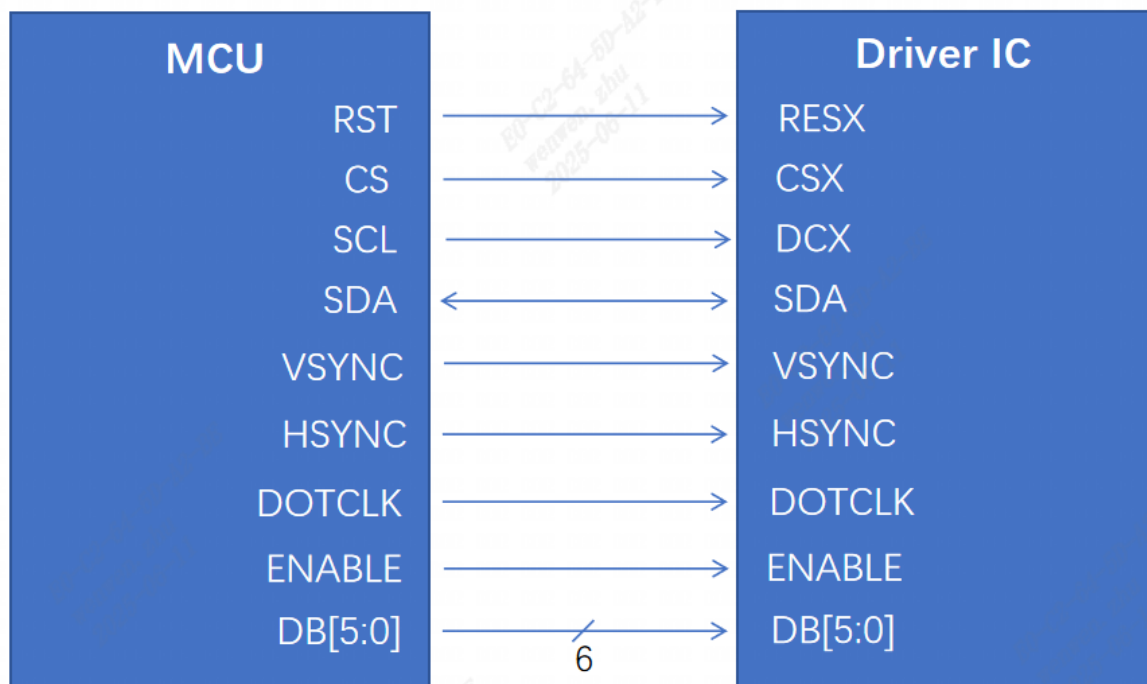
5 sRGB

5.1 环境准备

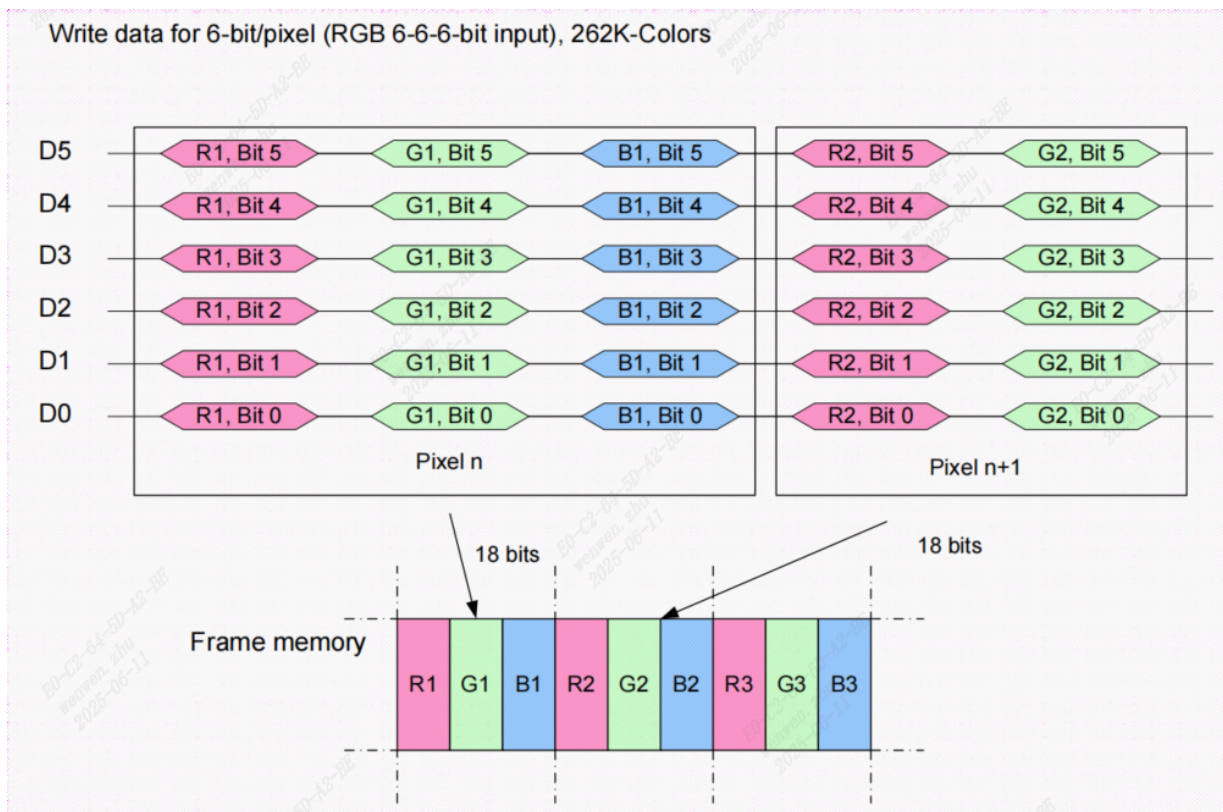
5.1.1 sRGB 接口介绍

sRGB 分为 3-cycle RGB 和 4-cycle SRGB, 3-cycle 是为了支持 Sitronix spec, 4-cycle RGB 是为了支持旭曜 spec。接口和时序相关信息如下:

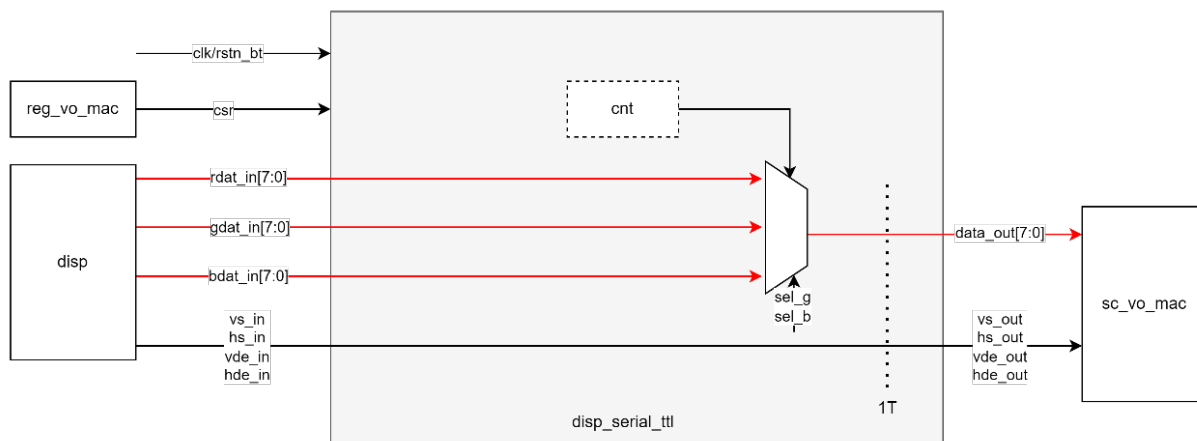
6-bit RGB interface



下图以 RGB666 为例, 介绍 pixel 排布情况:

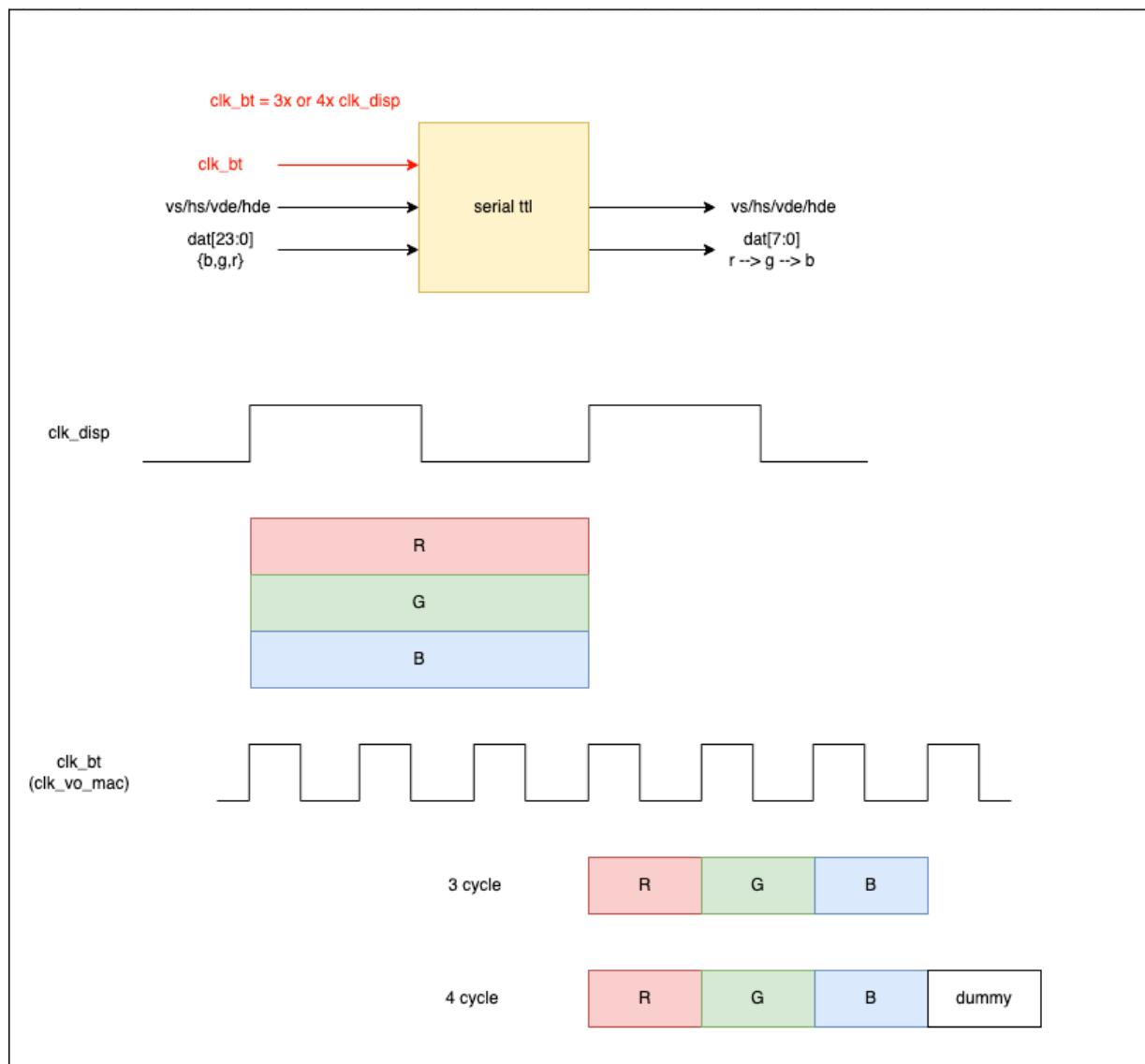


sRGB 由 disp_serial_ttl 模块实现，disp_serial_ttl 模块用来输出 serial RGB 格式，支持 RGB565/666/888 三种色深，RGB 输出顺序可以重排，其架构图如下：



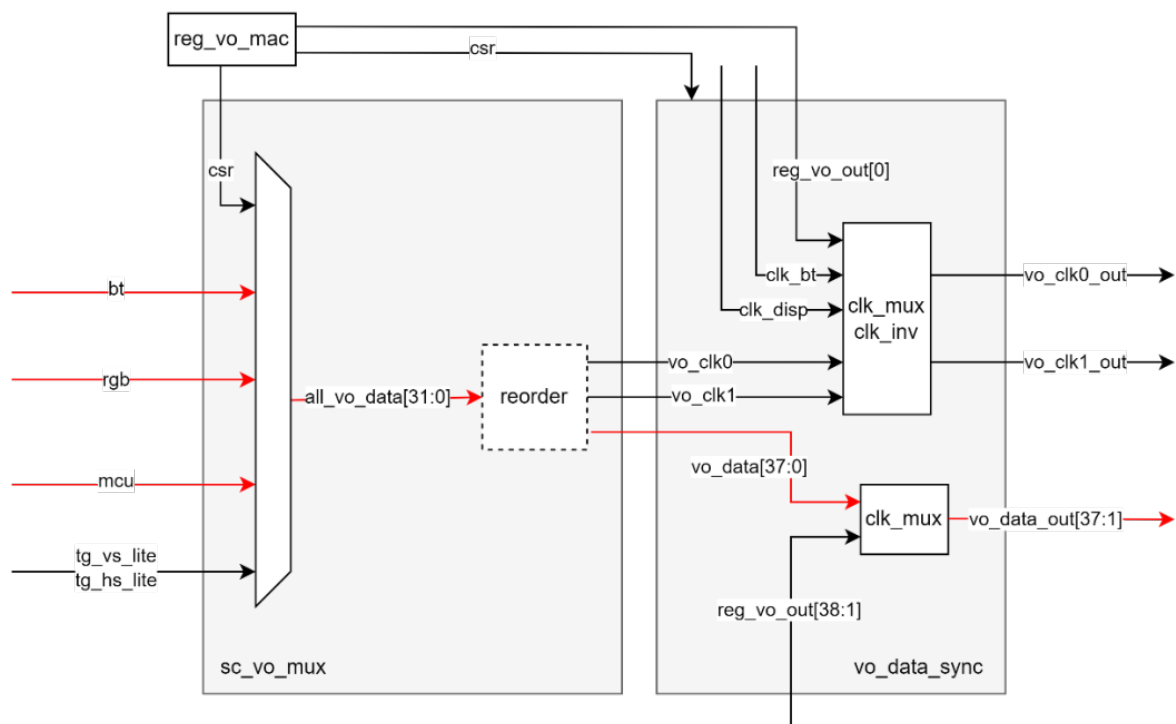
5.1.2 CLK 设置

可以设定 3T 或者 4T 打出一个 pixel，同时需要对应设定 $\text{clk_bt}=3*\text{clk_disp}$ 或者 $\text{clk_bt}=4*\text{clk_disp}$ 。设定 3T 或 4T 时，对应的时序如下：



5.1.3 VO MUX

sRGB 以及其他的 vo 协议 (BT, RGB, HW MCU, SW MCU) 模块的输出数据会进入到 `sc_vo_mux`, 在该模块中做 mux, 选择其中一种输出格式, 并可通过寄存器设定每个 bit 在输入和输出之间的映射关系, 进入到 `vo_data_sync` 模块后, 可以再次将该输出格式与 APB 设定的寄存器二则一输出。



5.1.4 如何配置 VO MUX

reg vo_sel[3:0]	4'h1 sel_rgb	4'h2 sel_sw	4'h3 sel_mcu	4'h4 sel_bt601	4'h5 sel_bt656	4'h6 sel_bt17(bt1120)	4'h7 sel_bt17_R	4'h8 sel_sttl	4'h9 sel_hw_mcu
vo_sig[31]	tg_vs_lite		tg_vs_lite	tg_vs_lite	tg_vs_lite	tg_vs_lite	tg_vs_lite	tg_vs_lite	tg_vs_lite
vo_sig[30]	tg_hs_lite		tg_hs_lite	tg_hs_lite	tg_hs_lite	tg_hs_lite	tg_hs_lite	tg_hs_lite	tg_hs_lite
vo_sig[29]	1'b0		1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0
vo_sig[28]	1'b0		1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0
vo_sig[27]	1'b0		1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0
vo_sig[26]	hde		1'b0	1'b0	1'b0	1'b0	1'b0	hde	1'b0
vo_sig[25]	hs		1'b0	1'b0	1'b0	1'b0	1'b0	hs	1'b0
vo_sig[24]	vs		1'b0	1'b0	1'b0	1'b0	1'b0	vs	1'b0
vo_sig[23]	R[7]		1'b0	1'b0	1'b0	1'b0	1'b0	R/G/B[7]	1'b0
vo_sig[22]	R[6]		1'b0	1'b0	1'b0	1'b0	1'b0	R/G/B[6]	1'b0
vo_sig[21]	R[5]		1'b0	1'b0	1'b0	1'b0	1'b0	R/G/B[5]	1'b0
vo_sig[20]	R[4]		1'b0	1'b0	1'b0	1'b0	1'b0	R/G/B[4]	1'b0
vo_sig[19]	R[3]		1'b0	1'b0	1'b0	1'b0	1'b0	R/G/B[3]	1'b0
vo_sig[18]	R[2]		1'b0	1'b0	1'b0	BT_D[15]	BT_D[15]	R/G/B[2]	1'b0
vo_sig[17]	R[1]		1'b0	1'b0	1'b0	BT_D[14]	BT_D[14]	R/G/B[1]	1'b0
vo_sig[16]	R[0]		1'b0	1'b0	1'b0	BT_D[13]	BT_D[13]	R/G/B[0]	1'b0
vo_sig[15]	G[7]		1'b0	1'b0	1'b0	BT_D[12]	BT_D[12]	1'b0	1'b0
vo_sig[14]	G[6]		1'b0	1'b0	1'b0	BT_D[11]	BT_D[11]	1'b0	1'b0
vo_sig[13]	G[5]		1'b0	1'b0	1'b0	BT_D[10]	BT_D[10]	1'b0	1'b0
vo_sig[12]	G[4]		1'b0	1'b0	1'b0	BT_D[9]	BT_D[9]	1'b0	1'b0
vo_sig[11]	G[3]		MCU_D[7]	1'b0	1'b0	BT_D[8]	BT_D[8]	1'b0	MCU_D[7]
vo_sig[10]	G[2]		MCU_D[6]	BT_D[7]	BT_D[7]	BT_D[7]	BT_D[7]	1'b0	MCU_D[6]
vo_sig[9]	G[1]		MCU_D[5]	BT_D[6]	BT_D[6]	BT_D[6]	BT_D[6]	1'b0	MCU_D[5]
vo_sig[8]	G[0]		MCU_D[4]	BT_D[5]	BT_D[5]	BT_D[5]	BT_D[5]	1'b0	MCU_D[4]
vo_sig[7]	B[7]		MCU_D[3]	BT_D[4]	BT_D[4]	BT_D[4]	BT_D[4]	1'b0	MCU_D[3]
vo_sig[6]	B[6]		MCU_D[2]	BT_D[3]	BT_D[3]	BT_D[3]	BT_D[3]	1'b0	MCU_D[2]
vo_sig[5]	B[5]		MCU_D[1]	BT_D[2]	BT_D[2]	BT_D[2]	BT_D[2]	1'b0	MCU_D[1]
vo_sig[4]	B[4]		MCU_D[0]	BT_D[1]	BT_D[1]	BT_D[1]	BT_D[1]	1'b0	MCU_D[0]
vo_sig[3]	B[3]		MCU_CTRL[3]	BT_D[0]	BT_D[0]	BT_D[0]	BT_D[0]	1'b0	MCU_RD[X]
vo_sig[2]	B[2]		MCU_CTRL[2]	HDE	HDE	HDE	HDE	1'b0	MCU_WR[X]
vo_sig[1]	B[1]		MCU_CTRL[1]	HS	HS	HS	HS	1'b0	MCU_CD[X]
vo_sig[0]	B[0]		MCU_CTRL[0]	VS	VS	VS	VS	1'b0	MCU_CS[X]

		rstn	reg_vo_voclkl_sel	12	8	5	h0	rw
		rstn	reg_vo_vod0_sel	20	16	5	h0	rw
		rstn	reg_vo_vod1_sel	28	24	5	h0	rw
h94	REG_VO_MUX_1	rstn	reg_vo_vod2_sel	4	0	5	h0	rw
		rstn	reg_vo_vod3_sel	12	8	5	h0	rw
		rstn	reg_vo_vod4_sel	20	16	5	h0	rw
		rstn	reg_vo_vod5_sel	28	24	5	h0	rw
h98	REG_VO_MUX_2	rstn	reg_vo_vod6_sel	4	0	5	h0	rw
		rstn	reg_vo_vod7_sel	12	8	5	h0	rw
		rstn	reg_vo_vod8_sel	20	16	5	h0	rw
		rstn	reg_vo_vod9_sel	28	24	5	h0	rw
h9C	REG_VO_MUX_3	rstn	reg_vo_vod10_sel	4	0	5	h0	rw
		rstn	reg_vo_vod11_sel	12	8	5	h0	rw
		rstn	reg_vo_vod12_sel	20	16	5	h0	rw
		rstn	reg_vo_vod13_sel	28	24	5	h0	rw
hA0	REG_VO_MUX_4	rstn	reg_vo_vod14_sel	4	0	5	h0	rw
		rstn	reg_vo_vod15_sel	12	8	5	h0	rw
		rstn	reg_vo_vod16_sel	20	16	5	h0	rw
		rstn	reg_vo_vod17_sel	28	24	5	h0	rw
hA4	REG_VO_MUX_5	rstn	reg_vo_vod18_sel	4	0	5	h0	rw
		rstn	reg_vo_vod19_sel	12	8	5	h0	rw
		rstn	reg_vo_vod20_sel	20	16	5	h0	rw
		rstn	reg_vo_vod21_sel	28	24	5	h0	rw
hA8	REG_VO_MUX_6	rstn	reg_vo_vod22_sel	4	0	5	h0	rw
		rstn	reg_vo_vod23_sel	12	8	5	h0	rw
		rstn	reg_vo_vod24_sel	20	16	5	h0	rw
		rstn	reg_vo_vod25_sel	28	24	5	h0	rw
hAC	REG_VO_MUX_7	rstn	reg_vo_vod26_sel	4	0	5	h0	rw
		rstn	reg_vo_vod27_sel	12	8	5	h0	rw
		rstn	reg_vo_voclkl0_inv	16	16	1	h0	rw
		rstn	reg_vo_voclkl1_inv	17	17	1	h0	rw
		rstn	reg_vo_voclkl0_as_dat	20	20	1	h0	rw
		rstn	reg_vo_voclkl1_as_dat	21	21	1	h0	rw
hB0	REG_VO_MUX_8	rstn	reg_vo_vod28_sel	4	0	5	h0	rw
		rstn	reg_vo_vod29_sel	12	8	5	h0	rw
		rstn	reg_vo_vod30_sel	20	16	5	h0	rw
		rstn	reg_vo_vod31_sel	28	24	5	h0	rw
hB4	REG_VO_MUX_9	rstn	reg_vo_vod32_sel	4	0	5	h0	rw
		rstn	reg_vo_vod33_sel	12	8	5	h0	rw
		rstn	reg_vo_vod34_sel	20	16	5	h0	rw
		rstn	reg_vo_vod35_sel	28	24	5	h0	rw
hB8	REG_VO_MUX_A	rstn	reg_vo_vod36_sel	4	0	5	h0	rw
		rstn	reg_vo_vod37_sel	12	8	5	h0	rw

上面两张图分别对应 VO signal out mapping 和 VO mux 的寄存器设置。以 3-cycle RGB 为例来展示如何配置 mux。由第一张图可以得到：

```
VO_16 -> data0
VO_17 -> data1
VO_18 -> data2
VO_19 -> data3
VO_20 -> data4
VO_21 -> data5
VO_22 -> data6
VO_23 -> data7
VO_24 -> vsync
VO_25 -> hsync
VO_26 -> hde
```

设置 vo mux 就是将对应的寄存器配置为：

```
_reg_write(REG_VO_MAC_VO_MUX1(inst), 0x00001A00);
_reg_write(REG_VO_MAC_VO_MUX3(inst), 0x16000000);
_reg_write(REG_VO_MAC_VO_MUX4(inst), 0x12131415);
_reg_write(REG_VO_MAC_VO_MUX5(inst), 0x19171011);
```

(下页继续)

(续上页)

```
_reg_write(REG_VO_MAC_VO_MUX6(inst), 0x18000000);
_reg_write(REG_VO_MAC_VO_MUX7(inst), 0x00030000);
```

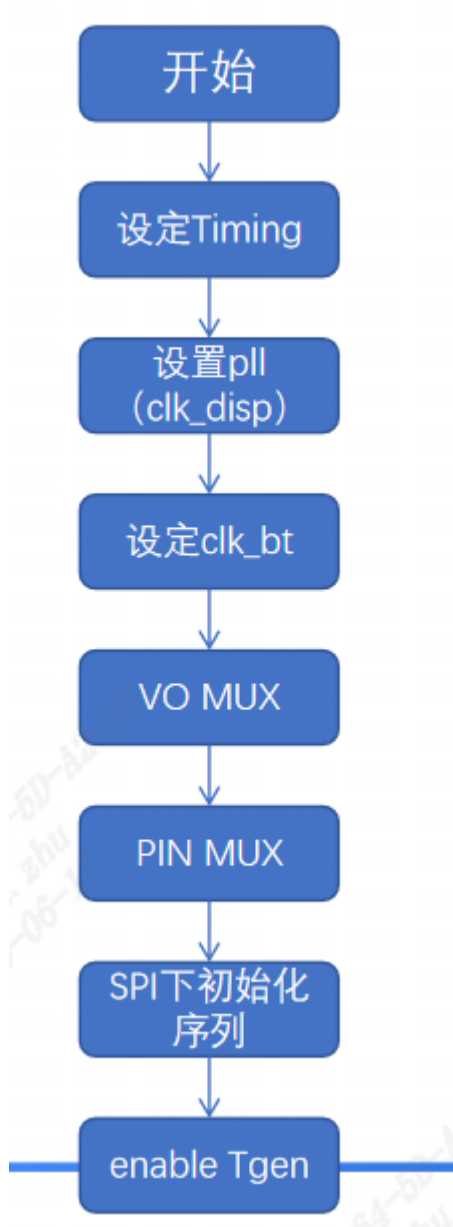
h144	h144	REG_IOCTL_VIVO_D7	rstn	funcsel_VIVO_D7	2	0	3	h3	rw	Others - Reserved IO VIVO_D6 function select : 0 : V12_D[6] 1 : V11_D[6] 2 : VO_D[18] 3 : XGPIOB[15] (default) 4 : RMI0_REFCLK 5 : SPI3_SCK 6 : UART2_TX 7 : CAM_VS0
h148	h148	REG_IOCTL_VIVO_D6	rstn	funcsel_VIVO_D6	2	0	3	h3	rw	Others - Reserved IO VIVO_D5 function select : 0 : V12_D[5] 1 : V11_D[5] 2 : VO_D[18] 3 : XGPIOB[16] (default) 4 : RMI0_RXD0 5 : SPI3_CS_X 6 : UART2_RX 7 : CAM_HS0
h14c	h14c	REG_IOCTL_VIVO_D5	rstn	funcsel_VIVO_D5	2	0	3	h3	rw	Others - Reserved IO VIVO_D4 function select : 0 : V12_D[4] 1 : V11_D[4] 2 : VO_D[17] 3 : XGPIOB[17] (default) 4 : RMI0_MDC 5 : IIC1_SDA 6 : UART2_CTS 7 : CAM_VS0
h150	h150	REG_IOCTL_VIVO_D4	rstn	funcsel_VIVO_D4	2	0	3	h3	rw	Others - Reserved IO VIVO_D3 function select : 0 : V12_D[3] 1 : V11_D[3] 2 : VO_D[16] 3 : XGPIOB[18] (default) 4 : RMI0_TXD0 5 : IIC1_SCL 6 : UART2_RTS 7 : CAM_HS0
h154	h154	REG_IOCTL_VIVO_D3	rstn	funcsel_VIVO_D3	2	0	3	h3	rw	Others - Reserved IO VIVO_D2 function select : 0 : V12_D[2] 1 : V11_D[2] 2 : VO_D[15] 3 : XGPIOB[19] (default) 4 : RMI0_TXD1 5 : CAM_MCLK1 6 : PWM[2] 7 : UART2_TX
h158	h158	REG_IOCTL_VIVO_D2	rstn	funcsel_VIVO_D2	2	0	3	h3	rw	Others - Reserved IO VIVO_D1 function select :

然后根据 pinout 表来 pin mux，类似于下面处理：

```
_reg_write(0x03001148, 0x02); // VO_D[19]
_reg_write(0x0300114C, 0x02); // VO_D[18]
_reg_write(0x03001150, 0x02); // VO_D[17]
_reg_write(0x03001154, 0x02); // VO_D[16]
_reg_write(0x03001158, 0x02); // VO_D[15]
_reg_write(0x0300115C, 0x02); // VO_D[14]
_reg_write(0x03001160, 0x02); // VO_D[13]
_reg_write(0x03001144, 0x02); // VO_D[20]
```

5.2 配置步骤

寄存器配置流程如下：



5.2.1 背光和电源

sRGB 的电源和背光都是外部控制的，不需要配置通过 GPIO 去控制。需要注意在配置 driver IC 的时候，需要先去拉 reset，需要先拉高再拉低。

5.2.2 spi 问题

- 在 spi 设置 pin mux 的时候, 需要注意 mars3 需要切换两次, 第一次切成 MUX_SPI_XX, 第二次再切成 SPI_XX。
- spi 在下初始化序列的时候, 需要通过 GPIO 拉低拉高来区分发送命令还是数据。发送命令需要拉低, 发送数据需要拉高。

5.3 问题 DEBUG

5.3.1 线序检查

sRGB 涉及到了两次 mux, 一次是 vo mux, 一次是 pin mux, 所以点不亮屏的时候, 检查线序是有必要的。检查线序的时候可以通过修改 vo mux 寄存器结合示波器观察波形来确认。比如说修改 vo mux 寄存器将 data0 设置成 0, 结合硬件同事给的线序, 用示波器去量转接板 data0 那根线是否还有波形, 如果有波形说明线序有问题。

5.3.2 示波器检查 CLK

Clk 分为了 clk_bt 和 clk_disp, 其中 clk_bt 容易测量, 直接通过示波器测量 clk lane 就可以了, 但是 clk_disp 无法直接测量出来, 但是可以通过测量 data lane 来反推 clk_disp。具体做法是: 把 data lane 波形展开, 根据 data 波形的规律, 可以测量出帧率是多少, 结合 htt 和 vtt 就可以推出 clk_disp 是多少。这样就可以看出来 clk_bt 和 clk_disp 是否符合预期。

5.3.3 波形电压

正常从主控板端测量到的波形电压一般在 1.8V 左右, 到了转接板后经过 level shift 芯片后电压一般会提升到 3.3V 左右。

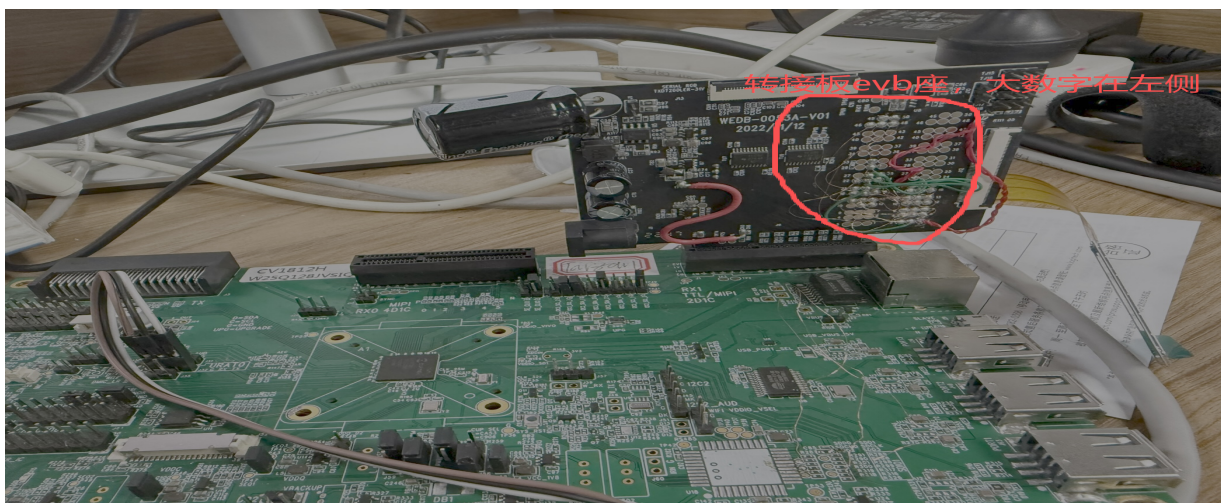
6 sRGB_bring_up_case

6.1 环境准备

6.1.1 JKC147H002 屏幕接口准备

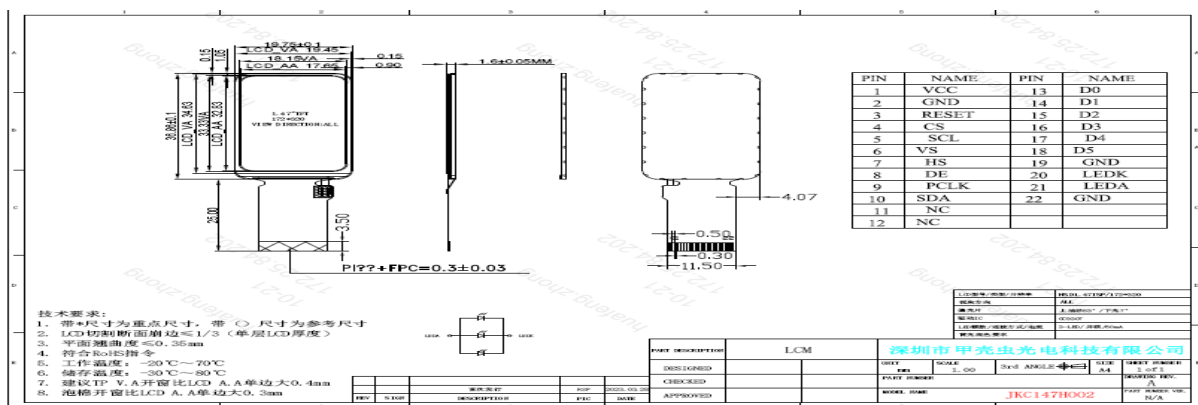
本屏幕为 3-cycle RGB，开发板型号为 cv1812h_0007a_spinor，sdk 版本为 cv181x (v4.1.0)，以下为屏幕、转接板 EVB 座，EVB 接口脚的对应信息：

	屏幕	转接板 EVB 座	EVB 接口脚
1	VCC		
2	GND		
3	RESET	3->TP46	IIC0_SCL(默认上拉)
4	CS	4->TP49	SD1_D3
5	SCL	5->TP47	SD1_CLK
6	VS	6->TP16	VIVO_D2
7	HS	7->TP15	VIVO_D1
8	DE	8->TP13	VIVO_D0
9	PCLK	9->TP12	VIVO_CLK
10	SDA	10->TP48	SD1_CMD(DATAOUT) &SD1_D0(DATAIN)
11	NC		
12	NC		
13	D0	13->TP18	VIVO_D3
14	D1	14->TP19	VIVO_D4
15	D2	15->TP21	VIVO_D5
16	D3	16->TP22	VIVO_D6
17	D4	17->TP24	VIVO_D7
18	D5	18->TP25	VIVO_D8
19	GND		
20	LEDK	GND	
21	LEDA	TP71	
22	GND	GND	

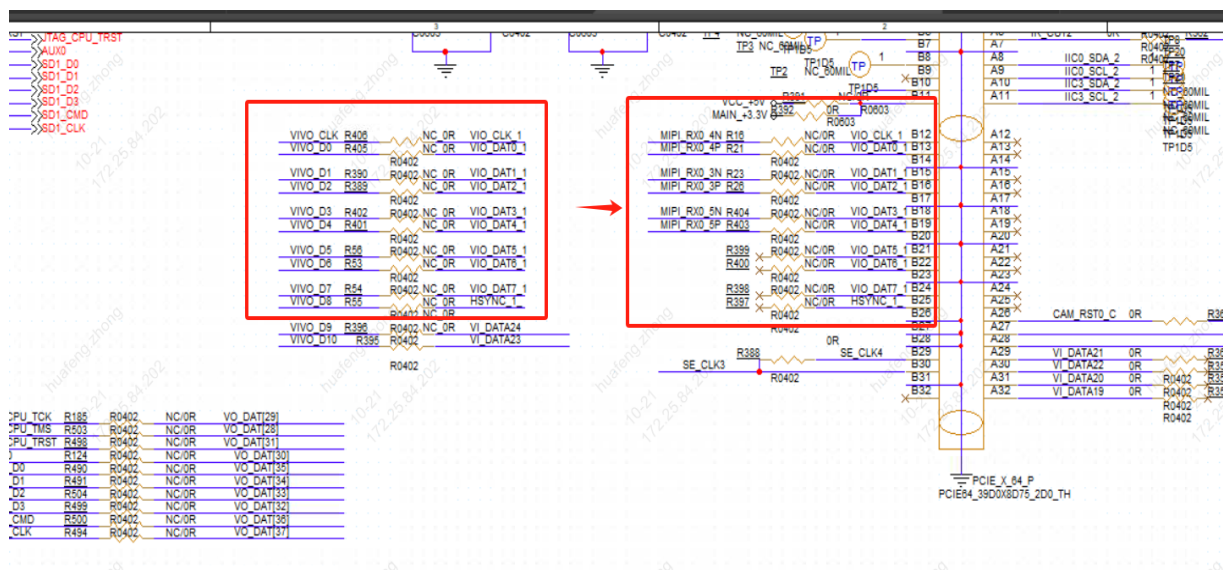


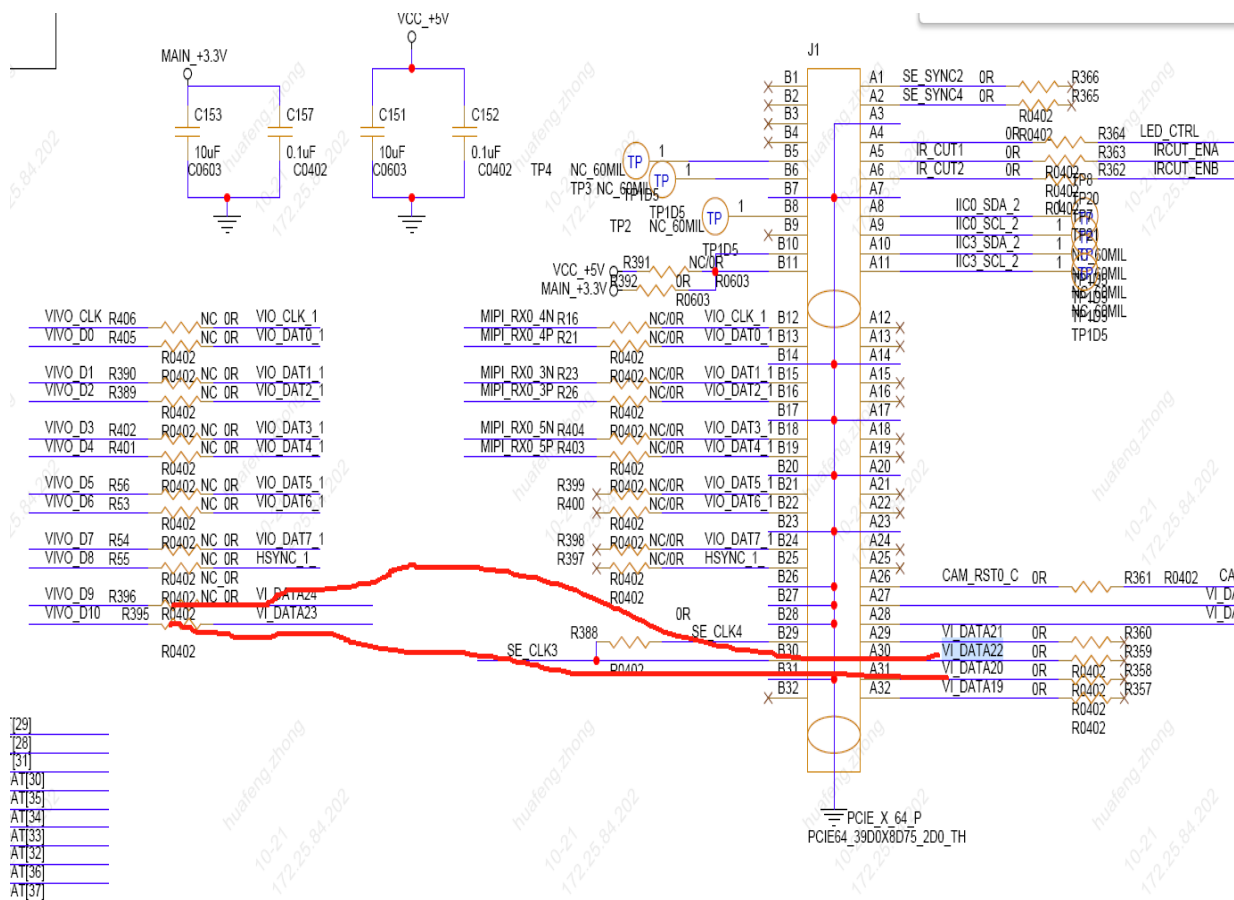
如图可以在转接板 EVB 座上找到对应的引脚连接。

下面屏幕的规格信息：

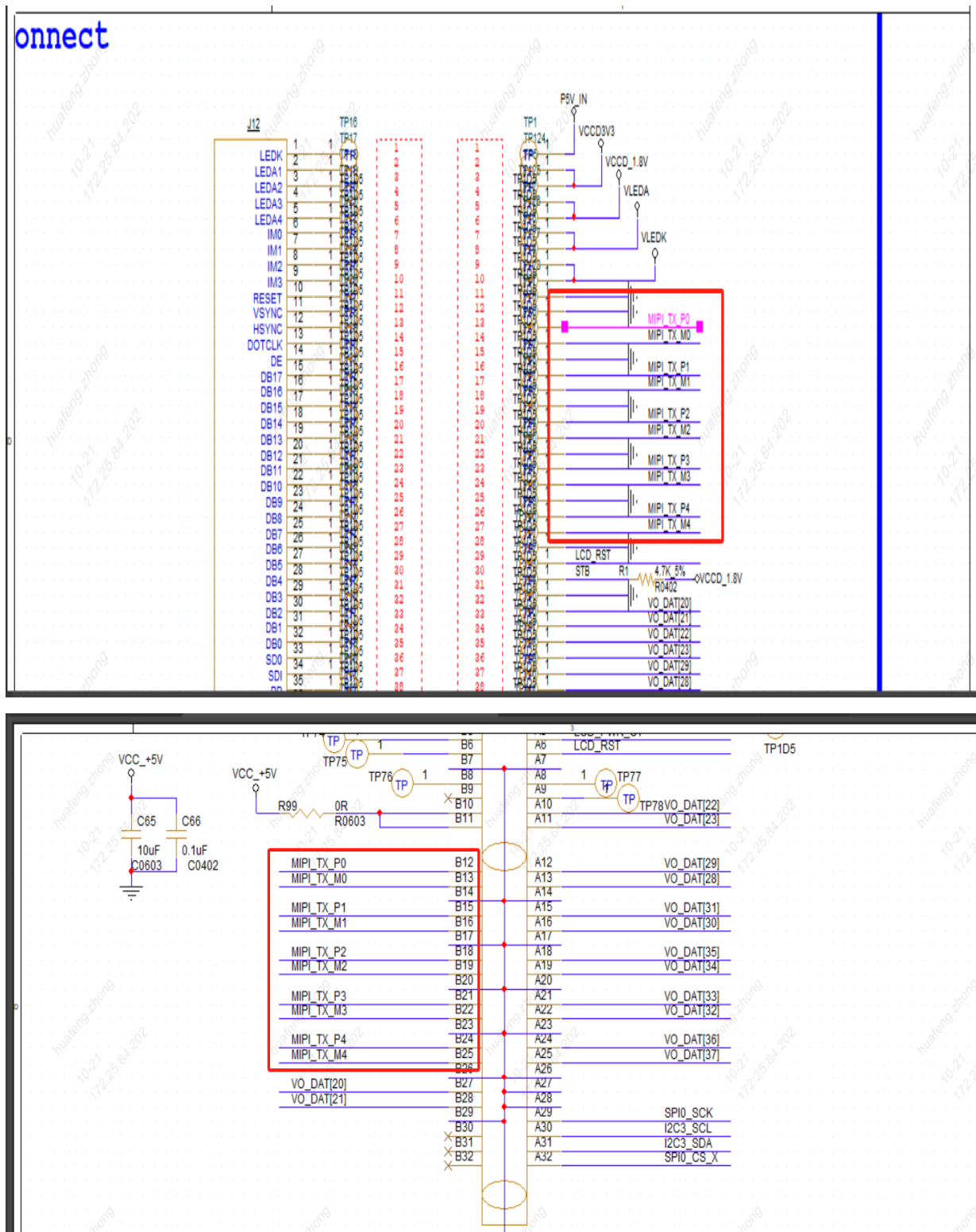


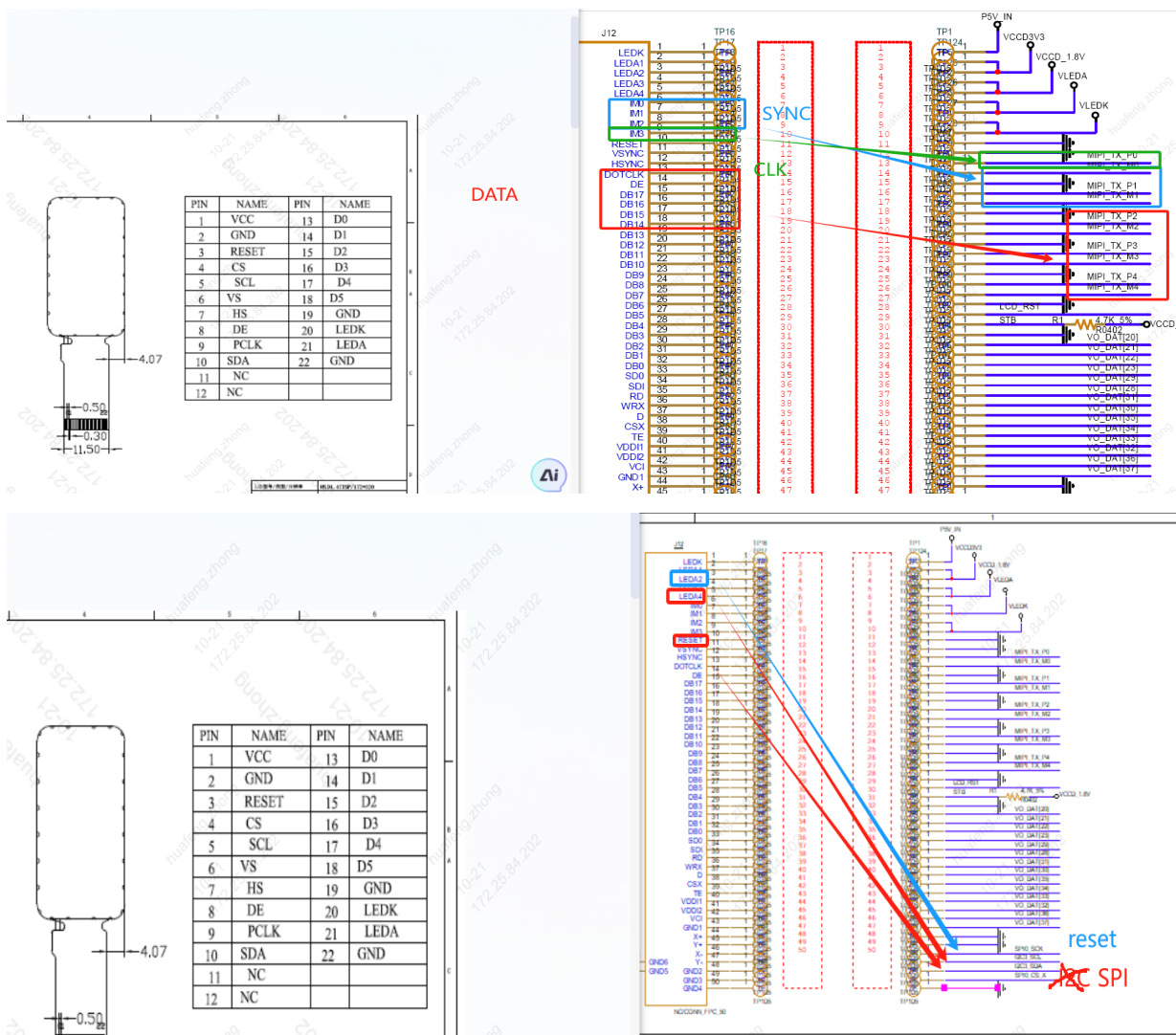
由于这个屏幕没有额外控制线, 所以对于区分命令与数据需要在发送有效数据 8bit 之前发送 1bit 的数据用于区分本次事务发送的是数据还是命令。具体实现就是在 SDA 上以 9bit 格式发送每组数据。以下是 EVB 板接口电路原理图:





转接板的电路原理图：

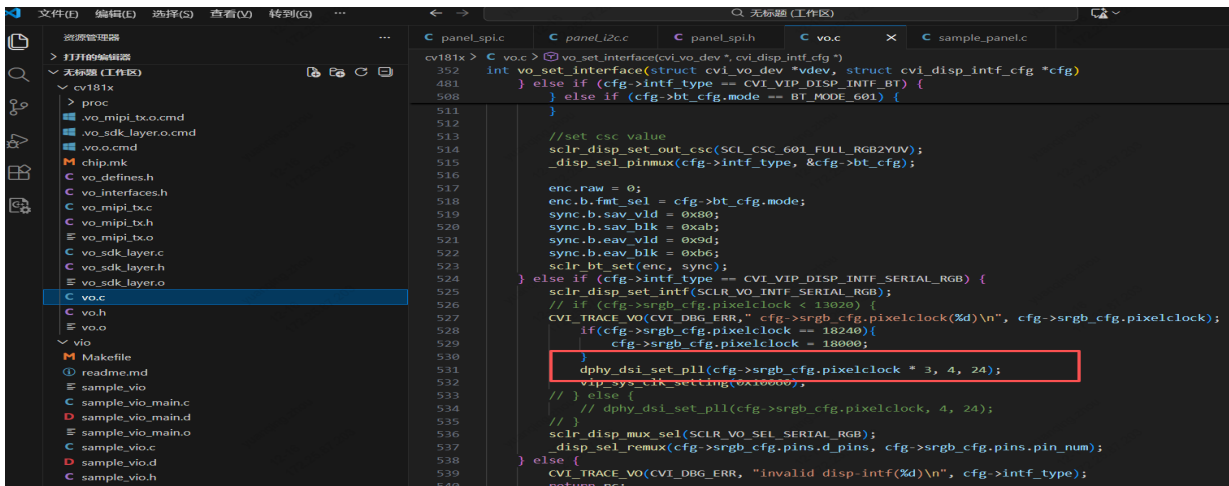




6.2 配置步骤

6.2.1 CLK 设置

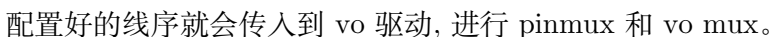
设定 3T 打出一个 pixel，同时需要对应设定 $\text{clk_bt} = 3 * \text{clk_disp}$.



6.2.2 线序配置

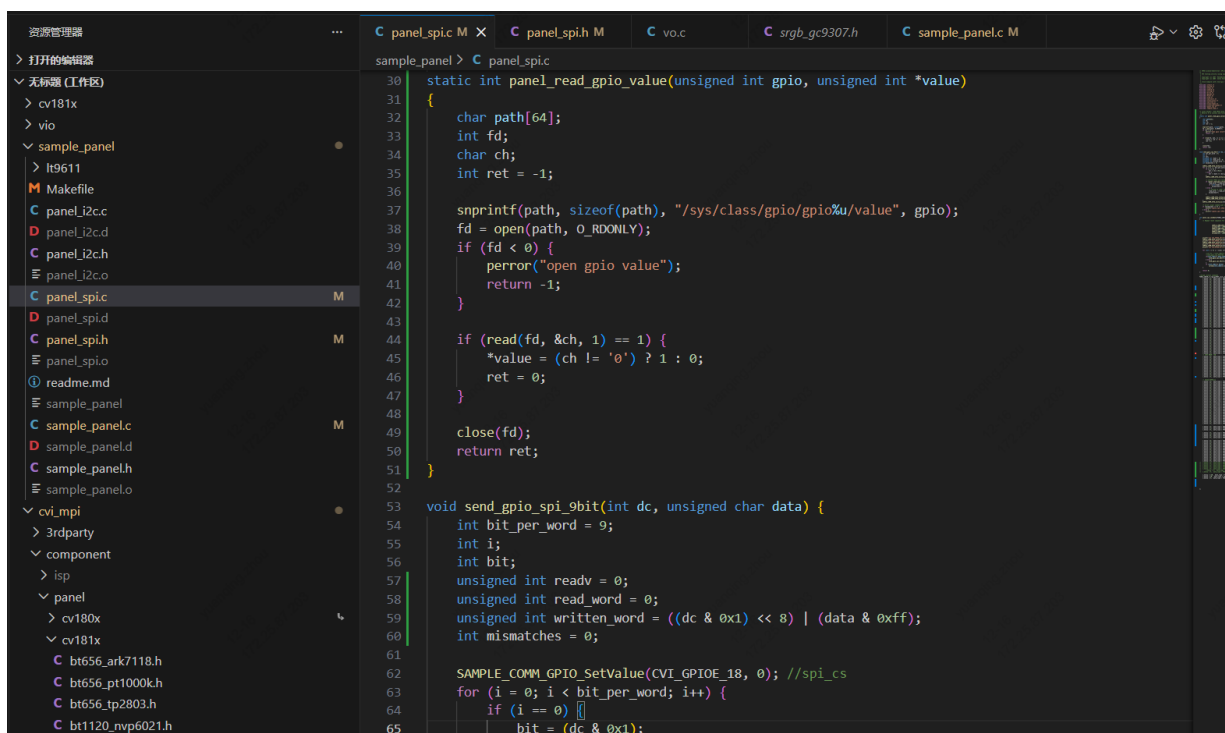
	管脚	VO (LC38D)		RAW8/12/16/65/61601 (ZD1C+RAW)		RAW		BT656		8080 8bit		BT601		RGB			
		功能	电平(V)	功能	VO	电平(V)	功能	VO	电平(V)	功能	VO	电平(V)	功能	VO	电平(V)	功能	VO
GPIO	VIVO_D10	180D33	VO_D[23] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D9	180D33	VO_D[22] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D8	180D33	VO_D[21] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D7	180D33	VO_D[20] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D6	180D33	VO_D[19] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D5	180D33	VO_D[18] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D4	180D33	VO_D[17] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D3	180D33	VO_D[16] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	VIVO_D2	180D33	VO_D[15] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_VS	meet spec	VORG8_VS	meet spec	
	VIVO_D1	180D33	VO_D[14] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_HS	meet spec	VORG8_HS	meet spec	
	VIVO_D0	180D33	VO_D[13] (O)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_DE	meet spec	VORG8_DE	meet spec	
	VIVO_CLK	180D33	VO_CLK1 (O) (CLK) (DATA)	180D33	GPIO	meet spec	VO_CLK1	meet spec	VO656_CLK	meet spec	8080_Data	meet spec	VO601_CLK	meet spec	VORG8_CLK	meet spec	
GPIO	MIPIRXSN	1.8V	VO_D[12] (O)	1.8V	GPIO	meet spec	VO_Data	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRXSP	1.8V	VO_D[11] (O)	1.8V	GPIO	meet spec	VO_Data	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRX4N	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRX4P	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRX3N	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRX3P	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO
	MIPIRX2N	1.8V	VO_D[10] (O)	1.8V	GPIO	meet spec	VO_Data	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	VO601_Data	meet spec	VORG8_Data	meet spec
	MIPIRX2P	1.8V	VO_D[9] (O)	1.8V	GPIO	meet spec	VO_Data	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	VO601_Data	meet spec	VORG8_Data	meet spec
	MIPIRX1N	1.8V	VO_D[8] (O)	1.8V	GPIO	meet spec	VO_Data	1.8V	GPIO	1.8V	GPIO	1.8V	GPIO	VO601_Data	meet spec	VORG8_Data	meet spec
	MIPIRX1P	1.8V	VO_D[7] (O)	1.8V	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPIRX0N	1.8V	VO_D[6] (O)	1.8V	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPIRX0P	1.8V	VO_D[5] (O)	1.8V	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
MIPI data 跟 CLK 可以自行定	MIPI_TXM4	1.8V	VO_D[24] (O)	meet spec	MIPI_TX_DM4	meet spec	VO_Data	1.8V	GPIO	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPI_TXP4	1.8V	VO_D[25] (O)	meet spec	MIPI_TX_DP4	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPI_TXM3	1.8V	VO_D[26] (O)	meet spec	MIPI_TX_DM3	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPI_TXP3	1.8V	VO_D[27] (O)	meet spec	MIPI_TX_DP3	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPI_TXM2	1.8V	VO_D[20] (O)	meet spec	MIPI_TX_DM2	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_DE	meet spec	VORG8_Data	meet spec	
	MIPI_TXP2	1.8V	VO_CLK0 (O) (CLK) (DATA)	meet spec	MIPI_TX_DP2	meet spec	VO_Data	meet spec	VO656_CLK	meet spec	8080_Data	meet spec	VO601_CLK	meet spec	VORG8_CLK	meet spec	
	MIPI_TXM1	1.8V	VO_D[2] (O)	meet spec	MIPI_TX_DM1	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_VS	meet spec	VORG8_Data	meet spec	
	MIPI_TXP1	1.8V	VO_D[1] (O)	meet spec	MIPI_TX_DP1	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_HS	meet spec	VORG8_Data	meet spec	
	MIPI_TXM0	1.8V	VO_D[4] (O)	meet spec	MIPI_TX_DM0	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	MIPI_TXP0	1.8V	VO_D[3] (O)	meet spec	MIPI_TX_DP0	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	JTAG_CPU_TMS(O)	180D33	VO_D[28] (O) (New)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	
	JTAG_CPU_TCK(CLK)(O)	180D33	VO_D[29] (O) (New)	180D33	GPIO	meet spec	VO_Data	meet spec	VO656_Data	meet spec	8080_Data	meet spec	VO601_Data	meet spec	VORG8_Data	meet spec	

按照上面的原理图的映射关系配置线序



```
c:\src\cvi_cp\sample_panel\sample_panel.c PatchSet Base 1 (GHRB)
54: 001_PANEL_STAT200L, ...skipped 52 common lines...+100,
55: 001_PANEL_87770L,
56: 100L_PANEL_1000000L,
57: 8T_PANEL_PT1000L_PT60L_1000x700_5099L_74M,
58: 8T_PANEL_PT1000L_PT60L_1000x1000_5099L_140M,
59: 8T_PANEL_PT1000L_PT110L_1000x1000_5099L_74M,
60: 8T_PANEL_PT200L_PT60L_1000x700_5099L_72M,
61: 8T_PANEL_08711L_PT60L_1000x700_5099L_72M,
62: 8T_PANEL_PT002L_PT110L_1000x1000_5099L_72M,
63: 100_PANEL_87770PT_PT_MCTL_24x20L_6099L,
64: |
65: |
66: PANEL_M01
67: PANEL_MODEL
68: |
69: |
70: |
71: |
72: |
73: |
74: |
75: |
76: |
77: |
78: |
79: |
80: |
81: |
82: |
83: |
84: |
85: |
86: |
87: |
88: |
89: |
90: |
91: |
92: |
93: |
94: |
95: |
96: |
97: |
98: |
99: |
100: |
101: |
102: |
103: |
104: |
105: |
106: |
107: |
108: |
109: |
110: |
111: |
112: |
113: |
114: |
115: |
116: |
117: |
118: |
119: |
120: |
121: |
122: |
123: |
124: |
125: |
126: |
127: |
128: |
129: |
130: |
131: |
132: |
133: |
134: |
135: |
136: |
137: |
138: |
139: |
140: |
141: |
142: |
143: |
144: |
145: |
146: |
147: |
148: |
149: |
150: |
151: |
152: |
153: |
154: |
155: |
156: |
157: |
158: |
159: |
160: |
161: |
162: |
163: |
164: |
165: |
166: |
167: |
168: |
169: |
170: |
171: |
172: |
173: |
174: |
175: |
176: |
177: |
178: |
179: |
180: |
181: |
182: |
183: |
184: |
185: |
186: |
187: |
188: |
189: |
190: |
191: |
192: |
193: |
194: |
195: |
196: |
197: |
198: |
199: |
200: |
201: |
202: |
203: |
204: |
205: |
206: |
207: |
208: |
209: |
210: |
211: |
212: |
213: |
214: |
215: |
216: |
217: |
218: |
219: |
220: |
221: |
222: |
223: |
224: |
225: |
226: |
227: |
228: |
229: |
230: |
231: |
232: |
233: |
234: |
235: |
236: |
237: |
238: |
239: |
240: |
241: |
242: |
243: |
244: |
245: |
246: |
247: |
248: |
249: |
250: |
251: |
252: |
253: |
254: |
255: |
256: |
257: |
258: |
259: |
260: |
261: |
262: |
263: |
264: |
265: |
266: |
267: |
268: |
269: |
270: |
271: |
272: |
273: |
274: |
275: |
276: |
277: |
278: |
279: |
280: |
281: |
282: |
283: |
284: |
285: |
286: |
287: |
288: |
289: |
290: |
291: |
292: |
293: |
294: |
295: |
296: |
297: |
298: |
299: |
300: |
301: |
302: |
303: |
304: |
305: |
306: |
307: |
308: |
309: |
310: |
311: |
312: |
313: |
314: |
315: |
316: |
317: |
318: |
319: |
320: |
321: |
322: |
323: |
324: |
325: |
326: |
327: |
328: |
329: |
330: |
331: |
332: |
333: |
334: |
335: |
336: |
337: |
338: |
339: |
340: |
341: |
342: |
343: |
344: |
345: |
346: |
347: |
348: |
349: |
350: |
351: |
352: |
353: |
354: |
355: |
356: |
357: |
358: |
359: |
360: |
361: |
362: |
363: |
364: |
365: |
366: |
367: |
368: |
369: |
370: |
371: |
372: |
373: |
374: |
375: |
376: |
377: |
378: |
379: |
380: |
381: |
382: |
383: |
384: |
385: |
386: |
387: |
388: |
389: |
390: |
391: |
392: |
393: |
394: |
395: |
396: |
397: |
398: |
399: |
400: |
401: |
402: |
403: |
404: |
405: |
406: |
407: |
408: |
409: |
410: |
411: |
412: |
413: |
414: |
415: |
416: |
417: |
418: |
419: |
420: |
421: |
422: |
423: |
424: |
425: |
426: |
427: |
428: |
429: |
430: |
431: |
432: |
433: |
434: |
435: |
436: |
437: |
438: |
439: |
440: |
441: |
442: |
443: |
444: |
445: |
446: |
447: |
448: |
449: |
450: |
451: |
452: |
453: |
454: |
455: |
456: |
457: |
458: |
459: |
460: |
461: |
462: |
463: |
464: |
465: |
466: |
467: |
468: |
469: |
470: |
471: |
472: |
473: |
474: |
475: |
476: |
477: |
478: |
479: |
480: |
481: |
482: |
483: |
484: |
485: |
486: |
487: |
488: |
489: |
490: |
491: |
492: |
493: |
494: |
495: |
496: |
497: |
498: |
499: |
500: |
501: |
502: |
503: |
504: |
505: |
506: |
507: |
508: |
509: |
510: |
511: |
512: |
513: |
514: |
515: |
516: |
517: |
518: |
519: |
520: |
521: |
522: |
523: |
524: |
525: |
526: |
527: |
528: |
529: |
530: |
531: |
532: |
533: |
534: |
535: |
536: |
537: |
538: |
539: |
540: |
541: |
542: |
543: |
544: |
545: |
546: |
547: |
548: |
549: |
550: |
551: |
552: |
553: |
554: |
555: |
556: |
557: |
558: |
559: |
560: |
561: |
562: |
563: |
564: |
565: |
566: |
567: |
568: |
569: |
570: |
571: |
572: |
573: |
574: |
575: |
576: |
577: |
578: |
579: |
580: |
581: |
582: |
583: |
584: |
585: |
586: |
587: |
588: |
589: |
590: |
591: |
592: |
593: |
594: |
595: |
596: |
597: |
598: |
599: |
600: |
601: |
602: |
603: |
604: |
605: |
606: |
607: |
608: |
609: |
610: |
611: |
612: |
613: |
614: |
615: |
616: |
617: |
618: |
619: |
620: |
621: |
622: |
623: |
624: |
625: |
626: |
627: |
628: |
629: |
630: |
631: |
632: |
633: |
634: |
635: |
636: |
637: |
638: |
639: |
640: |
641: |
642: |
643: |
644: |
645: |
646: |
647: |
648: |
649: |
650: |
651: |
652: |
653: |
654: |
655: |
656: |
657: |
658: |
659: |
660: |
661: |
662: |
663: |
664: |
665: |
666: |
667: |
668: |
669: |
670: |
671: |
672: |
673: |
674: |
675: |
676: |
677: |
678: |
679: |
680: |
681: |
682: |
683: |
684: |
685: |
686: |
687:
```

因为本屏幕不支持原生 spi 发送数据，故这里采用 gpio 模拟 spi 发送数据的方式，其原理就是通过控制 gpio 引脚进行拉低拉高操作模拟 spi 信号。



6.2.5 烧录测试点屏

在编写好应用层代码后，就可以编译固件烧录至开发板，并且挂载 sample_panel.c 到开发板，在串口输入初始化命令，并执行彩条测试命令，观察屏幕是否正常显示彩条。

```

devmem 0x03001150 32 0x02
devmem 0x0300114c 32 0x02
devmem 0x03001148 32 0x02
devmem 0x03001144 32 0x02
devmem 0x03001140 32 0x02
devmem 0x0300113c 32 0x02
devmem 0x03001160 32 0x02
devmem 0x03001154 32 0x02
devmem 0x03001158 32 0x02
devmem 0x0300115c 32 0x02
devmem 0x03001070 32 0x03
devmem 0x030010D0 32 0x03
devmem 0x030010E4 32 0x03
devmem 0x030010E0 32 0x03
devmem 0x0a088240 32 0x01

```

```
./sample_panel --panel=SRGB666_GC9307_240x320_60FPS --show-pattern=6
```

6.2.6 注意事项

- 1、首先需要强调的是严格执行 Checklist 可以很快排查出问题。
- 2、如果软件层面的逻辑没有问题，首要考虑引脚是否配置正确，其次用示波器去逐一测量引脚的波形和电压是否正确，一定要保证波形和电压同时正确，发送的数据才可以被识别到。
- 3、多与硬件同事沟通，屏幕点亮遇到的问题很多时候都是硬件层面的问题，比如连线错误，规格不适配等等。

6.3 Checklist

Cat.	Check 项目	Check 方法	参考结构	Check 结果	RP& 时间
检查屏幕资料是否与屏幕对应	屏幕规格书	找屏厂确认	与屏幕模组型号一致		
	Dirver ic Datasheet	找屏厂确认	与屏幕模组所使用的 dirver ic 一致		
	屏厂提供的初始码	找屏厂确认	与该屏幕所使用的初始码一致		
根据屏幕资料找出屏幕的基本信息	Power 电压范围	屏幕资料	1.8V 或者 3.3V		
	Reset 电压范围	屏幕资料	1.8V 或者 3.3V		
	Backlight 电压范围	屏幕资料	具体看原理图确认		
检查屏幕外围电路电压	量取 Power 电压	用电压表或者示波器	与屏幕资料描述的一致，1.8V 或者 3.3V		
	量取 Reset 电压	用电压表或者示波器	与屏幕资料描述的一致，1.8V 或者 3.3		
	量取 Backlight 电压	用电压表或者示波器	看屏幕有无背光		
Reset 屏幕上电时序	测量引脚的电平变化情况	示波器测量	电压变化现象是高-低-高或者低-高-低，根据屏幕资料确认		
时钟 clk 确认	量取屏幕时钟 clk 引脚	示波器测量	符合软件设置的时钟值		

7 I80

概述

I80 接口（也称为 8080 接口、MCU 屏接口）是一种并行接口标准，主要用于处理器与 MCU 型 LCD 屏幕之间的连接。常见的屏幕驱动芯片有 ST7789、ILI9341、NT35510 等。相比于 MIPI DSI、LVDS 等高速接口，I80 接口具有以下特点：

- 采用并行数据传输（8 位数据线）
- 独立的读写控制信号，支持双向通信
- 时序简单，抗干扰能力强
- 成本低廉，功耗较低
- 适用于 $240 \times 320 \sim 480 \times 720$ 分辨率的屏幕

本章介绍如何在 CVITEK 处理器解决方案上开发调试 I80 LCD 屏。

7.1 环境准备

7.1.1 I80 屏幕接口介绍

I80 屏幕一般有以下几种信号：

- **I80 数据线 (DATA)**：8 条数据线 (D0~D7)，用于传输像素和命令数据
- **I80 控制信号**：
 - CS (Chip Select)：低电平有效，选中屏幕芯片
 - WR (Write)：负脉冲信号，低有效，用于写数据
 - RD (Read)：负脉冲信号，低有效，用于读数据
 - DC/RS (Data/Command)：区分命令 (0) 和数据 (1)
- **复位信号 (RESET)**：低电平有效，上电后进行复位
- **背光控制 (BACKLIGHT)**：通过 GPIO 或 PWM 控制
- **电源控制 (POWER)**：通过 GPIO 控制或常时供电

I80 接口连线示意

屏幕信号	信号类型	连接处理器	说明
D0~D7	数据线	VO_DATA 数据引脚	8 条数据线
CS	控制信号	VO_DATA 数据引脚	芯片使能，低有效，可不接该引脚
RD	控制信号	VO_DATA 数据引脚	读使能，低脉冲有效
WR	控制信号	VO_DATA 数据引脚	写使能，低脉冲有效
DC(RS)	控制信号	VO_DATA 数据引脚	数据/命令选择
RESET	控制信号	GPIO	复位信号，低有效
POWER(VCC/VD)	电源控制	GPIO	屏幕电源供电控制，可选
BACKLIGHT	控制信号	GPIO 或 PWM	背光控制

7.1.2 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。

I80 数据线和 I80 控制信号一定要连接芯片端可以复用为 VO_D[X] 的引脚，否则无法输出数据。

7.2 配置 I80 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，本节介绍屏幕对接时的软件配置。

CVITEK 有两种方案进行 I80 屏幕的对接，分别是在 u-boot 及 kernel 中进行屏的初始化。u-boot 中进行初始化后开机可以显示 logo，kernel 方式适合无需显示 logo 的应用。实际应用中根据需求二者选其一。

7.2.1 在 u-boot 中配置 I80 屏

u-boot 中配置 I80 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 131072 0;startvl 0 0x84080000 0x81800000 0x80000;setvobg 0 0xffffffff
```

注：对于 I80 屏幕，startvo 中的第二个参数通常为 131072（表示 I80 模式）。

本文档重点讲解屏的初始化部分。其中，屏的初始化部分在” startvo 0 131072 0” 中实现。

7.2.1.1 配置 I80 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。

HW_I80_CFG_S 结构体定义

```
typedef struct _HW_I80_CFG_S {
    enum HW_I80_MODE mode;
    struct VO_PINMUX pins;
    struct HW_I80_INSTRS instrs;
    struct sync_info_s sync_info;
    unsigned short u16FrameRate;
} HW_I80_CFG_S;
```

成员名称	描述
mode	MCU 接口数据位数模式 VO_MCU_MODE_RGB565: 16 位色深, RGB565 格式 VO_MCU_MODE_RGB888: 24 位色深, RGB888 格式
pins	IO 脚位复用配置, 包含 MCU 接口数据线和控制信号线的映射配置
instrs	初始化指令序列, 包含 MCU 屏幕初始化过程中需要下发的命令和数据
sync_info	I80 设备的同步时序参数, 配置 MCU 接口的时序信息
u16FrameRate	刷新帧率, 单位为 Hz。取值范围: 30-120 Hz, 常用值为 30、50、60

VO_PINMUX 结构体定义

```
struct VO_PINMUX {
    unsigned char pin_num;
    struct VO_D_REMAP d_pins[MAX_VO_PINS];
};
```

成员名称	描述
pin_num	I80 pin 脚总数 (包括数据线和控制信号线)
d_pins	Pin 脚映射关系数组, 用于配置处理器物理引脚到 I80 逻辑信号的映射。 包含的映射关系: MCU_DATA0~MCU_DATA7: 8 条数据线 MCU_RD: 读使能信号 MCU_WR: 写使能信号 MCU_RS: 数据/命令选择信号 MCU_CS: 片选信号 (可选)

HW_I80_INSTR_S 结构体定义

```
typedef struct _HW_I80_INSTR_S {
    u8 delay;
```

(下页继续)

(续上页)

```

u8 data_type;
u8 data;
} HW_I80_INSTR_S;

```

成员名称	描述
delay	发送完此指令后的延时时间，单位为毫秒 (ms)。 常见值：0ms（无延时）、120ms（用于 SLPOUT 后） 某些初始化命令（如 SLPOUT 0x11）需要延时，以保证屏幕完成操作
data_type	数据类型，区分命令和数据 0：命令 (PANEL_COMM)，会通过 DC/RS 信号拉低发送 1：数据 (PANEL_DATA)，会通过 DC/RS 信号拉高发送
data	要发送的单字节数据 命令类型：为 MCU 屏幕的命令字，如 0x11(SLPOUT)、0x36(MADCTL)、0x29(DISPON) 等 数据类型：为对应命令的参数值

HW_I80_INSTRS 结构体定义

```

struct HW_I80_INSTRS {
    unsigned char instr_num;
    HW_I80_INSTR_S instr_cmd[MAX_MCU_INSTR];
};

```

成员名称	描述
instr_num	初始化指令的总数，最大值为 MAX_MCU_INSTR (256)
instr_cmd	初始化指令数组，存储所有初始化命令和其参数

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;
    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, 通常等于屏幕的宽度分辨率
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, 通常等于屏幕的高度分辨率
vid_vsa_pos_polarity	垂直同步脉冲极性, true 为低电平有效, false 为高电平有效
vid_hsa_pos_polarity	水平同步脉冲极性, true 为低电平有效, false 为高电平有效

ST7789V3 配置示例

以 ST7789V3 为例 (240 × 320 分辨率, 16bit RGB565 色深) 的配置示例:

```
#define PANEL_COMM 0
#define PANEL_DATA 1

const HW_I80_CFG_S st7789v3Cfg = {
    // 色深模式: RGB565 16位色深
    .mode = VO_MCU_MODE_RGB565,

    // IO脚位复用配置
    .pins = {
        .pin_num = 11,
        .d_pins = {
            {VO_VIVO_D6, VO_MUX_MCU_DATA0}, // 数据线D0
            {VO_VIVO_D5, VO_MUX_MCU_DATA1}, // 数据线D1
            {VO_VIVO_D4, VO_MUX_MCU_DATA2}, // 数据线D2
            {VO_VIVO_D3, VO_MUX_MCU_DATA3}, // 数据线D3
            {VO_VIVO_D2, VO_MUX_MCU_DATA4}, // 数据线D4
            {VO_VIVO_D1, VO_MUX_MCU_DATA5}, // 数据线D5
            {VO_VIVO_D0, VO_MUX_MCU_DATA6}, // 数据线D6
            {VO_VIVO_D7, VO_MUX_MCU_DATA7}, // 数据线D7
            {VO_MIPI_TXM1, VO_MUX_MCU_RD}, // 读使能信号RD
            {VO_MIPI_RXP5, VO_MUX_MCU_WR}, // 写使能信号WR
            {VO_MIPI_TXP1, VO_MUX_MCU_RS}, // 数据/命令选择RS
        }
    },

    // 初始化指令序列
    .instrs = {
        .instr_num = 74,
        .instr_cmd = {
            {.delay = 120, .data_type = PANEL_COMM, .data = 0x11}, // Sleep Out, 需延时120ms
            {.delay = 0, .data_type = PANEL_COMM, .data = 0x36}, // MADCTL - 显存访问方向
            {.delay = 0, .data_type = PANEL_DATA, .data = 0x00}, // MADCTL参数
            {.delay = 0, .data_type = PANEL_COMM, .data = 0x3A}, // COLMOD - 色深设置
            {.delay = 0, .data_type = PANEL_DATA, .data = 0x05}, // COLMOD参数: 0x05表示RGB565
            // ... 其他初始化命令 ...
            {.delay = 0, .data_type = PANEL_COMM, .data = 0x29}, // Display On - 显示打开
        }
    }
};
```

(下页继续)

(续上页)

```

    {.delay = 0, .data_type = PANEL_COMM, .data = 0x2C}, // RAMWR - 准备写显存
}
},

// 时序信息
.sync_info = {
    .vid_hsa_pixels = 5,           // 水平同步脉冲宽度：5像素
    .vid_hbp_pixels = 20,         // 水平消隐后肩：20像素
    .vid_hfp_pixels = 10,         // 水平消隐前肩：10像素
    .vid_hline_pixels = 240,      // 水平有效区：240像素
    .vid_vsa_lines = 5,           // 垂直同步脉冲宽度：5行
    .vid_vbp_lines = 5,           // 垂直消隐后肩：5行
    .vid_vfp_lines = 20,          // 垂直消隐前肩：20行
    .vid_active_lines = 320,      // 垂直有效区：320行
    .vid_vsa_pos_polarity = true, // 垂直同步脉冲为低有效
    .vid_hsa_pos_polarity = true, // 水平同步脉冲为低有效
},
.u16FrameRate = 60, // 刷新率：60Hz
};

```

常用 MCU 屏命令表

命令字	名称	说明
0x11	SLPOUT	退出睡眠模式（需延时 120ms）
0x29	DISPON	显示打开
0x2A	CASET	列地址设置
0x2B	RASET	行地址设置
0x2C	RAMWR	写显存
0x36	MADCTL	显存访问控制
0x3A	COLMOD	色深设置（0x05=RGB565）

7.2.1.2 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 中增加对上节中新增头文件的引用。

示例：

```

#ifdef I80_PANEL_ST7789V3
#include "i80_hw_st7789v3.h"
static struct panel_desc_s panel_desc = {
    .i80_hw_cfg = &st7789v3_cfg,
};
#endif

```


7.2.1.3 配置 RESET 管脚

查询硬件原理图，获取 RESET 管脚对应的 GPIO 号，在 build/default/dts/cv181x/cv181x_base.dtsi 中修改如下：

```
reset-gpio = <&porte 21 GPIO_ACTIVE_LOW>;
```

7.2.1.4 配置 BACKLIGHT 管脚

背光可配置为 GPIO 或 PWM。

GPIO 模式：

```
backlight-gpio = <&porta 4 GPIO_ACTIVE_HIGH>;
```

PWM 模式：参考其他屏幕类型的 PWM 配置。

7.2.1.5 配置 POWER 管脚

电源管脚用于控制屏幕的 VCC/VDD 电源供电，某些屏幕和应用场景需要通过 GPIO 控制电源的开关。查询硬件原理图，获取 POWER 管脚对应的 GPIO 号，在 build/default/dts/cv181x/cv181x_base.dtsi 中修改如下：

```
power-gpio = <&porta 1 GPIO_ACTIVE_HIGH>;
```

注：如果屏幕电源常时供电，无需配置此管脚。

7.2.1.6 配置 u-boot 环境变量，更换 logo 图片，编译烧写验证

具体步骤参考前面 MIPI 屏幕的内容。

7.2.2 在 kernel 中配置 I80 屏

在 kernel 中配置 I80 屏的方法与 u-boot 类似。当无需显示 logo 时，可选择此种方式。

7.2.2.1 配置 I80 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 middleware/v2/component/panel/cv181x/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 8.2.1.1 节

7.2.2.2 添加头文件的引用

在 `middleware/v2/sample/sample_panel/sample_panel.c` 中增加对 I80 屏幕头文件的引用。

示例：

如要点屏 ST7789V3，首先在 `middleware/v2/sample/sample_panel/sample_panel.h` 中 `include` 该屏幕的头文件 `hw_mcu_st7789v3.h`，然后确保 `middleware/v2/sample/sample_panel/sample_panel.c` 文件中的字符数组 `static char *s_panel_model_type_arr[]` 里有“ST7789V3”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 `ST7789V3`，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 I80_PANEL_ST7789V3_HW_MCU_240x320_60FPS:
    g_panel_desc.panel_type = PANEL_MODE_MCU;
    g_panel_desc.stVoPubAttr.enIntfType = VO_INTF_HW_MCU;
    g_panel_desc.stVoPubAttr.enIntfSync = VO_OUTPUT_USER;
    VO_SYNC_INFO_S st7789V3_SyncInfo = {.bSynm = 1, .bIop = 1, .u16FrameRate = 60,
    , .u16Vact = 320, .u16Vbb = 5, .u16Vfb = 20
    , .u16Hact = 240, .u16Hbb = 20, .u16Hfb = 10
    , .u16Vpw = 5, .u16Hpw = 5, .bIdv = 0, .bIhs = 1, .bIvs = 1};
    g_panel_desc.stVoPubAttr.stSyncInfo = st7789V3_SyncInfo;
    g_panel_desc.stVoPubAttr.stMcuCfg = st7789v3Cfg;
    break;
```

7.2.2.3 配置 I80 屏 RESET、POWER、BACKLIGHT 管脚

方法 1：

仅在 `cv181x` 单系统支持屏幕头文件中配置 I80 屏的 GPIO 信息，如果没有该管脚，则直接不写即可。

示例：

```
.lcd_power_gpio_num = GPIOB_03,
.lcd_power_avtive = GPIO_ACTIVE_HIGH,
.backlight_gpio_num = GPIOA_30,
.backlight_avtive = GPIO_ACTIVE_HIGH,
.reset_gpio_num = GPIOE_13,
.reset_avtive = GPIO_ACTIVE_LOW,
```

各 GPIO 配置说明：

- **power_gpio**：屏幕电源控制管脚，可选。极性通常为 `GPIO_ACTIVE_HIGH`（高电平有效）
- **reset_gpio**：屏幕复位管脚，必需。极性通常为 `GPIO_ACTIVE_LOW`（低电平有效）
- **backlight_gpio**：背光控制管脚，可选。极性通常为 `GPIO_ACTIVE_HIGH`（高电平有效）

方法 2：

直接在应用层直接控制 GPIO。

示例：假设 `reset`: `GPIOB5`, `pwm`: `GPIOB3`, `power`: `GPIOB4`, 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 GPIOB 的编号范围，若得出 gpiochip1 的范围是 448 至 479，则对于所要拉的 GPIOB5 来说，下面 echo 操作的号码可由 $448+5=453$ 得到。

因此需要以下操作去拉对应引脚：

```
1. echo 453 > /sys/class/gpio/export
   echo 451 > /sys/class/gpio/export
   echo 452 > /sys/class/gpio/export
2. echo out > /sys/class/gpio/gpio453/direction
   echo out > /sys/class/gpio/gpio451/direction
   echo out > /sys/class/gpio/gpio452/direction
3. echo 1 > /sys/class/gpio/gpio453/value
   echo 1 > /sys/class/gpio/gpio451/value
   echo 1 > /sys/class/gpio/gpio452/value
   echo 0 > /sys/class/gpio/gpio453/value
   echo 1 > /sys/class/gpio/gpio453/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。

7.2.2.4 编译验证

执行 build_middleware 编译 middleware，在路径 middleware/v2/sample/sample_panel/下会生成 sample_panel 可执行文件。该程序和 u-boot 中 “startvo 0 131072 0” 做的事情是一样的，设置 I80 设备属性并向屏幕发送初始化序列，显示图像。

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=ST7789V3_HW_MCU_RGB565_240x320_60FPS

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 0x0a088094 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，则需要再检查数据引脚、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。

7.3 问题 DEBUG

7.3.1 屏幕点不亮问题排查

第一步：检查背光信号

- 观察屏幕是否有背光，如果没有，检查背光控制 GPIO 配置和极性是否正确，确认 GPIO 是否有输出。

第二步：检查数据线和控制信号

- 用示波器观察 D0~D7、CS、WR、RD、DC 信号波形，确认信号逻辑正确，时序和电压符合屏幕规格书要求

第三步：检查复位信号

- 用示波器观察 RESET 管脚波形
- 上电后应该：拉低 10~20ms → 拉高 → 保持高电平 120ms 以上

第四步：检查初始化序列

- 确认 SLPOUT 命令后延时至少 120ms（可选）
- 检查色深设置 (COLMOD 0x3A) 与实际匹配
- 确认显存地址范围 (CASET/RASET) 正确

7.3.2 常见问题

现象	检查项
屏幕黑屏	检查电源、复位、初始化命令是否正确
屏幕显示异常（倒转、镜像等）	调整 MADCTL(0x36) 参数
屏幕颜色异常	检查 COLMOD 设置，确认色深正确
背光不亮	检查 GPIO 配置和极性
电源无法控制	检查 power-gpio 配置，确认 GPIO 和极性设置正确
屏幕闪烁/撕裂	启用 TE(0x35) 同步信号