# CV182x/CV181x
# 屏幕对接使用指南

Version: 1.2.2
Release date: 2022-06-23

# 修订记录

| Revision | Date | Author | Description |
|---|---|---|---|
| 0.1 | 2021/04/20 | Jammy Huang | Initial version |
| 1.1.1 | 2021/06/11 | Jammy Huang | Modify some typo and description |
| 1.2.0 | 2021/10/26 | Xinwei Jiang | Revision update |
| 1.2.1 | 2022/02/07 | Xinwei Jiang | Revision update |
| 1.2.1.0 | 2022/06/15 | Felix Peng | Update for CV181x |
| 1.2.2 | 2022/06/23 | Xinwei Jiang | Revision update |
| | | | |
| | | | |

# 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。

# 目录

# 1 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 芯片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。

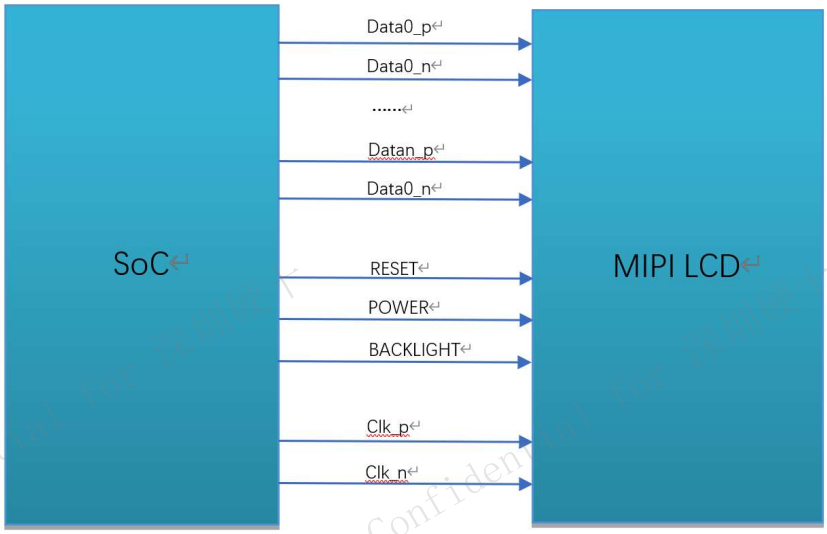
## 1.1 环境准备

### 1.1.1 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**



### 1.1.2 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。

## 1.2 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。

### 1.2.1 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，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。

#### 1.2.1.1 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/include/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 | 主控端和屏端 Lane 号的对应关系，未使用的 Lane 填-1 即可。<br>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。<br>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。<br>对应关系不正确，将导致屏幕无法点亮。 |
| 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 | 像素时钟，单位为 KHz。<br>计算公式：<br>pixel_clk=(htotal*vtotal)*fps/1000<br>其中：<br>htotal=vid_hsa_pixels+ vid_hbp_pixels+ vid_hfp_pixels+ vid_hline_pixels<br>vtotal= vid_vsa_lines+ vid_vbp_lines+ vid_vfp_lines+ vid_active_lines<br>fps:帧率，默认 60<br>lane_clk 根据 pixel_clk 反推，换算公式：<br>lane_clk= 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) |
| lane_pn_swap | MIPI Tx 的 Lane P/N 极是否交换<br>true：交换<br>false:不交换 |

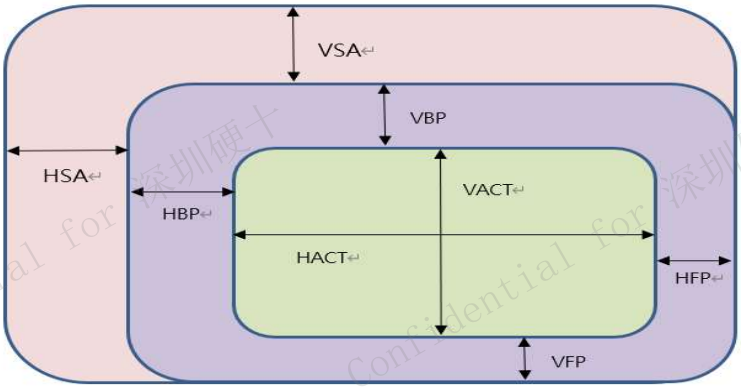
combo_dev_cfg_s 中 sync_info（MIPI Tx 设备的同步信息）比较难配置，下面详细介绍它的配置方法。一般开始会根据屏厂提供的规格书填写参考值，还有问题再根据现象调整。

**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;
        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**

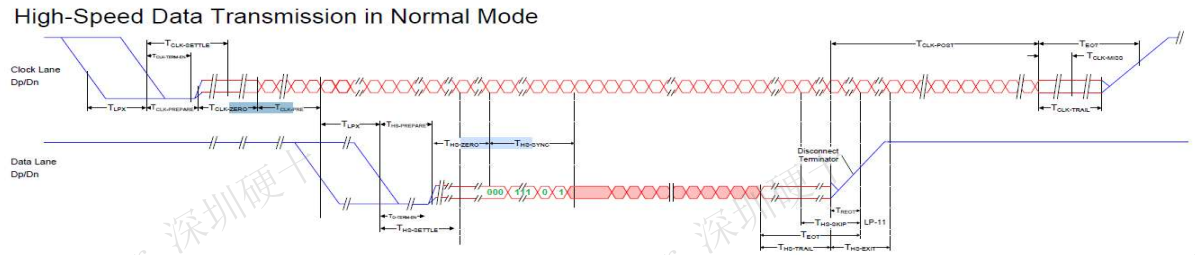


**hs_settle_s 结构体定义**

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

#### 1.2.1.2 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。<br>类型 1、当只有寄存器地址没有数据时，数据类型选择 0x05；<br>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；<br>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。<br>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。<br>例如当只有一个寄存器地址，填 1；<br>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。<br>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_xxxx_0[] = { 0xFF, 0x98, 0x81, 0x03 };
static u8 data_xxxx_1[] = { 0x01, 0x00 };
static u8 data_xxxx_2[] = {0x02, 0x00 };
......
static u8 data_xxxx_n[] = { 0x11 };
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 },
}
```

#### 1.2.1.3 添加头文件的引用

添加对该新增的头文件的引用，在 u-boot/include/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
```

#### 1.2.1.4 配置 MIPI 屏 RESET 管脚

在 u-boot/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《CV182X_PINOUT_CN》找到管脚对应的 GPIO 组号及序号。
- 修改 build/boards/cv182x/cv18xx/u-boot/cvitek.h 中 VO_GPIO_RESET_PORT、VO_GPIO_RESET_INDEX、VO_GPIO_RESET_ACTIVE 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/cv182x/cv18xx/u-boot/cvitek.h 修改如下：

```
#define VO_GPIO_RESET_PORT          porte
#define VO_GPIO_RESET_INDEX         2
#define VO_GPIO_RESET_ACTIVE        GPIO_ACTIVE_LOW
```

配置如下：

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

操作如下：

```
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIOD_ACTIVE_LOW ? 0 : 1);
mdelay(10);
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIOD_ACTIVE_LOW ? 1 : 0);
mdelay(10);
dm_gpio_set_value(&ctrl_gpios.disp_reset_gpio, ctrl_gpios.disp_reset_gpio.flags & GPIOD_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

#### 1.2.1.5 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/boards/cv182x/cv18xx/u-boot/cvitek.h 修改如下：

```
#define VO_GPIO_POWER_CT_PORT          porte
#define VO_GPIO_POWER_CT_INDEX         0
#define VO_GPIO_POWER_CT_ACTIVE        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, GPIOD_IS_OUT | GPIOD_IS_OUT_ACTIVE);
```

操作如下：

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

#### 1.2.1.6 配置 MIPI 屏 BACKLIGHT 管脚

MIPI 屏 BACKLIGHT 可以配置为 GPIO 或者 PWM。

##### 1.2.1.6.1 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/boards/cv182x/cv18xx/u-boot/cvitek.h 修改如下：

```
#define VO_GPIO_PWM_PORT          porte
#define VO_GPIO_PWM_INDEX         1
#define VO_GPIO_PWM_ACTIVE        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);
```

#### 1.2.1.6.2 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/board/cvitek/cv1822/board.c 的 board_init 函数中，配置 BACKLIGHT 管脚复用功能为 PWM 功能。
- 对照《CV182X 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《CV182X Preliminary Datasheet》，CV182X 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
```

#### 1.2.1.7 配置 u-boot 环境变量

修改 u-boot/include/configs/cv1822-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。

#### 1.2.1.8 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 。

#### 1.2.1.9 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。

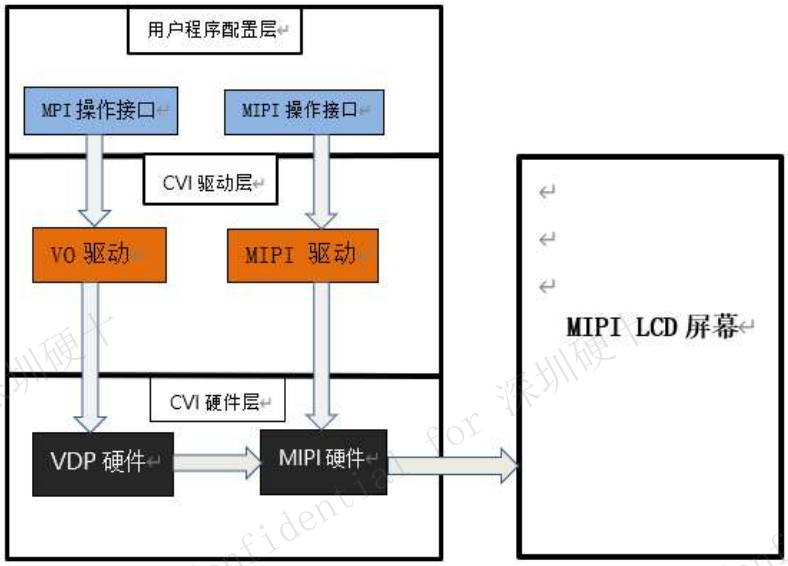
假如 BIST 正常，则说明以上配置正确，硬件电路没有异常，这时通常需要调整 sync_info_s 中的各参数。

## 1.2.2 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**




### 1.2.2.1 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/component/panel/cv182x/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 1.2.1.1 节

### 1.2.2.2 配置屏幕初始化序列

参见 1.2.1.2 节

#### 1.2.2.3 添加头文件的引用

添加对该新增的头文件的引用，在 middleware/sample/mipi_tx/sample_dsi_panel.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
```

#### 1.2.2.4 配置 MIPI 屏 RESET、POWER、BACKLIGHT 管脚

**方法 1：**

在路径 linux/arch/arm/boot/dts/cvitek/下找到对应的 dts 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 {
    compatible = "cvitek,mipi_tx";
    clocks = <&clk CV182X_CLK_DSI_MAC_VIP>, <&clk CV182X_CLK_DISP_VIP>;
    clock-names = "clk_dsi", "clk_disp";
    reset-gpio = <&portb 5 GPIO_ACTIVE_LOW>;
    pwm-gpio = <&portb 3 GPIO_ACTIVE_HIGH>;
    power-ct-gpio = <&portb 4 GPIO_ACTIVE_HIGH>;
};
```

说明：

```
pwm-gpio = <&portb 3 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：

```
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。

#### 1.2.2.5 编译验证

执行 build_middleware 编译 middleware，在路径 middleware/sample/mipi_tx/下会生成 sample_dsi 可执行文件。该程序和在 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_dsi 拷贝至设备，运行。此时不出意外，RESET 管脚在 MIPI Tx 驱动中会自动进行 high-low-high（RESET 初始电平设置为 low）电平变换，屏幕被供电，背光点亮。

说明：

RESET pin 会在执行 sample_dsi 后由驱动自动进行 RESET 时序控制，用户无需介入。

RESET 初始电平设置为 low,将产生 high-low-high 时序变化。

RESET 初始电平设置为 high,将产生 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 LVDS

## 概述

Low Votage 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 芯片解决方案上开发调试 LVDS LCD 屏。

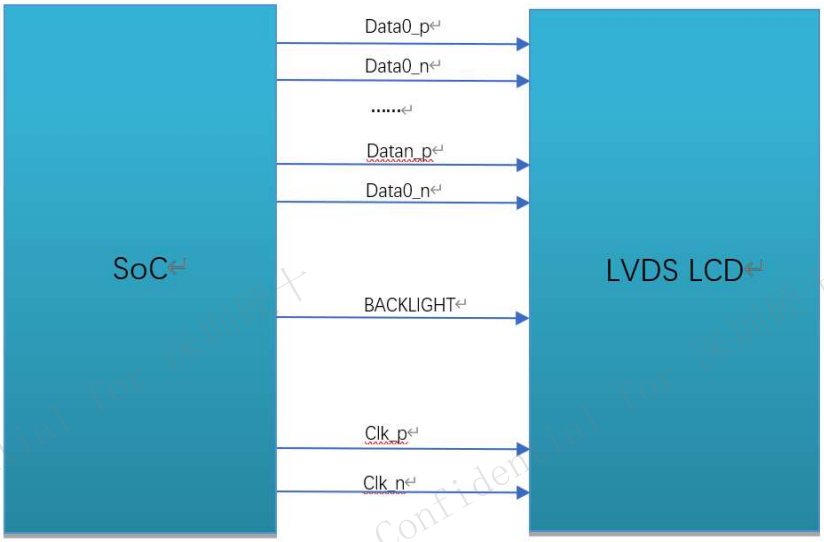
## 2.1 环境准备

### 2.1.1 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**



### 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 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。

### 2.2.1 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。

#### 2.2.1.1 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/include/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_JEIDA、LVDS_MODE_VESA，一般设置为 LVDS_MODE_VESA |
| chn_num | 通道数 1、2，现在芯片仅支持 1 通道 |
| data_big_endian | 发送数据的大小端顺序，一般设置 false |
| Lane_id | 主控端和屏端 Lane 号的对应关系，未使用的 Lane 填-1 即可。<br>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。<br>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。<br>对应关系不正确，将导致屏幕无法点亮。 |
| lane_pn_swap | LVDS 的 Lane P/N 极是否交换<br>true：交换<br>false:不交换 |
| sync_info | LVDS 设备的同步信息 |
| pixel_clk | 像素时钟，单位为 KHz。<br>计算公式：<br>pixel_clk=(htotal*vtotal)*fps/1000<br>其中：<br>htotal=vid_hsa_pixels+ vid_hbp_pixels+ vid_hfp_pixels+ vid_hline_pixels<br>vtotal= vid_vsa_lines+ vid_vbp_lines+ vid_vfp_lines+ vid_active_lines<br>fps:帧率，默认 60<br>lane_clk 根据 pixel_clk 反推，换算公式：<br>lane_clk= pixel_clk*24/4/2(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) |

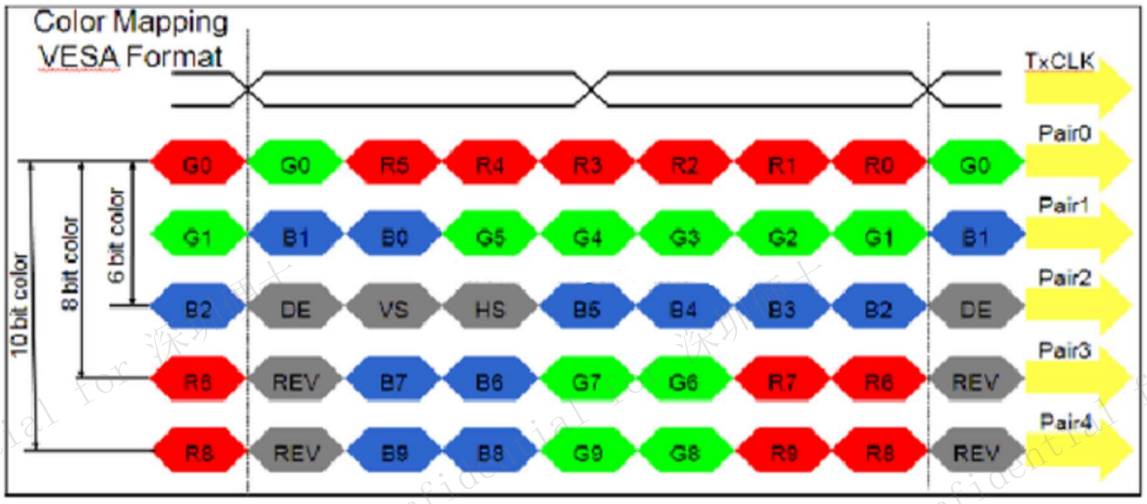
示例：

```
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 1.2.1.1。

**LVDS 时序图**



#### 2.2.1.2 添加头文件的引用

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

示例：

```
#if defined(LVDS_PANEL_EK79202)
#include "lvds_ek79202.h"
static struct panel_desc_s panel_desc = {
      .lvds_cfg = &lvds_ek79202_cfg
};
#endif
```

#### 2.2.1.3 配置 LVDS 屏 BACKLIGHT 管脚

LVDS 屏的 BACKLIGHT 可以配置为 GPIO 或者 PWM。

##### 2.2.1.3.1 配置为 GPIO

可通过修改 build/boards/cv182x/cv18xx/u-boot/cvitek.h 中 VO_GPIO_PWM_PORT、VO_GPIO_PWM_INDEX、VO_GPIO_PWM_ACTIVE 实现。

##### 2.2.1.3.2 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 1.2.1.6 小节。

#### 2.2.1.4 配置 u-boot 环境变量

实现与 MIPI 屏类似，参见 1.2.1.7 小节。

#### 2.2.1.5 更换 logo 图片

实现与 MIPI 屏类似，参见 1.2.1.8 小节。

#### 2.2.1.6 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。

## 2.2.2 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。

### 2.2.2.1 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/component/panel/cv182x/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 添加头文件的引用

添加对该新增的头文件的引用，在 middleware/component/panel/cv82x/lvds_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"
const VO_LVDS_ATTR_S *pstLvdsAttr = &lvds_ek79202_cfg;
#endif
```

#### 2.2.2.3 配置 LVDS 屏 BACKLIGHT 管脚

在路径 middleware/component/panel/cv82x/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。

#### 2.2.2.4 编译验证

LVDS 在未打开 logo 情况下，需运行 APP 才能出图。屏的初始化参见 middleware/sample/common/sample_common_platform.c，需修改 stDefDispRect 和 stDefImageSize 修改为屏实际大小，stVoConfig.stVoPubAttr.enIntfType 修改为 VO_INTF_LCD_24BIT（VO_INTF_LCD_18BIT、VO_INTF_LCD_30BIT），stVoConfig.stVoPubAttr.enIntfSync 修改为 VO_OUTPUT_1280x800_60，其他部分参见

middleware/sample/common/sample_common_vo.c 中 SAMPLE_COMM_VO_FillIntfAttr 和 SAMPLE_COMM_VO_StartDev 的实现。

运行 APP 中 CVI_VO_SetPubAttr 和在 u-boot 中“startvo 0 2048 0”做的事情是一样的，初始化 LVDS 屏并让于工屏处作状态。

如未能正常显示，可参见 2.2.1.6.