



# **AW LCD 开发指南**

**版本号: 0.2  
发布日期: 2024.9.4**

## 版本历史

版本号	日期	制/修订人	内容描述
0.1	2024.3.15	AWA2081	初始版本
0.2	2024.9.4	AWA2257	1.add dts lvds lane np reverse interface. 2.add dts tcon de hsync vsync polar reverse interface. 3.add dts dual-link even odd swap interface. 4.spread spectrum function configuraion guide.



# 目 录

<b>1 概述</b>	<b>1</b>
1.1 编写目的	1
1.2 适用范围产品	1
1.3 相关人员	1
<b>2 相关术语介绍</b>	<b>2</b>
<b>3 LCD 屏适配步骤</b>	<b>3</b>
<b>4 屏驱动接口</b>	<b>5</b>
4.1 LCD_open_flow	5
4.2 LCD_OPEN_FUNC	6
4.3 LCD_close_flow	7
4.4 LCD_CLOSE_FUNC	8
4.5 sunxi_lcd_delay_ms/us	9
4.6 sunxi_lcd_tcon_enable/disable	9
4.7 sunxi_lcd_backlight_enable/disable	9
4.8 sunxi_lcd_pwm_enable/disable	9
4.9 sunxi_lcd_power_enable/disable	10
4.10 sunxi_lcd_pin_cfg	10
4.11 sunxi_lcd_gpio_set_value	10
4.12 sunxi_lcd_gpio_set_direction	11
4.13 sunxi_lcd_dsi_clk_enable/disble	11
4.14 sunxi_lcd_dsi_dcs_wr	12
4.15 sunxi_lcd_dsi_dcs_wr_2para	12
4.16 sunxi_lcd_dsi_dcs_read	12
4.17 sunxi_lcd_dsi_gen_write	13
<b>5 DTS 适配步骤</b>	<b>14</b>
5.1 屏驱动选择	14
5.2 选择接口及模式。	14
5.3 配置屏时序	14
5.4 背光配置	15
5.5 接口的详细设置。	16
5.6 显示效果相关	17
5.7 管脚和电源的配置	17
5.7.1 lcd_power	18
5.7.2 lcd_pin_power	18
5.7.3 lcd_gpio_x	18

5.7.4	pinctrl-0 和 pinctrl-1	19
<b>6</b>	<b>MIPI-DSI</b>	<b>21</b>
6.1	LCD 节点接口类型	21
6.2	DTS 适配	23
6.2.1	single-mipi-dsi0	23
6.2.2	single-mipi-dsi1	24
6.2.3	dual-mipi-dsi	24
6.3	驱动移植	25
<b>7</b>	<b>LVDS</b>	<b>27</b>
7.1	LCD 节点接口类型	27
7.1.1	Single-lvds	27
7.1.2	dual-lvds	28
7.2	DTS 适配	31
7.2.1	LVDS Single link	31
7.2.2	LVDS Dual link	32
7.3	驱动移植	34
<b>8</b>	<b>RGB</b>	<b>36</b>
8.1	LCD 节点接口类型	36
8.2	DTS 适配	37
8.2.1	并行 RGB0 接口	37
8.2.2	串行 RGB0 接口	38
8.3	驱动移植	39
<b>9</b>	<b>RGB 转 VGA</b>	<b>41</b>
9.1	lcd 节点配置	41
9.1.1	lcd_driver_name	41
9.1.2	lcd_convert_if	41
9.2	驱动说明	41
9.2.1	驱动源码位置	41
<b>10</b>	<b>incell 屏</b>	<b>43</b>
10.0.1	概述	43
10.0.2	驱动移植适配	43
10.0.3	适配休眠唤醒机制	43
10.0.3.1	唤醒	43
10.0.3.2	休眠	44
<b>11</b>	<b>多屏兼容功能</b>	<b>45</b>
11.1	功能说明	45
11.2	使用方法	46
<b>12</b>	<b>判断是否支持某款 MIPI-DSI 屏</b>	<b>48</b>

<b>13 ESD 静电检测自动恢复功能</b>	<b>49</b>
13.1 kernel menuconfig 配置	49
13.2 功能使用范例	49
13.2.1 esd_check	50
13.2.2 reset_panel	51
13.2.3 set_esd_info	52
<b>14 展频</b>	<b>54</b>
14.1 展频概述	54
14.2 LVDS 展频功能和快速适配方法	55
<b>15 调试方法</b>	<b>57</b>
15.1 加快调试速度的方法	57
15.2 查看显示信息	57
15.3 查看电源信息	58
15.4 查看 pwm 信息	59
15.5 查看管脚信息	59
15.6 查看时钟信息	59
15.7 查看接口自带 colorbar	60
15.8 重启 lcd 显示通路	61
<b>16 F&amp;Q</b>	<b>62</b>
16.1 屏显示异常	62
16.2 黑屏-无背光	62
16.3 黑屏-有背光	62
16.4 闪屏	63
16.5 条形波纹	63
16.6 重启断电测试屏异常	63
16.7 RGB 接口或者 I8080 接口显示抖动有花纹	64
16.8 LCD 屏出现极化和残影	64
<b>17 总结</b>	<b>65</b>

## 插 图

图 3-1	屏驱动流程图	3
图 4-1	power on	6
图 4-2	power off	8
图 5-1	参考屏时序	15
图 6-1	dsi0 输出示意图	21
图 6-2	dsi1 输出示意图	21
图 6-3	dual-dsi 输出示意图	22
图 6-4	dual-dsi 输出示意图-1	22
图 7-1	lvds0 输出示意图	27
图 7-2	lvds1 输出示意图	27
图 7-3	lvds2 输出示意图	28
图 7-4	lvds3 输出示意图	28
图 7-5	dual-lvds0+1 输出示意图	29
图 7-6	dual-lvds0+1 输出示意图-1	29
图 7-7	dual-lvds2+3 输出示意图	30
图 7-8	dual-lvds2+3 输出示意图-1	30
图 8-1	rgb0 输出示意图	36
图 8-2	rgb1 输出示意图	36
图 10-1	incell_lcd_power_on	44
图 10-2	incell_lcd_power_off	44
图 11-1	compatible_lcd_macro	46
图 11-2	屏驱动切换示例	47
图 11-3	屏驱动切换延时	47
图 13-1	menuconfig	49
图 13-2	屏驱动方法结构体配置	49
图 13-3	0x0A 命令	51
图 13-4	复位函数示例 1	52
图 14-1	sscg	54
图 14-2	ssc_way	55
图 14-3	pll_video0	56
图 15-1	colorbar	61

# 1 概述

---

## 1.1 编写目的

本文档主要描述 sunxi 平台 display2 显示框架的底层 LCD 显示配置，以及相关调试手段。

## 1.2 适用范围产品

适用平台：A523、A527、T527、H135、H136

## 1.3 相关人员

系统整合人员，显示开发相关人员。

## 2 相关术语介绍

表 2-1: LCD 相关术语

术语	解释说明
SUNXI	Allwinner 一系列 SoC 硬件平台
LCD	Liquid Crystal Display, 液晶显示器
MIPI	Mobile Industry Processor Interface
DSI	Display Serial Interface, 显示串行接口
I8080	Intel 8080LCD 接口
RGB	这里指一种 LCD 接口, 该接口发送不经过任何编码的 RGB 分量 <sup>®</sup>
LVDS	Low-Voltage Differential Signaling 一种 LCD 接口, 低压和差分传输是它特点



### 3 LCD 屏适配步骤

适配一款 LCD 屏 uboot 和 kernel 都需要完成如下两部分操作：

#### 屏驱动适配

定义一个新的 struct \_\_lcd\_panel 结构体，实现.cfg\_open\_flow、.cfg\_close\_flow 的回调函数，具体流程如下图所示，各接口的入参不了解的请看[屏驱动接口](#)有相关介绍：

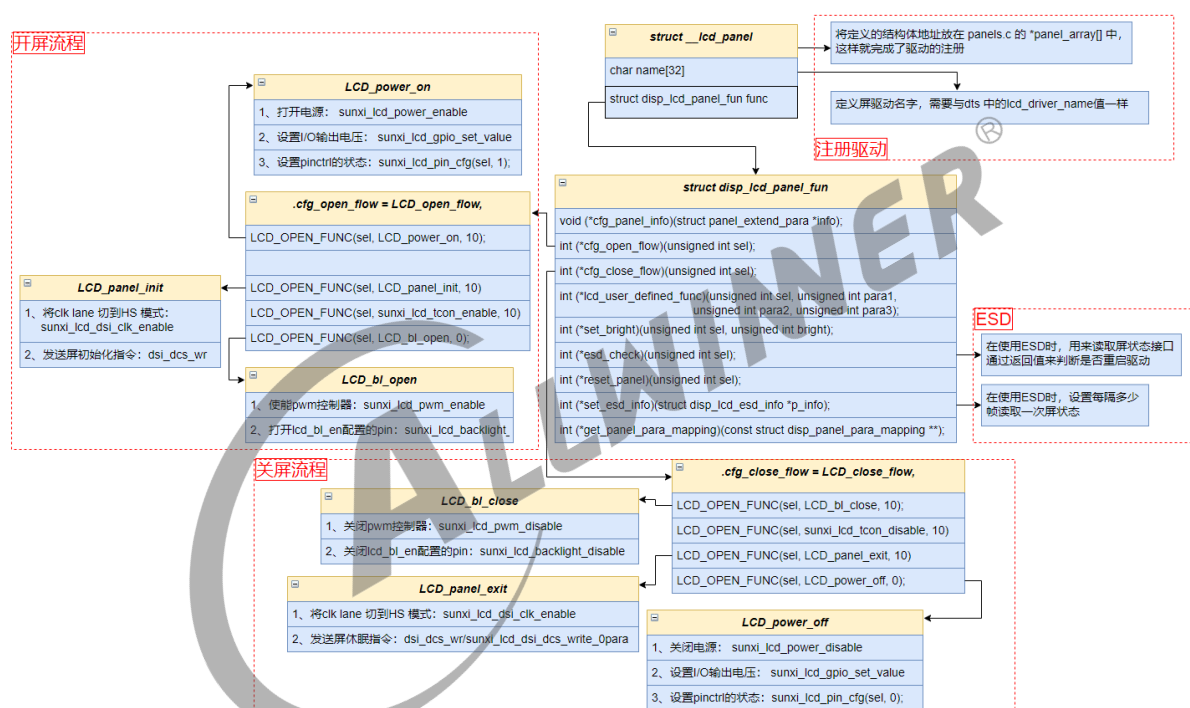


图 3-1: 屏驱动流程图

#### DTS 适配

在 dts 适配时，不同的接口需要放在对应的 lcd 节点上做适配，配错屏幕是无法显示的，如下表所示：

节点	支持的 lcd 显示接口
lcd0	dsi0、rgb0、lvds0、lvds1、lvds0+lvds1、dsi0+dsi1
lcd1	dsi1
lcd2	lvds2、lvds3、lvds2+lvds3、rgb1

## 1、disp 节点适配

- 该 lcd 屏做主显时：

screen0\_output\_type = <1>;

dev0\_output\_type = <1>;

screen0\_to\_lcd\_index 值是使用的 lcd 节点号

- 该 lcd 屏做幅显时：

screen1\_output\_type = <1>;

dev1\_output\_type = <1>;

screen1\_to\_lcd\_index 值是使用的 lcd 节点号

## 2、lcd 节点适配

按照DTS 适配步骤章节的各小节的顺序完成适配；需要注意：若有 I/O 的适配lcd\_gpio\_x、lcd\_bl\_en，uboot 和 kernel 的值是不一样的，如下配置 PD25 时：

uboot: lcd\_gpio\_0 = <&pio PD 25 1 0 3 1>;

kernel: lcd\_gpio\_0 = <&pio PD 25 GPIO\_ACTIVE\_HIGH>;

## 4 屏驱动接口

### 4.1 LCD\_open\_flow

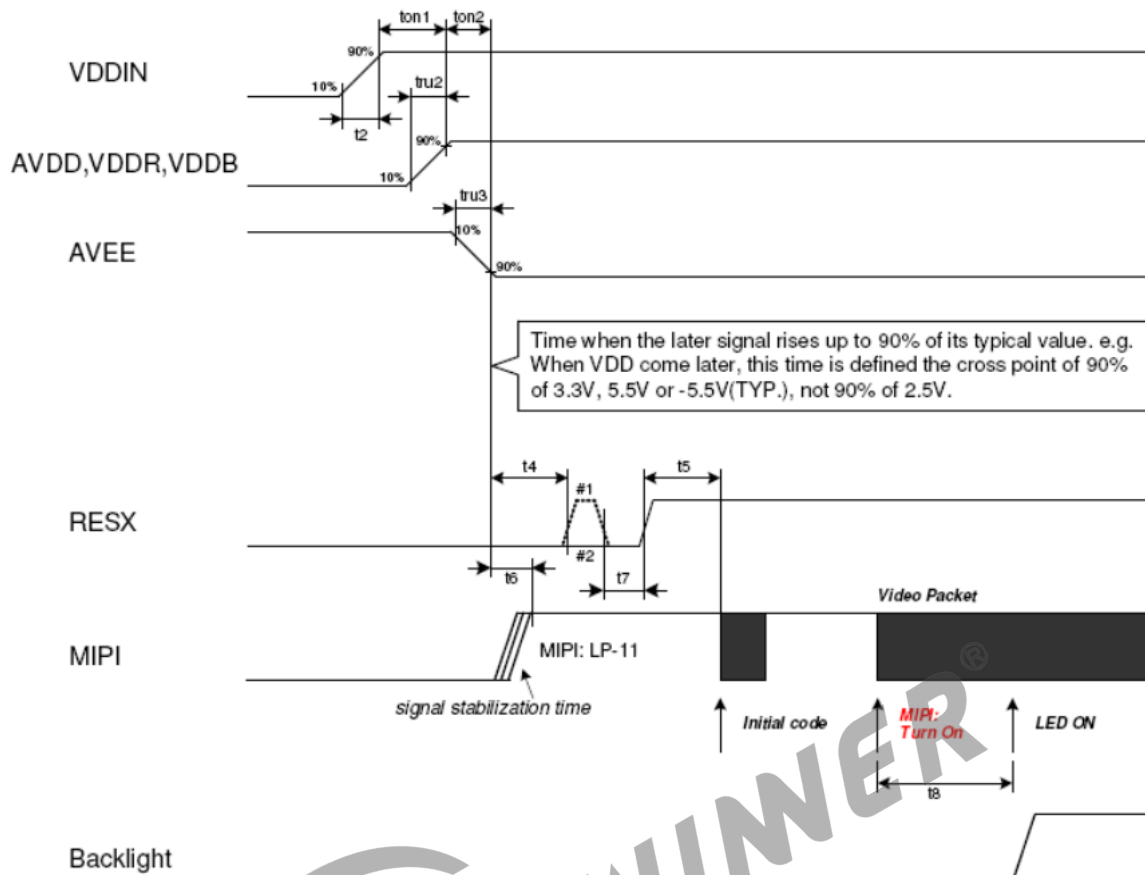
- 函数原型：static s32 \_\_LCD\_open\_flow(u32 sel);
- 功能：LCD\_open\_flow 函数只会在系统初始化的时候调用一次，执行每个 LCD\_OPEN\_FUNC 即是把对应的开屏步骤函数进行注册，先注册先执行，但并没有立刻执行该开屏步骤函数。
- 函数常用内容为：

```
static __s32 LCD_open_flow(__u32 sel)
{
    LCD_OPEN_FUNC(sel, LCD_power_on, 10);
    LCD_OPEN_FUNC(sel, LCD_panel_init, 50);
    LCD_OPEN_FUNC(sel, sunxi_lcd_tcon_enable, 100);
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0);
    return 0;
}
```

如上，调用四次 LCD\_OPEN\_FUNC 注册了四个回调函数，对应了四个开屏流程，先注册先执行。实际上注册多少个函数是用户自己的自由，只要合理即可。

1. **LCD\_power\_on**：打开 LCD 电源，再延迟 10ms；这个步骤一般用于打开 LCD 相关电源和相关管脚比如复位脚。
2. **LCD\_panel\_init**：即初始化屏，再延迟 50ms；不需要初始化的屏，可省掉此步骤，这个函数一般用于发送初始化命令给屏进行屏初始化。
3. **sunxi\_lcd\_tcon\_enable**：打开 TCON，再延迟 100ms；这一步是固定的，表示开始发送图像信号。
4. **LCD\_bl\_open**：打开背光，再延迟 0ms。前面三步搞定之后才开背光，这样不会看到闪烁。

如下图，这是屏手册中典型的上电时序图，我们编写屏驱动的时候，也要注意，该延时就得延时。



Note 1: Unless otherwise specified, timings herein show cross point at 50% of signal/power level.

Note 2: This power-on sequence is based on adding schottky diode on VGLX pin to ground.

Note 3: Reset signal H to L to H (#1) is better than only L to H (#2).

图 4-1: power on

## 4.2 LCD\_OPEN\_FUNC

- 函数原型：void LCD\_OPEN\_FUNC(u32 sel, LCD\_FUNC func, u32 delay);
- 作用：注册开屏步骤函数到开屏流程中，记住这里是注册不是执行！
- 参数：
- sel：lcd 节点索引。
- delay：是执行该步骤后，再延迟的时间，时间单位是毫秒。
- func：一个函数指针，其类型是：void (\*LCD\_FUNC) (\_\_u32 sel)，用户自己定义的函数必须也要用统一的形式。比如：

```
— c void user_defined_func(__u32 sel) { //do something }
```

## 4.3 LCD\_close\_flow

与 LCD\_open\_flow 对应的是 LCD\_close\_flow，它用于注册关屏函数。使用 LCD\_CLOSE\_FUNC 进行函数注册，先注册先执行。这里只是注册回调函数，不是立刻执行。

```
static s32 LCD_close_flow(u32 sel)
{
    LCD_CLOSE_FUNC(sel, LCD_bl_close, 0);

    LCD_CLOSE_FUNC(sel, sunxi_lcd_tcon_disable, 50);

    LCD_CLOSE_FUNC(sel, LCD_panel_exit, 100);

    LCD_CLOSE_FUNC(sel, LCD_power_off, 0);

    return 0;
}
```

1. **LCD\_bl\_close**：先关闭背光，这样整个关屏过程，用户不会看到闪烁的过程。
2. **sunxi\_lcd\_tcon\_disable**：关闭 TCON（即停止发送数据）再延迟 50ms；
3. **LCD\_panel\_exit**：执行关屏代码，再延迟 100ms；（不需要初始化的屏，可省掉此步骤）
4. **LCD\_power\_off**：最后关闭电源，再延迟 0ms。

如下图是典型关屏时序图。

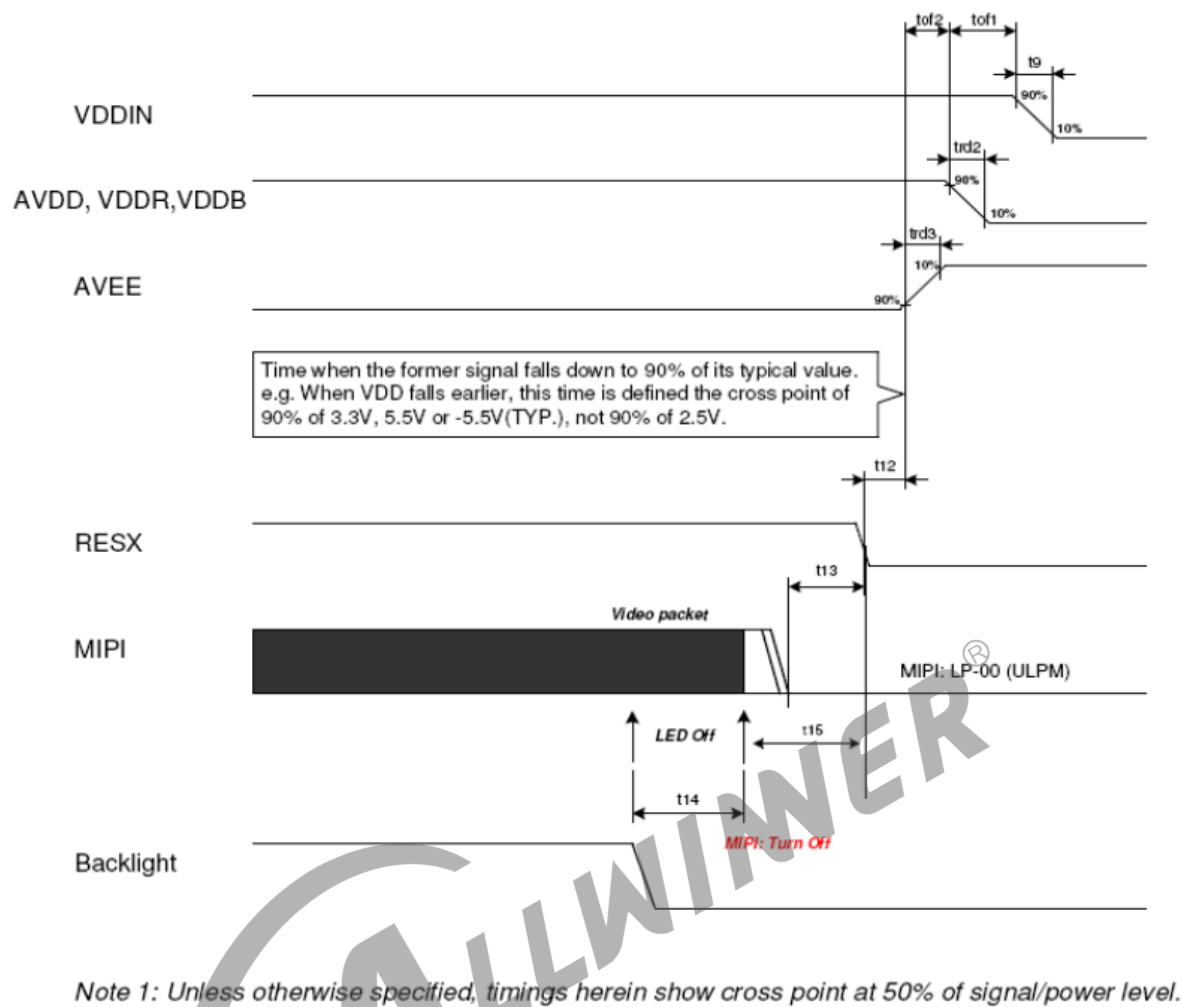


图 4-2: power off

## 4.4 LCD\_CLOSE\_FUNC

- 函数原型：void LCD\_CLOSE\_FUNC(u32 sel, LCD\_FUNC func, u32 delay);
- 作用：注册关屏步骤函数到关屏流程中，记住这里是注册不是执行！
- 参数：
- sel: lcd 节点索引。
- delay: 是执行该步骤后，再延迟的时间，时间单位是毫秒。
- func: 一个函数指针，其类型是：void (\*LCD\_FUNC) (\_\_u32 sel)，用户自己定义的函数必须也要用统一的形式。比如：

```
— c void user_defined_func(__u32 sel) { //do something }
```

## 4.5 sunxi\_lcd\_delay\_ms/us

- 原型：
- s32 sunxi\_lcd\_delay\_ms(u32 ms);
- s32 sunxi\_lcd\_delay\_us(u32 us)。
- 作用：延时函数，分别是毫秒级别/微秒级别的延时。
- 参数：
- ms/us：延时时间。

## 4.6 sunxi\_lcd\_tcon\_enable/disable

- 函数原型：
- void sunxi\_lcd\_tcon\_enable(u32 screen\_id);
- void sunxi\_lcd\_tcon\_disable(u32 screen\_id)。
- 作用：
- 打开 LCD 控制器，开始刷新 LCD 显示；
- 关闭 LCD 控制器，停止刷新数据。
- 参数：
- screen\_id：lcd 节点索引。

## 4.7 sunxi\_lcd\_backlight\_enable/disable

- 函数原型：
- void sunxi\_lcd\_backlight\_enable(u32 screen\_id);
- void sunxi\_lcd\_backlight\_disable(u32 screen\_id)。
- 作用：打开/关闭背光，操作的是 dts 中 lcd\_bl\_en 配置的 gpio。
- 参数：
- screen\_id：lcd 节点索引。

## 4.8 sunxi\_lcd\_pwm\_enable/disable

- 函数原型：

- s32 sunxi\_lcd\_pwm\_enable(u32 screen\_id);
- s32 sunxi\_lcd\_pwm\_disable(u32 screen\_id)。
- 作用：打开/关闭 pwm 控制器，打开时 pwm 将往外输出 pwm 波形。对应的是 lcd\_pwm\_ch 所对应的那一路 pwm
- 参数：
- screen\_id：lcd 节点索引。

## 4.9 sunxi\_lcd\_power\_enable/disable

- 函数原型：
- void sunxi\_lcd\_power\_enable(u32 screen\_id, u32 pwr\_id);
- void sunxi\_lcd\_power\_disable(u32 screen\_id, u32 pwr\_id)。
- 作用：打开/关闭 Lcd 电源，操作的是 board.dts 中的 lcd\_power/lcd\_power1/lcd\_power2。（pwr\_id 标识电源索引）
- 参数：
- screen\_id：lcd 节点索引。
- pwr\_id：标识电源索引：
  - 0：对应于 dts 中的 lcd\_power;
  - 1：对应于 dts 中的 lcd\_power1;
  - 2：对应于 dts 中的 lcd\_power2;
  - 3：对应于 dts 中的 lcd\_power3。

## 4.10 sunxi\_lcd\_pin\_cfg

- 函数原型：s32 sunxi\_lcd\_pin\_cfg(u32 screen\_id, u32 bon)。
- 作用：控制 lcd\_pin\_power 的电和 pinctrl-0 /pinctrl-1；若是 dsi 接口，会控制 dsi 的 dphy。
- 参数：
- 1: 为开；
- 0: 为配置成 disable 状态。

## 4.11 sunxi\_lcd\_gpio\_set\_value

- 函数原型：s32 sunxi\_lcd\_gpio\_set\_value(u32 screen\_id, u32 io\_index, u32 value);



- 作用：LCD\_GPIO PIN 脚上输出高电平或低电平
- 参数说明：
  - screen\_id：lcd 节点索引。
  - io\_index：标识电源索引：
    - 0：对应于 dts 中的 lcd\_gpio\_0；
    - 1：对应于 dts 中的 lcd\_gpio\_1；
    - 2：对应于 dts 中的 lcd\_gpio\_2；
    - 3：对应于 dts 中的 lcd\_gpio\_3。
- value：对应 IO 输出电平
  - 0：对应 IO 输出低电平
  - 1：对应 IO 输出高电平

## 4.12 sunxi\_lcd\_gpio\_set\_direction

- 函数原型：s32 sunxi\_lcd\_gpio\_set\_direction(u32 screen\_id, u32 io\_index, u32 direction);
- 作用：设置 LCD\_GPIO PIN 脚为输入或输出模式
- 参数说明：
  - screen\_id：lcd 节点索引。
  - io\_index：标识电源索引：
    - 0：对应于 dts 中的 lcd\_gpio\_0；
    - 1：对应于 dts 中的 lcd\_gpio\_1；
    - 2：对应于 dts 中的 lcd\_gpio\_2；
    - 3：对应于 dts 中的 lcd\_gpio\_3。
- value：对应 IO 模式
  - 0：对应 IO 设置为输入；
  - 1：对应 IO 设置为输出；

## 4.13 sunxi\_lcd\_dsi\_clk\_enable/disable

- 函数原型：
  - s32 sunxi\_lcd\_dsi\_clk\_enable(u32 scree\_id);
  - s32 sunxi\_lcd\_dsi\_clk\_disable(u32 scree\_id)。

- 作用：仅限 dsi 接口屏使用，使能/关闭 dsi 输出的高速时钟 clk 信号，必须在初始化的时候调用。
- 参数：
- screen\_id: lcd 节点索引。

## 4.14 sunxi\_lcd\_dsi\_dcs\_wr

- 函数原型：s32 sunxi\_lcd\_dsi\_dcs\_wr(u32 sel, u8 cmd, u8\* para\_p, u32 para\_num);
- 作用：对屏的 dcs 写操作
- 参数：
- cmd: dcs 写命令内容
- para\_p: dcs 写命令的参数起始地址
- para\_num: dcs 写命令的参数个数，单位为 byte

## 4.15 sunxi\_lcd\_dsi\_dcs\_wr\_2para

- 函数原型：s32 sunxi\_lcd\_dsi\_dcs\_wr\_2para(u32 sel, u8 cmd, u8 para1, u8 para2);
- 作用：对屏的 dcs 写操作，该命令带有两个参数
- 参数：
- cmd: dcs 写命令内容
- para1: dcs 写命令的第一个参数内容
- para2: dcs 写命令的第二个参数内容

sunxi\_dsi\_dcs\_wr\_0para, sunxi\_dsi\_dcs\_wr\_1para, sunxi\_dsi\_dcs\_wr\_3para, sunxi\_dsi\_dcs\_wr\_4para, sunxi\_dsi\_dcs\_wr\_5para 定义与 dsi\_dcs\_wr\_2para 类似，差别就是参数数量。

## 4.16 sunxi\_lcd\_dsi\_dcs\_read

- 函数原型：s32 sunxi\_lcd\_dsi\_dcs\_read(u32 sel, u8 cmd, u8 result, u32 num\_p)。
- 作用：dsi 读操作。
- 参数：
- sel: lcd 节点索引。

- cmd：要读取的寄存器
- result：用于存放读取接口的数组，用户必须自行保证其有足够空间保存读取的接口
- num\_p：指针用于存放读取字节数，用户必须保证其非空指针。

## 4.17 sunxi\_lcd\_dsi\_gen\_write

- 函数原型：s32 sunxi\_lcd\_dsi\_gen\_write(u32 screen\_id, u8 command, u8 \*para, u32 para\_num)
- 作用：对屏的 gen 写操作
- 参数：
  - cmd：gen 写命令内容
  - para\_p：gen 写命令的参数起始地址
  - para\_num：gen 写命令的参数个数，单位为 byte



## 5 DTS 适配步骤

### 5.1 屏驱动选择

Property	Value	Comment
lcd_driver_name	XXX	指定屏驱动，为屏驱动中 struct __lcd_panel 的 name 值；

### 5.2 选择接口及模式。

Property	Value	Comment
lcd_if	0, 3, 4	0: HV RGB 接口; 3: LVDS 接口 4: DSI 接口
lcd_dsi_if	0, 1, 2	0: Video mode; 1: Command mode; 2: video burst mode;
lcd_lvds_if	0, 1, 2	0: Single Link( 1 clock pair+3/4 data pair)1: Dual Link(8 data lane, 每 4 条 lane 接受一半像素, 奇数像素或者偶数像素)2: Dual Link (每 4 条 lane 接受全部像素, 常用于物理双屏, 且两个屏一样)
lcd_hv_if	0, 8	0: 并行 RGB 输出 8: 串行 RGB 输出

### 5.3 配置屏时序

若一款屏的时序如下

Parameter	Symbol	Min.	Typ.	Max.	Unit
MIPI data frequency	FDATA	955	999	1000	Mbps
Horizontal display area	THD	1200			pixel
HS period time	TH	1275	1341	1342	pixel
HS pulse width	THPW	1	1	1	pixel
HS back porch	THBP	32	60	60	pixel
HS front porch	THFP	42	80	81	pixel
Vertical display area	TVD	1920			H
VS period time	TV	1981	1981	1982	H
VS pulse width	TVPW	1	1	1	H
VS back porch	TVBP	25			H
VS front porch	TVFP	35	35	36	H

图 5-1: 参考屏时序

Property	Value	Comment
lcd_x	1200	显示屏的水平像素数量，也就是屏分辨率中的宽
lcd_y	1920	显示屏的垂直行数，也就是屏分辨率中的高
lcd_hspw	1	指行同步信号的宽度。单位为 1 个 dclk 的时间
lcd_hbp	61	指有效行间：行同步信号（hsync）开始，到有效数据开始之间的 dclk 的 cycle 个数，包括同步信号区； <b>lcd_hbp = 实际的 hbp + 实际的 hspw</b>
lcd_ht	1341	指一行总的 dclk 的 cycle 个数
lcd_vspw	1	指场同步信号的宽度；单位为行
lcd_vbp	26	指场同步信号（vsync）开始，到有效数据行开始之间的行数，包括场同步信号区； <b>lcd_vbp = 实际的 vbp + 实际的 vspw</b>
lcd_vt	1981	指一场的总行数
lcd_dclk_freq	165	传输像素传送频率（单位为 MHz）： <b>lcd_dclk_freq = lcd_ht * lcd_vt * fps</b>

## 5.4 背光配置

Property	Value	Comment
lcd_backlight	0~255	背光亮度默认值
lcd_pwm_used	0, 1	0：不使用 pwm 调节背光 1：使用 pwm 调节背光
lcd_pwm_ch		pwm 通道选择：通过查看原理图连接可知通道号
lcd_pwm_freq	50000	配置 PWM 信号的频率，单位为 Hz。
lcd_pwm_pol	0, 1	数配置 PWM 信号的占空比的极性 0：active high 1：active low；

Property	Value	Comment
lcd_bl_n_percent	0~100	背光映射值，n 为 (0-100)：此功能是针对亮度非线性的 LCD 屏的，按照配置的亮度曲线方式来调整亮度变化，以使亮度变化比如 lcd_bl_0_percent = 10，表明将 0% 的亮度值调整成 10%，即亮度比原来提高 10%

## lcd\_bl\_en

背光使能脚，非必须，看原理图是否有，用于使能或者禁止背光电路的电压。若背光使能 pin 是 PH18，dts 配置如下：

```
# uboot-board.dts 写法如下写：
lcd_bl_en = <&pio PH 18 1 0 3 1>;

# board.dts 写法如下：
lcd_bl_en = <&pio PH 18 GPIO_ACTIVE_HIGH>;
```

含义：PH18 输出高电平时打开 LCD 背光，下拉，默认高电平。

参数：

- 第一个数字：pin 序号；
- 第二个数字：功能分配：0 为输入，1 为输出；
- 第三个数字：内置电阻。
  - 0：标示内部电阻高阻态，
  - 1：则是内部电阻上拉，
  - 2：就代表内部电阻下拉。
  - default：代表默认状态，即电阻上拉。
- 第四个数字：驱动能力。3 表驱动能力是等级 3。
- 第五个数字：表示默认值；即是当设置为输出时，该引脚输出的电平：0 为低电平，1 为高电平。
- GPIO\_ACTIVE\_HIGH：代表配置成默认输出高电平；
- GPIO\_ACTIVE\_LOW：代表配置成默认输出低电平。

## 5.5 接口的详细设置。

Property	Value	Comment
lcd_dsi_lane	4	设置 dsi 输出 data lane 的数量；若是 dual-dsi，且总共输出 8 data lane，该值还是填 4。
lcd_dsi_port_num	0, 1	DSI 屏 port 数量（0：一个 port；1：两个 port）。
lcd_tcon_mode	0, 4	
lcd_tcon_en_odd_even_dir	0, 1	0: tcon 将一帧图像分左右两半来发送给两个 DSI 模块 1: tcon 将一帧图像分奇偶像素来发给两个 DSI 模块
lcd_lvds_colordepth	0, 1	0: 8bit per color(4 data pair) 1: 6bit per color(3 data pair)
lcd_lvds_mode	0, 1	0: NS mode 1: JEDIA mode
lcd_hv_clk_phase	0, 1, 2, 3	0: 0 degree 1: 90 degree 2: 180 degree 3: 270 degree
lcd_hv_sync_polarity	0, 1, 2, 3	0: vsync active low, hsync active low 1: vsync active high, hsync active low 2: vsync active low, hsync active high 3: vsync active high, hsync active high
lcd_hv_de_polarity	0, 1	0: de active low 1: de active high
lvds0_lane_np_inv	0~31	设置 lvds0 lane 的 NP 极性翻转，共 5 位，[bit0:bit3] 对应 data lane0~lane3, bit4 对应 CLK lane
lvds1_lane_np_inv	0~31	设置 lvds1 lane 的 NP 极性翻转，共 5 位，[bit0:bit3] 对应 data lane0~lane3, bit4 对应 CLK lane
lvds_even_odd_dir	0, 1	0: lvds dual-link even_odd order default 1: lvds dual-link even_odd order reverse

## 5.6 显示效果相关

Property	Value	Comment
lcd_frm	0, 1, 2	0: RGB888 -- RGB888 direct 1: RGB888 -- RGB666 dither 2: RGB888 -- RGB565 dither
lcd_rb_swap	1	1: 调换 tcon 模块 RGB 中的 R 分量和 B 分量
lcd_start_delay	0~10	mipi-dsi 屏前几行出现闪条时，可以调此参数

## 5.7 管脚和电源的配置

请根据电路图来配置，如果需要使用某路电源必须先在 [disp] 节点中定义，然后 [lcd] 部分使用的字符串则要和 disp 中定义的一致。比如下面的例子：

```
&disp {
    /* VCC-LCD */
    dc1sw-supply = <&reg_sw>;
```

```
/* VCC-LVDS and VCC-HDMI */  
bldo1-supply = <&reg_bldo1>;  
/* VCC-TV */  
cldo4-supply = <&reg_cldo4>;  
};
```

其中-supply 是固定的，在-supply 之前的字符串则是随意的，不过建议取有意义的名字。在 = 后面的像 <&reg\_sw> 则必须在 board.dts 的 regulator0 节点中找到。然后 lcd 节点中，如果要使用 reg\_sw，则像下面这样写就行，dc1sw 对应 dc1sw-supply。

```
lcd_power = "dc1sw";
```

由于 u-boot 中也有 axp 驱动和 display 驱动，和内核，它们都是读取同份配置，为了能互相兼容，取名时，有以下限制：

在 u-boot 2018 中，axp 驱动只认类似 bldo1 这样从 axp 芯片中定义的名字，所以命名 xxx-supply 的时候最好按照这个 axp 芯片的定义来命名。

### 5.7.1 lcd\_power

见上面概述的注意事项。该属性用来配置屏端所需要的电

```
示例：lcd_power = "vcc-lcd";
```

配置 regulator 的名字。配置好之后，需要在屏驱动调用相应的接口进行开、关的控制。

注意：如果有多个电源需要打开，则定义 lcd\_power1，lcd\_power2 等。

### 5.7.2 lcd\_pin\_power

用法 lcd\_power 一致，区别是用户设置之后，不需要在屏驱动中去操作，而是驱动框架自行在屏驱动之前使能，在屏驱动之后禁止。该属性用来配置 IO、phy 的电，屏端供电建议不要用该属性，可能会造成上电时序异常导致屏无法点亮的情况。

```
示例：lcd_pin_power = "vcc-pd";
```

注意：如果需要多组，则添加 lcd\_pin\_power1，lcd\_pin\_power2 等。除了 lcddx 之外，这里的电源还有可能是 pwm 所对应管脚的电源。

### 5.7.3 lcd\_gpio\_x

```
# u-boot-board.dts 写法如下：  
lcd_gpio_0 = <&pio PD 25 1 0 3 1>;  
  
# board.dts 写法如下：  
lcd_gpio_0 = <&pio PD 25 GPIO_ACTIVE_HIGH>;
```



含义：lcd\_gpio\_0 引脚为 PD25。

- 第一个数字：pin 序号；
- 第二个数字：功能分配：0 为输入，1 为输出；
- 第三个数字：内置电阻。
  - 0：标示内部电阻高阻态，
  - 1：则是内部电阻上拉，
  - 2：就代表内部电阻下拉。
  - default：代表默认状态，即电阻上拉。
- 第四个数字：驱动能力。3 表驱动能力是等级 3。
- 第五个数字：表示默认值；即是当设置为输出时，该引脚输出的电平：0 为低电平，1 为高电平。
- GPIO\_ACTIVE\_HIGH：代表配置成默认输出高电平；
- GPIO\_ACTIVE\_LOW：代表配置成默认输出低电平。

注意：如果有多个 gpio 脚需要控制，则定义 lcd\_gpio\_0, lcd\_gpio\_1 等。

#### 5.7.4 pinctrl-0 和 pinctrl-1

在配置 lcd 节点时，当碰到需要配置管脚复用时，你只要把 pinctrl-0 和 pinctrl-1 赋值好就行，用提前定义好的。例子：

```
pinctrl-0 = <&dsi0_4lane_pins_a>;
pinctrl-1 = <&dsi0_4lane_pins_b>;
```

表 5-7: 提前定义好的管脚名称

接口	管脚名称	描述
rgb	rgb24_pins_a 和 rgb24_pins_b	RGB 屏接口，而且数据位宽是 24，RGB888
rgb	rgb18_pins_a 和 rgb18_pins_b	RGB 屏接口，而且数据位宽是 16，RGB666
lvds0	lvds0_pins_a 和 lvds0_pins_b	Single link LVDS0 接口管脚定义 (lcd0)
lvds1	lvds1_pins_a 和 lvds1_pins_b	Single link LVDS1 接口管脚定义 (lcd0)
dual-lvds0+1	lvds0_pins_a、lvds1_pins_a 和 lvds0_pins_b、lvds1_pins_b	Dual link LVDS0+1 接口管脚定义 (lcd0)
dsi0	dsi0_4lane_pins_a 和 dsi0_4lane_pins_b	4 lane DSI0 屏接口管脚定义 (lcd0)
dsi1	dsi1_4lane_pins_a 和 dsi1_4lane_pins_b	4 lane DSI1 屏接口管脚定义 (lcd1)

接口	管脚名称	描述
dual-dsi0+1	dsi0_4lane_pins_a、dsi1_4lane_pins_a 和 dsi0_4lane_pins_b、dsi1_4lane_pins_b	Dual link DSI 接口管脚定义 (lcd0)
lvds2	lvds2_pins_a 和 lvds2_pins_b	Single link LVDS2 接口管脚定义 (lcd2)
lvds3	lvds3_pins_a 和 lvds3_pins_b	Single link LVDS3 接口管脚定义 (lcd2)
dual-lvds2+3	lvds2_pins_a、lvds3_pins_a 和 lvds2_pins_b、lvds3_pins_b	Dual link LVDS2+3 接口管脚定义 (lcd2)

为了规范，我们将在所有平台保持一致的名字，其中后缀为 a 为管脚使能，b 的为 io\_disable 用于设备关闭时。

有时候，你需要用两组不同功能的管脚，可以像下面这样定义即可。

```
pinctrl-0 = <&rgb24_pins_a>, <&xxx_pins_a>;  
pinctrl-1 = <&rgb24_pins_b>, <&xxx_pins_b>;
```

## 6 MIPI-DSI

### 6.1 LCD 节点接口类型

#### 1. Single-dsi

(1) 使用 dsi0 接口输出时，需要注意的主要配置如下：



图 6-1: dsi0 输出示意图

```
&lcd0 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <4>;  
};
```

(2) 使用 dsi1 接口输出时，需要注意的主要配置如下：

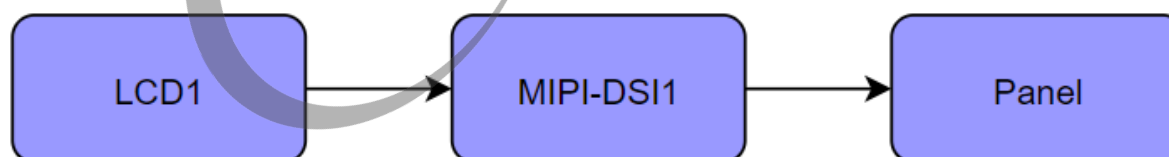


图 6-2: dsi1 输出示意图

```
&lcd1 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <4>;  
};
```

#### 2. Dual-dsi

(1) 屏接口是 dsi0 接收奇像素，dsi1 接收奇偶像素时，需要注意的主要配置如下：

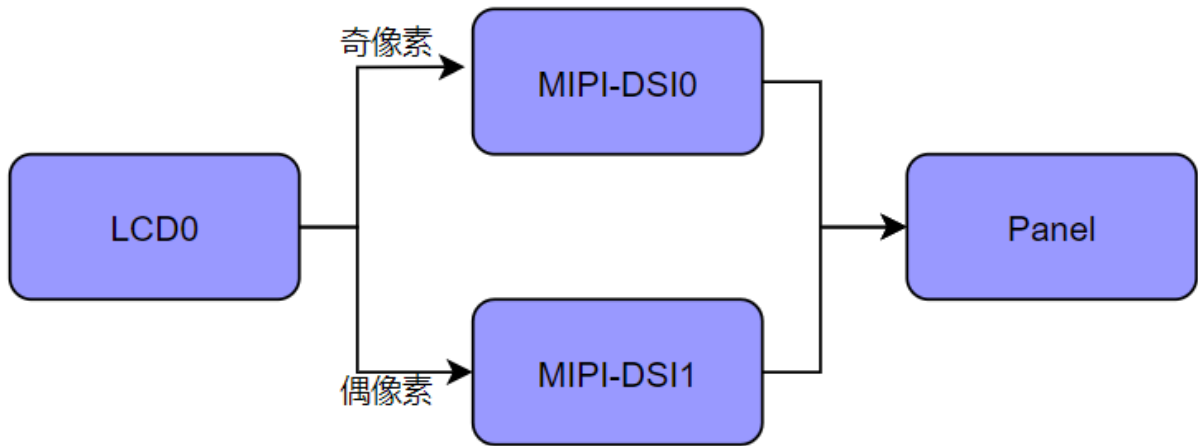


图 6-3: dual-dsi 输出示意图

```

&lcd0 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <4>;

    lcd_dsi_port_num = <1>;
    lcd_tcon_mode = <4>;

    lcd_tcon_en_odd_even_div = <0>;
};
  
```

(2) 屏接口是 dsi0 接收左半图，dsi1 接收右半图时，需要注意的主要配置如下：

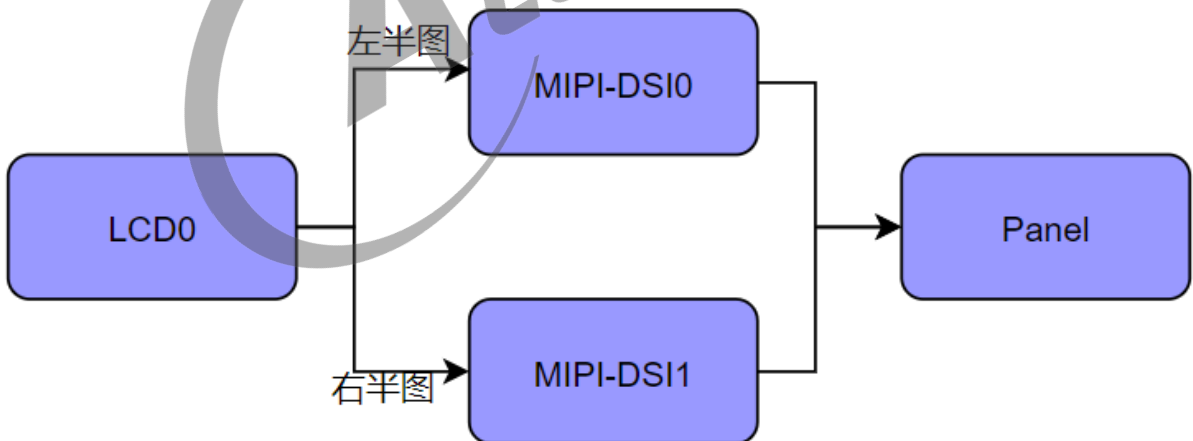


图 6-4: dual-dsi 输出示意图-1

```

&lcd0 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <4>;
    lcd_dsi_port_num = <1>;
    lcd_tcon_mode = <4>;
  }
  
```

```
lcd_tcon_en_odd_even_div = <1>;  
};
```

## 6.2 DTS 适配

对参数不了解的，请看[DTS 适配步骤](#)中的对应小节。

### 6.2.1 single-mipi-dsi0

```
&lcd0 {  
    /* 屏驱动选择 */  
    lcd_used      = <1>;  
    status        = "okay";  
    lcd_driver_name = "SQ101D_Q5DI404_84H501";  
  
    /* 选择接口及模式 */  
    lcd_if        = <4>;  
    lcd_dsi_if    = <0>;  
  
    /* 配置屏时序 */  
    lcd_x         = <1200>;  
    lcd_y         = <1920>;  
    lcd_dclk_freq = <157>;  
    lcd_hbp       = <50>;  
    lcd_ht        = <1330>;  
    lcd_hspw      = <10>;  
    lcd_vbp       = <20>;  
    lcd_vt        = <1960>;  
    lcd_vspw      = <4>;  
  
    /* 背光配置 */  
    lcd_backlight = <50>;  
    lcd_pwm_used  = <1>;  
    lcd_pwm_ch    = <0>;  
    lcd_pwm_freq  = <50000>;  
    lcd_pwm_pol   = <0>;  
    lcd_bl_en     = <&pio PH 18 1 0 3 1>;  
    lcd_bl_0_percent = <5>;  
  
    /* 接口详细设置 */  
    lcd_dsi_lane  = <4>;  
  
    lcd_start_delay = <5>; //显示前几行花屏，调此参数  
  
    /* I/O管脚和电源配置 */  
    lcd_power1 = "cldo4";  
    lcd_power2 = "cldo1";  
    lcd_gpio_0 = <&pio PD 22 1 0 3 1>;  
    pinctrl-0 = <&dsi0_4lane_pins_a>;  
    pinctrl-1 = <&dsi0_4lane_pins_b>;  
};
```

## 6.2.2 single-mipi-dsi1

```
&lcd1 {
    /* 屏驱动选择 */
    lcd_used      = <1>;
    status        = "okay";
    lcd_driver_name = "SQ101D_Q5DI404_84H501";

    /* 选择接口及模式 */
    lcd_if        = <4>;

    /* 配置屏时序 */
    lcd_x         = <1200>;
    lcd_y         = <1920>;
    lcd_dclk_freq = <157>;
    lcd_hbp       = <50>;
    lcd_ht        = <1330>;
    lcd_hspw      = <10>;
    lcd_vbp       = <20>;
    lcd_vt        = <1960>;
    lcd_vspw      = <4>;

    /* 背光配置 */
    lcd_backlight = <50>;
    lcd_pwm_used  = <1>;
    lcd_pwm_ch    = <0>;
    lcd_pwm_freq  = <50000>;
    lcd_pwm_pol   = <0>;
    lcd_bl_en     = <&pio PH 18 1 0 3 1>;
    lcd_bl_0_percent = <5>;

    /* 接口详细设置 */
    lcd_dsi_lane  = <4>;

    lcd_start_delay = <5>; //显示前几行花屏，调此参数

    /* I/O管脚和电源配置 */
    lcd_power1 = "cldo4";
    lcd_power2 = "cldo1";
    lcd_gpio_2 = <&pio PD 22 1 0 3 1>;
    pinctrl-0 = <&dsi1_4lane_pins_a>;
    pinctrl-1 = <&dsi1_4lane_pins_b>;
};
```

## 6.2.3 dual-mipi-dsi

分辨率达到 2k 以上的屏，实际上需要多达 8 条数据 lane 才能正常显示，其中四条 lane 发送一副图像中的奇像素，另外一副图像发送偶像素。配置 dts 时需要注意，dual-mipi-dsi 需要配置的是 lcd0 节点，

### 📖 说明

注意只有部分 IC 支持超高分辨率，具体查看芯片规格中的 MIPI-DSI 部分

```
&lcd0 {
    /* 屏驱动选择 */
    lcd_used      = <1>;
    status        = "okay";
    lcd_driver_name = "lq101r1sx03";

    /* 选择接口及模式 */
    lcd_if        = <4>;

    /* 配置屏时序 */
    lcd_x          = <2560>;
    lcd_y          = <1600>;
    lcd_dclk_freq  = <268>;
    lcd_hbp        = <80>;
    lcd_ht         = <2720>;
    lcd_hspw       = <32>;
    lcd_vbp        = <37>;
    lcd_vt         = <1646>;
    lcd_vspw       = <6>;

    /* 背光配置 */
    lcd_backlight  = <50>;
    lcd_pwm_used   = <1>;
    lcd_pwm_ch     = <0>;
    lcd_pwm_freq   = <50000>;
    lcd_pwm_pol    = <0>;
    lcd_bl_en      = <&pio PH 10 1 0 3 1>;

    /* 接口详细设置 */
    lcd_dsi_lane   = <4>;
    lcd_dsi_port_num = <1>;
    lcd_tcon_mode  = <4>;
    lcd_tcon_en_odd_even_div = <1>;

    /* I/O管脚和电源配置 */
    lcd_power      = "vcc18-lcd";
    lcd_power1     = "vcc33-lcd";
    lcd_pin_power  = "vcc-pd";
    lcd_gpio_0     = <&pio PH 11 1 0 3 1>;
    lcd_gpio_1     = <&pio PH 12 1 0 3 1>;
    pinctrl-0     = <&dsi0_4lane_pins_a>, <&dsi1_4lane_pins_a>;
    pinctrl-1     = <&dsi0_4lane_pins_b>, <&dsi1_4lane_pins_b>;
};
```

## 6.3 驱动移植

参考屏驱动：

uboot-2018：

brandy/brandy-2.0/u-boot-2018/drivers/video/sunxi/disp2/disp/lcd/  
SQ101D\_Q5DI404\_84H501H.c

kernel：

bsp/drivers/video/sunxi/disp2/disp/lcd/SQ101D\_Q5DI404\_84H501H.c

在如下几个地方修改后，即可满足当前屏的驱动需要（驱动中的函数解析请看[屏驱动接口](#)）：

- 根据屏规格书的上下电要求，完成 LCD\_power\_on、LCD\_power\_off 函数的定义；
- 若屏初始化指令都是 **dcs** 类型，将指令写在 lcm\_initialization\_setting[] 中即可；若指令中有 **generic** 类型，则应该在 LCD\_panel\_init 函数中调 **sunxi\_lcd\_dsi\_gen\_write**、**sunxi\_lcd\_dsi\_dcs\_write** 接口完成初始化设置（对于 **dual-dsi** 屏，两个 dsi 都要对屏进行初始化，则每个指令都需要调两次接口，第二次调时记得 sel + 1，参考驱动：drivers/video/sunxi/disp2/disp/lcd/lq079l1sx01.c）
- 若需要增加或减少屏的休眠唤醒耗时，可修改 LCD\_open\_flow、LCD\_close\_flow 中的延时，在测试没问题的前提下，到达自己需求。

完成如上修改后，若需要单独添加一个驱动，还需要的修改如下：

- 在屏驱动中修改 struct \_\_lcd\_panel 的变量，其中.name 不能跟其它屏驱动的.name 重复，且需要与 dts 中 lcd\_driver\_name 的值一样；
- 修改屏驱动目录下的 panel.c 和 panel.h，在全局结构体变量 panel\_array 中新增刚才添加 \*\*struct \_\_lcd\_panel\*\* 的变量指针。panel.h 中新增 \*\*struct \_\_lcd\_panel\*\* 的声明。
- 修改 Makefile，在 lcd 屏驱动目录的上级的 Makefile 文件中的 disp-objs 中新增刚才添加屏驱动.o



## 7 LVDS

### 7.1 LCD 节点接口类型

#### 7.1.1 Single-lvds

(1) 使用 lvds0 接口输出时，需要注意的主要配置如下：



图 7-1: lvds0 输出示意图

```
&lcd0 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <3>;  
  
    pinctrl-0 = <&lvds0_pins_a>;  
    pinctrl-1 = <&lvds0_pins_b>;  
};
```

(2) 使用 lvds1 接口输出时，需要注意的主要配置如下：



图 7-2: lvds1 输出示意图

```
&lcd0 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <3>;  
    lcd_lvds_if = <2>;  
  
    pinctrl-0 = <&lvds1_pins_a>;  
    pinctrl-1 = <&lvds1_pins_b>;  
};
```

(3) 使用 lvds2 接口输出时，需要注意的主要配置如下：



图 7-3: lvds2 输出示意图

```
&lcd2 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <3>;  
  
    pinctrl-0 = <&lvds2_pins_a>;  
    pinctrl-1 = <&lvds2_pins_b>;  
};
```

(4) 使用 lvds3 接口输出时，需要注意的主要配置如下：



图 7-4: lvds3 输出示意图

```
&lcd2 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <3>;  
    lcd_lvds_if = <2>;  
  
    pinctrl-0 = <&lvds3_pins_a>;  
    pinctrl-1 = <&lvds3_pins_b>;  
};
```

## 7.1.2 dual-lvds

(1) 使用 dual-lvds0+1 接口输出时，需要注意的主要配置如下：

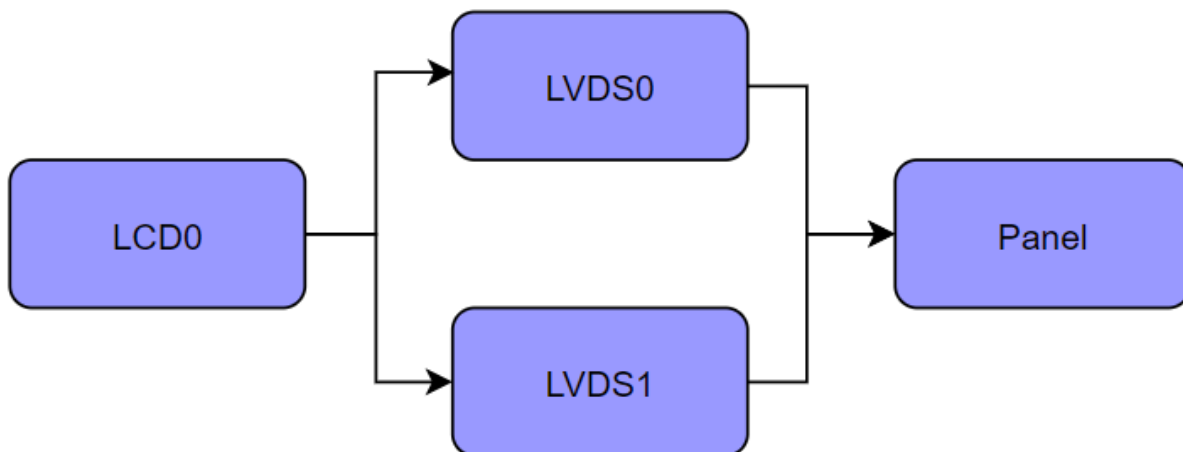


图 7-5: dual-lvds0+1 输出示意图

```
&lcd0 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <3>;
    lcd_lvds_if = <1>;

    pinctrl-0 = <&lvds0_pins_a>, <&lvds1_pins_a>;
    pinctrl-1 = <&lvds0_pins_b>, <&lvds1_pins_b>;
};
```

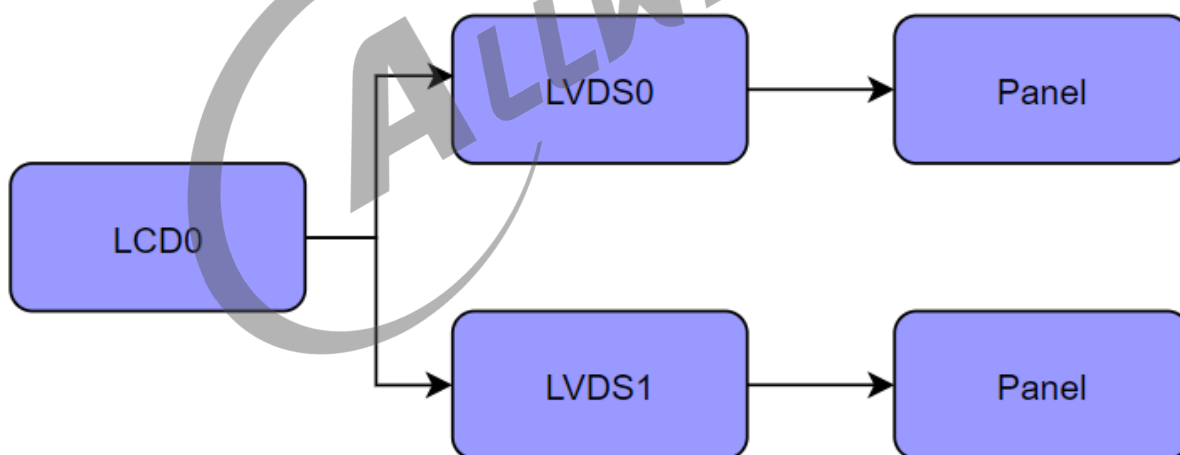


图 7-6: dual-lvds0+1 输出示意图-1

```
&lcd0 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <3>;
    lcd_lvds_if = <1>;

    pinctrl-0 = <&lvds0_pins_a>, <&lvds1_pins_a>;
    pinctrl-1 = <&lvds0_pins_b>, <&lvds1_pins_b>;
};
```

(2) 使用 dual-lvds2+3 接口输出时，需要注意的主要配置如下：

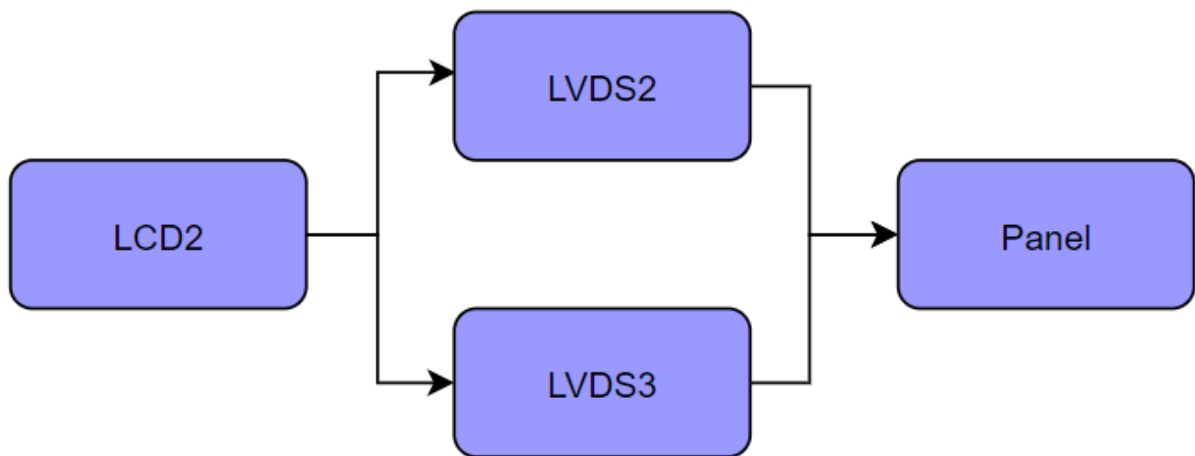


图 7-7: dual-lvds2+3 输出示意图

```

&lcd2 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <3>;
    lcd_lvds_if = <1>;

    pinctrl-0 = <&lvds2_pins_a>, <&lvds3_pins_a>;
    pinctrl-1 = <&lvds2_pins_b>, <&lvds3_pins_b>;
};
  
```

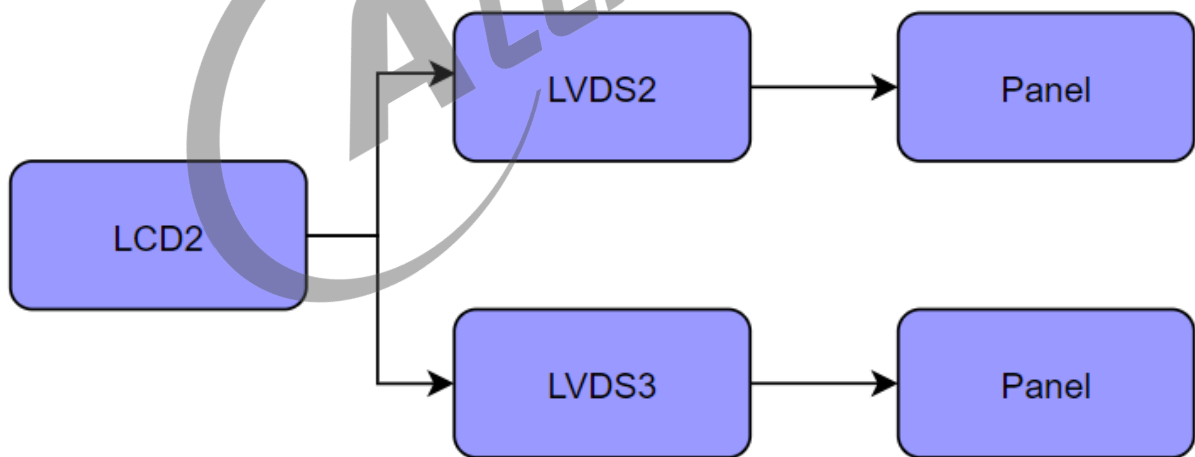


图 7-8: dual-lvds2+3 输出示意图-1

```

&lcd2 {
    lcd_used = <1>;
    status = "okay";

    lcd_if = <3>;
    lcd_lvds_if = <1>;

    pinctrl-0 = <&lvds2_pins_a>, <&lvds3_pins_a>;
    pinctrl-1 = <&lvds2_pins_b>, <&lvds3_pins_b>;
};
  
```

```
};
```

## 7.2 DTS 适配

由于 lvds 协议不具备传输数据之外的能力，一般屏端不需要任何初始化，只需要初始化 SoC 端即可。所以这里的 lcd\_driver\_name 依旧是 "default\_lcd"，当然你可以为初始化的启动延时做专门的优化。对参数不了解的，请看 [DTS 适配步骤](#) 中的对应小节。

### 7.2.1 LVDS Single link

#### 1) lvds0 配置

- 在 lcd0 节点适配；
- lcd\_if 为 3 代表 lvds 输出；
- 查看原理图，配置 I/O、PHY、屏供电，屏的复位及使能 I/O、背光 pwm 通道。
- 查看屏规格书，获取屏时序。

```
&lcd0 {
    /* 屏驱动选择 */
    lcd_used      = <1>;
    status        = "okay";
    lcd_driver_name = "default_lcd";

    /* 选择接口及模式 */
    lcd_if        = <3>;
    lcd_lvds_if   = <0>;

    /* 配置屏时序 */
    lcd_x         = <1280>;
    lcd_y         = <800>;
    lcd_dclk_freq = <70>;
    lcd_hbp       = <20>; //lcd_hbp = hbp + hspw
    lcd_ht        = <1418>;
    lcd_hspw      = <10>;
    lcd_vbp       = <10>; //lcd_vbp = vbp + vspw
    lcd_vt        = <814>;
    lcd_vspw      = <5>;

    /* 背光配置 */
    lcd_backlight = <50>;
    lcd_pwm_used  = <1>;
    lcd_pwm_ch    = <0>;
    lcd_pwm_freq  = <50000>;
    lcd_pwm_pol   = <0>;
    lcd_backlight = <50>;
    lcd_bl_en     = <&pio PD 21 1 0 3 1>;

    /* 接口详细设置 */
    lcd_lvds_colordepth = <0>;
}
```

```
lcd_lvds_mode    = <0>;

/* 显示效果相关 */
lcd_frm          = <0>;

/* I/O管脚和电源配置 */
lcd_power        = "vcc-lcd";
pinctrl-0 = <&lvds0_pins_a>;
pinctrl-1 = <&lvds0_pins_b>;
};
```

## 2) lvds1 配置

lvds1 与 lvds0 的主要区别是：

- lcd\_lvds\_if 的值为 2；
- pinctrl-0 = <&lvds1\_pins\_a>; pinctrl-1 = <&lvds1\_pins\_b>;

## 3) lvds2 配置

lvds2 与 lvds0 的主要区别是：

- 配置放在 lcd2 节点；
- pinctrl-0 = <&lvds2\_pins\_a>; pinctrl-1 = <&lvds2\_pins\_b>;

## 4) lvds3 配置

lvds3 与 lvds0 的主要区别是：

- 配置放在 lcd2 节点；
- lcd\_lvds\_if 的值为 2；
- pinctrl-0 = <&lvds3\_pins\_a>; pinctrl-1 = <&lvds3\_pins\_b>;

## 7.2.2 LVDS Dual link

1、物理上连接一个屏，8 data lane，SoC 向每 4 条 lane 传输一半的像素，奇数像素或者偶数像素。

### 1) dual\_lvds0+lvds1

```
&lcd0 {
/* 屏驱动选择 */
lcd_used    = <1>;
status      = "okay";
lcd_driver_name = "default_lcd";
lcd_backlight = <50>;
};
```

```

/* 选择接口及模式 */
lcd_if      = <3>;
lcd_lvds_if  = <1>;

/* 配置屏时序 */
lcd_x       = <1920>;
lcd_y       = <1200>;
lcd_dclk_freq = <150>;
lcd_hbp     = <80>;
lcd_ht      = <2064>;
lcd_hspw    = <16>;
lcd_vbp     = <7>;
lcd_vt      = <1211>;
lcd_vspw    = <3>;

/* 背光配置 */
lcd_backlight = <50>;
lcd_pwm_used  = <1>;
lcd_pwm_ch    = <2>;
lcd_pwm_freq  = <50000>;
lcd_pwm_pol   = <0>;
lcd_bl_en     = <&pio PJ 27 1 0 3 1>;

/* 接口详细设置 */
lcd_lvds_colordepth = <0>;
lcd_lvds_mode       = <0>;

/* 显示效果相关 */
lcd_frm             = <0>;

/* I/O管脚和电源配置 */
lcd_gpio_0          = <&pio PI 1 1 0 3 1>;
lcd_pin_power       = "bl05";
lcd_power           = "dc1sw";
pinctrl-0 = <&lvds0_pins_a>, <&lvds1_pins_a>;
pinctrl-1 = <&lvds0_pins_b>, <&lvds1_pins_b>;
};

```

## 2) dual\_lvds2+lvds3

dual\_lvds2+lvds3 与 dual\_lvds0+lvds1 的主要区别是：

- 配置放在 lcd2 节点；
- pinctrl-0 = <&lvds2\_pins\_a>, <&lvds3\_pins\_a>; pinctrl-1 = <&lvds2\_pins\_b>, <&lvds3\_pins\_b>;

2、物理上连接两个屏，每个屏各自 4 条 lane，两个屏是一样型号，分辨率和 timing 一样，IC 支持将全部像素发到每个屏上，实现双显（信号上的双显），

## 3) lvds0+lvds1

```

&lcd0 {
/* 屏驱动选择 */
lcd_used      = <1>;
status        = "okay";
lcd_driver_name = "default_lcd";
};

```

```

/* 选择接口及模式 */
lcd_if      = <3>;
lcd_lvds_if  = <0>;

/* 配置屏时序 */
lcd_x       = <1280>;
lcd_y       = <800>;
lcd_dclk_freq = <70>;
lcd_hbp     = <20>; //lcd_hbp = hbp + hspw
lcd_ht      = <1418>;
lcd_hspw    = <10>;
lcd_vbp     = <10>; //lcd_vbp = vbp + vspw
lcd_vt      = <814>;
lcd_vspw    = <5>;

/* 背光配置 */
lcd_backlight = <50>;
lcd_pwm_used  = <1>;
lcd_pwm_ch    = <0>;
lcd_pwm_freq  = <50000>;
lcd_pwm_pol   = <0>;
lcd_backlight = <50>;
lcd_bl_en     = <&pio PD 21 1 0 3 1>;

/* 接口详细设置 */
lcd_lvds_colordepth = <0>;
lcd_lvds_mode       = <0>;

/* 显示效果相关 */
lcd_frm             = <0>;

/* I/O管脚和电源配置 */
lcd_power           = "vcc-lcd";
pinctrl-0 = <&lvds0_pins_a>, <&lvds1_pins_a>;
pinctrl-1 = <&lvds0_pins_a>, <&lvds1_pins_a>;
};

```

#### 4) lvds2+lvds3

lvds2+lvds3 与 lvds0+lvds1 的主要区别是：

- 配置放在 lcd2 节点；
- lcd\_lvds\_if 的值为 2；
- pinctrl-0 = <&lvds2\_pins\_a>, <&lvds3\_pins\_a>; pinctrl-1 = <&lvds2\_pins\_b>, <&lvds3\_pins\_b>;

## 7.3 驱动移植

参考屏驱动：

uboot-2018:



brandy/brandy-2.0/u-boot-2018/drivers/video/sunxi/disp2/disp/lcd/default\_panel.c

kernel:

bsp/drivers/video/sunxi/disp2/disp/lcd/default\_panel.c

在如下几个地方修改后，即可满足当前屏的驱动需要（驱动中的函数解析请看[屏驱动接口](#)）：

- 根据屏规格书的上下电要求，完成 LCD\_power\_on、LCD\_power\_off 函数的定义；
- 若需要增加或减少屏的休眠唤醒耗时，可修改 LCD\_open\_flow、LCD\_close\_flow 中的延时，在测试没问题的前提下，到达自己需求。

完成如上修改后，若需要单独添加一个驱动，还需要的修改如下：

- 在屏驱动中修改 struct \_\_lcd\_panel 的变量，其中.name 不能跟其它屏驱动的.name 重复，且需要与 dts 中 lcd\_driver\_name 的值一样；
- 修改屏驱动目录下的 panel.c 和 panel.h，在全局结构体变量 panel\_array 中新增刚才添加 \*\*struct \_\_lcd\_panel\*\* 的变量指针。panel.h 中新增 \*\*struct \_\_lcd\_panel\*\* 的声明。
- 修改 Makefile，在 lcd 屏驱动目录的上级的 Makefile 文件中的 disp-objs 中新增刚才添加屏驱动.o

## 8 RGB

RGB 接口在全志平台又称 HV 接口（Horizontal 同步和 Vertical 同步）。

有些 LCD 屏支持高级的功能比如 gamma，像素格式的设置等，但是 RGB 协议本身不支持图像数据之外的传输，无法通过 RGB 管脚对 LCD 屏进行配置，所以拿到一款 RGB 接口屏，要么不需要初始化命令，要么这个屏会提供额外的管脚给 SoC 来进行配置，比如 SPI 和 I2C 等。

### 8.1 LCD 节点接口类型

1. 使用 RGB0 接口输出时，需要注意的主要配置如下：

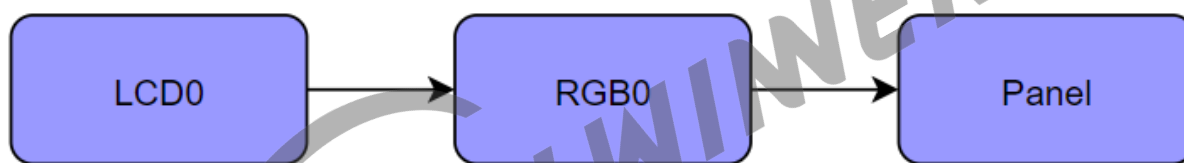


图 8-1: rgb0 输出示意图

```
&lcd0 {  
    lcd_used = <1>;  
    status = "okay";  
  
    lcd_if = <0>;  
};
```

2. 使用 RGB1 接口输出时，需要注意的主要配置如下：



图 8-2: rgb1 输出示意图

```
&lcd2 {  
    lcd_used = <1>;  
    status = "okay";  
};
```

```
lcd_if = <0>;  
};
```

## 8.2 DTS 适配

时钟周期数：一个像素发送完毕需要的时钟周期数。

当时钟周期为 1 时，我们称这种 RGB 接口为并行接口，其它的情况则是串行接口，更为普遍的原则就是要需要多个时钟周期才能发送完一个像素的接口都是串行接口。

如何判断是否支持 24bit 的位宽，最简单的方式就是看原理图上数据脚的数量，如果有 24 根则支持 24bit，如果只有 18 根则支持 18bit。对于并行 RGB 的接口，当位宽小于 24 时，硬件连接应该选择连接每个分量中的高位而放弃低位，这样做的原因是损失较少的颜色数量。

RGB 接口有两种同步方式，根据经验来说尽量使用第二种方式，硬件上请保证连接好 DE 脚。

1. Hsync+Vsync
2. DE (Data Enable)

### 8.2.1 并行 RGB0 接口

当我们配置并行 RGB 接口时，在配置里面并不需要区分是 24 位，18 位和 16 位，最大位宽是哪一种是参考 pin mux 表格，如果 LCD 屏本身支持的位宽比 SoC 支持的位宽少，当然只能选择少的一方。

```
&lcd0 {  
    /* 屏驱动选择 */  
    lcd_used      = <1>;  
    status        = "okay";  
    lcd_driver_name = "default_lcd";  
  
    /* 选择接口及模式 */  
    lcd_if        = <0>;  
    lcd_hv_if     = <0>;  
  
    /* 配置屏时序 */  
    lcd_x         = <800>;  
    lcd_y         = <480>;  
    lcd_dclk_freq = <33>;  
    lcd_hbp       = <46>;  
    lcd_ht        = <1055>;  
    lcd_hspw      = <0>;  
    lcd_vbp       = <23>;  
    lcd_vt        = <525>;  
    lcd_vspw      = <0>;  
  
    /* 背光配置 */  
    lcd_backlight = <50>;  
    lcd_pwm_used  = <1>;  
}
```

```

lcd_pwm_ch    = <8>;
lcd_pwm_freq  = <50000>;
lcd_pwm_pol   = <0>;
lcd_bl_en     = <&pio PD 27 GPIO_ACTIVE_HIGH>;

/* 接口详细设置 */
lcd_frm       = <0>;
lcd_hv_clk_phase = <0>;
lcd_hv_sync_polarity = <0>;

/* 显示效果相关 */
lcd_frm       = <0>;

/* I/O管脚和电源配置 */
lcd_power     = "vcc-lcd";
lcd_pin_power = "vcc-pd";
pinctrl-0 = <&rgb24_pins_a>;
pinctrl-1 = <&rgb24_pins_b>;
};

```

## 8.2.2 串行 RGB0 接口

串行 RGB 是相对于并行 RGB 来说，而并不是说它只用一根线来发数据，只要通过多个时钟周期才能把一个像素的数据发完，那么这样的 RGB 接口就是串行 RGB。

同样与并行 RGB 接口一样，配置中并不需要也无法体现具体是哪种串行 RGB 接口，你要做的就是将硬件连接对就行。

### 💡 技巧

这里需要注意的是，对于该接口，SoC 总共需要三个周期才能发完一个 pixel，所以我们配置时序的时候，需要满足  $\text{lcd\_dclk\_freq} * 3 = \text{lcd\_ht} * \text{lcd\_vt} * 60$ ，或者  $\text{lcd\_dclk\_freq} = \text{lcd\_ht} * 3 * \text{lcd\_vt} * 60$ ，要么 3 倍 lcd\_ht 要么 3 倍 lcd\_dclk\_freq。

### 📖 说明

下面实例的 lcd driver IC 是 stv7789v，是需要初始化，初始化的接口协议是 SPI，所以这多了几根 spi 管脚配置，驱动里面用 gpio 模拟 spi 协议，所以这里都是配置 gpio 功能。

```

&lcd0 {
    /* 屏驱动选择 */
    lcd_used    = <1>;
    status      = "okay";
    lcd_driver_name = "stv7789v";

    /* 选择接口及模式 */
    lcd_if      = <0>;
    lcd_hv_if   = <8>;

    /* 配置屏时序 */
    lcd_x       = <240>;
    lcd_y       = <320>;
    lcd_dclk_freq = <19>;
    lcd_hbp     = <120>;
    ;10 + 20 + 10 + 240*3 = 760 real set 1000
    lcd_ht      = <850>;
    lcd_hspw    = <2>;
}

```

```
lcd_vbp      = <13>;
lcd_vt       = <373>;
lcd_vspw     = <2>;

/* 背光配置 */
lcd_backlight = <50>;
lcd_pwm_used  = <1>;
lcd_pwm_ch    = <8>;
lcd_pwm_freq  = <50000>;
lcd_pwm_pol   = <0>;
lcd_bl_en     = <&pio PB 1 GPIO_ACTIVE_HIGH>;

/* 接口详细设置 */
lcd_hv_clk_phase = <0>;
lcd_hv_sync_polarity = <0>;

/* 显示效果相关 */
lcd_frm       = <1>;

/* I/O管脚和电源配置 */
lcd_power     = "vcc-lcd";
lcd_pin_power = "vcc-pd";
lcd_gpio_0    = <&pio PD 9 GPIO_ACTIVE_HIGH>; //reset
lcd_gpio_1    = <&pio PD 10 GPIO_ACTIVE_HIGH>; //cs
lcd_gpio_2    = <&pio PD 13 GPIO_ACTIVE_HIGH>; //sda
lcd_gpio_3    = <&pio PD 12 GPIO_ACTIVE_HIGH>; //sck
pinctrl-0     = <&rgb18_pins_a>;
pinctrl-1     = <&rgb18_pins_b>;
};
```

## 8.3 驱动移植

参考屏驱动：

uboot-2018:

brandy/brandy-2.0/u-boot-2018/drivers/video/sunxi/disp2/disp/lcd/default\_panel.c

kernel:

bsp/drivers/video/sunxi/disp2/disp/lcd/default\_panel.c

在如下几个地方修改后，即可满足当前屏的驱动需要（驱动中的函数解析请看[屏驱动接口](#)）：

- 根据屏规格书的上下电要求，完成 LCD\_power\_on、LCD\_power\_off 函数的定义；
- 若需要增加或减少屏的休眠唤醒耗时，可修改 LCD\_open\_flow、LCD\_close\_flow 中的延时，在测试没问题的前提下，到达自己需求。

完成如上修改后，若需要单独添加一个驱动，还需要的修改如下：

- 在屏驱动中修改 struct \_\_lcd\_panel 的变量，其中.name 不能跟其它屏驱动的.name 重复，且需要与 dts 中 lcd\_driver\_name 的值一样；
- 修改屏驱动目录下的 panel.c 和 panel.h，在全局结构体变量 panel\_array 中新增刚才添加 \*\*struct \_\_lcd\_panel\*\* 的变量指针。panel.h 中新增 \*\*struct \_\_lcd\_panel\*\* 的声明。
- 修改 Makefile，在 lcd 屏驱动目录的上级的 Makefile 文件中的 disp-objs 中新增刚才添加屏驱动.o



## 9 RGB 转 VGA

### 9.1 lcd 节点配置

配置时需要注意 `lcd_driver_name`、`lcd_convert_if` 的值，其它属性按照 **rgb 并行接口** 的属性配置即可。

#### 9.1.1 lcd\_driver\_name

取值：“gm7123c\_rgb2vga”；

功能：选择 VGA 屏驱动；

#### 9.1.2 lcd\_convert\_if

取值：4；

功能：使能 RGB 转 VGA 功能；

### 9.2 驱动说明

#### 9.2.1 驱动源码位置

bsp/drivers/video/sunxi/disp2/disp/lcd/GM7123C\_RGB2VGA.c

```
struct __lcd_panel gm7123c_rgb2vga_panel = {
    .name = "gm7123c_rgb2vga",
    .func = {
        .cfg_panel_info = LCD_cfg_panel_info,
        .cfg_open_flow = LCD_open_flow,
        .cfg_close_flow = LCD_close_flow,
        .lcd_user_defined_func = LCD_user_defined_func,
        .get_panel_para_mapping = LCD_get_panel_para_mapping,
    }
},
};
```

func 成员中的 `get_panel_para_mapping` 回调来实现多个 vga 屏的兼容功能。

把所有需要适配的 vga 屏时序填入 mapping 结构体中，配置示例如下：

```
static int LCD_get_panel_para_mapping(const struct disp_panel_para_mapping **p_mapping)
{
    static const struct disp_panel_para_mapping mapping[] = {
        {
            .tv_mode = DISP_TV_MOD_720P_60HZ,
            .panel_info = {
                .lcd_x = 1280,
                .lcd_y = 720,
                .lcd_dclk_freq = 74,
                .lcd_ht = 1650,
                .lcd_hbp = 287,
                .lcd_hspw = 110,
                .lcd_vt = 750,
                .lcd_vbp = 25,
                .lcd_vspw = 5,
            },
        },
        {
            .tv_mode = DISP_TV_MOD_1080P_60HZ,
            .panel_info = {
                .lcd_x = 1920,
                .lcd_y = 1080,
                .lcd_dclk_freq = 149,
                .lcd_ht = 2200,
                .lcd_hbp = 194,
                .lcd_hspw = 44,
                .lcd_vt = 1125,
                .lcd_vbp = 42,
                .lcd_vspw = 5,
            },
        },
        {
            .tv_mode = DISP_TV_MODE_NUM,
            .panel_info = {},
        }, /* must be the end */
    };

    *p_mapping = mapping;
    return 0;
}
```



## 10 incell 屏

### 10.0.1 概述

incell 屏幕，LCD 使用 MIPI-DSI 接口，TP 使用 SPI 或 I2C 接口。incell 屏的特点就是将显示及 TP 的控制器集成在了一起，使用同一路电源及中断、复位脚，所以在适配时需要格外注意与 TP 模块的同步及时序上的要求。

### 10.0.2 驱动移植适配

驱动移植适配与 mipi-dsi 屏驱动移植适配是一样的。

### 10.0.3 适配休眠唤醒机制

由于 incell 屏特性，电源与复位跟 TP 共用，在休眠唤醒时会进行掉电处理以保证设备功耗。因此在唤醒时，需要重新走上电流程，并在屏幕上电时序完成后通知 TP 驱动唤醒并重新加载固件。驱动中使用了内核 FB 框架的通知回调接口与 TP 驱动形成联系，在显示休眠或唤醒时，能够通知 TP 也进行休眠或唤醒操作。

这个休眠唤醒机制只需要在内核屏驱动中进行适配，uboot 屏驱动保持不变，按照原来的操作就可以了。

#### 10.0.3.1 唤醒

在 lcd\_power\_on 中，先按照屏幕所需要的上电时序进行适配，适配好后调用 schedule\_work 通知 TP 驱动进行唤醒操作，blank = 10 代表本次事件的 cmd，TP 驱动根据这个 cmd 来进行相应的操作，具体可参考《Android\_Input\_开发指南》NVT36XXX 模组使用 -> 适配休眠唤醒机制：

```
168 static void LCD_power_on(u32 sel)
169 {
170     panel_reset(0);
171     sunxi_lcd_power_enable(sel, 0); /* config lcd_power pin to open lcd power */
172     sunxi_lcd_delay_ms(5);
173     sunxi_lcd_power_enable(sel,
174                             1); /* config lcd_power pin to open lcd power1 */
175     sunxi_lcd_delay_ms(5);
176     sunxi_lcd_power_enable(sel,
177                             2); /* config lcd_power pin to open lcd power2 */
178     sunxi_lcd_delay_ms(5);
179     power_en(1);
180     sunxi_lcd_delay_ms(20);
181     power_gpio_en(1);
182
183     power_en_p(1);
184     sunxi_lcd_delay_ms(5);
185     power_en_n(1);
186     /* panel_reset(1); */
187     sunxi_lcd_delay_ms(40);
188     panel_reset(1);
189     sunxi_lcd_delay_ms(10);
190     panel_reset(0);
191     sunxi_lcd_delay_ms(5);
192     panel_reset(1);
193
194     screen_driver_work.blank = 10;
195     schedule_work(&screen_driver_work.tp_work);
196     sunxi_lcd_delay_ms(60);
197     /* sunxi_lcd_delay_ms(5); */
198
199     sunxi_lcd_pin_cfg(sel, 1);
200 }
201
```

图 10-1: incell\_lcd\_power\_on

### 10.0.3.2 休眠

在 lcd\_power\_off 中，先调用 schedule\_work 通知 TP 驱动进行休眠操作，blank = 9 代表本次事件的 cmd，TP 驱动根据这个 cmd 来进行相应的操作，具体可参考《Android\_Input\_开发指南》NVT36XXX 模组使用 -> 适配休眠唤醒机制，然后按照屏幕所需要的下电时序进行适配：

```
static void LCD_power_off(u32 sel)
{
    screen_driver_work.blank = 9;
    schedule_work(&screen_driver_work.tp_work);

    sunxi_lcd_pin_cfg(sel, 0);
    power_gpio_en(0);
    power_en(0);
    sunxi_lcd_delay_ms(20);
    panel_reset(0);
    sunxi_lcd_delay_ms(5);
    sunxi_lcd_power_disable(sel,
                            2); /* config lcd_power pin to close lcd power2 */
    sunxi_lcd_delay_ms(5);
    sunxi_lcd_power_disable(sel,
                            1); /* config lcd_power pin to close lcd power1 */
    sunxi_lcd_delay_ms(5);
    sunxi_lcd_power_disable(sel,
                            0); /* config lcd_power pin to close lcd power */
}
```

图 10-2: incell\_lcd\_power\_off

## 11 多屏兼容功能

### 11.1 功能说明

自适应多款 LCD 屏，其实就是单一固件同时支持多款接口、型号不一的 LCD 屏幕模组（主要特指屏驱动、时序不能共用），即支持动态识别 LCD 模组型号，并加载对应 LCD 驱动并以相应的 LCD 屏幕进行显示。

对于多屏兼容的实现，除了将需要兼容的屏全部在对应的平台调通点亮外，另一个关键是如何将当前正在使用的屏识别出来并告知驱动框架以加载对应的屏驱动，一般而言有如下几种方法：

- 当使用不同屏时，硬件上也将某些 io 同时拉高或拉低，用 io 电平甚至电压标识屏模组。
- 出厂时根据使用的屏型号，往 flash 中同时烧录一个屏模组型号标识。
- 对 mipi 等支持通信的屏，通过读屏 id 区分。

上述三种方法的缺点如下：

- 对硬件依赖高，需要占用 SOC 宝贵的 IO 口，屏端或者 SOC 的 PCB 端需要提前做好将 IO 电平拉高或拉低，当需要兼容屏较多时，不能简单通过高低电平区分。
- 需要额外烧录一个标志，产线成本有所增加；且一旦标识烧录完成，只支持对应的屏，不能混用其他屏，一定程度上不算多屏兼容。
- 限定屏的接口必须是 mipi-dsi，由于需要逐一 try，耗时较长。

上述三种方法的优点如下：

- 区分方法简单，读 io 软件实现简单可靠。短耗时。
- 不要求额外硬件资源，纯软实现，相对简单可靠，耗时可控。
- 使用上最灵活，不占用额外硬件资源，不需要额外产线操作。

#### 说明

**当且仅当使用屏为mipi-dsi接口时，才能同一板子烧录固件后不做任何修改换上任意兼容屏都可正常显示！**

上述方法 2，需要额外烧录一个表示屏物料的标识，烧录标识完成后，仅支持代表当前标识的屏，不能兼容其他屏。

上述方法 1，若 io 的拉高拉低不是在 lcd 端完成而是在 SOC 端的 PCB 完成的，一旦对应引脚的电平定好，将不能点亮其他屏；但如果对应引脚是在屏端确定电平，SOC 的 PCB 端是悬空的，实际也能实现同一块板子烧录完固件后不做任何修改换上任意兼容屏都可正常显示的效果，但硬件上屏端不一定可以实现将某 IO 电平拉高或拉低。

## 11.2 使用方法

功能支持在一个固件上让 LCD 输出接口兼容多款屏（lcd0、lcd1、lcd2 视平台而定）。在使用前为了避免不必要的麻烦需要先确保兼容的屏在不使用屏兼容功能时能正常点亮。以 lcd0 为例，具体使用方法如下：

1. 打开功能宏定义（一般默认开启），make menuconfig 确认以下宏定义开启：

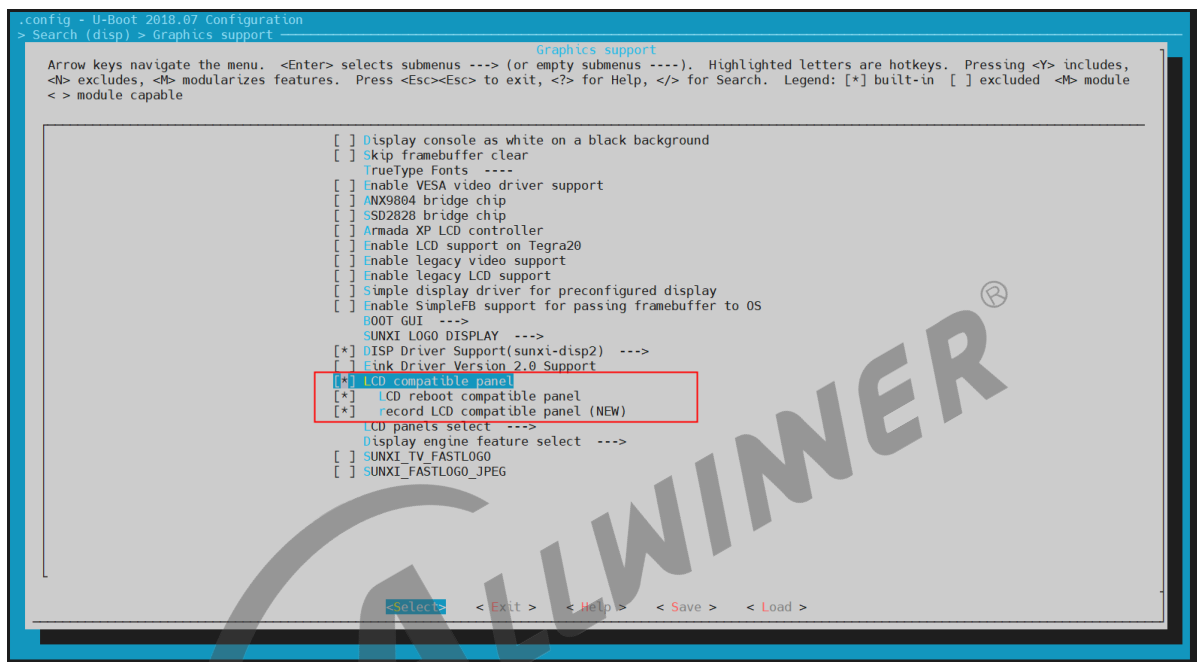


图 11-1: compatible\_lcd\_macro

CONFIG\_COMPATIBLE\_PANEL 配置功能开启，以下两个配置二选一：

- CONFIG\_COMPATIBLE\_PANEL\_RECORD 记录使用面板，减少后续启动时间，不支持关机换屏，换屏后需要重新烧录固件，
- CONFIG\_REBOOT\_COMPATIBLE 不记录使用面板，可关机换屏，但会增加启动时间 1s 左右；

编辑 configs 文件夹对应版型配置文件控制开启与关闭功能。

2. **uboot 和 kernel** 的 dts 中增加节点 lcd0\_1（或者 lcd0\_2、lcd0\_3，根据需要按顺序增加，参考 lcd0 节点确保编译通过），并填入需要对应的屏参数及配置。
3. 在默认的屏，也就是 lcd0 的屏驱动中添加读屏 id(或其他方法) 然后切换的逻辑，切换的接口为 sunxi\_lcd\_switch\_compat\_panel。具体可参照下图 11-2 “屏驱动切换示例”。
4. 在调用 sunxi\_lcd\_switch\_compat\_panel 前，确保执行了之前调用的开屏流程对应的关屏流程，类似图 “屏驱动切换示例” 中的 LCD\_power\_off。

5. 确保在 LCD\_open\_flow 中，调用 sunxi\_lcd\_switch\_compat\_panel 所在的开屏函数以及之前的开屏函数的延时都是 0，具体参照下图 11-3 “屏驱动切换延时”。

```
static void LCD_panel_try_switch(u32 sel)
{
    u8 result[16] = {0};
    u32 num = 0;
    sunxi_lcd_delay_ms(100);
    sunxi_lcd_dsi_dcs_read(sel, 0x04, result, &num);
    printf("get lcd id 0x%x\n", result[0]);

    if (result[0] == 0x93) {
        LCD_power_off(sel);
        sunxi_lcd_switch_compat_panel(sel, 1); /*switch to lcd0_1*/
        return;
    }
}
```

图 11-2: 屏驱动切换示例

```
static s32 LCD_open_flow(u32 sel)
{
    LCD_OPEN_FUNC(sel, LCD_power_on, 0); //must delay 0
    LCD_OPEN_FUNC(sel, LCD_panel_try_switch, 0); //must delay 0
    LCD_OPEN_FUNC(sel, LCD_panel_init, 200); //open lcd power, than delay 200ms
    LCD_OPEN_FUNC(sel, sunxi_lcd_tcon_enable, 50); //open lcd controller, and delay 50ms
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0); //open lcd backlight, and delay 0ms

    return 0;
}
```

图 11-3: 屏驱动切换延时

参考配置：

brandy/brandy-2.0/u-boot-2018/drivers/video/sunxi/disp2/disp/lcd/  
K080\_IM2HYL802R\_800X1280.c

其他说明：

1. 由于实现原理是通过 uboot 替换 kernel 的 lcd 相关的 dts 配置，kernel 的屏驱动无须做任何兼容处理，屏兼容对内核是完全透明的。
2. 为了加快机器的启动速度，默认在第一次启动时记录当前使用的兼容屏，后续启动时将不执行 try 屏驱动的逻辑而直接使用之前记录的屏驱动。如不需要该功能，需要手动关闭 uboot 配置项 “COMPATIBLE\_PANEL\_RECORD”。
3. 若驱动中没有 CONFIG\_REBOOT\_COMPATIBLE 配置，找原厂拿一下补丁。
4. 多屏兼容的重点是识别当前使用的屏物料，功能说明章节中有常用的识别方法及限制说明，请仔细阅读该部分内容，综合评估做好取舍。

## 12 判断是否支持某款 MIPI-DSI 屏

根据 lane 的速度限制，只要 lane\_speed 不超过 IC 规格规定的单 lane 速率，那么理论上是支持的。

```
lane_speed = vt * ht * fps * bit_per_pixel / lane_num / 1e9  
或  
lane_speed = x * y * 1.2 * fps * bit_per_pixel / lane_num / 1e9
```

- 单位：Gbps。
- fps：期望刷新率，通过屏手册可知道，一般是 60。
- bit\_per\_pixel：每个像素包含的比特数量，一般是 24 或者 18。
- lane\_num：lane 数量。
- 1e9：1000000000 的科学计数写法。

选择分辨率的同时需要考虑系统带宽，DE 能力，所以即使接口方面支持这个分辨率，对于整个系统来说不一定支持，比如说硬件为了节省成本选择了一款速度很慢的 DDR 内存然后同时又想选择高分辨率的屏幕，很明显这是不现实的。

## 13 ESD 静电检测自动恢复功能

### 13.1 kernel menuconfig 配置

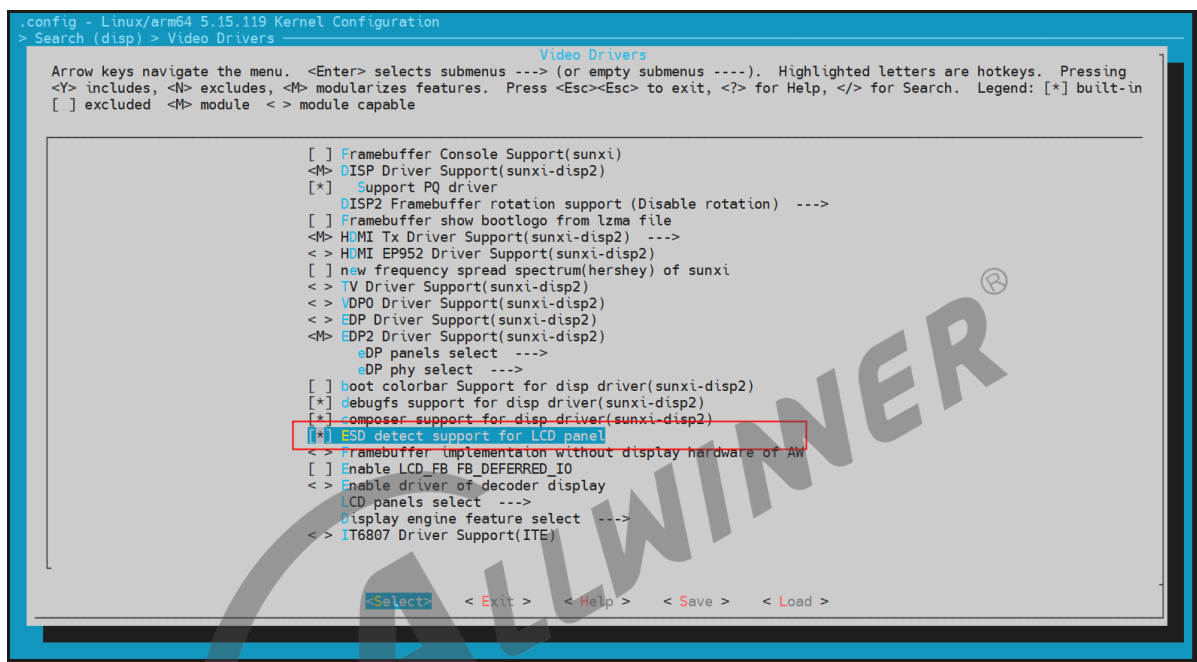


图 13-1: menuconfig

### 13.2 功能使用范例

如下示例，在屏 lq101r1sx03 上添加 esd 相关的回调函数（bsp/drivers/video/sunxi/disp2/disp/lcd/lq101r1sx03.c）：



图 13-2: 屏驱动方法结构体配置

## 13.2.1 esd\_check

- 函数原型：S32 esd\_check(u32 sel)。
- 作用：是给上层反馈当前屏的状态。
- 参数：

(1) sel：显示索引。

- 返回值：

(1) = 0：屏正常；

(2) != 0：屏显不正常。

示例如下：

```
static s32 lcd_esd_check(u32 sel)
{
    s32 ret = -1;
    u8 result[16] = {0};
    u32 num = 0;

    sunxi_lcd_dsi_dcs_read(sel, 0x0A, result, &num); // esd : gh8555bl 0A 9C
    if (result[0] == 0x9C) {
        ret = 0;
    }
    else {
        ret = -1;
    }

    return ret;
}
```

通过 dsi 接口读取 0x0A 命令（获取 power 模式）来判断屏是否正常



## 5.1.4 Read Display Power Mode (0Ah)

0AH	RDDPM (Read Display Power Mode)											
	DCX	RDX	WRX	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	0	0	0	0	1	0	1	0	0A
1 <sup>st</sup> Parameter	1	↑	1	D7	D6	D5	D4	D3	D2	0	0	xx
	This command indicates the current status of the display as described in the table below:											
	Bit		Description					Comment				
	D7		Booster Voltage Status					–				
	D6		Idle Mode On/Off					–				
	D5		Not Defined					Set to "0"				
	D4		Sleep In/Out					–				
	D3		Display Normal Mode On/Off					–				
	D2		Display On/off					–				
	D1		Not Defined					Set to "0"				
	D0		Not Defined					Set to "0"				

图 13-3: 0x0A 命令

## 13.2.2 reset\_panel

- 函数原型：s32 reset\_panel(u32 sel)。
- 作用：当屏幕异常的时候所需要的复位操作。
- 参数：

(1) sel：显示索引。

- 返回值：

(1) 0：复位成功；

(2) !0：复位失败。

每个屏的初始化都不同，顺序步骤都不一样，总的来说就是执行部分或者完整的屏驱动里面的 close\_flow 和 open\_flow 所定义的回调函数。根据实际情况灵活编写这个函数。值得注意的是：某些 dsi 屏中，需要至少执行过一次 sunxi\_lcd\_dsi\_clk\_disable（dsi 高速时钟禁止）和 sunxi\_lcd\_dsi\_clk\_enable（高速时钟使能），否则可能导致 dsi 的读函数异常。下图是复位函数示例：

```
static s32 lcd_reset_panel(u32 sel)
{
    sunxi_lcd_dsi_dcs_write_0para(sel, 0x28);
    sunxi_lcd_delay_ms(50);
    sunxi_lcd_dsi_dcs_write_0para(sel, 0x10);
    sunxi_lcd_delay_ms(200);

    sunxi_lcd_power_disable(sel, 1);
    sunxi_lcd_delay_ms(100);
    sunxi_lcd_power_enable(sel, 1);
    sunxi_lcd_delay_ms(260);

    lcd_panel_init(sel);
    sunxi_lcd_delay_ms(20);

    return 0;
}
```

图 13-4: 复位函数示例 1

### 13.2.3 set\_esd\_info

- 函数原型：s32 set\_esd\_info(struct disp\_lcd\_esd\_info \*p\_info)。
- 作用：控制 esd 检测的具体行为。比如间隔多长时间检测一次，复位的级别，以及检测函数被调用的位置。
- 参数：

(1) p\_info：需要设置的 esd 行为结构体。

- 返回值：

(1) 0：成功设置；

(2) !0：设置失败。

每隔 60 次显示中断检测一次（调用 esd\_check 函数，如果显示帧率是 60fps 的话，那么就是 1 秒一次），然后将在显示中断处理函数里面执行检测函数，由 esd\_check\_func\_pos 成员决定调用 esd\_check 函数的位置，如果是 0 则在中断之外执行检测函数，之所以有这个选项是因为显示中断资源（中断处理时间）是非常珍贵的资源，关系到显示帧率的问题。

level: 1: 表示复位全志 SoC 的 LCD 模块；2: 表示复位全志 SoC 的 disp 模块。

```
s32 set_esd_info(struct disp_lcd_esd_info *p_info) {
    if (!p_info)
        return -1;

    p_info->level = 1;
    p_info->freq = 60;
    p_info->esd_check_func_pos = 1;
    return 0;
}
```

可以通过 `cat /sys/class/disp/disp/attr/sys` 获取当前的 esd info。

```
screen 0:
de_rate 594000000 hz, ref_fps:60
mgr0: 2560x1600 fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits] unblank err[0] force_sync[0]
dmabuf: cache[0] cache max[0] umap skip[0] overflow[0]
capture: dis req[0] runing[0] done[0,0]
    lcd output(enable) backlight( 50) fps:60.9 esd level(1) freq(300) pos(1) reset(244) 2560x1600
    err:0 skip:0 skip T.O:50 irq:73424 vsync:0 vsync_skip:0
    BUF en ch[1] lyr[0] z[0] prem[N] fbd[N] a[global 255] fmt[ 0] fb[2560,1600;2560,1600;2560,1600] crop[ 0, 0,2560,1600]
        frame[ 0, 0,2560,1600] addr[98100000,00000000,00000000] right[00000000,00000000,00000000] flags[0x00] trd
        [0,0] depth[ 0]
acquire: 0, 25.5 fps
release: 0, 25.5 fps
display: 0, 25.5 fps
```

esd level(1) freq(300) pos(1) reset(244)

esd level 和 freq 和 pos 的意思请看上面 set\_esd\_info 函数原型的解释。Reset 后面的数字表示屏复位的次数（也就是 esd 导致屏挂掉之后，并且成功检测到并复位的次数）。

## 14 展频

### 14.1 展频概述

时钟展频通过频率调制的手段，利用低频调制，将集中在窄频带范围内的能量分散到设定的宽频带范围，通过降低时钟在基频和奇次谐波频率的幅度（能量），达到降低系统电磁辐射峰值的目的，其本质是把一个频率固定的信号变成频率周期变化的信号，

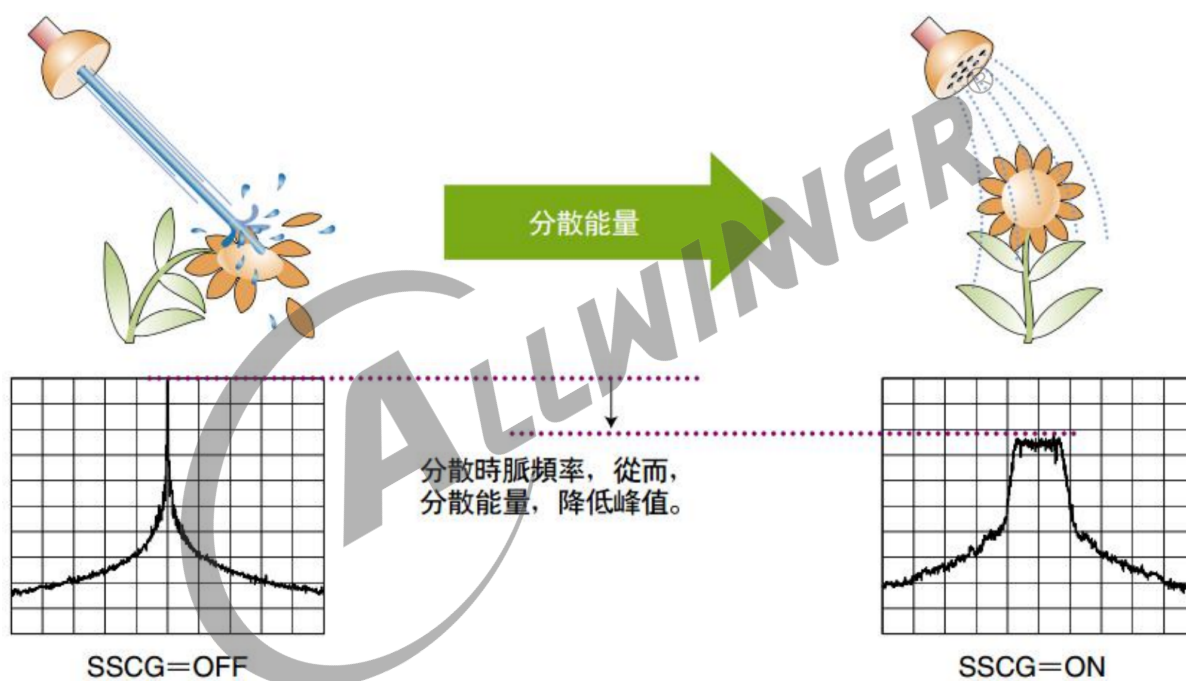


图 14-1: sscg

时钟展频一般包括 4 个参数：扩展率、展频类型、调制频率、调制波形。

- 扩展率：系统时钟频率变化的范围，即展频深度，一般为 0.5%。
- 展频类型：分为向下展频，中心展频，向上展频。
- 调制频率：系统时钟变化的频率，一般有 31.5K、32K、32.5K 和 33KHZ 可选。
- 调制波形：引起系统时钟变化规律的波形，一般为三角波或者 Hershey Kiss 波形。

向上展频和中见展频频谱图变化如下图所示：

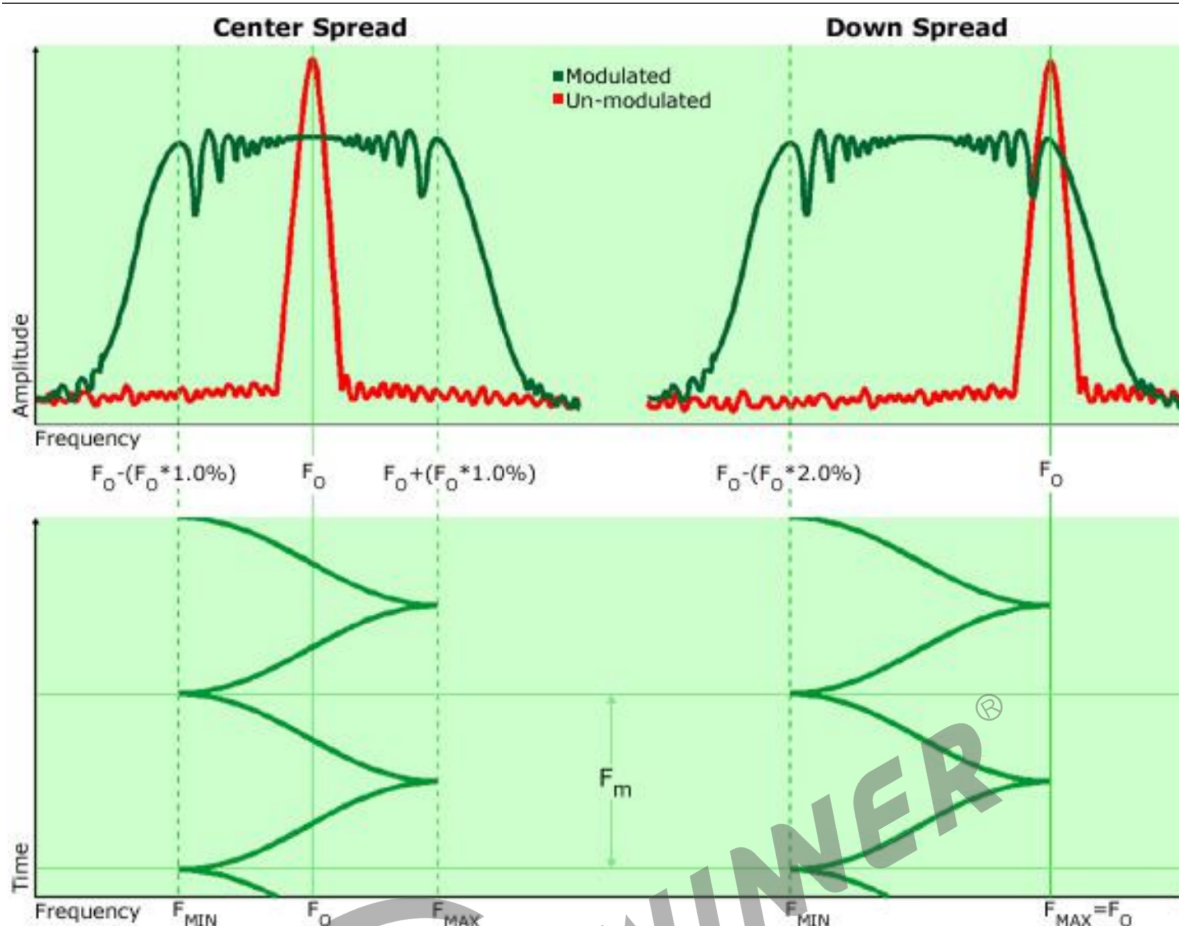


图 14-2: ssc\_way

## 14.2 LVDS 展频功能和快速适配方法

修改板级平台设备树文件 xxx.dts。只有 pll 类型的时钟才可以做展频，在某个模块需要展频时，实际上是要对模块时钟的 pll 时钟源做展频。引用硬件平台设备树头文件 xxx.dtsi 中的时钟模块 ccu 节点，添加 sdm\_info 配置。

```
&ccu {
    sdm_info: sdm_info { //固定名称
        pll-video0 { //配置展频时钟名称name
            sdm-enable = <1>; //使能展频
            sdm-factor = <1>; //展频系数1‰
            freq-mode = <1>; //展频模式，nbit triangular
            sdm-freq = <0>; //展频调制频率，0对应31.5KHz
        };
    };
};
```

- sdm-enable：展频使能，0 关闭展频，1 开启展频。
- sdm-factor：展频幅度系数，单位为 1/1000，范围为 1~99，即有效展频幅度系数范围为：1‰~99‰，注意此处是千分号。
- freq-mode：展频模式，TR\_1 (0) :1bit triangular，TR\_N (1) : nbit triangular。

- sdm-freq：展频调制频率，0~3 分别对应 31.5K、32K、32.5K 和 33K 频率展频。

展频时钟名称 name，与模块时钟源相关。例如：对 sun20iw1p1 的 LVDS 展频，找到 disp 节点下的 TCON\_LCD0 对应的时钟父节点中的 CLK\_PLL\_VIDEO0\_4X，通过输入以下代码挂载 debugfs，并通过读取 clk\_summary 查看时钟树。

```
mount -t debugfs none /sys/kernel/debug;  
cat /sys/kernel/debug/clk/clk_summary;
```

时钟树中找到 pll\_video0\_4x 对应的最顶端 pll 时钟，如图：

pll-video1-2x	0	0	0	594000000	0	0	50000	Y
pll-video0	1	1	0	258000000	0	0	50000	Y
fanout-27m	0	0	0	258000000	0	0	50000	N
tve	0	0	0	258000000	0	0	50000	N
pll-video0-4x	1	1	0	1032000000	0	0	50000	Y
tcon_lcd0	2	2	0	1032000000	0	0	50000	Y

图 14-3: pll\_video0

可以找打最顶端 pll 时钟是 pll-video0，把它作为展频时钟名称 name。

## 15 调试方法

系统起来之后可以读取 sysfs 一些信息，来协助调试。

### 15.1 加快调试速度的方法

很明显，如果你在安卓上调试 LCD 屏会比较不方便，安卓编译时间和安卓固件都太过巨大，每次修改内核后，可能都要经过 10 几分钟才能验证，这样效率就太低下了。

1. 使用 linux 固件而不是安卓固件。SDK 是支持仅仅编译 linux 固件，一般是配置 lichee 或者 longan 的时候选择 linux，打包的时候，用 lichee 或者 longan 根目录下的 build.sh 来打包就行。因为 linux 内核小得多，编译更快，更方便调试。
2. 使用内核来调试 LCD 屏。我们知道 Uboot 和内核都需要添加 LCD 驱动，这样才能快速显示 logo，但是 uboot 并不方便调试，所以有时候我们需要把 uboot 的显示驱动关掉，专心调试内核的 LCD 驱动，调好之后才移植到 uboot，另外这样做的一个优点是，我可以非常方便的修改 lcd timing 而不需要重烧固件。就是利用 uboot 命令的 fdt 命令修改 device tree。

比如说，`fdt set lcd0 lcd_hbp <40>` 更多命令看 `fdt help`。

如何关闭 uboot 显示呢，在 uboot 2018 则是注释掉 configs/平台 \_defconfig 中 `CONFIG_DISP2_SUNXI`。

### 15.2 查看显示信息

以下信息是所有信息中最重要。

```
cat /sys/class/disp/disp/attr/sys  
  
screen 0:  
de_rate 297000000 hz, ref_fps:60  
mgr0: 1280x800 fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits] err[0] force_sync[0] unblank direct_show[false]  
  lcd output backlight( 50) fps:60.9 1280x 800  
  err:0 skip:31 irq:1942 vsync:0 vsync_skip:0  
  BUF enable ch[1] lyr[0] z[0] prem[N] a[globl 255] fmt[ 8] fb[1280, 800;1280, 800;1280, 800] crop[ 0, 0,1280, 800]  
  frame[ 0, 0,1280, 800] addr[ 0, 0, 0] flags[0x 0] trd[0,0]
```

信息	意义
lcd output	表示当前显示接口是 LCD 输出。
1280x800	表示当前 LCD 的分辨率，与 board.dts 中 lcd0 的设置一样。
irq	这是 vsync 中断的次数，每加 1 都代表刷新了一帧，正常来说是一秒 60（期望的 fps）次，重复 cat sys，如果无变化，则异常，可能是屏驱动没有加载
err	这个表示缺数，如果数字很大且一直变化，屏幕会花甚至全黑，全红等，可以微调一下屏参
fps:60.9	后面的数值是实时统计的，正常来说应该是在 60(期望的 fps) 附近，如果差太多则不正常，重新检查屏时序，和在屏驱动的初始化序列是否有被调用到。

## 15.3 查看电源信息

kernel 提供调试节点供电源进行调试进行，我们可以通过 kernel 的调试节点获取各路电源的各个详细状态。

要求内核已打开DEBUG\_FS的宏

以 AXP2101 的设备举例，首先需要 mount debugfs 文件系统。然后就可以很直观的看出各路电源有哪几路设备请求，已经请求的值和目前各路电源的状态。

当然这个只是软件的，实际还是用万用表量为准。

```
mount -t debugfs none /sys/kernel/debug
cat /sys/kernel/debug/regulator/regulator_summary
```

regulator	use	open	bypass	voltage	current	min	max
regulator-dummy	0	1	0	0mV	0mA	0mV	0mV
uart0				0mV	0mV		
axp2101-dcdc1	0	7	0	3300mV	0mA	1500mV	3400mV
spi0				0mV	0mV		
sd0				0mV	0mV		
sd0				0mV	0mV		
sd0				0mV	0mV		
sd0				0mV	0mV		
sd0				0mV	0mV		
reg-virt-consumer.1					0mV	0mV	
axp2101-dcdc2	0	1	0	900mV	0mA	500mV	1540mV
reg-virt-consumer.2					0mV	0mV	
axp2101-dcdc3	0	2	0	1000mV	0mA	500mV	3400mV
cpu0				1000mV	1000mV		
reg-virt-consumer.3					0mV	0mV	
axp2101-dcdc4	0	1	0	1500mV	0mA	500mV	1840mV
reg-virt-consumer.4					0mV	0mV	
axp2101-dcdc5	0	1	0	1400mV	0mA	1400mV	3700mV
reg-virt-consumer.5					0mV	0mV	
axp2101-rtclldo	0	0	0	1800mV	0mA	1800mV	1800mV
axp2101-rtclldo1	0	0	0	1800mV	0mA	1800mV	1800mV
axp2101-aldo1	0	1	0	1800mV	0mA	500mV	3500mV
reg-virt-consumer.8					0mV	0mV	



```

axp2101-ald02      0 2 0 3300mV 0mA 500mV 3500mV
spi2               0mV 0mV
reg-virt-consumer.9      0mV 0mV
axp2101-ald03      0 1 0 3300mV 0mA 500mV 3500mV
reg-virt-consumer.10     0mV 0mV
axp2101-ald04      0 1 0 3300mV 0mA 500mV 3500mV
reg-virt-consumer.11     0mV 0mV
axp2101-bl0d01     0 1 0 1800mV 0mA 500mV 3500mV
reg-virt-consumer.12     0mV 0mV
axp2101-bl0d02     0 1 0 3300mV 0mA 500mV 3500mV
reg-virt-consumer.13     0mV 0mV
axp2101-dl0d01     1 1 0 1200mV 0mA 500mV 3500mV
reg-virt-consumer.14     0mV 0mV
axp2101-dl0d02     0 1 0 1200mV 0mA 500mV 1400mV
reg-virt-consumer.15     0mV 0mV
axp2101-cpusl0d0   0 0 0 900mV 0mA 500mV 1400mV

```

## 15.4 查看 pwm 信息

Pwm 的用处这里是提供背光电源。

```

cat /sys/kernel/debug/pwm

platform/7020c00.s_pwm, 1 PWM device
pwm-0 ((null)): period: 0 ns duty: 0 ns polarity: normal

platform/300a000.pwm, 2 PWM devices
pwm-0 (lcd): requested enabled period: 20000 ns duty: 3984 ns polarity: normal
pwm-1 ((null)): period: 0 ns duty: 0 ns polarity: normal

```

上面的“requested enabled”表示请求并且使能了，括号里面的 lcd 表示是由 lcd 申请的。

## 15.5 查看管脚信息

```

cat /sys/kernel/debug/pinctrl/2000000.pinctrl/pinmux-pins

pin 227 (PH3): twi1 (GPIO UNCLAIMED) function io_disabled group PH3
pin 228 (PH4): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 229 (PH5): (MUX UNCLAIMED) pio:229
pin 230 (PH6): (MUX UNCLAIMED) pio:230
pin 231 (PH7): (MUX UNCLAIMED) pio:231

```

上面的信息我们知道 PH5, PH6 这些 IO 被申请为普通 GPIO 功能，而 PH3 被申请为 twi1

## 15.6 查看时钟信息

```
cat /sys/kernel/debug/clk/clk_summary
```

这个命令可以看哪个时钟是否使能，然后频率是多少。与显示相关的是 tcon, pll\_video, mipi 等。

```
cat /sys/kernel/debug/clk/clk_summary | grep tcon  
cat /sys/kernel/debug/clk/clk_summary | grep pll_video  
cat /sys/kernel/debug/clk/clk_summary | grep mipi
```

## 15.7 查看接口自带 colorbar

显示是一整条链路，中间任何一个环节出错，最终的表现都是显示异常，图像显示异常几个可能原因：

1. 图像本身异常。
2. 图像经过 DE (Display Engine) 后异常。
3. 图像经过接口模块后异常。这是我们关注的点。

有一个简单的方法可以初步判断，接口模块 (tcon 和 dsi 等) 可以自己输出内置的一些 patten，比如说彩条，灰阶图，棋盘图等。当接口输出这些内置 patten 的时候，如果这时候显示就异常，这说明了：

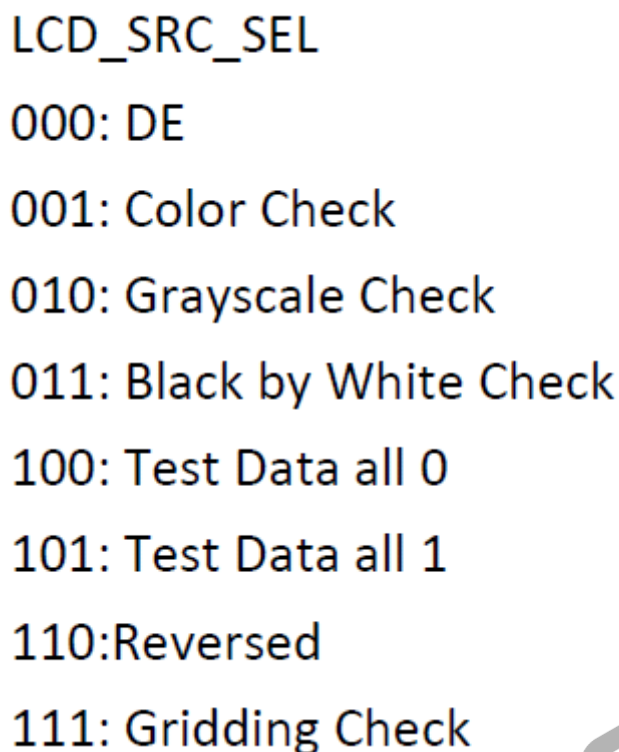
1. LCD 的驱动或者配置有问题
2. LCD 屏由于外部环境导致显示异常

显示自带 patten 的方式：

在 linux-4.9 及其以上版本的内核，disp 的 sysfs 中有一个 attr 可以直接操作显示：

```
echo X > /sys/class/disp/disp/attr/colorbar
```

上面的操作是显示 colorbar，其中的 X 可以是 0 到 8，对应的含义如下图所示：



LCD\_SRC\_SEL

000: DE

001: Color Check

010: Grayscale Check

011: Black by White Check

100: Test Data all 0

101: Test Data all 1

110: Reversed

111: Gridding Check

图 15-1: colorbar

如果有多个显示设备，想让第二个显示设备显示 colorbar 的话，那么先

```
echo 1 > /sys/class/disp/disp/attr/disp
```

然后再执行上面操作。

## 15.8 重启 lcd 显示通路

在 uboot 显示开启的情况下，kernel 不再重新开启显示通路。如果发现设备启动后显示异常，可以在内核通过以下脚本重启 lcd 显示通路，来定位内核部分的显示逻辑是否正常。

```
cd /sys/kernel/  
mount -t debugfs none debug/  
cd debug/dispdbg  
echo lcd0 > name; echo disable > command; echo 1 > start  
  
echo lcd0 > name; echo enable > command; echo 1 > start  
  
#lcd0 代表该屏dts使用的是lcd0节点，请以实际配置为准
```

如果重启 lcd 显示通路后，内核显示正常，则初步定位是 uboot 的显示有问题，需要排查 uboot 有没有正确加载屏驱动，以及执行情况。

## 16 F&Q

### 16.1 屏显示异常

总结过往经验，绝大部分屏显异常都是由于上下电时序和 timing 不合理导致。

### 16.2 黑屏-无背光

问题表现：完全黑屏，背光也没有

有三种可能：

1. 屏驱动添加失败。驱动没有加载屏驱动，导致背光电源相关函数没有运行到。
2. 屏驱动加载成功，但是没有执行到开背光函数（可以在 uboot 的屏驱动中加打印确认开屏流程的执行情况）。这时候大概率是屏驱动的开屏函数没有执行完，uboot 就执行完毕进入内核了。需要在满足屏手册上电时序要求的情况下，尽量减少延迟。
3. pwm 配置和背光电路的问题，另外就是直接测量下硬件测量下相关管脚和电压，再检查屏是否初始化成功。

### 16.3 黑屏-有背光

黑屏但是有背光，可能有多种原因导致，请依次按以下步骤检查：

1. 没送图层。如果应用没有送任何图层那么表现的现象就是黑屏，通过[查看显示信息](#)一小节可以确定有没有送图层。如果确定没有图层，可以通过[查看接口自带 colorbar](#)，确认屏能否正常显示。
2. SoC 端的显示接口模块没有供电。SoC 端模块没有供电自然无法传输视频信号到屏上。一般 SoC 端模块供电的 axp 名字叫做 vcc-lcd, vcc-dsi, vcc33-lcd, vcc18-dsi 等。
3. 复位脚没有复位。如果有复位脚，请确保硬件连接正确，确保复位脚的复位操作有放到屏驱动中。
4. 屏的初始化命令不对。包括各个步骤先后顺序，延时等，这个时候请找屏厂确认初始化命令。

## 16.4 闪屏

分为几种：

1. 屏的整体在闪：

这个最大可能是背光电路的电压不稳定，检查电压。

2. 屏部分在闪，而且是概率性：

board.dts 中的时序填写不合理。

3. 屏上由一个矩形区域在闪：

屏极化导致，需要关机放一边再开机则不会。

## 16.5 条形波纹

有些 LCD 屏的像素格式是 18bit 色深 (RGB666) 或 16bit 色深 (RGB565)，建议打开 FRM 功能，通过 dither 的方式弥补色深，使显示达到 24bit 色深 (RGB888) 的效果。如下图所示，上图是色深为 RGB66 的 LCD 屏显示，下图是打开 dither 后的显示，打开 dither 后色彩渐变的地方过度平滑。设置 lcd\_frm 属性可以改善这种现象。

## 16.6 重启断电测试屏异常

花屏的第一个原因是 fps 过高，超过屏的限制：

FPS 异常是一件非常严重的事情，关系到整个操作系统的稳定，如果 fps 过高会造成系统带宽增加，送显流程异常，fps 过高还会造成 LCD 屏花屏不稳定，容易造成 LCD 屏损坏，FPS 过低则造成用户体验过差。

1. 通过查看[查看显示信息](#)一节，可以得知现在的实时统计的 fps。
2. 如果 fps 离正常值差很多，首先检查 board.dts 中 [lcd0] 节点，所填信息必须满足下面公式：

$$\text{lcd\_dclk\_freq} * \text{num\_of\_pixel\_clk} = \text{lcd\_ht} * \text{lcd\_vt} * \text{fps} / 1e9$$

其中，num\_of\_pixel\_clk 通常为 1，表示发送一个像素所需要的时钟周期为 1 一个，低分辨率的 MCU 和串行接口通常需要 2 到 3 个时钟周期才能发送完一个像素。

如果上面填写没有错，通过查看[查看时钟信息](#)一节可以确认下几个主要时钟的频率信息，把这些信息和 board.dts 发给维护者进一步分析。

## 16.7 RGB 接口或者 I8080 接口显示抖动有花纹

1. 改大时钟管脚的管脚驱动能力，参考lcd\_gpio\_x一小节和pinctrl-0 和 pinctrl-1，修改驱动能力，改大。
2. 修改时钟相位，也就是修改 lcd\_hv\_clk\_phase。由于发送端和接收端时钟相位的不同导致接收端解错若干位。

## 16.8 LCD 屏出现极化和残影

何谓液晶极化现象：实际上就是液晶电介质极化。就是在外界电场作用下，电介质内部沿电场方向产生感应偶极矩，在电解质表明出现极化电荷的现象叫做电介质的极化。

通俗的讲就是在液晶面板施加一定电压后，会聚集大量电荷，当电压消失的时候，这些聚集的电荷也要释放，但由于介电效应，这些聚集的电荷不会立刻释放消失，这些不会马上消失的惰性电荷造成了液晶的 DC 残留从而形成了极化现象。

### 几种常见的液晶极化现象

1. 液晶长期静止某个画面的时候，切换到灰阶画面的时候出现屏闪，屏闪一段时间后消失。这种现象属于残留电荷放电的过程。
2. 液晶长期静止某个画面的时候，出现四周发黑中间发白的现象，业内称为黑白电视框异常。
3. 非法关机的时候，重新上电会出现屏闪，屏闪一定时间后消失。与第一种原因相同。
4. 残影现象：当液晶静止在一个画面比较久的情况下，切换其他画面出现的镜像残留。残影的本质来说是液晶 DC 残留电荷导致，某种意义上来说也属于液晶极化现象。

针对液晶屏出现极化和残影现象，有如下对策。

1. 调整 vcom 电压大小。

VCOM 是液晶分子偏转的参考电压，要求要稳定，对液晶显示有直接影响，具体的屏不同的话也是不同的。电压的具体值是根据输入的数据以及 Vcom 电压大小来确定的，用来显示各种不同灰阶，也就是实现彩色显示 GAMMA。Gamma 电压是用来控制显示器的灰阶的，一般情况下分为 G0~G14，不同的 Gamma 电压与 Vcom 电压之间的压差造成液晶旋转角度不同从而形成亮度的差异，Vcom 电压最好的状况是位于 G0 和 G14 的中间值，这样液晶屏的闪烁状况会最好。

调节 vcom 电压的方式，如果屏管脚有 vcom 管脚，直接调整相关电路，如果屏 driver IC 提供寄存器接口，可以通过寄存器接口来调整大小。

2. 严格按照屏规定的上下电时序来对屏进行开关屏。许多极化残影现象并非长时间显示静止显示某个画面导致的，而是由于关机或者关屏时没有严格按照下电时序导致的，比如该关的电没关，或者延时不够。

## 17 总结

调试 LCD 显示屏实际上就是调试发送端芯片（全志 SOC）和接收端芯片（LCD 屏上的 driver IC）的一个过程：

1. 添加屏驱动请看屏驱动接口适配。
2. 仔细阅读屏手册以及 driver IC 手册。
3. 仔细阅读DTS 适配步骤。
4. 确保 LCD 所需要的各路电源管脚正常。






## 著作权声明

版权所有 © 2024 珠海全志科技股份有限公司。保留一切权利。

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

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

## 商标声明

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

## 免责声明

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

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

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