



Tina Linux SPI LCD 调试指南

**版本号: 1.0
发布日期: 2020.06.19**

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.06.19	AWA1422	初始版本



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 menuconfig 配置说明	2
3 配置案例解析	3
3.1 典型 2 data lane 配置	3
3.2 原 SPI 接口屏配置	4
3.3 带 te 脚的屏	5
3.4 横竖屏旋转	6
3.5 帧率控制	6
3.6 背光控制	6
3.7 开机 logo	7
3.8 像素格式相关	7
3.9 电源配置	7
3.10 多个显示	8
3.11 依赖驱动配置	8
4 lcd_fb0 配置参数详解	10
4.1 lcd_driver_name	10
4.2 lcd_model_name	10
4.3 lcd_if	10
4.4 lcd_dbi_if	11
4.5 lcd_dbi_fmt	14
4.6 lcd_dbi_te	14
4.7 lcd_dbi_clk_mode	15
4.8 lcd_rgb_order	15
4.9 lcd_x	16
4.10 lcd_y	16
4.11 lcd_data_speed	16
4.12 lcd_fps	16
4.13 lcd_pwm_used	17
4.14 lcd_pwm_ch	17
4.15 lcd_pwm_freq	17
4.16 lcd_pwm_pol	17
4.17 lcd_pwm_max_limit	17
4.18 lcd_backlight	18
4.19 lcd_bl_en	18
4.20 lcd_gpio_x	18
4.21 lcd_spi_dc_pin	19

4.22 lcd_spi_bus_num	19
4.23 lcd_pixel_fmt	19
4.24 fb_buffer_num	20
5 编写屏驱动	21
5.1 编写步骤	21
5.2 开关屏流程函数解析	21
5.2.1 LCD_open_flow	22
5.2.2 LCD_close_flow	23
5.2.3 LCD_OPEN_FUNC	23
5.2.4 LCD_power_on	24
5.2.5 LCD_panel_init	24
5.2.6 lcd_fb_black_screen	24
5.2.7 LCD_bl_open	24
5.2.8 LCD_bl_close	25
5.2.9 LCD_power_off	25
5.2.10 sunxi_lcd_delay_ms	25
5.2.11 sunxi_lcd_backlight_enable	25
5.2.12 sunxi_lcd_pwm_enable	25
5.2.13 sunxi_lcd_power_enable	26
5.2.14 sunxi_lcd_cmd_write	26
5.2.15 sunxi_lcd_para_write	26
5.2.16 sunxi_lcd_gpio_set_value	26
5.2.17 sunxi_lcd_gpio_set_direction	27
6 linux fbdev 编程注意事项	28
7 一些有用调试手段	29
7.1 确认 fb 节点是否存在	29
7.2 查看电源信息	29
7.3 查看 pwm 信息	30
7.4 查看管脚信息	30
7.5 查看时钟信息	31
8 FAQ	32
8.1 怎么判断屏初始化成功	32
8.2 黑屏-无背光	32
8.3 黑屏-有背光	32
8.4 没有开机 logo	32
8.5 闪屏	33
8.6 条形波纹	34
8.7 画面偏移	35
9 总结	36

插 图

2-1 内核配置	2
2-2 内核配置 1	2
4-1 4line 模式	11
4-2 L3I1 写时序	12
4-3 L3I1 读时序	12
4-4 L4I 写时序	13
4-5 L4I 读时序	13
4-6 D2LI 写时序	14
4-7 rgb565 像素排列	16
5-1 开关屏函数流程	22
6-1 fb 使用流程	28
8-1 lcd_frm 关闭	34
8-2 lcd_frm 打开	34



1 概述

1.1 编写目的

介绍 R328、R329 以及 R528 平台中 SPI 屏调试使用指南。

1. LCD 调试方法，调试手段。
2. LCD 驱动编写。
3. lcd_fb0 节点下各个属性的解释。
4. 典型 LCD 接口配置。

1.2 适用范围

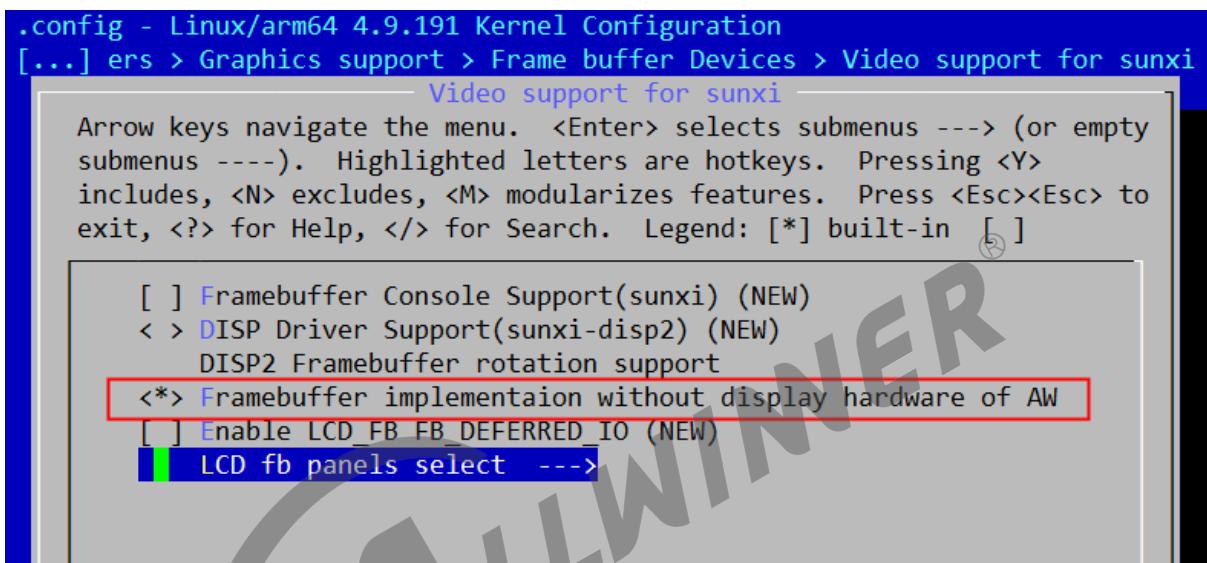
R328, R329 以及 R528 平台。

1.3 相关人员

系统整合人员，显示开发相关人员，客户。

2 menuconfig 配置说明

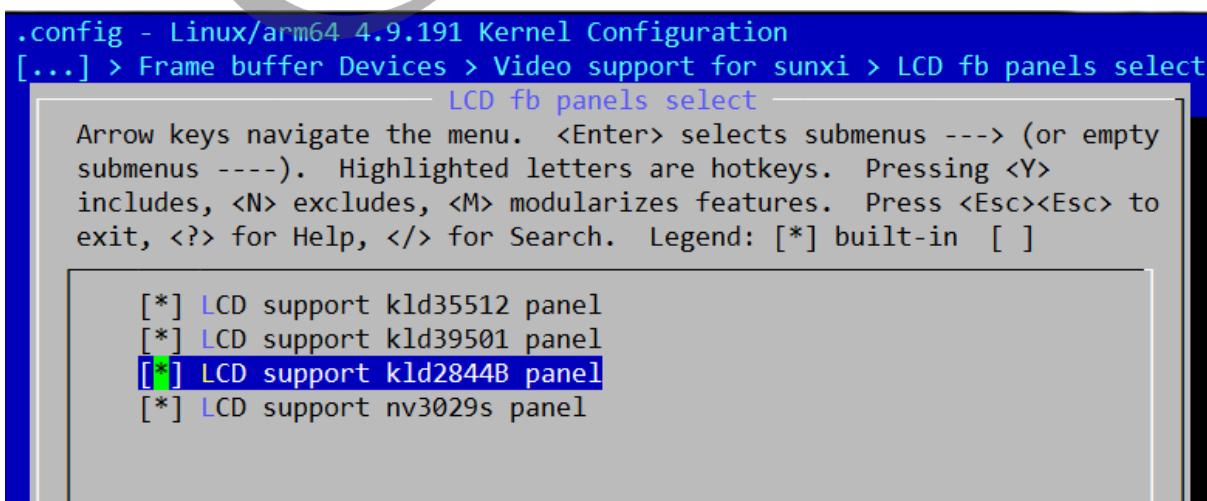
1. 选择 lcd_fb 驱动: Device Drivers -> Graphics support -> Frame buffer Devices -> Video support for sunxi -> Framebuffer implementation without display hardware of AW。



```
.config - Linux/arm64 4.9.191 Kernel Configuration
[...] ers > Graphics support > Frame buffer Devices > Video support for sunxi
                                         Video support for sunxi
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
[ ] Framebuffer Console Support(sunxi) (NEW)
< > DISP Driver Support(sunxi-disp2) (NEW)
      DISP2 Framebuffer rotation support
<*> Framebuffer implementation without display hardware of AW
[ ] Enable LCD_FB FB_DEFERRED_IO (NEW)
[ ] LCD fb panels select --->
```

图 2-1: 内核配置

2. 选择屏驱动: Device Drivers -> Graphics support -> Frame buffer Devices -> Video support for sunxi -> LCD fb panels select。



```
.config - Linux/arm64 4.9.191 Kernel Configuration
[...] > Frame buffer Devices > Video support for sunxi > LCD fb panels select
                                         LCD fb panels select
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
[*] LCD support kld35512 panel
[*] LCD support kld39501 panel
[ ] LCD support kld2844B panel
[*] LCD support nv3029s panel
```

图 2-2: 内核配置 1

3 配置案例解析

3.1 典型 2 data lane 配置

一些屏支持双数据线传输以加快数据传输速度，此时需要走 DBI 协议，典型配置如下：

```
lcd_fb0: lcd_fb0@0 {  
    lcd_used = <1>;  
    lcd_driver_name = "kld2844b";  
    lcd_if = <1>;  
    lcd_dbi_if = <4>;  
    lcd_data_speed = <60>;  
    lcd_spi_bus_num = <1>;  
    lcd_x = <320>;  
    lcd_y = <240>;  
    lcd_pwm_used = <1>;  
    lcd_pwm_ch = <4>;  
    lcd_pwm_freq = <5000>;  
    lcd_pwm_pol = <0>;  
    lcd_pixel_fmt = <0>;  
    lcd_dbi_fmt = <3>;  
    lcd_rgb_order = <0>;  
    fb_buffer_num = <2>;  
    lcd_backlight = <200>;  
    lcd_fps = <60>;  
    lcd_dbi_te = <0>;  
    lcd_bl_en = <&pio PB 2 1 1 2 1>;  
    lcd_gpio_0 = <&pio PB 1 1 1 2 1>;  
};
```

1. 硬件连接上，第二根数据脚连接到原来 1 data lane 的 DC 脚，可以这样理解：2 数据线在传输数据时就自带 D/C(Data/Command) 信息了，所以原来的 DC 脚就可以空出来作为第二根数据线了。对于 R329 来说就是 PH4。
2. 屏驱动上，需要使能 2 data lane 模式，具体寄存器查看对应 driverIC 手册或者询问屏厂。
3. 这里的针对对 2 data lane 的关键参数是 **lcd_if**, **lcd_dbi_if**, **lcd_dbi_fmt** 和 **lcd_spi_bus_num**。
4. **lcd_x** 和 **lcd_y** 是屏分辨率。如果屏支持旋转（横竖旋转），这里也需要对调。
5. **lcd_pwm** 开头，**lcd_backlight** 和 **lcd_bl_en** 是背光相关设置，如果有相关硬件连接的话。

6. `lcd_pixel_fmt`和`fb_buffer_num`是 linux fbdev 的设置。
7. `lcd_gpio_`开头的是自定义 gpio 的设置（比如复位脚）。
8. `lcd_fps`和`lcd_dbi_te`是刷新方式相关的设置。

3.2 原 SPI 接口屏配置

如果 IC 支持 DBI 接口，那么就没有必要用 SPI 接口，DBI 接口其协议能覆盖所有情况。

一些 IC 不支持 DBI(R328)，那么只能用 spi 接口（通过设置`lcd_if`），如果使用 spi 接口，它有一些限制。

1. 不支持 2 data lane。
2. 必须指定 DC 脚。这是由于 spi 协议不会自动控制 DC 脚来区分数据命令，通过设置`lcd_spi_dc_pin`可以完成这个目的，这跟管脚不必用 spi 里面的脚。
3. 只支持 rgb565 的像素格式。由于只有单 data lane，速度过慢，rgb565 以上格式都不现实。

```
lcd_fb0: lcd_fb0@0 {
    lcd_used = <1>;
    lcd_driver_name = "kld2844b";
    lcd_if = <0>;
    lcd_data_speed = <60>;
    lcd_spi_bus_num = <1>;
    lcd_x = <320>;
    lcd_y = <240>;
    lcd_pwm_used = <1>;
    lcd_pwm_ch = <4>;
    lcd_pwm_freq = <5000>;
    lcd_pwm_pol = <0>;
    lcd_pixel_fmt = <10>;
    lcd_rgb_order = <0>;
    fb_buffer_num = <2>;
    lcd_backlight = <200>;
    lcd_fps = <60>;
    lcd_dbi_te = <0>;
    lcd_bl_en = <&pio PB 2 1 1 2 1>;
    lcd_gpio_0 = <&pio PB 1 1 1 2 1>;
    lcd_spi_dc_pin = <&pio PB 3 1 1 2 1>
};
```

另外需要更改 spi 的配置，查看[依赖驱动配置](#)，把 `spi_dbi_enable` 置 0。

3.3 带 te 脚的屏

te 即 (Tearing effect) , 也就是撕裂的意思, 是由于读写不同步导致撕裂现象, te 脚的功能就是用于同步读写, te 脚的频率也就是屏的刷新率, 所以 te 脚也可以看做 vsync 脚 (垂直同步脚) 。

1. 硬件设计阶段, 需要将屏的 te 脚连接到 IC 的 DBI 接口的 te 脚, 比如 R329 里面的 PH5 管脚。
2. 配置上接口使用 dbi 接口。
3. 然后使能 `lcd_dbi_te`。
4. 屏驱动使能 te 功能, 寄存器一般是 35h, 详情看屏对应的 driver IC 手册。
5. 屏驱动设置帧率, 根据屏能接受的传输速度选择合理的帧率 (比如 ST7789H2 里面是通过 c6h 来设置 te 频率) 。

```
lcd_fb0: lcd_fb0@0 {
    lcd_used = <1>;
    lcd_driver_name = "kld2844b";
    lcd_if = <1>;
    lcd_dbi_if = <4>;
    lcd_data_speed = <60>;
    lcd_spi_bus_num = <1>;
    lcd_x = <320>;
    lcd_y = <240>;
    lcd_pwm_used = <1>;
    lcd_pwm_ch = <4>;
    lcd_pwm_freq = <5000>;
    lcd_pwm_pol = <0>;
    lcd_pixel_fmt = <0>;
    lcd_dbi_fmt = <3>;
    lcd_rgb_order = <0>;
    fb_buffer_num = <2>;
    lcd_backlight = <200>;
    lcd_fps = <60>;
    lcd_dbi_te = <1>;
    lcd_bl_en = <&pio PB 2 1 1 2 1>;
    lcd_gpio_0 = <&pio PB 1 1 1 2 1>;
};
```

3.4 横竖屏旋转

1. R328/R329/R528 没有硬件旋转功能，软件旋转太慢而且耗费 CPU。
2. 不少 spi 屏支持内部旋转，需要在屏驱动初始化的时候进行设置，一般是 **36h** 寄存器。

```
//转成横屏
sunxi_lcd_cmd_write(sel, 0x36);
sunxi_lcd_para_write(sel, 0xa0);
```

3. lcd_fb0 的配置中需要将 **lcd_x** 和 **lcd_y** 的值对调，此时软件将屏视为横屏。
4. 屏内部旋转出图效果可能会变差，建议选屏的时候直接选好方向。

3.5 帧率控制

屏的刷新率受限于多方面：

1. SPI/DBI 硬件传输速度，也就是时钟脚的频率。设置 **lcd_data_speed** 可以设置硬件传输速度，最大不超过 100MHz。如果屏能正常接收，这个值自然是越大越好。
2. 屏 driver IC 接收能力。Driver IC 手册中会提到屏的能接受的最大 sclk 周期。
3. 使用 2 data lane 还是 1 data lane，理论上 2 data lane 的速度会翻倍。见 [典型 2 data lane 配置](#)。
4. 像素格式。像素格式决定需要传输的数据量，颜色数量越小的像素格式，帧率越高，但是效果越差。
5. [带 te 脚的屏](#)一节中我们知道，te 相关设置直接影响到屏刷新率。
6. 如果不支持 te，可以通过设置 **lcd_fps** 来控制帧率，你需要根据第一点和第二点选择一个合适的值。

3.6 背光控制

1. 硬件需要支持 pwm 背光电路。
2. 驱动支持 pwm 背光调节，只需要配置好 **lcd_pwm** 开头，**lcd_backlight** 和 **lcd_bl_en** 等背光相关配置即可。
3. 应用层通过标准 linux backlight 接口进行背光控制，如下示例。

```
# 关背光
echo 4 > /sys/class/backlight/lcd_fb0/bl_power
# 开背光
```

```
echo 0 > /sys/class/backlight/lcd_fb0/bl_power
# 调整亮度。0到255，值越大越亮
echo 100 > /sys/class/backlight/lcd_fb0/brightness
```

3.7 开机 logo

必须满足以下条件：

1. bmp 图片的位宽必须是 24 或者 32。
2. 必须把 bmp 图片打包到 boot-resource 分区。
3. bmp 图片的宽高必须与 lcd_fb0 的 lcd_x 和 lcd_y 一样，不能大不能小。
4. lcd_fb 内核模块必须为 built-in。

3.8 像素格式相关

1. `lcd_pixel_fmt`，这个设置项用于设置 fbdev 的像素格式。
2. `lcd_dbi_fmt`，这个用于设置 DBI 接口发送的像素格式。

SPI/DBI 发送数据的时候没有必要发送 alpha 通道，但是应用层却有对应的 alpha 通道，比如 ARGB8888 格式。

这个时候硬件会自动帮我们处理好 alpha 通道，所以 `lcd_pixel_fmt` 选择有 alpha 通道的格式时，`lcd_dbi_fmt` 可以选 `rgb666` 或者 `rgb888`，不用和它一样。

3.9 电源配置

如果使用了 regulator 的话，在 `board.dts` 中的 `soc` 下新增 `lcdfb` 节点：

```
lcdfb: lcdfb@0 {
    /* 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，它之前的字符串则是随意的，不过建议取有意义的名字。而后面的像<®_sw>则必须在 board.dtsi 的 regulator0 节点中找到。

然后修改 lcd_fb0 节点中，如果要使用 reg_sw，则类似下面这样写就行，dc1sw对应dc1sw-supply。

```
lcd_fb0: lcd_fb0@0 {  
    lcd_power = "dc1sw"  
}
```

有多个电源的话，就用 lcd_power1, lcd_power2.....然后屏驱动里面调用[sunxi_lcd_power_enable](#)接口即可。

3.10 多个显示

1. 确定硬件有没有多余的 spi/dbi 接口。目前来说，R329 有一个 spi0 和一个 dbi/spi1，前者用于 spi-nand 和 spi-nor，后者只能二选一，所以 R329 不支持。
2. 需要在 board.dts 里面新增 lcd_fb1，配置方式与 lcd_fb0 一样，其中[lcd_spi_bus_num](#)不能一样。

3.11 依赖驱动配置

lcdfb 模块依赖 spi, pwm 等驱动，请参考相关驱动使用文档。

spi 驱动配置在 board.dts 中一般像下面这样，其中[spi_dbi_enable](#)是否使能 dbi 特性的关键。

```
spil: spi@04026000 {  
    pinctrl-0 = <&spil_pins_a>;  
    pinctrl-1 = <&spil_pins_c>;  
    spi_slave_mode = <0>;  
    spi_dbi_enable = <1>;  
    status = "okay";  
    spi_board1 {  
        device_type = "spi_board1";  
        compatible = "sunxi,spidbi";  
        spi-max-frequency = <0x5f5e100>;  
        reg = <0x0>;  
        status = "disabled";  
    };
```

{;

pwm 配置示例：

```
pwm4: pwm4@02000c00 {  
    pinctrl-names = "active", "sleep";  
    pinctrl-0 = <&pwm4_pin_a>;  
    pinctrl-1 = <&pwm4_pin_b>;  
};
```



4 lcd_fb0 配置参数详解

4.1 lcd_driver_name

Lcd 屏驱动的名字（字符串），必须与屏驱动中 `struct __lcd_panel` 变量的 `name` 成员一致。

4.2 lcd_model_name

Lcd 屏模型名字，非必须，可以用于同个屏驱动中进一步区分不同屏。

4.3 lcd_if

Lcd Interface

设置相应值的对应含义为：

- 0: spi接口
- 1: dbi接口

spi 接口就是俗称的 4 线模式，这是因为发送数据时需要额外借助 DC 线来区分命令和数据，与 sclk, cs 和 sda 共四线。

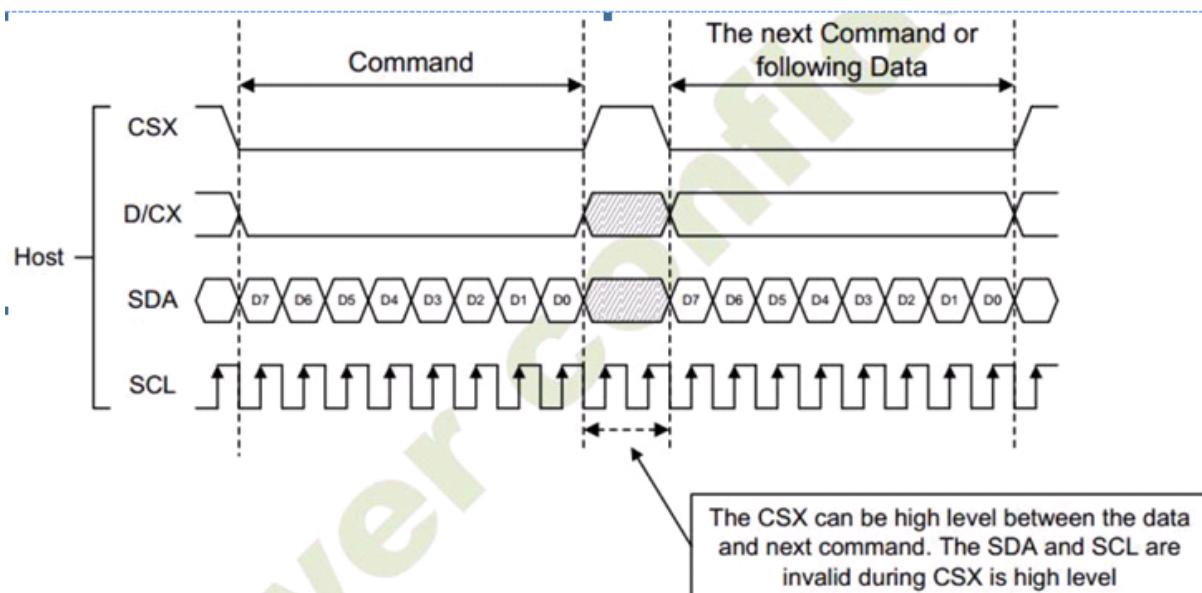


图 4-1: 4line 模式

如果设置了 dbi 接口，那么还需要进一步区分 dbi 接口，需要设置 `lcd_dbi_if`。

4.4 lcd_dbi_if

Lcd dbi 接口设置。

这个参数只有在 `lcd_if=1` 时才有效。

设置相应值的对应含义为：

- | |
|---------|
| 0: L3I1 |
| 1: L3I2 |
| 2: L4I1 |
| 3: L4I2 |
| 4: D2LI |

所有模式在发送数据时每个周期的比特数量根据不同像素格式不同而不同。

L3I1 和 L3I2 是三线模式（不需要 DC 脚），区别是读时序，也就是是否需要额外脚来读寄存器。读写时序图如下：

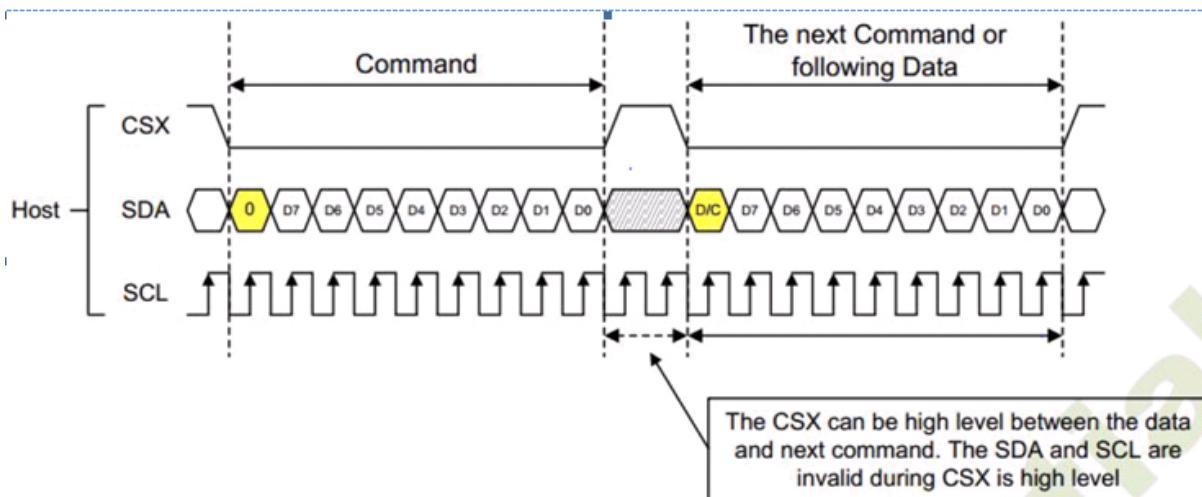


图 4-2: L3I1 写时序

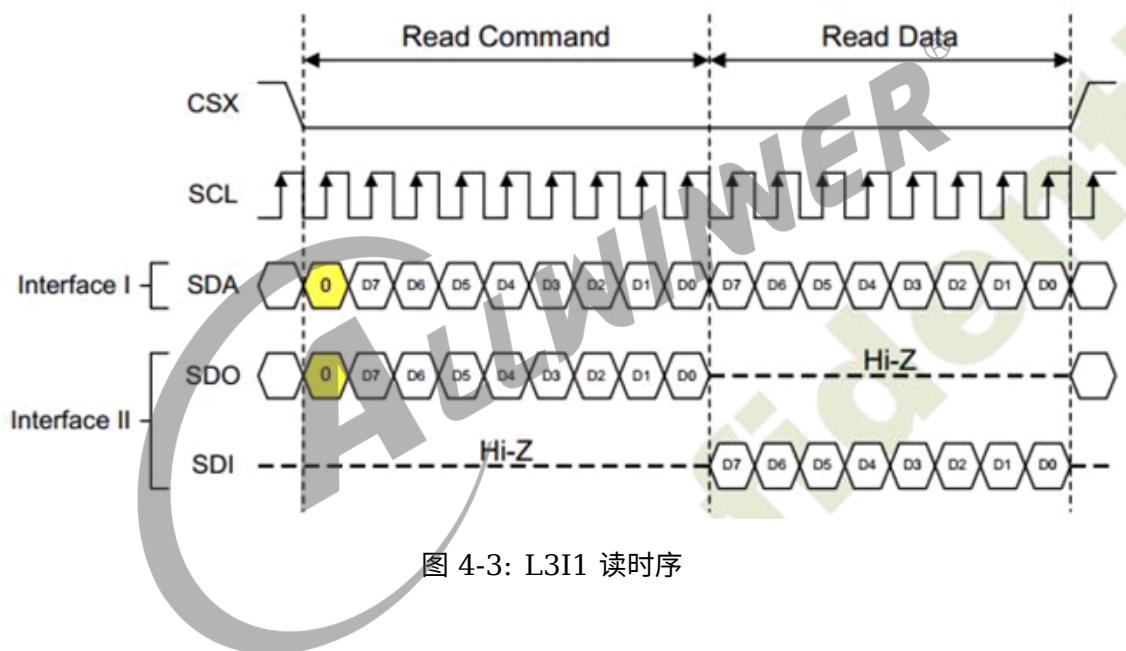


图 4-3: L3I1 读时序

L4I1 和 L4I2 是四线模式，与 spi 接口协议一样，区别是 DC 脚的控制是否自动化控制，另外 I2 和 I1 的区别是读时序，也就是是否需要额外脚来读取寄存器。

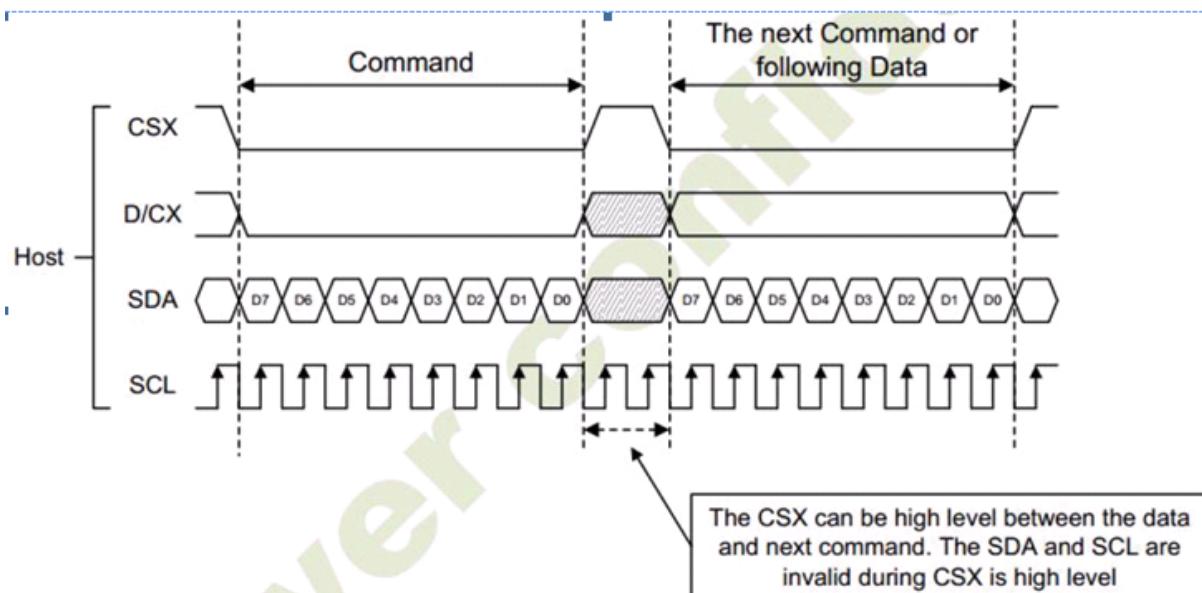


图 4-4: L4I 写时序

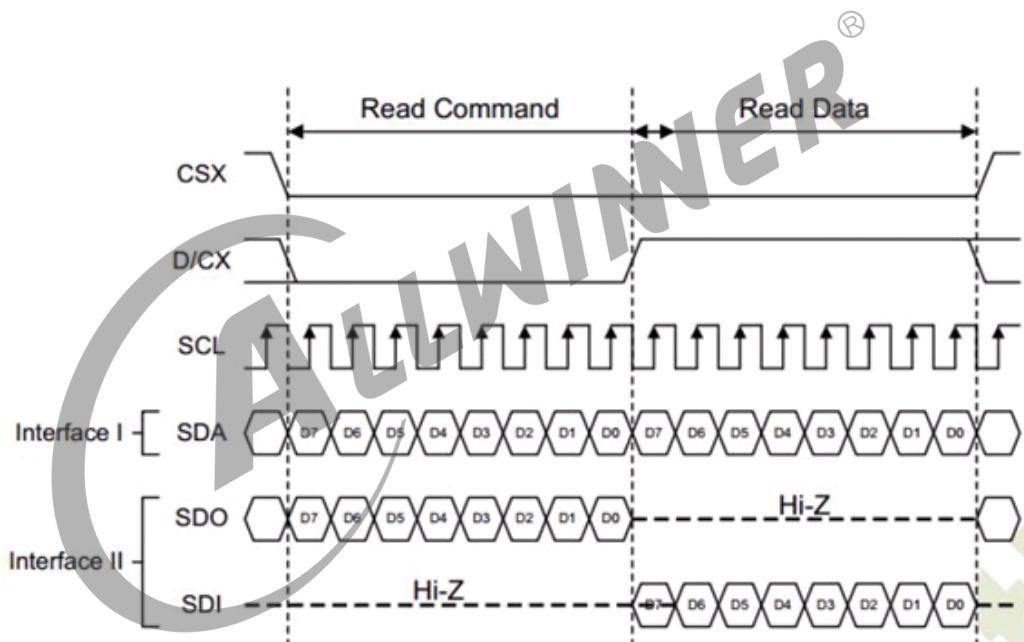
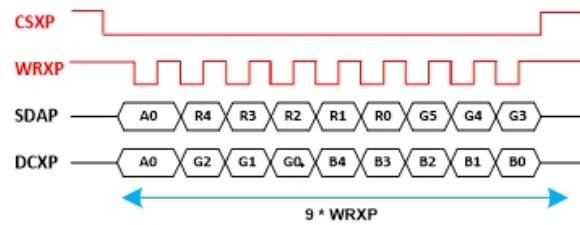


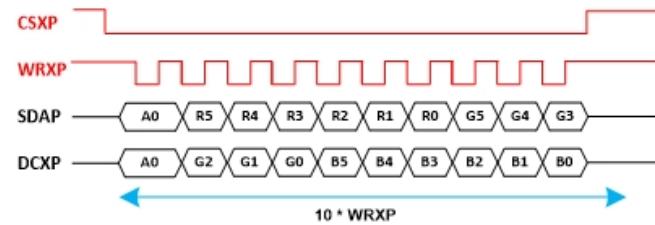
图 4-5: L4I 读时序

D2LI 是两 data lane 模式。发送命令部分时序与读时序与 L3I1 一致，下图是发送数据时的时序，不同像素格式时钟周期数量不一样。

■ RGB565



■ RGB666



■ RGB888

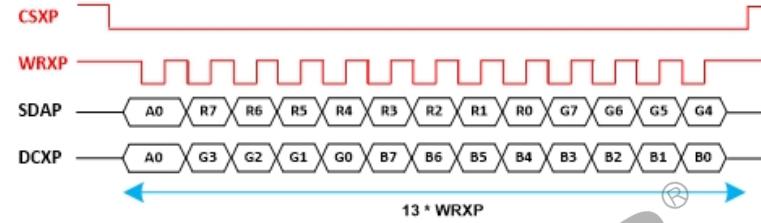


图 4-6: D2LI 写时序

4.5 lcd_dbf_fmt

DBI 接口像素格式。

- 0: RGB111
- 1: RGB444
- 2: RGB565
- 3: RGB666
- 4: RGB888

选择的依据是接收端屏 Driver IC 的支持情况, 请查看 Driver IC 手册或询问屏厂。

然后必须配合 `lcd_pixel_fmt` 的选择, 比如说选 RGB565 时, `lcd_pixel_fmt` 也要选 565 格式。

至于有 alpha 通道的格式, 可以看 [像素格式相关的说明](#)。

4.6 lcd_dbf_te

使能 te 触发。

te 即 (Tearing effect), 也就是撕裂的意思, 由于读写不同导致撕裂现象, te 脚的功能就是用于同步读写, te 脚的频率也就是屏的刷新率, 所以 te 脚也可以看做 vsync 脚 (垂直同步脚)

- 0: 禁止te
- 1: 下降沿触发
- 2: 上升沿触发

查看带 te 脚的屏进一步说明。

4.7 lcd_dbi_clk_mode

选择 dbi 时钟的行为模式。

- 0: 自动停止。有数据就有时钟，没发数据就没有
- 1: 一直保持。无论发不发数据都有时钟

注意上面的选项关系屏兼容性。

4.8 lcd_rgb_order

输入图像数据 rgb 顺序识别设置，仅当 lcd_if=1 时有效。

- 0:RGB
- 1:RBG
- 2:GRB
- 3:GBR
- 4:BRG
- 5:BGR
- 6:G_1RGB_0
- 7:G_0RGB_1
- 8:G_1BRG_0
- 9:G_0BRG_1

非 RGB565 格式用 0 到 5 即可。

针对 rgb565 格式说明如下：

rgb565 格式会遇到大小端问题，arm 平台和 PC 平台存储都是小端 (little endian, 低字节放在低地址，高字节放在高地址) ，但是许多 spi 屏都是默认大端 (BigEndian) 。

也就是存储的字节顺序和发送的字节顺序不对应。

这个时候选择 6 以下，DBI 接口就会自动将小端转成大端。

如果遇到默认是小端的 spi 屏，则需要选择 6 以上，DBI 接口会自动用回小端方式。

6 以上格式这样解释：

R 是 5 比特，G 是 6 比特，B 是 5 比特，再把 G 拆成高 3 位 (G_1) 和低 3 位 (G_0) 所以以下两种顺序：

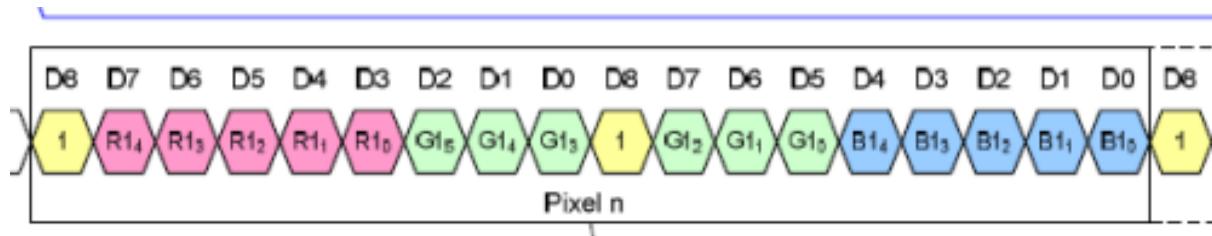


图 4-7: rgb565 像素排列

1. R-G_1-G_0-B, 大端。
2. G_0-B-R-G_1, 对应上面的 9, 小端。

4.9 lcd_x

显示屏的水平像素数量，注意如果屏支持横竖旋转，那么 lcd_x 和 lcd_y 也要对调。

4.10 lcd_y

显示屏的行数，注意如果屏支持横竖旋转，那么 lcd_x 和 lcd_y 也要对调。

4.11 lcd_data_speed

用于设置 spi/dbi 接口时钟的速率，单位 MHz。

1. 发送端 (SOC) 的最大限制是 100MHz。
2. 接收端 (屏 Driver IC) 的限制，请查看对应 Driver IC 手册或者询问屏厂支持。
3. 超出以上限制都有可能导致显示异常。

4.12 lcd_fps

设置屏的刷新率，单位 Hz。当 `lcd_dbi_te` 使能时，这个值设置无效。

4.13 lcd_pwm_used

是否使用 pwm。

此参数标识是否使用 pwm 用以背光亮度的控制。

4.14 lcd_pwm_ch

Pwm channel used

此参数标识使用的 Pwm 通道。

4.15 lcd_pwm_freq

Lcd backlight PWM Frequency

这个参数配置 PWM 信号的频率，单位为 Hz。

4.16 lcd_pwm_pol

Lcd backlight PWM Polarity

这个参数配置 PWM 信号的占空比的极性。设置相应值对应含义为：

0: active high
1: active low

4.17 lcd_pwm_max_limit

Lcd backlight PWM

最高限制，以亮度值表示。

比如 150，则表示背光最高只能调到 150，0~255 范围内的亮度值将会被线性映射到 0~150 范围内。用于控制最高背光亮度，节省功耗。

4.18 lcd_backlight

默认背光值，取值范围 0 到 255，值越大越亮。

4.19 lcd_bl_en

背光使能脚定义。

前提是硬件有做。

```
lcd_bl_en = <&pio PD 25 1 1 2 1>;
```

尖括号内部从左到右 7 个字段意思：

1. &pio这里是大 CPU 管脚，也可以是&r_pio，对应小 cpu 管脚。
2. PD表示 PD 组管脚。
3. 25表示第 25 跟管脚，联合第 2 点，就是 PD25 管脚。
4. 1表示将 PD25 选择为通用 GPIO 功能，0 为输入，1 为输出。
5. 内置电阻；使用 0 的话，标示内部电阻高阻态，如果是 1 则是内部电阻上拉，2 就代表内部电阻下拉。使用 default 的话代表默认状态，即电阻上拉。其它数据无效。
6. 驱动能力；1 是默认等级，数字越大驱动能力越强，最大是 3。
7. 默认输出电平。0 为低电平，1 为高电平。这里要注意的是：这里要默认需要填写使能背光的电平。

4.20 lcd_gpio_x

```
lcd_gpio_0 = <&pio PD 25 1 1 2 1>;
```

尖括号内部从左到右 7 个字段意思：

1. &pio这里是大 CPU 管脚，也可以是&r_pio，对应小 cpu 管脚。
2. PD表示 PD 组管脚。
3. 25表示第 25 跟管脚，联合第 2 点，就是 PD25 管脚。
4. 1表示将 PD25 选择为通用 GPIO 功能，0 为输入，1 为输出。
5. 内置电阻；使用 0 的话，标示内部电阻高阻态，如果是 1 则是内部电阻上拉，2 就代表内部电阻下拉。使用 default 的话代表默认状态，即电阻上拉。其它数据无效。
6. 驱动能力；1 是默认等级，数字越大驱动能力越强，最大是 3。
7. 默认输出电平。0 为低电平，1 为高电平。

注意：如果有多个 gpio 脚需要控制，则定义 lcd_gpio_0, lcd_gpio_1 等。

4.21 lcd_spi_dc_pin

指定作为 DC 的管脚，用于 spi 接口时。

```
lcd_spi_dc_pin = <&pio PD 25 1 1 2 1>;
```

尖括号内部从左到右 7 个字段意思：

1. &pio这里是大 CPU 管脚，也可以是&r_pio，对应小 cpu 管脚。
2. PD表示 PD 组管脚。
3. 25表示第 25 跟管脚，联合第 2 点，就是 PD25 管脚。
4. 1表示将 PD25 选择为通用 GPIO 功能，0 为输入，1 为输出。
5. 内置电阻；使用 0 的话，标示内部电阻高阻态，如果是 1 则是内部电阻上拉，2 就代表内部电阻下拉。使用 default 的话代表默认状态，即电阻上拉。其它数据无效。
6. 驱动能力；1 是默认等级，数字越大驱动能力越强，最大是 3。
7. 默认输出电平。0 为低电平，1 为高电平。

4.22 lcd_spi_bus_num

选择 spi 总线 id，只有 spi1 支持 DBI 协议，所以这里一般选择 1。

取值范围：0 到 1。

4.23 lcd_pixel_fmt

选择 linux fbdev 的像素格式，这个格式的选择将直接影响 fbdev 中的 var 和 fix info。

可选值如下，当你更换 RGB 分量顺序的时候，也得相应修改lcd_rgb_order，或者修改屏驱动的rgb 分量顺序（一般是 3Ah 寄存器）。

DBI 接口只支持 RGB32 和 RGB16 的情况。

SPI 接口只支持 RGB16 的情况。

```
enum lcdfb_pixel_format {
    LCDFB_FORMAT_ARGB_8888 = 0x00, // MSB A-R-G-B LSB
    LCDFB_FORMAT_ABGR_8888 = 0x01,
    LCDFB_FORMAT_RGBA_8888 = 0x02,
```

```
LCDFB_FORMAT_BGRA_8888 = 0x03,  
LCDFB_FORMAT_XRGB_8888 = 0x04,  
LCDFB_FORMAT_XBGR_8888 = 0x05,  
LCDFB_FORMAT_RGBX_8888 = 0x06,  
LCDFB_FORMAT_BGRX_8888 = 0x07,  
LCDFB_FORMAT_RGB_888 = 0x08,  
LCDFB_FORMAT_BGR_888 = 0x09,  
LCDFB_FORMAT_RGB_565 = 0x0a,  
LCDFB_FORMAT_BGR_565 = 0x0b,  
LCDFB_FORMAT_ARGB_4444 = 0x0c,  
LCDFB_FORMAT_ABGR_4444 = 0x0d,  
LCDFB_FORMAT_RGBA_4444 = 0x0e,  
LCDFB_FORMAT_BGRA_4444 = 0x0f,  
LCDFB_FORMAT_ARGB_1555 = 0x10,  
LCDFB_FORMAT_ABGR_1555 = 0x11,  
LCDFB_FORMAT_RGBA_5551 = 0x12,  
LCDFB_FORMAT_BGRA_5551 = 0x13,  
};
```

4.24 fb_buffer_num

linux fbdev 的 buffer 数量，为了平滑显示，这里一般是 2 个，为了省内存也可以改成 1。

5 编写屏驱动

5.1 编写步骤

屏驱动源码位置：

```
drivers/video/fbdev/sunxi/lcd_fb/panels
```

1. 在屏驱动源码位置下拷贝现有一个屏驱动，包括头文件和源文件，然后将文件名改成有意义的名字，比如屏型号。
2. 修改源文件中的`struct __lcd_panel`变量的名字，以及这个变量成员`name`的名字，这个名字必须和`board.dts`中`[lcd_fb0]`的`lcd_driver_name`一致。
3. 在屏驱动目录下修改`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。并用宏括起来。
4. 修改`drivers/video/fbdev/sunxi/lcd_fb/panels/Kconfig`，新增一个 config，与第三点提到的宏对应。
5. 修改`drivers/video/fbdev/sunxi/lcd_fb/Makefile`。给`lcd_fb-obj`变量新增刚才加入的源文件对应`.o`。
6. 根据本手册以及屏手册，Driver IC 手册修改`board.dts`中的`[lcd_fb0]`节点下面的属性，各个属性的含义请看`lcd_fb0` 配置参数详解。
7. 实现屏源文件中的`LCD_open_flow`, `LCD_close_flow`, `LCD_panel_init`, `LCD_power_on` 等函数，请看开关屏流程函数解析。

5.2 开关屏流程函数解析

开关屏的操作流程如图 10 所示。

其中，`LCD_open_flow` 和 `LCD_close_flow` 称为开关屏流程函数，该函数利用 `LCD_OPEN_FUNC` 进行注册回调函数，先注册先执行，可以注册多个，不限制数量。

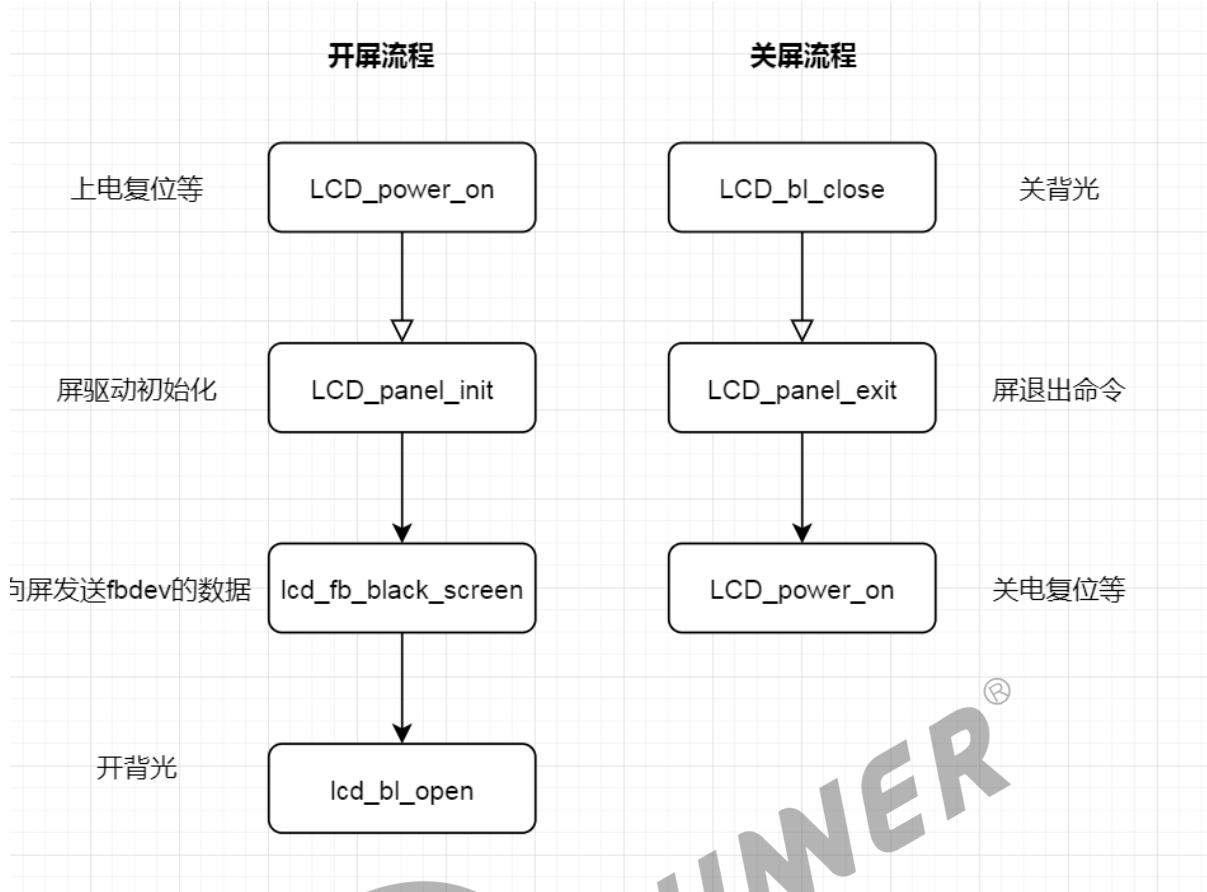


图 5-1: 开关屏函数流程

5.2.1 LCD_open_flow

功能：初始化开屏的步骤流程。

原型：

```
static __s32 LCD_open_flow(__u32 sel)
```

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, lcd_fb_black_screen, 100);
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0);
    return 0;
}
```

如上，初始化整个开屏的流程步骤为四个：

1. 打开 LCD 电源，再延迟 10ms。
2. 初始化屏，再延迟 50ms；（不需要初始化的屏，可省掉此步骤）。
3. 向屏发送 fbdev 的数据。这一步骤是必须的，而且需要在开背光之前。
4. 打开背光，再延迟 0ms。

LCD_open_flow 函数只会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，**并没有立即执行该开屏步骤函数**。LCD_open_flow 函数的内容必须统一用 LCD_OPEN_FUNC(sel, function, delay_time) 进行函数注册的形式，确保正常注册到开屏步骤中。

LCD_OPEN_FUNC 的第二个参数是前后两个步骤的延时长度，单位 ms，注意这里的数值请按照屏手册规定去填，乱填可能导致屏初始化异常或者开关屏时间过长，影响用户体验。

5.2.2 LCD_close_flow

功能：初始化关屏的步骤流程。

原型：

```
static __s32 LCD_close_flow(__u32 sel)
```

函数常用内容为：

```
static __s32 LCD_close_flow(__u32 sel)
{
    LCD_CLOSE_FUNC(sel, LCD_bl_close, 50);
    LCD_CLOSE_FUNC(sel, LCD_panel_exit, 10);
    LCD_CLOSE_FUNC(sel, LCD_power_off, 10);
    return 0;
}
```

1. LCD_bl_close，是关背光，关完背光在处理其它事情，不会影响用户视觉。
2. LCD_panel_exit，发送命令让屏退出工作状态。
3. 关电复位，让屏彻底关闭。

5.2.3 LCD_OPEN_FUNC

功能：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！

原型：

```
void LCD_OPEN_FUNC(__u32 sel, LCD_FUNC func, __u32 delay)
```

参数说明：

func 是一个函数指针，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：

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

delay 是执行该步骤后，再延迟的时间，时间单位是毫秒。

5.2.4 LCD_power_on

这是开屏流程中第一步，一般在这个函数使用 `sunxi_lcd_gpio_set_value` 进行 GPIO 控制，用 `sunxi_lcd_power_enable` 函数进行电源开关。

参考屏手册里面的上电时序（Power on sequence）。

5.2.5 LCD_panel_init

这是开屏流程第二步，一般使用 `sunxi_lcd_cmd_write` 和 `sunxi_lcd_para_write` 对屏寄存器进行初始化。

请向屏厂索要初始化寄存器代码或者自行研究屏 Driver IC 手册。

5.2.6 lcd_fb_black_screen

这是更新 linux fbdev 内存的函数，是必须的，否则打开背光后，呈现给你将是花屏。

5.2.7 LCD_bl_open

这是背光使能，固定调用。

1. `sunxi_lcd_backlight_enable`, 打开 `lcd_bl_en` 脚。
2. `sunxi_lcd_pwm_enable`, 使能 `pwm`。

5.2.8 LCD_bl_close

这是关闭背光。固定调用下面两个函数，分别是：

1. `sunxi_lcd_backlight_disable`, `lcd_bl_en` 关闭
2. `sunxi_lcd_pwm_disable`, 关闭 `pwm`。

5.2.9 LCD_power_off

这是关屏流程中最后一步，一般在这个函数使用`sunxi_lcd_gpio_set_value`进行 GPIO 控制，用`sunxi_lcd_power_enable`函数进行电源开关。

参考屏手册里面的下电时序（Power off sequence）。

5.2.10 sunxi_lcd_delay_ms

函数：**`sunxi_lcd_delay_ms`/`sunxi_lcd_delay_us`**

功能：延时函数，分别是毫秒级别/微秒级别的延时。

原型：s32 `sunxi_lcd_delay_ms(u32 ms);` / s32 `sunxi_lcd_delay_us(u32 us);`

5.2.11 sunxi_lcd_backlight_enable

函数：**`sunxi_lcd_backlight_enable`/`sunxi_lcd_backlight_disable`**

功能：打开/关闭背光，操作的是`lcd_bl_en`。

原型：void `sunxi_lcd_backlight_enable(u32 screen_id);`

void `sunxi_lcd_backlight_disable(u32 screen_id);`

5.2.12 sunxi_lcd_pwm_enable

函数：**`sunxi_lcd_pwm_enable` / `sunxi_lcd_pwm_disable`**

功能：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`。

原型：s32 `sunxi_lcd_pwm_enable(u32 screen_id);`

s32 `sunxi_lcd_pwm_disable(u32 screen_id);`

5.2.13 sunxi_lcd_power_enable

函数：**sunxi_lcd_power_enable / sunxi_lcd_power_disable**

功能：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) 。

原型：void sunxi_lcd_power_enable(u32 screen_id, u32 pwr_id);

void sunxi_lcd_power_disable(u32 screen_id, u32 pwr_id);

1. pwr_id = 0: 对应于 sys_config.fex 中的 lcd_power。
2. pwr_id = 1: 对应于 sys_config.fex 中的 lcd_power1。
3. pwr_id = 2: 对应于 sys_config.fex 中的 lcd_power2。
4. pwr_id = 3: 对应于 sys_config.fex 中的 lcd_power3。

5.2.14 sunxi_lcd_cmd_write

函数：**sunxi_lcd_cmd_write**

功能：使用 spi/dbi 发送命令。

原型：s32 sunxi_lcd_cmd_write(u32 screen_id, u8 cmd);

5.2.15 sunxi_lcd_para_write

函数：**sunxi_lcd_para_write**

功能：使用 spi/dbi 发送参数。

原型：s32 sunxi_lcd_para_write(u32 screen_id, u8 para);

5.2.16 sunxi_lcd_gpio_set_value

函数：**sunxi_lcd_gpio_set_value**

功能：LCD_GPIO PIN 脚上输出高电平或低电平。

原型：s32 sunxi_lcd_gpio_set_value(u32 screen_id, u32 io_index, u32 value);

参数说明：

- io_index = 0: 对应于 sys_config.fex 中的 lcd_gpio_0。

- `io_index = 1`: 对应于 `sys_config.fex` 中的 `lcd_gpio_1`。
- `io_index = 2`: 对应于 `sys_config.fex` 中的 `lcd_gpio_2`。
- `io_index = 3`: 对应于 `sys_config.fex` 中的 `lcd_gpio_3`。
- `value = 0`: 对应 IO 输出低电平。
- `Value = 1`: 对应 IO 输出高电平。

只用于该 GPIO 定义为输出的情形。

5.2.17 `sunxi_lcd_gpio_set_direction`

函数：**`sunxi_lcd_gpio_set_direction`**

功能：设置 LCD_GPIO PIN 脚为输入或输出模式。

原型：

```
s32 sunxi_lcd_gpio_set_direction(u32 screen_id, u32 io_index, u32 direction);
```

参数说明：

- `io_index = 0`: 对应于 `sys_config.fex` 中的 `lcd_gpio_0`。
- `io_index = 1`: 对应于 `sys_config.fex` 中的 `lcd_gpio_1`。
- `io_index = 2`: 对应于 `sys_config.fex` 中的 `lcd_gpio_2`。
- `io_index = 3`: 对应于 `sys_config.fex` 中的 `lcd_gpio_3`。
- `direction = 0`: 对应 IO 设置为输入。
- `direction = 1`: 对应 IO 设置为输出。

6 linux fbdev 编程注意事项

与正常的 fbdev 操作不同的是，write fbdev 内存不会立刻显示，需要调用 FBIO_PAN_DISPLAY 才会显示，即使你没有移动的需求（此时 FBIO_PAN_DISPLAY 的坐标是 0 和 0 即可）。

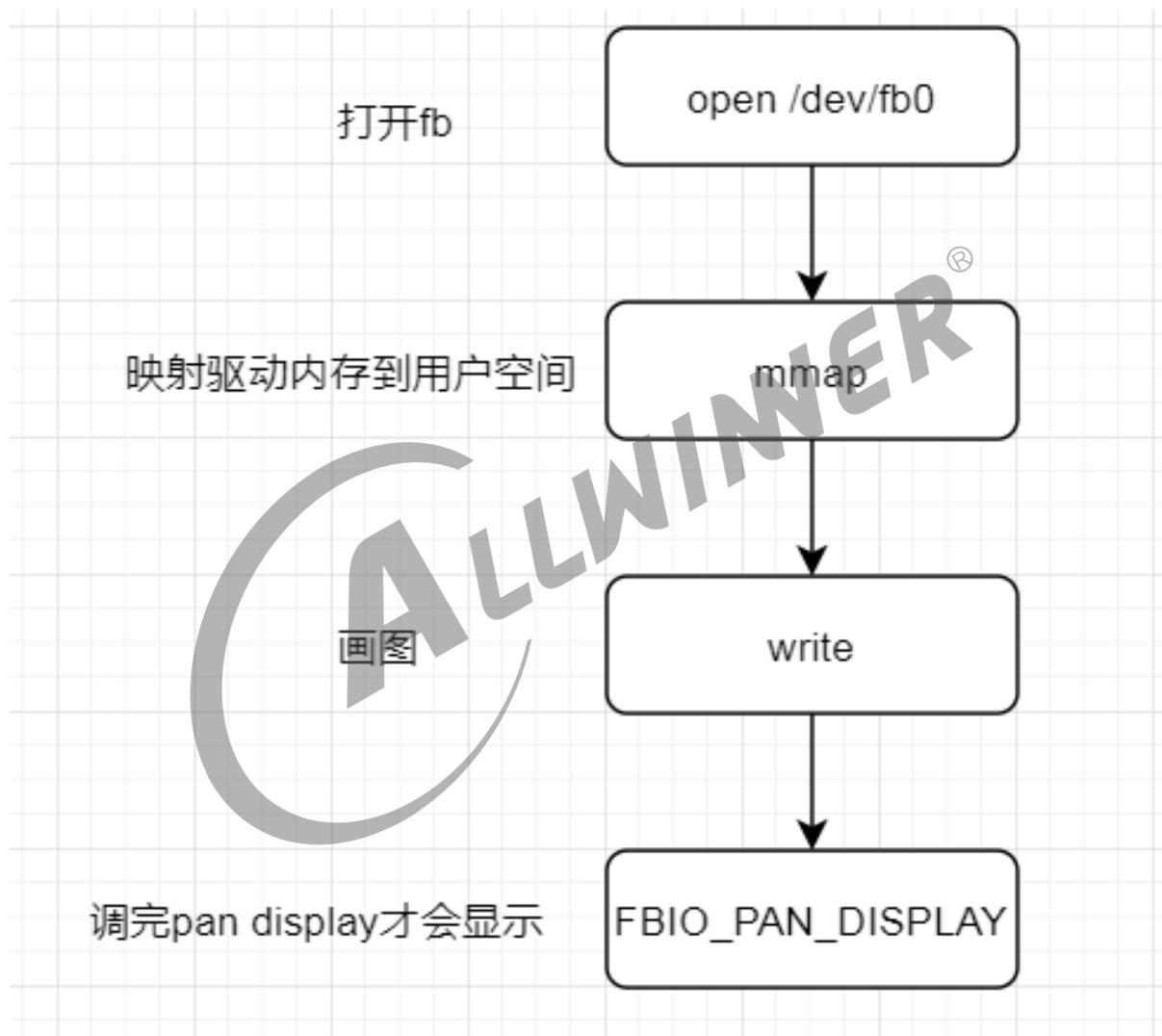


图 6-1: fb 使用流程

7 一些有用调试手段

7.1 确认 fb 节点是否存在

```
ls /dev/fb0
```

7.2 查看电源信息

查看 axp 某一路电源是否有 enable 可以通过下面命令查看。当然这个只是软件的，实际还是用万用表量为准。

```
cat /sys/class/regulator/dump
```

```
pmu1736_ldoio2 : disabled 0 700000 supply_name:
pmu1736_ldoiol : disabled 0 700000 supply_name:
pmu1736_dclsw : enabled 1 3300000 supply_name: vcc-lcd
pmu1736_cpus : enabled 0 900000 supply_name:
pmu1736_cldo4 : disabled 0 700000 supply_name:
pmu1736_cldo3 : disabled 0 700000 supply_name:
pmu1736_cldo2 : enabled 1 3300000 supply_name: vcc-pf
pmu1736_cldo1 : disabled 0 700000 supply_name:
pmu1736_bldo5 : enabled 2 1800000 supply_name: vcc-cpvin vcc-
pc
pmu1736_bldo4 : disabled 0 700000 supply_name:
pmu1736_bldo3 : disabled 0 700000 supply_name:
pmu1736_bldo2 : disabled 0 700000 supply_name:
pmu1736_bldo1 : disabled 0 700000 supply_name:
pmu1736_aldo5 : enabled 0 2500000 supply_name:
pmu1736_aldo4 : enabled 0 3300000 supply_name:
pmu1736_aldo3 : enabled 1 1800000 supply_name: avcc
pmu1736_aldo2 : enabled 0 1800000 supply_name:
pmu1736_aldo1 : disabled 0 700000 supply_name:
pmu1736_rtc : enabled 0 1800000 supply_name:
pmu1736_dcdo6 : disabled 0 500000 supply_name:
```

```
pmu1736_dcdc5 : enabled 0 1480000 supply_name:
pmu1736_dcdc4 : enabled 1 900000 supply_name: vdd-sys
pmu1736_dcdc3 : enabled 0 900000 supply_name:
pmu1736_dcdc2 : enabled 0 1160000 supply_name:
pmu1736_dcdc1 : enabled 4 3300000 supply_name: vcc-emmc vcc-io
vcc-io vcc-io
```

7.3 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。

7.4 查看管脚信息

```
cat /sys/kernel/debug/pinctrl/pio/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。

7.5 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。

这里相关的就是 spi:

```
cat /sys/kernel/debug/clk/clk_summary | grep spi
```



8 FAQ

8.1 怎么判断屏初始化成功

屏初始化成功，一般呈现的现象是雪花屏。

8.2 黑屏-无背光

一般是电源或者 pwm 相关配置没有配置好。参考 [lcd_pwm](#) 开头的相关配置。®

8.3 黑屏-有背光

排除步骤：

1. 首先用 linux fbdev 接口画图，如果没显示，看第二步。
2. 屏驱动里面，在 LCD_open_flow 中删除 lcd_fb_black_screen 的注册，启动后，如果屏初始化成功应该是花屏状态（大部分屏如此）。
3. 如果屏没有初始化成功，请检查屏电源，复位脚状态。
4. 如果屏初始化成功，但是发数据时又没法显示，那么需要检查是不是帧率过快，查看 [帧率控制](#)。
5. 如果电源复位脚正常，请检查配置，[lcd_dbi_if](#), [lcd_dbi_fmt](#) 是否正确，屏是否支持，如果支持，在屏驱动里面是否有对应上。
6. 尝试修改 [lcd_dbi_clk_mode](#)。

8.4 没有开机 logo

查看 [开机 logo](#) 小节，是否需要满足的条件。

然后观察 uboot 和 kernel 的 log 打印。

Uboot 如果读到 logo 的打印如下：

```
[00.824] bmp_name=bootlogo.bmp
307338 bytes read in 17 ms (17.2 MiB/s)
```

进入到内核后，可以 cat 以下节点确认 uboot 的 logo 是否读取成功，找到 disp_reserve 即成功。

```
cat /proc/cmdline | grep disp_reserve
disp_reserve=307338,0x46b5c900
```

内核打印中，查看 [LCD_FB] 开头的打印：

下面的打印有两种可能：

1. lcd_fb 非 built-in。
2. uboot 读取 logo 失败。

```
[LCD_FB] Fb_map_kernel_logo,line:201:
Fb_map_kernel_logo failed
```

下面打印由于 logo 大小不符合要求：

```
[LCD_FB] Fb_map_kernel_logo,line:241:
Bmp [120 x 24] is not equal to fb[320 x 240]
```

下面打印 logo 不是 bmp：

```
[LCD_FB] Fb_map_kernel_logo,line:223:
this is not a bmp picture.
```

下面打印 logo 不是 bmp24 或者 bmp32：

```
[LCD_FB] Fb_map_kernel_logo,line:232:
Only bmp24 and bmp32 is supported!
```

8.5 闪屏

非常有可能是速度跑太快，参考 [帧率控制](#) 一小节。

8.6 条形波纹

有些 LCD 屏的像素格式是 18bit 色深 (RGB666) 或 16bit 色深 (RGB565) , 而输入源却是 RGB888 这样造成颜色损失, 如下面两图的对比。

R328 /R329 没有 dither 功能, 要么是选择支持 full color 的屏 (rgb888), 要么接受这种情况。



图 8-1: lcd_frm 关闭

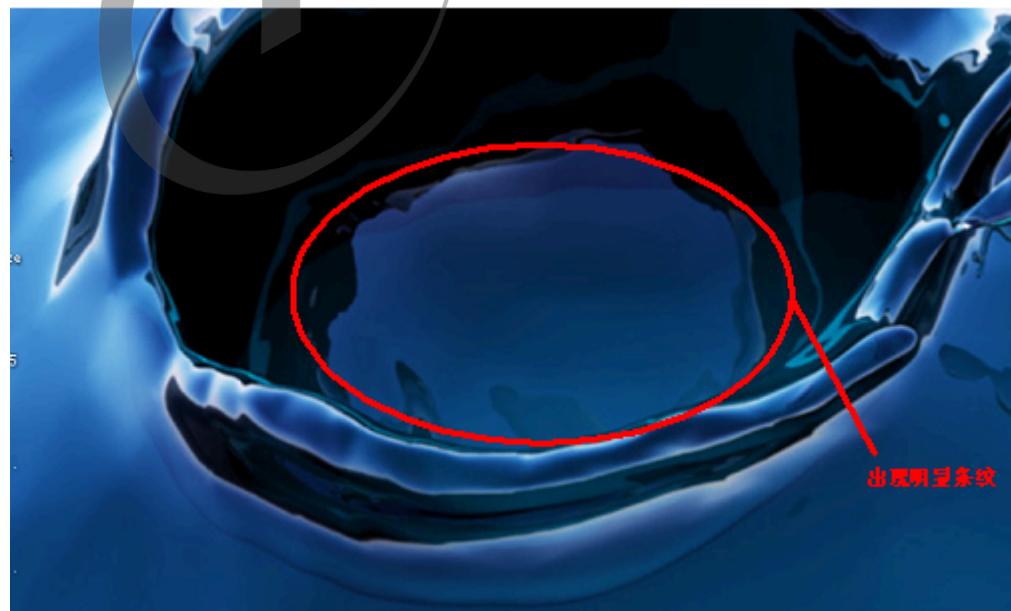


图 8-2: lcd_frm 打开

8.7 画面偏移

画面随着数据的发送偏移越来越大。

尝试修改 `lcd_dbi_clk_mode`。



9 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[编写屏驱动](#)。
2. 仔细阅读屏手册以及 driver IC 手册（有的话）。
3. 仔细阅读[lcd_fb0 配置参数详解](#)。
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 © 2021 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。