



LCD 调试使用说明书

文档版本号: SDK-V1.0

发布日期: 2019-03-30

版权所有©珠海全志科技股份有限公司 2019。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



、全志和其他全志商标均为珠海全志科技股份有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受全志公司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的购买或使用范围之内。除非合同另有约定，全志公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。



文档履历

版本号	日期	制/修订人	内容描述
V1.0	2019-03-30	Allwinner	V316 初始化版本



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	相关配置	2
2.1	menuconfig 配置说明	2
2.2	各种接口配置介绍和示例	3
2.2.1	并行 RGB 接口配置示例	3
2.2.2	串行 RGB 接口的典型配置	7
2.2.3	MIPI-DSI 高分辨率屏配置示例	10
2.2.4	MIPI-DSI VR 双屏配置示例	12
2.2.5	MIPI-DSI Command mode 屏配置示例	16
2.2.6	I8080 接口屏典型配置示例	18
2.2.7	LVDS 配置示例	21
2.2.8	RGB 和 I8080 管脚配置示意图	23
3	屏驱动源码位置	23
4	新屏驱动支持说明	25
5	硬件参数说明	26
5.1	LCD 接口参数说明	26
5.1.1	lcd_if	26

5.1.2	lcd_hv_if	26
5.1.3	lcd_hv_clk_phase	27
5.1.4	lcd_hv_sync_polarity	27
5.1.5	lcd_hv_srgb_seq	27
5.1.6	lcd_hv_syuv_seq	28
5.1.7	lcd_hv_syuv_fdly	28
5.1.8	lcd_cpu_if	29
5.1.9	lcd_cpu_te	29
5.1.10	lcd_lvds_if	30
5.1.11	lcd_lvds_colordepth	30
5.1.12	lcd_lvds_mode	30
5.1.13	lcd_dsi_if	32
5.1.14	lcd_dsi_lane	32
5.1.15	lcd_dsi_format	32
5.1.16	lcd_dsi_te	33
5.1.17	lcd_dsi_port_num	33
5.1.18	lcd_tcon_mode	34
5.1.19	lcd_slave_tcon_num	34
5.1.20	lcd_tcon_en_odd_even_div	34
5.1.21	lcd_sync_pixel_num	35
5.1.22	lcd_sync_line_num	35
5.1.23	lcd_cpu_mode	35
5.2	LCD 时序参数说明	36

5.2.1	lcd_x	36
5.2.2	lcd_y	36
5.2.3	lcd_ht	36
5.2.4	lcd_hbp	36
5.2.5	lcd_hspw	37
5.2.6	lcd_vt	37
5.2.7	lcd_vbp	37
5.2.8	lcd_vspw	38
5.2.9	lcd_dclk_freq	38
5.3	LCD 其它参数说明	38
5.3.1	lcd_width	38
5.3.2	lcd_height	39
5.3.3	lcd_pwm_used	39
5.3.4	lcd_pwm_ch	39
5.3.5	lcd_pwm_freq	39
5.3.6	lcd_pwm_pol	39
5.3.7	lcd_pwm_max_limit	40
5.3.8	lcd_rb_swap	40
5.3.9	lcd_frm	40
5.3.10	lcd_gamma_en	42
5.3.11	lcd_bl_n_percent	42
5.3.12	lcd_cmap_en	42
5.4	POWER 及 IO 说明	44

5.4.1	lcd_power	44
5.4.2	lcd_bl_en	44
5.4.3	lcd_gpio_0	45
5.4.4	lcddx	45
5.4.5	lcd_pin_power	46
6	屏驱动说明	47
6.1	屏驱动说明	47
6.1.1	开关屏流程	48
6.1.2	开关屏流程函数说明	49
6.1.3	屏驱动可使用接口说明	50
6.2	屏的初始化	52
6.2.1	MIPI DSI 屏的初始化	52
6.2.2	CPU/I80 屏的初始化	53
6.2.3	使用 IO 模拟串行接口初始化	54
6.2.4	使用 iic/spi 串行接口初始化	55
6.3	ESD 静电检测自动恢复功能	55
7	一些有用调试手段	62
7.1	查看显示信息	62
7.2	查看电源信息	62
7.3	查看 pwm 信息	63
7.4	查看管脚信息	64
7.5	查看时钟信息	64
7.6	查看接口自带 colorbar	65

8	FAQ	68
8.1	黑屏	68
8.2	闪屏	68
8.3	条形波纹	69
8.4	背光太亮或者太暗	69
8.5	重启断电测试屏异常	69
8.6	RGB 接口或者 I8080 接口显示抖动有花纹	69
9	总结	71
10	Declaration	72



表 目 录



图 目 录

2-1	menuconfig	3
2-2	RGB 管脚	4
2-3	pinmux	23
5-1	lvds mode	31
5-2	lcdht	36
5-3	lcdvt	37
5-4	good	41
5-5	bad	41
5-6	cmap	44
6-1	flow	48
6-2	menu	56
6-3	menu	57
6-4	menu	58
6-5	menu	58
6-6	menu	59
6-7	menu	60
7-1	colorbar	66

1 概述

1.1 编写目的

介绍 sunxi display2 平台中 LCD 模块中 lcd 调试方法。

1.2 适用范围

sunxi 平台 display2 模块。

1.3 相关人员

系统整合人员，显示开发相关人员，客户。

2 相关配置

2.1 menuconfig 配置说明

lcd 相关代码包含在 disp 驱动模块中，在命令行中进入内核根目录，执行 `make ARCH=arm menuconfig` 或者 `make ARCH=arm64 menuconfig` (64bit 平台) 进入配置主界面，其中。并按以下步骤操作：

具体配置目录为：

3.4 内核和 3.10 内核：

Device Drivers->Graphics support->Support for frame buffer devices->
Video Support for sunxi -> DISP Driver Support(sunxi-disp2)

如果是 linux-4.9 及其以上路径是：

Device Drivers->Graphics support->Framebuffer Devices >
Video Support for sunxi -> DISP Driver Support(sunxi-disp2)

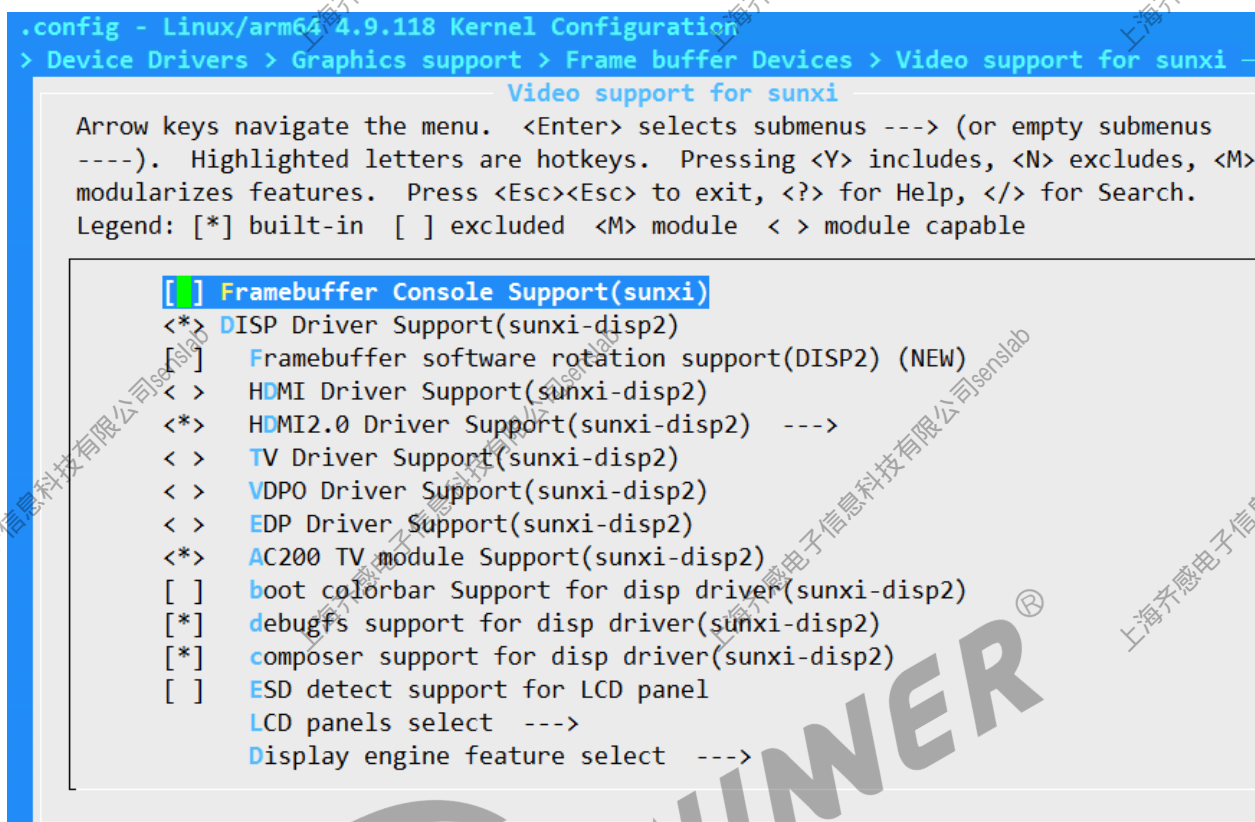


图 2-1: menuconfig

2.2 各种接口配置介绍和示例

下面介绍全志平台的各种 LCD 接口以及配置示例，至于配置示例的细节请看第五章。

2.2.1 并行 RGB 接口配置示例

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

和 RGB 相关的管脚有：

Table 1-1.HV Panel Signals

Signal	Description	Type
Vsync	Vertical sync, indicates one new frame	O
Hsync	Horizontal sync, indicate one new scan line	O
DCLK	Dot clock, pixel data are sync by this clock	O
DE	LCD data enable	O
D[23..0]	24Bit RGB output from input FIFO for panel	O

图 2-2: RGB 管脚

其中数据脚不一定是 24 根，有的 SOC 只支持 RGB666，也就是只有 18 根，以 SOC 规格文档说明为准。

上面这些脚具体到 SOC 哪根管脚以及第几个功能（管脚复用功能）请参考 pin mux 表格，管脚复用功能的名字一般以“LCDX_”开头，其中 X 是数字。

RGB 接口有两种同步方式，一种是 Hsync+Vsync，一种是 DE（Data Enable），请根据您的 LCD 屏支持情况，在硬件连接好这些管脚。

HV 接口又细分几种接口，通过设置 lcd_hv_if 来选择。

RGB 的协议本身不支持除了数据之外的传输，所以无法通过 RGB 管脚进行对 LCD 屏进行配置，所以拿到一款 RGB 接口屏，要么不需要配置，要么这个屏会提供额外的管脚给 SOC 来进行配置，比如 SPI，比如 I2C。

典型并行 RGB 接口 sys_config.fex 配置示例：

1. 红色部分决定了 SOC 中的 LCD 模块发送时序，这部分参数需要从屏手册中获取或者询问屏厂。
2. 粉红色决定下面的配置是一个并行 RGB 的配置。
3. 蓝色部分决定了背光（pwm 和 lcd_bl_en），SOC 模块供电，管脚供电，以及屏供电，管脚配置。请对照请看原理图。
4. lcd_driver_name 决定了用哪个屏驱动来初始化，这里是 default_lcd，是针对不需要初始化设置的 RGB 屏。
5. 绿色部分对于 RGB 接口也是非常有必要，除非你的产品不需要考虑关屏时的功耗。

```
[lcd0]
lcd_used = 1
```

```

lcd_driver_name    = "default_lcd"
lcd_if             = 0
lcd_hv_if         = 0
lcd_x             = 800
lcd_y             = 480
lcd_width         = 150
lcd_height        = 94
lcd_dclk_freq     = 33
lcd_pwm_used      = 1
lcd_pwm_ch        = 8
lcd_pwm_freq      = 10000
lcd_pwm_pol       = 1
lcd_hbp           = 46
lcd_ht            = 1055
lcd_hspw          = 0
lcd_vbp           = 23
lcd_vt            = 525
lcd_vspw          = 0
lcd_frm           = 0
lcd_io_phase      = 0x0000
lcd_gamma_en      = 0
lcd_bright_curve_en = 0
lcd_cmap_en       = 0

deu_mode          = 0
lcdgamma4iep      = 22
smart_color       = 90

;lcd_bl_en        = port:PH06<1><0><default><1>
lcd_power         = "vcc-lcd"

lcd_pin_power     = "vcc-pd"

lcdd2             = port:PD00<2><0><default><default>
lcdd3             = port:PD01<2><0><default><default>
lcdd4             = port:PD02<2><0><default><default>

```

```

lcedd5      = port:PD03<2><0><default><default>
lcedd6      = port:PD04<2><0><default><default>
lcedd7      = port:PD05<2><0><default><default>

lcedd10     = port:PD06<2><0><default><default>
lcedd11     = port:PD07<2><0><default><default>
lcedd12     = port:PD08<2><0><default><default>
lcedd13     = port:PD09<2><0><default><default>
lcedd14     = port:PD10<2><0><default><default>
lcedd15     = port:PD11<2><0><default><default>

lcedd18     = port:PD12<2><0><default><default>
lcedd19     = port:PD13<2><0><default><default>
lcedd20     = port:PD14<2><0><default><default>
lcedd21     = port:PD15<2><0><default><default>
lcedd22     = port:PD16<2><0><default><default>
lcedd23     = port:PD17<2><0><default><default>

lcedclk     = port:PD18<2><0><default><default>
lcedhs      = port:PD20<2><0><default><default>
lcedvs      = port:PD21<2><0><default><default>
lced_de     = port:PD19<2><0><default><default>

;for rgb24 please uncomment following line
;lcedd0      = port:PE16<3><0><default><default>
;lcedd1      = port:PE17<3><0><default><default>
;lcedd16     = port:PE20<3><0><default><default>
;lcedd17     = port:PE21<3><0><default><default>
;lcedd8      = port:PE18<3><0><default><default>
;lcedd9      = port:PE19<3><0><default><default>
;
[lcd0_suspend]
lcedd2      = port:PD00<7><0><default><default>
lcedd3      = port:PD01<7><0><default><default>
lcedd4      = port:PD02<7><0><default><default>
lcedd5      = port:PD03<7><0><default><default>

```



```

lcedd6      = port:PD04<7><0><default><default>
lcedd7      = port:PD05<7><0><default><default>

lcedd10     = port:PD06<7><0><default><default>
lcedd11     = port:PD07<7><0><default><default>
lcedd12     = port:PD08<7><0><default><default>
lcedd13     = port:PD09<7><0><default><default>
lcedd14     = port:PD10<7><0><default><default>
lcedd15     = port:PD11<7><0><default><default>

lcedd18     = port:PD12<7><0><default><default>
lcedd19     = port:PD13<7><0><default><default>
lcedd20     = port:PD14<7><0><default><default>
lcedd21     = port:PD15<7><0><default><default>
lcedd22     = port:PD16<7><0><default><default>
lcedd23     = port:PD17<7><0><default><default>

lcedclk     = port:PD18<7><0><default><default>
lcdhs       = port:PD20<7><0><default><default>
lcdvs       = port:PD21<7><0><default><default>
lcd_de      = port:PD19<7><0><default><default>

for rgb24 please uncomment following line
;lcedd0     = port:PE16<7><0><default><default>
;lcedd1     = port:PE17<7><0><default><default>
;lcedd16    = port:PE20<7><0><default><default>
;lcedd17    = port:PE21<7><0><default><default>
;lcedd8     = port:PE18<7><0><default><default>
;lcedd9     = port:PE19<7><0><default><default>

```

2.2.2 串行 RGB 接口的典型配置

串行 RGB 需要通过几个时钟周期才能发完一个 pixel 的所有字节。

1. 红色部分决定了 SOC 中的 LCD 模块发送时序，这部分参数需要从屏手册中获取或者

询问屏厂。lcd_fm 的设置对于 RGB666 的情况非常有必要。

2. 粉红色决定下面的配置是一个并行 RGB 的配置。
3. 蓝色部分决定了背光 (pwm 和 lcd_bl_en)，SOC 模块供电，管脚供电，以及屏供电，管脚配置。请对照请看原理图。
4. lcd_driver_name 决定了用哪个屏驱动来初始化，这里是 stv7789v，是需要初始化，初始化的接口协议是 SPI，所以第三点中多了几根 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）。
5. 绿色部分对于 RGB 接口也是非常有必要，除非你的产品不需要考虑关屏时的功耗。
6. SOC 总共需要三个周期才能发完一个 pixel，所以 $\text{lcd_dclk_freq} * 3 = \text{lcd_ht} * \text{lcd_vt} * 60$ ，或者 $\text{lcd_dclk_freq} = \text{lcd_ht} * 3 * \text{lcd_vt} * 60$ 要么增加 lcd_ht 要么增加 lcd_dclk_freq

```
[lcd0]
lcd_used          = 1

lcd_driver_name   = "stv7789v"
lcd_x             = 240
lcd_y             = 320
lcd_width        = 108
lcd_height       = 64
lcd_dclk_freq     = 19

lcd_pwm_used      = 1
lcd_pwm_ch        = 8
lcd_pwm_freq      = 50000
lcd_pwm_pol       = 1
lcd_pwm_max_limit = 255

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
```

```

lcd_frm          = 1
lcd_if           = 0
lcd_hv_if        = 8
lcd_hv_clk_phase = 0
lcd_hv_sync_polarity = 0
lcd_hv_srgb_seq  = 0

lcd_io_phase     = 0x0000
lcd_gamma_en     = 0
lcd_bright_curve_en = 1
lcd_cmap_en      = 0

lcd_rb_swap      = 0

deu_mode         = 0
lcdgamma4iep     = 22
smart_color      = 90

;lcd_bl_en       = port:PB03<1><0><default><1>
lcd_power        = "vcc-lcd"
lcd_pin_power    = "vcc-pd"

;reset
lcd_gpio_0       = port:PD09<1><0><default><1>
;cs
lcd_gpio_1       = port:PD10<1><0><default><0>
;sda
lcd_gpio_2       = port:PD13<1><0><default><0>
;sck
lcd_gpio_3       = port:PD12<1><0><default><0>

lcdd5            = port:PD03<2><0><2><default>
lcdd6            = port:PD04<2><0><2><default>
lcdd7            = port:PD05<2><0><2><default>
lcdd10           = port:PD06<2><0><2><default>

```

```

lcedd11      = port:PD07<2><0><2><default>
lcedd12      = port:PD08<2><0><2><default>

lcedclk      = port:PD18<2><0><3><default>
lcedde       = port:PD19<2><0><3><default>
lcedhsync    = port:PD20<2><0><3><default>
lcedvsync    = port:PD21<2><0><3><default>

[lcd0_suspend]

lcedd5       = port:PD03<7><0><2><default>
lcedd6       = port:PD04<7><0><2><default>
lcedd7       = port:PD05<7><0><2><default>
lcedd10      = port:PD06<7><0><2><default>
lcedd11      = port:PD07<7><0><2><default>
lcedd12      = port:PD08<7><0><2><default>

lcedclk      = port:PD18<7><0><3><default>
lcedde       = port:PD19<7><0><3><default>
lcedhsync    = port:PD20<7><0><3><default>
lcedvsync    = port:PD21<7><0><3><default>

```

2.2.3 MIPI-DSI 高分辨率屏配置示例

1. MIPI-DSI 的管脚是专用，在 sys_config.fex 不需要配置，硬件上连好就行。
2. 根据分辨率的高低通常分为几种模式来配置。1080p 分辨率及其以下：只需要设置 lcd_dsi_if 来控制就行。Command mode 一般是低分辨率屏，而 video mode 和 burst mode 则是用于高分辨率的。如果分辨率达到 2k（目前只有 a63 和 vr9 支持）则需要额外的另外的设置。

超高分辨率（大于 2k）典型配置：

1. 该屏实际上需要 8 条 lane 才能驱动，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。此时下面蓝色下划线三个选项需要特别设置，点击查看具体含义。L

2. lcd_dsi_lane 依旧设置成 4 条 lane 的原因，是因为这个是设置一个 dsi 的 lane 数量，这个屏要用两个 dsi。加起来就是 8 条 lane。
3. 红色部分是屏的时序。请参考屏手册和询问屏厂。
4. 粉红色部分决定了下面配置 DSI 屏的配置，而且是 video mode（高分辨率不可能是 command mode）
5. 蓝色部分是 pwm 背光，复位脚，电源使能脚，背光使能脚，DSI 模块供电，管脚供电，屏供电等。
6. 如果是 1080p 及其以下分辨率的屏，那么蓝色下划线三个配置默认 0 即可。

```
[lcd0]
lcd_used      = 1

lcd_driver_name = "iq101r1sx03"

lcd_backlight  = 50
lcd_if         = 4
lcd_x          = 2560
lcd_y          = 1600
lcd_width     = 216
lcd_height    = 135
lcd_dclk_freq  = 268
lcd_pwm_used   = 1
lcd_pwm_ch     = 0
lcd_pwm_freq   = 50000
lcd_pwm_pol    = 1
lcd_pwm_max_limit = 255

lcd_hbp       = 80
lcd_ht        = 2720
lcd_hspw      = 32
lcd_vbp       = 37
lcd_vt        = 1646
lcd_vspw      = 6
```

```

lcd_dsi_if      = 0
lcd_dsi_lane    = 4
lcd_dsi_format  = 0
lcd_dsi_te      = 0
lcd_dsi_eotp    = 0

lcd_frm        = 0
lcd_io_phase    = 0x0000
lcd_hv_clk_phase = 0
lcd_hv_sync_polarity = 0
lcd_gamma_en    = 0
lcd_bright_curve_en = 0
lcd_cmap_en     = 0

lcdgamma4iep    = 22

lcd_dsi_port_num = 1
lcd_tcon_mode    = 4
lcd_tcon_en_odd_even_div = 1

lcd_bl_en       = port:PH10<1><0><default><1>
lcd_power       = "vcc18-lcd"
lcd_power1      = "vcc33-lcd"
lcd_pin_power   = "vcc-pd"

lcd_gpio_0      = port:PH11<1><0><default><1>
lcd_gpio_1      = port:PH12<1><0><default><1>

```

2.2.4 MIPI-DSI VR 双屏配置示例

实际场景是两个物理屏，每个屏是 1080p，每个屏都是 4 条 lane，要求的是两个屏各自显示一帧图像的左右一半，由于宽高比和横竖屏以及 DE 处理能力的因素，一个 DE+ 一个 tcon+ 两个 DSI 已经无法满足，必须用两个 tcon 各自驱动一个 dsi，但是两路显示必须要同步，这就需要用到两个 tcon 的同步模式。

1. LCD0 标记为 **slave tcon**，它由 **master tcon** 来驱动。设置下面蓝色有下划线的属性：
`lcd_tcon_mode`。
2. LCD1 标记为 **master tcon**，并且负责两个屏的所有电源，背光，管脚的开关。
3. LCD0 先开，对应模块寄存器都初始化，但是电源不开，然后开 LCD1，LCD1 使能就会触发 LCD0 一起发数据。这样做到同时亮灭。

```
;slave
[lcd0]
lcd_used      = 1

lcd_driver_name  = "lpm025m475a"
;lcd_bl_0_percent  = 0
;lcd_bl_40_percent = 23
;lcd_bl_100_percent = 100

lcd_backlight   = 50
lcd_if          = 4
lcd_x           = 1080
lcd_y           = 1920
lcd_width       = 31
lcd_height      = 56
lcd_dclk_freq   = 141

lcd_pwm_used    = 0
lcd_pwm_ch      = 0
lcd_pwm_freq    = 20000
lcd_pwm_pol     = 0
lcd_pwm_max_limit = 255

lcd_hbp         = 100
lcd_ht          = 1212
lcd_hspw        = 5
lcd_vbp         = 8
lcd_vt          = 1936
lcd_vspw        = 2
```



```
lcd_dsi_if      = 0
lcd_dsi_lane    = 4
lcd_dsi_format  = 0
lcd_dsi_te      = 0
lcd_dsi_eotp    = 0

lcd_frm        = 0
lcd_io_phase    = 0x0000
lcd_hv_clk_phase = 0
lcd_hv_sync_polarity = 0
lcd_gamma_en    = 0
lcd_bright_curve_en = 0
lcd_cmap_en     = 0

lcdgamma4iep    = 22

lcd_dsi_port_num = 0
lcd_tcon_mode   = 3
lcd_slave_stop_pos = 0
lcd_syne_pixel_num = 0
lcd_sync_line_num = 0

lcd_io_regulator1 = "vcc-ph"

[lcd1]
lcd_used        = 1

lcd_driver_name  = "lpm025m475a"
;lcd_bl_0_percent = 0
;lcd_bl_40_percent = 23
;lcd_bl_100_percent = 100

lcd_backlight    = 50
lcd_if           = 4
lcd_x            = 1080
```



```
lcd_y          = 1920
lcd_width      = 31
lcd_height     = 56
lcd_dclk_freq  = 141

lcd_pwm_used   = 1
lcd_pwm_ch     = 0
lcd_pwm_freq   = 20000
lcd_pwm_pol    = 0
lcd_pwm_max_limit = 255

lcd_hbp        = 100
lcd_ht         = 1212
lcd_hspw       = 5
lcd_vbp        = 8
lcd_vt         = 1936
lcd_vspw       = 2

lcd_dsi_if     = 0
lcd_dsi_lane   = 4
lcd_dsi_format = 0
lcd_dsi_te     = 0
lcd_dsi_eotp   = 0

lcd_frm        = 0
lcd_io_phase   = 0x0000
lcd_hv_clk_phase = 0
lcd_hv_sync_polarity = 0
lcd_gamma_en   = 0
lcd_bright_curve_en = 0
lcd_cmap_en    = 0

lcdgamma4iep   = 22

lcd_dsi_port_num = 0
lcd_tcon_mode  = 1
```

```

lcd_tcon_slave_num = 0
lcd_slave_stop_pos = 0
lcd_sync_pixel_num = 0
lcd_sync_line_num = 0

lcd_bl_en          = port:PH10<1><0><default><1>
lcd_power           = "vcc-dsi"
lcd_power1          = "vcc18-lcd"
lcd_power2          = "vcc33-lcd"

lcd_gpio_0          = port:PH8<1><0><default><1>
lcd_gpio_1          = port:PH11<1><0><default><1>
lcd_gpio_2          = port:PH12<1><0><default><1>
;lcd_gpio_3         = port:PB4<1><0><default><0>
;lcd_gpio_4         = port:PB5<1><0><default><0>
;lcd_gpio_5         = port:PB6<1><0><default><0>
;lcd_gpio_2         = port:PH11<1><0><default><0>
;lcd_gpio_3         = port:PH10<1><0><default><0>
lcd_io_regulator1   = "vcc-ph"

```

2.2.5 MIPI-DSI Command mode 屏配置示例

Command mode 下的 DSI 屏类似与 I8080 接口，都是内部带 RAM 用于缓冲，这种情况一般都需要用屏的 te 脚来触发 vsync 中断，所以与其它类型的 DSI 屏不同的是，这里需要设置 lcd_vsync 脚，屏的 te 脚就连到 lcd_vsync 上，并且 lcd_dsi_te 设置成 1。

lcd_dsi_if 设置成 1 标明 command mode

```

[lcd0]
lcd_used          = 1

lcd_driver_name    = "h245qbn02"

lcd_backlight      = 100

```

```
lcd_if      = 4
lcd_x       = 240
lcd_y       = 432
lcd_width   = 52
lcd_height  = 52
lcd_dclk_freq = 18

lcd_pwm_used = 1
lcd_pwm_ch   = 0
lcd_pwm_freq = 50000
lcd_pwm_pol  = 1
lcd_pwm_max_limit = 255

lcd_hbp      = 96
lcd_ht       = 480
lcd_hspw     = 2
lcd_vbp      = 21
lcd_vt       = 514
lcd_vspw     = 2

lcd_dsi_if   = 1
lcd_dsi_lane = 1
lcd_dsi_format = 0
lcd_dsi_te   = 1
lcd_dsi_eotp = 0

lcd_frm      = 0
lcd_io_phase = 0x0000
lcd_hv_clk_phase = 0
lcd_hv_sync_polarity = 0
lcd_gamma_en = 0
lcd_bright_curve_en = 0
lcd_cmap_en  = 0

lcdgamma4iep = 22
```

```
lcd_bl_en      = port:PB03<1><0><default><1>
lcd_power      = "axp233_dc1sw"
lcd_power1     = "axp233_eldo1"

lcd_gpio_0     = port:PB02<1><0><default><0>
lcd_vsync      = port:PD21<2><0><3><default>
```

2.2.6 I8080 接口屏典型配置示例

Intel 8080 接口屏 (又称 MCU 接口) 很老的协议, 一般用在分辨率很小的屏上。

管脚的控制脚有 5 个:

- CS 片选信号, 决定该芯片是否工作。
- RS 寄存器选择信号, 低表示选择 index 或者 status 寄存器, 高表示选择控制寄存器。
实际场景中一般接 SOC 的 LCD_DE 脚 (数据使能脚)
- /WR (低表示写数据) 数据命令区分信号, 也就是写时钟信号, 一般接 SOC 的 LCD_CLK 脚
- /RD (低表示读数据) 数据读信号, 也就是读时钟信号, 一般接 SOC 的 LCD_HSYNC 脚
- RESET 复位 LCD (用固定命令系列 010 来复位)
- Data 是双向的

I8080 根据的数据位宽接口有 8/9/16/18, 连哪些脚参考, 即使位宽一样, 连的管脚也不一样, 还要考虑的因素是 rgb 格式

1. RGB565, 也就有 65K 这么多种颜色
2. RGB666, 也就是有 262K 那么多种颜色
3. 9bit 固定为 262K

从屏手册得知: 数据位宽, 颜色数量之和, 参考RGB 和 I8080 管脚配置示意图, 进行硬件连接。

下面的配置是一个 RGB565 的, 位宽为 8 位的 I8080 接口的屏。

1. 它需要两个周期才把一个像素发完，所以，设置像素时钟要满足以下公式：
 $\text{lcd_dclk_freq} * 2 \geq \text{lcd_ht} * \text{lcd_vt} * \text{fps}$ ，或者 $\text{lcd_dclk_freq} = \text{lcd_ht} * 2 * \text{lcd_vt} * 60$ ，也就是要么双倍 lcd_ht 要么双倍 lcd_dclk_freq
2. 粉红色的属性决定以下配置 I8080 配置，红色是时序，参考屏手册或者询问屏厂。
3. 蓝色是电源，管脚，如果一旦确定 SOC 型号，屏位宽以及 RGB 各式，那么除了复位脚和片选脚之外都是固定的。
4. `lcd0_suspend` 是必要的，除非你们不用考虑关屏功耗。

```
[lcd0]
lcd_used      = 1
lcd_driver_name = "s2003t46g"
lcd_if        = 1
lcd_x         = 240
lcd_y         = 320
lcd_width     = 108
lcd_height    = 64
lcd_dclk_freq = 16
lcd_pwm_used  = 1
lcd_pwm_ch    = 8
lcd_pwm_freq  = 50000
lcd_pwm_pol   = 1
lcd_pwm_max_limit = 255
lcd_hbp       = 20
lcd_ht        = 298
lcd_hspw      = 10
lcd_vbp       = 8
lcd_vt        = 336
lcd_vspw      = 4
lcd_frm       = 1
lcd_gamma_en  = 0
lcd_bright_curve_en = 1
lcd_cmap_en   = 0
lcd_cpu_mode  = 1
lcd_cpu_te    = 1
lcd_cpu_if    = 14
```

```

lcd_rb_swap      = 0

lcdgamma4iep     = 22

lcd_power        = "vcc-lcd"
lcd_pin_power    = "vcc-pd"

;reset pin
lcd_gpio_0       = port:PD09<1><0><2><1>
;cs pin
lcd_gpio_1       = port:PD10<1><0><2><0>

lcdd3            = port:PD1<2><0><2><default>
lcdd4            = port:PD2<2><0><2><default>
lcdd5            = port:PD3<2><0><2><default>
lcdd6            = port:PD4<2><0><2><default>
lcdd7            = port:PD5<2><0><2><default>
lcdd10           = port:PD6<2><0><2><default>
lcdd11           = port:PD7<2><0><2><default>
lcdd12           = port:PD8<2><0><2><default>
lcdclk           = port:PD18<2><0><3><default>
lcdde            = port:PD19<2><0><3><default>
lcdhsync         = port:PD20<2><0><3><default>
lcdvsync         = port:PD21<2><0><3><default>

[lcd0_suspend]
lcdd3            = port:PD1<7><0><2><default>
lcdd4            = port:PD2<7><0><2><default>
lcdd5            = port:PD3<7><0><2><default>
lcdd6            = port:PD4<7><0><2><default>
lcdd7            = port:PD5<7><0><2><default>
lcdd10           = port:PD6<7><0><2><default>
lcdd11           = port:PD7<7><0><2><default>
lcdd12           = port:PD8<7><0><2><default>
lcdclk           = port:PD18<7><0><3><default>
lcdde            = port:PD19<7><0><3><default>
lcdhsync         = port:PD20<7><0><3><default>

```

```
lcdvsync      = port:PD21<7><0><3><default>
```

2.2.7 LVDS 配置示例

LVDS 接口，lcd0 对应的 lvds 管脚和 lcd1 对应的 lvds 管脚是固定而且不一样。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”。

如果 Dual Link 的屏，那么除了要改 lcd_lvds_if 为 1 之外，管脚方面还要把 lcd1 的管脚一起搬到下面去，也就是总共需要配置 PD0 到 PD9，和配置 PD10 到 PD19 总共二十根脚为 lvds 管脚功能（功能 3）。当然屏的 timing 也是要根据屏来改的。

```
[lcd0]
lcd_used      = 1

lcd_driver_name = "default_lcd"
lcd_backlight  = 50
lcd_if        = 3
lcd_x         = 1280
lcd_y         = 800
lcd_width     = 150
lcd_height    = 94
lcd_dclk_freq = 70
lcd_pwm_used  = 1
lcd_pwm_ch    = 0
lcd_pwm_freq  = 50000
lcd_pwm_pol   = 0
lcd_pwm_max_limit = 255
lcd_hbp       = 20
lcd_ht        = 1418
lcd_hspw      = 10
lcd_vbp       = 10
lcd_vt        = 814
lcd_vspw      = 5
lcd_lvds_if   = 0
```



```

lcd_lvds_colordepth = 1
lcd_lvds_mode      = 0
lcd_frm           = 1
lcd_hv_clk_phase   = 0
lcd_hv_sync_polarity = 0
lcd_gamma_en       = 0
lcd_bright_curve_en = 0
lcd_cmap_en        = 0

deu_mode           = 0
lcdgamma4iep       = 22
smart_color        = 90

lcd_bl_en          = port:PD21<1><0><default><1>
lcd_power           = "vcc-lcd"

lcdd0              = port:PD00<3><0><default><default>
Lcdd1              = port:PD01<3><0><default><default>
Lcdd2              = port:PD02<3><0><default><default>
Lcdd3              = port:PD04<3><0><default><default>
Lcdd4              = port:PD05<3><0><default><default>
Lcdd5              = port:PD06<3><0><default><default>
Lcdd6              = port:PD07<3><0><default><default>
Lcdd8              = port:PD08<3><0><default><default>
Lcdd9              = port:PD09<3><0><default><default>

[lcd0_suspend]
lcdd0              = port:PD00<7><0><default><default>
Lcdd1              = port:PD01<7><0><default><default>
Lcdd2              = port:PD02<7><0><default><default>
Lcdd3              = port:PD04<7><0><default><default>
Lcdd4              = port:PD05<7><0><default><default>
Lcdd5              = port:PD06<7><0><default><default>
Lcdd6              = port:PD07<7><0><default><default>
Lcdd8              = port:PD08<7><0><default><default>
Lcdd9              = port:PD09<7><0><default><default>

```


2.2.8 RGB 和 I8080 管脚配置示意图

I/O	SYNC RGB				CPU ¹ cmd ¹ 18bit	CPU ¹ 16bit								CPU ¹ 8bit				CPU ¹ 9bit							
	1 st	2 nd	3 rd	4 th	256K	256K				65K				256K		65K		256K		1 st	2 nd	3 rd	4 th	5 th	6 th
						1 st	2 nd	3 rd	4 th	1 st	2 nd	1 st	2 nd	1 st	2 nd	1 st	2 nd								
IO0	VSYNC					CS																			
IO1	HSYNC					RD																			
IO2	DCLK					WR																			
IO3	DE					RS																			
D23	R7				D23	R5	B5	G5	R5		R5	B5	R4												
D22	R6				D22	R4	B4	G4	R4		R4	B4	R3												
D21	R5				D21	R3	B3	G3	R3		R3	B3	R2												
D20	R4				D20	R2	B2	G2	R2		R2	B2	R1												
D19	R3				D19	R1	B1	G1	R1		R1	B1	R0												
D18	R2				D18	R0	B0	G0	R0		R0	B0	G5												
D17	R1				D17																				
D16	R0				D16																				
D15	G7				D15	G5							G4												
D14	G6				D14	G4							G3												
D13	G5				D13	G3																			
D12	G4	D17	D27	D37	D7	D12	G2	G5	R5	B5	G5	B5	G5	G2	R5	G5	B5	R4	G2	R5	G2				
D11	G3	D16	D26	D36	D6	D11	G1	G4	R4	B4	G4	B4	G4	G1	R4	G4	B4	R3	G1	R4	G1				
D10	G2	D15	D25	D35	D5	D10	G0	G3	R3	B3	G3	B3	G3	G0	R3	G3	B3	R2	G0	R3	G0				
D9	G1				D9																				
D8	G0				D8																				
D7	B7	D14	D24	D34	D4	D7	B5	G2	R2	B2	G2	B2	G2	B4	R2	G2	B2	R1	B4	R2	B5				
D6	B6	D13	D23	D33	D3	D6	B4	G1	R1	B1	G1	B1	G1	B3	R1	G1	B1	R0	B3	R1	B4				
D5	B5	D12	D22	D32	D2	D5	B3	G0	R0	B0	G0	B0	G0	B2	R0	G0	B0	G5	B2	R0	B3				
D4	B4	D11	D21	D31	D1	D4	B2							B1				G4	B1	G5	B2				
D3	B3	D10	D20	D30	D0	D3	B1							B0				G3	B0	G4	B1				
D2	B2				D2	B0													G3	B0					
D1	B1				D1																				
D0	B0				D0																				

图 2-3: pinmux

3

屏驱动源码位置

3.4 版本内核:

linux3-4/drivers/video/sunxi/disp2/disp/lcd/

3.10 版本内核:

linux3-10/drivers/video/sunxi/disp2/disp/lcd/

4.9 版本内核:

linux-4.9/drivers/video/fbdev/sunxi/disp2/disp/lcd/

uboot-2014:

brandy/u-boot-2014.07/drivers/video/sunxi/disp2/disp/lcd



4

新屏驱动支持说明

在 LCD 源码目录（看第三章）下拷贝现有一个屏驱动，根据屏接口类型（LVDS，DSI，RGB 等）选择一个合适模板。

1. 如果是 LVDS 接口和不需要初始化命令的 RGB 接口的屏，那么可以直接用 default_panel.c 驱动即可，不需要新增文件，在 sys_config.fex 中修改 lcd_driver_name 为 default_lcd，然后根据屏手册修改其它时序参数。如果是 DSI 接口，可以参考 inet_dsi_panel.c 或者 WilliamLCD.c；如果是 I8080 接口（mcu 接口或者 cpu 接口）可以参考 cpu_gg1p4062utsw.c。
2. 如果需要新增文件，那么选择好模板文件之后，修改 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 sys_config.fex 中 [lcd0] 的 lcd_driver_name 一致。
3. 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。
4. 修改 Makefile。在 lcd 屏驱动目录的上一级在 disp-objs 中新增刚才添加屏驱动.o
5. 以上步骤，也必须在 uboot（看第三章）中完成，否则将无法在一秒内显示 logo，uboot 显示驱动的源码组织架构和 api 和内核的一致。编译 uboot 的方式请看《Uboot 调试说明书》，记住默认是不重新编译 uboot 的！
6. 根据屏手册修改 sys_config.fex 中的 [lcd0] 节点下面的属性，各个属性的含义请看第五章，记住必须严格按照屏手册上的时序来填，如果超过屏手册的所限定的范围将有可能导致花屏，闪屏等异常，长期有可能导致 LCD 屏损坏！

5

硬件参数说明

5.1 LCD 接口参数说明

5.1.1 lcd_if

Lcd Interface

设置相应值的对应含义为：

:

0HV 接口RGB:

1CPU/接口I80:

2Reserved:

3接口LVDS:

4接口DSI

5.1.2 lcd_hv_if

Lcd HV panel Interface

这个参数只有在 lcd_if=0 时才有效。定义 RGB 同步屏下的几种接口类型。

设置相应值的对应含义为：

:

0Parallel RGB:

8Serial RGB:

10Dummy RGB:

11RGB Dummy:

12Serial YUV (CCIR656)

5.1.3 lcd_hv_clk_phase

Lcd HV panel Clock Phase

这个参数只有在 lcd_if=0 时才有效。定义 RGB 同步屏的 clock 与 data 之间的相位关系。总共有 4 个相位可供调节。

设置相应值的对应含义为：

- 0: 0 degree
- 1: 90 degree
- 2: 180 degree
- 3: 270 degree

5.1.4 lcd_hv_sync_polarity

Lcd HV panel Sync signals Polarity

这个参数只有在 lcd_if=0 时才有效。定义 RGB 同步屏的 hsync 和 vsync 的极性。

设置相应值的对应含义为：

- 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;

5.1.5 lcd_hv_srgb_seq

Lcd HV panel Serial RGB output Sequence

这个参数只有在 lcd_if=0 且 lcd_hv_if=1 (Serial RGB) 时才有效。

定义奇数行 RGB 输出的顺序：

0: Odd lines R-G-B; Even line R-G-B
1: Odd lines B-R-G; Even line R-G-B
2: Odd lines G-B-R; Even line R-G-B
4: Odd lines R-G-B; Even line B-R-G
5: Odd lines B-R-G; Even line B-R-G
6: Odd lines G-B-R; Even line B-R-G
8: Odd lines R-G-B; Even line B-R-G
9: Odd lines B-R-G; Even line G-B-R
10: Odd lines G-B-R; Even line G-B-R

5.1.6 lcd_hv_syuv_seq

Lcd HV panel Serial YUV output Sequence

这个参数只有在 lcd_if=0 且 lcd_hv_if=2 (Serial YUV) 时才有效。

定义 YUV 输出格式:

:
0YUYV:
1YVYU:
2UYVY:
3VYUY

5.1.7 lcd_hv_syuv_fdly

Lcd HV panel Serial YUV F line Delay

这个参数只有在 lcd_if=0 且 lcd_hv_if=2 (Serial YUV) 时才有效。

定义 CCIR656 编码时 F 相对有效行延迟的行数:

:
OF toggle right after active video line:

1Delay 2 lines (CCIR NTSC):
2Delay 3 lines (CCIR PAL)

5.1.8 lcd_cpu_if

Lcd CPU panel Interface

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：

:
018bit/1cycle parallel (RGB666):
416bit/1cycle parallel (RGB565):
618bit/3cycle parallel (RGB666):
816bit/2cycle parallel (RGB565):
109bit/1cycle (RGB666):
128bit/3cycle (RGB666):
148bit/2cycle(RGB565)

5.1.9 lcd_cpu_te

Lcd CPU panel tear effect

设置相应值的对应含义为，设置为 0 时，刷屏间隔时间为 $\text{lcd_ht} \times \text{lcd_vt}$ ；设置为 1 或 2 时，刷屏间隔时间为两个 te 脉冲：

:
0frame triggered automatically:
1frame triggered by te rising edge:
2frame triggered by te falling edge

5.1.10 lcd_lvds_if

Lcd LVDS panel Interface

设置相应值的对应含义为：

:
0Single Link(1 clock pair+3/4 data pair):
1Dual Link(2 clock pair)

5.1.11 lcd_lvds_colordepth

Lcd LVDS panel color depth

设置相应值对应含义为：

:
08bit per color(4 data pair):
16bit per color(3 data pair)

5.1.12 lcd_lvds_mode

Lcd LVDS Mode

这个参数只有在 lcd_lvds_bitwidth=0 时才有效

设置相应值对应含义为 (见下图)：

:
0NS mode:
1JEIDA mode

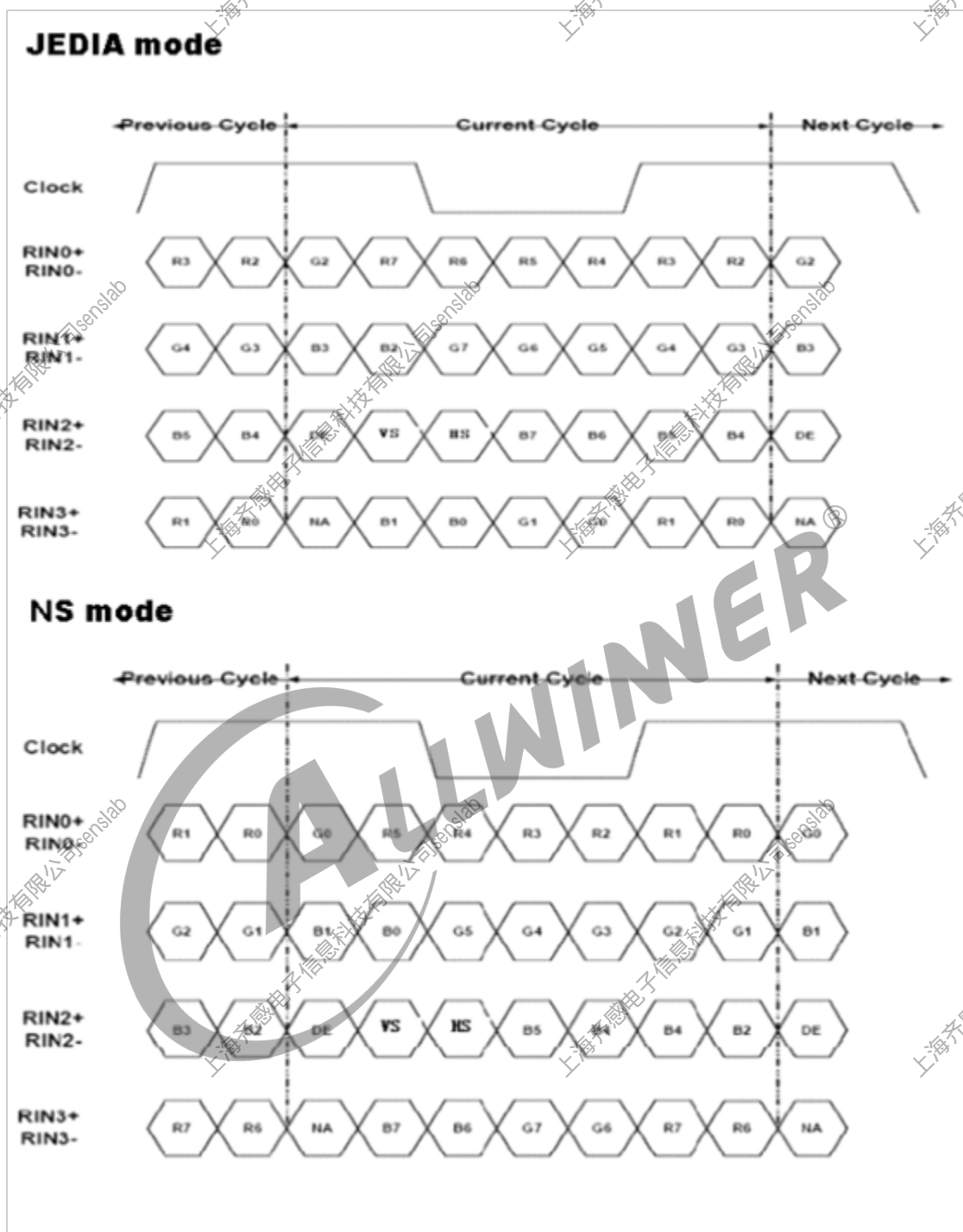


图 5-1: lvds mode

5.1.13 lcd_dsi_if

Lcd MIPI DSI panel Interface

这个参数只有在 lcd_if=4 时才有效。定义 MIPI DSI 屏的两种类型。

设置相应值的对应含义为：

：
0Video mode:
1Command mode:
2video burst mode

注：Video mode 的 LCD 屏，是实时刷屏的，有 ht, hbp 等时序参数的定义；Command mode 的屏，屏上带有显示 Buffer，一般会有一个 TE 引脚

5.1.14 lcd_dsi_lane

Lcd MIPI DSI panel Data Lane number

这个参数只有在 lcd_if=4 时才有效。

设置相应值的对应含义为：

：
11 data lane:
22 data lane:
33 data lane:
44 data lane

5.1.15 lcd_dsi_format

Lcd MIPI DSI panel Data Pixel Format

这个参数只有在 lcd_if=4 时才有效。

设置相应值的对应含义为：

：
0Package Pixel Stream, 24bit RGB:
1Loosely Package Pixel Stream, 18bit RGB:
2Package Pixel Stream, 18bit RGB:
3Package Pixel Stream, 16bit RGB

5.1.16 lcd_dsi_te

Lcd MIPI DSI panel Tear Effect

这个参数只有在 lcd_if=4 时才有效。

设置相应值的对应含义为：

：
0frame triggered automatically:
1frame triggered by te rising edge:
2frame triggered by te falling edge

注：设置为 0 时，刷屏间隔时间为 $\text{lcd_ht} \times \text{lcd_vt}$ ；设置为 1 或 2 时，刷屏间隔时间为两个 te 脉冲。

这个的作用就是屏一端发给 SOC 端的信号，用于同步信号，如果使能这个变量，那么 SOC 内部的显示中断将由这个外部脚来触发。

5.1.17 lcd_dsi_port_num

DSI 屏 port 数量

这个参数只有在 lcd_if=4 时才有效。

设置相应值的对应含义为：

： 一个

0port: 两个
1port

5.1.18 lcd_tcon_mode

Tcon 模式

这个参数只有在 lcd_if=4 时才有效。

设置相应值的对应含义为：

：

- 0normal mode:
- 1tcon master （在第一次发送数据同步） mode:
- 2:tcon master （每一帧都同步） mode:
- 3tcon slave （依靠modemaster 来启动） mode:
- 4one tcon driver two （条） dsi8lane

5.1.19 lcd_slave_tcon_num

Slave Tcon 的序号

这个参数只有在 lcd_if=4 时而且 lcd_tcon_mode 等于 1 或者 2 才有效。用于告诉 master 模式下的 tcon，从 tcon 的序号是多少。

设置相应值的对应含义为：

：

- 0tcon_lcd0:
- 1tcon_lcd1

5.1.20 lcd_tcon_en_odd_even_div

这个参数只有在 lcd_if=4 而且 lcd_tcon_mode=4 时才有效。

设置相应值的对应含义为：

：

0 将一帧图像分左右两半来发送给两个模块 tconDSI；

1 将一帧图像分奇偶像素来发给两个模块 tconDSI

5.1.21 lcd_sync_pixel_num

这个参数只有在 lcd_if=4 而且 lcd_tcon_mode 等于 2 或者 3 时才有效。

设置同步从 tcon 的起始 pixel

整数：不超过

lcd_ht

5.1.22 lcd_sync_line_num

这个参数只有在 lcd_if=4 而且 lcd_tcon_mode 等于 2 或者 3 时才有效。

设置同步从 tcon 的起始行

整数：不超过

lcd_vt

5.1.23 lcd_cpu_mode

Lcd CPU 模式，控制

设置相应值的对应含义为，设置为 0 时，刷屏间隔时间为 $\text{lcd_ht} \times \text{lcd_vt}$ ；设置为 1 或 2 时，刷屏间隔时间为两个 te 脉冲：

：中断自动根据时序，由场消隐信号内部触发

0：中断根据数据

1的触发或者由外部触发。Blockcounterte

5.2 LCD 时序参数说明

5.2.1 lcd_x

显示屏的水平像素点

5.2.2 lcd_y

显示屏的垂直像素点

5.2.3 lcd_ht

Horizontal Total time

指一行总的 dclk 的 cycle 个数。见下图：

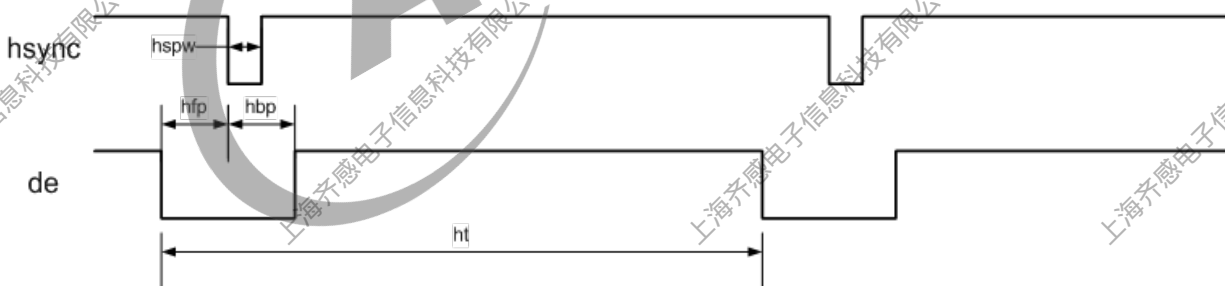


图 5-2: lcdht

5.2.4 lcd_hbp

Horizontal Back Porch

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。见上图，注意的是包含了 **hspw** 段。

lcd_hbp实际的=hbp实际的+hspw

5.2.5 lcd_hspw

Horizontal Sync Pulse Width

指行同步信号的宽度。单位为 1 个 dclk 的时间（即是 1 个 data cycle 的时间）。见上图。

5.2.6 lcd_vt

Vertical Total time

指一场的总行数。见下图：

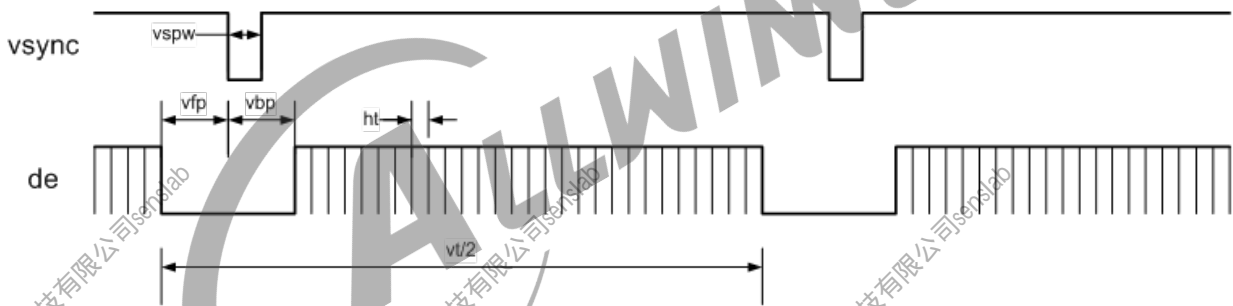


图 5-3: lcdvt

5.2.7 lcd_vbp

Vertical Back Porch

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。见上图，注意的是包含了 **vspw** 段。

也就是：

lcd_vbp实际的=vbp实际的+vspw

5.2.8 lcd_vspw

Vertical Sync Pulse Width

指场同步信号的宽度。单位为行。见上图。

5.2.9 lcd_dclk_freq

Dot Clock Frequency

传输像素传送频率。单位为 MHz

$$\text{fps} = (\text{lcd_dclk_freq} \times 1000 \times 1000) / (\text{ht} \times \text{vt})$$

这个值根据以下公式计算：

$$\text{lcd_dclk_freq} = \text{lcd_ht} \times \text{lcd_vt} \times \text{fps}$$

注意：

1. 后面的三个参数都是从屏手册中获得，fps 一般是 60
2. 如果是串行接口，发完一个像素需要 2 到 3 个周期的，那么

$$\text{lcd_dclk_freq} \times \text{cycles} = \text{lcd_ht} \times \text{lcd_vt} \times \text{fps}$$

或者

$$\text{lcd_dclk_freq} = \text{lcd_ht} \times \text{cycles} \times \text{lcd_vt} \times \text{fps}$$

5.3 LCD 其它参数说明

5.3.1 lcd_width

Width of lcd panel in mm

此参数描述 lcd 屏幕的物理宽度，单位是 mm。用于计算 dpi

5.3.2 lcd_height

height of lcd panel in mm

此参数描述 lcd 屏幕的物理高度，单位是 mm。用于计算 dpi

5.3.3 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。

5.3.4 lcd_pwm_ch

Pwm channel used

此参数标识使用的 Pwm 通道

5.3.5 lcd_pwm_freq

Lcd backlight PWM Frequency

这个参数配置 PWM 信号的频率，单位为 Hz。

5.3.6 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:

1active low

5.3.7 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 150，0₂₅₅ 范围内的亮度值将会被线性映射到 0₁₅₀ 范围内。用于控制最高背光亮度，节省功耗。

5.3.8 lcd_rb_swap

调换 tcon 模块 RGB 中的 R 分量和 B 分量。

: 不变
0: 调换
1分量和分量RB

5.3.9 lcd_frm

Lcd Frame Rate Modulator

FRM 是解决由于 PIN 减少导致的色深问题。

这个参数设置相应值对应含义为：

:
0RGB888 -- RGB888 direct:
1RGB888 -- RGB666 dither:
2RGB888 -- RGB565 dither

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。



图 5-4: good



图 5-5: bad

5.3.10 lcd_gamma_en

Lcd Gamma Correction Enable

设置相应值的对应含义为：

：

0的校正功能关闭LcdGamma:

1的校正功能开启LcdGamma

设置为 1 时，需要在屏驱动中对 lcd_gamma_tbl [256] 进行赋值。

5.3.11 lcd_bl_n_percent

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，以使亮度变化更线性。

比如 lcd_bl_50_percent = 60，表明将 50% 的亮度值调整成 60%，即亮度比原来提高 10%。

5.3.12 lcd_cmap_en

Lcd Color Map Enable

设置相应值的对应含义为：

：

0的色彩映射功能关闭Lcd:

1的色彩映射功能开启Lcd

设置为 1 时，需要对 lcd_cmap_tbl[2][3][4] 进行赋值 Lcd Color Map Table。

每个像素有 R、G、B 三个单元，每四个像素组成一个选择项，总共有 12 个可选。数组第一维表示奇偶行，第二维表示像素的 RGB，第三维表示第几个像素，数组的内容即表

示该位置映射到的内容。

LCD CMAP 是对像素的映射输出功能，只有像素有特殊排布的 LCD 屏才需要配置。

LCD CMAP 定义每行的 4 个像素为一个总单元，每个像素分 R、G、B 3 个小单元，总共有 12 个小单元。通过 `lcd_cmap_tbl` 定义映射关系，输出的每个小单元可随意映射到 12 个小单元之一。

```
_u32 lcd_cmap_tbl[2][3][4] = {  
{  
  {LCD_CMAP_G0,LCD_CMAP_B1,LCD_CMAP_G2,LCD_CMAP_B3},  
  {LCD_CMAP_B0,LCD_CMAP_R1,LCD_CMAP_B2,LCD_CMAP_R3},  
  {LCD_CMAP_R0,LCD_CMAP_G1,LCD_CMAP_R2,LCD_CMAP_G3},  
},  
{  
  {LCD_CMAP_B3,LCD_CMAP_G2,LCD_CMAP_B1,LCD_CMAP_G0},  
  {LCD_CMAP_R3,LCD_CMAP_B2,LCD_CMAP_R1,LCD_CMAP_B0},  
  {LCD_CMAP_G3,LCD_CMAP_R2,LCD_CMAP_G1,LCD_CMAP_R0},  
},  
};
```

如上，上三行代表奇数行的像素排布，下三行代表偶数行的像素排布；

每四个像素为一个单元，第一列代表每四个像素的第一个像素映射，第二列代表每四个像素的第二个像素映射，以此类推。

如上的定义，像素的输出格式如下图所示。

奇数行	G0	B1	G2	B3	G4	B5	G6	B7
	B0	R1	G2	R3	B4	R5	G6	R7
	R0	G1	R2	G3	R4	G5	R6	G7
偶数行	B3	G2	B1	G0	B7	G6	B5	G4
	R3	G2	R1	B0	R7	G6	R5	B4
	G3	R2	G1	R0	G7	R6	G5	R4

图 5-6: cmap

5.4 POWER 及 IO 说明

5.4.1 lcd_power

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。

Power 的配置时建议根据 axp 使用文档进行。

注意: 如果有多个电源需要打开, 则定义 lcd_power1, lcd_power2 等。

5.4.2 lcd_bl_en

示例:

```
lcd_bl_en = port:PD24<1><0><default><1>
```

含义: lcd_power 引脚为 PD24, PD24 输出高电平时打开 LCD 背光; 上下拉不使能。

- 第一个尖括号: 功能分配; 1 为输出;

- 第二个尖括号：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。
- 第三个尖括号：驱动能力；**default** 表驱动能力是等级 1
- 第四个尖括号：输出有效所需电平；LCD 背光工作时的电平，0 为低电平，1 为高电平。

需要在屏驱动调用相应的接口进行开、关的控制

5.4.3 lcd_gpio_0

示例：

```
lcd_gpio_0 = port:PD25<0><0><default><0>
```

含义：lcd_gpio_0 引脚为 PD25。

- 第一个尖括号：功能分配；0 为输入，1 为输出；
- 第二个尖括号：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。
- 第三个尖括号：驱动能力；**default** 表驱动能力是等级 1
- 第四个尖括号：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。

需要在屏驱动调用相应的接口进行拉高，拉低的控制。

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

5.4.4 lcddx

示例：

```
lcdd0 = port:PD00<3><0><default><default>
```

含义：lcdd0 这个引脚，即是 PD0，配置为 LVDS 输出。

- 第一个尖括号：功能分配；0 为输入，1 为输出，2 为 LCD 输出，3 为 LVDS 接口输出，7 为 disable。
- 第二个尖括号：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。
- 第三个尖括号：驱动能力；default 表驱动能力是等级 1
- 第四个尖括号：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。

LCD PIN 的配置如下：

LCD 为 HV RGB 屏，CPU/I80 屏时，必须定义相应的 IO 口为 LCD 输出（如果是 0 路输出，第一个尖括号为 2；如果是 1 路输出，第一个尖括号为 3）；

具体的 IO 对应关系可参考 user manual 手册进行配置。

LCD PIN 的所有 IO，均可通过注释方式去掉其定义，显示驱动对注释 IO 不进行初始化操作。

需要在屏驱动调用相应的接口进行开、关的控制。

注意：不是一定要叫做 lcdd0 这个名字，你改成其它名字对驱动不会造成任何影响，这里只是为了方便记忆。

5.4.5 lcd_pin_power

示例：

```
lcd_pin_power = "vcc-" pd
```

配置 lcddx 的电源。这电源会在 lcd 的 io 使能时自动开启。像 mipi-dsi 不用配管脚的，一般也需要配置 lcd_pin_power，因为如果用到 pwm，也需要配置这个。注意：如果要多组，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。

6

屏驱动说明

6.1 屏驱动说明

屏驱动中需要实现的函数接口有LCD_cfg_panel_info, LCD_open_flow, LCD_close_flow和LCD_get_panel_funs_0/ LCD_get_panel_funs_1是必须包含的 4 个函数。

函数：LCD_cfg_panel_info

功能：配置的 TCON 扩展参数

原型：

```
static void LCD_cfg_panel_info(__panel_extend_para_t * info)
```

TCON 的扩展参数只能在屏文件中配置，参数的定义见“5.3 LCD 其他参数说明”。

需要 gamma 校正，或色彩映射，在sys_config中将相应模块的 enable 参数置 1，lcd_gamma_en, lcd_cmap_en，并且填充 3 个系数表，lcd_gamma_tbl, lcd_cmap_tbl，如下所示红色代码部分。注意的是：gamma，模板提供了 18 段拐点值，然后再插值出所有的值（255 个）。如果觉得还不细，可以往相应表格里添加子项。cmap_tbl 的大小是固定了，不能减小或增加表的大小。

最终生成的 gamma 表项是由 rgb 三个 gamma 值组成的，各占 8bit，目前提供的模板中，三个 gamma 值是相同的。

函数：LCD_open_flow

功能：定义开屏的流程

原型：

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

具体说明见“6.2 开关屏流程”。

函数：LCD_close_flow

功能：定义关屏的流程

原型：

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

该函数与 LCD_open_flow 对应

屏驱动：__lcd_panel_t default_panel

功能：

原型：名字可改成屏相对应的名字。

6.1.1 开关屏流程

开关屏的操作流程如图 6-2 所示。

其中，LCD_open_flow 和 LCD_close_flow 称为开关屏流程函数，方框中的函数，如 LCD_power_on，TCON_open 等函数，称为开关屏步骤函数。

不需要进行初始化操作的 LCD 屏，LCD_panel_init 及 LCD_panel_exit 这函数可以为空。

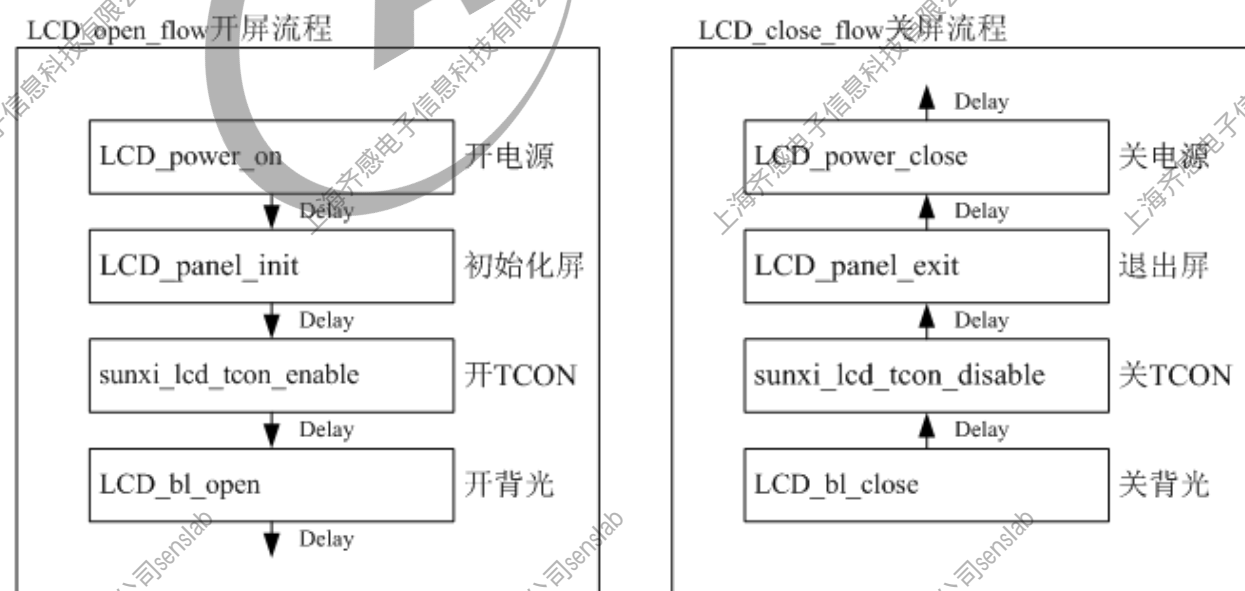


图 6-1: flow

6.1.2 开关屏流程函数说明

函数：LCD_open_flow

功能：初始化开关屏的步骤流程

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, sunxi_lcd_tcon_enable, 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. 打开 TCON，再延迟 100ms；
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。

函数：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。

6.1.3 屏驱动可使用接口说明

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);

函数：sunxi_lcd_tcon_enable/sunxi_lcd_tcon_disable

功能：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。

原型：void sunxi_lcd_tcon_enable(u32 screen_id);/ void sunxi_lcd_tcon_disable(u32 screen_id);

函数：sunxi_lcd_backlight_enable/sunxi_lcd_backlight_disable

功能：打开/关闭背光，操作的是 sys_config 中 lcd_bl 配置的 gpio。见 5.4.2 lcd_bl_en

原型：void sunxi_lcd_backlight_enable(u32 screen_id);

```
void sunxi_lcd_backlight_disable(u32 screen_id);
```

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);
```

函数: sunxi_lcd_power_enable / sunxi_lcd_power_disable

功能: 打开/关闭 Lcd 电源, 操作的是 sys_config 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

函数: sunxi_lcd_pin_cfg

功能: 配置 lcd 的 io。

```
原型: s32 sunxi_lcd_pin_cfg(u32 screen_id, u32 bon);
```

说明: 配置 lcd 的 data/clock 等 pin, 对应 sys_config 中的 lcdd0-lcdd23/lcddclk/lcdde/lcdhsync/lcdvsync。

由于 dsi 是专用 pin, 所以 dsi 接口屏不需要在 sys_config 中配置这组 pin, 但同样会在此函数接口中打开与关闭对应的 pin。

Bon: 1: 为开, 0: 为配置成 disable 状态。

函数: sunxi_lcd_dsi_clk_enable / sunxi_lcd_dsi_clk_disable

功能: dsi 接口屏使用, 使能/关闭 dsi 输出的 clk 信号。

```
原型: s32 sunxi_lcd_dsi_clk_enable(u32 screen_id);
```

```
s32 sunxi_lcd_dsi_clk_disable(u32 scree_id);
```

6.2 屏的初始化

一部分屏需要进行初始化操作，在开屏步骤函数中，对应于 LCD_panel_init 函数，提供了几种方式对屏的初始化。

对于 DSI 屏，是通过 DSI-D0 通道进行初始化。对于 CPU 屏，是通过 8080 总线的方式，使用的是 LCDIO (PD,PH) 进行初始化。这种初始化方式，其总线的引脚位置定义与 CPU 屏一致。

以下这些接口在 3.1 中提到路径的 lcd_source.c 和 lcd_source.h 中定义和实现。

6.2.1 MIPI DSI 屏的初始化

MIPI DSI 屏，大部分需要初始化，使用的是 DSI-D0 通道的 LP 模式进行初始化。提供的接口函数说明如下：

函数：sunxi_lcd_dsi_dcs_wr

功能：对屏的 dcs 写操作

原型：__s32 sunxi_lcd_dsi_dcs_wr(__u32 sel, __u8 cmd, __u8* para_p, __u32 para_num);

参数说明：

- 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

函数：sunxi_lcd_dsi_dcs_wr_2para

功能：对屏的 dcs 写操作，该命令带有两个参数

原型：__s32 sunxi_lcd_dsi_dcs_wr_2para(__u32 sel, __u8 cmd, __u8 para1, __u8 para2);

参数说明：

- 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

dsi_dcs_wr_0para, dsi_dcs_wr_1para, dsi_dcs_wr_3para, dsi_dcs_wr_4para, dsi_dcs_wr_5para 定义与 dsi_dcs_wr_2para 类似。

6.2.2 CPU/I80 屏的初始化

CPU 屏的初始化可以参考“附录 5.3.5”的实例。

显示驱动提供 5 个接口函数可供使用。如下：

函数：sunxi_lcd_cpu_write

功能：设定 CPU 屏的指定寄存器为指定的值

原型：void sunxi_lcd_cpu_write(__u32 sel, __u32 index, __u32 data)

函数内容为

```
Void sunxi_lcd_cpu_write(__u32 sel, __u32 index, __u32 data)
{
    Sunxi_lcd_cpu_write_index(sel, index);
    Sunxi_lcd_cpu_wirte_data(sel, data);
}
```

实现了 8080 总线上的两个写操作。

Sunxi_lcd_cpu_write_index 实现第一个写操作，这时 PIN 脚 RS (A1) 为低电平，总线数据上的数据内容为参数 index 的值。

Sunxi_lcd_cpu_wirte_data 实现第二个写操作，这时 PIN 脚 RS (A1) 为高电平，总线数据上的数据内容为参数 data 的值。

函数：Sunxi_lcd_cpu_write_index

功能：设定 CPU 屏为指定寄存器

原型：

```
void Sunxi_lcd_cpu_write_index(__u32 sel, __u32 index);
```

具体说明见 Sunxi_lcd_cpu_write。

函数：Sunxi_lcd_cpu_write_data

功能：设定 CPU 屏寄存器的值为指定的值

原型：

```
void Sunxi_lcd_cpu_write_data(__u32 sel, __u32 data);
```

具体说明见 Sunxi_lcd_cpu_write。

6.2.3 使用 IO 模拟串行接口初始化

IO 的位置（PIN 脚）定义，默认属性（输入输出）定义及默认输出值在 sys_config.fex。见 5.4.4 lcd_gpio_x

显示驱动提供 2 个接口函数可供使用。说明如下：

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。

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

6.2.4 使用 iic/spi 串行接口初始化

需要在屏驱动中注册 iic/spi 设备对串行接口的访问。

6.3 ESD 静电检测自动恢复功能

这个功能在 linux4.9 以及 linux 3.10 sunxi-product 分支上实现了, 如果需要这个功能, 需要完成以下步骤:

首先打开如下内核配置:

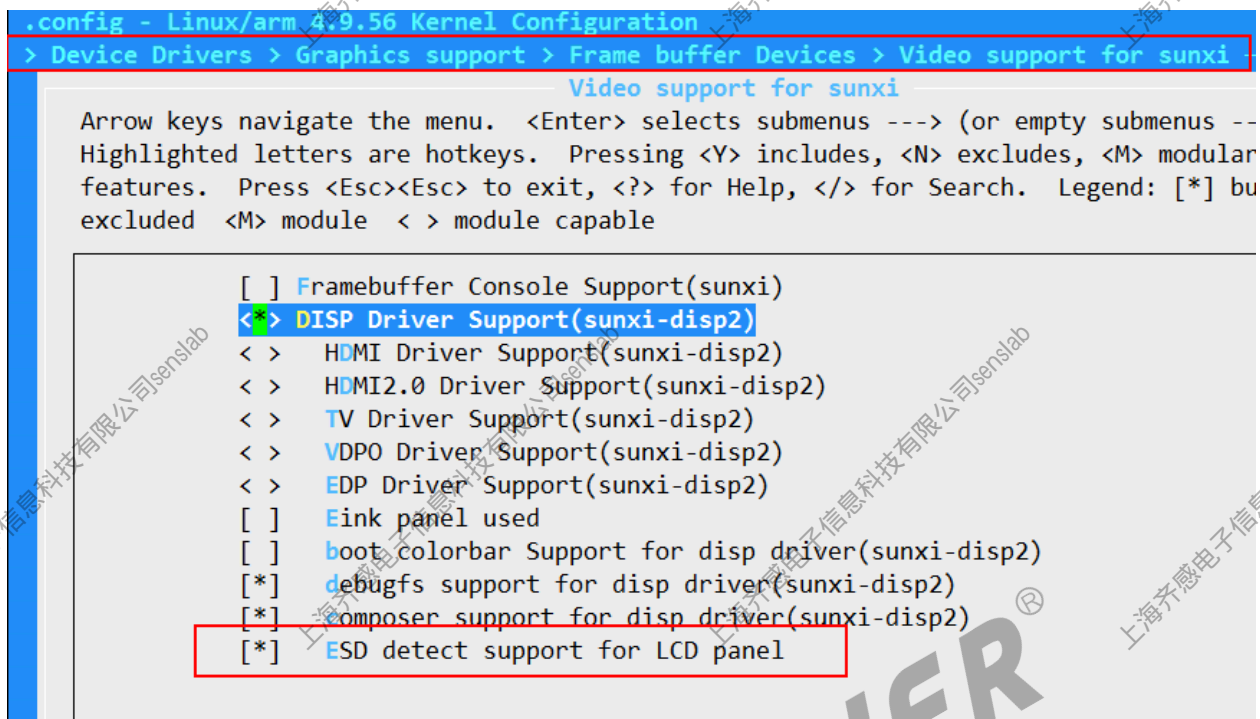


图 6-2: menu

修改屏驱动，实现三个回调函数

如下示例，在屏 he0801a068 上添加 esd 相关的回调函数

(linux-4.9/drivers/video/fbdev/sunxi/disp2/disp/lcd/he0801a068.c) :

```
struct __lcd_panel he0801a068_panel = {  
    /* panel driver name, must mach the name of  
     * lcd_drv_name in sys_config.fex  
     */  
    .name = "he0801a068",  
    .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,  
        .esd_check = lcd_esd_check,  
        .reset_panel = lcd_reset_panel,  
        .set_esd_info = lcd_set_esd_info,  
    },  
};
```

图 6-3: menu

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。

返回值: 如果屏正常的话就返回 0, 不正常的话就返回非 0。

sel: 显示索引。

由于屏的类型接口众多, 不同屏检测屏的状态各异, 一般来说是通过驱动接口读取屏的内部信息 (id 或者其它寄存器), 如果获取正常则认为屏是正常的, 获取失败则认为屏是异常的。比如下面 dsi 屏的做法:

```

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

    ret = sunxi_lcd_dsi_gen_short_read2p(0, 0x10, 0x0, result);
    if (result[0] != 0x3f)
        ret = -1;
    else
        ret = 0;
    return ret;
}

```

图 6-4: menu

此外，一般情况下，也会通过 dsi 接口读取 0x0A 命令（获取 power 模式）来判断屏是否正

sunxi_lcd_dsi_dcs_read(sel, 0x0A, result, &num)

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 st 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 6-5: menu

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。

返回值：复位成功就是 0，复位失败非 0

sel：显示索引

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 6-6: menu

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。

返回值：成功设置返回 0，否则非 0。

p_info：需要设置的 esd 行为结构体。

示例：下面图所示，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。下图中的 level 为 1 表示复位全志 SOC 的 LCD 相关模块以及 reset_panel 里面的操作，level 为 0 的时候表示仅仅执行 reset_panel 里面的操作。

```
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 6-7: menu

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) 2560
    x1600
    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 levele 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）

7

一些有用调试手段

系统起来之后可以读取 sysfs 一些信息，来协助调试。

7.1 查看显示信息

```
cat /sys/class/dispc/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[global 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]
```

其中 ref_fps:60 是根据你在 sys_config.fex 的 lcd0 填的时序算出来的理论值，而 fps:60.9 后面的数值则是实时统计的，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。

irq:1942 这是中断的次数，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。

BUF 开头的表示图层信息，一行 BUF 表示一个图层，如果一个 BUF 都没有出现，那么将是黑屏，不过和屏驱动本身关系就不大了，应该查看应用层 & 框架层

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_ldoio1 : disabled 0 700000 supply_name:
pmu1736_dc1sw : enabled 1 3300000 supply_name: vcc-lcd
pmu1736_cpui5 : 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_dcdc6 : 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。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
```

7.6 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 上，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 7-1: colorbar

如果有多个显示，想让第二个显示显示 colorbar 的话，那么先

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

然后再执行上上面操作。

如果没有这个 attr 的话，可以直接操作寄存器，也就是操作 tcon 寄存器的 040 偏移的最低 3 位。

在 linux 下，cd /sys/class/sunxi_dump 然后

```
echo 0x06511040 > dump;cat dump
```

这样会打印当前 tcon 的 040 偏移寄存器的值，然后在上面值的基础上修改最低 3 位为上图的值即可，修改方式，示例：

```
echo 0x06511040 0x800001f1 > write
```

注意 tcon 的基地址不一定是 0x06511000，不同平台不一样，请参考 SOC 文档获取 tcon 的基地址。



8 FAQ

8.1 黑屏

问题表现:

1. 完全黑屏, 背光也没有
2. 有背光, 但是显示黑屏

第一种现象很明显了, 肯定是 **pwm** 和背光电路的问题, **pwm** 的信息申请可以看 **pwm** 信息, 硬件测量下相关管脚和电压。

第二种现象, 有两种可能种导致, 第一个是屏初始化失败, 屏初始化失败的原因又有很多个

1. DSI 屏的 **io** 逻辑电源没开, DSI 屏通常需要一个 1.8v 的 **io** 逻辑电压 2. 复位脚没有复位, 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。3. 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。4. **sys_config.fex** 中的 **lcd** 时序写得太离谱。请严格按照屏手册中的提示来写! 手册没写就找屏厂, 这个时序只有屏厂清楚, 全志并不知道。

第二个则是, 应用层框架层没有送图层下来, 请看查看显示信息

8.2 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

sys_config.fex 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。

8.3 条形波纹

由于 RGB666 缺少两位导致现象，请看lcd_firm 解释

8.4 背光太亮或者太暗

请看lcd_bl_n_percent

8.5 重启断电测试屏异常

这种问题只能优化屏初始化的时序（上电顺序），而对屏时序最清楚的只有屏厂，请找屏厂支持。另外就是 sys_config.fex 中lcd0 的时序必须严格按照屏手册所写。

8.6 RGB 接口或者 I8080 接口显示抖动有花纹

2. 改大时钟管脚的管脚驱动能力

比如原来是：

```
lcdclk = port:PD18<2><0><2><default>
```

可以改成

```
lcdclk = port:PD18<2><0><3><default>
```

2. 修改时钟相位，也就是修改 lcd_hv_clk_phase



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. 添加屏驱动请看屏驱动源码和新添加屏驱动
2. 仔细阅读屏手册以及 driver IC 手册（有的话）
3. 仔细阅读第五章
4. 确保 LCD 所需要的各路电源管脚正常

10 Declaration

This document is the original work and copyrighted property of Allwinner Technology (‘ ‘Allwinner’ ’). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application. tates nor implies warranty of any kind, including fitness for any particular application.

