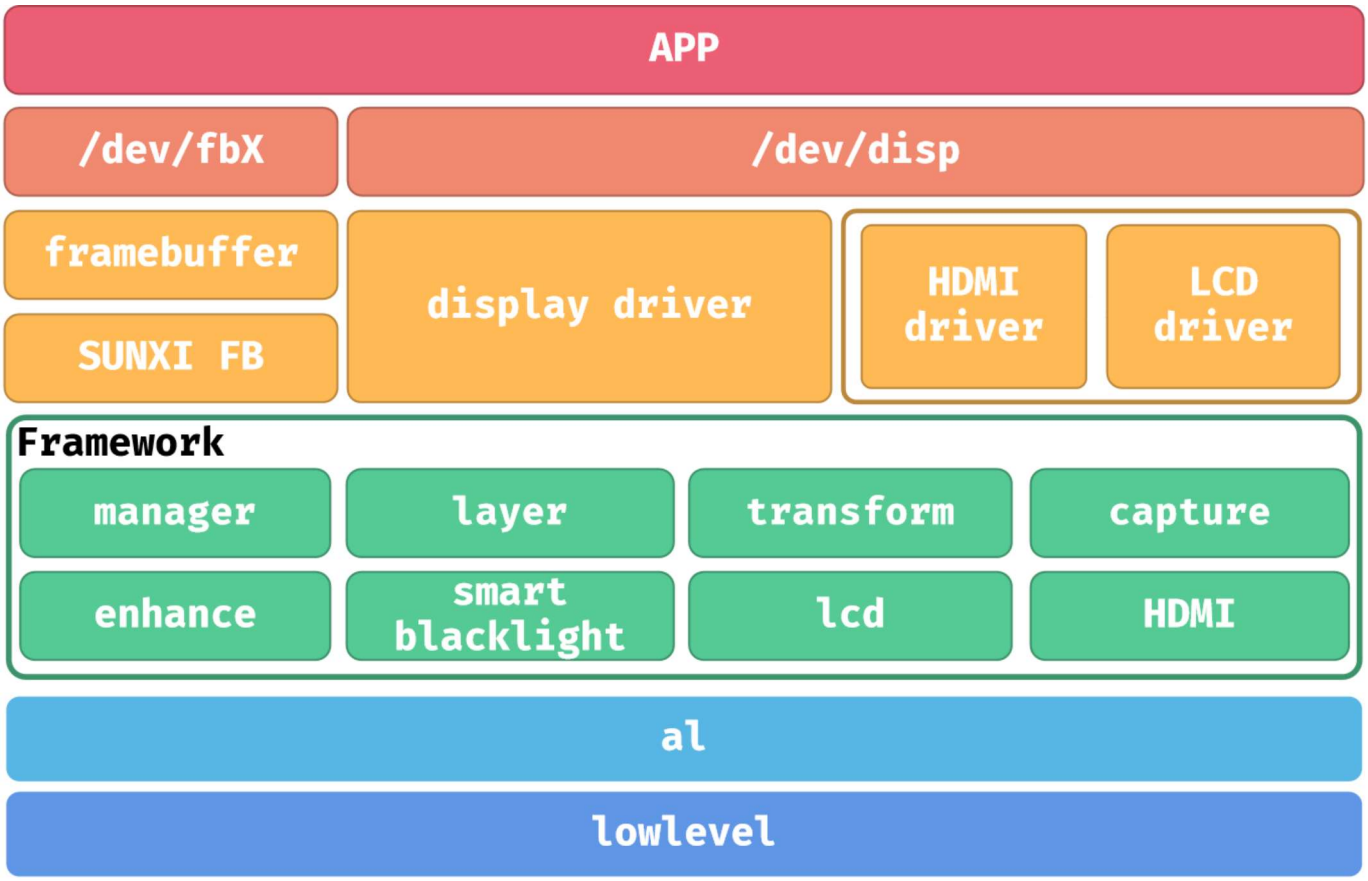
LCD 屏幕适配

Tina Linux 提供了一套完整的屏幕驱动，支持 RGB，MIPI DSI，eDP，LVDS，E-INK屏幕，也支持低分辨率的 SPI，IIC 屏幕。具体屏幕的驱动情况，需要根据芯片而确定。

本文将通过介绍 Linux 4.9 与 Linux 5.4 两个版本的 Kernel 中的 LCD 驱动，讲解配置屏幕驱动的基本方法。

LCD 的驱动



显示驱动可划分为三个层面：驱动层，框架层及底层。底层与图形硬件相接，主要负责将上层配置的功能参数转换成硬件所需要的参数，并配置到相应寄存器中。

显示框架层对底层进行抽象封装成一个个的功能模块。驱动层对外封装功能接口，通过内核向用户空间提供相应的设备结点及统一的接口。

在驱动层，分为三个驱动，分别是 `framebuffer` 驱动， `display` 驱动， `LCD&HDMI` 驱动。 `framebuffer` 驱动与 `framebuffer core` 对接，实现 `linux` 标准的 `framebuffer` 接口。 `display` 驱动是是整个显示驱动中的核心驱动模块，所有的接口都由 `display` 驱动来提供，包括 `lcd` 的接口。

LCD 配置

U-Boot

U-Boot 配置

对于 U-Boot 平台，一般推荐直接修改 `config` 文件，位于

```
lichee/brandy-2.0/u-boot-2018/configs/xxx_defconfig
```

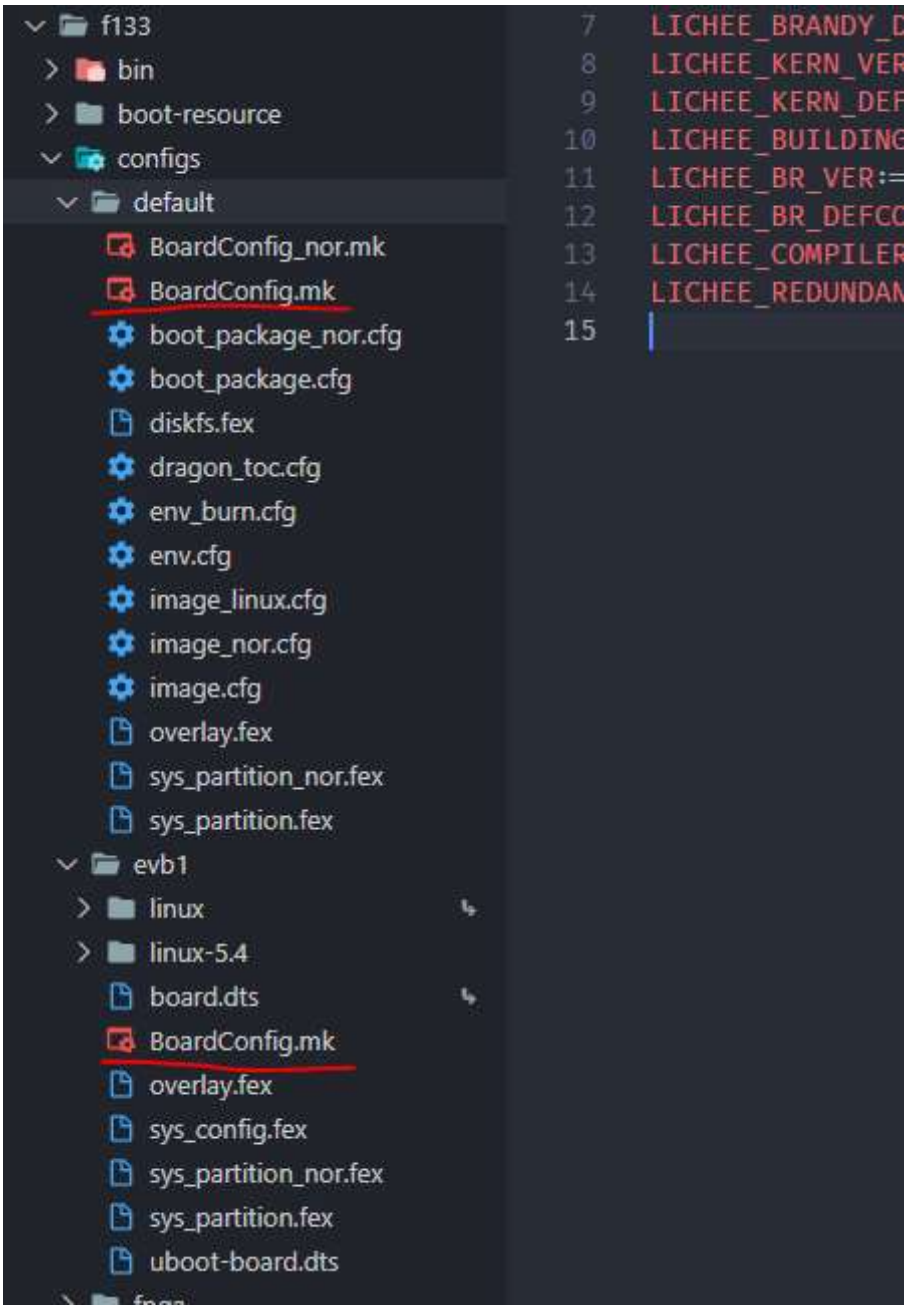
其中的 `xxx` 是芯片的软件代号。可以在 `BoardConfig.mk` 文件里找到。例如 `D1-H` 芯片的 `BoardConfig.mk` 文件，位于：

```
device/config/chips/d1-h/configs/default/BoardConfig.mk
```

不过也要注意，有些时候除了 `default` 文件夹内有 `BoardConfig.mk` 文件，不同的项目文件夹内也有 `BoardConfig.mk` 文件。这个时候就需要使用项目文件夹里的 `BoardConfig.mk` 配置文件。优先使用的是项目文件夹里的配置文件。

> 在这里， `evb1` 文件夹为项目文件夹，代表 `f133-evb1` 这个开发板

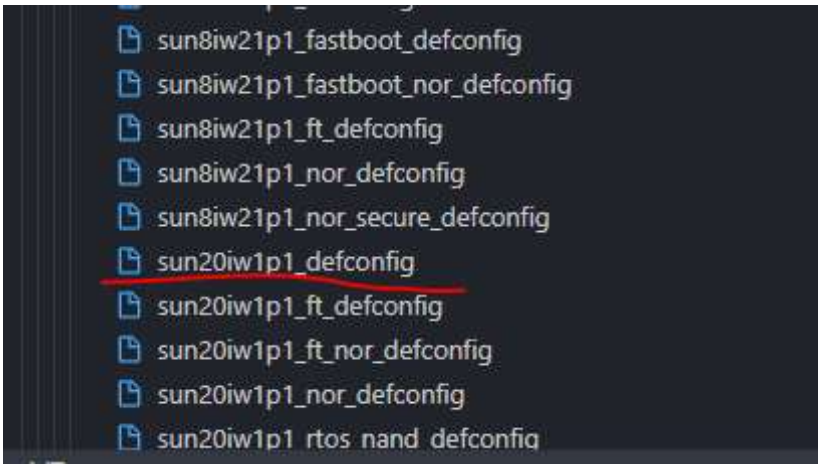
> `default` 是缺省文件夹，项目文件夹里找不到配置文件就会去缺省文件夹里寻找



我们打开 `BoardConfig.mk` 配置文件

```
LICHEE_CHIP:=sun20iw1p1
LICHEE_PRODUCT:=
LICHEE_BOARD:=
LICHEE_FLASH:=
LICHEE_ARCH:=riscv
LICHEE_BRANDY_VER:=2.0
LICHEE_BRANDY_DEFCONF:=sun20iw1p1_defconfig
LICHEE_KERN_VER:=5.4
LICHEE_KERN_DEFCONF:=sun20iw1p1_nodvfs_defconfig
LICHEE_BUILDING_SYSTEM:=buildroot
LICHEE_BR_VER:=201902
LICHEE_BR_DEFCONF:=sun20i_defconfig
LICHEE_COMPILER_TAR:=riscv64-glibc-gcc-thead_20200702.tar.xz
LICHEE_REDUNDANT_ENV_SIZE:=0×20000
```

可以看到这颗芯片使用的是 `sun20iw1p1_defconfig` 这个配置文件。就可以在 U-Boot 的 `config` 文件夹找到这个配置文件。

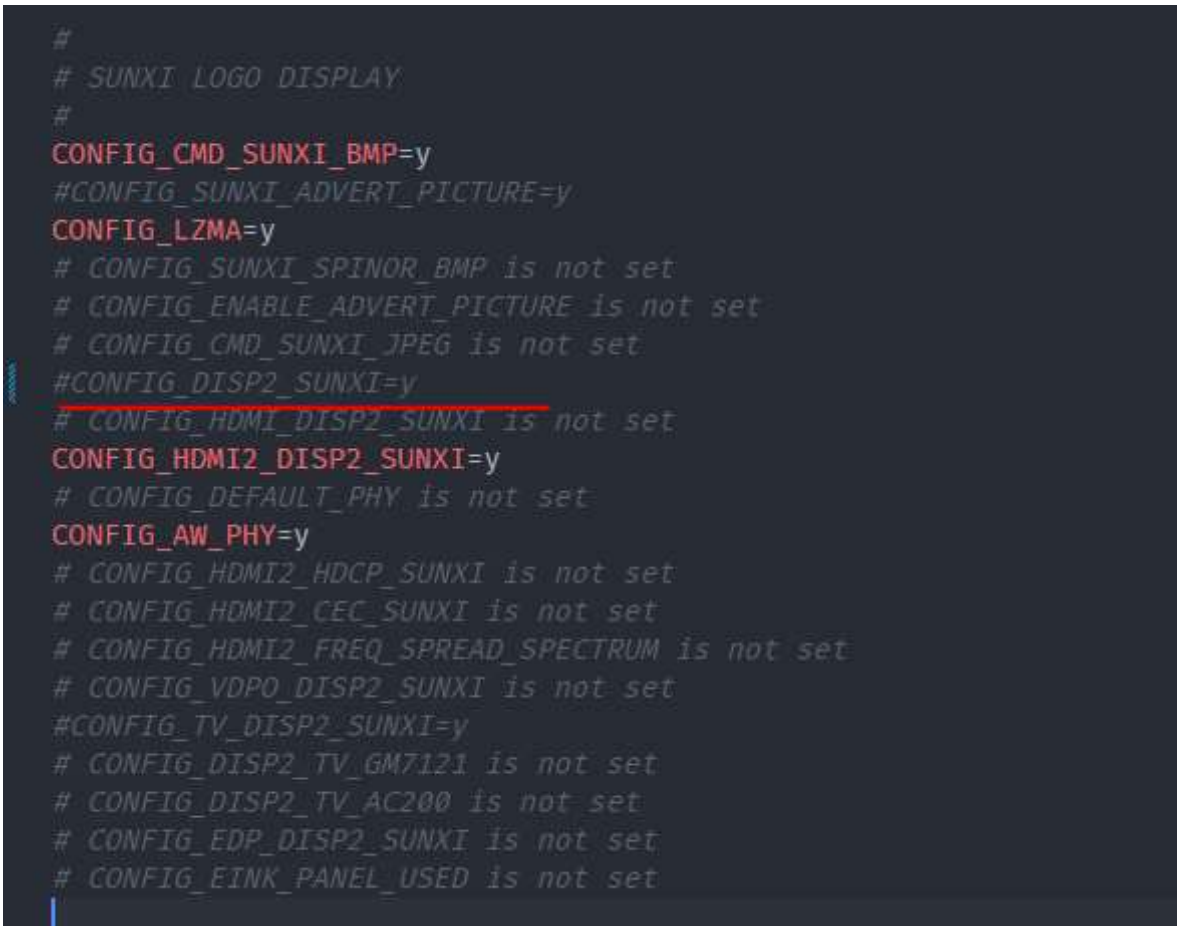


打开它，可以在文件中找到 `SUNXI LOGO DISPLAY` 后面的部分，看到这两部分配置：

红色框框住的是 `DISP` 驱动部分，主要部分如下：

```
CONFIG_CMD_SUNXI_BMP=y          # 开机 LOGO BMP 文件解析器
CONFIG_LZMA=y                   # 开机 LOGO 使用 LZMA 压缩解压工具
CONFIG_DISP2_SUNXI=y            # DISP 驱动
CONFIG_HDMI2_DISP2_SUNXI=y      # HDMI DISP 驱动
CONFIG_AW_PHY=y                 # DISP 驱动使用的是全志自研外设
```

如果希望关闭 U-Boot 的 LCD DISP 驱动输出，可以注释掉 `CONFIG_DISP2_SUNXI=y` 这一行。



关闭 HDMI DISP 驱动也相同，注释即可。

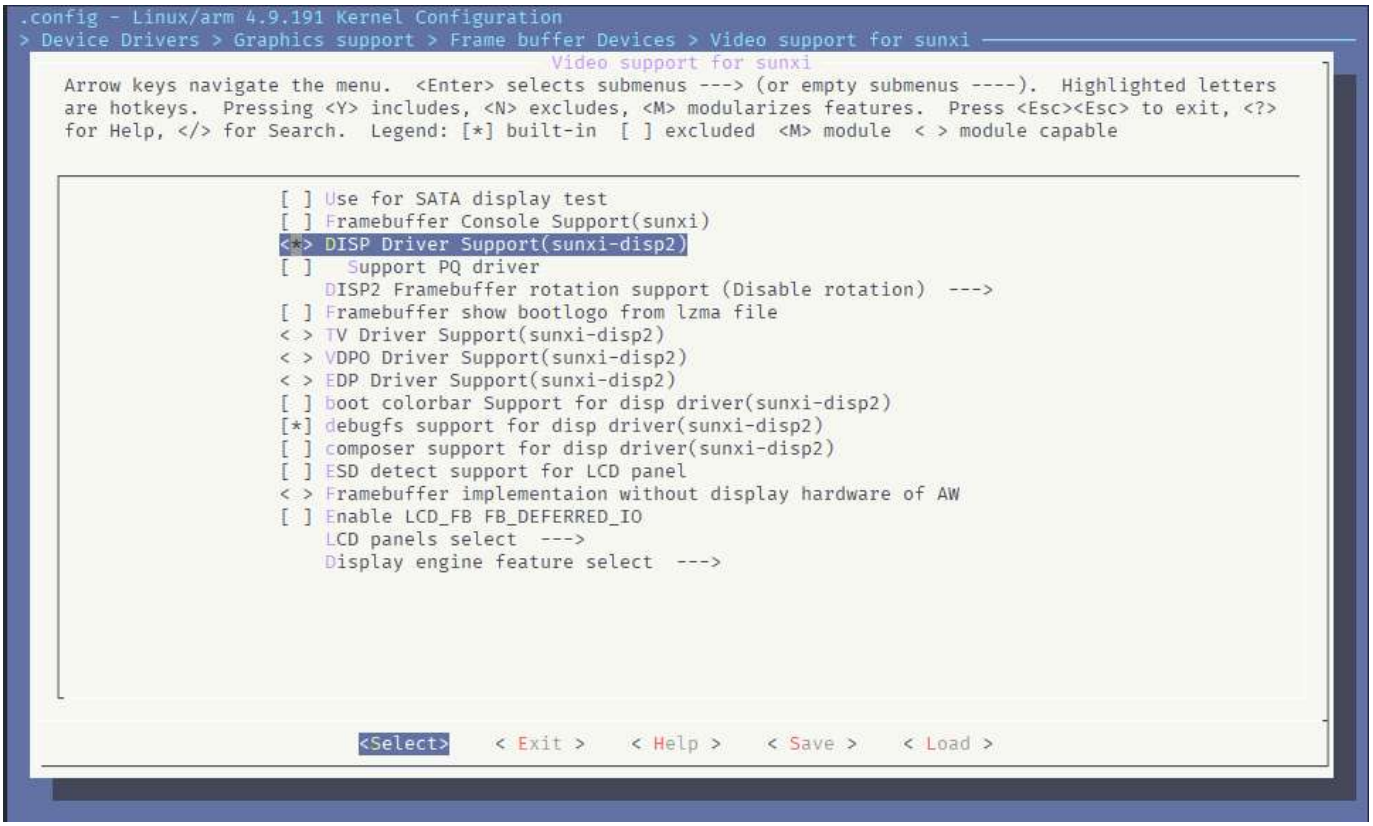
Linux

对于 Tina Linux，4.9 与 5.4 的 LCD 驱动是通用的，底层的差异做了抽象处理不需要另外修改代码。所以这里的配置方法也是相同的。这里以**Linux 4.9 的 V853 平台的配置**举例。

内核配置

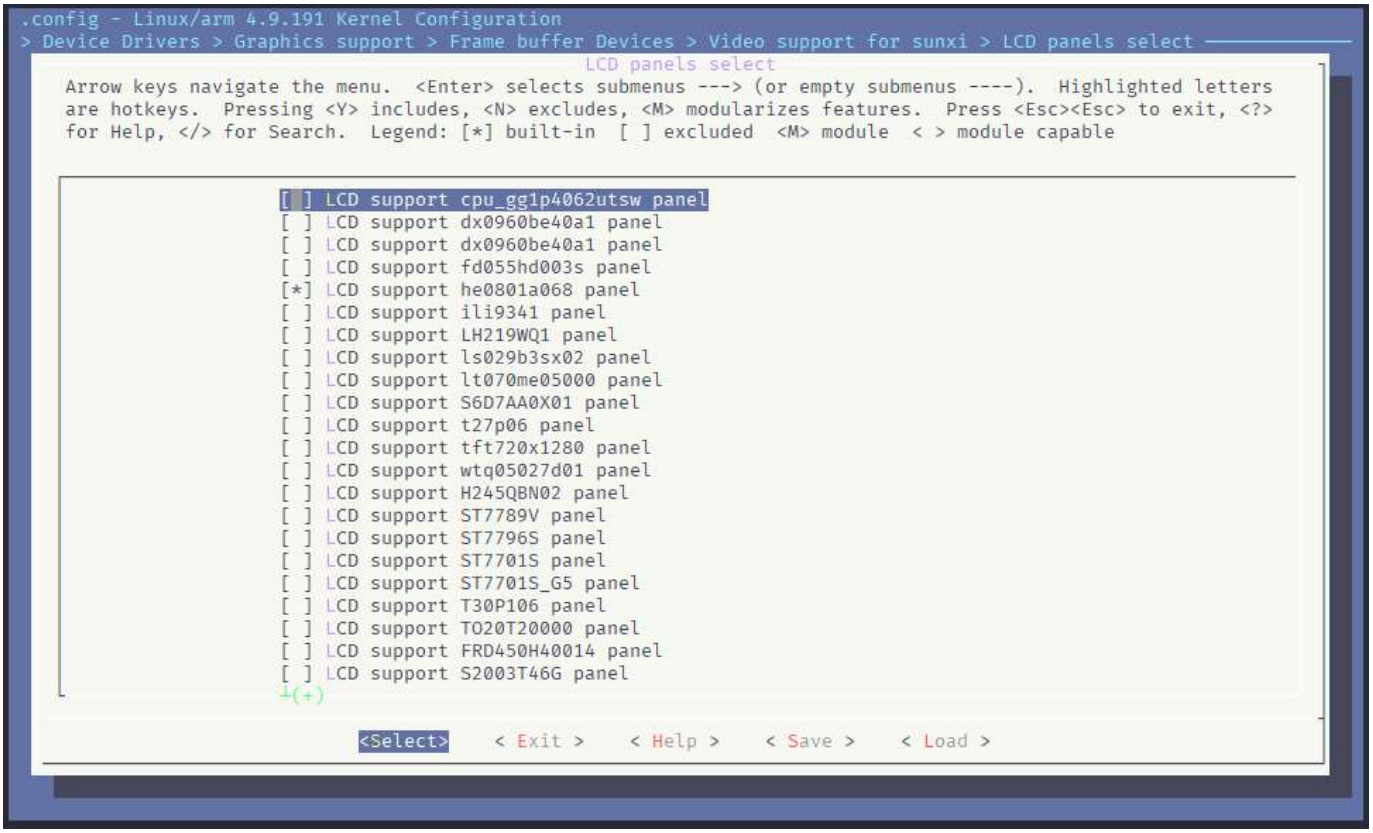
运行 `make kernel_menuconfig` 进入内核配置，找到 `Video support for sunxi`

```
Device Drivers > Graphics support > Frame buffer Devices > Video support for sunxi
```



在显示驱动中最主要的是 `<*> DISP Driver Support(sunxi-disp2)` ，勾选后可以看到其他的选项。包括驱动支持，调试接口和 LCD 面板的选择。（`LCD panels select`）

进入 LCD 面板选择可以看到许多已经适配了的显示屏可供选择使用。



设备树配置

与其他设备相同，屏幕驱动也使用了两份设备树。第一份配置了显示驱动的地址，时钟等等参数，位于 `kernel/linux-4.9/arch/arm/boot/dts/sun8iw21p1.dtsi` 一般来说这里的参数**都不需要修改**，默认即可，另外一份配置位于 `device/config/chips/v853/configs/vision/board.dts` 包括了两个配置节点。第一个是 `display` 所使用的节点，配置了屏幕的特性与功能，另外一个是 `lcd` 所使用的节点，配置了 LCD 面板的驱动与参数。

```
&disp {
    disp_init_enable                = <1>;
    disp_mode                       = <0>;

    screen0_output_type             = <1>;
    screen0_output_mode             = <4>;

    screen1_output_type             = <3>;
    screen1_output_mode             = <10>;

    screen1_output_format           = <0>;
    screen1_output_bits             = <0>;
    screen1_output_eotf             = <4>;
    screen1_output_cs               = <257>;
    screen1_output_range            = <2>;
    screen1_output_scan             = <0>;
    screen1_output_aspect_ratio     = <8>;

    dev0_output_type                = <1>;
    dev0_output_mode                = <4>;
    dev0_screen_id                  = <0>;
    dev0_do_hpd                     = <0>;

    dev1_output_type                = <4>;
    dev1_output_mode                = <10>;
    dev1_screen_id                  = <1>;
    dev1_do_hpd                     = <1>;

    def_output_dev                  = <0>;

    fb0_format                      = <0>;
    fb0_width                       = <0>;
    fb0_height                      = <0>;

    fb1_format                      = <0>;
    fb1_width                       = <0>;
    fb1_height                      = <0>;
    chn_cfg_mode                    = <1>;

    disp_para_zone                  = <1>;
};
```

```
&lcd0 {
    base_config_start   = <1>;
    lcd_used            = <1>;
    lcd_driver_name     = "icn6202";
    lcd_bl_0_percent    = <0>;
    lcd_bl_40_percent   = <23>;
    lcd_bl_100_percent  = <100>;
    lcd_backlight       = <150>;
    lcd_if              = <4>;
    lcd_x               = <1280>;
    lcd_y               = <800>;
    lcd_width           = <62>;
    lcd_height          = <110>;
    lcd_dclk_freq       = <71>;
    lcd_pwm_used        = <1>;
    lcd_pwm_ch          = <9>;
    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              = <830>;
    lcd_vspw            = <5>;
    lcd_dsi_if          = <0>;
    lcd_dsi_lane        = <4>;
    lcd_dsi_format      = <0>;
    lcd_dsi_te          = <0>;
    lcd_dsi_cotp        = <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_gpio_0          = <&pio PD 19 1 0 3 1>;
    lcd_gpio_1          = <&pio PD 20 1 0 3 1>;
    pinctrl-0           = <&dsi4lane_pins_a>;
    pinctrl-1           = <&dsi4lane_pins_b>;
    base_config_end     = <1>;
};
```

这里的驱动配置非常复杂，具体代表的含义请参考《Linux_LCD_开发指南.pdf》，这里不做过多说明。

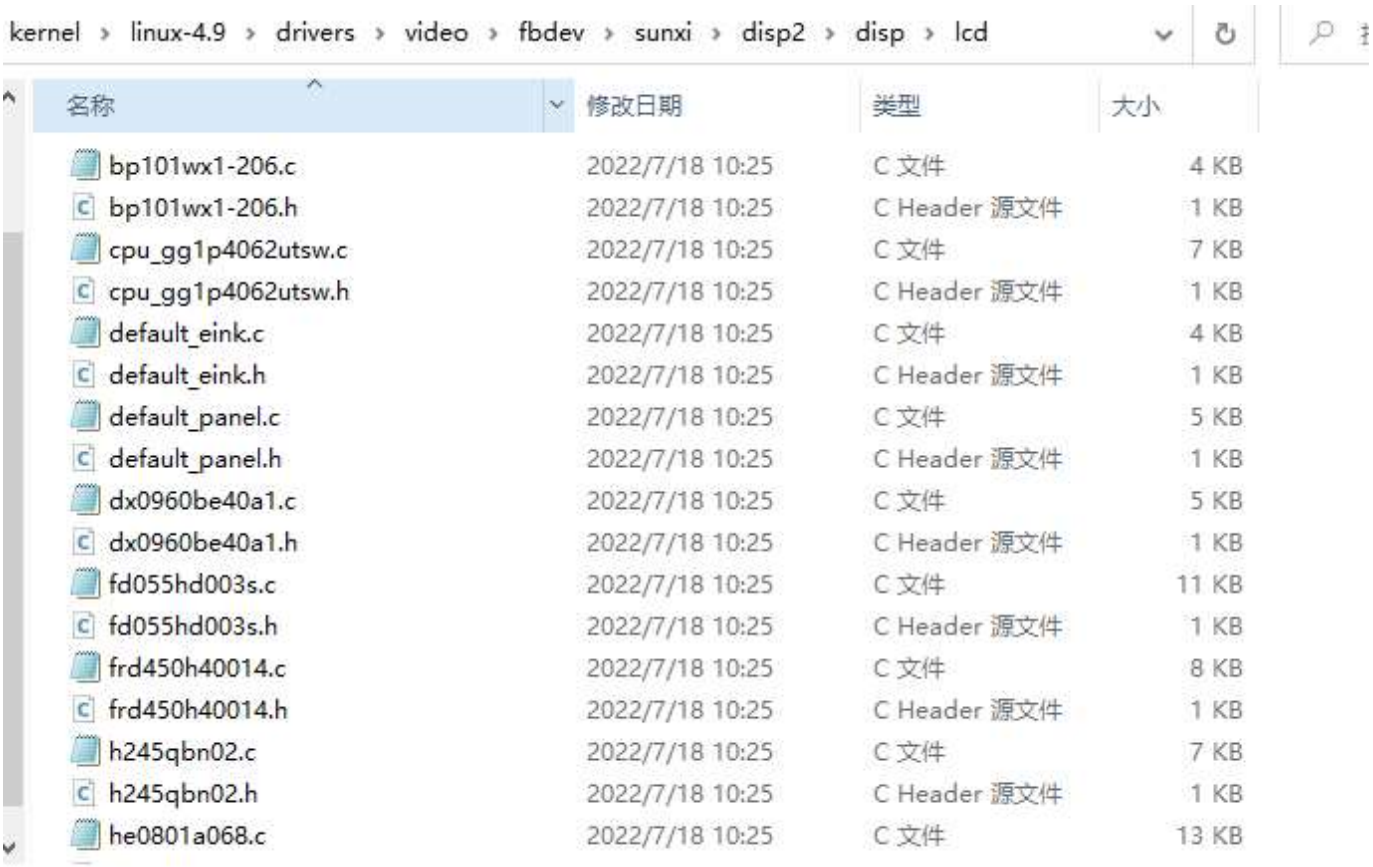
LCD 驱动

LCD 显示屏与其他驱动不一样，LCD 屏幕种类繁多，接口丰富，各式各样屏幕参数层出不穷。所以 LCD 屏幕驱动都是以单独的模块存在的，驱动文件位于：

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

对于 Tina Linux 5.0 以上的设备，驱动文件位于

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



从接口上来分，LCD 屏幕可以分为 RGB 屏幕，LVDS 屏幕，MIPI DSI 屏幕，I8080 屏幕，eDP 屏幕、SPI 屏幕、IIC 屏幕。这里讲解几个常用的屏幕。

(1) RGB 接口

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，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。RGB 屏幕有许多格式，不同的位宽，不同的时钟周期。下表是位宽与时钟周期的区别。

位宽	时钟周期数	颜色数量和格式	并行\串行 RGB
24 bits	1 cycle	16.7M colors, RGB888	并行
18 bits	1 cycle	262K colors, RGB666	并行
16 bits	1 cycle	65K colors, RGB565	并行
6 bits	3 cycles	262K colors, RGB666	串行
6 bits	3 cycles	65K colors, RGB565	串行

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。

（2）MIPI DSI 接口

MIPI-DSI，即 Mobile Industry Processor Interface Display Serial Interface，移动通信行业处理器接口显示串行接口。MIPI 有 2 种模式：

1. Command mode，类似MPU接口，需要IC内部有GRAM来缓冲。
2. Video mode。类似 RGB 接口，没有 GRAM，需要不停往 panel 刷数据。其中 video mode 又分为三个子 mode：

- Non-burst mode with sync pulses
- Non Burst mode with sync Events
- Burst mode。简单理解就是有效数据比率更高，传输效率更高。

MIPI-DSI 的管脚是差分的，分为两种管脚：一种是时钟管脚，另外一种是数据管脚。数据管脚的数量是可变的，数量的单位是 `lane` ， `lane` 也指一对差分管脚，每一条 `lane` 实际包含两条线。一般来说 LCD 屏说明书里面的说的 `lane` 的数量是指数据管脚的数量不包括时钟管脚。比如说某 `4 lane MIPI-DSI` 屏就总共有 `(4+1)*2` 根脚。

（3）LVDS 屏幕

LVDS 即 Low Voltage Differential Signaling 是一种低压差分信号接口。

由于 LVDS 不具备传输图像数据之外的能力，一般屏端不需要任何初始化，只需要初始化 SoC 端即可。

（4）I8080 屏幕

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6种：

- 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 0 1 0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，总共有 65K 这么多种颜色
2. RGB666，总共有 262K 那么多种颜色
3. 9bit 固定为 262K

（5）SPI 屏幕

SPI LCD 是使用 SPI 总线传输图像数据的屏幕，只会出现在很低分辨率的屏幕上。一般来说开屏前都需要初始化操作。SPI LCD 屏幕一般不使用 DISP 来驱动屏幕，使用自己独立的驱动即可。例如 `FBTFT` 驱动。

Linux Kernel 适配 LCD 屏幕

适配 LCD 屏幕的步骤

1. 确保全志显示框架的内核配置有使能
2. 前期准备以下资料和信息：
    - 屏手册。主要是描述屏基本信息和电气特性等，向屏厂索要。
    - Driver IC 手册。主要是描述屏 IC 的详细信息。这里主要是对各个命令进行详解，对我们进行初始化定制有用，向屏厂索要。
    - 屏时序信息。请向屏厂索要。
    - 屏初始化代码，请向屏厂索要。一般情况下 DSI 和 I8080 屏等都需要初始化命令对屏进行初始化。
    - 万用表。调屏避免不了测量相关电压。

3. 通过第2步屏厂提供的资料，定位该屏的类型，然后选择一个已有同样类型的屏驱动作为模板进行屏驱动添加或者直接在上面修改。
4. 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 `strcut __lcd_panel` 的变量指针。 `panel.h` 中新增 `strcut __lcd_panel` 的声明。
5. 修改Makefile。在lcd屏驱动目录的上一级的 `Makefile` 文件中的 `disp-objs` 中新增刚才添加屏驱动.o
6. 修改 `board.dts` 中的 `lcd0` 节点。
7. 编译测试

关闭 U-Boot 加快调试

适配屏幕之前，我们可以先关闭 Uboot 的屏幕驱动，保留 Kernel 的驱动方便调试

先前往 Uboot 配置文件夹修改配置文件关闭屏幕驱动。

```
brandy/brandy-2.0/u-boot-2018/configs/sun8iw21p1_defconfig
```

把 `CONFIG_DISP2_SUNXI=y` 注释了

```
#
# SUNXI LOGO DISPLAY
#
CONFIG_CMD_SUNXI_BMP=y
# CONFIG_SUNXI_SPINOR_BMP is not set
# CONFIG_ENABLE_ADVERT_PICTURE is not set
# CONFIG_SUNXI_SPINOR_JPEG is not set
# CONFIG_CMD_SUNXI_JPEG is not set
#CONFIG_DISP2_SUNXI=y
# CONFIG_VDPO_DISP2_SUNXI is not set
# CONFIG_TV_DISP2_SUNXI is not set
# CONFIG_EDP_DISP2_SUNXI is not set
# CONFIG_EINK200_SUNXI is not set

#
# LCD panels selec
```

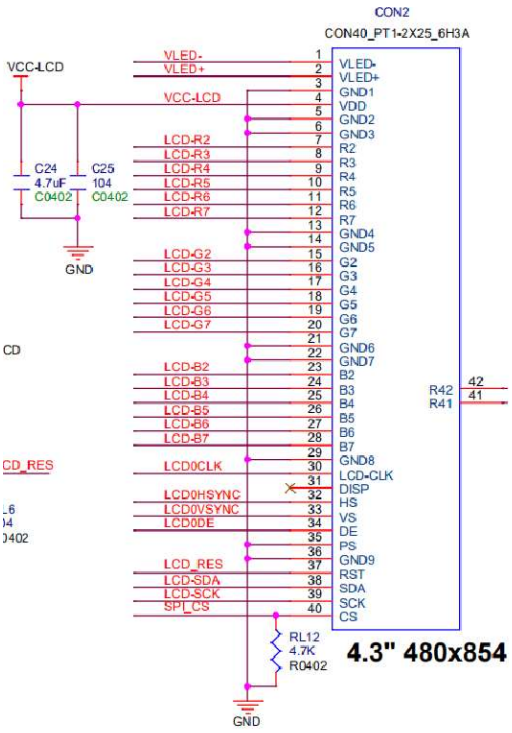
适配 RGB 屏幕

下面简述如何移植一款 RGB 屏幕，使用的开发板是 V853 开发板，屏幕型号是【D500T7009VC】，此 RGB 屏幕无需初始化。使用40Pin 排线，RGB666 连接开发板（可以看见，开发板的 RGB LCD 有一根飞线，飞线后文有说明）

由于此 RGB 屏幕不需要初始化，所以直接使用 `default_lcd` 驱动即可，在 kernel 内 `default_lcd` 是默认启用的，所以只需要修改设备树即可。

硬件连接

硬件连接如电路图与 LCD 屏幕手册所示：



Pin No.	Symbol	Description	When not in use
1	LEDK	B/L negative pin	--
2	LEDA	B/L positive pin	--
3	GND	Ground	--
4	VCC	Power supply	--
5-12	R0-R7	R data input	--
13-20	G0-G7	G data input	--
21-28	B0-B7	B data input	--
29	GND	Ground	--
30	CLK	Clock input	--
31	DISP	Display control / standby mode selection. DISP = "Low" : Standby; (Default) DISP = "High" : Normal display	--
32	HSYNC	Horizontal sync input	--
33	VSYNC	Vertical sync input	--
34	DEN	Data input enable. Active High to enable the data input.	Open
35	NC	Dummy	Open
36	GND	Ground	--
37	XR(NC)		Open
38	YD(NC)		Open
39	XL(NC)		Open
40	YU(NC)		Open

可以看到，屏厂提供的手册中的 31 脚是 DISP 脚，默认 LOW 模式也就是关闭显示输出。而开发板侧的 LCD 屏幕连接器的 DISP 脚是悬空的。如果不飞线会导致屏幕关闭显示输出而不显示。不过也有的屏幕不需要 DISP 信号控制。根据屏幕而定。

设备树配置

首先，我们在 `pio` 节点内增加 `rgb18_pin` 作为 LCD 屏幕的 Pin 绑定。

```
&pio {
    rgb18_pins_a: rgb18@0 {
        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,function = "lcd";
        allwinner,muxsel = <2>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;
    };

    rgb18_pins_b: rgb18@1 {
        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,function = "rgb18_suspend";
        allwinner,muxsel = <0xf>;
        allwinner,drive = <1>;
        allwinner,pull = <0>;
    };
};
```

然后修改 `lcd0` 节点。这里将其分为几个部分，下文会对这几个部分着重说明。另外比较重要的配置项添加了注释。

```
&lcd0 {
    lcd_used            = <1>;          # 启用lcd

    lcd_driver_name     = "default_lcd"; # 使用 default_lcd 驱动
    lcd_if              = <0>;          # 0:rgb 4:dsi

    # Part 1
    lcd_x               = <800>;        # 宽度
    lcd_y               = <480>;        # 高度
    lcd_width           = <108>;        # 屏幕物理宽度，单位 mm
    lcd_height          = <65>;         # 屏幕物理高度，单位 mm

    # Part 2
    lcd_pwm_used        = <1>;          # 启用背光 PWM
    lcd_pwm_ch          = <9>;          # 使用 PWM 通道 9
    lcd_pwm_freq        = <50000>;      # PWM 频率，单位 Hz
    lcd_pwm_pol         = <0>;          # 背光 PWM 的极性
    lcd_pwm_max_limit   = <255>;        # 背光 PWM 的最大值（<=255）

    # Part 3
    lcd_dclk_freq       = <24>;         # 屏幕时钟，单位 MHz
    lcd_ht              = <816>;        # hsync total cycle(pixel)
    lcd_hbp             = <12>;         # hsync back porch(pixel) + hsync plus
width(pixel);
    lcd_hspw            = <4>;          # hsync plus width(pixel)
    lcd_vt              = <496>;        # vsync total cycle(line)
    lcd_vbp             = <12>;         # vsync back porch(line) + vysnc plus
width(line)
    lcd_vspw            = <4>;          # vsync plus width(pixel)

    # Part 4
    lcd_lvds_if         = <0>;
    lcd_lvds_colordepth = <1>;
    lcd_lvds_mode       = <0>;
    lcd_frm             = <0>;          # 0:关闭; 1:启用rgb666抖动; 2:启用rgb656抖动
    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>;

    # Part 5
    pinctrl-0 = <&rgb18_pins_a>;
    pinctrl-1 = <&rgb18_pins_b>;
};
```

Part 1

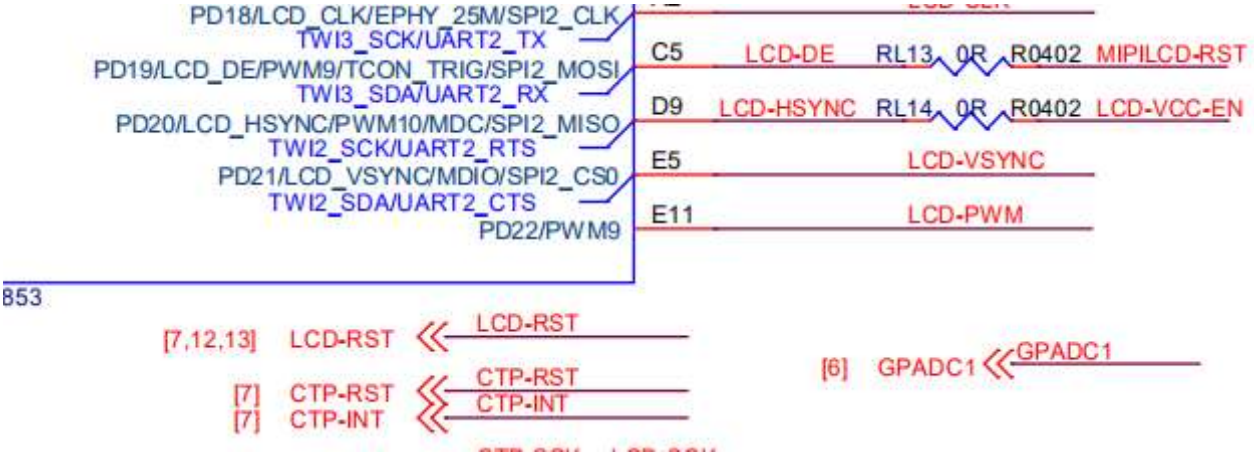
在这一部分中，我们设置了屏幕的像素宽度、高度，物理宽度、高度，这些数据都可以在屏厂提供的数据手册中查询到。有小数点的可以遵循四舍五入原则。

3.Mechanical Specification

Item	Specifications	Unit
Dimensional outline	120.7(W)*75.8(H)*2.83(T) (exclude FPC)	mm
Resolution	800(RGB)*480	Dots
Active area	108(W)*64.8(H)	mm
Dot Pitch	0.135 (W)*0.135(V)	mm
ASSY.TYPE	COG+FPC	--
WEIGHT	TBD	g

Part 2

在这一部分，我们配置了 LCD 屏幕的背光相关属性，使用 PWM 背光实现动态调整背光，可以在电路图中看到 LCD-PWM 使用的是 PWM9，所以配置 `lcd_pwm_ch = <9>;`



Part 3

这一部分是非常重要的一部分，能不能点亮 LCD 屏幕都靠这一部分。

我们打开屏幕手册，找到时序介绍这一部分。

8. Timing Characteristics

Please refer to the IC DATA SHEET of this module!

Parallel 24-bit RGB Interface Timing Table							
Item		Symbol	Min.	Typ.	Max.	Unit	Remark
DCLK Frequency		Fclk	23	25	27	MHz	
HSYNC	Period Time	Th	808	816	896	DCLK	
	Display Period	Thdisp		800		DCLK	
	Back Porch	Thbp	4	8	48	DCLK	
	Front Porch	Thfp	4	8	48	DCLK	
	Pulse Width	Thw	2	4	8	DCLK	
VSYNC	Period Time	Tv	488	496	504	HSYNC	
	Display Period	Tvdisp		480		HSYNC	
	Back Porch	Tvbp	4	8	12	HSYNC	
	Front Porch	Tvfp	4	8	12	HSYNC	
	Pulse Width	Tvw	2	4	8	HSYNC	

（1）DCLK

首先是 DCLK，这里显示的 DCLK 值是 25，需要注意的是，如果直接设置 `lcd_dclk_freq = <25>;` 会导致实际的频率变为 48MHz。这是因为当 DCLK 的频率小于 48MHz 时，其频率是从 288MHz 的主时钟分频到实际频率的。而 25MHz 无法被完整分频导致其使用下一个频点 48M，从而导致屏幕被超频的情况。

这里简单说明下分频系数与得到的频率的计算方法：

```
[DISP] lcd_clk_config,line:702:
disp 0, clk: pll(144000000),clk(144000000),dclk(24000000) dsi_rate(144000000)
     clk real:pll(288000000),clk(288000000),dclk(48000000) dsi_rate(0)
uart0: ttyS0 at MMIO 0x2500000 (irq = 289, base_baud = 1500000) is a SUNXI
```

在开机时，如果使用的是 RGB 屏幕，我们可以看到这样的输出：

```
disp 0, clk: pll(144000000),clk(144000000),dclk(24000000) dsi_rate(144000000)
     clk real:pll(288000000),clk(288000000),dclk(48000000) dsi_rate(0)
```

实际芯片的 LCD 是通过 PLL 时钟分频得到的。所以在这里会计算分频的分频值。使用 `dclk(24000000)` 的 24MHz 乘上倍频系数 `6`，得到 `pll(144000000)` 也就是144MHz，使用 `pll(144000000)` 去申请最近的时钟，这里申请到的是 `real:pll(288000000)` 也就是 288MHz。此时使用 `(int)(real:pll / pll)` 即可获得分频系数。由于不执行浮点运算，输出的结果会向下取

整。例如这里的分频系数是 `288MHz / 144MHz = 2` 。使用分频系数乘上倍频系数即可得到 288MHz 的实际分频结果，这里则是 `288MHz / (2 * 6) = 24MHz` 刚刚好。

如果我们设置为25MHz，按照上面的计算可知分频系数为 1，分频结果则是 `288MHz / (1 * 6) = 48MHz`

```
[DISP] lcd_clk_config,line:702:
disp 0, clk: pll(150000000),clk(150000000),dclk(25000000) dsi_rate(150000000)
     clk real:pll(288000000),clk(288000000),dclk(48000000) dsi_rate(0)
uart0: ttyS0 at MMIO 0x2500000 (irq = 289, base_baud = 1500000) is a SUNXI
sw_console_setup()2050 - console setup baud 115200 parity n bits 8, flow n
console [ttyS0] enabled
```

由于 Display 框架不单单需要支持 RGB 显示屏，也需要支持 MIPI-DSI，LVDS 等等接口的显示屏，并且不同的频率区段所使用的倍频系数也是不同的，所以倍频系数是不可以随意修改的。

在这里，最好的方法就是选取一个最接近的 DCLK 值即可。

48MHz	24MHz	16MHz	12MHz	8MHz	6MHz	4MHz

而对于 48MHz 以上的屏幕，由于使用了不同的时钟源，其分频更加精准，无需按照此处方法调优。

（2）HT（Hsync Total）

查询手册可知，HT 的值是 816，这个值不需要修改。填入即可。

（3）HBP（Hsync Back Porch）

查询手册可知，HBP 值为 8，不过由于全志平台的 HBP 的含义是 HBP + HSPW，所以需要加上 HSPW 的值填入。这里便是 `8 + 4 = 12`

（4）HSPW（Hsync Plus Width）

查询手册可知，HSPW 的值是 4，这个值不需要修改。填入即可。

（5）VT（Vsync Total）

查询手册可知，HT 的值是 496，这个值不需要修改。填入即可。

（6）VBP（Vsync Back Porch）

查询手册可知，VBP 值为 8，不过由于全志平台的 VBP 的含义是 VBP + VSPW，所以需要加上 VSPW 的值填入。这里便是 `8 + 4 = 12`

（7）VSPW（Vsync Plus Width）

查询手册可知，VSPW 的值是 4，这个值不需要修改。填入即可。

可以看到手册还提供了 HFP（Hsync Front Porch）和 VFP（Vsync Front Porch）的值。这个值不需要填写，因为驱动可以通过计算得出实际的值。

Part 4

这一部分是 LCD 屏幕扩展功能的区域，包括 LCD 屏幕的各类功能，可以参照手册内容设置。

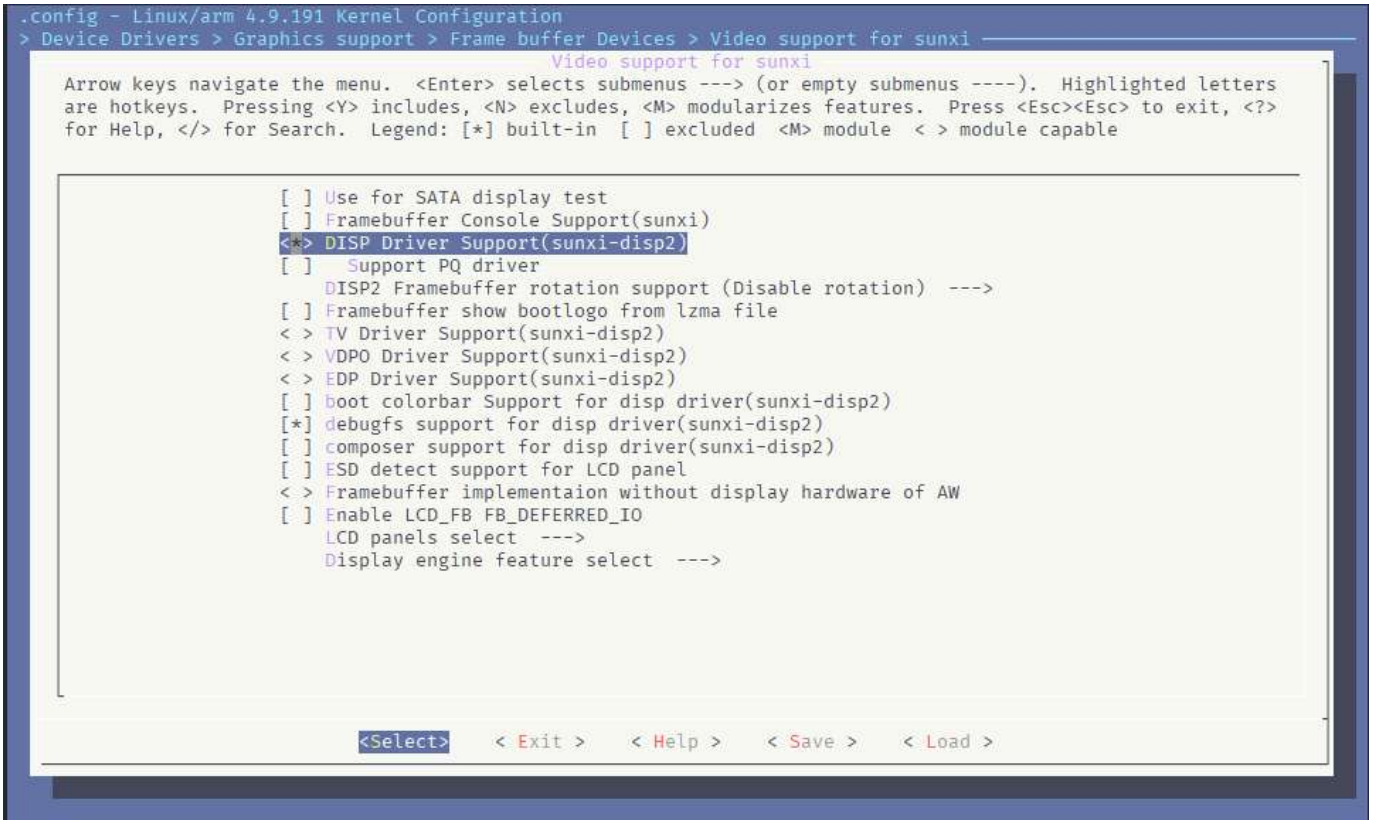
Part 5

这一部分是 Pin 的绑定，绑定上面创建的两个 RGB 节点。GPIO 与 Pin 绑定相关请查阅：【GPIO - V853】

驱动勾选

不需要初始化的 RGB LCD 的驱动比较简单，勾选 `Video support for sunxi` 即可

```
Device Drivers > Graphics support > Frame buffer Devices > Video support for sunxi
```



测试屏幕

首先编译，打包。烧录系统。

先 `ls` 命令打印 `/dev/` 目录看看有没有出现 `fb0` 这个节点

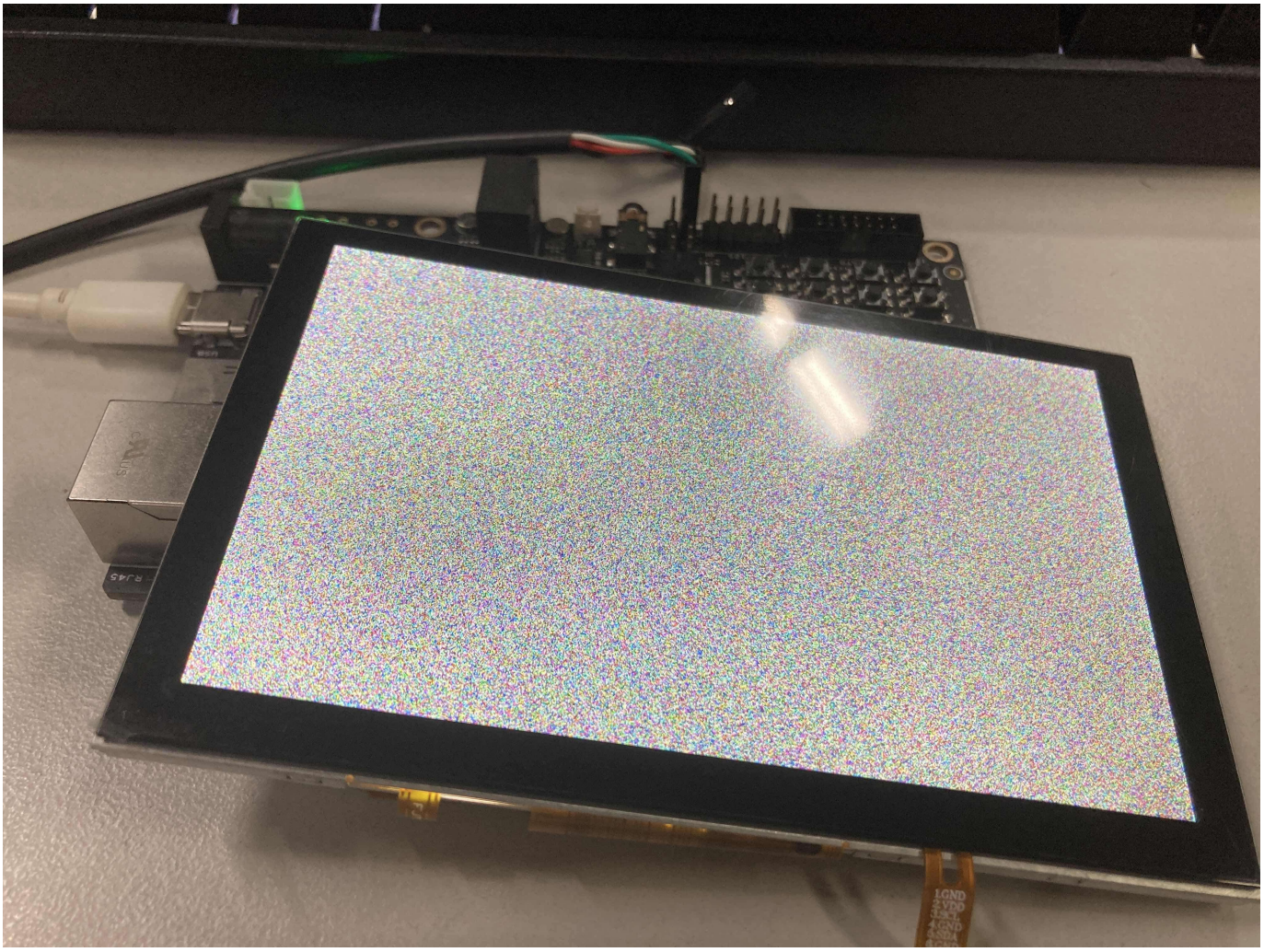
```
root@TinaLinux:/# ls /dev/
by-name              snd                  tty40                v4l-subdev15
cedar_dev            sunxi-reg            tty41                v4l-subdev16
console              sunxi-wlan           tty42                v4l-subdev17
cpu_dma_latency      sunxi_soc_info       tty43                v4l-subdev18
disp                 tty                  tty44                v4l-subdev19
fb0                  tty0                 tty45                v4l-subdev2
full                 tty1                 tty46                v4l-subdev20
g2d                  tty10                tty47                v4l-subdev21
gpiochip0            tty11                tty48                v4l-subdev22
i2c-0                tty12                tty49                v4l-subdev23
i2c-1                tty13                tty5                 v4l-subdev24
i2c-2                tty14                tty50                v4l-subdev25
i2c-3                tty15                tty51                v4l-subdev26
i2c-4                tty16                tty52                v4l-subdev27
input                tty17                tty53                v4l-subdev28
ion                  tty18                tty54                v4l-subdev29
kmsg                 tty19                tty55                v4l-subdev3
media0               tty2                 tty56                v4l-subdev30
memory_bandwidth     tty20                tty57                v4l-subdev31
mmcblk0              tty21                tty58                v4l-subdev32
mmcblk0boot0         tty22                tty59                v4l-subdev33
mmcblk0boot1         tty23                tty6                 v4l-subdev34
mmcblk0p1            tty24                tty60                v4l-subdev4
mmcblk0p2            tty25                tty61                v4l-subdev5
mmcblk0p3            tty26                tty62                v4l-subdev6
mmcblk0p4            tty27                tty63                v4l-subdev7
mmcblk0p5            tty28                tty7                 v4l-subdev8
mmcblk0p6            tty29                tty8                 v4l-subdev9
mmcblk0p7            tty3                 tty9                 vcs
mmcblk0rpmb          tty30                ttyS0                vcs1
network_latency      tty31                ttyS3                vcsa
network_throughput   tty32                urandom              vcsa1
null                 tty33                usb-ffs              vipcore
ptmx                 tty34                v4l-subdev0          watchdog
pts                  tty35                v4l-subdev1          watchdog0
random               tty36                v4l-subdev10         watchdog1
rfkill               tty37                v4l-subdev11         wiegand driver
rtc0                 tty38                v4l-subdev12         zero
shm                  tty39                v4l-subdev13
sid_efuse            tty4                 v4l-subdev14
root@TinaLinux:/#
```

可以看到这里有 `fb0`` 节点。那就进行下最简单的花屏测试。

```
cat /dev/urandom > /dev/fb0
```

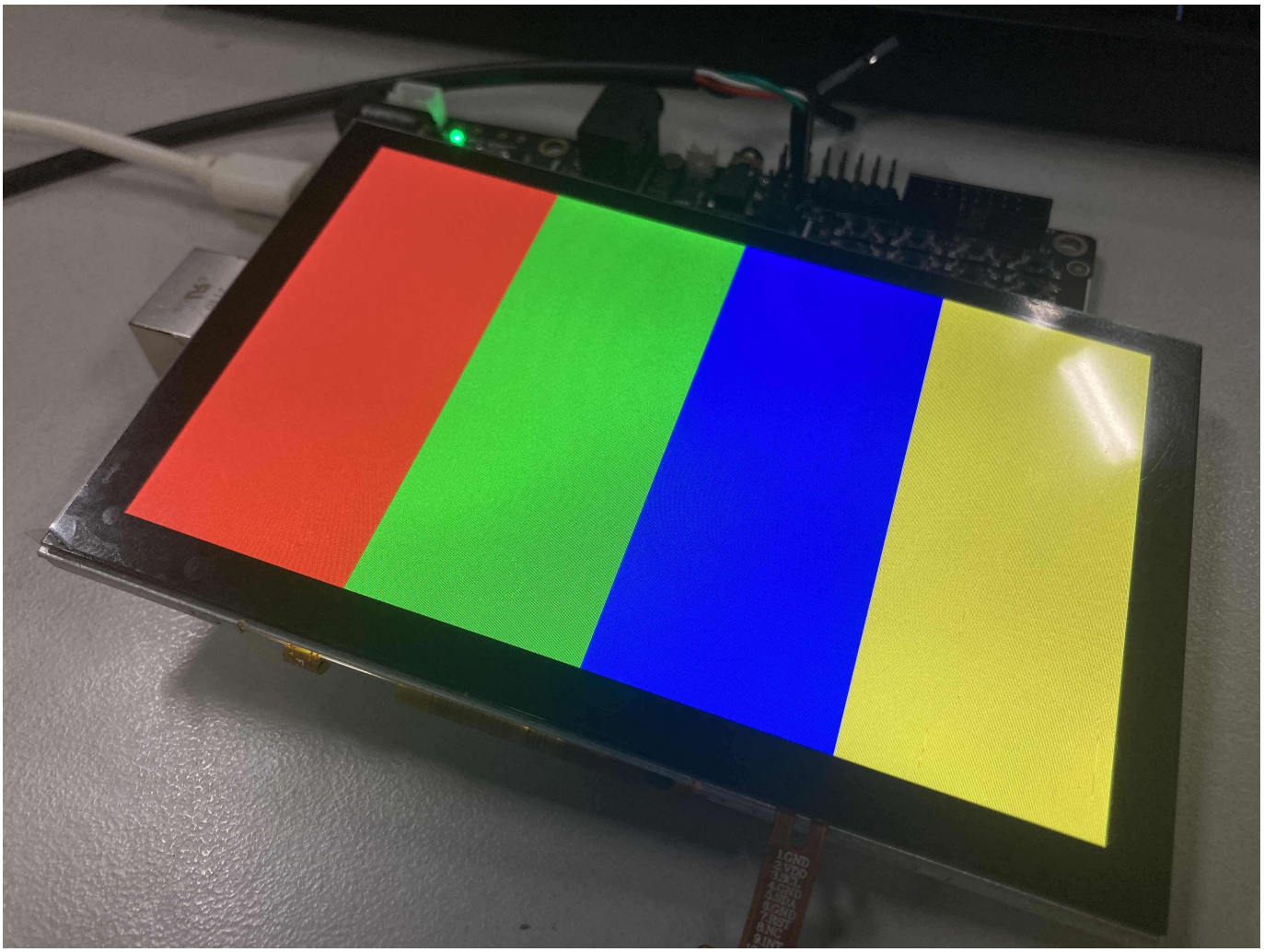
```
sid_efuse           tty4                v4l-subdev14
root@TinaLinux:/# cat /dev/urandom > /dev/fb0
cat: write error: No space left on device
root@TinaLinux:/#
```

这个测试一定会显示 `cat: write error: No space left on device` ，这样才是正常情况，因为 FB0 可以类比为屏幕的显存，显存是固定大小的可以被消耗完，当显存填满的时候就会报错 `No space left on device` 。如果执行这一行命令一直没出现这个报错则有可能底层显示驱动配置有问题。



另外也可以使用 `colorbar` 测试

```
echo 8 > /sys/class/disp/disp/attr/colorbar
```



也可以使用，获取屏幕的图层信息，帧率等等。

```
cat /sys/class/disp/disp/attr/sys
```

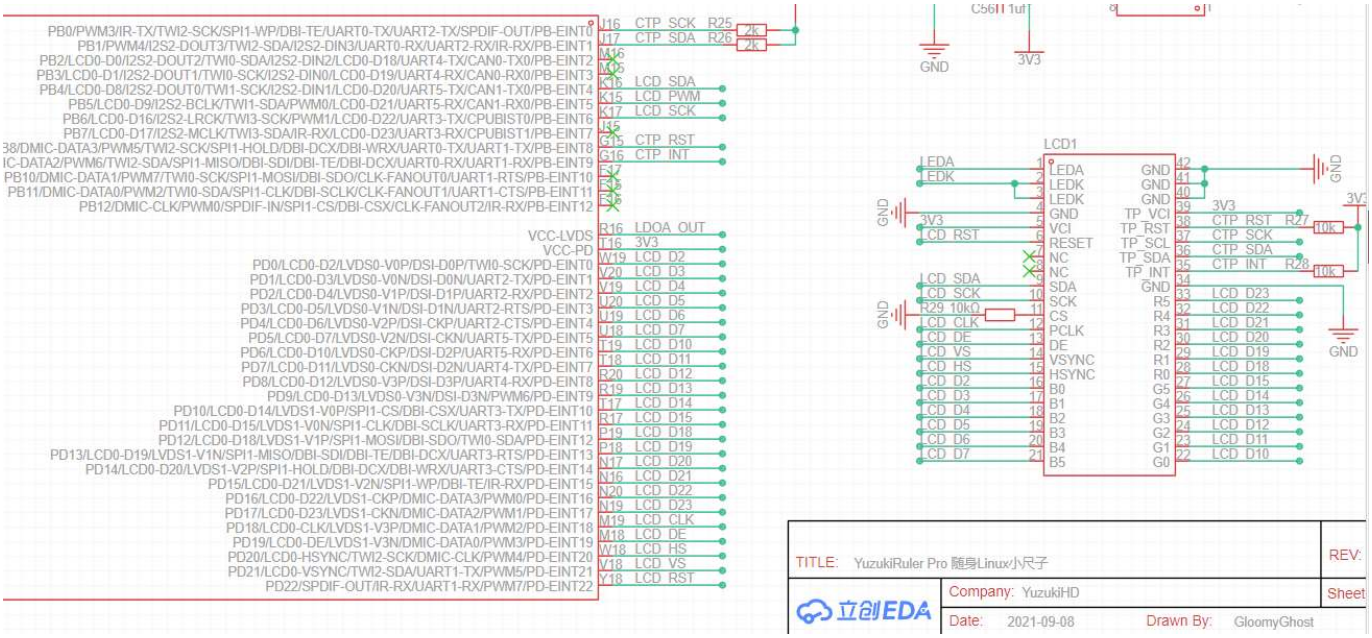
```
root@TinaLinux:/# cat /sys/class/disp/disp/attr/sys
screen 0:
de_rate 300000000 hz, ref_fps:59
mgr0: 800x480 fmt[rgb] cs[0x204] range[full] eotf[0x4] bits[8bits] err[0] force_sync[0] unblank direct_sho
dmabuf: cache[0] cache max[0] umap skip[0] umap skip max[22]
        lcd output      backlight(197)  fps:60.9         800x 480
        err:0   skip:87 irq:3830        vsync:0 vsync_skip:0
   BUF    enable ch[1] lyr[0] z[16] prem[N] a[pixel 255] fmt[  0] fb[ 800, 480; 800, 480; 800, 480] crop[
    0] flags[0x       0] trd[0,0]
depth[ 0] root@TinaLinux:/#
```

适配 RGB + SPI 初始化 屏幕

有些 RGB 屏幕需要使用 SPI 进行初始化，设置屏幕的 GAMMA，亮度，工作模式等。开屏前需要使用 SPI 发送数据初始化屏幕的寄存器，再使用 RGB 进行图像的输出。

适配前的准备

这里示例的开发板是 【YuzukiRuler Pro 随身Linux小尺子】，其硬件连接如下所示：



这里使用的屏幕是 【TL032FWV01-I1440A】驱动是 ST7701s，询问厂家得到了三份资料。

lcd.h
TL032FWV01-I1440A_specification(3).pdf
TL032FWV01-I1440A_ST7701S(SPI9bit+RGB16bit)G2.2_V1.0(1).INI

lcd.h

在这个文件中，定义了屏幕的一些时序信息。

```
/*--LCD部分定义---------------------------*/
#define hsize 1024
#define vsize 600

#define HBP         251
#define HT          576
#define HSPW        5
#define VBP         100
#define VT          930
#define VSPW        10
#define PCLK_CLK     35     //单位为MHZ 例如我定义了35  那就是35MHZ    //45


/*---SSD2828 PCLK 输出设置-----------------------*/

#define MIPI_CLK     0X8228  //MIPI信号输出CLK          //8228
```

TL032FWV01-I1440A_specification(3).pdf

在这个文件中，定义了屏幕的长宽高，分辨率等等

Feature		Specifications
Display Spec.	Display Size(Diagonal)	3.16 inch
	Resolution (H*V)	320(RGB）×820
	LCD type	a-Si TFT
	Pixel Configuration	R.G.B. Vertical Stripe
	Display Mode	IPS/Transmissive/Normally Black
	Viewing Direction	All
Mechanical Characteristics	OutlineDimensions (H xV x T) (mm)	31.2(H) ×76.6(V)×2.4(T)
	Active Area(mm)	26.4 (H)×67.65(V)
	With /Without Touch screen	Without
	Match Connector Type	0.5 PITCH 40PIN
	Backlight Type	White LED
	Weight (g)	TBD
Electrical Characteristics	Interface	3SPI+18RGB
	Number of color	262K
	Driver IC	ST7701S

TL032FWV01-I1440A_ST7701S(SPI9bit+RGB16bit)G2.2_V1.0(1).INI

这个文件里定义了 SPI 的初始化时序信息，这里的 `W_C` 代表的写命令， `W_D` 代表写数据。 `Delay` 代表延时。

```
256    W_C (0x11);
257
258    Delay(120);
259
260    W_C (0xFF);
261    W_D (0x77);
262    W_D (0x01);
263    W_D (0x00);
264    W_D (0x00);
265    W_D (0x13);
266
267    W_C (0xE8);
268    W_D (0x00);
269    W_D (0x0C);
270
271    Delay(10 );
272
273    W_C (0xE8);
274    W_D (0x00);
275    W_D (0x00);
276
277    W_C (0xFF);
278    W_D (0x77);
279    W_D (0x01);
280    W_D (0x00);
281    W_D (0x00);
282    W_D (0x00);
283
284    W_C (0x3A);
285    W_D (0x55);
286
287    W_C (0x36);
288    W_D (0x00);
289
290    W_C (0x35);
291    W_D (0x00);
292
293    W_C (0x29);
294
```

编写驱动程序

想要驱动需要初始化的屏幕，就需要使用屏幕驱动了。

对于 4.9 内核，屏幕驱动位于以下文件夹内

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

对于 5.4 内核，屏幕驱动位于以下文件夹内

```
lichee/linux-5.4/drivers/video/fbdev/sunxi/disp2/disp/lcd
```

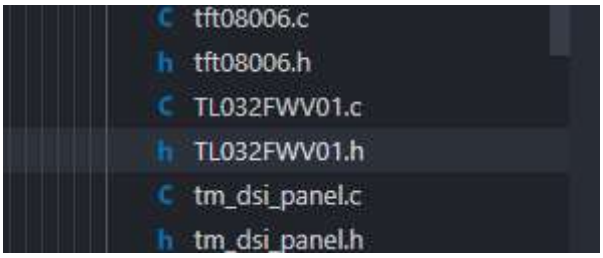
从现成的驱动开始

由于是 RGB + SPI 的屏幕，可以到屏幕驱动中找一份现成的相同驱动方式的驱动文件来修改，这里我们使用的是 `st7789v.c` 驱动，查看驱动可知他使用的是 `spi` 驱动模式

具体的 SPI 通讯的方法是使用函数模拟 SPI 的时序。并没有实际使用 SPI 外设。这样的好处是可以使用任意 GPIO 引脚，且初始化完成即可释放 GPIO。

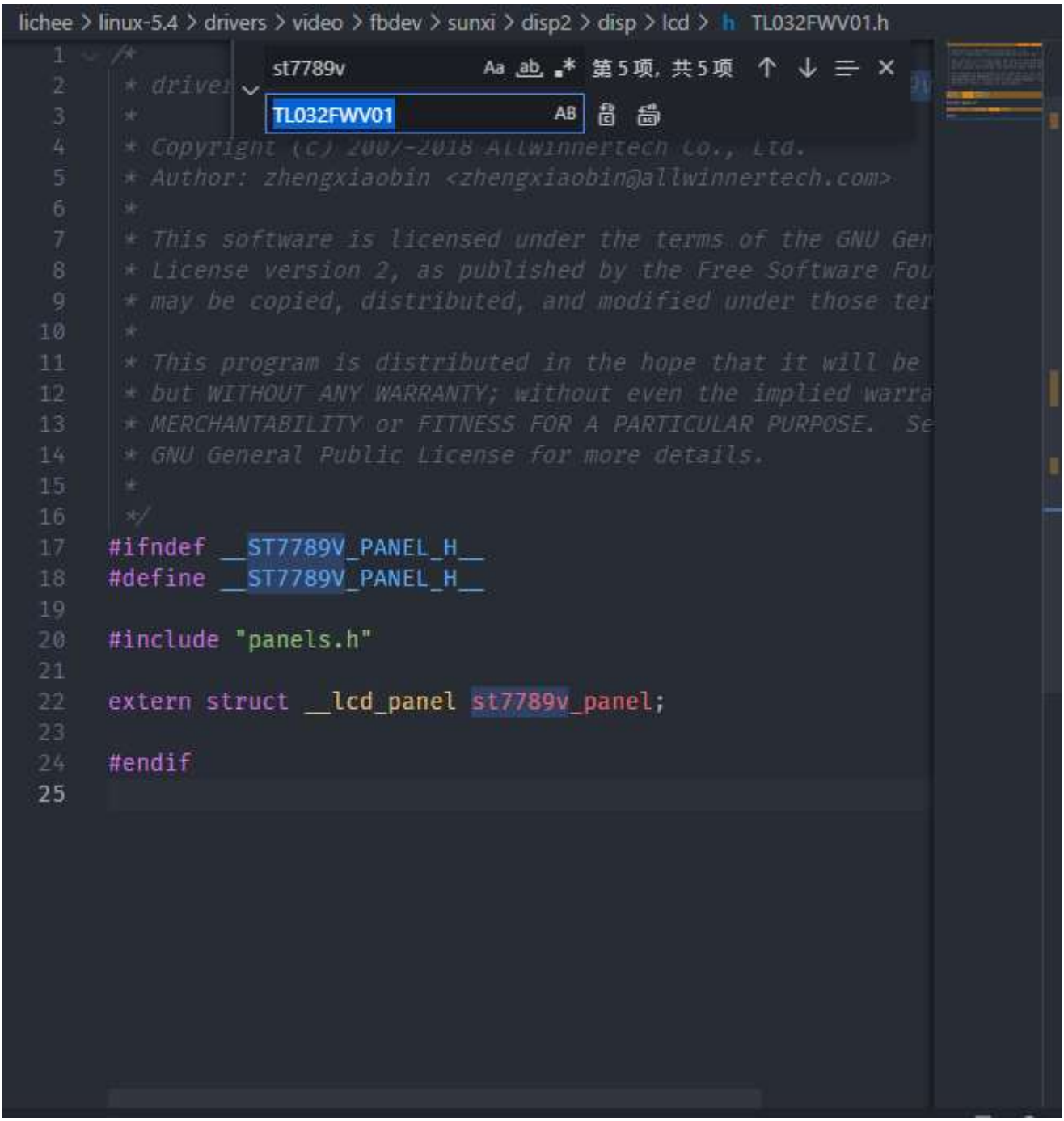
```c
static void st7789v_spi_write_cmd(u8 value)
{
    int i;
    spi_cs_0;
    spi_scl_0;
    spi_sdi_0;
    spi_scl_1;
    spi_scl_0;
    for (i = 0; i < 8; i++) {
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        value <<= 1;
        spi_scl_1;
        spi_scl_0;
    }
    spi_cs_1;
}

static void st7789v_spi_write_data(u8 value)
{
    int i;
    spi_cs_0;
    spi_scl_0;
    spi_sdi_1;
    spi_scl_1;
    spi_scl_0;
    for (i = 0; i < 8; i++) {
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        value <<= 1;
        spi_scl_1;
        spi_scl_0;
    }
    spi_cs_1;
}
```

首先为了方便区分，我们复制一份 `st7789v` 驱动，重命名为 `TL032FWV01`



并且把驱动内的 st7789v 全部改为 TL032FWV01

头文件也别忘记修改



配置屏幕驱动

第一部分是 `LCD_cfg_panel_info` 这一部分一般不需要修改，保留即可

```c
static void LCD_cfg_panel_info(struct panel_extend_para *info)
{
    u32 i = 0, j = 0;
    u32 items;
    u8 lcd_gamma_tbl[][2] = {
        /* {input value, corrected value} */
        {0, 0},     {15, 15},   {30, 30},   {45, 45},   {60, 60},
        {75, 75},   {90, 90},   {105, 105}, {120, 120}, {135, 135},
        {150, 150}, {165, 165}, {180, 180}, {195, 195}, {210, 210},
        {225, 225}, {240, 240}, {255, 255},
        {225, 225}, {240, 240}, {255, 255},
    };

    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},
        },
    };

    items = sizeof(lcd_gamma_tbl) / 2;
    for (i = 0; i < items - 1; i++) {
        u32 num = lcd_gamma_tbl[i + 1][0] - lcd_gamma_tbl[i][0];

        for (j = 0; j < num; j++) {
            u32 value = 0;

            value =
                lcd_gamma_tbl[i][1] +
                ((lcd_gamma_tbl[i + 1][1] - lcd_gamma_tbl[i][1]) *
                 j) /
                num;
            info->lcd_gamma_tbl[lcd_gamma_tbl[i][0] + j] =
                (value << 16) + (value << 8) + value;
        }
    }
    info->lcd_gamma_tbl[255] = (lcd_gamma_tbl[items - 1][1] << 16) +
                    (lcd_gamma_tbl[items - 1][1] << 8) +
                    lcd_gamma_tbl[items - 1][1];

    memcpy(info->lcd_cmap_tbl, lcd_cmap_tbl, sizeof(lcd_cmap_tbl));
}
```

第二部分是 LCD 上电下电相关，默认即可

```c
static s32 LCD_open_flow(u32 sel)
{
    /* open lcd power, and delay 50ms */
    LCD_OPEN_FUNC(sel, LCD_power_on, 200);
    /* open lcd power, than delay 200ms */
    LCD_OPEN_FUNC(sel, LCD_panel_init, 200);
    /* open lcd controller, and delay 100ms */
    LCD_OPEN_FUNC(sel, sunxi_lcd_tcon_enable, 150);
    /* open lcd backlight, and delay 0ms */
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0);

    return 0;
}

static s32 LCD_close_flow(u32 sel)
{
    /* close lcd backlight, and delay 0ms */
    LCD_CLOSE_FUNC(sel, LCD_bl_close, 50);
    /* close lcd controller, and delay 0ms */
    LCD_CLOSE_FUNC(sel, sunxi_lcd_tcon_disable, 10);
    /* open lcd power, than delay 200ms */
    LCD_CLOSE_FUNC(sel, LCD_panel_exit, 10);
    /* close lcd power, and delay 500ms */
    LCD_CLOSE_FUNC(sel, LCD_power_off, 10);

    return 0;
}
```

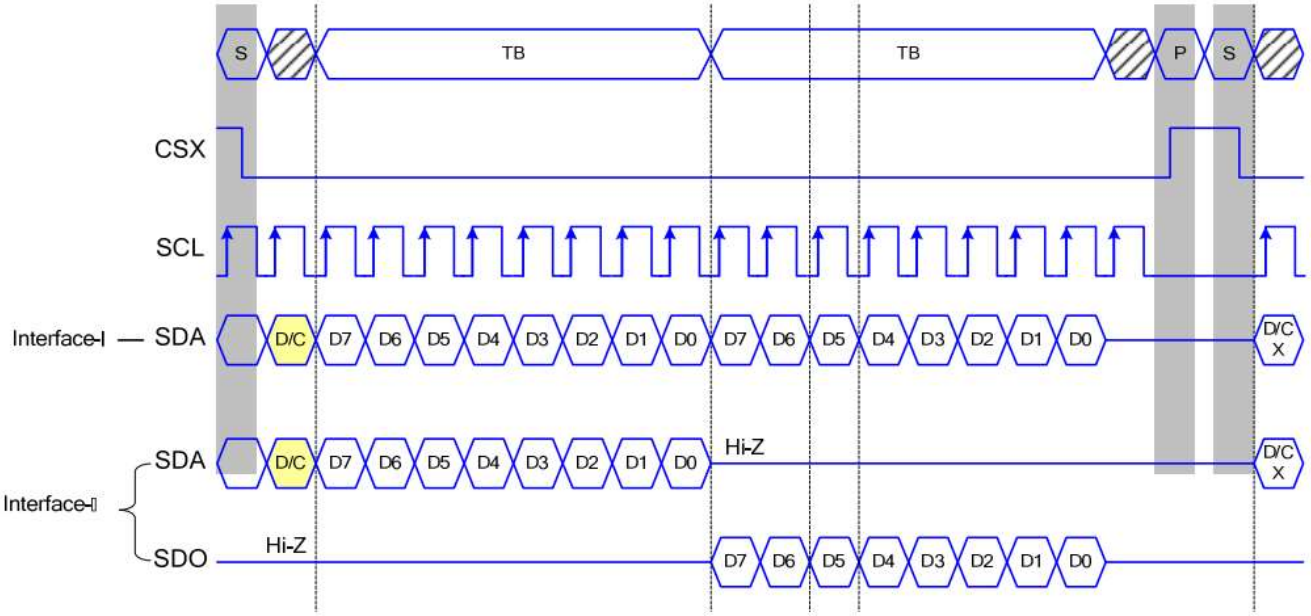
第三部分是屏幕开启关闭与背光相关，对于这款屏幕也不需要修改

```c
static void LCD_power_on(u32 sel)
{
    /* config lcd_power pin to open lcd power0 */
    sunxi_lcd_power_enable(sel, 0);
    /* pwr_en, active low */
    sunxi_lcd_gpio_set_value(sel, 3, 0);
    sunxi_lcd_pin_cfg(sel, 1);
}

static void LCD_power_off(u32 sel)
{
    sunxi_lcd_pin_cfg(sel, 0);
    /* pwr_en, active low */
    sunxi_lcd_gpio_set_value(sel, 3, 1);
    /* config lcd_power pin to close lcd power0 */
    sunxi_lcd_power_disable(sel, 0);
}

static void LCD_bl_open(u32 sel)
{
    sunxi_lcd_pwm_enable(sel);
    /* config lcd_bl_en pin to open lcd backlight */
    sunxi_lcd_backlight_enable(sel);
}

static void LCD_bl_close(u32 sel)
{
    /* config lcd_bl_en pin to close lcd backlight */
    sunxi_lcd_backlight_disable(sel);
    sunxi_lcd_pwm_disable(sel);
}
```

第四部分是 软件模拟 SPI 的部分，这里需要对照 ST7701s 数据手册修改为 ST7701s 的 SPI 时序。

3-line serial interface protocol

3-line serial protocol (for RDID1/RDID2/RDID3/0Ah/0Bh/0Ch/0Dh/0Eh/0Fh command: 8-bit read):



主要添加了延时函数，因为 ST7701s 的 SPI 时钟较低，并修改对应的上升、下降触发方式。

```c
static void TL032FWV01_spi_write_cmd(u32 value)
{
    int i;
    spi_cs_0;
    spi_sdi_0;
    spi_scl_0;
    sunxi_lcd_delay_us(10);
    spi_scl_1;
    for (i = 0; i < 8; i++) {
        sunxi_lcd_delay_us(10);
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        spi_scl_0;
        sunxi_lcd_delay_us(10);
        spi_scl_1;
        value <<= 1;
    }
    sunxi_lcd_delay_us(10);
    spi_cs_1;
}

static void TL032FWV01_spi_write_data(u32 value)
{
    int i;
    spi_cs_0;
    spi_sdi_1;
    spi_scl_0;
    sunxi_lcd_delay_us(10);
    spi_scl_1;
    for (i = 0; i < 8; i++) {
        sunxi_lcd_delay_us(10);
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        value <<= 1;
        sunxi_lcd_delay_us(10);
        spi_scl_0;
        spi_scl_1;
    }
    sunxi_lcd_delay_us(10);
    spi_cs_1;
}
```

```
//three line 9bit mode
static void TL032FWV01_spi_write_cmd(u32 value)
{
    int i;
    spi_cs_0;
    spi_sdi_0;
    spi_scl_0;
    sunxi_lcd_delay_us(10);
    spi_scl_1;
    for (i = 0; i < 8; i++) {
        sunxi_lcd_delay_us(10);
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        spi_scl_0;
        sunxi_lcd_delay_us(10);
        spi_scl_1;
        value <<= 1;
    }
    sunxi_lcd_delay_us(10);
    spi_cs_1;
}

static void TL032FWV01_spi_write_data(u32 value)
{
    int i;
    spi_cs_0;
    spi_sdi_1;
    spi_scl_0;
    sunxi_lcd_delay_us(10);
    spi_scl_1;
    for (i = 0; i < 8; i++) {
        sunxi_lcd_delay_us(10);
        if (value & 0x80)
            spi_sdi_1;
        else
            spi_sdi_0;
        value <<= 1;
        sunxi_lcd_delay_us(10);
        spi_scl_0;
        spi_scl_1;
    }
    sunxi_lcd_delay_us(10);
    spi_cs_1;
}
```

另外这里的 `SPI` 操作均定义在驱动头部，其中的 `sunxi_lcd_gpio_set_value` 中，1，2，3 来自设备树的 `lcd_gpio_1`，`lcd_gpio_2`，`lcd_gpio_3`。

例如：设备树配置了 `lcd_gpio_1 = <&pio PG 13 GPIO_ACTIVE_HIGH>;` 表示上电时 `PG13` 脚默认高电平，在屏幕驱动里可以使用 `sunxi_lcd_gpio_set_value(0, 1, 1)` 将 `PG13` 脚设置为高电平，`sunxi_lcd_gpio_set_value(0, 1, 0)` 将 `PG13` 脚设置为低电平。

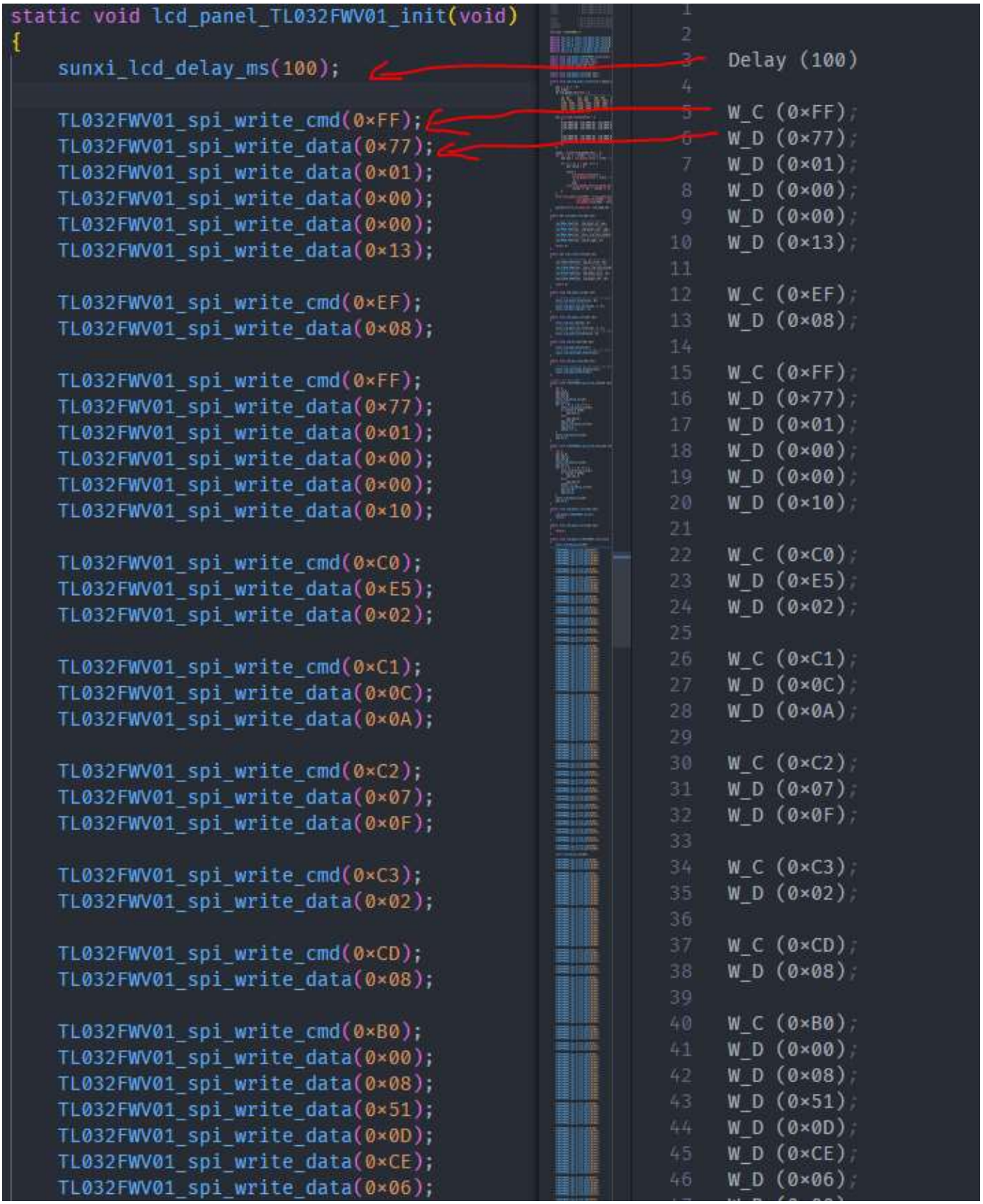
```
*/
#include "TL032FWV01.h"

#define spi_scl_1 sunxi_lcd_gpio_set_value(0, 3, 1)
#define spi_scl_0 sunxi_lcd_gpio_set_value(0, 3, 0)
#define spi_sdi_1 sunxi_lcd_gpio_set_value(0, 2, 1)
#define spi_sdi_0 sunxi_lcd_gpio_set_value(0, 2, 0)
#define spi_cs_1 sunxi_lcd_gpio_set_value(0, 1, 1)
#define spi_cs_0 sunxi_lcd_gpio_set_value(0, 1, 0)

static void lcd_panel_TL032FWV01_init(void);
static void LCD_power_on(u32 sel);
static void LCD_power_off(u32 sel);
static void LCD_bl_open(u32 sel);
static void LCD_bl_close(u32 sel);

static void LCD_panel_init(u32 sel);
static void LCD_panel_exit(u32 sel);

static void LCD_cfg_panel_info(struct panel_extend_para *info)
{
    u32 i = 0, j = 0;
```

第五部分是开关屏函数，原来的开关屏函数包括读取屏幕信息，这里我们不需要读取信息所以直接删除即可。开屏 `init` 部分调用 SPI 初始化屏幕参数即可。

```
static void LCD_panel_init(u32 sel)
{
    lcd_panel_TL032FWV01_init();
    return;
}

static void LCD_panel_exit(u32 sel)
{
    return;
}
```

```
static void LCD_panel_init(u32 sel)
{
    lcd_panel_TL032FWV01_init();
    return;
}

static void LCD_panel_exit(u32 sel)
{
    return;
}
```

接下来是最重要的一部分，屏幕初始化参数部分。这一部分使用 SPI 对屏幕进行初始化，打开屏厂提供的初始化代码，一一对应即可。


static void lcd_panel_TL032FWV01_init(void)
{
sunxi_lcd_delay_ms(100);
TL032FWV01_spi_write_cmd(0×FF);
TL032FWV01_spi_write_data(0×77);
TL032FWV01_spi_write_data(0×01);
TL032FWV01_spi_write_data(0×00);
TL032FWV01_spi_write_data(0×00);
TL032FWV01_spi_write_data(0×13);
TL032FWV01_spi_write_cmd(0×EF);
TL032FWV01_spi_write_data(0×08);
TL032FWV01_spi_write_cmd(0×FF);
TL032FWV01_spi_write_data(0×77);
TL032FWV01_spi_write_data(0×01);
TL032FWV01_spi_write_data(0×00);
TL032FWV01_spi_write_data(0×00);
TL032FWV01_spi_write_data(0×10);
TL032FWV01_spi_write_cmd(0×C0);
TL032FWV01_spi_write_data(0×E5);
TL032FWV01_spi_write_data(0×02);
TL032FWV01_spi_write_cmd(0×C1);
TL032FWV01_spi_write_data(0×0C);
TL032FWV01_spi_write_data(0×0A);
TL032FWV01_spi_write_cmd(0×C2);
TL032FWV01_spi_write_data(0×07);
TL032FWV01_spi_write_data(0×0F);
TL032FWV01_spi_write_cmd(0×C3);
TL032FWV01_spi_write_data(0×02);
TL032FWV01_spi_write_cmd(0×CD);
TL032FWV01_spi_write_data(0×08);
TL032FWV01_spi_write_cmd(0×B0);
TL032FWV01_spi_write_data(0×00);
TL032FWV01_spi_write_data(0×08);
TL032FWV01_spi_write_data(0×51);
TL032FWV01_spi_write_data(0×0D);
TL032FWV01_spi_write_data(0×CE);
TL032FWV01_spi_write_data(0×06);
Delay (100)
W_C (0×FF);
W_D (0×77);
W_D (0×01);
W_D (0×00);
W_D (0×00);
W_D (0×13);
W_C (0×EF);
W_D (0×08);
W_C (0×FF);
W_D (0×77);
W_D (0×01);
W_D (0×00);
W_D (0×00);
W_D (0×10);
W_C (0×C0);
W_D (0×E5);
W_D (0×02);
W_C (0×C1);
W_D (0×0C);
W_D (0×0A);
W_C (0×C2);
W_D (0×07);
W_D (0×0F);
W_C (0×C3);
W_D (0×02);
W_C (0×CD);
W_D (0×08);
W_C (0×B0);
W_D (0×00);
W_D (0×08);
W_D (0×51);
W_D (0×0D);
W_D (0×CE);
W_D (0×06);

```
static void lcd_panel_TL032FWV01_init(void)
{
    sunxi_lcd_delay_ms(100);

    TL032FWV01_spi_write_cmd(0xFF);
    TL032FWV01_spi_write_data(0x77);
    TL032FWV01_spi_write_data(0x01);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x13);

    TL032FWV01_spi_write_cmd(0xEF);
    TL032FWV01_spi_write_data(0x08);

    TL032FWV01_spi_write_cmd(0xFF);
    TL032FWV01_spi_write_data(0x77);
    TL032FWV01_spi_write_data(0x01);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x10);

    TL032FWV01_spi_write_cmd(0xC0);
    TL032FWV01_spi_write_data(0xE5);
    TL032FWV01_spi_write_data(0x02);

    TL032FWV01_spi_write_cmd(0xC1);
    TL032FWV01_spi_write_data(0x0C);
    TL032FWV01_spi_write_data(0x0A);

    TL032FWV01_spi_write_cmd(0xC2);
    TL032FWV01_spi_write_data(0x07);
    TL032FWV01_spi_write_data(0x0F);

    TL032FWV01_spi_write_cmd(0xC3);
    TL032FWV01_spi_write_data(0x02);

    TL032FWV01_spi_write_cmd(0xCD);
    TL032FWV01_spi_write_data(0x08);

    TL032FWV01_spi_write_cmd(0xB0);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x08);
    TL032FWV01_spi_write_data(0x51);
    TL032FWV01_spi_write_data(0x0D);
    TL032FWV01_spi_write_data(0xCE);
    TL032FWV01_spi_write_data(0x06);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x08);
    TL032FWV01_spi_write_data(0x08);
    TL032FWV01_spi_write_data(0x1D);
```

```
TL032FWV01_spi_write_data(0x02);
TL032FWV01_spi_write_data(0xD0);
TL032FWV01_spi_write_data(0x0F);
TL032FWV01_spi_write_data(0x6F);
TL032FWV01_spi_write_data(0x36);
TL032FWV01_spi_write_data(0x3F);

TL032FWV01_spi_write_cmd(0xB1);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x10);
TL032FWV01_spi_write_data(0x4F);
TL032FWV01_spi_write_data(0x0C);
TL032FWV01_spi_write_data(0x11);
TL032FWV01_spi_write_data(0x05);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x07);
TL032FWV01_spi_write_data(0x07);
TL032FWV01_spi_write_data(0x1F);
TL032FWV01_spi_write_data(0x05);
TL032FWV01_spi_write_data(0xD3);
TL032FWV01_spi_write_data(0x11);
TL032FWV01_spi_write_data(0x6E);
TL032FWV01_spi_write_data(0x34);
TL032FWV01_spi_write_data(0x3F);

TL032FWV01_spi_write_cmd(0xFF);
TL032FWV01_spi_write_data(0x77);
TL032FWV01_spi_write_data(0x01);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x11);

TL032FWV01_spi_write_cmd(0xB0);
TL032FWV01_spi_write_data(0x4D);

TL032FWV01_spi_write_cmd(0xB1);
TL032FWV01_spi_write_data(0x1C);

TL032FWV01_spi_write_cmd(0xB2);
TL032FWV01_spi_write_data(0x87);

TL032FWV01_spi_write_cmd(0xB3);
TL032FWV01_spi_write_data(0x80);

TL032FWV01_spi_write_cmd(0xB5);
TL032FWV01_spi_write_data(0x47);

TL032FWV01_spi_write_cmd(0xB7);
TL032FWV01_spi_write_data(0x85);

TL032FWV01_spi_write_cmd(0xB8);
```

```
TL032FWV01_spi_write_data(0x21);

TL032FWV01_spi_write_cmd(0xB9);
TL032FWV01_spi_write_data(0x10);

TL032FWV01_spi_write_cmd(0xC1);
TL032FWV01_spi_write_data(0x78);

TL032FWV01_spi_write_cmd(0xC2);
TL032FWV01_spi_write_data(0x78);

TL032FWV01_spi_write_cmd(0xD0);
TL032FWV01_spi_write_data(0x88);

sunxi_lcd_delay_ms(100);

TL032FWV01_spi_write_cmd(0xE0);
TL032FWV01_spi_write_data(0x80);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x02);

TL032FWV01_spi_write_cmd(0xE1);
TL032FWV01_spi_write_data(0x04);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x05);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x60);
TL032FWV01_spi_write_data(0x60);

TL032FWV01_spi_write_cmd(0xE2);
TL032FWV01_spi_write_data(0x30);
TL032FWV01_spi_write_data(0x30);
TL032FWV01_spi_write_data(0x60);
TL032FWV01_spi_write_data(0x60);
TL032FWV01_spi_write_data(0x3C);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x3D);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);

TL032FWV01_spi_write_cmd(0xE3);
TL032FWV01_spi_write_data(0x00);
```

```
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x33);
TL032FWV01_spi_write_data(0x33);

TL032FWV01_spi_write_cmd(0xE4);
TL032FWV01_spi_write_data(0x44);
TL032FWV01_spi_write_data(0x44);

TL032FWV01_spi_write_cmd(0xE5);
TL032FWV01_spi_write_data(0x06);
TL032FWV01_spi_write_data(0x3E);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x08);
TL032FWV01_spi_write_data(0x40);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x0A);
TL032FWV01_spi_write_data(0x42);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x0C);
TL032FWV01_spi_write_data(0x44);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);

TL032FWV01_spi_write_cmd(0xE6);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x33);
TL032FWV01_spi_write_data(0x33);

TL032FWV01_spi_write_cmd(0xE7);
TL032FWV01_spi_write_data(0x44);
TL032FWV01_spi_write_data(0x44);

TL032FWV01_spi_write_cmd(0xE8);
TL032FWV01_spi_write_data(0x07);
TL032FWV01_spi_write_data(0x3F);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x09);
TL032FWV01_spi_write_data(0x41);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x0B);
TL032FWV01_spi_write_data(0x43);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0x0D);
TL032FWV01_spi_write_data(0x45);
```

```
TL032FWV01_spi_write_data(0xA0);
TL032FWV01_spi_write_data(0xA0);

TL032FWV01_spi_write_cmd(0xEB);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x01);
TL032FWV01_spi_write_data(0x4E);
TL032FWV01_spi_write_data(0x4E);
TL032FWV01_spi_write_data(0xEE);
TL032FWV01_spi_write_data(0x44);
TL032FWV01_spi_write_data(0x00);

TL032FWV01_spi_write_cmd(0xED);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0x04);
TL032FWV01_spi_write_data(0x56);
TL032FWV01_spi_write_data(0x72);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0x27);
TL032FWV01_spi_write_data(0x65);
TL032FWV01_spi_write_data(0x40);
TL032FWV01_spi_write_data(0xFF);
TL032FWV01_spi_write_data(0xFF);

TL032FWV01_spi_write_cmd(0xEF);
TL032FWV01_spi_write_data(0x10);
TL032FWV01_spi_write_data(0x0D);
TL032FWV01_spi_write_data(0x04);
TL032FWV01_spi_write_data(0x08);
TL032FWV01_spi_write_data(0x3F);
TL032FWV01_spi_write_data(0x1F);

TL032FWV01_spi_write_cmd(0xFF);
TL032FWV01_spi_write_data(0x77);
TL032FWV01_spi_write_data(0x01);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x13);

TL032FWV01_spi_write_cmd(0xE8);
TL032FWV01_spi_write_data(0x00);
TL032FWV01_spi_write_data(0x0E);

TL032FWV01_spi_write_cmd(0xFF);
TL032FWV01_spi_write_data(0x77);
```

```
    TL032FWV01_spi_write_data(0x01);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);

    TL032FWV01_spi_write_cmd(0x11);

    sunxi_lcd_delay_ms(120);

    TL032FWV01_spi_write_cmd(0xFF);
    TL032FWV01_spi_write_data(0x77);
    TL032FWV01_spi_write_data(0x01);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x13);

    TL032FWV01_spi_write_cmd(0xE8);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x0C);

    sunxi_lcd_delay_ms(10);

    TL032FWV01_spi_write_cmd(0xE8);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);

    TL032FWV01_spi_write_cmd(0xFF);
    TL032FWV01_spi_write_data(0x77);
    TL032FWV01_spi_write_data(0x01);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);
    TL032FWV01_spi_write_data(0x00);

    TL032FWV01_spi_write_cmd(0x3A);
    TL032FWV01_spi_write_data(0x66);

    TL032FWV01_spi_write_cmd(0x36);
    TL032FWV01_spi_write_data(0x00);

    TL032FWV01_spi_write_cmd(0x35);
    TL032FWV01_spi_write_data(0x00);

    TL032FWV01_spi_write_cmd(0x29);
    return;
}
```

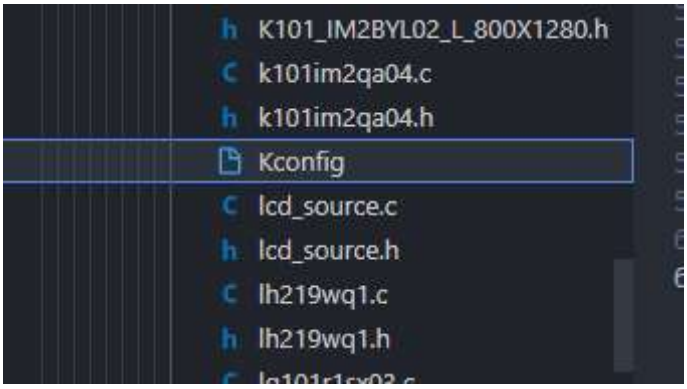
最后一部分，定义了用户自己的操作与屏幕的操作信息，可以在这里增加用户自定义的操作，这里没有操作就留空了，同时修改结构体内为对应的操作即可。

```c
/* sel: 0:lcd0; 1:lcd1 */
static s32 LCD_user_defined_func(u32 sel, u32 para1, u32 para2, u32 para3)
{
    return 0;
}

struct __lcd_panel TL032FWV01_panel = {
    /* panel driver name, must mach the name of lcd_drv_name in sys_config.fex
       */
    .name = "TL032FWV01",
    .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,
        },
};
```

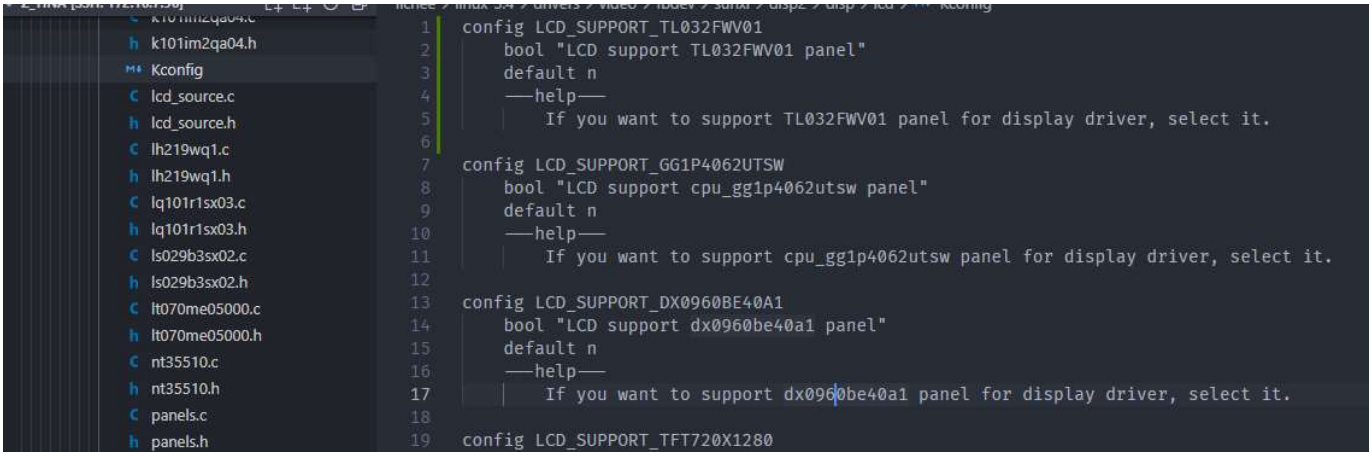
添加屏幕驱动到内核中

屏幕驱动修改完成了，现在需要把屏幕驱动添加到内核中。

首先在屏幕驱动文件夹内找到 `Kconfig` 文件

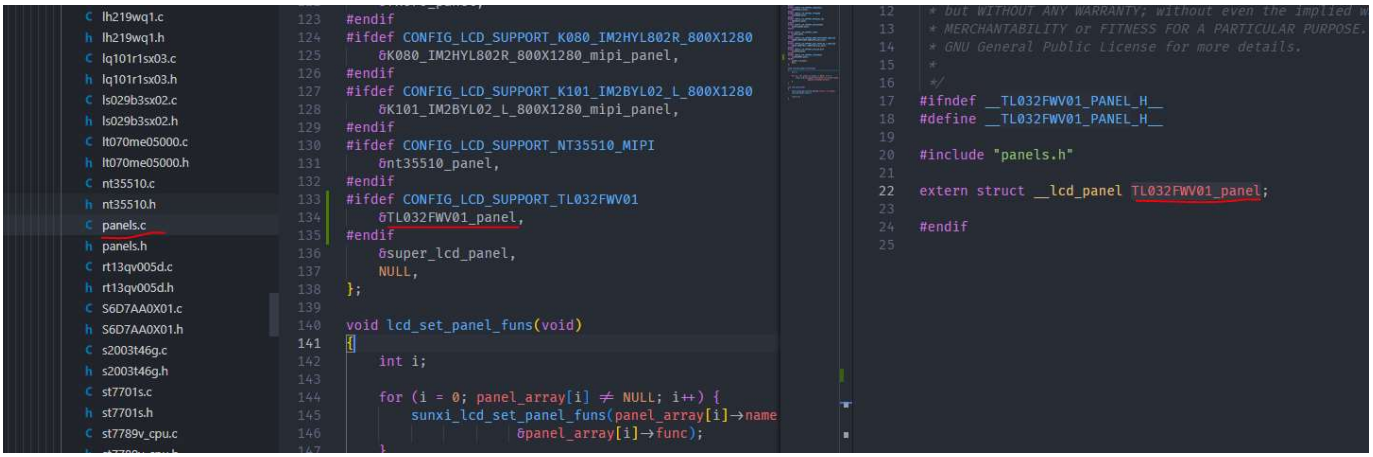


编辑这个文件，新增屏幕的引索。配置名称叫做 `LCD_SUPPORT_TL032FWV01`



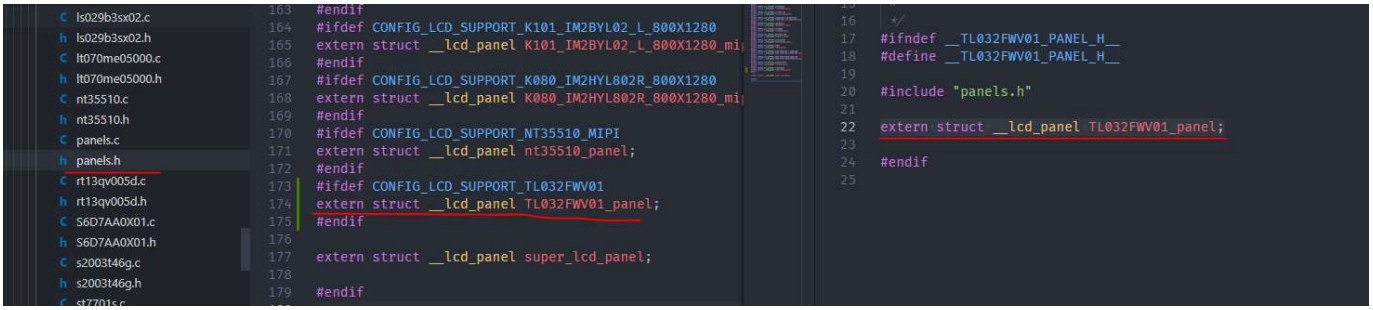
```
config LCD_SUPPORT_TL032FWV01
    bool "LCD support TL032FWV01 panel"
    default n
    ---help---
        If you want to support TL032FWV01 panel for display driver, select it.
```

同时在相同文件夹内找到 panel.c 和 panel.h 两个文件，修改 panel.c 增加屏幕指针。并使用 ifdef 宏来确定只有启用这款屏幕的时候才会编译这个屏幕的驱动。指针的声明位于刚才修改的头文件中，需要确认声明与 panel.c 中的名称是否一致。




```
#ifdef CONFIG_LCD_SUPPORT_TL032FWV01
    &TL032FWV01_panel,
#endif
```

头文件也一致，引用结构体




```
#ifdef CONFIG_LCD_SUPPORT_TL032FWV01
extern struct __lcd_panel TL032FWV01_panel;
#endif
```

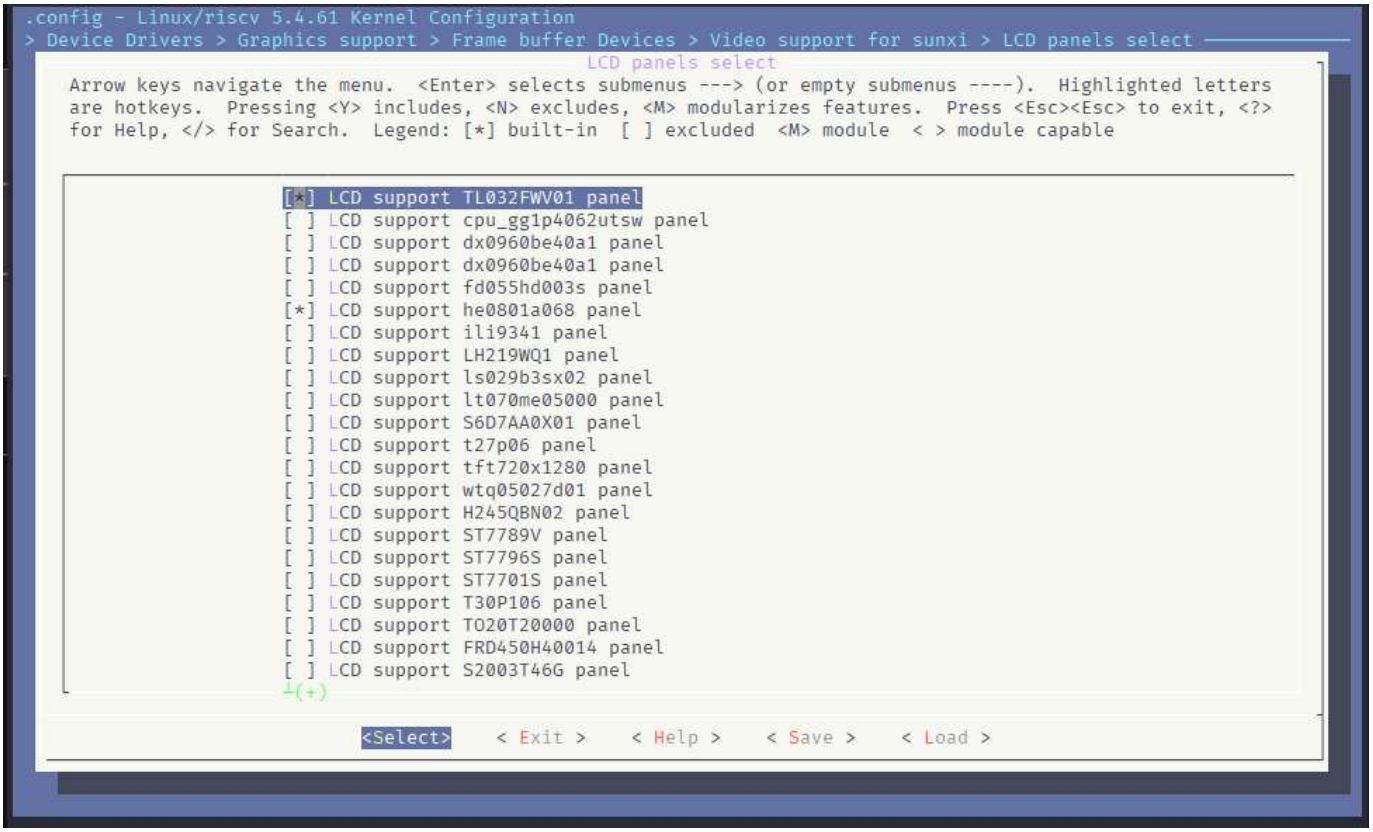
最后，到屏幕面板驱动的上一层文件夹，找到 Makefile 添加驱动编译选项。


```
disp-$(CONFIG_LCD_SUPPORT_TL032FWV01) += lcd/TL032FWV01.o
```

至此，屏幕驱动就添加到内核中了。

内核配置

make kernel_menuconfig 后，到 Device Drivers > Graphics support > Frame buffer Devices > Video support for sunxi > LCD panels select 找到屏幕，勾选即可。



可以在编译的时候查看驱动是否被编译上了

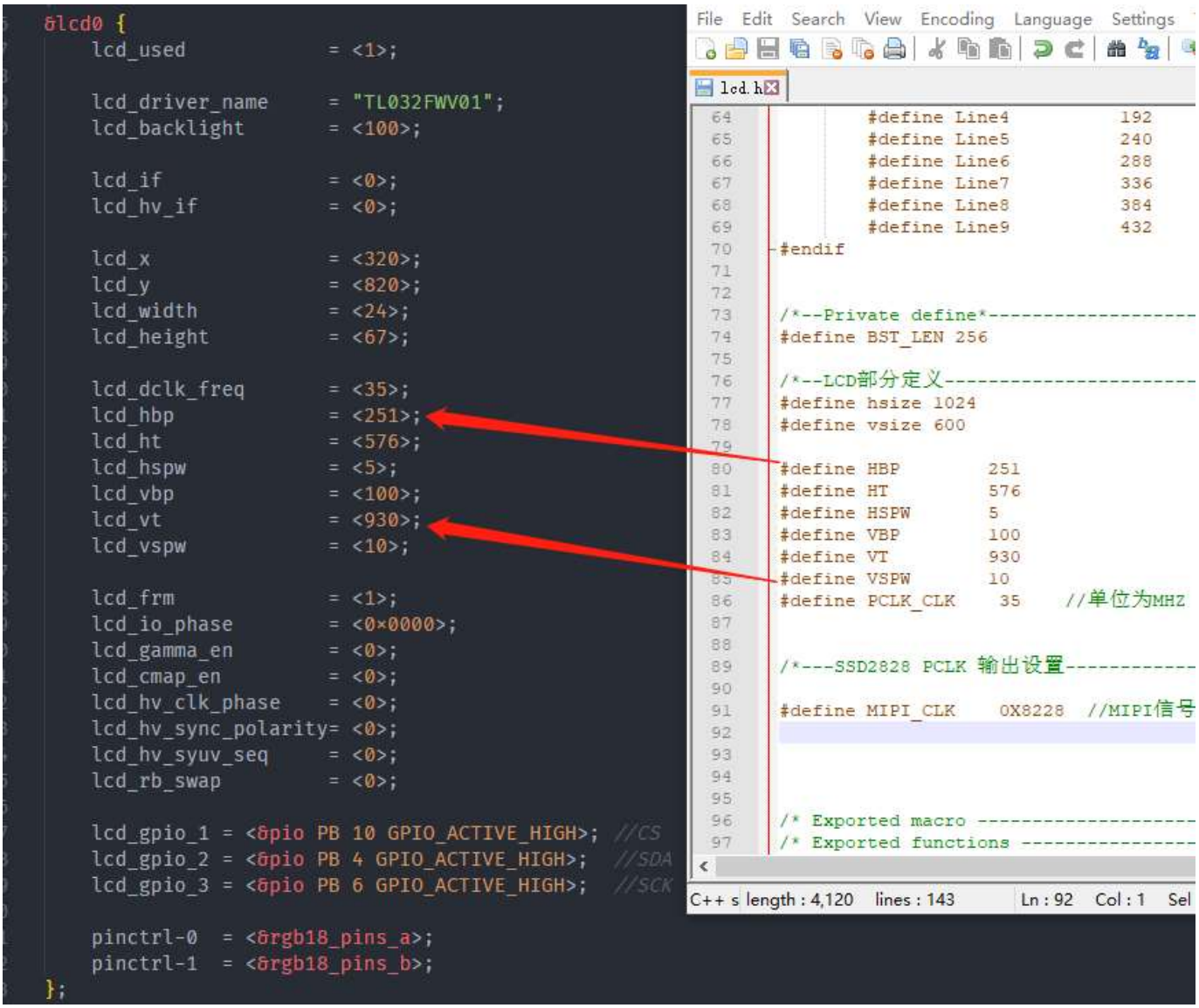
```
CC      drivers/video/fbdev/sunxi/disp2/disp/de/lowlevel_v2x/de_clock.o
CC      drivers/video/fbdev/sunxi/disp2/disp/de/lowlevel_v2x/de_dsi.o
CC      drivers/video/fbdev/sunxi/disp2/disp/de/lowlevel_v2x/de_lcd_sun50iw10.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/he0801a068.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/WilliamLcd.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/lq101r1sx03.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/inet_dsi_panel.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/bp101wx1-206.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/fx070.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/k101im2qa04.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/CC08021801_310_800X1280.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/K080_IM2HYL802R_800X1280.o
AR      drivers/usb/storage/built-in.a
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/K101_IM2BYL02_L_800X1280.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/TL032FWV01.o
CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/super_lcd_driver.o
AR      drivers/usb/sunxi_usb/built-in.a
AR      drivers/usb/built-in.a
AR      drivers/video/fbdev/sunxi/disp2/disp/built-in.a
AR      drivers/video/fbdev/sunxi/built-in.a
AR      drivers/video/fbdev/built-in.a
AR      drivers/video/built-in.a
AR      drivers/built-in.a
GEN     .version
CHK     include/generated/compile.h
LD      vmlinux.o
MODPOST vmlinux.o
MODINFO modules.builtin.modinfo
```

配置设备树

首先，我们在 `pio` 节点内增加 `rgb18_pins` 作为 LCD 屏幕的 Pin 绑定。

```
rgb18_pins_a: rgb18@0 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", \
        "PD6", "PD7", "PD8", "PD9", "PD10", "PD11", \
        "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", \
        "PD18", "PD19", "PD20", "PD21";
    function = "lcd0";
    drive-strength = <30>;
    bias-disable;
};

rgb18_pins_b: rgb18@1 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", \
        "PD6", "PD7", "PD8", "PD9", "PD10", "PD11", \
        "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", \
        "PD18", "PD19", "PD20", "PD21";
    function = "io_disabled";
    bias-disable;
};
```

设备树类似的，按照屏厂提供的参数填写即可。 `lcd_driver_name` 记得与之前驱动中的 `.name="TL032FWV01"` 相同。



```
&lcd0 {
    lcd_used            = <1>;

    lcd_driver_name     = "TL032FWV01";
    lcd_backlight       = <100>;

    lcd_if              = <0>;
    lcd_hv_if           = <0>;

    lcd_x               = <320>;
    lcd_y               = <820>;
    lcd_width           = <24>;
    lcd_height          = <67>;

    lcd_dclk_freq       = <35>;
    lcd_hbp             = <251>;
    lcd_ht              = <576>;
    lcd_hspw            = <5>;
    lcd_vbp             = <100>;
    lcd_vt              = <930>;
    lcd_vspw            = <10>;

    lcd_frm             = <1>;
    lcd_io_phase        = <0x0000>;
    lcd_gamma_en        = <0>;
    lcd_cmap_en         = <0>;
    lcd_hv_clk_phase    = <0>;
    lcd_hv_sync_polarity= <0>;
    lcd_hv_syuv_seq     = <0>;
    lcd_rb_swap         = <0>;

    lcd_gpio_1 = <&pio PB 10 GPIO_ACTIVE_HIGH>; //CS
    lcd_gpio_2 = <&pio PB 4 GPIO_ACTIVE_HIGH>;  //SDA
    lcd_gpio_3 = <&pio PB 6 GPIO_ACTIVE_HIGH>;  //SCK

    pinctrl-0   = <&rgb18_pins_a>;
    pinctrl-1   = <&rgb18_pins_b>;
};
```

File Edit Search View Encoding Language Settings

lcd.h

```
64          #define Line4           192
65          #define Line5           240
66          #define Line6           288
67          #define Line7           336
68          #define Line8           384
69          #define Line9           432
70  #endif
71
72
73  /*--Private define*-------------------
74  #define BST_LEN 256
75
76  /*--LCD部分定义-----------------------
77  #define hsize 1024
78  #define vsize 600
79
80  #define HBP         251
81  #define HT          576
82  #define HSPW        5
83  #define VBP         100
84  #define VT          930
85  #define VSPW        10
86  #define PCLK_CLK    35    //单位为MHZ
87
88
89  /*---SSD2828 PCLK 输出设置------------
90
91  #define MIPI_CLK    0X8228  //MIPI信号
92
93
94
95
96  /* Exported macro --------------------
97  /* Exported functions ----------------
```

C++ s length : 4,120 lines : 143 Ln : 92 Col : 1 Sel

测试屏幕

没啥好说的，自己就亮了

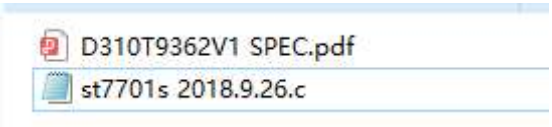
适配 MIPI DSI 屏幕

MIPI DSI 屏幕的适配较为简单，一样的只需要给定时序参数与初始化即可。不过部分 MIPI 屏幕包括其他功能，例如 OLED 屏幕的背光调节等，需要在驱动里实现相关的功能。这里不对这些操作做过多叙述，仅仅点亮屏幕即可。

开发板使用的是【基于 D1s 的带屏旁路由 Yuzuki RV Router】，屏幕使用的是【D310T9362V1】，2 lane MIPI DSI，硬件连接如下：

适配前的准备

同样的，向屏厂索要屏幕资料。这次提供的资料如下：



一份数据手册与一份屏幕初始化代码。

在数据手册中，我们可以知道这个屏幕使用的是 ST7701s 驱动芯片，分辨率 800x320.

2.Features

LCD type	IPS TFT
Display Format	Graphic 480*RGB*800 Dot-matrix
Interface Type	MIPI Video Mode
Viewing Direction	ALL
Driver	ST7701S
Backlight	White
Display color	16.7M

3.Mechanical Specification

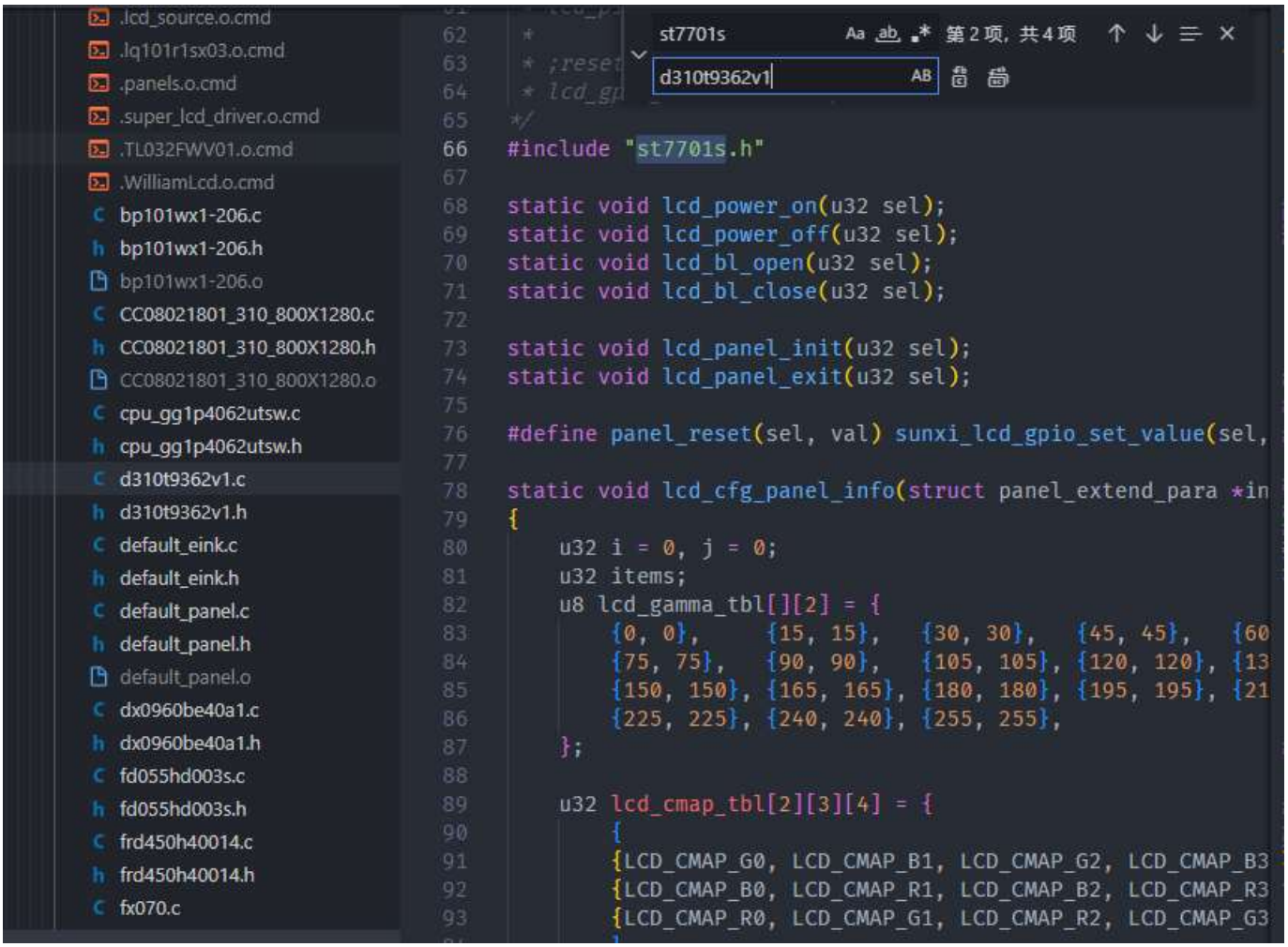
Item	Specifications	Unit
Dimensional outline	43.44(W)*75.3(H)*2.8 (T) (exclude FPC)	mm
Resolution	480*RGB*800	Dots
Active area	40.32(W)*67.2(H)	mm
Pixel Pitch	0.084 (W)*0.084(V)	mm
ASSY.TYPE	COG+FPC	--
WEIGHT	TBD	g

屏幕初始化代码中，配置了 MIPI 的时序，屏幕的亮度，GAMMA等等信息。

```
//////////////////init code //////////////////////////

{DSICMD_CMD,0x01},
{CMDDELAY_MS,120},

{DSICMD_CMD,0x11},
{CMDDELAY_MS,120},

{DSICMD_CMD,0xFF},
{DSICMD_DATA,0x77},
{DSICMD_DATA,0x01},
{DSICMD_DATA,0x00},
{DSICMD_DATA,0x00},
{DSICMD_DATA,0x11},

{DSICMD_CMD,0xD1},
{DSICMD_DATA,0x11},

{DSICMD_CMD,0x55},
{DSICMD_DATA,0xb0},  // 80 90 b0


{DSICMD_CMD,0xFF},
{DSICMD_DATA,0x77},
{DSICMD_DATA,0x01},
{DSICMD_DATA,0x00},
{DSICMD_DATA,0x00},
{DSICMD_DATA,0x10},


{DSICMD_CMD,0xC0},
{DSICMD_DATA,0x63},
{DSICMD_DATA,0x00},

{DSICMD_CMD,0xC1},
{DSICMD_DATA,0x09},
{DSICMD_DATA,0x02},
```

编写驱动程序

同样的，这次是 MIPI DSI 屏幕，就使用 `ST7701s.c` 这个驱动修改即可。复制一份驱动文件夹，改名 `d310t9362v1` ，并全局替换 `st7701s` 到 `d310t9362v1` 。



接下来修改源码，编辑 `d310t9362v1.c` 文件，修改源码，这里只讲述需要修改的部分，其他部分不做修改即可。

（1）开关屏幕部分

这里按照数据手册里的开关屏幕时序设置即可，ST7701s 的驱动开屏较慢，需要适当增加延时。

```c
static s32 lcd_open_flow(u32 sel)
{
    LCD_OPEN_FUNC(sel, lcd_power_on, 10);
    LCD_OPEN_FUNC(sel, lcd_panel_init, 120);
    LCD_OPEN_FUNC(sel, sunxi_lcd_tcon_enable, 120);
    LCD_OPEN_FUNC(sel, lcd_bl_open, 0);
    return 0;
}

static s32 lcd_close_flow(u32 sel)
{
    LCD_CLOSE_FUNC(sel, lcd_bl_close, 0);
    LCD_CLOSE_FUNC(sel, sunxi_lcd_tcon_disable, 0);
    LCD_CLOSE_FUNC(sel, lcd_panel_exit, 200);
    LCD_CLOSE_FUNC(sel, lcd_power_off, 500);
    return 0;
}

static void lcd_power_on(u32 sel)
{
    sunxi_lcd_pin_cfg(sel, 1);
    sunxi_lcd_delay_ms(50);
    panel_reset(sel, 1);
    sunxi_lcd_delay_ms(5);
    panel_reset(sel, 0);
    sunxi_lcd_delay_ms(10);
    panel_reset(sel, 1);
    sunxi_lcd_delay_ms(120);
}

static void lcd_power_off(u32 sel)
{
    sunxi_lcd_pin_cfg(sel, 0);
    sunxi_lcd_delay_ms(20);
    panel_reset(sel, 0);
    sunxi_lcd_delay_ms(5);
}
```

```
static s32 lcd_open_flow(u32 sel)
{
    LCD_OPEN_FUNC(sel, lcd_power_on, 10);
    LCD_OPEN_FUNC(sel, lcd_panel_init, 120);
    LCD_OPEN_FUNC(sel, sunxi_lcd_tcon_enable, 120);
    LCD_OPEN_FUNC(sel, lcd_bl_open, 0);
    return 0;
}

static s32 lcd_close_flow(u32 sel)
{
    LCD_CLOSE_FUNC(sel, lcd_bl_close, 0);
    LCD_CLOSE_FUNC(sel, sunxi_lcd_tcon_disable, 0);
    LCD_CLOSE_FUNC(sel, lcd_panel_exit, 200);
    LCD_CLOSE_FUNC(sel, lcd_power_off, 500);
    return 0;
}

static void lcd_power_on(u32 sel)
{
    sunxi_lcd_pin_cfg(sel, 1);
    sunxi_lcd_delay_ms(50);
    panel_reset(sel, 1);
    sunxi_lcd_delay_ms(5);
    panel_reset(sel, 0);
    sunxi_lcd_delay_ms(10);
    panel_reset(sel, 1);
    sunxi_lcd_delay_ms(120);
}

static void lcd_power_off(u32 sel)
{
    sunxi_lcd_pin_cfg(sel, 0);
    sunxi_lcd_delay_ms(20);
    panel_reset(sel, 0);
    sunxi_lcd_delay_ms(5);
}
```

（2）屏幕初始化部分

这也是最重要的部分，首先定义两个**屏幕初始化中没有使用的值**作为标志位使用，`REGFLAG_DELAY` 与 `REGFLAG_END_OF_TABLE` 。

然后依照屏厂提供的初始化参数编写初始化表

```
#define REGFLAG_DELAY 0XFC
#define REGFLAG_END_OF_TABLE 0x1FD /* END OF REGISTERS MARKER */

struct LCM_setting_table {
    u8 cmd;
    u32 count;
    u8 para_list[18];
};

static struct LCM_setting_table lcm_initialization_setting[] = {
    {0x01, 1, {0x00} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {120} },

    {0x11, 1, {0x00} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {120} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x11} },
    {0xd1, 1, {0x11} },
    {0x55, 1, {0xb0} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x10} },
    {0xc0, 2, {0x63, 0x00} }, // SCNL = (0x63 + 1) * 8 = 800
    {0xc1, 2, {0x09, 0x02} }, // VFB=0x09  VBF=0x02
    {0xc2, 2, {0x37, 0x08} }, // PCLK= 512 + (0x08 * 16) = 640

    {0xc7, 1, {0x00} }, // x-dir  rotate 0 : 0x00     rotate 180 :0x04

    {0xcc, 1, {0x38} },

    {0xb0, 16, {0x00, 0x11, 0x19, 0x0c, 0x10, 0x06, 0x07, 0x0a, 0x09, 0x22,
           0x04, 0x10, 0x0e, 0x28, 0x30, 0x1c} },

    {0xb1, 16, {0x00, 0x12, 0x19, 0x0d, 0x10, 0x04, 0x06, 0x07, 0x08, 0x23,
            0x04, 0x12, 0x11, 0x28, 0x30, 0x1c} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x11} }, //  enable  bk fun of  command 2  BK1
    {0xb0, 1, {0x4d} },
    {0xb1, 1, {0x5b} }, // 0x56  0x4a  0x5b
    {0xb2, 1, {0x07} },
    {0xb3, 1, {0x80} },
    {0xb5, 1, {0x47} },
    {0xb7, 1, {0x8a} },
    {0xb8, 1, {0x21} },
    {0xc1, 1, {0x78} },
    {0xc2, 1, {0x78} },
    {0xd0, 1, {0x88} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {100} },
```

这里需要一些转换，不能像之前的复制粘贴了，需要以每一个写命令（或延时）作为分隔，并与屏厂提供的命令一一对应即可。编写方法具体如下：

（1）写命令，没有数据

```
{DSICMDCMD,0x01},  -----> {0x01, 1, {0x00}},
```

（2）写命令，带有数据

```
{DSICMD_CMD,0xFF},
{DSICMD_DATA,0x77},
{DSICMD_DATA,0x01},
{DSICMD_DATA,0x00}, -----> {0xFF, 5, {0x77, 0x01, 0x00, 0x00, 0x11}},
{DSICMD_DATA,0x00},
{DSICMD_DATA,0x11},
```

其中的含义为：

```
{0xFF,    5,        {0x77,       0x01, 0x00, 0x00, 0x11}},
^ 命令地址 ^ 数据个数   ^ 第一个数据   ^ 第二个数据 ....
```

(3) 延时

```
{CMDDELAY_MS,120}, -----> {REGFLAG_DELAY, REGFLAG_DELAY, {120}},
```

转换完成的初始化表如下

```
static struct LCM_setting_table lcm_initialization_setting[] = {
    {0x01, 1, {0x00} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {120} },

    {0x11, 1, {0x00} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {120} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x11} },
    {0xd1, 1, {0x11} },
    {0x55, 1, {0xb0} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x10} },
    {0xc0, 2, {0x63, 0x00} }, // SCNL = (0x63 + 1) * 8 = 800
    {0xc1, 2, {0x09, 0x02} }, // VFB=0x09  VBF=0x02
    {0xc2, 2, {0x37, 0x08} }, // PCLK= 512 + (0x08 * 16) = 640

    {0xc7, 1, {0x00} }, // x-dir  rotate 0 : 0x00     rotate 180 :0x04

    {0xcc, 1, {0x38} },

    {0xb0, 16, {0x00, 0x11, 0x19, 0x0c, 0x10, 0x06, 0x07, 0x0a, 0x09, 0x22,
           0x04, 0x10, 0x0e, 0x28, 0x30, 0x1c} },

    {0xb1, 16, {0x00, 0x12, 0x19, 0x0d, 0x10, 0x04, 0x06, 0x07, 0x08, 0x23,
            0x04, 0x12, 0x11, 0x28, 0x30, 0x1c} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x11} }, //  enable  bk fun of  command 2  BK1
    {0xb0, 1, {0x4d} },
    {0xb1, 1, {0x5b} }, // 0x56  0x4a  0x5b
    {0xb2, 1, {0x07} },
    {0xb3, 1, {0x80} },
    {0xb5, 1, {0x47} },
    {0xb7, 1, {0x8a} },
    {0xb8, 1, {0x21} },
    {0xc1, 1, {0x78} },
    {0xc2, 1, {0x78} },
    {0xd0, 1, {0x88} },
    {REGFLAG_DELAY, REGFLAG_DELAY, {100} },

    {0xe0, 3, {0x00, 0x00, 0x02} },
    {0xe1, 11, {0x01, 0xa0, 0x03, 0xa0, 0x02, 0xa0, 0x04, 0xa0, 0x00, 0x44,
            0x44} },
    {0xe2, 12, {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x00} },
    {0xe3, 4, {0x00, 0x00, 0x33, 0x33} },
    {0xe4, 2, {0x44, 0x44} },
    {0xe5, 16, {0x01, 0x26, 0xa0, 0xa0, 0x03, 0x28, 0xa0, 0xa0, 0x05, 0x2a,
            0xa0, 0xa0, 0x07, 0x2c, 0xa0, 0xa0} },
    {0xe6, 4, {0x00, 0x00, 0x33, 0x33} },
    {0xe7, 2, {0x44, 0x44} },
```

```
    {0xe8, 16, {0x02, 0x26, 0xa0, 0xa0, 0x04, 0x28, 0xa0, 0xa0, 0x06, 0x2a,
            0xa0, 0xa0, 0x08, 0x2c, 0xa0, 0xa0} },
    {0xeb, 7, {0x00, 0x01, 0xe4, 0xe4, 0x44, 0x00, 0x40} },
    {0xed, 16, {0xff, 0xf7, 0x65, 0x4f, 0x0b, 0xa1, 0xcf, 0xff, 0xff, 0xfc,
            0x1a, 0xb0, 0xf4, 0x56, 0x7f, 0xff} },

    {0xff, 5, {0x77, 0x01, 0x00, 0x00, 0x00} },
    {0x36, 1, {0x00} }, // U&D  Y-DIR  rotate 0: 0x00 : rotate 180 :0x10
    {0x3a, 1, {0x55} },
    {0x29, 1, {0x00} },
    {REGFLAG_END_OF_TABLE, REGFLAG_END_OF_TABLE, {} }
};
```

（3）屏幕初始化、退出部分

这一部分也不需要修改，默认即可。

```
static void lcd_panel_init(u32 sel)
{
    u32 i = 0;

    sunxi_lcd_dsi_clk_enable(sel);
    sunxi_lcd_delay_ms(10);

    for (i = 0;; i++) {
        if (lcm_initialization_setting[i].cmd == REGFLAG_END_OF_TABLE)
            break;
        else if (lcm_initialization_setting[i].cmd == REGFLAG_DELAY)
            sunxi_lcd_delay_ms(lcm_initialization_setting[i].count);
        else {
            dsi_dcs_wr(0, lcm_initialization_setting[i].cmd,
                    lcm_initialization_setting[i].para_list,
                    lcm_initialization_setting[i].count);
        }
    }
}

static void lcd_panel_exit(u32 sel)
{
    sunxi_lcd_dsi_dcs_write_0para(sel, 0x28);
    sunxi_lcd_delay_ms(10);
    sunxi_lcd_dsi_dcs_write_0para(sel, 0x10);
    sunxi_lcd_delay_ms(10);
}
```

（4）屏幕定义部分

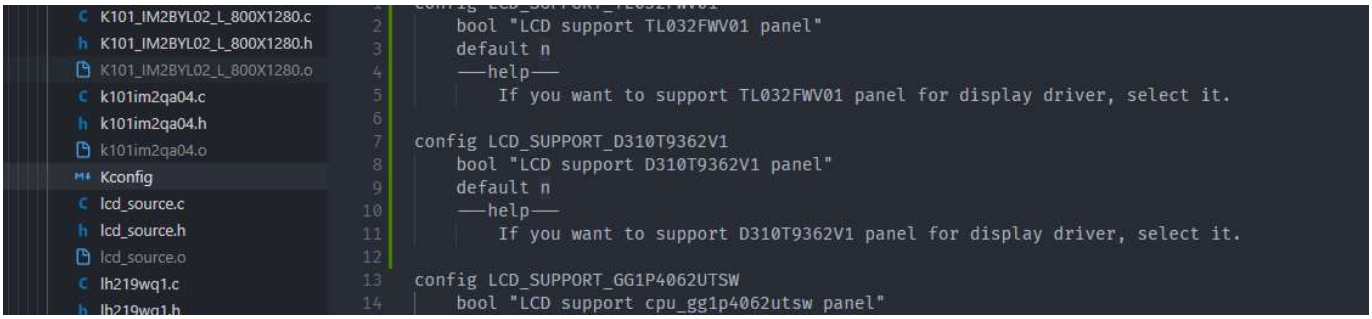
这一部分定义了屏幕的用户操作，屏幕的名称等等。

```c
/*sel: 0:lcd0; 1:lcd1*/
static s32 lcd_user_defined_func(u32 sel, u32 para1, u32 para2, u32 para3)
{
    return 0;
}

struct __lcd_panel d310t9362v1_panel = {
    /* panel driver name, must mach the name of
     * lcd_drv_name in sys_config.fex
     */
    .name = "d310t9362v1",
    .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,
    },
};
```

将屏幕驱动添加到内核中

与之前的一样，编辑屏幕驱动文件夹内的 `Kconfig` 文件，添加引索

```
config LCD_SUPPORT_D310T9362V1
    bool "LCD support D310T9362V1 panel"
    default n
    ---help---
        If you want to support D310T9362V1 panel for display driver, select it.
```

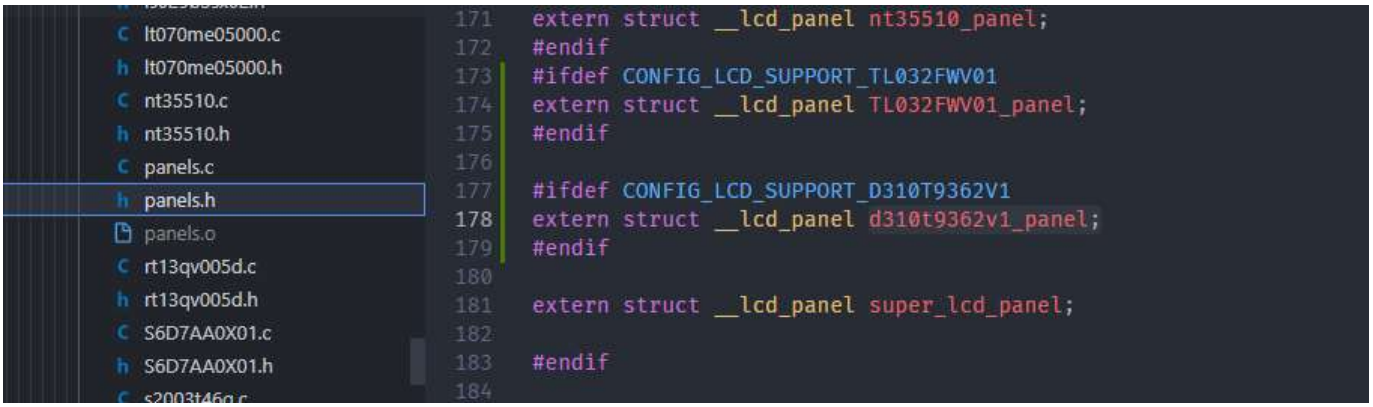


然后编辑同目录中的 `panel.c` ，添加结构体

```
#ifdef CONFIG_LCD_SUPPORT_D310T9362V1
    &d310t9362v1_panel,
#endif
```

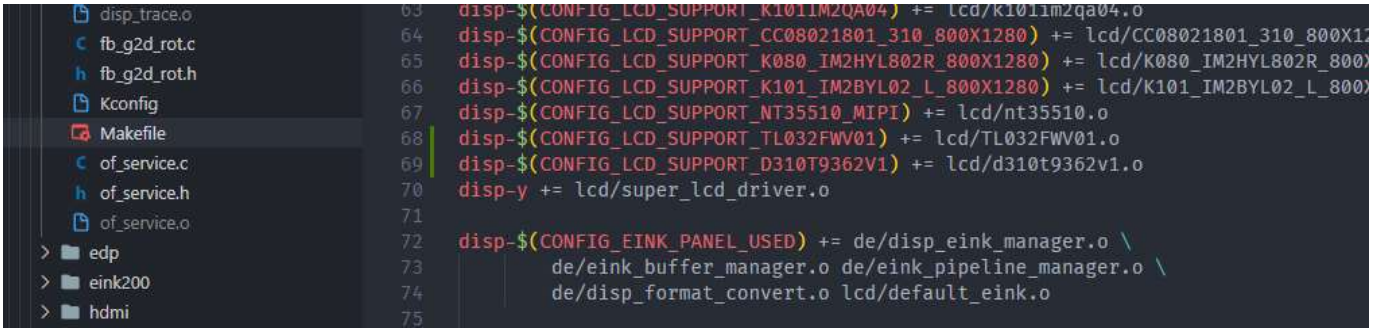
编辑同目录下的 `panel.h` 添加结构体

```
#ifdef CONFIG_LCD_SUPPORT_D310T9362V1
extern struct __lcd_panel d310t9362v1_panel;
#endif
```



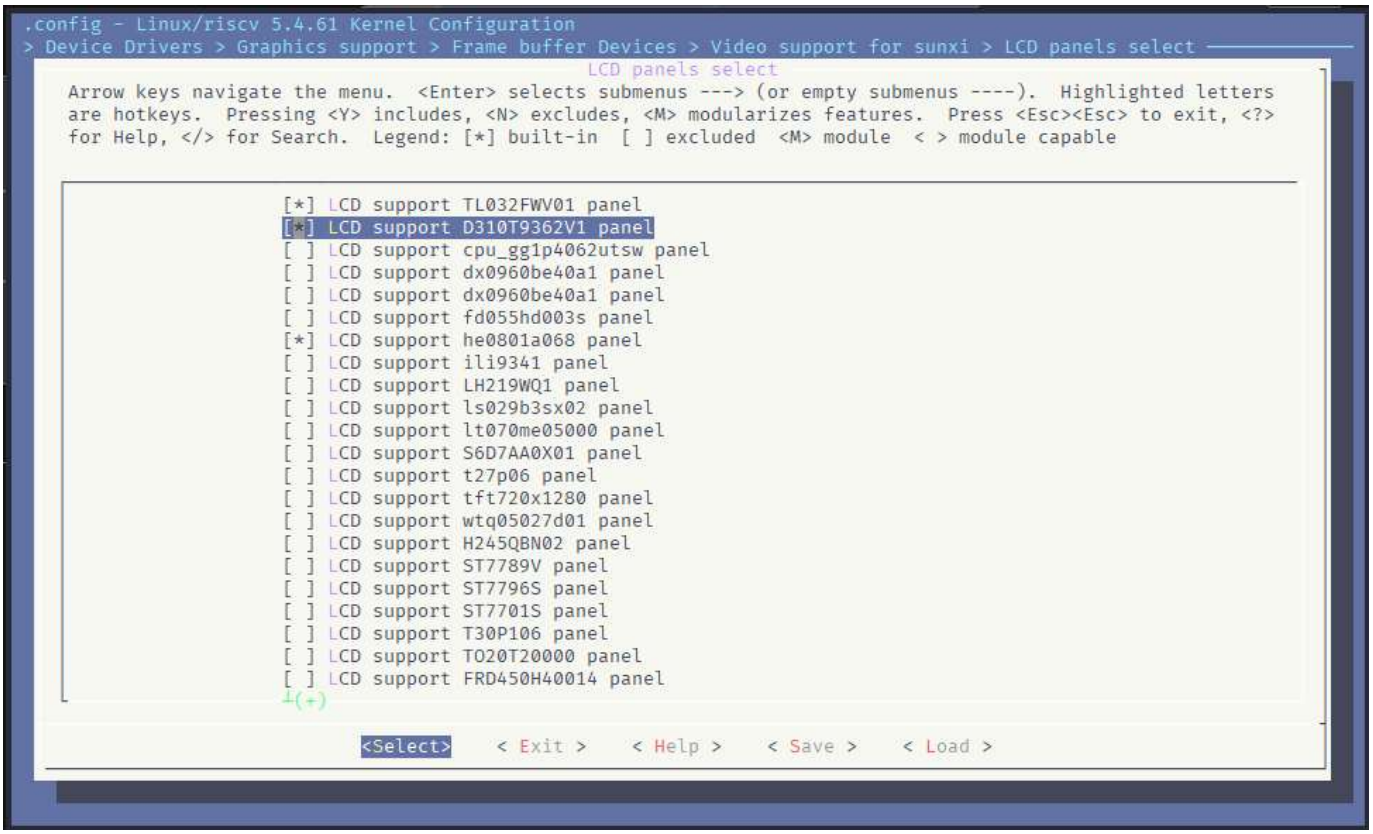
到上一级目录中，找到 `Makefile` 文件，增加编译项

```
disp-$(CONFIG_LCD_SUPPORT_D310T9362V1) += lcd/d310t9362v1.o
```



内核配置

`make kernel_menuconfig` 后，到 `Device Drivers > Graphics support > Frame buffer Devices > Video support for sunxi > LCD panels select` 找到屏幕，勾选即可。



勾选后编译可以看到驱动被编译了

```
./scripts/Makefile.asm-generic:25: redundant generic-y found in arch/riscv/include/asm/Kbuild: device.h
  CALL    scripts/checksyscalls.sh
  CALL    scripts/atomic/check-atomics.sh
  CHK     include/generated/compile.h
  CC      drivers/video/fbdev/sunxi/disp2/disp/lcd/d310t9362v1.o
  AR      drivers/video/fbdev/sunxi/disp2/disp/built-in.a
  AR      drivers/video/fbdev/sunxi/built-in.a
  AR      drivers/video/fbdev/built-in.a
  AR      drivers/video/built-in.a
  AR      drivers/built-in.a
  GEN     .version
```

配置设备树

首先，我们在 `pio` 节点内增加 `dsi2lane_pins` 作为 LCD 屏幕的 Pin 绑定。

```
dsi2lane_pins_a: dsi2lane@0 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5";
    function = "dsi";
    drive-strength = <30>;
    bias-disable;
};

dsi2lane_pins_b: dsi2lane@1 {
    pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5";
    function = "io_disabled";
    bias-disable;
};
```

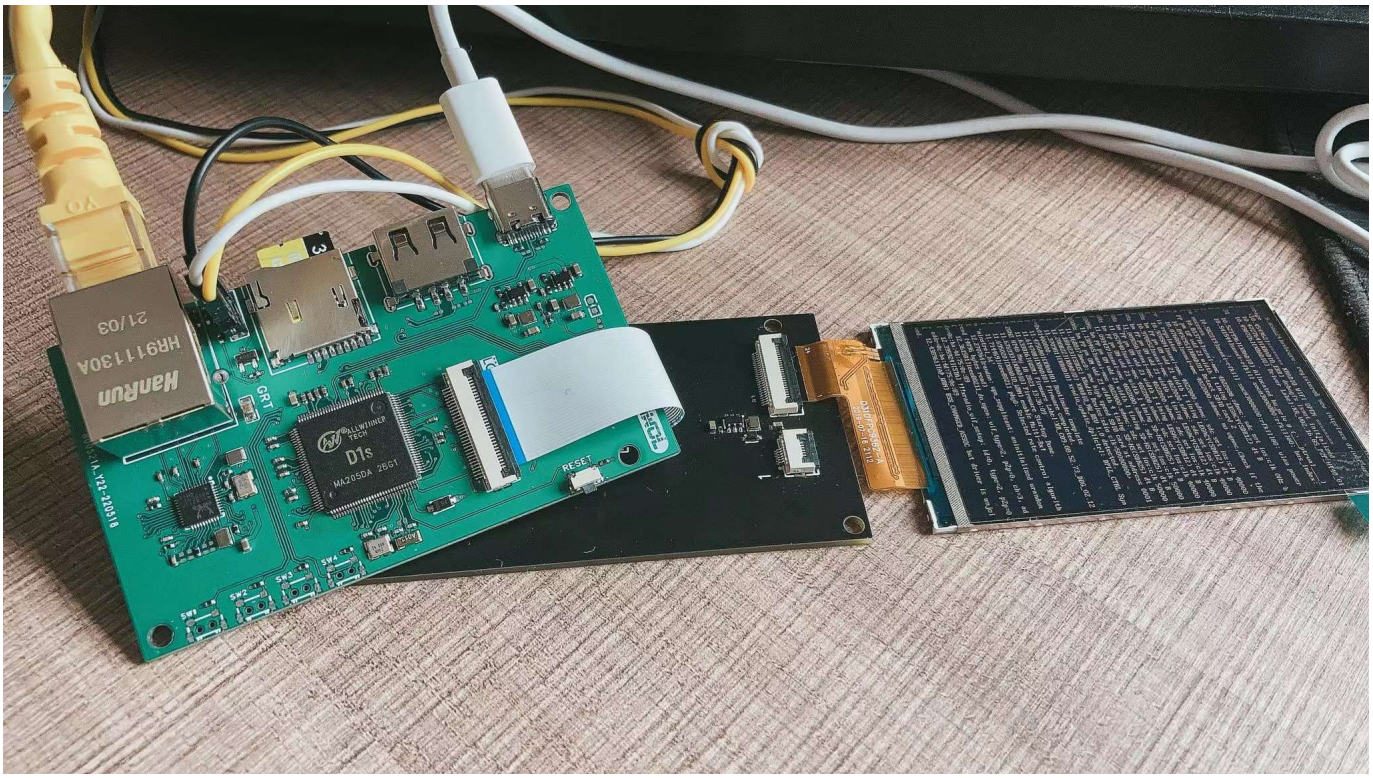
设备树类似的，按照屏厂提供的参数填写即可。 `lcd_driver_name` 记得与之前驱动中的 `.name="d310t9362v1"` 相同。

```
&lcd0 {
    lcd_used            = <1>;

    lcd_driver_name     = "d310t9362v1";
    lcd_backlight       = <50>;
    lcd_if              = <4>;

    lcd_x               = <480>;
    lcd_y               = <800>;
    lcd_width           = <40>;
    lcd_height          = <67>;

    lcd_pwm_used        = <1>;
    lcd_pwm_ch          = <2>;
    lcd_pwm_freq        = <1000>;
    lcd_pwm_pol         = <0>;
    lcd_pwm_max_limit   = <255>;

    lcd_dclk_freq       = <34>;
    lcd_hbp             = <120>;
    lcd_ht              = <624>;
    lcd_hspw            = <48>;
    lcd_vbp             = <28>;
    lcd_vt              = <908>;
    lcd_vspw            = <12>;

    lcd_dsi_if          = <0>;
    lcd_dsi_lane        = <2>;

    deu_mode            = <0>;
    lcdgamma4iep        = <22>;
    smart_color         = <90>;

    lcd_gpio_0 =  <&pio PD 9 GPIO_ACTIVE_HIGH>;
    pinctrl-0 = <&dsi2lane_pins_a>;
    pinctrl-1 = <&dsi2lane_pins_b>;
};
```

```
&lcd0 {
    lcd_used            = <1>;

    lcd_driver_name     = "d310t9362v1";
    lcd_backlight       = <50>;
    lcd_if              = <4>;

    lcd_x               = <480>;
    lcd_y               = <800>;
    lcd_width           = <40>;
    lcd_height          = <67>;

    lcd_pwm_used        = <1>;
    lcd_pwm_ch          = <2>;
    lcd_pwm_freq        = <1000>;
    lcd_pwm_pol         = <0>;
    lcd_pwm_max_limit   = <255>;

    lcd_dclk_freq       = <34>;
    lcd_hbp             = <120>;
    lcd_ht              = <624>;
    lcd_hspw            = <48>;
    lcd_vbp             = <28>;
    lcd_vt              = <908>;
    lcd_vspw            = <12>;

    lcd_dsi_if          = <0>;
    lcd_dsi_lane        = <2>;

    deu_mode            = <0>;
    lcdgamma4iep        = <22>;
    smart_color         = <90>;

    lcd_gpio_0 =  <&pio PD 9 GPIO_ACTIVE_HIGH>;
    pinctrl-0 = <&dsi2lane_pins_a>;
    pinctrl-1 = <&dsi2lane_pins_b>;
};
```

测试屏幕

开机就点亮了，下文介绍如何显示命令行到屏幕上。

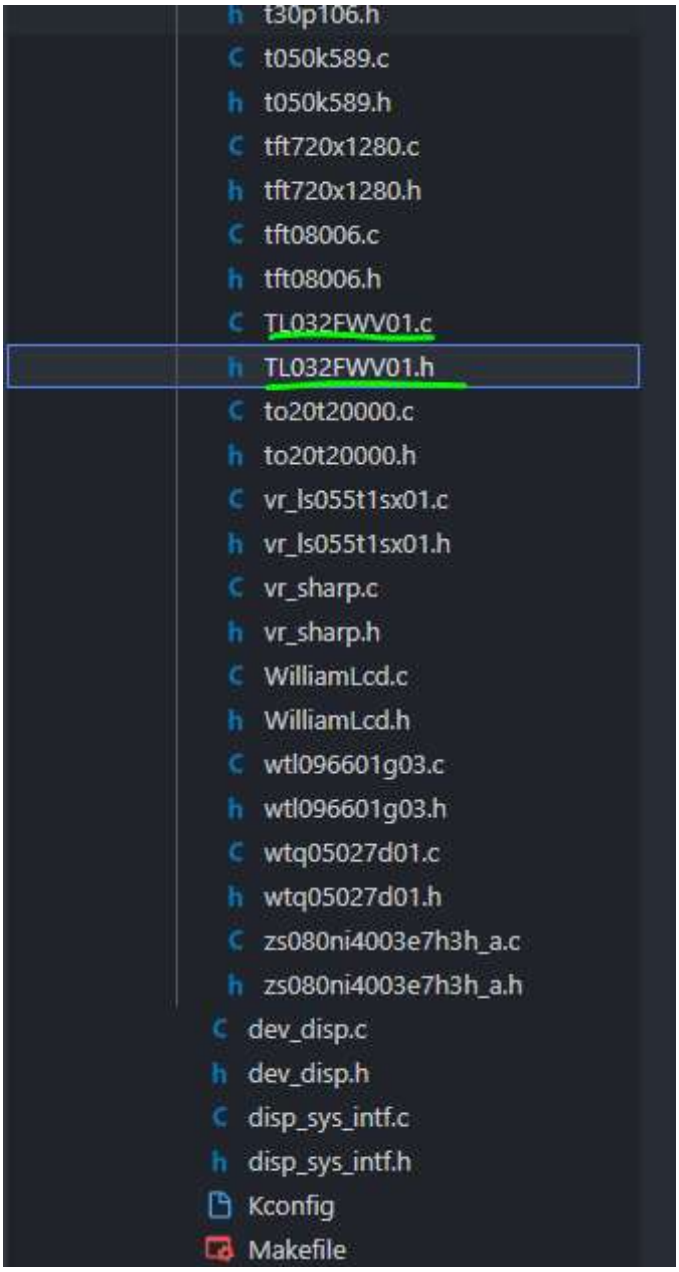


屏幕驱动移植 U-Boot

对于 U-Boot，需要对屏幕驱动进行修改。这里我们以上面移植过的【TL032FWV01】屏幕驱动作为示例，演示如何移植 Linux 的驱动到 U-Boot 内。首先找到屏幕驱动的路径：

```
brandy-2.0/u-boot-2018/drivers/video/sunxi/disp2/disp/lcd
```

我们先把在 Linux 内测试 OK 的驱动复制到 U-Boot 屏幕驱动目录中。



驱动源码的修改

找到屏幕驱动头文件 `TL032FWV01.h` 把

```
extern struct __lcd_panel_t TL032FWV01_panel;
```

改为

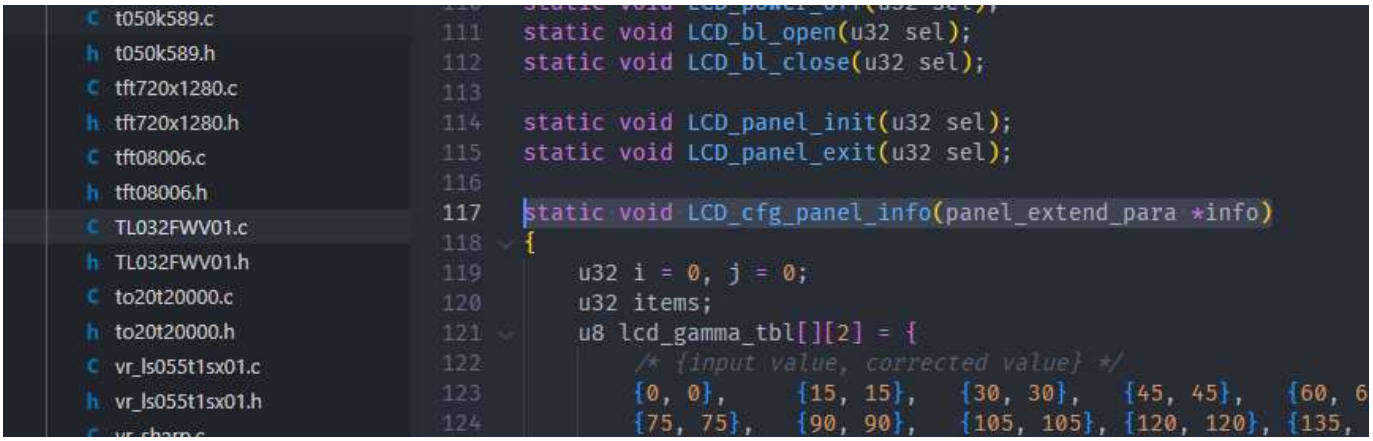
```
extern __lcd_panel_t TL032FWV01_panel;
```

找到源文件 `TL032FWV01.c` 把

```
static void LCD_cfg_panel_info(struct panel_extend_para *info)
```

改为

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



找到源文件末尾，把

```
struct __lcd_panel TL032FWV01_panel = {
```

改为

```
__lcd_panel_t TL032FWV01_panel = {
```

增加 U-Boot 引索

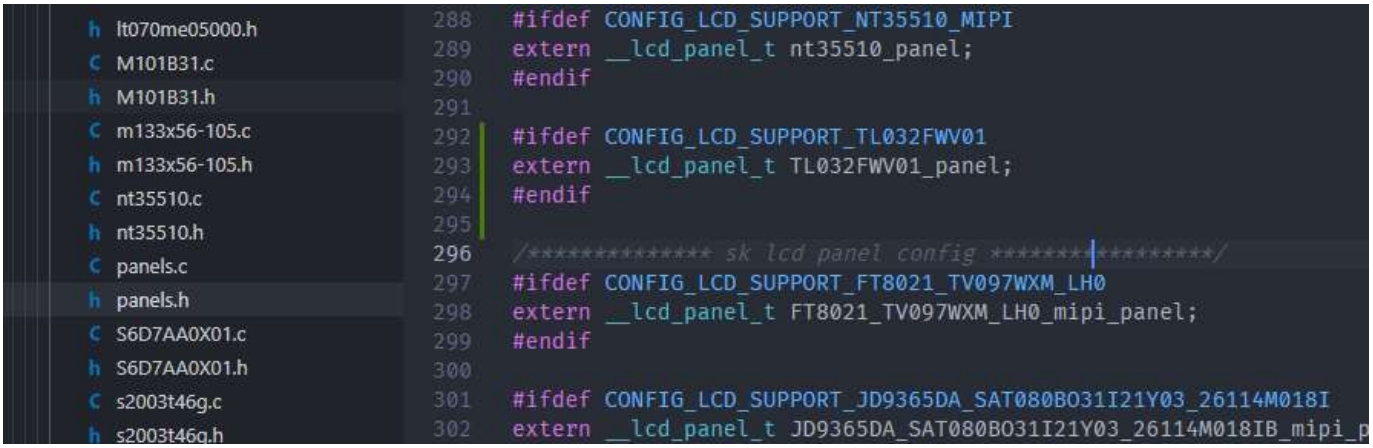
打开屏幕驱动同目录下的 `Kconfig`，增加引索


```
config LCD_SUPPORT_TL032FWV01
    bool "LCD support TL032FWV01 panel"
    default n
    ---help---
        If you want to support TL032FWV01 panel for display driver, select it.
```

找到同目录下的 `panel.c` 增加结构体。

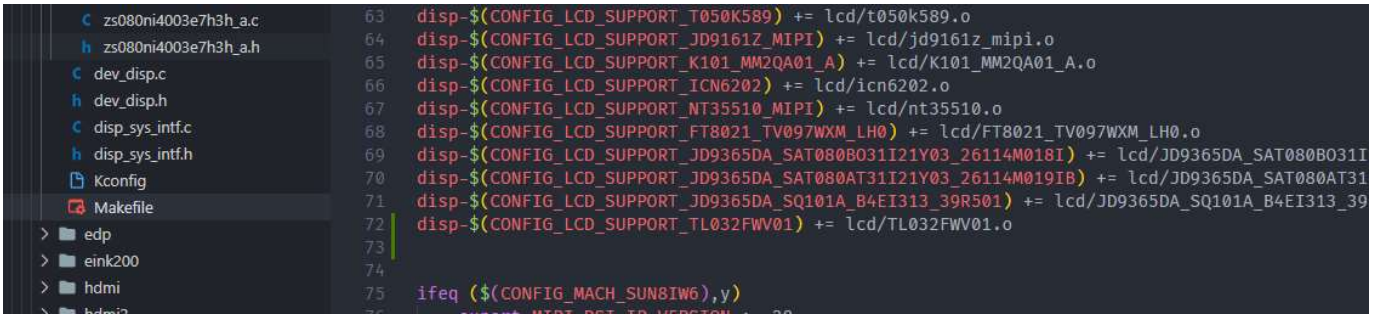
```
#ifdef CONFIG_LCD_SUPPORT_TL032FWV01
    &TL032FWV01_panel,
#endif
```

找到同目录下的 `panel.h` 增加指针。




```
#ifdef CONFIG_LCD_SUPPORT_TL032FWV01
extern __lcd_panel_t TL032FWV01_panel;
#endif
```

前往上一级目录，找到 `Makefile` 增加编译选项。




```
disp-$(CONFIG_LCD_SUPPORT_TL032FWV01) += lcd/TL032FWV01.o
```

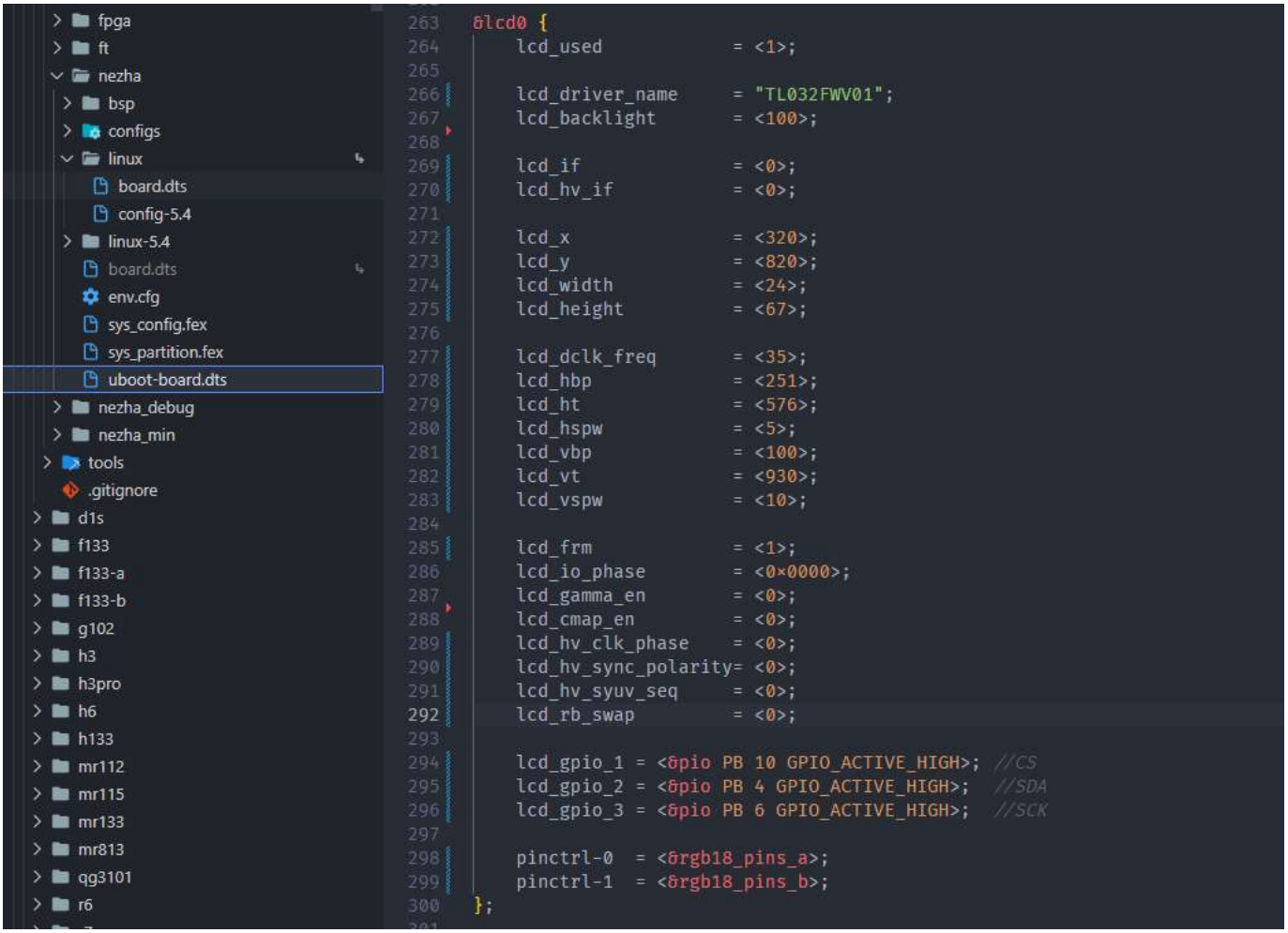
启用屏幕驱动

找到 U-Boot 所使用的 `defconfig` 文件，增加一行


```
CONFIG_LCD_SUPPORT_TL032FWV01=y
```

配置 U-Boot 设备树

打开项目对应的 `uboot-board.dts` 将设备树替换为 Linux 相同的即可。



编译测试

使用命令 `muboot` 编译 U-Boot，可以在编译的时候看到编译完成了。

```
CC      drivers/video/sunxi/disp2/disp/lcd/bp101wx1-206.o
CC      drivers/video/sunxi/disp2/disp/lcd/k101im2qa04.o
CC      drivers/video/sunxi/disp2/disp/lcd/k101im2byl02l.o
CC      drivers/video/sunxi/disp2/disp/lcd/fx070.o
CC      drivers/video/sunxi/disp2/disp/lcd/CC08021801_310_800X1280.o
CC      drivers/video/sunxi/disp2/disp/lcd/K080_IM2HYL802R_800X1280.o
LD      common/built-in.o
CC      drivers/video/sunxi/disp2/disp/lcd/tft08006.o
LD      cmd/built-in.o
CC      drivers/video/sunxi/disp2/disp/lcd/t050k589.o
CC      drivers/video/sunxi/disp2/disp/lcd/TL032FWV01.o
CC      lib/display_options.o
LD      board/sunxi/built-in.o
LD      lib/built-in.o
LD      drivers/video/sunxi/disp2/disp/disp.o
LD      drivers/video/sunxi/disp2/disp/built-in.o
LD      drivers/video/sunxi/disp2/built-in.o
LD      drivers/video/sunxi/built-in.o
LD      drivers/video/built-in.o
```

开机 LOGO

开机 LOGO 是由 UBOOT 所提供的支持，所以需要配置 UBOOT 的显示屏驱动。

在这之前，先前往 Uboot 检查是否开启了屏幕驱动。

```
brandy-2.0/u-boot-2018/configs/sun8iw21p1_defconfig
```

把 `CONFIG_DISP2_SUNXI=y` 取消注释

```
# CONFIG_ENABLE_ADVERT_PICTURE is not set
# CONFIG_SUNXI_SPINOR_JPEG is not set
# CONFIG_CMD_SUNXI_JPEG is not set
CONFIG_DISP2_SUNXI=y
# CONFIG_VDPO_DISP2_SUNXI is not set
# CONFIG_TV_DISP2_SUNXI is not set
# CONFIG_EDP_DISP2_SUNXI is not set
# CONFIG_EINK200_SUNXI is not set
```

然后如同 Kernel 一样，修改 Uboot 的设备树即可。

```
device/config/chips/v853/configs/vision/uboot-board.dts
```

```
&pio {
    rgb18_pins_a: rgb18@0 {
        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,function = "lcd";
        allwinner,muxsel = <2>;
        allwinner,drive = <3>;
        allwinner,pull = <0>;
    };

    rgb18_pins_b: rgb18@1 {
        allwinner,pins = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,pname = "PD0", "PD1", "PD2", "PD3", "PD4", "PD5", "PD6", "PD7",
"PD8", "PD9", \
        "PD10", "PD11", "PD12", "PD13", "PD14", "PD15", "PD16", "PD17", "PD18",
"PD19", \
        "PD20", "PD21";
        allwinner,function = "rgb18_suspend";
        allwinner,muxsel = <0xf>;
        allwinner,drive = <1>;
        allwinner,pull = <0>;
    };
};

&lcd0 {
    lcd_used            = <1>;

    lcd_driver_name     = "default_lcd";
    lcd_if              = <0>;
    lcd_x               = <800>;
    lcd_y               = <480>;
    lcd_width           = <108>;
    lcd_height          = <64>;
    lcd_dclk_freq       = <24>;

    lcd_pwm_used        = <1>;
    lcd_pwm_ch          = <9>;
    lcd_pwm_freq        = <50000>;
```

```
    lcd_pwm_pol          = <0>;
    lcd_pwm_max_limit    = <255>;

    lcd_ht               = <816>;
    lcd_hbp              = <12>;
    lcd_hspw             = <4>;
    lcd_vt               = <496>;
    lcd_vbp              = <12>;
    lcd_vspw             = <4>;

    lcd_lvds_if          = <0>;
    lcd_lvds_colordepth = <1>;
    lcd_lvds_mode        = <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>;
    pinctrl-0 = <&rgb18_pins_a>;
    pinctrl-1 = <&rgb18_pins_b>;
};
```

替换开机 LOGO 文件

如果希望修改开机 LOGO，可以修改下面路径里的这个图片文件

对于 Tina Linux 4.0（V853 之前的芯片，例如D1，D1s）

```
device/config/chips/d1-h/configs/nezha/configs
```

对于 Tina Linux 5.0 （V853 之后的芯片）

```
openwrt/target/v853/v853-common/boot-resource/boot-resource/bootlogo.bmp
```

注意，图片不宜过大，太大的图片容易导致加载缓慢。而且图片需要 BMP 格式，24位深

打包烧写即可，替换 bootlogo 不需要编译

显示命令行到屏幕上

这一部分仅 5.4 内核可以使用，4.9 内核不适用

Linux 自带有 FBCON 驱动，可以显示命令行到屏幕上作为小电脑使用。

配置 Kernel 选项

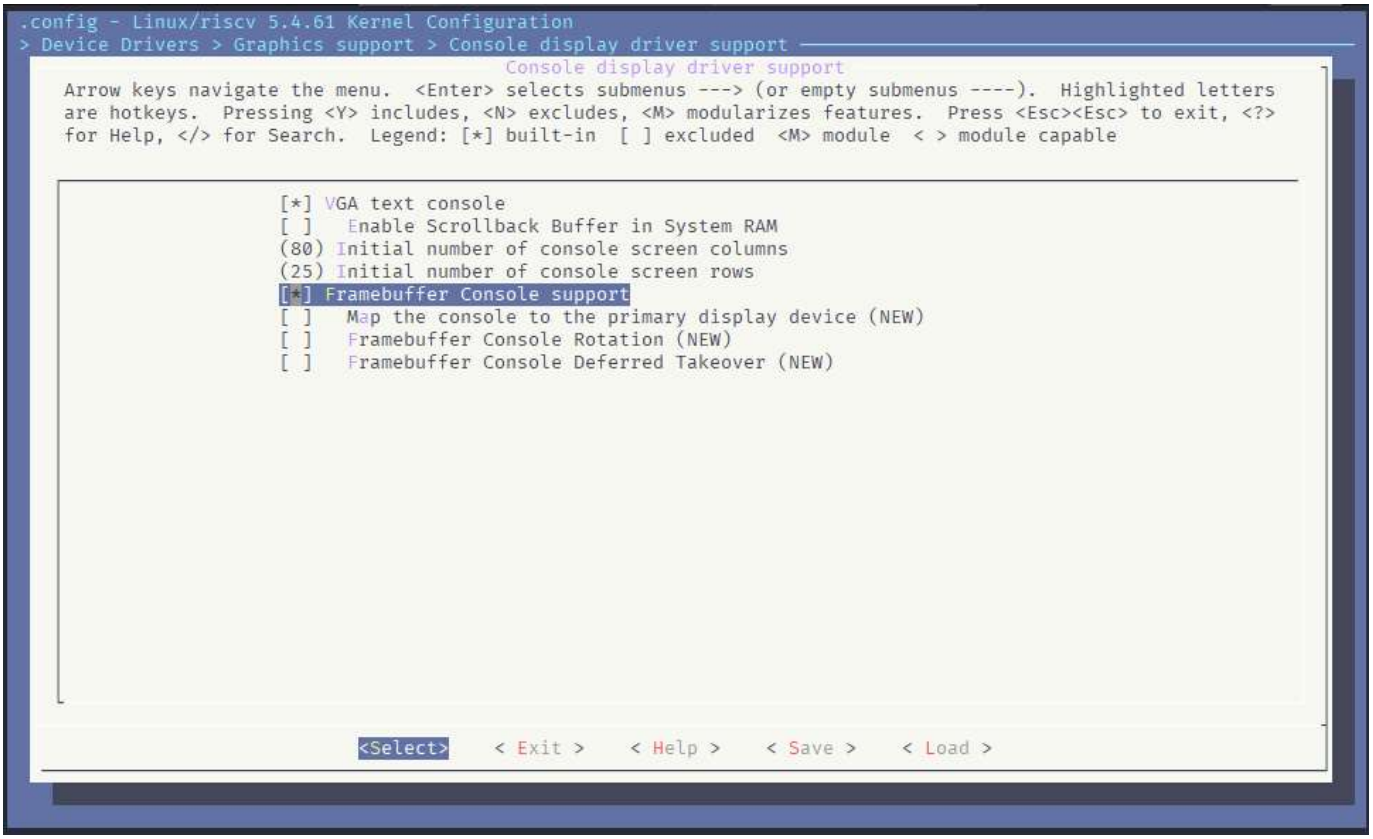
首先 `make kernel_menuconfig` 勾选以下配置

（1）选中 TTY 设备

```
Device Drivers > Character devices
    [*]   Virtual terminal
```

（2）启用 FBCON

```
Device Drivers > Graphics support > Console display driver support
    [*] VGA text console
    [*] Framebuffer Console support
```



（3）配置启动 `console` 输出

打开项目文件夹内的 `env.cfg` 在 `console=${console}` 前加入 `console=tty0`，对应自己的 `setargs`， `mmc` 启动使用 `setargs_mmc` ， `nand` 启动使用 `setargs_nand`，以此类推。

```
dsp0_partition=dsp0
#set kernel cmdline if boot.img or recovery.img has no cmdline we will use this
setargs_nand=setenv bootargs ubi.mtd=${mtd_name} ubi.block=0,${root_partition} earlyprintk=${earlyprintk} clk_ignore_unused initcall_debug=${initcall_debug}
setargs_nand_ubi=setenv bootargs ubi.mtd=${mtd_name} ubi.block=0,${root_partition} earlyprintk=${earlyprintk} clk_ignore_unused initcall_debug=${initcall_deb
setargs_mmc=setenv  bootargs earlyprintk=${earlyprintk} clk_ignore_unused initcall_debug=${initcall_debug} console=tty0 console=${console} loglevel=${logleve
#nand command syntax: sunxi_flash read address partition_name read_bytes
#0×4007f800 = 0×40080000(kernel entry) - 0×800(boot.img header 2k)
boot_dsp0=sunxi_flash read 45000000 ${dsp0_partition};bootr 45000000 0 0
boot_normal=sunxi_flash read 45000000 ${boot_partition};bootm 4500000
```

即可开机显示命令行