海奇LCD 用户使用手册

修订记录

版本号	日期	修改说明
V001-1-001	2023-07-10	提交LCD用户使用手册，支持2023年7月后的SDK，hcrtos 05y02，其他版本可以进行参考
V001-1-002	2024-01-02	LVDS添加了组之间的交换和lvds设置驱动能力的接口，实现LVDS显示屏与RGB显示能够快速进行切换，mipi增加了设置驱动能力的接口和NP交换的接口，规范DE的接口说明，增加了DGPLL的设置，132M和54M时钟，能够支持设置显示范围功能，各个SDK版本可以进行参考
V001-1-002	2024-05-16	增加6.10 系统打开pq时mipi的流程，该流程只针对2402Y.1及以上的SDK；增加8.5 backlight的接口功能说明；增加9.6 univdev通用的lcd初始化设备；增加9.7 LCD的接口说明；其他细节做了修改；

一、LCD显示屏及其应用领域

LCD（液晶显示器）是一种常见的平面显示技术，它使用液晶分子来控制光的透过和阻挡，从而产生图像。LCD显示屏具有轻薄、低功耗、高分辨率、色彩鲜艳等特点，因此在各种设备和应用领域中得到广泛使用。

LCD显示屏的应用领域包括但不限于以下几个方面：

- 1.消费电子：如电视机、智能手机、数字相框、投影仪、游戏机等。
- 2.工业自动化：如控制台显示器、工控机、自动售货机、ATM机等。
- 3.医疗设备：如X光显示屏、B超显示屏、血糖仪、心电监护器等。
- 4.车载信息娱乐系统：如汽车导航屏、后座娱乐系统、行车记录仪等。
- 5.安防监控：如监控器、视频门禁系统、可视对讲系统等。

随着科技的不断进步和广泛应用，LCD显示屏在未来的应用领域也将变得更加多样化和广泛化。

二、常用LCD显示接口简介

- RGB、MIPI和LVDS是三种常用于连接嵌入式系统和液晶显示屏的接口标准。这些标准定义了信号传输方式和数据格式，以确保正确的图像显示。
- 在嵌入式领域中，通过这些接口可连接各种尺寸和分辨率的液晶显示屏，如小型触摸屏幕、移动设备屏幕、医疗显示设备、控制台显示器等。选择合适的接口标准取决于具体应用需求和成本预算。
- RGB接口通常被用于连接较小的液晶显示屏，它可以提供高品质的图像，但需要较多的线路来传输数据，因此成本较高。
- LVDS接口则主要用于连接大型液晶显示屏，具有较高的带宽和抗干扰能力，但需要更高的设计难度和成本。
- MIPI接口使用低功耗、高速串行通信协议来传输数据，可以节省空间和功耗，适用于移动设备或带有内置触摸屏的液晶显示屏。

三、海奇芯片支持显示接口

3.1 海奇芯片支持的接口

目前海奇一代芯片能够支持RGB接口，二代芯片能够支持RGB，LVDS，MIPI，各芯片支持的列表，可供使用的时候进行参考。

✔代表支持，x代表不支持；

芯片	RGB	LVDS	MIPI
B100	✔	×	×
B200	✔	×	×
B210	✔	×	×
B300	✔	✔	×
B3100	✔	✔	×
B3120	✔	✔	×
C300	✔	✔	✔
C3000	✔	✔	✔
C3100	×	×	✔
C5200	✔	✔	✔
D3000	✔	✔	✔
D3100	✔	✔	×
D5200	✔	✔	✔

3.2 在hcrtos上面各 demo board 默认的配置

✔代表支持； x：不支持；*：demo板子未使用该接口。

配置	RGB	LVDS	MIPI
hichip_hc15xx_db_b100_v12_hcdemo_defconfig	1024*600（默认）	x	x
hichip_hc15xx_db_b200_v10_hcdemo_defconfig	800*480（默认）	x	x
hichip_hc15xx_db_b210_v10_hcdemo_defconfig	800*480（默认）	x	x
hichip_hc15xx_db_e100_v10_hcdemo_defconfig	1024*600（默认）	x	x
hichip_hc16xx_db_b300_v10_hcdemo_defconfig	*	1920×1080(VESA 默认)	x
hichip_hc16xx_db_b3100_v20_hcdemo_defconfig	800×480(RGB565 默认)	✔	x
hichip_hc16xx_db_b3120_v20_hcdemo_defconfig	800×480(RGB565 默认)	✔	x
hichip_hc16xx_db_c3000_v10_hcdemo_defconfig	*	1024×600(VESA默认)	*
hichip_hc16xx_db_c300_v10_hcdemo_defconfig	*	✔	1080×1920(默认)
hichip_hc16xx_db_c300_v20_hcdemo_defconfig	800×480(RGB666 默认)	*	✔
hichip_hc16xx_db_c3100_v20_hcdemo_defconfig	*	*	1920×1200(默认)
hichip_hc16xx_db_c5200_v10_hcdemo_defconfig	*	1024×600(VESA默认)	*
hichip_hc16xx_db_d3000_v10_hcdemo_defconfig	*	1024×600(VESA默认)	*
hichip_hc16xx_db_d3100_v10_hcdemo_defconfig	800×480(RGB565 默认)	✔	x
hichip_hc16xx_db_d3100_v20_hcdemo_defconfig	*	1024×600(VESA 默认)	x
hichip_hc16xx_db_d5200_v11_hcdemo_defconfig	800×480(RGB888 VESA 默认)	✔	✔

3.3 支持的显示标准

3.3.1 HC15XX的显示接口

显示接口	标准分辨率	协议标准
RGB	1920*1080@60HZ CLK:148.5MHZ	支持RGB565、RGB666、RGB888

3.3.2 HC16XX的显示接口

显示接口	标准分辨率	协议标准
RGB	1920*1080@60HZ CLK:148.5MHZ	支持RGB565、RGB666、RGB888
LVDS	双通道 1920*1080@60HZ CLK:148.5MHZ	支持 VESA 和 JEIDA格式
MIPI	单通道 1920*1200@60HZ CLK:148.5MHZ	支持RGB565、RGB666、RGB888

3.3.3 注意说明

> 目前海奇H1600在DE4K的模式下，MIPI理论上能够支持2.5K分辨率的显示屏，如2560*1440 P60，H16XX的mipi竖屏不能支持操作超过2047分辨率的显示屏，例如1080*2048分辨率的支持不了。

四、LCD显示屏的快速配置说明

一、前言说明
为了方便快速切换lcd显示屏，从tag-hcrtos-2023.05.y.2的版本开始，已经将H16xx 点亮过的所有lcd显示屏放在"hcrtos\board\hc16xx\common\dts\lcd\"的目录下了，需要配置lcd可以在这里进行参考；
为了方便快速切换lcd显示屏，从tag-hcrtos-2024.02.y的版本开始，已经将15xx 点亮过的lcd显示屏放在"hcrtos\board\hc15xx\common\dts\lcd\"的目录下了，需要配置lcd可以在这里进行参考；

二、H16xx LVDS快速配置
2.1、DTS配置说明
例如board\hc16xx\common\dts\hc16xx-db-d3100-v10.dts默认配置的是800×480的RGB显示屏，如果需要改成1024×600的LVDS显示屏，可以做以下修改；

```
lvds: lvds@0xb8860000 {
    status = "okay";//必要
};
// #include "lcd_rgb_800_480_rgb565.dtsi" /*注释掉默认的配置*/
#include "lcd_lvds_1024_600_vesa.dtsi" /*添加新的lcd显示屏配置*/
```

2.2、如有修改需要重新编译

```
cd hcrtos
make O=bl kernel-rebuild all
make kernel-rebuild all
```

三、H16xx RGB 快速配置说明
3.1、DTS配置说明
例如board\hc16xx\common\dts\hc16xx-db-d3100-v10.dts默认配置的是800×480的RGB显示屏；

```
lvds: lvds@0xb8860000 {
    status = "okay";//必要
};
#include "lcd_rgb_800_480_rgb565.dtsi" /*默认的配置*/
```

3.2、如有修改需要重新编译

```
cd hcrtos
make O=bl kernel-rebuild all
make kernel-rebuild all
```

四、H16xx MIPI 快速配置说明
4.1、DTS配置说明
例如board\hc16xx\common\dts\hc16xx-db-c3000-v11.dts默认配置的是 1200×1920的MIPI显示屏；

```
mipi: dsi0 {
    status = "okay";//必要
};
#include "lcd_mipi_1200_1920_rgb888.dtsi"/*默认的配置*/
```

4.2、如有修改需要重新编译

```
cd hcrtos
make O=bl kernel-rebuild all
make kernel-rebuild all
```

注意：2302y以上的版本才支持

五、H15xx RGB快速配置说明
5.1 DTS配置说明
例如board\hc15xx\common\dts\hc15xx-db-b100-hcdemo.dts 默认配置的是1024*600的RGB显示屏

```
    rgb: rgb {
        pinmux-active = <
                PINPAD_T01 PINMUX_T01_PRGB_G0
                PINPAD_R08 PINMUX_R08_PRGB_G1
                PINPAD_T00 PINMUX_T00_PRGB_G3
                PINPAD_T03 PINMUX_T03_PRGB_G2
                PINPAD_T02 PINMUX_T02_PRGB_G4
                PINPAD_T04 PINMUX_T04_PRGB_G6
                PINPAD_T05 PINMUX_T05_PRGB_G5
                PINPAD_T06 PINMUX_T06_PRGB_G7
                PINPAD_T07 PINMUX_T07_PRGB_B0
                PINPAD_T08 PINMUX_T08_PRGB_B1
                PINPAD_T09 PINMUX_T09_PRGB_B2
                PINPAD_T10 PINMUX_T10_PRGB_B3
                PINPAD_T11 PINMUX_T11_PRGB_B4
                PINPAD_T12 PINMUX_T12_PRGB_B5
                PINPAD_T13 PINMUX_T13_PRGB_B6
                PINPAD_T14 PINMUX_T14_PRGB_B7
                PINPAD_R00 PINMUX_R00_PRGB_R0
                PINPAD_R01 PINMUX_R01_PRGB_R1
                PINPAD_R02 PINMUX_R02_PRGB_R2
                PINPAD_R03 PINMUX_R03_PRGB_R3
                PINPAD_R04 PINMUX_R04_PRGB_R4
                PINPAD_R05 PINMUX_R05_PRGB_R5
                PINPAD_R06 PINMUX_R06_PRGB_R6
                PINPAD_R07 PINMUX_R07_PRGB_R7
                PINPAD_T15 PINMUX_T15_PRGB_CLK
                PINPAD_L08 PINMUX_L08_PRGB_HSYN
                PINPAD_L09 PINMUX_L09_PRGB_VSYNC
                PINPAD_L10 PINMUX_L10_PRGB_DE>;
        status = "okay";//必要
    };
#include "lcd_rgb_1024_600.dtsi"/*默认的配置*/

注意：2402y以上的版本才支持
```

五、HC16XX RGB 与LVDS的配置说明

5.1 配置demo板

配置rgb编译可以参考配置

```
cd hcrtos
rm -rf output*
rm -rf bl
make O=bl hichip_hc16xx_db_d5200_v11_hcdemo_bl_defconfig
make O=bl all
make hichip_hc16xx_db_d5200_v11_hcdemo_defconfig
make all
```

配置lvds可以参考配置

```
cd hcrtos
rm -rf output*
rm -rf bl
make O=bl hichip_hc16xx_db_d3100_v20_hcdemo_bl_defconfig
make O=bl all
make hichip_hc16xx_db_d3100_v20_hcdemo_defconfig
make all
```

5.2 查看menuconfig是否打开了lvds驱动(默认打开)

```
lvds和rgb都是打开CONFIG_DRV_LVDS的配置
1、查看bootloader是否配置了lvds，相关寄存器的配置是在bootloader阶段就已经完成了初始化。
make O=bl menuconfig
cd hcrtos
x There is no help available for this option.
  x Symbol: CONFIG_DRV_LVDS [=y]
  x Type  : bool
  x Prompt: lvds
  x   Location:
  x     -> Components
  x -> kernel (BR2_PACKAGE_KERNEL [=y])
  x -> Drivers
  x   Defined at drivers:165
  x   Depends on: BR2_PACKAGE_KERNEL [=y]

[*] lvds(必选)

2、查看kernel是否配置了lvds
make menuconfig
x There is no help available for this option.
  x Symbol: CONFIG_DRV_LVDS [=y]
  x Type  : bool
  x Prompt: lvds
  x   Location:
  x     -> Components
  x -> kernel (BR2_PACKAGE_KERNEL [=y])
  x -> Drivers
  x   Defined at drivers:165
  x   Depends on: BR2_PACKAGE_KERNEL [=y]

 [*] lvds(如果在bootloader阶段完成初始化后，不会重复执行初始化)
```

5.3 配置DE

```
例如 board\hc16xx\common\dts\lcd\lcd_rgb_800_480_rgb888.dtsi

&DE {
    tvtype = <15>;//1.一般配置为15
    VPInitInfo {
        rgb-cfg{
            b-rgb-enable = <1>;//2、使能RGB设置为 1
            /*
             * 0: MODE_PRGB
             * 1: MODE_SRGB
             */
```

```
            rgb-mode = <0>;//3、设置为并行模式
            /*
             * 0: MODE_PRGB_888_10bit
             * 1: MODE_PRGB_666
             */
            prgb-mode = <0>; //4、设置为RGB888的模式
            lcd-width = <800>;//5、lcd-width与h-active-len需要一致
            b-disable-ejtag = <1>;
            b-dlpc-enale = <0>;
            timing-para {
                /* bool type, 0 or 1*/
                b-enable = <1>;//6、使能为，使能之后才能配置以下的参数
                h-display_len = <0>;//7、可选，设置h方向的显示范围
                v-display_len = <0>;//   可选，设置v方向的显示范围
                /*
                 * VPO_RGB_CLOCK_27M = 1,
                 * VPO_RGB_CLOCK_33M = 2,
                 * VPO_RGB_CLOCK_49M = 3,
                 * VPO_RGB_CLOCK_66M = 4,
                 * VPO_RGB_CLOCK_74M = 5,
                 * VPO_RGB_CLOCK_85M = 6,
                 * VPO_RGB_CLOCK_108M = 7,
                 * VPO_RGB_CLOCK_6_6M = 8,
                 * VPO_RGB_CLOCK_9M = 9,
                 * VPO_RGB_CLOCK_39_6M = 10,
                 * VPO_RGB_CLOCK_74_25M = 11,
                 * VPO_RGB_CLOCK_148_5M = 12,
                 * VPO_RGB_CLOCK_54M = 13,
                 * VPO_RGB_CLOCK_132M = 14,//H1600
                 * VPO_RGB_CLOCK_297M = 15,//H1600 only De4k
                 * VPO_RGB_CLOCK_DPLL = 0xFF,//H1600
                */
                output-clock = <2>;//8、output-clock = h-total-len * v-total-len
* 频率（60HZ）
                                                9、按照屏厂提供参数填写相关屏参
                h-total-len = <1000>;//h-total-len = h-active-len + h-front-len
+ h-sync-len+h-back-len
                v-total-len = <550>;//v-total-len = v-active-len + v-front-len +
v-sync-len+v-back-len

                h-active-len = <800>;
                v-active-len = <480>;

                h-front-len = <184>;
                h-sync-len = <6>;
                h-back-len = <10>;

                v-front-len = <22>;
                v-sync-len = <28>;
                v-back-len = <20>;
                /* bool type, 0 or 1*/
                h-sync-level = <1>;
                /* bool type, 0 or 1*/
                v-sync-level = <1>;
                frame-rate = <60000>;//10、设置帧率 60HZ

                /*
                31  De_dig_pll_en
```

```
                25:16   DE_DIG_PLL_M_ctrl
                13:8    DE_DIG_PLL_N_ctrl
                5:0 DE_DIG_PLL_L_ctrl
                de_digpll_clock = 24 *(M+1)/((N+1)*(L+1))
                */
                dpll-clock-reg-value = <0x0>;
            };
        };
    };
};
```

5.4 配置RGB或者LVDS

- 5.4.1 配置RGB

```
1、lvds使能
  board\hc16xx\common\dts\hc16xx-db-d5200-v11.dts
  lvds: lvds@0xb8860000 {
    status = "okay";
  };

  2、配置lvds节点内容
  例如board\hc16xx\common\dts\lcd\lcd_rgb_800_480_rgb888.dtsi //配置rgb565 linux的节点
    &lvds {
        reg = <0xB8860000 0x200>, <0xB8800000 0x500>;
        /*
         * "lvds"       : LVDS screen
         * "rgb888"     : RGB screen
         * "rgb666"     : RGB screen
         * "rgb565"     : RGB screen
         * "i2so"       : I2S OUT
         * "gpio"       : only GPIO out
         */
        lvds_ch0-type = "rgb888";//1、必选，设置成RGB888模式
        lvds_ch1-type = "rgb888";//必选，设置成RGB888模式

        /*
         * 0: LVDS_TTL_DRIVE_STRENGTH_WEAKEST,
         * 1: LVDS_TTL_DRIVE_STRENGTH_NORMAL,
         * 2: LVDS_TTL_DRIVE_STRENGTH_ENHANCE,
         * 3: LVDS_TTL_DRIVE_STRENGTH_STRONGEST,
         */
        ttl-drive-strength = <1>;//2、可选，设置RGB的驱动能力，默认是1

        /*
         *0 : RGB CLOCK NO INVERSION
         *1 : RGB CLOCK INVERSION
         */
        rgb-clk-inv = <1>;//3、可选，设置RGB的时钟是否需要取反

        /*
         * rgb gbr bgr
         * rbg grb brg
         */
        rgb-src-sel = "rgb";//4、可选，设置RGB的显示方式，默认是rgb模式
```

```
        /*
         * 0: E_SRC_SEL_FXDE = 0
         * 1: E_SRC_SEL_4KDE = 1
         * 2: E_SRC_SEL_HDMI_RX = 2
         * 3: E_SRC_SEL_FXDE_ = 3
         * 4: E_SRC_2_SEL_FXDE_LOW_TO_HIGH = 4 //need clsoe pq
         * 5: E_SRC_2_SEL_DE4K_LOW_TO_HIGH = 5 //need clsoe pq
         * 6: E_SRC_2_SEL_HDMI_RX_LOW_TO_HIGH = 6 //need clsoe pq
         * 7: E_SRC_2_SEL_PQ_LOW_TO_HIGH = 7
         */
        src-sel = <CONFIG_DE4K_OUTPUT_SUPPORT>;

        pinmux-rgb888 = <PINPAD_B00 1 PINPAD_B01 1 PINPAD_B02 1 PINPAD_B03 1
PINPAD_B04 1 PINPAD_B05 1 PINPAD_B06 1 PINPAD_B07 1>;/*RGB888 other pin
set*///RGB888的模式下才会去调用
        pinmux-rgb666 = <PINPAD_B04 4 PINPAD_B07 4>; /*RGB666 other pin set*/
//RGB666的模式下才会去调用
        // status = "okay";
    };
```

- 5.4.2 配置lvds

```
1、使能lvds
  board\hc16xx\common\dts\hc16xx-db-d3100-v20.dts

  lvds: lvds@0xb8860000 {
    status = "okay";
  };

  2、配置lvds节点的内容
  board\hc16xx\common\dts\lcd\lcd_lvds_1024_600_vesa.dtsi //配置lvds vesa的节点
&lvds{
    reg = <0xB8860000 0x200>, <0xB8800000 0x500>;

    /*
     * "lvds"       : LVDS screen
     * "rgb888"     : RGB screen
     * "rgb666"     : RGB screen
     * "rgb565"     : RGB screen
     * "i2so"       : I2S OUT
     * "gpio"       : only GPIO out
     */
    lvds_ch0-type = "lvds";//1、必选，设置输出的通道 LVDS D0~D3
    lvds_ch1-type = "lvds";//               LVDS D4~D7

    /*
     * 0: E_CHANNEL_MODE_SINGLE_IN_SINGLE_OUT ,
     * 1: E_CHANNEL_MODE_SINGLE_IN_DUAL_OUT
     */
    channel-mode = <0>;//2、可选， 通道输出类型，单通道显示，双通道显示
    /*
     * 0: VESA
     * 1: JEDIA
     */
    map-mode = <0>;//3、可选,设置屏的信号格式 0:VESA 1:JEDIA
    ch0-src-sel = <0>;
    ch1-src-sel = <0>;
```

```
    ch0-invert-clk-sel = <0>;
    ch1-invert-clk-sel = <0>;
    ch0-clk-gate = <0>;
    ch1-clk-gate = <0>;
    hsync-polarity = <1>;//与 de-engine 的h-sync-level 保持一致
    vsync-polarity = <1>;//与 de-engine 的v-sync-level 保持一致
    /*
     * 0: E_ADJUST_MODE_FRAME_START
     * 1: E_ADJUST_MODE_HSYNC_POS
     * 2: E_ADJUST_MODE_VSYNC_POS
     */
    even-odd-adjust-mode = <0>;
    even-odd-init-value = <0>;//4、可选,奇偶校验初始化值，一般当channel-mode == 1时，
需要设置为1
    /*
     * 0: NOT SWAP
     * 1: ERROR SWAP
     * 2: FIX SWAP
     */
    chx-swap-ctrl = <0>;//5、可选,设置通道之间是否需要交换
    /*
     * 0: LVDS_LLD_PHY_DRIVE_STRENGTH_WEAKEST
     * 1: LVDS_LLD_PHY_DRIVE_STRENGTH_NORMAL,
     * 2: LVDS_LLD_PHY_DRIVE_STRENGTH_STRONGEST,
     * */
    lvds-drive-strength = <1>;//6、可选,设置LVDS的驱动能力，默认是
LVDS_LLD_PHY_DRIVE_STRENGTH_NORMAL
    /*
     * 0: E_SRC_SEL_FXDE = 0
     * 1: E_SRC_SEL_4KDE
     * 2: E_SRC_SEL_HDMI_RX
     */
    src-sel = <CONFIG_DE4K_OUTPUT_SUPPORT>;
    // status = "okay";
};
```

5.5 编译

- 5.1 重新编译

```
cd hcrtos
hcrtos$ make O=bl kernel-rebuild all; make kernel-rebuild all
```

5.6 注意事项：

- 5.6.1 dts注意

假设`board\hc16xx\common\dts\hc16xx-db-d5200-v11.dts`配置了mipi: dsi0，请将status = "disable"，并重新编译：

```
cd hcrtos
hcrtos$ make O=bl kernel-rebuild all; make kernel-rebuild all
```

- 5.6.2 lvds通道的说明

正确的配置，通道D0~D3可以配置成 "lvds","i2so","gpio",D4~D7可以配置成 "lvds","i2so","gpio"
例如正确的配置：

```
lvds_ch0-type = "lvds"
lvds_ch1-type = "gpio"

lvds_ch0-type = "i2so"
lvds_ch1-type = "lvds"

lvds_ch0-type = "i2so"
lvds_ch1-type = "i2so"
```

错误的配置：
不能够出现 rgb565 rgb666 rgb888 gpio i2so 同时组合的配置，例如：

```
lvds_ch0-type = "rgb565";//rgb666 rgb888
lvds_ch1-type = "gpio";

lvds_ch0-type = "rgb565";//rgb666 rgb888
lvds_ch1-type = "i2so";

lvds_ch0-type = "gpio";
lvds_ch1-type = "i2so";
```

- 5.6.3 lvds的节点说明

```
对应驱动的位置在 components\kernel\source\drivers\lvds\lvds.c
```

- 5.6.4 lvds作为gpio时，增加lvds gpio驱动能力的方法

```
&lvds{
    lvds_ch0-type = "lvds";
    lvds_ch1-type = "gpio";
    /*
    * 0: LVDS_TTL_DRIVE_STRENGTH_WEAKEST,
    * 1: LVDS_TTL_DRIVE_STRENGTH_NORMAL,
    * 2: LVDS_TTL_DRIVE_STRENGTH_ENHANCE,
    * 3: LVDS_TTL_DRIVE_STRENGTH_STRONGEST,
    */
    ttl-drive-strength = <1>;//add this，增加lvds通道1 gpio的驱动能力
};
```

5.7 lvds的接口说明

```
// lvds的接口路径
// components\kernel\source\include\uapi\hcuapi\lvds.h

#define LVDS_SET_CHANNEL_MODE       _IO (LVDS_IOCBASE, 1)
#define LVDS_SET_MAP_MODE       _IO (LVDS_IOCBASE, 2)
```

```
#define LVDS_SET_CH0_SRC_SEL         _IO (LVDS_IOCBASE, 3)
#define LVDS_SET_CH1_SRC_SEL         _IO (LVDS_IOCBASE, 4)
#define LVDS_SET_CH0_INVERT_CLK_SEL _IO (LVDS_IOCBASE, 5)
#define LVDS_SET_CH1_INVERT_CLK_SEL _IO (LVDS_IOCBASE, 6)
#define LVDS_SET_CH0_CLK_GATE        _IO (LVDS_IOCBASE, 7)
#define LVDS_SET_CH1_CLK_GATE        _IO (LVDS_IOCBASE, 8)
#define LVDS_SET_HSYNC_POLARITY      _IO (LVDS_IOCBASE, 9)
#define LVDS_SET_VSYNC_POLARITY      _IO (LVDS_IOCBASE, 10)
#define LVDS_SET_EVEN_ODD_ADJUST_MODE   _IO (LVDS_IOCBASE, 11)
#define LVDS_SET_EVEN_ODD_INIT_VALUE    _IO (LVDS_IOCBASE, 12)
#define LVDS_SET_SRC_SEL        _IO (LVDS_IOCBASE, 13)
#define LVDS_SET_RESET          _IO (LVDS_IOCBASE, 14)
#define LVDS_SET_POWER_TRIGGER      _IO (LVDS_IOCBASE, 15)
// 以上都是lvds硬件接口，客户可以不用关注

#define LVDS_SET_GPIO_OUT       _IOW (LVDS_IOCBASE, 16, struct lvds_set_gpio)
#define LVDS_SET_GPIO_BACKLIGHT     _IO (LVDS_IOCBASE, 17)      //!< param:
value: 0 1
#define LVDS_SET_PWM_BACKLIGHT      _IO (LVDS_IOCBASE, 18)      //!< param: Duty
cycle: 0~100
#define LVDS_SET_PWM_VCOM       _IO (LVDS_IOCBASE, 19)      //!< param: Duty
cycle: 0~100
#define LVDS_SET_GPIO_POWER     _IO (LVDS_IOCBASE, 20)      //!< param: value: 0
1

#define LVDS_SET_TRIGGER_EN     _IO (LVDS_IOCBASE, 21)//硬件接口，客户不必关注
#define LVDS_GET_CHANNEL_MODE       _IOR (LVDS_IOCBASE, 22, enum
LVDS_CHANNEL_MODE) //硬件接口，客户不必关注
#define LVDS_GET_CHANNEL_INFO       _IOR (LVDS_IOCBASE, 23, struct lvds_info)
//get lvds channel info
#define LVDS_SET_CHANNEL_INFO       _IOW (LVDS_IOCBASE, 24, struct lvds_info)
//set lvds channel info
```

5.7.1 LVDS_SET_GPIO_OUT -- 设置gpio输出接口

参数	描述
lvds_pad	struct lvds_set_gpio
返回	——
无	——

代码示例

```
dts上需要进行设置
&lvds{
    lvds_ch0-type = "gpio";
    lvds_ch1-type = "gpio";
    status = "okay";
};

void io_lvds_gpio_set_output(unsigned int pad,unsigned char value)
{
    int fd = 0;
    fd = open("/dev/lvds", O_RDWR);
```

```
    if( fd < 0){
        printf("open /dev/lvds failed, ret=%d\n",fd);
        return;
    }
    struct lvds_set_gpio lvds_pad;
    lvds_pad.padctl=pad;
    lvds_pad.value=value;
    printf("pad= %d value=%d\n",lvds_pad.padctl,lvds_pad.value);
    ioctl(fd,LVDS_SET_GPIO_OUT,&lvds_pad);
    close(fd);
}

io_lvds_gpio_set_output(PINPAD_LVDS_DP5, 0);// DP5 拉低
io_lvds_gpio_set_output(PINPAD_LVDS_DP5, 1);// DP5 拉高
```

5.7.2、LVDS_SET_GPIO_BACKLIGHT -- 设置背光值

参数	描述
val	0,1
返回	——
无	——

示例代码

```
dts上的设置
lvds@0xb8860000 {
    lcd-backlight-gpios-rtos = <PINPAD_T04 GPIO_ACTIVE_HIGH PINPAD_T02
GPIO_ACTIVE_LOW>;
    status = "okay";
};

    int lvds_fd = 1;
    lvds_fd = open("/dev/lvds",O_RDWR);
    if (lvds_fd < 0) {
        return -1;
    }
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,0);//PINPAD_T04 拉低，PINPAD_T02 拉高
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,1);//PINPAD_T04 拉高，PINPAD_T02 拉低
    close(lvds_fd);
```

5.7.3、LVDS_SET_PWM_BACKLIGHT -- 设置背光亮度

参数	描述
val	0~100
返回	——
无	——

示例代码

```
dts上的设置
lvds@0xb8860000 {
    backlight-pwmdev = "/dev/pwm2"; //背光设备
    backlight-frequency = <1000000>;//背光频率
    backlight-duty = <100>;          //背光占空比
    status = "okay";
};

    int lvds_fd = 1;
    lvds_fd = open("/dev/lvds",O_RDWR);
    if (lvds_fd < 0) {
        return -1;
    }
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,100);     //背光占空比设置为100
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,0);       //背光占空比设置为0
    close(lvds_fd);
```

5.7.4、LVDS_SET_PWM_VCOM -- 设置VCOM亮度

参数	描述
val	0~100
返回	——
无	——

示例代码

```
dts上的设置
lvds@0xb8860000 {
    vcom-pwmdev = "/dev/pwm2";  //vcom设备
    vcom-frequency = <1000000>;//vcom频率
    vcom-duty = <100>;          //vcom占空比
    status = "okay";
};

    int lvds_fd = 1;
    lvds_fd = open("/dev/lvds",O_RDWR);
    if (lvds_fd < 0) {
        return -1;
    }
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,100);     //vcom pwm 占空比设置为100
    ioctl(lvds_fd, LVDS_SET_GPIO_BACKLIGHT,0);       //vcom pwm 占空比设置为0
    close(lvds_fd);
```

5.7.5、LVDS_SET_GPIO_POWER -- 设置电源值

参数	描述
val	0,1
返回	——
无	——

示例代码

```
dts上的设置
lvds@0xb8860000 {
    lcd-power-gpios-rtos = <PINPAD_T04 GPIO_ACTIVE_HIGH PINPAD_T02
GPIO_ACTIVE_LOW>;
    status = "okay";
};

    int lvds_fd = 1;
    lvds_fd = open("/dev/lvds",O_RDWR);
    if (lvds_fd < 0) {
        return -1;
    }
    ioctl(lvds_fd, LVDS_SET_GPIO_POWER ,0);//PINPAD_T04 拉低，PINPAD_T02 拉高
    ioctl(lvds_fd, LVDS_SET_GPIO_POWER ,1);//PINPAD_T04 拉高，PINPAD_T02 拉低
    close(lvds_fd);
```

5.7.6、LVDS_GET_CHANNEL_INFO -- 获得LVDS通道的信息

参数	描述
lvds_info_t lvds_info	lvds通道的信息
返回	——
无	——

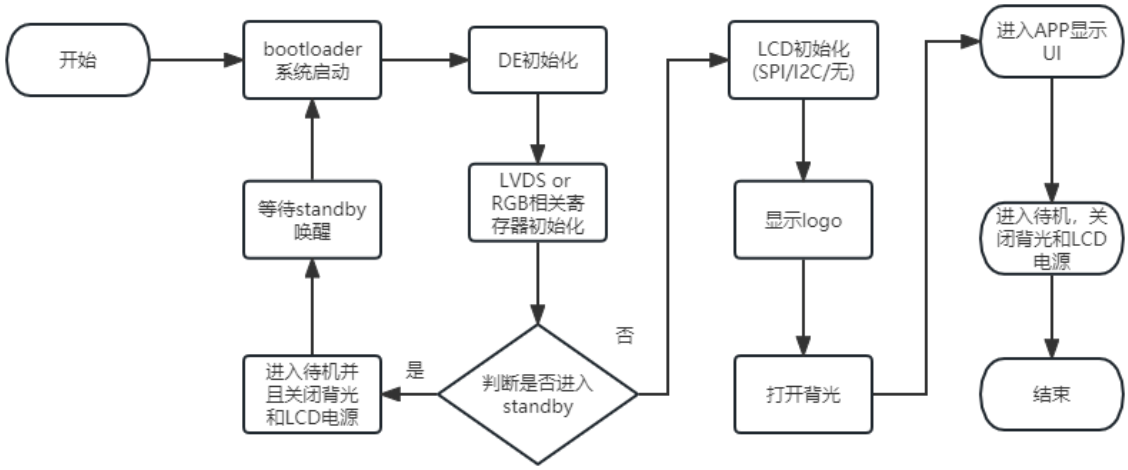
```
int lvds_fd = 1;
lvds_info_t lvds_info;
lvds_fd = open("/dev/lvds",O_RDWR);
if (lvds_fd < 0) {
    return -1;
}
ioctl(lvds_fd, LVDS_GET_CHANNEL_INFO ,&lvds_info);
close(lvds_fd);
```

5.7.7、LVDS_SET_CHANNEL_INFO -- 改变LVDS通道的信息

参数	描述
lvds_info_t lvds_info	lvds通道的信息
返回	——
无	——

```
int lvds_fd = 1;
lvds_info_t lvds_info;
lvds_fd = open("/dev/lvds",O_RDWR);
if (lvds_fd < 0) {
    return -1;
}
ioctl(lvds_fd, LVDS_GET_CHANNEL_INFO ,&lvds_info);
ioctl(lvds_fd, LVDS_SET_CHANNEL_INFO ,&lvds_info);
close(lvds_fd);
```

5.8 LVDS&RGB的显示流程



六、HC16XX MIPI的配置

6.1 板级参考配置

MIPI的相关配置只能在hcrtos里面进行配置

```
cd hcrtos
rm -rf output*
make O=bl hichip_hc16xx_db_c300_v10_hcdemo_bl_defconfig
make O=bl all
make hichip_hc16xx_db_c300_v10_hcdemo_defconfig
make all
```

6.2 查看menuconfig是否配置了mipi

```
cd hcrtos
1、查看bootloader是否配置了mipi，相关寄存器的配置是在bootloader阶段就已经完成了初始化。(必选)
make O=bl menuconfig
x There is no help available for this option.
  x Symbol: CONFIG_DRV_MIPI [=y]
  x Type  : bool
  x Prompt: mipi
  x   Location:
  x     -> Components
```

```
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
  x   Defined at drivers:177
  x   Depends on: BR2_PACKAGE_KERNEL [=y]

2、查看kernel是否配置了mipi(可选)
make menuconfig
x There is no help available for this option.
  x Symbol: CONFIG_DRV_MIPI [=y]
  x Type  : bool
  x Prompt: mipi
  x   Location:
  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
  x   Defined at drivers:177
  x   Depends on: BR2_PACKAGE_KERNEL [=y]
```

6.3 配置DE

```
board\hc16xx\common\dts\lcd\lcd_mipi_hq_1280_720_rgb888.dtsi
&DE {
    tvtype = <15>;//1.只能为15
    VPInitInfo {
        rgb-cfg{
            b-rgb-enable = <1>;//2、使能RGB设置为 1
            /*
                * 0: MODE_PRGB
                * 1: MODE_SRGB
                */
            rgb-mode = <0>;//3、设置为并行模式
            /*
                * 0: MODE_PRGB_888_10bit
                * 1: MODE_PRGB_666
                */
            prgb-mode = <0>;//4、设置为RGB888的模式
            lcd-width = <1080>;//5、lcd-width与h-active-len需要一致
            screen_timing: timing-para {
                /* bool type, 0 or 1*/
                b-enable = <1>;//6、使能为，使能之后才能配置以下的参数
                h-display_len = <0>;//设置h方向显示范围，为0会与h-active-len保持一致
                v-display_len = <0>;//设置h方向显示范围，为0会与h-active-len保持一致
                /*
                    * VPO_RGB_CLOCK_27M = 1,
                    * VPO_RGB_CLOCK_33M = 2,
                    * VPO_RGB_CLOCK_49M = 3,
                    * VPO_RGB_CLOCK_66M = 4,
                    * VPO_RGB_CLOCK_74M = 5,
                    * VPO_RGB_CLOCK_85M = 6,
                    * VPO_RGB_CLOCK_108M = 7,
                    * VPO_RGB_CLOCK_6_6M = 8,
                    * VPO_RGB_CLOCK_9M = 9,
                    * VPO_RGB_CLOCK_39_6M = 10,
                    * VPO_RGB_CLOCK_74_25M = 11,//H1600
                    * VPO_RGB_CLOCK_148_5M = 12,//H1600
                */
```

```
                output-clock = <11>;//7、output-clock = h-total-len * v-total-len
* 频率（60HZ）
                                                //8、按照屏厂提供参数填写相关屏参
                h-total-len = <1650>;
                v-total-len = <750>;

                h-active-len = <1280>;
                v-active-len = <720>;

                h-front-len = <110>;
                h-sync-len = <40>;
                h-back-len = <220>;

                v-front-len = <5>;
                v-sync-len = <5>;
                v-back-len = <20>;
                /* bool type, 0 or 1*/
                h-sync-level = <1>;
                /* bool type, 0 or 1*/
                v-sync-level = <1>;

                /*
                31  De_dig_pll_en
                25:16   DE_DIG_PLL_M_ctrl
                13:8    DE_DIG_PLL_N_ctrl
                5:0 DE_DIG_PLL_L_ctrl
                de_digpll_clock = 24 *(M+1)/((N+1)*(L+1))
                */
                dpll-clock-reg-value = <0x0>;//设置数值时钟，如果模拟时钟不满足需要时可
以用数字时钟进行配置
                frame-rate = <600000>;//9、设置帧率 60HZ
            };
        };
    };
};
```

6.4 配置mipi

```
1、使能mipi
board\hc16xx\common\dts\hc16xx-db-c300-v10.dts
mipi: dsi0 {
    //1、配置gpio
    enable-gpios=<PINPAD_L11 0 1>;//设置 GPIO L11引脚，设置为GPIO输出模式，高电平起作
用，可选
    reset-gpios=<PINPAD_L00 0 0>;//设置 GPIO L0引脚，设置为GPIO输出模式，低电平起作用，
可选
    //2、设置延时
    init-delay-ms = <20>;//初始化enable-gpios reset-gpios引脚之后，输出均为低，然后延时
20毫秒，之后，可选
    enable-delay-ms = <120>;//将enable-gpios 引脚使能有效值（本设置拉高），然后延时120毫
秒，之后，可选
    reset-delay-ms = <120>;//将reset-gpios 引脚使能有效值（本设置拉低），然后延时120毫
秒，然后将复位引脚释放（本设置拉高），之后，可选
    prepare-delay-ms = <120>;//延时120毫秒，开始发送命令，可选
    // default-off;//客户自定义，默认关闭，等待外部调用
    status = "okay";//必要
};
```

2、配置mipi节点的内容
board\hc16xx\common\dts\lcd\lcd_mipi_ILI7807D_1080_1920_rgb888.dtsi

```
&mipi {
        reg = <0xb884A000 0x300>, <0xb8800444 0x8>,<0xb8800080 0x8>;
        /*
         * MIPI_COLOR_RGB565_0  =   0
         * MIPI_COLOR_RGB565_1  =   1
         * MIPI_COLOR_RGB565_2  =   2
         * MIPI_COLOR_RGB666_0  =   3
         * MIPI_COLOR_RGB666_1  =   4
         * MIPI_COLOR_RGB888    =   5
         */
        dsi,format =<5>;//3、设置输出的格式，必要

        dsi,lanes = <4>;// 1= <dsi,lans <=4 //4、设置lane数，必要

        /*
         * bit 6:5 :Generic interface read-back Virtual Channel identification.
         * bit 4 :Enables CRC reception and error reporting.
         * bit 3 :Enables ECC reception, error correction and reporting
         * bit 2 :MIPI_Bus_Turn_request.
         * bit 1 :Enables EOTp reception.
         * bit 0 :Enables EOTp transmission.
         */
        dsi,cfg = <0x1c>;//5、可选，默认为 0x1C

        /* If there is no "dsi,flags", "dsi,flags" = <0x3FD>;
         * bit 11: Enables the request for an acknowledge response at the end of
a fram
         * bit 10: Enables the transmission of null packets in the HACT period
         * bit 9:  Enables the transmission of multi video packets in the HACT
period
         * bit 8:  Enables return to Low Power inside HFP period when timing
allows.
         * bit 7:  Enables return to Low Power inside HBP period when timing
allows
         * bit 6:  Enables return to Low Power inside VACT period when timing
allows.
         * bit 5:  Enables return to Low Power inside VFP period when timing
allows
         * bit 4:  Enables return to Low Power inside VBP period when timing
allows
         * bit 3:  Enables return to Low Power inside VSA period when timing
allows
         * bit 2:1 Selects video mode transmission type. 0: Non-burst with Sync
pulses; 1: Non-burst with Sync events; 2-3: Burst with Sync pulses.
         * bit 0:  Enables DPI Video mode transmission.
         */
        dsi,flags = <0x3FD>;//6、可选，不设置默认值为0x3fd，一般不需要进行设置

        /*
         * 0: MIPI_DRIVE_STRENGTH_MIDIMUN,
         * 1: MIPI_DRIVE_STRENGTH_MIDDLE,
         * 2: MIPI_DRIVE_STRENGTH_BIGGER, default
         * 3: MIPI_DRIVE_STRENGTH_MAXIMUM,
        */
        mipi-drive-strength = <3>;//7、可选，设置mipi的驱动能力，默认是3
```

```
        /*
         * bit 4 : MIPI_DSI_DATA3_LANE_NP_SWAP
         * bit 3 : MIPI_DSI_DATA2_LANE_NP_SWAP
         * bit 2 : MIPI_DSI_DATA1_LANE_NP_SWAP
         * bit 1 : MIPI_DSI_DATA0_LANE_NP_SWAP
         * bit 0 : MIPI_DSI_CLOCK_LANE_NP_SWAP
         */

        dsi,swap = <0x0>;//8、可选，实现PN反转的功能

        /*
         * 0: E_SRC_SEL_FXDE = 0
         * 1: E_SRC_SEL_4KDE
         * 2: E_SRC_SEL_HDMI_RX
         */
        src-sel = <CONFIG_DE4K_OUTPUT_SUPPORT>;
        clock-frequency = <0>;  //5、输出时钟，大于0有效；等于0，默认会自动计算
                                //一般 clock-frequency = h-total-len x v-total-
len x bit x 8 x (frame-rate\1000)HZ / dsi,lanes
                                //当使用RGB565的时候bit=2，或则bit=3
        //status = "okay";//6、设置状态
    };
```

6.5 mipi初始化配置

```
board\hc16xx\common\dts\lcd\lcd_mipi_ILI7807D_1080_1920_rgb888.dtsi

&mipi{//可选
panel-init-sequence = [
    39 00 04 FF 78 07 01
    15 00 02 42 11
    ......
    05 78 01 11
    05 78 01 29
    ];
};

5.1 初始化配置说明：
15 00 02 80 77
 |  |  | |  |
 |  |  | |  数据
 |  |  | 数据
 |  |  数据长度
 |  延时
 命令类型（0x05: 单字节数据 0x15: 双字节数据 0x39: 多字节数据）

 5.2 单字节数据举例：
 05 78 01 29

 5.3 双字节数据举例：
 15 00 02 42 11

 5.4 多字节数据举例：
 39 00 04 FF 78 07 01
```

6.6 重新编译

```
cd hcrtos
hcrtos$ make O=bl kernel-rebuild all; make kernel-rebuild all
```

6.7 注意事项：

- 6.7.1 dts配置注意事项

```
假如board\hc16xx\common\dts\hc16xx-db-c300-v10配置了lvds，可将status = "disable"(这只是关闭lvds的功能，C300芯片可以实现LVDS与MIPI双屏显示，但是需要显示屏的相关屏参是一致才能让显示效果一致)，重新编译
cd hcrtos
hcrtos$ make O=bl kernel-rebuild all; make kernel-rebuild all
```

- 6.7.2 mipi的驱动说明

```
mipi的驱动是以静态库的形式使用的，用户是看不到源代码的

bootloader阶段是放在
components\prebuilts\boot\sysroot\usr\lib\libmipidrv.a
app阶段的是放在
components\prebuilts\sysroot\usr\lib\libmipidrv.a
```

6.8 mipi的接口说明

```
// mipi的接口存放在这个位置
// components\kernel\source\include\uapi\hcuapi\mipi.h

#define MIPI_DSI_SET_ON                 _IO (MIPI_IOCBASE, 0)
#define MIPI_DSI_SET_OFF                _IO (MIPI_IOCBASE, 1)
#define MIPI_DSI_SET_FORMAT             _IO (MIPI_IOCBASE, 2)       //<! enum
MIPI_LCD_COLOR_MODE
#define MIPI_DSI_SET_CFG                _IO (MIPI_IOCBASE, 3)       //<! u8
config: [0, 3f]
#define MIPI_DSI_INIT                   _IO (MIPI_IOCBASE, 4)

#define MIPI_DSI_DCS_READ               _IOWR (MIPI_IOCBASE, 5, struct
hc_dcs_read)
#define MIPI_DSI_DCS_WRITE              _IOW (MIPI_IOCBASE, 6, struct
hc_dcs_write)
#define MIPI_DSI_SEND_DCS_CMDS              _IOW (MIPI_IOCBASE, 7, struct
hc_dcs_cmds)
#define MIPI_DSI_SEND_DCS_INIT_SEQUENCE         _IO (MIPI_IOCBASE, 8)       //<!
the init sequence is set in DTS
#define MIPI_DSI_TIMING_INIT                _IO (MIPI_IOCBASE, 9)
#define MIPI_DSI_SET_TIMING             _IOW (MIPI_IOCBASE, 10, struct
mipi_display_timing)
#define MIPI_DSI_GET_TIMING             _IOR (MIPI_IOCBASE, 11, struct
mipi_display_timing)
#define MIPI_DSI_GPIO_ENABLE                _IO (MIPI_IOCBASE, 12)      //!<
param: value: 0:disabled 1:enable
```

```
#define MIPI_DSI_SET_DCS_INIT_SEQUENCE          _IOW (MIPI_IOCBASE, 13, struct
hc_dcs_cmds)     //<! change init-sequence
```

6.8.1 MIPI_DSI_SET_ON -- 打开mipi的数据通道

参数	描述
无	——
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
ioctl(fd,MIPI_DSI_SET_ON,NULL);
close(fd);
```

6.8.2 MIPI_DSI_SET_OFF -- 关闭mipi的数据通道

参数	描述
无	——
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
ioctl(fd, MIPI_DSI_SET_OFF, NULL);
close(fd);
return 0;
```

6.8.3 MIPI_DSI_SET_FORMAT -- 设置mipi的RGB的格式

参数	描述
val	typedef enum MIPI_LCD_COLOR_MODE
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
    ioctl(fd, MIPI_DSI_SET_FORMAT, MIPI_COLOR_RGB565_0);
}
close(fd);
```

6.8.4 MIPI_DSI_SET_CFG -- 设置cfg

参数	描述
config	范围值为[0, 3f]
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
    ioctl(fd, MIPI_DSI_SET_CFG, 0x1c);
}
close(fd);
```

6.8.5 MIPI_DSI_INIT -- 执行完整的mipi 初始化流程

参数	描述
无	——
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
ioctl(fd, MIPI_DSI_INIT, NULL);
close(fd);
```

6.8.6 MIPI_DSI_DCS_READ -- 读取mipi的数据

参数	描述
rd_data	struct hc_dcs_read
返回	——
rd_data.received_data	返回读取到的数据

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
int ret =0;
struct hc_dcs_read rd_data = { 0 };
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
rd_data.type = 0x14;
rd_data.delay_ms = 0x00;
rd_data.received_size = 0x02;
rd_data.command[0] = 0x04;
rd_data.command[1] = 0x02;
ret = ioctl(fd, MIPI_DSI_DCS_READ, &rd_data);
if (ret == 0) {
    for (int i = 0; i < 3; i++)
        printf("mipi_rd data[%d]=%d\n", i,rd_data.received_data[i]);
}
ioctl(fd, MIPI_DSI_SET_ON, NULL);
close(fd);
```

6.8.7 MIPI_DSI_SEND_DCS_CMDS -- 发送mipi的数据

参数	描述
wd_data	struct hc_dcs_write
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
struct hc_dcs_write wd_data = { 0 };
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
wd_data.type=0x37;
wd_data.delay_ms=0x01;
wd_data.len=2;
wd_data.payload[0]=1;
wd_data.payload[1]=0;
ioctl(fd,MIPI_DSI_DCS_WRITE,&wd_data);
close(fd);
```

6.8.8 MIPI_DSI_SEND_DCS_CMDS -- 按格式发送mipi的数据

参数	描述
cmds	struct hc_dcs_cmds
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
struct hc_dcs_cmds cmds = {4, {0x05, 0x01, 0x01, 0x29}};
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
ioctl(fd,MIPI_DSI_SEND_DCS_CMDS,arg);
close(fd);
```

6.8.9 MIPI_DSI_SEND_DCS_INIT_SEQUENCE -- 发送dts上的初始化程序

参数	描述
无	——
返回	——
无	——

示例代码

```
// dts上的设置
&mipi{
panel-init-sequence = [
    05 78 01 11
    05 78 01 29
```

```
    ];
};

#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH,fd);
    return -1;
}
ioctl(fd, MIPI_DSI_SEND_DCS_INIT_SEQUENCE, NULL);//发送panel-init-sequence
close(fd);
```

6.8.10 MIPI_DSI_TIMING_INIT -- 生效mipi的timing，对应对应执行相关的初始化和配置

参数	描述
无	——
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
    ioctl(fd, MIPI_DSI_TIMING_INIT, NULL);

close(fd);
```

6.8.11 MIPI_DSI_SET_TIMING-- 设置mipi的timing

参数	描述
timing	struct mipi_display_timing
返回	——
无	——

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
struct mipi_display_timing timing={0};
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
```

```
ioctl(fd,MIPI_DSI_GET_TIMING,&timing);//获取mipi的timing
timing.h_sync_level = 0;
ioctl(fd,MIPI_DSI_SET_TIMING,&timing);// 设置mipi的timing
ioctl(fd, MIPI_DSI_TIMING_INIT, NULL);// init后才生效，也可使用 MIPI_DSI_INIT 接口
close(fd);
```

6.8.12 MIPI_DSI_GET_TIMING -- 设置mipi的timing

参数	描述
无	——
返回	——
timing	struct mipi_display_timing

示例代码

```
#define MIPIDEV_PATH  "/dev/mipi"
int fd = 0;
struct mipi_display_timing timing={0};
fd = open(MIPIDEV_PATH, O_RDWR);
if (fd < 0) {
    printf("open %s failed, ret=%d\n", MIPIDEV_PATH, fd);
    return -1;
}
ioctl(fd,MIPI_DSI_GET_TIMING,&timing);
printf("get timing -C %d -t %d -a %d -f %d -s %d -b %d -T %d -A %d -F %d -S %d -
B %d -c %d -o %d -l %d -m %d -H %d -V %d\n",
       timing.clk_frequency,timing.h_total_len,timing.h_active_len,\
       timing.h_front_len,timing.h_sync_len,timing.h_back_len,\
       timing.v_total_len,timing.v_active_len,timing.v_front_len,\
       timing.v_sync_len,timing.v_back_len,\
       timing.packet_cfg,timing.pclk,timing.lane_num,timing.color_mode,\
       timing.h_sync_level,timing.v_sync_level);

close(fd);
```

6.8.13 MIPI_DSI_GPIO_ENABLE -- 设置mipi 的使能gpio

参数	描述
pinpad	——
返回	——
无	——

示例代码

```
// dts上的设置
dsi0 {
    lcd-enable-gpios-rtos = <PINPAD_R08 GPIO_ACTIVE_HIGH PINPAD_R06
GPIO_ACTIVE_LOW>;
};

int fd;
fd = open("/dev/mipi",O_RDWR);
if(fd)
{
    ioctl(fd, MIPI_DSI_GPIO_ENABLE,0);//PINPAD_R08 电平为低 PINPAD_R06 电平为高
    close(fd);
}
```

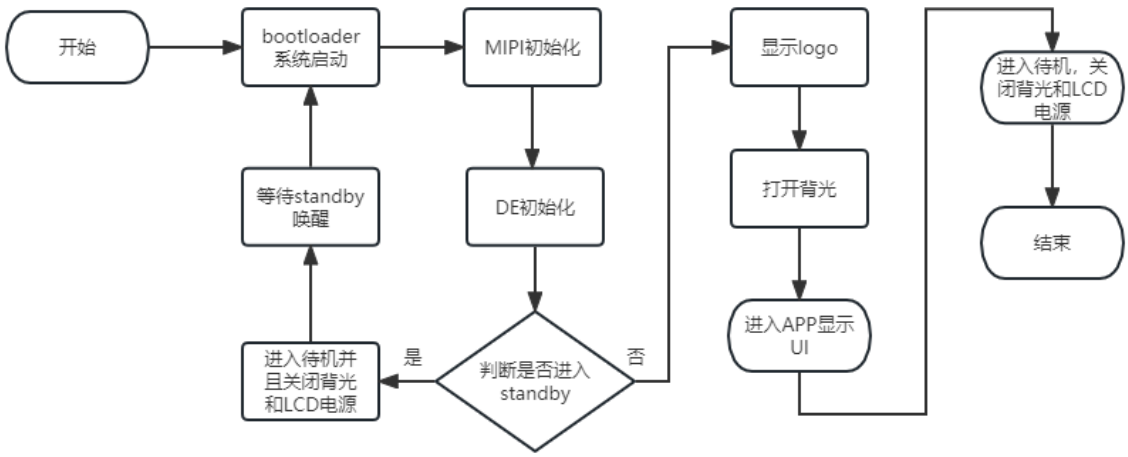
6.8.14 MIPI_DSI_SET_DCS_INIT_SEQUENCE -- 改变mipi init-sequence的内容

参数	描述
struct hc_dcs_cmds	设置init-sequence的内容
返回	——
无	——

示例代码

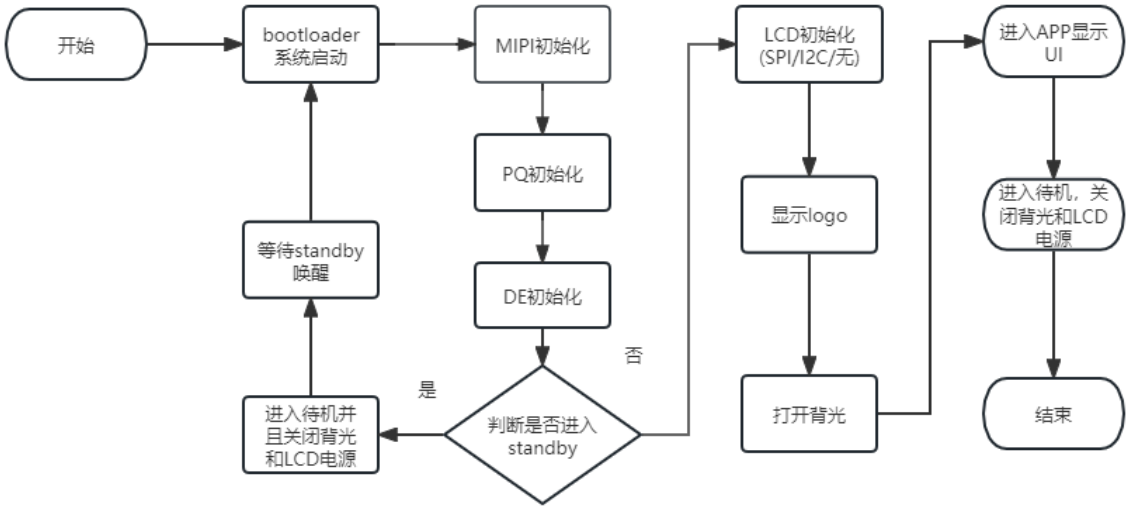
```
struct hc_dcs_cmds mipi_panel_cmds = {0x05,0x00,0x01,29};
int fd;
fd = open("/dev/mipi",O_RDWR);
if(fd)
{
    ioctl(fd, MIPI_DSI_SET_DCS_INIT_SEQUENCE, &mipi_panel_cmds);
    close(fd);
}
```

6.9 mipi的显示流程



6.10 系统打开pq时mipi的流程

系统打开PQ时，上电启动一定要按照 先初始化MIPI -> PQ -> DE 的流程才能让MIPI正常工作



七、HC15XX RGB的配置

7.1 参考某一个板极进行配置

```
cd hcrtos
rm -rf output*
rm -rf bl
make O=bl hichip_hc15xx_db_b200_v10_hcdemo_bl_defconfig
make all
make hichip_hc15xx_db_b200_v10_hcdemo_defconfig
make all
```

7.2 使能pinmix的功能

```
//RGB初始化的文件：components\kernel\source\drivers\rgb\rgb.c
//B100的配置

    rgb {
        pinmux-active = <// 1、使能pinmux的功能，必要
                PINPAD_T01 PINMUX_T01_PRGB_G0
                PINPAD_R08 PINMUX_R08_PRGB_G1
                PINPAD_T00 PINMUX_T00_PRGB_G3
                PINPAD_T03 PINMUX_T03_PRGB_G2
                PINPAD_T02 PINMUX_T02_PRGB_G4
                PINPAD_T04 PINMUX_T04_PRGB_G6
                PINPAD_T05 PINMUX_T05_PRGB_G5
                PINPAD_T06 PINMUX_T06_PRGB_G7
                PINPAD_T07 PINMUX_T07_PRGB_B0
                PINPAD_T08 PINMUX_T08_PRGB_B1
                PINPAD_T09 PINMUX_T09_PRGB_B2
```

```
                PINPAD_T10 PINMUX_T10_PRGB_B3
                PINPAD_T11 PINMUX_T11_PRGB_B4
                PINPAD_T12 PINMUX_T12_PRGB_B5
                PINPAD_T13 PINMUX_T13_PRGB_B6
                PINPAD_T14 PINMUX_T14_PRGB_B7
                PINPAD_R00 PINMUX_R00_PRGB_R0
                PINPAD_R01 PINMUX_R01_PRGB_R1
                PINPAD_R02 PINMUX_R02_PRGB_R2
                PINPAD_R03 PINMUX_R03_PRGB_R3
                PINPAD_R04 PINMUX_R04_PRGB_R4
                PINPAD_R05 PINMUX_R05_PRGB_R5
                PINPAD_R06 PINMUX_R06_PRGB_R6
                PINPAD_R07 PINMUX_R07_PRGB_R7
                PINPAD_T15 PINMUX_T15_PRGB_CLK
                PINPAD_L08 PINMUX_L08_PRGB_HSYN
                PINPAD_L09 PINMUX_L09_PRGB_VSYNC
                PINPAD_L10 PINMUX_L10_PRGB_DE>;
        rgb-clk-inv = <0>;//2、rgb clk是否需要翻转，不设置默认不开启，可选
        vcom-pwmdev = "/dev/pwm0";//3.1 添加pwm vcom 设备，可选
        vcom-frequency = <10000>;//3.2 设置vcom频率，可选
        vcom-duty = <20>;// 3.3 设置vcom占空比 ，可选
        status = "okay";//4、使能，必要
    };

//B200 的配置
    rgb: rgb {
        pinmux-active = <// 1、使能pinmux的功能，必要
            PINPAD_R08 PINMUX_R08_PRGB_R0
            PINPAD_T00 PINMUX_T00_PRGB_R1
            PINPAD_T01 PINMUX_T01_PRGB_R2
            PINPAD_T02 PINMUX_T02_PRGB_R3
            PINPAD_T03 PINMUX_T03_PRGB_R4
            PINPAD_T04 PINMUX_T04_PRGB_R5
            PINPAD_T05 PINMUX_T05_PRGB_R6
            PINPAD_T06 PINMUX_T06_PRGB_R7
            PINPAD_T07 PINMUX_T07_PRGB_G0
            PINPAD_T08 PINMUX_T08_PRGB_G1
            PINPAD_T09 PINMUX_T09_PRGB_G2
            PINPAD_T10 PINMUX_T10_PRGB_G3
            PINPAD_T11 PINMUX_T11_PRGB_G4
            PINPAD_T12 PINMUX_T12_PRGB_G5
            PINPAD_T13 PINMUX_T13_PRGB_G6
            PINPAD_T14 PINMUX_T14_PRGB_G7
            PINPAD_T15 PINMUX_T15_PRGB_B0
            PINPAD_L00 PINMUX_L00_PRGB_B1
            PINPAD_L01 PINMUX_L01_PRGB_B2
            PINPAD_L02 PINMUX_L02_PRGB_B3
            PINPAD_L03 PINMUX_L03_PRGB_B4
            PINPAD_L04 PINMUX_L04_PRGB_B5
            PINPAD_L05 PINMUX_L05_PRGB_B6
            PINPAD_L06 PINMUX_L06_PRGB_B7
            PINPAD_L07 PINMUX_L07_PRGB_CLK
            PINPAD_L08 PINMUX_L08_PRGB_HSYN
            PINPAD_L09 PINMUX_L09_PRGB_VSYNC
            PINPAD_L10 PINMUX_L10_PRGB_DE>;
        rgb-clk-inv = <0>;//2、rgb clk是否需要翻转，不设置默认不开启，可选
        vcom-pwmdev = "/dev/pwm0";// 3.1 添加pwm vcom 设备，可选
        vcom-frequency = <10000>;// 3.2 设置vcom频率，可选
```

```
        vcom-duty = <20>;// 3.3 设置vcom占空比 ，可选
        status = "okay";//4、使能，必要
    };
```

7.3 配置DE

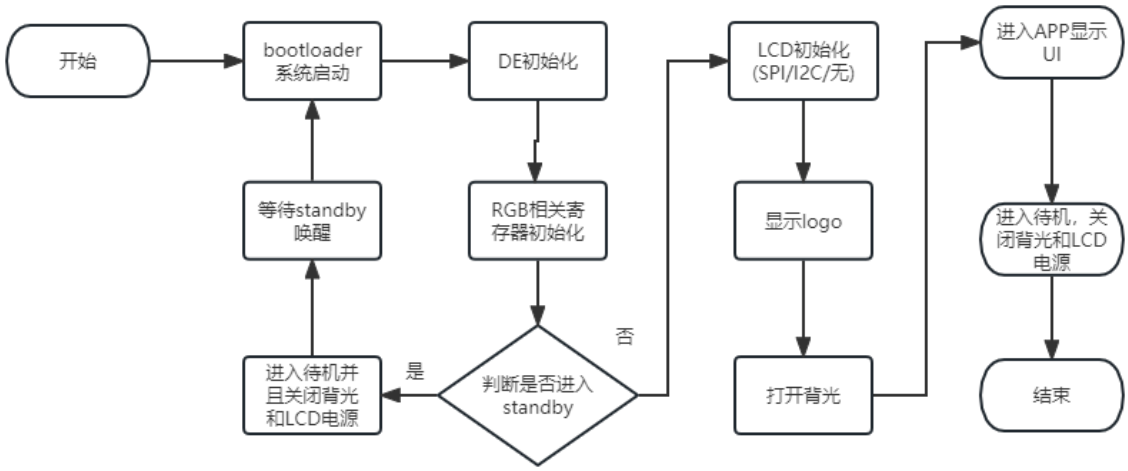
```
&DE {
    tvtype = <15>;//1.一般配置为15，此模式下才能进行配参
    VPInitInfo {
        rgb-cfg{
            b-rgb-enable = <1>;//2、使能RGB设置为 1
            /*
             * 0: MODE_PRGB
             * 1: MODE_SRGB
             */
            rgb-mode = <0>;//3、设置为并行模式
            /*
             * 0: MODE_PRGB_888_10bit
             * 1: MODE_PRGB_666
             */
            prgb-mode = <0>; //4、设置为RGB888的模式
            lcd-width = <800>;//5、lcd-width与h-active-len需要一致
            timing-para {
                /* bool type, 0 or 1*/
                b-enable = <1>;//6、使能为，使能之后才能配置以下的参数
                /*
                 * VPO_RGB_CLOCK_27M = 1,
                 * VPO_RGB_CLOCK_33M = 2,
                 * VPO_RGB_CLOCK_49M = 3,
                 * VPO_RGB_CLOCK_66M = 4,
                 * VPO_RGB_CLOCK_74M = 5,
                 * VPO_RGB_CLOCK_85M = 6,
                 * VPO_RGB_CLOCK_108M = 7,
                 * VPO_RGB_CLOCK_6_6M = 8,
                 * VPO_RGB_CLOCK_9M = 9,
                 * VPO_RGB_CLOCK_39_6M = 10,
                 * VPO_RGB_CLOCK_74_25M = 11,
                 * VPO_RGB_CLOCK_148_5M = 12,
                */
                output-clock = <2>;//7、output-clock = h-total-len * v-total-len
* 频率（60HZ）
                                                8、按照屏厂提供参数填写相关屏参
                h-total-len = <1026>;//h-total-len = h-active-len + h-front-len
+ h-sync-len+h-back-len
                v-total-len = <525>;//v-total-len = v-active-len + v-front-len +
v-sync-len+v-back-len

                h-active-len = <800>;
                v-active-len = <480>;

                h-front-len = <210>;
                h-sync-len = <6>;
                h-back-len = <10>;

                v-front-len = <22>;
                v-sync-len = <3>;
                v-back-len = <20>;
```

```
                /* bool type, 0 or 1*/
                h-sync-level = <0>;
                /* bool type, 0 or 1*/
                v-sync-level = <0>;
                frame-rate = <60000>;//9、设置帧率 60HZ
            };
        };
    };
};
```

7.4 编译

```
cd hcrtos
hcrtos$ make O=bl kernel-rebuild all; make kernel-rebuild all
```

7.5 RGB的显示流程



八、配置LCD背光说明

8.1 dts设置

```
    pwm@2 {
        pinmux-active = <PINPAD_R05 1>;
        devpath = "/dev/pwm2";
        status = "okay";
    };

    backlight {
        backlight-frequency = <1000000>;            //设置背光的频率，频率尽量设置在
411~270KHZ之间
        backlight-pwm-polarity = <0>;               //设置PWM的极性，0: 占空比为0，
PWM输出为0，依次递增；1：占空比为0，PWM输出3.3V，依次递减
        backlight-pwmdev = "/dev/pwm2";             //选择PWM的设备
        brightness-levels = <0 4 8 16 32 64 128 255>;//背光按照等分进行划分，255表示
pwm设置为最高，如果节点不存在，则默认设置成100级别
        default-brightness-level = <7>;             //PWM背光默认值，不设置默认时最高
        backlight-gpios-rtos = <PINPAD_B02 GPIO_ACTIVE_HIGH PINPAD_B03
GPIO_ACTIVE_HIGH>;//设置对应的GPIO背光高电平有效，最多支持5个
```

```
        default-off;                                    //默认不打开，等待外部调用
        status = "okay";
    };
```

8.2 查看menuconfig是否配置了背光

```
2.1 进行配置bootloader的背光
cd hcrtos
make O-bl menuconfig
There is no help available for this option.
  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
[*] lcd driver  --->（必选）

[*]   backlight support（必选）

2.2 进行配置主程序的背光
cd hcrtos
make menuconfig
There is no help available for this option.
  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
[*] lcd driver  --->（必选）

[*]   backlight support（必选）

程序位置
components\kernel\source\drivers\lcd\backlight.c
```

8.3 重新编译

```
make O=bl kernel-rebuild all
make kernel-rebuild all
```

8.4 使用backlight设备的使用说明

```
// 打开背光的位置
// components\applications\apps-bootloader\source\cmd\backlight.c
    int fd = 0;
    struct backlight_info info = {0};
    fd = open("/dev/backlight", O_RDWR);
    if( fd < 0){
        printf("open %s failed, ret=%d\n", "/dev/backlight",fd);
        return -1;
    }
    ioctl(fd,BACKLIGHT_GET_INFO,&info);//获得背光的信息
```

```
    info.brightness_value = 100;
    ioctl(fd,BACKLIGHT_SET_INFO,&info);//修改背光的信息
    ioctl(fd,BACKLIGHT_START);//执行这行之后才会生效
    close(fd);
```

8.5 backlight的接口功能说明

```
//背光接口存放的目录
//components\kernel\source\include\uapi\hcuapi\backlight.h

#include <hcuapi/iocbase.h>

#define BACKLIGHT_GET_INFO          _IOR (BACKLIGHT_IOCBASE, 1, struct
backlight_info)
#define BACKLIGHT_SET_INFO          _IOW (BACKLIGHT_IOCBASE, 2, struct
backlight_info)
#define BACKLIGHT_START             _IO (BACKLIGHT_IOCBASE, 3)
#define BACKLIGHT_STOP              _IO (BACKLIGHT_IOCBASE, 4)

#define BACKLIGHT_LEVEL_SIZE            (40)

struct backlight_info {
    unsigned int levels[BACKLIGHT_LEVEL_SIZE];  //获得背光的级别，信息从DTS上获取，不
可修改
    unsigned int levels_count;                  //获得背光的数量，信息从DTS上获取，不
可修改
    unsigned int pwm_frequency;                 //设置背光的频率，频率尽量设置在
411~270KHZ之间，可动态修改
    unsigned int pwm_polarity;                  //设置PWM的极性，可动态修改
    unsigned int default_brightness_level;      //获得背光的默认亮度，信息从DTS上获
取，不可修改
    unsigned int brightness_value;              //设置背光亮度，可动态修改
};
```

8.5.1 BACKLIGHT_GET_INFO -- 获得背光的信息

参数	描述
backlight_info	背光的信息
返回	——
无	——

示例代码

```
int fd = 0;
struct backlight_info info = {0};
fd = open("/dev/backlight", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/backlight",fd);
    return -1;
}
ioctl(fd,BACKLIGHT_GET_INFO,&info);//获得背光的信息
close(fd);
```

8.5.1 BACKLIGHT_SET_INFO-- 设备背光

参数	描述
backlight_info	背光的信息
返回	——
无	——

示例代码

```
int fd = 0;
struct backlight_info info = {0};
fd = open("/dev/backlight", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/backlight",fd);
    return -1;
}
ioctl(fd,BACKLIGHT_GET_INFO,&info);//获得背光的信息
info.brightness_value = 5;
ioctl(fd,BACKLIGHT_SET_INFO,&info);//修改背光的信息
ioctl(fd,BACKLIGHT_START);//执行这行之后才会生效
close(fd);
```

8.5.1 BACKLIGHT_START-- 启动背光功能

参数	描述
无	——
返回	——
无	——

示例代码

```
int fd = 0;
struct backlight_info info = {0};
fd = open("/dev/backlight", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/backlight",fd);
    return -1;
}
ioctl(fd,BACKLIGHT_GET_INFO,&info);//获得背光的信息
info.brightness_value = 5;
ioctl(fd,BACKLIGHT_SET_INFO,&info);//修改背光的信息
ioctl(fd,BACKLIGHT_START);//执行这行之后才会生效
close(fd);
```

8.5.1 BACKLIGHT_STOP-- 停止背光功能

参数	描述
无	——
返回	——
无	——

示例代码

```
int fd = 0;
fd = open("/dev/backlight", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/backlight",fd);
    return -1;
}
ioctl(fd,BACKLIGHT_STOP);//停止背光功能
close(fd);
```

九、LCD显示屏驱动和lcddev 设备说明

9.1 dts上的配置

```
lcd-jd9168{
    spi-gpio-sck    = <PINPAD_T01>;\\1.客户板子进行自定义lcd驱动需要用到的spi
    spi-gpio-mosi   = <PINPAD_T02>;
    spi-gpio-cs     = <PINPAD_T00>;
    spi-gpio-stbyb  = <PINPAD_T03>;
    default-off;//2.客户自定义，默认关闭，等待外部调用
    status = "okay";
};

lcd{
    lcd-map-name = "lcd-jd9168";\\1.进行驱动匹配，如果匹配成功，lcd将会使用lcd-jd9168提
供的接口进行初始化，
```

```
                                                这个设备主要时为了屏蔽standby之前调用了一次初始
化，根据实际的需要进行设置;
    default-off;//2.默认关闭，等待外部调用
    power-gpios-rtos = <PINPAD_T03 GPIO_ACTIVE_HIGH>;//3.设置gpio电源，高有效
    vcom-pwmdev = "/dev/pwm0";  //4.1 设置vcmo 设备
    vcom-frequency = <10000>;   //4.2 设置pwm 频率
    vcom-duty = <20>;           //4.3 设置pwm占空比
    vcom-polar = <1>;           //4.4 PWM 的极性
    lcd-reset-gpios-rtos = <PINPAD_T03 GPIO_ACTIVE_HIGH>;// 5 LCD复位脚，高有效
    status = "okay";
};
```

9.2 查看menuconfig的配置

```
1、查看bootloader的lcd设备
cd hcrtos
make O-bl menuconfig
There is no help available for this option.

  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
[*] lcd driver  --->（必选）

[*]   lcd dev （必选）
[*]   lcd jd9168 support（必选）

2、查看app 设置lcd设备
cd hcrtos
make menuconfig
There is no help available for this option.

  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
[*] lcd driver  --->（必选）

[*]   lcd dev （必选）
[*]   lcd jd9168 support（必选）


程序位置
components\kernel\source\drivers\lcd\
```

9.3 重新编译

```
make O=bl kernel-rebuild all
make kernel-rebuild all
```

9.4 配置lcd驱动说明

```
// 程序位置components\kernel\source\drivers\lcd\display\jd9168.c
static struct lcd_map_list jd9168_map = {
    .map = {
        .lcd_init = jd9168_display_init,    // 1. /dev/lcddev会调用lcd初始化函数（必要）
        .name = "lcd-jd9168",   // 2.如果与lcd-map-name = "lcd-jd9168"一致，那么/dev/lcddev就会匹配这个设备，并且调用相关的函数（必要）
    }
};

static int jd9168_probe(const char *node)
{
    int np = fdt_node_probe_by_path(node);

    if(np < 0){
        goto error;
    }

    memset(&jd9168dev,0,sizeof(struct jd9168_dev));

    jd9168dev.spi_clk_num = PINPAD_INVALID;//lcddrv->gpio_spi_config.sck;
    jd9168dev.spi_clk_vaild_edge=1;
    jd9168dev.spi_cs_polar=0;
    jd9168dev.spi_is_9bit=1;
    jd9168dev.spi_cs_num = PINPAD_INVALID;//lcddrv->gpio_spi_config.cs;
    jd9168dev.spi_mosi_num = PINPAD_INVALID;//lcddrv->gpio_spi_config.mosi;
    jd9168dev.spi_miso_num = PINPAD_INVALID;//lcddrv->gpio_spi_config.miso;
    jd9168dev.lcd_stbyb_num = PINPAD_INVALID;//lcddrv->gpio_spi_config.stbyb;
    jd9168dev.lcd_stbyb_polar = 0;

    //3.获取spi的信息
    fdt_get_property_u_32_index(np, "reset",            0, &jd9168dev.lcd_reset_num);
    fdt_get_property_u_32_index(np, "spi-gpio-sck",     0, &jd9168dev.spi_clk_num);
    fdt_get_property_u_32_index(np, "spi-gpio-mosi",    0, &jd9168dev.spi_mosi_num);
    fdt_get_property_u_32_index(np, "spi-gpio-miso",    0, &jd9168dev.spi_miso_num);
    fdt_get_property_u_32_index(np, "spi-gpio-cs",      0, &jd9168dev.spi_cs_num);
    fdt_get_property_u_32_index(np, "spi-gpio-stbyb",   0, &jd9168dev.lcd_stbyb_num);
    log_d("jd9168dev.lcd_stbyb_num = %d %d %d %d\n",jd9168dev.spi_clk_num,jd9168dev.spi_mosi_num,jd9168dev.spi_cs_num,jd9168dev.lcd_stbyb_num);

    int default_off = 0;
    fdt_get_property_u_32_index(np, "default-off",  0, &default_off);// 4.如果为default_off ==1，那么说明用户不想要在这里执行初始化，/dev/lcddev打开设备u之后，会执行lcd的初始化
    if(default_off ==0)
        jd9168_display_init();
```

```
    jd9168_map.map.default_off_val = default_off;// 5.如果为default_off ==0，那么说明已经lcd 执行了初始化，/dev/lcddev下一次不会再执行lcd初始化
    lcd_map_register(&jd9168_map);//6.初始化lcd map
error:
    return 0;
}

static int jd9168_init(void)
{
    jd9168_probe("/hcrtos/lcd-jd9168");
    return 0;
}

module_driver(jd9168, jd9168_init, NULL, 2)
```

9.5 bootloader执行lcd初始化说明

```
// 1.程序位置
// components\applications\apps-bootloader\source\cmd\boot_lcd.c

// 2、程序说明
int open_boot_lcd_init(int argc, char *argv[])
{
    int fd;
    int tmp = 0;

    fd = open("/dev/lcddev",O_RDWR);
    if(fd > 0)
    {
        /*get lcd init status*/
        ioctl(fd, LCD_GET_INIT_STATUS, &tmp);//1. 会读取LCD是否已经进行初始化
        if(tmp == LCD_NOT_INITED)//2. 如果LCD已经初始化，就不会执行下面操作
        {
            ioctl(fd, LCD_INIT);//3. 进行lcd的初始化
#ifdef CONFIG_BOOT_LCD_ROTATE
            uint8_t flip_flag = 0;
            if(sys_get_sysdata_flip_mode(&flip_flag)!=0)//4、使用LCD旋转的方式
            {
                flip_flag=0;
            }
            ioctl(fd, LCD_SET_ROTATE, flip_flag);//5、进行LCD旋转
#endif
        }
        close(fd);
    }

    return 0;
}
```

9.6 univdev通用的lcd初始化设备

```
/*
为了方便用户进行快速配置屏幕初始化，可以使用univdev，可以通过dts上的配置快速发送i2c和spi的初始化序列
*/
//1、 模式 dts的配置说明
//spi的
    spi-gpio {
        pinmux-active = <PINPAD_T15 0 PINPAD_T16 0 PINPAD_T17 0 PINPAD_T18 0>;//与spi的pin保持一致，对应是spi flash的才需要

        gpio-sck = <PINPAD_T18>;
        gpio-mosi = <PINPAD_T17>;
        gpio-miso = <PINPAD_T16>;
        num-chipselects = <1>;
        cs-gpios = <PINPAD_L15>;
        status = "okay";
        spidev@2 {
            devpath = "/dev/spidev2";
            cs_gpio = <PINPAD_B08>;
            spi-max-frequency = <50000000>;
            status = "okay";
            mutex-lock = "sf_lock";//配置flash的pin才需要
        };
    };

    univdev: lcd-univdev {
        spi-devpath = "/dev/spidev2";//必要，指定送序列的设备
        reset   = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        power = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        stbyb = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        pinpad0 = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        pinpad1 = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        default-off;//可选
        spi-bit = <9>;//可选，spi的命令格式，不设置默认为8
        rotate-sequence = [//可选，设置旋转的序列，格式：旋转角度 指令 数据长度 数据
            00 01 02 36 00
            00 02 02 00 01
            01 01 02 36 00
            01 02 02 10 01
        ];

        lcd-init-sequence = [//可选，设置屏幕初始化的序列，格式：指令 数据长度 数据
            01 02 11 00
            03 01 78
            01 02 29 00
        ];
        status = "okay";//必要
    };

    lcd{
        lcd-map-name = "lcd-univdev";//必要
        default-off;//可选
        status = "okay";
    };
```

```
// i2c模式配置

    gpio-i2c@0 {//i2c的配置
        sda-pinmux = <PINPAD_INVALID>;
        scl-pinmux = <PINPAD_INVALID>;
        status = "okay";
        simulate;
    };

    univdev: lcd-univdev {
        i2c-devpath = "/dev/gpio-i2c0";//必要，指定送序列的设备
        reset   = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        power = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        stbyb = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        pinpad0 = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        pinpad1 = <PINPAD_INVALID>;//可选，配合指令设置pin的状态
        rotate-sequence = [//可选，设置旋转的序列，格式：旋转角度 指令 数据长度 i2c地址 数
据
            00 01 03 ea 00 b1
            01 01 03 e0 0f 00
        ];
        lcd-init-sequence = [//可选，设置屏幕初始化的序列，格式：指令 数据长度 i2c地址 数
据
            01 03 ea 00 b1
            01 03 e0 0f 00
        ];
        default-off;
        status = "okay";
    };

    lcd{
        lcd-map-name = "lcd-univdev";//必要
        default-off;//可选
        status = "okay";
    };
```

2、关于序列指令说明

LCD的节点主要是使用到了lcd-univdev，一个通用的I2C和SPI的初始化节点，用户可以根据SPI或者I2C实际发送出来的命令进行解析，之后配置在lcd-univdev的初始化序列上(lcd-init-sequence)上，就能完成发送初始化命令。

spi程序上的规则，

```
01 02 29 00
 |  |  | |
 |  |  | 数据
 |  |  数据
 |  数据长度
```

命令类型（0x01：发送的命令，0x02：发送数据格式， 0x03：延时，0x04：设置POWER引脚的电平，0x05：设置RESET的电平，0x06：设置STBYB的电平，0x07：设置PINPAD0的电平，0x08：设置PINPAD1的电平）

i2c程序上的规则，

```
01 02 29 00
 |  |  | |
 |  |  | 数据
 |  |  地址
 |  数据长度
```

```
 命令类型（0x01：发送的命令，0x02：发送数据格式， 0x03：延时，0x04：设置POWER引脚的电平，0x05：设置RESET的电平，0x06：设置STBYB的电平，0x07：设置PINPAD0的电平，0x08：设置PINPAD1的电平）

3、打开对应的驱动
一般只需要在bootloader阶段去初始化
cd hcrots
make O=bl menuconfig

CONFIG_DRV_LCD
CONFIG_DRV_LCD_DEV
Symbol: CONFIG_DRV_LCD_DEV [=y]

  x Type  : bool
  x Prompt: lcd dev
  x   Location:
  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
  x           -> lcd driver (CONFIG_DRV_LCD [=y])

CONFIG_DRV_UNIVDEV//对应会勾选SPI相关的驱动

 Symbol: CONFIG_DRV_UNIVDEV [=y]
  x Type  : bool
  x Prompt: lcd univdev
  x   Location:
  x     -> Components
  x       -> kernel (BR2_PACKAGE_KERNEL [=y])
  x         -> Drivers
  x           -> lcd driver (CONFIG_DRV_LCD [=y])
[*]   lcd dev
[*]   lcd univdev

4、重新编译
cd hcrots
make O=output-bl kernel-rebuild all //bootloader
make all //主程序
```

9.7 LCD的接口说明

```
#include <hcuapi/iocbase.h>

#define LCD_INIT                    _IO  (LCD_IOCBASE, 1)
#define LCD_GET_INIT_STATUS             _IOR (LCD_IOCBASE, 2, enum
LCD_INIT_STATUS)
#define LCD_SEND_INIT_SEQUENCE              _IO  (LCD_IOCBASE, 3)   //<! the
init sequence is set in DTS
#define LCD_SET_INIT_SEQUENCE               _IOW (LCD_IOCBASE, 4, struct
hc_lcd_init_sequence)  //!< Change init sequence
#define LCD_SET_ROTATE                  _IO  (LCD_IOCBASE, 5)   //!< param:
lcd_rotate_type_e
#define LCD_SEND_DATA                   _IOW (LCD_IOCBASE, 5, struct
hc_lcd_write)
```

```
#define LCD_SEND_CMDS                   _IOW (LCD_IOCBASE, 7, struct
hc_lcd_write)
#define LCD_READ_DATA                   _IOWR(LCD_IOCBASE, 8, struct
hc_lcd_read)
#define LCD_SET_ONOFF                   _IO  (LCD_IOCBASE, 9)   //!< param: On:
1, Off: 0
#define LCD_SET_PWM_VCOM                _IO  (LCD_IOCBASE, 10)  //!< param: Duty
cycle: 0~100
#define LCD_GET_PWM_VCOM                _IOR (LCD_IOCBASE, 10, int)
#define LCD_SET_POWER_GPIO              _IO  (LCD_IOCBASE, 11)  //!< param: 0 or
1 as Power-GPIO value
#define LCD_SET_RESET_GPIO              _IO  (LCD_IOCBASE, 12)  //!< param: 0 or
1 as Reset-GPIO value
#define LCD_GET_MODE_INFO               _IOR (LCD_IOCBASE, 13, struct
hc_lcd_mode_info)
#define LCD_SET_MODE_INFO               _IOW (LCD_IOCBASE, 13, struct
hc_lcd_mode_info)
#define LCD_SET_AREA                    _IOW (LCD_IOCBASE, 14, struct
hc_lcd_area)
#define LCD_DRAW_POINT                  _IOW (LCD_IOCBASE, 15, struct
hc_lcd_drawpoint)

#define LCD_MAX_DATA_SIZE               32
#define LCD_MAX_COMMAND_SIZE                32
#define LCD_MAX_RECEIVED_SIZE               32
#define LCD_MAX_PACKET_SIZE                 4032

typedef enum LCD_ROTATE_TYPE {
    LCD_ROTATE_0 = 0,
    LCD_ROTATE_180,
    LCD_H_MIRROR,
    LCD_V_MIRROR,
} lcd_rotate_type_e;

typedef enum LCD_INIT_STATUS {
    LCD_NOT_INITED,
    LCD_IS_INITED,
} lcd_init_status_e;

typedef enum LCD_INIT_SEQUENCE_FLAG {
    LCD_INIT_SEQUENCE_DEFAULT = 0x0,
    LCD_INIT_SEQUENCE_SPI = 0x01,
    LCD_INIT_SEQUENCE_I2C = 0x02,
} lcd_init_sequence_flag_e;

struct hc_lcd_write {
    unsigned int count;
    unsigned int packet[LCD_MAX_DATA_SIZE];
};

struct hc_lcd_read {
    unsigned int command_size;
    unsigned int command_data[LCD_MAX_COMMAND_SIZE];
    unsigned int received_size;
    unsigned int received_data[LCD_MAX_RECEIVED_SIZE];
};

struct hc_lcd_spi_info {
```

```
    unsigned int mosi;
    unsigned int miso;
    unsigned int sck;
    unsigned int cs;
    unsigned int mode;
    unsigned int bit;
    unsigned int orcmds;
    unsigned int ordata;
};

struct hc_lcd_i2c_info {
    unsigned int sda;
    unsigned int scl;
    unsigned int addr;
    unsigned int orcmds;
    unsigned int ordata;
};

struct hc_lcd_pinpad_info {
    unsigned int power;
    unsigned int reset;
    unsigned int stbyb;
    unsigned int pinpad0;
    unsigned int pinpad1;
};

struct hc_lcd_init_sequence {
    unsigned int count;
    unsigned char packet[LCD_MAX_PACKET_SIZE];
    int flag;
};

struct hc_lcd_mode_info {
    int mode;
    struct hc_lcd_spi_info spi;
    struct hc_lcd_i2c_info i2c;
    struct hc_lcd_pinpad_info pinpad;
};

struct hc_lcd_area {
    int xStart;
    int xEnd;
    int yStart;
    int yEnd;
};

struct hc_lcd_drawpoint {
    int x;
    int y;
    unsigned int color;
};
```

9.6.1 LCD_INIT -- LCD完整流程的初始化

参数	描述
无	——
返回	——
无	——

示例代码

```
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd,LCD_INIT);//完整的初始化显示屏相关的内容
close(fd);

//对应驱动里需要完成int (*lcd_init)(void);的接口实现
```

9.6.2 LCD_GET_INIT_STATUS -- 获得LCD初始化的状态

参数	描述
int	状态值
返回	——
无	——

示例代码

```
int fd = 0;
int val = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_GET_INIT_STATUS, &val);//获得LCD初始化的状态
close(fd);
```

9.6.3 LCD_SEND_INIT_SEQUENCE -- 发送LCD初始化序列

参数	描述
无	——
返回	——
无	——

示例代码

```
//dts上的配置
    spi-gpio {
        pinmux-active = <PINPAD_T15 0 PINPAD_T16 0 PINPAD_T17 0 PINPAD_T18 0>;
        gpio-sck = <PINPAD_T18>;
        gpio-mosi = <PINPAD_T17>;
        gpio-miso = <PINPAD_T16>;
        num-chipselects = <1>;
        cs-gpios = <PINPAD_L15>;
        status = "okay";
        spidev@2 {
            devpath = "/dev/spidev2";
            cs_gpio = <PINPAD_B21>;
            spi-max-frequency = <50000000>;
            status = "okay";
            mutex-lock = "sf_lock";
        };
    };

    univdev: lcd-univdev {
        spi-devpath = "/dev/spidev2";
        default-off;
        #define LCD_UNIVDEV_ENABLED
        status = "okay";
    };

    lcd{
        reset = <PINPAD_B22>;
        lcd-map-name = "lcd-univdev";
        spi-bit = <9>;
        lcd-init-sequence = [ //执行该序列
            05 01 00
            03 01 78
            05 01 01
            03 01 78
            01 02 11 00
        ];
        default-off;
        status = "okay";
    };

int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd,LCD_SEND_INIT_SEQUENCE);//执行之后，会发送lcd-init-sequence序列的信息
close(fd);

//对应驱动里需要完成int (*lcd_init)(void);的接口实现
```

9.6.4 LCD_SET_INIT_SEQUENCE -- 修改dts上的lcd-init-sequence信息

参数	描述
struct hc_lcd_init_sequence	初始化序列的信息，最大4032个
返回	——
无	——

示例代码

```
struct hc_lcd_init_sequence lcd_panel_cmds = {.count = 4, .packet =
{0x01,0x02,0x29,0x00}};
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd,LCD_SET_INIT_SEQUENCE, &lcd_panel_cmds);//执行之后，会修改lcd-init-
sequence序列的信息
close(fd);

//对应驱动里需要完成int (*lcd_set_init_sequence)(struct hc_lcd_init_sequence *info);
的接口实现
```

9.6.5 LCD_SET_ROTATE-- 设置LCD旋转的角度

参数	描述
lcd_rotate_type_e	可以四个方向设置
返回	——
无	——

示例代码

```
lcd_rotate_type_e rotate = 0;
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_SET_ROTATE, rotate);//执行之后，会发送lcd-init-sequence序列的信息
close(fd);

//对应驱动里需要完成int (*lcd_rorate)(lcd_rotate_type_e direct);的接口实现
```

9.6.6 LCD_SEND_DATA -- LCD发送数据

参数	描述
struct hc_lcd_write	发送数据，一次最多32个
返回	——
无	——

示例代码

```
struct hc_lcd_write lcd_data = {.count = 2, .packet = {0x29,0x00}};
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd,LCD_SEND_DATA , &lcd_data);//发送数据
close(fd);


//对应驱动里需要完成int (*lcd_write_data)(u32 *cmd,u32 len);的接口实现
```

9.6.7 LCD_SEND_CMDS-- LCD发送命令

参数	描述
struct hc_lcd_write	发送数据，一次最多32个
返回	——
无	——

示例代码

```
struct hc_lcd_write lcd_data = {.count = 2, .packet = {0x29,0x00}};
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_SEND_CMDS, &lcd_data);//发送命令
close(fd);


//对应驱动里需要完成int (*lcd_write_cmds)(u32 *cmd,u32 len);的接口实现
```

9.6.8 LCD_READ_DATA -- 读取LCD的命令

参数	描述
struct hc_lcd_read	command_data是发送的数据，received_data是接收回来的数据
返回	——
无	——

示例代码

```
struct hc_lcd_read lcd_read_data = {.command_size = 2, .command_data =
{0x01,0x00}};
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_READ_DATA, &lcd_read_data);//发送命令，并读回数据
close(fd);

//对应驱动里需要完成int (*lcd_read_data)(u32 cmd);的接口实现
```

9.6.9 LCD_SET_ONOFF -- 显示开关接口

参数	描述
int	param: On: 1, Off: 0
返回	——
无	——

示例代码

```
int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_SET_ONOFF , 1);//显示画面
close(fd);

//对应驱动里需要完成int (*lcd_onoff)(u32 onoff);的接口实现
```

9.6.10 LCD_SET_PWM_VCOM -- 控制PWM VCOM的占空比

参数	描述
int	//!< param: Duty cycle: 0~100
返回	——
无	——

```
//dts上的设置
lcd{
    vcom-pwmdev = "/dev/pwm0";  //4.1 设置vcmo 设备
    vcom-frequency = <10000>;   //4.2 设置pwm 频率
    vcom-duty = <20>;           //4.3 设置pwm占空比
    vcom-polar = <1>;           //4.4 PWM 的极性
    status = "okay";
};

int fd = 0;
int tmp = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_GET_PWM_VCOM, &tmp);//获得当前VCOM的占空比
tmp = 100;
ioctl(fd, LCD_SET_PWM_VCOM, tmp);//占空比设置成100
close(fd);
```

9.6.11 LCD_GET_PWM_VCOM -- 获得PWM VCOM的占空比

参数	描述
int	//!< param: Duty cycle: 0~100
返回	——
无	——

```
//dts上的设置
lcd{
    vcom-pwmdev = "/dev/pwm0";  //4.1 设置vcmo 设备
    vcom-frequency = <10000>;   //4.2 设置pwm 频率
    vcom-duty = <20>;           //4.3 设置pwm占空比
    vcom-polar = <1>;           //4.4 PWM 的极性
    status = "okay";
};

int fd = 0;
int tmp = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
```

```
}
ioctl(fd, LCD_GET_PWM_VCOM, &tmp);//获得当前VCOM的占空比
tmp = 100;
ioctl(fd, LCD_SET_PWM_VCOM, tmp);//占空比设置成100
close(fd);
```

9.6.12 LCD_SET_POWER_GPIO -- 设置LCD的电源

参数	描述
int	//!< param: 0 or 1 as Power-GPIO value
返回	——
无	——

```
//dts上的设置
lcd{
    power-gpios-rtos = <PINPAD_T03 GPIO_ACTIVE_HIGH PINPAD_T04
GPIO_ACTIVE_LOW>;//3.设置gpio电源，PINPAD_T03高有效，PINPAD_T04低有效
    status = "okay";
};

int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_SET_POWER_GPIO, 1);//将PINPAD_T03 拉高，将PINPAD_T04 拉低
close(fd);
```

9.6.13 LCD_SET_RESET_GPIO -- 设置复位键

参数	描述
int	//!< param: 0 or 1 as Reset-GPIO value
返回	——
无	——

```
//dts上的设置
lcd{
    lcd-reset-gpios-rtos = <PINPAD_T03 GPIO_ACTIVE_LOW>;// 5 LCD复位脚，低电平有效，
启动复位之后会将PINPAD_T03拉高
    status = "okay";
};

int fd = 0;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
```

```
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_SET_POWER_GPIO, 1);//将PINPAD_T03 拉低，复位生效
close(fd);
```

9.6.14 LCD_GET_MODE_INFO -- 获得显示屏当前的配置的信息

参数	描述
struct hc_lcd_mode_info	显示屏当前的配置信息
返回	——
无	——

```
int fd = 0;
struct hc_lcd_mode_info info;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_GET_MODE_INFO , &info);//获得信息
close(fd);

//对应驱动需要实现struct hc_lcd_mode_info* (*lcd_get_mode_info)(void);的接口
```

9.6.15 LCD_SET_MODE_INFO -- 更新显示屏当前的配置的信息

参数	描述
struct hc_lcd_mode_info	显示屏当前的配置信息
返回	——
无	——

```
int fd = 0;
struct hc_lcd_mode_info info;
fd = open("/dev/lcddev", O_RDWR);
if( fd < 0){
    printf("open %s failed, ret=%d\n", "/dev/lcddev",fd);
    return -1;
}
ioctl(fd, LCD_GET_MODE_INFO , &info);//获得信息
info.spi.mode = 3;
ioctl(fd, LCD_SET_MODE_INFO , &info);//更新信息
close(fd);

//对应驱动需要实现int (*lcd_set_mode_info)(struct hc_lcd_mode_info *info);的接口
```