Security Classification:

Most Confidential(　　) Confidential(　　) Internal(　) Public(√)

Rockchip_DSS Development Guide

（Technical Department，MID Group）

Mark：	Current Ver.：	V1.2
[　] Editing	Drafted By：	YXJ,XJH,HHB,ZWL
[√] Issued	Translated By:	ZY,ZXZ,GLN
	Fulfill Date：	2014-06-17
	Checked By：	XXX
	Completed Date：	2014-XX-XX

Note: The document is specific to DSS (Display Subsystem) kernel framework which is made up of display interfaces such as FB, LCDC, LVDS, MIPI, HDMI and so on since platforms RK30XX, RK2928, RK31XX and RK32XX

福州瑞芯微电子有限公司

Fuzhou　Rockchips　Electronics　Co．, Ltd

Revision History

Ver. No	Author	Revised Date	Note for Revision	Remark
V1.0		2013-12-22	Initial released	
V1.1		2014-02-26	Add Q&A	
V1.2		2014-06-18	Support for linux 3.10 version kernel	

Index

1 LCD Hardware Principle

LCD (Liquid Crystal Display) is divided into Static Drive, Simple Matrix Drive and Active Matrix Drive according to the drive mode. Among them, Simple Matrix Drive can be subdivided into TN (Twisted Nematic) and STN (Super Twisted Nematic), while TFT (Thin Film Transistor) is the mainstream of Active Matrix Drive. The table below lists the differences among TN, STN and TFT.

Category	**TN**	**STN**	**TFT**
Theory	Liquid crystal molecule, twist 90 degrees	Twist 180~270 degrees	Liquid crystal molecule, twist 90 degrees
Character	Black and white, monochrome,low contrast	Black and white, color, low contrast	Color, high contrast
Animation Display	No	No	Yes
View Angle	Under 30°	Under 40°	Under 80°
Panel Size	1~3 inches	1~12 inches	37 inches

Table (1) TN, STN, TFT LCD Comparision

It can be seen from Table (1), TFT LCD is better than TN and STN LCD on display quality, and it has high contrast, high reaction speed, rich colors due to manufacturing process. So TFT LCD is widely used in current ARM market.

Not only LCD driver, but also relative LCDC (Liquid Crystal Display Controller) is needed to display image on LCD panel. Usually, LCD driver is made with LCD glass substrate together in form of COG/COG, LCDC is implemented by external circuit. A lot of LCDC are integrated directly inside MCU such as CPU of RK series which can control STN and TFT LCD through LCDC more conveniently.

Figure (1) shows the tipical timing sequence of TFT LCD. In the sequence diagram, VCLK, HSYNC, VSYNC are pixel clock signal (used to latch image data), horizontal synchronization signal, vertical synchronization signal respectively, VDEN is data effective indication signal, VD is data signal of image.

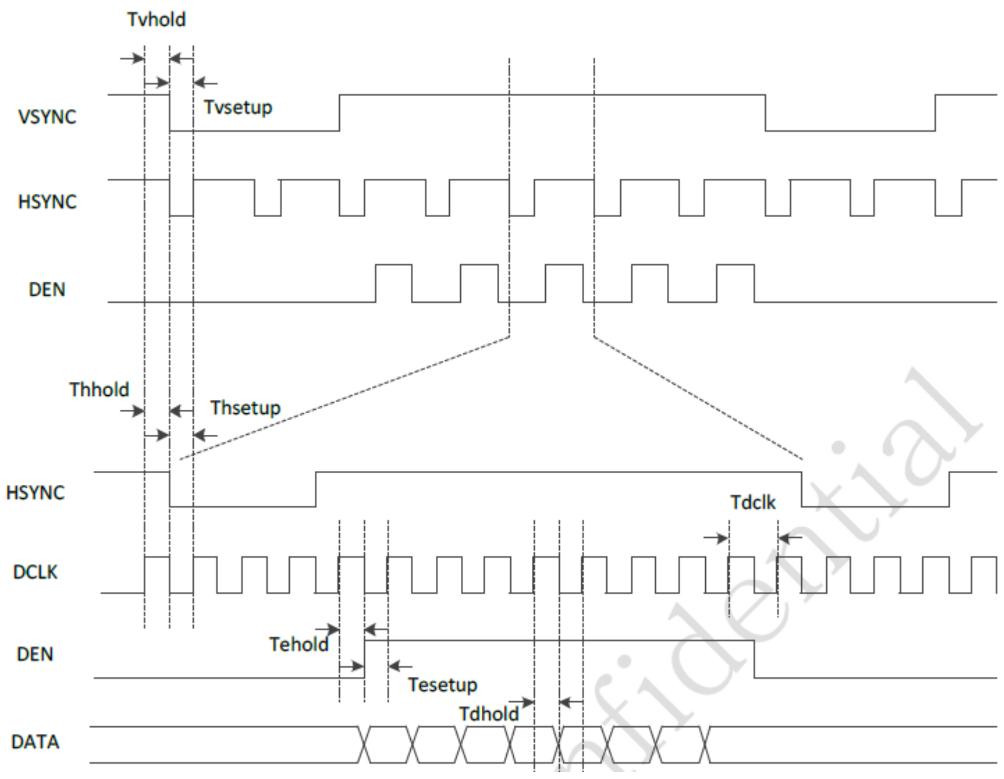


Figure (1) LCD Sequence Diagram of Row&field Control

VSYNC sends a pulse each time, which indicates that the new screen image data start sending. HSYNC sends a pulse each time, which indicates that the new row image data start sending. Flyback time must be reserved at both head and tail of vertical synchronization and horizontal synchronization. The arrangement of sequence stems from the time needed by CRT monitor electronic gun deflection, then becomes the actual industrial standard, therefore TFT LCD also includes flyback time although it is not necessary for TFT.

Figure (2) shows sequence parameters of TFT LCD which should be set up in

LCDC, upper margin (vertical back porch) and lower margin (vertical front porch) is flyback time of vertical switching, left margin (horizontal back porch) and right margin (horizontal front porch) is flyback time of horizontal switching, hsync (horizontal pulse width) and vsync (vertical pulse width) are the time needed by horizontal and vertical synchronization respectively. Xres (horizontal valid data) and yres (vertical valid data), in our MID systems, commom LCD resolutions are 800*480, 1024*600, 1024*768, 1280*800, 1920*1200, 2048*1536 and so on.

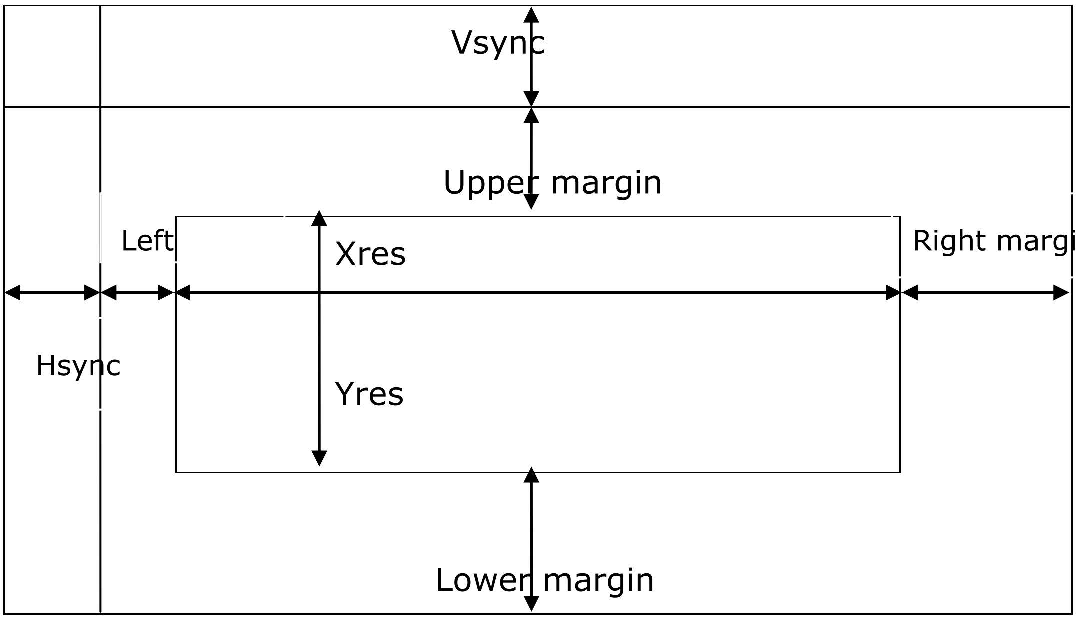
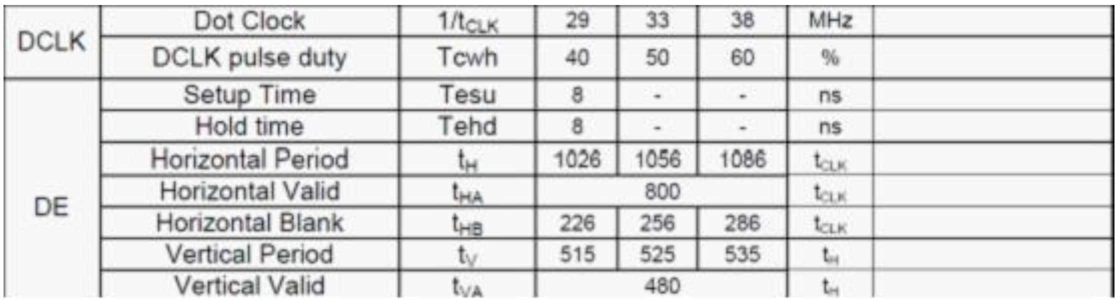
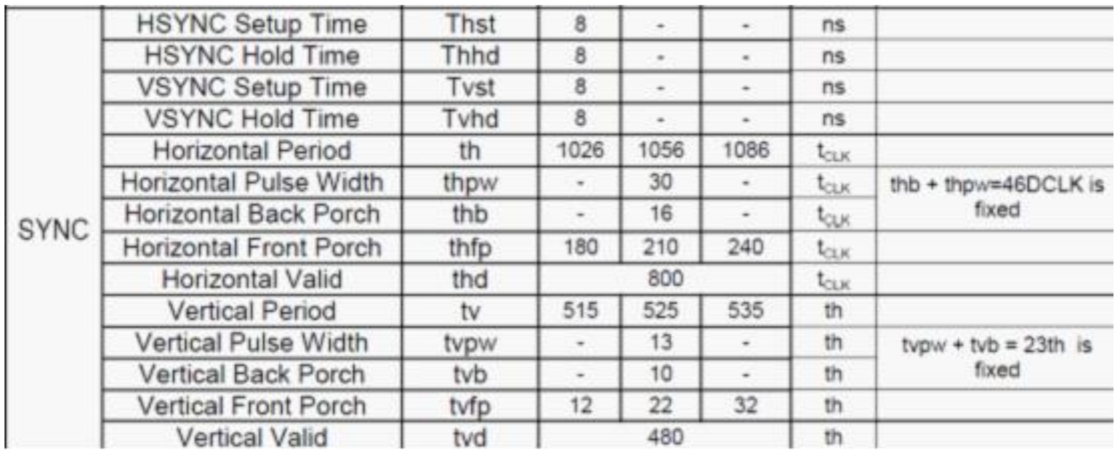


Figure (2) Sequence Parameters of LCD

Figure (3) is a typical timing characteristics list of LCD:



DCLK	Dot Clock	$1/t_{CLK}$	29	33	38	MHz	
	DCLK pulse duty	Tcwh	40	50	60	%	
DE	Setup Time	Tesu	8	-	-	ns	
	Hold time	Tehd	8	-	-	ns	
	Horizontal Period	t_H	1026	1056	1086	t_{CLK}	
	Horizontal Valid	t_{HA}		800		t_{CLK}	
	Horizontal Blank	t_{HB}	226	256	286	t_{CLK}	
	Vertical Period	t_V	515	525	535	t_H	
	Vertical Valid	t_{VA}		480		t_H	



SYNC	HSYNC Setup Time	Thst	8	-	-	ns	
	HSYNC Hold Time	Thhd	8	-	-	ns	
	VSYNC Setup Time	Tvst	8	-	-	ns	
	VSYNC Hold Time	Tvhd	8	-	-	ns	
	Horizontal Period	th	1026	1056	1086	t_{CLK}	
	Horizontal Pulse Width	thpw	-	30	-	t_{CLK}	thb + thpw=46DCLK is fixed
	Horizontal Back Porch	thb	-	16	-	t_{CLK}	
	Horizontal Front Porch	thfp	180	210	240	t_{CLK}	
	Horizontal Valid	thd		800		t_{CLK}	
	Vertical Period	tv	515	525	535	th	
	Vertical Pulse Width	tvpw	-	13	-	th	tvpw + tvb = 23th is fixed
	Vertical Back Porch	tvb	-	10	-	th	
	Vertical Front Porch	tvfp	12	22	32	th	
	Vertical Valid	tvd		480		th	

Figure (3) Timing Characteristics

It can be seen from Figure (3), this kind of LCD can be drived in two modes DE and SYNC, DE mode and SYNC mode are the same in software, SYNC is commomly used. We can obtain the parameters below from Figure (3):

```
Left_margin     = HBP（Horizontal Back Porch）    = 16;
Right_margin    = HFP（Horizontal Front Porch）    = 210;
Hsync           = HPW（Horizontal Pulse Width）  = 30;
Xres            = HVD (Horizontal Valid)          = 800;
Upper_margin    = VBP（Vertical Back Porch）       = 10;
Lower_margin    = VFP（Vertical Front Porch）      = 22;
Vsync           = VPW（Vertical Pulse Width）      = 13;
Yres            = VVD（Vertical Valid）            = 480;
```

And these parameters satisfy the following formulas:

```
Left_margin + right_margin + hsync + xres = horizontal period
Upper_margin + lower_margin + vsync + yres = vertical period
```

All of these parameters should be written into corresponding LCDC registers.

Otherwise, for some LCDs of DE mode, the parameters HFP、HSYNC、HBP、VFP、VSYNC、VBP are not told directly, Horizontal blank time and Vertical blank time are given instead, in this case, we only need to guarantee the following relationships:

```
HFB + HSYNC + HBP = Horizontal blank time
```

VFP + VSYNC + VBP = Vertical blank time

In LCD driver, there is another important parameter--dot clock, in the data sheet of LCD is commonly measured in MHz, named PCLK or DCLK. For example, if the clock is 28.37516 MHz, the time needed to draw a pixel is 35242 ps:

1/(28.37516E6 Hz) = 35.242E-9 s

If the resolution of LCD is 640x480, the time of showing one line is:

640*35.242E-9 s = 22.555E-6 s

Every scanning line is 640 pixels, horizontal flyback and horizontal synchronization also need some time, assume horizontal flyback and synchronization need 272 pixel clock, hence, the total time of drawing a scanning line is:

(640+272)*35.242E-9 s = 32.141E-6 s

It can be calculated that horizontal scan rate is about 31 kHz:

1/(32.141E-6 s) = 31.113E3 Hz

A whole screen has 480 lines, vertical flyback and vertical synchronization also need some time, assume vertical flyback and synchronization need 49 pixel clock, hence, the total time of drawing a whole screen is:

(480+49)*32.141E-6 s = 17.002E-3 s

It can be calculated that vertical scan rate is about 59 kHz:

1/(17.002E-3 s) = 58.815 Hz

It means that the screen data refresh 59 times per second.

Thus the below formula can be obtained:

Refresh rate =dotclock/(（ xres+left_margin+right_margin+hsync ）*(yres+upper_margin+low_margin+vsync))

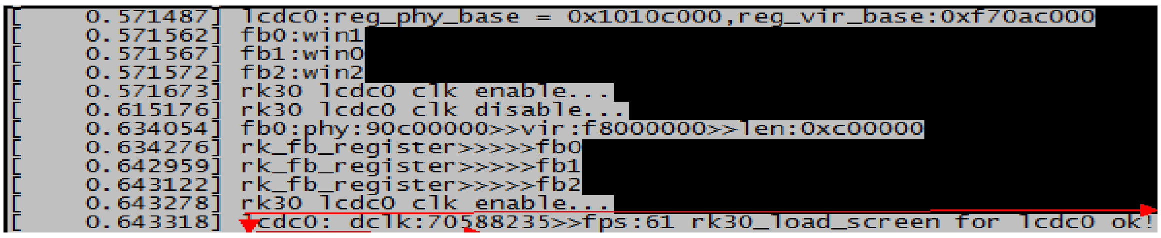
In linux frame buffer subsystem, there is also a parameter should be used--pixclock.

Pixclock = 1/dotclock

For LCDC driver, it should send signals which meet the requirements according to these sequence parameters (DCLK、HSYNC、HBP、HVD、HFP、VSYNC、VBP、VVD、VFP) of screen.

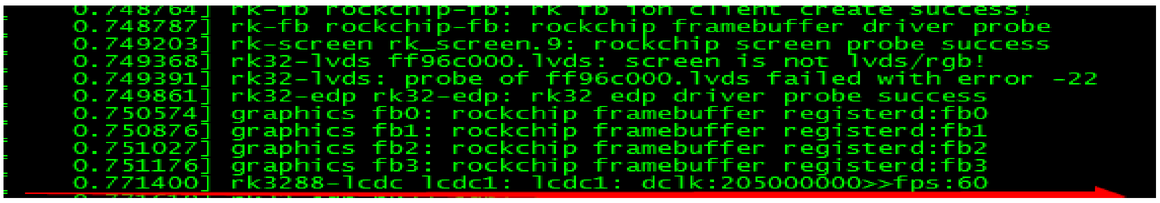
One point needs to be illustrate, the highest refresh rate of Android is 60fps, so we'd better to guarantee the refresh rate of LCDC is also 60fps, it can be known from LCDC refresh rate calculate formula of part one that LCDC refresh rate is in direct proportion to DCLK, and is in inverse proportion to product of sum of horizontal parameters and vertical parameters. We can see from the datasheet of LCD, for a kind of LCD, H_VD and V_VD are related to the resolution of LCD and cannot be modified. BP, FP and PW all have value ranges, when the desired frequence cannot be allocated to DCLK, we can adjust BP, FP and PW to make LCDC refresh rate approach 60 fps.

We can check LCDC refresh rate from log:



```
[      0.571487] lcdc0:reg_phy_base = 0x1010c000,reg_vir_base:0xf70ac000
[      0.571562] fb0:win1
[      0.571567] fb1:win0
[      0.571572] fb2:win2
[      0.571673] rk30 lcdc0 clk enable...
[      0.615176] rk30 lcdc0 clk disable...
[      0.634054] fb0:phy:90c00000>>vir:f8000000>>len:0xc00000
[      0.634276] rk_fb_register>>>>>fb0
[      0.642959] rk_fb_register>>>>>fb1
[      0.643122] rk_fb_register>>>>>fb2
[      0.643278] rk30 lcdc0 clk enable...
[      0.643318] lcdc0: dclk:70588235>>fps:61 rk30_load_screen for lcdc0 ok!
```

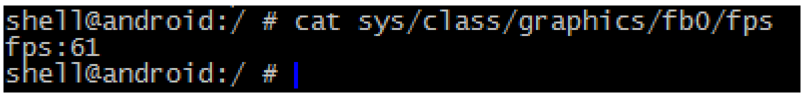
Linux3.0.36 Kernel



```
    0.748764] rk-fb rockchip-fb: rk fb ion client create success!
    0.748787] rk-fb rockchip-fb: rockchip framebuffer driver probe
    0.749203] rk-screen rk_screen.9: rockchip screen probe success
    0.749368] rk32-lvds ff96c000.lvds: screen is not lvds/rgb!
    0.749391] rk32-lvds: probe of ff96c000.lvds failed with error -22
    0.749861] rk32-edp rk32-edp: rk32 edp driver probe success
    0.750574] graphics fb0: rockchip framebuffer registerd:fb0
    0.750876] graphics fb1: rockchip framebuffer registerd:fb1
    0.751027] graphics fb2: rockchip framebuffer registerd:fb2
    0.751176] graphics fb3: rockchip framebuffer registerd:fb3
    0.771400] rk3288-lcdc lcdc1: lcdc1: dclk:205000000>>fps:60
```

Linux3.10 Kernel

Or check through sys node of fb:



```
shell@android:/ # cat sys/class/graphics/fb0/fps
fps:61
shell@android:/ #
```

2 Framework and Implementation of LCD Driver

In linux kernel, associated drivers of LCD are called fb (framebuffer) drivers. On platforms RK30XX, RK292X and RK31XX, to reuse codes, fb drivers are divided into associated part of fb framework, associated part of LCDC, associated part of LCD and associated board configuration part of LCD power operation.

2.1 Associated Codes of fb Framework

drivers/video/fbmem.c

drivers/video/rockchip/rk_fb.c

drivers/video/rockchip/rkfb_sysfs.c

include/linux/rk_fb.h

These codes implement associated framework of fb, hardware operations are not involved, all LCDC (RK3066、 RK302X、RK292X、RK3188、RK3288 and so on) drivers use in common. Among them, fbmem.c is linux kernel native code, it provides user space interaction interfaces (open, read, write, ioctl and so on) upward, and connects with fb driver rk_fb.c of platform downward.

Note: Since platform RK30, we use new framework of fb drivers, not follow codes of RK29 fb any more, thus associated peripheral driver such as LCD configuration file, if you need to use associated data structure of RK fb, please use #include<linux/rk_fb.h> directly, and no longer use old include files such as rk29_fb.h, screen.h and so on of platforms before RK2918, otherwise it will bring relative compatibility problems.

2.2 Associated Codes of LCDC

drivers/video/rockchip/lcdc/rkxxx_lcdc.c:

drivers/video/rockchip/lcdc/rk30_lcdc.c

drivers/video/rockchip/lcdc/rk3066b_lcdc.c

drivers/video/rockchip/lcdc/rk2928_lcdc.c drivers/video/rockchip/lcdc/rk3188_lcdc.c

drivers/video/rockchip/lcdc/rk3288_lcdc.c

These codes are related to specific LCDC, associated codes of different LCDCs are different.

2.3 Associated Codes of LCD Configuration

drivers/video/rockchip/screen/rk_screen.c

include/linux/rk_screen.h

drivers/video/rockchip/screeen/lcd_xxx.c

rk_screen.c is common code of LCD configuration files, mainly realize the following interfaces:

For 3.0.36 version kernel:

set_scaler_info: configuration interface only use one LCDC+RK610 of master control or scaler of RK616 to implement dual-display, it is used only in dual-display mode (CONFIG_ONE_LCDC_DUAL _OUTPUT_INF).

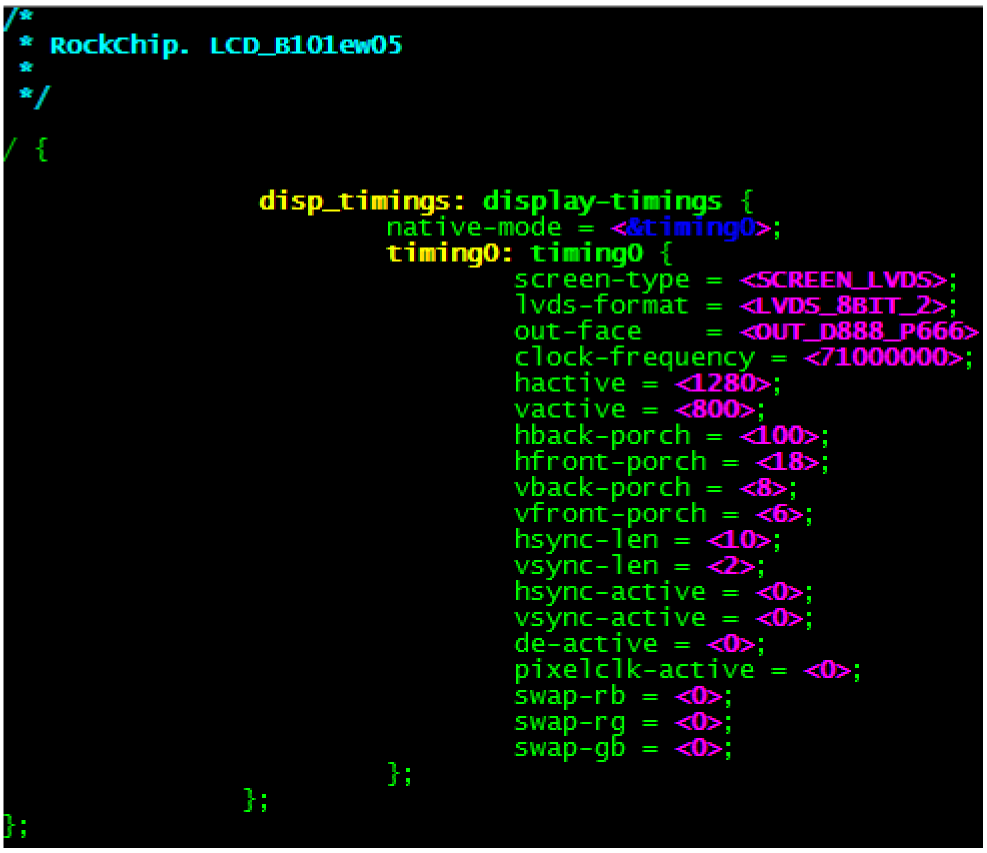
set_lcd_info: LCD parameters configuration interface, used by all LCD.

get_fb_size: to calculate how much fb space to allocate accrdding to LCD resolution, for triple buffer, calculate formula is X*Y*4*3, for double buffer, calculate formula is X*Y*4*2, X and Y represent horizontal resolution and vertical resolution respectively. Currently, we all use triple buffer.

Suggest every member of related project to learn the implementation of interfaces in rk_screen.c, to make it easier to deal with debug problems.

For 3.10 version kernel:

Obtain LCD parameters through probe function (since kernel version 3.10, LCD parameters are passed through dts (lcd_xxx.dtsi)).



```
/*
 * RockChip. LCD_B101ew05
 *
 */

/ {

                disp_timings: display-timings {
                        native-mode = <&timing0>;
                        timing0: timing0 {
                                screen-type = <SCREEN_LVDS>;
                                lvds-format = <LVDS_8BIT_2>;
                                out-face    = <OUT_D888_P666>;
                                clock-frequency = <71000000>;
                                hactive = <1280>;
                                vactive = <800>;
                                hback-porch = <100>;
                                hfront-porch = <18>;
                                vback-porch = <8>;
                                vfront-porch = <6>;
                                hsync-len = <10>;
                                vsync-len = <2>;
                                hsync-active = <0>;
                                vsync-active = <0>;
                                de-active = <0>;
                                pixelclk-active = <0>;
                                swap-rb = <0>;
                                swap-rg = <0>;
                                swap-gb = <0>;
                        };
                };
};
```

screen-type: it can be assigned values SCREEN_LVDS, SCREEN_RGB, SCREEN_EDP and SCREEN_MIPI

lvds-format: if LVDS LCD need to set this format, it can be assigned values LVDS_8BIT_1, LVDS_8BIT_2 and LVDS_8BIT_3

out-face: LCD connecting format

Values of the three parameters above are all defined in include/dt-bindings/rkfb/rk_fb.h.

clock-frequency：frequrncy of dclk,unit hz

hactive：LCD horizontal resolution

vactive：LCD vertical resolution

hsync-len：LCD horizontal synchronization length

vsync-len：LCD vertical synchronization length

Hsync-active, vsync-active, de-active , pixelclock-active: they are polarity control of hsync, vsync, DEN and dclk respectively. Set 1 is to reverse clock polarity.

Swap-rb, swap-rg, swap-gb: Set 1 is to reverse corresponding color.

The SDK we have released already contains below typical screen configuration dtsi files:

Lcd-b1010ew05.dtsi /*1280*800 的 LVDS screen*/

Lcd-LP097QX1.dtsi /*2048*1536 的 EDP screen */

Lcd-wqxga-mipi.dtsi /*mipi screen */

In project development, above screen parameters could be references, in board level dts files include corresponding screen dtsi files.

Provide rk_fb_get_prmry_screen (struct rk_screen *screen) interface to other modules of kernel to get screen parameters.

In linux 3.0.36 version kernel:

lcd_xxx.c is screen parameter configuration file, and must be named in the format of lcd_xxx.c (begin with lcd_). After special processing by Makefile when compiling, the content of lcd_xxx.c will be copied into the generated file lcd.h, rk_screen.c will contains lcd.h head file, then use correlated parameters defined by corresponding screen configuration files to initialize.

This part codes are designed according to LCD datasheet and configuration parameters associated with LCD screen. These codes are also modified most during products development, the contents involved will be explained in details as following. As lcd_b101ew05.c for example, it is the corresponding driver of the screen used by RK3066X SDK board.

Below macros are important:

```
#if   defined(CONFIG_RK610_LVDS) || defined(CONFIG_RK616_LVDS)
#define SCREEN_TYPE              SCREEN_LVDS
#else
#define SCREEN_TYPE              SCREEN_RGB
```

```
#endif

#define LVDS_FORMAT             LVDS_8BIT_2

#define OUT_FACE               OUT_D888_P666




#define DCLK                   71000000

#define LCDC_ACLK               300000000           //29 lcdc axi DMA frequency


/* Timing */

#define H_PW            10

#define H_BP            100

#define H_VD            1280

#define H_FP            18


#define V_PW            2

#define V_BP            8

#define V_VD            800

#define V_FP            6


#define LCD_WIDTH               216

#define LCD_HEIGHT              135




#if defined(CONFIG_RK610_LVDS) || defined(CONFIG_RK616_LVDS)

#define DCLK_POL        1

#else
```

```
#define DCLK_POL        0
#endif
#define DEN_POL         0
#define VSYNC_POL       0
#define HSYNC_POL       0

#define SWAP_RB         0
#define SWAP_RG         0
#define SWAP_GB         0
```

SCREEN_TYPE means screen type (RGB, LVDS, MIPI, HDMI, MCU), if the screen is LVDS screen, and corresponding LVDS must be drived by procedure (for example, RK610, RK616 or rk292x, RK302X platforms have integrated lvds module inside), so the macro should be defined as SCREEN_LVDS. If the screen is RGB screen, the macro should be defined as SCRREN_RGB, MIPI screen should be defined as SCREEN_MIPI.

LVDS_FORMAT indicates LVDS signal distribution method, combine the connection methods of screen and LVDS on hardware defined as one of LVDS_8BIT_1, LVDS_8BIT_2, LVDS_8BIT_3 and LVDS_6BIT, the definition of this macro is in rk_screen.h and the commends indicate the corresponding connection way, specific LVDS screen datasheet also explains the distribution method of signal on each LVDS channel. As show in Figure (4):

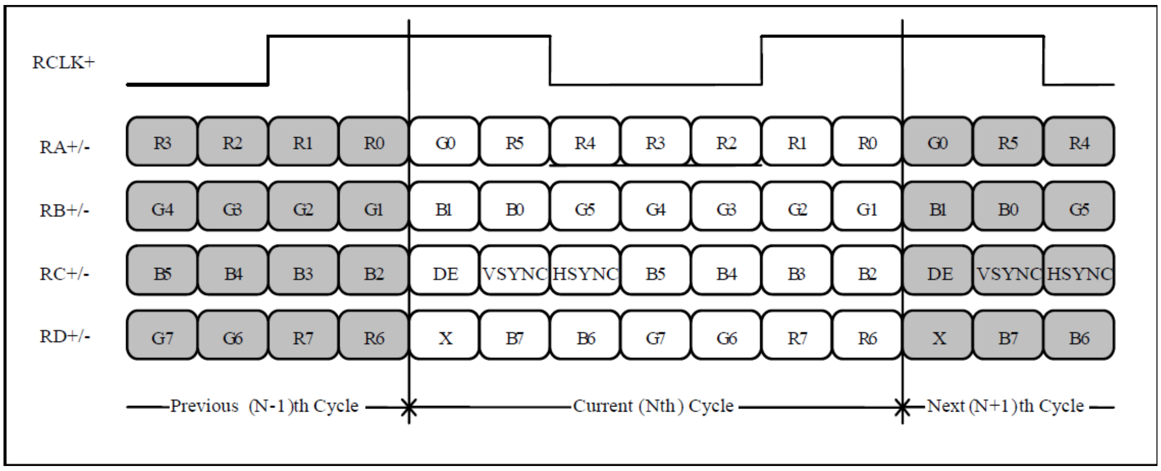

RCLK+
RA+/-
RB+/-
RC+/-
RD+/-
R3 R2 R1 R0 G0 R5 R4 R3 R2 R1 R0 G0 R5 R4
G4 G3 G2 G1 B1 B0 G5 G4 G3 G2 G1 B1 B0 G5
B5 B4 B3 B2 DE VSYNC HSYNC B5 B4 B3 B2 DE VSYNC HSYNC
G7 G6 R7 R6 X B7 B6 G7 G6 R7 R6 X B7 B6
Previous (N-1)th Cycle
Current (Nth) Cycle
Next (N+1)th Cycle

Figure (4) LVDS Data Format

The figure above illustrates that the R0~R5 and G0 are transmitted on Y0 channel, G1~G5, B0 and B1 are transmitted on Y1 channel, B2~B5, VSYNC, HSYNC and DEN are transmitted on Y2 channel, two high bits of various colors are transimmitted on Y3 channel. Refer to the definition in rk_screen.h, we know that the format is corresponding to LVDS_8BIT_1.

OUT_FACE represents how many bits of connection way should be used by screen, if screen is 24 bit, it should be OUT_P888. If screen is 18 bit, data wire should be connected to high 6 bit of each unit color respectively and defined as OUT_P888_P666, if data wire is connected to LCDC low 16 bit, it is defined as OUT_P999. If the configuration is incorrect, for example, 24 bit connection is defined as OUT_D888_P666 or OUT_P666, the color gradation may appear. For screen connection methods, please refer to “LVDS application description.pdf”.

More attention, a lot of screens have pins connected to high level or low level to control the screen use 18 bit or 24 bit, we should pay more attention during actual development process.

Another point to note, for 18 bit screen, LVDS data format of this screen generally as Figure (5) show:

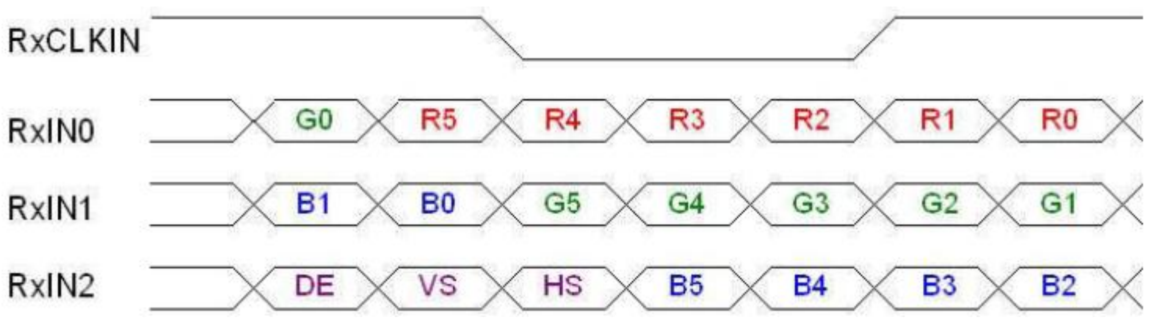
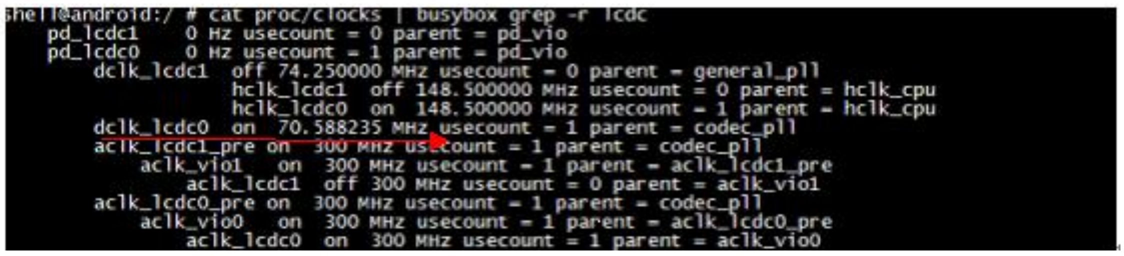


Figure (5)

18 bit screens usually use three LVDS differential channels, corresponding format is LVDS_6BIT.

However, if only use one LCDC+RK610/RK616 Scaler of master control to realize dual display, OUT_FACE must be defined as OUT_D888_P666, LVDS_FORMAT must be defined as LVDS_8BIT_2, otherwise in HDMI mode, screen will display abnormally.

DCLK is screen dot clock, set according to LCD data sheet. Due to clock divider limitation, it may not divide an accurate desired CLK, maybe you want to define a 65MHZ clk, but actually you get a 63MHZ clk. Because most clock frequencies are integer, in order to be assigned frequency as close as possible, please adjust values set by DCLK according to actual values. Associated clock of LCDC can be obtained by the below command cat proc/clocks | busybox grep -r lcdc or measured by oscilloscope.



```
shell@android:/ # cat proc/clocks | busybox grep -r lcdc
    pd_lcdc1     0 Hz usecount = 0 parent = pd_vio
    pd_lcdc0     0 Hz usecount = 1 parent = pd_vio
        dclk_lcdc1  off 74.250000 MHz usecount = 0 parent = general_pll
                    hclk_lcdc1  off 148.500000 MHz usecount = 0 parent = hclk_cpu
                    hclk_lcdc0  on  148.500000 MHz usecount = 1 parent = hclk_cpu
        dclk_lcdc0  on  70.588235 MHz usecount = 1 parent = codec_pll
        aclk_lcdc1_pre on  300 MHz usecount = 1 parent = codec_pll
            aclk_vio1   on  300 MHz usecount = 1 parent = aclk_lcdc1_pre
                aclk_lcdc1  off 300 MHz usecount = 0 parent = aclk_vio1
        aclk_lcdc0_pre on  300 MHz usecount = 1 parent = codec_pll
            aclk_vio0   on  300 MHz usecount = 1 parent = aclk_lcdc0_pre
                aclk_lcdc0  on  300 MHz usecount = 1 parent = aclk_vio0
```

LCDC_ACLK is AXI bus clock of LCDC, which is a clock used by LCDC moving data in fb through DMA. On later RK30XX platforms, it is locked by clock management code, and cannot be set by external driver. No matter how much the macro defines is doesn’t matter. Keep it is just to be compatible with the previous codes before RK29.

H_PW, H_FP, H_BP, H_VD, V_PW, V_FP, V_BP, V_VD refer to the foregoing description, set according to lcd datasheet.

LCD_WIDTH, LCD_HIGHT are length and width of the screen, unit mm, set according to lcd datasheet.

DCLK_POL is polarity of dclk clock, is defined as 0 by default, but the pixels on the screen display like a shaking feel, make it to 1, rollover may give a better effect.

In set_lcd_info function, it has the following codes:

```
screen->pin_vsync = VSYNC_POL;
screen->pin_den = DEN_POL;
screen->pin_dclk = DCLK_POL;
```

They are used to set signal polarity of vsync, hsync, dclk and den, if lcd datasheet has requirements, it can do corresponding invert to set 1, while most lcd do not need to invert.

SWAP_RB, SWAP_RG and SWAP_GB are used to exchange RGB signals, for some projects, LCDC output data display reverse color on screen (RGB three primary colors reversed, such as red to green, green to blue) due to wiring connection or screen self-property, then set the corresponding macro as 1 to do color correct. As for how to judge which color is reversed quickly, please refer to Appendix "Judge whether LCD screen color display normal quickly ".

If use RK610 or RK616, and master control only uses one LCDC + RK610 or RK616 scaler to realize dual display, scaler parameters need to be added in screen driver configuration. For more details, please refer to "RK610 Configuration _V1.2.pdf" and "Rockchip_RK61X Development Guide".

If Board ID mechanism is used in screen, the macro of RK_USE_SCREEN_ID should be defined in screen configuration file, and implement set_lcd_info_by_id interface.

If some screens need to initialize, the macro of RK _SCREEN_INIT should be defined in screen configuration file, and implement rk_lcd_init and rk_lcd_standby interface.

If some screens need to adjust display effect by LUT, the macro of USE_RK_DSP_LUT should be defined in screen configuration file, and define default dsp_lut array.

Some screen configuration files in SDK are usually used on our SDK boards or development prototypes, for reference:

(1) drivers/video/rockchip/screen/lcd_b101ew05.c

(2) drivers/video/rockchip/screen/lcd_LP097QX1.c

(3) drivers/video/rockchip/screen/lcd_hdmi_800x480.c

(4) drivers/video/rockchip/screen/lcd_hdmi_1024x600.c

(5) drivers/video/rockchip/screen/lcd_hdmi_1024x768.c

(6) drivers/video/rockchip/screen/lcd_hdmi_1280x800.c

(7) drivers/video/rockchip/screen/lcd_hsd100pxn.c

(1) is LVDS screen used on rk30xx and rk31xx sdk boards, resolution is 1280*800, support single LCDC + RK610 dual display solution.

(2) is HD screen, resolution is 2048*1536, interface is EDP.

(3), (4), (5), (6) are configuration files of several commonly used resolution screens in single LCDC + RK610 dual display solution.

(7) is LVDS screen used on rk29xx sdk boards, support LVDS interface of RK2918 and RK2918.

For Linux 3.10 later version of kernel, screen parameters are transmitted by dts files (lcd_xxx.dtsi). Screen catalog only keep lcd_general.c and lcd_mipi.c two screen configuration-related files. lcd_general.c is kept for subsequent expansion, if some screens need special control, they could implement related control interface in this file, then register in rk_screen.c, similar to linux 3.0.36 processing mechanism. lcd_mipi.c must be selected when using mipi screen.

2.4 Associated Configuration of LCD Power Operation

2.4.1 Board Configuration Based on Linux 3.0.36 Kernel

Due to different designs to different LCD power control in different projects, with different timing sequence. In addition, LCD screen and HDMI can choose flexibly to connect to LCDC0 or LCDC1 based on demands of hardware design for platforms with two LCDC (like RK3066, RK3168 and RK3188). These codes implement in board files, specific interfaces are shown below:

```
#ifdef CONFIG_FB_ROCKCHIP

#define LCD_CS_PIN          INVALID_GPIO
#define LCD_CS_VALUE         GPIO_HIGH

#define LCD_EN_PIN          RK30_PIN0_PB0
#define LCD_EN_VALUE         GPIO_HIGH

static int rk_fb_io_init(struct rk29_fb_setting_info *fb_setting)
{
    int ret = 0;

    if(LCD_CS_PIN !=INVALID_GPIO)
    {
        ret = gpio_request(LCD_CS_PIN, NULL);
        if (ret != 0)
        {
            gpio_free(LCD_CS_PIN);
            printk(KERN_ERR "request lcd cs pin fail!\n");
            return -1;
        }
        else
        {
            gpio_direction_output(LCD_CS_PIN, LCD_CS_VALUE);
        }
    }

    if(LCD_EN_PIN !=INVALID_GPIO)
    {
        ret = gpio_request(LCD_EN_PIN, NULL);
        if (ret != 0)
        {
            gpio_free(LCD_EN_PIN);
            printk(KERN_ERR "request lcd en pin fail!\n");
            return -1;
        }
        else
        {
            gpio_direction_output(LCD_EN_PIN, LCD_EN_VALUE);
        }
    }
    return 0;
```

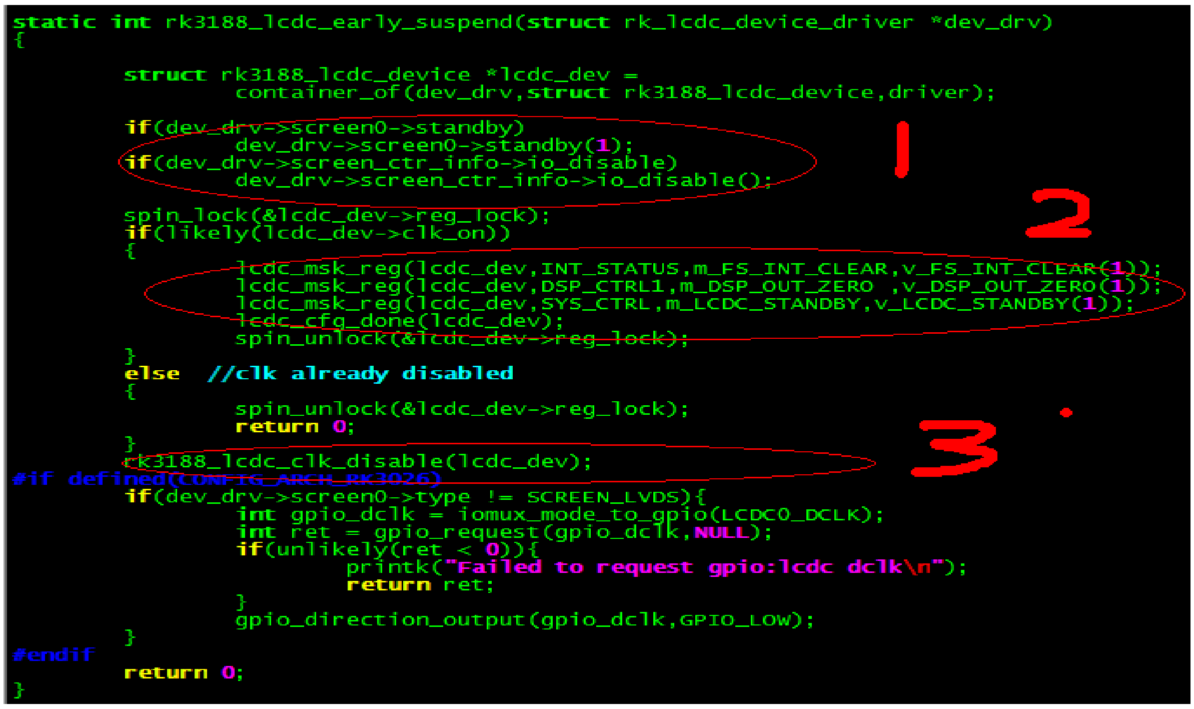
```
}
static int rk_fb_io_disable(void)
{
    if(LCD_CS_PIN !=INVALID_GPIO)
    {
        gpio_set_value(LCD_CS_PIN, !LCD_CS_VALUE);
    }
    if(LCD_EN_PIN !=INVALID_GPIO)
    {
        gpio_set_value(LCD_EN_PIN, !LCD_EN_VALUE);
    }
    return 0;
}
static int rk_fb_io_enable(void)
{
    if(LCD_CS_PIN !=INVALID_GPIO)
    {
        gpio_set_value(LCD_CS_PIN, LCD_CS_VALUE);
    }
    if(LCD_EN_PIN !=INVALID_GPIO)
    {
        gpio_set_value(LCD_EN_PIN, LCD_EN_VALUE);
    }
    return 0;
}
```

```
#if defined(CONFIG_LCDC0_RK3066B) || defined(CONFIG_LCDC0_RK3188)
struct rk29fb_info lcdc0_screen_info = {
#if defined(CONFIG_RK_HDMI) && defined(CONFIG_HDMI_SOURCE_LCDC0) &&
defined(CONFIG_DUAL_LCDC_DUAL_DISP_IN_KERNEL)
    .prop        = EXTEND,   //extend display device
    .io_init     = NULL,
    .io_disable = NULL,
    .io_enable = NULL,
    .set_screen_info = hdmi_init_lcdc,
#else
    .prop        = PRMRY,         //primary display device
    .io_init    = rk_fb_io_init,
    .io_disable = rk_fb_io_disable,
    .io_enable = rk_fb_io_enable,
    .set_screen_info = set_lcd_info,
#endif
};
#endif

#if defined(CONFIG_LCDC1_RK3066B) || defined(CONFIG_LCDC1_RK3188)
struct rk29fb_info lcdc1_screen_info = {
#if defined(CONFIG_RK_HDMI) && defined(CONFIG_HDMI_SOURCE_LCDC1) &&
defined(CONFIG_DUAL_LCDC_DUAL_DISP_IN_KERNEL)
    .prop        = EXTEND,   //extend display device
    .io_init     = NULL,
    .io_disable = NULL,

    .io_enable = NULL,

    .set_screen_info = hdmi_init_lcdc,

#else

    .prop       = PRMRY,        //primary display device

    .io_init    = rk_fb_io_init,

    .io_disable = rk_fb_io_disable,

    .io_enable = rk_fb_io_enable,

    .set_screen_info = set_lcd_info,

#endif

};

#endif




static struct resource resource_fb[] = {

    [0] = {

        .name   = "fb0 buf",

        .start = 0,

        .end    = 0,//RK30_FB0_MEM_SIZE - 1,

        .flags = IORESOURCE_MEM,

    },

    [1] = {

        .name   = "ipp buf",  //for rotate

        .start = 0,

        .end    = 0,//RK30_FB0_MEM_SIZE - 1,
        .flags = IORESOURCE_MEM,
    },
    [2] = {
        .name   = "fb2 buf",
        .start = 0,
        .end    = 0,//RK30_FB0_MEM_SIZE - 1,
        .flags = IORESOURCE_MEM,
    },
};

static struct platform_device device_fb = {
    .name         = "rk-fb",
    .id       = -1,
    .num_resources     = ARRAY_SIZE(resource_fb),
    .resource    = resource_fb,
};
#endif

#if defined(CONFIG_LCDC0_RK3066B) || defined(CONFIG_LCDC0_RK3188)
static struct resource resource_lcdc0[] = {
    [0] = {
        .name   = "lcdc0 reg",
        .start = RK30_LCDC0_PHYS,
        .end    = RK30_LCDC0_PHYS + RK30_LCDC0_SIZE - 1,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .name   = "lcdc0 irq",
        .start = IRQ_LCDC0,
        .end    = IRQ_LCDC0,
        .flags = IORESOURCE_IRQ,
    },
};

static struct platform_device device_lcdc0 = {
    .name           = "rk30-lcdc",
    .id         = 0,
    .num_resources        = ARRAY_SIZE(resource_lcdc0),
    .resource       = resource_lcdc0,
    .dev          = {
        .platform_data = &lcdc0_screen_info,
    },
};
#endif
#if defined(CONFIG_LCDC1_RK3066B) || defined(CONFIG_LCDC1_RK3188)
static struct resource resource_lcdc1[] = {
    [0] = {
        .name   = "lcdc1 reg",
        .start = RK30_LCDC1_PHYS,
        .end    = RK30_LCDC1_PHYS + RK30_LCDC1_SIZE - 1,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .name   = "lcdc1 irq",
        .start = IRQ_LCDC1,
        .end    = IRQ_LCDC1,
        .flags = IORESOURCE_IRQ,
    },
};

static struct platform_device device_lcdc1 = {
    .name           = "rk30-lcdc",
    .id             = 1,
    .num_resources          = ARRAY_SIZE(resource_lcdc1),
    .resource       = resource_lcdc1,
    .dev            = {
        .platform_data = &lcdc1_screen_info,
    },
};
#endif
```

Here is the power control interface of rk30xx sdk board. rk_fb_io_init is the initialization function of relevant power control IO, when initializing fb lcdc system, need call the following:

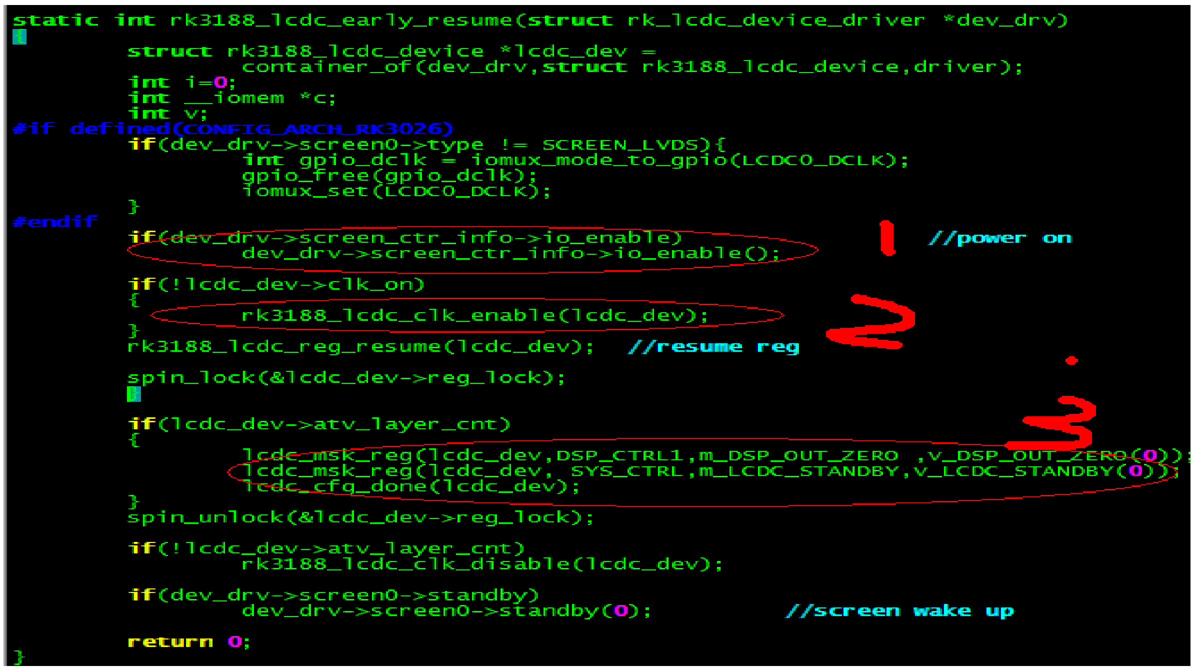
```
static int __devinit rk3188_lcdc_probe(struct platform_device *pdev)
{
        struct rk3188_lcdc_device *lcdc_dev = NULL;
        struct device *dev = &pdev->dev;
        rk_screen *screen;


        ......

        if(screen_ctr_info->set_screen_info)

        {


screen_ctr_info->set_screen_info(screen,screen_ctr_info->lcd_info)

                if(screen_ctr_info->io_init)

                        screen_ctr_info->io_init(NULL);//rk_fb_io_init

        }


       rk_fb_register(&(lcdc_dev->driver),&lcdc_driver,lcdc_dev->id);


        }


        printk("rk3188 lcdc%d probe ok!\n",lcdc_dev->id);
```

rk_fb_io_disable and rk_fb_io_enable are interfaces of power opening and closing. When system enters into suspend, resume and shutdown, call these:



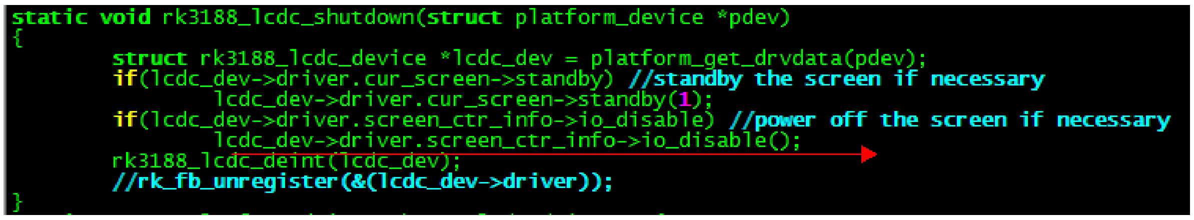
```
static int rk3188_lcdc_early_suspend(struct rk_lcdc_device_driver *dev_drv)
{

	struct rk3188_lcdc_device *lcdc_dev =
		container_of(dev_drv,struct rk3188_lcdc_device,driver);

	if(dev_drv->screen0->standby)
		dev_drv->screen0->standby(1);
	if(dev_drv->screen_ctr_info->io_disable)
		dev_drv->screen_ctr_info->io_disable();

	spin_lock(&lcdc_dev->reg_lock);
	if(likely(lcdc_dev->clk_on))
	{
		lcdc_msk_reg(lcdc_dev,INT_STATUS,m_FS_INT_CLEAR,v_FS_INT_CLEAR(1));
		lcdc_msk_reg(lcdc_dev,DSP_CTRL1,m_DSP_OUT_ZERO ,v_DSP_OUT_ZERO(1));
		lcdc_msk_reg(lcdc_dev,SYS_CTRL,m_LCDC_STANDBY,v_LCDC_STANDBY(1));
		lcdc_cfg_done(lcdc_dev);
		spin_unlock(&lcdc_dev->reg_lock);
	}
	else  //clk already disabled
	{
		spin_unlock(&lcdc_dev->reg_lock);
		return 0;
	}
	rk3188_lcdc_clk_disable(lcdc_dev);
#if defined(CONFIG_ARCH_RK3026)
	if(dev_drv->screen0->type != SCREEN_LVDS){
		int gpio_dclk = iomux_mode_to_gpio(LCDC0_DCLK);
		int ret = gpio_request(gpio_dclk,NULL);
		if(unlikely(ret < 0)){
			printk("Failed to request gpio:lcdc dclk\n");
			return ret;
		}
		gpio_direction_output(gpio_dclk,GPIO_LOW);
	}
#endif
	return 0;
}
```

These are LCDC early suspend codes of RK3188 above, and will be called after the system into primary sleep. These codes are divided into three parts: first part, call rk_fb_io_disable is used to close display related power, second part is used to close LCDC data output, and third part is used to close related clock output of LCDC like DCLK.



```
static int rk3188_lcdc_early_resume(struct rk_lcdc_device_driver *dev_drv)
{
	struct rk3188_lcdc_device *lcdc_dev =
		container_of(dev_drv,struct rk3188_lcdc_device,driver);
	int i=0;
	int __iomem *c;
	int v;
#if defined(CONFIG_ARCH_RK3026)
	if(dev_drv->screen0->type != SCREEN_LVDS){
		int gpio_dclk = iomux_mode_to_gpio(LCDC0_DCLK);
		gpio_free(gpio_dclk);
		iomux_set(LCDC0_DCLK);
	}
#endif
	if(dev_drv->screen_ctr_info->io_enable)
		dev_drv->screen_ctr_info->io_enable();              //power on

	if(!lcdc_dev->clk_on)
	{
		rk3188_lcdc_clk_enable(lcdc_dev);
	}
	rk3188_lcdc_reg_resume(lcdc_dev);   //resume reg

	spin_lock(&lcdc_dev->reg_lock);

	if(lcdc_dev->atv_layer_cnt)
	{
		lcdc_msk_reg(lcdc_dev,DSP_CTRL1,m_DSP_OUT_ZERO ,v_DSP_OUT_ZERO(0));
		lcdc_msk_reg(lcdc_dev, SYS_CTRL,m_LCDC_STANDBY,v_LCDC_STANDBY(0));
		lcdc_cfg_done(lcdc_dev);
	}
	spin_unlock(&lcdc_dev->reg_lock);

	if(!lcdc_dev->atv_layer_cnt)
		rk3188_lcdc_clk_disable(lcdc_dev);

	if(dev_drv->screen0->standby)
		dev_drv->screen0->standby(0);          //screen wake up

	return 0;
}
```

These are late resume function of RK3188 LCDC above called when waking up system. These codes can also be divided into three parts: first part, call rk_fb_io_enable to open display relevant power, second part, used to open display related clock output like DCLK, third part, LCDC starts to export data.



```
static void rk3188_lcdc_shutdown(struct platform_device *pdev)
{
        struct rk3188_lcdc_device *lcdc_dev = platform_get_drvdata(pdev);
        if(lcdc_dev->driver.cur_screen->standby) //standby the screen if necessary
                lcdc_dev->driver.cur_screen->standby(1);
        if(lcdc_dev->driver.screen_ctr_info->io_disable) //power off the screen if necessary
                lcdc_dev->driver.screen_ctr_info->io_disable();
        rk3188_lcdc_deint(lcdc_dev);
        //rk_fb_unregister(&(lcdc_dev->driver));
}
```

Above is shutdown function of RK3188 LCDC, call this function when system powers off. If we need to process the related power when system powers off, it can be done in this function.

These interfaces can realize similarly in other platforms like RK30XX, RK292X and so on. And these functions need fine adjustment for specific projects, including operation order of these three parts, and whether add delay or not between each part, they are both related to specific hardware design and screen.

lcdc0_screen_info and lcdc1_screen_info are control interfaces of display devices connected with LCDC0 and LCDC1 respectively. All these settings are set according to specific connection situation of LCD, HDMI and LCDC. Among them, prop is the property of LCDC, and if LCDC is connected with LCD screen, the prop of this LCDC should be set as PRMRY, which means the primary display device, then make the related power control interface of LCD mount on the lcdcx_screen_info of lcdc (x=0,1). If LCDC is connected with HDMI, set the prop of this LCDC as EXTEND, which means extend display device, then mount the initialization information of HDMI on lcdcx_screen_info of this LCDC (x=0,1), mainly hdmi_init_lcdc. If only use single LCDC and RK61X to realize dual display, the prop of this LCDC should be set as PRMRY and its lcdcx_screen_info (x=0,1) should be connected with related power control interface of LCD.

Attention on RK3028, it implements MCP packaging based on RK3168 and RK610, and is compatible with RK2928 Pin to Pin. On this platform, the connection relationship between LCDC and RK610 is: LCDC0 of master control is connected with LCD0 of RK610, LVDS output port of RK610 is parallel with LCDC1 output port of master control. When we connect with LVDS screen, display signal is outputted from LCDC0 of master control, transmitted through RK610's LCD0 to LVDS. LCDC1 of master control is not in use, dual display can only be implemented by scaler modual of RK610 through setting the prop of Lcdc0_screen_info as PRMRY. If connected with RGB screen, display signal on RGB LCD screen is outputted from LCDC1 of master control, and delivered to RGB screen directly. Dual display can be realized by dual LCDC, LCDC0 of master control sends HDMI signal to HDMI via LCD0 of RK610. In this case, the prop of lcdc0_screen_info should be set as EXTEND, the prop of lcdc1_screen _info should be set as PRMRY as the primary display device.

device_fb, device_lcdc0, device_lcdc1 are corresponding platform device structures of fb, lcdc0, lcdc1, these three structures cannot be changed at will.

To realize flexible connection configuration of LCD, HDMI and LCDC, there is a specific function rk_platform_add_display_devices() to register fb, lcdc, backlight devices in board files to guarantee these devices loaded as the order rk fb driver, primary display device driver (PRMRY), extend display device driver (EXTEND), then backlight driver.

```
static int rk_platform_add_display_devices(void)
{
    struct platform_device *fb = NULL;   //fb
    struct platform_device *lcdc0 = NULL; //lcdc0
    struct platform_device *lcdc1 = NULL; //lcdc1
    struct platform_device *bl = NULL; //backlight
#ifdef CONFIG_FB_ROCKCHIP
```

```
    fb = &device_fb;
#endif


#if defined(CONFIG_LCDC0_RK3066B) || defined(CONFIG_LCDC0_RK3188)
    lcdc0 = &device_lcdc0,
#endif


#if defined(CONFIG_LCDC1_RK3066B) || defined(CONFIG_LCDC1_RK3188)
    lcdc1 = &device_lcdc1,
#endif


#ifdef CONFIG_BACKLIGHT_RK29_BL
    bl = &rk29_device_backlight,
#endif
    __rk_platform_add_display_devices(fb,lcdc0,lcdc1,bl);


    return 0;


}
```

The function is called in .init_machine interface:

```
static void __init machine_rk30_board_init(void)
{
    //avs_init();
    gpio_request(POWER_ON_PIN, "poweronpin");
    gpio_direction_output(POWER_ON_PIN, GPIO_HIGH);

    pm_power_off = rk30_pm_power_off;


   gpio_direction_output(POWER_ON_PIN, GPIO_HIGH);



    rk30_i2c_register_board_info();

    spi_register_board_info(board_spi_devices, ARRAY_SIZE(board_spi_devices));

    platform_add_devices(devices, ARRAY_SIZE(devices));

    rk_platform_add_display_devices();

    board_usb_detect_init(RK30_PIN0_PA7);


#ifdef CONFIG_WIFI_CONTROL_FUNC

    rk29sdk_wifi_bt_gpio_control_init();

#endif

}
```

fb device must be allocated memory which is continuous in physics according to screen resolution in advance, the operation is realized in .reserve interface:

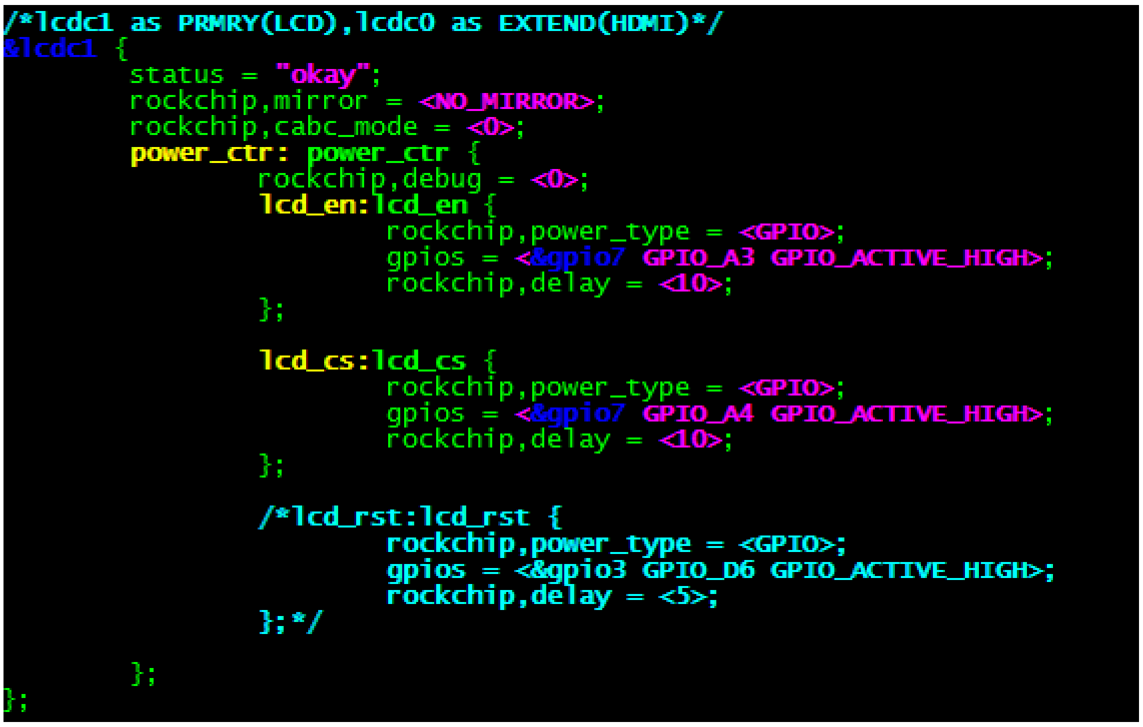
```
static void __init rk30_reserve(void)

{

#ifdef CONFIG_ION

    rk30_ion_pdata.heaps[0].base        =        board_mem_reserve_add("ion",
ION_RESERVE_SIZE);

#endif

#ifdef CONFIG_FB_ROCKCHIP

    resource_fb[0].start = board_mem_reserve_add("fb0 buf", get_fb_size());

    resource_fb[0].end = resource_fb[0].start + get_fb_size()- 1;
#if defined(CONFIG_FB_ROTATE) || !defined(CONFIG_THREE_FB_BUFFER)
    resource_fb[2].start = board_mem_reserve_add("fb2 buf",get_fb_size());
    resource_fb[2].end = resource_fb[2].start + get_fb_size() - 1;
#endif
#endif

#ifdef CONFIG_VIDEO_RK29
    rk30_camera_request_reserve_mem();
#endif

#ifdef CONFIG_GPS_RK
    //it must be more than 8MB
    rk_gps_info.u32MemoryPhyAddr = board_mem_reserve_add("gps", SZ_8M);
#endif
    board_mem_reserved();
}
```

The size of allocated memory is obtained from get_fb_size() interface which is realized in rk_screen.c:

```
size_t get_fb_size(void)
{
    size_t size = 0;
    #if defined(CONFIG_THREE_FB_BUFFER)
        size = ((H_VD)*(V_VD)<<2)* 3; //three buffer
    #else
        size = ((H_VD)*(V_VD)<<2)<<1; //two buffer
    #endif
    return ALIGN(size,SZ_1M);
}
```

2.4.2 dts Configuration based on Linux 3.10 Kernel

For Linux 3.10 later version kernel, board files are no longer used, corresponding hardware configuration parameters are transmitted by dts. To realize power control, power_ctr control node is realized in lcdc node of dts:



```
/*lcdc1 as PRMRY(LCD),lcdc0 as EXTEND(HDMI)*/
&lcdc1 {
        status = "okay";
        rockchip,mirror = <NO_MIRROR>;
        rockchip,cabc_mode = <0>;
        power_ctr: power_ctr {
                rockchip,debug = <0>;
                lcd_en:lcd_en {
                        rockchip,power_type = <GPIO>;
                        gpios = <&gpio7 GPIO_A3 GPIO_ACTIVE_HIGH>;
                        rockchip,delay = <10>;
                };

                lcd_cs:lcd_cs {
                        rockchip,power_type = <GPIO>;
                        gpios = <&gpio7 GPIO_A4 GPIO_ACTIVE_HIGH>;
                        rockchip,delay = <10>;
                };

                /*lcd_rst:lcd_rst {
                        rockchip,power_type = <GPIO>;
                        gpios = <&gpio3 GPIO_D6 GPIO_ACTIVE_HIGH>;
                        rockchip,delay = <5>;
                };*/

        };
};
```

"power_ctr", "rockchip, debug", "rockchip, power_type", "gpios", "rockchip, delay" are key words used in systems, cannot be modified.

"rockchip, debug" node, if set as 1, it will open debug function of power analysis in fb.

"rockchip, power_type" node, set power control type, if use GPIO, set as GPIO, if use regular of PMU, set as REGULATOR.

"rockchip, delay" is the delay time needed after opening the power, unit is ms.

lcd_en, lcd_cs and lcd_rst whose name can be modified according to the function. Kernel resolves these power control nodes in the order which they appears in dts. So please program as the power on order for these nodes.

In fb codes, associated interfaces of power control are shown below:

int rk_disp_pwr_ctr_parse_dt(struct rk_lcdc_driver *dev_drv) //Resolve dts

int rk_disp_pwr_enable/disable(struct rk_lcdc_driver *dev_drv) //Switch interface of power, fb codes call this interface directly to switch related power of display. If some project has special power control requirements and it is diffcult to describe dts, we can modify the two interfaces directly to meet the project requirements.

2.4.3 make menuconfig Configuration

```
Device Drivers    --->
    Graphics support    --->
    <*>Frame buffer support for Rockchip --->
    [   ]   Mirroring Support
              Dual display ploy select(implement dual display in kernel with dual lcdc)
    [   ]   FB rotate support (NEW)
    [ * ]   Three fb buffer support
    [ * ]   rk3188 lcdc support
    [ * ]   lcdc0 support
    [   ]   lcdc0 1.8V io support
    [ * ]   lcdc1 support
    [   ]   lcdc1 1.8V io support
              LCD Panel Select (Display Port screen LP097QX1)    --->
    [   ]   RockChip display transmitter support    --->
```

a.   [ ] Mirroring support

This configuration item is related with wimmo function and has nothing to do with normal projects. There is no need to configure unless you clearly know how it works and your purpose.

b. Dual display poly select ( )

This configuration provides selection for dual display, dual display has multiple ways to achieve by SOC plan later than RK30XX. RK30XX/RK3168/RK3188/RK302X have two LCDC, R292X has only one LCDC.

(1) Double lcdc realizes dual display such as rk3066 which uses two lcdc to realize double display.
(2) Combination of single lcdc and rk610 realizes dual display such as rk3066 LCDC0+rk610.
(3) Unilateral display, which is switching to HDMI mode, the screen of pad goes off such as rk2916.

Which ploy is chosen depends on hardware design:

Dual display ploy select (implement dual display in kernel with dual lcdc) --->

( ) implement dual display in kernel with dual lcdc

This configuration uses double lcdc to realize dual display

( ) one lcdc dual output display interface support

This configuration uses single lcdc and rk61X to realize dual display

( ) no dual display needed

This configuration only supports unilateral display such as rk2926, rk3026

Highlight: Implement dual display by using LCDC+RK61X has convenience for hardware design but also many limitations and is not recommended.

(1) Due to RK61X scaler does not support rotation function, dual LCDC is the only

way to implement dual display instead of LCDC+RK61X if vertical screen needs to do 180° rotation.

(2) In case of sensitivity to time sequence, MIPI screen can only receive fixed time sequence when system is running, otherwise abnormal warning is coming. When we use LCDC+RX61X scaler function to realize dual display, we need to switch time sequence of LCD screen at the moment that HDMI is launched. If we need to launch HDMI to realize dual display by using MIPI screen, we must implement LCDC mode.

(3) Dual display implemented by LCDC+RK61X scaler need adjustment of LCD time sequence when HDMI is launched, some LCD screen requiring strict time sequence may not display very well.

What's more, if only one lcdc is used, please select the corresponding lcdc at "make menuconfig" and do not attach another free lcdc to save power. For example, if we use lcdc0+rk610 to implement dual display, we should do such configuration:

```
<*>     rk3188 lcdc support
[ * ]     lcdc0 support
    [   ]     lcdc0 1.8V io support
    [   ]     lcdc1 support
    [   ]     lcdc1 1.8V io support
```

LCDC of RK3188/RK3168 support IO output of 1.8V, hardware designed for 1.8V IO should select corresponding 1.8V IO support.

Kernel later than linux3.10 has such setting in dts: rochchip, pwr18=<1>; this selection represents that hardware is designed for 1.8V voltage.

c. [ ] Three fb buffer support

We use 3 buffer to improve fluency, and it is necessary for projects in Android system higher than 4.1.

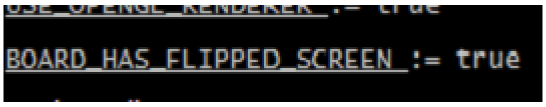
d. [ ] FB rotate support

In case that part of model and hardware were designed with mistakes leads to LCD screen display the wrong direction, FB rotate support interface can adjust the display direction. Other circumstances do not need this support. Method is shown below:

If some mistakes occur and screen display direction is wrong (rotate 180 or 90 degrees) while hardware cannot adjust (normal LCD screen has two pins U/D and L/R for direction adjustment), software can execute certain rotation, firstly, modify ro.sf.hwrotation in build.prop in order to adjust the screen display direction. When HDMI is launched, HDMI direction also displays the opposite direction because of the edition of ro.sf.hwrotation, then we should select FB rotate support in make menuconfig of kernel and input the value of ro.sf.hwrotation in rotate orientation. Caution: Rotation by this way will increase occupation of memory and decrease display fluency. Therefore, hardware adjustment is more recommended. This adjustment requires the screen with resolution ratio lower than 1280*800 and only suits for dual lcdc implementing dual display like rk30xx.

For linux 3.10 version kernel, set this option as: rockchip, mirror=<X_MIRROR> in dts, and set mirror as X_MIRROR/Y_MIRROR/X_Y_MIRROR for your own requirement.

After such adjustment, LCD screen still displays the opposite direction in recovery interface due to ro.sf.hwrotation does not work in recovery. In order to rotate 180° of recovery display, we should open the macro ro.sf.hwrotation below in device/rockchip/rkxxsdk/BoardConfig.mk.



```
BOARD_HAS_FLIPPED_SCREEN := true
```

Then execute the following orders, regenerating recovery.img:

(1) touch bootable/recovery/minui/graphics.c

(2) make out/target/product/rk30sdk/recovery.img

(3) ./mkimage.sh

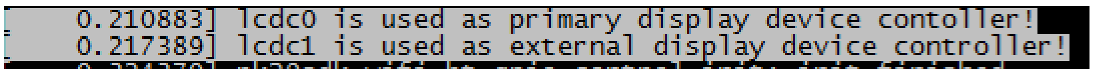
e. [ ] RockChip display transmitter support

This option is designed for showing configuration selections of switching chip, such as driver for LVDS/EDP/MIPI. If certain chip does not need programme to drive, this setting is not necessary.

HDMI configuration:

```
Device Drivers   --->
  Graphics support   --->
      [*] Rockchip HDMI support   --->
      --- Rockchip HDMI support
      [*]    CAT66121 HDMI support
      HDMI Source LCDC select (lcdc1)   --->
```

After selecting certain driver of HDMI, we need to select the corresponding LCDC: HDMI source. LCDC for HDMI is external LCDC and can be diagnosed and analyzed by logs from start.



```
[    0.210883] lcdc0 is used as primary display device contoller!
[    0.217389] lcdc1 is used as external display device controller!
```

This log illustrates that LCDC0 is used for primary display device, LCDC1 is used for extend display device, outputting display signal for HDMI.

Following are some references for developing process:

(1) Dual LCDC dual display, LCDC0 for LCD and LCDC1 for HDMI:

Configuration file: rk3066_sdk_defconfig

Board file: board-rk30-sdk.c

(2) Dual LCDC dual display, LCDC0 for HDMI and LCDC1 for LCD:

Configuration file: rk3188_ds1006h_defconfig

Board file: board-rk3188-ds1006h.c

(3) EDP HD screen, dual LCDC dual display, LCDC0 FOR LCD, LCDC1 for HDMI:

Configuration file: rk3188_LR097_defconfig

Board file: board-rk3168-LR097.c

(4) Singal LCDC+RK610 for dual display, LCDC1 for LCD:

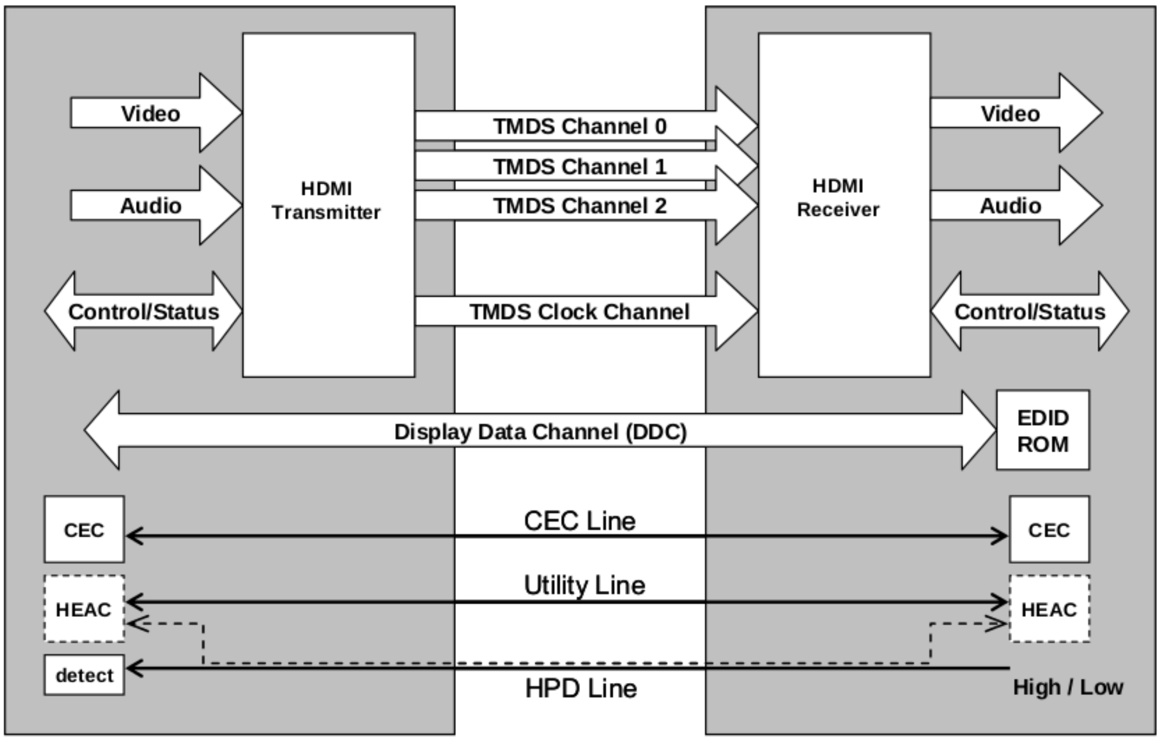
Configuration file: rk3168m_tb_defconfig

Board file: board-rk3168m-tb.c

3 HDMI

3.1 HDMI Introduction

HDMI，which full name is High Definition Multimedia Interface, is a digital image and sound transmission interface which can transmit uncompressed audio and video signal. HDMI is applied to set-top boxes, DVD players, PC, TV game equipment, comprehensive expansion machine, digital stereo, digital TV and other equipments. HDMI is composed mainly of "HDMI cable", "HDMI source" and "HDMI sink", as shown

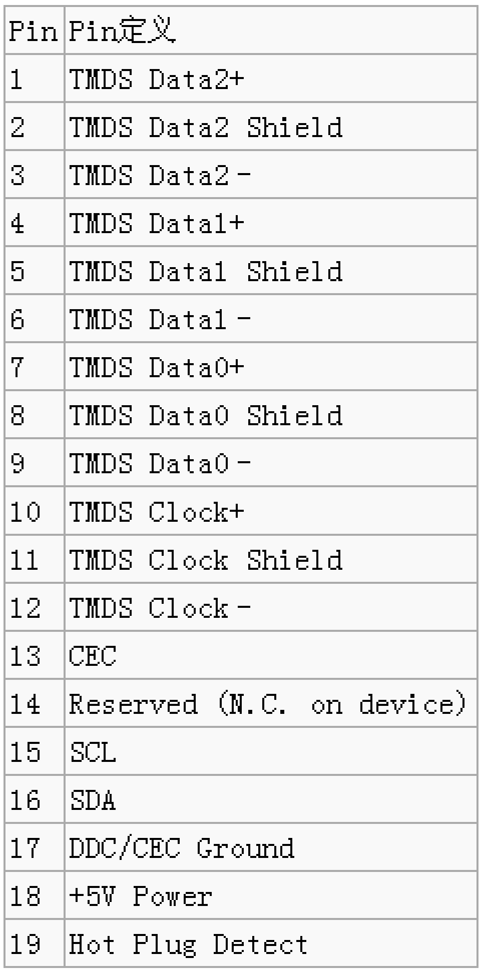


in Figure 1-1:

Figure 1-1

(1) Pin Definition

HDMI is used to transmit audio and video signal by TMDS (Transition Minimized Differential Signal) technology, and has 4 types of connectors which are the standard type A HDMI connector composed of 19 pins, the standard type B HDMI connector composed of 29 pins, and supporting higher resolution and tramsmiting twice TMDS data as type A, and type C and Type D HDMI connectors for other application.The pin definition of the standard type A connector is shown in Figure2-1:

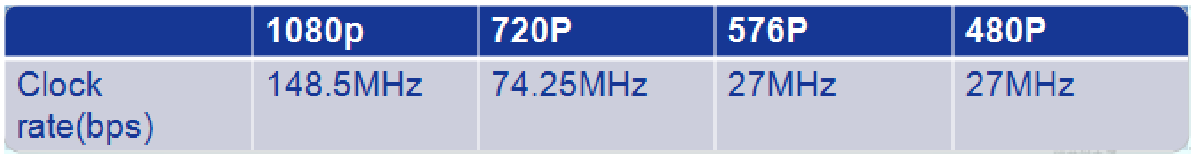


Pin	Pin定义
1	TMDS Data2+
2	TMDS Data2 Shield
3	TMDS Data2－
4	TMDS Data1+
5	TMDS Data1 Shield
6	TMDS Data1－
7	TMDS Data0+
8	TMDS Data0 Shield
9	TMDS Data0－
10	TMDS Clock+
11	TMDS Clock Shield
12	TMDS Clock－
13	CEC
14	Reserved (N.C. on device)
15	SCL
16	SDA
17	DDC/CEC Ground
18	+5V Power
19	Hot Plug Detect

Figure 2-1

(2) TMDS

TMDS which full name is Transition Minimized Differential Signaling, used to transmit audio,video and all kinds of secondary data for HDMI, can support several pixel encoding such as RGB 4:4:4, YCbCr 4:4:4 (8-16 bits per component) and YCbCr 4:2:2 (12 bits per component). Diffrent resolution is given corresponding to different TMDS clock frequency, as shown in Figure 2-2:



	1080p	720P	576P	480P
Clock rate(bps)	148.5MHz	74.25MHz	27MHz	27MHz

Figure 2-2

(3) DDC

DDC which full name is Display Data Channel, used to inform the sender and receiver of transmitting and reception capacity. However HDMI is only used to obtain reception capacity unidirectionally, which means HDMI can read EDID data directly from receiver such as display screen. EDID which full name is Extended Display Identification Data, a VESA standard data format, provides a general description of almost all display parameters. The signal on the pins of SCL and SDA in DDC is a kind of I2C sigal which frequency is 100KHz.

(4) CEC

CEC which full name is Consumer Electronics Control, used for remote control of consumer electronics products.

(5) HPD

Hot Plug, interrupt detection pin of HDMI, is used to determine whether HDMI insert or pull out.

3.2 Framework and Implementation of HDMI Driver

This section describes the source code distribution framework of the system, including directory structure, function of source file, and software process flow and configuration instructions.

3.2.1 Framework of HDMI Codes

(1) Directory Structure

Directory Name	**Commentary**	**List of Source Files**
drivers/video/rockchip/hdmi/	Related codes to implement HDMI framework	rk_hdmi.h rk_hdmi_task.c rk_hdmi_lcdc.c rk_hdmi_sysfs.c rk_hdmi_edid.c
drivers/video/rockchip/hdmi/chips/rk3288/ drivers/video/rockchip/hdmi/chips/rk616/ drivers/video/rockchip/hdmi/chips/cat66121/ drivers/video/rockchip/hdmi/chips/rk30/	HDMI chip driver codes	For rk3288: Rk3288_hdmi.c Rk3288_hdmi.h rk3288_hdmi_hw.c rk3288_hdmi_hw.h

(2) Description of Source Files

Source Files	**Description**
HDMI Module	
rk_hdmi.h	Related structure definition
rk_hdmi_task.c	Implementation of HDMI framework and driver processing flow
rk_hdmi_lcdc.c	Implementation of associated interfaces of LCDC
rk_hdmi_sys.c	The code implementation of HDMI Sysfs debugging node

rk_hdmi_edid.c	The related functions of EDID data analysis
rkxxx_hdmi.c	Implementation of interfaces insert, remove and shutdown, registering HDMI structure to the core layer
rkxxx_hdmi.h	Related structure definition
rkxxx_hdmi_hw.c	The HMDI function implementation of detecting hot plug, reading EDID data, configuration of audio and video parameters, system output and sleeping
rkxxx_hdmi_hw.h	Each register value and structure of HDMI, as well as the external function declarations

3.2.2 HDMI Software Flow

HDMI driver framework mainly implement hot plug detection, EDID data reading and analysis, configuration of audio and video parameters, system output and sleeping, etc. The main process flow is shown in Figure 3-1:

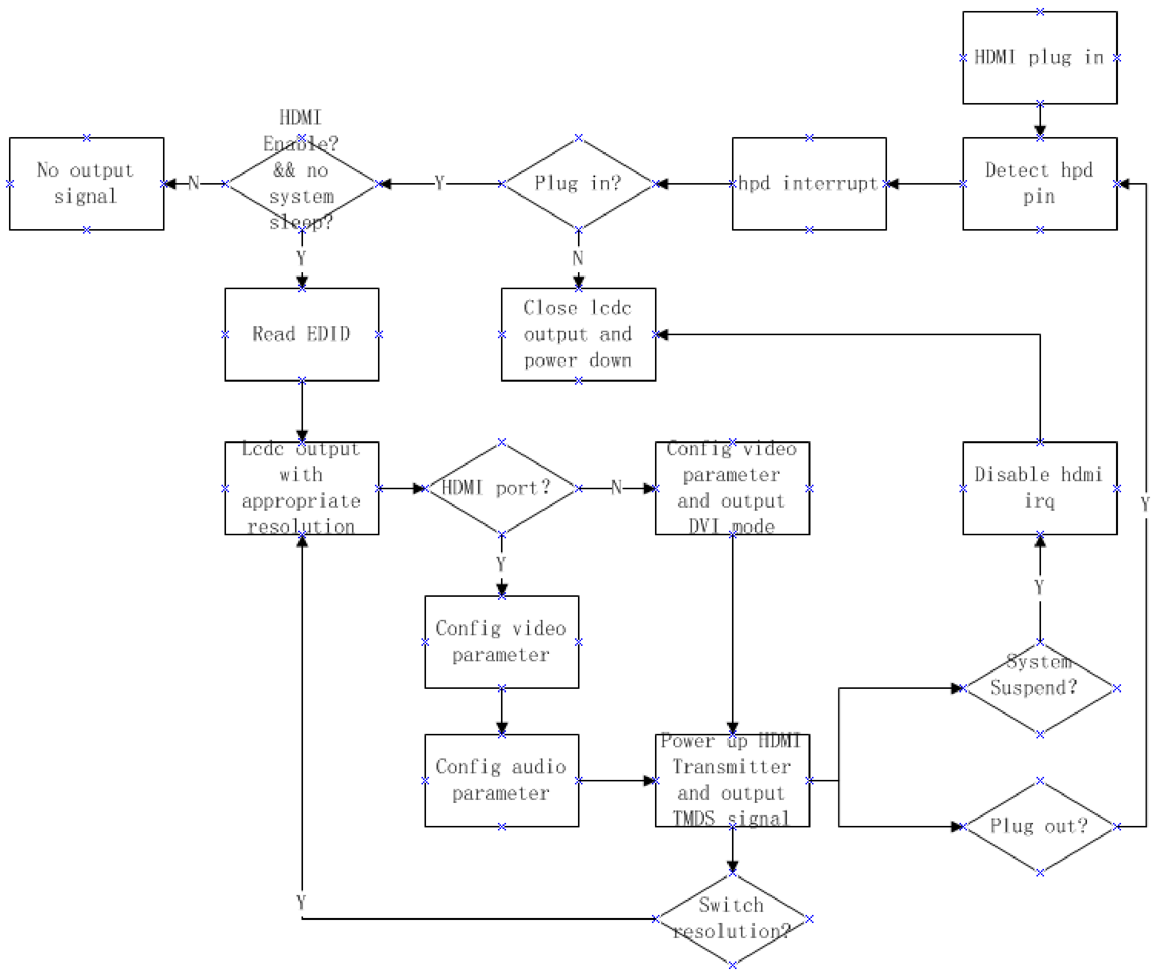

HDMI plug in
Detect hpd pin
hpd interrupt
Plug in?
HDMI Enable? && no system sleep?
No output signal
Y
N
Read EDID
Close lcdc output and power down
Lcdc output with appropriate resolution
HDMI port?
Config video parameter and output DVI mode
Disable hdmi irq
Config video parameter
Config audio parameter
Power up HDMI Transmitter and output TMDS signal
System Suspend?
Plug out?
Switch resolution?

Figure 3-1

3.3 Associated Configuration of HDMI

3.3.1 Linux 3.0.36 Kernel Configuration

(1) MAKE MENUCONFIG Configuration

```
Device Drivers   --->

        Graphics support   --->

           [*] Rockchip HDMI support   --->

           --- Rockchip HDMI support

           [ ]    CAT66121 HDMI support
           [*]    RK616 HDMI support
           [*]    RK616 HDCP support
                  HDMI Source LCDC select (lcdc1)   --->
           [ ]    Rockchip HDMI Debugging
           -*-    Mute Codec When HDMI Actived
```

(a) HDMI Chip Driver Selection

If using the chip ITE66121, select the option "CAT66121 HDMI support". If using the chip RK616, select the option "RK616 HDMI support". If using the chip RK610, select the option "RK610 HDMI support ".

(b) [ ] RK616 HDCP Support

This option is used to configure whether HDCP function is available, and HDCP function depends on HDMI. If you want to make RK616 HDCP function available, please select the option "K616 HDCP support".

(c) HDMI Source LCDC Select ( )

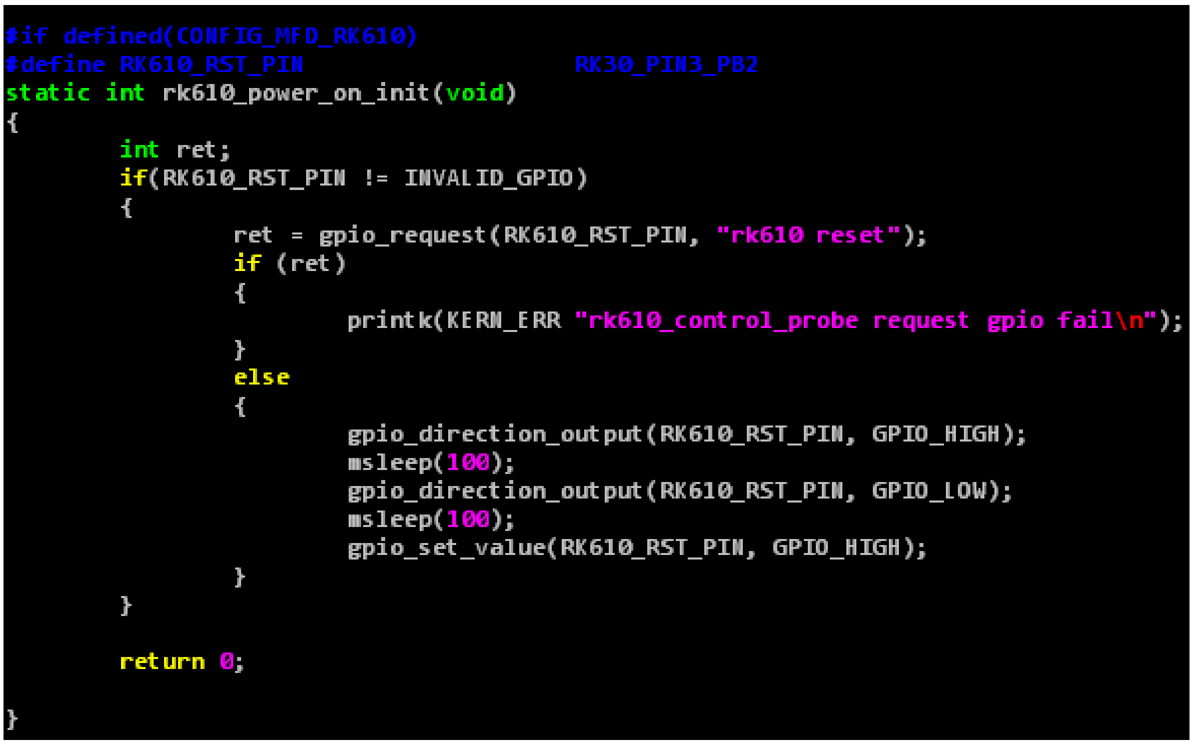
This option is used to configure which LCDC in master chip is connected to HDMI chip, which is configured according to the actual circuit connection.

(d) [ ] Rockchip HDMI Debugging

This option is for debugging switch, generally does not need to open.

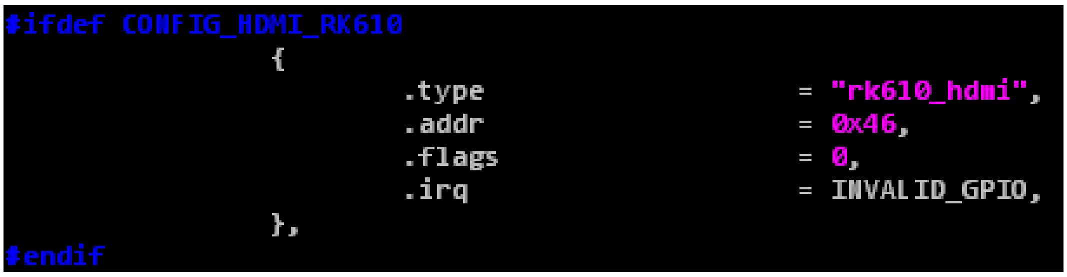
(2) Board File Configuration

HDMI chip RESET pin and INT pin are configured in the board file, the following figure is RK610 RESET pin configuration (using GPIO3_B2 as RK610 RESET pin), please configure related pins according to hardware connection.



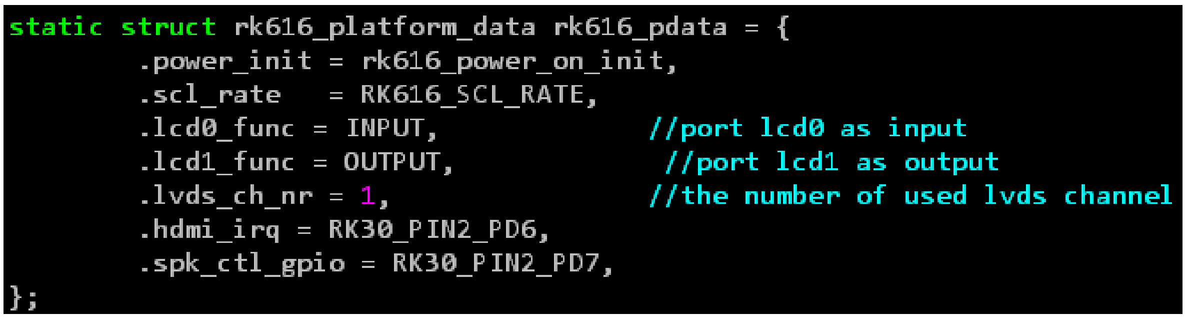
```
#if defined(CONFIG_MFD_RK610)
#define RK610_RST_PIN                   RK30_PIN3_PB2
static int rk610_power_on_init(void)
{
        int ret;
        if(RK610_RST_PIN != INVALID_GPIO)
        {
                ret = gpio_request(RK610_RST_PIN, "rk610 reset");
                if (ret)
                {
                        printk(KERN_ERR "rk610_control_probe request gpio fail\n");
                }
                else
                {
                        gpio_direction_output(RK610_RST_PIN, GPIO_HIGH);
                        msleep(100);
                        gpio_direction_output(RK610_RST_PIN, GPIO_LOW);
                        msleep(100);
                        gpio_set_value(RK610_RST_PIN, GPIO_HIGH);
                }
        }

        return 0;

}
```

The following figure is RK610 interrupt pin configuration.If use GPIO port as the interrupt pin, please configure IRQ into the corresponding GPIO port. If the INT pin does not have any hardware connection, please configure IRQ into INVALID_GPIO.



```
#ifdef CONFIG_HDMI_RK610
                {
                        .type                   = "rk610_hdmi",
                        .addr                   = 0x46,
                        .flags                  = 0,
                        .irq                    = INVALID_GPIO,
                },
#endif
```

If you use RK616 (or RK618) chip, you need to configure interfaces lcd0 and lcd1 of RK616 (or RK618) in the board-xxx.c file. If using lcd0 (or lcd1) as an input port, configure port as INPUT. If using lcd1 as an output port, configure port as OUTPUT. If not used, configure port as UNUSED, please configure according to hardware connection. The following figure shows that if lcd0 port of RK616 is configured as an input port, lcd1 port configured as an output port and INT connecting to the pin GPIO2_D7. Please configure .lcd0_func = INPUT, .lcd1_func = OUTPUT, .hdmi_irq = RK30_PIN2_PD6:



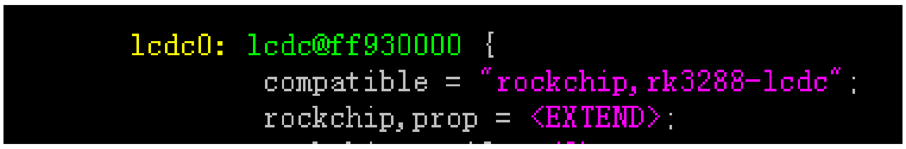
```
static struct rk616_platform_data rk616_pdata = {
        .power_init = rk616_power_on_init,
        .scl_rate   = RK616_SCL_RATE,
        .lcd0_func = INPUT,                  //port lcd0 as input
        .lcd1_func = OUTPUT,                  //port lcd1 as output
        .lvds_ch_nr = 1,                     //the number of used lvds channel
        .hdmi_irq = RK30_PIN2_PD6,
        .spk_ctl_gpio = RK30_PIN2_PD7,
};
```

3.3.2 Linux3.10 Kernel Configuration

For Linux3.10 later versions of kernel, the board file is no longer used, so the related hardware configuration parameters are passed through the file dts.

(1) HDMI Video Source Select

For RK3288, HDMI selects LCDC0 as the default input source, as shown in the following figure:

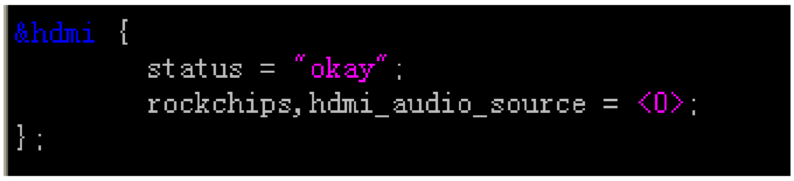


```
        lcdc0: lcdc@ff930000 {
                compatible = "rockchip,rk3288-lcdc";
                rockchip,prop = <EXTEND>;
```

The LCDC with property value "<EXTEND>" is used to be configured as HDMI input source. If you need to use LCDC1 as a HDMI source, modify the value of property named "rockchip, prop" as "<EXTEND>" in lcdc1 node, and modify the value of property named "rockchip, prop" as "<PRMRY>" in lcdc0 node in the dtsi file.

(2) HDMI Audio Source Select

I2S interface is selected as default audio input interface.



```
&hdmi {
        status = "okay";
        rockchips,hdmi_audio_source = <0>;
};
```

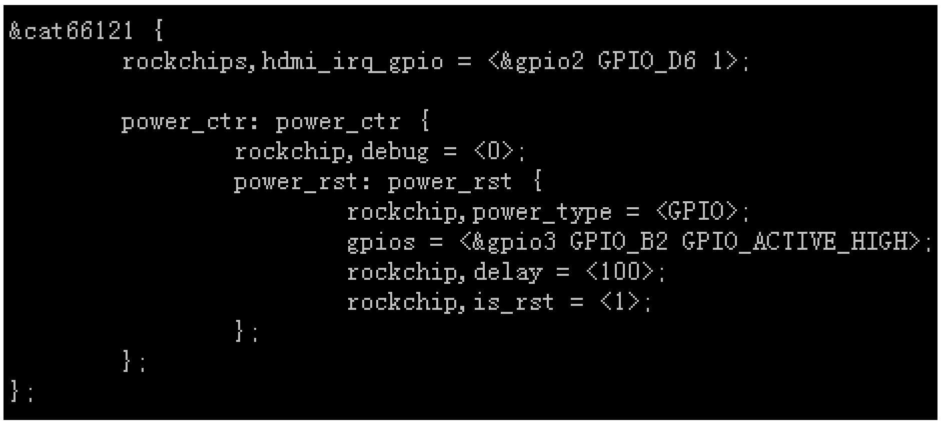
Choose I2S as audio input interface: you can configure the HDMI node like "rockchip, hdmi_audio_source = <0>;", and at the same time you should configure the file "XX_defconfig" like "CONFIG_SND_RK_SOC_I2S=y" and

"CONFIG_SND_RK_SOC_HDMI_I2S=y".

Choose SPDIF as audio input interface: you can configure the HDMI node like "rockchip, hdmi_audio_source = <1>;", and at the same time you should configure the file "XX_defconfig" like "CONFIG_SND_RK_SOC_SPDIF=y"and "CONFIG_SND_RK_SOC_HDMI_SPDIF =y".

(3) External HDMI Chip Configuration

If master control chip has no HDMI interface, you can use external HDMI chip such as chip RK616, CAT66121 and so on, and you have to configure interrupt corresposnding GPIO port and the pin “power ctrl”. If the pin GPIO2_D6 is configured as interrupt pin, the pin GPIO3_B2 should be configured as Reset pin. The configuration is shown in the following figure:



```
&cat66121 {
        rockchips,hdmi_irq_gpio = <&gpio2 GPIO_D6 1>;

        power_ctr: power_ctr {
                rockchip,debug = <0>;
                power_rst: power_rst {
                        rockchip,power_type = <GPIO>;
                        gpios = <&gpio3 GPIO_B2 GPIO_ACTIVE_HIGH>;
                        rockchip,delay = <100>;
                        rockchip,is_rst = <1>;
                };
        };
};
```

3.4 HDMI Debugging Node Description

Several major properties of HDMI are listed in the sys/class/display/HDMI node:

(1) connect: HDMI or DVI is connected or not, 1 is connected, and 0 is not connected.

(2) enable: HDMI is available or not, 1 is available, and 0 is not available.

(3) mode: current setting resolution, for example by executing the command “echo 1920x1080p-60 > mode”, the resolution is modified to 1080p and the screen

refresh rate is 60Hz.

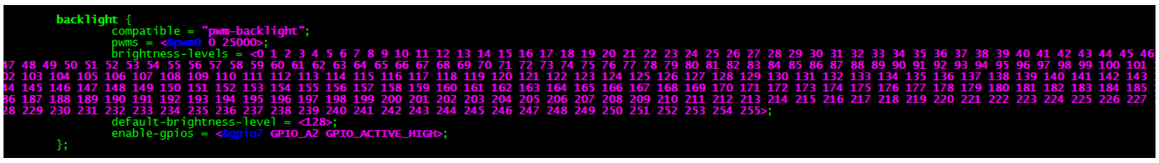
(4) modes: list all resolutions supported by both Sink side and Source side.

(5) scale: horizontal and vertical scaling values, xscale and yscale are equal to100 by default.

4 Backlight Control

In linux3.10 version kernel, the backlight driver file directory is drivers/video/backlight/pwm_bl.c.

The related control parameters are passed from the file dts in following figure:



```
        backlight {
                compatible = "pwm-backlight";
                pwms = <&pwm0 0 25000>;
                brightness-levels = <0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101
02 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
44 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
86 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
28 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255>;
                default-brightness-level = <128>;
                enable-gpios = <&gpio7 GPIO_A2 GPIO_ACTIVE_HIGH>;
        };
```

Pwms = <&pwm0 0 25000>: backlight pwm control port definition, 25000 is pwm cycle, unit ns.

Brighness-levels: specific backlight brightness list, use positive sequence or reverse order depending on whether the hardware design of backlight is positive polarity or negative polarity.

default-brighness-level: default backlight brightness.

enable-gpios: backlight power control enabling IO.

Appendix A LVDS,EDP and MIPI display interface conversion chips and their drivers

There are about four types of screen that we are using distinguished by their interfaces.That is RGB,LVDS,EDP and MIPI. Generally speaking,small size screen(7 inches below) uses RGB interface.Big size and high definition screen uses LVDS,EDP or MIPI interface.

For screen with RGB interface:LCDC can transfer signal to the screen directly,without any conversion.

For screen with LVDS interface:as RK30XX LCDC outputs RGB signal,so you need to convert RGB signal to LVDS signal,this conversion is implemented by external LVDS chip.RK292X and RK302X have LVDS modules inside,they can not only output LVDS signal,but also output RGB signal by bypass function.

Some of the LVDS conversion chips do not need drivers.They can run when they are powered on.For example TI.

LVDS conversion chips like SN75LVDS83A series. If you use them on RK30XX platform,you don't need to make special configuration in your codes,just like using RGB screens.For some other LVDS chips, you need to make special configuration in your codes when you use them on RK30XX platform,for example LVDS conversion chip that integrated in RK61X.

On our SDK platform,there are three LVDS chips that need drivers.That is RK61X and the inside LVDS modules of RK292X and RK302X,so you need to make specified configuration in your codes before you use them.

RK610 is a multifunction device,before you use it,follow configuration as below:

```
Device Driver
[*] Multifunction device drivers   --->
```

```
        [*]    RK610(Jetta) Multimedia support s   --->


Device Driver
   Graphics support   --->
     <*>Frame buffer support for Rockchip --->
       [*]    RockChip display transmitter support   --->
         ---RockChip display transmitter support
             [*]   RK610(Jetta) lvds transmitter support
             [ ]   RGB to Display Port transmitter anx6345,anx9804,anx9805 support
             [ ]RockChip MIPI DSI Support
```

On RK2928 platform, follow configuration as below:

```
Device Drivers   --->
         Graphics support   --->
         <*>Frame buffer support for Rockchip --->
               [*] RK_LVDS support
```

Our SDK supports four kinds of EDP conversion chips.ANX9804、ANX9805、ANX6345 and DP501. ANX series of chips among them are compatible, follow configuration as below:

```
Device Driver
   Graphics support   --->
        Display device support   --->
[*]     RockChip display transmitter support   --->
[*]     RGB to Display Port transmitter anx6345,anx9804,anx9805
[ ]    RGB to Display Port transmitter dp501 support (NEW)
```

Appendix B Usage of MIPI screen

On our RK platform,we support three types of MIPI driver IC, SSD2828、TC358768 and RK618.

B-1 Related codes

the related driver code lists are below:

drivers/video/rockchip/transmitter/mipi_dsi.c

drivers/video/rockchip/transmitter/mipi_dsi.h

drivers/video/rockchip/transmitter/ssd2828.c

drivers/video/rockchip/transmitter/tc358768.c

drivers/video/rockchip/transmitter/rk616_mipi_dsi.c

drivers/video/rockchip/transmitter/rk616_mipi_dsi.h

drivers/video/rockchip/screen/lcd_LD089WU1_mipi.c

B-2 Kernel configuration

if you use screen with MIPI interface, follow your kernel configuration as below:

```
Device Drivers   --->
    Graphics support   --->
    [*]   RockChip display transmitter support   --->
          [*]   RockChip display transmitter support   --->
              [*]   Rockchip MIPI DSI support
```

check the MIPI chip Model that you use:

```
    < >      toshiba TC358768 RGB to MIPI DSI
    < >      solomon SSD2828 RGB to MIPI DSI
    < >      RK616(JettaB) mipi dsi support
```

B-3 Screen configuration file

Please refer to "drivers/video/rockchip/screen/lcd_LD089WU1_mipi.c"

```
#include "../transmitter/mipi_dsi.h"

#define SCREEN_TYPE   SCREEN_MIPI
```

- If SCREEN_TYPE is not defined as SCREEN_MIPI,MIPI's related drivers won't execute.

```
#define MIPI_DSI_LANE 4
```

MIPI_DSI_LANE is the data lane number of LCD,it must match your LCD hardware. Now most of the high definition LCD has 4 lanes, you don't need to modify it defaultly.

```
#define MIPI_DSI_HS_CLK 1000*1000000          //DSI_HS_CLK
```

MIPI_DSI_HS_CLK is the transmission rate of each lane(DDR_Dual-Edge),the minimum rate is 80Mbps,the max rate is 1Gbps(default).Unless there is something wrong with your PCB board wiring that it could not run in 1Gbps rate,otherwise you don't need to modify it basically.

If the MIPI screen itself needs to be initialized,you need to define RK_SCREEN_INIT.

```
#define RK_SCREEN_INIT   1
```

And implement the following two functions:

```


int rk_lcd_init(void) {
    u8 dcs[16] = {0};
    if(dsi_is_active() != 1)
        return -1;
    /*below is changeable*/
    msleep(50);
    dsi_enable_hs_clk(1);

```

```
        dcs[0] = LPDT;

        dcs[1] = dcs_exit_sleep_mode;

        dsi_send_dcs_packet(dcs, 2);

        msleep(1);

        dcs[0] = LPDT;

        dcs[1] = dcs_set_display_on;

        dsi_send_dcs_packet(dcs, 2);

        msleep(10);

        dsi_enable_video_mode(1);

}


int rk_lcd_standby(u8 enable) {


    u8 dcs[16] = {0};

    if(dsi_is_active() != 1)

        return -1;


    if(enable) {


        /*below is changeable*/

        dcs[0] = LPDT;

        dcs[1] = dcs_set_display_off;

        dsi_send_dcs_packet(dcs, 2);

        msleep(1);

        dcs[0] = LPDT;

        dcs[1] = dcs_enter_sleep_mode;
        dsi_send_dcs_packet(dcs, 2);

        msleep(1);



    } else {

        /*below is changeable*/

        rk_lcd_init();

    }
```

Instruction:

1, dsi_is_active() is used to judge whether DSI is in a working state.If yes, 1 is returned.If no,0 is returned.Otherwise if there is an error, -1 is returned.

2,

```
dcs[0] = LPDT;

dcs[1] = dcs_enter_sleep_mode;

dsi_send_dcs_packet(dcs, 2);
```

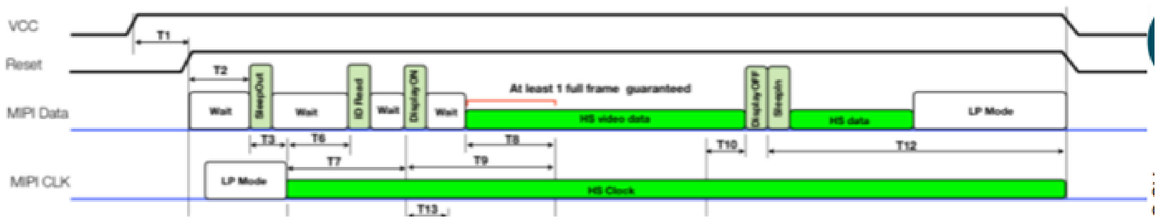
MIPI screen's initialization uses DSI to transmit data,we implement the function interface dsi_send_dcs_packet() as needed.For DCS Array,dcs[0] is used to store write mode,dcs[1] is used to store DCS command,the other members of dcs are used to store DCS command parameters.

the codes above are used to send "dcs_enter_sleep_mode" command.These DCS commands are defined in "drivers/video/rockchip/transmitter/mipi_dsi.h".For detailed meaning,please refer to MIPI DCS Specification.Certainly,the customer’s screen commands can also be sended by this interface.

3.

```
dsi_enable_hs_clk(1);    //enable highspeed clock
```

When you are implementing LCD initialization function.You must operate LCD based on its initilization sequenze,As below:



```
dsi_enable_hs_clk(1);
    dcs[0] = LPDT;
    dcs[1] = dcs_exit_sleep_mode;
    dsi_send_dcs_packet(dcs, 2);
    msleep(1);
    dcs[0] = LPDT;
    dcs[1] = dcs_set_display_on;
    dsi_send_dcs_packet(dcs, 2);
    msleep(10);
    dsi_enable_video_mode(1);
```

4.When you see this comment:/*below is changeable*/,you can initialize LCD based on yourself requirement,including operations like LCD wakeup,LCD reset and so on.

Most of the MIPI screens can meet #define MIPI_DSI_LANE 4,#define MIPI_DSI_HS_CLK 1000*1000000,and do not need to be special initialized(just sending "sleep out" and "display on" to LCD is enough,or nothing).When you are implementing LCD drivers for MIPI screens,set OUT_TYPE as SCREEN_MIPI is enough.

5.Coufiguration for the board level is the same with general LCD,no special process is needed.

Appendix C Usage of RK32XX MIPI screen

RK32XX has the internal dual-channel MIPI control IC.

C-1 Related codes

the related driver code lists are below:

drivers/video/rockchip/transmitter/mipi_dsi.c

drivers/video/rockchip/transmitter/mipi_dsi.h

drivers/video/rockchip/transmitter/rk32_mipi_dsi.c

drivers/video/rockchip/transmitter/rk32_mipi_dsi.h

drivers/video/rockchip/screen/ lcd_mipi.c

C-2 kernel configuration

Screen with MIPI interface,configure LCD as below:

```
Device Drivers  --->
    Graphics support  --->
[*]   rk3288 lcdc support
      LCD Panel Select (General lcd panel) --->
( ) General lcd panel
(*) rk mipi dsi lcd
```

configure display transmitter as below:

```
Device Drivers  --->
    Graphics support  --->
    [*]   RockChip display transmitter support  --->
          [*]   Rockchip MIPI DSI support
                <*>    rk32 mipi dsi support
```

C-3 Screen configuration files

DTS configuration file of dual-channel MIPI screen,please refer to "arch/arm/boot/dts/lcd-wqxga-mipi.dtsi"

DTS configuration file of single-channel MIPI screen,please refer to "arch/arm/boot/dts/lcd-ld089wu1-mipi.dtsi"

Take "arch/arm/boot/dts/lcd-wqxga-mipi.dtsi" as an example,

```
/* about mipi */
disp_mipi_init: mipi_dsi_init{
    compatible = "rockchip,mipi_dsi_init";
    rockchip,screen_init     = <1>;// If MIPI screen needs to be initialized,set the value as 1,otherwise set as 0
    rockchip,dsi_lane        = <4>;//define the number of lanes
    rockchip,dsi_hs_clk      = <940>;//set each MIPI lane's transmitting rate
    rockchip,mipi_dsi_num = <2>;//set the value as 2 for dual-channel MIPI and 1 for single-channel MIPI
};
disp_mipi_power_ctr: mipi_power_ctr {
    compatible = "rockchip,mipi_power_ctr";
//if MIPI screen has rst pin,you need to define mipi_lcd_rst
    mipi_lcd_rst:mipi_lcd_rst{
            compatible = "rockchip,lcd_rst";
            rockchip,gpios = <&gpio7 GPIO_A4 GPIO_ACTIVE_HIGH>;//set MIPI screen's rst pin
            rockchip,delay = <10>;//set rst's delay time,the unit is ms
    };
    // if MIPI screen has en pin,you need to define mipi_lcd_en
    mipi_lcd_en:mipi_lcd_en {
            compatible = "rockchip,lcd_en";
            rockchip,gpios  =  <&gpio7  GPIO_A3  GPIO_ACTIVE_HIGH>;//set  MIPI
screen's enable pin
            rockchip,delay = <10>;//set enable's delay time,the unit is ms
    };
};
// If MIPI screen needs to be initialized,you need to define mipi_lcd_en
disp_mipi_init_cmds: screen-on-cmds {
    rockchip,cmd_debug = <0>;
    compatible = "rockchip,screen-on-cmds";
            rockchip,on-cmds1 {
            compatible = "rockchip,on-cmds";
            rockchip,cmd_type = <LPDT>;//command type，LPDT or HSDT
            rockchip,dsi_id = <2>;//0：send command only through mipi0
                                  //1: send command only through mipi1
                                  //2: send command through both mipi0 and mipi1
            rockchip,cmd = <0x05 0x01>; //command to be sended
            rockchip,cmd_delay = <10>;//delay time after sending the command
    };
    ……
    rockchip,on-cmds19 {
            compatible = "rockchip,on-cmds";
            rockchip,cmd_type = <LPDT>;
            rockchip,dsi_id = <2>;
            rockchip,cmd = <0x05 dcs_set_display_on>;
            rockchip,cmd_delay = <10>;
    };
```

```
};
//define MIPI screen's timing
disp_timings: display-timings {
        native-mode = <&timing0>;
        compatible = "rockchip,display-timings";
        timing0: timing0 {
                screen-type = <SCREEN_DUAL_MIPI>;//dual-channel MIPI screen
uses SCREEN_DUAL_MIPI,single-channel MIPI screen uses SCREEN_MIPI
                lvds-format = <LVDS_8BIT_2>;//Regardless
                out-face    = <OUT_P888>;//set pixel format
                clock-frequency = <265000000>;//set dclk，the unit is HZ
                hactive = <2560>;
                vactive = <1600>;
                hsync-len = <38>;
                hback-porch = <40>;
                hfront-porch = <108>;
                vsync-len = <4>;
                vback-porch = <4>;
                vfront-porch = <12>;
                hsync-active = <0>;
                vsync-active = <0>;
                de-active = <0>;
                pixelclk-active = <0>;
                swap-rb = <0>;
                swap-rg = <0>;
                swap-gb = <0>;
```

```
            };
        };
```

C-4 Board-level configure file

board-level configure file for MIPI control IC is located in "arch/arm/boot/dts/rk3288.dtsi".

You only need to enable dsi host0 for single-channel MIPI screen.For dual-channel MIPI screen, dsi host1 also need to be enabled.

```
dsihost0: mipi@ff960000{
        compatible = "rockchip,rk32-dsi";
        rockchip,prop = <0>;
        reg = <0xff960000 0x4000>;
        interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clk_gates5 15>, <&clk_gates16 4> , <&pd_mipidsi>;
        clock-names = "clk_mipi_24m", "pclk_mipi_dsi", "pd_mipi_dsi";
        status = "okay";
    };
    dsihost1: mipi@ff964000{
        compatible = "rockchip,rk32-dsi";
        rockchip,prop = <1>;
        reg = <0xff964000 0x4000>;
        interrupts = <GIC_SPI 20 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clk_gates5 15>, <&clk_gates16 5>, <&pd_mipidsi>;
        clock-names = "clk_mipi_24m", "pclk_mipi_dsi", "pd_mipi_dsi";
    status = "okay";//set as okay for dual-channel MIPI screen,and disabled for
 single-channel MIPI screen.
    };
```

C-5 Initialization codes for MIPI screen

If the MIPI screen itself needs to be initialized,you need to set " rockchip,screen_init" as 1 in dtsi file.

Initialization codes are in the file "lcd_mipi.c",here is each function's description:

1. void rk_mipi_screen_pwr_disable(struct mipi_screen *screen)

   Description:power off MIPI screen by controlling rst pin and en pin
2. void rk_mipi_screen_pwr_enable(struct mipi_screen *screen)

   Description:power on MIPI screen by controlling rst pin and en pin
3. void rk_mipi_screen_cmd_init(struct mipi_screen *screen)

   Description:If the MIPI screen needs to be initialized,this function will be used to analyze initialization command sequenze in the dtsi file,and then initialize it.
4. int rk_mipi_screen_init_dt(struct mipi_screen *screen)

   Description:implement analysing dtsi file of MIPI screen
5. int rk_mipi_screen_standby(u8 enable)

   Description:enable or disable MIPI screen. enable=0 means open,enable=1 means close.

Appendix D Quick judge on whether there is a reverse of RGB color

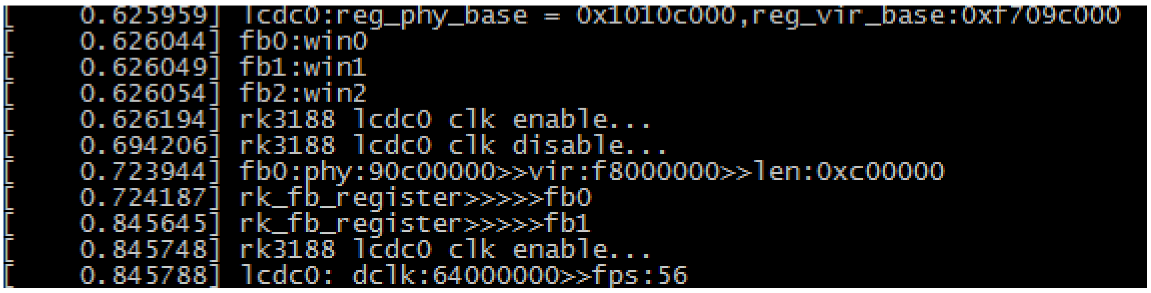
On some projects,because there is something wrong with LCD screecn itself or other hardware connections,so that RGB color is reverse.Such as R and G is reverse,or R and B is reverse,or B and G is reverse.And at last display color is abnormal.In this situation,we can correct it by mdoifying software configuration.

First,judge which two colors are reverse.By IO commands that comes with RK system,writing single red（0x00ff0000）,green（0x0000ff00）,blue （0x000000ff） to FB,and see related effects,then you can judge which two colors are reverse.Detail steps are below:

(1)run "stop" in your serial terminal(need root authority),then Android will stop screen refreshing which may influence testing.

(2)query LCDC register,fetch current FB address:

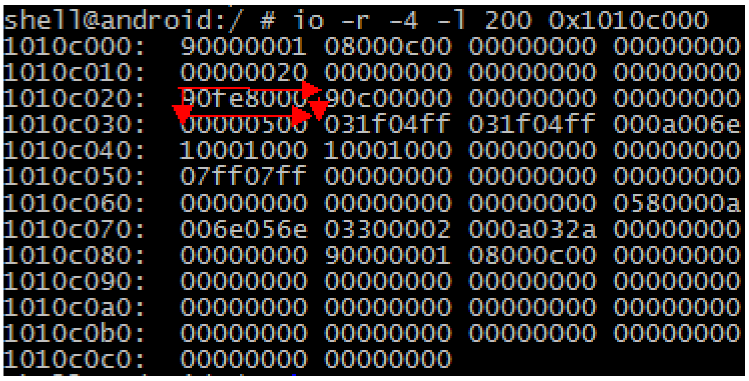
during the system booting,it will print LCDC register base address,for example,RK3188 system booting log is below:



```
[    0.625959] lcdc0:reg_phy_base = 0x1010c000,reg_vir_base:0xf709c000
[    0.626044] fb0:win0
[    0.626049] fb1:win1
[    0.626054] fb2:win2
[    0.626194] rk3188 lcdc0 clk enable...
[    0.694206] rk3188 lcdc0 clk disable...
[    0.723944] fb0:phy:90c00000>>vir:f8000000>>len:0xc00000
[    0.724187] rk_fb_register>>>>>fb0
[    0.845645] rk_fb_register>>>>>fb1
[    0.845748] rk3188 lcdc0 clk enable...
[    0.845788] lcdc0: dclk:64000000>>fps:56
```

According to the above log,we can see that LCDC0 register's physical address is 0x1010c000,virtual address is 0xf709c000,and FB0 matches LCDC0's Win0,its physical address is 0x90c00000,size is 0xc00000.As our system uses dual-buffer or tri-buffer strategy,FB current dispaly address will swith from 0x90c00000 to 0x90c00000 + 0xc00000 dynamically,that is（0x90c00000/0x90fe8000/0x913d0000）.

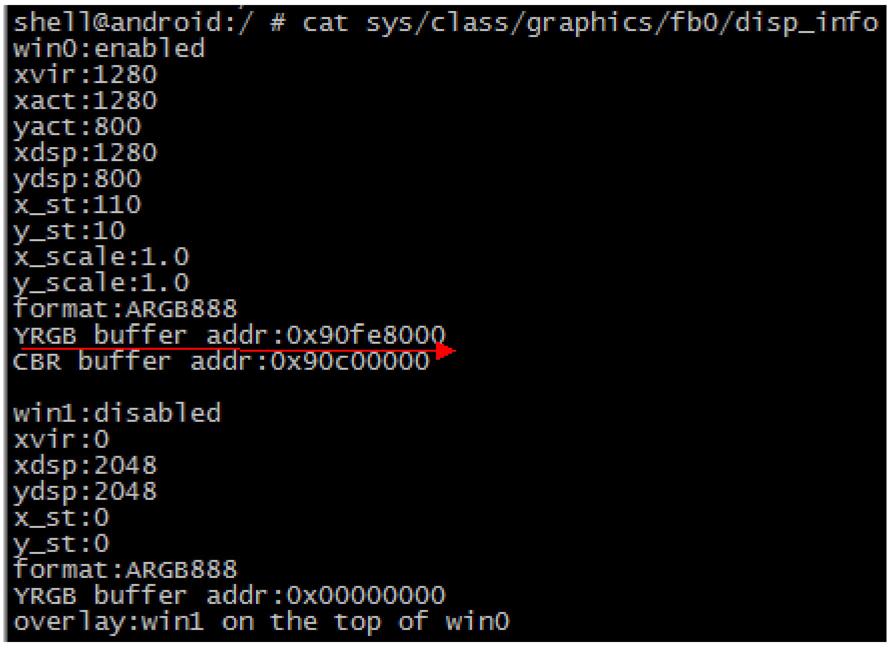
Then run "io -r -4 -l 200 0x1010c000" in serial terminal to read LCDC register’s configuration information.



```
shell@android:/ # io -r -4 -l 200 0x1010c000
1010c000:  90000001 08000c00 00000000 00000000
1010c010:  00000020 00000000 00000000 00000000
1010c020:  90fe8000 90c00000 00000000 00000000
1010c030:  00000500 031f04ff 031f04ff 000a006e
1010c040:  10001000 10001000 00000000 00000000
1010c050:  07ff07ff 00000000 00000000 00000000
1010c060:  00000000 00000000 00000000 0580000a
1010c070:  006e056e 03300002 000a032a 00000000
1010c080:  00000000 90000001 08000c00 00000000
1010c090:  00000000 00000000 00000000 00000000
1010c0a0:  00000000 00000000 00000000 00000000
1010c0b0:  00000000 00000000 00000000 00000000
1010c0c0:  00000000 00000000
```

According to RK3188 LCDC datasheet,we can know that WIN0 RGB data's base address register is 0x1010c020.we see the value stored in this register is 0x90fe8000. As FB0 matches LCDC's Win0,so 0x90fe8000 is the current FB's address.

The other way to print current FB's address is to read sys node:



```
shell@android:/ # cat sys/class/graphics/fb0/disp_info
win0:enabled
xvir:1280
xact:1280
yact:800
xdsp:1280
ydsp:800
x_st:110
y_st:10
x_scale:1.0
y_scale:1.0
format:ARGB888
YRGB buffer addr:0x90fe8000
CBR buffer addr:0x90c00000

win1:disabled
xvir:0
xdsp:2048
ydsp:2048
x_st:0
y_st:0
format:ARGB888
YRGB buffer addr:0x00000000
overlay:win1 on the top of win0
```

According to the above print information,Win0 RGB data's current FB address is 0x90fe8000.

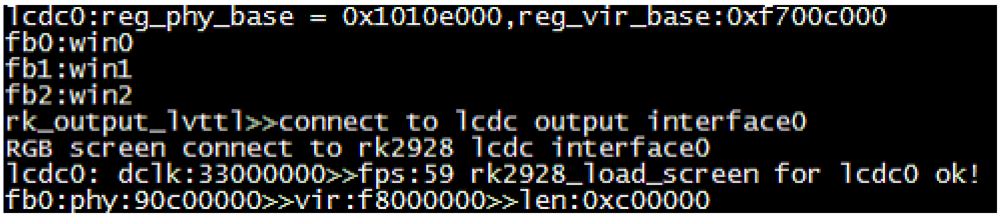
(3)Use IO commands to write RGB single color data in this address,and see corresponding display situation:

```
io -w -4 -l 0x3e8000 0x90fe8000 0x00ff0000       //RED
io -w -4 -l 0x3e8000 0x90fe8000 0x0000ff00       //GREEN
io -w -4 -l 0x3e8000 0x90fe8000 0x000000ff       //BLUE
```

Attention:the parameter behind -l is the data length that we write.Because we write full screen, the screen definition of the device we are using is 1280*800,so a frame of data's size is 1280*800*4 = 0x3e8000.

Normally,when you put in the above commands in turn,the screen will display red,green,and blue color in turn.If it displays wrong color,for example,you write red,but the screen displays blue,maybe R and B is reverse,you need to add"#define SWAP_RB   1 //switch R and B" in your screen driver codes.However if the screen displays neither red nor green or blue,then probably there is something wrong with the screen, hardware design,or the intermediate conversion chip such as LVDS,or LVDS_FORMAT setting isn'.

On RK2928 platform,the quick judgement way is similar,but RK2928 LCDC's base address is 0x1010e000,



```
lcdc0:reg_phy_base = 0x1010e000,reg_vir_base:0xf700c000
fb0:win0
fb1:win1
fb2:win2
rk_output_lvttl>>connect to lcdc output interface0
RGB screen connect to rk2928 lcdc interface0
lcdc0: dclk:33000000>>fps:59 rk2928_load_screen for lcdc0 ok!
fb0:phy:90c00000>>vir:f8000000>>len:0xc00000
```

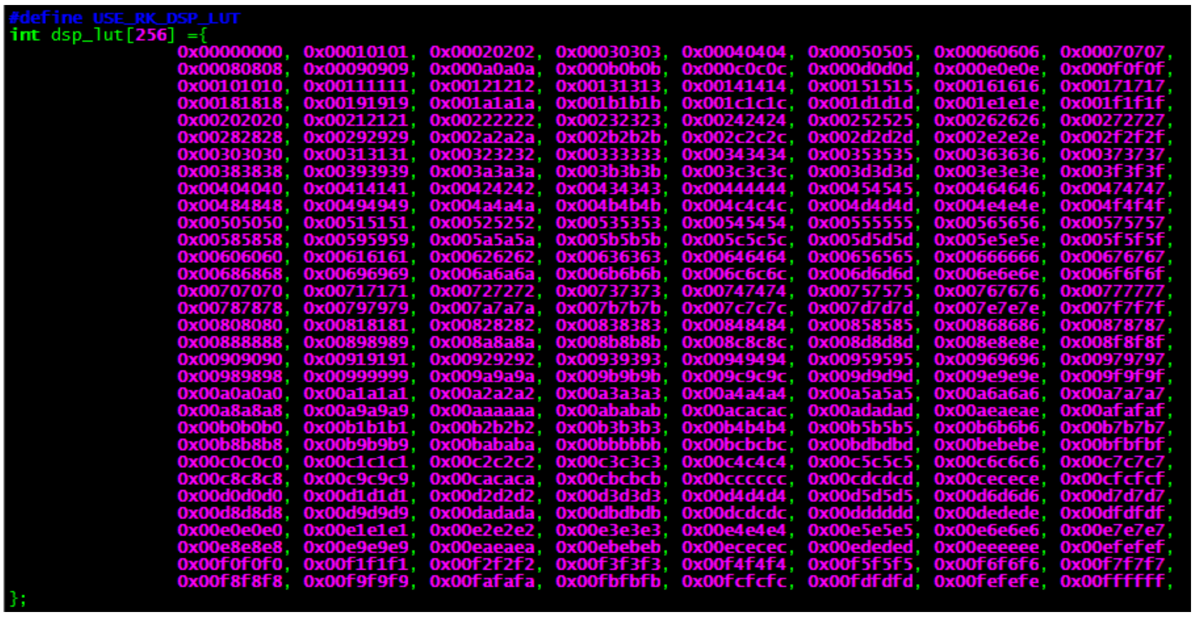
and FB0 matches LCD0's Win0,Win0 RGB data's base address register is 0x1010e01c.

Appendix E Use LUT function to improve the display quality

Because each kind of screen has its own gamma feature,when the same RGB data is transmitted to different screens,the display quality may be different.To make up for different screen gamma's inconsistencies,RK30XX and RK31XX(RK292X platform doesn't have LUT function) platforms provide LUT function that can adjust display quality,similar to gamma adjustment.Driver usage is below:

(1)First,add default LUT table to your screen's driver configuration file (lcd_xxx.c).This table is an ideal linear LUT table,you can copy it from kernel SDK.Then define macro USE_RK_DSP_LUT.

kernel/drivers/video/display/screen/lcd_b101ew05.c



```
#define USE_RK_DSP_LUT
int dsp_lut[256] ={
                 0x00000000, 0x00010101, 0x00020202, 0x00030303, 0x00040404, 0x00050505, 0x00060606, 0x00070707,
                 0x00080808, 0x00090909, 0x000a0a0a, 0x000b0b0b, 0x000c0c0c, 0x000d0d0d, 0x000e0e0e, 0x000f0f0f,
                 0x00101010, 0x00111111, 0x00121212, 0x00131313, 0x00141414, 0x00151515, 0x00161616, 0x00171717,
                 0x00181818, 0x00191919, 0x001a1a1a, 0x001b1b1b, 0x001c1c1c, 0x001d1d1d, 0x001e1e1e, 0x001f1f1f,
                 0x00202020, 0x00212121, 0x00222222, 0x00232323, 0x00242424, 0x00252525, 0x00262626, 0x00272727,
                 0x00282828, 0x00292929, 0x002a2a2a, 0x002b2b2b, 0x002c2c2c, 0x002d2d2d, 0x002e2e2e, 0x002f2f2f,
                 0x00303030, 0x00313131, 0x00323232, 0x00333333, 0x00343434, 0x00353535, 0x00363636, 0x00373737,
                 0x00383838, 0x00393939, 0x003a3a3a, 0x003b3b3b, 0x003c3c3c, 0x003d3d3d, 0x003e3e3e, 0x003f3f3f,
                 0x00404040, 0x00414141, 0x00424242, 0x00434343, 0x00444444, 0x00454545, 0x00464646, 0x00474747,
                 0x00484848, 0x00494949, 0x004a4a4a, 0x004b4b4b, 0x004c4c4c, 0x004d4d4d, 0x004e4e4e, 0x004f4f4f,
                 0x00505050, 0x00515151, 0x00525252, 0x00535353, 0x00545454, 0x00555555, 0x00565656, 0x00575757,
                 0x00585858, 0x00595959, 0x005a5a5a, 0x005b5b5b, 0x005c5c5c, 0x005d5d5d, 0x005e5e5e, 0x005f5f5f,
                 0x00606060, 0x00616161, 0x00626262, 0x00636363, 0x00646464, 0x00656565, 0x00666666, 0x00676767,
                 0x00686868, 0x00696969, 0x006a6a6a, 0x006b6b6b, 0x006c6c6c, 0x006d6d6d, 0x006e6e6e, 0x006f6f6f,
                 0x00707070, 0x00717171, 0x00727272, 0x00737373, 0x00747474, 0x00757575, 0x00767676, 0x00777777,
                 0x00787878, 0x00797979, 0x007a7a7a, 0x007b7b7b, 0x007c7c7c, 0x007d7d7d, 0x007e7e7e, 0x007f7f7f,
                 0x00808080, 0x00818181, 0x00828282, 0x00838383, 0x00848484, 0x00858585, 0x00868686, 0x00878787,
                 0x00888888, 0x00898989, 0x008a8a8a, 0x008b8b8b, 0x008c8c8c, 0x008d8d8d, 0x008e8e8e, 0x008f8f8f,
                 0x00909090, 0x00919191, 0x00929292, 0x00939393, 0x00949494, 0x00959595, 0x00969696, 0x00979797,
                 0x00989898, 0x00999999, 0x009a9a9a, 0x009b9b9b, 0x009c9c9c, 0x009d9d9d, 0x009e9e9e, 0x009f9f9f,
                 0x00a0a0a0, 0x00a1a1a1, 0x00a2a2a2, 0x00a3a3a3, 0x00a4a4a4, 0x00a5a5a5, 0x00a6a6a6, 0x00a7a7a7,
                 0x00a8a8a8, 0x00a9a9a9, 0x00aaaaaa, 0x00ababab, 0x00acacac, 0x00adadad, 0x00aeaeae, 0x00afafaf,
                 0x00b0b0b0, 0x00b1b1b1, 0x00b2b2b2, 0x00b3b3b3, 0x00b4b4b4, 0x00b5b5b5, 0x00b6b6b6, 0x00b7b7b7,
                 0x00b8b8b8, 0x00b9b9b9, 0x00bababa, 0x00bbbbbb, 0x00bcbcbc, 0x00bdbdbd, 0x00bebebe, 0x00bfbfbf,
                 0x00c0c0c0, 0x00c1c1c1, 0x00c2c2c2, 0x00c3c3c3, 0x00c4c4c4, 0x00c5c5c5, 0x00c6c6c6, 0x00c7c7c7,
                 0x00c8c8c8, 0x00c9c9c9, 0x00cacaca, 0x00cbcbcb, 0x00cccccc, 0x00cdcdcd, 0x00cecece, 0x00cfcfcf,
                 0x00d0d0d0, 0x00d1d1d1, 0x00d2d2d2, 0x00d3d3d3, 0x00d4d4d4, 0x00d5d5d5, 0x00d6d6d6, 0x00d7d7d7,
                 0x00d8d8d8, 0x00d9d9d9, 0x00dadada, 0x00dbdbdb, 0x00dcdcdc, 0x00dddddd, 0x00dedede, 0x00dfdfdf,
                 0x00e0e0e0, 0x00e1e1e1, 0x00e2e2e2, 0x00e3e3e3, 0x00e4e4e4, 0x00e5e5e5, 0x00e6e6e6, 0x00e7e7e7,
                 0x00e8e8e8, 0x00e9e9e9, 0x00eaeaea, 0x00ebebeb, 0x00ececec, 0x00ededed, 0x00eeeeee, 0x00efefef,
                 0x00f0f0f0, 0x00f1f1f1, 0x00f2f2f2, 0x00f3f3f3, 0x00f4f4f4, 0x00f5f5f5, 0x00f6f6f6, 0x00f7f7f7,
                 0x00f8f8f8, 0x00f9f9f9, 0x00fafafa, 0x00fbfbfb, 0x00fcfcfc, 0x00fdfdfd, 0x00fefefe, 0x00ffffff,
};
```

If your screen's driver configuration file doesn't have this table,when you adjust LUT,the screen will show blurred screen.

(2)After system starts up,there will be a node" sys/class/graphics/fb0/dsp_lut".You need to add appropriate permission to the node,so that BizlineAdjust_3XXX can write it.If the permission is not right, BizlineAdjust_3XXX can't read or write the node,adjustment will be useless.

(3)Install BizlineAdjust_30XX,adjust brightness and contrast status bar,to get satisfied display quality. BizlineAdjust_30XX will store related LUT data to below file: /data/data/com.rockchip.graphics/files/dsp_lut_bkp.

(4)Copy data from dsp_lut_bkp to related screen driver's dsp_lut[256] array.When the system restarts,it will read data from dsp_lut,and then write to LCDC's LUT table.

Appendix F Booting logo which supports BMP format

When Linux system boots,the default booting logo format is RGB565（16）(ppm).To improve the display quality of booting logo,we add support for BMP format.The configuration steps of BMP format logo is shown below:

1)Prepare a 24 bit size BMP format LOGO picture(Picture with 32bit size is not supproted).And resolution of the picture can't be over the screen's resolution.

2)Name the picture as xxx_bmp.bmp(must end with _bmp.bmp,for example logo_android_bmp.bmp),and put it under the directory"drivers/video/logo".

3)Modify Makefile、Kconfig、logo.c linux_logo.h：

```
--- a/drivers/video/logo/Kconfig
+++ b/drivers/video/logo/Kconfig
@@ -98,16 +98,21 @@ config LOGO_LINUX_800x480_CLUT224
  menuconfig LOGO_LINUX_BMP
                bool "Bmp logo support"
                default n
+config LOGO_LINUX_BMP_ANDROID
+        bool "Bmp logo android"
+         depends on  LOGO_LINUX_BMP
+         default n

index c7396a4..ccae7cc 100644
--- a/drivers/video/logo/Makefile
+++ b/drivers/video/logo/Makefile
obj-$(CONFIG_LOGO_G3_CLUT224)           += logo_g3_clut224.o
obj-$(CONFIG_LOGO_LINUX_BMP_SUNSET)     += logo_sunset_bmp.o
+obj-$(CONFIG_LOGO_LINUX_BMP_ANDROID)     += logo_android_bmp.o

--- a/drivers/video/logo/logo.c
+++ b/drivers/video/logo/logo.c
@@ -141,9 +141,14 @@ const struct linux_logo * __init_refok fb_find_logo(int depth)
```

```
                    if (depth >= 24)
                    {
                              #ifdef   CONFIG_LOGO_LINUX_BMP
                             #ifdef CONFIG_LOGO_LINUX_BMP_SUNSET
                              logo = &logo_sunset_bmp;
                              #endif
+
+                              #ifdef CONFIG_LOGO_LINUX_BMP_ANDROID
+                              logo = &logo_android_bmp;
+                              #endif
+
                              #endif
                    }
--- a/include/linux/linux_logo.h
+++ b/include/linux/linux_logo.h
@@ -54,6 +54,7 @@ extern const struct linux_logo logo_m32r_clut224;
 extern const struct linux_logo logo_spe_clut224;
 extern const struct linux_logo logo_g3_clut224;
 extern const struct linux_logo logo_sunset_bmp;
+extern const struct linux_logo logo_android_bmp;
```

4)In “make menuconfig”,remove support for general logo,check 'Bmp logo support' and related logo.

```
Graphics support   --->
          [*] Bootup logo   --->
          [*]    Bmp logo support   --->
          [*]    Bmp logo android
```

Attention: As BMP format picture is 24bit size.The generated kernel image will be bigger if using BMP format logo,maybe over partition or loader's support area.So it's better to use 'make zkernel.img' command to generate compressed kernel image when you are compiling kernel.

Appendix G High definition optimization

On RK3188 platform,the hightest screen definition we support can reach 2048*1536. For this kind of super resolution screen,it has high requirement with system bandwith(each frame data is 12M size).To get more fluent user experience for this kind of high definition screen,we take specific optimization strategy.To open such optimization test,we need to prepare two steps:

1)Kernel opens 512M memory for GPU,this step is implemented in related project's board codes.

```
#define HD_SCREEN_SIZE 1920UL*1200UL*4*3
static void __init rk30_reserve(void)
{
        int size, ion_reserve_size;
#if defined(CONFIG_ARCH_RK3188)
        /*if lcd resolution great than or equal to 1920*1200,reserve the ump memory
*/
        if(!(get_fb_size() < ALIGN(HD_SCREEN_SIZE,SZ_1M)))
        {
                int ump_mem_phy_size=512UL*1024UL*1024UL;
                resource_mali[0].start = board_mem_reserve_add("ump buf",
ump_mem_phy_size);
                resource_mali[0].end = resource_mali[0].start +
ump_mem_phy_size -1;
```

```
        }
#endif
```

HD_SCREEN_SIZE is the lowest value that we define as high definition screen.That is,defaultly if the definition of some kind of screen is more than 1920*1200,we think that it opens the high definition optimization,and we allocate 512M memory for GPU.If your screen definiton is less than 1920*1200,but you also want to open high definition optimization,then you need to modify HD_SCREEN_SIZE.Take a screen with 1920*1080 definition as an example,if you want to open high definition optimization,you need to define HD_SCREEN_SIZE a little lower:

```
    #define HD_SCREEN_SIZE 1920UL*1080UL*4*3
```

2)open the Android's HD optimization macro swith:

In the file"device/rockchip/rk30sdk/BoardConfig.mk",define BOARD_USE_LCDC_COMPOSER as true: BOARD_USE_LCDC_COMPOSER ?= true.Then make clean and recompile Android system.

Appendix H application development based on RK FB driver

H-1 summary

First RK FB driver follows Linux frame buffer driver extension,so application development based on RK FB is almost the same with development based on standard linux frame buffer.

RK main controller's LCDC is layered inside.Each layer is called win,each layer can display any size of picture anywhere on the screen that it supports,and each layer can implement overlay combination output by alpha blending or color key.

RK3066's LCDC has 3 layers:win0,win1,win2.Win0 and win1 support RGB and

YUV format,win2 only supports YUV format.RK292X,RK3168 and RK3188's LCDC have 2 layers,win0 supports RGB and YUV format,win1 only supports win1 format.

In RK FB,each win layer matches a FB device,their corresponding device node in Linux system is "/dev/graphics/fbx ",of which the corresponding relationship between win and fb can be set by program.Take RK3066 as an example:

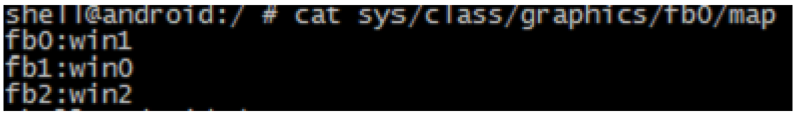
/dev/graphics/fb0-----win1

/dev/graphics/fb1-----win0

/dev/graphics/fb2-----win2

The detail corresponding relationship can be viewed by the command:

cat sys/class/graphics/fb0/map



```
shell@android:/ # cat sys/class/graphics/fb0/map
fb0:win1
fb1:win0
fb2:win2
```

H-2 Related system calls

FB device is a kind of standard character device, you need to operate it through related system call.The main system call are open,close,mmap and ioctl related,you can refer to Hal layer codes of the android SDK: androidsrc/hardware/rk29/libgralloc_ump/framebuffer_device.cpp

（1）open

before you use a FB device,you need to open it:

Fd = open(/dev/graphics/fbx, O_RDWR, 0);

（2）mmap

The call is used to map fb address allocated in kenrnel to the user space,so that it's convenient for application to read and write.

void* vaddr = mmap(0, fbSize, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

（3）ioctl：FBIOGET_FSCREENINFO

Linux FB driver native code,used to get fb_fix_screeninfo

（4）ioctl:FBIOGET_VSCREENINFO

Linux FB driver native code,used to get fb_var_screeninfo

（5）ioctl：FBIOPUT_VSCREENINFO

Linux FB driver native code,used to modify fb_var_screeninfo

Display related information can be set by modifying fb_var_screeninfo,and calling this ioctl.

struct fb_var_screeen_info is a very important struct of FB,large amount of display related parameters are configured by this struct.

```
struct fb_var_screeninfo {
	__u32 xres;			/* visible resolution		*/
	__u32 yres;
	__u32 xres_virtual;		/* virtual resolution		*/
	__u32 yres_virtual;
	__u32 xoffset;			/* offset from virtual to visible */
	__u32 yoffset;			/* resolution			*/

	__u32 bits_per_pixel;		/* guess what			*/
	__u32 grayscale;		/* != 0 Graylevels instead of colors */

	struct fb_bitfield red;		/* bitfield in fb mem if true color, */
	struct fb_bitfield green;/* else only length is significant */
	struct fb_bitfield blue;
	struct fb_bitfield transp;	/* transparency			*/

	__u32 nonstd;			/* != 0 Non standard pixel format */
```

```
    __u32 activate;           /* see FB_ACTIVATE_*       */


    __u32 height;             /* height of picture in mm      */

    __u32 width;              /* width of picture in mm       */


    __u32 accel_flags;        /* (OBSOLETE) see fb_info.flags */


    /* Timing: All values in pixclocks, except pixclock (of course) */

    __u32 pixclock;           /* pixel clock in ps (pico seconds) */

    __u32 left_margin;        /* time from sync to picture      */

    __u32 right_margin;           /* time from picture to sync      */

    __u32 upper_margin;           /* time from sync to picture      */

    __u32 lower_margin;

    __u32 hsync_len;          /* length of horizontal sync*/

    __u32 vsync_len;          /* length of vertical sync    */

    __u32 sync;               /* see FB_SYNC_*        */

    __u32 vmode;              /* see FB_VMODE_*           */

    __u32 rotate;             /* angle we rotate counter clockwise */

    __u32 reserved[5];        /* Reserved for future compatibility */

};
```

xres: When the kernel boots,it is initialized as the horizontal resolution of corresponding screen,in actual use,it indicates actual width of the data to be displayed(xact).

yres: When the kernel boots,it is initialized as the vertical resolution of corresponding screen,in actual use,it indicates actual height of the data to be displayed (yact).

xres_virtual:virtual width and stride of the data to be displayed.

yres_virtual:it is set to be result of vertical resolution of the screen multiples number of the buffer.

xoffset：horizontal offset of the data to be displayed in memory.

yoffset：vertical offset of the data to be displayed in memory.

FB driver will use xoffset and yoffset to determine what address to fetch data to display.

bits_per_pixel:the number of bits for each pixel.RGB565 is 16 bits,ARGB888 is 32bits.

There are other display related parameters need to be configured,but the native FB driver doesnot support related interfaces,we make use of members of fb_var_screen_info struct which defaultly haven't been used by the system.They are corresponding data's display position on the screen,corresponding data's display size on the screen(xsize,ysize),and data's display format.

xsize: corresponding data's display length on the screen, represented by grayscale's bit8~bit19.

ysize: corresponding data's display length on the screen, represented by grayscale's bit20~bit31.

so grayscale = (xsize<<8) | (ysize<<20)

xpos: corresponding data's display x cordinate on the screen, represented by nonstd's bit8~bit19.

ypos: corresponding data's display y cordinate on the screen, represented by nonstd's bit20~bit31.

data_format:data format that application writes to FB, represented by nonstd's bit0~bit7.

so nonstd = （xpos<<8）| (ypos<<20) | (data_format) data_format's value is defined as the following macro in rk_fb.h:

```
enum {
    HAL_PIXEL_FORMAT_RGBA_8888          = 1,
    HAL_PIXEL_FORMAT_RGBX_8888          = 2,
    HAL_PIXEL_FORMAT_RGB_888           = 3,
    HAL_PIXEL_FORMAT_RGB_565           = 4,
    HAL_PIXEL_FORMAT_BGRA_8888          = 5,
    HAL_PIXEL_FORMAT_RGBA_5551          = 6,
    HAL_PIXEL_FORMAT_RGBA_4444          = 7,
    .........................
    HAL_PIXEL_FORMAT_YCrCb_NV12         = 0x20,
   }
```

These macros are defined by the Android system,to be sure,ARGB's order defined by Android Macro is contrary to data's actual order in the memory.For example, RGBA_8888's actual order in the memory is ABGR, RGBX_8888's actual order in the memory is XBGR8888, BGRA_8888's actual order in the memory is ARGB8888,and so on.’X’ indicates that data in this position is invalid.

(6) ioctl：FBIOPAN_DISPLAY

Linux FB native system call.When application calls this ioctl,it will call pan_diaplay function registered in platform related FB driver(rk_fb.c).The driver will set a new frame of data's display address according to xoffset and yoffset of fb_var_screen_info.In addition,when you are running ioctl FBIOPUT_VSCREENINFO,the linux system will call function pan_display registered in platform related FB driver automaticly.

(7) ioctl：RK_FBIOGET_PANEL_SIZE

This system call is used to get screen definition,it returns a pointer that points to u32 panel_size[2],of which, panel_size[0] instores the screen's x definition, panel_size[1] instores the screen's y definition.

(8) ioctl：RK_FBIOSET_YUV_ADDR

This system call is used to set FB's base address.Defaultly RK FB buffer only allocates buffer for fb0,if the application wants to use fb1 and fb2,it needs to allocated continuous physical address for them,and pass address to fb through this system call.

This system call passes pointer that points to u32 addr[2].If the application draws RGB data,you only need to save RGB data's physical address to addr[0],data in addr[1] is unuseful.else if the application draws YUV data,then addr[0] saves Y data's physical address,addr[1] saves UV data's physical address.

(9) ioctl：RK_FBIOSET_OVERLAY_STATE

This system call is used to exchange win0 and win1 corresponding zorder.In RK LCDC,win0 and win1's position can be changed.Defaultly win0 layer is under win1 layer.If necessary, you can change their position by this system call. This system call passes a pointer that points to int ovl.If ovl=1,then win0 is located above win1.Otherwise if ovl = 0,win0 is located under win1.

(10) ioctl：RK_FBIOGET_OVERLAY_STATE

This system call is used to get win0 and win1's location,it returns a pointer that points to int ovl.If ovl=1,then win0 is above win1.Otherwise if ovl = 0,win0 is under win1.

(11) ioctl：RK_FBIOSET_ENABLE

This system call is used to enable and disable FB corresponding win layer.Now when you open FB device,the FB driver will enable corresponding win layer,but when you close fb device,it doesn't disable corresponding win layer.So if you want to disable FB corresponding win,you need to use this system call. This system call passes a pointer that points to int enable. If enable=0,then FB and corresponding win layer is closed,otherwise if enable = 1,then fb and related win layer is open.

(12) ioctl：RK_FBIOGET_ENABLE

This system call is used to return FB corresponding win's state,it returns a pointer that points to int enable. If enable=0,then FB corresponding win layer is closed,otherwise if enable = 1,then FB corresponding win layer is open.

(13) ioctl：RK_FBIOSET_VSYNC_ENABLE

After Android4.1,only when FB driver transmits vsync information to upper layer that the upper layer will start to render. This system call is used to enable or disable kernel's vsync reporting function.It returns a pointer that points to int enable. If enable=1,then kernel's vsync reporting function is open,otherwise if enable = 1,then kernel's vsync reporting function is closed.

(14) ioctl：RK_FBIOPUT_COLOR_KEY_CFG

This system call is used to configure LCDC's color key function. This system call passes a pointer that points to struct color_key_cfg clr_key_cfg.Each member of struct color_key_cfg matches a color key configure word of win layer.Bit24 is used to configure color key function switch,if bit24=1,then its corresponding color key function is open,bit0~bit23 is used to configure color value.

(15) ioctl：RK_FBIOSET_CONFIG_DONE

For applications that uses multiple win layers to implement overlay display function.To ensure each layer's configuration is synchronous,we add this ioctl.After you finish relavant configuration,once you call this ioctl,the relavant configuration can take effective at the same.Meanwhile,this ioctl pass a pointer points to int wait_fs.If wait_fs=1,then this system call will wait LCDC refresh display with these configuration and return.Otherwise if wait_fs=0,then it will return directly.After Android4.1,the recommended synchronization is implemented in Android,so defaultly wait_fs=0.If waiting refresh happens in kernel,the display fluency will be slowed.

H-3 Related system call process

system call in user space is dealed by linux system first,then passed to RK FB driver,finally passed to related LCDC driver,the process is below:

User system call ---->fbmem.c ----->rk_fb.c ---->rkxxx_lcdc.c

If you meet some problems when you are developing applications.You can debug and trace according to this process.The call process of open,ioctl is below:

Open---->fb_open(fbmem.c)--->rk_fb_open(rk_fb.c)--->

rkxxx_lcdc_open(rkxxx_lcdc.c)

Ioctl--->fb_ioctl(fbmem.c)--->do_fb_ioctl(fbmem.c)-->

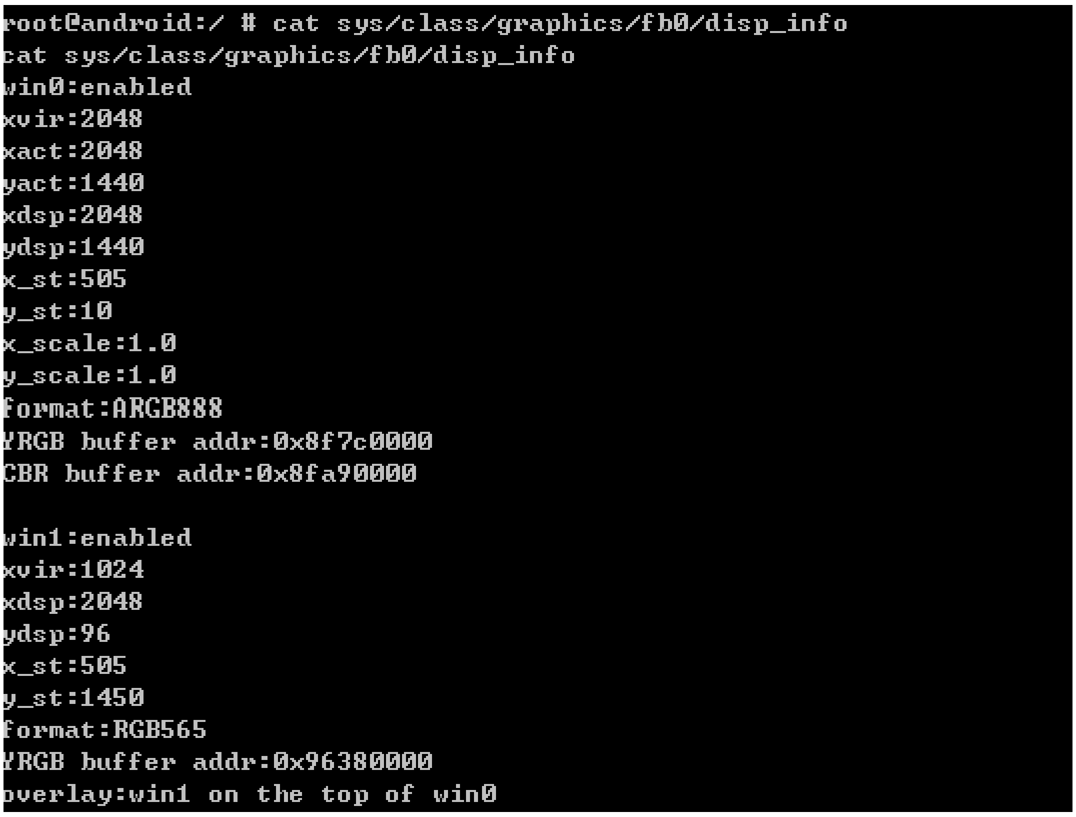
rk_fb_ioctl(rk_fb.c)--->rkxxx_lcdc_ioctl(rkxxx_lcdc.c)

There is a swith in LCDC driver,which you can use to open and close corresponding debug infomation dynamically to check whether your setting is normal.

echo x > sys/module/rk30_lcdc/parameters/dbg_thresd

The print is categorized,x=0 represents printing nothing;x=1 represents printing part important information;x=2 represents printing more information.Meanwhile,you can get more detailed display information by cating the below node:

cat sys/class/graphics/fb0/disp_info



```
root@android:/ # cat sys/class/graphics/fb0/disp_info
cat sys/class/graphics/fb0/disp_info
win0:enabled
xvir:2048
xact:2048
yact:1440
xdsp:2048
ydsp:1440
x_st:505
y_st:10
x_scale:1.0
y_scale:1.0
format:ARGB888
YRGB buffer addr:0x8f7c0000
CBR buffer addr:0x8fa90000

win1:enabled
xvir:1024
xdsp:2048
ydsp:96
x_st:505
y_st:1450
format:RGB565
YRGB buffer addr:0x96380000
overlay:win1 on the top of win0
```

Appendix I Q&A

(1)If you have a new designed board,when the system starts up,the screen does't display,how do you check?

Is the backlight on?If the backlight isn't on,you need to check whether the power of backlight is running normally,and whether the configuration of PWM channel is right.

Can you detect related signals that comes out from LCDC(DCLK,Data signal)?

Whether the power supply of LCD related screen is normal?

Whether the LCD screen related parameters are strictly configured according to Datasheet?

(2)When the system starts up,some UI displays spots or noise?

Try to reverse DCLK,if there is no effect,ask hardware engineers to help you check related signal quality.

(3)When the system starts up, the main UI doesn't slide smoothly,the frame rate is less than 45fps?

Run "cat sys/class/graphics/fb0/fps" in the serial terminal to view frame rate of LCDC,check whether it is close to 60fps,if not,please check whether the configuration of DCLK is consistence with datasheet.

Is this a HD screen(more than 1920*1080),if yes,please ensure DDR frequency reaches 533MHZ,and the HD optimization swith in Android layer is turned on.

(4)Use BizlineAdjust_30XX apk to make gamma adjustment,but the screen doesn't change.

Have you changed the permission of "sys/class/graphics/fb0/dsp_lut" node,so that upper apk can read or write this node.For detail information,please refer to Appendix5 Use LUT function to improve the display quality.

(5)Use BizlineAdjust_30XX apk to make gamma adjustment,but the screen blurred when you adjust.

Check the kernel log,do you see the printing “no buffer to backup lut data” ,if yes,please check your screen configure file,to see whether there is definition for Macro USE_RK_DSP_LUT and Table dsp_lut. For detail information,please refer to Appendix5 Use LUT function to improve the display quality.