全 志 科 技
Allwinner Technology

A20 显示驱动模块说明书

V1.0

2014-01-18

Revision History

Version	Date	Changes compared to previous issue
V1.0	2014-01-18	初建版本

目录

1. 概述

1.1. 编写目的

介绍显示模块使用方法。

1.2. 适用范围

适用于 A20 平台.

1.3. 相关人员

A20 显示开发人员及 FAE。

2. 模块介绍

2.1. 模块功能介绍

本模块主要处理显示相关功能，主要功能如下：

- 支持 linux 标准的 framebuffer 接口
- 支持 lcd(hv/lvds/cpu/dsi)/hdmi 输出
- 支持两路显示输出
- 支持多图层叠加混合处理
- 支持硬件鼠标
- 支持多种显示效果处理（alpha, colorkey, 图像细节增加，亮度/对比度/饱和度/色度调整）
- 支持多种图像数据格式输入(arg,yuv)
- 支持图像缩放处理
- 支持 3D 显示

2.2. 模块配置介绍

2.2.1. Menuconfig 配置：

在 linux 目录下，#make ARCH=arm menuconfig，进入配置界面

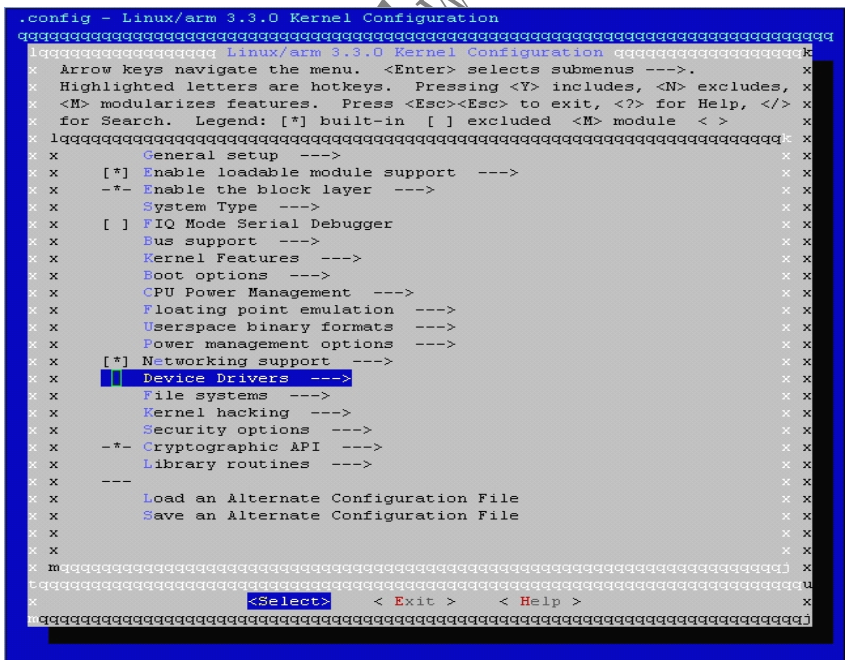


图 2-1 menuconfig-device drivers

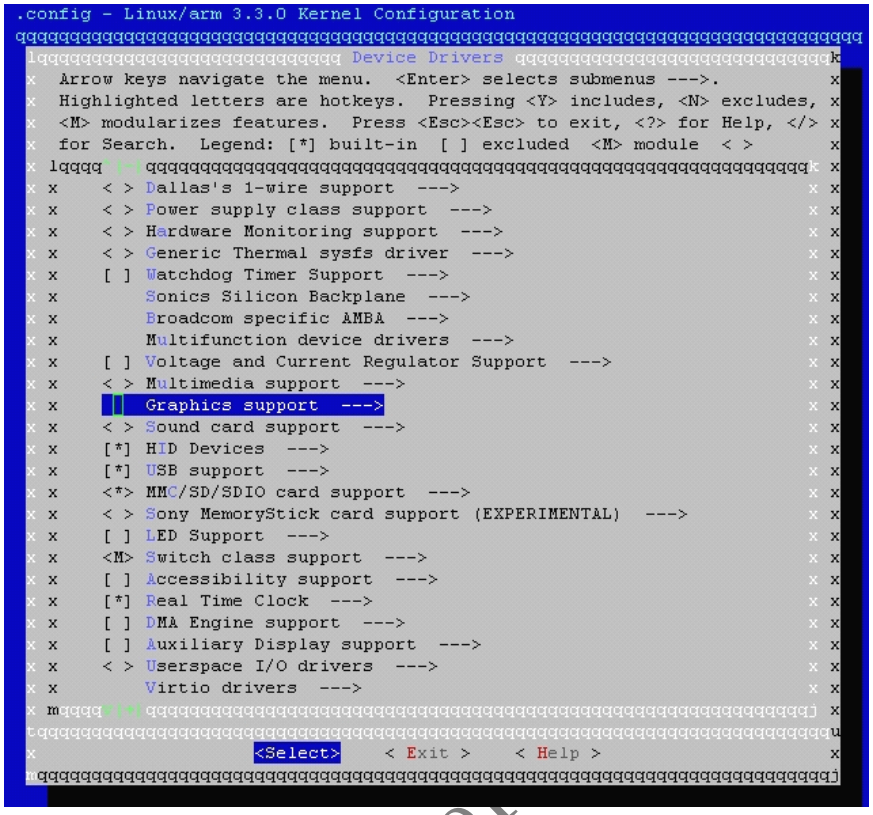


图 2-2 menuconfig-graphics support

Framebuffer 驱动：

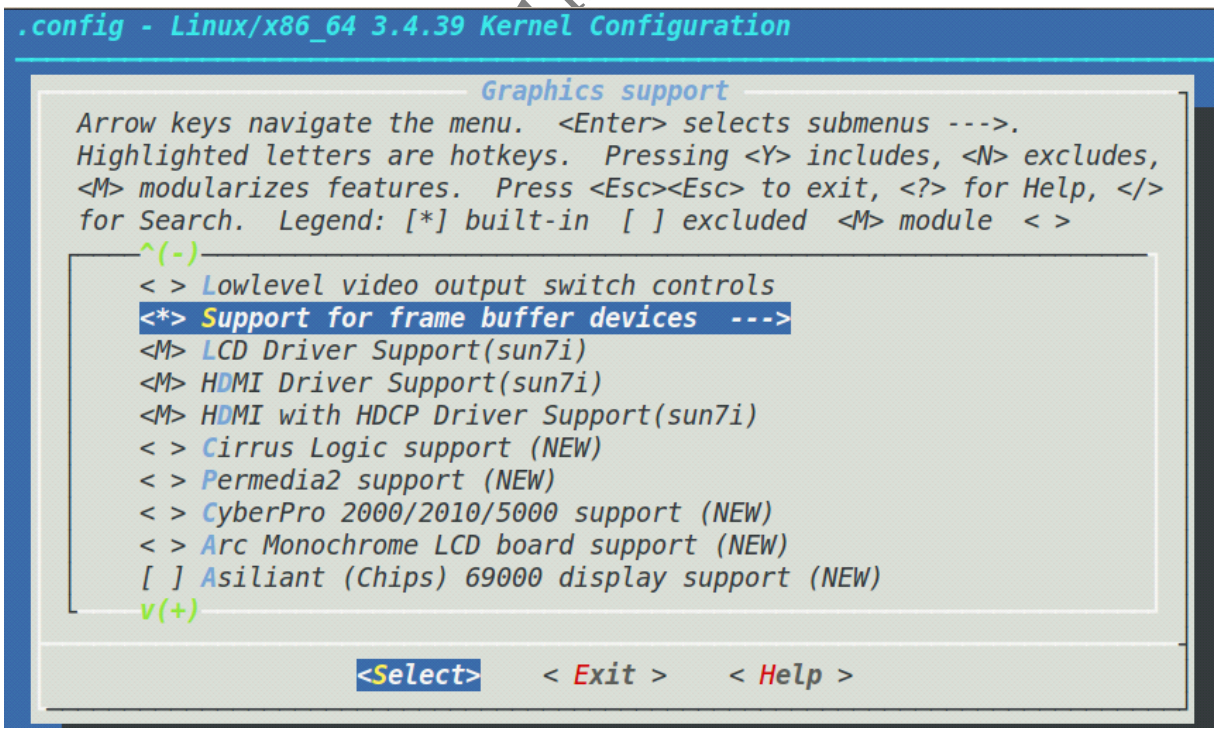


图 2-3 menuconfig-support for frame buffer devices

Disp 驱动：

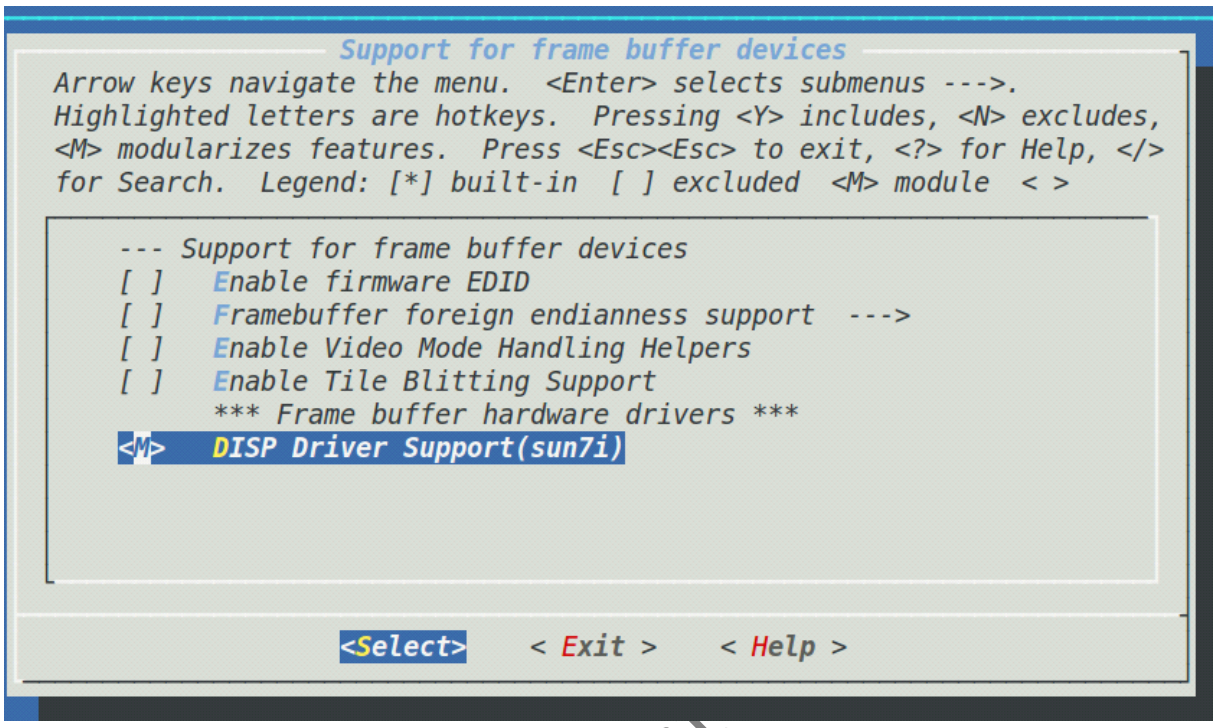


图 2-4 menuconfig –disp driver support(sun7i)

Lcd 驱动：

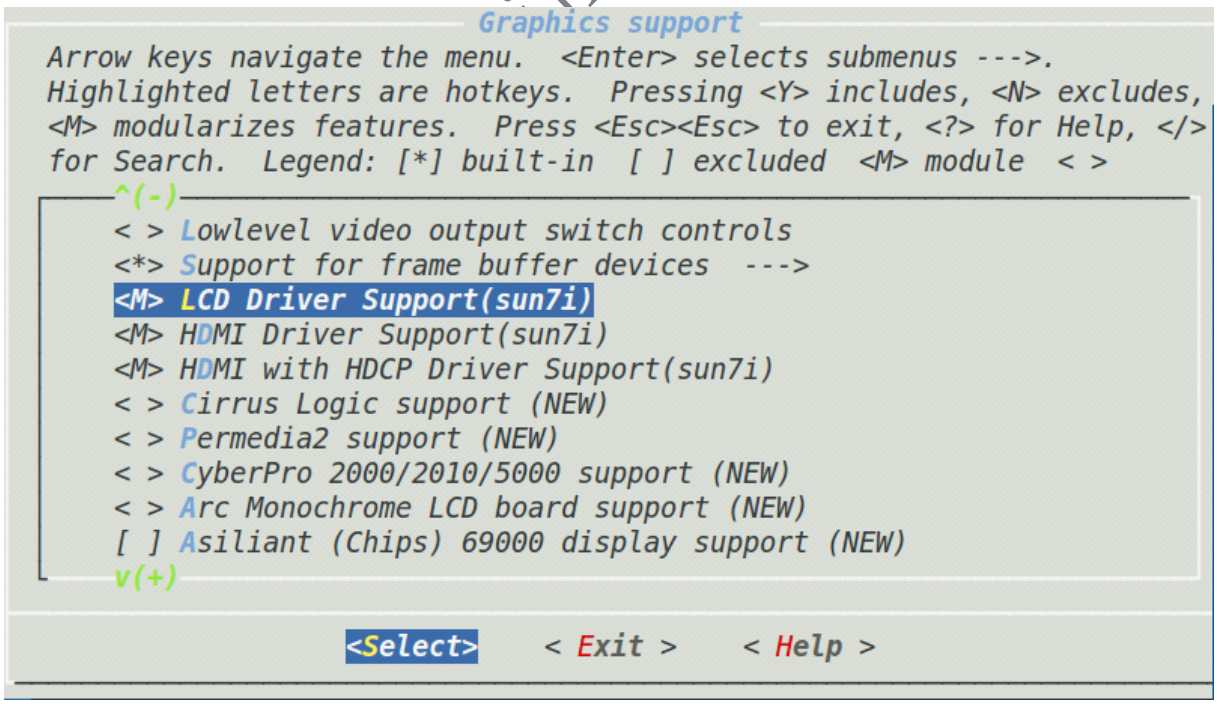


图 2-5 menuconfig-lcd driver support(sun7i)

Hdmi 驱动：

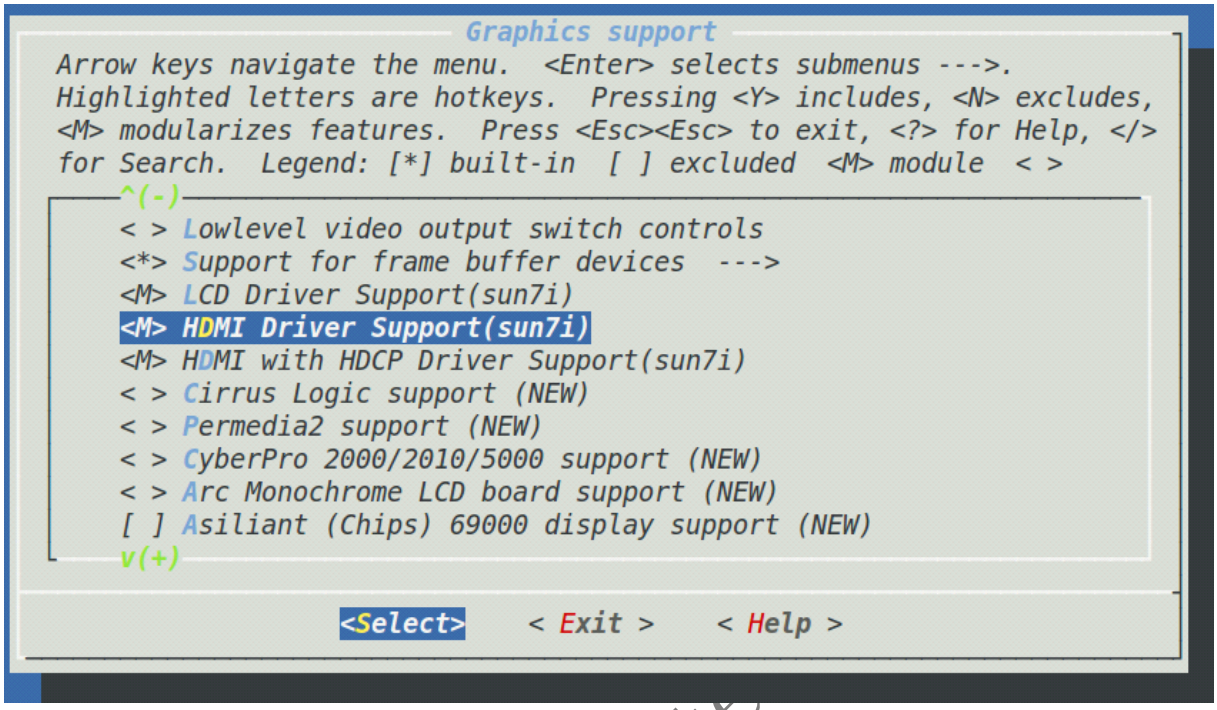


图 2-6 menuconfig-hdmi driver support(sun7i)

2.2.2. Sys_config.fex 显示相关配置

☞ DE 部分配置

```
[disp_init]
disp_init_enable        = 1
disp_mode               = 0

screen0_output_type     = 1
screen0_output_mode     = 4

screen1_output_type     = 1
screen1_output_mode     = 4

fb0_framebuffer_num     = 2
fb0_format              = 10
fb0_pixel_sequence      = 0
fb0_scaler_mode_enable  = 0
fb0_width               = 0
fb0_height              = 0

fb1_framebuffer_num     = 2
fb1_format              = 10
```

```
fb1_pixel_sequence      = 0
fb1_scaler_mode_enable  = 0
fb1_width               = 0
fb1_height              = 0
lcd0_backlight          = 197
lcd1_backlight          = 197
lcd0_bright             = 50
lcd0_contrast           = 50
lcd0_saturation         = 57
lcd0_hue                = 50
lcd1_bright             = 50
lcd1_contrast           = 50
lcd1_saturation         = 57
lcd1_hue                = 50
```

☞ Lcd 屏相关的配置

```
[lcd0_para]
lcd_used                = 1
lcd_x                   = 1024
lcd_y                   = 600
lcd_width               = 155
lcd_height              = 85
lcd_dclk_freq           = 51
lcd_pwm_not_used        = 0
lcd_pwm_ch              = 0
lcd_pwm_freq            = 10000
lcd_pwm_pol             = 1
lcd_if                  = 0
lcd_hbp                 = 158
lcd_ht                  = 1344
lcd_vbp                 = 25
lcd_vt                  = 1270
lcd_vspw                = 3
lcd_hspw                = 20
lcd_hv_if               = 0
lcd_hv_smode            = 0
lcd_hv_s888_if          = 0
lcd_hv_syuv_if          = 0
lcd_lvds_ch             = 0
lcd_lvds_mode           = 0
lcd_lvds_bitwidth       = 0
```

```
lcd_lvds_io_cross          = 0
lcd_cpu_if                 = 0
lcd_frm                    = 1
lcd_io_cfg0                = 0x00000000
lcd_gamma_correction_en    = 0
lcd_gamma_tbl_0            = 0x00000000
lcd_gamma_tbl_1            = 0x00010101
lcd_gamma_tbl_255          = 0x00ffffff

lcd_bl_en_used             = 1
lcd_bl_en                  = port:PH07<1><0><default><1>
lcd_power_used             = 1
lcd_power                  = port:PH08<1><0><default><1>
lcd_pwm_used               = 1
lcd_pwm                    = port:PB02<2><0><default><default>
lcdd0                      = port:PD00<2><0><default><default>
lcdd1                      = port:PD01<2><0><default><default>
lcdd2                      = port:PD02<2><0><default><default>
lcdd3                      = port:PD03<2><0><default><default>
lcdd4                      = port:PD04<2><0><default><default>
lcdd5                      = port:PD05<2><0><default><default>
lcdd6                      = port:PD06<2><0><default><default>
lcdd7                      = port:PD07<2><0><default><default>
lcdd8                      = port:PD08<2><0><default><default>
lcdd9                      = port:PD09<2><0><default><default>
lcdd10                     = port:PD10<2><0><default><default>
lcdd11                     = port:PD11<2><0><default><default>
lcdd12                     = port:PD12<2><0><default><default>
lcdd13                     = port:PD13<2><0><default><default>
lcdd14                     = port:PD14<2><0><default><default>
lcdd15                     = port:PD15<2><0><default><default>
lcdd16                     = port:PD16<2><0><default><default>
lcdd17                     = port:PD17<2><0><default><default>
lcdd18                     = port:PD18<2><0><default><default>
lcdd19                     = port:PD19<2><0><default><default>
lcdd20                     = port:PD20<2><0><default><default>
lcdd21                     = port:PD21<2><0><default><default>
lcdd22                     = port:PD22<2><0><default><default>
lcdd23                     = port:PD23<2><0><default><default>
lcdclk                     = port:PD24<2><0><default><default>
lcdde                      = port:PD25<2><0><default><default>
lcdhsync                   = port:PD26<2><0><default><default>
```

lcdvsync = port:PD27<2><0><default><default>

3. 硬件框架

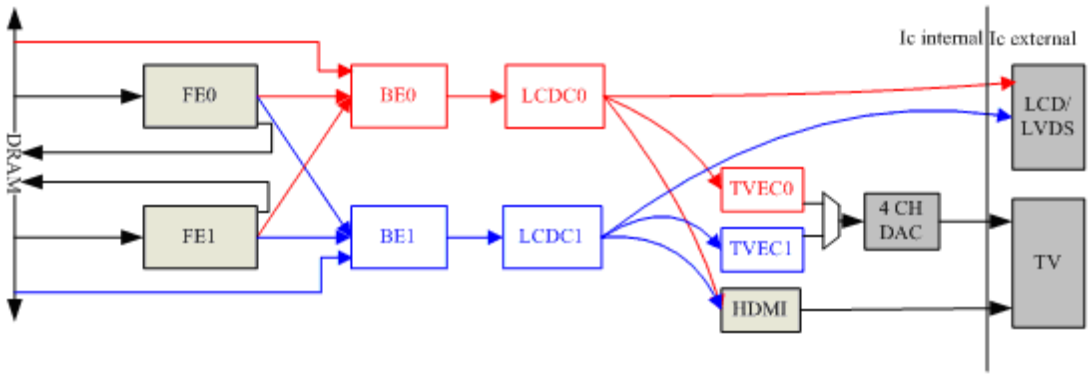


图 3-1 A20 显示框架图

从上图可以看出，A20 显示涉及到的主要模块包括：FE,BE,LCDC,TVE,HDMI。
目前 A20 常见的数据流走向主要有以下三种方式：

- 经过 FE 直接回写到 DRAM。

FE 具有数据回写功能，正常流程下，数据通过 FE 走向下一级显示接口，比如 BE，最后到相应的显示设备。但是 FE 也可以直接回写到 DRAM。
应用场景：
截屏功能，wifidisplay。

- 经过 FE 进入 BE 再由 LCDC 输出到显示设备。

FE 具有 3D 合成，对图层进行缩放的功能，播放 3D 视频时，每一帧图像都需要经过 FE 进行合成；同样视频播放，需要做图像缩放功能，那也是少不了 FE 的功劳。
应用场景：
3D,视频播放，颜色空间转换(YUV->RGB),de-interlace(去隔行)。

- 直接由 BE 进入到 LCDC 再到显示设备。

应用场景：
常见的 UI 场景。

4. 功能模块介绍

4.1. FE Feature

- Output scan type: interlace/progressive
- De-interlace method: weave/bob/motion-adaptive/motion-adaptive-bob
- Input format: YUV444/YUV422/YUV420/YUV411/RGB
- Direct display output format: RGB
- Writeback output format : RGB/YUV444/YUV422/YUV420/YUV411
- 3 channel scaling pipelines for scaling up/down
- Programmalbe source image size from 8X4 to 8192X8192 resolution
- Programmalbe destination image size from 8X4 to 8192X8192 resolution
- 8 tap scale filter in horizontal and 4 tap in vertical direction
- 32 programmable coefficients for each tap
- Color space conversion betwwen RGB and YUV
- Output support directly display and write back to memory
- Input support from dram, image and interface of lcd with image
- Support 3D format content input/output format convert/display(including HDMI)

4.2. BE Feature

- 4 moveable & size-adjustable layers
- Layer size up to 2048x2048 pixels
- Alpha blending support
- Colorkey support
- Write back function support
- 1/2/4/8 palette support
  16/24/32 bpp color support(external frambuffer)
  5/6/5，1/5/5/5，0/8/8/8，8/8/8/8，4/4/4/4
- Hardware cursor support
  32x32 @ 8bpp
  64x64 @ 2bpp
  64x32 @ 4bpp
  32x64 @ 4bpp
- Sprite function support
- 32bpp true color or 8bpp palette mode
- up to 32 independent sprite blocks
- each block can set arbitrary coordinate，the block size can be adjust
- YUV intput channel support
- Output color correction

4.3. LCDC(TCON) Feature

- System clock : 270~297MHZ
- 3 input source: 2 DE sources and 1 DMA source
- Support simultaneous display(differenct picture and video)for both LCD and TV
- Support HV-DE-sync digital parallel RGB input LCD panels
- Support HV-DE-sync digital serial RGB(both delta and stripe panel) input LCD panels
- Support analogRGB input LCD panels
- Support 18/16/9/8bit 8080 cpu-if panels
- CCIR 565 output interface for LCD panels or TVE
- Up to full HDTV timing for TVE and HDMI transmitter
- Internal line scaling and gamma correction

4.4. HDMI Feature

- HDMI V1.3 compliance
- support up to 165M pixel/second
- support Max 4K*4K resolution
- support 480I/576I/480P/576P/720P/1080I/1080P at 24/25/30/50/59.9hz
- support 24/30/36/48-bit RGB data format, with 2X/4X repeater
- support up to 8 channel, 24 bit PCM(IEC60958)
- support IEC61937 compress audio formats
- support 1-bit audio
- support HD audio(DTS-HD and Dolly MAT, IEC61937 format)
- hardware receiver active sense and hot plug detect

5. 驱动框架

5.1. 源码 Tree

```
drivers
├─video                              显示驱动目录
│   ├─fbmem.c                        framebuffer core
│   ├─sun7i                          display driver for sun7i
│   │   ├─disp/
│   │   │   ├─dev_disp.c      display driver 层
│   │   │   ├─dev_fb.c        framebuffer driver 层
│   │   │   ├─de_bsp/         de_bsp 层
│   │   │   │   ├─de/
│   │   │   │   ├─disp_layer.c    disp_scaler.c disp_lcd.c…
│   │   │   │   │  └─ebios/                  de_bios 层
│   │   │   │   │  ├─de_lcdc.c de_fe.c de_be.c …
│   │   │   │   │
│   │   │   │   └─OSAL/    OSAL 层,与操作系统相关层
│   │   ├─lcd/         lcd driver
│   │   └─hdmi/            hdmi driver
```

5.2. 显示驱动总体框架

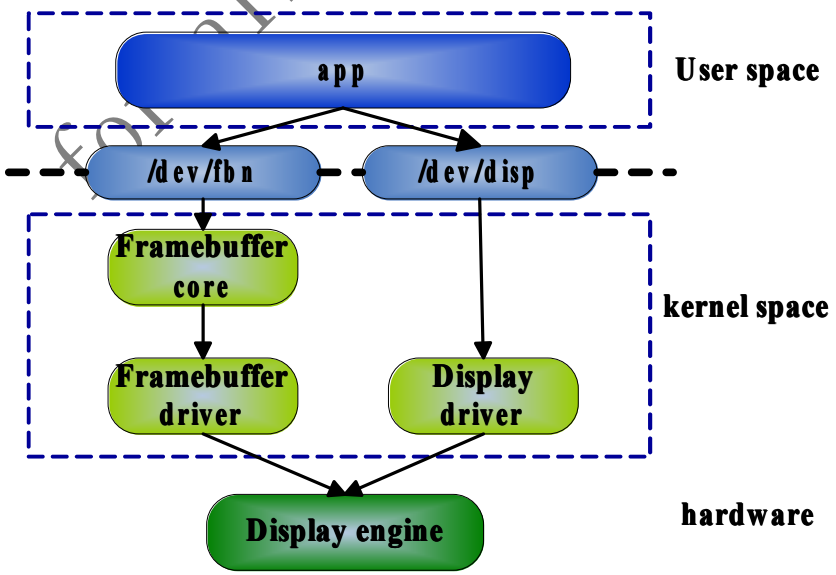


图 5-1 显示驱动整体框架图

从图 5-1 可以看出，显示系统可划分为三个层面，用户空间的应用层，内核空间的显示驱动层，硬件层。显示驱动模块直接与显示引擎、输出设备控制器等相关硬件模块打交

道，并通过内核向用户空间提供相应的设备结点及统一的接口。应用层则通过显示设备结点 dev/fb0(1..7)、dev/disp 控制显示设备的开关，图像数据的流向。
显示驱动模块分为 framebuffer driver 与 display driver。其中 framebuffer driver 是对 Linux 标准的 framebuffer 设备驱动的具体实现，display driver 则是自定义的接口实现。

5.3. Framebuffer 驱动模块结构图

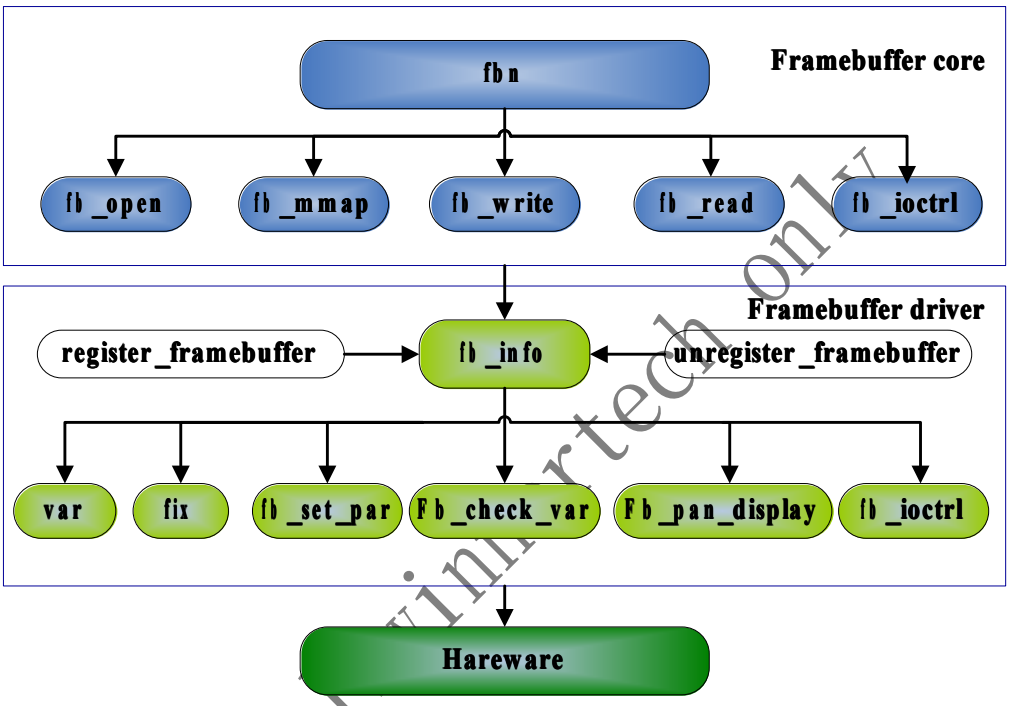


图 5-2 framebuffer 驱动模块结构图

Framebuffer 驱动模块分两个层次，第一层为 framebuffer 核心层，定义了统一的接口并实现了 framebuffer 的基本架构，第二层为具体的 framebuffer 驱动，是针对 A20 平台的具体实现。针对 A20，framebuffer 驱动模块对标准的 framebuffer 设备驱动进行了扩展，提供了两个 IOCTL CMD，FBIOGET_LAYER_HDL_0/ FBIOGET_LAYER_HDL_1，功能为获取 framebuffer 对应的图层句柄，获取到图层句柄后，可调用 display 驱动模块修改该图层的所有参数，以达到特定的显示效果，满足用户更高级的需求。

5.4. Display 驱动模块结构图

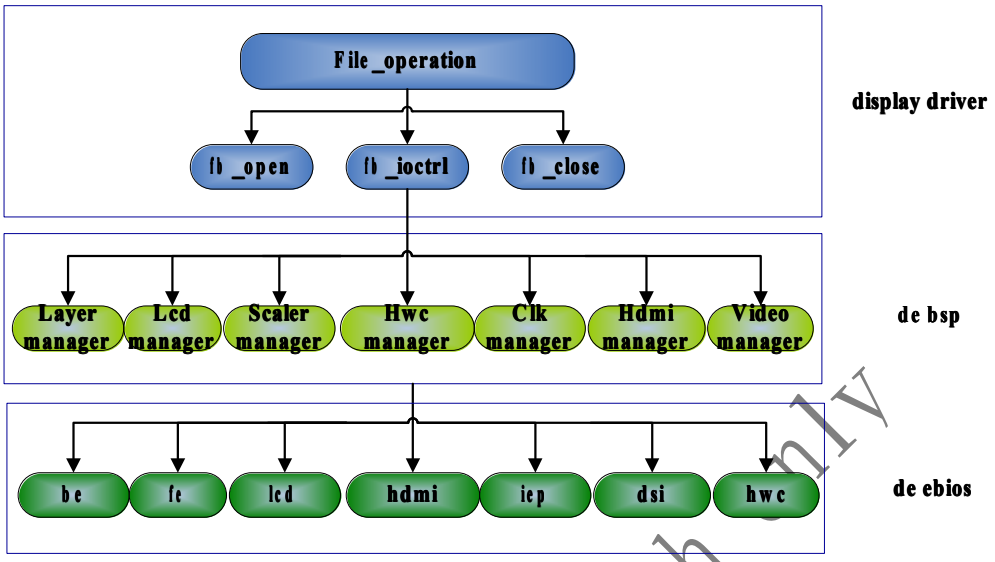


图 5-3 驱动模块结构图

分为三层，displaye driver 层，负责向上提供接口，主要通过 ioctrl 方式提供，与操作系统密切相关。

de_bsp 层，主要负责对硬件功能模块进行抽象，是显示处理的逻辑规则主要体现， 此层按功能模块分类。

- layer manager 主要管理各个图层的申请释放、参数设置，各个图层的关系；
- lcd manager 主要管理 lcd 的显示打开关闭、设置背光；
- scaler manager 则管理 scaler 的申请释放、启动；
- hwc manager 则管理硬件鼠标的显示；
- clk manager 负责所有显示模块的 clk 管理，clk 源的选择，clk 的频率设置，clk 的打开关闭等；
- hdmi manager 负责 hdmi 的显示功能；
- video manager 负责 video 播放功能的管理，提供给用户打开关闭视频、设置视频帧信息、获取当前播放帧信息几个比较简单的接口，用户可以设置新一帧的信息，而 video manager 会在适当的时候将新帧的信息配置给硬件让其生效，尽量简化 video 操作。

de_ebios 层，为硬件相关层，主要与硬件模块打交道，以硬件模块分类。

Display driver module 是采用 platform device driver 的驱动注册与管理机制，驱动和资源的管理较为独立，更方便系统对驱动进行电源管理等。

5.5. HDMI 模块驱动

Hdmi 驱动在初始化的时候创建了一个主线程，此线程为驱动的最主要部分，维护着一个状态机，如图 5-4 所示。

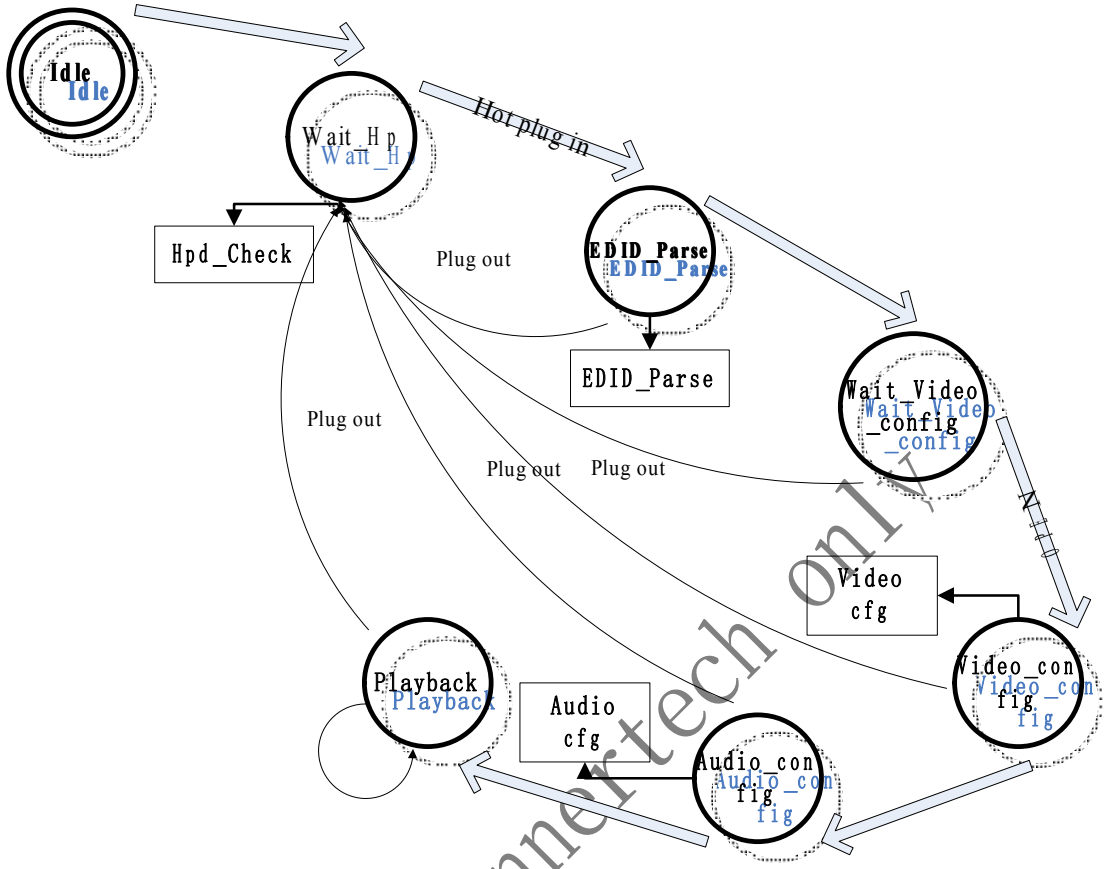


图 5-4 HDMI 状态图

初始时为 idle 状态，然后直接进入 wait_hp 状态，此时如果有 hdmi 电视接入，则进入 EDID_Parse 的状态，在这个状态下，hdmi 模块将会解析 hdmi 电视的 EDID 信息，EDID 信息包含显示设备的基本参数，如制造厂商、产品名称、最大行场频、可支持的分辨率等，通过 EDID 信息可以判断该电视支持的模式；解析完 EDID 信息，将会进入 wait_video_config 状态；如果 HDMI 显示输出已经打开，则由 wait_video_config 进入 video_config 状态，并进行 video config 操作，接着进入 audio_config 状态，进行 audio config 操作，最终进入 play back 状态，并一直保持着，直到 plug out 或者 hdmi close。在状态机中的任何状态下，如果 hdmi 线拔出，将会回到 wait_hpd 状态。

5.6. Lcd 驱动结构

Lcd 驱动是为了方便用户配置与 Lcd 相关的参数及初始化流程而独立出来的，非常的简单，只提供了一些抽象的接口方便用户配置。

6. 驱动流程

6.1. Dispaly driver 初始化流程

Display driver 的初始化主是针对各个功能模块的初始化，流程如下图所示。Fe init 主要工作是初始化 fe 时钟，使能中断，注册中断服务程序；be init 中初始化 be 时钟，使能中断；iep 初始化中分别初始化 deu/cmu/drc 三个模块；lcd 初始化中首先获取用户配置信息，然后初始化 lcd 时钟，注册 lcd vblanking 中断服务程序，如果用户配置中说明使用 lcd，将会获取 lcd 的参数信息，对 lcd 控制器进行配置，然后配置用于背光的 pwm 参数。Video 初始化只做一个事情，即申请 de-interlace 需要的 flag memory。

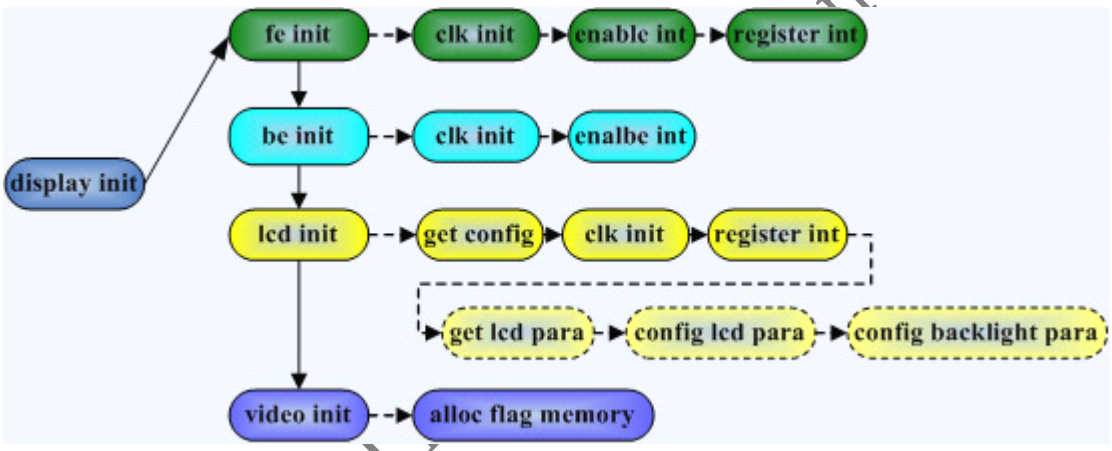
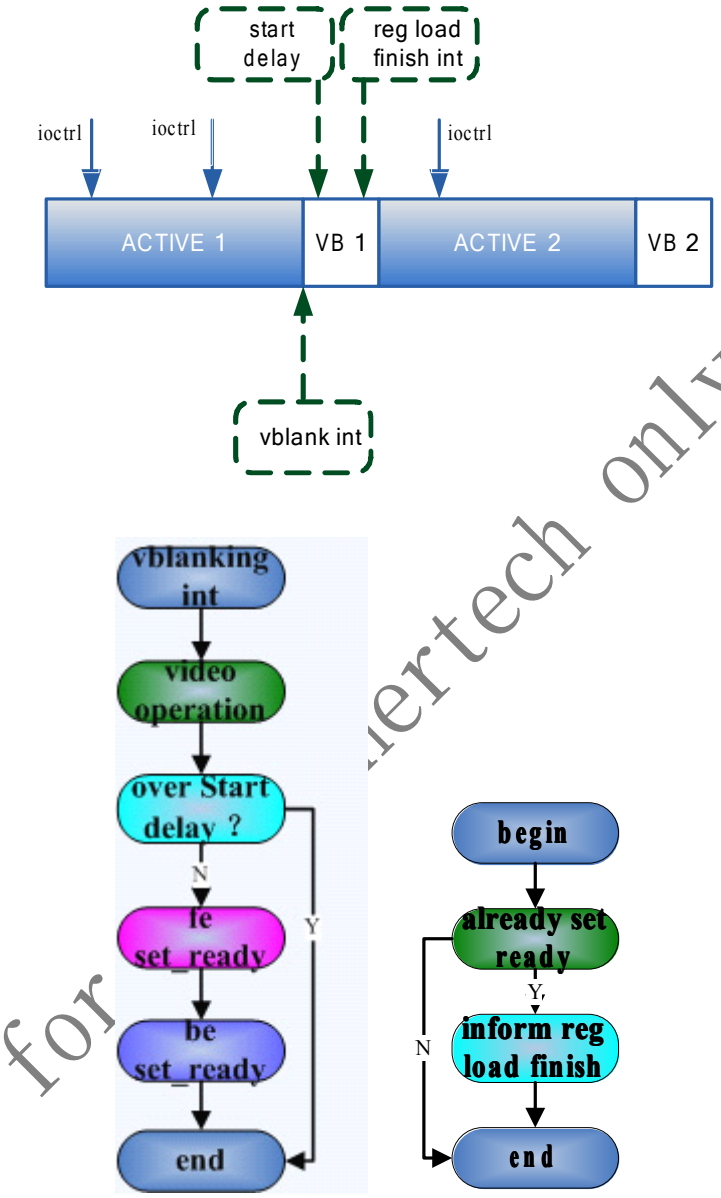


图 6-1

中断流程：

在显示中，一个完整的图像帧包含两个部分，一个是 active 区，有效图像数据部分，一个是 vblanking 区，即垂直消隐区。在 vblanking 区的起始位置硬件会产生一个 vb 中断。lcd vblanking 中断和 fe 的 register load finish 中断。为了保证显示的正确性，避免出现图片撕裂等现象，相关寄存器都是 doubole buffer 的，软件可读写的是后端寄存器，而前端寄存器则为硬件在用的。软件修改参数配置后，将通知硬件将新的参数配置从后端寄存器 load 到前端寄存器中，如此新的参数配置才会真正生效。
Lcd vblanking 中断主要负责 set ready，即告诉硬件，所有新的参数已配置到寄存器中，可以启用新的参数配置。
Set ready 和 load 寄存器不是同步完成的，set ready 后硬件将会在新帧的 start_delay 处进行前后端寄存器参数同步的动作，动作完成后将会触发 fe 的 regitser load finish 中断。Fe 的 register load finish 中断服务函数做的事情很简单，就是通知驱动，新的参数配置已经生效。
V 中断及 load finish 中断的处理流程如下图所示：



上图 左为 vblanding 中断的处理流程 右为 reg load finish 中断的处理流程。

v 中断上图中的左图所示，video operation 处理 video 功能相关的操作，详见 video manager；over start delay 是判断当前是否已经超过了 start delay 时间点，如果过了则直接退出，因为过了 start delay 后，即使 set ready，新的参数配置也已经不会再生效了；然后是 fe 的 set ready 操作，这个操作的前提是 fe 模块在使用中；最后是 be 的 set ready 操作。在做完这些操作之后，会标志 set ready 操作已经完成。

Reg load finish 中断处理流程如上图的右图所示，非常简单，如果 set ready 操作已经完成，即已经经过了 v 中断的处理，那么将通知驱动 load finish。可以通过信号量，等待

队列多种方式实现，目前采用的是等待队列的方式。

如果在 v 中断与 load finish 中断之间调用显示驱动的话，那么将会等待两次 load finish 中断才能返回，如下面的第一种情况；其他情况将按照这样的顺序，调用，v 中断，load finish 中断，调用返回，如下面的第二种情况。

V int---ioctrl---wait_event---load finish int---v int---load finish int---wake_up—ioctrl return

Ioctrl---wait_event---v int---load finish int---wake_up---ioctrl return

7. DISPLAY API

7.1. 图层操作相关接口

DISP_CMD_LAYER_REQUEST

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LAYER_REQUEST；
aux 显示通道 0/1；
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为__disp_layer_work_mode_t 图层工作模式;

- **RETURNS**

如果成功，则返回图层句柄；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于申请一个图层，该接口没有对该图层作任何设置，默认按 normal 工作模式申请，故推荐在申请图层后随即调用 DISP_CMD_LAYER_SET_PARA 命令对该图层的参数进行设置。

- **DEMO**

```
//申请图层，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                arg[3];
__hdle          hlay;

arg[0] = (__u32)DISP_LAYER_WORK_MODE_NORMAL;
hlay    = ioctl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);
```

DISP_CMD_LAYER_RELEASE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LAYER_RELEASE；
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于释放一个图层，对该图层的所有参数被清成默认值，如想使用该图层必须重新申请。

- **DEMO**

```
//释放图层，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];

arg[0] = (__u32)hlay;
ioctl(disphd, DISP_CMD_LAYER_RELEASE, sel, (void*)arg);
```

DISP_CMD_LAYER_OPEN

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

  hdle　　显示驱动句柄；
  cmd　　DISP_CMD_LAYER_OPEN；
  pBuffer　　pBuffer[0]为显示通道 0/1；
  　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于打开一个图层，即该图层在输出设备上显示出来。

- **DEMO**

```
//打开图层，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];

arg[0] = (__u32)hlay;
ioctl(disphd, DISP_CMD_LAYER_OPEN, sel, (void*)arg);
```

DISP_CMD_LAYER_COLSE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

  hdle　　显示驱动句柄；
  cmd　　DISP_CMD_LAYER_CLOSE；
  pBuffer　　pBuffer[0]为显示通道 0/1；
  　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于关闭一个图层，即该图层在输出设备上不显示。

- **DEMO**

```
//关闭图层，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];

arg[0] = (__u32)hlay;
ioctl(disphd, DISP_CMD_LAYER_CLOSE, sel, (void*)arg);
```

DISP_CMD_LAYER_SET_FB

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_SET_FB；
aux　　显示通道 0/1；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为指向__disp_fb_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于设置图层 framebuffer，保持原来的 source window 设置。

- **DEMO**

```
//设置图层 fb，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];
__disp_fb_t     layer_fb;
__u32           width = 800;
__u32           height = 480;

memset(&layer_fb, 0, sizeof(__disp_fb_t));
layer_fb.addr[0]        = (__u32)mem_in; //FB 地址
layer_fb.size.width     = width;
layer_fb.mode           = DISP_MOD_INTERLEAVED;
layer_fb.format         = DISP_FORMAT_ARGB8888;
layer_fb.br_swap        = 0;
layer_fb.seq            = DISP_SEQ_BGRA;

arg[0] = (__u32)hlay;
arg[1] = (__u32)&layer_fb;
ioctl(disphd, DISP_CMD_LAYER_SET_FB, sel, (void*)arg);
```

DISP_CMD_LAYER_GET_FB

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LAYER_GET_FB；
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
pBuffer[2]为图层 FB 信息，指向__disp_fb_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于获取图层 framebuffer。

- **DEMO**

```
//获取图层 fb，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                arg[3];
__disp_fb_t      layer_fb;
__u32            ret = 0;

arg[0] = (__u32)hlay;
arg[1] = (__u32)&layer_fb;
ret = ioctl(disphd, DISP_CMD_LAYER_GET_FB, sel, (void*)arg);
```

DISP_CMD_LAYER_SET_SRC_WINDOW

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LAYER_SET_SRC_WINDOW；
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
pBuffer[2]为 source window 信息，指向__disp_rect_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于设置图层 source window。

- **DEMO**

```
//设置图层 src window，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                arg[3];
```

```
__disp_rect_t    src_window;
__u32            ret = 0;

src_window.x = 0;
src_window.y = 0;
src_window.width = 800;
src_window.height = 480;

arg[0] = (__u32)hlay;
arg[1] = (__u32)& src_window;
ret = ioctl(disphd,DISP_CMD_LAYER_SET_SRC_WINDOW,sel, (void*)arg);
```

DISP_CMD_LAYER_GET_SRC_WINDOW

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_GET_SRC_WINDOW；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
　　pBuffer[2]为 source window 信息，指向__disp_rect_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于获取图层 source window 信息。

- **DEMO**

```
//获取图层 src window，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32            arg[3];
__disp_rect_t    src_window;
__u32            ret = 0;

arg[0] = (__u32)hlay;
arg[1] = (__u32)& src_window;
ret = ioctl(disphd,DISP_CMD_LAYER_GET_SRC_WINDOW,sel, (void*)arg);
```

DISP_CMD_LAYER_SET_SCN_WINDOW

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；

cmd　　DISP_CMD_LAYER_SET_SCN_WINDOW；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
　　pBuffer[2]为 screen window 信息，指向__disp_rect_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于设置图层 screen window 信息。

- **DEMO**

```
//设置图层 scn window，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                arg[3];
__disp_rect_t    scn_window;
__u32            ret = 0;

scn_window.x = 0;
scn_window.y = 0;
scn_window.width = 800;
scn_window.height = 480;

arg[0] = (__u32)hlay;
arg[1] = (__u32)& scn_window;
ret = ioctl(disphd, DISP_CMD_LAYER_SET_SCN_WINDOW,sel, (void*)arg);
```

DISP_CMD_LAYER_GET_SCN_WINDOW

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_GET_SCN_WINDOW；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
　　pBuffer[2]为 screen window 信息，指向__disp_rect_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于获取图层 screen window 信息。

- **DEMO**

```
//获取图层 scn window，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                arg[3];
__disp_rect_t    scn_window;
__u32            ret = 0;
```

```
arg[0] = (__u32)hlay;
arg[1] = (__u32)& scn_window;
ret = ioctl(disphd,DISP_CMD_LAYER_GET_SCN_WINDOW,sel, (void*)arg);
```

DISP_CMD_LAYER_SET_PARA

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LAYER_SET_PARA；
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
pBuffer[2]为图层参数信息，指向__disp_layer_info_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于设置图层参数。

- **DEMO**

```
//设置图层参数，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];
__disp_layer_info_t layer_para;
__u32               width = 800;
__u32               height = 480;
__u32               ret = 0;

memset(&layer_para, 0, sizeof(__disp_layer_info_t));
layer_para.fb.addr[0]       = (__u32)mem_in; //FB 地址
layer_para.fb.size.width    = width;
layer_para.fb.mode          = DISP_MOD_INTERLEAVED;
layer_para.fb.format        = DISP_FORMAT_ARGB8888;
layer_para.fb.br_swap       = 0;
layer_para.fb.seq           = DISP_SEQ_ARGB;
layer_para.ck_enable        = 0;
layer_para.alpha_en         = 1;
layer_para.alpha_val        = 0xff;
layer_para.pipe             = 1;
layer_para.src_win.x        = 0;
layer_para.src_win.y        = 0;
layer_para.src_win.width    = width;
layer_para.src_win.height   = height;
```

```
layer_para.scn_win.x            = 0;
layer_para.scn_win.y            = 0;
layer_para.scn_win.width        = width;
layer_para.scn_win.height       = height;

arg[0] = (__u32)hlay;
arg[1] = (__u32)& layer_para;
ret = ioctl(disphd, DISP_CMD_LAYER_SET_PARA,sel, (void*)arg);
```

DISP_CMD_LAYER_GET_PARA

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_GET_PARA；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；
　　pBuffer[2]为图层参数信息，指向__disp_layer_info_t 数据结构的指针；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于获取图层参数。

- **DEMO**

```
//获取图层参数，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                       arg[3];
__disp_layer_info_t layer_para;
__u32                   ret = 0;

memset(&layer_para, 0, sizeof(__disp_layer_info_t));

arg[0] = (__u32)hlay;
arg[1] = (__u32)& layer_para;
ret = ioctl(disphd, DISP_CMD_LAYER_GET_PARA,sel, (void*)arg);
```

DISP_CMD_LAYER_TOP

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_TOP；

pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于将图层置顶,其它图层优先级之间的相对关系保持不变。

- **DEMO**

```
//将图层置顶，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];

arg[0]  = hlay;
ioctl(disphd, DISP_CMD_LAYER_TOP, sel, (void*)arg);
```

DISP_CMD_LAYER_BOTTOM

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle　　显示驱动句柄；
cmd　　DISP_CMD_LAYER_BOTTOM；
pBuffer　　pBuffer[0]为显示通道 0/1；
　　pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于将图层置底,其它图层优先级之间的相对关系保持不变。

- **DEMO**

```
//将图层置底，disphd 为显示驱动句柄，sel 为屏 0/1, hlay 为图层句柄
    __u32                   arg[3];

arg[0]  = hlay;
ioctl(disphd, DISP_CMD_LAYER_BOTTOM, sel, (void*)arg);
```

7.2. Scaler 操作接口

DISP_CMD_SCALER_REQUEST

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_SCALER_REQUEST
pBuffer pBuffer[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 scaler 句柄；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于申请 scaler 用于回写。

- **DEMO**

```
//申请 scaler，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];
__hdle              scaler_hdl;

scaler_hdl      = ioctl(disphd, DISP_CMD_SCALER_REQUEST, sel, (void*)arg)
```

DISP_CMD_SCALER_RELEASE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_SCALER_RELEASE
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为要释放的 scaler 句柄；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于释放 scaler。

- **DEMO**

```
//释放 scaler，disphd 为显示驱动句柄，sel 为屏 0/1，scaler_hdl 为句柄
    __u32                   arg[3];

arg[0] = scaler_hdl;
ioctl(disphd, DISP_CMD_SCALER_RELEASE, sel, (void*)arg);
```

DISP_CMD_SCALER_RELEASE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_SCALER_RELEASE
pBuffer pBuffer[0]为显示通道 0/1；

pBuffer[1]为要释放的 scaler 句柄；

7.3. 视频播放接口

DISP_CMD_VIDEO_START

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_VIDEO_START
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于开启视频播放。

- **DEMO**

```
//开启视频播放，disphd 为显示驱动句柄，sel 为屏 0/1,hlay 为图层句柄
    __u32                   arg[3];

arg[0] = hlay;
ioctl(disphd, DISP_CMD_VIDEO_START, sel, (void*)arg);
```

DISP_CMD_VIDEO_STOP

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_VIDEO_STOP
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于停止视频播放，停止之后不能再调用 DISP_CMD_VIDEO_SET_FB 接口。

- **DEMO**

```
//停止视频播放，disphd 为显示驱动句柄，sel 为屏 0/1,hlay 为图层句柄
    __u32                   arg[3];
```

```
arg[0] = hlay;
ioctl(disphd, DISP_CMD_VIDEO_STOP, sel, (void*)arg);
```

DISP_CMD_VIDEO_SET_FB

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_VIDEO_SET_FB
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于设置视频下一帧的 frameBuffer。

- **DEMO**

```
//设置视频的 frameBuffer，disphd 为显示驱动句柄，sel 为屏 0/1,hlay 为图层句柄; mem 是 FB 地址
    __u32                   arg[3];
__disp_video_fb_t    video_fb;

memset(&video_fb, 0, sizeof(__disp_video_fb_t));
video_fb.addr[0] = (__u32)(mem[j] + 800*120*0);
video_fb.addr[1] = (__u32)(mem[j] + 800*120*1);
video_fb.addr[2] = (__u32)(mem[j] + 800*120*2);
video_fb. interlace = 0;

arg[0] = hlay;
arg[1] = (__u32)&video_fb;
ioctl(disphd, DISP_CMD_VIDEO_SET_FB, sel, (void*)arg);
```

DISP_CMD_VIDEO_GET_FRAME_ID

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_VIDEO_GET_FRAME_ID
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；；

- **RETURNS**

  如果成功，则返回当前播放帧的 id；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于获取当前播放帧的 id。

- **DEMO**

```
//获取当前播放帧的 id，disphd 为显示驱动句柄，sel 为屏 0/1,hlay 为图层句柄
    __u32                   arg[3];
__disp_video_fb_t   frame_id;

arg[0] = hlay;
frame_id = ioctl(disphd, DISP_CMD_VIDEO_GET_FRAME_ID, sel, (void*)arg);
```

DISP_CMD_VIDEO_GET_DIT_INFO

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

  hdle　　显示驱动句柄；

  cmd　　DISP_CMD_VIDEO_GET_DIT_INFO

  pBuffer　　pBuffer[0]为显示通道 0/1；

  pBuffer[1]为图层句柄，从 DISP_CMD_LAYER_REQUEST 获得；

  pBuffer[2]为视频 deinterlace 信息，指向__disp_dit_info_t 数据结构的指针

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于获取视频显示时的 deinterlace 信息。如果 maf_enable 为 true 时，需要提供 maf_frag 给 DE，如果 pre_frame_enable 为 true 时，须注意不能释放当前播放帧的前一帧的 Y 数据。

- **DEMO**

```
//获取视频显示时的 deinterlace 信息，disphd 为显示驱动句柄，sel 为屏 0/1,hlay 为图层句柄
    __u32                   arg[3];
__disp_dit_info_t    dit_info;

arg[0] = hlay;
arg[1] = (__u32)&dit_info;
frame_id = ioctl(disphd, DISP_CMD_VIDEO_GET_DIT_INFO, sel, (void*)arg);
```

7.4. LCD 操作接口

DISP_CMD_LCD_ON

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LCD_ON
pBuffer pBuffer[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于打开 LCD 显示。

- **DEMO**

```
//打开 LCD 显示，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];

ioctl(disphd, DISP_CMD_LCD_ON, sel, (void*)arg);
```

DISP_CMD_LCD_OFF

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_LCD_OFF
pBuffer pBuffer[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于关闭 LCD 显示。

- **DEMO**

```
//关闭 LCD 显示，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];

ioctl(disphd, DISP_CMD_LCD_OFF, sel, (void*)arg);
```

7.5. HDMI 接口

DISP_CMD_HDMI_ON

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_HDMI_ON
pBuffer pBuffer[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于打开 HDMI 显示。

- **DEMO**

```
//打开 HDMI 显示，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];

ioctl(disphd, DISP_CMD_HDMI_ON, sel, (void*)arg);
```

DISP_CMD_HDMI_OFF

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_HDMI_OFF
pBuffer pBuffer[0]为显示通道 0/1；

- **RETURNS**

如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于关闭 HDMI 显示。

- **DEMO**

```
//关闭 HDMI 显示，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];

ioctl(disphd, DISP_CMD_HDMI_OFF, sel, (void*)arg);
```

DISP_CMD_HDMI_SET_MODE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

  hdle　　显示驱动句柄；
  cmd　　DISP_CMD_HDMI_SET_MODE
  pBuffer　　pBuffer[0]为显示通道 0/1；
  　　pBuffer[1]为__disp_tv_mode_t　TV 模式；

- **RETURNS**

  如果成功，则返回 DIS_SUCCESS；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于设置 HDMI 模式。

- **DEMO**

```
//设置 HDMI 模式，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];
__disp_tv_mode_t   hdmi_mode;

hdmi_mode = DISP_TV_MOD_1080P_50HZ;
arg[0] = hdmi_mode;
ioctl(disphd, DISP_CMD_HDMI_SET_MODE, sel, (void*)arg);
```

DISP_CMD_HDMI_GET_MODE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

  hdle　　显示驱动句柄；
  cmd　　DISP_CMD_HDMI_GET_MODE
  pBuffer　　pBuffer[0]为显示通道 0/1；

- **RETURNS**

  如果成功，则返回当前 HDMI 模式；如果失败，则返回失败号。

- **DESCRIPTION**

  该函数用于获取当前 HDMI 模式。

- **DEMO**

```
//获取 HDMI 模式，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];
__disp_tv_mode_t   hdmi_mode;

hdmi_mode = (__disp_tv_mode_t)ioctl(disphd, DISP_CMD_HDMI_GET_MODE,
sel, (void*)arg);
```

DISP_CMD_HDMI_GET_HPD_STATUS

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_HDMI_GET_HPD_STATUS
aux 显示通道 0/1；
pBuffer NULL；

- **RETURNS**

如果成功，则返回 HDMI 热拔拔状态(0: plug out; 1: plug in)；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于获取 HDMI HPD 状态。

- **DEMO**

```
//获取 HDMI HPD 状态，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];
__u32                   hdmi_hpd;

Hdmi_hpd = ioctl(disphd, DISP_CMD_TV_GET_DAC_STATUS, sel, (void*)arg);
```

DISP_CMD_HDMI_SUPPORT_MODE

- **PROTOTYPE**

```
int ioctl(int handle, int cmd,int *arg);
```

- **ARGUMENTS**

hdle 显示驱动句柄；
cmd DISP_CMD_HDMI_SUPPORT_MODE
pBuffer pBuffer[0]为显示通道 0/1；
pBuffer[1]为__disp_tv_mode_t HDMI 模式；

- **RETURNS**

如果成功，则返回支持信息(0:不支持；1:支持)；如果失败，则返回失败号。

- **DESCRIPTION**

该函数用于查询 HDMI 电视是否支持该输出模式。

- **DEMO**

```
//查询 HDMI 电视是否支持该输出模式，disphd 为显示驱动句柄，sel 为屏 0/1
    __u32                   arg[3];
__u32                   mode_supported;
__disp_tv_mode_t        hdmi_mode;

hdmi_mode = DISP_TV_MOD_1080P_50HZ;
arg[0] = hdmi_mode;
mode_supported = ioctl(disphd, DISP_CMD_HDMI_SUPPORT_MODE, sel,
(void*)arg);
if(mode_supported == 1)
```

```
{
        __inf("hdmi mode %d supported",hdmi_mode);
}
```

8. Declaration

This(A20 显示模块使用文档) is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

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