目錄

基于rockchip kernel 4.4 sdk, 介绍drm相关配置与概念

本文档地址

https://github.com/markyzq/rockchip_drm_integration_helper

本文档pdf版本下载

libdrm下载

git clone git://anongit.freedesktop.org/mesa/drm

kernel源码地址

https://github.com/rockchip-linux/kernel

测试用例

https://github.com/markyzq/rockchip_drm_demos

RockChip开源官网

http://opensource.rock-chips.com

Rockchip平台一些调试手段

https://markyzq.gitbooks.io/rockchip_debugs

基于drm的显示平台

Xserver

weston/wayland

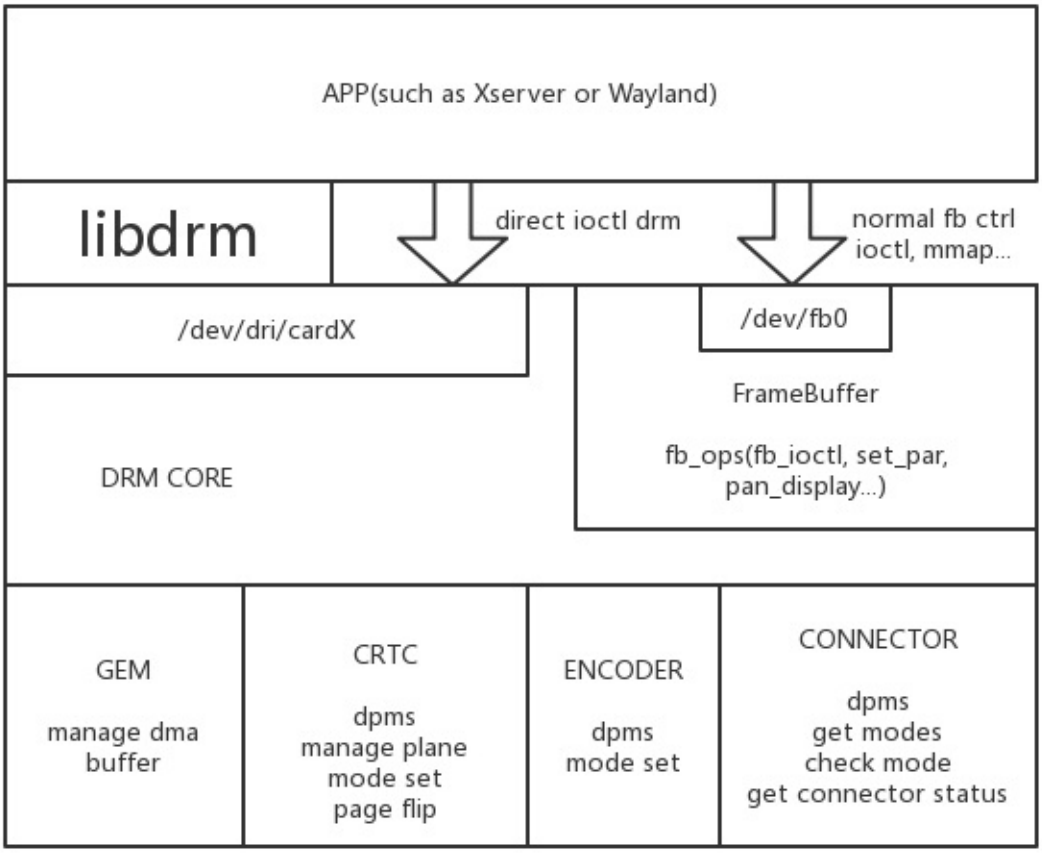
drm_hwcomposer

概述

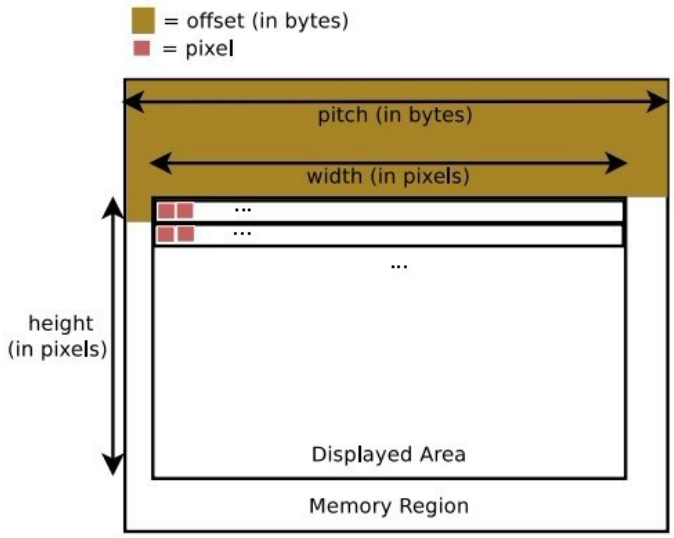
DRM全称是Direct Rendering Manager，管理进行显示输出的, buffer分配, 帧缓冲.

libdrm库提供了一系列友好的控制封装, 使用户可以方便的进行显示的控制, 但并不是 只能通过libdrm库来控制drm, 用户可以直接操作drm的ioctl或者是使用framebuffer的 接口实现显示操作. 后面重点介绍kernel态drm的机制.

以下为drm显示大致框架:

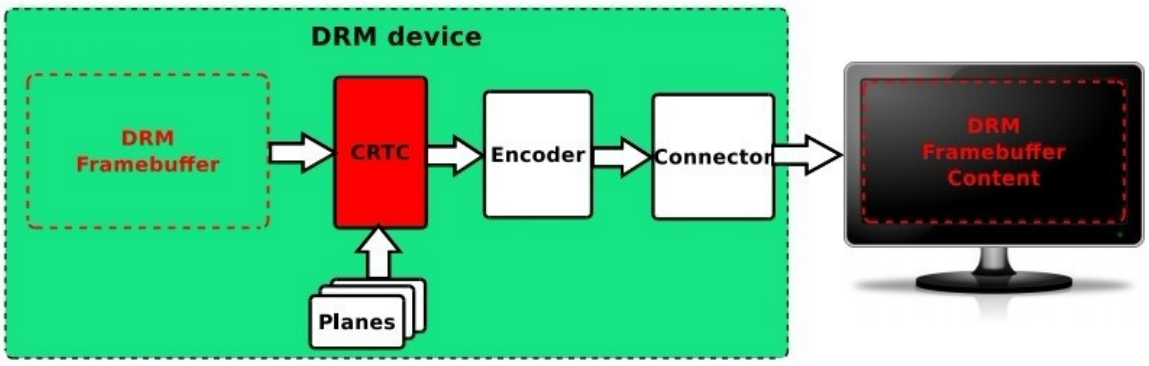


Framebuffer

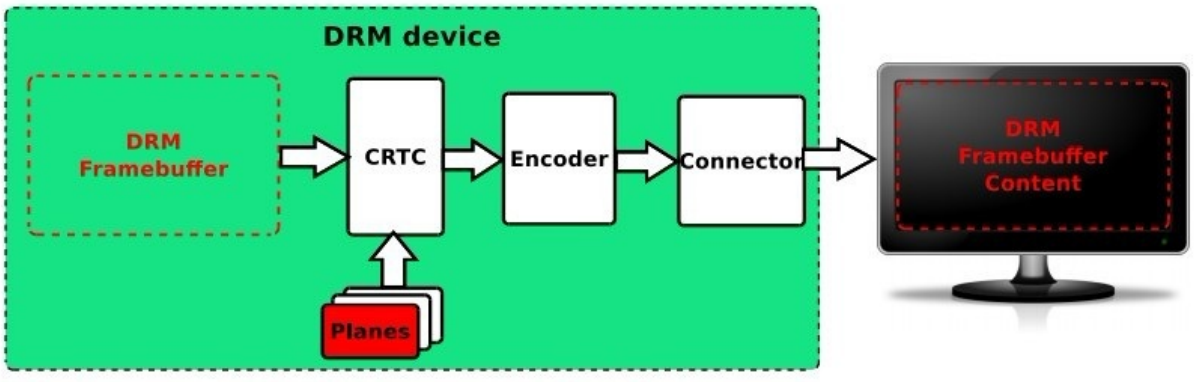



```
struct drm_framebuffer {
[...]
  unsigned int pitches[4];
  unsigned int offsets[4];
  unsigned int width;
  unsigned int height;
[...]
};
```

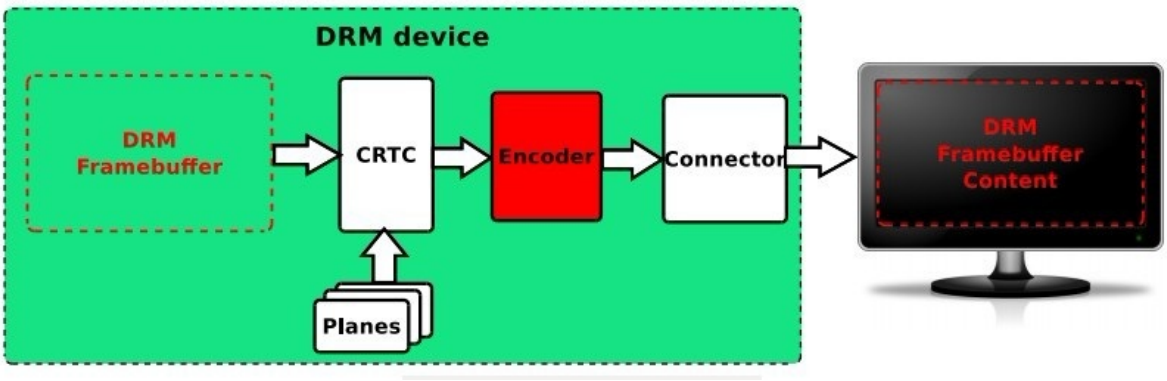
Crtc


DRM device
DRM Framebuffer
CRTC
Encoder
Connector
Planes
DRM Framebuffer Content

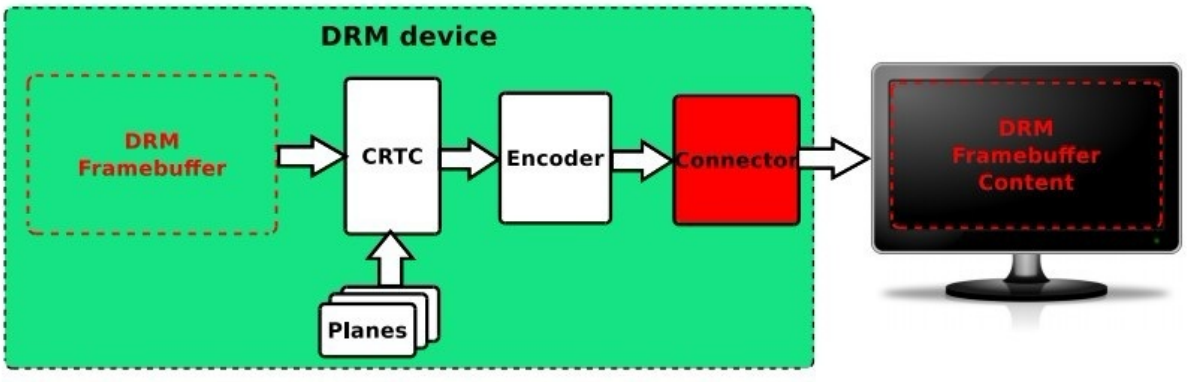
Plane


DRM device
DRM
Framebuffer
CRTC
Encoder
Connector
Planes
DRM
Framebuffer
Content

Encoder


DRM device
DRM
Framebuffer
CRTC
Encoder
Connector
Planes
DRM
Framebuffer
Content

Connector


DRM device
DRM Framebuffer
CRTC
Encoder
Connector
Planes
DRM Framebuffer Content

设备节点

DRM 的设备节点为 "/dev/dri/cardX", X为0-15的数值.

默认使用的是/dev/dri/card0

用户空间内存操作

为了便于说明, 额外定义一个外部内存结构:

```
// drm相关操作需要引用该头文件
#include <drm.h>
struct bo {
   int fd;
   void *ptr;
   size_t size;
   size_t offset;
   size_t pitch;
   unsigned handle;
};
```

获取设备节点:

```
bo->fd = open("/dev/dri/card0", O_RDWR, 0);
```

alloc

```
struct drm_mode_create_dumb arg;
int handle, size, pitch;
int ret;
memset(&arg, 0, sizeof(arg));
arg.bpp = bpp;
arg.width = width;
arg.height = height;
ret = drmIoctl(bo->fd, DRM_IOCTL_MODE_CREATE_DUMB, &arg);
if (ret) {
    fprintf(stderr, "failed to create dumb buffer: %s\n", strerror(errno));
    return ret;
}
bo->handle = arg.handle;
bo->size = arg.size;
bo->pitch = arg.pitch;
```

mmap

```
struct drm_mode_map_dumb arg;
void *map;
int ret;

memset(&arg, 0, sizeof(arg));
arg.handle = bo->handle;
ret = drmIoctl(fd, DRM_IOCTL_MODE_MAP_DUMB, &arg);
if (ret)
    return ret;
map = drm_mmap(0, bo->size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, arg.offset);
if (map == MAP_FAILED)
   return -EINVAL;
bo->ptr = map
```

unmap

```
drm_munmap(bo->ptr, bo->size);
bo->ptr = NULL;
```

free

```
struct drm_mode_destroy_dumb arg;
int ret;

memset(&arg, 0, sizeof(arg));
arg.handle = bo->handle;
ret = drmIoctl(bo->fd, DRM_IOCTL_MODE_DESTROY_DUMB, &arg);
if (ret)
    fprintf(stderr, "failed to destroy dumb buffer: %s\n", strerror(errno));
```

free gem handle:

```
// gem handle减引用操作
struct drm_gem_close args;
memset(&args, 0, sizeof(args));
args.handle = bo->handle;
drmIoctl(bo->fd, DRM_IOCTL_GEM_CLOSE, &args);
```

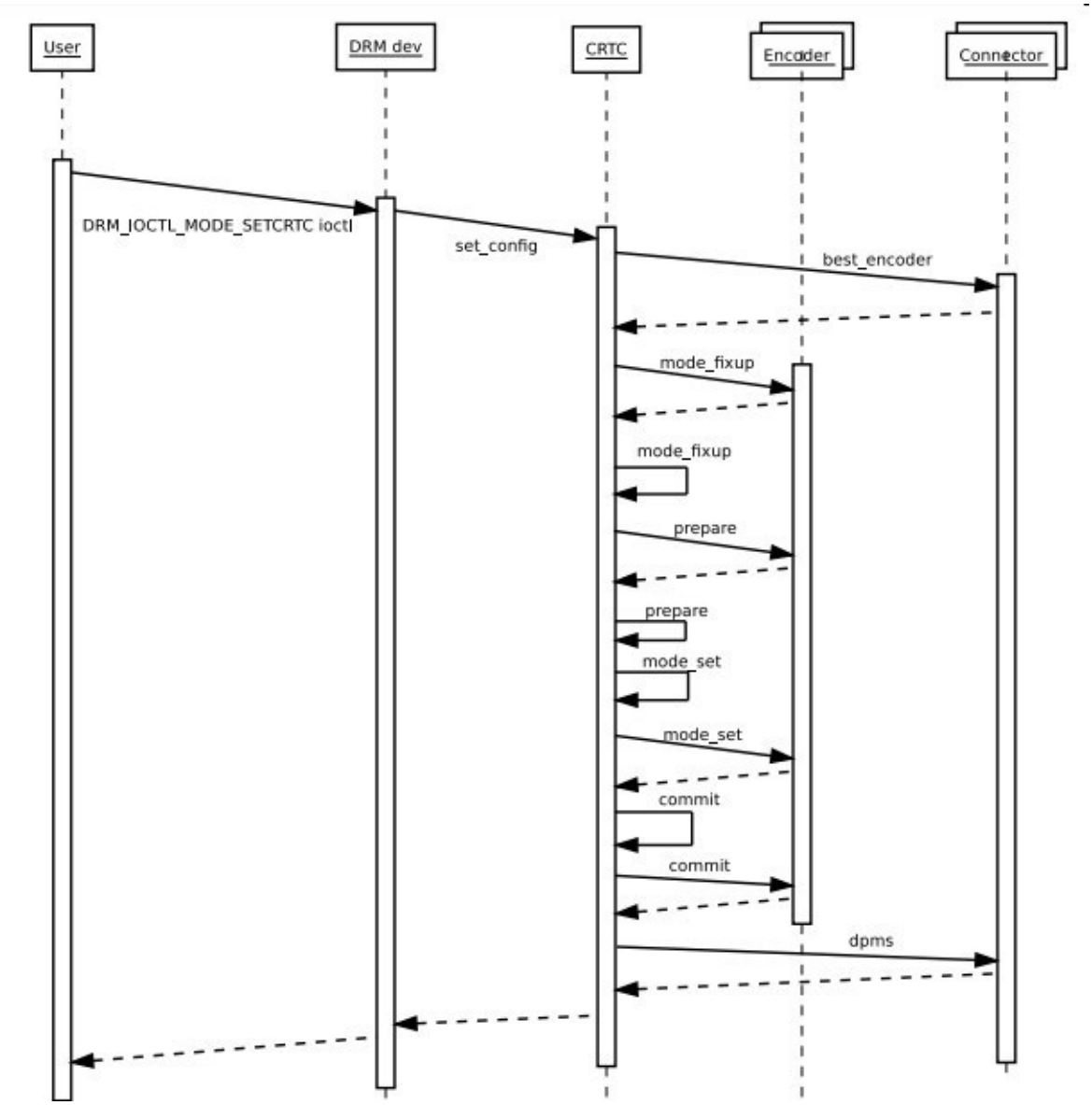
export dmafd

```
int export_dmafd;
ret = drmPrimeHandleToFD(bo->fd, bo->handle, 0, &export_dmafd);
// drmPrimeHandleToFD是会给dma_buf加引用计数的
// 使用完export_dmafd后, 需要使用close(export_dmafd)来减掉引用计数
```

import dmafd

```
ret = drmPrimeFDToHandle(bo->fd, import_dmafd, &bo->handle);
// drmPrimeHandleToFD是会给dma_buf加引用计数的
// 使用完bo->handle后, 需要对handle减引用, 参看free gem handle部分.
```

Drm Mode Setting Sequence Diagram



驱动	文件清单
Core	drivers/gpu/drm/rockchip/rockchip_drm_drv.c
framebuffer	drivers/gpu/drm/rockchip/rockchip_drm_fb.c
gem	drivers/gpu/drm/rockchip/rockchip_drm_gem.c
vop	drivers/gpu/drm/rockchip/rockchip_drm_vop.c drivers/gpu/drm/rockchip/rockchip_vop_reg.c
lvds	drivers/gpu/drm/rockchip/rockchip_lvds.c
rga	drivers/gpu/drm/rockchip/rockchip_drm_rga.c
mipi	drivers/gpu/drm/rockchip/dw-mipi-dsi.c
hdmi	drivers/gpu/drm/rockchip/dw_hdmi-rockchip.c drivers/gpu/drm/bridge/dw-hdmi.c
inno hdmi	drivers/gpu/drm/rockchip/inno_hdmi.c
edp	drivers/gpu/drm/rockchip/analogix_dp-rockchip.c drivers/gpu/drm/bridge/analogix/analogix_dp_core.c drivers/gpu/drm/bridge/analogix/analogix_dp_reg.c
dp	drivers/gpu/drm/rockchip/cdn-dp-core.c drivers/gpu/drm/rockchip/cdn-dp-reg.c

Component Framework

框架代码:

```
drivers/base/component.c
```

功能:

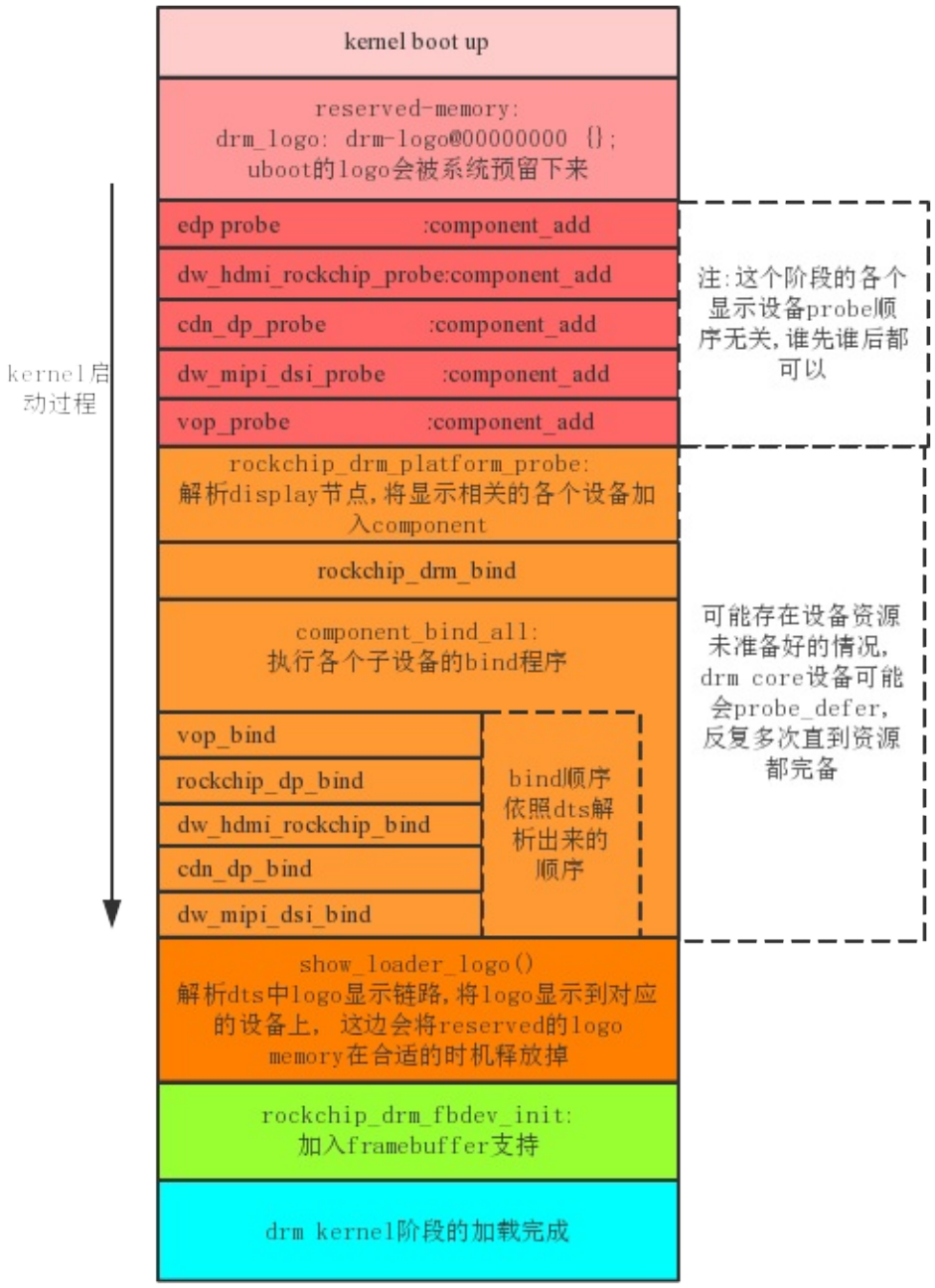
component framework作者Russell King对该机制的描述:

```
Subsystems such as ALSA, DRM and others require a single card-level
device structure to represent a subsystem.  However, firmware tends to
describe the individual devices and the connections between them.

Therefore, we need a way to gather up the individual component devices
together, and indicate when we have all the component devices.

We do this in DT by providing a "superdevice" node which specifies
the components, eg:

    imx-drm {
            compatible = "fsl,drm";
            crtcs = <&ipu1>;
            connectors = <&hdmi>;
    };

The superdevice is declared into the component support, along with the
subcomponents.  The superdevice receives callbacks to locate the
subcomponents, and identify when all components are present.  At this
point, we bind the superdevice, which causes the appropriate subsystem
to be initialised in the conventional way.

When any of the components or superdevice are removed from the system,
we unbind the superdevice, thereby taking the subsystem down.
```

在dts中, 有一个总的设备节点, 所有的子设备信息都通过dts描述关联起来, 这样系统 开机后, 就能统一的管理各个设备.

Rockchip drm主设备

框图:



Probe过程:

基于component框架, 在probe阶段解析dts中各个设备的信息, 加到component match 列表中, 等所有设备加载完毕后, 就会引发master设备的bind.

drm/rockchip为什么需要使用Component?

```
drm下挂了许多的设备, 启动顺序经常会引发各种问题:
  1.一个驱动完全有可能因为等另一个资源的准备, 而probe deferral, 导致顺序不定
  2.子设备没有加载好, 主设备就加载了, 导致设备无法工作
  3.子设备相互之间可能有时序关系,不定的加载顺序,可能带来有些时候设备能工作,有些时候又不能工作
  4.现在编kernel是多线程编译的,编译的前后顺序也会影响驱动的加载顺序.
这时就需要有一个统一管理的机制, 将所有设备统合起来, 按照一个统一的顺序加载,
Display-subsystem正是用来解决这个问题的, 依赖于component的驱动, 通过这个驱动,
可以把所有的设备以组件的形式加在一起, 等所有的组件加载完毕后, 统一进行bind/unbind.
```

以下为rockchip drm master probe阶段component主要逻辑, 为了减小篇幅, 去掉了无关的代码:

```
static int rockchip_drm_platform_probe(struct platform_device *pdev)
{
    for (i = 0;; i++) {
        /* ports指向了vop的设备节点 */
        port = of_parse_phandle(np, "ports", i);
        component_match_add(dev, &match, compare_of, port->parent);
    }
    for (i = 0;; i++) {
        port = of_parse_phandle(np, "ports", i);
        /* 搜查port下的各个endpoint, 将它们也加入到match列表 */
        rockchip_add_endpoints(dev, &match, port);
    }
    return component_master_add_with_match(dev, &rockchip_drm_ops, match);
}

static void rockchip_add_endpoints(...)
{
    for_each_child_of_node(port, ep) {
        remote = of_graph_get_remote_port_parent(ep);
            /* 这边的remote即为和vop关联的输出设备, 即为edp, mipi或hdmi */
        component_match_add(dev, match, compare_of, remote);
    }
}
```

Rockchip drm主设备

设备device tree:

```
display_subsystem: display-subsystem {
    compatible = "rockchip,display-subsystem";
    ports = <&vopl_out>, <&vopb_out>;
    status = "disabled";
};

- compatible: Should be "rockchip,display-subsystem"
- ports: Should contain a list of phandles pointing to display interface port
  of vop devices. vop definitions as defined in
  kernel/Documentation/devicetree/bindings/display/rockchip/rockchip-vop.txt
```

代码位置:

```
drivers/gpu/drm/rockchip/rockchip_drm_drv.c
drivers/gpu/drm/rockchip/rockchip_drm_drv.h
```

主要结构:

```
static struct drm_driver rockchip_drm_driver = {
    .driver_features    = DRIVER_MODESET | DRIVER_GEM |
                  DRIVER_PRIME | DRIVER_ATOMIC |
                  DRIVER_RENDER,
    .preclose        = rockchip_drm_preclose,
    .lastclose        = rockchip_drm_lastclose,
    .get_vblank_counter    = drm_vblank_no_hw_counter,
    .open            = rockchip_drm_open,
    .postclose        = rockchip_drm_postclose,
    .enable_vblank        = rockchip_drm_crtc_enable_vblank,
    .disable_vblank        = rockchip_drm_crtc_disable_vblank,
    .gem_vm_ops        = &rockchip_drm_vm_ops,
    .gem_free_object    = rockchip_gem_free_object,
    .dumb_create        = rockchip_gem_dumb_create,
    .dumb_map_offset    = rockchip_gem_dumb_map_offset,
    .dumb_destroy        = drm_gem_dumb_destroy,
    .prime_handle_to_fd    = drm_gem_prime_handle_to_fd,
    .prime_fd_to_handle    = drm_gem_prime_fd_to_handle,
    .gem_prime_import    = drm_gem_prime_import,
    .gem_prime_export    = drm_gem_prime_export,
    .gem_prime_get_sg_table    = rockchip_gem_prime_get_sg_table,
    .gem_prime_import_sg_table    = rockchip_gem_prime_import_sg_table,
    .gem_prime_vmap        = rockchip_gem_prime_vmap,
    .gem_prime_vunmap    = rockchip_gem_prime_vunmap,
    .gem_prime_mmap        = rockchip_gem_mmap_buf,
#ifdef CONFIG_DEBUG_FS
    .debugfs_init        = rockchip_drm_debugfs_init,
    .debugfs_cleanup    = rockchip_drm_debugfs_cleanup,
#endif
    .ioctls            = rockchip_ioctls,
    .num_ioctls        = ARRAY_SIZE(rockchip_ioctls),
    .fops            = &rockchip_drm_driver_fops,
    .name    = DRIVER_NAME,
    .desc    = DRIVER_DESC,
    .date    = DRIVER_DATE,
    .major    = DRIVER_MAJOR,
    .minor    = DRIVER_MINOR,
};
```

VOP驱动解读

代码位置:

```
drivers/gpu/drm/rockchip/rockchip_drm_vop.c
drivers/gpu/drm/rockchip/rockchip_vop_reg.c
```

结构介绍:

```
struct vop;
// vop 驱动根结构, 一个vop对应一个struct vop结构

struct vop_win;
// 描述图层信息, 一个硬件图层对应一个struct vop_win结构
```

寄存器读写:

```
为了兼容各种不同版本的vop, vop驱动里面使用了寄存器级的抽象, 由一个结构体来保
存抽象关系, 这样主体逻辑只需要操作抽象后的功能定义, 由抽象的读写接口根据抽象
关系写到真实的vop硬件中.

示例:
  static const struct vop_win_phy rk3288_win23_data = {
       .enable = VOP_REG(RK3288_WIN2_CTRL0, 0x1, 4),
  }
  static const struct vop_win_phy rk3368_win23_data = {
       .enable = VOP_REG(RK3368_WIN2_CTRL0, 0x1, 4),
  }
rk3368和rk3288图层的地址分布不同, 但在结构定义的时候, 可以将不同的硬件图层bit
映射到同一个enable功能上, 这样vop驱动主体调用VOP_WIN_SET(vop, win, enable, 1);
时就能操作到真实的vop寄存器了.
```

图层接口:

```
static const struct drm_plane_helper_funcs plane_helper_funcs = {
    // 预先对图层进行处理
    .prepare_fb = vop_plane_prepare_fb,
    // 图层显示完成后的处理
    .cleanup_fb = vop_plane_cleanup_fb,
    // 在显示前进行参数检查
    .atomic_check = vop_plane_atomic_check,
    // 更新图层参数
    .atomic_update = vop_plane_atomic_update,
    // 关闭图层
    .atomic_disable = vop_plane_atomic_disable,
};
```

vop接口:

```
static const struct drm_crtc_helper_funcs vop_crtc_helper_funcs = {
    // 使能vop, 在这里面会将timing配好
    .enable = vop_crtc_enable,
    // 关闭vop
    .disable = vop_crtc_disable,
    // 对timing进行检查修正
    .mode_fixup = vop_crtc_mode_fixup,
    // 在一帧显示开始前做的处理
    .atomic_begin = vop_crtc_atomic_begin,
    // 检查显示的参数
    .atomic_check = vop_crtc_atomic_check,
    // 提交硬件显示
    .atomic_flush = vop_crtc_atomic_flush,
};
```

vop版本	**vop**版本号	基地址
rk3036-vop	VOP_VERSION(2, 2)	0x10118000
rk3288-vop-big	VOP_VERSION(3, 1)	0xff930000
rk3288-vop-lit	VOP_VERSION(3, 1)	0xff940000
rk3368-vop	VOP_VERSION(3, 2)	0xff930000
rk3366-vop-big	VOP_VERSION(3, 4)	0xff930000
rk3366-vop-lit	VOP_VERSION(3, 4)	0xff8f0000
rk3399-vop-big	VOP_VERSION(3, 5)	0xff900000
rk3399-vop-lit	VOP_VERSION(3, 6)	0xff8f0000
rk32xx-vop	VOP_VERSION(3, 7)	0xff8f0000

DRM_IGNORE_IOTCL_PERMIT

```
这个是由我们添加的, 用于解决android上其他进程模块无法访问drm的问题,
目前仅使用在android项目上
```

CONFIG_DRM_FBDEV_EMULATION

使能该功能后, drm将模拟一个framebuffer设备, 可用于基于fbdev的基础显示框架.

CONFIG_DRM_LOAD_EDID_FIRMWARE

```
allow loading an EDID as firmware to override broken monitor

Broken monitors and/or broken graphic boards may send erroneous or no
EDID data. This also applies to broken KVM devices that are unable to
correctly forward the EDID data of the connected monitor but invent
their own fantasy data.

This patch allows to specify an EDID data set to be used instead of
probing the monitor for it. It contains built-in data sets of frequently
used screen resolutions. In addition, a particular EDID data set may be
provided in the /lib/firmware directory and loaded via the firmware
interface. The name is passed to the kernel as module parameter of the
drm_kms_helper module either when loaded
  options drm_kms_helper edid_firmware=edid/1280x1024.bin
or as kernel commandline parameter
  drm_kms_helper.edid_firmware=edid/1280x1024.bin

It is also possible to restrict the usage of a specified EDID data set
to a particular connector. This is done by prepending the name of the
connector to the name of the EDID data set using the syntax
  edid_firmware=[<connector>:]<edid>
such as, for example,
  edid_firmware=DVI-I-1:edid/1920x1080.bin
  edid_firmware=eDP-1:edid/1280x480.bin,DP-2:edid/1920x1080.bin
in which case no other connector will be affected.

The built-in data sets are
Resolution    Name
--------------------------------
1024x768      edid/1024x768.bin
1280x1024     edid/1280x1024.bin
1680x1050     edid/1680x1050.bin
1920x1080     edid/1920x1080.bin

They are ignored, if a file with the same name is available in the
/lib/firmware directory.

The built-in EDID data sets are based on standard timings that may not
apply to a particular monitor and even crash it. Ideally, EDID data of
the connected monitor should be used. They may be obtained through the
drm/cardX/cardX-<connector>/edid entry in the /sys/devices PCI directory
of a correctly working graphics adapter.
```

一级休眠

drm 原生只支持pm runtime 的休眠方式，不支持一级二级休眠，为了支持android的一级 休眠功能，在drm上采用如下的方式进行一级休眠：

```
使能CONFIG_DRM_FBDEV_EMULATION功能

echo 3 > /sys/class/graphics/fb0/blank //进入一级待机
echo 0 > /sys/class/graphics/fb0/blank //退出一级待机

使用fbdev的blank时，kernel会发notify到需要一级待机的设备，如touchscreen,
keyboard等，将这些设备suspend掉
```

二级休眠

```
走pm runtime通路
```

DRM DEVICE TREE 解析

总的drm device结构如下:

```
// rockchip drm core 设备
display_subsystem: display-subsystem {
    compatible = "rockchip,display-subsystem";
    ports = <&vopl_out>, <&vopb_out>;
};

// 显示控制器vop驱动
vopl: vop@ff8f0000 {
    compatible = "rockchip,rk3399-vop-lit";
    vopl_out: port {
        vopl_out_edp: endpoint@1 {
            reg = <1>;
            remote-endpoint = <&edp_in_vopl>;
        };
    };
};

// edp驱动
edp: edp@ff970000 {
    compatible = "rockchip,rk3399-edp";
    ports {
        edp_in: port@0 {
            edp_in_vopl: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&vopl_out_edp>;
            };
        };
        edp_out: port@1 {
            edp_out_panel: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&panel_in_edp>;
            };
        };
    };
};

edp屏驱动
edp_panel: edp-panel {
    compatible = "lg,lp079qx1-sp0v", "panel-simple";
    //通过短字符串"lg,lp079qx1-sp0v"来匹配写死在kernel源码中的timing

    ports {
        panel_in_edp: endpoint {
            remote-endpoint = <&edp_out_panel>;
        };
    };
};
```

屏配置

通用屏驱动:

```
drivers/gpu/drm/panel/panel-simple.c
```

device-tree:

```
Required properties:
  power-supply: regulator to provide the supply voltage

Optional properties:
- compatible: value maybe one of the following
              "simple-panel";
              "simple-panel-dsi";

- ddc-i2c-bus: phandle of an I2C controller used for DDC EDID probing
- enable-gpios: GPIO pin to enable or disable the panel
- backlight: phandle of the backlight device attached to the panel

Required properties when compatible is "simple-panel" or "simple-panel-dsi":
- display-timings: see display-timing.txt for information

Optional properties:
- delay,prepare: the time (in milliseconds) that it takes for the panel to
                 become ready and start receiving video data
- delay,enable: the time (in milliseconds) that it takes for the panel to
                display the first valid frame after starting to receive
                video data
- delay,disable: the time (in milliseconds) that it takes for the panel to
                 turn the display off (no content is visible)
- delay,unprepare: the time (in milliseconds) that it takes for the panel
                   to power itself down complete
- bus-format:


Optional properties when compatible is a dsi devices:
- dsi,flags: dsi operation mode related flags
- dsi,format: pixel format for video mode
- dsi,lanes: number of active data lanes
```

屏配置方式一: 使用短字符串匹配写死的timing

1, 把timings写在drivers/gpu/drm/panel/panel-simple.c中, 直接以短字符匹配, 该方式为upstream推荐的使用方式.

DeviceTree:

```
panel: panel {
    compatible = "cptt,claa101wb01";
    ddc-i2c-bus = <&panelddc>;
    power-supply = <&vdd_pnl_reg>;
    enable-gpios = <&gpio 90 0>;
    backlight = <&backlight>;
    delay,prepare = <10>;
    delay,enable = <10>;
    delay,disable = <10>;
    delay,unprepare = <10>;
};
```

drivers/gpu/drm/panel/panel-simple.c:

```
static const struct drm_display_mode lg_lp097qx1_spa1_mode = {
    .clock = 205210,
    .hdisplay = 2048,
    .hsync_start = 2048 + 150,
    .hsync_end = 2048 + 150 + 5,
    .htotal = 2048 + 150 + 5 + 5,
    .vdisplay = 1536,
    .vsync_start = 1536 + 3,
    .vsync_end = 1536 + 3 + 1,
    .vtotal = 1536 + 3 + 1 + 9,
    .vrefresh = 60,
};
static const struct panel_desc lg_lp097qx1_spa1 = {
    .modes = &lg_lp097qx1_spa1_mode,
    .num_modes = 1,
    .size = {
        .width = 320,
        .height = 187,
    },
};
static const struct of_device_id platform_of_match[] = {
    [...]
    }, {
      .compatible = "lg,lp097qx1-spa1",
      .data = &lg_lp097qx1_spa1,
    }, {
    [...]
}
```

屏配置方式二: 直接将timing写在dts文件中

2, 把使用display-timings结构, 直接把timing写在dts文件中, 这类适用于简单的屏配置

```
    panel: panel {
        compatible = "simple-panel-dsi";
        ddc-i2c-bus = <&panelddc>;
        power-supply = <&vdd_pnl_reg>;
        enable-gpios = <&gpio 90 0>;
        backlight = <&backlight>;
        dsi,flags = <MIPI_DSI_MODE_VIDEO |
                 MIPI_DSI_MODE_VIDEO_BURST |
                 MIPI_DSI_MODE_VIDEO_SYNC_PULSE>;
        dsi,format = <MIPI_DSI_FMT_RGB888>;
        dsi,lanes = <4>;
        delay,prepare = <10>;
        delay,enable = <10>;
        delay,disable = <10>;
        delay,unprepare = <10>;

        display-timings {
            native-mode = <&timing0>;
            timing0: timing0 {
                clock-frequency = <160000000>;
                hactive = <1200>;
                vactive = <1920>;
                hback-porch = <21>;
                hfront-porch = <120>;
                vback-porch = <18>;
                vfront-porch = <21>;
                hsync-len = <20>;
                vsync-len = <3>;
                hsync-active = <0>;
                vsync-active = <0>;
                de-active = <0>;
                pixelclk-active = <0>;
            };
        };
    };
```

屏配置方式三: 使用edid

不填写任何timing, 直接使用edid来获取timing

```
panel: panel {
    compatible = "simple-panel-dsi";
    //compatible = "simple-panel";
    ddc-i2c-bus = <&panelddc>;
    power-supply = <&vdd_pnl_reg>;
    enable-gpios = <&gpio 90 0>;
    backlight = <&backlight>;
    delay,prepare = <10>;
    delay,enable = <10>;
    delay,disable = <10>;
    delay,unprepare = <10>;
};
```

Document:

```
Documentation/devicetree/bindings/display/panel/simple-panel.txt
```

步骤一

参考《屏配置的几个方式》章节，先使用edid的方式，在dts里面写入:

```
panel: panel {
    compatible = "simple-panel";
    ddc-i2c-bus = <&panelddc>;
    power-supply = <&vdd_pnl_reg>;
    enable-gpios = <&gpio 90 0>;
    backlight = <&backlight>;
};
```

如果屏的edid正常， power, gpio, backlight配置正常，应该就能看到显示了。

步骤二

由于屏的edid是有机率损坏的，做产品是有风险的，所以建议把timing写死到代码里面:

完成步骤一后，可以从uboot串口上获取当前的屏的信息:

```
Maximum visible display size: 26 cm x 17 cm
Power management features: no active off, no suspend, no standby
Estabilished timings:
Standard timings:
        2400x1600       59 Hz (detailed)
Monitor name: LQ123P1JX31
Detailed mode clock 252750 kHz, flags[a]
    H: 2400 2448 2480 2560
    V: 1600 1603 1613 1646
bus_format: 1009

参考《屏配置的几个方式》章节, 屏配置方式一, 将该timing写死到kernel中.
```

LVDS屏配置

驱动:

```
drivers/gpu/drm/rockchip/rockchip_lvds.c
```

参考配置:

```
arch/arm64/boot/dts/rockchip/rk3368-sheep-lvds.dts
```

文档

```
Documentation/devicetree/bindings/video/rockchip-lvds.txt:

Required properties:
- compatible: matching the soc type, one of
    - "rockchip,rk3288-lvds";
    - "rockchip,rk33xx-lvds";

- reg: physical base address of the controller and length
    of memory mapped region.
- reg-names: the name to indicate register. example:
    - "mipi_lvds_phy": lvds phy register, this's included in the MIPI phy module
    - "mipi_lvds_ctl": lvds control register, this's included in the MIPI
        controller module
- clocks: must include clock specifiers corresponding to entries in the
    clock-names property.
- clock-names: must contain "pclk_lvds"

- avdd1v0-supply: regulator phandle for 1.0V analog power
- avdd1v8-supply: regulator phandle for 1.8V analog power
- avdd3v3-supply: regulator phandle for 3.3V analog power

- rockchip,grf: phandle to the general register files syscon

- rockchip,data-mapping: should be "vesa" or "jeida",
    This describes how the color bits are laid out in the
    serialized LVDS signal.
- rockchip,data-width : should be <18> or <24>;
- rockchip,output: should be "rgb", "lvds" or "duallvds",
    This describes the output face.

Optional properties
- pinctrl-names: must contain a "default" entry.
- pinctrl-0: pin control group to be used for this controller.
- pinctrl-1: pin control group to be used for gpio.

Required nodes:
```

```
The lvds has two video ports as described by
    Documentation/devicetree/bindings/media/video-interfaces.txt.
Their connections are modeled using the OF graph bindings specified in
    Documentation/devicetree/bindings/graph.txt.

- video port 0 for the VOP inputs
- video port 1 for either a panel or subsequent encoder

Example:

For Rockchip RK3288:

    lvds: lvds@ff96c000 {
        compatible = "rockchip,rk3288-lvds";
        rockchip,grf = <&grf>;
        reg = <0xff96c000 0x4000>;
        clocks = <&cru PCLK_LVDS_PHY>;
        clock-names = "pclk_lvds";
        avdd1v0-supply = <&vdd10_lcd>;
        avdd1v8-supply = <&vcc18_lcd>;
        avdd3v3-supply = <&vcca_33>;
        rockchip,data-mapping = "jeida";
        rockchip,data-width = <24>;
        rockchip,output = "rgb";
        ports {
            #address-cells = <1>;
            #size-cells = <0>;

            lvds_in: port@0 {
                reg = <0>;

                lvds_in_vopb: endpoint@0 {
                    reg = <0>;
                    remote-endpoint = <&vopb_out_lvds>;
                };
                lvds_in_vopl: endpoint@1 {
                    reg = <1>;
                    remote-endpoint = <&vopl_out_lvds>;
                };
            };

            lvds_out: port@1 {
                reg = <1>;

                lvds_out_panel: endpoint {
                    remote-endpoint = <&panel_in>;
                };
            };
        };
    };

For Rockchip RK3368:
```

```
    lvds: lvds@ff968000 {
        compatible = "rockchip,rk33xx-lvds";
        reg = <0x0 0xff968000 0x0 0x4000>, <0x0 0xff9600a0 0x0 0x20>;
        reg-names = "mipi_lvds_phy", "mipi_lvds_ctl";
        clocks = <&cru PCLK_DPHYTX0>, <&cru PCLK_MIPI_DSI0>;
        clock-names = "pclk_lvds", "pclk_lvds_ctl";
        power-domains = <&power RK3368_PD_VIO>;
        rockchip,grf = <&grf>;
        pinctrl-names = "lcdc", "gpio";
        pinctrl-0 = <&lcdc_lcdc>;
        pinctrl-1 = <&lcdc_gpio>;

        ports {

        ...

        };
    };
```

开机logo device tree配置说明

参考配置: arch/arm64/boot/dts/rockchip/rk3399-android.dtsi

DeviceTree 解析:

```
reserved-memory {

    //在reserved memory划一块内存做为logo使用
    drm_logo: drm-logo@00000000 {
        compatible = "rockchip,drm-logo";

        //size填0, uboot将会根据实际logo占用buffer的大小分配
        //用户无需填写
        reg = <0x0 0x0 0x0 0x0>;
    };
};

&display_subsystem {

    //drm驱动将解析drm_logo的buffer, 用于logo显示
    memory-region = <&drm_logo>;
    route {
        //每一组route_xxx代表一路显示输出
        route_hdmi: route-hdmi {

            //在uboot loader阶段所使用的图片, 名称可根据resource.img打包的logo图片名称定义.
            //当该属性留空或者指定的图片找不到时, uboot将不显示
            logo,uboot = "logo.bmp";

            //在uboot loader阶段所使用的图片, 名称可根据resource.img打包的logo图片名称定义.
            //当该属性留空或者指定的图片找不到时, uboot将不显示
            logo,kernel = "logo_kernel.bmp";

            //支持两种显示模式: "fullscreen"全屏和"center'居中
            logo,mode = "fullscreen";

            //充电logo, 支持两种显示模式: "fullscreen"全屏和"center'居中
            charge_logo,mode = "center";

            //指定具体的显示通路, 如下, 为hdmi使用vopb输出
            connect = <&vopb_out_hdmi>;
        };
        route_mipi: route-mipi {
            logo,uboot = "logo_mipi.bmp";
            logo,kernel = "logo_kernel_mipi.bmp";
            logo,mode = "fullscreen";
            charge_logo,mode = "center";
            connect = <&vopb_out_mipi>;
        };
        route_edp: route-edp {
```

```
            logo,uboot = "logo_edp.bmp";
            logo,kernel = "logo_kernel_edp.bmp";
            logo,mode = "fullscreen";
            charge_logo,mode = "center";
            connect = <&vopb_out_edp>;
        };
    };
};
```

开机**logo** 双屏异显配置

arch/arm64/boot/dts/rockchip/rk3399-android.dtsi

两路输出显示不同的内容:

如下, route_hdmi使用logo_hdmi.bmp, route_mipi使用logo_mipi.bmp

```
&display_subsystem {
    route {
        route_hdmi: route-hdmi {

            logo,uboot = "logo_hdmi.bmp";
            logo,kernel = "logo_kernel_hdmi.bmp";
            connect = <&vopl_out_hdmi>;
        };
        route_mipi: route-mipi {
            logo,uboot = "logo_mipi.bmp";
            logo,kernel = "logo_kernel_mipi.bmp";
            logo,mode = "fullscreen";
            charge_logo,mode = "center";
            connect = <&vopb_out_mipi>;
        };
    };
};
```

hdmi和mipi就将显示不同的图片内容.

两路输出同时显示:

不同route使用不同的vop即可实现显示的独立. 例如:

```
route_hdmi的connect配置成vopl_out_hdmi,
route_mipi的connect配置成vopb_out_hdmi
```

hdmi和edp就将使用不同的vop独立显示

多屏抢占及热拔插

```
&display_subsystem {
  route {
      route_hdmi: route-hdmi {
          connect = <&vopb_out_hdmi>;
      };

      route_mipi: route-mipi {
          connect = <&vopb_out_mipi>;
      };

      route_edp: route-edp {
          connect = <&vopl_out_edp>;
      };
  };
};
```

抢占

route的节点是有顺序优先关系的, 如上, route_hdmi在route_mipi之前, 且它们都使用vopb做为显示输出, 当hdmi和mipi同接入时, hdmi会先将vopb抢走, 这样mipi就分配不到vop了, 现象为: hdmi显示, mipi不显示

热拔插

如上抢占内容可知, 当hdmi插入时, 现象为hdmi显示, mipi不显示.

但当hdmi处于拔出状态时, route_hdmi这一路将不会工作, 也即可以实现: hdmi不显示, mipi显示.

由此实现同一配置, 插入hdmi和拔出hdmi启动过程通路状态不同.

cat /sys/kernel/debug/dri/0/summary

```
vop name: ff900000.vop status=active
Display mode: 1280x1024 fps[75] clk[135000] type[64] flag[5]
    H: 1280 1296 1440 1688
    V: 1024 1025 1028 1066
win0-0: status=disabled
win1-0: status=disabled
win2-0: status=active
    format: AB24 little-endian (0x34324241)
    zpos: 0
    src: pos[0x0] rect[1280x1024]
    dst: pos[0x0] rect[1280x1024]
    buf[0]: addr: 0x000000000370e000 pitch: 5120 offset: 0
win2-0: status=disabled
win2-1: status=disabled
win2-2: status=disabled
win3-0: status=disabled
win3-0: status=disabled
win3-1: status=disabled
win3-2: status=disabled
vop name: ff8f0000.vop status=disabled
```

cat /sys/kernel/debug/dri/0/mm_dump

```
0x0000000000000000-0x00000000004f2000: 5185536: used
0x00000000004f2000-0x0000000000d0a000: 8486912: used
0x0000000000d0a000-0x0000000001522000: 8486912: used
0x0000000001522000-0x0000000001d0b000: 8294400: used
0x0000000001d0b000-0x00000000024f4000: 8294400: used
0x00000000024f4000-0x0000000002d0c000: 8486912: used
0x0000000002d0c000-0x0000000002d0d000: 4096: used
0x0000000002d0d000-0x0000000002d0e000: 4096: used
0x0000000002d0e000-0x000000000320e000: 5242880: used
0x000000000320e000-0x000000000370e000: 5242880: used
0x000000000370e000-0x0000000003c0e000: 5242880: used
0x0000000003c0e000-0x0000000003d52000: 1327104: used
0x0000000003d52000-0x0000000004d24000: 16588800: free
0x0000000004d24000-0x000000000550d000: 8294400: used
0x000000000550d000-0x0000000005cf6000: 8294400: used
0x0000000005cf6000-0x00000000064df000: 8294400: free
0x00000000064df000-0x00000000080ff000: 29491200: used
0x00000000080ff000-0x0000000008101000: 8192: used
0x0000000008101000-0x00000000088ea000: 8294400: used
0x00000000088ea000-0x00000000090d3000: 8294400: used
0x00000000090d3000-0x00000000098bc000: 8294400: used
0x00000000098bc000-0x000000000a0a5000: 8294400: used
0x000000000a0a5000-0x0000000100000000: 4126519296: free
total: 4294967296, used 143564800 free 4151402496
```

modetest使用指南

modetest是libdrm源码自带的调试工具, 可以对drm进行一些基础的调试.

获取:

android平台:

```
mmm external/libdrm/tests
```

linux平台 - ARM64

```
git clone git://anongit.freedesktop.org/mesa/drm && cd drm
//ARM64
CC=aarch64-linux-gnu-gcc ./autogen.sh --host=aarch64-linux --disable-freedreno --disabl
e-cairo-tests --enable-install-test-programs
//ARM32
CC=arm-linux-gnueabihf-gcc ./autogen.sh --host=arm-linux --disable-freedreno --disable-
cairo-tests --enable-install-test-programs
make -j4 && make install DESTDIR=`pwd`/out
```

modetest帮助信息:

```
(shell)# modetest -h
usage: modetest [-cDdefMPpsCvw]
 Query options:
    -c    list connectors
    -e    list encoders
    -f    list framebuffers
    -p    list CRTCs and planes (pipes)
 Test options:
    -P <crtc_id>:<w>x<h>[+<x>+<y>][*<scale>][@<format>]    set a plane
    -s <connector_id>[,<connector_id>][@<crtc_id>]:<mode>[-<vrefresh>][@<format>]    se
t a mode
    -C    test hw cursor
    -v    test vsynced page flipping
    -w <obj_id>:<prop_name>:<value>    set property
 Generic options:
    -d    drop master after mode set
    -M module    use the given driver
    -D device    use the given device
    Default is to dump all info.
```

使用案例:

modetest不带参(有非常多的打印,这边截取部分关键的):

```
//由于rockchip driver的一些配置未upstream到libdrm上, 所以从libdrm upstream
//下载编译的modetest默认不带rockchip支持, 需要在使用的时候加个-M rockchip.
(shell)# modetest -M rockchip
Encoders:
id    crtc    type    possible crtcs    possible clones
71    27    TMDS    0x00000001    0x00000000
73    0    TMDS    0x00000002    0x00000000
Connectors:
id    encoder    status        name        size (mm)    modes    encoders
72    71    connected    HDMI-A-1          410x260        19    71
  modes:
    name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1440x900 60 1440 1520 1672 1904 900 903 909 934 flags: nhsync, pvsync; type: preferre
d, driver
  1280x1024 75 1280 1296 1440 1688 1024 1025 1028 1066 flags: phsync, pvsync; type: dri
ver
  [...]
74    0    disconnected    DP-1              0x0        0    73
CRTCs:
id    fb    pos    size
27    0    (0,0)    (1280x1024)
64    0    (0,0)    (0x0)
Planes:
id    crtc    fb    CRTC x,y    x,y    gamma size    possible crtcs
23    0    0    0,0        0,0    0          0x00000001
[...]
```

如上modetest -M rockchip的信息,我们知道当前系统有两个vop, 有两个输出设备:

测试显示输出: 测试显示输出时要把当前系统其他的显示关掉, 因为drm只能允许一个显示输出程序.

```
android: adb shell stop
linux ubuntu: service lightdm stop
```

显示输出命令

```
(shell)# modetest -M rockchip -s 72@27:1440x900 -v
setting mode 1440x900-60Hz@XR24 on connectors 72, crtc 27
freq: 60.53Hz

屏幕上即可看到闪烁的彩条显示,
如需使用dp输出,将命令中的connector的id换成dp的即可.
如需使用另一个crtc输出, 将命令中的crtc的id换成另一个crtc的id即可
如需使用别的分辨率输出, 将命令中1440x900换成connectors modes里面别的分辨率即可
```

设置drm的调试log等级:

```
sys结点位置: /sys/module/drm/parameters/debug

debug:Enable debug output, where each bit enables a debug category.
    Bit 0 (0x01) will enable CORE messages (drm core code)
    Bit 1 (0x02) will enable DRIVER messages (drm controller code)
    Bit 2 (0x04) will enable KMS messages (modesetting code)
    Bit 3 (0x08) will enable PRIME messages (prime code)
    Bit 4 (0x10) will enable ATOMIC messages (atomic code)
    Bit 5 (0x20) will enable VBL messages (vblank code) (int)

示例:  echo 0x0c > /sys/module/drm/parameters/debug
     打开了KMS, PRIME这些的打印, kernel驱动中使用DRM_DEBUG_KMS和
     DRM_DEBUG_PRIME打印的信息都可以打印出来
```

查看显示时钟:

```
(shell)# cat /sys/kernel/debug/clk/clk_summary | grep vop
  dclk_vop0                     2            2   135000000         0 0
  dclk_vop1                     0            1           0         0 0
  aclk_vop0                     2            3   594000000         0 0
  aclk_vop1                     0            2   594000000         0 0
  hclk_vop0                     2            3   198000000         0 0
  hclk_vop1                     0            2   198000000         0 0
```

需要关注的显示时钟为:

dclk_vop:

即pixel clock, 像素时钟, 该时钟由具体的显示timing决定, 如果dclk不正确, 可能导致fps不对或直接不显示. edp, mipi, lvds等显示接口对应dclk的容忍性较好, 有些偏差也不影响正常显示. 但hdmi, dp等高清显示接口, 是有严格要求的, 这类显示接口的频率要给的很精准.

aclk_vop:

如果该时钟频率太低, 可能会导致显示出现抖动, 另外如果aclk 没有使能的话, 访问vop的寄存器也可能引发总线挂死

hclk_vop:

如果该时钟未使能, 不能访问vop的寄存器, 一但访问vop寄存器, 会造成总线挂死.

```
(shell)# cat /sys/kernel/debug/gpio

GPIOs 0-31, platform/pinctrl, gpio0:
GPIOs 32-63, platform/pinctrl, gpio1:
GPIOs 64-95, platform/pinctrl, gpio2:
 gpio-90  (                    |bt_default_wake     ) out hi
GPIOs 96-127, platform/pinctrl, gpio3:
 gpio-98  (                    |enable             ) out hi
 gpio-111 (                    |mdio-reset         ) out hi
GPIOs 128-159, platform/pinctrl, gpio4:
```

drm panel驱动默认使用enable做为屏的gpio使能脚, 所以这边重点看一下enable的状态, 是否与预期的一致.

不管hdmi接没接， 强行使能hdmi

```
echo on > /sys/devices/platform/display-subsystem/drm/card0/card0-HDMI-A-1/status
```

关闭hdmi

```
echo off > /sys/devices/platform/display-subsystem/drm/card0/card0-HDMI-A-1/status
```

监测hdmi插拔，正常情况以就是在这个状态

```
echo detect > /sys/devices/platform/display-subsystem/drm/card0/card0-HDMI-A-1/status
```

Reference Documents

- brezillon-drm-kms.pdf
- gfx-docs
- LWN article about dma_buf sharing
- DMA Buffer Sharing FrameWork
- Mainline Explicit Fencing