LINUX系统DRM平台讲解

|  |  |  |
| --- | --- | --- |
| 文件状态：  🗹 草稿  🞎 正式发布  🞎 正在修改 | 作 者： | John |
| 当前版本： | V0.1 |
| 完成日期： |  |
| 审 定： |  |

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| V0.1 |  |  | 2015.11.15 | 新建 |
|  |  |  |  |  |
|  |  |  |  |  |

#### 概述

本文档的目的是帮助LINUX系统的图形图像处理DRM平台的组件及其各个组件的功能介绍。DRM平台广泛应用在video，3D图像加速处理，视频采集等用途，本文档描述DRM平台在LINUX内核的框架结构，以及描述DRM平台各个模块直接的相互关系。

#### DRM平台框架介绍



DRM 平台架设在文件系统之下，对文件系统提供一致性的接口，对下通过DRM mid-layer中间层提供对三个重要模块的管理，TTM，GEM，KMS组件。下面介绍三个重要部件的功能与相互关系。

1. TTM模块

LINUX内核对于图像处理的缓存提供一致性接口，TTM模块对于所有显存提供通用的平台功能，但是这样作出的模块代码庞大，不利于驱动开发，现在大部分的图像内存管理，都采用GEM的模式。

1. GEM模块

GEM的构造必TTM要简单，并未给所有的显卡提供一致性接口，只是在通用代码与库直接提供共享通道，提供给用户可以使用的API函数，但是GEM并没有video ram的管理能力。

GEM模块主要提供以下几点重要功能：

1. GEM的主要功能内存分配与释放
2. 命令执行
3. 命令执行间隙管理

在GEM初始化之后KMS初始化，UMA设备通常提供初始化frambuffer空

间和连续的内存区域。可以通过ioctl中do\_mmap函数进行内存的控制，Dumb Gem模块一定程度上缓和了标准API在图像扫描输出上的问题，KMS 框架内存经常使用scanout内存对象。

通过KMS的CRTC的ioctl控制命令，得到用户的创建framebuffer命令，GEM则调用struct drm\_framebuffer \*(\*fb\_create)(struct drm\_device \*dev………….，创建frambuffer，本文档的图像处理使用GEM的内存管理模块。

1. KMS模块

KMS设备可以抽象为平面，CTRC，ENCODER编码与CONNECTOR连接器，在初始化模式设定是必须初始化以上几个组件。

CRTC保护了一个指向scanout buffer的指针，crtc包含了几个重要的内容，指向video内存的指针，显示模式，在显存中要显示的xy坐标点以及被多个crtc测量配置的显示片段。处理器在crtc组件中，通过fram buffer中可以扫描出xy坐标，显示模式以及与crtc对接的连接器驱动的指针，调度翻页机制是crtc的一个重要功能，应用请求翻页是，drm内核校验新页面的内存对于crtc scan out出的内存是匹配的情况下，才调用翻页机制处理新页面。如果需要翻页操作更快更同步需要使能中断的机制操作。plane代表了一个图片的基本要素，在frambuffer中的图像正方形有字体宽度、高度以及参数值的xy坐标确定，目的正方形有crtc中的各个变量决定，源和目的的scale即比例有设备决定

ENCODER模块用于从CRTC中获取像素点数据并转化成核实后的格式提供给连接器驱动即CONNECTOR, 可以有多个encoder，用于收发, encoder模块必须与crtc相关联使用，在初始化阶段是非关联的，应用必须将两者指定的关联起来.

CONNECTOR对于像素数据是最终的目的地，用于直接配置外部设备，像LCD之类的设备，其必须关联ENCODER模块，从而在DRM系统中成为插件。CONNECTOR可以支持的设备接口，例如LVDS，TV，VGA等接口。另外DPRM模块多用于智能电源管理模块，由于本次设计不涉及，所以不考虑。

DRM核包含了一个中间层组件，中间层提供了crt，encoder，connector函数的执行以及请求的预处理和底层驱动函数的调用，为了使用中间层，必须使用以下几个函数去初始化它drm\_crtc\_helper\_add, drm\_encoder\_helper\_add and drm\_connector\_helper\_add。为了使用中间层，必须将KMS组件中的crtc，encoder和connector初始化完整。

#### DRM平台架构初始化流程

设备通过函数module\_platform\_driver(zynqlcdc\_encoder\_driver)注册平台驱动结构体，其结构体如下定义：

static struct platform\_driver zynqlcdc\_encoder\_driver = {

.driver = {

.name = "zynqlcdc",

.owner = THIS\_MODULE,

.of\_match\_table = zynqlcdc\_encoder\_of\_match,},

.probe = zynqlcdc\_platform\_probe,

.remove = zynqlcdc\_platform\_remove,};

static const struct of\_device\_id zynqlcdc\_encoder\_of\_match[] = {

{

.compatible = "avnet,ali3-controller-1.00.a",

},

{},

};

该结构的初始化实质函数是zynqlcdc\_platform\_probe，该函数是在有如下的变量声明后才调用probe，

static const struct of\_device\_id zynqlcdc\_encoder\_of\_match[] = {

{

.compatible = "avnet,ali3-controller-1.00.a",

},

{},

};

MODULE\_DEVICE\_TABLE(of, zynqlcdc\_encoder\_of\_match);

通过变量.compatible的字符串一致，则调用设备树中的相关定义节点。这点与老的设备驱动程序不同，老的需要声明平台设备结构体，然后注册，内核中有同名的驱动模块，则调用probe函数。目前采用设备树，相关设备信息从设备树中获得。下面各个模块的初始化流程：

zynqlcdc\_platform\_probe

dma\_request\_slave\_channel(&pdev->dev, "ali3\_video")//查询设备树ali3\_video初始化//vdma，视频图像用到vdma方式

drm\_platform\_init//初始化drm平台结构体

drm\_dev\_alloc//分配内存

drm\_dev\_register//注册drm结构体，并初始化大量子结构

drm\_core\_check\_feature//检查当前的宏定义，是否是当前模块特性

dev->driver->load//此函数调用极为重要，此load函数是在此处定义

static struct drm\_driver zynqlcdc\_driver = {

.driver\_features= DRIVER\_BUS\_PLATFORM |

DRIVER\_MODESET | DRIVER\_GEM,

.load= zynqlcdc\_load,//drm设备注册时，调用钩子函数初始化zynq的drm平台各个drm组件

zynqlcdc\_load

drm\_mode\_config\_init//将drm各个组件的属性添加到list中

zynqlcdc\_mode\_config\_init//挂接钩子函数，用于后续初始化drm平台的cma结构

dev->mode\_config.funcs = &zynqlcdc\_mode\_config\_funcs

static struct drm\_mode\_config\_funcs zynqlcdc\_mode\_config\_funcs = {

.fb\_create = drm\_fb\_cma\_create,//初始化fb结构

.output\_poll\_changed = zynqlcdc\_output\_poll\_changed,

};

zynqlcdc\_crtc\_create//初始化KMS结构的CRTC结构体

drm\_crtc\_init(dev, crtc, &zynqlcdc\_crtc\_funcs)//初始化CRTC结构体，并插入到KMS

//的CRTC设备列表中

drm\_mode\_object\_get

list\_add\_tail//增加到设备链表中

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

static struct drm\_crtc\_funcs zynqlcdc\_crtc\_funcs = {

.set\_config = drm\_crtc\_helper\_set\_config//对crtc的设备helper函数进行配置

.destroy = zynqlcdc\_crtc\_destroy,

};

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

drm\_crtc\_helper\_add(crtc, &zynqlcdc\_crtc\_helper\_funcs)//增加CRTC helper钩子函数

zynqlcdc\_encoder\_create//encoder成员创建

drm\_encoder\_init

drm\_encoder\_helper\_add

zynqlcdc\_connector\_init//此处需要列出所有与当前的encoder可以匹配的connector

drm\_fbdev\_cma\_init//初始化fbdev结构体

fbdev\_cma->fb\_helper.funcs = &drm\_fb\_cma\_helper\_funcs//挂接重要的fix,var变量

//初始化函数

drm\_fb\_helper\_single\_add\_all\_connectors//为fbdev结构体增加所有的connector

drm\_fb\_helper\_initial\_config//

drm\_setup\_crtcs//寻找一个最近的crtc并使能connector状态

drm\_fb\_helper\_single\_fb\_probe//此函数极为重要根据当前的bpp值，设置宽度，高//度以及注册framebuffer结构体fb

(\*fb\_helper->funcs->fb\_probe)即drm\_fbdev\_cma\_create调用

drm\_gem\_cma\_create//通过GEM创建cma内存

framebuffer\_alloc(0, dev->dev)//分配framebuffer

drm\_framebuffer\_init//初始化DRM的framebuffer结构

drm\_fb\_helper\_fill\_fix//初始化fb\_info的固定参数

drm\_fb\_helper\_fill\_var//设置rgb的占的像素位数以及像素偏移值

register\_framebuffer(info)//注册frambuffer作为图像缓存

以上一大堆的代码最终完成了drm\_dev\_register设备的注册，对于DRM的fbdev结构体相关内存的初始化占了很大一部分此部分错综复杂，会给调试增加较大难度。