Q/LS

龙芯中科技术有限公司企业标准

Q/LS 0013-2014

**龙芯LS2K1000集显设备树使用规范**

发布 实施

龙芯中科技术有限公司 批准

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档更新记录 | | 文档编号 | |  |
| 文档名 | | 龙芯2K1000集显设备树使用规范 |
| 版本号 | | V0.1 |
| 创建人 | | 系统研发部图形组 |
| 创建日期 | | 2022-06-16 |
| 更新历史 | | | | |
| 序号 | 更新日期 | | 版本号 | 更新内容 |
| 1 | 2022.06.28 | | V0.1 | 发布文档初始版本V0.1版。 |
| 2 |  | |  |  |

目录

1. 术语与定义 1

2. 显示方案 2

2.1 采用透明视频转换芯片配置方案 2

2.2 采用非透明视频转换芯片的显示配置方案 2

2.3 液晶屏显示配置方案 3

2.4 其它视频转换芯片显示配置方案 4

2.5 LS2K1000多屏显示配置方案总结 5

3. LS2K1000 SoC集显设备树传参规范概述 7

3.1 LS2K1000集显设备树的基本写法 9

3.2 从设备树给集显驱动传递EDID的方法 11

3.3 从设备树给集显驱动传递显示时序的方法 12

3.3.1传递给驱动的各个显示时序参数的意义 14

3.3.2液晶屏幕的每秒钟刷新频率计算方法 15

3.4 禁止使用某一路显示管线的设备树写法 16

3.5 某一路显示从一开机就克隆另一路显示的设备树写法 17

3.6 背光支持 18

3.6.1 背光亮度计算公式 21

3.6.2 通过sysfs接口调节背光 22

3.6.3 通过电源首选项对话框调节背光 22

4. 基于OF GRAPH的新设备树传参规范 23

4.1 新设备树传参数规范下集显设备树的基本书写方法 24

4.2 LT8618 DVO转HDMI编码器设备树书写示例 26

4.3 Sii9022 DVO转HDMI编码器设备树书写示例 28

4.4 透明类型DVO转VGA编码器在新设备树规范下的书写示例 29

4.5 液晶屏显示设备在新设备树规范下的书写示例 31

5. LS2K1000的图形驱动软件构架简介 33

6. 参考文献 35

前 言

本规范是给使用龙芯2K系列SoC的开发者制定的, 旨在帮助他们制做出满足其显示相关需求的产品。龙芯2K系列嵌入式SoC主要包括LS2K1000 MIPS架构和LS2K1000LA架构的芯片等。

本规范介绍了几种龙芯LS2K1000与外部视频转换芯片之间的连接方案，本规范还介绍了如何通过设备树来描述主板上显示接口、显示转换芯片和LS2K1000中的DVO输出接口、I2C控制器、ＰＷＭ控制器以及专用GPIO之间的拓扑结构和连接关系;使得显示控制器设备驱动能和显示编码转换芯片驱动能有序地结合在一起，工作起来。

设备树也被用来给内核、设备驱动传递参数, 从而达到修改程序工作行为和模式的目的。支持什么样的参数、在哪里写参数、支持那些巧妙用法需要一篇文档来解释说明。希望本文档能够帮助龙芯LS2K系列SoC的使用者解决开发和配适过程中遇到的各类显示相关问题。

本规范的起草单位：龙芯中科系统研发部

本规范的主要起草人：隋景峰

本规范审核人：李轶

龙芯2K1000集显设备树使用规范

# 术语与定义

本规范所用术语定义如下：

1. **DT**：设备树（Device Tree）的简称。
2. **BIOS**：Basic Input Output System的简写, 也被称为固件。是一组固化到主板上ROM芯片上的程序, 它保存着计算机基本输入输出设置程序、系统配置信息、开机上点后自检程序和内存训练程序等。对于龙芯MIPS架构和LoongArch构架的嵌入式SoC来说, 固件通常指的是LOONGSON公司维护的PMON。
3. **Encoder**/**Transmitter**/**Transcoder**/**Display bridge** 都指显示编码器（视频转换芯片）, 用于将 DVO（RGB888）输出信号转换为其它显示信号, 如LVDS、HDMI、DVI、VGA和eDP等。不同的厂商和芯片手册用不同的名字, 本文中出现的**显示编码器**和**视频转换芯片**等术语都是这里的编码器的意思。
4. **Connector**: 显示连接器, 也称显示接口。常见的显示器为VGA、DVI、HDMI和DP等显示接口, 而常用液晶屏多为eDP, LVDS或DPI等显示接口。
5. **EDID**: Extended display identification data, 扩展显示器识别数据。是VESA在制定的有关显示器识别数据的标准。 EDID通常存储在显示器的EEPROM中, 从EDID1.0到EDID1.3, EDID的容量都是128字节。
6. **DDC**: Display Data Channel, 显示数据通道。 显示控制器驱动通过DDC探测显示器是否存在以及从显示器中获得EDID数据。
7. **DC**: Display Controller, 显示控制器。
8. **GPU**: 图形处理单元, 负责图形的渲染。
9. **显示管线：** 一条显示管线包括一个CRTC、一个Encoder、一个Connector、一个i2c设备等必备成分（i2c是可选的, 必要时可以通过在设备树中提供显示时序参数的方式代替）。下文中出现的显示输出也是显示管线的简化说法。

# 显示方案

龙芯LS2K系列SoC显示相关的设备驱动为了满足各种嵌入式应用的需求, 支持多种类型的显示编码器和硬件连接方案。其中显示编码器分为**透明类型的显示编码器**和**非透明类编码器**, **透明类的显示编码器**, **指的是不需要程序配置的视频转换芯片**。比如ADV7123、ADV7125、GM7123C等DVO转VGA视频转换芯片就是透明类型的显示编码器。**非透明类编码器是需要写驱动程序配置的编码器芯片**，比如lt8618、sii9022、ch7033b等芯片。还有一类芯片既能以透明方式工作又能以非透明的方式工作，如TFP 410、sii9022等。对于这类芯片应该根据显示编码器芯片手册推荐的电路图选择最佳的显示配置方案。

本章2.1、2.2、2.3和2.4小节将介绍4种基础的多屏显示配置方案，实际的板卡基本上就是这四种显示方案的变种和组合。2.5小节列举出由龙芯中科设计制造的几款板卡的显示配置方案的框图, 作为范例介绍给读者参考。

## 2.1 采用透明视频转换芯片配置方案

如果选用透明输出类型的显示编码器转换芯片, 那么就不需要在内核态写驱动进行配置。因为透明输出类型的显示编码器转换芯片只具备转换视频信号的功能不具备I2C通信功能。电路板布线时, 某一路显示管线的**显示连接器的I2C引脚直接与这路显示管线所用的I2C设备相连接**, 两路显示输出均采用透明类型的编码器的显示配置方案如**图2-1**所示。

**图 2-1：由透明输出类型的显示编码器所组成的显示连接方案**

DDC

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

Encoder0

Encoder1

Connector0

显示设备

I2C-X

I2C-Y

显示设备

Connector1`

DVO0

DVO1

虽然这里选用的是透明视频转换芯片, 但是不代表这种类型的视频转换芯片就完全不受控制。比如ADV7125 编码器除了视频转换功能外, 还具有功耗控制和硬件复位等功能管脚。如果板卡对待机功耗和硬件复位功能有要求, 可以把它们连接到2K系列SoC的GPIO上。

TI的TFP410（DVO转HDMI或DVI）当成透明式编码器使用时, 可以归结到此类显示方案之中。

## 2.2 采用非透明视频转换芯片的显示配置方案

如果选用的是非透明类型的显示编码转换芯片, 那么需要在内核态下写驱动对编码器芯片进行配置, 整个系统才能正常显示和工作。两路输出均选用非透明的编码器的显示方案如**图2-2**所示。

**图 2-2: 非透明输出类型的显示编码器的显示连接方案**

DDC

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

Encoder0

Encoder0

Connector0

显示设备

I2C-X

I2C-Y

显示设备

Connector0

DVO0

DVO1

具备DDC功能的非透明类型的显示编码转换芯片具备一个I2C Master通信接口和一个I2C Slave通信接口。**显示编码转换芯片的这个I2C Master通信接口应该连接到显示连接器的I2C引脚上**, **而编码器芯片的I2C Slave接口应该连接到该条显示管线所用的I2C设备**（这个I2C设备指的是LS2K1000的 SoC中的I2C， 可以是硬件I2C也可以是GPIO模拟I2C）。这时, **该条显示管线所用的I2C设备无需再与显示连接器的I2C管脚连接**。简单的说, 就是以显示编码器为中间媒介将EDID数据从显示器读回来, 然后显示编码器再把这个EDID送回给LSDC设备驱动, 而不是LSDC设备驱动直接操作I2C从显示器来读取。这样做的好处是能够简化主板电路走线, 使得主板做工精美, 显示编码器的设备驱动程序也能写的十分规范。

非透明的显示编码器除了具备DDC功能外, 还具备热插拔中断转发的功能。这时, 需要把显示连接器的热插拔中断功能管脚连接到显示编码器的热插拔中断**输入**引脚上, 再将显示编码器的热插拔中断**输出**引脚连接到LS2K1000 SoC的GPIO上。

## 2.3 液晶屏显示配置方案

龙芯派的第一路显示管线用的就是DPI接口的液晶屏幕, 这种接口通过FPC排线直接接到LS2K1000的DVO输出接口上。无需其它厂家的视频编码转换芯片, 比较节约成本, 适合低成本嵌入式类应用。系统原理框图如**图2-3**所示, 这里只是用了LS2K1000的第一路显示管线作为例子, 若第二路显示管线也需要作为显示使用, 只需按照本文档中的所述配置方案连接布线即可。

**图 2-3: 液晶屏幕显示配置方案**

GPIO（液晶电源控制、背光使能控制）

专用液晶屏接口，通过FPC排线连接, 包括由PWM控制的背光、液晶电源、RGB888视频输出, VSYNC, HSYNC等。

液晶屏

PWM（背光亮度控制）

**DC IN LS2K1000**

CRTC0

DVO0

DVO1

CRTC1

需要注意的是, 这种DPI接口的液晶屏幕不具备I2C接口, 无法通过I2C读取其显示屏幕的参数。因此需要通过设备树传递EDID或显示时序。另外和显示器不同的是, 液晶屏幕往往需要通过PWM控制器来控制液晶屏幕的背光亮度; 液晶屏幕一般还会提供控制液晶电源的开与关的GPIO接口和背光控制的使能与否的GPIO接口。这些控制管脚需要连接到LS2K1000 SoC的PWM控制器或GPIO上。具体连接哪个PWM或那个GPIO本文档不做限制, 只是要求所用硬件资源不要和其它子系统功能模块重叠, 优先用那些没有管脚复用功能的硬件资源, 以免限制整个系统的灵活性。

透明式DVO转LVDS视频编码器（比如 TI的SN65LVDS93A）也可以归结为此类显示方案中。

## 2.4 其它视频转换芯片显示配置方案

还有一类显示编码器芯片具有通过I2C接口（它相对于LS2K1000来说是个I2C Slave）配置该芯片的功能, 但是却**不具备**从显示器获取EDID的功能（没有I2C master功能管脚）。TI的TFP410就是这类显示芯片中的一个。这种情况下, **显示管线中I2C总线即要连接到视频转换芯片上又要连接到显示接口的管脚上**。连接到视频转换芯片是为了配置该芯片, 连接到显示接口上是为了从显示器获取EDID。**图2-4**给出了示意图。

**图 2-4：选用非透明输出类型的显示编码器的显示连接方案**

DDC

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

Encoder0

Encoder0

Connector0

液晶屏

I2C-X

I2C-Y

显示器

Connector0

DVO0

DVO1

采用这种显示方案的另外一种情况是把非透明视频转换芯片当成透明类型的转换芯片来用。比如sii9022芯片就支持这种工作模式。这种情况下, 仍然需要在内核中写驱动配置sii9022, LSDC驱动通过i2c设备从显示连接器来读取EDID, sii9022芯片自带的ddc功能没有用起来。龙芯派（龙芯LS2K1000派电路板）的第二路显示默认使用的就是上述方案。**如果有更好的设计方案，这种设计一般不推荐**。当然为了维护后向兼容性以及历史遗留产品, 我们当前的设备驱动和设备树还是支持这种显示方案的。

## 2.5 LS2K1000多屏显示配置方案总结

实际项目中的板卡多为本章节中2.1、2.2、2.3、2.4小节所介绍的4种显示配置方案的组合和变种。下面介绍几款有龙芯中科设计的板卡作为参考。

龙芯派（LS2K1000\_PAI\_UDB\_V1.5）的显示配置方案如**图 2-5**所示, 这款主板的第一路显示管线配适的飞凌嵌入式出品的1024x600（或800x480）的液晶屏幕。采用LS2K1000 SoC中的PWM0来做的背光控制, 背光电源使能和背光调节使能引脚用的都是用GPIO 3来控制的, 新设计的板卡可以采用两个不同的GPIO进行分别控制。这款主板的第二路显示管线选用的是sii9022 DVO转HDMI编码器, 使用LS2K1000中的硬件I2C-1来配置sii9022和从显示器中读取EDID。这里设计的有点不够规范, 因为sii9022编码器芯片本身具备DDC功能, 能够从显示器读取EDID并转发给LS2K1000, 建议**新设计的主板能够采用本章2.2章节所描述的那样，采用非透明输出的方式使用sii9022编码器芯片**。

**图 2-5: 龙芯派显示配置方案**

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

sii9022

专用液晶屏接口，通过FPC排线连接;包含 RGB888视频输出, VSYNC, HSYNC等。

飞凌1024x600或800x480

液晶屏

硬件i2c-1

显示器

HDMI-A

DVO0

DVO1

PWM0（用于控制背光亮度）

GPIO3（用于控制背光电源）

龙芯教育派的显示配置方案如**图2-6所示**, 龙芯教育派是龙芯派的简化版, 它只使用第一路显示输出。它选取了sii9022作为其第一路显示管线的编码器芯片, 对编码器的配置方法与上文中介绍的龙芯派的第二路显示管线相同。**新设计的产品推荐采用非透明输出的方式使用sii9022编码器芯片。**

LS2K1000 测试机（LS2K1000\_PC\_EVB\_V1.2）电路板的实际显示配置方法如**图2-7**所示, 该版卡两路输出均采用透明类型的编码器, 是本章中2.1小节所介绍的显示连接方案的一个实例。 值得注意的是, 由于LS2K1000中只有两个硬件I2C, I2C资源不够用, 所以这款主板用的是GPIO模拟I2C来充当DDC总线的功能。

**图 2-6: LS2K1000 教育派显示输出配置方案**

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

sii9022

HDMI-A

有HDMI接口的显示器

硬件i2c-0

DVO0

DVO1

DVO1显示管脚没有引出，不使用

**图2-7：LS2K1000测试（开发）机器显示输出配置方案**

DDC

DDC

**DC IN LS2K1000**

CRTC0

CRTC1

ADV7125

TFP410

VGA

有VGA接口的显示器

GPIO-I2C-0

GPIO-I2C-1

有DVI接口的显示器

DVI-D

DVO0

DVO1

# LS2K1000 SoC集显设备树传参规范概述

龙芯LS2K1000 SoC中集成了显示控制器（DC）和图形处理器（GPU）, 设备树支持了两种传参数的方法, 一种是**支持上游内核社区常用的设备树规范**, 下文中简称为**新设备树规范**。 另一种是为了**支持各种基于龙芯2K系列SoC定制板卡和项目需求**以及在**维护后向兼容性制约**所形成的特有的传参规范, 下文中简称为**老设备树规范**。

LSDC设备驱动在诸多透明编码器主板上编写和被应用, 后来为了支持非透明类型的编码器, 采用基于 DRM bridge 框架的方式来编写非透明类型的编码器的设备驱动。采用DRM Bridge框架编写的编码器设备驱动能够去耦合, 每个编码器的设备驱动都是一个独立的设备驱动模块。基于DRM Bridge框架编写的编码器设备驱动能被许多不同种类的显示控制器驱动所使用, 实现了代码共享。LSDC设备驱动程序根据设备树在运行时绑定编码器驱动模块, 从而达到协同工作的目的。

**新设备树规范既能支持透明类型的编码器又能支持非透明类型的编码器**。但并不是新设备树规范就一定比老设备树规范好。老设备树传参数规范支持**从设备树传 EDID**、**从设备树传显示时序**、**龙芯显示控制器独有的硬件克隆工作模式**等功能, 但是目前老设备树传参数规范只适用于透明编码器。实际产品可以根据需求灵活选用。

**目前LSDC设备驱动能同时支持前文所述中的两种设备树书写规范**。区分也十分简单: 在设备树的显示控制器设备节点中, **如果存在output-ports及其相关属性**, **就表示用户用的是老设备树传参规范**。 在设备树的显示控制器设备节点中, **如果dc@0x400c0000设备节点中写了ports 属性**, **就表示用的是新设备树传参数规范**。

**图 3-1: 新设备树规范和老设备树规范的书写范式对比**

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## **output-ports** **=** <&**dvo0** &**dvo1**>;

## **dvo0**: dvo@0 {

## reg = <0>; 　/\* 0 for display pipe 0 \*/

## };

## **dvo1**: dvo@1 {

## reg = <1>; 　/\* 1 for display pipe 1 \*/

## };

## };

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## **ports {**

## **port@0 {**

## reg = <0>; /\* 0 for display pipe 0\*/

## **};**

## **port@1 {**

## reg = <1>; /\* 0 for display pipe 0\*/

## **};**

## **};**

## };

如上边的文本框所示的设备树代码所示, dc@0x400c0000 代表显示控制器的设备树节点, 左半边文本框中 dc@0x400c0000设备节点中有**output-ports** **=** <&**dvo0** &**dvo1**>代码且没有ports属性, 这种情况使用的就是老设备树规范; 右半边文本框dc@0x400c0000设备节点中存在ports属性, 代表使用的是新设备树规范。 这里为了简洁删除许多必要的设备树属性, 只是为了说明问题, 较为完整的设备树请参看下文中的示例。

总之, **LSDC设备驱动程序在初始化时检测设备树中是否有ports属性, 如果显示控制器的设备节点中有ports属性则优先用新设备树规范程序解析设备树； 如果显示控制器的设备节点中没有ports属性则检测output-ports属性, 如果存在就用老设备树规范**。

本章余下篇幅主要先介绍我们自己的老设备树传参数规范, 本文中简称之为老**设备树规范**。第四章介绍一种上游内核支持的、也是众多芯片系统采用的一种设备树规范, 也就是前文所述的**新设备树规范**。LS2K1000 SoC的开发者可以根据需求灵活选择一个来使用。

需要注意的是： 受制于篇幅的原因, 本章节中多是以片段的形式给出设备树示例, 在实际的使用场景下需要将我们给出的示例同LS2K1000 SoC和主板上的其它设备的设备节点放到一起, 描述完整使用。请读者结合龙芯产品内核中arch/loongarch/boot/dts/loongson/和arch/mips/boot/dts/loongson/目录中给出的设备树文件参看本文档。

## **3.1 LS2K1000**集显设备树的基本写法

LS2K1000 集显设备树的基本写法如**图3-2**中设备树代码所示, 其中左边是显示控制器相关设备树, 右边是LS2K1000中i2c设备的设备树。设备树编写者可以根据电路板的布线情况去更改哪一路I2C设备被哪一路显示管线所用。从我们这里给出的样例可以看出, 第一路显示用的是 **i2c0**, 第二路显示管线用的**i2c1**。

**图3-2: LS2K1000集显设备树示例**

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## reg = <0 0x400c0000 0 0x00010000>;

## interrupt-parent = <&icu>;

## interrupts = <28>;

## **output-ports** **=** <&**dvo0** &**dvo1**>;

## #address-cells = <1>;

## #size-cells = <0>;

## **dvo0: dvo@0 {**

## reg = <0>;

## ddc-i2c-bus = <&**i2c0**>;

## connector = "vga-connector";

## status = "okay";

## **};**

## **dvo1: dvo@1 {**

## reg = <1>;

## connector = "dvi-connector";

## ddc-i2c-bus = <&**i2c1**>;

## status = "okay";

## **};**

## };

## **i2c0: i2c-gpio@0** {

## compatible = "i2c-gpio";

## gpios = <&pioA 1 0 &pioA 0 0>;

## i2c-gpio, delay-us = <5>; /\* ~100 kHz \*/

## };

## **i2c1**: **i2c-gpio@1** {

## compatible = "i2c-gpio";

## gpios = <&pioA 32 0 &pioA 33 0 >;

## i2c-gpio, delay-us = <5>; /\* ~100 kHz \*/

## };

## i2c2: i2c@1fe21000 {

## compatible = "loongson, ls-i2c";

## reg = <0 0x1fe21000 0 0x8>;

## interrupt-parent = <&icu>;

## interrupts = <22>;

## };

## i2c3: i2c@1fe21800 {

## compatible = "loongson, ls-i2c";

## reg = <0 0x1fe21800 0 0x8>;

## interrupt-parent = <&icu>;

## interrupts = <23>;

## };

**说明:**

**LS2K1000 中的显示控制器有两路dvo显示输出**, 在设备树中用**output-ports** **=** <&**dvo0** &**dvo1**>列举出来。其中&dvo0表示对dvo@0设备节点的引用, &dvo1表示对**dvo@1**设备节点的引用。 LSDC设备驱动程序是通过ouput-ports属性的值找到dvo@0和dvo@1设备节点的, **dvo@0**和**dvo@1**中能包括子设备节点。 如果有**3**路显示输出则可以写为**output-ports** **=** <&**dvo0** &**dvo1** &**dvo2**>, 但是目前龙芯所有的显示控制器都只有两路显示输出。

**dvo@0 设备节点表示第一路显示管线的相关属性, 表征LS2K1000 的dvo0输出接口及其下游配套的显示编码器、显示连接器等的一切属性的集合, 用标号dvo0表示。**

**dvo@1 设备节点表示第二路显示管线的相关属性, 表征LS2K1000 的dvo1输出接口及其下游的一切属性的集合, 用标号dvo1表示。**

dvo0和dvo1中status属性的值为 "**ok**" 、"**okay**" 或**默认不写status属性**都表示这路显示为可用的状态。 如果dvo0和dvo1中两路中有一路不可用, 或者暂时不打算使用这路显示管线, 则可以把status的值改为"**disabled**"。 注意： 只要是设备树中写了status属性**并且**它的值不是"ok" 或 "okay", 那么这条显示管线就会被驱动程序解析为不可用状态。支持status属性的好处是将一块电路板的设备树用到另一块电路板上时, 如果某个设备没有被电路板上引出不可用, 可以直接将status的属性值由"**ok**"改为"**disabled**"。 这样做可以最小化对设备树的改动, 提高配适工作的效率。另外一个好处是可以节约资源, **被disabled了的这路显示管线所占用的硬件资源可以给其它驱动或设备使用**。

**ddc-i2c-bus 属性用来描述分配给某一路显示管线用的i2c设备**, 格式为**ddc-i2c-bus** = <&**i2c0**>;这里表示的是该路显示管线引用的是i2c0, 如果引用的是i2c1则写成**ddc-i2c-bus** = <&**i2c1**>。老设备树中没有ddc-i2c-bus属性, 这种情况下程序为了保持向前兼容, 默认DVO0显示管线用I2C0, DVO1显示管线用I2C1。本规范文档发布之后, 推荐用户在设备树中通过ddc-i2c-bus设备节点提供该路显示管线所用的i2c设备。 **ddc-i2c-bus 属性给显示方案增加了灵活性, 因为它允许用户根据电路板的布线情况指定实际分配给某一路显示用的i2c设备**。

**如果电路板上采用是LVDS、eDP或DPI等不支持DDC的液晶屏, 那ddc-i2c-bus 属性就不需要写, 此时设备树中需要提供edid或显示时序。**LSDC用**ddc-i2c-bus** 属性所指定的i2c总线用来探测显示器的插拔状态和从显示器读取EDID信息。一旦某一路显示管线提供了EDID或显示时序数据, 那么这路显示管线就总被视为连接状态, LSDC设备驱动程序就不会用这路i2c去探测显示器是否存在了, 而是按照给定的显示时序（或显示模式）强制输出。

**connector** 属性用来指定连接器的类型, 根据电路板上实际情况来填写。可选的值为: "**vga-connector**" , "**dvi-connector**", "**hdmi-connector**", "**dpi-connector**", "**lvds-connector**"。其中 "dpi-connector" 就是常用的DPI接口(就是RGB888直接输出不加任何转换芯片的输出接口)的屏幕, 与LS2K1000龙芯派配套的飞凌电容屏就可以认为是 "dpi-connector"。LS2K1000 测试机（LS2K1000\_PC\_EVB\_V1.2）主板的第一路是vga接口, 这里就填写 "vga-connector" 。第二路显示为dvi, 那么connecter属性值就填写为 "dvi-connector"。

**reg**属性本意是用来标识设备树中显示输出管线的硬件序号, 从0开始, 属于冗余设计。**dvo@0设备节点中就写0, dvo@1设备节点中就写1**。

我们的老设备树规范中除了支持这四个基本属性还支持EDID属性, display-timings 属性, clone属性, 将在本章后面小节中一一介绍。

## **3.2 从设备树给集显驱动传递EDID的方法**

**图3-3**给出了一个通过设备树传递EDID数据的例子, **图3-3**中左边的文本框给出了简要说明。

**图3-3：设备树传递EDID数据样例**

假如电路板的第二路显示管线（dvo1）连接了一个不支持ddc的液晶屏幕, 液晶屏幕的分辨率为1920x1080, 那么设备树可以写成如右边文本框中的那样。

**如果所用液晶屏的分辨率不是1920x1080, 用户需要自己根据液晶屏幕的分辨率生成相应的EDID数据, 放在edid=[]；这行代码的方括号中。**目前支持的edid长度为128字节, 每个字节与每个字节用之间用空格或Tab键或回车键分隔开就行。

如果是第一路显示管线dvo0连接了一个不支持ddc的液晶屏幕, 那就在dvo@0节点中填写edid属性。如果两路都接了不支持ddc的液晶屏, 那就两路都需要提供edid。

edid blob和显示时序可以同时提供。

由于历史原因, edid比显示时序具有更高的优先级, 设备树中提供了edid, 就优先用edid。没有edid再检测设备树中有没有提供显示时序, 如果时序也没有提供我们就去用i2c去读取显示器的edid。

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## reg = <0 0x400c0000 0 0x00010000>;

## interrupt-parent = <&icu>;

## interrupts = <28>;

## **output-ports** = <&**dvo0** &**dvo1**>;

## #address-cells = <1>;

## #size-cells = <0>;

## **dvo0**: dvo@0 {

## reg = <0>; /\* 0 for display pipe 0 \*/

## ddc-i2c-bus = <&i2c0>;

## connector = "vga-connector";

## status = "ok";

## };

## **dvo1**: dvo@1 {

## reg = <1>; /\* 1 for display pipe 1 \*/

## connector = "hdmi-connector";

## /\* 1920x1080@60Hz \*/

## **edid = [** 00 ff ff ff ff ff ff 00 1e 6d 54 5b 0b cc 04 00

## 02 1c 01 03 6c 30 1b 78 ea 31 35 a5 55 4e a1 26

## 0c 50 54 a5 4b 00 71 4f 81 80 95 00 b3 00 a9 c0

## 81 00 81 c0 90 40 02 3a 80 18 71 38 2d 40 58 2c

## 45 00 e0 0e 11 00 00 1e 00 00 00 fd 00 38 4b 1e

## 53 0f 00 0a 20 20 20 20 20 20 00 00 00 fc 00 4c

## 47 20 46 55 4c 4c 20 48 44 0a 20 20 00 00 00 ff

## 00 38 30 32 4e 54 43 5a 39 38 33 37 39 0a 00 35

## **];**

## status = "ok";

## };

## };

需要注意的是**EDID数据是人类不可读的**（或者说不方便识别理解的）, linux内核社区反对在设备书中写EDID属性。而且嵌入式系统用的液晶屏多是支持任意刷新率的非标准分辨率的。比如1024x600@70Hz, 800x480@40Hz等分辨率。edid blob可能不能对所用液晶屏的时序提供准确的描述。而且液晶屏幕厂商大多提供的是时序数据, 在这种情况下, 还可以在设备树中提供显示时序。

## **3.3 从设备树给集显驱动传递显示时序的方法**

嵌入式设备常用的液晶屏幕通常没有DDC支持, 从设备传递所用显示屏的显示时序信息是常用的做法。**图3-4**所示的文本框给出了一个通过设备树传递显示时序的样例。绝大多数液晶屏幕只支持一种显示分辨率, 但是如果您所用的屏幕支持多个分辨率，那么我们的老设备树传参规范也支持同时给定多组显示时序, 具体细节见**图3-5**。

dc@0x400c0000 {

compatible = "loongson, display-subsystem";

output-ports = <&dvo0 &dvo1>;

dvo0: dvo@0 {

reg = <0>;

status = "ok";

connector = "dpi-connector";

**display-timings {**

**native-mode = <&**mode\_1024x600\_60**>;**

mode\_1024x600\_60**: display-timing@0 {**

**clock-frequency = <51200000>;**

**hactive = <1024>;**

**hsync-len = <4>;**

**hfront-porch = <160>;**

**hback-porch = <156>;**

**vactive = <600>;**

**vfront-porch = <11>;**

**vback-porch = <23>;**

**vsync-len = <1>;**

**};**

**};**

};

dvo1: **dvo@1** {

reg = <1>;

status = "ok";

connector = "hdmi-connector";

type = "hdmi-a";

ddc-i2c-bus = <&i2c1>;

};

};

**紫色的部分为一组显示时序**, 右边文本框中提供的是飞凌嵌入式的一款液晶屏幕（1024x60@60Hz）。如果用户所用液晶屏不是这款, 那么用户需要根据自己的液晶屏幕参数填写正确的显示时序。

display-timings设备节点中的native-mode属性用于指定用户用的液晶屏所支持的最佳分辨率。开机进入桌面默认就会使用这个分辨率。

如果是第二路显示管线连接了这种液晶屏幕, 那就在**dvo@1**节点中填写display-timings设备节点及其包含的native-mode属性和显示时序参数子节点。 **如果两路显示都接了液晶屏**, **那就两路显示输出属性设备节点中都需要提供相应的显示时序**。

**注意：** 提供了显示时序的显示管线会被驱动程序认为总是处于连接的状态。LSDC设备驱动和用户态驱动以及X Server等软件会往这块屏幕上更新画面, 即使用户不接这块屏幕。

**图 3-4： 设备树传递显示时序的示例**

属于同一组的显示时序放于display-timing@0设备节点中, 而display-timing@0又被包含在display-timings {}设备节点中。 多数显示液晶显示屏只支持一种显示分辨率, 正常情况下用户只需提供一组时序参数就行。mode\_1024x600\_60 为display-timing@0设备节点的标号, 它代表display-timing@0 设备节点, 起到被其它属性引用目的。　display-timing@0中的所有属性都是必须的, 用户只需改变这些属性的值（就是<>中的数字）就可以了。具体的每个参数典型值, 需要从液晶屏厂商提供的数据手册中获取。

**图3-5: display-timings设备节点中包含多组显示时序参数的设备树样例**

dc@0x400c0000 {

compatible = "loongson, display-subsystem";

output-ports = <&dvo0 &dvo1>;

dvo0: dvo@0 {

reg = <0>;

status = "ok";

**display-timings {**

**native-mode = <&mode\_800x480\_60>;**

mode\_1024x600\_60: **display-timing@0** {

clock-frequency = <51200000>;

hactive = <1024>;

hsync-len = <4>;

hfront-porch = <160>;

hback-porch = <156>;

vactive = <600>;

vfront-porch = <11>;

vback-porch = <23>;

vsync-len = <1>;

};

**mode\_800x480\_60: display-timing@1 {**

**clock-frequency = <30066000>;**

**hactive = <800>;**

**hfront-porch = <50>;**

**hback-porch = <70>;**

**hsync-len = <50>;**

**vactive = <480>;**

**vback-porch = <0>;**

**vfront-porch = <0>;**

**vsync-len = <50>;**

**};**

**};**

};

};

## 如果显示屏幕支持多个分辨率,　那么在**display-timings {}**设备节点中依次罗列**display-timing@1，display-timing@2，...，** **display-timing@n**设备节点即可。只要包含每组显示时序信息的设备节点名字互不相同。其中每个**display-timing@n（ｎ＝1,2,...,n）**设备节点中都包含一组显示时序参数。

如果一个电路板可以配适多种不同的屏幕, 那么一个设备树文件的**display-timings {}**设备节点中也可以包含多组不同的显示时序参数。更换显示屏时, 只须改动native-mode属性所指向的显示时序即可。起到工程意义上的便捷、复用的目的。

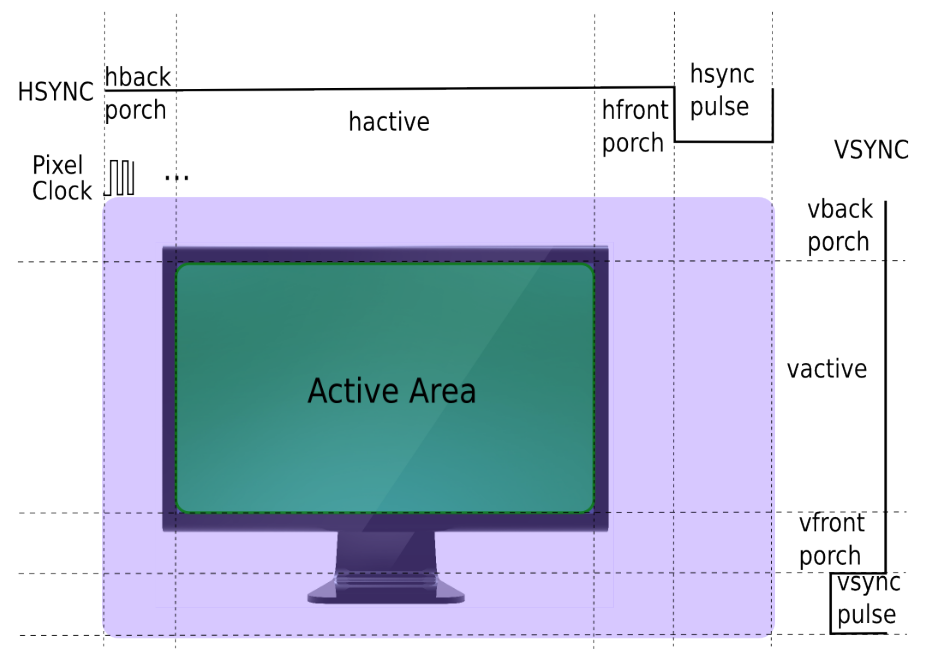
如果用户实际的板卡是第二路显示输出连接的液晶屏幕，请把**display-timings {}**设备节点及其包含的子设备节点写到dvo@1设备节点中。这里为了节约篇幅, 略去了对dvo@1设备节点的书写。

当**display-timings {}**设备节点中包含多组正确的显示时序信息时, 打开显示器首选项对话框会的分辨率菜单会出现多组时序参数。如**图3-6**所示, 感兴趣的读者请自行实验观看。



**图 3-6: 观察设备树中提供的显示时序是否正常的方法**

#### **3.3.1传递给驱动的各个显示时序参数的意义**

**图 3-7: 液晶屏幕显示时序示意图**

|  |  |
| --- | --- |
| 参数 | 意义 |
| **clock-frequency** | **单位为Hz**, 视频硬件在显示器上绘制像素的速率, 一个时钟周期完成一个像素点的显示。这个参数表征的是视频硬件在一秒钟所绘制的显示个数。 |
| **hsync-len** | 水平同步信号的宽度, 以像素个数为单位, 用来**告诉显示硬件换行的**。hsync-len就是**图 3-7**中的hsync pulse。 |
| **hactive** | 水平像素数目, 单位为个。表示显示屏一**行**中可显示像素个数, 就是显示屏幕的水平分辨率。 |
| **hfront-porch** | 水平同步信号的前肩, 表示从一行的有效数据结束到下一个水平信号开始之间的像素时钟的个数。 |
| **hback-porch** | 水平同步信号的后肩, 表示从水平同步信号开始到一行的有效数据开始之前的像素时钟的个数。 |
| **vactive** | 垂直像素数目, 单位为个。表示显示屏一**列**中可显示像素个数, 就是显示屏幕的垂直分辨率。 |
| **vsync-len** | 垂直同步信号的宽度, 以行为单位, 用来**告诉显示硬件这一帧画面显示完毕的, 开始显示下一帧画面**。vsync-len就是**图 3-7**中的vsync pulse。 |
| **vfront-porch** | 垂直同步信号的前肩, 表示一帧图像结束后, 垂直同步信号**以前**的无效行数 |
| **vback-porch** | 垂直同步信号的后肩, 表示一帧图像开始时, 垂直同步信号**以后**的无效行数 |

#### **3.3.2液晶屏幕的每秒钟刷新频率计算方法**

这一小节主要是说明display-timing@0设备树节点中clock-frequency参数和其它参数之间的关系。以上边1024x600@60Hz的显示时序参数举例来说：

**mode\_1024x600\_60: display-timing@0 {**

**clock-frequency = <51200000>;**

**hactive = <1024>;**

**hsync-len = <4>;**

**hfront-porch = <160>;**

**hback-porch = <156>;**

**vactive = <600>;**

**vsync-len = <1>;**

**vfront-porch = <11>;**

**vback-porch = <23>;**

**};**

那么显示屏的刷新频率为：

**refresh\_rate = clock-frequency /（h\_total × v\_total）;**

其中：

**h\_total = （hactive + hsync-len + hfront-porch + hback-porch）;**

**v\_total = （vactive + vsync-len + vfront-porch + vback-porch）;**

所以, 若想提高液晶刷新频率, 可以将clock-frequency里边儿的数值写的高一些。

## **3.4 禁止使用某一路显示管线的设备树写法**

LS2K1000的DVO0 显示输出接口与 LIO 以及 UART 有复用关系, DVO1 显示输出接口与CAMERA功能接口有复用关系。 LS2K1000 作为嵌入式类型的SoC在某些应用场合可能只需要单路显示输出, 这时只需要将不使用的那路显示输出的status属性的值改为disabled就行, 无需大量修改其余属性。如**图3-8**设备树样例所示的那样。

**图3-8: 禁止使用某一路显示管线的设备树书写样例**

假如电路板只引出了第一路dvo显示, 第二路显示接口没有被引出（或者说不打算被使用, 或者用于做其它用途）, 则可以将第二路显示管线中的status属性改为**disabled**。

一旦某一条显示管线中的status属性被设置为“disabled”值, 那么这条显示管线中的其他属性即使写了也不会被使用, 因为程序检测到了disabled值后就不会再解析这条显示管线中的其余属性。驱动程序不会再为这条显示管线分配资源。这里为了清晰和节约篇幅就删去了其余属性。

LS2K1000教育派开发板第一路显示管线外接的显示转换芯片是sii9022, 第二路显示管线没有引出, 因此可以在DVO@1设备树节点中填写**status = "disabled"**设备树代码, 以反映硬件的真实工作状态。

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## reg = <0 0x400c0000 0 0x00010000>;

## interrupt-parent = <&icu>;

## interrupts = <28>;

## **output-ports** = <&**dvo0** &**dvo1**>;

## #address-cells = <1>;

## #size-cells = <0>;

## **dvo0**: dvo@0 {

## reg = <0>; /\* 0 for display pipe 0 \*/

## ddc-i2c-bus = <&**i2c1**>;

## connector = "hdmi-connector";

## **status = "ok";**

## };

## **dvo1**: dvo@1 {

## reg = <1>; /\* 1 for display pipe 1 \*/ **status = "disabled";**

## };

## };

## **3.5 某一路显示从一开机就克隆另一路显示的设备树写法**

龙芯LS2K1000 SoC中的显示控制器支持独特的硬件克隆模式, 当开启时这种显示模式时, 某一路显示管线会作为另一路显示管线的附庸而存在。这种工作模式完全是由硬件做的克隆工作, 从LSDC设备驱动角度来看, 此时系统中只有一路**可控**的显示管线。这种工作模式可以为低成本镜像屏应用提供解决方案。**图3-9**给出了一个通过设备树来配置显示控制器工作在硬件克隆模式的例子。

**图 3-9: 通过设备树设置显示控制器工作在硬件clone工作模式的样例**

如果想第二路显示管线（dvo1）从开机上电到进入图形工作一直克隆第一路显示管线（dvo0）, 则可以在的dvo1设备节点中添加**clone**属性, 如右面文本框所示的那样书写设备树。这时, 第二路显示管线作为第一路显示管线的附庸而存在, 第一路显示管线用什么分辨率, 第二路显示管线就跟着用什么分辨率。

如果想第一路显示管线（dvo0）一直克隆第二路显示管线（dvo1）, 则可以在的dvo@0设备节点中添加clone属性。这时, 第一路显示管线作为第二路显示管线的附庸而存在, 第二路显示管线用什么分辨率, 第一路显示管线就用什么分辨率。

如果不需要这种工作模式, dvo@0和dv

o@1设备节点中则不可以写clone属性。用户也不要在两条显示输出管线的设备结点中同时书写clone属性。

## **dc**@**0x400c0000** {

## compatible = "**loongson, display-subsystem**";

## reg = <0 0x400c0000 0 0x00010000>;

## interrupt-parent = <&icu>;

## interrupts = <28>;

## **output-ports** = <&**dvo0** &**dvo1**>;

## #address-cells = <1>;

## #size-cells = <0>;

## **dvo0**: dvo@0 {

## reg = <0>; /\* 0 for display pipe 0 \*/

## ddc-i2c-bus = <&i2c0>;

## connector = "hdmi-connector";

## status = "ok";

## };

## **dvo1: dvo@1** {

## reg = <1>; /\* 1 for display pipe 1 \*/ ` **clone;**

## };

## };

这种方法使能了硬件克隆方案, **不但能节省带宽, 还能够规避因LS2K1000中两路CRTC只有一个硬件光标无法在两路显示管线上同时使用的问题。** 使用硬件克隆方案时, 硬件光标能同时显示在两个屏幕上, 因为这种硬件克隆方案只有一路CRTC用DMA从系统内存读取数据, 另一路显示管线直接使用被拷贝的那路CRTC从系统内存读取Framebuffer数据和光标数据混合完成后的结果, 而不是自己用DMA去系统内存中读取Framebuffer和光标的拷贝。

这种工作模式唯一的小缺点是不能运行时修改克隆模式为扩展模式。 因为驱动程序只在初始化时解析一次设备树, 除status属性外被标记为clone的那路显示管线不会再解析其它属性了, 驱动程序也不会为这路显示管线创建资源。**被标记为clone的那路显示管线也不再需要占用一个i2c硬件资源**。

## **3.6 背光支持**

**背光驱动也是液晶屏驱动的一部分**, **如果背光没有成功点亮**, **即使显示屏本身有输出**, **那么屏幕看上去也是黑的**。**图3-10**给出了一个关于背光支持的设备树样例, 左边是背光设备节点的设备树样例, 背光设备节点的名字为backlight;右边是背光所依赖的pwm控制器的设备树书写方法。其中lcd\_backlight是背光设备节点的标号, 用于被其他设备节点所引用。pwm0是pwm控制器的设备树的标号, 背光设备节点引用了pwm0设备节点, 它们之间存在一定的联系和制约。

**图 3-10：背光设备驱动节点及其属性（左）和PWM（右）设备树书写示例**

## **lcd\_backlight**: **backlight** {

## **compatible** = "**pwm-backlight**";

## **pwms** = <&**pwm0** **0 4000000**>;

## 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 >;

## default-brightness-level = <80>;

## };

如果不需要这种工作模式, dvo@0和dv

o@1设备节点中就不要写clone属性。

**pwm0: pwm@1fe22000{**

compatible = "loongson,ls2k-pwm";

reg = <0 0x1fe22000 0 0x10>;

clock-frequency = <125000000>;

interrupt-parent = <&icu>;

interrupts = <24>;

**#pwm-cells = <2>;**

**};**

**pwm1: pwm@1fe22010{**

compatible = "loongson,ls2k-pwm";

reg = <0 0x1fe22010 0 0x10>;

clock-frequency = <125000000>;

interrupt-parent = <&icu>;

interrupts = <25>;

**#pwm-cells = <2>;**

**};**

背光设备节点中第一个属性为**compatible**, 属性的值为 "**pwm-backlight**"。这个表示我们用的是基于pwm的背光设备驱动, 其代码位于linux内核中的drivers/video/backlight/pwm\_bl.c中。这个基本不可更改, 除非你的背光控制不是PWM的, 需要自己写驱动。

背光设备节点中的第二个属性为**pwms**属性, pwms = <&**pwm0** **0 4000000**>表示背光使用了**pwm0**控制器设备来驱动液晶屏背光灯泡, LS2K1000 中集成了4路硬件pwm控制器, 选用哪一个控制器是pcb工程师根据电路板的用途和需求布线时决定的。如果你的电路板用pwm1来做背光控制, 那么就可以改成pwms = <&**pwm1** **0 4000000**>。

**pwm控制器的设备树节点中必须用#pwm-cells设备节点指定pwm控制器驱动接受的参数的数量**, 这里 **#pwm-cells = <2>** 表示pwm控制器驱动接受两个参数。 相应地, 背光设备驱动节点中的pwms = <&**pwm0** **0 4000000**>中的 **0** 和 **4000000**就是这两个参数。第二个参数**0** 表示这个PWM驱动所控制的PWM设备的序号, 当前我们产品内核中每个pwm驱动实例都只控制一个硬件pwm设备, 所以填写0即可。第三个参数**4000000** 则是用来设置背光驱动所用的这个pwm设备的工作周期的, **单位是纳秒**。

如果**#pwm-cells**写成**#pwm-cells = <3>**, 那就表示pwm控制器驱动接受三个参数, 那么背光设备节点中的pwms属性就可以写成：

pwms = <&**pwm0** **0 4000000 PWM\_POLARITY\_INVERTED**>

或

pwms = <&**pwm0** **0 4000000 PWM\_POLARITY\_NORMAL**>

也就是说 linux内核中pwm控制器的设备驱动接受的第三个参数是用来设定该pwm设备的初始工作极性的。

brightness-levels属性用来存放用来指示背光亮度级别的数值, 这里我们给定了100个背光级别。在 PWM控制器设备驱动正常工作情况下, 0 表示最暗, 达到关屏的状态; 99 表示最亮, 是背光能达到的最亮的亮度。

需要注意的是, 有时PWM设备驱动的工作极性与正常期望是相反的, 就是给定PWM控制驱动的占空比越大, PWM控制器输出的电平平均越低, 显示屏亮度越暗。在这种情况下可以将brightness-levels中的数值反过来写, 如**图3-11中左边设备树**所示。

**图 3-11: 背光设备驱动节点及其属性（左）和PWM（右）的设备树书写示例**

## **lcd\_backlight**: **backlight** {

## **compatible** = "**pwm-backlight**";

## **pwms** = <&**pwm0** **0 4000000**>;

## brightness-levels = <

## 99 98 97 96 95 94 93 92

## 91 90 89 88 87 86 85 84

## 83 82 81 80 79 78 77 76

## 75 74 73 72 71 70 69 68

## 67 66 65 64 63 62 61 60

## 59 58 57 56 55 54 53 52

## 51 50 49 48 47 46 45 44

## 43 42 41 40 39 38 37 36

## 35 34 33 32 31 30 29 28

## 27 26 25 24 23 22 21 20

## 19 18 17 16 15 14 13 12

## 11 10 9 8 7 6 5 4

## 3 2 1 0 >;

## default-brightness-level = <80>;

## };

**pwm0: pwm@1fe22000{**

compatible = "loongson,ls2k-pwm";

reg = <0 0x1fe22000 0 0x10>;

clock-frequency = <125000000>;

interrupt-parent = <&icu>;

interrupts = <24>;

**#pwm-cells = <2>;**

**};**

**pwm1: pwm@1fe22010{**

compatible = "loongson,ls2k-pwm";

reg = <0 0x1fe22010 0 0x10>;

clock-frequency = <125000000>;

interrupt-parent = <&icu>;

interrupts = <25>;

**#pwm-cells = <2>;**

**};**

default-brightness-level 属性是用来设定背光设备的默认工作的档位的, 取值范围根据需求, 在brightness-levels 属性的最小值和最大值之间取值即可, 开机pwm设备驱动初始化程序。

如果觉得示例中的背光级别过多, 那么可以写少一些。 但是填写的级别越多, 背光亮度就能够调节的越细腻。下面两行设备树代码给出了一种精简背光调级的设备树写法。

brightness-levels = < 90 80 70 60 50 40 30 20 10 0 >;

default-brightness-level = <8>;

这时背光驱动只有10个档位（0～9）, 档位 0 最亮, 档位 9 最暗。

#### **3.6.1 背光亮度计算公式**

**duty** = levels [brightness] x period / scale

其中 levels 是存放背光调节档位的数组, 就是由设备树中提供的 brightness-levels属性所形成的数组。scale是level[]数组中存储的数值的最大值, 代表最大的背光档位。brightness 代表的是用户输入的一个索引值, 比如设备树中如果写了default-brightness-level = <8>这样的设备树代码, 那么液晶屏开机上电默认的背光亮度就会被背光驱动的初始化程序调节到level[8]所代表的亮度值。 如果是运行时通过sysfs接口调节背光, 比如：

echo 80 > /sys/class/backlight/backlight/brightness

那么这时就代表要把背光调节到 level[80] 这个数值代表的档位。当然这个数值也不能无限的大, 如果用户给定的数值超过最大档位数量, 背光档位就按最大档位算。

period就是设备树中指定的PWM控制器要工作在的周期（单位是纳秒）。duty 就是一个周期中高电平的时间（单位是纳秒）。

按**图 3-10**所提供的设备树示例的设定, duty 的数值为80 x 4000000 / 100 = 3200000, 背光驱动初始化程序再调用pwm\_config() 函数将PWM控制器应工作的周期（period）和这一个周期中高电平时间（3200000, 单位纳秒）传递给PWM控制器驱动。

#### **3.6.2 通过sysfs接口调节背光**

# **cd /sys/class/backlight/backlight/**

查看当前背光亮度级别：

# **cat brightness**

查看最大背光亮度级别：

**# cat max\_brightness**

在 0 到 max\_brightness的数值中选择一个档位， 写到brightness中。

假设最多是100个档， 设置到最强亮度的80% ：

**# echo 80 > brightness**

假设最多是10个档， 设置到最强亮度的80% ：

**# echo 8 > brightness**

#### **3.6.3 通过电源首选项对话框调节背光**

如果您使用的是Loongnix系统, 那么打开mate桌面的控制中心的电源管理选项卡, 拖动按钮进行设置, 如**图 3-12**所示。

****

**图 3-12: 拖动按钮来改变背光强度**

# 4. 基于OF GRAPH的新设备树传参规范

为了能优雅地支持非透明类型的编码器而又不把其他厂家的显示编码器驱动耦合到我们LSDC设备驱动程序中来, 我们采用了 DRM Bridge框架编写了显示编码器驱动, 这也间接导致了我们需要采用新的设备树规范。老的设备树规范不是不能支持非透明类型的编码器驱动, 老设备树规范实际上还是具有可扩展性的, 但是设计之初是为了LS2K系列芯片内部的编码器准备的。必要时我们会在这个老设备树规范的基础上继续扩展, 来对非透明类型的编码器进行支持。

对于LS2K1000芯片外部的视频编码器, 由于和LS2K1000在物理上不具备紧耦合的关系。我们选择基于DRM Bridge框架来书写几款编码器的驱动。支持DRM Bridge框架, 不但能利用社区已经写好的显示编码器驱动, 也能将我们的DC设备驱动和显示编码器驱动解耦合, 实现独立开发和发行的功能。当需要支持一款新的编码器驱动时, 只需按照DRM Bridge框架写出足够品质的设备驱动即可。只要使用这款电路板的开发者根据电路图写好设备树, LSDC驱动就能自动绑定编码器的设备驱动, 协同完成显示工作, LSDC设备驱动本身不需要修改一行代码。

本章节所介绍的设备树书写方法又被称为OF GRAPH, 读者可以结合龙芯产品内核源码目录下的

Documentation/devicetree/bindings/graph.txt和

Documentation/devicetree/bindings/video-interface.txt

两篇文档来参看本文档。

本章节的余下小节, 主要以示例的方式介绍DRM Bridge类型显示编码器驱动及其设备树书写方法。

## **4.1 新设备树传参数规范下集显设备树的基本书写方法**

新设备树传参规范下, 显示控制器设备节点的写法如下**图4-1**所示。

**图 4-1: 新设备数传参数规范下显示控制器设备节点的书写方法**

**dc@0x400c0000** {

compatible = "loongson,display-subsystem";

reg = <0x400c0000 0x00010000>;

interrupt-parent = <&icu>;

interrupts = <36>;

dma-mask = <0x00000000 0xffffffff>;

#address-cells = <1>;

#size-cells = <0>;

**ports {**

#address-cells = <1>;

#size-cells = <0>;

port@0 {

reg = <0>;

dvo0：**endpoint** {

remote-endpoint = <&vga\_encoder\_in>;

};

};

port@1 {

reg = <1>;

dvo1：**endpoint** {

remote-endpoint = <&hdmi\_encoder\_in>;

};

};

**};**

};

在dc@0x400c0000 设备节点中没有书写output-ports属性, 而是ports设备节点。ports设备节点是dc@0x400c0000 设备节点的子节点, port@0设备节点和port@1设备节点是ports设备节点的子节点, 简称为端口。port@0代表的是第一路显示管线的相关属性, 可以简称为端口0, 它代表我们显示控制器的dvo0输出端口; port@1代表的是第二路显示节点的相关属性, 可以简称为端口1, 它代表我们显示控制器的dvo1输出端口。

每个port设备节点中可以包含多个endpoint设备节点, 但是由于我们的显示控制器设备比较简单。我们的port设备节点中只包含一个endpoint设备节点, 简称端点。endpoint设备节点用于被其它设备节点所引用。每个端点必须包含一个remote-endpoint属性, 用于表示这个端口连接到哪个远程设备。　同理, 这个远程设备的endpoint属性中也必须包含一个remote-endpoint属性, 这个remote-endpoint属性对我们显示控制器中的设备节点中的endpoint节点进行引用。这样两个物理设备之间的连接关系就建立起来了。

LS2K1000中的显示控制器有两个显示输出端口, 每个显示输出端口中只有一个端点, 用于被其他设备节点所引用; 每个端点中都有一个remote-endpoint属性, 用于指定和显示编码器设备节点的连接关系。**具有连接关系的两个设备节点之间需要互相指定连接关系。**

需要注意的是port、endpoint和remote-endpoint设备节点本身只用来表示连接关系的存在, 不能用来表示连接的类型, 也不能用来表示连接关系中的数据流方向。对于**图4-1**中所示的显示控制器设备节点来说, port@0和part@1都是输出端口。**图4-1**中的vga\_encoder\_in和hdmi\_encoder\_in则是显示编码器设备节点的endpoint设备节点的标号, 读者需要根据产品的实际情况填写即可。读者可以对照本章节中的**图4-2**和**图4-4**综合来看。用户可以根据自己板卡的实际情况更改标号的名字, 只要是在设备树中是合法的字符串就行。

受制于篇幅的原因, 本章节中多是以片段的形式给出设备树示例, 在实际的使用场景下需要将LS2K1000 SoC和主板上的设备描述完整才能使用。请读者结合龙芯产品内核中arch/loongarch/boot/dts/loongson/和arch/mips/boot/dts/loongson/目录中给出的完整设备树文件参看本文档。

## **4.2 LT8618 DVO转HDMI编码器设备树书写示例**

对于LS2K1000 SoC来说, 如果要实现HDMI显示输出, 那么显示编码器是整个显示管线中的必不可少的一个硬件功能单元。LT8618是Lontium的一款低功耗版的DVO信号到HDMI信号的转换芯片。它支持HDMI1.4协议规范。它也与SiI9030和ANX9030等型号的转换器管脚功能兼容。

**图 4-2： LT8618 DVO转HDMI设备树节点书写示例**

**i2c1**: **i2c@1fe01800** {

compatible = "loongson,ls-i2c";

reg = <0x1fe01800 0x8>;

#address-cells = <1>;

#size-cells = <0>;

**lt8618 {**

**compatible = "lontium,lt8618";**

**reg = <0x39>;**

**/\* hpd-gpios 　　= <&pioA 0 GPIO\_ACTIVE\_HIGH>; \*/**

**/\* reset-gpios = <&pioA 39 GPIO\_ACTIVE\_LOW>; \*/**

**ports {**

**#address-cells = <1>;**

**#size-cells = <0>;**

**port@0 {**

**reg = <0>;**

hdmi\_encoder\_in**: endpoint {**

**remote-endpoint = <&dvo1>;**

**};**

**};**

**port@1 {**

**reg = <1>;**

**hdmi\_encoder\_out: endpoint {**

**remote-endpoint = <&hdmi\_connector\_in>;**

**};**

**};**

**};**

**};**

};

**hdmi-connector** {

type = "hdmi-a";

status = "ok";

port@0 {

**hdmi\_connector\_in**: endpoint {

remote-endpoint = <&**hdmi\_encoder\_out**>;

};

};

};

LT8618芯片被不少LS2K1000的开发板使用, 龙芯中科系统研发部图形组向龙芯产品内核提交了LT8618编码器的设备驱动。 **图4-2**给出了一个LT8618的设备树书写示例, 在这个设备树示例中, 我们用LT8618设备节点描述LT8618芯片所具备的属性。**LT8618中ports节点以及ports节点中的子节点用于描述LT8618与主板上其它硬件设备的连接关系**。因为在我们的开发评估板子上这款芯片是通过ls2k1000中的i2c控制器（i2c@1fe01800）来控制的, 所以它被置于i2c1设备节点之中。这样做表示的是**LT8618就是i2c1总线控制下的一个子设备**, 这也隐含说明了LT8618和i2c1设备之间具有连接关系。注意, 在其它按非透明方式使用lt8618的开发板上, **lt8618设备节点及其属性需要被书写到用于配置该编码器的i2c控制器的设备树节点之中**。

LT8618设备节点中的compatible属性是lontium,lt8618, 这个写成其他的字符串会导致lt8618的内核设备驱动模块不被加载。

LT8618设备节点中的reg属性是用来指定i2c总线和lt8618设备之间的通信编程地址的，这个地址实际上可以通过 LT8618芯片的CI2CA管脚控制, 当CI2CA管脚被拉高时, LT8618作为从机的I2C设备地址是0x76; 当CI2CA被拉低时, LT8618作为从机的I2C设备地址是0x72。在i2c通信协议中用最低位表示读写, 所以真正有效的地址是7位。0x76 >>1=0x3b, 0x72 >>1 = 0x39。所以对于 LT8618芯片来说 reg的有效取值只有0x3B和0x39。使用LS2K1000 SoC的开发者需要根据自己电路板的布线情况如实填写这个地址到reg属性中。

hpd-gpios 用于指定LT8618的热插拔输出管脚连接到ls2k1000的64个GPIO中的哪一个上了。如果电路板上没有连接请不要写这个属性。

reset-gpios 用于指定LT8618的硬件复位管脚连接到ls2k1000的64个GPIO中的哪一个上了。如果电路板上没有连接请不要写这个属性。

**LT8618 设备节点中的ports属性及其子节点用于指定LT8618设备与其它设备之间的连接关系**。 **ports属性中的port@0设备节点用于指定 LT8618 与LS2K1000的显示控制器输出端口的连接关系**。**ports属性中的port@1 设备节点用于指定 LT8618 与显示连接器的连接关系**。**图4-2**中所示的设备树表示LT8618编码器输入端口连接到了dvo1标号所代表的设备节点上, LT8618编码器输出端口连接到了HDMI显示接口上了。也就是说在物理上, LT8618编码器的显示数据输入端口连接到了LS2K1000的DVO1数据输出端口（第二路显示管线）上了。LT8618编码器输出连接到了主板上的HDMI显示接口。这里面hdmi-connector也需要作为一个不可探测且不需要驱动的设备节点出现在设备树中, 用于表现连接关系。

其它的非透明类型编码器芯片的设备树书写方法与**图4-2**给出的示例类似。

## **4.3 Sii9022 DVO转HDMI编码器设备树书写示例**

**图4-3**给出了sii9022编码器当作非透明编码器使用的设备树写法的示例。

**图 4-3： Sii9022非透明类型编码器在新规范中的设备树书写示例**

i2c1: **i2c@1fe01800** {

compatible = "loongson,ls-i2c";

reg = <0x1fe01800 0x8>;

interrupt-parent = <&icu>;

interrupts = <31>;

#address-cells = <1>;

#size-cells = <0>;

**sil9022acnu** {

compatible = "sil,sii9022";

reg = <0x39>;

hpd-gpios = <&pioA 0 GPIO\_ACTIVE\_HIGH>;

interrupt-parent = <&pioA>;

**ports** {

#address-cells = <1>;

#size-cells = <0>;

port@0 {

reg = <0>;

hdmi\_encoder\_in: endpoint {

remote-endpoint = <&dvo1>;

};

};

port@1 {

reg = <1>;

**hdmi\_encoder\_out**: endpoint {

remote-endpoint = <&**hdmi\_connector\_in**>;

};

};

};

};

};

**hdmi-connector** {

type = "hdmi-a";

status = "ok";

port@0 {

**hdmi\_connector\_in**: endpoint {

remote-endpoint = <&**hdmi\_encoder\_out**>;

};

};

};

## **4.4 透明类型DVO转VGA编码器在新设备树规范下的书写示例**

透明类型的编码器由于不需要被程序配置就能使用, 在实际应用中较为广泛。尽管不需要被显示控制器驱动本身来配置, 但还是要在内核态针对这款编码器写设备驱动。现代DRM驱动框架也将一路Encoder外加一路Connector封装成一个DRM Bridge对象作为一路显示输出, 实现了编码器驱动和显示控制器驱动的解耦合。实际上即使是透明类型的编码器芯片, 也往往提供一些能通过GPIO来控制的固定功能的管脚（如功耗控制管脚）。**图4-4**以透明式 DVO转 VGA类型的编码器为例提供了一种设备树的书写方法。

**图 4-4: ADV7125 DVO转VGA设备树书写示例**

**vga-encoder** {

compatible = "adi,adv7123", "dumb-vga-dac";

ports {

#address-cells = <1>;

#size-cells = <0>;

port@0 {

reg = <0>;

vga\_encoder\_in: endpoint {

remote-endpoint = <&**dvo0**>;

};

};

port@1 {

reg = <1>;

**vga\_encoder\_out**: endpoint {

remote-endpoint = <&**vga\_connector\_in**>;

};

};

};

};

**vga-connector** {

ddc-i2c-bus = <&i2c0>;

port {

**vga\_connector\_in**: endpoint {

remote-endpoint = <&**vga\_encoder\_out**>;

};

};

};

与非透明类型编码器不同的是透明类型的编码器的设备节点不需要i2c进行配置, 硬件上没有也没有连接关系, 所以就不需要写到i2c控制器的设备节点中。但是，**透明类型编码器需要在与该编码器相连接的Connector设备节点中用ddc-i2c-bus属性来指定这路显示管线所用的i2c设备**。

其次, compatible的属性的值需要这款芯片的内核态驱动的兼容列表来更改。 因为不同的编码器使用的驱动是不同的, 所以需要给不同类型的编码器的设备树节点写对compatible属性的值, 写错或者所写compatible所代表的驱动与设备不兼容将会导致整个显示子系统不能正常工作。

**vga-encoder设备节点中的port@0设备节点用于指定与这路vga encoder相连接的显示数据输出端口**。从图中可以看出这个DVO转VGA显示编码器与LS2K1000 SoC的DVO0（也就是第一路显示管线）相连接。如果在您的主板上这个DVO转VGA显示编码器与LS2K1000 SoC的第二路显示管线相连接, 那么port@0节点中remote-endpoint属性可以写为remote-endpoint = <&**dvo1**>。

**vga-encoder中的port@1设备节点用于指定与这路DVO转VGA编码器相连接的显示连接器设备**。**与vga-encoder设备节点相连接的vga-connector设备节点则需要用ddc-i2c-bus 属性指定VGA这路显示管线所用的i2c设备**。

其他透明类型的编码器芯片的设备树书写方法与**图4-3**给出的示例类似, 无非是换个名字。比如电路板上用的是DVO转DVI编码器, 就把vga-encoder换成dvi-encoder, 然后将comtipable属性的值替换成与所用视频转换芯片驱动相兼容的名字即可。

## **4.5 液晶屏显示设备在新设备树规范下的书写示例**

现代DRM驱动框架将一路Encoder外加一路Connector封装成一个DRM Bridge对象作为一路显示输出, 实现了编码器驱动和显示控制器驱动的解耦。液晶屏幕作为一个显示设备, 它以及控制它所需要的GPIO、ＰＷＭ背光和电源控制等也需要被整合成一个DRM Bridge类型的设备驱动模块, 不管这款液晶屏是否需要编码器芯片来驱动。**图4-5**给出了一个DPI 接口的液晶显示屏的设备树书写示例。

panel: **display@0** {

#if 1

compatible = "avic,tm070ddh03";　/\* 1024X600 \*/

#else

compatible = "innolux,at070tn92";　/\* 800x480 \*/

#endif

**backlight** = <&**lcd\_backlight**>;

**power-supply** = <&**vdd\_lcd**>;

status = "ok";

#address-cells = <1>;

#size-cells = <0>;

**port@0 {**

reg = <0>;

**panel\_in**: endpoint@0 {

remote-endpoint = <&**dvo0**>;

};

**}**;

};

## **lcd\_backlight**: **backlight** {

## compatible = "pwm-backlight";

## pwms = <&**pwm0** 0 4000000>;

## brightness-levels = < 90 80 70 60 50

## 40 30 20 10 00 >;

## default-brightness-level = <8>;

## };

**regulators** {

compatible = "simple-bus";

**vdd\_lcd**: **regulator@3** {

compatible = "regulator-fixed";

regulator-name = "+VDD\_LED";

regulator-min-microvolt = <5000000>;

regulator-max-microvolt = <5000000>;

regulator-always-on;

regulator-boot-on;

enable-active-high;

**gpio** = <&pioA **3** 0>;

vin-supply = <&vdd\_5v0\_sys>;

};

};

**图 4-5: DPI液晶显示屏在新设备树规范下的书写示例**

DPI 接口的液晶显示屏不需要视频转换芯片, 它的视频数据输入端口直接连接到LS2K1000 SoC的显示输出端口就行。DPI 接口的液晶显示屏只有一个视频数据输入端口, 即**图4-5**示例的设备树中用的display@0设备节点中的子节点port@0设备节点表示这个端口与LS2K1000 SoC显示输出的连接关系。图4-5所示的例子表示的是液晶显示屏连接到LS2K1000的第一路（DVO0）显示输出上了；若在您的电路板上, 液晶显示屏连接到LS2K1000的第二路（DVO1）显示输出上了, 请将port@0设备节点中的remote-endpoint = <&dvo0>代码属性改写为remote-endpoint = <&dvo1>。

液晶屏显示所必须的背光设备所用的背光控制和电源控制则分别用backlight和power-supply属性来给出。这里表示的是panel驱动用pwm0控制器来调节背光, backlight背光设备节点的书写方法我们在本规范文档的3.6章节已经介绍过了, 新设备树规范和老设备树规范下的背光设备的设备树书写方法都一样。

power-supply属性用于描述显示设备和背光设备的供电状况, 详情参见linux内核中的regulator子系统设备树书写规范。笔者所用的开发板是通过ls2k1000的GPIO 3这一个输出管脚控制的, 液晶屏的供电电压是５v, GPIO 3管脚输出高电平则液晶显示设备和背光设备都给电, GPIO 3 输出低电平则液晶显示设备和背光设备都断电。读者可根据自己电路的实际情况对该示例设备树进行稍加调整即可。

这里边需要注意的是display@0中compatible属性的书写方法, 每一个compatible都对应一种液晶屏时序。换句话说simple-panel这个设备驱动用不同的compatible值对应不同的液晶屏显示时序。compatible值一般是液晶屏幕的名字, 用户需要根据自己的液晶屏幕的时序参数去龙芯产品内核中的drivers/gpu/drm/panel/panel-simple.c文件中寻找兼容的compatible。panel-simple驱动已经支持了122种的液晶屏幕。如果不想每换一次屏幕就改一次compatible或者是用户用的液晶屏幕恰好不在这已经支持的122种的范围内, 可以将compatible写成panel-dpi, 然后将该液晶屏幕的时序信息填写到display@0设备节点中的panel-timing设备节点中。**只有将compatible写成panel-dpi时, panel-simple这个设备驱动才会从设备树中读取显示时序**。**panel-simple这个设备驱动只支持传一组显示时序**。

panel: **display@0** {

compatible = "panel-dpi";

**backlight** = <&**lcd\_backlight**>;

**power-supply** = <&**vdd\_lcd**>;

status = "ok";

#address-cells = <1>;

#size-cells = <0>;

**panel-timing** {

clock-frequency = <51200000>;

hactive = <1024>;

vactive = <600>;

hsync-len = <4>;

hfront-porch = <160>;

hback-porch = <156>;

vfront-porch = <11>;

vback-porch = <23>;

vsync-len = <1>;

};

port@0 {

reg = <0>;

panel\_in: endpoint@0 {

remote-endpoint = <&dvo0>;

};

};

};

这里需要注意的是, 显示时序的设备节点的名字要写成panel-timing, 不可以改成其他的名字。而panel-timing中包含的具体时序参数的意义和格式则是本文档**3.3.1**章节介绍的一样。

# 5. LS2K1000的图形驱动软件构架简介

为防止用户在定制基于LS2K1000 SoC相关的软件产品时少编、漏编相应的驱动软件包, 这一章对LS2K1000上使用的驱动软件进行简要的介绍。

LS2K1000中的显示控制器和GPU是两个设备, 所以LS2K1000 SoC的产品内核有两个**主要**的图形设备驱动： 显示控制器内核态驱动（lsdc.ko）和GPU内核态驱动（etnaviv.ko）。除了内核态驱动外还包括用户态MESA 3D驱动（lsdc\_dri.so）和用户态2D图形驱动xf86-video-loongson(loongson\_drv.so)。整套驱动采用KMS-RO构架, 其系统软件框架如**图5-1**所示。

**图 5-1: LS2K1000驱动软件组成部分与构架**

**用户态驱动层**

**drm/lsdc**

**lsdc.ko**

**GC 1000**

**GPU**

**libdrm**

**libdrm\_etnaviv**

**Mesa（lsdc\_dri.so）**

**显示控制器**

**DC**

视频编码器

**I2C**

**CMA**

**Xlib/XCB**

**xf86-video-loongson**

**loongson\_drv.so**

**X server**

**drm/etnaviv**

**etnaviv.ko**

**RAM**

**FB**

**内核层**

**LS2K1000硬件层**

**系统库层**

显示设备

LSDC依赖于drm核心层提供的CMA帮助函数来实现对dumb buffer的内存管理和PRIME技术的支持; 其中PRIME技术又依赖于DMABUF技术, 它们是被用来在内核态实现进程间buffer共享机制的。LS2K1000 SoC中的GPU是一个嵌入式类型的GPU, 由于这个GPU中不包含显示控制器硬件单元, 也不支持创建适合送显的缓冲区, 所以这个GPU还要配合公司研发的显示控制器硬件一起使用才能将渲染出来的内容显示出来。

xf86-video-loongson目前支持两种软件加速构架, 可通过conf来控制, 当 "AccelMethod" 设置为"exa"时, 用户态驱动则开启了3D支持; 若"AccelMethod"被设置为"none"时, 用户态驱动则使提供一个非硬件加速的纯软件实现。LS7A1000和LS7A2000平台用20-loongson.conf, **LS2K1000的嵌入式SoC用20-lsdc.conf**, 目前驱动已经自动包含这两个配置文件, 能被X server加载并且自动配置驱动。**图 5-2**给出了主要选项的对比。

**图 5-2: 20-loongson.conf(左) 对比 20-lsdc.conf(右)**

**Section** "OutputClass"

Identifier "LOONGSON-DC"

MatchDriver "**loongson-drm**"

Driver "**loongson**"

Option "PrimaryGPU" "true"

Option "AccelMethod" "**none**"

Option "SWcursor" "**true**"

Option "ShadowFB" "**true**"

**EndSection**

**Section** "OutputClass"

Identifier "LSDC"

MatchDriver "**lsdc**"

Driver "**loongson**"

Option "PrimaryGPU" "true"

Option "AccelMethod" "**exa**"

Option "SWcursor" "**false**"

Option "ShadowFB" "**false**"

**EndSection**

Mesa软件库中有一个负责驱动加载的模块, 是根据设备的PCI ID加载相应的驱动来使用。对于平台设备而言则是根据内核态drm驱动的名子来给不同的设备选择不同的驱动gallium 3D驱动来使用。lsdc驱动注册到drm core的驱动名为**lsdc**。若想使用3D加速, 需要保证**lsdc\_dri.so**存在且被正确加载。lsdc\_dri.so 是LS2K1000的用户态3D驱动，在mesa包中。

# 6. 参考文献

[1] <https://www.kernel.org/doc/html/latest/devicetree/usage-model.html>

[2] <https://www.devicetree.org/specifications/>