**DTB的基本调试方法**

|  |  |  |
| --- | --- | --- |
| **文件状态： [√] 草稿 [ ] 正在修改**  **[ ] 正式发布** | **部门** | **系统软件组** |
| **版本** | 0.1 |
| **作者** | **雷登均** |
| **完成时间** | **20170308** |
| **审核** |  |
| **审核时间** |  |
| **密级状态：绝密( ) 秘密( ) 内部资料(√) 公开( )** | | |

**修改记录:**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订者 | 时间 | 说明 |
| 0.1 | 雷登均 | 2017 / 3 / 7 | DTB 的基本调试方法 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

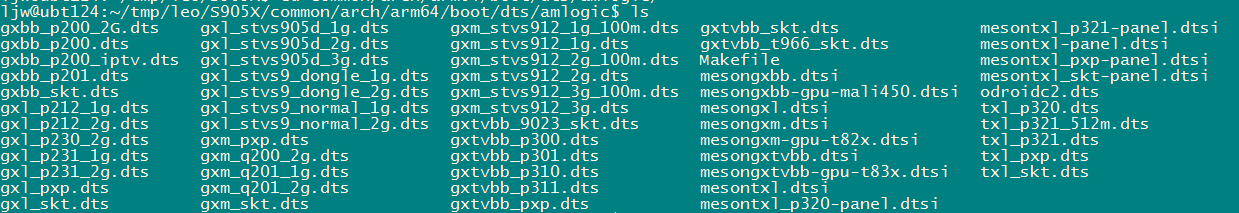
**一. 背景及问题:**

Linus Torvalds 在 2011 年 3 月 17 日的 ARM Linux 邮件列表宣称“this whole ARM thing

is a f\*cking pain in the ass” ， 引发 ARM Linux 社区的地震， 随后 ARM 社区进行了一系列的重大修正。 在过去的 ARM Linux 中， arch/arm/plat-xxx 和 arch/arm/mach-xxx 中充斥着大量的垃圾代码，相当多数的代码只是在描述板级细节， 而这些板级细节对于内核来讲，不过是垃圾， 如板上的 platform 设备、 resource、 i2c\_board\_info、 spi\_board\_info 以及各种硬件的 platform\_data。读者有兴趣可以统计下常见的 s3c2410、 s3c6410 等板级目录，代码量在数万行。

社区为了改变这种局面，就引入了DTS即Device Tree Source 设备树源码；Device Tree是一种描述硬件的数据结构，它起源于 OpenFirmware (OF)。在Linux 2.6中，ARM架构的板极硬件细节过多地被硬编码在arch/arm/plat-xxx和arch/arm/mach-xxx，采用Device Tree后，许多硬件的细节可以直接透过它传递给Linux，而不再需要在kernel中进行大量的冗余编码。

那么上文提到的DTS与我们的DTB有什么关系呢。其实DTB是dts文件的二进制表现形式是通过DTC(Device tree compiler)工具将对应DTS(Device tree source)文件转换成DTB文件;所以要修改DTB的内容就是通过修改.dts文件来达到修改DTB目的。.dts文件一般放在内核的arch/arm/boot/dts/或者arch/arm64/boot/dts/amlogic目录



**二. 思路和方法:**

**1.驱动程序与dts的关联：**

假如我们某个项目中增加了一个外设设备，这个外设设备也没有相应的驱动程序来驱动。这时我们就需要为这个外设设备重新添加一个驱动程序，并且还要修改相应的 .dts文件。下面我们以在amlogic 平台下添加一个camera的驱动为例：

1. 我们在写驱动程序的时候，所需要与之匹配的设备的资源都需要在DTB中描述出来。这时就需要在相应的.dts文件中添加我们的设备描述。如：

aml\_cams{

compatible = "amlogic,cams\_prober"; //与驱动匹配的设备描述

status = "disabled";

pinctrl-names = "gpio","csi";

pinctrl-0 = <&aml\_cam\_gpio\_pins> ;

pinctrl-1 = <&aml\_cam\_csi\_pins> ;

……

/// - ov5647

//$$ DEVICE="ov5647"

//$$ L2 PROP\_STR = "status"

//$$ L2 PROP\_U32 = "front\_back"

//$$ L2 PROP\_STR = "i2c\_bus"

//$$ L2 PROP\_STR = "gpio\_pwdn"

//$$ L2 PROP\_STR = "gpio\_rst"

//$$ L2 PROP\_U32 = "mirror\_flip"

//$$ L2 PROP\_U32 = "vertical\_flip"

//$$ L2 PROP\_STR = "bt\_path"

//$$ L2 PROP\_STR = "interface"

//$$ L2 PROP\_STR = "clk\_channel"

cam\_1{

cam\_name = "ov5647"; //camera的型号

front\_back = <0>; //前camera或者后camera 标志

i2c\_bus = "i2c\_bus\_d"; //挂载 i2c 总线的位置。

gpio\_pwdn = "GPIOH\_5"; //电源开关的gpio

gpio\_rst = "GPIOH\_4"; //复位控制的gpio

mirror\_flip = <0>; //左右镜像 控制

vertical\_flip = <0>; //上下翻转 控制

config\_path = "/system/etc/camera\_isp\_cfg/ov5647\_cw0767"; // isp 配置文件位置

bt\_path = "csi"; //csi 控制器

interface = "mipi"; //接口属性

clk\_channel = "b"; //clk的通道

status = "okay";

};

……

};

b.在程序中：设备和驱动的匹配

static const struct of\_device\_id cams\_prober\_dt\_match[] = {{

.compatible =

"amlogic, cams\_prober", //与dts文件中的aml\_cams ->compatible 进行匹配 找到amlogic camera的设备

}, {},

};

static struct platform\_driver aml\_cams\_prober\_driver = {

.probe = aml\_cams\_probe, .remove = aml\_cams\_remove, .driver = {

.name = "aml\_cams\_prober",

.owner = THIS\_MODULE,

.of\_match\_table = cams\_prober\_dt\_match,

},

};

static struct i2c\_driver ov5647\_i2c\_driver = {

.driver = {

.name = "ov5647", //在i2c总线上 与dts文件中cam\_1->cam\_name 进行匹配找到 ov5647的sensor

},

.probe = ov5647\_probe,

.remove = ov5647\_remove,

.id\_table = ov5647\_id,

};

在程序中需要获取到dts 中的属性值的方法：调用系统提供的api （common\include\linux\of.h)

I2c\_bus → of\_property\_read\_string(p\_node, "i2c\_bus", &str);

Cam\_name → of\_property\_read\_string(p\_node, "cam\_name", &cam\_dev->name);

Front\_back → of\_property\_read\_u32(p\_node, "front\_back", &cam\_dev->front\_back);

Mirror\_flip → of\_property\_read\_u32(p\_node, "mirror\_flip", &cam\_dev->m\_flip);

这样我们的驱动程序就和我们的dtb即.dts关联起来。

（这里只是一个简单的例子，如需要深入了解dts的属性。请查看跟多关于dts的资料）

**2.编译：**

在amlogic平台中单独编译 dtb 方法：

**source build/envsetup.sh**

**lunch 选择自己的项目平台 xxx**

**make dtbimage**

在out\target\product\xxx\ 目录下就会生成dtb.img

**3.烧录:**

在sd卡的更目录下放dtb.img文件 插入要烧录的平台。

在amlogic 平台的uboot下用命令烧录：

**Uboot$: mmc dev 0**

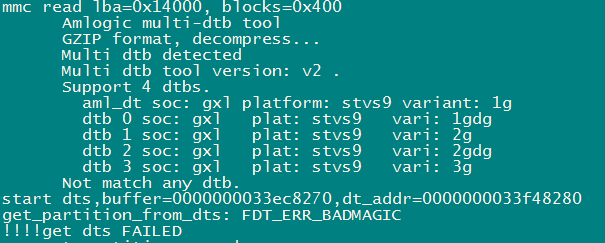
**Uboot$: mmcinfo**

**Uboot$: fatload mmc 0 $loadaddr dtb.img**

**Uboot$: store dtb write $loadaddr**

**4.DTB的调试方法：**

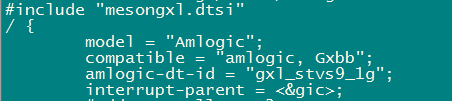
1)假如我们烧错了dtb 文件，在进入uboot的时候会提示：



在uboot启动的时候会根据

soc:**gxl** platform: **stvs9** variant:**1g**

与我们打包的dts文件中的amlogic-dt-id 的值进行对比。如：gxl\_stvs9\_normal\_1g.dts



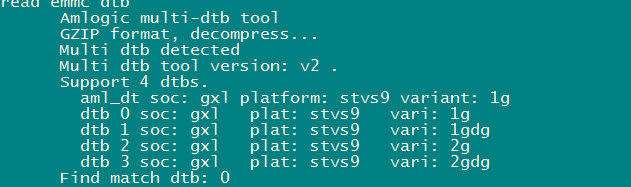
如果能找到**gxl**\_**stvs9\_1g** 系统就可以正常启动。如果不能找到就会提示：

get\_partition\_from\_dts: FDT\_ERR\_BADMAGIC

!!!!get dts FAILED

机器就会停止在uboot，不能进入系统。这种情况就是我们的dtb文件烧录错误；需要检测核对dts文件。

如果dtb没有错误：就会找出现 “ **Find match dtb: 0 ”**系统就会启动

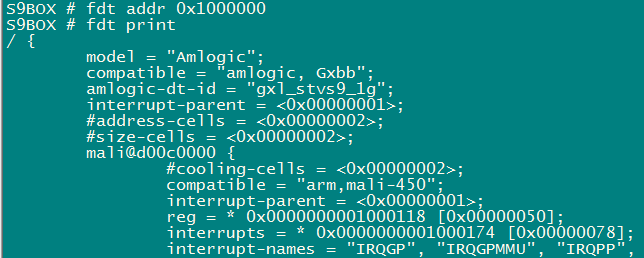


2）在uboot中对dtb进行检查核对：

执行： S9BOX # fdt addr 0x1000000

S9BOX # fdt print

就会打印当前的 所用的dtb 对应的文本。我们就可以检查我们的dtb文件配置时候有错误。



3)利用工具将烧录的dtb.img 反向提取 dts文件

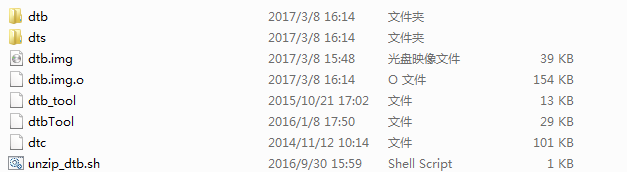
利用我们提供的工具包**dtb\_tools.rar；** 加压到dtb\_tools文件夹

****

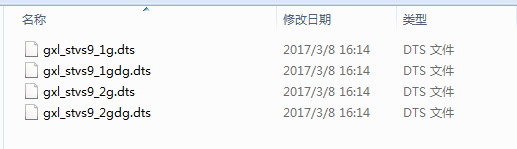
将dtb.img 文件放入有dtb\_tools文件夹。

执行：**ljw@ubt124:~/tmp/leo/S905X/dtb\_tools$ chmod +x unzip\_dtb.sh**

**ljw@ubt124:~/tmp/leo/S905X/dtb\_tools$ ./unzip\_dtb.sh dtb.img**



生成的dts文件：



4）在什么情况下要修改dtb文件。

1. 在编写驱动程序的时候，要获取硬件设备资源；需要修改dts文件。

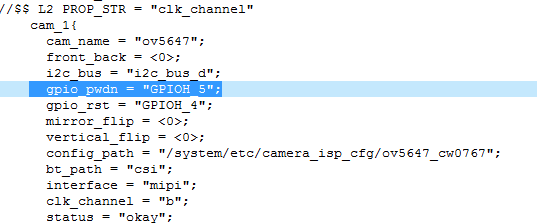
如前面所提到的增加一个camera驱动。

2. 在有些项目中所用的硬件平台修改了一些gpio的控制，其他和以前的一样。

例如：原来用GPIOH\_5 控制camera的电源，而现在硬件电路改成了GPIOH\_6.控制

那么这时候我们就只需要修改 dtb即（dts）。而不需要修改内核的程序。

**“gpio\_pwdn = "GPIOH\_5”” -----→“gpio\_pwdn = "GPIOH\_6””**



3.当发现内核启动过程中，设备的驱动没有加载，或者加载出错的时候。我们也要去检查我们的dtb文件配置是否有错误。

**三. 技术总结**

1. 在编写外设驱动的时候要更新对应的dts文件。
2. 在内核支持外挂的设备时，pcb板做了一些修改，或者外挂的一些设备发生改变，要修改dts文件