dtb 介绍

扁平设备树 ，又叫FDT，flatted device tree。如果想理解dts文件的相关配置方法，可以参考另外一份文档《DTB的基本调试方法》。dts可理解为将部分设备信息结构存放到device tree文件中。uboot会将其device tree的dts文件编译成dtb文件，使用过程中通过解析该dtb来获取板级设备信息。uboot的dtb和kernel中的dtb是一致的，uboot启动其中一个动作就是加载dtb，这个dtb和内核是共用的。

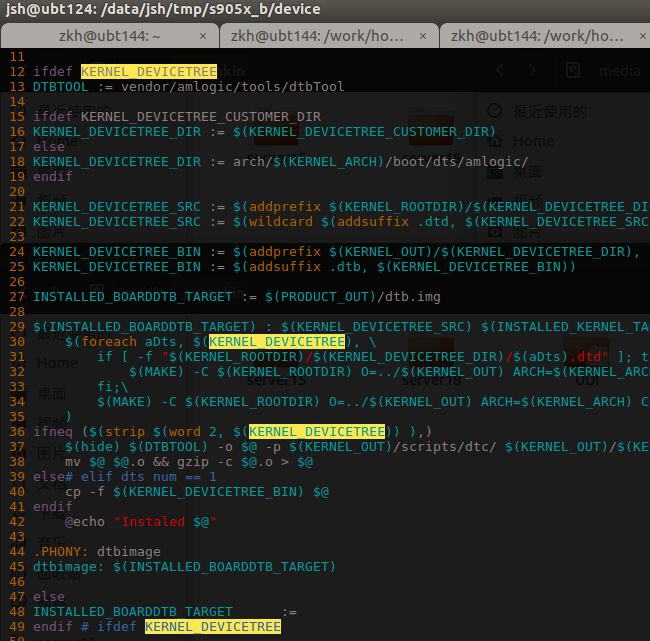
文中代码以公司 S905x为例。

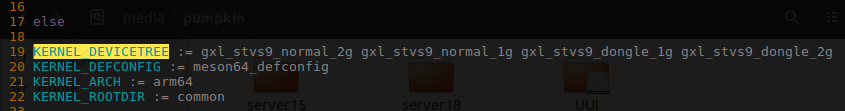
**1.编译dts：**

dts文件的编译分为两种模式，一种是编入到uboot里面，另外一种是和uboot分开编译。我们S905X项目里面现在是和uboot分开编译。

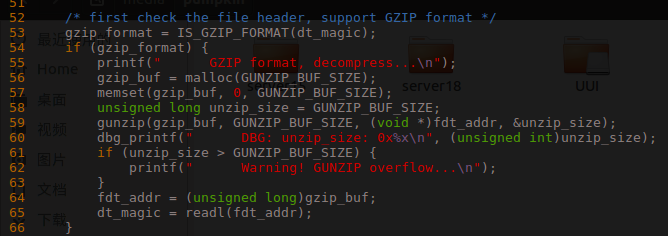
编译dtb的方法是执行完环境和lunch完产品后，直接make dtbimage就可以编译。

编译dtb的mk文件是：device/amlogic/common/factory.mk，编译规则：

 编译了哪几个dts文件，由KERNEL\_DEVICETREE决定：device/amlogic/stvs9/Kernel.mk

 factory.mk 里面有两种情况：第一种是用将单个dts编译成dtb文件，然后直接把dtb拷贝成镜像文件。第二种，假如我们编译多个dtb文件的时候，也就是支持多个dts，这时会将这几个dtb文件最终用gzip压缩到dtb.img里面。注意这个压缩，因为这个文件需要在uboot再将它做一个判断，如果是gzip格式的文件，会解压后读取里面的内容，若不是gzip格式则就会直接返回dtb的地址。

dtb是否是gzip格式识别：uboot/common/aml\_dt.c

 dtb编译出来后，我们会讲dtb烧录到指定的地址上去：

fatload usb 0 1080000 dtb.img;

store dtb write 1080000;

**2.单个dtb 的dtb.img与S905X 多个 dtb.img的数据存储结构**

单个dtb的dtb.img的来源是直接将后缀dtb的文件拷贝成img后缀的文件，其数据存储方式：

|  |
| --- |
| DTB header |
| alignment gap |
| memory reserve map |
| alignment gap |
| device-tree structure |
| alignment gap |
| device-tree string |

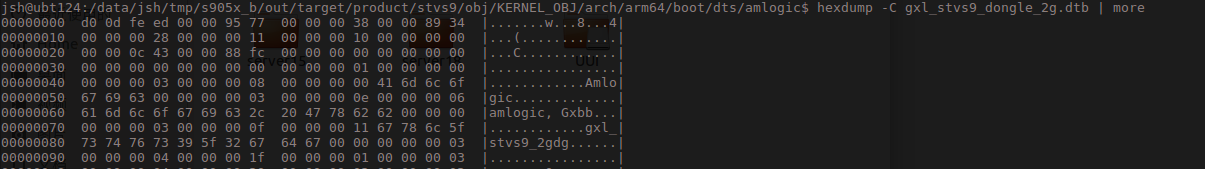
dtb header结构：

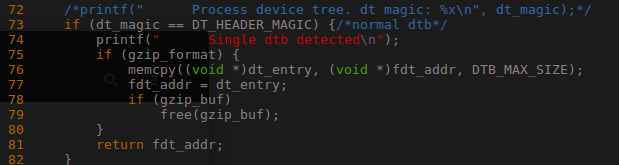
|  |
| --- |
| magic |
| totalsize |
| off\_dt\_struct |
| off\_dt\_strings |
| off\_mem\_rsvmap |
| version |
| … |

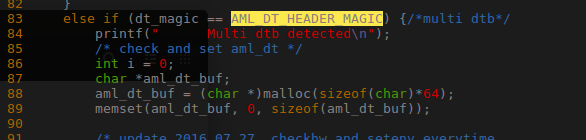
多个dtb的 Dtb.img 结构

|  |  |  |
| --- | --- | --- |
| 数据 | 偏移 | 值 |
| 头 | 第0~3字节 | 0x5f4c4d41 |
| 版本控制 | 第4~7字节 |  |
| dtb数量 | 第8～11字节 |  |
| 第一个dtb数据 | 起始12～ |  |
| 第二个dtb数据 | ... |  |
| ... | ... |  |
| 第N个dtb数据 |  |  |

其中，单个dtb做的dtb.img的dtb 头的magic是一个固定的值，0xd00dfeed（大端）或者0xedfe0dd0（小端）。用命令可以直接看到：

 可以看到dtb的前面4个字节就是**0xd00dfeed**，也就是magic。综上，我们只要提取待验证dtb的地址上的数据的前四个字节，与**0xd00dfeed**（大端）或者**0xedfe0dd0**（小端）进行比较，如果匹配的话，就说明对应待验证dtb就是一个合法的dtb，代码common/aml\_dt.c里面判断如果是单个dtb的dtb.img，会直接把fdt的地址返回回去。

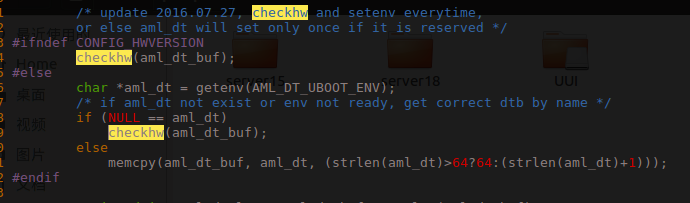
 amlogic 多个dtb打包成dtb.img的头是不一样的，前面的四个字节是：0x5f4c4d41，

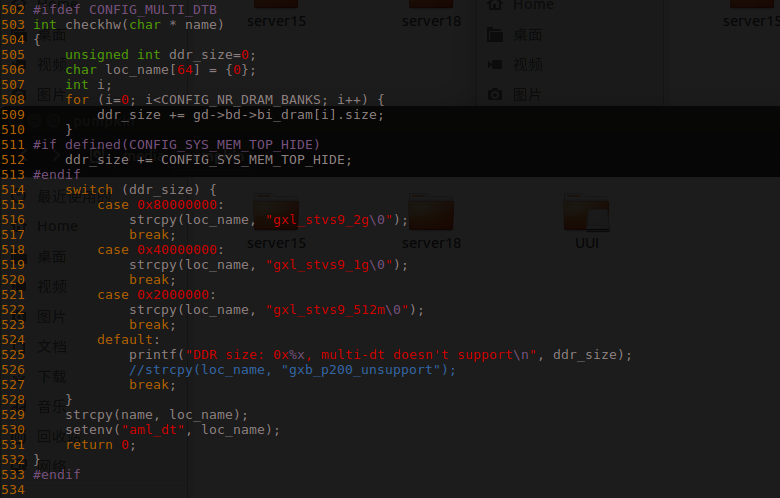
 因此，通过判断文件的头4个字节，即可知道这个dtb.img文件是不是Multi dtb文件。

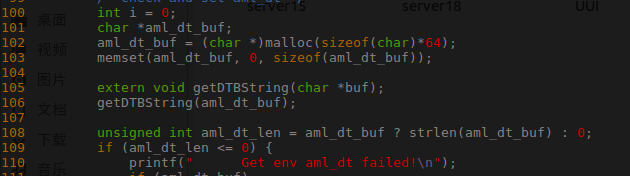
**3.多个dts打包成一个dtb，并匹配板子上的版本信息原理**

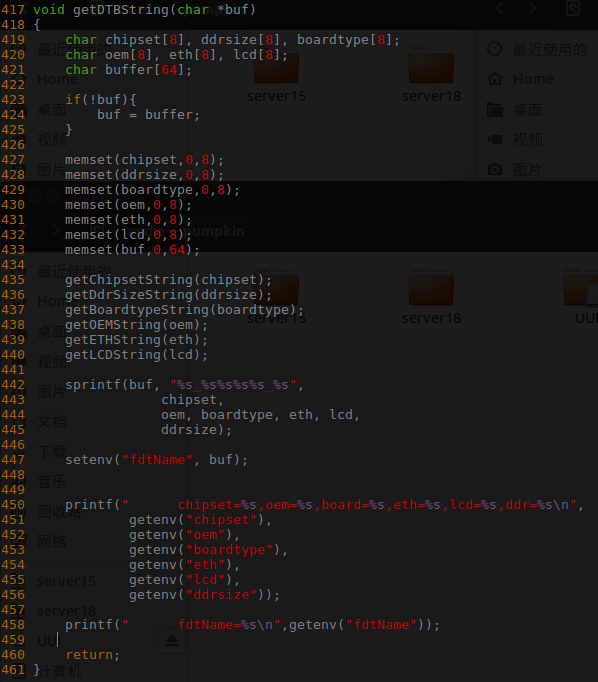
当单个dtb打包成dtb.img时，我们获取这个dtb的内存地址时，在程序上是返回找到的地址。

多个dtb打包成dtb.img文件后，我们会先判断是否是gzip格式，是的话解压出来，然后识别img里面的dtb的数量，然后获取每个dtb的信息，并和板子烧录进去的dtb对比，找到我们需要的dtb，然后通过地址偏移计算出实际的dtb的起始地址。我们来看看获取板子上的dtb的信息函数：checkhw

 其中CONFIG\_HWVERSION 在board/amlogic/configs/gxl\_stvs9\_v1.h配置为1，因此会执行 checkhw(aml\_dt\_buf).在board/amlogic/gxl\_stvs9\_v1/gxl\_stvs9\_v1.c：

 这里我们发现，这个函数没有去板子上读烧录进去的版本信息文件。只是通过读取CONFIG\_SYS\_MEM\_TOP\_HIDE的配置，case 对应的配置，然后返回对应的dts的id名。这里这个操作为什么不读板子上烧录的版本信息？暂且先不纠结这里，看一下RK3288上获取板子上版本信息的文件看看：

 106行：

 在./common/hwversion.c里面：

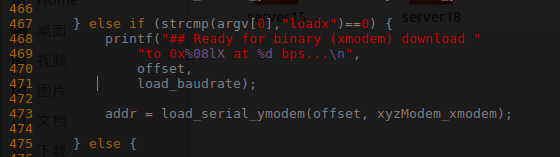
追踪getDTBString函数，我们可以看到rk3288是有通过i2c去获取板子上版本信息，然后根据芯片、DDR、board类型、oem、eth、lcd的信息，组合成dts文件的信息。

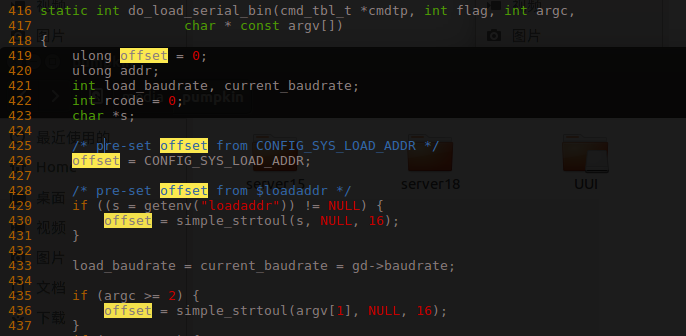
因此我们知道，它是读取板子上烧录的版本信息，然后和dtb.img的多个dtb进行匹配。

到这里，我们已经知道uboot怎么将板子上的版本信息和dtb.img里面的dtb匹配上了，但是板子里面的版本信息是怎么烧录进去的呢？

4.版本信息烧录，以rk3288为例

rk3288的版本信息用串口烧录，uboot命令行下执行loadx，然后通过xmodem把版本信息文件传输到板子上。下图可以看到loadx的命令的执行的过程，代码./common/cmd\_load.c

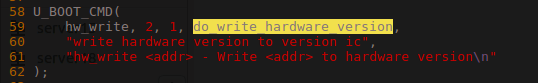
 loadx把文件下载到哪里去了？我们看看offset

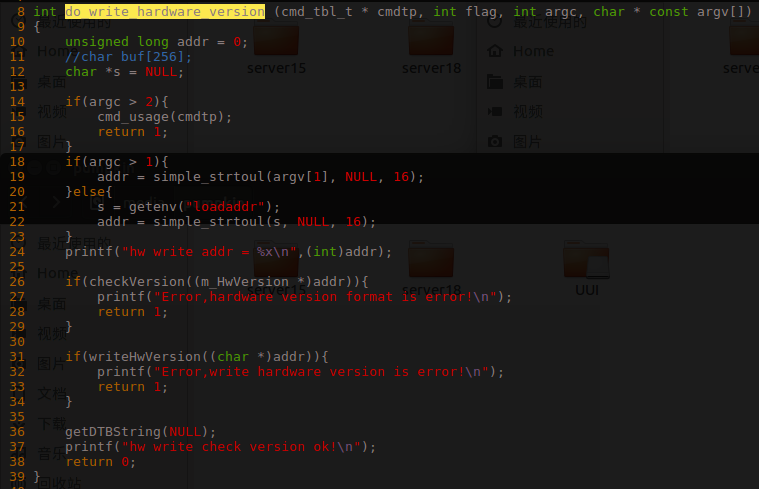
 由于直接执行loadx，没有跟参数，所以用的是默认的CONFIG\_SYS\_LOAD\_ADDR地址，这个地址最终在./include/configs/rk32plat.h里面定义：



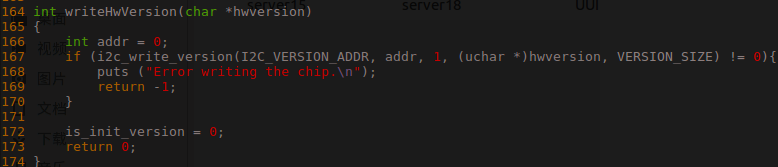
把版本信息文件读到这个地址后，用hw\_write写入保存，我们再看看这个hw\_write。

/common/cmd\_hwversion.c:

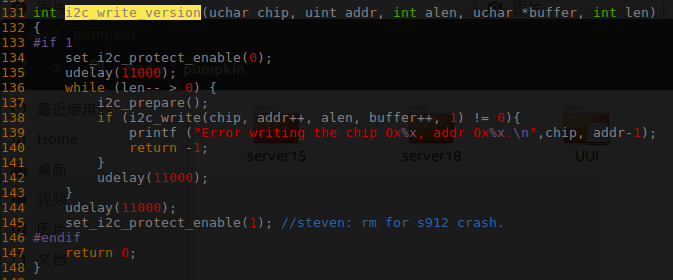




其中的writeHwVersion就是我们想要看到的，./common/hwversion.c：

 其中的I2C\_VERSION\_ADDR地址可以搜索到：

 暂且不清楚这个i2c设备是什么设备，先继续看：

 这里的i2c写入的数据是内存上的一个区域的数据，所以这里参数传入的是地址和长度，i2c\_write 已经在./drivers/i2c/rk\_i2c.c中实现了i2c驱动。