Device Tree(DTS)

1. 可描述的信息：

- CPU的数量和类别
- 内存基地址和大小
- 总线和桥
- 外设连接
- 中断控制器和中断的使用情况
- GPIO控制器和GPIO使用情况
- clock控制器和clock使用情况

各个外设的dts参考文档：U:\rk3399pro_linux\kernel\Documentation\devicetree\bindings\

2.DTS基本知识：

- dts：硬件的相应信息都会写在.dts为后缀的文件中，每一款硬件可以单独写一份xxxx.dts，一般在Linux源码中存在大量的dts文件，对于arm架构可以在arch/arm/boot/dts找到相应的dts，一个dts文件对应一个ARM的machie
- dtsi：值得一提的是，对于一些相同的dts配置可以抽象到dtsi文件中，然后类似于C语言的方式可以include到dts文件中，对于同一个节点的设置情况，dts中的配置会覆盖dtsi中的配置。
- dtc：dtc是编译dts的工具，可以在Ubuntu系统上通过指令apt-get install device-tree-compiler安装dtc工具，不过在内核源码scripts/dtc路径下已经包含了dtc工具；
- dtb：dtb(Device Tree Blob)，dts经过dtc编译之后会得到dtb文件，dtb通过Bootloader引导程序加载到内核。所以Bootloader需要支持设备树才行；Kernel也需要加入设备树的支持；
- 系统启动之后可以在根文件系统里面看到节点信息cat /proc/device-tree/,内核在启动的时候会解析设备树然后在这个目录下解析出来

3.DTS节点：

- aliases节点：是用来起别名的一个节点，比如：iic0=&iic1 iic1：iic1@0x00010
- chosen节点：并不是一个真实的设备，主要是将uboot里面的bootargs环境变量值传递给linux内核作为命令行参数cmdline
- uboot是如何向内核传递bootargs的：在uboot的fdt函数中会查找chosen节点并且在里面添加bootarg属性，属性值为bootargs值

4.DTS语法：

设备树是采用树形结构来描述板子上的设备信息的文件，每个设备都是一个节点，叫做**设备节点**，每个节点都通过一些属性信息来描述节点信息。

- 根节点："/" 每个设备树只有一个根节点（从此处开始）
- 设备树也可以有头文件dtsi
- 子节点：根节点下面的节点是子节点，设备树中节点的命名格式如下：**node-name@unit-address**

node-name：是节点名称；unit-address是节点地址或者寄存器的地址，如果没有的话可以不要。

还有一种形式：**cpu0:cpu@0（label: node-name@unit-address）**前面是节点标签，引入节点标签的目的是方便访问节点，直接通过&label来访问节点

- 属性：节点是由一堆属相组成，用户可以自定义属性，除了自定义属性，有很多属性是标准属性

①：compatible属性：属性值是一个字符串列表(兼容性列表)，compatible属性将设备与驱动程序绑定起来

```
1                    compatible = "rockchip,rk3399pro-evb-v11-linux", "rockchip,rk3399pro
```

设备首先使用第一个compatiable在linux内核里面查找与之匹配的驱动文件，如果找不到的话查找第二个。

一些驱动程序文件都会有一个OF匹配表，此OF匹配表会存储一些compatible值，值，如果设备节点的 compatible属性值和 OF匹配表中的任何一个值相等，那么就表示设备可以使用这个驱动。比如在文件 imx-wm8960.c中有如下内容：

```
632 static const struct of_device_id imx_wm8960_dt_ids[] = {
633     { .compatible = "fsl,imx-audio-wm8960", },
634     { /* sentinel */ }
635 };
636 MODULE_DEVICE_TABLE(of, imx_wm8960_dt_ids);
637
638 static struct platform_driver imx_wm8960_driver = {
639     .driver = {
640         .name = "imx-wm8960",
641         .pm = &snd_soc_pm_ops,
642         .of_match_table = imx_wm8960_dt_ids,
643     },
644     .probe = imx_wm8960_probe,
645     .remove = imx_wm8960_remove,
646 };
```

第 632~635 行的数组 imx_wm8960_dt_ids 就是 imx-wm8960.c 这个驱动文件的匹配表，此匹配表只有一个匹配值“fsl,imx-audio-wm8960”。如果在设备树中有哪个节点的 compatible 属性值与此相等，那么这个节点就会使用此驱动文件。

第 642 行，wm8960 采用了 platform_driver 驱动模式，关于 platform_driver 驱动后面会讲解。此行设置.of_match_table 为 imx_wm8960_dt_ids，也就是设置这个 platform_driver 所使用的 OF 匹配表。

可以通过根节点的compatible属性知道我们所使用的设备。内核支持的设备跟这个设备树的属性对应起来，内核才会启动

- 根节点下的compatible 用来描述支持哪个设备以及内核

②：model属性：model属性值也是一个字符串，一般 model属性描述设备模块信息，比如名字什么的。

③：status属性：状态属性（）

值	描述
“okay”	表明设备是可操作的。
“disabled”	表明设备当前是不可操作的，但是在未来可以变为可操作的，比如热插拔设备插入以后。至于 disabled 的具体含义还要看设备的绑定文档。
“fail”	表明设备不可操作，设备检测到了一系列的错误，而且设备也不大可能变得可操作。
“fail-sss”	含义和“fail”相同，后面的 sss 部分是检测到的错误内容。

④：#address-cells和 #size-cells属性：#address-cells和 #size-cells这两个属性可以用在任何拥有子节点的设备中，用于描述子节点的地址信息。#address-cells属性值决定了子节点 reg属性中地址信息所占用的字长 (32位)，，#size-cells属性值决定了子节点 reg属性中长度信息所占的字长 (32位)。#address-cells和 #size-cells表明了子节点应该如何编写 reg属性值，一般 reg属性都是和地址有关的内容，和地址相关的信息有两种：起始地址和地址长度， reg属性的格式一为：

```
reg = <address1 length1 address2 length2 address3 length3.....>
```

```
spi4{
#address-cells=<1>;
#size-cells=<0>;
 gpio_spi: gpio_spi@0{
    compatible = "fairchild,74hc595";
    reg=<0>; //设置了起始地址，没有设置长度
 }
}
```

父节点的address-cell和size-cell 描述的是子节点的信息

⑤：reg属性：reg属性的值一般是 (address length)对。注意：这两个属性一定是父节点来决定的。

⑥：ranges属性：

ranges 属性值可以为空或者按照(child-bus-address,parent-bus-address,length)格式编写的数字矩阵，ranges 是一个地址映射/转换表，ranges 属性每个项目由子地址、父地址和地址空间长度这三部分组成：

child-bus-address：子总线地址空间的物理地址，由父节点的#address-cells 确定此物理地址所占用的字长。

parent-bus-address：父总线地址空间的物理地址，同样由父节点的#address-cells 确定此物理地址所占用的字长。

length：子地址空间的长度，由父节点的#size-cells 确定此地址长度所占用的字长。

如果 ranges 属性值为空值，说明子地址空间和父地址空间完全相同，不需要进行地址转换，对于我们所使用的 I.MX6ULL 来说，子地址空间和父地址空间完全相同，因此会在 imx6ull.dtsi 中找到大量的值为空的 ranges 属性，如下所示：

⑦device_type:一般只会用在cpu和memory

- 编写外设器件的device-tree时，不同芯片平台有不同的写法，看绑定文档。kernel/documentation/device-tree/bindings

5、Linux内核的OF操作函数：内核驱动如何获取到设备树的节点信息

- 在驱动中使用OF函数获取设备树信息：/kernel/include/linux/of.h

掌握常用的OF函数（大概10个）

①extern struct device_node *of_find_node_by_name(struct device_node *from,const char *name);

from:开始查找的节点，为NULL时表示从根节点开始查找

name：要查找的节点名字。

②extern struct device_node *of_find_node_by_type(struct device_node *from,const char *type);

type：查找节点对应的字符串，也就是device_type属性，为NULL时表示忽略这个属性

③：extern struct device_node *of_find_compatible_node(struct device_node*from,const char *type, const char *compat);

compat：表示要查找的节点所对应的compatible属性列表

④static inline struct device_node *of_find_node_by_path(const char *path)

path：通过路径查找节点：比如/父节点/子节点名称（不是标签）/

返回值：查找到节点（属性/这里的属性指的是节点的各个属性值）

- **提取属性信息（提取节点的某一个属性）**

  linux中使用property表示属性

```
struct property {
    char    *name;
    int     length;
    void    *value;
    struct property *next;
    unsigned long _flags;
    unsigned int unique_id;
    struct bin_attribute attr;
};
```

Linux提供了提取某一个属性值的OF函数

①extern struct property *of_find_property(const struct device_node *np,const char *name,int *lenp);

np：设备节点

name:属性名字

lenp：属性值的字节数

返回值：找到的属性。