Linux设备树使用方法

**作者:向仔州**

设备树文件类型………………………………………………………………………………………………………..…………..2

设备树程序使用实验………………………………………………………………………………………………..…………..3

设备树格式………………………………………………………………………………………………………..………………....5

of\_property\_read\_u32\_index函数使用……………………………………………………………..………....7

of\_property\_read\_string函数使用.....................................................................................7

of\_property\_read\_u8\_array函数使用…………………………………………………………………………...7

of\_property\_read\_string\_helper函数使用……………………………………………………………..……..8

如果两个节点，节点名一样，compatible名也是一样，会出什么问题…………………..…….….8

of\_find\_node\_by\_path函数使用……………………………………………………………………………..…….9

节点之间的引用……………………………………………………………………………………………………………….…..11

Pinctrl用法………………………………………………………………………………………………………………………..….13

of\_find\_property函数使用…………………………………………………………………………………………...13

of\_get\_child\_count函数使用………………………………………………………………………………………..16

for\_each\_child\_of\_node函数使用………………………………………………………………………………..18

中断使用....................................................................................................................................20

#adderss-cells，#size-cells关键字使用………………………………………………………………………..20

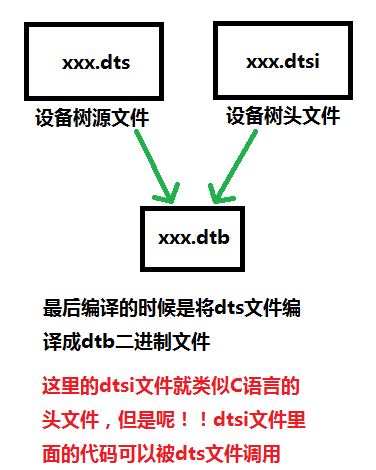
设备树时钟使用方法…………………………………………………………………………………………………………..25

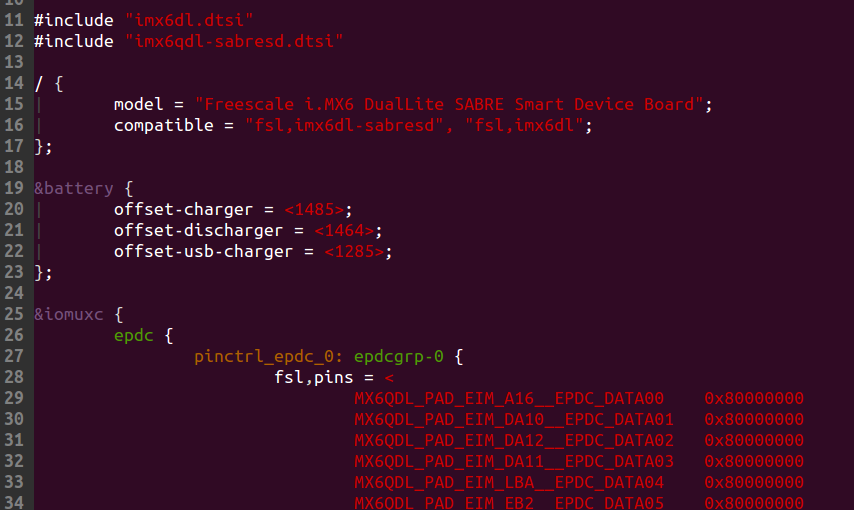
如果设备树时钟节点有多个时钟怎么操作？…………………………………………………………….26

Of\_count\_phandle\_with\_args函数使用……………………………………………………………………….26

**设备树文件类型**

设备树支持linux3.x以上的内核

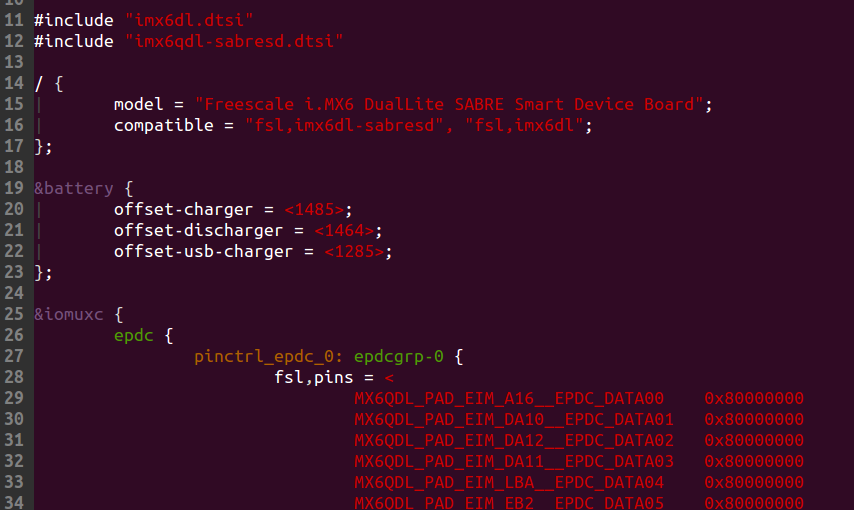


这是打开的dts文件 这是dtsi文件

dts文件才是我们要使用的文件，dtsi文件里面的代码一般都是用来写一个CPU系列通用的程序(比如我这里是IMX6)，然后根据CPU 不同的子型号(IMX6子型号有imx6Q，imx6dl，imx6s)，决定用哪一个dts文件，然后需要用到的通用程序，就不需要单独再在dts文件里面编写，直接用&调用dtsi文件里面的程序就是了。

我使用的平台是IMX6dl，所以这里的dts文件是imx6dl-sabresd.dts，我的dts文件会去调用一些dtsi文件里面的通用程序。

然后我又给max8903@0增加了新的字段

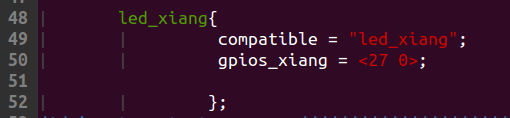
我这里的dts就调用了dtsi文件里面的battery对应的max8903@0程序

这样的结果就是驱动程序文件一旦匹配了dts文件，就会去获取max8903@0里面字符变量对应的数据，而且除了本身8903@0的数据外，还可以再获取新增加的offset...里面的数据。

这里就是给个印象，记住就是，后面会将怎么使用设备树。

**设备树程序使用实验**

我在dtsi文件里面新增加一个内容，当然你在dts增加也可以，看自己喜好。



这里的compatible属性是系统固定的，后面的字符必须和驱动程序字符写的一样

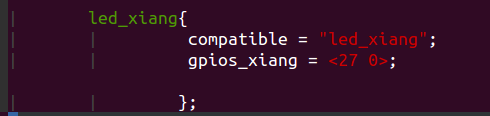
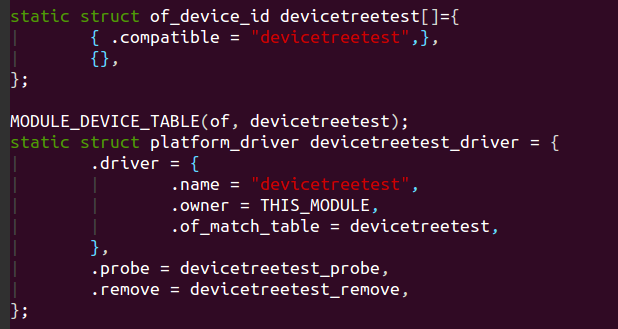
设备节点名字自己取

这就是我增加的内容

这里的compatible就取代了以前老版本platform\_device的做法，以前是xxxdevice匹配xxxdriver文件，现在是dts里面的设备节点匹配xxxdriver文件，但是dts只有一个文件，如何能匹配不同的xxxdriver文件呢? 所以这里的compatible里面的字符决定了该设备节点匹配什么驱动文件.

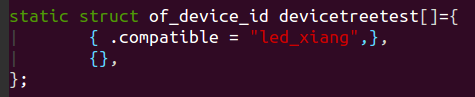


执行insmod之后发现没有执行probe函数

设备树compatible是led\_xiang 驱动文件里面compatible是devicetreetest

所以要将驱动文件里面的compatible字符改成和设备树一样

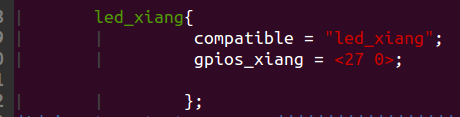
在驱动文件把compatible修改后

驱动匹配成功执行probe函数

设备树的设备节点和驱动程序匹配成功后，驱动程序就要想办法去获取设备树设备节点里面的数据了

把获取的值赋值给整形变量

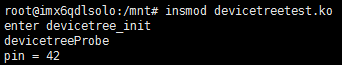
of\_property\_u32\_array(设备节点，属性名称，获取的值，获取几个值) //函数使用方法

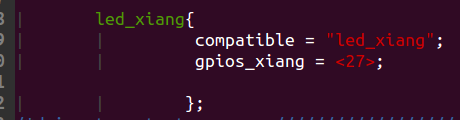
 

获取一个值

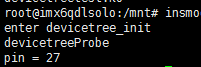
节点里面有很多属性，到底获取哪一个属性

这里的设备节点用pdev里面的of\_node获取，节点是led\_xiang

但是为什么执行驱动后得到的pin是42呢？我要求获取<27 0>尖括号的27值啊

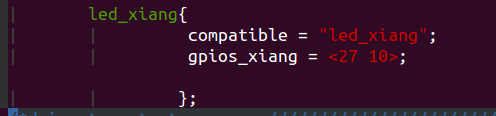


我将设备树尖括号修改成1个值

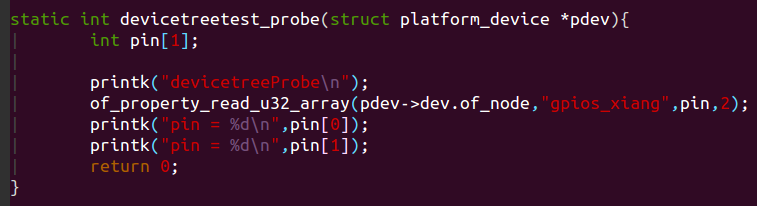


你看获取值就正常了

所以使用设备树of函数一定要注意of\_property\_u32\_array，设备树尖括号1个值或者多个值解析方式都是不一样的



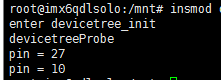
如果我把设备树改成2个值



获取尖括号的某个值就得用数组下标来获取

而且of\_property\_u32\_array必须获取2个值，如果获取1个值就会变成我前面实验的悲剧，读出的值不对

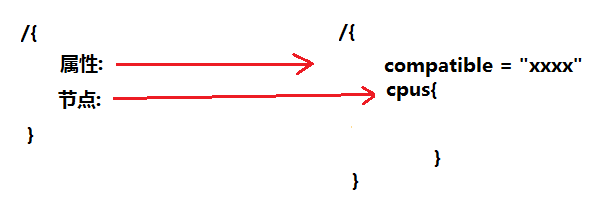
驱动程序就必须建个数组变量来承载

执行结果正确

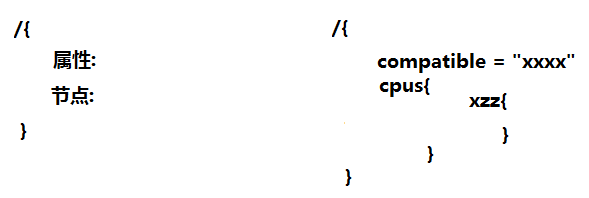
这就是设备树的特性，设备树的属性有多少个值，你驱动程序就得匹配多少变量去获取。

**设备树格式**

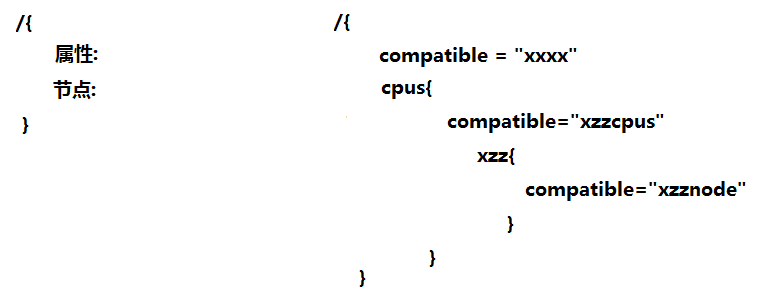
每个根/目录下只能有一个compatible，这个compatible是关键字，不能乱写，驱动就是靠compatible去找对应的节点



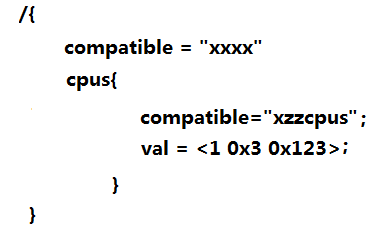
每个根节点下可以有很多子节点，比如cpus就是我取名的一个子节点



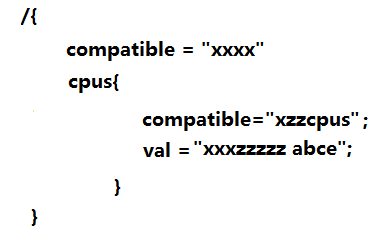
子节点下面还可以增加子节点



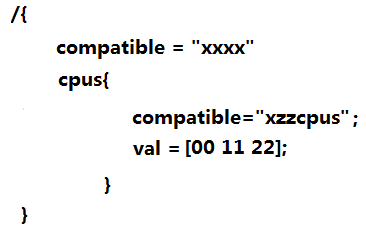
但是每个子节点只能有一个compatible，方便驱动程序去寻找



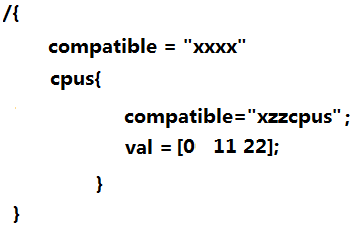
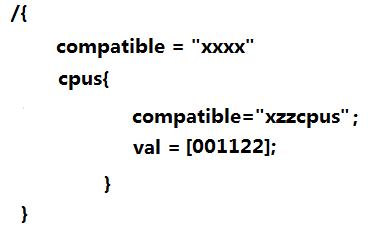
除了compatible和其它少许的关键字属性名不能变化以外，其余的属性名可以随便取，我这里取了个val属性，<>尖括号里面表示多个32位的数据值，分号不要忘记写了



属性也可以传字符串



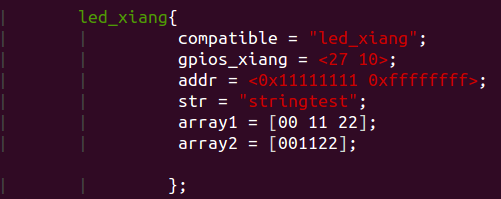
属性如果用[]中括号，就表示里面存放的是多个16进制的字节数据

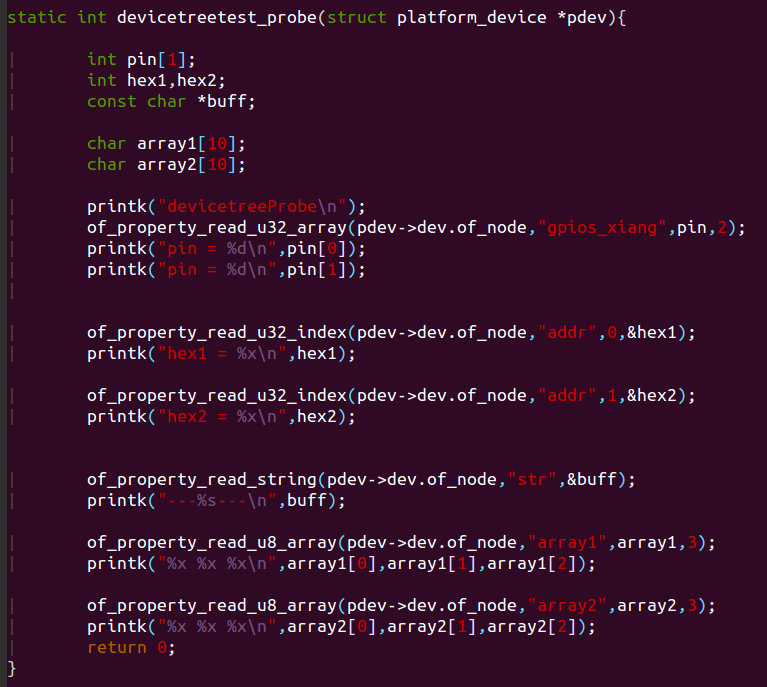
[]中括号里面可以不用空格，字节数据可以挨着写，但是驱动程序解析的时候还是一个字节一个字节获取

[]中括号里面必须是两个16进制一个字节，如果写一个16进制就会出错

下面我们用几个of函数来试试

这是设备树内容

of\_property\_read\_u32\_index(设备节点，属性名称，获取尖括号第几项数据，存入获取的数据)



这是获取第1项0xffffffff



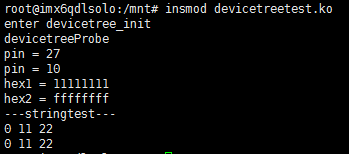
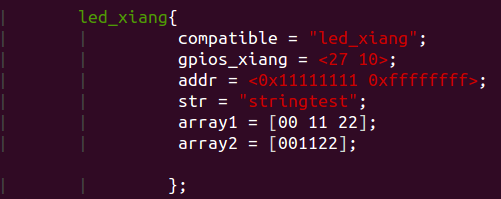
像这种两个32位的数据，就要用index函数来指定下标一项一项获取，这是获取第0项0x11111111

of\_property\_read\_string(设备节点，属性名称，用const \*char变量来接受字符串)

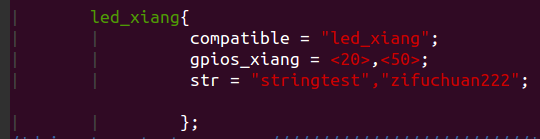
像这种字符串的设备树就要用这个函数。

of\_property\_read\_u8\_array(设备节点，属性名称，存放字节数组变量，获取中括号第几项数据)

不管是分开的字节数据，还是连在一起的字节数据，最后of\_property\_read\_u8\_array获取出来都会将其分开成一个字节一个字节的。

驱动程序获取结果和设备树定义的数据是一样的，没有问题。



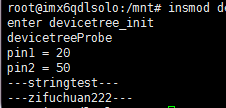
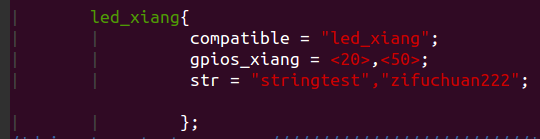


获取设备树属性的第几项字符串，主要是看index这个值

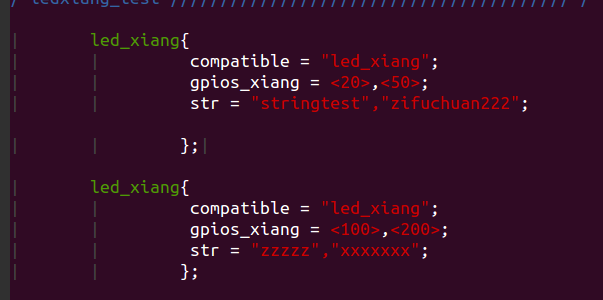
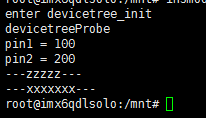
获取几个字符串，这里默认写1就是了

这种两个尖括号也是用index去查询

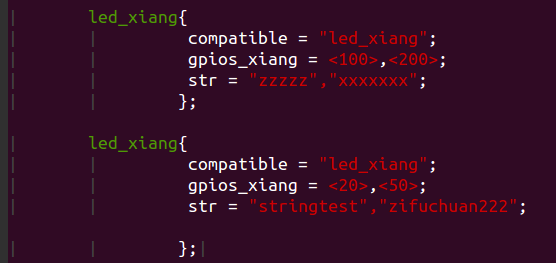
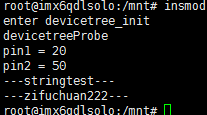
of\_property\_read\_string\_helper(设备节点，属性名称，获取值的变量一定是const char\*变量，获取多少个这里就写1，设备树第几项字符串)

**如果两个节点，节点名一样，compatible名也是一样，会出什么问题**

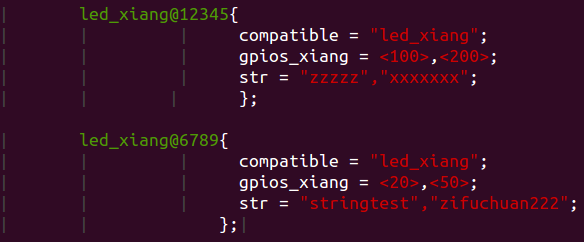
 

如果有两个同名，同compatible的设备节点，那么驱动程序会忽略前面同名的设备节点，而去读取最后一个位置的设备节点

我把<20>,<50>放到后面位置，你看驱动就去读取<20>,<50>的设备节点

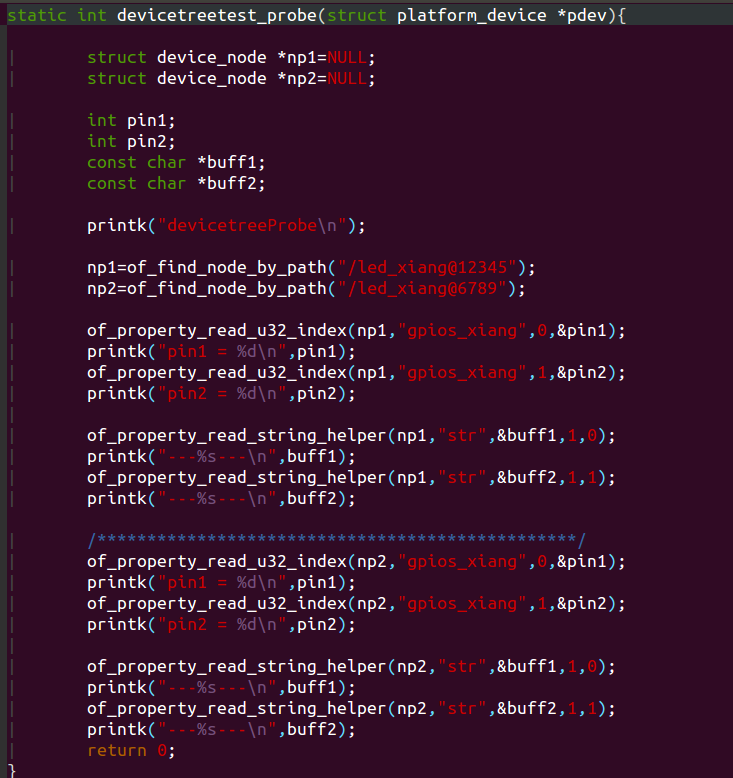
像这种两个节点名字一样，compatible一样的设备树，用@符号将其区分



这样@后面的数字就是区分这个设备节点的序号

这样@后面的数字就是区分这个设备节点的序号

device\_node \*变量 = of\_find\_node\_by\_path(写入设备节点路径) //获取指定路径下的设备节点

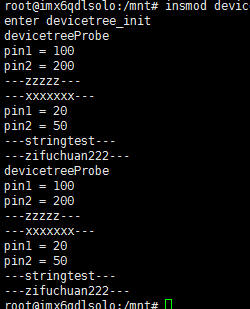
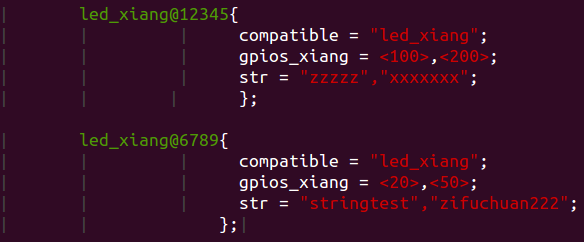


打印led\_xiang@6789设备节点下面的属性内容

打印led\_xiang@12345节点下面属性的内容

np2也和np1一样，只是设备节点不同

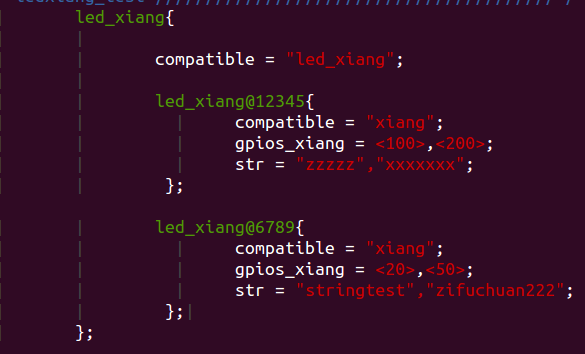
np1获取序列号为led\_xiang@12345下面的设备节点，这里有”/”符号，是因为我的这个设备节点是写在设备树/目录下的

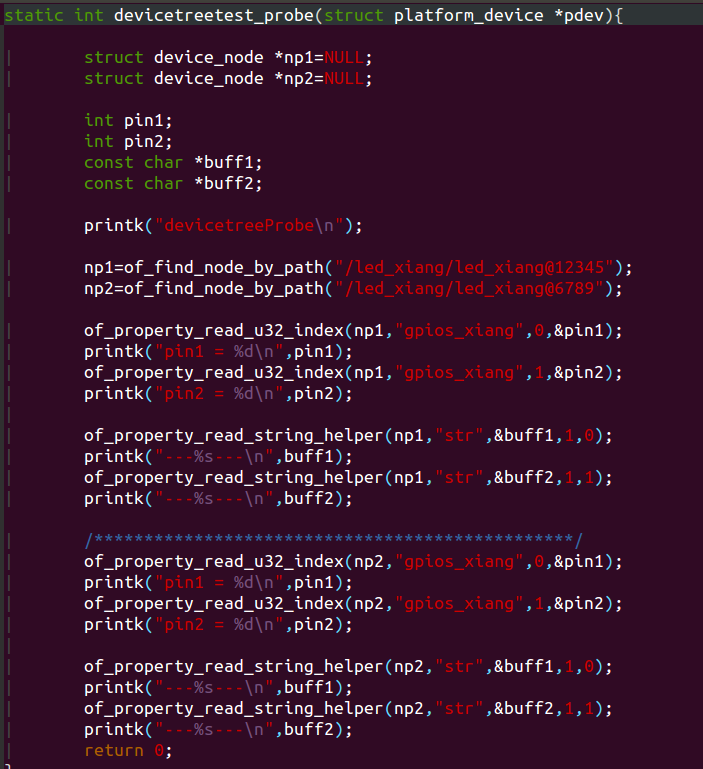
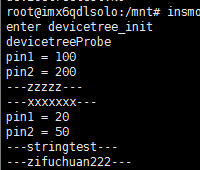
但是为什么执行了两次probe

输出打印正常

这是因为这里写了两次compatible，导致驱动识别了两次



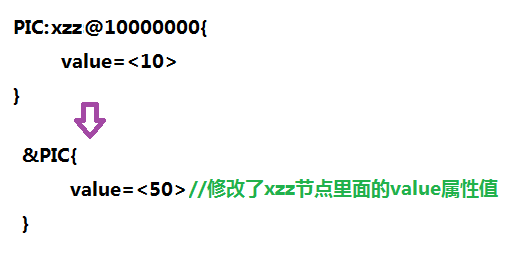
我们可以给两个@带编号的节点加一个父节点，这样我驱动程序compatible去匹配父节点

这样就不会因为子节点compatible字符一样重复执行两次probe

然后用of\_find\_node\_by\_path寻找绝对路径下的节点

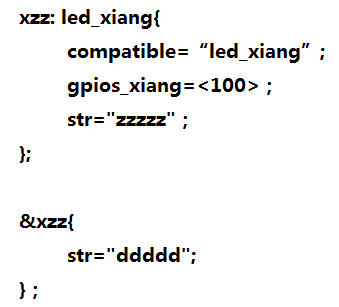
**节点之间的引用**



在另外一个dts文件里面只要包含了该dtsi文件，就可以用&节点名，使用该xzz节点，而且还可以在当前dts文件里修改dtsi文件里面xzz节点的属性

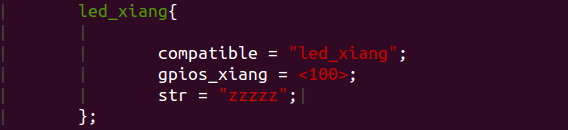
假如我们在dtsi里面定义个节点，给节点取了个别名PIC

这样做有什么好处

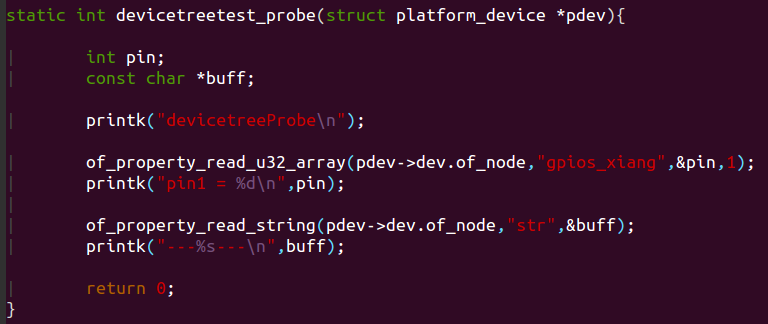
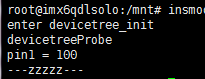


现在我想修改一个属性值，这样适合我的客户需求，但是我又不能把原来的其它属性修改到了，这时候你就可以用&获取你要修改节点属性的别名，比如led\_xiang节点的别名是xzz，你就获取&xzz，然后修改str里面的值，这样led\_xiang里面的str值就被修改了，其它属性还是保持不变

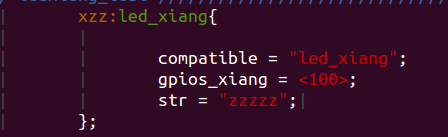
比如说我定义了gpios\_xiang和str属性，然后驱动程序是根据compatible去找到这个设备文件，然后获取这些属性



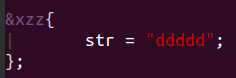
这是先写好的节点属性，和上面的操作方法一样

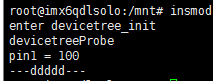
直接在驱动程序解析没有问题



我给节点加了个别名

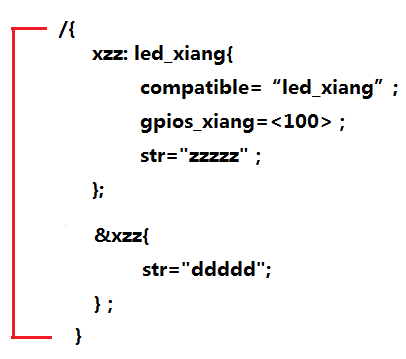


在某个时候我想修改下str的值，然后又不会让原节点的其它值不发生变化，那我就用&别名引用

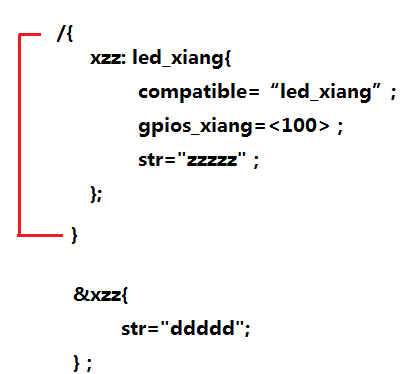


你看str值被修改了，但是gpios\_xiang的属性值还是保持不变。

这就是别名引用的好处，下面注意调用别名的语法规则



我在跟节点下面调用跟节点里面的别名是不对的，会编译报错



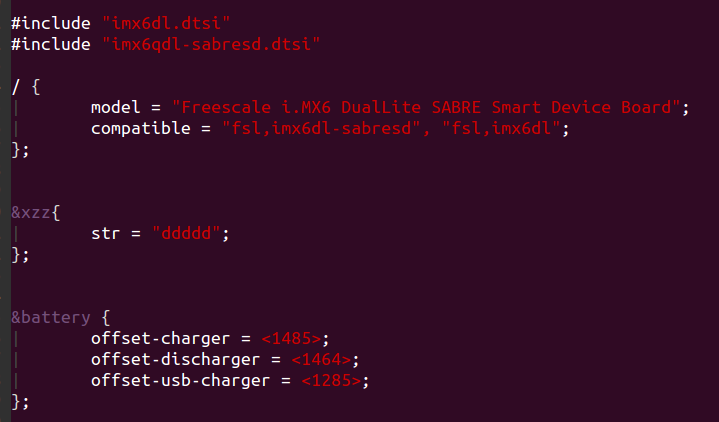
别名在跟节点外调用才是正确的，编译设备树通过

既然别名能在跟节点外使用，是不是其他dts文件也可以使用呢？

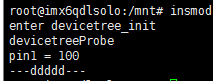
别名的初始定义在dtsi文件 现在修改dts文件来使用别名

打开dts文件



那么当前的dts文件就可以调用dtsi文件里面的别名节点，从而修改里面的属性

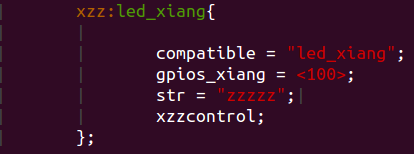
只要包含了我写好别名节点的dtsi文件



执行没有问题

**Pinctrl用法**

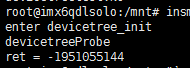
先介绍of\_find\_property(设备节点,要查找的属性名称，默认填写NULL)函数使用

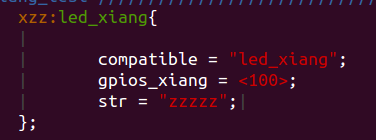
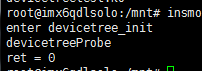


我在设备树增加了一个属性，但是这个属性没有赋任何值，这种格式，驱动可以用来判断这个属性在不在，从而决定执不执行相关代码



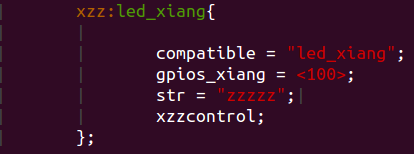
设备树的纯属性信息获取就要用of\_find\_property函数，这里填写要确认哪一个属性是否存在设备树

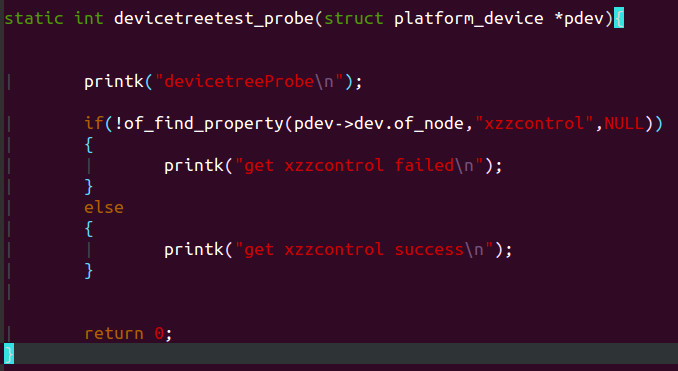
如果设备树存在驱动要找的属性值，那么of\_find\_property返回负值

如果xzzcontrol属性不存在，of\_find\_property返回0

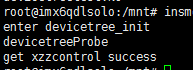
一般情况下of\_find\_property函数的返回直接用if判断

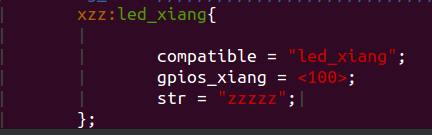


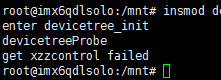


如果设备树有xzzcontrol属性，of\_find返回负数，取反之后还是负数，所以执行else

一般设备树没有xzzcontrol属性，of\_find返回0，取反之后返回1执行failed

这就表示设备树里面有xzzcontrol属性

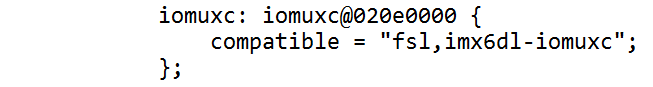




这就表示设备树里面没有xzzcontrol属性

1.GPIO初始化

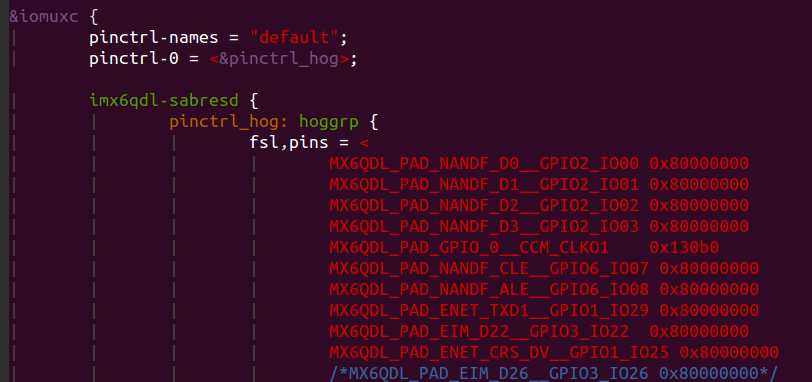
内核最先在imx6dl.dtsi只定义了iomux gpio初始化驱动程序的名字





然后在imx6qdl-sarebsd里面调用了imx6dl.dtsi的iomuxc节点，从而给iomuxc节点增加了新的节点epdc节点

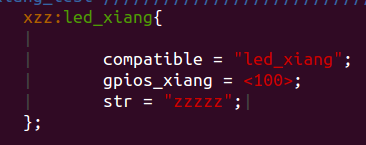
然后其余地方我也想自定义GPIO初始化状态，所以调用iomuxc再次增加了GPIO管脚节点



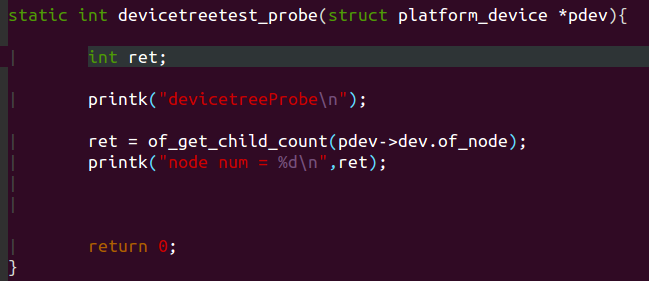
到现在，iomuxc有了子节点edpc和子节点imxpqdl-sabresd

驱动程序如何去计算子节点数量然后去自动配置设备树这些GPIO值呢？

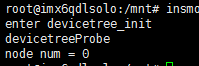
我们先了解下 of\_get\_child\_count(主设备节点) 函数使用

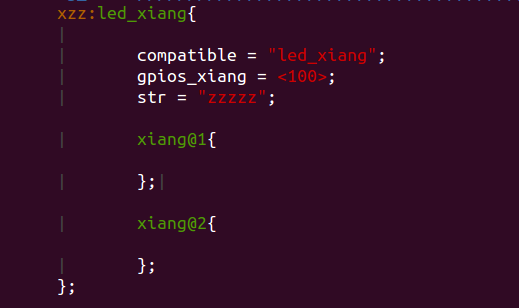


自定义主节点下面没有其他子节点

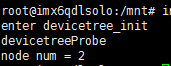


返回值是int类型，执行of\_get\_child\_count之后没有查找到子节点数量

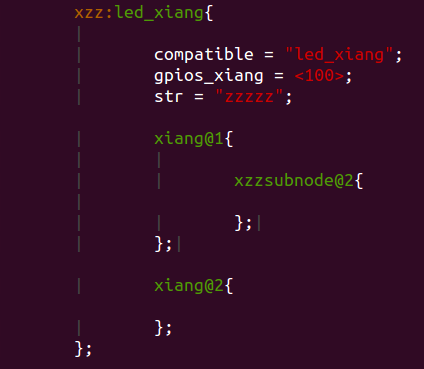




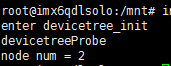
自定义主节点下，建立两个子节点

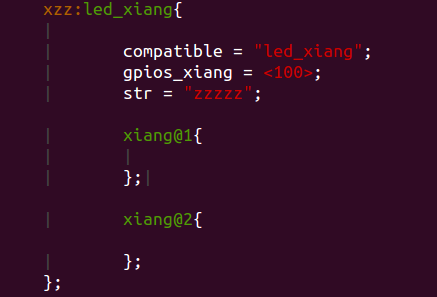
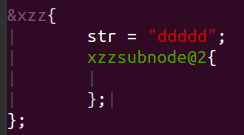
查找到了子节点数量2个

如果我再在子节点下面建立子节点呢？

编译能正常通过

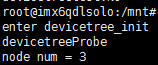
像这样，子节点下添加子节点

这种子节点下的子节点是不会计算进去的，of\_get\_child\_count只计算compatible下面的子节点

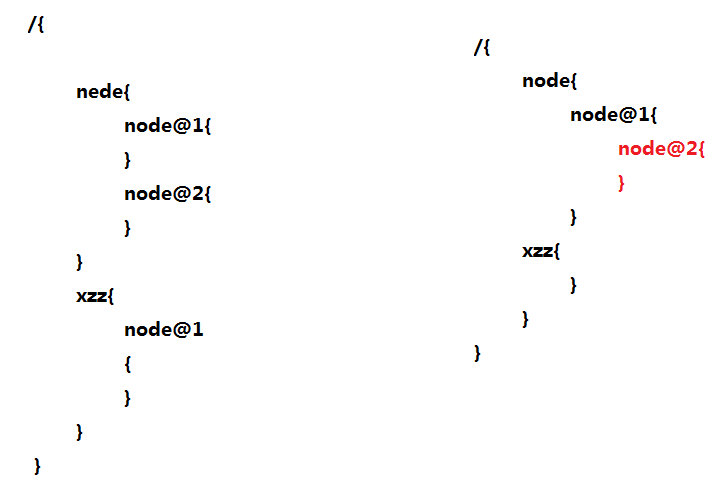
 

调用xzz节点里面添加子节点

dtsi文件里面有了两个子节点 调用xzz节点的dts文件又添加了个子节点

调用节点里面的子节点of\_get\_child\_count也能计算进去

在根节点下可以添加一个子节点，在子节点下可以再添加一个子节点

所以设备树最多两个子节点

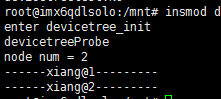
如果是在子节点下，最多只能添加一个子节点，如果子节点下添加子节点的子节点就不行

用for\_each\_child\_of\_node(父节点，要读取的子节点) 宏 循环读取每个子节点

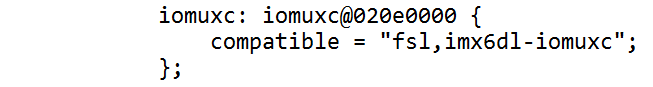
 

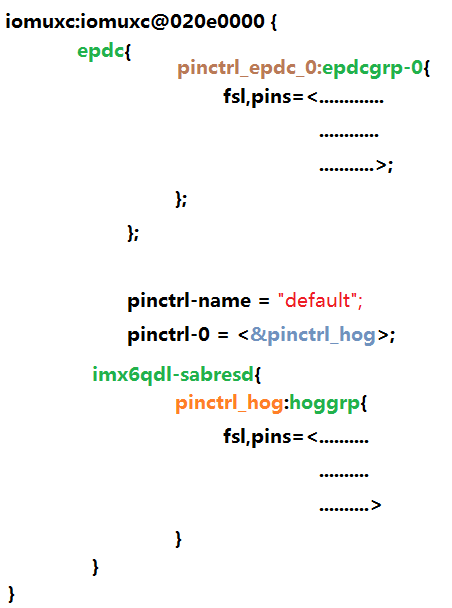
for\_each\_child\_of\_node功能是你把父节点传进去pdev->dev.of\_node

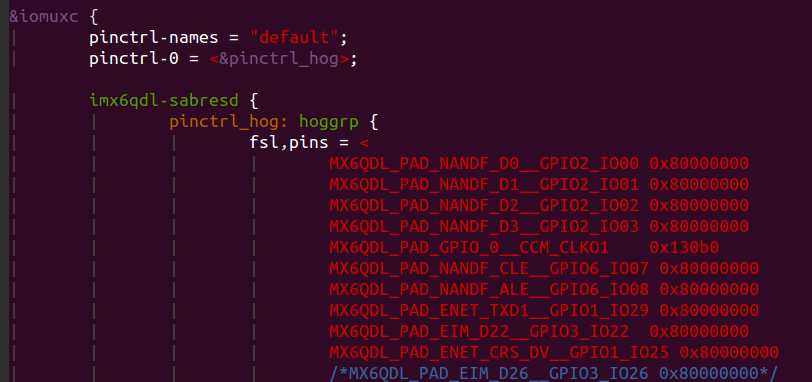
该函数会自动循环寻找父节点下面有多少个子节点，有多少个子节点就循环几次，记住每次循环的子节点你都要记得用device\_node变量去获取到

输出结果和我设备树两个子节点的str一致，没有问题

现在回过头来我们再来看iomuxc驱动程序如何实现的

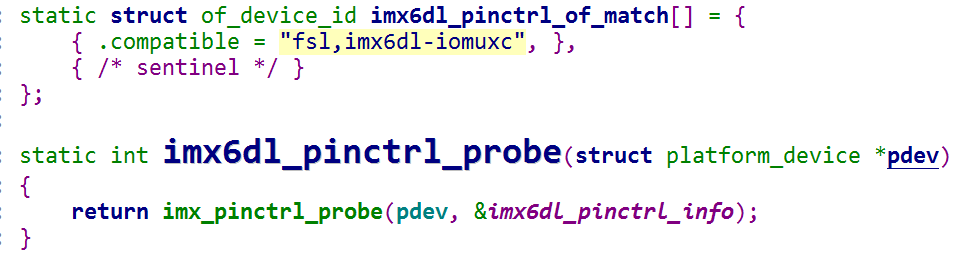




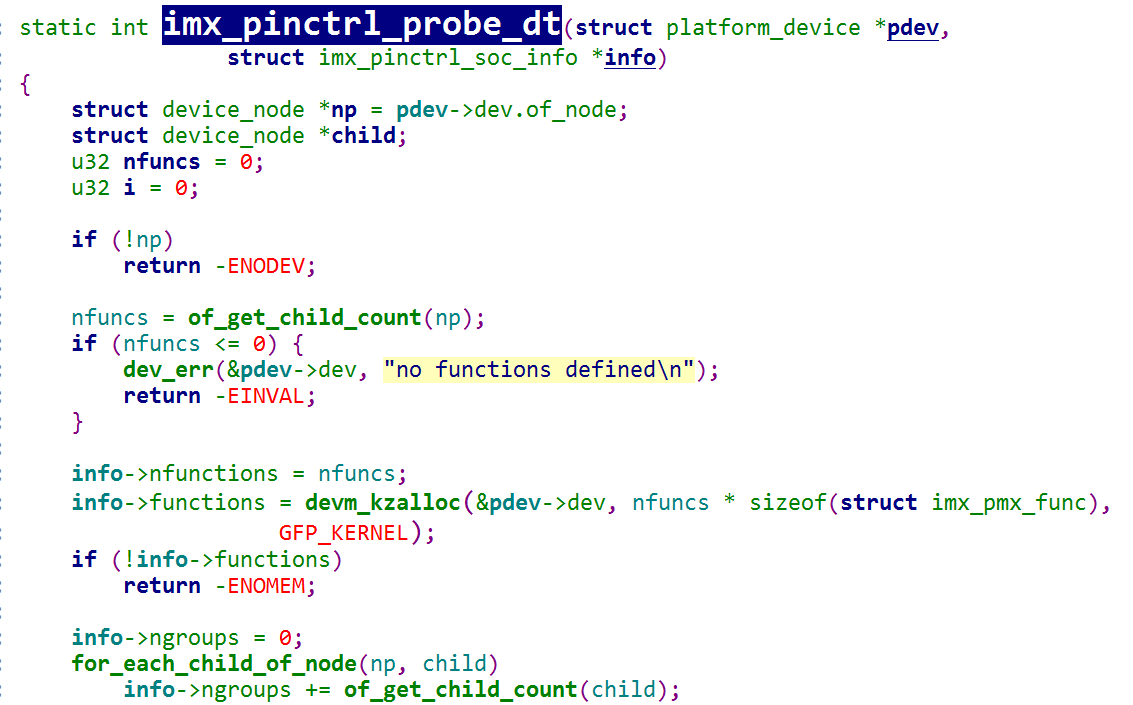


左边设备树展开就是右边这种形式，其实都是写在一个设备节点里面的。

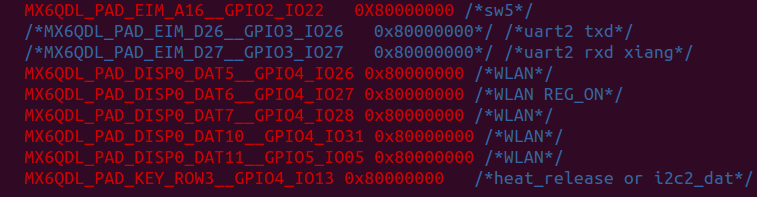
下面看看内核驱动文件是怎么去识别的







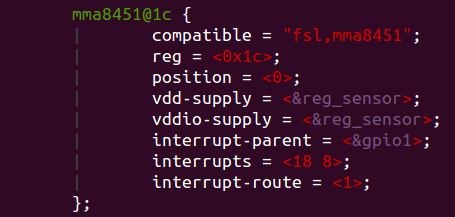
在这里将iomux设备节点的数据获取出来，进行一个一个的GPIO管脚映射，所以这里的for\_each\_child\_of\_node函数的功能我上一页已经讲清楚了



所以我只要调用iomux在fsl,pins属性里面写入我要初始化的引脚，内核启动后都会自动计算初始化引脚的个数，帮我进行初始化。

**中断使用**

设备树中断格式

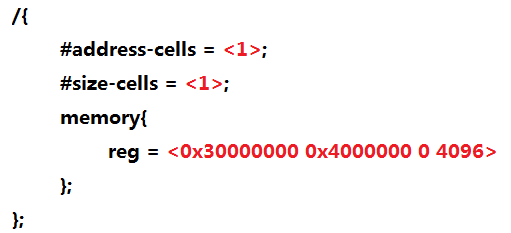


这个8就是该中断是上升沿触发还是下降沿触发还是电平触发还是双边沿触发

设备树标准中断写法是，先确定该IO口属于哪一个中断控制器，这里指定gpio1中断控制器

然后确定该中断控制器的中断号，这里是18，也是就是gpio1\_18引脚作为中断引脚

#adderss-cells，#size-cells关键字使用

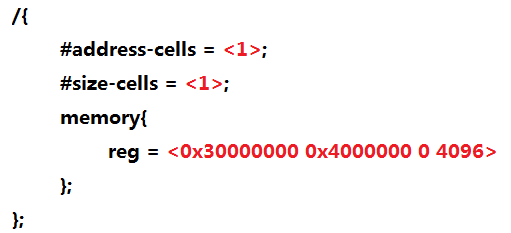


size-cell=<1>表示下面的reg属性<>第2个数据表示大小

Address-cell=<1>表示下面的reg属性<>里面第1个数据表示地址

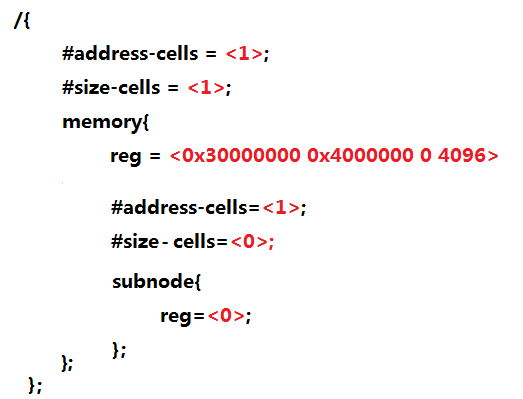
#address-cells和size-cells一定要写在所有子节点前面

下面总结reg属性



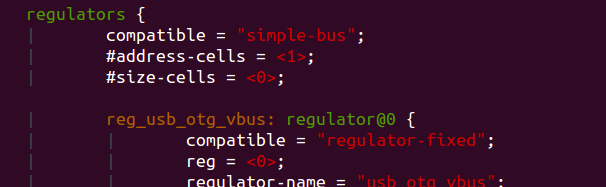
另一个地址和一个大小为第2组数据

所以address-cell=1和size-cells=1的缘故，表示reg是一个地址加一个大小值为一组数据



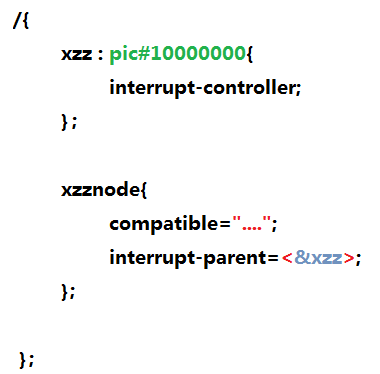
因为#size-cells=<0>，所以reg的每项数据都表示地址，不表示大小

在子节点下面也可以增加address-cell和size-cells，这种情况下就表示子节点下面的子节点reg属性范围



这就是个很明显的案例，看reg属性范围。

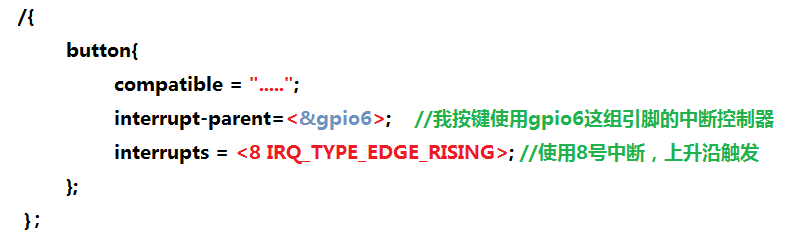
下面说说中断的设备树节点



然后我另外一个设备节点的GPIO或者外设使用到了pic#10000000这个中断控制器，那么我就要用interrupt-parent去引用它，所以&xzz

比如我在这个设备里面定义了一个中断控制器，写入关键字interrupt-controller就表示这个设备节点是一个中断控制器，用别名xzz好让其它设备节点调用

设备树设置按键中断的方法

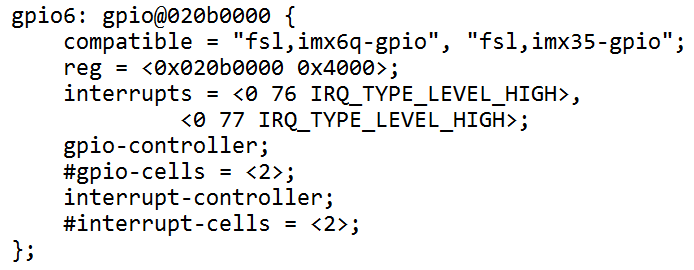


Interrupts = <IO口号，触发方式>

如果按键在GPIO6\_8引脚上，那么先设置GPIO6的中断控制器

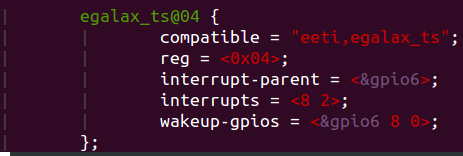
比如IMX6的中断控制器设备树规则

在最初总的dtsi文件定义了gpio6设备节点引用

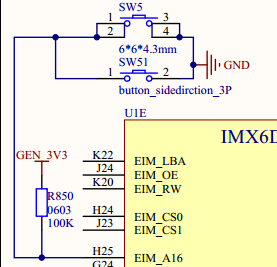


Interrupt-cells =2 表示引用gpio6的节点使用interrupts字符时用两个32位数据表示

而且gpio6这组IO控制器具备中断功能，所以这里指定了中断控制器字符

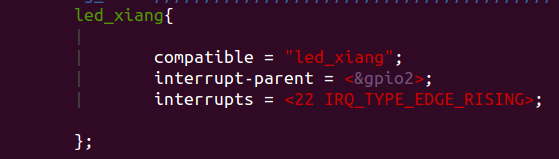


用IMX6按钮来举例外部中断使用





这里使用的是GPIO2\_22，确定了中断控制器是GPIO2，中断号是22



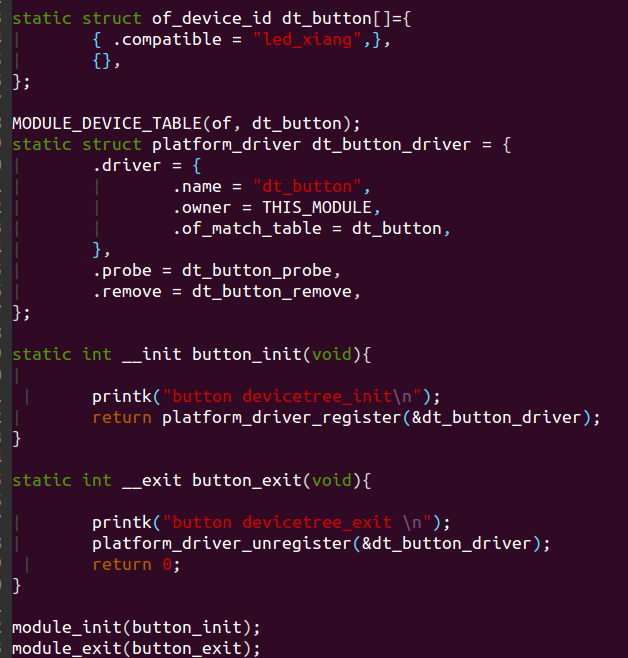
中断触发方式上升沿

按键接在GPIO2的22引脚上，所以中断号是22

设备树节点写入使用的中断控制器gpio2

下面介绍驱动层如何获取中断号

定义一大堆头文件



这是常规驱动匹配设备树代码，重点代码在下一页



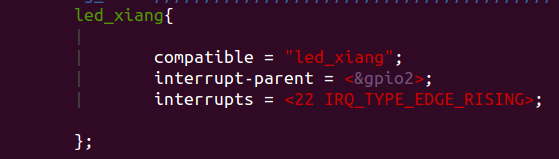
在设备树里面就设置了中断边沿触发方式，所以这里就填0就是了

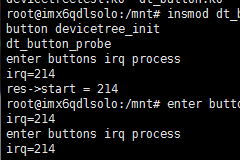
中断处理函数

res->start就是获取的中断号，可以用%d打印出来，这里就是申请这个中断号

这里填入0，就是获取interrupts第0个尖括号的中断号

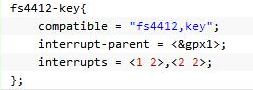
使用platform\_get\_resource获取设备节点，填入IORESOURCE\_IRQ获取中断部分内容



  
加载驱动程序之后，按下按钮中断程序已经执行。

这就是这个按键的外部中断号

如果是两个外部中断怎么做？我们用按键来模拟两个外部中断



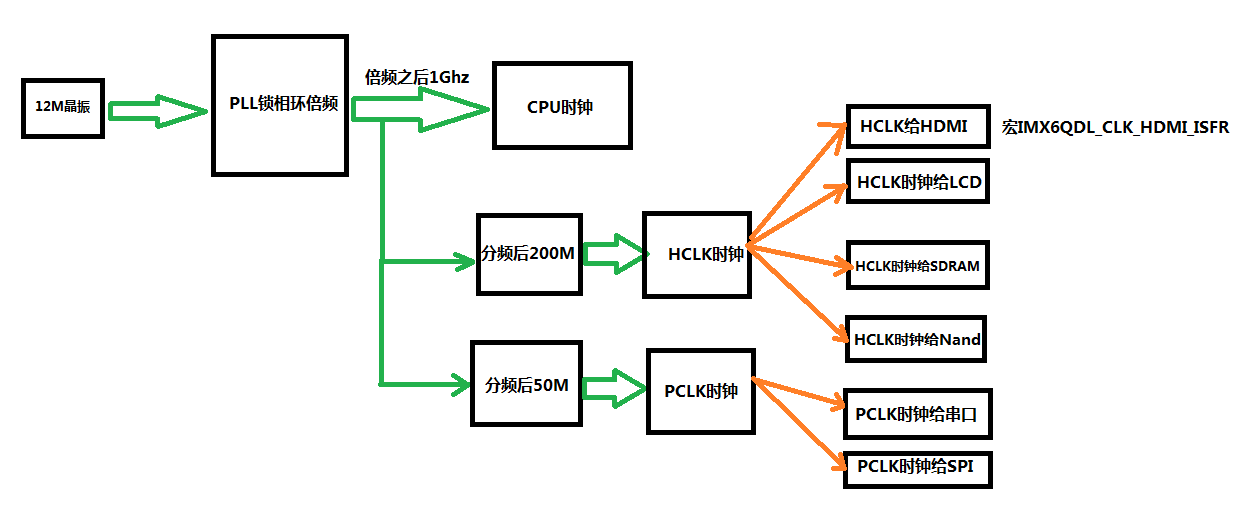
如果是两个外部中断，那么这两个外部中断必须在同一组GPIO里面，这样设备树才能用同一个中断控制器去映射，然后interrupts写两个尖括号，代表两个中断号



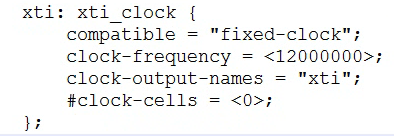
在驱动层就用0，1来区别获取设备树的两个中断号

**设备树时钟使用方法**

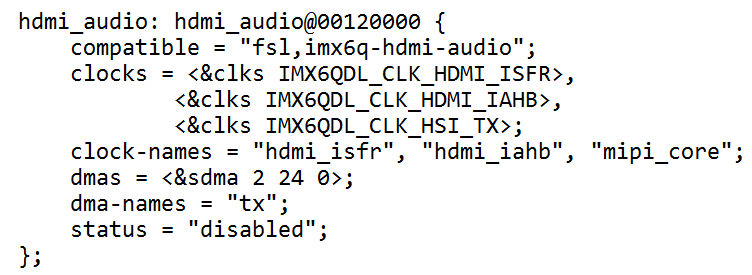
下面以4412设备树为例来讲解时钟

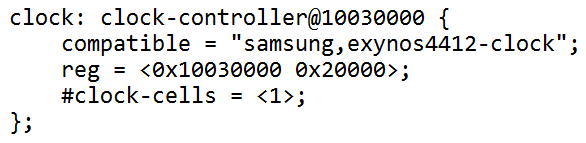


clock属性，里面传入&clk就表示需要操作时钟，后面的宏就表示要操作时钟的哪一部分，这里是操作hdmi时钟



这个设备节点是设置晶振和倍频的，有专门的驱动程序去匹配compatible执行

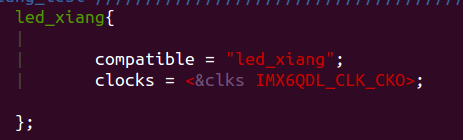




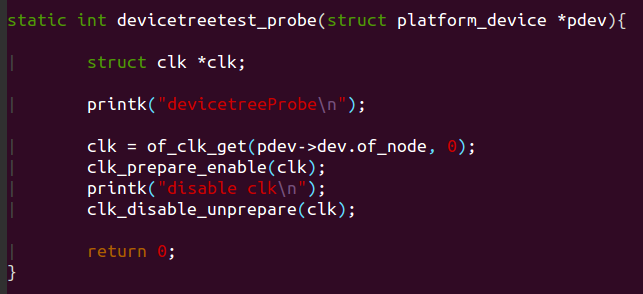
这个节点是设置时钟分频控制器的，有专门的驱动compatible去匹配

下面以IMX6为例来实际操作时钟案例

这里clock关键字属性就是要求写入&clk表示操作时钟，宏IMX6QDL\_CLK\_CKO表示我要操作摄像头时钟



然后驱动程序compatible去匹配，获取clocks属性里面的数据，这个&clks在其他dts文件已经定义好的引用节点

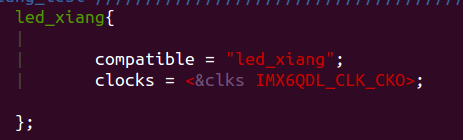


关闭摄像头时钟

填入clk是将IMX6QDL\_CLK\_CKO 摄像头时钟打开

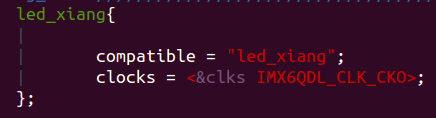
填0就是获取clocks第0项<>括号里面数据

of\_clk\_get就是获取设备节点的clock属性的值



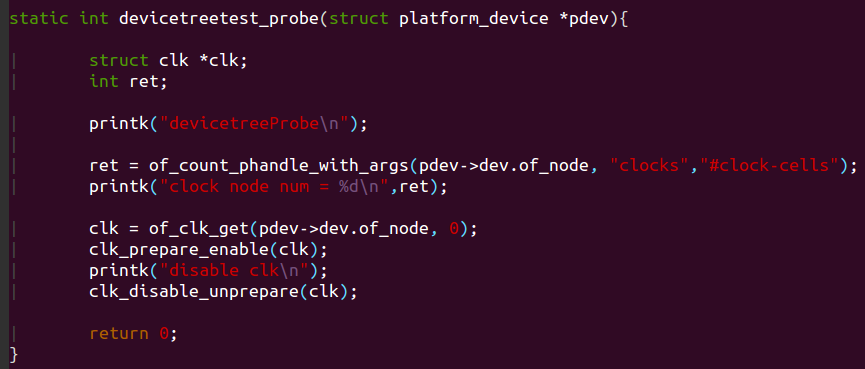
如果设备树时钟节点有多个时钟怎么操作？

设备节点只有1个时钟的操作方法，现在回顾下



这里只有一个时钟

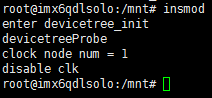
of\_count\_phandle\_with\_args(设备节点，属性，名称) //返回节点中时钟个数



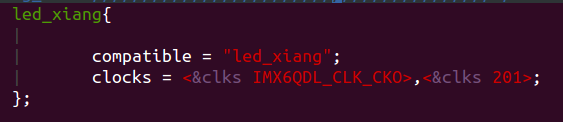
1个时钟就是编号0，所以这里获取填入0，就获取IMX6QDL\_CLK\_CKO时钟

返回1表示1个时钟

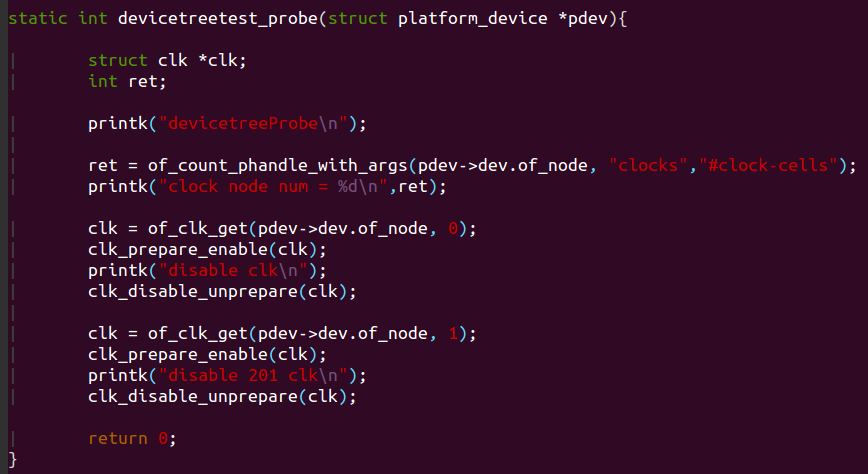
用of\_count\_phandle\_with\_args计算设备节点时钟个数



现在加入第2个时钟



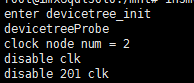
这里加入了clks时钟的IMX6QDL\_CLK\_CKO摄像头时钟和201时钟，这里就有两个时钟了



1表示获取201时钟，如果还有多个时钟就循环用of\_clk\_get 0 1 2 3…这样获取下去

0表示获取IMX6QDL\_CLK\_CKO摄像头时钟

返回2，表示获取到了两个时钟



这就是多个时钟的操作方法。

设备树使用的其它函数和零散的方法请参考我的github

[IMX6\_devicetree\_test](https://github.com/xxxzzzzzz/IMX6_devicetree_test)/[IMX6\_devicetree\_test](https://github.com/xxxzzzzzz/IMX6_devicetree_test/tree/master/IMX6_devicetree_test)/**imx6\_devicetree\_document**/文档