

第一步：我们在driver.c里面 从device.c 获取硬件平台获取资源，因为平台总线将驱动拆成了两个部分，第一部分是device.c，第二部分就是driver.c。 我们匹配成功之后driver.c就需要从device.c里面获得硬件资源。

那么实际上就是在probe函数里面实现获得硬件平台资源。

第二步：获得硬件平台资源后，我们就可以在probe函数里面注册一个杂项设备或者字符设备。注册完成之后，还要去完成file\_operation结构体。并且生成设备节点。

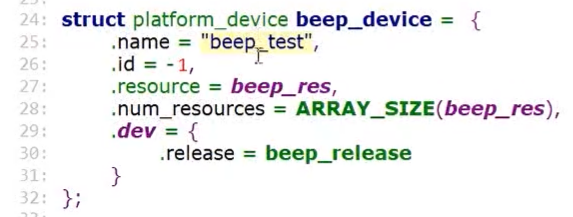
1. 从device.c获取硬件资源
   * 1. 直接获取：

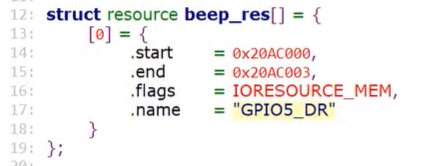
在driver.c里面的beep\_probe结构体里：

有一个platform\_device的结构体指针 的形参：pdev



pdev就指向了driver.c里面的 platform\_device beep\_device

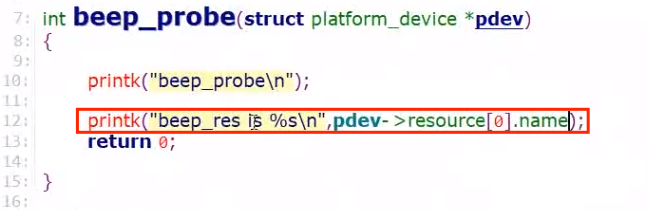




那么我们就可以直接通过pdev这个结构体指针来访问 beep\_device里面的结构体成员变量:

比如现在要访问resource(也就是beep\_res的内容)，如name:

在driver.c里面的实现：



有下面的实验结果可知，我们能够成功访问。所以我们同样能够访问start、end成员的内容。

但是这种方法是不安全的，不推荐使用。

* + 1. 使用函数platform\_get\_resource来获得相关的资源：

其返回值是struct resource

这个函数定义在include/linux/platform\_device.h里面：

IMG_256

参数：platform\_device 即pdev这个形参。 unsigned int资源的类型

Unsigned int 资源处在哪个位置上。

1. 注册杂项/字符设备，完善file\_operation结构体，并生成设备节点：

获得相关的之后，我们就可以直接来操作相关寄存器的地址了。我们就可以直接io\_remap像注册杂项设备那样操作寄存器，但是在操作之前，我们需要先在内核登记一下。

1.注册设备之前要登记：

内核登记的函数是request\_mem\_region(start,n,name)。

参数：

Start:起始地址

N：长度

Name:名字

返回值：

struct resource\*

位于include/linux/ioport.h文件里面：

IMG_256

注销登记的函数：

IMG_256

参数：

Start：起始地址

N:长度

我们之所以要登记，是因为我们要告诉内核，我们要用到这个寄存器，那么别的设备就不要再使用了，再申请的时候就会失败。同理，如果内核有别的驱动用到这个寄存器了。那么我们这里在申请的时候就会失败。

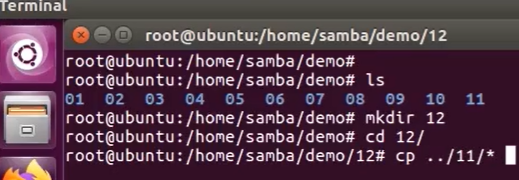
由于这里实验只用到DR寄存器，所以不登记也是没有问题的，但正规流程是需要登记的。

2.注册字符/杂项设备：

——————————————————————————————————

实操：

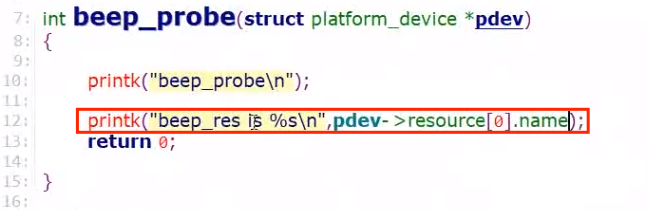
在11节实验上更改：



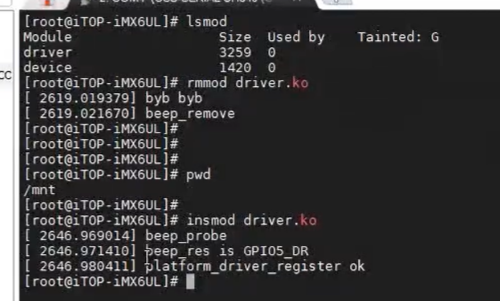
打开driver.c：

在driver.c里面的实现：

/\*直接获得硬件平台资源：



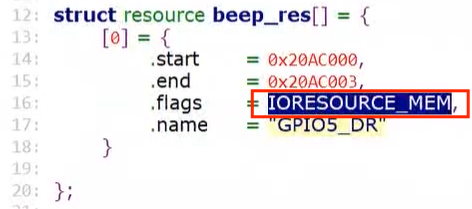
然后编译拷贝加载模块：



\*/

/\*

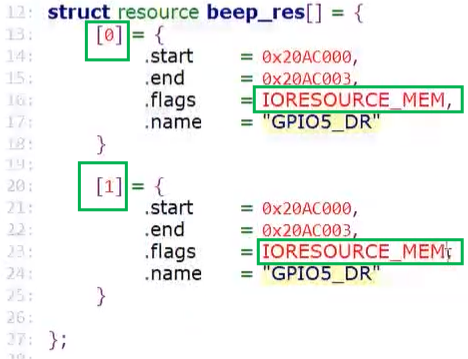
通过函数获得硬件平台资源：

资源类型：  
 

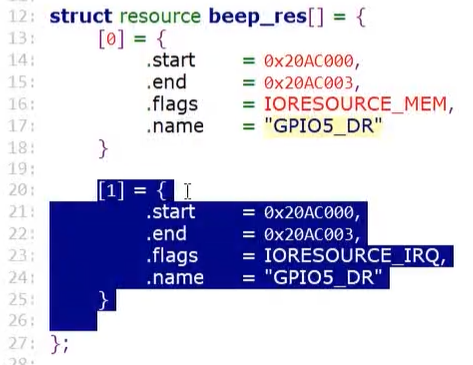
同类资源所处位置：

/\*

如果都是MEM类型，则资源的位置可以由0、1确定



如果类型不同，则IRQ资源所处的位置也是第0个。



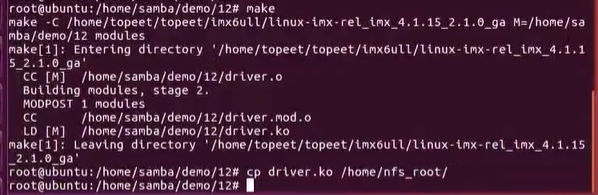
\*/

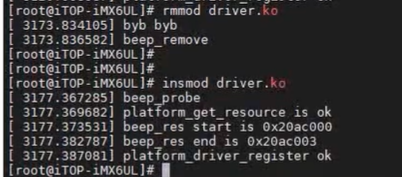
所以platform\_get\_resource函数体为:



\*/

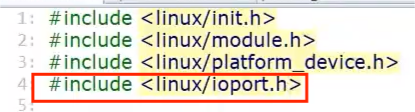
编译拷贝加载：





注册杂项/字符设备：

添加头文件：



定义用于接收 注册函数的 返回值的结构体指针变量：

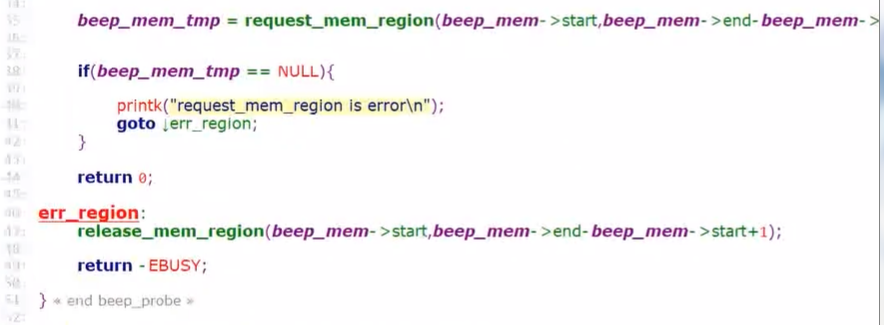
IMG_256

Init函数里面完成注册：

IMG_256

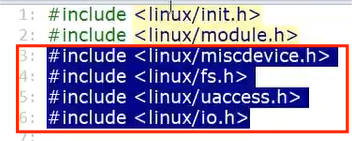
IMG_256

判断是否注册成功，如果没有成功就需要注销掉：



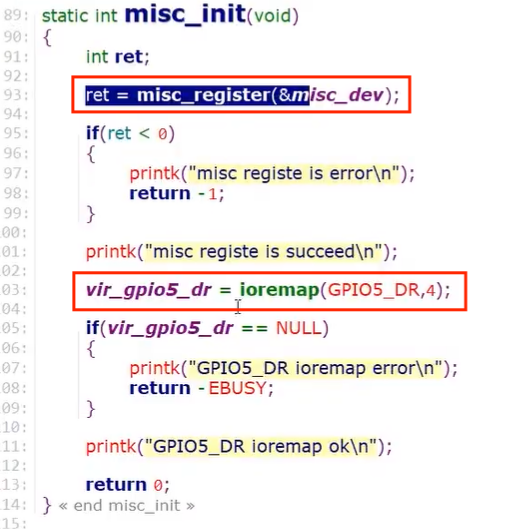
注册杂项设备,参考实验5的beep.c

包含头文件：



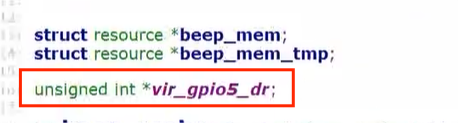
在beep.c里面：



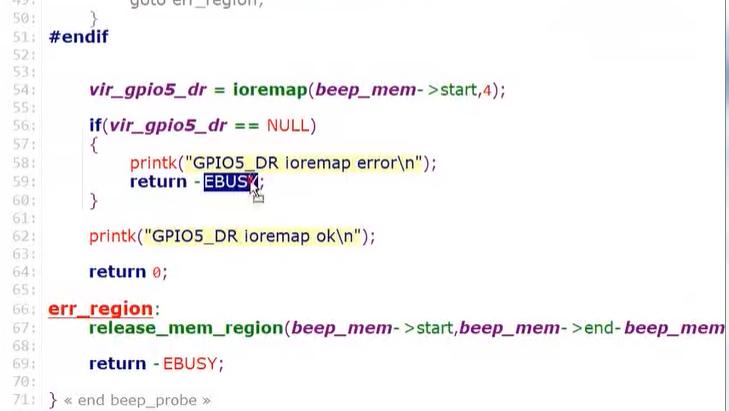


在driver.c中：

定义变量 接收ioremap的返回值：

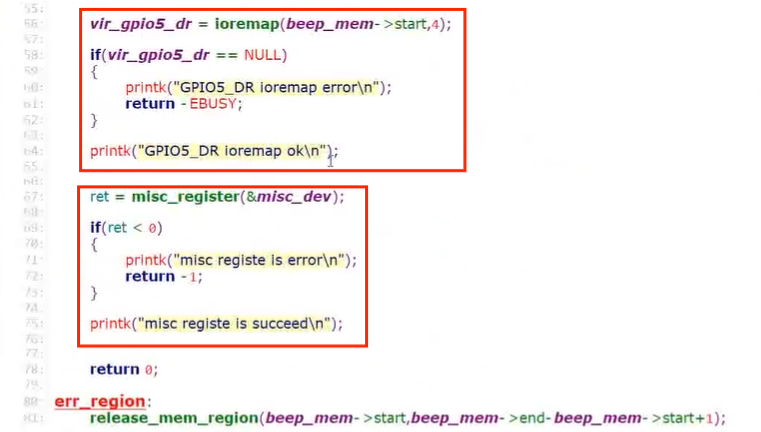


在probe函数里面先实现 IO地址空间 到 内核虚拟地址空间 的地址映射：



再实现注册：

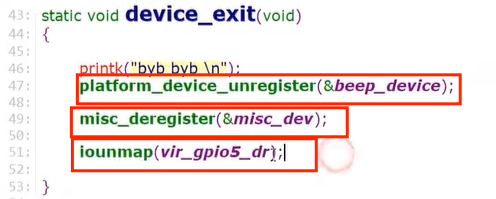




再将misc\_open 、read、release、write 、file\_operations、struct miscdevice等拷贝过来。

最后完成注销：

杂项设备的注销。Iounmap的注销：



编译拷贝加载：

加载成功后，它会进入probe函数

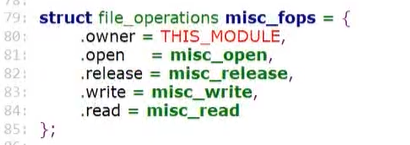
然后获取硬件平台资源

我们将获取到的DR的地址使用ioremap内存映射到内核虚拟地址。

再使用misc注册杂项设备。

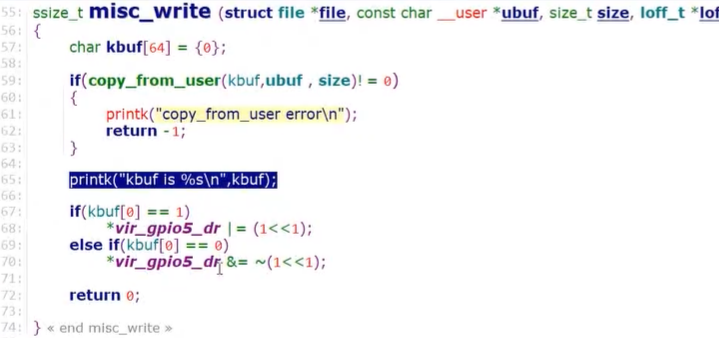
注册杂项设备的时候完善了结构体：misc\_open、write、read等。

由于我们在struct miscdevice misc\_dev结构体里面使用宏MISC\_DYNAMIC\_MINOR动态分配次设备号minor。他就会生成hello\_misc的设备节点：



那么在应用的时候，我们打开节点就会打印hello misc\_open

Write的时候，就会完成蜂鸣器的逻辑控制功能。



加载模块后：

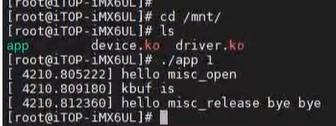
在/dev/下生成了对应的 /dev/hello\_misc节点。说明这一步是没有问题的。

编写APP来验证：

App直接使用第5次实验的即可：



编译执行：



课程：

<https://www.bilibili.com/video/BV1Vy4y1B7ta?spm_id_from=333.788.player.switch&vd_source=af21ee908c04895a95a22f7a0c9e0013&p=22>