http://www.wowotech.net/device_model/429.html

致驱动工程师的一封信

作者：smcdef 发布于：2018-4-14 21:00 分类：统一设备模型

引言

作为一个算是合格的驱动工程师，总是有很多话想说。代码看的多了总是有些小感悟。可能是吧。那就总结一下自己看的代码的一些感悟和技巧。如何利用你看的这些代码？如何体现在工作的调试中。作为驱动工程师，主要的工作就是移植各种驱动，接触各种硬件。接触最多的就是dts、中断、gpio、sysfs、proc fs。如何利用sysfs、proc fs及内核提供的接口为我们降低调试难度，快速解决问题呢？

注：部分代码分析举例基于linux-4.15。

如何利用dts

首先我们关注的主要是两点，gpio和irq。其他的选择忽略。先展示一下我期望的gpio和irq的使用方法。dts如下。

```
1. device {
2.     rst-gpio = <&gpioc_ctl 10 OF_GPIO_ACTIVE_LOW>;
3.     irq-gpio = <&gpioc_ctl 11 0>;
4.     interrupts-extended = <&vic 11 IRQF_TRIGGER_RISING>;
5. };
```

对于以上的dts你应该再熟悉不过，当然这里不是教你如何使用dts，而是关注gpio和irq最后一个数字可以如何利用。例如rst-gpio的OF_GPIO_ACTIVE_LOW代表什么意思呢？可以理解为低有效。什么意思呢？举个例子，正常情况下，我们需要一个gpio口控制灯，我们认为灯打开就是active状态。对于一个程序员来说，我们可以封装一个函数，写1就是打开灯，写0就是关灯。但是对于硬件来说，变化的是gpio口的电平状态。如果gpio输出高电平灯亮，那么这就是高有效。如果硬件设计是gpio输出低电平灯亮，那么就是低有效。对于一个软件工程师来说，我们的期望是写1就是亮灯，写0就是关灯。我可不管硬件工程师是怎么设计的。我们可以认为dts是描述具体的硬件。因此对于驱动来说，硬件的这种变化，只需要修改dts即可。软件不用任何修改。软件可以如下实现。

```
1. int device_probe(struct platform_device *pdev)
2. {
3.     rst_gpio = of_get_named_gpio_flags(np, "rst-gpio", 0, &flags);
4.     if (flags & OF_GPIO_ACTIVE_LOW) {
5.         struct gpio_desc *desc;
```

```
6.
7.          desc = gpio_to_desc(rst_gpio);
8.          set_bit(FLAG_ACTIVE_LOW, &desc->flags);
9.      }
10.
11.     irq = of_irq_get(np, 0);
12.     trigger_type = irq_get_trigger_type(irq);
13.     request_threaded_irq(irq, NULL, irq_handler, trigger_type, "irq", NULL);
14. }
```

驱动按照以上代码实现的话，如果修改中断触发类型或者电平有效状态只需要修改dts即可。例如不同的IC复位电平是不一样的，有的IC是高电平复位，有的IC却是低电平复位。其实这就是一个电平有效状态的例子。

如何调试gpio

移植驱动阶段或者调试阶段的工程中，难免想知道当前gpio的电平状态。当然很easy。万用表戳上去不就行了。是啊！硬件工程师的思维。作为软件工程师自然是要软件的方法。下面介绍两个api接口。自己摸索使用吧。点到为止。

```
1. static inline int gpio_export(unsigned gpio, bool direction_may_change);
2. static inline int gpio_export_link(struct device *dev, const char *name,
3.                 unsigned gpio);
```

在你的driver中调用以上api后，编译下载。去/sys/class/gpio目录看看有什么发现。

如何调试irq

调试的时候也难免会确定硬件是否产生中断。我们该怎么办呢？也很easy。

```
1. cat /proc/interrupts
```

输出信息不想多余的介绍。看代码去。根据输出的irq num，假设是irq_num。请进入以下目录。看看下面都有什么文件。摸索这些文件可以为你调试带来哪些方便。

```
1. cd /proc/irq/irq_num
```

dts和sysfs有什么关联

曾经写过一篇dts解析的文章http://www.wowotech.net/device_model/dt-code-file-struct-parse.html。最后一节其实说了一个很有意思的东西。但是仅仅是寥寥结尾。并没有展开。因为他不关乎我写的文章的的主题。但是，他却和调试息息相关。

```
1. cd /sys/firmware/devicetree/base
```

该目录的信息就是整个dts。将整个dts的节点以及属性全部展现在sysfs中。他对我们的调试有什么用呢？Let me tell you。如果你的项目非常的复杂，例如一套代码兼容多种硬件配置。这些硬件配置的差异信息主要是依靠dts进行区分。当编译kernel的时候，你发现你很难确定就是你用的是哪个dts。可能你就会凭借自己多年的工作经验去猜测。是的，你的经验很厉害。但是，如何确定你的dts信息是否添加到kernel使用的dts呢？我觉得你应该got it。就到这个目录去查找是否包含你添加的ndoe和property。dts中有status属性可以设置某个devicec是否使能。当你发现你的driver的probe没有执行的时候，我觉得你就需要确定一遍status是不是“ok”、“okay”或者没有这个属性。
远不止我所说的这个功能，还可以判断当前的硬件是匹配哪个dts文件。你还不去探索一波源码的设计与实现吗？节点为什么出现在某些目录？为什么有些节点的属性cat却是乱码？就是乱码，你没看错。至于为什么。点到为止。

sysfs可以看出什么猫腻

sysfs有什么用？sysfs可以看出device是否注册成功、存在哪些device、driver是否注册、device和driver是都匹配、device匹配的driver是哪个等等。先说第一项技能。deivce是否注册。
就以i2c设备为例说明。/sys/bus/i2c/devices该目录下面全是i2c总线下面的devices。如何确定自己的device是否注册呢？首选你需要确定自己的device挂接的总线是哪个i2c（i2c0, i2c1...）。假设设备挂接i2c3，从地址假设0x55。那么devices目录只需要查看是否有3-0055目录即可。如何确定device是否匹配了驱动？进入3-0055目录，其实你可以看到driver的符号链接。如果没有，那么就是没有driver。driver是否注册如何确定呢？方法类似。/sys/bus/i2c/drivers目录就是所有注册的i2c driver。方法如上。不再列举。
你以为就这些简单的功能了吗？其实不是，还有很多待你探讨。主要是各种目录之间的关系，device注册会出现什么目录？那么driver呢？各种符号链接会在那些目录？等等。

如何排查driver的probe没有执行问题

我想这类问题是移植过程中极容易出现的第一个拦路虎。这里需要声明一点。可能有些驱动工程师认为probe没有执行就是driver没有注册成功。其实这两个没有半毛钱关系。probe是否执行只和device和driver的是否匹配成功有关。我们有什么思路排查这类问题呢？主要排查方法可以如下。

1. 通过sysfs确定对应的device是否注册成功。
2. 通过sysfs确定对应的driver是否注册成功。
3. 通过sysfs中dts的展开信息得到compatible属性的值，然后和driver的compatible进行对比。

如果发现device没有注册，如何确定问题。首先通过sysfs中dts展开文件查看是否有你添加的device信息。如果没有的话，没有device注册就是正常的，去找到正确的dts添加正确的device信息。如果发现sysfs有device的dts信息。那么就看看status属性的值是不是ok的。如果也是ok的。那么就非常奇怪了。这种情况一般出现在kernel的device注册路径出错了。曾经就有一个小伙伴提出问题，现象就是这样。最后我帮他确定问题原因是dts章reg属性的地址是0xc0。这是一个大于0x7f的值。在i2c总线注册驱动的时候会解析当前总线下的所有device。然后注册所有的从设备。就是这里的地址检查出了问题，因此这个device没有注册上。

如果发现driver没有注册，那么去看看对应的Makefile是否参与编译。如果参与了编译，就查看log信息，是不是驱动注册的时候有错误信息。

最后一点的compatible属性的值只需要cat一下，然后compare即可。不多说。

后记

sysfs还有很多的其他的调试信息可以查看。因此，我建议驱动工程师都应该掌握sysfs的使用和原理。看看代码实现。我始终坚信只有更好地掌握技术的原理才能更好地利用技术。文章内容不想展开。我告诉你的结果，你永远记忆不深刻，而且我说的也不一定对。还是需要自己专研。我只是给你指条明路，剩下的就需要自己去走。最后说一句，代码不会骗你，还会告诉你别人不能告诉你的。