Tech-fantastic

Technology will make this world fantastic!

用Device tree overlay掌控Beaglebone Black的硬件资源

November 15, 2013 Beaglebone Black Beaglebone Black, device tree, hardware

经过一晚上的Google，终于大致明白device tree是怎么用的了，这里简单梳理一下思路。

一、简介

==========

device tree是ARM linux 3.7开始使用的系统控制硬件资源的方式，这里说的硬件资源既包括片上的诸如GPIO、PWM、I2C、ADC等资源，也包括外部拓展的如FLASH、LCD等。ARM使用device tree目前还是很新的东西(2012年低诞生的)，自带3.8版本Linux系统的Beaglebone Black是第一批使用device tree的ARM设备之一。难怪目前关于它的中文资料还很少。之前的linux如果要配置硬件，需要重新编译内核，但用device tree就不必重新编译内核，甚至不必重启系统就能实现。如此方便的工具让我们来看一看庐山面目！

从单片机、STM32过渡到Cortex-A8，之前直接操作寄存器来控制硬件的思路已经不好使了，但是device tree提供了一种很类似直接操作寄存器，但是比它更有条理，更容易理解的方式。首先需要编写一个.dts文件（device tree source），在文件中说明我要设置的硬件和它的各种属性，然后使用dtc命令编译这个.dts文件生成对应的二进制文件.dtb（device tree blob），系统启动时就会加载这个device tree并配置各种硬件资源。实际上Beaglebone Black自带系统中`/boot/`目录下已经包含了一些编译好的.dtb文件，从文件名来看似乎每个.dtb文件都能配置一款beagleboard.org的开发板，其中有一个叫做`am335x-boneblack.dtb`的文件，没猜错的话应当负责了Beaglebone black的缺省硬件配置。但因为已经编译成了二进制文件，所以我们无法读取其内容。

那么我们如果想要自己修改某些功能改怎么办呢？我们肯定不能重新编译一个`am335x-boneblack.dtb`代替原来的文件，那样会疯掉的。不过我们可以使用device tree overlay来动态重定义某些功能。device tree overlay与device tree类似，同样是编写一个.dts文件，编译成.dtbo文件(末尾的o应该代表overlay)。不同的是我们不把它放到`/boot/`目录中去，它也不必在启动时加载，而可以在需要时随时进行动态加载。另外device tree overlay的.dts文件跟device tree的.dts文件格式还是有一点区别的，下面要介绍的是device tree overlay的.dts。接下来我们上机操作一下。

二、编写.dts文件

==========

用ssh连接好Beaglebone black以后，我们先来找找Angstrom系统自带的.dts文件，看看它们长什么样子。用下面的命令搜索一下dts结尾的文件

```
# find / -name *dts
```

会得到下面这个列表(部分省略)

```
/lib/firmware/cape-bone-dvi-00A0.dts
/lib/firmware/bone_pwm_P8_45-00A0.dts
/lib/firmware/BB-SPI1A1-00A0.dts
/lib/firmware/BB-ADC-00A0.dts
/lib/firmware/BB-I2C1A1-00A0.dts
```

```
/lib/firmware/BB-BONE-SERL-01-00A1.dts
/lib/firmware/cape-bone-dvi-00A2.dts
/lib/firmware/bone_pwm_P8_13-00A0.dts
/lib/firmware/cape-bone-hexy-00A0.dts
/lib/firmware/BB-BONE-LCD7-01-00A2.dts
...
```

我们发现它们都在同一个目录内，`/lib/firmware/`，事实上系统自带的dts文件确实全部都在这个目录中，从文件名上我们会发现这里几乎包含了所有Beaglebone硬件资源的overlay，也包含了一些官方硬件外设(如lcd屏等，它们管自己的外设叫做cape)的overlay，因此以后有需要就可以直接到这里找了。下面随便打开其中一个看看(BB-UART1-00A0.dts)

```
/*
 * Copyright (C) 2013 CircuitCo
 *
 * Virtual cape for UART1 on connector pins P9.24 P9.26
 *
 * This program is free software; you can redistribute it and/or m
 * it under the terms of the GNU General Public License version 2
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

        /* identification */
        part-number = "BB-UART1";
        version = "00A0";

        /* state the resources this cape uses */
        exclusive-use =
                /* the pin header uses */
                "P9.24",        /* uart1_txd */
                "P9.26",        /* uart1_rxd */
                /* the hardware ip uses */
                "uart1";

        fragment@0 {
                target = <&am33xx_pinmux>;
                __overlay__ {
                        bb_uart1_pins: pinmux_bb_uart1_pins {
                                pinctrl-single,pins = <
                                        0x184 0x20 /* P9.24 uart1_
                                        0x180 0x20 /* P9.26 uart1_
                                >;
                        };
                };
        };

        fragment@1 {
                target = <&uart2>;    /* really uart1 */
                __overlay__ {
                        status = "okay";
                        pinctrl-names = "default";
                        pinctrl-0 = <&bb_uart1_pins>;
                };
        };
};
```

它的语法跟c语言有点类似。我先从中抽掉不重要的内容，把它写成下面的伪代码

```
/ {
        fragment@0 {
                target = <&am33xx_pinmux>;
                __overlay__ {
                        bb_uart1_pins: pinmux_bb_uart1_pins {
                                pinctrl-single,pins = <
```

```
                                        0x184 0x20 /* P9.24 uart1_
                                        0x180 0x20 /* P9.26 uart1_
                                >;
                        };
                };
        };

        fragment@1 {
                target = <&uart2>;
                __overlay__ {
                        status = "okay";
                        pinctrl-names = "default";
                        pinctrl-0 = <&bb_uart1_pins>;
                };
        };
};
```

从这里就能看出.dts文件的结构了——是一个树形结构。第一行的/代表根，下面的fragment@0和fragment@1是其两个分支节点。每个fragment节点下面又各有一个__overlay__节点（这些节点的名字都是固定的）。每个fragment节点下面相邻的target说明这个节点要修改的对象，在__overlay__节点下面的内容阐明了要修改的属性。

具体来说，am33xx_pinmux可以定义芯片功能复用引脚的具体功能，它使用了pinctrl-single,pins这个驱动，其中第一项0x184代表要修改的引脚，第二项0x20代表要修改成哪个功能(pinctrl-single的具体用法见这里)。这里把P9.24和P9.26两个引脚定义成了uart1的TX和RX。uart2这个target则使能了uart1(这个uart2实际上对应的是硬件的uart1)。

如果把树形结构什么的都忽略掉，就会发现其实它实现了我之前用寄存器干的事：定义引脚功能，然后使能串口。

那么当我想自己写device tree的时候，一上来就会遇到一个问题：我怎么知道我想控制的对象target名字是什么？我怎么知道它有哪些属性？取值范围是什么？答案在linux官网上。(不过有一些在这里似乎找不到，回头解决)

了解了dts文件的基本框架，我们再把之前丢掉的细节拿回来说明一下。(这些细节有些是非常重要的，实际使用中一定不要随意丢掉！)
首先这两行说明了dts的版本号，声明了这个文件的内容是一个plugin

```
/dts-v1/;
/plugin/;
```

根节点下面的一行说明了它的适用平台，这个是必须要写的。

```
compatible = "ti,beaglebone", "ti,beaglebone-black";
```

接下来的部分说明了这个device tree overlay的名字和版本号(版本号似乎只能是00A0)

```
/* identification */
part-number = "BB-UART1";
version = "00A0";
```

再下面的部分说明了要使用的引脚和硬件设备

```
/* state the resources this cape uses */
        exclusive-use =
                /* the pin header uses */
                "P9.24",        /* uart1_txd */
```

```
                "P9.26",         /* uart1_rxd */
                /* the hardware ip uses */
                "uart1";
```

接下来就是device tree overlay的具体内容，前面已经简单解释过了，但似乎还是看不太明白，也写不出来。实际上我们并不需要自己从头开始写，因为在系统/lib/firmware/目录中已经自带了很多.dts文件，我们只需要在它们的基础上进行修改就行了。需要提示一点，在.dts文件里我们经常会看到target = &ocp，这里的ocp是on chip peripherals的缩写，我猜想可能是用来描述连接到芯片的其他外设的(如按键、lcd等)。后面的日志里我再记录一下详细的操作细节。这篇先简单介绍这么多。

三、编译.dts文件

==========

写好.dts文件以后需要用dtc编译器编译一下，生成.dtbo文件才能使用。
假设我们写好了一个名为ADAFRUIT-SPI0-00A0.dts的文件，编译指令如下

```
# dtc -O dtb -o ADAFRUIT-SPI0-00A0.dtbo -b 0 -@ ADAFRUIT-SPI0-00A0.dts
```

然后就会生成ADAFRUIT-SPI0-00A0.dtbo文件。下面解释一下各个参数
-O dtb 声明输出格式为dtb文件
-o 输出文件名
-b 设置启动CPU
-@ (我不太清楚这项是干嘛的，似乎是overlay专有的一项)
注意文件的命名，一定是“程序名-版本号.dtbo(.dts)”的形式。
编译完成以后，一定要把.dtbo文件放到/lib/firmware/目录下才能使用

```
# cp ADAFRUIT-SPI0-00A0.dtbo /lib/firmware
```

四、overlay的使用 (Exporting and Unexporting an Overlay，加载和卸载)

==========

所有已经加载的overlay列表都在/sys/devices/bone_capemgr.*/slots这个文件中。
(bone_capemgr.*中的*号实际是一个数字，但是每次系统启动时这个数字可能会变化，所以我们用通配符*代替。)我们打开这个文件看一看

```
# cat /sys/devices/bone_capemgr.*/slots
```

```
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
```

我们看到系统已经自动加载了两个overlay，eMMC和HDMI。下面我们把之前讲解的BB-UART1-00A0.dtbo加载一下，方法是

```
# echo BB-UART1 > /sys/devices/bone_capemgr.*/slots
```

然后我们再打开slots文件看看有什么变化

```
 0: 54:PF---
```

```
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
6: ff:P-O-L Override Board Name,00A0,Override Manuf,BB-UART1
```

会发现多了一项，说明加载成功了，下面就可以使用外设了。

外设使用完毕以后，如何卸载呢？一种方法是重启系统，另一种是

```
# echo -6 > /sys/devices/bone_capemgr.*/slots
```

但是在最近的Angstrom系统中，用这种方法会导致kernel panic，然后ssh会断开，所以现在还是用重启系统的方法吧。相信今后这个问题应该很快会解决的。

注：adafruit的这篇Introduction to the BeagleBone Black Device Tree令我受益匪浅，在此表示一下感谢。

Be the first to like this.

Related

Q&A2 - 进一步理解和使用 device tree
In "Beaglebone Black"

Q&A1
In "Beaglebone Black"

使用BBB的SPI
In "Beaglebone Black"