Device Tree - 战斗机中的菜鸟

设备树笔记

参考资料：http://www.wowotech.net/linux_kenrel/why-dt.html

一、背景

设想一下：bootloader将Linux内核复制到内存中，然后跳到内核的入口点开始执行。此时内核就像运行在处理器上的一个裸机程序。需要配置处理器，设置虚拟内存，向控制台打印一些信息。但是这些事情如何完成？所有的这些操作都要通过写寄存器来实现，但Linux内核如何知道这些寄存器的地址？如何知道当前有多少个CPU核可以使用？有多少内存可以访问？最直接的办法就是在内核代码里为指定平台写好这些代码，由内核配置参数决定哪些平台代码将被启用。但每一块ARM芯片都有自己的寄存器地址和不同的配置方式以及外设,这导致内核充斥大量垃圾代码,所以人们希望内核能以某种方式识别硬件并加载驱动,于是设备树出现了,他用于指明系统所使用的设备及相应的配置信息。

随着越来越多的芯片厂商加入ARM阵营，各个ARM 供应商的SOC 家族的CPU越来越多，不同厂商的周边硬件设备又各不相同，加之芯片供应商开发人员为了更快的开发效率，使得很多SOC 特定的代码都是通过复制现有代码并稍作修改就提交到ARM Linux。这导致

1、越来越多的ARM 平台相关的代码被加入到Linux内核， #ifdef充斥在各个源代码中，让ARM mach-和plat-目录下的代码有些不忍直视。

2、系统充斥大量重复代码。

因此，内核社区成立了一个“ARM 子架构”的团队，该团队主要负责协调各个ARM厂商的代码（not ARM core part），检查各个子架构维护者提交的代码，并建立一个ARM 平台合并树来维护这些代码。针对不同的SOC共用的IP block（知识产权块，例如I2C controller），将其驱动代码从各个arch/arm/mach-xxx中独立出来，变成通用的模块移动到kernel/drivers目录。而如clock control、interrupt control等并不是ARM特殊部分，将其驱动放在linux/kernel目录下，属于core-Linux-kernel frameworks。此外，对于ARM平台，需要保存一些和framework交互的代码，这些代码叫做ARM SoC core architecture code。总结如下：

1、ARM的核心代码仍然保存在arch/arm目录下，ARM SoC core architecture code（与系统内核交互的代码）也保存在arch/arm目录下；

2、ARM SOC的周边外设模块（如I2C控制器）的驱动保存在drivers目录下，通用的设备（如中断控制器）驱动直接集成到系统内核中；

3、ARM SOC的专有代码在arch/arm/mach-xxx目录下；

4、ARM SOC board specific的代码被移除，由设备树机制来负责传递硬件拓扑和硬件资源信息。

本质上，设备树改变了原来用hardcode方式将硬件配置信息嵌入到内核代码的方法，改用bootloader传递一个DB的形式。对于基于ARM CPU的嵌入式系统，我们习惯于针对每一个平台进行内核的编译。但是我们期望ARM能够像X86那样用一个kernel image来支持多个平台。在这种情况下，如果我们认为内核是一个黑盒，那么其输入参数应该包括：识别平台的信息、runtime的配置参数、设备的拓扑结构以及特性。在linux kernel中，设备树就是为了把上述的三个参数信息通过bootloader传递给kernel，以便kernel可以有较大的灵活性。

为一个外设写一个设备树entry（http://blog.csdn.net/klaus_wei/article/details/42915545）：1、为"compatible"赋一个字符串"magicstring"，自动生成工具的生成格式一般是：名字+版本。2、在数据手册里查看总线上设备的地址分配信息， 写一条 "reg=" 语句。3、"interrupt-parent=<&gic>"4、中断号 "interrupt="5、最后加上一些设备的自定义参数Porting操作系统到硬件平台：1、自己撰写一个bootloader并传递适当的参数给kernel。除了传统的 command line以及tag list之类的，最重要的是申请一个machine type，当拿到属于自己项目的machine type ID的时候，当时心情雀跃，似乎自己已经是开源社区的一份子了（其实当时是有意愿，或者说有目标是想将大家的代码并入到linux kernel main line的）。2、在内核的arch/arm目录下建立mach-xxx目录，这个目录下，放入该SOC的相关代码，例如中断 controller的代码，时间相关的代 码，内存映射，睡眠相关的代码等等。此外，最重要的是建立一个board specific文件，定义一个machine的宏：

```
MACHINE_START(project name, "xxx公司的xxx硬件平台")
        .phys_io        = 0x40000000,
        .boot_params        = 0xa0000100,
        .io_pg_offst        = (io_p2v(0x40000000) >> 18) & 0xfffc,
        .map_io              = xxx_map_io,
        .init_irq       = xxx_init_irq,
        .timer               = &xxx_timer,
        .init_machine       = xxx_init,
MACHINE_END
```

在xxx_init函数中，一般会加入很多的platform device。因此，伴随这个board specific文件中是大量的静态table，描述了各种硬件设备信息。 3、调通了system level的driver（timer，中断处理，clock等）以及串口terminal之后，linux kernel基本是可以起来了，后续各种driver不断的添加，直到系统软件支持所有的硬件。

二、设备树

2.1 概念

如果要使用Device Tree，首先用户要了解自己的硬件配置和系统运行参数，并把这些信息组织成Device Tree source file。通过DTC（Device Tree Compiler），编译为Device Tree binary file（有一个更好听的名字，DTB，device tree blob）。系统启动时被加载到内存并将起始地址传给OS内核。

另外，设备树中不用描述所有硬件信息，对于可以动态探测到的设备不必在其中描述，如USB设备、PCI 设备，但usb host controller是无法动态识别的，PCI bridge如果不能被探测，则需要在device tree中描述。对于同一系列的SOC家族，通常将公共的硬件描述保存在一个单独的dtsi文件中，方便大家include共用，省去代码的重复。

2.2 设备树源文件

2.2.1 语法

device tree的基本单元是node。这些node被组织成树状结构，除了root node，每个node都只有一个parent。一个device tree文件中只能有一个root node。每个node中包含了若干的property/value来描述该node的一些特性。在linux kernel中，扩展名是dts的文件就是描述硬件信息的device tree source file，在dts文件中，一个node被定义成：

```
[label:] node-name[@unit-address] {
    [properties definitions]
    [child nodes]
}
```

说明：

[]表示可选项；

label方便在dts文件中引用；

每个node用节点名字（node name）标识，节点名字的格式是node-name@unit-address；@unit-address的格式和设备挂在哪个bus上相关，如cpu，其unit-address就是从0开始编址，如以太网控制器，其unit-address就是寄存器地址，如果该node没有reg属性（寻址需求属性），那么该节点名字中必须不能包括@和unit-address。root node的node name是确定的，必须是“/”。

属性（property）值标识了设备的特性，它的值（value）是多种多样的：

1、可能是空，也就是没有值的定义。

2、可能是一个u32、u64的数值（值得一提的是cell这个术语，在Device Tree表示32bit的信息单位）。例如#address-cells = <1> 。当然，可能是一个数组。例如<0x00000000 0x00000000 0x00000000 0x20000000>

3、可能是一个字符串。例如device_type = "memory" ，当然也可能是一个string list。例如"PowerPC,970"

child node的格式和node是完全一样的。

2.2.2 节点和属性

根节点/	节点	属性	属性说明	节点说明	
		#address-cells	#是数量的意思，#address-cells属性用来描述sub node中的reg属性的地址域特性，也就是说需要用多少个u32的cell来描述该地址域。	如果节点中包含了有寻址需求reg的子节点，则需要定义这两个属性，	
		#size-cells			
		Compatible 该属性的值是string list，定义了一系列的modle（每个string是一个model）。这些字符串列表被操	model属性指明了该设备属于哪个设备生产商的哪一个model。一般model赋值“生产商，模型（系列）” 。对于root node，compatible属性用来匹配machine type，对于普通的HW block的节		

	作系统用来选择用哪一个driver来驱动该设备。	点，如中断控制器，属性被用来匹配适合的driver。	
	interrupt-parent	用于标识能产生中断的设备连接到哪个中断控制器	
chosen { }	bootargs	传递命令行参数	描述由系统firmware指定的runtime parameter。如果存在chosen这个node，其parent node必须是根节点。
	initrd-start	传递initrd的开始地址	
aliases { }			定义了一些别名，方便引用节点时省写完整路径
memory { }	device_type	对于memory node，其device_type必须为memory。	是所有设备树文件的必备节点，它定义了系统物理内存的布局
	reg属性定义了访问该device node的地址信息，	该属性的值被解析成任意长度的（address，size）数组， address和size在其父节点中定义（#address-cells和#size-cells）。对于device node，reg描述了memory-mapped IO register的offset和length。对于memory node，定义了该memory的起始地址和长度。	
interrupt-controller @4a000000{}	#interrupt-cells	用多少个u32（即cells）来标识一个interrupt source	中断控制器节点，其中包含属性值。4a000000表示中断控制器寄存器的起始地址
Serial @50000000{}	interrupts	对于一个能产生中断的设备，必须定义interrupts这个属性。也可以定义interrupt-parent这个属性，如果不定义，则继承其parent node的interrupt-parent属性。	
	status		

	Cpus{}		对于cpus node，#address-cells 是1，而#size-cells是0。	对于根节点，必须有一个cpus的child node来描述系统中的CPU信息。	

2.3 设备树二进制文件

设备树二进制文件的组织格式如下：

说明：

1 DTB header其各个成员解释如下：

header field name	description
magic	用来识别DTB的。通过这个magic，kernel可以确定bootloader传递的参数block是一个DTB还是tag list。
totalsize	DTB的total size
off_dt_struct	device tree structure block的offset
off_dt_strings	device tree strings block的offset
off_mem_rsvmap	offset to memory reserve map。有些系统，我们也许会保留一些memory有特殊用途（例如DTB或者initrd image），或者在有些DSP+ARM的SOC platform上，有写memory被保留用于ARM和DSP进行信息交互。这些保留内存不会进入内存管理系统。
version	该DTB的版本。
last_comp_version	兼容版本信息
boot_cpuid_phys	我们在哪一个CPU（用ID标识）上booting
dt_strings_size	device tree strings block的size。和off_dt_strings一起确定了strings block在内存中的位置
dt_struct_size	device tree structure block的size。和和off_dt_struct一起确定了device tree structure block在内存中的位置

3、 memory reserve map的格式描述

这个区域包括了若干的reserve memory描述符。每个reserve memory描述符是由address和size组成。其中address和size都是用U64来描述。

4、device tree structure block的格式描述

device tree structure block区域是由若干的分片组成，每个分片开始位置都是保存了token，以此来描述该分片的属性和内容。共计有5种token：

（1）FDT_BEGIN_NODE (0x00000001)。该token描述了一个node的开始位置，紧挨着该token的就是node name（包括unit address）

（2）FDT_END_NODE (0x00000002)。该token描述了一个node的结束位置。

（3）FDT_PROP (0x00000003)。该token描述了一个property的开始位置，该token之后是两个u32的数据，分别是length和name offset。length表示该property value data的size。name offset表示该属性字符串在device tree strings block的偏移值。length和name offset之后就是长度为length具体的属性值数据。

（4）FDT_NOP (0x00000004)。

（5）FDT_END (0x00000009)。该token标识了一个DTB的结束位置。

一个可能的DTB的结构如下：

（1）若干个FDT_NOP（可选）

（2）FDT_BEGIN_NODE

node name

paddings

（3）若干属性定义。

（4）若干子节点定义。（被FDT_BEGIN_NODE和FDT_END_NODE包围）

（5）若干个FDT_NOP（可选）

（6）FDT_END_NODE

（7）FDT_END

5、device tree strings bloc的格式描述

device tree strings bloc定义了各个node中使用的属性的字符串表。由于很多属性会出现在多个node中，因此，所有的属性字符串组成了一个string block。这样可以压缩DTB的size。

2.4 设备树数据流

1、在ARM的汇编启动代码中，定义了两个变量_machine_rach_type（保存了机器类型ID）、_atags_pointer（保存了设备树/标签列表指针）；

2、上电后，引导加载程序被加载到内存（ARM没有BIOS这类固件程序），在将控制权交给内核时，将设备树指针传给内核；

3、将DTB转换为树状结构，节点用一个结构体标识；

4、扫描DTB，获取chosen节点的bootargs、initrd属性的值，并保存在全局变量中，其中保存了一些系统参数；

5、内核根据机器类型ID扫描机器描述符列表（机器描述符在编译时被保存在一个特殊段，并使用一个数据结构标识），确定机器描述符。

三、代码分析

请参考http://www.wowotech.net/linux_kenrel/dt-code-analysis.html