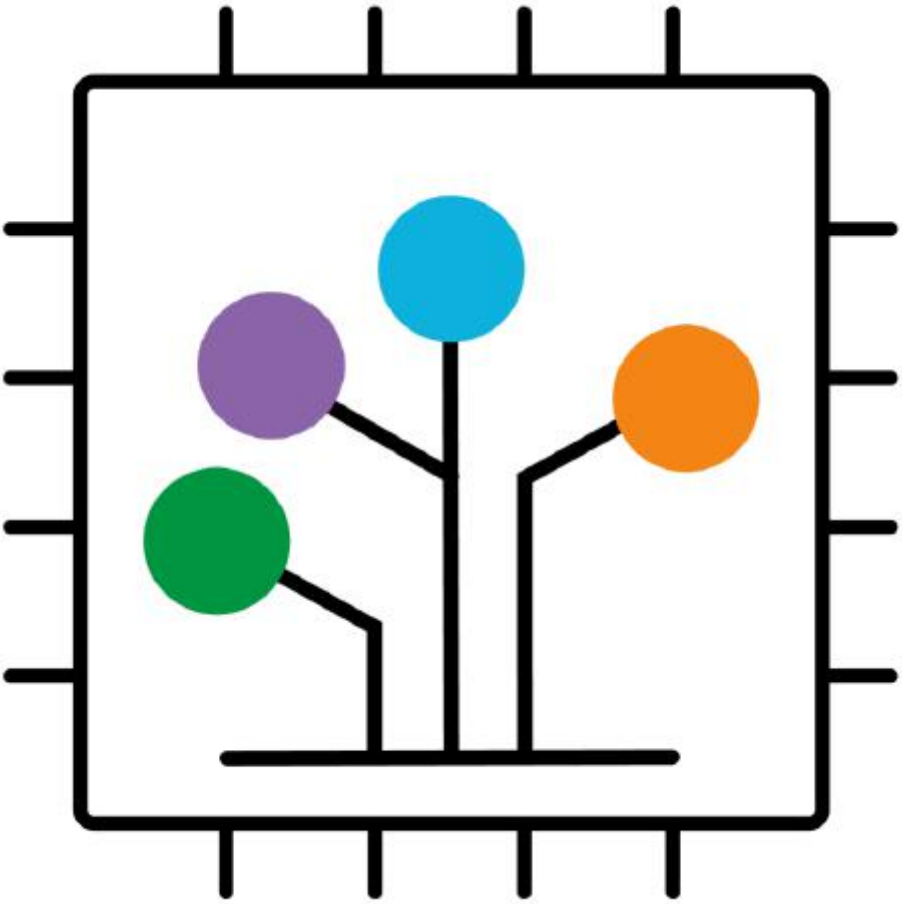
设备树

官网：devicetree.org

翻译：Ethan-Gao



0.翻译说明

此文只是对设备树官方文档的大致翻译，目前是第一个版本，存在许多不太准确的地方，后续会进行改进，如果感觉有出入，请对照官方文档进行查看，一切以官方文档为准，官方文档下载地址:https://www.devicetree.org/specifications/，同时推荐宋宝华大神讲解的设备树的一篇文章，比较好，地址:http://blog.csdn.net/21cnbao/article/details/8457546。
本文档下载地址：https://github.com/Ethan-Gao

1.本文说明

目的和范围

为了启动一台电脑，需要很多软件系统的协同合作。我们可能需要固件完成底层硬件的初始化，然后向 BIOS 和其他引导程序传递控制权，接着 BIOS 和其他引导程序会像操作系统传递控制器。通常来说，如果我们能够定义一些统一的接口和规则规范，那么各个组件之间的相互交互将会不那么麻烦。在这篇文章里面，我们一般说的引导程序是指一个程序组件，它用于初始化系统运行状态和执行其他的程序组件(可以称作专用计算机系统)。引导程序包括：固件、引导程序、监督程序，专用计算机系统包括：固件、引导程序、监督程序、操作系统、专用软件，很多软件可能既存在于引导程序里面，又存在于客户程序里面，比如监督程序。

这篇文档，针对一个最小系统运行的必要条件，提供了一个完整的从引导程序到专用计算机系统的接口定义。

本文主要针对嵌入式系统，其一般由硬件、操作系统、专用软件组成，主要用于完成特定任务而定制的计算机。和通用计算机不同，它一般由用户根据特定硬件定制开发而成，其他的一些特点还有：
a.拥有特定的外设，比如可能因为应用要求特殊定制
b.拥有优化和精简的的操作操作，便携编译
c.限制了用户可用的接口
d.资源一般有限，比如只有较小的内存和特定的存储设备
e.实时性受限
f.运行各种各样的操作系统，比如 linux、RTOS、客户定制的
本文档答题结构：
第一章：本篇文章的结构
第二章：介绍设备树的特点和属性
第三章：设备节点的定义
第四章：设备的匹配
第五章：设备树的物理结构
一些名词的说明：
Shall:指代根据标准的规定，强制的要求，不可有偏差，必须严格按照标准来进行

Should:指代根据标准的规定，在很多种可能的情况下，推荐的使用方法，这个并不是必须的，也可以说是推荐这么做
May:表示根据标准的限制，允许这么做

与 IEE1275、ePAR 标准关系

设备树跟 IEEE 1275 Open Firmware 标准有些关联，原本的标准和其衍生的标准(比如 CHRP 和 PAPR)主要是针对通用计算机的，比如同一个操作系统如何运行于具有相同处理器家族的不同计算机和不同外设的计算机。由于嵌入式计算机的特殊性，上面的这些为解决通用计算机的跨平台标准，在嵌入式系统上并未使用。设备树标准跟 IEEE 1275 标准有如下不同之处：
a.插拔插设备驱动
b.Fcode(不知道是什么)
c.可编程的固件接口
d.Fcode 调试(不知道是什么)
e.操作系统调试
IEEE-1275 保留的内容是引导程序可以通过的设备架构的概念，描述和传达系统硬件信息给客户端程序，从而消除对客户端程序难以访问硬件的问题。该规范部分取代了 ePAPR [EPAPR]规范，ePAPR 规范描述了 PowerISA 是如何使用设备树概念的，并且也涵盖一些通用的概念，以及 PowerISA 规则的设备匹配。本文档源自 ePAPR，但是删除了特定体系结构的匹配，或者说把他们移动到了最后的附录章节。

32 位和 64 位支持

设备树同时支持具有 32 位和 64 位寻址能力的 CPU，以下部分描述了使用在 32 位或者 64 位 CPU 时的要求和相关注意事项。

术语定义

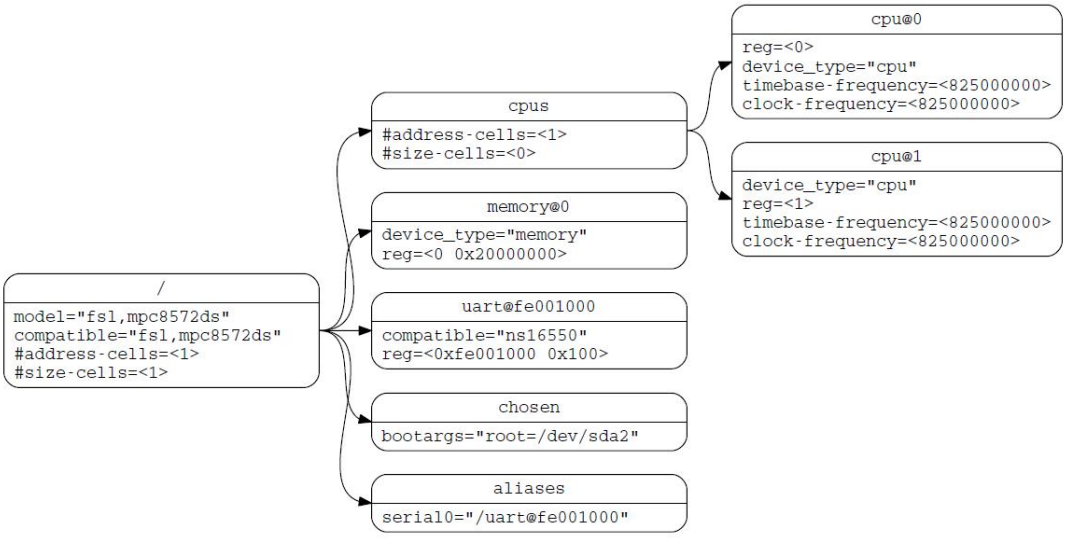
AMP:不对称多处理。计算机多个 CPU 被分成若干组，每一组运行一个不同的操作系统映像，所有的 CPU 可相同可不同
Boot CPU:引导程序指向操作系统入门点的第一个 CPU
Book II-E:嵌入式环境。 Power ISA 规范规定了管理模式的操作方式和嵌入式 Power 处理器使用的相关资源。
Boot program:一般用于指代软件组件，其初始化系统状态并执行另一个软件组件，通常称为称为客户端程序，其组成包括：固件、引导程序、管理程序。
Client program:包含操作系统和应用的完整系统。其组成包括：引导程序、管理程序、操作系统、应用软件
Cell:一个 32 位的信息单元
DMA:直接存储控制
DTB:设备树可执行文件，设备树编译之后产生的可执行二进制文件
DTC:设备树编译器，用于将 DTS 源文件编译成 DTB 可执行文件

DTS:设备树源文件，设备树的文本文件
Effective address:有效地址，CPU 可以访问到的所有存储地址
Physical address:物理地址，CPU 可以访问到的外设地址，多指内存控制器可访问地址
Power ISA:电源指令集架构
interrupt specifier:中断属性的描述，通常包含指定中断号、灵敏度、触发机制等信息
secondary CPU:除了引导程序以外的 CPU
SMP:对称多处理。两个或者多个 CPU 共享相同的体系结构和外设，并且运行当个操作系统
Soc:片上系统，集成当个或者多个 CPU 和一些外设的计算机芯片
Unit address:一个父节点的地址空间中表示节点地址的名字
quiescent CPU:静态 CPU,是一种处于不能干扰其他 CPU 的正常操作的状态，其他运行的 CPU 的正常运行也不会影响到其状态，除了明确启用否则一直处于这种状态

2.设备树简介

总述

DTS 规范定义了一种叫做设备树的概念，用于描述系统硬件。引导程序加载设备树到客户端程序，并且向客户程序传递一个设备树地址的指针。本节描述了设备树的逻辑结构，并且解析了一些用于描述设备节点的基本属性。第六章描述了如何匹配 DTS 设备，第八章描述了设备树在内存中的存放和编码方式。
设备树是一种树状数据结构，使用节点来描述系统的设备。每个节点都有属性值，用来描述系统将要如何使用这个设备。除了根节点没有父节点，其他所有节点都有。DTS 规范描述的设备树，并不一定会被客户应用程序检测并使用。比如，PCI 架构开启了客户程序对设备的匹配和检测，然而设备树并非一定需要有描述 PCI 设备的节点信息，但是如果主机上无法检测到 PCI 主机桥，此时我们就需要一个 PCI 主机控制器的设备节点了。
实例：
下面是一个简单的设备树示例，它足以启动一个简单的操作系统，具有平台类型、CPU、内存信息的描述，设备节点旁边显示了属性和值。



设备树结构和转换

节点名字

节点名字规范
每一个设备树节点都以下面方式命名:node-name@unit-address
Node-name 表示节点的名字，以下字母、数字、下划线组成，Node-name 应该以大写或者小写字母开头，用以描述设备的通用类，如下所示

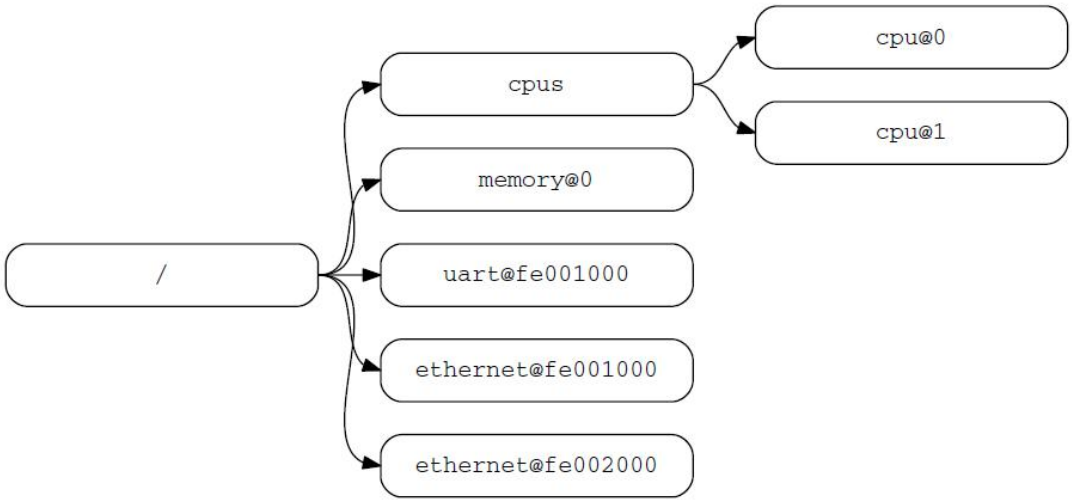
Table 2.1: Valid characters for node names

Character	Description
0-9	digit
a-z	lowercase letter
A-Z	uppercase letter
,	comma
.	period
_	underscore
+	plus sign
-	dash

unit-address 用户描述节点位于特定类的总线的位置，其由 1 个或者多个上述表格的字符组成，unit-address 必须和节点的 reg 属性匹配，如果节点没有 reg 属性，unit-address 如果不存在，节点就不会有 reg 属性，此时 node-name 就会单独存在，看起来和其他节点会有所不同。对于设备的匹配，一般都会要求指定同时 reg 和 unit-address 的格式。根节点没有 node-name 和 unit-address，它由一个分号符/来进行区别和辨识。
下图中:

含有 CPU 名字的：通过 cpu 后面的值 0 和 1 进行区分

含有 Ethernet 名字的：通过 Ethernet 后面的值 FE001000 和 FE002000 进行区分



命名推荐

节点的名字最好通用，不需要使用它的编程模块名称，但是要能够反映设备的功能。一般来说，节点的名字有以下几个选择:

atm cache-controller compact-flash can cpu crypto disk display dma-controller ethernet ethernet-phyfdc flash gpio i2c ide interrupt-controller isa keyboard mdio memory memory-controller mouse nvram parallel pc-card pci pcie rtc sata scsi serial sound spi timer usb vme watchdog

路径名称

通过指定完整路径，我们可以唯一的辨别一个节点，从根节点开始顺着后续节点到达期望的节点。指定设备完整路径如下：/node-name-1/node-name-2/node-name-N，例如在上面 2.21a 图中，我们可以这样指定完整路径：/cpus/cpu@1

根节点是/,如果节点的完整路径是确定的，这可以省略单元地址，若路径是模糊的，则其行为是未定义的。

d.属性

设备树上的每个节点都有对属性的相关描述，由一个名称和数值组成

e.属性名称

主要有以下内容组成，非标准属性名字应指定唯一的字符串前缀，例如股票代码符号，公司或组织的名称，比如:

fsl,channel-fifo-len

ibm,ppc-interrupt-server#s

linux,network-index

属性值

它是由一个含有数组组成，数组可以是 0 或者包含与属性相关信息的很多字节组成。如果设置 true 和 false 字段，这个属性可能为空，在这种情况下，属性属于过度描述。下图描述了一些 DTS 规范定义的属性值

Value	Description
<empty>	Value is empty. Used for conveying true-false information, when the presence of absence of the property itself is sufficiently descriptive.
<u32>	A 32-bit integer in big-endian format. Example: the 32-bit value 0x11223344 would be represented in memory as: address 11 address+1 22 address+2 33 address+3 44
<u64>	Represents a 64-bit integer in big-endian format. Consists of two <u32> values where the first value contains the most significant bits of the integer and the second value contains the least significant bits. Example: the 64-bit value 0x1122334455667788 would be represented as two cells as: <0x11223344 0x55667788>. The value would be represented in memory as: address 11 address+1 22 address+2 33 address+3 44 address+4 55 address+5 66 address+6 77 address+7 88
<string>	Strings are printable and null-terminated. Example: the string “hello” would be represented in memory as: address 68 'h' address+1 65 'e' address+2 6C 'l' address+3 6C 'l' address+4 6F 'o' address+5 00 '\0'
<prop-encoded-array>	Format is specific to the property. See the property definition.
<phandle>	A <u32> value. A *phandle* value is a way to reference another node in the devicetree. Any node that can be referenced defines a phandle property with a unique <u32> value. That number is used for the value of properties with a phandle value type.
<stringlist>	A list of <string> values concatenated together. Example: The string list “hello”,”world” would be represented in memory as: address 68 'h' address+1 65 'e' address+2 6C 'l' address+3 6C 'l' address+4 6F 'o' address+5 00 '\0' address+6 77 'w' address+7 6f 'o' address+8 72 'r' address+9 6C 'l' address+10 64 'd' address+11 00 '\0'

标准属性

DTS 规范规定了一个标准的设备节点的设置，本章节将进行这些属性的详细描述。对于使用标准属性，我们需要指定额外的要求或者约束，第四章将讲解这些要求和约束。
注意:这些实例中，并非所有设备树节点和属性都使用了 DTS 格式

Compatible

属性名称:compatible
数值类型:字符串列表
描述:
此属性由一个或者多个字符串组成，用来定义针对特定开发平台的特定设备，字符串列表会被客户程序的设备驱动程序使用，此属性值由一个都好连接符连接各个字符串知道字符结束，这些属性一般从最特殊到最一般。这种表示方式允许设备对一个家族的相同设备进行兼容，这样一个设备驱动能够匹配多个设备。
推荐格式:”manufacturer,model”
manufacturer 用来描述厂商，model 用来描述具体产品
例子：compatible = "fsl,mpc8641-uart", "ns16550";
这个例子中，操作系统会首先尝试使用 fsl,mpc8641-uart 进行设备驱动程序的匹配，如果没有发现匹配的设备，接下来会尝试匹配 ns16550 进行设备驱动的匹配

Model

属性名字:model
数值类型:字符串列表
描述:
此属性是一个字符串，用来指定从设备的厂商的模型

推荐格式:”manufacturer,model”
manufacturer 用来描述厂商，model 用来描述具体产品
例子：
model = "fsl,MPC8349EMITX";

Phandle

名字：phandle
类型：u32
描述：

此属性为设备树的节点指定一个唯一的数字标识符，当其他节点需要使用这个属性时，可以直接使用此属性值
例子：

```
pic@10000000 {
phandle = <1>;
interrupt-controller;
};
interrupt-parent = <1>;
```

phandle 值设置为 1，其他设备节点可以直接通过这个值应用此节点，比如 interrupt-parent 就是对它的引用

Status

名称：status
类型：字符串
描述：
此属性描述此设备是否可使用，可使用字符串如下

Value	Description
"okay"	Indicates the device is operational
"disabled"	Indicates that the device is not presently operational, but it might become operational in the future (for example, something is not plugged in, or switched off). Refer to the device binding for details on what disabled means for a given device.
"fail"	Indicates that the device is not operational. A serious error was detected in the device, and it is unlikely to become operational without repair.
"fail-sss"	Indicates that the device is not operational. A serious error was detected in the device and it is unlikely to become operational without repair. The sss portion of the value is specific to the device and indicates the error condition detected.

#address-cells 和#size-cells

名称：#address-cells, #size-cells
类型：u32
描述：
这两个属性可以在任何具有子节点的设备节点中使用，其描述了如何处理子设备节点
#address-cells 定义了用于对子节点的 reg 中的 address 段进行编码的<u32>单元的数量
#size-cells 定义了用于对子节点的 reg 中的 size 段进行编码的<u32>单元的数量
这两个属性不会从设备树的祖先进行继承，他们必须被明确定义，符合标准规范的引导程序应该在所有具有子节点的节点上提供 address 单元和 size 单元。如果没有定义，客户程序默认初始#address-cells=2,#size-cells=1
例子：

```
soc {
        #address-cells = <1>;
        #size-cells = <1>;
```

```
            serial {
            compatible = "ns16550";
            reg = <0x4600 0x100>;
            clock-frequency = <0>;
            interrupts = <0xA 0x8>;
            interrupt-parent = <&ipic>;
        };
};
```

这个例子里面，#address-cells, #size-cells 都设置成 1，此设置指定：需要一个单元来表示地址，并且也需要一个单元来表示该节点的子节点的大小。
此串口设备的 reg 属性需要明确指定地址(0x4600)和大小(0x100)

Reg

名字：reg
数值：符合(address number)的任意数字对
描述：
此属性描述了由其父总线定义的地址空间所描述的设备资源的地址，最常见的是内存 IO 寄存器的偏移量及其长度块，但在某些总线类型上可能有不同含义，地址空间中定义的根节点是 cpu 的真实地址。这个值由任意符合(address number)这种类型的数字组成，这两个数值都是 u32 类型，我们需要在父节点指定#address-cells, #size-cells 以便使用这个属性，如果父节点里面指定了#size-cells 这个值，reg 属性里面可以省略这个长度值
例子：
假设一个设备有一个 soc 芯片，其包含 2 块寄存器(a:32bit 偏移 0x3000;b:256bit 偏移 0xfe00)，那么 reg 属性值如下(假设#address-cells=1,#size-cells=1)
reg = <0x3000 0x20 0xFE00 0x100>;

Virtual-reg

类型:u32
描述:
此属性指定了一个有效的地址，此地址映射了设备节点的 reg 属性中定义的第一个地址。另外，此属性允许引导程序向操作系统提供一个物理地址到虚拟地址的转换表。

Ranges

类型:<空类型>或者<子地址空间 父地址空间 长度>
描述:
此属性提供了一种在子节点地址空间和父节点地址空间之间的映射，格式为<子地址空间 父

地址空间 长度>这种三元组，如果此属性定义为空，表示父节点和子节点的地址空间相同，不需要地址转换，如果当年设备树没有此节点，默认在子节点地址空间和父节点地址空间之间没有映射。
地址转换例子：

```
soc {
    compatible = "simple-bus";
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = <0x0 0xe0000000 0x00100000>;
    serial {
        device_type = "serial";
        compatible = "ns16550";
        reg = <0x4600 0x100>;
        clock-frequency = <0>;
        interrupts = <0xA 0x8>;
        interrupt-parent = < &ipic >;
    };
};
```

上面例子中的 ranges 操作，将父地址空间的 0x0 地址映射到子地址空间的 0xe0000000，映射长度为 0x00100000, 通过此映射，如果我们要访问串口设备，只需访问地址 0xe0000000+0x4600=0xe0004600

Dma-ranges

类型:<空类型>或者<子地址空间 父地址空间 长度>
描述:
此属性更 ranges 类似，但是需要父节点地址空间支持 DMA 操作

2.4 中断及其映射

设备树中断继承自 Open Fireware Recommended 规范，在设备中存在一个表示中断的逻辑中断树，其指明了硬件中断中的中断层次和路由，一般称为中断树，在技术上它有针对性的非循环图。中断控制器的中断源的物理路径可以用以下路径表示：中断-父属性，能够形成中断的设备节点都会包含一个中断父属性，这个父属性有一个 phandle 属性值用以表示中断的发生路径，通常是中断控制器。如果一个可产生中断的设备没有中断父属性，则认为其中断父属性就是自己。
现在，所有可产生中断的设备都包含一个中断属性，这个属性用来表示设备的一个或者多个

中断源，每个中断源都用一个中断号表示，这个中号与中断体系有关，它依赖于根节点的中断域。通常使用#interrupt-cells 属性来表示根节点的中断域，它是一个 u32 类型的数值用来指定中断号，例如，对于 PIC 中断控制器，一个中断选择器需要两个 32 位值，包括中断号和电平状态。

中断域是解释中断说明的关键所在，域的根不是中断控制器就是中断连接器

1.中断控制器

物理设备，需要驱动程序来控制其传输，它也可能级连到另一个中断域，设备树使用中断需要在节点指定一个 interrupt-controller 属性

2.中断连接器

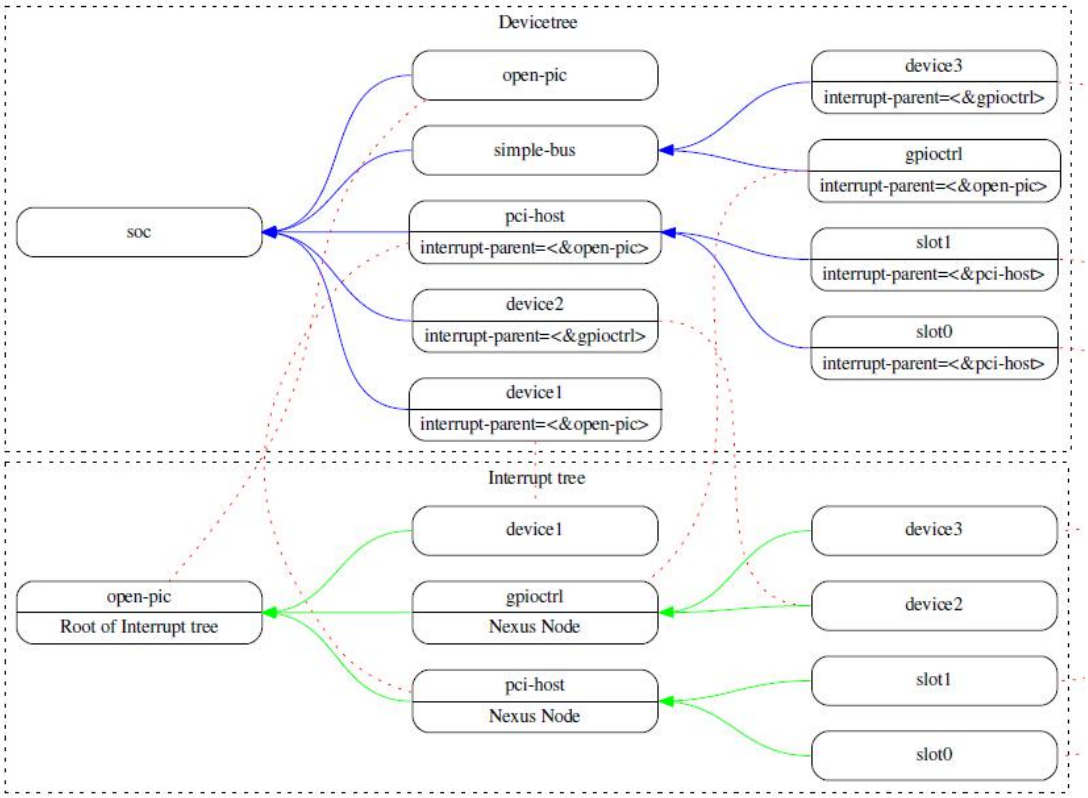
软件节点，他定义了一个中断域到另一个中断域之间的转换，转换过程基于域特定信息和总线信息，域之间的这种转换关系是基于中断映射属性完成的，例如，PIC 控制器设备节点可能是中断连接，他定义了中断命名空间到中断控制器硬件之间的转换。寻找中断树的根节点方法为：遍历中断树，直到到达了中断控制器节点，要求此节点没有中断属性，并且没有明确的中断父节点。下图是一个实例，显示了中断的层级关系，它显示了设备的自然结构以及每个节点位于逻辑中的位置。

a.Open-pic:中断树根

b.中断树根有三个子节点：devicel、PCI 主机控制器、GPIO 控制器，中断路线直接通过 open-pic

c.存在三个中断域：open-pic 节点、PCI 主机桥节点、GPIO 控制器节点

d.两个中断连接点:PCI 主机桥、GPIO 控制器



中断产生设备属性

Interrputs

类型:

描述：
此属性定义了设备产生的中断，此属性数值由一个任意数字的中断描述符组成，中断描述符格式根据中断根域的匹配来完成
例子：
对于 PIC 兼容中断域，中段描述符通常由两部分组成：中断号、级别信息。下面例子定义了一个中断描述符，包含一个中毒啦号 0xa 和电平值 8
interrupts = <0xA 8>;

Interrupt-parent
类型:phandle
描述:
由于中断树中节点的层次结构可能与设备不匹配，所以我们可以显式定义父中断的属性，这个属性值就是 phandle，如果设备中不存在此属性，则假设其中断父属性就是他的设备树父母。

中断控制器属性

#interrupt-cells
类型:u32
描述:
此属性定义了中断域里面的中断描述符所所要求的解码单元的数目

#interrupt-controller
类型:空
描述:
定义一个节点当作中断控制器节点

中断连接器属性

Interrupt-map:
类型:
描述:
此属性主要用于中断映射，主要指明中断如何映射。中断映射是一个表，由五部分组成:子单元地址、子中断描述符、父中断、父中断单元地址、父中断描述符

Ineterrupt-map-mask
类型:
描述：
此属性指定一个掩码，用于在 interrupt-map 属性中查找闯入单元的中断描述符

#interrupt-controller

类型::u32
描述:
此属性定义了中断域里面的中断描述符所所要求的解码单元的数目

中断映射例子

以下例子显示了具有 PCI 总线控制器和用于描述两个 PCI 插槽(IDSEL 0x11 0x12)的中断路线，插槽 1 和 2 的 INTA、INTB 、INTC、INTD 引脚连接到 Open PCI 中断控制器

```
soc {
    compatible = "simple-bus";
    #address-cells = <1>;
    #size-cells = <1>;

    open-pic {
        clock-frequency = <0>;
        interrupt-controller;
        #address-cells = <0>;
        #interrupt-cells = <2>;
    };

    pci {
        #interrupt-cells = <1>;
        #size-cells = <2>;
        #address-cells = <3>;
        interrupt-map-mask = <0xf800 0 0 7>;
        interrupt-map = <
            /* IDSEL 0x11 - PCI slot 1 */
            0x8800 0 0 1 &open-pic 2 1 /* INTA */
            0x8800 0 0 2 &open-pic 3 1 /* INTB */
            0x8800 0 0 3 &open-pic 4 1 /* INTC */
            0x8800 0 0 4 &open-pic 1 1 /* INTD */
            /* IDSEL 0x12 - PCI slot 2 */
            0x9000 0 0 1 &open-pic 3 1 /* INTA */
            0x9000 0 0 2 &open-pic 4 1 /* INTB */
            2.4. Interrupts and Interrupt Mapping 19
            Devicetree Specification, Release 0.1
            0x9000 0 0 3 &open-pic 1 1 /* INTC */
            0x9000 0 0 4 &open-pic 2 1 /* INTD */
        >;
    };
};
```

上面设备树之中，Open PIC 中断控制器通过 interrupt-controller 属性进行表示，interrupt-map 列表中的每一行由五个部分组成：子单元地址和中断描述符，这个 interrupt-map 被映射到

interrupt-parent 节点，它包含有一个特定的父单元地址和中断描述符。
例如，interrupt-map 第一行指定了插槽 1 的 INTA 的映射地址，0x8800 0 0 1 &open-pic 2 1 /* INTA */，详细解释如下
0x8800 0 0-子单元地址，1-子中断描述符，&open-pic:父中断，父单元地址:空(由于 open-pic 节点里面#address-cells = <0>)，2 1-父中断描述符，

3.节点要求

基本设备节点类型

本章将根据 DTS 标准规范，介绍设备节点的基本要求。所有设备树必须有一个更节点，下面这些也是在所有设备的根节点里面必须存在的：
a.一个/cpu 节点
b.最少一个内存节点

根节点

设备树具有一个根节点，其他所有后续节点都是根节点的子节点，根节点的完整路径是/，根节点的所有属性如下表所示:

Property Name	Usage	Value Type	Definition
`#address-cells`	R	`<u32>`	Specifies the number of `<u32>` cells to represent the address in the `reg` property in children of root.
`#size-cells`	R	`<u32>`	Specifies the number of `<u32>` cells to represent the size in the `reg` property in children of root.
`model`	R	`<string>`	Specifies a string that uniquely identifies the model of the system board. The recommended format is "manufacturer,model-number".

`compatible`	R	`<stringlist>`	Specifies a list of platform architectures with which this platform is compatible. This property can be used by operating systems in selecting platform specific code. The recommended form of the property value is:
			`"manufacturer,model"`
			For example:
			`compatible = "fsl,mpc8572ds"`

Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

/aliases 节点

设备树可能具有别名节点，它用于定义一个或者多个别名属性。别名节点必须位于设备树的根节点，也必须具有别名。别名节点的每一个属性都定义一个别名，属性名字就是别名，属性值指定了设备树节点的完整路径，例如，serial0 ="/simple-bus@fe000000/serial@llc500"，

这里的 serial0 就属于别名。
使用别名时，必须小写，可包含以下所有字符，如表:

Character	Description
0-9	digit
a-z	lowercase letter
-	dash

别名属性值是设备路径，其被编码为一个字符串，这个值表示此节点的完整路径(此路径无需包含叶子节点的路径)。操作系统可能会使用一个别名值来引用它的完整路径，对于操作系统来说，当把字符串当作路径时，必须能够检测和使用别名。
例如：

```
aliases {
    serial0 = "/simple-bus@fe000000/serial@llc500";
    ethernet0 = "/simple-bus@fe000000/ethernet@31c000";
}
```

给操作系统传递 serial0 这个别名，操作系统会检查此别名节点，然后引用其完整路径，也就是/simple-bus@fe000000/serial@llc500 这个路径

/meory 节点

所有的设备，都要求提供一个内存节点，用以告诉系统可用的物理内存。如果一个系统有多个内存可使用的内存范围，则会创建多个内存节点，或者也可以在 reg 这个属性值里面指定一个单一的内存节点，另外这个节点的名字必须使用 memory。操作系统可以使用任何内存，除非这段内存没有设置为预留内存。然后，在更改用于访问真实页面的存储属性之前，操作系统会负责执行与架构和需求相关的具体操作，这个操作可能包括从缓存中刷新真实页面。引导程序需要能够保证，在没有更改存储属性之前，操作系统能够安全的访问所有内存(包括预留的内存)，当 WIMG=0b001x 表示：
a.不需要写入 b.不禁止缓存 c.内存一致性
如果 VLE 存储属性受支持，那么 VLE=0，内存属性如下表所示:

Property Name	Usage	Value Type	Definition
device_type	R	<string>	Value shall be “memory”
reg	R	<prop-encoded-array>	Consists of an arbitrary number of address and size pairs that specify the physical address and size of the memory ranges.
initial-mapped-area	O	<prop-encoded-array>	Specifies the address and size of the Initial Mapped Area Is a prop-encoded-array consisting of a triplet of (effective address, physical address, size). The effective and physical address shall each be 64-bit (<u64> value), and the size shall be 32-bits (<u32> value).
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

例子：

假设有一个 64 位的操作系统，具有以下内存布局：
RAM:起始地址 0x0，长度 0x80000000(2GB)
RAM:起始地址 0x10000000,长度 0x10000000(4GB)
内存节点定义方式如下，这里假设#address-cells = <2>,#size-cells=<2>，这节点如下
Example #1

```
memory@0 {
        device_type = "memory";
        reg = <0x000000000 0x00000000 0x00000000 0x80000000
        0x000000001 0x00000000 0x00000001 0x00000000>;
};
```

Example #2

```
memory@0 {
        device_type = "memory";
        reg = <0x000000000 0x00000000 0x00000000 0x80000000>;
};
memory@100000000 {
        device_type = "memory";
        reg = <0x000000001 0x00000000 0x00000001 0x00000000>;
};
```

这里的 reg 属性用于定义两块内存的地址和长度，这里 2GB 的内存区域省略了。注意这里假设了根节点里面#address-cells = <2>,#size-cells=<2>，他们意味着这两个 u32 定义的单元的地址和大小。

/chosen 节点

这个节点并不代表世界的系统设备，它只是用来描述系统固件在运行时所选择/指定的参数，它应该是根节点的一个子节点，这个节点属性如下表所示

Table 3.4: /chosen Node Properties

Property Name	Usage	Value Type	Definition
bootargs	O	<string>	A string that specifies the boot arguments for the client program. The value could potentially be a null string if no boot arguments are required.
stdout-path	O	<string>	A string that specifies the full path to the node representing the device to be used for boot console output. If the character ":" is present in the value it terminates the path. The value may be an alias. If the stdin-path property is not specified, stdout-path should be assumed to define the input device.
stdin-path	O	<string>	A string that specifies the full path to the node representing the device to be used for boot console input. If the character ":" is present in the value it terminates the path. The value may be an alias.
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

Note: All other standard properties (section *2.3*) are allowed but are optional.

例子：

```
chosen {
        bootargs = "root=/dev/nfs rw nfsroot=192.168.1.1 console=ttyS0,115200";
```

};
早期版本的设备树可能不支持这种方式，因此为了更好的兼容性，操作系统可能希望支持 linux 和 stdout-path，这两个属性的用法是相同的。

/cpus 节点属性

所有设备都要求拥有此节点，它不代表系统中的真实设备，而是作为代表系统的 CPU 及其子 CPU 的容器。这个节点属性如下，

Table 3.5: /cpus Node Properties

Property Name	Usage	Value Type	Definition
#address-cells	R	<u32>	The value specifies how many cells each element of the reg property array takes in children of this node.
#size-cells	R	<u32>	Value shall be 0. Specifies that no size is required in the reg property in children of this node.
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

Note: All other standard properties (section *2.3*) are allowed but are optional.

/cpus/cpu*节点属性

cpu 节点代表一个硬件可执行块，它能够运行操作系统，同时也不会干扰可能正在运行其他操作系统的 CPU。共享 MMU 的硬件线程一般会用一个 CPU 节点表示，如果要设计其他更复杂的 CPU，对于 CPU 的匹配必须有足够清晰的描述，比如哪些县城不共享 MMU。CPU 和线程通过统一的数字空间进行编号，编号时应尽可能和 CPU 中断控制器的线程编号匹配。如果 cpus 节点里面，拥有和 CPU 节点里面完全相同的属性，此属性可能会被替代。操作系统首先会检查一个特定的 CPU 节点，如果期望节点不存在，接着我们就必须查找父 cpus 节点。对于每一个 cpu 而言，节点名字必须是 cpu.

/cpus/cpu*节点常用属性

下表是对/cpus/cpu*节点常用属性的描述，有些属性是标准属性

Property Name	Usage	Value Type	Definition
device_type	R	<string>	Value shall be "cpu".
reg	R	array	The value of *reg* is a <prop-encoded-array> that defines a unique CPU/thread id for the CPU/threads represented by the CPU node. If a CPU supports more than one thread (i.e. multiple streams of execution) the *reg* property is an array with 1 element per thread. The *#address-cells* on the /cpus node specifies how many cells each element of the array takes. Software can determine the number of threads by dividing the size of *reg* by the parent node's *#address-cells*. If a CPU/thread can be the target of an external interrupt the *reg* property value must be a unique CPU/thread id that is addressable by the interrupt controller. If a CPU/thread cannot be the target of an external interrupt, then *reg* must be unique and out of bounds of the range addressed by the interrupt controller If a CPU/thread's PIR is modifiable, a client program should modify PIR to match the *reg* property value. If PIR cannot be modified and the PIR value is distinct from the interrupt controller numberspace, the CPUs binding may define a binding-specific representation of PIR values if desired.
clock-frequency	R	array	Specifies the current clock speed of the CPU in Hertz. The value is a <prop-encoded-array> in one of two forms: • A 32-bit integer consisting of one <u32> specifying the frequency. • A 64-bit integer represented as a <u64> specifying the frequency.

Continued on next page

Table 3.6 continued from previous page

Property Name	Usage	Value Type	Definition
timebase-frequency	R	array	Specifies the current frequency at which the timebase and decrementer registers are updated (in Hertz). The value is a <prop-encoded-array> in one of two forms: • A 32-bit integer consisting of one <u32> specifying the frequency. • A 64-bit integer represented as a <u64>.
status	SD	<string>	A standard property describing the state of a CPU. This property shall be present for nodes representing CPUs in a symmetric multiprocessing (SMP) configuration. For a CPU node the meaning of the "okay" and "disabled" values are as follows: **"okay"** : The CPU is running. **"disabled"** : The CPU is in a quiescent state. A quiescent CPU is in a state where it cannot interfere with the normal operation of other CPUs, nor can its state be affected by the normal operation of other running CPUs, except by an explicit method for enabling or reenabling the quiescent CPU (see the enable-method property). In particular, a running CPU shall be able to issue broadcast TLB invalidates without affecting a quiescent CPU. Examples: A quiescent CPU could be in a spin loop, held in reset, and electrically isolated from the system bus or in another implementation dependent state.

enable-method SD	<stringlist>	Describes the method by which a CPU in a disabled state is enabled. This property is required for CPUs with a status property with a value of "disabled". The value consists of one or more strings that define the method to release this CPU. If a client program recognizes any of the methods, it may use it. The value shall be one of the following: **"spin-table"**: The CPU is enabled with the spin table method defined in the DTSpec. **"[vendor],[method]"**: Implementation dependent string that describes the method by which a CPU is released from a "disabled" state. The required format is: "[vendor],[method]", where vendor is a string describing the name of the manufacturer and method is a string describing the vendorspecific mechanism. Example: "fsl,MPC8572DS" **Note:** Other methods may be added to later revisions of the DTSpec specification.
cpu-release-addr SD	<u64>	The cpu-release-addr property is required for cpu nodes that have an enable-method property value of "spin-table". The value specifies the physical address of a spin table entry that releases a secondary CPU from its spin loop.
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition		

例子：

以下是拥有一个子 cpu 节点的 cpus 节点例子

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        device_type = "cpu";
        reg = <0>;
        d-cache-block-size = <32>; // L1 - 32 bytes
        i-cache-block-size = <32>; // L1 - 32 bytes
        d-cache-size = <0x8000>; // L1, 32K
        i-cache-size = <0x8000>; // L1, 32K
        timebase-frequency = <82500000>; // 82.5 MHz
        clock-frequency = <825000000>; // 825 MHz
    };
};
```

TLB 属性

L1 缓存属性

实例

多重缓存共享节点(/cpus/cpu*/l?-cache)

处理器和系统可能实现了额外的缓存层次结构，例如:二级缓存 L2、三级缓存 L3，这些缓存通常与 CPU 紧密结合或者可能在多个 CPU 之间共享。如果一个设备节点包含有 cache 属性，则它具有以上的描述类型。

缓存节点必须定义一个 phandle 属性，并且所有有关联或者共享的 cpu 节点和缓存节点也必须包含一个 next-level-cache 的属性值，它用来指定这个缓存节点的 phandle 节点值。缓存节点可用于在 CPU 节点或者设备的任何其他合适的位置使用，多重缓存和共享缓存属性如下:

Property Name	Usage	Value Type	Definition
compatible	R	<string>	A standard property. The value shall include the string "cache".
cache-level	R	<u32>	Specifies the level in the cache hierarchy. For example, a level 2 cache has a value of 2.
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

例子

下面例子中:设备树包含两个 CPU，每个 CPU 都有两个片上 L2 和共享 L3 缓存

```
cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
        device_type = "cpu";
        reg = <0>;
        cache-unified;
        cache-size = <0x8000>; // L1, 32KB
        cache-block-size = <32>;
        timebase-frequency = <82500000>; // 82.5 MHz
        next-level-cache = <&L2_0>; // phandle to L2
        L2_0:l2-cache {
            compatible = "cache";
            cache-unified;
            cache-size = <0x40000>; // 256 KB
            cache-sets = <1024>;
            cache-block-size = <32>;
            cache-level = <2>;
            next-level-cache = <&L3>; // phandle to L3
            L3:l3-cache {
                compatible = "cache";
                cache-unified;
                cache-size = <0x40000>; // 256 KB
                cache-sets = <0x400>; // 1024
                cache-block-size =
                cache-level = <3>;
```

```
                };
            };
        };
        cpu@1 {
            device_type = "cpu";
            reg = <0>;
            cache-unified;
            cache-block-size = <32>;
            cache-size = <0x8000>; // L1, 32KB
            timebase-frequency = <82500000>; // 82.5 MHz
            clock-frequency = <825000000>; // 825 MHz
            cache-level = <2>;
            next-level-cache = <&L2_1>; // phandle to L2
            L2_1:l2-cache {
                compatible = "cache";
                cache-unified;
                cache-size = <0x40000>; // 256 KB
                cache-sets = <0x400>; // 1024
                cache-line-size = <32> // 32 bytes
                next-level-cache = <&L3>; // phandle to L3
            };
        };
};
```

4.设备匹配

本章主要讲设备树的设备如何操作系统的驱动进行匹配，设备节点的兼容属性描述了如何匹配特定设备。匹配过程是一个双向的过程，例如一个新的总线类型可以当作一个简单的总线，从而实现设备的匹配，在这种情况下，兼容节点需要使用多个字符串列表以便能够匹配更多设备，从最特别的属性到最一般的属性。

匹配指南

通用规则

当在设备树上面创建一个新设备时，必须具有相应的匹配规则，它用来描述设备的相关属性，通常应当包含下述要求：

1.定义一个 compatible 字符串属性

2.对于新设备，使用标准属性，一把来说，最少包含有 reg 和 interrupts 两个属性值

3.如果一个新设备符号 DTS 标准规范，可以直接使用通用属性。
4.通用属性见下表
5.为了匹配一个新设备，推荐使用如下格式:
"<company>,<property-name>",其中<company>代表厂商，<property-name>产品名称
例如:"ibm,ppc-interrupt-server#s"

通用属性

本章列举了一些很有用的属性，他们可以应用在很多类型的场合和设备上，下面列举的一些属性比较通用，可以认为就是标准规定的属性。

Table 4.1: clock-frequency Property

Property	clock-frequency
Value type	<prop-encoded-array>
Description	Specifies the frequency of a clock in Hz. The value is a <prop-encoded-array> in one of two forms: a 32-bit integer consisting of one <u32> specifying the frequency a 64-bit integer represented as a <u64> specifying the frequency

Table 4.2: reg-shift Property

Property	reg-shift
Value type	<u32>
Description	The reg-shift property provides a mechanism to represent devices that are identical in most respects except for the number of bytes between registers. The reg-shift property specifies in bytes how far the discrete device registers are separated from each other. The individual register location is calculated by using following formula: "registers address" << reg-shift. If unspecified, the default value is 0. For example, in a system where 16540 UART registers are located at addresses 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, and 0x1C, a reg-shift = <2> property would be used to specify register locations.

Table 4.3: label Property

Property	label
Value type	<string>
Descriptio	The label property defines a human readable string describing a device. The binding for a given device specifies the exact meaning of the property for that device.

串行设备

串行类的匹配

串行设备类有各种各样的点对点的串行设备组成，常用的串行设备有:8250 UART、16550 UART、HDLC、BISYNC，这些设备大多数和 RS232 串口类具有兼容性。I2C 和 SPI 不应该被当

作串行设备，因为他们有他们自己的协议和规范，

clock-frequency Property

Table 4.4: clock-frequecy Property

Property	clock-frequency
Value type	<u32>
Description	Specifies the frequency in Hertz of the baud rate generator's input clock.
Example	clock-frequency = <100000000>;

Table 4.5: current-speed Property

Property	current-speed
Value type	<u32>
Description	Specifies the current speed of a serial device in bits per second. A boot program should set this property if it has initialized the serial device.
Example	115,200 Baud: current-speed = <115200>;

国际半导体 16450/16550 兼容串口要求

要想与国际半导体 16450/16550 通用窗口兼容，设备树需要使用下面的属性：

Table 4.6: ns16550 UART Properties

Property Name	Usage	Value Type	Definition
compatible	R	<string list>	Value shall include "ns16550".
clock-frequency	R	<u32>	Specifies the frequency (in Hz) of the baud rate generator's input clock
current-speed	OR	<u32>	Specifies current serial device speed in bits per second
reg	R	<prop encoded array>	Specifies the physical address of the registers device within the address space of the parent bus
interrupts	OR	<prop encoded array>	Specifies the interrupts generated by this device. The value of the interrupts property consists of one or more interrupt specifiers. The format of an interrupt specifier is defined by the binding document describing the node's interrupt parent.
reg-shift	O	<u32>	Specifies in bytes how far the discrete device registers are separated from each other. The individual register location is calculated by using following formula: "registers address" << reg-shift. If unspecified, the default value is 0.
virtual-reg	SD	<u32> or <u64>	See section 2.3.7. Specifies an effective address that maps to the first physical address specified in the reg property. This property is required if this device node is the system's console.

Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition

Note: All other standard properties (section 2.3) are allowed but are optional.

网络设备

网络设备是面向分组的通信设备，我们假设这种设备实现了 7 层 ISO 网络模型的数据链路层，并且使用 MAC 地址，例如常用的包括以太网、FDDI、802.11、令牌环行网

网络设备的匹配

Table 4.7: address-bits Property

Property	address-bits
Value type	<u32>
Description	Specifies number of address bits required to address the device described by this node. This property specifies number of bits in MAC address. If unspecified, the default value is 48.
Example	address-bits = <48>;

Table 4.9: mac-address Property

Property	mac-address
Value type	<prop-encoded-array> encoded as an array of hex numbers
Description	Specifies the MAC address that was last used by the boot program. This property should be used in cases where the MAC address assigned to the device by the boot program is different from the local-mac-address property. This property shall be used only if the value differs from local-mac-address property value.
Example	mac-address = [0x01 0x02 0x03 0x04 0x05 0x06];

max-frame-size Property

Table 4.10: max-frame-size Property

Property	max-frame-size
Value type	<u32>
Descriptio	Specifies maximum packet length in bytes that the physical interface can send and receive.
Example	max-frame-size = <1518>;

网络特别说明

除了上述网络设备类的属性之外，基于 IEEE 802.3 网络设备集合的以太网可以使用如下属性进行表示，下面列出的属性是对上面基本网络属性的补充，如下所示

max-speed Property

Table 4.11: max-speed Property

Property	max-speed
Value type	<u32>
Description	Specifies maximum speed (specified in megabits per second) supported the device.
Example	max-speed = <1000>;

phy-connection-type Property

Table 4.12: phy-connection-type Property

Property	phy-connection-type
Value type	<string>
Description	Specifies interface type between the Ethernet device and a physical layer (PHY) device. The value of this property is specific to the implementation. Recommended values are shown in the following table.
Example	phy-connection-type = "mii";

下面这些值是 phy-connection-type 这个属性的可选值，

Connection type	Value
Media Independent Interface	mii
Reduced Media Independent Interface	rmii
Gigabit Media Independent Interface	gmii
Reduced Gigabit Media Independent	rgmii
rgmii with internal delay	rgmii-id
rgmii with internal delay on TX only	rgmii-txid
rgmii with internal delay on RX only	rgmii-rxid
Ten Bit Interface	tbi
Reduced Ten Bit Interface	rtbi
Serial Media Independent Interface	smii
Serial Gigabit Media Independent Interface	sgmii
Reverse Media Independent Interface	rev-mii
10 Gigabits Media Independent Interface	xgmii
Multimedia over Coaxial	moca
Quad Serial Gigabit Media Independent Interface	qsgmii

Power ISA PIC 中断控制器

本节内容主要是 PIC 中断控制器的规范要求，其实现了 Open PIC 架构，并在其中指定了 PIC 中断寄存器的接口。Open PIC 中断控制器存在于 PIC 控制器域，由两个单元编码而成，第一个是中断号，第二个是中断触发方式，其中中断触发方式主要包含以下几种:
0-从低到高的跳变电平 1-低电平触发 2-高电平触发 3-从高到低的跳变电平

Simple-bus 兼容值

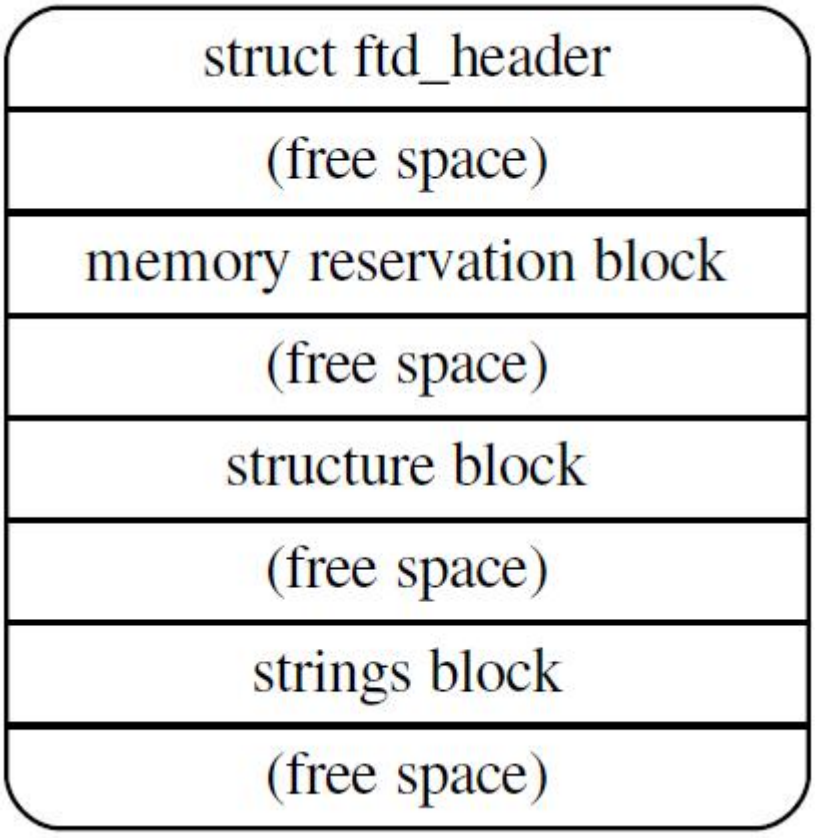
SOC 处理器可能具有内部 IO 总线，这些总线不能被设备探测和匹配，这种 IO 设备通常可以直接访问而无需额外的配置，像这种具有可以使用 simple-bus 这个属性值来表示。

Table 4.16: `simple-bus` Compatible Node Properties

Property Name	Usage	Value Type	Definition
`compatible`	R	`<string>`	Value shall include "simple-bus".
`ranges`	R	`<prop encoded array>`	This property represents the mapping between parent address to child address spaces (see section *2.3.8*, ranges).
Usage legend: R=Required, O=Optional, OR=Optional but Recommended, SD=See Definition			

5.物理架构

根据标准规定，一般认为设备树具有如下特点：包含了一个单、线性、无指针的数据结构，里面含有设备的信息。设备树结构以下几个部分：头部、预留内存区(可能不存在)、设备树结构区、字符串区域，结构图如下



版本

从原始设备树发展而来，现在有几种版本的设备树，我们可以通过头部信息判断其版本，因此操作系统可以通过这个信息来判断设备树时候能够匹配当前设备驱动程序。此文档讲述的版本是 17，按照 DTS 标准规定，如果需要较好的兼容性，一般需要提供 16 版本以后的 DTS 信息，操作系统也是这样。

头部

设备树头部由 C 结构定义而来，头部所有信息按照 32 位整型、大端格式存在。Flattended DTS 头部信息如下

```
struct fdt_header {
	uint32_t magic;
	// 0xd00dfeed
	uint32_t totalsize;
	// 整个设备树结构的总长度
	uint32_t off_dt_struct;
	// structure block 相对于头部的偏移
	uint32_t off_dt_strings;
	// string block 相对于头部的偏移
	uint32_t off_mem_rsvmap;
	// memory reserver block 相对于头部的偏移
	uint32_t version;
	// 版本信息
	uint32_t last_comp_version;
	// 兼容的最低设备树数据结构的版本
	uint32_t boot_cpuid_phys;
	// CPU 的物理 ID
	uint32_t size_dt_strings;
	// string block 的大小
	uint32_t size_dt_struct;
	// structure block 的大小
};
```

memory reserver block

作用

用于给操作系统提供保留内存，所谓保留内存，是指操作系统无法直接申请使用的内存，通

常用于特殊用途，比如用于保存 MMU 映射表、保存固件信息等等。如果需要使用保留内存，一般需要提供一些特殊属性，然后直接在驱动程序里面直接使用。

格式

```
struct fdt_reserve_entry {
    uint64_t address;
    uint64_t size;
};
```

在 C 结构里面可以看到，预留内存主要需要两个信息：预留物理内存的起始地址、预留内存的长度，特别注意

1.这两个值都是 64 位格式，如果实在 32 位系统上，高 32 位将会自动被忽略

2.从设备树开头位置起，预留内存的偏移地址需要 8 字节对齐

structure block

此区域为设备树的实体区域，所有设备树下相关的结构和内容都存在于此区域，这些结构是线性组织在一起的，从设备树开头位置起，此区域也需要位于偏移地址为 4 字节对齐的地址。

构成

此区域有多个部分组成，每部分含有一个标记(32 位整数)，有些标记含有额外的数据，这些数据的格式与这个标记有关。所有标记应该位于 32 对齐的地址，有的可能需要插入填充字节 0x0，5 个标记如下所示：

FDT_BEGIN_NODE (0x00000001)

这个标记代表一个设备节点的开始，后面紧跟着节点名字，名字是由字符串组成由 null 结束符结束，同时还应该包含节点的单元地址，如果没有地址的话，为了能够满足对齐的条件，可能需要使用 0x0 进行填充。然后依次添加下一个节点，直到到达 FDT_END_NODE 标志。

FDT_END_NODE (0x00000002)

标志着节点的结束

FDT_PROP (0x00000003)

设备树中的各种信息包含在此区域，各种属性值也存在与此区域，这些数据大致由下面格式组成：

```
struct {
    uint32_t len;          属性值的总长度
    uint32_t nameoff;  属性名字在 strings block 中的偏移
}
```

FDT_NOP (0x00000004)

此标记用于跳过一段设备节点信息，每当读取到这个标记，则解析程序会直接此节点信息，转而去读取下一个有效的节点信息

FDT_END (0x00000009)
整个设备树结构的结束标记

Tree struct

设备树是一个线性树，每个节点都是以 FDT_BEGIN_NODE 为起始以 FDT_END_NODE 标志位结束，所有节点和子节点属性值都存在 FDT_BEGIN_NODE 和 FDT_END_NODE 之间，有许多节点组成整个设备信息，最后会以 FDT_END 标志表示整个设备树的结束。更仔细的说，每个节点由以下部分组成:
1.需要忽略的节点，包含 FDT_NOP 标志
2.节点其实标志 FDT_BEGIN_NODE（包含节点名字和结束符，可能需要补充 0x0）
3.对于每个而言：包含任意数量的 FDT_NOP 和 FDT_PROP
4.按照上述格式描述的所有子节点
5.节点结束标志 FDT_END_NODE

Strings Block

此区域包含所有属性名字的字符串，没有终止符的字符串会简单的连接在一起，这个区域不要求对齐，可能位于设备树起始位置的任意偏移位置。

Alignment

对于 memory reserver block 和 structure block 区域，为了确保操作的数据是对齐的，我们应该把它们放到适合对齐的内存地址之上，具体来说，memory reserver block 需要边界 8 直接对齐而 structure block 需要边界 4 直接对齐。此外，每个 blob 可以作为一个整体进行重新定位，而不会影响子块的对齐。

6.dts 格式

设备树源文件为 dts 格式，通过使用 dtc 编译器将其编译成二进制 dtb 文件，然后传递给可以将这个 dtb 传递给内核，以下描述并不是十分正式的语法，但描述了表示设备的基本结构。

节点和属性定义

设备树使用节点名称和单元地址进行定义，大括号表示节点的开始和结束，他们之间可能有一个标签。

[label:] node-name[@unit-address] {
    [properties definitions]
    [child nodes]
}
节点可能包含属性定义，这个属性可能包含有子节点定义也可能没有，如果有，则需要将属性放在子节点之前，属性定义是通过名字值来进行匹配的，如下使用 value 来匹配
[label:] property-name = value;
若属性为空，则写法如下
[label:] property-name;
属性值可能是使用 32 位整数定义，使用 null 结束符结束
1.单元数组可以使用 C 语言风格的整数来表示，比如
Interrupts = <17 0xc>
2.使用 64 位数值代表 32 位单元
clock-frequency = <0x00000001 0x00000000>;
3.使用双引号表示以 null 字符结束的字符串(默认属性值包含有 NULL 字符)，比如
compatible = "simple-bus";
4.使用[]括起来，里面每个直接由两个十六进制数字表示，字节之间的可要可不要，以下两种方式等价
local-mac-address = [00 00 12 34 56 78];
local-mac-address = [000012345678];
5.各个部分使用逗号隔开，然后组合在一起
compatible = "ns16550", "ns8250";
example = <0xf00f0000 19>, "a strange property format";
6.使用&引用其他节点的属性，&紧跟其他节点或者某个节点的绝对路径
interrupt-parent = < &mpic >;
interrupt-parent = < &{/soc/interrupt-controller@40000} >;
7.使用&引用其他节点，但是没有<>，它会将当前节点信息添加到引用的节点上，
ethernet0 = &EMAC0;
8.可以省略 lable
reg = reglabel: <0 sizelabel: 0x1000000>;
prop = [ab cd ef byte4: 00 ff fe];
str = start: "string value" end: ;

DTS 文件布局

Ver1 版本的 DTS 文件一般写法如下
/dts-v1/;
[memory reservations]
    / {
        [property definitions]
        [child nodes]
    };
a./dts-v1/; 指明当前的版本(如果没有此信息，则默认版本为 0，类型会有些区别)

b.[memory reservations] 用来表示需要保留的内存信息，这部分格式如下：
/memreserve/ <address> <length>;<address> <length>均为 64 位 C 风格整数
c./ {}; 定义了根节点的段
d.支持 C/c++风格注释方式:/**/ //