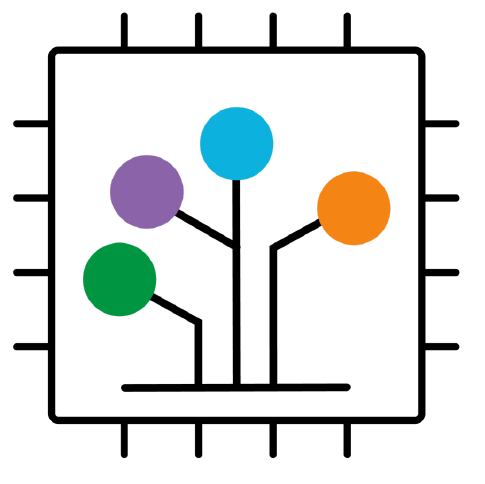
**设备树**

**官网：devicetree.org**

**翻译：Ethan-Gao**

****

# 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

# 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：

1. 拥有特定的外设，比如可能因为应用要求特殊定制
2. 拥有优化和精简的的操作操作，便携编译
3. 限制了用户可用的接口
4. 资源一般有限，比如只有较小的内存和特定的存储设备
5. 实时性受限
6. 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：

1. ：本篇文章的结构
2. ：介绍设备树的特点和属性
3. ：设备节点的定义
4. ：设备的匹配
5. ：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：

1. 插拔插设备驱动
2. Fcode(不知道是什么)
3. 可编程的固件接口
4. Fcode调试(不知道是什么)
5. 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

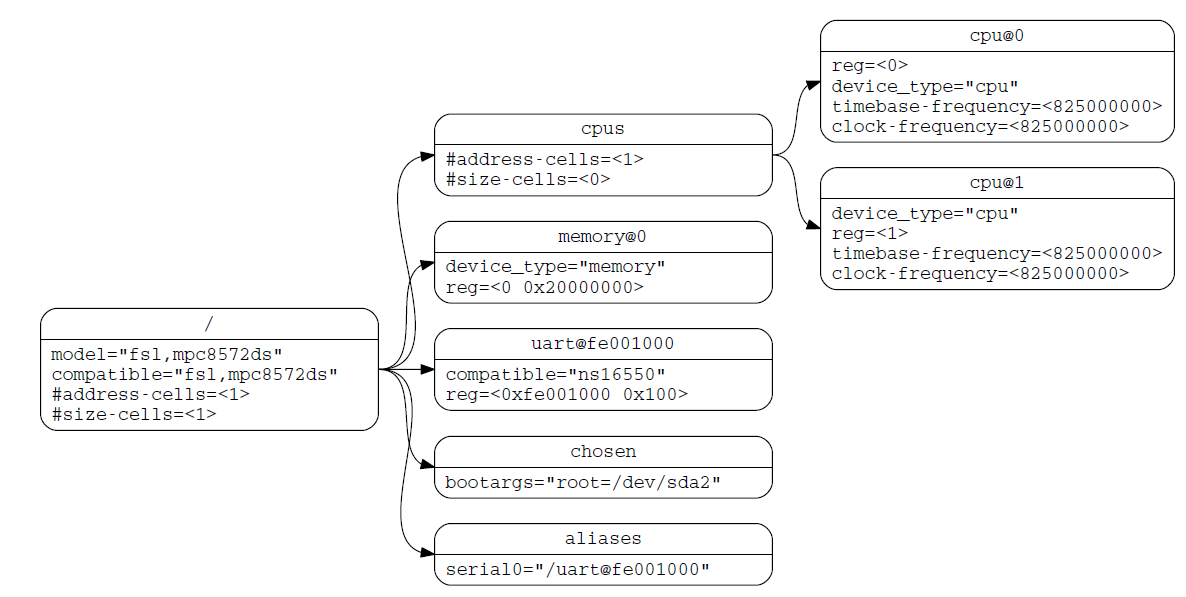
# 设备树简介

## 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。

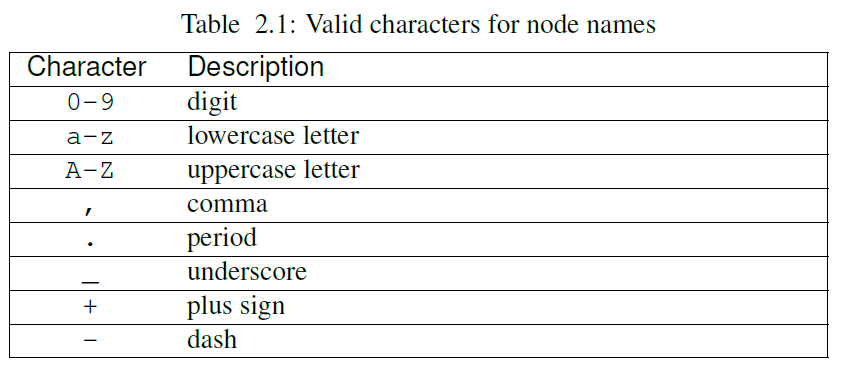
## 设备树结构和转换

### 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

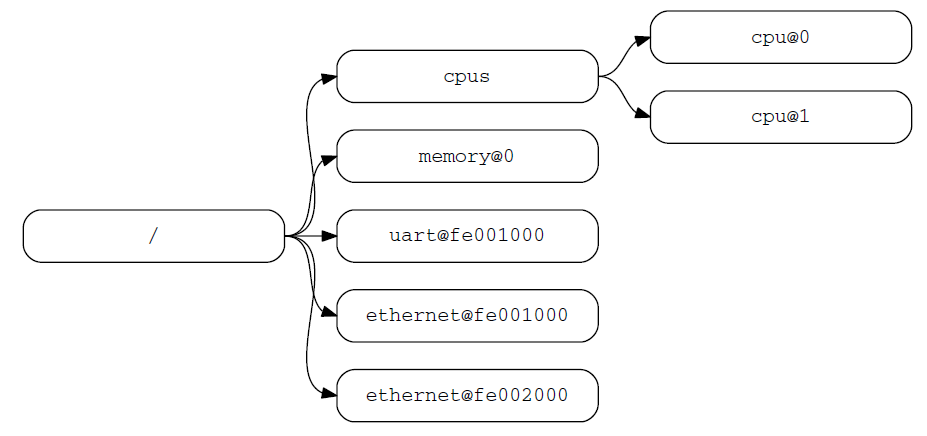


unit-address用户描述节点位于特定类的总线的位置，其由1个或者多个上述表格的字符组成，unit-address必须和节点的reg属性匹配，如果节点没有reg属性，unit-address如果不存在，节点就不会有reg属性，此时node-name就会单独存在，看起来和其他节点会有所不同。对于设备的匹配，一般都会要求指定同时reg和unit-address的格式。根节点没有node-name和unit-address，它由一个分号符/来进行区别和辨识。

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:

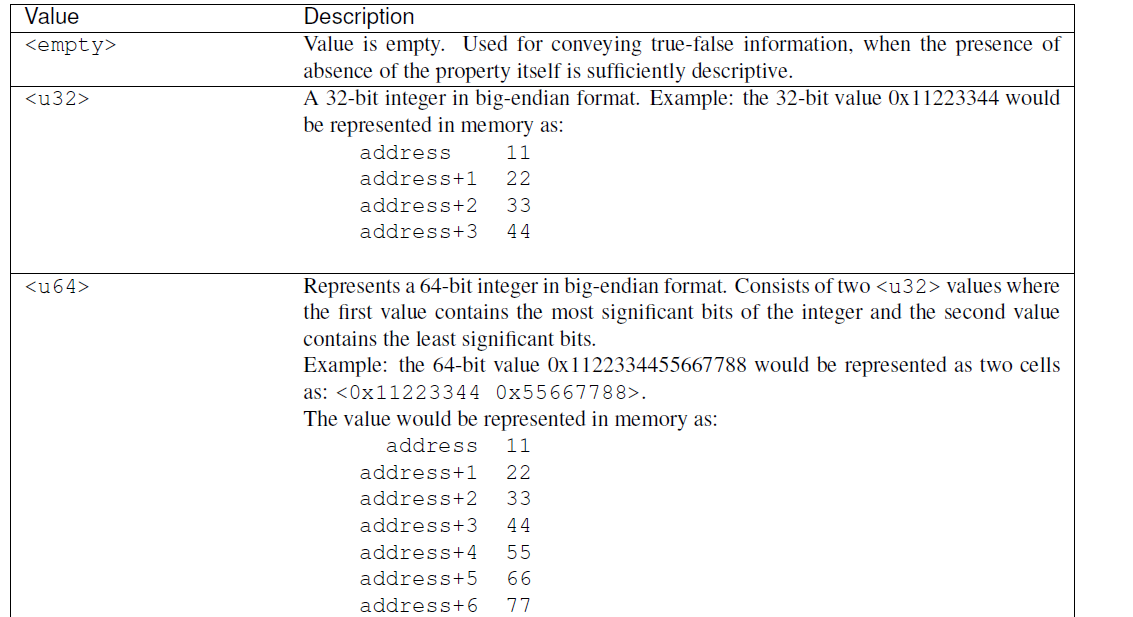
fsl,channel-fifo-len

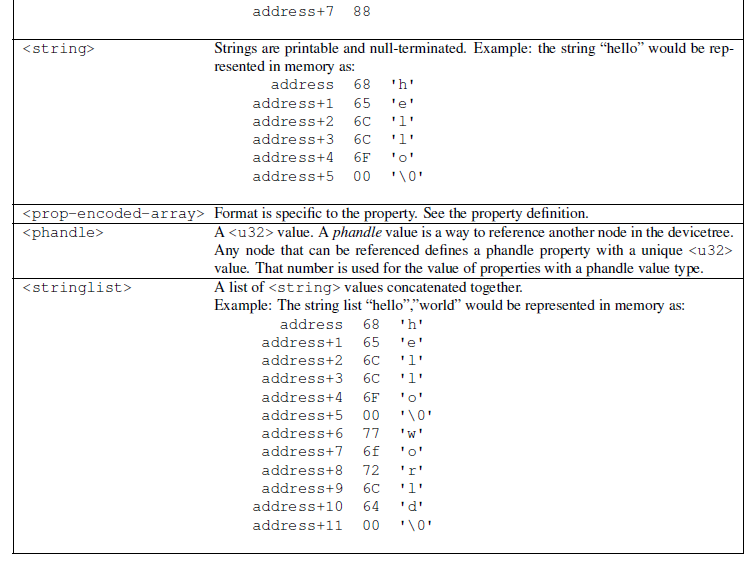
ibm,ppc-interrupt-server#s

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





## 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

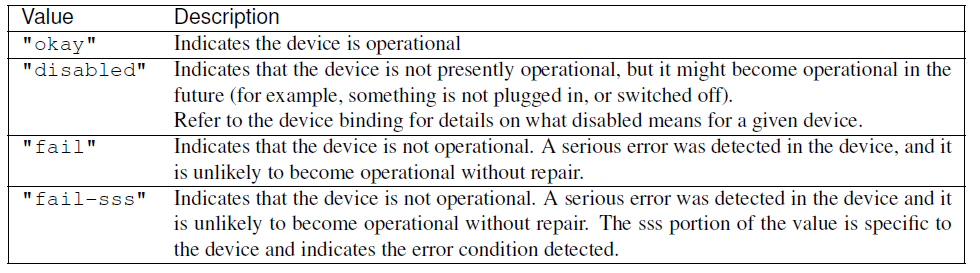
### Status

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



### #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

1. 中断连接器

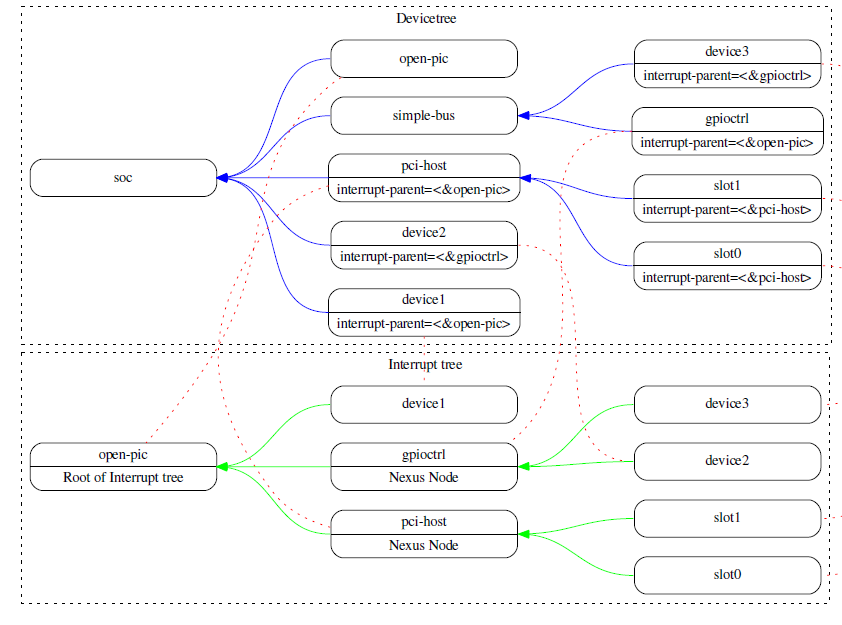
软件节点，他定义了一个中断域到另一个中断域之间的转换，转换过程基于域特定信息和总线信息，域之间的这种转换关系是基于中断映射属性完成的，例如，PIC控制器设备节点可能是中断连接，他定义了中断命名空间到中断控制器硬件之间的转换。寻找中断树的根节点方法为：遍历中断树，直到到达了中断控制器节点，要求此节点没有中断属性，并且没有明确的中断父节点。下图是一个实例，显示了中断的层级关系，它显示了设备的自然结构以及每个节点位于逻辑中的位置。

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，

# 节点要求

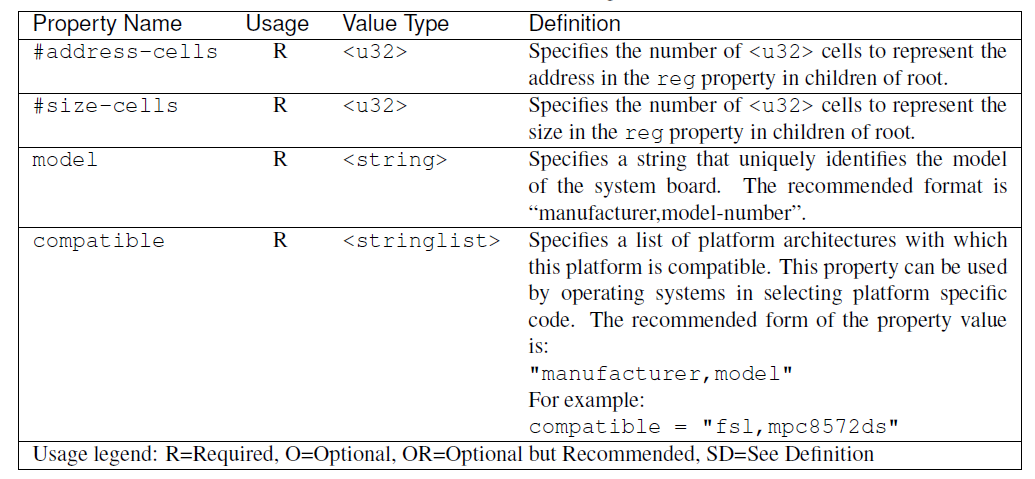
## 基本设备节点类型

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：

1. 一个/cpu节点
2. 最少一个内存节点

## 根节点

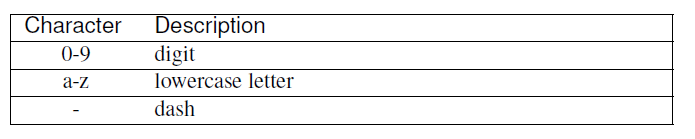
设备树具有一个根节点，其他所有后续节点都是根节点的子节点，根节点的完整路径是/，根节点的所有属性如下表所示:



## /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:



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";

}

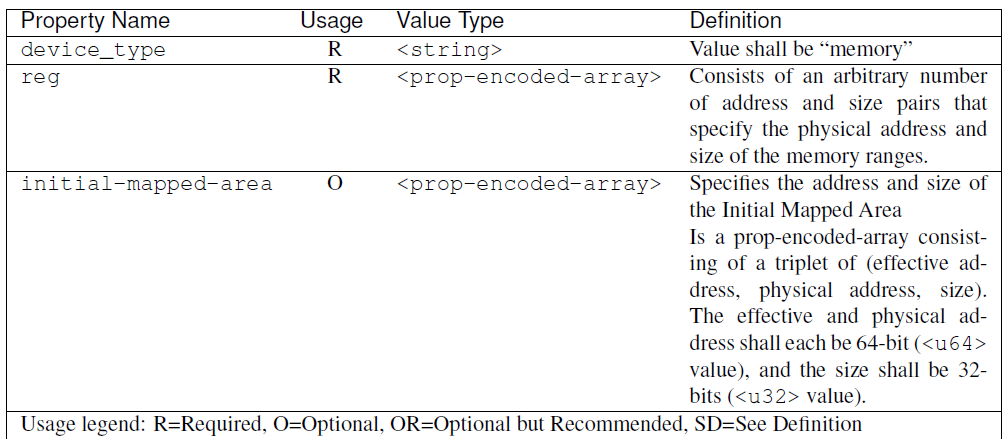
给操作系统传递serial0这个别名，操作系统会检查此别名节点，然后引用其完整路径，也就是/simple-bus@fe000000/serial@llc500这个路径

## /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：

1. 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:



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";

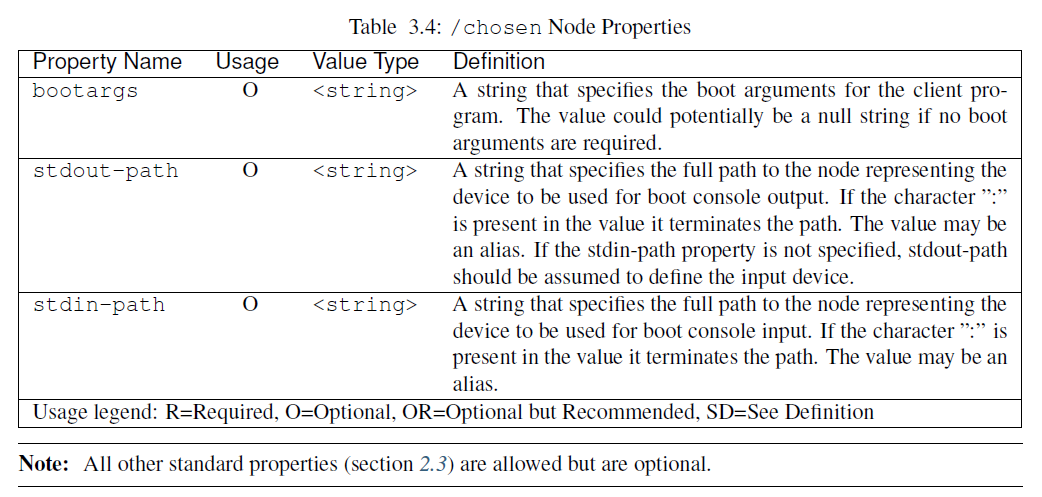
reg = <0x000000001 0x00000000 0x00000001 0x00000000>;

};

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



例子：

chosen {

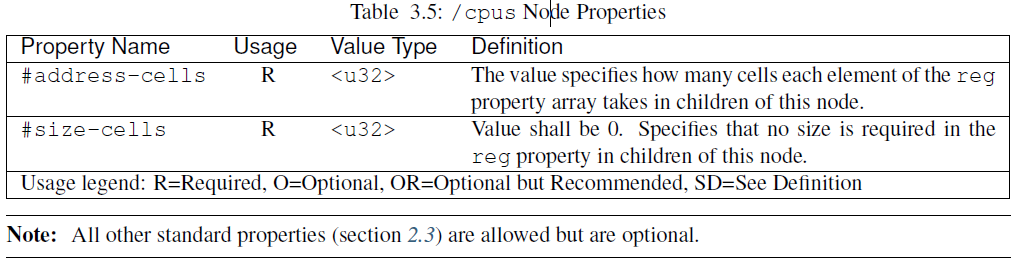
bootargs = "root=/dev/nfs rw nfsroot=192.168.1.1 console=ttyS0,115200";

};

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，



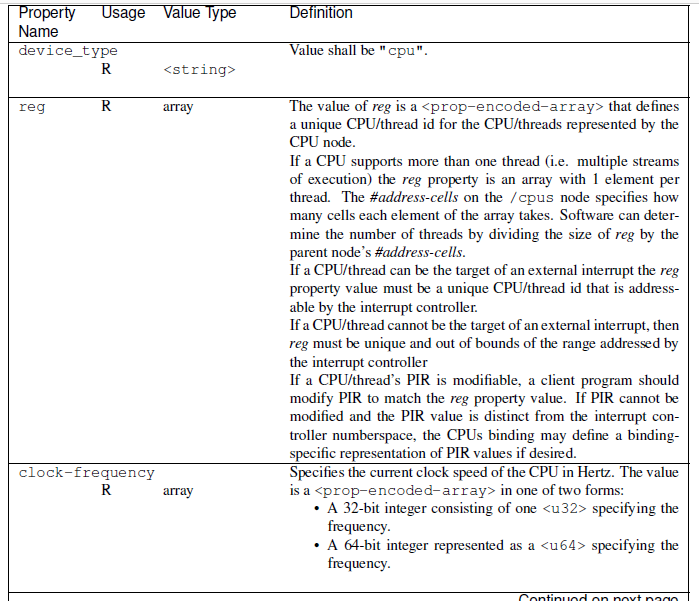
## /cpus/cpu\*节点属性

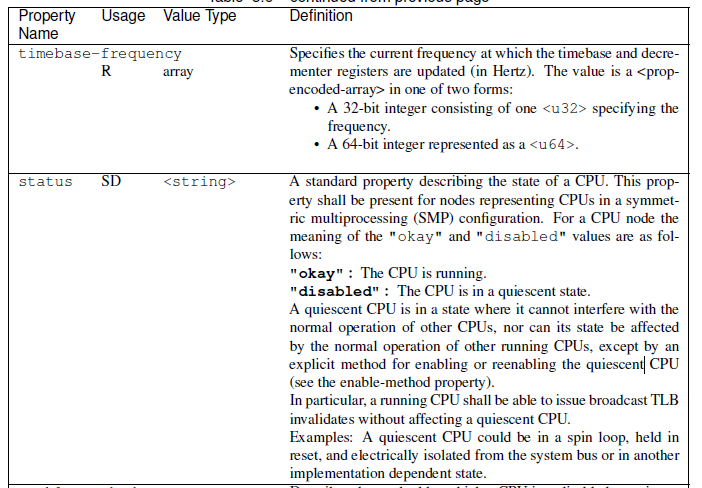
cpu节点代表一个硬件可执行块，它能够运行操作系统，同时也不会干扰可能正在运行其他操作系统的CPU。共享MMU的硬件线程一般会用一个CPU节点表示，如果要设计其他更复杂的CPU，对于CPU的匹配必须有足够清晰的描述，比如哪些县城不共享MMU。CPU和线程通过统一的数字空间进行编号，编号时应尽可能和CPU中断控制器的线程编号匹配。如果cpus节点里面，拥有和CPU节点里面完全相同的属性，此属性可能会被替代。操作系统首先会检查一个特定的CPU节点，如果期望节点不存在，接着我们就必须查找父cpus节点。

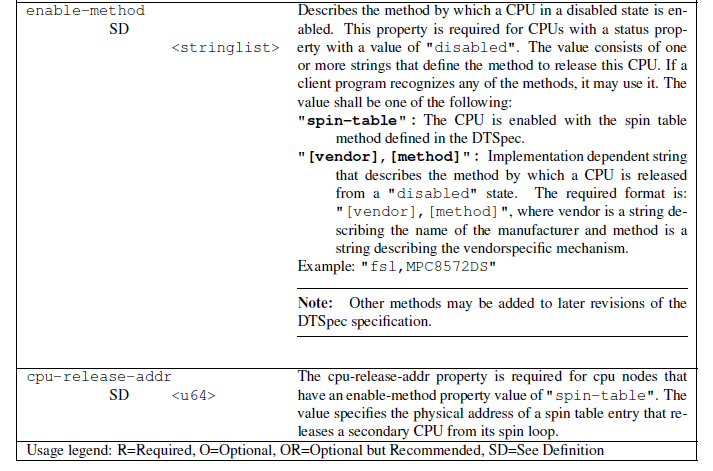
对于每一个cpu而言，节点名字必须是cpu.

### /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







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

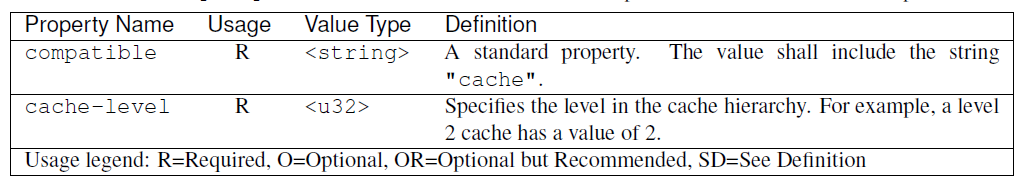
### L1缓存属性

### 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:



### 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

};

};

};

# 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：

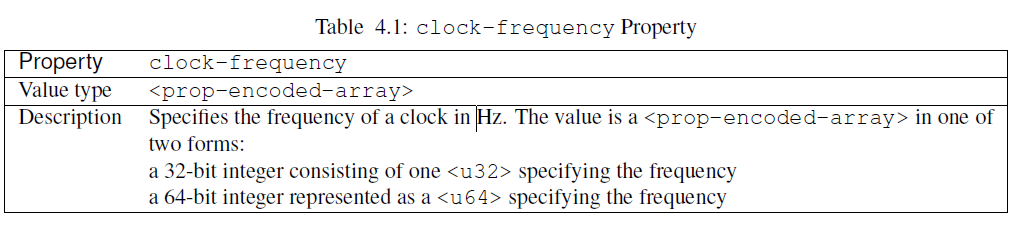
1. 定义一个compatible字符串属性
2. 对于新设备，使用标准属性，一把来说，最少包含有reg和interrupts两个属性值
3. 如果一个新设备符号DTS标准规范，可以直接使用通用属性。
4. 通用属性见下表
5. 为了匹配一个新设备，推荐使用如下格式:

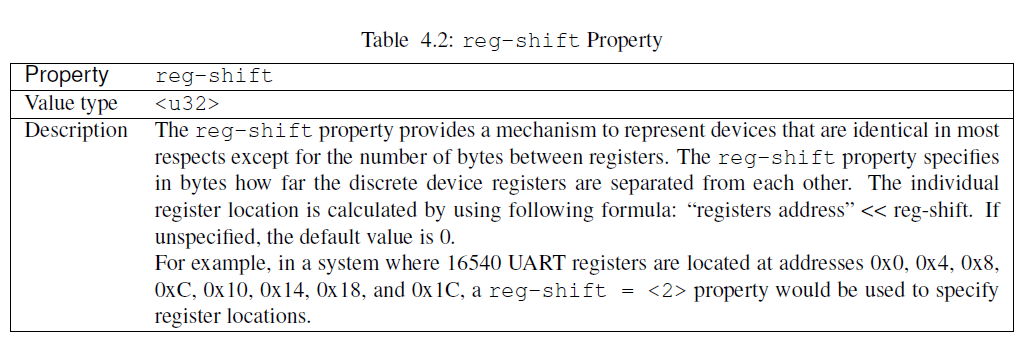
"<company>,<property-name>",其中<company>代表厂商，<property-name>产品名称

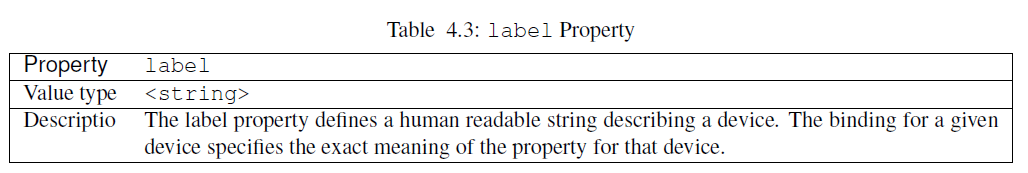
例如:"ibm,ppc-interrupt-server#s"

### 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。



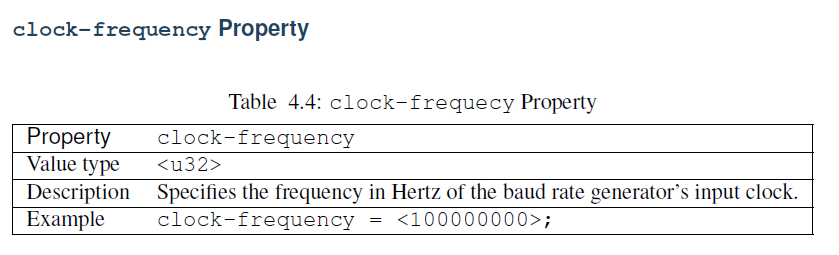


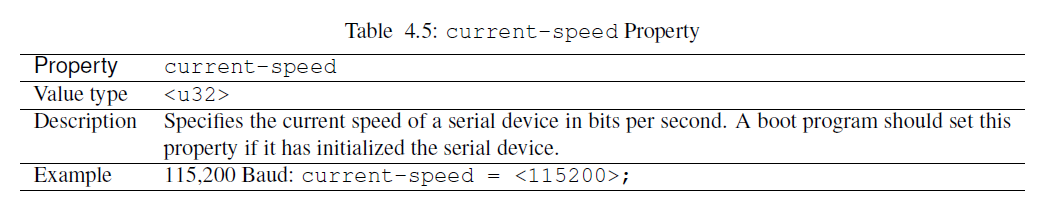


## 串行设备

### 串行类的匹配

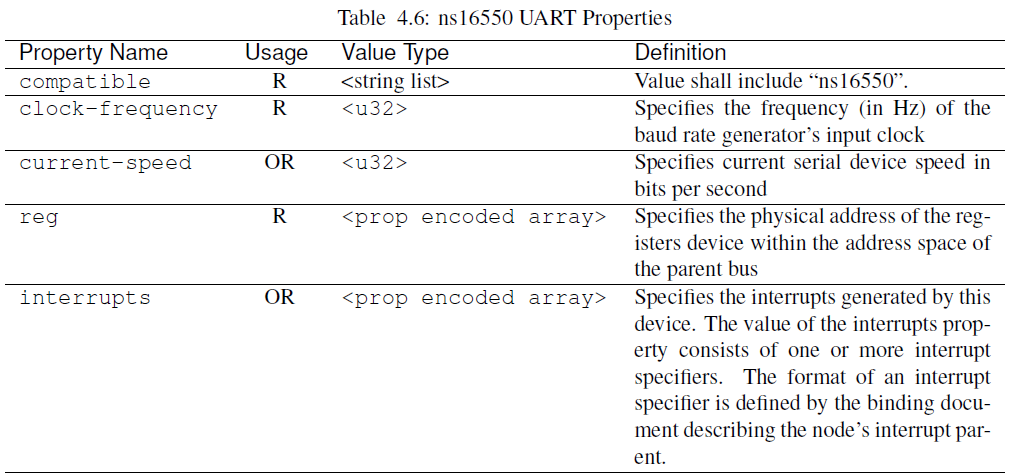
串行设备类有各种各样的点对点的串行设备组成，常用的串行设备有:8250 UART、16550 UART、HDLC、BISYNC，这些设备大多数和RS232串口类具有兼容性。I2C和SPI不应该被当作串行设备，因为他们有他们自己的协议和规范，

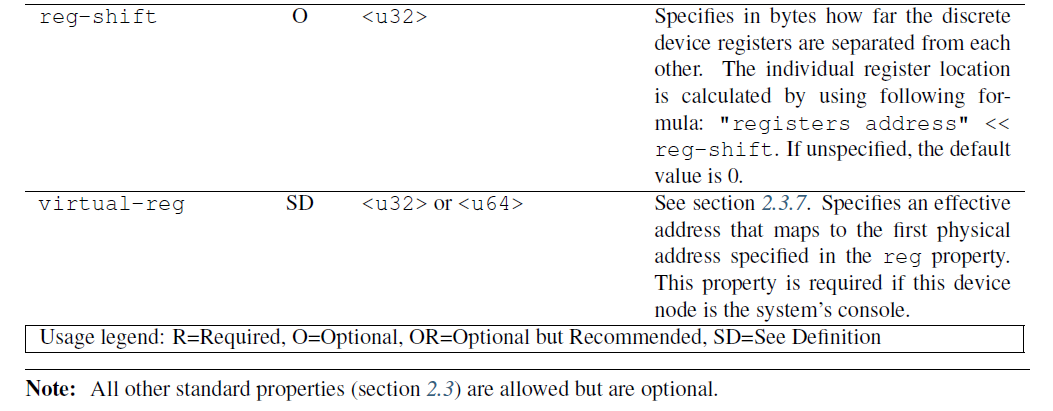




### 国际半导体16450/16550兼容串口要求

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：

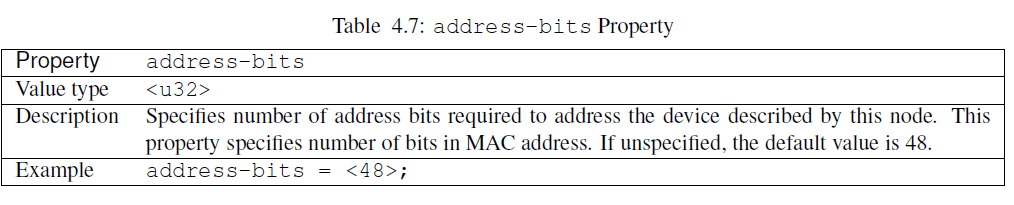


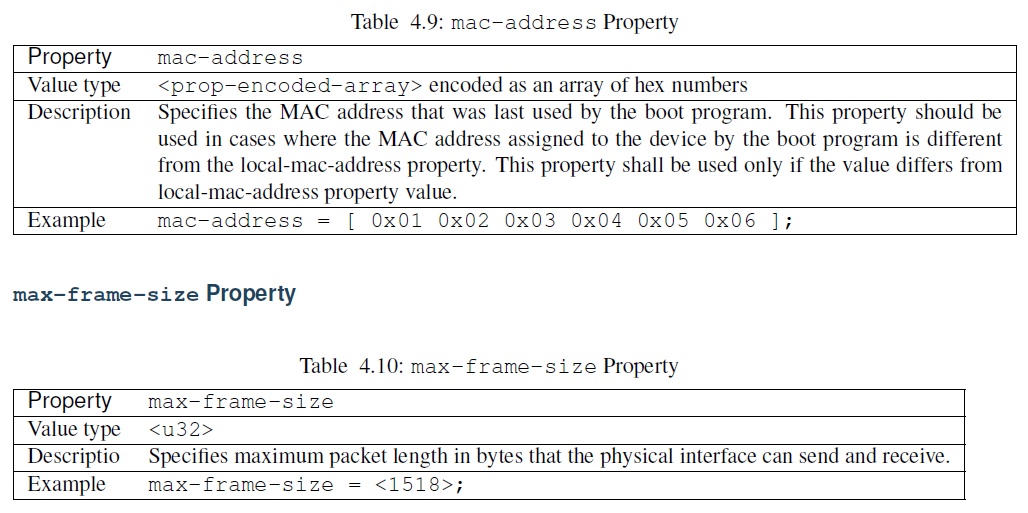


## 网络设备

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

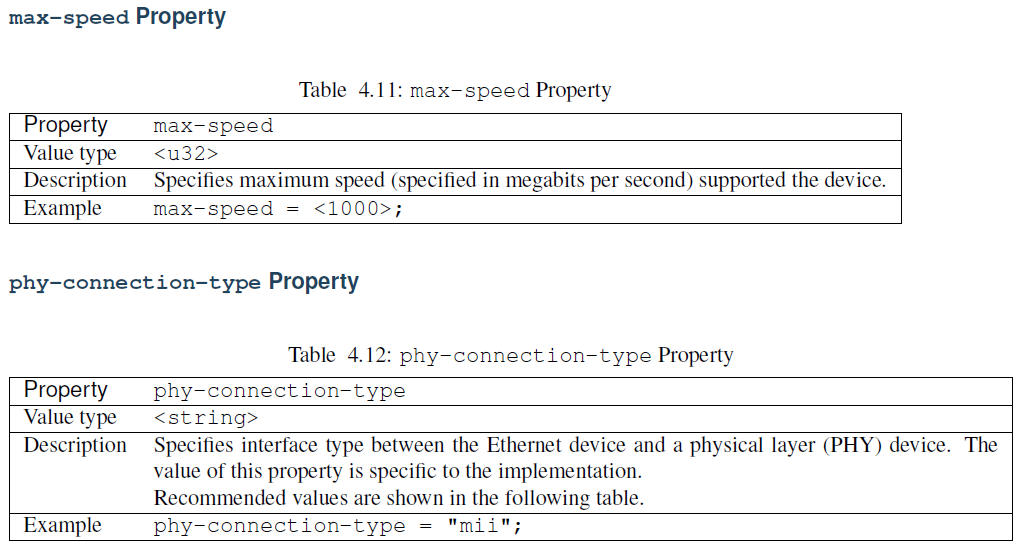
### 网络设备的匹配



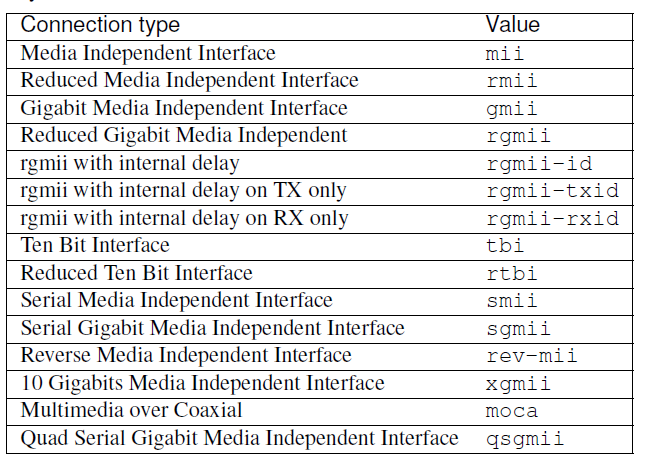


### 网络特别说明

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



下面这些值是phy-connection-type这个属性的可选值，



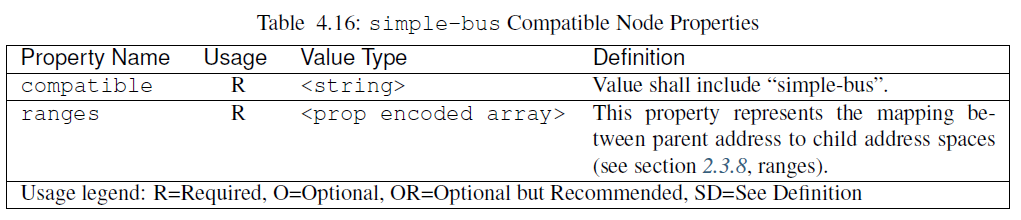
## Power ISA PIC中断控制器

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

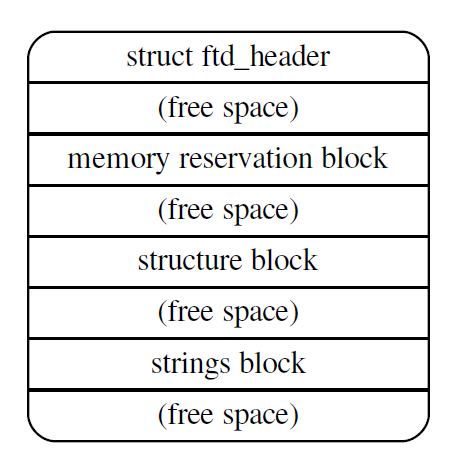
## Simple-bus 兼容值

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。



# 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。

# 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>

1. 使用64位数值代表32位单元

clock-frequency = <0x00000001 0x00000000>;

1. 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";

1. 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];

1. 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";

1. 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} >;

1. 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;

1. 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:/\*\*/ //