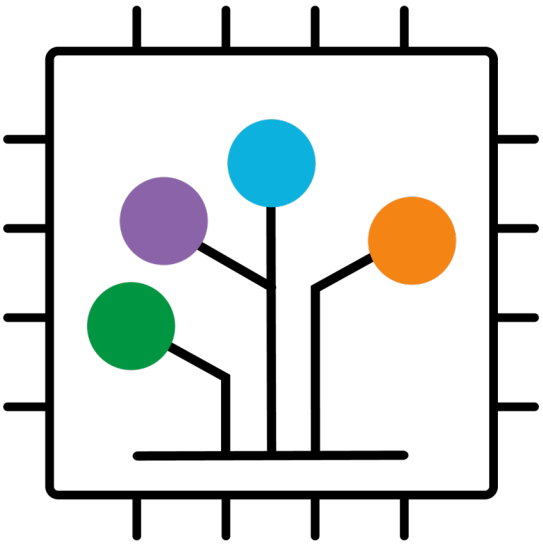

Devicetree Specification

Release v0.2

devicetree.org

20 December 2017

CONTENTS

Copyright

License Information

n. 顺从，服从；符合；屈从；可塑性

Acknowledgements

n. 操纵；指导；掌舵
v. 驾驶；掌舵（steer的ing形式）

n. 综述；评论；复习（review的复数）；复核
v. 回顾；复习（review的三单形式）；检验

The devicetree.org Technical Steering Committee would like to thank the many individuals and companies that contributed to the development of this specification through writing, technical discussions and reviews.

We want to thank the power.org Platform Architecture Technical Subcommittee who developed and published ePAPR. The text of ePAPR was used as the starting point for this document.

Significant aspects of the Devicetree Specification are based on work done by the Open Firmware Working Group which developed bindings for IEEE-1275. We would like to acknowledge their contributions.

We would also like to acknowledge the contribution of the PowerPC and ARM Linux communities that developed and implemented the flattened devicetree concept.

Table 1: Revision History

Revision	Date	Description
0.1	2016-MAY-24	Initial prerelease version. Imported ePAPR text into reStructured Text format and removed Power ISA specific elements.
0.2	2017-DEC-20	• Added more recommended generic node names • Added interrupts-extended • Additional phy times • Filled out detail in source language chapter • Editorial changes • Added changebar version to release documents

CHAPTER

ONE

INTRODUCTION

引导加载程序和管理程序反过来可以将控制加载并传输给操作系统

1.1 Purpose and Scope 目的和范围

n. 幕间剧；幕间休息
vt. 互相影响；互相作用
vi. 互相影响；互相作用

n. 超级监督者；管理程序

To initialize and boot a computer system, various software components interact. Firmware might perform low-level initialization of the system hardware before passing control to software such as an operating system, bootloader, or hypervisor. Bootloaders and hypervisors can, in turn, load and transfer control to operating systems. Standard, consistent interfaces and conventions facilitate the interactions between these software components. 标准、一致的接口和约定促进了这些软件组件之间的交互 In this document the term *boot program* is used to generically refer to a software component that initializes the system state and executes another software component referred to as a *client program*. Examples of a boot program include: firmware, bootloaders, and hypervisors. Examples of a client program include: bootloaders, hypervisors, operating systems, and special purpose programs. A piece of software may be both a client program and a boot program (e.g. a hypervisor).

客户端程序　　一个引导程序

This specification, the Devicetree Specification (DTSpec), provides a complete boot program to client program interface definition, combined with minimum system requirements that facilitate the development of a wide variety of systems.

vt. 促进；帮助；使容易

This specification is targeted towards the requirements of embedded systems. An embedded system typically consists of system hardware, an operating system, and application software that are custom designed to perform a fixed, specific set of tasks. This is unlike general purpose computers, which are designed to be customized by a user with a variety of software and I/O devices. Other characteristics of embedded systems may include:

- a fixed set of I/O devices, possibly highly customized for the application
  一组固定的I/O设备
- a system board optimized for size and cost
- limited user interface
- resource constraints like limited memory and limited nonvolatile storage 有限的非易失存储器
- real-time constraints 实时约束
- use of a wide variety of operating systems, including Linux, real-time operating systems, and custom or proprietary operating systems

adj. 专卖的，专营的；所有的，所有权的；（行为）像所有者那样的，所有人（似）的
n. 所有权，所有人

Organization of this Document

- Chapter *1* introduces the architecture being specified by DTSpec.
- Chapter *2* introduces the devicetree concept and describes its logical structure and standard properties.
- Chapter *3* specifies the definition of a base set of device nodes required by DTSpec-compliant devicetrees.
  指定符合dtspec的设备树所需的设备节点的基本集的定义
- Chapter *4* describes device bindings for certain classes of devices and specific device types.
  描述某些设备类别和特定设备类型的设备绑定
- Chapter *5* specifies the DTB encoding of devicetrees.
  指定设备树的DTB编码
- Chapter *6* specifies the DTS source language.
  指定DTS源语言

bindings
n. 捆绑；捆缚物；镶边；粘合剂（binding的复数

Conventions Used in this Document 本文档中使用的约定

The word *shall* is used to indicate mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).

“应”一词用来表示为了符合标准而必须严格遵守的强制性要求，并且不允许有任何偏差(“应”等于“要求”)。

The word *should* is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (*should* equals *is recommended that*).

在标准范围内允许采取的行动(也许等于是允许的)。

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted*).

vt. 表明；指出；预示；象征

Examples of devicetree constructs are frequently shown in *Devicetree Syntax* form. See section *6* for an overview of this syntax.

n. 构念（construct的复数）；建筑物；构图
v. 设计（construct的三单形式）；建造

1.2 Relationship to IEEE™ 1275 and ePAPR

adv. 宽松地；放荡地；轻率地

DTSpec is loosely related to the IEEE 1275 Open Firmware standard—*IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices [IEEE1275]*.

The original IEEE 1275 specification and its derivatives such as CHRP *[CHRP]* and PAPR *[PAPR]* address problems of general purpose computers, such as how a single version of an operating system can work on several different computers within the same family and the problem of loading an operating system from user-installed I/O devices.

Because of the nature of embedded systems, some of these problems faced by open, general purpose computers do not apply. Notable features of the IEEE 1275 specification that are omitted from the DTSpec include:

- Plug-in device drivers
- FCode
- The programmable Open Firmware user interface based on Forth
- FCode debugging
- Operating system debugging

v. 保留；保存（retain的过去式形式）
adj. 保留的

n. 概念，观念；思想（concept复数形式）

What is retained from IEEE 1275 are concepts from the devicetree architecture by which a boot program can describe and communicate system hardware information to client program, thus eliminating the need for the client program to have hard-coded descriptions of system hardware.

本规范部分取代了ePAPR [ePAPR]规范

This specification partially supersedes the ePAPR [EPAPR] specification. ePAPR documents how devicetree is used by the Power ISA, and covers both general concepts, as well as Power ISA specific bindings. The text of this document was derived from ePAPR, but either removes architecture specific bindings, or moves them into an appendix.

adj. 导出的；衍生的，派生的
v. 从……衍生出，源于；（从……中）得到，提取；导出（derive 的过去式和过去分词）

n. 附录；阑尾；附加物

1.3 32-bit and 64-bit Support

The DTSpec supports CPUs with both 32-bit and 64-bit addressing capabilities. Where applicable, sections of the DTSpec describe any requirements or considerations for 32-bit and 64-bit addressing.

1.4 Definition of Terms

adj. 不对称的；非对称的　n. 多重处理

AMP Asymmetric Multiprocessing. Computer available CPUs are partitioned into groups, each running a distinct operating system image. The CPUs may or not may not identical.

boot CPU The first CPU which a boot program directs to a client program's entry point.

Book III-E Embedded Environment. Section of the Power ISA defining supervisor instructions and related facilities used in embedded Power processor implementations.

参考；涉及；指的是；适用于

boot program Used to generically refer to a software component that initializes the system state and executes another software component referred to as a client program. Examples of a boot program include: firmware, bootloaders, and hypervisors.

n. 超级监督者；管理程序

引导装载程序

client program Program that typically contains application or operating system software. Examples of a client program include: bootloaders, hypervisors, operating systems, and special purpose programs.

由32位组成的信息单位

cell A unit of information consisting of 32 bits.

DMA Direct memory access

n. 一滴；一抹；难以名状的一团
vt. 弄脏；把……做错
vi. 得零分；弄错

DTB Devicetree blob. Compact binary representation of the devicetree.

DTC Devicetree compiler. An open source tool used to create DTB files from DTS files.

DTS Devicetree syntax. A textual representation of a devicetree consumed by the DTC. See Appendix A Devicetree Source Format (version 1).

DTC使用的设备树的文本表示。参见附录A Devicetree来源格式(版本1)

effective address Memory address as computed by processor storage access or branch instruction.

由处理器存储、访问或转移指令计算的内存地址

physical address Address used by the processor to access external device, typically a memory controller.

处理器用来访问外部设备(通常是内存控制器)的地址。

Power ISA Power Instruction Set Architecture.

电源指令集体系结构

中断说明符

interrupt specifier A property value that describes an interrupt. Typically information that specifies an interrupt number and sensitivity and triggering mechanism is included.

secondary CPU CPUs other than the boot CPU that belong to the client program are considered *secondary CPUs*.

属于客户机程序的引导CPU以外的CPU被认为是辅助CPU

SMP Symmetric multiprocessing. A computer architecture where two or more identical CPUs can share memory and IO and operate under a single operating system.

SoC System on a chip. A single computer chip integrating one or more CPU core as well as number of other peripherals.

unit address The part of a node name specifying the node's address in the address space of the parent node.

节点名称中指定父节点地址空间中的节点地址的部分

adj. 静止的；不活动的；沉寂的

quiescent CPU 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 re-enabling the quiescent CPU.

[数] 显式方法

CHAPTER

TWO

THE DEVICETREE

2.1 Overview

DTSpec指定了一个称为devicetree的构造来描述系统硬件。引导程序将devicetree加载到客户机程序的内存中，并将指向devicetree的指针传递给客户机

DTSpec specifies a construct called a *devicetree* to describe system hardware. A boot program loads a devicetree into a client program's memory and passes a pointer to the devicetree to the client.

指定用于描述设备节点的基本属性集

This chapter describes the logical structure of the devicetree and specifies a base set of properties for use in describing device nodes. Chapter *3* specifies certain device nodes required by a DTSpec-compliant devicetree. Chapter 6 describes the DTSpec-defined device bindings—the requirements for representing certain device types or classes of devices. Chapter 8 describes the in-memory encoding of the devicetree.

A devicetree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent.

探测和检测附加的设备

A DTSpec-compliant devicetree describes device information in a system that cannot necessarily be dynamically detected by a client program. For example, the architecture of PCI enables a client to probe and detect attached devices, and thus devicetree nodes describing PCI devices might not be required. However, a device node is required to describe a PCI host bridge device in the system if it cannot be detected by probing.

一个设备节点被要求在系统中描述一个PCI主机桥接设备，如果它不能被探测检测

Example

Fig. 2.1 shows an example representation of a simple devicetree that is nearly complete enough to boot a simple operating system, with the platform type, CPU, memory and a single UART described. Device nodes are shown with properties and values inside each node.

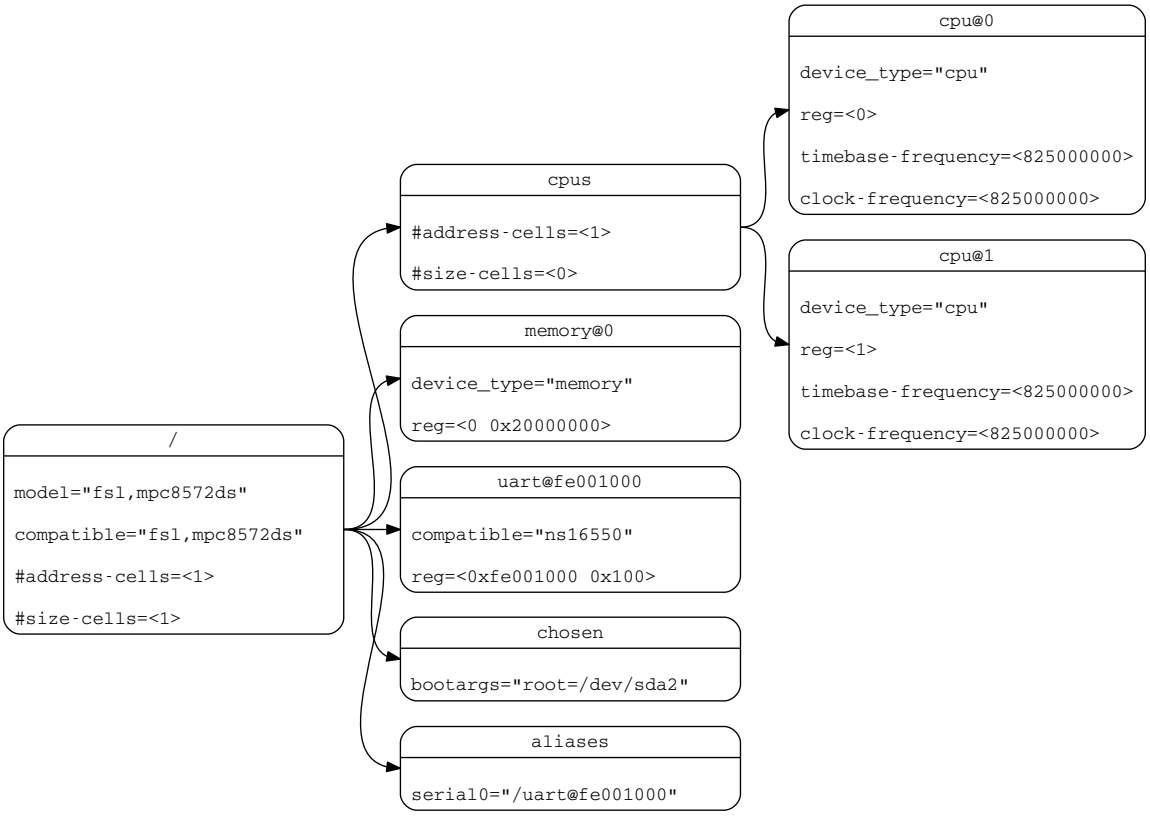

Fig. 2.1: Devicetree Example

2.2 Devicetree Structure and Conventions

2.2.1 Node Names

Node Name Requirements

n. 大会；[法] 惯例；[计] 约定；[法] 协定；习俗

Each node in the devicetree is named according to the following convention:

```
node-name@unit-address
```

The *node-name* component specifies the name of the node. It shall be 1 to 31 characters in length and consist solely of characters from the set of characters in Table 2.1.

adj. 组成的；构成的
n. 组成部分；成分；组件，元件

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

n. 周期，期间；时期；一段时间；经期；课时；句点，句号
adj. 某一时代的

The *node-name* shall start with a lower or uppercase character and should describe the general class of device.

The *unit-address* component of the name is specific to the bus type on which the node sits. It consists of one or more ASCII characters from the set of characters in Table 2.1. The unit-address must match the first address specified in the *reg* property of the node. If the node has no *reg* property, the *@unit-address* must be omitted and the *node-name* alone differentiates the node from other nodes at the same level in the tree. The binding for a particular bus may specify additional, more specific requirements for the format of *reg* and the *unit-address*.

节点位置

特定总线的绑定可以为reg和单元地址的格式指定额外的、更具体的要求。

The root node does not have a node-name or unit-address. It is identified by a forward slash (/).

它由正斜杠标识

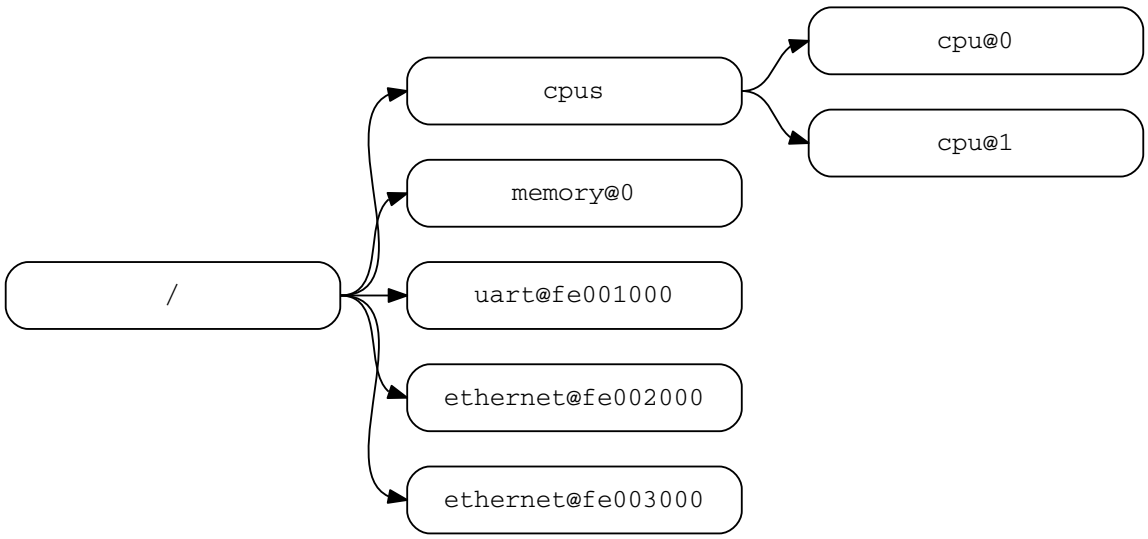

Fig. 2.2: Examples of Node Names

In Fig. 2.2:

- The nodes with the name `cpu` are distinguished by their unit-address values of 0 and 1.
- The nodes with the name `ethernet` are distinguished by their unit-address values of `fe002000` and `fe003000`.

2.2.2 Generic Names Recommendation

adj. 精确的；明确的；严格的

The name of a node should be somewhat generic, reflecting the function of the device and not its precise programming model. If appropriate, the name should be one of the following choices:

adj. 适当的；恰当的；合适的
vt. 占用，拨出

- adc
- accelerometer
- atm
- audio-codec
- audio-controller
- backlight
- bluetooth
- bus
- cache-controller
- camera
- can
- charger
- clock
- clock-controller
- compact-flash
- can
- cpu
- cpus
- crypto
- disk
- display
- dma-controller
- dsp
- eeprom
- efuse
- endpoint
- ethernet
- ethernet-phy
- fdc
- flash
- gpio
- gpu
- gyrometer
- hdmi
- i2c
- ide
- interrupt-controller
- isa
- keyboard
- key
- keys
- lcd-controller
- led
- leds
- led-controller
- light-sensor
- magnetometer
- mailbox
- mdio
- memory
- memory-controller
- mmc
- mmc-slot
- mouse
- nand-controller
- nvram
- oscillator
- parallel

- pc-card
- pci
- pcie
- phy
- pinctrl
- pmic
- pmu
- port
- ports
- pwm
- regulator
- reset-controller
- rtc
- sata
- scsi
- serial
- sound
- spi
- sram-controller
- ssi-controller
- syscon
- temperature-sensor
- timer
- touchscreen
- usb
- usb-hub
- usb-phy
- video-codec
- vme
- watchdog
- wifi

2.2.3 Path Names

n. 后裔，子孙；（由过去类似物发展来的）派生物；（机器等）后继型产品
adj. 下降的；祖传的

A node in the devicetree can be uniquely identified by specifying the full path from the root node, through all descendant nodes, to the desired node.

The convention for specifying a device path is:

```
/node-name-1/node-name-2/node-name-N
```

For example, in Fig. 2.2, the device path to cpu #1 would be:

```
/cpus/cpu@1
```

The path to the root node is /.

v. 遗漏，省略（omit的过去分词）
adj. 省略了的；省去的

A unit address may be omitted if the full path to the node is unambiguous.

adj. 不含糊的；清楚的；明白的

If a client program encounters an ambiguous path, its behavior is undefined.

n. 遭遇战（encounter的复数）；相见
v. 遭遇（encounter的第三人称单数）；邂逅

2.2.4 Properties

Each node in the devicetree has properties that describe the characteristics of the node. Properties consist of a name and a value.

Property Names

Property names are strings of 1 to 31 characters from the characters show in Table 2.2

Table 2.2: Valid characters for property names

Character	Description
0-9	digit
a-z	lowercase letter
A-Z	uppercase letter
,	comma
.	period
_	underscore
+	plus sign
?	question mark
#	hash

Nonstandard property names should specify a unique string prefix, such as a stock ticker symbol, identifying the name of the company or organization that defined the property. Examples:

```
fsl,channel-fifo-len
ibm,ppc-interrupt-server#s
linux,network-index
```

Property Values

v. 联系（associate的过去式和过去分词）
adj. 关联的；联合的

A property value is an array of zero or more bytes that contain information associated with the property.

Properties might have an empty value if conveying true-false information. In this case, the presence or absence of the property is sufficiently descriptive. 如果传递真-假信息，属性可能具有空值。在这种情况下，属性的存在或不存在就足够说明了

Table 2.3 describes the set of basic value types defined by the DTSpec.

Table 2.3: Property values

n. 输送，传输
v. 传达（信息），表达（思想）；运输；转让（财产）
（convey 的现在分词）

Value	Description
<empty>	Value is empty. Used for conveying true-false information, when the presence of absence of the property itself is sufficiently descriptive. 充分说明这一点
<u32> n. 整数	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

Continued on next page

Table 2.3 – continued from previous page

Value	Description
<string>	Strings are printable and null-terminated. Example: the string “hello” would be represented in memory as: 字符串可打印并以null结尾。示例:字符串“hello”在内存中表示为: 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. 串级 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'

2.3 Standard Properties

可以指定关于标准特性使用的附加要求或约束条件

DTSpec specifies a set of standard properties for device nodes. These properties are described in detail in this section. Device nodes defined by DTSpec (see Chapter *3*) may specify additional requirements or constraints regarding the use of the standard properties. Chapter *4* describes the representation of specific devices and may also specify additional requirements. 描述特定装置的描述，也可以说明附加要求。

Note: All examples of devicetree nodes in this document use the DTS (Devicetree Source) format for specifying nodes and properties.

2.3.1 compatible

adj. 兼容的；能共处的；可并立的

Property name: compatible

Value type: <stringlist>

adj. 一般的，普通的；综合的；大体的
n. 一般；将军，上将；常规
n. (General)人名；(英)杰纳勒尔

Description: n. 性质，性能；财产；所有权

The *compatible* property value consists of one or more strings that define the specific programming model for the device. This list of strings should be used by a client program for device driver selection. The property value consists of a concatenated list of null terminated strings, from most specific to most general. They allow a device to express its compatibility with a family of similar devices, potentially allowing a single device driver to match against several devices.

The recommended format is "manufacturer,model", where manufacturer is a string describing the name of the manufacturer (such as a stock ticker symbol), and model specifies the model number.

Example:

```
compatible = "fsl,mpc8641", "ns16550";
```

In this example, an operating system would first try to locate a device driver that supported fsl,mpc8641. If a driver was not found, it would then try to locate a driver that supported the more general ns16550 device type.

2.3.2 model

Property name: `model`

Value type: `<string>`

Description:

The model property value is a `<string>` that specifies the manufacturer's model number of the device.

The recommended format is: `"manufacturer,model"`, where `manufacturer` is a string describing the name of the manufacturer (such as a stock ticker symbol), and model specifies the model number.

Example:

```
model = "fsl,MPC8349EMITX";
```

2.3.3 phandle

Property name: `phandle`

Value type: `<u32>`

Description:

数字标识符

The *phandle* property specifies a numerical identifier for a node that is unique within the devicetree. The *phandle* property value is used by other nodes that need to refer to the node associated with the property.

Example: phandle属性值由需要引用与该属性关联的节点的其他节点使用

v. 联系（associate的过去式和过去分词）
adj. 关联的；联合的

See the following devicetree excerpt:

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

A *phandle* value of 1 is defined. Another device node could reference the pic node with a phandle value of 1:

```
another-device-node {
  interrupt-parent = <1>;
};
```

Note: Older versions of devicetrees may be encountered that contain a deprecated form of this property called `linux,phandle`. For compatibility, a client program might want to support `linux,phandle` if a `phandle` property is not present. The meaning and use of the two properties is identical.

---

adj. 明确的；清楚的；直率的；详述的

vi. 含有；自制
vt. 包含；控制；容纳；牵制（敌军）

Note: Most devicetrees in DTS (see Appendix A) will not contain explicit phandle properties. The DTC tool automatically inserts the `phandle` properties when the DTS is compiled into the binary DTB format.

2.3.4 status

Property name: `status`

Value type: `<string>`

Description:

The `status` property indicates the operational status of a device. Valid values are listed and defined in Table 2.4.

v. 表明（indicate的第三人称单数形式）；指示，显示

Table 2.4: Values for status property

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.

v. 修理；修复，补救；（使）重归于好；（结伴）去
n. 修理；修补过的部位；具体情况，物质条件；常去；常去的场所

2.3.5 #address-cells and #size-cells

Property name: `#address-cells`, `#size-cells`

Value type: `<u32>`

Description:

n. 层级；等级制度

The *#address-cells* and *#size-cells* properties may be used in any device node that has children in the device-tree hierarchy and describes how child device nodes should be addressed. The *#address-cells* property defines the number of `<u32>` cells used to encode the address field in a child node's *reg* property. The *#size-cells* property defines the number of `<u32>` cells used to encode the size field in a child node's *reg* property.

The *#address-cells* and *#size-cells* properties are not inherited from ancestors in the devicetree. They shall be explicitly defined.

A DTSpec-compliant boot program shall supply *#address-cells* and *#size-cells* on all nodes that have children.

If missing, a client program should assume a default value of 2 for *#address-cells*, and a value of 1 for *#size-cells*.

Example:

See the following devicetree excerpt:

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

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

In this example, the *#address-cells* and *#size-cells* properties of the `soc` node are both set to 1. This setting specifies that one cell is required to represent an address and one cell is required to represent the size of nodes that are children of this node.

The serial device *reg* property necessarily follows this specification set in the parent (`soc`) node—the address is represented by a single cell (0x4600), and the size is represented by a single cell (0x100).

2.3.6 reg

Property name: `reg`

Property value: `<prop-encoded-array>` encoded as an arbitrary number of (*address*, *length*) pairs.

Description:

The *reg* property describes the address of the device's resources within the address space defined by its parent bus. Most commonly this means the offsets and lengths of memory-mapped IO register blocks, but may have a different meaning on some bus types. Addresses in the address space defined by the root node are CPU real addresses.

The value is a <*prop-encoded-array*>, composed of an arbitrary number of pairs of address and length, <*address length*>. The number of <*u32*> cells required to specify the address and length are bus-specific and are specified by the *#address-cells* and *#size-cells* properties in the parent of the device node. If the parent node specifies a value of 0 for *#size-cells*, the length field in the value of *reg* shall be omitted.

Example:

Suppose a device within a system-on-a-chip had two blocks of registers, a 32-byte block at offset 0x3000 in the SOC and a 256-byte block at offset 0xFE00. The *reg* property would be encoded as follows (assuming *#address-cells* and *#size-cells* values of 1):

假设一个系统芯片中的一个设备有两个寄存器块，一个是SOC中偏移0x3000的32字节块，另一个是偏移0xFE00的256字节块。reg属性的编码方式如下(假设#地址单元格和#大小单元格值为1)：

```
reg = <0x3000 0x20 0xFE00 0x100>;
```

2.3.7 virtual-reg

Property name: `virtual-reg`

Value type: `<u32>`

Description:

The *virtual-reg* property specifies an effective address that maps to the first physical address specified in the *reg* property of the device node. This property enables boot programs to provide client programs with virtual-to-physical mappings that have been set up.

2.3.8 ranges

Property name: `ranges`

Value type: `<empty>` or `<prop-encoded-array>` encoded as an arbitrary number of (*child-bus-address*, *parent-bus-address*, *length*) triplets.

Description:

The *ranges* property provides a means of defining a mapping or translation between the address space of the bus (the child address space) and the address space of the bus node's parent (the parent address space).

The format of the value of the *ranges* property is an arbitrary number of triplets of (*child-bus-address*, *parent-bus-address*, *length*)

- The *child-bus-address* is a physical address within the child bus' address space. The number of cells to represent the address is bus dependent and can be determined from the *#address-cells* of this node (the node in which the *ranges* property appears).

  表示地址的单元格数依赖于总线，可以从该节点(出现ranges属性的节点)的#address-cells中确定。

- The *parent-bus-address* is a physical address within the parent bus' address space. The number of cells to represent the parent address is bus dependent and can be determined from the *#address-cells* property of the node that defines the parent's address space.
- The *length* specifies the size of the range in the child's address space. The number of cells to represent the size can be determined from the *#size-cells* of this node (the node in which the *ranges* property appears).

If the property is defined with an `<empty>` value, it specifies that the parent and child address space is identical, and no address translation is required.

If the property is not present in a bus node, it is assumed that no mapping exists between children of the node and the parent address space.

Address Translation Example:

```
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>;
   };
};
```

The `soc` node specifies a *ranges* property of

```
<0x0 0xe0000000 0x00100000>;
```

This property value specifies that for an 1024KB range of address space, a child node addressed at physical 0x0 maps to a parent address of physical 0xe0000000. With this mapping, the `serial` device node can be addressed by a load or store at address 0xe0004600, an offset of 0x4600 (specified in *reg*) plus the 0xe0000000 mapping specified in *ranges*.

2.3.9 dma-ranges

Property name: `dma-ranges`

Value type: `<empty>` or `<prop-encoded-array>` encoded as an arbitrary number of (*child-bus-address*, *parent-bus-address*, *length*) triplets.

Description:

The *dma-ranges* property is used to describe the direct memory access (DMA) structure of a memory-mapped bus whose devicetree parent can be accessed from DMA operations originating from the bus. It provides a means of defining a mapping or translation between the physical address space of the bus and the physical address space of the parent of the bus.

adj. [数] 任意的；武断的；专制的

The format of the value of the *dma-ranges* property is an arbitrary number of triplets of (*child-bus-address*, *parent-bus-address*, *length*). Each triplet specified describes a contiguous DMA address range.

每个指定的三元组描述一个连续的DMA地址范围

- The *child-bus-address* is a physical address within the child bus' address space. The number of cells to represent the address depends on the bus and can be determined from the *#address-cells* of this node (the node in which the *dma-ranges* property appears).
- The *parent-bus-address* is a physical address within the parent bus' address space. The number of cells to represent the parent address is bus dependent and can be determined from the *#address-cells* property of the node that defines the parent's address space.

- The *length* specifies the size of the range in the child's address space. The number of cells to represent the size can be determined from the *#size-cells* of this node (the node in which the dma-ranges property appears).

2.3.10 name (deprecated)

v. 不赞成；弃用；不宜用（deprecate的过去式及过去分词形式）

Property name: `name`

Value type: `<string>`

Description:

vt. 强烈反对，抨击；对……表示不赞成；贬低（deprecate 的过去式和过去分词）

The *name* property is a string specifying the name of the node. This property is deprecated, and its use is not recommended. However, it might be used in older non-DTSpec-compliant devicetrees. Operating system should determine a node's name based on the *node-name* component of the node name (see section 2.2.1).

2.3.11 device_type (deprecated)

cascade

英 /kæˈskeɪd/ 美 /kæˈskeɪd/
n. 小瀑布，瀑布状物；串联
vi. 像瀑布般大量倾泻下来
vi. 像瀑布般悬挂着

technically
英 /ˈteknɪkli/ 美 /ˈteknɪkli/
adv. 技术上；专门地；学术上；工艺上

Property name: `device_type`

Value type: `<string>`

Description:

The *device_type* property was used in IEEE 1275 to describe the device's FCode programming model. Because DTSpec does not have FCode, new use of the property is deprecated, and it should be included only on `cpu` and `memory` nodes for compatibility with IEEE 1275–derived devicetrees.

represent
英 /ˌreprɪˈzent/ 美 /ˌreprɪˈzent/
vt. 代表；表现；描绘；回忆；再赠送
vi. 代表；提出异议

2.4 Interrupts and Interrupt Mapping

在设备树中存在一个逻辑中断树

DTSpec adopts the interrupt tree model of representing interrupts specified in *Open Firmware Recommended Practice: Interrupt Mapping, Version 0.9 [b7]*. Within the devicetree a logical interrupt tree exists that represents the hierarchy and routing of interrupts in the platform hardware. While generically referred to as an interrupt tree it is more technically a directed acyclic graph更严格地说，它是一个有向无环图

adv. 一般地；属类地

The physical wiring of an interrupt source to an interrupt controller is represented in the devicetree with the *interrupt-parent* property. Nodes that represent interrupt-generating devices contain an *interrupt-parent* property which has a *phandle* value that points to the device to which the device's interrupts are routed, typically an interrupt controller. If an interrupt-generating device does not have an *interrupt-parent* property, its interrupt parent is assumed to be its devicetree parent. 如果中断生成设备没有中断父属性，则假定它的中断父设备是它的devicetree父设备。

Each interrupt generating device contains an *interrupts* property with a value describing one or more interrupt sources for that device. Each source is represented with information called an *interrupt specifier*. The format and meaning of an *interrupt specifier* is interrupt domain specific, i.e., it is dependent on properties on the node at the root of its interrupt domain. The *#interrupt-cells* property is used by the root of an interrupt domain to define the number of `<u32>` values needed to encode an interrupt specifier. For example, for an Open PIC interrupt controller, an interrupt-specifer takes two 32-bit values and consists of an interrupt number and level/sense information for the interrupt.

中断域是解释中断说明符的上下文

An interrupt domain is the context in which an interrupt specifier is interpreted. The root of the domain is either (1) an interrupt controller or (2) an interrupt nexus. 一个中断联系

1. An *interrupt controller* is physical device and will need a driver to handle interrupts routed through it. It may also cascade into another interrupt domain. An interrupt controller is specified by the presence of an *interrupt-controller* property on that node in the devicetree. 中断控制器是由devicetree中该节点上的中断控制器属性指定的

它也可以级联到另一个中断域

v. 执行，表演（perform的过去分词形式

2. An *interrupt nexus* defines a translation between one interrupt domain and another. The translation is based on both domain-specific and bus-specific information. This translation between domains is performed with the *interrupt-map* property. For example, a PCI controller device node could be an interrupt nexus that defines a translation from the PCI interrupt namespace (INTA, INTB, etc.) to an interrupt controller with Interrupt Request (IRQ) numbers.

n. 关系；连结，连系

cascade
英 /kæˈskeɪd/ 美 /kæˈskeɪd/
n. 小瀑布，瀑布状物；串联
vi. 像瀑布般大量倾泻下来
vi. 像瀑布般悬挂着

n. [计] 遍历；横越；横断物

The root of the interrupt tree is determined when traversal of the interrupt tree reaches an interrupt controller node without an *interrupts* property and thus no explicit interrupt parent. 因此没有显式的中断父节点

中断树的根是在中断树的遍历到达一个没有interrupts属性的中断控制器节点时确定的，因此没有显式的中断父节点

See Fig. 2.3 for an example of a graphical representation of a devicetree with interrupt parent relationships shown. It shows both the natural structure of the devicetree as well as where each node sits in the logical interrupt tree.

它显示了devicetree的自然结构以及每个节点在逻辑中断树中的位置。

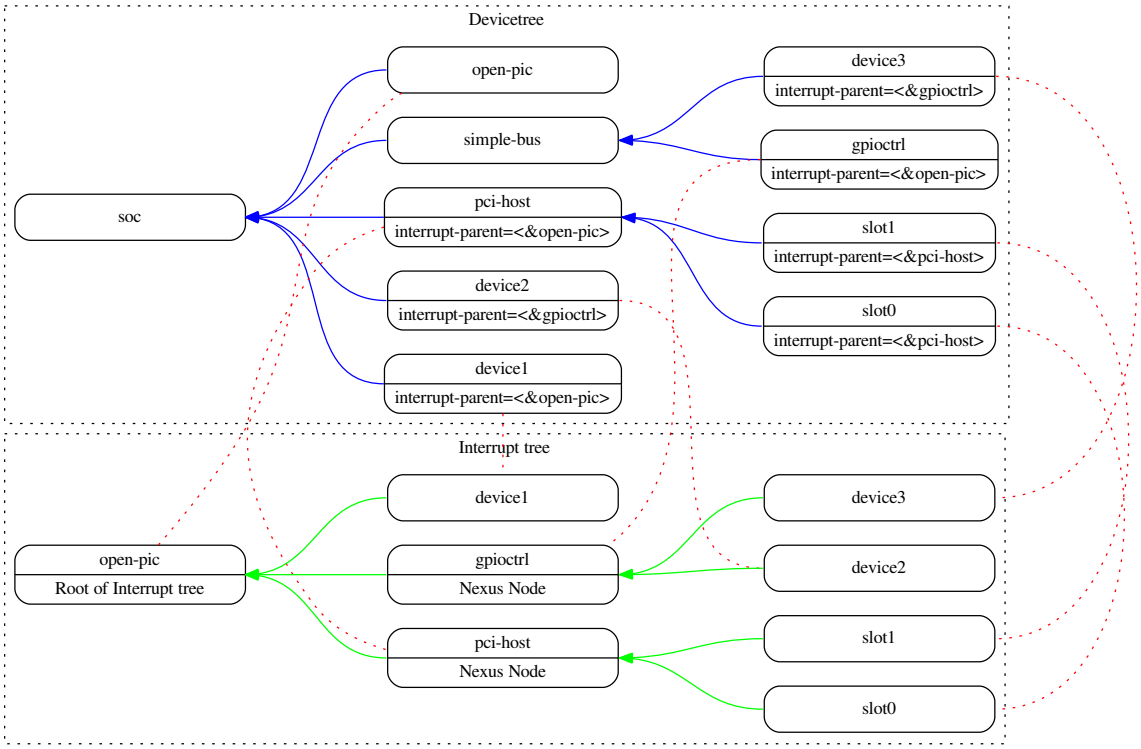

Fig. 2.3: Example of the interrupt tree

In the example shown in Fig. 2.3:

- The `open-pic` interrupt controller is the root of the interrupt tree.
- The interrupt tree root has three children—devices that route their interrupts directly to the `open-pic`
  - device1
  - PCI host controller
  - GPIO Controller
- Three interrupt domains exist; one rooted at the `open-pic` node, one at the `PCI host bridge` node, and one at the `GPIO Controller` node.
- There are two nexus nodes; one at the `PCI host bridge` and one at the `GPIO controller`.

  n. 关系；连结，连系

2.4.1 Properties for Interrupt Generating Devices

interrupts

Property: `interrupts`

adj. [数] 任意的；武断的；专制的

Value type: `<prop-encoded-array>` encoded as arbitrary number of interrupt specifiers

Description:

The *interrupts* property of a device node defines the interrupt or interrupts that are generated by the device. The value of the *interrupts* property consists of an arbitrary number of interrupt specifiers. The format of an interrupt specifier is defined by the binding of the interrupt domain root.

interrupts is overridden by the *interrupts-extended* property and normally only one or the other should be used. 中断由中断扩展属性覆盖，通常只能使用其中一个。

Example: n. [计] 说明符；指示语；[计] 区分符

A common definition of an interrupt specifier in an open PIC–compatible interrupt domain consists of two cells; an interrupt number and level/sense information. See the following example, which defines a single interrupt specifier, with an interrupt number of 0xA and level/sense encoding of 8.

```
interrupts = <0xA 8>;
```

interrupt-parent

Property: `interrupt-parent`

Value type: `<phandle>`

Description:

adj. 明确的；清楚的；直率的；详述的

Because the hierarchy of the nodes in the interrupt tree might not match the devicetree, the *interrupt-parent* property is available to make the definition of an interrupt parent explicit. The value is the phandle to the interrupt parent. If this property is missing from a device, its interrupt parent is assumed to be its devicetree parent.如果某个设备缺少此属性，则假定它的中断父设备是它的devicetree父设备

interrupts-extended 中断扩展

Property: `interrupts-extended`

Value type: `<phandle> <prop-encoded-array>`

Description:

The *interrupts-extended* property lists the interrupt(s) generated by a device. *interrupts-extended* should be used instead of *interrupts* when a device is connected to multiple interrupt controllers as it encodes a parent phandle with each interrupt specifier.

Example:

This example shows how a device with two interrupt outputs connected to two separate interrupt controllers would describe the connection using an *interrupts-extended* property. `pic` is an interrupt controller with an *#interrupt-cells* specifier of 2, while `gic` is an interrupt controller with an *#interrupts-cells* specifier of 1.

```
interrupts-extended = <&pic 0xA 8>, <&gic 0xda>;
```

The *interrupts* and *interrupts-extended* properties are mutually exclusive. A device node should use one or the other, but not both. Using both is only permissible when required for compatibility with software that does not understand *interrupts-extended*. If both *interrupts-extended* and *interrupts* are present then *interrupts-extended* takes precedence.
如果扩展中断和中断同时存在，则扩展中断优先

2.4.2 Properties for Interrupt Controllers

#interrupt-cells n. [细胞] 细胞；单元格（cell的复数）；牢房；小屋 v. 住在牢房中（cell的三单形式）

Property: `#interrupt-cells`

Value type: `<u32>`

Description:

The *#interrupt-cells* property defines the number of cells required to encode an interrupt specifier for an interrupt domain. #interrupt-cells属性定义了为中断域编码中断指示符所需的单元数。

interrupt-controller

Property: `interrupt-controller`

Value type: `<empty>`

Description: n. 存在；出席；参加；风度；仪态

The presence of an *interrupt-controller* property defines a node as an interrupt controller node.

2.4.3 Interrupt Nexus Properties

An interrupt nexus node shall have an *#interrupt-cells* property.

interrupt-map

Property: `interrupt-map`

Value type: `<prop-encoded-array>` encoded as an arbitrary number of interrupt mapping entries.

Description:

一个连接节点

An *interrupt-map* is a property on a nexus node that bridges one interrupt domain with a set of parent interrupt domains and specifies how interrupt specifiers in the child domain are mapped to their respective parent domains.

The interrupt map is a table where each row is a mapping entry consisting of five components: *child unit address*, *child interrupt specifier*, *interrupt-parent*, *parent unit address*, *parent interrupt specifier*.

child unit address The unit address of the child node being mapped. The number of 32-bit cells required to specify this is described by the *#address-cells* property of the bus node on which the child is located.

被映射的子节点的中断说明符

child interrupt specifier The interrupt specifier of the child node being mapped. The number of 32-bit cells required to specify this component is described by the *#interrupt-cells* property of this node—the nexus node containing the *interrupt-map* property.

interrupt-parent A single *<phandle>* value that points to the interrupt parent to which the child domain is being mapped. 单个<phandle>值，该值指向正在映射子域的中断父域

parent unit address The unit address in the domain of the interrupt parent. The number of 32-bit cells required to specify this address is described by the *#address-cells* property of the node pointed to by the interrupt-parent field.

parent interrupt specifier The interrupt specifier in the parent domain. The number of 32-bit cells required to specify this component is described by the *#interrupt-cells* property of the node pointed to by the interrupt-parent field.

Lookups are performed on the interrupt mapping table by matching a unit-address/interrupt specifier pair against the child components in the interrupt-map. Because some fields in the unit interrupt specifier may not be relevant, a mask is applied before the lookup is done. This mask is defined in the *interrupt-map-mask* property (see section *2.4.3.2*).

Note: Both the child node and the interrupt parent node are required to have *#address-cells* and *#interrupt-cells* properties defined. If a unit address component is not required, *#address-cells* shall be explicitly defined to be zero. 如果不需要一个单元地址组件，#address-cells将被显式定义为零。

---

interrupt-map-mask

Property: `interrupt-map-mask`

Value type: `<prop-encoded-array>` encoded as a bit mask

Description:

> An *interrupt-map-mask* property is specified for a nexus node in the interrupt tree. This property specifies a mask that is applied to the incoming unit interrupt specifier being looked up in the table specified in the *interrupt-map* property.

#interrupt-cells

Property: `#interrupt-cells`

Value type: `<u32>`

Description:

> The *#interrupt-cells* property defines the number of cells required to encode an interrupt specifier for an interrupt domain.

2.4.4 Interrupt Mapping Example

The following shows the representation of a fragment of a devicetree with a PCI bus controller and a sample interrupt map for describing the interrupt routing for two PCI slots (IDSEL 0x11,0x12). The INTA, INTB, INTC, and INTD pins for slots 1 and 2 are wired to the Open PIC interrupt controller.

```
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 */
         0x9000 0 0 3 &open-pic 1 1 /* INTC */
         0x9000 0 0 4 &open-pic 2 1 /* INTD */
      >;
   };
};
```

One Open PIC interrupt controller is represented and is identified as an interrupt controller with an *interrupt-controller* property. 一个开放PIC中断控制器被表示为一个带有中断控制器属性的中断控制器

Each row in the interrupt-map table consists of five parts: a child unit address and interrupt specifier, which is mapped to an *interrupt-parent* node with a specified parent unit address and interrupt specifier.

- For example, the first row of the interrupt-map table specifies the mapping for INTA of slot 1. The components of that row are shown here

  child unit address: `0x8800 0 0`
  child interrupt specifier: `1`
  interrupt parent: `&open-pic`
  parent unit address: (empty because `#address-cells = <0>` in the open-pic node)
  parent interrupt specifier: `2 1`

  - The child unit address is `<0x8800 0 0>`. This value is encoded with three 32-bit cells, which is determined by the value of the *#address-cells* property (value of 3) of the PCI controller. The three cells represent the PCI address as described by the binding for the PCI bus.
    - The encoding includes the bus number (0x0 << 16), device number (0x11 << 11), and function number (0x0 << 8).
  - The child interrupt specifier is `<1>`, which specifies INTA as described by the PCI binding. This takes one 32-bit cell as specified by the *#interrupt-cells* property (value of 1) of the PCI controller, which is the child interrupt domain.
  - The interrupt parent is specified by a phandle which points to the interrupt parent of the slot, the Open PIC interrupt controller.
  - The parent has no unit address because the parent interrupt domain (the open-pic node) has an *#address-cells* value of `<0>`.
  - The parent interrupt specifier is `<2 1>`. The number of cells to represent the interrupt specifier (two cells) is determined by the *#interrupt-cells* property on the interrupt parent, the open-pic node.
    - The value `<2 1>` is a value specified by the device binding for the Open PIC interrupt controller (see section *4.5*). The value `<2>` specifies the physical interrupt source number on the interrupt controller to which INTA is wired. The value `<1>` specifies the level/sense encoding.

In this example, the interrupt-map-mask property has a value of `<0xf800 0 0 7>`. This mask is applied to a child unit interrupt specifier before performing a lookup in the *interrupt-map* table.

To perform a lookup of the open-pic interrupt source number for INTB for IDSEL 0x12 (slot 2), function 0x3, the following steps would be performed:

- The child unit address and interrupt specifier form the value `<0x9300 0 0 2>`.
  - The encoding of the address includes the bus number (0x0 << 16), device number (0x12 << 11), and function number (0x3 << 8).
  - The interrupt specifier is 2, which is the encoding for INTB as per the PCI binding.
- The interrupt-map-mask value `<0xf800 0 0 7>` is applied, giving a result of `<0x9000 0 0 2>`.
- That result is looked up in the *interrupt-map* table, which maps to the parent interrupt specifier `<4 1>`.

CHAPTER

THREE

DEVICE NODE REQUIREMENTS

3.1 Base Device Node Types

The sections that follow specify the requirements for the base set of device nodes required in a DTSpec-compliant devicetree.

All devicetrees shall have a root node and the following nodes shall be present at the root of all devicetrees:

- One `/cpus` node
- At least one `/memory` node

3.2 Root node

The devicetree has a single root node of which all other device nodes are descendants. The full path to the root node is `/`.

Table 3.1: Root Node Properties

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			

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

3.3 /aliases node

A devicetree may have an aliases node (/aliases) that defines one or more alias properties. The alias node shall be at the root of the devicetree and have the node name /aliases.

Each property of the /aliases node defines an alias. The property name specifies the alias name. The property value specifies the full path to a node in the devicetree. For example, the property serial0 = "/simple-bus@fe000000/serial@llc500" defines the alias serial0.

Alias names shall be a lowercase text strings of 1 to 31 characters from the following set of characters.

Table 3.2: Valid characters for alias names

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

An alias value is a device path and is encoded as a string. The value represents the full path to a node, but the path does not need to refer to a leaf node.

A client program may use an alias property name to refer to a full device path as all or part of its string value. A client program, when considering a string as a device path, shall detect and use the alias.

Example

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

Given the alias serial0, a client program can look at the /aliases node and determine the alias refers to the device path /simple-bus@fe000000/serial@llc500.

3.4 /memory node

A memory device node is required for all devicetrees and describes the physical memory layout for the system. If a system has multiple ranges of memory, multiple memory nodes can be created, or the ranges can be specified in the *reg* property of a single memory node.

The *unit-name* component of the node name (see section *2.2.1*) shall be memory.

The client program may access memory not covered by any memory reservations (see section *5.3*) using any storage attributes it chooses. However, before changing the storage attributes used to access a real page, the client program is responsible for performing actions required by the architecture and implementation, possibly including flushing the real page from the caches. The boot program is responsible for ensuring that, without taking any action associated with a change in storage attributes, the client program can safely access all memory (including memory covered by memory reservations) as WIMG = 0b001x. That is:

- not Write Through Required
- not Caching Inhibited
- Memory Coherence
- Required either not Guarded or Guarded

If the VLE storage attribute is supported, with VLE=0.

Table 3.3: /memory Node Properties

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			

Note: All other standard properties (section 2.3) are allowed but are optional. 所有其他标准属性(章节2.3)是允许的，但是可选的

Examples

Given a 64-bit Power system with the following physical memory layout: n. 布局；设计；安排；陈列

- RAM: starting address 0x0, length 0x80000000 (2GB)
- RAM: starting address 0x100000000, length 0x100000000 (4GB)

Memory nodes could be defined as follows, assuming #address-cells = <2> and #size-cells = <2>.

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>;
};
```

The reg property is used to define the address and size of the two memory ranges. The 2 GB I/O region is skipped. Note that the #address-cells and #size-cells properties of the root node specify a value of 2, which means that two 32-bit cells are required to define the address and length for the reg property of the memory node.

3.5 /chosen Node

The /chosen node does not represent a real device in the system but describes parameters chosen or specified by the system firmware at run time. It shall be a child of the root node.

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.

Example

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

Older versions of devicetrees may be encountered that contain a deprecated form of the *stdout-path* property called *linux,stdout-path*. For compatibility, a client program might want to support *linux,stdout-path* if a *stdout-path* property is not present. The meaning and use of the two properties is identical.

3.6 /cpus Node Properties

A /cpus node is required for all devicetrees. It does not represent a real device in the system, but acts as a container for child cpu nodes which represent the systems CPUs.

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.

The /cpus node may contain properties that are common across cpu nodes. See section 3.7 for details.

For an example, see section 3.8.1.

3.7 /cpus/cpu* Node Properties

A cpu node represents a hardware execution block that is sufficiently independent that it is capable of running an operating system without interfering with other CPUs possibly running other operating systems.

n. 复合体；综合设施
adj. 复杂的；合成的

Hardware threads that share an MMU would generally be represented under one cpu node. If other more complex CPU topographies are designed, the binding for the CPU must describe the topography (e.g. threads that don't share an MMU).

CPUs and threads are numbered through a unified number-space that should match as closely as possible the interrupt controller's numbering of CPUs/threads.

cpu和线程通过一个统一的数字空间进行编号，这个数字空间应该尽可能接近中断控制器的cpu /线程编号

Properties that have identical values across cpu nodes may be placed in the /cpus node instead. A client program must first examine a specific cpu node, but if an expected property is not found then it should look at the parent /cpus node. This results in a less verbose representation of properties which are identical across all CPUs.

vt. 检查；调查；检测；考试
vi. 检查；调查

这使得所有cpu上相同的属性的表示更简洁

The node name for every CPU node should be cpu.

3.7.1 General Properties of /cpus/cpu* nodes

The following table describes the general properties of cpu nodes. Some of the properties described in Table 3.6 are select standard properties with specific applicable detail. 表3.6中描述的一些属性是精选的标准属性，具有特定的适用细节

Table 3.6: /cpus/cpu* Node General Properties

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 (pending interrupt register) 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 number space, the CPUs binding may define a binding-specific representation of PIR values if desired.

Continued on next page

Table 3.6 – continued from previous page

Property Name	Usage	Value Type	Definition
`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.
`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 re-enabling 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.

Continued on next page

一个静止的CPU可以在一个自旋循环中，
保持在重置中，并从系统总线
或在另一个实现依赖的状态中电气隔离。

Table 3.6 – continued from previous page

Property Name	Usage	Value Type	Definition
`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 vendor specific 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			

vendor是描述制造商名称的字符串，而method是描述供应商特定机制的字符串

该值指定从自旋循环中释放辅助CPU的自旋表条目的物理地址。

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

Table 3.7: `/cpus/cpu*` Node Power ISA Properties

Property Name	Usage	Value Type	Definition
`power-isa-version`	O	`<string>`	A string that specifies the numerical portion of the Power ISA version string. For example, for an implementation complying with Power ISA Version 2.06, the value of this property would be `"2.06"`.

Continued on next page

Table 3.7 – continued from previous page

Property Name	Usage	Value Type	Definition
power-isa-*	O	<empty>	If the power-isa-version property exists, then for each category from the Categories section of Book I of the Power ISA version indicated, the existence of a property named power-isa-[CAT], where [CAT] is the abbreviated category name with all uppercase letters converted to lowercase, indicates that the category is supported by the implementation. For example, if the power-isa-version property exists and its value is "2.06" and the power-isa-e.hv property exists, then the implementation supports [Category:Embedded.Hypervisor] as defined in Power ISA Version 2.06.
cache-op-block-size	SD	<u32>	Specifies the block size in bytes upon which cache block instructions operate (e.g., dcbz). Required if different than the L1 cache block size.
reservation-granule-size	SD	<u32>	Specifies the reservation granule size supported by this processor in bytes.
mmu-type	O	<string>	Specifies the CPU's MMU type. Valid values are shown below: • "mpc8xx" • "ppc40x" • "ppc440" • "ppc476" • "power-embedded" • "powerpc-classic" • "power-server-stab" • "power-server-slb" • "none"
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.

v. 遇到；曾遭遇（encounter的过去式）

Older versions of devicetree may be encountered that contain a bus-frequency property on CPU nodes. For compatibility, a client-program might want to support bus-frequency. The format of the value is identical to that of clock-frequency. The recommended practice is to represent the frequency of a bus on the bus node using a clock-frequency property.

n. 实践；练习；惯例
vt. 练习；实习；实行
vi. 练习；实习；实行

3.7.2 TLB Properties

The following properties of a cpu node describe the translate look-aside buffer in the processor's MMU.

Table 3.8: /cpu/cpu* Node Power ISA TLB Properties

Property Name	Usage	Value Type	Definition
tlb-split n. 劈开；裂缝 adj. 劈开的 vt. 分离；使分离；劈开；离开；分解 vi. 离开；被劈开；断绝关系	SD	<empty>	If present specifies that the TLB has a split configuration, with separate TLBs for instructions and data. If absent, specifies that the TLB has a unified configuration. Required for a CPU with a TLB in a split configuration.
tlb-size	SD	<u32>	Specifies the number of entries in the TLB. Required for a CPU with a unified TLB for instruction and data addresses.
tlb-sets	SD	<u32>	Specifies the number of associativity sets in the TLB. Required for a CPU with a unified TLB for instruction and data addresses.
d-tlb-size	SD	<u32>	Specifies the number of entries in the data TLB. Required for a CPU with a split TLB configuration.
d-tlb-sets	SD	<u32>	Specifies the number of associativity sets in the data TLB. Required for a CPU with a split TLB configuration.
i-tlb-size	SD	<u32>	Specifies the number of entries in the instruction TLB. Required for a CPU with a split TLB configuration.
i-tlb-sets	SD	<u32>	Specifies the number of associativity sets in the instruction TLB. Required for a CPU with a split TLB configuration.
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.

3.7.3 Internal (L1) Cache Properties

The following properties of a cpu node describe the processor's internal (L1) cache.

Table 3.9: /cpu/cpu* Node Power ISA Cache Properties

Property Name	Usage	Value Type	Definition
cache-unified	SD	<empty>	If present, specifies the cache has a unified organization. If not present, specifies that the cache has a Harvard architecture with separate caches for instructions and data.
cache-size	SD	<u32>	Specifies the size in bytes of a unified cache. Required if the cache is unified (combined instructions and data).
cache-sets	SD	<u32>	Specifies the number of associativity sets in a unified cache. Required if the cache is unified (combined instructions and data)
cache-block-size	SD	<u32>	Specifies the block size in bytes of a unified cache. Required if the processor has a unified cache (combined instructions and data)
cache-line-size	SD	<u32>	Specifies the line size in bytes of a unified cache, if different than the cache block size Required if the processor has a unified cache (combined instructions and data).
i-cache-size	SD	<u32>	Specifies the size in bytes of the instruction cache. Required if the cpu has a separate cache for instructions.
i-cache-sets	SD	<u32>	Specifies the number of associativity sets in the instruction cache. Required if the cpu has a separate cache for instructions.
i-cache-block-size	SD	<u32>	Specifies the block size in bytes of the instruction cache. Required if the cpu has a separate cache for instructions.
i-cache-line-size	SD	<u32>	Specifies the line size in bytes of the instruction cache, if different than the cache block size. Required if the cpu has a separate cache for instructions.
d-cache-size	SD	<u32>	Specifies the size in bytes of the data cache. Required if the cpu has a separate cache for data.
d-cache-sets	SD	<u32>	Specifies the number of associativity sets in the data cache. Required if the cpu has a separate cache for data.
d-cache-block-size	SD	<u32>	Specifies the block size in bytes of the data cache. Required if the cpu has a separate cache for data.
d-cache-line-size	SD	<u32>	Specifies the line size in bytes of the data cache, if different than the cache block size. Required if the cpu has a separate cache for data.
next-level-cache	SD	<phandle>	If present, indicates that another level of cache exists. The value is the phandle of the next level of cache. The phandle value type is fully described in section 2.3.3.
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.

Older versions of devicetrees may be encountered that contain a deprecated form of the next-level-cache property called l2-cache. For compatibility, a client-program may wish to support l2-cache if a next-level-cache property is not

present. The meaning and use of the two properties is identical.

3.7.4 Example

Here is an example of a /cpus node with one child cpu node:

```
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
    };
};
```

3.8 Multi-level and Shared Cache Nodes (/cpus/cpu*/l?-cache)

Processors and systems may implement additional levels of cache hierarchy. For example, second-level (L2) or third-level (L3) caches. These caches can potentially be tightly integrated to the CPU or possibly shared between multiple CPUs.

A device node with a compatible value of "cache" describes these types of caches. 具有兼容值“ cache ”的设备节点描述了这些类型的缓存

The cache node shall define a phandle property, and all cpu nodes or cache nodes that are associated with or share the cache each shall contain a next-level-cache property that specifies the phandle to the cache node.

A cache node may be represented under a CPU node or any other appropriate location in the devicetree. 缓存节点可以在CPU节点下表示，也可以在devicetree中的任何其他适当位置表示

Multiple-level and shared caches are represented with the properties in Table 3-9. The L1 cache properties are described in Table 3-8. 多级缓存和共享缓存用表3-9中的属性表示

Table 3.10: /cpu/cpu*/l?-cache Node Power ISA Multiple-level and Shared Cache Properties

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			

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

3.8.1 Example

See the following example of a devicetree representation of two CPUs, each with their own on-chip L2 and a shared L3.

```
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 = <32>;
                cache-level = <3>;
            };
        };
    };

    cpu@1 {
        device_type = "cpu";
        reg = <1>;
        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
        };
    };
};
```

CHAPTER

FOUR

DEVICE BINDINGS

This chapter contains requirements, known as bindings, for how specific types and classes of devices are represented in the devicetree. The compatible property of a device node describes the specific binding (or bindings) to which the node complies. 遵从，依从，服从（comply的第三人称单数

Bindings may be defined as extensions of other each. For example a new bus type could be defined as an extension of the simple-bus binding. In this case, the compatible property would contain several strings identifying each binding—from the most specific to the most general (see section *2.3.1*, compatible).

4.1 Binding Guidelines 绑定的指导方针

4.1.1 General Principles

When creating a new devicetree representation for a device, a binding should be created that fully describes the required properties and value of the device. This set of properties shall be sufficiently descriptive to provide device drivers with needed attributes of the device.

Some recommended practices include: 一些建议的做法包括：

1. Define a compatible string using the conventions described in section *2.3.1*.
2. Use the standard properties (defined in sections *2.3* and *2.4*) as applicable for the new device. This usage typically includes the `reg` and `interrupts` properties at a minimum.
3. Use the conventions specified in section *4* (Device Bindings) if the new device fits into one the DTSpec defined device classes.
4. Use the miscellaneous property conventions specified in section *4.1.2*, if applicable.
5. If new properties are needed by the binding, the recommended format for property names is: "`<company>,<property-name>`", where `<company>` is an OUI or short unique string like a stock ticker that identifies the creator of the binding.

   Example: "`ibm,ppc-interrupt-server#s`"

4.1.2 Miscellaneous Properties 其他属性

This section defines a list of helpful properties that might be applicable to many types of devices and device classes. They are defined here to facilitate standardization of names and usage.

本节定义了一系列有用的属性，这些属性可能适用于许多类型的设备和设备类。这里定义它们是为了促进名称和用法的标准化

clock-frequency Property

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

reg-shift Property

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.

label Property

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.

4.2 Serial devices

4.2.1 Serial Class Binding

The class of serial devices consists of various types of point to point serial line devices. Examples of serial line devices include the 8250 UART, 16550 UART, HDLC device, and BISYNC device. In most cases hardware compatible with the RS-232 standard fit into the serial device class.

I^2C and SPI (Serial Peripheral Interface) devices shall not be represented as serial port devices because they have their own specific representation.

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>;

current-speed Property

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>;

4.2.2 National Semiconductor 16450/16550 Compatible UART Requirements

Serial devices compatible to the National Semiconductor 16450/16550 UART (Universal Asynchronous Receiver Transmitter) should be represented in the devicetree using following properties.

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.

4.3 Network devices

该类中的设备假定实现七层OSI模型的数据链路层(第2层)并使用媒体访问控制(MAC)地址。

网络设备是面向包的通信设备

Network devices are packet oriented communication devices. Devices in this class are assumed to implement the data link layer (layer 2) of the seven-layer OSI model and use Media Access Control (MAC) addresses. Examples of network devices include Ethernet, FDDI, 802.11, and Token-Ring.

网络设备的例子包括以太网、FDDI、802.11和令牌环。

4.3.1 Network Class Binding

address-bits Property

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>;

local-mac-address Property

Table 4.8: local-mac-address Property

Property	local-mac-address
Value type	<prop-encoded-array> encoded as an array of hex numbers
Description	Specifies MAC address that was assigned to the network device described by the node containing this property.
Example	local-mac-address = [0x00 0x00 0x12 0x34 0x56 0x78];

mac-address Property

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

4.3.2 Ethernet specific considerations 以太网的具体注意事项

Network devices based on the IEEE 802.3 collections of LAN standards (collectively referred to as Ethernet) may be represented in the devicetree using following properties, in addition to properties specified of the network device class.

The properties listed in this section augment the properties listed in the network device class.
本节中列出的属性扩充了网络设备类中列出的属性

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";

Table 4.13: Defined values for the phy-connection-type Property

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
Turbo Reduced Gigabit Media Independent Interface	trgmii

phy-handle Property

Table 4.14: phy-handle Property

Property	phy-handle
Value type	<phandle>
Description	Specifies a reference to a node representing a physical layer (PHY) device connected to this Ethernet device. This property is required in case where the Ethernet device is connected a physical layer device.
Example	phy-handle = <&PHY0>;

4.4 Power ISA Open PIC Interrupt Controllers

This section specifies the requirements for representing Open PIC compatible interrupt controllers. An Open PIC interrupt controller implements the Open PIC architecture (developed jointly by AMD and Cyrix) and specified in The Open

Programmable Interrupt Controller (PIC) Register Interface Specification Revision 1.2 *[b18]*.

Interrupt specifiers in an Open PIC interrupt domain are encoded with two cells. The first cell defines the interrupt number. The second cell defines the sense and level information.

Sense and level information shall be encoded as follows in interrupt specifiers:

```
0 = low to high edge sensitive type enabled
1 = active low level sensitive type enabled
2 = active high level sensitive type enabled
3 = high to low edge sensitive type enabled
```

Table 4.15: Open-PIC properties

Property Name	Usage	Value Type	Definition
`compatible`	R	`<string>`	Value shall include `"open-pic"`
`reg`	R	`<prop encoded array>`	Specifies the physical address of the registers device within the address space of the parent bus
`interrupt-controller`	R	`<empty>`	Specifies that this node is an interrupt controller
`#interrupt-cells`	R	`<u32>`	Shall be 2.
`#address-cells`	R	`<u32>`	Shall be 0.
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.

4.5 `simple-bus` Compatible Value

System-on-a-chip processors may have an internal I/O bus that cannot be probed for devices. The devices on the bus can be accessed directly without additional configuration required. This type of bus is represented as a node with a compatible value of “simple-bus”.

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			

CHAPTER

FIVE

FLATTENED DEVICETREE (DTB) FORMAT

扁平的设备树(DTB)格式

The Devicetree Blob (DTB) format is a flat binary encoding of devicetree data. It used to exchange devicetree data between software programs. For example, when booting an operating system, firmware will pass a DTB to the OS kernel.

Note: IEEE1275 Open Firmware [IEEE1275] does not define the DTB format. On most Open Firmware compliant platforms the devicetree is extracted by calling firmware methods to walk through the tree structure.

The DTB format encodes the devicetree data within a single, linear, pointerless data structure. It consists of a small header (see section *5.2*), followed by three variable sized sections: the memory reservation block (see section *5.3*), the structure block (see secton *5.4*), and the strings block (see section *5.5*). These should be present in the flattened devicetree in that order. Thus, the devicetree structure as a whole, when loaded into memory at address, will resemble the diagram in Fig. 5.1 (lower addresses are at the top of the diagram).

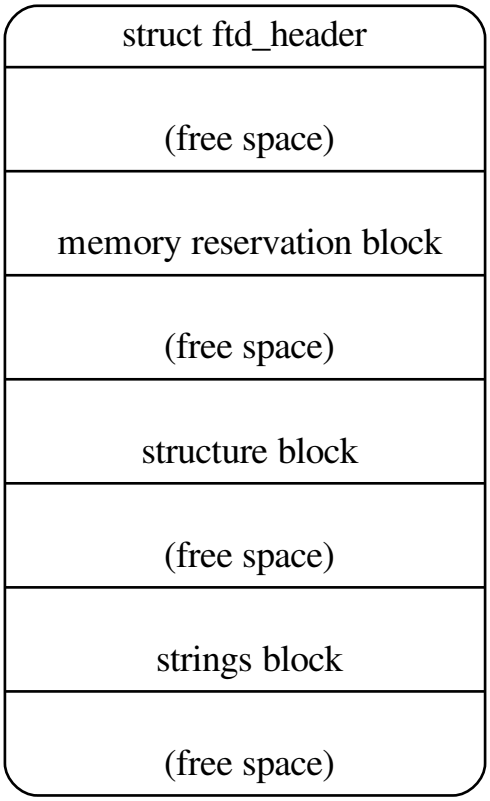

Fig. 5.1: Devicetree .dtb Structure

The (free space) sections may not be present, though in some cases they might be required to satisfy the alignment constraints of the individual blocks (see section *5.6*).
(空闲空间)部分可能不存在，尽管在某些情况下可能需要它们来满足各个块的对齐约束

5.1 Versioning

Several versions of the flattened devicetree structure have been defined since the original definition of the format. Fields in the header give the version, so that the client program can determine if the devicetree is encoded in a compatible format.

This document describes only version 17 of the format. DTSpec compliant (adj. 顺从的；服从的；应允的) boot programs shall provide a devicetree of version 17 or later, and should provide a devicetree of a version that is backwards compatible with version 16. DTSpec compliant client programs shall accept devicetrees of any version backwards compatible with version 17 and may accept other versions as well.

Note: The version is with respect to the binary structure of the device tree, not its content.

5.2 Header

The layout of the header for the devicetree is defined by the following C structure. All the header fields are 32-bit integers, stored in big-endian format.

Flattened Devicetree Header Fields

```
struct fdt_header {
    uint32_t magic;
    uint32_t totalsize;
    uint32_t off_dt_struct;
    uint32_t off_dt_strings;
    uint32_t off_mem_rsvmap;
    uint32_t version;
    uint32_t last_comp_version;
    uint32_t boot_cpuid_phys;
    uint32_t size_dt_strings;
    uint32_t size_dt_struct;
};
```

magic This field shall contain the value 0xd00dfeed (big-endian).

totalsize This field shall contain the total size in bytes of the devicetree data structure. This size shall encompass all sections of the structure: the header, the memory reservation block, structure block and strings block, as well as any free space gaps between the blocks or after the final block.

off_dt_struct This field shall contain the offset in bytes of the structure block (see section *5.4*) from the beginning of the header. 这个字段应该包含结构块(参见5.4节)从头开始的字节偏移量。

off_dt_strings This field shall contain the offset in bytes of the strings block (see section *5.5*) from the beginning of the header.

off_mem_rsvmap This field shall contain the offset in bytes of the memory reservation block (see section *5.3*) from the beginning of the header.

version This field shall contain the version of the devicetree data structure. The version is 17 if using the structure as defined in this document. An DTSpec boot program may provide the devicetree of a later version, in which case this field shall contain the version number defined in whichever later document gives the details of that version.

last_comp_version This field shall contain the lowest version of the devicetree data structure with which the version used is backwards compatible. So, for the structure as defined in this document (version 17), this field shall contain 16 because version 17 is backwards compatible with version 16, but not earlier versions. As per section *5.1*, a DTSpec boot program should provide a devicetree in a format which is backwards compatible with version 16, and thus this field shall always contain 16.

boot_cpuid_phys This field shall contain the physical ID of the system's boot CPU. It shall be identical to the physical ID given in the `reg` property of that CPU node within the devicetree.

size_dt_strings This field shall contain the length in bytes of the strings block section of the devicetree blob.

size_dt_struct This field shall contain the length in bytes of the structure block section of the devicetree blob.

5.3 Memory Reservation Block

5.3.1 Purpose

The *memory reservation block* provides the client program with a list of areas in physical memory which are *reserved*; that is, which shall not be used for general memory allocations. It is used to protect vital data structures from being overwritten by the client program. For example, on some systems with an IOMMU, the TCE (translation control entry) tables initialized by a DTSpec boot program would need to be protected in this manner. Likewise, any boot program code or data used during the client program's runtime would need to be reserved (e.g., RTAS on Open Firmware platforms). DTSpec does not require the boot program to provide any such runtime components, but it does not prohibit implementations from doing so as an extension.

More specifically, a client program shall not access memory in a reserved region unless other information provided by the boot program explicitly indicates that it shall do so. The client program may then access the indicated section of the

reserved memory in the indicated manner. Methods by which the boot program can indicate to the client program specific uses for reserved memory may appear in this document, in optional extensions to it, or in platform-specific documentation.

The reserved regions supplied by a boot program may, but are not required to, encompass the devicetree blob itself. The client program shall ensure that it does not overwrite this data structure before it is used, whether or not it is in the reserved areas.

Any memory that is declared in a memory node and is accessed by the boot program or caused to be accessed by the boot program after client entry must be reserved. Examples of this type of access include (e.g., speculative memory reads through a non-guarded virtual page).

This requirement is necessary because any memory that is not reserved may be accessed by the client program with arbitrary storage attributes.

Any accesses to reserved memory by or caused by the boot program must be done as not Caching Inhibited and Memory Coherence Required (i.e., WIMG = 0bx01x), and additionally for Book III-S implementations as not Write Through Required (i.e., WIMG = 0b001x). Further, if the VLE storage attribute is supported, all accesses to reserved memory must be done as VLE=0.

This requirement is necessary because the client program is permitted to map memory with storage attributes specified as not Write Through Required, not Caching Inhibited, and Memory Coherence Required (i.e., WIMG = 0b001x), and VLE=0 where supported. The client program may use large virtual pages that contain reserved memory. However, the client program may not modify reserved memory, so the boot program may perform accesses to reserved memory as Write Through Required where conflicting values for this storage attribute are architecturally permissible.

5.3.2 Format

The memory reservation block consists of a list of pairs of 64-bit big-endian integers, each pair being represented by the following C structure.

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

Each pair gives the physical address and size in bytes of a reserved memory region. These given regions shall not overlap each other. The list of reserved blocks shall be terminated with an entry where both address and size are equal to 0. Note that the address and size values are always 64-bit. On 32-bit CPUs the upper 32-bits of the value are ignored.

Each uint64_t in the memory reservation block, and thus the memory reservation block as a whole, shall be located at an 8-byte aligned offset from the beginning of the devicetree blob (see section *5.6*).

5.4 Structure Block

结构块描述了devicetree本身的结构和内容　　它由一系列带有数据的令牌组成，如下所述

The structure block describes the structure and contents of the devicetree itself. It is composed of a sequence of tokens with data, as described below. These are organized into a linear tree structure, as described below.

它们被组织成线性树结构，如下所述

Each token in the structure block, and thus the structure block itself, shall be located at a 4-byte aligned offset from the beginning of the devicetree blob (see *5.6*).

5.4.1 Lexical structure

结构块由一系列块组成，每个块都以一个令牌开始，即一个32位的大端整数。

The structure block is composed of a sequence of pieces, each beginning with a token, that is, a big-endian 32-bit integer. Some tokens are followed by extra data, the format of which is determined by the token value. All tokens shall be aligned on a 32-bit boundary, which may require padding bytes (with a value of 0x0) to be inserted after the previous token's data.

The five token types are as follows:

五种令牌类型如下：

FDT_BEGIN_NODE (0x00000001) The FDT_BEGIN_NODE token marks the beginning of a node's representation. It shall be followed by the node's unit name as extra data. The name is stored as a null-terminated string, and shall include the unit address (see section *2.2.1*), if any. The node name is followed by zeroed padding bytes, if necessary for alignment, and then the next token, which may be any token except FDT_END.

如果需要对齐，节点名后面跟着零填充字节，然后是下一个标记，它可以是除FDT_END之外的任何标记

FDT_END_NODE (0x00000002) The FDT_END_NODE token marks the end of a node's representation. This token has no extra data; so it is followed immediately by the next token, which may be any token except FDT_PROP.

这个令牌没有额外的数据；

FDT_PROP (0x00000003) The FDT_PROP token marks the beginning of the representation of one property in the devicetree. It shall be followed by extra data describing the property. This data consists first of the property's length and name represented as the following C structure:

```
struct {
    uint32_t len;
    uint32_t nameoff;
}
```

Both the fields in this structure are 32-bit big-endian integers.

- len gives the length of the property's value in bytes (which may be zero, indicating an empty property, see section *2.2.4.2*).
- nameoff gives an offset into the strings block (see section *5.5*) at which the property's name is stored as a null-terminated string.

After this structure, the property's value is given as a byte string of length len. This value is followed by zeroed padding bytes (if necessary) to align to the next 32-bit boundary and then the next token, which may be any token except FDT_END.

任何解析设备树的程序都会忽略FDT_NOP标记

FDT_NOP (0x00000004) The FDT_NOP token will be ignored by any program parsing the device tree. This token has no extra data; so it is followed immediately by the next token, which can be any valid token. A property or node definition in the tree can be overwritten with FDT_NOP tokens to remove it from the tree without needing to move other sections of the tree's representation in the devicetree blob.

FDT_END (0x00000009) The FDT_END token marks the end of the structure block. There shall be only one FDT_END token, and it shall be the last token in the structure block. It has no extra data; so the byte immediately after the FDT_END token has offset from the beginning of the structure block equal to the value of the size_dt_struct field in the device tree blob header.

它没有额外的数据；后面的字节
FDT_END标记从结构块开始的偏移量等于设备树blob头中的size_dt_struct字段的值

5.4.2 Tree structure

The devicetree structure is represented as a linear tree: the representation of each node begins with an FDT_BEGIN_NODE token and ends with an FDT_END_NODE token. The node's properties and subnodes (if any) are represented before the FDT_END_NODE, so that the FDT_BEGIN_NODE and FDT_END_NODE tokens for those subnodes are nested within those of the parent.

The structure block as a whole consists of the root node's representation (which contains the representations for all other nodes), followed by an FDT_END token to mark the end of the structure block as a whole.

More precisely, each node's representation consists of the following components:

- (optionally) any number of FDT_NOP tokens
- FDT_BEGIN_NODE token
  - The node's name as a null-terminated string
  - [zeroed padding bytes to align to a 4-byte boundary]
- For each property of the node:
  - (optionally) any number of FDT_NOP tokens
  - FDT_PROP token
    - property information as given in section *5.4.1*

将零填充字节对齐到4字节边界

  * [zeroed padding bytes to align to a 4-byte boundary]

- Representations of all child nodes in this format
- (optionally) any number of FDT_NOP tokens
- FDT_END_NODE token

n. 详细说明；个别项目
adj. 特别的；详细的；独有的；挑剔的

Note that this process requires that all property definitions for a particular node precede any subnode definitions for that node. Although the structure would not be ambiguous if properties and subnodes were intermingled, the code needed to process a flat tree is simplified by this requirement.

尽管如果属性和子节点混杂在一起，结构就不会含糊不清，但是根据这个要求，处理平面树所需的代码被简化了

5.5 Strings Block

v. 终止；结束；终结（terminate的过去分词）
adj. 终止的；有限的

字符串块包含表示树中使用的所有属性名的字符串

The strings block contains strings representing all the property names used in the tree. These null terminated strings are simply concatenated together in this section, and referred to from the structure block by an offset into the strings block.

The strings block has no alignment constraints and may appear at any offset from the beginning of the devicetree blob.

5.6 Alignment

For the data in the memory reservation and structure blocks to be used without unaligned memory accesses, they shall lie at suitably aligned memory addresses. Specifically, the memory reservation block shall be aligned to an 8-byte boundary and the structure block to a 4-byte boundary.

Furthermore, the devicetree blob as a whole can be relocated without destroying the alignment of the subblocks.

此外，devicetree blob作为一个整体可以重新定位，而不会破坏子块的对齐

As described in the previous sections, the structure and strings blocks shall have aligned offsets from the beginning of the devicetree blob. To ensure the in-memory alignment of the blocks, it is sufficient to ensure that the devicetree as a whole is loaded at an address aligned to the largest alignment of any of the subblocks, that is, to an 8-byte boundary. A DTSpec compliant boot program shall load the devicetree blob at such an aligned address before passing it to the client program. If an DTSpec client program relocates the devicetree blob in memory, it should only do so to another 8-byte aligned address.

CHAPTER

SIX

DEVICETREE SOURCE (DTS) FORMAT (VERSION 1)

The Devicetree Source (DTS) format is a textual representation of a devicetree in a form that can be processed by dtc into a binary devicetree in the form expected by the kernel. The following description is not a formal syntax definition of DTS, but describes the basic constructs used to represent devicetrees.

The name of DTS files should end with ".dts".

6.1 Compiler directives

Other source files can be included from a DTS file. The name of include files should end with ".dtsi". Included files can in turn include additional files.

```
/include/ "FILE"
```

6.2 Labels

源格式允许将标签附加到设备树中的任何节点或属性值。

The source format allows labels to be attached to any node or property value in the device tree. Phandle and path references can be automatically generated by referencing a label instead of explicitly specifying a phandle value or the full path to a node. Labels are only used in the devicetree source format and are not encoded into the DTB binary.

A label shall be between 1 to 31 characters in length, be composed only of the characters in the set Table 6.1, and must not start with a number.

Labels are created by appending a colon (':') to the label name. References are created by prefixing the label name with an ampersand ('&').

Table 6.1: Valid characters for DTS labels

Character	Description
0-9	digit
a-z	lowercase letter
A-Z	uppercase letter
_	underscore

6.3 Node and property definitions

Devicetree nodes are defined with a node name and unit address with braces marking the start and end of the node definition. They may be preceded by a label.

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

Nodes may contain property definitions and/or child node definitions. If both are present, properties shall come before child nodes.

Previously defined nodes may be deleted.

```
/delete-node/ node-name;
/delete-node/ &label;
```

Property definitions are name value pairs in the form:

```
[label:] property-name = value;
```

except for properties with empty (zero length) value which have the form:

```
[label:] property-name;
```

Previously defined properties may be deleted.

```
/delete-property/ property-name;
```

Property values may be defined as an array of 32-bit integer cells, as null-terminated strings, as bytestrings or a combination of these.

- Arrays of cells are represented by angle brackets surrounding a space separated list of C-style integers. Example:

```
interrupts = <17 0xc>;
```

- values may be represented as arithmetic, bitwise, or logical expressions within parenthesis.

```
Arithmetic operators

+   add
-   subtract
*   multiply
/   divide
```

```
Bitwise operators

&    and
|    or
^    exclusive or
~    not
<<  left shift
>>  right shift
```

```
Logical operators

&&   and
||   or
!    not
```

```
Relational operators

<    less than
>    greater than
<=   less than or equal
>=   greater than or equal
==   equal
!=   not equal
```

```
Ternary operators  三目运算操作

?:   (condition ? value_if_true : value_if_false)
```

- A 64-bit value is represented with two 32-bit cells. Example:

```
clock-frequency = <0x00000001 0x00000000>;
```

以null结尾的字符串值使用双引号表示

- A null-terminated string value is represented using double quotes (the property value is considered to include the terminating NULL character). Example:

```
compatible = "simple-bus";
```

- A bytestring is enclosed in square brackets [ ] with each byte represented by two hexadecimal digits. Spaces between each byte are optional. Example:

```
local-mac-address = [00 00 12 34 56 78];
```

or equivalently:

```
local-mac-address = [000012345678];
```

- Values may have several comma-separated components, which are concatenated together. Example:

```
compatible = "ns16550", "ns8250";
example = <0xf00f0000 19>, "a strange property format";
```

- In a cell array a reference to another node will be expanded to that node's phandle. References may be & followed by a node's label. Example:

```
interrupt-parent = < &mpic >;
```

or they may be & followed by a node's full path in braces. Example:

```
interrupt-parent = < &{/soc/interrupt-controller@40000} >;
```

- Outside a cell array, a reference to another node will be expanded to that node's full path. Example:

```
ethernet0 = &EMAC0;
```

- Labels may also appear before or after any component of a property value, or between cells of a cell array, or between bytes of a bytestring. Examples:

```
reg = reglabel: <0 sizelabel: 0x1000000>;
prop = [ab cd ef byte4: 00 ff fe];
str = start: "string value" end: ;
```

6.4 File layout

Version 1 DTS files have the overall layout:

```
/dts-v1/;
[memory reservations]
    / {
        [property definitions]
        [child nodes]
    };
```

The /dts-v1/; shall be present to identify the file as a version 1 DTS (dts files without this tag will be treated by dtc as being in the obsolete version 0, which uses a different format for integers in addition to other small but incompatible changes).

Memory reservations define an entry for the devicetree blob's memory reservation table. They have the form: e.g., /memreserve/ <address> <length>; Where <address> and <length> are 64-bit C-style integers.

- The / { }; section defines the root node of the devicetree.
- C style (/* ... */) and C++ style (//) comments are supported.

BIBLIOGRAPHY

n. 参考书目；文献目录

[b1] *Power ISA™*, Version 2.06 Revision B, July 23, 2010. It is available from power.org (http://power.org)

[IEEE1275] *Boot (Initialization Configuration) Firmware: Core Requirements and Practices*, 1994, This is the core standard (also known as IEEE 1275) that defines the devicetree concept adopted by the DTSpec and ePAPR. It is available from Global Engineering (http://global.ihs.com/).

[b3] *PowerPC Processor Binding to IEEE 1275-1994 Standard for Boot (Initialization, Configuration) Firmware*, Version 2.1, Open Firmware Working Group, (http://playground.sun.com/1275/bindings/ppc/release/ppc-2_1.html), 1996, This document specifies the PowerPC processor specific binding to the base standard.

[b4] *booting-without-of.txt*, Ben Herrenschmidt, Becky Bruce, et al., From the Linux kernel source tree (http://www.kernel.org/), Describes the devicetree as used by the Linux kernel.

[b5] *Device Trees Everywhere*, David Gibson and Ben Herrenschmidt (http://ozlabs.org/~dgibson/home/papers/dtc-paper.pdf), An overview of the concept of the devicetree and devicetree compiler.

[b6] *PCI Bus Binding to: IEEE Std 1275-1994 Standard for Boot (Initialization Configuration) Firmware*, Revision 2.1, Open Firmware Working Group, 1998 (http://playground.sun.com/1275/bindings/pci/pci2_1.pdf)

[b7] *Open Firmware Recommended Practice: Interrupt Mapping*, Version 0.9, Open Firmware Working Group, 1996 (http://playground.sun.com/1275/practice/imap/imap0_9d.pdf)

[b8] *Open Firmware Recommended Practice: Device Support Extensions*, Version 1.0, Open Firmware Working Group, 1997, (http://playground.sun.com/1275/practice/devicex/dse1_0a.html) This document describes the binding for various device types such as network, RTC, keyboard, sound, etc.

[b9] *Open Firmware Recommended Practice: Universal Serial Bus Binding to IEEE 1275*, Version 1, Open Firmware Working Group, 1998 (http://playground.sun.com/1275/bindings/usb/usb-1_0.ps)

[CHRP] *PowerPC Microprocessor Common Hardware Reference Platform (CHRP) Binding*, Version 1.8, Open Firmware Working Group, 1998 (http://playground.sun.com/1275/bindings/chrp/chrp1_8a.ps). This document specifies the properties for Open PIC-compatible interrupt controllers.

[b11] *CHRP ISA Interrupt Controller Device Binding*, Unapproved Draft version 1.1, Open Firmware Working Group, Aug 19, 1996 (http://playground.sun.com/1275/bindings/devices/postscript/isa-pic-1_1d.ps)

[b12] *The Open Programmable Interrupt Controller (PIC) Register Interface Specification*, Revision 1.2, Advanced Micro Devices and Cyrix Corporation, October 1995

[b13] *PCI Local Bus Specification*, Revision 2.2, PCI Special Interest Group

[b14] *PCI Express Base Specification*, Revision 1.0a, PCI Special Interest Group

[b15] *PCI-Express Binding to OF*, P1275 Openboot Working Group Proposal, 18 August 2004

[PAPR] *Power.org Standard for Power Architecture Platform Requirements*, power.org

[b17] *System V Application Binary Interface, Edition 4.1*, Published by The Santa Cruz Operation, Inc., 1997

[b18] *The Open Programmable Interrupt Controller (PIC) Register Interface Specification Revision 1.2*, AMD and Cyrix, October 1995

[b19] *RFC 2119, Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt

[b20] *64-bit PowerPC ELF Application Binary Interface Supplement 1.9*, Ian Lance Taylor, 2004

[EPAPR] *Power.org Standard for Embedded Power Architecture Platform Requirements*, power.org, 2011, https://www.power.org/documentation/power-org-standard-for-embedded-power-architecture-platform-requirements-epapr-v1-1-2/

[ARMv8] *ARM DDI 0487 ARM(c) Architecture Reference Manual, ARMv8 for ARMv8-A architecture profile*, ARM, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487a.h/index.html

INDEX