第5章 PCI

PCI的英文全称为Periheral Component Interconnect。正如它的名称一样， PCI局部总线是微型计算机系统上处理器/存储器与外围控制部件、外围附加板之间的互连机构。“ PCI局部总线规范3 ”规定了互连机构的协议、电气、机械以及配置空间规范。本章主要介绍 Linux的内核如何初始化系统的PCI总线和设备。

图1-5-1是一个基于PCI局部总线的系统逻辑示意图。PCI局部总线和PCI-PCI桥是将系统的部件连接起来的连接器。 CPU连接到PCI局部总线0 ，这条总线主要用于连接视频设备。被称为PCI-PCI桥的特别PCI设备将PCI局部总线0与从PCI局部总线连接起来。这种“ 从 PCI总线 ”被称为 PCI局部总线1。按照PCI规范的术语来讲， PCI局部总线1被称作位于PCI-PCI桥的下游(downstream)，而PCI局部总线0被称为桥的上游(up-stream)。从PCI局部总线主要连接系统的 SCSI设备和以太网设备。桥、从PCI局部总线以及上面的两种设备都可以物理地集成在同一个 PCI集成卡中。系统中的 PCI-ISA桥支持老式、遗留下来的 ISA设备。在图1-5-1中画出了一个超级 I/O控制器芯片，它可以控制键盘、鼠标和软盘驱动器。

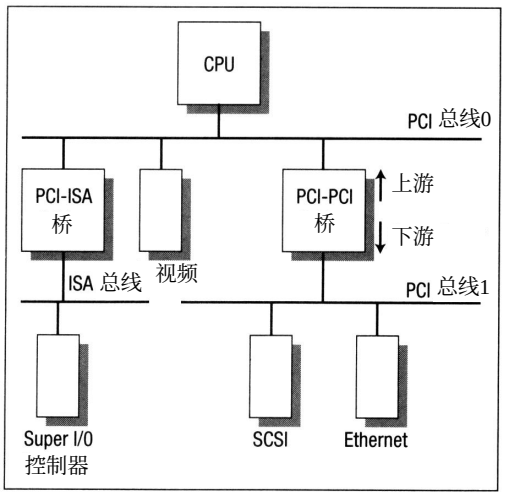

图1-5-1 基于PCI局部总线系统的示例

5.1 PCI的地址空间

CPU和PCI的设备需要访问在二者之间共享的存储空间。这部分存储器用于设备驱动程序控制PCI设备并在驱动程序和设备之间传递信息。典型的共享存储器包括设备的控制和状态寄存器，这些寄存器可以控制 PCI设备并读取设备的状态信息。例如： PCI的SCSI设备驱动程序从SCSI设备读取状态信息，以确定 SCSI设备是否可以向SCSI磁盘写入一整块的数据。 SCSI设备驱动程序还可以在启动后，向控制寄存器写入控制命令以启动 SCSI设备运行。

CPU的系统存储器可以用作共享存储器。但如果这样做的话，每次 PCI设备访问存储器时，CPU不得不暂停，等待PCI设备完成访存操作。这样访存操作就变成每次只有一个系统部件可以访存，导致整个系统的性能下降。让系统的外设以不加控制的方式来访问主存本身就不是一个好办法。这样做非常危险，因为一个劣质的外设会使系统非常不稳定。因此外设可以有自己的存储空间， CPU可以访问这些存储空间而外设对系统存储空间的访问用 DMA(Direct Memory Access，直接存储访问)通道的方式来严加控制。 ISA设备可以访问两类地址空间：ISA的I/O空间和ISA存储空间。 PCI设备可以访问三类地址空间： PCI的I/O空间、 PCI的存储空间和PCI的配置空间。所有的这些空间对 CPU来说都是可访问的，其中 PCI的I/O空间和PCI

存储空间被设备驱动程序使用， PCI的配置空间被Linux内核中的初始化程序使用。

Alpha AXP处理器不支持对除系统地址空间之外的其他地址空间的直接访问。它使用支持芯片组来访问像PCI配置空间这样的其他地址空间。实现上一般采用分析地址映射机制，即从整个虚拟地址空间中窃取一部分地址空间，并把它映射到 PCI的地址空间上。

5.2 PCI配置头

系统中的每个PCI设备(包括PCI-PCI桥设备)都在PCI的配置地址空间的某处有一个配置数据结构。PCI的配置头允许系统来标识、控制设备。而 PCI配置头在PCI配置地址空间的精确位置依赖于设备在PCI拓朴结构图中的位置。例如：一块 PCI视频卡插到PCI主板的一个PCI插槽中，它的配置头会出现在 PCI配置空间的一个地址上；如果把它换到另一个 PCI插槽中，那么它的配置头会出现在另一个地址上。但这种情况不会造成混淆，因为无论 PCI设备和桥在哪里，系统都会使用它们的配置头中的状态和配置寄存器找到并配置这些设备。

典型的系统一般都被设计成使每个PCI插槽的配置头的偏移量与插槽在主板上的位置相关。例如：主板上的第一个PCI插槽的PCI配置头的偏移量是0，第二个的偏移量是256(注：所有的配置头有相同的长度 —— 256个字节)，其他的以此类推。PCI总线规范定义了一种与系统相关的硬件机制，使得PCI的配置程序通过检验PCI总线上所有可能的PCI配置头中的一个域，就能区分哪些设备是连接在总线上的，那些是断开的。“ PCI局部总线规范3 ”定义了在读取空PCI插槽的厂商标识和设备标识域时，会返回值为0XFFFFFFFF的错误信息。因此PCI的配置程序一般读取配置头中的厂商标识域来判定PCI插槽的状态。如果返回错误信息0XFFFFFFFF，则说明插槽为空。

图1-5-2给出了256字节的PCI配置头的格式，它包含下列几个域：

- 厂商标识(Vendor Id) 是一个用于唯一标识 PCI设备生产厂商的数值。 Digital公司的 PCI厂商标识为0x1011，而Intel的是0x8086。
- 设备标识(Device Id) 用于唯一标识一个特定设备的数值。例如Digtal公司的21141快速以太网设备的设备标识为0x0009。
- 状态(Status) 根据 “ PCI局部总线规范3 ”定义的域中每个位的含义来给出设备的状态。
- 命令(Command) 系统通过向这个域中写入命令来控制设备。例如：可以写入允许设备访问 PCI I/O地址空间的命令。
- 分类码(Glass Code) 是设备类型的标识。规范对每种设备进行了标准的分类。如视频设备、 SCSI设备等等。SCSI的分类码为0x0100。
- 基地址寄存器(Base Address Register) 这些寄存器用于决定设备可以使用的 PCI I/O空间和存储空间的类型、大小，并指定它们的起始位置。

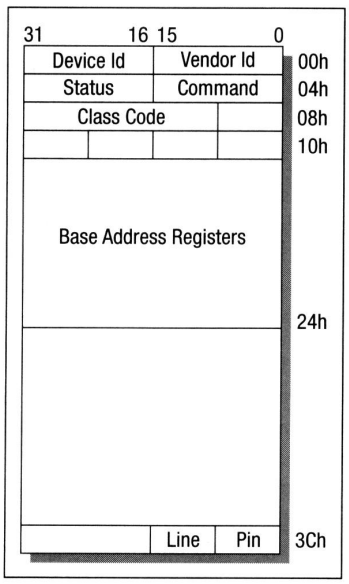

图1-5-2 PCI配置头

- 中断引脚(Pin) PCI卡有4个物理引脚，用于把中断从卡上发送到 PCI总线上。它们的标准标号为A、B、C、D。中断引脚(interrupt pin)域用于标识该PCI设备使用哪个引脚。一

般这种功能是由每个设备硬件实现的。也就是说，每次系统启动时，该设备都使用相同的中断引脚。这个信息的用处是让中断处理子系统可以处理来自该设备的中断。

- 中断线(Line) 设备的PCI配置头的中断线域用于在设备驱动程序的PCI初始化代码和Linux中断处理子系统间传递中断处理。写入中断线(interrupt line)域的数字对设备驱动程序来说是无意义的。但它允许中断处理器正确地把来自PCI设备的中断传送到操作系统中对应的设备驱动程序中断处理例程中。若想更进一步了解Linux如何处理中断，请看第6章。

5.3 PCI的I/O和存储地址空间

PCI设备共有两类地址空间可以用于与在核态运行的设备驱动程序进行通信。例如： DEC公司的21141快速以太网设备芯片把它的内部寄存器映射到 PCI的I/O地址空间中，这样它的设备驱动程序就可以通过读、写这些寄存器来控制快速以太网设备。典型的视频设备驱动程序通常会使用大量的PCI存储地址空间来获得视频信息。

在PCI系统建立之前和使PCI配置头中的命令域允许设备访问这些地址空间之前， PCI设备是不可设问的。在 Linux系统中，只有 PCI配置程序能读写 PCI的配置地址空间，而 Linux的设备驱动程序只能读写PCI的I/O和存贮器地址空间。

5.4 PCI-ISA桥

PCI-ISA桥通过把对PCI的I/O和存储地址空间的访问转换成对 ISA的I/O和存储器的访问，来实现对老的ISA设备的支持。现在售出的大量 PC系统都包括几个ISA总线插槽和几个PCI总线插槽。随着时间的推移，对这种向后兼容能力的需求会不断减少，最后会出现只有 PCI总线的PC系统。在ISA地址空间中， ISA设备有一些由早期基于 Intel 8080的PC定义的一些固定的寄存器。例如即使一台售价为 5000美元的Alpha AXP计算机系统，也会像第一台 IBM PC一样在ISA I/O地址空间的相同位置留有ISA软盘控制器。 PCI规范通过将PCI的I/O和存贮地址空间的低端保留给系统中的 ISA外设使用，用一个 PCI-ISA桥把对这部分 PCI存储空间的访问转换为对ISA地址空间访问的方式，解决了上述兼容问题。

5.5 PCI-PCI 桥

PCI-PCI桥是将系统中的所有PCI总线连接在一起的特殊的PCI设备。简单的系统可以只有一条PCI总线。但是一条总线可以支持的 PCI总线数受它的电气特性的限制。通过 PCI桥增加更多的PCI总线可以使系统支持更多的 PCI设备，这对一台高性能的服务器来说是至关重要的。当然， Linux完全支持使用PCI-PCI桥。

5.5.1 PCI-PCI桥：PCI I/O和存储地址空间的窗口

PCI-PCI桥只是把对部分PCI I/O和存储地址空间的读写请求传送到下游。例如：对图1-5-1，如果读写的是SCSI设备或以太网设备的 PCI I/O和存储地址空间， PCI-PCI桥会把这些读写从PCI总线0传送到PCI总线1，而忽略对其他PCI I/O和存储地址空间的访问。这种过滤功能防止了系统中不必要的地址传播。为了达到这个目标，必须对系统中的 PCI-PCI桥进行编程，为从主总线传送到从总线的PCI I/O和存储地址访问指定基址和上界。一旦系统中的 PCI-PCI桥配置成功了，那么只要Linux设备驱动程序，通过这些窗口访问 PCI的I/O和存贮器地址空间， PCI-

PCI桥对它们来说就是不可见的。这对 Linux PCI 设备驱动程序的编写者来说就是一个减轻工作量的重要特征。但它却使得 Linux配置PCI-PCI桥变得非常复杂。

5.5.2 PCI-PCI桥：PCI配置周期和PCI总线编号

由于CPU的PCI初始化代码可以查找出不在主 PCI总线上的设备。因此 PCI-PCI桥也一定有一种机制，使得它可以决定是否把配置周期从主接口传送到从接口上去。周期在 PCI总线上总以地址的形式出现。PCI规范定义了两种类型的 PCI配置地址：0型和1型，图1-5-3和图1-5-4给出了它们的格式。0型的PCI配置周期不包含总线号，由该总线上的设备像解释 PCI配置地址一样进行解释。0型配置周期的11至31位是设备选择域。一种设计系统的方式是由设备选择域的每一位选择一个设备，这时第 11位可以选择在插槽 0的PCI设备，第12位可以选择在插槽 1的PCI设备，并以此类推；另一种实现方式是把 PCI设备号直接写到第 11到31位中。一个系统使用哪种实现方式依赖于系统的 PCI存储控制器。

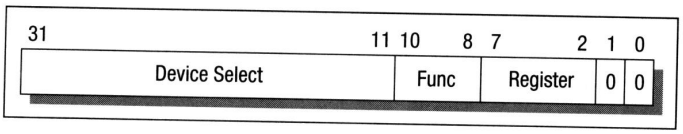

图1-5-3 0型 PCI配置周期

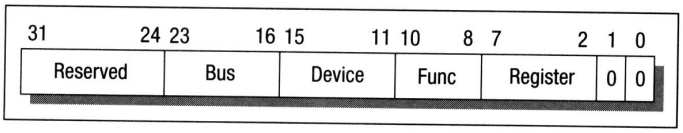

图1-5-4 1型PCI配置周期

1型PCI配置周期包含PCI总线号。除了PCI-PCI桥之外所有的PCI设备都忽略它。所有看到1型配置周期的PCI-PCI桥可以选择把它们传送到与自己相连的下游 PCI总线上。而PCI-PCI桥是忽略1型配置周期还是把它传送到下游的 PCI总线取决于PCI-PCI桥的配置。每个PCI-PCI桥都有一个主总线接口号和一个从总线接口号。其中主总线接口指离 CPU近的一端，而从总线接口指离CPU较远的一端。每个PCI-PCI桥还有一个下级总线号，它是从从总线接口桥接出去的所有PCI总线中最大的总线号。从另一角度来讲，下级总线号指位于 PCI-PCI桥下游的最大PCI总线号。在PCI-PCI桥收到1型PCI配置周期时，它可以做下列操作之一：

1) 如果指出的总线号不在桥的从总线号和下级总线号 (包括下级总线号)之间，桥简单地忽略它。

2) 如果指出的总线号等于桥的从总线号，桥将 1型配置周期转变为0型配置周期。

3) 如果指出的总线号大于桥的从总线号，而小于等于桥的下级总线号。桥将其无变化地传送到从总线接口上。所以要想在图 1-5-9的系统拓扑图中寻址总线 3上的设备1，CPU会在总线0上产生一个1型配置命令。桥1将此命令无变化地传到总线 1上，桥2会忽略它，而桥 3会把它转化为0型配置命令，发送到总线3上。在那儿设备1对其作出响应。

在PCI配置期间，由操作系统独立地分配总线号。但无论采用什么编号方式，系统中的PCI-PCI桥必须遵守如下规则：

所有在PCI-PCI桥下游的PCI总线号必须在该桥的从总线号和下级总线号 (包含它)之间。

如果这个规则被打破了，那么PCI-PCI桥就无法正确地传递和转换1型PCI配置周期，而操作

系统也无法发现、初始化系统中的所有PCI设备。为了遵循上述编号方式，Linux按一种特别的方式来配置这些特殊设备。5.6.2节根据一个工作示例详细讲述了Linux的PCI桥和总线编号机制。

5.6 Linux PCI初始化

Linux系统中的PCI初始化程序分成三个逻辑部分。

- PCI设备驱动程序　这个伪设备驱动程序从总线0开始搜索PCI系统，定位系统中所有的PCI设备和桥。它建立一个数据结构的链表来描述系统的拓扑结构。另外，它还为所有找到的桥分配编号。
- PCI BIOS　这一软件层提供了“PCI BIOS只读存储器规范 4”指出的所有服务。尽管Alpha AXP没有BIOS服务，但在Linux内核中有等价的代码来提供相同的功能服务。
- PCI修理部分　与系统相关的修理程序，用于整理与系统相关的PCI初始化过程中的故障点。

5.6.1 Linux内核PCI数据结构

Linux内核初始化PCI系统时，它将建立起能反映系统中实际的PCI拓扑结构的数据结构。图1-5-5给出了各数据结构之间的关系。它是从图1-5-1中的示例性PCI系统导出的。每个PCI设备由pci_bus数据结构来描述。最后产生的结果是一种树形结构的PCI总线，其中每个总线有若干个子PCI设备连接在上面。由于一个PCI总线只能通过PCI-PCI桥才能到达，所以除总线0以外，每个pci_bus数据结构都包括指向与其相连的位于上游的PCI-PCI桥的指针。PCI设备是所在PCI总线的父总线的子辈。在图1-5-5中给出了名为pci_devices的指向系统中所有PCI设备的指针，系统中每个PCI设备都将它们的pci_dev数据结构加入到这个队列中，该队列由Linux内核使用，以快速找到系统中所有的PCI设备。

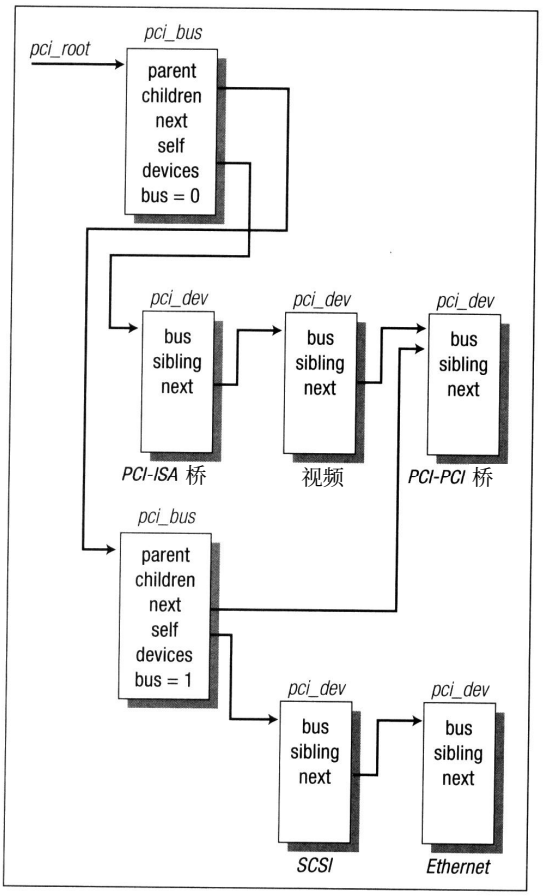

图1-5-5 Linux内核中PCI数据结构

5.6.2 PCI设备驱动程序

PCI设备驱动程序根本就不是一个真正的驱动程序，它只是在系统初始化时由操作系统调用的一个函数。PCI初始化程序必须先扫描系统中的所有PCI总线，查找系统中的所有PCI设备(包含PCI-PCI桥)。它使用PCI BIOS中的例程来确定当前正在扫描的PCI总线的每个PCI插槽是否是空的。如果PCI插槽被占用了，

它会建立一个描述该设备的pci_dev数据结构，并把它连入由pci_devices指针指向的已知PCI设备链表中。

PCI初始化程序从PCI总线0开始扫描，它对每个PCI插槽中的每个可能的PCI设备都读取厂商标识域和设备标识域。一旦找到了一个被占用的插槽，它会建立一个描述该设备的pci_dev数据结构，由PCI初始化程序建立的所有pci_dev数据结构(包括PCI-PCI桥)都被链入pci_devices指向的链表中。

如果PCI初始化程序发现某个PCI设备是桥，就会建立一个pic_bus数据结构，并把它链接到由pci_root指向的pci_bus和pci_dev数据结构的树中。由于PCI-PCI桥的分类码为0X060400，所以PCI初始化程序可以区分某个设备是否为PCI-PCI桥。Linux内核会立即配置在PCI-PCI桥下游的PCI总线；如果在那条总线上还有多个PCI-PCI桥，那么它们也会被配置。这个过程被称为深度优先算法(depthwise algorithm)。因此在广度搜索之前，系统的PCI拓扑结构先按深度优先的方式进行映射。图1-5-1中，Linux在配置PCI总线0上的视频设备之前，首先配置PCI总线1上的以太网和SCSI设备。

在Linux插索下游PCI总线时，它还要配置PCI-PCI桥的从总线号和下级总线号之间的间隔。这将在5.6.2小节详细介绍。

配置PCI-PCI桥——分配PCI总线号

要想让PCI-PCI桥传递对PCI I/O地址空间、存储地址空间和PCI配置地址空间的读写，需要知道下列的信息：

- 主总线(primary bus)号　与PCI-PCI桥直接相连的上游PCI总线的总线号。
- 从总线(secondary bus)号　与PCI-PCI桥直接相连的下游PCI总线的总线号。
- 下级总线(subordinate bus)号　由该桥的下游可以到达的所有PCI总线中最大的总线号。
- PCI I/O和存储器窗口　PCI-PCI桥所有下游地址的PCI I/O地址空间和PCI存储地址空间的窗口的基址和大小。

问题是当你想配置一个给定的PCI-PCI桥时，却无法知道该桥下级总线的数目。即使知道了，但仍不知道下游是否还有PCI-PCI桥以及给它们分配什么标号。解决的办法是使用反向的深度优先搜索算法，先搜索与PCI桥相连的每条总线，并为找到的总线分配标号。在找到每一个PCI-PCI桥和它的从总线号后，先为PCI-PCI桥分配一个0xFF的临时下级总线号，再去搜索它的下游的所有PCI-PCI桥并为其分配标号。上述算法看起来很复杂，但下面的例子有助于理解该算法。

1. PCI-PCI桥标号过程：第1步

使用图1-5-6的拓扑结构图，搜索过程会先发现桥1。与桥1相连的下游总线被标号为1，所以桥1的从总线标号为1而临时的下级总线标号为0XFF。这表示所有指出的PCI总线号大于等于1的1型配置地址都会穿过桥1，到达PCI总线1上。如果1型配置地址指出的总线号为1，则会被转换成0型配置周期，其他的总线号会被无变化的传送到总线1上。上面的过程正是Linux PCI初始化程序为了继续搜索PCI总线1所做的。

2. PCI-PCI桥标号过程：第2步

由于Linux使用深度优先算法，所以初始化程序会继续搜索总线1，在总线1上它找到了PCI-PCI桥2，由于在PCI-PCI桥2下面再没有PCI-PCI桥了，所以它为桥2分配了值为2的下级总线号。它与桥2从接口总线号相同。图1-5-7给出了在这一时刻初始化程序是如何为总线和PCI-PCI桥标号的。

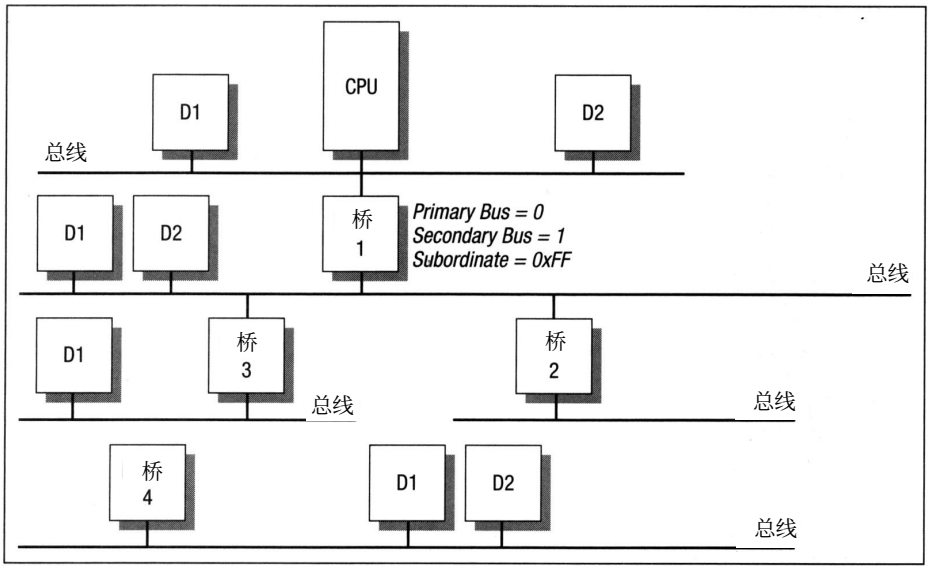

图1-5-6 配置一个PCI系统：第1步

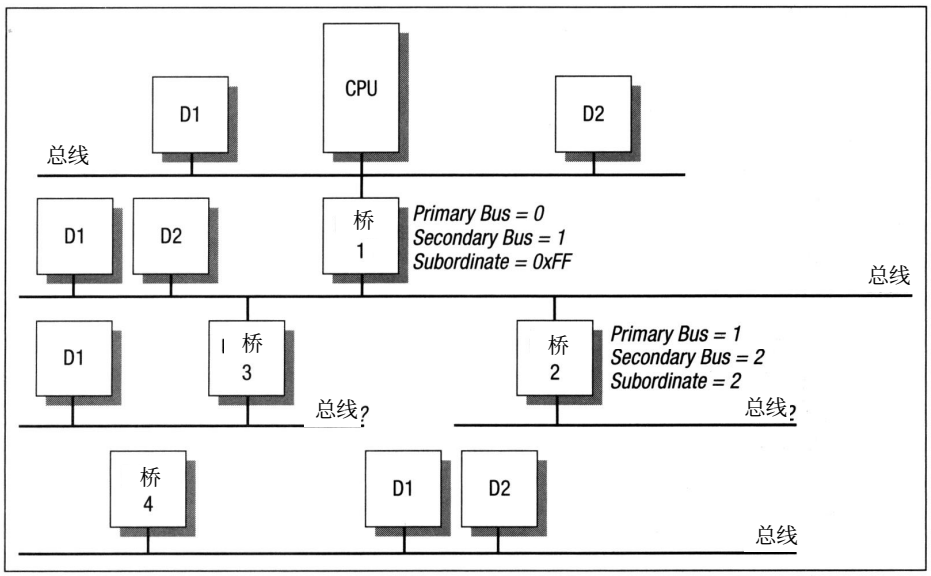

图1-5-7 配置一个PCI系统：第2步

3. PCI-PCI桥标号过程：第3步

PCI初始化程序回头继续搜索 PCI总线1，找到了另一个 PCI-PCI桥 —— 桥3。因此桥 3的主总线接口号为 1，从总线接口号为 3，临时的下级总线号为 0xFF。图1-5-8给出了目前系统的配置信息， 这时总线号为 1、2、3的1型PCI配置周期能被正确地传送到对应的 PCI总线上。

4. PCI-PCI桥标量过程：第4步

Linux开始搜索PCI-PCI桥4下游的PCI总线3。在PCI总线3上有另一个PCI桥4，因此桥4的

主总线号为3，从总线号为4，由于它是在这条分支上的最后一个桥，所以其下级总线号为 4。初始化回到PCI-PCI桥3时，给它分配4作为下级总线号。PCI初始化程序给桥1最后分配的下级总线号为4。图1-5-9给出了最终的总线和桥的标号情况。

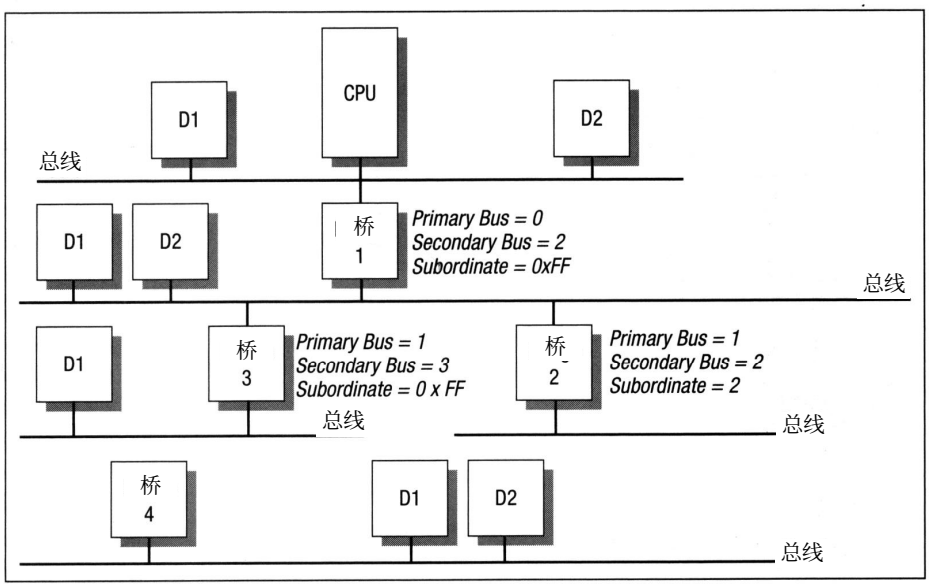

图1-5-8 配置一个PCI系统：第3步

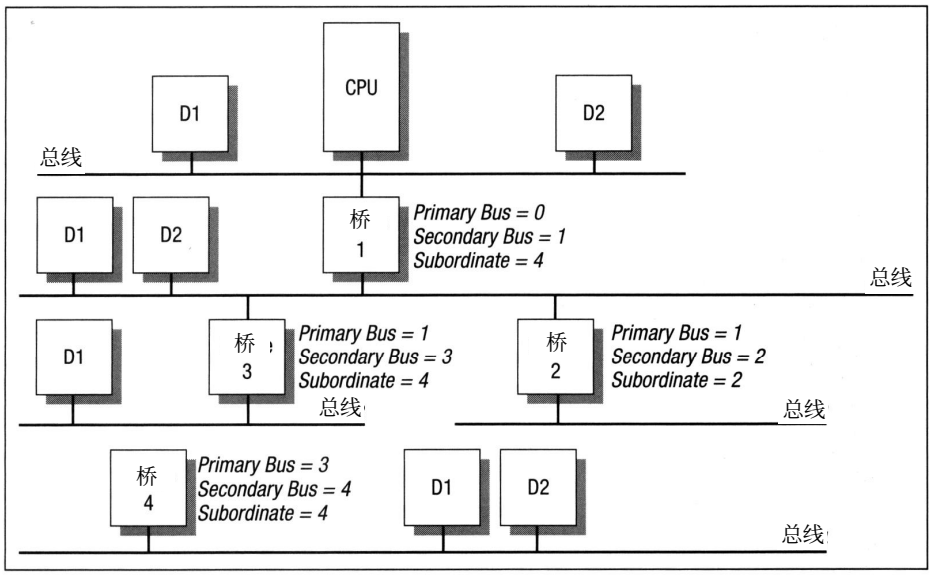

图1-5-9 配置一个PCI系统：第4步

5.6.3 PCI的BIOS函数

PCI的BIOS函数是一套跨越平台的通用标准例程。例如，在 Intel平台和Alpha AXP平台上它们都是相同的。BIOS函数允许CPU控制对所有PCI地址空间的访问，但只有 Linux内核和设

备驱动程序可以使用它们。

5.6.4 PCI修正过程

Alpha AXP 系统的PCI修正程序做的工作要比 Intel 多得多。由于Intel系统有系统BIOS，它在系统启动时运行，几乎完全配置好了 PCI系统。所以Intel系统的PCI修正程序除了对配配信息进行映射外，几乎没什么可做的。对于非 Intel的系统则需要做如下的进一步配置工作：

- 为每个设备分配PCI I/O空间和PCI 存贮器空间。
- 为系统中的每个PCI-PCI桥配置PCI I/O和存贮器地址窗口。
- 为设备设置中断线的值，通过这种方法控制设备的中断处理。

下面几小段讲述修正程序的工作过程。

1. 确定每个设备需要的PCI I/O空间和PCI存储空间的大小

修正程序对找到的每个PCI进行询问，以确定它需要的 PCI I/O空间和PCI存储地址空间的大小。为了达到这个目的，修正程序向每个设备的基址(Base Address)寄存器写入全1，然后读出。设备会在它不关注的地址返回 0，在其他位明确指出所需的地址空间大小。

系统中有两种基本类型的基址寄存器，用于指出设备寄存器是位于 PCI的I/O空间中，还是在PCI的存贮空间中。这是由寄存器的第 0位来指示的。图1-5-10给出了两种分别放在PCI的存贮空间和PCI的I/O空间的基址寄存器的形式。

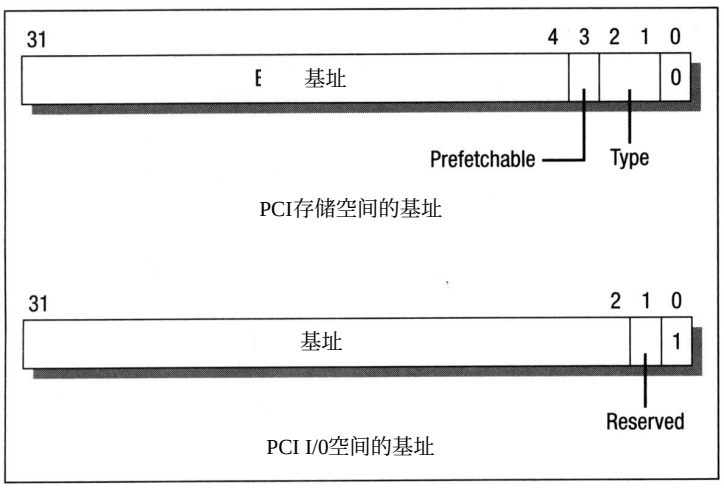

图1-5-10 PCI配置头：基址寄存器

为了确定一个基址寄存器需要多少地址空间，修正程序要先向寄存器中写入全 1，然后再从该寄存器中读取。设备会把不关注的地址位设为 0，在其余位明确给出所需的地址空间。这种方式表明使用的所有地址空间的大小都是 2的幂次，并且按自然边界对齐。例如：当你在初始化DEC片组的21142 PCI快速以太网设备时，该设备会通知你，它需要 0x100字节的PCI I/O地址空间或者 PCI存贮空间。初始化程序为它分配空间，一旦分配了空间以后， 21142的控制和状态寄存器在这些地址空间就是可见的了。

2. 为PCI-PCI桥和设备分配PCI I/O空间和PCI存贮空间

像所有的存贮器一样， PCI I/O和存贮空间是有限的，并且十分匮乏。非 Intel系统的PCI修正程序必须在考虑效率的原则下，为每个设备分配其所需数量的存储器。分配给每个设备的PCI I/O空间和存储空间，必须按自然边界对齐。例如：如果一个设备请求 0xB0大小的PCI I/O空间，那么它必须按照能被 0xB0整除的地址进行对齐。除此之外，任何一个桥的 PCI I/O空间和PCI存储空间的基址必须分别按照 4K和1M的边界进行对齐。由于任何一个下游设备的地址空间必须位于所有的上游 PCI-PCI桥存贮空间的地址范围内，所以有效的分配空间是比较困难的问题。

Linux使用的算法依赖于由 PCI设备驱动程序建立的总线 /设备树中的设备来分配 PCI的I/O地址空间，它是按照升幂的顺序来分配的。 Linux再次使用了一个反向算法来遍历由 PCI初始化程序建立的pci_bus和pci_dev数据结构。 BIOS的修正程序从由 pci_root指向的根PCI总线开始执行算法。

- 分别为当前的全局PCI I/O空间和存贮空间的基址按 4K和1M进行对齐。
- 对当前总线上的每个设备 (按所需PCI I/O空间大小的升幂顺序)执行下列操作：

—— 为其在PCI I/O空间和/或PCI存贮空间分配空间。

—— 按适合的数量调整全局 PCI I/O和存储空间的基址。

—— 允许设备使用PCI I/O和存储空间。

- 为当前总线的所有下游总线递归分配空间。注意这会改变全局 PCI I/O和存储空间的基址。
- 分别按4K和1M字节边界对当前的全局 PCI I/O和存储空间的基址执行对齐操作。在对齐操作的过程中，计算出当前 PCI-PCI桥所需的PCI I/O窗口和PCI存储窗口的大小和基址。
- 对PCI-PCI桥进行编程，将当前总线与它的 PCI I/O空间和PCI存储空间的基址、上界链接起来。
- 打开PCI-PCI桥对PCI I/O空间和PCI存储空间访问的桥接功能。这表示任何对桥的主 PCI总线的PCI I/O 空间和PCI存贮空间的访问，如果落在它的PCI I/O和PCI存储地址窗口内，都会被桥接到它的从PCI总线上。

以图1-6-1中的PCI系统作为例子， PCI修正程序将按如下的方式建立系统：

- 对齐PCI基址 PCI的I/O空间是按4K边界进行对齐的，而PCI存储空间是按1M边界对齐。这种方式允许PCI-ISA桥把到它下游的所有地址转换成ISA地址周期。

视频设备 视频设备需要2M的PCI存储空间，因此我们在当前的 PCI存储空间中为它分配基址为0x200000的2M空间。很明显它与设备要求的内存大小是自然对齐的。当前 PCI存贮空间的基址移到了0x400000，而PCI I/O空间的基址仍保持在0x4000处。

PCI-PCI桥 现在通过PCI-PCI桥为它下游的设备分配了 PCI存储空间。由于基址已经正确地对齐了，所以不必再去做基址对齐操作了。

a. 以太网设备 该设备要求0xB0字节的PCI I/O空间和PCI存储空间。因此修正程序为该设备的PCI I/O空间分配的基址为 0x4000，为PCI存储空间分配的基址为 0x400000。当前PCI存储空间的基址移到0x400B0处， PCI I/O空间的基址移到0x40B0处。

b. SCSI设备　该设备需要1K的PCI存储空间。因此在执行自然边界对齐操作后，为该设备分配的PCI存储空间的基址为0x401000。当前的PCI I/O空间的基址仍为0x40B0，而当前PCI存储空间的基址移到了0x402000。

PCI-PCI桥的PCI I/O空间和存储空间的窗口：

我们再回到上游的PCI-PCI桥，为它设置PCI I/O空间的窗口的范围为0x4000到0x40B0，它的PCI存储空间窗口的范围为0x400000到0x402000。上面的设置表示，如果是以太网设备和SCSI设备的话，PCI-PCI桥会忽略对视频设备的PCI存储空间的访问，并把它们转发到下游总线上。