[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 13/28 (2006.01)

[12] 发明专利说明书

专利号 ZL 200510076945.2

[45] 授权公告日 2008 年 8 月 13 日

[11] 授权公告号 CN 100410912C

[22] 申请日 2005.6.9

[21] 申请号 200510076945.2

[30] 优先权

[32] 2004. 8.19 [33] US [31] 10/922276

[73] 专利权人 国际商业机器公司

地址 美国纽约阿芒克

[72] 发明人 维沙尔·奇特兰简·阿斯洛特

布鲁斯·G·米利

詹姆斯·安东尼·帕弗米

詹姆斯·布里特·帕特里奇

克里斯·艾伦·施文德曼

[56] 参考文献

WO2004001615A1 2003.12.31

CN1482522A 2004.3.17

CN1462395A 2003.12.17

WO03038582A1 2003.5.8

CN1326569A 2001.12.12

审查员 韩倩倩

[74] 专利代理机构 北京市金杜律师事务所

代理人 王茂华

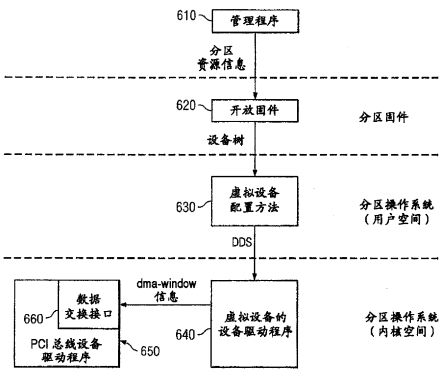
权利要求书 3 页 说明书 21 页 附图 9 页

[54] 发明名称

从一个设备驱动程序向另一个传送信息的系统和方法

[57] 摘要

本发明提供了一种为初始化用于 DMA 操作的设备而从一个设备驱动程序向另一个传送直接存储器存取(DMA)配置信息的系统和方法。 更具体地说，提供了从子虚拟设备向虚拟 I/O(VIO)总线传送用于标识为子虚拟设备分配的 DMA 地址空间的信息的机制，从而可以经由 VIO 总线为子虚拟设备初始化 DMA 操作。 根据该系统和方法，通过使用 I/O 控制(IOCTL)接口和操作系统内核服务之一，子虚拟设备的设备驱动程序直接向 VIO 总线的设备驱动程序传送有关子虚拟设备的支持 DMA 的地址空间的信息。 然后 VIO 总线设备驱动程序使用该信息设置用于来自子虚拟设备的 DMA 操作的 VIO 总线。



1. 一种方法，在数据处理系统中用于为初始化用于直接存储器存取 DMA 操作的设备而在第一设备驱动程序和第二设备驱动程序之间传送数据，包括：

在该第一设备驱动程序中确定将向该第二设备驱动程序提供的数据；

由该第一设备驱动程序调用内核空间中的数据交换接口；以及

向该数据交换接口提供该数据，其中该内核空间中的该数据交换接口将该数据写入到由该第二设备驱动程序维护并使用的数据结构中，

其中该第一设备驱动程序和设备树中的子设备相关联，并且该第二设备驱动程序和该设备树中的该子设备的双亲设备相关联，以及其中该子设备为子虚拟设备，并且该双亲设备为虚拟输入/输出总线。

2. 根据权利要求 1 的方法，其中该数据交换接口包括至少一个输入/输出控制方法，用于向设备树中的双亲设备注册和注销子设备。

3. 根据权利要求 1 的方法，其中该数据交换接口包括一个或多个内核服务，用于向设备树中的双亲设备注册和注销子设备。

4. 根据权利要求 1 的方法，进一步包括：

获取该双亲设备的设备标识符，其中调用该内核空间内的该数据交换接口包括，通过使用该设备标识符调用该数据交换接口的内核服务和输入/输出控制方法之一。

5. 根据权利要求 4 的方法，其中该双亲设备的该设备标识符是从与该子设备关联的配置方法生成的设备相关结构中获得的。

6. 根据权利要求 1 的方法，其中提供给该第二设备驱动程序的该数据包括与该子设备关联的直接存储器存取窗口的标识符。

7. 根据权利要求 6 的方法，其中该直接存储器存取窗口的该标识符是从与该子设备关联的配置方法生成的设备相关结构中获得

的。

8. 根据权利要求 6 的方法，其中该数据交换接口基于直接存储器存取窗口的该标识符，初始化用于该子设备的物理到直接存储器存取地址的转换表。

9. 根据权利要求 8 的方法，其中该直接存储器存取窗口的该标识符保存在与该双亲设备关联的存储器中，用于当来自该第一设备驱动程序的直接存储器存取请求到达时写入和读取物理到 DMA 地址的转换表条目。

10. 一种系统，用于为初始化用于直接存储器存取 DMA 操作的设备而在第一设备驱动程序和第二设备驱动程序之间传送数据，包括:

用于在该第一设备驱动程序中确定将向该第二设备驱动程序提供的数据的装置;

用于由该第一设备驱动程序调用内核空间中的数据交换接口的装置; 以及

用于向该数据交换接口提供该数据的装置，其中该内核空间中的该数据交换接口将该数据写入到由该第二设备驱动程序维护并使用的数据结构中，

其中该第一设备驱动程序和设备树中的子设备相关联，并且该第二设备驱动程序和该设备树中的该子设备的双亲设备相关联，以及其中该子设备为子虚拟设备，并且该双亲设备为虚拟输入/输出总线。

11. 根据权利要求 10 的系统，其中该数据交换接口包括至少一个输入/输出控制方法，用于向设备树中的双亲设备注册和注销子设备。

12. 根据权利要求 10 的系统，其中该数据交换接口包括一个或多个内核服务，用于向设备树中的双亲设备注册和注销子设备。

13. 根据权利要求 10 的系统，进一步包括:

用于获取该双亲设备的设备标识符的装置，其中用于调用该内核空间内的该数据交换接口的装置包括，用于通过使用该设备标识符调用该数据交换接口的内核服务和输入/输出控制方法之一的装置。

14. 根据权利要求 13 的系统，其中该双亲设备的该设备标识符是从与该子设备关联的配置方法生成的设备相关结构中获得的。

15. 根据权利要求 10 的系统，其中提供给该第二设备驱动程序的该数据包括与该子设备关联的直接存储器存取窗口的标识符。

16. 根据权利要求 15 的系统，其中该直接存储器存取窗口的该标识符是从与该子设备关联的配置方法生成的设备相关结构中获得的。

17. 根据权利要求 15 的系统，其中该数据交换接口基于直接存储器存取窗口的该标识符，初始化用于该子设备的物理到直接存储器存取地址的转换表。

18. 根据权利要求 17 的系统，其中该直接存储器存取窗口的该标识符保存在与该双亲设备关联的存储器中，用于当来自该第一设备驱动程序的直接存储器存取请求到达时写入和读取物理到 DMA 地址的转换表条目。

从一个设备驱动程序向另一个传送信息的系统和方法

技术领域

本发明一般涉及从一个设备驱动程序直接向另一个传送信息的改进的数据处理系统。更具体地说，本发明涉及为初始化用于 DMA 操作的设备而从一个设备驱动程序向另一个传送直接存储器存取（DMA）配置信息的系统和方法。

背景技术

在周边元件互连（PCI）系统体系结构中，PCI 总线具有多个与该 PCI 总线相连的子设备。直接存储器存取（DMA）操作是此类子设备经由该 PCI 总线向和从系统存储器传送数据时使用的一种机制。该 PCI 总线具有可以用来执行 DMA 操作的系统存储器的指配部分，即，支持 DMA 的地址空间或存储区域。为了执行此类 DMA 操作，为每个子设备指派支持 DMA 的地址空间内的某个地址范围。配置子设备以便在该 PCI 总线的支持 DMA 的地址空间的指配部分内执行 DMA 操作。

使用三个参数描述该支持 DMA 的存储区域：标识该支持 DMA 的存储区域开始的起始地址，指示该支持 DMA 的存储区域有多大的大小，以及与该支持 DMA 的区域有关的唯一标识符。由于该支持 DMA 的存储区域全部属于该 PCI 总线，并且该支持 DMA 的存储区域的所有部分是为每个子设备划出的，所以有关该支持 DMA 的存储区域的信息保存在开放固件设备树中的该 PCI 总线节点层上。

开放固件提供生成计算机系统的与硬件无关的引导代码，固件和设备驱动程序的能力。引导代码的基本任务是：1）构建设备树，操作系统使用该设备树来发现其可利用的设备和使用它们的方法，以及 2）引导设备驱动程序。该设备树的特定格式依赖于操作系统，但

是所有设备树有许多共性。可以利用与操作系统无关的通用语言来表示以上共性。开放固件设备树的格式就是此种通用语言。

在一个典型安装中，操作系统使用客户接口调用把开放固件设备树转换为操作系统自己的格式。开放固件设备树是设备树的普通和简单的格式。可以利用它来表达几乎任何一种操作系统的设备树中的各种条目。就像所有树一样，它是由节点组成的，某些节点为叶节点，即没有孩子的节点。除顶节点之外，每个节点有一个双亲。每个节点有一个节点名，以及一个“属性”和“方法”的列表。事实上，该节点名为一个属性，每个节点需要一个这样的属性。属性为指定的数据。开放固件并不限制操作系统怎样最终使用该指定的数据，只是保留某些名称具有某些含义：例如，“name”属性总是节点名称。该指定的数据也可以为任意类型：明确支持 string 和 integer 类型，而 int、string 和 bytes 的组合可以组成复合类型，其中 bytes 为任何长度的任意连续字节。

如上所述，对于此类开放固件设备树，有关支持 DMA 的存储区域或地址空间的信息是该开放固件设备树中的 PCI 总线节点的一部分。在配置该系统及其设备期间，把该 DMA 存储区域信息传送给 PCI 总线驱动程序，从而在配置 PCI 总线及其子设备期间，PCI 总线驱动程序为每个子设备划出该支持 DMA 的存储区域的一部分，作为有关 PCI 总线驱动程序的 DMA 操作的设备寄存器。因此，完全由 PCI 总线驱动程序判断为每个子设备指派支持 DMA 的存储区域的哪些部分。换句话说，为了向子设备分配支持 DMA 的存储区域的一部分，PCI 总线驱动程序无需来自子设备的任何附加信息。

对于虚拟输入/输出（I/O）总线和设备，引入不同情况。对于虚拟 I/O（VIO），每个设备有一个支持 DMA 的地址空间。这些支持 DMA 的地址空间并不是一个更大的地址空间的一部分，例如以上为 PCI 总线分配的支持 DMA 的地址空间。换句话说，为每个设备分配的支持 DMA 的地址空间并不是从 VIO 总线拥有的更大的支持 DMA 的地址空间中划出的，而是该设备完全拥有的支持 DMA 的地址空间。因此，

以上描述的三部分信息，即，虚拟地址空间的起始地址、大小和标识符在开放固件设备树的设备节点内。

设备配置的发生方式类似于以上描述的有关 PCI 总线的发生方式。亦即，相同的分级顺序，总线是在使用其子设备执行设备配置之前配置的。然而，问题是此刻 VIO 总线并不了解为该设备分配的支持 DMA 的地址空间，因此 DMA 操作不能开始。为此，需要一种用来通知 VIO 总线为与该 VIO 总线相连的设备分配的支持 DMA 的地址空间的机制，从而可以经由 VIO 总线到达系统存储器执行 DMA 操作。

发明内容

本发明提供从一个设备驱动程序直接向内核空间中的另一个设备驱动程序传送信息的系统和方法。根据本发明的一个示例性实施方式，提供为初始化用于 DMA 操作的设备而从一个设备驱动程序向另一个传送直接存储器存取（DMA）配置信息的机制。更具体地说，本发明的该实施方式提供从子设备向虚拟 I/O（VIO）总线传送用于标识为该子设备分配的 DMA 虚拟地址空间的信息的机制，从而可以经由 VIO 总线为该子设备初始化 DMA 操作。

根据本发明的系统和方法，该子设备的设备驱动程序通过使用 I/O 控制（IOCTL）接口和操作系统内核服务中的一个，直接向 VIO 总线的设备驱动程序传送有关该子设备的支持 DMA 的地址空间的信息。设备驱动程序加载在内核空间中。因此，在内核空间而不是用户空间中的设备驱动程序之间传送有关支持 DMA 的地址空间的信息。

在本发明的一个示例性实施方式中，所有 VIO 总线和子设备信息均存储在诸如开放固件设备树之类的设备树中，该设备树是在引导时由例如开放固件生成的。配置管理器在引导时运行，并从该设备树中读取信息，并且在它分析该设备树时，调用不同设备的合适配置方法（例如，每个设备有一个配置方法）。

在是 VIO 总线及其子设备的情况下，该配置管理器调用 VIO 总线的配置方法，然后调用其子设备的配置方法。由于该设备树中的 VIO

总线节点并不包含有关支持 DMA 的地址空间的任何信息，所以该 VIO 总线的配置方法不向 VIO 总线设备驱动程序传送任何支持 DMA 的地址空间信息。此后，在配置子设备时，有关每个子设备的支持 DMA 的地址空间的信息可从设备树中的它的节点那里获得，然后向下传送给该子设备的设备驱动程序。

本发明提供从该子设备的设备驱动程序直接向 VIO 总线的设备驱动程序传送支持 DMA 的地址空间的机制。在本发明的一个示例性实施方式中，该机制是以一个或多个输入/输出控制（IOCTL）方法的方式实现的，IOCTL 方法是 VIO 总线设备驱动程序的一部分，并且子设备的设备驱动程序可以调用它们。这些 IOCTL 方法允许子设备的设备驱动程序利用标识该子设备的 DMA 地址空间的参数调用这些 IOCTL 方法，然后将该参数写入到由 VIO 总线的设备驱动程序维护的数据结构中，以便在用该子设备执行 DMA 操作时使用。上述 IOCTL 方法可以采用一对 IOCTL 方法的形式，一个用于向 VIO 总线注册子设备，一个用于向 VIO 总线注销子设备，或者为单一注册/注销 IOCTL 方法，其中利用一个比特指示该方法是执行注册操作还是执行注销操作。

在本发明的选择性实施方式中，通过使用允许向 VIO 总线设备驱动程序注册子设备的新的内核服务，可以提供类似的功能。根据该选择性实施方式，子设备的设备驱动程序可以调用内核服务，以便向 VIO 总线设备驱动程序注册或注销子设备。对该内核服务的调用可以包括用来标识分配给该子设备的 DMA 地址空间的参数。然后该内核服务向 VIO 总线设备驱动程序提供该 DMA 地址空间信息，该 VIO 总线设备驱动程序将该信息写入到由该 VIO 总线设备驱动程序维护的数据结构中，以便在用该子设备执行 DMA 操作时使用。

优选实施方式的下述详细描述将描述本发明的这些和其它特征和优点，考虑到优选实施方式的下述详细描述，本发明的这些和其它特征和优点对本领域的一般技术人员是显而易见的。

附图说明

所附权利要求书阐述了被认为是本发明之特征的新颖特征。然而，通过参考连同附图一起阅读说明性的实施方式的下述详细描述，将更好地理解该发明本身、其优选使用方式、其它目的及其优点，其中：

图 1 是可以实施本发明的数据处理系统的框图；

图 2 是说明根据已知配置机制的 PCI 总线的 DMA 配置操作的示意图；

图 3A-3C 是根据已知配置机制的 PCI 总线及其子设备的设备树的一部分的示意图；

图 4 是根据本发明一个示例性实施方式的 VIO 总线及其子设备的设备树的一部分的示意图；

图 5 是说明根据本发明一个示例性实施方式的 VIO 总线的 DMA 配置操作的示意图；

图 6 是说明根据本发明一个示例性实施方式的配置操作的示意图；

图 7 是根据本发明的数据交换接口的一个示例性实施方式的示意图；

图 8 是根据本发明的数据交换接口的另一个示例性实施方式的示意图；以及

图 9 是一个流程图，概述了配置用于 DMA 操作的 VIO 总线及其子设备时的本发明的示例性操作。

具体实施方式

本发明提供了一种在内核空间内的设备驱动程序之间直接传送信息的机制。在一个示例性实施方式中，本发明的机制允许在配置系统期间在设备驱动程序之间直接传送设备的配置信息。按照配置用于直接存储器存取（DMA）操作的设备并在设备驱动程序之间传送 DMA 信息来描述本发明的示例性实施方式，然而，本发明并不限于仅

仅交换此类信息。相反，通过使用本发明的机制，可以交换希望在内核空间的设备驱动程序之间直接进行传送的各类信息。例如，可以在设备驱动程序之间传送的其它种类的信息包括：与中断有关的信息，与存储映象的 I/O 有关的信息，或需要共享的任何设备特有的信息。

然而，为了以下描述的目的，假设在设备驱动程序之间传送的信息为 DMA 配置信息，以便在用总线的子设备执行 DMA 操作时配置 DMA 硬件和操作系统资源。在该假设下，本发明提供以下机制，即方便子设备的设备驱动程序从它们的设备驱动程序直接向内核空间中的虚拟 I/O（VIO）总线设备驱动程序发送支持 DMA 的地址空间信息的机制。这样，为 VIO 总线设备驱动程序提供了 DMA 配置信息，可以利用该信息初始化 DMA 硬件和操作系统资源，从而经由 VIO 总线，子设备可以对系统存储器执行 DMA 操作。

现在参照附图，具体地，参照图 1，该图描述了一个可以实施本发明的数据处理系统的框图。数据处理系统 100 可以是一个包括与系统总线 106 相连的多个处理器 101、102、103 和 104 的对称多处理器（SMP）系统。例如，数据处理系统 100 可以为 IBM eServer，后者为位于纽约阿芒克的国际商业机器公司的产品，用作网络内的服务器。作为选择，可以使用单处理器系统。同时与系统总线 106 连接的是存储控制器/高速缓冲存储器 108，它提供多个局部存储器 160-163 的接口。I/O 总线桥 110 与系统总线 106 相连，并提供 I/O 总线 112 的接口。可以按所示方式集成存储控制器/高速缓冲存储器 108 和 I/O 总线桥 110。

数据处理系统 100 为逻辑分区的（LPAR）数据处理系统。因此，数据处理系统 100 可以有多个同时运行的异构操作系统（或单一操作系统的多个实例）。这些多个操作系统的每个可以有许多在其内执行的软件程序。对数据处理系统 100 进行逻辑分区，以便把不同的 PCI I/O 适配器 120-121、128-129 和 136，图形适配器 148 和硬盘适配器 149 分配给不同的逻辑分区。在这种情况下，图形适配器

148 提供用于显示设备（未示出）的连接，而硬盘适配器 149 提供用于控制硬盘 150 的连接。

因此，例如，假设数据处理系统 100 被划分为三个逻辑分区 P1、P2 和 P3。把 PCI I/O 适配器 120-121、128-129 和 136 的每一个，图形适配器 149，硬盘适配器 149，主处理器 101-104 的每一个，以及局部存储器 160-163 中的存储器分配给三个分区中的每个分区。在这些例子中，存储器 160-163 可采取双列直插存储模块（DIMM）的形式。DIMM 通常不是以每 DIMM 为基础分配给所有分区的。相反，一个分区获得该平台发现的全部存储器的一部分。例如，处理器 101，局部存储器 160-163 中的存储器的某些部分，以及 I/O 适配器 120、128 和 129 可以分配给逻辑分区 P1；处理器 102-103，局部存储器 160-163 中的存储器的某些部分，以及 PCI I/O 适配器 121 和 136 可以分配给分区 P2；处理器 104，局部存储器 160-163 中的存储器的某些部分，图形适配器 148 和硬盘适配器 149 可以分配给逻辑分区 P3。

在数据处理系统 100 内执行的每个操作系统分配给不同的逻辑分区。因此，在数据处理系统 100 内执行的每个操作系统只能访问在其逻辑分区内的那些 I/O 装置。例如，可以在分区 P1 内执行高级交互执行（AIX）操作系统的一个实例，可以在分区 P2 内执行 AIX 操作系统的第二实例（映象），而可以在逻辑分区 P3 内运行 Linux 或 OS/400 操作系统。

与 I/O 总线 112 相连的周边元件互连（PCI）主桥 114 提供到局部总线 115 的接口。多个 PCI 输入/输出适配器 120-121 可以通过 PCI 到 PCI 桥 116、PCI 总线 118、PCI 总线 119、I/O 插槽 170 和 I/O 插槽 171 连接到 PCI 总线 115。PCI 到 PCI 桥 116 提供到 PCI 总线 118 和 PCI 总线 119 的接口。PCI I/O 适配器 120 和 121 分别插入在 I/O 插槽 170 和 171 中。典型 PCI 总线实现将支持四个和八个 I/O 适配器（即，用于插式连接器的扩展插槽）。各 PCI I/O 适配器 120-121 提供数据处理系统 100 和诸如其它是数据处理系统 100 的客户机的

网络计算机之类的输入/输出设备之间的接口。

附加 PCI 主桥 122 提供用于附加 PCI 总线 123 的接口。PCI 总线 123 连接到多个 PCI I/O 适配器 128-129。PCI I/O 适配器 128-129 可以通过 PCI 到 PCI 桥 124、PCI 总线 126、PCI 总线 127、I/O 插槽 172 和 I/O 插槽 173 连接到 PCI 总线 123。PCI 到 PCI 桥 124 提供到 PCI 总线 126 和 PCI 总线 127 的接口。PCI I/O 适配器 128 和 129 分别插入在 I/O 插槽 172 和 173 中。照这样，通过各 PCI I/O 适配器 128-129，可以支持诸如调制解调器或网络适配器之类的附加 I/O 设备。照这样，数据处理系统 100 允许连接多个网络计算机。

插入在 I/O 插槽 174 中的存储映象的图形适配器 148 可以通过 PCI 总线 144、PCI 到 PCI 桥 142、PCI 总线 141 和 PCI 主桥 140 连接到 I/O 总线 112。硬盘适配器 149 可以插入在与 PCI 总线 145 相连的 I/O 插槽 175 中。该总线 145 连接到 PCI 到 PCI 桥 142，利用 PCI 总线 141，PCI 到 PCI 桥 142 连接到 PCI 主桥 140。

PCI 主桥 130 提供 PCI 总线 131 连接到 I/O 总线 112 的接口。PCI I/O 适配器 136 连接到 I/O 插槽 176，利用 PCI 总线 133，该 I/O 插槽 176 连接到 PCI 到 PCI 桥 132。PCI 到 PCI 桥 132 连接到 PCI 总线 131。同时，该 PCI 总线 131 把 PCI 主桥 130 连接到服务处理器邮箱接口和 ISA 总线访问通过逻辑 194 以及 PCI 到 PCI 桥 132。服务处理器邮箱接口和 ISA 总线访问通过逻辑 194 转发前往 PCI/ISA 桥 193 的 PCI 访问。NVRAM 存储器 192 连接到 ISA 总线 196。服务处理器 135 通过其局部 PCI 总线 195 连接到服务处理器邮箱接口和 ISA 总线访问通过逻辑 194。同时，服务处理器 135 经由多个 JTAG/I^2C 总线 134，连接到处理器 101-104。JTAG/I^2C 总线 134 为 JTAG/扫描总线（参见 IEEE 1149.1）和飞利浦 I^2C 总线的组合。然而，作为选择，可单独利用飞利浦 I^2C 总线或单独利用 JTAG/扫描总线替换 JTAG/I^2C 总线 134。主处理器 101、102、103 和 104 的所有 SP-ATTN 信号连接在一起，作为服务处理器的中断输入信号。服务处理器 135 有它自己的局部存储器 191，并且有权使用硬件 OP 面板 190。

在最初给数据处理系统 100 加电时，服务处理器 135 使用 JTAG/I^2C 总线来询问系统（主）处理器 101-104、存储控制器/高速缓冲存储器 108 和 I/O 桥 110。在完成此步骤后，服务处理器 135 具有数据处理系统 100 的详细目录和拓扑理解。同时，服务处理器 135 对通过询问主处理器 101-104、存储控制器/高速缓冲存储器 108 和 I/O 桥 110 发现的所有元件执行内置自检（BIST）、基本正确性检测（BAT）和存储器测试。由服务处理器 135 收集并报告 BIST、BAT 和存储器测试期间检测出的故障的出错信息。

如果在去除 BIST、BAT 和存储器测试期间发现的故障元件后系统资源的有意义的/有效配置仍然可行，则允许数据处理系统 100 继续向局部（主）存储器 160-163 中加载可执行代码。接着，服务处理器 135 释放主处理器 101-104 以执行加载到局部存储器 160-163 中的代码。在主处理器 101-104 执行数据处理系统 100 内的各个操作系统的代码时，服务处理器 135 进入监控和报告错误的模式。由服务处理器 135 监控的项目类型包括：例如，风扇速度和操作，热传感器，电源稳压器，以及由处理器 101-104、局部存储器 160-163 和 I/O 桥 110 报告的可恢复的和不可恢复的错误。

服务处理器135负责保存并报告与数据处理系统100中的所有被监控项目有关的出错信息。同时，服务处理器 135 根据错误类型和定义的阈值采取行动。例如，服务处理器 135 可以注意到某个处理器的高速缓冲存储器上过多的不可恢复的错误，并确定这是硬盘故障的预兆。基于上述确定，服务处理器 135 可以标记该资源以便在当前的运行会话和未来的初始程序装入（IPL）期间解除配置。IPL 有时也称为“引导”或“自举”。

可以使用可从市场上买到的各种计算机系统来实现数据处理系统 100。例如，可使用可从国际商业机器公司获得的 IBM eServer iSeries Model 840 系统来实现数据处理系统 100。此系统可以支持使用 OS/400 操作系统的逻辑分区，OS/400 操作系统也可以从国际商业机器公司获得。

本领域的一般技术人员可以理解，图 1 描述的硬件可以改变。例如，除所示硬件之外，也可以使用诸如光盘驱动器之类的其它外围设备，或代替所示硬件。所示例子并不意味着对本发明的体系结构的限制。

本发明提供了为初始化用于 DMA 操作的设备而从一个设备驱动程序向另一个传送直接存储器存取（DMA）配置信息的机制。更具体地说，本发明提供了从子设备向虚拟 I/O（VIO）总线传送用于标识为子设备分配的 DMA 虚拟地址空间的信息的机制，从而可以经由 VIO 总线为子设备初始化 DMA 操作。

根据本发明的系统和方法，子设备的设备驱动程序通过使用 I/O 控制（IOCTL）接口和操作系统内核服务中的一个，直接向 VIO 总线的设备驱动程序传送有关该子设备的支持 DMA 的地址空间的信息。因此，在内核空间而不是用户空间中的设备驱动程序之间传送有关支持 DMA 的地址空间的信息。

在本发明的一个示例性实施方式中，所有 VIO 总线和子设备信息均存储在诸如开放固件设备树之类的设备树中，该设备树是在引导时由例如开放固件生成的。配置管理器在引导时运行，并从设备树中读取信息，在它分析设备树时，调用不同设备的合适配置方法（例如，每个设备有一个配置方法）。在是 VIO 总线及其子设备的情况下，配置管理器调用 VIO 总线的配置方法，然后调用其子设备的配置方法。由于 VIO 总线节点并不包含有关支持 DMA 的地址空间的任何信息，所以 VIO 总线的配置方法不向 VIO 总线设备驱动程序传送任何支持 DMA 的地址空间信息。此后，在配置子设备时，有关每个子设备的支持 DMA 的地址空间的信息可从设备树中的它的节点那里获得，然后向下传送给该子设备的设备驱动程序。

本发明提供了从子设备的设备驱动程序直接向 VIO 总线的设备驱动程序传送支持 DMA 的地址空间的机制。在本发明的一个示例性实施方式中，该机制是以一个或多个输入/输出控制（IOCTL）方法的形式实现的，IOCTL 方法是 VIO 总线设备驱动程序的一部分，并且

子设备的设备驱动程序可以调用它们。上述 IOCTL 方法允许子设备的设备驱动程序利用标识该子设备的 DMA 地址空间的参数调用这些 IOCTL 方法，然后将该参数写入到由 VIO 总线的设备驱动程序维护的数据结构中，以便在用该子设备执行 DMA 操作时使用。上述 IOCTL 方法可以采用一对 IOCTL 方法的形式，一个用于向 VIO 总线注册子设备，一个用于向 VIO 总线注销子设备，或者为单一注册/注销 IOCTL 方法，其中利用一个比特指示该方法是执行注册操作还是执行注销操作。

在本发明的选择性实施方式中，通过使用允许向 VIO 总线设备驱动程序注册子设备的新的内核服务，可以提供类似的功能。根据该选择性实施方式，子设备的设备驱动程序可以调用内核服务，以便向 VIO 总线设备驱动程序注册或注销子设备。对该内核服务的调用可以包括用来标识分配给该子设备的 DMA 地址空间的参数。然后内核服务可以向 VIO 总线设备驱动程序提供该 DMA 地址空间信息，VIO 总线设备驱动程序将该信息写入到由该 VIO 总线设备驱动程序维护的数据结构中，以便在用该子设备执行 DMA 操作时使用。

图 2 是说明根据已知配置机制的 PCI 总线的 DMA 配置操作的示意图。正如图 2 所示，作为用于管理逻辑分区的计算系统中的分区的固件，管理程序 210 向生成设备树，例如开放固件设备树，的开放固件 220 提供分区资源信息。管理程序 210 存储有关向与数据处理系统的分区关联的各种设备分配的存储器的信息。该信息标识用于一个分区的 DMA 操作的可用系统存储器的数量。分配的 DMA 可用的系统存储器，即 dma-window（dma 窗口），传递给设备树，直至最底层的 PCI 总线，即，与输入/输出设备最靠近的 PCI 总线。另外，管理程序 210 确定向该 PCI 总线的子设备分配该 dma-window。

在配置分区时，操作系统中的配置管理器接收来自开放固件的设备树，并遍历该设备树以配置用于该分区的设备。操作系统调用该设备树中节点的合适配置方法，上述配置方法然后执行必要的内核服务调用以便实际配置要使用的设备。该配置方法使设备从定义的

（不能在该系统中使用）变成可用的（可以在该系统中使用）。如果该设备有设备驱动程序，则该配置方法把设备驱动程序加载到操作系统内核中，并使用设备相关结构（DDS）向该设备驱动程序描述设备特征。

在设备树中，由于 PCI 总线节点在其子设备节点之前出现，所以操作系统的配置管理器首先调用 PCI 总线配置方法 230，以便配置该 PCI 总线。此后，通过调用它们各自的配置方法，配置该 PCI 总线的子设备。

PCI 总线配置方法 230 接收来自开放固件 220 的设备树，并生成提供给 PCI 总线设备驱动程序 240 的设备相关结构（DDS）。DDS 包含向该设备驱动程序描述设备实例的信息。它通常包含有关设备相关属性的信息以及该设备驱动程序与该设备通信所需的其它信息。设备的 DDS 是在每次配置设备被构建的。配置方法可以用固定值、计算值和来自设备树的信息填充 DDS。

传送给 PCI 总线驱动程序 240 的 DDS 包括有关该 PCI 总线的 dma-window 的信息。接着，PCI 总线设备驱动程序 240 根据从管理程序 210 那里获得的信息，把该 dma-window 按比例分配给将向该 PCI 总线注册的子设备。此后，当配置管理器调用子设备的配置方法 250 时，子设备的配置方法 250 可以从管理程序 210 那里获得其它设备特有的信息（例如，与中断有关的信息），并向子设备的设备驱动程序 260 提供该信息，以便在配置该设备时使用（例如，可使用中断信息来初始化该设备的中断控制硬件）。

图 3A-3C 是根据已知配置机制的 PCI 总线及其子设备的设备树的一部分的示意图。正如描述的设备树显示的那样，“dma-window”属性正好在 PCI 总线节点 /pci@8000000020000001/pci@2,2 的下面。在配置总线时，PCI 总线设备驱动程序使用“dma-window”属性中包含的信息来初始化 DMA 硬件，如物理到 DMA 地址的转换表。仅仅与该设备有关的转换表被初始化。同时，使用“dma-window”信息来分配并初始化总线驱动程序实现的与 DMA 有关的内核服务使用的内

部数据结构。

PCI 总线是在分区初始化时在它们下面的子设备之前配置的。在描述的设备树中，在该 PCI 总线上提供两个集成通用串行总线(USB)设备。请注意，“dma-window”属性属于该 PCI 总线，而不属于该 PCI 总线下面的子设备。在描述的例子中，子设备，即两个 USB 设备，共享该 PCI 总线的同一“dma-window”属性。

可以把“dma-window”属性视为输入/输出端的平面地址空间。就像具有虚拟地址空间和物理地址空间的现代操作系统中的进程一样，有物理地址空间和 I/O 地址空间（也称为总线地址空间），其中在上述进程中，操作系统的虚拟存储管理器（VMM）子系统把虚拟地址转换为实际地址，或者相反。DMA 内核服务执行从物理地址到 I/O 或总线地址的转换。

利用固件对 DMA 硬件进行编程以理解上述地址空间，从而它能够进行反向转换，即，从总线地址到物理地址的转换。固件经由“dma-window”属性向 DMA 内核服务报告地址空间信息，DMA 内核服务是作为 PCI 总线驱动程序的一部分实现的。

“dma-window”有三个部分：逻辑总线标识符，起始地址，以及窗口大小。在该描述的示例中，这些部分分别为：0x00000001、0x02000000 以及 0x10000000。因此，用于 /pci@8000000020000001/pci@2,2 总线的全部 DMA 总线地址空间的大小为 256MB，其中以总线地址 0x02000000 开始。两个 USB 设备共享 256MB 空间，以执行它们的 DMA 操作。从 0x02000000 开始到 0x02000000+0x10000000-1 的每个地址依靠 DMA 内核服务转换地址的方式转换为物理存储器中的某些地址。

从图 3A-3C 所示例子中获得的主要观察是，“dma-window”属性属于设备树中的 PCI 总线节点，并且 PCI 总线节点是在 USB 设备之前配置的。这使得在 USB 设备驱动程序能够发起第一个 DMA 操作之前，该 DMA 硬件是“准备好的”。然而，对于虚拟输入/输出（VIO），由于用于 DMA 操作的地址空间是分配给子虚拟设备本身的，并且不

是 VIO 总线拥有的，所以不在子虚拟设备的第一个 DMA 操作之前配置用于 DMA 操作的 DMA 硬件和操作系统资源。亦即，为每个子虚拟设备配备利用设备树中的子虚拟设备的节点中的“dma-window”属性表示的它们自己的全部 4GB 的总线地址空间，而不是子虚拟设备共享 VIO 总线拥有的 “dma-window”。

在配置 VIO 总线时，VIO 总线设备驱动程序不了解每个子虚拟设备的“dma-window”。显然，只有在 VIO 总线设备驱动程序完成 DMA 设置从而配置了供子虚拟设备使用的 DMA 硬件和操作系统资源，例如，地址转换表等后，子虚拟设备才能执行 DMA 操作。本发明解决了该问题，并且提供了在用于子虚拟设备和用于它们“连接到的”VIO 总线的设备驱动程序之间传送地址空间信息的机制，以便使得在由子虚拟设备发起第一个 DMA 操作时，DMA 硬件是“准备好”的。

图 4 是根据本发明一个示例性实施方式的 VIO 总线及其子虚拟设备的设备树的一部分的示意图。在图 4 所示设备树的一部分的例子中，在用于 VIO 总线的设备树节点，即/vdevice 节点，和子虚拟设备，即，/vdevice/v-scsi-host@30000002 中，“dma-window”属性属于虚拟设备节点。所描述的虚拟设备有两个窗口，每一个都有逻辑总线标识符、起始地址和窗口大小。在本例子中，dma-window 的大小为 256MB，从支持 DMA 的地址 0 开始。同时请注意，起始地址和大小为 64 比特值，而逻辑总线标识符为 32 比特值。起始地址的大小、窗口数量的大小和逻辑总线标识符是操作系统和体系结构特有的。在虚拟 SCSI 目标设备的特定例子中，将向 VIO 总线驱动程序传送有关两个窗口的信息，然而，实际上仅仅使用第一个 dma-window 信息来初始化 DMA 硬件和内部数据结构。

图 5 是说明根据本发明一个示例性实施方式的 VIO 总线的 DMA 配置操作的示意图。正如图 5 所示，该配置操作类似于图 2 的配置操作，只是由虚拟输入/输出（VIO）总线的配置方法 530 向 VIO 总线设备驱动程序 540 提供的设备相关结构（DDS）不包括 VIO 总线或每个子设备的 dma-window 属性。因此，VIO 总线设备驱动程序不能

给子虚拟设备分配部分 dma-window，并且没有有关每个子虚拟设备的 dma-window 的任何信息。

相反，子虚拟设备的 dma-window 信息存在于开放固件 520 生成的设备树内的子虚拟设备节点中。因此，在调用子虚拟设备的配置方法 550 时，子虚拟设备的配置方法 550 可以从开放固件设备树中获得该子虚拟设备的必要的 dma-window 信息，生成带有该信息的 DDS 并向子虚拟设备的设备驱动程序 560 提供该 DDS。然而，由于 VIO 总线设备驱动程序 540 没有有关分配给子设备的 dma-window 的任何信息，所以不能为来自该子虚拟设备的 DMA 操作配置与 VIO 总线关联的 DMA 硬件和操作系统资源。

图 6 是说明根据本发明一个示例性实施方式的配置操作的示意图。在该描述的例子中，假设已经按照与参照图 5 描述的方式类似的方式配置了 VIO 总线。正如图 6 所示，配置操作类似于图 5 中配置 VIO 总线的子虚拟设备的配置操作。亦即，子虚拟设备的配置方法 630 接收来自开放固件 620 的设备树，并生成描述该子虚拟设备的 DDS，其中该设备树是根据从管理程序 610 那里获得的分区资源信息生成的。该 DDS 包括子虚拟设备的 dma-window 信息。向子虚拟设备的设备驱动程序 640 提供该 DDS。

然而，根据本发明，为了配置用于来自子虚拟设备的 DMA 操作的 DMA 硬件和操作系统资源，提供了从分区操作系统内核空间中的子虚拟设备驱动程序 640 直接向 VIO 总线设备驱动程序 650 传送 dma-window 信息的机制。在图 6 中，该机制称为数据交换接口 660。子虚拟设备的设备驱动程序 640 利用传送的标识子虚拟设备的 dma-window 的参数，调用数据交换接口 660。接着，数据交换接口 660 将该信息写入到由虚拟总线驱动程序 650 维护的数据结构中，并利用该信息配置 DMA 硬件和操作系统资源。子虚拟设备驱动程序现在可以用于 DMA 操作了。

图 7 是根据本发明的数据交换接口的一个示例性实施方式的示意图。正如图 7 所示，在本发明的一个示例性实施方式中，数据交

换接口包括称为虚拟设备注册（VDEVREG）的第一输入/输出控制（IOCTL）方法 720，用于向虚拟输入/输出（VIO）总线设备驱动程序 740 注册子虚拟设备。另外，提供称为虚拟设备注销（VDEVUNREG）的第二 IOCTL 方法 730，用于向 VIO 总线设备驱动程序 740 注销子虚拟设备。

子虚拟设备驱动程序 750 可以调用上述 IOCTL 方法 720 和 730，以便向 VIO 总线设备驱动程序 740 注册或注销子虚拟设备。为了能够调用上述 IOCTL 方法 720 和 730，子虚拟设备驱动程序 750 必须了解“连接”该子虚拟设备的 VIO 总线的设备标识符。子设备的配置方法可以基于从设备树中获得的信息在 DDS 中提供该信息。通过了解 VIO 总线的设备标识符，子虚拟设备的设备驱动程序 750 可以在该特定总线设备的设备驱动程序上调用 IOCTL 方法 720 和 730，以便向 VIO 总线注册/注销子虚拟设备。

在调用上述 IOCTL 方法 720 和 730 时，子设备驱动程序 750 以参数的方式传送借助配置方法生成的 DDS 从子虚拟设备的配置方法中获得的 dma-window 信息。接着，IOCTL 方法 720 和 730 使用该 dma-window 信息，或者使用分配的 dma-window 为子虚拟设备的 DMA 操作初始化合适的 DMA 硬件和操作系统资源（注册），或者释放上述操作系统资源和 DMA 硬件（注销）。例如，为了执行上述操作，虚拟设备驱动程序使用 VIO 总线的设备标识符（例如，主和辅号码）调用 fp_open()方法来打开 VIO 总线驱动程序。如果 fp_open()成功，则虚拟设备驱动程序调用 fp_ioctl()向 VIO 总线驱动程序进行注册或注销。

用于注册的 IOCTL 取得“dma-window”信息，并初始化用于该设备的物理到 DMA 地址的转换表。例如，它将表条目中的许可位初始化为“不能访问”。周边主桥（PHB）硬件读取该表以执行实际的 DMA 操作。除转换表之外，VIO 总线驱动程序基于“dma-window”信息分配并初始化 VIO 总线驱动程序实现的与 DMA 有关的内核服务使用的内部数据结构。同时，注册接口在 VIO 总线驱动程序的存储器中保

存“dma-window”信息，从而在来自虚拟设备驱动程序的DMA请求到达时，可以有效读取转换表条目或写入到转换表条目中。在虚拟设备驱动程序调用它们时，诸如d_map_page()和d_map_list()之类的DMA内核服务写出转换信息。上述服务确保映射是仅仅在该特定驱动程序的dma-window，即，dma-window指定的地址范围内执行的。同样，d_unmap_page()和d_unmap_list()服务仅仅使指定dma-window内的转换表条目无效。

注销接口释放在注册时分配的所有内部数据结构。只有在证实没有正在进行的或未决的DMA操作后才这么做。因此，既不期待也不允许来自该驱动程序的更多的DMA操作。

注册和注销通常为在配置和解除配置时执行的一次操作。当设备在使用时，在注册和注销操作之间可以执行许多次映射和解除映射操作。在完成注册之前和注销之后，不允许映射和解除映射操作。

在选择性实施方式中，可以给单一IOCTL方法配备一个参数，而不是具有两个新的IOCTL方法720和730，该参数标识该IOCTL是向VIO总线设备驱动程序注册还是注销子设备。在此种实施方式中，子虚拟设备的设备驱动程序可以调用该IOCTL方法，并以参数的方式提供dma-window信息和该输入参数的值，标识是向VIO总线设备驱动程序注册还是注销子虚拟设备。IOCTL将包括包含注册和注销操作在内的功能，其中基于输入参数的设置选择注册或注销。

本发明的IOCTL方法实施方式提供“清除”方法，以解决能轻易实现的在内核空间中的设备驱动程序之间传送信息的问题。然而，上述IOCTL方法必须在VIO总线设备驱动程序中提供。因此，需要从子虚拟设备中获取信息的每个VIO总线设备驱动程序必须实现上述IOCTL方法，以便能够从子虚拟设备中获取必要的信息。

在使用IOCTL接口时有两个主要困难。第一，传送信息的设备驱动程序需要了解目标设备驱动程序的标识符。这未必是一项简单任务，因为它需要访问目的设备，即向其传送信息的设备驱动程序的设备配置信息。源设备驱动程序，即传送信息的设备驱动程序，也

并无权从内核空间中访问该信息，因此，它的配置方法需要访问该信息并作为 DDS 的一部分向下传送。该特殊步骤可能需要改变源设备的配置方法。

第二，操作系统还需要在内核空间中提供 open()、close()和 ioctl()接口。这些是文件系统接口，通常用于来自用户空间的内核空间操作。换句话说，用户空间应用程序可以调用内核扩展或设备驱动程序的 IOCTL 来完成某些特殊任务。在调用 ioctl()之前，用户空间应用程序用其名称打开内核扩展。例如，open("/dev/hdisk0", O_RDONLY)（如上所述，在内核空间中由设备标识符代替更自然更方便的设备名称，因为诸如"/dev/hdisk0"之类的名称在内核空间中没有直接含义）。AIX 操作系统以 fp_open()、fp_ioctl()和 fp_close()提供上述接口。然而，其它操作系统未必在内核空间中提供此类接口。

图 8 是根据本发明的数据交换接口的另一个示例性实施方式的示意图。提供图 8 所示的数据交换接口，作为避免与提供新的 IOCTL 方法关联的问题的选择性实施方式。正如图 8 所示，在该选择性实施方式中，操作系统内核 820 输出可以由子虚拟设备的设备驱动程序 810 调用的两个新的内核服务 830-840，或者可选择地，输出带有操作参数的单个内核服务，而不是提供两个新的 IOCTL 方法，其中该操作参数类似于如上参照单个 IOCTL 方法描述的参数。操作系统内核调用总线特有的注册/注销例程 860-870。操作系统内核 820 切断 bus-id 以查找正确的总线特有的例程，并利用函数指针调用它。bus-id 是识别需要调用哪个总线驱动程序的注册或注销服务的一种方式。例如，系统可以支持多种类型的总线，如 ISA、PCI、VIO 等。每个总线有其特有的总线驱动程序，并且可以实现注册和注销服务。内核服务基于 bus-id 确定总线驱动程序。诸如 d_map_init()之类的其它内核服务也使用 bus-id，因此，设备驱动程序总是可以使用 bus-id。尽管本发明根据 VIO 总线驱动程序的使用描述，但是本发明可以使用包括 ISA、PCI 等在内的任何类型的总线。

一旦确定总线驱动程序，内核服务需要调用总线驱动程序中的函数。在优选实施方式中，这是用于借助于函数指针调用函数的标准 C 接口实现的。函数指针是方便的，因为引导时内核只需保存每个总线驱动程序的两个函数指针。在图 2 所示的配置期间，总线驱动程序借助于内核服务向内核传送函数指针。例如，在 AIX 操作系统中，可以利用总线驱动程序用来向内核注册各种信息的 bus_register() 内核服务，向内核传送上述函数指针。由于总线驱动程序是在它们子设备之前配置的，所以在配置子设备时已经建立了函数指针。

总线特有的例程 860-870 向 VIO 总线注册或注销虚拟子设备。在本示例性实施方式中，VIO 总线驱动程序的注册/注销函数执行以下任务，初始化物理到 DMA 地址的转换表以及由 VIO DMA 内核服务使用的内部数据结构。另一方面，内核服务只是传递接口，输出该接口以便任何一种设备驱动程序或内核扩展调用。如上所述，内核服务仅仅使用函数指针调用合适的总线驱动程序的注册/注销函数。内核服务不关心该信息的特性或其被使用的方式，并且不对该信息进行任何确认。所有确认是由内核服务借助于函数指针调用的总线驱动程序特有的例程完成的。成功和失败返回代码也可以传递，即，如果总线驱动程序特有的例程由于某些原因失败，则内核服务可以将失败上传给调用设备驱动程序。

把内核服务输出到全局内核名字空间中。在本实施方式中，它们称为 register_device()和 unregister_device()。通过把这些内核服务输出到全局内核名字空间中使得所有内核扩展和设备驱动程序都可以使用它们。相反，总线驱动程序特有的例程在驱动程序的内部，只有绑定到该驱动程序的对象才能使用它们。换句话说，它们只在驱动程序的名字空间内可见，不具备类似于内核服务的全局可见性。因此，其它设备驱动程序和内核扩展不能直接调用它们。

图 9 是一个流程图，概述了配置用于 DMA 操作的 VIO 总线及其子设备时的本发明的示例性操作。可以理解，可利用计算机程序指令实现流程图所示的每个方块，以及流程图所示的方块的组合。可以

向处理器或其它可编程的数据处理装置提供上述计算机程序指令以产生机器，以至在处理器或其它可编程的数据处理装置上执行的指令创建用于实现流程图方块中指定的功能的装置。也可以把上述计算机程序指令存储到计算机可读存储器或存储介质中，这些计算机程序指令可以控制处理器或其它可编程的数据处理装置以特定方式运行，以至计算机可读存储器或存储介质中存储的指令产生一个包括实现流程图方块中指定的功能的指令装置的产品。

因此，流程图所示的方块支持用于执行指定功能的装置的组合，用于执行指定功能的步骤和用于执行指定功能的程序指令装置的组合。同时可以理解，可以利用执行指定功能或步骤的基于专用目的硬件的计算机系统，或利用专用目的硬件和计算机指令的组合，实现流程图所示的每个方块，以及流程图所示中的方块的组合。

正如图 9 所示，该操作首先通过调用与子虚拟设备关联的配置方法来初始化子虚拟设备的配置（步骤 910）。为了本描述的目的，假设已经配置了 VIO 总线。子虚拟设备的配置方法读取固件生成的设备树（步骤 920），并生成用于子虚拟设备的设备相关结构（DDS）（步骤 930）。该 DDS 包括连接子虚拟设备的 VIO 总线的标识符以及与子虚拟设备关联的 dma-window。将 DDS 发送给子虚拟设备的设备驱动程序（步骤 940）。

接着，子虚拟设备的设备驱动程序依据上面的特定实施方式，调用 VIO 总线设备驱动或操作系统内核的数据交换接口（步骤 950）。子虚拟设备的设备驱动程序向操作系统内核/VIO 总线设备驱动程序的数据交换接口传送数据（步骤 960）。例如，该数据可以指示子虚拟设备的 dma-window。

数据交换接口向 VIO 总线注册/注销子虚拟设备（步骤 970），并初始化用于子虚拟设备的 DMA 操作的 DMA 硬件和操作系统资源（步骤 980）。然后该操作终止。

这样，本发明提供了为了向用于 DMA 操作的 VIO 总线注册/注销子虚拟设备而从子虚拟设备的设备驱动程序直接向 VIO 总线设备驱

动程序传送 dma-window 信息的机制。尽管上述实施方式是按照配置用于子虚拟设备的 DMA 操作的系统来描述的，但是本发明并不限于此。相反，可使用本发明的机制在操作系统内核空间中的设备驱动程序之间传送任何合适信息。因此，也可以使用本发明来传送需要与另一个内核扩展或驱动程序共享的任意一种设备特有的信息，而不是仅仅使用本发明配置用于带有 VIO 总线的 DMA 操作的子虚拟设备。

重要的是请注意，尽管本发明是在全功能数据处理系统的情况下描述的，但是本领域的一般技术人员可以理解，可以以指令的计算机可读介质的形式，以及多种形式分布本发明的进程，并且不论实际完成分布的信号承载介质的特定类型，本发明同样适用。计算机可读介质的例子包括可记录类型的介质，如软盘、硬盘驱动器、RAM、CD-ROM、DVD-ROM，以及传输类型的介质，如数字和模拟通信链路，使用诸如射频和光波传输之类的传输形式的有线或无线通信链路。计算机可读介质可以采取编码的格式的形式，其中当在特定数据处理系统中实际使用时进行解码。

提供本发明的说明书的目的是为了说明和描述，而不是用来穷举或将本发明限制为所公开的形式。对本领域的一般技术人员而言，许多修改和变更都是显而易见的。选择并描述实施方式是为了更好地解释本发明的原理，其实际应用，并使本领域的其他一般技术人员理解带有各种修改的各种实施方式的本发明同样适用于设想的特定用途。

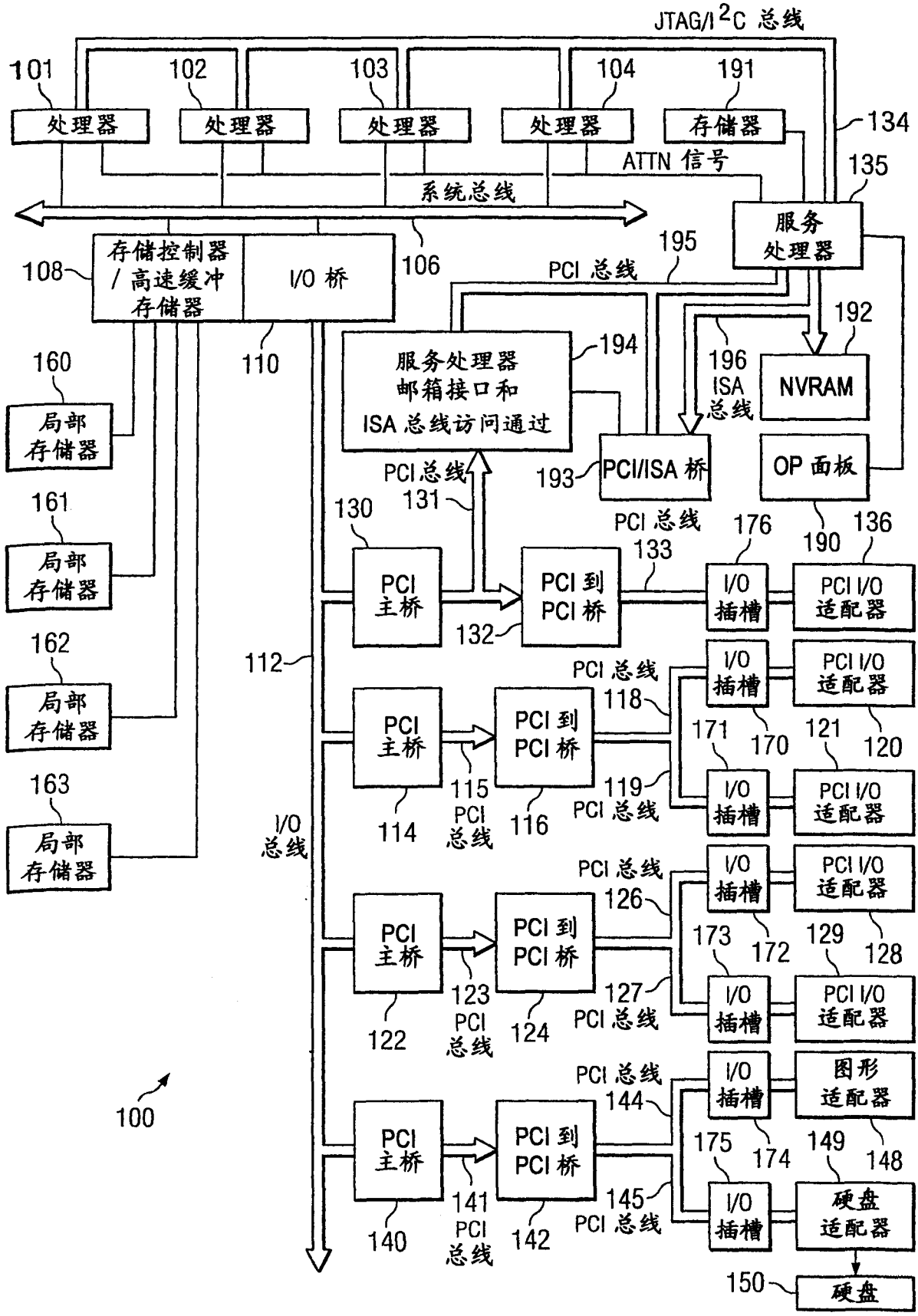

JTAG/I2C 总线
101
102
103
104
191
处理器
处理器
处理器
处理器
存储器
134
ATTN 信号
135
系统总线
服务
处理器
106
195
PCI 总线
108
存储控制器
/ 高速缓冲
存储器
I/O 桥
192
服务处理器
邮箱接口和
ISA 总线访问通过
194
196
ISA
总线
NVRAM
160
局部
存储器
110
PCI总线
193
PCI/ISA 桥
OP 面板
161
局部
存储器
130
131
PCI 总线
176
190
136
133
PCI
主桥
PCI 到
PCI 桥
I/O
插槽
PCI I/O
适配器
132
162
局部
存储器
112
PCI 总线
118
I/O
插槽
PCI I/O
适配器
171
170
121
120
PCI
主桥
PCI 到
PCI 桥
163
局部
存储器
115
PCI
总线
119
PCI 总线
114
116
I/O
总线
PCI 总线
126
I/O
插槽
PCI I/O
适配器
173
172
129
128
PCI
主桥
PCI 到
PCI 桥
123
PCI
总线
127
PCI 总线
122
124
PCI 总线
144
I/O
插槽
图形
适配器
175
174
149
148
100
PCI
主桥
PCI 到
PCI 桥
141
PCI
总线
145
PCI 总线
140
142
I/O
插槽
硬盘
适配器
150
硬盘

图 1

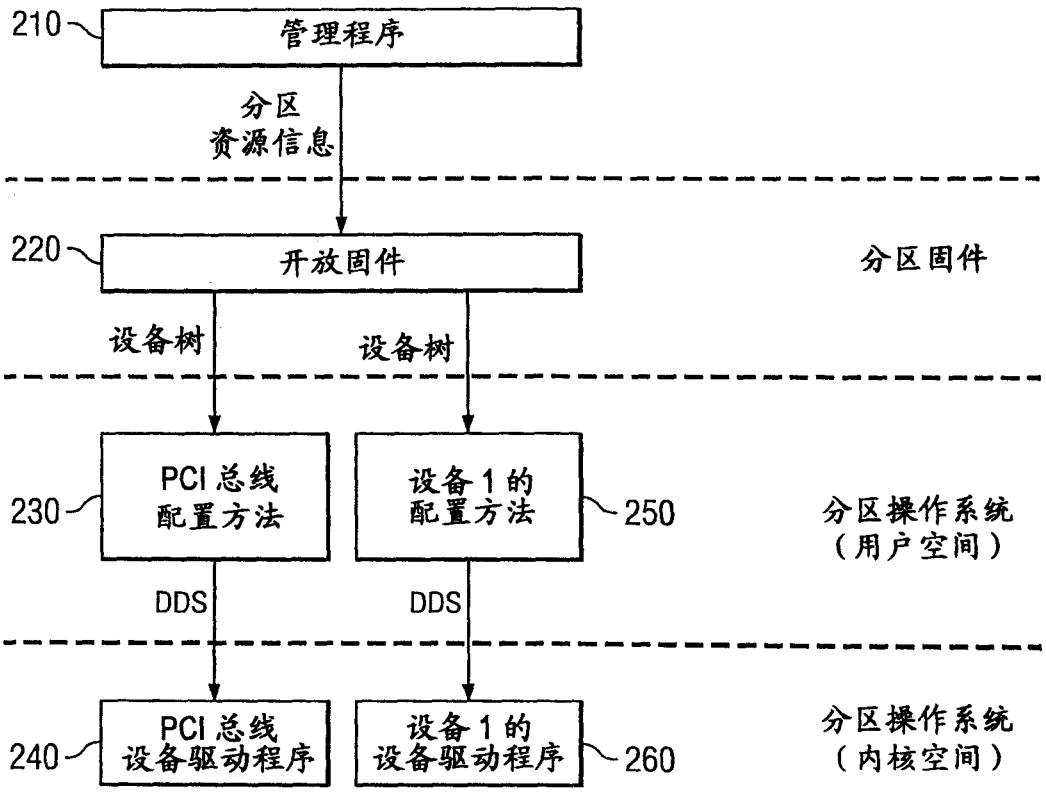

210
管理程序
分区
资源信息
220
开放固件
分区固件
设备树
设备树
PCI 总线
配置方法
230
设备 1 的
配置方法
250
分区操作系统
（用户空间）
DDS
DDS
PCI 总线
设备驱动程序
240
设备 1 的
设备驱动程序
260
分区操作系统
（内核空间）

图 2

/pci@800000020000001/pci@2,2 <<<< 这是由 PCI 到 PCI 桥生成的总线的节点（没有示出全部属性）

```
                70636900                                [pci.............]
device_type
                70636900                                [pci.............]
reg
                00001000 00000000 00000000 00000000     [................]
                00000000                                [................]
#address-cells
                00000003                                [................]
#size-cells
                00000002                                [................]
compatible
                70636931 3031342c 31383800 70636963     [pci1014,188.pcic]
                6c617373 2c303630 3430660               [lass,06040f.....]
vendor-id
                00001014                                [................]
device-id
                00000188                                [................]
revision-id
                00000002                                [................]
class-code
                0006040f                                [................]
devsel-speed
                00000002                                [................]
66mhz-capable
cache-line-size
                00000020                                [................]
bus-range
                00000040 00000043                       [...@...c.......]
ranges
                81000000 00000000 000f0000 81000000     [................]
                00000000 000f0000 00000000 00010000     [................]
                82000000 00000000 d0000000 82000000     [................]
                00000000 d0000000 00000000 10000000     [................]
#interrupt-cells
                00000001                                [................]
interrupt-map-mask
                00000800 00000000 00000000 00000007     [................]
interrupt-map
                00000800 00000000 00000000 00000001     [................]
                25000001 00000053 00000001              [%....S.........]
dma-window
                00000001 02000000 10000000              [................]
```

图 3A

```
/pci@800000020000001/pci@2,2/usb@1      <<<< 集成 USB，端口 #1，在上面的 PCI
总线之上
ibm,loc-code
                55373837 392e3030 312e3033 39353535    [U7879.001.039555]
                382d5031 00                            [8-P1.....]
vendor-id
                00001033                               [...3.....]
device-id
                00000035                               [...5.....]
revision-id
                00000043                               [...C.....]
class-code
                000c0310                               [.....]
interrupts
                00000001                               [.....]
min-grant
                00000001                               [.....]
max-latency
                0000002a                               [...*.....]
subsystem-vendor-id
                00001033                               [...3.....]
subsystem-id
                00000035                               [...5.....]
devsel-speed
                00000001                               [.....]
built-in
name
                75736200                               [usb .....]
compatible
                70636931 3033332c 33350070 63693130    [pci1033,35.pci10..]
                33332c33 35007063 69636c61 73732c30    [33,35.pciclass,0...]
                63303331 3000                          [c0310 .....]
reg
                00480800 00000000 00000000 00000000    [.H.....]
                00000000 02480810 00000000 00000000    [.....H.....]
                00000000 00001000                      [.....]
#address-cells
                00000001                               [.....]
#size-cells
                00000000                               [.....]
assigned-addresses
                82480810 00000000 c0001000 00000000    [.H.....]
                00001000                               [.....]
```

图 3B

```
/pci@800000020000001/pci@2,2/usb@1,1    <<<< 集成 USB，端口 #2，在上面的 PCI
总线之上
ibm,loc-code
                55373837 392e3030 312e3033 39353535    [U7879.001.039555]
                382d5031 00                            [8-P1.....]
vendor-id
                00001033                               [...3.....]
device-id
                00000035                               [...5.....]
revision-id
                00000043                               [...C.....]
class-code
                000c0310                               [.....]
interrupts
                00000002                               [.....]
min-grant
                00000001                               [.....]
max-latency
                0000002a                               [...*.....]
subsystem-vendor-id
                00001033                               [...3.....]
subsystem-id
                00000035                               [...5.....]
devsel-speed
                00000001                               [.....]
built-in
name
                75736200                               [usb .....]
compatible
                70636931 3033332c 33350070 63693130    [pci1033,35.pci10..]
                33332c33 35007063 69636c61 73732c30    [33,35.pciclass,0...]
                63303331 3000                          [c0310 .....]
reg
                00480900 00000000 00000000 00000000    [.H.....]
                00000000 02480910 00000000 00000000    [.....H.....]
                00000000 00001000                      [.....]
#address-cells
                00000001                               [.....]
#size-cells
                00000000                               [.....]
assigned-addresses
                82480910 00000000 c0000000 00000000    [.H.....]
                00001000                               [.....]
```

图 3C

/vdevice << 虚拟（VIO）总线节点。注意它没有“dma-window”属性。

```
  name
                76646576 69636500                    [vdevice ..........]
  device_type
                76646576 69636500                    [vdevice ..........]
  compatible
                49424d2c 76646576 69636500           [IBM, vdevice .....]
  #address-cells
                00000001                             [.................]
  #size-cells
                00000000                             [.................]
  #interrupt-cells
                00000002                             [.................]
  interrupt-ranges
                000a0000 00000009                    [.................]
  interrupt-controller
```

/vdevice/v-scsi-host@30000002 << 虚拟设备，在该例子中是虚拟 SCSI 目标设备。

```
  name
                762d7363 73692d68 6f737400           [v-scsi-host.......]
  device_type
                762d7363 73692d68 6f737400           [v-scsi-host.......]
  compatible
                49424d2c 762d7363 73692d68 6f737400  [IBM, v-scsi-host...]
  ibm, loc-code
                55393131 372e3537 302e3131 30303432  [U9117.570.110042]
                412d5631 2d433200                    [A-V1-C2 ..........]
  reg
                30000002                             [0.................]
  interrupts
                000a002 00000000                     [.................]
  ibm, #dma-address-cells
                00000002                             [.................]
  ibm, #dma-size-cells
                00000002                             [.................]
```

dma-window << 在该例子中，有两个逻辑总线标识符、开始地址、以及窗口大小的 3 元组。

```
                19000002 00000000 00000000 00000000  [.................]
                10000000 13000002 00000000 00000000  [.................]
                00000000 10000000                    [.................]
  ibm, vserver
```

图 4

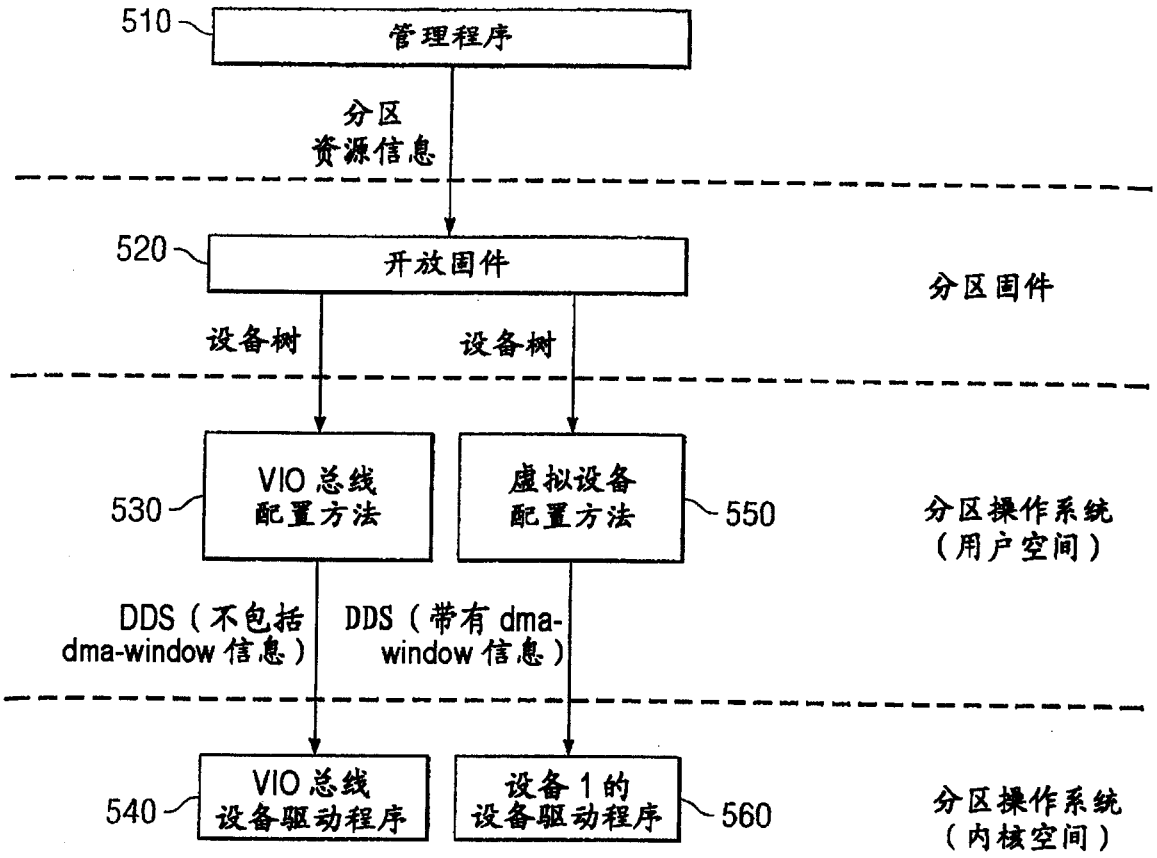

510
管理程序
分区
资源信息
520
开放固件
分区固件
设备树
设备树
VIO 总线
配置方法
530
虚拟设备
配置方法
550
分区操作系统
（用户空间）
DDS（不包括
dma-window 信息）
DDS（带有 dma-
window 信息）
VIO 总线
设备驱动程序
540
设备 1 的
设备驱动程序
560
分区操作系统
（内核空间）

图 5

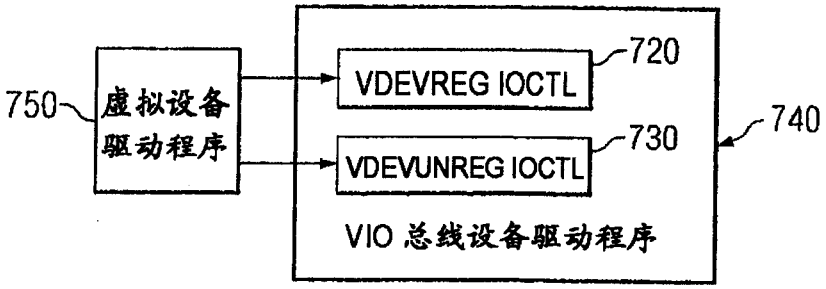

750
虚拟设备
驱动程序
VDEVREG IOCTL
720
VDEVUNREG IOCTL
730
740
VIO 总线设备驱动程序

图 7

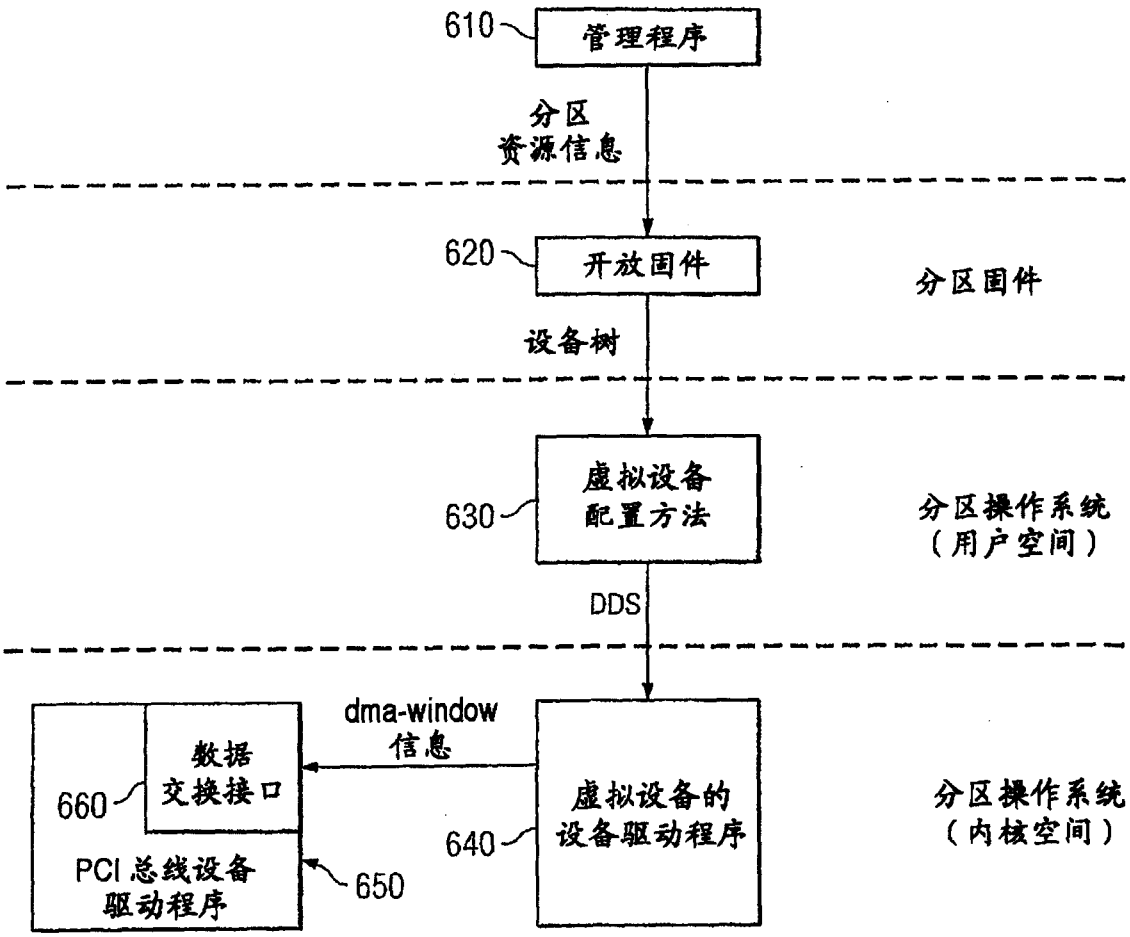

610
管理程序
分区
资源信息
620
开放固件
分区固件
设备树
630
虚拟设备
配置方法
分区操作系统
（用户空间）
DDS
dma-window
信息
660
数据
交换接口
PCI 总线设备
驱动程序
650
640
虚拟设备的
设备驱动程序
分区操作系统
（内核空间）

图 6

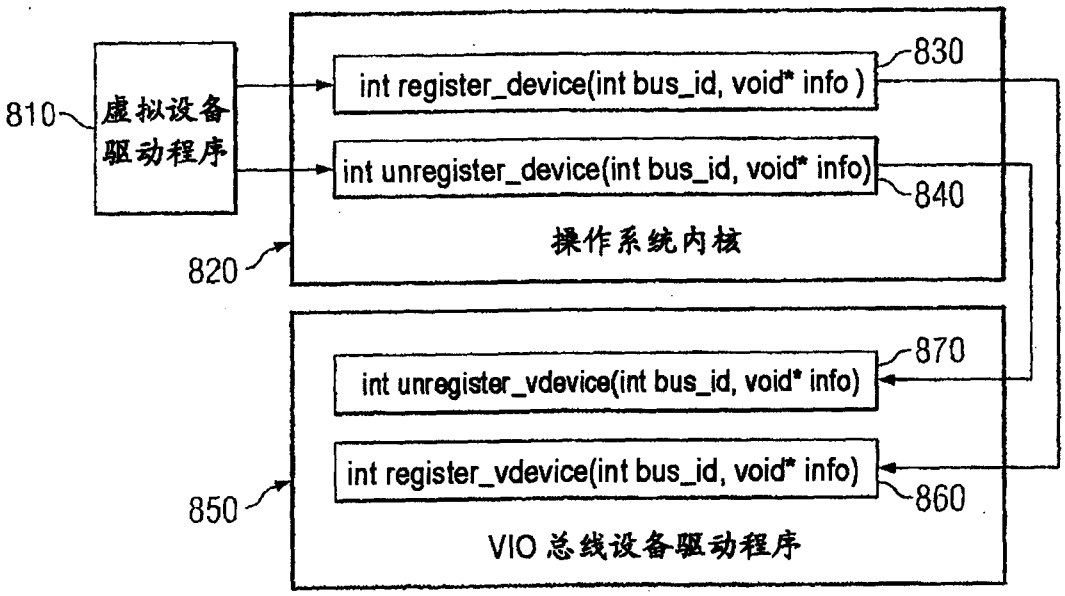

810
虚拟设备
驱动程序
int register_device(int bus_id, void* info)
830
int unregister_device(int bus_id, void* info)
840
操作系统内核
820
int unregister_vdevice(int bus_id, void* info)
870
int register_vdevice(int bus_id, void* info)
860
850
VIO 总线设备驱动程序

图 8

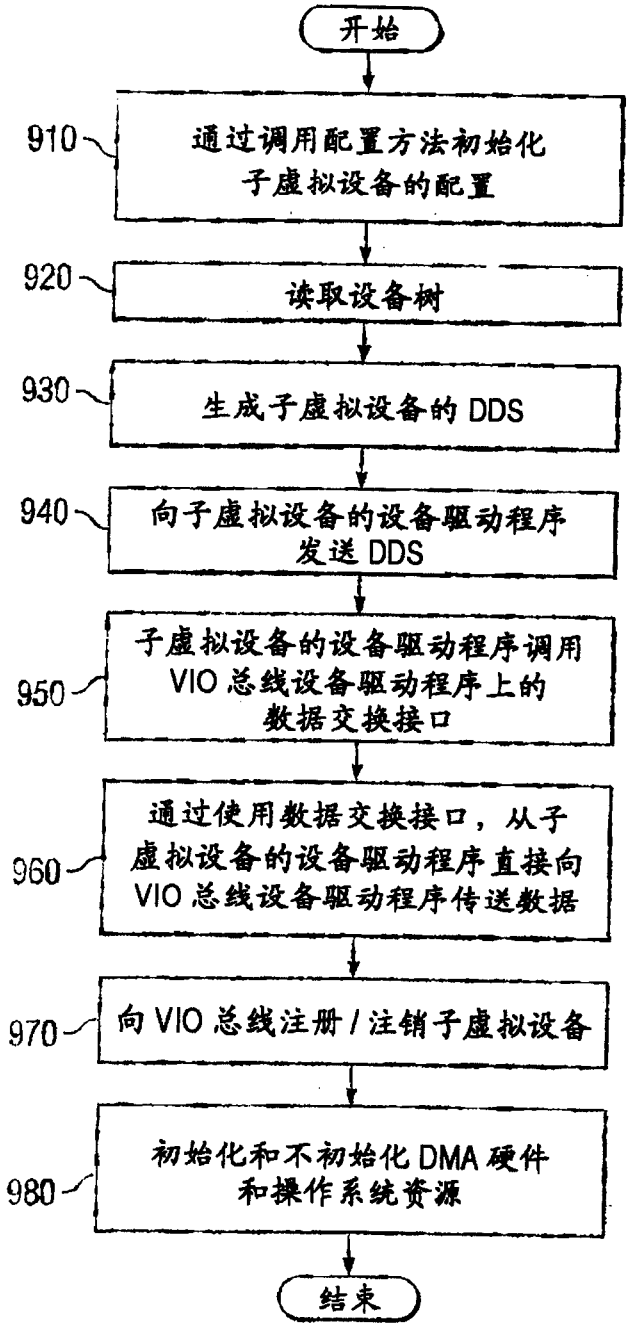

开始
910
通过调用配置方法初始化子虚拟设备的配置
920
读取设备树
930
生成子虚拟设备的 DDS
940
向子虚拟设备的设备驱动程序发送 DDS
950
子虚拟设备的设备驱动程序调用 VIO 总线设备驱动程序上的数据交换接口
960
通过使用数据交换接口，从子虚拟设备的设备驱动程序直接向 VIO 总线设备驱动程序传送数据
970
向 VIO 总线注册 / 注销子虚拟设备
980
初始化和不初始化 DMA 硬件和操作系统资源
结束

图 9