



(12) 发明专利

(10) 授权公告号 CN 101040268 B

(45) 授权公告日 2010. 10. 06

(21) 申请号 200580033560. 6

(22) 申请日 2005. 10. 05

(30) 优先权数据

10/959, 635 2004. 10. 05 US

(85) PCT申请进入国家阶段日

2007. 04. 02

(86) PCT申请的申请数据

PCT/JP2005/018737 2005. 10. 05

(87) PCT申请的公布数据

W02006/038717 EN 2006. 04. 13

(73) 专利权人 索尼计算机娱乐公司

地址 日本东京

(72) 发明人 铃置雅一 山崎刚

(74) 专利代理机构 中国国际贸易促进委员会专

利商标事务所 11038

代理人 李德山

(51) Int. Cl.

G06F 12/08 (2006. 01)

G06F 15/167 (2006. 01)

(56) 对比文件

US 4622631, 1986. 11. 11, 全文.

WO 03/003379 A1, 全文.

CN 1221919 A, 1999. 07. 07, 全文.

审查员 李佳

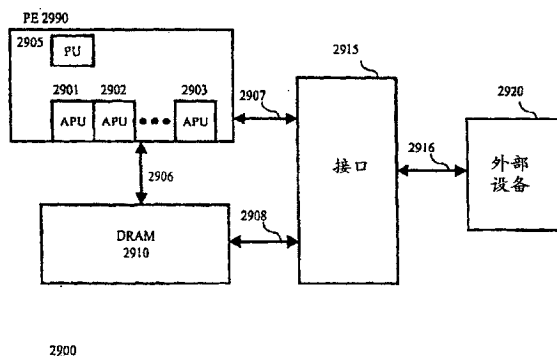
权利要求书 1 页 说明书 22 页 附图 48 页

(54) 发明名称

宽带网络计算机体系结构中的外部数据接口

(57) 摘要

系统配置包括处理元件 (PE), 输入 / 输出 (I/O) 接口设备和共享存储器。PE 进一步包括至少一个处理单元 (PU), 以及一个或更多附连处理单元 (APU)。APU 中的至少一个通过从连接到 I/O 接口设备的外部设备读取数据、并且将数据写入其中来执行 I/O 功能。利用数据级同步机构、经由共享存储器, 在 APU 和 I/O 接口设备之间交换数据。



1. 一种用于经由接口设备在存储器和外部设备之间传送数据的设备,包括:
接口设备,用于向外部设备传送数据和从外部设备接收数据;以及
存储器,具有多个存储器位置,每个存储器位置用于存储数据,

所述存储器能够存储与相应的存储器位置相关的状态信息,该状态信息包括第一字段和地址字段,使得对于指定的存储器位置,当相关状态信息的第一字段的值等于第一状态值,而且相关状态信息的地址字段的值等于预定地址值时,对该存储器位置的写操作在结束时使新存储在其中的数据被写到由所述预定地址值表示的地址。

2. 如权利要求 1 所述的设备,其中第一状态值表示相应存储器位置有效,而第二状态值表示相应存储器位置无效。

3. 如权利要求 2 所述的设备,还包括:
与存储器连接以进行数据访问的处理器,

其中当处理器发出有关从指定存储器位置到另一存储器的读操作的请求并且相关状态信息的第一字段的值等于第二状态值时,与第一字段的值变成第一状态值相同步地,新存储在该存储器位置的数据被写到所述另一存储器中由所述预定地址值表示的地址。

4. 如权利要求 2 所述的设备,其中当指定存储器位置中存储的数据被请求传送到外部设备并且相关状态信息的第一字段的值等于第二状态值时,与第一字段的值变成第一状态值相同步地,新存储在该存储器位置的数据被传送到外部设备中由所述预定地址值表示的地址。

5. 如权利要求 1、2、3 或 4 所述的设备,其中所述接口设备利用直接存储器访问在外部设备和存储器之间传送数据。

6. 一种用于经由接口设备在具有均能够存储数据的多个存储器位置的存储器和外部设备之间传送数据的方法,所述存储器能够存储与相应的存储器位置相关的状态信息,该状态信息包括第一字段和地址字段,该方法包括:

对存储器位置执行第一写操作;以及

当第一写操作使得相关状态信息的第一字段的值等于第一状态值,而且相关状态信息的地址字段的值等于预定地址值时,执行第二写操作,所述第二写操作用于在第一写操作结束时将新存储在该存储器位置的数据写到由所述预定地址值表示的地址。

宽带网络计算机体系结构中的外部数据接口

[0001] 对相关专利申请的交叉引用

[0002] 本申请是以下共同待审、共同受让的美国专利申请的部分延续：2001 年 3 月 22 提交的、申请号码为 09/816,004 的“宽带网络的计算机体系结构和软件单元”；2001 年 3 月 22 提交的、申请号码为 09/815,554 的“用于宽带网络计算机体系结构的数据同步的系统和方法”；2001 年 3 月 22 提交的、申请号码为 09/816,020 的“用于宽带网络计算机体系结构的存储器保护系统和方法”；2001 年 3 月 22 提交的、申请号码为 09/815,558 的“用于宽带网络计算机体系结构的资源专用系统和方法”；以及 2001 年 3 月 22 提交的、申请号码为 09/816,752 的“宽带网络计算机体系结构的处理模块”；所有这些在此被引用为参考。

技术领域

[0003] 本发明涉及计算机处理器和计算机网络的体系结构，并且特别涉及宽带环境中计算机处理器和计算机网络的体系结构。

背景技术

[0004] 对于当前计算机网络，例如办公室网络中所使用的局域网 (LAN)，以及诸如 Internet 的全球网络，其计算机和计算设备主要是为单机计算设计的。通过计算机网络共享数据和应用程序 (“应用”) 不是这些计算机和计算设备的主要设计目的。通常还使用各种类型的、由各不同生产商，例如摩托罗拉、Intel、德州仪器、索尼以及其它公司生产的处理器来设计这些计算机和计算设备。这些处理器中的每个都具有其独特的指令集和指令集体系结构 (ISA)，即，具有其独特的汇编语言指令集，和用于执行这些指令的基本计算单元和存储器单元的结构。因此，程序员需要理解每个处理器的指令集和 ISA，以便写出针对这些处理器的应用程序。当前计算机网络上的计算机和计算设备的这种异构组合使得数据和应用程序的处理和共享复杂化。而且，经常需要同一应用程序的多个版本，以便适应这种异构环境。

[0005] 连接到全球网络，尤其是连接到 Internet 的计算机和计算设备的种类是非常广泛的。除个人计算机 (PC) 和服务器之外，这些计算设备还包括蜂窝电话、移动计算机、个人数字助理 (PDA)、机顶盒、数字电视和许多其它设备。在如此多样的计算机和计算设备之间共享数据和应用程序带来了相当大的问题。

[0006] 为克服这些问题，已经使用了很多技术。尤其是，这些技术包括高级的接口和复杂的编程技术。这些解决方案的实现通常需要处理能力有相当大的增加。它们还经常导致在网络上处理应用程序和发送数据所需的时间大幅度增加。

[0007] 通常，通过 Internet 从相应的应用程序分别发送数据。该方法避免了利用对应于应用程序的每组发送数据发送应用程序的必要性。虽然该解决方案将所需带宽最小化，但是它还是常常引起用户之间的干扰。在客户端计算机上可能得不到用于发送数据的正确应用程序或最新应用程序。该方案还需要针对网络上处理器所使用的不同 ISA 和指令集的多样性编写每个应用程序的多个版本。

[0008] Java 模型试图解决这个问题。该模型使用服从严格安全协议的小应用程序 (“applet”)。通过网络从服务器计算机发送 Applet 以便由客户端计算机 (“client”) 运行。为避免必须向使用不同 ISA 的客户端发送相同 applet 的不同版本,所有 Java applets 均在客户端的 Java 虚拟机上运行。Java 虚拟机是模拟具有 Java ISA 和 Java 指令集的计算机的软件。然而,该软件在客户端的 ISA 和客户端的指令集上运行。Java 虚拟机的版本被提供给客户端的每个不同 ISA 和指令集。因此,就不需要每个 applet 的多个不同版本。每个客户端仅下载针对其特定的 ISA 和指令集的正确 Java 虚拟机来运行所有 Java applets。

[0009] 虽然给必须针对每个不同 ISA 和指令集编写应用程序的不同版本的问题提供了解决方案,但是 Java 处理模型在客户端计算机上仍需要附加的软件层。该附加的软件层明显降低了处理器的处理速度。这种速度降低对于实时、多媒体应用来说尤其明显。下载的 Java applet 还可能包含病毒、处理故障等等。这些病毒和故障会破坏客户端的数据库,并且造成其它损坏。虽然 Java 模型中所采用的安全协议试图通过实现软件“沙箱”来解决这个问题 (“沙箱”是客户端的存储器中的空间,超出该空间,Java applet 不能写数据),但是该软件驱动的安全模型在其实现中常常是不安全的,并且需要更多的处理。

[0010] 实时、多媒体、网络应用正变得越来越重要。这些网络应用需要极快的处理速度。将来,这种应用程序可能需要每秒若干千兆位的数据。目前的网络体系结构,尤其是 Internet 的体系结构,以及当前在例如 Java 模型里嵌入的编程模型使得达到这种处理速度极其困难。

[0011] 因此,需要一种新的计算机体系结构、一种新的计算机网络体系结构和一种新的编程模型。该新的体系结构和编程模型应当在不产生增加计算负担的情况下解决各种网络成员之间共享数据和应用程序的问题。该新的计算机体系结构和编程模型还应当解决在网络成员之间共享应用程序和数据时所固有的安全问题。

发明内容

[0012] 提供了一种用于通过宽带网络进行高速处理的计算机体系结构。具体地说,根据本发明,系统包括用于传送数据的接口设备和用于存储数据的存储器,其中存储器包括至少一个存储器位置以及与其相关的状态信息,其中状态信息包括第一字段和地址字段,并且其中当第一字段的值等于第一值,并且地址字段的值等于第二值时,数据到相关存储器位置的后续写入使得存储在其中的数据被写到由地址字段值表示的地址。

[0013] 在本发明的一个实施例中,系统配置包括处理元件 (PE),输入 / 输出 (I/O) 接口设备和共享存储器。PE 进一步包括至少一个处理单元 (PU),以及一个或多个附连处理单元 (APU)。APU 中的至少一个通过从连接到 I/O 接口设备的外部设备读取数据以及将数据写入此外部设备来执行 I/O 功能。利用数据级同步机构,在 APU 和 I/O 接口设备之间经由共享存储器交换数据。具体地说,共享存储器包括与至少一个用于存储数据的存储器位置相关的至少一个状态信息位置,其中状态信息位置包括第一字段和地址字段,其中当第一字段的值等于第一值,并且地址字段的值等于第二值时,数据到相关存储器位置的后续写入使得存储在其中的数据被写到由地址字段值表示的地址。

附图说明

- [0014] 图 1 说明了本发明的计算机网络的整体体系结构
- [0015] 图 2 说明了本发明的处理器单元 (PE) 的结构。
- [0016] 图 3 说明了本发明的宽带引擎 (BE) 的结构。
- [0017] 图 4 说明了本发明的附连处理单元 (APU) 的结构。
- [0018] 图 5 说明了本发明的处理器元件、观察器 (VS, visualizer) 和光学接口的结构。
- [0019] 图 6 说明了本发明的处理器元件的一种组合。
- [0020] 图 7 说明了本发明的处理器元件的另一种组合。
- [0021] 图 8 说明了本发明的处理器元件的另一种组合。
- [0022] 图 9 说明了本发明的处理器元件的另一种组合。
- [0023] 图 10 说明了本发明的处理器元件的另一种组合。
- [0024] 图 11A 说明了本发明的芯片封装内光学接口的集成。
- [0025] 图 11B 是使用图 11A 的光学接口的处理器的一种结构的图。
- [0026] 图 11C 是使用图 11A 的光学接口的处理器的另一种结构的图。
- [0027] 图 12A 说明了本发明的存储器系统的结构。
- [0028] 图 12B 根据本发明说明了从第一宽带引擎到第二宽带引擎的数据写入。
- [0029] 图 13 是本发明的处理器元件的共享存储器的结构图。
- [0030] 图 14A 说明了图 13 中所示的存储器的存储体的一种结构。
- [0031] 图 14B 说明了图 13 中所示的存储器的存储体的另一种结构。
- [0032] 图 15 说明了本发明的直接存储器存取控制器的结构。
- [0033] 图 16 说明了本发明的直接存储器存取控制器的可选结构。
- [0034] 图 17A-17D 说明了本发明的数据同步操作。
- [0035] 图 18 是根据本发明的数据同步模式说明存储器位置的不同状态的三状态存储器图。
- [0036] 图 19 说明了本发明的硬件沙箱的关键字控制表的结构。
- [0037] 图 20 说明了用于存储本发明的硬件沙箱的存储器存取关键字的方案。
- [0038] 图 21 说明了用于本发明的硬件沙箱的存储器存取控制表的结构。
- [0039] 图 22 是使用图 19 的关键字控制表和图 21 的存储器存取控制表访问存储器沙箱的步骤的流程图。
- [0040] 图 23 说明了本发明的软件单元的结构。
- [0041] 图 24 是根据本发明向 APU 发出远程过程调用的步骤的流程图。
- [0042] 图 25 说明了本发明用于处理流数据的专用流水线的结构。
- [0043] 图 26A-26B 是根据本发明由图 25 所示的专用流水线在处理流数据时执行的步骤的流程图。
- [0044] 图 27 说明了根据本发明用于处理流数据的专用流水线的可选结构。
- [0045] 图 28 说明了根据本发明用于协调由 APU 对应用程序和数据的并行处理的绝对定时器的方案。
- [0046] 图 29-30 示出了根据本发明的原理使用数据级同步的结构的图解实施例。
- [0047] 图 31-32 示出了根据本发明的原理用于执行数据级同步的示例性流程图。
- [0048] 图 33-34 说明了在 APU 和外部设备之间利用数据级同步。

- [0049] 图 35 示出了根据本发明的原理示例性数据级同步状态信息。
- [0050] 图 36 示出了根据本发明的原理使用数据级同步的结构的一个图解实施例。
- [0051] 图 37 示出了示例性握手序列。
- [0052] 图 38 示出了根据本发明的原理使用数据级同步的结构的一个图解实施例。
- [0053] 图 39 示出了根据本发明的原理使用数据级同步的结构的一个图解实施例。
- [0054] 图 40 示出了根据本发明的原理用于执行数据级同步的示例性流程图。
- [0055] 图 41 提供了在 APU 和外部设备之间利用数据级同步的另一个说明。
- [0056] 图 42 示出了根据本发明的原理使用数据级同步的结构的一个图解实施例。

具体实施方式

- [0057] 图 1 中示出了根据本发明的计算机系统 101 的整体体系结构。
- [0058] 如该图所示,系统 101 包括多个计算机和计算设备连接到的网络 104。网络 104 可以是 LAN、诸如 Internet 的全球网络或任何其他计算机网络。
- [0059] 连接到网络 104 的计算机和计算设备(网络的“成员”)包括例如客户端计算机 106、服务器计算机 108、个人数字助理(PDA)110、数字电视(DTV)112、以及其它有线或无线的计算机和计算设备。网络 104 的成员所采用的处理器由相同的公用计算模块构建。这些处理器最好还都具有相同的 ISA,并且根据相同的指令集执行处理。任何一个特定处理器内包含的模块数量取决于该处理器所需的处理能力。
- [0060] 例如,由于系统 101 的服务器 108 比客户端 106 执行更多的数据处理和应用,所以服务器 108 比客户端 106 包含更多的计算模块。另一方面,PDA 110 执行的处理量最少。因此,PDA 110 包含的计算模块数量最少。DTV 112 执行的处理量介于客户端 106 和服务器 108 的之间。因此,DTV 112 包含的计算模块数量介于客户端 106 和服务器 108 的之间。如下面所讨论,每个计算模块包含一个处理控制器和多个相同处理单元,用于执行通过网络 104 传送的数据和应用程序的并行处理。
- [0061] 系统 101 的这种同构结构有利于提高适应性、处理速度和处理的效率。因为系统 101 的每个成员利用一个或多个(或某些部分)相同计算模块来执行处理,所以执行数据和应用程序的实际处理的特定计算机或计算设备是不重要的。而且,特定应用程序和数据的处理可在网络的成员间分享。通过在整个系统唯一标识含有由系统 101 处理的数据和应用程序的单元,则无论处理在哪里进行,都可以将该处理结果传送到请求处理的计算机或计算设备。因为执行该处理的模块具有公用结构并使用公用 ISA,所以避免了为实现处理器之间的兼容性而附加的软件层的计算负担。该体系结构和编程模型有利于得到执行例如实时、多媒体应用程序所必需的处理速度。
- [0062] 为进一步利用由系统 101 带来的处理速度和效率,将由该系统处理的数据和应用程序封装到唯一标识的、统一格式化的软件单元 102 中。每个软件单元 102 包含或可以包含应用程序和数据。每个软件单元还包含在整个网络 104 和系统 101 中全局性地标识该单元的 ID。软件单元结构的这种一致性以及软件单元在整个网络中的唯一标识,有利于在网络的任何一个计算机或计算设备上处理应用程序和数据。例如,客户端 106 可以形成软件单元 102,但是因为客户端 106 的处理能力有限,可以将该软件单元发送到服务器 108 来处理。因此,软件单元可以在整个网络 104 中迁移,以便根据网络上的处理资源的可用性进行

处理。

[0063] 系统 101 的处理器和软件单元的同构结构还避免了现今的异构网络的许多问题。例如,设法允许在使用任何指令集的任何 ISA,例如 Java 虚拟机这样的虚拟机上进行应用程序处理的低效编程模型得以避免。因此,系统 101 可实现远比现今网络能有效和高效的宽带处理。

[0064] 用于网络 104 的所有成员的基本处理模块是处理器元件 (PE)。图 2 说明了 PE 的结构。如图所示,PE 201 包含处理单元 (PU) 203、直接存储器存取控制器 (DMAC) 205 和多个附连处理单元 (APU),即 APU207、APU 209、APU 211、APU 213、APU 215、APU 217、APU 219 和 APU 221。局部 PE 总线 223 在 APU、DMAC 205 和 PU 203 间传输数据和应用程序。局部 PE 总线 223 可具有例如传统体系结构,或者可以被实现成分组交换网络。虽然需要更多的硬件,然而实现成分组交换网络可增加可用带宽。

[0065] 可使用用于实现数字逻辑的各种方法来构建 PE 201。然而,PE201 最好被构造成使用硅衬底上的互补金属氧化物半导体 (CMOS) 的单个集成电路。可选的衬底材料包括砷化镓,镓铝砷化物,以及其它使用各种掺杂物的所谓 III-B 化合物。还可使用超导材料,例如快速单磁通量子 (RSFQ) 逻辑来实现 PE 201。

[0066] PE 201 通过高带宽存储器连接 227 与动态随机存取存储器 (DRAM) 225 紧密相关。DRAM 225 用作 PE 201 的主存储器。虽然 DRAM 225 最好是动态随机存取存储器,但也可使用其它装置将 DRAM 225 实现成例如静态随机存取存储器 (SRAM)、磁性随机存取存储器 (MRAM)、光存储器或全息存储器。DMAC 205 利于在 DRAM225 与 PE 201 的 APU 和 PU 之间传输数据。如下面进一步讨论的,DMAC 205 在 DRAM 225 中为每个 APU 指定一个独占区域,只有该 APU 可以往里写数据或从中读数据。这个独占区域被称为“沙箱”。

[0067] PE 203 可以是例如能够独立处理数据和应用程序的标准处理器。在操作中,PU 203 调度并编排 APU 对数据和应用程序的处理。APU 最好是单指令多数据 (SIMD) 处理器。在 PU 203 的控制下,APU 以并行和独立的方式执行这些数据和应用程序的处理。DMAC 205 控制 PU203 和 APU 对存储在共享 DRAM 225 中的数据和应用程序的存取。虽然 PE 201 最好包括八个 APU,但也可根据所需的处理能力在 PE 中使用更多或更少数目的 APU。PU203 和某些 APU,或全部 APU 可以具有相同的硬件结构和 / 或功能。必要时,通过软件将个别处理器配置成控制或被控制处理器。例如,图 3 中,PE 201 可包括 9 个具有相同体系结构的处理器。9 个处理器中的一个可被指定为控制处理器 (例如 PU203),而剩余的处理器可被指定为被控制处理器 (例如 APU207、209、211、213、215、217、219 和 221)。并且,可将若干像 PE 201 这样的 PE 连接或封装在一起以提供增强的处理能力。

[0068] 例如,如图 3 所示,可将四个 PE 一起封装或连接在例如一个或多个芯片封装内,以形成用于网络 104 的成员的单个处理器。这种结构被称为宽带引擎 (BE)。如图 3 所示,BE 301 包含四个 PE,即 PE 303、PE 305、PE 307 和 PE 309。通过 BE 总线 311 进行这些 PE 之间的通信。宽带存储器连接 313 提供共享 DRAM 315 与这些 PE 之间的通信。作为 BE 总线 311 的替代,可通过 DRAM 315 和这个存储器连接进行 BE 301 的 PE 之间的通信。

[0069] 输入 / 输出 (I/O) 接口 317 和外部总线 319 提供宽带引擎 301 和网络 104 的其它成员之间的通信。与 PE 的 APU 执行的并行与独立的应用程序和数据处理相类似,BE 301 的每个 PE 以并行和独立的方式执行数据和应用程序的处理。

[0070] 图4说明了APU的结构。APU 402包括局部存储器406,寄存器410,四个浮点单元412和四个整数单元414。P15然而,根据所需要的处理能力,可以使用更多或更少数目的浮点单元412和整数单元414。在优选实施例中,局部存储器406包含128千字节的存储量,并且寄存器410的容量是 128×128 位。浮点单元412最好以320亿(32billion)次浮点运算每秒(32GFLOPS)的速度工作,整数单元414最好以320亿次运算每秒(32GOPS)的速度工作。

[0071] 局部存储器402不是高速缓冲存储器。局部存储器402最好被构造成SRAM。对APU的高速缓存一致性支持是没有必要的。对于由PU启动的直接存储器存取,PU会需要高速缓存一致性支持。然而,对于由APU启动的直接存储器存取或针对外部设备的存取,则不需要高速缓存一致性支持。

[0072] APU 402还包含总线404,用于针对APU传送应用程序和数据。在优选实施例中,该总线的宽度是1024位。APU 402还包括内部总线408、420和418。在优选实施例中,总线408的宽度是256位,提供局部存储器406与寄存器410之间的通信。总线420和418分别提供寄存器410和浮点单元412之间,寄存器410和整数单元414之间的通信。在优选实施例中,从寄存器410到浮点单元或整数单元的总线418和420的宽度是384位,从浮点或整数单元到寄存器410的总线418和420的宽度是128位。这些总线中,从寄存器410到浮点或整数单元的宽度比从这些单元到寄存器410的宽度大,这样可适应处理期间来自寄存器410的较大数据流。每次计算最多需要三个字。然而,每次计算的结果通常只是一个字。

[0073] 图5-10进一步说明了网络104的成员的模块化的结构。例如,如图5所示,处理器可以包含单个PE 502。如上所述,该PE通常包含一个PU、一个DMAC和八个APU。每个APU包括局部存储器(LS)。另一方面,处理器可包含观察器(VS, visualizer)505的结构。如图5所示,VS 505包含PU 512、DMAC 514和四个APU,即APU 516、APU 518、APU 520和APU 522。在这种情况下,通常由PE的其它四个APU占有的芯片封装内空间则由像素引擎508,图象高速缓存510,以及阴极射线管控制器(CRTC)504占用。根据PE 502或VS 505所需要的通信速度,光学接口506也可被包括在芯片封装内。

[0074] 使用这种标准模块化的结构,可以很容易和有效地构建处理器的许多其他变种。例如,图6所示的处理器包含两个芯片封装,即包含一个BE的芯片封装602和包含四个VS的芯片封装604。输入/输出(I/O)606提供芯片封装602的BE与网络104之间的接口。总线608提供芯片封装602与芯片封装604之间的通信。输入输出处理器(IOP)610控制数据流入和流出I/O 606。I/O 606可以制造成专用集成电路(ASIC)。从VS输出的是视频信号612。

[0075] 图7说明了带有两个光学接口704和706的BE 702的芯片封装,所述光学接口用于提供与网络104的其它成员(或本地连接的其它芯片封装)的甚高速通信。BE 702可用作例如网络104上的服务器。

[0076] 图8的芯片封装包含两个PE 802和804,以及两个VS 806和808。I/O 810提供芯片封装与网络104间的接口。从芯片封装输出的是视频信号。这种结构可用作例如图形工作站。

[0077] 图9说明了另外一种结构。该结构包含图8中说明的结构的处理能力的一半。提

供一个 PE 902,而不是两个 PE,并且提供一个 VS 904,而不是两个 VS。I/O 906 的带宽是图 8 中说明的 I/O 带宽的一半。然而,这样的处理器也可用作图形工作站。

[0078] 最后一种结构如图 10 所示。该处理器仅由一个单一的 VS 1002 和 I/O 1004 组成。这种结构可用作例如 PDA。

[0079] 图 11A 说明了光学接口在网络 104 的处理器芯片封装中的集成。这些光学接口将光信号转换成电信号,并将电信号转换成光信号,它们可以用各种材料制成,包括例如砷化镓、镓铝砷化物、锗以及其它元素和化合物。如图中所示,光学接口 1104 和 1106 被做在 BE 1102 的芯片封装上。BE 总线 1108 提供 BE 1102 的多个 PE,即 PE 1110、PE 1112、PE 1114、PE 1116 与这些光学接口之间的通信。光学接口 1104 包括两个端口,即端口 1118 和端口 1120,并且光学接口 1106 也包括两个端口,即端口 1122 和端口 1124。端口 1118、1120、1122 和 1124 分别连接到光波导 1126、1128、1130 和 1132。通过光学接口 1104 和 1106 的端口,经由这些光波导针对 BE 1102 传送光信号。

[0080] 使用这种光波导和每个 BE 的四个光端口可将多个 BE 按各种结构连接在一起。例如,如图 11B 所示,通过这种光端口可将两个或多个 BE,例如 BE 1152、BE 1154 和 BE 1156 串行连接在一起。在该例中,BE 1152 的光学接口 1166 通过其光端口连接到 BE 1154 的光学接口 1160 的光端口。以类似的方式,BE 1154 上光学接口 1162 的光端口连接到 BE 1156 上光学接口 1164 的光端口。

[0081] 图 11C 说明了一种矩阵结构。在该结构中,每个 BE 的光学接口连接到两个其它的 BE 上。如图中所示,BE 1172 的光学接口 1188 的一个光端口连接到 BE 1176 的光学接口 1182 的光端口。光学接口 1188 的另一个光端口连接到 BE 1178 的光学接口 1184 的光端口。以类似的方式,BE 1174 的光学接口 1190 的一个光端口连接到 BE 1178 的光学接口 1184 的另一个光端口。光学接口 1190 的另一个光端口连接到 BE 1180 的光学接口 1186 的光端口。这种矩阵结构可按类似的方式延及其它 BE。

[0082] 通过使用串联结构或矩阵结构,可构建网络 104 的具有任何期望大小和能力的处理器。当然,可以向 BE 的光学接口,或其 PE 数目多于或少于 BE 的 PE 数目的处理器增加附加端口,以形成其它的结构。

[0083] 图 12A 说明了用于 BE 的 DRAM 的控制系统和结构。类似的控制系统和结构被用于具有其它大小和包含或多或少的 PE 的处理器中。如图中所示,交叉开关将构成 BE 1201 的四个 PE 的各个 DMAC 1210 连接到八个存储体控制器 1206。每个存储体控制器 1206 控制 DRAM 1204 的八个存储体 1208(图中只画出了四个)。因此,DRAM 1204 总共包含 64 个存储体。在优选实施例中,DRAM 1204 具有 64 兆字节的容量,并且每个存储体具有 1 兆字节的容量。在该优选实施例中,每个存储体中的最小可寻址单位是 1024 位块。

[0084] BE 1201 还包括开关单元 1212。开关单元 1212 能够使 BE 上的其它 APU 紧密耦合到 BE 1201,以便访问 DRAM 1204。因此,第二 BE 可以紧密耦合到第一 BE,并且每个 BE 的每个 APU 可以寻址的存储器位置的数量是通常可由 APU 访问的存储器位置的数量两倍。通过例如开关单元 1212 的开关单元,可以直接从第一 BE 的 DRAM 到第二 BE 的 DRAM,或从第二 BE 的 DRAM 到第一 BE 的 DRAM 的读取或写入数据。

[0085] 例如,如图 12B 所示,为完成这种写操作,第一 BE 的 APU,例如, BE 1222 的 APU 1220 向第二 BE 的 DRAM,例如 BE 1226 的 DRAM 1228(而不是象通常的情况那样向 BE 1222

的 DRAM 1224) 的存储器位置发出写命令。BE 1222 的 DMAC 1230 通过交叉开关 1221 向存储体控制器 1234 发送写命令, 并且存储体控制器 1234 向连接到存储体控制器 1234 的外部端口 1232 发送该命令。BE 1226 的 DMAC 1238 接收该写命令, 并将该命令传送到 BE 1226 的开关单元 1240。开关单元 1240 识别包含在写命令中的 DRAM 地址, 并通过 BE 1226 的存储体控制器 1242 向 DRAM 1228 的存储体 1244 发送存储在该地址中的数据。因此, 开关单元 1240 使 DRAM 1224 和 DRAM 1228 能够充当 BE1222 的 APU 的单独存储空间。

[0086] 图 13 说明了 DRAM 的 64 存储体的结构。这些存储体排列成八行, 即行 1302、1304、1306、1308、1310、1312、1314 和 1316, 和八列, 即列 1320、1322、1324、1326、1328、1330、1332 和 1334。每行由一个存储体控制器控制。因此, 每个存储体控制器控制八兆字节的存储器。

[0087] 图 14A 和 14B 说明了用于存储和访问 DRAM 的最小可寻址存储单位, 例如 1024 位块的不同结构。在图 14A 中, DMAC 1402 在单个存储体 1404 中存储八个 1024 位块 1406。另一方面, 在图 14B 中, 当 DMAC 1412 读和写包含 1024 位的数据块时, 这些块在两个存储体, 即存储体 1414 和存储体 1416 之间交错。因此, 这些存储体中的每一个包含 16 个数据块, 并且每个数据块包含 512 位。这种交错可有利于更快速地访问 DRAM, 并且在某些应用程序的处理中是非常有用的。

[0088] 图 15 说明了 PE 中的 DMAC 1504 的体系结构。如图中所示, 构成 DMAC 1506 的结构硬件在整个 PE 中分布, 使得每个 APU 1502 直接访问 DMAC 1506 的结构节点 1504。每个节点执行适合于由该节点直接访问的 APU 进行存储器访问的逻辑。

[0089] 图 16 示出了 DMAC, 即非分布式体系结构的一个可选实施例。在这种情况下, DMAC 1606 的结构硬件是集中式的。APU 1602 和 PU1604 通过局部 PE 总线 1607 与 DMAC 1606 通信。DMAC 1606 通过交叉开关连接到总线 1608。总线 1608 连接到 DRAM 1610。

[0090] 如上所述, PE 的所有多个 APU 可独立访问共享 DRAM 中的数据。结果, 在第二 APU 请求第一 APU 的局部存储器中的特定数据时, 第一 APU 可能正对该数据进行操作。如果此时将数据从共享 DRAM 提供到第二 APU, 则数据可能是无效的, 因为第一 APU 正在进行的处理可能改变数据的值。因此, 如果此时第二处理器接收到了来自共享 DRAM 的数据, 则第二处理器可能会产生错误的结果。例如, 数据可能是全局变量的特定值。如果第一处理器在其处理过程中改变了该值, 则第二处理器会接收到一个过期值。因此, 必须有一种方案来同步 APU 对共享 DRAM 中的存储器位置进行的数据读出和写入。该方案必须防止从这样的存储器位置读取数据, 其中另一 APU 当前正在其局部存储器中对该存储器位置进行操作, 因此此时读取的数据不是最新的, 并且防止向存储当前数据的存储器位置写数据。

[0091] 为了解决这些问题, 对于 DRAM 的每个可寻址存储器位置, 在 DRAM 中分配一个附加存储段以存储有关该存储器位置中存储的数据的状态信息。这种状态信息包括满 / 空 (F/E) 位, 向存储器位置请求数据的 APU 的标识 (APU ID), 以及从其读取所请求数据的 APU 局部存储器的地址 (LS 地址)。DRAM 的可寻址存储器位置可以具有任意大小。在优选实施例中, 该大小是 1024 位。

[0092] 将 F/E 位设成 1 表示存储在相关存储器位置中的数据是最新的。相反, 将 F/E 位设成 0 表示存储在相关存储器位置中的数据不是最新的。如果 APU 在该位被设成 0 时请求数据, 则禁止 APU 立即读取数据。在这种情况下, 标识请求数据的 APU 的 APU ID, 和 LS 地址被记入到附加存储段中, 其中 LS 地址标识在数据变为最新数据时, 数据将被读入的这个

APU 的局部存储器中的存储器位置。

[0093] 对于 APU 的局部存储器中的每个存储器位置,也分配附加存储段。该附加存储段存储一个位,称为“忙位 (busy bit)”。该忙位用于保留相关的 LS 存储器位置,用于存储从 DRAM 取回的特定数据。如果针对局部存储器中特定存储器位置的忙位被设成 1,则 APU 只能将该存储器位置用于写入这些特定数据。反之,如果针对局部存储器中特定存储器位置的忙位被设成 0,则 APU 可将该存储器位置用于写入任何数据。

[0094] 图 17A-17O 说明了使用 F/E 位, APU ID, LS 地址以及忙位来同步针对 PE 的共享 DRAM 的数据读写的方式的几个例子。

[0095] 如图 17A 所示,一或多个例如 PE 1720 的 PE 与 DRAM 1702 交互。PE 1720 包括 APU 1722 和 APU 1740。APU 1722 包括控制逻辑 1724,而 APU 1740 包括控制逻辑 1742。APU 1722 还包括局部存储器 1726。该局部存储器包括多个可寻址存储器位置 1728。APU 1740 包括局部存储器 1744,该局部存储器还包括多个可寻址存储器位置 1746。所有这些可寻址存储器位置的大小最好是 1024 位。

[0096] 附加存储段与每个 LS 可寻址存储器位置相关。例如,存储段 1729 和 1734 分别与局部存储器位置 1731 和 1732 相关,并且,存储段 1752 与局部存储器位置 1750 相关。如上所述,“忙位”存储在这些附加存储段的每一个中。局部存储器位置 1732 如图所示具有几个 X,表示该存储器位置包含数据。

[0097] DRAM 1702 包含多个可寻址存储器位置 1704,包括存储器位置 1706 和 1708。这些存储器位置的大小最好也是 1024 位。附加存储段也与这些存储器位置中的每一个相关。例如,附加存储段 1760 与存储器位置 1706 相关,并且附加存储段 1762 与存储器位置 1708 相关。与存储在每个存储器位置的数据相关的状态信息被存储在与该存储器位置相关的存储段中。如上所述,该状态信息包括 F/E 位, APU ID 和 LS 地址。例如,对于存储器位置 1708,该状态信息包括 F/E 位 1712, APU ID 1714 和 LS 地址 1716。

[0098] 通过使用状态信息和忙位,能够实现在 PE 或一组 PE 的 APU 中间对共享 DRAM 的数据读写同步。

[0099] 图 17B 说明了从 APU 1722 的 LS 存储器位置 1732 向 DRAM 1702 的存储器位置 1708 的数据同步写入的启动。APU 1722 的控制器 1724 启动这些数据的同步写入。因为存储器位置 1708 是空的,所以 F/E 位 1712 设为 0。结果,可以将 LS 存储器位置 1732 中的数据写入存储器位置 1708 中。另一方面,如果该位设为 1 以指示存储器位置 1708 是满的,并且包含的数据是最新和有效的,则控制器 1722 将收到错误消息,并且被禁止向该存储器位置写入数据。

[0100] 图 17C 示出了成功地向存储器位置 1708 同步写入数据的结果。写入的数据被存储在存储器位置 1708 中,并将 F/E 位 1712 设成 1。该设定指示存储器位置 1708 是满的,并且该存储器位置中的数据是最新和有效的。

[0101] 图 17D 说明了从 DRAM 1702 的存储器位置 1708 同步读取数据到局部存储器 1744 的 LS 存储器位置 1750 的启动。为启动该读取操作,将 LS 存储器位置 1750 的存储段 1752 中的忙位设成 1,以为这些数据保留该存储器位置。将该忙位设成 1 可禁止 APU 1740 在该存储器位置存储其它数据。

[0102] 接着如图 17E 所示,控制逻辑 1742 针对 DRAM 1702 的存储器位置 1708 发出同步

读命令。因为与该存储器位置相关的 F/E 位 1712 被设成 1, 所以存储在存储器位置 1708 中的数据被认为是最新和有效的。因此, 将 F/E 位 1712 设成 0, 以准备将数据从存储器位置 1708 传送到 LS 存储器位置 1750。该设定如图 17F 所示。将该位设成 0 表示: 在读取这些数据之后, 存储器位置 1708 中的数据无效。

[0103] 接下来, 如图 17G 所示, 将存储器位置 1708 中的数据从存储器位置 1708 读到 LS 存储器位置 1750 中。图 17H 示出了最终的状态。将存储器位置 1708 中的数据的副本存储到 LS 存储器位置 1750 中。F/E 位被设成 0 以指示存储器位置 1708 中的数据无效。这种无效是 APU 1740 对这些数据进行的修改的结果。存储段 1752 中的忙位也被设成 0。该设定指示 LS 存储器位置 1750 现在可被 APU 1740 用于任何目的, 即该 LS 存储器位置不再处于等待接收特定数据的保留状态。因此, APU 1740 可对 LS 存储器位置 1750 进行任何目的的访问。

[0104] 图 17I-17O 说明了当针对 DRAM 1702 的存储器位置的 F/E 位设成 0 以指示该存储器位置中的数据不是最新或有效时, 从 DRAM 1702 的存储器位置 (例如存储器位置 1708) 同步读数据到 APU 的局部存储器的 LS 存储器位置 (例如局部存储器 1744 的 LS 存储器位置 1752) 的操作。如图 17I 所示, 为启动这种传送, 将 LS 存储器位置 1750 的存储段 1752 中的忙位设成 1, 以保留该 LS 存储器位置用于该数据传输。接下来, 如图 17J 所示, 控制逻辑 1742 针对 DRAM 1702 的存储器位置 1708 发出同步读命令。由于与该存储器位置相关的 F/E 位, 即 F/E 位 1712 被设成 0, 所以存储器位置 1708 中存储的数据是无效的。结果, 向控制逻辑 1742 发送信号, 以阻止从该存储器位置立即读取数据。

[0105] 接着, 如图 17K 所示, 将用于该读命令的 APU ID 1714 和 LS 地址 1716 写入到存储段 1762 中。在这种情况下, 将针对 APU 1740 的 APU ID 和针对 LS 存储器位置 1750 的 LS 存储器位置写入到存储段 1762 中。因此, 当存储器位置 1708 中的数据变成最新时, 使用该 APUID 和 LS 存储器位置来确定当前数据要发送到的位置。

[0106] 当 APU 将数据写入该存储器位置时, 存储器位置 1708 中的数据变为有效和最新的。图 17L 说明了从例如 APU 1722 的存储器位置 1732 到存储器位置 1708 的数据同步写操作。因为针对该存储器位置的 F/E 位被设为 0, 所以这些数据的这种同步写是许可的。

[0107] 如图 17M 所示, 在写操作之后, 存储器位置 1708 中的数据变成最新和有效的。因此, 立即从存储段 1762 读取来自存储段 1762 的 APUID 1714 和 LS 地址, 然后将该信息从该存储段中删除。还将 F/E 位 1712 设成 0, 以期立即读取存储器位置 1708 中的数据。如图 17N 所示, 当读取 APU ID 1714 和 LS 地址 1716 后, 立即将该信息用于将存储器位置 1708 中的有效数据读到 APU 1740 的 LS 存储器位置 1750 中。最终状态示于图 17O。该图示出从存储器位置 1708 复制到存储器位置 1750 的有效数据, 存储段 1752 中的忙位设为 0, 并且存储段 1762 中的 F/E 位 1712 设为 0。将此忙位设成 0 允许 LS 存储器位置 1750 现在可被 APU 1740 进行任何目的的访问。将 F/E 位设成 0 指示存储器位置 1708 中的数据不再是最新和有效的。

[0108] 图 18 根据对应于存储器位置的存储段中存储的 F/E 位的状态, APU ID 和 LS 地址, 概括了上述操作和 DRAM 的存储器位置的各种状态。存储器位置可有三种状态。这三种状态是: 空状态 1880, 其中 F/E 位设成 0, 并且不针对 APU ID 或 LS 地址提供信息; 满状态 1882, 其中 F/E 位设成 1, 并且不针对 APU ID 或 LS 地址提供信息; 阻塞状态 1884, 其中 F/E 位设

成 0,并且针对 APU ID 和 LS 地址提供信息。

[0109] 如该图所示,在空状态 1880 中,允许同步写操作,并且导致切换到满状态 1882。而同步读操作导致切换到阻塞状态 1884,因为当存储器位置处于空状态时,存储器位置中的数据不是最新的。

[0110] 在满状态 1882 中,允许同步读操作,并导致切换到空状态 1880。另一方面,满状态 1882 下的同步写操作被禁止,以防止覆盖有效数据。如果在这种状态中试图进行这种写操作,则不发生状态改变,并且向 APU 的相应控制逻辑发送错误消息。

[0111] 在阻塞状态 1884 中,允许对存储器位置进行同步数据写操作,并导致切换到空状态 1880。另一方面,阻塞状态 1884 下的同步读操作被禁止,以防止与较早的、导致这种状态的同步读操作发生冲突。如果在阻塞状态 1884 下试图进行同步读操作,则不发生状态改变,并且向 APU 的对应控制逻辑发送错误消息。

[0112] 上述用于针对共享 DRAM 的同步读和写数据操作的方案还可用于节省通常被处理器专用于针对外部设备读写数据的计算资源。该输入 / 输出 (I/O) 功能可由 PU 执行。然而,通过使用该同步方案的修改方案,运行适当程序的 APU 可执行该功能。例如,通过使用该方案,接收由外部设备发出的、用于从 I/O 接口传输数据的中断请求的 PU 可以委托该 APU 来处理这个请求。然后,APU 向 I/O 接口发出同步写命令。该接口则通知外部设备:现在可以向 DRAM 写入数据。接下来,APU 向 DRAM 发出同步读命令,以将 DRAM 的有关存储空间设成阻塞状态。APU 还将 APU 局部存储器中接收数据所需的存储器位置的忙位设成 1。在阻塞状态中,与 DRAM 的有关存储空间相关的附加存储段包含 APU 的 ID 和 APU 局部存储器的有关存储器位置的地址。接着,外部设备发出同步写命令,以直接向 DRAM 的有关存储空间写入数据。由于该存储空间处于阻塞状态,所以立即将数据从该空间读到附加存储段中所标识的 APU 局部存储器的存储器位置上。然后将针对这些存储器位置的忙位设成 0。当外部设备完成数据的写操作时,APU 向 PU 发出表明传输完成的信号。

[0113] 因此,通过使用这个方案,可处理来自外部设备的数据传输,并使 PU 的计算负荷最小。然而,被委派此功能的 APU 应当能够向 PU 发出中断请求,并且外部设备应当直接访问 DRAM。

[0114] 每个 PE 的 DRAM 包括多个“沙箱”。沙箱定义了共享 DRAM 的一个区域,如果超出此区域,特定 APU 或特定一组 APU 不能读或写数据。这些沙箱提供了防止正被一个 APU 处理的数据遭到正被另一个 APU 处理的数据破坏的安全性。这些沙箱还允许从网络 104 向特定沙箱下载软件单元,而不存在软件单元破坏整个 DRAM 中的任何数据的可能性。在本发明中,用 DRAM 和 DMAC 的硬件来实现沙箱。通过用硬件而不是软件来实现沙箱,可获得速度和安全性方面的优势。

[0115] PE 的 PU 控制分配给 APU 的沙箱。由于 PU 通常只操作受信的程序,例如操作系统,该方案不会危及安全性。根据该方案,PU 建立并维护关键字控制表。图 19 示出了该关键字控制表。如该图所示,关键字控制表 1902 的每个条目包含针对 APU 的标识 (ID) 1904,针对该 APU 的 APU 关键字 1906,和关键字掩码 1908。该关键字掩码的使用将在后面进行说明。关键字控制表 1902 最好存储在相对较快的存储器中,例如静态随机存取存储器 (SRAM),并且与 DMAC 相关。关键字控制表 1902 中的条目由 PU 控制。当 APU 请求针对 DRAM 的特定存储器位置读写数据时,DMAC 对照与该存储器位置相关的存储器存取关键字,来评估关键字

控制表 1902 中分配给该 APU 的 APU 关键字 1906。

[0116] 如图 20 所示,给 DRAM 2002 的每个可寻址存储器位置 2006 分配一个专用存储段 2010。在该专用存储段中存储针对存储器位置的存储器存取关键字 2012。如上所述,另一个附加专用存储段 2008 也与每个可寻址存储器位置 2006 相关,并且存储用于针对存储器位置读写数据的同步信息。

[0117] 在操作中,APU 向 DMAC 发出 DMA 命令。该命令包括 DRAM2002 的存储器位置 2006 的地址。在执行该命令之前,DMAC 使用 APU 的 ID 1904 在关键字控制表 1902 中查找发出请求的 APU 的关键字 1906。然后,DMAC 将发出请求的 APU 的 APU 关键字 1906 与专用存储段 2010 中存储的、和 APU 试图访问的 DRAM 存储器位置相关的存储器存取关键字 2012 进行比较。如果这两个关键字不匹配,则不执行 DMA 命令。另一方面,如果这两关键字匹配,则 DMA 命令继续进行,并且执行所请求的存储器访问。

[0118] 图 21 说明了另一可选实施例。在该实施例中,PU 同样也维护存储器存取控制表 2102。存储器存取控制表 2102 包含针对 DRAM 中每个沙箱的条目。在图 21 的特定例子中,DRAM 包含 64 个沙箱。存储器存取控制表 2102 中的每个条目包含沙箱的标识 (ID) 2104,存储器基地址 2106,沙箱大小 2108,存储器存取关键字 2110 和存取关键字掩码 2110。存储器基地址 2106 提供 DRAM 中的、开始特定存储器沙箱的地址。沙箱大小 2108 提供沙箱的大小,并且因此提供特定沙箱的终点。

[0119] 图 22 是使用关键字控制表 1902 和存储器存取控制表 2102 执行 DMA 命令的步骤的流程图。在步骤 2202 中,APU 向 DMAC 发出用于访问沙箱中的一个特定存储器位置或多个位置的 DMA 命令。该命令包括沙箱 ID 2104,用于标识被请求访问的特定沙箱。在步骤 2204 中,DMAC 使用 APU 的 ID 1904 在关键字控制表 1902 中查找发出请求的 APU 的关键字 1906。在步骤 2206 中,DMAC 使用命令中的沙箱 ID 2104 在存储器存取控制表 2102 中查找与该沙箱相关的存储器存取关键字 2110。在步骤 2208 中,DMAC 将分配给发出请求的 APU 的 APU 关键字 1906 与和沙箱相关的存取关键字 2110 进行比较。在步骤 2210 中,确定这两个关键字是否匹配。如果这两个关键字不匹配,则处理转到步骤 2212,其中 DMA 命令不继续执行,并向发出请求的 APU,PU 或二者发送错误消息。另一方面,如果在步骤 2210 中发现两个关键字匹配,则处理进行到步骤 2214,其中 DMAC 执行 DMA 命令。

[0120] 针对 APU 关键字和存储器存取关键字的关键字掩码为该系统提供了更大的灵活性。针对关键字的关键字掩码将被屏蔽位转换成通配符。例如,如果与 APU 关键字 1906 相关的关键字掩码 1908 的最后两位被设成“掩码”(例如通过将关键字掩码 1908 中的这些位设为 1 来指定),那么 APU 关键字可以是 1 或 0,并且仍然与存储器存取关键字匹配。例如,APU 关键字可以是 1010。该 APU 关键字通常只允许访问具有存取关键字 1010 的沙箱。然而,如果将用于该 APU 关键字的 APU 关键字掩码设成 0001,则可使用该 APU 关键字来实现对存取关键字为 1010 或 1011 的沙箱的访问。类似地,掩码设为 0001 的存取关键字 1010 可被 APU 关键字为 1010 或 1011 的 APU 访问。因为可同时使用 APU 关键字掩码和存储器关键字掩码,所以可确定 APU 对沙箱的各种可访问性。

[0121] 本发明还为系统 101 的处理器提供了新的编程模型。该编程模型使用软件单元 102。可以将这些单元发送到网络 104 上的任何处理器来进行处理。这种新的编程模型还利用系统 101 的独特模块化体系结构和系统 101 的处理器。

[0122] APU 直接从 APU 局部存储器处理软件单元。APU 不直接对 DRAM 中的任何数据或程序进行操作。在 APU 处理这些数据 and 程序之前,将 DRAM 中的数据和程序读取到 APU 的局部存储器中。因此,APU 的局部存储器包括程序计数器,堆栈,和用于执行这些程序的其它软件组件。PU 通过向 DMAC 发出直接存储器存取 (DMA) 命令来控制 APU。

[0123] 图 23 示出了软件单元 102 的结构。如该图所示,软件单元 (例如软件单元 2302) 包含路由信息部份 2304 和主体 2306。路由信息部份 2304 中包含的信息取决于网络 104 的协议。路由信息部份 2304 包含标头 (header) 2308,目的 ID 2310,源 ID 2312 和应答 ID 2314。目的 ID 包括网络地址。例如,在 TCP/IP 协议下,网络地址是网际协议 (IP) 地址。目的 ID 2310 还包括 PE 和 APU 的身份信息,其中应当向该 PE 和 APU 发送单元以进行处理。源 ID 2314 包含网络地址,并标识发出单元的 PE 和 APU,以便在必要时允许目的 PE 和 APU 得到关于单元的附加信息。应答 ID 2314 包含网络地址,并标识关于单元的查询,和单元处理的结果应当被送往的 PE 和 APU。

[0124] 单元主体 2306 包含与网络协议无关的信息。图 23 的分解部分示出了单元主体 2306 的详细情况。单元主体 2306 的标头 2320 标识单元主体的开始。单元接口 2322 包含使用单元所必需的信息。该信息包括全局唯一 ID 2324,所需的 APU 2326,沙箱大小 2328,以及前一个单元 ID 2330。

[0125] 全局唯一 ID 2324 在整个网络 104 中唯一标识软件单元 2302。根据源 ID 2312,例如源 ID 2312 中的 PE 或 APU 的唯一标识,以及软件单元 2302 的产生或发送时间和日期产生全局唯一 ID 2324。所需的 APU 2326 提供执行该单元所需的最小数量的 APU。沙箱大小 2328 提供所需 APU 中为执行单元而需要的相关 DRAM 中的受保护存储器的数量。前一个单元 ID 2330 提供需要顺序执行 (例如流数据) 的一组单元中的前一个单元的身份信息。

[0126] 实现部份 2332 包含单元的核心信息。该信息包括 DMA 命令表 2334,程序 2336,以及数据 2338。程序 2336 包含要由 APU 运行的程序 (称为“apulets”),例如 APU 程序 2360 和 2362,而数据 2338 包含要通过这些程序处理的数据。DMA 命令表 2334 包含开始程序所需要的一系列 DMA 命令。这些 DMA 命令包括 DMA 命令 2340、2350、2355 和 2358。PU 向 DMAC 发出这些 DMA 命令。

[0127] DMA 命令 2340 包括 VID 2342。VID 2342 是 APU 的虚拟 ID,它在 DMA 命令发出时被映射到物理 ID。DMA 命令 2340 还包括装入命令 2344 和地址 2346。装入命令 2344 指示 APU 将特定信息从 DRAM 读取到局部存储器中。地址 2346 提供 DRAM 中包含该信息的虚拟地址。该信息可以是例如来自程序部份 2336 的程序,来自数据部份 2338 的数据,或者其它数据。最后,DMA 命令 2340 包括局部存储器地址 2348。该地址标识局部存储器中应当装入信息的地址。DMA 命令 2350 包含类似的信息。也可能有其它的 DMA 命令。

[0128] DMA 命令表 2334 还包括一系列的启动 (kick) 命令,例如启动命令 2355 和 2358。启动命令是由 PU 向 APU 发出的、用来启动单元的处理的命令。DMA 启动命令 2355 包括虚拟 APU ID 2352,启动命令 2354,以及程序计数器 2356。虚拟 APU ID 2352 标识要被启动的 APU,启动命令 2354 提供有关的启动命令,程序计数器 2356 提供针对用于执行程序程序的程序计数器的地址。DMA 启动命令 2358 提供针对同一 APU 或另一 APU 的类似信息。

[0129] 如前面提到的,PU 把 APU 看作独立的处理器,而不是协处理器。因此,为了控制 APU 执行的处理,PU 使用类似于远程过程调用的命令。这些命令被叫做“APU 远程过程调

用”(ARPC)。PU 通过向 DMAC 发出一系列 DMA 命令来执行 ARPC。DMAC 将 APU 程序及其相关的栈帧装入到 APU 的局部存储器中。然后,PU 向 APU 发出初始启动来执行 APU 程序。

[0130] 图 24 说明了用于执行 apulet 的 ARPC 的步骤。图 24 的第一部分 2402 说明了在启动指定 APU 对 apulet 的处理时 PU 执行的步骤,并且图 24 的第二部分 2404 说明了指定 APU 在处理 apulet 时执行的步骤。

[0131] 在步骤 2410 中,PU 对 apulet 进行评估,并且然后指定用于处理 apulet 的 APU。在步骤 2412 中,PU 通过向 DMAC 发出设定用于所需的一或多个沙箱的存储器存取关键字的命令,在 DRAM 中分配用于执行 apulet 的空间。在步骤 2414 中,PU 使能针对指定 APU 的中断请求,以通知 apulet 的完成。在步骤 2418 中,PU 向 DMAC 发出将 apulet 从 DRAM 装入到 APU 的局部存储器中的 DMA 命令。在步骤 2420 中,执行该 DMA 命令,并且将 apulet 从 DRAM 读取到 APU 的局部存储器中。在步骤 2422 中,PU 向 DMAC 发出将与 apulet 相关的栈帧从 DRAM 装入到 APU 的局部存储器中的 DMA 命令。在步骤 2423 中,执行该 DMA 命令,并且将栈帧从 DRAM 读取到 APU 的局部存储器中。在步骤 2424 中,PU 发出使 DMAC 向 APU 分配关键字的 DMA 命令,以允许 APU 针对在步骤 2412 中指定的一或多个硬件沙箱读/写数据。在步骤 2426 中,DMAC 用分配给 APU 的关键字更新关键字控制表 (KTAB)。在步骤 2428 中,PU 向 APU 发出启动程序处理的 DMA 命令“启动”。依据特定的 apulet,PU 在执行特定 ARPC 时可以发出其它 DMA 命令。

[0132] 如上面所指出的,图 24 中的第二部分 2404 说明了 APU 在执行 apulet 时所执行的步骤。在步骤 2430 中,APU 响应在步骤 2428 发出的启动命令而开始执行 apulet。在步骤 2432 中,在 apulet 的指示下,APU 评估 apulet 的相关栈帧。在步骤 2434 中,APU 向 DMAC 发出多个 DMA 命令,以便将根据栈帧的需要而指定的数据从 DRAM 装入到 APU 的局部存储器中。在步骤 2436 中,执行这些 DMA 命令,并且将数据从 DRAM 读取到 APU 的局部存储器中。在步骤 2438 中,APU 执行 apulet 并产生结果。在步骤 2440 中,APU 向 DMAC 发出把结果存储在 DRAM 中的 DMA 命令。在步骤 2442 中,执行该 DMA 命令,并将 apulet 的结果从 APU 的局部存储器写到 DRAM 中。在步骤 2444 中,APU 向 PU 发出中断请求,以通知 ARPC 已经完成。

[0133] APU 在 PU 的指示下独立执行任务的能力允许 PU 将一组 APU,以及与一组 APU 相关的存储器资源专用于执行长时间 (extended) 任务。例如,PU 可以将一或多个 APU,以及与这一或多个 APU 相关的一组存储器沙箱专用于接收在长时间段内通过网络 104 发送的数据,并且将该期间内接收到的数据指引到一或多个 APU 及其相关的存储器沙箱,以便进行进一步的处理。该能力尤其利于处理通过网络 104 传输的流数据,例如 MPEG 流或流式 ATRAC 音频或视频数据。PU 可将一或多个 APU 及其相关的存储器沙箱专用于接收这些数据,并将一或多个其它 APU 及其相关的存储器沙箱专用于解压缩和进一步处理这些数据。换句话说,PU 可在一组 APU 及其相关的存储器沙箱中间建立用于处理这些数据的专用流水线关系。

[0134] 然而,为了有效地执行这种处理,在没有发生包含数据流的 apulet 处理时,流水线的专用 APU 和存储器沙箱应当保持为该流水线专用。换句话说,在此期间专用 APU 及其相关的沙箱应处于保留状态。在完成 apulet 处理时进行的 APU 及其相关一或多个存储器沙箱的保留被称为“常驻终止 (resident termination)”。响应来自 PU 的指令发生常驻终止。

[0135] 图 25、图 26A 和图 26B 说明了包含用于处理流数据 (例如 MPEG 流数据) 的一组

APU 及其相关沙箱的专用流水线结构的建立。如图 25 所示,流水线结构的部件包括 PE 2502 和 DRAM 2518。PE 2502 包括 PU 2504、DMAC 2506 和多个 APU,包括 APU 2508、APU 2510 和 APU 2512。PU 2504、DMAC 2506 和这些 APU 中间的通信通过 PE 总线 2514 进行。宽带总线 2516 将 DMAC 2506 连接到 DRAM 2518。DRAM 2518 包括多个沙箱,例如沙箱 2520、沙箱 2522、沙箱 2524 和沙箱 2526。

[0136] 图 26A 说明了建立专用流水线的步骤。在步骤 2610 中,PU 2504 分配 APU 2508 以处理网络 apulet。网络 apulet 包含用于处理网络 104 的网络协议的程序。在这种情况下,该协议是传输控制协议/网际协议(TCP/IP)。符合该协议的 TCP/IP 数据分组通过网络 104 进行传输。一旦接收到数据分组,APU 2508 处理这些数据分组,并将分组中的数据装配到软件单元 102 中。在步骤 2612 中,在网络 apulet 的处理完成时,PU 2504 便命令 APU 2508 执行常驻终止。在步骤 2614 中,PU 2504 指定 PU 2510 和 2512 来处理 MPEG apulet。在步骤 2615 中,当 MPEG apulet 处理完成时,PU 2504 也命令 APU 2510 和 2512 执行常驻终止。在步骤 2616 中,PU 2504 指定沙箱 2520 作为由 APU 2508 和 APU 2510 访问的源沙箱。在步骤 2618 中,PU 2504 指定沙箱 2522 作为由 APU 2510 访问的目的沙箱。在步骤 2620 中,PU 2504 指定沙箱 2524 作为由 APU 2508 和 APU 2512 访问的源沙箱。在步骤 2622 中,PU 2504 指定沙箱 2526 作为由 APU 2512 访问的目的沙箱。在步骤 2624 中,APU 2510 和 APU 2512 分别向源沙箱 2520 和源沙箱 2524 中的存储器块发送同步读命令,以便将这些存储器块设成阻塞状态。最后,处理转到步骤 2628,其中完成专用流水线的建立,并且保留专用于该流水线的资源。因此,APU 2508、2510、2512 及其相关沙箱 2520、2522 和 2524 进入保留状态。

[0137] 图 26B 说明了通过这个专用流水线处理 MPEG 流数据的步骤。在步骤 2630 中,处理网络 apulet 的 APU 2508 在其局部存储器中接收来自网络 104 的 TCP/IP 数据分组。在步骤 2632 中,APU 2508 处理这些 TCP/IP 数据分组,并将这些分组中的数据装配到软件单元 102 中。在步骤 2634 中,APU 2508 检查软件单元的标头 2320(图 23),以确定单元中是否包含 MPEG 数据。如果单元中不包含 MPEG 数据,则在步骤 2636 中,APU 2508 将单元发送到 DRAM 2518 中指定的通用沙箱,以便由专用流水线中未包含的其它 APU 处理其它数据。APU 2508 还将此次发送通知给 PU 2504。

[0138] 另一方面,如果软件单元包含 MPEG 数据,则在步骤 2638 中,APU 2508 检查单元的前一个单元 ID 2330(图 23),以识别单元所属的 MPEG 数据流。在步骤 2640 中,APU 2508 选择专用流水线中用于处理该单元的 APU。在这种情况下,APU 2508 选择 APU 2510 来处理这些数据。该选择是根据前一个单元 ID 2330 和负荷平衡因素来进行的。例如,如果前一个单元 ID 2330 表明该软件单元所属的 MPEG 数据流的前一个软件单元被发送到 APU 2510 进行处理,则通常也会将当前软件单元发送到 APU 2510 进行处理。在步骤 2642 中,APU 2508 发出将 MPEG 数据写到沙箱 2520 的同步写命令。由于该沙箱在之前被设成阻塞状态,所以在步骤 2644 中,自动将 MPEG 数据从沙箱 2520 读到 APU 2510 的局部存储器中。在步骤 2646 中,APU 2510 在其局部存储器中处理 MPEG 数据以产生视频数据。在步骤 2648 中,APU 2510 将视频数据写到沙箱 2522 中。在步骤 2650 中,APU 2510 向沙箱 2520 发出同步读取命令,以使该沙箱准备接收另外的 MPEG 数据。在步骤 2652 中,APU 2510 处理常驻终止。这种处理使这个 APU 进入保留状态,在保留状态期间,该 APU 等待处理 MPEG 数据流中另外的 MPEG

数据。

[0139] 在一组 APU 及其相关沙箱中间可建立用于处理其它类型据的其它专用结构。例如,如图 27 所示,可以建立专用 APU 组,例如 APU2702、2708 和 2714,用于对三维对象执行几何变换以产生二维显示表。其它 APU 可进一步处理(呈现)这些二维显示表以产生像素数据。为执行这种处理,将沙箱专用于 APU 2702、2708 和 2714,以存储三维对象,以及对这些物体进行处理而产生的显示表。例如,源沙箱 2704、2710 和 2716 专用于存储分别由 APU 2702、APU 2708 和 APU 2714 处理的三维对象。以类似的方式,目的沙箱 2706、2712 和 2718 专用于存储分别由 APU 2702、APU 2708 和 APU 2714 对这些三维对象进行处理而得到的显示表。

[0140] 协调 APU 2720 专用于在其局部存储器中接收来自目的沙箱 2706、2712 和 2718 的显示表。APU 2720 在这些显示表中进行仲裁,并把它们发送到用于呈现像素数据的其他 APU。

[0141] 系统 101 的处理器还使用绝对定时器。绝对定时器向 APU 及 PE 的其它单元提供时钟信号,该时钟信号独立于驱动这些单元的时钟信号,并比其更快。绝对定时器的使用示于图 28。

[0142] 如图所示,绝对定时器为 APU 执行任务确定时间预算。该时间预算提供完成这些任务的时间,该时间要比 APU 处理这些任务所需的时间要长。结果,对于每项任务,在时间预算内存在忙时间段和等待时间段。根据这个时间预算编写所有 apulet 以用于处理,而不管 APU 的实际处理时间和速度如何。例如,对于 PE 的特定 APU,特定任务可在时间预算 2804 的忙时间段 2802 中被执行。由于忙时间段 2802 比时间预算 2804 短,所以在时间预算中出现等待时间段 2806。在该等待时间段中,APU 进入休眠模式,期间 APU 消耗的功率较小。

[0143] 在时间预算 2804 期满之前,其它 APU 或 PE 的其它单元不期望得到任务处理的结果。因此,通过使用由绝对定时器确定的时间预算,不管 APU 的实际处理速度如何,APU 的处理结果总是协调的。

[0144] 将来,APU 的处理速度会变得更快。然而,由绝对定时器确定的时间预算将保持相同。例如,如图 28 所示,将来的 APU 将在更短的时间内执行任务,因而将有更长的等待时间段。因此,忙时间段 2808 比忙时间段 2802 短,而等待时间段 2810 比等待时间段 2806 长。然而,由于根据由绝对定时器确定的相同时间预算编写程序以进行处理,因此,仍然保持 APU 间处理结果的协调。结果,较快的 APU 可以处理为较慢 APU 编写的处理程序,而不会在期待该处理结果时引起冲突。

[0145] 作为对用于建立 APU 之间的协调的绝对定时器的代替,对于因增强的或不同的运算速度而产生的 APU 并行处理的协调问题,PU 或一或多个指定 APU 可分析 APU 在处理 apulet 时执行的特定指令或微代码。可以在指令中插入“无操作”(“NOOP”)指令,并由某些 APU 来执行,以便保持 APU 的处理按 apulet 预期的正确顺序完成。通过在指令中插入这些 NOOP,可以保持 APU 执行所有指令的正确定时。

[0146] 如上所述,每个处理元件(PE)包括至少一个处理单元(PU)和一个或多个附连处理单元(APU),以便利用由 PU 协调和控制的 APU 执行依据一个或多个应用程序的数据的并行处理。本文中,暂时地回到图 18,示出了用于在 APU 和共享存储器之间来回传递数据的示例性数据级同步(DLS)机制。然而,如前所述,该 DLS 机制还可以有利地用于处理到达和 /

或来自外部设备的数据。实际上,如从下面描述所看到的,将 DLS 用于 I/O 设备处理提供了其它重要好处。例如,现在存在隐藏设备实现细节的能力、处理多设备的灵活性、以及 APU 处理来自一个或多个外部源的流数据的能力。现在注意力应当被引到图 29,该图示出了根据发明构思的示例性 DLS 体系结构。

[0147] 图 29 中,结构 2900 包括 PE 2990、DRAM(或共享存储器)2910、接口设备(接口)2915 和外部设备 2920。PE 2990 代表一个或多个处理元件,并且进一步包括至少一个 PU 2905,以及如 APU 2901、2902 和 2903 所代表的一个或多个 APU。经由总线 2906 将 PE 2990 连接到共享存储器,并且 PE 2990 使用先前所描述的用于将数据写入 DRAM 2910,以及从 DRAM 2910 读取数据的 DLS 和存储器保护机制。根据发明的构思,现在 DLS 机制被延伸用于外部设备。同样地,如从图 29 所看到的,经由总线 2907 将 PE 2990 连接到接口 2915,而经由总线 2908 将 DRAM

[0148] 2910 连接到接口 2915。后者经由路径 2916 提供了到由外部设备 2920 所代表的一个或多个外部设备的连接。外部设备 2920 可以是任意数目的设备之一,包括但不限于诸如硬盘或光盘的存储器,诸如 I/O 处理器或图形处理器的处理器,以及收发器(例如网络收发器)。路径 2916 代表用于将外部设备 2920 与接口 2915 连接的一个或多个通信路径,例如分组网络连接、交换网络连接、串行总线、并行总线等。如下所述,总线 2908 提供了由接口 2915 到 DRAM 2910 的直接访问。

[0149] 现在参照图 30,首先将在外部存储器 3020 作为外部设备的情况下说明本发明构思。具体地说,每个 APU 具有与其相关的一个或多个通道,所述通道用于将请求经由总线 2907 传送到接口 2915 和/或从接口 2915 接收应答。图 30 中用通道 0(请求)(2930)和通道 1(应答)(2935)针对 APU 2901 对此进行了说明。元件 2930 和 2935 两个都是示例性的一个或多个先进先出(FIFO)缓冲器。为简单起见,随后通过总线 2907 传送的其它 APU 的请求和应答通道由图 30 中虚线箭头表示。此时假定元件 2930 和 2935 中仅存储了命令和状态信息。请求和应答通道通过总线 2907 与传送控制器 2945 交换命令和状态信息。传送控制器 2945 是示例性的外部 DMAC,其处理来自 APU 的请求。从图 30 中可以看到接口 2915 包括保护表 2940。这是在 PU 2905 的内部 DMAC(未示出)中对参考图 19 如上所述的关键字控制表 1902 的补充。通过利用外部保护表,例如保护表 2940,有利于数据直接从外部存储器 3020 输送到 DRAM2910 中的一个或多个沙箱中。保护表 2940 的值由 PU 以类似于先前所述的方式示例性地分配。利用保护表 2940,传送控制器 2945 控制沿 DRAM2910 中的沙箱和外部存储器 3020 之间任意方向的用于传送数据的传送路径 2950。应当注意到,不需要外部保护表。然而,如果没有外部保护表,则最好从 PU,例如从与 PU 相关的类似构造的请求/应答通道传递合适的许可。正如以上的讨论,PU 执行“受托程序”,并且因而应当正分配相关的读/写许可。进一步地,保护表 2940 可被扩充到控制访问外部设备或其部分,如由外部存储器 3020 的特定存储器地址区域所表示的。

[0150] 继续参考图 30,并且特别参考图 31 和 32,说明一种用于将数据从外部存储器 3020 传送到 APU 2901 的局部存储器(未示出)的示例性 DLS 方法。该方法是上面所描述的图 24 的一个变化,并且同样地,下面不重复或在图 31 中示出全部步骤。在该例子中,假定 apulet(先前描述的一种软件)将由 APU 2901 执行,其需要对当前存储在外部存储器 3020 中的数据进行操作。首先参照图 31 的步骤 3105,PU 2905 为 APU2901 的使用分配 DRAM 2910

的存储器。此外,在步骤 3110 中,PU 2905 经由总线 2907 将所分配存储器的适当关键字保护值提供给接口 2915 的传送控制器 2945,以用在用于控制访问 DRAM 2915 的存储器的保护表 2940 中。除了更新其自身关键字控制表以控制 APU 2901 的访问之外,PU 2905 以如上参考图 19 所述的方式实现此操作。在步骤 3115 中,PU2905 启动 APU 2901 对 apulet 的执行,所述 apulet 需要访问当前存储在外存储器 3020 中的数据。现在参照 APU 2901,在步骤 3130 中,该 APU 经由通道 0 向接口 2915 提供请求,以从外部设备(例如外部存储器 3020)向 DRAM 2910 传送数据。该请求用于针对 DRAM 2910 的同步写操作,并且方式类似于先前所描述的 DMA 命令。示例性地,该请求包括例如外部设备(此处为外部存储器 3020)以及要使用的外部设备的部分(此处为外部存储器 3020 中的数据的位置)的标识、将被传送的量、DRAM2910 中的相应地址,以及操作种类是读还是写操作(此处为对 DRAM2910 的同步写操作)。例如,在 apulet 相关的信息中,提供了外部设备信息,例如要使用的设备、设备的一部分以及大小信息。也应注意到,可以使用其它类型外部设备参数,包括但不限于设备和端口号、虚通道标识符等。

[0151] 现在参照图 32,并且继续参考图 30,在步骤 3205 中,接口 2915 的传送控制器 2945 从 APU 2901 接收传送请求。在步骤中 3210,传送控制器 2945 检查存储在保护表 2940 中的值,以确定 DRAM 2910 中的所请求的存储器范围是否有效。如果存储器范围是无效的,则在步骤 3220 中,传送控制器 2945 经由通道 1 拒绝传送。然而,如果存储器范围有效,则在步骤 3225 中,传送控制器开始传送,并且经由传送路径 2950 以 DMA 方式继续传送数据(步骤 3230),直至结束。当传送完成时,在步骤 3235 中,传送控制器 2945 经由应答通道发送传送完成消息。应当注意,如果外部设备信息也被存储在保护表 2940 中,例如在步骤 3210 和 3215 中,在继续执行传送之前,传送控制器 2945 还将检查 APU 2901 是否具有使用该外部设备的许可。

[0152] 根据本发明,利用如上参考图 18 中的状态图描述的 DLS 进行从外部存储器 3020 到 APU 2901 的数据传送。图 33 中说明了本文中 DLS 的使用,其示出了数据流(由虚线箭头所表示的)。由接口 2915 对 DRAM2910 执行同步写操作,而由 APU 2901 执行同步读操作,以便从 DRAM2910 读取数据来存储在局部存储器 2981 中。如上参考图 17B 到 17D 所述,由 DRAM 2910 执行的单独同步写操作和同步读操作的处理由共享存储器的空、阻塞和满状态(例如,用 F/E 位的值、APU ID 和相关状态信息的 LS 地址字段表示)来控制。因此,可以以一种高效的,即低系统开销的方式从外部设备传送数据到 APU 的局部存储器,而不需要例如为了 APU 使用,由 PU 首先传送数据到 DRAM。换言之,为了由 APU 执行,将可能已经由 PU 执行的 I/O 处理以一种安全和同步的方式向更下层推入到体系结构中。这使 APU 能够例如在多流应用中迅速从外部存储器的多缓冲区得到数据并且对数据进行操作。

[0153] 应当注意,同步读操作,即从 DRAM 2910 到外部设备(例如外部存储器 3020)的传送,也可以以类似于上面描述的方式进行。图 34 中继续参考图 31 和 32 的流程图说明了这种方式。例如,在步骤 3115 中,PU 2905 启动 APU 2901 对 apulet 的执行,所述 apulet 需要传送数据以便在外存储器 3020 中存储。同样地,APU 2901 经由通道 0 给接口 2915 提供请求,以便进行同步读操作,即从 DRAM 2910 传送数据到外部设备,即外部存储器 3020。该请求形式上类似于先前所描述的 DMA 命令。该请求包括例如外部设备(此处为外部存储器 3020)、要使用的外部设备的一部分(此处为外部存储器 3020 中的数据的位置)的标识,将

被传送的量, DRAM 2910 中的相应地址, 以及操作外部设备是读还是写操作 (此处为从 DRAM 2910 的同步读操作)。

[0154] 本文中 DLS 的使用允许接口 2915 以同步方式从 DRAM 2910 读取数据, 同时 APU 2901 正执行对 DRAM 2910 的写操作。这在图 34 中加以说明, 其中数据流为从 APU 2901 的局部存储器 2981 到外部设备, 如虚线箭头表示。本文中, DLS 的不同状态由 F/E 位的值以及接口 2915 是否已尝试读取数据来表示。同样地, 以上所描述的 DRAM 2915 的相关存储器位置的状态信息如图 35 中所示被修改。状态信息包括如上所述的 F/E 位 3501、设备 ID 3502, 以及数据应当被传送到的地址 3503。设备 ID 3502 标识设备的类型, 例如 APU 或诸如接口的另一种设备。本文中应当观察到, 由于接口 2915 已经包含具有保护表的 DMA 类型控制器, 所以可以不需要地址字段。即, 当允许同步读操作时, DRAM 2910 只需简单地给接口 2915 (如设备 ID 3502 所标识的) 提供通知。

[0155] 如上所述, 由图 30 说明的结构仅仅是示例性的, 并且可以以直接方式扩展成其它类型外部设备。例如, 考虑图 36。该图类似于图 30。同样地, 类似附图标记代表类似单元, 这里不做进一步描述。图 36 中, 外部设备由物理接收器 (PHY) 3620 表示。后者从网络 (未示出), 例如以太网经由路径 3621 接收分组通信, 并且执行接收比特流的解码以提供分组流。在该例子中, 由 APU 2901 执行的 apulet 代表接收的分组流的链路层处理。即, APU 2901 利用以上所描述的、用于处理来自 PHY 3620 的数据的 DLS 机制, 执行协议堆栈 (例如, 诸如 TCP/IP (传输控制协议 / 网际协议) 的协议堆栈) 的一部分的处理。

[0156] 继续参考图 36, 并且尤其参考图 37, 其中示出了示例性的握手序列, 以进一步说明与外部设备的数据级同步的使用。该图中, 假定共享存储器 (例如 DRAM 2910) 的相应部分始于空状态。最初, APU (例如 APU 2901) 经由请求通道向请求从外部设备到共享存储器的所标识部分的同步写操作的接口 (例如接口 2915) 发送命令。假定保护表检查通过, 则接口 2915 启动与外部设备的传送。同时, APU 试图执行从共享存储器的这些部分的同步读操作。因为接口还没有写数据, 所以共享存储器的相应部分进入阻塞状态。在这种状态下, 如前所述, APU 的局部存储器的地址被存储在共享存储器的这些部分的相关状态信息字段内。随后, 在接口对共享存储器的所标识部分执行同步写操作之后, APU 对由存储的地址值表示的地址执行同步读操作。因此, 共享存储器的这些部分回到空状态。代表其它排列的状态变化 (如图 18 所示) 的类似握手序列是简单的, 此处不做描述。

[0157] 应当观察到, 来自一个或多个 APU 的请求不必等待接口对每个请求的完成。例如, 针对来自一个或多个外部设备的数据, APU 可以发出多个请求给接口。DLS 的使用保证了数据将被保持在共享存储器中, 直到从 APU 或从外部设备 (经由接口) 进行同步读操作。同样地, DLS 的使用使突发传送和流水线突发传送能在 APU 和一个或多个外部设备之间执行。

[0158] 示例性地使用图 36 中所示的设备的本发明构思的另一种应用在图 38 中示出。APU 2901 经由接口 2915 连接到若干外部设备。这些外部设备是: USB (通用串行总线) 设备、IEEE 1394 (通常被称作“火线”) 设备、串行设备、硬盘、网络设备 (例如网络收发器) 以及光盘。其它外部设备 (未示出) 可包含诸如 I/O 处理器或图形处理器的处理器。APU 2901 发送用于从一个或多个这些外部设备读取数据的请求 4301。同样地, 通过接口 2915 的控制, 每个设备随后对其分配的 DRAM 2910 的存储器位置执行同步写操作, 所述存储器位置由存储器位置 4311、4312、4313、4314、4315 和 4316 表示 (其中的每个还说明了包含相关状

态信息,其简单地由 F/E 位表示)。通过执行同步读操作,APU 2901 经由总线 2906 从 DRAM 2910 得到数据。

[0159] 现在注意力应当被引到图 39,其说明了发明构思的另一个实施例。在图 39 中,示出了简化的流结构 3700。结构 3700 包括 PE 3790、DRAM(或共享存储器)3710、接口设备(接口)3715,以及由物理收发器(PHY)3720 表示的外部设备。应注意到,外部设备 3720 也可以是接口 3715 的一部分,例如与其集成。PE 3790 代表一个或多个处理元件,并且进一步包括至少一个 PU 3705,以及由 APU 3701、3702 和 3703 所表示的一个或多个 APU。经由总线 3706 将 PE 3790 连接到共享存储器,并且 PE 3790 使用先前所描述的用于将数据写入 DRAM 3710,以及从 DRAM 3710 读取数据的 DLS 和存储器保护机制。根据发明构思,现在 DLS 机制被延伸用于外部设备。同样地,可以从图 39 看到,经由总线 3707 将 APU 连接到接口 3715 上。具体地说,每个 APU 具有与其相关的一个或多个通道,所述通道用于经由总线 3707 将请求传送到接口 3715 和 / 或从接口 3715 接收应答,和 / 或与接口 3715 交换数据。图 39 中通过通道 0(输入)(3730)和通道 1(输出)(3735)针对 APU 3701 对此加以说明。元件 3730 和 3735 两个都是示例性的一个或多个先进先出(FIFO)缓冲器。对于其它 APU,其通过总线 3707 传送的输入输出通道由图 39 中虚线箭头表示。

[0160] 接口 3715 包括通道接口元件 3745 和 3750,其连接 APU 3701 和 PHY 3720 之间相应的输入和输出数据流。后一种元件是示例性的收发器,其用于(a)编码从 APU 3701 的输出通道经由路径 3707 和 3716 提供的数据,用于通过路径 3721 传送;以及(b)解码从路径 3721 接收的数据,以便经由路径 3716 和 3707 给 APU 3701 的输入通道提供解码数据。路径 3721 代表一个或多个通信路径,其用于将外部设备 3720 连接到例如分组网络连接、交换网络连接、串行总线、并行总线等。

[0161] 如上所述,接口 3715 包含通道接口元件 3745 和 3750。通道接口元件 3745 接收来自 APU 3701 的输出通道。根据本发明,输出通道本身包括控制通道和数据通道。控制通道传送以上所描述的请求和应答通道,而数据通道传送将被提供给外部设备的数据。在该例子中,请求通道传送命令信息,其包括例如外部设备标识(此处为 PHY 3720),以及将被执行的操作(此处为写操作)。输出通道的数据通道部分代表将由 PHY 3720 编码以便经由路径 3721 发送的数据。类似地,将通道接口元件 3745 连接到 APU 3701 的输入通道。输入通道本身包括控制通道和数据通道。控制通道传送以上所描述的请求和应答通道,而数据通道传送将被提供给 APU 的数据。在该例子中,请求通道传送命令信息,其包括例如外部设备标识(此处为 PHY 3720),以及将被执行的操作(此处为读操作)。输入通道的数据通道部分代表作为从路径 3721 接收的信号的接收结果而由 PHY 3720 解码的数据。

[0162] 应观察到,在该图解实施例中,接口 3715 内不需要保护表。具体地说,并且如下所述,根据先前所描述的 DLS 和图 18 所示的存储器保护机制,每个 APU 针对 DRAM 3710 读写数据。

[0163] 继续参考图 39,图 40 中示出了用于从外部设备传送数据到 APU2901 的局部存储器(未示出)的示例性方法。该方法假定仍执行以上所描述的图 24 的流程图,即:由 PU 分配保护表值、将 apulet 分配给例如 APU 3701 等。在该例子中,由 APU 3701 执行的 apulet 代表了接收的分组流的链路层处理,并且用于传送的分组流。即,APU2901 利用以上所描述的 DLS 机制执行协议堆栈(例如 TCP/IP)的一部分的处理。

[0164] 首先参照图 40 的步骤 3830, APU 3701 提供用于针对 PHY 3720 经由总线 3707 传送数据的请求给接口 3715。在该例子中, 该请求通过输入通道和输出通道二者的命令部分发生。然而, 注意到, 不需要使用输入通道和输出通道二者。该请求包括例如外部设备 (例如为 PHY3720) 的标识, 并且也可包括要使用的外部设备的部分 (例如特定端口), 以及外部设备操作的种类 (此处为针对输入通道的外部设备读, 以及针对输出通道的外部设备写)。假定例如在 apulet 相关信息中提供外部设备信息, 例如设备、要使用的设备的部分等。也应注意到, 可以使用其它类型的外部设备参数, 包括但不限于设备和端口号、虚通道标识符等。

[0165] 在步骤 3805 中, 接口 3715 接收请求并建立到达和来自 PHY 3720 的传送。应注意到, 就 PHY 3720 所需的特定握手而言, 假定该握手由接口 3715 执行, 并且对于 APU 3701 来说是透明的。在步骤 3710 中, 接口 3715 开始传送, 并且继续传送来自 PHY 3720 的数据, 直至结束 (步骤 3815)。例如, 这种数据传送可以一直继续, 直到针对远端点的连接 (未示出) 被破坏。针对 PHY 3720 的数据传送的该部分可以根据外部设备的种类同步或异步进行。在输入通道的数据通道部分内传送数据例如到达元件 3735 的 FIFO 的一部分。当传送完成时, 在步骤 3820 中, 接口 3715 经由输入输出通道的命令部分的应答通道发送传送完成消息。

[0166] 根据本发明, 由 APU 3701 针对外部设备 3720 的数据传送使用 DLS 来进行, 前面参考图 18 描述了 DLS 的状态图。图 41 中说明了本文中 DLS 的使用。APU 和外部设备之间的数据流由虚线箭头表示。从图 41 可以看到, APU 3701 经由总线 3707 所传送的输入通道将数据直接接收到如上所述的局部存储器中。类似地, APU 3701 经由总线 3707 所传送的输出通道直接从局部存储器提供数据。这产生了快速、低系统开销的通信通道。例如, 要经由 PHY 3720 传送的数据最初可被存储在 DRAM3710 中。APU 3701 使用 DLS、通过由图 41 中对应虚线箭头所示的同步读操作, 从 DRAM 3710 提取该数据。然后 APU 3701 根据链路层 apulet 处理该数据, 并提供处理的数据, 例如已格式化的分组流给 PHY 3720 (经由接口 3715), 以便为发送而进行进一步的编码。通过类似的方式, 由 PHY 3720 接收的数据被解码, 并经由输入通道提供给 APU3701, 以便为 APU 3701 的处理而存储在局部存储器 3781 中。APU 3701 根据链路层 apulet 处理数据, 例如移除标头、错误检查, 并且使用 DLS、通过由图 41 中相应虚线箭头所描述的同步写操作, 将处理的数据提供给 DRAM 3710。

[0167] 图 42 说明了上面参照图 39 示出并描述的简化流结构的一种变化。因为本实施例以类似于图 39 中示出的方式操作, 所以此处不详细描述全部图。图 42 的结构 4000 包括 PE 4090、DRAM (或共享存储器) 4010、接口 4015, 以及由物理收发器 (PHY) 4020 和物理收发器 (PHY) 4080 代表的外部设备。如可以从图 42 中看到的, 示例性地将接口 4015 与外部设备 4020 和 4080 集成。PE 4090 代表一个或多个处理元件, 并且进一步包括至少一个 PU 4005, 以及由 APU 4001、4002 和 4003 所代表的一个或多个 APU。经由总线 4006 将 PE 4090 连接到共享存储器, 并且 PE4090 使用先前所描述的用于将数据写入 DRAM 4010, 以及从 DRAM4010 读取数据的 DLS 和存储器保护机制。可以从图 42 看到, 经由总线 4007 将 APU 连接到接口 4015。具体地说, 每个 APU 具有与其相关的一个或多个通道, 所述通道用于经由总线 4007 将请求传送到接口 4015, 和 / 或从接口 4015 接收应答, 和 / 或与接口 4015 传送数据。图 42 中通过输入 / 输出通道 4050 和输入 / 输出通道 4055 针对 APU 4001 对此加以说明。这

两个元件都是示例性的一个或多个先进先出 (FIFO) 缓冲器。图 42 中由虚线箭头表示通过总线 4007 传送的用于其它 APU 的其它输入和输出通道。

[0168] 接口 4015 包括元件 4030 和 4035, 其中的每个分别包括两个通道接口元件 4031 和 4032, 以及 4036 和 4037。(这些是图 39 中所描述的类型)。这些元件将数据流连接到相应外部设备(此处为 4020 和 4080)。外部设备是用于向和从相应路径 4021 和 4081 传送数据的示例性收发器。外部设备 4020 和 / 或 4080 也可包括诸如硬盘或光盘的存储器, 诸如 I/O 处理器或图形处理器的处理器, 以及收发器(例如网络收发器)。

[0169] 如上所述, 在该例子中, APU 4001 访问两个输入 / 输出通道 (4050 和 4055), 每个通道进一步包括上述用于传送命令信息和数据的输入和输出通道。以类似于上面所描述的方式, APU 4001 执行用于在外部设备 (4020 和 4080) 之间转换数据的 apulet。因为 APU 4001 访问多个输入 / 输出通道, 所以 APU 4001 不需要访问 DRAM 4010。同样地, 图 42 中所示的变化提供了用于外部设备之间转换数据的高效、高速结构。

[0170] 上文仅仅说明了本发明的原理, 因而应该理解, 本领域技术人员会能够想到许多可选结构, 虽然此处未明确描述这些结构, 但是这些结构体现了本发明的原理并且在本发明的实质和范围之内。例如, 虽然已经在执行数据级同步的输入 / 输出接口的情况下描述了本发明的构思, 但是不要求接口对两种输入和输出都执行, 或仅是每种类型中的一个, 例如接口可以只是输入设备, 或只是输出设备, 或是公用输入 / 输出框架中所支持的输入设备和输出设备的任意组合。

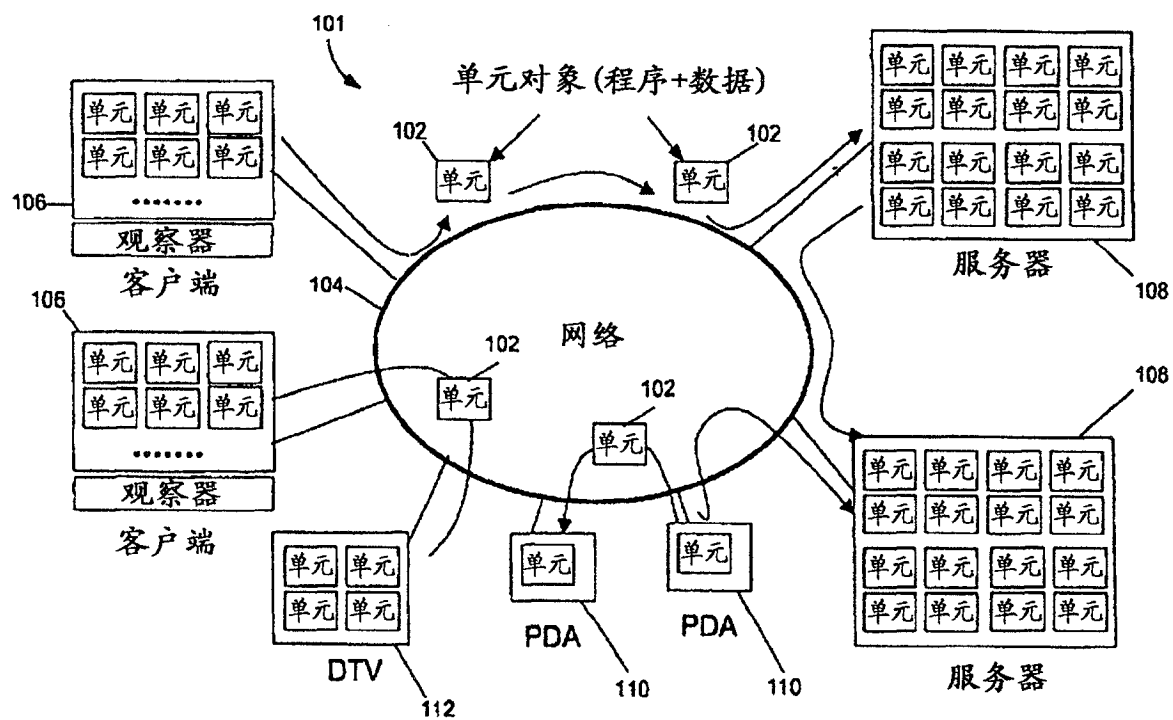


图 1

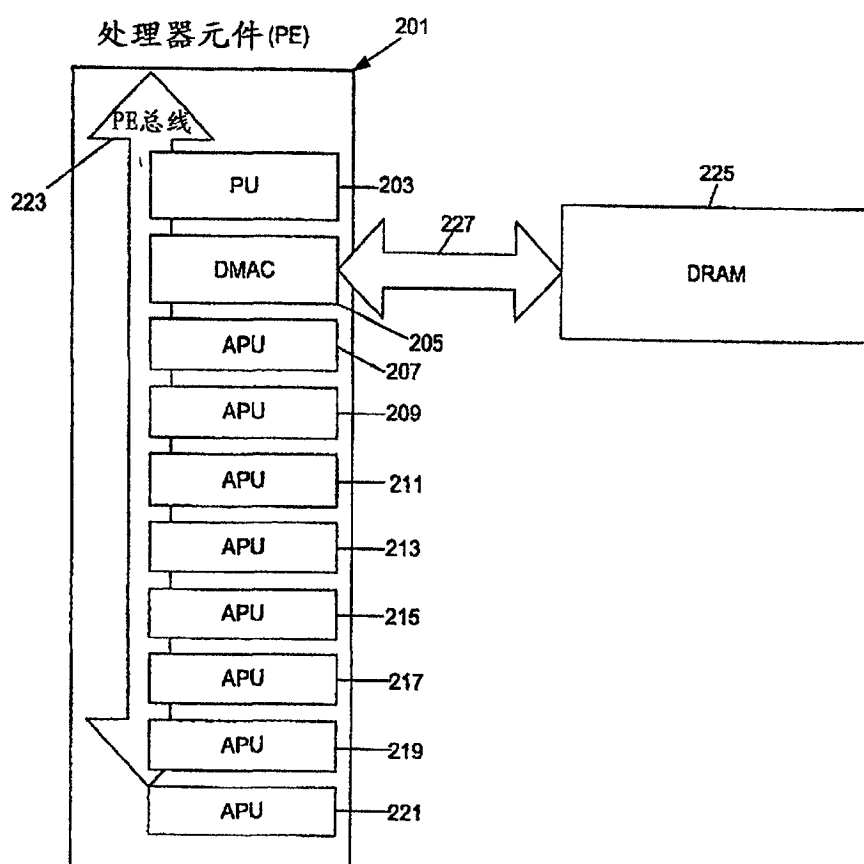


图 2

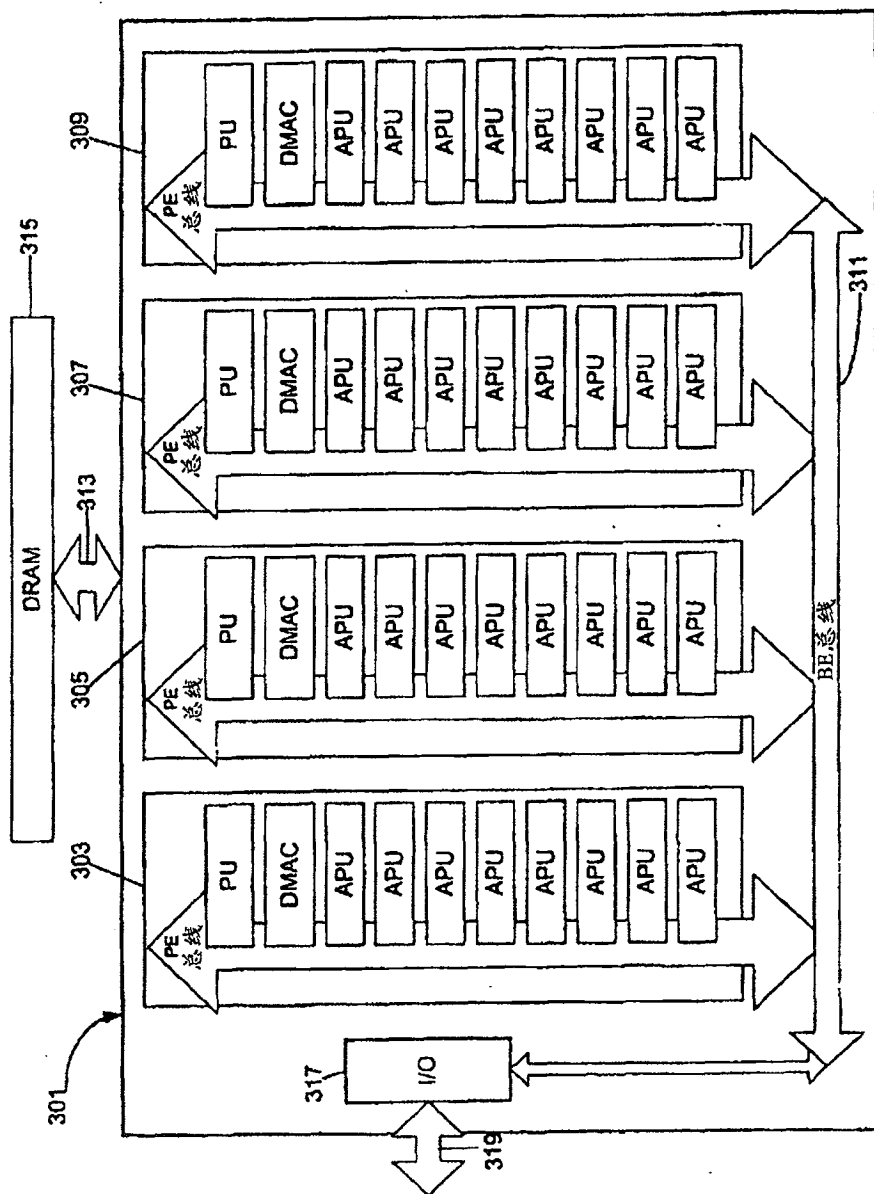


图 3

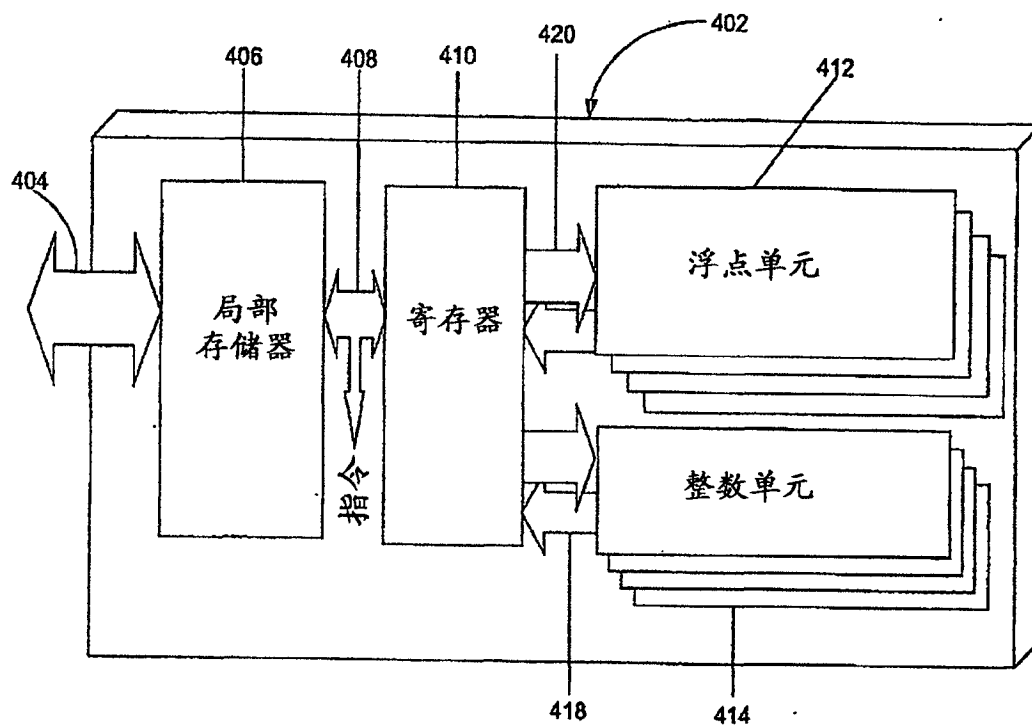


图 4

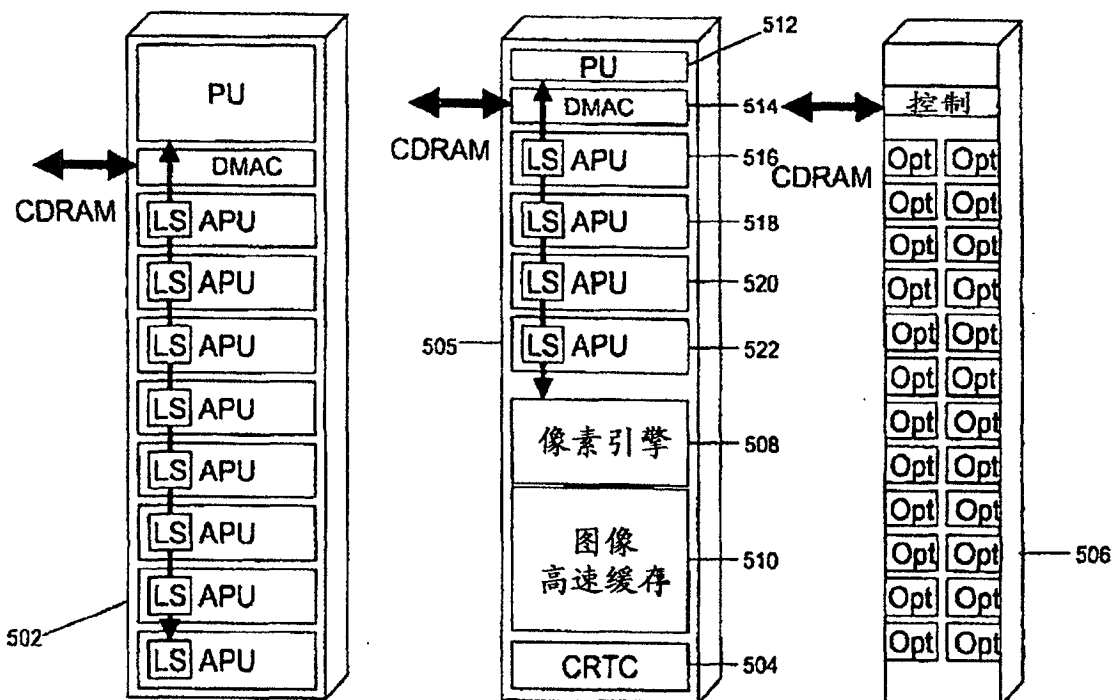


图 5

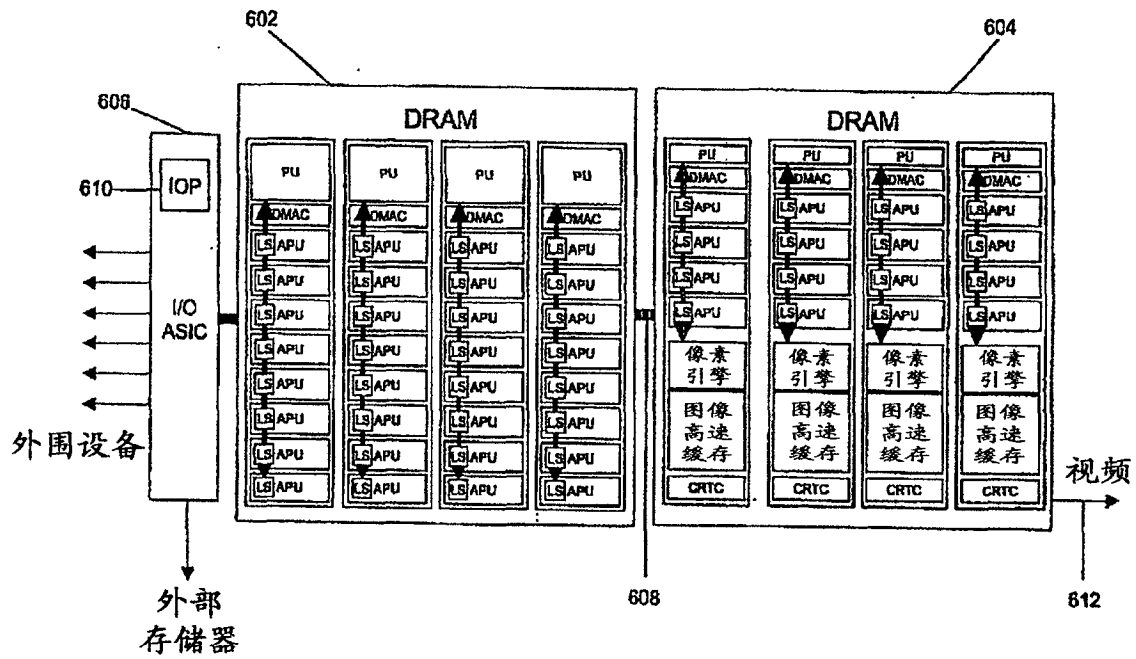


图 6

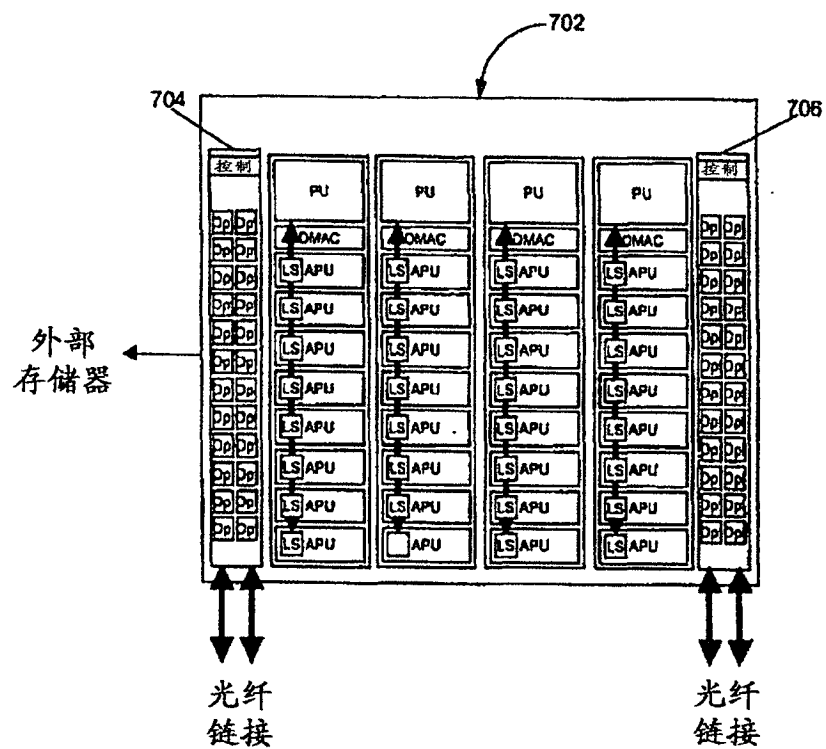


图 7

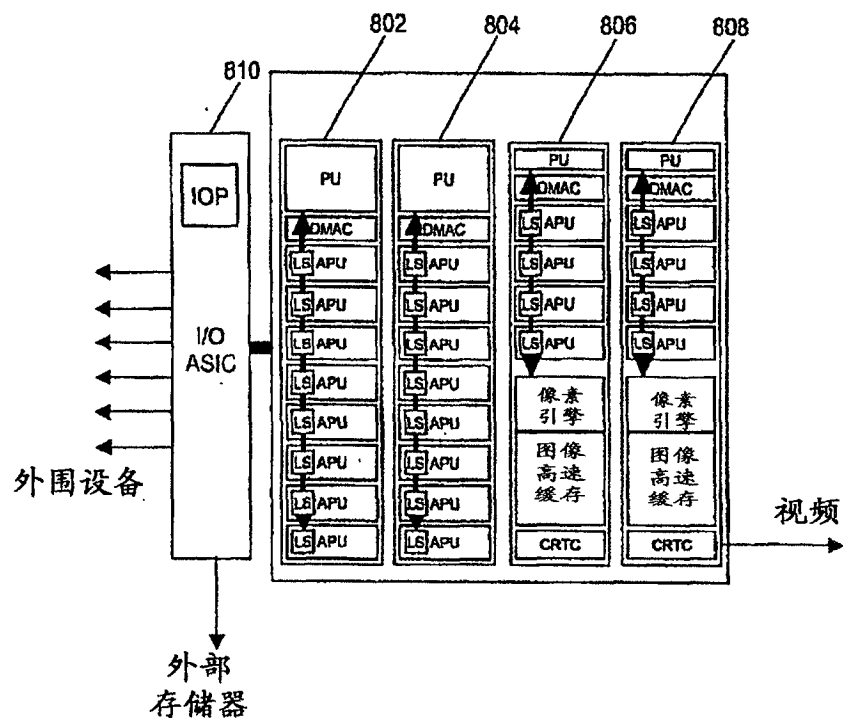


图 8

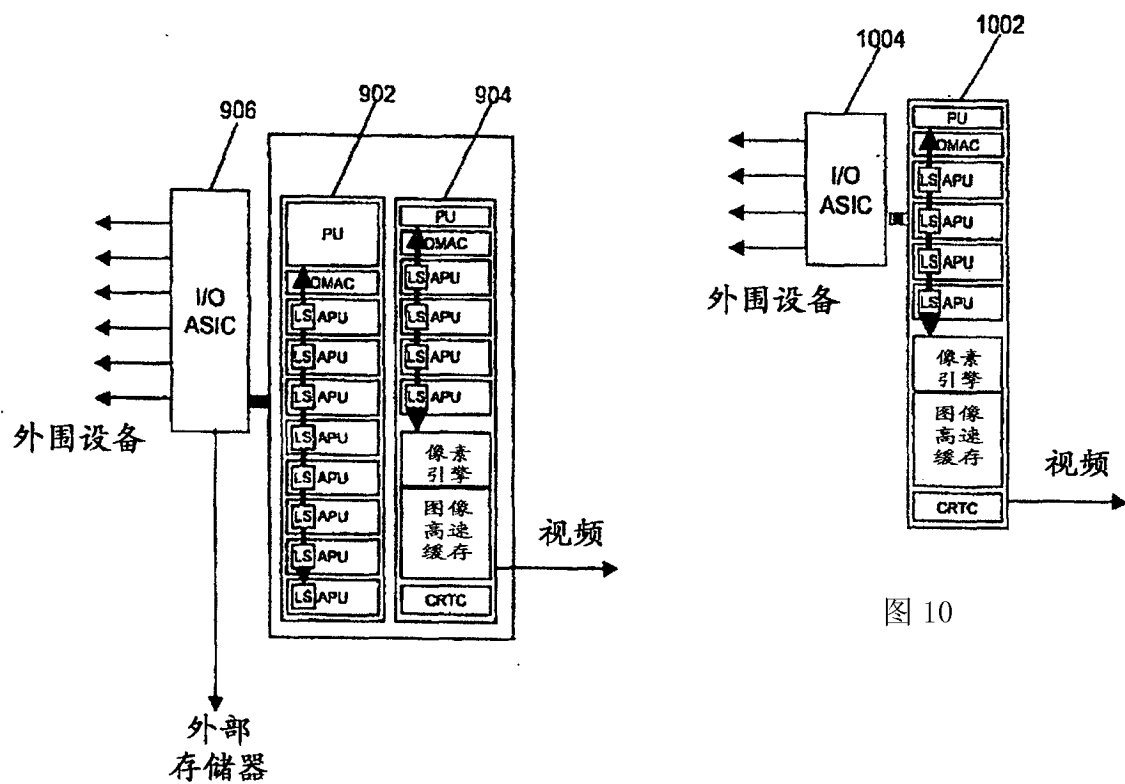


图 10

图 9

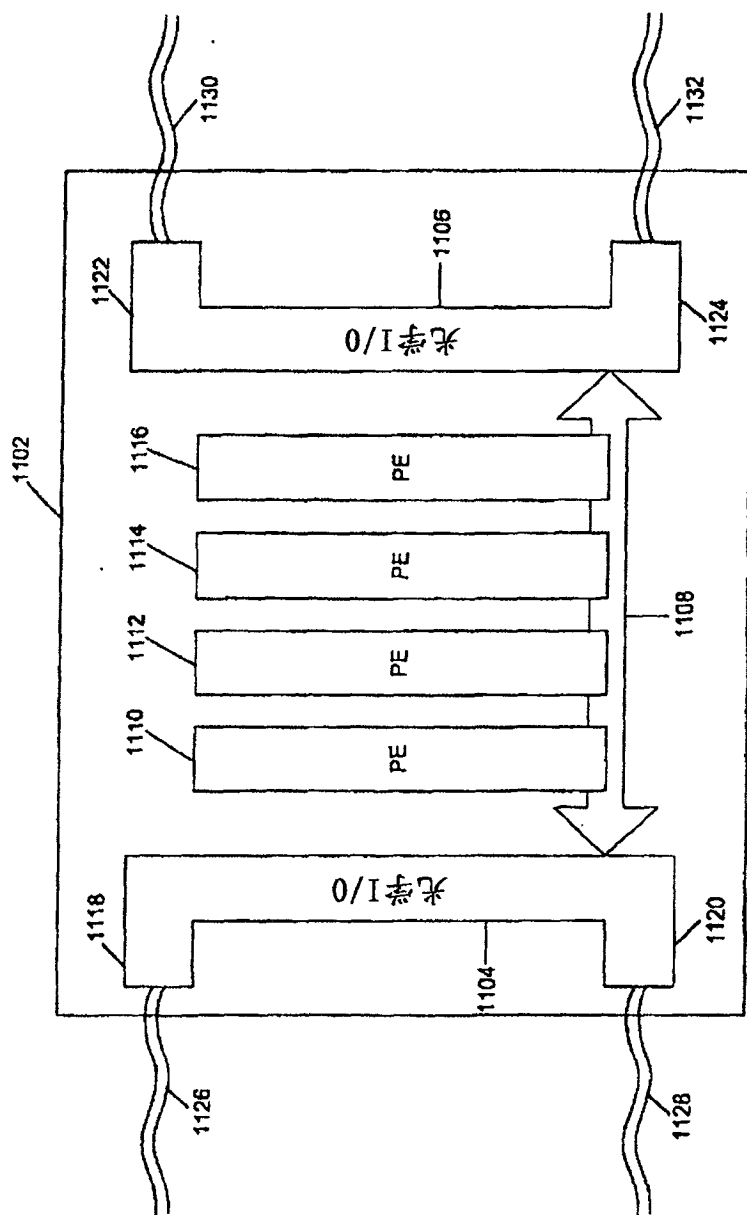


图 11A

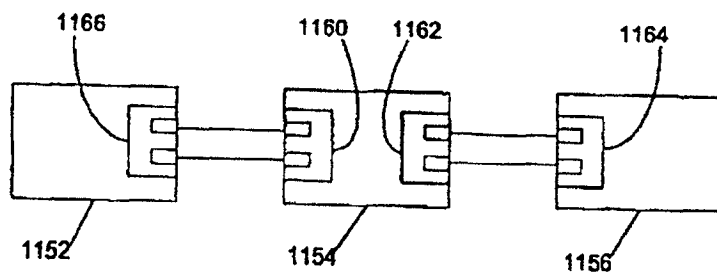


图 11B

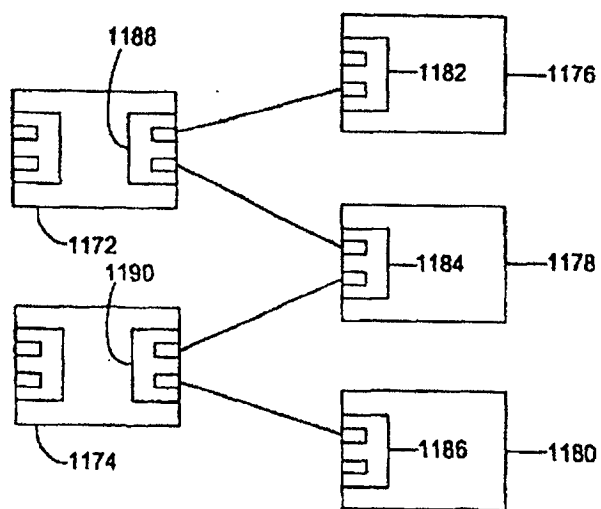


图 11C

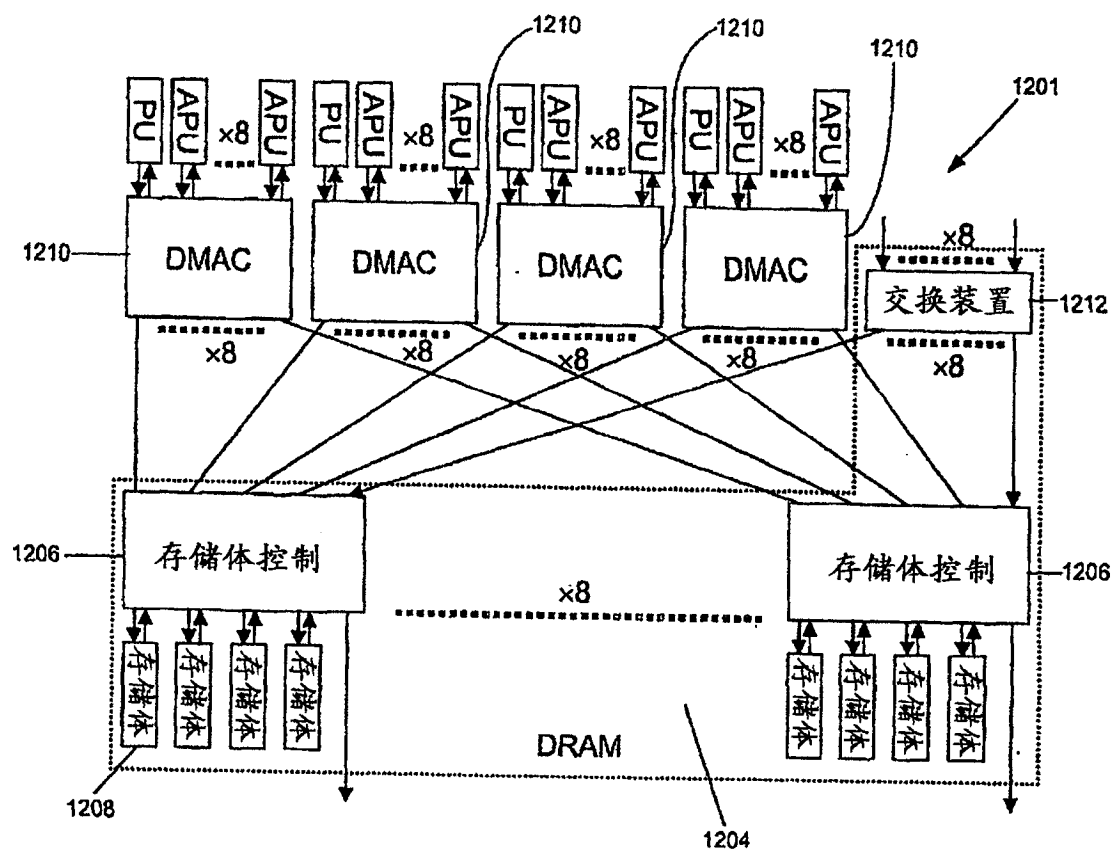


图 12A

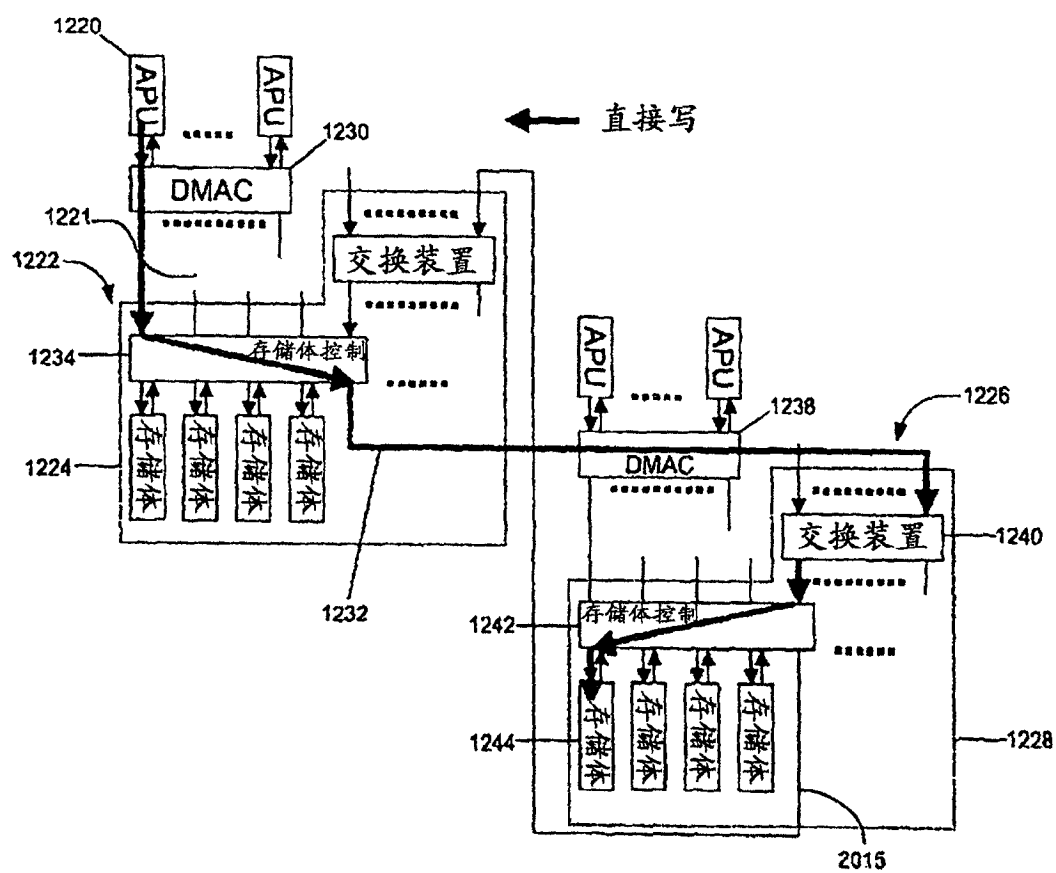


图 12B

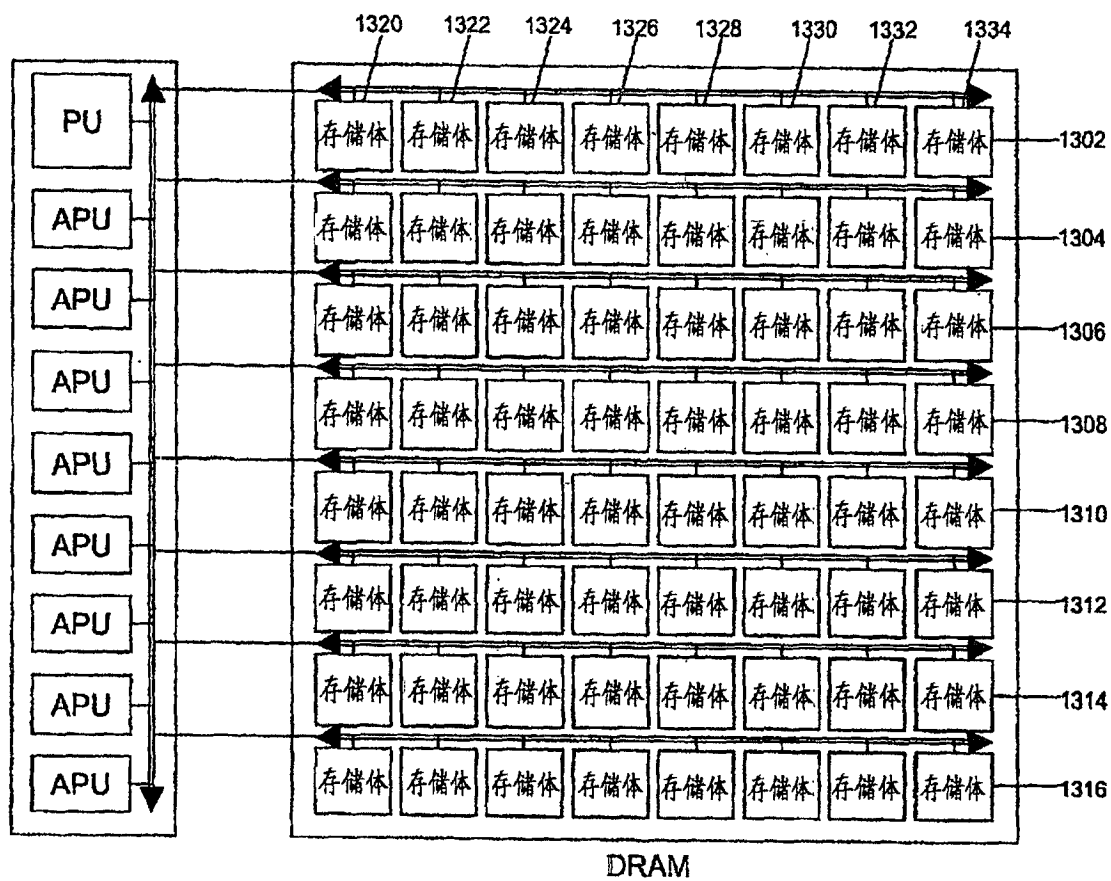


图 13

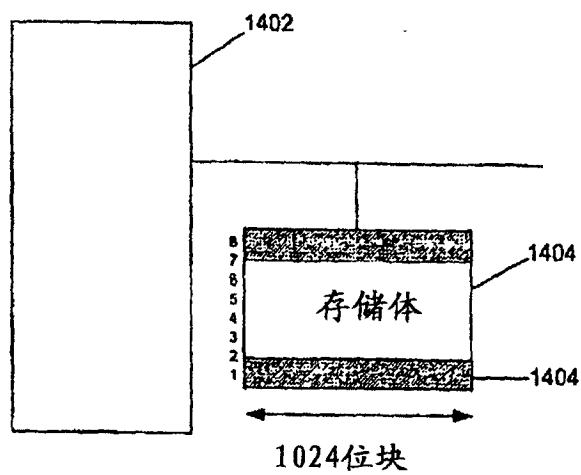


图 14A

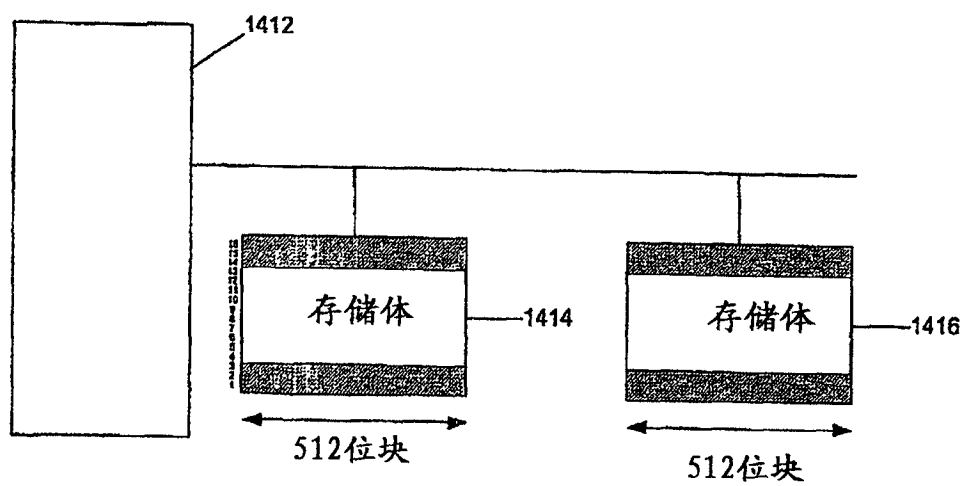


图 14B

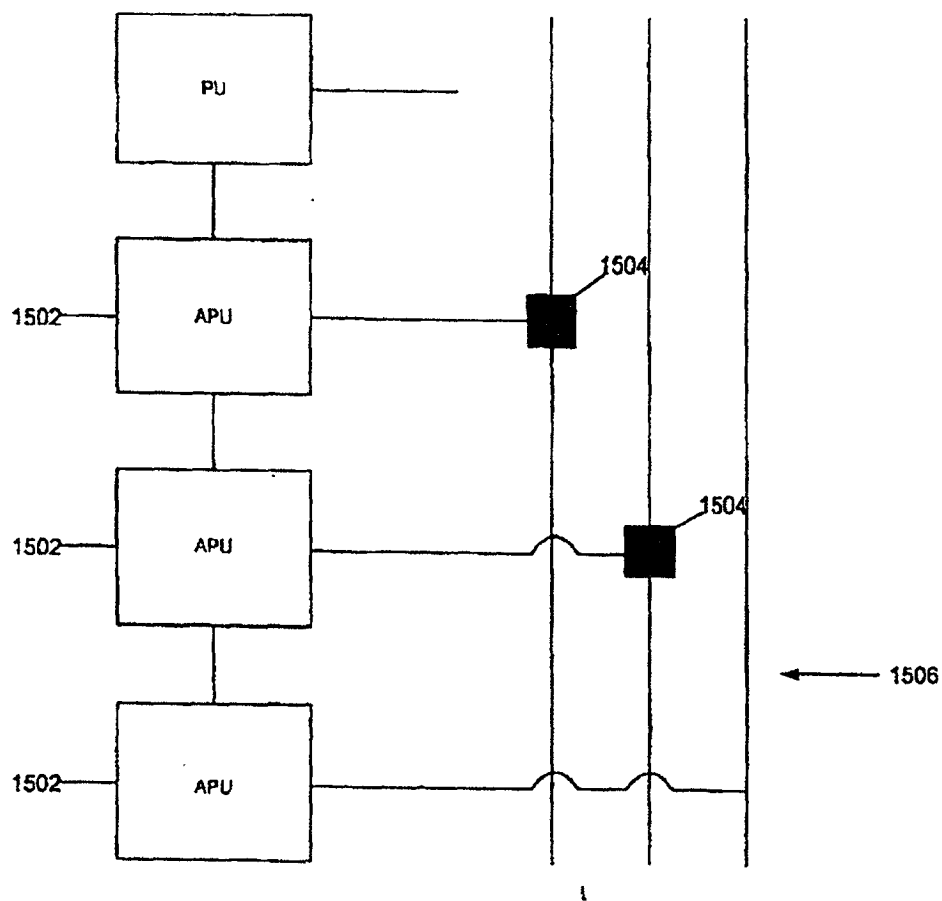


图 15

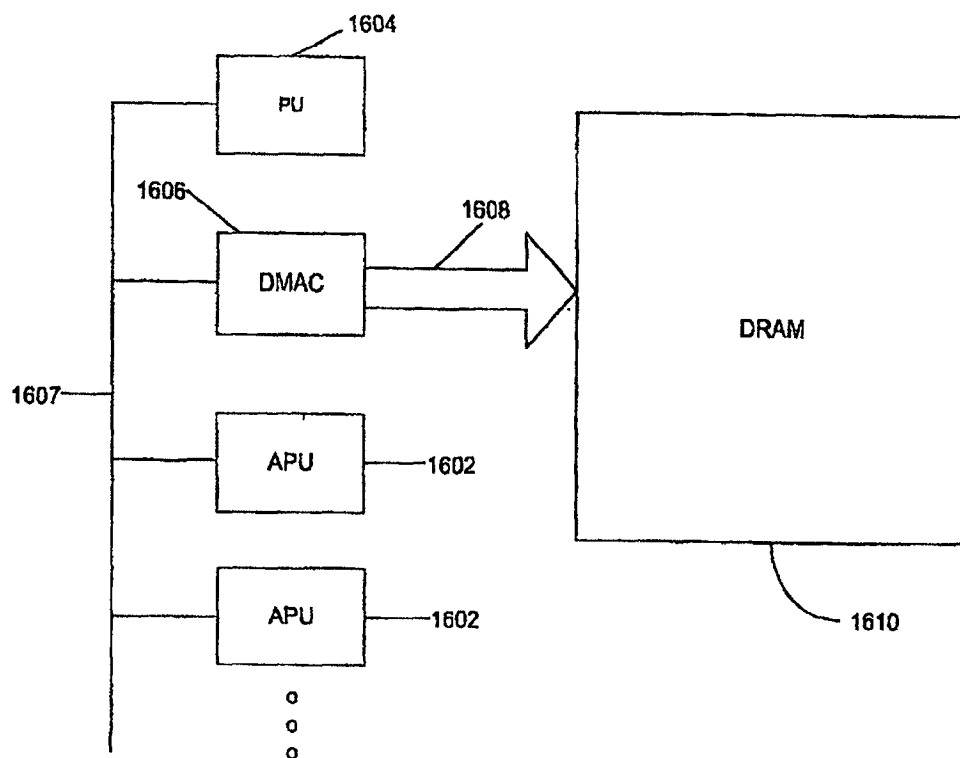


图 16

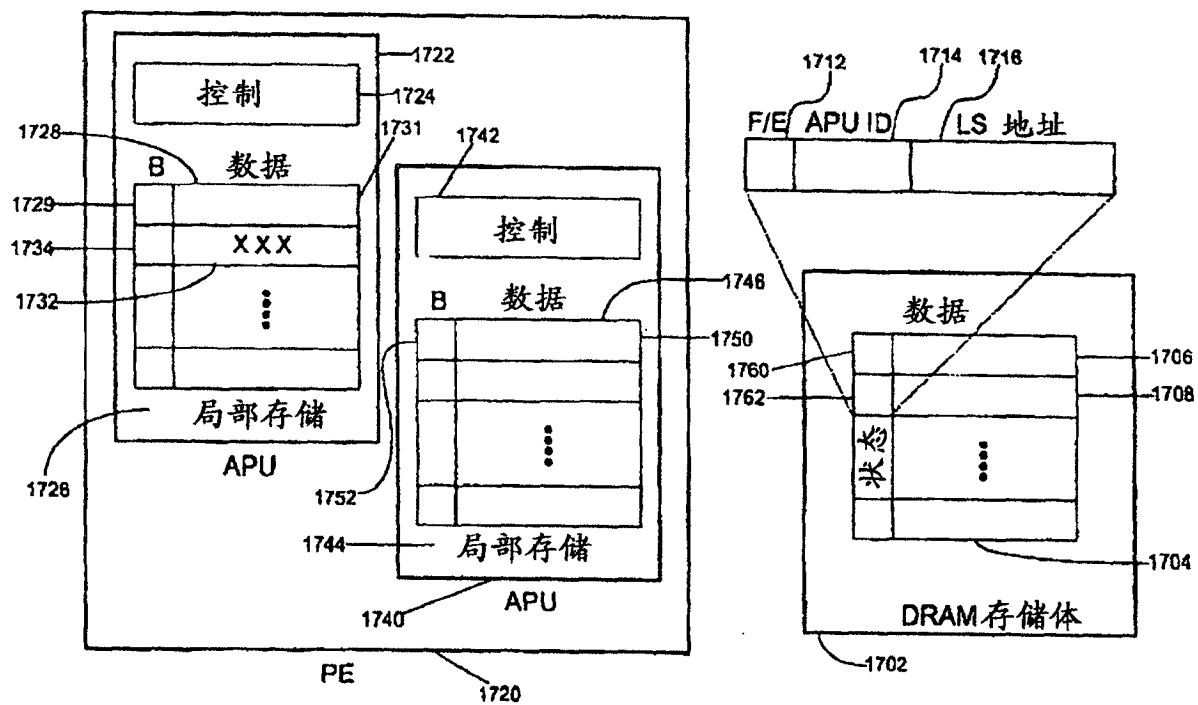


图 17A

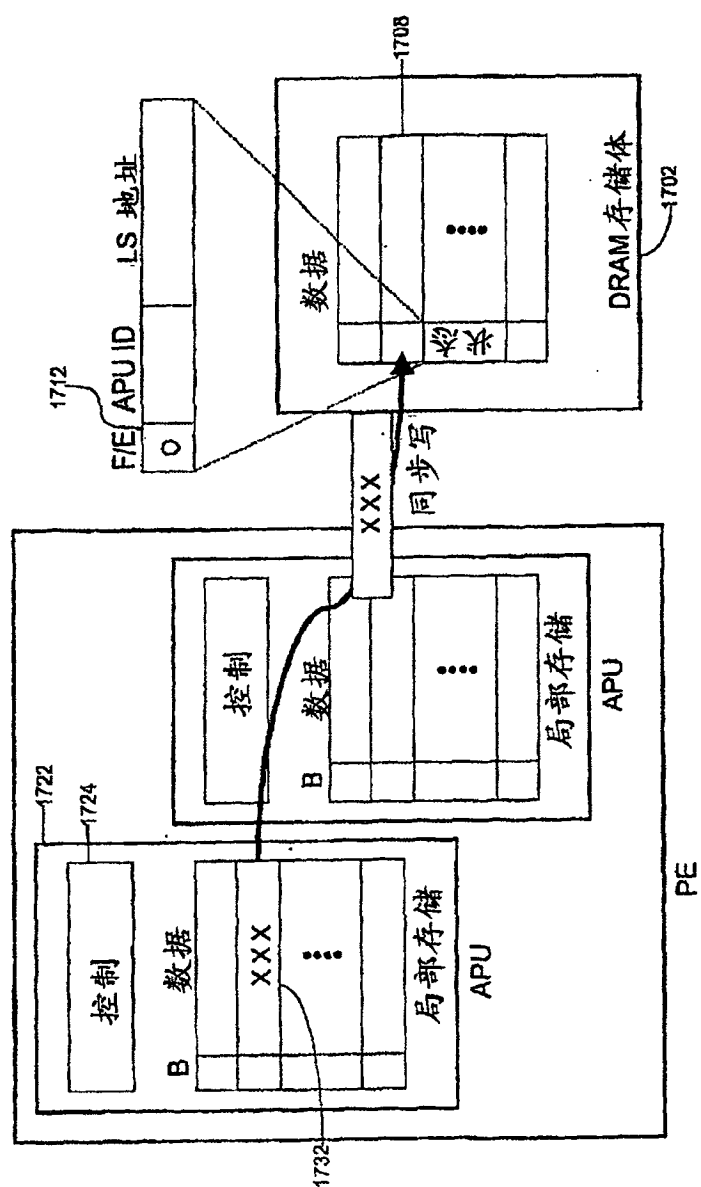


图 17B

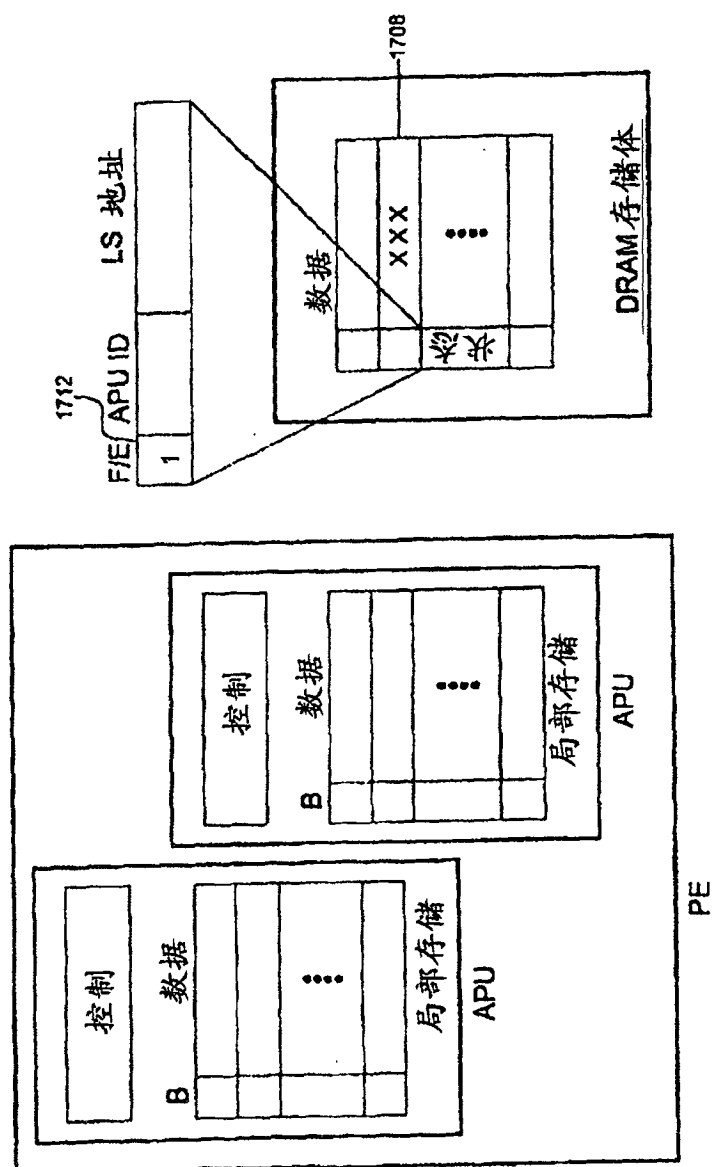


图 17C

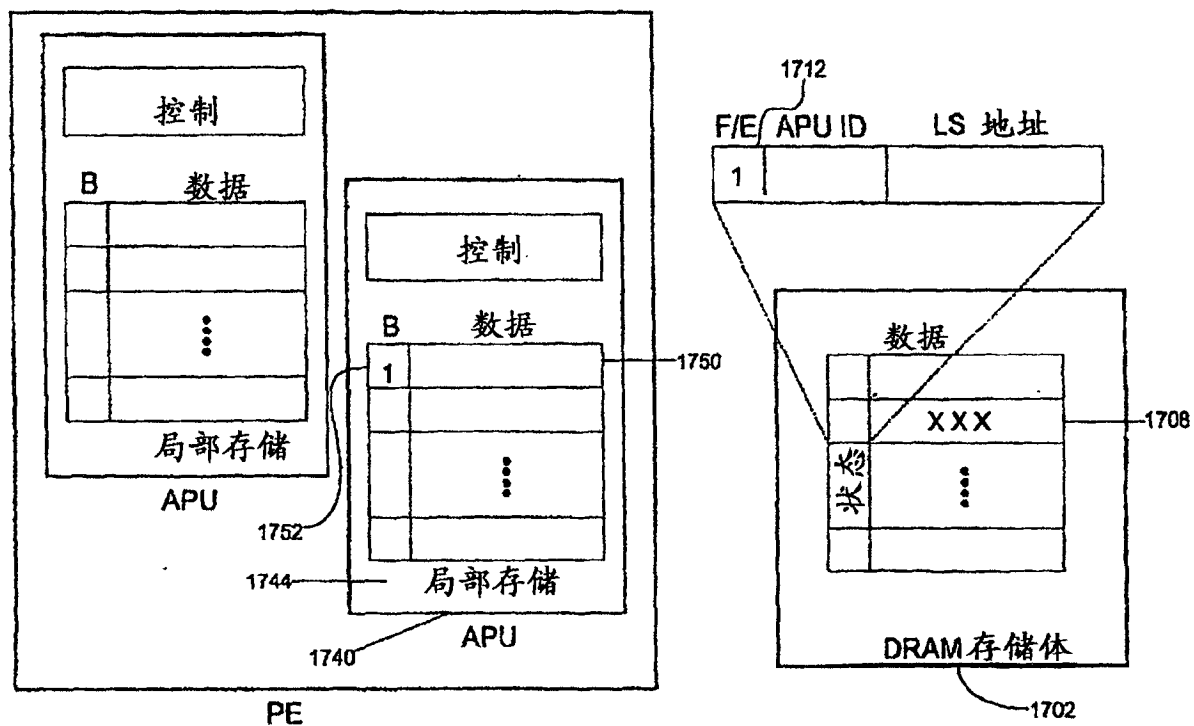


图 17D

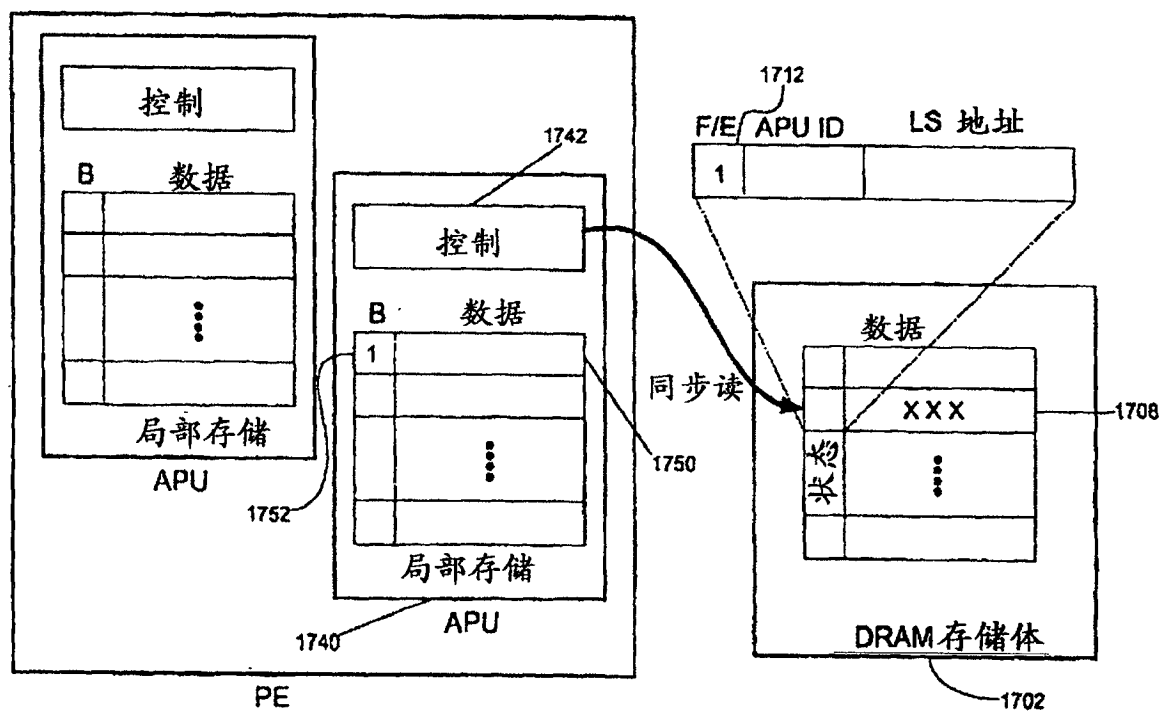


图 17E

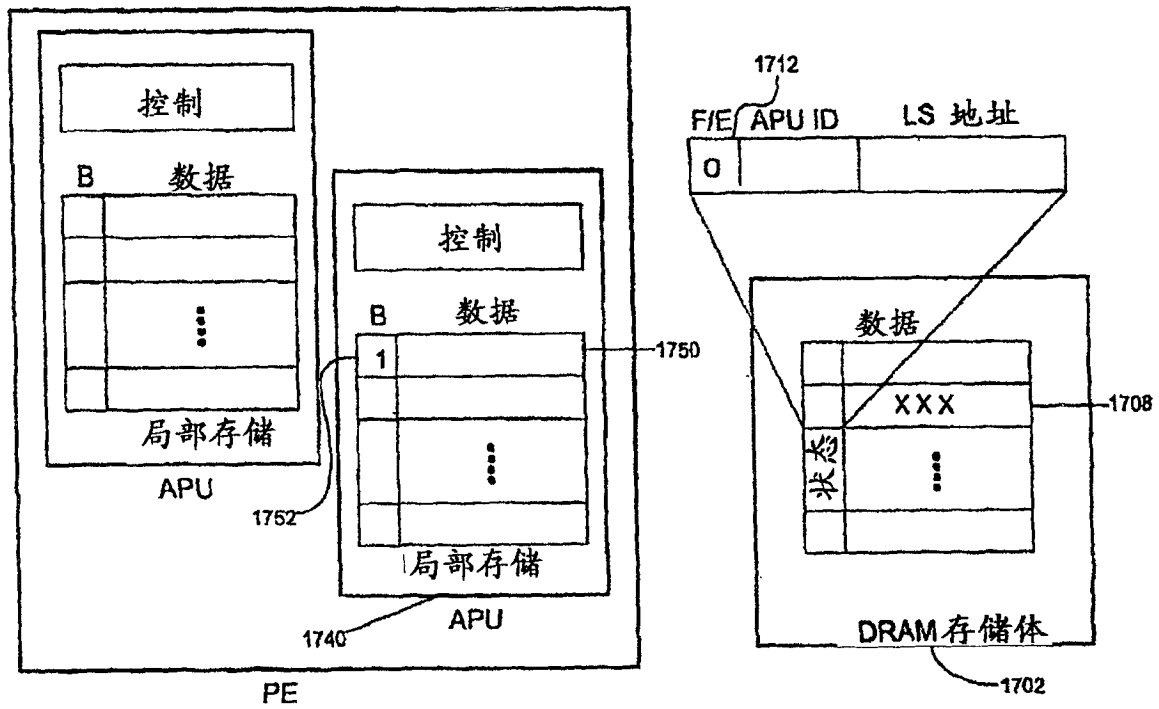


图 17F

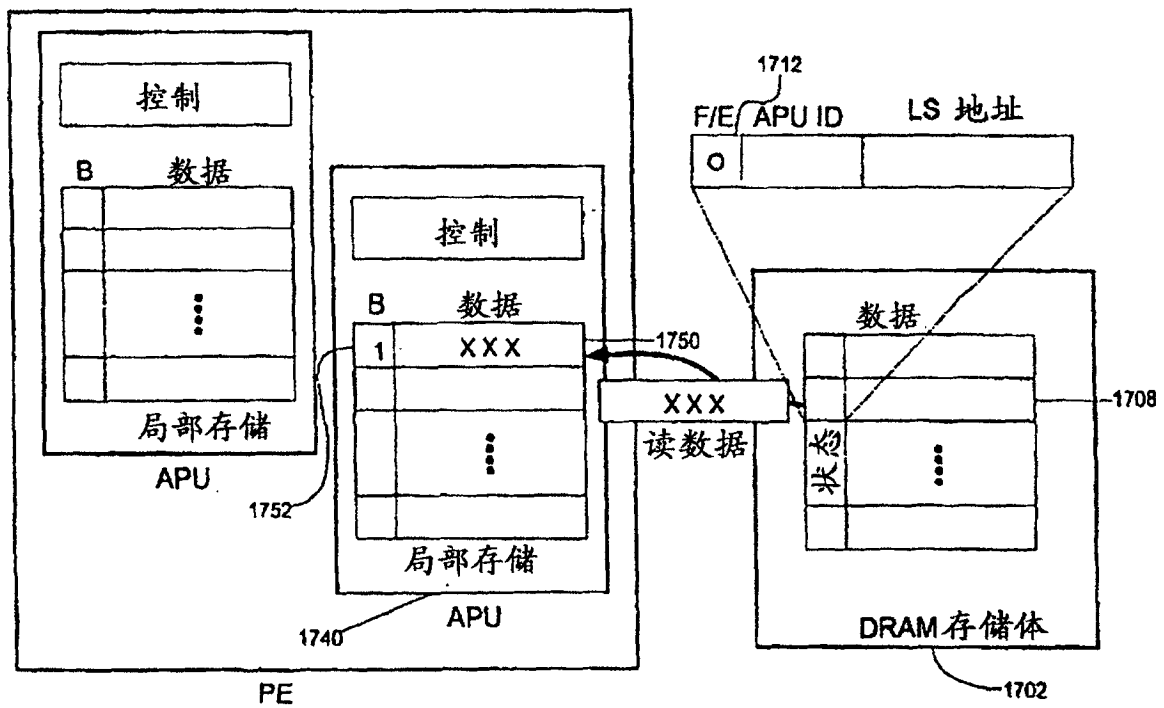


图 17G

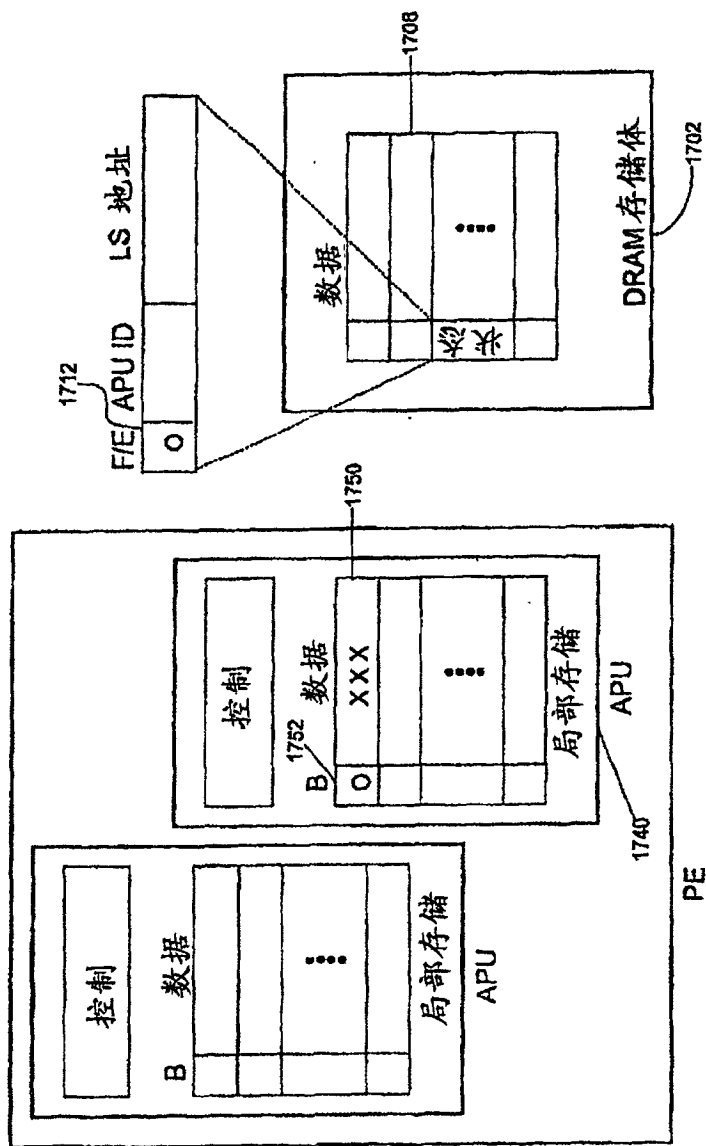


图 17H

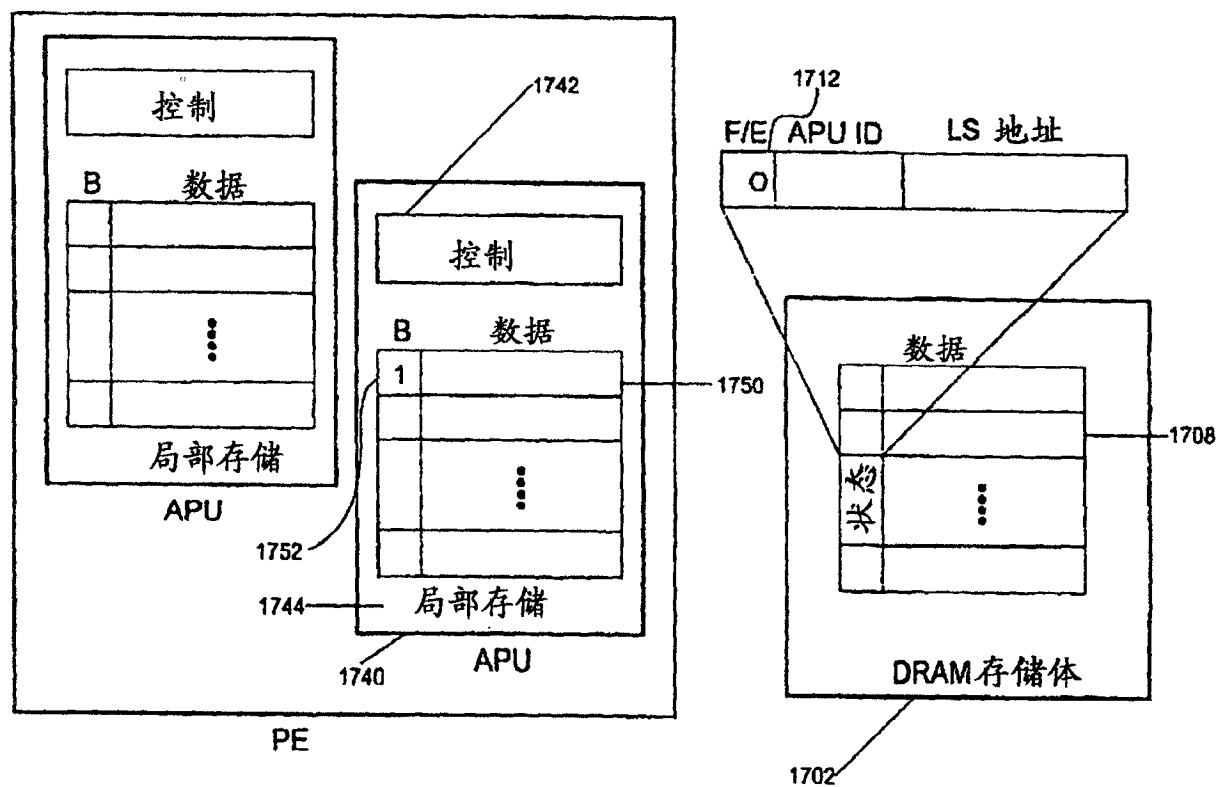


图 17I

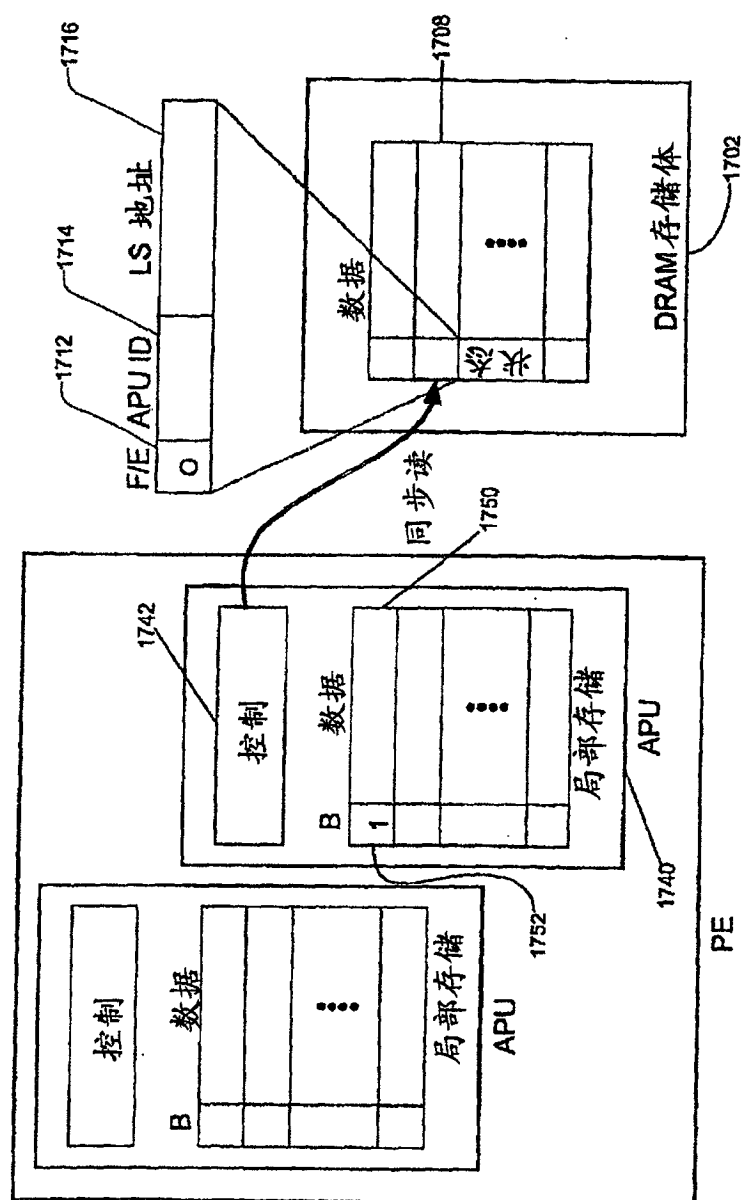


图 17J

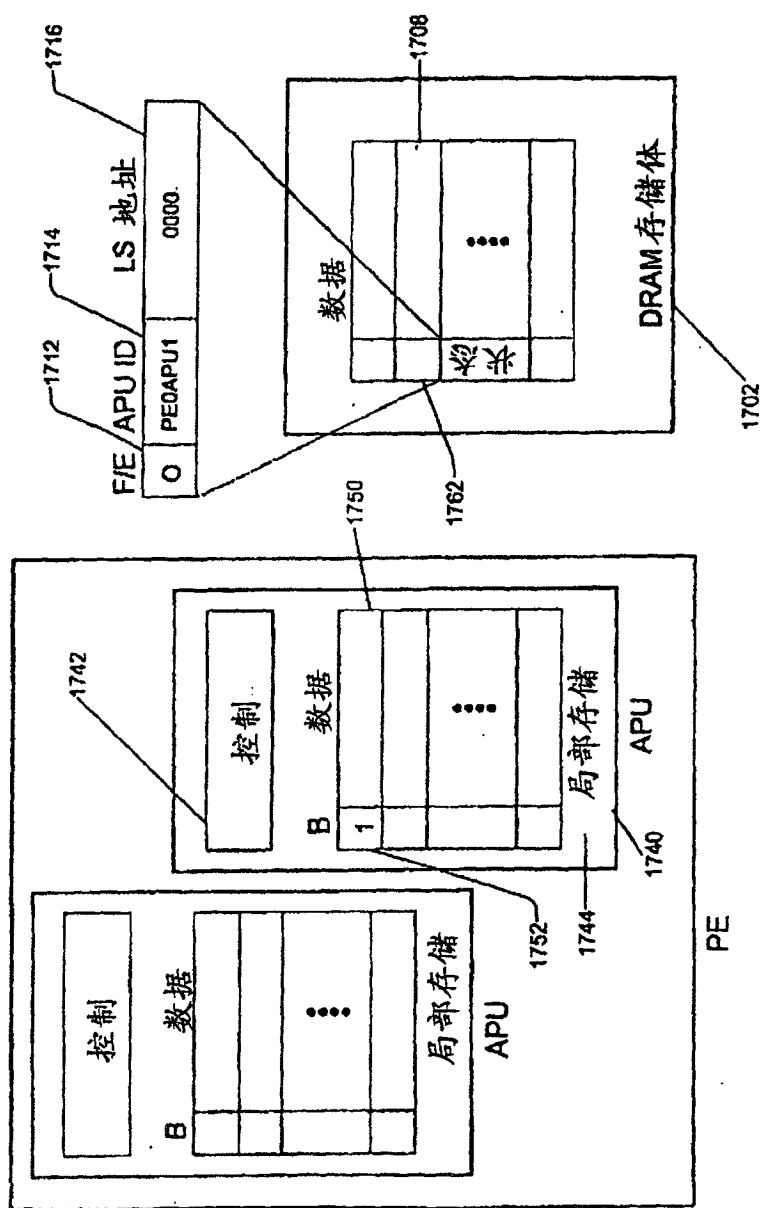


图 17K

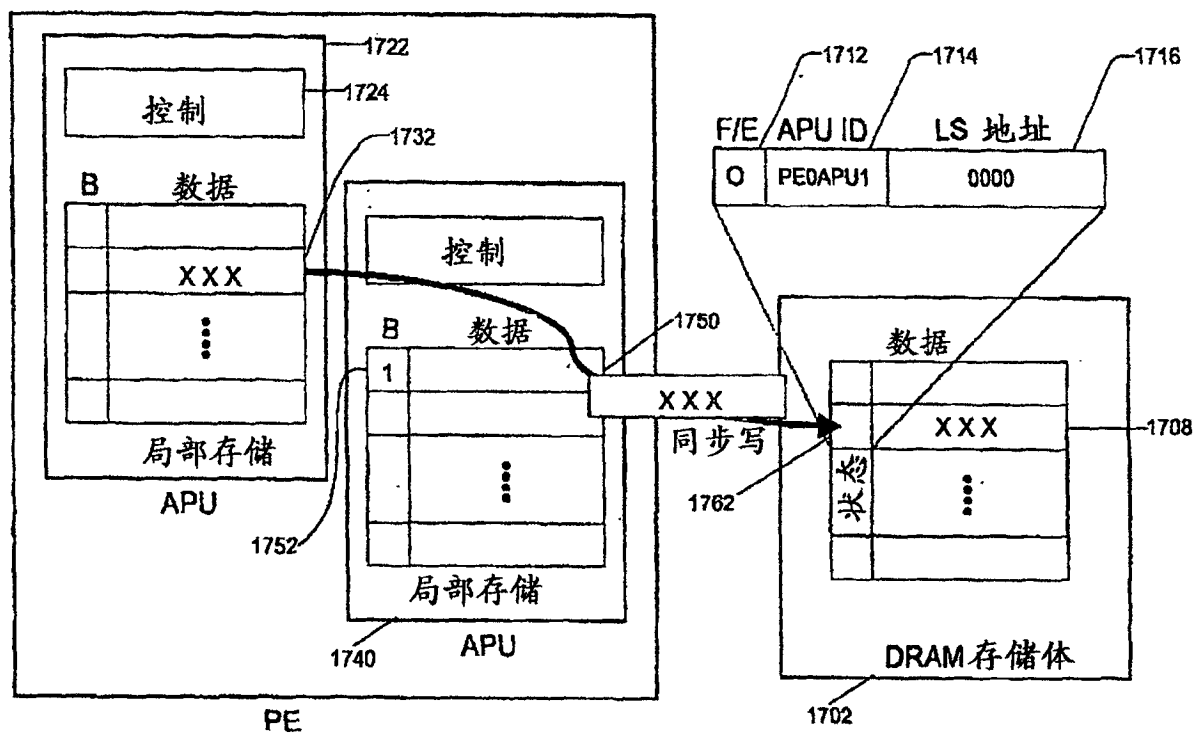


图 17L

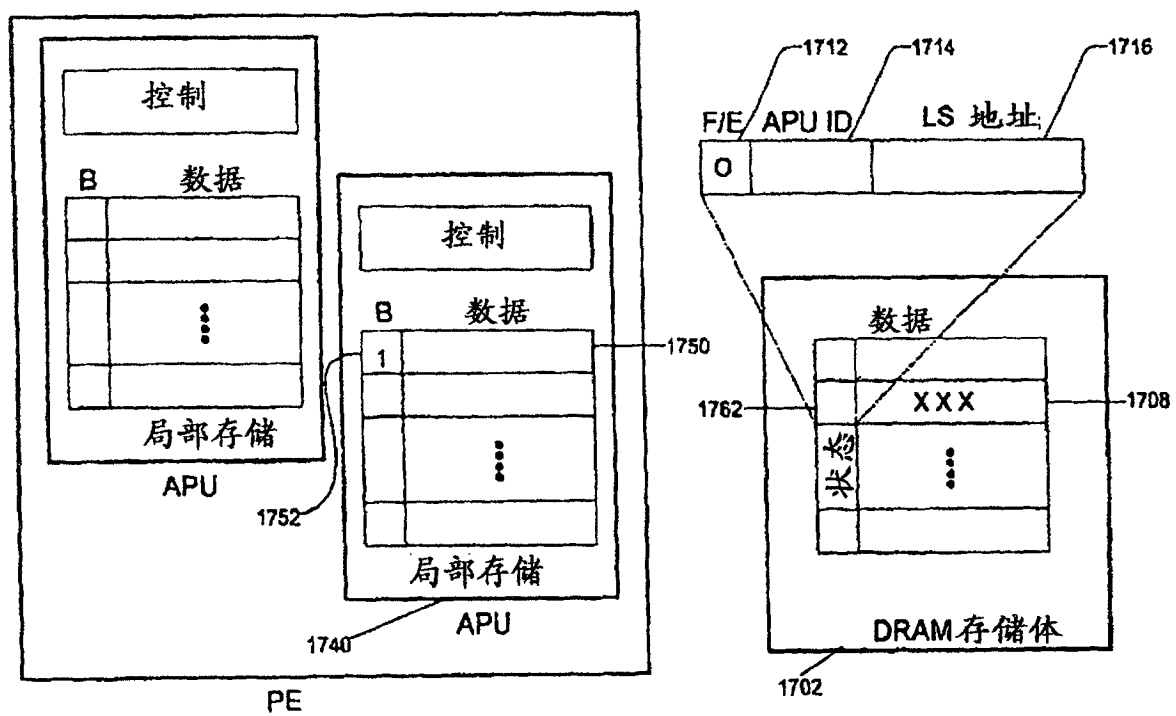


图 17M

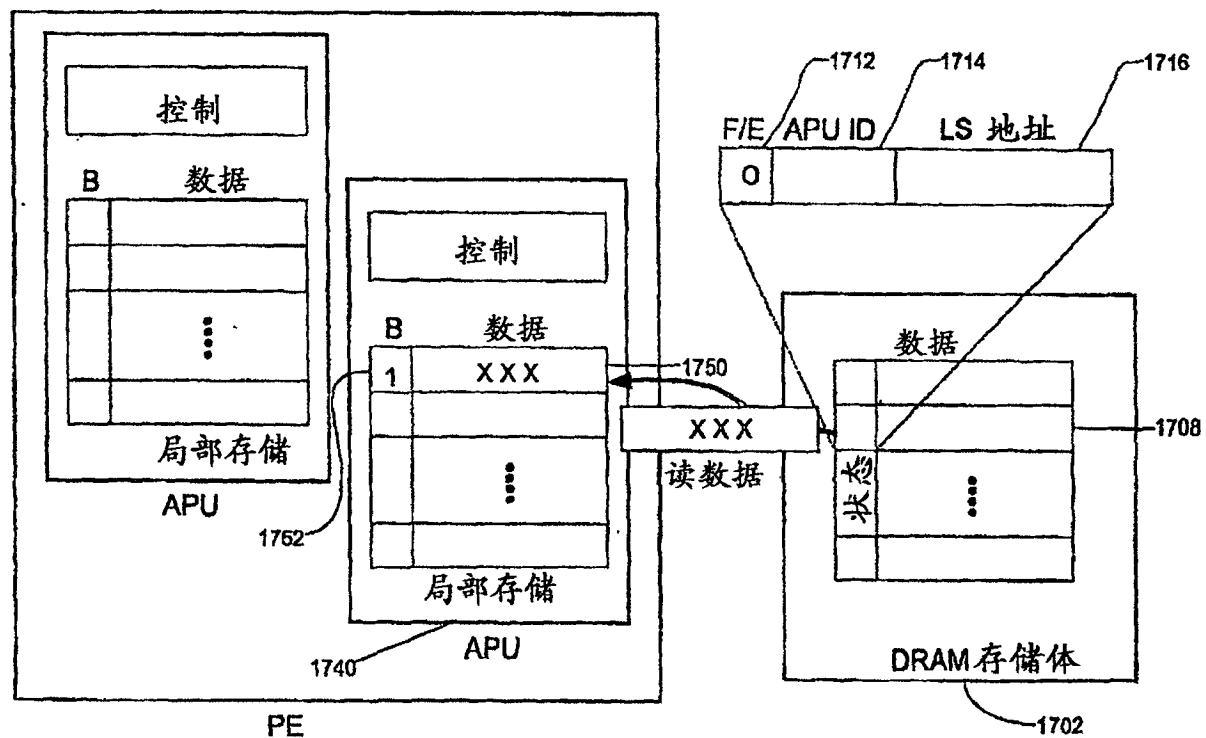


图 17N

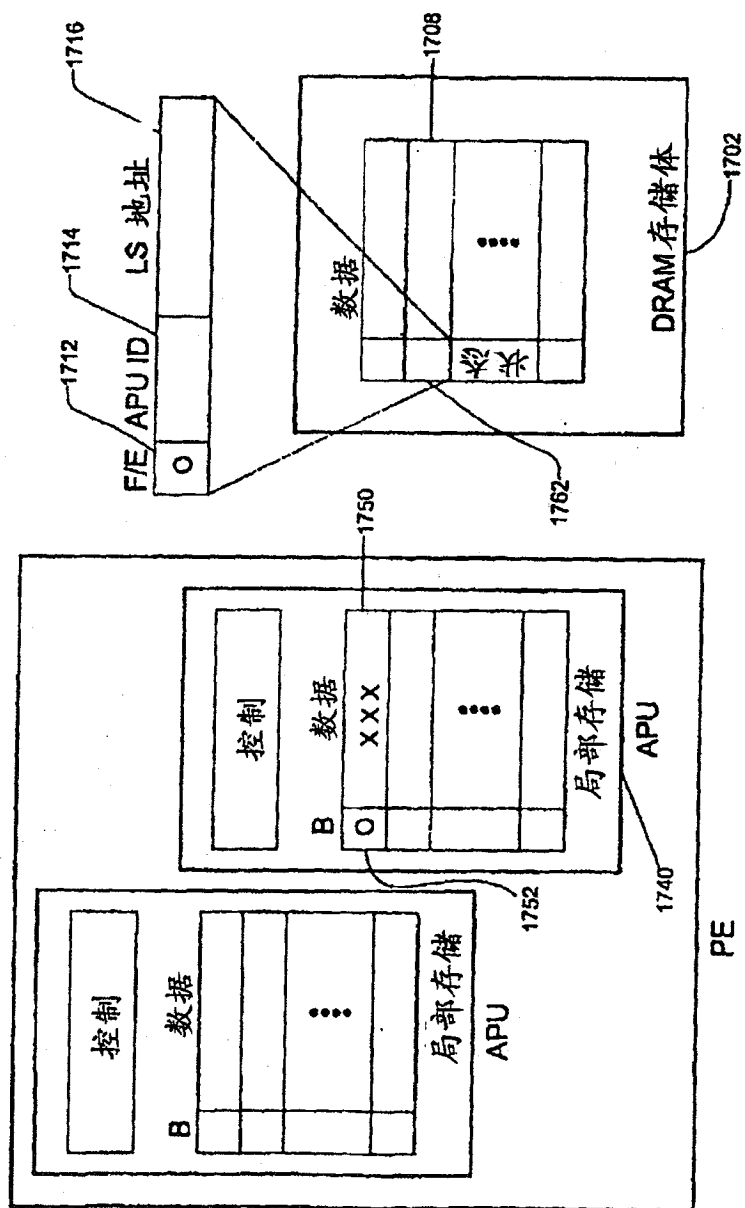


图 170

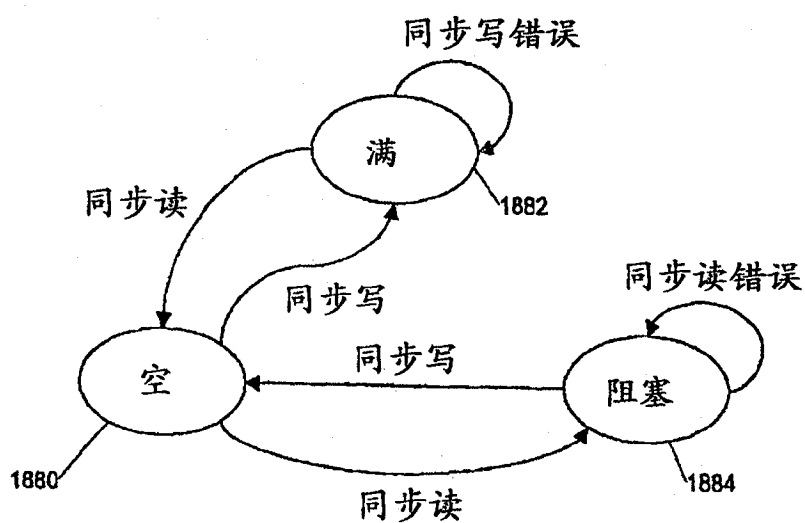


图 18

关键字控制表

1904 ID	1906	1908
0	APU关键字	关键字掩码
1	APU关键字	关键字掩码
2	APU关键字	关键字掩码
		⋮
7	APU关键字	关键字掩码

1902

图 19

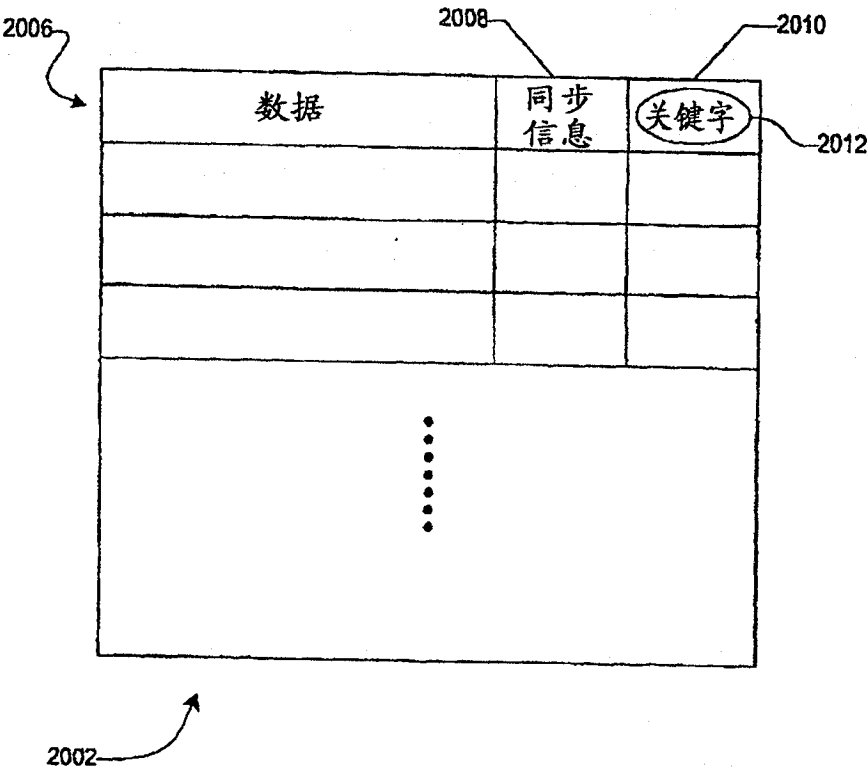


图 20

存储器存取控制表

ID	2106	2108	2110	2112
0	基	大小	存取关键字	存取关键字掩码
1	基	大小	存取关键字	存取关键字掩码
2	基	大小	存取关键字	存取关键字掩码
		⋮		
63	基	大小	存取关键字	存取关键字掩码

2104

2102

图 21

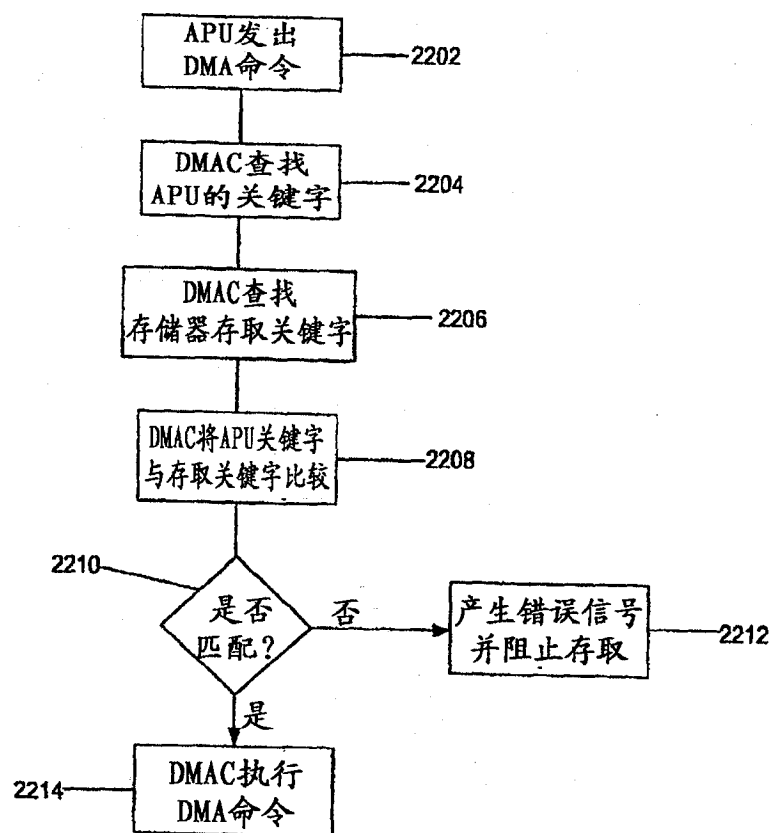


图 22

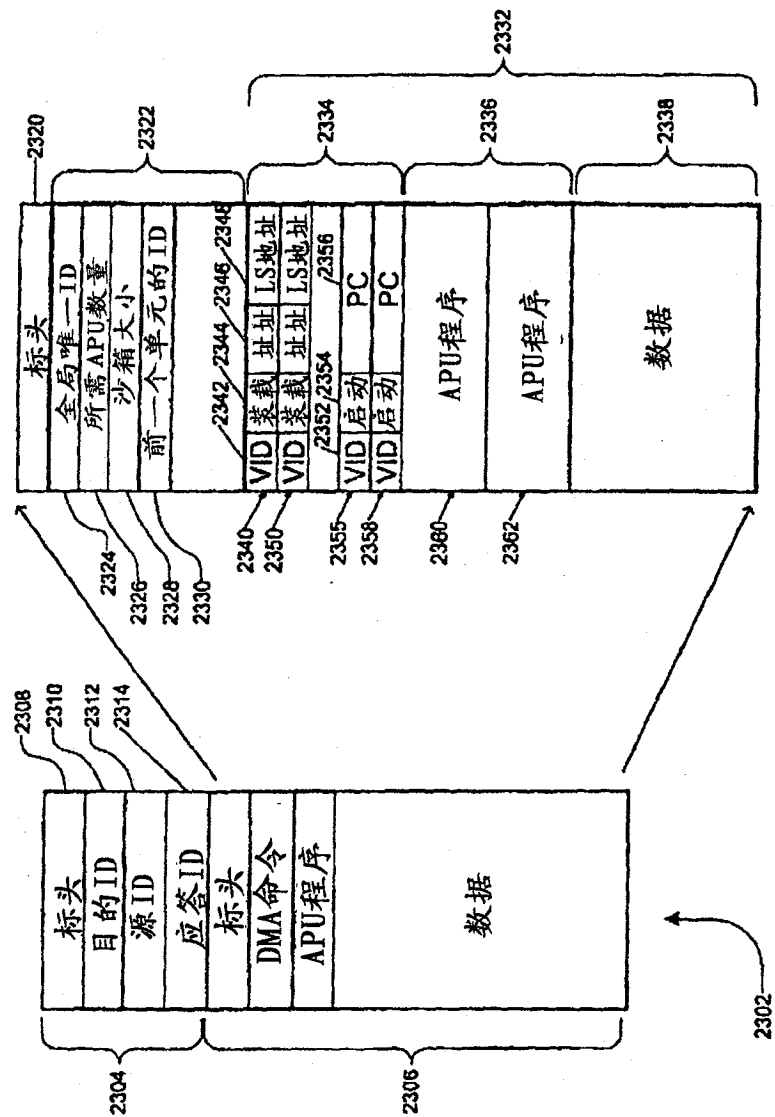


图 23

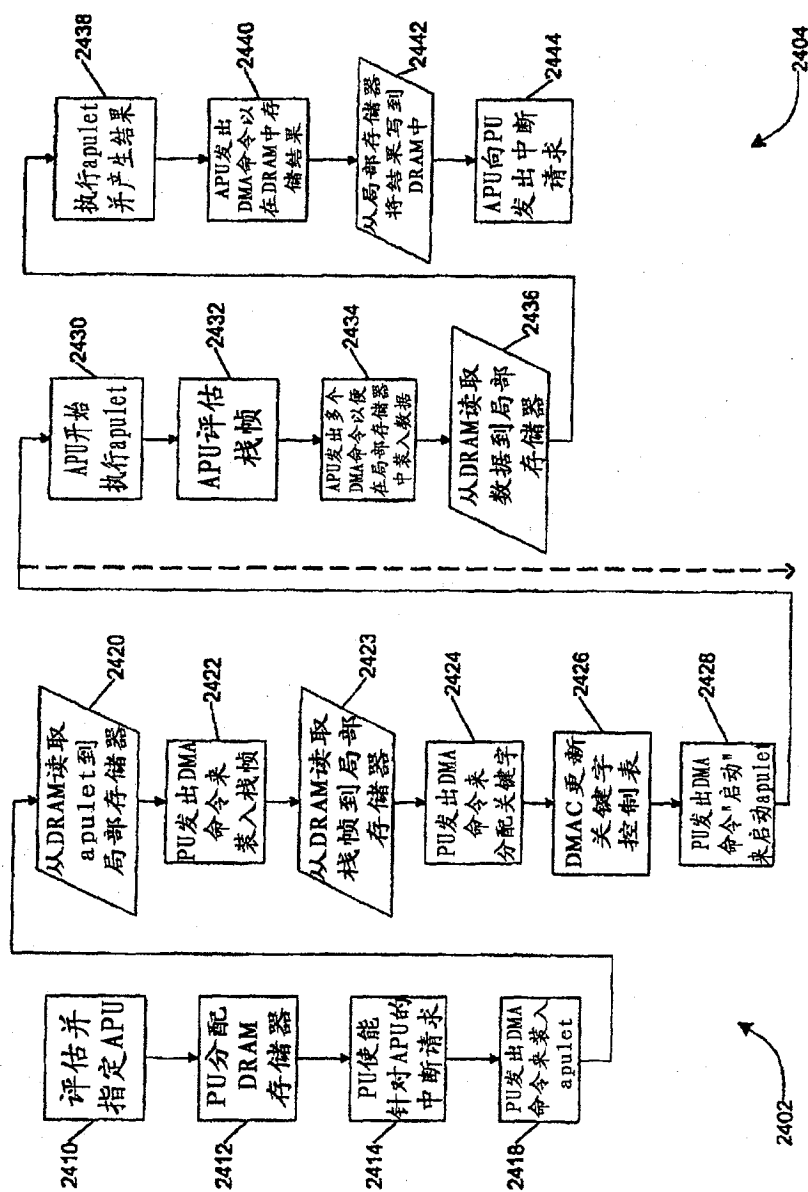


图 24

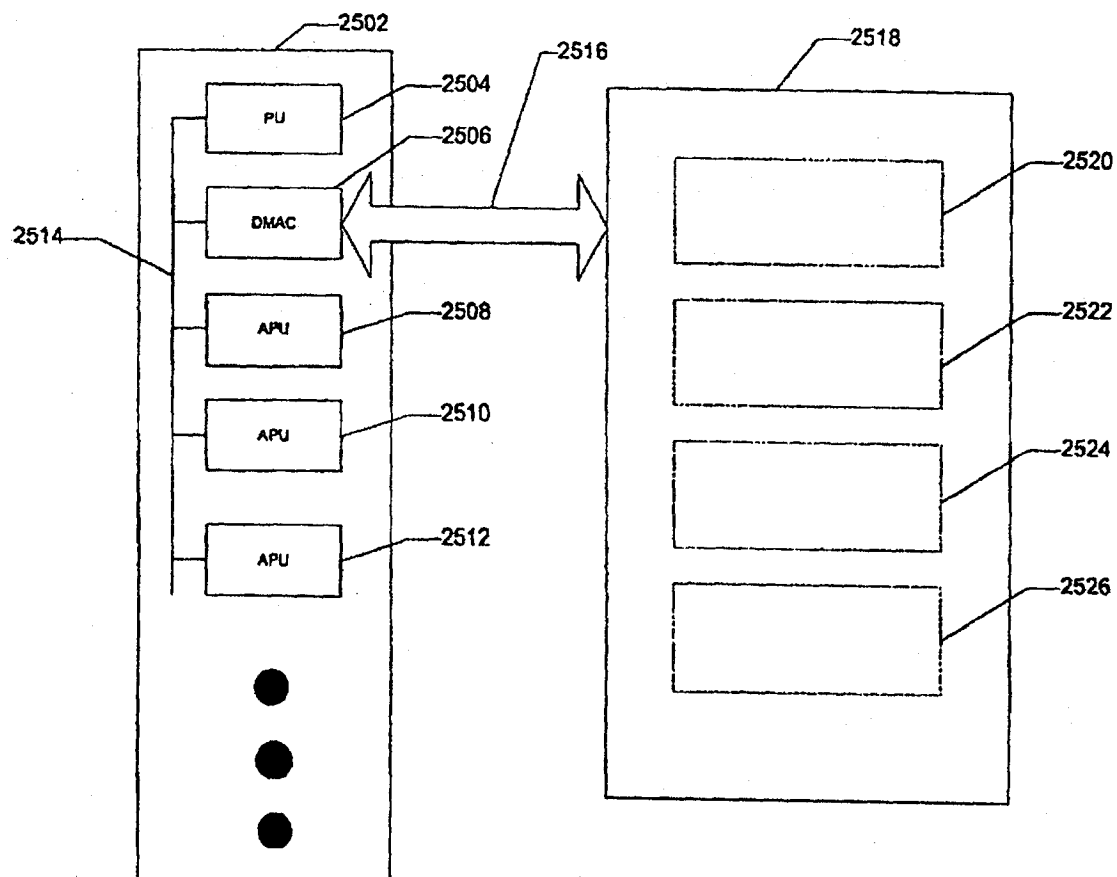


图 25

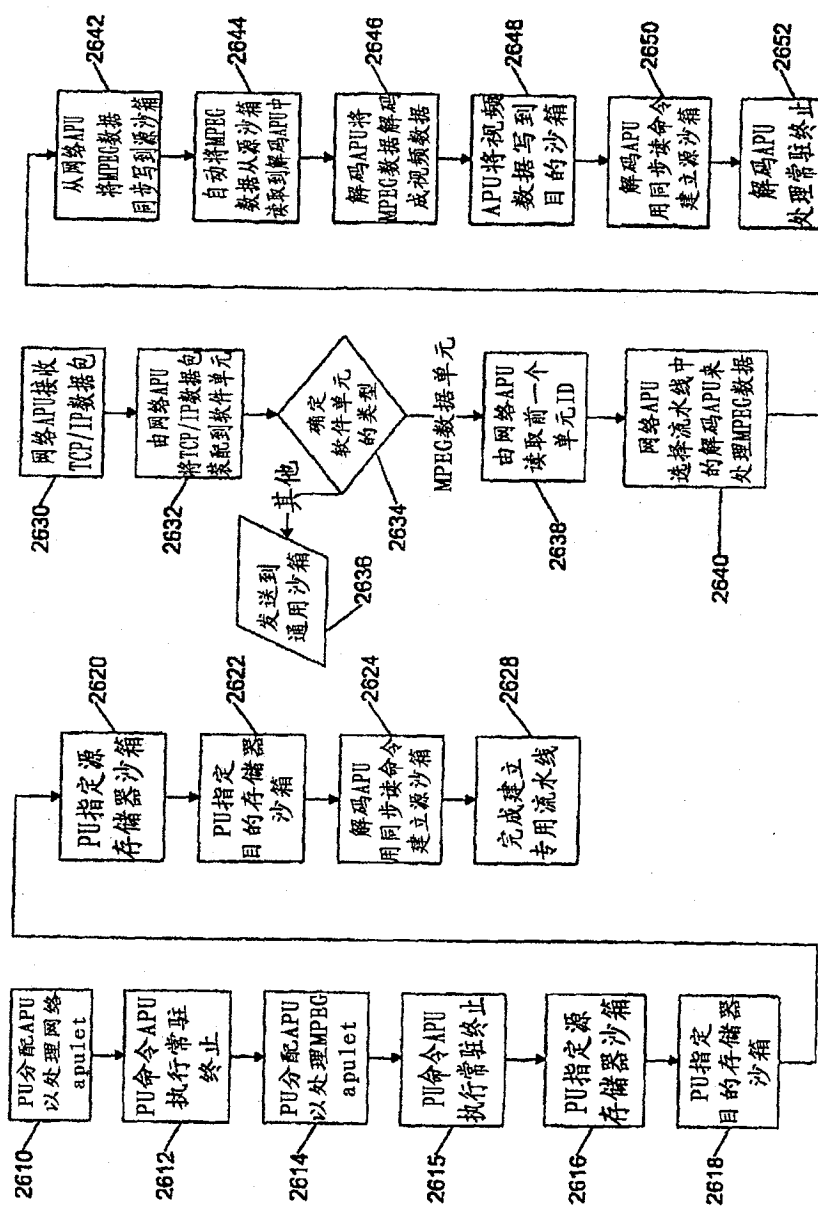


图 26B

图 26A

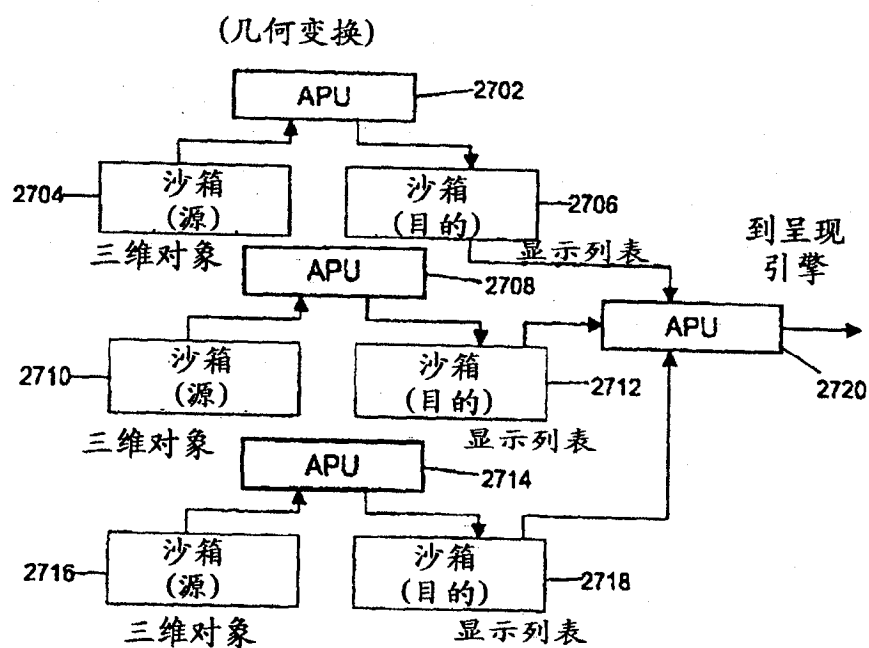


图 27

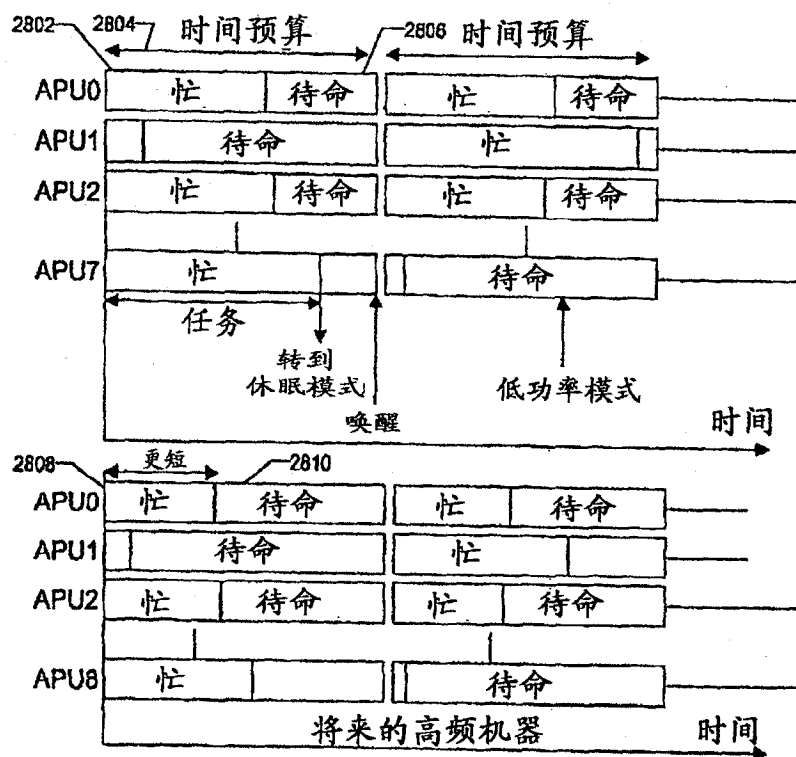


图 28

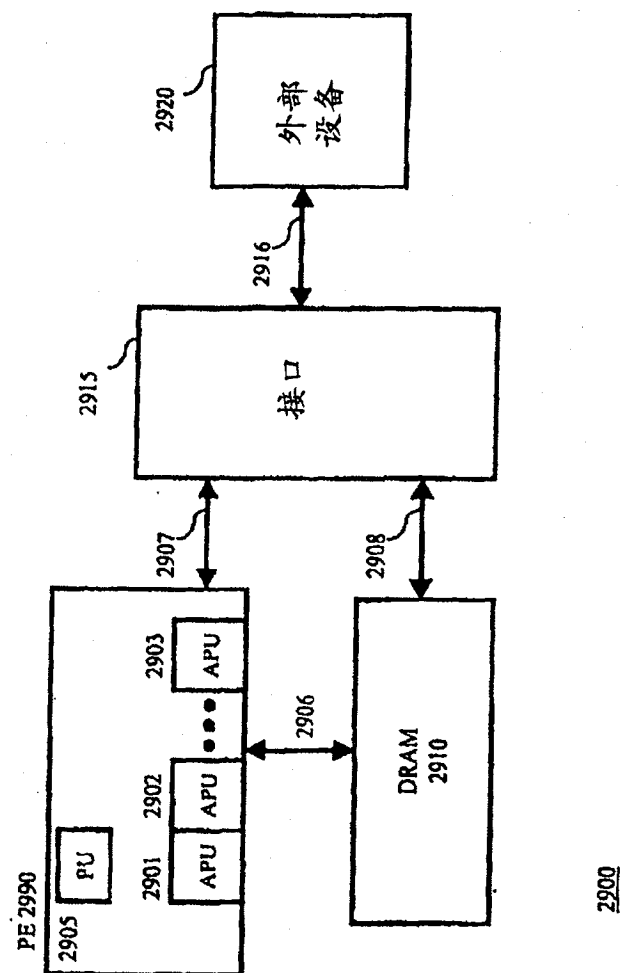


图 29

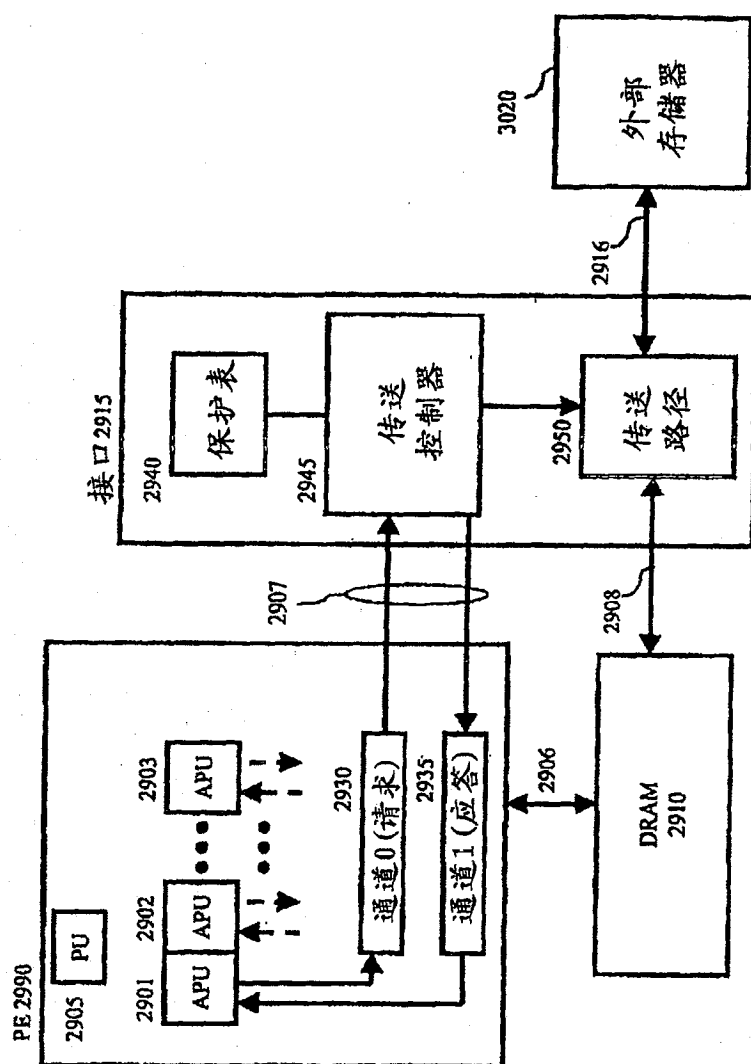


图 30

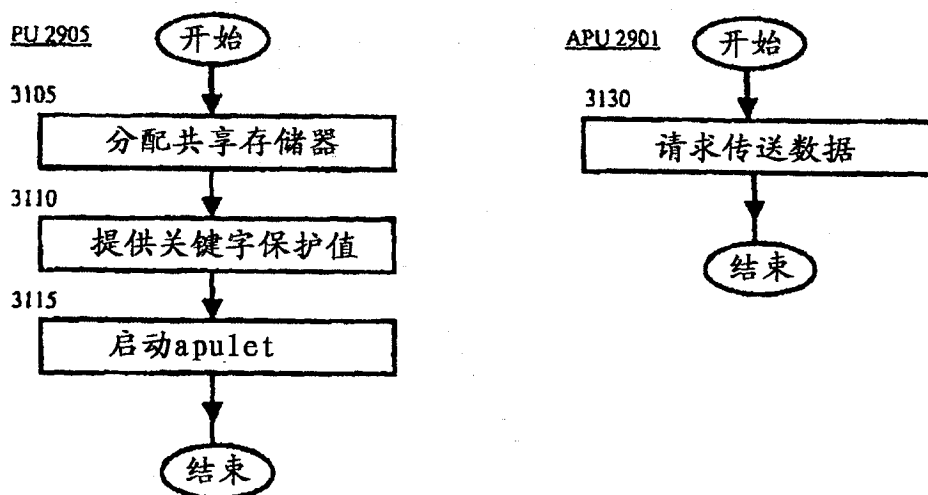


图 31

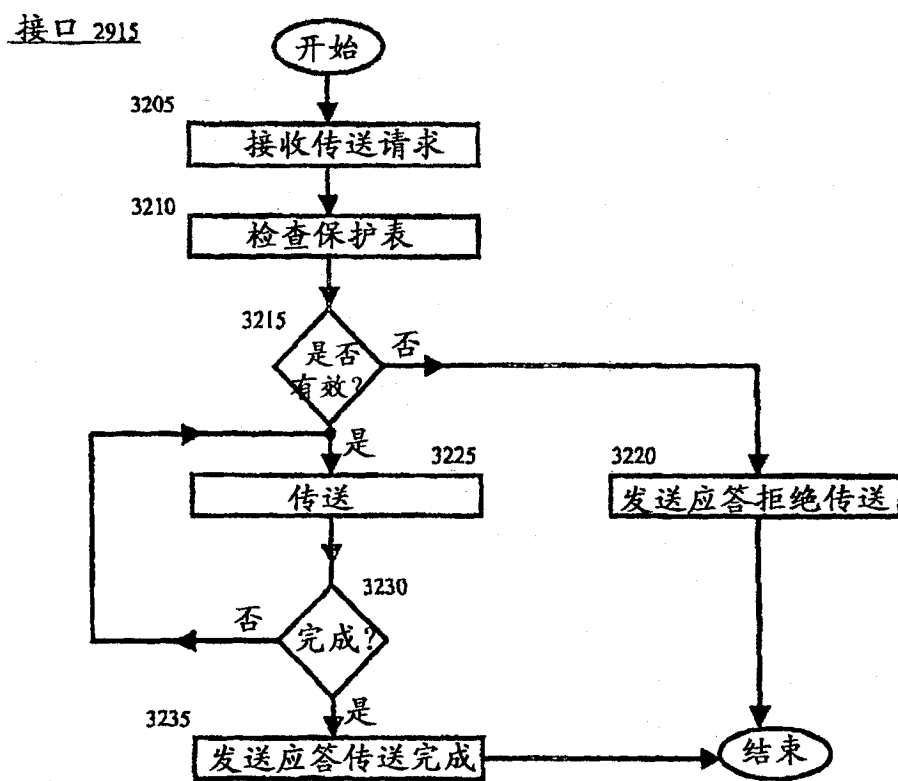


图 32

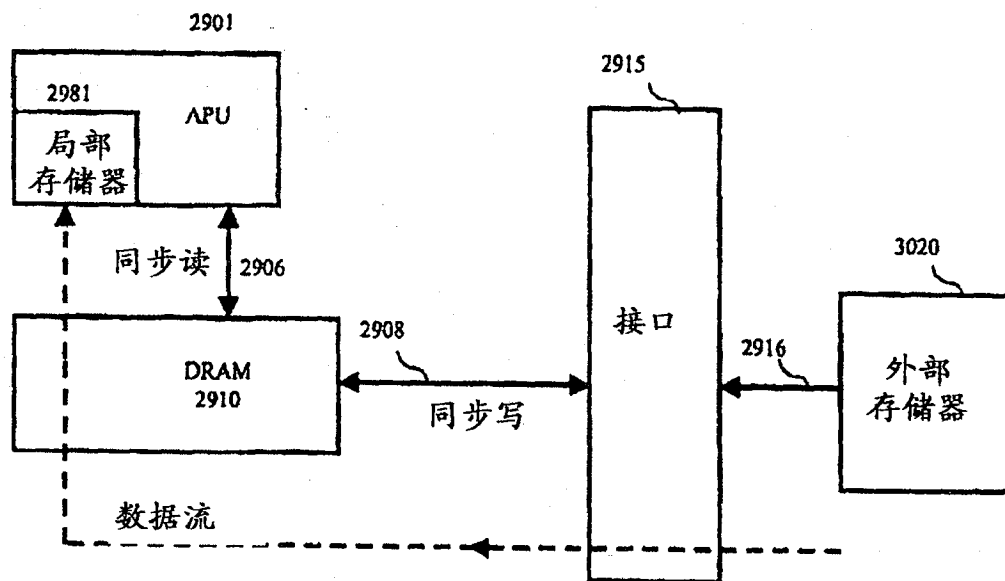


图 33

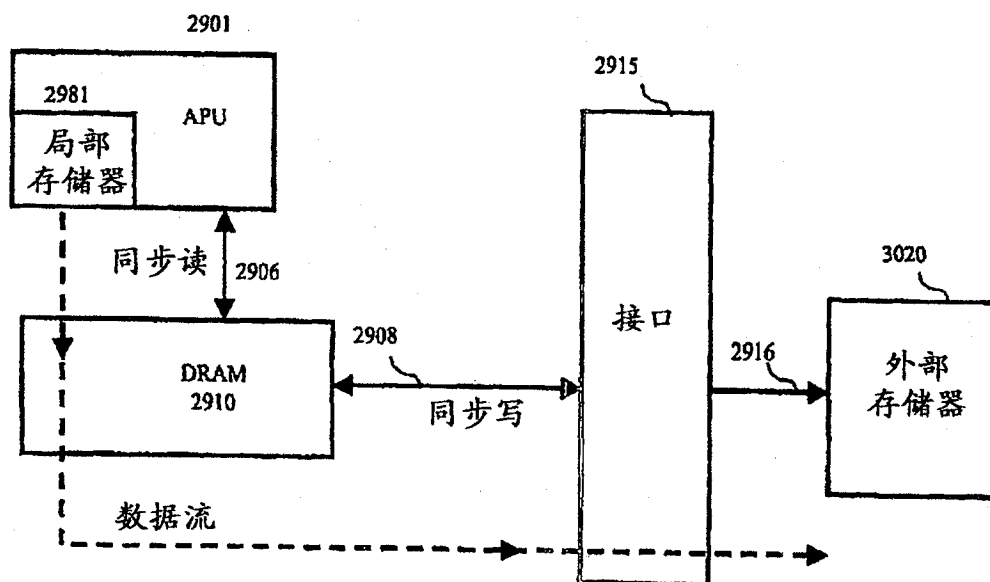


图 34

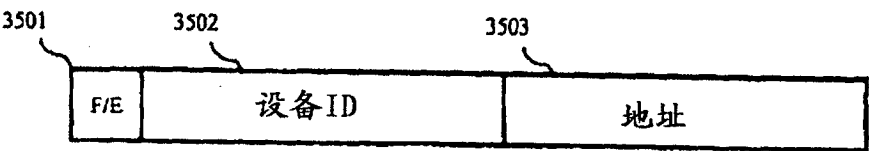


图 35

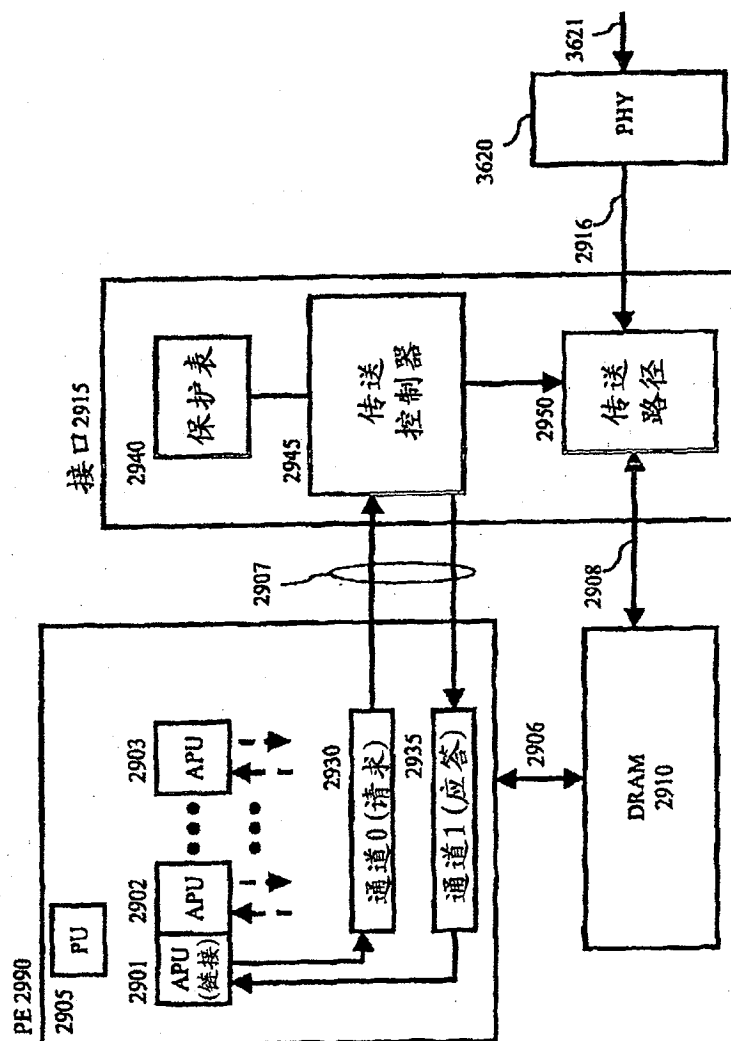


图 36

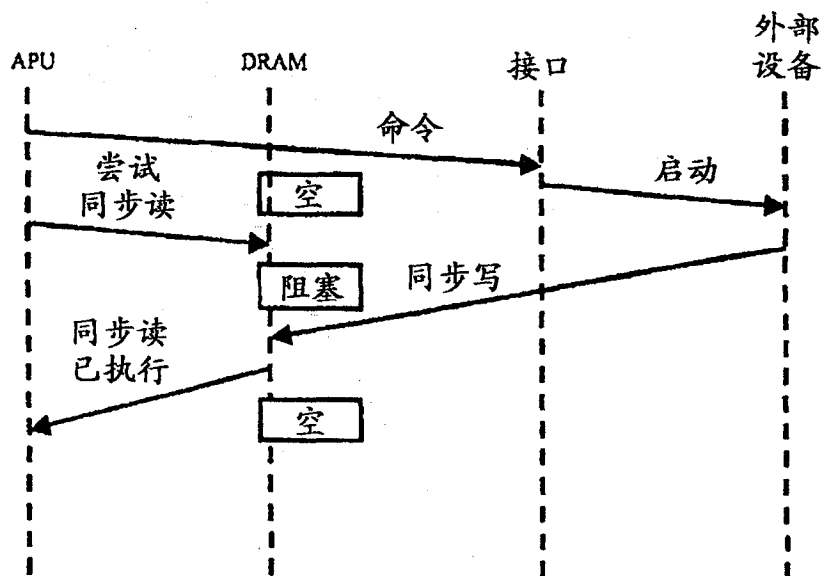


图 37

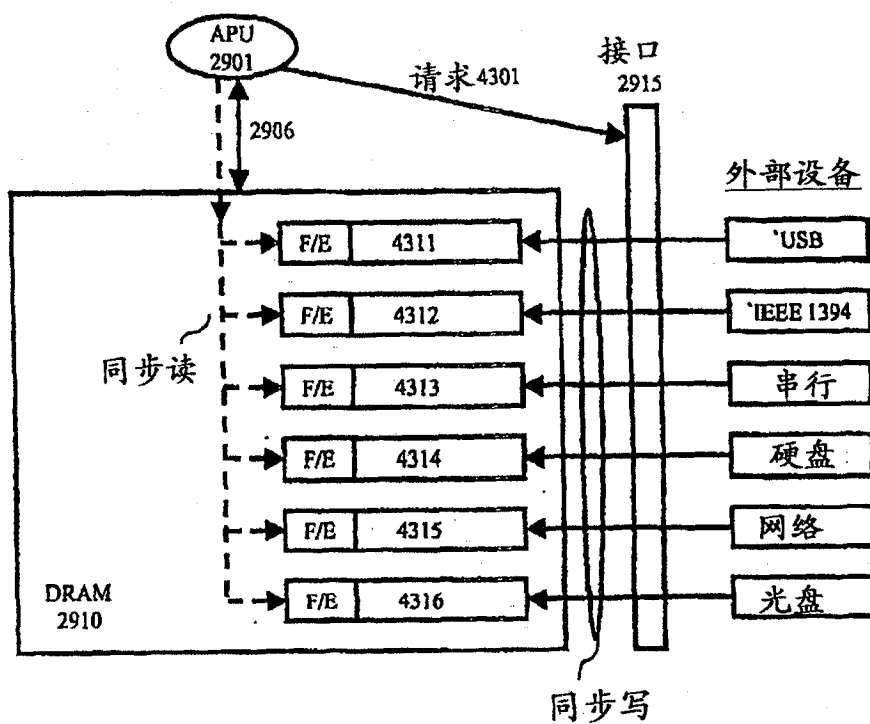


图 38

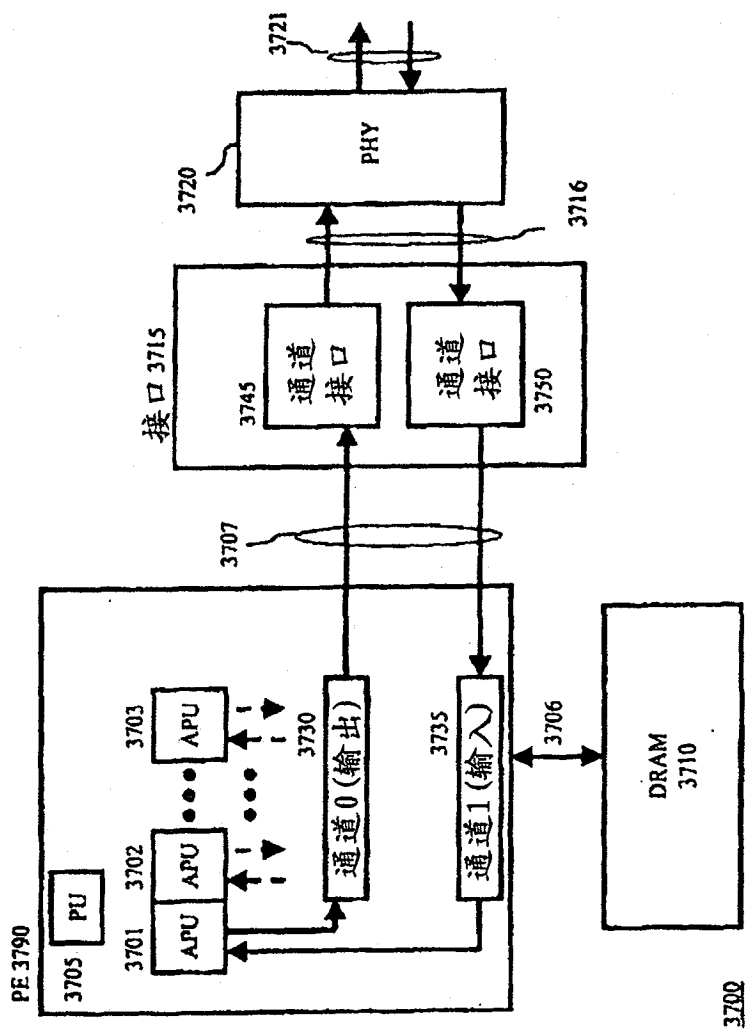


图 39

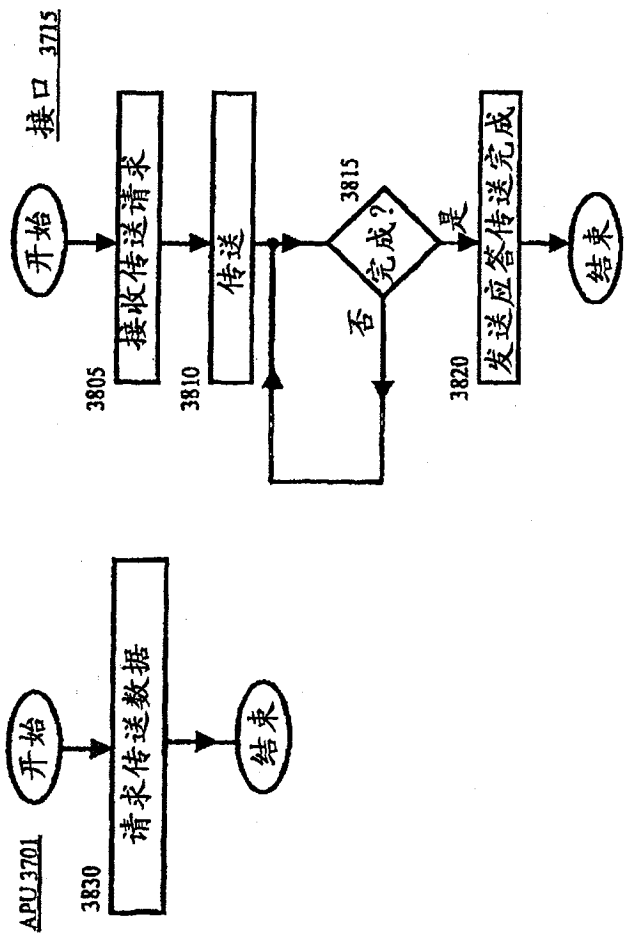


图 40

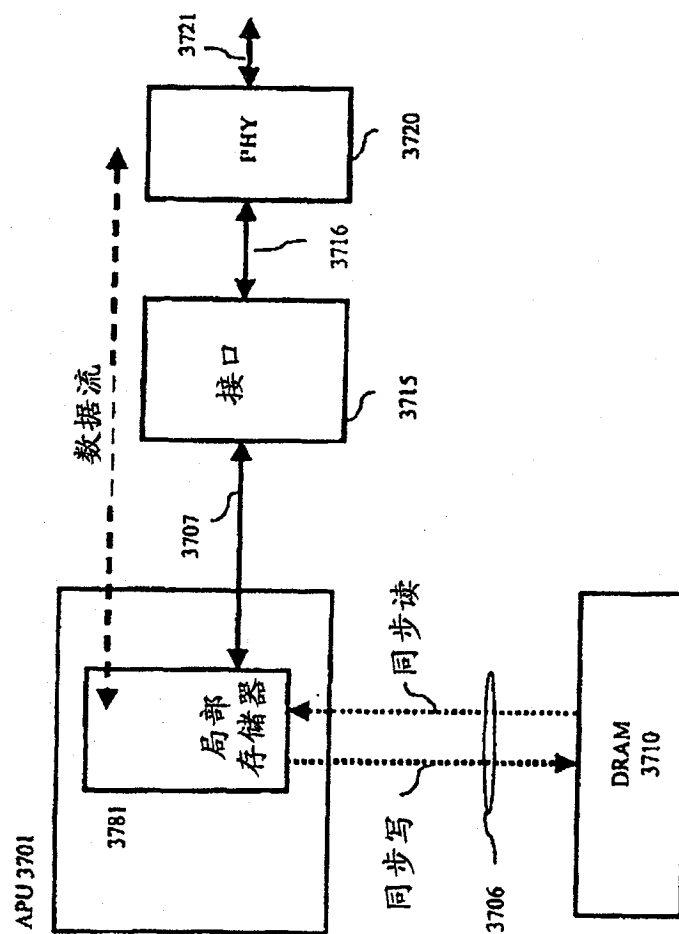


图 41

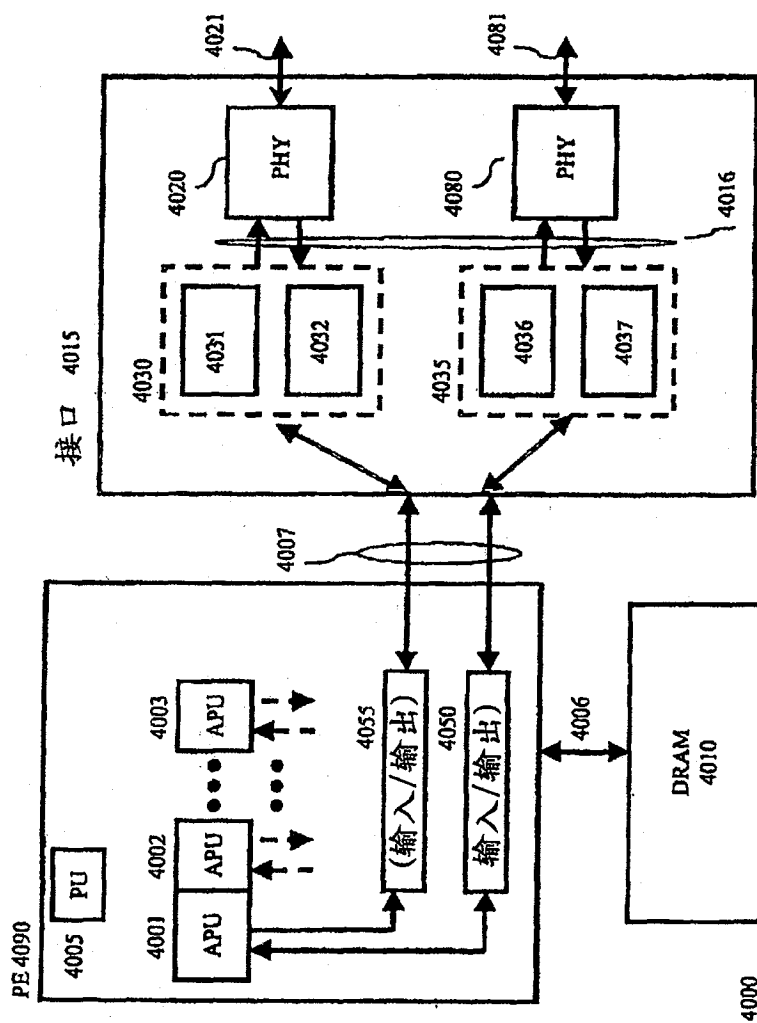


图 42