[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/38 (2006.01)
G06F 9/30 (2006.01)
H04L 12/56 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200710180338.X

[43] 公开日 2008 年 7 月 16 日

[11] 公开号 CN 101221493A

[22] 申请日 2000.8.17
[21] 申请号 200710180338.X
分案原申请号 00815246.2
[30] 优先权
[32] 1999.8.31 [33] US [31] 09/387,046
[71] 申请人 英特尔公司
地址 美国加利福尼亚州
[72] 发明人 D·伯恩斯坦因 D·F·胡珀
M·J·阿迪莱塔 G·沃尔里奇
W·维勒

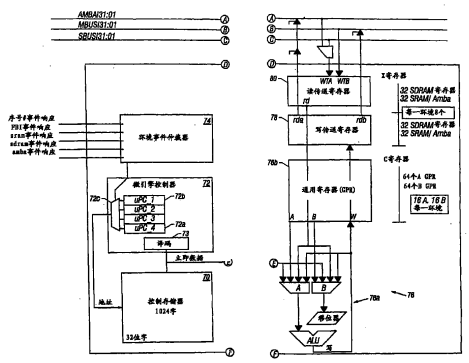
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 顾嘉运
权利要求书 3 页 说明书 23 页 附图 23 页

[54] 发明名称

并行处理器中的多线程执行

[57] 摘要

说明并行的硬件多线程处理器。该处理器包含一个谐调系统功能的通用处理器和支持多个硬件线程和多个微引擎。该处理器还包含具有第 1 存储控制器和第 2 存储控制器的存储器控制系统，第 1 存储控制器根据存储器访问是指向偶数存储组还是指向奇数存储组，将存储器访问加以分类，第 2 存储控制器则根据存储器访问是读访问还是写访问，对存储器访问进行优化。



1．一种处理引擎处所用的方法，该处理引擎处于一具有多个可编程多线程处理引擎的处理器内，其特征在于，所述方法包括：

执行具有第一程序计数器的第一线程的至少一个指令，所述至少一个指令包括向所述多个处理引擎所共用的资源发出一请求的至少一个指令；

在处理引擎执行所述至少一个指令向所述共用的资源发出所述请求之后，将执行对换至一具有第二程序计数器的第二线程；以及

在检测出响应对所述共用的资源的请求而生成的信号之后，将执行对换至所述第一线程。

2．如权利要求1所述的方法，其特征在于，还包括：选择一线程以便由所述处理引擎执行。

3．如权利要求2所述的方法，其特征在于，所述处理引擎的线程包括具有下列状态其中之一的线程：

当前正由所述引擎执行；

可用于执行、但当前未执行；以及

在处于可用于执行之前等待对一信号的检测，其中所述选择步骤包从可用于执行、但当前未执行的各线程当中选择一线程。

4．如权利要求3所述的方法，其特征在于，所述选择所述线程的步骤包括按循环在可用于执行、但当前未执行的各线程当中选择所述线程。

5．如权利要求2所述的方法，其特征在于，选择所述线程的步骤包括在检测出所述信号之后但将执行对换至所述第一线程之前选择一不同于所述第一线程的线程。

6．如权利要求1所述的方法，其特征在于，将执行对换的步骤包括选择一与所选定线程相关联的程序计数器。

7．如权利要求1所述的方法，其特征在于，还包括：在所述至少一个指令向所述共用的资源发出所述请求之后但将所述第一线程对换出去之前执行所述第一线程的另外指令。

8．如权利要求1所述的方法，其特征在于，还包括：

执行一所述第一线程明确请求线程对换的指令；以及

响应所述明确请求线程对换的指令，将执行对换至所述第二线程。

9．如权利要求8所述的方法，其特征在于，所述第一线程明确请求线程对换的所述指令，不包括一向共用的资源发出一请求的指令。

10．如权利要求1所述的方法，其特征在于，所述至少一个指令对所述信号进行识别。

11．如权利要求1所述的方法，其特征在于，所述信号包括一响应对所述请求提供的服务而生成的信号。

12．如权利要求1所述的方法，其特征在于，所述共用的资源包括下列其中之一：

一存储器，为处于所述处理器内部的所述多个处理引擎所共用；以及

一存储器，为处于所述处理器外部的所述多个处理引擎所共用。

13．如权利要求1所述的方法，其特征在于，还包括：

接收一分组；以及

用所述第一线程处理所述分组。

14．一种网络设备，其特征在于，包括：

至少一个以太网媒体存取控制器；

至少一个处理器，与所述至少一个以太网媒体存取控制器以通信方式连接，所述至少一个处理器包括：

多个可编程的多线程处理引擎，这些引擎中的单个引擎均包括一仲裁器来选择一线程加以执行；

至少一个存储器的至少一个接口，处于所述网络处理器的外部；以及

所述至少一个以太网媒体存取控制器的至少一个接口，其中所述至少一个存储器处于所述网络处理器的外部。

15．如权利要求14所述的设备，其特征在于，所述处理引擎中的各个处理引擎的各线程包括具有下列状态其中之一的诸多线程：

当前正由所述处理引擎执行；

可用于执行、但当前未由所述处理引擎执行；以及

在处于可用于执行之前等待对一与针对所述处理引擎所共用资源的请求相关联的信号的检测，其中一单个处理引擎的所述仲裁器从可用于执行、但当前未执行的各线程当中选择一线程。

16．如权利要求15所述的设备，其特征在于，所述仲裁器按循环在可用于执行的各线程当中选择。

17．如权利要求14所述的设备，其特征在于，各所述单个处理引擎用一与所述处理引擎的仲裁器所选定线程相关联的程序计数器。

18．如权利要求14所述的设备，其特征在于，各所述处理引擎以一包括至少一个明确请求当前线程对换的指令在内的指令集为特征。

19．一种处理器，其特征在于，包括：

多个可编程的多线程处理引擎，这些引擎中的单个引擎均包括一仲裁器来选择一线程加以执行；

一处于所述网络处理器内部的存储器，为多处理引擎所共用；

至少一个存储器的至少一个接口，处于所述网络处理器的外部；以及

至少一个媒体存取控制器的至少一个接口。

20．如权利要求 19 所述的处理器，其特征在于，

所述处理引擎中的各个处理引擎的各线程包括具有下列状态其中之一的诸多线程：

当前正由所述处理引擎执行；

可用于执行、但当前未由所述处理引擎执行；以及

在处于可用于执行之前等待对一与针对所述处理引擎所共用资源的请求相关联的信号的检测，其中所述处理引擎的所述仲裁器从可用于执行、但当前未执行的各线程当中选择一线程。

21．如权利要求 20 所述的处理器，其特征在于，所述仲裁器按循环在可用于执行的各线程当中选择。

22．如权利要求 19 所述的处理器，其特征在于，各所述处理引擎用一与所述处理引擎的仲裁器所选定线程相关联的程序计数器。

23．如权利要求 19 所述的处理器，其特征在于，各所述处理引擎以一包括至少一个明确请求线程对换的指令在内的指令集为特征。

并行处理器中的多线程执行

本申请是申请日:2000.08.17,申请号为00815246.2(国际申请号为PCT/US00/22650），名称为“在并行处理器中执行多线程”的申请的分案申请。

背景技术

本发明涉及一种并行处理器。

并行处理是在计算处理中进行并发事件信息处理的有效形式。并行处理需要在计算机同时执行许多程序，与串行处理不同。在并行处理器中的环境中，并行处理涉及同时做多件事。与在一个站串行执行全部任务的串行范例或在专门的站执行任务的流不机不同，并行处理配备多个站，每个站能执行全部任务。也就是说，全部的站或多个站对问题的相同或共同要素一般同时且独立地工作。有些问题适合用并行处理解决。

因此，计算机处理类型包括单指令流单数流型，串行的冯.诺依曼型计算机中存在单一指令流。第2种处理类型是单指令流多数据流处理（SIMD）。这种处理能有多个算术逻辑处理器和一个控制处理器。每一处理器在锁步执行数据操作。这些处理器由控制处理器进行同步。第3种类型是多指令流单数据流（MISD）处理。这种处理具有流经执行不同指令流的线性处理器阵的相同数据流。第4种是多指令流多数据流（MIMD）处理。这种处理采用多个单元器，每一处理器执行自己的指令流，以处理供给各处理器的数据流。多指令流多数据流（MIMD）处理器可具有若干指令处理单元，因而有若干数据流。

发明内容

根据本发明的一个方面，微控制功能执行单元包含存放微程序的控制存储器和保持多个微程序计算器的微引擎控制器。该单元还包含对指令进行译码的译码逻辑和一个现境事件仲裁器，该仲裁器响应外部标志，判断微控制功能执行单元中可执行的多个线程中的哪一个升级到执行状态。

本发明的一个或多个方面可提供以下一个或多个优点。

微引擎能处理多个硬件线程。每一微引擎保持多个硬件形式的程序计数器

和该程序计数器相关联的状态。实际上，相应的多个线程组可在一微引擎上同时工作，但任何时仅有一个线程真正运作。硬件环境对换使任务完成同步。举例来说，2 个线程可试图存取相同共用资源。资源完成一个微引擎线程环境所请求的任务时，该资源便回报一通知操作完成的标志。微引擎收到该标志时，便可判定开通哪一线程。

硬件多线程处理可用于使微引擎上的第 1 线程启动例如存取存储器这种事务。存储器存取期间，若微引擎例如仅具有一个可运作的线程，该微引擎就会休眠直到从存储器返回数据为止。通过在微引擎内利用硬件环境对换，该硬件环境对换使具有独行程序计数器的其他环境能在相同微引擎中执行。因此，第 1 线程等待存储器送回读出数据的同时，可执行另一线程。这些特征可扩展到微引擎中同时工作的多个线程，以便在数据通路中处理更多工作。

附图简要说明

图 1 是采用基于硬件的多线程处理器的通信系统的框图。

图 2 是图 1 中基于硬件的多线程处理器的具体框图。

图 3 是图 1 和图 2 中基于硬件的多线程处理器所用的微引擎功能单元的框图。

图 3－3 是图 3 中微引擎流水线的框图。

图 3－4 是表示环境切换指令格式的示意图。

图 3－5 是表示通用寄存器地址安排的框图。

图 4 是基于硬件的多线程处理器中所用的强化带宽操作用存储控制器的框图。

图 4－3 是表示图 4 中 SDRAM 控制器的仲裁策略的流程图。

图 4－4 是说明对 SDRAM 控制器进行优化的好处的时序图。

图 5 是基于硬件的多线程处理器中所用的等待时间有限的操作用存储控制器的框图。

图 5－3 是表示对 SRAM 控制器进行优化的好处的时序图。

图 6 是图 1 中处理器的通信总线接口的框图。

说明

体系结构

参照图 1，通信系统 10 包括一并行、基于硬件的多线程处理器 12。该基于硬件的多线程处理器 12 与诸如 PCI 总线 14 这种总线、存储器系统 16 和第二总线 18 连接。系统 10 对可分解为并行子任务或功能的任务尤其有用。具体来说，硬件多线程处理器 12 对那些面向带宽的任务而非面向等待时间的任务有用。硬件多线程处理器 12 具有多重微引擎 22，分别配备可对一任务同时作用和独立工作的多重硬件控制的线程。

硬件多线程处理器 12 还包括一中央控制器 20，该控制器有助于对硬件多线程处理器 12 的其他资源加载微码控制，并执行其他通用计算机类型功能，诸如处理协议、异常以及微引擎在边界状态等条件下传出数据分组进行更为细节处理场合对分组处理的额外支持。一实施例中，处理器 20 是一基于 Strong Arm® (Arm 是英国 Arm 有限公司的商标)的体系结构。通用微处理器 20 具有一操作系统。通过该操作系统，处理器 20 可调用功能在微引擎 22a～22f 上操作。处理器 20 可利用任何得到支持的操作系统，最好用实时操作系统。对按 Strong Arm 结构实现的核心处理器来说，可用诸如微软 NT 实时、VXWorks 和 μCUS 这种操作系统和可从互联网得到的免费件操作系统。

硬件多线程处理器 12 还包含多个功能微引擎 22a～22f。这些功能微引擎（微引擎）22a～22f 分别在硬件及其关联状态方面保持多个程序计数器。实际上，各微引擎 22a～22f 中可同时使相应的多组线程工作，而任何时候仅一个真正操作。

一实施例中，存在 6 个微引擎 22a～22f，如图所示，每一微引擎 22a～22f 能处理 4 个硬件线程。6 个微引擎 22a～22f 以包括存储器系统 16 以及总线接口 24 和 28 在内的共用资源进行工作。存储器系统 16 包含同步动态随机存取存储器（SDRAM）的控制器 26a 和静态随机存取存储器（SRAM）的控制器 26b。SDRAM 存储器 16a 和 SDRAM 控制器 26a 通常用于处理大量数据，例如处理来自网络数据分组的网络有效负载。SRAM 控制器 26b 和 SRAM 存储器 16b 在网络实施例当中实现较小等待时间、迅速存取任务，例如存取查找表、核心处理器 20 的存储器等。

6 个微引擎 22a～22f 根据数据特性存取 SDRAM16a 或 SRAM16b。因此，对 SRAM 存入并读取短等待时间且小带宽的数据，而对 SDRAM 则存入并读取等待时间不重要的大带宽数据。微引擎 22a～22f 可对 SDRAM 控制器 26a 或 SRAM 控制器 26b 执行存储器指针指令。

可通过 SRAM 或 SDRAM 的存储器存取说明硬件多线程的优点。举例来说，一线程 0 所请求的微引擎对 SRAM 的存取会使 SRAM 控制器 26b 启动对 SRAM 存储器 16b 的存取。SRAM 控制器控制对 SRAM 总线的仲裁，对 SRAM16 进行存取，从 SRAM16b 读取数据，并使数据返回至提出请求的微引擎 22a-22b。SRAM 存取期间，若微引擎（例如 22a）仅具有一个可运作的线程，该微引擎会休眠直到从 SRAM 返回数据。通过采用每一微引擎 22a～22f 内的硬件环境对换，该硬件环境对换能使具有独特程序计数器的其他环境在该相同微引擎中执行。因此，第一线程（例如线程 0）等待所读出数据返回的同时，另一线程例如线程 1 可起作用。执行期间，线程 1 可存取 SDRAM 存储器 16a。线程 1 在 SDRAM 单元上运作，线程 0 在 SRAM 单元上运作的同时，新线程例如线程 2 可在微引擎 22a 中现场运作。线程 2 可运作一些时间直到该线程需要存取存储器或执行某些其他长等待时间操作，诸如对总线接口进行存取。因此，处理器 12 可同时使总线运作、SRAM 运作和 SDRAM 运作均得到完成，或由一个微引擎 22a 在其上运作，全部完成或正在工作的总线操作、SRAM 操作和 SDRAM 操作，并且使另一个线程可用于处理数据通路中更多的工作。

硬件环境对换还使任务完成同步。举例来说，2 个线程会同时命中相同的共用资源例如 SRAM。诸如 FBUS（F 总线）接口 28、SRAM 控制器 26a 和 SDRAM26b 等每一个独立功能单元当其完成其中一个微引擎线程环境所请求的任务时，便回报一通知完成运作的标志。微引擎收到该标志，便能判断开通哪一线程。

利用硬件多线程处理器的一个例子是用作网络处理器。作为网络处理器，硬件多线程处理器 12 作为与诸如媒体存取控制器（例如 10/100BaseT8 进制 MAC 13a 或千兆位以太网 13b）这种网络装置的接口。通常作为一网络处理器，硬件多线程处理器 12 可作为与任何类型通信设备的接口或收发大量数据的接口。在联网应用中起作用的通信系统 10 可从设备 13a 和 13b 接收多个网络数据分组，以并行方式处理这些数据分组。可采用硬件多线程处理器 12 独立处理各网络数据分组。

处理器 12 的另一应用例是一附录处理器用打印引擎或作为存储子系统（即 RAID 磁盘存储器）的处理器。再一应用是作为配对引擎。举例来说，在证券业中，电子交易的出现要求用电子配对引擎来搓合买方和卖方的报单。可在系统 10 上完成上述和其他并行任务。

处理器 12 包括一将该处理器与第 2 总线 18 连接的总线接口 28。一实施例

中，总线接口 28 将处理器 12 与所谓的 FBUS18（FIFO 总线）连接。FBUS 接口 28 负责控制处理器 12 和形成该处理器与 FBUS18 的接口。FBUS18 是 64 位宽的 FIFO 总线，用于形成与媒体存取控制器（MAC）设备的接口。

处理器 12 包括一第二接口（例如 PCI 总线接口 24），该接口将驻留 PCI 总线 14 上的其他系统组成部分与处理器 12 连接。PCI 总线接口 24 提供一至存储器 16（例如 SDRAM 存储器 16a）的高速数据通路 24a。通过该通路，数据可从 SDRAM16a 通过直接存储器存取（DMA）的传送经 PCI 总线 14 转移。硬件多线程处理器 12 支持图像传送。硬件多线程处理器 12 能用多个 DMA 通道，因而若 DMA 传送的一个目标忙，另一 DMA 通道便可接管 PCI 总线对另一目标传送信息，以维持处理器 12 高效。此外，PCI 总线接口还支持目标和主机操作。目标操作是总线 14 上的从属装置通过对该操作起从属作用的读和写存取 SDRAM 场合的操作。主机操作中，处理器核心 20 直接对 PCI 接口 24 收、发数据。

每一功能单元连接 1 条或多条内部总线。下文将说明，内部总线是 32 位的双总线（即 1 条总线用于读，另 1 条用于写）。硬件多线程处理器 12 结构上还做成处理器 12 中内部总线带宽之和大于处理器 12 所接外部总线的带宽。处理器 12 包含内部核心处理器总线，例如 ASB 总线（高级系统总线：Advanced system Bus），该总线将处理器核心接到存储控制器 26a、26b 和 ASB 译码器 30，后文将说明。ASB 总线是配合 Strong Arm 处理器核心用的所谓 AMBA 总线的子集。处理器 12 还包含将微引擎单元接到 SRAM 控制器 26b、ASB 变换器 30 和 FBUS 接口 28 的专用总线 34。存储器总线 38 将存储控制器 26a、26b 接到总线接口 24、28 和包含用于引导操作等的快速擦写 ROM16c 的存储器系统 16。

参照图 2，每一微引擎 22a～22f 包含仲裁器，检查标志以判定可提供的工作线程。来自任一微引擎 22a～22f 的任何线程都可访问 SDRAM 控制器 26a、SRAM 控制器 26b 或 FBUS 接口 28。存储控制器 26a 和 26b 分别包含多个队列，以存放待处理的存储器指针请求。这些队列保持存储器指针的次序或者安排存储器指针来优化存储器带宽。举例来说，若线程 0 相对线程 1 独立或者无关，线程 1 和 0 便没有理由无法不按顺序完成其存储器指针指向 SRAM 单元。微引擎 22a～22f 对存储控制器 26a 和 26b 发布存储器指针请求。微引擎 22a～22f 以足够的存储器指针操作充满存储器子系统 26a 和 26b，使得该存储器子系统 26a 和 26b 成为处理器 12 运作的瓶颈。

若存储器子系统 16 充满本质上独立的存储器请求，处理器 12 便可进行存

储器指针分类。该存储器指针分类改善能达到的存储器带宽。如下文所述，存储器指针分类减少存取 SRAM 时所出现的空载时间或空泡。通过存储器指针指向 SRAM，在读写之间对信号线电流方向的切换，产生等待 SRAM16b 与 SRAM 控制器 26b 连接的导体上的电流稳定的空泡或空载时间。

也就是说，在总线上驱动电流的驱动器需要在状态变化前稳定。因此，读后接着写的重复周期会使峰值带宽下降。存储器指针分类允许组织指针指向存储器，使得长串的读出后面接着长串的写入。这可用于使流水线的空载时间最少，以有效达到接近最大可用带宽。指针分类有助于保持并行硬件环境线程。对于 SDRAM，指针分类使一存储组对另一存储组可隐藏预充电。具体而言，若存储器系统 166 组织成奇数存储组和偶数存储组，而处理器在奇数存储组上工作，存储控制器便可在偶数存储组启动预充电。若存储器指针在奇数和偶数存储组之间交替变化，便可预充电。通过安排存储器指针的顺序交替访问相对存储组，处理器 12 改善 SDRAM 的带宽。此外，还可采用其他优化。举例来说，可采用可归并的运作在存储器存取前归并场合的归并优化、通过检查地址不重新打开存储器开放页面场合的开放页面优化、将在下面说明的链接以及刷新机制。

FBUS 接口 28 支持 MAC 装置所支持的各端口用的收和发标志，还支持表明业务何时得到保证的中断标志。FBUS 接口 28 还包含对来自 FBUS18 的输入数据组首部进行处理的控制器 28a。控制器 28a 提取该分组的首部，并且在 SRAM 中进行可微编程源/宿/协议散列查找（用于地址平滑）。如果散列未成功分辨，将分组首部送到处理器核心 20 进行附加处理。FBUS 接口 28 支持下列内部数据事务：

FBUS 单元	（共用总线 SRAM）	至/来自微引擎
FBUS 单元	（经专用总线）	从 SDRAM 单元写入
FBUS 单元	（经 MBUS）	读出至 SDRAM

FBUS18 是标准业界总线，其中包含例如 64 位宽的数据总线、地址边带控制和读/写控制。FBUS 接口 28 能用一系列输入输出 FIFO29a～29b 输入大量数据。微引擎 22a～22f 从 FIFO29a～29b 取得数据，命令 SDRAM 控制器 26a 将来自己从总线 18 上的装置得到数据的接收 FIFO 的数据移入 FBUS 接口 28。该数

据可通过存储控制器 26a，经直接存储器存取送到 SDRAM 存储器 16a。同样，微引擎能将从 SDRAM26a 至接口 28 的数据经 FBUS 接口 28 移出到 FBUS18。

在微引擎之间分配数据功能。经由命令请求连接 SRAM26a、SDRAM26b 和 FBUS28。命令请求可以是存储器请求或 FBUS 请求。例如，命令请求能将数据从位于微引擎 22a 的寄存器移到共用资源，例如移到 SDRAM 位置、SRAM 位置、快速擦写存储器或某 MAC 地址。这些命令送出到每一功能单元和共用资源。然而，共用资源不需要保持数据的局部缓存。反之，共用资源存取位于微引擎内部的分布数据。这使微引擎 22a～22f 可局部存取数据，而不是对总线上存取的仲裁，冒争用总线的风险。利用此特征，等待数据内部到达微引擎 22a～22f 的阻塞周期为 0。

连接这些资源（例如存储控制器 26a 和 26b）的例如 ASB 总线 30、SRAM 总线 34 和 SDRAM 总线 38 等数据总线具有足够的带宽，使得无内部阻塞。因此，为了避免阻塞，处理器 12 具有每一功能单元配备内部总线最大带宽至少 2 倍的带宽要求。例如，SDRAM 可以 83MHz 运作于 64 位宽总线。SRAM 数据总线可具有读写分开的总线，例如可以是运作于 166MHz 的 32 位宽读出总线和运作于 166MHz 的 32 位宽写入总线。也就是说，运作于 166Mhz 的 64 位实际上是 SDRAM 带宽的 2 倍。

核心处理器 20 也可存取共用资源。核心处理器 20 经总线 32 直接与 SDRAM 控制器 26a、总线接口 24 和 SRAM 控制器 26b 通信。然而，为了访问微引擎 22a～22f 和任一微引擎 22a～22f 的传送寄存器，核心处理器 24 经 ASB 变换器 30 在总线 34 上存取微引擎 22a～22f。ASB 变换器 30 可在实体上驻留于 FBUS 接口 28，但逻辑上不同。ASB 变换器 30 进行 FBUS 微引擎传送寄存器位置与核心处理器地址（即 ASB 总线）之间的地址变换，以便核心处理器 20 能访问属于微引擎 22a～22c 的寄存器。

虽然如下文所述微引擎 22 可用寄存器组交换数据，但还提供便笺存储器 27，使微引擎能将数据写到该存储器，供其他微引擎读取。便笺式存储器 27 连接总线 34。

处理器核心 20 包含在 5 级流水线实现的 RISC 核心 50，该流水线进行在单个周期内对 1 个操作数或 2 个操作数的单周期移位，提供乘法支持和 32 位滚筒型移位支持。此 RISC 核心 50 具有标准 Strong Arm®体系结构，但由于性能上的原因，用 5 级流水线实现。处理器核心 20 还包含 16K 字节指令快速缓存

器 52、8K 字节数据快速缓存器 54 和预读取流缓存器 56。核心处理器 20 执行与存储器写入和指令读取并行的算术运算。核心处理器 20 经 ARM 规定的 ASB 总线与其他功能单元形成接口。ASB 总线是 32 位双向总线 32。

微引擎

参照图 3，示出微引擎 22a～22f 中一示范例，例如微引擎 22f。微引擎包含控制存储器 70，在一实施例中该存储器包含这里达 1024 字（每字 32 位）的 RAM。该 RAM 存储一微程序。该微程序可由核心处理器 20 加载。微引擎 22f 还包含控制器逻辑 72。该控制器逻辑包含指令译码器 73 和程序计数器（PC）单元 72a～72d。按硬件维持 4 个微程序计数器 72a～72d。微引擎 2f 还包含环境事件切换逻辑 74。该环境事件逻辑 74 从例如 SRAM26a、SDRAM26b 或处理器核心 20、控制及状态寄存器等共用资源的每一个接收消息（例如：序号#事件响应 SEQ_#_EVENT_RESPONSE、FBI 事件响应 FBI_EVENT_RESPONSE、SRAM 事件响应 SRAM_EVENT_RESPONSE、SDRAM 事件响应 SDRAM_EVENT_RESPONSE 和 ASB 事件响应 ASB_EVENT_RESPONSE）。这些消息提供有关请求的功能是否完成的信息。根据线程请求的功能是否完成和信号传送是否完成，线程需要等待该完成信号，如果线程能工作，便将该线程放到可用线程列表（未示出）。微引擎 22a 可具有最多例如 4 个可用线程。除执行线程局部所有的事件信号外，微引擎 22 还用全局的信令状态。借助该信令状态，执行线程可对全部微引擎 22 广播信号状态。接收请求可行信号后，微引擎中的全部任何线程可按这些信令状态分支。可用这些信令状态判定资源的可用性或资源提供服务是否适当。

环境事件逻辑 74 具有对 4 个线程的仲裁。一实施例中，仲裁是一循环机制。可用包括优先级排队或加权合理排队在内的其他技术。微引擎 22f 还包括一执行框（EBOX）数据通路 76，其中包含算术逻辑单元 76a 和通用寄存器组 76b。算术逻辑单元 76a 执行算术和逻辑功能以及移位功能。寄存器组 76b 具有数量较多的通用寄存器。图 3－4 中将说明，此实施例中，第 1 组（组 A）具有 64 个通用寄存器，第 2 组（组 B）也具有 64 个。这些通用寄存器形成窗口（后文将说明），以便其可相对寻址和绝对寻址。

微引擎 22f 还包含写传送寄存器堆栈 78 和读传送堆栈 80。这些寄存器也形成窗口，以便其可相对寻址和绝对寻址。资源的写入数据位于写传送寄存器堆栈 78。读寄存器堆栈 80 则用于从共用资源返回的数据。分别来自例如 SRAM

控制器 26a、SDRAM 控制器 26b 或核心处理器 20 之类共用资源的事件信号，与数据的到达串行或并行提供给环境事件仲裁器 74，提醒线程数据可用或数据已发送。传送寄存器组 78 和 80 都通过数据通路连接执行框（EBOX）76。一实施例中，读传送寄存器具有 64 个寄存器，写传送寄存器也有 64 个。

如图 3－3 所示，微引擎数据通路维持 5 级微流水线 82。该流水线包含查找微指令字 82a、形成寄存器文件地址 82b、从寄存器文件读操作数 82c、ALU 或移位或比较运算 82d 和结果写回寄存器 82e。通过提供回写数据旁路至 ALU/移位器单元，假定按寄存器文件而非 RAM 实现寄存器，微引擎可执行寄存器文件同时读写，完全隐藏写操作。

SDRAM 接口 26a 在读数据上对提出请求的微引擎送回一表明是否在读请求上出现奇偶差错的信号。微引擎采用任何送回的数据时，微引擎的微码负责校验 SDRAM 读奇偶标志。校验该标志时，如果设定该标志，分支动作将其消除。仅在启用 SDRAM 供校验时发送奇偶标志，SDRAM 受奇偶性防护。微引擎和 PCI 单元是通知奇偶差错的唯一请求者。因此，如果处理器核心 20 或 FIFO 要求奇偶性保护，微引擎就按请求帮助。微引擎 22a～22f 支持条件分支。分支判决结果是先前微控制指令设定条件码时，出现条件分支执行时间（不包含转移）最坏的情况。表 1 示出该等待时间如下：

	1	2	3	4	5	6	7	8
微存储查找	n1	cb	n2	XX	b1	b2	b3	b4
寄存器地址生成		n1	cb	XX	XX	b1	b2	b3
寄存器文件查找			n1	cb	XX	XX	b1	b2
ALU/shifter/cc				n1	cb	XX	XX	b1
写回			m2		n1	cb	XX	XX

其中，

nx 是预分支微字（n1 设定为 cc）

cb 是条件转移

bx 是后分支微字

xx 是异常微字

如表 1 所示，直到周期 4 才设定条件码 n1，能进行分支判决，这在本例中使用周期 5 上查找分支路径。微引擎由于必须在分支路径用操作 b1 填充流水

线前吸收流水线中操作 n2 和 n3（紧接分支后的 2 个微字），带来 2 个周期的分支等待时间损失。如果不进行分支，就不吸收微字，按常规继续执行。微引擎有若干机制用于减少或消除有效分支等待时间。

微引擎支持延迟分支。延迟分支是微引擎允许在分支后实施分支前出现 1 个或 2 个微字的情况（即分支作用在时间上“延迟”）。因此，如果能找到有用的工作填补分支微字后所浪费的周期，就能隐去分支等待时间。下面示出延迟 1 周期的分支，其中允许在 cb 后、b1 前执行 n2：

	1	2	3	4	5	6	7	8
微存储查找	n1	cb	n2	XX	b1	b2	b3	b4
寄存器地址生成		n1	cb	n2	XX	b1	b2	b3
寄存器文件查找			n1	cb	n2	XX	b1	b2
ALU/shifter/cc				n1	cb	n2	XX	b1
写回					n1	cb	n2	XX

下面示出 2 周期的延迟分支，其中 n2 和 n3 都允许在分支到 b1 出现前完成。注意，仅分支前在微字设定条件码时，允许 2 周期分支延迟。

	1	2	3	4	5	6	7	8	9
微存储查找	n1	cb	n2	n3	b1	b2	b3	b4	b5
寄存器地址生成		n1	cb	n2	n3	b1	b2	b3	b4
寄存器文件查找			n1	cb	n2	n3	b1	b2	b3
ALU/shifter/cc				n1	cb	n2	n3	b1	b2
写回					n1	cb	n2	n3	b1

微引擎也支持条件码估值。如果分支前判决分支的条件码设定 2 个或多个微字，则由于能早 1 周期进行分支判决，能消除 1 周期的等待时间如下：

	1	2	3	4	5	6	7	8
微存储查找	n1	n2	cb	XX	b1	b2	b3	b4
寄存器地址生成		n1	n2	cb	XX	b1	b2	b3
寄存器文件查找			n1	n2	cb	XX	b1	b2
ALU/shifter/cc				n1	n2	cb	XX	b1
写回					n1	n2	cb	XX

此例中，n1 设定条件码，n2 不设定条件码。因此，可在周期 4 而不是周

期 5 进行分支判决，以消除 1 周期的分支等待时间。下面的例子中，将 1 周期延迟与条件码提早设定加以组合，以完全隐去分支等待时间：

	1	2	3	4	5	6	7	8
微存储查找	n1	n2	cb	n3	b1	b2	b3	b4
寄存器地址生成		n1	n2	cb	n3	b1	b2	b3
寄存器文件查找			n1	n2	cb	n3	b1	b2
ALU/shifter/cc				n1	n2	cb	n3	b1
写回					n1	n2	cb	n3

其中，在 1 周期延迟分支前的 2 周期设定条件码 cc。

在不能提前设定条件码的情况下（即条件码设定在分支前的微字中时），微引擎支持试图减少 1 周期留下的暴露分支等待时间的分支猜测。通过“猜测”分支路径或串行路径，微定序器在确切知道执行何路径前预先取得猜测路径 1。猜测如果正确，就消除 1 周期分支等待时间如下：

	1	2	3	4	5	6	7	8
微存储查找	n1	cb	n1	b1	b2	b3	b4	b5
寄存器地址生成		n1	cb	XX	b1	b2	b3	b4
寄存器文件查找			n1	cb	XX	b1	b2	b3
ALU/shifter/cc				n1	cb	XX	b1	b2
写回					n1	cb	XX	b1

其中猜测进行分支而且也进行分支。如果微码猜测进行的分支不正确，微引擎仍然浪费 1 周期：

	1	2	3	4	5	6	7	8
微存储查找	n1	cb	n1	XX	n2	n3	n4	n5
寄存器地址生成		n1	cb	n1	XX	n2	n3	n4
寄存器文件查找			n1	cb	n1	XX	n2	n3
ALU/shifter/cc				n1	cb	n1	XX	n2
写回					n1	cb	n1	XX

其中猜测进行分支但不进行分支。

然而，微码猜测不进行分支时，有差异地分配等待时间损失。

对猜测不进行分支而且也不进行分支而言，不存在浪费周期，如下所示：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

微存储查找	n1	cb	n1	n2	n3	n4	n5	n6
寄存器地址生成		n1	cb	n1	n2	n3	n4	n5
寄存器文件查找			n1	cb	n1	n2	n1	b4
ALU/shifter/cc				n1	cb	n1	n2	n3
写回					n1	cb	n1	n2

然而，对猜测不进行分支但进行分支而言，存在 2 个浪费周期如下：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

微存储查找	n1	cb	n1	XX	b1	b2	b3	b4
寄存器地址生成		n1	cb	XX	XX	b1	b2	b3
寄存器文件查找			n1	cb	XX	XX	b1	b2
ALU/shifter/cc				n1	cb	XX	XX	b1
写回					n1	cb	XX	XX

微引擎可组合分支猜测和 1 周期分支延迟，以进一步改善结果。对猜测进行分支加上 1 周期延迟分支而且也进行分支而言，其结果为：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

微存储查找	n1	cb	n2	b1	b2	b3	b4	b5
寄存器地址生成		n1	cb	n2	b1	b2	b3	b4
寄存器文件查找			n1	cb	n2	b1	b2	b3
ALU/shifter/cc				n1	cb	n2	b1	b2
写回					n1	cb	n2	b1

上述情况下，通过执行 n2 和正确猜测分支方向，隐去 2 周期分支等待时间。如果微码猜测不正确，仍暴露 1 周期等待时间如下：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

微存储查找	n1	cb	n2	XX	n3	n4	n5	n6	n7
寄存器地址生成		n1	cb	n2	XX	n3	n4	n5	n6
寄存器文件查找			n1	cb	n2	XX	n3	n4	n5
ALU/shifter/cc				n1	cb	n2	XX	n3	n4
写回					n1	cb	n2	XX	n3

其中，猜测进行分支加上 1 周期延迟分支但不进行分支。

如果微码正确猜测不进行分支，流水线按常规不受干扰的情况顺序进行。微码错误猜测不进行分支，微引擎又暴露 1 周期非生产性执行如下：

```
                    | 1| 2| 3| 4| 5| 6| 7| 8| 9|
  ----------------+----+----+----+----+----+----+----+----+----+
  微存储查找          |n1|cb|n2|XX|b1|b2|b3|b4|b5|
  寄存器地址生成    |  |n1|cb|n2|XX|b1|b2|b3|b4|
  寄存器文件查找  |  |  |n1|cb|n2|XX|b1|b2|b3|
  ALU/shifter/cc  |  |  |  |n1|cb|n2|XX|b1|b2|
  写回              |  |  |  |  |n1|cb|n2|XX|b1|
```

其中，猜测不进行分支但进行分支，而且

nx 是预分支微字（n1 设定为 cc）

cb 是条件转移

bx 是后分支微字

xx 是异常微字

在转移指令的情况下，由于直到 ALU 级中存在转移的周期结束才知道分支地址，造成 3 个额外周期的等待时间如下：

```
                    | 1| 2| 3| 4| 5| 6| 7| 8| 9|
  ----------------+----+----+----+----+----+----+----+----+----+
  微存储查找          |n1|jp|XX|XX|XX|j1|j2|j3|j4|
  寄存器地址生成    |  |n1|jp|XX|XX|XX|j1|j2|j3|
  寄存器文件查找  |  |  |n1|jp|XX|XX|XX|j1|j2|
  ALU/shifter/cc  |  |  |  |n1|jp|XX|XX|XX|j1|
  写回              |  |  |  |  |n1|jp|XX|XX|XX|
```

环境切换

参照图 3－4，其中示出环境切换指令的格式。环境切换是促使选择不同环境（及其相关联 PC）的特殊分支形式。环境切换也引入一些分支等待时间。考虑以下的环境切换：

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

微存储查找	o1	ca	br	n1	n2	n3	n4	n5	n6
寄存器地址生成		o1	ca	XX	n1	n2	n3	n4	n5
寄存器文件查找			o1	ca	XX	n1	n2	n3	n4
ALU/shifter/cc				o1	ca	XX	n1	n2	n3
写回					o1	ca	XX	n1	n2

其中，

ox 是旧环境流

br 是旧环境中分支微字

ca 是环境再仲裁（致使环境切换）

nx 是新环境流

XX 是异常微字

环境切换中，致使“br”微字异常，以避免保留正确旧环境 PC 会造成的控制和定时复杂性。

按照分支前在微字上设定的 ALU 条件码操作的条件分支可选择 0、1 或 2 周期分支延迟模式。所有其他分支（包括环境再仲裁）可选择 0 或 1 周期分支延迟模式。可设计体系结构使环境仲裁微字在前置分支的分支延迟窗内、转移或使该微字为非法任选项。也就是说，某些实施例中，由于如上所述原因，其会造成保存旧环境 PC 过度复杂，流水线中分支转换时不允许发生环境切换。还可设计体系结构，使在前置分支的分支延迟窗内的分支、转换或者环境仲裁微字非法，以免分支行为复杂且不可预测。

每一微引擎 22a～22f 支持 4 个环境的多线程执行。其原因之一是使 1 个线程可正好在另一线程发布存储器指针后开始执行，并且必须等待直到该指针完成后才进行更多工作。由于存储器等待时间显著，此性能对维持微引擎硬件有效执行至关重要。换句话说，若仅支持一个线程执行，微引擎就会闲置大量周期，等待指针返回，从而使总的计算吞吐量减少。多线程执行通过跨越多个线程执行有用的独立工作，使微引擎可隐去存储器等待时间。提供 2 种同步机制，以便使线程可发布 SRAM 或 SDRAM 指针，并且在完成该访问时，接着与下一个时间点同步。

一种机制是立即同步。立即同步中，微引擎发布指针，并且立即换出该环

境。相应的指针完成时便发信号通知该环境。一旦发信号，便换回环境，以便当出现环境对换事件且轮到其运作时执行。因此，从单一环境指令流的角度看，微字在发出存储器指针后要到指针完成才得以执行。

第 2 种机制是延迟同步。延迟同步中，微引擎发布指针后，继续执行一些其他与指针无关的有用工作。过些时间，变成需要线程执行流与所发布指针的完成同步后再进一步工作。这时，执行同步微字，换出当前线程，并且在过一些时间完成指针时将其换回，或者由于已经完成指针而继续执行当前线程。用以下 2 种不同信令方案实现延迟同步。

若存储器指针与传送寄存器关联，在设定或消除相应传送寄存器有效位时便产生触发线程的信号。举例来说，设定寄存器 A 有效位时，便发出在传送寄存器 A 中存数据这种 SRAM 读出的信号。若存储器指针与传送 FIFO 或接收 FIFO 关联，而不是与传送寄存器关联，在 SDRAM 控制器 26a 中完成访问时产生信号。微引擎调度器中仅保持每一环境一种信号状态，因而此方案中只能存在一个待处理信号。

至少有 2 种可设计微控制器微程序的一般操作范例。一种是优化总体的微控制器计算吞吐量和总体的存储器带宽，其代价是花费一个线程执行等待时间。当系统具有对非相关数据分组执行每一微引擎多线程的多重微引擎时，此范例会有意义。

第 2 种范例是以总体的微引擎计算吞吐量和总体存储器带宽的代价来优化微引擎执行等待时间。此范例涉及以实时约束执行线程，该约束支配按某规定时间必须绝对完成的某工作。这种约束要求给单一线程执行的优化比诸如存储器带宽或总体计算吞吐量之类其他考虑高的优先级。实时线程隐含仅执行一个线程的单一微引擎。目标是使单一实施线程尽快执行，而多线程的执行妨碍此性能，所以不处理多线程。

在发布存储器指针和环境切换方面，这 2 种范例的编码方式显著不同。在实时情况下，目标是尽快发布尽量多的存储器指针，以便使这些指针所带来的存储器等待时间最短。已尽量提早发布尽量多的指针，目标是微引擎与指针尽可能并行执行尽量多的计算。与实时优化时相对应的计算流是：

o) 发布存储器指针 1

o) 发布存储器指针 2

o) 发布存储器指针 3

o) 进行与存储器指针 1、2、3 无关的工作

o) 与存储器指针 1 的完成同步

o) 进行取决于存储器指针 1 且与存储器指针 2 和 3 无关的工作

o) 根据前面的工作发布新存储器指针

o) 与存储器指针 2 的完成同步

o) 进行取决于存储器指针 1 和 2 且与存储器指针 3 无关的工作

o) 根据前面的工作发布新存储器指针

o) 与存储器指针 3 的完成同步

o) 进行取决于全部 3 个指针完成的工作

o) 根据前面的工作发布新存储器指针

反之，对吞吐量和带宽的优化则采取不同方法。对微引擎计算吞吐量和总体存储器带宽而言，不考虑单线程执行等待时间。为了实现这点，目标是对每一线程在微程序上均匀分隔存储器指针。这将给 SRAM 和 SDRAM 的控制器提供均匀的存储器指针流，并且使总可获得一线程的概率最大，以隐去换出另一线程时带来的存储器等待时间。

寄存器文件地址类型

参照图 3－5，所存在的 2 个寄存器地址空间是局部可存取寄存器和全部微引擎均可存取的全局可存取寄存器。通用寄存器（GRP）做成 2 个分开组（A 组和 B 组），其地址逐字交错，使得 A 组寄存器具有 LSB=0，B 组寄存器具有 LSB=1。每组可进行本组内 2 个不同字的同时读写。

整个组 A 和组 B 上，寄存器集合 76b 也组织成每一线程具有可相对寻址的 32 个寄存器的 4 个窗 $76b_0$～$76b_3$。因此，线程 0 在 77a（寄存器 0）找到其寄存器 0，线程 1 在 77b（寄存器 32）找到其寄存器 0，线程 2 在 77c（寄存器 64）找到其寄存器 0，线程 3 在 77d（寄存器 96）找到其寄存器 0。支持相对寻址，以便多线程能准确使用相同的控制存储器和位置，但访问不同的宁寄存器窗，并执行不同功能。寄存器窗寻址和寄存器组寻址的使用，仅在以微引擎 22f 中以双端口 RAMS 提供必要的读带宽。

这些开窗的寄存器不需要保存环境切换之间的数据，从而消除环境对换文件或堆栈的常规推入和推出。这里环境切换对从一环境到另一环境的变化具有 0 周期的开销。相对寄存器寻址将寄存器组划分成跨越通用寄存器集合地址宽

度的窗。相对寻址允许访问相对于窗起始点的任何窗。此体系结构内也支持绝对寻址，其中，通过提供寄存器的准确地址，任何线程可访问任一绝对寄存器。

通用寄存器 48 的寻址可出现 2 种方式，取决于微字格式。这 2 种方式是绝对方式和相对方式。绝对方式中，在 7 位源段（a6～a0 或 b6～b0）直接指定寄存器地址的寻址：

```
                7 6 5 4 3 2 1 0
               +…+…+…+…+…+…+…+…+
A GPR:         |a6|0|a5|a4|a3|a2|a1|a0|  a6=0
B GPR:         |b6|1|b5|b4|b3|b2|b1|b0|  b6=0
SRAM/ASB:      |a6|a5|a4|0|a3|a2|a1|a0|  a6=1,a5=0,a4=0
SDRAM:         |a6|a5|a4|0|a3|a2|a1|a0|  a6=1,a5=0,a4=1
```

在 8 位宿段（d7～d0）直接指定寄存器地址：

```
                7 6 5 4 3 2 1 0
               +…+…+…+…+…+…+…+…+
A GPR:         |d7|d6|d5|d4|d3|d2|d1|d0|   d7=0,d6=0
B GPR:         |d7|d6|d5|d4|d3|d2|d1|d0|   d7=0,d6=1
SRAM/ASB:      |d7|d6|d5|d4|d3|d2|d1|d0|   d7=1,d6=0,d5=0
SDRAM:         |d7|d6|d5|d4|d3|d2|d1|d0|   d7=1,d6=0,d5=1
```

若<a6:a5>=1,1，<b6:b5>=1,1 或<d7:d6>=1,1，低端位便变换成环境相对地址字段（下文说明）。当 A、B 绝对字段指定非相对 A 或 B 源地址时，仅能对 SRAM/ASB 和 SDRAM 地址空间的低端半部分寻址。实际上，读绝对方式的 SRAM/SDRAM 具有有效地址空间。但由于该限制不适用宿段，写 SRAM/SDRAM 可用全地址空间。

相对方式中，在按 5 位源段（a4～a0 或 b4～b0）规定的环境空间内偏置指定地址的寻址：

```
                7 6 5 4 3 2 1 0
               +…+…+…+…+…+…+…+…+
A GPR:         |a4|0|环境|a2|a1|a0|      a4=0
B GPR:         |b7|1|环境|b2|b1|b0|      b4=0
SRAM/ASB:      |ab4|0|ab3|环境|b1|ab0|   ab4=1,ab3=0
SDRAM:         |ab4|0|ab3|环境|b1|ab0|   ab4=1,ab3=1
```

或者在按 6 位宿段（d5～d0）规定的环境空间内偏置指定地址的寻址：

7 6 5 4 3 2 1 0

+…+…+…+…+…+…+…+…+

A GPR: |d5|d4|环境 d2|d1|d0| d5=0, d4=0

B GPR: |d5|d4|环境|d2|d1|d0| d5=0, d4=1

SRAM/ASB: |d5|d4|d3|环境|d1|d0| d5=1, d4=0, d3=0

SDRAM: |d5|d4|d3|环境|d1|d0| d5=1, d4=0, d3=1

如果<d5:d4>=1, 1，则宿地址找不到有效寄存器，因而不回写宿操作数。

从微引擎和存储控制器可全局存取以下寄存器：

散列单元寄存器

便笺和共用寄存器

接收 FIFO 和接收状态 FIFO

发送 FIFO

发送控制 FIFO

不中断驱动微引擎。执行每一微流直到完成为止，根据处理器 12 中其他装置用信号通知的状态选择一新流。

参照图 4，SDRAM 存储控制器 26a 包含存储器指针队列 90，其中存储器指针请求从各微引擎 22a～22f 到达。存储控制器 26a 包括一仲裁器 91，该仲裁器选择下一微引擎指针请求至任何功能单元。设一个微引擎提出访问请求，该请求会来到 SDRAM 控制器 26a 内部的地址和命令队列 90。若该访问请求具有称为“经优化存储位”的位集合，便将进入的指针请求分类为偶数组队列 90a 或奇数组队列 90b。若存储器指针请求没有存储器优化位集合，系统设定就转入一排序队列 90c。SDRAM 控制器 26 是一 FBUS 接口 28、核心处理器 20 和 PCI 接口 24 所共用的资源。SDRAM 控制器 26 还维持一用于执行读出-修改-写入自动操作的状态机。SDRAM 控制器 26 还对 SDRAM 的数据请求进行字节对准。

命令队列 90c 保持来自微引擎的指针请求的排序。根据一系列奇数和偶数组指针，可要求信号仅在指向偶数组和奇数组两者的存储器指针序列完成时返回信号。若微引擎 22f 将存储器指针分类成奇数组指针和偶数组指针，并且存储器指针在奇数组前漏出其中一个组（例如偶数组），但在最后的偶数指针上信号肯定，便可想象存储控制器 26a 可返回信号通知微引擎已完成存储器请求，即使奇数组指针不提供服务也这样。这种现象可造成一相干问题。通过提供排

序队列 90c 使微引擎可让多个待处理存储器指针中仅最后的存储器指针通知完成，从而可避免上述情况。

SDRAM 控制器 26a 还包含高优先级队列 90d。高优先级队列 90d 中，来自一个微引擎的输入存储器指针直接进入高优先级队列，并且以高于其他队列中其他存储器指针的优先级进行工作。偶数组队列 90a、奇数组队列 90b、命令队列 90c 和高优先级队列 90d，所有这些队列都在一个 RAM 结构中实现。该结构逻辑上分成 4 个不同的窗，各窗分别具有其本身首部和尾部指针。由于填入和漏出操作仅是单一输入和单一输出，可将它们置于同一 RAM 结构以提高 RAM 结构密度。

SDRAM 控制器 26a 还包括核心总线接口逻辑即 ASB 总线 92。ASB 总线接口逻辑 92 形成核心处理器 20 与 SDRAM 控制器 26a 的接口。ASB 总线是一包含 32 位数据通路和 28 位地址通路的总线。通过 MEM ASB 数据设备 98（例如缓存器）对存储器存取数据。MEM ASB 数据设备 98 是一写数据队列。若有从核心处理器 20 经 ASB 接口 92 进入的数据，该数据可存入 MEM ASB 设备 98，接着通过 SDRAM 接口 110 从 MEM ASB 设备 98 移至 SDRAM 存储器 16a。虽然未示出，但可对读出提供相同的队列结构。SDRAM 控制器 26a 还包括一引擎 97 从微引擎和 PCI 总线拉进数据。

附加队列包括保持若干请求的 PCI 地址队列 94 和 ASB 读/写队列 96。存储器请求经复用器 106 送至 SDRAM 接口 110。复用器 106 由 SDRAM 仲裁器 91 控制，该仲裁器检测各队列满员程度和请求状态，并根据优先级业务控制寄存器 100 存放的可编程值，从所检测的情况判定优先级。

一旦对复用器 106 的控制选择一存储器指针请求，就将此存储器指针请求送至一译码器 108 对其进行译码，并生成一地址。该经过解码的地址送至 SDRAM 接口 110，将其分解成行地址和列地址选通信号来存取 SDRAM16a，并通过将数据送至总线 112 的数据线 16a 读出或写入数据。一实施例中，总线 112 实际上是 2 条分开的总线，而不是单一总线。该分开的总线会包含连接分布式微引擎 22a～22f 的读出总线和连接分布式微引擎 22a～22f 的写入总线。

SDRAM 控制器 26a 其特征在于，在队列 90 中存储存储器指针时，除可设定该经优化存储位外，还有“链接位”。该链接位当其设定时允许对邻接存储器指针专门处理。如上文所述，仲裁器 12 控制选择哪一微引擎在命令总线上将存储器指针请求提供给队列 90（图 4）。对链接位的肯定将控制仲裁器使之选

择先前请求该总线的功能单元，这是因为对链接位的设定表明微引擎发出一链接请求。

设定链接位时，会按队列 90 接收邻接存储器指针。由于邻接存储器指针是来自单一线程的多存储器指针，这些邻接指针通常按排序队列 90c 存储。为了提供同步，存储控制器 26a 仅需要在完成时在链接存储器指针末端给出信号。但经优化存储器链接中，（例如经优化存储位和链接位设定时）存储器指针便会进入不同组，并且在其他组充分露出前有可能对发布信号“完成”的其中一个存储组完成，这样就破坏相干性。因此，控制器 110 用链接位保持来自当前队列的存储器指针。

参照图 4－3，示出 SDRAM 控制器 26a 中仲裁策略的流程表示。仲裁策略优待链接微引擎存储器请求。处理过程 115 通过链接微引擎存储器指针请求的检查 115a 开始进行。过程 115 停留于链接请求，直到使链接位清零。过程对后面接着 PCI 总线请求 115c、高优先级队列业务 115d、相对组请求 115e、排序队列请求 115f 和相同组请求 115g 的 ASB 总线请求 115b 进行检查。对链接请求提供完整的业务，而对业务 115b～115d 则按循环顺序提供。仅当业务 115a～115d 完全漏出时，该过程才进行业务 115e～115g 的处理。当先前 SDRAM 存储器请求对链接位设定时，设定所链接的微引擎存储器指针请求。当设定链接位时，仲裁引擎便再次仅对相同队列服务，直到链接位清零。ASB 处于等待状态时对 Strong Arm 核心带来严重性能损失，故 ASB 优先级高于 PCI。因为 PCI 的等待时间要求，PCI 优先级高于微引擎。但就其他总线而言，该仲裁优先级可能不同。

如图 4－4 所示，示出的是不具有有效存储器优化和具有有效存储器优化的典型存储器定时。可以知道，对有效存储器优化的利用使总线应用程度最大，从而隐去实际 SDRAM 装置内固有的等待时间。本例中，非优化存取可用 14 周期，而优化存取则可用 7 周期。

参照图 5，其中示出 SRAM 的存储控制器 26b。该存储控制器 26b 包括一地址及命令队列 120。存储控制器 26a（图 4）具有一基于奇数和偶数分组的存储器优化队列，存储控制器 26b 则根据存储器操作类型（即读出或写入）优化。地址及命令队列 120 包括一高优先级队列 120a、一作为 SRAM 执行的主导存储器指针功能的读队列 120b、以及一通常包括要非优化的全部 SRAM 读写的排序队列 120c。尽管未图示，地址及命令队列 120 也可包括一写队列。

SRAM 控制器 26b 还包括核心总线接口逻辑即 ASB 总线 122。ASB 总线接口逻辑 122 形成核心处理器 20 与 SRAM 控制器 26b 的接口。ASB 总线是一包括 32 位数据通路和 28 位地址通路的总线。通过 MEM ASB 数据设备 128（例如缓存器）对存储器存取数据。MEM ASB 数据设备 128 是一写数据队列。若有从核心处理器 20 通过 ASB 接口 122 的进入数据，该数据可存入 MEM ASB 设备 128，接着通过 SRAM 接口 140 从 MEM ASB 设备 128 移至 SRAM 存储器 16b。虽未图示，但可提供相同的队列结构用于读出。SRAM 控制器 26b 还包括一引擎 127 从微引擎和 PCI 总线拉进数据。

存储器请求经复用器 126 送至 SDRAM 接口 140。复用器 126 由 SDRAM 仲裁器 131 控制，该仲裁器检测各队列满员程度和请求状态，并根据优先级业务控制寄存器 130 存放的可编程值，从所检测的情况判定优先级。一旦对复用器 126 的控制选择一存储器指针请求，就将此存储器指针请求送至一译码器 138 对其进行译码，并生成一地址。SRAM 单元保持对经存储器映射的离片 SRAM 和扩展 ROM 的控制。SRAM 控制器 26b 可对例如 16M 字节寻址，而例如用于 SRAM16b 的 8M 字节映射保留用于专用功能，其中包括通过快速擦写 ROM16 的引导空间；MAC 器件 13a、13b 用的控制台端口存取访问；以及对关联（RMON）计数器的存取。SRAM 用于局部查找表和队列管理功能。

SRAM 控制器 26b 支持下列事务：

微引擎请求 （经专用总线） 至/自 SRAM

核心处理器 （经 ASB 总线） 至/自 SRAM。

SRAM 控制器 26b 进行存储器指针分类以使 SRAM 接口 140 至存储器 16b 的流水线中延迟（空泡）最少。SRAM 控制器 26b 根据读功能进行存储器指针分类。一个空泡可以是 1 周期或者 2 周期，取决于所用存储器件的类型。

SRAM 控制器 26b 包括一锁定查找器件 142，是一 8 输入项地址内容可寻址存储器用于对读出锁定的查找。每一位置包括一由后续读出-锁定请求检查的有效位。地址及命令队列 120 还包括一读出锁定失效队列 120d。该队列 120d 用于保持因存储器一部分存在锁定而失效的读存储器指针请求。也就是说，其中一个微引擎所发布的存储器请求所具有的读出锁定请求在地址及控制队列 120 中处理。该存储器请求将对排序队列 120c 或读队列 120b 运作，并将其识别为读出锁定请求。控制器 26b 将存取锁定查找器件 142 以判断该存储位置是否已经锁定。若此存储位置根据任何先前的读出锁定请求而被锁定，该存储锁

定请求将失效，并将存入读出锁定失效队列 120d。若解锁或者 142 表示该地址未锁定，SRAM 接口 140 就将用该存储器指针的地址对存储器 16b 进行常规的 SRAM 地址读/写请求。命令控制器及地址发生器 138 也会将该锁定输入锁定查找器件 142，以便后续读出锁定请求将发现该存储位置被锁定。锁定需要结束后通过对程序中微控制指令的运作来使存储位置解锁。通过对 CAM 中有效位清零将该位置解锁。解锁后，读出锁定失效队列 120d 变成最高优先级队列，给丢失的全部排队的读出锁定一次机会发出一存储器锁定请求。

如图 5－3 所示，示出的是没有有效存储器优化和具有有效存储器优化的静态随机存取存储器的典型时序。可以知道，分组读写改善消除死周期的周期时间。

参照图 6，示出的是微引擎 22 和 FBUS 接口逻辑（FBI）之间的通信。网络应用中的 FBUS 接口 28 可对来自 FBUS18 的来向数据分组进行首部处理。FBUS 接口 28 所进行的一项关键功能是对数据分组首部的提取和对 SRAM 中可微编程源/宿/协议散列查找。若该散列不能成功分辨，数据分组首部就提升至核心处理器 28 用于更为高级的处理。

FBI 28 包含发送 FIFO 182、接收 FIFO 183、散列单元 188 以及 FBI 控制及状态寄存器 189。这 4 个单元通过对微引擎中与传送寄存器 78、80 连接的 SRAM 总线 28 的时间复用存取与微引擎 22 通信。也就是说，全部对微引擎的收发通信都通过传送寄存器 78、80。FBUS 接口 28 包括一用于在 SRAM 不用 SRAM 数据总线（部分总线 38）的时间周期期间将数据推入传送寄存器的推状态机 200 以及一用于从相应微引擎中的传送寄存器当中读取数据的拉状态机 202。

散列单元包括一对 FIFO 18a、188b。该散列单元判定 FBI 28 收到一 FBI 散列请求。散列单元 188 从进行调用的微引擎 22 当中取得散列键。读取该键并散列后，将索引号送回进行调用的微引擎 22。在单个 FBI 散列请求下，进行多达 3 次散列。总线 34 和 38 均为单向：SDRAM 推/拉数据总线以及 S 总线推/拉数据总线。上述总线每一条需要对适当微引擎 22 的传送寄存器提供读/写控制的控制信号。

传送寄存器通常要求保护其环境控制以保证读出的正确性。具体来说，若线程 1 用写传送寄存器对 SDRAM 16a 提供数据，线程 1 必须等到 SDRAM 控制器 16a 返回的信号表明该寄存器已提升并可重新使用才重写此寄存器。每次写均不需要表明已经完成该功能的目的地所返回的信号，其原因在于若该线程用多

个请求对该目的地的相同命令队列写入，该命令队列内确保完成命令，因而仅最后的命令需要将信号传回该线程。但若该线程采用多个命令队列（排序和读出），这些命令请求就必需分解成独立的环境任务，以便通过环境对换保持排序。本节开头提出的特例涉及对 FBUS 状态信息采用 FBI 至传送寄存器的非请求型推（PUSH）的某类操作。为了保护传送寄存器的读/写判定，FBI 当设定这些专用 FBI 推操作时提供一专用的推保护信号。

采用 FBI 非请求型推方法的任何微引擎 22 必须在存取传送寄存器同意的 FBUS 接口/微引擎前测试该保护标志。若该标志并非肯定，该微引擎便可存取传送寄存器。若该标志肯定，则存取寄存器前环境应等待 N 周期。根据所推的传送寄存器个数加上前端保护窗，确定先验的此计数。基本思想是，微引擎必须测试此标志，然后以连续周期快速将希望从所读传送寄存器读出的数据移到 GPR，因而推引擎不与微引擎读出冲突。

其他实施例

应理解，虽然结合详细说明描述了本发明，但上述描述用于说明，并非限定本发明范围，该范围由所附权利要求的范围限定。其他方面、优点和修改均在以下权利要求书范围内。

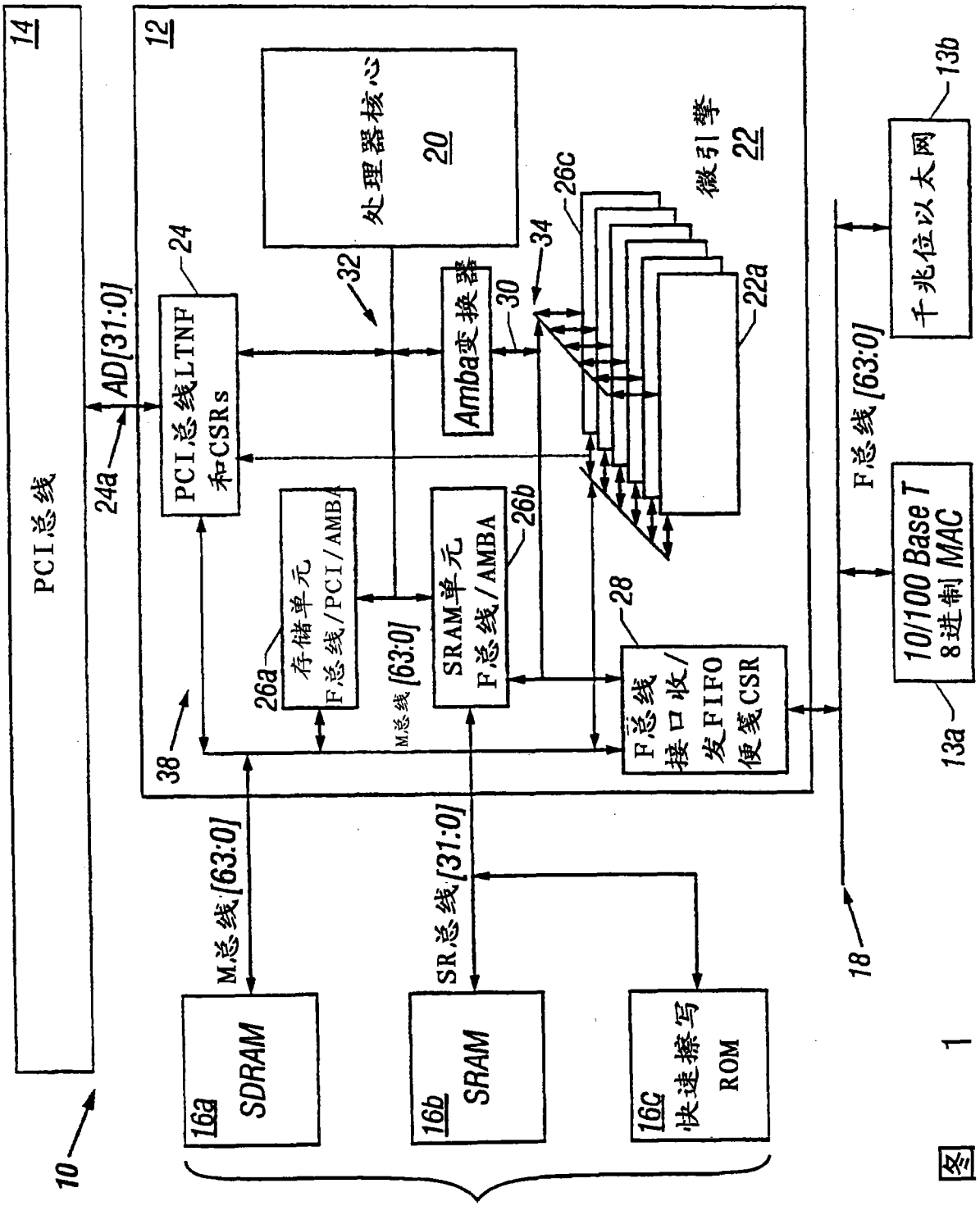

14
PCI总线
12
处理器核心
20
微引擎
22
22a
24
PCI总线LTNF
和CSRs
AD[31:0]
24a
32
34
Amba变换器
30
26c
存储单元
F总线/PCI/AMBA
26a
M总线[63:0]
SRAM单元
F总线/AMBA
26b
28
F总线
接口收/
发FIFO
便笺CSR
38
F总线[63:0]
千兆位以太网
13b
10/100 Base T
8进制MAC
13a
18
M总线[63:0]
SR总线[31:0]
16a
SDRAM
16b
SRAM
16c
快速擦写
ROM
16
10

图 1

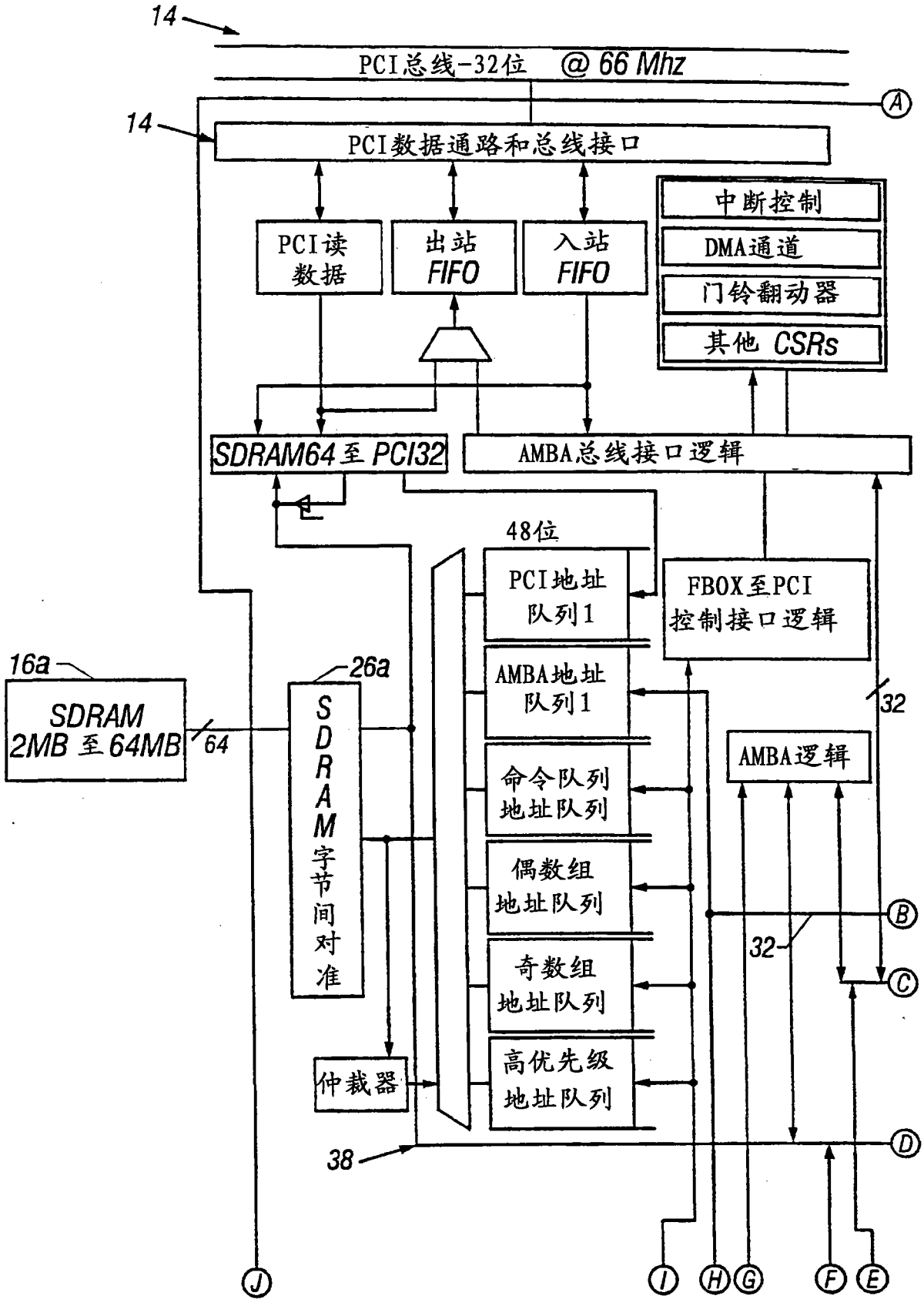

14
PCI总线-32位 @ 66 Mhz
A
14
PCI数据通路和总线接口
PCI读数据
出站FIFO
入站FIFO
中断控制
DMA通道
门铃翻动器
其他 CSRs
SDRAM64至PCI32
AMBA总线接口逻辑
48位
PCI地址队列1
FBOX至PCI控制接口逻辑
16a
SDRAM 2MB 至64MB
64
26a
SDRAM字节间对准
AMBA地址队列1
32
AMBA逻辑
命令队列地址队列
偶数组地址队列
B
32
奇数组地址队列
C
仲裁器
高优先级地址队列
38
D
J
I
H
G
F
E

图 2-1

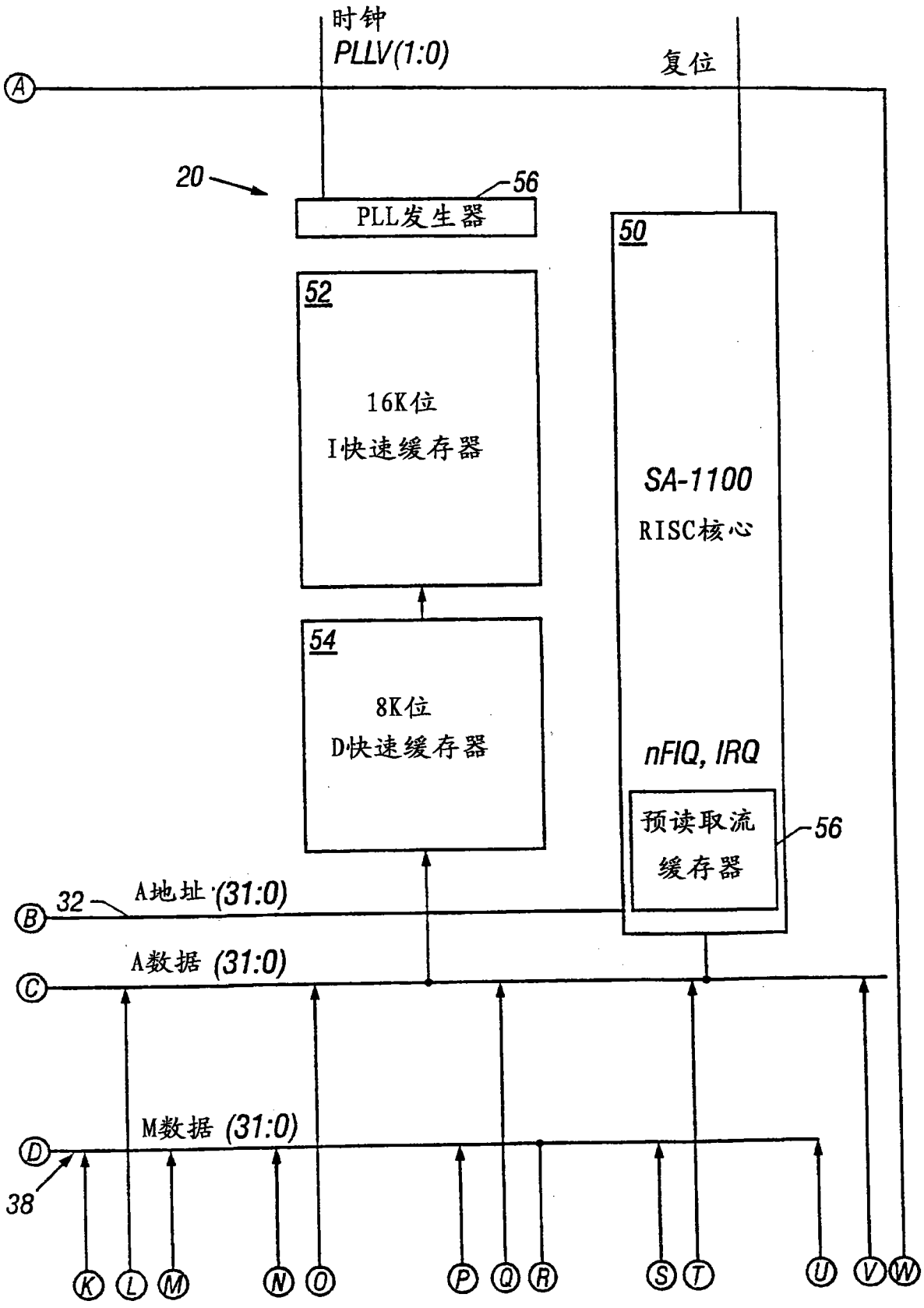

时钟
PLLV(1:0)
复位
A
20
56
PLL发生器
50
52
16K位
I快速缓存器
SA-1100
RISC核心
54
8K位
D快速缓存器
nFIQ, IRQ
预读取流
缓存器
56
B
32
A地址 (31:0)
C
A数据 (31:0)
D
M数据 (31:0)
38
K
L
M
N
O
P
Q
R
S
T
U
V
W

图 2-2

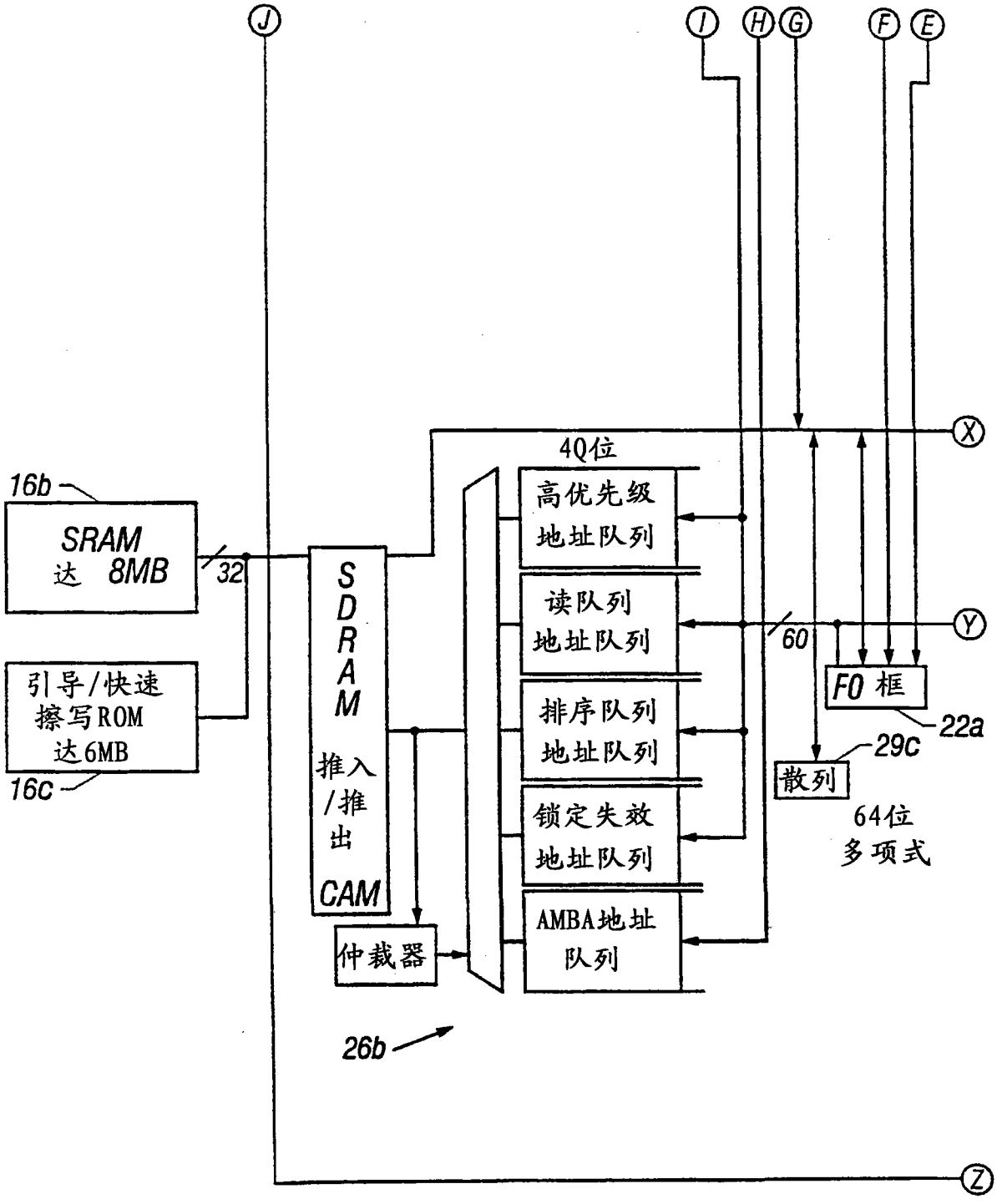

J
I
H
G
F
E
X
Y
Z
16b
SRAM
达 8MB
32
引导/快速
擦写ROM
达6MB
16c
SDRAM
推入
/推
出
CAM
仲裁器
26b
4Q位
高优先级
地址队列
读队列
地址队列
排序队列
地址队列
锁定失效
地址队列
AMBA地址
队列
60
F0 框
22a
29c
散列
64位
多项式

图 2-3

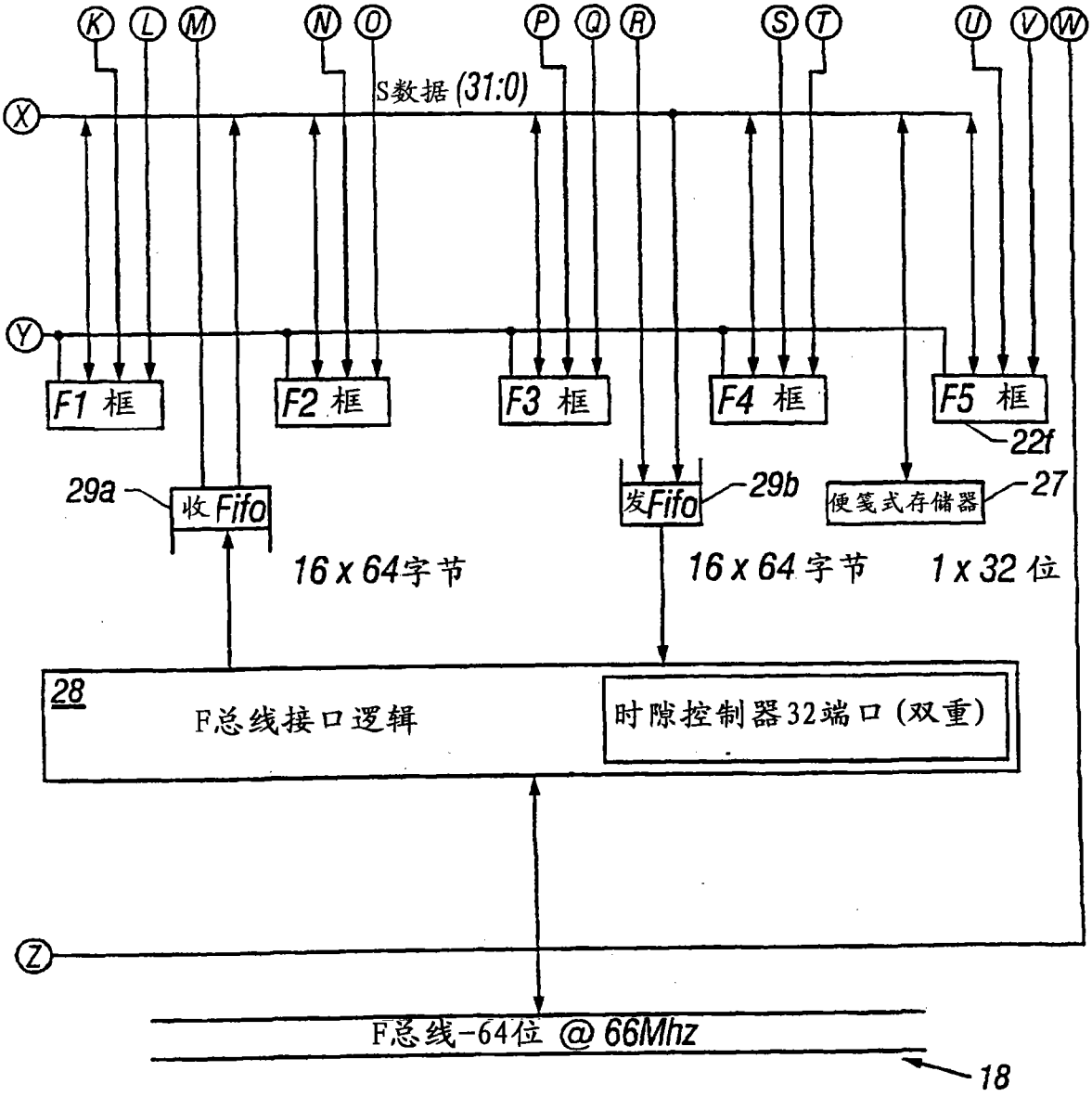

K
L
M
N
O
P
Q
R
S
T
U
V
W
S数据 (31:0)
X
Y
F1 框
F2 框
F3 框
F4 框
F5 框
22f
29a
收 Fifo
16 x 64字节
发Fifo
29b
16 x 64 字节
便笺式存储器
27
1 x 32 位
28
F总线接口逻辑
时隙控制器32端口 (双重)
Z
F总线-64位 @ 66Mhz
18

图 2-4

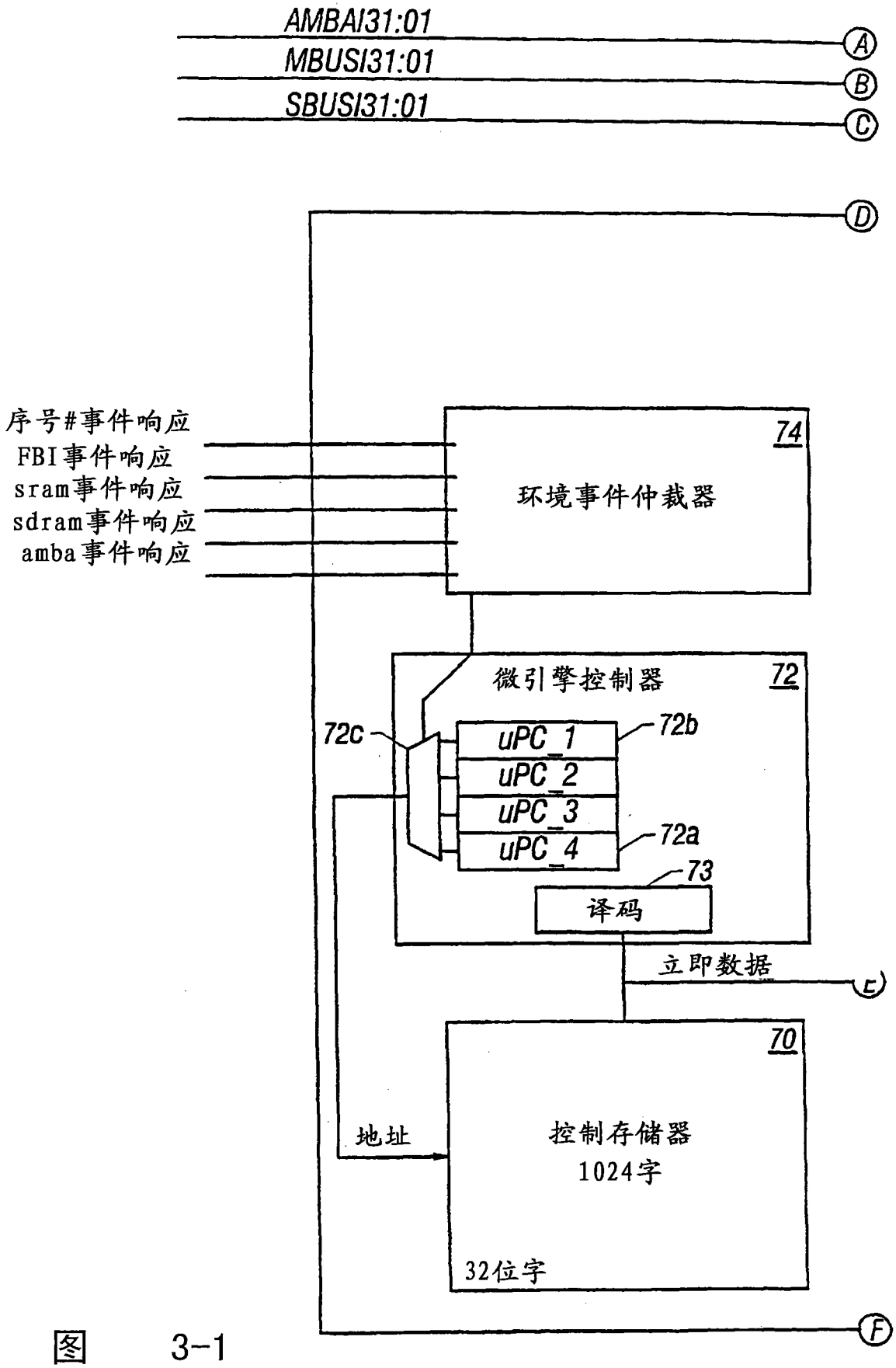

AMBAI31:01
MBUSI31:01
SBUSI31:01
A
B
C
D
序号#事件响应
FBI事件响应
sram事件响应
sdram事件响应
amba事件响应
74
环境事件仲裁器
微引擎控制器
72
72c
uPC_1
uPC_2
uPC_3
uPC_4
72b
72a
73
译码
立即数据
E
70
地址
控制存储器
1024字
32位字
F

图 3-1

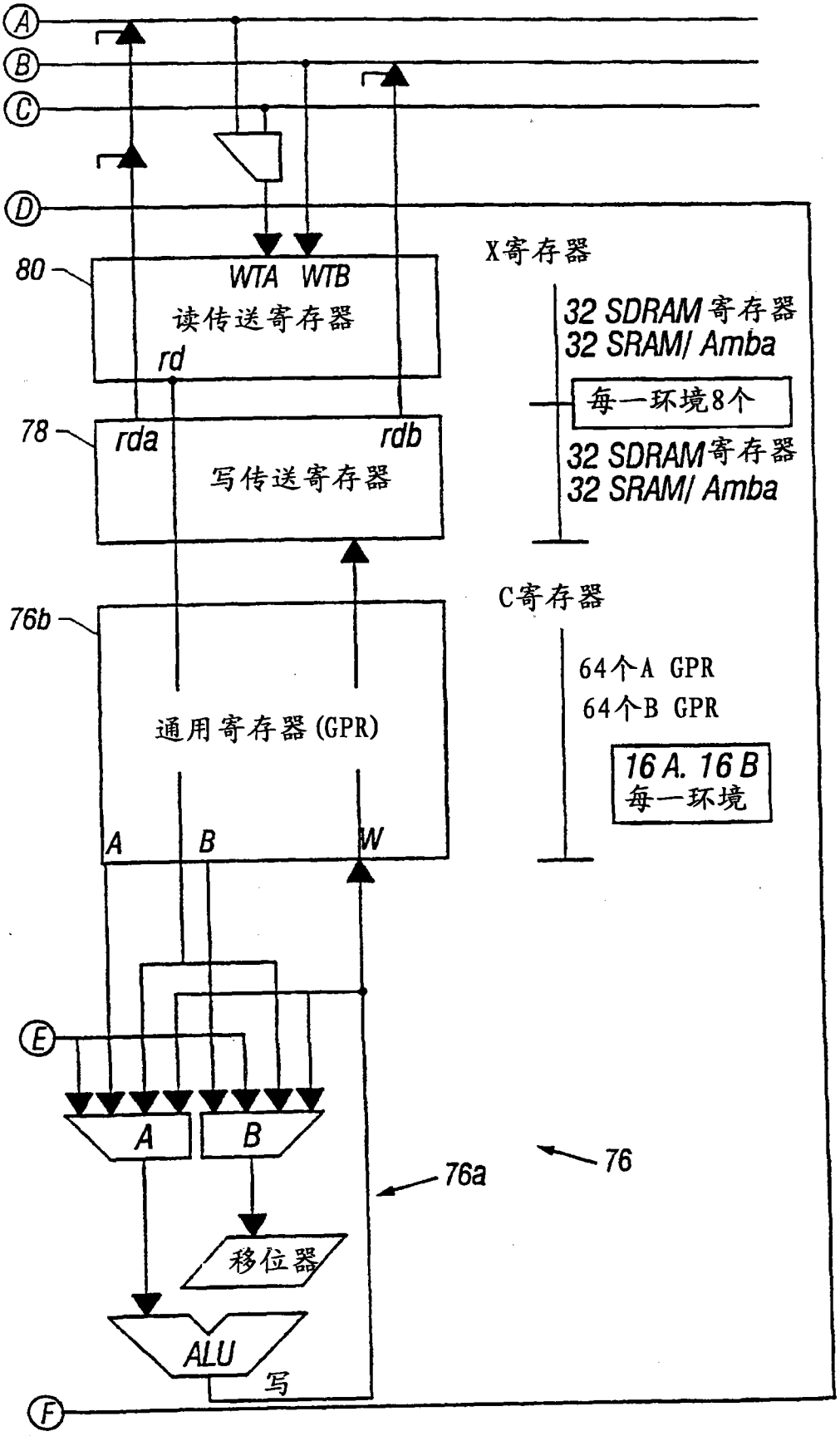

A
B
C
D
80
WTA WTB
读传送寄存器
rd
78
rda
rdb
写传送寄存器
76b
通用寄存器(GPR)
A
B
W
E
A
B
移位器
ALU
写
F
76a
76
X寄存器
32 SDRAM寄存器
32 SRAM/ Amba
每一环境8个
32 SDRAM寄存器
32 SRAM/ Amba
C寄存器
64个A GPR
64个B GPR
16 A. 16 B
每一环境

图 3-2

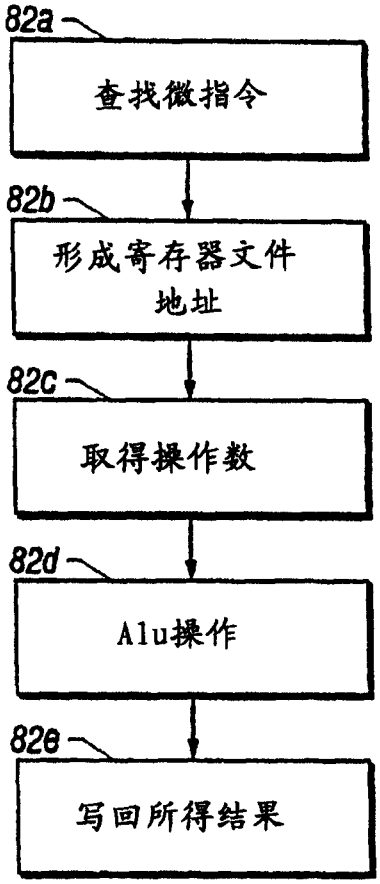
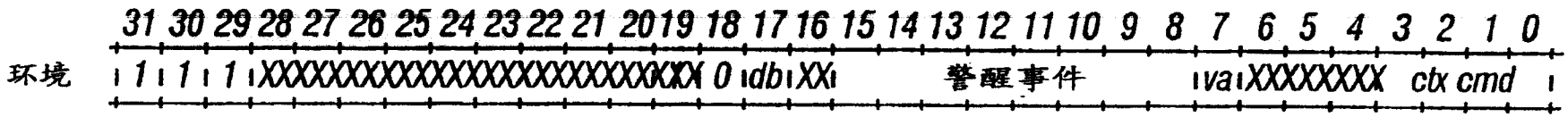


图3－3



环境描述符:

1) 警醒事件(位8 – 位15)
   0 = 删除
   1 = 随意
   2 = SRAM
   4 = SDRAM
   8 = FBI
  16 = 线程间隙
  32 = PCI_DMA_1
  64 = PCI_DMA_2
 128= SEQ_NUM_LSB
2) db -> 分支延迟量 (位 17)
3) va -> 序列号值 (位 7)
4) OPCODE 位 (29-31)
5) cxt_cmd

图3—4

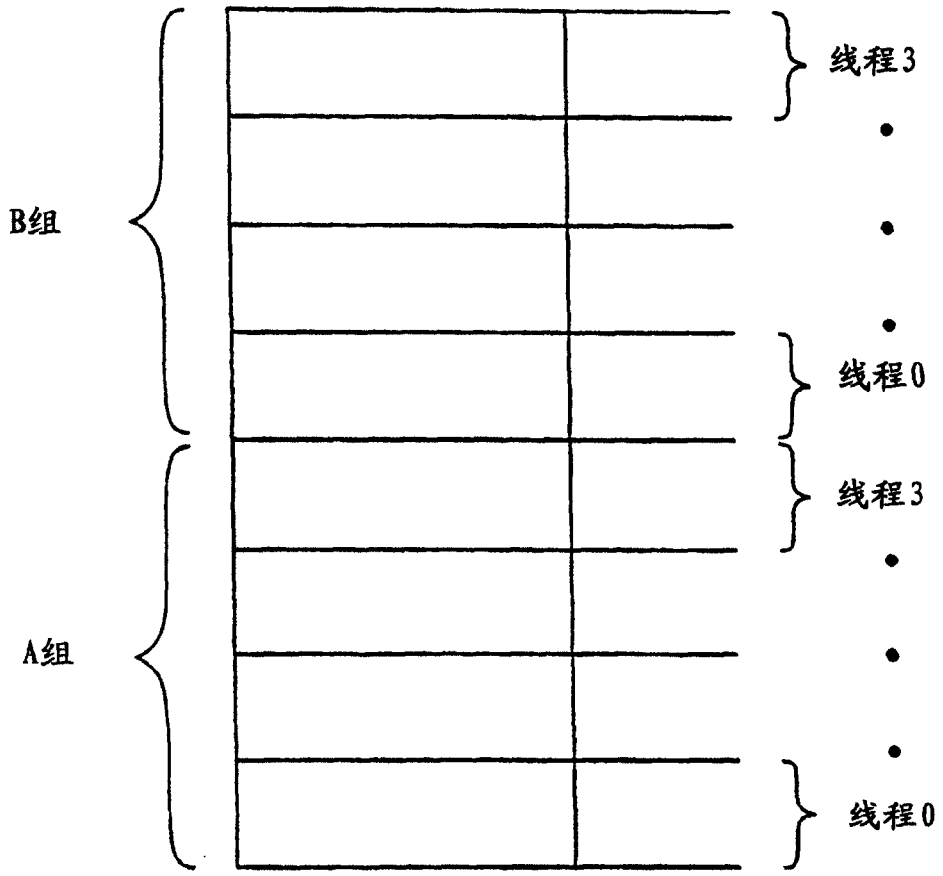


图3—5

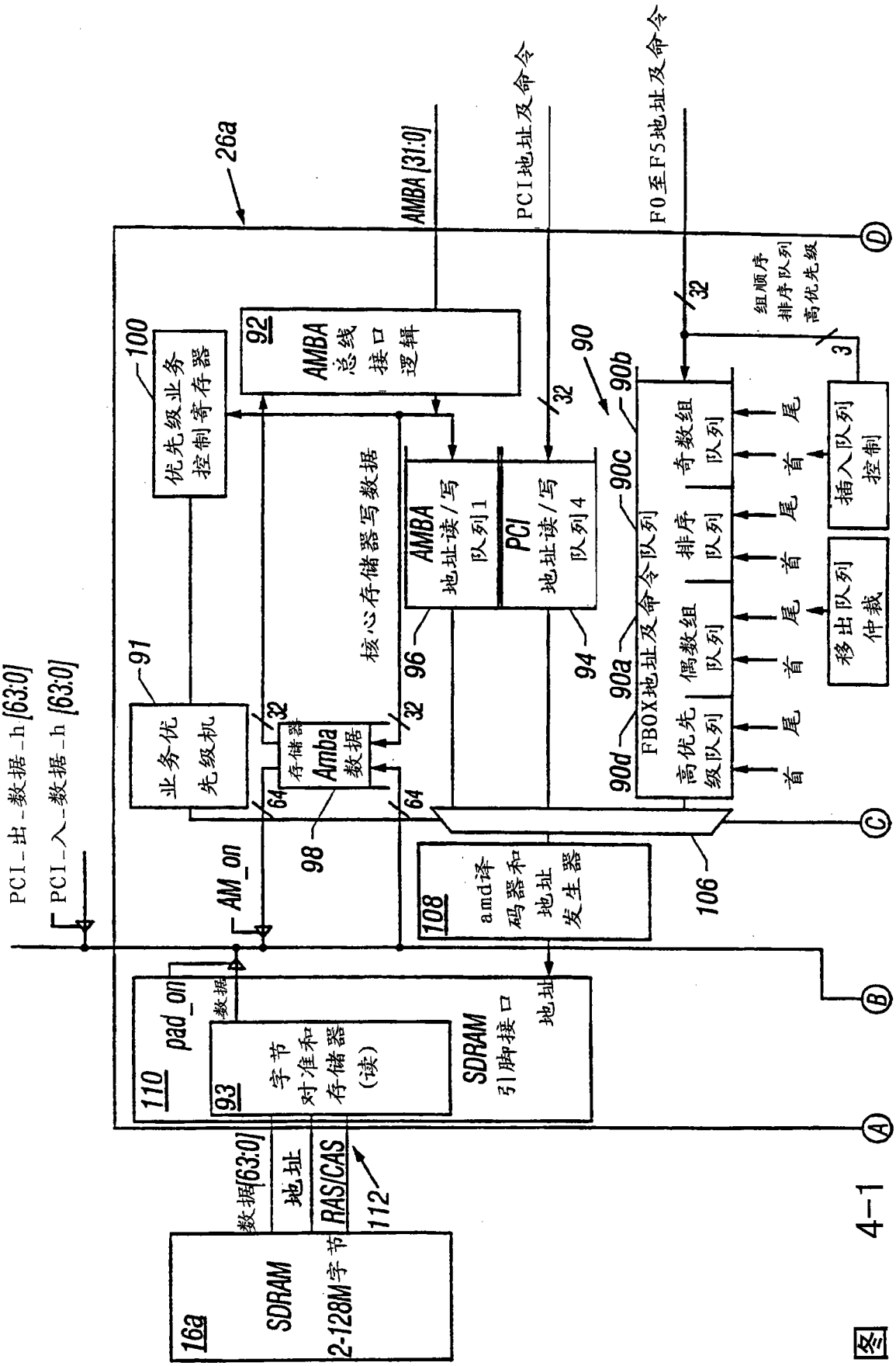

PCI_出_数据_h [63:0]
PCI_入_数据_h [63:0]
AM_on
26a
AMBA [31:0]
PCI地址及命令
F0至F5地址及命令
100
优先级业务控制寄存器
92
AMBA总线接口逻辑
91
业务优先级机
32
存储器
Amba数据
64
核心存储器写数据
96
AMBA地址读/写队列1
PCI地址读/写队列4
94
90
90b
90c
90a
90d
FBOX地址及命令队列
高优先级队列
偶数组队列
排序队列
奇数组队列
首尾
移出队列仲裁
插入队列控制
组顺序排序队列高优先级
3
98
108
amd译码器和地址发生器
106
110
pad_on
数据
93
字节对准和存储器(读)
SDRAM引脚接口
地址
数据[63:0]
地址
RAS/CAS
112
16a
SDRAM
2-128M字节
A
B
C
D

图 4-1

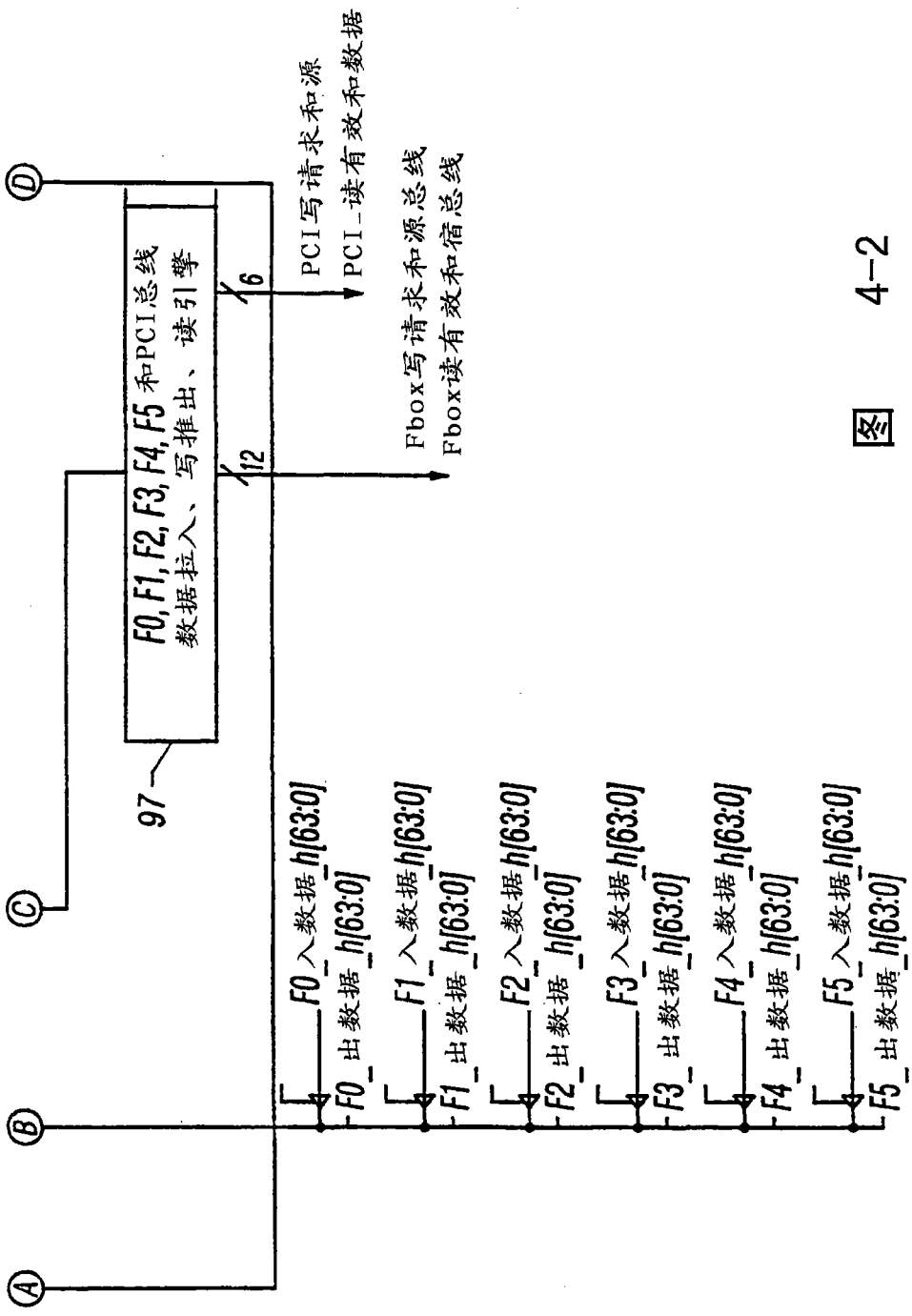

A
B
C
D
97
F0, F1, F2, F3, F4, F5 和PCI总线
数据拉入、写推出、读引擎
12
6
PCI写请求和源
PCI_读有效和数据
Fbox写请求和源总线
Fbox读有效和宿总线
F0_入数据_h[63:0]
F0_出数据_h[63:0]
F1_入数据_h[63:0]
F1_出数据_h[63:0]
F2_入数据_h[63:0]
F2_出数据_h[63:0]
F3_入数据_h[63:0]
F3_出数据_h[63:0]
F4_入数据_h[63:0]
F4_出数据_h[63:0]
F5_入数据_h[63:0]
F5_出数据_h[63:0]

图 4-2

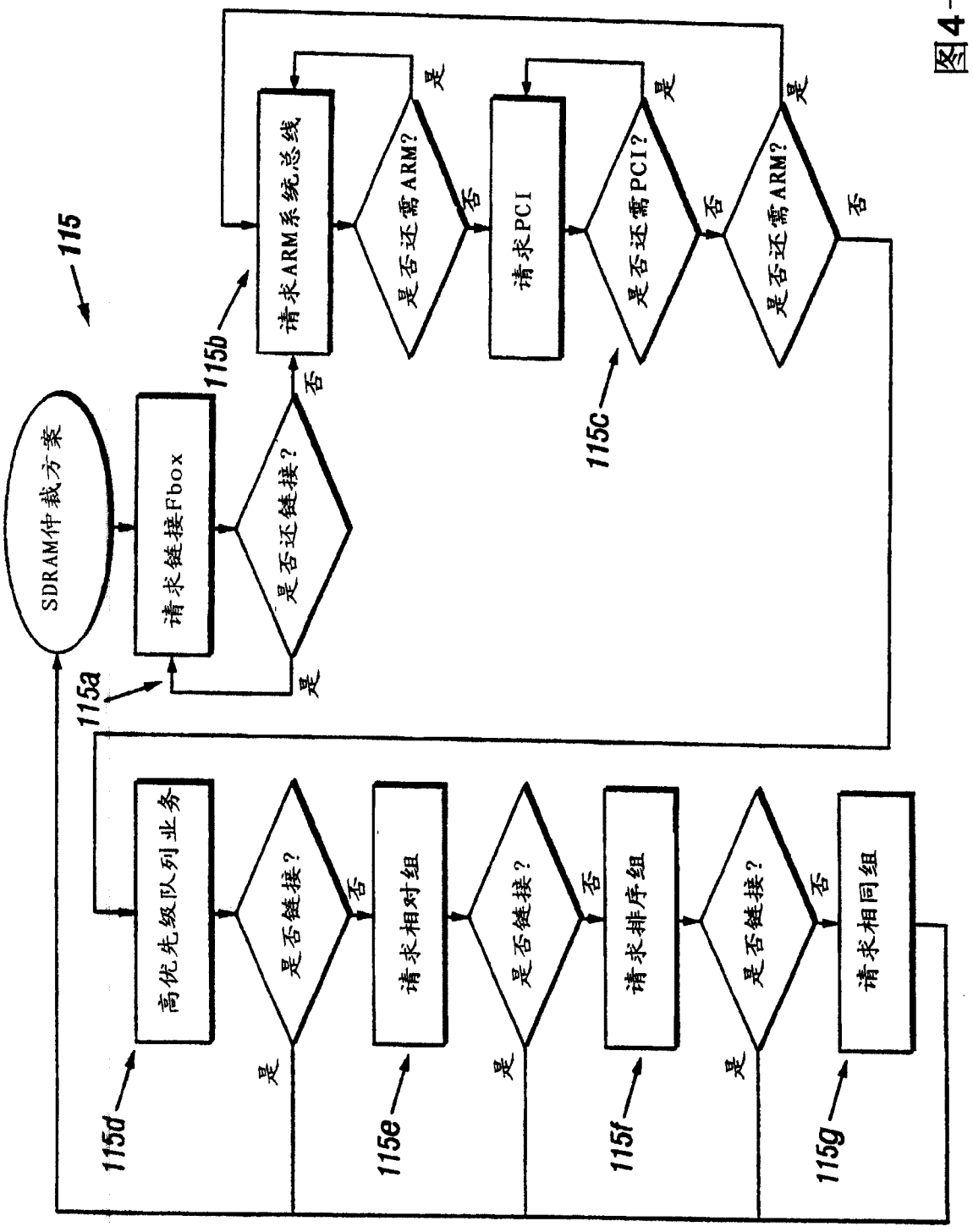

SDRAM仲裁方案
115
115a
请求链接Fbox
是否还链接?
是
否
115b
请求ARM系统总线
是否还需ARM?
是
否
115c
请求PCI
是否还需PCI?
是
否
是否还需ARM?
是
否
115d
高优先级队列业务
是否链接?
是
否
115e
请求相对组
是否链接?
是
否
115f
请求排序组
是否链接?
是
否
115g
请求相同组

图4—3

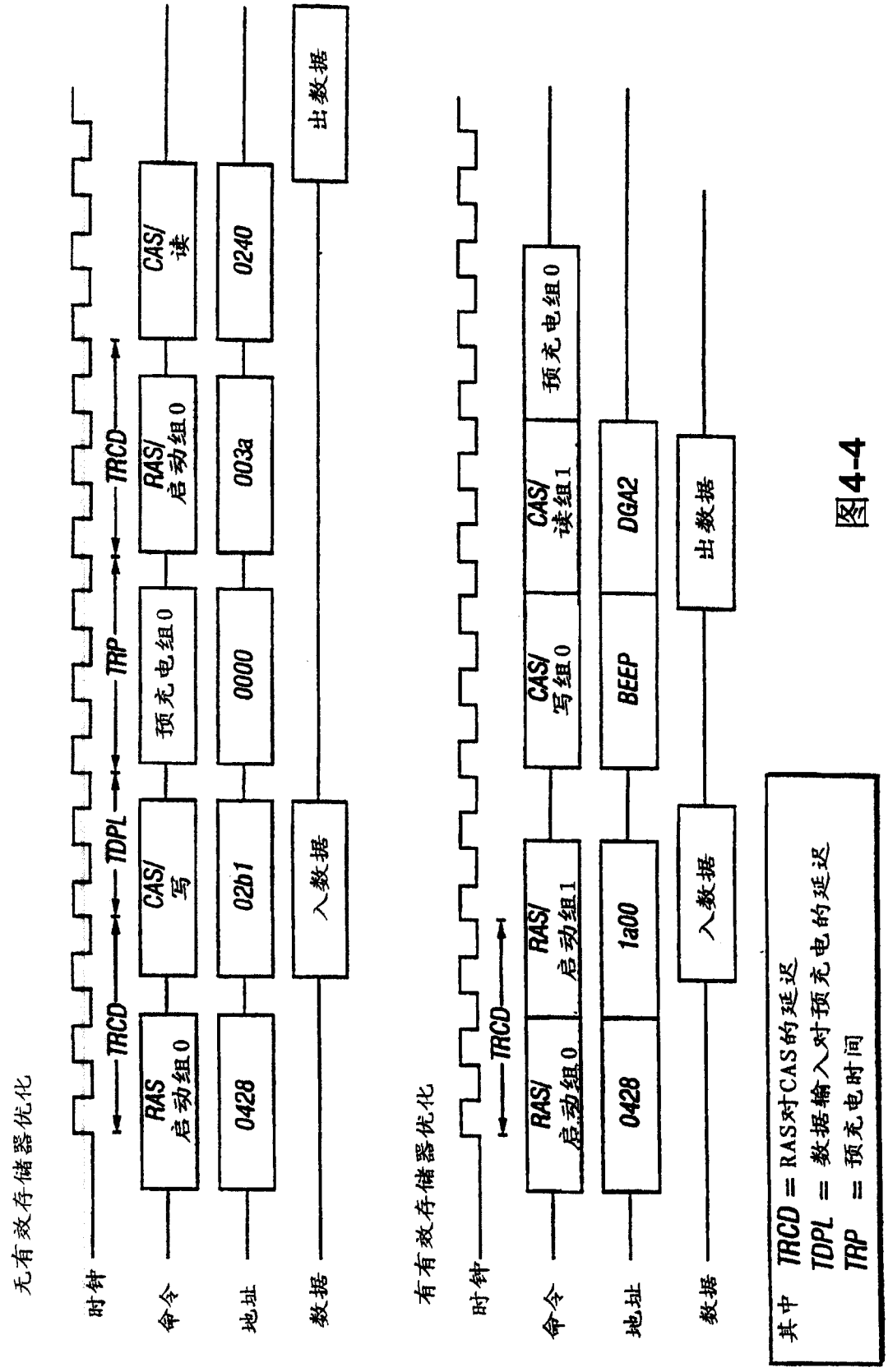

无有效存储器优化
时钟
命令
地址
数据
TRCD
TDPL
TRP
TRCD
RAS 启动组0
CAS/ 写
预充电组0
RAS/ 启动组0
CAS/ 读
0428
02b1
0000
003a
0240
入数据
出数据
有有效存储器优化
时钟
命令
地址
数据
TRCD
RAS/ 启动组0
RAS/ 启动组1
CAS/ 写组0
CAS/ 读组1
预充电组0
0428
1a00
BEEP
DGA2
入数据
出数据
其中 TRCD = RAS对CAS的延迟
TDPL = 数据输入对预充电的延迟
TRP = 预充电时间

图4-4

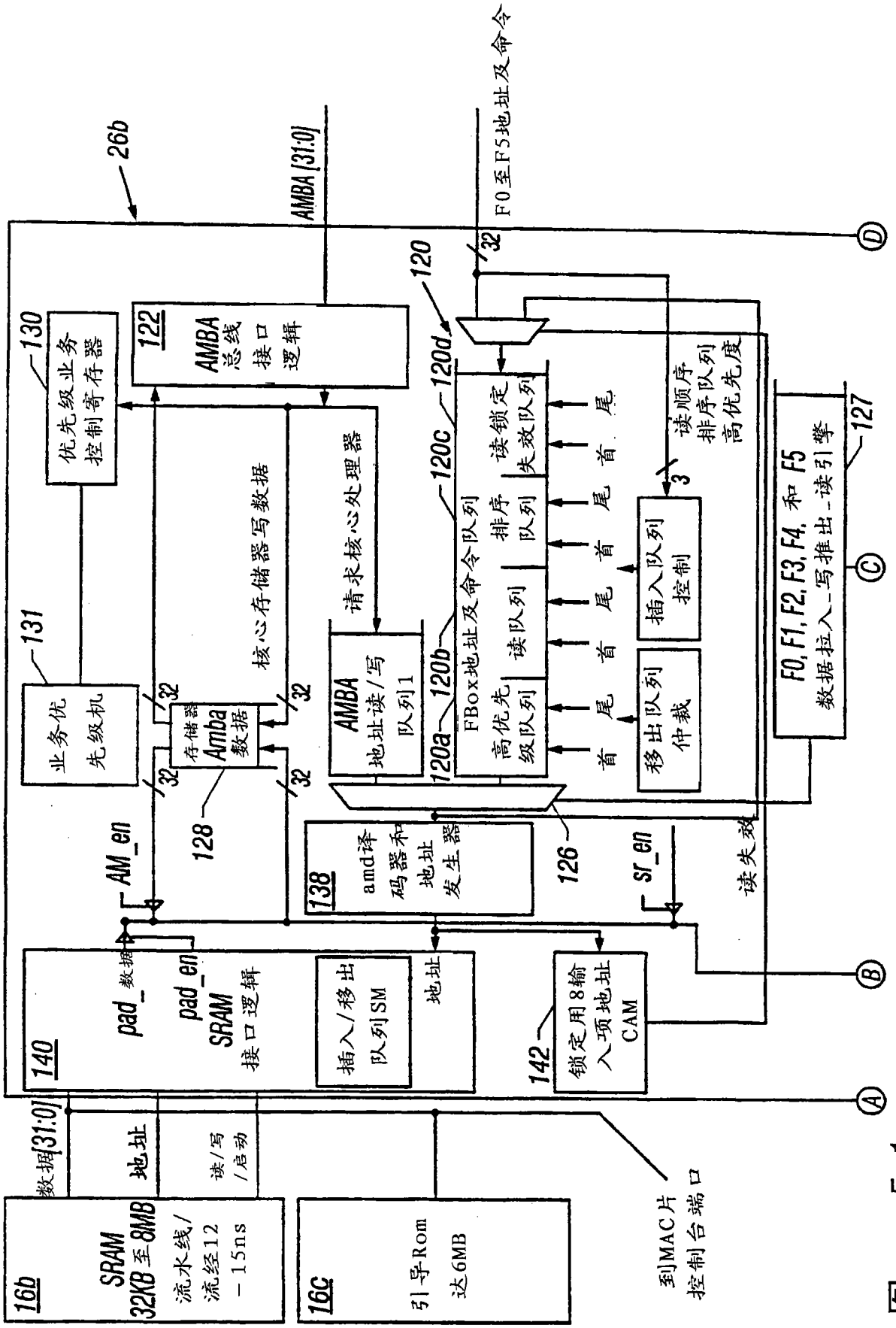

16b
SRAM
32KB至8MB
流水线/
流经12
-15ns
数据[31:0]
地址
读/写
/启动
16c
引导Rom
达6MB
到MAC片
控制台端口
140
pad_数据
pad_en
SRAM
接口逻辑
插入/移出
队列SM
地址
142
锁定用8输
入项地址
CAM
AM_en
SI_en
读失效
138
amd译
码器和
地址
发生器
126
128
存储器
Amba
数据
32
核心存储器写数据
请求核心处理器
AMBA
地址读/写
队列1
120a
120b
FBox地址及命令队列
高优先
级队列
读队列
排序
队列
120c
读锁定
失效队列
首尾
移出队列
仲裁
插入队列
控制
3
读顺序
排序队列
高优先度
120d
120
F0至F5地址及命令
F0, F1, F2, F3, F4, 和 F5
数据拉入-写推出-读引擎
127
131
业务优
先级机
130
优先级业务
控制寄存器
122
AMBA
总线
接口
逻辑
AMBA[31:0]
26b
A
B
C
D

图 5-1

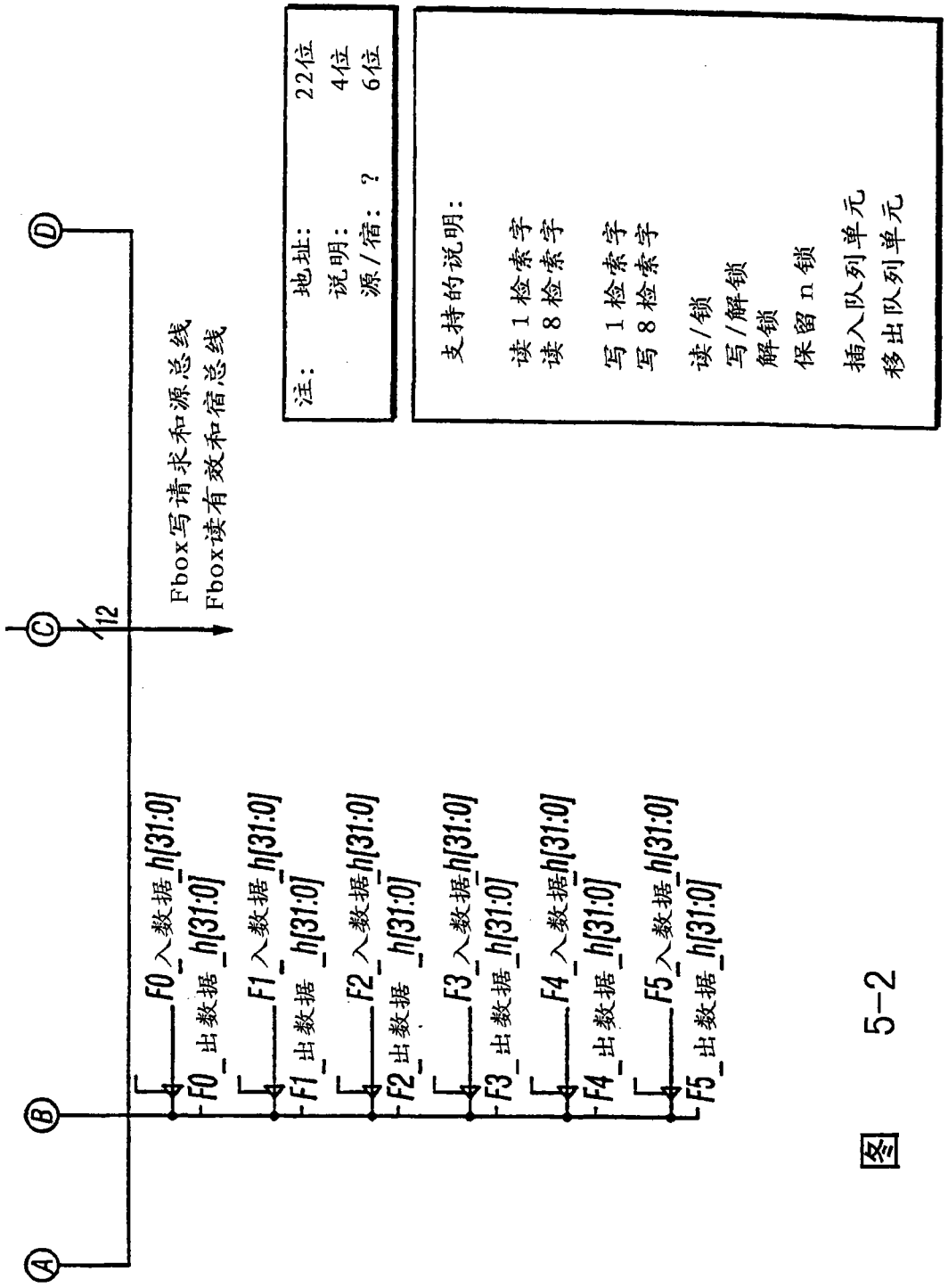

Fbox写请求和源总线
Fbox读有效和宿总线
12
F0_入数据_h[31:0]
F0_出数据_h[31:0]
F1_入数据_h[31:0]
F1_出数据_h[31:0]
F2_入数据_h[31:0]
F2_出数据_h[31:0]
F3_入数据_h[31:0]
F3_出数据_h[31:0]
F4_入数据_h[31:0]
F4_出数据_h[31:0]
F5_入数据_h[31:0]
F5_出数据_h[31:0]
注：
地址： 22位
说明： 4位
源/宿：？ 6位
支持的说明：
读 1 检索字
读 8 检索字
写 1 检索字
写 8 检索字
读/锁
写/解锁
解锁
保留 n 锁
插入队列单元
移出队列单元

图 5-2

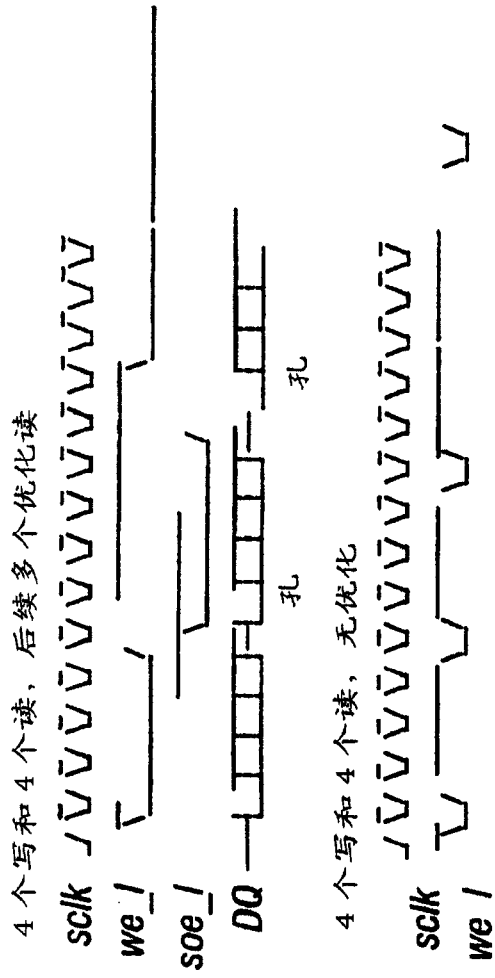

4个写和4个读，后续多个优化读
sclk
we_l
soe_l
DQ
孔
4个写和4个读，无优化
sclk
we_l
soe_l
DQ
孔 孔 孔 孔 孔
10周期对14

图5—3

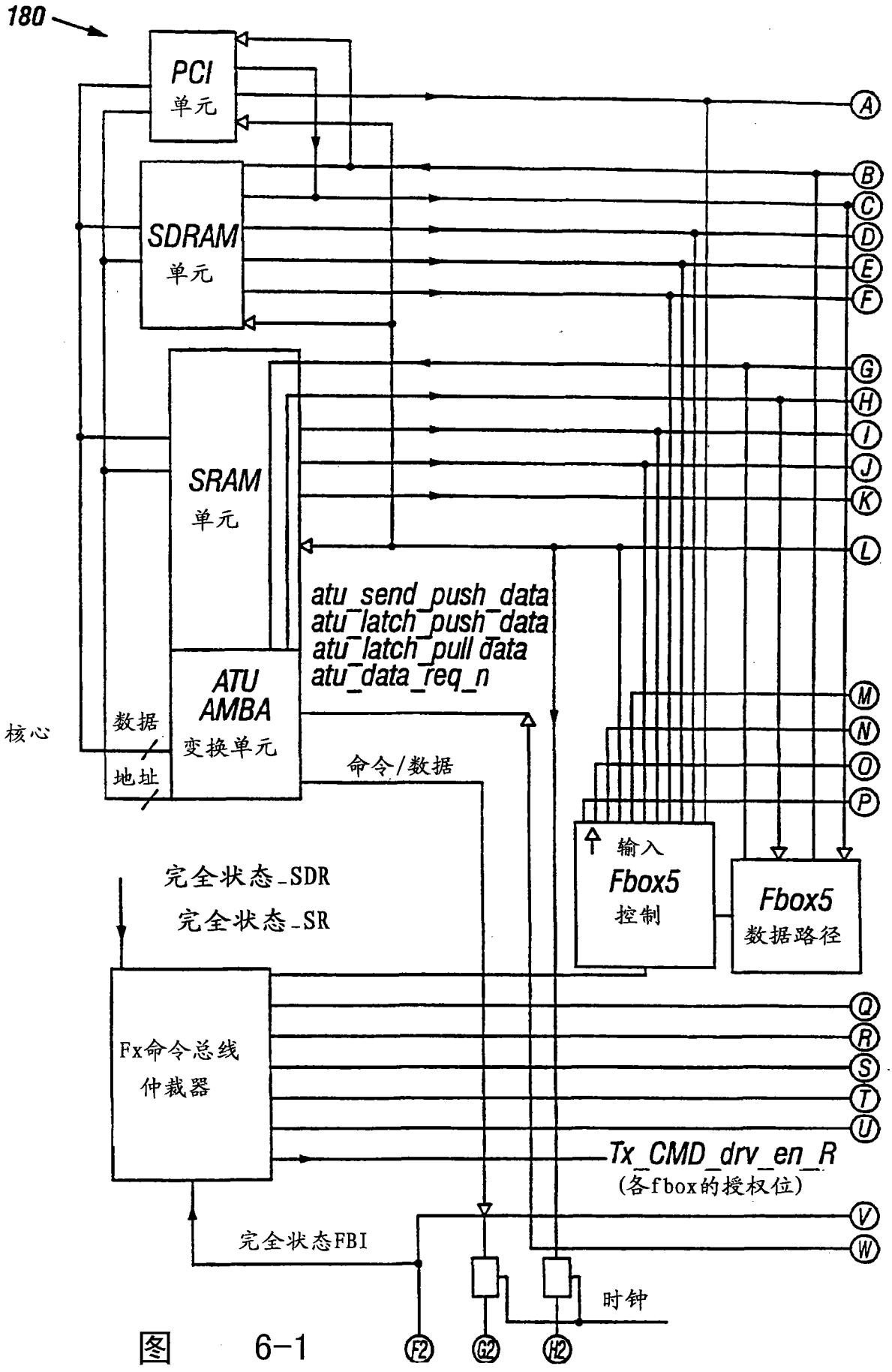

180
PCI
单元
SDRAM
单元
SRAM
单元
ATU
AMBA
变换单元
核心
数据
地址
atu_send_push_data
atu_latch_push_data
atu_latch_pull data
atu_data_req_n
命令/数据
完全状态_SDR
完全状态_SR
输入
Fbox5
控制
Fbox5
数据路径
Fx命令总线
仲裁器
Tx_CMD_drv_en_R
(各fbox的授权位)
完全状态FBI
时钟
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
F2
G2
H2

图 6-1

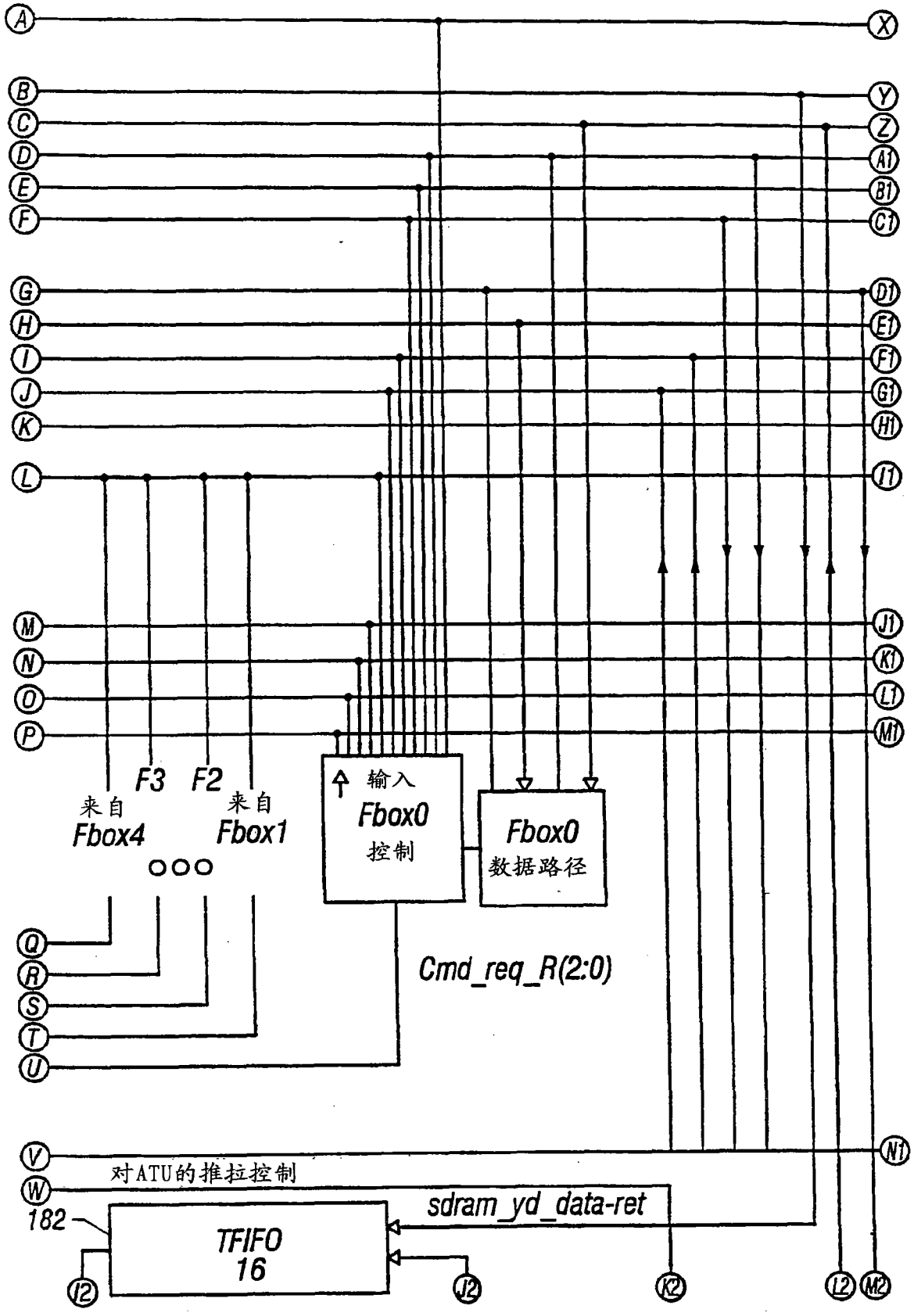

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
A1
B1
C1
D1
E1
F1
G1
H1
I1
J1
K1
L1
M1
N1
F3
F2
来自
Fbox4
来自
Fbox1
输入
Fbox0
控制
Fbox0
数据路径
Cmd_req_R(2:0)
对ATU的推拉控制
182
TFIFO
16
sdram_yd_data-ret
I2
J2
K2
L2
M2

图 6-2

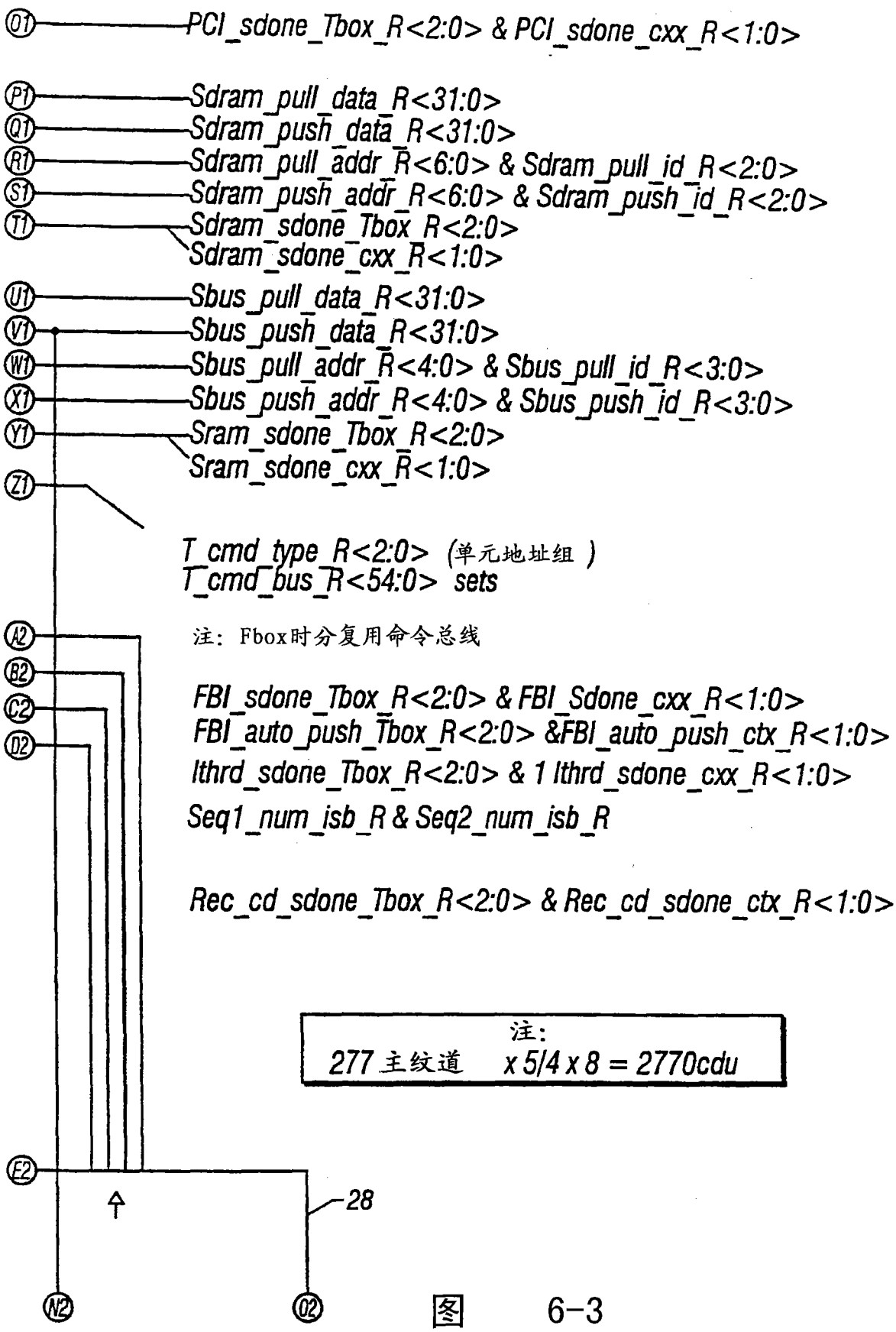
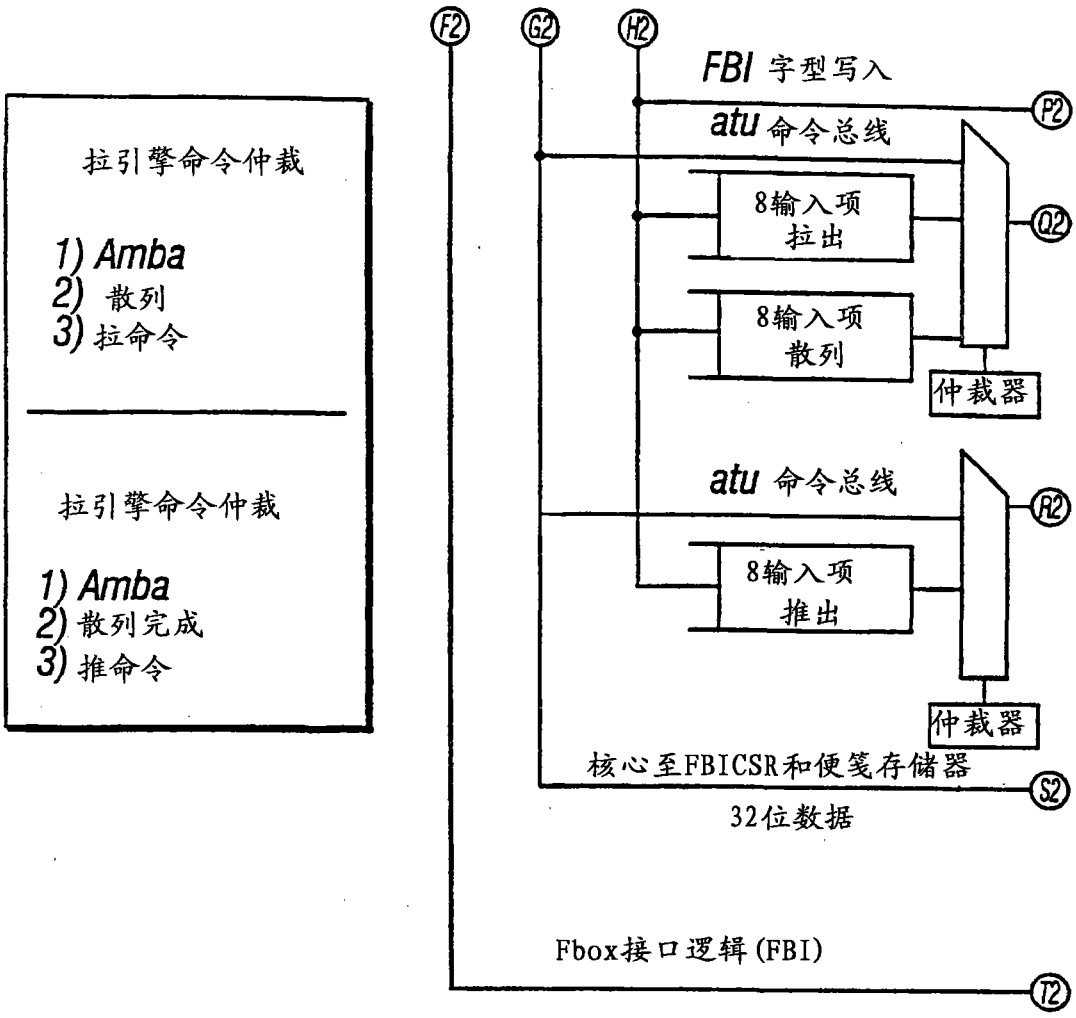

O1
PCI_sdone_Tbox_R<2:0> & PCI_sdone_cxx_R<1:0>
P1
Sdram_pull_data_R<31:0>
Q1
Sdram_push_data_R<31:0>
R1
Sdram_pull_addr_R<6:0> & Sdram_pull_id_R<2:0>
S1
Sdram_push_addr_R<6:0> & Sdram_push_id_R<2:0>
T1
Sdram_sdone_Tbox_R<2:0>
Sdram_sdone_cxx_R<1:0>
U1
Sbus_pull_data_R<31:0>
V1
Sbus_push_data_R<31:0>
W1
Sbus_pull_addr_R<4:0> & Sbus_pull_id_R<3:0>
X1
Sbus_push_addr_R<4:0> & Sbus_push_id_R<3:0>
Y1
Sram_sdone_Tbox_R<2:0>
Sram_sdone_cxx_R<1:0>
Z1
T_cmd_type_R<2:0> (单元地址组)
T_cmd_bus_R<54:0> sets
注：Fbox时分复用命令总线
A2
B2
C2
D2
FBI_sdone_Tbox_R<2:0> & FBI_Sdone_cxx_R<1:0>
FBI_auto_push_Tbox_R<2:0> &FBI_auto_push_ctx_R<1:0>
Ithrd_sdone_Tbox_R<2:0> & 1 Ithrd_sdone_cxx_R<1:0>
Seq1_num_isb_R & Seq2_num_isb_R
Rec_cd_sdone_Tbox_R<2:0> & Rec_cd_sdone_ctx_R<1:0>
注:
277 主纹道 x 5/4 x 8 = 2770cdu
E2
28
N2
O2

图 6-3



ATU注:

a) 核心至Fbox存储器:
用SRAM推数据总线

b) 核心至FBI寄存器:
用专用ATU/FBI命令/数据总线

c) 核心读Fbox寄存器:
用SRAM拉数据总线

d) 核心读FBI寄存器:
用sram推数据总线(使sram在sram推总线上对FBI呈现类似于另一Fbox)

Cmd_Req_R<2:0>
000 无
001 Sram 链接
010 SDR 链接
011 Sram
100 SDR
101 FBI
110 PCI
111

Tx_CMD_drv_en_R<1:0>
0 无
1 授权

图 6-4

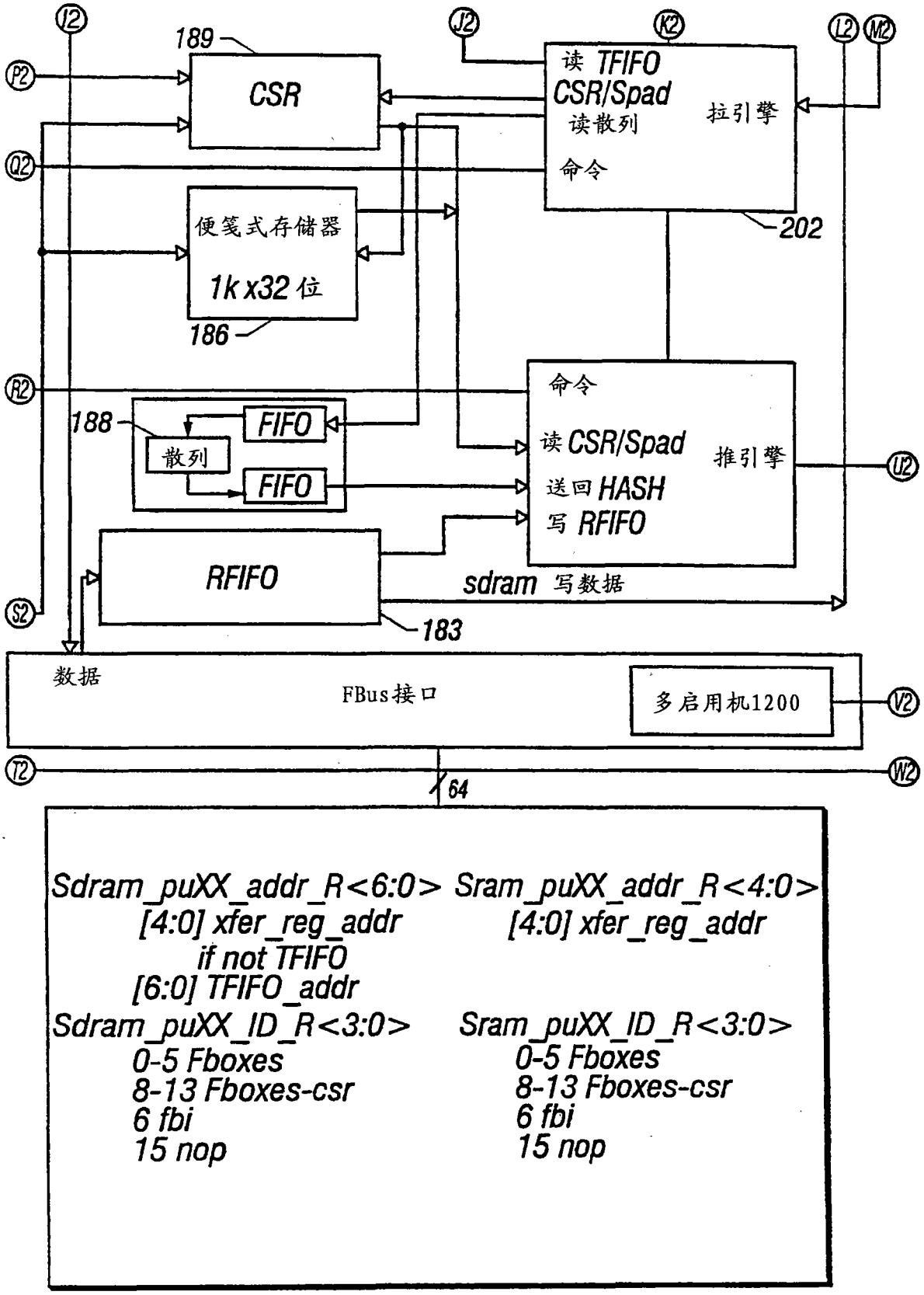

I2
189
CSR
P2
J2
K2
读 TFIFO
CSR/Spad
读散列
拉引擎
L2
M2
Q2
命令
202
便笺式存储器
1k x32 位
186
R2
命令
188
FIFO
散列
FIFO
读 CSR/Spad
推引擎
U2
送回 HASH
写 RFIFO
RFIFO
sdram 写数据
S2
183
数据
FBus接口
多启用机1200
V2
T2
W2
64
Sdram_puXX_addr_R<6:0>
[4:0] xfer_reg_addr
if not TFIFO
[6:0] TFIFO_addr
Sram_puXX_addr_R<4:0>
[4:0] xfer_reg_addr
Sdram_puXX_ID_R<3:0>
0-5 Fboxes
8-13 Fboxes-csr
6 fbi
15 nop
Sram_puXX_ID_R<3:0>
0-5 Fboxes
8-13 Fboxes-csr
6 fbi
15 nop

图 6-5

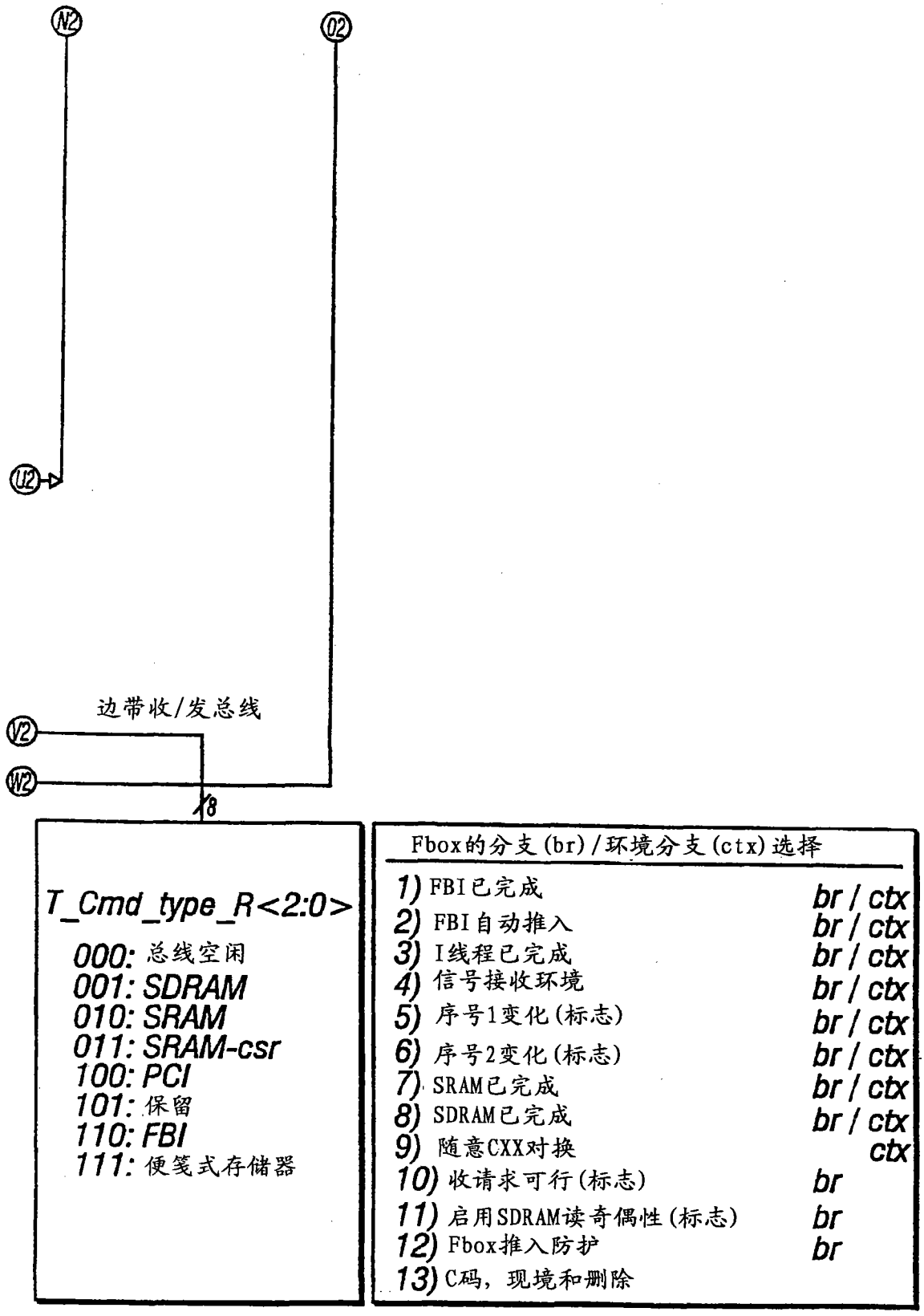

N2
O2
U2
边带收/发总线
V2
W2
18
T_Cmd_type_R<2:0>
000: 总线空闲
001: SDRAM
010: SRAM
011: SRAM-csr
100: PCI
101: 保留
110: FBI
111: 便笺式存储器
Fbox的分支(br)/环境分支(ctx)选择
1) FBI已完成 br / ctx
2) FBI自动推入 br / ctx
3) I线程已完成 br / ctx
4) 信号接收环境 br / ctx
5) 序号1变化(标志) br / ctx
6) 序号2变化(标志) br / ctx
7) SRAM已完成 br / ctx
8) SDRAM已完成 br / ctx
9) 随意CXX对换 ctx
10) 收请求可行(标志) br
11) 启用SDRAM读奇偶性(标志) br
12) Fbox推入防护 br
13) C码，现境和删除

图 6-6