[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/00

[12] 发明专利申请公开说明书

[21] 申请号 00815123.7

[43] 公开日 2003 年 3 月 12 日

[11] 公开号 CN 1402844A

[22] 申请日 2000.8.31 [21] 申请号 00815123.7
[30] 优先权
[32] 1999. 9. 1 [33] US [31] 60/151,961
[86] 国际申请 PCT/US00/23995 2000.8.31
[87] 国际公布 WO01/16703 英 2001.3.8
[85] 进入国家阶段日期 2002.4.28
[71] 申请人 英特尔公司
地址 美国加利福尼亚州
[72] 发明人 G·沃尔瑞奇 M·J·艾迪莱塔
W·威勒 D·伯恩斯坦因
D·胡伯

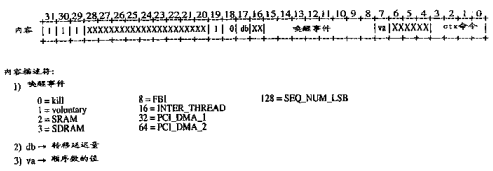
[74] 专利代理机构 上海专利商标事务所
代理人 孙敬国

权利要求书 3 页 说明书 24 页 附图 17 页

[54] 发明名称 用于多线程并行处理器的指令

[57] 摘要

本发明揭示了一种基于硬件的并行多线程处理器。处理器包括协调系统功能的通用处理器和支持多个硬件线程或上下文(CONTEXT)的多个微引擎。处理器还包括存储控制系统，它具有根据存储器调用是否针对偶数存储区或奇数存储区排序存储器的第一存储控制器，还具有根据存储器调用是读调用还是写调用优化存储器调用的第二存储控制器。还揭示了根据执行上下文(上下文描述符：content descriptor)切换的转移的指令。



ISSN1008-4274

1、一种计算机指令，其特征在于，包括

上下文交换指令，它将在指定的微引擎中当前运行的上下文交换到存储器，使另外的上下文在该微引擎中执行，并导致选择不同的上下文及相关的程序计数器。

2、如权利要求 1 所述的计算机指令，其特征在于，

当指定的信号被激活时，所述上下文交换指令唤醒被交换出去的上下文。

3、如权利要求 1 所述的计算机指令，其特征在于，

将所述信号指定为所述指令的参数，并指定事件的发生。

4、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“sram swap”，并且所述上下文交换指令交换出当前的上下文，直到收到线程的 SRAM 信号时唤醒它。

5、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“sram swap”，并且所述上下文交换指令交换出当前的上下文，直到收到线程的 SDRAM 信号时唤醒它。

6、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“FBI”，它交换出当前的上下文，并当收到指出已完成 FBI CSR，Scratchpad，TFIFO，或 RFIFO 操作的线程的 FBI 信号时唤醒它。

7、如权利要求 3 所述的计算机指令，其特征在于，

所述的参数指定为“seq_num1_change/seq_num2_change”，它交换出当前的上下文，并当顺序数的值改变时唤醒它。

8、如权利要求 3 所述的计算机指令，其特征在于，

所述的参数指定为“inter_thread”，它交换出当前的上下文，并当接收到线程的内线程信号时唤醒它。

9、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“Voluntary”，如果其他线程已准备好运行则交换出当前的上下文，并且如果该线程被交换出则在某个后续上下文判优点所述交换出的线程自动地再使能运行。

10、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“auto_push”，它交换出当前的上下文，并当 SRAM 传输

读寄存器数据自动地被 FBUS 接口压入时唤醒它。

11、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“start_receive”，它交换出当前的上下文，并当在接收 FIFO 中新的数据对于该线程用来处理有效时唤醒它。

12、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“kill”，它在对应该线程的适当的使能位并在 CTX_ENABLES 寄存器中置位以前，防止当前的上下文或线程再次执行。

13、如权利要求 3 所述的计算机指令，其特征在于，

所述参数指定为“PCI”，它交换出当前的上下文，并当 PCI 单元发信号，通知 DMA 传输已经完成时唤醒它。

14、如权利要求 3 所述的计算机指令，其特征在于，还包括

选项_标记(option_token)“defer one”，它规定在所述上下文被交换之前并在此调用后将执行一条指令。

15、一种操作处理器的方法，其特征在于，包括

计算指定的参数以确定一个执行上下文过程的状态，和

完成交换操作，以导致按照指定参数的值选择不同的上下文和相关的程序计算计数器。

16、如权利要求 15 所述的方法，其特征在于，

完成将当前在指定微引擎内运行的上下文交换到存储器，使另外上下文在该微引擎中执行。

17、如权利要求 15 所述的方法，其特征在于，

所述参数指定一个事件的发生。

18、如权利要求 15 所述的方法，其特征在于，

所述的参数指定为“sram swap”，并完成一个交换，该交换包括交换出当前上下文并当收到线程的 SRAM 信号时唤醒它。

19、如权利要求 15 所述的方法，其特征在于，

所述参数指定为“sram swap”，并完成一个交换，该交换包括交换出当前上下文并当收到线程的 SDRAM 信号时唤醒它。

20、如权利要求 15 所述的方法，其特征在于，

所述参数指定为“inter_thread”，它交换出当前的上下文，并当收到该线程的内线程信号时唤醒它。

21、如权利要求 15 所述的方法，其特征在于，还包括

选项_标志(optional_token)“defer one”，它规定在所述上下文被交换之前并在此调用后将执行一条指令。

22、一种能执行多个上下文的处理器，其特征在于，包括

寄存器堆栈，

用于每个执行的上下文的程序记数器，

连接到寄存器堆栈和存储上下文交换指令的程序控制存储的算术逻辑单元，它使处理器

计算指定的参数以确定执行的上下文过程的状态，和

完成交换操作，导致按照指定参数的值选择不同的上下文及相关的程序计数器，并且它保存老的程序计数器值。

23、如权利要求 22 所述的处理器，其特征在于，

当指安的信号被激活时，所述上下文交换指令唤醒被交换出的上下文。

24、一种计算机程序产品，驻留在计算机可读媒体上，用于使处理器完成由指令组成的功能，其特征在于，所述指令使处理器

计算指定的参数以确定执行上下文过程的状态，和

完成交换操作，使得按照该指定参数的值选择不同的上下文和相关的程序计数器。

25、如权利要求 24 所述的产品，其特征在于，

当指定的信号被激活时，处理器唤醒交换出去的上下文。

用于多线程并行处理器的指令

背景

本发明涉及计算机处理器的指令。

并行处理是在计算过程中并发事件信息处理的有效形式。与顺序处理相反，并行处理要求在一台计算机上同时执行多个程序。在并行处理器的范围，并行性意味着在同一时刻做一件以上的事情。不象所有任务在单个工作站上顺序完成的串行样式，或各任务在专门的工作站上完成的管线机器，对于并行处理提供多个工作站，每个工作站能完成所有任务。即，通常全部或者多个工作站同时且独立地在一个问题的同一个或共同的单元上工作。某些问题适合于应用并行处理来解决。

附图简述

图 1 表示使用基于硬件的多线程(multithreaded)处理器的一个通讯系统的方框图。

图 2 表示图 1 中基于硬件的多线程处理器的详细方框图。

图 3 表示使用在图 1 及图 2 中基于硬件的多线程处理器的微引擎(microengine)功能单元的方框图。

图 4 表示在图 3 中微引擎的管线的方框图。

图 5A 和图 5B 表示与指令有关的上下文(Content)的示例性格式图。

图 6 表示通用寄存器地址安排的方框图。

图 7 表示用于在基于硬件的多线程处理器中使用的增强带宽操作的存储控制器的方框图。

图 7A 表示在图 7 的 SDRAM 控制器中的判优策略的流程图。

图 7B 表示优化 SDRAM 控制器的优点的时序图。

图 8 表示在基于硬件的多线程处理器中使用的等待时间限止的操作的存储控制器的方框图。

图 8A 表示优化 SRAM 控制器的优点的时序图。

图 9 表示图 1 中处理器的通讯总线接口方框图。

描述

参考图 1，通讯系统 10 数据包括并行的基于硬件的多线程处理器 12。基于硬件的多线程处理器连接到如 PCI 的总线 14，存储器系统 16 和第二总线 18。系统 10 对于能分解成各并行子任务或功能的任务特别有用。具体说来，基于硬件的多线程处理器 12 对于面向带宽而非面向等待时间的任务是有用的。基于硬件的多线程处理器 12 具有多个微引擎 22，每个具有能同时独立地在一个任务上工作的多个硬件控制的线程。

基于硬件的多线程处理器 12 还包括一个中央控制器 20，它帮助加载对基于硬件的多线程处理器 12 的微码控制，并完成其他通用计算机类型的功能，如处理协议，例外处理，对数据包处理的额外支持，其中微引擎将数据包送出，进行如边界条件那样的更详细的处理。在实施例中，处理器 20 是基于 Strong Arm®(Arm 是英国 ARM 有限公司的商标)的结构。通用微处理器 20 具有操作系统。通过操作系统，处理器 20 能调用功能在微引擎 22a～22f 上操作。处理器 20 能使用任何支持的操作系统，最好是实时操作系统。对于作为实现 Strong Arm 的核心处理器，能够使用的操作系统如 Microsoft NT®。Real time，VXWorks 和□CUS，一个在因特网上可用的自由件(Freeware)操作系统。

基于硬件的多线程处理器 12 还包括多个功能微引擎 22a～22f。功能微引擎(微引擎)22a～22f 中每个保持多个硬件的程序计数器和与这些程序记数器有关的状态。实际上，虽然在任何时刻只有一个微引擎在操作，对应的多个线程组能在每个微引擎 22a～22f 上同时活动。

在实施例中，有 6 个微引擎 22a～22f，如图所示，每个微引擎 22a～22f 具有处理 4 个硬件线程的能力。6 个微引擎 22a～22f 带着包括存储器系统 16 和总线接口 24 和 28 的共享资源操作。存储器系统 16 包括同步动态随机存储器(SDRAM)控制器 26a 和静态随机存储器(SRAM)控制器 26b。SDRAM 存储器 16a 和 SDRAM 控制器 26a 通常用于处理大量的数据，如处理从网络数据包来的网络有效载荷(Payload)的处理。SRAM 控制器 26b 和 SRAM 存储器 16b 用于对低等待时间，快速访问任务的网络实现，如访问查找表，用于核心处理器 20 的存储器等。

6 个微引擎 22a～22f 根据数据的特性访问 SDRAM16a 或 SRAM16b。因此，低等待时间低带宽的数据存入 SRAM 并从中取出，而等待时间不重要的较高带

宽的数据存入 SDRAM 并从中取出。微引擎 22a～22f 能执行到 SDRAM 控制器 26a 或 SRAM 控制器 26b 的存储器调用。

硬件多线程的优点能能过 SRAM 或 SDRAM 存储器的访问解释。作为一个例子，从一个微引擎通过 Thread-0 请求的 SRAM 访问引起 SRAM 控制器 26b 起动对 SRAM 存储器 16b 的访问。SRAM 控制器对 SRAM 总线判优，访问 SRAM16b，从 SRAM16b 取数据，并将数据返回到请求的微引擎 22a～22f。在 SRAM 访问期间，如果如 22a 那样的微引擎只有单个线程能操作，那个微引擎在数据从 SRAM 返回前休眠。通过使用每个微引擎 22a～22f 中的硬件内容(Contert)交换，硬件内容交换使得带单独程序计数器的其他上下文在那同一个微引擎中执行。因此，在第一线程，如 Thread_0，在等待读数据返回的同时，另外的线程，如 Thread_1，能运行。在执行期间，Thread_1 可能访问 SDRAM 存储器 16a。在 Thread_1 在 SDRAM 单元上操作，且 Thread_0 在 SRADM 上操作的同时，一个新的线程，如 Thrad_2 现能在微引擎 22a 中操作。Thrad_2 能操作一定时间，直至它需要访问存储器或完成某些长等待时间的操作，如作出对总线接口的访问。因此，处理器 12 能同时具有总线操作，SRAM 操作和 SDRAM 操作，所有均由一个微引擎操作或完成，并还有一个线程可用于在数据通道中处理更多的工作。

硬件内容交换还与任务的完成同步。例如，两个线程能到达同一个共享资源，如 SRAM。如 FBUS 接口 28，SRAM 控制器 26a，和 SDRAM 控制器 26b 这些分别的功能单元的每一个，当它们完成一个从一个微引擎线程上下文不断的请求的任务时，回报一个标志，通知一个操作的完成。当微引擎接收到该标志时，该微引擎决定打开哪个线程。

基于硬件的多线程处理器 12 的应用一个例子是用作一个网络处理器。作为一个网络处理器，基于硬件的多线程处理器 12 接口到网络设备，如媒体访问控制设备，例如 10/100BaseT Octal MAC13a 或 Gigabit Ethernet 设备 13b。通常，作为一个网络处理器。基于硬件的多线程处理器 12 能接口到接收/发送大量数据的任何类型的通讯设备或接口。在网络应用中运行的通讯系统 10 能从设备 13a，13b 接收多个网络数据包并以并行方式处理那些数据包。采用基于硬件的多线程处理器 12，能互相独立的处理每个网络数据包。

使用处理器 12 的另一个例子是用于页面图象(Postscript)处理器的打印机引擎，或作为用于如 RAID 盘存储器那样的存储子系统。另一个应用是作为

匹配引擎来匹配买方和卖方之间的订单。在系统 10 上能完成这些和其他并行类型的任务。

处理器 12 包括将处理器连接到第二总线 18 的一个总线接口 28。在实施例中，总线接口 28 将处理器 12 连接到所谓的 FBUS18 CFIFO 总线)。FBUS 接口 28 负责控制并将处理器接口到 FBUS18。FBUS18 是 64 位宽的 FIFO 总线，用于接口到媒体访问控制器(MAC)设备。

处理器 12 包括如 PCI 总线接口 24 的第二接口，它将插在 PCI 总线上的系统部件连接到处理器 12。PCI 总线提供到如 SDRAM 存储器 16a 那样的存储器 16 的高速数据通道 24a。通过该通道，数据能借助直接存储器访问(DMA)从 SDRAM 16a 穿过 PCI 总线 14 快速地移动。基于硬件的多线程处理器 12 支持图象传输。基于硬件的多线程处理器 12 能使用多个 DMA 通道，所以如果 DMA 传输的一个目标忙，另一个 DMA 通道能担当 PCI 总线将信息提交到另一个目标，以保持高的处理器的效率。此外，PCI 总线接口 24 支持目标和基本的操作。目标操作是那样的操作，其中在总线 14 上的从属设备通过从属于目标操作的读和写访问 SDRAM。在基本操作中，处理器 20 直接将数据发送到 PCI 接口或直接从中接收数据。

每个功能单元连结到一个或多个内部总线。如下面所述，内部总线是双 32 位总线(即一根总线用于读，一根总线用于写)。基于硬件的多线程处理器 12 还构造成使得在处理器 12 中内部总线的带宽之和超过连结到处理器 12 的外部总线的带宽。处理器 12 包括内部核心处理器总线 32，如 ASB 总线(先进系统总线)，它将处理器核心 20 连接到存储控制器 26a，26c 并连接到下述的 ASB 翻译点 30。ASB 总线是与 Strong Arm 处理核心一起使用的所谓的 AMBA 总线的子集。处理器 12 还包括专用总线 34，它将微引擎单元连续到 SRAM 控制器 26b，ASB 翻译器 30 和 FBUS 接口 28。存储器总线 38 将存储控制器 26a，26b 连接到总线接口 24 和 28，以及包括用于自引导操作的闪存随机存储器的存储器系统 16 等。

参考图 2，每个微引擎 22a～22f 包括一个判优器，它检查标志以确定要操作的可用线程。从任何微引擎 22a～22f 来的任何线程能访问 SDRAM 控制器 26a，SDRAM 控制器 26b 或 FBUS 接口 28。存储控制器 26a 和 26b 的每一个包括多个队列，存储未完成的存储器调用请求。此队列保持存储器调用的次序或者安排存储器调用以优化存储器带宽。例如，如果 thread_0(thread_线程)与 Thread_1

没有任何依赖关系，thread_0 和 thread_1 没有理由不能不按次序地完成它们的存储器调用。微引擎 22a～22f 发出存储器调用请求到存储控制器 26a 和 26b。微引擎 22a～22f 用足够的存储器调用操作充满存储器子系统 26a 和 26b，使得存储器子系统 26a 和 26b 成为处理器 12 操作的瓶颈。

如果存储器子系统用本质上独立的存储器请求充满，处理器 12 能实现存储器调用排序。存储调用排序能改善可得到的存储器带宽。如下所述，存储器调用排序减少了在访问 SRAM 中发生的停顿时间或泡沫。对于对 SRAM 的存储器调用，将在信号线上的电流方向在读和写之间切换产生一个泡沫或停顿时间，等待在将 SRAM 16b 连接到 SRAM 控制器 26b 的导线上的电流稳定。

即，驱动在总线上电流的驱动器需要在改变状态的前稳定下来。因此，一个读接着一个写的重复周期能破坏峰值带宽。存储器调用排序允许处理器 12 组织对存储器的调用，使得一长串的读后面接着一长串的写。这样能用于使管线中的停顿时间最少，有效地达到接近于最大可用带宽。调用排序帮助维持并行的硬件内容线程。在 SDRAM，调用排序使隐去了从一个存储区到另一个存储区的预加载。具体说来，如果存储器系统 16b 被组织成一个奇数存储区和一个偶数存储区，在处理器在奇数存储区运行的同时，存储控制器能开始预加载偶数存储区。如果存储器调用在奇数和偶数存储区之间交替，预加载是可能的。通过排列存储器调用的次序交替访问相反的存储区，处理器 12 改善了 SDRAM 的带宽。此外，也可使用另外的优化。例如，可以使用将可以合并的操作在存储器访问前合并的合并优化，通过检查地址使已打开的存储器页不再重新打开的打开页优化，如下所述的链接，和刷新机构。

FBUS 接口 28 支持对每个 MAC 设备支持的发送的接收标志，以及指出何时需要服务的中断标志，FBUS 接口 28 还包括控制器 28a，它完成对从 FBUS18 进入的数据包的头标处理。控制器 28a 提取数据包的头标并完成在 SRAM 中的可编微程序源/目标/协议/散列的查找(用于地址光滑)。如果散列未能成功地分解，将数据包的头标送到处理器核心 20 用于进一步处理。FBUS 接口 28 支持下列内部数据事务，

FBUS 单元(共享总线 SRAM)到/从微引擎

FBUS 单元(通过专用总线)从 SDRAM 单元写

FBUS 单元(通过 MBUS)读入 SDRAM

FBUS18 是标准的工业总线并包括如 64 位宽的数据总线以及对地址和读/

写控制的边带控制。FBUS 接口 28 提供使用一系列输入和输出 FIFO29a～29b 输入大量数据的能力。从 FIFO 29a～29b 微引擎 22a～22f 取出数据，或命令 SDRAM 控制器 26b 将从总线 18 上的设备来到接收 FIFO 的数据传到 FBUS 接口 28。该数据能通过存储控制器 26a，借助直接存储器访问，送到 SDRAM 存储器 16a。类似地，微引擎能将数据从 SDRAM26a 移到接口 28，经过 FBUS 接口 28 移出 FBUS18。

在微引擎之中数据功能是分布式的。到 SRAM26a，SDRAM 26b 和 FBUS28 的连接是通过命令请求。例如，一个命令请求能将数据从位于微引擎 22a 中的寄存器移动共享资源，如 SDRAM 位置，SRAM 位置，闪存存储器或某些 MAC 地址。命令被发出到每个功能单元的共享资源。然而，共享资源不需要保持数据的局部缓冲。相反，共享资源访问位于微引擎内部的分布式数据。这使微引擎 22a～22f 局部访问数据，而不是判优访问总线去冒险竞争总线。以此特征，对于微引擎 22a～22f 内部数据的等待是 0 个周期停顿。

连结如存储控制器 26a 和 26b 那样的共享资源的数据总线，如 ASB 总线 30，SRAM 总线 34 和 SDRAM 总线 38，有足够的带宽，使得没有瓶颈。因此，为了避免瓶颈，处理器 12 具有带宽要求，为每个功能单元至少提供两倍内部总线的最大带宽。作为例子，SDRAM 能在 83MHz 运行 64 们宽的总线。SRAM 数据总线能具有分别的读写总线，如能有在 166MHz 下运行的 32 位宽的读总线和 166MHz 下 32 位宽的写总线。实际上是在 166MHz 运行 64 位，是 SDRAM 带宽的两倍。

核心处理器 20 也能访问共享资源。核心处理器通过总线 32 具有到 SDRAM 控制器 26a，到总线接口 24 和到 SRAM 控制器 26b 的直接通讯。然而，为了访问微引擎 22a～22f 和位于任何微引擎 22a～22f 的传输寄器，核心处理器 20 通过在总线上的 ASB 翻译器访问微引擎 22a～22f。ASB 翻译器 30 在物理上能驻留在 FBUS 接口 28，但在逻辑上是单纯的。ASB 翻译器 30 完成在 FBUS 微引擎舆寄存器位置和核心处理器地址(即 ASB 总线)之间的地址翻译，使得核心处理器 20 能访问属于微引擎 22a～22c 有的寄存器。

虽然微引擎能如下所述那样使用寄存器组交换数据，还提供便笺式存储器 27，以允许微引擎将数据写出到该存储器为其他微引擎读取。便笺式存储器 27 连接到总线 34。

处理器核心 20 包括一个 RISC 核心 50，它实现在 5 个阶段管理中在一周期中完成一个或两个操作数的单个循环移位，处理器核心还提供乘法支持和 32

位滚动移位支持。此 RISC 核心 50 是标准的 Strong Arm®结构，但为了性能的原因以 5 阶段管线实现。处理器核心 20 还包括 16 千字节指定高速缓存 52 和 8 千字节数据高速缓存 54 和预取流缓存 56。核心处理器 20 与存储器写及取指令并行地完成算术运算。核心处理器 20 通过 ARM 确定的 ASB 总线与其他功能单无接口。ASB 总线是 32 位双向总线 32。

微引擎

参考图 3，示出微引擎 22a～2f 中的示例性的一个，如微引擎 22f。微引擎包括一个控制存储 70，在实施例中它包括 1024 个 32 位字的 RAM。该 RAM 存储微程序。微程序可通过核心处理器 20 加载。微引擎 22f 还包括控制器逻辑 72。控制器逻辑包括一个指令解码器 73 和程序计数器(PC)单元 72a～72d。以硬件形式保持 4 个微程序计数器 72a～72d。微引擎 22f 还包括上下文事件切换逻辑 74。上下文事件逻辑 74 接收从每个共享资源，如 SRAM 26a，SDRAM 26b，或处理器核心 20，控制和状态寄存器等，接收消息（如 SEQ_#_EVENT_RESPONSE;FBI_EVENT_RESPONSE;SRAM_EVENT_RESPONSE; SDRAM_EVENT_PESPONSE;和 ASB-EVENT-RESPONSE)。这些消息提供有关请求的功能是否已完成的信息，根据由一个线程请求的功能是否完成以及是否已发完成信号，该线程需要等待完成信号，而且如果该线程能够操作，则该线程被放置在可用线程表(未示出)。微引擎 22f 能具有最大的可用的线程，如 4 个。

除了对执行线程是局部的事件信号以外，微引擎 22 使用全局的信号状态。带着信号状态，执行的线程能对所有微引擎 22 广播信号状态。接收请求有效(Receive Request Available)信号，在微引擎中任何所有的线程能接这些信号状态转移。能使用这些信号状态确定资源的可用性或是否资源准备服务。

上下文事件逻辑 74 具有对 4 个线程的判优。在实施例中，判优是一个循环机构。也能使用其他技术，包括优先级队列或加权的公平队列。微引擎 22f 还包括一个执行框(EBOX)数据通道 76，它包括算术逻辑单元 76a 和通用寄存器组 76b。算术逻辑单元 76a 完成算术和逻辑功能以及移位功能。寄存器组 76b 具有相当大量的通用寄存器。如图 6 所描述，在此实施中，在第一存储区 Bank A 有 64 个通用寄存器，在第 2 存储区 Bank B 中也有 64 个。如下面描述，通用寄存器分窗，使得它们能相对地和绝对地编址。

微引擎 22f 还包括一个写传输寄存器堆栈 78 和一个读传输寄存器堆栈 80。

这些寄存器也分窗，所以它们可以相对地和绝对地编址。写宁夏存器堆栈 78 是写到资源的数据放置之处。类似地，读寄存器堆栈 80 用于从共享资源返回的数据。在数据到达之后或同时，从如 SRAM 控制器 26a，SDRAM 控制器 26b 或核心处理器 20 那样的有关共享资源提供一个事件信号给上下文事件判优器 74，它随后提醒线程数据可用或已发出。传输寄存器存储区 78 和 80 均通过数据通道连结到执行框(EBOX)76。在一个实施中，读传输寄存器有 64 个寄存器且写传输寄存器也有 64 个寄存器。

参考图 4，微引擎数据通道保持 5 阶段微管线 82。该管线包括微指令查找 82a，形成寄存器文件地址 82b，从寄存器文件读操作数 82c，ALU，移位或比较操作 82d，和结果写回到寄存器 82e。通过提供写回数据旁路到 ALU/移位单元，并通过假设寄存器作为寄存器文件实现(而不是 RAM)，微引擎能同时完成寄存器文件的读和写，这就完全隐去了写操作。

SDRAM 接口 26a 提供一个返回信号到请求读的微引擎，指出在读请求中是否发生奇偶校验错误。当微引敬使用任何返回数据时，微引擎的微码负责检查 SDRAM 的读奇偶校验标志。在检查此标志时，如果它被置位，根据它转移的指令清除它。只有当 SDRAM 能够校验且 SDRAM 是奇偶校验保护的，才能发出奇偶校验标志。微引擎和 PCI 单元是仅有被告知奇偶校验错误的请求者。因此，如果处理器核心 20 或 FIFO 需要奇偶校验保护，微引擎在请求中予以协助。微引擎 22a～22f 支持条件转移。当转移决定是由以前的微控制指令置位的条件码的结果，发生最坏情况的条件转移等待(不包括跳转)。等待时间示于下面表 1 中，

	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 是预转移微字(n1 置位条件码)

cb 是条件转移

bx 是转移后的微码

XX 是放弃的微码

如表 1 所示，直到周期 4n1 的条件码被置位，且作出转移决定(在此情况导致在周期与查找转移路径)。微引擎招致 2 个周期的转移等待时间的损失，因为在转移路径开始用操作 b1 弃满管线以前它必须在管线中放弃操作 n2 和 n3(紧接着转移后的 2 个微字)。如果转移未发生，没有微字放弃，且执行正常延续。微引擎具有若干机构来减少或消除有效的转移等待时间。

微引擎支持延迟(deferred)转移。延迟转移进在传移发生以后在转移生效以前，微引擎允许 1 或 2 个微字(即转移生效是在时间上延迟)。因此，如果能找到有用的工作来填补在转移微字以后的浪费的周期，则转移的等待时间能被隐去。下面示出 1 个周期的延迟转移，其中 n2 被允许在 cb 之后但在 b1 之前执行，

	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 周期延迟的转移，其中 n2 和 n3 均允许在转移到 b1 发生前完成，注意，2 周期转移延迟仅当条件码在转移以前的微字上设置时才允许。

	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计算，如果在转移以前，作出转移决定的条件码被设置成 2 个或更多的微字，则能消除 1 个周期的等待时间，因为转移决定能在 1 个周期以前作出，

```
                    | 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|
```

在此例中，n1 设置条件码且 n2 不设置条件码，因此，在周期 4(而非周期 5)能作出转移决定，以消除 1 个周期的转移等待时间。在下例中，1 个周期转移延迟和较早设置条件码结合起来完全隐去转移等待时间。

在 1 周期延迟转移前的 2 个周期设置条件码(cc's)，

```
                    | 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|
```

在条件码不能提前置位(即在转移前在微字中置位)的情况，微引擎支持转移推测，试图减少 1 个周期的转移等待时间。借助“推测”转移路径或顺序路径，微引擎在确切知道执行什么路径前一个周期预取推测的路径。如果推测正确，如下所述消取了 1 个周期的转移等待时间。

推测发生转移/转移发生

```
                    | 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|
```

如果推测转移发生的微码不正确，微引擎仍然只浪费 1 个周期，

推测发生转移/转移未发生

周期结束前转移地址是未知的。

| 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

微引擎支持各种标准类型的 ALU 指令，包括逻辑和算术操作，它们在一个或二个操作数上的 ALU 操作并将结果放入目标寄存器。ALU 按照操作的结果更新所有的 ALU 条件码。在上下文交换期间条件码的值丢失。

参考图 5A，示出上下文转移指令 BR=CTX，BR! =CTX。上下文转移指令导致处理器，如微引擎 22f，根据当前执行指令是否为指定的上下文数转移到在指定标号处的指令。如图 5A 所示，上下文转移指令从转移掩码字段何时等于“8”或“9”而确定。上下文转移指令能具有下述格式，

br=ctx[ctx,label#], option_token(选项_标记)

br! =ctx[ctx,label#], option_token(选项_标记)

字段 label#是对应于指令地址的符号标号。字段 ctx 是上下文数。在实施例中，有效的 ctx 值是 0，1，2，3。上下文转移指令可以具有选项_标记。选项标记“defer one”(延迟 1)导致微引擎在完成转移指令前执行跟在此指令后的一条指令。

如果该上下文是指定数，指令 br=ctx 转移，且如果该上下文不是指定数，指令 br!=ctx 转移。

参考图 5B，上下文交换指令转移的特殊形式，它导致选择不同的上下文(且与 PC 相关)。上下文切换或交换也引入某些转移等待时间。考虑下列上下文切

| 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

换，

	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				

在上述情况通过 n2 的执行并由于正确地推测了转移方向隐去了 2 周期的转移等待时间。如果微码推测不正确，对地推测转移发生带 1 周期延迟转移/转移未发生，如下所述仍遭受 1 周期的转移等待时间。

	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				

如果微码不正确地推测转移未发生，则管线在正常平静的情况下顺序流动。如果微码不正确地推测转移未发生，如下所述对推测转移未发生/转移发生情况，微引擎再次遭受 1 周期的没有结果的执行。

	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 nx 是预转移微字(n1 设置条件码)

cb 是条件转移

bx 是转移后微字

XX 是放弃的微码

在跳转指令情况，招致 3 个额外周期的等待时间，因为跳转在 ALU 阶段的

	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

然而，当微码推测转移未发生时，等待时间损失被不同地分布，对推测转移未发生/转移未发生，如下所示没有浪费的周期。

	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

然而对于推测转移未发生/转移发生，有 2 个浪费周期。

	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

微引擎能将转移推测与 1 周期转移延迟相结合，以进一步改善结果。对于推测转移发生带 1 周期延迟转移/转移发生是，

其中 ox 是老的上下文流

br 是在老的上下文中的转移微字

ca 是上下文的重判优(引起上下文切换)

nx 是新的上下文流

XX 是被放弃的微字

在上下文切换中，“br”微字被丢弃，以避免由于保存正确的老的上下文 PC 引起的控制和时序的复杂化。

按转移前在微字上设置的 ALU 条年码操作的条件转移能选择 0，1 或 2 周期的转移延迟方式。在按照微字操作的条件转移能选择 0 或 1 周期转移延迟方式以前，条件码设置 2 或更多的微字，所有其他转移(包括上下文重新判优)能选择 0 或 1 周期的转移延迟方式。结构能设计成使得在以前转移，跳传或上下文判优微字的转移延迟窗中的一个上下文判优微字成为非法选项。即，在某些实施例中，在转移过程中不允许发生上下文交换，因为如上所述，保存老的上下文程序计数器(PC)可能过分的复杂化。结构也能设计成使得在以前的转移，跳转或上下文判优微字的转移延迟窗中的转移成为非法，以避免复杂的和可能未预见的转移行为。

上下文交换指令 CT_ARB 将当前在指定微引擎中运行的上下文交换出去到存储器中，让其他上下文在那个微引擎中执行。当指定的信号激活时，上下文交换指令 CT_ARB 还唤醒被交换出的上下文。上下文交换指令的格式是

ctx_arb[parameter], option_token

“parameter”字段能具有若干值中的一个。如果该参数据定为“sram swap”，该上下文交换指令将交换出当前的上下文并当收到线程的 SRAM 信号时唤醒它。如果该参数指定为“sram swap”，该上下文交换指令将交换出当前的上下文并当收到线程的 SDRAM 信号时唤醒它。该参数也能指定为”FBI“。并交换出当前的上下文，并当收到线程的 FBI 信号时唤醒它。FBI 信号指出 FBI CSR，Scratchpad(便笺存储器)，TFIFO 或 RFIFO 操作已完成。

该参数能指定成“seq_num1_chamge/seq_num2_chamge”，它交换出当前的上下文并当顺序数的值改变时唤醒它。该参数能指定为“inter_thread”，它交换出当前的上下文，并当收到线程的内线程(interthread)信号时唤醒它，或指定为“Voluntary”，如果另外的线程准备运行，交换出当前的上下文，否则不作交换。如果该线程被交换出，在某个后续的上下文判化点它被自动地

重新使能运行。该参数能是“auto_push”，它交换出当前的上下文并当 SRAM 传输读寄存器数据自动地被 FBUS 接口压入时唤醒它，该参数或者是“start)receive”，它交换出当前的上下文并当在接收 FIFO 中的新的数据包数据可用于为此线程处理时唤醒它。

该参数也能是“kill”，它防止在对该线程的适当使能位在 CTX_ENABLES 寄存器中置位以前再次执行当前的上下文或线程，也能是“PCI”，它交换出当前的上下文且在 PCI 单元发生 DMA 传输已经完成的信号时唤醒它。

上下文交换指令 CTX_ARB 能具有下列 option_token(选项_标记)，defer one(延迟 1)，它规定在上下文被交换以前，在此调用后执行 1 条指令。

每个微引擎 22a～22f 支持 4 个上下文的多线程执行。其一个理由是在一个线程刚发出存储器调用并在做其他工作以前必须等待该调用完成的情况允许另一个线程开始执行。为了维持微引擎的有效的硬件执行，此行为是重要的，因为存储器等待时间是显著的。换言之，如果只支持单线程，微引擎将空等许多周期，等待调用返回，因而减少了整个计算产生率。多线程执行通过在若干线程间完成有用的独立工作允许一个微引擎隐去存储器等待时间。提供两个同步机构，以便允许一个线程发出 SRAM 或 SDRAM 调用，并随后同步到调用完成的时间点。

一个机构是即时同步(Immediate Synchronization)。在即时同步时，微引擎发出调用并立即交换出上下文。当对应的调用完成，该上下文收到信号。一旦收到信号，当发生上下文交换(context_swap)事件并轮到它适行时，该上下文将交换回来执行。因此，从单个上下文指令流动观点，在发出存储器调用以后的微字直到调用完成以前不执行。

第二机构是延迟同步(Delayed synchronization)。在延迟同步中，微引擎发出该调用，并随后继续执行某些其他与该调回无关的有用工作。若干时间以后，有必要在另外的工作完成以前，将线程的执行流同步到发生调用的完成。在这点上执行上下文交换指令，它或者交换出当前的线程，并在调用完成时的以后某时刻将其交换回来，或者因为调用已经完成，继续执行当前线程。使用两个不同的发信号方案实现延迟同步。

如果存储器调用与传输寄存器有关，当对应的传输寄存器的有效位置或清除时产生触发该线程的信号。例如，将数据放入传输寄存器 A 的 SRAM 读在对 A 的有效位被置位时将收到信号。如果存储器调用与传输 FIFO 或接收 FIFO 有关

而不是一个传输寄存器，则当在 SDRAM 控制器 26a 中完成调用时产生信号。每个上下文只有一个信号状态被保持在微引擎调度器中，因此在此方案中只有一个未完成的信号存在。

至少有两个普通的操作算法，由此能设计微控制器的微程序。一个是以单个线程的执行等待时间为代价优化整个微控制器的计算产生率和整个存储器带宽。当系统具有多个微引擎执行多个线程，每个微引擎操作互不联系的数据数据包时，此算法有意义。

第二个是以整个微引擎的计算产出量的整个存储器带宽为代价优化微引擎执行等待时间。此算法能涉及执行具有实时限制的线程，那是这样的限制，它规定某个工作绝对必须在某指定时刻做完。那样的限制需要将单线程执行的优化赋予高于如存储器带宽或整个计算产生量等其他考虑的优先权。实时线程隐含着只执行一个线程的单个微引擎。将不处理多线程，因为目的是允许单个实时线程改可能快地执行-多线程的执行将阻碍此能力。

在发出存储器调用命令和上下文切换方面这两种算法的编码内格大不相同。在实时情况，目的是尽可能快地发出尽可能多的存储器调用，以便使由这些调用招致的存储器等待时间最小。尽可能早地发出尽可能多的调用以后，目的是尽可能使诸微引擎以与这些调用并行的方工完成尽可能多的计算。对应于实时优化的计算流程是，

1)发出存储器调用 1

2)发出存储器调用 2

3)发出存储器调用 3

4) 完成与存储器调用 1，2，和 3 无关的工作

5)同步到存储器调用 1 的完成

6)完成与存储器调用 1 有关但与存储器调用 2 和 3 无关的工作

7)根据以前的工作发出任何新的存储器调用

8)同步到存储器调用 2 的完成

9)完成与存储器调用 1 和 2 有关但与存储器调用 3 无关的工作

10)根据以前的工作发出任何新的存储器调用

11)同步到存储器调用 3 的完成

12)完成与所有 3 个调用的完成有关的工作

13) 根据以前的工作发出新的存储器调用。

相反，对产生量和带宽的优化采取不用的方法。对于微引擎计算产出量及整个存储器带宽的优化，在单个线程执行等待时间方面给于较少考虑。为此，目的是在对每个线程的整个微程序中等间隔地存储器调用，这将提供对 SRAMt SDRAM 控制器的均匀的存储器调用流，并使 1 个线程永远可用的概率最大，以便隐去在另外线程被交换出时的存储器等待时间。

寄存器文件地址类型

参考图 6，存去的两个寄存器地址空间是局部可访问的寄存器和可由所有微引擎访问的全局可访问寄存器。通用寄存器(GPR)作为两个分别的存储区(A 存储工和 B 存储区)实现，它们的地址是逐字交替，使 A 存储区寄存器具有 lsb=0，而 B 存储区寄存器有 lsb=1(lsb 是最低位)。每个存储区在其存储区由能完成对两个不同字的同时读和写。

在整个存储区 A 和 B，寄存器 76b 也组织成各 32 个寄存器的 4 个窗 76b0-76b3，它们对每个线程可相对编址。因此，thread_0 在 77a(寄存器 0)处找到其寄存器 0，thread_1 在 77b(寄存器 32)处找到其寄存器 0，therad_2 在 77c(寄存器 64)处找到其寄存器 0，thread_3 在 77d(寄存器 96)处找到其寄存器 0。支持相对编址，所以多个线程能使用完全相同的控制存储和位置而访问不同的寄存器窗并完成不同的功能。只要在微引擎 22f 中使用双口 RAM，采用寄存器窗口编址和存储区编址提供必要的读写宽。

这些分窗的寄存器从上下文切换到上下文切换不必要保存数据，从而消除了上下文交换文件或堆栈的正常压入和弹出操作。对于从一个上下文改变到另一个上下文，这里的上下文切换具有 0 周期的有效负载。相对寄存器编址在整个通用寄存器组的地址宽度上将寄存器存储区分成窗。相对编址允许相对于窗的起点访问任何窗。在此结构中也支持绝对地址，其中通过提供确切的寄存器地址，任何一个绝对寄存器能被任何线程访问。

根据极据微字的格式通用寄存器 78 的编址能以 2 种方工发生。两种方式是绝对编址和相对编址。在绝对方式中，寄存器地址的编址直接以 7 位源字段(a6～a0 或 b6～b0)指定，

```
          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
```

寄存器地址直接以 8 位目标字段(d7～d0)指定，

```
          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
```

如果<a6:a5>=1,1; <b6,b5>=1,1，或<d7:d6>=1,1，则较低位被解释为与上下文相关的地址字段(如下所述)。当在 A、B 绝对字段中规定非相所的 A 或 B 的源地址时，只有 SRAM/ASB 和 SDRAM 地址空间的较低部分能被编址。实际上，读绝对 SRAM/SDRAM 设备具有有效的地址空间，但是因为此限止不应用于目标字段，写 SRAM/SDRAM 仍然使用全地址空间。

在相对方工中，指定地址的编址是在上下文空间中的偏移量，由 5 位源字段(a4～a0 或 b4～b0)确定，

```
                7  6  5  4  3  2  1  0
              +---+---+---+---+---+---+---+---+
A GPR:        |a4|0|上下文|a3|a2|a1|a0|        a4=0
B GPR:        |b4|1|上下文|b3|b2|b1|b0|        b4=0
SRAM/ASB:|ab4|0|ab3|上下文|b2|b1|ab0|   ab4=1, ab3=0
SDRAM:        |ab4|0|ab3|上下文|b2|b1|ab0|   ab4=1, ab3=1
```

或由 6 位目标字段(d5～d0)确定，

```
                7  6  5  4  3  2  1  0
              +---+---+---+---+---+---+---+---+
A GPR:        |d5|d4|上下文|d3|d2|d1|d0|   d5=0, d4=0
B GPR:        |d5|d4|上下文|d3|d2|d1|d0|   d5=0, d4=1
SRAM/ASB:  |d5|d4|d3|上下文|d2|d1|d0|   d5=1, d4=0, d3=0
```

SDRAM: |d5|d4|d3|上下文|d2|d1|d0| d5=1,d4=0,d3=1

如果<de:d4>=1,1，则目标地址不确定一个有效寄存器地址，因此没有目标操作数写回。

下列寄存器是从微引擎和存储控制器全局可访问的，

散列单元寄存器

便笺存储器和公用寄存器

接收 FIFO 及接收状态 FIFO

发送 FIFO

发送控制 FIFO

微引擎不是中断驱动的。每个微流执行到完成，然后根据由在处理器 12 中其他的设备发信号的状态选择新的流。

参考图 7，SDRAM 存储器 26a 数据包括存储调用请求队列 90，从各个微引擎 22a～22f 来的存储调用请求到达那里。存储控制器 26a 包括一个判优器 91，它选择下一个微引擎调用请求到任何一个功能单元。如果一个微引擎是提供一个调用请求，该调用请求将通过地址和命令队列 90，进入 SDRAM 控制器 26a。如果调用请求是称为“optimized MEM bit-优化的存储器位”的位置位，进入的调用请求将被排序到偶数存储区队列 90a 或奇数存储区队列 90b。如果存储器调用请求不具有存储器优化位置位，默认地进入到命令队列 90c。SDRAM 控制器 26 是 FBUS 接口 28，核心处理器 20 和 PCI 接口 24 所共享的资源。SDRAM 控制器 26 还维持一状态机用于完成 READ-MODIFY-WRITE(读-修改-写)原子操作。SDRAM 控制器 26 也完成字节对齐用于从 SDRAM 来的数据请求。

命令队列 90c 保持从各微引擎来的调用请求的次序。对于一系列奇数与偶数存储区调用，只有完成一系列到奇数及偶数存储区的存储区调用时才需要返回信号。如果微引擎 22f 将存储器调用排序到奇数存储器和偶数存储区调用，且存储区之一如偶数存储区，在奇数存储区前被排出存储器调用，但信号恰肯定最后是偶数存储区调用，虽然奇数存储区调用尚未提供服务，存储控制器 26a 可以想象地发信号给微引擎，通知存储器请求已完成。这种现象就引起一个相干(conherency)问题。为了避免此情况，提供命令队列 90c，允许一个微引擎有多个未完成的存储器调用，只有其最后一个存储器调用需要发完成信号。

SDRAM 控制器 26a 还饭知一个高优先级队列 90d。在高优先级队列 90d 中从一个微引擎进入的存储器调用直接进入到高优先级队列，并以比其他队列中

的存储器调用更高的优先级操作。偶数存储区队列 90a，奇数存储区队列 90b，命令队列 90c，高优先级队列 90d，所有这些队列在单个 RAM 结构中实现，它在逻辑上分成 4 个不同的窗，每个窗具有其自己的头指针及尾指针。因为充满及排出操作仅是单个输入和单个输出，它们能放在同一个 RAM 结构中以增加 RAM 结构的密度。

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 将核心处理器 92 接口到 SDRAM 控制器 26a。ASB 总线是包括 32 位数据通道和 28 位地址通道的总线。通过 MEM ASB 数据设备 98，如一个缓冲器，数据存入到存储器或从存储器取出。MEM ASB 数据设备 98 是用于写数据的队列。如果有数据经过 ASB 接口 92 从核心处理器 20 来，该数据能存入 MEM ASB 设备 98，并随后从 MEM ASB 设备 98 通过 SDRAM 接口 110 传到 SDRAM 存储器 16a。虽然未显示，对读也有同样的队列结构。SDRAM 控制器 26a 还包括一个引擎 97，从微引擎及 PCI 总线弹出数据。

另外的队列包括 PCI 地址队列 94 和保存数个请求的 ASB 读/写队列 96。存储器请求经过多路复用器 106 被送到 SDRAM 接口 110。多路复用器 106 由 SDRAM 判优器 91 控制，后者检测每个队列的满度和请求的状态，并由此根据存在优先级服务控制寄存器 100 中的一个可编程值决定优先级。

一旦控制多路复用器 106 选定一个存储器调用请求，该存储器调用请求被送到解码器 108，在那里解码并产生地址。解码的地址送到 SDRAM 接口 110，在那里被分解成行和列的地址选通，以访问 SDRAM 16a，并经数据线 16a 写或读数据，将数据送到总线 112。在一实施中，总线 112 实际上是两个分别的总线而不是单根总线，分别的总线包括一根连结到分布式微引擎 22a～22f 的读总线和一根连接到分布工微引擎 22a～22f 的写总线。

SDRAM 控制器 26a 的一个特征是当一个存储器调用存入队列 90，除了优化的 MEM 位能被置位，还有一个“chaining bit-链结位”。链结位在置位时允许对连续的存储器调用作专门处理。如前提到，判优器 12 控制选中哪个微引擎来通过命令总线(commander bus)将存储器调用请求提供到队列 90(图 7)。链接位的确定将控制判优器选择以前请求该总线的功能单元，因为链接位的置位指出该微引擎发出一个链接请求。

当链接位置位时，连续存储器调用被收在队列 90 中。那些连续存储器调用通常存入命令队列 90c，因为连续存储器调用是从单个线程来的多个存储器

调用。为了提供同步，存储控制器 26a 只需要在链接的存储器调用做完的终点发出信号。然而，在一个优化的存储器链接中(如当优化的 EME 位及链接位被置位)，存储器调用能进入不同的存储区，并在另一个存储区完全排出以前可能在一个存储区上发出信号“done-完成”，因此破坏了相干性。因此，控制器 110 使用链接位维持从当前的队列的存储器调用。

参考图 7A，示出在 SDRAM 控制器 26a 中判优策略的流程表示。判优策略有利于链接的微引擎存储器请求。过程 115 通过检查链接的微引擎存储器调用请求 115a 开始。过程 115 停留在链接请求处，直到链接位被清除。过程检查 ASB 总线请求 115b，然后是 PCI 总线请求 115c，高优先级队列服务 115d，相反的存储区请求 115e，命令队列请求 115f，和同一存储区请求 115g。链接请求是完全地操作，而 115d～115d 以循环方式操作。只有当操作 115a～115d 完全退出时过程处理 115e～115g。当以前的 SDRAM 存储器请求已将链接位置位时，才是链接的微引擎存储器调用请求。当链接位置位时，判优引擎简单地再次操作同一队列，直到链接位被清除。由于当 ASB 在等待状态时在 strong arm 核心上有严重的性能损失，ASB 比 PCI 有较高的优先级。由于 PCI 的等待时间要求，PCI 比微引擎有更高的优先级。但是对其他总线，判优的优先级是不同的。

如图 7B 所示，示出没有活动的存储器优化和带有存储器优化通常的存储器时序。可以看到，使用活动的存储器优化使得总线的使用最大，因此隐去了在物理的 SDRAM 设备中内在的等待时间。在比例中，非优化的访问占用 14 个周期而优化的访问占用 7 个周期。

参考图 8，示出对 SDAM 的存储控制器 26b。存储控制器 26b 包括夺址和命令队列 120。虽然存储控制器 26a(图 7)对根据奇数和偶数的存储区划分的存储器优化有一个队列，存储控制器 26b 根据存储器操作的类型，存储控制器 26b 根据存储器操作的类型，即读或写，进行优化。地址和命令队列 120 包括一个高优先级队列 120a，一个读队列 120b，它是 SRAM 实现的主要存储器调用功能，以及一个命令队列 120c，通常包括所有到 SRAM 的写和未经优化的读。

虽然未示出，地址和命令队列 120 也能包括一个写队列。

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 将核心处理器 20 接口到 SRAM 控制器 26b。ASB 总线是包括 32 位数据通道和 28 位地址通道的总线。数据经过 MEM ASB 数据设备，如缓冲器，存入存储器或从中取出。MEM ASB 数据设备 128 是对写数据的队列。如果存在从

核心处理器 20 经过 ASB 接口 122 的进入数据，读数据能存入 MEM ASB 设备 128 并随后从 MEM ASB 设备 128 经过 SRAM 接口 140 到 SRAM 存储器 16b。虽然未示出，对读能有同样的队列结构。SRAM 控制器 26b 还包括一个引擎 127，将数据从微引擎及 PCI 总线弹出。

存储器请求经过比路复用器 126 送到 SRAM 接口 140。多路复用器 126 由 SRAM 判优器 131 换制，后者检测每个队列的满度和请求的状态，并由此根据存储在优先级服务控制寄存器 130 的一个可编程值确定优先级。一旦对多路复用器 126 的控制选择了一个存储器调用请求，该存储器调用请求被送到解码器 138，在那里解码并产生地址，SRAM 单元保持对存储器映射芯片外 SRAM(Menory Mapped off chip SRAM)和扩展 ROM(Expansion ROM)的控制。SRAM 控制器 26b 能够编址如 16 兆字节，如 8 兆字节映射到 SRAM 16b，8 兆字为特殊功能保留，包括由闪存随机存储器 16c 组成的自引导空间，对 MAC 设备 13a，13b 的控制台端口访问和对有关(RWON)计数器的访问。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(通过专用总线)到/从 SRAM

核心处理器(通过 ASB 总线)到/从 SRAM

SRAM 控制器 26b 完成存储器调用排序以使在从 SRAM 接口 140 到存储器 16b 的管线中的延迟(泡沫)最小。SRAM 控制器 26b 根据读功能进行存储器调用排序。泡沫可以是 1 或 2 个周期，取决于使用的存储器设备的类型。

SRAM 控制器 26b 包括一个锁查找设备(Lock Lookup device)142，它是用于查找读锁(read lock)的 8 个条目地址上下文的可编址存储器。每个位置包括一有效位，它由后续的读锁请求检查。地址和命令队列 120 也包括一读锁失败队列(Read Lock Faik Queue)120d。读锁失败队列用于保持由于在存储器位置存在锁而失败的读存储器调用请求。即，一个微引擎发出一个具有读锁请求的存储器请求，该请求在地址和控制队列 120 中被处理。存储器请求在命令队列 120c 或读队列 120b 上操作，并将其识别为读锁请求。控制器 26b 访问锁查找设备 142，以确定此存储器位置是否已经锁定。如果此存储器位置从任何以前的读锁请求被锁定，则此存储器锁定请求将失败，并将被存入读锁失孜队列 120d。如果它被解锁或如果 142 在那个地址显示没有锁，则 SRAM 接口 140 使用那个存储器调用实现对存储器 16b 传统的 SRAM 地址读/写请求。命令控制器

和地址生成器 138 也将锁输入到锁查找设备 142，使得后续的读锁请求找到锁定的存储器位置。在对锁的需要结束以后，借助于程序中的微控制指令的操作解锁一存储器位置。通过清创造在 SAM 中的有效位解锁位置。解锁以后，读锁失败队列成为最高优先级队列，给所有排队的读锁失败一个发出存储器锁请求的机会。

参考图 9，示出微引擎和 FBUS 接口逻辑(FBI)之间的通讯。在网络应用中的 FBUS 接口 28 能完成从 FBUS18 进入的数据包的头标处理。FBUS 接口完成的关键功能是提取数据包的头标，以及在 SRAM 中微可编程源/目标/协议的散列查找。如果该散列未能成功地分解，该数据包的头标被送到核心处理器 28 作更复杂的处理。

FBI 28 包含一个发送 FIFO 182，一个接收 FIFO 183，一个散列单元 188 和 FBI 控制及状态寄存器 189。这 4 个单元与微引擎 22 通讯，通过时间多路复用(time-multipexed)访问到 SRAM 总线 28，后者连接到在微引擎中的传输寄存器 78，80。即，所有与微引擎的来往通讯是经过传输寄存器 78，80。FBUS 接口 28 包括一个用于在 SRAM 不使用 SRAM 数据总线(总线 38 的部发)的时间周期内将数据压入传输寄存器的压入状态机(push state machine 200)，和一个用于从对应微引擎的传输寄存器取出数据的弹出状态机(pull state machine)202。

散列单元包括一对 FIFO=s 188a, 188b。散列单元确定 FBI28 接收一个 FBI_hash(散列)请求。散列单元 188 从发出调用的微引擎 22 取得散列键在键被取出并作散列以后，索引被送回到发出调用的微引擎 22。在单个 FBI_hash 请求下完成最多 3 个散列。总线 34 和 38 每个都是单向的，SDRAM_push/pull_date,和 sbus_push/pull_data。每个这样的总线需要控制信号，它提供对适当的微引擎 22 的传输寄存器的读/写控制。

通常，传输寄存器需要防止内存控制它们以保证读的正确性。如果 thread_1 使用写传输寄存器提供数据到 SDRAM 16a，在从 SDRAM 控制器 26a 回来的信号指出此寄存器已被更新(promoted)并现在可以再使用以前 thread_1 必须不改写此寄存器。每次写不需要从目的地返回指出功能已完成的信号，因为如果线程将多个请求写到在那个目标的同一命令队列，在那个命令队列中完成的次序是保证的，因此只有最后命令需要发回信号到该线程。然而，如果该线程使用多个命令队列(命令或读)，则这些命令请求必须分解成分别的上下文

任务，使得通过上下文交换保持次序。在本章节开头指出的例外情况是关于某个类型的操作，它对 FBUS 状态信息使用从 FBI 到传输寄存器的未经请求的 PUSH(压入)操作。为了保护在传输寄存器上的读/写决定(determinism)，FBI 在建立这些专门的 FBI 压入操作时，提供专门的 Push_protect(压入_保护)信号。任何使用 FBI 未经请求压入技术的微引擎 22 在访问 FBUS 接口/微引擎一致的传输寄存器(microengine agreed upon transfer register)以前必须测试该保护标志。如果该标志未肯定。则传输寄存器可由微引擎访问。如果该标志肯定，则在访问该寄存器前上下文必须等待 N 周期。事先，此计数由被压入的传输寄存器数加上前端保护窗(frontend protection window)而确定。基本概念是微引擎必须测试此标志，随后在连续周期内迅速将它希望从读通用寄存器传输寄存器读的数据移到使得压入引擎与微引擎读不会有碰撞。

其它实施例在下列权利要求范围之中。

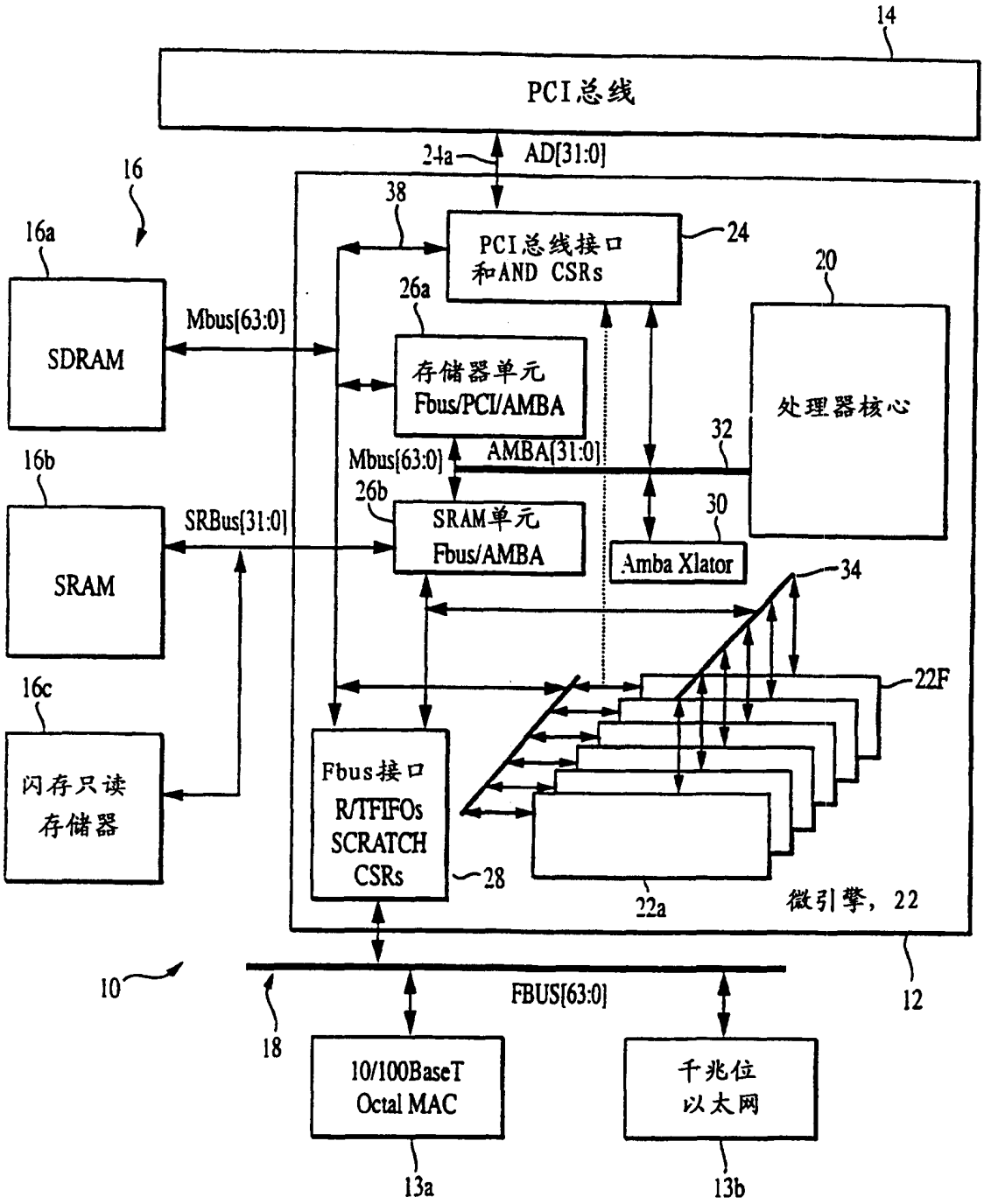

14
PCI总线
24a
AD[31:0]
16
38
16a
PCI总线接口
和AND CSRs
24
20
26a
SDRAM
Mbus[63:0]
存储器单元
Fbus/PCI/AMBA
处理器核心
32
Mbus[63:0]
AMBA[31:0]
16b
26b
SRBus[31:0]
SRAM单元
Fbus/AMBA
30
SRAM
Amba Xlator
34
16c
22F
闪存只读
存储器
Fbus接口
R/TFIFOs
SCRATCH
CSRs
28
22a
微引擎，22
10
FBUS[63:0]
12
18
10/100BaseT
Octal MAC
千兆位
以太网
13a
13b

图 1

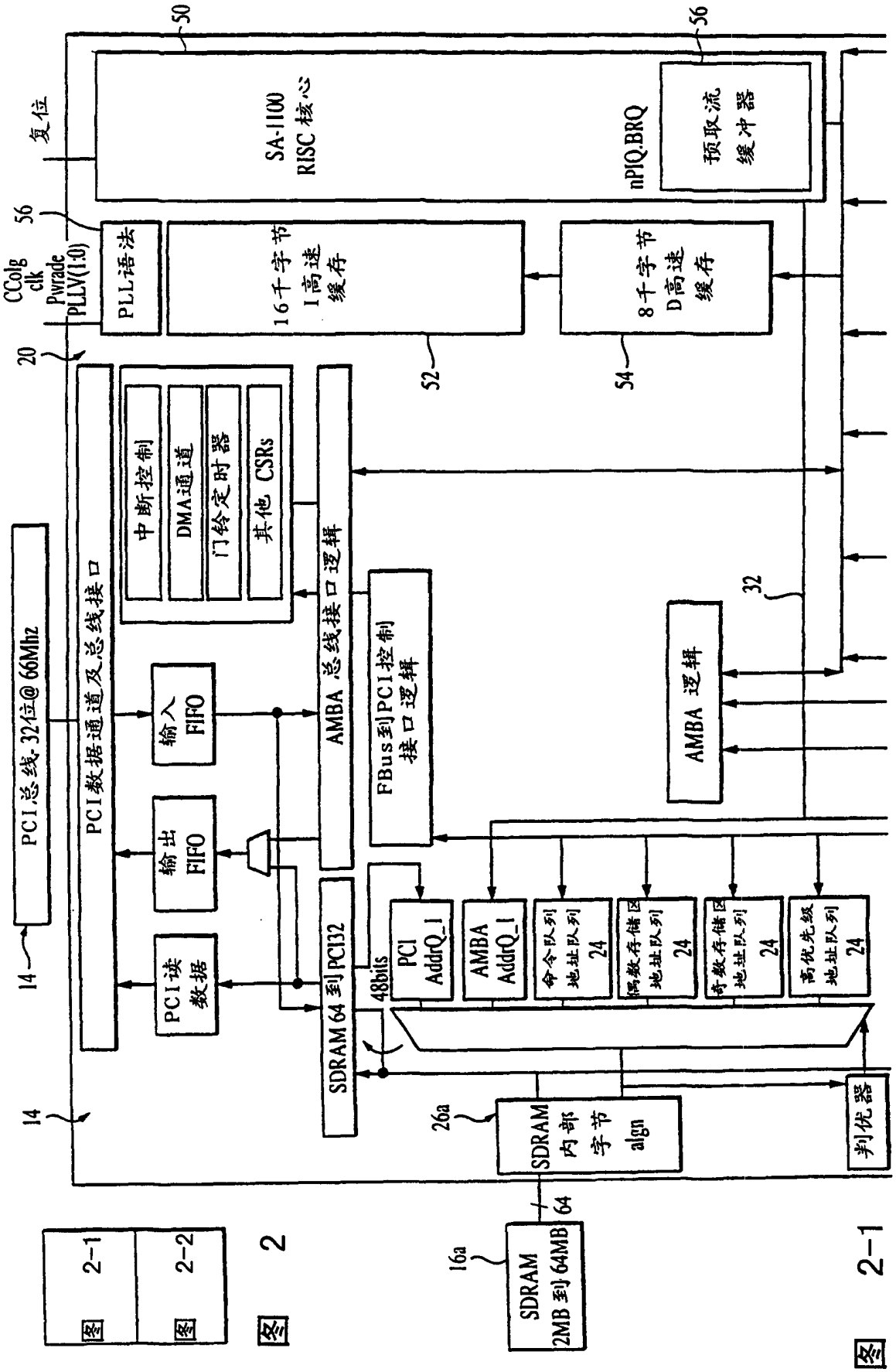

50
56
复位
SA-1100
RISC 核心
nPIQ.BRQ
预取流
缓冲器
CColg
clk
Pwrade
PLLV(1:0)
PLL语法
16千字节
I高速
缓存
8千字节
D高速
缓存
52
54
20
中断控制
DMA通道
门铃定时器
其他 CSRs
PCI总线,32位@66Mhz
PCI数据通道及总线接口
AMBA 总线接口逻辑
FBus到PCI控制
接口逻辑
AMBA 逻辑
32
输入
FIFO
输出
FIFO
PCI读
数据
14
SDRAM 64 到 PCI32
48bits
PCI
AddrQ_1
AMBA
AddrQ_1
命令队列
地址队列
24
偶数存储区
地址队列
24
奇数存储区
地址队列
24
高优先级
地址队列
24
26a
SDRAM
内部
字节
align
判优器
16a
SDRAM
2MB 到 64MB
64

图 2

图 2-1

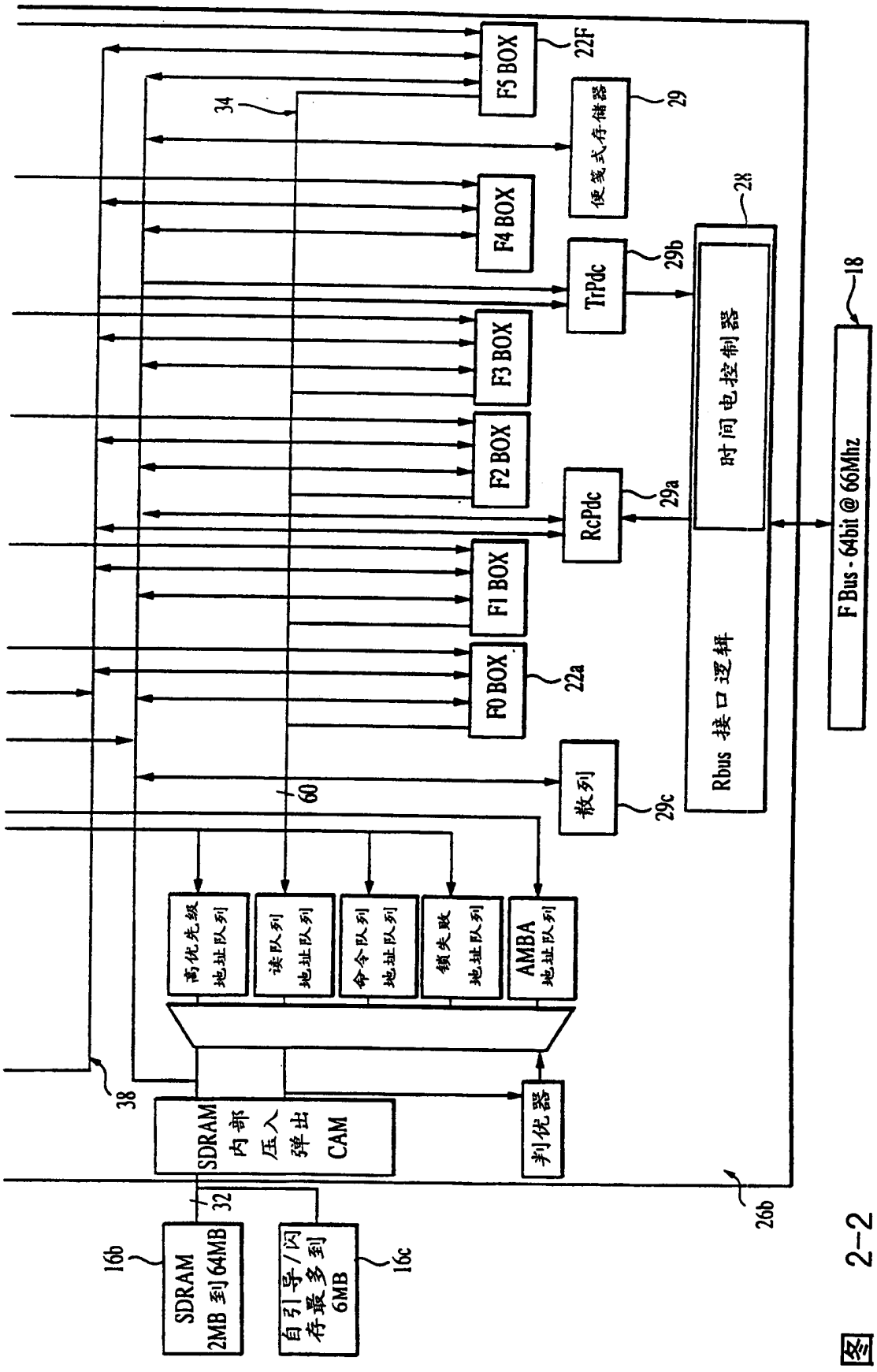

F5 BOX
22F
34
便笺式存储器
29
F4 BOX
TrPdc
29b
28
F3 BOX
时间电控制器
18
F2 BOX
RcPdc
29a
F1 BOX
F0 BOX
22a
Rbus 接口逻辑
F Bus - 64bit @ 66Mhz
60
散列
29c
高优先级地址队列
读队列地址队列
命令队列地址队列
锁失败地址队列
AMBA地址队列
38
SDRAM内部压入弹出CAM
判优器
26b
32
16b
SDRAM 2MB 到 64MB
自引导/闪存最多到6MB
16c

图 2-2

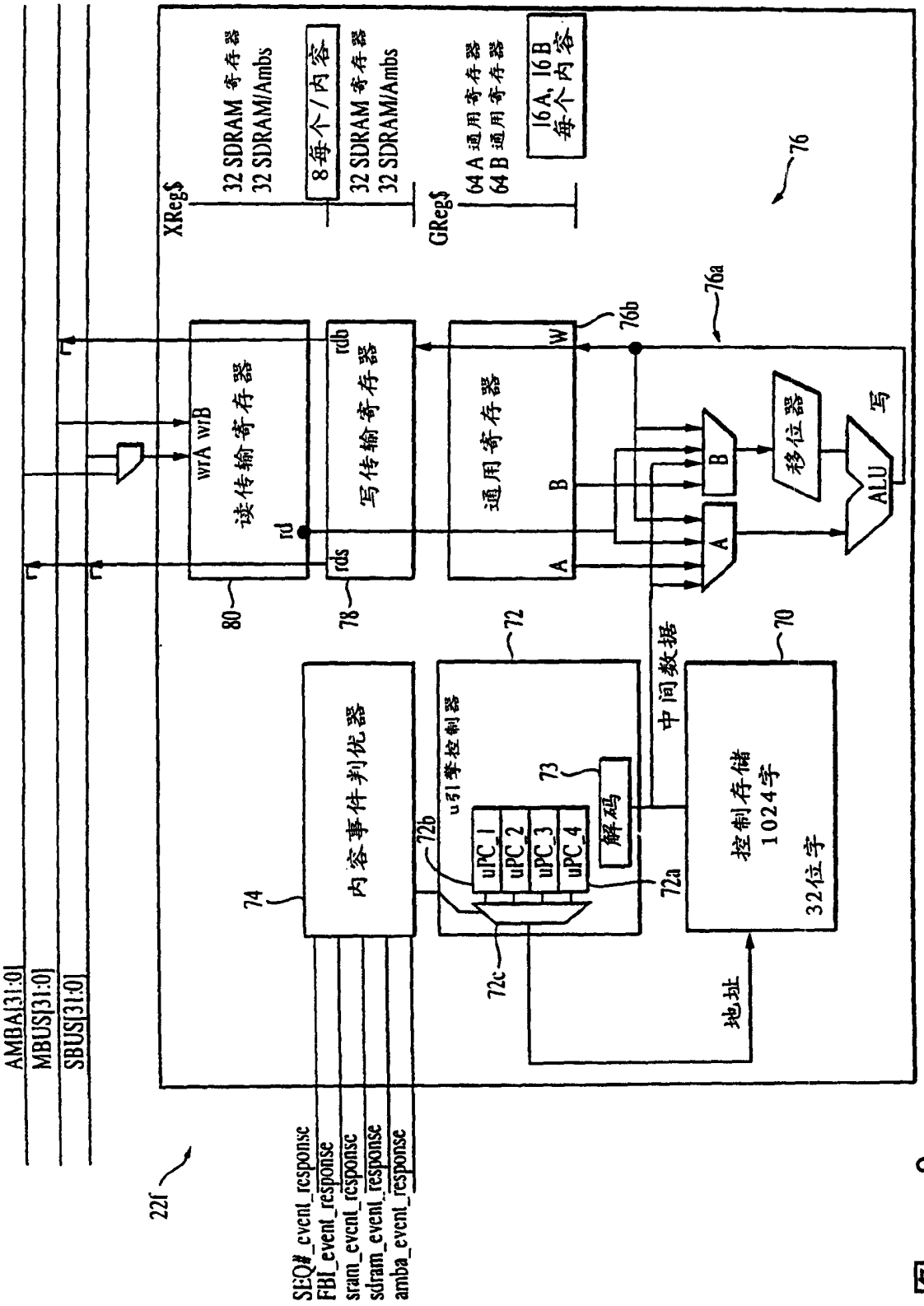

AMBA[31:0]
MBUS[31:0]
SBUS[31:0]
22f
SEQ#_event_response
FBI_event_response
sram_event_response
sdram_event_response
amba_event_response
XReg$
32 SDRAM 寄存器
32 SDRAM/Ambs
8每个/内容
GReg$
32 SDRAM 寄存器
32 SDRAM/Ambs
64 A 通用寄存器
64 B 通用寄存器
16 A, 16 B
每个内容
76
76a
76b
wrA wrB
读传输寄存器
rd
80
rdb
写传输寄存器
rds
78
通用寄存器
W
A
B
A
B
移位器
ALU
写
74
内容事件判优器
72
u引擎控制器
uPC_1
uPC_2
uPC_3
uPC_4
73
解码
72b
72a
72c
中间数据
70
控制存储
1024字
32位字
地址

图 3

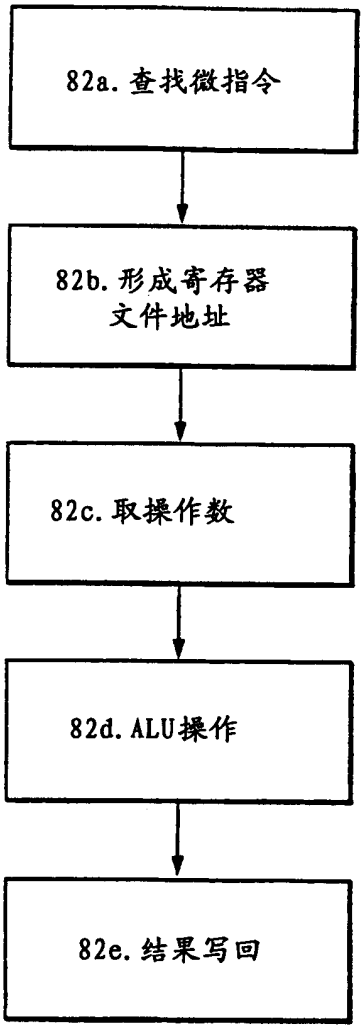
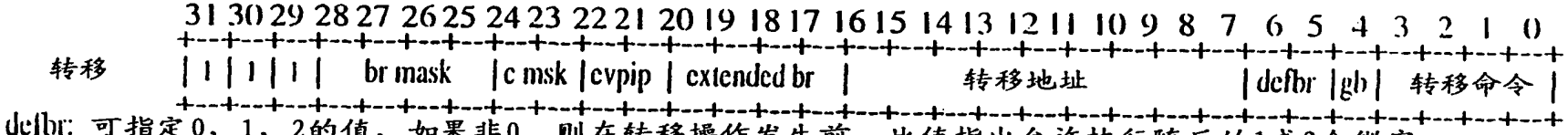


图 4

转移指令:



defbr: 可指定0，1，2的值，如果非0，则在转移操作发生前，此值指出允许执行随后的1或2个微字

gb: 若置位，推测取转移路径，因此预取转移微字地址，否则预取非转移路径。此字段只有当defer = 0或defer = 1时才允许置位

转移地址: 条件地或非条件地选择转移地址

br_mask: 被解码成下列选项:

1) 无条件转移
2) 当 ALU<31>=1 (<0) 时转移
3) 当 ALU<31>=0 (>=0) 时转移
4) 当 ALU<31>=1 或 ALU<31:0>=0 (<=0) 时转移
5) 当 ALU<31>=0 和 ALU<31:0>!=0 (>0) 时转移
6) 当 ALU<31:0>=0 (=0) 时转移
7) 当 ALU<31:0>=1 (!=0) 时转移
8) 当指定内容掩码 = 当前内容时转移
9) 当指定内容掩码!=当前内容时转移
10) 根据carry-out置位转移
11) 根据carry-out清除转移
15) 审查扩展转移字段，进一步解码转移类型

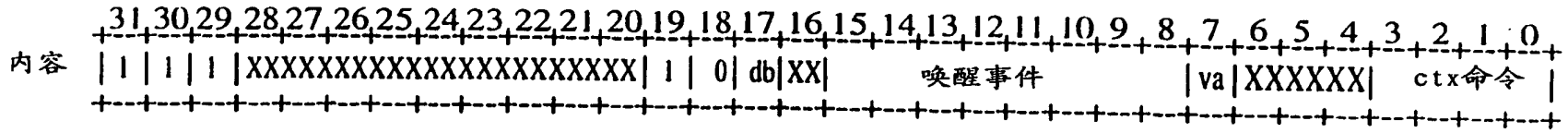
extend_br: 根据各种内容交换信号或其他信号转移

evpip: 指出此应计算此转移的管线阶段

c msk: 指定一个内容数，据此数作条件转移

转移指令: 进一步指定转移的类型，如审查某些其他转移准则的条件码

图 5A



内容描述符:

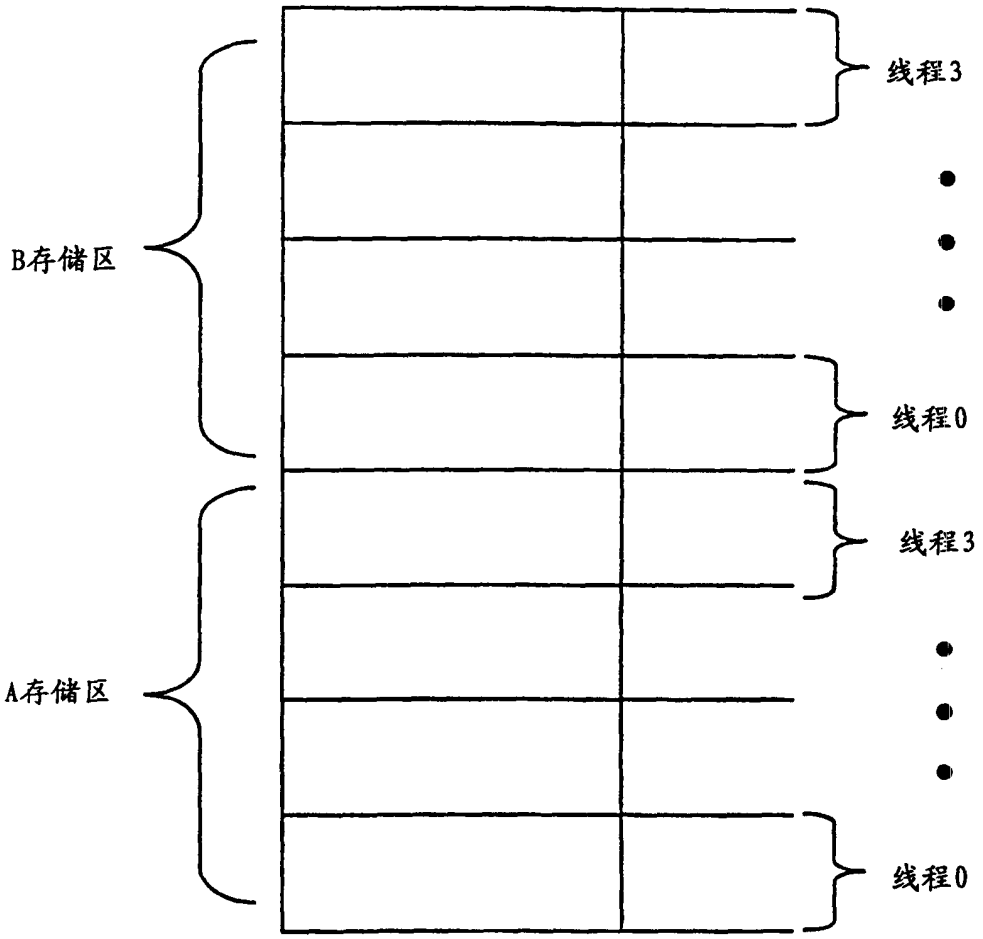
1) 唤醒事件

0 = kill	8 = FBI	128 = SEQ_NUM_LSB
1 = voluntary	16 = INTER_THREAD	
2 = SRAM	32 = PCI_DMA_1	
3 = SDRAM	64 = PCI_DMA_2	

2) db → 转移延迟量

3) va → 顺序数的值

图 5B


线程3
B存储区
线程0
线程3
A存储区
线程0

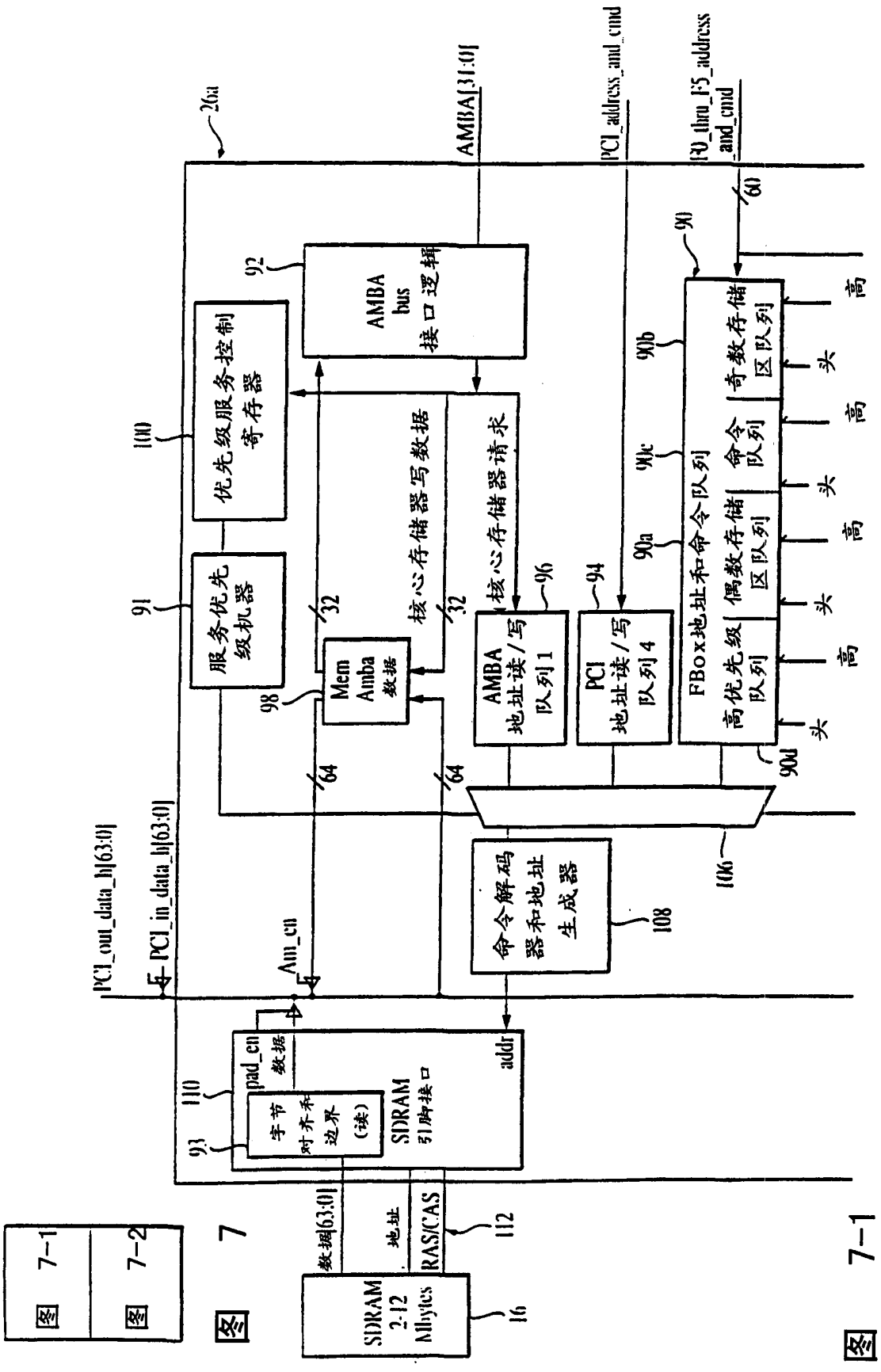


图 7-1
图 7-2
图 7
SDRAM 2-12 Mbytes
16
数据[63:0]
地址
RAS/CAS
112
SDRAM 引脚接口
字节对齐和边界（读）
数据
pad_cn
addr
93
110
PCI_out_data_h[63:0]
PCI_in_data_h[63:0]
Am_cn
命令解码器和地址生成器
108
106
64
64
32
32
Mem Amba 数据
98
服务优先级机器
91
优先级服务控制寄存器
100
核心存储器写数据
核心存储器请求
AMBA 地址读/写队列 1
96
PCI 地址读/写队列 4
94
FBox地址和命令队列
高优先级队列
偶数存储区队列
命令队列
奇数存储区队列
90a
90b
90c
90d
头
高
AMBA bus 接口逻辑
92
AMBA[31:0]
26a
PCI_address_and_cmd
FQ_thru_FS_address and_cmd
60
91
图 7-1

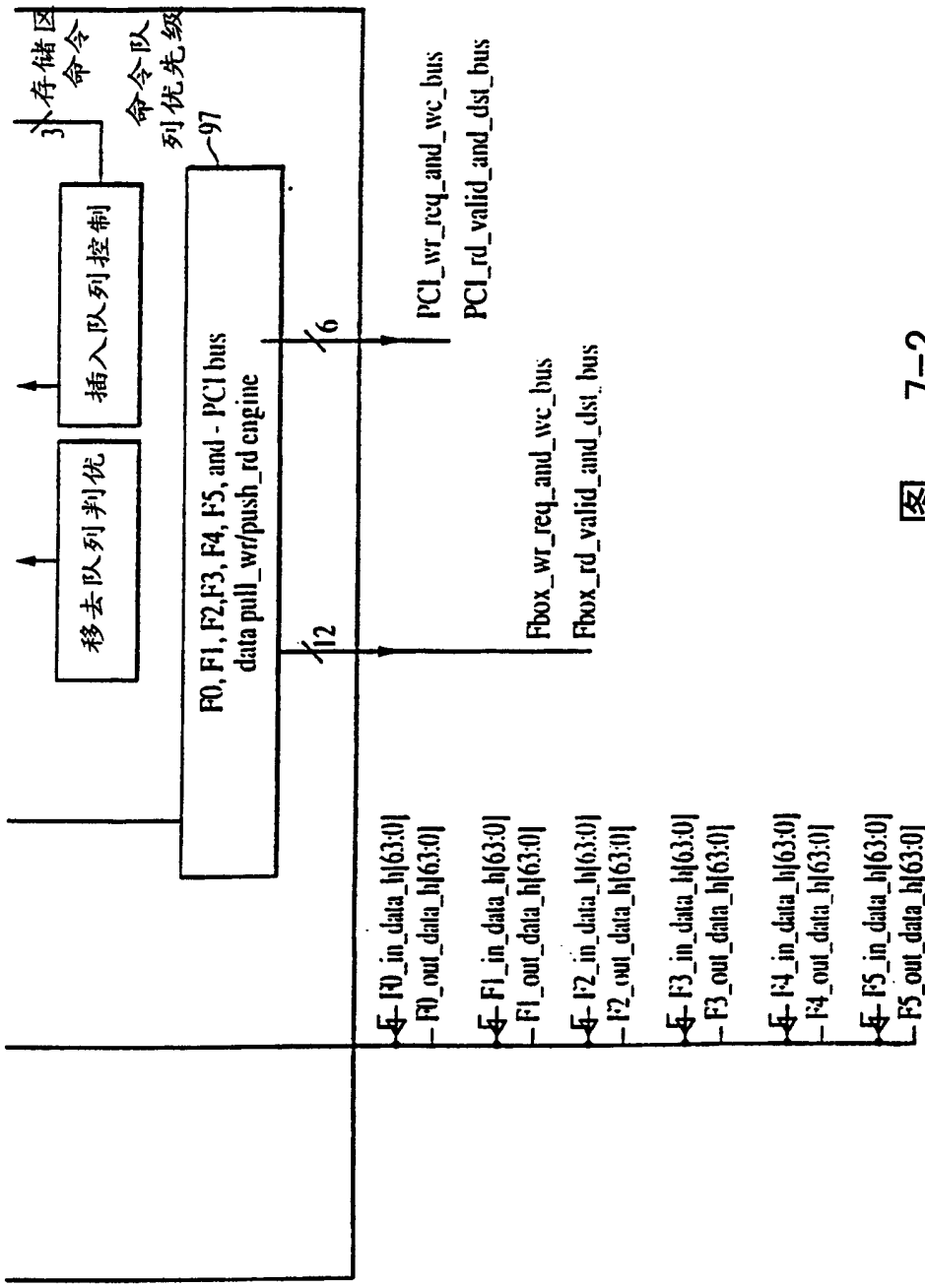

存储区命令
命令队列优先级
97
插入队列控制
移去队列判优
F0, F1, F2, F3, F4, F5, and - PCI bus
data pull_wr/push_rd engine
6
12
PCI_wr_req_and_wc_bus
PCI_rd_valid_and_dst_bus
Fbox_wr_req_and_wc_bus
Fbox_rd_valid_and_dst_bus
F0_in_data_b[63:0]
F0_out_data_b[63:0]
F1_in_data_b[63:0]
F1_out_data_b[63:0]
F2_in_data_b[63:0]
F2_out_data_b[63:0]
F3_in_data_b[63:0]
F3_out_data_b[63:0]
F4_in_data_b[63:0]
F4_out_data_b[63:0]
F5_in_data_b[63:0]
F5_out_data_b[63:0]

图 7-2

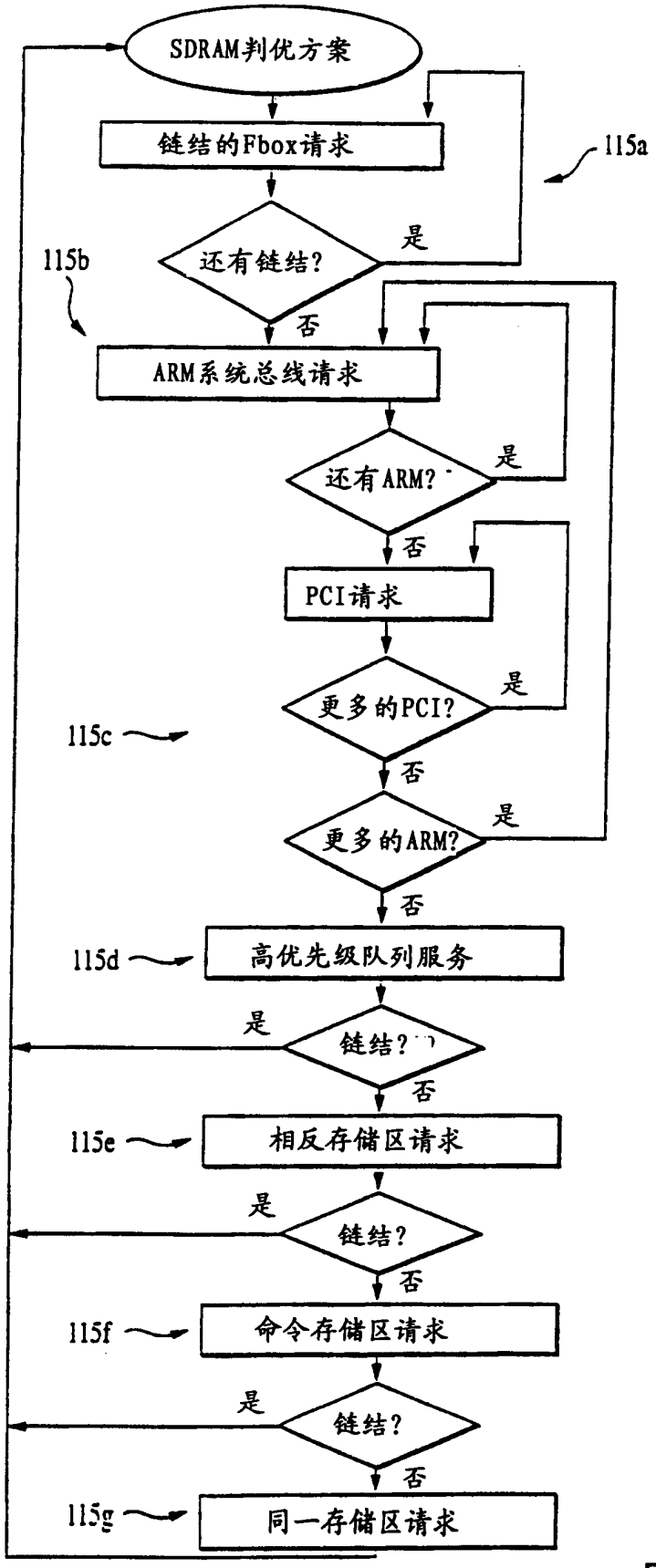

SDRAM判优方案
链结的Fbox请求
115a
还有链结?
是
否
115b
ARM系统总线请求
还有ARM?
是
否
PCI请求
更多的PCI?
是
115c
否
更多的ARM?
是
否
115d
高优先级队列服务
是
链结?
否
115e
相反存储区请求
是
链结?
否
115f
命令存储区请求
是
链结?
否
115g
同一存储区请求

图 7A

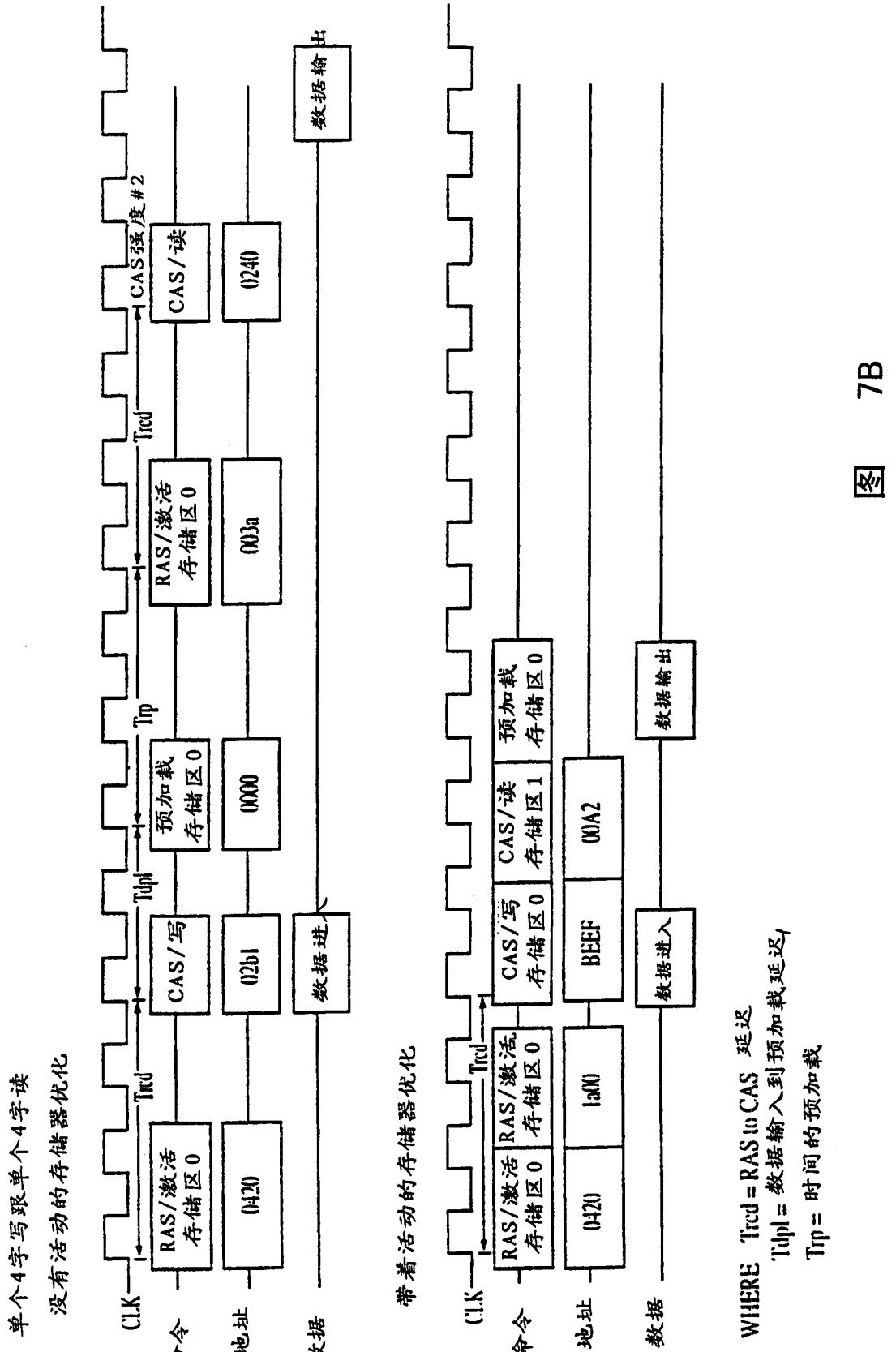

单个4字写跟单个4字读
没有活动的存储器优化
CLK
命令
地址
数据
RAS/激活 存储区 0
CAS/写
预加载 存储区 0
RAS/激活 存储区 0
CAS/读
0420
02b1
0000
003a
0240
数据进
数据输出
Trcd
Tdpl
Trp
CAS强度#2
带着活动的存储器优化
CAS/写 存储区 0
CAS/读 存储区 1
预加载 存储区 0
BEEF
00A2
数据进入
数据输出
WHERE Trcd = RAS to CAS 延迟
Tdpl = 数据输入到预加载延迟
Trp = 时间的预加载

图 7B

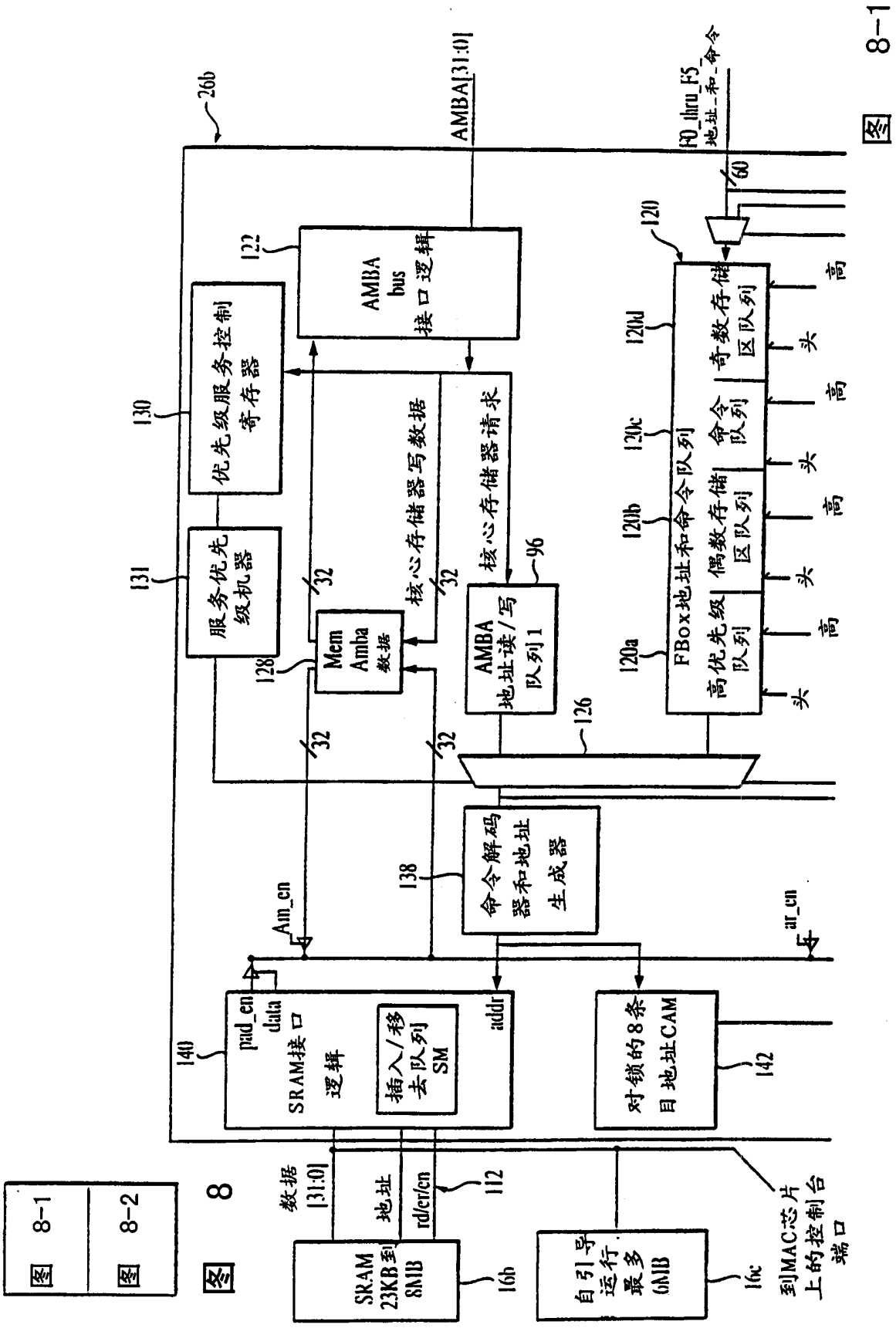


图 8-1
图 8-2
图 8
图 8-1
26b
AMBA[31:0]
FQ_thru_F5
地址_和_命令
60
130
131
122
AMBA
bus
接口逻辑
优先级服务控制
寄存器
服务优先
级机器
128
Mem
Amba
数据
32
核心存储器写数据
核心存储器请求
96
AMBA
地址读/写
队列1
126
120
120d
奇数存储
区队列
120c
命令
队列
120b
偶数存储
区队列
120a
FBox地址和命令队列
高优先级
队列
头
高
Am_cn
138
命令解码
器和地址
生成器
ar_cn
140
pad_en
data
SRAM接口
逻辑
插入/移
去队列
SM
addr
对锁的8条
目地址CAM
142
数据
[31:0]
地址
rd/cr/cn
112
SRAM
23KB到
8MB
16b
自引导
运行
最多
6MB
16c
到MAC芯片
上的控制台
端口

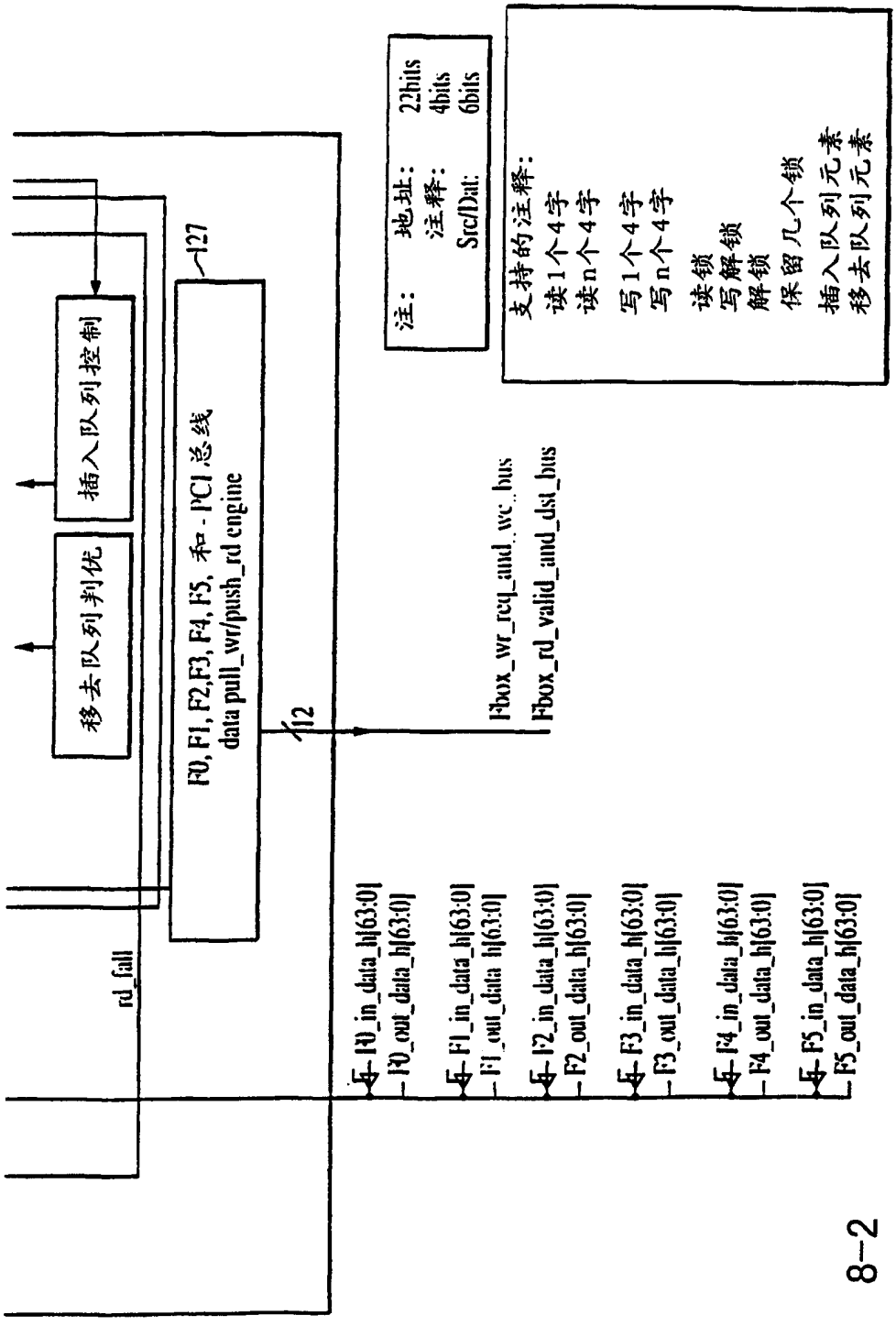

移去队列判优
插入队列控制
F0, F1, F2,F3, F4, F5, 和 - PCI 总线
data pull_wr/push_rd engine
127
12
rd_fail
F0_in_data_h[63:0]
F0_out_data_h[63:0]
F1_in_data_h[63:0]
F1_out_data_h[63:0]
F2_in_data_h[63:0]
F2_out_data_h[63:0]
F3_in_data_h[63:0]
F3_out_data_h[63:0]
F4_in_data_h[63:0]
F4_out_data_h[63:0]
F5_in_data_h[63:0]
F5_out_data_h[63:0]
Fbox_wr_req_and_wc_bus
Fbox_rd_valid_and_dst_bus
注： 地址: 22bits
注释: 4bits
Src/Dat: 6bits
支持的注释:
读1个4字
读n个4字
写1个4字
写n个4字
读锁
写解锁
解锁
保留几个锁
插入队列元素
移去队列元素

图 8-2

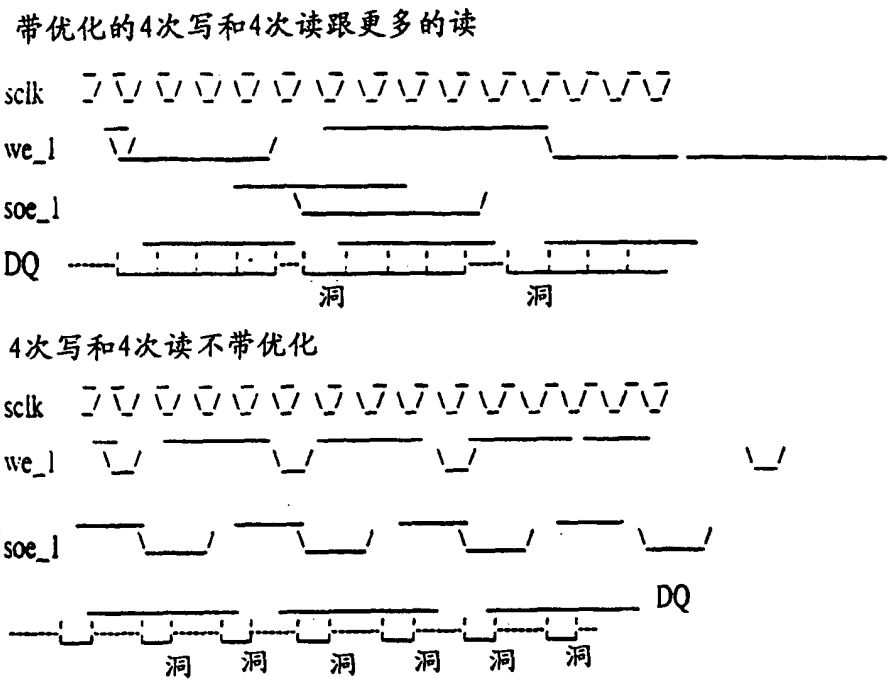
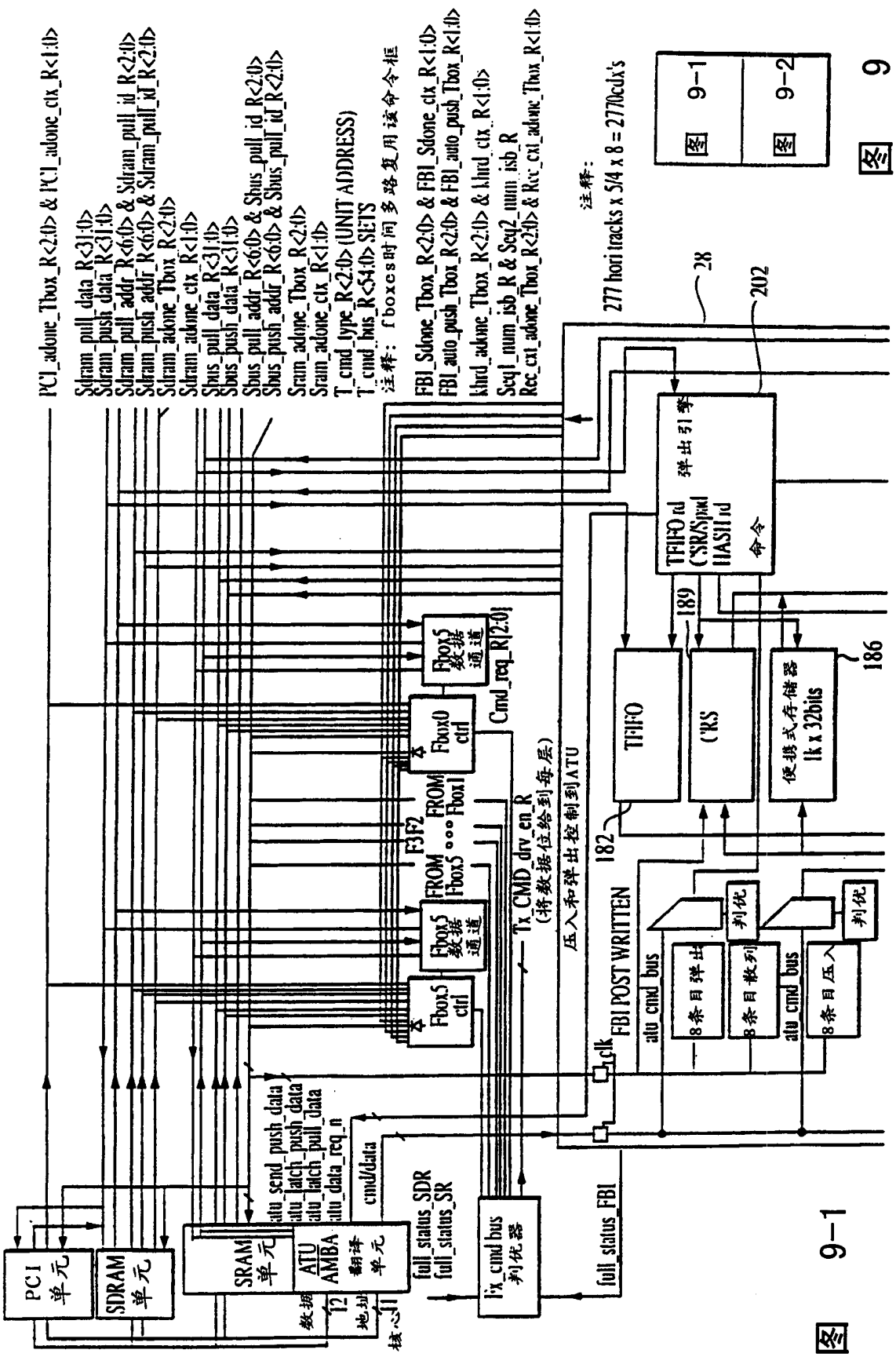

带优化的4次写和4次读跟更多的读
sclk
we_l
soe_l
DQ
洞
洞
4次写和4次读不带优化
sclk
we_l
soe_l
DQ
洞
洞
洞
洞
洞
洞
10周期对14

图 8A


PCI 单元
SDRAM 单元
SRAM 单元
ATU AMBA 翻译单元
PCI_adone_Tbox_R<2:0> & PCI_adone_ctx_R<1:0>
Sdram_pull_data_R<31:0>
Sdram_push_data_R<31:0>
Sdram_pull_addr_R<6:0> & Sdram_pull_id_R<2:0>
Sdram_push_addr_R<6:0> & Sdram_pull_id_R<2:0>
Sdram_adone_Tbox_R<2:0>
Sdram_adone_ctx_R<1:0>
Sbus_pull_data_R<31:0>
Sbus_push_data_R<31:0>
Sbus_pull_addr_R<6:0> & Sbus_pull_id_R<2:0>
Sbus_push_addr_R<6:0> & Sbus_pull_id_R<2:0>
Sram_adone_Tbox_R<2:0>
Sram_adone_ctx_R<1:0>
T_cmd_type_R<2:0> (UNIT ADDRESS)
T_cmd_bus_R<54:0> SETS
注释：fboxes时间多路复用该命令框
FBI_Sdone_Tbox_R<2:0> & FBI_Sdone_ctx_R<1:0>
FBI_auto_push_Tbox_R<2:0> & FBI_auto_push_Tbox_R<1:0>
khrd_adone_Tbox_R<2:0> & khrd_ctx_R<1:0>
Seq1_num_isb_R & Seq2_num_isb_R
Rec_ctl_adone_Tbox_R<2:0> & Rec_ctl_adone_Tbox_R<1:0>
注释：
277 hori tracks x 5/4 x 8 = 2770cdx's
28
202
TFIFO rd (CSR/S)pad HASH1 rd
弹出引擎
命令
189
TFIFO
CRS
便携式存储器 1k x 32bits
186
182
Fbox5 数据通道
Fbox0 ctrl
Cmd_req_R[2:0]
FROM Fbox5 ... FROM Fbox1
F3F2
Fbox5 数据通道
Fbox5 ctrl
Tx_CMD_drv_en_R
(将数据往给到每层)
压入和弹出控制到ATU
clk
FBI POST WRITTEN
atu cmd bus
8条目弹出
判优
8条目散列
atu cmd bus
判优
8条目压入
atu_send_push_data
atu_latch_push_data
atu_latch_pull_data
atu_data_req_n
cmd/data
full_status_SDR
full_status_SR
full_status_FBI
Tx_cmd bus 判优器
数据 32
地址 11
核心 11
图 9-1
图 9-2
图 9

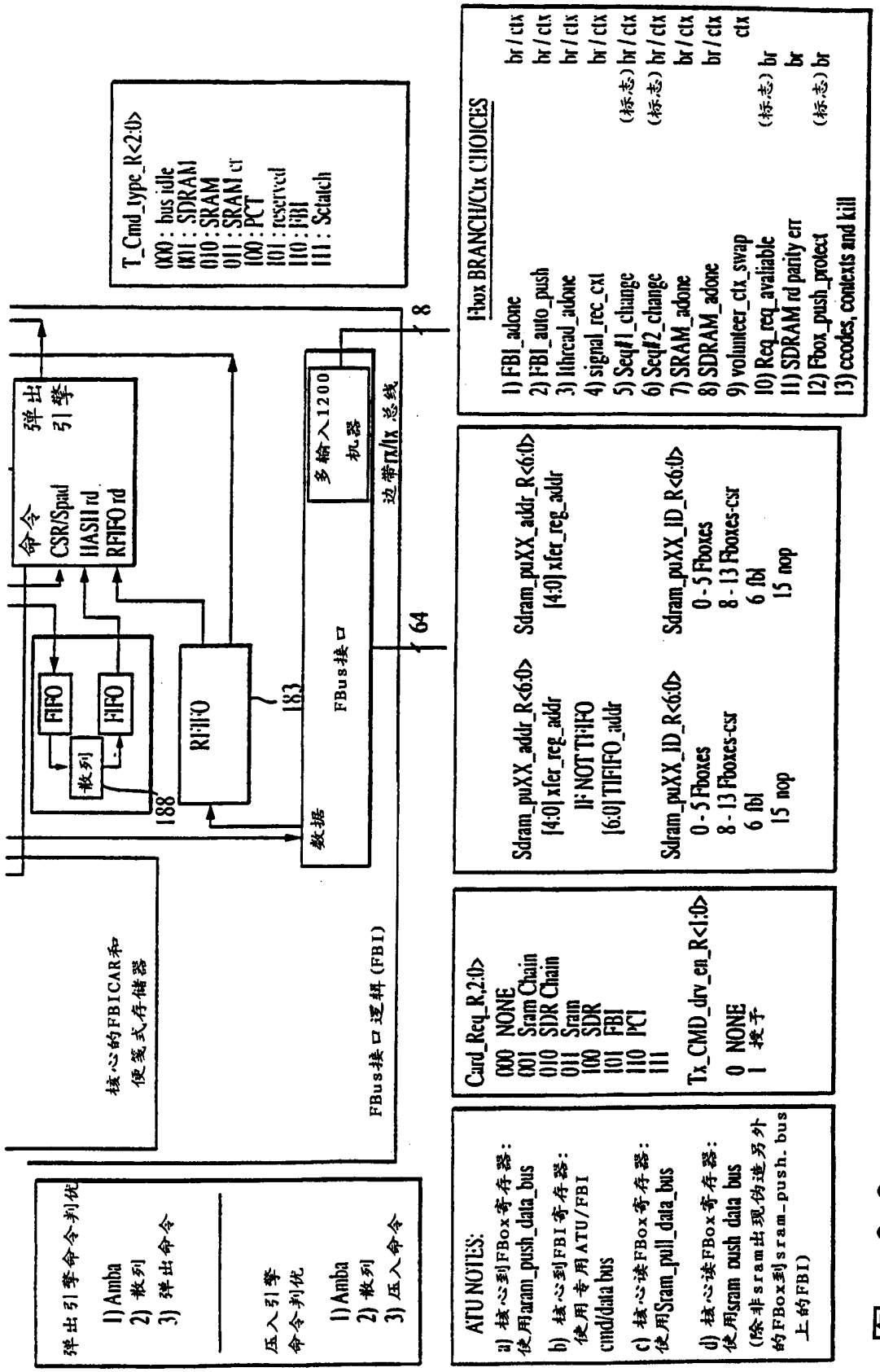

T_Cmd_type_R<2:0>
000 : bus idle
001 : SDRAM
010 : SRAM
011 : SRAM ctr
100 : PCT
101 : reserved
110 : FBI
111 : Scratch
Fbox BRANCH/Ctx CHOICES
1) FBI_adone
2) FBI_auto_push
3) Ithread_adone
4) signal_rec_ctx
5) Sequ1_change
6) Sequ2_change
7) SRAM_adone
8) SDRAM_adone
9) volunteer_ctx_swap
10) Req_req_avaliable
11) SDRAM rd parity err
12) Fbox_push_protect
13) ccodes, contexts and kill
br / ctx
(标志) br / ctx
(标志) br
(标志) br
命令
CSR/Spad
HASH rd
RFIFO rd
弹出引擎
FIFO
FIFO
数列
RFIFO
188
183
数据
FBus接口
多输入1200机器
64
8
边带TX/RX 总线
核心的FBICAR和便笺式存储器
FBus接口逻辑 (FBI)
弹出引擎命令判优
1) Amba
2) 数列
3) 弹出命令
压入引擎命令判优
1) Amba
2) 数列
3) 压入命令
Sdram_puXX_addr_R<6:0>
[4:0] xfer_reg_addr
If NOT TFIFO
[6:0] TFIFO_addr
Sdram_puXX_addr_R<6:0>
[4:0] xfer_reg_addr
Sdram_puXX_ID_R<6:0>
0 - 5 Fboxes
8 - 13 Fboxes-csr
6 fbi
15 nop
Sdram_puXX_ID_R<6:0>
0 - 5 Fboxes
8 - 13 Fboxes-csr
6 fbi
15 nop
Curd_Req_R.2:0>
000 NONE
001 Sram Chain
010 SDR Chain
011 Sram
100 SDR
101 FBI
110 PCI
111
Tx_CMD_drv_en_R<1:0>
0 NONE
1 接手
ATU NOTES:
a) 核心到FBox寄存器:
使用sram_push_data_bus
b) 核心到FBI寄存器:
使用专用ATU/FBI
cmd/data bus
c) 核心读FBox寄存器:
使用Sram_pull_data_bus
d) 核心读FBox寄存器:
使用sram push data bus
(除非sram出现仿造另外
的FBox到sram_push. bus
上的FBI)

图 9-2