[19]中华人民共和国国家知识产权局

[51]Int. Cl[6]
G06F 9/46

[12]发明专利申请公开说明书

[21]申请号 99103962.9

[43]公开日 1999年10月27日

[11]公开号 CN 1233015A

[22]申请日 99.3.10 [21]申请号 99103962.9

[30]优先权

[32]98.3.10 [33]US [31]077,384

[32]98.12.17 [33]US [31]213,102

[71]申请人 朗迅科技公司

地址 美国新泽西

[72]发明人 威尔海尔姆斯·约瑟弗斯·迪普斯特拉藤

迈克尔·A·非斯切尔

威斯利·D·哈德尔

[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

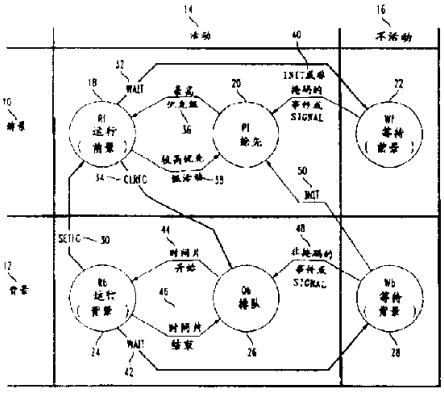
代理人 付建军

权利要求书3页 说明书35页 附图页数21页

[54]发明名称 具有节能模式自动进入的历境控制器以及采用其的处理器

[57]摘要

用于管理处理器中的多任务处理的一种历境控制器及一种方法。在一个实施例 中,该历境控制器包括:(1)前景和背景任务控制器,它们分别把处理器资 源分配给与前景和背景任务相应的活动历境;以及,(2)模式切换电路,它 与所述前景和背景任务控制器耦合,并当所有所述历境都不活动时将所述处理 器置于一种闲置状态和一种省电模式。



ISSN1008-4274

权 利 要 求 书

1. 用于管理处理器中的多任务处理的一种历境控制器，包括:

前景和背景任务控制器，它们分别把处理器资源分配给与前景和背景任务相应的活动历境；以及

模式切换电路，它与所述前景和背景任务控制器耦合，并当所有所述历境都不活动时将所述处理器置于一种闲置状态和一种省电模式。

2. 根据权利要求1的历境控制器， 其中一个软件切换状态区分所述前景任务和所述背景任务。

3. 根据权利要求1的历境控制器， 其中所述前景和背景任务控制器只把所述处理器资源分配给所述活动历境。

4. 根据权利要求1的历境控制器， 其中所述背景任务控制器根据各个所述背景任务所执行的指令的数目激活与背景任务相应的所述历境。

5. 根据权利要求1的历境控制器， 其中所述历境被存储在不同的寄存器组中。

6. 根据权利要求1的历境控制器， 其中所述前景任务控制器根据优先级并响应于事件而激活与前景任务相应的历境且所述背景任务控制器服从于与所述前景任务相应的所述历境的激活，周期性地激活与背景任务相应的历境。

7. 根据权利要求1的历境控制器， 其中所述前景任务控制器适合于通过指向一个软件可选择的存储位置而激活与一个具体前景任务相应的历境。

8. 管理处理器中的多任务处理的一种方法，包括以下步骤:

把处理器资源分配给分别与前景和背景任务相应的活动历境；以及

当所有所述历境都不活动时把所述处理器置于一种闲置状态和一种省电模式。

9. 根据权利要求8的方法， 其中一种软件切换状态区分所述前景任务和所述背景任务。

10. 根据权利要求8的方法，其中所述前景和背景任务控制器只把所

述处理器资源分配给所述活动历境。

11. 根据权利要求8的方法，其中所述背景任务控制器根据各个所述背景任务所执行的指令的数目激活与背景任务相应的历境。

12. 根据权利要求8的方法，其中所述历境被存储在不同的寄存器组中。

13. 根据权利要求8的方法，其中所述前景任务控制器根据优先级并响应于事件而激活与前景任务相应的历境且所述背景任务控制器服从于与所述前景任务相应的所述历境的激活，周期性地激活与背景任务相应的历境。

14. 根据权利要求8的方法，其中所述前景任务控制器适合于通过指向一个软件可选择的存储位置而激活与一个具体前景任务相应的历境。

15. 一种处理器，包括:

一个指令解码器，它对接收到所述处理器中并与多个任务相应的指令进行解码;

多个寄存器组，它们与所述多个任务相应，并包含所要操纵的操作数;

一个执行内核，它与所述指令解码器和所述多个寄存器组耦合，并执行与所述多个任务中的一个活动的任务相应的指令以操纵所述操作数中的一些操作数; 以及

一个历境控制器，它与所述指令解码器和所述执行内核耦合，并管理处理器中的多任务处理，包括:

前景和背景任务控制器，它们分别把处理器资源分配给与前景和背景任务相应的活动历境; 以及

模式切换电路，它与所述前景和背景任务控制器耦合，并当所有所述历境都不活动时将所述处理器置于一种闲置状态和一种省电模式。

16. 根据权利要求15的处理器， 其中一个软件切换状态区分所述前景任务和所述背景任务。

17. 根据权利要求15的处理器， 其中所述前景和背景任务控制器只把所述处理器资源分配给所述活动历境。

18. 根据权利要求15的处理器， 其中所述背景任务控制器根据各个所述背景任务所执行的指令的数目激活与背景任务相应的所述历境。

19. 根据权利要求15的处理器， 其中所述历境被存储在单独的寄存器组中。

20. 根据权利要求15的处理器， 其中所述前景任务控制器根据优先级并响应于事件而激活与前景任务相应的历境且所述背景任务控制器服从于与所述前景任务相应的所述历境的激活，周期性地激活与背景任务相应的历境。

21. 根据权利要求15的处理器， 其中所述前景任务控制器适合于通过指向一个软件可选择的存储位置而激活与一个具体前景任务相应的历境。

22. 根据权利要求15的处理器， 其中所述处理器形成了一个通用计算机的一部分。

说　　明　　书

具有节能模式自动进入的
历境控制器以及采用其的处理器

相关申请的交叉引用

序号：60/077，454

题目：事件驱动和循环历境控制器以及采用其的处理器

发明人：Diepstraten等

递交日：1998年3月10日

序号：60/077，469

题目：具有基于指令的时间分割任务切换能力的历境控制器以及采用其的处理器

发明人：Diepstraten等

递交日：1998年3月10日

序号：60/077，461

题目：具有基于状态的背景任务资源分配能力的历境控制器以及采用其的处理器

发明人：Diepstraten等

递交日：1998年3月10日

序号：60/077，406

题目：具有历境特定事件选择机制的历境控制器以及采用其的处理器

发明人：Diepstraten等

递交日：1998年3月10日

序号：60/077，575

题目：具有事件相关矢量选择的历境控制器和采用其的处理器

发明人：Diepstraten等

递交日：1998年3月10日

上述申请与本发明一起被转让且它们全部都被作为参考文献。

本申请还要求了基于1998 年3 月10 日递交的、题目为“具有节能模式自动进入的历境控制器以及采用其的处理器”的美国临时申请序号第60/077，384号的权利， 该申请与本发明一起被转让并在此被作为参考文献。

本发明一般地涉及计算机处理器，且更具体地说是涉及具有节能模式自动进入的历境控制器和采用该历境控制器的处理器。

通用计算机中以及被用作嵌入控制器的处理器，通常得到编程，以同时处理多个任务。这些任务的一个子组可以响应于具体的、外界的事件而按照时间来执行，而这些任务中其余的则可在没有严格的实时限制的情况下执行。为了用单个的数据路径来处理这两组任务，这些处理器需要有效的机制，来迅速地响应外界事件，同时在没有外界事件得到处理时允许非实时处理。

事件响应的主要机制是程序中断，它是在50年代中期首先得到采用的。在过去的40年中，大多数的处理器构造包括了程序中断功能—它在发生外界事件时使“背景”任务的执行被暂停，并启动“前景”任务的执行。各个程序中断,通常称为“中断”，（与处理器的指令流适当地同步）根据一个适当的事件的确立，造成处理器执行状态的可逆改变。

在50年代后期发展起来的优先中断是对程序中断功能的一种共同的增强。在一个支持优先中断的处理器中，以静态或动态的方式把分立的优先级分配给多个事件（中断请求）信号。与这些信号中的每一个相关的是处理器的执行状态下的可逆改变的一种唯一的可识别结

果状态。优先中断的每一次发生都选择与中断状态改变被启动时确立的最高优先中断请求相关的结果状态。

当进行处理器的程序执行状态的可逆改变时的基本行动，是保存中断程序的执行地址（以及隐含的指令间状态，诸如状态码），并在与造成中断的事件相关的程序地址处开始中断处理。这种程序地址通常是从称为中断矢量的预定存储位置获得的。在中断处理程序结束时，保存的执行地址（以及状态值—如果有的话）得到恢复，使得中断的程序的执行在中断点处得到恢复。在多数中断处理过程中，需要保存并随后恢复额外的处理器状态，以执行响应中断所需的操作。这种额外的状态主要指程序计数器之外的处理器寄存器的内容。

把这些寄存器保存到堆栈或专用的存储块或从中恢复这些寄存器，会消耗大量的时间。因此，在60年代中当集成电路使硬件寄存器的成本和尺寸减小时，某些处理器具有了多组寄存器。不同组寄存器的选择，不论是通过中断支持硬件还是通过中断处理软件，通过消除至主存储器的保存和从其的恢复的消耗，而实现了快得多的中断响应。

在1970年引入的IBM系统/7上，多寄存器组概念达到了其现在的形式。系统/7具有用于各个中断级的专用、硬件选择寄存器组，并通过在各个组中包括一个寄存器以保存执行地址（程序计数器值）而在该级被处于较高优先级的一个中断所占用时进一步减小了中断历境切换时间。结果是得到了800ns的中断历境切换时间和400ns的中断返回时间，这两个指标对于采用1969年技术的16位小型计算机来说是非常出色的语音。系统/7还开创了动态中断分配，其中各个中断源使用的优先级由软件设定，并可在系统操作期间得到改变。

这种寄存器组加程序计数器技术的最终普及，使得事件能够在它们最后的执行地址处启动处理过程，而不是要求它们总是利用一个中断矢量地址来启动。为了控制I/O装置，数据通信和网络协议，以及以通信状态机定义的其他过程，这是一个主要的好处，因为状态机可用用于指令寻址和作为（暗指）状态寄存器的等级程序计数器（the

level's program counter）来实施。这不仅消除了对单独的状态寄存器的需要，而且还消除了用于根据状态寄存器的值来选择适当处理过程的调度过程的消耗。实际上，这种寄存器组加程序计数器结构对操作系统软件通常支持的“任务”或“执行线索”概念提供了直接的硬件支持。

为了用这种技术实施I/O 控制状态机而开发的第一个机器，是CharlesThacker于1972年在Xerox Palo Alto Research Center设计的“Alto”实验个人计算机。从70年代早期以来，为单片微机和微处理器开发了很多种这些中断和历境切换机器。然而，这些机器都没有引入用于响应外界事件的迅速历境切换的基本新机制。

在高性能的系统中，（一或多个）处理器经常可被用于I/O控制和/或外界事件处理。然而，如果用与用在系统的中央处理中的技术类似的技术实施，这些I/O处理器的利用一般是很低的。这是由于这样的事实，即对于任何具体的电路技术，用于实施处理器数据路径的逻辑器件的运行比用于实施主存储器的存储器件要快得多，且逻辑和存储器件都能够支持比所有相连的外设高的数据带宽。

在60年代，对有多个I/O控制器的高性能系统结构的需求发展了一种技术，用于在多个控制器功能之间共享单个的数据路径，虽然这些功能在逻辑上是不相联系的。该技术利用了单个的物理数据路径和指令解码器在循环的基础上处理多个逻辑处理器的指令流。各个逻辑处理器的仅有的专用资源，是保持其执行状态（程序计数器和寄存器值）的存储器。控制电路允许在顺序、循环的基础上为各个逻辑处理器执行预定数目的指令（通常1个）。这种控制电路在不同逻辑处理器的指令周期之间改变哪一个存储执行状态能访问数据路径。这种技术在60年代早期首先被Seymour Cray用来利用单个、共享的数据路径在Control Data Corporation (CDC) model 6600上实施10个I/O控制器（称为周边处理器或“PPU”）。

注意这种逻辑处理器状态切换是严格地按照时间发生的，且不响应于外界事件。实际上，Control Data 6600 PPU 的某些继承者在

他们的逻辑处理器上实施了一种优先级中断方案。更近些时候这种数据通路共享技术被应用于中央处理器，其中它被称为“共享资源多路处理”。在此情况下，来自不同的CPU任务或程序的多个独立的指令流被交叠起来，以减小流水线依赖性，从而改进超标量数据通路的资源利用率。

因此，在该技术中需要一种具有更为一般的灵活性的配置、分配和管理历境的方式。

为了解决现有技术的上述缺陷，本发明提供了一种管理处理器中的多任务处理的历境控制器及其控制方法。在一个实施例中，该历境控制器包括：（1）前景和背景任务控制器，它们分别把处理器资源分配给与前景和背景任务相应的活动历境；以及，（2）模式切换电路，它与前景和背景任务控制器耦合，并当所有历境都不活动时把处理器置于一种闲置状态和一种省电模式。

本发明因而引入了一个广泛的概念，即当处理器所能够处理的所有任务都不活动时自动进入一个省电模式。随着具有基于硬件省电模式的处理器的发展，在不需要软件检测实现省电的机会的情况下较早自动进入该模式能够保存较多的电力并能够简化控制软件。

在本发明的一个实施例中，一个软件切换操作区分前景任务和背景任务。在所示和描述的实施例中，一个软件切换被包含在与各个历境相联系的一个任务可编程寄存器中。就本发明来讲，“历境”被定义为可用于把处理器恢复到一个给定状态的所有的处理器状态信息（或其任何子集，诸如寄存器值）。历境控制器检测切换的状态（0或1）以确定是否相联系的任务是前景任务还是背景任务。当然，前景和背景任务的指定可在硬件中作出，但灵活性会受到影响。

在本发明的一个实施例中，前景和背景任务控制器只把处理器资源分配给活动历境。在此实施例中，不活动的任务不被分配任何处理器执行时间。或者，不活动的任务可被分配一些很少的执行时间，而使效率受到一定的影响。

在本发明的一个实施例中，背景任务控制器根据各个背景任务所执

行的指令的数目而在与背景任务相应的历境之间切换。这在此被称为“指令片（instruction slice）”。背景历境的执行也可以根据时间（“时间片（time slice）”）来进行。当然，用于周期激活的其他依据也处于本发明的范围之内。

在本发明的一个实施例中，各个历境的状态被存储在单独的寄存器组中。某些处理器结构支持多物理寄存器组，在任务被切换时把逻辑寄存器重新映象到其上。或者，可以用主存储器的部分来存储不同集中的寄存器内容。

在本发明的一个实施例中，前景任务控制器根据优先级并响应于事件而激活与前景任务相应的历境，且背景任务控制器服从于与前景任务相应的历境的激活，周期性地执行与背景任务相应的历境。“事件”被定义为能够使历境控制器通过从一个前景任务切换到另一任务而作出响应的刺激。任务可被分成前景和背景任务并按照非常不同的标准分配处理器资源。当然，前景任务也可根据时间片得到处理，也可以根据指令计数。

在本发明的一个实施例中，前景任务控制器适合于通过指向一个软件可选择存储位置，而激活与具体的前景任务相应的历境。通过允许具体的前景任务的进入点发生变化，可以建立一个状态机—在其上历境激活的初始执行地址也被作为状态表示符，从而使前景任务能够作为引起其执行的事件的函数而得到执行。

以上相当概括地列出了本发明的较好的替换特征，从而使本领域的技术人员能够对以下进行的本发明的详细描述有更好的理解。以下将描述本发明的其他的特征—它们构成了本发明的主题。本领域的技术人员应该理解的是，它们能够方便地利用所公布的概念和具体实施例作为设计或修正用于实现与本发明相同的目的的其他结构的基础。本领域的技术人员还应该理解的是，这些等价的结构并未脱离本发明的精神和范围。

通过以下结合附图进行的描述，可以对本发明得到更完全的理解。在附图中：

图1显示了一个状态转换图， 从一个单独的历境的角度显示了本发明的一个处理器的一个实施例的操作;

图2 示例性地显示了在五个前景历境和三个背景历境上运行的一个处理器上的可能的处理流、抢先、以及历境间通信;

图3 示例性地显示了在采用本发明的一个实施例的处理器中执行的软件所能够存取的每个历境的控制和状态寄存器;

图4显示了包括本发明的历境控制器的一个实施例I/O控制器或典型处理器的系统图;

图5显示了一个相互作用图，显示了图4中显示的历境控制器的内部结构;

图6显示了图5显示的事件同步处理的处理图;

图7A、7B、7C和7D一起显示了图5所示的事件优先化处理的处理图;

图8显示了由本发明控制的历境切换的时序图， 其中当前的历境的状态被存储到同步（自定时）SRAM或寄存器文件中，且下一个历境的状态从该SRAM或寄存器文件装载;

图9显示了本发明控制的历境切换的时序图，其中当前的历境的状态被存储到异步SRAM或寄存器文件中，且下一个历境的状态从该SRAM或寄存器文件装载;

图10显示了用于实施各个激活事件的事件记录、事件掩码和事件确立、以及历境激活位的管理的电路的一个实施例的示意图，包括初始化请求和等候请求逻辑;

图11显示了根据本发明的一个实施例的与历境控制和指令集中的历境间通信有关的机器指令的字段和位分配;

图12显示了根据本发明的一个实施例的用于产生处理器上的控制存储地址的位的源;

图13 显示了根据本发明的一个实施例的控制存储中用于初始化矢量的示例性数据结构图;

图14显示了根据本发明的一个实施例的用于优先化和解码在处理器

上的具体历境激活位的矢量指令的目标地址产生的设定图；

先参见图1，其中显示了一个状态转换图， 它从单个历境的角度显示了本发明的处理器的一个实施例的操作。

本发明提供了一种管理处理器中的多任务处理的历境控制器，该历境控制器包括：前景和背景任务控制器，它们分别把处理器资源分配给与前景和背景任务相应的活动历境；以及，模式切换电路，它与前景和背景任务控制器耦合，并当所有历境都不活动时把处理器置于一种闲置状态和一种省电模式。

本发明因而引入了一个广泛的概念，即当处理器能够执行的所有任务都不活动时，自动进入一个省电模式，而不需要软件检测这种进入的可能。随着具有基于硬件的省电模式的处理器的发展，能够较早地自动进入节电模式从而能够节约更多的电力。

在本发明的一个实施例中，前景任务控制器根据优先级并响应于事件而激活与前景任务相应的历境，且背景任务控制器服从于与前景任务的历境的激活，周期性地执行与背景任务相应的历境。“事件”被定义为能够使历境控制器通过从一个前景任务切换到另一任务而作出响应的刺激。任务可被分成前景和背景任务并按照基本不同的标准分配处理器资源。当然，前景任务也可根据时间片得到处理，也还可以根据指令计数得到处理。

在处理器工作的任何给定时刻，各个历境处于六种状态中的一个，这些状态在逻辑上被分成2行乘2列（2×2）矩阵形式的四组。顶行或前景行10包含三种状态：前景历境使用的一个Rf状态18、一个Pf状态20和一个Wf状态22（其中每一个都包括一个“f”以表示前景）。底行或背景行12 包含三种状态：背景历境使用的Rb状态24、Qb状态26以及Wb状态28（其中每一个都包括一个“b”以表示背景）。活动列14包括活动历境使用的四种状态18、20、24、26，而不活动列16分别包含不活动历境所使用的两个状态22和26。

前景行10的状态可进一步被定义为Rf 18（运行，前景）、Pf 20（抢先，前景）以及Wf 22（等候，前景）。背景行12 的状态可进一步

被定义为Rb 24（运行，背景）、Qb 26（排队，背景）以及Wb 28（等候，分支）。在每一个指令周期中，只有一个历境可以得到“运行”（在处理器上执行一个指令），或者处理器也可以处于闲置。如果被占用，运行的历境是在前景运行状态Rf下的唯一历境。或者如果状态Rf 18未被占用， 运行的历境是背景运行状态Rb 24（如果被占用）下的唯一历境。 历境的执行状态一般被存储在单独的寄存器组中。

多数历境转换是被允许在前景行或背景行12内发生的，因为行间转换只在历境在可由软件切换操作区分的前景和背景操作任务之间切换时才是需要的。在本发明的一个实施例中，一个软件切换操作区分前景任务和背景任务。在所示和描述的实施例中，一个软件切换被包含在与各个历境相联系的一个任务可编程寄存器中。就本发明来讲，“历境”被定义为可用于把处理器恢复到一个给定状态的所有的处理器状态信息（或其任何子集，诸如寄存器值）。历境控制器检测切换的状态（0或1）以确定是否相联系的任务是前景任务还是背景任务。当然，前景和背景任务的指定可在硬件中作出，但灵活性会受到影响。

然而，这发生得不如历境激活、抢先和等候频繁。从前景至背景的转换只在运行的前景历境Rf 18执行一个CLRFG（“清除前景”）功能34...它引起从前景运行状态Rf 18至背景排队状态Qb 26的转换—时才可发生。由于背景历境之间没有相对优先级区分，给予执行CLRFG功能34的历境在背景排队中的位置是任意的。

执行CLRFG功能34的一个历境将离开前景操作，并且最好放弃对处理器的控制最少一个指令周期（就象执行WAIT功能32或42的历境一样）。如果较低优先级前景的历境处于抢先状态Pf 20， 该较低优先级前景历境将接着运行（经过一个最高优先级转换36）。如果抢先状态Pf 20未被占用，已经在背景状态Rb 24中的一个抢先历境将接着运行， 除非背景状态Rb 24也未被占用。在此情况下，在背景排队状态Qb 26下的背景排队的开头的历境，经过一个时间片启动转换44，将接着运行。在所示的实施例中，这在处理器闲置的单个指令周期之后发生，因为前景运行状态Rf 18和背景运行状态Rb 24都未被占用。

背景和前景之间的转换通常是当处于背景运行状态Rb 24 的历境执行一个SETFG（设定前景）功能30时发生的，这导致其从背景运行状态Rb 24至前景运行状态Rf 18的转换。 特定历境的前景激活也可通过指向到一个可软件选择的存储位置而发生。

在本发明的一个实施例中，前景任务控制器适合于通过引向一个软件可选择的存储位置而激活与一个具体的前景任务相应的历境。通过允许具体前景任务的进入点发生变化，能够建立一个状态机，从而允许前景任务作为引起其执行的事件的函数而进行执行。当然，对于背景任务的激活，也可发生相同的状态机过程。

为了防止历境操作被错误破坏，在历境控制器中最好不包括这样的机制，该机制使一个正在运行的历境改变任何其他历境的前景或背景设定而不强迫该历境初始化（INIT）。该INIT功能可由运行的历境以任何其他的历境作为目标而执行。一个INIT功能可对正运行的历境执行，但不存在这样做的理由，除非一个具体的实施例附属额外的初始化副作用给这种INIT功能。INIT功能的执行使目标历境处于前景抢先状态Pf 20， 且其程序计数器被置于预定的初始化矢量地址，如将在后面所详细描述的。

在正常情况下，一个INIT功能的目标位于前景等候状态Wf 22， 并经过一个转换40进入前景抢先状态Pf 20。或者， 它可以位于背景等候状态Wb 28并进入前景抢先状态Pf 20，从背景经过一个转换50切换至前景。实际上，转换50也是可能和等价的—如果目标历境位于背景运行状态Rb 24或背景排队状态Qb 26，但图1未显示这两种情况。

在处理器复位结束时，所有历境都处于前景等候状态Wf 22， 但最低优先级历境除外—它处于前景运行状态Rf 18。在历境前景运行状态Rf 18上执行的软件可通过执行一个WAIT 功能32 而启动至历境前景等候状态Wf22的一个转换。一个前景等候状态Wf 22历境， 在确立该历境被该历境的事件掩码所使能的的任何激活事件时，或当运行的历境执行经过转换40至该历境前景抢先状态Pf 20的INIT功能时，转换至前景抢先状态Pf 20。

在所示的实施例中，在每一个指令周期结束时，都会发生一个抢先历境切换，其中在前景抢先状态Pf 20 中的最高优先级历境（如果有的话）经过最高优先级转换36进入前景运行状态Rf 18， 且在前景运行状态Rf18 中的前面的历境（如果有的话）经过一个较高优先级活动转换38而进入前景抢先状态Pf20。

在历境背景运行状态Rb 24下执行的软件可通过执行一个WAIT 功能42而启动至背景等候状态Wb 28的一个转换。在背景等候状态Wb 28下的一个历境，在确立了该历境被该历境的事件掩码所使能的任何激活事件的情况下，转换至背景排队状态Qb 26。从背景排队状态Qb 26 至背景运行状态Rb 24的转换，只可在没有前景历境运行（无历境处于状态Rf 18）时发生。在此情况下， 运行的历境（如果有的话）处于背景运行状态Rb 24，或者处理器处于一种闲置状态，因为没有历境准备好在背景或前景下运行。

在每一个指令周期结束时，在历境运行在背景状态Rb 24的情况下， 时间片计数被减少，且最好在该计数达到零的指令周期时发生一个时间片历境切换。此时，在背景排队的开头的历境经过转换44进入背景运行状态Rb 24，且先前处于背景运行状态Rb 24的历境经过转换46进入背景排队状态Qb 26。

一般地，背景排队历境被组织成一种先进先出（FIFO）排列，其中当前面运行的背景历境进入背景排队状态Qb 26 时进行从最高历境号向最低历境号的“绕回”。应该注意的是，前景抢先涉及经过转换36的状态转换，而借助前景的背景抢先则不是。在此情况下，前面运行的背景历境保持在状态Rb 24，直到前景运行状态Rf 18重新未被占用且一个背景历境能够运行。

现在参见图2， 其中示例性显示了在运行有五个前景历境和三个背景历境的处理器上的一种可能的处理流、抢先和历境间通信。在本发明的一个实施例中，前景和背景任务控制器只把处理器资源分配给活动历境。在此实施例中，不活动的任务不被分配任何处理器执行时间。或者可以给不活动的任务分配某些少量的执行时间。

一个历境可通过确立一个事件信号而得到激活。与各个历境相联系的可以是零或更多的外界事件信号和零或更多个内生事件信号。外界和内生事件信号的主要不同，是外界信号在被用于历境控制器之内的历境激活判定之前要先与处理器的时钟同步。相比之下，内生信号被假定是与处理器的时钟同步产生并直接使用的。

每一个历境激活事件都可通过设置和清除具体历境的事件掩码寄存器而在软件控制下得到使能和禁止。除了由于确立来自外界源（诸如外部接口）或来自内生源（诸如间隔定时器、协处理器或数据传送逻辑电路）的硬件信号的激活事件确立，某些或所有事件，可利用指定目标历境号和处于与目标历境相关的事件集内的事件数的信号指令，由软件确立。由于任何历境都可向其自身或向其他历境表示事件，这使得所示的实施例能够作为有效的历境内和历境间通信机制，以及作为优先级中断控制器和时间片控制器。

在本发明的一个实施例中，背景任务控制器根据各个背景任务所执行的指令的数目而在与背景任务相应的历境之间切换。这在此被称为“指令片”。历境的执行也可以根据时间（“时间片”）来进行。当然，用于周期激活的其他依据也处于本发明的范围之内。

在图2中，纵轴表示历境， 而横轴表示示例性的历境控制器所支持的八个历境的每一个的历境活动。横轴是时间，其单位是指令执行周期。用于前景历境的宽黑线和用于背景历境的宽交叉影线显示了运行的历境。带有箭头的纵向线显示了历境切换并用于识别历境切换发生的原因。与宽线交叉的小的垂直线表示指令周期。在用于背景历境的各个指令周期间隔之上的数是当该指令被执行时时间片指令计数器的值。用于前景历境的窄的黑虚线和用于背景历境的窄的交叉影线虚线，显示了活动的抢先历境。窄的点线显示了活动的、排队的背景历境。该实施例有八个历境， 用历境0（最高优先级）至历境7（最低优先级）表示， 且在这个例子中是以每时间片8个指令运行的。

在该例子开始时，历境0、2、4和5都是不活动的前景历境（状态Wf）。历境3、6和7都是背景历境，其中历境3是不活动的（状态Wb），

历境7 在排队（状态Qb）和历境6正在运行（状态Rb）。

历境1是不活动的，并具有一个未知（或未确定）的前景/背景设定。所示的一个第一指令周期46由背景历境6 在其时间片计数值减小至二时执行。在下一个指令周期47，背景历境6执行至背景历境3的SIGNAL功能。其结果，背景历境3 在随后的指令周期变成了活动进入状态Qb 。 在送出了SIGNAL功能之后，背景历境6在其时间片计数减至0的同时执行另一指令周期48。这使得历境切换到活动历境背景排队状态Qb中的次最高历境号，即背景历境7。在指令周期50，在时间片计数值为7时，历境6进入Qb 状态且历境7进入Rb状态。在历境7已经执行了三个指令之后，一个外界事件激活前景历境4。因此，在下一个指令周期52结束时，背景历境7 被前景历境4所抢先，而其时间片计数值在抢先期间仍然是四。

前景历境4在一个外界事件激活前景历境2的同时执行其第一个指令。因此，在下一个指令周期54结束时，在前景历境2进入运行状态Rf的同时前景历境4被进入抢先状态Pf的前景历境2（处于抢先点58）所抢先。在执行了两个指令以处理其激活事件之后，前景历境2 在一个第三指令周期56期间执行一个WAIT功能。该WAIT功能清除了前景历境2的活动触发器， 且在又一个指令周期之后，前景历境2变成不活动并返回到等候状态Wf。 这使得被抢先的前景历境4返回到运行状态Rf并执行另一指令周期58。 由于前景历境4已经在抢先点53之前执行了其自己的WAIT功能， 这是前景历境4在返回到等候状态Wf并允许被抢先的背景历境7在指令周期60恢复运行之前执行的最后指令。在又执行了四个指令之后，背景历境7 完成其时间片62，由于从历境7至历境0的历境号绕回而导致至下一个顶级Qb历境—它是背景历境3—的历境切换。

在相同的指令周期64期间，背景历境3执行其时间片7的第一个指令，一个外界事件66激活前景历境0。因此，在该指令周期64结束时，背景历境3被前景历境0所抢先，而其时间片计数值在抢先期间保持为七。在执行了三个指令以处理其激活事件之后，前景历境0在一个第四指令周期69 期间执行一个WAIT功能。该WAIT功能清除了用于前景历境0

的活动触发器， 且在又一个指令周期之后前景历境0变成不活动的，返回到等候状态Wf。这通常使得被抢先的背景历境3能够恢复运行， 但在此例子中一个外界事件68已经在前景历境0运行的同时激活了前景历境5。注意这种激活把前景历境5的状态从等候状态Wf改变到抢先状态Pf，显示了前景历境在从激活起不执行任何指令的情况下进入抢先状态的方式。

如果背景历境3是在前景下运行的，则前景历境5在前景历境0 返回到等候状态Wf时处于抢先状态Pf这一事实是无关的，因为背景历境3 是优先级比前景历境5高。然而，历境3是在背景运行的，所以前景历境0所执行的一个WAIT功能69导致了至前景历境5的历境，而前景历境5进入运行状态Rf并在背景历境3在状态Rb保持在被抢先的同时开始执行一个指令70。

在执行了两个指令以处理其激活事件之后，前景历境5 在一个第三指令周期71期间执行一个WAIT功能。该WAIT功能清除了活动触发器f 或前景历境5，且在又一个指令周期之后，历境5变成不活动的并返回到等候状态Wf。由于此时没有其他活动的前景历境，抢先的背景历境3在状态Rb 恢复运行并执行其时间片72的第二个指令。在下一个指令周期，背景历境3执行一个WAIT功能73。WAIT功能73清除了用于背景历境3的活动触发器， 且在又一个指令周期之后，背景历境3变成不活动的，返回到等候状态Wb。这使得排队的背景历境6能够返回到一个指令周期74的运行状态Rb。注意，即使这种历境切换不是在时间片计数减小到零时启动的，背景历境6 以计数值为7的全时间片在指令周期74进入运行状态Rb的， 而不是继承背景历境执行WAIT功能73时剩余的部分时间片。

作为其第二个指令，背景历境6执行至前景历境1的INIT功能76，以迫使背景历境1进入一个已知的状态—该状态可以是对历境1所执行的编码中的软件错误进行恢复所必需的。这种INIT功能激活历境1作为一个前景历境抢先状态Pf，并设置为在控制存储器中的历境1的初始化矢量地址处开始执行。 由于现在存在一个活动的前景历境，背景历境6 在执行了又一个指令之后经一个历境切换被历境1所抢先（在一个抢先点

77）。 作为其第二个指令，历境1执行一个CLRFG（清除前景位）功能78—它使历境1 进入背景排队状态Qb。由于历境1现在处于背景排队且状态Rb已经有一个历境，历境1在执行CLRFG功能78的指令周期之后放弃对处理器的控制（在放弃点80），从而使状态Rb的历境6能够恢复执行其时间片82的剩余部分。

在该详细描述的剩余部分，数字常数是十进制的，除非其前面有“0x”—在此情况下它们是十六进制的，且位位置被编号，零位处于最低有效位。

现在看图3， 其中显示了采用本发明的一个实施例的处理器中执行的软件所能够存取的示例性的历境相关控制和状态寄存器。每个历境84的九个控制位具有由软件确定的值，且每个历境86的九个状态位具有由历境控制器硬件确定的值，但这些值可以由软件以其他的方式读取或测试。历境控制器保持各个历境的状态的一部分。这些状态位不是执行状态（执行状态在历境切换期间得到保存和恢复）的一部分，因为历境控制器内的争对历境的状态必须是连续的，以被活动的逻辑电路所使用并作为至历境切换判定逻辑的输入。

历境相关控制位84包括一个前景（FG）位88和一个事件掩码寄存器90。FG位88在历境处于前景时等于一。该FG位88在显示中是作为被INIT功能的硬件复位执行所设定的—而该历境被作为指定的目标，或者是在该历境正在运行时被SETFG功能的执行所设定。FG位88 被显示为在历境正在运行时被CLRFG功能的执行所清除。事件掩码寄存器90 有一个位与同历境相关的各个激活事件相应。

在所示的实施例中，各个历境被分配有八个激活事件；因而事件掩码寄存器90包含八个位。给定的激活事件只在等于事件数的相应的位位置数具有历境事件掩码寄存器90中的一个值时才能够激活一个历境。然而，如将在下面详细描述的，一个激活事件的确立被记录在一个事件触发器中，该触发器保持被设定直到为指定的位执行一个确认（ACK）功能。 事件触发器的设定不受事件掩码寄存器90的内容的影响。

历境相关状态位86包括一个ACT位92和一个事件状态寄存器94。ACT

位92在历境活动时等于一。ACT位92被一个非掩码的激活事件的确立、一个被确立的未确认激活事件的事件掩码位的设定或以该历境作为指定目标的INIT功能的执行所设定。ACT位92被硬件复位（历境7除外，在那里ACT位被硬件复位所设定）和在历境运行的同时执行WAIT功能所清除。 事件状态寄存器94具有与同历境相关的各个激活事件相应的一个位。这些位在被详细描述的某些部分也被称为事件触发器。

如上所述，在所示的实施例中，各个历境都被分配了八个激活事件，要求事件状态寄存器94必须包含至少八位。与读取的确立的事件相应的位等于一，且与读取的未确立事件相应的位—包括确认的事件，等于零。各个事件状态寄存器位（事件触发器）在检测到外界或内生事件信号的确立（通常是零至一的转换）时被历境控制器硬件设定。各个事件状态位也可以在执行把在此历境中的对象事件指定为目的地的SIGNAL功能时得到设定。各个事件状态寄存器位，在这种历境正在运行的同时，在对象事件作为指定的目标的情况下，通过硬件复位或通过执行一个ACK功能而被清除。 在某些情况下，一个具体的ACK 功能也可作为执行其他指令或存取具体的数据通路（通常是I/O端口）寄存器的副作用而得到产生。

以下提供了一个实施的例子， 它显示了历境定义和IEEE 802.11媒体存取控制(MAC)控制器的用途。MAC控制器的功能已经被分成了八个历境，用0至7表示，其中0是最高优先级。历境0至5 较好地是前景且6和7较好地是背景。各个历境都具有八个激活事件且这些激活事件每一个通常都应用以下的缺省:

A. 不能利用SIGNAL功能来确立一个事件（除非该事件是专为此目的而保留的）;

B. 一个事件利用ACK功能而得到清除;

C. 当相应的定时器减小至零时发生了定时器终止计数事件;

D. 定时器终止计数事件，通过把等于一而不是ACK功能的ClearTC（位2）写入相应定时器的控制寄存器，而得到清除;

F. “一个外界事件信号的“确立”被定义为0至1的转换;

G.外界事件信号的“非”操作意味着1至0的转换；且

H.控制位名称得到选择以在位等于一时有意义。

这些示例性的历境和它们相应形成激活事件在下面得到描述。

历境0-Debug支持（及高优先级、实时事件）：

激活事件：

0）硬件断点（BKPTin）；

1）软件断点（信号0，1）；

2）GP串行移位完成或UART发送器完成（GPDN/UTXDN）；

3）间隔定时器A终止计数（INTATC）；

4）UART接收器完成（URXDN）；

5）间隔定时器B计数（INTBTC）；

6）主机（计算机系统）注意（HATN）；以及

7）协处理器注意（CPATN）。

历境1-较低MAC（LMAC）例外处理

激活事件：

0）调制解调器数据接口注意（MDIATN）；

1）物理层数据不能获得（！PDA）；

2）IFS（帧间空间）定时器终止计数（IFSTC）；

3）来自MMAC至LMAC的历境间通信；

4）物理层发送器未准备好；

5）beacon/dwell定时器比较器相等（BCNTC）；

6）调制解调器数据接口可编程位边界（MDIBIT）；以及

7）调制解调器管理接口传送完成（MMIDN）；

历境2-较低MAC（LMAC）数据传送：

激活事件：

0）调制解调器数据接口注意（MDIATN）；

1）间隔定时器B终止计数（INTBTC）；

2）IFS（帧间空间）定时器终止计数（IFSTC）；

3）来自MMAC至LMAC的历境间通信（信号2，3）；

4）TSFT（同步功能定时器）绕回（TSFWRP）；
5）NAV（网络分配矢量）定时器终止计数（INTCTC）；
6）物理层介质繁忙（MBUSY）；以及
7）物理层介质不繁忙（！MBUSY)。
历境3-主机接口支持:
激活事件:
0）缓存器存取路径0偏离（offset）分辨率（BUFATN0）；
1）缓存器存取路径1偏离分辨率（BUFATN1）；
2)用于至主机的状态报告的历境间通信（信号3，2）；
3）缓存存取路径0块边界交叉（BLKATN0）；
4）缓存存取路径1块边界交叉（BLKATN1）；
5）用于至主机的状态报告的历境间通信（信号3，5）
6）主机接口寄存器注意（HATN）；以及
7）从背景的历境间通信（信号3，7）；
历境4-中间MAC（MMAC）介质存取和时序:
激活事件:
0）来自LMAC或HMAC的历境间通信（信号4，0）；
1）原来繁忙的介质变得可用了（MAVL）；
2）IFS/时隙定时器终止计数（IFSTC）；
3）间隔定时器A终止计数（INTATC）；
4）beacon/dwell定时器比较器（BCNTC）；
5）介质数据接口注意（MDIATN）；
6）软件标记3-0（与历境7、事件7共享）；以及
7）调制解调器管理接口传送完成（MMIDI）。

历境5-WEP（有线等价保密）解码支持:
激活事件:
0）用于状态报告的历境间通信（信号5，0）；
1）解码密钥流（keystream）值准备好（DECRYPT）；

2）GP串行移位完成或UART发送器完成（GPDN/VTXDN）；
3）历境间通信（信号5，3）；
4）UART接收器传送完成（URXDN）；
5）历境间通信（信号5，5）；
6）间隔定时器D终止计数（INTDTC）；以及
7）调制解调器管理接口传送完成（MMIDN）。

历境6-附加的存取点功能:
激活事件:
0）软件标记11-8;
1）软件标记15-12;
2）GP串行移位完成或UART发送器完成（GPDN/UTXDN）；
3）间隔定时器A终止计数（INTATC）；
4）软件标记7-4;
5）间隔定时器B终止计数（INTBTC）；
6）间隔定时器D终端计数（INTDTC）；以及
7）协处理器注意（CPATN）。

历境7-上MAC（UMAC）和其它支持;
激活事件:
0）软件标记19-16;
1）软件标记23-20;
2）软件标记27-24;
3）间隔定时器A终止计数（INTATC）；
4）beacon/dwell定时器比较器（BCNTC）；
5）间隔定时器B终止计数（INTBTC）；
6）间隔定时器D终止计数（INTDTC）；以及
7）软件标记3-0（与历境4、历境6共享）。

现在参见图4， 其中显示了包含本发明的历境控制器的一个实施例的普通处理器或I/O控制器的系统图。该图（以及图5、6和7中显示的）是利用国际通信联盟在ITU-T Recommendation Z.100（03/93 ）标准化的说明书与描述语言9SDL）的众所周知的图形句法而提供的。

该系统行为是利用这种正规的描述语言给出的，因为能够实现更为准确而宽广的一般应用。例如，一个示意的部分可被用来强调所示的实施例的实施特性。然而，由于这种历境控制器可应用于几乎任何类型的处理器，借助一种具体处理器的示意表示可能忽略对于该种处理器来说是暗指的而对于采用一种不同结构的另一种处理器来说可能是相关的控制序列方面。另外，一种传统的状态图是一种更不正规的表示并具有与SDL 过程图类似的目的。SDL具有严格定义的图形句法且其任意性较小。实际上， 已经发现很多这种控制器的行为中的“边界条件”用传统的状态图没有得到适当的解释。这种边界条件的例子（它们都在此用SDL描述覆盖）表示: （1）如果在执行WAIT功能与执行WAIT功能之后的指令之间历境被抢先将发生什么？（2）如果一个历境在执行WAIT 功能之后的指令期间执行了造成其激活的事件的ACK功能，将发生什么？以及，（3）如果背景历境的时间片在它执行一个SETFG功能的同一指令周期上结束， 该历境是否继续在前景上运行，或者处于状态Qb的下一个历境在被新的前景历境抢先之前执行一个指令？另外，SDL能够比用英语散文更准确而明确地描述历境控制器的行为。因此，以下给出的SDL描述被用作对本发明的若干个实施例的重要特征的结构和目的的一般和详细说明。

SDL系统图100显示了所示实施例中采用的处理器的相关顶级功能框。文本符号102和104包含了有关系统的至SDL预定数据类型的扩展、用于经过出口/ 进口机制的隐含框间通信的遥程变量的声明和用于显含框间通信的信号的名称和参数类型。系统图100 包括五个功能框: 一个时钟发生器106、一个定序器108、一个指令解码器112、 一个数据通路与接口资源管理器114和一个历境控制器110。

时钟发生器106，经过ClocksIn信道122接受一个输入时钟或时基基

准（例如晶体控制下的信号）—从其产生一个时钟信号，并经过ResetIn信道120接收一个硬件复位信号。时钟发生器106 产生所有其他框使用的周期时钟信号。这些周期时钟信号把指令周期分成四个基本相等的部分。这是利用一对正交 方波得到实现的， 产生了四个时钟边缘—在这些边缘启动各种动作。实际的时钟波形在图8和9中显示，其中一个主时钟MCLK信号504 限定了指令周期的边界且一个正交时钟QCLK信号506 提供了各个指令周期内的额外时钟边缘。这四个边缘依次是：以Mr 517表示的MCLK信号504 的上升边缘，它标志了一个指令周期的结束和下一个指令周期的开始；QCLK信号506的上升缘，用Qr 518表示并发生在通过各个指令周期25%处；MCLK信号504的下降缘，用Mf 519表示，它发生在通过各个指令周期的50%处；以及，QCLK信号506的下降缘，用Qf 520表示， 它发生在通过各个指令周期的75%处。

在SDL模型中，时钟发生器106把适当的Mr 517、Mf 519、或Qf 520信号以及一个复位信号送到所有其他的框。时钟发生器106 在处理器运行或闲置的同时工作， 但可以在非常低电力睡眠模式（它是在时钟发生器106接收到经信道ClkCct1 140而来自历境控制器110的一个睡眠信号时进入的）下关闭其大部分电路，包括MCLK信号504与QCLK信号506的产生。

在很多实施中，不能在每一个时钟周期中执行一个指令。其结果，指令解码器112、定序器108和历境控制器110只在当指令被实际执行时的周期中执行它们的功能，如遥程布尔变量“ien”为真所表示的（见文本符号102）。

定序器108产生指令地址并启动经过一个ToCS信道116的指令提取周期。这些地址连接到一个在逻辑上处于处理器100之外的控制存储阵列117。注意，根据实施技术和所希望的性能等级，控制存储阵列117 和相关的数据存储127可以是在物理上分离的、在单个的存储器件中处于同一单元中的、或者它们的任何混合形式。定序器108经过Cct1Seq信道141 从历境控制器110 接收历境切换信号（CsLoad （用于恢复保存的历境状态信息）、 CsStore（用于保存历境状态信息）以及InitSeq

（用于把一个历境执行地址置入适当的初始化矢量）。

指令解码器112在定序器108的控制下经过一个FromCS信道118 接收所提取的指令字。解码的指令作为信号—其中指令字段值被作为参数—被送到所有其他适当的框。 历境控制器110 中的的处理所要求的指令经过一个InstCctl信道142而得到发送。

数据通路与接口资源管理器114表示了处理器的其余部分，包括ALU、程序员可见寄存器等等。I/O装置、 主计算机（如果有的话）以及本地数据存储器接口（信道126、128、130、132）都与该功能框相连。数据通路与接口资源管理器114把事件信号送到历境控制器110 并经过一个CctlIDP信道143从历境控制器110接收一个AckEv 信号（它表示软件已经执行了一个ACK功能以确认一个具体的在前事件）、IsLoad和CsStore信号（以恢复和保存历境状态信息）、以及SetCy和ClearCy信号（用于设定和清除硬件复位和INIT功能之后所用的运送标记）。这种功能框还输出ien 值（如果当前时钟周期是一个指令执行周期则为真）和片（软件为各个背景时间片的初始指令计数指定的最后一个值）。

历境控制器110有利地经过一个EventsIn信道124而接受外界事件信号，并与上述的其他功能框进行通信。这种功能框还输出布尔变量正在睡眠的值（当处于睡眠模式时为真）、CSW （在历境切换周期的下半部分为真）以及闲置（当没有活动的历境时为真）、CtxNum（历境数）、变量历境（运行的历境号）、以及nctx（执行所 切换至的历境号）。且这种功能框还输出BitString 变量事件（当前历境的事件状态寄存器）和掩码（当前历境的事件掩码寄存器值）。

现在参见图5，其中显示了一个SDL过程作用图，显示了图4 所示的历境控制器110的内部结构。其他顶级 框的内部结构在此没有给出，因为它们不是本发明的部分且不是理解历境控制器110的行为所需的。

在所示的历境控制器框110中包含了两个处理。一个事件同步器150从一个AsyncEvents信号路由158接收外界事件信号并让它们与主时钟上升缘Mr 517同步，主时钟上升缘由时钟发生器106经过一个ClkSyn信号

路由156提供。这些事件经过一个SyncEvents信号路由166而作为事件信号被传送，就象在一个PriDP信号路由164上从内生源传送事件信号一样。

在此实施例中，基本的历境控制状态机在一个事件优先处理152 中工作。事件优先化器152接收经过一个ClkPri信号路由154来自时钟发生器106的输入信号，经过一个SyncEvents信号路由166来自事件同步器150 的事件信号和经过一个PriDP源164的数据通路CctlDP功能143。另外，还经过InstCctl信道142 和一个InstPri信号路由162从指令解码器接收各种与历境控制及历境间通信有关的信号。

现在参见图6，其中显示了描述事件同步器150的操作的图5 中显示的事件同步处理的处理图。该处理保证了各个进入的ExtEvent信号208被保存至发生主时钟上升缘Mr 206—在这一时间所有保存的ExtEvent信号214 都得到接收并立即作为Event信号218而被传送到事件优先级设定器152。

现在参见图7A、7B、7C和7D，它们一起显示了图5 中显示的事件优先级设定处理的处理图，限定了事件优先级设定器152处理的状态转换。 这种处理实现了本发明的该实施例的事件驱动和时间片历境切换功能。

图7A定义了启动和复位序列。在“所有状态”符号272， 一个复位信号274优先于所有其他的输入信号发生并使得处理输入排队（符号276-280）在参加从一个启动符号254开始的启动初始化（符号282）之前被清空。一个序列（符号256-270）对所有相关变量进行初始化，清除事件掩码、 事件状态寄存器和等候触发器、把所有的历境置于前景，并清除除了历境7 的触发器（它被强迫处于活动状态）之外的所有ACT触发器。

图7B定义了各个周期的下半部和Mf至Mr期间（从主时钟的下降缘Mf至其下一个上升缘Mr）的操作，以及在主时钟的上升缘Mr 292的接收之后紧接着的事件。运行和闲置状态284都具有相同的转换， 因为一个指令是在接着一个WAIT功能的周期中得到执行的，且因为事件可在任何周

期中发生和得到处理，包括当处理器处于闲置状态的时候。在Mf至Mr期间，除了了一个ACK（AckInst）、一个WAIT或SLEEP功能300之外的所有指令解码信号立即得到处理。这三个信号被保存起来在主时钟上升缘Mr 292之后处理，因为它们必须在所有事件信号288已经被处理之后得到处理。 在主时钟上升缘Mr292之前得到处理的指令（即信号286、290、294、296、209），在历境切换发生的情况下，可改变在主时钟上升缘Mr292时保存的信息。

在主时钟上升缘Mr 292之后，在ien等于真（一）的周期（293），CSW（处理标记中的历境切换）、CTX（当前历境号）、NCTX（（下一个历境号）以及事件掩码和事件事件状态寄存器得到更新（符号320、321）。处理器可进入一个睡眠状态（符号338）—其间处理器时钟光圈， 且只有一个低频睡眠定时器工作直到一个Sleeping定时结束（一个Wake信号，符号340）或一个硬件复位发生。如果没有睡眠， 在背景历境正在运行的情况下（符号326、328），一个时间片指令计数被减值（符号330）。 如果片计数减值至零（符号332），一个时间片历境切换通过把循环curBg（当前背景历境）指针推进一而得到启动，对历境号取模（符号334）， 且时间片指令计数被复位（符号335）至其编程的值。 随后进入一个优先级设定状态336，以处理一个Mr至Qr时期（从主时钟上升缘Mr 至嵌套正交时钟上升缘Qr的时期）。

图7C定义了各个周期的第一个四分之一（Mr至Qr时期）的操作。这是当在主时钟上升缘Mr 292取样的被掩码且ACT 触发器在更新以为正交时钟上升缘Qr 380作准备时的时间。一个ACK（AckInst）信号352、 一个WAIT信号360和一个SLEEP信号366在正交时钟上升缘Qr 380之前得到处理， 且一个掩码和ACT更新序列（符号386-392）接着正交时钟上升缘Qr 380发生。

ACT位的更新被描述为澄清所进行的操作的重复的过程（符号388-392）。该操作通常是为所有历境并行进行的。图7C中一个微妙但重要的活动，是处理一个WAIT功能360，其中WAIT功能360的发生在前面（prev）历境（符号362）（它是当WAIT功能360被解码时在主时钟上升

缘Mr 292之前运行的历境）的指标处得到记录。随后ACT触发器的清除（符号382-384）在当前（ctx）历境的指标处进行。prev和ctx的值在除了当历境切换刚好在主时钟上升缘Mr 292 之前发生时之外的所有情况下， 在正交时钟上升缘Qr380之前与之后都是相等的。 这意味着在一个历境切换之前的最后一个周期的执行一个WAIT功能的历境仍然是活动的，但其Wait触发器（等候位串中的位）等于一直到该历境再次能够运行并执行WAIT功能之后的指令。图7C中另一个有意思的活动，是当一个ACK功能352被处理时一个AckEv 信号356被送到Data Path。这是为了允许装置或主机接口逻辑的副效果在一个具体的事件被确认时能够得到执行。

图7D定义了各个周期的第二个四分之一即Qr至Mf时期中的操作（从正交时钟上升缘Qr至下一个主时钟下降缘Mf的时期）。这是当事件被设定优先级且历境切换判定被做出的时期。首先组活动（符号422-428 ）搜索可能的抢先。该搜索被描述为澄清有关正在进行的操作的重复处理。这种操作通常是为所有历境并行地进行的。如果运行的历境处于前景，该搜索在范围0: ctx上进行，而如果运行的历境处于背景该搜索在范围0: 7上进行，因为所有的前景都具有高于背景历境的优先级（符号423）。 优先级编码器（符号424）是隐含在增大的历境号424（优先级降低）序列中的。如果发现了一个活动的前景历境，其号被记录在nctx（符号452）中，否则， 从所距离的当前背景历境开始对一个活动的背景历境进行搜索并继续进行到较高的历境号（模8）。

如果时间片（图7B 中的符号334 ）结束在该周期的主时钟的上升缘Mr292，表示的curBg将已经被增值，意味着搜索将从当前运行的历境之后的历境开始并在没有其他历境处于排队状态Qb的情况下将只重新选择相同的历境。在现在能够被恢复的背景运行状态Rb中的抢先历境的情况下，这种测试（符号430）将立即退出，以设定新的历境号(nctx)A450。 如果一个前景（设定新的历境号（nctx）452或一个背景（符号450）搜索发现了一个要运行的历境，新的历境号（nctx）被与当前的历境号（ctx ）（符号454）相比较以判定是否需要历境切换。

如果不需要历境切换， 在此周期期间不发生进一步的历境控制激活且控制器返回到运行状态458。

如果需要历境切换，控制器进入Start-CSW状态456 ， 保存输入信号462直到发生一个主时钟下降缘Mf（符号460）。随后CSW 得到确立（符号474-476），且保存的下一个历境的状态（符号478）的装载在当前历境状态（符号480）得到请求的同时得到启动。 装载在存储之前得到请求的原因将在下面结合 图8和9得到更为详细的说明。

如果没有活动的历境，控制器保存所有的输入信号（符号440 ）直到发生主时钟下降缘Mf（符号438），随后表示一个Idle 状态442 并请求保存实际进入一个Idle状态448之前保存当前历境状态446。历境状态得到保存，因为不能保证该相同的历境将是在闲置时期结束时运行的第一个历境。实际上，至和从Idle状态448的转换是分裂的历境切换， 在至闲置（符号442-446）的转换期间进行保存， 且在从闲置往回转换的期间进行装载（符号466-470）。在闲置状态下，时钟继续运行且事件继续得到取样， 但指令既不被提取也不被执行。

如果处理器是利用互补金属氧化半导体（CMOS）实施，或者是借助另一种处理技术—其中当电路元件没有被时钟信号作用或改变电平时电力消耗非常低或几乎为零，Idle状态448 为大多数处理器—包括定序器、指令解码器和数据通路—提供了一种固有的省电模式。如果希望更低的电力运行模式，SLEEP功能（图7C）可停止高速时钟信号并暂停事件监测， 只留下低频睡眠定时器进行工作。

现在参见图8，其中显示了本发明控制的历境切换的时序图， 其中当前历境的状态被存储到一个同步（自同步）SRAM或寄存器文件中且下一个历境的状态被从该SRAM或寄存器文件装载。图8和9中显示的时序图识别了采用用于存储非运行历境的执行状态的两种不同类型的存储技术中的每一种所需的差别。这些时序序列每一个都具有这样的好处，即历境切换操作不需要额外的周期来保存和恢复历境执行状态，而是与正在切换的历境的最后一个指令的执行并行地进行这种功能。为了采用这种技术，处理器数据通路应该包括执行状态下的各个寄存器所专用的寄存

器文件或静态RAM（SRAM）。 本发明的所示的实施例可与不提供这种存储的处理器数据通路结合使用。然而，由于可能的额外周期和执行额外的指令以存储和恢复历境执行状态，在这种处理器上的历境切换将有更多的开销。

当保存的阵列是利用同步静态（自定时）RAM（SRAM）实施时，图8所示的更简单的同步和控制信号序列得到了实现。这也是从根据图7 定义的SDL处理的直接实施获得的时序。虽然程序员可见的行为是相同的， 把异步静态RAM用于保存阵列（如结合图9所讨论的）要求更大的复杂性。采用同步SRAM的方法允许较短的周期时间和较低的电力消耗，因为信号转换的数目较小且消除了短于指令周期时间的50%的控制信号占空比， 假定同步SRAM与异步SRAM装置的性能相同的话。

同步SRAM在各个写入使能脉冲的前缘获得写入地址和数据，并利用内部产生的控制信号完成这种写入操作，而在写入周期的其余部分中不需要稳定的输入信号（除了电力）。基于电池的、采用同步SRAM的半常规集成电路—它采用同时采用具有独立地址的读取端口和写入端口的寄存器文件单元—是容易获得的。 用于历境切换的控制信号时序， 当采用这些同步SRAM单元以实施保存阵列时，变得比较简单，如图8所示。

在各个指令执行周期500、502中，历境控制器514 在主时钟上升缘Mr517对激活事件信号进行取样， 使得周期的第一个四分之一能够设定和选通同步的信号（时间间隔532）。在正交时钟的上升缘Qr 518，所有的ACT触发器都得到更新且优先级编码器和比较操作判定历境切换的需要，在需要时选择下一个历境（时间间隔533）。与这些历境控制器活动并行地， 一个处理器（时间间隔516）已经在执行在在主时钟上升缘Mr 517 开始的一个指令，而不管在这个指令执行周期里是否需要一个历境切换。如果一个处理器数据通路具有来自预期在整个执行周期中都是稳定的内部寄存器源的组合通路，这些通路的值必须在主时钟下降缘Mf 519得到锁存，以允许开始读出所保存的下一个历境的状态（时间间隔540）。或者， 如果处理器设计者更希望添加用于读出保存的历境状态的开销周期，这种锁存是不需要的。但是在多数情况下，一或多个

周期被插入，且纯效果将是在这些锁存被消除的情况下处理和实时响应的减慢，从而产生一个时期—其中在一个老的历境的最后一个指令周期与新的历境的第一个指令周期之间不能执行指令。

在主时钟下降缘Mf 519，历境控制器能够判定是否需要一个历境切换（时间间隔534），并在需要时确立一个CSW 信号522。所要恢复的目标状态通过设置在一个NCTX（2：0）信号组530 上的下一个历境的历境号来表示。这与当前历境（其历境号仍然在CTX（2：0）信号组524上）的最后一个指令的完成相并行地开始了下一个历境（时间间隔540 ）的一个“保存的状态”的读出，其中采用了一个NCTX（2：0）信号组512来寻址保存的阵列。

在主时钟上升缘Mr 517（把周期500与周期502分开）指定的历境切换周期的结束，通过利用一个CTX（2：0）信号组510来寻址保存的阵列，当前历境的一个执行状态—包括在该执行周期500 期间产生的一个结果—得到存储（时间间隔542）。该保存阵列写入操作（时间间隔542 ）， 是在CSW信号508被确立时（时间间隔522），由主时钟上升缘Mr 517启动的。 由于写入至同步SRAM的有利特性，下一个历境的第一个指令能够立即开始执行（时间间隔536）， 因为地址和正在写入到保存阵列的数据都不用在结束周期500的主时钟上升缘Mr 517发生之后得到保持。为了适当的执行，包括写入恢复的同步SRAM周期时间，可不超过指令周期的50%。 启动一个SRAM写入的相同的主时钟上升缘Mr517也可被有利地用于借助“非”的CSW信号508和更新至新的历境号526的CTX（2：0）信号组510来完成一个历境切换。

现在参见图9，其中显示了由本发明控制的历境切换的时序图， 其中当前历境的状态被存储到一个异步SRAM或寄存器文件，且下一个历境的状态从该SRAM或寄存器文件装载。传统的，或异步的SRAM要求写入地址和数据在写入周期的相关部分中都是稳定的。这要求在写入使能脉冲的下降缘之前有一个设定时间，且有时要求在该下降缘之后有一个短的保持时间。很多半常规的集成电路技术能够提供采用提供可用于读出或写入的单个地址和数据端口的异步SRAM的RAM阵列或寄存器文件。 以这

种方式工作的单独的SRAM和寄存器文件芯片也是可广泛地获得的。

为了使用这种传统的单端口SRAM来实施保存阵列，用于历境切换的控制信号同步变得有效复杂了，如图9所示。一般的时序与图8中的相同，且类似的元件用相同的标号表示。主要的不同是NCTX（2：0）信号组512 在工作中借助历境控制器514的产生，和在确立了CSW信号548 期间和刚好之后的数据通路516（如图9中的时刻522、528、530、534、535、537、540、541、543所详细描述的）。假定在保存和恢复历境状态时没有执行指令，对不超过包括写入恢复的指令周期的25%的周期时间，需要采用异步SRAM，以避免开销周期的插入。这种速度要求是当采用同步SRAM时实现相同的处理器循环速率所需要的速度的两倍。在历境切换周期的前一半（时间间隔532、533、538）中，历境切换活动是相同的。 在历境切换周期的主时钟下降缘Mf 519，CSW信号508得到确立（时间间隔522）且一个NCTX（2：0）信号组512被置于下一个历境号（时间间隔534）。地址和数据信息在把当前历境执行的最后一个指令的结果写入保存阵列时必须是稳定的。因此，只有从主时钟下降缘Mf 519至下一个正交（quadrature）时钟下降缘Qf 520 520的时期对于下一个历境的保存状态的读出是可获得的（时间间隔540）。 这种结果随后较好地是得到锁存并在从正交时钟下降缘Qf 520 至下一个主时钟上升缘Mr 517的时期中得到发生。随后，这些锁存的值被有利地传送到处理器的工作寄存器时间间隔543）。在正交时钟下降缘Qf 520，NCTX（2：0）信号组512的值切换回当前历境号（时间间隔535），使得包括该指令（周期500）的结果的当前历境状态能够被写入到保存阵列（时间间隔541）。在主时钟上升缘Mr 517，NCTX（2：0）信号组512 切换回下一个历境号（时间间隔530），且下一个历境的第一个指令的执行开始（时间间隔537）。

与同步SRAM实施不同，写入操作在主时钟上升缘Mr 517 开始。 异步SRAM的采用要求数据通路结果在较早的时候是稳定的，以使得从正交时钟下降缘Qf520 至主时钟上升缘Mr 517的间隔中至保存阵列的写入能够进行。而对于同步SRAM，数据通路结果直到刚好在主时钟上升缘Mr

517开始都是不需要的，这便利了较短的指令周期和较快的处理。

现在参见图10，其中显示了一个电路的一个实施例的示意图，该电路适合于实施各个激活事件的事件记录、事件掩码和事件确认，以及历境活动位—包括初始化请求和等候请求逻辑—的管理，其中在历境控制器中的事件记录、掩码和确认可得到更好的理解。

为包括一个ACT位和与事件相关的历境的WAIT功能逻辑， 提供了历境控制器事件逻辑的一个一般化的“片”的示意部分。在此图中，所有的逻辑信号都被认为是在“高”真（逻辑1）状态下被确立的。 这种示意部分显示了事件逻辑的一个实施例，但不应该被作为对本发明的做法的一种限定。

一个外界事件信号550可用两种极性中的任何一种来确立， 所以一个可编程倒相功能560在软件信号551的控制下可得到提供，以为内部使用建立一个高真信号。由于这种外界信号具有与内部时钟的未确定的相位关系，因而采用了一种同步器562 —它在输入信号使用之前把它与主时钟上升缘Mr 517同步。多个源可被用来设定一个事件触发器570， 包括同步的外部信号564的前缘、一个内部源566的前缘、或者指定这种历境和事件的软件SIGNAL功能552。这些事件源被一个或门568所结合—其输出使得一个事件触发器570能够被设定在主时钟上升缘Mr 517。

由于事件触发器D输入570被硬连接至真（如所示是置于逻辑一），在设定了事件触发器570之后事件信号的“非”操作不取消该事件。 如果处理器提供了诸如所示的实施例的SKPn的指令（如下面所述的），事件触发器输出570 可作为事件状态寄存器 94 中的一个位和作为一个事件条件信号组596中的一个可测试条件而被软件所读取。事件触发器570可被一个硬件预定555或通过一个或门574所施加的一个与门572 —其“与”输入包括历境正在运行的同时此事件的一个ACK（确认）功能554的执行—的输出所清除。

来自历境的事件掩码寄存器94的用于该历境事件的适当的位，事件掩码位558在“与”门580中与一个事件触发器输出570进行“与”操作， 并通过一个“或”门584被加到一个ACT触发器590的输入端。该

“与”门580的输出，当进行用于VECTOR功能的历境事件的优先级编码时，也得到采用，如在下面所详细描述的。来自“与”门580的一个掩码的事件信号在“或”门584 中与来自与该历境有关的所有其他事件的掩码的事件信号—包括通过一个“与”门582 而来自等候逻辑的输出门的一个信号—进行“或”操作。

“或”门584的逻辑真输出条件使能了活动触发器590，使得活动触发器590能够在正交时钟上升缘Qr 518被设定在一个“非”倒相器586的输出值。通过利用“与”门582的输出与同一经过“非”倒相器586的倒相，活动触发器590的D输入可得到使能。如果一或多个激活事件得到确立，活动触发器590被置于正交时钟上升缘Qr 518， 且在前一个指令周期期间没有WAIT功能被执行。活动触发器590也可被一个INIT功能588的执行直接设定到该历境，并被一个硬件复置信号555所直接清除。活动触发器输出590还被历境优先级逻辑所使用并被一个“非”倒相器592所倒相， 以明确一个WAIT触发器578。如果在前一个指令周期中一个WAIT功能得到执行， 不论是否确立了任何激活事件，ACT触发器590都通过“非”倒相器586 得到清除。

WAIT触发器578是需要的，因为一个历境可在执行一个WAIT 功能与执行一个跟随该WAIT功能的指令之间被抢先。（在图2中的53、54和58显示了发生这种情况的一个例子）。当在该历境正在运行的同时一个WAIT功能557被一个“与”门576所解码时（一个信号556），WAIT触发器578得到使能以设定在主时钟上升缘Mr 517。由于一个历境必须是活动的以执行一个WAIT功能，这种活动记录了WAIT功能的发生，因为活动触发器590 的处于真状态的输出对WAIT触发器578通过“非”倒相器592的清除输入进行了“非”操作。

在下一个正交时钟上升缘Qr 518 —其中该历境处于运行状态（信号556），活动触发器590由于“与”门输出582的确立而被清除。 如果该历境在WAIT触发器578被设定的同一指令周期边界（主时钟上升缘Mr 517 ）处被抢先或时间分割，该历境将不被运行。因此，历境运行信号556 将在下一个正交时钟上升缘Qr 518之前被“非”操作，且活动触发

器590 将保持被设定。当该历境恢复运行时，活动触发器590 将在在执行这一个指令之后造成历境该历境不活动的第一个指令周期的正交时钟上升缘Qr 518处得到清除。活动触发器590的输出的“非”输出经过“非”倒相器592而清除了WAIT触发器578。

现在参见图11，其中显示了与根据本发明的一个实施例的指令集中的历境控制和历境间通信有关的机器指令的字段和位分配。指令解码和字段编码的细节不与本发明直接有关，且包括该图主要是为了说明提供历境控制器所需的信息的操作数字段。

利用SKPx指令600，能够最有效地实现历境事件状态寄存器94 中的位的测试。这些事件在八个有关信号组成的指定的“条件组”（C组）604与包含在指令字中的一个八位掩码值605之间的位比较或掩码下，进行一个测试。如果测试操作603所指定的条件是真，跟随SKPx的指令被跳过。 与本发明相关的是C组01即一个“EVENT”组608， 它不受事件掩码的影响且它测试运行的历境的事件状态寄存器94的内容。

一个VECTOR指令610从与SKPx指令相同的操作码（opcode） 得到解码但在其“测试操作”字段612中具有不同的值。VECTOR指令字的其他10 个位是一个矢量基地址613，其使用将在下面描述。

一个SIGNAL 指令620 被用来实施前面描述的历境间软件信令功能。 SIGNAL指令620是基于具有不同次解码值623的的一个扩展操作码字段622 的值的处理器控制指令之一。当SIGNAL指令被执行时，在历境控制器内两个参数字段得到解码。一个指定的事件号624标明了与指定的历境号625相关的事件中所要确立的具体事件。所有的事件都可以是SIGNAL指令620的目标，但该历境控制器及相关事件源的具体例子中的实施细节使得难于允许SIGNAL指令620确立一定的条件。

一个ACK指令630和一个INIT指令640，以与SIGNAL指令620类似的方式得到格式化和解码，但每一个只具有一个参数字段。该ACK指令630只携带一个事件号624，因为历境的事件的确认只被在相同历境中执行的码所允许，所以历境号参数是多余的。INIT指令640只携带历境号625，因为初始化功能是针对历境的，而不是针对与一个历境相联系的一个事

件的。

一个STROBE指令650能够从多至32个分立、强制的控制功能653中产生一个指定的功能。一个WAIT指令654与历境控制器有关，它清除运行的历境的ACT位；一个SETFG指令655设定运行的历境的FG位；一个CLRFG指令656清除运行的历境的FG 位；且一个SLEEP指令657使得历境控制器暂停操作并使处理器能够进入一个极端低电力睡眠模式。

INIT指令640用于强迫目标历境进入一个已知的状态， 以进行初始化或错误恢复。INIT指令640的执行把该指令中指定的历境中的ACT和FG位都置于逻辑真。它还设定一个历境CY（进位(carry)）标记，以使历境能够区分硬件复置（当CY等于零）和INIT（当CY等于一）并强迫历境在历境相关的初始化矢量处开始执行。

现在参见图12，其中显示了根据本发明的一个实施例的用于产生在处理器上的控制存储器地址的位的源。用于上述历境相关初始矢量的初始化矢量地址，可通过把INIT指令640（图11）的一个历境号字段625的内容置于包含在图12所示的INIT指令666 的一个输入项中可见的所有零的一个地址字的第五至第三位存储位置中，而得到形成。

现在参见图13，其中显示了根据本发明的一个实施例的在控制存储器中的初始化矢量的示例性数据结构图。如所示，该实施例采用了位于相继的四字内部的一组八个初始化矢量670-677，控制存储器地址图案678开始于控制存储器地址0x0000。一个四字矢量节距得到选择，因为在此处理器上的一个长、绝对的分支要求三个字，且除了最后一个（历境7）矢量677之外的所有矢量都可能要求这种分支。历境7不要求分支是有用的， 因为历境7是在硬件复置之后活动的唯一历境。

因此，在历境7 初始化矢量处的编码被用来在硬件复置之后对其他的历境进行初始化并用于处理对历境7的INIT功能。 用于其他处理器上的矢量节距能够以取决于实施例的方式得到选择。还希望的是在某些处理器上，初始化矢量的内容是作为要通过该矢量进行间接分支的地址，而不是在该矢量地址开始程序执行。图11所示的VECTOR指令610 对于造成历境激活的事件的基于优先级的解码是有用的。

现在参见图14，其中显示了在根据本发明的一个实施例的处理器上，借助用于优先级设定和解码具体的历境激活位的矢量指令产生目标地址。如前所述，VECTOR指令610对于造成历境激活的事件的基于优先级的解码是有用的。当执行时，这种指令转移（分支）到位于控制存储器中的一个矢量表中的一组八个处理程序680-687中的一个。

在VECTOR指令字610的十个最低位中指定了一个矢量表基地址613。通过对用历境事件掩码寄存器90进行“与”操作的历境事件状态寄存器94进行优先级编码，选择出一个具体的矢量。 随后， 利用所产生的事件号694作为矢量地址678的位位置的第六至四位，与矢量地址678中为零的第三到第0位602一起（如图13所示），使执行在分配给最高优先级（最低编号的）的所确立的非掩码事件的八字处理程序位置680-687的开始处继续进行。由于在图11 中显示的VECTOR指令610通常在跟随WAIT指令654的重新激活之后不久得到采用，有理由预期至少一个非掩码的事件触发器将为真（等于一）。如果不是这种情况，历境将不活动。然而，对于其中无事件位得到设定的情况，可以在Base+64字688处包括一个矢量。

对于当前的实施例的指令集，八字的矢量节距允许很多处理程序在处理该事件的同时整个地适合于不要求分支的矢量表之内。对于提供一个这种矢量解码功能的实施例，该间距可得到适当选择以实现使整个处理程序组适合于矢量表与由于处理程序区远长于通常所要求的长度而使较大量的控制存储器未得到使用之间的平衡。

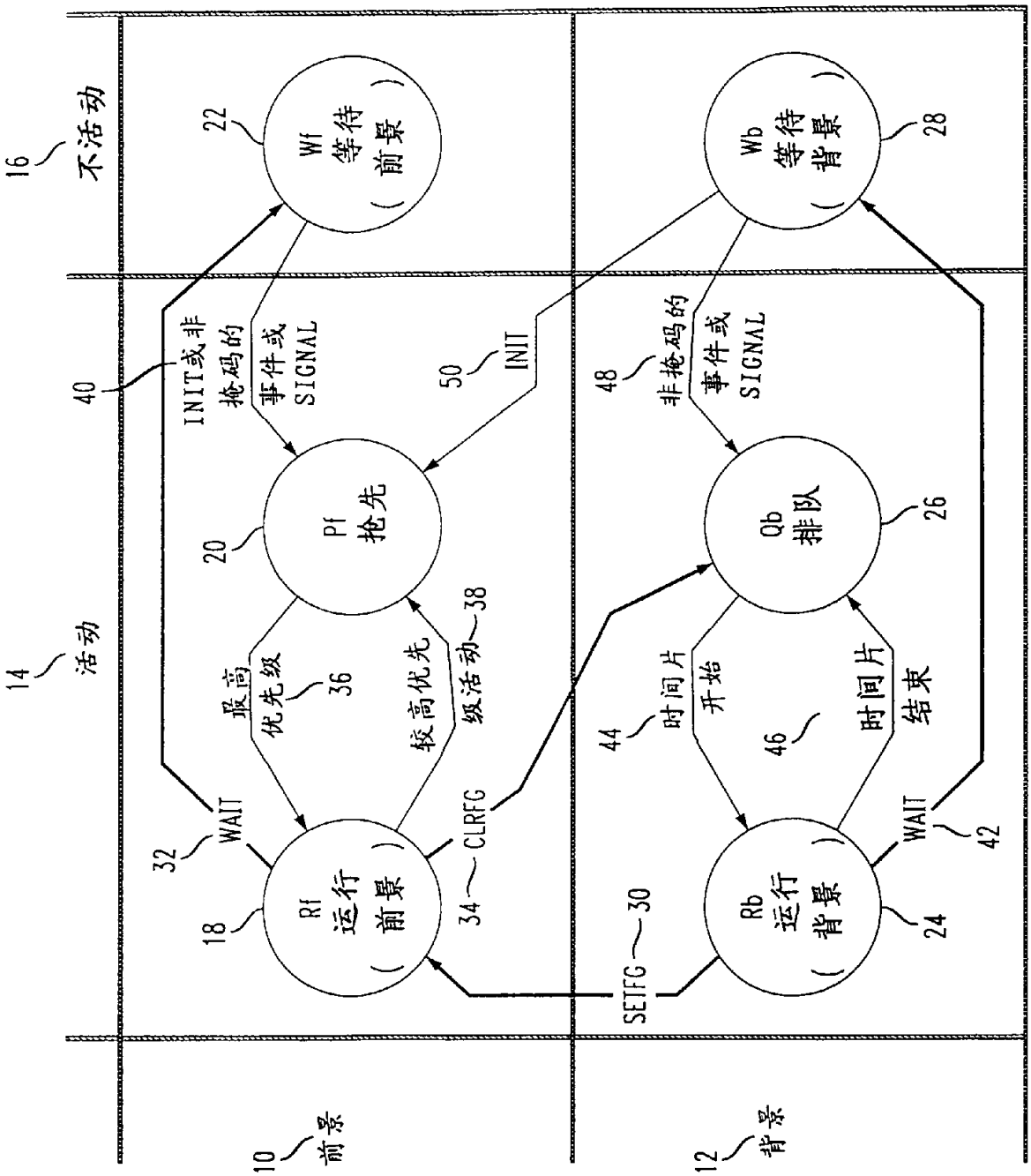
从上可见，显然的是，本发明提供了一种用于管理处理器中的多任务处理的历境控制器及其操作方法。在一个实施例中，该历境控制器包括：（1）一个事件记录器，它记录事件的发生；以及，（2）一个编码器， 它与事件记录器相联系，并响应于一个软件指令，对与至少某些事件相应的位进行优先级编码，以从中产生一个事件相关矢量，以使处理器能够作为其一个功能而发生分支。矢量化是取决于矢量解码软件指令的具体例子的，而不是取决于事件或历境的。

虽然已经详细描述了本发明，本领域的技术人员应该理解的是，在

不脱离本发明的精神和范围的前提下，它们能够进行各种改变、替换和代替。

说 明 书 附 图

图 1


10
前景
12
背景
14
活动
16
不活动
18
Rf
运行
(前景)
20
Pf
抢先
22
Wf
等待
(前景)
24
Rb
运行
(背景)
26
Qb
排队
28
Wb
等待
(背景)
30
SETFG
32
WAIT
34
CLRFG
36
最高优先级
38
较高优先级活动
40
INIT或非抢码的事件或SIGNAL
42
WAIT
44
时间片开始
46
时间片结束
48
非抢码的事件或SIGNAL
50
INIT

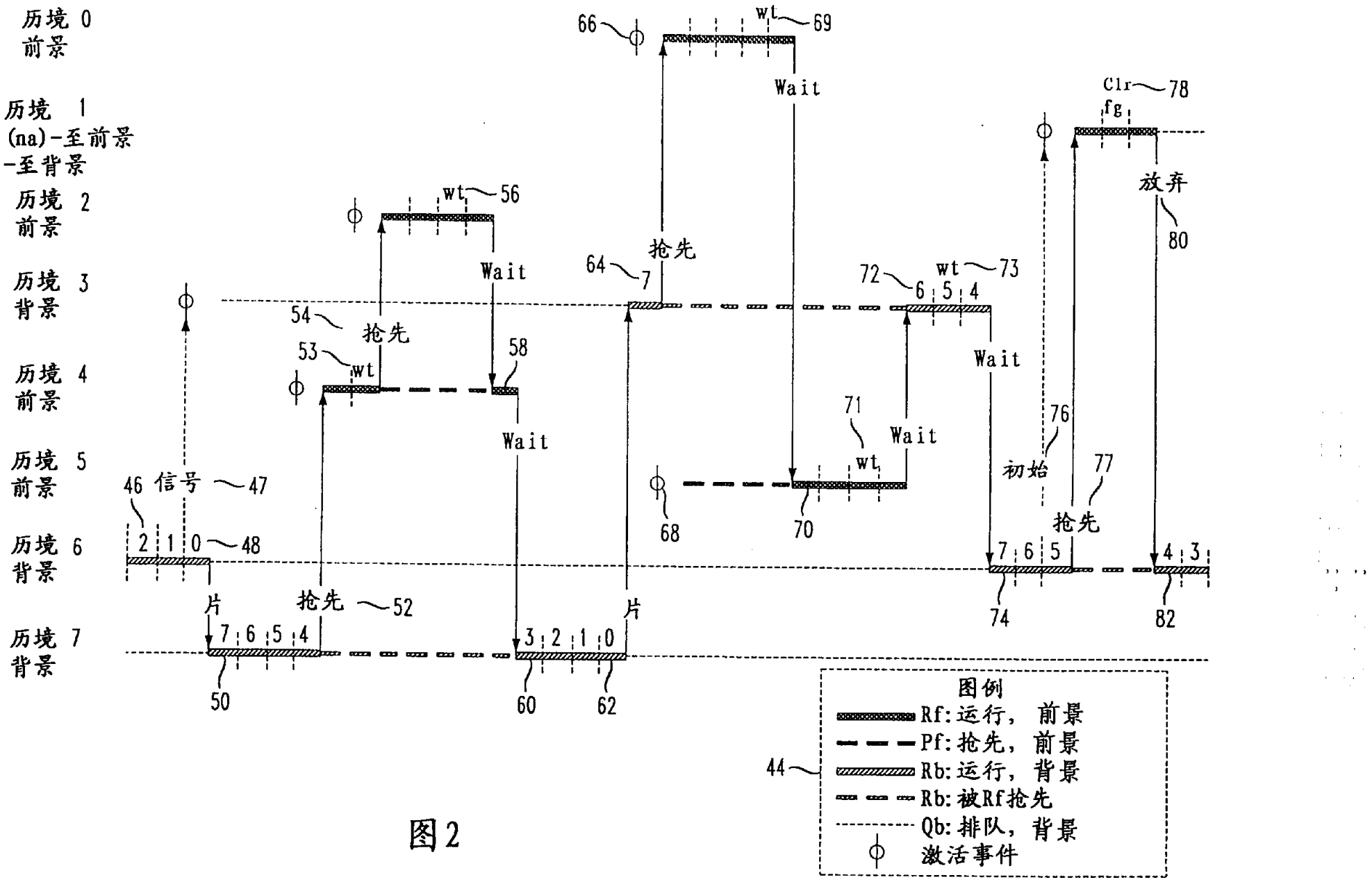

历境 0
前景
历境 1
(na)-至前景
-至背景
历境 2
前景
历境 3
背景
历境 4
前景
历境 5
前景
历境 6
背景
历境 7
背景
wt
Wait
抢先
片
信号
初始
放弃
Clr
fg
图例
Rf: 运行, 前景
Pf: 抢先, 前景
Rb: 运行, 背景
Rb: 被Rf抢先
Qb: 排队, 背景
激活事件

图2

图 3

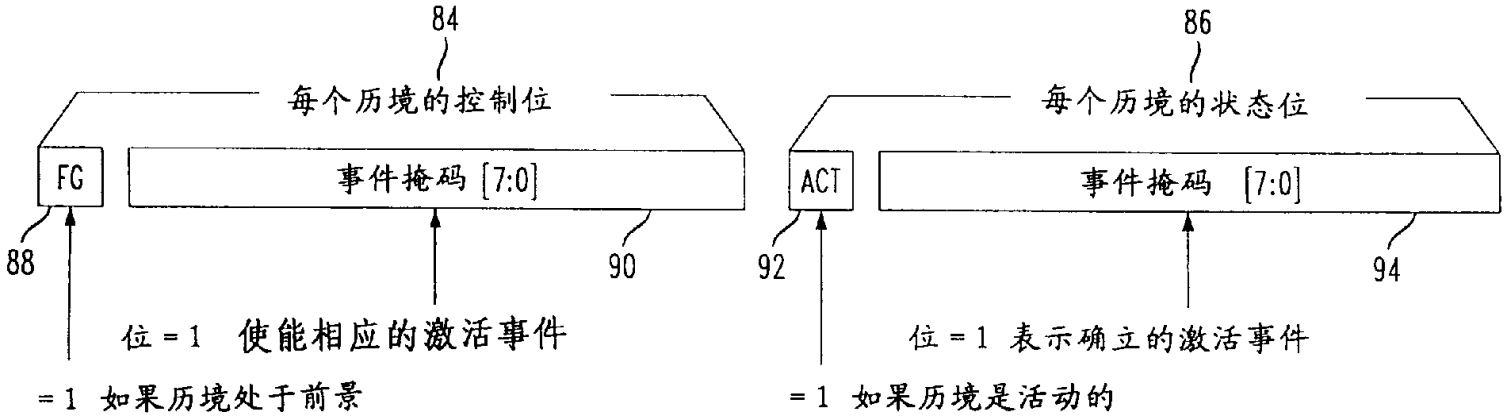
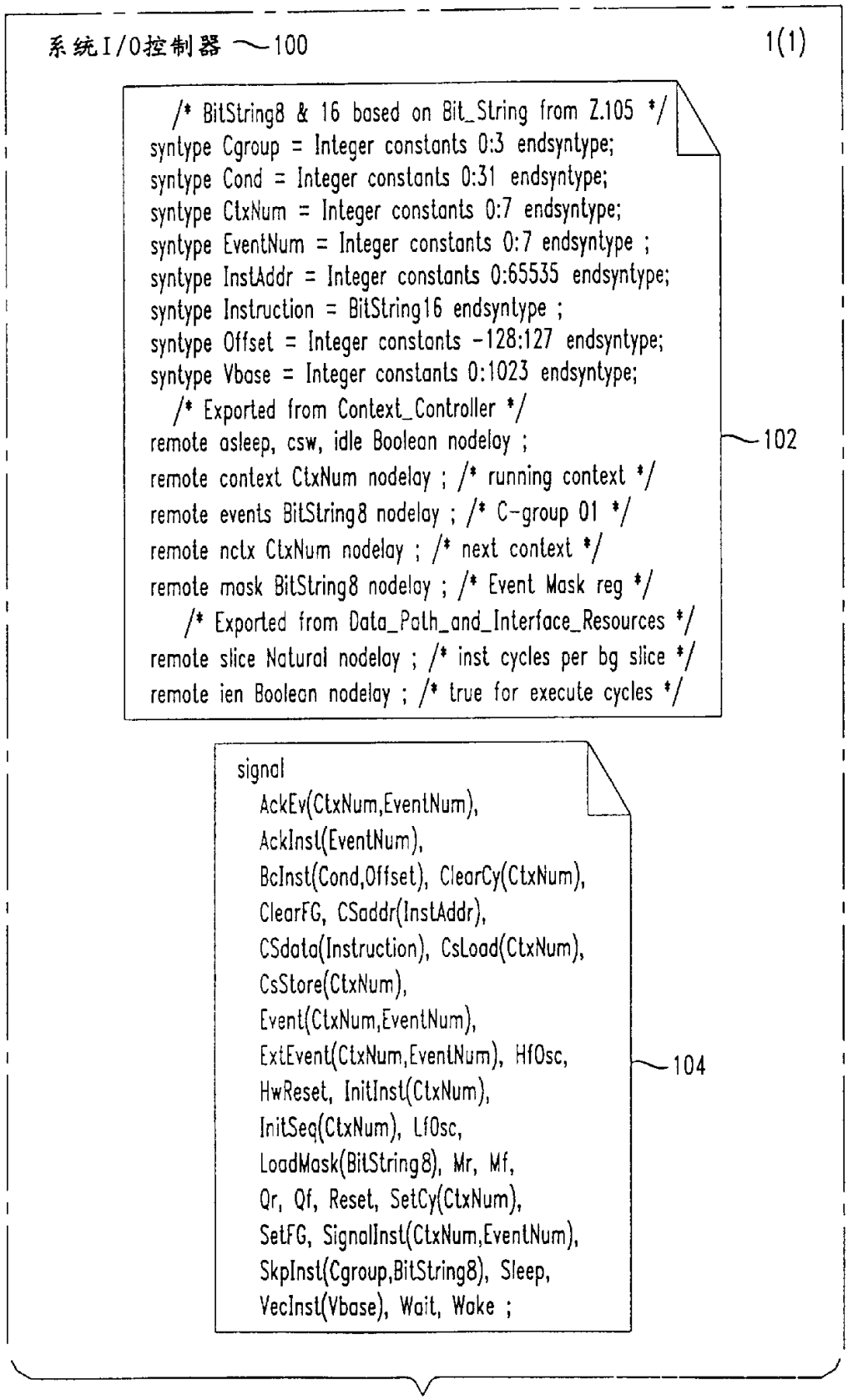

84
每个历境的控制位
FG
事件掩码 [7:0]
88
90
位 = 1 使能相应的激活事件
= 1 如果历境处于前景
86
每个历境的状态位
ACT
事件掩码 [7:0]
92
94
位 = 1 表示确立的激活事件
= 1 如果历境是活动的

图4A


系统I/O控制器 ~100
1(1)
/* BitString8 & 16 based on Bit_String from Z.105 */
syntype Cgroup = Integer constants 0:3 endsyntype;
syntype Cond = Integer constants 0:31 endsyntype;
syntype CtxNum = Integer constants 0:7 endsyntype;
syntype EventNum = Integer constants 0:7 endsyntype ;
syntype InstAddr = Integer constants 0:65535 endsyntype;
syntype Instruction = BitString16 endsyntype ;
syntype Offset = Integer constants -128:127 endsyntype;
syntype Vbase = Integer constants 0:1023 endsyntype;
/* Exported from Context_Controller */
remote asleep, csw, idle Boolean nodelay ;
remote context CtxNum nodelay ; /* running context */
remote events BitString8 nodelay ; /* C-group 01 */
remote nctx CtxNum nodelay ; /* next context */
remote mask BitString8 nodelay ; /* Event Mask reg */
/* Exported from Data_Path_and_Interface_Resources */
remote slice Natural nodelay ; /* inst cycles per bg slice */
remote ien Boolean nodelay ; /* true for execute cycles */
~102
signal
AckEv(CtxNum,EventNum),
AckInst(EventNum),
BcInst(Cond,Offset), ClearCy(CtxNum),
ClearFG, CSaddr(InstAddr),
CSdata(Instruction), CsLoad(CtxNum),
CsStore(CtxNum),
Event(CtxNum,EventNum),
ExtEvent(CtxNum,EventNum), HfOsc,
HwReset, InitInst(CtxNum),
InitSeq(CtxNum), LfOsc,
LoadMask(BitString8), Mr, Mf,
Qr, Qf, Reset, SetCy(CtxNum),
SetFG, SignalInst(CtxNum,EventNum),
SkpInst(Cgroup,BitString8), Sleep,
VecInst(Vbase), Wait, Wake ;
~104

至图4B

图4B

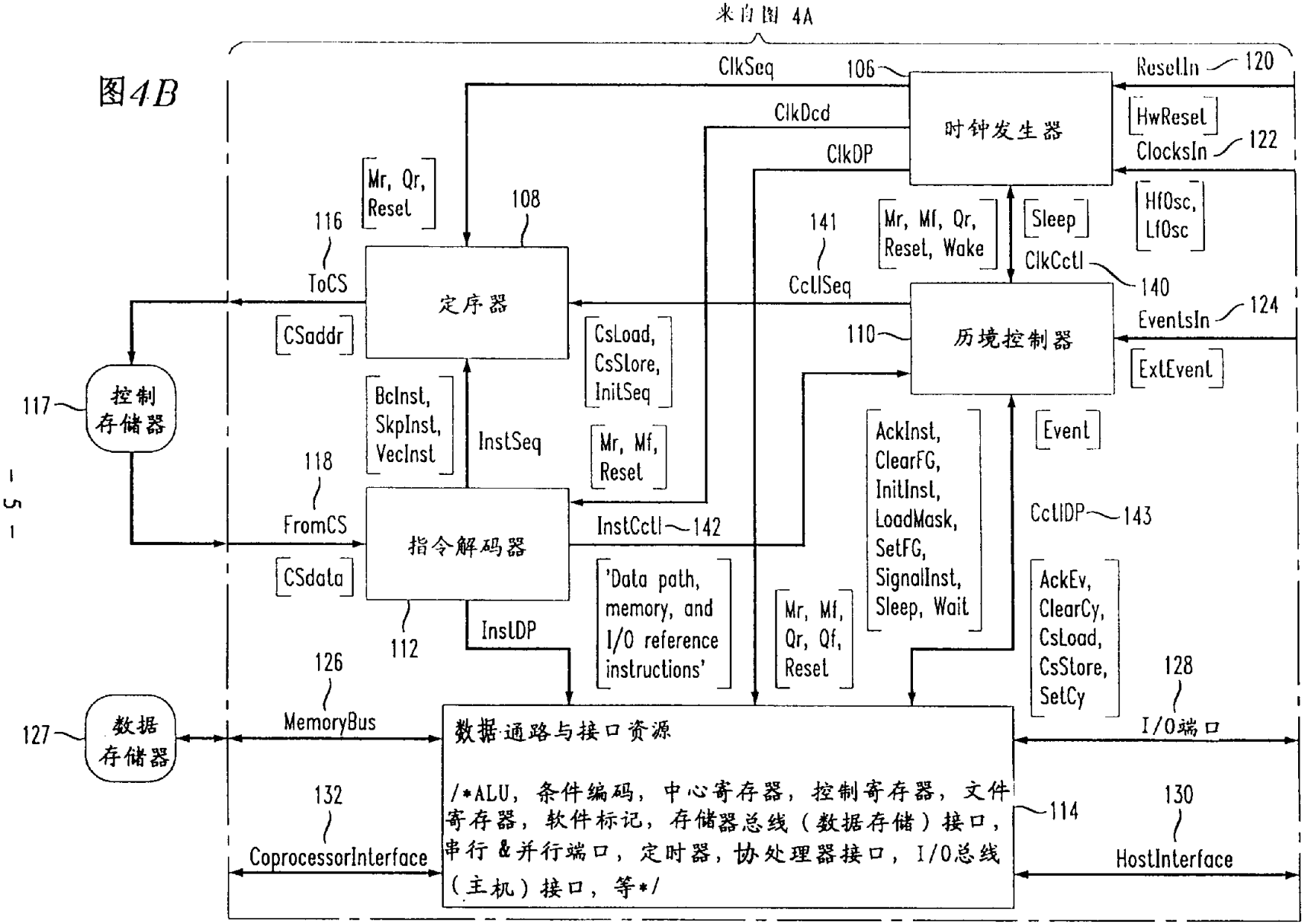

来自图 4A
ClkSeq
106
时钟发生器
ResetIn
120
HwReset
ClkDcd
ClkDP
ClocksIn
122
Mr, Qr, Reset
116
108
141
Mr, Mf, Qr, Reset, Wake
Sleep
HfOsc, LfOsc
ClkCtl
140
ToCS
定序器
CctlSeq
历境控制器
EventsIn
124
CSaddr
CsLoad, CsStore, InitSeq
110
ExtEvent
117
控制存储器
BcInst, SkpInst, VecInst
InstSeq
Mr, Mf, Reset
AckInst, ClearFG, InitInst, LoadMask, SetFG, SignalInst, Sleep, Wait
Event
118
FromCS
指令解码器
InstCctl
142
CctlDP
143
CSdata
'Data path, memory, and I/O reference instructions'
Mr, Mf, Qr, Qf, Reset
AckEv, ClearCy, CsLoad, CsStore, SetCy
112
InstDP
126
128
127
数据存储器
MemoryBus
数据通路与接口资源
I/O端口
/*ALU，条件编码，中心寄存器，控制寄存器，文件寄存器，软件标记，存储器总线（数据存储）接口，串行&并行端口，定时器，协处理器接口，I/O总线（主机）接口，等*/
114
132
130
CoprocessorInterface
HostInterface

图 5

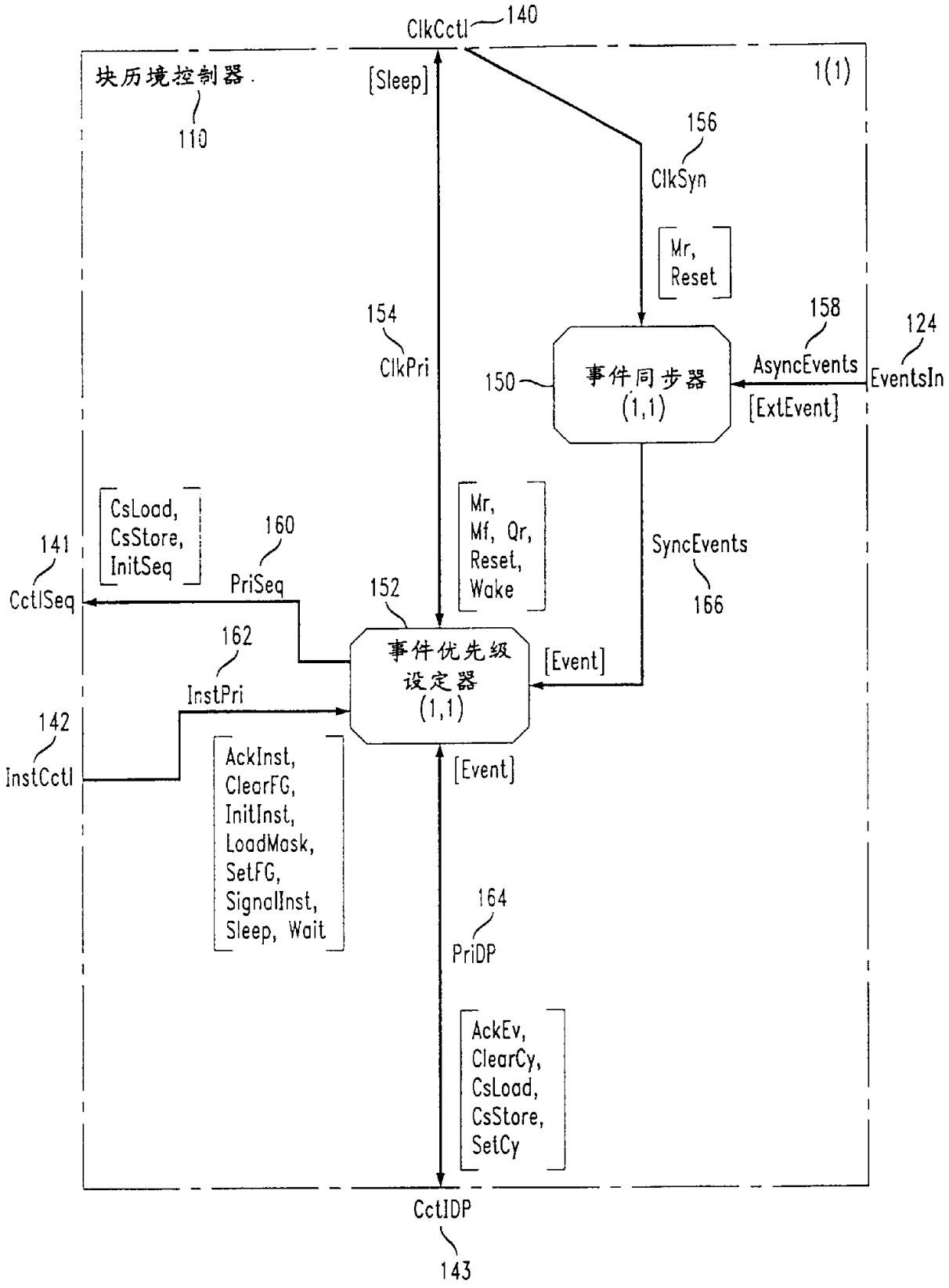

ClkCctl
140
块历境控制器
110
[Sleep]
1(1)
156
ClkSyn
Mr,
Reset
154
ClkPri
158
124
AsyncEvents
EventsIn
150
事件同步器
(1,1)
[ExtEvent]
CsLoad,
CsStore,
InitSeq
160
141
CctlSeq
PriSeq
Mr,
Mf, Qr,
Reset,
Wake
SyncEvents
166
152
162
InstPri
事件优先级
设定器
(1,1)
[Event]
142
InstCctl
AckInst,
ClearFG,
InitInst,
LoadMask,
SetFG,
SignalInst,
Sleep, Wait
[Event]
164
PriDP
AckEv,
ClearCy,
CsLoad,
CsStore,
SetCy
CctlDP
143

图 6

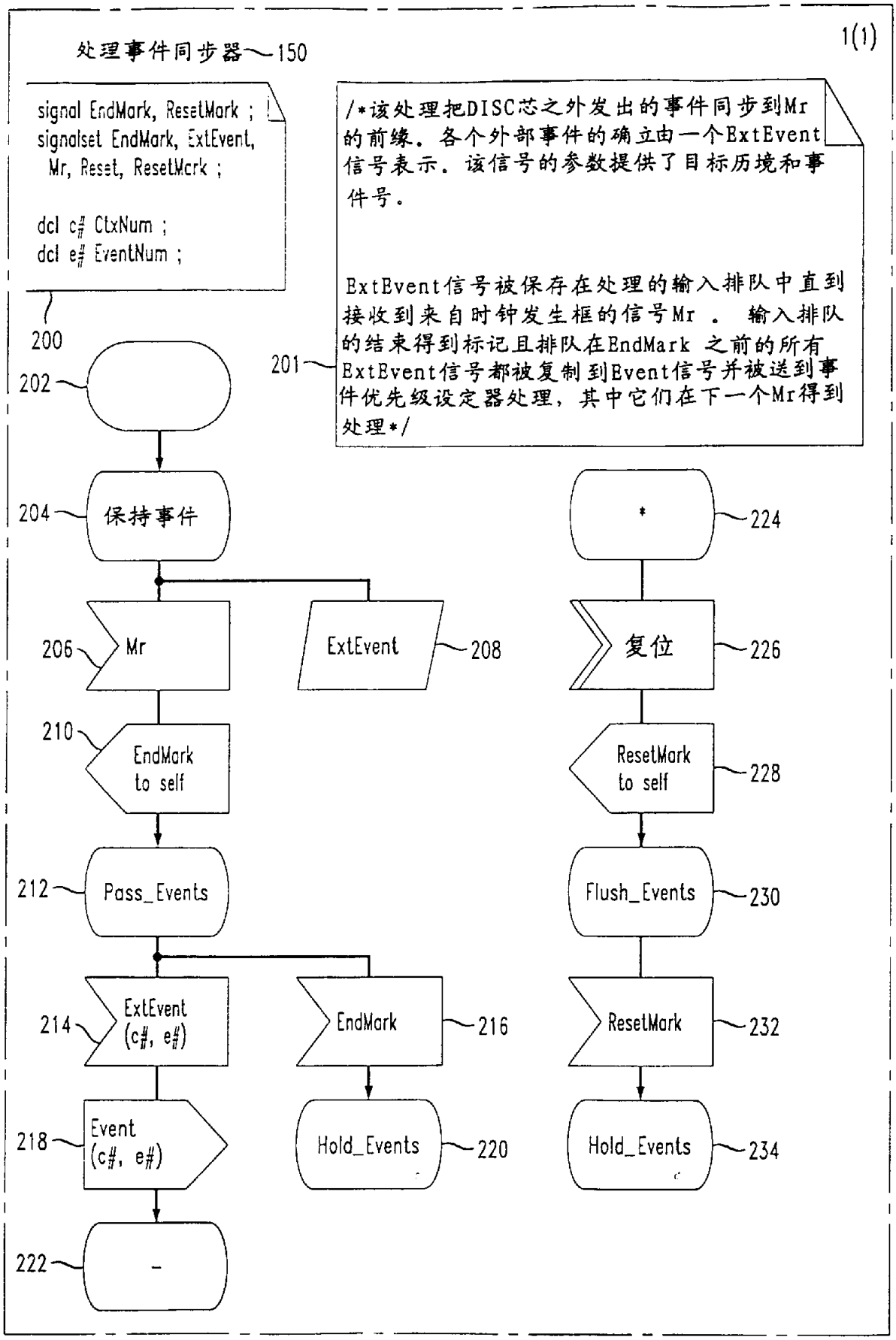
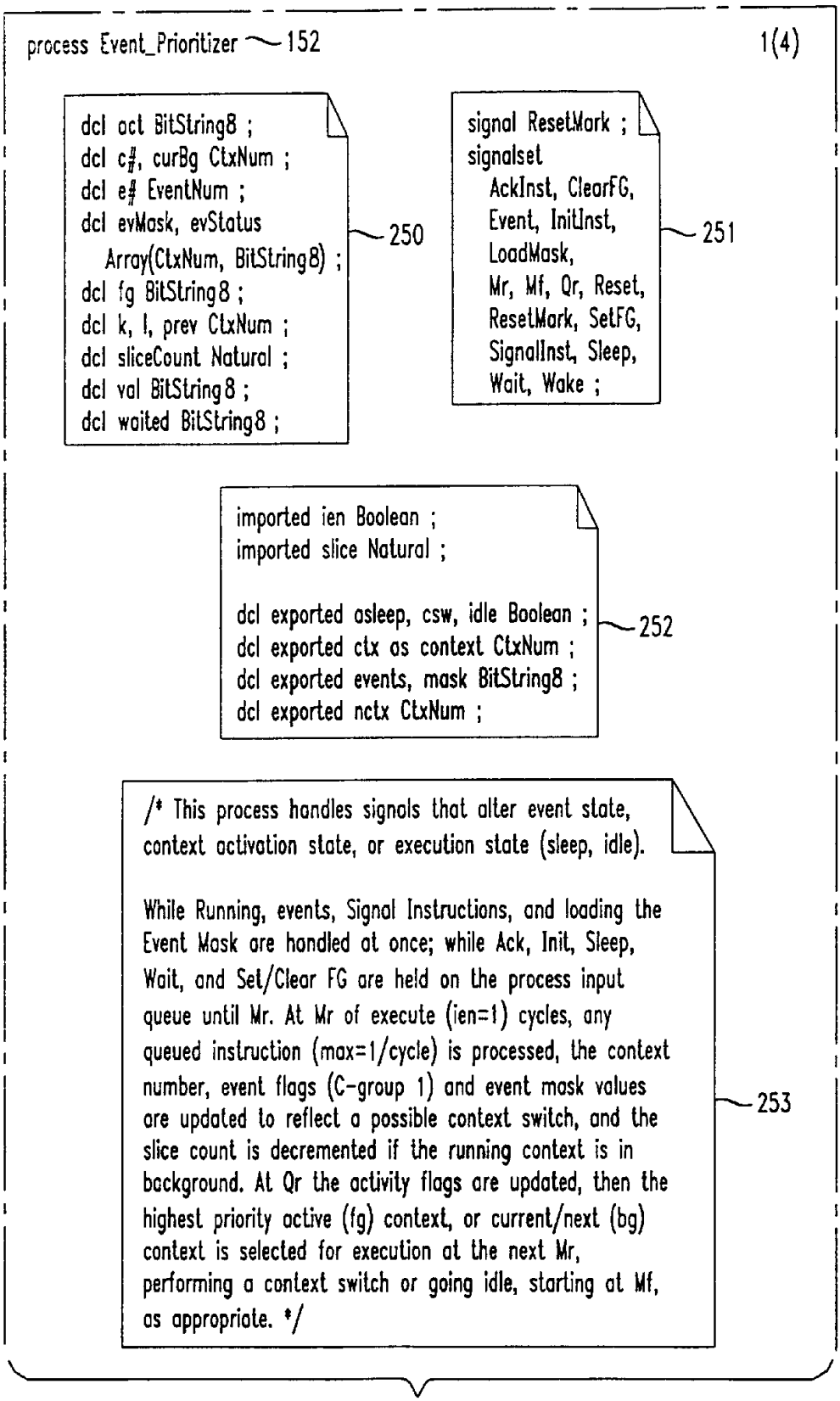
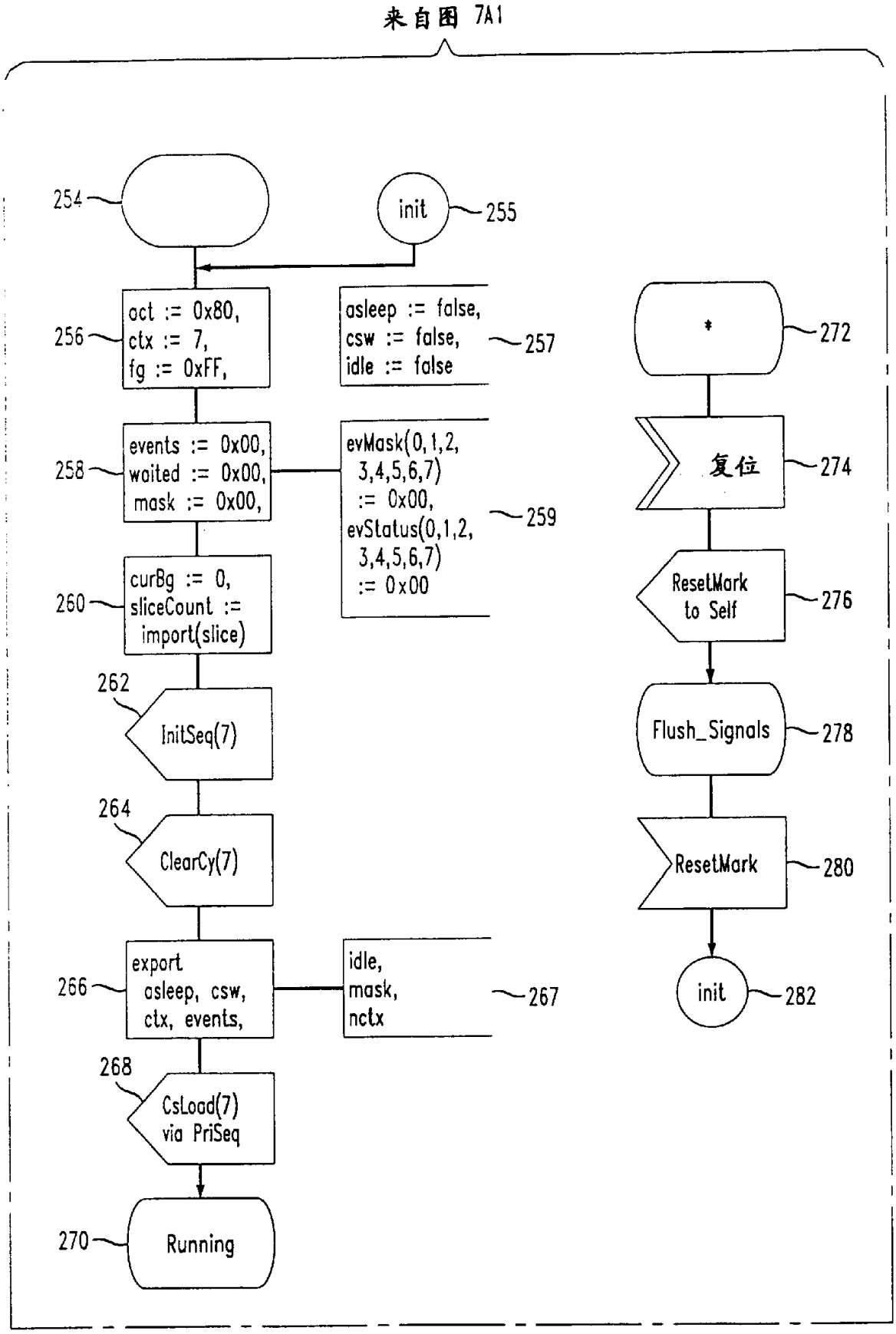

1(1)
处理事件同步器 150
signal EndMark, ResetMark ;
signalset EndMark, ExtEvent,
Mr, Reset, ResetMark ;
dcl c# CtxNum ;
dcl e# EventNum ;
/*该处理把DISC芯之外发出的事件同步到Mr的前缘。各个外部事件的确立由一个ExtEvent信号表示。该信号的参数提供了目标历境和事件号。
ExtEvent信号被保存在处理的输入排队中直到接收到来自时钟发生框的信号Mr 。 输入排队的结束得到标记且排队在EndMark 之前的所有ExtEvent信号都被复制到Event信号并被送到事件优先级设定器处理，其中它们在下一个Mr得到处理*/
200
201
202
204
保持事件
206
Mr
208
ExtEvent
210
EndMark
to self
212
Pass_Events
214
ExtEvent
(c#, e#)
216
EndMark
218
Event
(c#, e#)
220
Hold_Events
222
224
*
226
复位
228
ResetMark
to self
230
Flush_Events
232
ResetMark
234
Hold_Events

图 7A1


process Event_Prioritizer
152
1(4)
dcl oct BitString8 ;
dcl c#, curBg CtxNum ;
dcl e# EventNum ;
dcl evMask, evStatus
Array(CtxNum, BitString8) ;
dcl fg BitString8 ;
dcl k, l, prev CtxNum ;
dcl sliceCount Natural ;
dcl val BitString8 ;
dcl waited BitString8 ;
250
signal ResetMark ;
signalset
AckInst, ClearFG,
Event, InitInst,
LoadMask,
Mr, Mf, Qr, Reset,
ResetMark, SetFG,
SignalInst, Sleep,
Wait, Wake ;
251
imported ien Boolean ;
imported slice Natural ;
dcl exported asleep, csw, idle Boolean ;
dcl exported ctx as context CtxNum ;
dcl exported events, mask BitString8 ;
dcl exported nctx CtxNum ;
252
/* This process handles signals that alter event state,
context activation state, or execution state (sleep, idle).
While Running, events, Signal Instructions, and loading the
Event Mask are handled at once; while Ack, Init, Sleep,
Wait, and Set/Clear FG are held on the process input
queue until Mr. At Mr of execute (ien=1) cycles, any
queued instruction (max=1/cycle) is processed, the context
number, event flags (C-group 1) and event mask values
are updated to reflect a possible context switch, and the
slice count is decremented if the running context is in
background. At Qr the activity flags are updated, then the
highest priority active (fg) context, or current/next (bg)
context is selected for execution at the next Mr,
performing a context switch or going idle, starting at Mf,
as appropriate. */
253

至图 7A2

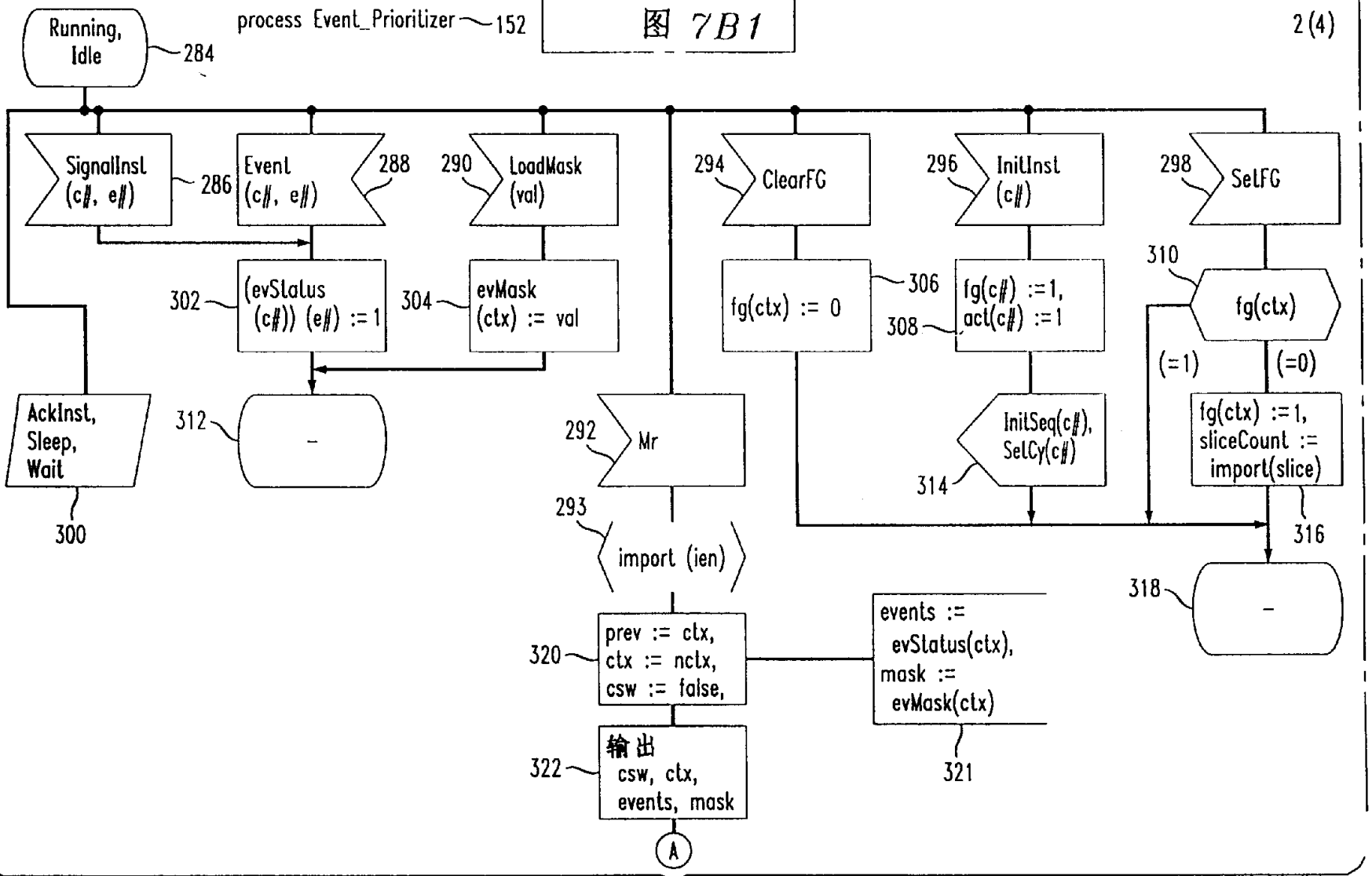
图 7A2


来自图 7A1
254
init
255
oct := 0x80,
ctx := 7,
fg := 0xFF,
256
asleep := false,
csw := false,
idle := false
257
events := 0x00,
waited := 0x00,
mask := 0x00,
258
evMask(0,1,2,
3,4,5,6,7)
:= 0x00,
evStatus(0,1,2,
3,4,5,6,7)
:= 0x00
259
curBg := 0,
sliceCount :=
import(slice)
260
262
InitSeq(7)
264
ClearCy(7)
export
asleep, csw,
ctx, events,
266
idle,
mask,
nctx
267
268
CsLoad(7)
via PriSeq
270
Running
*
272
复位
274
ResetMark
to Self
276
Flush_Signals
278
ResetMark
280
init
282

process Event_Prioritizer 152

图 7B1

2(4)



至图 7B2

图 7B2

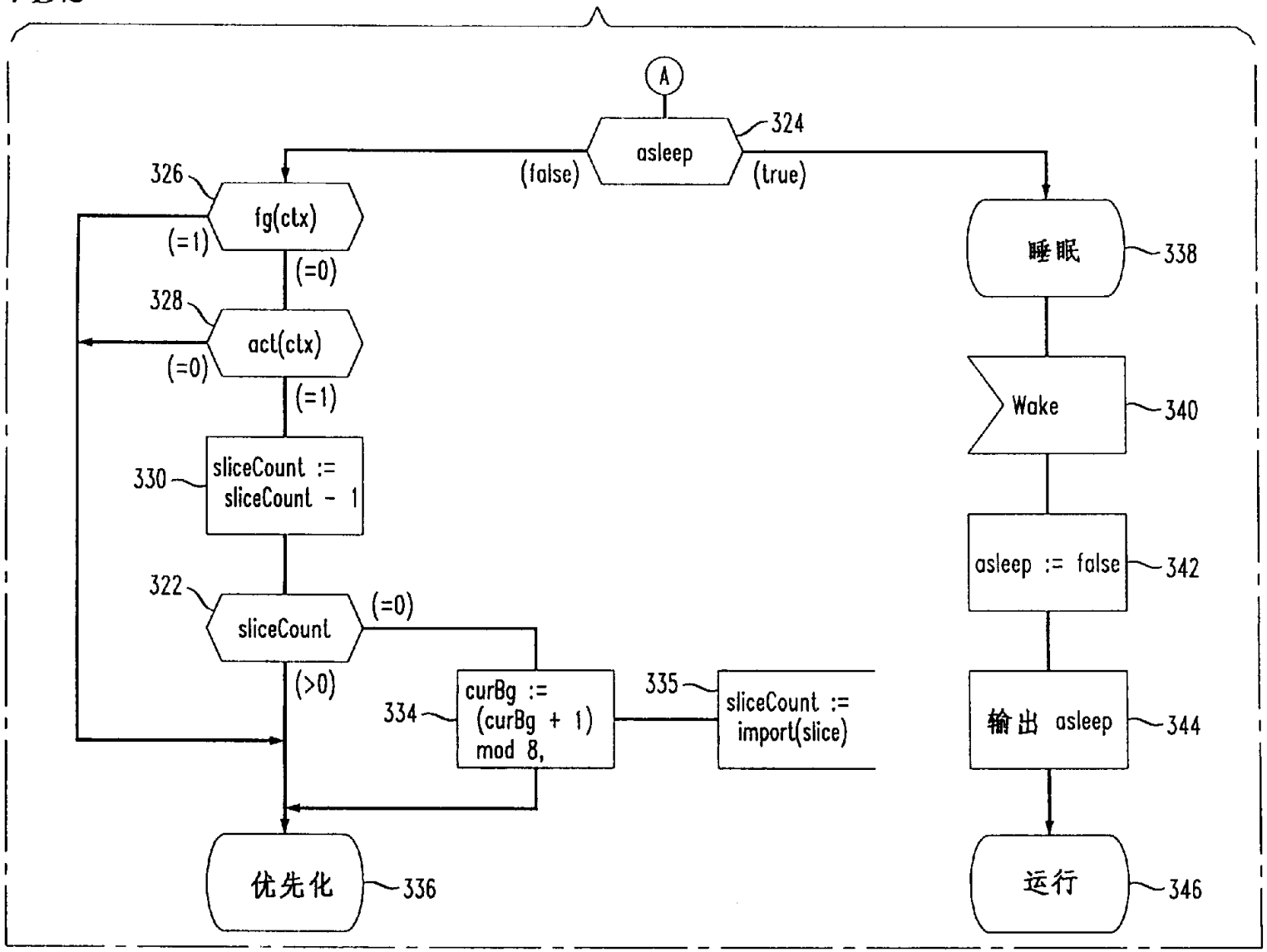

来自图 7B1
A
asleep
324
(false)
(true)
326
fg(ctx)
(=1)
(=0)
328
act(ctx)
(=0)
(=1)
330
sliceCount :=
sliceCount - 1
322
sliceCount
(=0)
(>0)
334
curBg :=
(curBg + 1)
mod 8,
335
sliceCount :=
import(slice)
优先化
336
睡眠
338
Wake
340
asleep := false
342
输出 asleep
344
运行
346

图 7C

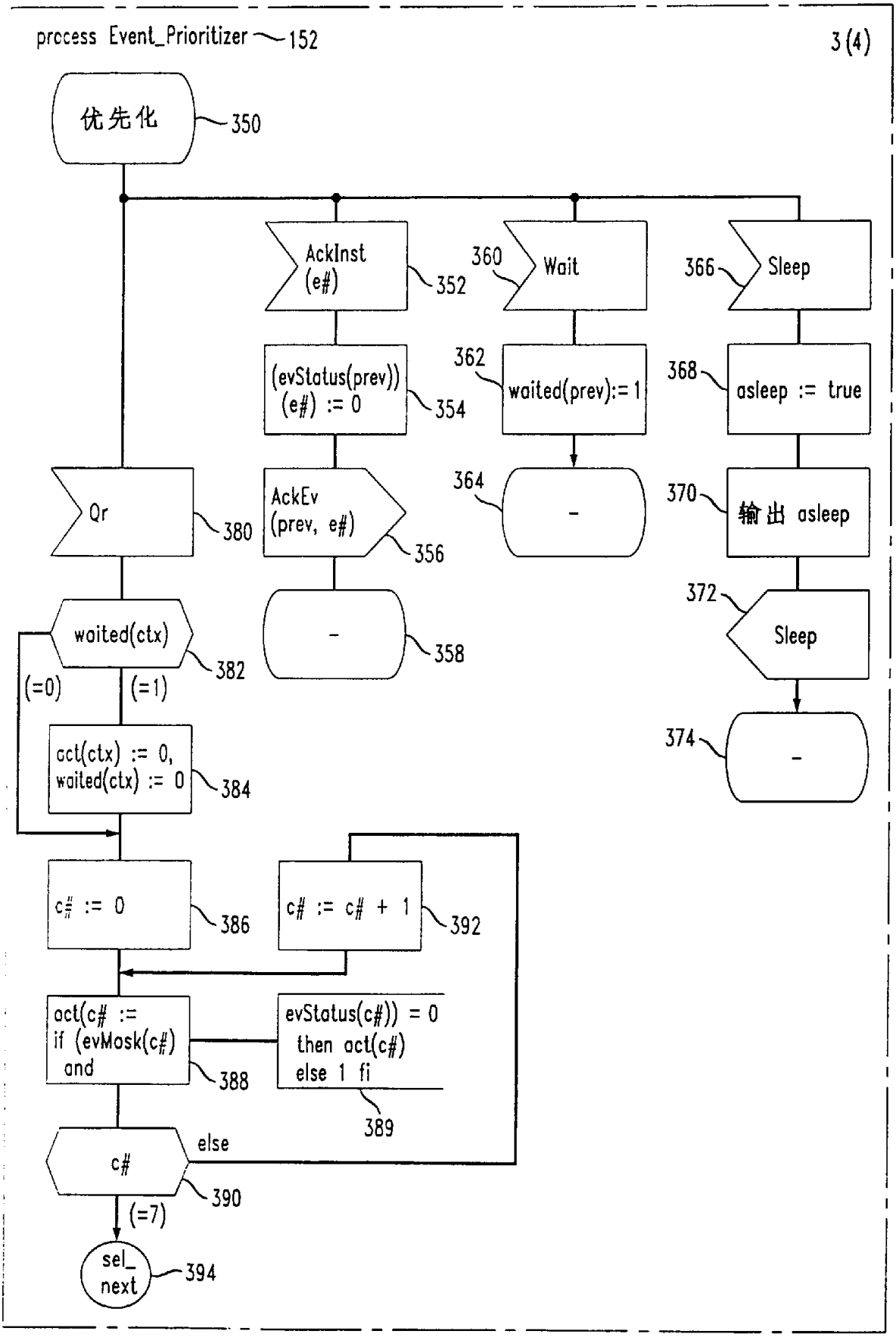
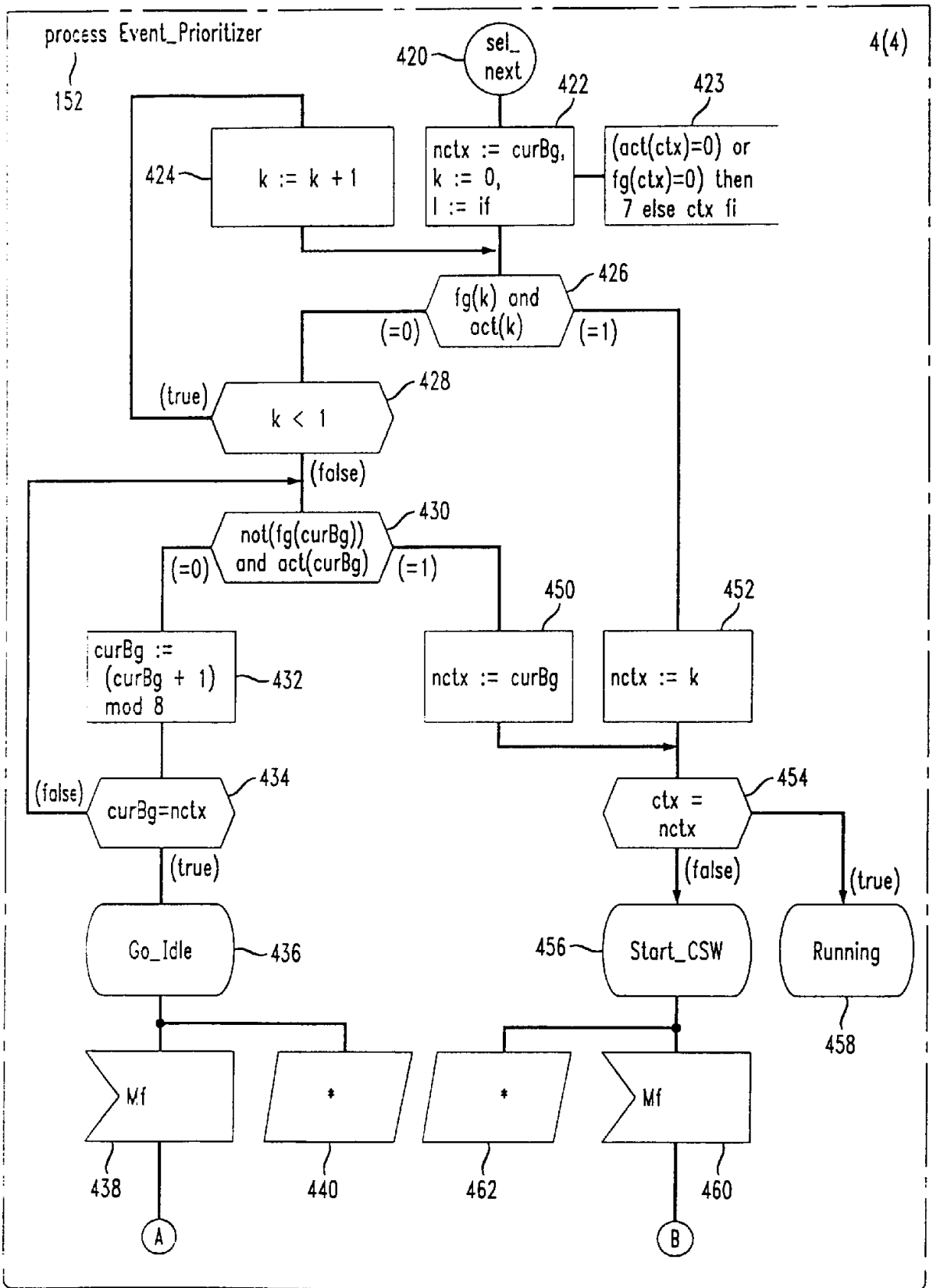

process Event_Prioritizer 152
3(4)
优先化
350
AckInst (e#)
352
(evStatus(prev)) (e#) := 0
354
AckEv (prev, e#)
356
-
358
360
Wait
362
waited(prev):= 1
364
-
366
Sleep
368
asleep := true
370
输出 asleep
372
Sleep
374
-
Qr
380
waited(ctx)
382
(=0)
(=1)
act(ctx) := 0, waited(ctx) := 0
384
c# := 0
386
c# := c# + 1
392
act(c# := if (evMask(c#) and
388
evStatus(c#)) = 0 then act(c#) else 1 fi
389
c#
390
else
(=7)
sel_ next
394

图 7D1


process Event_Prioritizer
152
4(4)
420
sel_
next
422
423
nctx := curBg,
k := 0,
l := if
(act(ctx)=0) or
fg(ctx)=0) then
7 else ctx fi
424
k := k + 1
426
fg(k) and
act(k)
(=0)
(=1)
(true)
428
k < 1
(false)
430
not(fg(curBg))
and act(curBg)
(=0)
(=1)
450
452
curBg :=
(curBg + 1)
mod 8
432
nctx := curBg
nctx := k
(false)
434
curBg=nctx
(true)
454
ctx =
nctx
(false)
(true)
Go_Idle
436
456
Start_CSW
Running
458
Mf
*
*
Mf
438
440
462
460
A
B

至图 7D2

图 7D2

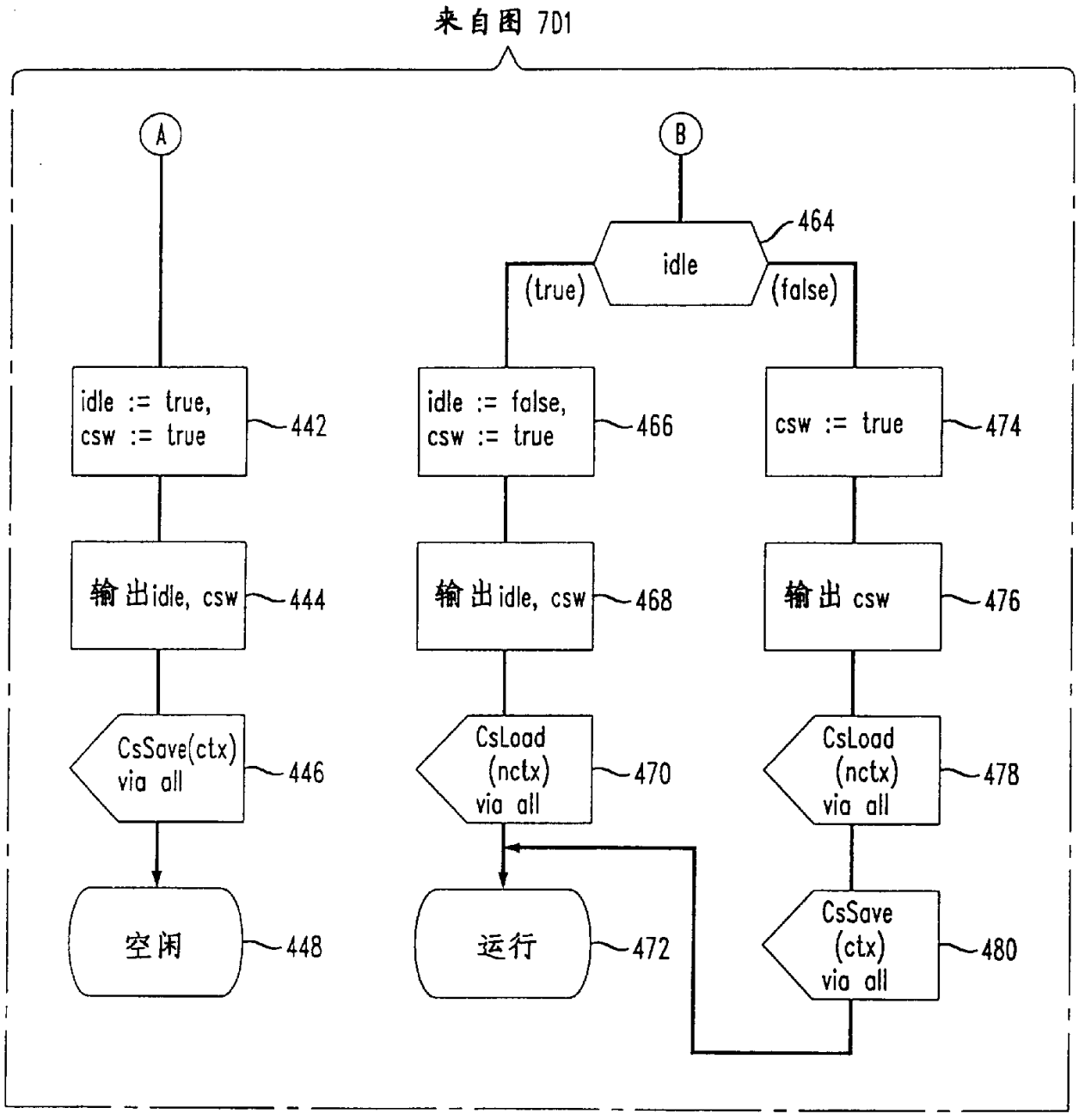

来自图 7D1
A
B
idle
464
(true)
(false)
idle := true,
csw := true
442
idle := false,
csw := true
466
csw := true
474
输出idle, csw
444
输出idle, csw
468
输出 csw
476
CsSave(ctx)
via all
446
CsLoad
(nctx)
via all
470
CsLoad
(nctx)
via all
478
空闲
448
运行
472
CsSave
(ctx)
via all
480

图 8

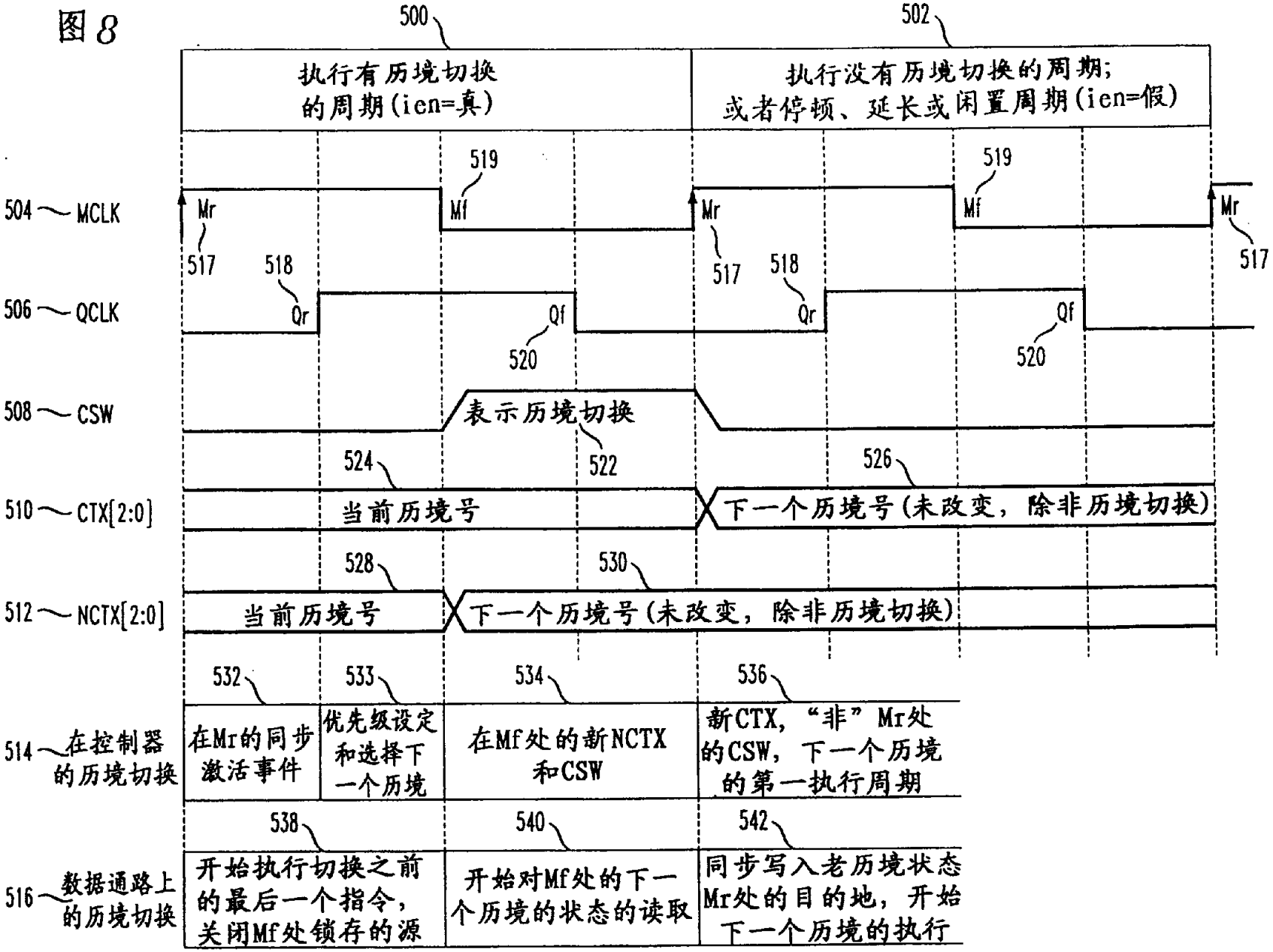

500
执行有历境切换的周期(ien=真)
502
执行没有历境切换的周期;或者停顿、延长或闲置周期(ien=假)
504 MCLK
506 QCLK
508 CSW
510 CTX[2:0]
512 NCTX[2:0]
514 在控制器的历境切换
516 数据通路上的历境切换
Mr
Mf
Qr
Qf
517
518
519
520
表示历境切换
522
524
526
528
530
当前历境号
下一个历境号(未改变，除非历境切换)
532
533
534
536
在Mr的同步激活事件
优先级设定和选择下一个历境
在Mf处的新NCTX和CSW
新CTX，“非”Mr处的CSW，下一个历境的第一执行周期
538
540
542
开始执行切换之前的最后一个指令，关闭Mf处锁存的源
开始对Mf处的下一个历境的状态的读取
同步写入老历境状态Mr处的目的地，开始下一个历境的执行

图 9

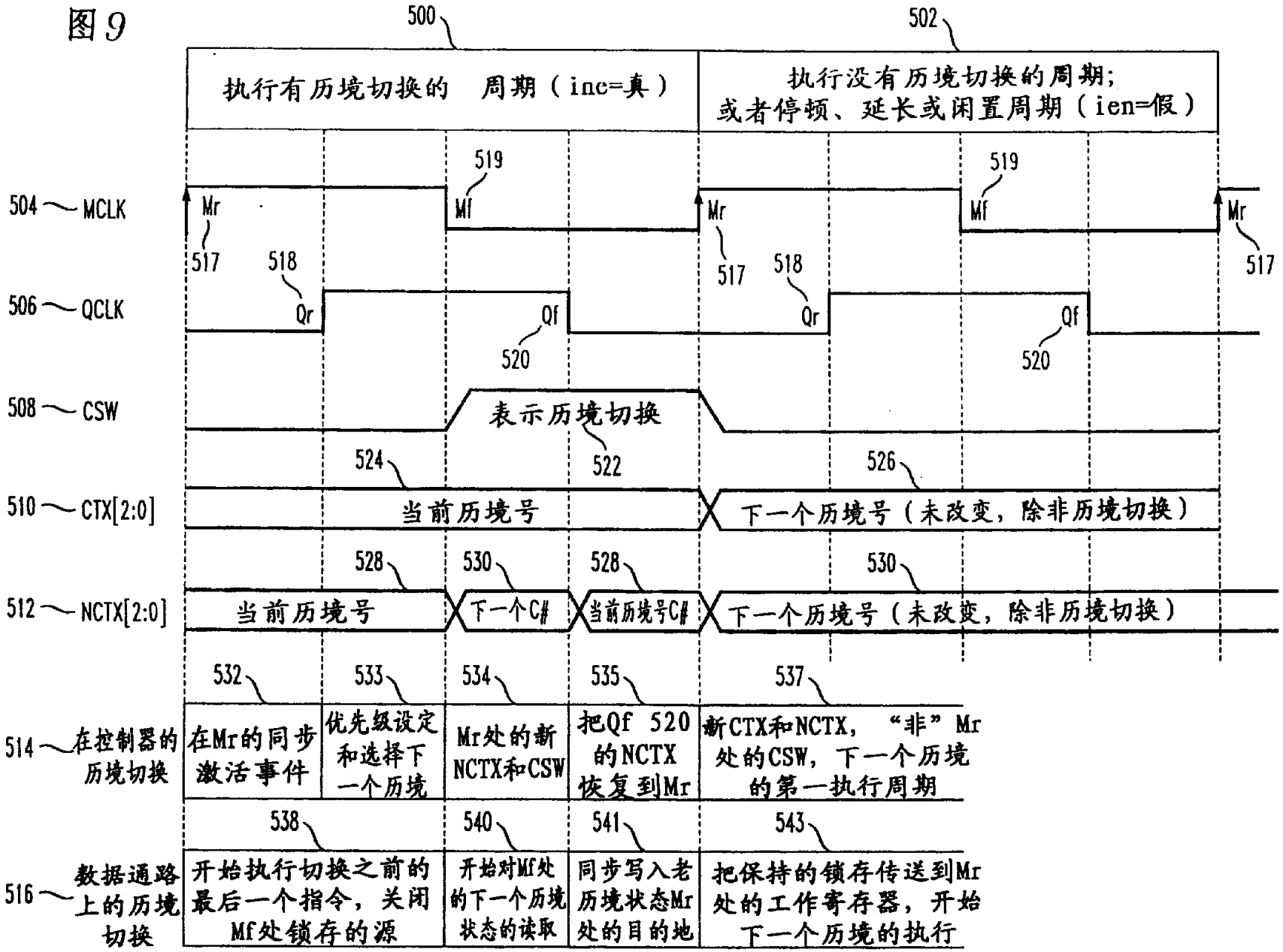

500
502
执行有历境切换的 周期（inc=真）
执行没有历境切换的周期；
或者停顿、延长或闲置周期（ien=假）
504 MCLK
506 QCLK
508 CSW
510 CTX[2:0]
512 NCTX[2:0]
514 在控制器的历境切换
516 数据通路上的历境切换
Mr
Mf
Qr
Qf
517
518
519
520
表示历境切换
522
524
526
当前历境号
下一个历境号（未改变，除非历境切换）
528
530
下一个C#
当前历境号C#
532
533
534
535
537
在Mr的同步激活事件
优先级设定和选择下一个历境
Mr处的新NCTX和CSW
把Qf 520的NCTX恢复到Mr
新CTX和NCTX，“非”Mr处的CSW，下一个历境的第一执行周期
538
540
541
543
开始执行切换之前的最后一个指令，关闭Mf处锁存的源
开始对Mf处的下一个历境状态的读取
同步写入老历境状态Mr处的目的地
把保持的锁存传送到Mr处的工作寄存器，开始下一个历境的执行

图10

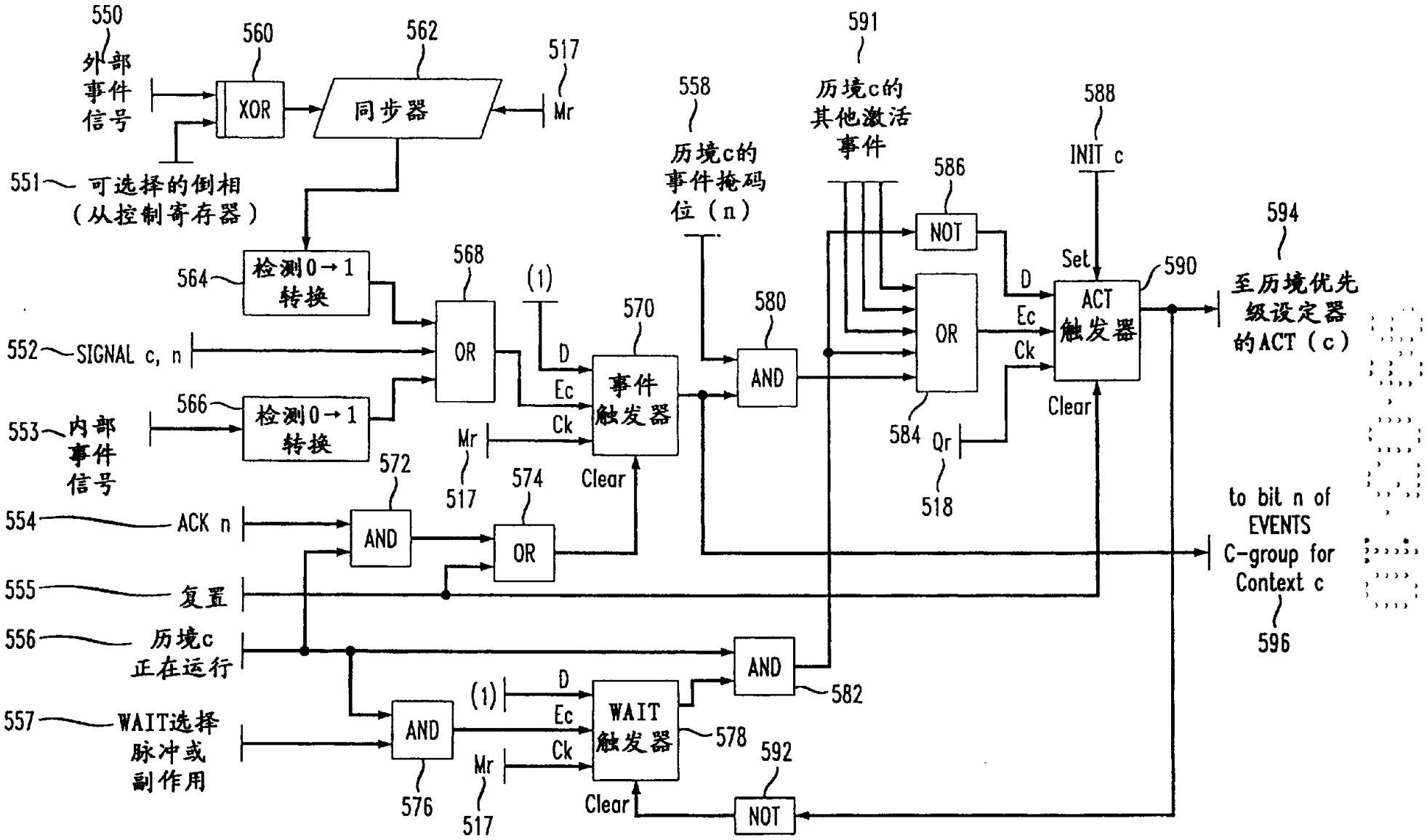

550
外部事件信号
560
XOR
562
同步器
517
Mr
551
可选择的倒相（从控制寄存器）
564
检测0→1转换
568
OR
552
SIGNAL c, n
566
检测0→1转换
553
内部事件信号
(1)
D
Ec
Ck
570
事件触发器
Mr
517
Clear
572
AND
574
OR
554
ACK n
555
复置
556
历境c正在运行
557
WAIT选择脉冲或副作用
AND
576
(1)
D
Ec
Ck
WAIT触发器
578
Mr
517
Clear
AND
582
NOT
592
558
历境c的事件掩码位（n）
580
AND
591
历境c的其他激活事件
586
NOT
584
Qr
518
OR
D
Ec
Ck
588
INIT c
Set
ACT触发器
590
Clear
594
至历境优先级设定器的ACT（c）
to bit n of EVENTS C-group for Context c
596

图11

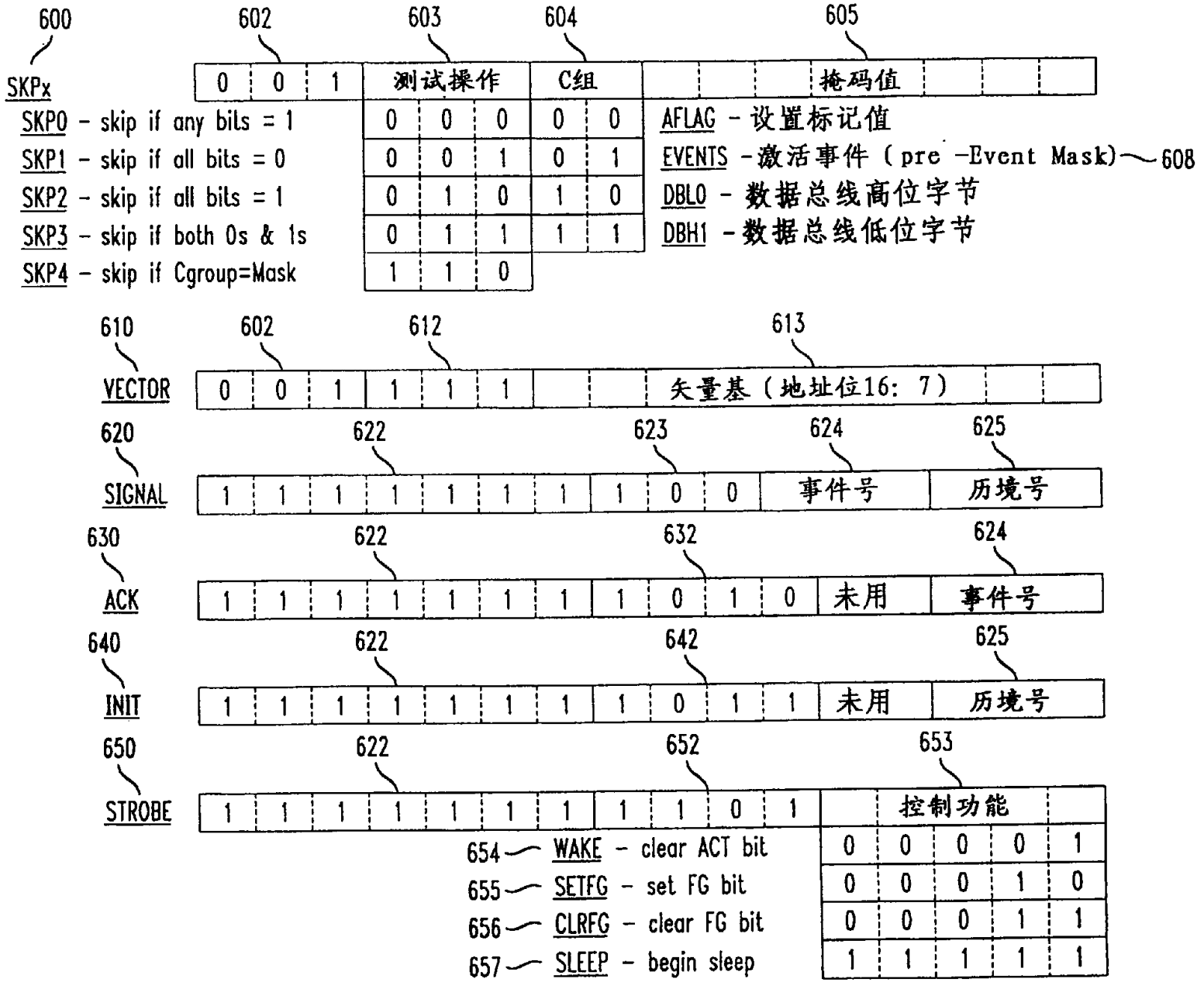

600
602
603
604
605
SKPx
0 0 1
测试操作
C组
掩码值
SKP0 - skip if any bits = 1
SKP1 - skip if all bits = 0
SKP2 - skip if all bits = 1
SKP3 - skip if both 0s & 1s
SKP4 - skip if Cgroup=Mask
0 0 0 0 0
0 0 1 0 1
0 1 0 1 0
0 1 1 1 1
1 1 0
AFLAG - 设置标记值
EVENTS - 激活事件 (pre -Event Mask)
608
DBL0 - 数据总线高位字节
DBH1 - 数据总线低位字节
610
602
612
613
VECTOR
0 0 1 1 1 1
矢量基（地址位16: 7）
620
622
623
624
625
SIGNAL
1 1 1 1 1 1 1 1 0 0
事件号
历境号
630
622
632
624
ACK
1 1 1 1 1 1 1 1 0 1 0
未用
事件号
640
622
642
625
INIT
1 1 1 1 1 1 1 1 0 1 1
未用
历境号
650
622
652
653
STROBE
1 1 1 1 1 1 1 1 1 1 0 1
控制功能
654 WAKE - clear ACT bit
655 SETFG - set FG bit
656 CLRFG - clear FG bit
657 SLEEP - begin sleep
0 0 0 0 1
0 0 0 1 0
0 0 0 1 1
1 1 1 1 1

图12

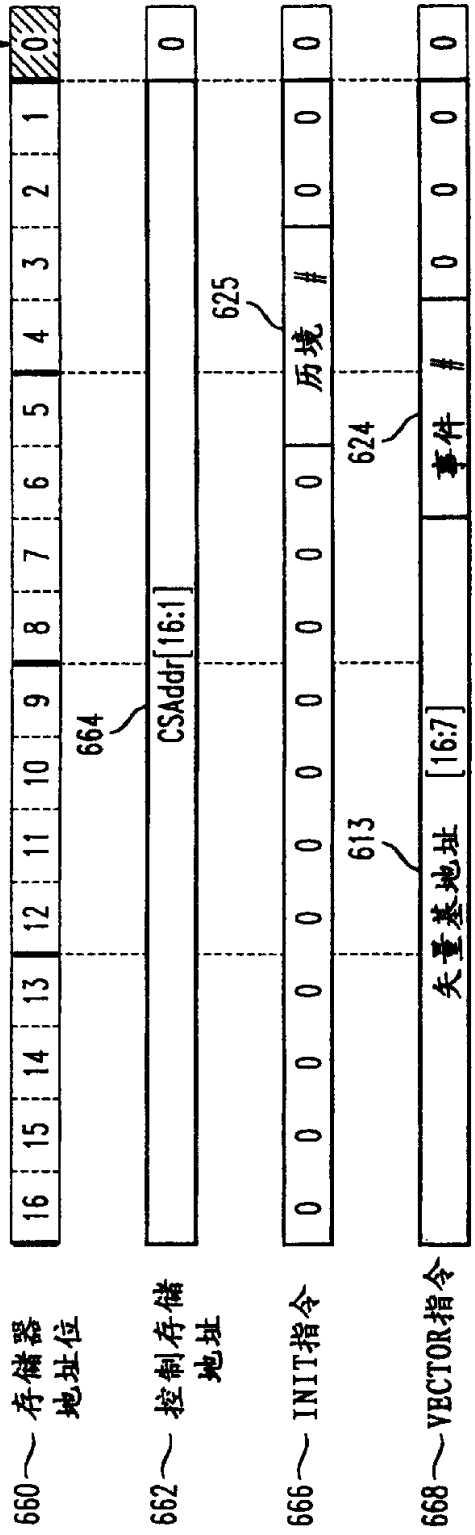

661～ 使用偶（字）地址存取的控制存储器
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
664
CSAddr[16:1]
625
历境 #
624
事件 #
613
矢量基地址 [16:7]
660～ 存储器地址位
662～ 控制存储地址
666～ INIT指令
668～ VECTOR指令

图13

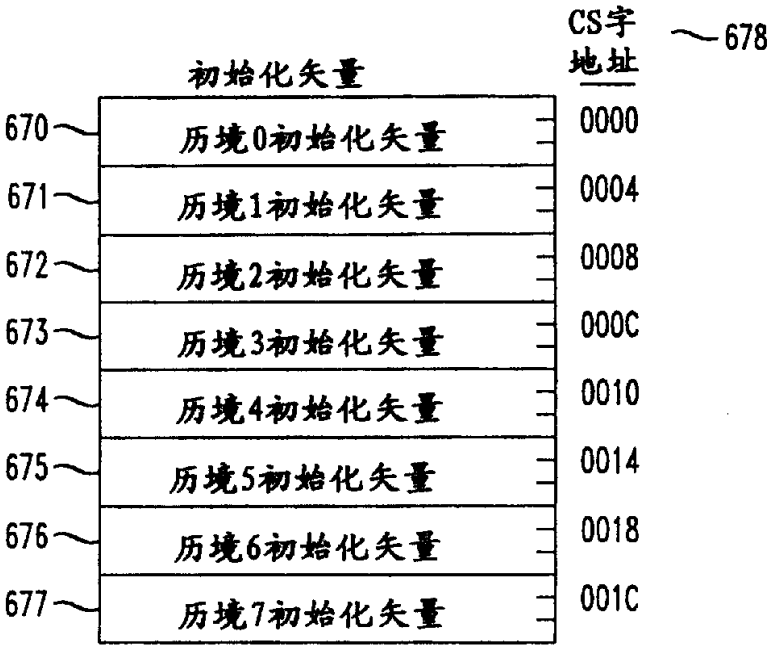
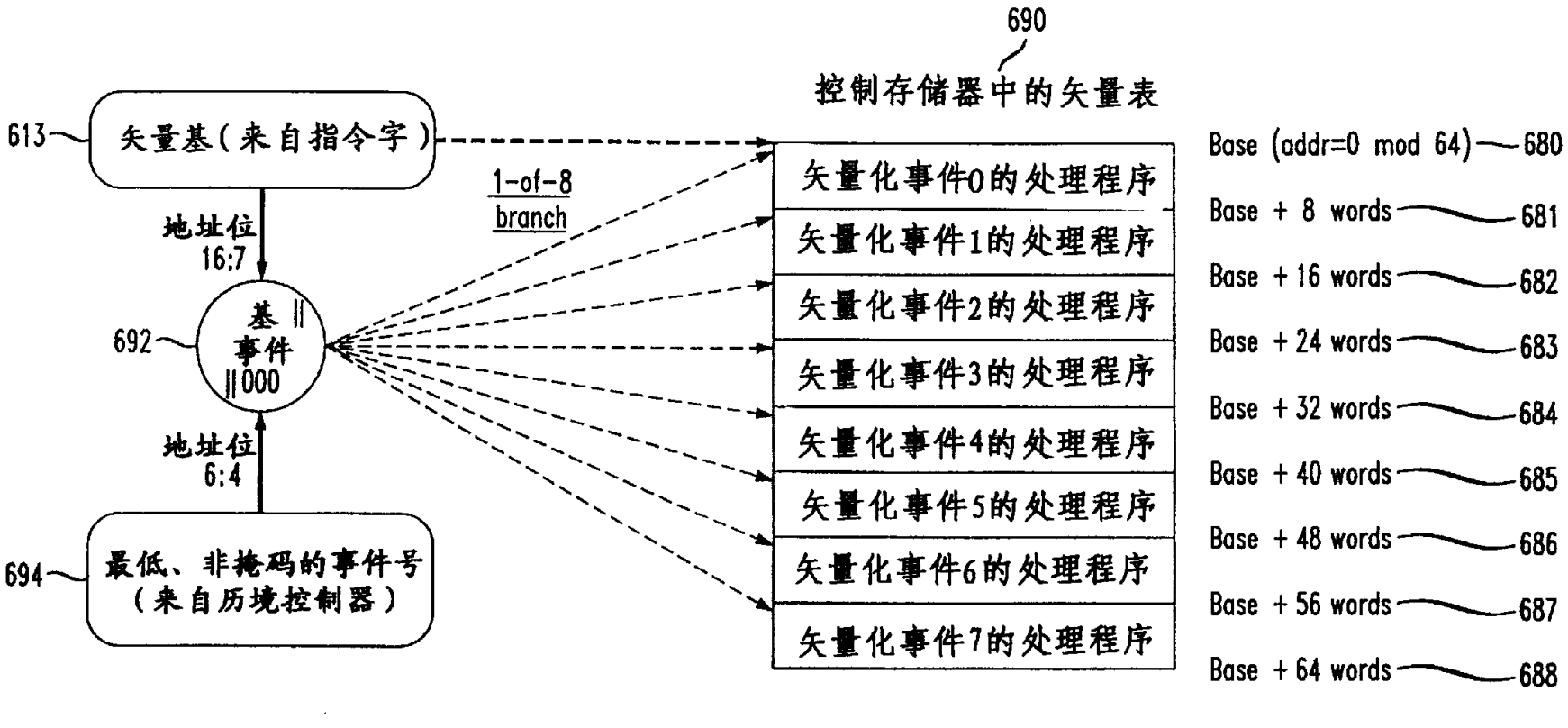

CS字地址
678
初始化矢量
670
历境0初始化矢量
0000
671
历境1初始化矢量
0004
672
历境2初始化矢量
0008
673
历境3初始化矢量
000C
674
历境4初始化矢量
0010
675
历境5初始化矢量
0014
676
历境6初始化矢量
0018
677
历境7初始化矢量
001C

图14


690
控制存储器中的矢量表
613
矢量基(来自指令字)
1-of-8
branch
地址位
16:7
基 ||
事件
||000
692
地址位
6:4
694
最低、非掩码的事件号
(来自历境控制器)
矢量化事件0的处理程序
矢量化事件1的处理程序
矢量化事件2的处理程序
矢量化事件3的处理程序
矢量化事件4的处理程序
矢量化事件5的处理程序
矢量化事件6的处理程序
矢量化事件7的处理程序
Base (addr=0 mod 64) 680
Base + 8 words 681
Base + 16 words 682
Base + 24 words 683
Base + 32 words 684
Base + 40 words 685
Base + 48 words 686
Base + 56 words 687
Base + 64 words 688