[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 15/78
G06F 12/08

[12]发明专利申请公开说明书

[21]申请号 99805452.6

[43]公开日 2001年6月6日

[11]公开号 CN 1298520A

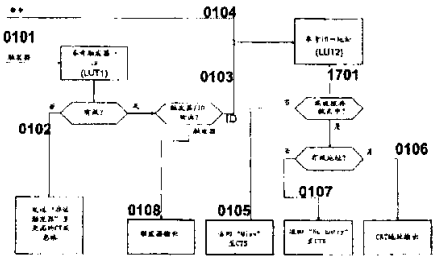
[22]申请日 1999.2.25 [21]申请号 99805452.6
[30]优先权
[32]1998.2.25 [33]DE [31]19807872.2
[86]国际申请 PCT/DE99/00504 1999.2.25
[87]国际公布 WO99/44147 德 1999.9.2
[85]进入国家阶段日期 2000.10.25
[71]申请人 PACT信息技术有限公司
地址 德国慕尼黑
[72]发明人 M·福尔巴赫
R·芒克
[74]专利代理机构 上海专利商标事务所
代理人 李家麟

权利要求书2页 说明书40页 附图页数32页

[54]发明名称 按照层结构具有二维或多维可编程序的单元结构(FPGAs、DPGAs等)的数据流处理器和模块的高速缓存配置数据方法

[57]摘要

至今,中央和全局单元已经组成一个对所有配置请求进行处理的模块。本 发明提供了多个能够执行该任务的工作单元。这些单元排列成一种结构。如果 该请求是无法处理的,则来自最低层的请求仅传送到下一个最高层。最高层与 内部的或外部的更高级别的配置存储器相连,该配置存储器含有所有该程序运 行所需的配置数据。配置单元的树形结构使得能够对配置数据进行高速缓存。配置主要是在本地进行访问的。在最坏的一种情况下,如果层结构中任何一个 CT(配置表)中都没有相关的数据,那么就必须从更高阶的配置存储器中载入 一种配置。



ISSN1008-4274

00·10·30

权　利　要　求　书

1．一种高速缓冲存储器存储指令的方法，这些指令是存在于由许多计算器组成的微处理机和具有二维或多维单元结构(比如 FPGA、DPGA、DFP 等)的功能模块之中的，其特征在于，

1.1 许多元件和可配置元件(CEL)结合成一组，而每个分组都和一个高速缓冲存储器单元(CT)相匹配，

1.2 每个分组的高速缓存器单元通过一个树形结构转换成上置的高速缓存器单元(ROOT－CT)，这个单元占据了指令存储器(ECR)的存取指令并中断指令，

1.3 指令汇总指令序列(KR)，可以整体存储并在存储器之间传输，

1.4 位于树形结构底层或中间层的每一个高速缓存器单元要求用于上置高速缓冲器单元的必要指令，

1.5 只要指令序列在本地存储器中，上置高速缓存器单元就会将要求的指令序列发送到下置单元中，

1.6 只要指令序列不在本地存储器中，上置高速缓存器就要求有必要的指令序列。

2．如权利要求 1 所述的方法，其特征在于，指令序列会完全消除代码。

3．如权利要求 1 至 2 所述的方法，其特征在于，如果本地存储器中要求的指令序列没有足够的载入空间，那么高速缓存器单元的指令序列就会清除代码。

4．如权利要求 1 至 2 所述的方法，其特征在于，指令序列中的一条指令(移动)可以通过高速缓存器单元中的指令序列来中断一条程序。

5．如权利要求 1 至 4 所述的方法，其特征在于，指令序列中的一条指令(执行)可以中继一条确定的指令序列的载入。

6．如权利要求 1 至 5 所述的方法，其特征在于，高速缓存器单元之间总线上的任意一条指令(执行、移动等)可以中断任意地址上的高速缓存器单元中和指令相匹配的作用。

7．如权利要求 1 至 6 所述的方法，其特征在于，一个程序序列设有有效的缓存作用，是因为这个序列只能用于一个高速缓存器单元中并且只能在小的分序列中分解，而许多高速缓存器单元需要分序列，一个附加的分序列(IKR)包括指令序列中没有缓存作用的部分和分序列中有缓存作用的部分。

7．如权利要求 1 至 6 所述的方法，其特征在于，每一条指令序列都符合统计

学，它给出了时效信息，也就是说高速缓存器单元和指令序列中缓存器的停留时间。

8．如权利要求 1 至 6 所述的方法，其特征在于，每一条指令序列都符合统计学，它给出了有关指令序列的调入情况。

9．如权利要求 1 至 6 所述的方法，其特征在于，每一条指令序列都符合统计学，它给出了指令序列的长度。

10．如权利要求 1 至 9 所述的方法，其特征在于，清除代码路径可以设计参数，它可以评估每个指令序列的统计性并去除和规则系统相匹配的不重要的指令序列。

11．如权利要求 1 至 10 所述的方法，其特征在于，清除代码路径和可编程规则系统相匹配。

00.10.30

说　明　书

按照层结构具有二维或多维可编程序的单元结构(FPGAs、DPGAs 等)的数据流处理器和模块的高速缓存配置数据方法

本发明的背景

技术水平

本专利说明书所依据的相关技术在专利申请书 196 54 846.2－53 对具有二维或多维可编程序单元矩阵(FPGAs，DPGAs 等等)的数据流处理器和模块进行自动动态的重加载的方法以及专利申请书 196 54 593.5－53(可编程模块的运行时间重构方法)中已被阐明。其中阐述了按照相关技术对 DFPs，以及 FPGAs，DPGAs 和类似的模块进行配置和重构的一种方法，采用这种方法时一种独立配置的中央高阶微控制器型模块承担了对多个低阶，且多半是无源的控制单对于配置数据的分配。

相关技术的缺点

通过使用一个中央全局单元控制对一个或多个模块的部分(例如单元)的重构，如果许多不同的重构请求必须同时进行处理的话，可能出现瓶颈。上述的模块平行性优点，受到了这种中央单元相当大的限制，因为这种机构显示了典型的“瓶颈”，并且明显地减慢了数据的处理。

另外将事件源指派给要被加载的配置表现出一些问题，因为配置存储器使用的是绝对地址。因此，该重构单元必须包含一种存储器管理系统，这种系统就象在操作系统中那样，还要有记录哪一部分存储器区域用于哪一部分配置的各种文档。

资源(例如 CELs)的管理是一个附加的问题，必须保证，每个 CEL 只能精确指派给由重构请求启动的每个算法一次特别是对于还使用了其余周围 CELs 的算法更是如此，否则会出现死锁。

为了再一次明确重构的问题下面举出例子；

CELs 的一个矩阵是被重构的并处于复位状态。每个 CEL 应当能指示，它是否处于重构的状态。在矩阵中的所有的 CELs 均已经处于被配置状态，因而处于重构状态。第 1 个配置例程(KR1)被加载，这时没有充分利用矩阵。被配置的 CELs

清除它们处于可配置状态的指示：与第一配置例程独立的第二配置例程(KR3)被加载到一组尚未配置的 CELs 中。第三配置不能加载，因为这需要第 1 和第 2 配置资源(KR3)的 CELs，可是第 1 和第 2 配置的 CELs 正在使用，因此还没有处于可重构状态。

KR3 必须停止直到需要的 CEL 释放，即 KR1 和 KR2 已终结为止。

在 KR1 和 KR2 执行过程中用于第 4 配置例程(KR4)和第 5 配置例程(KR5)的加载请求到达了，它们关不能立刻加载，因为它们使用的 CELs 正被 KR1 和 KR2 的使用，KR3 和 KR4 部分地使用了同样的 CELs，KR5 不利用 KR3 和 KR4 的 CELs。

为了重新加载 KR3－KR5，有如下的要求：

1. KR3－KR5 应当尽可能按加载请求顺序予以加载。

2. 应当尽可能加载许多互相独立的 KR，即不具有共同 CELs 的 KR，以便获得最大的平行性。

3. 这些 KRs 不应该互相阻塞，即 KR3 被部分加载，但不能继续加载，因为其它的 CELs 被部分加载的 KR4 阻塞，而 KR4 也不能继续加载，因为再次需要的 CELs 被 KR3 阻塞。这样会导致典型的死锁情况。

4. 生成了 KRs 的编译器不能识别和取消 KRs 的时间的相互作用，使得不出现冲突情况。

实施电路的代价和最佳的结果之间的比率必须尽可能良好，即发明的目的是提供灵活的，平行的，无死锁的配置，它能以昼小的代价化费适中的时间和计算资源被执行。对此必须解决以下基本问题：

－如果只有 KR3 被加载，这种方法没有死锁，但不是最佳的，因为 KR5 也会(可能)被加载。

－如果 KR3 被加载，KR4 未加载，然而 KR5(已被)和 KR4 必须被预先标记，让它在后面的加载顺序中具有最大的优先权，这说明要有高的开销。

由以下过程来确保无死锁操作：

本发明任务和达到的改善

本发明的基本任务是一个单元－以下称为配置表(CT)，它具有层次结构的，并且可在每一层次上出现多次，同时，CT5 的数目以最低结构分层到最高结构分层逐步减少，直到最高层次刚好只有一个 CT。每个 CT 各自独立地和平行地配置和控制多数可配置元素(CELs)。较高结构分层的 CTs 可以缓冲存储用于较低层的 CTs 的配置例程。如果若干个处于较低层的 CTs 需要相同的配置例

程，该配置例程可被缓冲存储在较高层的 CT 中，并且由各介 CTs 调出，同时较高层的 CT 只需要一次就可把有关的配置例程比一个全局共同的配置存储器中调出，这就获得了一个高速缓冲存储效应。在除了用于可配置模块外本发明还可在微处理器，DFP 等中具有多个算术单元的用作指令和数据的高速缓冲过程。同时可以按照应用需要取消几个下面所述的单元(例如 FILMO)，但是在层次结构上根本一点也没有改变。因此，这种使用可看作一个子集并且不再详述。与常规的高速缓存过程相比，上述方法的一个明显的优点是，数据和/或代码是有选择性地被高速缓冲，即使用了精确地适用于算法的方法。

同样，本发明能使大的单元结构被重构而完全无死锁现象。

发明的说明

代替象至今统一在模块中的一个集中的和全局的单位，该单位处理所有的配置要求，因有许多划分成等级(枝叉结构)排列的活动的单位，这些单位能够承接这些任务。

同时，最低平面的一个要求，如果该要求不能予以处理的话，那么只能继续导入下一个高一些的平面上。对所有存在的平面均要重复该步骤，直到达到最高的平面为止。

最高的平面连接一个置于内部的或置于外部的配置存储器，该存储器包含着所有适用这种程序运行所需要的配置数据。

通过配置单位的枝叉结构达到配置数据的一种隐藏处(内存)。

存取到配置上主要是以局部进行。如果有关的数据在划分成等级排列的 CTs 中不存在，那么在最不利的情况下必须把来自于置于上面的配置存储器的一个配置下载。以固定的时间的顺序输入应予以下载的配置，并且把这些配置归纳成为一个明细表，这样就避免了死锁。在下载前固定 CEL 的配置信息，这样在完成配置的整个明细表的工作过程中，CEL 的状态信息保持不变。

CT 的基本原理

一个配置表(CT)是一个活动的单位，它是根据同步，信息所谓的触发器，作出反应。触发器是通过一个两维或多维矩阵，由电子结构组件，用于通常的算术的或逻辑的单位，地址发生器，计算单位等等——以下称作可配置的元件(CEL)——所产生的。籍助于出现的触发器触发在 CT 内的一个规定的行动。这

时 CT 的任务是控制多数的 CELs 并且规定它们的算术的和/或逻辑的运算。尤其是必须对 CELs 配置和变化配置。该任务 CT 承担，CT 管理多数可能的配置例程(KR)，这些例程在它们方面总是由单独的配置词语(KW)的多数组成，并且 CELs 的多数根据触发器条件以一个或若干个 KR 配置。同时，一个 CEL 总是包含一个或多个配置词语，这些配置词语配备了需要配置的 CEL 的地址。此时必须把一个 KR 完整地及准确地描摹到 CELs 的多数上，同时可以把若干个 CELs 联合成组。这些组以不同的但结构完整的 KRs 配置。在一个组内的所有的 CELs 进行错接，按照一个必需的重构的规定，通过一个共同的信号(ReConfig)通知所有编入组内的 CELs，使每个 CEL 结束数据处理并且必须转入一个重构的状态。

无死锁的重构的基本原理

在达到运行时间的重构的系统上出现这样的问题，该系统会到达一种状态，在这个状态中总是有两部分互相等待，以致于进入了死锁状态。

通过一种新的配置总是只能完全地或者一点也没有地下载到系统中，或者使用一种 Timeout 方法，这个问题是可以避免的。

这样就产生了一系列的不利情况(需要的位置，运行时间等等)，例如：

－如果一个配置不能下载的话，走到前面。

－配置下载的顺序的管理。

－动作中断，因为没有注意可能会下载到 CELs 中的其它的配置。

采用下面说明的方法可以排除这些问题。这是根据技术水平从一个 DFP 系统出发的。

从一个 CEL 开始，一个触发器信号发送给一个 CT。该 CT 确定触发器源并且通过一个 Look－Up 表选出一个应下载的配置(KR)。阻止触发器信号继续深入，在当时的配置全部做完之前不再继续接受触发器。一个配置由若干个命令组成，它被传送给一定数量的 CELs。当然，在一个达到运行时间的可配置的系统中不保证每个配置命令(KW)也能执行。这可能例如碰到下面情况而失败，已有地址的可配置的元件(CEL)还没有结束它的任务，这样就不能收到新的配置数据。为了避免动作中断，所有不能做完的配置命令(因为相应的 CEL 处于一个不可重构的状态，并且拒绝配置(REJECT)，按照一个 FIFOs 在一个(后面作进一步的说明)专门的存储器(FILMO)中的最后的配置命令之后写出。然后，按照相同的方法，做完以下的配置命令。这一直予以重复，直到达到一个配置的终结为

止。

然后，CT 行进，又进入接受触发器的状态，以便可能继续下载配置。在这种状态下，CT 通过一个计时器控制，每隔一定的时间做完 FILMO。

在对原来(本来)应下载的配置进行处理之前，CT 通过存储器 FILMO，达到对这些应下载的配置的优先。通过 FILMO 的一个 FIFO 类似的结构来保证，在前面的触发器要求过程中不能完全做完的 KWs，在新的应做完的 WK 之前自动保持一个比较高的优先权。在存储器 FILMO 做完时，对每个通过一个配置命令确定地址的可配置的元件，在发送一个 KWs 之前或发送一个 KWs 的过程中，进行试验，以确定其是否处于“重构”的状态。如果状态是“可配置”的(ACCEPT)，那么传输，并且把数据从存储器 FILMO 清除。如果状态为“不可重构”(REJECT)，那么数据保留在 FILMO 中，并在下次通过时重新做完。CT 处理 FILMOK 中的下一个项目。

这种过程一再重复，直到 FILMO 达到终结为止。然后把通过触发器信号的出现而激活的原来(本来)的配置予以做完。同时 FILMOs 的结构符合 FIFO 的原则，即首先处理最老的项目。如果没有新的 KR 下载，为了能把 FILMO 也做完，则由一个时间器控制，每隔一定的时间通过 FILMO。

其它的，没有参加的可配置元件(CEL)在该阶段平行地继续地工作，并且不影响其功能。这样就会出现这种情况，即在 CT 做完 FILMO 的过程中，一个或几个可配置元件(CELs)转为“重构”状态。由于 CT 随着做完会处于 FILMO 中的一个任意的位置上，因此可能出现下列情况 ：

CT 试图做完第 1 个命令，其已有地址的配置的元件(CEL)不是处于“重构”的状态。这样 CT 就带着下一个命令(KW)继续前行。在同样的时间一个或几个可配置元件转入“重构”状态，其中也有通过第 1 个配置命令可能已作说明的可配置元件。CT 处理第 2 个配置命令(KW)，该配置命令利用这种相同的可配置元件(CEL)，就象第 1 个配置命令那样，当然，这是来自于一个不同的配置。到这一时刻，可配置元件(CEL)处于“重构”状态，并且这命令能够成功地做完。

由此不再保证，应当首先下载的配置实际上也首先予以完成。那么为了完全地下载，会存在两个局部完成的配置，这两个配置总是需要其它配置的可配置的元件。进入了在图 18 中说明的一种死锁情况。配置 A 和配置 B 应当予以配置。CT 已经把配置 A 和配置 B 的划影线的部分予以下载。配置 A 为了完成还需要配置 B 的浅双影线的范围，以及配置 B 为了完成或还需要深双影线的范围。

由于两个配置还没有完全结束，这样也就没有作用功能，因此，对于两个配置没有进入终结状态中，两个配置的一个配置要予以删除。两个配置等待，仍然需要的可配置元件予以释放。

在本方法中采用下列方式避免死锁，即 CT 在做完 FILMOs 之前把握住所有可配置元件的这些状态，并且在结束该过程之允许再发生变化，或者不能识别出现的变化。换言之，或者固定在做完 FILMOs 之前所有可配置元件的这些状态，或者避免在做完 FILMOs 过程中这些状态的变化。一种可能实现的技术结构为，在每个可配置元件中使用一个寄存器，在该寄存器中固定做完 FILMOs 之前的状态。CT 只根据把握住的状态工作，而不能跟随着可配置的元件的当前的状态。这样一来确保了每个应作处理的命令(KW)找到可配置的元件(CELs)的相同的状态。这个步骤不排除，在做完 FILMOs 的过程中，一个或几个可配置的元件转入“重构”的状态。这种变化对于 CT 来说，在进行处理的过程中并不是立刻就可以发现的，而是直到开始下一次通过时才发现。

配置顺序

准确地保持写入 CEL 的 KW 的顺序，对于达到已确定了算法的配置是绝对必要的。例如，把 CEL 连接到一个接口系统之前，首先应对接口系统配置，这样该 CEL 就不会连接到由其它程序所利用的接口上去了。换言之，如果事先能对相应的接口连接配置，CEL 也只能配置了。

按照发明的方法，如下所述应保持一个固定的流程：

配置词，其输出对于后面的 KWs 的配置是十分重要的，尤其要作出标记(以下简称 KWR)。如果一种这样的 KWR 打不中，那么在有关的配置程序内的所有的后面的 KWs 均要写到 FILMO 上，并且在该通过中不再输出。即使在 FILMOs 通过时，处于一个 KWR 后面的顺序中的全部的 KWs 在当时的流程中也不输出。

内存方法

CT 结构是划分成等级建立的，即在一个模块中有几个 CT 平面。这种排列主要按照一种树枝结构(CT－Tree)。同时，根 CT(Root－CT)是一个外部的配置存储器(CER)，该存储器按配位包含全部的 KRs，而可配置元件(CELs)排列成树叶片，它们调入各外 KRs。以相同划分成等级的可配置元件总是配合均匀平面的 CT。

每个 CT 分配一个局部的内存储器。如果新的 KRs 需要贮存而没有位置，或者这 KRs 明显地要求通过一个专门的 CT 命令(REMOVE)的话，该存储器局部地予以清除。此时根据一种清除策略实现 KR 方式的清除，最好的情况是，只清除那些不再要求的，或者明显地要通过 REMOVE 命令说明的 KR。

同样，这些 KR 单独予以清除，清除的数量正好达到，为了把新的需要下载的 KR 写入到存储器中去，存储器恰好必需空出的位置为止。

该优点在于，每个处于一个随机的 CTx 下的 CT，则下面在 CT 树枝(主干)的上部有一个 KR，这个 KR 贮存在 CTx 中，不是通过外部的配置存储器 ECR 要求的，而是直接由 CTx 获得。以此产生了几个平面的内存结构。在 CT 树枝(主干)中的数据传递化费以及尤其是 ECR 所需要的存储器带宽明显降低。

换言之，每个 CT 中间贮存着处于它们之中的 CT 的 KRs。即处于比较低的位置的 CTs 直接从处于比较高的位置的 CTs 获得需要的 KRs，不需要把存储器存取到外部的 ECR 上。只是如果在一个处于高一些 CTs 中已经没有所需要的 KR，那么才必须通过存取到 ECR 上下载 KR。这样就得到了一个对于 KRs 特别有效率的划分等级的内存结构。

根据这种结构也能获得清除策略，当然应当视用途的不同根据经验来确定策略。可以有几种方法：

—清除最老的项目

—清除最小的项目

—清除最大的项目

—清除调用最少的项目

CT 划分等级的基本原理

为了获得一个内存效应，CTs 联接一个在树枝结构(主干结构)中的划分的等级。在各个接头(CTs)之间有一个接口系统(Inter-CT-Bus)，总是一个上面的接头(CTs)连接几个下面的接头(CTs)。同时，下面的接头(CTs)从上面的接头(CTs)那里要求数据，上面的接头把数据输送给下面的接头。下面的接头互相之间交换状态信息，于此在比较高的接头之间使用网络，这些网络必须按照地址予以解开的。

CT 划分等级和定地址

CT 划分等级是这样安排的，即可以使用一个二进制树枝(主干)进行各个 CTs 选定地址。这表示，最低价的地址二进位制标出树枝(主干)的各个叶片(分片)并且每个进一步的地址二进位制总是选择比较高的一个划分平面。因此每个 CT 就具备一个明确的地址。

以下的表格表示，各个地址二进位制如何分配各自的平面：

3	2	1	0		地址宽度
-	-	-	*	平面 0：叶片	1
-	-	*	*	中间平面： 1	2
-	*	*	*	中间平面： 2	3
*	*	*	*	中间平面： 3	4

* ＝已使用地址二进位制

- ＝没有使用地址二进位制

如果一个置于上面的 CT 配入一组 CTs，则相应地汇总这组的几个地址二进位制。

下面的表格表示，各个地址二进位制如何分配各自的平面，同时在平面 0 上的一组带 8 个 CTs(地址二进位制 2…0)：

5	4	3	2…0		地址宽度
-	-	-	*	平面 0：叶片	3
-	-	*	*	中间平面： 1	4
-	*	*	*	中间平面： 2	5
*	*	*	*	中间平面：3	6
				…	

* ＝已使用地址二进位制

- ＝没有使用地址二进位制

二进制树杆(主干)的结构可以是一维或多维，采用的方法是，每个维建立一个二进制树杆(主干)。

一个已确定的 CT(TARGET)选定地址，采用的方法是，这个要开始的 CT 或

者说明精确的目的地址，或者对 TARGET 相对选定地址。

下面进一步地说明对一个相对地址的处理：

这是一个用于二维选定地址的相对地址区域的例子：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
高划分等级的 CT	传播	+/- Y-地址							+/-X-地址						

如果应当选择下一个高的划分等级的 CT，则代入比特 15。比特 14 标出传播，那么选择所有的 CT。X/Y 地址列出从 Initiator 地址发出的 TARGET 地址。

这些地址为带有符号的“signet”整数。通过相对于现实的地址位置的地址区域有 Y/X 地址的加法确定 TARGET。每个平面具备一个规定的地址宽度(Address-width)。加法器按照这个宽度。

在进行加法时有上越界或下越界则说明，已选定位置的 CT 不是处于实际的接头的下面，并且向处于这上面的 CT 继续给出位置要求(后面上一位的接头)。

如果没有出现上越界或上越界，TSRGET 位于实际的接头下面。在实际的平面上计算出的地址二进制位(比特)(相应的表格)这样直接处于实际接头下的 CT。由此出发，根据相应的地址二进制位(比特)总是选择下一个较低的 CT(接头)。

在 CT 划分等级中存取的优先权

存取到 Inter-CT-接口上是由一个促裁器进行管理的，在这方面，所有下面的接头有同样的优先权。上面这个接头具有高的优先第一位。因此，存取就是从一个较高的接头向下面传递，或者已经把一个由 INITIATOR 开始的宽的通道放在一边，考虑了其它的存取。

一个 CT 的基本结构

下面的有关 CT 的概要可以综合说明各个结构组类的情况。下面对结构组类进行详细的说明。

一个 CT 的核心是控制状态机(CTs)，该控制机控制配置例程(KRs)全部做完。配合 CTs 的有，Garbage 收集器，该收集器控制从 CT 的存储器(CTR)中清除 KR；FILMO，该 FILMO 管理仍需要做完的 KWs，以及 LOAD一状态机，该机器

控制 KRs 的下载。

该存储器(CTR)的装备是作为通常的书写－阅读－存储器，同时可以用来执行所有的技术任务，以及用来对各自的 CT 以及置于其下面的 CTs 的 KRs 进行局部贮存。作为特殊情况存储器(CTR)也可以装备为 ROM，EPROM，EEPROM，Flash-ROM 等等，为了给模块配备一个固定的，ASIC 或类似 PLD(见技术的水平)的功能作用。

为了设计 CTR 地址使用 4 个作为下载的计数器装备的指示器：

1.空位指示器(FP)。显示在 CTR 中最后的 KR 之后的第 1 个空的存储器位置。

2.Garbage 指示器(GP)。显示通过 Garbage 收集器(GC)应当从 CTR 中清除的一个项目。

3.运动指示器(MP)。显示在 CTR 中的一个存储器位置，从这个位置上一个有效的，不应当清除的配置词(KW)，就是说一个 KR 的一个项目，向着通过 GP 确定的项目上复制/运动。

4.程序指示器(PP)。显示瞬时由 CTs 输出的 KW。

KW 通过一个输出分界面继续输向有关的 CELs 上。一旦它们是在一个重构的状态接受 KE 的话，CELs 应答(指令)(ACCERT)。如果一个 KW 没有被应答(REJECT)，那么在一个类似 FIFO 的存储器中根据时间予以中间贮存，以便在后面时间内，不利用程序指示器，而重新在选定地址的 CEL 上写出。

做完一个 KR 的要求包括通过触发器信号的这些 CTs。触发器信号经过一个屏蔽，这是一个滤波器，它把不需要的触发器滤掉。按照技术水平可以通过 UND 取能元件建立一个屏蔽，该屏蔽把一个触发器和一个输出信号进行 UND 连接。触发器经过具有优选作用的 Round-Robin-仲裁器(SCRR－ARB)转换成二进制信号。一个具有优先作用的 Round-Robing-仲裁器把一个 Round-Robing-仲裁器的同步解答的优点和以一个节拍对下一个输出的识别结合在一起，就是说，这是一个优先第一位－仲裁器的优点。

已屏蔽的触发器作为地址连接到一个第 1 Lookup 表(LUT1)上面，这是一个存储器，这个存储器把有关的 KR 的 ID 配位给作为地址进入的触发器，并且在数据线路上输出。

在一个第 2 Lookup 表(LUT2)中，KR 的 ID 配位给在 CTR 中的 KR 的存储器位置的地址。该第 2 Lookup 表不仅进行触发器信号的配位，更多的是利用命令，

这些命令使用一个 ID 作为参数，LUT2 同样进行地址配位。

触发器信号针对有关的 Ids 的配位是下面阐述的命令(REFERENCE)记入到 LUT1 中。LUT2 的管理，就是说 Ids 针对 CTR 中的地址的配位，通过 CTs 和 GC 自动发生。

为了更好地理解 CT，下面列出一个可能的命令句子：

1. BEGIN <ID>

通过 BEGIN <ID> 表示一个配置例程的开始. <ID>说明配置例程的明确的识别号。

2. STOP

通过 STOP 表示一个配置例程的结束。在这个位置上，配置表(CT)结束配置例程的处理。Garbage一收集器(GC)结束该配置例程的项目的清除。

3. EXECUTE <ID>

跳向一个配置例程的开始(BEGIN <ID>)。如果该配置例程没有在该 CT 的存储器中，那么由置于其上面的 CT 对其予以要求，或者从存储器中出来予以下载。

4. LOAD <ID>

要求置于其上面的 CTE KR <ID>。

5. REMOVE <ID>

调入 GC，以便肥从开始(BEGIN <ID>)到结束(STOP)的配置例程<ID>从该 CT 的存储器中予以清除，并且把后面的配置例程不断推进，直到不存在由于清除配置例程而形成存储器孔为止。

6. PUSH <FORCED> <ADDRESS> <DATA> <EXIT>

把配置数据<DATA>写到寄存器上<ADDRESS>。如果代入<FORCED>，即使不代入有关的目的寄存器的 RECONFIG一Flag(旗)，也要写上数据。使用<EXIT>并应当显示，这关系到一个 KWR，它通过 REJECT 中断后面的 KWRs 继续输出。

7. MASK <SR> <TRIGGER>

采用<TRIGGER>代入触发器屏蔽，或采用<TRIGGER>把它复位，取决于<SR>(Set/Reset)。

8. WAIT <UNMASKED> <TRIGGER>

停止配置例程的处理并且等待触发器<TRIGGER>。如果代入了<UNMASKED>，则不管触发器一屏蔽的状态，记录到所希望的触发器上。

9. TRIGGER <TRIGGER> <CT#>

把一个触发器的二进制值发送给置于上面的通过 CT＃选定地址的 CT。

10. GETBUS/GETCTS

建立相对于 Inter-CT-Bus 的连接。

11. LOOSEBUS/LOSECTS

解除相对 Inter-CT-Bus 的连接。

12. REFERENCE <TRIGGER> <id>

把数值<ID>在地址<TRIGGER>条件下写入 LUT1，以此把一个规定的 KR 配位给一个触发器信号。

这些命令 EXECUTE，LOAD，REMOVE，PUSH，MASK，WAIT，TRIGGER，REFERENCE 只在 BEGIN…STOP 括号以内有效。在这个括号以外不输出这些命令。

一个配置例程(KR)的结构看上去就象如下这些情况。

BEGIN <ID>;

…

适用的命令

…

STOP;

间接寻址(引用)

CT 中的高速缓冲存储器原理，能够在 CT 上对 KR 进行缓冲存储，并可从多个不同的较低层 CTs 或 CELs 中使用 KR。

若较低层的单元访问模块(如 RAM，外围设备)的外部接口，则有必要存储不同的地址或外部接口的各个部分。这将使所需的各个 KR 的内容根本不同。无高速缓存的可能性。

间接引用提供了一种补救办法。为此须使用包含和设定了必要的外部参数的特殊 KR(以下称 IKR)。也许其它不同的 KRs 可以经过触发器调入到不同的分级层面。从 IKR 的末端调入实际的 KR。不过，IKR 是无法高速缓存的，而被调入的 KR 是完全一致的，因此是可以高速缓存的。建议将 IKR 的大小减少至绝对最低值，也就是外部的和不同的参数以及对一致的 KR 的调入。

间接配置例程(IKR)可如下设定：

BEGIN <ID>;

…

***;有效命令，仅可启动外围设备，

TRIGGER <ID>；启动、停止或加载对外围处理的请求。

…

GOTO <ID>；跳跃至一致的 KR。

特例：

1.WAITL_FOR_BOOT

此命令仅适用于 CTR 的第一地址。在引导过程中，完整的 Boot-KR 初始化时被写入 CTR，但不写入 Boot-KR BEGIN <0>的开始序列。在 Boot-KR BEGIN <0>处(地址 1)是 WAIT－FOR－BOOT,可在 RESET 时自动设置。只有当整个 Boot-KR 被写至 CTR 之后，WAIT－FOR－BOOT 可用 BEGIN<0>进行复盖，CTS 开始处理 Boot-KR.

WAIT-FOR-BOOT 不得在一个程序内出现。

2.BOOT<CT-ID>

BOOT<CT-ID>表明了后面的 Boot-KR 应写至哪个 CT 中。在 Boot-<CT－ID>之后不是 BEGIN，Boot-KR 不是通过 STOP，而是通过随后的一个 BOOT<CT_ID>关闭。STOP 结束引导过程。

BOOT<CT-ID>不得在一个程序内出现。

引导过程

在 RESET 之后，最上面的分级面(ROOT－CT)的 CT 将 Boot-KR 调入下面分级面的 CTs 中。因此，在到达一个所规定的地址(BOOT－ADR)时出现跳跃，即列入 ROOT－CT 的外部配置存储器(ECR)。ROOT－CT 执行此跳跃，获得引导程序。可进行如下设置：

BOOT<CT-TD1>;COMMAND;COMMAND;…

BOOT<CT-TD0>;COMMAND;COMMAND;…

…

BOOT<CT-TDn>;COMMAND;COMMAND;…

STOP;

在引导过程中，整个 Boot-KR 首先写至从通过<CT-ID>列出的 CT 的地址 2

开始的 CTR 中。Boot-KR(BEGIN<0>)的开始程序并不是写至地址 1 中。在此位置上是 WAIT－FOR－BOOT，可在 RESET 时自动设置。只要当整个 Boot-KR 被写至 CTR 之后，并且 ROOT－CT 到达随后的 BOOT<CT-ID>,STOP 才可被写至 Boot-KR 末端的 CTR 上，用 BEGIN<0>修改 WAIT_FOR_BOOT。CTS 开始处理 Boot-KR。

配置程序的调入

为了在 Boot-KR 之外请求一个配置程序，须有三大基本机理：

1. 通过 CTS 执行 LOAD<ID>

2. 通过 CTS 执行 EXECUTE<ID>，在 CTS 中并不含有相应 ID 的 KR。

3. 经 LUT1 传输至一个<ID>的触发器的出现，在 CTR 中没有所属的 KR。

这三种情况的过程是一样的：

在作为地址的 LUT2 中列出被请求的 KR 的 ID。此 LUT2 检查，是否在 CTR 中有一个有效的地址。若没有，也就是说，<ID>在 LUT2 中显示为 0 值，则 load<ID>被传至 CTS。

接着，在分级的上级 CT 时，CTS 请求与<ID>有关的 KR。此请求在一个触发器的模子中到达上级的 CT，并相应地对其进行计算。

此上级的 CT 将被请求的 KR 传送到发出请求的 CT。这些数据从 FREE－POINTER(FR)所显示的地址起被写进 CTR，此 FR 在每一次记录存取后提高一位。

如果 FR 到达了 CTR 的上限，则须调入无用数据集合器(GC)，以清除 CTR 内的最下层 KR 以及压缩 CTR。FR 就这样重新设置了。此过程一直到即将调入的 KR 完全与 CTR 相配时才算结束。

配置存储器的跳跃表

分配到 ROOT－CT 的配置存储器包括在使用时必须调入的所有 KR。在外部配置存储器(EDR)中，在一个所规定的地址(ADR－BOOT)有一个进入引导配置程序的跳跃。而在另外一个所规定的存储器范围(LUT－ECR)中，任意的然而在使用内部预定的长度可跳跃到各自的 KRs。任何一个 KR 的<ID>可在 ECR 中作为地址使用，而任何一个 KR 的起动地址均在那儿；KRs 间接寻址如下：

ID→LUT－ECR→KR

在配置存储器中改变 KR

应改变 ID<A>的 KR。首先，HOST 将 ID<A>中的新 KR 写到 ECR 中的一个空余的存储器位置。ID<A>将与配置存储器中的 KR 新地址一起，从上级单位(HOST)写到一个预定的 ROOT－CT 寄存器上。此 ROOT－CT 将命令 REMOVE<A>发送到所有下级 CT 中。然后在达到 STOP 或者在 IDLE 循环(只要不执行 KR)时，所有 CTs 消除了 CTR 中 的与 ID 发生关系的 KR，将 LUT2 设置到“NoAdy”的地址<A>，也就是说，在 LUT2 的 ID<A>中不存在有效的地址输入。若 ID<A>重新获得请求，进入 LUT2 的位置<A>处的那个缺少的输入(“NoAdy”)强制每个 CT 重新请求 ECR 中的 KR<A>。

FILMO

KR 主要由命令 PUSH 组成，此 PUSH 可将新的配置语言写到一个特定的地址上。如果无法写入型号 KW 的配置语言，因为此具有地址的可以配置的元件(CEL)不愿意接收新配置(REJECT)，则此配置语言并非被写到那个具有地址的可以配置的元件(CEL)而是被写到下面称之为 FILMO 的一个存储器上。后面的命令可得到正常运算，直到一配置语言无法重新写入为止，该配置语言然后再写 FILMO 中。

若无法写入型号 KWR 的配置语言，因为此具有地址的可以配置的元件(CEL)不愿意接收新配置(REJECT)，对此配置语言并非被写到那个具有地址的可以配置的元件(CEL)而是被写到下面称之为 FILMO 的一个存储器上。所有后面的命令(至 KR 末端以下)封锁地写到 CEL 而是直接写入 FILMO 上。

FILMO 将在 IDLE 循环时和在一个新的 KR 每次执行前得到完全运行。因此，在启动最老的数据字时并且与技术状态的 FIFOs 相一致，任何一个被读出的 FILMO 语言须传送至具有地址的元件上；因而，此具有地址的元件必须乐意接收此配置语言。只要这些数据字从一开始就可以写入的话(也就是说，那些具有地址的可以配置的元件(CEL)已作好准备)，则必须按照一个 FIFOs 的方式清除 FILMO 中的输入。如果无法写入配置语言，那么须跳过它，不要从 FILMO 中清除。与一个 FILMO 相反的是，这些数据可以在跳过的配置语言之后继续读出。在一个跳过的配置语言之后可以写入的配置语言可根据 FILMO 的自动编码或者①：写入时作下标记，不要从 FILMO 中清除，这样写入时作下标记的配置语言在随后的运行时不能被读出，而是马上被清除，只要没有一个被跳过的配置语言在它们前面出现的话；或者②：从 FILMO 清除，这样这些配置语言在被清除

的配置语言之前和之后均继续存在，因而为了得到清除，须将后面的语言移至前面(上面)或者须将前面的语言移至后面(下面)，因此无论如何须保留配置语言的顺序。

若执行新的 KR，则那些从 CTS 中无法写入具有地址的元件(CELs)的配置语言(KW)将重新跟踪至 FILMO，也就是说，KW 将写入 FILMO 的末端(读带方向)。如果 FILMO 已满，即不存在配置语言的空余输入，则 KR 的执行将终止。FILMO 一直等到写入了足够的配置语言而且相应地出现了许多空余的输入时才结束运行，而 KR 将继续执行运算。FILMO 提供了一个与 FIFO 相类似的存储器，可始终从最老的输入开始直线运行，而与 FIFO 不同的是，可跳过输入(首先在线性多路出口)。

配置表 Statemachine(CTS)的功能

CTS 行使对 CT 的控制。它执行 KR 的命令，并反应到触发器上，它行使对 FILMO 的管理，可以在 IDLE 循环以及在执行 KR 前读出 FILMO。它反应到由 LUT 结构所产生的信号 illegal<TRG>(非法触发器，可参见图 1，0102)和 load<ID>。当 LUT2 上出现高速缓冲存储器失败(0105)或者通过 ID 而查询的 KR/IKR 消除标记(0107)时，才会产生 load<ID>。它反应到上级 CT 的控制信号上。

处理命令的自动编码举例，请参见图 2－7。

控制信号的上级 CT

－illegal<TRG>(0102)

向上级 CT 表示，出现了一个不明触动器<TRG>。

－load<ID>(0105/0107)

询问上级 CT 以调入<ID>。

－trigger<TRG>(CT#)(0108)

发送一个触动器(TRG)到上级的或者具有地址的 CT<CT#>。

上级 CTs 的控制信号

－remove<ID>(参见图 15，1513)

询问 CT 以清除<ID>

－write_to_FP<data>(参见图 2，0205)

发送数据到 CT。数据跟踪到被占用的存储器的末端。

无用数据集合器的功能(GC)

CTR 受到两个问题的制约：

1. 指点 LOAD 或者 EXECUTE 命令以及触发器注意 ID，ID 中的 KR 在 CTR 中是没有的，必须由 KR 自行稍后调入。然而有时候在 CTR 中调入被请求的 KR 时又没有足够的空间。

2. 出现 REMOVE<ID>时须从 CTR 中清除相应的 KR。这样，只要不位于 CTR 的末端，就会出现一个间隔。在调入新的 KR 时，有时候此间隔不能重新完全被填满，或者此间隔对新的 KR 来说是太小。这将会导致一个 CTR 的局部图。无用数据集合器的任务就在于，从 CTR 中清除 KR，以便为新的输入创造空间而且在清除输入后重新组织 CTR 时须将所有剩余的 KR 作为闭合程序块串联在存储器上，将那些空置存储块作为一个闭合程序块安置于 CTR 的末端。

这样，新的 KR 就可以在最佳方式和毫无损失地补充调入到存储位置上。

触发信号的利用

每一个 CT 均可连接到属于其各自的分级面的多个触发器信号上，这些触发器信号再聚集到总线上。这些详细的触发器将在一个屏蔽上得到利用，也就是说，仅能中继传输自由连接的触发器信号。这些自由连接的触发器信号将节拍同步地中间存储在一个抽样寄存器中。操作员可选择其中的一个被存储的触发器信号，将此信号转化成一个二进制向量。被选择的触发器信号可从抽样寄存器中得到清除。此二进制向量可中继传输到第一个搜索图 1(LUT1)，该 LUT1 再将二进制向量转换成得到请求的配置程序(KR)的标识号码(ID)。此 ID 在第二个搜索图(LUT2)中再转换成 CT 存储器(CTR)中的 KR 地址。CTS(CT－Statemachine)将其程序指示器(PP)设置到这个地址上，并开始执行 KR。前提条件是，任何一个在屏蔽上自由连接的触发器在 LUT 外均有一个相应的输入。如果没有，那么这一错误状态将中继传输到 CTS 上(非法触发器)，因此等同于“NoAdr”的每一个 ID 可计算为非存在的输入。“NoAdr”是一个依赖于自动编码的可选择的标识。

若在 LUT2 上缺少输入的话，也就是说，与 ID 有关的 KR 并不在 CTR 上，那么调入请求须被发送至 CTS 上(load<ID>)。

触发信号发送到上级 CT

在已说明过的到达上极 CT 的接口以调入 KR 之外，还存在着另外一个接口，可对自由定义的命令尤其是触发器向量进行互换。因此一个 CT 或者向所有其它 CTs 发送一个命令(BROADCAST)，或者向任意一个具有地址的 CT 发送一个命令(ADDRESSED)。

此命令“触发器向量”(Triggervector)提供了一个二进制值，可在接收到 CT 的 LUT2 中加以输入时查询。

发送触发器向量是很有必要的，如在一个 IKR 内部在另外一个 CT 中起动 KR，如调节外围设备或存储器。

为了中继传输触发器向量到一个上级 CT，须存在两个机理：

1.此 LUT1 须补充一个比特，说明存储器的内容被视作 KRID 或者被视作触发信号的二进制值。若有一个触发信号，则 LUT1 的数据内容可直接作为触发器发送至上极 CT。

2.一个触发器的二进制值以命令 TRIGGER 就可说明它可直接发送至上级 CT。(也可有选择性地直接传输 ID 而不是触发器值)。

在触发器向量上的一个外部 CT 中起动 KR 时，为获得无死锁必须实施同步方法。此方法必须注意的是，仅仅是在 CTs 特定的小组内的一个 KR 在这个小组内的其它 CTs 上起动其它 KR。多个 KR 的起动同时可以导致 CTs 之间的死锁，与 CEL 等级上已经描述过的死锁相类似。

上述方法的基本原则如下：

一个 KR 的结构如下：

…

GETCTS/GETBUS

TRIGGER<ID>,<CT#>

TRIGGER<ID>,<CT#>

…

LOOSECTS/LOOSEBUS

…

CT(INITIATOR)中的 KR 内部的命令“GETCTS”表明下面信号将发送到其它 CTs〔TARGET〕中。用 Trigger<ID>,<CT#>，一个正在起动的 KR 的 ID 将随同唯一的 IDCT＃发送到 CT 上。触发器的发送首先到直接的上级 CT，它与 CT＃相一

致将触发器发送至 CT 范围内的一个新的下级 CT 或者发送至它那儿的上级 CT(可参见 CT 分级)。若这一命令到达了 TARGET，则 TARGET 可应答接收。

在通过一个 CT 的命令运行时，此命令的优先识别总是上升一位。结果一个命令的中继传输请求碰到一个 CT 内的另外一个请求时，那这一命令将会拒绝最低级的优先。因此 a)须确保，在一个相冲突的系统内同一时间只传输一个因而也只能起动一个 KR，而一个 KR 可引起所需要的无死锁。

b)须确保，拒绝迄今为止至少仍在传播的命令，而此命令可引起性能的提高。

在拒绝一个命令后，在 GETCTS/LOOSECTS 内部的所有上述的命令均被同样拒绝，也就是说，INITIATOR 向所有 TARGET 发送信号 DISMISS，而执行 KR 将在一段等候时间之后重新在 GETCTS 时起动。

在一个指令区段 GETCTS…LOOSECTS 内的所有触发器的应答将被发送到 INITIATOR－CT。在每一次即将到达的应答时，对下一个指令的处理仍将继续进行。

在获得指令 LOOSECTS 时，INITIATOR 向所有 TARGET 发送了信号 GO。因此 TARGET－CTs 与由触发器传输的 ID 一起起动执行 KR。

TARGETs 在出现一次触发器之后变化成下面一种情况，即此时它们在等待着 GO 或 DISMISS 信号的出现。

由于更好的可自动编码性，另外可以看到一种很容易修改的方法：在分级层面的一个小组的 CTs 之间有一个总线系统(Intex-CT-Bus)。此总线系统连接了所有该小组的 CTs 和一个直接归属于该小组的 CT。

通过此指令 GETBNS(与功能性的 GETCTS 相类似)，此总线系统将由一个 CT 判断。这些指令将经此总线系统被中继传输至同一小组的 CTs。如果在该小组内没有具有地址的 CT#，则可通过此上级 CT 自行判断其上级总线，此指令将得到中继传输。这些受到判断的总线将仍然分配 INITIATOR 并因此禁止所有其它的 CTs，直至要么发生拒绝，要么指令 LOOSEBUS 分解总线为止。LOOSEBUS 可与 LOOSECTS 相比较。在执行指令 LOOSEBUS 前，GO 信号被发送至所有参与的 CTs。这可以要么通过指令 LOOSEBUS 要么通过一个特有的预起动的指令进行。指令和触发器将同样按照早已描述过的基本方法予以处理。当一个总线系统无法得到判断时才会发生拒绝。在判断时，一个等级的 CTs 总是马上获得优先，上级 CT 具有更高的优先。在 Intet-CT-Bus 上面发送一个指令时，此指令必须一直保持

活动状态，直至具有地址的 CT 接受该指令(ACCEPT)或拒绝(REJECT)为止。

优先的 Round-Robin-仲裁器

优先的 Round-Robin-仲裁器(单循环-Round-Robin-仲裁器 SCRR-ARB)具有时钟同步的构造，即在每个-根据执行情况正的或负的-时钟周期间隔(TFI)仲裁器都提供一结果。进入信号(ARB-IN)通过一个屏蔽(ARB-MASK)传送，这个屏蔽由仲裁器本身根据下述过程自动管理。屏蔽的输出信号根据相关提供给技术优先仲裁器(ARB-PRIO)，对每次周期间隔(TFI)仲裁器与系统时钟同步地输出一个结果(ARB-OUT)，即：在屏蔽(ARB-MASK)后的最高优先级的信息的二进数值。一个信号(VALID)被指定给该结果，此信号表示这个二进数值是否有效。根据优先仲裁器的实施情况可以在有信号 0 时和在无信号时产生同样的二进数值：在这种情况下 VALID 表示：如果无信号存在则结果是无效的。这个信号是：

1. 作为仲裁器的结果被输出。

2. 送到一个解码器，这个解码器把一个 3-位二进数值解码为例如下表所示的二进数值。(这个编码过程按照此原则适合于任何所需的二进数值)。

二进数值	解码	注释
(ARB-OUT)	(ARB-DEC)	
111	01111111	
110	00111111	
101	00011111	
100	00001111	
011	00000111	
010	00000011	
001	00000001	
000	11111111	复位状态和 当二进数值(ARB-OUT)无效时

一个寄存器(ARB-REG)被附加于解码器，这个寄存器在反问于 TF1 的节拍脉冲(TF2)时接收解码器的解码的数值(ARB-DEC)。ARB-DEC 被反馈到屏蔽上(ARB-MASK)并且释放单个的输入信号(ARB-IN)。

在处理器中的作用过程如下：

1. 在一次复位 RESET 以后所有的 ARB-NI(输入信号)通过 ARB-MASK(屏蔽)被释放，因为 ARB-DEC(解码器)把所有的信号都调到“释放”上。

2. 被设置为最优先的 ARB-IN(例如上述表格中信号 7(二进 111)拥有最优先权和信号 0(二进 000)最不优先)被作为二进数值输出。

3. 经过 ARB-DEC 信号被闭锁，一旦所有其它输入更为优先却未被设置。

4. 下面的第 5 和 6 步一直重复，直到信号 0(二进 000)被到达，或在 ARB-MASK 后面没有信号为止。然后 ARB-DEC(参见解码表)释放所有通过 ARB-MASK 经过 ARB-DEC 的信号并且流程在第 2 步处开始。

5. 此时起最优先设置的 ARB-IN 被作为二进数值而输出。

6. 一旦所有其它的输出更为优先却未被设置，信号经过 ARB-DEC 就被锁闭。(继续第 4 步)。

这样就可以同等地处理所有输入信号并且在输入信号(ARB-IN)中的一个的每个节拍循环时把所有输入信号解译成二进值和输出(ARB-OUT)。ARB-REG 可以配一个激活一输入(EN)，它只有在 TF2 时才允许寄存器内容的修改，如果有相应的信号存在。由此，一个二进矢量不再在每个节拍时被输出，而是同通过 EN 和 TF2 产生的释放相关。当后面的转换在一个节拍循环中不能执行处理，而需要多个循环并且然后才接受下一个二进矢量时，输入对于同步就是必要的。

在信号的大多数同等优先时，也许一系列信号通过处理器被认为较为优先是有意义的。这例如在前面描述的用于在 CT 之间继续传送信号的方法时是必要的。为了使一个信号更为优先，ARB-PRIO 的最优先的连接并不遮蔽，即：从屏蔽(ARB-MASK)旁传送过。由此信号被优先处理。

在微控制器的基础上的 CT 的构造

同至此的描述不同，一个 CT 也可以在一个微控制器结构里被执行。

很容易理解：那些基本功能，如触发器控制，查看表 LUT1 和 LUT2，以及 CT 间通信和把 KW 写入 CEL 都直接可以由一个微控制器执行完成。只是一个效率高的 FILMO 的结构有个首先在可达到一性能里就被查觉出的问题。因此，FILMO 的结构被特别地研究了。

FILMO 的结构

FILMO 未被作为单独的存储器。这个普通的程序存储器更多地被扩展了 FILMO-功能。因此一个附加的位(FILMO-位)被分配给每个 KW，它表明相应的 KW

是否已被写入 CEL。如果 FILMO-位被设置，那么相应的 KW 就不被执行。在把一个 KW 写入存储器时 FILMO-位被复位。在 CT 内的所有 KR 都经过一个链接表(FILMO-表)按顺序相互连接，就象它们被触发器或 LOAD<ID>调入那样。一个 KR 一直在 FILMO-表中，直到它完全被执行，然后它被从表中去除。FILMO-表按照 FILMI-方法运行并由此成为 FILMO-存储器的一个直接的替代物。

(为了完善性，可以看出同原来的 FILOM-方法相反在表中一个 KR 不能再次出现。如果一个还在 FILMO-表中的 KR 被调用，那么它的执行必须被延迟，直至它被从 FILMO-表上去除。)

一个 FILMO-存储器位置的结构如下：

FILOM-位	KW

命令

微控制器支持下列对 FILMO 有直接影响的命令：

PUSH　把一个 KW 写入 CEL

PUSHSF　把一个 KW 写入 CEL 并且当 KW 被接受(ACCEPT)时设置 FILMO-位

PUSHRET　把一个 KW 写入 CEL 并且从子程序中返回(RETURN)，当 KW 未被 CEL 接受时(REJECT)。这个命令被使用，当在 KR 中的接着的 KW 取决于这个 KW 的配置时(ACCEPT)；其配置由从 KR 返回而妨碍，直到 PUSHRET 成功(ACCEPT)为止。

PUSHNR　把一个 KW 写入 CEL，只有在前面在 KR 内部未出现 REJECT 时。作用同 PUSHRET 相似，用于使用在配置顺序中对 KW 的依赖性。

垃圾箱

同至此的说明相符合，一个垃圾箱(GC)被用于删除不再需要的 KR。当在存储器中没有足够的空间装载新的 KR 和 ID 必须被删除时；或者当一个 KR 明显地由命令 REMOVE-显示所要删除的 KR 的 ID-被删除时，GC 启动。

为了使 GC-过程尽可能简单，所有的 KR 都通过一个链接的清单相互联系。GC 流览一遍清单并且删除那些不再需要的 KR，通过这些 KR 被其它 KR 覆盖和清单登录被相应地适配来进行删除。此时所有在存储器中留下的 KR 相应地移动，使由于删除了 KR 而产生的存储器空位被填补并且在存储器的最后产生一个更大

的相连的自由空间。

一个 KR 的结构

在下列表中列出了一个 KR 的可能有的基本构造：

jmp START
length
garbage-previous
garbage-next
FILMO-previous
FILMO-NEXT
CACHE-statistic
KR-statistic
START
ret

在 KR 开始时产生一个跳跃经过后面的控制器来启动命令序列。紧接着是双倍链接的用于垃圾箱的清单，在这个清单中所有的 KR 相互连接。

“length”说明了 KR 的长度。这个信息可以根据技术版本状况用于 Block-Move-命令(程序块-移动-命令)，这些命令在 KR 必须在存储器中移动时(垃圾，装载等)被应用。FILMO 在紧接着的双重链接的清单中组成，此时只有含有那些尚未被写入 CEL 的 KW 的 KR 是相互连接的。产生一个关于高速缓冲-状态的统计，它包含例如 KR 的调入次数(每调入一次数值被提高 1)、时效(根据通过 KR 的 GC-过程的数量可测得)等。这个统计可以在一个 KR 必经被从存储器空间中删除时统计分析 GC。对于高速缓冲，通过这些统计产生了很大的优点。这样可以例如根据所使用的高速缓冲一算法，同应用的要求相适应来给微控制器编程，以致

1. 最旧的/最新的 KR
2. 最小的/最大的 KR(参见登录“length”)
3. 最少的/最频繁的调入的 KR 被从高速缓冲中删除，当需要空的存储器时。此时显而易见的其它有用的状态信息可以被存储，此种选择性的缓冲在现

有所熟知的高速缓冲-结构中已不可能了。特别是在高速缓冲中可自由编程的高速缓冲算法根据现有的技术不被支持。

最后有一个 KR 一统计，它包含例如未被配置的(REJECT)或被配置的(ACCEPT)KW 的数量。同时第一个还需必配置的 KW 的地址可以被存储。它的优点在于：在一次 FILMO-过程中可以直接跳到 KW 上面不必经过 KR，这样就大大提高了性能。

最后对于 KR 要说明的是被链接的清单优先地通过登录前一个/后一个-ID 被组成，因为由此绝对的存储器地址可以由 GC 毫无问题地移动。在一个 KR 中应只应用相对跳跃而不是绝对跳跃，来避免在装载 KR 时和在 GC-运行过程中产生问题，因为此时绝对地址会改变。

为了完善性，须提出的是根据已说明的原则在使用微控制器时在执行一个新的 KR 前(基于一个触发器或一个命令，也从另一个 CT 开始)FILMO 也要经过并且在经过 FILMO 之前 CEL 的状态(可变换配置的或不可变换配置的)要被保证。

图

下面描述的图借助于一个执行例子说明了根据所介绍的方法进行的配置数据的管理：

图 1：在查寻表中产生地址的方法

图 2-7：处理命令和状态仪器的功能

图 8：SCRR-ARB 的结构

图 9：LUT1 & LUT2 的结构

图 10：指针算术和 CTR 的结构

图 11：FILMO 的结构

图 12a：CT 的等级配置

图 12b：在 CT 之间的触发器的发送

图 12c,d：触发器矢量的发送方法

图 13：通过多个 IKR 来调入一个 KR

图 14：一个 ROOT-CT 的 LUT1 的结构

图 15：一个 ROOT-CT 的 HOST-控制结构

图 16：说明 LUT 和 ECR 概念

图 17：中间等级层的 CT 以及一个 ROOT-CT 的过程控制

图 18：在一个 2-因次的数组的配置时的死锁扣问题(参见专利说明)

图 19：说明 FILMO-概念

图 20：CT 间通信的基本原理

图 21：根据 GETCTS-方法的 CT 间通信的执行举例

图 22：根据 GETBUS-方法的 CT 间通信的执行举例

图 23：CT 间母线的排线结构

图 24：在 CT-等级里的地址

图 25：垃圾-清单

图 26：FILMO-清单

图 27：在 KR 里的 FILMO 功能

图 28：在执行一个 KR 或 FILMO 前存储状态

图的说明：

图 1 显示了在一个 CT 里的 CTR-地址产生的过程。此时在 LUT1 里的一个详细的二进的触发器矢量(0101)被翻译到一个有效的 KR 或 IKR ID 上。如果没有有效的 ID 存在，那么就产生一个信号“非法触发器”(0102)，此信息说明触发器在 LUT1 里不被识别。这个问题可以作为错误信息被中继传输到上一级 CT 上或被忽略。从“触发器”到“ID”的翻译借助于“REFERENCE”(参考)命令被登录到 LUT1 上。一个有效的 ID(0103)被中继传输到 LUT2 上。在命令内部通过一个运算数说明的 ID(0104)直接到达 LUT2 上。LUT2 把一个详细的 ID 翻译到在 CTR 内部的 KR/IKR 的地址里。如果 KR/IKR 未被存储到 CTR 里(不在高速缓冲里)，那么信息“Miss”(缺少)将被产生(0105)。如果 KR/IKR 的被翻译的地址标有信号“NoAdr”，那么用“No Entry”(0107)表示地址被删除。“Miss”和“No Entry”表示不可以翻译到 CTR-内部的地址上。在此信号的基础上，装载-状态机用一个相应的置于其上的 CT 的 ID 再装载 KR/IKR。

只要存在一个有效的地址，那么这个地址就被传送到地址发生器的指针算法上(0106)。在 LUT1 中一个详细的二进触发器矢量被翻译成一个 ID 或者另一个触发器矢量，此时在这种情况下触发器矢量被输出(0108)。

在图 2 中，在装载一个 KR/IKR 时的过程被说明。首先，要被装载的 KR/IKR 的 ID(0201)被传送到置于其上的 CT 上。然后，在所要求的 ID 的登录处自由指针(FP)的数值被登记到 LUT2 中。FP 指向在 CTR 中的最后一个用于一个 KR/IKR 的登记后面的登记，这是第一个在其上存储了要装载的 KR/IKR 的登记。

状态机等待置于其上的 CT 的数据字句。一旦字句可用，它就被写入由 FP

指出的位置上。FP 被增量，如果 FP 指向 CTR 的末端后的登录，那么在 CTR 里的第一个登录被去除，用以创造空间(0202)；此时 FP 被实现。如果由在其上的 CT 发送的数据字句为“STOP”，那么装载过程被停止(0203)，否则，继续等待一个新的数据字句(0204)。

在图 3a 中，表示了“MASK”一命令。命令的运算数被写入 MASK-寄存器中。MASK-寄存器位于 LUT1 前的触发器信号的输入之前并且标出无效的触发器。

在图 3b 中，命令的运算数通过命令“FRIGGER”被作为触发器矢量发送到其它 CT 上。

在图 3c 中，对于相应的 KR/IKR ID 的触发器翻译被通过命令“REFERENCE”写入 LUT1。

在图 4a 中表示命令“WAIT”。命令的运算数据写入 WAITMASK-寄存器。所有触发器，直到所等待的和由此在 WAITMASK 中释放的，都被忽略。只有在触发器后才返回到程序流。在图 4b 中，“PUSH”-命令被描绘。配置字句被发送到定址的可配置的元素(CEL)处。如果 CEL 不接受配置字句，由于例如 CEL 处在“非配置”状态下，那么配置字句被写入 FILMO 中(0401)。

图 5 指出“REMOVE”-命令的过程。有 2 个调入变量：

1. 第 1 个在 CTR 中的 KR/IKR 被从 CTR 中删除，CTR 的地址被分配给垃圾指针(GP)(0501)。

2. 一个专用的由其 ID 说明的 KR/IKR 被从 CTR 中删除。在 CTR 中的要删除的 KR/IKR 的第 1 个地址被分配给垃圾指针(GP)(0502)。

移动指针被装载了 GP 的数值。就算首个 KR/IKR 应被从 CTR 中删除，GP 和 MP 也要指向 CTR 中的一个“BEGIN<ID>”一命令。相关的 ID 的 LUT2 中被标为无效。MP 被增量，直到下一个在存储器的 KR/IKR 的“BEGIN<ID>”被达到，ODER MP 等于自由指针(FP)，这表示要删除的 KR/IKR 为 CTR 中的最后一个(0504)。

在这种情况下，FP 装上 GP 的数值，这样由要删除的 KR/IKR 占据的存储位置被标为空的；功能“REMOVE”结束(0505)。

否则(“BEGIN<ID>”被达到(0506))，由 MP 指出的数据被复制到由 GP 指出的存储器位置上。MP 和 GP 被增量。这个过程一直进行到 MP 到达 CTR 末端或 FP 的位置为止(0507)。如果在过程中一个在其中有“BEGIN<ID>”的存储器位置被 MP 指出，那么在 LUT2 中的用于相应的 ID 的登录便被 MP 改写(0508)，由此在查看时正确的存储器位置被输出。

图 6 指出 FILMO 的过程图。一个 FILMO 含有 3 个指针：

1. Write P: FIUMO-RAM 的写指针

2. Write P: FILMO-RAM 的读指针

3. Write P: 代表 FILMO-RAM 的“填充状态”和防止过多以及不足的状态指针。

一个一位的寄存器“BeginF”显示实际的读存取是否位于 FILMO-RAM 的开始(TRUE)，即：没有未删除的登录位于读指针和 FILMO-RAM 的开始之间；或读指针位于 FILMO-RAM 的中央(FALSE)，即有用的登录位于读指针和 FILMO-RAM 的开始之间。另外，还有两个寄存器存在用于存储 ReadP 和 FullP 的状态。在出现第一个未被删除的登录时必须保护两个寄存器，因为在一个后面进行的读存取时在这个登录的位置上必须开始读出。另一方面，ReadP 和 FullP 却必须在实际读过程中继续被修改，来获取下面的读地址以及确定 FILMO-RAM 的终端。由于 FILMO 的结构类似于 FIFO-结构-作为所谓的圈存储器-所以存储器的开端和末端都不能借助于一个地址 0 或一个最大地址来被确定。从基本状态中引出两条运行路径：

1. 读路径(0601)

FullP 和 Read P 被确保在寄存器。处理回路开始：

BeginF 为 TRUE。

如果 FullP 等于 0，那么 Read P 和 FullP 被从其寄存器中读回(0602)并且状态机返回基本状态。

此外(0603)被测试是否在 FILMO 中的 RaedP 指向的登录等于“NOP”，即：涉及一个在 FILMO 中央的标为删除的登录。如果不是这样(0604)，那么就会把登录写入可配置的元素中(CEL)。如不行的话(REJECT，0605)，由于 CEL 是不可改变配置的，那么 BeginF 被设置为 FALSE，FullP 减量而 RaedP 增量。状态机跳到处理回路(0606)的开始处。

如果把登录写入 CEL 成功(0607)，或登录为 NOP，那么 BeginF 被测试：BeginF＝TRUE(0608)：在它之前没有未删除的登录。FullP 被增量，ReadP 被保证在分配给的寄存器中来保持 FILMO 的新的开始。FullP 被保证用于保持现实的数据量；ReadP 被增量。

BeginF＝FALSE(0609)：FullP 被增量和在 FILMO-RAM 中的实际登录被用 NOP 改写；即：登录被删除，ReadP 被增量。

在两种情况下状态机都跳到处理回路的开始处。

2. 写路径(0610)

通过检验最大数值上的 FullP 来测试 FILMO-RAM 是否满。

如果已满(0611)，则跳到读路径里来获取空间。

否则，数据字句被写入 FILMO-RAM 中并且 WriteP 和 FullP 增量。

图 7 显示在主状态机中的过程。基本状态(IDLE)被离开，一旦

1. 出现一个位于其上的 CT 的 REMOVE-Kommando(移动一命令)(0701)；移动命令被执行，状态机返回 IDLE。

2. 在 CT 之间出现一个用于产生触发器的触发信号(0702)；触发器被输出。状态机跳入“STOP”一命令，而后返回 IDLE。

3. 出现一个用于执行 KR/IKR<ID>触发器信号(0703)：程序指针(PP)被由 LUT2 产生的地址装载。如果地址无效，即：在存在用于要装载的 KR/IKR 的登录，则 KR/IKR 被装载(0704)和 PP 重新设置。

执行回路开始：

PP 被增量(在首次回路运行时 BEGIN<ID>命令便由此被跳过)，其它触发器被禁止出现，RECONFIG 被锁闭。命令被执行和跳到执行回路的开始(0707)。

命令“STOP”被特别执行(0705)。触发器和 RECONFIG 被重新释放并且状态机跳到 IDLE。

命令“EXECUTE”也被特别执行(0706)。在 EXECUTE<ID>中给出的 ID 被写入 ID-REG。PP 被重新装载并且由 ID 给出的 KR/IKR 被执行(0708)。

在 CT 复位后基本配置被装入 CTR 中并直接跳入基本配置的执行中(0709)。

图 8 显示一个 SCRR-ARB 的结构。要判断的信号经过 DataIn 到达一个屏蔽(0801)，它根据所知的表连接信号的相关联部分以及锁闭。根据现有技术一个普通的优先处理器(0802)由连接的信号的数量判断一个信号并且把它的二进矢量 (Bi-naryOut)同一个有效的/无效的-标识(ValidOut)一起(也根据现有技术)提供为 SCRR-ARB 的输出。

这个信号根据所知的表被翻译编码(0803)并且传递到用于同步节拍的寄存器上(0804)。经过这个寄存器 DataIn 屏蔽被打开。此时寄存器或者由一个节拍或者由一个询问下一个有效二进制矢量的下个信号(Euable EN)控制。在复位时或当标识(ValidOut)显示无效时，寄存器被接通，以致 DataIn 屏蔽连接所有的信号。

屏蔽的构造在0805中被说明，在0806中屏蔽再一次被说明，此时根据SCRR-原则，信号DataIn 0.. DataInl为同等优先，而DataIn m...DataIn n较为优先。

在图9中LUT-结构被描绘。判断的触发器的二进制矢量(BinaryIn)被传到LUT1(0901)的地址输入上。LUT1 把这个二进制矢量或者翻译到有效的触发器里来把它中继传输到另一个CT上，或者翻译到一个有效的ID里、两个都经过0910被输出。0911显示出是涉及一个触发器还是一个ID。如果经过命令“REFERNCE”没有详细的二进矢量的翻译被登录到 LUT1 中，那么-借助于一个位登录或一个比较器以一定的方式(例如“VOID”)-产生信号“非法触发器”0914。一个触发顺经过0912被引导到外部CT上，ID经过乘法器(0902)被继续处理。0902或者接通说明一个有效ID的LUT1的数据输出，或者接通CT的ID-寄存器(0903)到LUT2的地址输入上(0904)。0904具有一个高速缓冲-相类似的结构，即：0902的数值输出的低值部分(0906)被接到0904的地址输入上，而较高值的部分(0907)被接通到 0904 的数据输入上。属于 0907 的数据输出经过一个比较器(0905)同0907比较。这种方法的优点是0904不必显示用于翻译所有ID的深度，而是可以分析出更小的深度。类似于普通的高速缓冲，只有ID的一个部分被翻译，同时在LUT2里借助于0907可以确定所选择的ID是否符合由LUT1说明的ID。根据现有技术，这符合Cache/TAG-方法。

一个乘法器 0908 被分配给 0904 的第二个数据输入，它根据操作把自由指针(FP，操作LOAD)，垃圾指针(GP，操作REMOVE)或一个无效-标识/记号(NoAdr，操作 REMOVE)送到 LUT2 上用于存储。这两种指针指向 CTR 中的存储器位置，“NoAdr”表明不存在适合的 ID 的登录，登录被删除。这是通过数据在记号“NoAdr”上经过比较器 0909 被比较来被在数据输出上确定的。下列被继续传导到状态机上：

-通过“ValidIn”出现一个二进制矢量(比较图8)。

-说明在翻译到 LUT1 里时是涉及到一个触发器还是涉及到一个 ID(0911，“Trigger/ID Out”)。触发器被经过0912传送到其它CT上。ID在自有的CT里被处理并继续传送到LUT2上。

-0905的结果，它说明相应的ID是否被存储在0904中(“Hit/Miss Out”)。

-0909的结果，它说明相应的ID是否指向CTR中的一个有效的地址(“No Entry Out”)。

由 0904 产生的地址被继续传送到 CTR 上(“CTR Address Out”)。

LUT1 经过命令“REFERENCE”被用详细的二进制矢量的翻译装载到一个触发器或 ID 上。命令的这算数被经过母线 0913 传导到 LUT1 上。ID-寄存器(0909)经过相同的线母被装载。

图 10 显示垃圾指针(PG)、程序指针(PP)、移动指针(MP)和自由指针(FP)的指针算法。每个指针由一个可分开控制装载的上/下-计数器组成。每个计数器都可以-只要有必要-用其它的计数器的数值来装载；同用 LUT2 的输出一样(1007)。

通过比较器来确定是否

1. PP 等于 MP
2. MP 等于 FP
3. FP 等于在 CTR 中的最大位置

这些结果被用于控制状态机。

指针中的一个经过乘法器(1001)被传导到 CTR 的地址输入处。数据经过乘法器(1002)或者从置于其上的 CT(1005)处或者从一个寄存器里(1003)到达 CTR 上。经过一个乘法器(1004)或者是上一级的 CT 的数据或者是 CTR 的数据被继续传送到状态机和 FILMO(1006)处。此时在出现一个 REMOVE-命令时命令直接由上级 CT 经过 1004 传送到状态机上，而不然的话，命令被从 CTR 传到状态机上。寄存器 1003 用于把命令存储和反馈到 CTR 输入上，这些命令在垃圾箱运行时被从一个地址移动到另一个地址。

一个 FILMO 的结构被在图 11 中被说明。数据从 CTR(1101)到达 FILMO 并且或者经过乘法器(1102)被写入 FILMO-RAM(1103)或者经过乘法器(1104)被发送到可配置的元素(1116)上。如果数据在 1103 中被删除，那么一个“NOP”-信号就经过 1102 被写往 1103。“NOP”-信号经过数据输出口的比较器(1105)被识别并且防止写入可配置的元素。经过乘法器 1106，不是写指针 WriteP(1107)就是读指针(1108)被导往 1103 的地址输入上。在寄存器 1109 中，读指针被保证，以便可以复位(参见图 6)。

1103 的填充状态计数器满(1110)被根据图 6 存储在寄存器 1111 中用于复位。两个比较器测试 1103 是空的(1112)还是满的(1113)。经过乘法器 1115 来选择是状态机(1101)的控制信号还是 FILMO 的控制信号被发送到 1116 上。

图 12a 指出 CT 的等级结构。所有的 CT 从 ROOT-CT 中(1201)和从属于它的

ECT(1204)中取得数据。对于在一个组件中的每个执行层面都存在一个或数个CT。每个 CT 都负责管理其层面和下层的 CT。树的所有树枝都一样深是没有必要的。例如，用于控制一个组件的外围设备(1202)的层面要比用于控制工作元件(1203)的层面少。数据传递如树状进行。每个 CT 作为用于置于其下的 CT 的高速缓冲而运行。

图 12b 显示在 CT 之间的触发器流。当数据流按树状移动时，触发器流未被固定。每个 CT 都可以发送一个触发器给其它的 CT。一般而言，触发器交换只由页(1203)朝 ROOT-CT 的方向(1201)进行。而有时传递也可以朝相反的方向移动。

在图 12C 中一个触发器矢量 Broadcast(播送)被说明，同时 1205 向所有 CT 发送一个触发器矢量。

图 12d 显示一个较高级的触发器矢量，它被 1206 发送到置于其上的 CT 上。1208 把一个直接编址的(ADDRESSED)-触发器矢量传输到一个特定的 CT 上，此 CT 并不直接同 1207 相连。

在图 13 中两个独立的 IKRn 和 m 请求一个共同的在置于其上的 CT 中高速缓冲的 KRx。这表明这个 KR 由整个分支缓冲并且也在一个相邻分支中(1301)可使用一个共同的 CT。

图 14 表明一个相对于图 9 修改过的 LUT-系统，这个系统在 ROOT-CT 和 CT 中借助于等级层面被使用。其同至此所描述的 CT 的根本区别在于：必须由 CT 来管理 ID-和/或触发器矢量，而不是单个触发器信号。此时一个手摇一信号(RDY)被分配给每个矢量以显示矢量的有效性，这个显示被传到一个处理器(1401)。经过乘法器(1402，1403)，不是触发器矢量中的一个(1404)就是 ID-矢量之一(1405)被选择。触发器矢量直接到达 LUT1 的地址输出上(1406)，这些矢量否则就按图 9 被接线，ID-寄存器(1407)也按图 9 接线。同图 9 相反，乘法器 1408 有三个输入口(比较 0902)。同时乘法器除了由状态机控制外，另外也由处理器 1404 控制。ID-矢量经过另外的输入口被直接经过 1403 中继传输到 LUT2 上。母线 1409 即用于此。(原则上，在 CT 处根据图 9ID 也可以按照乘法器(1408)直接转换到 LUT2 上。然后 ID 可以在未翻译的情况下直接由 CEL 发送到 LUT2)。“Trigger/ID Out”被按图 9 产生。一个按照图 9 被中继传输到“Valid Out”上的“Valid In”信号不存在。取而代之的是根据判断由 1401 产生一个“Valid Trigger Out”用于触发器矢量和一个“Valid ID Out”用于 ID-矢量，未规定

状态机如何进行处理。母线 1409 被经过 1410 引入一个只存在于 ROOT-CT 里的在图 15 中说明的另一个单位。

一个 ROOT-CT 除普通 CT-功能外，还需要一个通向外部配置存储器(ECR)的界面，以及必要的地址发生器和用于管理存取到 ECR 上的单位。

一个普通的 CT 在 LUT1 中把详细的触发器矢量翻译到一个 ID 上并在 LUT2 里把 ID 翻译到 CTR 中的一个存储器位置上(参见图 16a)。在存取到 ECR 上时一个 ROOT-CT 把一个在 ECR 内部的 ID 翻译到 ECR 里的地址上，在这个地址上由 ID 指出的 KR/IKR 开始。为此在 ECR 中一个其大小相当于 ID 上可能的数量的存储范围被确定(如果一个 ID 例如为 10-位宽，得 $2^{10}=1024$ 个可能的 ID，也就是说在 ECR 中预留了 1024 个登录)。在下列举例中这个存储范围位于 ECR 的下端末尾处并且 LUT-ERC 被列举出用以强调同 LUT2 的相似性。此时根据在 LUT1 中已熟知的 CT 把一个触发器翻译到一个 ID 上(1601)。为了更好地理解，图 16b 说明了到 ECR 上的存取。

在图 15 中一个 ID 经过图 14 上的 1410 到达乘法器 1501。ID 经过 1501 被写入可装载的计数器 1502 中。1502 的输出口经过乘法器 1503 导入 ECR 的地址母线(1504)上。ID 的翻译经过数据母线 1505 到达经过 1501 上的乘法器/除法器(1506)的一个存储器地址，这个乘法器把存储器地址装到 1502 上。接着，相应的 KR/IKR 的数据字句被经过主机 LOAD-ECR(参见图 17)从 ECR 中读出并且写入 CTR，同时 1502 根据每次读过程被提高，直到命令“STOP”被读取。

经过界面 1507，上级的 HOST 经过 1503/1506 把 KR/IKR 写入 ECR。此时经过状态机(CT)被判断是 HOST 还是 ROOT-CT 有到 ECT 上的存取。

在组件复位后，一个基本配置(BOOT-KR)必须被装载。为此引进一个固定的指向 BOOT-KR 的第一个存储位置的存储地址。作为 BOOT-ADR，存储位置 0h 被推荐。只要 ID 在 1 时开始，否则 2^{ID} 或任意一个其它的存储位置被使用，在执行例子中 2^{ID} 被使用。

ROOT-CT 进行查寻用以在 BOOT-ADR 位置上装载 BOOT-KR，一旦一个 BOOT-KR 被装载。ROOT-CT 把数据写往 1502，用以从那里装载 BOOT-KR 直到出现一个“STOP”命令。

在 ROOT-CT 内部的一个监控单位接收 HOST 同组件的同步。这如下进行：

地址小 2^{ID} 由 1508 监控，即：在由 HOST 存取到这些地址上时一个信号(ACC-ID)被发送到状态机(CT)上。同样地，BOOT-ADR 被经过 1509 监控并且把一个信号

ACC-BOOT 发送到状态机(CT)上。状态机(CT)如下反应：

-如果 HOST 写入 BOOT-ADR，这就导致 BOOT-KR 的装载。

-如果 HOST 把数据字句 0(1512) 写入 BOOT-ADR，那么就被经过比较器 1510 确定并且导致组件的停止。

-如果 HOST 写入地址较小的 2^{ID}，那么地址被装载到 REMOVE-寄存器(1511)上。因为地址符合 ID(参见 ECR-LUT)，所以被修改的 KR/IKR 的 ID 在 1511 中。命令 REMOVE<ID>被发送到所有 CT 上用于立即执行(1513)。接着 CT 从其 CTR 以及 CUT2 中删除相应的 ID 的 KR/IKR。在接着存取 KR/IKR 时 CT 必须强制性地从 ECR 中装载新的 KR/IKR。

图 17 指出在从 ECR 中装载 KR/IKR 时在 ROOT-CT 里的过程。如果一个 ID 不在内部的 CTR 中(比较图 1，1701)，那么 ID 就被写入计数器 1502 中(1703)。把 1502 中的地址存取到 ECR 上提供了 KR/IKR 的基本地址。这个地址被写入 1502 中(1704)。根据图 2 的 LOAD 进行了(1702)。此时数据不是从上级 CT 中而是从 ECR 中被读取(1705)并且不仅被写入自有的 CTR 中，而且被发送到下属的 CT 上(1706)。在较中等的等级层面的 CT 中触发器的翻译运行类似于图 1，触发器矢量和 ID-矢量按图 14 被处理是个例外。KR/IKR 被按图 2 装载，数据字未被写入自有的 CTR(0210)而是同时被发送到下级的 CT 上为例外。

图 19 说明了 FILMO 原理。FILMO(1901)在写和读的存取时总是从开始运行到最后(1902)。如果登录被从 FLIMO 的开始被写入和删除(1903)，那么读指针移到第一个未被删除的登录上(1904)。如果登录被从 FILMO 的中间写入(1905)，那么读指针保持不变(1906)，登录使用“NOP”作标记(1907)。如果数据被写入 FILMO，那么它被挂在最后，在最后的登录后面(1909)，读指针(1910)保持不变。

当然，一个 CT 可以由只有一个存储器、LUT1、LUT2 和 CTR 包围着构成。而为此的控制却较浪费。这时 CT 的构造同已把 LUT2 和 CTR 集合入 ECR 的 ROOT-CT 相类似。不必描述这个 CT 来理解这种方法。

如果一个 CT 被作为高速缓冲系统用于数据，那么触发器被引进用于把数据写入 CTR。此时数据被 CEL 写入 CTR。在此必要的修改是很普通的，FILMO 可以完全不要。

在数据缓冲时出现数据密度的问题，它可以通过使用一种根据 DE42 21 278A1 的方法来在单个等级层面中标识数据和其有效性来解决。如果数据被要求用于执行一个读-修改-写-循环(RMW-循环)，那么这些数据在所有的等级层面

上借助于一个附加的登录在 CTR/ECR 中被标识为“无效”(INVATID)。为此，使用这些数据的 KR/IKR 的单一 ID 被登记到登录中。这些数据不可以由带其它 ID 的 KR/IKR 使用，直到使用这些数据的 KR/IKR 写回了这些数据(比较现有技术的写-回-方法)并且删除了它们的 ID。

图 20 指出了一个执行举例：

在图 20a 中 CT2007 要求置于其上的 CT 的数据，这个置于其上的 CT 要求 ROOT-CT2004 的数据；随着数据要求，要求的 KR/IKR 的(2001)ID 被传输。数据(2002)被发送到 2007 上。所有其它的后面的存取都被拒绝(2003)。

在图 20b 中数据被写回(2005)，其它的后来的存取重新被接受(2006)。

在图 20c 中数据被由含有数据的中等等级的一个 CT 要求，并且被发送到 2007 上。用于锁闭数据的 ID 被发送到在等级中的所有 CT 上。(2001)在图 20d 中写回数据(Write-Back)时数据被写到等级中的所有 CT 上并且删除 ID。

图 21 显示一个 INITIATOR CT(2101)经过几个中间-CT(2104，2105，2106)同一个 TARGET CT(2102)的通信，以及根据 GETCTS/LOOSECTS-方法进行的没有中间层面的同一个 TARGET CT(2103)的直接通信。

2101 建立了同 2103 的联系。在成功的建立后，2101 和 2103 处获得一个 GRANT 作为建立的确认。此后 2101 经过 2104、2105、2106 建立起同 2102 的联系。只有 2102 被到达时，同 2102 的联系才被确认(GRANT)。如果不能建立联系，由于母线中的一根被占线，那么一个 REJECT 被送到 2101 上并且 2101 中断该过程。这表明，同 2103 的连接也被中断而且一个 REJECT 被发送到 2103 上。

如果 2102 用 GRANT 确认了连接，那么 2101 把一个 GO-命令发送到 2103 和 2102 上，用以同时向 2103 和 2102 确认成功的母线建立和同步。通过这个记录数据或命令可同步地和不锁扣地传输，因为经过 GO 确保了所有 TARGET 正确地接收了命令。

图 22 显示按照 GETBUS/LOOSEBUS-方法的 CT 间通信的过程。在按照图 21 的方法中各个上级的 CT 拥有控制和优先任务时，在此控制就由 CT 间母线(2201)来承担了。

通过 INITIATOR-CT(2101)要求其局部的 CT 间母线(2202)来建立同 2103 的连接。当母线是畅通的(ACCEPT)时要求被确认，或者当母线占线时(REJECT)，要求被拒绝。然后它把地址从 2102 发送到母线上。根据定址图表母线系统控制识别出地址位于局部母线地址之外并且经过上级 CT2104 建立起同其局部母线的

连接(2203)。由于 2102 的地址位于其地址范围内，所以同 2102 的局部的母线的连接被经过 2106 建立(2204)。由于 2101 现在是所有用于数据通信所必要的数据总线的唯一的数据主线，所以就确保了通信顺利地无锁闭地运行，因为用于其它所有 CT 的通信渠道被锁闭了。2102 和 2103 也不能使用数据总线，因为这些母线在其 TARGET-角色中只能接收命令并且只能应 INITIATOR(2101)的要求自己发送数据。一旦通信结束，数据总线便由 2101 的一个信号去除连接。如果在数据总线建立时 2101 到达一个被使用的母线，那么一个 REJECT 便被发送到 2101 上，而 2101 重新去除数据母线系统的连接并且试图重新在后面的时间里建立。如果多个 CT 同时要求同一个母线，那么上级的 CT 更为优先(2205)。由此避免一个已运行经过了多个层面的先进得多的母线结构被一个还很局部的母线结构所中断。

通过一个扩展的记录可以在 REJECT 时只拆除那些被较为优先的母线结构所需要的母线。这可以大大提高性能，因为不是所有的母线都能在后面的时间里重新建立的。

用于根据图 22 的方法的 CT 间母线的结构被在图 23 中说明了。CT2301-2304 被经过其界面(2308-2311)同上级的 CT2305 一起(界面 2307) 连接到 CT 间母线 2312 上。连通到 CT 间母线经过一个 Ronnd-Robin-处理器发生，它同 2308-2311 同样优先而比 2307 更为优先，它控制一个乘法器用于联接母线(2306)。一个处理控制信号(例如：建立/拆除、接受、拒绝.....)的状态机被分配给这个处理器。

图 24 显示一个一维的 CT-树的地址示意图的结构。长方形表示一个 CT。CT 的地址就登录在那里。一标出那些不被处理的不相关的地址位，相关的地址位被用二进制 0 或 1 表示，*表示每个任意的地址位。很容易理解，通过这个示意图的投影也可以在多维树上被应用，此时被说明的地址各表示轴中的一个；也就是说：每根轴有一个相应的单独的地址系统。

图 24a 显示 CT0001 的定址。此时说明相对于地址 1。通过计算-1+1=00(“相对运动”＋“在实际界面上的 INITIATOR-CT 的地址”)可以计算出在同一个局部数据母线上被接通的 CT0000、图 24b 中 CT0010 调入相对地址+10。10＋0＝10 的计算(“相对移动“＋”在实际层面上的 INITIATOR-CT 的地址”产生传输 1，因为最低的局部数据总线的地址范围正好为一位。由此下一个较高的数据总线被选择。它的地址计算随着 10+10＝100(“相对移动”＋“在实际层面上的

INITIATOR-CT 的地址”)重新产生一个传输，因为它的 2 位的地址范围正好比最低的数据总线的地址范围大 1。在下一个层面上在计算 10＋010＝0100 时不出现传输，以致第三位(从左起)选择地址在带下一个较低层面的路径 1**上，第 2 位(从左起)把一下最低的层面的路径 10*定为地址并且最终最后一位选择 TARGET-CT。

图 24c 在正方向显示经过 2 个层面的已知方法，而图 24d 在带负的超程的负方向上的经过 3 个层面的方法。

图 25 显示一个 2-维的 CT 树的结构，在最低的层面上(2502)有 2-维设置的 CT(2501)。维的地址在各 CT 中用 x/y 表示，置于 2502 上的是下一个较高的层面(2504)。其 CT(2503)各控制层面 2502 的 4 个 CT 的一个组，在 2504 上的 CT 的地址空间宽了一位，*作为层面 2502 的同 2504 上的 CT 的选择无关的地址位。ROOT-CT2505 位于 2504 之上。2505 的地址又大一位，*的作用是相同的。

图 26 显示在微控制器-执行时的垃圾箱的链接。此时所有的 KR 一起经过页眉登录(垃圾-前一个/垃圾下一个)被相互链接。在垃圾箱通过列表时，KR 的年龄通过把登录提高(+1)被记录用于高速缓冲-统计(2602)。垃圾箱注意 KR-统计(2601)的登录，它显示 KR 是否还挂在 FILMO-列表中。在这种情况下 KR 不允许被从 GC 处删除，因为它还含有未被配置的 KW。作为另一种选择，这个测试也可以经过登录 FILMO-下一个和 FILMO-前一个。在图 27 中说明了 FILMO-列表的链接。

此时链接可以完全不同于在垃圾-清单中的链接(图 26)。KR 经过 FILMO-前一个和 FILMO-下一个被链接。登录 KR-统计(2701)指向在各自的 KR 中的第一个还未被配置的 KW。一个 FILMO-过程如此形成，以致 KR 在第一个 KD 中被启动。在执行以后，未被执行的 KW 被写往 2701。如果 KR 被完全执行，那么 KR 被从链接的 FILMO-清单中删除，却留在存储器中。此后，经过 FILMO-清单跳到也被处理的下一个 KR 去。

图 28 说明了在微控制器控制时 KR 的结构。开始时有一个跳跃命令，它跳到 KR 的页眉(2801)后面。FILMO-位(2802)被分配给每个 KW，一个 1(2803)显示 KW 被 CEL 接受(ACCEPT)和在下一次通过时不再执行。一个 0(2804)表示拒绝，KW 必须在下一次通过时重新被执行。选择的 KR-统计(2701)指向第一个用 0 作标记的 KW。如果 PUSHRET(2805)收到一个拒绝，那么 KR 的数据处理就在此中断并且在下一次通过时不是在第一个 KW 时就是在指向 2701 的位置上重新被装上。

否则，KR 在其终端在 2806 处顺序地离开。

图 29 显示用于防止 CEL 的状态信息经过 FILMO 或启动 KR 的接通。状态信息从 CEL(2901)到达一个寄存器处(2902)。在通过 FILMO 或启动一个 KR 前 CF 把一个释放信号(2903)发送到 2902 上。接着，状态信息被接收并且中断传输到 CT 上(2904)。2904 保持不变直到 2903 的下一次发送。

概念定义

接收信号 这个信号表明有地址的可配置元件处在可配置状态中并且采用发送来的配置代码。

信息组——指令(或指令组——移动)指令将大部分数据(1 个信息组)移入存储器中或存储器和外部设备之间。为此需要给出被移动数据的原始地址，数据的目标地址和数据信息组的长度。

中继 把一条信息发放到多接收机中

数据接收 继续处理可配置元件结果的单元

数据发送 数据作为可配置操作数的单元

数据代码 数据代码由任意一个长的二进制组构成。这个二进制组表示机器的加工单元。在数据代码中可以对处理程序和功能模块的指令以及纯数据进行编码。

闭锁 由于相互封闭而不能进行数据处理的状态

DFP 依照专利 DE4416881 的数据流处理程序

DPGA 动态可配置 FPGA。技术状态

元素 所有具有闭锁单元的总称。元素可以作为块进入电子模块。元素指：

—所有类型的可配置元件

—组件

—随机存取存储器程序段

—逻辑电路

—计算器

—寄存器

—乘法器

—电路板的输入/输出线

事件 一个事件可以通过硬件元素在任何一种方式中得到评估。评估的结果是释放限定反应。事件有下列几种类型：

—计算机的循环节拍

—内部或外部中断信号

—功能模块中其它元素的起动信号

—数据流和/或指令流与数值的对比

—输入/输出事件

—指针的流出、溢流和重新设置等

—对比的评估

FIFO 技术状态下首次输入、首次输出存储器

FILMO 从线性数据中可以读出的变化的 FIFO。在存储器始端不存在对读取指针的限制。

FPGA 技术状态下的可编程序逻辑块。

在 F—PLUREG 调节器中设置可配置元件的功能。同样可以设置一次使用和睡眠模式。调节器功能由 PLU 说明。

碎片 把存储器分到许多小的又没有用处的存储器区域。

无用数据收集器 管理存储器的单元，防止碎片。

H—电平 逻辑 1 电平，从属于应用技术。

HOST 上级计算机的功能模块或标准组件

无效循环 在这种循环中状态机不能进行数据处理。这是状态机的一种基本状态。

起动—高速缓存器单元—总线 位于平面高速缓存器单元和高层高速缓存器单元(或单元组)之间的总线系统

起动器 在起动—高速缓存器单元—总线上存取数据的高速缓存器单元

指针 指示地址或数据代码的指针

可配置元素(KE)可配置元素表示一个逻辑块单元，这个单元可以通过配置代码调节特殊功能。可配置元素包括随机存取存储器元件、乘法器、逻辑算术单元和寄存器以及内外部交联描述器等元件的所有类型。

可配置元件(CEL) 参见逻辑元素

配置 功能设置和逻辑单元、(FPGA)元件或 CEL 之间的交联(试比较循环配置)

配置数据 随机产生的配置代码

配置路径(KR)许多配置代码组合成一条规则系统。

配置存储器 配置存储器包括一条或多余配置代码。

配置代码(KW)配置代码由随机产生的一个长的二进制组构成。这个二进制组对可配置元素进行有效调整，因而产生一个功能单元。

逻辑载入 可配置元件的配置和循环配置单元。通过和装载任务匹配的微控制器调节

逻辑元件 用于 DFP、FPGA 和 DPGA 的可配置元件，可以通过配置来完成简单的逻辑或算术任务。

一览表指令 技术状态下转换数据的方法

LUT1 一览表指令将同步信号转换成识别码并确定同步信号是否和有效的识别码相匹配。

LUT2 一览表指令将识别码转换成局部存储器相应的配置路径地址并确定局部存储器是否存在配置路径。

L一电平 逻辑 0 电平，依赖于应用技术

屏蔽 对大部分有效信号进行说明的二进制位组合

优先权 确定顺序

重新配置 可配置元件的重新配置状态

重新配置一一同步信号 将可配置元件设置成重新配置状况。

拒绝 这个信号表明可配置元件不在可配置状态中并且不采用接收的配置代码

消除一<识别码> 1.用配置路径内的指令去除以识别码为基准的配置路径。

2.用分离界面上的上置交流器指令或下置交流器的符号交换来消除以识别的为基础的配置路径。

重设 在定义的基本状态下重新设置功能模块或整个计算机系统

根部一高速缓存器单元具有在外部配置存储器上直接存取功能的最高级高速缓存器单元

循环一Robin一工人 工人循环工作并且最终信号总是和最低优先权相匹配

状态机 参见状态机(zustandsmachine)

同步信号 从可配置元素或计算器中产机的状态信号，这个信号对其它可配置元素或计算器的数据处理产生同步性。同步信号可能在导回可配置元素或计算器时滞后(存储)。

目标 高速缓存器是起动一高速缓存器一总线上的一个存取对象

Trigger 同步信号的同义词

重新配置 在任意剩余的可配置元件继续保持自身功能时对任意一组少配置元

件进行重新配置(比较“配置”)。

联接清单 技术状态下描针上结合的数据结构

元件 可配置元素的同义词

状态机 采用不同状态的逻辑过程。状态之间的过渡依赖于不同的输入参数。机器控制复杂的功能并且符合技术状态。

说 明 书 附 图

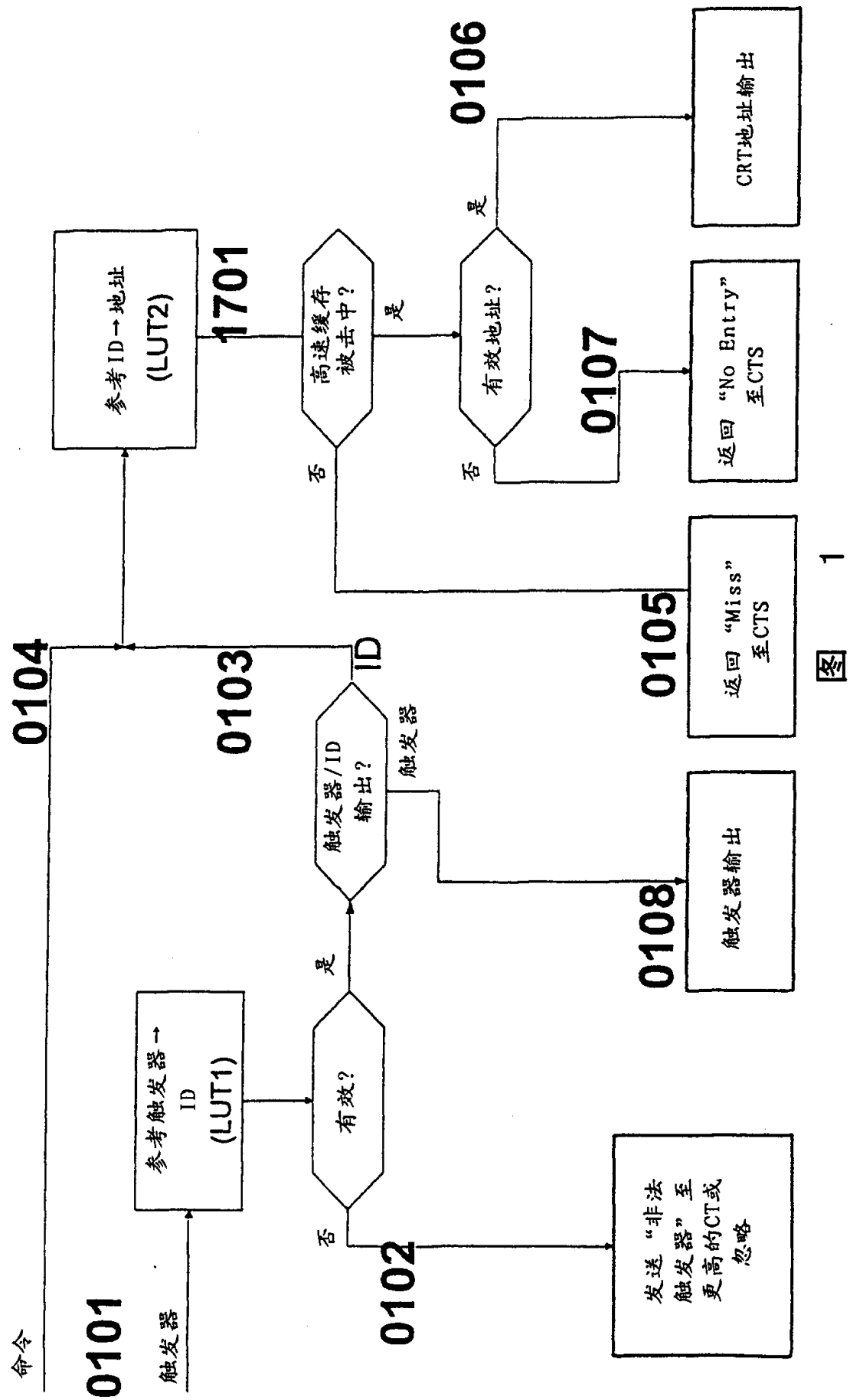

命令
触发器
0101
参考触发器→ID (LUT1)
有效?
是
否
0102
发送“非法触发器”至更高的CT或忽略
触发器/ID输出?
触发器
ID
0103
0104
参考ID→地址 (LUT2)
1701
高速缓存被击中?
是
否
有效地址?
是
否
0106
CRT地址输出
0107
返回“No Entry”至CTS
0105
返回“Miss”至CTS
0108
触发器输出

图 1

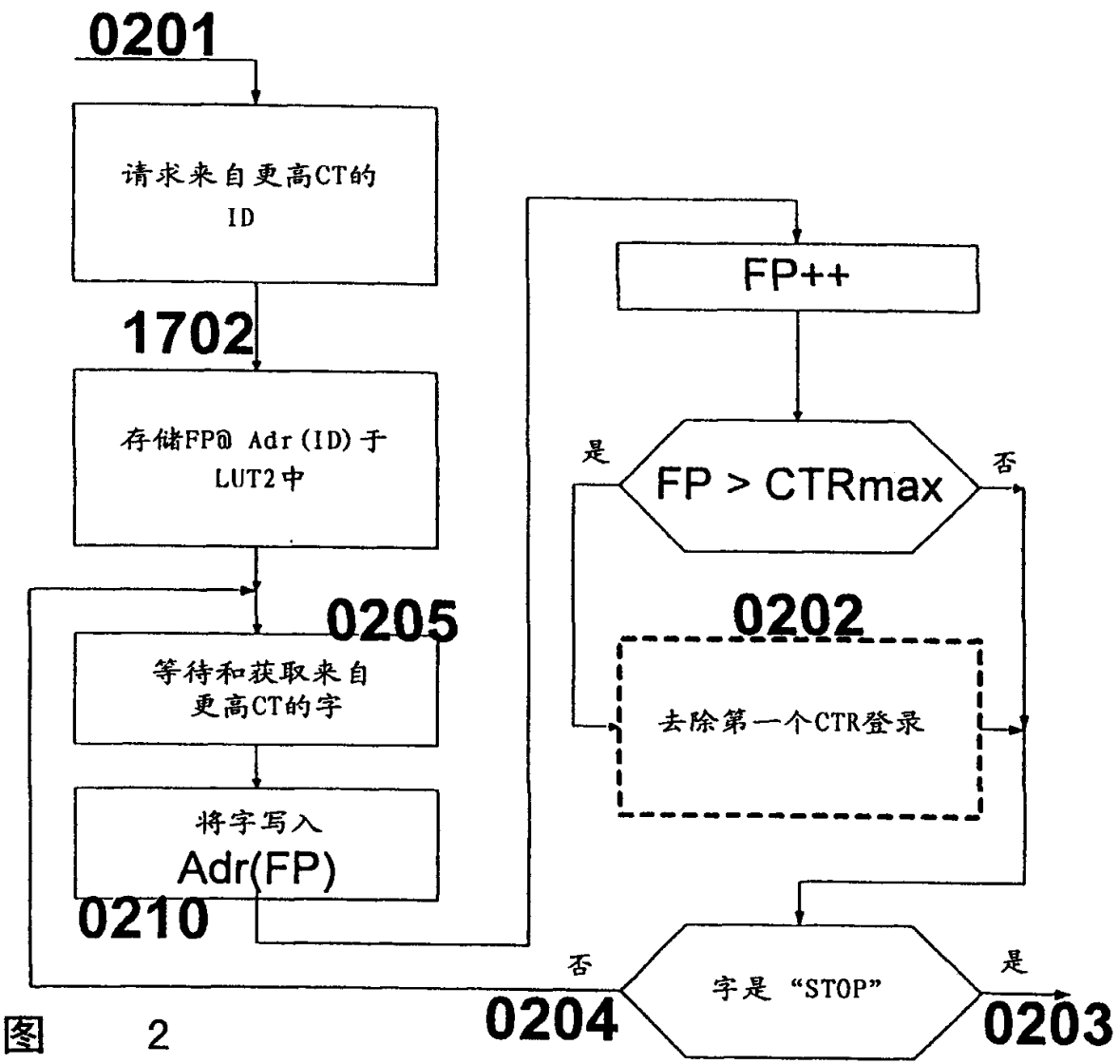

0201
请求来自更高CT的ID
1702
存储FP@ Adr(ID)于LUT2中
0205
等待和获取来自更高CT的字
将字写入Adr(FP)
0210
FP++
是
FP > CTRmax
否
0202
去除第一个CTR登录
否
字是 "STOP"
是
0204
0203

图 2

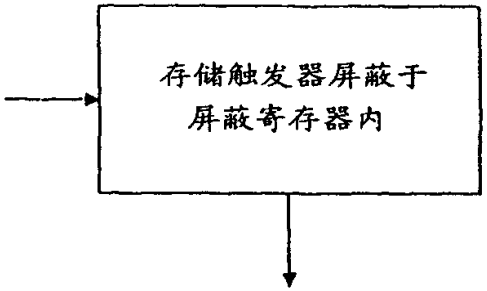


图 3a

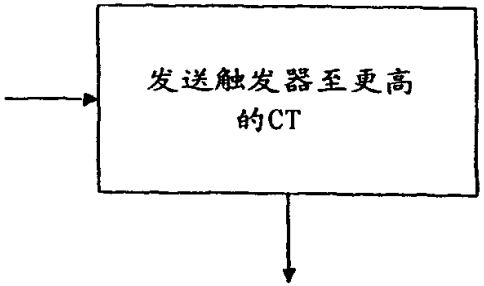


图 3b

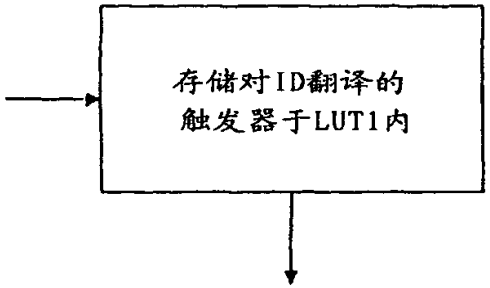


图 3c

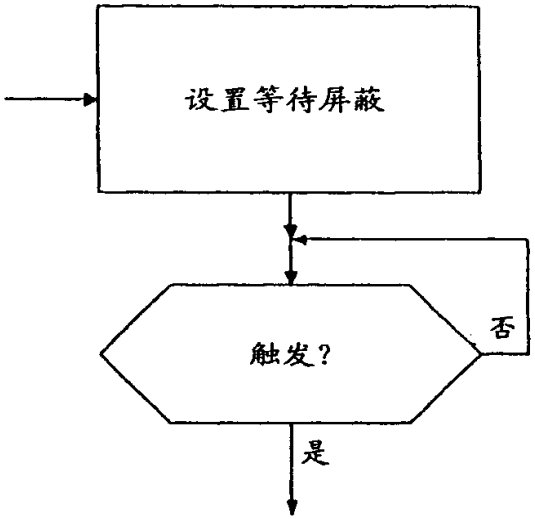


图 4a

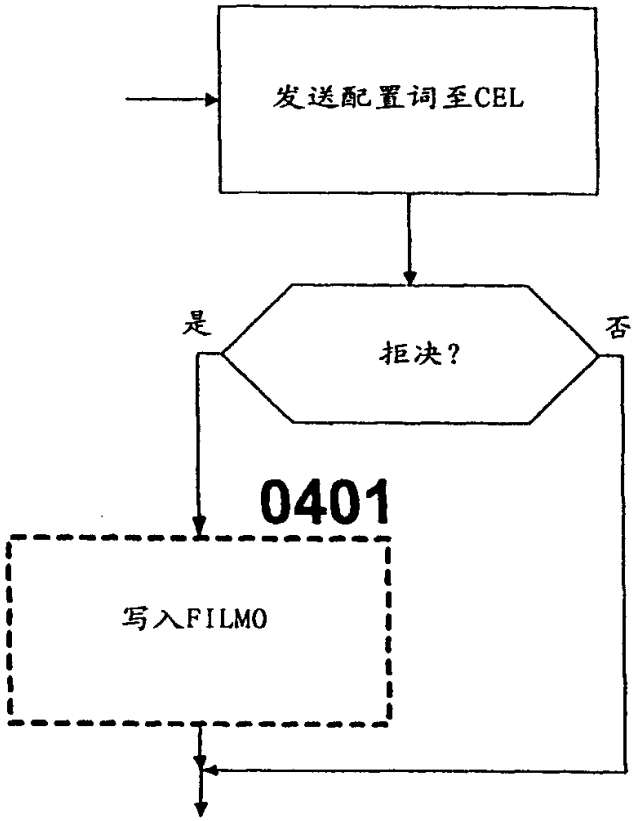


图 4b

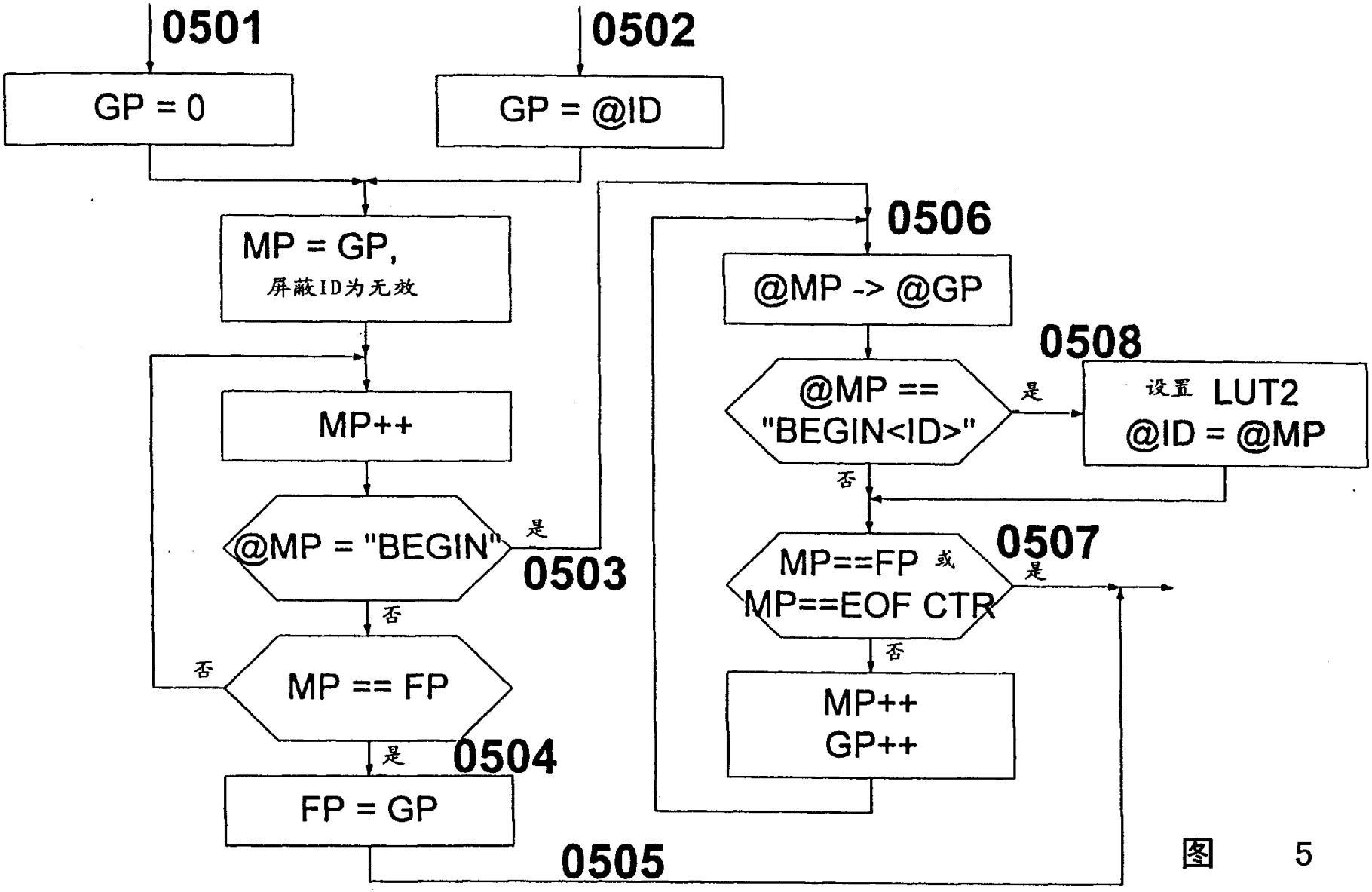

0501
GP = 0
0502
GP = @ID
MP = GP,
屏蔽ID为无效
MP++
@MP = "BEGIN"
是
0503
否
MP == FP
否
是
0504
FP = GP
0505
0506
@MP -> @GP
0508
@MP ==
"BEGIN<ID>"
是
设置 LUT2
@ID = @MP
否
MP==FP 或
MP==EOF CTR
0507
是
否
MP++
GP++

图 5

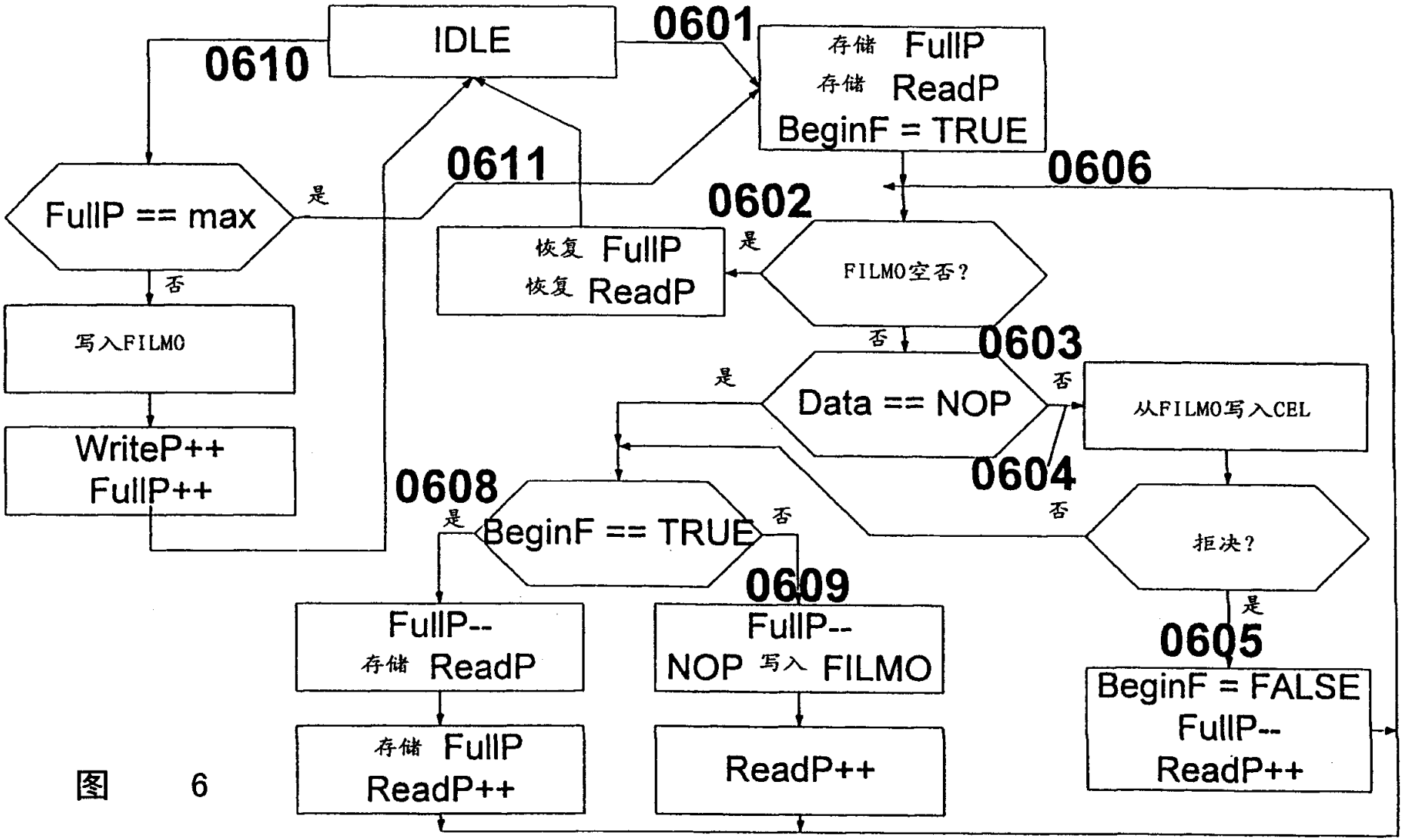

IDLE
0601
存储 FullP
存储 ReadP
BeginF = TRUE
0606
0610
0611
FullP == max
是
否
写入FILMO
WriteP++
FullP++
0602
恢复 FullP
恢复 ReadP
是
FILMO空否?
否
0603
是
Data == NOP
否
从FILMO写入CEL
0604
否
拒决?
是
0605
BeginF = FALSE
FullP--
ReadP++
0608
是
BeginF == TRUE
否
0609
FullP--
存储 ReadP
存储 FullP
ReadP++
FullP--
NOP 写入 FILMO
ReadP++

图 6

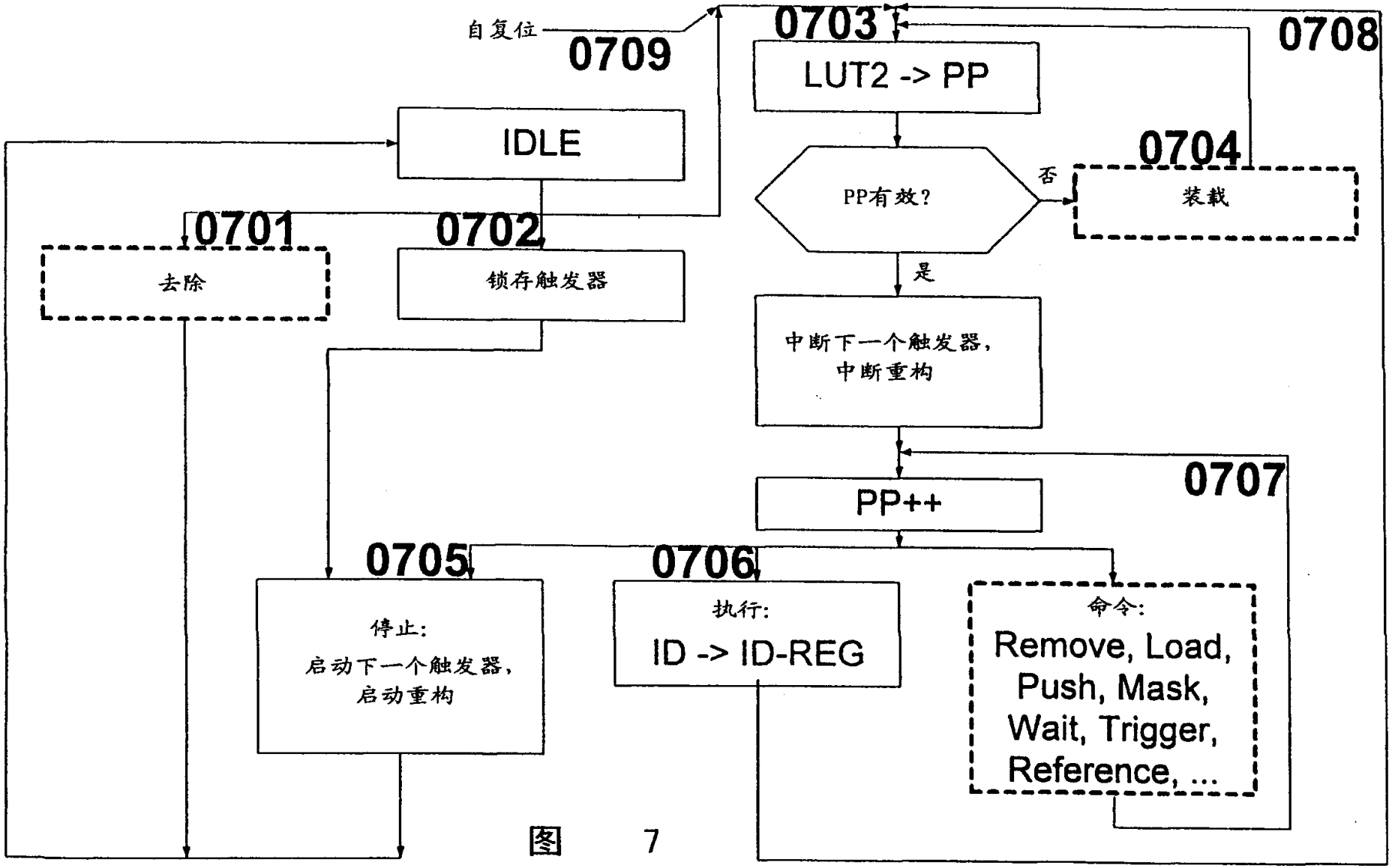

自复位
0709
0703
0708
LUT2 -> PP
IDLE
0704
PP有效?
否
装载
0701
0702
去除
锁存触发器
是
中断下一个触发器，
中断重构
0707
PP++
0705
0706
停止:
启动下一个触发器，
启动重构
执行:
ID -> ID-REG
命令:
Remove, Load,
Push, Mask,
Wait, Trigger,
Reference, ...

图 7

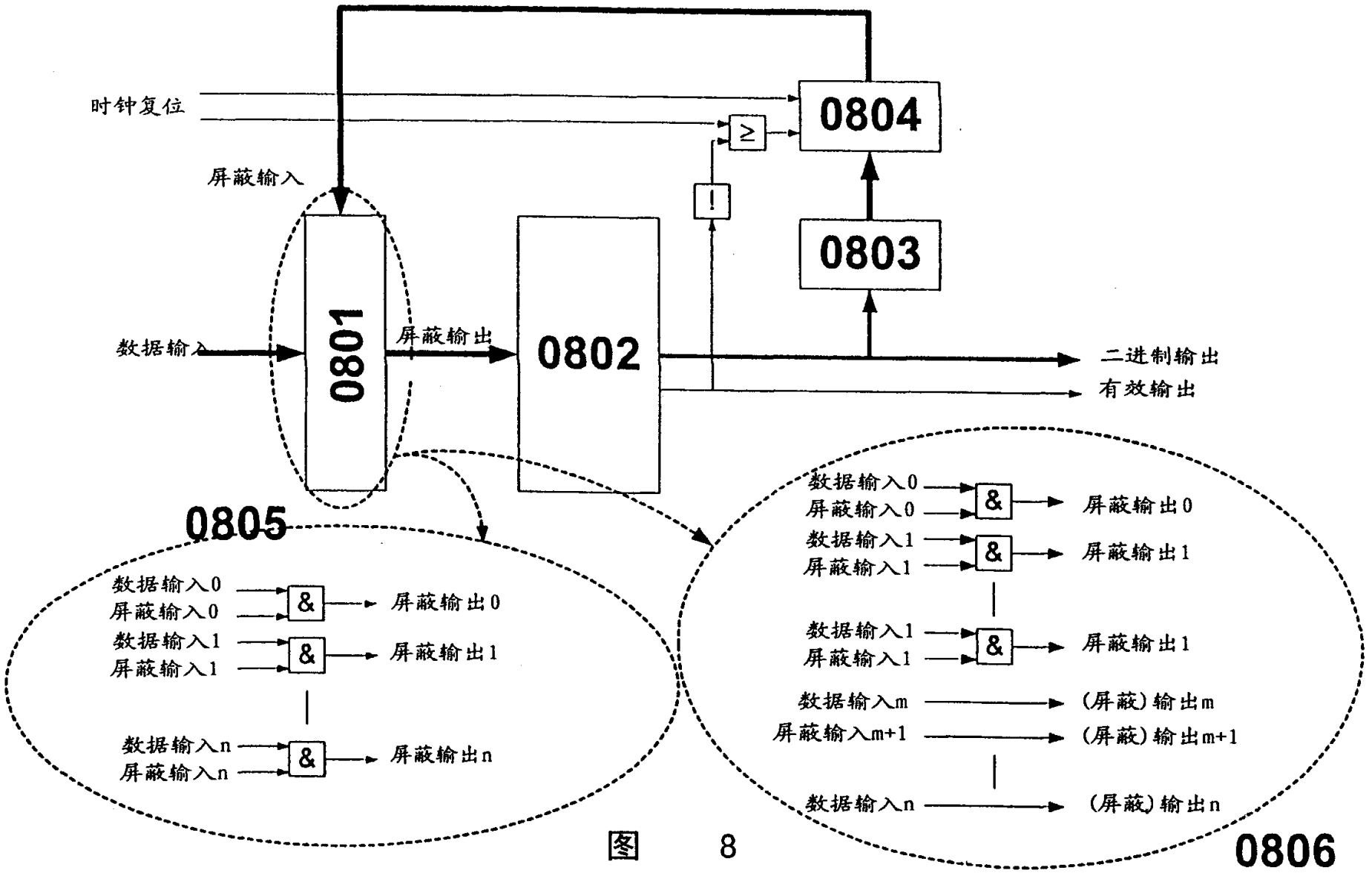

时钟复位
屏蔽输入
数据输入
0801
屏蔽输出
0802
0803
0804
≥
!
二进制输出
有效输出
0805
数据输入0
屏蔽输入0
数据输入1
屏蔽输入1
数据输入n
屏蔽输入n
&
屏蔽输出0
屏蔽输出1
屏蔽输出n
0806
数据输入0
屏蔽输入0
数据输入1
屏蔽输入1
数据输入1
屏蔽输入1
数据输入m
屏蔽输入m+1
数据输入n
屏蔽输出0
屏蔽输出1
屏蔽输出1
(屏蔽)输出m
(屏蔽)输出m+1
(屏蔽)输出n

图 8

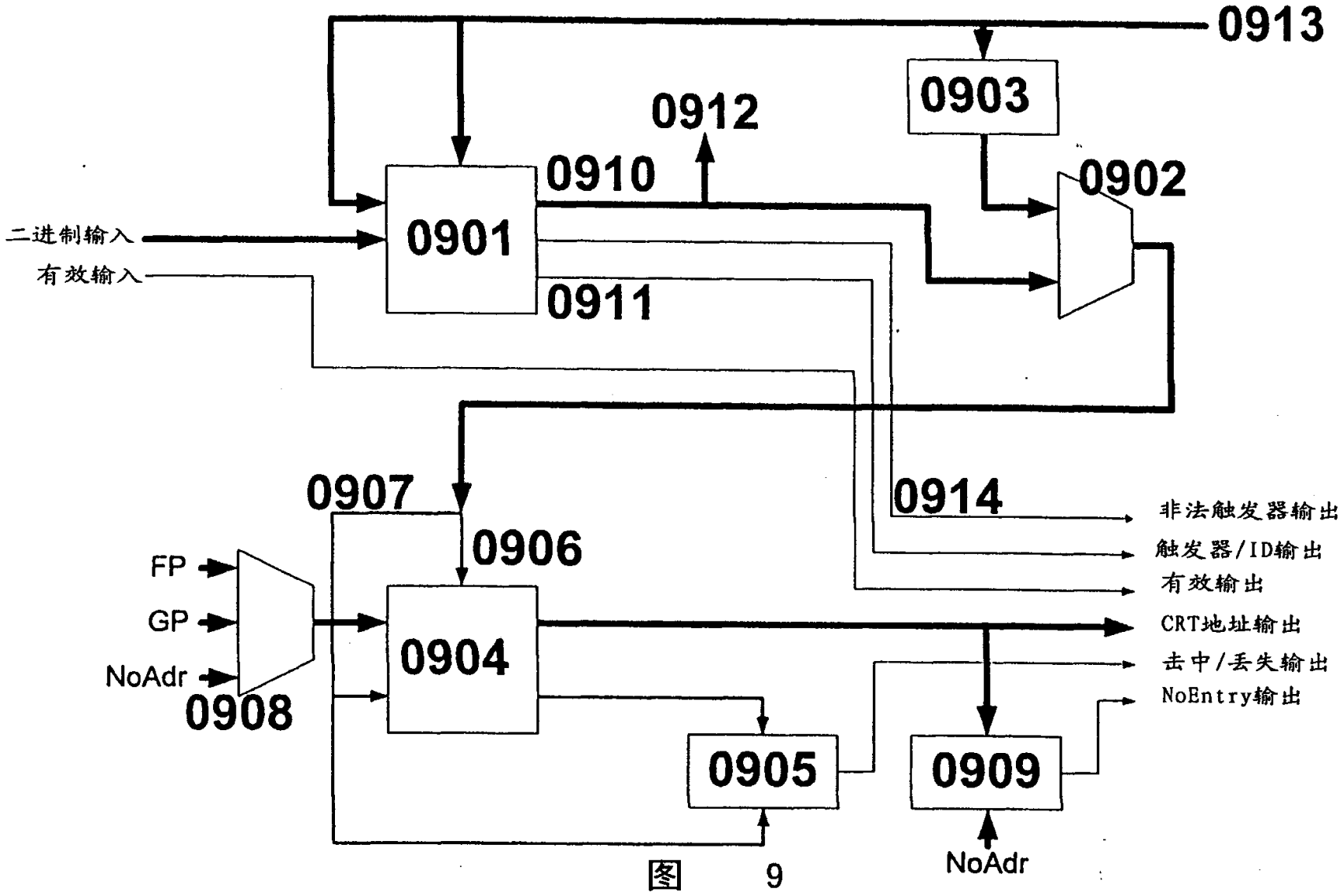

0913
0903
0912
0910
0902
0901
二进制输入
有效输入
0911
0907
0914
0906
非法触发器输出
触发器/ID输出
有效输出
FP
GP
NoAdr
0908
0904
CRT地址输出
击中/丢失输出
NoEntry输出
0905
0909
NoAdr

图 9

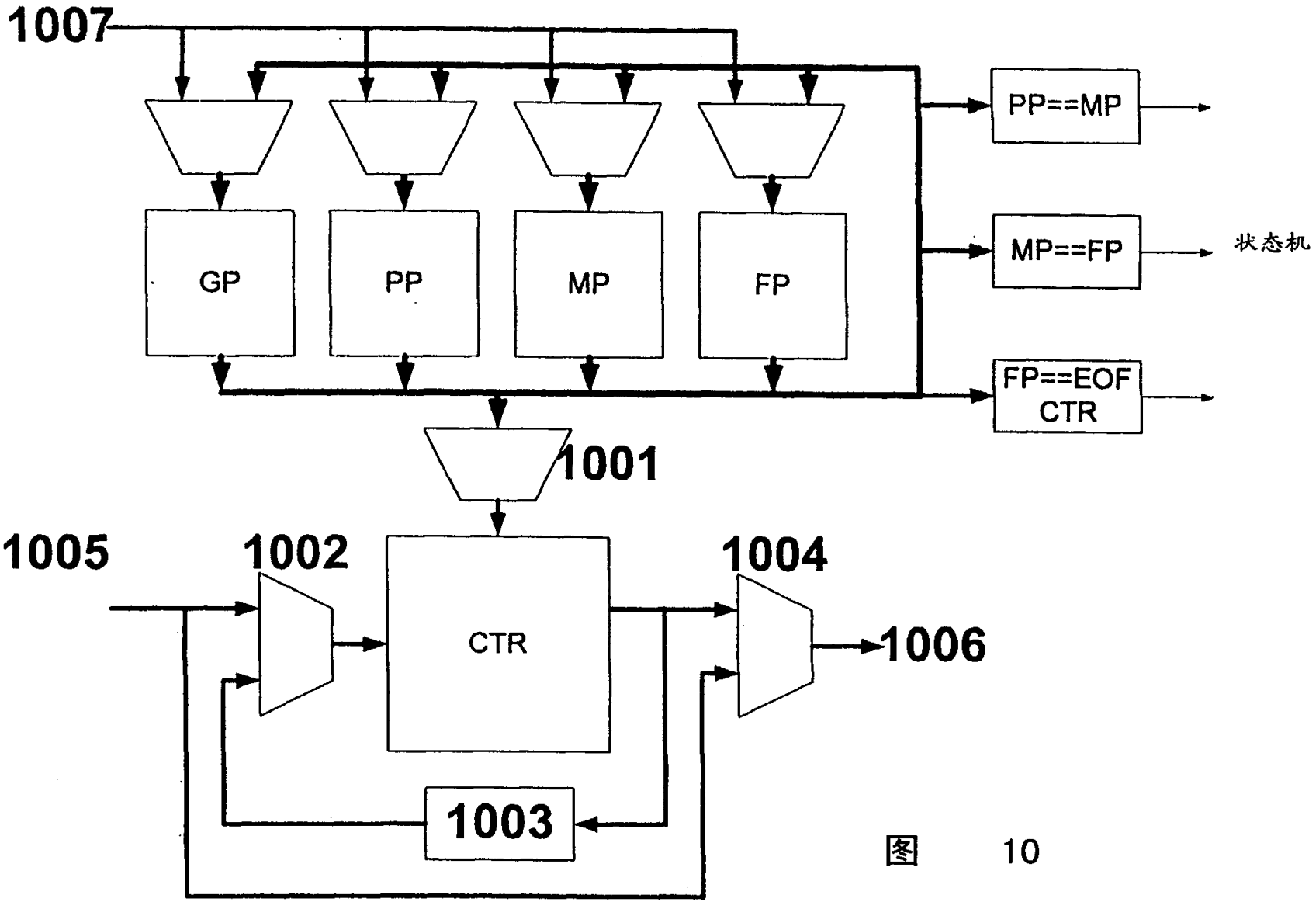

1007
GP
PP
MP
FP
PP==MP
MP==FP
FP==EOF
CTR
状态机
1001
1005
1002
CTR
1004
1006
1003

图 10

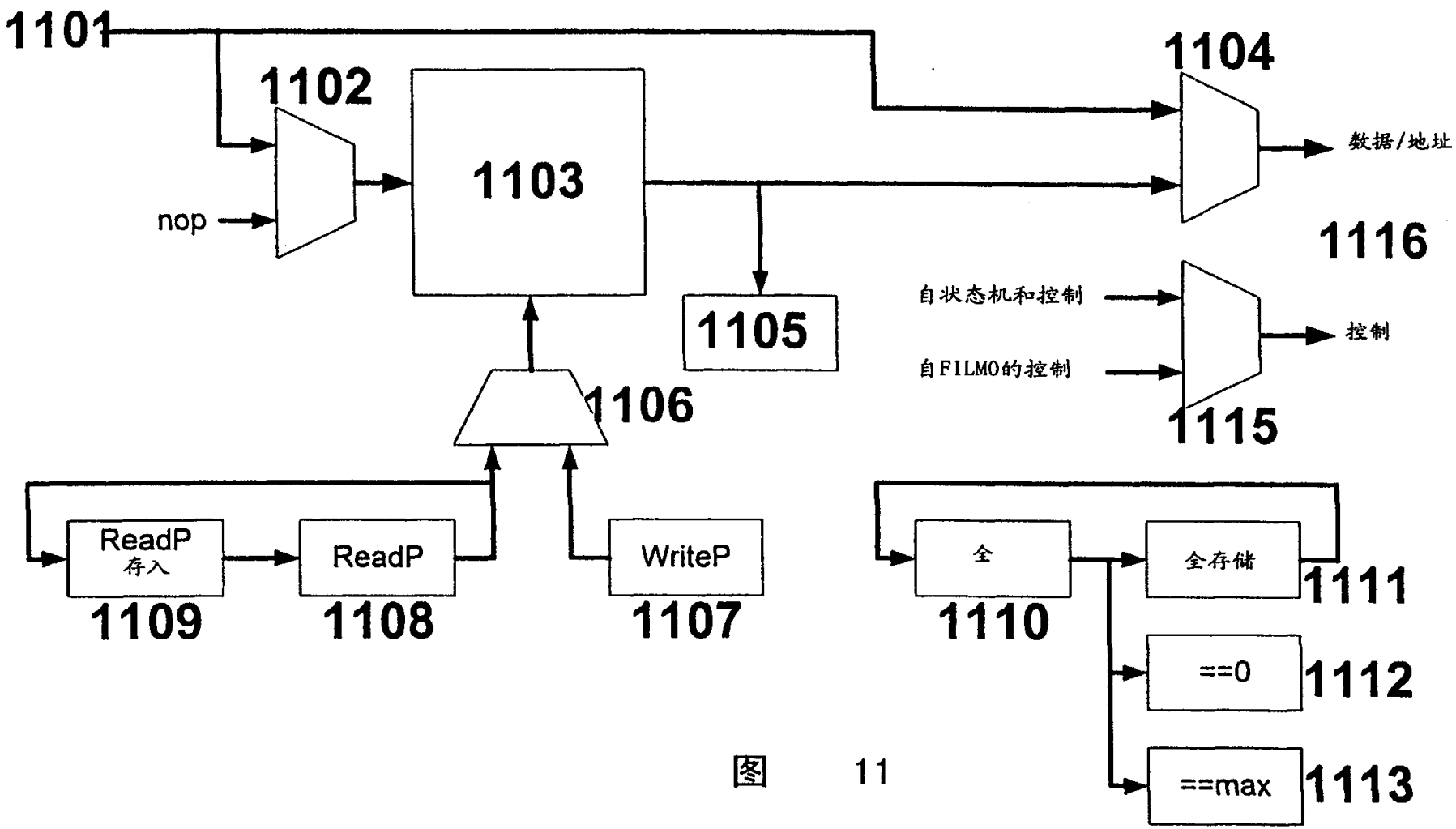

1101
1102
1103
1104
nop
数据/地址
1116
1105
自状态机和控制
控制
自FILMO的控制
1115
1106
ReadP
存入
ReadP
WriteP
全
全存储
1111
1109
1108
1107
1110
==0
1112
==max
1113

图 11

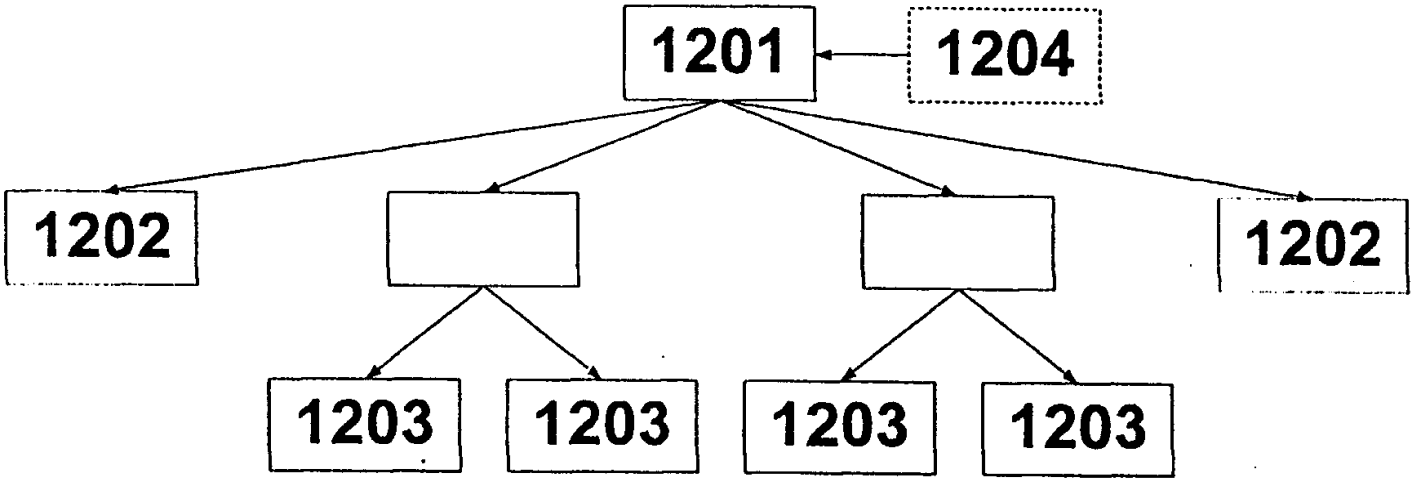

1201
1204
1202
1202
1203
1203
1203
1203

图 12a

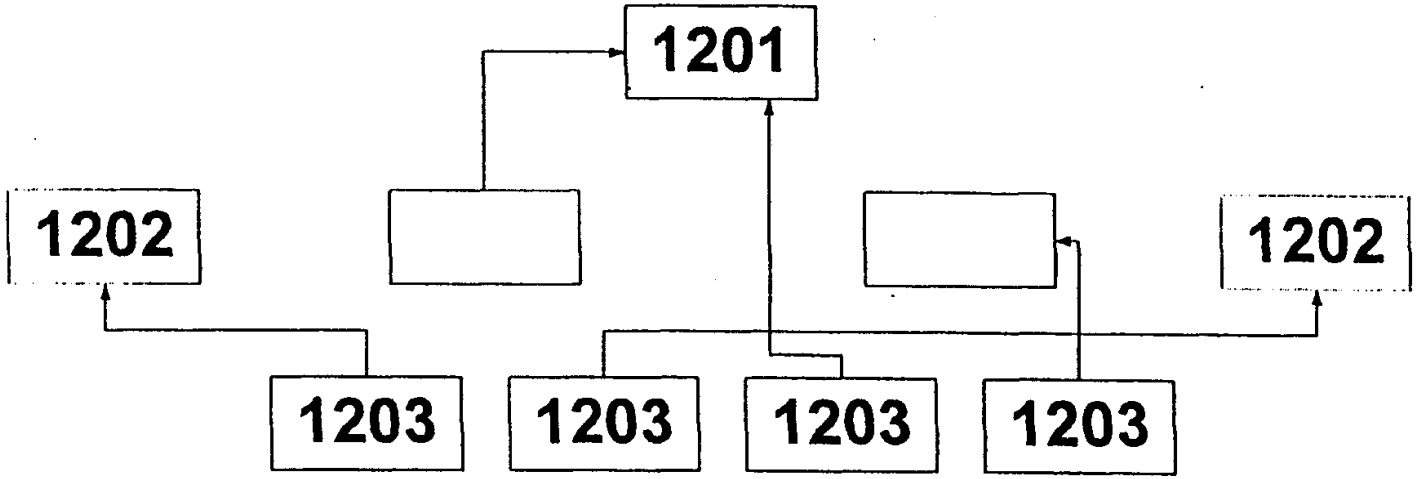

1201
1202
1202
1203
1203
1203
1203

图 12b

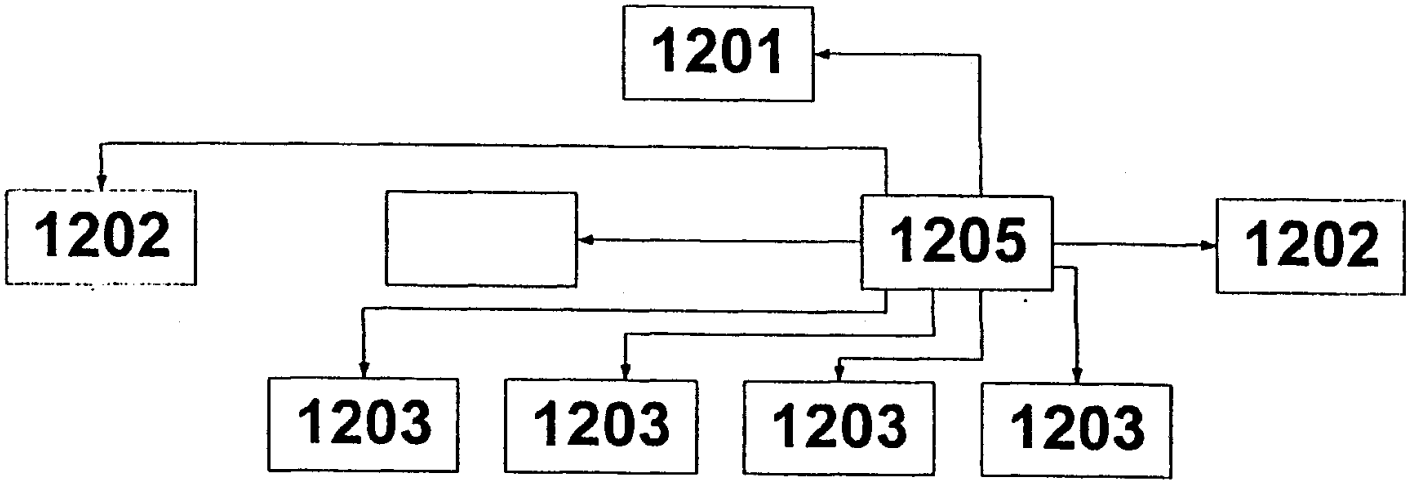

1201
1202
1205
1202
1203
1203
1203
1203

图 12c

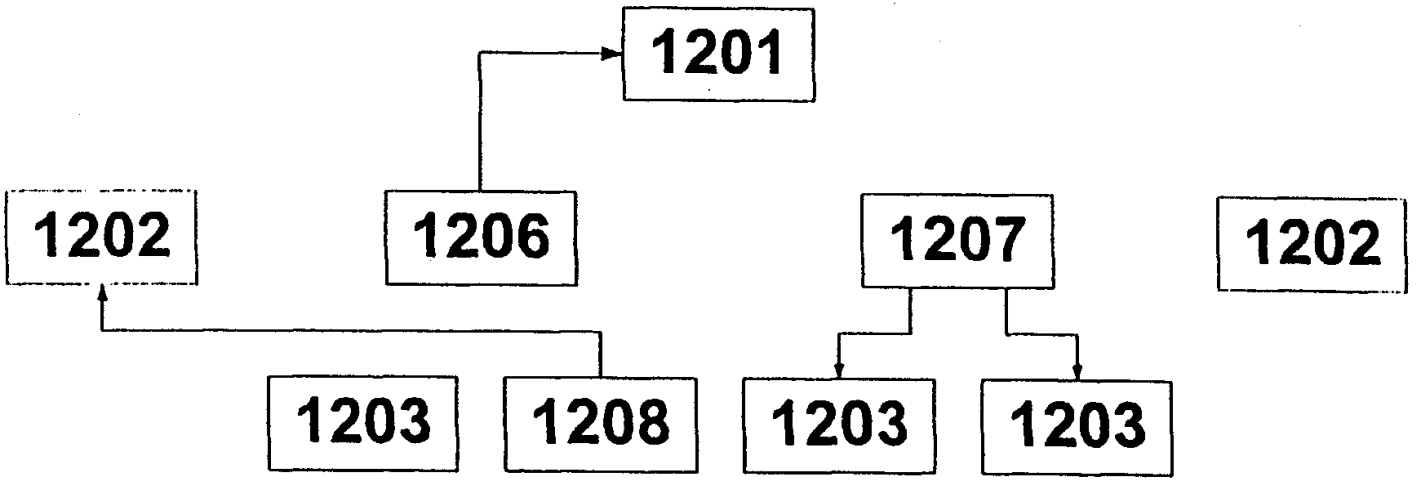

1201
1202
1206
1207
1202
1203
1208
1203
1203

图 12d

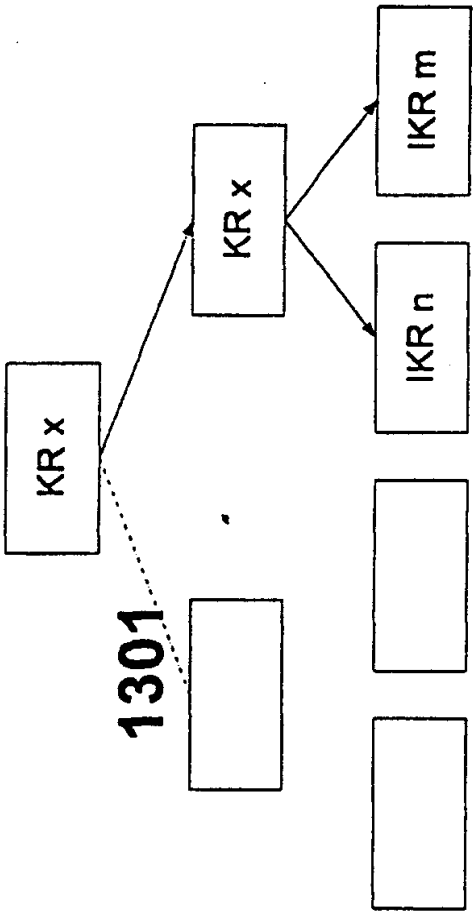


図 13

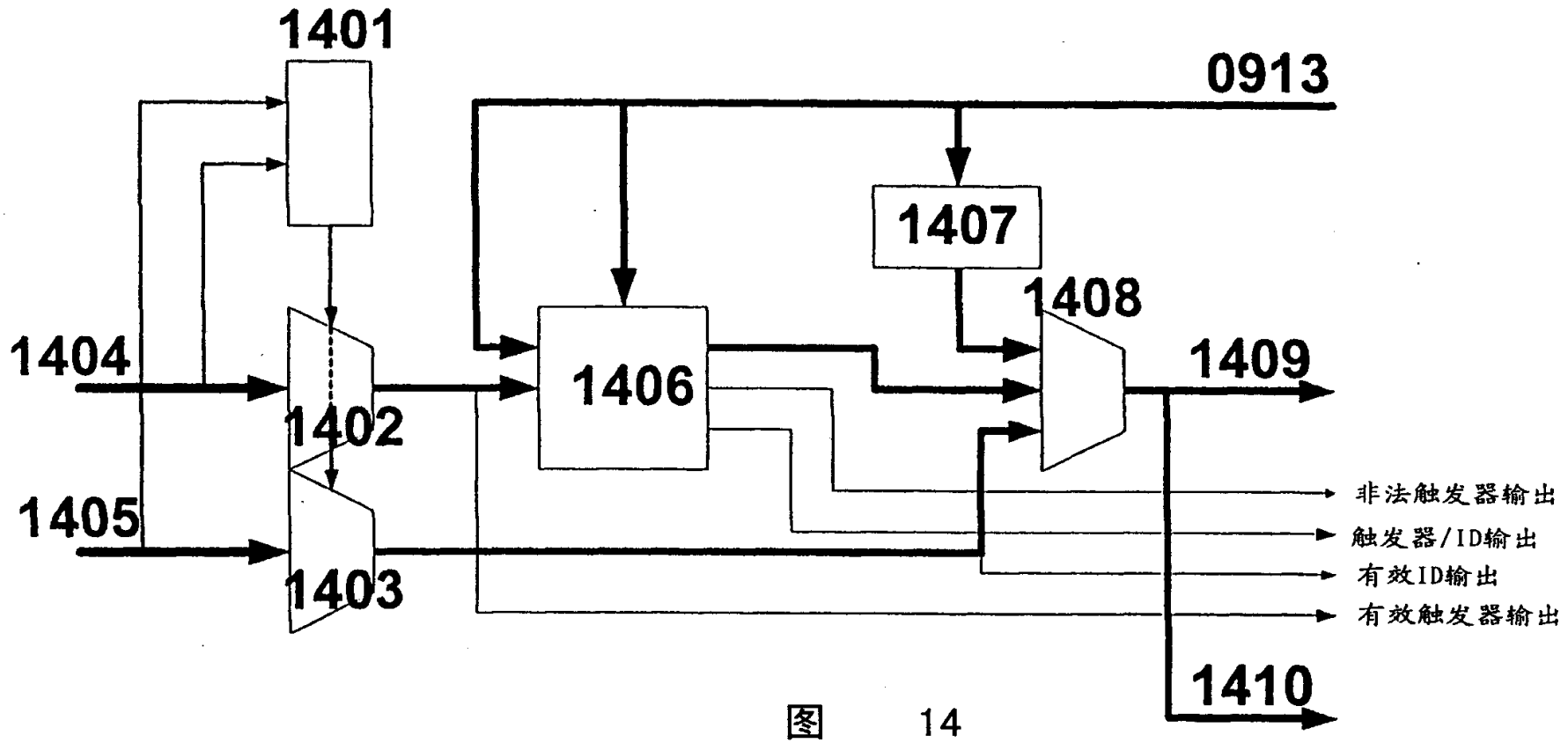

1401
0913
1407
1408
1404
1406
1409
1402
1405
1403
非法触发器输出
触发器/ID输出
有效ID输出
有效触发器输出
1410

图 14

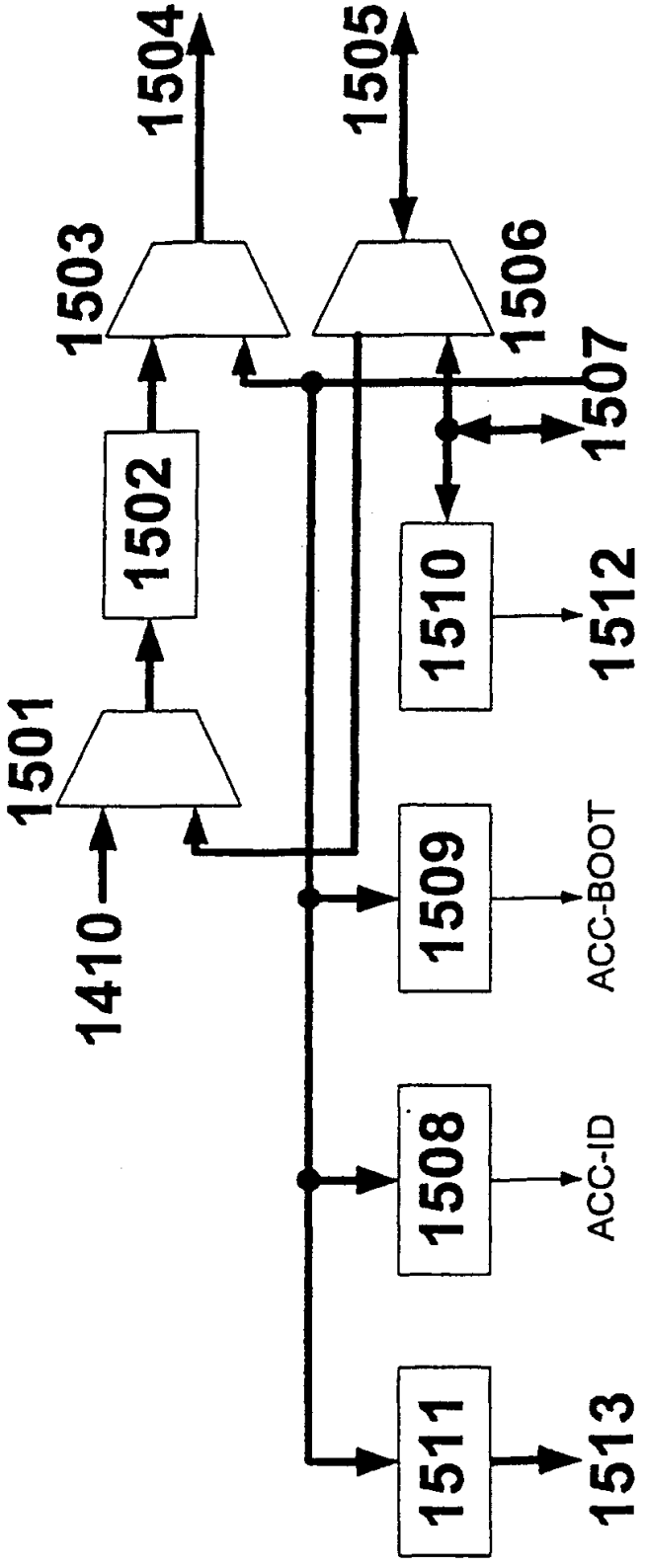

1504
1505
1503
1506
1507
1502
1510
1512
1501
1410
1509
ACC-BOOT
1508
ACC-ID
1511
1513

图 15

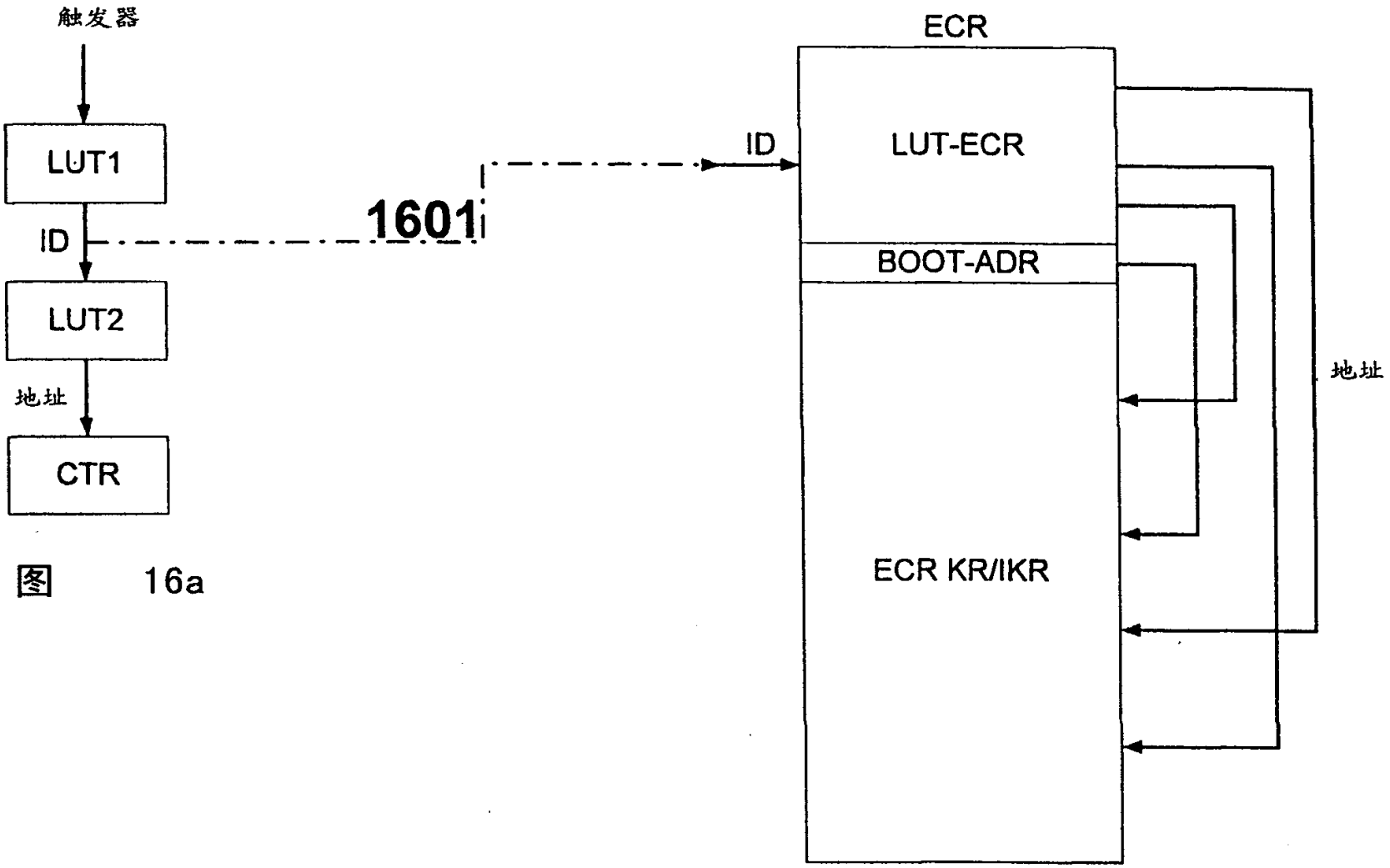

触发器
LUT1
ID
LUT2
地址
CTR
1601
ECR
ID
LUT-ECR
BOOT-ADR
ECR KR/IKR
地址

图 16a

图 16b

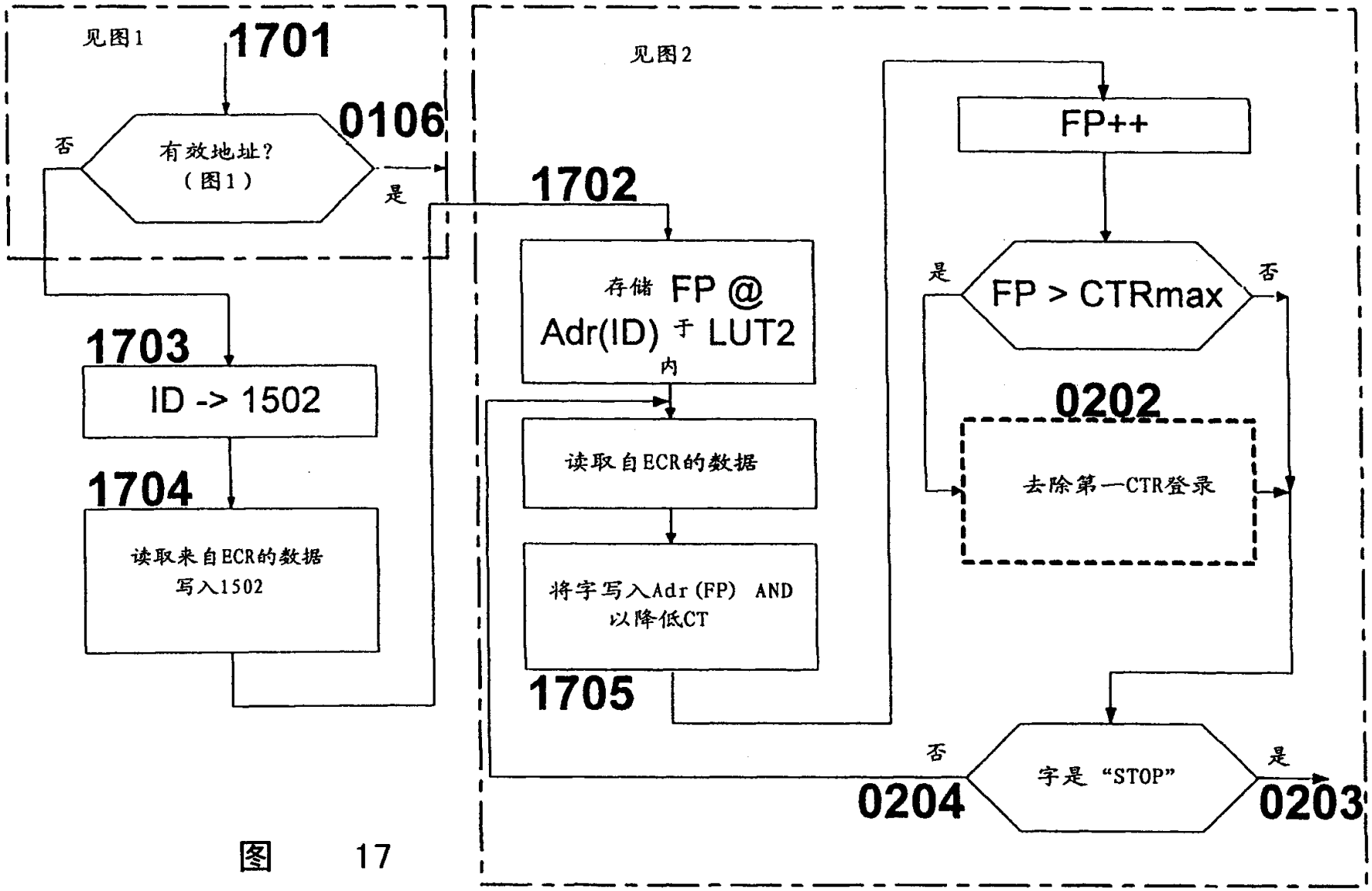

见图1
1701
0106
否
有效地址？
（图1）
是
1703
ID -> 1502
1704
读取来自ECR的数据
写入1502
见图2
1702
存储 FP @
Adr(ID) 于 LUT2
内
读取自ECR的数据
将字写入Adr (FP) AND
以降低CT
1705
FP++
是
FP > CTRmax
否
0202
去除第一CTR登录
否
字是 “STOP”
是
0204
0203

图 17

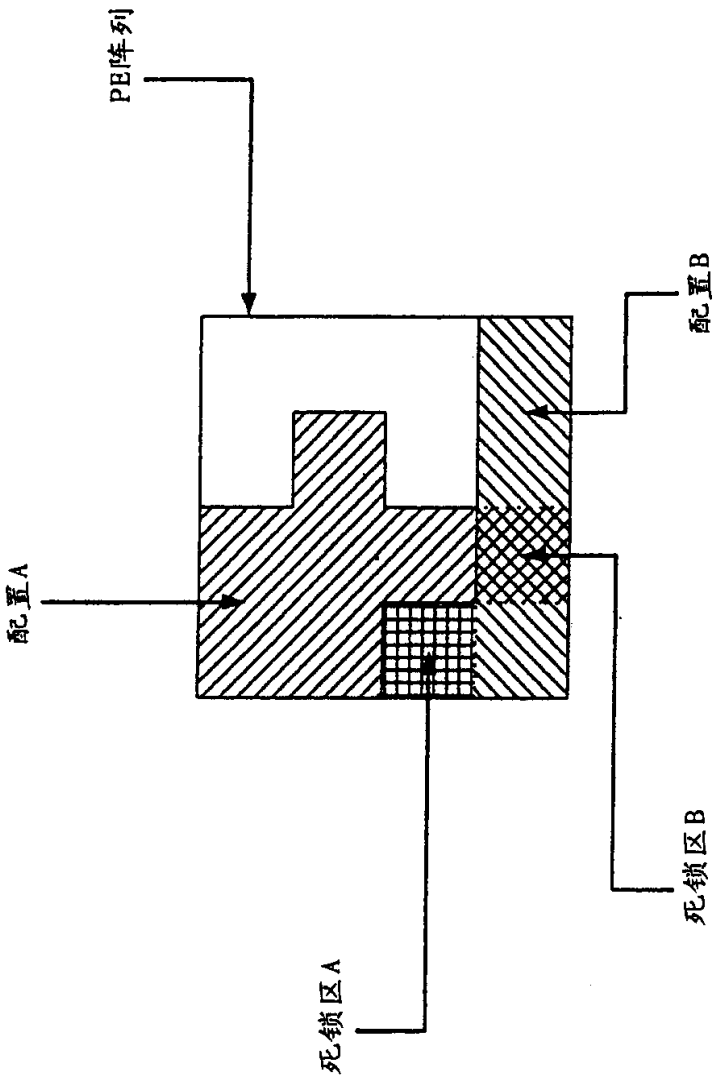


图 18

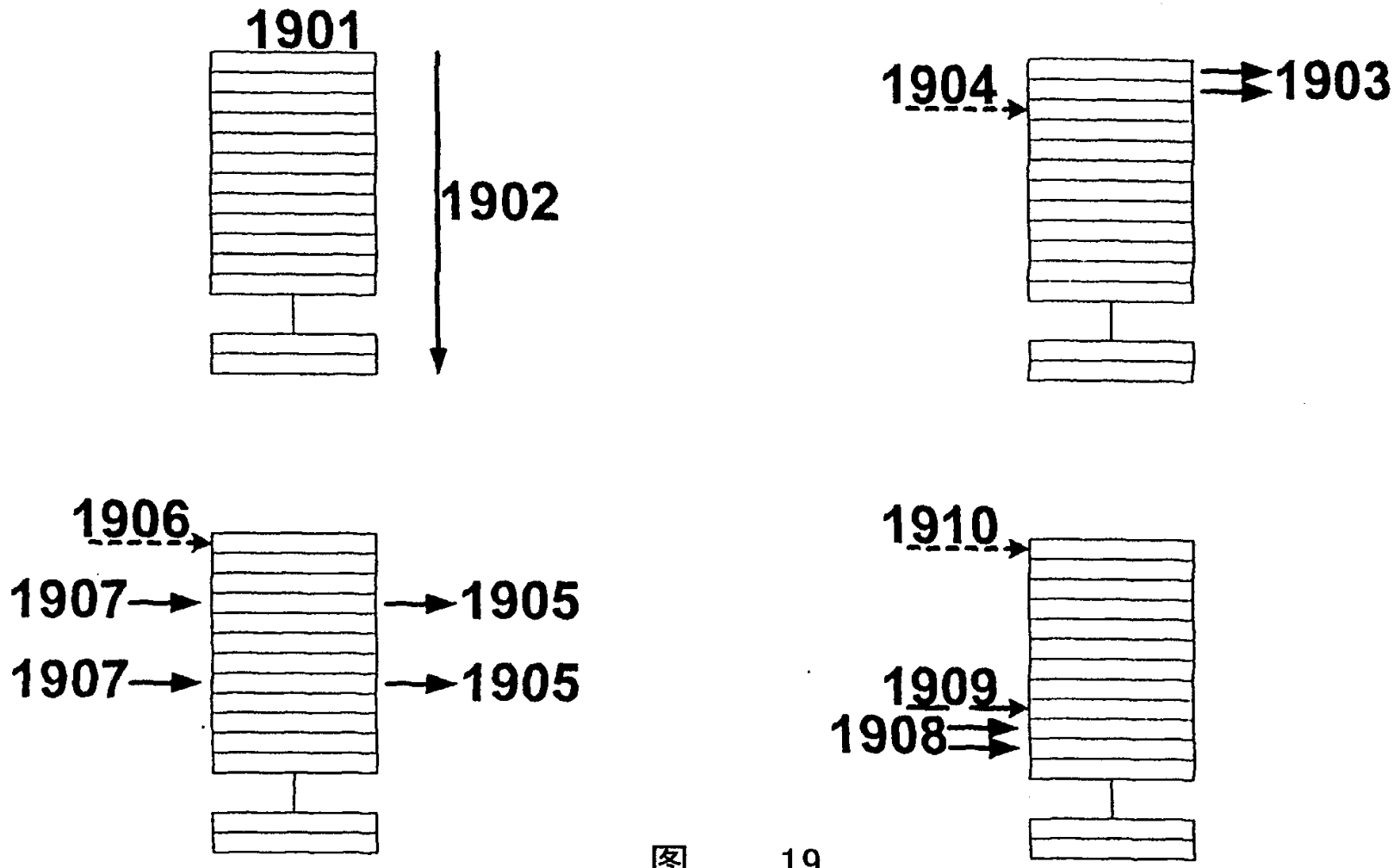


图 19

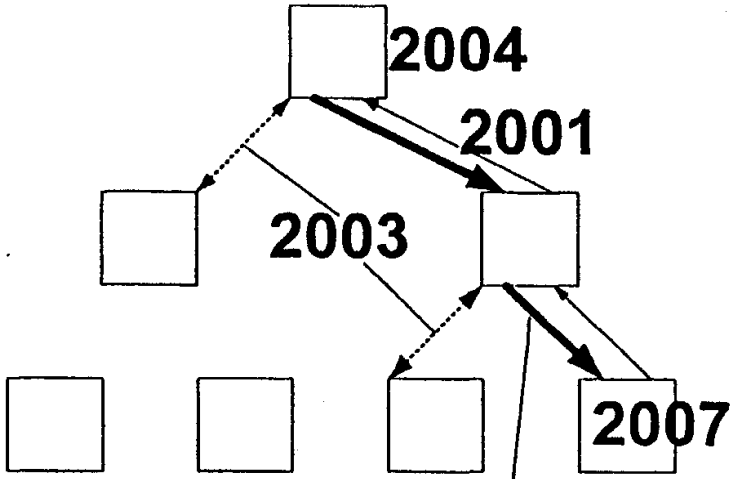


图 20a

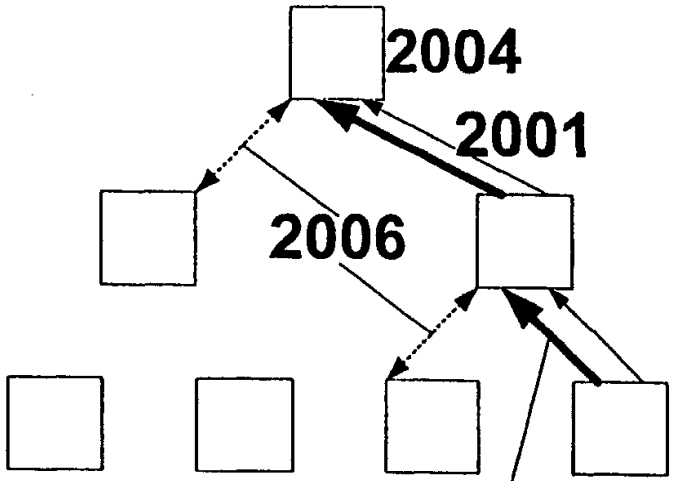


图 20b

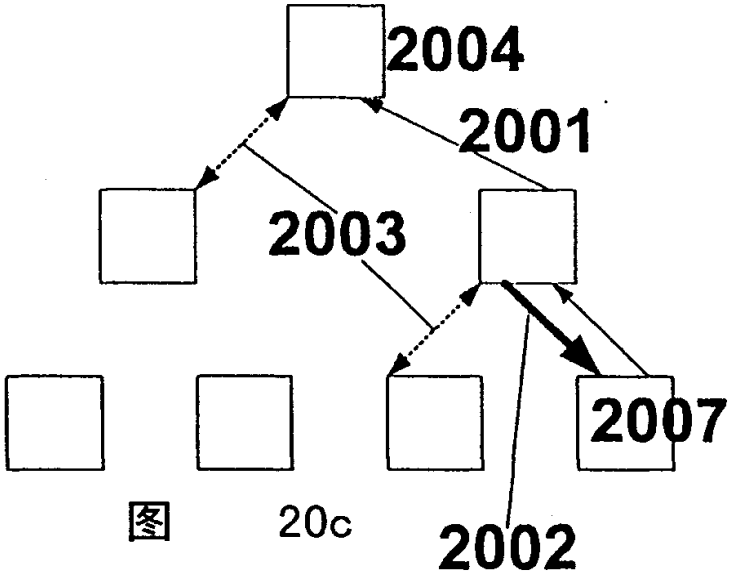


图 20c

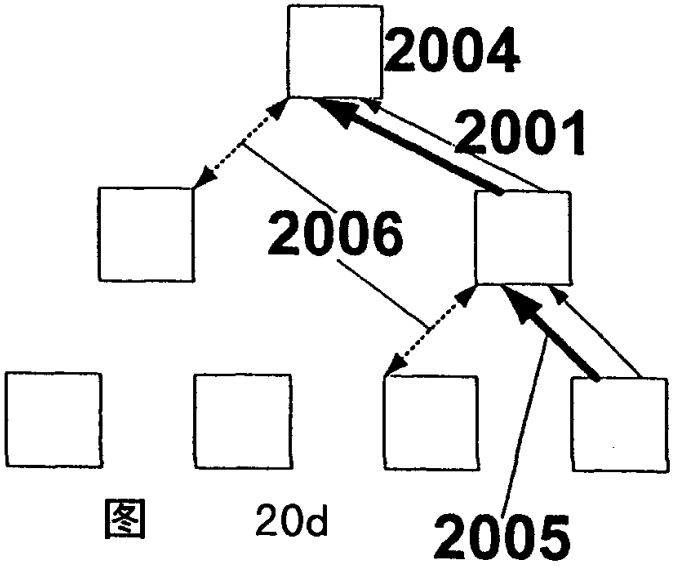


图 20d

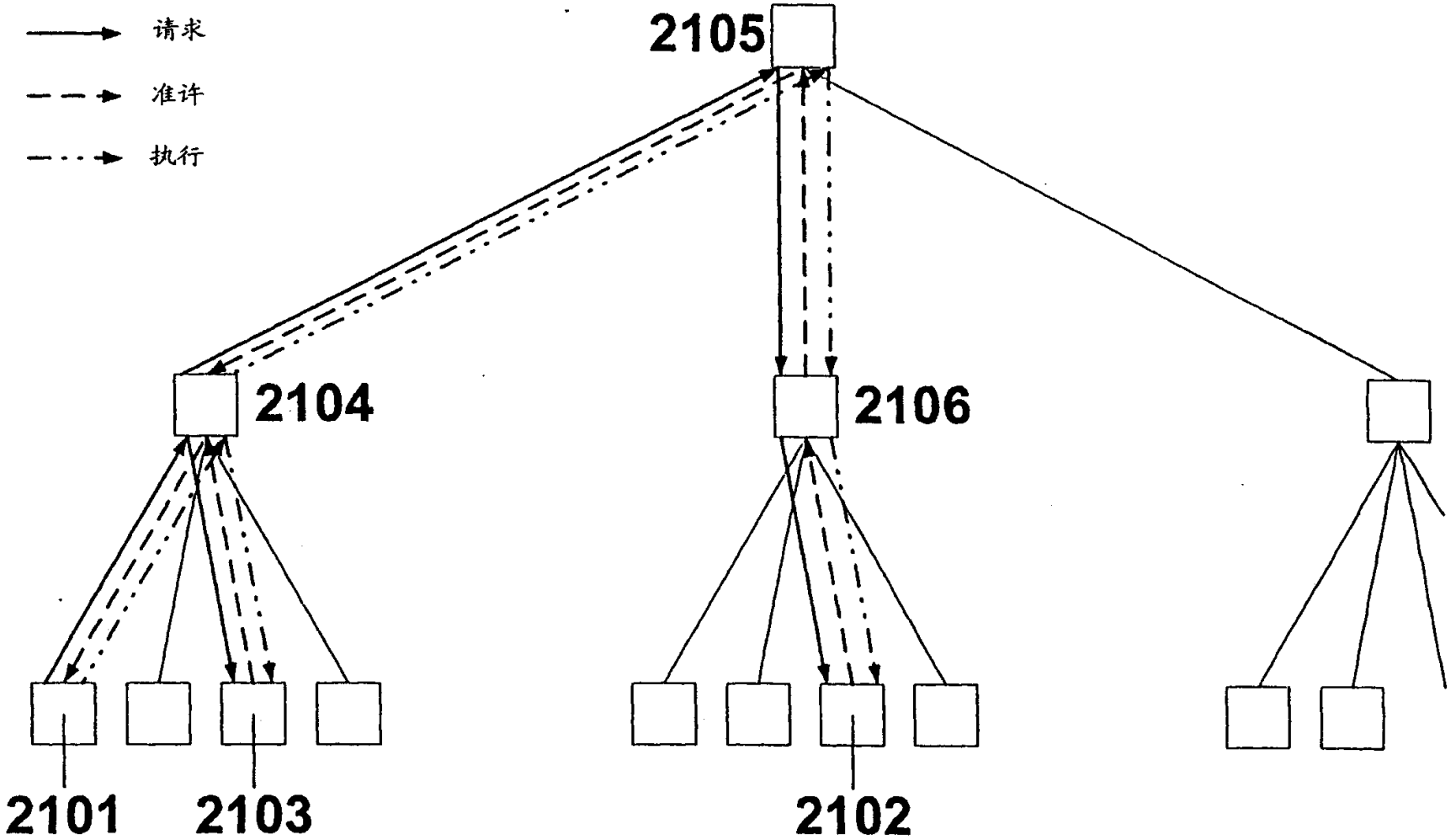


图 21

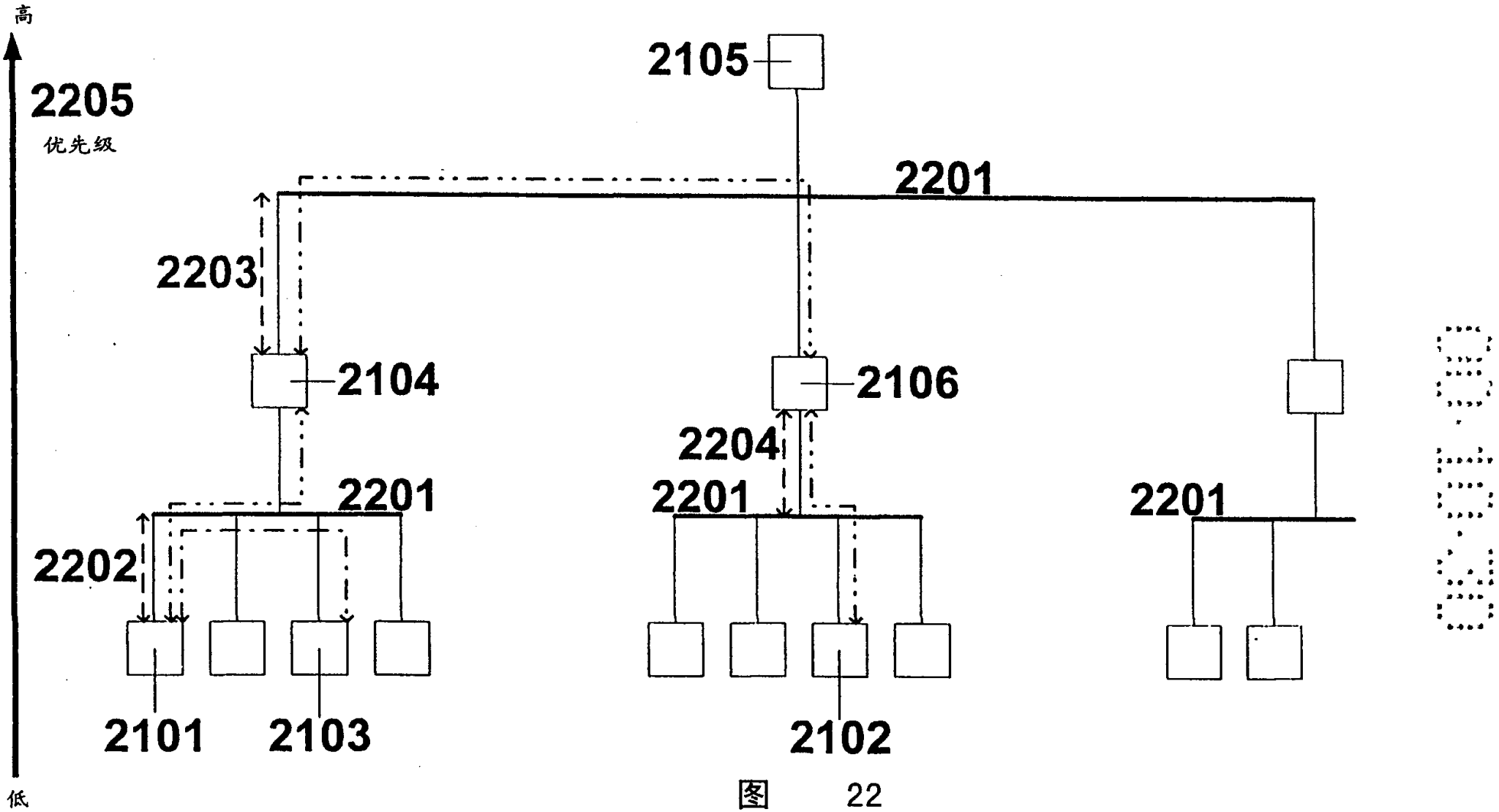

高
2205
优先级
低
2105
2201
2203
2104
2106
2204
2201
2201
2201
2202
2101
2103
2102

图 22

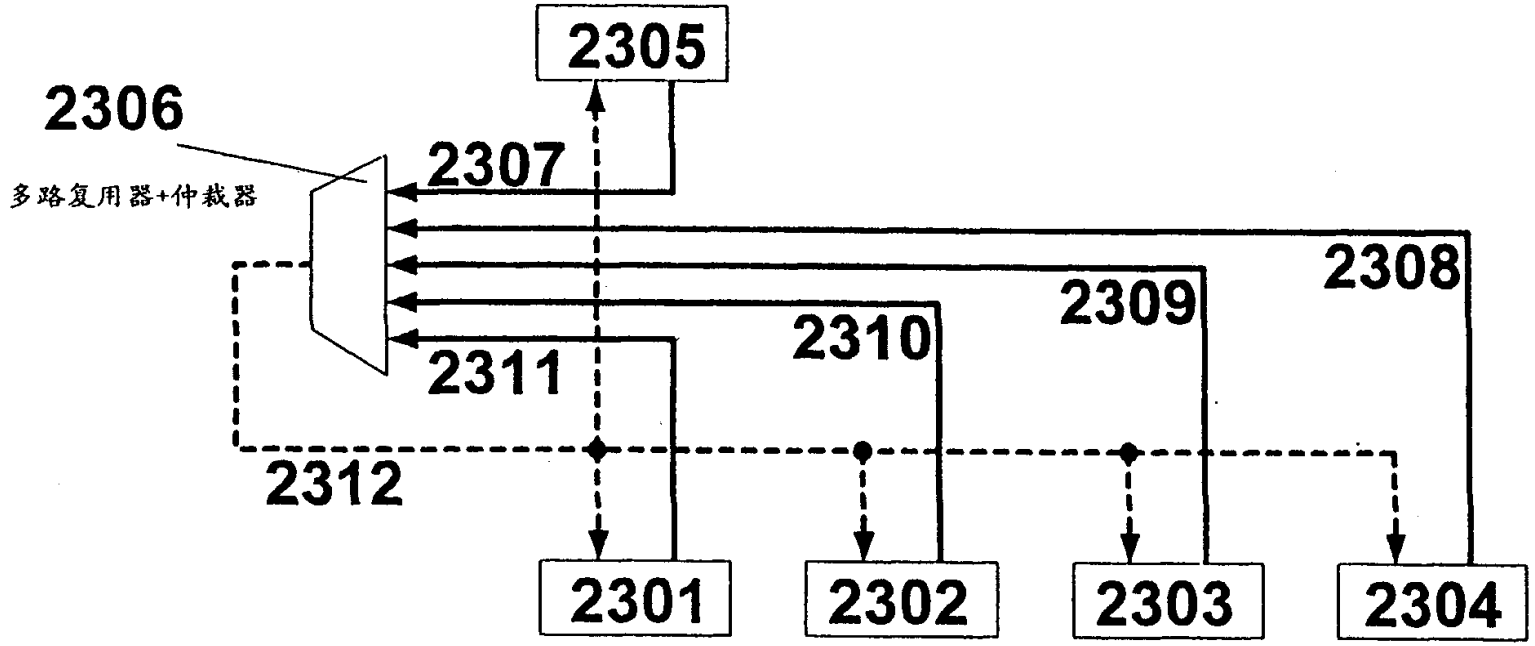

2305
2306
多路复用器+仲裁器
2307
2308
2309
2310
2311
2312
2301
2302
2303
2304

图 23

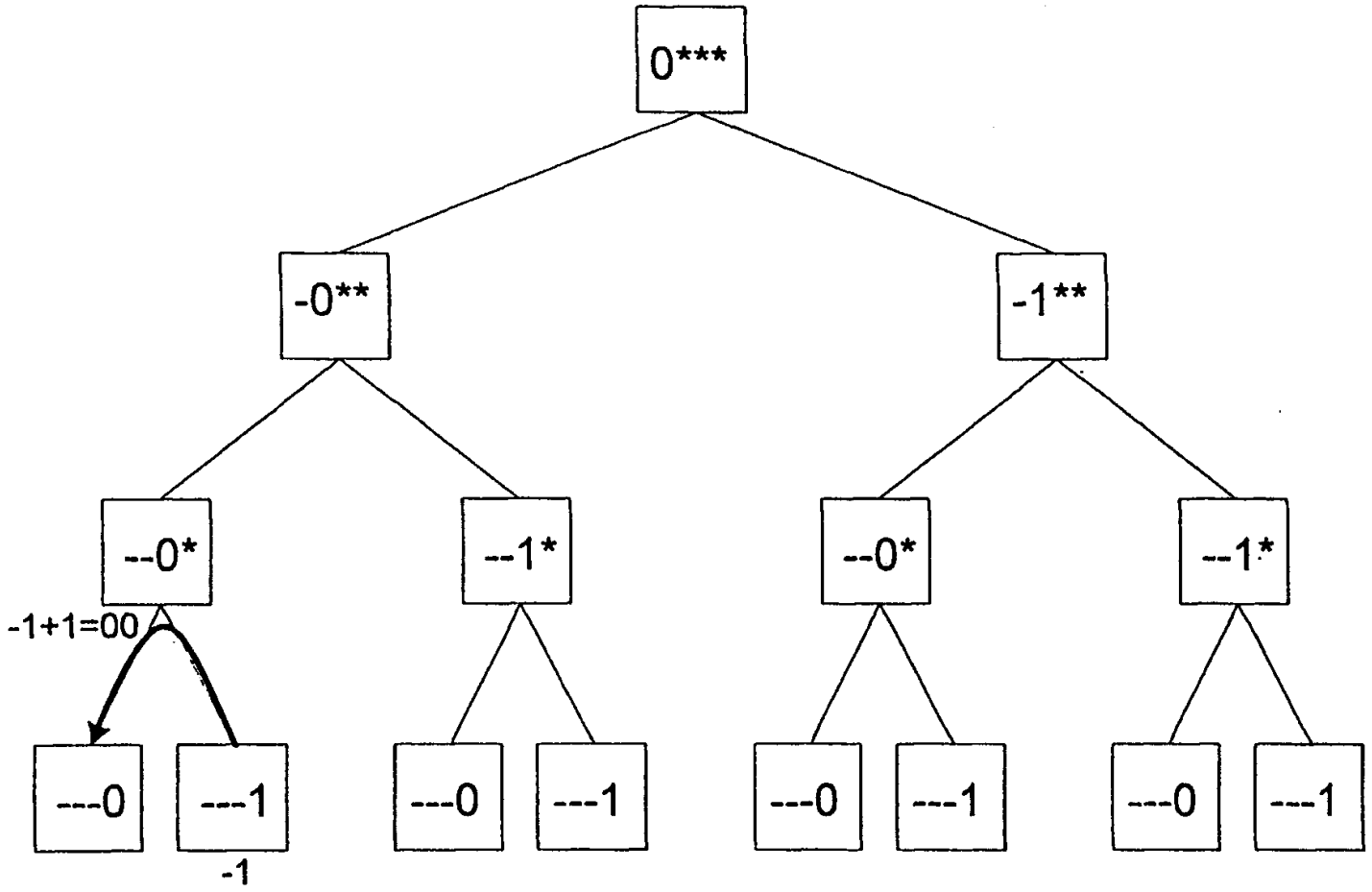


图 24a

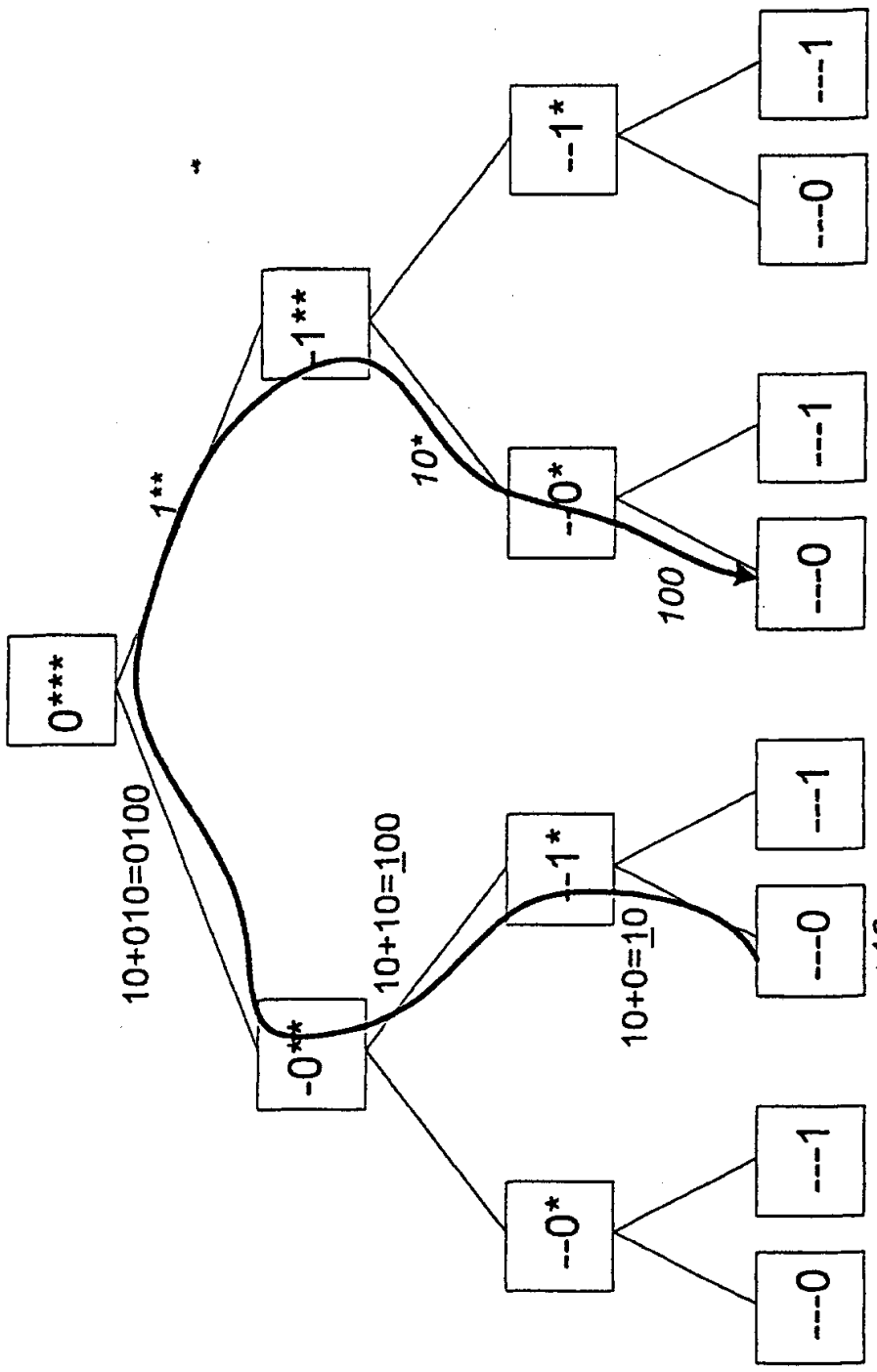


图 24b

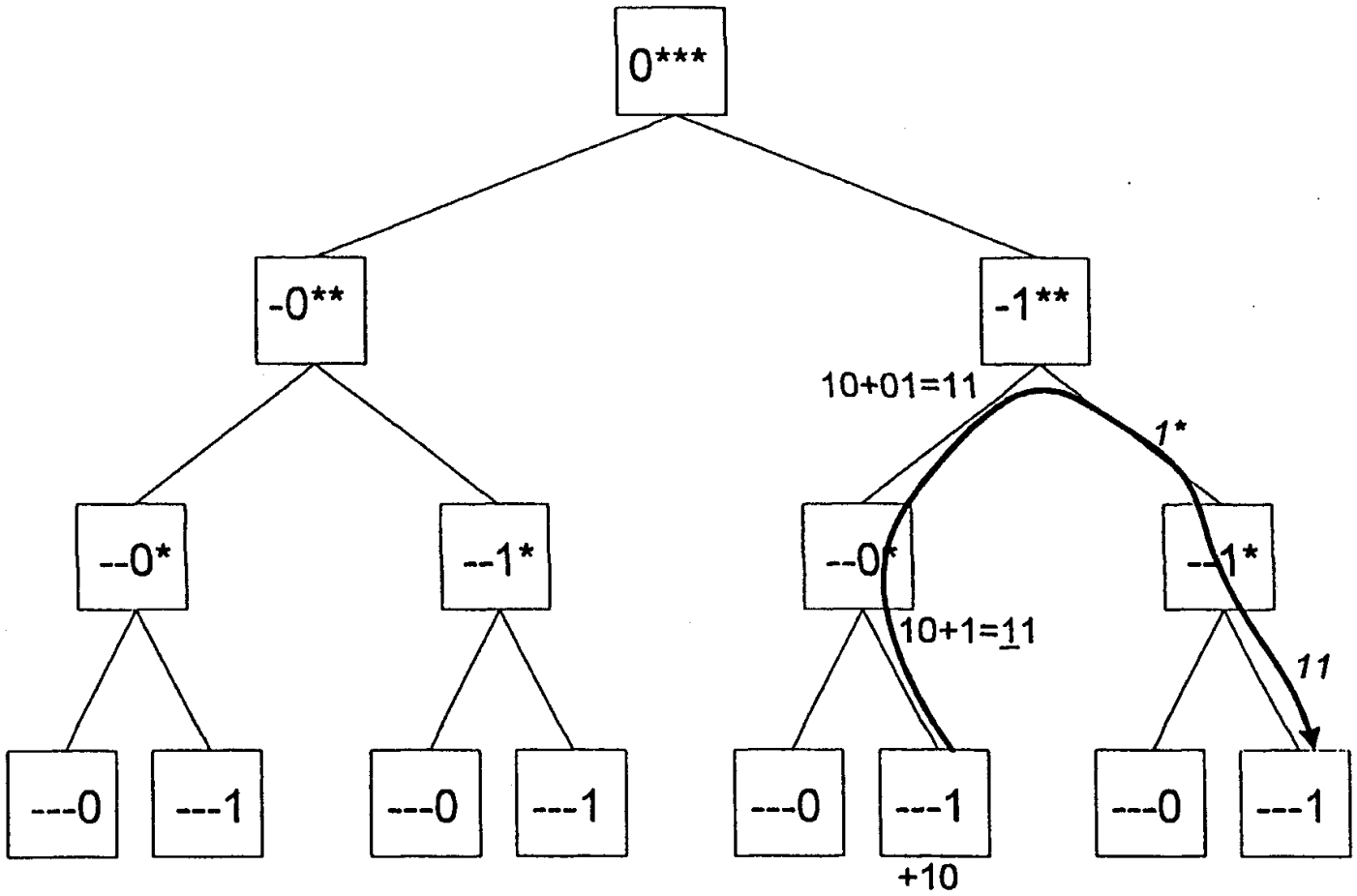


图 24c

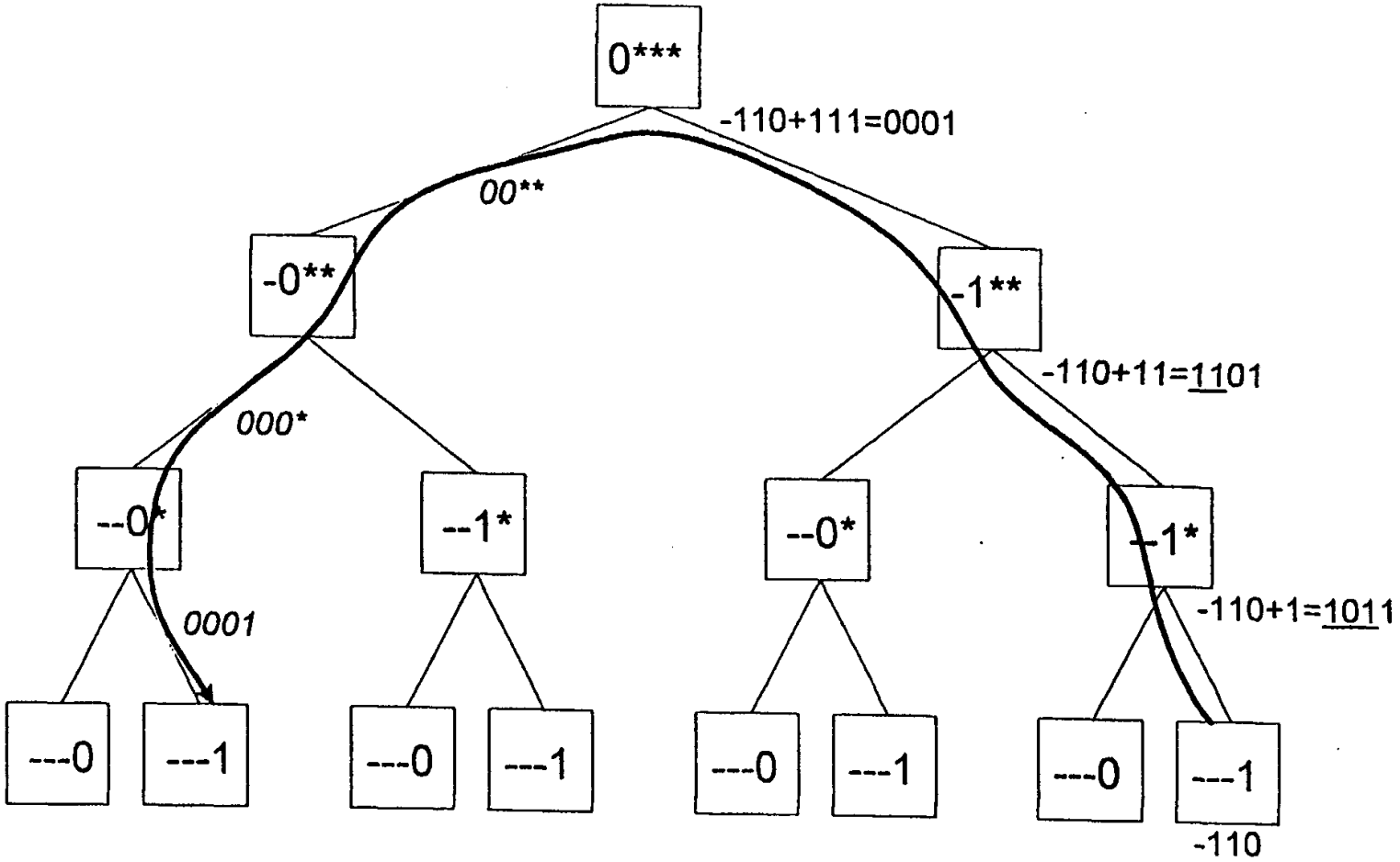

0***
-110+111=0001
00**
-0**
-1**
-110+11=1101
000*
--0*
--1*
--0*
--1*
-110+1=1011
0001
---0
---1
---0
---1
---0
---1
---0
---1
-110

图 24d

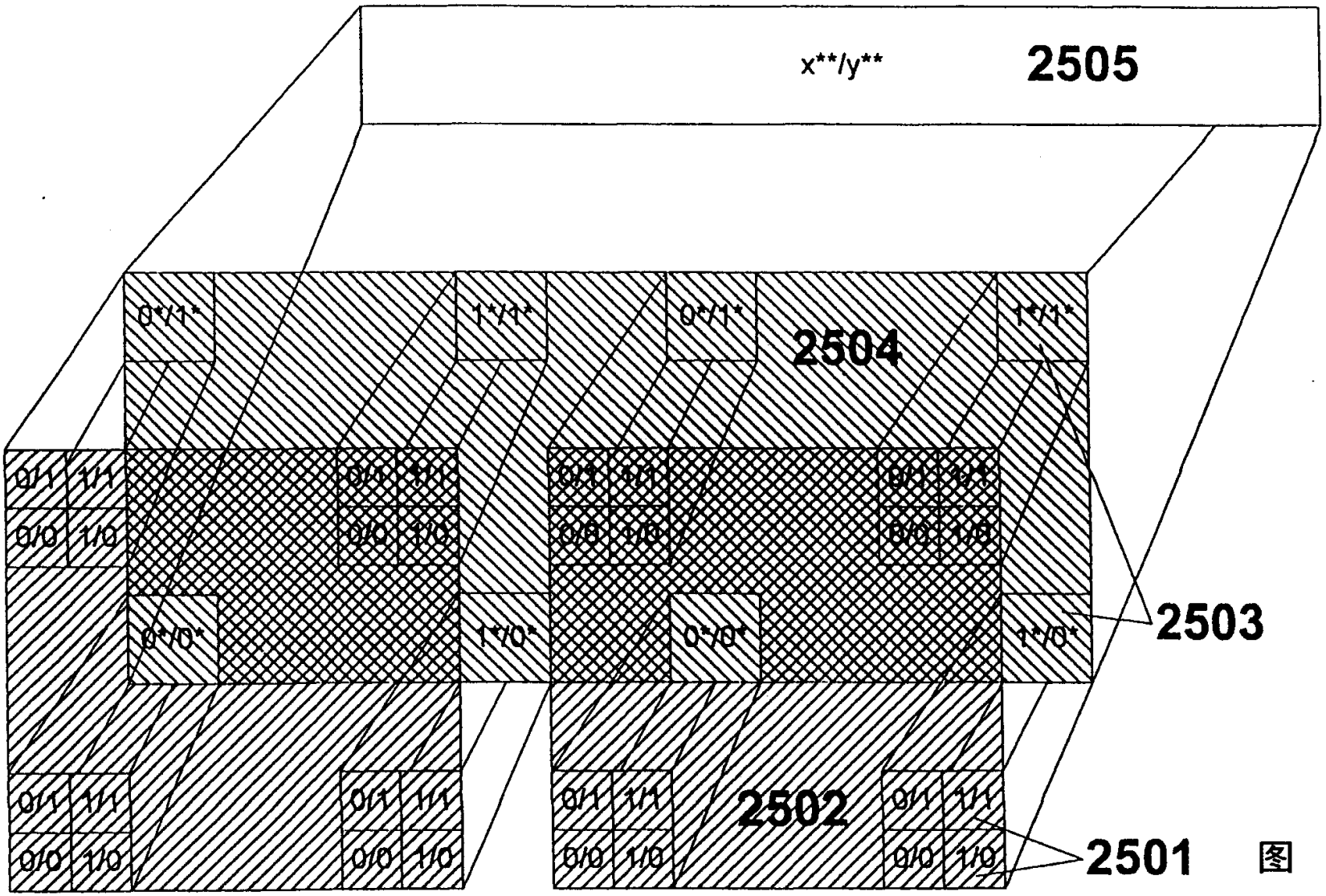


图 25

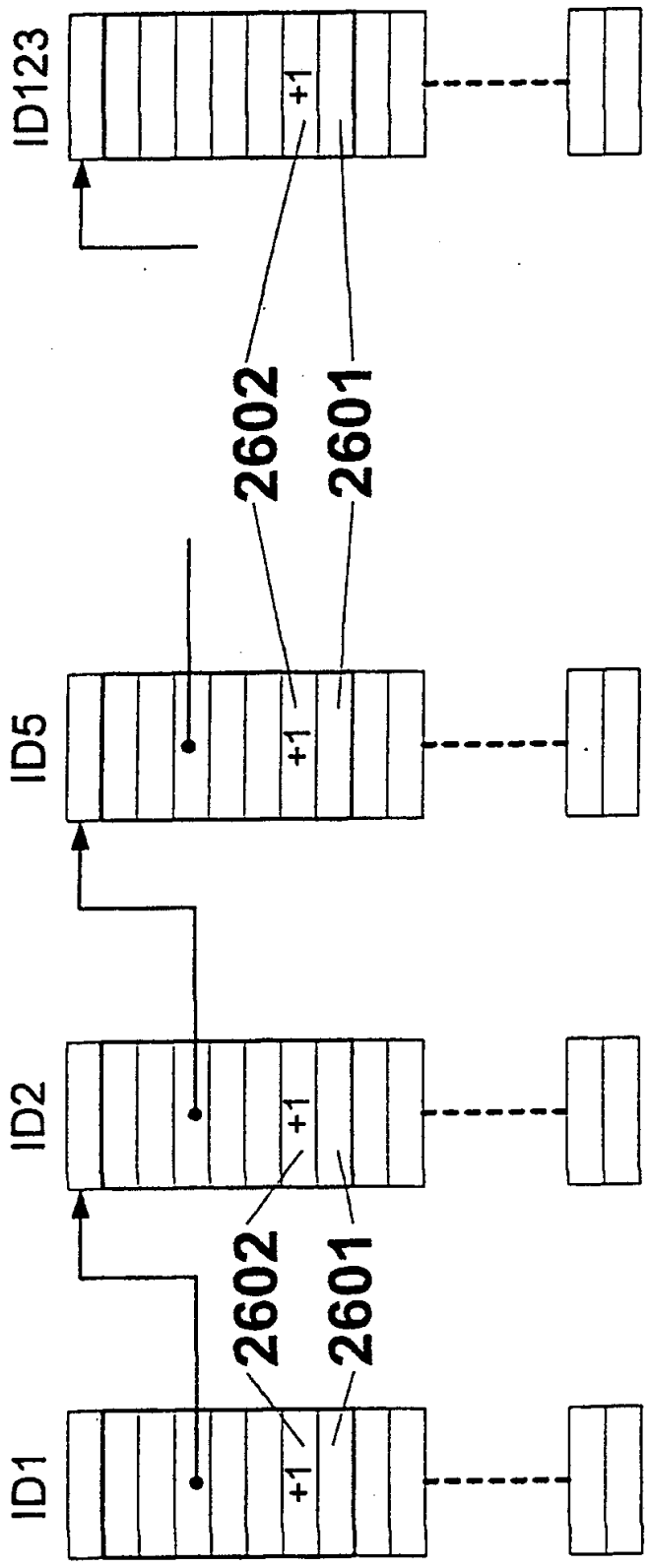


图 26

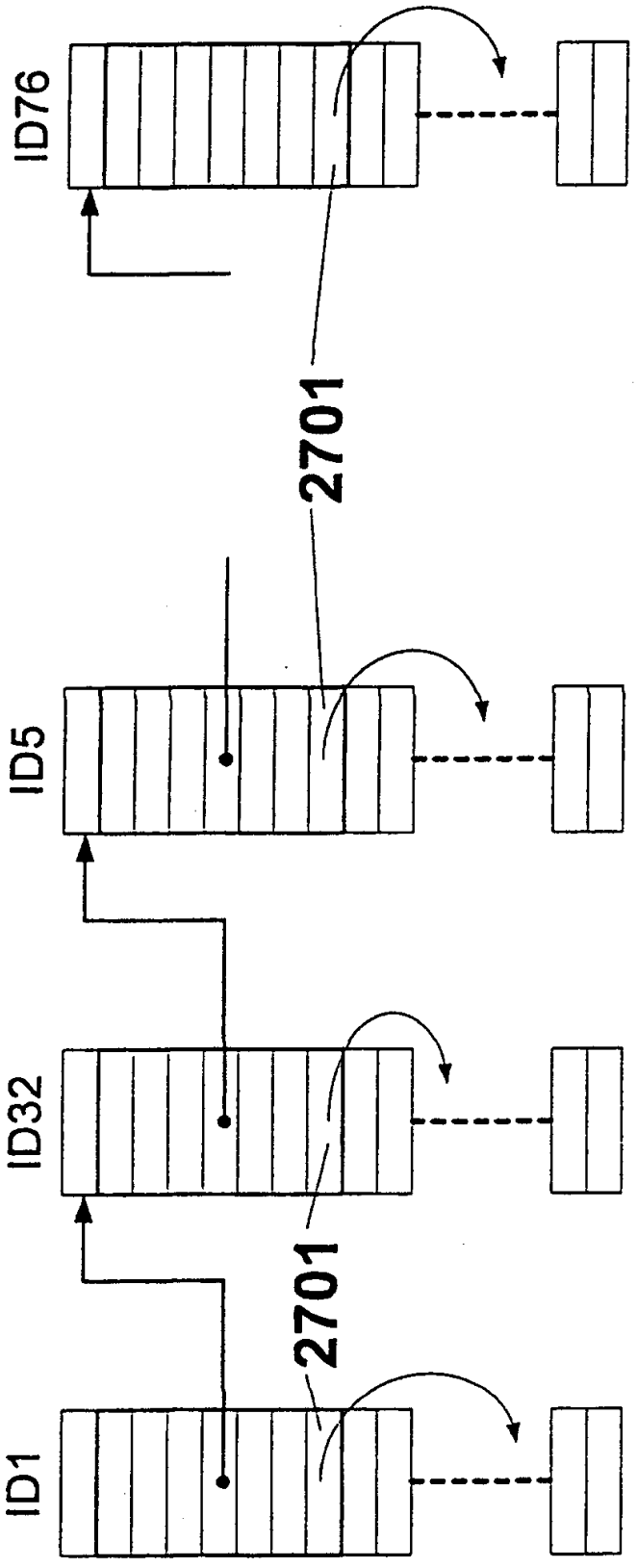


图 27

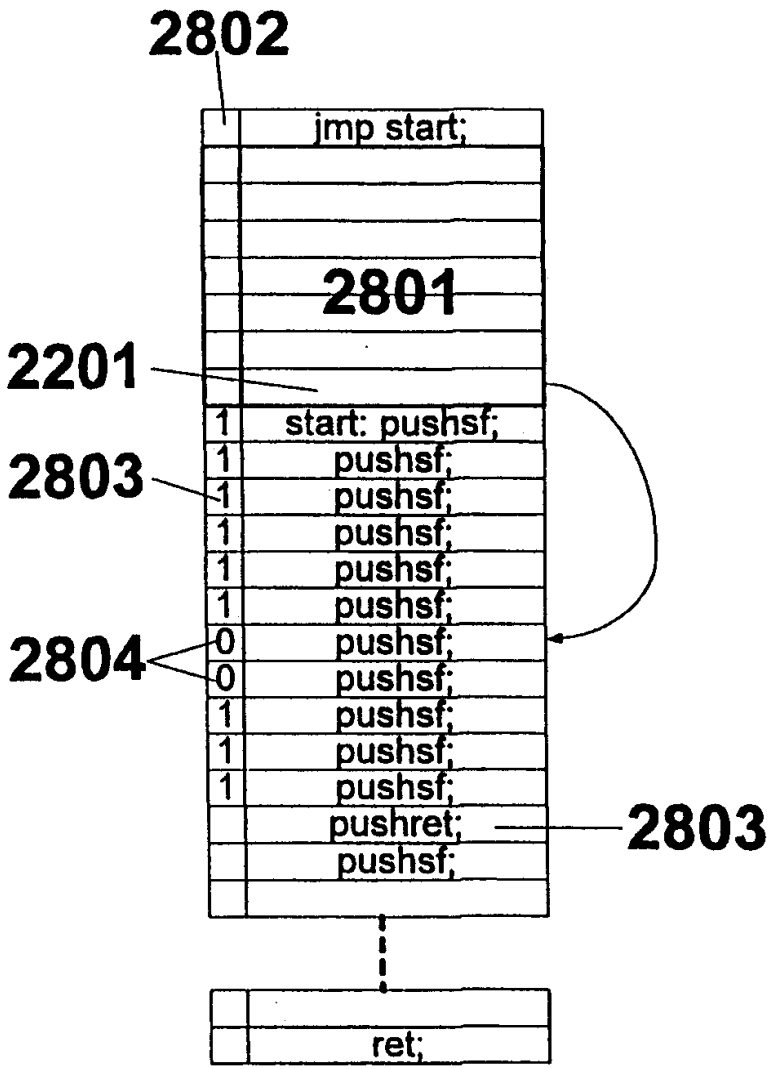

2802
jmp start;
2801
2201
1 start: pushsf;
2803
1 pushsf;
1 pushsf;
1 pushsf;
1 pushsf;
1 pushsf;
2804
0 pushsf;
0 pushsf;
1 pushsf;
1 pushsf;
1 pushsf;
pushret;
2803
pushsf;
ret;

图 28

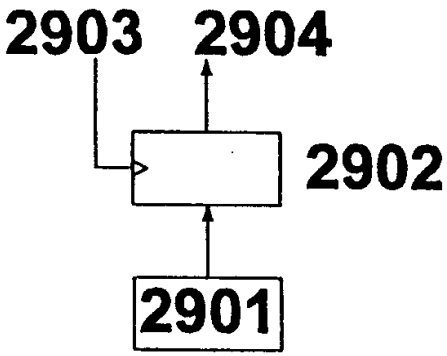

2903
2904
2902
2901

图 29