[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
H04L 9/00 (2006.01)
G06F 7/72 (2006.01)

[12] 发明专利说明书

专利号 ZL 03824410.1

[45] 授权公告日 2010 年 2 月 24 日

[11] 授权公告号 CN 100592676C

[22] 申请日 2003.8.25 [21] 申请号 03824410.1

[30] 优先权

[32] 2002. 8.26 [33] US [31] 10/228,151

[86] 国际申请 PCT/CA2003/001279 2003.8.25

[87] 国际公布 WO2004/019548 英 2004.3.4

[85] 进入国家阶段日期 2005.4.19

[73] 专利权人 睦塞德技术公司

地址 加拿大安大略

[72] 发明人 哈非兹·扎阿比

[56] 参考文献

WO02/052777A2 2002.7.4

US6088800A 2000.7.11

CN1355632A 2002.6.26

审查员 汤广强

[74] 专利代理机构 中科专利商标代理有限责任公司

代理人 朱进桂

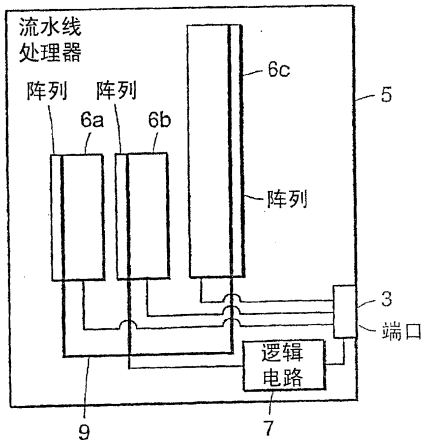
权利要求书 4 页 说明书 27 页 附图 15 页

[54] 发明名称

以相似的效率处理任意密钥位长加密操作的方法和设备

[57] 摘要

一种具有诸如以流水线形式排列的多级的计算设备或系统，该计算设备或系统具有沿着级安置的计时轨或导体。 在大量的即几百个级并行地排列成子阵列的情况下，计时导体沿着子阵列迂回延伸。在单个级中排列成，在级中进行的两个计算中的最短计算出现在返回路径中。 可以将阵列分成分开的部件用于独立处理。



1、一种处理加密过程中的数据的方法，所述方法包括以下步骤：

a) 提供用于加密的第一流水线处理器，包括以串行阵列的形式排列的 n 个单独处理部件，其中第一处理部件在领先于作为最终处理部件的第 n 处理部件的第二处理部件之前；

b) 提供加密密钥给第一流水线处理器；

c) 将指示加密密钥长度的数据提供给第一流水线处理器；

d) 将数据提供给第一处理部件以便加密处理；

e) 至少根据指示加密密钥长度的数据来确定第 N 个处理部件为最末处理部件，其中 N 小于或等于 n；

f) 将指示处理部件是最末处理部件的信号提供给最末处理部件；

g) 从第一处理部件以逐步正向串行的形式来传送用于进一步加密处理的数据，仅仅到最末处理部件为止；以及

其中，同一流水线处理器内的至少两个处理部件能够被确定为最末处理部件。

2、根据权利要求 1 所述的处理数据的方法，其中最末处理部件不同于最终处理部件。

3、根据权利要求 1 所述的处理数据的方法，其中加密密钥被填补到整数倍的预定位数，以及其中预定位数等于多个处理部件中的单个处理部件在一个处理周期期间处理的位数的整数倍。

4、根据权利要求 3 所述的处理数据的方法，其中只有能够被确定为最末处理部件的那些处理部件才装备有用于接收所述信号的电路。

5、根据权利要求 4 所述的处理数据的方法，其中少于总数目的处理部件能够被确定为最末处理部件。

6、根据权利要求 1 所述的处理数据的方法，包括以下步骤：

h) 将指示加密密钥长度的数据和要处理的数据提供给第一处理部件；

i) 利用第一处理部件内部的电路处理指示加密密钥长度的数据；

以及

j) 将经过处理的指示加密密钥长度的数据传送给至少下一个处理部件，以便利用所述至少下一个处理部件的内部电路进行附加处理。

7、根据权利要求 6 所述的处理数据的方法，其中处理指示加密密钥长度的数据的步骤是递减操作和递增操作之一。

8、根据权利要求 7 所述的处理数据的方法，包括以下步骤：

k) 当利用至少下一个处理部件的内部电路来处理指示加密密钥长度的数据的步骤返回指示该处理部件是最末处理部件的预定值时，在处理之后，仅仅沿前面的处理部件的方向上的返回处理路径来传送数据。

9、根据权利要求 1 所述的处理数据的方法，包括以下步骤：

提供第二流水线处理器，所述第二流水线处理器包括以串行阵列的形式排列的多个单独处理部件，其中第一处理部件在领先于最终处理部件的第二处理部件之前；

其中，组合使用第一流水线处理器和第二流水线处理器来进行加密处理。

10、根据权利要求 9 所述的处理数据的方法，其中第一流水线处理器和第二流水线处理器在第一模式下是独立的流水线处理器，以及在第二模式下是单一的组合流水线处理器。

11、一种处理加密过程中的数据的设备，所述设备包括：

以串行阵列的形式排列的 n 个单独处理部件，其中第一处理部件在领先于作为最终处理部件的第 n 处理部件的第二处理部件之前；

端口，至少与串行阵列的处理部件电通信，以便提供数据给串行阵列进行加密处理；以及

逻辑电路，该逻辑电路与所述端口以及串行阵列的至少两个处理部件电通信，用于在使用中处理指示加密密钥长度的数据以便在所述至少两个处理部件之中确定串行阵列的最末处理部件，以及根据指示加密密钥长度的数据而将信号提供给作为串行阵列的被确定最末处理部件的至少两个处理部件之一，其中所述至少两个处理部件的数目小于或等于 n，

其中，从第一处理部件以逐步正向串行的形式来传送用于进一步加密处理的数据，仅仅到最末处理部件为止。

12、根据权利要求 11 所述的处理数据的设备，其中多个单独处理部件排列成两个或多个流水线处理阵列，其中阵列用于独立地接收要处理的数据，或者用于当最末处理部件状态是由超出单个流水线处理阵列的最末处理部件以外的处理部件引起时，被连接到单个更大阵列中。

13、根据权利要求 11 所述的处理数据的设备，其中少于总数目的处理部件能够被确定为最末处理部件。

14、根据权利要求 13 所述的处理数据的设备，其中能够被确定为最末处理部件的处理部件进一步包括用于接收所述信号的电路。

15、根据权利要求 14 所述的处理数据的设备，其中逻辑电路是用于通过可寻址数据路径将信号提供给确定的处理部件的门逻辑开关电路。

16、根据权利要求 14 所述的处理数据的设备，其中逻辑电路包括：

第二处理器，用于执行用来根据指示密钥长度的数据来确定最末处理部件的程序代码以及提供所述信号给确定的处理部件；以及

在第二处理器与串行阵列的至少两个处理部件之间延伸的通信路径，所述通信路径用于将所述信号从第二处理器引导至确定的处理部件。

17、一种处理加密过程中的数据的设备，所述设备包括：

以串行阵列的形式排列的多个单独加密处理部件，其中第一加密处理部件在领先于作为最终加密处理部件的第 n 加密处理部件的第二加密处理部件之前；以及

第一加密处理部件与第 n 加密处理部件之间的每个单独加密处理部件内部的逻辑电路，用于在使用中处理指示加密密钥长度的数据，以及在加密处理部件的内部提供指示最末加密处理部件状态的信号，该信号根据指示加密密钥长度的数据来提供，

其中，从第一加密处理部件以逐步正向串行的形式来传送用于进一步加密处理的数据，仅仅到最末加密处理部件为止。

18、根据权利要求 17 所述的处理数据的设备，其中多个单独加密处理部件排列成两个或多个流水线处理阵列，其中阵列用于独立地接收要加密处理的数据，或者用于当最末加密处理部件状态是由超出单个流水线处理阵列的最末加密处理部件以外的加密处理部件引起时，被连接到单个更大阵列中。

以相似的效率处理任意密钥位长加密操作的方法和设备

本申请是2002年8月26日提交的美国申请No. 10/228,151的继续，并要求其优先权。以上申请的整个教导在此通过参考而被引入。

技术领域

本发明一般涉及并行处理器，并尤其涉及一种能够利用相同处理器、以相似的效率来处理任意密钥位长加密操作的并行处理器。

背景技术

例如通过诸如万维网（WWW）的因特网的广泛分布式信息网络在各方之间交换电子存储的文件正变得较为常见。因特网的常见问题是缺少安全通信信道。因而，为了使医院、政府、银行、股票经纪人和信用卡公司利用因特网，必须确保保密性和安全性。解决上述问题的一种方法是在发送之前使用数据加密。在现有技术系统中，主计算机系统装备有加密单元，例如与用于至少存储私有加密密钥的至少一个存储电路进行电通信的加密处理器。当信息要从主计算机系统、通过因特网发送给接收器、并且具有机密性质时，信息首先被传递到加密处理器，以便利用存储的私有密钥对信息加密。典型地，每当执行加密操作时，都使用相同的私有密钥。作为替换，从与加密处理器进行电通信的至少一个存储电路中存储的一组有限的加密密钥中选择加密密钥。

当然，由加密处理器执行的数据加密操作是算术算法，其中输入数据值，例如散列型式的电子文件，是唯一变量值。因此，有可能对加密处理器进行优化，以便利用最少量的处理器资源来执行期望的加密功能。另外，在现有技术加密单元中，优化的加密处理器典型地与主计算机系统的微处理器分开，因为加密单元这样被最佳地优化。

现今，对于通过加密/解密的因特网上保密性和强鉴定有几种标准。典型地，根据意图允许在各方之间在公开信道上进行数据传送，同时维持消息内容的保密性的算法，来执行加密/解密。这是通过由发送器利用加密密钥对数据加密，并由接收器利用解密密钥对数据解密来实现的。在对称密钥密码术中，加密密钥和解密密钥相同。

加密算法典型地被分类为公开密钥（public-key）和秘密密钥（secret-key）算法。在秘密密钥算法中密钥是秘密的，而在公开密钥算法中，使密钥之一广为公众所知。分组密码是当今使用的秘密密钥密码系统的代表。通常，对于分组密码，使用对称密钥。分组密码取一块数据，典型地为 32-128 位，作为输入数据，并产生相同的位数作为输出数据。利用具有典型地在 56-128 位范围内的长度的密钥，来执行加密和解密操作。加密算法被设计成在不知道密钥的情况下很难对消息解密。

除分组密码（block cipher）以外，因特网安全协议也依赖于基于公开密钥的算法。诸如 Pogue 和 Rivest 的美国专利 No. 5,144,667 中描述的 Rivest、Shamir、Adelman（RSA）加密系统的公开密钥加密系统使用两个密钥，其中一个是秘密—私有的，而另一个是公开可得的。一旦某人公开了公开密钥，任何人都可以向那个人发送利用那个公开密钥加密的秘密消息；然而，只能利用私有密钥来实现消息的解密。这种公开密钥加密的优点是，事先不将私有密钥分发给会话的所有方。相反，当使用对称加密时，多个秘密密钥被产生，想要接收消息的每一方一个秘密密钥，并且每个秘密密钥被秘密地传送。试图以安全的形式分发秘密密钥导致了与只利用秘密密钥加密发送消息所面临的问题类似的问题；这典型地被称为密钥分发问题。

密钥交换是公开密钥技术的另一个应用。在密钥交换协议中，双方能够约定秘密密钥，即使第三方截取了他们的会话。美国专利 No.4,200,770 中描述的 Diffie-Hellman 指数密钥交换方法是这种协议的例子。

大多数的公开密钥算法，诸如 RSA 和 Diffie-Hellman 密钥交换，是基于模取幂，模取幂是α^x模 p 的计算。该表达式表示，“使α乘以它

自己 x 倍，使答案除以 p，并取余数”。由于以下原因，该计算执行起来计算量非常大。为了执行该操作，需要许多重复的乘法操作和除法操作。诸如“Modular Multiplication Without Trial Division”，Mathematics of Computation, Vol.44, No.170, April, 1985 中描述的 Montgomery 方法的技术能够减少除法操作数，但是没有克服该总体计算开销。另外，对于当前的数据加密系统，使用的数非常大（典型地为 1024 位或更多），因此普通中央处理器（CPU）中存在的乘法和除法指令不能直接使用。而是，使用特定算法来将大的乘法操作和除法操作分解成小得足以在 CPU 上执行的操作。这些算法通常具有与所涉及的机器字的数量的平方成比例的运行时间。例如，Pentium®处理器能够在 10 个时钟周期内执行 32×32 位乘法。2048 位的数可以表示为 64 个 32 位的字。2048×2048 位的乘需要 64×64 个单独的乘法操作，这在 Pentium®处理器上花费 40960 个时钟。如果直接执行，则对 2048 位指数取幂需要直到 4096 次乘法操作，这需要大约 167 百万个时钟周期。如果 Pentium®处理器以 166MHz 运行，则全部操作大概需要一秒。当然，除法操作进一步增加了整个计算时间。显然，诸如 Pentium®的普通 CPU 不能期望以任何高的速率来执行密钥产生和交换。

包括以串行阵列形式排列的多个独立处理部件尤其是大量处理部件的流水线处理器在现有技术中是周知的，并且尤其适于执行数据加密算法。两种类型的流水线处理器是周知的：一端进且另一端出类型的处理器，其中存在单一处理方向；以及同一端进和出类型的双向处理器，其中存在正向处理方向和返回处理方向。考虑双向流水线处理器的特定例子，第一数据块从存储缓冲器被读入串行阵列的第一处理部件，第一处理部件执行第一阶段处理，然后将第一数据块传递给第二处理部件。第二处理部件执行第二阶段处理，同时第一处理部件并行地从存储缓冲器读出第二数据块，并对第二数据块执行相同的第一处理阶段。依次地，每个数据块以逐步的形式沿串行阵列的正向处理方向从一个处理部件传播到下一个处理部件。在每一步都有处理阶段，该处理阶段对所提供的每一个数据块执行相同的数学运算。同时，关于返回处理方向，在每一个处理部件中计算的结果被提供给串行阵列

的前一个处理部件，这些结果总计包括了加密处理器返回的经过处理的数据。这种使用大量处理部件的流水线数据处理方法是执行前述的大计算量的数据加密算法的非常有效方法。当然，用于执行大计算量处理操作的流水线处理器的应用并不严格限于已经举例详细讨论的数据加密算法。

现有技术加密处理器的缺点是，处理器限于固定的密钥大小或者作为替换地限于固定的性能。因而，或者处理器只处理例如 128 位的密钥，或者处理器需要用与处理 40 位加密操作所花时间相等的时间，来处理 128 位加密操作。典型地，这两种设计要求考虑到处理器的最佳性能。

提供一种能够利用相同处理器以相似的效率来处理任意密钥位长加密操作的并行处理器将是有利的。

发明内容

根据本发明，提供一种处理数据的方法，该方法包括以下步骤：

a) 提供包括以串行阵列的形式排列的 n 个单独处理部件的流水线处理器，以致第一处理部件在领先于第 n 处理部件的第二处理部件之前；

b) 提供 m 位加密密钥给流水线处理器；

c) 将指示加密密钥长度的数据提供给流水线处理器；

d) 将数据提供给第一处理部件以便处理；

e) 至少根据指示加密密钥长度的数据来确定最末处理部件；

f) 将指示处理部件是最末处理部件的信号提供给最末处理部件；

g) 从第一处理部件以逐步正向串行的形式来传送用于进一步处理的数据，仅仅到最末处理部件为止；以及其中，同一流水线处理器内的至少两个处理部件可以被确定为最末处理部件。

根据本发明，提供一种处理数据的设备，备包括：

以串行阵列的形式排列的多个单独处理部件，其中第一处理部件在领先于第 n 处理部件的第二处理部件之前；

端口，至少与串行阵列的处理部件电通信以便提供数据给串行阵

列进行处理；以及

逻辑电路，该逻辑电路与所述端口以及串行阵列的至少两个处理部件电通信，用于在使用中处理指示加密密钥长度的数据以便确定串行阵列的最末处理部件，以及根据指示加密密钥长度的数据将信号提供给作为串行阵列的被确定处理部件的至少两个处理部件之一。

根据本发明，提供一种处理数据的设备，包括：

以串行阵列的形式排列的多个单独处理部件，其中第一处理部件在领先于第 n 处理部件的第二处理部件之前；以及

每个单独处理部件内部的逻辑电路，用于在使用中处理指示加密密钥长度的数据，以及将指示最末处理部件状态（last processing element status）的信号提供给处理部件的内部，该信号根据指示加密密钥长度的数据来提供。

附图说明

由以下连同附图的优选实施例的说明，本发明将更容易理解，其中：

图 1 显示了根据先有技术的流水线处理器的实施例的简化框图；

图 2 显示了根据先有技术的流水线处理器的另一个实施例的简化框图；

图 3a 显示了根据本发明第一实施例的具有分布式递减电路的流水线处理器的简化框图；

图 3b 显示了与分布式递减电路通信的图 3a 的流水线处理器的串行处理器阵列的简化框图；

图 4a 显示了根据本发明第二实施例的具有电路的流水线处理器的简化框图；

图 4b 显示了与该电路通信的图 4a 的流水线处理器的串行处理器阵列的简化框图；

图 5a 显示了根据本发明第三实施例的流水线处理器的简化框图；

图 5b 显示了其中每个处理部件都具有内部递减电路的图 4a 的流水线处理器的串行处理器阵列的简化框图；

图 6 显示了根据本发明的流水线处理器的第四优选实施例的简化框图；

图 7 显示了根据本发明的流水线处理器的第五优选实施例的简化框图；

图 8 是供用于执行加密功能的流水线阵列处理器之用的资源高效处理部件设计的框图；

图 9 是用于模乘的脉动阵列（systolic array）的框图；

图 10 是其输入路径被显示的单个单元的框图；

图 11 是 DP RAM Z 单元的框图；

图 12 是 Exp RAM 单元的框图；

图 13 是 Prec RAM 单元的框图；

图 14 是供用于执行加密功能的流水线阵列处理器之用的速度高效处理部件设计的框图；

图 15 是用于模乘的脉动阵列的框图；

图 16 是其输入路径被显示的单一单元的框图；以及

图 17 是 DP RAM Z 单元的框图。

具体实施方式

本发明涉及供加密操作之用的并行处理器的实施过程，该实施过程使得并行处理器在没有大大牺牲效率的情况下支持可变长度加密密钥。

参考图 1，图 1 显示了根据现有技术的流水线处理器 1 的简化框图。处理器 1 包括多个处理部件，例如支持 256 位加密的串行阵列 2。输入/输出端口 3 与阵列 2 的第一处理部件（未显示）通信，用于从例如也在操作中与端口 3 通信的客户站（client station）（未显示）接收要被处理器 1 处理的数据。

为了执行加密操作，将预定长度的加密密钥提供给处理器 1，并且执行加密操作。作为替换，如果加密密钥少于预定长度，则填补加密密钥以使加密密钥变成预定长度，然后执行操作。在任何一种情况下，操作都花费近似相同的预定量时间。不幸的是，当密钥长度变得

更长时，用于处理以前的更短密钥的现有技术处理器 1 的效率降低了。例如，被设计成可供 256 位密钥使用的处理器将以用于处理仅仅 40 位密钥的大约六分之一“高效”时间，来处理被填补到 256 位的 40 位密钥。这是很差的资源分配。为了更好地分配资源，一些处理器包括多个处理器阵列，每个处理器阵列用于处理不同长度的加密密钥，如图 2 所示。因而，处理器将包括 40 位加密处理器阵列 2a，128 位加密处理器阵列 2b 和 256 位加密处理器阵列 2c。输入/输出端口 3 分别与每个阵列 2a、2b、2c 的第一处理部件通信，用于从例如也在操作中与端口 3 通信的客户站（未显示）接收要被处理器 4 处理的数据。这种高效实施的资源使用的效率低，因而是不合需要的。

现在参考图 3a，图 3a 显示了根据本发明第一实施例用于以相似的效率来处理任意密钥位长加密操作的流水线处理器 5 的简化框图。流水线处理器 5 包括至少一个阵列，并且在图 3a 中显示了多个处理器部件（处理器部件未显示）阵列 6a、6b、6c，例如阵列 6a 和 6b 的每一个都支持 256 位加密，而阵列 6c 支持 512 位加密。输入/输出端口 3 分别与每个阵列 6a、6b、6c 的第一处理部件通信，用于从也在操作中与端口 3 通信的客户站（未显示）接收要被流水线处理器 5 处理的数据。另外，逻辑电路 7 与输入/输出端口 3 以及与末位信号导体（conductor）9 进行电通信，该导体 9 以曲折的形式沿着每个阵列 6a、6b、6c 内的每个部件延伸。逻辑电路 7 用于接收指示加密密钥长度的数据、以及取决于所述数据而通过导体 9 向每个阵列 6a、6b、6c 内的每个部件提供信号。

现在参考图 3b，图 3b 显示了包括处理器部件 8^1、8^2、8^3、...、及 8^n 的串行阵列 6a（为简明起见省略了阵列 6b 和 6c）的简化框图。每个处理器部件 8 分别通过连接 11 与末位信号导体 9 进行电通信。作为特例，每个处理器部件是 8 位处理器部件，以致于串行阵列 6a 包括 32 个单独的处理器部件以便支持 256 位加密。因而，处理器资源的最有效分配需要 5 个单独的处理器部件来执行 40 位加密操作，8 个单独处理器部件来执行 64 位加密操作，以及 16 个单独处理器部件来执行 128 位加密操作等等。任选地，可以使用不同于 8 位处理器部件的处

理器部件。

说明性地考虑特定例子，例如需要至少 5 个单独的 8 位处理器部件来完成的 40 位加密操作。在使用中，客户站（未显示）通过端口 3 提供要加密的数据，例如作为总计包括完整数据文件的单个数据块的流。在第一处理周期开始时，阵列 6a 中的第一处理器部件 8^1 从端口 3 的缓冲存储器（未显示）接收第一数据块，并对第一数据块执行预定的第一阶段处理。在该例子中，第一阶段处理对应于利用加密密钥的 8 位段的加密操作。当然，第一处理器部件 8^1 与端口 3 的缓冲存储器（未显示），以及与逻辑电路 7 是时间同步的，以致数据块的流被同步地选通到第一处理器部件 8^1。在第二处理周期开始时，第一处理器部件 8^1 通过端口 3 接收第二数据块。近似同时地，第一处理器部件 8^1 以第一数据块的形式将输出沿正向处理路径提供给第二处理器部件 8^2。另外，第一处理器部件 8^1 把在其中计算的第一结果沿返回处理路径提供给端口 3 的缓冲存储器。这种处理数据的流水线方法继续，直到最终结果沿返回处理路径被提供给端口 3 的缓冲存储器为止。

有利地，数据块的流被同步地选通到第一处理器部件 8^1，如前面所述的。在每个处理周期开始时，逻辑电路 7 使指示加密密钥长度的数据递减预定量。在该例子中，加密密钥长度是 40，需要 5 个处理器部件来完成加密操作，因而指示加密密钥长度的数据代表 5 值。于是，在第一处理周期开始时，逻辑电路 7 使值 5 递减 1，以指示剩下 4 个处理周期。在第二至第五处理周期开始时，该值进一步被递减，此时逻辑电路 7 返回零值。如果指示加密数据已结束的零值产生了，则逻辑电路 7 通过末位信号导体 9 向每个处理器部件发送末位信号。刚好沿正向处理路径收到末位信号的处理器部件，在该情况下为第五处理器部件，立刻“知道”它是最末处理器部件、并使数据转向，以致数据不沿正向处理路径传播通过所述处理器部件。在最末处理器部件之前的每一个处理器部件在末位信号被发送时，既沿正向处理路径又沿返回处理路径接收数据，这表示不是最末部件状态。

当然，如果在特殊处理周期期间逻辑电路 7 的值达到非零值，则处理正常地继续。例如，在第二处理周期期间，第一处理器部件 8^1 对

第二数据块执行相同的第一处理操作，并且第二处理器部件 8^2 对第一数据块执行第二处理操作。在第二处理周期末尾，第一数据块沿着第二处理器部件 8^2 与第三处理器部件 8^3 之间的正向处理路径传播。同时，第二数据块沿着第一处理器部件 8^1 与第二处理器部件 8^2 之间的正向处理路径传播。另外，第二处理器部件 8^2 把在其中计算的结果沿着返回处理路径提供给第一处理器部件 8^1。当然，同时沿着相邻处理器部件之间的正向处理路径及返回处理路径选通数据块典型地涉及同步定时。

利用所示的双向流水线设计，高效地计算结果而与密钥长度无关，并且避免了附加的处理周期。用于支持不同长度密钥的处理器在设备中的使用同时支持多个高效加密过程一每个过程都具有最大的密钥大小。也就是说，在普通加密处理系统中，容易将统计学用于选择处理器大小，以便在统计学上为给定的资源使用提供最佳的性能。

作为替换，末位信号导体 9 只与处理器部件 8 的子集进行电连接。例如，在串行阵列 6a 中，末位信号导体 9 任选地连接到每第四个处理器部件。因此利用 8 位处理器部件，处理器 5 将加密数据处理为一系列的 32 位段。对于支持通过 32 位段处理的直到 256 位加密的处理器，支持 8 种可能的长度。有利的是，用于将末位信号从逻辑电路 7 引导至处理器部件的数据路进的数量从 32 减少到仅仅 8 个单独的数据路径，大大方便了处理器 5 的制造的容易。不幸的是，对于不能被 32 位除的加密密钥长度，例如在处理之前被填补到至少 64 位的 40 位加密密钥，处理资源以较低效率被分配。因而，数据在第八个处理器部件而不是在如上所讨论的第五个处理器部件被转向。

现在参考图 4a 和图 4b，图 4a 和 4b 显示了根据本发明第二实施例用于以相似的效率来处理任意密钥位长加密操作的流水线处理器 20 的简化框图。在此，与先前参考图 3a 和图 3b 描述的部件相同的部件具有相同的附图标记，并且为简洁起见省略了对它们的论述。例如包括一系列门（gate）的开关网络 21 另外与输入/输出端口 3 电通信，以及与一系列硬连线的（hardwired）可寻址数据路径 22 电通信。开关网络 21 用于接收指示加密密钥长度的数据，以及用于通过一系列硬

连线的可寻址数据路径 22 将信号提供给确定的处理部件。在使用中，指示加密密钥长度的数据被提供给开关网络 21，以致该一系列门确定指示最末处理部件状态的信号要发送给哪个处理器部件 8。开关网络 21 通过硬连线的可寻址数据路径 22 的选定数据路径将所述信号发送给确定的处理部件。例如，该信号用于将确定的处理部件的位设置为指示最末处理部件状态的值。有利地，当确定的处理部件完成数据加密处理时，该处理部件立即使数据转向，使得数据被端口 3 的缓冲存储器（未显示）读出。当然，任选地在开关网络 21 与处理部件的预定子集，例如每第四个处理部件之间提供数据路径。有利的是，数据路径的总数以及开关网络的复杂性减小了。不幸的是，对于不能被 32 位除的加密密钥长度，例如在处理之前被填补到至少 64 位的 40 位加密密钥，处理资源以较低效率被分配。

现在参考图 5a 和图 5b，图 5a 和 5b 显示了根据本发明第三实施例用于以相似的效率来处理任意密钥位长加密操作的流水线处理器 12 的简化框图。在此，与先前参考图 3a 和图 3b 描述的部件相同的部件具有相同的附图标记，并且为简洁起见省略了对它们的论述。根据本发明第三实施例，每个单独处理部件 14 都包括专用逻辑电路 15。在使用中，指示加密密钥长度的数据连同要加密的数据一起被提供给第一处理器部件 14^1。第一处理器部件 14^1 对第一数据块执行预定的第一阶段处理，并且另外使指示加密密钥长度的数据递减预定量。递减的指示加密密钥长度的数据连同第一数据块一起被提供给第二处理器部件 14^2。第二处理器部件 14^2 的逻辑电路接收该递减的指示加密密钥长度的数据，并使所述数据递减附加的预定量。如果第二处理器部件 14^2 的逻辑电路计算零值，则第二处理器部件 14^2 的逻辑电路在第二处理器部件 14^2 内部产生指示最末处理器部件状态的信号。近似与第二处理器部件 14^2 完成预定的第二阶段处理同时地，第二处理器部件 14^2 使数据转向，并且经过处理的数据被读出串行阵列 13a，并被读入端口 3 的缓冲存储器（未显示）。

参考图 6，图 6 显示了根据本发明第四优选实施例的流水线处理器 16 的简化框图。流水线处理器 16 包括多个处理器部件（未显示的

处理器部件）阵列 6a、6b 和 6c，例如阵列 6a 和 6b 的每一个都支持 256 为加密操作，而阵列 6c 支持 512 位加密操作。虚线 17a 和 17b 分别表示用于在阵列 6a 的最末处理部件与阵列 6b 的最末处理部件之间提供电通信的任选电耦合，以及用于在阵列 6b 的第一处理部件与阵列 6c 的第一处理部件之间提供电通信的任选电耦合。与每个阵列 6a、6b、6c 的第一处理部件通信的输入/输出端口 3 用于接收由也在操作中与输入/输出端口 3 通信的客户站（未显示）提供的数据，该数据要被阵列 6a、6b 和 6c 中的一个专门阵列处理。在此，提供了三个处理器，每个处理器具有支持的最大加密密钥大小，但是其中三个处理器可任选地被连接以形成一个 1024 位处理器。当然，也有可能装备任意长度的处理器，但是这常常招致大量的寻址开销，而这是不合乎需要的。当然，当希望最大限度的灵活性时，按照该实施例来把许多例如用于处理 64 位密钥的较小处理阵列连在一起。

因为指示密钥长度的数据连同待处理的数据及加密密钥一起被提供给处理器，因此处理器能够分配足够的处理单元给任务，并由此高效地分配资源。所示的处理器 16 具有逻辑电路 7，如先前参考图 3a 和图 3b 所讨论的情况一样。因而，在该实施例中，逻辑电路计算被处理器 16 处理的位数，将计数与指示加密密钥长度的数据进行比较，以及取决于指示加密数据结束的比较，来通过最末处理器信号导体 9 发送通用信号。任选地，使用用于指示最末处理器部件状态的其它系统，例如参考本发明第二和第三实施例之一所述的系统。

现在参考图 7，图 7 显示了根据本发明第五优选实施例的流水线处理器 18 的简化框图。流水线处理器 18 包括多个处理器部件（未显示的处理器部件）阵列 6a、6b 和 6c，例如阵列 6a 和 6b 的每一个都支持 256 为加密操作，而阵列 6c 支持 512 位加密操作。阵列 6a 的最末处理部件与阵列 6b 的最末处理部件通过硬件连接 19a 进行电通信，并且阵列 6b 的第一处理部件与阵列 6c 的第一处理部件通过硬件连接 19b 进行电通信。与阵列 6a 的第一处理部件通信的输入/输出端口 3 用于接收由也在操作中与输入/输出端口 3 通信的客户站（未显示）提供的数据，该数据要被阵列 6a、6b 和 6c 的串行配置处理。任选地，

提供单独的输入（未显示），以便直接将数据选通到至少除阵列 6a 的第一部件之外的处理器部件。

在此，阵列 6b 是双向的，并且因为流水线过程被实施为双向流水线过程，因此一旦阵列 6b 完成了相对于其另一个方向上发生的操作的处理，就有可能利用阵列 6b 的最末部件来开始。因而，大大地提高了效率。

因为指示密钥长度的数据连同待处理的数据及加密密钥一起被提供给处理器，因此处理器能够分配足够的处理单元给任务、并由此高效地分配资源。所示的处理器 18 具有逻辑电路 7，如先前参考图 3a 和图 3b 所讨论的情况一样。因而，在该实施例中，逻辑电路计算被处理器 18 处理的位数，将计数与指示加密密钥长度的数据进行比较，以及根据指示加密数据结束的比较，来通过最末处理器信号导体 9 发送通用信号。任选地，使用用于指示最末处理器部件状态的其它系统，例如参考本发明第二和第三实施例之一所述的系统。

图 6 和图 7 的流水线处理器 16 和 18 分别可以在以下模式下操作：其中使被选通到阵列 6a 的最末处理器部件的数据可以被阵列 6b 的最末处理器部件得到。例如，当某一特殊处理操作需要超过 256 个处理器部件时，通过在第二个不同阵列中继续该处理操作来增加处理器阵列的有效长度。当然，当某一特殊处理操作需要超过 512 个处理器部件时，通过在第三个不同阵列中继续该处理操作来增加处理器阵列的有效长度。例如，图 6 和图 7 中所示的流水线处理器中的任何一个都可以操作以便执行：利用一个阵列的 256 位加密；利用两个不同阵列的 512 位加密；以及利用所有 3 个不同阵列的 1024 位加密。

有利的是，因为知道处理器什么时候将完成处理，因此有可能指示那个处理器进行另一个处理器的下游的处理。例如，假定处理器 6a 具有用于处理 256 位加密操作的处理部件，并开始处理 256 位加密操作。假定 6b 是类似的处理器。如果，有时在处理部件 6a 开始处理之后并且在它被完成之前用于 512 位操作的处理请求到来了，则有可能在知道当数据被传播到处理阵列 6a 的最末部件时那个部件将完成当前处理中的处理工作的处理的处理部件 6b 上开始操作。这通过减少处

理器的停机时间同时等待其它处理器可以用来支持连起来的阵列处理，提高了整体系统性能。

基于 Montgomery 的加密数据流水线处理

应用 Montgomery 算法，模取幂的成本被降低到一系列的超长整数的加法。位避免乘法/加法体系结构中的进位传送，几种解决方法是周知的。这些解决方法将 Montgomery 算法与冗余基数系统或余数系统结合使用。

在 S.E.Eldridge and C.D.Walter. Hardware implementation of Montgomery's modular algorithm. IEEE Transactions on Computers, 42(6): 693-699, July 1993 一文中，Montgomery 模乘算法适于高效的硬件实施。由于更简单的组合逻辑，由更高的时钟频率获得了速度的增加。与基于 Brickell 算法的现有技术相比，报道了加速系数 2。

在 J.E.Vuillemin, P.Bertin, D.Roncin, M.Shand, H.H.Touati, and P.Boucard. Programmable active memories: Reconfigurable systems come of age. IEEE Transactions on VLSI Systems, 4(1): 56-59, March 1996 和 M.Shand and J.Vuillemin. Fast implementations of RSA cryptography. In Proceedings 11th IEEE Symposium on Computer Arithmetic, pages 252-259, 1993 中报道的 Research Laboratory of Digital Equipment Corp.（数字设备公司研究实验室），使用包括中国余数定理、异步进位完成加法器和开窗口取幂方法的几种加速方法的 16 个 XILINX 3090 FPGA（现场可编程门阵列）的阵列用于实施模取幂。该实施过程以 185kb/s 的速率计算 970 位 RSA 解密（每 970 位解密 5.2 ms），以及以超过 300kb/s 的速率计算 512 位 RSA 解密（每 512 位解密 1.7 ms）。该解决方法的缺点是，模数的二进制表示被硬连接到逻辑表示（logic representation）中，使得必须重新为体系结构配置每个新的模数（modulus）。

在Montgomery模乘算法中使用高基数的问题是更复杂地确定商。该行为造成了算法流水线操作不直接进行。在 H.Orup. Simplifying quotient determination in high-radix modular multiplication. In

Proceedings 12th Symposium on Computer Arithmetic, pages 193-9, 1995 中，算法被重写以避免在确定商的过程中涉及的任何操作。只为给定的模数执行一次必需的预先计算。

P.A.Wang 在 New VLSI architectures of RSA public key crypto systems. In Proceedings of 1997 IEEE International Symposium on Circuits and Systems, volume 3, pages 2040-3, 1997 文章中提出了用于 Montgomery 模乘算法的新 VLSI 体系结构。确定时钟速度的关键途径是流水线。这是通过使算法的每次迭代交错来实现的。与先前的提议相比，报道了系数 2 的时间面积乘积的改进。

J.Bajard, L.Didier, and P.Kornerup 在 An RNS Montgomery modular multiplication algorithm. IEEE Transactions on Computers, 47(7): 766-76, July 1998 文章中描述了一种使用余数系统（RNS）的新方法。在 n 个相当简单的处理器商利用 RNS 中的 n 个模数来实施该算法。由此引起的处理时间为 O(n)。

当然，以上引用的参考文献的大多数都涉及几乎没有或没有灵活性的处理器硬件实施。

也有许多用于模运算的脉动阵列体系结构的提议。这些提议在复杂性和灵活性方面不同。

在 E.F.Brickell. A survey of hardware implementations of RSA. In Advances in Cryptology—CRYPTO'89, pages 368-70. Springer-Verlag, 1990 文章中，E.F.Brickell 总结了 1990 年可得的用于执行 RSA 加密的芯片。

在 N.Takagi. A radix-4 modular multiplication hardware algorithm efficient for iterative modular multiplication operations. In Proceedings 10th IEEE Symposium on Computer Arithmetic, pages 35-42, 1991 文章中，作者提出了一种基 4（radix-4）硬件算法。冗余数被使用，并由此避免了加法中的进位传送。与先前工作相比，报道了大约 6 倍的处理加速。

更近一些，提出了这样一种方法，该方法利用预先计算的模数的补数，并且基于 J.Yong-Yin and W.P.Burleson. VLSI array algorithms and

architectures for RSA modular multiplication. IEEE Transactions on VLSI Systems, 5(2): 211-17, June 1997 文章中的迭代 Horner 规则。与 Montgomery 算法相比，这些方法利用中间结果的最高有效位来决定要减去模数的哪些乘。这些解决方法的缺点是，它们或者需要大量存储空间、或者需要许多时钟周期，来完成模乘。

最流行的模取幂算法是自乘与乘（square & multiply）算法。公开密钥加密系统典型地基于模取幂或重复的点加法（point addition）。这两种操作是由自乘与乘（square & multiply）算法执行的最基本形式。

方法 1.1 计算 $Z=X^E \bmod M$，其中 $E=\sum_{i=0}^{n-1} e_i 2^i, e_i \in \{0,1\}$

1. Z=X
2. FOR i=n-2 down to 0 DO
3. $Z=Z^2 \bmod M$
4. IF e_i=1 THEN $Z=Z \cdot X \bmod M$
5. END FOR

方法 1.1 在最坏情况下需要 2(n-1)次操作，并且平均需要 1.5(n-1)次操作。为并行地计算自乘和乘法，可以使用以下型式的自乘与乘（square & multiply）方法：

方法 1.2 计算 $P=X^E \bmod M$，其中 $E=\sum_{i=0}^{n-1} e_i 2^i, e_i \in \{0,1\}$

1. P_0=1, Z_0=X
2. FOR i=1 to n-1 DO
3. $Z_{i+1}=Z_i^2 \bmod M$
4. IF e_i=1 THEN $P_{i+1}=P_i \cdot Z_i \bmod M$
   ELSE $P_{i+1}=P_i$
5. END FOR

方法 1.2 在最坏情况下需要 2n 次操作，并且平均需要 1.5n 次操作。通过应用 l-ary 方法，如 D.E.Knuth, The Art of Computer Programming. Volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981 中公开的作为方法 1.1 的一般化的 l-ary

方法，来实现加速。l-ary 方法每次处理 l 阶位。在此的缺点是必须预先计算并存储 X 的(2^l-2)次乘。减少到 2^{l-1} 次预先计算是有可能的。最后所得的复杂性粗略地为 n/l 次乘法操作和 n 次自乘操作。

如上所示，利用 Montgomery 方法将模取幂简化为一系列的模乘操作和自乘步骤。P.L.Montgomery 在 P.L.Montgomery. Modular multiplication without trial division. Mathematics of Computation, 44(170): 519-21, April 1985 文章中提出了用于下述模乘的方法。该方法是使两个整数相乘并以 M 为模、同时避免除以 M 的方法。其思想是将整数变换成 m 余数，并计算这些 m 余数的乘法。最后，表示被变换回到其正规表示。只有当计算变换域中的一系列乘法操作（例如模取幂）时，该方法才有益。

为计算 Montgomery 乘法，基数 R>M，并选择 gcd(M,R)=1。除以 R 优选地是计算量小的，因而如果 $M=\sum_{i=0}^{m-1}m_i 2^i$ 、则最佳的选择是 $R=2^m$。x 的 m 余数是 xR mod M。也计算 $M'=M^{-1} \bmod R$。提供函数 MRED(T)来计算 $TR^{-1} \bmod M$。假定 T 是 m 余数，该函数计算 T 的正规表示。

方法 1.3 MRED(T)：计算 T 的 Montgomery 简化

$T<RM$, $R=2^m$, $M=\sum_{i=0}^{m-1}m_i 2^i$, gcd(M,R)=1

1. $U=TM' \bmod R$

2. $t=(T+UM)/R$

3. IF $t \geq M$ return t-M

   ELSE return t

MRED(T)的结果是 $t=TR^{-1} \bmod M$。

现在使两个整数 a 和 b 在变换域中相乘，其中它们各自的表示是(aR mod M)和(bR mod M)，将这两个表示的乘积提供给 MRED(T)。

$MRED((aR \bmod M)\cdot(bR \bmod M)) = abR^2R^{-1} = abR \bmod M$

对于模取幂，依据方法 1.1 或 1.2 重复该步骤许多次，以便获得最终结果 ZRmod M 或 $P_nR \bmod M$。将这些值之一提供给 MRED(T)，以获得 $Z \bmod M$ 或 $P_n \bmod M$。

初始变换步骤仍然需要代价高的模简化，为避免涉及除法，利用

除法计算 $R^2 \bmod M$。只需要为给定密码系统执行一次该步骤。为获得变换域中的 a 和 b，执行 MRED($a \cdot R^2 \bmod M$)和 MRED($b \cdot R^2 \bmod M$)，以获得 $aR \bmod M$ 和 $bR \bmod M$。显然，可以照这样变换任何变量。

对于方法 1.3 的硬件实施：利用 m×m 位乘法和 2m 位加法来计算步骤 2。中间结果可以具有和 2m 一样多的位。不是立刻计算 U，而是每次计算 r 基表示的一个数位。选择基数 r，使得 gcd(M,r)是优选的。除以 r 优选地也是计算量小的，因而最佳选择是 $r=2^k$。现在，所有变量都以 r 基表示来表示。另一个改进是在算法中包括乘法 A×B。

方法 1.4 用于计算 A·B mod M 的 Montgomery 模乘，其中

$$M=\sum_{i=0}^{m-1}(2^k)^i m_i, m_i \in \{0,1,...,2^k-1\}; B=\sum_{i=0}^{m-1}(2^k)^i b_i, b_i \in \{0,1,...,2^k-1\};$$

$$A=\sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0,1,...,2^k-1\};$$

$$A,B<M; M<R=2^{km}; M'=-M^{-1} \bmod 2^k; \gcd(2^k,M)=1$$

1. $S_0=0$

2. FOR i=0 to m-1 DO

3. $q_i=(((S_i+a_iB) \bmod 2^k)M') \bmod 2^k$

4. $S_{i+1}=(S_i+q_iM+a_iB)/2^k$

5 END FOR

6. IF $S_m \geq M$ 返回 S_m-M

ELSE 返回 S_m

应用方法 1.4 的结果是 $S_m=ABR^{-1} \bmod M$。对于基数 2^k，需要最多两个 k×k 位乘法操作和 k 位加法来计算步骤 3。对于步骤 4，需要两个 k×m 位乘法操作和两个 m+k 位加法。与方法 1.3 的 2m 位相比，S 的最大位长被减小到 m+k+2 位。

方法 1.5 是相对于 r=2 的方法 1.4 的简化。对于基数 r=2，对方法 1.4 的步骤 3 中的操作执行模 2。由于条件 $\gcd(M,2^k)=1$，使得模数 M 为奇数。立即得出，$M=1 \bmod 2$。因此，$M'=-M^{-1} \bmod 2$ 也退化为 M'=1。因而，任选地省略步骤 3 中的乘 M' mod 2。

方法 1.5 用于计算 A·B mod M 的 Montgomery 模乘（基数 r=2），

其中

$M=\sum_{i=0}^{m-1}(2^k)^i m_i, m_i \in \{0,1\}; B=\sum_{i=0}^{m-1}(2^k)^i b_i, b_i \in \{0,1\};$

$A=\sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0,1\}; A,B<M; M<R=2^m; \gcd(2,M)=1$

1. $S_0=0$

2. FOR i=0 to m-1 DO

3. $q_i=(S_i+a_iB) \bmod 2$

4. $S_{i+1}=(S_i+q_iM+a_iB)/2$

5 END FOR

6. IF $S_m \geq M$ 返回 S_m-M

ELSE 返回 S_m

方法 1.5 的步骤 6 中的最终比较和减法执行起来是高代价的，因为 m 位比较是很慢的，并且在资源使用方面是高花费的。它也使算法的流水线执行成为不可能。可以容易地验证，如果 A,B<M，则 $S_{i+1}<2M$ 总是保持。然而，S_m 不能重新用作下一次模乘的输入 A 或 B。如果在 $a_{m+1}=0$ 且输入 A,B<2M 的情况下来执行 for 循环的两次多余运行，则满足不等式 $S_{m+2}<2M$。现在，S_{m+2} 可以用作下一次模乘的输入 B。

为进一步减小方法 1.5 的复杂性，使 B 上移一个位置，即使 B 乘以 2。这导致 $a_i \cdot B \bmod 2=0$，并且避免了步骤 3 中的加法。在 S_{i+1} 的更新期间，用 $(S_i+q_iM)/2+a_iB$ 来代替 $(S_i+q_iM+a_iB)/2$。该简化的成本是在 $a_{m+2}=0$ 的情况下多运行一次循环。以下方法包括这些优化。

方法 1.6 用于计算 A·B mod M 的 Montgomery 模乘（基数 r=2），其中

$M=\sum_{i=0}^{m-1}(2^k)^i m_i, m_i \in \{0,1\}; B=\sum_{i=0}^{m-1}(2^k)^i b_i, b_i \in \{0,1\};$

$A=\sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0,1\}; A,B<2M; M<R=2^{m+2}; \gcd(2,M)=1$

1. $S_0=0$

2. FOR i=0 to m+2 DO

3. $q_i=S_i \bmod 2$

4. $S_{i+1}=(S_i+q_iM)/2+a_iB$

5 END FOR

以上算法计算 $S_{m+3}=(2^{-(m+2)}AB) \bmod M$。为得到正确的结果，执行额外的 Montgomery 模乘 $2^{2(m+2)} \bmod M$。然而，如果如在取幂算法中一样需要进一步的乘法操作，最好使所有的输入预先乘以系数 $2^{2(m+2)} \bmod M$。因而，每个中间结果都携带系数 $2^{2(m+2)}$。使结果与“1”进行 Montgomery 相乘来消除该系数。

与“1”的最终 Montgomery 乘确保了最终的结果小于 M。

高基数 Montgomery 算法

通过避免代价高的步骤 6 的比较和减法操作、并将条件改变为 $4M<2^{km}$ 且 $A,B<2M$，产生了相对于在硬件中实施方法 1.4 的某些优化。其代价是多运行两次循环。最终所得的方法如下：

方法 1.7 用于计算 A·B mod M 的 Montgomery 模乘，其中

$M=\sum_{i=0}^{m-3}(2^k)^i m_i, m_i \in \{0,1,...,2^k-1\};$

$\tilde{M}=(M' \bmod 2^k)M, \tilde{M}=\sum_{i=0}^{m-2}(2^k)^i \tilde{m}_i, \tilde{m}_i \in \{0,1,...,2^k-1\};$

$B=\sum_{i=0}^{m-1}(2^k)^i b_i, b_i \in \{0,1,...,2^k-1\}; A=\sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0,1,...,2^k-1\};$

$A,B<2\tilde{M}; 4\tilde{M}<2^{km}; M'=-M^{-1} \bmod 2^k$

1. $S_0=0$

2. FOR i=0 至 m-1 DO

3. $q_i=(S_i+a_iB) \bmod 2^k$

4. $S_{i+1}=(S_i+q_i\tilde{M}+a_iB)/2^k$

5 END FOR

通过用 B·2k 代替 B，进一步减小商 q_i 确定的复杂性。因为 $a_iB \bmod 2^k=0$，步骤 3 被简化为 $q_i=S_i \bmod 2^k$。以附加的循环迭代为代价来避免步骤 3 中的加法，以补偿 B 中的额外系数 2^k。为硬件实施而优化的 Montgomery 方法如下所示：

方法 1.8 用于计算 A·B mod M 的 Montgomery 模乘，其中

$M=\sum_{i=0}^{m-3}(2^k)^i m_i, m_i \in \{0,1,...,2^k-1\};$

$\tilde{M}=(M' \bmod 2^k)M, \tilde{M}=\sum_{i=0}^{m-2}(2^k)^i \tilde{m}_i, \tilde{m}_i \in \{0,1,...,2^k-1\};$

$B=\sum_{i=0}^{m-1}(2^k)^i b_i, b_i \in \{0,1,...,2^k-1\}; A=\sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0,1,...,2^k-1\}, a_m=0;$

$A,B<2\tilde{M}; 4\tilde{M}<2^{km}; M'=-M^{-1} \bmod 2^k$

1. $S_0=0$

2. FOR i=0 to m-1 DO

3. $q_i=S_i \bmod 2^k$

4. $S_{i+1}=(S_i+q_i\tilde{M})/2^k+a_iB$

5 END FOR

然后，使最终结果与“1”进行 Montgomery 相乘，以消除如上所述的其中的系数。

在此处被引入作为参考的、Thomas Blum 于 1999 年 4 月 8 日提交给 Faculty of the Worcester Polytechnic Institute 的题为 Modular Exponentiation on Reconfigurable Hardware 的论题中，Thomas Blum 提出了两种不同的用于利用模乘和 Montgomery 空间执行加密功能的流水线体系结构：基于方法 1.6 的面积高效体系结构和速度高效体系结构。使用 Xilinx XC4000 系列设备作为目标设备。

一般 2 基脉动阵列使用 m×m 处理部件，其中 m 为模数的位数，并且每一部件处理一位。可以同时处理 2m 个模乘操作，以每时钟周期一个模乘的处理量以及 2m 个周期的等待时间为特色。因为对于现代公开密钥方案中所需的典型位长、该方法导致了不实际的大 CLB 计数，因此只实施一行处理部件。利用该方法，可以同时处理 2 个模乘操作，而执行降低到每 2m 个周期 2 个模乘操作的处理量。等待时间保持为 2m 周期。

第二考虑事项是基数 $r=2^k$ 的选择。增大 k 将减少方法 1.8 中要执行的步骤的数量。然而，这种方法需要更多的资源。主要的花费在于 2^k 与 M 和 B 相乘的计算。这些或者被预先计算并存储在随机存储器（RAM）中，或者被多路复用器网络计算。无疑，对于 r=2、CLB 基数变成最小，因为不必计算或预先计算 M 或 B 的乘。

利用基数 r=2，来计算依据方法 1.6 的方程式。为进一步减少所需的 CLB 数量，任选地采取以下措施：每个单元处理多于一位。单个加法器用于预先计算 B+M，以及执行正常处理期间的其它加法操作。并行地计算自乘和乘法操作。该设计按等级被分为三个级别。

处理部件：计算模乘的 u 位。

模乘：处理部件阵列计算模乘

模取幂：将模乘操作与根据算法 1.2 的模取幂相结合

处理部件

图 8 显示了处理部件的实施过程。

在处理部件中，存在以下寄存器：

- M-Reg（u 位）：存储模数
- B-Reg（u 位）：存储 B 乘数
- B+M-Reg（u 位）：存储中间结果 B+M
- S-Reg（u+1 位）：存储中间结果（包括进位）
- S-Reg-2（u-1 位）：存储中间结果
- Control-Reg（3 位）：控制多路复用器和时钟使能
- a_i、q_i（2 位）：乘数 A、商 Q
- Result-Reg（u 位）：存储乘法最后的结果

寄存器需要总共(6u+5)/2 个 CLB，加法器需要 u/2+2 个 CLB，多路复用器需要 4·u/2 个 CLB，以及解码器需要 2 个 CLB。将寄存器重新用于组合逻辑的可能性允许 CLB 的某些节省。将 Mux_B 和 Mux_{Res} 实施在 B-Reg 和 Result-Reg 的 CLB 中，将 Mux_1 和 Mux_2 部分地实施在 M-Reg 和 B+M-Reg 中。最终所得的成本是大约每 u 位处理单元 3u+4 个 CLB。即，每位 3 到 4 个 CLB，这取决于单元大小 u。

在单元能够计算模乘之前，必须加载系统参数。将 M 存储到单元的 M-Reg 中。在模乘开始时，根据多路复用器 B-Mux 的选择线，从 B-in 或 S-Reg 加载操作数 B。下一步是计算一次 M+B，并将结果存储在 B+M-Reg 中。该操作需要两个时钟周期，因为结果首先随着时钟进入到 S-Reg 中。分别通过 a_i 或控制字来控制 Mux_1 和 Mux_2 的选择线。

在接着的 2(m+2)个周期中，根据方法 1.6 来计算模乘。多路复用器 Mux_1 根据二进制变量 a_i 和 q_i 的值，来选择要馈入加法器中的其输入 0、M、B、B+M 之一。Mux_2 将先前结果 $S\text{-}Reg_2$ 的 u-1 个最高有效位加上下一单元（除以 2/右移）的最低有效结果位馈入加法器的第二输入端。将结果存储在 S-Reg 中一个周期。最低有效位进入向右的单

元（除以 2/右移），而进位进入向左的单元。在该周期中，利用更新的 S-Reg_2、a_i、q_i 值在加法器中计算第二模乘。第二模乘使用相同的操作数 B，但是不同的操作数 A。

在模乘的末尾，在加法器的输出端处，S_{m+3} 在一个周期内有效。将该值存储到 Result-Reg 中，如通过 S-Reg 馈入 B-Reg 中一样。在一个周期之后将第二乘法的结果馈入 Result-Reg 中。

图 9 显示了处理部件怎样连接到用于计算 m 位模乘的阵列。为执行相对于 m 的方法，使用每单元处理 u 位的 m/u+1 个单元。$Unit_0$ 只有 u-1 个 B 输入，因为 B_0 被加到移位值 S_i+q_iM 上。根据 Montgomery 算法的特性，结果位 S-Reg_0 总是为 0。$Unit_{m/u}$ 处理 B 的最高有效位以及中间结果 S_{i+1} 的临时溢出。没有到该单元的 M 输入。

单元的输入和输出以以下方式相互连接。控制字、q_i 和 a_i 被从右向左泵送（bumped）通过单元。结果从左向右泵送。carry-out（载出）信号被馈给向右的 carry-in（载入）输入。输出 S_0_Out 总是连接到向右的单元的输入 S_0_In。这代表方程式除以 2。

首先将模数 M 馈入单元中。为允许信号有足够的时间传播到所有单元，M 在两个时钟周期内有效。我们使用两个 M-Bus，M-even-Bus 连接到所有的偶数单元、M-odd-Bus 连接到所有的奇数单元，该方法允许每时钟周期将 u 位馈给单元。因而，需要 m/u 个周期来加载完整模数 M。

类似地加载操作数 B。信号也在两个时钟周期内有效。在加载操作数 B 之后，方法 1.6 的步骤的执行就开始了。

从最右边的单元 $unit_0$ 开始，将控制字、a_i 和 q_i 馈入它们的寄存器中。加法器根据 a_i 和 q_i、在一个时钟周期内计算 S-Reg-2 加上 B、M 或 B+M。读回结果的最低有效位作为下一次计算的 q_{i+1}。将所得到的进位位、控制字、a_i 和 q_i 泵入向左的单元中，其中在下一个时钟周期中发生相同的计算。

以这种脉动的形式，将控制字、a_i、q_i 和进位位从右向左泵送通过整个单元阵列。方法 1.6 中的除以 2 也导致了右移操作。单元的加法（S_0）的最低有效位总是被反馈到向右的单元中。在完成模乘之后，

将结果从左向右泵送通过单元、并连续地存储在 RAM 中用于进一步处理。

单个处理部件计算 $S_{i+1}=(S_i+q_i\cdot M)/2+a_i\cdot B$ 的 u 位。在时钟周期 i 中，$unit_0$ 计算 S_i 的位 0...u。在时钟周期 i+1 中，$unit_1$ 使用所得的进位、并计算 S_i 的位 u...2u。$unit_0$ 在时钟周期 i+2 中、利用 $S_i(S_0)$的右移（除以 2）位 u 来计算 S_{i+1} 的位 0...u-1。在单元 $unit_0$ 中时钟周期 i+1 是非生产性的、同时等待 $unit_1$ 的结果。通过依据方法 1.2 并行地计算自乘和乘法操作，来避免这种低效率。p_{i+1} 和 z_{i+1} 都取决于 z_i。因此，将中间结果 z_i 存储在 B-Register 中，并将 z_i 和 p_i 一起馈入单元的 a_i 输入中用于自乘和乘法。

图 10 显示了单元阵列怎样用于模取幂。在设计的核心是具有以下 17 种状态的有限状态机（FSM）：空闲状态，用于加载系统参数的 4 种状态，以及用于计算模取幂的 4×3 种状态。实际的模取幂是在预先计算 1、预先计算 2、计算和计算后 4 种主要状态下执行的。这些主要状态的每一种都细分为 3 种子状态：load-B、B+M 和计算乘法。根据状态来对馈入 control-in 的控制字编码。以二分之一时钟频率对 FSM 计时。同样对加载和读取 RAM 和 DP RAM 元件也成立。该措施确保最大传播时间是在单元中。因而，最小时钟周期时间和最终所得的模取幂的速度与单元中的有效计算时间有关，而与计算开销无关。

在计算模取幂之前，加载系统参数。从 I/O 将模数 M 的 2u 位读入 M-Reg 中。从低数位开始向高数位读。从 M-Reg 将 M 的 u 位交替地馈给 M-even-Bus 和 M-odd-Bus。每次信号在两个周期内有效。当时，从 I/O 读取指数 E 的 16 位、并存储到 Exp-RAM 中。来自 I/O 的最初 16 位宽的字指定了指数的位长。直到 64 个后面的字包含了实际的指数。当时，从 I/O 读出预先计算系数 $2^{2(m+2)}$ mod M 的 2u 位。将该 2u 位存储到 Prec-RAM 中。

在预先计算 1 状态下，我们从 I/O 读取 X 值，每个时钟周期 u 位，并将它存储到 DP RAM Z 中。同时，从 Prec-RAM 读出预先计算系数 $2^{2(m+2)}$ mod M、并且每时钟周期将 u 位交替地通过 B-even-Bus 和 B-odd-Bus 馈给单元的 B-Register。在接下来的两个时钟周期中，在单

元中计算 B+M。

用于方法 1.2 的初值是可得的。两个值都必须乘以 2，这可以被并行地执行，因为两个乘法操作都使用已经存储在 B 中的公共操作数 $2^{2(m+2)} \bmod M$。时分复用（TDM）单元从 DP RAM Z 读出 X，并将 X 与 1 复用。在 2(m+3)个时钟周期之后，结果的低数位出现在 Result-Out，并被存储在 DP RAM Z 中。在一个周期之后，下一个结果的低数位出现在 Result-Out，并被存储在 DP RAM P 中。该过程重复 2m 个周期，直到两个结果的所有数位都被存储在 DP RAM Z 和 DP RAM P 中为止。结果 $X \cdot 2^{m+2} \bmod M$ 也被存储在单元的 B-Register 中。

在预先计算 2 状态下，方法 1.2 的实际步骤开始。对于 Z1 和 P1 两者的计算，都将 Z0 用作操作数。该值被存储在 B-Register 中。第二操作数 Z0 或 P0 分别从 DP RAM Z 和 DP RAM P 被读出，并作为 a_i、通过 TDM 被“泵”入单元中。在另外的 2(m+3)个时钟周期之后，Z1 和 P1 的结果的低数位出现在 Result-Out。Z1 被存储在 DP RAM Z 中。只有当指数 e0 的第一位等于“1”时，才需要 P1。取决于 e0，P1 被存储在 DP RAM P 中、或者被丢弃。

在计算状态下，方法 1.2 的循环被执行 n-1 次。在每个周期之后 DP RAM Z 中的 Z_i 都被更新，并作为 a_i 被“泵”回到单元中。只有当指数 e_i 的相关位等于“1”时，才更新 DP RAM P 中的 P_i。这样，总是最后存储的 P 被“泵”回到单元中。

在 e_{n-1} 的处理之后，FSM 进入计算后状态。为了从结果 P_n 消除系数 2m+2，计算最终的 Montgomery 乘 1。首先交替地通过 B-even-Bus 和 B-odd-Bus、将向量 0,0,...0,1 馈入单元的 B-Register 中。从 DP RAM P 将 P_n 作为 a_i 泵入单元中。在执行计算后状态之后，结果 $P_n=XE \bmod M$ 的 u 位在 I/O 端口处有效。每两个时钟周期另外的 u 位出现在 I/O 处。现在可以立即重新进入预先计算 1 状态，以计算另一 X 值。

在 2(n+2)(m+4)个时钟周期内计算完整的模乘。那是从把 X 的最初 u 位插入设备中，直到最初的 u 个结果位出现在输出处为止的延迟。在那一点上，另一个 X 值可以进入设备中。在附加的 m/u 个时钟周期的等待时间之后，最后的 u 位出现在输出总线上。

以下，说明图 10 中的功能块。图 11 显示了 DP RAM Z 的设计。m/u×u 位 DP RAM 在该单元的核心。它具有分开的写（A）和读（DPRA）地址输入。计数直到 m/u 的写计数器计算写地址（A）。当 Z_i 的最初 u 位出现在 data-in 时，写计数器在 B-load 子状态下开始计数（时钟使能）。同时，DP RAM 的使能信号有效，并且数据被存储在 DP RAM 中。当达到 m/u 时，终端计数（terminal-count）使 DP RAM 的计数使能和写使能复位。在计算子状态下，读计数器被允许操作。当读计数器达到其上限 m+2 时，终端计数（terminal-count）触发 FSM 转换为子状态 B-load。读计数器值（q-out）的 $\log_2(m/u)$最高有效位对 DP RAM 的 DPRA 寻址。每 u 个周期读出 DP RAM 中存储的另一个值。当 q-out 的 $\log_2(u)$最低有效位达到 0 时，将该值加载到移位寄存器中。在接下来的 u 个周期，u 位逐位地出现在移位寄存器的串行输出处。Z_i 的最后值被存储在 u 位寄存器中。该办法允许我们选择 m/u×u 位 DP RAM，而不是 2m/u×u 位 DP RAM（m=2x，x=8,9,10）。

DP RAM P 几乎以同样的方式工作。它具有附加的输入 e_i，在 e_i=1 的情况下，e_i 激活 DP RAM 的写使能信号。

图 12 显示了 Exp RAM 的设计。在加载指数状态（load-exponent state）的第一周期中，从 I/O 读出第一个字，并存储到 10 位寄存器中。它的值指定了指数的位长。在后续的周期中，每次读出指数的 16 位、并存储在 RAM 中。通过 6 位写计数器来计算存储地址。在每一个计算状态开始时，10 位读计数器都被允许操作。它的 6 个最高有效位计算存储器地址。因而，每第 16 次激活，就从 RAM 读出新的值。在读计数器的 4 个最低有效位等于 0 的同时，将该值存储在 16 位移位寄存器中。当读计数器达到 10 位寄存器中指定的值时，终端计数（terminal-count）信号触发 FSM 进入计算后状态。

图 13 显示了 Prec RAM 的设计。在加载预先系数状态下，当时从 I/O 读出预先计算系数的 2u 位、并存储在 RAM 中。计数直到 m/2u 的计数器对 RAM 寻址。当所有的 m/2u 个值都被读出时，终端计数（terminal-count）信号触发 FSM 离开加载预先系数状态。

在预先计算 1 状态下，从 RAM 读出预先计算系数，并馈给单元

的 B-Register。每个时钟周期都递增计数器，并将 2u 位加载到 2u 位寄存器中。每个时钟周期的正沿从那里将 u 位馈在 B-even-Bus 上。在负时钟边沿，将 u 位馈在 B-odd-Bus 上。

速度高效体系结构

以上设计是在资源使用方面进行优化的。利用基数 $r=2^k$，$k>1$，将使方法 1.6 中的步骤数量减少 k 倍。方法 1.8 的计算倍执行 m+3 次（i=0 至 m+2）。

很容易按等级将速度高效设计分为 3 个级别。

处理部件：计算模乘的 4 位

模乘：处理部件阵列计算模乘

模取幂：将模乘操作与根据方法 1.2 的模取幂相结合

图 14 显示了处理部件的实施。

提供了以下元件：

- B-Reg（4 位）：存储 B 乘数
- B-Adder-Reg（5 位）：存储 B 的倍数
- S-Reg（4 位）：存储中间结果 S_i
- Control-Reg（3 位）：控制多路复用器和时钟使能
- a_i-Reg（4 位）：乘数 A
- q_i-Reg（4 位）：商 Q
- Result-Reg（4 位）：存储在乘法末尾的结果
- B-Adder（4 位）：将 B 加到先前计算的 B 的倍数上
- B+M‾-Adder（4 位）：将 M‾的乘加到 B 的倍数上
- S+B+M‾-Adder（5 位）：将 M‾S_i 加到 B+M‾-Adder 上
- B-RAM（16×4 位）：存储 B 的 16 倍数
- M‾-RAM（16×4 位）：存储 M‾的 16 倍数

从以上参考的 T.Blum 的论题，以及从图的回顾，单元的操作是显然的。

图 15 显示了处理部件怎样连接到用于计算全尺寸模乘的阵列。

图 16 显示了单元阵列怎样用于模取幂。

图 17 显示了 DP RAM Z 的设计。m×4 位 DP RAM 位于该单元的核心。它具有分开的写（A）和读（DPRA）地址输入。计数直到 m+2 的两个计数器计算这些地址。当 Z_i 的第一数位出现在 data-in 时，写计数器在 B-load 子状态下开始计数（时钟使能）。同时，DP RAM 的使能信号有效，并且数据被存储在 DP RAM 中。当达到 m+2 时，写计数器的终端计数（terminal-count）信号使两个使能信号复位。在计算子状态下读计数器被允许操作。DP RAM 的数据被读计数器的 q-out 寻址、并立即出现在 DPO 处。当读计数器达到 m+2 时，终端计数（terminal-count）触发 FSM 转变为 B-load 子状态。最后两个 z_i 值的每一个都被存储在 4 位寄存器中。

该办法允许我们选择 100%利用的 m×4 位 DP RAM，而不是仅仅 50%利用的 2m×4 位 DP RAM。DP RAM P 几乎以同样的方式工作。它具有附加的输入 e_i，在 e_i=“1”的情况下，e_i 激活 DP RAM 的写使能信号。

因为以上的流水线处理器体系结构体现了许多流水线处理部件，所以常常难以、并且代价高地在同一集成电路内使每个部件与时钟脉冲源同步。因此，本发明的极其有利之处在于，通过简化时钟分配问题来减少总的资源需求。而且，因为在一个方向上需要加法，而在另一个方向上需要乘法，所以显然沿一条路径所需的时间比沿另一条路径所需的时间多，因此根据本发明的实施例，路径的时间平均是有可能的。

在不背离本发明的精神和范围的情况下，可以设想大量的其它实施例。

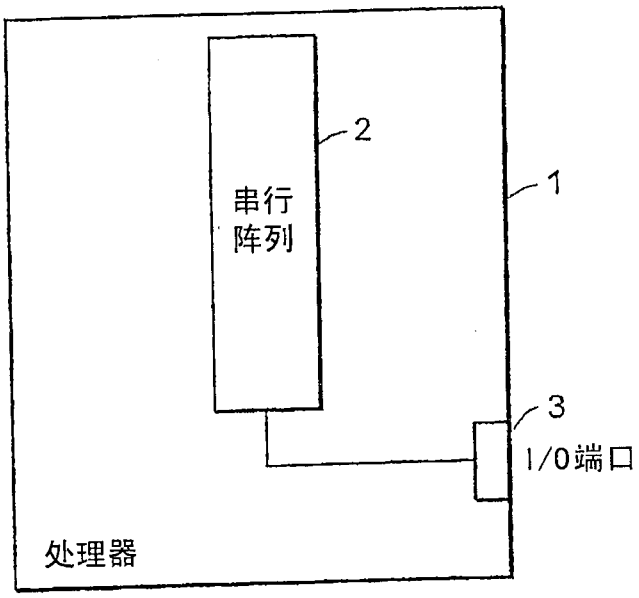


图 1

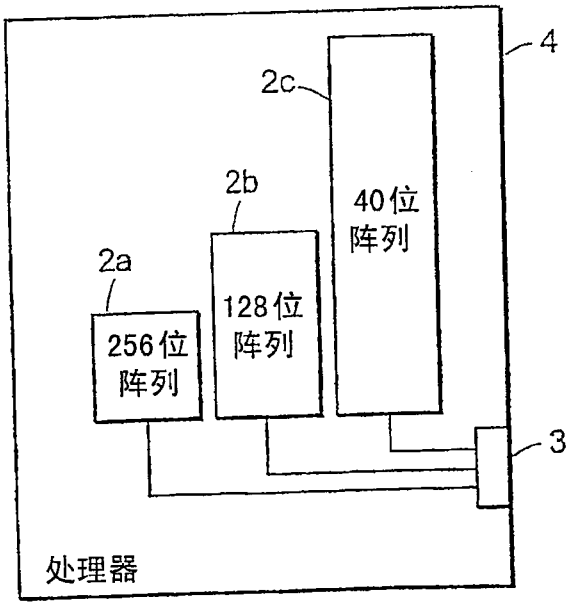


图 2

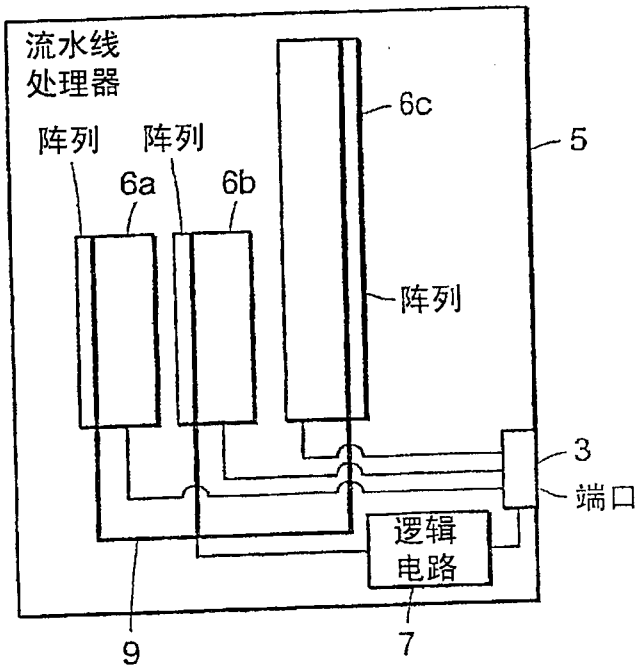

流水线
处理器
阵列
阵列
6a
6b
6c
5
阵列
3
端口
逻辑
电路
9
7

图 3A

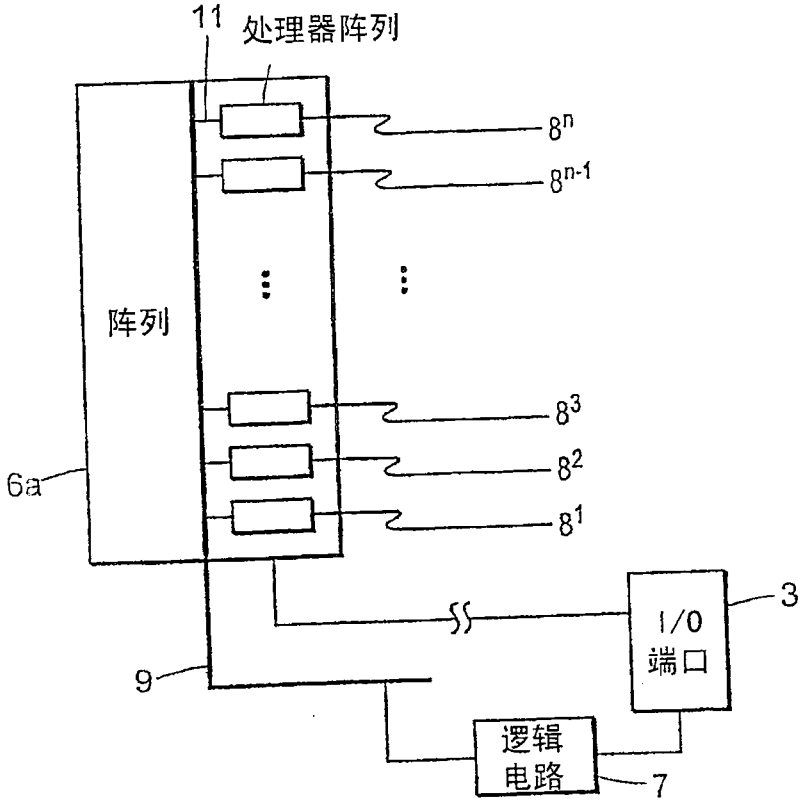

11
处理器阵列
8^n
8^{n-1}
阵列
8^3
8^2
8^1
6a
I/O
端口
3
9
逻辑
电路
7

图 3B

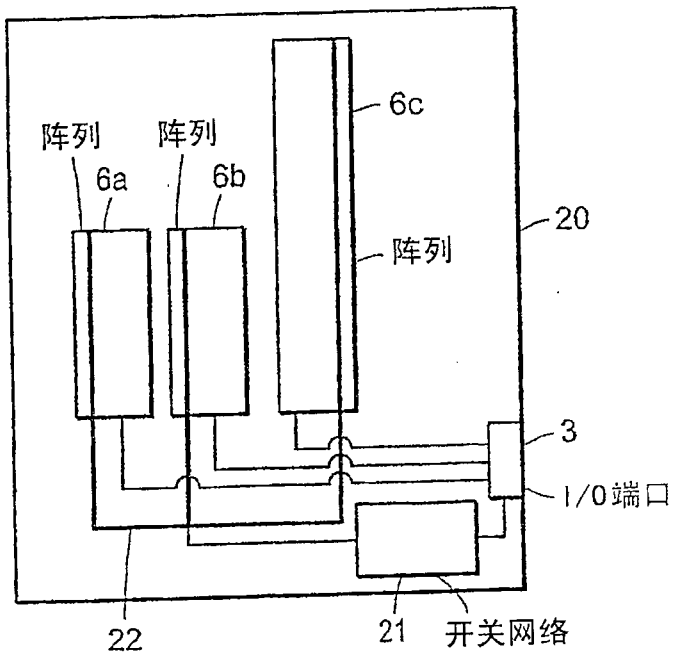

阵列
阵列
6a
6b
6c
阵列
20
3
I/O端口
22
21
开关网络

图 4A

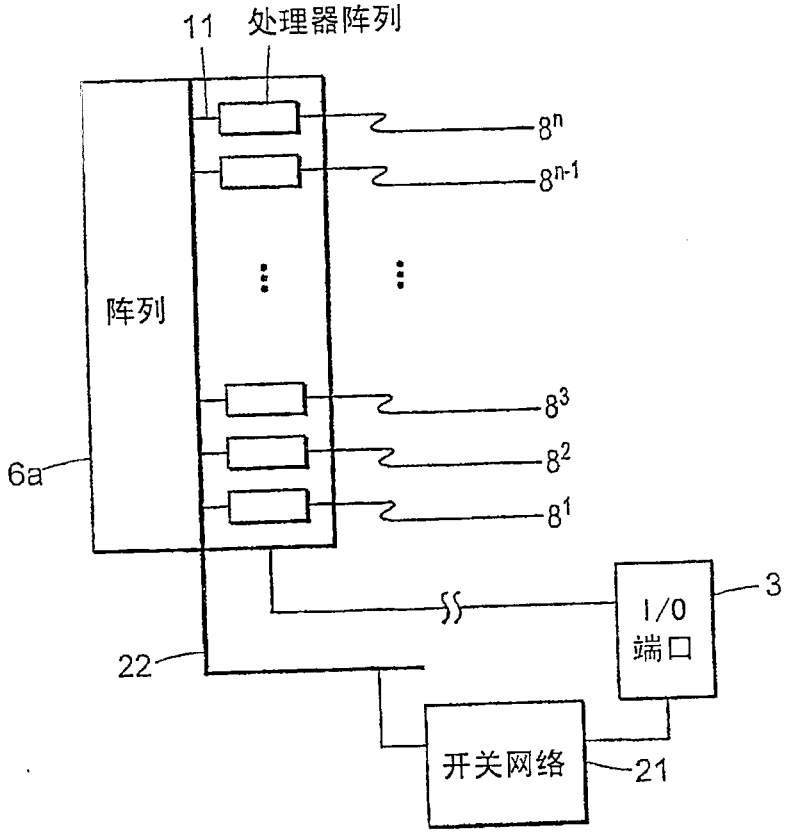

11
处理器阵列
8^n
8^{n-1}
阵列
8^3
8^2
8^1
6a
I/O
端口
3
22
开关网络
21

图 4B

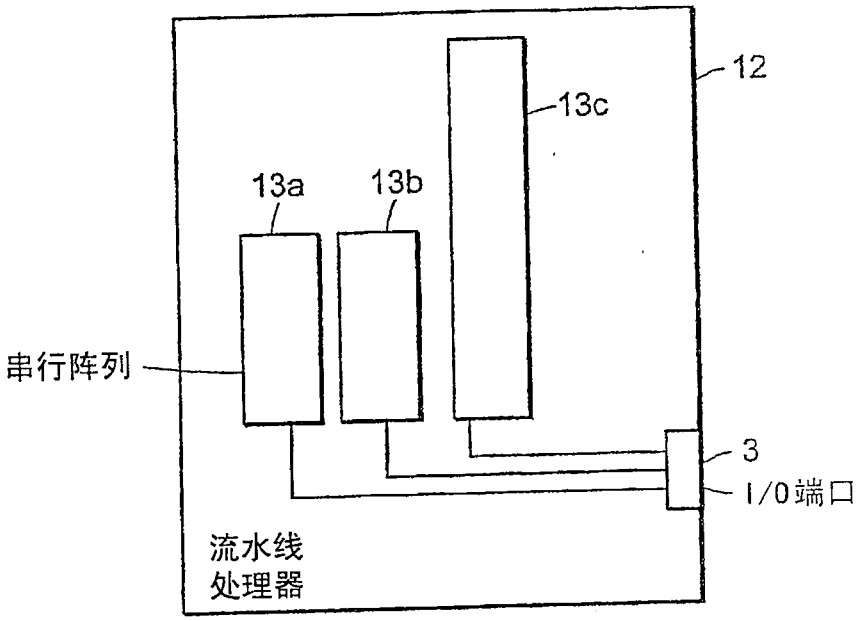

12
13c
13a
13b
串行阵列
3
I/O端口
流水线
处理器

图 5A

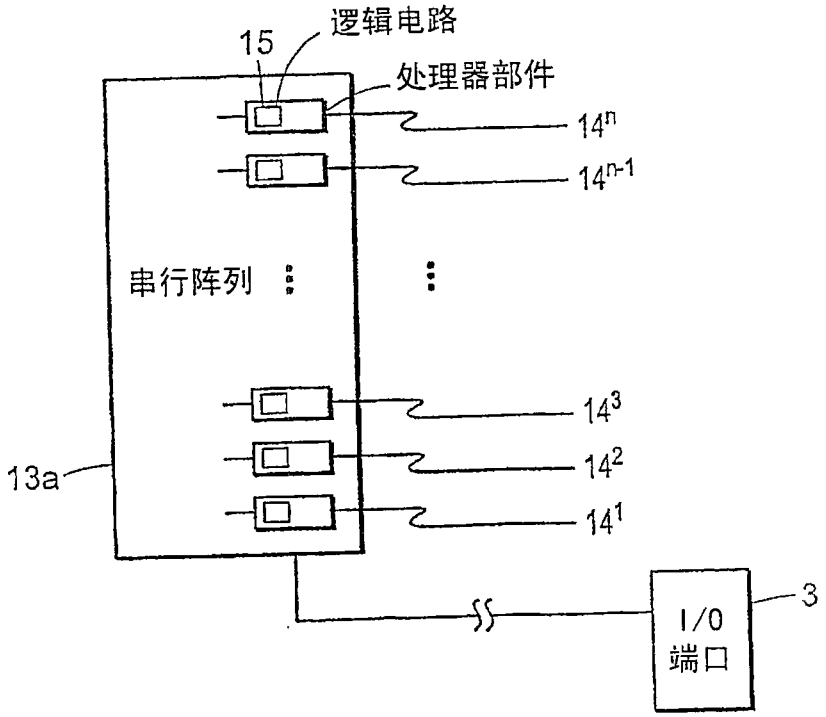

逻辑电路
15
处理器部件
14^n
14^{n-1}
串行阵列
14^3
14^2
14^1
13a
I/O
端口
3

图 5B

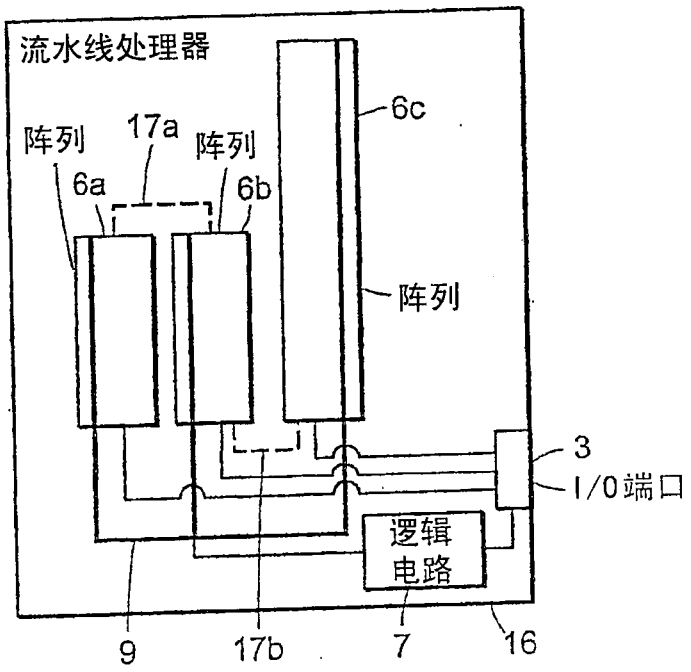

流水线处理器
阵列
17a
阵列
6a
6b
6c
阵列
3
I/O端口
逻辑
电路
9
17b
7
16

图 6

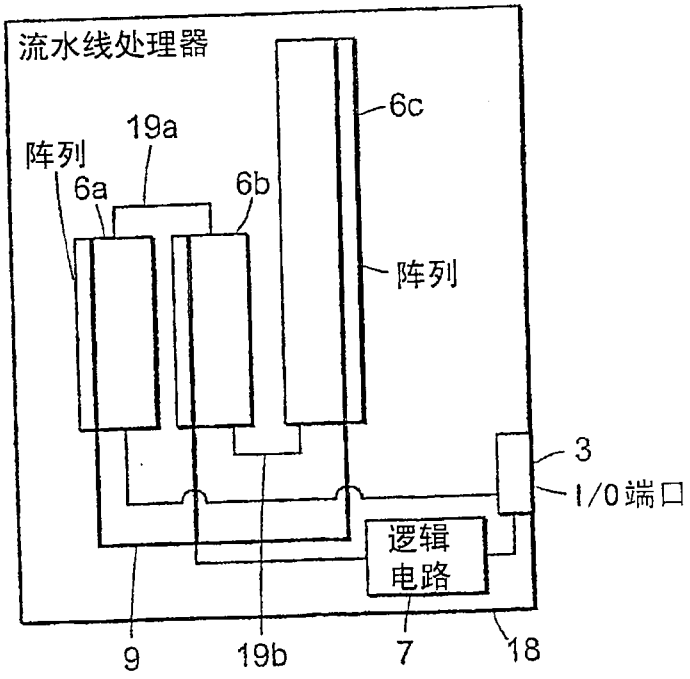

流水线处理器
19a
阵列
6a
6b
6c
阵列
3
I/O端口
逻辑
电路
9
19b
7
18

图 7

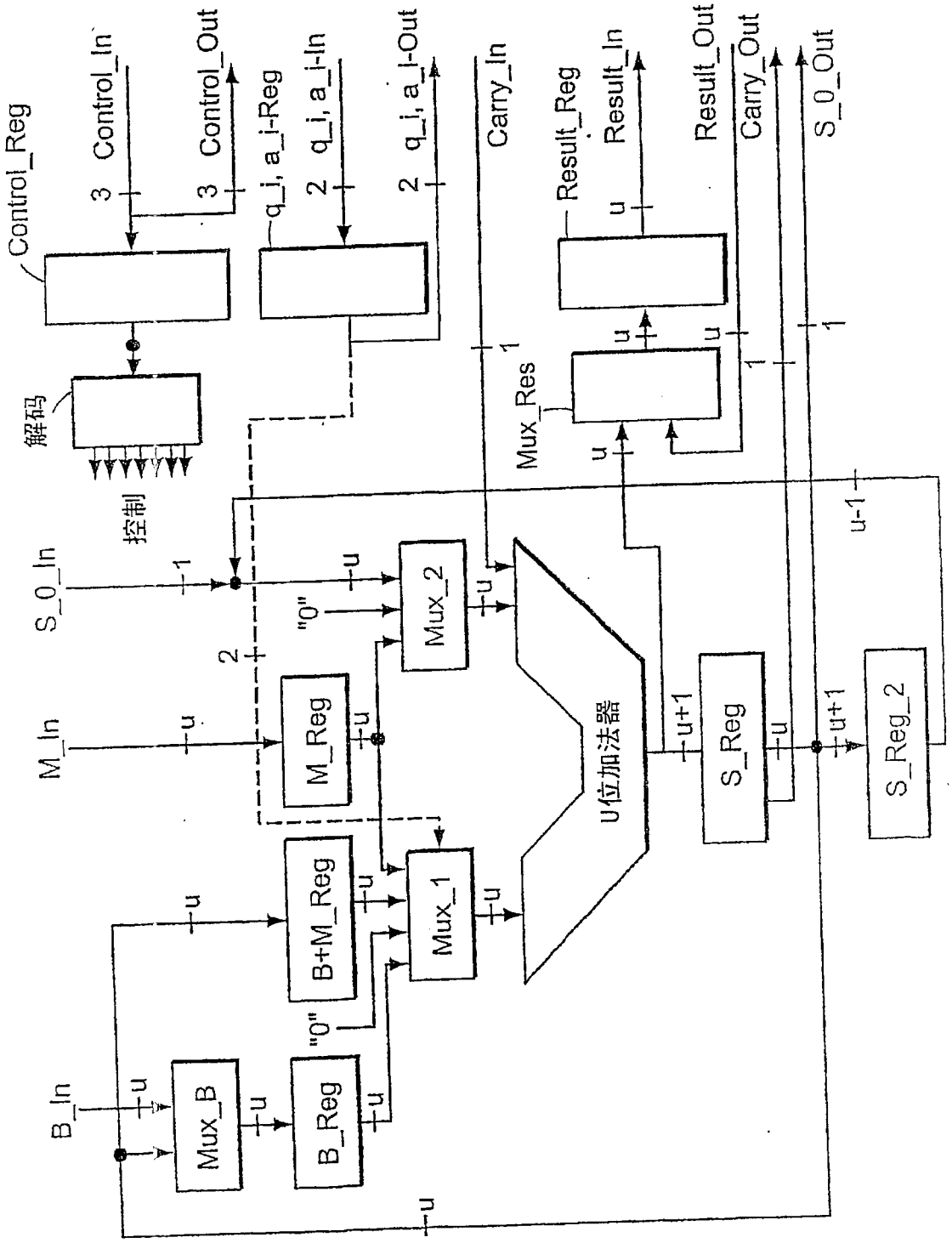

Control_Reg
3 Control_In
3 Control_Out
q_i, a_i-Reg
2 q_i, a_i-In
2 q_i, a_i-Out
Carry_In
Result_Reg
Result_In
Result_Out
Carry_Out
S_0_Out
解码
控制
Mux_Res
S_0_In
M_In
B_In
"0"
Mux_2
M_Reg
B+M_Reg
Mux_1
Mux_B
B_Reg
U位加法器
S_Reg
S_Reg_2
u
u+1
u-1
1
2

图 8

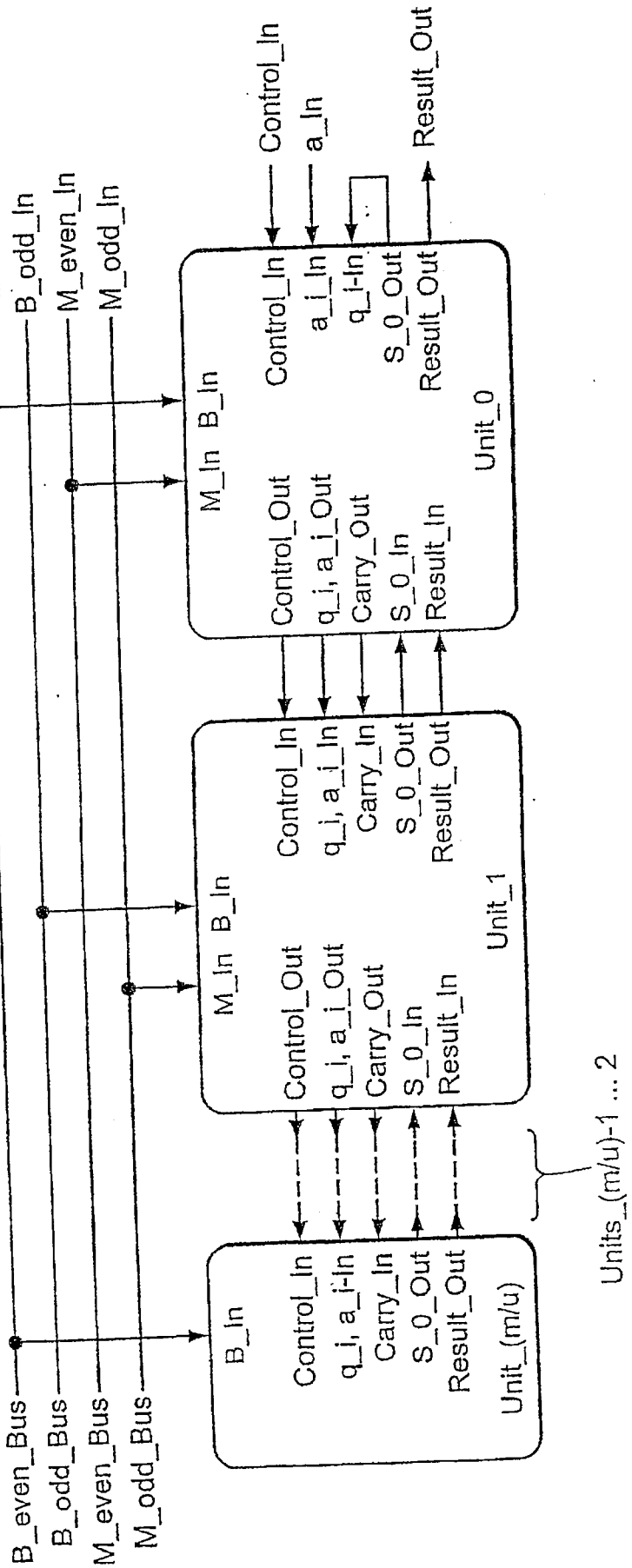

B_even_In
B_odd_In
M_even_In
M_odd_In
Control_In
a_In
Result_Out
M_In B_In
Control_In
a_i_In
q_i-In
S_0_Out
Result_Out
Unit_0
Control_Out
q_i, a_i_Out
Carry_Out
S_0_In
Result_In
Control_In
q_i, a_i_In
Carry_In
S_0_Out
Result_Out
Unit_1
Control_Out
q_i, a_i_Out
Carry_Out
S_0_In
Result_In
Units_(m/u)-1 ... 2
B_In
Control_In
q_i, a_i-In
Carry_In
S_0_Out
Result_Out
Unit_(m/u)
B_even_Bus
B_odd_Bus
M_even_Bus
M_odd_Bus

图 9

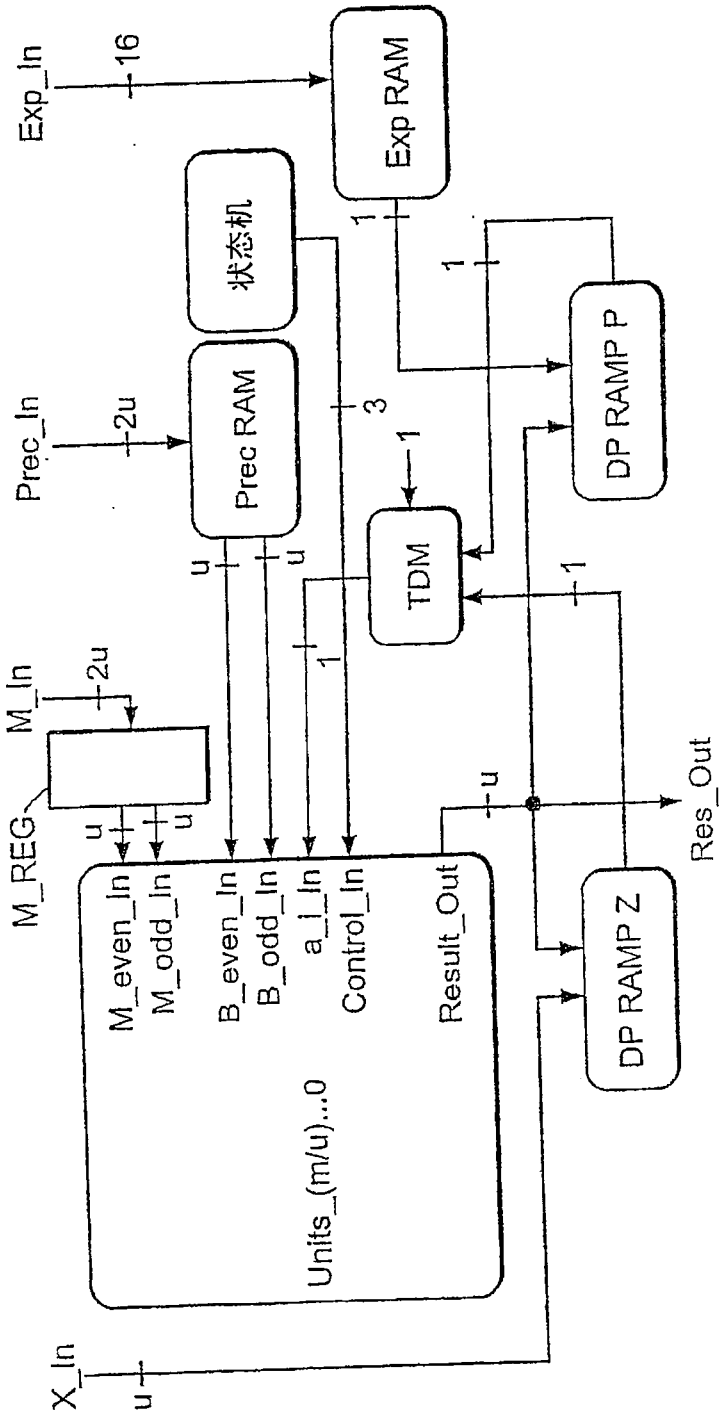

Exp_In
16
Exp RAM
状态机
Prec_In
2u
Prec RAM
3
1
DP RAMP P
TDM
M_In
2u
M_REG
u
M_even_In
M_odd_In
B_even_In
B_odd_In
a_l_In
Control_In
Result_Out
Units_(m/u)...0
Res_Out
DP RAMP Z
X_In

图 10

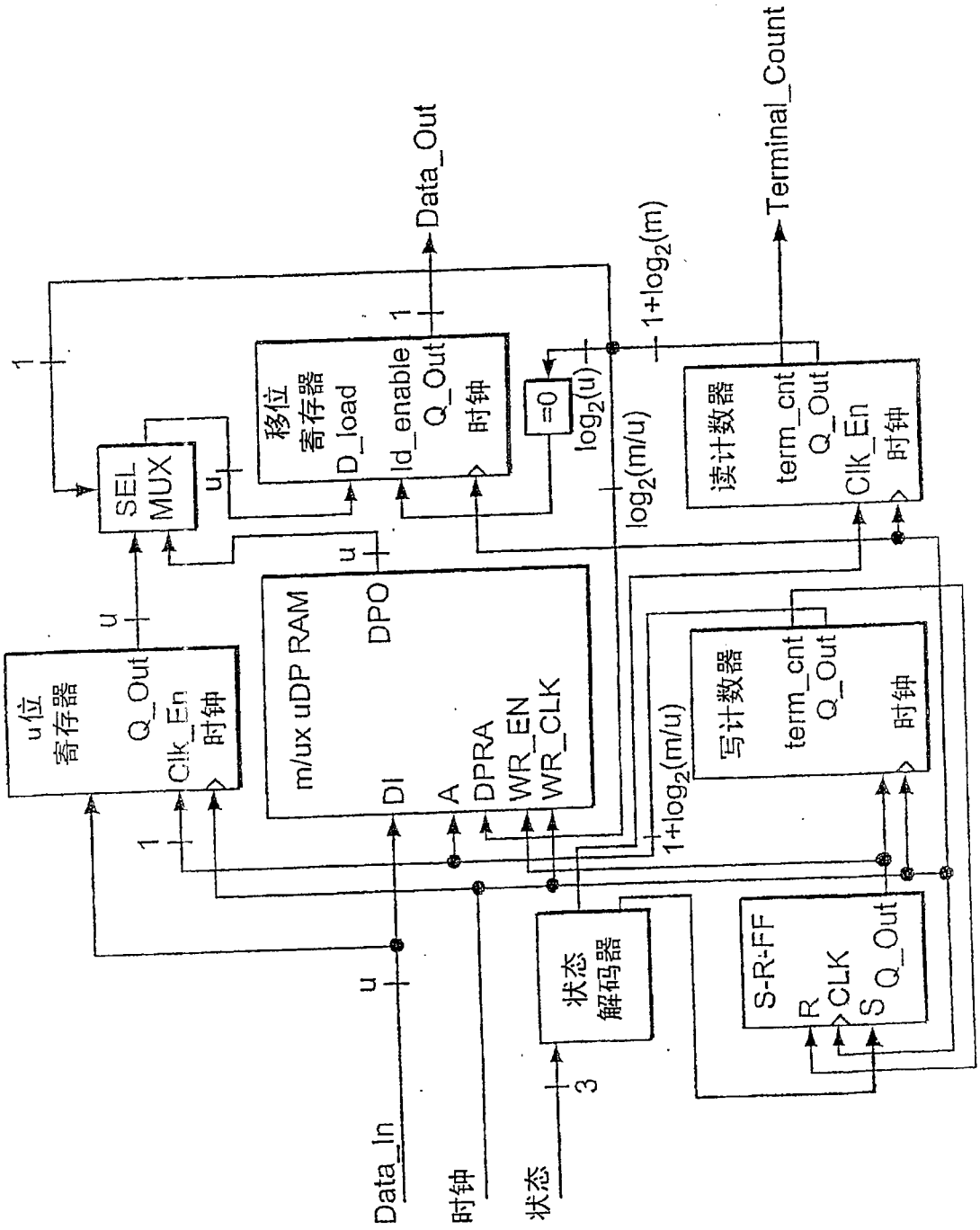

Data_Out
Terminal_Count
1+log2(m)
移位
寄存器
D_load
ld_enable
Q_Out
时钟
=0
log2(u)
log2(m/u)
读计数器
term_cnt
Q_Out
Clk_En
时钟
SEL
MUX
u位
寄存器
Q_Out
Clk_En
时钟
m/ux uDP RAM
DPO
DI
A
DPRA
WR_EN
WR_CLK
1+log2(m/u)
写计数器
term_cnt
Q_Out
时钟
状态
解码器
S-R-FF
R
CLK
S
Q_Out
Data_In
时钟
状态

图 11

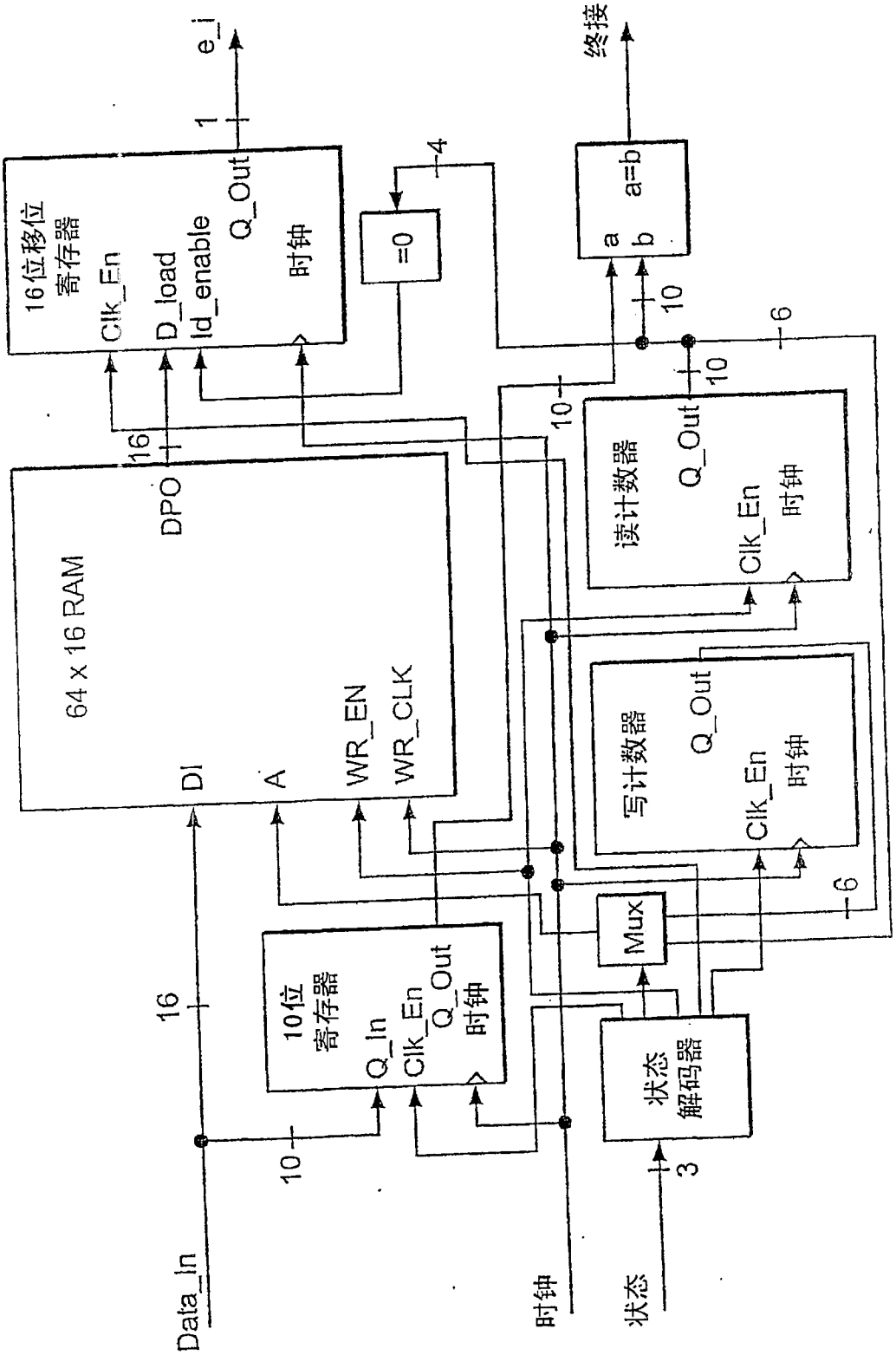

e_i
1
16位移位
寄存器
Clk_En
D_load
ld_enable
Q_Out
时钟
=0
4
终接
a
a=b
b
10
6
10
10
16
DPO
64 x 16 RAM
DI
A
WR_EN
WR_CLK
读计数器
Q_Out
Clk_En
时钟
写计数器
Q_Out
Clk_En
时钟
6
Mux
16
10位
寄存器
Q_In
Clk_En
Q_Out
时钟
状态
解码器
10
3
Data_In
时钟
状态

图 12

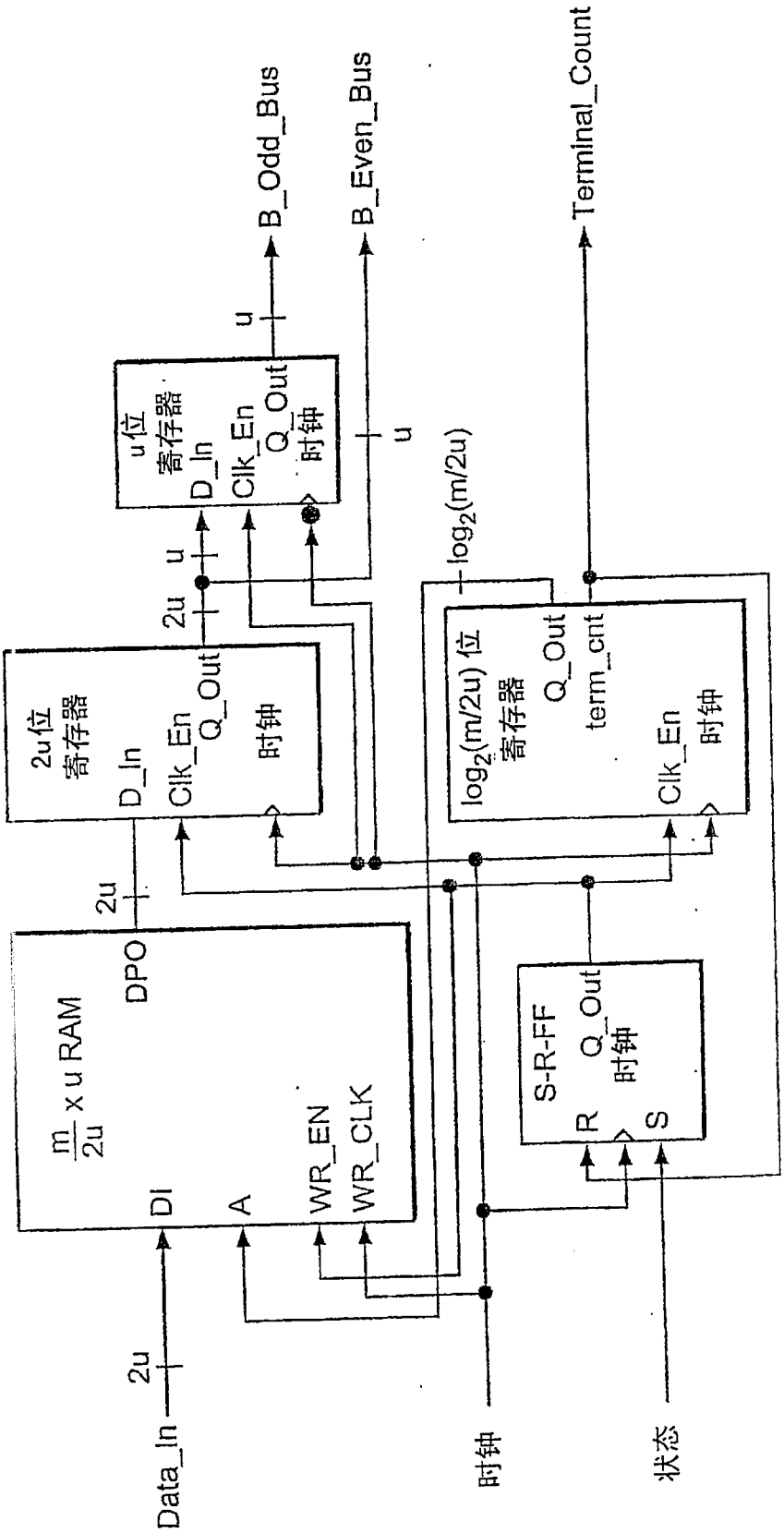

B_Odd_Bus
B_Even_Bus
Terminal_Count
u位
寄存器
D_In
Clk_En
Q_Out
时钟
u
2u
2u位
寄存器
D_In
Clk_En
Q_Out
时钟
log2(m/2u)
log2(m/2u)位
寄存器
Q_Out
term_cnt
Clk_En
时钟
m/2u x u RAM
DPO
DI
A
WR_EN
WR_CLK
S-R-FF
Q_Out
时钟
R
S
Data_In
时钟
状态

图 13

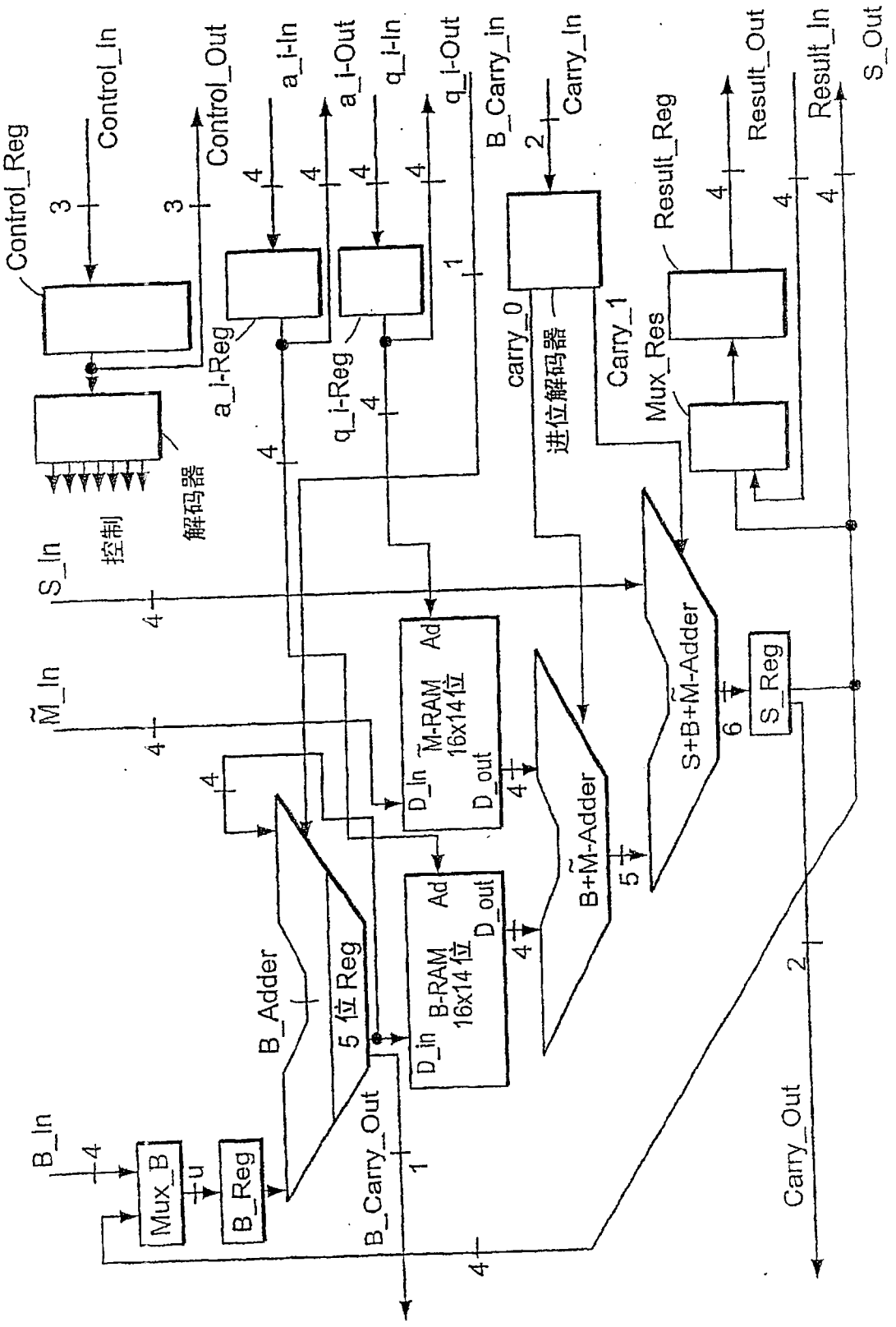

Control_Reg
Control_In
Control_Out
a_i-In
a_i-Out
q_i-In
q_i-Out
B_Carry_in
Carry_In
Result_Reg
Result_Out
Result_In
S_Out
a_i-Reg
q_i-Reg
carry_0
进位解码器
Carry_1
Mux_Res
控制
解码器
S_In
M̃_In
Ad
D_In
M̃-RAM
16x14位
D_out
B_In
Mux_B
B_Reg
B_Adder
5 位 Reg
B_Carry_Out
D_in
B-RAM
16x14 位
B+M̃-Adder
S+B+M̃-Adder
S_Reg
Carry_Out

图 14

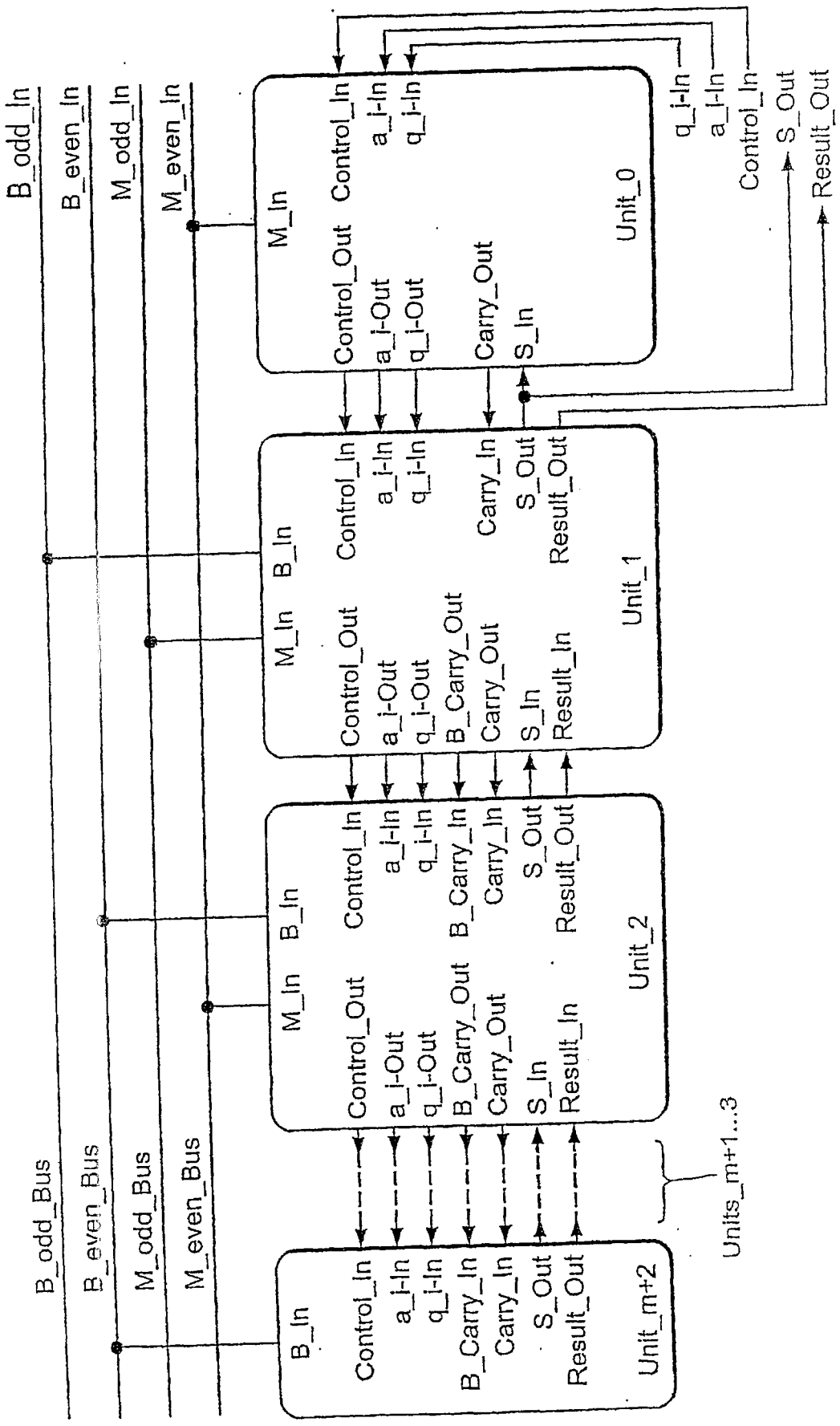

B_odd_In
B_even_In
M_odd_In
M_even_In
B_odd_Bus
B_even_Bus
M_odd_Bus
M_even_Bus
Unit_0
Unit_1
Unit_2
Unit_m+2
Units_m+1...3
M_In
B_In
Control_In
Control_Out
a_i-In
a_i-Out
q_i-In
q_i-Out
Carry_In
Carry_Out
B_Carry_In
B_Carry_Out
S_In
S_Out
Result_In
Result_Out

图 15

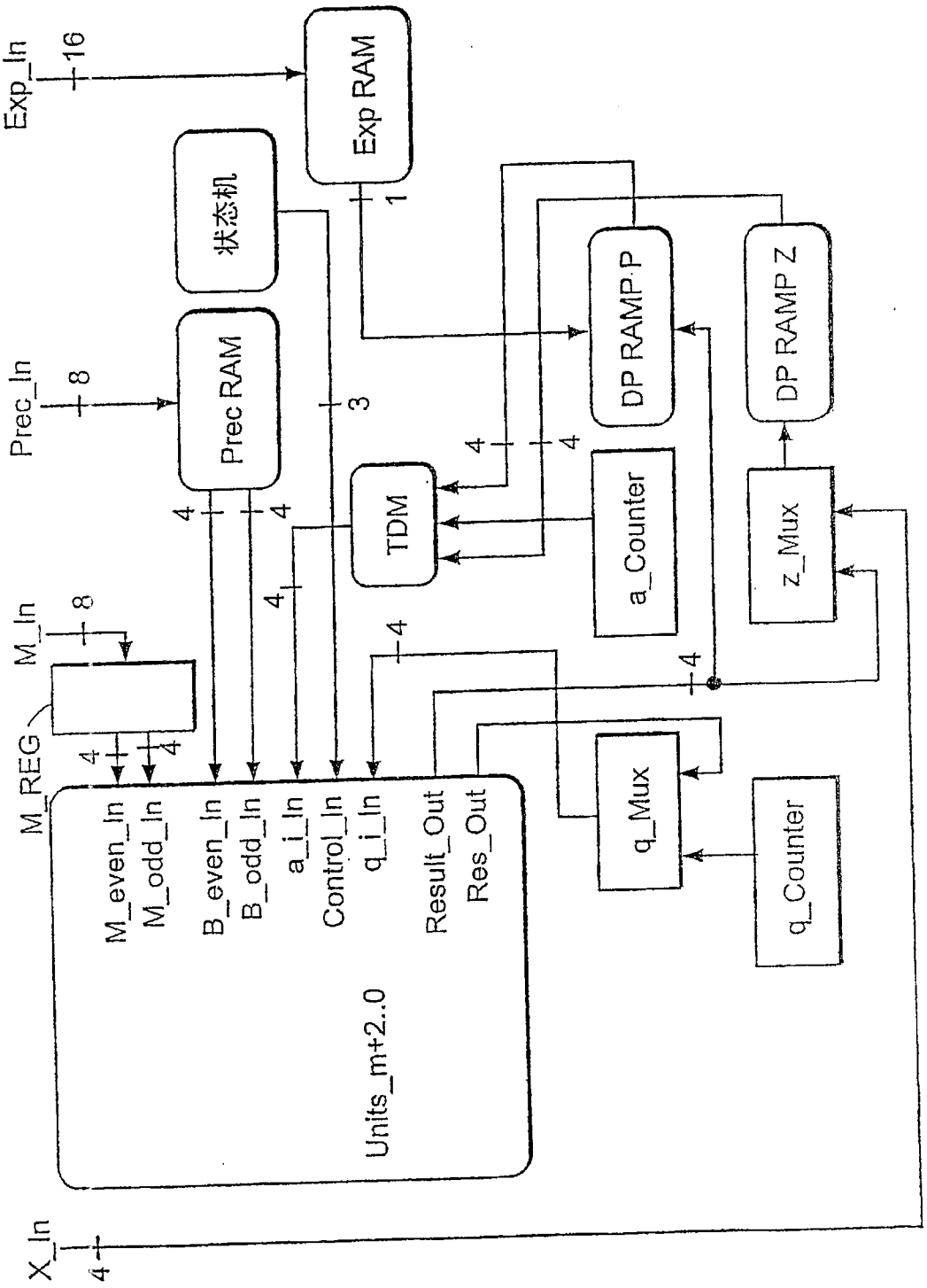

Exp_In
16
Exp RAM
状态机
Prec_In
8
Prec RAM
3
1
DP RAMP.P
DP RAMP.Z
4
4
TDM
a_Counter
z_Mux
M_In
8
4
4
4
4
4
M_REG
M_even_In
M_odd_In
B_even_In
B_odd_In
a_i_In
Control_In
q_i_In
Result_Out
Res_Out
Units_m+2.0
q_Mux
q_Counter
X_In
4

图 16

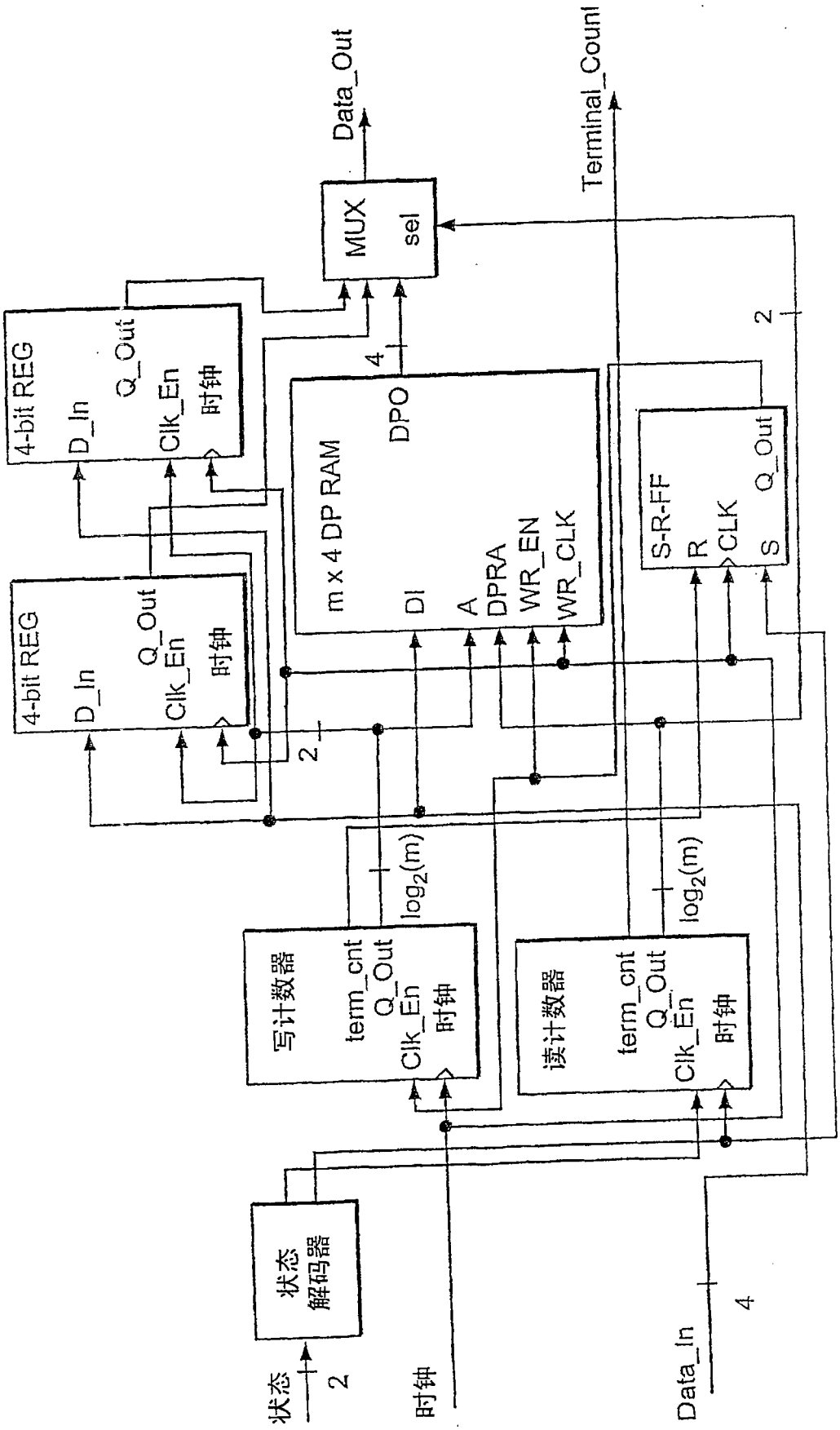

Data_Out
Terminal_Count
MUX
sel
4-bit REG
D_In
Q_Out
Clk_En
时钟
m x 4 DP RAM
DPO
DI
A
DPRA
WR_EN
WR_CLK
S-R-FF
R
CLK
S
Q_Out
写计数器
term_cnt
读计数器
log2(m)
状态
解码器
状态
时钟
Data_In
2
4

图 17