[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 17/50

[12] 发明专利说明书

[21] ZL 专利号 97199308.4

[45] 授权公告日 2003 年 9 月 10 日

[11] 授权公告号 CN 1121016C

[22] 申请日 1997.10.15 [21] 申请号 97199308.4
[30] 优先权
[32] 1996.10.30 [33] US [31] 08/739,606
[86] 国际申请 PCT/US97/18363 1997.10.15
[87] 国际公布 WO98/19256 英 1998.5.7
[85] 进入国家阶段日期 1999.4.29
[71] 专利权人 爱特梅尔股份有限公司
地址 美国加利福尼亚州
[72] 发明人 M·T·梅森 S·C·埃文斯
S·S·阿拉耐克
审查员 邹 斌

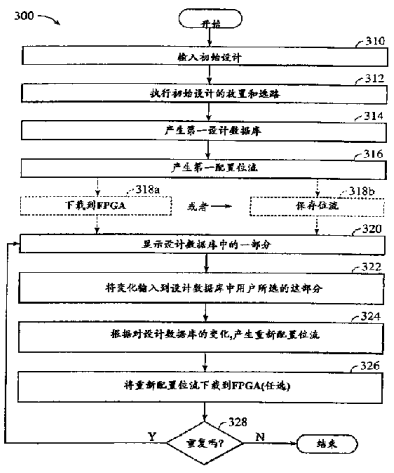
[74] 专利代理机构 上海专利商标事务所
代理人 李 玲

权利要求书 3 页 说明书 14 页 附图 13 页

[54] 发明名称 配置逻辑器件阵列的方法和系统

[57] 摘要

一种部分重新配置门阵列的系统和方法，通过逻辑电路的放置和路径(312)产生一个设计数据库(314)，访问该数据库以修改由放置和选路所创建的逻辑单元配置(322)。 根据修改情况，创建仅包含实现修改逻辑单元的位流的部分配置位流(324)。 将部分配置位流下载到门阵列(326)，由此执行门阵列的部分重新配置。 在另一个实施例中，根据本发明的系统包括软件实用程序(604)，它允许应用程序(602)在包含可编程门阵列的系统中执行，在运行过程中对门阵列进行重新配置。 实用程序包括根据在运行时间期间检测到的外部条件而修改设计的程序。 这种方法避免了需要提供一组预定备用设计，而允许应用程序自身作出决定。



ISSN1008-4274

1. 一种配置可编程逻辑单元阵列的方法，其特征在于：所述阵列包括多个可编程互连关系，每个所述逻辑单元在所述阵列中具有一个唯一单元位置，所述阵列具有一种相关的逻辑设计，所述方法包括：

访问一个设计数据库，该数据库代表逻辑单元定义和对应于所述逻辑设计的逻辑单元互连关系的一种配置；

把所述设计数据库的一部分提供给用户；

输入用户指定的变化，对所述设计数据库的用户所选部分进行重新定义；

产生一部分配置位流，仅描述与所述设计数据库的所述重新定义部分相关的所述逻辑单元和所述的互连关系；以及

将所述部分配置位流下载到所述逻辑单元阵列中，由此仅配置所述逻辑单元阵列中对应于所述用户指定变化的这些部分。

2. 如权利要求 1 所述的方法，其特征在于：所述的提供步骤包括识别被配置为逻辑门的特定逻辑单元以及对于每一个所述特定逻辑单元显示其单元位置和显示其相应逻辑门的图形图象或文本表示。

3. 如权利要求 2 所述的方法，其特征在于：所述的输入用户指定变化的步骤包括识别有待重新配置的逻辑单元、显示备用逻辑门的表以及从所述表中选择一个备用逻辑门，对所述被识别逻辑单元进行重新配置。

4. 如权利要求 3 所述的方法，其特征在于：所述的显示备用逻辑门表的步骤包括显示所述备用逻辑门的图形表示。

5. 如权利要求 2 所述的方法，其特征在于：所述的提供步骤进一步包括：

识别所述设计数据库中的固定值逻辑单元；

对于不与其它任何固定值逻辑单元邻接的固定值逻辑单元，显示所述固定值逻辑单元的单元位置和当前值；以及

对于一组相邻的固定值逻辑单元，将所述这组的单元位置显示为单元位置的范围以及将所述组中的逻辑单元的当前值显示为位流。

6. 如权利要求 1 所述的方法，其特征在于：所述的提供步骤包括仅显示被限定输出常数逻辑“1”或常数逻辑“0”的这些逻辑单元。

7. 如权利要求 1 所述的方法，其特征在于：所述的提供步骤包括：

识别所述设计数据库中的固定值逻辑单元；

对于不与其它任何固定值逻辑单元邻接的固定值逻辑单元，显示所述固定值逻辑单元的单元位置和当前值；以及

对于一组相邻的固定值逻辑单元，将所述这组的单元位置显示为单元位置的范围以及将所述组中的每个逻辑单元的当前值集中地显示为位流。

8.一种在 FPGA 中产生所述 FPGA 的配置位流的方法，所述 FPGA 包括可编程逻辑单元和 I/O 块以及设置在所述逻辑单元与所述 I/O 块之间的可编程互连关系，所述方法包括：输入一逻辑电路，所述逻辑电路包括多个逻辑门和所述逻辑门之间的连接；选择逻辑单元和互连关系来实现所述逻辑电路；形成一个包含所述所选逻辑单元和互连关系的放置和选路信息的设计数据库；根据所述放置和选路信息产生第一配置位流，所述第一配置位流包括所有的所述所选逻辑单元和互连关系的定义，由此而实现所述逻辑电路；把所述第一配置位流存储到数据存储器中，接着下载到所述 FPGA 中；以及将变化加入到所述逻辑电路设计中；所述方法的改进包括：

识别所述设计数据库中的第一类逻辑单元，它们被配置为逻辑门；

对于所述第一类逻辑单元中的每一个，显示其单元位置和其相应逻辑门的表示；

识别所述设计数据库中的第二类逻辑单元，它们被配置为产生常数逻辑值；

显示所述第二类逻辑单元的单元位置和逻辑值；

输入对所述设计数据库所选部分的修改；

产生第二配置位流，包括仅针对所述逻辑单元和所述互连关系中与所述设计数据库的所述所选部分的所述修改相关的这些的新定义。

9. 如权利要求 8 所述的方法，其特征在于：在对所述第一配置位流进行下载的所述步骤后进一步包括把第二位流下载到所述 FPGA 中的步骤。

10. 如权利要求 8 所述的方法，其特征在于：所述的显示所述第二类逻辑单元包括形成包含所述第二类逻辑单元中相互相邻这些逻辑单元的单元组，以及利用单元位置的范围显示每一个所述单元组中逻辑单元的单元位置。

11. 如权利要求 10 所述的方法，其特征在于：所述的形成单元组的子步骤包括形成所述第二类逻辑单元中这一些与其它所述第二类逻辑单元水平相邻的水平组，所述第二类逻辑单元中这一些与其它所述第二类逻辑单元垂直相邻的垂

直组，如果给定的逻辑单元既属于水平组又属于垂直组，那么，把所述的给定逻辑单元分配给所述的水平组。

12. 如权利要求 10 所述的方法，其特征在于：所述的形成单元组的子步骤包括形成所述第二类逻辑单元中这一些与其它所述第二类逻辑单元水平相邻的水平组，所述第二类逻辑单元中这一些与其它所述第二类逻辑单元垂直相邻的垂直组，如果给定的逻辑单元既属于水平组又属于垂直组，那么，把所述的给定逻辑单元分配给所述的垂直组。

13. 如权利要求 8 所述的方法，其特征在于：所述的显示所述第二类逻辑单元包括形成包含所述第二类逻辑单元中用户指定逻辑单元的单元组，以及显示每一个所述单元组中逻辑单元的单元位置和逻辑值。

14. 一种配置 FPGA 的系统，所述 FPGA 具有至少一个编程逻辑设计，所述系统包括：

在所述 FPGA 的运行时间操作期间检测所述 FPGA 外部条件的装置；

响应于所述 FPGA 外部条件的检测结果重新设计一部分所述逻辑设计的装置；

仅基于所述重新设计的部分形成部分配置位流的装置；

将所述部分配置位流发送到所述 FPGA，由此在运行时间中对所述 FPGA 进行部分重新配置以实现所述逻辑设计的所述重新设计部分的装置。

15. 如权利要求 14 所述的系统，其特征在于进一步包括至少存储设计数据库中代表所述逻辑设计的一部分的装置，所述的重新设计的装置包括访问所述设计数据库，由此提供对所述逻辑设计进行重新设计基础的装置。

16. 一种配置 FPGA 的系统，所述 FPGA 具有至少一个编程逻辑常数，所述系统包括：

在所述 FPGA 的运行时间操作期间检测所述 FPGA 外部条件的装置；

响应于所述 FPGA 外部条件的检测结果确定所述逻辑常数的一个新值装置；

仅基于所述重新设计的部分形成部分配置位流的装置；

将所述部分配置位流发送到所述 FPGA，由此在运行时间中对所述 FPGA 进行部分重新配置的装置。

配置逻辑器件阵列的方法和系统

技术领域

本发明总的涉及可编程逻辑器件，更具体地涉及现场可编程门阵列(FPGA)的配置.

背景技术

在集成电路技术的早期，诸如移位寄存器、多路转换器、加法器等的逻辑电路是由数字集成电路构成的．这些小规模集成(SSI)电路通常含有数目较少(例如四至八个)的逻辑门，如“与”门、“或”门、触发电路和锁存器等，需要几十个晶体管．随着技术的发展，能够将越来越多的晶体管装入到 IC 电路中．目前，半导体制造商能够将数百万个晶体管装在单个晶片上，从而形成高度复杂的芯片，如现代微处理器.

装有这些 VLSI(特大规模集成)和 ULSI(超大规模集成)芯片的系统同样是复杂的．这种系统通常采用若干个常规逻辑芯片提供各种各样的支撑逻辑功能．已研制出允许制造商快速地实现客户的逻辑设计的门阵列．这些器件是由利用标准方法制造的逻辑门阵列组成的．器件的定制是在最后几步制造步骤中发生的，在这几步中形成与逻辑门连接的金属层以实现所需的逻辑功能.

这些门阵列演变为可编程的器件，给仅需要少量器件或者还未完全开发逻辑设计但是需要少量样品作测试的设计人员提供了更大的灵活性．另一种可编程逻辑器件采用熔断器提供器件中逻辑门之间的互连．熔断器被熔断便断开连接，或者在所谓的抗熔断器的情况中形成连接．因此，这种器件只能使用一次，仅存储一组逻辑功能.

可编程逻辑器件的继续发展已经导致可重新编程的互连关系的发展，更重要的是可配置的逻辑单元的发展．顾名思义，可配置逻辑单元允许设计人员能够对逻辑单元进行编程，使其具有许多基本逻辑门中任何一个门的功能或者更高级的逻辑功能．当前的制造技术使得生产具有数以千计个可配置逻辑单元及其相关互

连的高密度器件，即称为现场可编程门阵列(FPGA)成为可能．提供这种高密度器件的能力使得设计人员能够采用越来越复杂的逻辑功能．象以前的产品一样，FPGA 包括可编程的互连关系．此外，互连关系是可重复编程的，进一步增大了FPGA 的利用价值．

然而，以往对这些可重复编程 FPGA 的重新配置通常需要对整个器件进行重新配置． Atmel 公司(本发明的受让人)制造的 FPGA 代表了这种器件向前又迈进了一步．将这种器件称为可动态重新配置的 FPGA，允许仅对所选的一部分逻辑阵列进行重新配置．用这种方法能够对 FPGA 作出变化，不必对整个器件进行编程，从而允许仅对所选的一部分阵列进行重新配置．

参考图 1，典型的 FPGA 100 包括多个可配置的逻辑单元 130、可配置的 I/O 块 110 和可配置的互连关系 120、 122，共同称之为 FPGA 资源．尽管互连关系 120、 122 是以各个互连线网格示出的，但是每条“线”实际上是一组互连线，例如，如图 3B 所示．每个逻辑单元 130 和 I/O 块 110 包括数据线 140、 142，它们能够有选择地耦合到互连关系 120、 122 上．

典型设计周期从设计一个或多个然后将在 FPGA 中实现的逻辑电路开始．逻辑设计包括逻辑门和这些逻辑门之间的互连关系．然而，特定的设计，如数字滤波器利用“常数”，即 1 和 0 的串来限定其特性．为了描述本发明，把这种常数称为设计的一部分，也可以称为逻辑门．

例如，图 2 示出一种简单的逻辑设计．逻辑设计中的每个元件用一个实例名来识别．因此，图 2 中的与门和或门取名为 G1-G3．图 3A 示出逻辑设计是如何在 FPGA 100’中出现的．将图 2 中所示的门 G1-G3 中的每一个门和互连关系映射到图 3A 中所示的所选逻辑单元和互连关系上．同样，将输入 A-D 和输出 OUT(图 2)映射到所选 I/O 块．因此，互连关系 120a-120c 和 122a-122c(以较淡的线表示)与逻辑单元 G1-G3 和 I/O 块 110a-110e 连接在一起．图 3B 示出了图 3A 配置中一部分放大图．说明不同逻辑单元、互连关系和 I/O 块之间的具体互连情况．尽管在该图中的设计并未示出常数的使用，但是，众所周知，能够将现代 FPGA 中逻辑单元构造为输出逻辑“ 1 ”或逻辑“ 0 ”，能够根据需要如此构造一组逻辑单元，产生一个或多个 1 和 0 的串．

现在将说明把图 2 的设计转换到如图 3A 中所示的 FPGA 的步骤．由于绝大多数的目前设计在功能上趋向于相当复杂，通常采用计算机辅助设计(CAD)工具

以便于设计过程．因此，在图 4 中，设计流程图 200 从输入逻辑电路初始设计，即步骤 210 开始，例如通过利用 CAD 工具．

接着是逻辑电路的逻辑门的放置和选路，步骤 212．作为放置和选路步骤的结果，产生设计数据库，步骤 214．设计数据库规定 FPGA 中将参与实现逻辑电路的这些逻辑单元、 I/O 块和互连关系(即资源)，包括所选资源的位置及其路经或逻辑结构．图 1 示出用于识别逻辑单元位置的若干坐标系中的一个．在图 1 所示的传统坐标系中，逻辑单元按照从左到右和从下到上的次序编号，从左下角单元(0， 0)开始，到右上角单元(3， 3)结束．通常，设计数据库另外还包括在设计阶段期间给逻辑电路元件指定的实例名，图 2．

从包含在设计数据库中的信息，由通常称为位流编译器的工具产生配置位流，步骤 216．位流编译器获取存储在设计数据库中的位置和配置信息，产生将配置 FPGA 中各种资源的限定位流．在物理级上，限定位流代表 FPGA 中晶体管(开关)的开/关(ON/OFF)状态，它实际上控制每个逻辑单元和 I/O 块的配置以及逻辑单元与 I/O 块之间的互连．

在这里，或是可以把配置位流下载到逻辑阵列，由此而配置器件，步骤 218a，或是可以把位流保存在磁盘上，步骤 218b．图 4 中用虚线表示这两种可供选择的方案．

即使已经对设计作了调试并按照预期的进行操作，偶然也会发生需要对初始设计进行变化．例如，新的要求可能导致功能定义的变化，这就必须改变初始设计．利用通常的现有技术，对初始设计的最终版本的修改，步骤 220 导致上述步骤的重复．因此，设计人员利用 CAD 工具访问初始设计的最终版本并对该设计作出所需改变．执行第二次放置和选路步骤，步骤 222，由此产生第二个设计数据库，步骤 224．然后，根据新的设计数据库由位流编译器产生第二个配置位流，步骤 226．

与第一配置位流一样，可以把也可以不把第二配置位流下载到 FPGA．如果需要下载，可提供两种选择：即能够把位流整体下载到 FPGA，由此重新配置整个阵列，以包含改进的设计．另一种选择，这里 FPGA 是可动态重新配置的，意思是指器件能够被部分重新配置，能够仅对第二配置位流中对应于设计变化的这些部分进行下载．通过确定第一与第二配置位流之间的差异产生部分(可重新配置的)位流首先实现之，步骤 228．然后，把部分位流下载到可动态重新配置的

FPGA，步骤 230，由此而实行 FPGA 的部分重新配置，其中，仅对改进设计中所涉及的这些逻辑单元、互连关系和 I/O 块进行重新编程．最后，对于另外的设计改进重复循环，就象出现设计功能要求上变化一样．

设计中逻辑门的放置和选路是一项计算强度高的活动．当设计人员由于高密度 FPGA 的提供能力趋于采用越来越复杂的设计，放置和选路操作会大大增加．在图 4 所示的现有方法中，考虑到设计循环中的每一次迭代可能需要完全的放置和选路操作，专用于放置和选路计算的时间量会达到惊人的程度．

现有技术方法的另一个方面是可能会出现改进后的设计将导致放置和选路配置不同于以前设计的放置和选路配置．如果以前的设计已经精心地调谐到提供特定临界时序特性，这就成为一个问题．通过全部放置和选路操作进行的下一设计会导致具有不同净长度的逻辑的不同放置，由此对电路的时序产生不利影响．在涉及实时应用的地方，这是一种不可以接受的情况．

需要一种对于完全完成和调试逻辑设计的重新设计在周转时间方面能改善现有技术的方法．还需要一种开发方法，维持现有设计中未涉及设计修改的部分不受影响．

到这里，讨论一直集中围绕逻辑电路的设计是在设计人员的工作实验室的环境中．然而，已观察到，FPGA 的现场应用会不适合于单个静态设计．尽管 FPGA 可以在实际工作条件下在现场重新配置，但是，新的配置通常由存储在诸如 EPROM 的非易失性装置中的完整设计组成．从 EPROM 读出配置位流并将其下载到 FPGA 中．因此，可供使用的配置的尺寸受 EPROM 的存储容量的限制．在诸如自适应滤波的应用中，迫切需要滤波器的响应能够根据诸如被滤波数据的频率和相位特性，即通常不可预料的条件，在运行时变化．在这种情况中，事前便不能指定新的配置，如滤波器参数．

然而，还需要在运行过程中根据装置正在工作的环境对 FPGA 进行配置的能力．此外，由于仅需要对 FPGA 中一部分进行重新配置，如在自适应滤波器的情况中，需要具有运行过程中进行部分重新配置的能力．

发明概要

根据本发明，一种对可动态重新配置 FPGA 进行配置的方法，从输入初始逻辑设计开始．在该设计上进行放置和选路，导致将位置分配给实现该设计的 FPGA

中的逻辑单元和互连关系．设计数据库是放置和选路操作的结果．接着，在设计数据库上工作的位流编译器产生配置位流．位流或是可以被下载到 FPGA，由此对装置进行配置，或是可以简单地被保存在存储媒介上．

根据相应的设计数据库作出对初始设计最后工作版本的变化．修改后的设计数据库被保存起来，从修改后的设计数据库产生第二配置位流．根据本发明，第二配置位流仅由对应于修改后逻辑设计的这些逻辑单元、I/O 块和互连关系的定义位串组成．

通过直接访问设计数据库以执行设计的变化，完全可避免本来需要的耗费在放置和选路计算上的时间．当必须对逻辑设计的最后工作版本作修改时，例如，由于在装入 FPGA 的系统的功能要求上的变化需要修改时，这是尤其有用的，也是想要的特性．经常的情况是，设计变化自然越来越多，仅涉及到整个设计的一个局部部分．在其它时间，设计人员希望对逻辑设计的最后工作版本的变化做实验．本发明的配置方法可保证用很短的检修时间来实现设计变化，所以，渐增的变化和“ What-if ”实验是可行的和便利的，不会对项目开发产生不利影响也不产生停止工作时间表．此外，由于放置和选路操作是旁通的，设计中设计人员未作修改的关键部分的任何时序将不受影响并保证提供已知的时序性能．更通俗地说，设计中未作修改的任何部分将继续按预期行使其特性．因此，采用本发明的方法，保证产生的 FPGA 应用在结构上是正确的，因此功能上是正确的．

在本发明的一个较佳实施例中，通过图形接口把设计数据库提供给用户．接口允许用户指定一部分设计数据库进行显示．设计数据库通过其实例名和在 FPGA 中的位置确定逻辑单元、I/O 块和互连关系．给用户提供逻辑单元，包括被配置为输出常数逻辑值的这些逻辑单元的名称、位置和当前配置．也给出逻辑单元当中的互连关系．

通过从一张以图形方式或者文本方式表示的逻辑门和互连线选项的表或菜单中进行选择可以作出对设计数据库的修改．对于常数的修改可以以文本方式通过输入新的常数值或者以图形方式通过操纵一个或多个能够打开（逻辑“ 0 ”）或关闭（逻辑“ 1 ”）的开关的图标进行．

在本发明的另一个实施例中，对 FPGA 进行实时重新配置的系统包括与装置的硬件接口和为了下载新配置数据而访问该装置的软件实用程序．软件实用程序包括产生部分配置位流的手段，允许应用程序根据运行中检测到的操作环境中的

特定条件对 FPGA 进行配置．硬件接口和软件实用程序允许在装置使用中访问 FPGA，从而能够对装置中的一部分进行重新配置．

附图简述

图 1 是典型 FPGA 的结构．

图 2 是逻辑电路的一个例子．

图 3A 和 3B 示出图 2 的逻辑电路在图 1 的 FPGA 中的实施．

图 4 是在 FPGA 设计中通用步骤的要点．

图 5 是本发明设计方法的一个实施例的要点．

图 6 是按本发明配置 FPGA 的系统的方框图．

图 7A-7C 示出数字滤波器设计．

图 8A 和 8B 是本发明的用户接口的屏幕样板．

图 9A-9D 示出利用基于 ROM 查找表实现的加法器电路和减法器电路．

图 10A-10B 示出本发明的用户接口的屏幕样板．

图 11 示出一个实时 FPGA 重新配置的系统．

实现本发明的最佳方式

参考图 5，按照本发明一个实施例的配置方法 300 包括创建一种初始设计，步骤 310，和执行放置位置和选路径的操作，获得一个第一设计数据库，步骤 312 和 314．然后，位流编译器创建配置位流，步骤 316，可以将其下载到 FPGA 上，配置装置，步骤 318a．另一方面，配置位流可以简单存储在存储器装置中，步骤 318b．配置位流包括多个定义位流，它指定 FPGA 中的资源将如何进行配置．

当需要对初始逻辑设计的最终工作版本进行变化时，设计修改可以直接对设计数据库进行．然后，继续图 5 所示的方法 300，向设计人员显示设计数据库中的一部分，步骤 320．设计人员选择设计数据库中需要作修改的这些部分和输入变化，步骤 322．根据输入的修改情况，产生部分配置位流(也称为重新配置位流)，它仅由对应于修改的定义位流构成．然后，可以将这部分位流下载或保存到存储媒体中，步骤 326．

图 6 示出对动态可配置 FPGA 进行配置的系统，它体现了图 5 所示的方法．系统 400 包括数据存储器 420 和各个模块 402-420．采用诸如 CAD 工具、简图捕

获程序等设计输入模块 402 来创建初始设计 420a 并存储到磁盘 420 上. 放置和选路模块 404 获取初始设计 420a 并创建设计数据库 420b，它也存储在磁盘上. 位流编译器 406 产生配置位流 420c，下载模块 408 把配置位流下载到 FPGA150 .

设计人员通过图形用户接口(GUI)模块 410 输入对初始逻辑设计最终工作版本的修改. GUI 从设计数据库 420b 进行读出并显示由设计人员所选的设计数据库中的一部分. GUI 接收对设计数据库的修改并仅根据输入的修改创建部分位流. GUI 装入位流编译器的功能，产生部分位流. 然后将部分位流 420d 存储到磁盘上，接着通过下载模块 408 下载到 FPGA . 另一方面，GUI 能够把部分位流直接传送到下载模块 408 . 在这里应当注意：如果不需要间接下载到 FPGA，那么能够采用另一种存储媒体来存储配置(和重新配置)位流. 例如，可以把配置(和重新配置)位流下载到 EEPROM 等存储媒体上，以供下一步分配.

为了说明图 6 所示的方法，考虑一个自适应数字滤波器，如图 7A 中所示. 滤波器设计 500 由串联级联的延迟寄存器 R1-R7 、乘法器 M1-M8 和加法器 A1-A7 组成. 数据宽带为八位(一字节). 一组滤波器系数 C_0-C_7 起乘法运算的被乘数的作用. 由于数据中的每个字节通过寄存器移位和传播，所有乘法器形成系数与数据之间的乘积项. 然后由加法器 A1-A7 对乘积项求和.

回想一下逻辑设计，如滤波器设计 500 中的每个元件具有一个由设计人员开始指定的实例名. 因此，参考图 7A，寄存器取名为 R1-R7，乘法器取名为 M1-M8，加法器取名为 A1-A7 . 还给系数 C_0-C_7 指定了实例名. 具体说，包含系数的每一个位具有一个名称. 从图 7C 中所示的系数 C0 的扩展图中更加清楚地示出了这一情况. 系数是一个八位量，这里，将位取名为 C0_0 至 C0_7 . 再回到图 7A，应当明白，也可以给元件 R1-R7 、 M1-M8 和 A1-A7 当中的互连关系取名. 然而，为了避免图面的杂乱，已经省略了这些互连关系的实例名.

滤波器设计 500 的放置和选路导致设计数据库，由实例名、每个寄存器和算术算符以及被选作实现这些功能的逻辑单元的位置和配置指定. 以同样 的方式，设计数据库列出位的实例名，包括系数 C_0-C_7 、它们的相应逻辑单元的位置和逻辑单元是否被配置为产生逻辑 “ 0 ” 或逻辑 “ 1 ” . 通过 FPGA 中的坐标系，如图 1 中所示的坐标系识别出现在设计数据库中的逻辑单元.

图 7B 示出图 7A 的数字滤波器的改进设计 500’，这里已经指定了一组不同的系数 C'_0-C'_7 . 希望根据本发明的方法作这种修改的设计人员开始访问基于初

始设计 500 创建的第一设计数据库．这最好是利用图形用户接口(GUI)实现，当然基于文本的接口也一样是有效的，也许使用更困难和更低效些．

在一个实施例中， GUI 有一种模式，用于对逻辑设计中所定义的逻辑常数进行编辑．例如，图 8A 的屏幕样板显示了设计 500 中的每个常数 C_0-C_7．在“实例名”列下是包括每个系数 C_0-C_7 位的名称．每个位由一个配置为输出常数逻辑电平的逻辑单元实现．回想一下，每个逻辑单元具有一个相应的实例名．在单元被范围识别的地方，如在图 8A 的情况中，显示范围中第一和最后单元的实例名．在“位置”列下是在相应逻辑单元的 FPGA 中的位置．为了说明起见假设，被使用的坐标编号惯例遵循图 1 中所示的惯例．因此，根据图 8A，包括系数 C_0 的逻辑单元位于 FPGA 中第 10 行的第 7 - 14 列中；系数 C_1 的逻辑单元位于第 12 行的第 7 - 14 列中，依此类推．“现行配置”列显示由逻辑单元产生的当前逻辑值．例如，包括系数 C_0 的第 7、8、9、10、11、12、13 和 14 列中八个逻辑单元显示，为当前配置相应的输出逻辑值“1”、“0”、“1”、“0”、“1”、“0”、“1”和“1”．“新配置”列由输入字段组成，允许用户输入新的逻辑值，作为一位串．这位串位中的每个位与这位串所表示的常数单元具有一一对应关系．

由于设计数据库包含产生图 8A 所示显示屏所需的所有信息， GUI 通过设计数据库进行简单搜索，找出已经被配置为产生常数逻辑电平的这些逻辑单元，访问它们的实例名、单元位置和当前配置信息．访问设计数据库和产生如图 8A 中所示显示信息的具体软件实施细节将随任何给定设计数据库格式所使用的精确数据结构而变化．在计算机程序员的范围和能力之内，这些实施细节是专业人员所公知的．

图 8A 中所示的逻辑常数最好按照水平相邻性分组．在一个给定的常数逻辑单元块中，可以将逻辑单元分组为水平单元组或垂直单元组．例如，在图 8A 中，可以看出，单元位置(7， 18)至(14， 21)限定 32 个逻辑单元的块．这些单元被显示为四个由八个水平相邻单元组成的组．因此，四个水平组是：(7， 18)至(14， 18)；(7， 19)至(14， 19)；(7， 20)至(14， 20)和(7， 21)至(14， 21)．

另一方面，块(7， 18)至(14， 21)中的单元最好能够按照垂直相邻性分组．在图 8B 的屏幕样板中示出这一情况．在这种情况下，有八个由四个单元组成的垂直组，每一个为：(7， 18)至(7， 21)；(8， 18)至(8， 21)；(9， 18)至(9，

21); (10, 18)至(10, 21); (11, 18)至(11, 21); (12, 18)至(12, 21); (13, 18)至(13, 21)和(14, 18)至(14, 21). 于是将这些单元的其它字段修改反映其垂直分组.

根据较佳实施例, 将相邻常数单元的组表示为单元的范围. 由于构成系数 C_0-C_3 的单元仅呈现水平相邻性它们显示成水平组. 同样, 对仅显示垂直相邻性的单元进行垂直分组. 能够对常数单元进行垂直或水平分组的地方, 按照用户指定的优先选择进行分组. 因此, 根据设计人员的优先选择, 能够对系数 C_4-C_7 进行水平或垂直分组, 如图 8A 和 8B 所示. 在图中所示的例子中, 优先选择可以针对水平相邻性, 如图 8A 所示. 然而, 在另一种设计中, 可以按垂直方式排列系数, 在这种情况中, 可以作出图 8B 所示的垂直优选.

在本发明的另一个实施例中, 可以将非相邻的常数单元分组在一起. GUI 允许用户选择常数单元中的任意组合并将所选单元作为一个组处理. 尽管这些单元是非相邻的, 但是, GUI 可显示作为一个组的单元, 并允许用户对作为一个组的单元的逻辑值进行修改. 因此, 以一一对应关系把由用户输入的代表新逻辑值的一位值串映射到组中的常数单元, 与以上结合图 8A 和 8B 所说明的情况差不多.

输入新系数值 C_0'-C_7'后, GUI 按照图 5 中的步骤 324 产生部分配置位流. 由于 GUI 既拥有单元位置(从设计数据库获得)又拥有它们的新配置(从设计人员获得), GUI 能够产生配置位流, 该配置位流将实现设计人员指定的设计变化.

部分配置位流仅仅基于设计人员输入的变化, 并包括仅针对 FPGS 中参与设计变化的这些单元的配置信息. 不需要对整个设计进行放置和选路操作, 导出配置位流的变化, 正如采用现有技术方法的情况那样, 见图 4. 相反地, GUI 从用户输入直接进入到部分配置位流. 这种方法显著地缩短了实施设计变化所需的时间, 从而允许设计人员快速实现和测试设计方案的替换.

较佳实施例的 GUI 进一步包括一种对配置成逻辑门的逻辑单元进行修改的模式. 例如, 考虑图 9A 中所示的基于 ROM 的查找表, 它由 ROM 阵列和解码器组成. 查找表通常是用于执行诸如全加器和全减器的逻辑功能的功能发生器, 图 9B 示出其真值表. 图 9C 和 9D 分别示出加法器和减法器的实现, 这里, 图 9A 中解码器的八个输出(0 - 7)在图 9C 和 9D 中示为 A0-A7. 可以看出, 在四

个地方加法器电路不同于减法器电路：加法器中“与”门 G8、12 在减法器中被“或”门 G7、G11 所替代；加法器中“或”门 G24、G28 在减法器中被“与”门 G23、G27 所替代.

利用本发明的方法，设计人员通过简单地对感兴趣的四个门进行定位或识别以及变化它们的相应逻辑单元的配置能够方便地从一种电路切换到另一种电路. 参考图 10A 和 10B，GUI 能够在窗口中显示放置在 FPGA 中的逻辑设计. 通过滚动窗口，能够把阵列中的任何一部分带入到视窗中. 另一方面，GUI 通过把适当的信息输入到图 10A 中所示的单元位置字段或实例名字段中，能够定位部分阵列.

对于以下的讨论假设：初始逻辑设计装入图 9C 中的加法器电路，需要转换到图 9D 中的减法器电路. GUI 访问在加法器电路的初始设计上进行放置和选路操作而产生的设计数据库，并根据 FPGA 的元件显示设计布局. 显示可以由构成 FPGA 的逻辑单元的图标或其它图形表示组成. 另一方面，GUI 可以显示代表由每个逻辑单元所提供逻辑功能的图标. 另一种选择是提供 FPGA 组成的文本表示，然而这一方法在传送和引导有关 FPGA 信息方面不如图形方法有效.

设计人员或是通过滚动或是通过指定门的实例名而显示待改变的部分加法器电路. 图 10A 的屏幕样板上显示了以一部分加法器电路，即门 G6、G8、G10 和 G12 为中心的一部分设计数据库. 可以看出，“或”门 G6 已经指定为 56 行、100 列中的逻辑单元，“与”门 G8 已经指定为 56 行、101 列中的逻辑单元，依此类推. 为了把门 G8 从“与”门变为“或”门，设计人员首先选择该门，例如通过将鼠标器光标置于该门的图标上并接下鼠标器按钮. 图 10B 示出被突出的门 G8，表示该门已经被选定. 单击鼠标器另一按钮（或揿键）产生一弹出菜单，显示一张可能的逻辑门表. 从菜单中选择“或”条目，引起所选门 G8 被配置为“或”门. 以同样的方式改变门 G12. 接着，设计人员通过滚动窗口将门 G24 和 G28 带入视屏. 然后，以如上所述的方式对门 G24 和 G28 进行修改.

当修改完成时，揿一下 OK 按钮. 正如图 5 所示的方法中描述的，GUI 仅根据修改产生部分配置位流. 产生的位流仅由把初始设计的四个单元（加法器电路中的 G8、G12、G24 和 G28）重新定义为减法器的门 G7、G11、G23 和 G27 的特性所需的位串构成. 因此，通过直接访问初始全加器设计的设计数据库和在其上作出变化，设计人员能够避免执行完全放置和选路操作步骤而耗费的时间.

最后要说明一点，可以观察到，以上讨论的特定门的选择和菜单方法不是实施本发明的关键所在．基于图形的其它输入/选择方法也是同样有效的．

通常相对于查找表，如图 9B 中所示的查找表，由查找表实施的逻辑功能完全由表的输出所限定，因此，图 9B 中列 SUM 和 C_{out} 完全限定全加器功能，列 SUB 和 B_{out} 完全限定全减器功能． GUI 无需显示执行特定查找表的特定逻辑单元，而是能够仅列出表输出和允许用户输入对表输出的变化．然后， GUI 指定所需逻辑单元完成新的查找表．以这种方式在更高的抽象水平上代表查找表，以一种更有意义的方法将该表提供给用户，从而对用户隐去意义不大的实施细节，因此，具有更大的适用性．

从图 9C 和 9D 可以看出，由 2 个输入“与”门和 2 个输入“或”门的级联串联连接能够实现查找表．例如，全加器的 SUM 输出由门 G2、 G6、 G10、 G14、 G18、 G22、 G26 和 G30 来实现，而 C_{out} 输出则由门 G4、 G8、 G12、 G16、 G20、 G24、 G28 和 G32 来实现．因此， GUI 简单地通过改变实现查找表的适当门，能够方便地重新配置一个限定第一功能查找表来执行第二功能．

现在，讨论将针对本发明的另一实施例，它允许设计人员对可动态重新配置的 FPGA 进行实时重新配置，即在重新配置时可以运行的系统中的阵列．参考图 11，典型系统 600 包括 CPU 或微处理器 610、 FPGA 620、 EPROM622 或其它一些非易失性 RAM 器件、以及诸如磁盘存储器的数据存储器 630．在 CPU610 上运行的系统软件包括配置 FPGA 的专用应用软件 602 和 FPGA 实用程序 604．在系统引导时，已经编程并在 EPROM622 中的配置位流载入到 FPGA．另一种选择是，能够用存储在数据存储器 630 中的配置位流初始配置 FPGA．

在系统 600 操作期间，软件能够检测有关其操作环境的特定条件．例如，相对于图 8A 和 8B 中所示的数字滤波器设计，软件 602 可以确定需要作调节的滤波器截止频率．相应地，可以对以前积累的数据进行分析，根据分析结果计算新的系数．然后，调用特定的 FPGA 实用程序 604 来创建包含定义新系数的位流的部分配置位流．接着，应用软件 602 调用其它 FPGA 实用程序把部分配置位流下载到 FPGA，由此实现该滤波器设计中的变化．

实时作出的设计变化的范围不受 FPGA 实用程序 604 的限制，但是受装载到 FPGA 的特定设计的复杂性程度的限制．在运行过程中的变化可以由应用软件限制，限于仅改变逻辑常数或者限于各逻辑门．然而，这种限制是应用所特有的，

由计算机电源、存储器、操作环境等的提供能力确定，并不是由于本发明造成的限制.

FPGA 实用程序 604 的一组应用编程接口（ API ）包括:

```
ConstantCell
        Function:
                Generate a configuration bitstream which configures each
                logic cell in the specified range to produce either a
                logic one or a logic zero.
        Parameters:
                xStart - X coordinate of the first logic cell
                yStart - Y coordinate of the first logic cell
                xEnd - X coordinate of the last logic cell
                yEnd - Y coordinate of the last logic cell
                value_string - the corresponding string of 1's and 0's
                buf - pointer to a memory store for storing the
                        configuration bitstring
        Return:
                the number of bytes in the bitstream

DownLoad
        Function:
                Download a configuration bitstring to the FPGA.
        Parameters:
                buf - pointer to the configuration bitstring to be downloaded
        Return:
                n/a

GateCell
        Function:
                Generate a configuration bitstream which reconfigures a
                previously defined logic gate to implement a different logic
                function.  Only the logic function of the gate is changed.
                Other aspects of the gate remain the same, such as the number
                of input and output terminals.

        Parameters:
                x_coord - X coordinate location of the logic cell
                y_coord - Y coordinate location of the logic cell
                logic - the specific logic function: AND, NAND, OR, NOR, XOR,
                        INVERTER, etc.
                buf - pointer to a memory store for storing the configuration
                        bitstring
        Return:
                the number of bits in the bitstream
```

```
ReadCell
        Function:
                Return the current configuration for the specified logic cell.
        Parameters:
                x_coord - X coordinate location of the logic cell
                y_coord - Y coordinate location of the logic cell
                buf - pointer to a memory store for storing the cell
                      configuration
        Return:
                n/a
```

以下的 C 语言代码片段说明如何在典型应用中使用这些实用程序. 给出的例子针对图 8A 的数字滤波器，说明如何根据外部操作环境来调节系数 C_0-C_7.

```
:
:

/* ... a determination has been made that the filter
                    needs adjustment ... */

:
:

/* read the current values of filter coefficients C0 - C7 */
for( i = 0; i < 8; ++i ) /* each coefficient */
      for( j = 0, j < 8; ++j )/* each bit */
            ReadCell(xloc_coeff[i][j], yloc_coeff[i][j],
                  current_buf[i] + j);

/* derive new coefficient values and store in new_buf */

:
:

/* create configuration bitstreams for the new values */
z = 0; /* point to beginning of bstream */
for( i = 0; i < 8; ++i )
{
      z1 = ConstantCell(xloc_coeff[i][0], yloc_coeff[i][0],
            xloc_coeff[i][7], yloc_coeff[i][7], new_buf[i],
                  bstream + z);
      z += z1; /* point to end of bstream */
}

/* download bitstream to FPGA */
DownLoad(bstream);
```

根据本发明的对FPGA重新配置的系统允许FPGA的重新配置实时进行和允许仅对FPGA中的一部分进行重新配置．此外，本发明允许应用软件根据外部操作环境确定设计中部分的变化．这种方法的优点在于：设计人员不需要事前确定所有可能的备用设计及其相应的配置位流．而是，应用软件能够确定设计变化，在运行过程中创建部分配置位流，根据它们所出现的环境中的条件而实施这些变化．

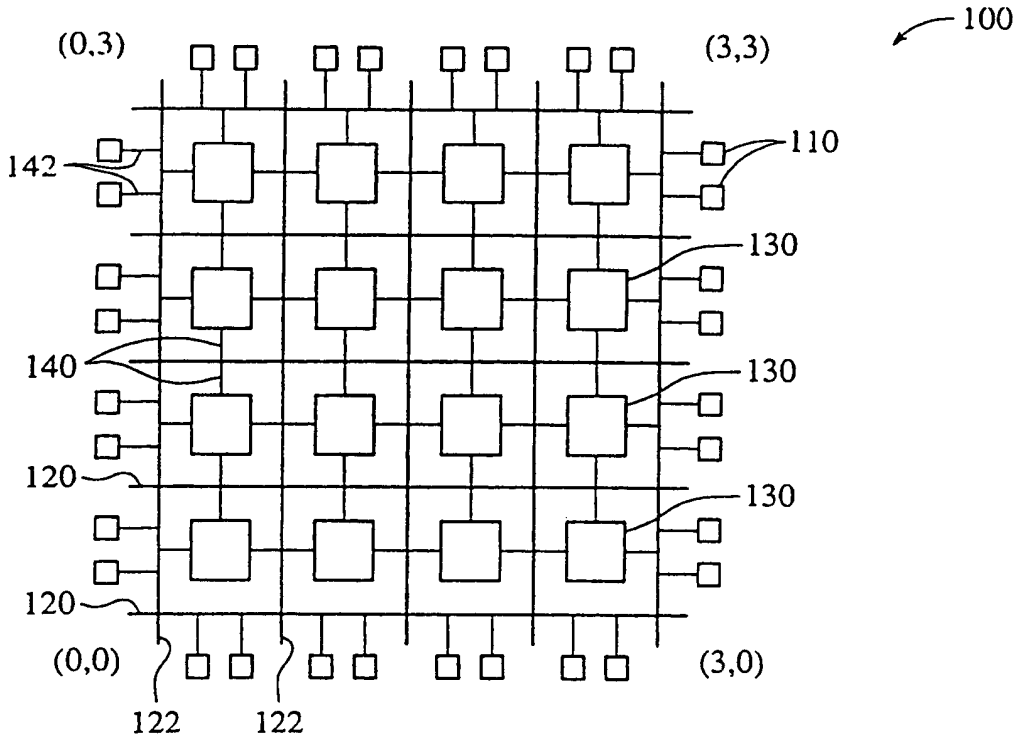

100
(0,3)
(3,3)
110
142
130
140
130
120
130
120
(0,0)
(3,0)
122
122

图 1

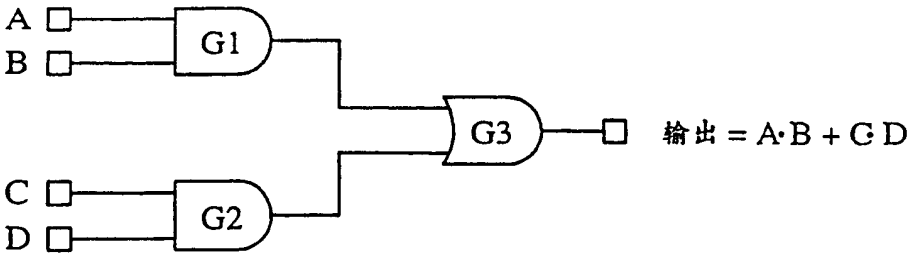

A
B
G1
G3
输出 = A·B + C·D
C
D
G2

图 2

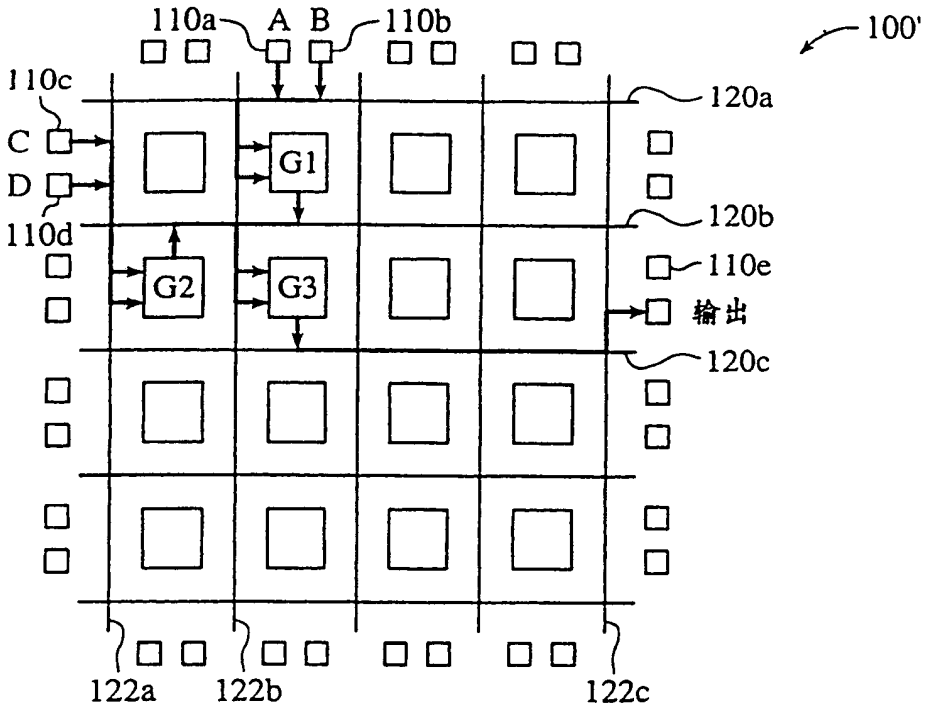

110a
A
B
110b
100'
110c
C
D
110d
120a
G1
120b
G2
G3
110e
输出
120c
122a
122b
122c

图 3A

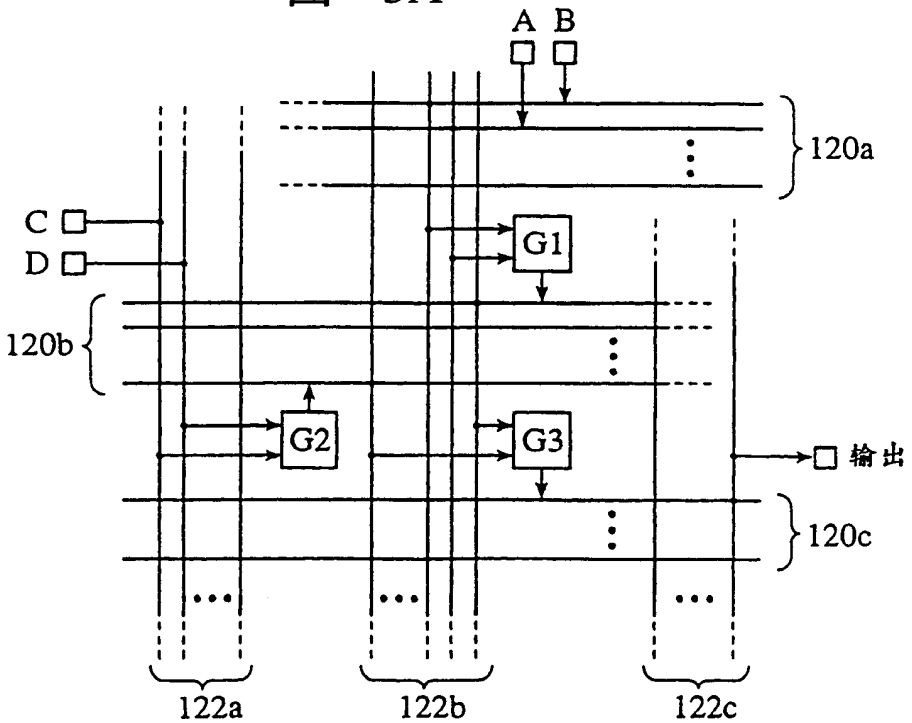

A
B
120a
C
D
G1
120b
G2
G3
输出
120c
122a
122b
122c

图 3B

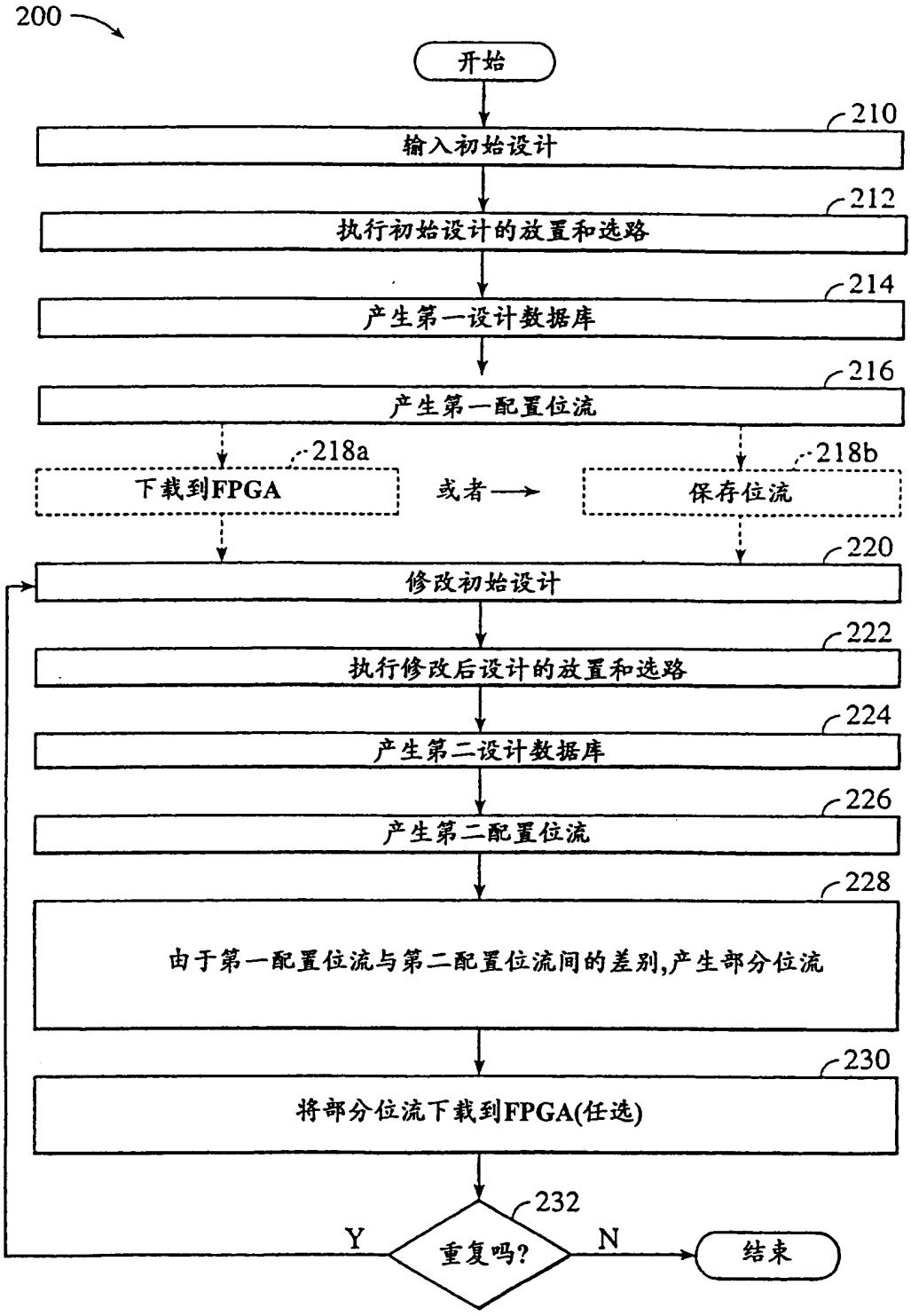

200
开始
210
输入初始设计
212
执行初始设计的放置和选路
214
产生第一设计数据库
216
产生第一配置位流
218a
下载到FPGA
或者
218b
保存位流
220
修改初始设计
222
执行修改后设计的放置和选路
224
产生第二设计数据库
226
产生第二配置位流
228
由于第一配置位流与第二配置位流间的差别,产生部分位流
230
将部分位流下载到FPGA(任选)
232
重复吗?
Y
N
结束

图 4

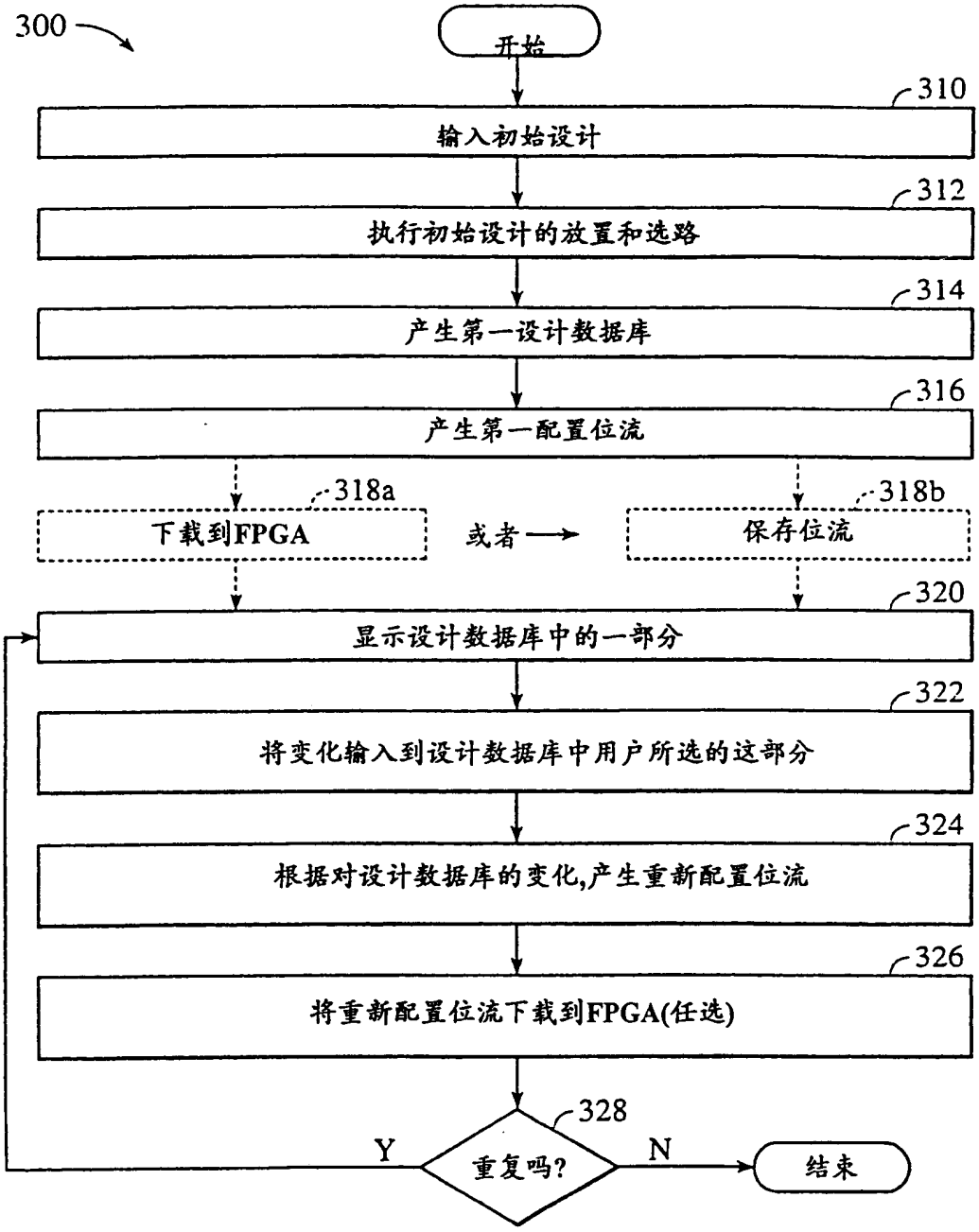

300
开始
310
输入初始设计
312
执行初始设计的放置和选路
314
产生第一设计数据库
316
产生第一配置位流
318a
下载到FPGA
或者
318b
保存位流
320
显示设计数据库中的一部分
322
将变化输入到设计数据库中用户所选的这部分
324
根据对设计数据库的变化,产生重新配置位流
326
将重新配置位流下载到FPGA(任选)
328
Y
重复吗?
N
结束

图 5

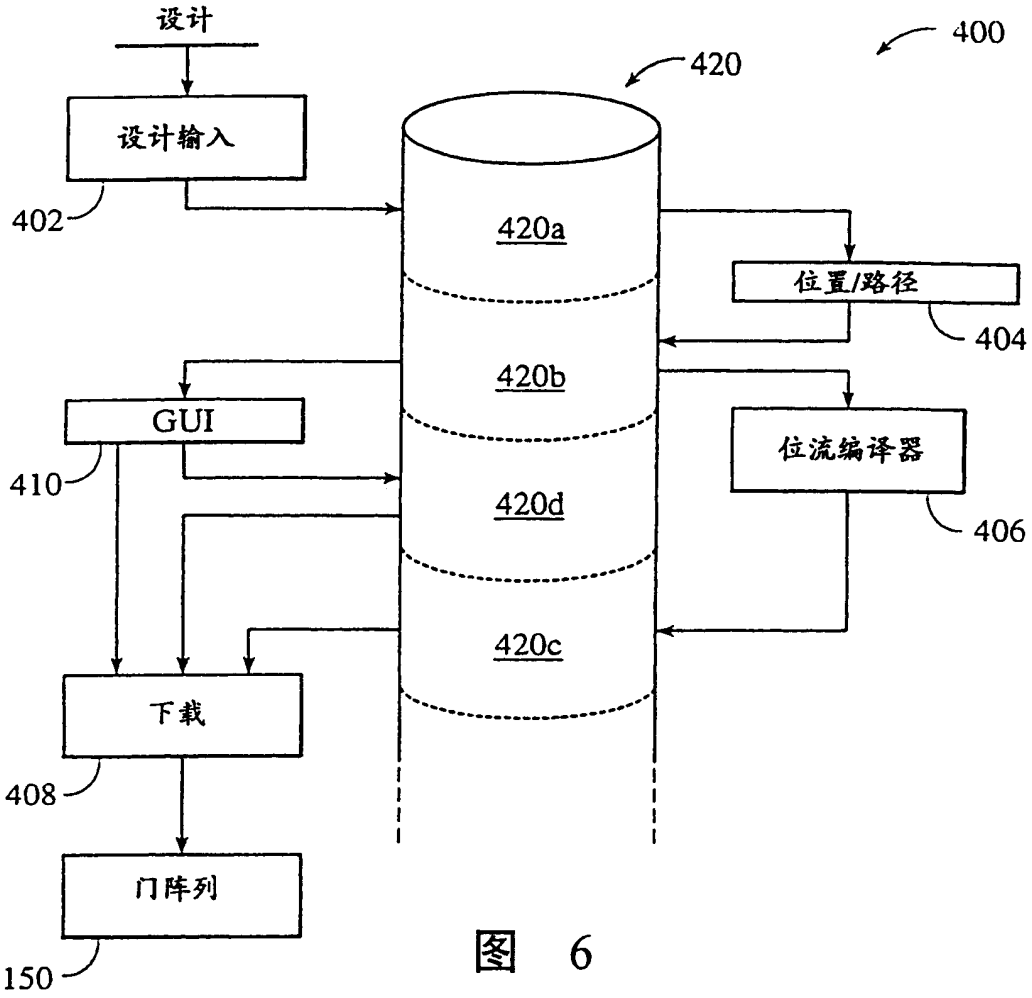

设计
设计输入
402
400
420
420a
420b
420d
420c
位置/路径
404
位流编译器
406
GUI
410
下载
408
门阵列
150

图 6

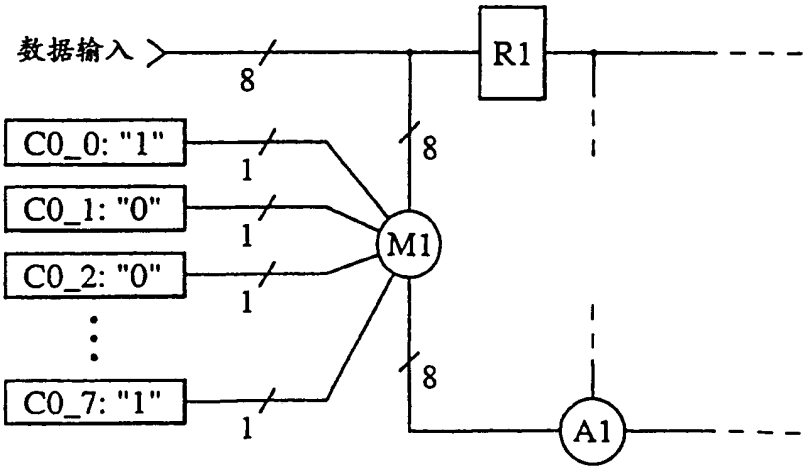

数据输入
8
R1
C0_0: "1"
C0_1: "0"
C0_2: "0"
C0_7: "1"
1
1
1
1
8
8
M1
A1

图 7C

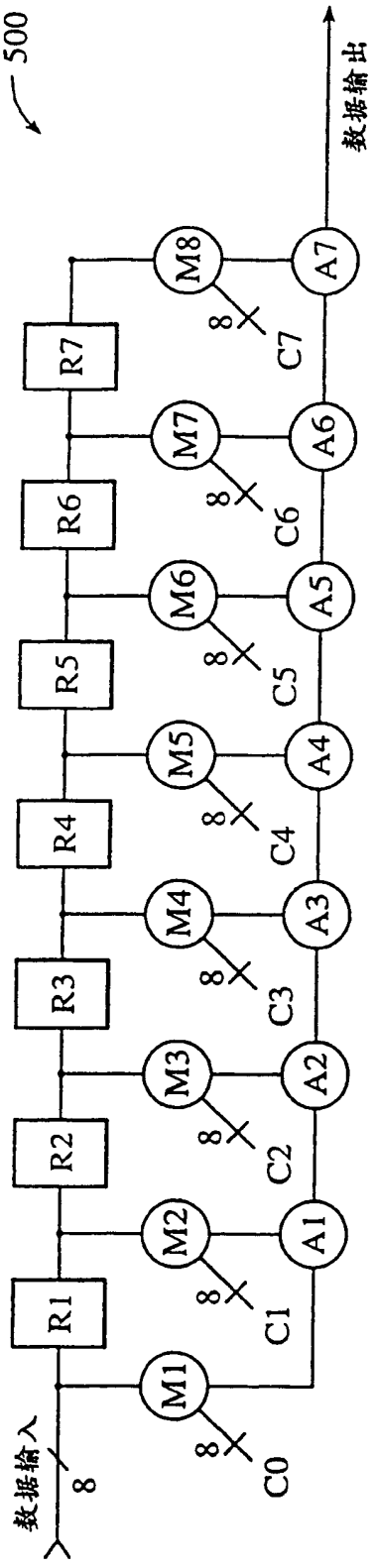

500
数据输入
8
R1
R2
R3
R4
R5
R6
R7
M1
M2
M3
M4
M5
M6
M7
M8
A1
A2
A3
A4
A5
A6
A7
数据输出
8
C0
C1
C2
C3
C4
C5
C6
C7

图 7A

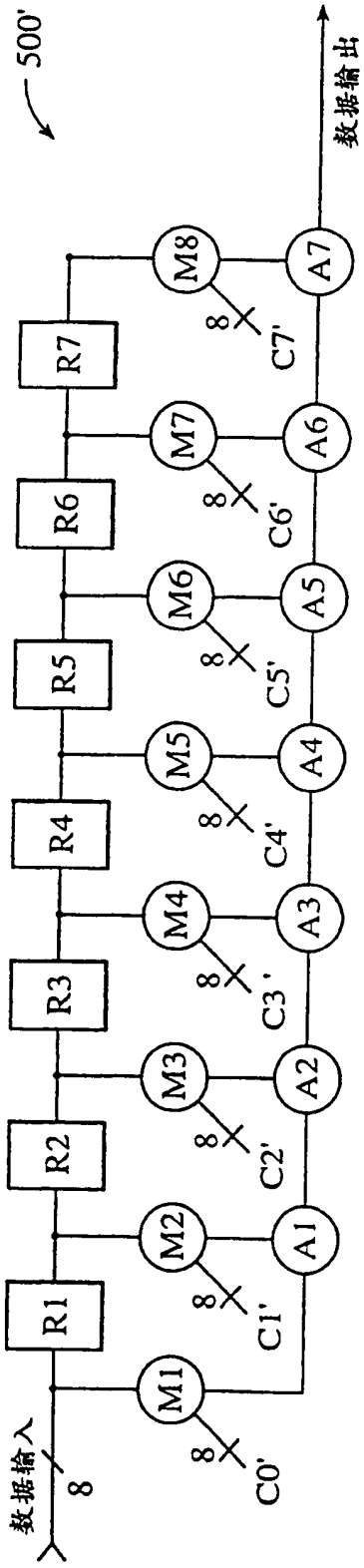

500'
数据输入
8
R1
R2
R3
R4
R5
R6
R7
M1
M2
M3
M4
M5
M6
M7
M8
A1
A2
A3
A4
A5
A6
A7
数据输出
8
C0'
C1'
C2'
C3'
C4'
C5'
C6'
C7'

图 7B

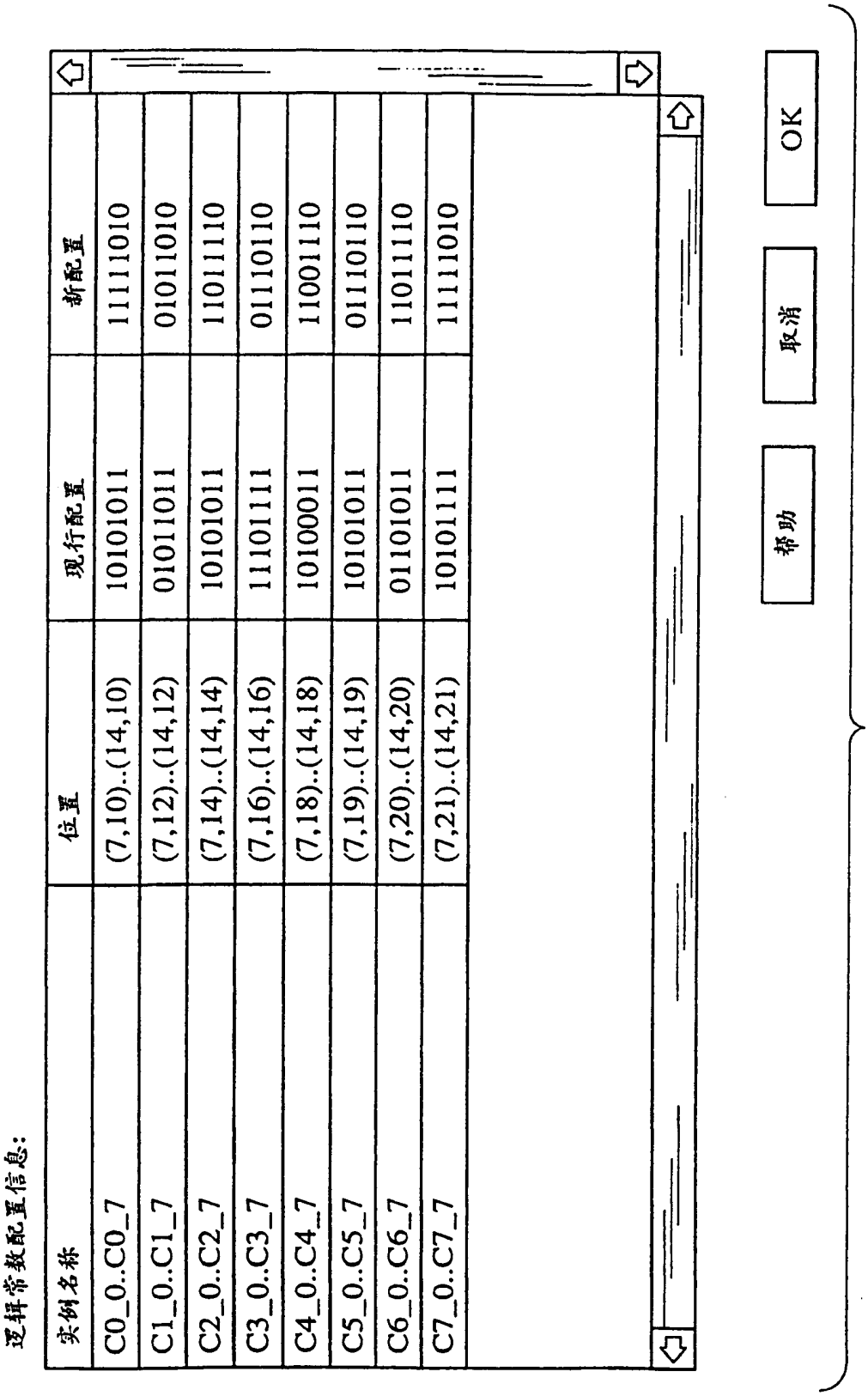

逻辑常数配置信息:
实例名称
位置
现行配置
新配置
C0_0..C0_7
(7,10)..(14,10)
10101011
11111010
C1_0..C1_7
(7,12)..(14,12)
01011011
01011010
C2_0..C2_7
(7,14)..(14,14)
10101011
11011110
C3_0..C3_7
(7,16)..(14,16)
11101111
01110110
C4_0..C4_7
(7,18)..(14,18)
10100011
11001110
C5_0..C5_7
(7,19)..(14,19)
10101011
01110110
C6_0..C6_7
(7,20)..(14,20)
01101011
11011110
C7_0..C7_7
(7,21)..(14,21)
10101111
11111010
帮助
取消
OK

图 8A

逻辑常数配置信息:

实例名称	位置	现行配置	新配置
C0_0..C0_7	(7,10)..(14,10)	10101011	11111010
C1_0..C1_7	(7,12)..(14,12)	01011011	01011010
C2_0..C2_7	(7,14)..(14,14)	10101011	11011110
C3_0..C3_7	(7,16)..(14,16)	11101111	01110110
C4_0..C7_0	(7,18)..(7,21)	1101	1011
C4_1..C7_1	(8,18)..(8,21)	0010	1111
C4_2..C7_2	(9,18)..(9,21)	1111	0101
C4_3..C7_3	(10,18)..(10,21)	0000	0111
C4_4..C7_4	(11,18)..(11,21)	0111	1011
C4_5..C7_5	(12,18)..(12,21)	0001	1110
C4_6..C7_6	(13,18)..(13,21)	1111	1111
C4_7..C7_7	(14,18)..(14,21)	1111	0000

帮助　取消　OK

图　8B

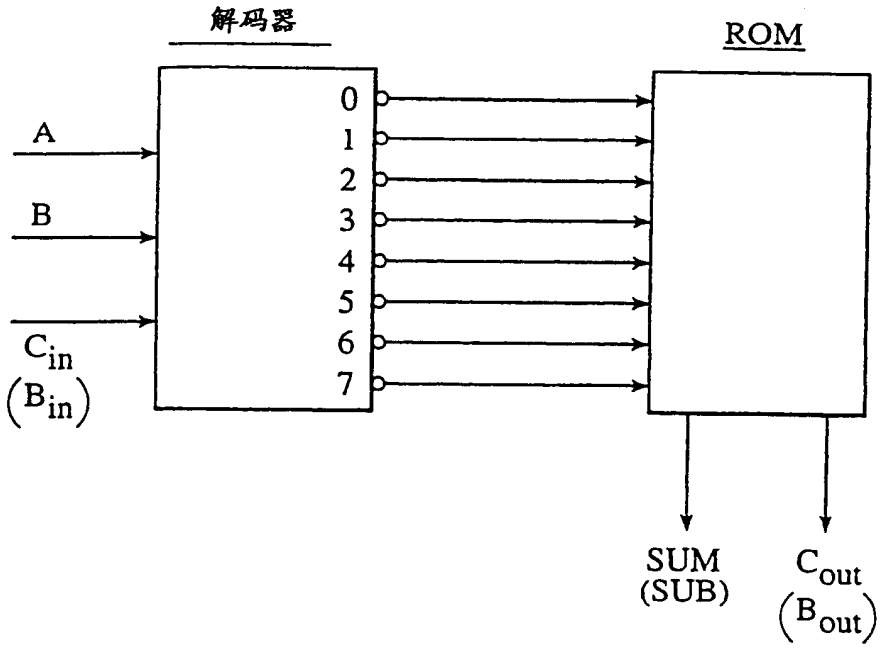


图 9A

全加器					全减器				
A	B	C_{in}	Sum	C_{out}	A	B	B_{in}	Sub	B_{out}
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	1	0	1	0	0	1	0	1	1
0	1	1	0	1	0	1	1	0	1
1	0	0	1	0	1	0	0	1	0
1	0	1	0	1	1	0	1	0	0
1	1	0	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1

图 9B

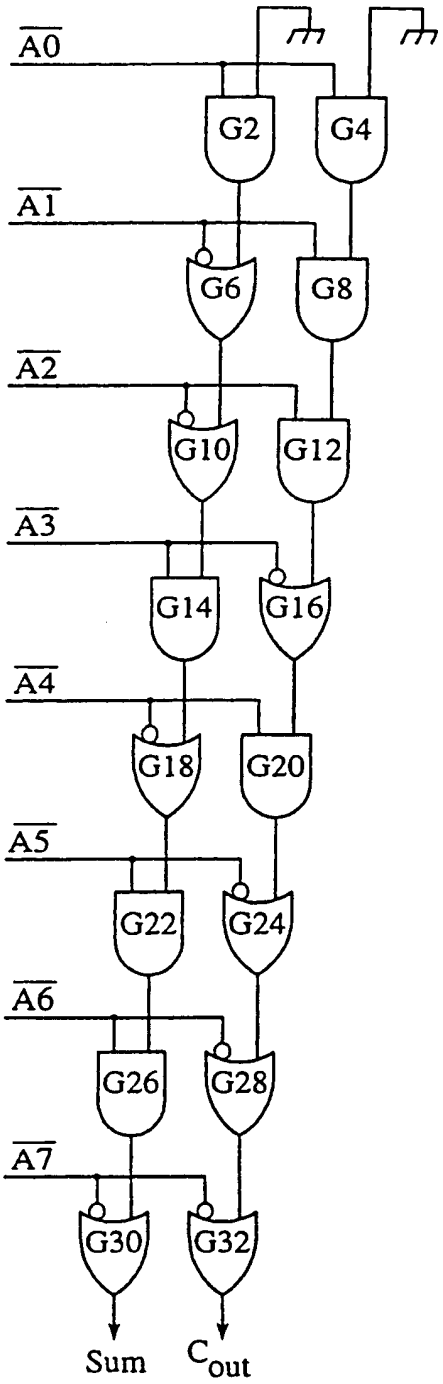

A0
G2
G4
A1
G6
G8
A2
G10
G12
A3
G14
G16
A4
G18
G20
A5
G22
G24
A6
G26
G28
A7
G30
G32
Sum
Cout

图 9C

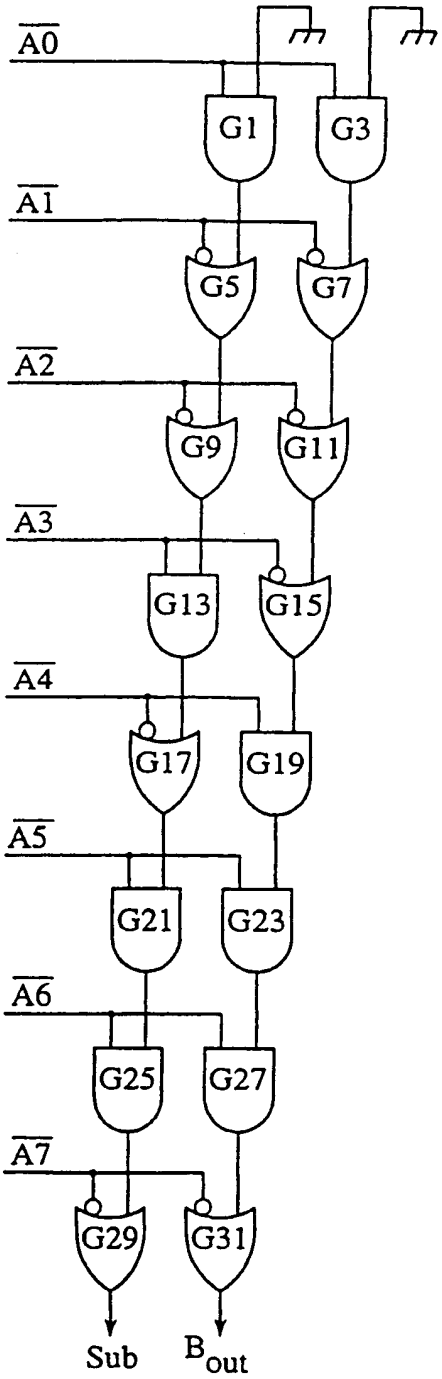

A0
G1
G3
A1
G5
G7
A2
G9
G11
A3
G13
G15
A4
G17
G19
A5
G21
G23
A6
G25
G27
A7
G29
G31
Sub
Bout

图 9D

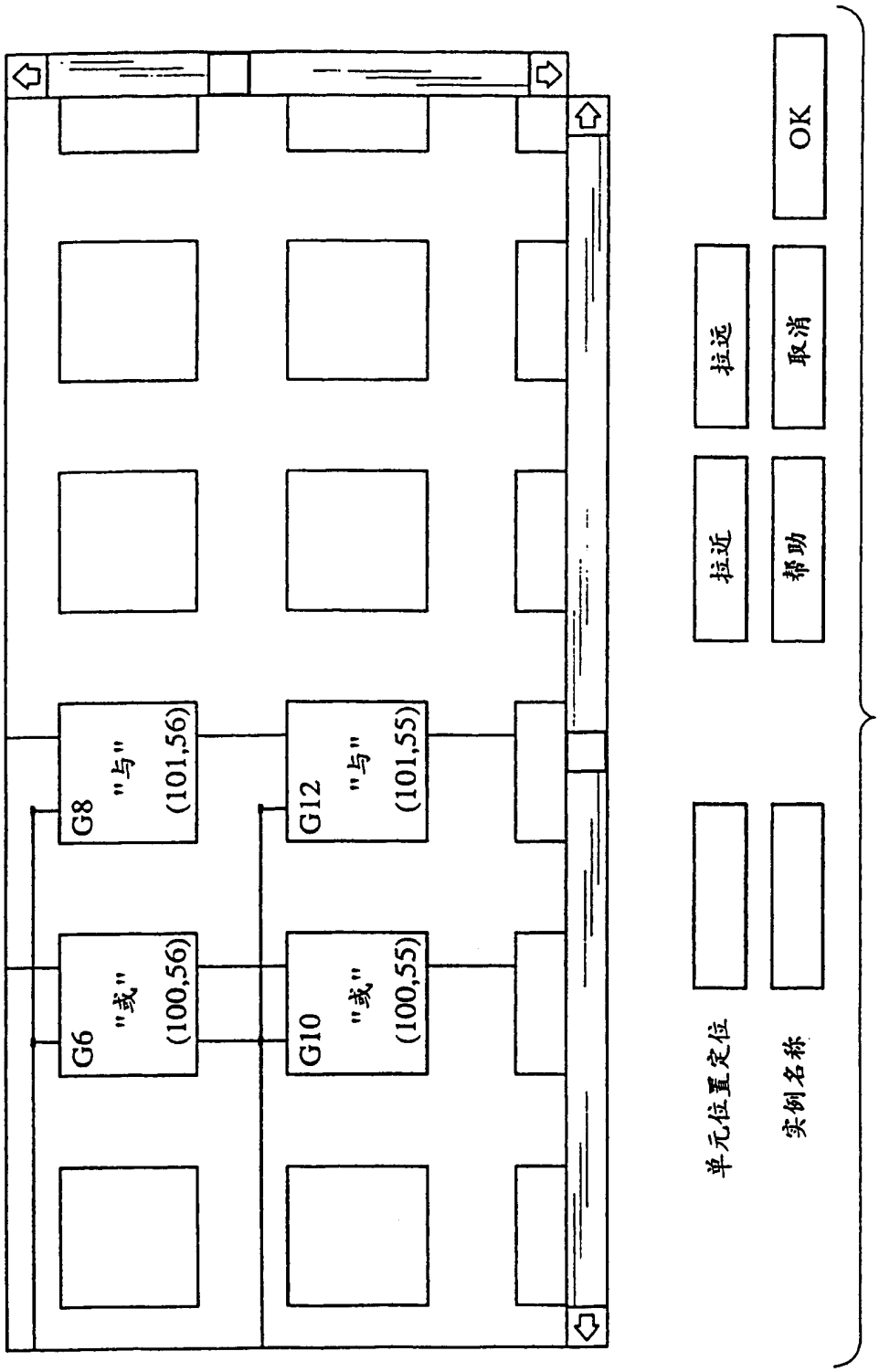

G8
"与"
(101,56)
G12
"与"
(101,55)
G6
"或"
(100,56)
G10
"或"
(100,55)
拉近
帮助
拉远
取消
OK
单元位置定位
实例名称

图 10A

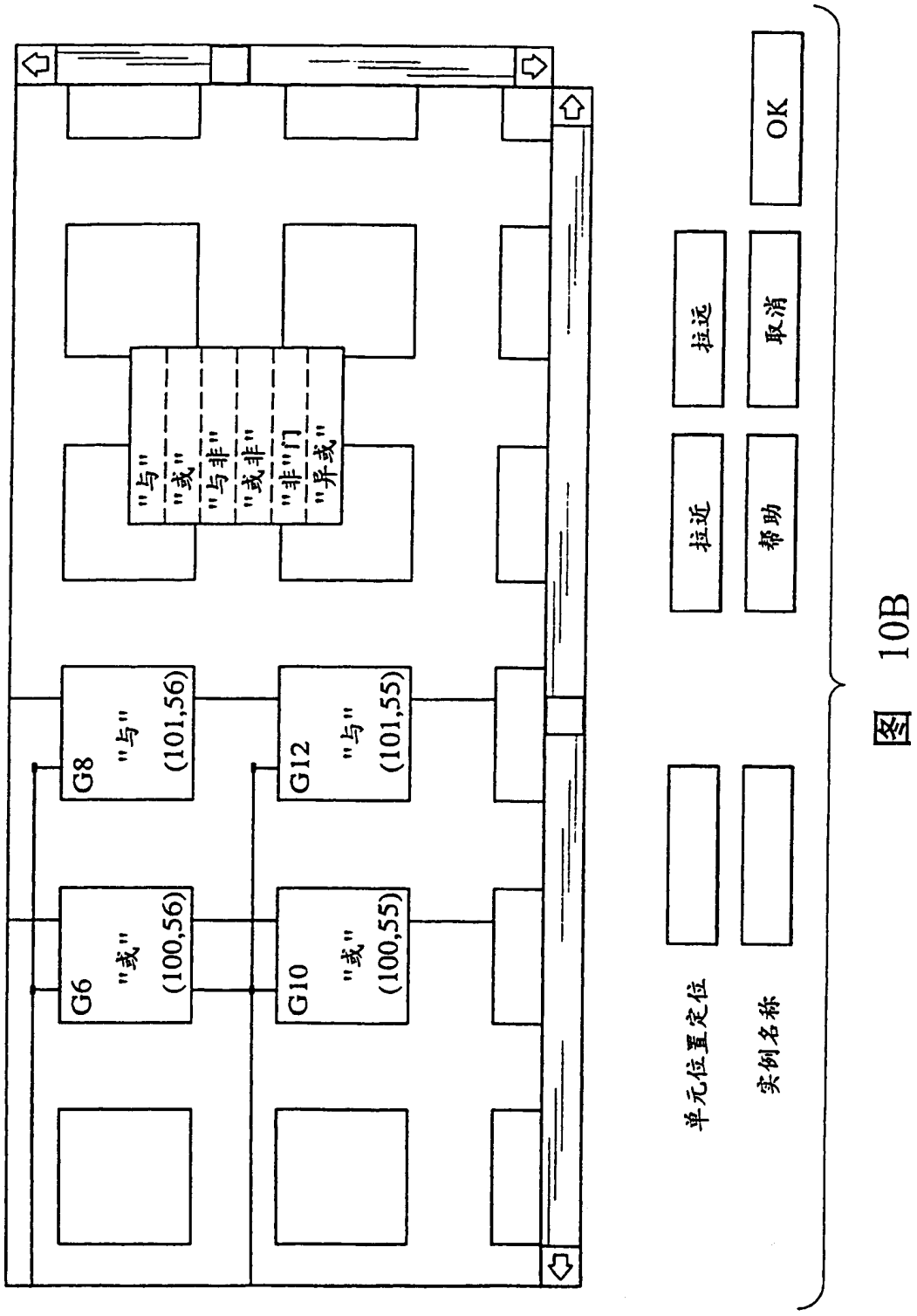

"与"
"或"
"与非"
"或非"
"非"门
"异或"
G8
"与"
(101,56)
G12
"与"
(101,55)
G6
"或"
(100,56)
G10
"或"
(100,55)
拉远
取消
OK
拉近
帮助
单元位置定位
实例名称

图 10B

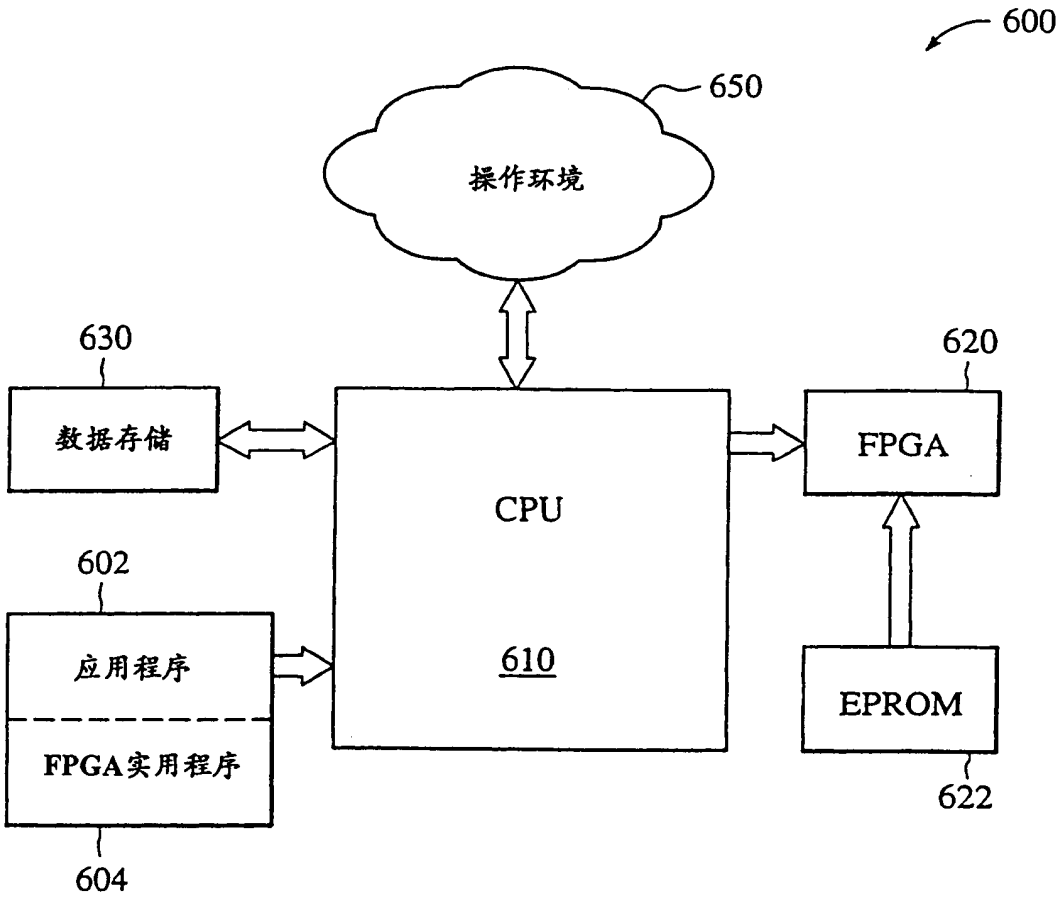

600
650
操作环境
630
620
数据存储
FPGA
CPU
602
610
应用程序
FPGA实用程序
EPROM
622
604

图 11