[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/50 (2006.01)

[12] 发明专利说明书

专利号 ZL 200510078068.2

[45] 授权公告日 2008 年 11 月 19 日

[11] 授权公告号 CN 100435154C

[22] 申请日 2005.6.14
[21] 申请号 200510078068.2
[30] 优先权
[32] 2004.7.29 [33] US [31] 10/902,595
[73] 专利权人 国际商业机器公司
地址 美国纽约阿芒克
[72] 发明人 沃尔夫冈·罗斯纳
德里克·E·威廉斯
[56] 参考文献
US 5995736 A 1999.11.30
CN 1382280 A 2002.11.27
基于 VHDL 语言的数字电路设计. 刘淑荣，蒋彬. 长春工程学院学报（自然科学报），第 3 卷第 4 期. 2002
数字逻辑电路的描述及模块化综合方法. 钱培怡，于德泳. 系统工程与电子技术，第 24 卷第 3 期. 2002
审查员 刘长勇
[74] 专利代理机构 北京市柳沈律师事务所
代理人 蒲迈文 黄小临

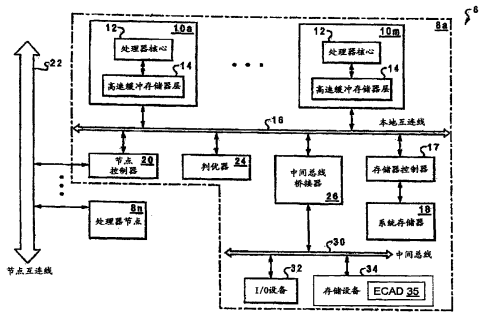
权利要求书 4 页 说明书 96 页 附图 57 页

[54] 发明名称

支持配置实体的选择性表示的配置说明语言的方法和系统

[57] 摘要

在至少一个硬件定义语言(HDL)文件中，指定包含数字系统的功能部分的至少一个设计实体。 设计实体逻辑上包含具有独立的数个不同可能锁存器值的锁存器。 借助于一个或多个文件中的一个或多个语句，将配置实体与锁存器相联系。 配置实体含有数个不同设置和每个设置反映数个不同可能值的哪一个被装入相关锁存器中。 在一个或多个文件中还定义配置实体的至少一个实例的控制值集。 控制值集指示配置实体实例的当前设置的表示受到限制的至少一个控制值。 此后，响应至少表示数字系统的部分状态的请求，通过参照指示控制值集的配置数据库，将配置实体实例的当前设置排除在表示之外。



1. 一种在一个或多个文件中指定可选择表示数字系统的方法，所述方法包括:

在所述一个或多个文件当中的至少一个硬件定义语言文件中，指定包含数字系统的功能部分的至少一个设计实体，所述至少一个设计实体逻辑上包含具有独立的数个不同可能锁存器值的锁存器;

借助于所述一个或多个文件中的一个或多个语句，将配置实体与所述锁存器相联系，其中，所述配置实体含有数个不同设置，和每个设置反映数个不同可能值的哪一个被装入所述相关锁存器中；和

借助于所述一个或多个文件中的一个或多个语句，为所述配置实体的至少一个实例定义控制值集，其中，所述控制值集指示配置实体实例的当前设置的表示受到限制的至少一个控制值。

2. 根据权利要求 1 所述的方法，其中，所述配置实体包括一含有数个拨号器的拨号器组，和其中所述控制值集指示所述拨号器组内所述数个拨号器当中的一个拨号器的设置。

3. 根据权利要求 1 所述的方法，其中所述至少一个控制值包括所述配置实体实例的至少一个设置。

4. 根据权利要求 3 所述的方法，其中所述至少一个控制值包括另一个配置实体实例的至少一个设置。

5. 根据权利要求 1 所述的方法，进一步包括编译所述至少一个硬件定义语言文件，以生成所述数字系统的模拟模型，所述模拟模型包括所述设计实体和所述锁存器。

6. 根据权利要求 5 所述的方法，所述编译进一步包括从所述一个或多个语句中生成配置数据库，所述配置数据库包括代表所述配置实体的至少一个数据结构和指示所述控制值集。

7. 根据权利要求 6 所述的方法，进一步包括:

响应至少表示数字系统的部分状态的请求，根据所述配置数据库，表示至少一个其它配置实体实例的状态，和参照所述配置数据库中的至少一个数据结构，将配置实体实例的当前设置排除在表示之外。

8. 一种在一个或多个文件中指定可选择表示数字系统的数据处理系统，

所述数据处理系统包括:

有助于处理数据的处理装置;

在所述一个或多个文件当中的至少一个硬件定义语言文件中，指定包含数字系统的功能部分的至少一个设计实体的装置，所述至少一个设计实体逻辑上包含具有独立的数个不同可能锁存器值的锁存器;

借助于所述一个或多个文件中的一个或多个语句，将配置实体与所述锁存器相联系的装置，其中，所述配置实体含有数个不同设置，和每个设置反映数个不同可能值的哪一个被装入所述相关锁存器中；和

借助于所述一个或多个文件中的一个或多个语句，为所述配置实体的至少一个实例定义控制值集的装置，其中，所述控制值集指示配置实体实例的当前设置的表示受到限制的至少一个控制值。

9. 根据权利要求 8 所述的数据处理系统，其中，所述配置实体包括一含有数个拨号器的拨号器组，和其中所述控制值集指示所述拨号器组内所述数个拨号器当中的一个拨号器的设置。

10. 根据权利要求 8 所述的数据处理系统，其中所述至少一个控制值包括所述配置实体实例的至少一个设置。

11. 根据权利要求 10 所述的数据处理系统，其中所述至少一个控制值包括另一个配置实体实例的至少一个设置。

12. 根据权利要求 8 所述的数据处理系统，进一步包括编译所述至少一个硬件定义语言文件，以生成所述数字系统的模拟模型的编译器，所述模拟模型包括所述设计实体和所述锁存器。

13. 根据权利要求 8 所述的数据处理系统，进一步包括从所述一个或多个语句中生成配置数据库的装置，所述配置数据库包括代表所述配置实体的至少一个数据结构和指示所述控制值集。

14. 根据权利要求 13 所述的数据处理系统，进一步包括:

接口装置，响应至少表示数字系统的部分状态的请求，参照所述配置数据库，表示至少一个其它配置实体实例的状态，和参照所述配置数据库中的至少一个数据结构，将配置实体实例的当前设置排除在表示之外。

15. 一种表示与作为模拟系统和硬件系统集合当中的一个的系统有关的信息的方法，所述方法包括:

在与系统相联系的配置数据库中保存将配置实体的实例与所述系统内的

锁存器相联系的至少一个数据结构，其中，所述配置实体含有每一个都反映数个不同可能值的哪一个被装入所述相关锁存器中的数个不同设置，所述至少一个数据结构进一步指示所述配置实体的实例的控制值集，其中所述控制值集指示配置实体实例的当前设置的表示受到限制的至少一个控制值；和

响应至少表示系统的部分状态的请求，参照所述配置数据库，表示至少一个其它配置实体实例的状态，和参照所述配置数据库中的所述至少一个数据结构所指的所述控制值集，将配置实体实例的当前设置排除在表示之外。

16. 根据权利要求 15 所述的方法，其中，所述配置实体包括一含有数个拨号器的拨号器组，和其中，所述控制值集指示所述拨号器组内所述数个拨号器当中的一个拨号器的设置。

17. 根据权利要求 15 所述的方法，其中所述至少一个控制值包括所述配置实体实例的至少一个设置。

18. 根据权利要求 17 所述的方法，其中所述至少一个控制值包括另一个配置实体实例的至少一个设置。

19. 一种有选择地表示与作为模拟系统和硬件系统集合当中的一个的正被测试的系统有关的信息的数据处理系统，所述数据处理系统包括:

与系统相联系的配置数据库装置，所述配置数据库装置的配置数据库包含将配置实体的实例与所述系统内的锁存器相联系的至少一个数据结构，其中所述配置实体含有每一个都反映数个不同可能值的哪一个被装入所述相关锁存器中的数个不同设置，所述至少一个数据结构进一步指示所述配置实体的实例的控制值集，其中所述控制值集指示配置实体实例的当前设置的表示受到限制的至少一个控制值；和

接口装置，响应至少表示系统的部分状态的请求，参照所述配置数据库，表示至少一个其它配置实体实例的状态，和参照所述配置数据库中的所述至少一个数据结构所指的所述控制值集，将配置实体实例的当前设置排除在表示之外。

20. 根据权利要求 19 所述的数据处理系统，其中，所述配置实体包括一含有数个拨号器的拨号器组，和其中所述控制值集指示所述拨号器组内所述数个拨号器当中的一个拨号器的设置。

21. 根据权利要求 19 所述的数据处理系统，其中，所述至少一个控制值包括所述配置实体实例的至少一个设置。

22. 根据权利要求 21 所述的数据处理系统，其中，所述至少一个控制值包括另一个配置实体实例的至少一个设置。

支持配置实体的选择性表示的配置说明语言的方法和系统

技术领域

本发明一般涉及设计、模拟和配置数字设备、模块和系统，尤其涉及计算机辅助设计、模拟、和配置通过硬件描述语言（HDL）模型描述的数字设备、模块和系统的方法和系统。

背景技术

在典型数字设计过程中，核实数字设计的逻辑正确性和调试设计（如果有必要）是在开发电路布线图之前进行的设计过程的重要步骤。尽管完全可以通过实际制作数字设计来测试数字设计，但是，部分由于集成电路制造所需的时间和费用，通常在计算机上模拟数字设计来核实和调试数字设计，尤其通过集成电路实现的那些数字设计。

在典型自动设计过程中，电路设计者将利用诸如VHDL之类的硬件描述语言（HDL）对待模拟的数字设计的高级描述输入电子计算机辅助设计（ECAD）系统中，从而生成各种各样电路块和它们的互连的数字表示。在数字表示中，整个电路设计经常被划分成下文称为设计实体的较小部分，它们往往由不同设计者分别设计，然后以分层方式组合起来变成整个模型。这种分层设计技术在管理整个设计的极其复杂性方面非常有用，并且，在模拟期间有助于错误检测。

ECAD系统将设计的数字表示编译成具有最适合模拟的形式的模拟模型。然后，模拟器运用模拟模型以检测数字设计中的逻辑错误。

模拟器通常是通过应用代表数字系统的输入的一系列输入激励对模拟模型进行操作的软件工具。模拟器生成电路对输入激励的响应的数字表示，然后，输入激励的响应可以在显示屏上被看作一系列值，或者，往往由独立软件程序作进一步翻译，以图形方式呈现在显示屏上。模拟器可以在通用计算机上运行，也可以在为模拟专门设计的另一个电子设备上运行。完全以软件方式在通用计算机上运行的模拟器被称为“软件模拟器”，和借助于专门设计

电子设备运行的模拟器被称为“硬件模拟器”。

随着数字设计越来越复杂，一般在几个抽象层面上，例如，在功能、逻辑和电路层上模拟数字设计。在功能层上，用寄存器、加法器、存储器和其它功能单元之间的一系列事件处理的术语描述系统操作。功能层上的模拟用于核实数字系统的高级设计。在逻辑层上，用诸如逻辑门和触发器之类的逻辑单元的术语描述数字系统。逻辑层上的模拟用于核实逻辑设计的正确性。在电路层上，用诸如晶体管、电阻、电容、和其它这样的器件之类它的电路部件的术语描述每个逻辑门。电路层上的模拟提供了有关电压电平和开关速度的详细信息。

为了核实任何给定模拟运行的结果，将称为参考模型的用诸如 C 或 C++ 之类的高级语言写成的定制程序写成处理输入激励（也称为测试矢量），以生成模拟运行的预期结果。然后，由模拟器对照模拟执行模型运行测试矢量。然后，将模拟运行的结果与通过参考模型预测的结果相比较，以检测标记为错误的差异。这样的模拟检验在现有核实技术中被称作“端到端”检验。

在当代数据处理系统，尤其大型服务器类计算机系统中，必须装入以配置操作（或模拟）系统的锁存器的数量显著增加。配置锁存器增加的一个原因是许多芯片被设计成支持多种不同配置和操作模式，以提高制造者利润率和简化系统设计。例如，存储器控制器一般需要基本配置信息来适当地交接不同类型、大小、和工作频率的存储卡。

配置锁存器增加的第二个原因是处理器和其它集成电路芯片内的晶体管预算甚至也增加。可用在下一代芯片中的附加晶体管往往用在重复复制现有功能单元上，以便提高容错性和并行性。但是，由于经过芯片内连线引起的传输等待时间不随功能逻辑单元的工作频率的增加成正比地增加，所以一般认为不希望将所有相似功能单元的配置锁存器集中在一起。因此，尽管经常相同地配置重复功能单元的所有实例，但每个实例往往被设计成拥有它自己的配置锁存器复制品。因此，配置只有几个有效值（例如，总线时钟频率和处理器时钟频率之间的比率）的工作参数可能涉及到在处理器芯片中设置数百个配置锁存器。

传统上，配置锁存器和它们的允许值域通过创建和维护都令人乏味的易出错纸质文档指定。难以维护精确配置文档和需要为配置锁存器而付出总的原因是一个公司内的不同部门（例如，功能模拟团队、实验室调试团队、和

一个或多个定制固件团队）往往根据配置文档独立开发配置软件。由于每个部门独立开发配置软件，每个团队可能引入它自己的错误和应用它自己的术语学和命名惯例。因此，由不同团队开发的配置软件不兼容，不同团队也不容易共享它。

除了前面的缺点之外，在开发配置代码的过程中，传统配置软件在编码上非常令人乏味。尤其，用于将各种各样配置位编入文档的词汇往往相当麻烦。例如，配置代码必须每个配置锁存器位指定可能包括50个或更多个ASCII字符的锁存器全名。另外，必须分别指定每组配置锁存器的有效二进制位模式。

在模拟和硬件数字系统的模拟和调试中遇到的另一个问题是模拟或硬件数字系统的状态难以以方便形式表示。传统上，调试数字系统的人员将获得数字系统内的数千个锁存器、寄存器或配置结构的一“群”原始值。然后，人工地或利用一个脚本处理这群数据，除去大量“不感兴趣”的数据，估计会留下有助于用户调试硬件系统的一组可管理数据（它们可能被进一步分析和/或转化）。

尽管这种查明数字设计的状态的传统技术降低了分析和翻译系统“群”的结果的难度，但负责调试设计的个体往往不知道底层锁存器和配置结构的细节，因此，将许多设计留给“逆向工程师”去理解它的操作，或向原始设计团队寻求帮助。此外，由于一个设计内信号和锁存器的名称往往随设计的修订而改变，为翻译系统的状态和有助于调试而开发的脚本和其它调试工具不能重新用于多个设计。

鉴于前面的情况，本发明认识到，提供配置和表示通过HDL模型，尤其，按照设计者或其它用户的偏爱支持配置信息的选择性表示的那一种描述的数字系统的状态的改进方法是有用的和人们所希望的。

发明内容

本发明公开了指定诸如一个集成电路或一批互连集成电路之类的数字系统的配置的改进方法、系统、和程序产品。根据一种方法，在至少一个硬件定义语言（HDL）文件中指定包括数字系统的功能部分的至少一个设计实体。设计实体逻辑上包含具有独立的数个不同可能锁存器值的锁存器。借助于一个或多个文件中的一个或多个语句，将配置实体与锁存器相联系。配置实体

含有数个不同设置，和每个设置反映数个不同可能值的哪一个被装入相关锁存器中。在一个或多个文件中还定义用于配置实体的至少一个实例的控制值集。控制值集表示配置实体实例的当前设置的表示受到限制的至少一个控制值。此后，响应至少表示数字系统的部分状态的请求，通过参照表示控制值集的配置数据库，将配置实体实例的当前设置排除在表示之外。

本发明的所有目的、特征和优点可从如下的详述中清楚看出。

附图说明

被认为是本发明特征的新特征阐述在所附的权利要求书中。但是，通过结合附图阅读例示性实施例的如下详细描述，可以更好地理解本发明，以及优选使用模式，在附图中：

图 1 是可以用于实现本发明的数据处理系统的高级方块图；

图 2 是用 HDL 代码描述的设计实体的示意性表示；

图 3 例示了包括数个分层排列设计实体的示范性数字设计；

图 4A 描绘了按照本发明的包括嵌入式配置说明语句的示范性 HDL 文件；

图 4B 例示了按照本发明的包括嵌入式配置文件引用语句的示范性 HDL 文件，嵌入式配置文件引用语句引用包含配置说明语句的外部配置文件；

图 5A 是按照本发明的 LDial 原语的示意性表示；

图 5B 描绘了按照本发明的包括例示 LDial 的数个分层排列设计实体的示范性数字设计；

图 5C 例示了包括 LDial 应用于配置设计分层结构的多个不同层上的信号状态的数个分层排列设计实体的示范性数字设计；

图 5D 是按照本发明的 Switch 的示意性表示；

图 6A 是按照本发明的 IDial 的示意性表示；

图 6B 是按照本发明的存在分输出端的 IDial 的示意性表示；

图 7A 是按照本发明的应用于控制其它 Dial 的 CDial 的示意性表示；

图 7B 描绘了包括 CDial 应用于控制用于配置信号状态的低层 Dial 的数个分层排列设计实体的示范性数字设计；

图 7C 是按照本发明的 Register 的示意性表示；

图 8 是按照本发明的用于生成模拟可执行模型和相关模拟配置数据库的模型建立过程的高级流程图；

图 9A 例示了例示配置编译器实现的跟踪过程在配置信号和相关配置锁存器之间的信号路径中检测反相器的方式的一部分数字设计;

图 9B 是按照本发明优选实施例的配置编译器实现的示范性跟踪过程的高级流程图;

图 10 是按照本发明优选实施例的配置编译器分析配置说明语句内的每个信号或 Dial 标识符的示范性方法的高级逻辑流程图;

图 11A 描绘了 Dial 组的示意性表示;

图 11B 例示了包括在多个分层排列 Dial 组中分组的 Dial 的示范性模拟模型;

图 12A 描绘了按照本发明的模拟配置数据库的示范性实施例;

图 12B 是按照本发明的包括代表 Dial 和 Register 的数据结构的示范性模拟配置数据库的更详细图形;

图 13 是按照本发明的在数据处理系统的易失性存储器内扩展配置数据库的例示性方法的高级逻辑流程图;

图 14 是描绘按照本发明在模拟模型的模拟运行期间易失性系统存储器的内容的方块图;

图 15 是在配置数据库中定位通过在 API 调用中供应的实例限定符和拨号器名限定符识别的一种或多种 Dial 实例数据结构(DIDS)的示范性方法的高级逻辑流程图;

图 16A 是按照本发明在数字设计的模拟期间以交互模式读取 Dial 实例的示范性方法的高级逻辑流程图;

图 16B 是按照本发明在数字设计的模拟期间以交互模式读取 Dial 组实例的示范性方法的高级逻辑流程图;

图 17A 是按照本发明在数字设计的模拟期间以交互模式设置 Dial 实例的示范性方法的高级逻辑流程图;

图 17B 是按照本发明在数字设计的模拟期间以交互模式设置 Dial 组实例的示范性方法的高级逻辑流程图;

图 18A 是按照本发明在数字设计的模拟期间以成批模式设置 Dial 实例或 Dial 组实例的示范性方法的高级逻辑流程图;

图 18B 是在如图 18A 所示的过程内调用的 end_phase API 的更详细流程图;

图 18C 是程序可以用于访问和修改数据库，以便指定默认值应用的阶段调整的数据处理系统环境的方块图；

图 19 是描绘按照本发明的示范性实验室测试系统的方块图；

图 20 是形成图 19 的实验室测试系统的一部分的数据处理系统内的集成电路芯片的更详细方块图；

图 21 是转换模拟配置数据库以获取适合用在配置数字设计的硬盘实现中的芯片硬件数据库的例示性过程的高级流程图；

图 22A 是按照本发明转换配置数据库以获取芯片硬件数据库的示范性过程的高级逻辑流程图；

图 22B 描绘了遵从如图 22A 所示的转换过程的芯片硬件数据库内的锁存器数据结构的例示性实施例；

图 23A 是将硬件配置数据库从非易失性存储器装入支持硬件配置数据库，以及任意大小或配置的数字系统的使用的易失性存储器中的示范性方法的高级逻辑流程图；

图 23B 例示了按照本发明一个实施例的数字系统的硬件配置数据库的示范性实施例；

图 24 是参照硬件配置数据库，识别数字系统中与 API 调用有关的一个或多个 Dial 实例或 Dial 组实例的示范性方法的高级逻辑流程图；

图 25 是为了商业推广可以压缩在系统固件的实验室开发和测试期间开发的硬件配置数据库的示范性过程的高级逻辑流程图；

图 26A-26C 一起形成按照本发明利用软件压缩工具压缩硬件配置数据库的例示性方法的高级逻辑流程图；

图 27 是按照本发明的包括 Dial 和只读 Dial 两者的示范性配置数据库的内容的图形表示；

图 28A-28B 分别例示了按照本发明的一个实施例使只读父字段包括在配置数据库的 Dial 实例数据结构和锁存器数据结构内，以便支持只读 Dial 和只读 Dial 组；

图 29 是将包含 RDial 和/或 RDial 组的配置数据库扩展到易失性存储器中的示范性方法的高级逻辑流程图；

图 30 是按照本发明的、分析硬件系统的所选状态，尤其，硬件系统的失败状态的示范性过程的高级流程图；

图 31 是按照本发明的、图 30 的芯片分析工具生成用于分析硬件失败的芯片配置报告和模拟设置文件的示范性方法的高级逻辑流程图;

图 32 描绘了按照本发明的、支持诸如 Dial、Dial 组、和 Register 之类的配置实体实例的选择性表示的配置数据库的示范性实施例;

图 33 是有选择地表示描述模拟或硬件系统的状态的配置实体实例的设置的示范性过程的高级逻辑流程图;

图 34A 例示了按照本发明的表示模拟或硬件系统的示范性图形用户界面(GUI);

图 34B 描绘了按照本发明的所显示设计分层结构深度受到限制的表示在图 34A 的示范性 GUI 内的系统的视图;

图 34C 例示了按照本发明的演示可以直观穿过设计分层结构的方式的表示在图 34A 的示范性 GUI 内的系统的视图;

图 34D 描绘了按照本发明的演示可以暴露配置层的其它层的方式的表示在图 34A 的示范性 GUI 内的系统的视图;

图 34E 例示了按照本发明的根据配置数据库设置从表示中有选择地删除配置实体实例的表示在图 34A 的示范性 GUI 内的系统的视图; 和

图 34F 描绘了按照本发明的以不同图形方式显示相关度发生改变的配置实体实例的表示在图 34A 的示范性 GUI 内的系统的视图。

具体实施方式

本发明应用用于配置和控制数字系统(例如,一个或多个集成电路或它(们)的模拟模型)的设置的配置说明语言和相关方法、系统和程序产品。在至少一个实施例中,负责相关设计实体的设计者用 HDL 代码创建对数字系统中的信号的配置说明。因此,最能指定信号名和相关合法值的处在设计过程前端的设计者负责创建配置说明。在模型建立时与描述数字系统的 HDL 一起编译配置说明,以获得卷入设计、模拟和配件实现过程中的下游组织团队以后可以使用的配置数据库。

现在参照附图,尤其参照图 1,描绘按照本发明的数据处理系统的示范性实施例。所描绘的实施例可以作为,例如,工作站、服务器、或大型计算机来实现。

如图所示,数据处理系统 6 包括一个或多个处理节点 8a-8n,如果实施

多于一个的处理节点 8，则通过节点互连线 22 互连处理节点 8a-8n。处理节点 8a-8n 的每一个都可以包括一个或多个处理器 10、本地互连线 16、和通过存储器控制器 17 访问的系统存储器 18。处理器 10a-10m 最好（但未必）相同，可以包括可从国际商用机器（IBM）公司（Armonk，New York）购得的处理器的 PowerPC™系列的处理器。除了寄存器、一般被叫作处理器核心 12 的用于执行程序指令的指令流逻辑和执行单元之外，处理器 10a-10m 的每一个也包括用于使数据从系统存储器 18 登台（stage）到相关处理器核心 12 的单片高速缓冲存储器层。

处理节点 8a-8n 的每一个进一步包括耦合在本地互连线 16 和节点互连线 22 之间的各自节点控制器 20。每个节点控制器 20 通过执行至少两种功能用作远程处理节点 8 的本地代理器。首先，每个节点控制器 20 窥探相关本地互连线 16 和促进本地通信事务到远程处理节点 8 的传送。其次，每个节点控制器 20 窥探节点互连线 22 上的通信事务和掌握相关本地互连线 16 上的相关通信事务。每条本地互连线 16 上通信受判优器 24 控制。判优器 24 根据处理器 10 生成的总线请求信号调节对本地互连线 16 的访问和编译对在本地互连线 16 上窥探的通信事务的相干响应。

本地互连线 16 通过中间总线桥接器 26 与中间总线 30 耦合。中间总线桥接器 26 提供处理器 10 可以直接映射到总线存储器和/或 I/O 地址空间的访问 I/O 设备 32 和存储设备 34 当中的设备的低等待时间路径和 I/O 设备 32 和存储设备 34 可以访问系统存储器 18 的高带宽路径两者。I/O 设备 32 可以包括，例如，显示设备、键盘、图形指示器、和与外部网络或附属设备连接的串行和并行端口。存储设备 34 可以包括，例如，为操作系统、中间件和应用软件提供非易失性存储的光盘或磁盘。在本实施例中，这样的应用软件包括可以用于按照本发明的方法和系统开发、核实和模拟数字电路设计的 ECAD 系统 35。

利用 ECAD 系统 35 创建的模拟数字电路设计模型包括至少一个，通常为多个的下文称为设计实体的子单元。现在参照图 2，图 2 例示了可以利用 ECAD 系统 35 创建的示范性设计实体 200 的方块图表示。设计实体 200 通过许多部件定义：实体名、实体端口、和由设计实体 200 执行的功能的表示。给定模型内的每个设计实体具有在设计实体的 HDL 描述中说明的唯一实体名（在图 2 中未明确示出）。并且，每个设计实体通常包含到设计实体之外的信号

的称为端口的许多信号互连。这些外部信号可以是整个设计的基本输入/输出（I/O）或与整个设计内的其它设计实体连接的信号。

通常，端口被分类成属于三种不同类型之一：输入端口、输出端口、和双向端口。设计实体 200 被描绘成具有将信号传送给设计实体 200 的许多输入端口 202。输入端口 202 与输入信号 204 连接。另外，设计实体 200 包括将信号传送到设计实体 200 以外的许多输出端口 206。输出端口 206 与一组输出信号 208 连接。双向端口 210 用于将信号传送给设计实体 200 和设计实体 200 以外。双向端口 210 依次与一组双向信号 212 连接。诸如设计实体 200 之类的设计实体不需要包含所有三种类型的端口，在退化情况下，一个端口也不包含。为了实现设计实体与外部信号的连接，利用称为“端口图”的映射技术。端口图（在图 2 中未明确示出）包括实体端口名和实体与之连接的外部信号之间的特定对应关系。当建立模拟模型时，ECAD 软件 35 用于根据端口图说明将外部信号与实体的适当端口连接。

如图 2 进一步所示，设计实体 200 包含描述设计实体 200 执行的一个或多个功能的主体部分 214。在数字设计的情况中，主体部分 214 包含逻辑门、存储元件等的互连，以及其它实体的示例。通过例示另一个实体内的实体，实现了整个设计的分层描述。例如，微处理器可以包含相同功能单元的多个实例。这样，微处理器本身往往被模型化成单个实体。在微处理器实体内，将出现任何重复功能实体的多个实例。

每个设计实体通过包含描述设计实体所需的信息的一个或多个 HDL 文件指定。尽管不是本发明所要求的，但为了易于理解，下文假设每个设计实体通过各自 HDL 文件指定。

现在参照图 3，图 3 例示了本发明的优选实施例中可以由 ECAD 系统 35 用于表示数字设计（例如，集成电路芯片或计算机系统）的示范性模拟模型 300 的示意性表示。为了看起来简单清楚，没有明确示出互连模拟模型 300 内的设计实体的端口和信号。

模拟模型 300 包括许多分层排列设计实体。与任何模拟模型内部一样，模拟模型 300 包括唯一一个包罗模拟模型 300 内的所有其它实体的“顶层实体”。也就是说，顶层实体 302 直接或间接地例示了数字设计中的所有后代实体。具体地说，顶层实体 302 直接例示了同一定点执行单元（FXU）304 的两个实体 304a 和 304b、和浮点单元（FPU）实体 314 的单个实例（即，是它们

的直接祖辈)。分别具有示例名 FXU0 和 FXU1 的 FXU 实体实例 304 又例示了包括分别具有示例名 A0 和 A1 的实体 A 306 的多个示例的另外设计实体。

设计实体的每个示例都具有包含实体名和示例名（若有的话）的相关描述，实体名和示例名在直接祖辈实体的所有后代当中必须是唯一的。例如，顶层实体 302 具有包括实体名 332（即，冒号前面的“TOP”）描述 320，它还包括示例名 324（即，冒号后面的“TOP”）。在实体描述内，当在祖辈实体内只例示那个特定实体的一个实例时，实体名与示例名一致是常见的。例如，在 FXU 实体示例 304a 和 304b 的每一个内例示的实体 B310 和实体 C312 的单个实例具有一致的实体名和示例名。但是，正如 FPU 实体 314 所示的那样(即，示例名是 FPU0，而实体名是 FPU)，这种命名惯例不是本发明所要求的。

数字设计中实体在其它实体内的嵌套可以延伸到任意复杂程度，只要无论单个还是多个示例的所有实体具有唯一实体名和任何直接祖辈实体内的所有后代实体的示例名相互之间是唯一的即可。

与每个设计实体示例相联系的是所谓的“示例标识符”。给定示例的示例标识符是包括从顶层实体示例名出发的封闭实体示例名的字符串。例如，FXU 实体 304 的示例 304a 内实体 C312 的示例 312a 的设计示例标识符是“TOP.FXU0.B.C”。这个示例标识符用于唯一地标识模拟模型内的每个示例。

正如上面所讨论的那样，无论利用物理集成电路还是作为诸如模拟模型 300 之类的软件模型实现的数字设计通常包括用于将数字设计配置成适合运算的配置锁存器。与应用实现了设计之后创建的独立配置软件将值装入配置锁存器中的现有技术设计方法不同，本发明引入了允许数字设计者为信号指定配置值作为设计过程的自然部分的配置说明语言。尤其，本发明的配置说明语言允许利用嵌在指定数字设计的一个或多个 HDL 文件（如图 4A 所示）中或嵌在由指定数字设计的一个或多个 HDL 文件引用的一个或多个外部配置文件（如图 4B 所示）中的语句指定设计配置。

现在参照图 4A，图 4A 例示了按照本发明的包括嵌入式配置语句的示范性 HDL 文件 400，在这种情况下，VHDL 文件。在本例中，HDL 文件 400 指定模拟模型 300 的实体 A 306，和包括三个部分：VHDL 代码，即，指定端口 202、206 和 210 的端口表 402；指定主体部分 214 内的信号的信号说明 404；和指定主体部分 214 的逻辑关系和功能的设计说明 406。混杂在三个部分内的是前面用双虚线“--”表示的传统 VHDL 注释。另外，嵌在设计说明 406 内的是

用标号 408 和 410 集体表示的按照本发明的一个或多个配置说明语句。如图所示，这些配置说明语句以用“--##”开头的特殊注释形式写入，以便使编译器可以容易地将配置说明语句与传统 HDL 代码和 HDL 注释区分开。配置说明语句最好应用对格和空白不敏感的语法。

现在参照图 4B，图 4B 例示了按照本发明的包括对外部配置文件的引用的示范性 HDL 文件，外部配置文件包含一个或多个配置说明语句。如撇号(')所指的那样，除了配置说明语句 408，410 被引用包含配置说明语句 408，410 的独立配置文件 414 的一个或多个（在这种情况下，只有一个）配置文件引用语句 412 所取代之外，HDL 文件 400'在其它所有方面都与 HDL 文件 400 相同。

与如图 4A 所示的嵌入式配置说明语句一样，配置文件引用语句 412 通过标识符“--##”标识成配置语句。配置文件引用语句 412 包括指令编译器定位独立配置文件 414 的指令“cfg_file”、和配置文件的文件名（即，“file00”）。诸如配置文件 412 之类的配置文件最好都应用所选文件扩展名（例如，“.cfg”），以便在数据处理系统 6 应用的文件系统内可以容易地定位、组织和管理它们。

正如下面参照图 8 进一步讨论的那样，无论嵌在 HDL 文件内还是集中在一个或多个配置文件 414 中的配置说明语句都与相关 HDL 文件一起由编译器处理。

按照本发明的优选实施例，诸如配置说明语句 408，410 之类的配置说明语句有助于通过例示这里统称为“Dial”的配置实体的一个或多个实例在数字设计中配置配置锁存器。Dial 的功能是在一个输入值和一个或多个输出值之间映射。一般说来，这样的输出值最终直接或间接地指定配置锁存器的配置值。每个 Dial 与数字设计中的特定设计实体相联系，按照惯例，该特定设计实体是由包含使 Dial 得到例示的配置说明语句或配置文件引用语句的 HDL 源文件指定的设计实体。因此，通过它们与都具有唯一示例标识符的特定设计实体的联系，只要在任何给定设计实体内应用唯一 Dial 名，数字设计中的 Dial 可以唯一地得到标识。显而易见，从 Latch Dial（或“LDial”）开始，可以定义许多不同类型的 Dial。

现在参照图 5A，图 5A 描绘了示范性 LDial 500 的表示。在这个特例中，具有名称“bus ratio”的 LDial 500 用于按照代表部件时钟频率和总线时钟

频率之间的所选比率的枚举输入值，在数字设计中为配置锁存器指定值。

如图所示，与所有 Dial 一样，LDial 500 逻辑地含有单个输入端 502、一个或多个输出端 504、和将每个输入值映射成每个输出端 504 的各自相关输出值的映射表 503。也就是说，映射表 503 在一个或多个唯一输入值的每一个和各自相关唯一输出值之间指定一一对应映射。由于 LDial 的功能是指定配置锁存器的合法值，LDial 500 的每个输出端 504 在逻辑上控制装入各自配置锁存器 505 中的值。为了防止相冲突的配置，每个配置锁存器 505 由能够设置配置锁存器 505 的任何类型的唯一一个 Dial 直接指定。

在输入端 502 上，LDial 500 接收包括“2:1”、“3:1”和“4:1”的一组合法值当中的枚举输入值（即，字符串）。正如下面参照图 7A 进一步讨论的那样，枚举输入值可以由软件（例如，软件模拟器或服务处理器固件）直接提供，或者，可以由另一个 Dial 的输出端提供。对于每个枚举输入值，LDial 500 的映射表 503 都指出每个配置锁存器 505 的所选二进制值（即，“0”或“1”）。

现在参照图 5B,图 5B 例示了逻辑上包括 Dial 的模拟模型的示意性表示。如撇号所指包括以与图 3 的模拟模型 300 相同的分层关系排列的相同设计实体的图 5B 的模拟模型 300′例示了 Dial 的两个特性，即，重复和范围。

重复是每当例示相关设计实体时，自动例示在设计实体的 HDL 文件中指定的或由设计实体的 HDL 文件引用的 Dial 的过程。重复有利地减少了设计者创建 Dial 的多个相同实例所需的数据输入量。例如，为了例示如图 5B 所示的 LDial 的 6 个实例，设计者只需利用如图 4A 和 4B 所示的两种技术之一编码两个 LDial 配置说明语句。也就是说，设计者将第一 LDial 配置说明语句（或指向相关配置文件的配置文件引用语句）编码成设计实体 A 306 的 HDL 文件，以便自动例示分别在实体 A 的示例 306a0、306a1、306b0 和 306b1 内的 LDial 506a0、506a1、506b0 和 506b1。设计者将第二 LDial 配置说明语句（或指向相关配置文件的配置文件引用语句）编码成设计实体 FXU 304 的 HDL 文件,以便自动例示分别在 FXU 实体的示例 304a 和 304b 内的 LDial 510a 和 510b。然后，随着编译器重复相关设计实体，自动创建 LDial 的多个实例。因此，与设计者必须人工地在配置软件中逐个枚举每个配置锁存器值的现有技术的方法相比，数字设计中 Dial 的重复可以显著减轻对设计者的输入负担。应该注意到,重复的特性未必要求 Dial 的所有实例都生成相同的输出值;

通过将不同输入提供给它们，同一 Dial 的不同实例可以被设置成生成不同输出。

Dial 的“范围”这里被定义成 Dial 在它的说明中可以引用的一组实体。按照惯例，Dial 的范围包括与 Dial 相联系的设计实体（即，通过包含使 Dial 得到例示的配置说明语句或配置文件引用语句的 HDL 源文件指定的设计实体）和包含在相关设计实体内的任何设计实体（即，相关设计实体和它的后代）。因此，Dial 不被限制成在设计分层结构中例示它的层上工作，而是也可以在它的范围内在设计分层结构的任何较低层上指定配置锁存器。例如，尽管分别与 FXU 实体示例 304a 和 304b 相联系，但 LDial 510a 和 510b 也可以分别在实体 C 的示例 312a 和 312b 内指定配置锁存器。

图 5B 例示了 LDial（和直接指定配置锁存器的其它 Dial）的另一个重要特性。具体地说，如图 5B 所示，在配置说明语句中允许习惯于在 HDL 文件中指定信号的设计者指定通过 Dial 设置的信号状态，而不是要装入确定信号状态的“上游”配置锁存器中的值。因此，在指定 LDial 506 的过程中，设计者可以为配置锁存器 512 设置的信号 514 指定可能信号状态。类似地，在指定 LDial 510 的过程中，设计者可以为配置锁存器 520 设置的信号 522 指定可能信号状态。正如下面进一步讨论的那样，指定信号状态而不是锁存器值的能力不仅与设计者思考数字设计的习惯方式相符，而且减少了在配置锁存器 512，520 和感兴趣信号 514，522 之间存在反相器引起的可能错误。

现在参照图 5C，图 5C 例示了包括 LDial 的模拟模型的另一个示意性表示。如撇号所指，图 5C 的模拟模型 300″包括以与图 3 的模拟模型 300 相同的分层关系排列的相同设计实体。

如图所示，图 5C 的模拟模型 300″包括与顶层设计实体 302 相联系的 LDial 524。LDial 524 指定由各自配置锁存器 512 确定的每个信号 sig1 514 的信号状态、由各自配置锁存器 520 确定的每个信号 sig2 522 的信号状态、由各自配置锁存器 530 确定的每个信号 sig4 532 的信号状态、和由各自配置锁存器 534 确定的每个信号 sig3 522 的信号状态。因此，LDial 524 配置了都在 LDial 524 的层（顶层）上或以下例示的许多不同信号的信号状态。

正如上面参照图 4A 和 4B 所讨论的那样，通过将指定 LDial 524 的配置说明语句或引用包含指定 LDial 524 的配置说明语句的独立配置文件的配置文件引用语句嵌入顶层实体 302 的 HDL 文件内，在模拟模型 300″的顶层实体

302 内例示 LDial 524。在每一种情况中，LDial 524 的示范性配置说明语句如下：

```
LDial bus ratio(FXU0.A0.SIG1,FXU0.A1.SIG1,
                FXU0.B.C.SIG2(0..5),
                FXU1.A0.SIG1,FXU1.A1.SIG1,
                FXU1.B.C.SIG2(0..5).
                FPU0.SIG3,SIG4(0..3)
                )=
               {2:1=>0b0,0b0,0x00,
                    0b0,0b0,0x00,
                    0b0,0x0;
                3:1=>0b1,0b1,0x01,
                    0b1,0b1,0x01,
                    0b0,0x1;
                4:1=>0b1,0b1,0x3F,
                    0b1,0b1,0x3F,
                    0b1,0xF
                };
```

上面给出的示范性配置说明语句从指定正被说明的 Dial 的类型是 LDial 的关键字“LDial”、和在这种情况下是“bus ratio”的 Dial 名开始。接着，配置说明语句枚举其状态受 LDial 控制的信号名。正如上面所指的那样，相对于相关设计实体的默认范围分层指定每个信号的信号标识符（例如，对于信号 514a0，FXU0.A0.SIG1），以便具有相同信号名的不同信号实体是可区分的。在信号标识符被枚举之后，配置说明语句包括列出 LDial 的允许枚举输入值和每个枚举输入值的相应信号值的映射表。信号值与说明信号名的顺序所暗示的信号名相联系。还应该注意到，为所有枚举值指定的信号状态是唯一的，集体表示信号状态的唯一合法模式。

几种不同语法可以应用于指定信号状态。在上面给出的例子中，以指定前面加上前缀“0b”的二进制常数的二进制格式，或以指定前面加上前缀“0x”

的十六进制常数的十六进制格式指定信号状态。尽管未示出，但也可以以整数格式指定信号状态，在整数格式的情况下，不加任何前缀。为了使数据易于输入，ECAD 系统 35 的配置说明语言最好也支持用前导零自动扩充的一个常数值用于代表所有所需信号值的并置的并置语法。在这种并置语法中，上面给出的配置说明语句的映射表可以重写成:

```
{2:1=>      0,
 3:1=>      0x183821,
 4:1=>      0x1FFFFF,
};
```

以便将枚举输入值 2:1 与所有 0 的并置位模式相联系，将枚举输入值 3:1 与‘0b110000011100000100001’的并置位模式相联系，和将枚举输入值 4:1 与所有 1 的并置位模式相联系。

现在参照图 5D，图 5D 例示了这里被定义成 Switch 的、存在 1-位输出的 LDial 的特殊情况的示意性表示。如图所示，Switch 540 含有单个输入端 502、控制配置锁存器 505 的设置的单个 1-位输出端 504、和将可以在输入端 502 上接收的每个枚举输入值映射成在输出端 504 上驱动的 1-位输出值。

由于 Switch 经常包括应用在数字设计中的绝大多数 Dial，最好在数字设计的模拟模型中所有 Switch 的枚举值集都相同（例如，“ON”/“OFF”）。在 Switch 的典型实施例中，通过映射表 503 将“肯定”的枚举输入值（例如，“ON”）映射成 0b1 的输出值，和将“否定”的枚举输入值（例如，“OFF”）映射成 0b0 的输出值。为了有助于相反极性的逻辑关系的使用，最好也支持使映射表 503 中输入值和输出值之间的这种默认对应关系反过来的 Negative Switch 或 NSwitch 说明。

定义 Switch 原语的主要优点是减少要求设计者输入的数据量。尤其，为了指定可比 1-位 LDial，要求设计者输入如下形式的配置说明语句:

```
LDial mode(signal)=
        {ON=>b1;
         OFF=>b0
```

```
};
```

另一方面，可以利用如下配置说明语句指定执行相同功能的 Switch:

```
Switch mode(signal);
```

尽管当只考虑单个 Switch 时，使用 Switch 除去的数据输入量不是特别显著，但当考虑复杂数字设计中的数以千计 Switch 时，数据输入的总减少量就显著了。

现在参照图 6A，图 6A 描绘了按照本发明优选实施例的 Integer Dial（“IDial”）的示意性表示。与 LDial 一样，IDial 通过在映射表 603 中指出在输入端 602 上接收的每个输入值和每个输出端 604 的输出值之间的对应关系，直接指定装入一个或多个配置锁存器 605 的每一个中的值。但是，与只能接收在它们的映射表 503 中明确阐述的枚举输入值作为合法输入值的 LDial 不同，IDial 的合法输入值集包括输出端 604 的位长内的所有可能整数值（右对齐和用 0 填充所有可用位扩充位数比输出端 604 的位长少的输入整数值）。由于枚举映射表 603 中的所有可能整数输入值既不方便又令人乏味，映射表 603 简单地指出在输入端 602 上接收的整数输入值施加在一个或多个输出端 604 上的方式。

IDial 合乎理想地适合于必须初始化一个或多个多位寄存器和合法值的个数包括寄存器的大多数值的应用。例如，如果包括 4 个配置锁存器的 4-位配置寄存器和包括 11 个配置锁存器的 11-位配置寄存器两者都利用 LDial 来配置，设计者在 LDial 的映射表中必须明确地枚举多达 2^{15} 个输入值和相应输出位模式。借助于利用如下配置说明语句的 IDial 就可以简单得多地管理这种情况:

```
IDial cnt_value(sig1(0..3),sig2(0..10));
```

在上面的配置说明语句中，“IDial” 将配置实体说明成 IDial，“cnt_value” 是 IDial 的名称，“sig1” 是 4-位配置寄存器输出的 4-位信号和 “sig2” 是与 11-位配置寄存器耦合的 11-位信号。另外，与 sig1 和 sig2 的每一个相联系的位的排序和个数指出整数输入值的 4 个高序位用于配置与 sig1 相联系的

4-位配置寄存器和 11 个低序位用于配置与 sig2 相联系的 11-位配置寄存器。重要的是，尽管映射表 603 指出整数输入值的哪些位被路由到哪些输出端，但在映射表 603 中没有明确指定输入值和输出值之间的对应关系。

正如在图 6B 中所描绘的那样，IDial 也可以用于为多个重复配置寄存器指定相同值。在所示的实施例中，可以被描述成 IDial“分路器”的 IDial 610 根据单个 15-位整数输入值指定每一组包括 15 个配置锁存器 605 的三组重复寄存器的配置。例示 IDial 610 的示范性配置说明语句可以像下面那样给出：

```
IDial cnt_value(A0.sig(0..7),A0.sig2(8..14);
                A1.sig(0..7),A1.sig2(8..14);
                A3.sig(0..7),A1.sig2(8..14)
                );
```

在上面的配置说明语句中，“IDial”将配置实体说明成 IDial，和“cnt_value”是 IDial 的名称。接在 IDial 名称之后的是用分号（“;”）分开的三个范围字段。每个范围字节指出整数输入值的位如何施加给特定信号。例如，第一范围字段指定整数输入值的 8 个高序位用于配置与信号 A0.sig1 相联系的 8-位配置寄存器和 7 个低序位用于配置与信号 A0.sig2 相联系的 7-位配置寄存器。第二和第三范围字段指定相似地配置设计实体 A1 和 A2 内的相应配置寄存器。重要的是，只要在每个范围字段中指定的总位数相同，可以在每个范围字段中不同地分配整数输入位。

尽管单独利用 LDial 或利用 LDial 和 IDial 完全可以指定数字设计的配置，在许多情况下，这样做是不够的和不方便的。尤其，对于诸如如图 5C 所示的分层数字设计之类的分层数字设计，LDial 和/或 IDial 的单独使用将许多 Dial 推向比负责包含 Dial 控制的配置锁存器的设计实体的设计分层结构的层更高的、从组织的观点来看可能是不同设计者或设计组负责的设计分层结构的层。结果，配置寄存器的适当配置不仅要求设计组之间的重要组织协调，而且要求负责更高数字设计层的设计者获知和在他们的 HDL 文件内包括与较低层设计实体的配置有关的细节。此外，在分层结构的较高层上实现 Dial 意味着，由于控制较低层设计实体的配置的 Dial 本身没有包含在较低层设计实体内，所以不能独立地模拟分层结构的较低层。

鉴于前面的情况，本发明认可提供支持Dial的分层组合的配置实体，以允许通过较低层 Dial 配置设计分层结构的较低层和通过一个或多个较高层 Dial 控制较低层 Dial 的使用。本发明的配置说明语言将控制一个或多个较低层Dial的较高层Dial称为Control Dial（“CDial”）。

现在参照图7A，图7A描绘了按照本发明的CDial 700a的示意性表示。与所有Dial一样，CDial 700a含有单个输入端702、一个或多个输出端704、和将每个输入值映射成每个输出端 704 的各自相关输出值的映射表 703。与直接指定配置锁存器的 LDial 和 IDial 不同，CDial 700 不直接指定配置锁存器。取而代之，CDial 700 以 n 路“Dial 树”的方式控制与 CDial 700 逻辑耦合的一个或多个其它 Dial（即，CDial 和/或 LDial 和/或 IDial），在 n 路“Dial 树”中，每个较低层 Dial 至少形成最终终止在配置锁存器的“叶”上的“分支”的一部分。Dial 树最好被构造成在任何 Dial 树中没有 Dial 被例示两次。

在在图 7A 中给出的示范性实施例中，CDial 700a 在输入端 702 上接收包括“A”,...，“N”的一组合法值当中的枚举输入值（即，字符串）。如果CDial 700a（或 LDial 或 IDial）是顶层 Dial（即，在 Dial 树中在它“上面”再也没有 Dial），CDial 700a 接收直接来自软件（例如，模拟软件或固件）的枚举输入值。可替代地，如果 CDial 700a 形成 Dial 树的“分支”部分，那么，CDial 700a 从另一个 CDial 的输出端接收枚举输入值。对于可以在输入端 702 上接收的每个合法枚举输入值，CDial 700a 都在映射表 703 中为每个相连 Dial（例如，Dial 700b，500 和 600）指定所选枚举值或位值。与每个输出端 704 相联系的映射表 703 中的值由 ECAD 系统 35 按照与输出端 704 耦合的较低层 Dial 的类型翻译。也就是说，为 LDial 和 CDial 指定的值被翻译成枚举值，而为 IDial 指定的值被翻译成整数值。借助于这些值，Dial 700b、500 和 600 的每一个最终直接或间接地指定一个或多个配置锁存器 705 的值。

现在参照图 7B，图 7B 例示了包含 Dial 树的模拟模型的另一个示意性表示，Dial 树包括控制多个较低层 LDial 的顶层 CDial。正如撇号所指的那样，图 7B 的模拟模型 300'''包括以与图 3 的模拟模型 300 相同的分层关系排列的相同设计实体，和包含与图 5C 的模拟模型 300''相同的配置锁存器和相关信号。

如图所示，图 7B 的模拟模型 300'''包括与顶层设计实体 302 相联系的顶

层 CDial 710。模拟模型 300'''进一步包括 4 个 LDial 712a、712b、714 和 716。与实体示例 A0 304a 相联系的 LDial 712a 控制由各自配置锁存器 512a 确定的每个信号 sig1 514a 的信号状态、和由配置锁存器 520a 确定的信号 sig2 522a 的信号状态。与实体示例 A1 304b 相联系的是 LDial 712a 的重复的 LDial 712b 类似地控制由各自配置锁存器 512b 确定的每个信号 sig1 514b 的信号状态、和由配置锁存器 520b 确定的信号 sig2 522b 的信号状态。与顶层实体 302 相联系的 LDial 714 控制由配置锁存器 530 确定的信号 sig4 532 的信号状态。最后，与实体示例 FPU0 314 相联系的 LDial 716 控制由配置锁存器 534 确定的信号 sig3 536 的信号状态。这 4 个 LDial 的每一个受与顶层实体 302 相联系的 CDial 710 控制。

正如上面参照图 4A 和 4B 所讨论的那样，通过将配置说明语句（或指向包含配置说明语句的配置文件的配置文件引用语句）嵌入相关设计实体的 HDL 文件中，在相关设计实体内例示 CDial 710 和在如图 7B 中描绘的 4 个 LDial 的每一个。下面给出用于例示如图 7B 所示的每个 Dial 的示范性配置说明语句：

```
CDial BusRatio(FXU0.BUSRATIO,FXU1.BUSRATIO,FPU0.BUSRATIO,
                    BUSRATIO)=
                {2:1=>2:1,2:1,2:1,2:1;
                3:1=>3:1,3:1,3:1,3:1;
                4:1=>4:1,4:1,4:1,4:1
                 };

LDial BusRatio(A0.sig1,A1.sig1,B.C.sig2(0..5))=
                {2:1=>0b0,0b0,0x00;
                3:1=>0b1,0b1,0x01;
                4:1=>0b1,0b1,0x3F
                 };

LDial BusRatio(sig3)=
                {2:1=>0b0;
```

```
                3:1=>0b1;
                4:1=>0b1
                 };

LDial BusRatio(sig4(0..3)=
                {2:1=>0x0;
                3:1=>0x1;
                4:1=>0xF
                 };
```

通过以这种方式实现分层 Dial 树，取得了几方面的优点。首先，由于 FXU 实体示例 304a 和 304b 内 LDial 712 的自动重复使指定 LDial 712 的代码只输入一次，减少了必须输入的软件代码量。其次，通过使每个设计者（设计团队）在他负责的设计实体内指定信号的配置，设计过程的组织边界得到尊重。最后，极大地简化了上层 Dial（即，CDial 710）的编码，降低了出错可能性。因此，例如，上面立即指定的 CDial 和 LDial 集合执行与上面参照图 5C 指定的“大” LDial 相同的功能，但在任何一个 Dial 中复杂性降低了许多。

许多 Dial，例如，在检测到不可纠正错误的情况下用于禁用特定设计实体的 Switch 具有 Dial 在几乎所有环境下应该具有的特定输入值。对于这样的 Dial，本发明的配置说明语言允许设计者在配置说明语句中明确地为 Dial 指定默认输入值。在示范性实施例中，通过在 Dial 的说明之后和结束分号之前引入“=默认值”指定默认值。例如，可以像下面那样给出 CDial 的默认值:

```
CDial BusRatio(FXU0.BUSRATIO,FXU1.BUSRATIO,FPU0.BUSRATIO,
                        BUSRATIO)=
                {2:1=>2:1,2:1,2:1,2:1;
                3:1=>3:1,3:1,3:1,3:1;
                4:1=>4:1,4:1,4:1,4:1
                 }=2:1;
```

应该注意到，对于 CDial 和 LDial，要求指定默认值是一般（即，除了 Switch 之外）列在映射表中的合法枚举值之一。对于 Switch，默认值必须是预定枚举值 “ON” 和 “OFF” 之一。

IDial 的默认值可以类似地像下面那样指定:

```
IDial cnt_value(A0.sig1(0..7),A0.sig2(8..14);
                A1.sig10..7),A1.sig2(8..14);
                A3.sig1(0..7),A1.sig2(8..14)
                )=0x7FFF;
```

在这种情况下，可以以十六进制、十进制或二进制格式给出的常数提供了由 IDial 控制的每个信号的默认输出值。为了使指定常数施加在所指信号上，如有需要，截去或用 0 填充高序位。

本发明的配置说明语言还允许控制施加特定默认值的时刻。例如，在模拟或硬件执行集成电路的引导序列的过程中，控制默认值的施加很重要。在引导序列的最初阶段，到集成电路的不同部分的时钟信号可能在不同时刻开始，这意味着按照指定 Dial 默认值，必须在不同时刻装载集成电路的不同部分中的锁存器。

按照本发明，通过将一个或多个阶段标识符（ID）与默认值相联系支持对默认值施加时序的控制。阶段 ID 是标记应该基本上同时施加默认值的 Dial 集合的字符串。可以将多个阶段 ID 与特定 Dial 相联系以提高灵活性。例如，在不同系统配置中，分集成电路的引导序列可能不同。于是，取决于系统配置，有必要或最好在不同阶段将默认值施加给特定 Dial。

在一种示范性语法中，可以像下面那样，在 Dial 说明语句中的默认说明之后，在用括号封闭的逗号隔开列表中可选地指定一个或多个阶段 ID（例如，phaseid0 和 phaseid1）:

```
CDial BusRatio(FXU0.BUSRATIO,FXU1.BUSRATIO,FPU0.BUSRATIO,
                    BUSRATIO)=
               {2:1=>2:1,2:1,2:1,2:1;
```

```
3:1=>3:1,3:1,3:1,3:1;
4:1=>4:1,4:1,4:1,4:1
 }=2:1(phaseid0,phaseid1);
```

最好，为没有指定默认值的 Dial 指定阶段 ID 是错误的，如上所述，正如前面给出的示范性 CDial 和 IDial 说明所指的那样，任何阶段 ID 的指定最好完全可选。

将默认值用于 Dial 受许多规则支配。首先，可以为包括 LDial、IDial（包括带有分输出端的那些）和 CDial 的任何类型 Dial 指定默认值。对于 Dial 组（下面参照图 11A-11B 讨论的），最好不支持默认值。其次，如果为多层 Dial 树中的多个 Dial 指定默认值，只施加影响 Dial 树的每条“分支”的最高层默认值（包括为顶层 Dial 指定的那个），并且，忽略其余默认值（若有的话）。尽管有这条规则，但正如在上面所讨论的那样，由于在独立模拟模型的较小部分的情况下，可以施加默认值，所以为 Dial 树中的较低层 Dial 指定默认值仍然是有益的。在为形成 Dial 树的“分支”的较低层 Dial 指定的默认值的组合不对应于为较高层 Dial 设置的合法输出值的情况下，编译器将标出错误。最后，当 Dial 接收到主动设置 Dial 的输入时，取代默认值。

通过为 Dial 指定默认值，设计者通过减少必须为模拟或硬件配置明确设置的 Dial 的个数，极大地简化了下游组织团队对 Dial 的使用。另外，正如下面进一步讨论的那样，默认值的使用有助于审查哪些 Dial 已被主动设置。

在本发明的至少一个实施例中，本发明的配置说明语言支持这里称为 Register 的另外结构的定义和使用。Register 将逻辑名与任意锁存器集合相联系，从而允许参照逻辑名设置和读取锁存器的值。关于这一点，Register 与如上所述的 Dial 类似。但是，与如上所述的 LDial 和 IDial 不同，Register 可以包括其它 Dial，譬如，LDial 和 IDial 引用的锁存器（以及任何 Dial 不引用的锁存器）。

现在参照图 7C，图 7C 例示了按照本发明的 Register 720 的示意性表示。Register 720 含有输入端 722 和一个或多个 1-位输出端 724（例如，输出端 724a-724h）。如图所示，Register 720 的输出端 724a-724h 每一个都与 1-位锁存器 705a-705h 的相应一个逻辑耦合。在这些锁存器 705 当中，锁存器 705a-705b 与将 CDial 700 作为顶层 Dial 的 Dial 树逻辑耦合。另外，锁存

器 705c-705d 与 LDial 500 逻辑耦合，和锁存器 705e-705f 与 IDial 600 逻辑耦合。锁存器 705g-705h 没有被任何 Dial 引用。

Register 720 和 Dial 500、600 和 700 的所示排列有利地允许利用锁存器 705a-705f 的最方便概念化设置和读取锁存器 705a-705f。往往出现这样的情况，诸如锁存器 705a-705f 之类的锁存器集合被方便地概念化成可以，例如，从不同来源导出它们的值，代表不同数据类型（例如，整数或枚举值），具有不同合法值范围等的许多不同子域。因此，通过应用多个 Dial 来描写一些或所有子域，可以在定义 HDL 模型的代码内不同地将子域的合法值和数据类型编入文档。

另一方面，卷入开发数字设计的某些部门可能发现概念化和访问作为在这种情况下是 Register 720 的单片实体的锁存器 705a-705h 更加方便。例如，由于寻址作为数个不同子域的锁存器 705a-705f，因此，通过 Dial 500，600，700 访问锁存器 705a-705f 的代码比寻址作为单片寄存器的锁存器 705a-705h 的代码多，固件开发者和对最小化代码图像的尺寸感兴趣的其它人员可能更喜欢访问作为 Register 720 的锁存器 705a-705h，把进行获取构成 Register 值的位序列所需的子域的组合交给使用人员去做。

尽管各种各样的语法可以用于说明 Register，但在 HDL 文件 400 或配置文件 414 内用于说明 Register 的 Register 说明语句的示范性语法可以像下面那样给出：

```
REGISTER my_reg(x.y.signal(0to4),signal2(0to6));
```

在这个示范性说明语句中，关键字“REGISTER”将语句标识成寄存器说明，“my_reg”是 Register 的名称，和“x.y.signal(0 到 4)”和“signal2(0 到 6)”是包括在 Register 中的一组锁存器的输出信号（当然，在 Register 内可以包括任意个数的信号）。

与如上所述的 Dial 一样，可以构造出各种各样的规则集来定义 Register 的允许使用和功能。在一个示范性实施例中，通过下面参照图 8 所述的模型建立过程为 Register 实施如下规则：

（1）Register 不支持默认值；

（2）Register 可以引用也被 Dial 引用的锁存器；

（3）Register 不能引用被另一个 Register 引用的锁存器（或信号）；

（4）Register 不能被 Dial（例如，CDial）引用，因此，是顶层实体；

（5）Register 可以引用没有被任何 Dial 引用的锁存器（在退化情况下，Register 引用的所有锁存器只能被那个 Register 引用）。

正如本领域的普通技术人员所认识到的那样，本发明的其它实施例可以应用对于寄存器组、Register 的分层排列可以支持默认值的不同规则集、和其它可替代或附加规则。

除了为指定 Dial 和 Register 的配置说明语句定义语法之外，本发明的配置说明语言还至少支持两种附加 HDL 语义结构：注释和属性说明语句。注释可以具有如下形式：

```
BusRatio.comment="The bus ratio Dial configures the circuit in accord ance with a selected processor/interconnect frequency ratio";
```

这个注释允许设计者将引号界定的任意字符串与特定 Dial 名相联系。正如下面参照图 8 所讨论的那样，这些注释在编译期间得到处理和包括在配置文档文件内，以便说明 Dial 的功能、关系、和适当设置。

属性说明语句是说明属性名和属性值和将属性名与特定实体名相联系的语句。例如，属性说明语句可以具有如下形式：

```
BusRatio.attribute(myattribute)=scom57(0:9);
```

在本例中，“BusRatio.attribute”说明这个语句是将属性与将“BusRatio”作为它的 Dial 名的 Dial 相联系的属性说明语句，“myattribute”是属性的名称，和“scom57(0:9)”是指定属性值的字符串。属性支持定制特征和对基本配置说明语言的语言扩充。

现在参照图 8，图 8 描绘了编译包含配置语句的 HDL 文件，为数字设计获取模拟可执行模型和模拟配置数据库的模型建立过程的高级流程图。该过程从包括配置说明语句和/或配置文件引用语句的一个或多个设计实体 HDL 源代码文件 800，和可选地，一个或多个配置文件引用文件 802 开始。HDL 编译器 804 处理 HDL 文件 800 和配置说明文件 802（若有的话），从模拟模型的

顶层实体开始和对描述完整模拟模型的所有 HDL 文件 800 以递归方式进行下去。随着 HDL 编译器 804 处理每个 HDL 文件 800，HDL 编译器 804 在存储器中形成的设计中间文件 806 中创建“标记”，以标识嵌在 HDL 代码和嵌入式配置文件引用语句引用的任何配置说明文件中的配置语句。

此后，配置编译器 808 和模型建立工具 810 处理存储器中的设计中间文件 806，以完成模型建立过程。模型建立工具 810 将设计中间文件 806 处理成模拟可执行模型 816，当被执行时，模拟可执行模型 816 模型化可以代表，例如，一部分集成电路、整个集成电路或模块、或包括多个集成电路或模块的数字系统的数字设计的逻辑功能。在这种处理中，模型建立工具 810 最好生成指示构成模拟模型的设计实体之间的分层关系的 m 路树。

配置编译器 808 处理在设计中间文件 806 中标记的配置说明语句和从那些语句中创建配置文档文件 812 和配置数据库 814。配置文档文件 812 以人可读的格式列出描述与模拟模型相联系的 Dial 的信息。该信息包括 Dial 的名称、它们的映射表、Dial 树的结构（若有的话）、实例信息等。另外，如上所述，配置文档文件 812 包括包含在在数字设计中描述 Dial 的功能和设置的注释语句中的字符串。这样，从负责创建 Dial 的设计者那里以“自底向上”的方式汇集适合供数字设计的模拟模型和配件实现两者使用的配置文档。然后，使配置文档适合于卷入设计、模拟、实验室硬件评估、和数字设计的商业硬件实现中的所有下游组织团队。

配置数据库 814 最好包含描述模拟可执行模型 816 内的设计实体的分层关系的通过模型建立工具 810 生成的 m 路树，以及与 Dial 和其它配置实体有关的许多数据结构。正如下面详细描述的那样，这些数据结构包括描述 Dial 实体的 Dial 数据结构、锁存器数据结构、和 Dial 实例数据结构。这些数据结构将特定 Dial 输入与用于配置数字设计（即，模拟可执行模型 816）的特定配置值相联系。在优选实施例中，可以针对信号状态或配置锁存器值指定配置值，使用那些值的选择是用户可选的。在数字设计的模拟期间，利用模拟可执行模型 816，通过应用程序编程接口（API）例程访问配置数据库 814，和配置数据库 814 进一步用于生成配置数字设计的物理实现的相似配置数据库。在优选实施例中，API 被设计成只能设置顶层 Dial（即，逻辑上没有 CDial 在它们“上面”的 LDial、IDial 或 CDial）和可以读取所有 Dial 值。

如上所述，本发明的配置说明语言有利地允许参照信号名（例如，

“sig1”）指定 LDial 和 IDial 的输出值。如上所述，提供这种特征的主要动机是设计者往往从对特定信号状态配置可操作信号的角度，而不是从配置相关配置锁存器的角度来考虑问题。但是，实际上，设计者希望对特定状态配置的信号可能不是直接连接到相关配置锁存器的输入端。相反，配置信号可能通过诸如缓冲器和反相器之类的一个或多个中间电路元件耦合到相关配置锁存器。与其人工跟踪到相关配置锁存器的每个可配置信号，然后为配置锁存器确定适当值来加重设计者负担，倒不如让配置编译器 808 自动跟踪到与信号耦合的第一存储元件（即，配置锁存器）的指定信号，和对设计者指定信号状态值进行任何必要反相，以获得要装入配置锁存器的适当值。

现在参照图 9A，图 9A 例示了包括控制数字设计中的数个信号 904a-904e 的状态的 LDial 900 的一部分数字设计。当配置编译器 808 进行信号 904a 的跟踪时，不要求反相设计者指定信号状态，因为信号 904a 直接连接到配置锁存器 902a。于是，配置编译器 808 将来自 LDial 900 的配置说明语句的设计者指定值存储到配置数据库 814 中，作为要装入配置锁存器 902a 中的值。到配置锁存器 902b 的信号 904b 的跟踪类似地不会导致来自 LDial 900 的配置说明语句的设计者指定值反相，因为信号 904b 和配置锁存器 902b 之间的唯一插入元件是非反相缓冲器 906。

通过使引用 HD 设计库中的锁存器原语的 HDL 语句包括在 HDL 文件 800 中，设计者经常例示诸如配置锁存器 902c 和 902d 之类的配置锁存器。响应这样的 HDL 库引用插入模拟可执行模型中的锁存器实体 903a，903b 可能包括在 HDL 代码中设计者不能明确“看见”的诸如反相器 908，910 之类的反相器。配置编译器 808 进行的自动跟踪仍然检测到这些反相器，从而防止可能的配置错误。

于是，当进行信号 904c 的跟踪时，由于在信号 904c 和配置锁存器 902c 之间存在反相器 908，在将配置锁存器的配置值存储在配置数据库 814 中之前，配置编译器 808 使为信号 904c 指定的设计者指定配置值反相。但是，当配置编译器 808 进行信号 904d 的跟踪时，尽管在信号路径中存在反相器 910，914 和缓冲器 912，但由于逻辑单元总体不反相，配置编译器 808 不使设计者指定信号状态值反相。应该注意到，配置编译器 808 可以精确地处理诸如反相器 910 之类的“隐藏”反相器和诸如反相器 914 之类的明确说明反相器。

图 9A 最后例示了通过中间 AND 门 916 耦合到多个配置锁存器 902e 和 902f

的信号 904e。在像这样跟踪过程在指定信号和最近配置锁存器之间检测到扇出逻辑单元的情况下，可以将配置编译器 808 配置成根据信号 904e 的设计者指定信号状态值为配置锁存器 902e，902f 生成适当配置值。但是，由于编译器为配置锁存器 902e，902f 选择的值可能以难以预料的方式影响从配置锁存器 902 接收配置值的其它电路，如果配置编译器 808 将 LDial 900 的配置说明语句标记成包含一个错误，那就更好了。

现在参照图 9B，图 9B 描绘了配置编译器 808 为在配置说明语句中指定的每个信号名实现的跟踪过程的高级逻辑流程图。如图所示，该过程从方块 920 开始，然后转到方块 922-924，方块 922-924 例示了配置编译器 808 将反相计数初始化成 0，然后，定位由在配置说明语句中指定的信号名标识的信号。

然后，该过程进入包括方块 926-936 的循环，方块 926-936 集体代表配置编译器 808 在信号路径中跟踪到第一锁存器元件的指定信号。具体地说，如方块 926-930 所示，配置编译器 808 确定信号路径中的下一个“上游”电路元件是锁存器（926），缓冲器（928）还是反相器（930）。如果电路元件是锁存器，该过程从循环中退出，转到下面描述的方块 940。但是，如果电路元件是缓冲器，该过程转到方块 934，方块 934 例示了配置编译器没有将反相计数加 1 地移动到下一个要处理的上游电路元件。如果电路元件是反相器，该过程转到方块 936 和 934，方块 936 和 934 描绘了将反相计数加 1，然后，移动到下一个要处理的上游电路元件。这样，配置编译器一边跟踪到配置锁存器的指定信号，一边确定由路径中的电路元件实现的信号状态反相的次数。如上所述，如果配置编译器 808 在信号路径中检测到除了缓冲器或反相器之外的电路元件，如方块 946 所示，配置编译器 808 最好标上错误。此后，在方块 950 上终止该过程。

在方块 926 上检测到配置锁存器之后，配置编译器 808 确定反相计数是奇数还是偶数。如方块 940-944 所示，如果反相计数是奇数，配置编译器在将值插入配置数据库 814 之前，在方块 942 上使信号的设计者指定配置值反相。如果反相计数是偶数，在将配置值插入配置数据库 814 之前，不进行反相。此后，在方块 950 上终止该过程。

正如已经描述的那样，本发明提供了允许数字系统的设计者利用嵌在描述数字系统的 HDL 设计文件中的配置语句为数字系统指定配置的配置说明语

言。配置语句在数字设计中逻辑地例示响应特定输入为数字设计提供配置值的一个或多个 Dial。与构成数字设计的设计实体一样，可以分层排列 Dial。配置说明语句与描述数字设计的 HDL 文件一起编译，生成可以被访问以便配置数字设计的模拟可执行模型或（在适当转换之后）物理实现的配置数据库。配置说明语句的编码最好支持响应耦合在信号和相关配置锁存器之间的奇数个反相器的检测，使信号的设计者指定配置值反相的跟踪过程。

再次参照图 5C，回忆一下 LDial 524 的示范性配置说明语句包括如下形式的带括号信号枚举:

```
LDial bus ratio(FXU0.A0.SIG1,FXU0.A1.SIG1,
                FXU0.B.C.SIG2(0..5),
                FXU1.A0.SIG1,FXU1.A1.SIG1,
                FXU1.B.C.SIG2(0..5).
                FPU0.SIG3,SIG4(0..3)
                )=
                ...
```

应该注意到，配置说明语句的信号枚举部分从与 Dial 相联系的设计实体（按照惯例，这是将例示 Dial 的配置说明语句或配置引用语句嵌入它的 HDL 文件中的设计实体）的范围开始，分别、分层和明确地枚举通过 Dial 配置的每个信号实例的信号标识符。这里将这种语法称为信号标识符的“完整表达式”。在 LDial 或 IDial 的配置说明语句的信号枚举部分或在 CDial 的配置说明语句的 Dial 枚举部分中应用“完整表达式”要求设计者知道和正确输入受 Dial 控制的信号（或较低层 Dial）的每个实例的分层标识符。因此，如果同一信号（或较低层 Dial）的新实例以后被加到数字设计中，设计者必须仔细地回顾引用同一信号（或 Dial）的其它实例的 Dial 的配置说明语句和将信号（或 Dial）枚举部分更新成包括新加入实例的完整表达式。

为了在将新信号和 Dial 实体加入数字设计时，减少输入配置说明语句的信号（或 Dial）枚举部分所需的输入量和减轻代码维护的负担，按照本发明的 ECAD 系统 35 还支持配置说明语句的信号（或 Dial）枚举部分的“简明表达式”语法。这里，当应用于 LDial 和 IDial 的配置说明语句时，将这种语

法更具体地称为“简明信号表达式”，和当应用于CDial的配置说明语句时，将这种语法更具体地称为“简明Dial表达式”。

在信号或Dial枚举的简明表达式中，可以用单个标识符枚举希望共同配置的所选范围内的实体的所有实例。例如，在图5C中，如果设计者想要对信号 sig1 514 的所有 4 个示例共同配置，设计者可以用单个简明信号表达式“[A].sig1”枚举LDial 524的配置说明语句中的所有4个示例，其中，括号项是出现感兴趣信号的实例的名称。在简明表达式中，表达式的默认范围暗指与Dial相联系的设计实体（在这种情况下，顶层实体302）的范围。因此，标识符“[A].sig1”指定顶层实体 302 的默认范围内，A 实体示例 304 内信号 sig1 514 的所有 4 个示例。

通过明确地枚举设计分层结构的所选层，可以进一步缩小简明表达式中标识符的范围。例如，简明表达式“FXU1.[A].sig1”仅指FUX1实体示例304b内的信号sig1示例514b0和514b1，但不包括FUX0实体示例304a内的信号sig1示例514a0和514a1。

当然，当在设计分层结构的较高层上只例示信号或Dial的单个实例时，简明表达式和完整表达式几乎要求相同的输入量（例如，“FPU0.sig3”与“[FPU].sig3”标识信号 sig3 536）。但是，应该注意到，如果以后将另一个FPU实体314加入模拟模型300″中，标识符的简明表达式将有利地应用于顶层实体302的范围内任何以后加入的FPU实体。

利用简明表达式，现在可以更简明地将LDial 524的配置说明语句重写如下:

```
LDial bus ratio([A].SIG1,[C].SIG2(0..5),
                FPU0.SIG3,SIG4(0..3)
                )=
                {2: 1=>0b0,0x00,0b0,0x0;
                 3: 1=>0b1,0x01,0b0,0x1;
                 4: 1=>0b1,0x3F,0b1,0xF
                };
```

如果将如上所述的并置语法应用于映射表，可以进一步将映射表缩减成:

```
{2: 1=>0;
 3: 1=>0x821;
 4: 1=>0xFFF
};
```

在并置语法中，与实际实体实例的个数无关，对于每个实体标识符，用单个相应位字段在映射表中指定信号值。例如，“[A].sig1”包括的所有实例用指定配置值的1个位表示，“[C].sig2”包括的所有实例用指定配置值的6个位表示，用“FPU0.sig3”标识的单个实例用指定配置值的 1 个位表示，和“sig4(0..3)”的单个实例用指定配置值的 4 个位表示。因此，利用并置语法，用 LDial 524 集体指定的 21 个位可以用等效的 12-位模式指定。

编码器以与简明信号表达式相同方式构造和分析简明 Dial 表达式。例如，利用简明 Dial 表达式可以将图 7B 的 CDial 710 的配置说明语句重写如下：

```
CDial BusRatio([FXU].BUSRATIO, [FPU].BUSRATIO, BUSRATIO)=
                {2: 1=>2: 1, 2: 1, 2: 1;
                 3: 1=>3: 1, 3: 1, 3: 1;
                 4: 1=>4: 1, 4: 1, 4: 1
                 };
```

此外，这种配置说明语句有利地允许 CDial 710 不作任何代码修改地通过附加 FXU 实体 304 或 FPU 实体 314 的示例自动控制以后加入模拟模型 300'''中的名为“Bus ratio”的任何附加 LDial。

现在参照图 10，图 10 描绘了按照本发明优选实施例的配置编译器 808 分析配置说明语句内的每个信号或 Dial 标识符的示范性方法的高级逻辑流程图。如上所述，每个信号或 Dial 标识符由用圆点（“.”）分开的一个或多个字段分层构成。最后字段指定信号（例如，“sig1”）或 Dial（例如，“Bus_Ratio”）的实例名，和前面的字段从默认范围开始压缩范围，按照惯例，默认范围是与 Dial 相联系的设计实体的范围。

如图所示，该过程从方块 1000 开始，然后，转到方块 1002，方块 1002 例示了配置编译器 808 确定信号或 Dial 标识符的第一或当前字段是否包含封装在括号中的实体标识符（例如，“[A]”），即，标识符是否是简明表达式。如果是的话，该过程转到下面将说明的方块 1020。如果不是的话，配置编译器 808 在方块 1004 上通过确定标识符的第一或当前字段是否是标识符的最后字段，确定标识符是否是完整表达式。如果是的话，信号或 Dial 表达式是完整表达式，该过程转到方块 1010。另一方面，如果标识符的当前字段不是最后字段，如方块 1006 所示，配置编译器 808 将当前范围压缩成在标识符的当前字段中标识的设计实体示例。例如，如果配置编译器 808 正在处理图 7B 的 CDial 710 的配置说明语句内的标识符 “FPU0.SIG3”，配置编译器 808 将范围从顶层实体 302 的默认范围压缩到 FPU 实体示例 314。如果如方块 1008 所示，存在标识符的当前字段所指的实体示例，如方块 1009 所示，在将当前字段更新成下一个字段之后，该过程返回到方块 1002。但是，如果在当前范围内不存在标识符的当前字段所指的实体示例，配置编译器 808 在方块 1032 上标出错误，并且终止信号或 Dial 标识符的处理。

再次参照方块 1004，当配置编译器 808 检测到已经到达完整表达式的最后字段时，如图 10 所示的过程从方块 1004 转到方块 1010。方块 1010 例示了配置编译器 808 试图在当前范围内定位名称与在信号或 Dial 标识符的最后字段中指定的名称匹配的单个信号或 Dial 实例。如果配置编译器 808 在方块 1012 上确定在当前范围内没有找到匹配实例，该过程转到方块 1032，并且，配置编译器 808 标出错误。但是，如果配置编译器 808 定位了匹配信号或 Dial 实例，那么，如方块 1014 所示，配置编译器 808 使配置数据库 814 中的实体将信号或 Dial 实例与在正被处理的 Dial 的配置说明语句的映射表中指定的参数结合在一起。此后，在方块 1030 上终止信号或 Dial 标识符的处理。

现在参照方块 1020 和随后的方块，描述应用简明表达式的信号或 Dial 标识符的处理。方块 1020 描绘了配置编译器 808 试图在带括号字段所指的实体的当前范围内一个或多个实例的每一个中定位与在信号或 Dial 标识符指定的那个匹配的每个信号或 Dial 实例。例如，当处理图 7B 的模拟模型 300''' 的简明表达式 “FXU1.[A].sig1” 时，一旦到达字段 “[A]”，就在 FXU1 中搜索实体 A 306 的示例，和一旦找到实体示例 306a0 和 306a1，就在这两个实体示例的每一个内搜索，以定位信号示例 sig1 514a0 和 514a1。如果配置编

译器 808 在方块 1022 上确定在当前范围内没有找到匹配信号或 Dial 实例，该过程转到方块 1032，方块 1032 描绘了配置编译器 808 在标出错误之后，终止信号或 Dial 标识符的处理。但是，如果配置编译器 808 定位了一个或多个匹配信号或 Dial 实例，那么，该过程从方块 1022 转到方块 1024。方块 1024 例示了配置编译器 808 使配置数据库 814 中的一个或多个实例将每个匹配信号或 Dial 实例与在正被处理的 Dial 的配置说明语句的映射表中指定的参数结合在一起。此后，在方块 1030 上终止信号或 Dial 标识符的处理。

利用本发明支持的简明表达式，可以有利地减少设计者必须输入配置说明语句中的代码量。简明表达式的使用不仅降低了输入要求和输入错误的可能性，而且通过使指定配置参数自动应用于落在所选范围内的信号和 Dial 的以后输入实例，简化了代码维护。

如上所述，每个 Dial 在它的输入值的每一个和 Dail 的唯一输出值之间存在一一对应映射关系。换句话说，每个输入值具有与任何其它输入值的输出值不同的唯一输出值。对于 CDial 和 LDial，映射表必须明确地枚举每个合法输入值和它的相关映射关系。

在映射表中必须明确枚举输入值的要求限制了任何给定 LDial 或 CDial 的整体复杂性。例如，考虑包含每一个具有 5 到 20 之间合法值的 10 到 20 个配置寄存器的集成电路（例如，存储器控制器）的情况。在许多情况下，这些寄存器相互依赖－装在一个寄存器中的值可以影响一个或多个其它寄存器的合法可能性。理论上，利用单个 CDial 控制的 Dial 树为所有寄存器指定值将是方便的。这样，可以将所有 10 到 20 个寄存器的配置作为一个组来控制。

不幸的是，在如上所述的假设下，10 到 20 个寄存器总共可能存在超过 300,000 种的合法值组合。尽管理论上是可能的，但在这样情况下的 CDial 指定是不合乎需要的，实际上也是不可行的。此外，即使循环结构可以用于自动构建 CDial 的配置说明语句，配置说明语句尽管可以告诉输入值合法的模拟软件，但也不会告诉用户如何设置这个大小的 CDial。

认识到上面的情况之后，本发明的配置说明语言提供了“Dial 组”结构。Dial 组是设计者希望建立联系的一群 Dial。用于提供 Dial 输入值的运行期 API 通过防止 Dial 组内的各自 Dial 被分别设置，遵守这种联系。换句话说，必须同时设置 Dial 组中的所有 Dial，以便无需关心 Dial 之间的交互地不独立设置各自 Dial。由于软件强化了形成 Dial 组的 Dial 的分组守则，Dial 组

的使用还提供了设计者可以警告“下游”用户团体在构成 Dial 组的 Dial 之间存在一组未阐明相互依赖关系的机制。

现在参照图 11A，图 11A 例示了 Dial 组 1100a 的示意性表示。Dial 组 1100a 通过组名 1102（例如，“GroupG”）和列出一个或多个 Dial 或其它 Dial 组的 Dial 表 1104 来定义。Dial 组不存在任何输入端或输出端。列在 Dial 表 1104 内都是顶层 Dial 1110a-1110f 的 Dial 可以是 LDial、CDial 和/或 IDial。

图 11A 例示了 Dial 组 1100a 可以作为引用它的 Dial 表 1104 中的一个或多个其它 Dial 组 1100b-1100n 的分层 Dial 组来实现。这些较低层 Dial 组又引用它们各自 Dial 表中的一个或多个顶层 Dial 1110g-1110k 和 1110m-1110r（或其它 Dial 组）。

分层实现 Dial 组的一个动机是协调横跨组织边界的 Dial 的组的配置。例如，考虑 30 个 Dial 逻辑地属于一个 Dial 组和其中 10 个 Dial 包含在第一设计者负责的第一设计实体内和其中 20 个 Dial 包含在第二设计者负责的第二设计实体内的数字系统。如果没有分层 Dial 组，必须在包括第一和第二设计实体的设计分层结构的较高层上指定在它的 Dial 表 1104 中明确列出所有 30 个 Dial 的单个 Dial 组。这种实现的不便之处在于，负责较高层设计实体的设计者（或设计团队）必须知道较低层设计实体中的所有相关 Dial 和具体识别 Dial 组的 Dail 表 1104 中的 30 个 Dial 的每一个。

可替代分层方法需要创建包含第一设计实体内的 10 个 Dial 的第一 Dial 组、包含第二设计实体内的 20 个 Dial 的第二 Dial 组、和引用第一和第二 Dial 组的第三较高层 Dial 组。重要的是，较高层 Dial 组的 Dial 表 1104 必须只引用两个较低层 Dial 组，因此，使负责设计分层结构的较高层的设计者无需了解较低层细节。另外，由于改变属于两个较低层 Dial 组的哪些 Dial 不影响最高层 Dial 组的 Dial 表 1104，减轻了代码维护负担。

Dial 组受许多规则支配。首先，在不止一个 Dial 组的 Dial 表 1104 中可以不列出 Dial 或 Dial 组。其次，Dial 组必须引用它的 Dial 表 1104 中的至少一个 Dial 或其它 Dial 组。第三，在它的 Dial 表 1104 中，Dial 组只能引用它的范围内的 Dial 或 Dial 组，按照惯例（和像应用于 Dial 的范围的概念那样），它的范围是它的相关设计实体（即，设计实体本身和设计实体内的任何较低层设计实体）的范围。最后，在 Dial 组的 Dial 表 1104 中引用的每

个 Dial 必须是顶层 Dial。

现在参照图 11B，图 11B 描绘了例示 Dial 组的使用的示范性模拟模型 1120。示范性模拟模型 1120 包括具有示例标识符 “TOP:TOP” 的顶层设计实体 1122。在顶层设计实体 1122 内，例示了分别具有实体名 FBC 和 L2 的两个设计实体 1124 和 1126。FBC 实体示例 1124 又例示了具有 Dial 名 “C” 的 Dial 实例 1130、包含具有 Dial 名 “B” 的 Dial 实例 1134 的 Z 实体示例 1132、和分别名为 “X0” 和 “X1” 的实体 X 1136 的两个示例。每个实体 X 示例 1136 包含两个实体 Y 示例 1138，每个实体 Y 示例 1138 进一步例示具有 Dial 名 “A” 的 Dial 实例 1140。L2 实体示例 1126 包含具有 Dial 名 “D” 的 Dial 实例 1150 和两个实体 L 示例 1152，每个实体 L 示例 1152 包含具有 Dial 名 “E” 的 Dial 实例 1154。

如图所示，FBC 实体示例 1124 含有具有组名 “F” 的相关 Dial 组实例 1160。正如箭头所指的那样，Dial 组实例 1160 包括 FBC 实体示例 1124 内的 Dial 实例 1130、1134 和 1140 的每一个。L2 实体示例 1126 类似地含有包括 L2 实体示例 1126 内的 Dial 实例 1150 和 1154 的每一个的相关 Dial 组实例 1162。这两个 Dial 组实例又属于与顶层设计实体 1122 相联系的、具有组名 “H” 的较高层 Dial 组实例。

每个 Dial 组实例是通过使适当的配置语句包括在相关设计实体的 HDL 文件内创建的。例如，创建 Dial 组 “F”、“G” 和 “H” 的配置语句的示范性语法像下面那样分别给出：

```
GDial F(C,[Z].B,[Y].A);
GDial G(D,[L].E);
GDial H(FBC.F,L2.G);
```

在每个配置语句中，通过后面接着代表组名的字符串（例如，“F”）的关键字 “GDial” 说明 Dial 组。在组名后面的括号内，指定了 Dial 组的 Dial 表。正如在 Dial 组 “H” 的配置语句中所指的那样，分层 Dial 组的 Dial 表以与 Dial 相同的方式指定其它 Dial 组。还应该注意到，正如在 Dial 组 “F” 和 “G” 的配置语句中所指的那样，上面讨论的简明 Dial 表达式语法可以用在指定 Dial 表中的 Dial 或 Dial 组中。

既然已经描述了 Dial 的基本类型、用于它们指定的语法、和 Dial 组的应用。下面提供配置数据库 814 的示范性实现和它的使用的描述。为了更好地理解可以在配置数据库 814 中访问特定 Dial 示例（或 Dial 的多个示例）的方式，首先描述配置数据库 814 内 Dial 的命名法。

应用在本发明的优选实施例中的命名法首先要求设计者唯一地命名在任何给定设计实体内指定的每个 Dial，即，设计者不能用相同的 Dial 名说明同一设计实体内的任何两个 Dial。遵守这个要求可以防止在同一设计实体中例示的 Dial 之间的名称冲突，和促进在任意大小的模型中设计实体的任意重新使用。这种约束不会太麻烦，因为给定设计实体通常由特定设计者在特定时刻创建的，和在这样的有限环境内维护唯一 Dial 名只带来适中负担。

由于人们希望能够分别访问在给定模拟模型中可能含有多个示例（例如，由于重复）的 Dial 实体的特定示例，单独使用 Dial 名不能保证唯一地标识模拟模型中的特定 Dial 实体示例。于是，在优选实施例中，Dial 的命名法辅助支持了原始 HDL 所要求的相关设计实体的唯一示例标识符，以便对于模拟模型内的每个 Dial，用“扩展 Dial 标识符”分清同一 Dial 实体的多个实例。

另一方面，应该认识到，一些 HDL 不严格执行对唯一实体名的要求。例如，传统 VHDL 实体命名结构允许两个设计实体共享同一实体名 entity_name。但是，VHDL 要求这样的相同命名实体必须封装在从中可以构造有效 VHDL 模型的不同 VHDL 库内。在这样的环境下，entity_name 等效于用圆点（“.”）与在实体说明中说明的实体名并置的 VHDL 库名。因此，将不同 VHDL 库名预先附在实体名上可以分清共享同实体名的实体。大多数 HDL 都包括像这种唯一命名每个设计实体的机制那样的机制。

在优选实施例中，唯一标识 Dial 实体的特定示例的扩展 Dial 标识符包括三个字段：示例标识符字段、设计实体名、和 Dial 名。扩展 Dial 标识符可以表达成像下面那样用圆点（“.”）分开相邻字段的字符串：

<示例标识符>.<设计实体名>.<Dial 名>

在扩展 Dial 标识符中，设计实体名包含例示 Dial 的设计实体的实体名，和 Dial 名字段包含在 Dial 配置说明语句中为 Dial 说明的名称。如上所述，

在示例标识符字段中指定的示例标识符是一系列示例标识符，从模拟模型的顶层实体出发一直到给定 Dial 实例的直接祖辈设计实体，相邻实例标识符用圆点（“.”）分开。由于没有设计实体可以包括两个相同名称的 Dial，示例标识符对于模型内 Dial 的每一个实例来说都是唯一的。

设计实体名称字段中名称的唯一性是 Dial 之间的基本区分因素。通过将设计实体名包括在扩展 Dial 标识符中，实际上给予每个设计实体以与那个设计实体相联系 Dial 的唯一名称空间，即，给定设计实体内的 Dial 不能与其它设计实体相联系的 Dial 发生名称冲突。还应该注意到，仅仅利用示例标识符字段可以唯一地命名每个 Dial。也就是说，由于示例标识符的唯一性，只由示例标识符字段和 Dial 名字段形成的 Dial 标识符必然是唯一的。但是，这样的命名方案没有将 Dial 与给定设计实体相联系。实际上，人们希望通过将设计实体字段包括进来将 Dial 与它们所在的设计实体相联系，因为以后可以集中引用所有 Dial 示例，而无需查明包含 Dial 的所有设计实体示例的名称。

如上所述，扩展 Dial 标识符的使用使 Dial 的特定示例得到唯一标识和没有任何 Dial 名冲突风险地使设计实体在任意模型内得到重新使用。例如，再次参照图 11B，通过如下扩展 Dial 标识符可以唯一地分别标识 Dial A 实体示例 1140a0、1140a1、1140b0 和 1140b1:

FBC.X0.Y0.Y.A

FBC.X0.Y1.Y.A

FBC.X1.Y0.Y.A

FBC.X1.Y1.Y.A

借助于对 Dial 的优先命名法的了解，现在介绍图 12A，图 12A 是配置编译器 808 创建的配置数据库 814 的示范性格式的示意性表示。在这个示范性实施例中，配置数据库 814 包括至少 4 种不同类型的数据结构：Dial 定义数据结构（DDDS）1200、Dial 实例数据结构（DIDS）1202、锁存器数据结构 1204 和顶层指针阵列 1206。可选地，配置数据库 814 可以包括如虚线所示的附加数据结构，譬如，Dial 指针阵列 1208、锁存器指针阵列 1210、实例指针阵列 1226 和其它数据结构，正如下面进一步讨论的那样，可替代地，当装入配

置数据库 814 中时，可以在易失性存储器中构造它们。只有在配置数据库 814 被装入易失性存储器之后才生成这些附加数据结构有助于建立更加紧凑的配置数据库 814。

在数字系统中，在配置数据库 814 内为每个 Dial 或 Dial 组创建各自 Dial 定义数据结构（DDDS）1200。最好，在数字系统中，与 Dial（或 Dial 组）的示例个数无关地在配置数据库 814 中创建唯一一个 DDDS 1200。正如下面所讨论的那样，在分立 DIDS 1202 中指定与在 DDDS 1200 中描述的 Dial 的特定示例有关的信息。

如图所示，每个 DDDS 1200 包括表示 DDDS 1200 描述 Dial 还是 Dial 组的类型字段 1220，和如果 DDDS 1200 描述 Dial，那么，类型字段 1220 是 Dial 的类型。在一个实施例中，为类型字段 1220 设置的值包括 Dial 组的“G”、整数 Dial（IDial）的“I”、锁存器 Dial（LDial）的“L”、和控制 Dial（CDial）的“C”。DDDS 1200 进一步包括指定 DDDS 1200 描述的 Dial 或 Dial 组的名称的名称字段 1222。这个字段最好包含 Dial（或 Dial 组）的设计实体名，后面接着圆点（“.”），再接着在 Dial（或 Dial 组）的配置说明语句中给出的 Dial（或 Dial 组）的名称。名称字段 1222 的内容对应于 Dial 的扩展 Dial 标识符的设计实体名和 Dial 名字段。

如有需要，DDDS 1200 还包括包含从给定 Dial 的输入到它的输出的映射关系的映射表。对于 LDial 和 CDial，映射表 1224 与用于这些 Dial 的配置说明语句非常相似地指定输入值和输出值之间的关系。对于 Dial 组和没有分输出端的 IDial，映射表是空数据结构，不使用它。在 IDial 带有分输入端的情况下，映射表 1220 指定重复整数字段的宽度和那个字段的复制品的个数。这个信息用于将整数输入值映射到整数输出字段的各种各样复制品上。

最后，DDDS 1200 可以包括实例指针阵列 1226，实例指针阵列 1226 包含指向 DDDS 1200 定义的 Dial 或 Dial 组的每个实例的一个或多个实例指针 1228a-1228n。实例指针阵列 1226 有助于访问特定 Dial 或 Dial 组的多个实例。

如图 12A 进一步所示，配置数据库 814 包含与数字设计中的每个 Dial 示例或 Dial 组示例相对应的 DIDS 1202。每个 DIDS 1202 包含定义字段 1230，定义字段 1230 包含指向 DIDS 1020 为其描述特定实例的 Dial 的 DDDS 1200 的定义指针 1231。一旦识别出特定 Dial 实例，定义指针 1231 就允许容易地

访问实例的Dial名、Dial类型和映射表。

DIDS 1202进一步包括父字段1232，在IDial、CDial或LDial的情况中，父字段1232包含指向存在与相应Dial实例的输入端逻辑连接的输出端的较高层Dial实例（若有的话）的DIDS 1202的父指针1233。在Dial组的情况中，父指针1233指向分层包括当前Dial组的较高层Dial组（若有的话）的DIDS 1202。如果与DIDS 1202对应的Dial实例是顶层Dial和不属于任何Dial组，父字段1232中的父指针1233是NULL（空）指针。应该注意到，Dial可以是顶层Dial，但仍然属于Dial组。在那种情况下，父指针1233不是NULL，而是指向包含顶层Dial的Dial组的DIDS 1202。

因此，配置数据库814中DIDS 1202的父字段1232集体描述在数字设计中例示了的Dial实体和Dial组的分层排列。如下所述，父字段1232提供的分层信息有利地在给定最终受任何顶层Dial控制的配置锁存器的配置值的情况下，使那个顶层Dial的值得到确定。

DIDS 1202的实例名字段1234给出来自数字设计的顶层设计实体的由DIDS 1202描述的Dial实例的完全限定实例名。对于与顶层实体相联系的Dial实例，实例名字段1234最好包含NULL字符串。

DIDS 1202可以进一步包括默认值字段1229、阶段ID字段1227、和实例设置字段1239。在编译时，配置编译器808最好首先将默认值字段1229插入相关Dial的配置说明语句含有指定的默认值的至少每个DIDS 1202中。默认值字段1229存储指定的默认值；如果没有指定默认值，默认值字段1229是NULL，或省略默认值字段1229。配置编译器808随后利用递归穿越分析配置数据库814，和除去含有具有默认值的祖辈Dial实例的任何Dial实例的默认值字段1229（或将它设置成NULL）。这样，在分层结构中较高的Dial实例的默认值超过为较低层Dial实例指定的默认值。对于每个其余（或非NULL）默认值字段1229，配置编译器808将阶段ID字段1227插入DIDS 1202中，以便存储与默认值相联系的一个或多个阶段（若有的话）。存储在阶段ID字段1227内的阶段ID可以在HDL文件800或配置说明文件802内的Dial定义语句中指定，或者，可替代地，正如下面参照图18C所讨论的那样，可以由下游用户直接操纵配置数据库814来供应。

正如虚线记号所指的那样，当将配置数据库814装入易失性存储器中时，最好将实例设置字段1239插入配置数据库814中的每个DIDS 1302内。实例

设置字段 1239 是被初始化成 FALSE（假）和当相关 Dial 实例得到明确设置时被更新成 TRUE（真）的布尔（Boolean）值字段。

最后，DIDS 1202 包括输出指针阵列 1236，输出指针阵列 1236 包含指向描述与相应 Dial 实例或 Dial 组实例相联系的较低层示例的数据结构的指针 1238a-1238n。具体地说，在 IDial 和 LDial 的情况中，输出指针 1238 引用与 Dial 实例耦合的配置锁存器相对应的锁存器数据结构 1204。对于非分裂 IDial，输出指针 1238a 引用的配置锁存器实体接收整数输入值的高序位，和输出指针 1238n 引用的配置锁存器实体接收整数输入值的低序位。在 CDial 的情况中，输出指针 1238 引用与受 CDial 控制的 Dial 实例相对应的其它 DIDS 1202。对于 Dial 组，输出指针 1238 引用分层包括在与 DIDS 1202 相对应的 Dial 组实例内的顶层 Dial 实例或 Dial 组实例。

配置数据库 814 进一步包括模拟可执行模型 816 中 LDial 或 IDial 的输出端与之逻辑耦合的每个配置锁存器的各自锁存器数据结构 1204。每个锁存器数据结构 1204 包括父字段 1240，父字段 1240 包含指向直接控制相应配置锁存器的 LDial 或 IDial 的 DIDS 1200 的父指针 1242。另外，锁存器数据结构 1204 包括与包含通过父指针 1242 识别的 Dial 示例的实体有关的、指定分层锁存器名的锁存器名字段 1244。例如，如果含有示例标识符 a.b.c 的 LDial X 引用具有分层名“a.b.c.latch1”的配置锁存器，锁存器名字段 1244 将包含字符串“d.latch1”。因此，将通过父指针 1242 识别的 DIDS 1202 的实例名字段 1234 的内容预先附在锁存器名字段 1244 的内容上提供了可利用配置数据库 814 配置的给定配置锁存器的任何实例的完全限定名。

仍然参照图 12A，如上所述，配置数据库 814 包括顶层指针阵列 1206，和可选地，Dial 指针阵列 1208 和锁存器指针阵列 1210。顶层指针阵列 1206 包含对于每个顶层 Dial 和每个顶层 Dial 组，指向顶层实体实例的相关 DIDS 1202 的顶层指针 1250。Dial 指针阵列 1208 包括指向配置数据库 814 中的每个 DDDS 1200，以便允许通过 Dial 和/或实体名间接访问特定 Dial 实例的 Dial 指针 1252。最后，锁存器指针阵列 1210 包括指向配置数据库 814 内的每个锁存器数据结构，以便允许容易访问所有配置锁存器的锁存器指针 1254。

一旦配置数据库 814 被构造出来，就可以将配置数据库 814 的内容装入非易失性存储器，譬如，图 1 的数据处理系统 8 的系统存储器 18 中，以便适当地配置用于模拟的模拟模型。一般说来，可以将数据结构 1200、1202、1204

和 1206 直接装入系统存储器 18 中，和可选地，如下所述，可以用附加字段扩充。但是，如上所述，如果希望使配置数据库 814 的非易失性图像更紧凑，生成系统存储器 18 中的易失性配置数据库中，诸如 Dial 指针阵列 1208、锁存器指针阵列 1210 和实例指针阵列 1226 之类的附加数据结构是有用的。

现在参照图 12B，图 12B 例示了按照本发明的包括代表 Dial 和 Register 的数据结构的示范性模拟配置数据库 814 的一部分的更详细图形。为了避免不必要的复杂，从图 12B 中除去如图 12A 所示的模拟配置数据库 814 的一些特征。

现在转到对该图的考虑，图 12B 例示了代表被 Dial 和 Register 引用的、诸如图 7C 的配置锁存器 705a 之类的锁存器的锁存器数据结构 1204′。正如上面参照图 12 所讨论的那样，在模拟配置数据库 814 内通过 DIDS 1202a 表示引用锁存器的 Dial。为了表示 Dial 和锁存器之间的逻辑连接，锁存器数据结构 1204 包括 Dial 父字段 1240a，Dial 父字段 1240a 包含指向 DIDS 1202a 的 Dial 父指针 1242a，和 DIDS 1202a 包括指向锁存器数据结构 1204′以便将锁存器标识成父 Dial 的“孩子”的输出指针阵列 1236 内的输出指针 1238n。

在模拟配置数据库 814 内通过 DIDS 1202b 表示引用锁存器的 Register。正如相似的标号所指的那样，在优选实施例中，与 DIDS 1202a 相似地构造 DIDS 1202b 是有利的。具体地说，DIDS 1202b 包括定义字段 1230，定义字段 1230 包含指向 DDDS 1200(在图 12B 中未示出)的定义指针 1231，DDDS 1200 定义 Register 实体，因此，含有对于 Register 具有“R”的值的类型字段 1220（应该注意到，由于 Register 不映射输入值或输出端，定义 Register 实体的 DDDS 1200 含有 NULL 或缺值映射表 1224）。正如上面参照图 12A 所讨论的那样，DIDS 1202b 还可以包括实例名字段 1234 和输出指针阵列 1236。DIDS 1202b 的输出指针阵列 1236 内的输出指针 1238a 将锁存器数据结构 1204′标识成代表 Register 的子锁存器。类似地，Register 和锁存器之间的关系通过锁存器数据结构 1204′的 Register 父字段 1240b 内的 Register 父指针 1242b 编入文档。

如图所示，正如上面参照图 12A 所讨论的那样，DIDS 1202b 还可以包括父字段 1232、默认值字段 1229、和阶段 ID 字段 1227。但是，如果实施上面参照图 7C 讨论的规则集，那么，父字段 1232、默认值字段 1229、和阶段 ID 字段 1227 是 NULL，或者可以省略父字段 1232、默认值字段 1229、和阶段 ID

字段1227，因为，根据上面给出的规则集，Register是没有父辈和不允许使用默认值的顶层实体。

在如下的描述中，详细描述Dial实例和它们的底层锁存器的命名、设置、和读取。除了下面加注解的地方之外，凭借代表配置数据库814中的Dial和Register实例的数据结构的公用设计以相同的方式访问Register实例和它们的底层锁存器（可以与Dial实例共享）。

现在参照图13，图13描绘了在数据处理系统的易失性存储器，譬如，数据处理系统8的系统存储器18内扩展配置数据库814的方法的高级逻辑流程图。由于图13描绘了逻辑步骤，而不是操作步骤，应该明白，例示在图13中的许多步骤可以同时进行或以与所示的顺序不同的顺序进行。

如图所示，该过程从方块1300开始，然后转到方块1302，方块1302例示了数据处理系统6将配置数据库814内的现在数据结构从非易失性存储器（例如，盘状存储器或闪速存储器）复制到易失性系统存储器18。接着，在方块1304上，确定配置数据库814的顶层指针阵列1206内的所有顶层指针1250是否都得到处理。如果是，该过程转到下面讨论的方块1320。如果不是，该过程转到方块1306，方块1306例示了从顶层指针阵列1206中选择下一个要处理的顶层指针1250。

然后，在方块1308上确定通过所选顶层指针1250识别的DIDS 1202内的父指针1233是否是NULL指针。如果不是，这表明DIDS 1202描述属于Dial组的顶层Dial，该过程返回到方块1304，表明当处理它所属的Dial组时，将处理顶层Dial和它的相关较低层Dial。

响应在方块1308上父指针1233是NULL指针的确定，像在方块1310上描绘的那样，数据处理系统8在DIDS 1202的定义字段1230中的定义指针1231所指的DDDS 1200的实例指针阵列1226中创建指向DIDS 1202的实例指针1228。接着，如果Dial指针1252不是多余的，在方块1312上，数据处理系统8在Dial指针阵列1208中创建指向顶层Dial的DDDS 1200的Dial指针1252。另外，如方块1314所示，数据处理系统8创建指向顶层Dial的DIDS 1202的输出指针1238引用的每个锁存器数据结构1204（若有的话）、Dial指针阵列1210内的锁存器指针1254。然后，如方块1316所示，通过执行如方块1310-1316所示的功能，类似地处理所选顶层指针1250引用的顶层Dial领头的Dial树（若有的话）的每个较低层上的每条分支，直到找出和

处理完终止在那个分支上的锁存器数据结构 1204 为止。然后，该过程返回到方块 1304，代表处理顶层指针阵列 1206 内的每个顶层指针 1250。

响应在方块 1304 上所有顶层指针 1250 都得到处理的确定，如图 13 所示的过程转到方块 1320。方块 1320 例示了在配置数据库中的每个 DIDS 1320 中创建实例设置字段 1239。如上所述，实例设置字段 1239 是被初始化成 FALSE 和当相关 Dial 或 Register 实例得到明确设置时被更新成 TRUE 的布尔值字段。另外，正如在方块 1322 上所描绘的那样，数据处理系统 8 在每个锁存器数据结构 1204 中创建锁存器值字段 1246、锁存器 Register 设置字段 1247、锁存器设置字段 1248 和设置历史字段 1249，分别指示相关配置锁存器的当前设置值，指示是否通过相关 Register 实例设置了配置锁存器，指示当前是否通过明确设置命令设置了配置锁存器，和指示是否曾经明确设置过配置锁存器。尽管为了简洁起见，与在方块 1304-1316 上描绘的处理分开地例示了在方块 1320-1322 上所指的 5 个字段的创建，但是，应该认识到，像处理每个 DIDS 1202 那样创建实例设置字段 1239 和像到达处在每个 Dial（或 Register）树低部的锁存器数据结构 1204 那样创建字段 1246、1247、1248 和 1249 更加有效。此后在方块 1324 上终止将配置数据库装入易失性存储器的过程。

借助于装入易失性存储器的配置数据库，模拟模型可以被配置成用于通过模拟软件的执行模拟数字设计。参照图 14，图 14 例示了描绘在模拟模型的模拟运行期间系统存储器 18（图 1）的内容的方块图。如图所示，系统存储器 18 包括模拟模型 1400，以及包括配置 API 1406、模拟器 1410 和 RTX（运行期执行程序）1420 的软件，模拟模型 1400 是要模拟的数字设计的逻辑表示。

模拟器 1410 将诸如模拟模型 1400 之类的模拟模型装入系统存储器 18 中。在模拟运行期间，模拟器 1410 通过各种各样 API 1416 重新设置时钟和评估模拟模型 1400。另外，模拟器 1410 利用 GETFAC API 1412 读取模拟模型 1400 中的值，和利用 PUTFAC API 1414 将值写入模拟模型 1400 中。尽管在图 14 中模拟器 1410 完全用软件实现的，但应该认识到，在下文中，可替代地，模拟器也可以至少部分用软件来实现。

配置 API 1406 包括支持模拟模型 1400 的配置、通常用诸如 C 或 C++之类的高级语言写成的软件。在需要的时候由模拟器动态地装入的这些 API 包

括从非易失性存储器装入配置数据库814和以上面参照图13所述的方式扩展它以提供配置数据库1404的存储图像的第一API。正如下面详述的那样，配置API 1406进一步包括访问和操纵配置数据库1404的附加API。

RTX 1420控制诸如模拟模型1400之类的模拟模型的模拟。例如,RTX 1420装载测试用例以便应用于模拟模型1400。另外，RTX 1420将一组API调用传送给配置API 1406和模拟器1410提供的API，以便初始化、配置和模拟模拟模型1400的操作。在模拟期间和模拟之后，RTX 1420还调用配置API 1406和模拟器140提供的API，通过访问模拟模型1400内的Dial、Register、配置锁存器、计数器和其它实体，检验模拟模型1400的正确性。

RTX 1420还拥有两种它访问在模拟模型1400中例示的Dial的模式：交互模式和成批模式。在交互模式中，RTX 1420调用第一组API从配置数据库1404中读取特定Dial的一个或多个实例或将特定Dial的一个或多个实例写入配置数据库1404中。通过参照配置数据库1404获得的锁存器值直接影响模拟模型1400。在成批模式中，RTX 1420调用不同的第二组API，读取配置数据库1404中的多个Dial的示例或将多个Dial的示例写入配置数据库1404中，然后，同时对模拟模型1400作一些改变。

在交互或成批模式中，RTX 1420都必须在它的API调用中应用某种语法，以指定要访问模拟模型1400内的哪些Dial或Dial组实例。尽管可以应用许多不同语法，包括应用通配符的传统常规表达式，但在例示性实施例中，用于在API调用中指定Dial或Dial组实例的语法与上文所述的简明表达式相似。上面讨论的简明表达式与用于在RTX API调用中指定Dial或Dial组实例的语法之间的主要差异是，在例示性实施例中，在RTX API调用中，参照模拟模型1400的顶层设计实体，而不是与指定Dial或Dial组的设计实体有关地指定Dial和Dial组实例。

在例示性实施例中，目标是模拟模型1400中的一个或多个Dial或Dial组实例的每个RTX API调用利用两个参数指定Dial或Dial组实例：实例限定符和拨号器名限定符。为了只引用单个Dial或Dial组示例，实例限定符采取“a.b.c.d”的形式，“a.b.c.d”是出现单个Dial或Dial组示例的设计实体的分层示例标识符。为了引用多个Dial或Dial组示例，实例限定符采取“a.b.c.[X]”的形式，“a.b.c.[X]”标识在实体实例a.b.c的范围内实体X的所的示例。在退化形式中，实例限定符可以只是“[X]”，“[X]”标识模

拟模型 1400 内任何地方的实体 X 的所的示例。

拨号器名限定符最好采取“Entity.dialname”的形式，其中，“Entity”是例示 Dial 或 Dial 组的设计实体，和“dialname”是在它的配置说明语句中指定给 Dial 或 Dial 组的名称。如果应用带括号语法来指定实例限定符，那么，由于与带括号实体名一致，可以从拨号器名限定符中去掉“Entity”字段。

现在参照图 15，图 15 描绘了按照本发明，配置 API 1406 根据实例限定符和拨号器名限定符对，在配置数据库 1404 中定位特定 Dial 或 Dial 组实例的示范性过程的高级逻辑流程图。如图所示，响应配置 API 1406 对包含上面所讨论的实例限定符和拨号器名限定符的来自 RTX 1420 的 API 调用的接收，该过程从方块 1500 开始。正如在方块 1502 上所描绘的那样，响应 API 调用，配置 API 1406 在 Dial 指针阵列 1208 上输入配置数据库 1404，并且，正如在方块 1504 上所例示的那样，利用 Dial 指针 1252 定位具有与指定拨号器名限定符完全匹配的名称字段 1222 的 DDDS 1200。

接着，在方块 1506 上，配置 API 1406 确定实例限定符是否像上述那样，应用带括号语法。如果是，该过程转到如下所述的方块 1520。但是，如果实例限定符不应用带括号语法，配置 API 1406 沿着匹配 DDDS 1200 的实例指针 1228 定位具有与指定实例限定符完全匹配的实例名字段 1234 的单个 DIDS 1202。正如在方块 1510-1512 上所指的那样，如果发现不匹配，该过程以错误告终。但是，如果匹配 DIDS 1202 得到定位，在方块 1524 上创建标识单个匹配 DIDS 1202 的临时“结果”指针。此后，在方块 1526 上终止该过程。

返回到方块 1520，如果应用带括号语法，配置 API 1406 利用匹配 DDDS 1200 的实例指针 1228 在括号之前的实例标识符的前缀部分指定的范围内定位 Dial 或 Dial 组实例的一个或多个 DIDS 1202。也就是说，如果 DIDS 1202 的实例名字段 1234 包含实例标识符的前缀部分，那么，认为 DIDS 1202 匹配。此外，如果发现不匹配，该过程穿过方块 1522 和在方块 1512 上以错误告终。但是，如果一个或多个 DIDS 1202 与实例限定符“匹配”，在方块 1524 上构建标识匹配 DIDS 1202 的临时结果指针。此后，在方块 1526 上终止如图 15 所示的过程。

现在参照图 16A，图 16A 例示了按照本发明，RTX 1420 以交互模式读取一个或多个 Dial 实例的值的示范性过程的高级逻辑流程图。如图所示，响应

配置 API 1406 对来自 RTX 1420 的 read_Dial()API 调用的接收，该过程从方块 1600 开始。正如在方块 1602 上所指的那样，配置 API 1406 通过利用上面参照图 15 所述的过程，在配置数据库 1404 中定位响应 API 调用的 Dial 实例的一个或多个 DIDS 1202，对 read_Dial()API 调用作出响应。

然后，该过程在方块 1604 上进入处理图 15 的过程生成的每个临时结果指针的循环。如果图 15 的过程返回的所有结果指针都得到处理，该过程转到如下所述的方块 1640。如果没有，该过程从方块 1606 转到方块 1608，方块 1608 例示了配置 API 1406 选择下一个要处理的结果指针。接着，在方块 1608 上，配置 API 1406 通过引用与当前结果指针标识的 DIDS 1202 相联系的 DDDS 1200 的类型字段 1220，确定 DIDS 1202 是否与 Dial 组相对应。如果是，如图 16A 所示的过程在方块 1610 上以错误状态告终，方块 1610 指出 RTX 1420 利用错误 API 调用读取 Dial 实例。

响应在方块 1608 上当前结果指针标识的 DIDS 1202 不对应于 Dial 组实例的确定，该过程转到方块 1620。方块 1620 描绘了配置 API 1406 利用 DIDS 1202 的输出指针 1238(和 Dial 树中任何较低层 DIDS 1202 的那些输出指针)，从与最终受在 API 调用中指定的 Dial 实例控制的所有配置锁存器相对应的锁存器数据结构 1204 的锁存器名字段 1244 中建立包含锁存器名的数据集。接着，正如在方块 1622 上所描绘的那样，配置 API 1406 使一个或多个 API 调用模拟器 1410 的 GETFAC()API 1412，从模拟模型 1400 中获取列在在方块 1620 上构成的数据集中的所有配置锁存器的锁存器值。

然后，如方块 1624 所示，配置 API 1406 参照配置数据库 1404，核实从模拟模型 1400 中获得的锁存器值。为了核实锁存器值，配置 API 1406 利用映射表 1224 使锁存器值从相应锁存器数据结构经过中间 DIDS 1202（若有的话）沿着 Dial 树向上传播，直到所请求 Dial 实例的输入值得到确定为止。如果在这个核实过程中的任何点上，由核实过程生成的 Dial 实例输出值不对应于在它的映射表 1224 中枚举的合法值之一，就会在方块 1626 上检测到一个错误。于是，正如在方块 1630 上所指的那样，将从模拟模型 1400 读取的锁存器值和错误指示符放入结果数据结构中。如果没有检测到错误，如方块 1628 所示，将由核实过程生成的 Dial 输入值和成功指示符放入结果数据结构中。

正如返回到方块 1604 的过程所指的那样，对于图 15 的过程返回的每个

临时结果指针，重复上述过程。一旦所有结果指针都得到处理，该过程从方块 1604 转到方块 1640-1642，方块 1640-1642 例示了配置 API 1406 将结果数据结构返回给 RTX 1420，然后终止。

例如，RTX 1420 利用图 16A 的方法，以交互模式读取 Dial 实例，以便初始化在模拟运行期间监视模拟模型 1400 的一些部分的检验器。感兴趣的 Dial 设置不仅包括顶层 Dial 实例的那些，而且包括与检验器监视的模拟模型 1400 的一些部分有关系的较低层 Dial 实例的那些。

现在参照图 16B，图 16B 例示了按照本发明，RTX 1420 以交互模式读取一个或多个 Dial 组实例的值的示范性过程的高级逻辑流程图。比较图 16A 和 16B 可以看出，读取 Dial 组实例的过程与读取 Dial 实例的过程相似，但返回可能不同 Dial 实体的一个或多个顶层 Dial 实例的值，而不是相同 Dial 实体的一个或多个实例。

如图所示，响应配置 API 1406 对来自 RTX 1420 的 read_Dial_group()API 调用的接收，如图 16B 所示的过程从方块 1650 开始。正如在方块 1652 上所指的那样，配置 API 1406 通过利用上面参照图 15 所述的过程，在配置数据库 1404 中定位响应 API 调用的 Dial 组实例的一个或多个 DIDS 1202，对 read_Dial_group ()API 调用作出响应。

然后，该过程在方块 1654 上进入处理图 15 的过程生成的每个临时结果指针的循环。如果图 15 的过程返回的所有结果指针都得到处理，该过程转到如下所述的方块 1680。如果没有，该过程从方块 1654 转到方块 1656，方块 1656 例示了配置 API 1406 选择下一个要处理的结果指针。接着，在方块 1658 上，配置 API 1406 标识和创建临时指针，临时指针指向属于与当前结果指针引用的 DIDS 1202 相对应的 Dial 组实例的所有顶层 Dial 组实例。顶层 Dial 组实例通过对相关 DDDS 1220 中的类型字段 1220 指定除了 Dial 组之外的类型的每个输出指针 1238 定位最高层 DIDS 1202 来识别。换句话说，配置 API 1406 必须向下搜索一个或多个分层 Dial 组，以定位相关顶层 Dial 实例。

然后，如图 16B 所示的过程进入从方块 1659 开始的循环，在这个循环中分别处理属于与当前结果指针引用的 Dial 组 DIDS 1202 相对应的 Dial 组的每个顶层 Dial 实例，以获得顶层 Dial 实例值。该过程接着转到方块 1660，方块 1660 描绘了配置 API 1406 利用第一（或下一个）顶层 Dial 实例的 DIDS 1202 的输出指针 1238(和 Dial 树中任何较低层 DIDS 1202 的那些输出指针)，

从与最终受顶层 Dial 实例控制的所有配置锁存器相对应的锁存器数据结构 1204 的锁存器名字段 1244 中构建包含锁存器名的数据集。接着，正如在方块 1662 上所描绘的那样，配置 API 1406 使一个或多个 API 调用模拟器 1410 的 GETFAC() API 1412，从模拟模型 1400 中获取列在在方块 1660 上构成的数据集中的所有配置锁存器的锁存器值。

然后，在方块 1664 上，配置 API 1406 利用上面参照图 16A 的方块 1624 所述的相同技术，参照配置数据库 1404，核实从模拟模型 1400 中获得的锁存器值。如果在这个核实过程中的任何点上，由核实过程生成的 Dial 实例输出值不对应于在它的映射表 1224 中枚举的合法值之一，就会在方块 1666 上检测到一个错误。于是，正如在方块 1670 上所例示的那样，将从模拟模型 1400 读取的锁存器值和错误指示符放入结果数据结构中。如果没有检测到错误，如方块 1668 所示，将由核实过程生成的 Dial 输入值和成功指示符放入结果数据结构中。

在方块 1668 或方块 1670 之后，该过程返回到方块 1659，方块 1659 代表属于与当前结果指针引用的 DIDS 1202 相对应的 Dial 组的所有顶层 Dial 是否都得到处理的确定。如果没有，该过程返回到已经描述过的方块 1660。但是，如果所有顶层 Dial 都已经得到处理，该过程返回到方块 1654，方块 1654 例示了所有结果指针是否都得到处理的确定。如果没有，在已经描述的方块 1656 和随后方块上处理下一个结果指针。但是，如果所有结果指针都得到处理，该过程转到方块 1680-1682，方块 1680-1682 例示了配置 API 1406 将结果数据结构返回给 RTX 1420，然后终止。

以 RTX 1420 的成批模式读取 Dial 和 Dial 组实例最好由配置 API 1406 以与交互模式相同的方式管理，只有一种例外。虽然在交互模式中，在方块 1622 和 1662 上通过对 GETFAC() API 1412 的调用总是从模拟模型 1440 中读取锁存器数据，但是在成批模式中，如果锁存器设置字段 1248 指示相应配置锁存器都已经得到设置，最好从配置数据库 1404 中的锁存器数据结构 1204 的锁存器值字段 1246 中获取锁存器值。如果配置锁存器还没有得到设置，通过对 GETFAC() API 1412 的调用从模拟模型 1440 中获取锁存器数据。这种差异保证了可能还没有反映在模拟模型 1400 中的、在成批模式下作出的 Dial 设置得到正确报告。

现在参照图 17A，图 17A 例示了按照本发明，RTX 以交互模式设置 Dial

实例的示范性过程的高级逻辑流程图。响应配置 API 1406 对来自 RTX 1420 的 set_Dial() API 调用的接收，该过程从方块 1700 开始。正如在方块 1702 上所例示的那样，响应 set_Dial() API 调用，配置 API 1406 首先利用上面参照图 15 所述的技术，定位和生成指向在 set_Dial() API 调用中指定的 Dial 实例的 DIDS 1202 的临时结果指针。接着，配置 API 1406 在方块 1704 上确定所有临时结果指针是否都指向顶层 Dial 实例的 DIDS 1202。这个确定可以通过，例如，检查每个这样的 DIDS 1202 的父指针 1233（和通过父指针 1233 链接的任何较高层 DIDS 1202 的父指针）和相关 DDDS 1200 的类型字段 1220 作出。顶层 Dial 实例的 DIDS 1202 拥有 NULL 父指针 1233 或指向相关 DDDS 1200 的类型字段 1220 所指的另一个 DIDS 1202 的非 NULL 父指针 1233。如果结果指针引用的任何 DIDS 1202 不对应于顶层 Dial 实例，该过程在方块 1708 上以错误状态告终。

响应在方块 1704 上结果指针引用的所有 DIDS 1202 都对应于顶层 Dial 实例的确定，在方块 1706 上进一步确定要对 Dial 实例设置的指定值是否是在相关 DDDS 1200 的映射表 1224 中指定的值之一。如果不是，该过程在方块 1708 上以错误告终。但是，响应在方块 1706 上要对 Dial 实例设置的指定值是合法值之一的确定，该过程进入包括方块 1710-1716 的循环，在这个循环中处理每个结果指针以设置各自 Dial 实例。

在方块 1710 上，配置 API 1406 确定所有结果指针是否都得到处理。如果是，在方块 1720 上终止该过程。但是，如果另外的结果指针仍然需要处理，在方块 1712 上选择下一个要处理的结果指针。接着，在方块 1714 上，配置 API 1406 沿着与当前结果指针引用的 DIDS 1202 相联系的顶层 Dial 实例领头的 Dial 树向下传播在 set_Dial() API 调用中指定的 Dial 设置。为了传播所需 Dial 设置，如有必要（即，对于 CDial 和 LDial），首先引用与当前结果指针引用的 DIDS 1202 相联系的 DDDS 1200 中的映射表 1224，以便为当前结果指针引用的 DIDS 1202 的输出指针阵列 1236 中的每个输出指针 1238 确定输出值。沿着 Dial 树向下传播这些输出值，作为与输出指针 1238 引用的 DIDS 1202 相对应的下一个较低层 Dial 实例（若有的话）的输入值。这个传播一直持续到为处在 Dial 树末端的每个配置锁存器（在配置数据库 1404 中用锁存器数据结构 1204 表示它们）确定锁存器值为止。如方块 1716 所示，随着配置锁存器的每个锁存器值得到确定，配置 API 1406 调用 PUTFAC() API

1414，利用在相应锁存器数据结构 1204 的锁存器名字段 1244 内指定的锁存器名将模拟模型 1400 中的配置锁存器设置成确定的值。

此后，该过程返回到方块 1710，方块 1710 代表与下一个结果指针相对应的顶层 Dial 的处理。在所有结果指针都得到处理之后，在方块 1720 上终止该过程。

现在参照图 17B，图 17B 例示了按照本发明，RTX 以交互模式设置 Dial 组实例的示范性过程的高级逻辑流程图。响应配置 API 1406 对来自 RTX 1420 的 set_Dial_group() API 调用的接收，该过程从方块 1730 开始。正如在方块 1732 上所描绘的那样，响应 set_Dial_group() API 调用，配置 API 1406 首先利用上面参照图 15 所述的技术，定位和生成指向在 set_Dial_group() API 调用中指定的 Dial 组实例的 DIDS 1202 的临时结果指针。接着，配置 API 1406 在方块 1734 上确定所有临时结果指针是否都指向顶层 Dial 组实例的 DIDS 1202。这个确定可以通过，例如，检查每个这样的 DIDS 1202 的父指针 1233 以查明父指针 1233 是否是 NULL 作出。如果结果指针引用的任何 DIDS 1202 都不对应于顶层 Dial 组实例（即，存在非 NULL 父指针 1233），该过程在方块 1736 上以错误状态告终。

响应在方块 1734 上结果指针引用的每个 DIDS 1202 都对应于顶层 Dial 组的确定，该过程转到方块 1738-1740。方块 1738 例示了配置 API 1406 在结果指针引用相应 DIDS 1202 的每个 Dial 组内定位所有顶层 Dial 实例。然后，正如在方块 1740 上所描绘的那样，配置 API 1406 确定要对每个顶层 Dial 实例设置的指定值是否是在相关 DDDS 1200 的映射表 1224 中指定的值之一。如果不是，该过程在方块 1736 上以错误告终。

在所例示的实施例中，由于人们认为将设置 Dial 组实例实现成成功设置所有相关顶层 Dial 实例或完全失败的不可分操作更可取，所以在设置任何 Dial 组实例之前，执行在方块 1734、1738 和 1740 上例示的预先确认步骤。这样，可以避免 Dial 组实例内的一些顶层 Dial 实例得到设置，而其它则没有的复杂状况。

响应在方块 1740 上要对每个顶层 Dial 实例设置的指定值是合法值之一的确定，该过程进入包括方块 1750-1756 的循环，在这个循环中处理每个结果指针以设置属于每个 Dial 组实例的顶层 Dial 实例。

在方块 1750 上，配置 API 1406 确定所有结果指针是否都得到处理。如

果是，在方块 1760 上终止该过程。但是，如果还有另外的结果指针需要处理，在方块 1752 上选择下一个要处理的结果指针。接着，在方块 1754 上，配置 API 1406 沿着属于与当前结果指针引用的 DIDS 1202 相对应的顶层 Dial 组实例的顶层 Dial 实例的 Dial 树向下传播在 set_Dial_group() API 调用中为每个顶层 Dial 指定的 Dial 设置。Dial 设置沿着 Dial 树向下传播以上面参照图 17A 的方块 1714 所讨论的相同方式进行，如方块 1756 所示，随着配置锁存器的每个锁存器值得到确定，配置 API 1406 调用 PUTFAC() API 1414，利用在相应锁存器数据结构 1204 的锁存器名字段 1244 内指定的锁存器名将模拟模型 1400 中的配置锁存器设置成确定的值。此后，该过程返回到方块 1750，方块 1710 代表与下一个结果指针（若有的话）相对应的顶层 Dial 的处理。

现在参照图 18A，图 18A 例示了按照本发明以成批模式设置 Dial 实例、Dial 组实例、和 Register 实例的示范性方法的高级逻辑流程图。如图所示，该过程从方块 1800 开始，然后转到方块 1802，方块 1802 例示了 RTX 1420 通过调用配置 API 1406（例如，start_batch()）初始化配置数据库 1404，以便初始化配置数据库 1404。start_batch() API 例程通过，例如，将配置数据库 1404 中的每个实例设置字段 1239、锁存器 Register 设置字段 1247、锁存器设置字段 1248、和设置历史字段 1249 设置成 FALSE，初始化配置数据库 1404。正如下面所讨论的那样，通过重新设置配置数据库 1404 中的所有“设置”字段，可以容易地检测当前成批模式调用序列未设置的 Dial、Register 和配置锁存器。重要的是，如果随后在成批模式调用序列中设置了任何锁存器 Register 设置字段 1247 或设置历史字段 1249，这些字段在默认值应用的所有阶段都维持设置不变（即，这些字段是永久的）。

在配置数据库 1404 在方块 1802 上被初始化之后，如图 18A 所示的过程转到方块 1804。方块 1804 例示了 RTX 1420 可选地发出一个或多个 read_Dial()或 read_Dial_group() API 调用，以便像上面参照图 16A 和 16B 所讨论的那样，读取一个或多个 Dial、Register 或 Dial 组，和可选地发出一个或多个成批模式 set_Dial()或 set_Dial_group() API 调用，以便将 Dial 和 Register 实例和它们的底层配置锁存器的设置输入配置数据库 1404 中。配置 API 1406 以上面参照图 17A（对于设置 Dial 和 Register 实例）或图 17B（对于设置 Dial 组实例）所述的相同方式响应“设置” API 调用，只有两种

例外。第一，当设置任何顶层或较低层 Dial 或 Register 实例时，无论是作为 set_Dial() API 调用的结果还是作为 set_Dial_group() API 调用的结果，都将相应 DIDS 1202 的实例设置字段 1239 设置成 TRUE。第二，正如在图 17A-17B 的方块 1716 和 1756 上所例示的那样，通过“设置” API 例程不将锁存器值写入模拟模型 1400 中。取而代之，将锁存器值写入与每个受影响配置锁存器相对应的锁存器数据结构 1204 的锁存器值字段 1246 中，并且将锁存器设置字段 1248 更新成 TRUE。这样，在随后处理中可以容易地识别通过 API 调用明确设置的 Dial 和 Register 实例和配置锁存器。

在方块 1804 之后，该过程转到方块 1806，方块 1806 例示了 RTX 1420 调用配置 API 1406 当中的 end_batch() API 例程，以便完成默认值应用的当前阶段。正如在方块 1806 上所指的那样和正如下面参照图 18B 详述的那样，end_ batch() API 例程将所选默认值(若有的话)应用于指定 Dial 和 Register 实例和将这些默认值传播到底层配置锁存器，再传播到配置数据库 1404。然后，明确地或用默认值设置的所有配置锁存器的锁存器值有可能应用于模拟模型内的锁存器。最后，为下一个阶段（若有的话）作准备。

如果 RTX 1420 存在默认值应用的另外阶段，该过程从方块 1806 转到方块 1808，然后，返回到方块 1804，方块 1804 表示 RTX 1420 初始化默认值应用的下一个阶段。但是，如果默认值应用的所有阶段都得到处理，如图 18A 所示的过程从方块 1806 经过方块 1808 转到方块 1810，在方块 1810 上终止成批过程。

现在参照图 18B,图 18B描绘了在图 18A 的方块 1806 上调用的 end_phase() API 例程的示范性实施例的高级逻辑流程图。如图所示，当 RTX 1420 借助于，例如，如下语句调用 end_phase() API 例程时，该过程从方块 1820 开始:

```
End_phase(phases,unnamed,instance_qualifier,apply)
```

在这个示范性 API 调用中，“phases”参数是指定在当前阶段结束时要应用的默认值阶段 ID 的字符串；“unnamed”是指示在当前阶段中是否应该应用没有任何相关阶段 ID 的默认值的布尔参数；“apply”是指示配置锁存器值是否应该立即应用于模拟模型 1400 的布尔值参数；和“instance_qualifier”是可以用于限制处理特定 Dial 的哪些实例以便应用默认值的一个或多个常

规表达式。

通过为 end_phase() API 例程指定 instance_qualifier 参数，用户可以使默认值的应用只限于模拟模型 1400 的一部分。在模拟模型 1400 的两个部分（例如，代表两个不同集成电路芯片的部分）存在不同阶段调整要求但使用相同阶段 ID 的情况下，以这种方式限制默认值的应用的能力尤其有用。因此，通过适当地指定与阶段 ID 一起使用的 instance_qualifier，可以解决阶段 ID 冲突问题。

然后，end_phase() API 例程进入包括方块 1822-1838 的处理循环，在这个处理循环中处理配置数据库 1404 内的 DIDS 1202，以便应用适当的 Dial 默认值（若有的话）。首先参照方块 1822，end_phase() API 例程确定顶层指针阵列 1206 内的所有顶层指针 1250 是否都得到处理。如果是，该过程从方块 1822 转到如下所述的方块 1840。如果顶层指针阵列 1206 内的顶层指针 1250 还没有都得到处理，该过程转到方块 1824。方块 1824 代表 end_phase() API 例程递归地扫描下一个顶层指针 1250 所指的 DIDS 1202 和它的后代 DIDS 1202（若有的话），以便应用 end_phase() API 调用的参数所指的默认值。如果 end_phase() API 例程在方块 1826 上确定通过当前顶层指针 1250 识别的顶层 DIDS 1202 的子树中的所有必要 DIDS 1202 都得到处理，那么，该过程返回到已经描述过的方块 1822。但是，如果通过当前顶层指针 1250 识别的顶层 DIDS 1202 的子树中的至少一个 DIDS 1202 仍然需要处理，该过程从方块 1826 转到方块 1828。

方块 1828 例示了检查下一个 DIDS 1202 以确定它的默认值字段 1229 是否是非 NULL 值的 end_phase() API 例程。如果当前 DIDS 1202 没有包含非 NULL 默认值字段 1229，该过程返回到方块 1824，方块 1824 代表 end_phase() API 例程继续进行当前顶层指针 1250 所指的顶层 DIDS 1202 的子树中的 DIDS 1202 的递归处理。如果默认值字段 1229 包含非 NULL 值，该过程转到方块 1830，方块 1830 描绘是否设置实例设置字段 1239，即，以前在图 18A 的方块 1804 上是否明确地设置了 Dial 实例的确定。如果设置了实例设置字段 1239，忽略包含在默认值字段 1229 中的默认值（由于模拟用户已经明确地指定了相关 Dial 实例的值）。并且，由于模拟模型 1400 被构造成具有指定默认值的 DIDS 1202 的任何后代都不能有默认值，该过程转到方块 1836，方块 1836 例示了 end_phase() API 例程跳过当前 DIDS 1202 的子树中的任何 DIDS 1202

的处理。此后，该过程返回到已经描述过的方块1824。

返回到方块1830，响应没有设置当前DIDS 1202的实例设置字段1239的确定，该过程转到方块1832。方块1832例示了end_phase() API查询当前DIDS 1202的阶段ID字段1227，以确定存储在默认值字段1229中的默认值是否含有一个或多个相关阶段ID。如果没有，该过程转到如下所述的方块1833。响应在方块1832上阶段ID字段1227存储至少一个阶段ID的确定，end_phase() API接着在方块1834上确定end_phase() API调用的阶段参数是否指定了与包含在阶段ID字段1227内的阶段ID匹配的阶段ID。如果发现不匹配，该过程从方块1834转到已经描述过的方块1836。另一方面，如果在end_phase() API调用的阶段参数中指定的阶段ID与包含在当前DIDS 1202的阶段ID字段1227内的阶段ID匹配，end_phase() API接着在方块1835上确定包含在当前DIDS 1202的实例名字段1234中的Dial实例名是否与作为end_phase() API调用的instance_qualifier参数传递的限定表达式匹配。此外，响应在方块1835上的否定确定，该过程转到已经描述过的方块1836。另一方面，如果instance_qualifier参数限定了包含在实例名字段1234内的Dial实例名，该过程转到如下所述的方块1838。

返回到方块1833，如果当前DIDS 1202不含在阶段ID字段1227内指定的一个或多个阶段ID，进一步确定end_phase() API调用的unnamed参数是否含有指示在当前阶段应该应用没有任何相关阶段信息的默认值的TRUE的值。如果没有，该过程从方块1833转到已经描述过的方块1836。另一方面，如果end_phase() API在方块1833上确定在当前阶段应该应用没有任何相关阶段信息的默认值，该过程转到上面已经描述过的方块1835。

因此，当end_phase() API到达方块1838时，end_phase() API通过在1830、1832、1833、1834和1835上的确定，已经确定在成批模式执行的当前阶段应该应用为与当前DIDS 1202相对应的Dial实例指定的默认值。于是，在方块1838上，end_phase() API例程将在默认值字段1229中指定的默认值应用于映射表1224，生成然后以如上所述的方式沿着当前DIDS 1202的Dial树向下传播的一个或多个Dial输出信号。最后，如果没有设置锁存器数据结构1204的锁存器Register设置字段1247，将配置数据库1404内每个底层锁存器数据结构1204的锁存器值字段1246和锁存器设置字段1248设置成与Dial默认值相对应的值。也就是说，默认值最好只有在以前没有通过

Register 设置锁存器时才应用于锁存器。如果在配置过程的任何前面阶段曾经通过 Register 设置过锁存器，不应用默认值（至少直到再次调用 start_batch() API 为止）。然后，该过程从方块 1838 转到已经描述过的方块 1836。

返回到方块 1822，响应顶层指针 1250 所指的所有 DIDS 1202 的 Dial 树已经得到处理以如上所述的方式应用任何适当默认值的确定，该过程接着转到方块 1840。方块 1840 描绘了 end_phase() API 检查 end_phase() API 调用的 apply 参数，以确定锁存器数据结构 1204 内的配置锁存器值是否应该应用于模拟模型 1400。加上这个确定所表示的控制度的有利之处在于，可以在不同阶段在配置数据库 1404 内独立地配置可能存在冲突阶段 ID 的模拟模型 1400 的不同部分，但如有需要，所得配置锁存器值可以同时应用于模拟模型 1400。如果 apply 参数具有值 FALSE，意味着在当前阶段配置锁存器值不能应用于模拟模型 1400，那么，该过程直接转到方块 1844。

但是，如果像 TRUE 的 apply 参数值所指的那样，在当前阶段配置锁存器值可以应用于模拟模型 1400，end_phase() API 例程转到方块 1842。在方块 1842 上，end_phase() API 利用锁存器指针阵列 1210 检查配置数据库 1404 中的每个锁存器数据结构 1204。对于锁存器设置字段 1248 具有值 TRUE 的每个锁存器数据结构 1204，end_batch() API 例程发出对模拟器 1410 的 PUTFAC() API 1414 的调用，以便用包含在锁存器值字段 1246 中的锁存器值更新模拟模型 1400。另外，如方块 1844 所示，end_phase() API 在锁存器设置字段 1248 和设置历史字段 1249 之间进行逻辑 OR（“或”）运算，将结果存储在设置历史字段 1249 中。这样，每个设置历史字段 1249 保存是否在成批模式过程的任何阶段都设置了相应配置锁存器的指示符。

在方块 1844 之后，end_batch() API 转到方块 1846，方块 1846 描绘了 end_batch() API 例程重新设置 DIDS 1202 中的所有实例设置字段 1239 和所有锁存器设置字段 1248，为下一个阶段（若有的话）作好准备。此后，在方块 1848 上终止 end_phase() API 例程。

总之，end_phase() API 例程按照 apply 参数将 Dial 默认值应用于与限制 phase 和 instance_qualifer 匹配的配置数据库 1404，然后，可选地将所得配置锁存器值应用于模拟模型 1400。最后，end_phase() API 例程利用设置历史字段 1249 跟踪已经设置了哪些锁存器数据结构 1204，和重新设置各种各样设置字段，为下一个阶段（若有的话）作好准备。

到此为止，仅仅参照在 HDL 文件 800 或配置说明文件 802 中指定的设计者供应阶段信息描述了默认值。对于许多模拟模型 1400 来说，设计者只限制了模拟模型 1400 和相应硬件实现的引导序列的知识，因此，限制了对适当地初始化模拟模型 1400 或相应硬件实现所需的默认值的阶段调整的了解。于是，将指定支配 Dial 默认值的应用的阶段信息的能力提供给下游用户，譬如，模拟用户、实验室用户或推广支持人员是人们所希望的。

如图 18C 所示，在一个实施例中，允许用户利用程序 1860 供应和/修改存储在配置数据库 1404 或相应硬件配置数据库（下面讨论）的阶段 ID 字段 1227 中的阶段 ID。程序 1860 包括一组数据库操纵 API 例程 1862，当用适当的参数调用时，数据库操纵 API 例程 1862 允许用户读取配置数据库 1404（或相应硬件配置数据库）内的阶段 ID 或将阶段 ID 写入配置数据库 1404（或相应硬件配置数据库）内。

再次参照图 14，配置 API 1406 最好进一步包括 find_unset_latch() API，在配置数据库 1404 中的 Dial 或 Dial 组实例的成批模式设置之后，find_unset_latch() API 通过引用锁存器指针阵列 1210 审查配置数据库 1204 中的所有锁存器数据结构 1204，以便检测还没有通过明确的或默认的设置配置的配置锁存器（即，设置历史字段 1249 被设置成 FALSE 的那些）。对于每个这样的未设置配置锁存器，find_unset_latch() API 最好返回来自相应锁存器数据结构 1204 中的锁存器名字段 1244 的配置锁存器的完全限定实例名、和控制未设置锁存器的顶层 Dial 实例的完全限定示例标识符。因此，find_unset_latch() API 为用户提供了核实为模拟运行适当地配置要求明确或默认设置的所有 Dial 和锁存器实例的自动机制。

配置 API 1406 最好进一步包括 check_model() API，当被调用时，check_model() API 利用顶层指针阵列 1206，通过参考适当的映射表 1224 核实模拟模型 1400 中的每个顶层 CDial 和 LDial 实例被设置成它的合法值之一。设置成非法值的任何顶层 LDial 或 CDial 都被 check_model() API 返回。

本发明引入的 Dial 和 Dial 组原语不仅可以应用于配置如上所述的数字设计的模拟模型，而且可以应用于配置用于实验室测试和定制使用的数字设计的配件实现。按照本发明的一个重要方面，数字设计的硬件实现是参照硬件配置数据库配置的，与上面讨论的配置数据库 814 和 1404 一样，硬件配置数据库来源于设计者编码的配置说明语句。这样，配置方法从设计、经过模

拟和实验室测试，到数字设计的商业推广存在连续性。

现在参照图 19，图 19 例示了按照本发明实施例，测试和调试一个或多个数字设计的硬件实现的实验室测试系统的高级方块图。如图所示，实验室测试系统 1900 包括打算用于商业销售和推广的数据处理系统 1902。为了实验室测试和调试，数据处理系统 1902 通过测试接口 1903 与工作站计算机 1904 耦合，工作站计算机 1904 通过测试接口 1903 与数据处理系统 1902 通信，以便将数据处理系统 1902 的各种各样部件配置成执行合适的操作。当作商业推广时，数据处理系统 1902 包括所示的部件，但通常不通过测试接口 1903 与工作站计算机 1904 耦合。

数据处理系统 1902 可以是，例如，像图 1 的数据处理系统 6 那样的微处理器计算机系统。这样，数据处理系统 1902 包括代表数据处理系统的各种各样处理单元、控制器、桥接器和其它部件的多个集成电路芯片 1910。作为典型商用数据处理系统，数据处理系统 1902 可以包含像集成电路芯片 1910a 那样的一些集成电路芯片的多个实例、和像集成电路芯片 1910n 那样的其它集成电路芯片的单个实例。

除了它们各自的功能逻辑单元之外，正如下面参照图 20 详细讨论的那样，集成电路芯片 1910 的每一个含有利用多条扫描链支持集成电路芯片的外部配置的各自测试端口控制器 1912。为了支持这样的外部配置，每个测试端口控制器 1912 通过测试访问端口（TAP）1914 与数据处理系统 1902 内的服务处理器 1920 耦合。

服务处理器 1920 是用于，例如，在通电的时候或响应重新引导，初始化和配置数据处理系统 1902 的通用或专用计算机系统。服务处理器 1920 执行软件指令的至少一个处理单元 1922a、为软件和数据提供非易失性存储的闪速只读存储器（ROM）1924、使服务处理器 1920 与测试端口控制器 1912 交接的 I/O 接口 1926a、和缓存指令和数据供处理单元 1922a 访问的易失性存储器 1928a。

存储在闪速 ROM 1924 中的软件和数据当中有系统固件 1930a。系统固件 1930a 由服务处理器 1920 的处理单元 1922a 在通电的时候执行，以便对集成电路芯片 1910 顺序加电，执行各种各样初始化过程和测试，使集成电路芯片 1910 之间的通信同步，和初始化功能时钟的操作。系统固件 1930a 通过经由测试访问端口 1914 的通信，控制集成电路芯片 1910 的启动行为。

除了系统固件 1930a 之外，闪速 ROM 1924 存储描述集成电路芯片 1910 的硬件（HW）配置 API 1934a 和 HW 配置数据库 1932a。如下所述，在商业推广期间，处理单元 1922a 调用各种各样 HW 配置 API 1934a 来访问 HW 配置数据库 1932a，以便通过 I/O 接口 1926a 和 TAP 1914 适当地配置集成电路 1910。

可以作为，例如，像图 1 的数据处理系统 6 那样的微处理器计算机系统实现的工作站计算机 1904 包括在功能上与服务处理器 1920 的那些相似的许多部件。于是，用相似的标号表示处理单元 1922b、易失性存储器 1928b、I/O 接口 1926b、和驻留在非易失性存储器 1940（例如，盘存储器）中的系统固件 1936b、HW 配置数据库 1932b 和 HW 配置 API 1934b。本领域的普通技术人员应该认识到，由于驻留在非易失性存储器 1940 中的系统固件 1936b、HW 配置数据库 1932b 和 HW 配置 API 1934b 被专门设计成在实验室测试和调试的背景下初始化和配置数据处理系统 1902，它们比闪速 ROM 1924 内的相应软件和数据可能具有较小、较大或简单的不同特征集和能力。

在实验室测试和调试期间，工作站计算机 1904 承担服务处理器 1920 的大多数功能。例如，工作站计算机 1904 通过执行系统固件 1930b 和各种各样 HW 配置 API 1934b 以便生成各种各样 I/O 命令来初始化和配置数据处理系统 1902。然后，通过测试接口 1903 和 I/O 接口 1926a 和 1926b 将这些 I/O 命令传送给数据处理系统 1902。以禁止其大多数自然功能的“旁路”模式在服务处理器 1920 中执行的系统固件 1930a 通过经由测试访问端口 1914 将这些外部 I/O 命令发送给集成电路芯片 1910，以便初始化和配置集成电路芯片 1910 来响应这些外部 I/O 命令。

现在参照图 20，图 20 例示了按照本发明的示范性集成电路芯片 1910 的更详细方块图。如上所述，集成电路芯片 1910 包括测试端口控制器 2000，测试端口控制器 2000 支持与图 19 的服务处理器 1920 的 I/O 接口 1926 的外部通信、和集成电路芯片 1910 的各种各样内部功能的控制，包括功能时钟 2002 和扫描时钟 2010 的操作。集成电路芯片 1910 进一步包括功能逻辑单元（未明确示出），功能逻辑单元包括响应功能时钟 2002 的时钟脉冲，完成计划让集成电路做的“工作”，例如，处理软件指令的数字集成电路。许多功能锁存器 2004 分布在整个功能逻辑单元上，在功能逻辑单元的正常功能操作期间（即，当功能时钟 2002 为功能逻辑单元计时时），功能锁存器 2004 保存代表功能逻辑单元的动态和数据和/或指令的位。这些功能锁存器 2004 包括保

存用于配置所需配置中的功能逻辑单元的模式和配置位的那些功能锁存器。

如图所示，数组功能锁存器 2004 互连在一起，形成多条测试扫描链 2006 和多条 SCOM（扫描通信）链 2008。尽管为了简洁起见未示出，但一些功能锁存器 2004 实际上是测试扫描链 2006 和 SCOM 链 2008 的成员。测试扫描链 2006 用于响应扫描时钟 2010 的脉冲扫描到功能锁存器 2004 的位，和 SCOM 链 2008 用于响应功能时钟 2002 的脉冲扫描到功能锁存器 2004 的位。功能时钟 2002 和扫描时钟 2010 两者不同时输出脉冲，以防止装入功能锁存器 2004 中的值之间的冲突。

正如所描绘的那样，测试扫描链 2006 中的每个功能锁存器 2004 包括至少两个数据输入端，即，扫描输入端（scanin）和功能输入端（D_{in}）、和两个时钟输入端，即，扫描时钟输入端（sclk）和功能时钟输入端（fclk）。每个功能锁存器 2004 进一步包括至少两个数据输出端，即，扫描输出端（scanout）和功能输出端（D_{out}）。为了形成测试扫描链 2006，第一功能锁存器 2004 的扫描输入端和最后功能锁存器 2004 的扫描输出端与测试端口控制器 2000 耦合，和测试扫描链 2006 中的每个功能锁存器 2004（除了最后一个之外）的扫描输出端与下一个功能锁存器 2004 的扫描输入端连接。

每个功能锁存器 2004 锁入出现在它的 scanin 上的数据位和响应在 sclk 上的扫描时钟 2010 的脉冲在 scanout 上锁出它的旧值，和锁入出现在 D_{in} 上的数据位和响应在 fclk 上的功能时钟 2010 的脉冲的接收锁出它的旧值。因此，通过扫描时钟 2010 的重复脉冲生成，形成测试扫描链 2006 的功能锁存器 2004 以“位桶编队（bit-bucket brigade）”方式将数据位传入和传出测试端口控制器 2000，从而使测试端口控制器 2000 可以读写测试扫描链 2006 中的一个或多个功能锁存器 2004。

SCOM 链 2008 用于当功能时钟 2002 有效和扫描时钟 2010 失效时读写功能锁存器 2004。每条 SCOM 链 2008 包括多个依次相连 SCOM 单元 2012，SCOM 单元 2012 的第一个和最后一个与测试端口控制器 2000 连接，使测试端口控制器 2000 可以扫描进入 SCOM 单元 2012 和从 SCOM 单元 2012 出来的数据位。正如所描绘的那样，在示范性实施例中，每个 SCOM 单元 2012 包含形成“SCOM 寄存器”的一部分的功能锁存器 2004，以及形成“影子寄存器”的一部分的影子锁存器 2014。如果所有影子锁存器 2014 像功能锁存器 2004 那样也属于调试扫描链 2006 那就更好了。

如图所示，每个SCOM单元2012中的每个功能锁存器2000与相关多路复用器2020连接，多路复用器2020拥有与相应影子锁存器2014的输出端耦合的扫描输入端（scomin）和通过保持路径与相关功能锁存器2004的数据输出端（D_{out}）耦合的数据输入端（D_{in}）。多路复用器2020响应选择信号sel2，选择出现在数据输入端（D_{in}）和scomin其中之一上的数据位，作为功能锁存器2004的输入。功能锁存器2004响应功能时钟fclk锁存所选数据位。

每个SCOM单元2012中的影子锁存器2014类似地与相关多路复用器2022连接，多路复用器2022拥有与功能锁存器2004的数据输出端（D_{out}）耦合的数据输入端（D_{in}）、通过保持路径与影子锁存器2014的输出端耦合的保持输出端、和扫描输入端（scomin）。在第一SCOM单元2012中，扫描输入端与测试端口控制器2000连接，和在其余SCOM单元2012中，扫描输入端与前一个SCOM单元2012中的影子锁存器2014的输出端连接。每条SCOM链中的最后一个SCOM单元2012的影子寄存器2014的输出端与测试端口控制器2000连接。多路复用器2022响应选择信号sel1，在出现在它的输入端上的数据位当中选择一个，作为相关影子锁存器2014的输入。影子锁存器2014响应功能时钟fclk锁存所选数据位。

影子寄存器链用于从相关SCOM寄存器中读值和将值写入相关SCOM寄存器中。例如，为了设置SCOM寄存器，测试端口控制器2000通过维护选择信号sel1的适当值，扫描经由多路复用器2022的scomin输入端到影子锁存器2014的新值。一旦所有影子锁存器2014都已装载，测试端口控制器2000控制选择输入sel2使功能寄存器2004装载来自影子锁存器2014的值。为了读取来自SCOM寄存器的值，测试端口控制器2000驱动sel1将功能锁存器2004当中的值读入影子锁存器2014，然后，通过维护选择信号sel1的适当值，扫描影子锁存器2014当中的值。

在示范性实施例中，SCOM链2008应用影子锁存器2014来读写功能锁存器2004，以免破坏集成电路芯片1910，甚至数据处理系统1902的适当功能操作。通过在更新任何功能锁存器2004之前装载所有影子锁存器2014，不用在功能时钟2002的多个周期内破坏它们的值，可以一次性更新SOM链2008内的所有功能锁存器2004。应该明白，如图20所示的SCOM链2008的具体实现不是实施本发明所必需的，可以应用其它可替代设计，包括不包括影子锁存器2014的一些设计。

因此，通过将适当值装入功能锁存器 2004 中和通过适当控制功能时钟 2002 和扫描时钟 2010，每个测试端口控制器 2000 可以根据来自服务处理器 1920 和/或工作站计算机 1904 的输入，以所需方式初始化和配置它的集成电路芯片 1910。

为了以如上所述的方式配置硬件功能锁存器 2004，必须生成考虑模拟环境和硬件环境之间的差异的 HW 配置数据库 1932。一般说来，HW 配置数据库 1932 的结构和内容至少反映两个方面与上述用于模拟的配置数据库 814 的主要差异。

第一差异是在硬件中寻址锁存器的方式。具体地说，取代像在模拟中那样，将完全限定示例标识符用于配置锁存器，通过由指定特定测试扫描链 2006 的扫描链（或环）标识符和指示特定测试扫描链 2006 中锁存器位位置的偏移组成的有序对为测试扫描寻址和访问特定集成电路 1910 中的每个硬件功能锁存器 2004。利用由指定特定 SCOM 链 2008 的环标识符和相应影子锁存器 2014 的偏移组成的相似有序对（环标识符，偏移）可以类似地为 SCOM 扫描寻址和访问 SCOM 环 2008 内的功能锁存器 2004。重要的是，特定功能锁存器 2004 的 SCOM 环标识符和偏移不具有与相应测试扫描环标识符和偏移相同的值。事实上，在可替代 SCOM 实现中，可以使用不同 SCOM 硬件，和可以将偏移表达成一个元组：（环 ID，寄存器，偏移）。因此，应该认识到，可以利用多种访问方法寻址和访问功能锁存器 2004，这些访问方法的每一种可以具有它自己的寻址方案，所有这些寻址方案都有可能与应用在模拟中的寻址方案不同。

HW 配置数据库 1932 和应用在模拟中的配置数据库 814 之间的第二重要差异是整个数据库结构。如上所述，配置数据库 814 是可以用于通过分层嵌套设计实体表示任何大小或复杂性的任意选择数字设计的单片数据库。对于模拟的每个不同数字设计，配置编译器 808 生成新的配置数据库 814。尽管这种方法在模拟环境下是令人满意的，但用在模拟中的单片数据库结构不对应于在硬件数字设计中用于访问和设置锁存器的实际物理机制。此外，在实验室环境下最好避免为每次不同硬件置换开发全新的系统固件 1930 和 HW 配置数据库 1932。例如，最好通过重新使用特定 HW 配置数据库 1932 和系统固件 1930 的一些或全部来初始化和配置支持 8 和 32 个处理单元和 1 到 4 个不同存储器控制器之间的服务器产品线中的每个服务器计算机，使开发时间和

成本达到最小。

因此，正如下面详述的那样，HW 配置数据库 1932 最好被构造成每一个对应于出现在硬件数字设计中的特定类型（不是实例）集成电路芯片的较小数据库的联盟。这种数据库结构支持从相同“积木”按芯片型数据库中为任何所需大小和复杂性的硬件系统构造出 HW 配置数据库 1932。此外，这种数据库结构反映了系统固件按芯片访问硬件锁存器的事实。

现在参照图 21，图 21 描绘了转换每个集成电路芯片的模拟配置数据库 814 以获取用于构造适用于实验室测试和调试和商业推广的 HW 配置数据库 1932 的芯片 HW 数据库的示范性的高级流程图。所例示的过程可以通过在图 1 的数据处理系统 6 上执行软件来实现。

该过程从执行扫描链检测工具 2100 开始。扫描链检测工具 2100 处理诸如数据处理系统 1902 之类的目标硬件系统内的每个集成电路芯片 1910 模拟模型 1400，为集成电路芯片 1910 内的锁存器生成与每条（种）功能锁存器访问路径/方法相对应的各自输出文件。例如，在示范性实施例中，扫描链检测工具 2100 生成与测试扫描相对应的测试扫描定义文件 2104 和与 SCOM 扫描相对应的 SCOM 定义文件 2102。这些文件 2102、2104 的每一个为模拟模型 1400 内的锁存器提供了锁存器的扫描环标识符和偏移（或相关访问方法的其它硬件地址）与它用于模拟的完全限定锁存器实例名之间的对应关系。

然后，数据库转换工具 2106 处理测试扫描定义文件 2104 和 SCOM 定义文件 2102 和集成电路芯片的模拟配置数据库 814，生成可以作为积木用于为任意系统大小和部件列表的硬件系统获取 HW 配置数据库 1932 的芯片 HW 数据库 2108。

现在参照图 22A，图 22A 例示了数据库转换工具参照测试扫描定义文件 2104 和 SCOM 定义文件 2102，从集成电路芯片的相应模拟配置数据库 814 中生成芯片 HW 数据库 2108 的示范性过程的高级逻辑流程图。如图所示，该过程从方块 2200 开始，然后转到方块 2201，方块 2201 例示了以上面参照图 13 讨论的方式将模拟配置数据库 814 从非易失性数据存储器装入易失性存储器中和扩充它的字段，以获得扩展配置数据库 1404。测试扫描定义文件 2104 和 SCOM 定义文件 2102 也被装入中易失性存储器中。

接着，在方块 2202 上，确定通过锁存器指针阵列 1210 引用的所有锁存器数据结构 1204 是否都得到处理。如果是，在方块 2204 上终止该过程。但

是，如果所有锁存器数据结构 1204 还没有都得到处理，该过程从方块 2202 转到方块 2206，方块 2206 例示了处理锁存器指针阵列 1210 中下一个锁存器指针 1254 所指的锁存器数据结构 1204 的选择。接着，在方块 2208 上，利用父指针 1242 形成与正在考虑之中的锁存器数据结构 1204 相对应的锁存器的完全限定锁存器名，以便访问控制锁存器和将其内容附在锁存器名字段 1244 的内容上的 Dial 实例的实例名字段 1234 的内容。

然后，正如在方块 2210 上所描绘的那样，从测试扫描定义文件 2104 中搜索这个完全限定锁存器名。如果在测试扫描定义文件 2104 中没有找到完全限定锁存器名，在方块 2212 上标上错误，因为在示范性实施例中，所有可配置锁存器必须是可扫描的。否则，数据库转换工具 2106 在方块 2214 上调用 API 例程 add_access_method(method_id,method_name)，扩充锁存器数据结构 1204 以形成新的锁存器数据结构 2230。API 调用的 method_id 参数标识特定访问方法（例如，用字符串或整数），和 method_name 参数指定相关访问方法硬件访问与新锁存器数据结构 2230 相对应的锁存器所使用的“名称”。正如在图 22B 中所例示的那样，新锁存器数据结构 2230 是在方块 2214 上通过将指定这种访问方法的方法标识符（按照惯例，它是“0”）的方法 ID 字段 2232a 和为锁存器指定测试扫描环标识符和偏移值的方法名字段 2234a 加入锁存器数据结构 1204 中创建的。

该过程从方块 2214 转到方块 2216，方块 2216 代表利用下一种访问方法的定义文件，在这种情况下，SCOM 定义文件 2102，重复在方块 2210 上执行的对完全限定锁存器实例名的搜索。如果在 SCOM 定义文件 2102 内发现与完全限定锁存器实例名不匹配，那么，不记录错误，因为并非所有锁存器都属于 SCOM 链，并且，该过程简单地转到如下所述的方块 2220。另一方面，如果发现匹配，在方块 2218 上再次调用 add_access_method() API 例程，扩充带有指定这种访问方法的方法标识符的方法 ID 字段 2232a 和为锁存器指定 SCOM 扫描环标识符和偏移值的方法名字段 2234n 的锁存器数据结构 2230。

最后，在方块 2220 上，调用 API 例程 delete_latch_name()，从锁存器数据结构 2230 中删除锁存器名字段 1244。因为环标识符和偏移对唯一地标识集成电路芯片 1910 内的任何锁存器，所以不再需要锁存器名字段 1244。然后，该过程返回到已经描述过的方块 2202。

因此，图 22A 的方法将每个集成电路芯片的模拟配置数据库变更成包括

指示适用于每个硬件功能锁存器的访问方法和对于每种适用访问方法锁存器的“方法名”（即，标识符）的信息。尽管所例示的过程描绘了将模拟配置数据库修改成支持两种特定访问方法，但所例示的过程可以应用于管理任何数量或类型的访问方法。

一旦以例示在图 21 和 22A 中的方式处理了用于系统中的每个集成电路的所有模拟配置数据库，接着可以组合所得芯片硬件数据库 2108，形成例示在图 19 中的 HW 配置数据库 1932。在优选实施例中，通过创建芯片指针数据结构 2320（图 23B）从芯片 HW 数据库 2108 中构建 HW 配置数据库 1932，芯片指针数据结构 2320 包含引用数据处理系统 1902 中的每种类型芯片的芯片 HW 数据库 2108 的各自芯片数据库指针 2322。例如，如果数据处理系统 1902 包括 32 个相同集成电路处理器芯片，芯片指针数据结构 2320 将只包含一个芯片数据库指针 2322（除了与其它类型的集成电路芯片相对应的其它芯片数据库指针 2322 之外），这个芯片数据库指针 2322 指向描述通过 32 个集成电路处理器芯片具体化的数字设计的单个芯片 HW 数据库 2108。然后，如图 19 所示，将这个 HW 配置数据库 1932 存储在诸如非易失性存储器 1940 或闪速 ROM 1924 之类的非易失性存储器中。

为了利用 HW 配置数据库 1932 配置硬件数字设计，首先按照在图 23A 中描绘的示范性过程将 HW 配置数据库 1932 从非易失性存储器装入易失性存储器中。例如，在实验室环境下，通过让处理单元 1922b 执行系统固件 1930b，工作站计算机 1904 可以完成如图 23A 所示的过程。类似地，当在商业上推广数据处理系统 1902 时，服务处理器 1920 根据图 23A 的过程执行系统固件 1930a，将 HW 配置数据库 1932a 从闪速 ROM 1924 装入易失性存储器 1928a 中。

如图所示，图 23A 的过程从方块 2300 开始，然后转到方块 2302，方块 2302 例示了出现在诸如数据处理系统 1902 之类的目标数据处理系统中的集成电路芯片的类型和每种类型的个数的确定。在示范性实施例中，例示在方块 2302 上的确定是由系统固件 1930 作出的，系统固件 1930 查阅一组所谓的极重要产品数据（VPD），以确定数据处理系统 1902 代表数千种可能机器配置的哪一个。

然后，该过程转到方块 2306-2310，方块 2306-2310 集体形成挪动芯片指针数据结构 2320 以处理构成数据处理系统 1902 的集成电路芯片的芯片 HW 数据库 2108 的循环。首先，在方块 2306 上，确定数据处理系统 1902 内的每

种类型集成电路芯片的芯片 HW 数据库 2108 是否都得到处理。如果是，HW 配置数据库 1932 到易失性存储器的装入已经完成，在方块 2312 上终止该过程。但是，如果与通过 VPD 标识的每种类型集成电路芯片相对应的芯片 HW 数据库 2108 还没有处理完，在方块 2308 上将下一个芯片 HW 数据库 2108 装入工作站 1904 的易失性存储器 1928 中加以处理。

如图 23B 所示，图 23B 描绘了 HW 配置数据库 1932 的存储器内图形，芯片 HW 数据库 2108 的装入创建了如上所述的存储器内数据结构，譬如，Dial 指针阵列 1208、锁存器指针阵列 1210、和每个 DDDS 1200 内的实例指针阵列 1226（参见图 12）。另外，在每个锁存器数据结构 2230 内创建了锁存器值字段 2324、锁存器设置字段 2326、和设置历史字段 2325，和在每个 DIDS 1202 内创建了实例设置阵列 2328。这三个字段的每一个是作为每个项目对应于与当前芯片 HW 数据库 2108 相对应的集成电路芯片 1910 的特定实例的阵列实现的。最后，创建空芯片映射表 2325。

接着，在方块 2310 上，将各自项目加入与当前芯片 HW 数据库 2108 相对应的集成电路芯片类型的每个实例的芯片映射表 2325 中。这个步骤最好由系统固件 1930 通过调用 HW 配置 API 1934 来完成，HW 配置 API 1934 访问 VPD 以确定与当前芯片 HW 数据库 2108 相对应的集成电路芯片类型的多少个实例包含在当前硬件数字设计中。按照惯例，芯片映射表 2325 内项目的顺序对应于实例设置字段 2328、锁存器值字段 2324 和锁存器设置字段 2326 中阵列项目的顺序。

如图 23B 所示，芯片映射表 2325 内的每个项目与两个固件供应值相联系：（1）芯片实例名，它是像在数据处理系统 1902 的模拟模型中标识代表集成电路芯片的设计实体的字符串那样的字符串（例如，a.b.c.d）；和（2）芯片 ID，它指定服务处理器 1920 与那个集成电路芯片实例的测试访问端口 1914 的标识符。因此，现在通过元组（芯片 ID，扫描环，偏移）可以容易地寻址数据处理系统 1902 中的任何锁存器，元组（芯片 ID，扫描环，偏移）通过芯片映射表 2325 与 HW 配置 API 19340 应用的完全限定锁存器名的芯片标识部分相联系。此后，该过程返回到已经描述过的方块 2306。

因此，在图 23A 中描绘的过程允许单个 HW 配置数据库 1932 用于为任意大小或配置的数据处理系统构建存储器内 HW 配置数据库，无需为每种可能系统大小和配置开发和存储分立单片配置数据库。

借助于装入易失性存储器 1928 中的 HW 配置数据库 1932，服务处理器 1920 的处理单元 1922a 或工作站计算机 1904 的处理单元 1922b 接着可以执行系统固件 1930，以便调用 HW 配置 API 1934 读取或设置数据处理系统 1902 的一个或多个集成电路芯片 1910 的配置。与在模拟中一样，HW 配置 API 1934 最好包括分立 API 例程，以便以交互或成批模式读取 Dial 和 Dial 组。还像模拟那样，通过系统固件 1930 的 API 调用为要设置或读取的每个 Dial 或 Dial 组实例指定实例限定符（例如，a.b.c.d 或 a.b.c.[X]）和拨号器名限定符（例如，Entity.dialname）。

由于多种访问方法可以用于设置或读取 Dial 或 Dial 组，设置或读取 Dial 或 Dial 组实例的 API 调用最好包括附加参数 access_method。在优选实施例中，access_method 参数可以取指示测试扫描的值 SCAN、指示 SCOM 扫描的 SCOM 和指示 HW 配置 API 1934 选择访问方法的 AUTO。响应 access_method 参数的 AUTO 值，HW 配置 API 1934 根据 API 调用针对的锁存器数据结构 2230 中的方法 ID 2232 所指的受支持访问方法和根据功能时钟 2002 和扫描时钟 2010 的哪一个正在工作选择访问方法。如上所述，当功能时钟 2002 正在工作时，只有 SCOM 扫描可用，和当扫描时钟 2010 正在工作时，只有测试扫描可用。

在 HW 配置 API 1934 可以设置或读取 Dial 或 Dial 组实例之前，HW 配置 API 1934 首先必须确定在 API 调用中指定的实例限定符和拨号器名限定符标识了哪些 Dial 或 Dial 组实例。现在参照图 24，图 24 描绘了按照本发明，HW 配置 API 1934 在 HW 配置数据库 1932 中定位特定 Dial 或 Dial 组实例的示范性过程的高级逻辑流程图。所例示的过程与在图 15 中描绘的和如上所述的过程类似。

如图所示，正如上面所讨论的那样，响应 HW 配置 API 1934 对含有一个或多个 Dial 或 Dial 组实例的实例限定符和拨号器名限定符作为变元的来自固件 1930 的 API 调用的接收，该过程从方块 2400 开始。响应 API 调用，配置 API 1934 在芯片指针阵列 2320 上输入 HW 配置数据库 1932，和正如在方块 2402 上所描绘的那样，进入处理芯片数据库指针 2322 的循环，直到在特定芯片 HW 数据库 2108 中定位一个或多个匹配 Dial 实例为止或直到所有芯片数据库指针 2322 都得到处理为止。响应在方块 2402 上所有芯片数据库指针 2322 都得到处理，但没有定位任何匹配 Dial 实例的确定，该过程在方块 2403

上以错误告终。但是，如果所有芯片数据库指针2322还没有全部处理，那么，正如在方块2406上所描绘的那样，从芯片指针数据结构2320中选择下一个芯片数据库指针2322加以处理。所选的芯片数据库指针2322用于定位相关芯片HW数据库2108。

在方块2406之后，该过程转到方块2408和随后的方块，方块2408和随后的方块代表处理当前芯片HW数据库2108的Dial指针阵列1208中的每个Dial指针1252，直到定位与API调用匹配的特定Dial为止，或直到所有Dial指针1252（图12）都得到处理，但没有找到任何匹配Dial实例的循环。响应在方块2208上所有Dial指针1252都得到处理，但没有定位任何匹配Dial实体的确定，该过程从方块2408返回到方块2402，以便处理芯片指针阵列2320中的下一个芯片数据库指针2322(即，处理下一个芯片HW数据库2108)。另一方面，如果在2408上确定并非Dial指针阵列1208内的所有Dial指针1252都得到处理，该过程转到方块2410，方块2410例示了从Dial指针阵列1208中选择下一个Dial指针1252加以处理。

接着，在方块2412上确定当前Dial指针1252引用的DDDS 1200是否拥有与指定拨号器名限定符完全匹配的名称字段1222。对于名称字段1222，可以有两种实现。第一，可以禁止重新使用Dial名，以便每个Dial名不仅在它的自己的整个集成电路芯片内，而且在整个系统(例如，数据处理系统1902)内都是唯一的。第二，限制较少的方法要求每个Dial名只在它的集成电路芯片1910内是唯一的，允许在不同集成电路中多次使用Dial名。为了支持第二种方法，名称字段1222采取“chiptype.Dial.name”的形式，其中，“chiptype”是标识集成电路芯片1910的类型的唯一字符串，从而分清应用于在不同集成电路芯片1910中例示的Dial实体的相同Dial名。

响应在方块2412上名称字段1222不与指定拨号器名限定符的确定，该过程返回到方块2408，像上述那样，对下一个Dial指针1252（若有的话）加以处理。但是，如果发现匹配，该过程接着进入包括方块2420-2434的处理循环，在这个处理循环中利用匹配Dial实体的DDDS 1200的实例指针阵列1226中的实例指针1228检查各自DIDS 1202所代表的Dial实例是否与API调用的实例限定符匹配。在这个处理循环中，首先在方块2420上确定当前DDDS 1200内的所有实例指针1228是否都得到处理。如果是，在方块2434上进一步确定是否找到与当前DDDS 1200相对应的Dial实体的至少一个匹配

实例。作为这个确定是因为 HW 配置数据库 1932 的构造保证了至少一个芯片 HW 数据库 2108 中的至多一个匹配 Dial（不是 Dial 实例）与在 API 调用中指定的实例限定符和拨号器名字符匹配。因此，如果对于特定 Dial 实体找到了匹配实例，就无需进一步搜索 Dial 实体或芯片 HW 数据库 2108 了。于是，如果对于与当前 DDDS 1200 相对应的 Dial 实例，已经找到至少一个匹配 Dial 实例，该过程从方块 2434 转到方块 2438，然后终止。但是，如果在方块 2434 上确定没有找到匹配，该过程穿过页面连接符 A，并且在方块 2403 上以错误告终。

返回到方块 2420，响应当前 DDDS 1200 的所有实例指针 1228 还没有得到处理的确定，该过程转到方块 2422，方块 2422 例示了选择下一个实例指针 1228 和它的相关 DIDS 1202 加以处理。然后，在方块 2424 上确定是否通过处理芯片映射表 2326 中的每个项目，已经与与当前芯片 HW 数据库 2108 相对应的每个集成电路芯片 1910 中的 Dial 实例相关地处理了 DIDS 1202。如果是，该过程转到如下所述的方块 2436。如果芯片映射表 2325 中的所有项目的处理还没有完成，该过程转到方块 2426。

方块 2426 描绘了通过将芯片映射表 2325 的下一个项目中的芯片实例名预先附在当前 DIDS 1202 的实例名字段 1234 上，形成要与在 API 调用中指定的实例限定符匹配的下一个完全限定 Dial 实例名。然后，在方块 2430 上将这个完全限定 Dial 实例名实例与限定符相比较。如果它们不匹配，该过程返回到已经描述过的方块 2424。如果它们匹配，在方块 2432 上创建临时结果指针和相关的芯片矢量（如果它们还不存在的话）。临时结果指针指向当前 DIDS 1202，以便将相应 Dial 实例识别成与在访问请求中指定的实例限定符匹配。还将一个项目放入相关芯片矢量中，以指示这个匹配 Dial 实例所处的特定集成电路芯片实例 1910。在示范性实施例中，与芯片映射表 2325 中存在项目一样，芯片矢量可以简单地包括个数相同的位，位值“1”表示相应集成电路芯片实例 1910 包含匹配 Dial 实例。在方块 2432 之后，该过程返回到方块 2424。

对于芯片映射表 2325 中的每个项目重复方块 2424-2432 所代表的处理循环。在所有项目都得到处理之后，该过程从方块 2424 转到方块 2436，方块 2436 描绘了是否利用无括号语法指定拨号器名限定符，如果是，对于当前 DIDS 1202 所代表的 Dial 实例当中的指定拨号器名限定符是否找到匹配的确

定。如果确定是否定的，有可能存在与另一个 DIDS 1202 相联系的另外匹配 Dial 实例。于是，该过程返回到方块 2420，处理当前 DDDS 1200 的下一个实例指针 1228，但是，如果在方块 2436 上的确定是肯定的，那么知道用临时结果指针和相关芯片矢量已经定位和识别了所有匹配 Dial 实例。因此，在方块 2438 上终止该过程。

在通过如图 24 所示的过程已经确定了通过实例限定符和拨号器名限定符指定的 Dial 或 Dial 组实例之后，以与上面参照图 16A（以交互方式读取 Dial 实例）、16B（以交互方式读取 Dial 组实例）、17A（以交互方式设置 Dial 实例）、17B（以交互方式设置 Dial 组实例）、和 18A-18B（以成批方式设置 Dial 实例或 Dial 组实例）所述的方式几乎相同的方式设置或读取 Dial 或 Dial 组实例。但是，对于单个芯片 HW 数据库 2108 用于表示可能多个集成电路芯片 1910 和对于多种不同访问方法可用于访问集成电路芯片 1910，要求考虑少数几个差异。下面详述这些差异。

当读取 Dial 实例或 Dial 组实例时，正如参照图 16A 的方块 1624 所述的那样，在配置数据库中通过沿着 Dial 树向上传播锁存器值来核定锁存器值。相反，当设置 Dial 实例或 Dial 组实例时，正如参照图 17A 的方块 1714 所述的那样，在配置数据库中沿着 Dial 树向下传播锁存器值。在模拟中，“向下”到任何一个锁存器数据结构 1204 或从任何一个锁存器数据结构 1204 “向上”每次只传播一个锁存器值。但是，由于 HW 配置数据库 1932 代表带有单个芯片 HW 数据库 2108 的相同类型的多个集成电路芯片 1910，参照代表多个物理集成电路芯片 1910 的芯片 HW 数据库 2108 读取或设置 Dial 或 Dial 组需要并行地沿着 Dial 树向上或向下传播值集的多个元素，其中，值集的每个元素是通过在图 24 中构成的临时结果指针和芯片矢量识别的特定芯片实例的值。

类似地，在模拟中，配置数据库 1404 内的实例设置字段 1239、锁存器值字段 1246、锁存器设置字段 1248、和设置历史字段 1249 的每一个只包含单个值。相反，配置数据库 1932 内的相应实例设置字段 2328、锁存器值字段 2324、锁存器设置字段 2326、和设置历史字段 2325 被当作对于特定集成电路芯片 1910 每个元素对应于各个 Dial 和锁存器实例的阵列来实现。于是，当设置 Dial、Dial 组和锁存器实例时，按照在图 24 中构成的临时结果指针和芯片矢量更新与设置实例相对应的实例设置字段 2328、锁存器值字段 2324、锁存器设置字段 2326、和设置历史字段 2325 内的元素。

由于 HW 配置数据库 1932 的实验室或商业使用需要利用多种可能访问方法访问物理硬件（即，集成电路芯片 1910），在优选实施例中注意到与模拟环境不同的另外三种差异。第一，如果 HW 配置 API 1934 确定包含在 API 调用内的 access_method 参数所指的访问方法不适合于通过图 24 的过程获得的临时结果指针和芯片矢量所识别的任何 Dial 实例，在 API 调用中请求的设置或读取操作最好失败（即，不进行）。如上所述，可以设置或访问每个锁存器的访问方法通过每个锁存器数据结构 2230 的方法 ID 字段 2232 指出。

第二，如果 HW 配置 API 1934 确定 API 调用针对的每个集成电路芯片 1910 内的功能时钟 2002 和扫描时钟 2010 处在适合于包含在 API 调用内的 access_method 参数的状态下，在 API 调用中请求的设置或读取操作最好取得成功。也就是说，如果 access_method 参数具有值 SCAN，必须禁用功能时钟 2002，和必须启用扫描时钟 2010。相反，如果 access_method 参数具有值 SCOM，必须启用功能时钟 2002，和必须禁用扫描时钟 2010。如果 access_method 参数具有值 AUTO，包含 API 调用针对的锁存器的每个集成电路芯片 1910 的功能时钟 2002 和扫描时钟 2010 必须处在允许每个这样锁存器的至少一种访问方法得到应用的状态下。

第三，用于读取和设置硬件锁存器的 HW 配置 API 1934，read_latch() 和 write_latch() 最好在易失性存储器中实现影子扫描链缓冲器，和如有可能，访问这样的扫描链缓冲器来代替扫描集成电路芯片 1910 中的扫描链，使对集成电路芯片 1910 的扫描访问达到最少。例如，在知道易失性存储器 1928 中的锁存器值是当前值的情况下，与应用在模拟中的 GETFAC() API 1412 相对应的 read_latch() HW 配置 API 1934 最好从易失性存储器 1928 中的相应影子扫描链缓冲器中获取锁存器值。另外，通过与应用在模拟中的 PUTFAC() API 1414 相对应的 write_latch() API 对锁存器值的多次更新最好缓存在易失性存储器 1928 中的相应影子扫描链缓冲器中。这样，只扫描特定扫描链一次就可以作出对集成电路芯片 1910 的特定扫描链的多次写入。

HW 配置 API 1934 最好进一步包括与可用在模拟中的 check_model() API 相似的 check_chip() API。当被调用时，check_chip() API 利用指定芯片 HW 数据库 2108 内的顶层指针阵列 1206 来核实芯片 HW 数据库 2108 内的每个顶层 CDial 和 LDial 实例是否被设置成它的合法值之一。具体地说，check_chip() API 通过参照它的映射表 1224 和在它的 Dial 树中任何较低层

Dial 实例的映射表 1224，沿着每个顶层 CDial 和 LDial 实例的 Dial 树向上传播底层硬件锁存器值。被设置成非法值的任何顶层 CDial 或 LDial 实例通过 check_chip() API 返回。

再次参照图 19，在数据处理系统 1902 的许多商业实施例中，服务处理器 1920 内的非易失性存储器（例如，闪速 ROM）1924 的存储容量显著地小于用于存储系统固件 1930b 和 HW 配置数据库 1932b 的工作站计算机 1904 的非易失性存储器（例如，硬盘存储器）1940 的存储容量。于是，通常希望或有必要缩小在实验室硬件测试环境下开发的系统固件 1930b 和 HW 配置数据库 1932b 的大小，以获得在数据处理系统 1902 的闪速 ROM 1924（或其它非易失性存储器）内用于商业推广的系统固件 1930a 和 HW 配置数据库 1932a。

于是，现在参照图 25，图 25 例示了通过删除不必要信息可以压缩在系统固件 1930 的实验室开发和测试期间开发的每个芯片 HW 数据库 2108，以便获得适用于商业推广的 HW 配置数据库 1932a 的示范性过程的高级逻辑流程图。该过程从生成 Dial 用法信息 2500 开始，Dial 用法信息 2500 指示已经设置和/或读取了特定类型集成电路芯片 1910 内的 Dial 实例和设置给 Dial 实例的值。

设置或读取 Dial 实例和设置给 Dial 实例的值的确定可以以本领域的普通技术人员都已知的许多种方式完成。例如，可以人工地检查系统固件 1930 以生成 Dial 用法信息 2500。可替代地，可以在许多可能机器配置中执行系统固件 1930，这些可能机器配置涵盖可以设置给正被考虑的那种类型集成电路芯片 1910 中的 Dial 实例的所有设置。然后，可以将设置和读取的 Dial 实例和设置给 Dial 实例的值记录成 Dial 用法信息 2500。

在优选实施例中，对于 IDial 实例记录在 Dial 用法信息 2500 中的全部东西是是否设置或读取 IDial 实例。没有记录 IDial 实例值，这是因为，为了生成 Dial 用法信息 2500，假设如果设置了 IDial 实例，可以使用它的所有可能值。但是，开发者知道只有特定 IDial 被设置成单一值。为了允许从 HW 配置数据库 1932a 中删除这些 IDial，开发者可以在优先文件 2502 中可选地指定这些 IDial 和它们的相关值。与是否设置或读取 IDial 实例无关，优先文件 2502 也可以包含开发者希望明确保存在 HW 配置数据库 1932a 内的 Dial 实例（若有的话）的列表。

因此，对于每个芯片 HW 数据库 2108，最好获取集体包含至少如下信息

的 Dial 用法信息 2500 和优先文件 2502:

1)任何配置中设置在任何集成电路芯片实例内的所有顶层非 IDial 实例的列表和任何配置中的任何集成电路芯片实例内设置成任何值的任何顶层 IDial 实例的列表;

2) 设置的每个非 Dial 实例的所有值的列表;

3) 设置成单一值的 IDial 的分立表; 和

4) 读取的所有 Dial 实例的列表。

正如在图 25 中进一步例示的那样, 接着, 软件压缩工具 2504 (例如, 由工作站计算机 1904 执行)利用这种信息从相关芯片 HW 数据库 2108 中删除不必要信息。压缩工具 2504 生成两种输出: (1) 形成 HW 配置数据库 1932a 的一部分的压缩芯片 HW 数据库 2506, 和(2)用于开发在执行系统固件 1930a 期间对其初始化集成电路芯片 1910 中的测试扫描链 2006 的扫描链图像的最初扫描链图像 2508。正如所指的那样, 可以将这些最初扫描链图像 2508 与另外扫描链图像 2510 非破坏性地组合在一起, 以获得最后扫描链图像 2512。

现在参照图 26A-26C, 图 26A-26C 描绘了按照本发明, 压缩工具 2504 压缩芯片 HW 数据库 2108 的方法的高级逻辑流程图。正如下面详述的那样, 所例示的方法至少实现三种大小优化。

首先, 如果系统固件 1930a 决不会设置或读取 Dial 实例, 可以从芯片 HW 数据库 2108 中删除与 Dial 实例有关的信息。由于这样的 Dial 实例决不会被系统固件 1930a 设置或读取, 在 HW 配置数据库 1932a 内决不会引用, 于是, 可以除去与这样 Dial 实例相对应的 DIDS 1202。注意到系统固件 1930a 不设置或读取 Dial 实例的事实未必意味着在模拟或实验室调试期间不设置或读取 Dial 实例是重要的。许多 Dial 实例 (例如, 模式 Switch) 决不会由系统固件 1930a 设置, 但在模拟期间加以测试, 以保证如果以后的固件版本需要, 模式 Switch 可以适当地工作。

与 Dial 实例有关的信息可能无用的第二个原因是在所有配置中将 Dial 实例设置成唯一值。在这种情况下, 可以从芯片 HW 数据库 2108 中除去与 Dial 实例相对应的 DIDS 1202, 这是因为, 设置 Dial 实例的效果可以改为借助于通过设置 Dial 实例获得的锁存器值设置扫描到集成电路芯片 1910 中的最后扫描链图像 2512 取得。同样, 可以删除设置 Dial 实例的系统固件 1930b 内的代码, 以缩小最终从实验室测试和调试中获得的系统固件 1930a 的大小。

第三，通过删除系统固件1930a决不会设置给Dial的值，可以优化DDDS 1200中的映射表1224。

在作出前面的优化时，对读取的Dial实例加以特别考虑。一般说来，当读取Dial实例时，在如下所述的示范性压缩方法中假设必须将包含读取的Dial实例的整个Dial树保存在它的芯片HW数据库内。另外，假设必须保存包含读取的Dial实例的Dial树中Dial的映射表内所有项目，因为在商业推广中，硬件可能将底层锁存器设置成除了系统固件读取的那些之外的值。因此，事先不能确定需要哪些映射表项目来读取Dial实例。尽管这些假设限制了压缩，但它们保证了可以容易地访问读取的每个Dial实例，与Dial实例是顶层Dial实例还是较低层Dial实例无关。

首先参照图26A，该过程从方块2600开始，然后转到方块2602，方块2602例示了如上所述，将芯片HW数据库2108装入易失性存储器1928b中的和创建存储器内数据结构1208、1210和2325的压缩工具2504。另外，正如在方块2604上所描绘的那样，压缩工具2504与每个DIDS 1202相联系地在存储器中创建只供压缩工具2506使用的一些附加临时字段。这些临时字段包括用于存储Dial用法信息2500内设置给相关Dial实例的值（若有的话）的Dial实例值结构（DIVS）。对于IDial实例，特别管理DIVS。具体地说，DIVS或者是空的，或者包含指示设置IDial实例的记号，或者，只对于顶层IDial实例，如果可应用，包含设置给IDial实例的单一值。在方块2604上为每个DIDS 1202创建的临时字段还包括Dial实例保存字段（DIPF），如果应该保存相关DIDS（即，不从压缩芯片HW数据库中删除），DIPF被设置成TRUE，否则，被设置成FALSE。明确地列在优先文件2502中作为要保存的DIDS的每个DIDS 1202的DIPF（若有的话）被初始化成TRUE，所有其它DIPF被初始化成FALSE。

然后，该过程从方块2604转到方块2606，方块2606例示了压缩工具2504进入处理顶层指针阵列1206中的每个顶层指针1250，以便将来自Dial用法信息2500的相关信息输入每个DIDS 1202的DIPF和DIVS中的循环。如果所有顶层指针1250都得到处理，该过程穿过页面连接符B到如下所述的图26B。但是，如果所有顶层指针1250还没有全部得到处理，在方块2608上选择顶层指针阵列1206内的下一个顶层指针1250加以处理。

然后，该过程从方块2608转到方块2610和2612。方块2610例示了压

缩工具2504处理与当前顶层指针1250引用的DIDS 1202相对应的Dial实例领头的Dial树中的每个非IDial。压缩工具2504将包含在Dial用法信息2500内的相应Dial实例的值加入每个这样DIDS 1202的DIVS中。另外，如方块2612所示，压缩工具2504处理与当前顶层指针1250引用的DIDS 1202相对应的Dial实例领头的Dial树内的每个IDial。对于每个这样的IDial，如果Dial用法信息2500指示IDial已被设置，压缩工具2504将设置记号加入DIVS中。

接着，在方块2614上，如果Dial用法信息2500指示读取了Dial树中的某一个Dial，压缩工具2504设置与当前顶层指针1250引用的DIDS 1202相对应的Dial实例领头的Dial树中每个DIDS 1202的DIPF。换句话说，如果读取了Dial树中的某一个Dial实例，Dial树中的每个DIPF都被设置成TRUE。然后，该过程转到方块2616，方块2616例示了压缩工具2504检查与当前顶层指针1250引用的DIDS 1202相对应的每个顶层IDial（若有的话），以确定优先文件2502是否指示IDial只被设置成单一值。如果是，压缩工具2504将包含在优先文件2502内的值加入那些顶层IDial的DIVS中，并且，除去设置的记号（如果存在的话）。

此后，该过程返回到方块2606，方块2606例示了处理循环的继续，直到顶层指针阵列1206内的所有顶层指针1250都得到处理为止。一旦所有顶层指针1250都得到处理，该过程穿过页面连接符B到图26B。

现在参照图26B，该过程从页面连接符B转到方块2620，方块2620例示了处理顶层指针阵列1206内的每个顶层指针1250的第二处理循环。如果在方块2620上确定在当前处理循环中已经处理了顶层指针阵列1206内的所有顶层指针1250，该过程穿过页面连接符C和在图26C中继续进行。否则，该过程转到方块2622，方块2622描绘了选择顶层指针阵列1206内的每一个顶层指针1250加以处理。

在方块2622之后，检查与当前顶层指针1250引用的DIDS 1202相联系的DIVS和DIPF是否符合判定方块2624、2630、和2640分别代表的三个条件之一。如果在方块2624上确定DIPF具有TRUE的值或如果相关DDDS 1200中的类型字段1220指示DIDS 1202对应于Dial组，该过程简单地从方块2624返回到方块2620对下一个顶层指针1250（若有的话）加以处理。

但是，如果在方块2630上确定与当前顶层指针1250引用的DIDS 1202

相联系的DIPF具有FALSE的值和相关DIVS是空的，那么，压缩工具2504可以从芯片HW数据库2108中除去DIDS 1202，因为相应Dial实例没有一个被设置或读取。于是，正如在方块2632上所例示的那样，压缩工具2504从芯片HW数据库2108中删除DIDS 1202，以及已删顶层DIDS 1202领头的Dial树中的每个较低层DIDS 1202（若有的话）。另外，压缩工具2504从顶层指针阵列1206中删除相关顶层指针1250，并且将指向每个已删DIDS 1202的实例指针1228设置成NULL。然后，在方块2634上确定已删DIDS 1202的父指针1233是否被设置成NULL。如果是，该过程返回到已经描述过的方块2620。另一方面，如果父指针不是NULL，那么，与已删DIDS 1202相对应的顶层Dial实例属于Dial组实例。由于顶层Dial实例决不会被设置或读取，可以从它各自的Dial组实例中安全地除去每个这样的顶层Dial实例。于是，如方块2636所示，压缩工具2504从与Dial组实例相对应的DIDS 1202中删除指向顶层Dial实例的已删DIDS 1202的输出指针1238。如果从Dial组实例的DIDS 1202中删除输出指针1238除去了Dial组的最后一个成员，也从芯片HW数据库2108中删除与Dial组实例相对应的DIDS 1202。这个过程继续下去，如有可能，破坏Dial组的分层结构层。在方块2636之后，该过程返回到已经描述过的方块2620。

返回到方块2640，压缩工具2504确定与当前顶层指针1250引用的DIDS 1202相联系的DIPF是否含有FALSE的值和相关DIVS是否包含单一值。如果不是，该过程返回到已经描述过的方块2620。如果是，在方块2642上参照DIDS 1202的父字段1232进一步确定Dial实例是否属于Dial组。如果是，该过程最好不作进一步处理地返回到方块2620，表示DIDS 1202将得到保存。最好保存DIDS 1202是因为设置Dial组的操作是不可分的，如果在set_Dial_group() API调用中引用已除Dial实例，将会出现失败。响应在方块2642上与顶层指针1250引用的DIDS 1202相对应的Dial实例不属于Dial组的确定，该过程转到方块2644。

方块2644例示了参照映射表1224（如有必要）沿着Dial树向下传播包含在DIVS中的单个Dial值，以便确定在Dial树末端的锁存器的锁存器值。然后，正如在方块2646所例示的那样，将在方块2644上确定的锁存器值放入参照芯片映射表2325确定的扫描链位置上的最初扫描链图像2508中。因此，如方块2648所示，正如上面参照方块2632所述的那样，当前顶层指针

1250 引用的 DIDS 1202、它的较低层 Dial 树、和顶层指针 1250 本身统统从芯片 HW 数据库 2108 中除去。另外，如方块 2650 所示，从系统固件 1930b 中除去（通常由编程人员）用于设置与已删 DIDS 1202 相对应的顶层 Dial 实例的 set_Dial() API 调用。此后，该过程返回到已经描述过的方块 2620。

现在参照图 26C，该过程从页面连接符 C 开始，然后转到方块 2660，方块 2660 例示了处理 Dial 指针阵列 1208 内的所有 Dial 指针 1252，以便从芯片 HW 数据库 2108 中删除任何无用 DDDS 1200 和删除映射表 1224 内的任何无用项目的处理循环。在 Dial 指针阵列 1208 内的所有 Dial 指针 1252 都得到处理之后，该过程转到如下所述的方块 2690。但是，如果并非所有 Dial 指针 1252 都得到处理，该过程从方块 2660 转到方块 2662，方块 2662 例示了选择下一个 Dial 指针 1252 加以处理。

在选择了下一个 Dial 指针 1252 之后，压缩工具 2504 在方块 2664 上确定当前 Dial 指针 1252 引用的 DDDS 1200 的实例指针阵列 1226 是否是 NULL。如果是，如方块 2666 所示，整个 DDDS 1200 都是无用的，从芯片 HW 数据库 2108 中除去它。在方块 2666 之后，该过程返回到已经描述过的方块 2660。

响应在方块 2664 上当前 Dial 指针 1252 引用的 DDDS 1200 内的所有实例指针 1228 不是 NULL 的确定，在方块 2670 上进一步确定类型字段 1220 是否指示 DDDS 1200 定义 IDial。如果是，可能没有对映射表 1224 优化，该过程返回到方块 2660。如果压缩工具 2504 在方块 2670 上确定当前 Dial 指针 1252 引用的 DDDS 没有定义 IDial，该过程转到方块 2672。方块 2672 描绘了与实例指针 1228 引用的某个 DIDS 1202 相联系的某个 DIPF 是否具有 TRUE 的值的确定。如果是，这个状况表示通过 DDDS 1200 定义的 Dial 的至少一个 Dial 实例已被读取，因此，需要整个映射表 1224。于是，该过程无需对映射表 1224 进行任何优化地返回到方块 2660。

但是，如果压缩工具 2504 在方块 2672 上确定与实例指针 1228 引用的 DIDS 1202 相联系的所有 DIPF 都具有 FALSE 的值，该过程从方块 2672 转到例示在方块 2674、2676、和 2678 上的处理循环。这个处理循环代表压缩工具 2504 处理当前 Dial 指针 1252 引用的 DDDS 1200 的实例指针阵列 1226 内的每个实例指针 1228，以便建立包含系统固件 1930 设置给与 DIDS 1202 相对应的 Dial 实例的所有值的 Dial 值集。正如在方块 2678 上所指的那样，Dial 值是从与每个 DIDS 1202 相联系的 DIVS 中获得的。在通过处理每个实例指针

1228 建立了 Dial 值集之后，该过程从方块 2674 转到方块 2680。方块 2680 例示了压缩工具 2504 除去在 Dial 值集内没有找到 Dial 输入值的当前 Dial 指针引用的 DDDS 1200 的映射表 1224 中的每个项目。这个过程沿着 Dial 树向下继续下去，删除没有用于生成 Dial 值集的映射表项目。因此，通过除去无用项目优化了各自 Dial 的映射表 1224。此后，该过程返回到方块 2660。

响应在方块 2660 上 Dial 指针阵列 1206 内的所有 Dial 指针 1252 都得到处理的确定，压缩工具 2504 在方块 2690 上用指向提供实例全名部分的“词典”的指针取代实例名字段 1234 内的实例名的公共部分进行最后一次压缩。这种本领域普通技术人员熟知的压缩技术用指针取代实例名（或它们的一部分），这些指针通常显著地短于它们取代的实例名或实例名部分。然后，作为将 HW 配置数据库 1932a 装入服务处理器 1920 的非易失性存储器 1928a 中的过程中的步骤，可以在实例名字段 1234 内取代这些指针。在方块 2690 之后，压缩工具 2504 在方块 2692 上终止该处理。

在压缩工具 2504 按照在图 26A-26C 中描绘的方法压缩了所有芯片 HW 数据库 2108 之后，压缩芯片 HW 数据库 2108 然后可以用于通过简单地构建芯片指针数据结构 2320，构建存储在闪速 ROM 1924 内的硬件配置数据库 1932a。应该注意到，通过压缩工具 2504 实现的压缩方法不是排他的。HW 配置 API 1934b 最好包括允许开发者除去各个 DIDS 1202，除去映射表 1224 的中的项目，和进行与例示在图 26A-26C 中的那些类似的其它优化的一整套 API。

在如上所述的本发明的实施例中，已经假设与模拟配置锁存器或硬件锁存器逻辑耦合的每个 Dial（即，LDial 或 IDial）可以设置包含在模拟配置锁存器或硬件锁存器中的值。但是，实际上，往往希望能够读取这样的锁存器，而不是允许系统固件或模拟器设置（或变更）锁存器值。

鉴于前面情况，本发明的优选实施例支持这里称为只读 Dial 或 RDial 的另外一类配置实体。最好存在与如上所述的每种类型 Dial 或 Dial 组相对应的只读配置实体，即，只读 LDial、CDial、IDial 和 Dial 组。为了易于理解，每个只读配置实体在这里通过在 Dial 或 Dial 组类型名（例如，LDial、CDial、IDial 和 Dial 组）前面加上将配置实体指定为只读的“R”（例如，RLDial、RCDial、RIDial 和 RDial 组）来引用。

RDial 和 RDial 组受许多规则集支配。第一，RDial 和 RDial 组是只读的，并且，按定义，可以通过模拟器或系统固件设置。因此，不能将默认值指定

给 RDial 和 RDial 组。

第二，在配置说明语句内定义 RDial 和 RDial 组的语法最好与上面针对相应非只读配置实体所述的语法相同，除了在定义配置实体的关键字前面加上“R”之外，例如，可以像下面那样给出 RLDial 的示范性配置说明语句：

```
RLDial state_machine(state_vector(0..1)
                    )=
                    {idle=>0b00;
                    start=>0b01;
                    wait=>0b10;
                    end=>0b11
                     };
```

上面给出的示范性配置说明语句从关键字“RLDial”和 RDial 名开始，“RLDial”指定正被说明的 RDial 的类型是 RLDial，和在这种情况下，RDial 名是“state_machine”。接着，配置说明语句枚举了其状态通过 RLDial 读取的信号名。在枚举了信号标识符之后，配置说明语句包括列出 RLDial 的允许枚举“输入”值（或设置）和每个枚举输入值的相应信号（即，“输出”）值的映射表。还应该注意到，为所有枚举值指定的信号状态是唯一的，并且，集体表示信号状态的唯一合法模式。

第三，RDial 拥有有关与 Dial 和 RDial 的互连和分组 Dial 和/或 RDial 以形成 RDial 组的一组不同规则。下面参照图 27 详细阐述这些规则，图 27 是包括与模拟模型或硬件系统的锁存器 2760-2778 存在指定逻辑连接的 Dial 和 RDial 的示范性配置数据库 2700 的一部分的图形表示。

首先，正如上面针对相应 Dial 所阐述的那样，RDDial 在与其它 RDial 和锁存器互连方面受到相似的限制。也就是说，在优选实施例中，RIDial 或 RLDial，但不是 RCDial，可以含有它与锁存器直接耦合的输出端，和 RCDial，但不是 RIDial 或 RLDial，可以含有它与较低层 RDial 的输入端连接的输出端。因此，例如，RCDial 2740 含有与 RCDial 2742 的输入端连接的输出端，RCDial 2742 又含有分别与 RLDial 2744 和 RIDial 2746 的输入端连接的两个输出端。RLDial 2744 和 RIDial 2746 分别含有与锁存器 1770 和 2772 连

接的输出端。

另外，RCDial 可以含有它与任何类型 Dial 的输入端连接的输出端，但不允许 Dial 含有它与任何 RDial 的输入端连接的输出端。例如，RCDial 2740 含有与 CDial 2724 的输入端连接的输出端。尽管在图 27 中没有明确示出，但应该注意到，RDial 可以含有与处在同一子树的多个不同层上的 RDial 和/或 Dial 的输入端连接的输出端。

为了防止冲突设置，上文定义的 Dial 和 Dial 组允许每个锁存器、Dial、和 Dial 组含有在 n 路 Dial 树中作为在分层结构中在它“上面”的父辈的至少一个 Dial 或 Dial 组。例如，CDial 2722 和 CDial 2724 的每一个只含有一个 Dial 父辈（即，CDial 2720），CDial 2726 和 CDial 2728 的每一个只含有一个 Dial 父辈（即，CDial 2722）和 LDial 2730 和 IDial 2732 的每一个只含有一个 Dial 父辈（即，CDial 27424）。但是，由于按照定义，RDial 和 RDial 组是只读的，任何 Dial 或 RDial 可以含有一个或多个 RDial 或 RDial 父辈，而在 Dial 设置之间不会有任何冲突可能。也就是说，RDial 可以含有它与另一个 RDial 或 Dial 的输出端也与之连接、受其它规则支配、和假设没有形成闭环的锁存器、Dial 或 RDial 连接的输出端。换句话说，允许每个锁存器和 Dial 含有至少一个 Dial 父辈，但每个锁存器、Dial 和 RDial 可以含有一个或多个 RDial 父辈，与锁存器或 Dial 是否还含有 Dial 父辈无关。例如，在图 27 的配置数据库 2700 中，RCDial 2740 和 RCDial 2750 每一个的输出端与 RCDial 2742 的输入端连接。类似地，CDial 2720 和 RCDial 2740 每一个都含有与 CDial 2724 的输入端连接的输出端。此外，RLDial 2752 和 LDial 2754 每一个都含有与锁存器 2776 连接的输出端。

最后的规则与 RDial 组的结构有关。正如上面参照图 11A 所详述的那样，在优选实施例中，Dial 组可能只包含顶层 Dial 和/或分层嵌套 Dial 组。相反，RDial 组可能包含在分层结构的任何层上的 RDial 或 Dial、和/或 Dial 组或 RDial 组。允许这种附加灵活性是因为 RDial 组像 RDial 那样决不会被模拟器或系统固件设置。

配置数据库内的 RDial 和 RDial 与以前按照如上所述的规则描述的 Dial 和 Dial 组组合在一起的实现允许构造出三种类型的树。第一，如 Dial 树 2702 和 2708 所示范的那样，Dial 树可以包括 Dial 和锁存器，但没有 RDial。第二，RDial 树，例如，RDial 树 2706 可以包括 RDial 和锁存器，但没有 Dial。

第三，正如混合树 2704 所例示的那样，混合树可以被构造成包含一个或多个 RDial、一个或多个 Dial、和一个或多个锁存器。

为了支持 RDial 和 RDial 组，可以对模拟配置数据库和 HW 配置数据库作一些修改。首先，将每个 DDDS 1200 内的类型字段 1220 的值集扩充成包括标识 RDial 组的附加值和 RDial 的附加类型。例如，可以用值 RL、RC、RI 和 RG 扩充值集，以便分别标识与 RLDial、RCDial、RIDial、和 RDial 组相对应的 DDDS 1200。这些新值的加入保证了在试图设置任何实例之前最好设置相关 DDDS 1200 的类型字段 1220 的 set_Dial()或 set_Dial_group() API 调用将不再试图设置 RDial 或 RDial 组。

另外，正如在图 28A 中所例示的那样，将每个 DIDS 1202 扩展成包括只读父字段 2800，只读父字段 2800 包括 0 个或更多个只读父指针 2801。每个非 NULL 只读父指针 2801 定义 DIDS 1202 所代表的实例的输入端和较高层 RDial 的输出端之间的连接或 DIDS 1202 所代表的实例包括在 RDial 组内。如上所述，除了 Dial 或 Dial 组父辈（若有的话）之外，DIDS 1202 所代表的实例可以含有多个 RDial 父辈和/或属于多个 RDial 组。

正如在图 28B 中所描绘的那样，类似地将配置数据库内的锁存器数据结构（例如，HW 配置数据库的锁存器数据结构 2230 或模拟配置数据库的锁存器数据结构 1204）扩充成包括只读父字段 2802，只读父字段 2802 包括一个或更多个只读父指针 2803。每个非 NULL 只读父指针 2803 定义锁存器数据结构所代表的锁存器实例的输入端和 RIDial 或 RLDial 的输出端之间的连接。如上所述，在模拟中，最好参照父指针 1242 所指的 LDial 或 IDial 的范围指定锁存器名字段 1244（图 12）内的锁存器名。如果父指针 1242 是 NULL，表明与锁存器数据结构 1204 相对应的配置锁存器没有 Dial 父辈，那么，最好参照与只读父字段 2802 内的第一只读父指针 2803 所指的 DIDS 1202 相对应的 RLDial 或 RIDial 的范围指定包含在锁存器名字段 1244 内的锁存器名。

最后，虽然在结构上不作改变，但在长度上延长顶层指针阵列 1206（图 12），以支持 RDial 和 RDial 组。具体地说，顶层指针阵列 1206 包括指向每个顶层 RDial 组、包括在 RDial 组（即，含有非 NULL 只读父字段 2800）的每个顶层 RDial、和不包括在 RDial 组（即，含有 NULL 只读父字段 2800）的每个顶层 RDial 的 DIDS 1202 的顶层指针 1250。

前面为了支持 RDial 和 RDial 组而对配置数据库中的数据结构加以修改

必然伴随着对上面参照图 13 所述的将配置数据库从非易失性存储器装入易失性存储器中和扩展配置数据库的方法加以修改。图 29 是按照本发明的优选实施例将包含RDial和/或RDial组的配置数据库从非易失性存储器装载到易失性存储器中的示范性方法的高级逻辑流程图。正如使用的相似标号所指的那样，例示在图 29 中的方法基本上与上面参照图 13 所述的方法相似，加上一些步骤以保证只处理每个数据结构一次。

正如撇号（'）所指的那样，在方块 1308'上对如上所述的方法作出第一次修改。在图13的方法中，方块1308代表当前顶层指针1250引用的DIDS 1202是否对应于属于Dial组的Dial或Dial组的确定。图 29 中的方块 1308'将当前顶层指针 1250 引用的 DIDS 1202 是否对应于属于 RDial 组的 Dial、RDial、Dial 组或 RDial 组的进一步确定加入这个确定中。如果每个确定都获得肯定响应，正如返回到方块 1304 的过程所指的那样，终止当前顶层指针 1250 的处理，因为当处理 Dial 组或 RDial 组时，将处理当前顶层指针 1250 引用的 DIDS 1202。这个确定保证了只处理顶层 Dial 和 RDial 的 DIDS 1202 一次。

为了保证在将配置数据库装入易失性存储器中的过程中，也只处理较低层数据结构一次，在方块 2900 上进一步确定当前顶层指针 1250 引用的 DIDS 1202 是否对应于 RDial 或 RDial 组。如果不是，也就是说，如果根在 DIDS 1202 上的树对应于 Dial 或 Dial 组，那么，树中没有一个“孩子”可以是 RDial 或 RDial 组。于是，正如从方块 2900 到方块 1316 的过程所指的那样，可以像以前那样处理在当前 DIDS 1202 下面的子树。

但是，响应在方块 2900 上当前顶层指针 1250 引用的 DIDS 1202 对应于 RDial 或 RDial 组的确定，该过程转到方块 2902 和随后的方块，方块 2902 和随后的方块代表处理 RDial 或 RDial 组的子树，以保证只处理配置数据库中的每个数据结构一次。为了跟踪哪些数据结构已经得到处理，正如方块 2902 上的处理那样，首先标记当前 DIDS 1202。然后，正如在方块 2904 上所指的那样，该过程进入处理当前顶层 DIDS 1202 的输出指针阵列 1236 内的每个输出指针 1238 的处理循环。一旦所有输出指针 1238 都得到处理，该过程从处理循环退出，返回到方块 1304，方块 1304 代表还有没有另外顶层指针需要处理的确定。

如果在方块2904上确定并非所有输出指针1238都得到处理，在方块2906上选择输出指针阵列 1236 内的下一个输出指针 1238 加以处理。然后，该过

程转到方块 2910 和 2912，方块 2910 和 2912 分别例示了所选输出指针 1238 是否指向与 Dial 或 Dial 组相对应的 DIDS 1202，或输出指针引用的 DIDS 1202 是否是像以前处理的那样已经标记的 RDial 或 RDial 组的确定。如果在方块 2910 上获得肯定结果，RDial 或 RDial 组和 Dial 或 Dial 组之间的接口得到定位。由于当选择另一个顶层指针 1250 加以处理时，处理 Dial 或 Dial 组领头的子树，终止这个子树的处理，和该过程返回到方块 2904。响应在方块 2912 上像以前处理的那样标记了当前输出指针 1238（与 RDial 或 RDial 组相对应）引用的 DIDS 1202 的确定，类似地终止子树的处理。

另一方面，如果例示在方块 2910 和 2912 上的确定得出否定的结果，在方块 2914 上标记和处理当前输出指针 1238 引用的 DIDS 1202 或锁存器数据结构 1204。在方块 2914 上进行的处理与例示在方块 1310、1312、1314 和 1316 上和如上所述的处理相同。正如在方块 2914 上进一步所指的那样，受在方块 2912 和 2914 上描绘的两个条件支配，类似地标记和处理直到和包括在子树末端的锁存器的子树中的每个较低层数据结构。也就是说，如果检测到与 Dial 或 Dial 组的接口，或者，如果检测到已经标记的数据结构（例如，与 RDial 或 RDial 组对应的锁存器数据结构 1204 或 DIDS 1202），中断对任何子树的处理。在方块 2914 之后，该过程返回到已经描述过的方块 2904。

RDial 和 RDial 组的实现还需要以对于数字设计的模拟和配件实现两者都读取 Dial、Dial 组、RDial、和 RDial 组的方式作一些调整。具体地说，随着在，例如，方块 1620（图 16A）和 1660（图 16B）上穿过树创建 read_Dial() 或 read_Dial_group() API 调用最终针对的感兴趣锁存器组，最好记录或标记为了创建锁存器组而穿过的“分支”（即，与 Dial 或 RDial 相对应的 DIDS 1202）。正如在，例如，方块 1624（图 16A）和 1664（图 16B）上所例示的那样，当沿着树“向上”传播锁存器组中的锁存器的锁存器值以获取 Dial 和 RDial 设置时，从锁存器数据结构 1204 开始向上穿过正确分支以获取感兴趣的 Dial 或 RDial 设置。换句话说，由于除了单个 Dial 父辈（若有的话）之外，Dial 或 RDial 可以含有一个或多个 RDial 父辈，必须记录或标记向下穿过以获取锁存器值的分支的父指针，以保证向上穿过相同分支以获取所需 Dial 或 RDial 设置。

另一个调整最好对例示在图 26A-26C 中的压缩例程作出。在所述的实施例中，图 26B 的方块 2632 描绘了除去 Dial 用法信息 2500（因此，DIPF）指

示不被设置或读取的顶层 DIDS 1202 的整个 Dial 树。对于如图 27 所示，允许树向上分支的 RDial 和 RDial 组的实现，如果将这个步骤修改成保存也属于读取的 Dial 实例的子树的任何较低层 DIDS 1202，那就更好了。在这种修改中，在除去顶层 DIDS 1202 之后，测试已删 DIDS 1202 的子树中每个较低层 DIDS 1202 的 DIPF，以确定它是否含有指示较低层 DIDS 1202 也属于读取的树的值 TRUE。如果不是，也可以除去较低层 DIDS 1202，并且，除去过程沿着子树向下继续下去。但是，如果含有设置成 TRUE 的 DIPF 的较低层 DIDS 1202 得到定位，不除去较低层 DIDS 1202 和它的子树。但是，将它的父指针 1233 设置成 NULL，以反映父指针 1233 引用的父 DIDS 1202 的除去。

当在实验室环境下调试和测试硬件数字设计或对推广硬件系统失败作出响应时，分析失败以确定它们的原因是至关紧要的任务。传统上，为了有助于失败原因的确定，获取硬件数字系统内的所有测试扫描链的扫描群。然后，分析扫描链图像以确定失败的原因。在试图再现模拟失败的过程中，时常人工选择特定扫描链位和将它们输入数字系统的模拟模型中。硬件失败的模拟使信号可见度得到提高和使模拟器的步进能力得到辅助支持，以帮助失败原因的确定。

这种传统失败分析是令人乏味的和易出错的，因为用户首先必须试图确定扫描群提供的“位海”中的那些位对于到模拟系统的端口来说是重要的，以便再造错误条件。然后，用户必须参照可能错误纸质文档人工浏览扫描群，以便确定感兴趣位的值。最后，用户必须编程 RTX 或其它软件程序，将适当的位值装入模拟模型的锁存器中。

本发明通过辅助支持如上所述的配置说明语言和硬件和模拟配置数据库的特征，改善了这样的现有分析技术。现在参照图 30，图 30 描绘了利用模拟模型分析硬件系统的所选状态，尤其，硬件系统的失败状态的示范性过程的高级流程图。如图所示，该过程从芯片分析工具 3004 的操作开始，芯片分析工具 3004 最好包括在计算机系统，譬如，图 1 的数据处理系统 6 上执行的软件。芯片分析工具 3004 接收测试扫描链图像 3000 作为输入，测试扫描链图像 3000 集体代表系统失败状态和每一个包含硬件数字设计（例如，正在测试之中的服务器计算机）中的各自集成电路芯片的所有锁存器的锁存器值。另外，芯片分析工具 3004 接收硬件数字设计中的每种类型集成电路芯片的每一芯片型芯片 HW 数据库 2108。最后，芯片分析工具 3004 配有所选 Dial 表

3002，Dial 表 3002 标识每个芯片 HW 数据库 2108 内的那些 Dial 被认为在模拟中相对接近硬件失败状态。

芯片分析工具 3004 参照芯片 HW 数据库 2108 处理扫描链图像 3000 和所选 Dial 表 3002，以便为硬件数字设计中的每个集成电路芯片生成各自芯片配置报告 3006 和模拟设置文件 3008。每个芯片配置报告 3006 包括与硬件数字设计中的特定集成电路芯片相联系的所有 Dial 实例的人可读和可打印列表，以及在失败点每个 Dial 实例的设置（如果合法值可用的话）。对于合法值不可用的 Dial 实例，报告底层锁存器值。每个模拟设置文件 3008 是指定在与相应集成电路芯片相联系的所选 Dial 表 3002 中标识的每个 Dial 的设置（如果合法值可用的话）的机器可读文件。正如下面所说明的那样，RTX 1420（图 14）利用模拟设置文件 3008 将硬件数字设计的模拟模型 1400 配置成接近硬件数字设计的失败状态的状态。

现在参照图 31，图 31 例示了按照本发明，图 30 的芯片分析工具 3004 生成用于分析硬件失败的芯片配置报告 3006 和模拟设置文件 3008 的例示性方法的高级逻辑流程图。如图所示，该过程从方块 3100 开始，然后转到方块 3102，方块 3102 描绘了芯片分析工具 3004 确定硬件数字设计中的每个集成电路芯片的扫描链图像 3000 是否都得到处理。如果所有集成电路芯片的扫描链图像 3000 都得到处理，在方块 3130 上终止该过程。但是，如果并非所有扫描链图像 3000 都得到处理，在方块 3104 上选择要处理的下一个集成电路芯片的扫描链图像 3000 和芯片 HW 数据库 2108。

然后，如图 31 所示的过程进入方块 3106-3110 上的处理循环，在这个处理循环中参照芯片 HW 数据库 2108 的锁存器指针阵列 1210 中的锁存器指针 1254 处理从当前集成电路芯片中扫描的每个感兴趣锁存器值。具体地说，芯片分析工具 3004 在方块 3106 上确定是否所有锁存器指针 1254 都得到处理。如果是，该过程从方块 3106 转到如下所述的方块 3120。但是，如果所有锁存器指针 1254 还没有全部得到处理，在方块 3108 上选择锁存器指针阵列 1210 内的下一个锁存器指针 1254 加以处理。接着，在方块 3110 上，芯片分析工具 3004 利用包含在当前锁存器指针 1254 引用的锁存器数据结构 2230 的方法名字段 2234a（图 23B）中的测试扫描环标识符和偏移值对，以便在扫描环图像 3000 内定位与锁存器数据结构 2230 相对应的硬件锁存器的锁存器值。然后，将这个锁存器值存储在参照芯片映射表 2325 内当前集成电路芯片的芯片

ID 的位置确定的锁存器值字段 2324 的适当项目中。此后，该过程返回到方块 3106。

响应在方块 3106 上当前芯片 HW 数据库 2108 的锁存器指针阵列 1210 内的所有锁存器指针 1254 都得到处理的确定，该过程转到方块 3120。方块 3120 描述了芯片分析工具 3004 参照映射表 1224，沿着芯片 HW 数据库 2108 内的 DIDS 树的所有分支向上传播包含在每个锁存器值字段 2324 中的锁存器值集，以便如有可能，获取设置每个 Dial 和 RDial 的设置（即，输入值）。在锁存器值字段 2324 内的锁存器值对应于硬件失败状态的情况下，经常出现对于 Dial 或 RDial 实例，试图沿着树向上传播至少一些锁存器值将导致至少一个“输出”值不在在映射表 1224 内指定的合法输出值当中的情况。在这样的情况下，Dial 或 RDial 实例（和在相同树中在它上面的任何 RDial 或 Dial）被标志成具有非法值。这样的非法值经常暗示硬件失败的原因。

应该注意到，从锁存器值中导出 Dial 和 RDial 值的能力取决于本发明引入的配置说明语言的可逆性。也就是说，如方块 3120 所示，如果在 Dial（和 RDial）输入和输出之间不存在一一对应映射，不能从锁存器值明确地确定 Dial（和 RDial）设置。

在方块 3120 之后，该过程转到方块 3122，方块 3122 描绘了芯片分析工具 3004 创建当前集成电路芯片的芯片配置报告 3006。如上所述，芯片配置报告 3006 是包含在方块 3120 上确定的当前芯片 HW 数据库 2108 内的所有 Dial 和 RDial 实例的列表和它们的相应设置（若有的话）的人可读文件。在芯片配置报告 3006 中标出具有非法值的 Dial 和 RDial 实例，和列出底层锁存器的锁存器值以便于分析。如方块 3124 所示，芯片分析工具 3004 还创建当前集成电路的 RTX 兼容模拟设置文件 3008。模拟设置文件 3008 最好包括只在所选 Dial 表 3002 内指定的 Dial 实例的 Dial 设置，和如果在所选 Dial 表 3002 中指定的 Dial 实例具有非法值，由 Dial 控制锁存器组中的底层锁存器的锁存器值。然后，正如下面所说明的那样，通过在模拟环境下运行的 RTX 1420，这些 Dial 实例设置和锁存器值可以自动应用于模拟模型 1400。

应该认识到，由于受 Dial 控制的锁存器的数量通常只占集成电路中的锁存器的总量的很小份额，通过使用本发明的配置说明语言将 Dial 与特定配置锁存器相联系，数字系统的设计者已经极大地减少了在再造系统失败状态时要考虑的锁存器值的数量和识别出最有可能是再现硬件失败状态所必不可少

的那些锁存器。通过指定感兴趣的特定用户选择Dial实例(不是RDial实例),所选Dial表3002进一步减少了从端口传回模拟模型1400的硬件状态信息的数量。

在方块3124之后,描绘在图31中的过程返回到方块3102,以便处理硬件数字设计中的下一个集成电路芯片(若有的话)。在硬件数字设计内的所有集成电路芯片都得到处理之后,在方块3130上终止该过程。

再次参照图30,在按照图31的过程为硬件数字设计内的每个集成电路芯片创建了各自模拟配置文件3008之后,通过执行RTX 1420在数字设计的模拟模型1400内近似硬件失败状态。另一方面,应该注意到,一般不希望在模拟中再现完全相同的硬件失败状态,因为,按照定义,数字设计不会从失败状态开始正确地工作。

为了在模拟中近似硬件失败状态,RTX 1420首先对模拟器1410提供的API作出标准API调用,以便为模拟执行用于初始化模拟模型1400的正常初始化过程。接着,RTX 1420根据用户提供定制初始化修改文件3010,可选地对模拟模型1400的配置作出单独用户指定定制。这些定制修改可以为了,例如,调整暴露特定失败模式或提供某些类型失败的可见度的参数而作出。最后,RTX 1420应用包括在模拟配置文件3008中的Dial实例设置和锁存器值。如像上面参考图14和17A所详细描述的,RTX 1420通过对配置API 1406的set_Dial() API调用设置Dial实例,在反映了模拟配置数据库1404中的Dial实例设置之后,配置API 1406调用PUTFAC() 1414将相应锁存器值设置在模拟模型1400中。RTX 1420类似地利用API调用将包含在与非法Dial值相对应的模拟设置文件3008内的锁存器值设置给模拟模型1400的配置锁存器和配置数据库1404的锁存器值字段1246(图12)。借助于如此设置的模拟模型1400,RTX 1420直接执行模拟器1400对以模拟模型1400为背景的一个或多个测试用例,以便试图在模拟中再现硬件失败状态。

如上所述,诸如Dial、Dial组和Register,尤其,RDial和RDial组之类的配置实体的使用借助于翻译和分析数字系统(例如,模拟模型或硬件系统)的一“群”状态,因为诸如Dial和Register名和值之类,与系统状态有关的信息以人可读的形式表示,而不是被表示成“位海”。但是,当需要大型数字系统的一“群”完整状态时,返回的数据量仍然十分巨大,即使使用通过Dial、Dial组和Register名和值提供的解释性引导语。因此,人们希

望能够有选择地控制响应对模拟或硬件系统的一部分或整群状态的请求表示特定 Dial、Dial 组和 Register 的条件。

为了允许以这种方式有选择地表示 Dial、Dial 组和 Register，将本发明的配置说明语言推广到允许在 Dial、Dial 组或 Register 的定义中指定配置实体（例如，Dial、Dial 组或 Register）的“控制值集”，以便控制表示 Dial、Dial 组或 Register 的设置的条件。在许多可能实施例之一中，在正好出现在 Dial、Dial 组或 Register 定义的结束分号之前（即，在 Dial 默认值和阶段 ID(若有的话)之后)的带括号字段中指定 Dial、Dial 组或 Register 的控制值集，控制值集指定要表示的 Dial、Dial 组或 Register 的 Dial 或 Register 设置。当然，在其它实施例中，可以用不同语法表达控制值集，和控制值集指定要表示的 Dial、Dial 组或 Register 的 Dial 或 Register 设置。

为了例示定义控制值集的示范性语法，再次考虑上面定义的示范性 RL-Dial state_machine。当采用对于控制值的指定集合不显示 RDial 的控制值集表达式的默认解释时，如果应用如下语法，那么，如果含有设置“idle”或“end”，在系统群中不表示 RLDial state_machine，和如果含有设置“start”或“wait”，在系统群中表示 RLDial state_machine:

```
RLDial state_machine(state_vector(0..1)
                    )=
                    {idle=>0b00;
                    start=>0b01;
                    wait=>0b10;
                    end=>0b11
                     } (idle,end);
```

将传统表示法用于否定，通过像下面那样可替代地指定 RLDial state_machine，可以取得相同结果:

```
RLDial state_machine(state_vector(0..1)
                    )=
                    {idle=>0b00;
```

```
start=>0b01;
wait=>0b10;
end=>0b11
 }(!(start,wait);
```

在上面的例子中，对于 RLDial，控制值被指定成合法设置集当中的枚举值。当然，对于其它 Dial 和 Register 类型，将控制值调整成适合 Dial 和 Register 的类型。例如，对于 IDial，控制值是整数，也可以用十进制、十六进制或八进制指定控制值。

在优选实施例中，控制值字段进一步支持支配表示配置实体的方式或条件（若有的话）的一个或多个关键字。例如，可以应用关键字<no_display>来指定在系统群中决不会表示配置实体和它的设置。尖括号(即，“<”和“>”)或一些其它特殊字符最好用于定界出现在控制值字段内的关键字，以防分析相似枚举 Dial 设置时出现混乱。

对于 Dial 组，成员 Dial 的选择性表示通过指定分层包含在 Dial 组内的一个或多个控制 Dial 和为每个控制 Dial 指定一个或多个控制值的集合来完成。例如，考虑通过如下语句定义的图 11B 的示范性 Dial 组 F:

```
GDial F(C,[Z].B,[Y].A);
```

通过像下面那样加入控制值字段，可以将 Dial C 选作 Dial 组 F 的控制 Dial:

```
GDial F(C,[Z].B,[Y].A)(C=idle);
```

这个配置说明语句使得如果 Dial C 含有设置“idle”，不表示 Dial 组 F 内的 Dial 的设置。可替代地，如果希望如果 Dial C 处在“idle”设置下，表示构成 Dial 组 F 的 Dial 的设置，那么，可以像下面那样表达配置语句:

```
GDial F(C,[Z].B,[Y].A)(C!=idle);
```

本领域的普通技术人员应该认识到，对于 Dial、Dial 组或 Register，

也可以应用更复杂的控制值表达式。例如，取决于不止一个 Dial 或 Register 的设置的逻辑表达式可以用于控制 Dial、Dial 组和 Register 的表示。例如，再次考虑 Dial 组 F，可以像下面那样取决于 Dial C 和 D 两者的设置，指定用于表示 Dial 组 F 的控制值集:

```
GDial F(C,[Z].B,[Y].A)(C!=idle AND B=idle);
```

应用逻辑运算符 AND、OR、和 NOT 的类似逻辑表达式可以用于定义 Dial 和 Register 的控制值集。

还应该认识到，控制值表达式也可以用简明、带括号语法来表达。在这样的情况下，Dial、Dial 组或 Register 的设置不会得到表示，除非控制值表达式所指的所有 Dial、Dial 组和/或 Register 具有指定的控制值。

现在参照图 32，图 32 描绘了按照本发明，支持 Dial、Dial 组和 Register 的选择性表示的配置数据库 814、1404、1932 或 2108 的示范性实施例。也就是说，配置数据库 814（和从中导出的任何模拟配置数据库 1404 或硬件配置数据库 1932，2108）将指定的 Dial、Dial 组和 Register 的控制值集记录在定义设计的 HDL 文件中。本领域的普通技术人员从如下的描述中可以认识到，所例示的数据库结构只是存储配置实体的控制值信息的大量可能数据结构之一。

将图 32 与图 12A 进行比较可以看出，图 32 的配置数据库 814 与上面参照图 12A 所述的配置数据库 814 相同，除了用一些附加字段扩充 Dial 实例数据结构（DIDS）1202′，以支持 Dial、Dial 组和 Register 的选择性表示之外。具体地说，DIDS 1202′被扩展成包括控制值集表 3200，控制值集表 3200 存储与 DIDS 1202′相对应的配置实体实例（例如，Dial、Dial 组或 Register 实例）的控制值集。另外，DIDS 1202′被扩展成包括控制 Dial 字段 3204，对于与 Dial 组相对应的 DIDS，控制 Dial 字段 3204 包含指向 Dial 组的一个或多个控制 Dial 的每一个的各自指针。对于其它配置实体实例的 DIDS，控制 Dial 字段 3204 是 NULL。

利用记录在数据库 814（或从中导出的任何已装载模拟配置数据库 1404 或硬件配置数据库 1932，2108），接着可以实现各种各样的算法，以便响应相应配置数据库内的控制值信息，有选择地表示模拟或硬件系统的 Dial、Dial

组和 Register 设置。在一个优选实施例中，这样的算法可以作为可以被模拟软件（例如，图 14 的 RTX 1420）或固件（例如，图 19 的系统固件 1930）调用的 API 来实现。在图 33 中例示了和下面将描述 API 可以实现以便在模拟或硬件系统的一群状态下有选择地表示配置实体实例（例如，Dial、Dial 组和 Register）的过程的示范性实施例。

现在参照图 33，图 33 例示了按照本发明，有选择地表示硬件或模拟系统的状态的示范性过程的高级逻辑流程图。所例示的过程参照与感兴趣模拟或硬件系统相联系的配置数据库，以文本和/或图形形式有选择地表示 Dial、Dial 组和 Register 的设置。因此，对于包括其中每一个都含有相关配置数据库的多个集成电路芯片的大型硬件系统，最好在调用所描绘过程的 API 调用指定的参数内对每个这样的数据库执行所例示的过程。本领域的普通技术人员应该认识到，作为逻辑流程图，可以同时地或以与所例示的顺序不同的顺序执行许多所指步骤。

如图 33 所示，该过程从方块 3300 开始，此后转到方块 3302，方块 3302 代表在配置数据库 814、1404、1932 或 2108 的顶层指针阵列 1206 内的每个顶层指针 1250 上的叠代，以处理配置数据库内的每个 Dial 组实例。如果顶层指针阵列 1206 内的所有顶层指针 1250 都被访问过，该过程转到如下所述的方块 3350。但是，如果并非顶层指针阵列 1206 内的所有顶层指针 1250 都被访问过，该过程从方块 3302 转到方块 3304，方块 3304 例示了访问顶层指针阵列 1206 内的下一个顶层指针 1250。

接着在方块 3306 上确定当前顶层指针 1250 是否指向 Dial 组实例。这个确定可以通过，例如，确定当前顶层指针 1250 所指的 DIDS 1202′的父指针 1233 所指的 DDDS 1200 的类型字段 1220 是否指示 DIDS 1202′定义 Dial 组实例作出。如果当前顶层指针 1250 所指的 DIDS 1202′未定义 Dial 组实例，该过程返回到已经描述过的方块 3302。

但是，如果正被考虑的当前 DIDS 1202′的确定义了 Dial 组实例，该过程转到方块 3310。方块 3310 描绘了调用图 33 的过程的 API 调用是否包括指定其表示受到限制的系统内的感兴趣区域的“区域”参数，如果是，实例名字段 1234 是否指示目前 Dial 组实例在感兴趣区域内的确定。例如，可以利用指定设计实体的常规表达式或其它已知技术表达区域参数，以便缩小受到表示的系统的范围。如果当前顶层指针 1250 所指的 DIDS 1202′定义的 Dial 组

实例落在在 API 调用中指定的区域参数的范围之外，该过程返回到已经描述过的方块 3302。另一方面，如果当前顶层指针 1250 所指的 DIDS 1202′定义的 Dial 组实例落在在 API 调用中指定的区域参数（若有的话）的范围之内，该过程转到方块 3312。

方块 3312 描绘了当前 DIDS 1202′的控制值集表 3200 是否指定 Dial 组实例的控制值集。如果是，该过程转到方块 3330，方块 3330 例示了在控制值集表 3200 内指定的控制值集是否指示应该表示 Dial 组实例。这个确定可以通过，例如，确定通过控制 Dial 字段 3204（其 Dial 设置可以按照，例如，图 16A 的过程获得）识别的控制 Dial 的当前设置是否指示应该表示 Dial 组实例作出。如果如方块 3330 所示的确定具有否定结果，该过程返回到方块 3302。如果该确定具有肯定结果，该过程转到如下所述的方块 3334。

返回到方块 3312，如果当前 DIDS 1202′未指定其控制值集表 3200 内的控制值集，在方块 3320 作出 API 调用本身或它的参数之一是否指示与包括或排除没有控制值集的 Dial 组有关的默认政策的另一个确定。如果在方块 3320 上确定 API 调用或它的参数所指的默认政策将表示没有控制值集的 Dial 组实例排除在外，那么，该过程返回到方块 3302。但是，如果在方块 3320 上确定默认政策将表示那些没有控制值集的 Dial 组实例包括在内，该过程转到方块 3334。

方块 3334-3336 代表建立用于表示包含当前 DIDS 1202′定义的 Dial 组实例内的顶层 Dial 的设置的输出文件。尽管在本发明的可替代实施例中，一旦作出在表示中包括 Dial 组实例的确定，属于 Dial 组实例的所有 Dial 实例的设置都可以包括在表示中，但是在本实施例中，根据它们的控制值集（若有的话），仍然可以将属于正被考虑的顶层 Dial 组实例的 Dial 和 Dial 组实例排除在表示之外。为了建立表示输出文件，如方块 3334 所示，该过程首先“挪动”正被考虑的顶层 Dial 组实例的输出指针阵列 1236。正如在方块 3334 上所指的那样，从表示输出文件中删除含有指示不应该表示较低层 Dial 组实例的在其各自 DIDS 1202′中的控制值集表 3200 的顶层 Dial 组实例内的任何 Dial 组实例，以及在它的子树内的任何 Dial 或 Dial 组实例。另外，正如在方块 3334 上进一步所指的那样，给定当前 Dial 设置（如按照图 16A 确定的那样），如果它们各自 DIDS 1202′中的字段 3200 指示应该删除 Dial 实例，那么，从表示输出文件中删除顶层 Dial 实例。在方块 3336 上，将正被考虑的

顶层 Dial 组内和通过在方块 3334 上描绘的顶层 Dial 组实例的 Dial 树的另外“修剪”不排除在表示输出文件之外的任何顶层 Dial 实例包括在表示输出文件内。例如，表示输出文件可以指示每个这样顶层 Dial 实例的实例名和当前设置。

从方块 3334-3336 的描述中应该认识到，包括在唯一顶层 Dial 实例的表示输出文件中是设计选择，在可替代实施例中，非顶层 Dial 实例名和设置也可以包括在表示输出文件中。此外，应该明白，包含或排除没有相关控制值集的 Dial 和 Dial 组的默认政策不仅可以应用于顶层 Dial 组实例（在方块 3320 上），而且可以应用于较低层 Dial 和 Dial 组实例（例如，在方块 3334 上）。在方块 3336 之后，该过程返回到已经描述过的方块 3302。

再次参照方块 3302，响应在方块 3302 上顶层指针阵列 1206 内的所有顶层指针 1250 都被访问过的确定，该过程转到方块 3350，方块 3350 例示了返回到顶层指针阵列 1206 的顶部项目。然后，该过程进入处理不属于任何 Dial 组实例的顶层 Dial 实例、由方块 3352 表示的、穿过顶层指针阵列 1206 的第二次叠代。于是，在方块 3352 上，确定顶层指针阵列 1206 内的所有顶层指针 1250 是否都被访问过。如果是，该过程转到如下所述的方块 3378。但是，如果在方块 3352 上确定并非顶层指针阵列 1206 内的所有顶层指针 1250 都被访问过，该过程从方块 3352 转到方块 3360，方块 3360 例示了访问顶层指针阵列 1206 内的下一个顶层指针 1250。

接着，在方块 3362 上，参照当前顶层指针 1250 所指的 DIDS 1202′的父指针 1233 标识的 DDDS 1200 的类型字段 1220，确定 DIDS 1202′是否定义 Dial 组实例。如果是，不再对 Dial 组实例作进一步处理（在方块 3302 所代表的第一处理循环中已经完成）并且该过程转到方块 3352。但是，如果当前顶层指针 1250 所指的 DIDS 1202′定义了 Dial 或 Register 实例，在方块 3364 上确定实例名字段 1234 是否指示正被研究的实体实例是否落在 API 调用（若有的话）的区域参数指定的感兴趣范围内。这个确定与在方块 3310 上进行的确定相似。如果正被研究的 Dial 或 Register 实例未落在 API 调用的区域参数指定的感兴趣范围内，那么，该过程返回到已经描述过的方块 3352。但是，如果在 API 调用中未指定区域参数或如果实体实例满足区域参数，该过程转到方块 3366。

方块 3366 描绘了在表示输出文件中是否包括与实例有关的信息的确定。

具体地说，如果实体实例含有在它的DIDS 1202'的控制值集表 3200 中指定的控制值集，根据实体实例的设置（按照图 16A 的方法确定），确定应该将当前实体实例排除在表示之外还是包括在表示之中。如果实体实例没有在它的 DIDS 1202'的控制值集表 3200 内指定的控制值集，那么，参照 API 调用或它的参数之一的默认包括或排除政策作出在方块 3366 上描绘的确定。如果在方块 3366 上确定与实体实例有关的信息不应该包括在表示输出文件内，那么，不将与实体实例有关的信息包括在表示输出文件中，该过程简单地返回到方块 3352。但是，如果在方块 3366 上确定将与实体实例有关的信息包括在表示输出文件内，该过程转到方块 3370，方块 3370 描绘了将实体实例的实例名（来自 DIDS 1202'的实例名字段 1234）和实体实例的设置包括在例示输出文件内。此后，该过程返回到已经描述过的方块 3352。

响应在方块 3352 上顶层指针阵列 1206 内的所有顶层指针 1250 都被访问过的确定，该过程从方块 3352 转到方块 3378。方块 3378 描绘了通过，例如，打印 Dial 组、Dial 和 Register 实例名和相关设置的硬拷贝报告或图形显示，将表示输出文件展示给用户。在一个特定优选实施例中，在允许用户图形地和直观地导航模拟或硬件系统和检查感兴趣的特定设置的图形用户界面中将表示输出文件展示给用户。在方块 3378 之后，在方块 3380 上终止描绘在图 33 中的过程。

现在参照图 34A，图 34A 描绘了按照本发明展示模拟或硬件系统和它们的部件的图形表示的示范性图形用户界面（GUI）3400。正如在现有技术中众所周知的那样，GUI 3400 通常由软件（例如，实现图 33 的过程的 API 或分立程序）表示在数据表示系统，譬如，图 1 的数据处理系统 6 或图 19 的工作站计算机 1904 的显示设备内。能够使数据表示系统根据底层配置数据库（例如，模拟配置数据库 1404 或硬件配置数据库 1932）的内容和可选地一个或多个其它数据资源（例如，模拟模型 1400 或模拟可执行模型 816）的内容表示 GUI 3400 的任何软件在这里都被称为“图形界面软件”。

如图所示，GUI 3400 包括受诸如鼠标之类的用户输入设备控制的传统 GUI 部件，譬如，窗口 3402、控制按键 3404，下拉菜单 3406、和光标 3408。尽管未示出，GUI 3400 当然可以包括其它附加传统或非传统 GUI 特征件，譬如，工具栏、滚动条等，以方便用户与 GUI 3400 交互和操作 GUI 3400。

窗口 3402 包含给出模拟系统（或它的部件）的图形表示的图文框 3410。

在所描绘的例子中，图形表示在图文框 3410 中的系统几乎与例示在图 11B 中和如上所述的系统相同，因此，应用相似的标号来标识相似的特征件。也就是说，正如配置数据库内的上述或可替代数据结构所指的那样，该系统包括分层包含设计实体实例 FBC: FBC 1124 和 L2: L2 1126 的顶层设计实体实例（名为 TOP: TOP）1122。设计实体实例 1124 又包含设计实体 X（每一个都包含设计实体 Y 的两个实例 1138）的实例 1136a，1136b 和设计实体 Z 1132。并且，设计实体实例 1126 包含设计实体 L 的实例 1152a，1152b。正如上面详述的那样，无论属于模拟系统还是硬件系统，设计实体实例最初都通过一个或多个 HDL 文件定义。本领域的普通技术人员应该认识到，在至少一个实施例中，图形界面软件参照描述设计实体分层结构的 m 路树，生成模拟系统内的设计实体的表示。正如上面参照图 8 所述的那样，这种数据结构可以包括在模拟数据库内。

配置数据库进一步将各种各样配置实体实例（例如，Dial、Dial 组和/或 Register）与各种各样 HDL-定义设计实体相联系。在本例中，配置数据库将具有名称“H”的 RGDial（即，只读 Dial 组）实例 1164 与设计实体实例 1122 相联系，将 GDial 实例 1160 与设计实体实例 1124 相联系，和将 GDial 实例 1162 与设计实体实例 1126 相联系等。在一个优选实施例中，通过在图元的边界内表示设计实体实例，但在图元的边界外表示相关设计实体实例包含的任何较低层设计实体实例地显示表示配置实体实例的图元（例如，长方形），在 GUI 3400 内图形表示配置实体实例和它的相关设计实体实例之间的联系。当然，在其它实施例中，其它技术可以应用于表示设计实体实例和配置实体实例之间的联系。

GUI 3400 最好提供允许用户提高它们的系统可视性的许多工具和/或选项。例如，在至少一种模式中，GUI 3400 最好与所选或所有配置实体实例的图形表示相联系地表示那些配置实体实例的设置。例如，在如图 34A 所示的例子中，用户选择了只表示 IDial 实例，譬如，IDial 实例 1132 和 1150 的设置的表示模式。这样的表示也可以以用户用光标 3408 图形选择感兴趣配置实体实例为条件。

在至少一个优选实施例中，GUI 3400 进一步包括允许用户使数据表示系统只表示系统设计分层结构的所选层的深度控制。深度控制可以通过，例如，一个或多个下拉菜单 3406 来访问和调整。在一个优选实施例中，深度控制允

许用户指定一次表示的设计分层结构的所需层数。允许值包括大于等于 2 的整数和使设计分层结构的所有层同时得到表示的“A11”。例如，图 34A 描绘了深度控制具有“A11”的值的示范性表示情形，而图 34B 例示了同一系统深度控制具有 2 的值的示范性表示情形。应该注意到，在如图 34B 所示的示范性实施例中，表示了与设计分层结构的所有层而不是最低显示层相联系的配置实体实例，在这种情况下，它是单个配置实体实例 1164。因此，用户可以控制同时表示的信息量，这极大地方便了对含有多个芯片和/或设计分层结构的多个层的复杂系统的结构和设置的理解。

GUI 3400 最好允许用户图形地和直观地改变设计分层结构的表示层，以便在设计分层结构内逻辑地上升和下降。例如，在一个实施例中，GUI 3400 响应用户利用光标 3408 对当前视图内的较低层设计实体内的位置的选择，从设计分层结构的较低层的角度表示系统的设计分层结构。因此，图 34C 描绘了响应用户利用光标 3408 对图 34B 的设计实体实例内的位置的选择表示的视图。通过与图 34A 作比较可以再次注意到，如图 34C 所示的视图只表示了设计分层结构的两个层，以及与设计分层结构的所有层而不是最低显示层相联系的配置实体实例（即，GDial 1160 和 LDial 1130）。用户可以类似地利用光标 3408 选择设计实体的当前最高显示层之外图文框 3410 之内的位置，使设计分层结构的表示视图上升（如有可能）。因此，响应图 34C 的视图中设计实体实例 1124 的边界之外图文框 3410 之内的位置的用户选择，再次显示在图 34B 中给出的视图。

在至少一个实施例中，GUI 3400 还允许用户有选择地暴露配置实体分层结构的其它层。例如，再次参照图 34C，在一个优选实施例中，用户选择父配置实体实例（例如，GDial 1160）的输出 3420 将使处在配置实体分层结构的下一个较低层上的所有子配置实体实例得到表示。因此，如图 34D 所示，用户利用光标 3408 在表示在图 34C 中的视图中选择 GDial 1160 的 Dial 输出 3420 将使 GUI 3400 展示处在配置实体分层结构的下一个较低层上的 GDial 1160 的所有子配置实体实例。应该注意到，由于配置实体实例的父子关系可以推广到设计分层结构的多个层上，也如图 34D 所示，配置实体分层结构的下一个较低层的表示可能需要显示设计分层结构的多个其它层。当用户试图调试设置成错误值的 CDial 或 RDial，因此，希望观看它的子 Dial 的设置时，有选择地暴露配置实体分层结构的其它层的能力尤其有用。

正如上面在图 33 的描述中简要提及的那样，在本发明的一些优选实施例中，GUI 3400 进一步支持相关控制使数据表示系统根据如通过它们的控制值集（若有的话）和当前设置确定的那样的它们的相关性限制配置实体实例的表示方式。在一个优选实施例中，相关性控制至少存在与三种表示模式 - Full view（全视图）、Selective view（选择视图）和 Mixed view（混合视图）相对应的三种设置。这三种相互独立，但可以与以前的表示模式结合在一起应用的表示模式分别例示在图 34A、34E 和 34F 中。

再次参照图 34A，通过 GUI 3400 表示的系统视图与它们在系统的配置数据库内的值或控制值无关地描绘了配置实体实例。因此，关于配置实体实例的表示，图 34A 例示了 Full view。

现在转到图 34E，应该注意到，配置实体实例 1134、1140a0、1140b1、3414、3416 和 1154b 已根据在配置数据库中指定的相关控制值和它们的当前设置有选择地从按照描绘在图 33 中的过程通过 GUI 3400 的表示中删除了。因此，图 34E 描绘了从表示中滤出根据它们的当前设置不那么可能与用户有关的配置实体实例，以便允许用户把注意力集中在最有可能适合理解系统状态的那些配置实体实例上的 Selective view。

最后参照图 34F，图 34F 例示了以与其它配置实体实例不同的图形方式表示与用户的相关性较低的配置实体实例的系统的 Mixed view。在如图 34F 所示的特定实施例中，通过利用不同线权重（例如，虚线例示）显示相关性较低的配置实体实例，给出相关性较高的配置实体实例和相关性较低的配置实体实例之间的图形差异。当然，在其它实施例中，可以利用不同颜色、亮度、尺寸、形状和/或诸如表示的信息量之类的其它图形特性，可替代地或另外给出表示配置实体实例的图元之间的差异。例如，可以表示相关性较高的配置实体实例的设置，而可以从表示中去掉相关性较低的配置实体实例的设置。因此，Mixed view 向用户表示了给定所选深度控制设置显示的所有配置实体实例，但以不同图形方式表示了相关性较低的配置实体实例，以便用户能够可电地将它们与更适合于理解系统状态的配置实体实例区分开。

虽然像参照优选实施例所述的那样已经具体显示了本发明，但本领域的普通技术人员应该明白，可以在形式和细节上对它们作出各种各样的改变，而不偏离本发明的精神和范围。例如，应该认识到，本文公开的概念可以推广到或修改成应用于除了本文公开的那些之外的其它类型配置实体。另外，

本领域的普通技术人员还应该明白，各种各样规则的任何一种都可以用于确定表示哪些配置实体，这意味着本文描述的特定语法和表示只是示范性的，而不是穷举性的。

并且，尽管本发明的这些方面是针对执行支配本发明功能的软件的计算机系统加以描述的，但应该明白，可替代地，本发明也可以作为供数据处理系统使用的程序产品来实现。定义本发明功能的程序可以通过包括（不限于这些）不可重写存储媒体（例如，CD-ROM（只读光盘存储器））、和可重写存储媒体（例如，软盘或硬盘驱动器）的各种各样信号载送媒体、和诸如数字和模拟网络之类的通信媒体传送到数据处理系统。因此，应该明白，这样的信号载送媒体当携带或编码支配本发明功能的计算机可读指令时，代表本发明的可替代实施例。

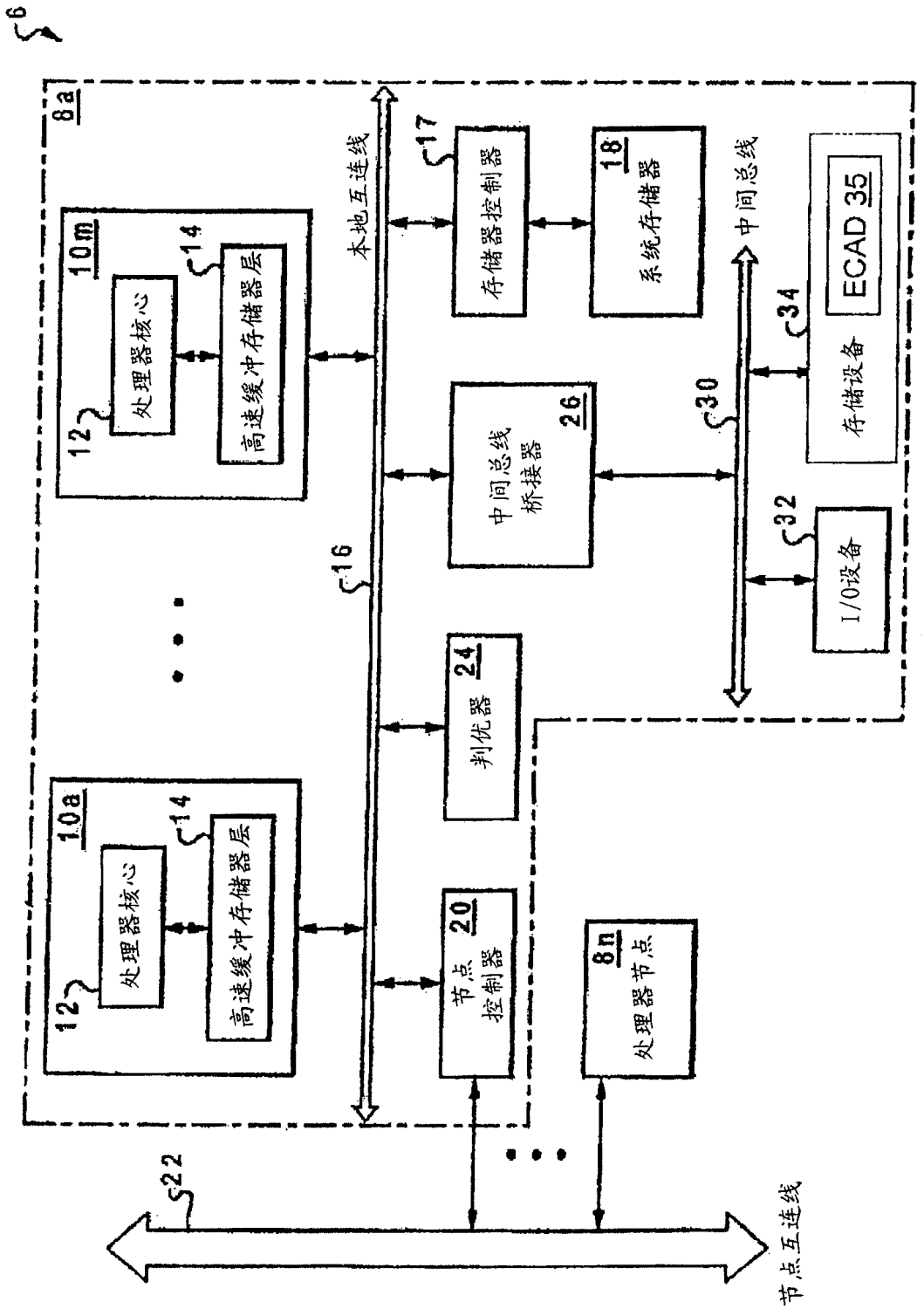

6
8a
10m
12
处理器核心
14
高速缓冲存储器层
本地互连线
17
存储器控制器
18
系统存储器
中间总线
ECAD 35
存储设备
34
30
中间总线桥接器
26
16
32
I/O设备
24
判优器
10a
12
处理器核心
14
高速缓冲存储器层
节点控制器
20
8n
处理器节点
22
节点互连线

图 1

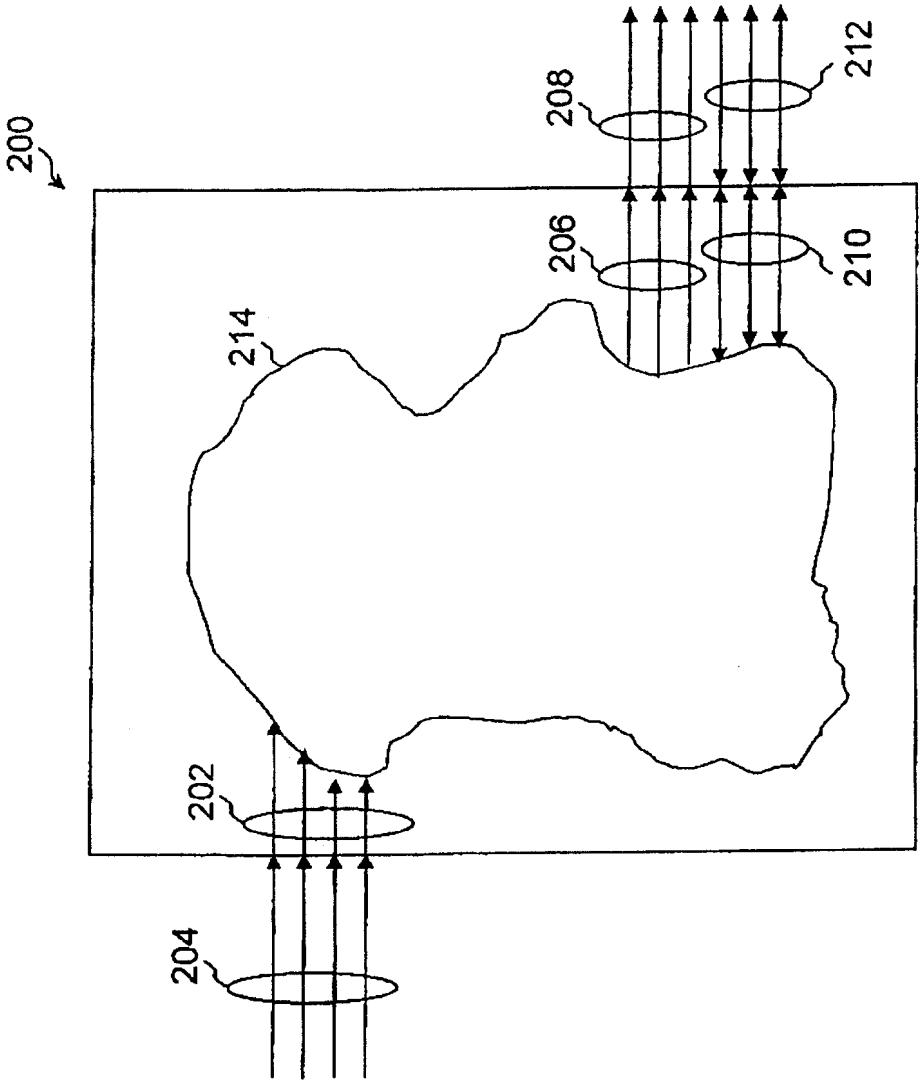


图 2

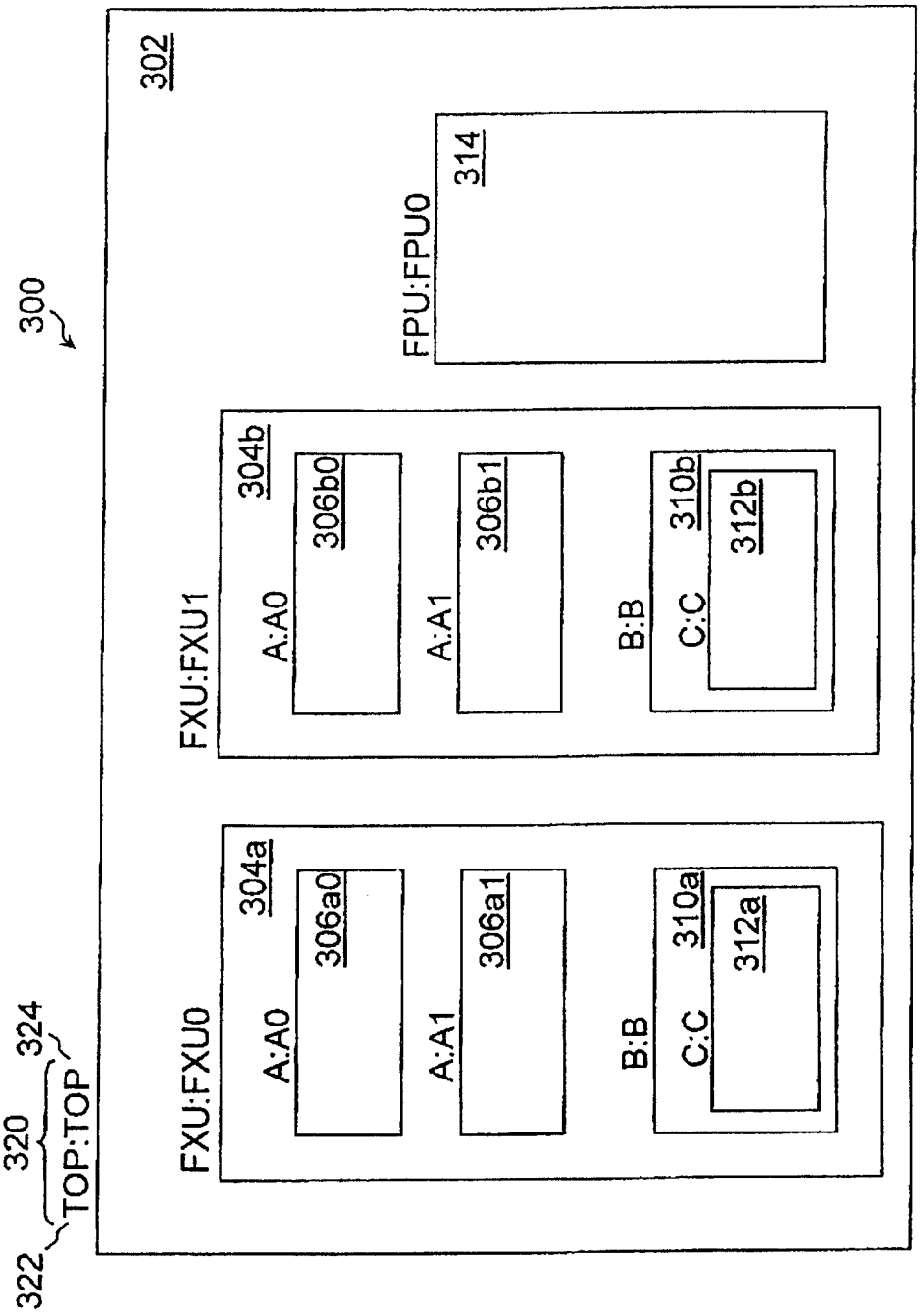

300
302
322
320
324
TOP:TOP
FXU:FXU0
304a
A:A0
306a0
A:A1
306a1
B:B
C:C
310a
312a
FXU:FXU1
304b
A:A0
306b0
A:A1
306b1
B:B
C:C
310b
312b
FPU:FPU0
314

图 3

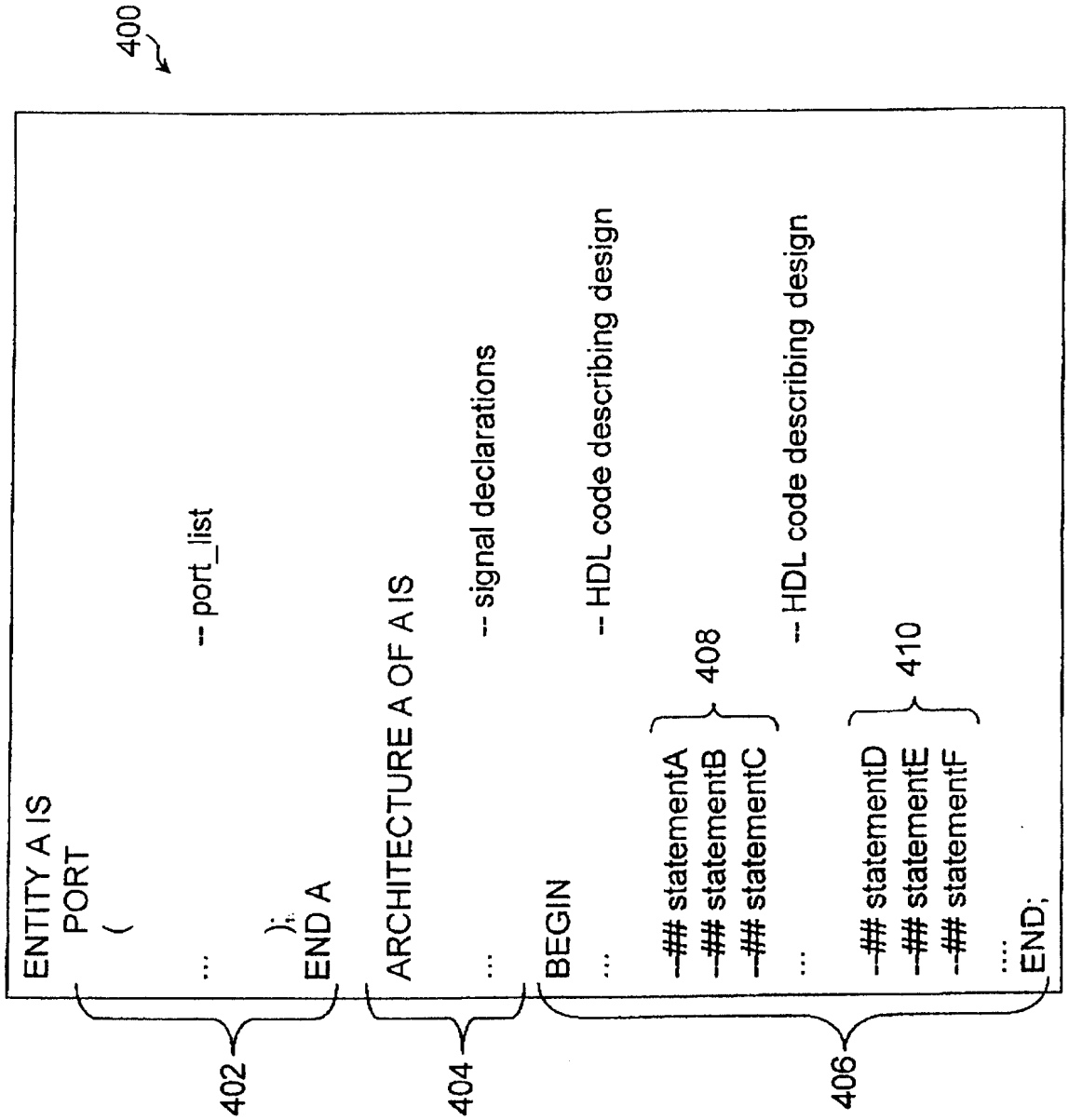

400
ENTITY A IS
PORT
(
-- port_list
);
END A
ARCHITECTURE A OF A IS
-- signal declarations
BEGIN
-- HDL code describing design
statementA
statementB
statementC
408
-- HDL code describing design
statementD
statementE
statementF
410
END;
402
404
406

图 4A

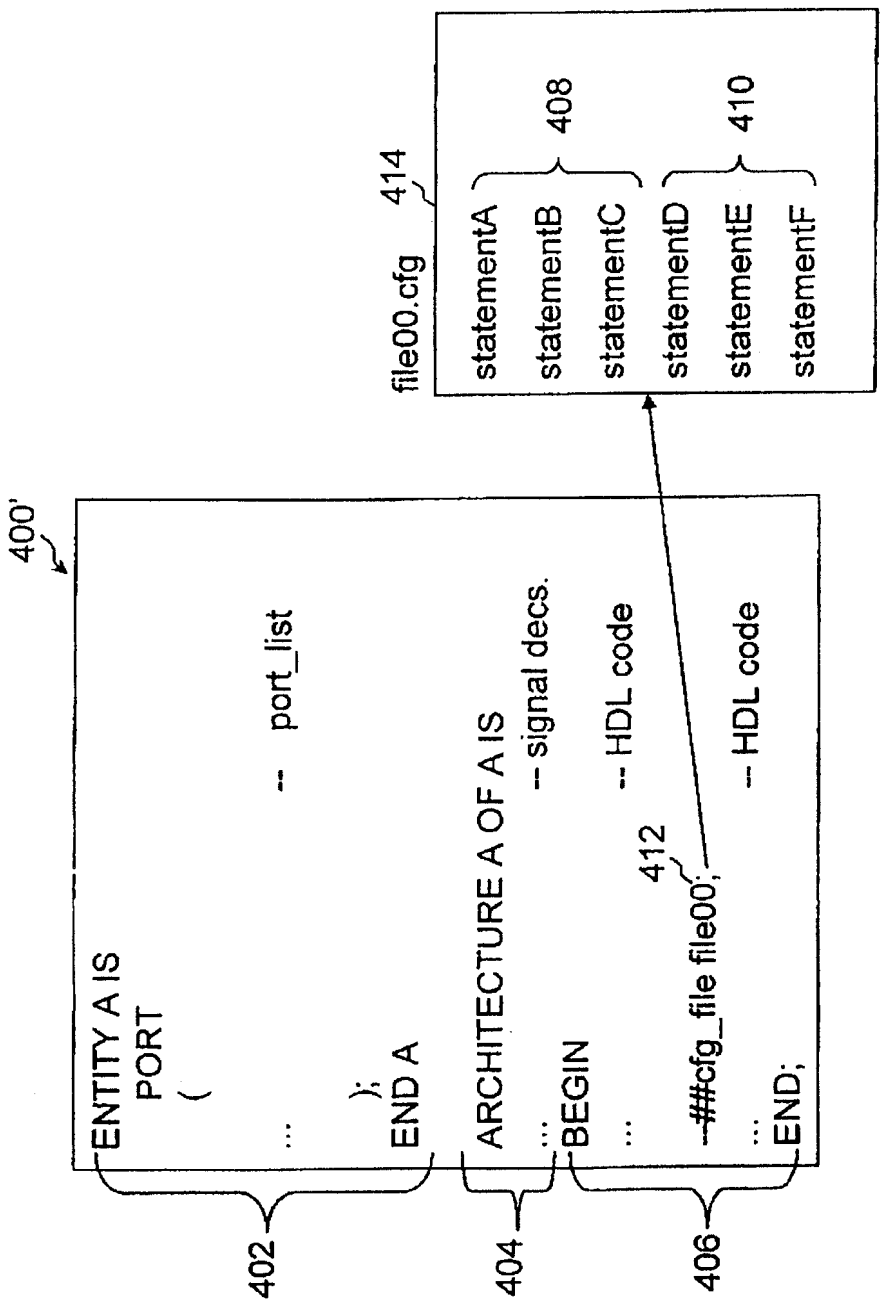

400'
ENTITY A IS
PORT
(
...
-- port_list
);
END A
ARCHITECTURE A OF A IS
-- signal decs.
...
BEGIN
...
-- HDL code
412
##cfg_file file00;
...
-- HDL code
END;
402
404
406
file00.cfg
414
statementA
statementB
statementC
statementD
statementE
statementF
408
410

图 4B

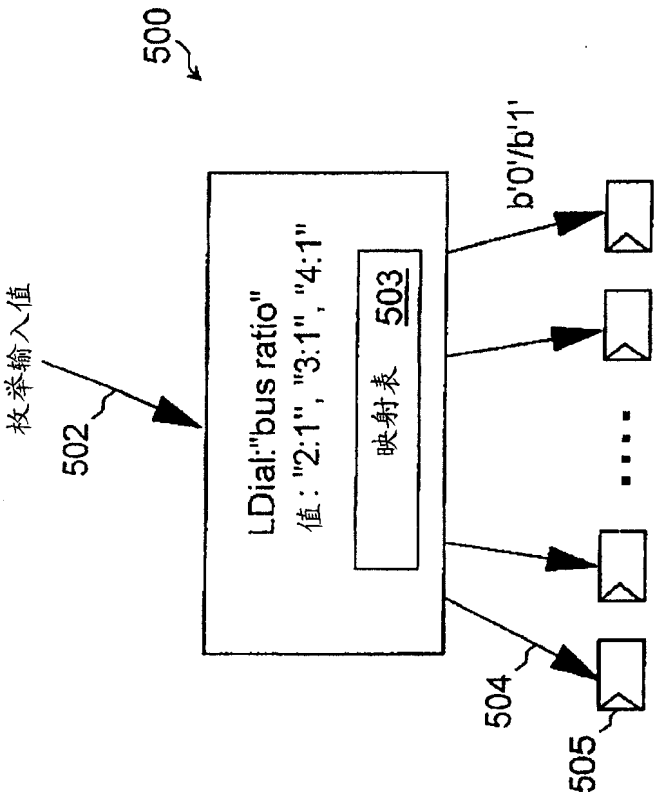


图 5A

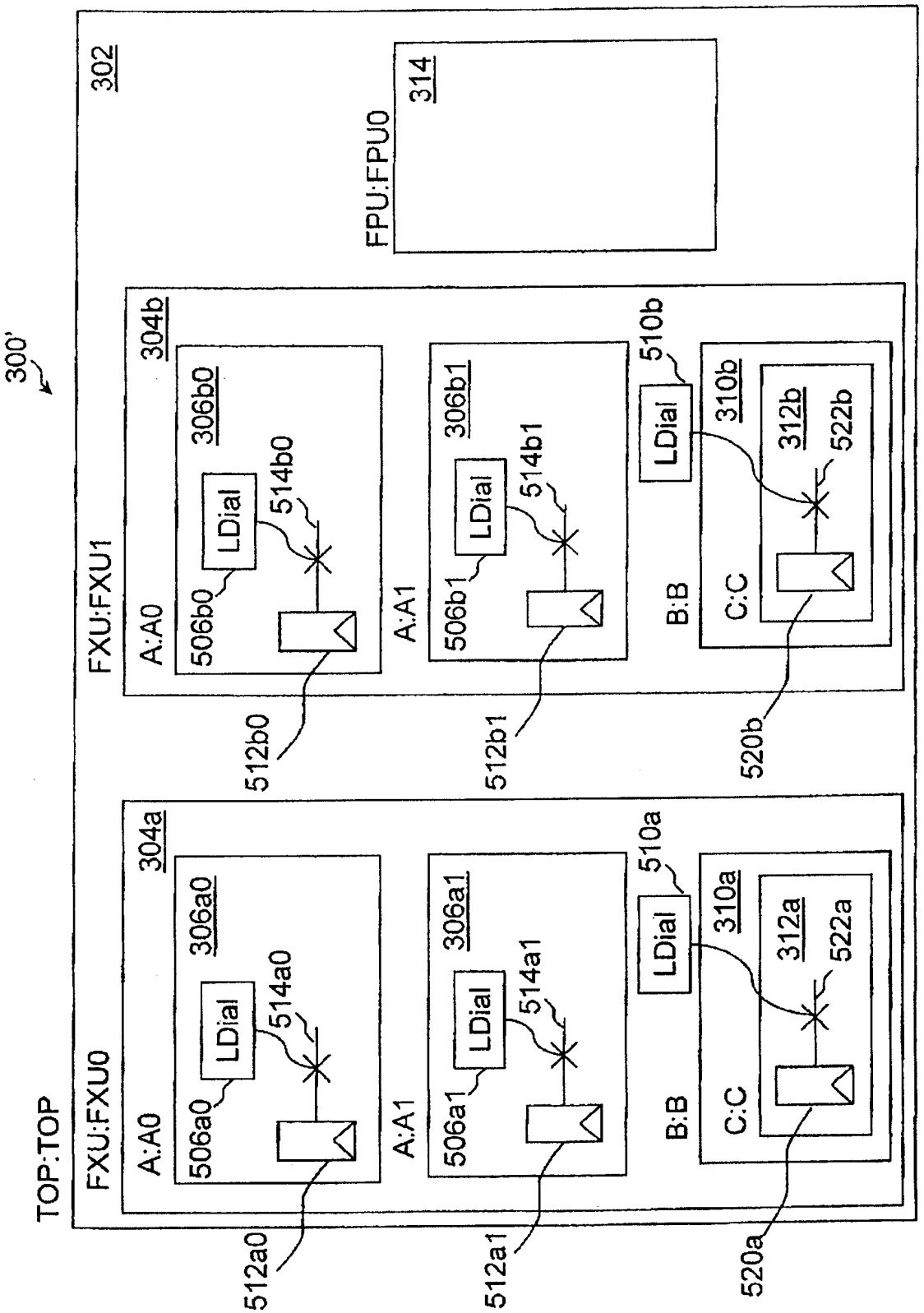

300'
TOP:TOP
FXU:FXU0
302
FXU:FXU1
FPU:FPU0
314
304a
304b
A:A0
A:A1
B:B
C:C
306a0
306a1
306b0
306b1
LDial
506a0
506a1
506b0
506b1
514a0
514a1
514b0
514b1
510a
510b
310a
310b
312a
312b
522a
522b
512a0
512a1
512b0
512b1
520a
520b

图 5B

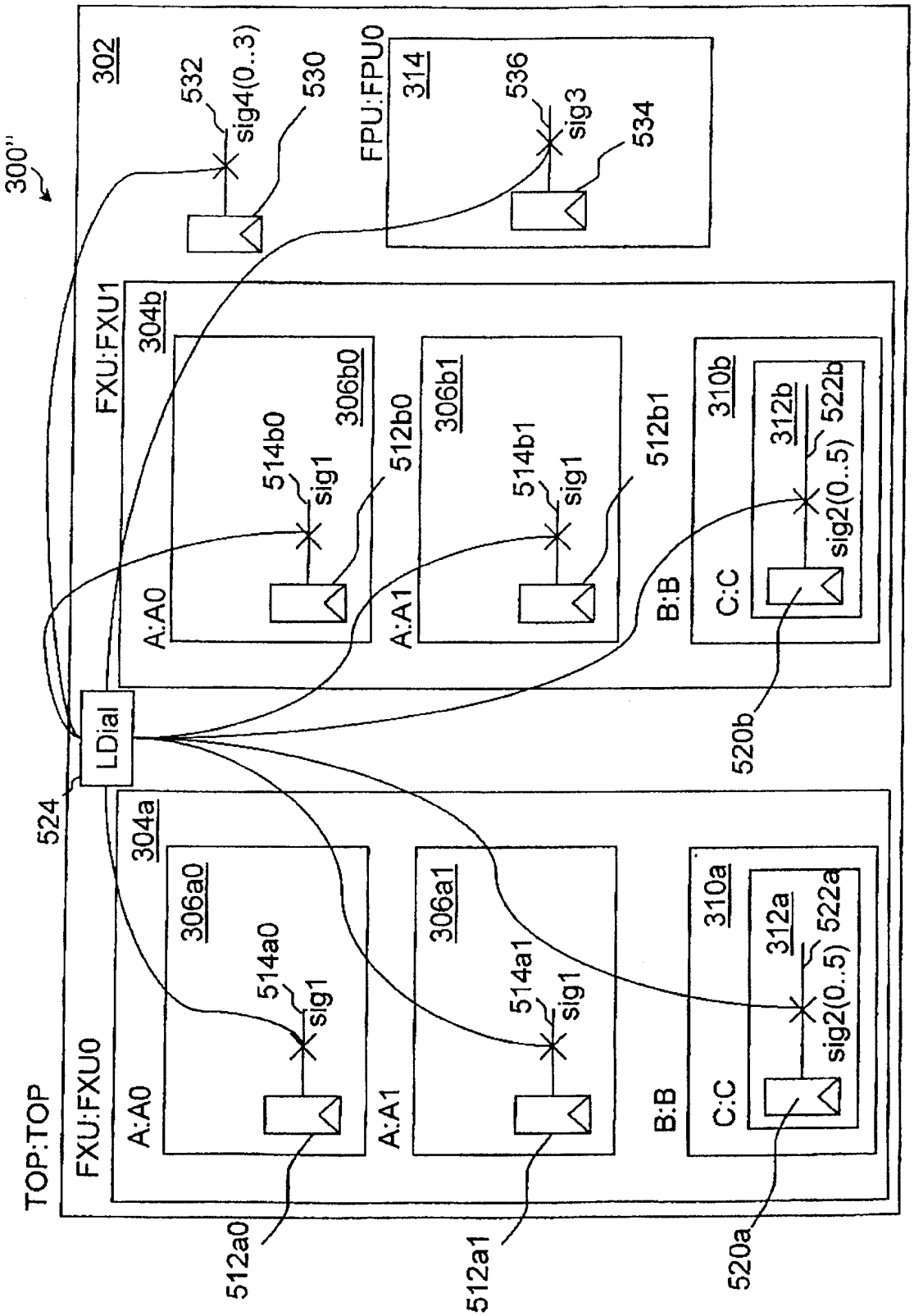

300''
302
532
sig4(0..3)
530
FPU:FPU0
314
536
sig3
534
FXU:FXU1
304b
306b0
514b0
sig1
512b0
306b1
514b1
sig1
512b1
310b
312b
522b
sig2(0..5)
A:A0
A:A1
B:B
C:C
LDial
524
520b
304a
306a0
514a0
sig1
306a1
514a1
sig1
310a
312a
522a
sig2(0..5)
TOP:TOP
FXU:FXU0
A:A0
A:A1
B:B
C:C
512a0
512a1
520a

图 5C

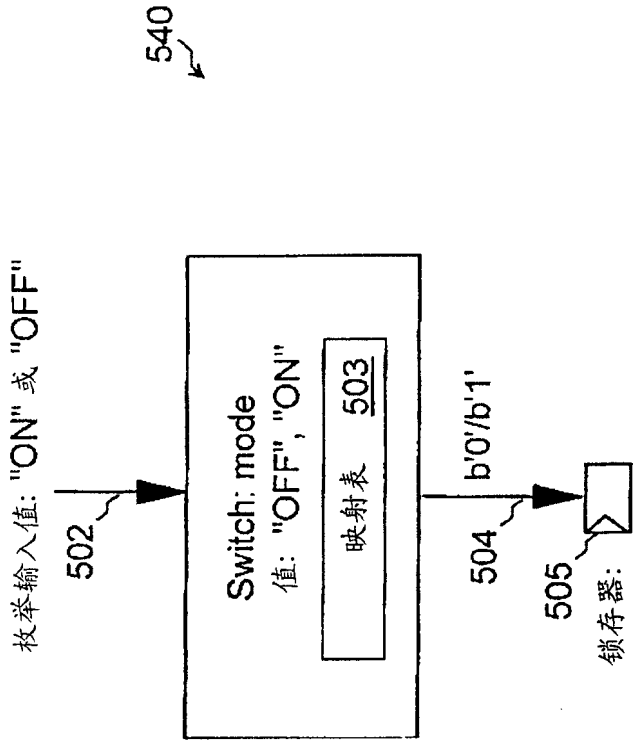

540
枚举输入值: "ON" 或 "OFF"
502
Switch: mode
值: "OFF", "ON"
映射表 503
b'0'/b'1'
504
505
锁存器:

图 5D

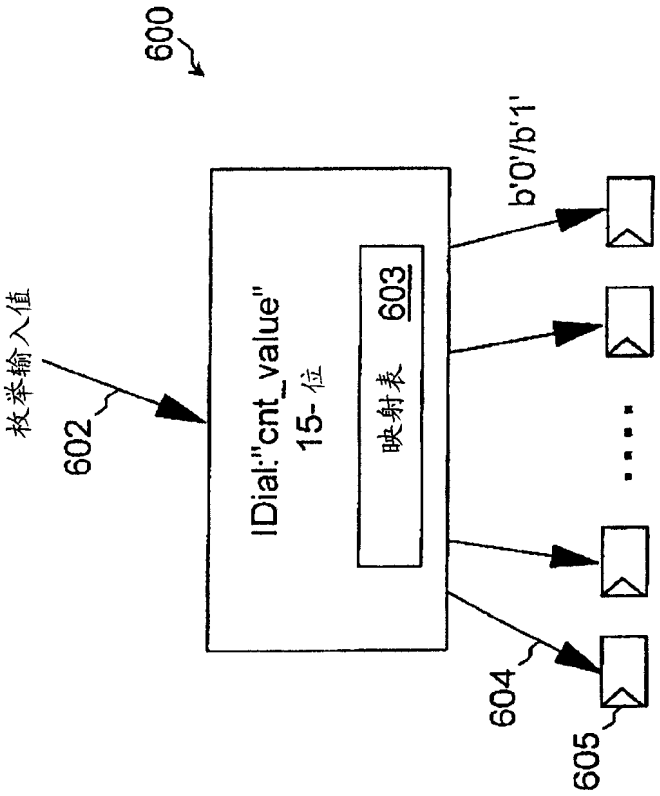


图 6A

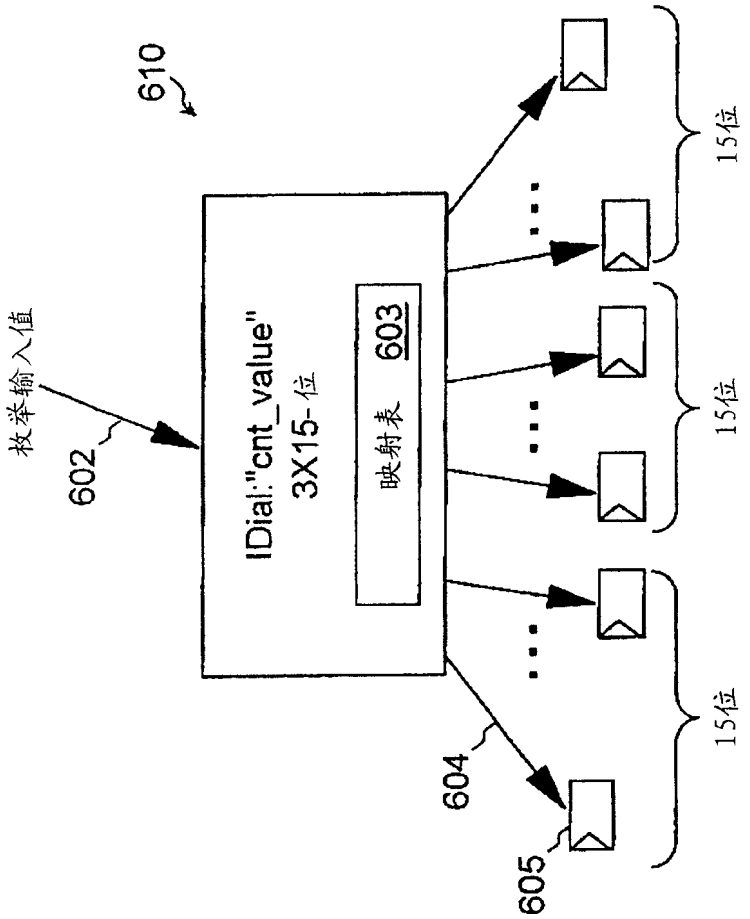

610
枚举输入值
602
IDial:"cnt_value"
3X15-位
映射表 603
604
605
15位
15位
15位

图 6B

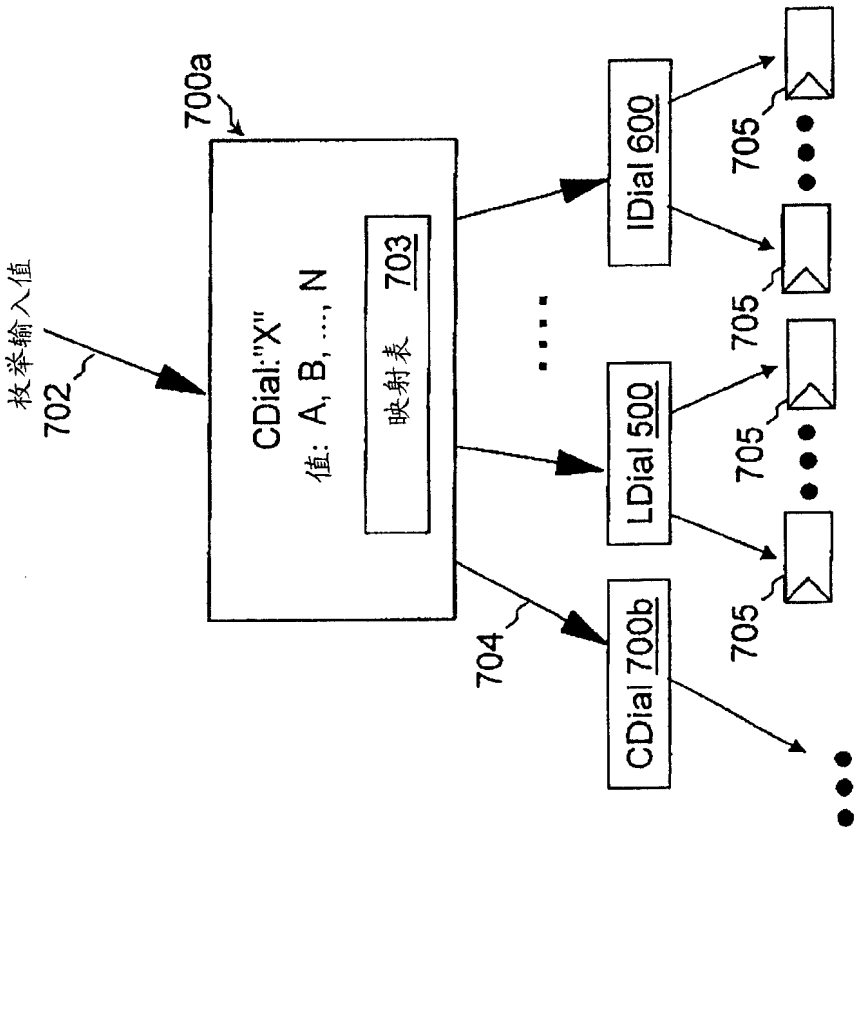

枚举输入值
702
700a
CDial:"X"
值: A, B, ..., N
映射表 703
704
CDial 700b
LDial 500
IDial 600
705

图 7A

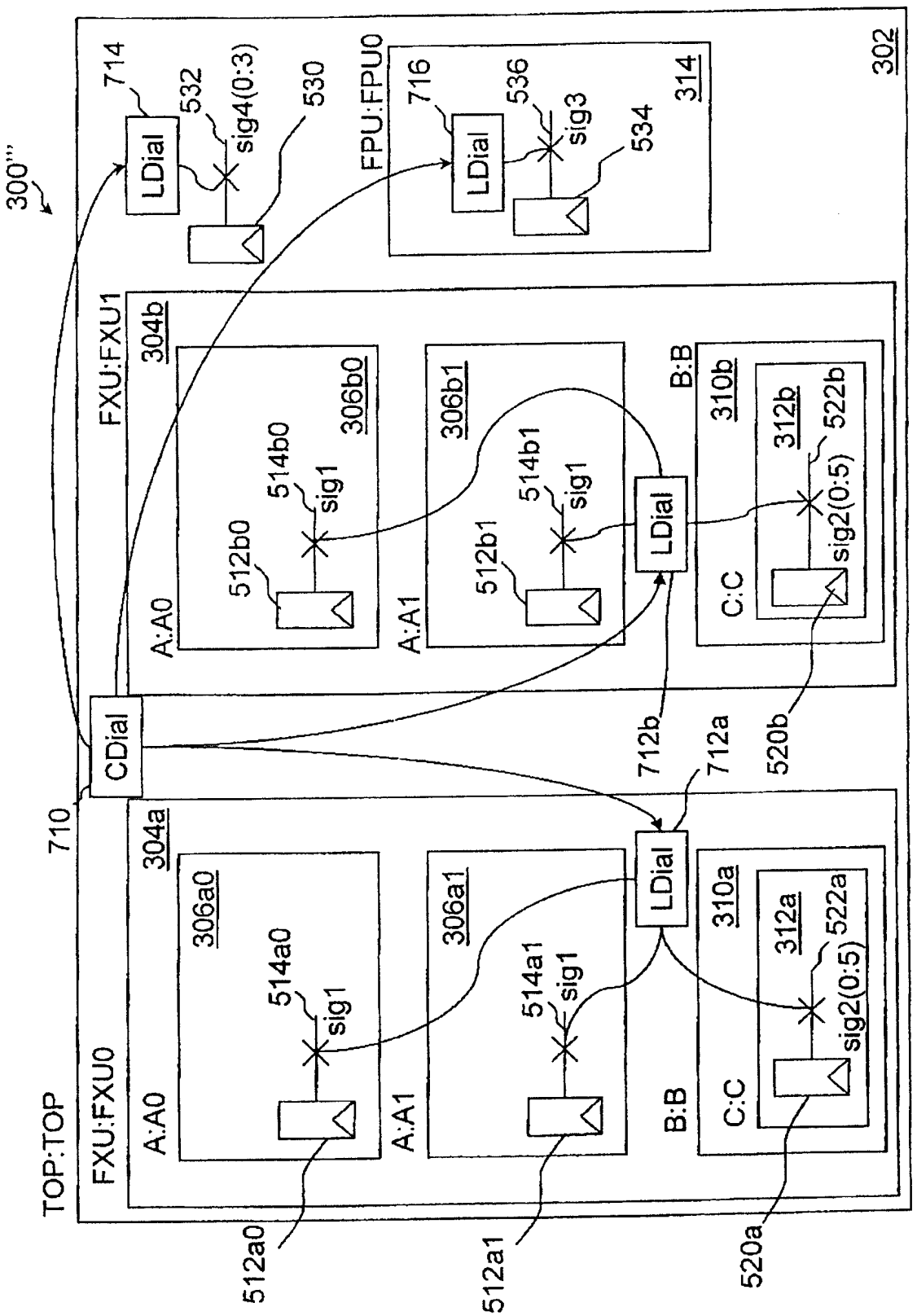

300'''
302
TOP:TOP
FXU:FXU0
A:A0
A:A1
B:B
C:C
304a
306a0
306a1
310a
312a
514a0
514a1
sig1
sig2(0:5)
522a
512a0
512a1
520a
710
CDial
LDial
712a
712b
520b
FXU:FXU1
304b
306b0
306b1
310b
312b
514b0
514b1
512b0
512b1
522b
714
532
sig4(0:3)
530
FPU:FPU0
716
536
sig3
534
314

图 7B

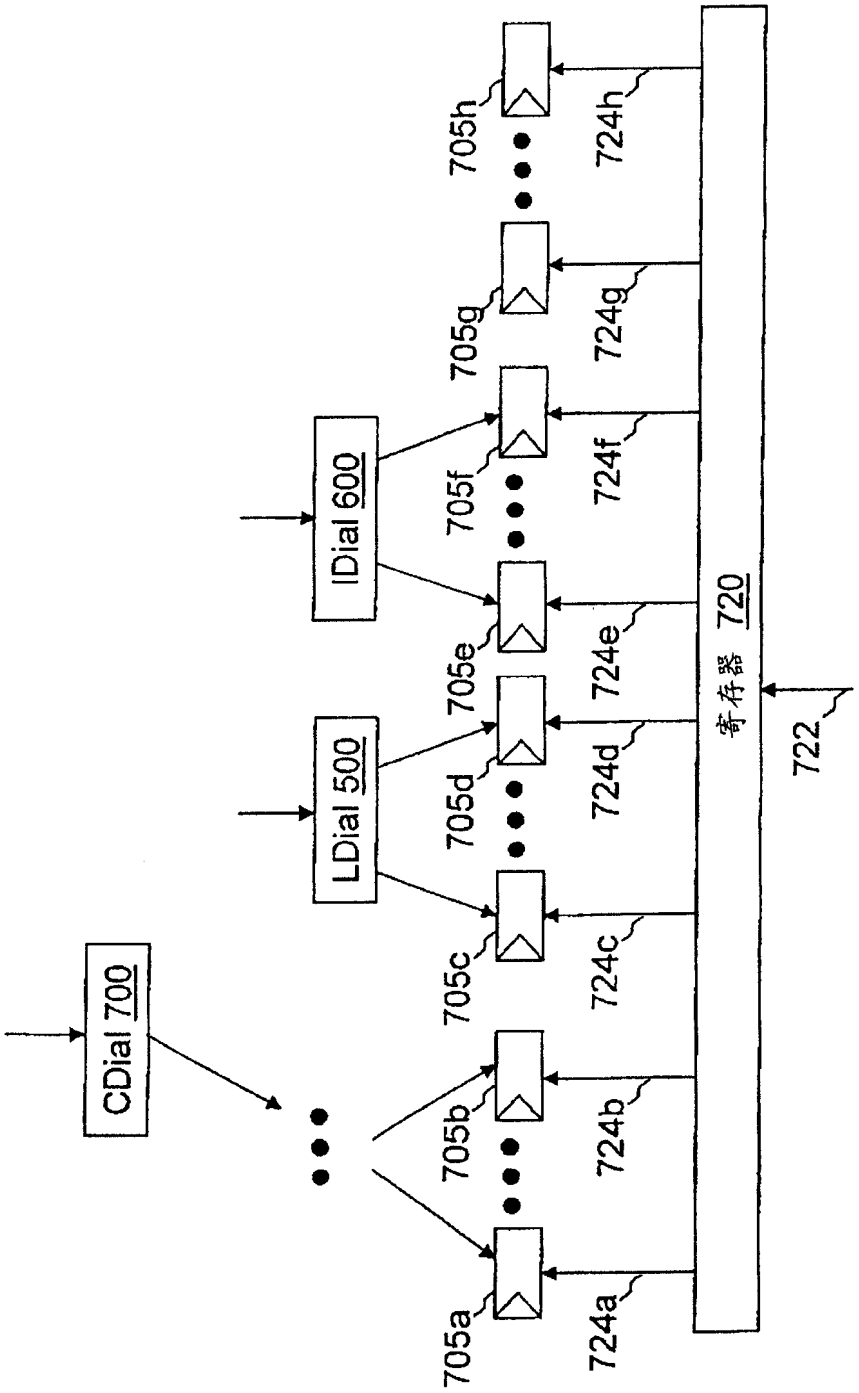

CDial 700
LDial 500
IDial 600
705a
705b
705c
705d
705e
705f
705g
705h
724a
724b
724c
724d
724e
724f
724g
724h
寄存器 720
722

图 7C

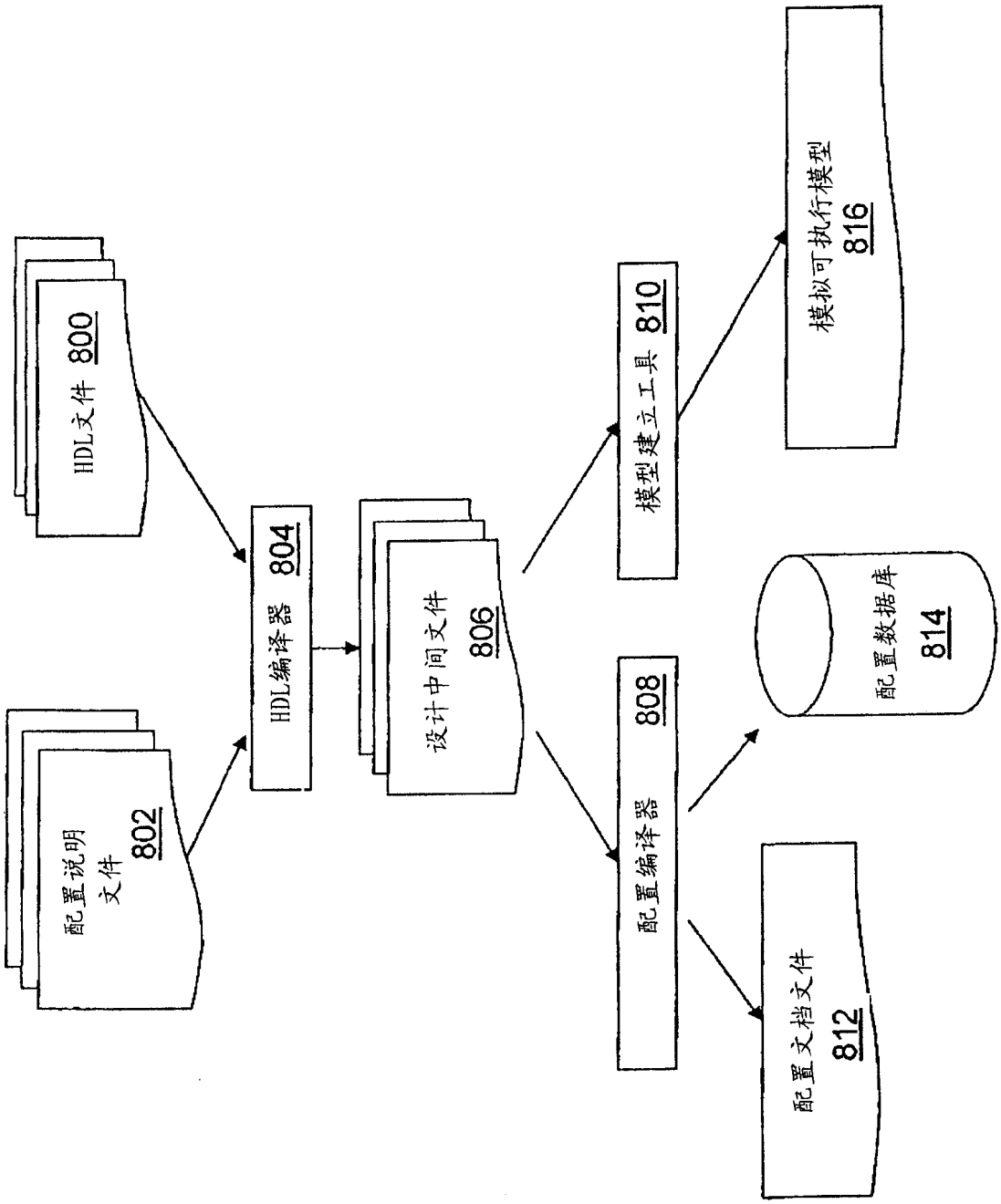

HDL文件 800
配置说明文件 802
HDL编译器 804
设计中间文件 806
模型建立工具 810
模拟可执行模型 816
配置编译器 808
配置数据库 814
配置文档文件 812

图 8

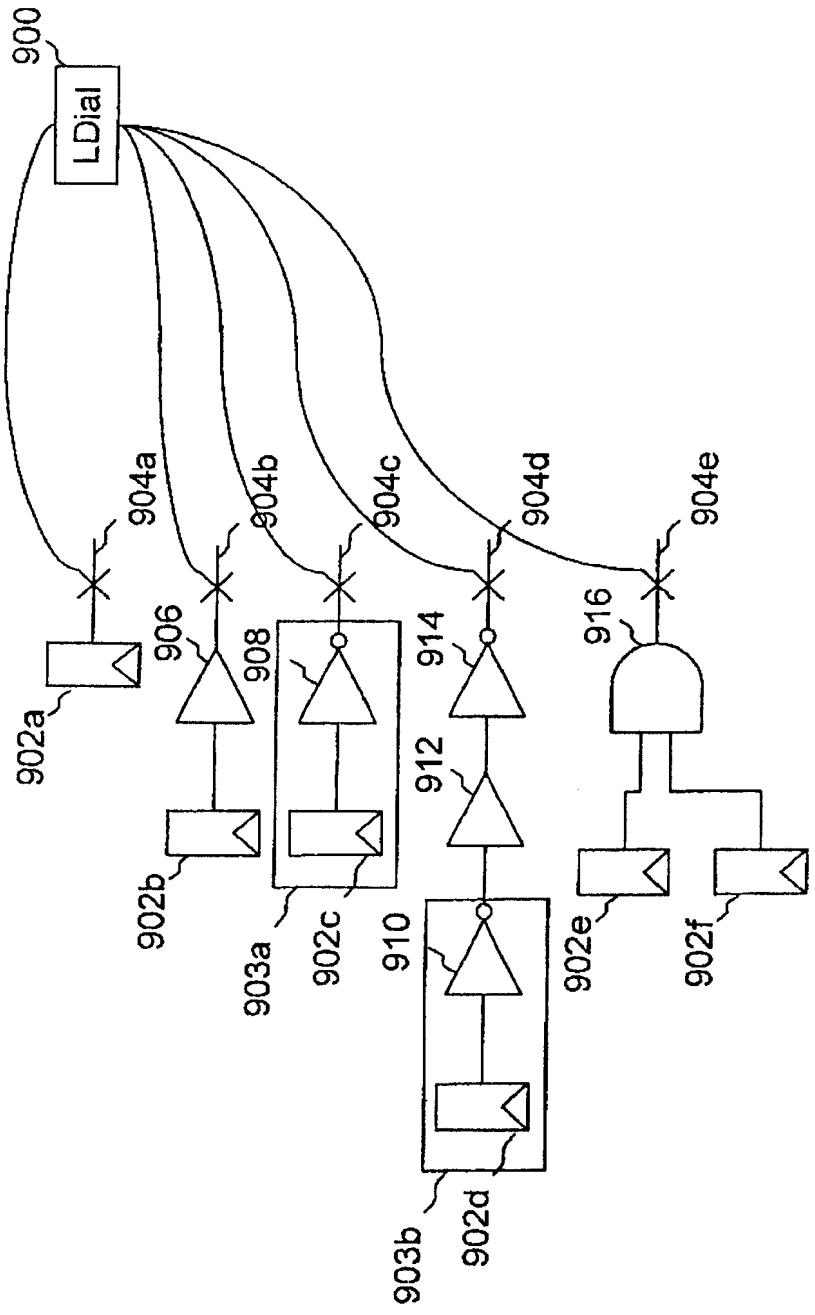

900
LDial
904a
904b
904c
904d
904e
906
908
914
916
912
902a
902b
903a
902c
910
902e
902f
903b
902d

图 9A

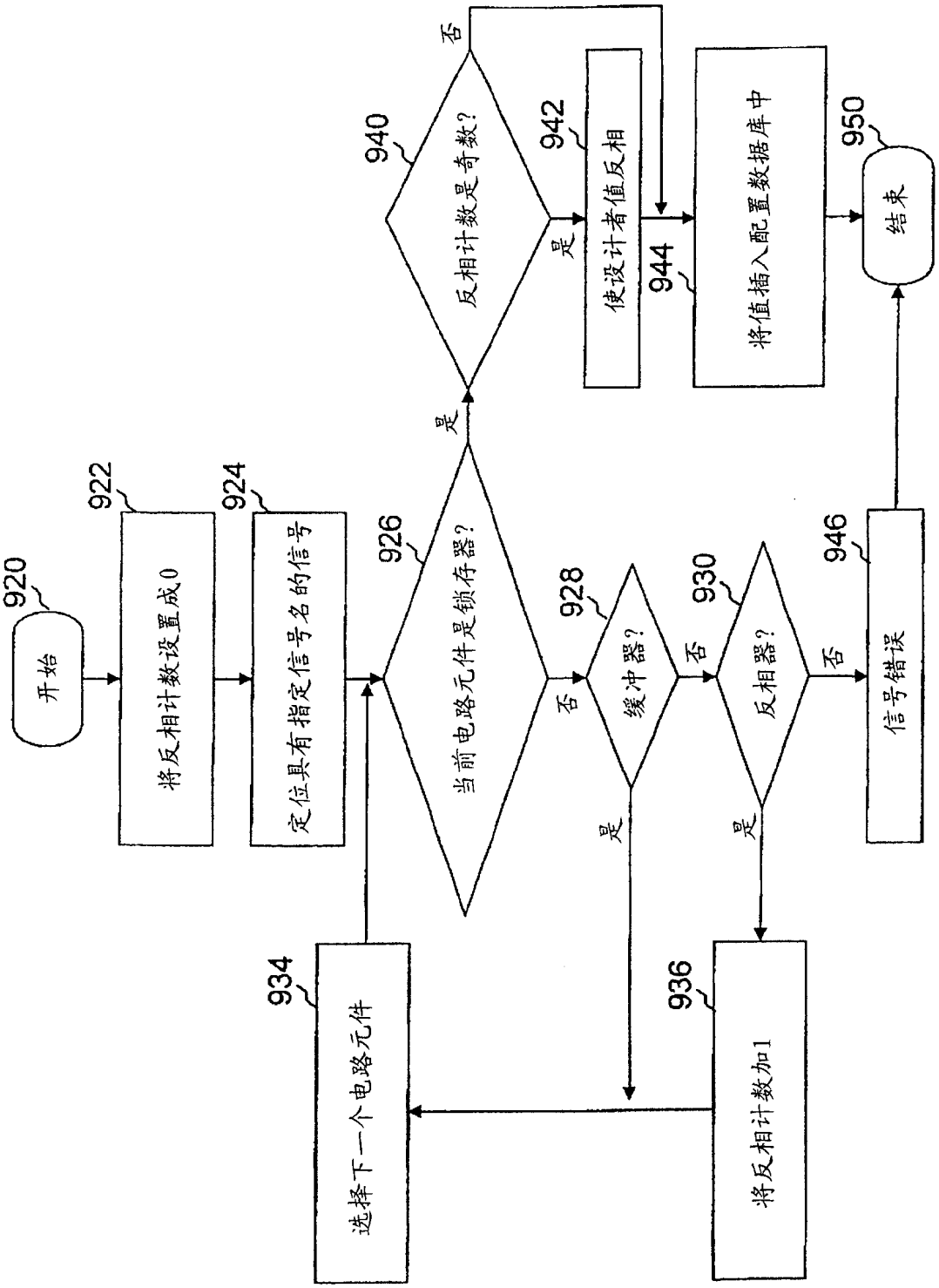

开始
920
将反相计数设置成0
922
定位具有指定信号名的信号
924
当前电路元件是锁存器?
926
缓冲器?
928
反相器?
930
信号错误
946
选择下一个电路元件
934
将反相计数加1
936
反相计数是奇数?
940
使设计者值反相
942
将值插入配置数据库中
944
结束
950
是
否

图 9B

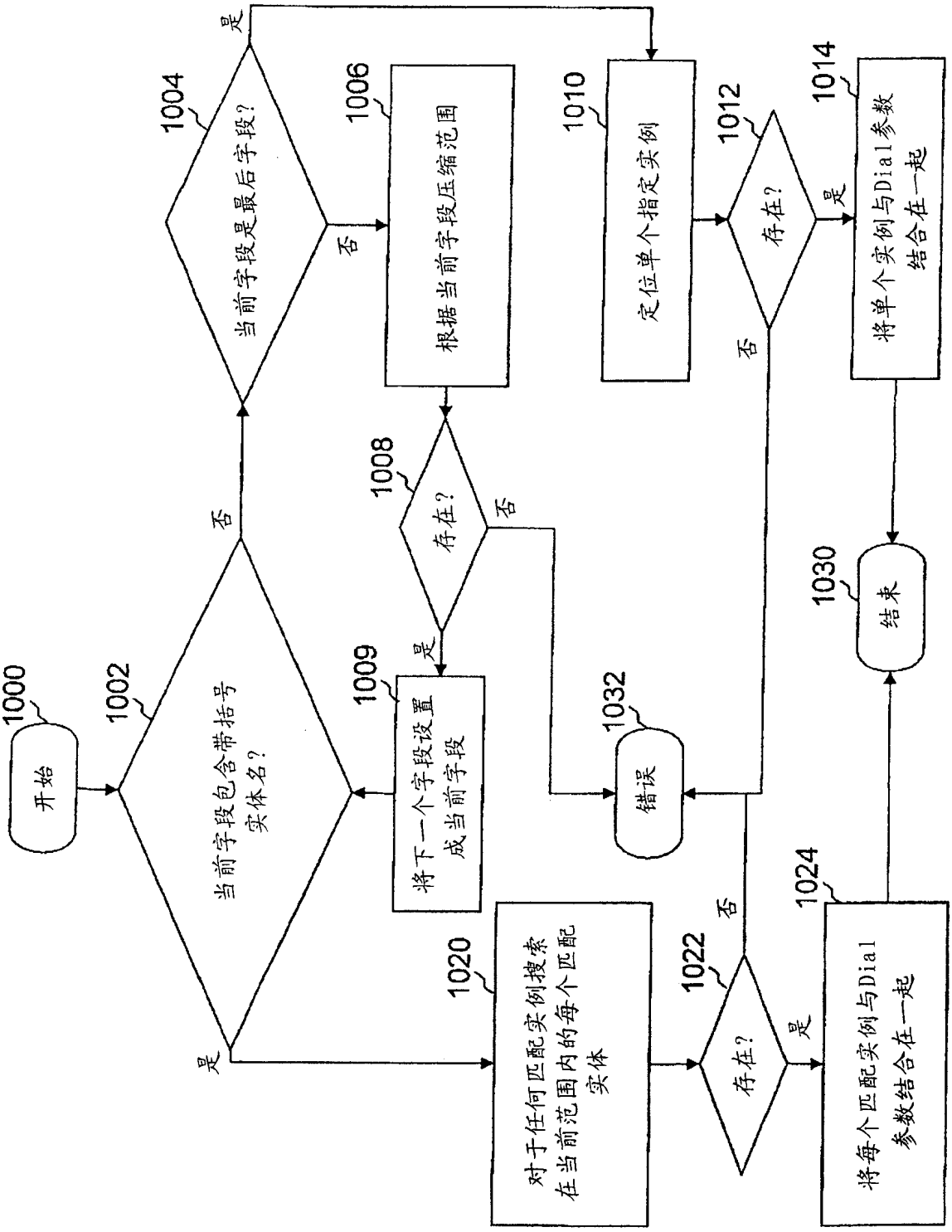

1000
开始
1002
当前字段包含带括号实体名?
是
否
1004
当前字段是最后字段?
是
否
1006
根据当前字段压缩范围
1008
存在?
是
否
1009
将下一个字段设置成当前字段
1010
定位单个指定实例
1012
存在?
是
否
1014
将单个实例与Dial参数结合在一起
1020
对于任何匹配实例搜索在当前范围内的每个匹配实体
1022
存在?
是
否
1024
将每个匹配实例与Dial参数结合在一起
1030
结束
1032
错误

图 10

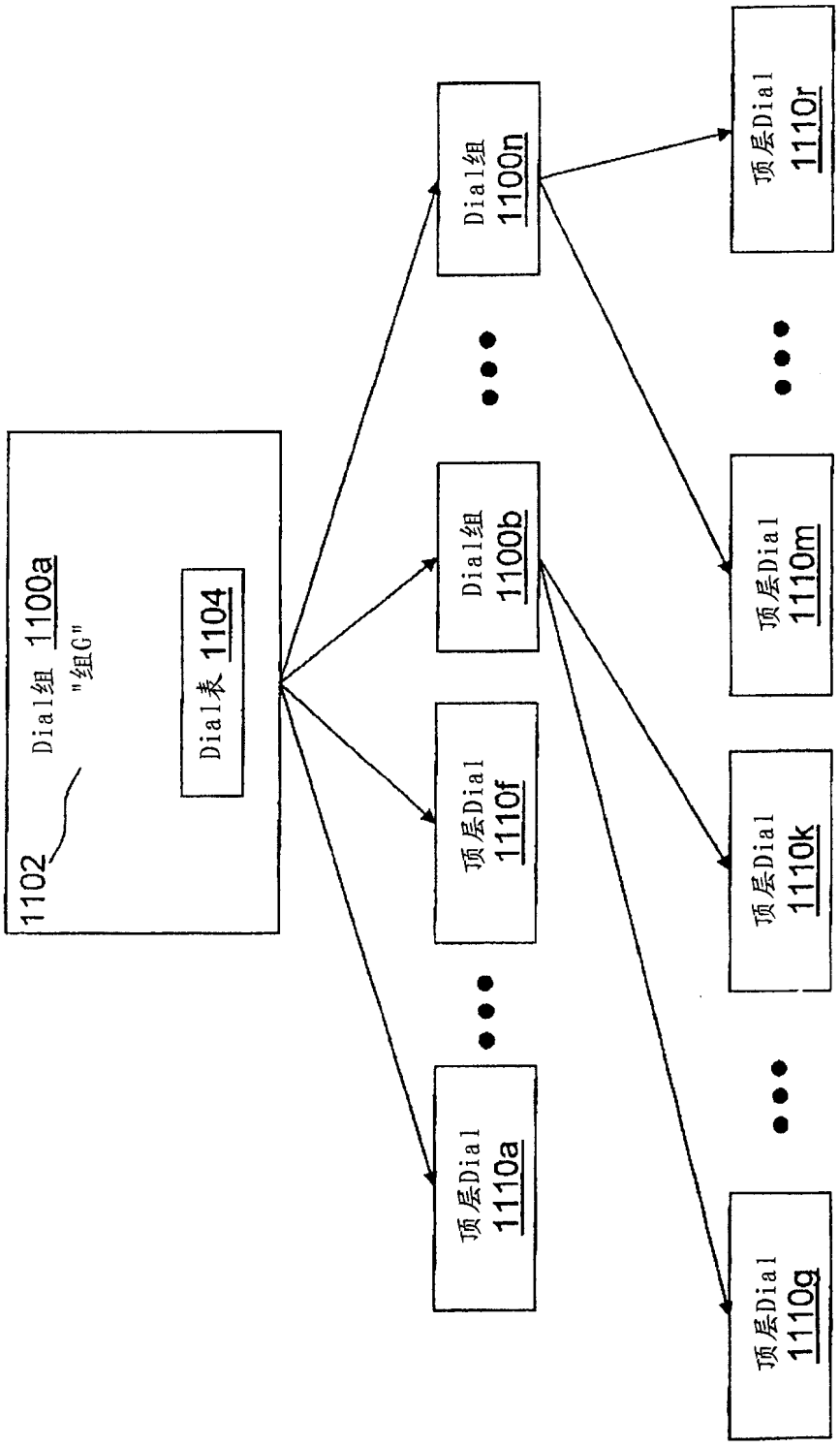

1102
Dial组 1100a
"组G"
Dial表 1104
Dial组 1100n
Dial组 1100b
顶层Dial 1110a
顶层Dial 1110f
顶层Dial 1110g
顶层Dial 1110k
顶层Dial 1110m
顶层Dial 1110r

图 11A

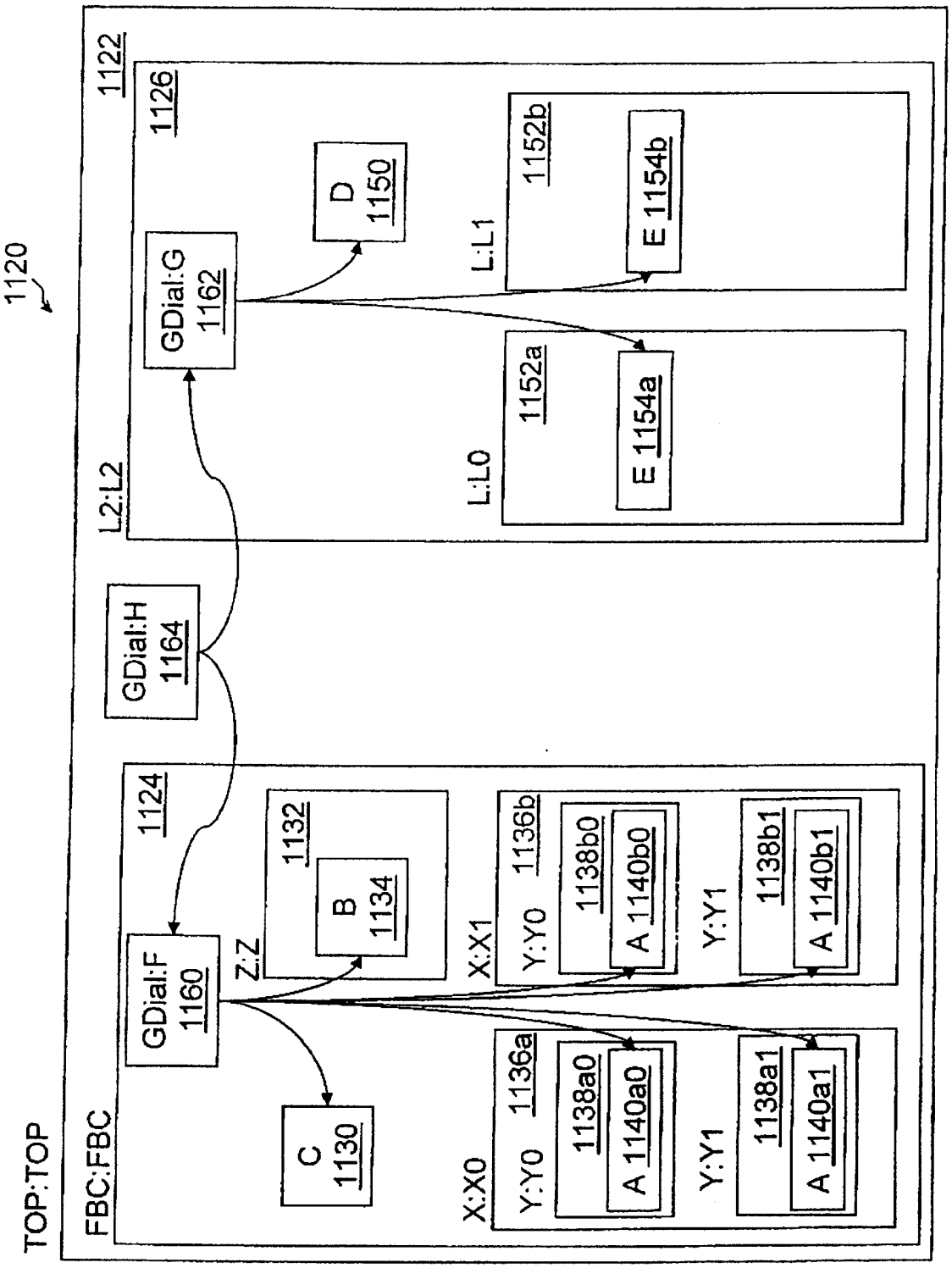

1120
TOP:TOP
FBC:FBC
1122
L2:L2
1126
GDial:G
1162
D
1150
L:L1
1152b
E 1154b
L:L0
1152a
E 1154a
GDial:H
1164
1124
GDial:F
1160
Z:Z
1132
B
1134
C
1130
X:X1
1136b
Y:Y0
1138b0
A 1140b0
Y:Y1
1138b1
A 1140b1
X:X0
1136a
Y:Y0
1138a0
A 1140a0
Y:Y1
1138a1
A 1140a1

图 11B

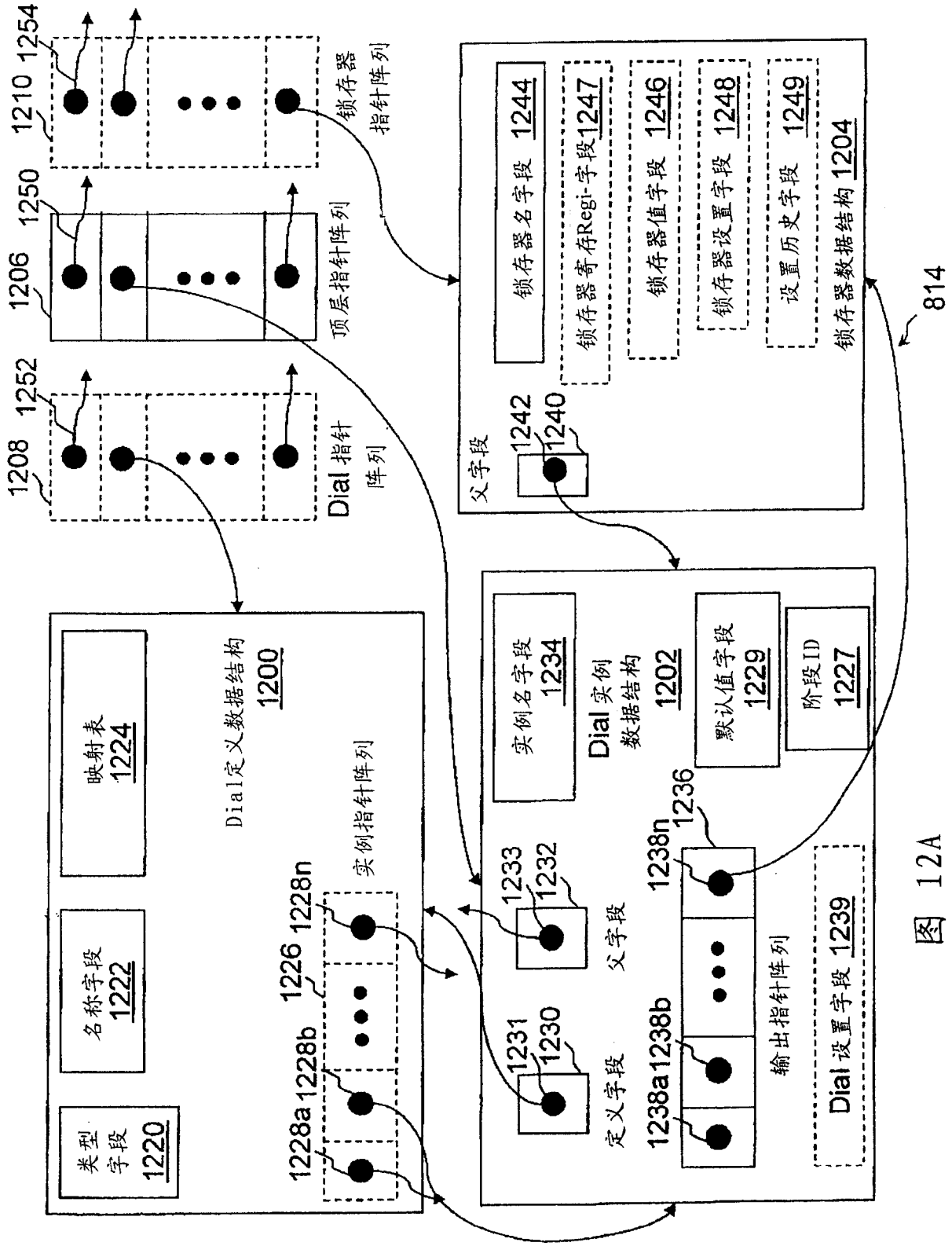

1210 1254
1206 1250
1208 1252
顶层指针阵列
锁存器指针阵列
Dial 指针阵列
Dial定义数据结构 1200
映射表 1224
名称字段 1222
类型字段 1220
实例指针阵列
1228a 1228b 1226 1228n
父字段
锁存器名字段 1244
锁存器寄存Regi-字段1247
锁存器值字段 1246
锁存器设置字段 1248
设置历史字段 1249
锁存器数据结构 1204
1242 1240
814
实例名字段 1234
Dial 实例数据结构 1202
默认值字段 1229
阶段 ID 1227
1233 1232
父字段
1231 1230
定义字段
1238a 1238b 1238n 1236
输出指针阵列
Dial 设置字段 1239

图 12A

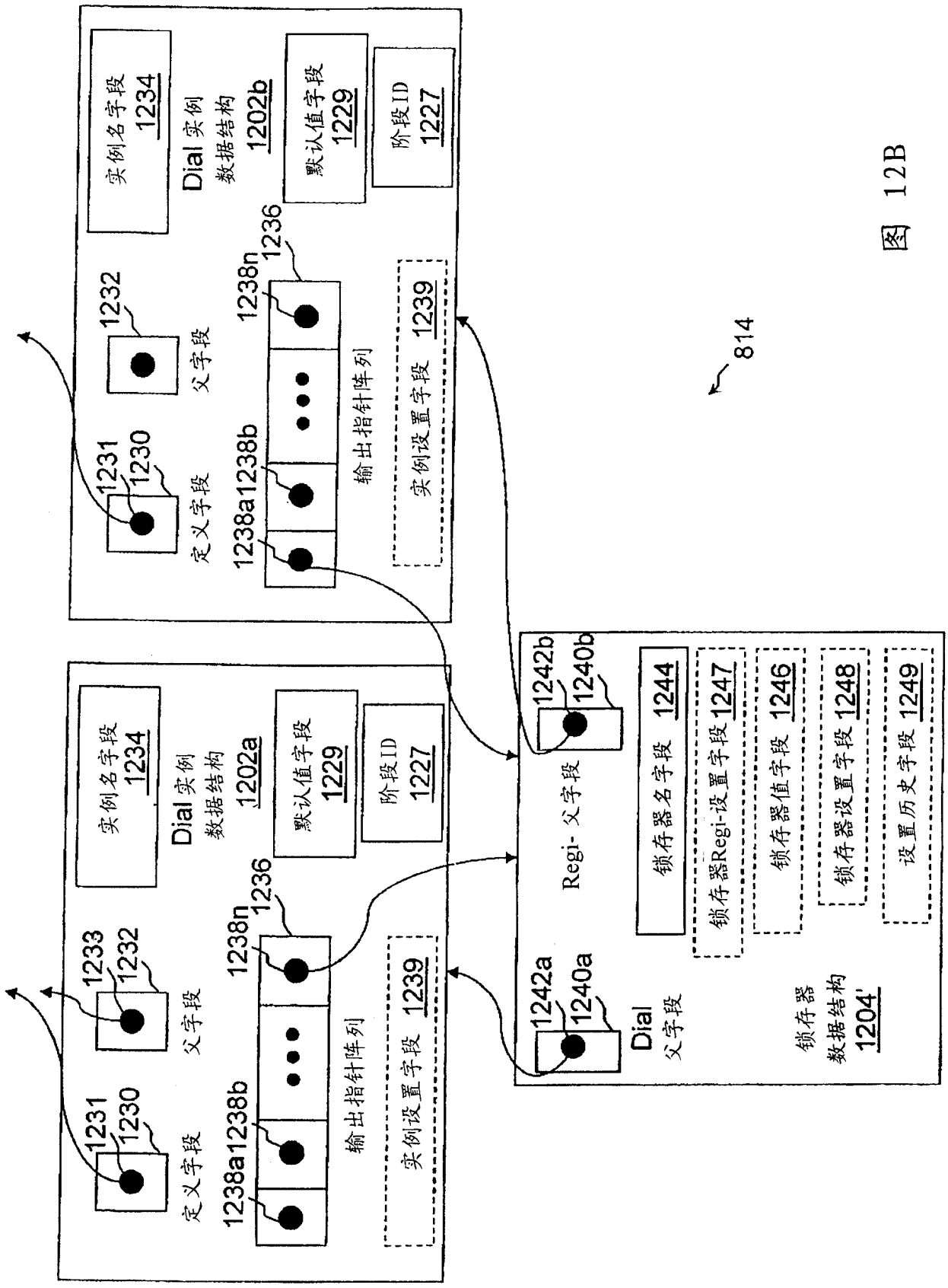

实例名字段 1234
Dial 实例 数据结构 1202b
默认值字段 1229
阶段ID 1227
1232
父字段
1231
1230
定义字段
1238n
1236
1238a
1238b
输出指针阵列
实例设置字段 1239
814
实例名字段 1234
Dial 实例 数据结构 1202a
默认值字段 1229
阶段ID 1227
1233
1232
父字段
1231
1230
定义字段
1238n
1236
1238a
1238b
输出指针阵列
实例设置字段 1239
1242b
1240b
Regi- 父字段
锁存器名字段 1244
锁存器Regi-设置字段 1247
锁存器值字段 1246
锁存器设置字段 1248
设置历史字段 1249
1242a
1240a
Dial
父字段
锁存器 数据结构 1204'

图 12B

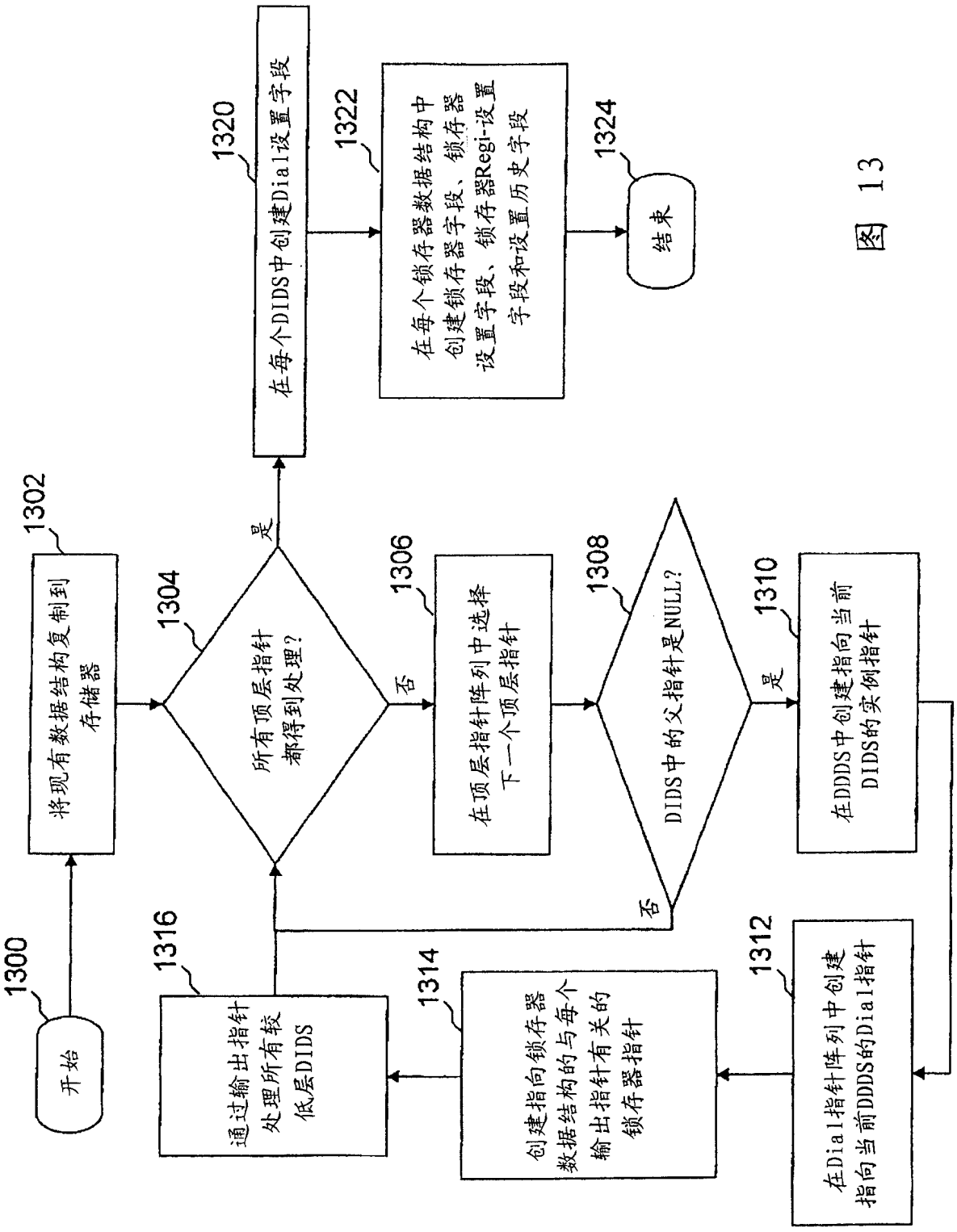

1300
开始
1302
将现有数据结构复制到存储器
1304
所有顶层指针都得到处理?
是
否
1306
在顶层指针阵列中选择下一个顶层指针
1308
DIDS中的父指针是NULL?
是
否
1310
在DDDS中创建指向当前DIDS的实例指针
1312
在Dial指针阵列中创建指向当前DDDS的Dial指针
1314
创建指向锁存器数据结构的与每个输出指针有关的锁存器指针
1316
通过输出指针处理所有较低层DIDS
1320
在每个DIDS中创建Dial设置字段
1322
在每个锁存器数据结构中创建锁存器字段、锁存器设置字段、锁存器Regi-设置字段和设置历史字段
1324
结束

图 13

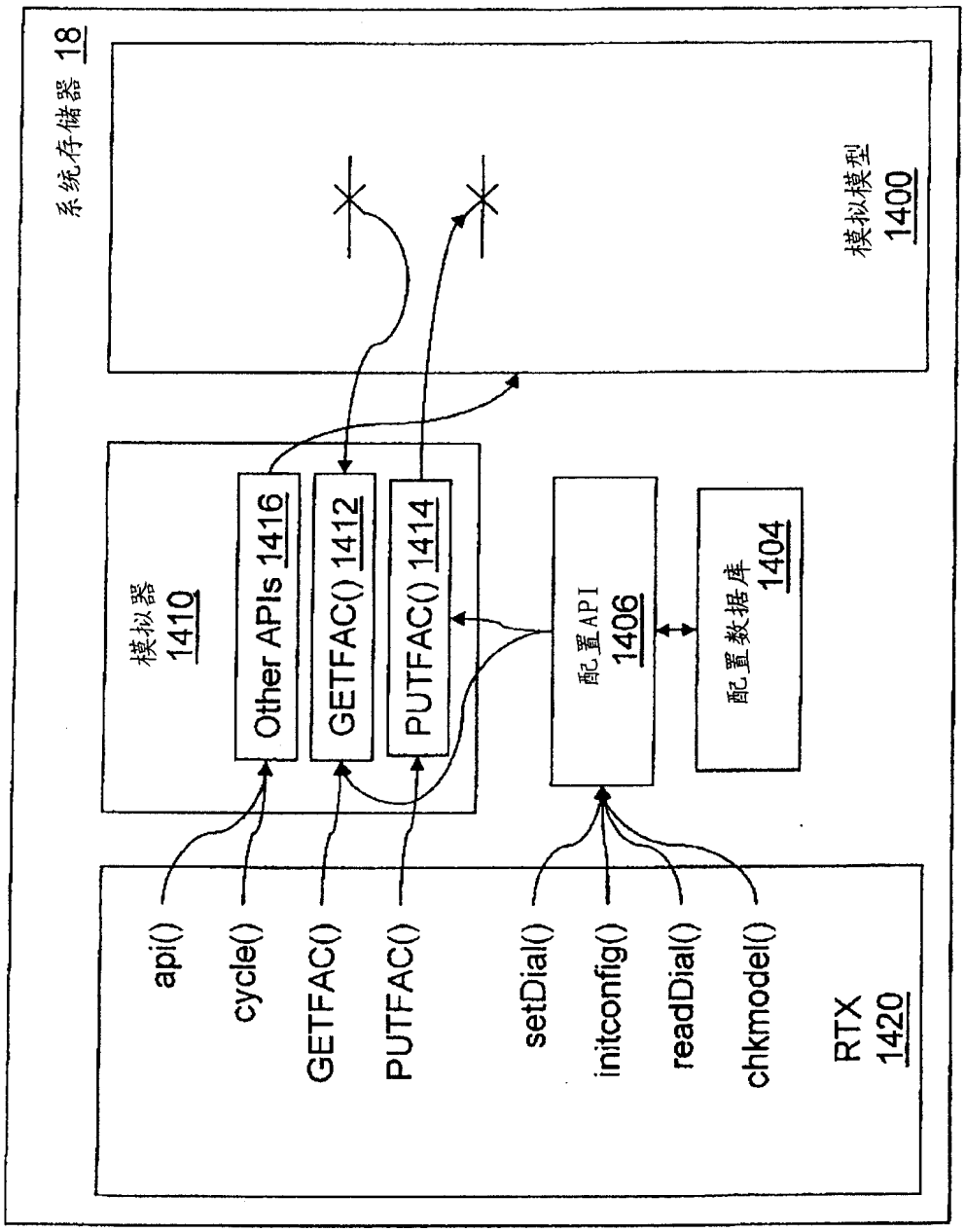

系统存储器 18
模拟模型
1400
模拟器
1410
Other APIs 1416
GETFAC() 1412
PUTFAC() 1414
配置API
1406
配置数据库
1404
api()
cycle()
GETFAC()
PUTFAC()
setDial()
initconfig()
readDial()
chkmodel()
RTX
1420

图 14

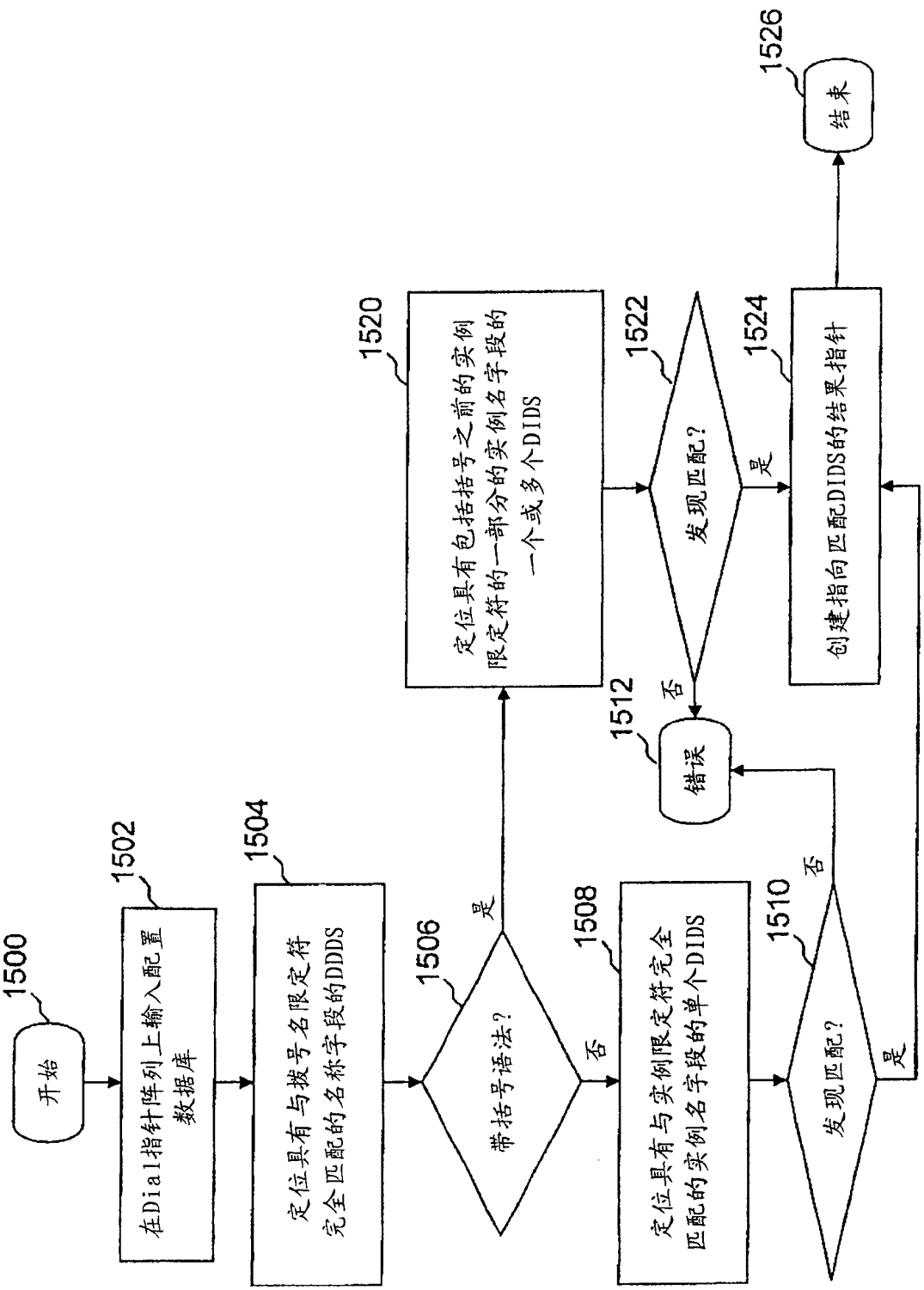

1500
开始
1502
在Dial指针阵列上输入配置数据库
1504
定位具有与拨号名限定符完全匹配的名称字段的DDDS
1506
带括号语法?
是
否
1508
定位具有与实例限定符完全匹配的单个DIDS
1510
发现匹配?
是
否
1512
错误
1520
定位具有包括括号之前的实例限定符的一部分的实例名字段的一个或多个DIDS
1522
发现匹配?
是
否
1524
创建指向匹配DIDS的结果指针
1526
结束

图 15

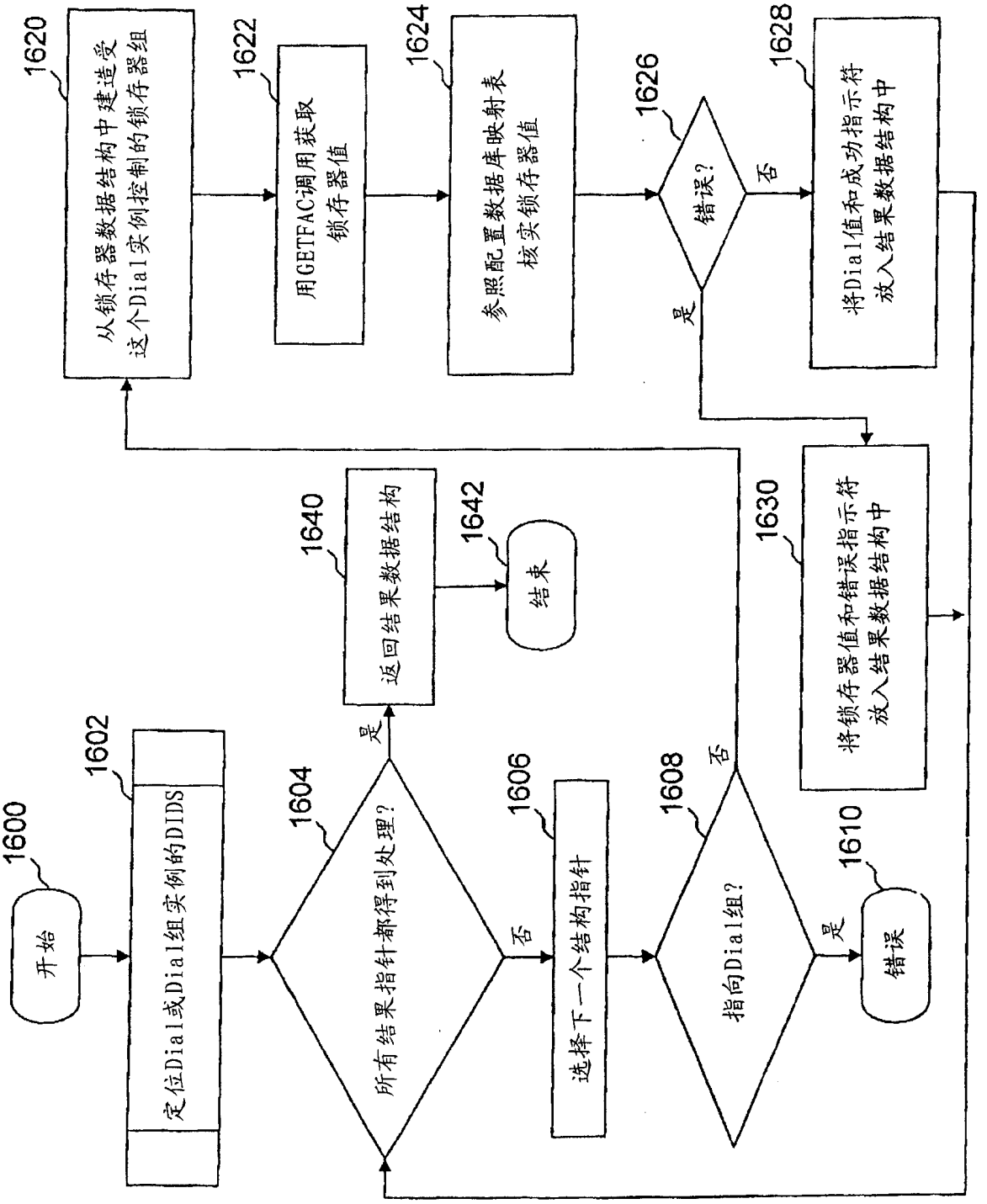

1600
开始
1602
定位Dial或Dial组实例的DIDS
1604
所有结果指针都得到处理?
是
否
1606
选择下一个结构指针
1608
指向Dial组?
是
否
1610
错误
1620
从锁存器数据结构中建造受这个Dial实例控制的锁存器组
1622
用GETFAC调用获取锁存器值
1624
参照配置数据库映射表核实锁存器值
1626
错误?
是
否
1628
将Dial值和成功指示符放入结果数据结构中
1630
将锁存器值和错误指示符放入结果数据结构中
1640
返回结果数据结构
1642
结束

图 16A

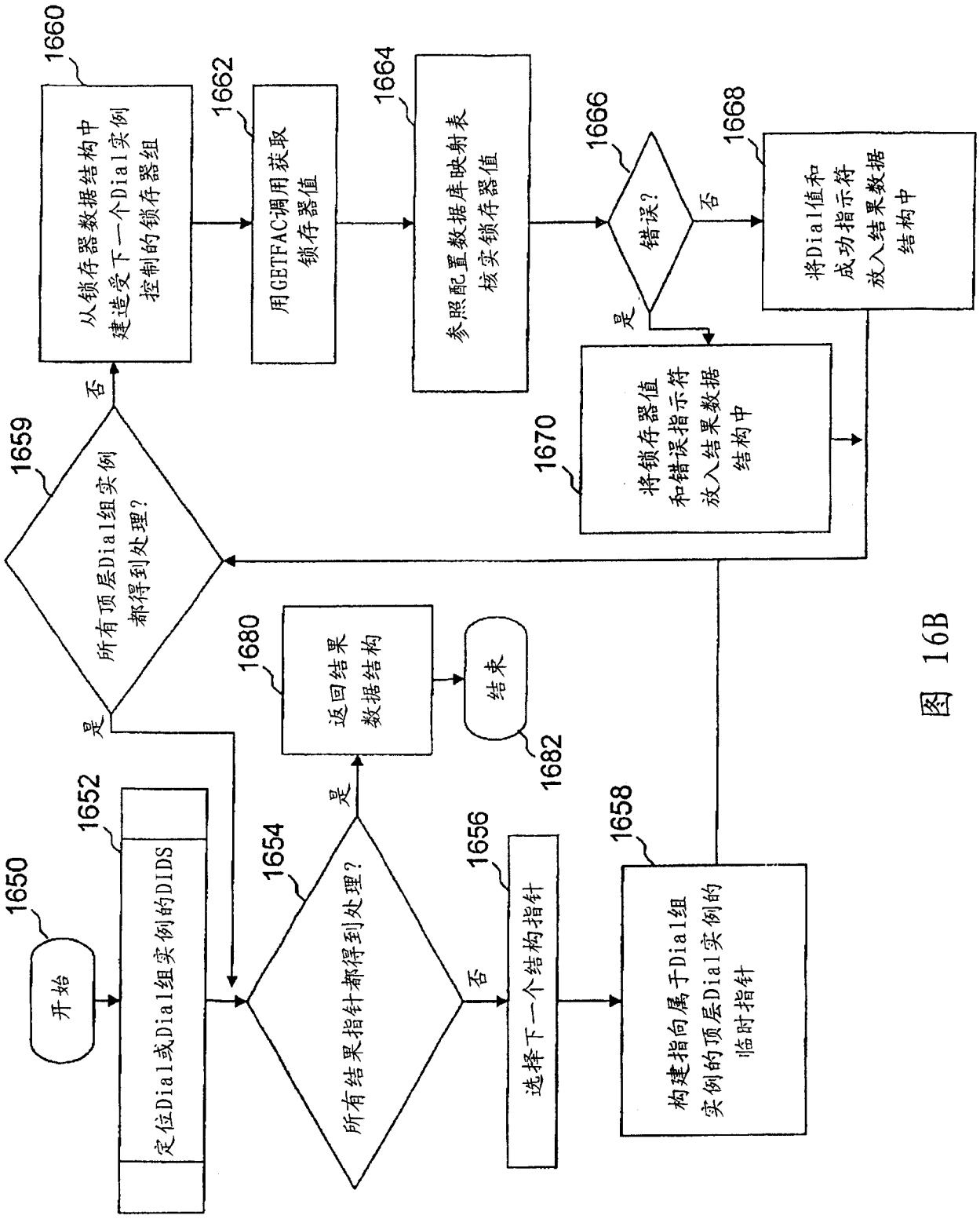

1650
开始
1652
定位Dial或Dial组实例的DIDS
1654
所有结果指针都得到处理?
是
否
1656
选择下一个结构指针
1658
构建指向属于Dial组实例的顶层Dial实例的临时指针
1659
所有顶层Dial组实例都得到处理?
是
否
1660
从锁存器数据结构中建造受下一个Dial实例控制的锁存器组
1662
用GETFAC调用获取锁存器值
1664
参照配置数据库映射表核实锁存器值
1666
错误?
是
否
1668
将Dial值和成功指示符放入结果数据结构中
1670
将锁存器值和错误指示符放入结果数据结构中
1680
返回结果数据结构
1682
结束

图 16B

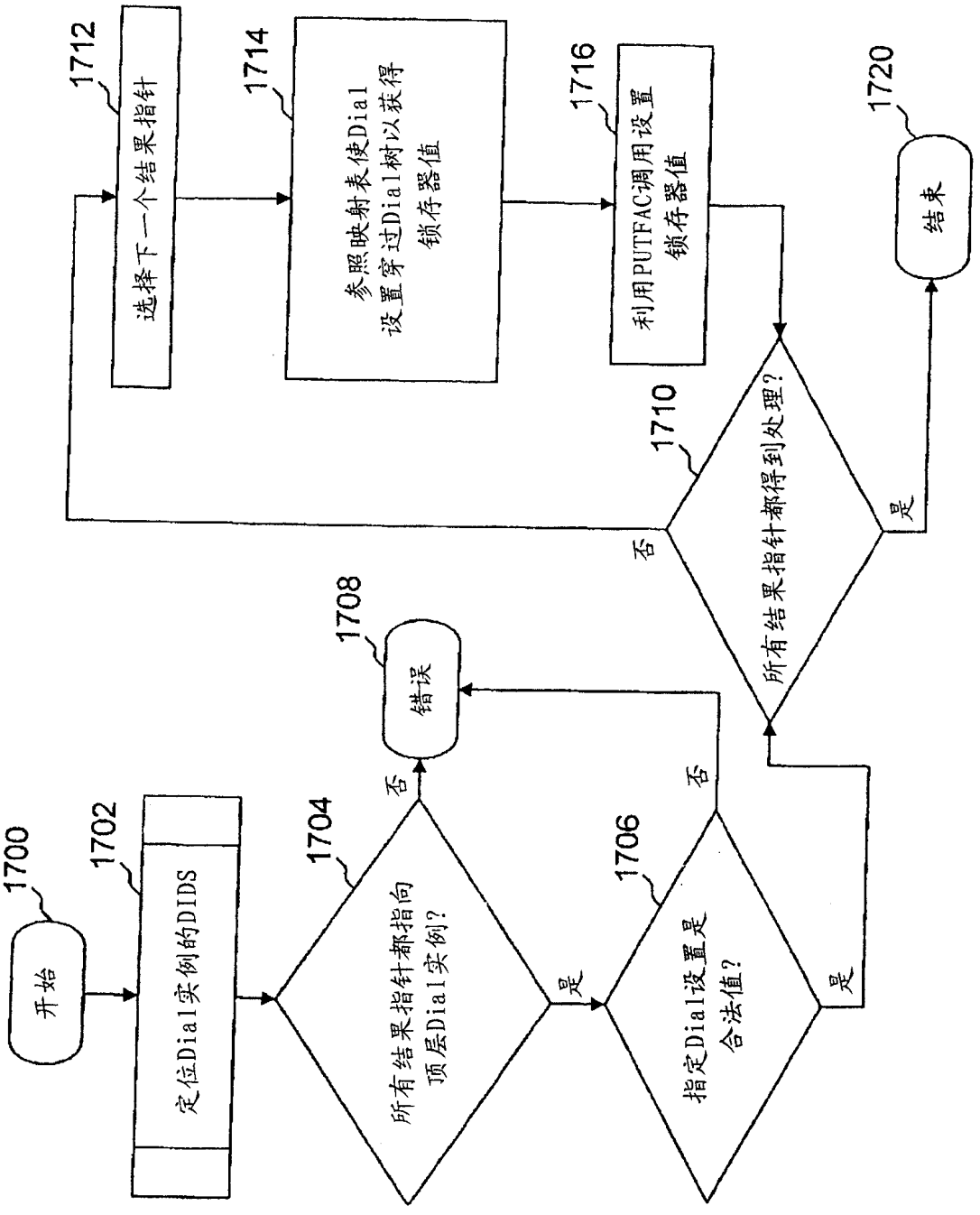

1700
开始
1702
定位Dial实例的DIDS
1704
所有结果指针都指向顶层Dial实例?
否
1708
错误
是
1706
指定Dial设置是合法值?
否
是
1710
所有结果指针都得到处理?
否
是
1712
选择下一个结果指针
1714
参照映射表使Dial设置穿过Dial树以获得锁存器值
1716
利用PUTFAC调用设置锁存器值
1720
结束

图 17A

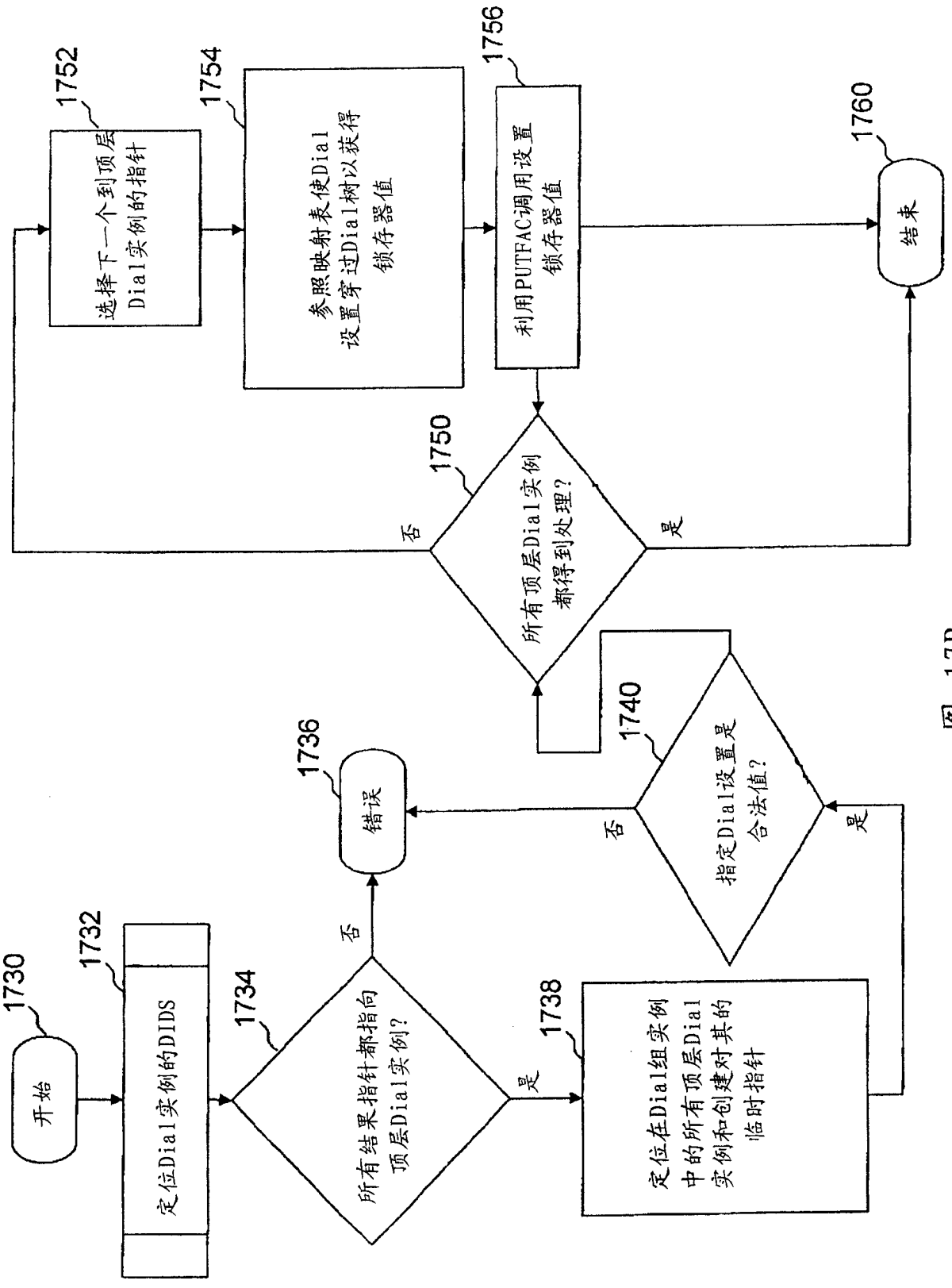

开始
1730
定位Dial实例的DIDS
1732
所有结果指针都指向顶层Dial实例?
1734
否
是
错误
1736
定位在Dial组实例中的所有顶层Dial实例和创建对其的临时指针
1738
指定Dial设置是合法值?
1740
否
是
所有顶层Dial实例都得到处理?
1750
否
是
选择下一个到顶层Dial实例的指针
1752
参照映射表使Dial设置穿过Dial树以获得锁存器值
1754
利用PUTFAC调用设置锁存器值
1756
结束
1760

图 17B

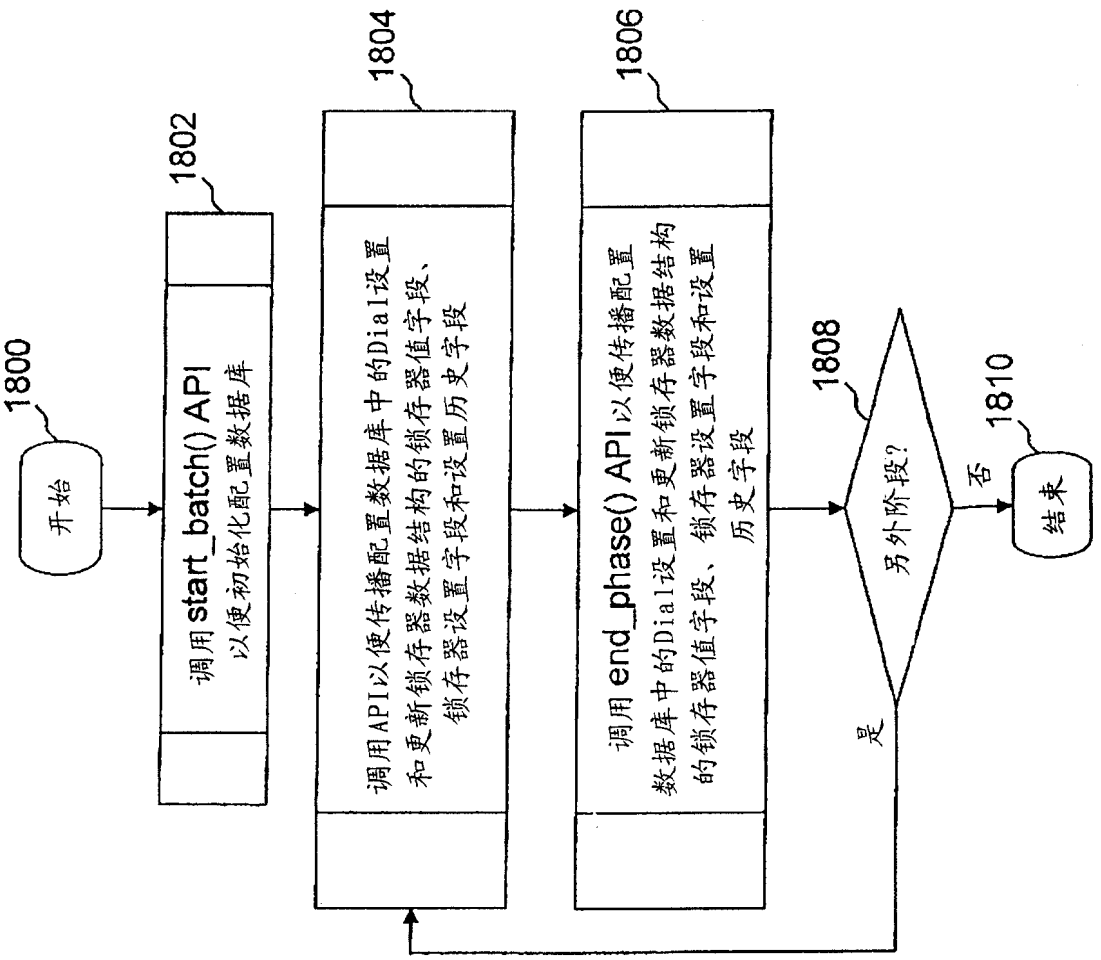

1800
开始
1802
调用start_batch() API
以便初始化配置数据库
1804
调用API以便传播配置数据库中的Dial设置
和更新锁存器数据结构的锁存器值字段、
锁存器设置字段和设置历史字段
1806
调用end_phase() API以便传播配置
数据库中的Dial设置和更新锁存器数据结构
的锁存器值字段、锁存器设置字段和设置
历史字段
1808
另外阶段?
是
否
1810
结束

图 18A

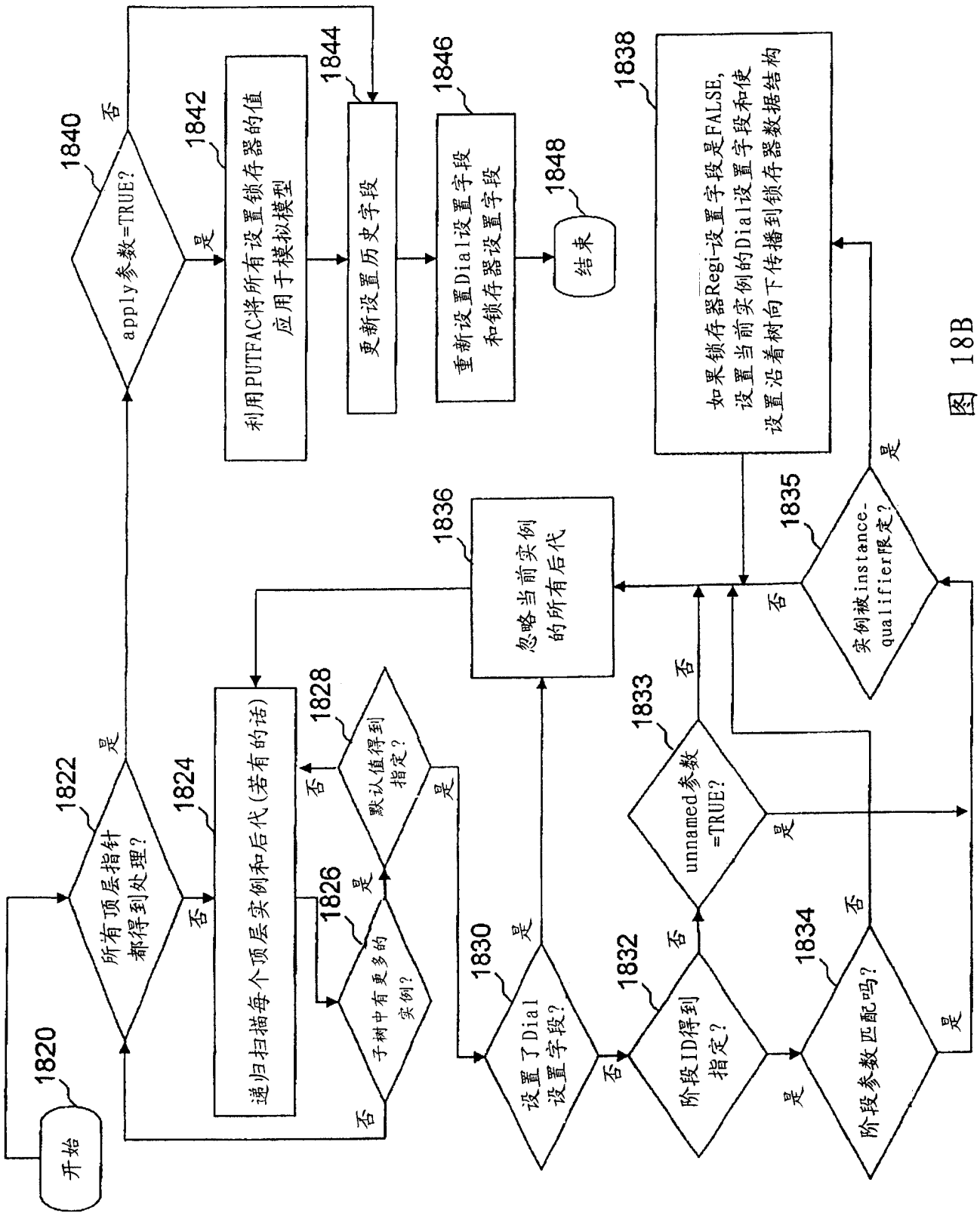

开始
1820
所有顶层指针都得到处理?
1822
是
否
递归扫描每个顶层实例和后代(若有的话)
1824
子树中有更多的实例?
1826
是
否
默认值得到指定?
1828
是
否
忽略当前实例的所有后代
1836
设置了Dial设置字段?
1830
是
否
阶段ID得到指定?
1832
是
否
阶段参数匹配吗?
1834
是
否
unnamed参数=TRUE?
1833
是
否
实例被instance-qualifier限定?
1835
是
否
如果锁存器Regi-设置字段是FALSE,设置当前实例的Dial设置字段和使设置沿着树向下传播到锁存器数据结构
1838
apply参数=TRUE?
1840
是
否
利用PUTFAC将所有设置锁存器的值应用于模拟模型
1842
更新设置历史字段
1844
重新设置Dial设置字段和锁存器设置字段
1846
结束
1848

图 18B

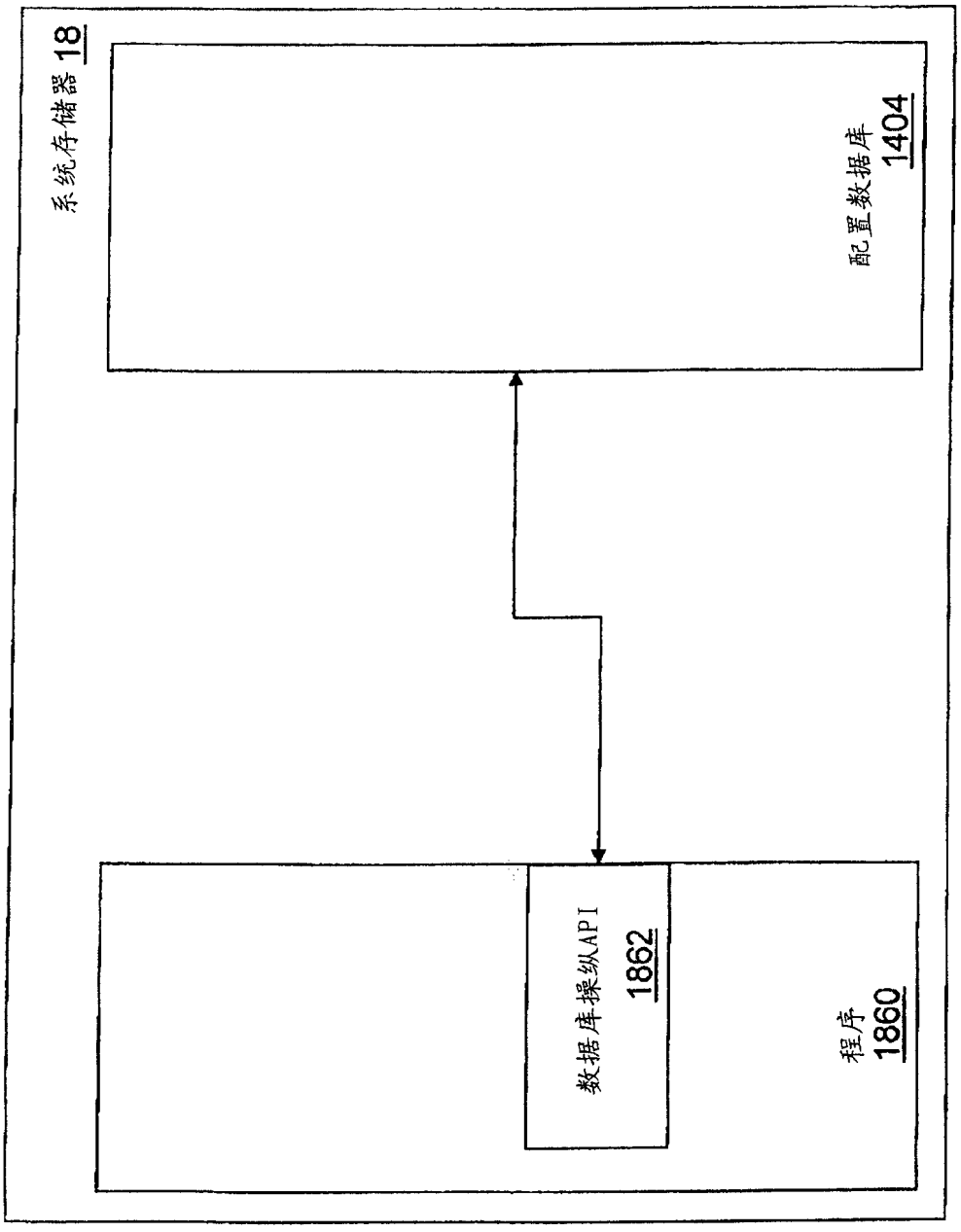


图 18C

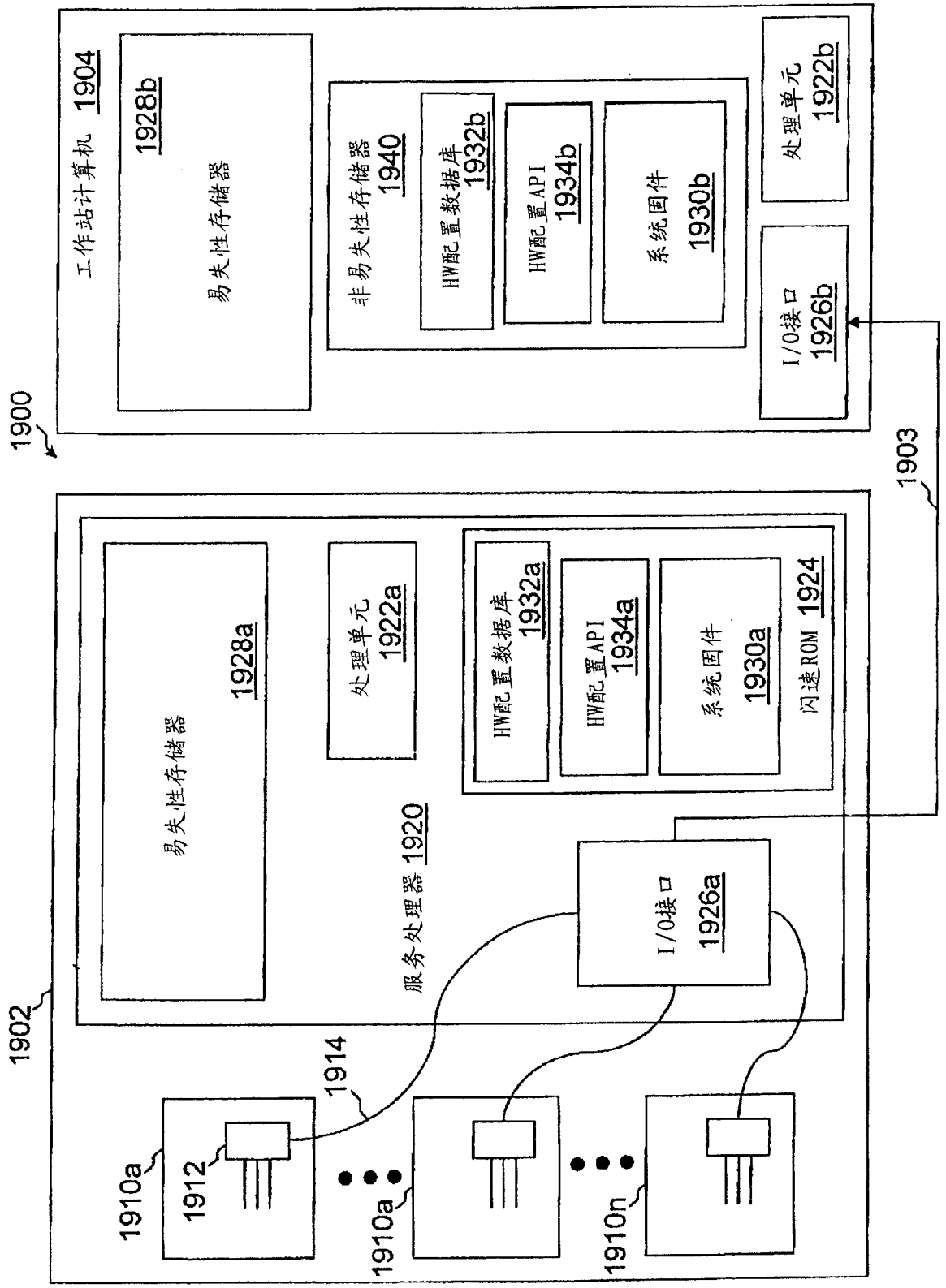

1900
工作站计算机 1904
1928b
易失性存储器
非易失性存储器 1940
HW配置数据库 1932b
HW配置API 1934b
系统固件 1930b
处理单元 1922b
I/O接口 1926b
1903
1902
易失性存储器 1928a
处理单元 1922a
服务处理器 1920
HW配置数据库 1932a
HW配置API 1934a
系统固件 1930a
闪速ROM 1924
I/O接口 1926a
1914
1910a
1912
1910a
1910n

图 19

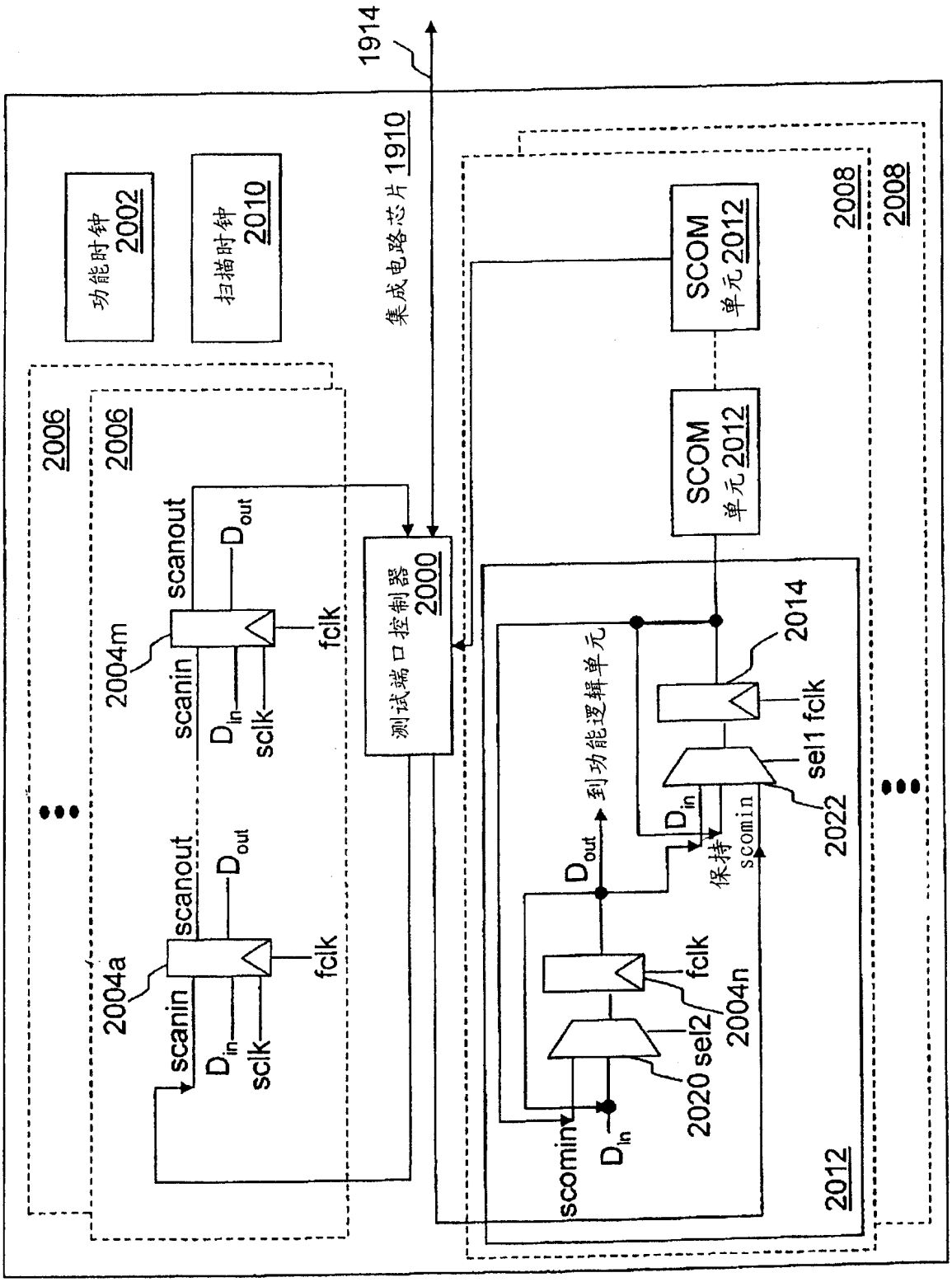

1914
集成电路芯片 1910
功能时钟 2002
扫描时钟 2010
测试端口控制器 2000
2006
2006
2004m
2004a
scanout
scanin
D_{out}
D_{in}
sclk
fclk
SCOM 单元 2012
SCOM 单元 2012
2008
2008
2012
2014
2022
2020
2004n
sel1
sel2
scomin
保持
到功能逻辑单元

图 20

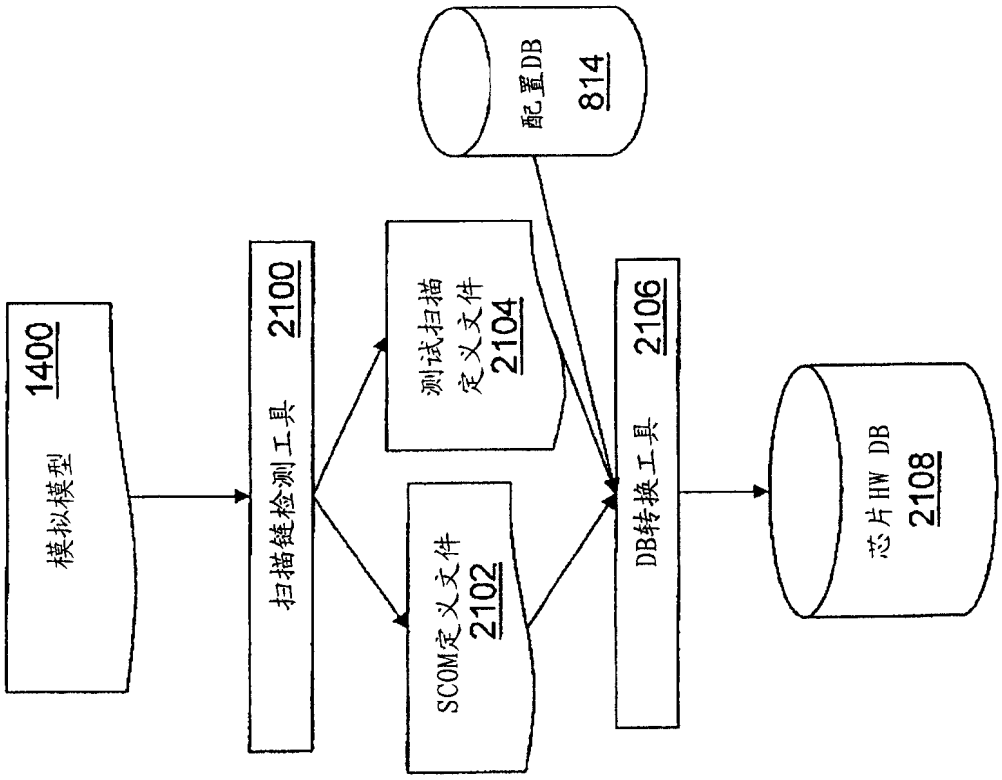

模拟模型
1400
扫描链检测工具
2100
测试扫描
定义文件
2104
SCOM定义文件
2102
配置DB
814
DB转换工具
2106
芯片HW DB
2108

图 21

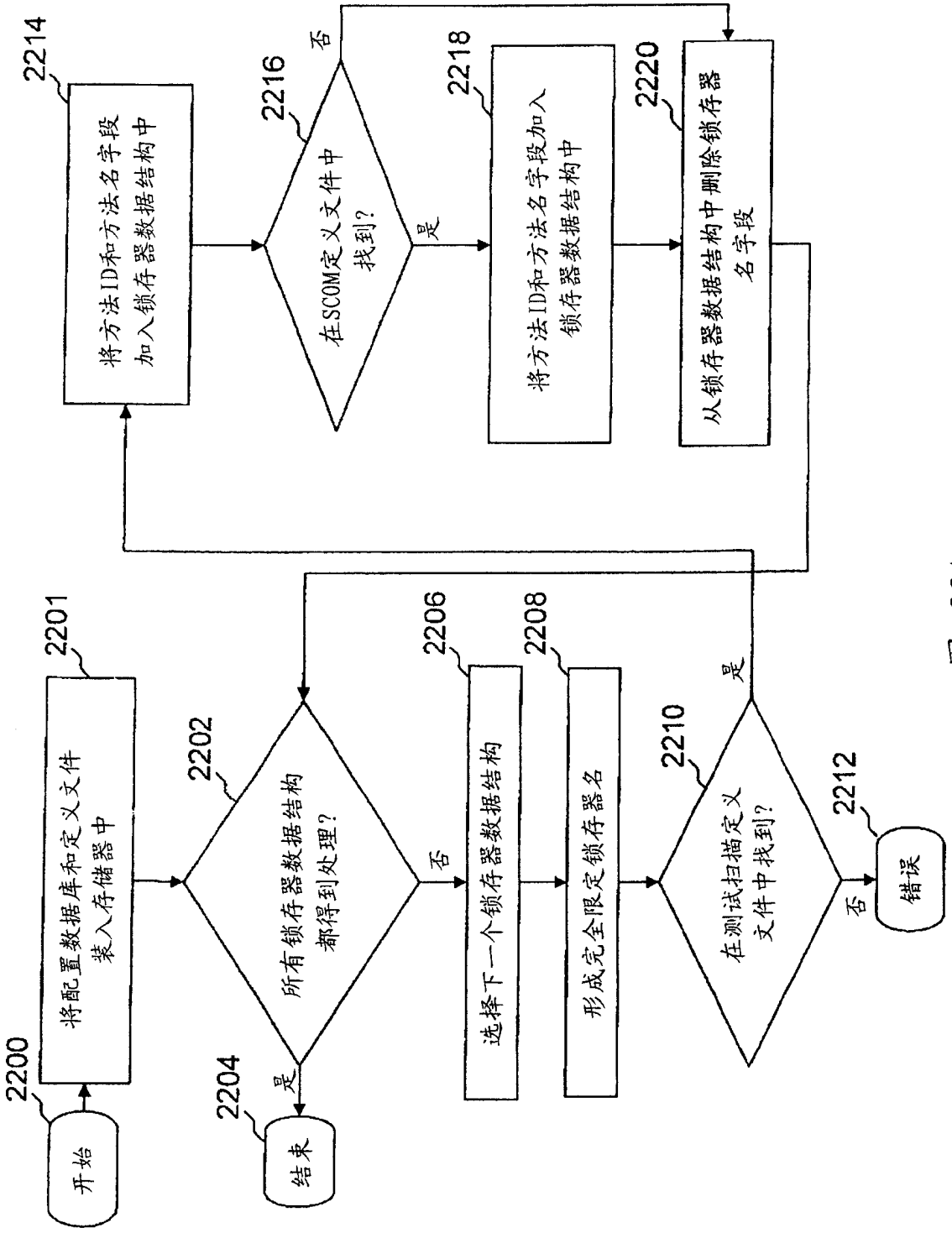

2200
开始
2201
将配置数据库和定义文件装入存储器中
2202
所有锁存器数据结构都得到处理?
是
2204
结束
否
2206
选择下一个锁存器数据结构
2208
形成完全限定锁存器名
2210
在测试扫描定义文件中找到?
是
否
2212
错误
2214
将方法ID和方法名字段加入锁存器数据结构中
2216
在SCOM定义文件中找到?
是
否
2218
将方法ID和方法名字段加入锁存器数据结构中
2220
从锁存器数据结构中删除锁存器名字段

图 22A

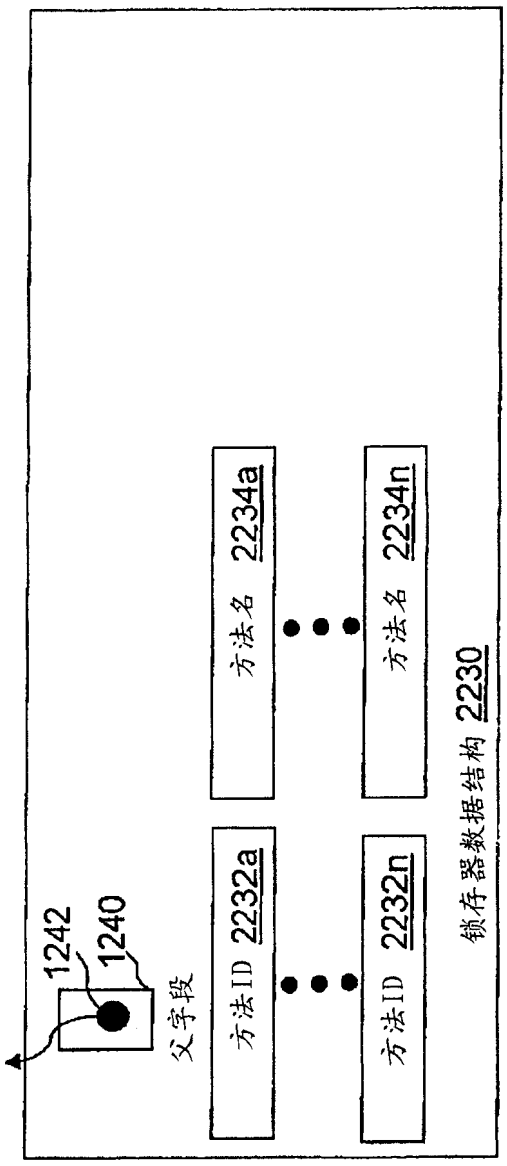

1242
1240
父字段
方法ID 2232a
方法ID 2232n
方法名 2234a
方法名 2234n
锁存器数据结构 2230

图 22B

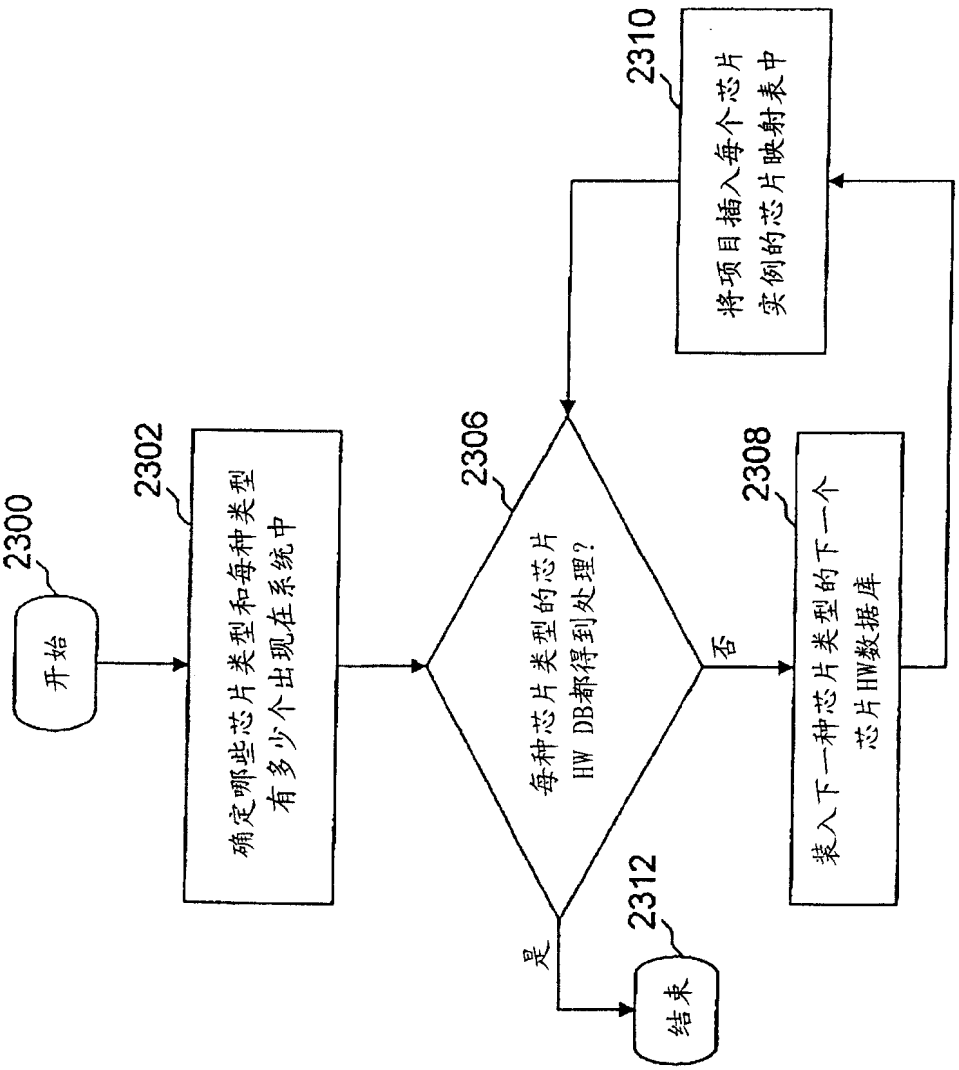

2300
开始
2302
确定哪些芯片类型和每种类型有多少个出现在系统中
2306
每种芯片类型的芯片HW DB都得到处理?
否
2308
装入下一种芯片类型的下一个芯片HW数据库
2310
将项目插入每个芯片实例的芯片映射表中
是
2312
结束

图 23A

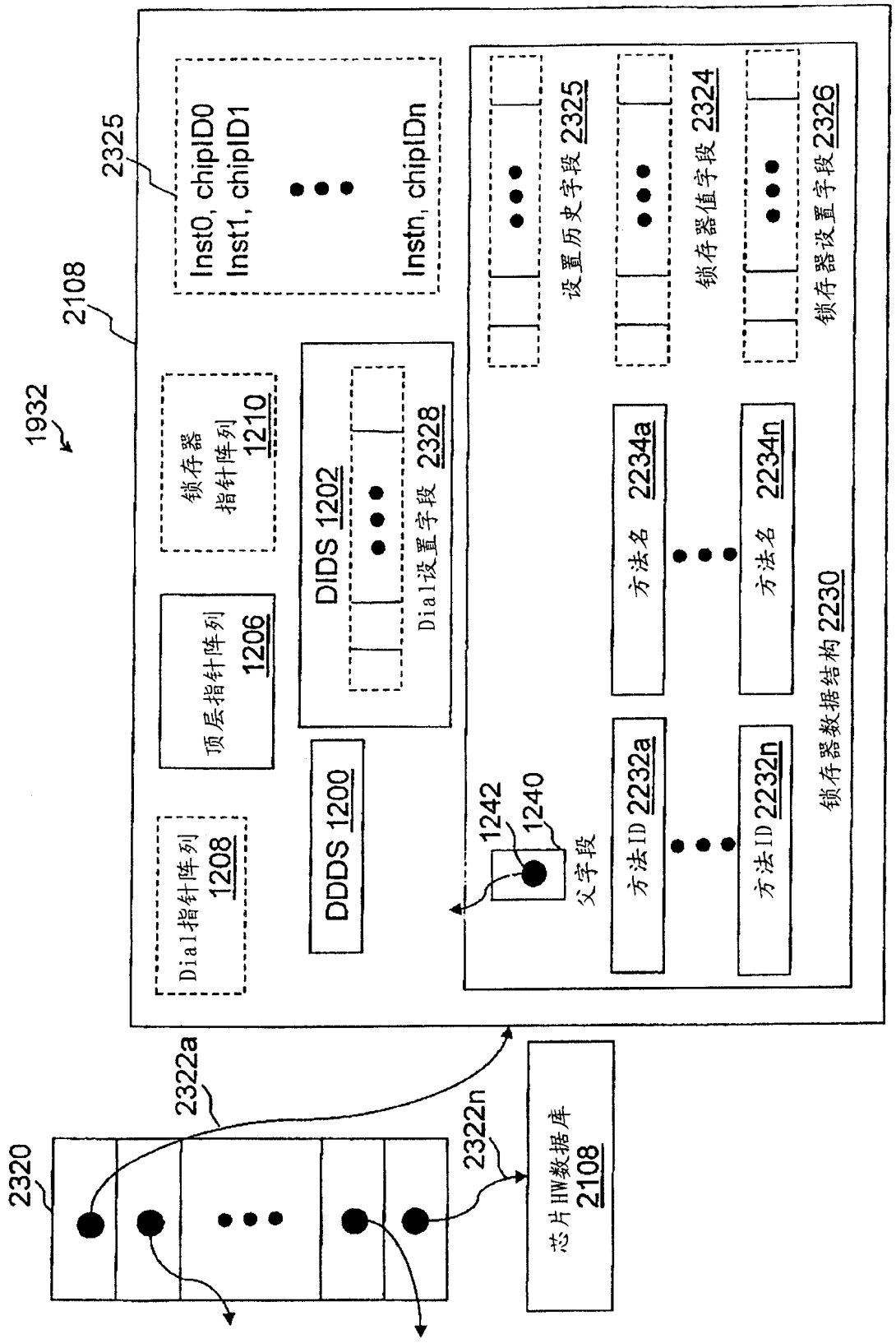

1932
2108
2325
Inst0, chipID0
Inst1, chipID1
Instn, chipIDn
锁存器
指针阵列
1210
顶层指针阵列
1206
DIDS 1202
Dial设置字段 2328
Dial指针阵列
1208
DDDS 1200
1242
1240
父字段
设置历史字段 2325
锁存器值字段 2324
锁存器设置字段 2326
方法名 2234a
方法名 2234n
方法ID 2232a
方法ID 2232n
锁存器数据结构 2230
2320
2322a
2322n
芯片HW数据库
2108

图 23B

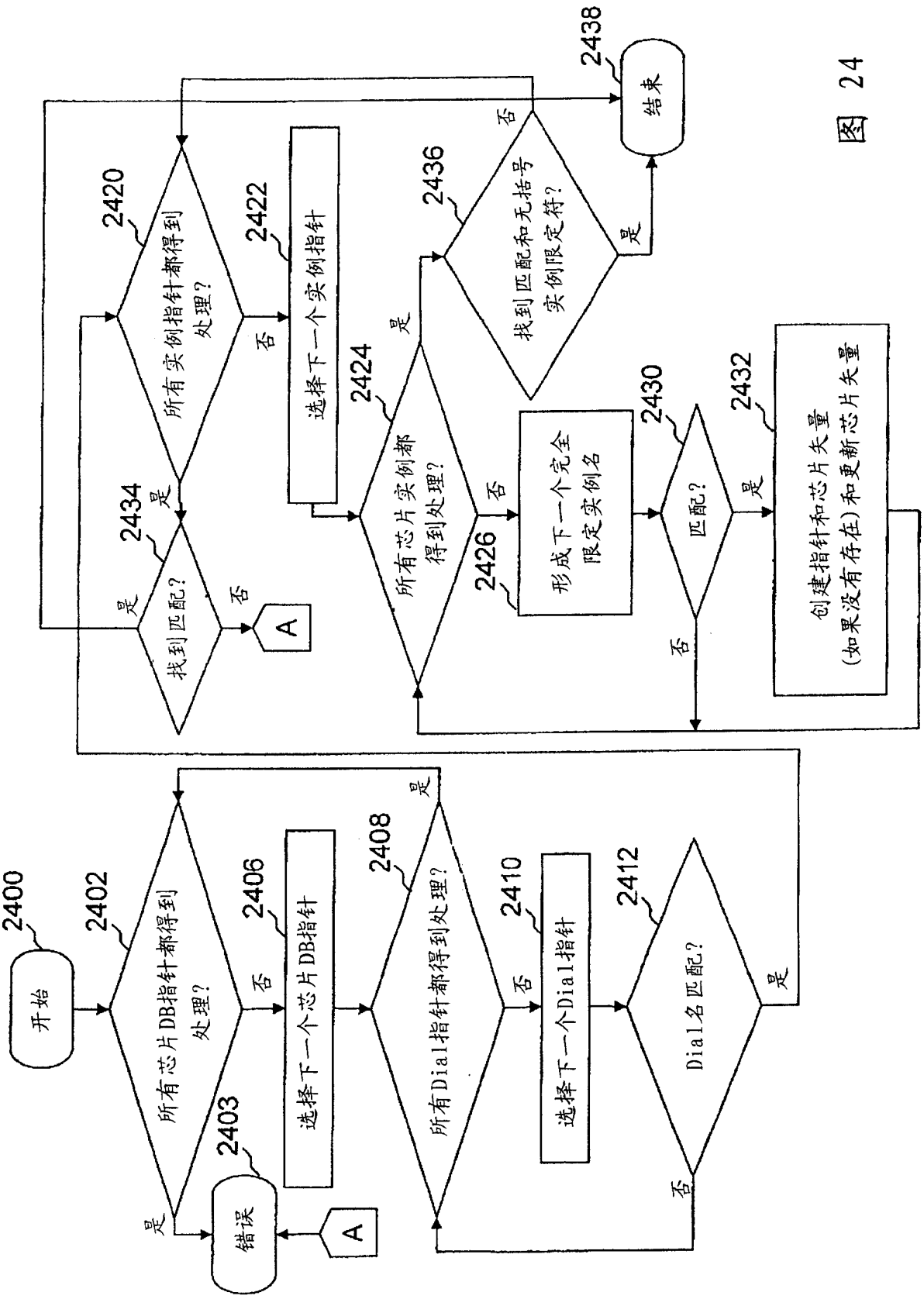

2400
开始
2402
所有芯片DB指针都得到处理?
是
否
2403
错误
2406
选择下一个芯片DB指针
A
2408
所有Dial指针都得到处理?
是
否
2410
选择下一个Dial指针
2412
Dial名匹配?
是
否
2420
所有实例指针都得到处理?
是
否
2422
选择下一个实例指针
2424
所有芯片实例都得到处理?
是
否
2426
形成下一个完全限定实例名
2430
匹配?
是
否
2432
创建指针和芯片矢量
(如果没有存在)和更新芯片矢量
2434
找到匹配?
是
否
A
2436
找到匹配和无括号实例限定符?
是
否
2438
结束

图 24

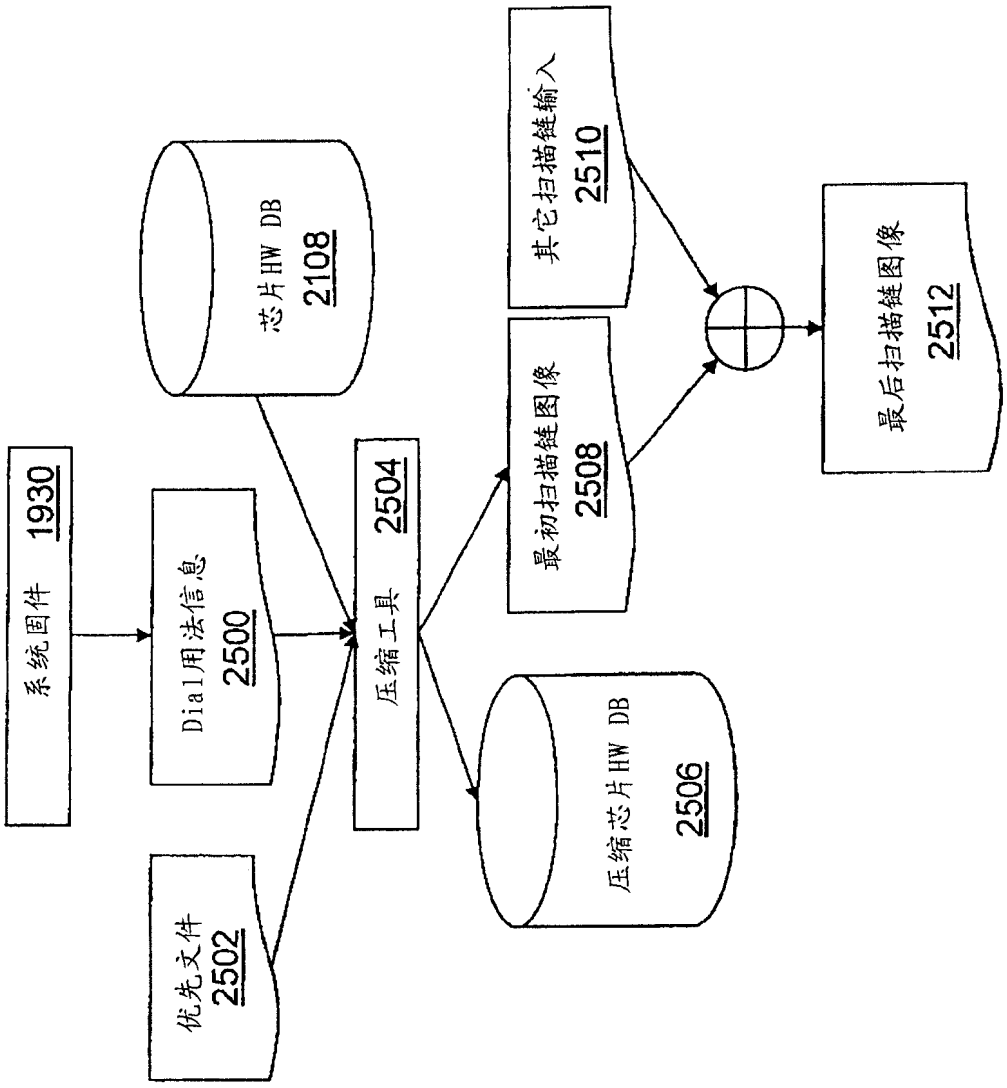

系统图件 1930
Dial用法信息 2500
优先文件 2502
压缩工具 2504
芯片HW DB 2108
压缩芯片HW DB 2506
最初扫描链图像 2508
其它扫描链输入 2510
最后扫描链图像 2512

图 25

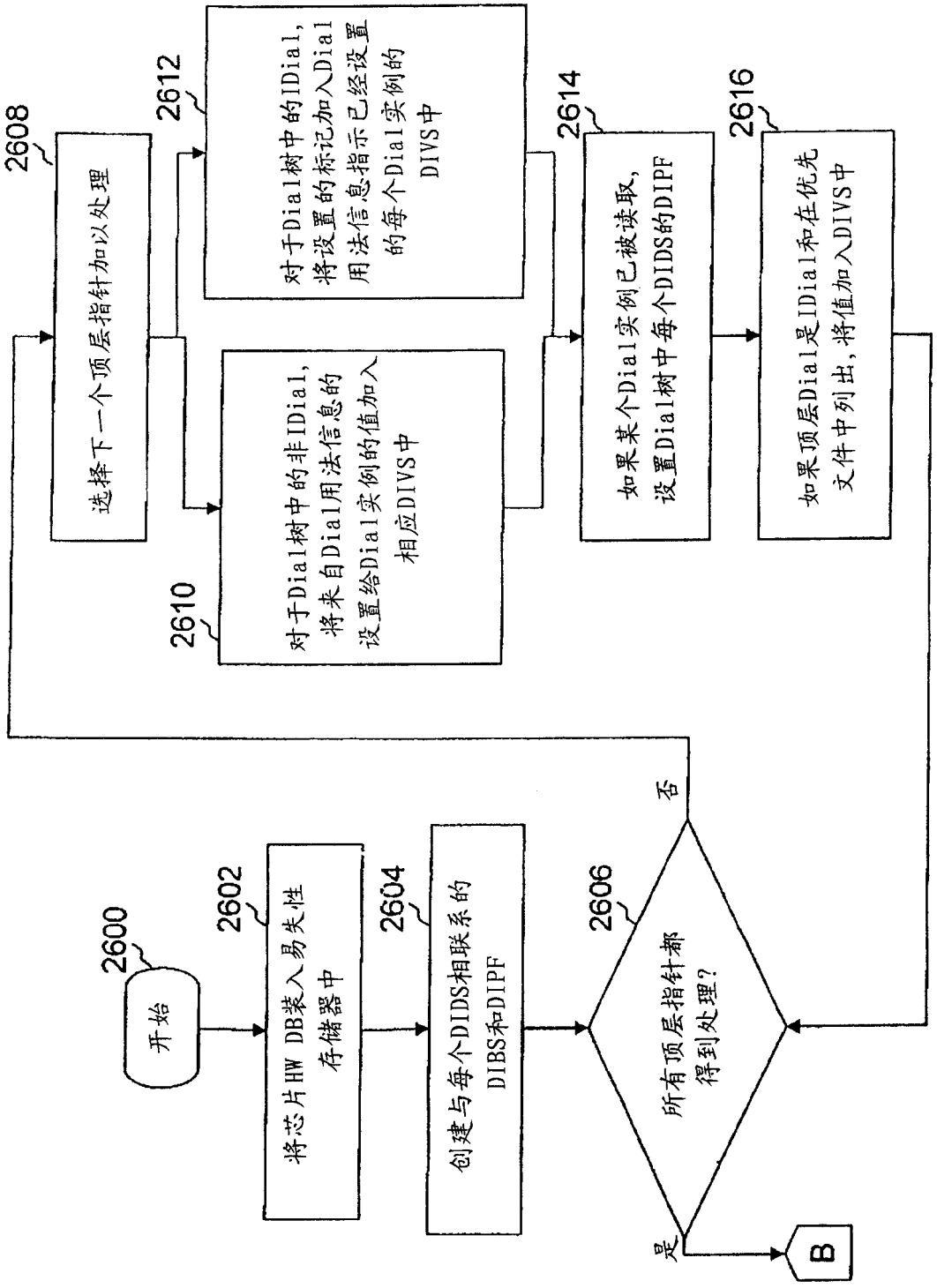

2600
开始
2602
将芯片HW DB装入易失性存储器中
2604
创建与每个DIDS相联系的DIBS和DIPF
2606
所有顶层指针都得到处理?
否
是
B
2608
选择下一个顶层指针加以处理
2610
对于Dial树中的非IDial，将来自Dial用法信息的设置给Dial实例的值加入相应DIVS中
2612
对于Dial树中的IDial，将设置的标记加入Dial用法信息指示已经设置的每个Dial实例的DIVS中
2614
如果某个Dial实例已被读取，设置Dial树中每个DIDS的DIPF
2616
如果顶层Dial是IDial和在优先文件中列出，将值加入DIVS中

图 26A

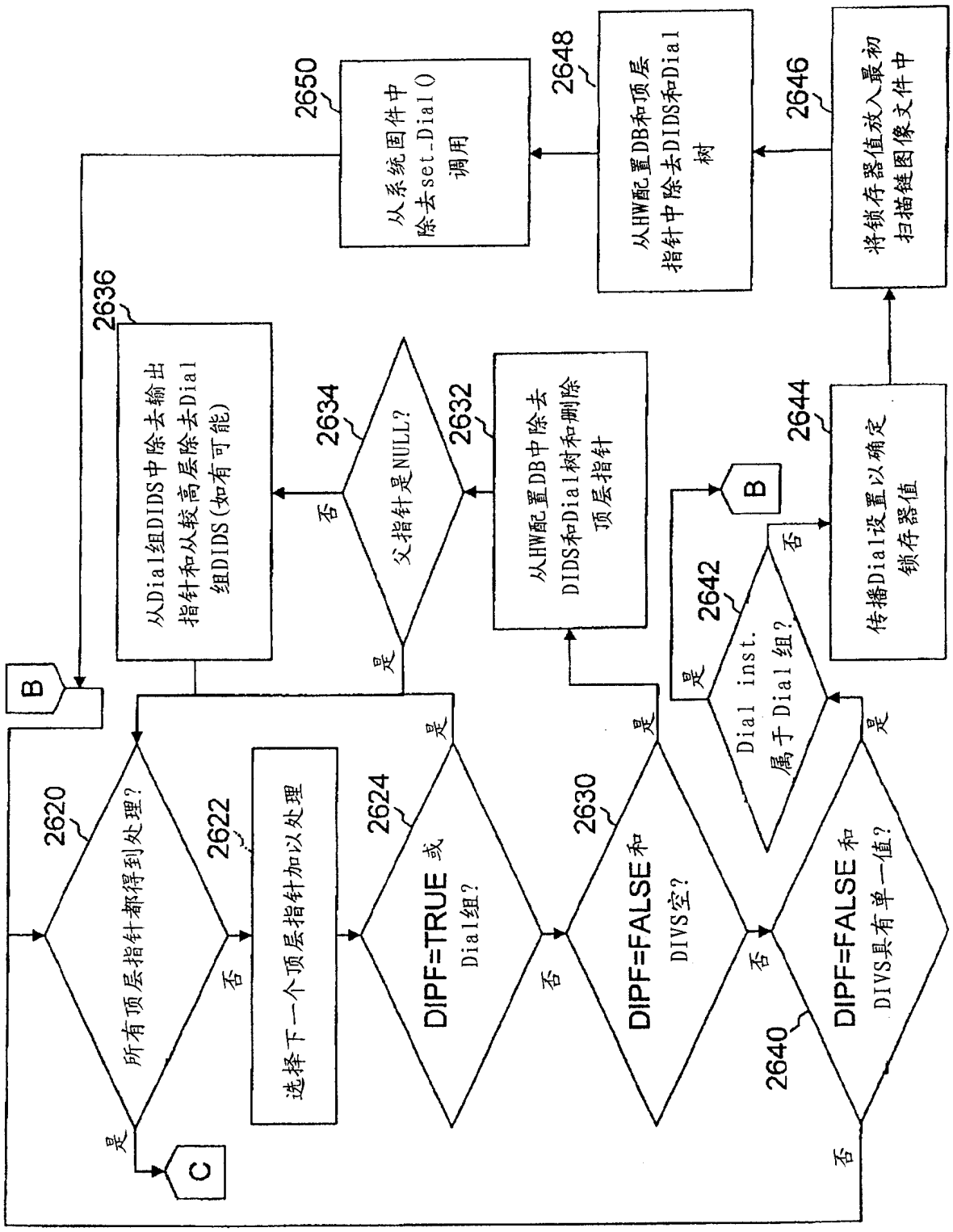

2650
从系统固件中除去set_Dial()调用
2648
从HW配置DB和顶层指针中除去DIDS和Dial树
2646
将锁存器值放入最初扫描链图像文件中
2636
从Dial组DIDS中除去输出指针和从较高层除去Dial组DIDS(如有可能)
2634
父指针是NULL?
2632
从HW配置DB中除去DIDS和Dial树和删除顶层指针
2644
传播Dial设置以确定锁存器值
B
2642
Dial inst.属于Dial组?
2620
所有顶层指针都得到处理?
2622
选择下一个顶层指针加以处理
2624
DIPF=TRUE或Dial组?
2630
DIPF=FALSE和DIVS空?
2640
DIPF=FALSE和DIVS具有单一值?
C
是
否

图 26B

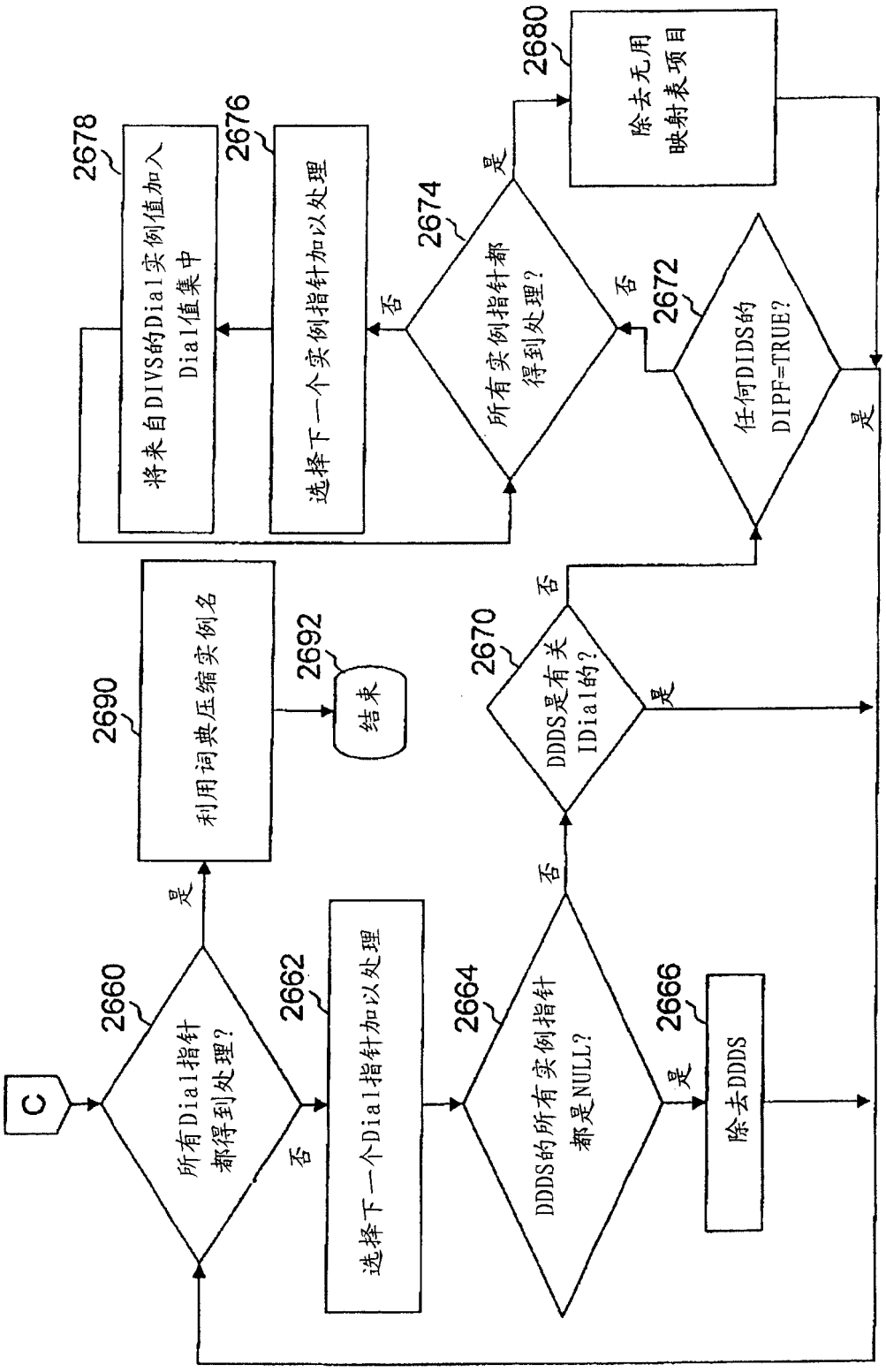

C
2660
所有Dial指针都得到处理?
是
2690
利用词典压缩实例名
2692
结束
否
2662
选择下一个Dial指针加以处理
2664
DDDS的所有实例指针都是NULL?
否
2666
是
除去DDDS
2670
DDDS是有关IDial的?
否
是
2672
任何DIDS的DIPF=TRUE?
否
2674
所有实例指针都得到处理?
否
2676
选择下一个实例指针加以处理
2678
将来自DIVS的Dial实例值加入Dial值集中
是
2680
除去无用映射表项目

图 26C

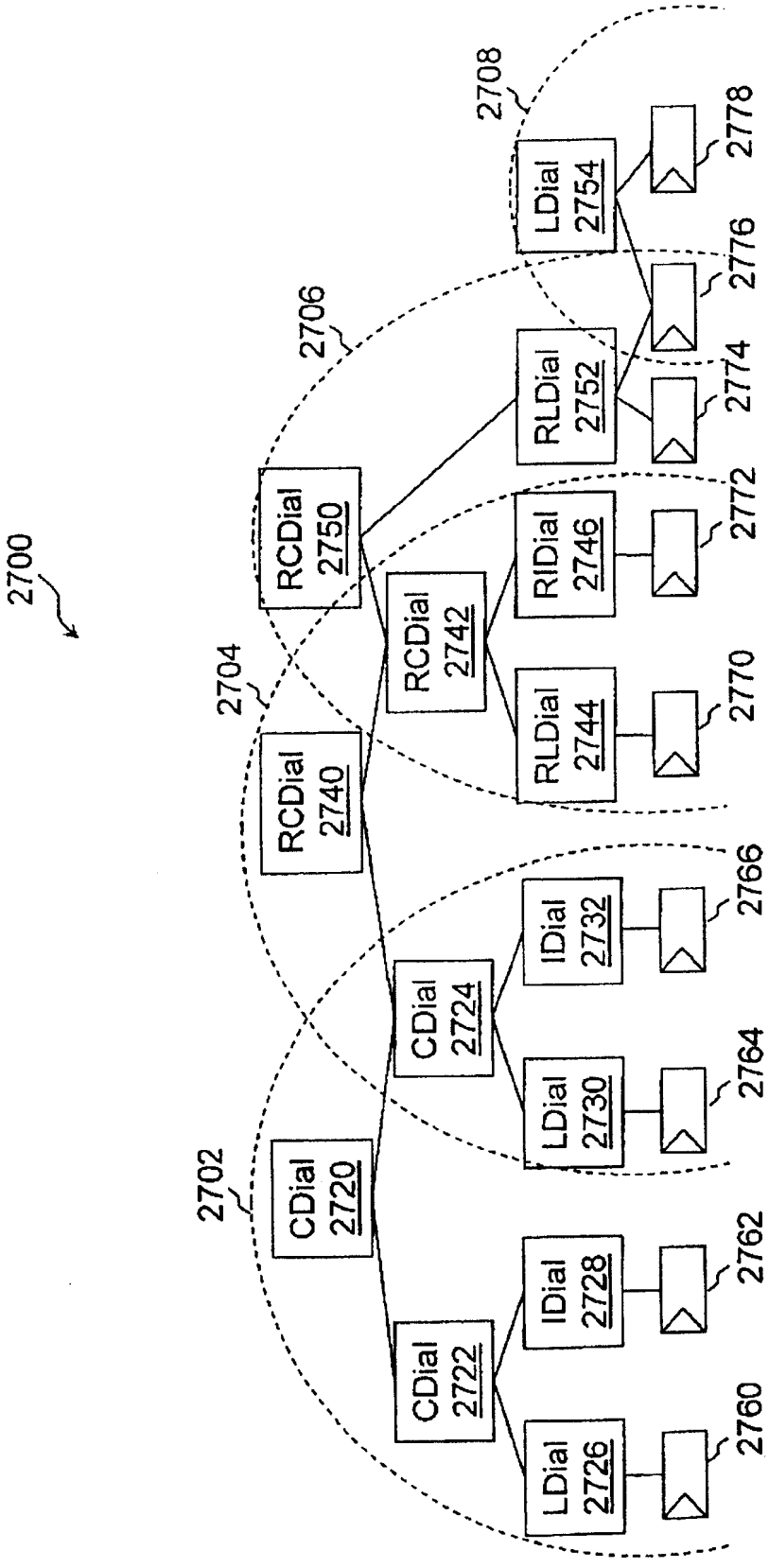

2700
2702
2704
2706
2708
CDial 2720
CDial 2722
CDial 2724
RCDial 2740
RCDial 2742
RCDial 2750
IDial 2728
IDial 2732
LDial 2726
LDial 2730
RIDial 2746
RLDial 2744
RLDial 2752
LDial 2754
2760
2762
2764
2766
2770
2772
2774
2776
2778

图 27

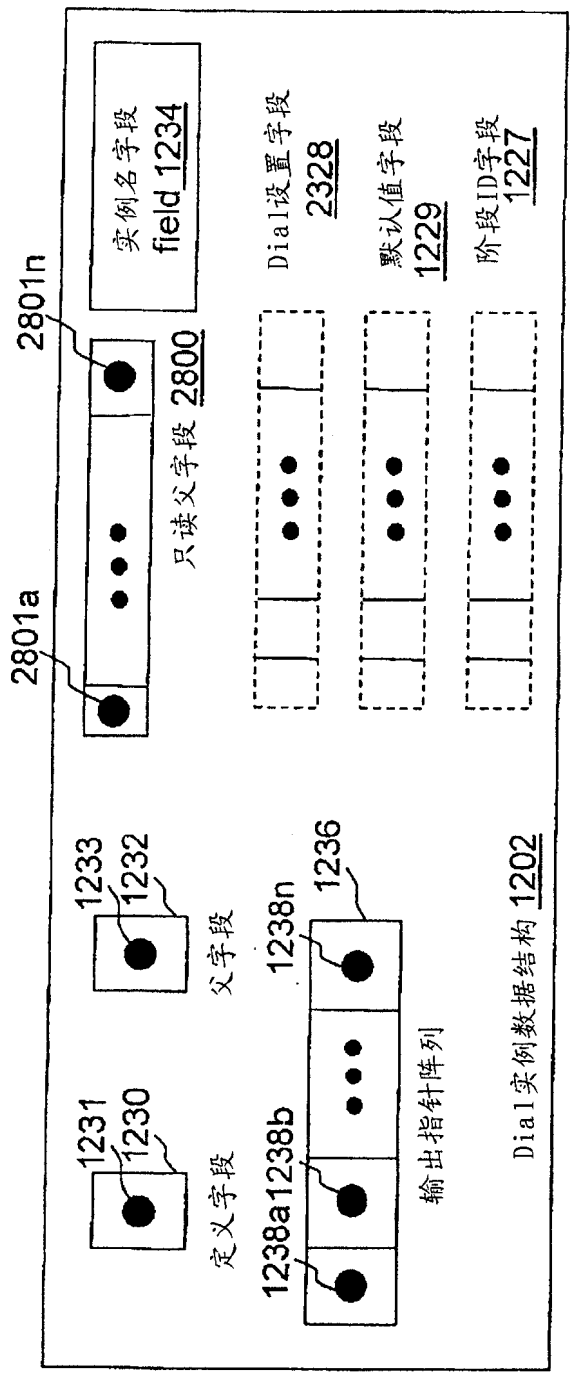

2801a
2801n
实例名字段
field 1234
只读父字段 2800
Dial设置字段
2328
默认值字段
1229
阶段ID字段
1227
1231
1230
定义字段
1233
1232
父字段
1238a
1238b
1238n
1236
输出指针阵列
Dial实例数据结构 1202

图 28A

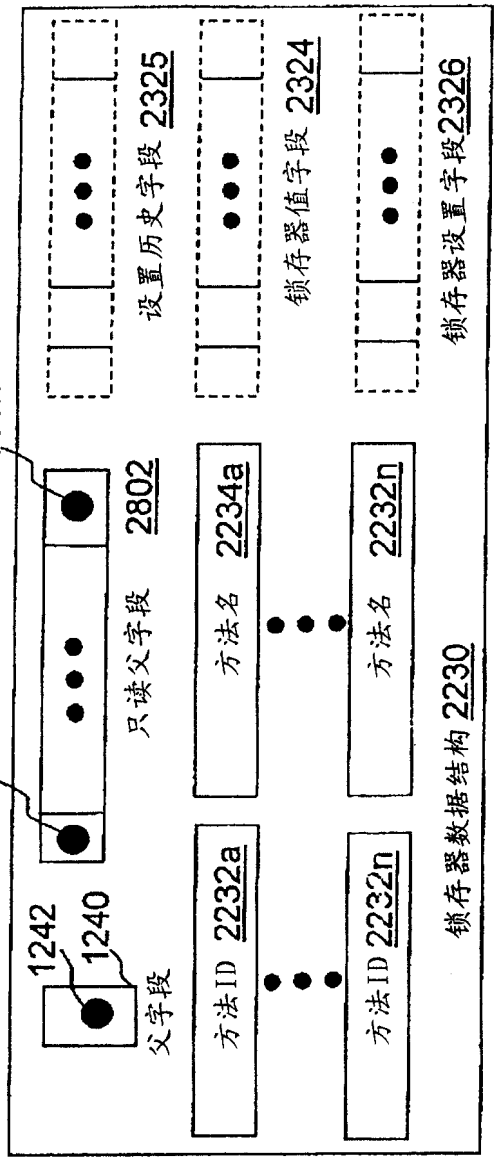

2803a
2803n
1242
1240
父字段
只读父字段 2802
设置历史字段 2325
锁存器值字段 2324
锁存器设置字段2326
方法ID 2232a
方法ID 2232n
方法名 2234a
方法名 2232n
锁存器数据结构 2230

图 28B

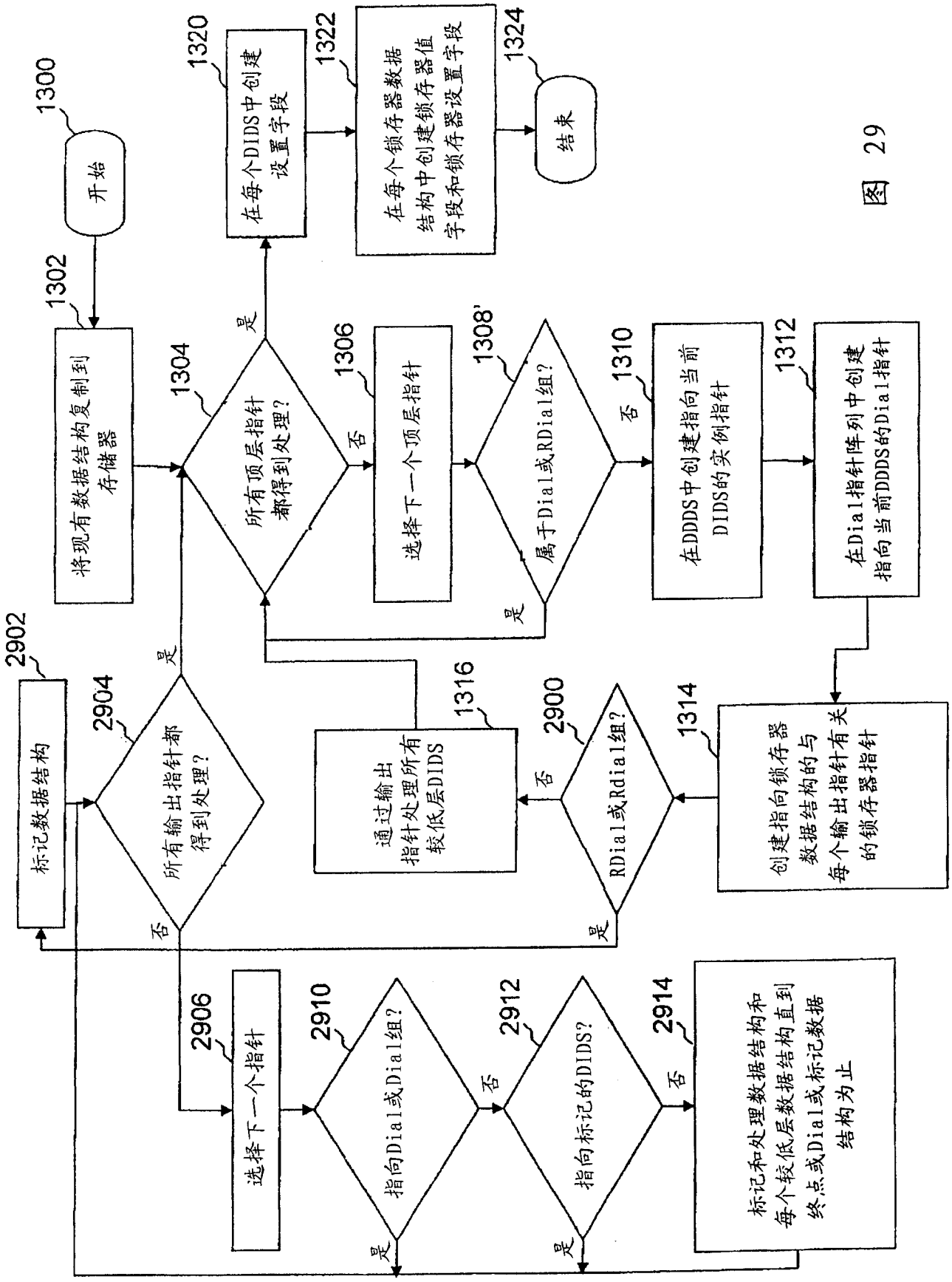

1300
开始
1302
将现有数据结构复制到存储器
1304
所有顶层指针都得到处理?
是
否
1306
选择下一个顶层指针
1308'
属于Dial或RDial组?
是
否
1310
在DDDS中创建指向当前DIDS的实例指针
1312
在Dial指针阵列中创建指向当前DDDS的Dial指针
1314
创建指向锁存器数据结构的与每个输出指针有关的锁存器指针
1316
通过输出指针处理所有较低层DIDS
1320
在每个DIDS中创建设置字段
1322
在每个锁存器数据结构中创建锁存器值字段和锁存器设置字段
1324
结束
2900
RDial或Rdial组?
是
否
2902
标记数据结构
2904
所有输出指针都得到处理?
是
否
2906
选择下一个指针
2910
指向Dial或Dial组?
是
否
2912
指向标记的DIDS?
是
否
2914
标记和处理数据结构和每个较低层数据结构直到终点或Dial或标记数据结构为止

图 29

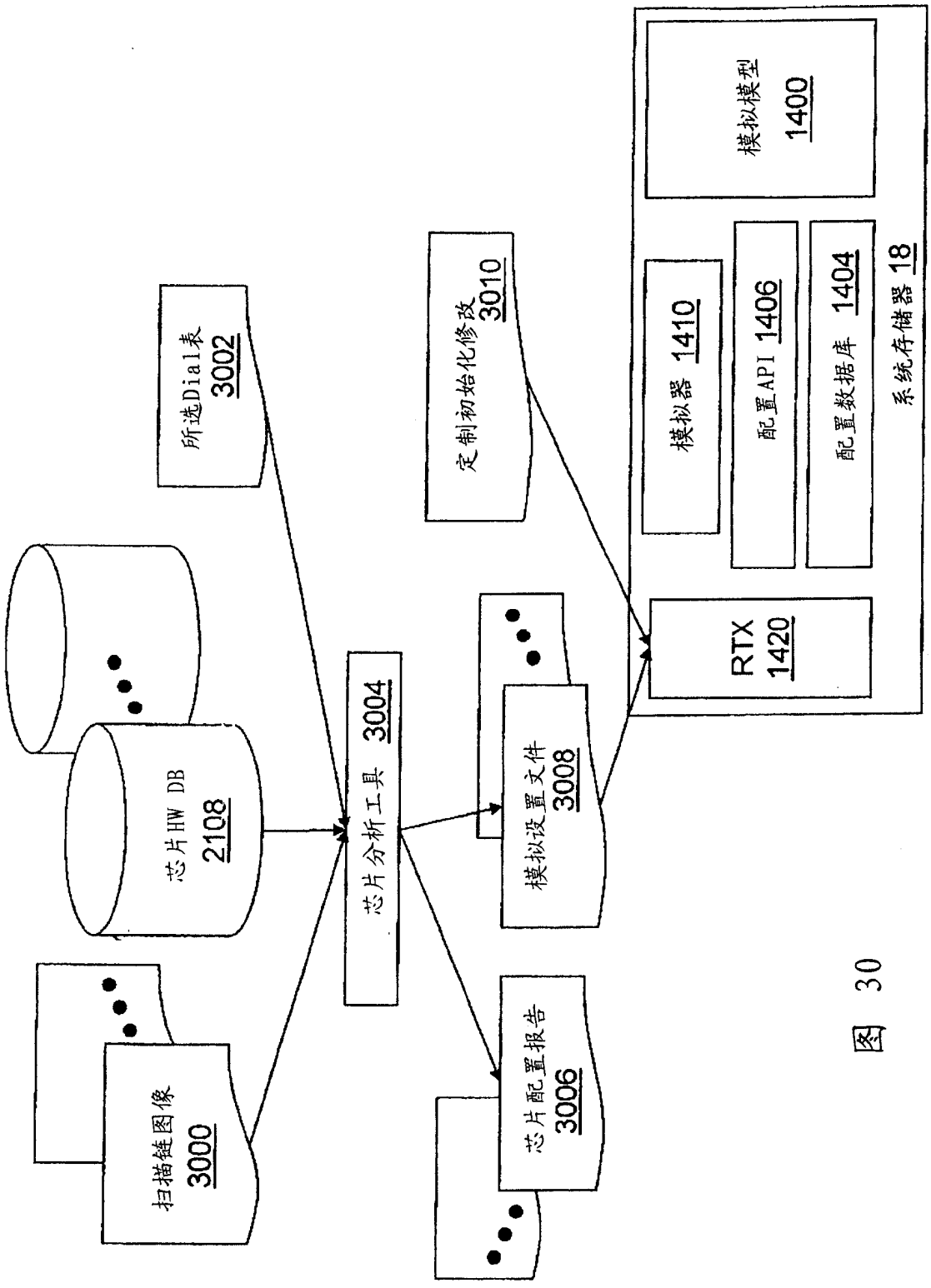

扫描链图像
3000
芯片HW DB
2108
所选Dial表
3002
芯片分析工具 3004
芯片配置报告
3006
模拟设置文件
3008
定制初始化修改
3010
系统存储器 18
RTX
1420
模拟器 1410
配置API 1406
配置数据库 1404
模拟模型
1400

图 30

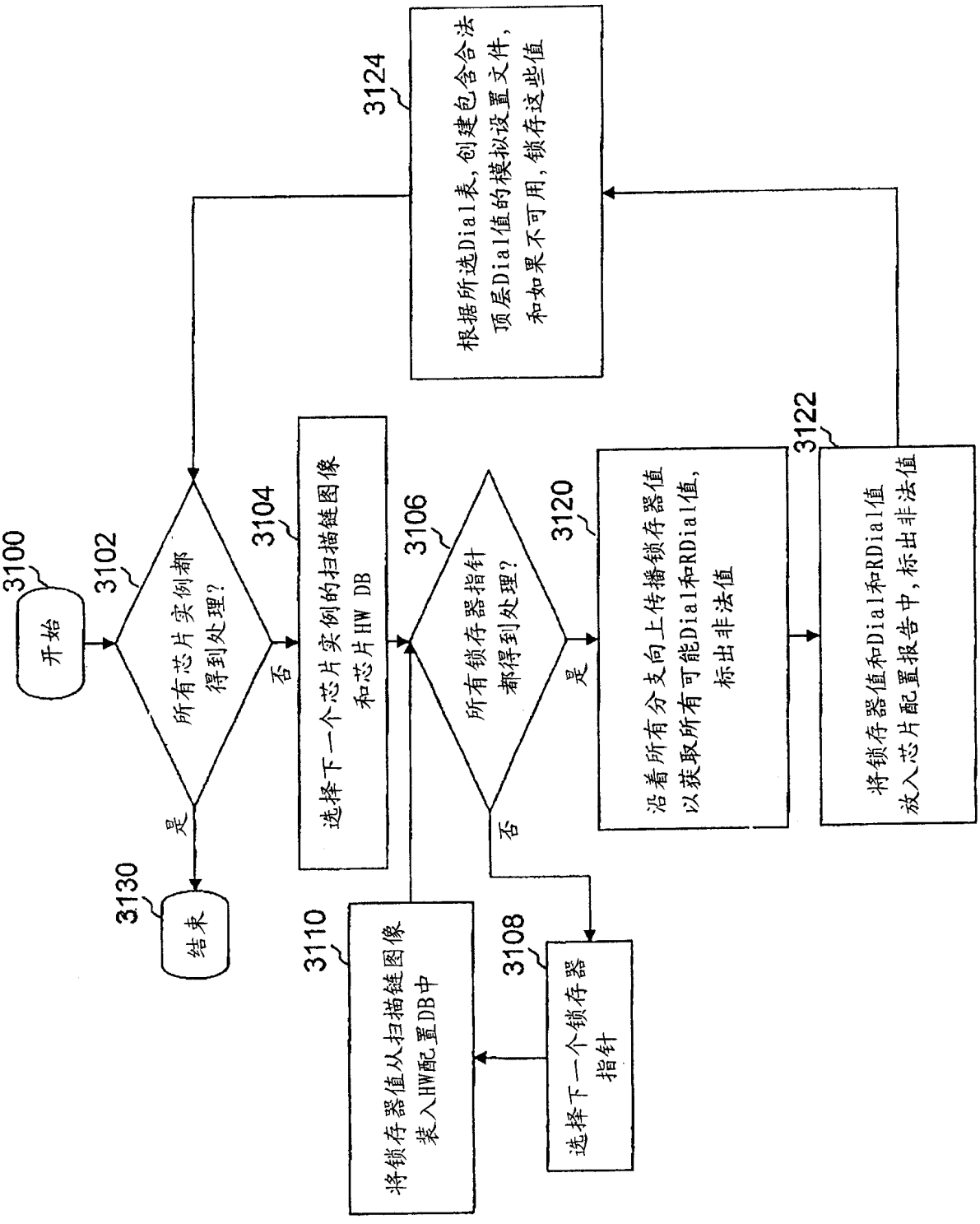
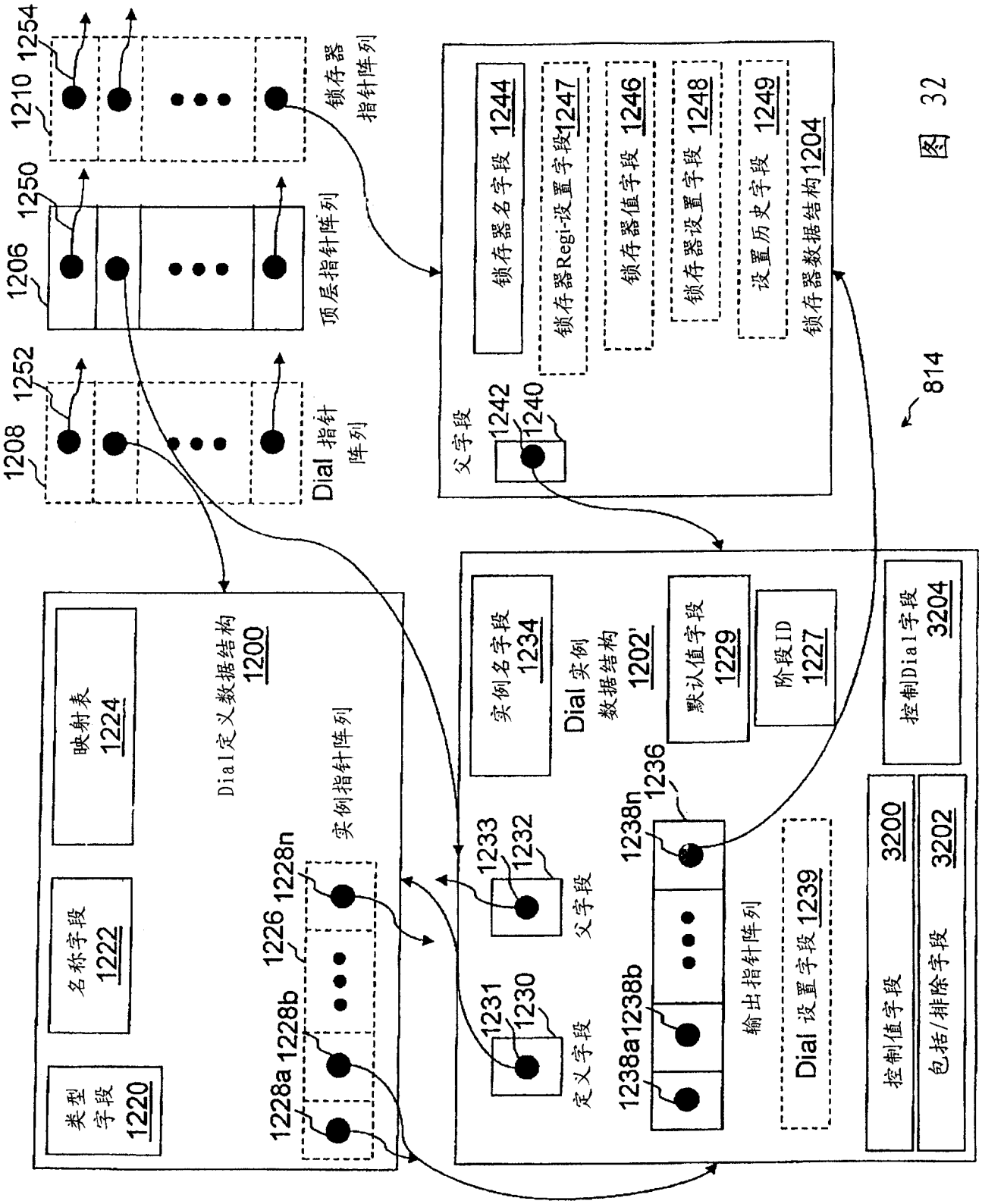

3100
开始
3102
所有芯片实例都得到处理?
是
否
3130
结束
3104
选择下一个芯片实例的扫描链图像和芯片HW DB
3106
所有锁存器指针都得到处理?
否
是
3110
将锁存器值从扫描链图像装入HW配置DB中
3108
选择下一个锁存器指针
3120
沿着所有分支向上传播锁存器值以获取所有可能Dial和RDial值，标出非法值
3122
将锁存器值和Dial和RDial值放入芯片配置报告中，标出非法值
3124
根据所选Dial表，创建包含合法顶层Dial值的模拟设置文件，和如果不可用，锁存这些值

图 31


1254
1210
锁存器指针阵列
1250
1206
顶层指针阵列
1252
1208
Dial 指针阵列
父字段
1242
1240
锁存器名字段 1244
锁存器Regi-设置字段1247
锁存器值字段 1246
锁存器设置字段 1248
设置历史字段 1249
锁存器数据结构1204
814
图 32
映射表 1224
名称字段 1222
类型字段 1220
Dial定义数据结构 1200
实例指针阵列
1228a 1228b 1226 1228n
1231 1230
定义字段
1233 1232
父字段
实例名字段 1234
Dial 实例数据结构 1202'
1238n 1236
1238a 1238b
输出指针阵列
默认值字段 1229
阶段ID 1227
Dial 设置字段 1239
控制值字段 3200
包括/排除字段 3202
控制Dial字段 3204

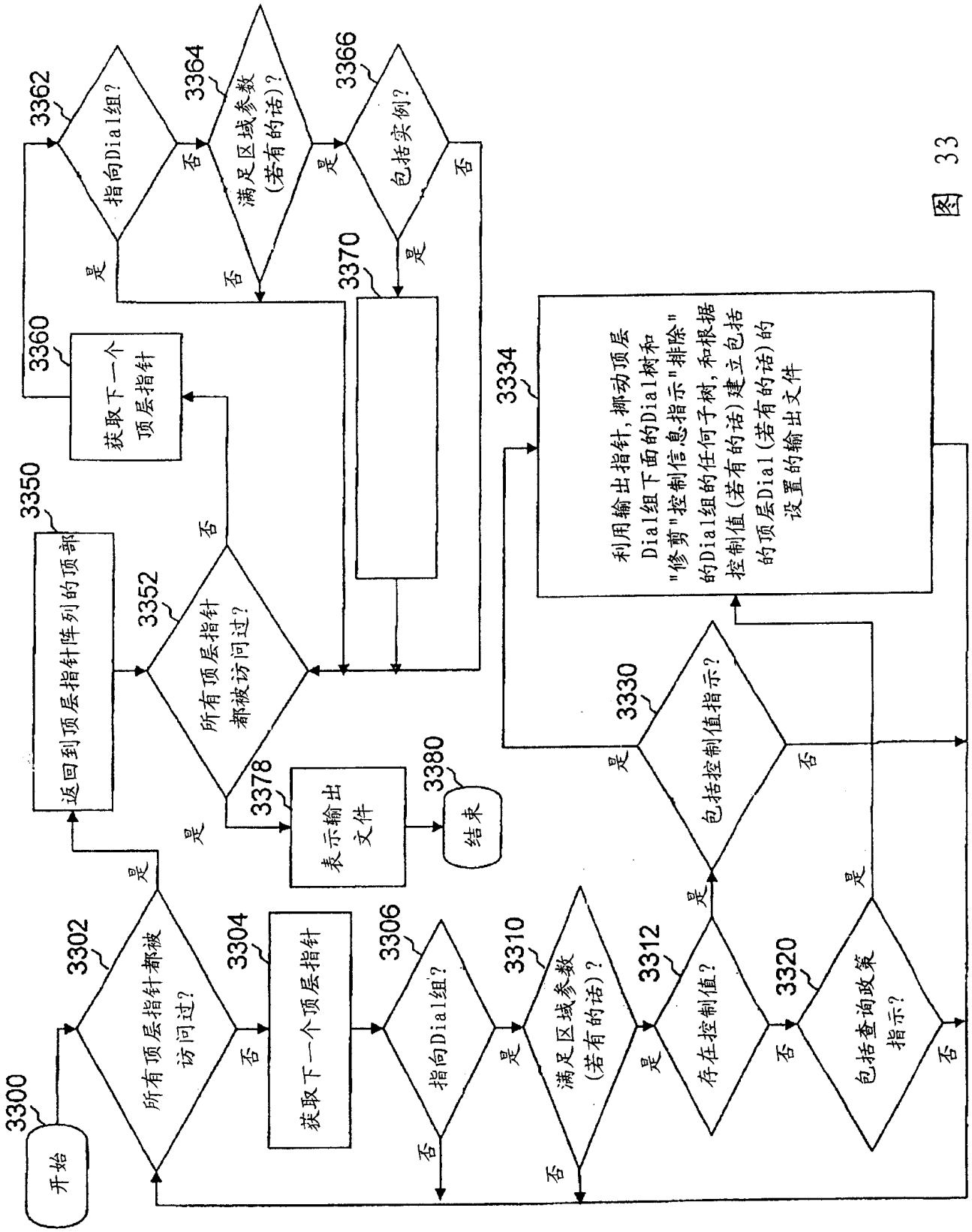

3300
开始
3302
所有顶层指针都被访问过?
否
是
3304
获取下一个顶层指针
3306
指向Dial组?
否
是
3310
满足区域参数(若有的话)?
否
是
3312
存在控制值?
否
是
3320
包括查询政策指示?
否
是
3330
包括控制值指示?
否
是
3334
利用输出指针,挪动顶层Dial组下面的Dial树和"修剪"控制信息指示"排除"的Dial组的任何子树,和根据控制值(若有的话)建立包括的顶层Dial(若有的话)的设置的输出文件
3350
返回到顶层指针阵列的顶部
3352
所有顶层指针都被访问过?
否
是
3360
获取下一个顶层指针
3362
指向Dial组?
否
是
3364
满足区域参数(若有的话)?
否
是
3366
包括实例?
否
是
3370
3378
表示输出文件
3380
结束

图 33

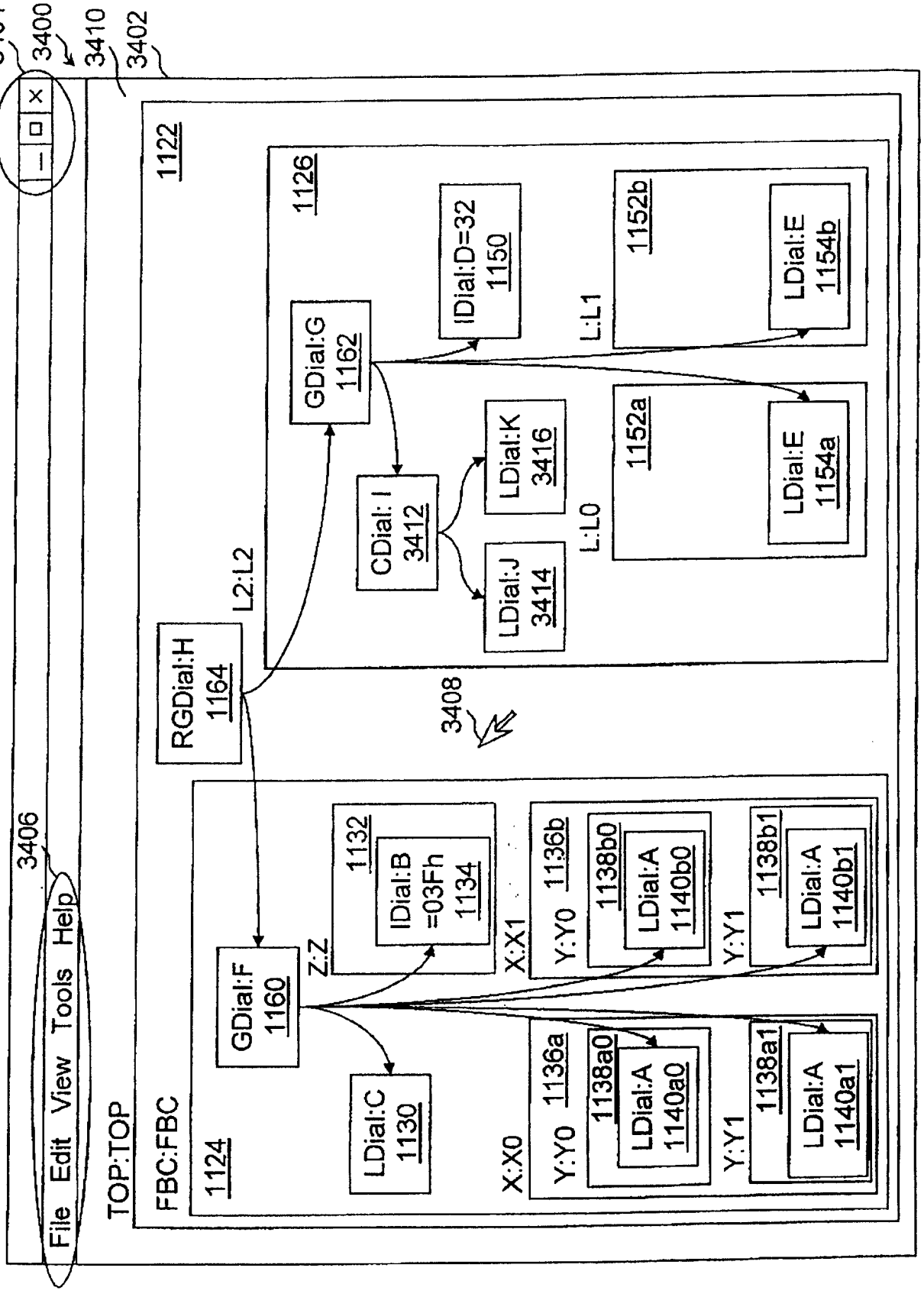

3404
3400
3410
3402
3406
File Edit View Tools Help
TOP:TOP
1122
FBC:FBC
1124
1126
RGDial:H
1164
L2:L2
GDial:F
1160
GDial:G
1162
IDial:D=32
1150
1152b
LDial:E
1154b
L:L1
CDial: I
3412
LDial:K
3416
LDial:J
3414
1152a
LDial:E
1154a
L:L0
3408
1132
IDial:B
=03Fh
1134
Z:Z
LDial:C
1130
X:X1
X:X0
1136b
1136a
Y:Y0
Y:Y0
1138b0
LDial:A
1140b0
1138a0
LDial:A
1140a0
Y:Y1
Y:Y1
1138b1
LDial:A
1140b1
1138a1
LDial:A
1140a1

图 34A

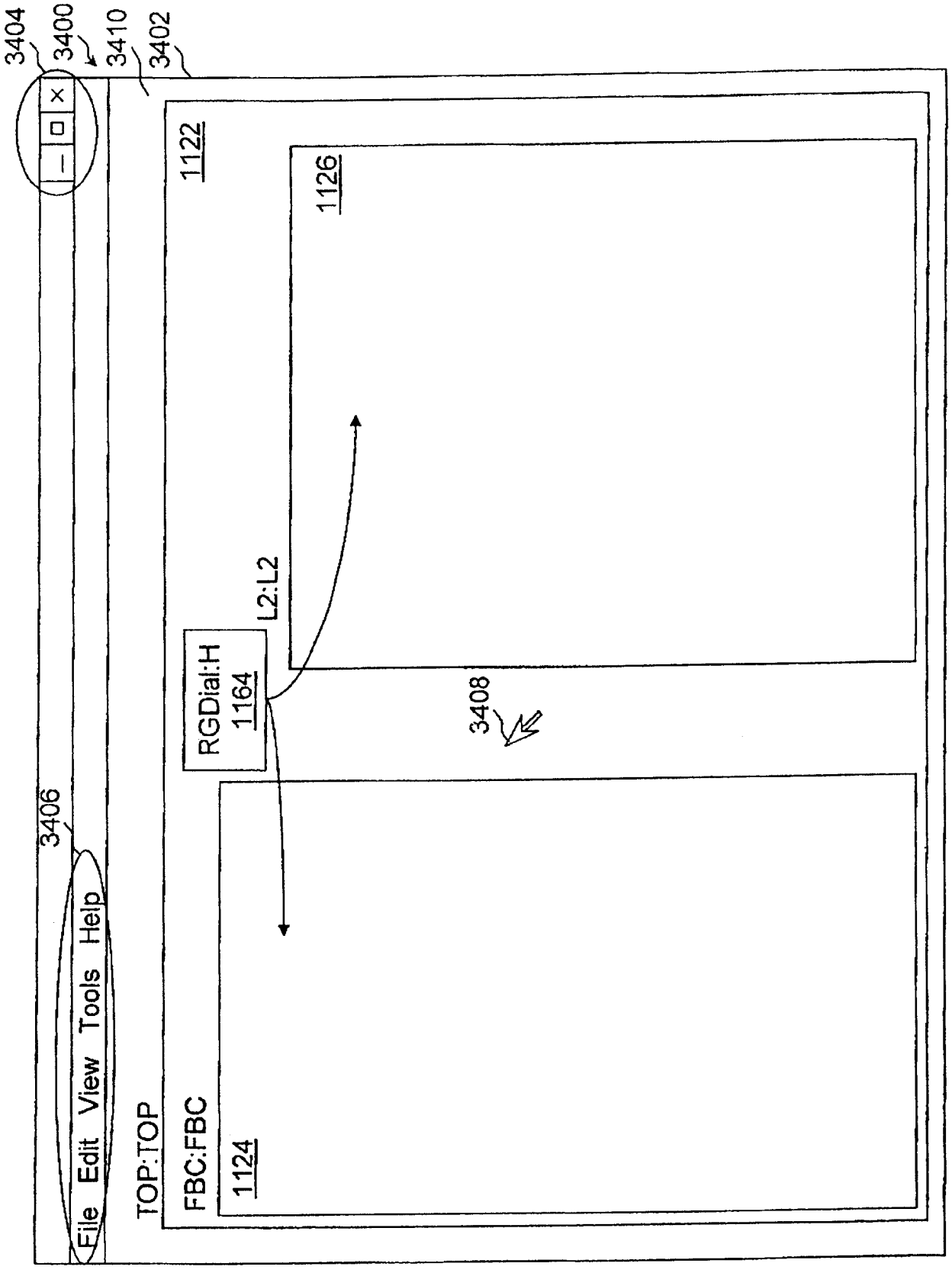

3404
3400
3410
3402
1122
1126
L2:L2
RGDial:H
1164
3408
3406
File Edit View Tools Help
TOP:TOP
FBC:FBC
1124

图 34B

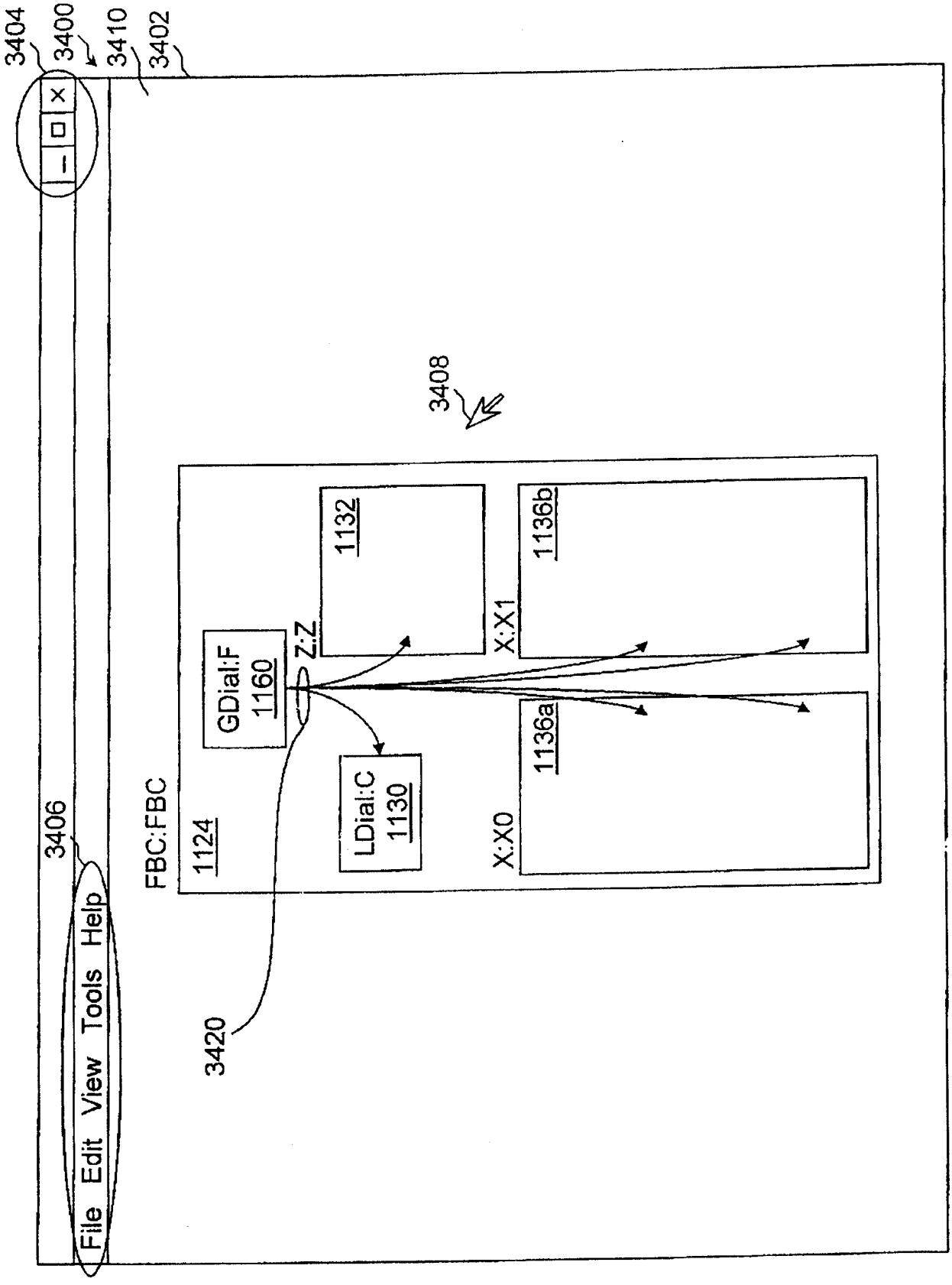

3404
3400
3410
3402
3408
3406
File Edit View Tools Help
FBC:FBC
1124
GDial:F
1160
Z:Z
LDial:C
1130
1132
X:X1
1136b
X:X0
1136a
3420

图 34C

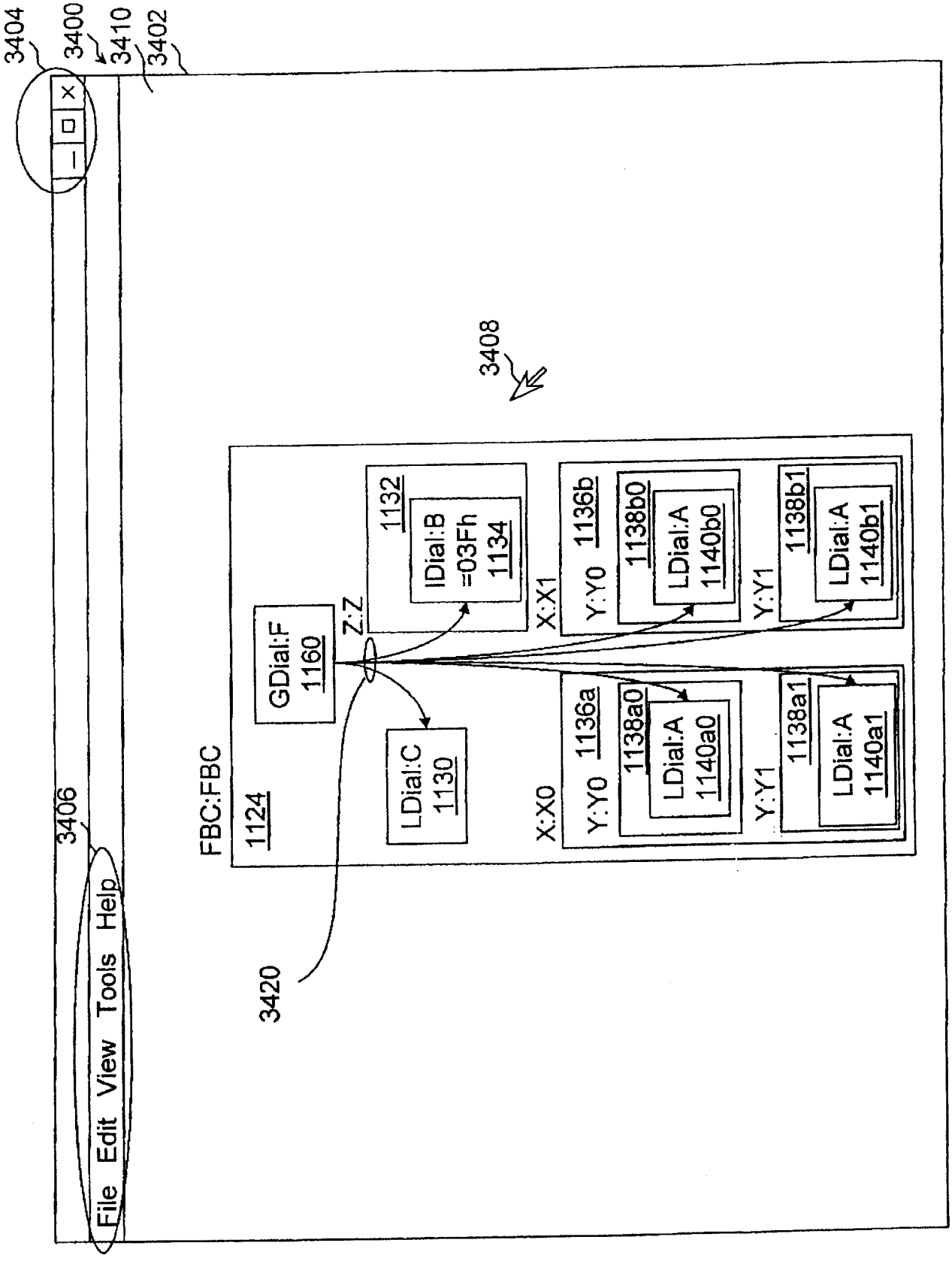

3404
3400
3410
3402
3406
File Edit View Tools Help
3408
FBC:FBC
1124
GDial:F
1160
Z:Z
1132
IDial:B
=03Fh
1134
LDial:C
1130
X:X1
Y:Y0 1136b
1138b0
LDial:A
1140b0
Y:Y1
1138b1
LDial:A
1140b1
X:X0
Y:Y0 1136a
1138a0
LDial:A
1140a0
Y:Y1
1138a1
LDial:A
1140a1
3420

图 34D

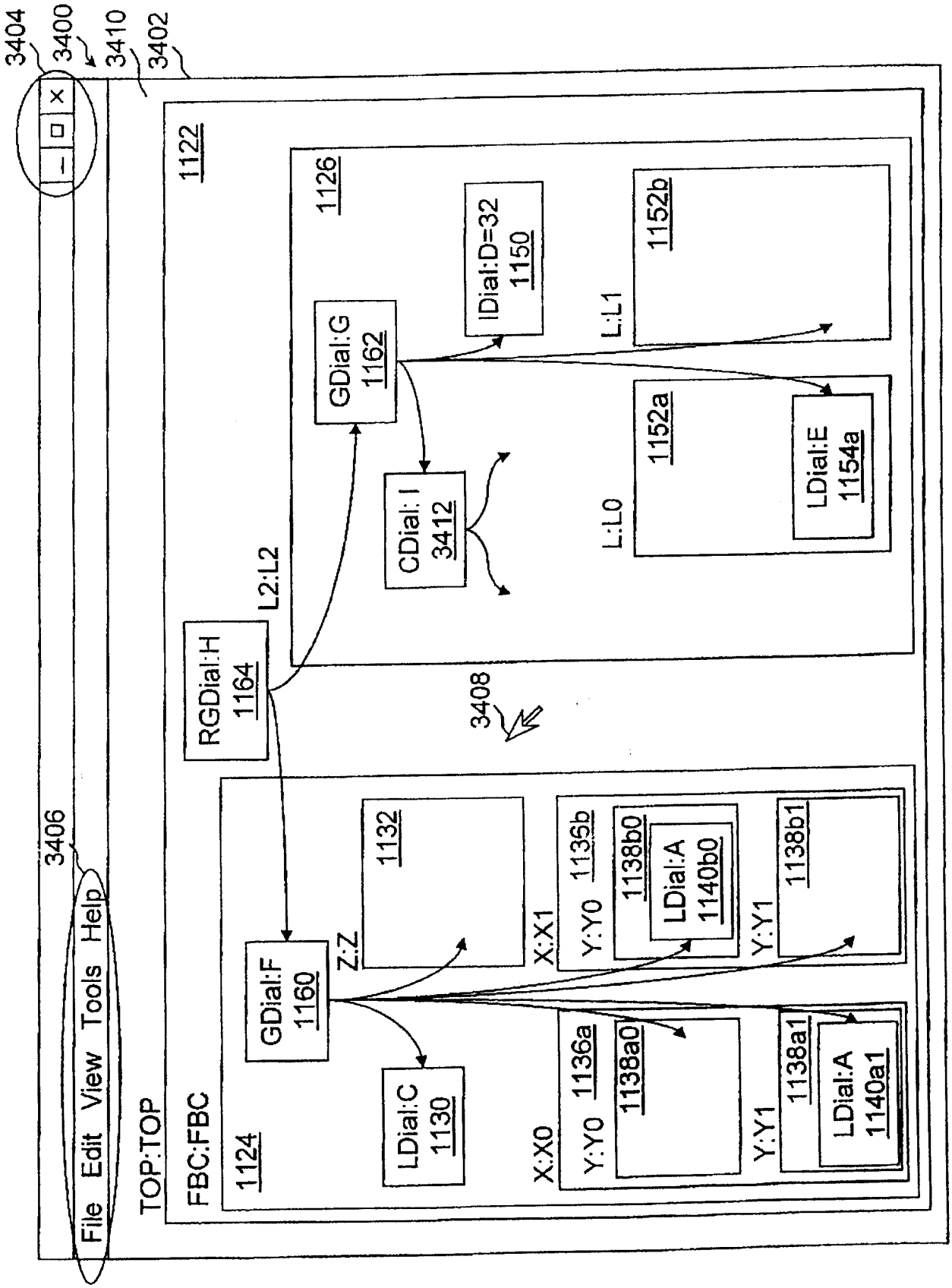

3404
3400
3410
3402
3406
File Edit View Tools Help
TOP:TOP
1122
FBC:FBC
1124
1126
RGDial:H
1164
L2:L2
GDial:F
1160
GDial:G
1162
IDial:D=32
1150
CDial: I
3412
LDial:C
1130
Z:Z
1132
X:X0
X:X1
Y:Y0
1136a
1138a0
1136b
1138b0
LDial:A
1140b0
Y:Y1
1138a1
LDial:A
1140a1
1138b1
Y:Y1
L:L1
1152b
L:L0
1152a
LDial:E
1154a
3408

图 34E

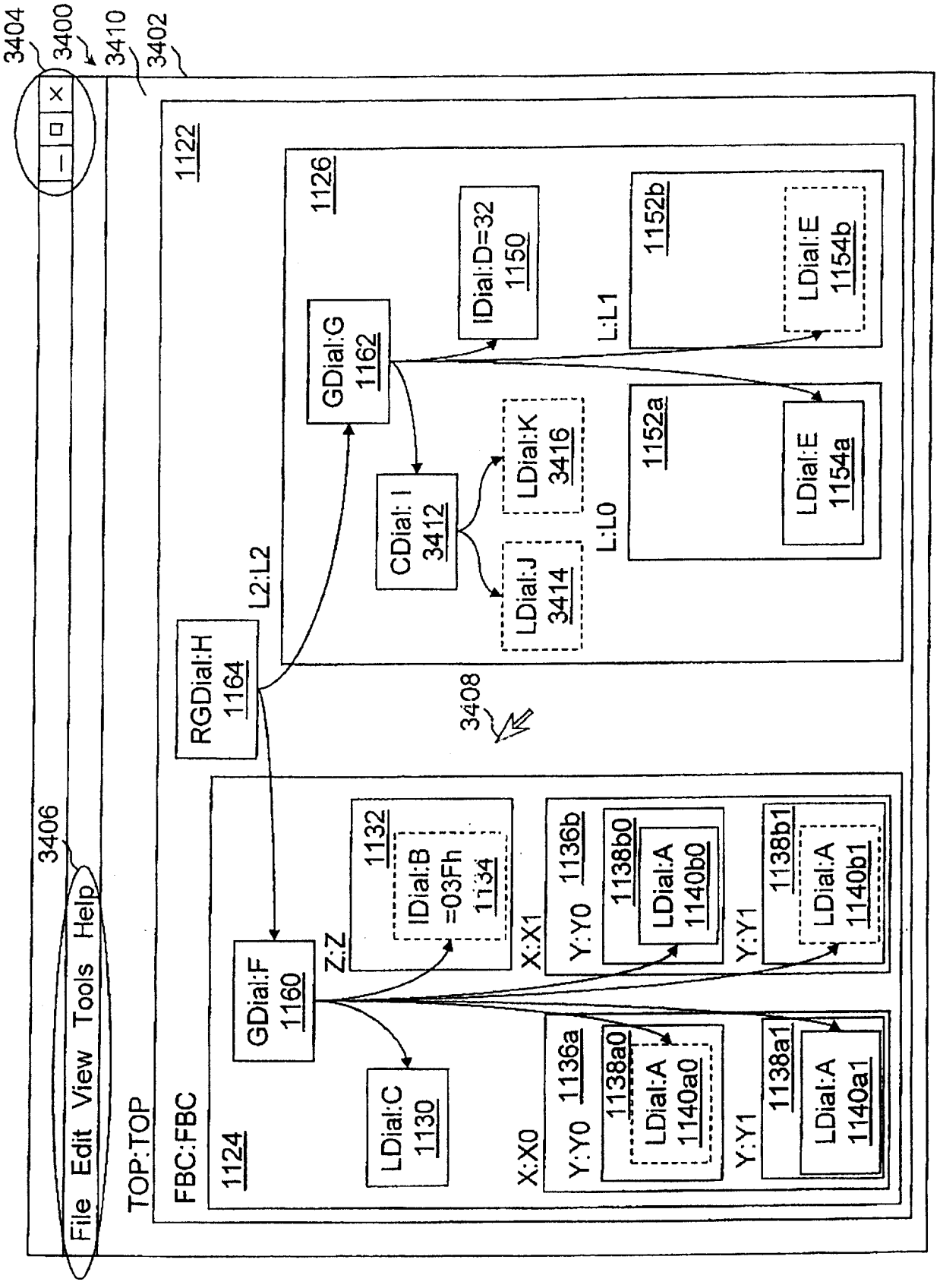

3404
3400
3410
3402
3406
File Edit View Tools Help
TOP:TOP
1122
FBC:FBC
1124
1126
RGDial:H
1164
L2:L2
GDial:G
1162
IDial:D=32
1150
L:L1
1152b
LDial:E
1154b
CDial: I
3412
LDial:K
3416
LDial:J
3414
L:L0
1152a
LDial:E
1154a
3408
GDial:F
1160
LDial:C
1130
Z:Z
1132
IDial:B
=03Fh
1134
X:X1
Y:Y0
1136b
1138b0
LDial:A
1140b0
Y:Y1
1138b1
LDial:A
1140b1
X:X0
Y:Y0
1136a
1138a0
LDial:A
1140a0
Y:Y1
1138a1
LDial:A
1140a1

图 34F