



(12) 发明专利

(10) 授权公告号 CN 102598001 B

(45) 授权公告日 2014. 09. 10

(21) 申请号 201080046143. 6

(56) 对比文件

(22) 申请日 2010. 10. 05

US 2005/0268265 A1, 2005. 12. 01,
US 2004/0044510 A1, 2004. 03. 04,
US 2002/0040465 A1, 2002. 04. 04,
US 2002/0040465 A1, 2002. 04. 04,
CN 100543757 C, 2009. 09. 23,

(30) 优先权数据

12/580, 330 2009. 10. 16 US

审查员 边臻

(85) PCT国际申请进入国家阶段日

2012. 04. 13

(86) PCT国际申请的申请数据

PCT/EP2010/064814 2010. 10. 05

(87) PCT国际申请的公布数据

W02011/045203 EN 2011. 04. 21

(73) 专利权人 国际商业机器公司

地址 美国纽约

(72) 发明人 M · L · 卡斯 R · L · 坎茨尔曼

J · R · 鲍姆加特纳 H · 莫尼

(74) 专利代理机构 北京市中咨律师事务所

11247

代理人 于静 张亚非

(51) Int. Cl.

G06F 17/50 (2006. 01)

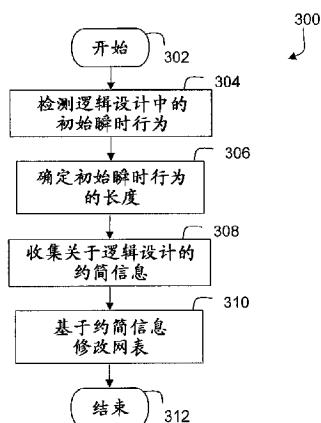
权利要求书3页 说明书17页 附图2页

(54) 发明名称

用于执行对逻辑设计的分析的方法和系统

(57) 摘要

一种用于执行对逻辑设计的分析的技术，包括检测体现于网表内的逻辑设计中的初始瞬时行为。还确定所述初始瞬时行为的持续时间。基于所述初始瞬时行为收集关于所述逻辑设计的约简信息。然后基于所述约简信息修改所述网表。



1. 一种用于执行对逻辑设计的分析的方法,所述方法包括 :

使用数据处理系统的一个或多个处理器来检测 (304) 体现在网表内的逻辑设计中的初始瞬时行为 ;

使用所述一个或多个处理器中的至少一个处理器来确定 (306) 所述初始瞬时行为的持续时间 ;

使用所述一个或多个处理器中的至少一个处理器来基于所述初始瞬时行为收集 (308) 关于所述逻辑设计的约简信息 ;以及

使用所述一个或多个处理器中的至少一个处理器来基于所述约简信息修改 (310) 所述网表,

所述方法还包括 :

使用所述数据处理系统将所述逻辑设计的验证问题分解为第一阶段和第二阶段,其中所述第一阶段对应于启动阶段,并且所述第二阶段对应于在所述启动阶段之后发生的阶段 ;以及

使用所述一个或多个处理器中的至少一个处理器基于知晓所述逻辑设计未在所述第二阶段中展现瞬时初始行为而最小化所述验证问题的所述第二阶段。

2. 如权利要求 1 的方法,其中所述初始瞬时行为与一个或多个瞬时信号关联。

3. 如权利要求 1 的方法,其中所述初始瞬时行为与一个或多个初始化输入关联。

4. 如权利要求 1 的方法,其中使用三元仿真来检测所述逻辑设计中的所述初始瞬时行为。

5. 如权利要求 1 的方法,其中修改所述网表包括 :从所述逻辑设计消除瞬时逻辑。

6. 如权利要求 5 的方法,其中消除瞬时逻辑包括 :将无界验证问题分解为在所述初始瞬时行为所发生的初始时间帧内的有界验证问题和在所述初始时间帧之后的后瞬时时间帧内的无界验证问题。

7. 如权利要求 6 的方法,还包括 :

通过将所有瞬时视为已稳定于后瞬时恒定值而简化在所述后瞬时时间帧内的所述无界验证问题。

8. 如权利要求 5 的方法,还包括 :

结合结构分析而使用有界模型检查以在消除所述瞬时逻辑之后识别初始化输入。

9. 如权利要求 1 的方法,其中使用对所述网表的行为进行过度近似的三元仿真来检测所述瞬时初始行为。

10. 如权利要求 1 的方法,还包括 :

使用所述一个或多个处理器中的至少一个处理器将所述网表时移最大瞬时持续时间,以促进在所述瞬时初始行为之后检查所述逻辑设计。

11. 如权利要求 1 的方法,其中所述分析对应于形式验证、仿真、硬件加速或合成中的一个。

12. 一种用于执行对逻辑设计的分析的系统,所述系统包括 :

用于使用数据处理系统的一个或多个处理器来检测 (304) 体现在网表内的逻辑设计中的初始瞬时行为的装置 ;

用于使用所述一个或多个处理器中的至少一个处理器来确定 (306) 所述初始瞬时行

为的持续时间的装置；

用于使用所述一个或多个处理器中的至少一个处理器来基于所述初始瞬时行为收集
(308) 关于所述逻辑设计的约简信息的装置；以及

用于使用所述一个或多个处理器中的至少一个处理器来基于所述约简信息修改 (310)
所述网表的装置，

所述系统还包括：

用于使用所述数据处理系统将所述逻辑设计的验证问题分解为第一阶段和第二阶段的装置，其中所述第一阶段对应于启动阶段，并且所述第二阶段对应于在所述启动阶段之后发生的阶段；以及

用于使用所述一个或多个处理器中的至少一个处理器基于知晓所述逻辑设计未在所述第二阶段中展现瞬时初始行为而最小化所述验证问题的所述第二阶段的装置。

13. 如权利要求 12 的系统，其中所述初始瞬时行为与一个或多个瞬时信号关联。

14. 如权利要求 12 的系统，其中所述初始瞬时行为与一个或多个初始化输入关联。

15. 如权利要求 12 的系统，其中使用三元仿真来检测所述逻辑设计中的所述初始瞬时行为。

16. 如权利要求 12 的系统，其中修改所述网表包括：从所述逻辑设计消除瞬时逻辑。

17. 如权利要求 16 的系统，其中消除瞬时逻辑包括：将无界验证问题分解为在所述初始瞬时行为所发生的初始时间帧内的有界验证问题和在所述初始时间帧之后的后瞬时时间帧内的无界验证问题。

18. 如权利要求 17 的系统，还包括：

用于通过将所有瞬时视为已稳定于后瞬时恒定值而简化在所述后瞬时时间帧内的所述无界验证问题的装置。

19. 如权利要求 16 的系统，还包括：

用于结合结构分析而使用有界模型检查以在消除所述瞬时逻辑之后识别初始化输入的装置。

20. 如权利要求 12 的系统，其中使用对所述网表的行为进行过度近似的三元仿真来检测所述瞬时初始行为。

21. 如权利要求 12 的系统，还包括：

用于使用所述一个或多个处理器中的至少一个处理器将所述网表时移最大瞬时持续时间，以促进在所述瞬时初始行为之后检查所述逻辑设计的装置。

22. 如权利要求 12 的系统，其中所述分析对应于形式验证、仿真、硬件加速或合成中的一个。

23. 一种用于执行对逻辑设计的分析的系统，所述系统包括：

存储子系统；以及

与所述存储子系统耦合的一个或多个处理器，其中所述一个或多个处理器配置为：

检测体现在网表内的逻辑设计中的初始瞬时行为；

确定所述初始瞬时行为的持续时间；

基于所述初始瞬时行为收集关于所述逻辑设计的约简信息；以及

基于所述约简信息修改所述网表，其中修改所述网表以消除瞬时逻辑和初始化输入，

所述系统还包括：

用于使用所述数据处理系统将所述逻辑设计的验证问题分解为第一阶段和第二阶段的装置，其中所述第一阶段对应于启动阶段，并且所述第二阶段对应于在所述启动阶段之后发生的阶段；以及

用于使用所述一个或多个处理器中的至少一个处理器基于知晓所述逻辑设计未在所述第二阶段中展现瞬时初始行为而最小化所述验证问题的所述第二阶段的装置。

用于执行对逻辑设计的分析的方法和系统

技术领域

[0001] 本发明一般地涉及集成电路逻辑设计分析,更具体地说,涉及用于分析具有瞬时逻辑的集成电路逻辑设计的技术。

背景技术

[0002] 通常,形式验证涉及严谨地证明集成电路(IC)逻辑设计(设计)满足相关联的规范。典型地,验证问题的规范包括设计的网表表示和所述网表的指定网的一组预期值。“网表”包含具有各种功能的门(其评估随时间推移的布尔值)和边(其表示所述门之间的互连)。“迹线”可以是门随时间推移的二进制(即,“0”或“1”)值的序列或门随时间推移的三进位值(即,“0”、“1”或“X”,其中值“X”指未知值)的序列。

[0003] 例如,门可属于四个广义功能种类之一:恒定门、随机门、组合门以及状态元件(例如,寄存器和顺序门(sequential gate),诸如锁存器和触发器)。恒定门产生不随时间推移而变化的逻辑电平。随机门(亦称为主输入)可以在任何时步中采用独立于所有其他门的任何逻辑电平。组合门是诸如“与”门、“或”门、“与非”门、“或非”门之类的逻辑元件。顺序门具有关联的初始值函数和下一状态函数。顺序门在时间“0”(t_0)处的值为初始值函数的值。顺序门在时间“ $i+1$ ”处的值为顺序门的下一状态函数在时间“ i ”处的值。

[0004] 作为一个实例,验证问题可包括判定是否存在其中断言特定信号的状态,其中该特定信号的断言指示错误。使用形式验证,试图找到导致特定信号的断言的包括随时间(状态)推移的网值序列的反例迹线(counter-example trace),或试图证明不存在导致特定信号的断言的反例迹线。形式验证经常使用状态空间搜索算法来执行,所述状态空间搜索算法包括无界和有界穷举搜索算法。有界穷举搜索算法(Bounded exhaustive search algorithms)试图找到在从设计的初始状态开始的“N”个时步(time-steps)内发生的特定信号的断言。无界穷举搜索算法增大“N”直至不遇到对于“N”的较小值尚未遇到的状态为止(称为“固定点”的条件)。如果在达到固定点之前并没有从初始状态至违规状态(即,其中断言特定信号的状态)的路径,则可推断设计的正确性。

[0005] 执行穷举状态空间搜索所需的验证循环的数目随状态元件(例如,寄存器、锁存器、触发器等)的数目以指数方式增大。此指数关系使得形式验证对于含有大量状态元件(例如,一百或更多状态元件)的设计而言并不实用。因此,使用半形式验证作为大型设计的验证技术。半形式验证通过以资源定界方式将形式算法应用于大型设计而利用形式算法。虽然需要较少计算时间(与形式验证相比),但是半形式验证可能仅实现部分验证覆盖。

发明内容

[0006] 根据本发明的一个方面,一种用于执行对逻辑设计的分析的技术包括检测体现于网表内的逻辑设计中的初始瞬时行为。确定所述初始瞬时行为的持续时间。基于所述初始瞬时行为收集关于所述逻辑设计的约简信息。然后基于所述约简信息修改所述网表。

附图说明

[0007] 通过实例的方式例示了本发明并且本发明并非旨在受附图限制,在附图中,相同标号指示相同元件。为了简洁和清晰而在图中例示各元件并且所述元件的绘制不必成比例,这些附图是:

[0008] 图 1 是示出逻辑设计中的三个寄存器的状态的信号图;

[0009] 图 2 是可用于执行根据本发明配置的工具的实例计算机系统的图;

[0010] 图 3 是根据本发明的又一实施例的用于分析设计的实例过程的流程图。

具体实施方式

[0011] 所属技术领域的技术人员知道,本发明可以实现为方法、系统或计算机程序产品。因此,本发明可以具体实现为以下形式,即:可以是完全的硬件、也可以是完全的软件(包括固件、驻留软件、微代码等),还可以是硬件和软件结合的形式,本文一般称为“电路”、“模块”或“系统”。此外,本发明还可以实现为在计算机可用存储介质中的计算机程序产品的形式,该计算机可用存储介质中包含计算机可用程序代码。

[0012] 可以使用一个或多个计算机可用或计算机可读存储介质的任意组合。所述计算机可用或计算机可读存储介质例如可以是(但不限于)电子、磁、光、电磁、红外或半导体系统、装置或设备。计算机可读存储介质的更具体的实例(非穷举的列表)将包括以下项:便携式计算机软盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦写可编程只读存储器(EPROM)或闪存、便携式光盘只读存储器(CD-ROM)、光存储设备或磁存储设备。注意,所述计算机可用或计算机可读存储介质甚至可以是程序被打印在其上的纸张或其他适合的介质,因为所述程序可以通过例如光扫描所述纸张或其他介质被电子地捕获,然后被编译、解释或另外以适合的方式被处理(如果必要),然后被存储在计算机存储器中。在本发明的上下文中,计算机可用或计算机可读存储介质可以是任何能够包含或存储由指令执行系统、装置或设备使用或与指令执行系统、装置或设备有关的程序的介质。所述计算机可用介质可以包括其中包含计算机可用程序代码(在基带中或作为载波的一部分)的传播数据信号。可以使用任何适当的介质(包括但不限于无线、有线、光缆、射频(RF)等)来传输计算机可用程序代码。

[0013] 用于执行本发明的操作的计算机程序代码可以以面向对象的编程语言(如 Java、Smalltalk 或 C++)来编写。但是,用于执行本发明的操作的所述计算机程序代码也可以以传统的过程编程语言(如“C”编程语言或类似编程语言)来编写。

[0014] 参考根据本发明的实施例的方法、装置(系统)和计算机程序产品的方块图和/或流程图在下面描述了本发明。将理解,所述方块图和/或流程图的每个方块以及所述方块图和/或流程图中的方块的组合可以由计算机程序指令来实现。这些计算机程序指令可以被提供给通用计算机、专用计算机或其他可编程数据处理装置的处理器以产生一种机器,以便通过所述计算机和/或其他可编程数据处理装置的处理器执行的所述指令将创建用于实现所述方块图和/或流程图方块(多个)中指定的功能/操作的装置。

[0015] 这些计算机程序指令也可以被存储在能够以特定方式引导计算机或其他可编程数据处理装置执行功能的计算机可读存储器中,以便存储在所述计算机可读存储器中的所

述指令将产生一件包括实现在所述方块图和 / 或流程图方块（多个）中指定的功能 / 操作的指令的制品。

[0016] 所述计算机程序指令还可以被加载到计算机或其他可编程数据处理装置上以导致在所述计算机或其他可编程装置上执行一系列操作步骤以产生计算机实现的过程，以便在所述计算机或其他可编程装置上执行的所述指令将实现在所述方块图和 / 或流程图方块（多个）中指定的功能 / 操作。如本文中使用的，术语“耦合”包括块或组件之间的直接电连接以及块或组件之间使用一个或多个中间块或组件而实现的间接电连接。

[0017] 瞬时信号是在接通电源（或分析的开始）之后的固定数目个时步期间采用任意逻辑值的信号。在固定数目个时步之后，瞬时信号稳定并采用较具限制性的行为（例如，最初双态触发的门对于所有未来时间点稳定于恒定值，以不同方式初始化的两个门在时间点之后变得相等，或时钟分布树中的门可在最初若干时步不规律地行动且此后开始周期性地双态触发）。参看图 1，信号图 100 例示逻辑设计中的三个寄存器的状态。在图 1 中，对于指定时间处的所有可能输入值而言，寄存器在其逻辑赋值始终等于“0”时被指派“0”或在其逻辑赋值始终等于“1”时被指派“1”。如果三元仿真（ternary simulation）无法确定寄存器的值，则该寄存器被指派值“X”。在图 1 中，寄存器“C”在时间“2”处采用值“0”，且贯穿所例示的时步保持为“0”。当寄存器针对大于特定时间的所有时步采用恒定值时，该寄存器被称为“瞬时逻辑”。瞬时持续时间对应于在瞬时信号稳定于恒定值之前的时步的数目。如图 1 中例示的，寄存器“C”的瞬时持续时间为“2”。

[0018] 逻辑设计的瞬时行为可归因于各种原因。例如，逻辑设计的瞬时行为可以是手动逻辑设计的结果，因为保守的后期重设初始状态可造成此后无法被观测的特定量的瞬时设计行为。作为另一实例，逻辑设计的瞬时行为可由不允许寄存器具有固定初始状态（而替代地要求许多寄存器初始被解释为具有非确定性状态）的设计式样引起。在此情形下，如果设计中使用的重设机制无法保证某些寄存器的固定初始状态，则可将这些寄存器的初始状态解释为在时间“0”处具有任意的随机值。因为部分重设机制的实施通常导致减小的电路大小（与重设所有寄存器的架构相比），所以部分重设机制经常是所期望的。为了使逻辑设计（其实施部分重设机制）正确地起作用，经由初始化序列（其为逻辑设计所采取的一系列操作，在这些操作之后所有寄存器值变为确定性值）使得所述设计的大多数寄存器在特定数目个时间帧内为确定性状态。在初始化序列完成之前，寄存器采用非确定性的随机值。在初始化（启动）阶段期间所见的赋值可不同于可在初始化序列终止之后观测到的赋值。

[0019] 作为又一实例，逻辑设计的瞬时行为可以是在分析期间限制逻辑设计的行为的常见做法的结果。例如，逻辑设计经常具备定义设计的操作模式的配置输入（通常称为“紧急按钮（chicken switch）”）。在初始化以后，设计处理这些配置输入且接着稳定成指定操作模式。在期望操作模式中所见的寄存器赋值可不同于在初始化阶段中所见的赋值。此初始瞬时行为倾向于出于若干原因而致使对相应设计的分析并非最佳。例如，在形式验证中，初始瞬时寄存器赋值使表征一组可达到的状态的努力复杂化。另外，逻辑设计中的许多门可能仅在初始化阶段中需要，且如果可移除初始化阶段，则可使得逻辑设计较小。在仿真和硬件加速架构中，冗余门使建立模型（以及对该模型的后续分析）的过程显著变慢。在逻辑合成和设计中，可利用瞬时行为和冗余门信息来优化逻辑设计且增强所制造的相关半导体

器件的特性（例如，减小硅面积，减小功率消耗，以及增大时钟速度）。

[0020] 多种逻辑设计包括仅在初始时间帧期间具有相关性的瞬时逻辑。通常，瞬时逻辑可由以下项表示：在某一数目的时间帧之后稳定于确定性恒量的信号；以及用于列举复杂初始状态的初始化输入，其在初始状态之后变得无关。大部分逻辑设计（工业逻辑设计和基准逻辑设计两者）包括在分析期间产生开销的瞬时逻辑。根据本发明的各方面，实施自动技术以检测并消除瞬时逻辑。自动技术在较大的逻辑约简、较深的有界模型检查 (BMC) 及使用（例如）归纳和内插的增强的证明能力方面促进了验证效率。

[0021] 顺序硬件设计的自动验证是 PSPACE (即，可由图灵机 (Turing machine) 使用一定量的多项式空间求解的所有决策问题的集合) 问题，该问题经常是在计算上具有挑战性的任务。取决于处于验证下的逻辑设计的大小，自动解决方案可以是难处理的。许多逻辑（硬件）设计包括外来人为因素，虽然这些外来人为因素很大程度上与验证无关，但其产生验证过程中的瓶颈。如上所述，两种特定类型的人为因素是瞬时信号（其在特定数目个时步之后稳定于固定的恒定值）和初始化输入（其用于编码在特定数目个时步之后变得无关的复杂初始状态）。根据本发明的各方面，可使用各种技术来使瞬时信号和初始化输入的识别和消除自动化以增强分析。再次参看图 1，因为寄存器“C”在时间“2”处开始采用值“0”并在所有时间都保持该值，所以寄存器“C”是瞬时寄存器。相比之下，因为寄存器“A”及“B”两者的值保持未定且不稳定于固定的恒定值，所以寄存器“A”及“B”不是瞬时寄存器。

[0022] 如先前所指出，瞬时逻辑可归因于实施初始化序列的初始化逻辑。例如，常见设计式样允许逻辑设计在非确定性状态下接通电源，一系列状态转变使设计从所述非确定性状态变为设计以一致方式行动的“重设状态”。通常，专用初始化逻辑用于强制逻辑设计经过初始化阶段，且在此情形下，验证假设 / 检查器通常被定制为仅仅致力于完成初始化阶段。因为在初始化过程完成之后初始化逻辑的大部分稳定于固定的恒定值，所以在分析期间可消除初始化逻辑。

[0023] 已提出至少一种常规方法以通过使用初始化阶段的三值仿真（应用“X”值来反映非确定性）来保守地确定逻辑设计在后初始化中可驻留的一组状态而减小验证开销。然而，该常规方法需要专用手动工作来分解总体验证过程。通常，因为常规方法将非确定性信号保守地处理为非恒定，所以这些方法最终对该组后初始化状态过度近似，从而损失了可在后初始化状态上存在的较细微约束。过度近似又会提示逻辑设计师以手动方式添加初始化逻辑，以避免可能导致非最佳的所制造半导体器件的假性故障 (spurious failure)。

[0024] 验证约束（约束）是可在设计验证应用中使用的构造。可将约束实施为设计的网表中的被特殊标记的门（即，约束门）。通常，约束表示对验证工具探查设计的状态空间的自由度的限制。例如，约束可防止验证应用探查任何“j”时步迹线，在所述“j”时步迹线中，一个或多个约束中的任一个在“j”个时步中的任一个期间评估为逻辑零。通常，约束定义设计的状态空间的与验证目的无关的一部分，且因此将在验证该约束时不必要的消耗验证资源。作为约束的一个实例，设计可被约束为防止缓冲器已满时的数据传输。通常，当缓冲器已满时约束设计的输入以禁止数据传输意谓着验证工具不涵盖表示设计在缓冲器已满时接受新的数据传输的状态。

[0025] 在缺少约束的情况下，将典型验证问题陈述为（例如）找到展现对性质的违反的

“j”时步迹线或证明对于任一“j”而言不存在此类迹线。在具有约束的情况下，同一验证问题可表达为（例如）找到展现对性质的违反且在“j”个时步中的任一个中对于任一约束不展现逻辑零值的“j”时步迹线，或证明对于任一“j”而言不存在此类迹线。因为约束更改了验证问题的语义，所以约束有可能使可通过设计达到的性质变得不可达到。因此，需要周全地选择约束。通常，约束不应更改验证问题的语义。例如，不应准许会阻止验证工具发现信号的有效断言的约束。因为约束禁止探查某些原本可达到的状态，所以冗余移除算法可利用约束来实现较大门合并。具体地说，冗余移除算法可合并在沿不违反任何约束的路径可达到的所有状态中等效的门，即使合并后的门在仅可在违反约束之后达到的一些状态中不等效时也是如此。

[0026] 可通过以随机门替代原始网表中的顺序门而引入割点门 (cutpoint gate) (引入至修改后的网表中)。随机门的输出驱动修改后的网表中的与在原始网表中驱动的关联顺序门相同的输入。然而，与原始网表中的顺序门的输入不同，随机门的输入是随机输入，这些随机输入不连接至修改后的网表的任何其他元件。至随机门的输入可在任何门循环采用任何值，与施加于设计的其他刺激无关。这样，将割点引入网表中的净效应可以是对设计的行为进行过度近似，因为随机门可仿真顺序门的行为，而顺序门对随机门的仿真未必真实。作为原始网表的过度近似模型，修改后的网表可包括无法用于在原始网表中断言目标门的状态。

[0027] 最近已建议最初开发为用于增强的合成的重定时技术来经由锁存器（触发器）计数的减小而增强验证（即，减少验证时间）。一般而言，重定时指跨组合门移动锁存器的过程。通常，许多现有技术重定时算法将处于验证下的设计中的每个门移位任意量，这可对在存在约束的情况下设定的验证中使用重定时造成挑战。

[0028] 瞬时逻辑也可归因于验证测试台。通常，测试台包括三个组件：包括足够输入假设以提供有意义的输入激励的驱动器；处于验证下的逻辑设计；及用于在给定输入下验证该设计的正确性的检查器。测试台驱动器可被构建为过度约束设计的输入（例如，仅对照可能设计行为的子集来测试设计以促进情况细分策略 (case-splitting strategy)）。在给定输入的减小集合的情形下，本来不会正常地稳定于恒定行为的各种信号可在特定数目个时步之后稳定于恒定行为。由验证测试台引起的瞬时逻辑的一个实例可见于浮点单元 (FPU) 验证方法中，该浮点单元 (FPU) 验证方法检查经由空管线传播的单个操作码的正确性。例如，当在处于评估下的单个操作码之后驱动非操作 (NOP) 操作码时，FPU 的内部状态稳定于恒定的 NOP 行为。在此情形下，可将 FPU 中的所有信号视为瞬时逻辑。

[0029] 瞬时逻辑还可归因于外来初始化输入。如先前指出的，设计具有一组可能的初始状态是很常见的。在此情形下，测试台驱动器可通过引入初始化输入（引入至主输入）而以非确定性方式（自该组可能的初始状态）选择单个初始状态。然而，在第一个时间帧之后，初始化输入的值是无关的。根据本发明的一个实施例，揭示一种技术，该技术识别初始化输入的可安全地被恒定值取代的子集，从而增强消除来自设计的瞬时的能力而不显著增大设计的总大小。

[0030] 本文中所揭示的技术（其需要相对较小的运行时间）通过将恒量注入至设计中而优化逻辑设计。所揭示的技术使得能够按比例调整至对于常规方法而言可能难处理的大型工业设计。根据本发明的一个实施例，揭示一种技术，该技术自动检测逻辑设计中的瞬时初

始行为的存在（以及瞬时初始行为的长度）并为设计提供约简（优化）信息。根据本发明的另一实施例，揭示一种用于将分析（例如，验证问题）分解成两部分的技术：设计在初始阶段期间以一致方式行动的检查（第一分解问题）；及设计在初始阶段之后以一致方式行动的检查（第二分解问题）。通常，本文中所揭示的技术可应用于形式验证、仿真、硬件加速和合成。

[0031] 根据又一实施例，揭示一种用于基于知晓设计未在初始阶段中操作而最小化第二分解问题的大小的技术。通常，最小化第二分解问题的大小减小了设计复杂性且简化了第二分解问题。对分解的资源定界减小了两个所得到的分解问题的复杂性且促进了技术对多种工业设计的适用性。通常，这些技术提供优化信息，所述优化信息可在合成 / 设计流程中使用以在减小的功耗、减小的面积等方面改进所制造的半导体器件的效率。此外，所揭示的技术进一步增强了合成中的后瞬时约简的适用性，并且通过减小使用后瞬时约简通常所必需的初始化结构的大小而进一步优化验证分析。

[0032] 作为一个实例，识别瞬时信号的存在和持续时间的技术可使用三元仿真（其相对较快且可按比例调整）。根据本发明的另一实施例，可通过将无界验证问题分解为两个验证子问题（即，在瞬时行为发生的初始时间帧内的有界验证问题和在剩余时间帧内的无界验证问题）而（自验证过程）消除瞬时逻辑。在此情形下，无界验证问题可安全地假设所有瞬态已稳定于它们的后瞬时恒定值以简化无界验证问题。

[0033] 根据本发明的各方面，用于识别可安全地以恒量替代的初始化输入的 可按比例调整技术使用与结构分析组合的有界模型检查 (BMC)。通常，对于接通电源的非确定性的建模而言，消除存在于测试台中的初始化输入是有用的。因为用于消除瞬时逻辑的所揭示技术可产生初始化输入，所以可在从逻辑设计消除瞬时逻辑之后有利地实施消除初始化输入的技术。

[0034] 通常，三元仿真有效地对逻辑设计中的一组可达到的状态进行过度近似。三元仿真工作的方式为：保守地使用三元“X”值为主输入建模并仿真一系列 3 值状态直至状态被重复为止。在收敛之后，该组观测后的 3 值状态构成对该组可达到的状态的过度近似。在时间“y”处的状态在时间“x+y”处被重复的实例三元仿真运行中，可达到的状态被过度近似。虽然对于性质检查而言与三元仿真关联的过度近似时常过于粗略，但过度近似对于有效地识别特定设计特征是有用的。例如，特定的恒定和等效信号可为可检测的（使用三元仿真），此情形促进了设计简化。作为另一实例，可通过使用三元仿真来检测类似时钟的振荡信号而促进时间相位抽象。除了瞬时持续时间（在该瞬时持续时间之后，瞬时信号稳定于恒定行为）以外，三元仿真还可容易地被扩充为有效地检测瞬时信号的子集。

[0035] 可经由各种技术来实施使用三元仿真以找到瞬时信号的直接实现。在收敛之后，可执行对所有信号的扫描 (sweep) 以判定哪些信号在状态重复循环 (loop) 内保持恒定（即，哪些信号为瞬时信号）。然后可将在状态重复循环内保持恒定的信号添加至瞬时列表（连同信号所稳定于的恒量）及信号被评估处于非恒定值的最后时间帧。通常，信号被评估处于非恒定值的最后时间帧表示信号的瞬时持续时间的上界（归因于三元仿真的过度近似）。

[0036] 例如，瞬时信号的检测可经由如下函数来实施：

[0037] 01 :function detectTransients(design)

```
[0038] 02 :// 执行三元仿真直至收敛为止
[0039] 03 :History := Φ ;
[0040] 04 :ternaryState := getTernaryInitialState(design)
[0041] 05 :for(time = 0 ; ;time++) do
[0042] 06 :if(ternaryState ∈ History) then
[0043] 07 :cycleStartTime := calculateCycleStartTime(History, ternaryState)
[0044] 08 :break
[0045] 09 :end if
[0046] 10 :History := History ∪ ternaryState
[0047] 11 :ternaryState := getNextTernaryState(design, ternaryState)
[0048] 12 :end for
[0049] 13 :
[0050] 14 :// 提取瞬时信号
[0051] 15 :transients := Φ ;
[0052] 16 :for all(signals s in design) do
[0053] 17 :for all(constants C in {0,1}) do
[0054] 18 :if( ∀ time > cycleStartTime, s = C) then
[0055] 19 :duration := latestToggle(s, History) // ≤ cycleStartTime
[0056] 20 :transients := transients ∪(s, C, duration)
[0057] 21 :end if
[0058] 22 :end for
[0059] 23 :end for
[0060] 24 :
[0061] 25 :return transients
[0062] 26 :end function
```

[0063] 在以上 detectTransients 函数中,在一组瞬时信号内的每一瞬时信号的最大瞬时持续时间之前,瞬时信号中的一个或多个可采用不同于所稳定的恒定值的值。在此情形下,可使用 BMC 以检查任何性质在初始时间帧内的有效性。在最大瞬时持续时间之后,一组瞬时信号中的所有瞬时信号已稳定于其对应的恒定值。然后,可通过以瞬时信号的相应恒定值替代这些瞬时信号而简化逻辑设计(体现在网表中)。可接着实施无界验证过程以检查逻辑设计的简化后的模型中的剩余时间帧。

[0064] 例如,可在基于变换的验证(TBV)架构内优化无界验证。在 TBV 架构中,在调用终端验证技术之前将一系列变换应用于逻辑设计(体现在网表中),从而允许变换所提供的约简产生针对终端验证技术的显著加速。在此情形下,瞬时简化例程可扮演网表变换的角色(而非针对现有验证技术的定制)以促进与任何下游合成或验证技术的兼容性。

[0065] 通常,存在可用于检测瞬时的众多可能技术。典型地,需要使瞬时检测技术穷举地分析网表的行为以便确保看起来在给定时间帧之后终止的瞬时行为实际上在稍后时间帧处不会重新发生。根据本发明的一个实施例,使用三元仿真来过度近似网表的行为。作为另一实例,可根据以下提出的 analyzeTernarySim 函数(其类似于 detectTransients 函数

且以伪代码提供) 来实施用于检测瞬时的三元仿真:

```
[0066]  function analyzeTernarySim(design) {
[0067]    01 :currentState := initial state
[0068]    02 :inputValues := " XXXXXX... "
[0069]    03 :stateHistory := empty array
[0070]    05 :transientSignals := empty array
[0071]    06 :settledValues := empty array
[0072]    07 :transientDurations := empty array
[0073]    09 :for(timeStep := 0 ; ;timeStep := timeStep+1) {
[0074]      10 :// 执行一个仿真步骤
[0075]      11 :currentState := ternarySimulate(currentState, inputValues)
[0076]      12 :X saturate
[0077]      13 :// 查找最近稳定的瞬时信号
[0078]      14 :for all signals in design and not in transientSignals{
[0079]        15 :if(signal == 0 in currentState or signal == 1 in currentState) {
[0080]          16 :add signal to transientSignals
[0081]          17 :transientDurations[signal] := timeStep
[0082]          18 :settledValues[signal] := value of signal in currentState
[0083]        19 :}
[0084]      20 :}
[0085]      22 :// 检查先前恒定的信号是否仍恒定
[0086]      23 :for all signals in transientSignals{
[0087]        24 :if(value of signal in currentState != settledValues[signal]) {
[0088]          25 :if(signal == 0 in currentState or signal == 1 in currentState) {
[0089]            26 :transientDurations[signal] := timeStep
[0090]            27 :settledValues[signal] := value of signal in currentState
[0091]          28 :} else{
[0092]            29 :remove signal from transientSignals
[0093]          30 :}
[0094]        31 :}
[0095]      32 :}
[0096]      34 :// 检查收敛
[0097]      35 :if(stateHistory contains currentState) {
[0098]        36 :startLoopTime := time at which first occurrence of currentState was seen
[0099]        37 :
[0100]        38 :// 舍弃稳定不够快的信号
[0101]        39 :for all signals in transientSignals{
[0102]          40 :if(transientDurations[signal] > startLoopTime)
[0103]            41 :remove signal from transientSignals
```

```
[0104] 42 :}
[0105] 43 :hash value(modulo inversion) of all signals *not* in transientSignals
[0106] 44 :create equivBuckets[signal] for each signal which matches > 1 hashed
entry
[0107] 45 :return(transientSignals, settledValues, equivBuckets,
transientDuration) ;
[0108] 46 :}
[0109] 47 :add currentState to stateHistory
[0110] 48 :}
[0111] }
```

[0112] 参看以上所阐述的 analyzeTernarySim 函数, 在行 1 至行 3 中, 假设设计处于设计师所指定的初始状态下。如果未给定寄存器的初始状态, 则该寄存器被指派值“X”, 值“X”指示该寄存器可为“0”或“1”。假设设计输入可采取任何值(其由针对输入的向量“Xs”来表示)。三元仿真维护所有可视状态的列表(最初该列表为空)。在行 5 及行 6 中, 假设不存在瞬时信号。这样, settledValues 及 transientDurations 数组为空。在行 9 至行 12 中, 通过遍历所有时步来探查设计的时间行为。在每一时步中, 使用三元仿真来形成逻辑设计的当前状态。应了解, 可使用“X 饱和”技术来概括所述状态, 这些“X 饱和”技术促进三元仿真例程的相对较快的收敛。

[0113] 在行 13 至行 20 中, 在当前时步中, 信号可采用恒定的“0”或“1”赋值。如果一个信号对于所有未来时间点保持恒定, 则该信号是瞬时的, 且当前时步表示瞬时持续时间。相应地更新瞬时追踪数据结构。在行 22 至行 32 中, 先前估计为瞬时的所有信号已在先前时步处采用“0”或“1”赋值。该函数接着检查信号在当前时步处于相同恒定值。如果信号在当前时步不处于相同恒定值, 则函数改进对哪些信号是瞬时信号的估计。在行 34 至行 44 中, 如果当前状态等于在某一先前时步处所见的状态, 则状态的进度已完成循环, 且所有未来时间处的所有未来状态继续遵循该循环。因此, 函数断定已遇到了每一可能状态, 且函数返回已找到的瞬时信号的列表。参看行 39 至行 42, 在状态的循环开始之前, 瞬时信号应已稳定。如果此情形为真, 则信号在循环上保持恒定, 且因此对于所有未来时间点保持恒定。如果信号稳定得过晚, 则从瞬时信号列表舍弃该信号。

[0114] 移至行 43 至行 44, 试图识别在初始瞬时阶段之后等效的信号。例如, 可通过使用散列表记录每一信号(除已被识别为恒量的那些信号外)在后瞬时阶段期间所展现的值来识别等效信号。在瞬时阶段之后等效的信号将具有相对于散列的后瞬时值相同的值。可使用(例如)模逆运算来进行该比较, 以将在后瞬时行为期间评估为相反值的两个信号识别为冗余候选项。

[0115] 三元仿真技术是实用的, 因为该技术相对较快、可按比例调整且连同简单的后处理技术一起使能对稳定于恒定值的信号的实时计算以识别稳定于等效(或相反)后瞬时行为的信号。然而, 三元仿真为过度近似, 且因此是有损耗的。备选地, 可使用一种更准确(近似性较小)的方法。例如, 可使用二元决策图(BDD)来执行可达性分析以识别从逻辑设计的初始状态开始可达到的所有状态。在该可达性分析期间, 可维护表示在时间“0”(初始状态)、时间“1”、时间“2”、...、时间“j”处可达到的所有状态的 BDD, 直至不再发现新的状态

为止。接下来,可以以迭代方式分析每一寄存器(或者,设计中的每个门)在时间“j”处可达到的状态,从而评估哪些状态恒定且哪些配对在于时间j处可达到的所有状态中等效/相反。

[0116] 恒定时间检查通过线性(每个寄存器/门一次)探测所得到的时间“j”可达到状态集合而直接进行,从而评定该可达到状态集合与对应恒定条件的交集是否为空。等效/相反关系的判定通过对称配对是等效/相反的若干二次(每对寄存器/门一次)交集检查而直接进行。该检查产生具有瞬时持续时间“j”的冗余候选项的集合。备选地,可对时间“i”…“j”的已达到状态集合执行分析,从而产生具有瞬时持续时间“i”的后瞬时冗余候选项的集合。作为另一备选实施例,在任何时间处可达到的状态的集合可被过度近似,因为如此操作可减小分析运行时间。

[0117] 某些基于可满足性的技术也产生过度近似的可达到状态集合表示。例如,内插使用可满足性分析以迭代方式近似在时间0、1、…、“j”处可达到的状态的集合,直至判定在时间“j”处不再遇到额外状态为止。经常使用网表类型的表示来表示所得到的状态集合。可直接将基于BDD的分析应用于该状态集合表示。备选地,可使用用于直接针对时间“j”(或备选地,按照基于BDD的技术,时间“i”…“j”)的状态集合表示识别网表电路中的冗余的技术来识别后瞬时冗余。

[0118] 为了使用通用验证技术检查瞬时持续时间之后的性质,通常有必要对设计进行时移。在此过程中,可调整设计的时间基线以使得原始设计中的最大瞬时持续时间(maxTransientDuration)的时间对应于时移后的设计中的时间“0”。在此情形下,修改设计的初始状态以使得时移后的设计在maxTransientDuration时步中可达到的任何状态下开始。例如,可通过展开转变关系且使用展开后的结构的输出来驱动初始状态而实现该变换。这类似于使用结构的符号化仿真来计算新的初始值集合。

[0119] 可接着使用采用一组检测后的瞬时信号的相对直接的程序以简化设计。在此实施例中,可使用BMC来检查最大瞬时持续时间之前的性质。可接着对设计进行时移,并且可通过将瞬时信号与相应稳定恒量合并来简化网表。因为通常可迅速地获得大部分益处,所以可使运行时间限于相对较低的时间“t”,例如,十秒钟。例如,瞬时信号的简化可经由如下函数来实施:

```

[0120] 01 :function simplifyTransients(design, transients)
[0121] 02 :maxTransientDuration := computeMaxTransientDuration(transients)
[0122] 03 :for(t = 0 ;(t < maxTransientDuration) ;t++) do
[0123] 04 :Validate that properties are not falsifiable at time t with BMC
[0124] 05 :
[0125] 06 :// 时移 1 个时钟周期
[0126] 07 :for all(registers r ∈ design) do
[0127] 08 :initialValue(r) := nextState(r)
[0128] 09 :end for
[0129] 10 :
[0130] 11 :// 逐渐简化设计
[0131] 12 :for all(g ∈ transients) do

```

```
[0132] 13 :if(transients[g]. settlingTime ≤ t) then  
[0133] 14 :merge(design, g, transients[g]. settledConstant)  
[0134] 15 :end if  
[0135] 16 :end for  
[0136] 17 :end for  
[0137] 18 :end function
```

[0138] 在以上 simplifyTransients 函数中, 以稳定恒量来替代瞬时。以上 simplifyTransients 函数的一种应用是简化验证(例如, 形式验证、仿真或硬件加速)架构。在如下 decomposedVerification 函数(以伪代码呈现)中例示在验证中利用后瞬时冗余信息的另一方法:

```
[0139] function decomposedVerification(design) {  
[0140] 01 :transientSignals := identify_posttransient_redundancies(design)  
[0141] 02 :if(transientSignals is empty)  
[0142] 03 :return " Cannot decompose verification"  
[0143] 04 :maxDuration : = maximum amount of time any transientSignals are  
transient  
[0144] 06 :// 分解验证的部分 1 :瞬时帧的有界验证  
[0145] 07 :if(boundedVerification(design, maxDuration) = " Counter-example  
Found" )  
[0146] 08 :return " Counter-example Found"  
[0147] 09 :// 分解验证的部分 2 :简化和时移后的设计的无界验证  
[0148] 10 :design := timeShift(design, maxDuration)  
[0149] 11 :design := simplifyDesign(design, transientSignals)  
[0150] 12 :return unboundedVerification(design)  
[0151] }
```

[0152] 在以上所阐述的 decomposedVerification 函数的行 1 至行 4 中, 可使用任意技术来识别在某一初始瞬时时段之后展现冗余的门。通常, 所述分析识别瞬时信号和瞬时信号的持续时间两者。如果未找到瞬时信号, 则约简机会不存在。然而, 归因于常见设计式样及常见验证方法, 实际中经常找到瞬时信号。在此情形下, 该函数计算所有瞬时的最大持续时间, 且在若干时步之后, 该函数断定所有瞬时已稳定于其恒定值。在行 7 至行 8 中, 该函数检查设计在接通电源之后的最初 #maxDuration 个时步正确地行动。例如, 可使用 BMC 检查正确设计行为。归因于 maxDuration 通常相对较小的事实, 以此方式检查正确设计行为可容易地按比例调整。

[0153] 如果在最初时步中找到反例, 则设计无法如预期地行动且出于调试目的而返回该反例。备选地, 可在试图找到反例中使用欠近似技术(如仿真或硬件加速)。参看行 10 及行 11, 该逻辑设计的时间基准在时间上前进至所有瞬时已稳定于其恒定值的时刻。取决于验证设置, 此操作可以以若干方式进行。对于形式验证而言, 可使用时移来确保精确的分析结果。对于仿真或加速而言, 可确定在有界验证周期之后可达到的一组初始状态, 且可将所得到的简化后的设计初始化成这些状态以实现增强的仿真或加速分析。应注意, 在所有这

些架构中,可通过以每个瞬时信号的相应恒定值替代每个瞬时信号来简化网表。参看行 12,接着使用无界验证技术(例如,归纳、内插、显式可达性等)来验证时移和简化后的设计。典型地,无界验证极其困难。然而,设计简化有助于减小验证过程的复杂性。对于一些设计式样而言,不可能在没有简化的情况下在合理的时间限制内完成验证。

[0154] 通常,因为不正确地行动的设计将具有可使用分解验证找到的有效反例,所以分解验证策略健全且完善。正确设计将不具有此类反例,并且每个分解验证问题将成功完成。根据一个或多个实施例,在已检测到瞬时信号且最多到最大瞬时持续时间的有界验证完成之后,逻辑的时间基准被移位。在以下所阐述的伪代码中例示实例 timeShift 函数(其关注于穷举形式验证架构):

```
[0155] function timeShift(design, timeSteps) {
[0156]   1 :unrolledDesign := concatenate #timeSteps transition relations
[0157]   3 :for each register in design
[0158]     4 :initialValue[register] := signal in unrolledDesign corresponding to
register at time timeSteps
[0159]   6 :return design
[0160] }
```

[0161] 参看该 timeShift 函数的行 1,根据当前状态和逻辑设计输入(称为转变关系)确定下一状态。通过将转变关系串接在一起而构建针对最初 #timeSteps 的设计行为的模型。时间“0”转变关系中的寄存器被其设计师所指定的初始值(展开后)取代。信号“X”的时间“K”实例是一个逻辑节点,该逻辑节点可采用“X”可在原始逻辑设计中在“K”个时步之后采用的逻辑值中的任一个。参看行 3 至行 4,每一寄存器的初始状态被修改为来自展开后的设计,从而确保在时间“0”处修改后的逻辑设计中的寄存器可采用原始逻辑设计中在时间 #timeSteps 处可能的任一值。以此方式,修改后的设计被时移。

[0162] 以穷举形式呈现 timeShift 和 decomposedVerification 函数以完成准确形式验证(即,这些函数促进对要向前移位以使能后瞬时约简的时间帧的穷举性分析),该准确形式验证针对初始时间帧的“有界验证”和移位之后的“无界验证”两者。在一些情形下(例如,当使用不完善验证技术,诸如,仿真或硬件加速),可能需要对分析进行欠近似。在使用不完善验证技术的架构中,有界验证(boundedVerification)是可选的。如果执行有界验证,则有界验证还可以可选地使用仿真或硬件加速架构,以在初始瞬时时间帧期间直接欠近似地评估网表。类似地, timeShift 函数可被欠近似以计算与所得到的展开后的设计兼容的状态的任意子集(再次使用任意技术集合,例如,要用于无界验证的仿真或硬件加速方法),并且可相对于任意状态子集执行欠近似的无界验证。在所有瞬时信号的时移之后,在所有时间内将恒定值指派给这些瞬时信号。可接着以瞬时信号的相应恒量来替代这些瞬时信号,并且可使用恒定传播以简化设计。

[0163] 可使用任一数目的方法来最小化 timeShift 函数所产生的展开后的逻辑的大小。例如,可通过如下所阐述的 reduceUnrolledTimeShift 函数来最小化展开后的逻辑:

```
[0164] function reduceUnrolledTimeShift(design) {
[0165]   1 :for(i = 0 ;NOT termination criterion ;i++)
[0166]     2 :registerList = registers whose next-state functions are sensitized to
```

other registers with nonconstant initial values due to timeShift

[0167] 3 :perform boundedVerification at time " i "

[0168] 4 :create unrolledDesign representing registerList valuations for time " i "

[0169] 5 :perform an arbitrary set of reduction steps on unrolledDesign

[0170] 6 :if any random gates no longer fanout to registerList, tie to an arbitrary constant

[0171] }

[0172] }

[0173] 在 reduceUnrolledTimeShift 函数中,通过消除“随机”门而最小化逻辑,这些“随机”门的值被确定为在特定数目个时间帧内与网表的分析无关。例如,“终止准则”可包括用于排除对未来时间帧的分析的任一任意停止准则、用户指定的参数、时间,或存储器限制,等等。应注意,在纯粹合成架构中(即,当试图使用该技术自动地优化增强的半导体器件(需要较小面积、较少功耗等)的设计表示时),表示符号化的非恒定初始值的需要可排除使合成程序完全自动化的能力。具体地说,因为优化后的网表可能与原始网表的难以关联的未来时间帧相关,所以用于确认合成未更改设计行为的自动等效性检查可能变得困难。因此,可能需要将所得到的优化可能性用作对设计师的反馈,如果所得到的优化后的网表具有符号化的初始状态,则设计师可相应地以手动方式优化逻辑设计。然而,在许多情形下,本文中所揭示的时移优化技术能够消除大部分(如果不是全部)符号化初始值,使得该技术能够在自动化合成流程中无缝地使用以实现减小的功率、面积等。

[0174] 展开寄存器的初始状态可增大设计的大小,且即使稍后恒定传播,此技术亦可能实际上增大总体设计大小。此情形可通过逐渐展开初始状态且仔细地监视设计的大小来克服。一旦设计大小增大超过预定阈值,则可停止展开。在此情形下,应丢弃具有长于展开的量的持续时间的任何瞬时。在实际中,此过程促进利用大部分已找到的瞬时信号,不会因不合理地增大设计大小而负面影响验证。

[0175] 通常,某一量的大小增大是可接受的。可认为逻辑设计为两部分:计算寄存器的初始值且仅在时间“0”处使用的逻辑以及在所有大于零的时间使用的逻辑。本文中所揭示的技术可增大初始值逻辑,同时减小其他逻辑。许多形式验证技术(例如,归纳)只是最低程度地利用初始化逻辑,并且从这些技术的角度,初始化逻辑的增大是无意义的。在仿真或硬件加速中,因为计算并重用时移后的初始值的子集,所以逻辑膨胀(logic bloat)可以是不相关的。

[0176] 时移意谓着修改后的设计从预定数目个时步后将不仿真任何行为。为了使形式验证健全,通常应单独地验证早期时步。例如,可通过 BMC 来验证早期时步(其数目等于最大瞬时持续时间)。如果时步的数目较大,则在合理时间限制内完成 BMC 可能不切实际。此情形可通过逐渐地使 BMC 个别地检查每个时步来克服。如果在任何检查期间超过了计算资源,则无法完成整个 BMC 问题。在此情形下,至少已检查了初始时步的一个子集。可丢弃持续时间超过所检查的持续时间的瞬时信号,且可进行使用剩余瞬时信号的验证分解。

[0177] 如果设计未如预期地行动,则可能在验证时移后的模型时找到反例。在此情形下,反例也被时移。应务必不使反例时移,以便根据原始设计而不是时移后的中间表示将该反

例报告给用户。具体地说,假设将网表时移“ k ”个时步,则来自网表的顺序部分的反例值应向前时移“ k ”个时间帧以抵销对整体验证的转变的影响。另外,可使用用于表示展开后的初始时间帧的逻辑值来填充顺序网表赋值的向前移位在这“ k ”个时间帧期间所遗留下的间隙。

[0178] 时常,验证工程师使用约束,这些约束指示验证环境应仅在保持指定条件时探查设计的状态。在使设计时移之后,应务必确保在已移位掉的时步中遵守(honor)这些约束。这可通过在时移后的设计中将这些约束变换为针对展开后的初始值的复杂限制来实现。具体地说,不仅通过保留时移后的约束,而且还通过使用上述展开程序来针对每个时间帧 $0\dots k-1$ (假设总时移为“ k ”个时步)添加约束,从而表示该约束在时间“0”(即,针对直接馈入该约束的寄存器的初始值)、时间“1”(针对馈入在时间“0”处相关的寄存器的寄存器)、...处将具有的值。

[0179] 如前所述,一些测试台包括初始化输入以为复杂初始值建模。初始化输入也可由于在时移网表中使用符号化仿真以实现瞬时信号的简化而出现。初始值的类似复杂性以外围重定时的副产品的形式存在。由于时移后的初始值引起的大小的增大是不期望的,因为其可使由瞬时信号的合并而引起的大小的减小偏移至恒量。虽然特定技术(例如,归纳)可不受初始值复杂性增大的影响,但在TBV设置中,一些技术会受增大的复杂性的影响。如前所述,初始化输入是值仅在时间“0”处影响逻辑设计的输入,且经常用于编码复杂的初始状态。通常,时移固有地引入大量初始化输入。然而,并非所有引入的初始化输入皆可与设计行为相关。因此,通常可通过移除引入的初始化输入的至少一个子集来优化时移后的设计。

[0180] 例如,在可在四个可能初始状态下开始的设计中,可使用两个初始化输入来为该组初始状态建模。在状态转变图(STG)中的所有路径在两个时步之后皆通过单个支配状态的情形下,有可能在不影响设计的行为的情况下减小可能初始状态的数目。因为在网表中使用额外初始化输入来表示该组初始值,所以可通过使用恒量来替代初始化输入来执行简化。此类型的简化是基于不关心可观测性(observability don't care,ODC)的简化的形式,因为在有限数目个时步之后不再可观察到个别初始状态。

[0181] 可单独使用结构分析来检测与设计的行为无关的初始化输入的子集。例如,可使用影响锥(COI)分析来识别在固定数目个时步“ t ”之后对网表没有影响的输入的子集。一旦识别了无关初始化输入,则可在时间“ t ”之后在不影响设计的行为的情况下以任意值来替代该无关初始化输入。在此情形下,通常保证修改后的设计在时间“ t ”之后与原始设计等效,但在时间“ t ”之前,修改后的设计仅可访问原始设计的状态的一个子集。为了确保在简化期间不遗漏有效反例,通常有必要在简化之前确认性质直至时间“ t ”的正确性。

[0182] 例如,可通过以下所阐述的 simplifyInputs 函数来消除无关输入。

```
[0183] 01 :function simplifyInputs(design, maxTime)
[0184] 02 :unrolledModel := Φ ;
[0185] 03 :for all(registers r ∈ design) do
[0186] 04 :instance of r in unrolledModel := initialValue(design, r)
[0187] 05 :end for
[0188] 06 :for(t = 0 ; t < maxTime ; t++) do
```

```

[0189] 07 :Validate that properties are not falsifiable at time t
[0190] 08 :
[0191] 09 :// 逐渐展开模型
[0192] 10 :Append one transition relation to unrolledModel
[0193] 11 :unrolledModel := resynthesize(unrolledModel)
[0194] 12 :
[0195] 13 :// 计算展开后的 COI
[0196] 14 :C := Φ ;
[0197] 15 :for all gates g ∈ (next-state functions ∪ properties) do
[0198] 16 :C := C ∪ COI(unrolledModel, last temporal instance of g)
[0199] 17 :end for
[0200] 18 :
[0201] 19 :// 移除不必要的输入
[0202] 20 :for all (primary input s ∈ 2 design) do
[0203] 21 :if (I ⊈ C) then
[0204] 22 :merge(design, i, 0)
[0205] 23 :end if
[0206] 24 :end for
[0207] 25 :end for
[0208] 26 :end function

```

[0209] 该 simplifyInputs 函数逐步地增大时间“t”直至超过计算资源为止。对于每个“t”而言,执行验证以确保性质在该时间帧处无法被窜改。接下来,逐渐展开设计并检验其 COI。为了减小 COI 的大小且增强技术的约简潜力,可针对展开后的设计使用合成技术(布尔可满足性 (SAT) 扫描、重写,等等)。在此情形下,不属于下一状态函数和性质的 COI 的输入通过将这些输入与恒量“0”合并而被从设计移除。与大部分基于 ODC 的简化例程不同,以上函数所识别的所有简化固有地兼容。这些简化可在不彼此干扰的情况下被同时利用,从而导致较大的效率。另外,因为该技术依赖于电路结构,所以其可按比例调整性很高。然而,因为该技术是不完善的,所以可能无法识别一些无关的初始化输入。可通过传统的基于 ODC 的简化的后处理来补充该技术。

[0210] 通常,无关输入消除技术减小了计算在瞬时简化中所产生的时移后的网表的初始值所必要的符号化仿真的开销。通常,初始化输入简化在减轻由瞬时简化所引起的逻辑膨胀方面是有效的。

[0211] 如以上所指出,用于消除无关输入的所揭示技术使用 BMC 和结构分析来移除初始化输入。通常,该技术相对较快,但无法识别所有不必要的 初始化输入,且因此,可通过在以上技术之后运行更彻底的初始化输入简化例程来识别额外不必要的初始化输入。以上技术与传统基于 ODC 的优化技术的类似之处在于,其针对每个候选简化产生设计的逻辑窗口的副复本(side copy),从而评定特定简化是否可被视为相对于该窗口更改设计行为。此特定技术限于相对于可配置的顺序深度的逻辑窗口来评定合并初始化输入的有效性。

[0212] 通常,瞬时简化主要利用三元仿真和 BMC,并且初始化简化利用结构方法和 BMC。

这些技术是有效且可按比例调整的,且可逐渐地执行大部分分析。这些简化技术皆可被实施为随时间帧逐渐地简化设计,直至耗尽预定计算限制为止。

[0213] 本文中所描述的技术可实施于根据 TBV 架构构建的验证工具中,在 TBV 架构中,在利用终端验证引擎来试图解决简化后的问题之前,各种引擎逐渐地简化复杂性质。在鲁棒的验证设置中,可能需要在以上简化之前和之后调用内插以最大化获得确凿结果的机会。

[0214] 在顺序上等效的信号(有时称为信号对应)的检测和简化是减小顺序逻辑设计的大小的有效方式。时常,这些约简证明了安全性性质,或在针对另一下游验证引擎而简化问题方面是有效的。本文中所揭示的技术涉及重定时,因为两种方法皆使设计时移且作为副产品而遗留下复杂的初始值。

[0215] 在各种情形下,时移并简化瞬时信号对于通过归纳完成证明而言可以是至关重要的,且因此,所揭示的技术可形成各种验证方法的总体可按比例调整性的关键部分。具体地说,在顺序等效检查(SEC)中,非常需要成对等效的内部点的识别来确保输入/输出等效的归纳证明的成功完成。内部点的这些配对中的许多配对初始地由于接通电源不确定性而不等效,且输入-输出等效的归纳证明因为不再保持内部等效而失败。通常,在 SEC 流程的特定方面,瞬时消除比重定时更为需要,因为 SEC 的可按比例调整性在一定程度上依赖于跨越正进行等效检查的设计的寄存器的基于名称和基于结构的相关性。然而,重定时可任意地更改寄存器布置,从而减小了 SEC 试探的可用性。

[0216] 通常,本发明涉及存在于 RT 级设计中的两种类型的冗余信息:瞬时信号和初始化输入。本文中已提出了用于识别并移除两种现象的技术。已在工业验证环境中将所建议的技术实施为在重量型形式验证技术的调用之前的轻量型设计简化步骤。设计中的许多安全性性质不可通过内插证明。在简化之后,额外安全性性质通常可通过内插证明。在信号对应之前应用简化提供了“与”门和寄存器的改进的约简。本文中所揭示的技术在于最小寄存器重定时之后简化设计、减小“与”门、寄存器及输入方面是有效的。将所揭示的简化技术应用为预处理步骤倾向于改进大型工业设计上的输入/输出等效的归纳证明的完成。

[0217] 参看图 2,例示了一个实例计算机系统 200,其可配置为执行根据本发明的各种实施例而配置的工具(该工具配置为分析集成电路逻辑设计(设计))。计算机系统 200 包括耦合至存储子系统 204 的处理器 202、显示器 206、输入设备 208,以及大容量存储设备 210。存储子系统 204 包括适合应用的量的易失性存储器(例如,动态随机存取存储器(DRAM))和非易失性存储器(例如,只读存储器(ROM))。例如,显示器 206 可以是阴极射线管(CRT)或液晶显示器(LCD)。例如,输入设备 208 可以是鼠标和键盘。大容量存储设备 210(例如,其可包括光盘只读存储器(CD-ROM)驱动器及/或硬盘驱动器(HDD))配置为接收或包括存储适当程序代码(例如,操作系统(OS)、分析工具、验证工具等)的盘。

[0218] 参看图 3,例示了用于执行逻辑设计(其体现于网表中)的分析的过程 300(其可通过一个或多个处理器来实施),过程 300 在块 302 中开始,在此点处控制转移至块 304。在块 304 中,过程 300 检测逻辑设计中的初始瞬时行为。接下来,在块 306 中,确定初始瞬时行为的持续时间。接着,在块 308 中,基于初始瞬时行为来收集关于逻辑设计的约简信息。接下来,在块 310 中,接着基于约简信息来修改网表。在块 310 之后,控制转移至块 312,在块 312 处,控制返回到调用例程。

[0219] 因此,本文中已揭示总体上减小集成电路逻辑设计的分析(例如,验证)时间的若

干技术。

[0220] 附图中的流程图和框图,图示了按照本发明各种实施例的系统、方法和计算机程序产品的可能实现的体系架构、功能和操作。在这点上,流程图或框图中的每个方框可以代表一个模块、程序段、或代码的一部分,所述模块、程序段、或代码的一部分包含一个或多个用于实现规定的逻辑功能的可执行指令。也应当注意,在有些作为替换的实现中,方框中所标注的功能也可以以不同于附图中所标注的顺序发生。例如,两个接连地表示的方框实际上可以基本并行地执行,它们有时也可以按相反的顺序执行,这依所涉及的功能而定。也要注意的是,框图和 / 或流程图中的每个方框、以及框图和 / 或流程图中的方框的组合,可以用执行规定的功能或操作的专用的基于硬件的系统来实现,或者可以用专用硬件与计算机指令的组合来实现。

[0221] 本文中所用的术语,仅仅是为了描述特定的实施例,而不意图限定本发明。本文中所用的单数形式的“一”、“一个”和“该”,旨在也包括复数形式,除非上下文中明确地另行指出。还要知道,“包含”和 / 或“包括”一词在本说明书中使用时,说明存在所指出的特征、整体、步骤、操作、单元和 / 或组件,但是并不排除存在或增加一个或多个其它特征、整体、步骤、操作、单元和 / 或组件,以及 / 或者它们的组合。

[0222] 以下的权利要求中的对应结构、材料、操作以及所有功能性限定的装置或步骤的等同替换,旨在包括任何用于与在权利要求中具体指出的其它单元相组合地执行该功能的结构、材料或操作。所给出的对本发明的描述其目的在于示意和描述,并非是穷尽性的,也并非是要把本发明限定到所表述的形式。对于所属技术领域的普通技术人员来说,在不偏离本发明范围和精神的情况下,显然可以作出许多修改和变型。对实施例的选择和说明,是为了最好地解释本发明的原理和实际应用,使所属技术领域的普通技术人员能够明了,本发明可以有适合所要的特定用途的具有各种改变的各种实施方式。

[0223] 已如此详细并参考本发明的优选实施例描述本发明,显然,在不脱离所附权利要求中限定的本发明的范围的情况下,修改及变型是可能的。

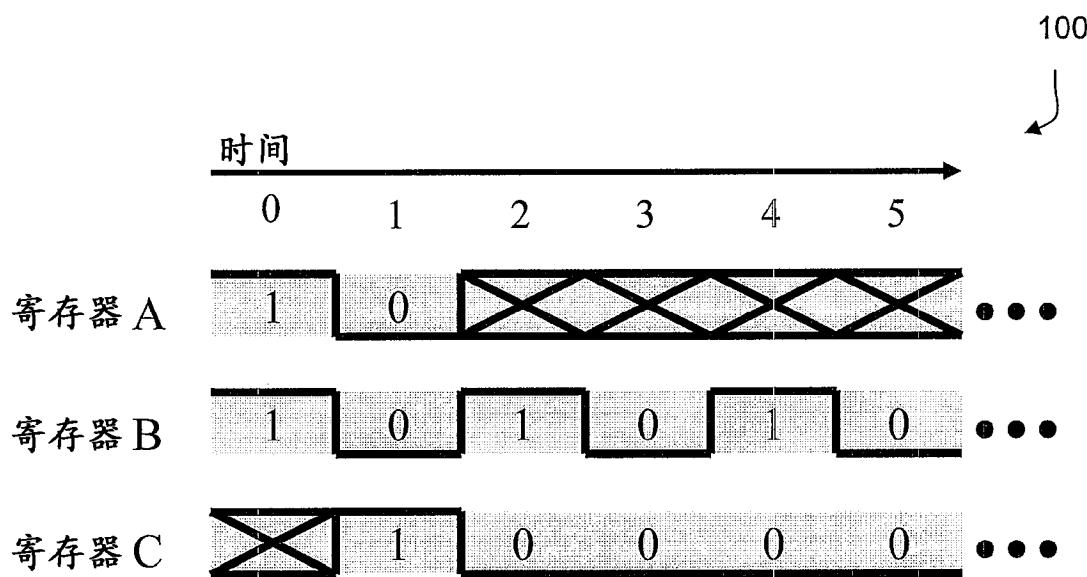


图 1

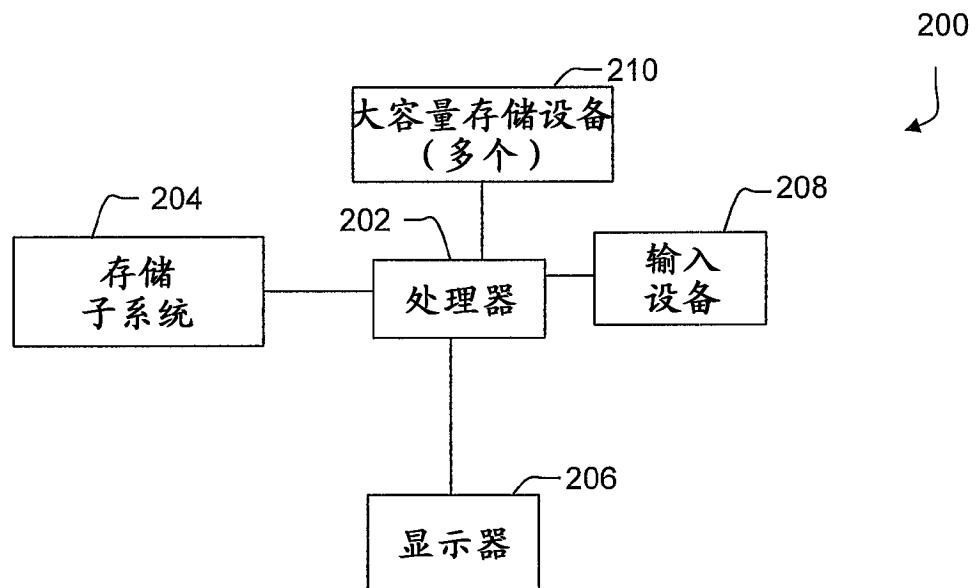


图 2

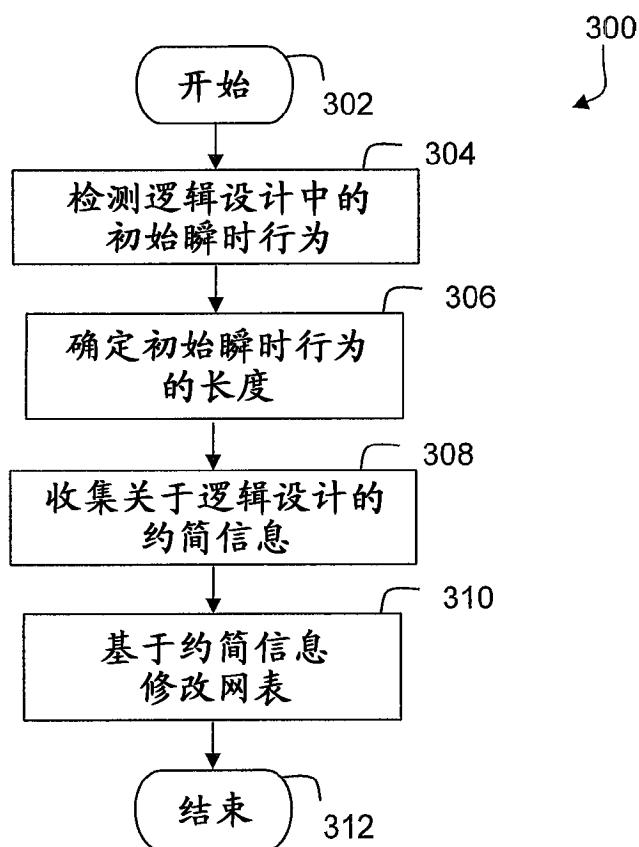


图 3