(19) Deutsches Patent- und Markenamt

(10) **DE 10 2011 077 177 A1** 2012.12.13

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2011 077 177.8**
(22) Anmeldetag: **08.06.2011**
(43) Offenlegungstag: **13.12.2012**

(51) Int Cl.: ***G06F 17/50*** (2011.01)
G06F 9/44 (2011.01)

(71) Anmelder:
Universität Bremen, 28359, Bremen, DE

(74) Vertreter:
Fink Numrich Patentanwälte, 80634, München, DE

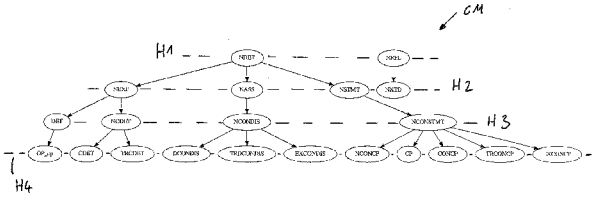
(72) Erfinder:
Fey, Görschwin, Dr.-Ing., 28215, Bremen, DE;
Sülflow, André, Dr.-Ing., 28201, Bremen, DE;
Drechsler, Rolf, Prof. Dr., 28359, Bremen, DE

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Verfahren zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache**

(57) Zusammenfassung: Die Erfindung betrifft ein Verfahren zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache, wobei mit der Hardware- Beschreibungssprache der Aufbau und die Operation einer integrierten Schaltung beschrieben wird und der fehlerhafte Quellcode zu einer fehlerhaften Ausgabe der integrierten Schaltung führt. Erfindungsgemäß wird ein Korrekturmodell (CM) vorgegeben, welches eine hierarchische Struktur von in mehreren Hierarchieebenen (H1, H2, H3, H4) angeordneten Knoten in der Form von Transformationsvorschriften umfasst, wobei eine Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) eine Gruppe von Transformationen beschreibt, welche auf zumindest einen Typ eines Quellcodeabschnitts anzuwenden sind und hierdurch den Quellcodeabschnitt verändern und wobei eine Transformationsvorschrift (NDET, NRPL, ..., EXCONCP), welche ein Kindknoten (NDET, NRPL, ..., EXCONCP) einer anderen Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) ist, eine Teilmenge der Gruppe von Transformationen der anderen Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) darstellt. Im Verfahren der Erfindung werden die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) aus der hierarchischen Struktur auf den fehlerhaften Quellcode angewendet und diejenigen Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) ermittelt, welche den Quellcode derart verändern, dass der veränderte Quellcode zu einer korrekten Ausgabe der integrierten Schaltung führt, wobei als Korrekturen zumindest ein Teil der ermittelten Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) zusammen mit dem oder den zugehörigen Quellcodeabschnitten ausgegeben werden, auf welche die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) angewendet wurden.



Beschreibung

[0001] Die Erfindung betrifft ein Verfahren zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache sowie ein entsprechendes Computerprogrammprodukt und ein entsprechendes Computerprogramm.

[0002] Die Erfindung liegt auf dem technischen Gebiet der Simulation von integrierten Schaltungen mit Hilfe von Hardware-Beschreibungssprachen. Hardware-Beschreibungssprachen sind hinlänglich aus dem Stand der Technik bekannt. Mit diesen Sprachen wird basierend auf einer entsprechenden Syntax ein Quellcode generiert, der das Design der Schaltung und damit durchgeführte Operationen spezifiziert. Mit einem solchen Quellcode kann dann über entsprechende Eingabesequenzen das Verhalten der integrierten Schaltung simuliert werden. Dabei werden im Rahmen der Simulation entsprechende Ausgaben durch die Schaltung generiert. Weichen die generierten Ausgaben von erwarteten Ausgaben ab, die mit dem Schaltungsentwurf erzeugt werden sollen, liegt ein Fehler im Quellcode vor. Das manuelle Auffinden solcher Fehler durch den Designer der integrierten Schaltung (auch als Debugging bezeichnet) ist in der Regel sehr zeitaufwändig und es besteht ein Bedürfnis, den Designer bei der Ermittlung von Fehlern im Quellcode mit einem automatisierten Verfahren zu unterstützen.

[0003] In der Druckschrift US 5 862 361 A wird ein Verfahren beschrieben, bei dem aus einer funktionalen Beschreibung eines Hardware-Bausteins ein Quellcode in den Hardware-Beschreibungssprachen VHDL oder Verilog generiert wird. Dieser Quellcode kann dann effizient simuliert werden.

[0004] Das Dokument DE 102 43 598 A1 offenbart ein Verfahren zur funktionalen Verifikation integrierter Schaltungen, mit denen Fehler im Schaltungsentwurf nachgewiesen werden können.

[0005] Das Dokument US 2009/0125766 A1 offenbart Diagnosemethoden für Hardware-Debugging. Dabei wird die Berechnung möglicher Fehlerstellen basierend auf Boolescher Erfüllbarkeit beschrieben. Ferner werden Erweiterungen basierend auf quantifizierten Booleschen Formeln, hierarchischem Wissen, Abstraktion und unerfüllbaren Kernen angegeben.

[0006] Aufgabe der Erfindung ist es, ein Verfahren zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache zu schaffen, mit dem das Auffinden der Fehler im Quellcode erleichtert wird.

[0007] Diese Aufgabe wird durch das Verfahren gemäß Patentanspruch 1 gelöst. Weiterbildungen der Erfindung sind in den abhängigen Ansprüchen definiert.

[0008] Das erfindungsgemäße Verfahren dient zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache, wobei mit der Hardware-Beschreibungssprache der Aufbau und die Operation einer integrierten Schaltung beschrieben wird und wobei der fehlerhafte Quellcode zu einer fehlerhaften Ausgabe der integrierten Schaltung führt. D.h., eine mit dem Quellcode durchgeführte Simulation der integrierten Schaltung führt zu einer Ausgabe, welche von einem erwarteten Sollwert abweicht.

[0009] Im Rahmen des Verfahrens wird ein Korrekturmodell vorgegeben, welches eine hierarchische Struktur von in mehreren Hierarchieebenen angeordneten Knoten in der Form von Transformationsvorschriften umfasst, wobei eine Transformationsvorschrift eine Gruppe von Transformationen beschreibt, welche auf zumindest einen Typ eines Quellcodeabschnitts anzuwenden sind und hierdurch den Quellcodeabschnitt verändern und wobei eine Transformationsvorschrift, welche ein Kindknoten einer anderen Transformationsvorschrift ist, eine Teilmenge der Gruppe von Transformationen der anderen Transformationsvorschrift darstellt. In einer bevorzugten Variante wird die hierarchische Struktur durch einen oder mehrere Hierarchiebäume realisiert. Vorzugsweise wird die Menge von Transformationen für die jeweilige Transformationsvorschrift durch die für die Transformationsvorschrift realisierbaren Funktionen festgelegt.

[0010] Das Korrekturmodell stellt somit ein verfeinerndes Modell einer Vielzahl von Transformationsvorschriften dar, mit denen immer feiner eine Menge an Transformationen spezifiziert wird, wobei die Verfeinerung über eine Eltern-Kind-Beziehung (d.h. durch eine entsprechende Kante) in der hierarchischen Struktur repräsentiert wird. Der obige Begriff des Quellcodeabschnittes ist dabei weit zu verstehen und kann beliebige Einheiten im Quellcode, bspw. abgeschlossene Schleifen bzw. auch syntaktische Aussagen, wie z.B. einzelne Programmzeilen, oder beliebige andere Einheiten umfassen.

[0011] Im Rahmen des erfindungsgemäßen Verfahrens werden die Transformationsvorschriften (d.h. die Transformationen aus der entsprechenden Gruppe von Transformationen) aus der hierarchischen Struktur auf den fehlerhaften Quellcode angewendet und diejenigen Transformationsvorschriften ermittelt, welche den Quellcode derart verändern, dass der veränderte Quellcode zu einer korrekten Ausgabe der integrierten Schaltung führt. Eine Transformationsvorschrift führt dabei dann zu einer korrekten Ausgabe, wenn es in der durch die Transformationsvorschrift spezifizierten Gruppe von Transformationen eine Transformation gibt, die zu einer korrekten Ausgabe führt. Dabei werden als (mögliche) Korrekturen zumindest ein Teil der ermittelten Transformationsvorschriften zusammen mit dem oder den zugehörigen Quellcodeabschnitten ausgegeben, auf welche die ermittelten Transformationsvorschiften angewendet wurden. Vorzugsweise werden auch explizit die Transformationen mit deren Parametern angegeben, welche zu der korrekten Ausgabe geführt haben. In einer bevorzugten Variante werden als Korrekturen für jeden Quellcodeabschnitt, auf den die Transformationsvorschriften angewendet werden, diejenigen ermittelten Transformationsvorschriften angegeben, an welche sich keine ermittelten Transformationsvorschriften als Kindknoten anschließen.

[0012] Das erfindungsgemäße Verfahren zeichnet sich dadurch aus, dass mögliche Fehlerquellen im Quellcode einer mit einer Hardware-Beschreibungssprache simulierten integrierten Schaltung über ein sich verfeinerndes Korrekturmodell genau beschrieben werden. Die durch das Verfahren ermittelten Korrekturen geben dem Entwerfer der integrierten Schaltung wertvolle Hinweise, an welcher Stelle im Quellcode sich ein entsprechender Fehler befinden könnte und mit welcher Änderung im Quellcode dieser Fehler möglicherweise korrigiert werden kann.

[0013] In einer besonders bevorzugten Ausführungsform des erfindungsgemäßen Verfahrens werden die Korrekturen mit zugeordneten Prioritäten ausgegeben, wobei die Korrekturen für solche ermittelten Transformationsvorschriften eine höhere Priorität haben, an welche sich keine ermittelten Transformationsvorschriften als Kindknoten anschließen. Auf diese Weise wird dem Entwerfer der integrierten Schaltung vermittelt, welche Transformationsvorschriften besonders relevant bzw. prägnant sind, d.h. welche Transformationsvorschriften sich auf eine besonders kleine Teilmenge an Transformationen beziehen.

[0014] In einer besonders bevorzugten Ausführungsform der Erfindung läuft die Ermittlung der Transformationsvorschriften, welche zu einer korrekten Ausgabe der integrierten Schaltung führen, derart ab, dass die Transformationsvorschriften aufeinander folgend von den höheren zu den tieferen Hierarchieebenen auf den fehlerhaften Quellcode angewendet werden, wobei nach dem Anwenden der Transformationsvorschrift verifiziert wird, ob der hierdurch veränderte Quellcode zu einer korrekten Ausgabe der integrierten Schaltung führt, wobei nur bei einer korrekten Ausgabe Transformationsvorschriften, welche Kindknoten der angewendeten Transformationsvorschrift bilden, auf den fehlerhaften Quellcode angewendet werden. Dabei kann das Verfahren in der obersten Hierarchieebene beginnen, jedoch kann das Verfahren auch in tieferen Hierarchieebenen starten. Man macht sich in dieser Ausführungsform die Erkenntnis zunutze, dass im Falle, dass eine Transformationsvorschrift nicht zu einer korrekten Ausgabe führt, auch alle Transformationsvorschriften, welche Kindknoten dieser Transformationsvorschrift bilden, nicht zu einer korrekten Ausgabe führen können, denn es handelt sich bei den Kindknoten um eine Teilmenge der Transformationen gemäß der Transformationsvorschrift des Elternknotens.

[0015] Die Transformationsvorschriften können basierend auf verschiedenen Kriterien definiert werden. In einer bevorzugten Ausführungsform wird zwischen deterministischen und nicht-deterministischen Transformationsvorschriften unterschieden. Eine deterministische Transformationsvorschrift ist dabei in geeigneter Weise durch eine deterministische Funktion gegeben, welche von einem oder mehreren Parametern der integrierten Schaltung und insbesondere vom Inhalt des Quellcodeabschnitts abhängt, auf den die deterministische Transformationsvorschrift angewendet wird. Demgegenüber bezeichnet eine nicht-deterministische Transformationsvorschrift eine Vorschrift, welche unabhängig von einer deterministischen Funktion ist und somit Transformationen mit beliebigen, im Rahmen der Transformationsvorschrift verwendbaren Parametern umfasst. Aus dieser Definition der deterministischen und nicht-deterministischen Transformationsvorschriften ergibt sich, dass eine deterministische Transformationsvorschrift der Kindknoten einer nicht-deterministischen Transformationsvorschrift sein kann, da eine deterministische Transformationsvorschrift als eine spezielle Variante einer nicht-deterministischen Transformationsvorschrift angesehen werden kann. Der umgekehrte Fall, dass eine nicht-deterministische Transformationsvorschrift der Kindknoten einer deterministischen Transformationsvorschrift ist, ist nicht möglich.

[0016] In einer weiteren Ausführungsform des erfindungsgemäßen Verfahrens umfasst die hierarchische Struktur lokale Transformationsvorschriften, deren jeweilige Transformationen immer nur auf einen einzelnen Quellcodeabschnitt angewendet werden, wobei jede Transformation in der Transformationsvorschrift jedoch

auf verschiedene Quellcodeabschnitte vom gleichen Typ angewendet werden kann. Gegebenenfalls kann die hierarchische Struktur auch globale Transformationsvorschriften umfassen, deren Transformationen auf mehrere Quellcodeabschnitte gleichzeitig angewendet werden und z.B. Datenstrukturen verändern. Die nachfolgend erläuterten Transformationsvorschriften stellen vorzugsweise lokale Transformationsvorschriften dar.

[0017] Im Rahmen einer bevorzugten Ausführungsform des erfindungsgemäßen Verfahrens umfasst die hierarchische Struktur in der obersten Hierarchieebene eine nichtdeterministische Transformationsvorschrift, welche einen Quellcodeabschnitt durch einen neuen Quellcodeabschnitt ersetzt. Vorzugsweise umfasst diese nicht-deterministische Transformationsvorschrift zumindest eine der folgenden nicht-deterministischen Transformationsvorschriften als Kindknoten:

– eine nicht-deterministische Einzel-Ersetzungs-Transformationsvorschrift, welche einen zugewiesenen Wert einer einzelnen Zuweisung im Quellcode durch einen neuen Wert ersetzt;

– eine nicht-deterministische Mehrfach-Ersetzungs-Transformationsvorschrift, welche alle syntaktischen Aussagen und insbesondere alle Zuweisungen in einem Quellcodeabschnitt durch neue Zuweisungen ersetzt;

– eine nicht-deterministische Ergänzungs-Transformationsvorschrift, welche eine oder mehrere syntaktischen Aussagen einem Quellcodeabschnitt hinzufügt.

[0018] Unter einer Zuweisung ist hier und im Folgenden ein Ausdruck mit einem Gleichheits-Zeichen zu verstehen, wobei der zugewiesene Wert der Zuweisung die rechte Seite das Gleichheitszeichens darstellt. Der Begriff der syntaktischen Aussage ist hier und im Folgenden weit zu verstehen und kann jede Art von abgeschlossenem Inhalt im Quellcode umfassen. Vorzugsweise betrifft die syntaktische Aussage einen abgeschlossenen Ausdruck, wie z.B. ein Grundblock, eine Zuweisung bzw. eine Bedingung.

[0019] In einer weiteren, bevorzugten Ausführungsform des erfindungsgemäßen Verfahrens umfasst die oben definierte nicht-deterministische Einzel-Ersetzungs-Transformationsvorschrift zumindest eine der folgenden Transformationsvorschriften als Kindknoten:

– eine deterministische Einzel-Ersetzungs-Transformationsvorschrift, welche einen zugewiesenen Wert einer einzelnen Zuweisung (ohne Bedingung) durch einen neuen Wert ersetzt;

– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift, welche einen zugewiesenen Wert einer einzelnen Zuweisung durch einen neuen Wert unter Berücksichtigung einer Bedingung ersetzt, welche von einem nicht-deterministisch bestimmten Wert abhängt.

[0020] Unter dem Begriff „unter Berücksichtigung einer Bedingung“ ist hier und im Folgenden zu verstehen, dass die Transformationsvorschrift nur angewendet wird, wenn die Bedingung erfüllt ist. Unter einem „nicht-deterministisch bestimmten Wert“ ist hier und im Folgenden analog zur nicht-deterministischen Transformationsvorschrift ein Wert zu verstehen, der unabhängig von einer deterministischen Funktion festgelegt ist.

[0021] In einer weiteren Ausführungsform des erfindungsgemäßen Verfahrens umfasst die deterministische Einzel-Ersetzungs-Transformationsvorschrift als Kindknoten die folgende Transformationsvorschrift:

– eine deterministische Einzel-Operator-Ersetzungs-Transformationsvorschrift, welche einen Operator im zugewiesenen Wert einer einzelnen Zuweisung durch einen neuen Operator ersetzt.

[0022] In einer weiteren Variante der Erfindung umfasst die deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift zumindest eine der folgenden Transformationsvorschriften als Kindknoten:

– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift, welche einen zugewiesenen Wert einer einzelnen Zuweisung unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe (d.h. einem oder mehreren aktuellen Eingabewerten) der integrierten Schaltung abhängt;

– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift, welche einen zugewiesenen Wert einer einzelnen Zuweisung unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehrerer vergangener Eingaben der integrierten Schaltung abhängt, wobei eine vergangene Eingabe eine oder mehrere vergangene Eingabewerte umfasst.

[0023] Der aktuelle bzw. die vergangenen Zustände und die aktuelle bzw. die vergangenen Eingaben der integrierten Schaltung ergeben sich dabei im Rahmen der Simulation der integrierten Schaltung durch den Quellcode.

[0024] In einer weiteren Ausführungsform des erfindungsgemäßen Verfahrens umfasst eine nicht-deterministische Mehrfach-Ersetzungs-Transformationsvorschrift die folgende Transformationsvorschrift als Kindknoten:

– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift, welche eine syntaktische Aussage in einem Quellcodeabschnitt unter Berücksichtigung einer Bedingung ersetzt, welche von einem nicht-deterministisch bestimmten Wert abhängt.

[0025] In einer weiteren Variante umfasst die nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift zumindest eine der folgenden Transformationsvorschriften als Kindknoten:

– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift, welche eine syntaktische Aussage unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift, welche eine syntaktische Aussage unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehrerer vergangener Eingaben der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift, welche eine syntaktische Aussage in Abhängigkeit von einer vorgegebenen (d.h. bereits existierenden) Bedingung ersetzt.

[0026] In einer weiteren Variante des erfindungsgemäßen Verfahrens umfasst die nicht-deterministische Ergänzungs-Transformationsvorschrift die folgende Transformationsvorschrift als Kindknoten:

– eine nicht-deterministische bedingte Ergänzungs-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussagen einem Quellcodeabschnitt hinzufügt und diesen zugefügten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von einem nicht-deterministisch bestimmten Wert abhängt.

[0027] In einer weiteren Variante des erfindungsgemäßen Verfahrens umfasst die nicht-deterministische Ergänzungs-Transformationsvorschrift zumindest eine der folgenden Transformationsvorschriften:

– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von einem nicht-deterministisch bestimmten Wert abhängt;
– eine nicht-deterministische Kopier-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussaben (ohne Bedingung) kopiert;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von dem aktuellen Zustand und der aktuellen Eingabe der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehrerer vergangener Eingaben der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift, welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer vorgegebenen (bereits existierenden) Bedingung aktiviert.

[0028] Das erfindungsgemäße Verfahren kann auf fehlerhaften Quellcode in beliebigen Hardware-Beschreibungssprachen angewendet werden. In bevorzugten Varianten wird das Verfahren für die Hardware-Beschreibungssprachen Verilog und/oder VHDL und/oder SystemC eingesetzt, welche hinlänglich aus dem Stand der Technik bekannt sind.

[0029] Neben dem oben beschriebenen Verfahren umfasst die Erfindung ferner ein Computerprogrammprodukt mit einem auf einem maschinenlesbaren Träger gespeicherten Programmcode zur Durchführung des erfindungsgemäßen Verfahrens bzw. einer oder mehrerer bevorzugter Varianten des erfindungsgemäßen Verfahrens, wenn der Programmcode auf einem Computer ausgeführt wird. Die Erfindung betrifft darüber hinaus ein Computerprogramm mit einem Programmcode zur Durchführung des erfindungsgemäßen Verfahrens bzw. einer oder mehrerer Varianten des erfindungsgemäßen Verfahrens, wenn der Programmcode auf einem Computer ausgeführt wird.

[0030] Ausführungsbeispiele der Erfindung werden nachfolgend anhand der beigefügten Fig. 1 detailliert beschrieben. Diese Figur zeigt eine Ausführungsform einer hierarchischen Struktur in der Form eines Hierarchie-

baums aus Transformationsvorschriften, der in einer Variante des erfindungsgemäßen Verfahrens eingesetzt wird.

[0031] Im Folgenden wird das erfindungsgemäße Verfahren basierend auf der Hardware-Beschreibungssprache Verilog beschrieben, mit der eine entsprechende integrierte Schaltung bzw. ein Chip entworfen wird und mit der das Verhalten der integrierten Schaltung basierend auf einer Spezifikation einer Eingabesequenz zeitlich simuliert werden kann. Das erfindungsgemäße Verfahren ist jedoch nicht auf die Hardware-Beschreibungssprache Verilog beschränkt, sondern kann ggf. auch für andere Beschreibungssprachen zum Einsatz kommen.

[0032] Ziel der Erfindung ist die Lokalisation von Fehlern in einem fehlerhaften Quellcode einer Hardware-Beschreibungssprache, wobei der fehlerhafte Quellcode für eine bestimmte Eingabesequenz eine von der erwarteten korrekten Ausgabe abweichende Ausgabe basierend auf der Simulation der integrierten Schaltung liefert. Das erfindungsgemäße Verfahren verwendet zur Lokalisation von Fehlern im Quellcode ein verfeinerndes Korrekturmodell CM, welches als Hierarchiebaum aus einer Vielzahl von Transformationsvorschriften aufgebaut ist, die in geeigneter Weise Gruppen von Transformationen zusammenfassen. Die Transformationen verändern dabei den Quellcode in geeigneter Weise.

[0033] Fig. 1 zeigt ein Ausführungsbeispiel eines solchen Hierarchiebaums. Dieser Baum umfasst vier Hierarchieebenen, H1, H2, H3 und H4, wobei die einzelnen Knoten des Baums als Ellipsen dargestellt sind, welche mit entsprechenden Bezugszeichen die den jeweiligen Knoten zugeordnete Transformationsvorschrift nennen. Zwischen den Knoten der verschiedenen Hierarchieebenen existieren Kanten, die ausgehend von einem Elternknoten in der einen Hierarchieebene zu einem oder mehreren Kindknoten der nächsttieferen Hierarchieebene führen. Zwischen einem Elternknoten und einem Kindknoten besteht dabei eine Relation dahingehend, dass ein Kindknoten eine Verfeinerung der Transformationsvorschrift des Elternknotens darstellt. Verfeinerung bedeutet hierbei, dass die verfeinerte Transformationsvorschrift des Kindknotens eine Teilmenge der Gruppe von Transformationen darstellt, welche durch die Transformationsvorschrift des Elternknotens repräsentiert wird. Das heißt, falls mit einer Transformationsvorschrift, die der Kindknoten einer anderen Transformationsvorschrift ist, ein Fehler im Quellcode korrigiert werden kann, so kann dieser Fehler auch mit der Transformationsvorschrift gemäß dem Elternknoten korrigiert werden.

[0034] Im Folgenden werden die oben beschriebenen Transformationsvorschriften auch als generische Transformationen bezeichnet, wohingegen die darin enthaltenen Transformationen die tatsächlichen Transformationen darstellen. In der weiter unten dargestellten Tabelle 1 werden die in Fig. 1 in einzelnen Hierarchieebenen gezeigten generischen Transformationen im Detail erläutert. Die erste Spalte der Tabelle bezeichnet dabei den Namen der generischen Transformation. Die zweite Spalte enthält den Namen der generischen Transformation (d.h. den Elternknoten), welche durch die generische Transformation der ersten Spalte verfeinert wird. Die dritte Spalte nennt ein Muster, welches den Quellcodeabschnitt beschreibt, auf den die entsprechende generische Transformation gemäß der ersten Spalte angewendet wird. Die vierte Spalte gibt den entsprechend transformierten Quellcodeabschnitt nach dem Anwenden der generischen Transformation wieder. Die fünfte Spalte enthält eine textuelle Beschreibung der generischen Transformation.

[0035] Die in der Tabelle 1 gezeigten generischen Transformationen sind lediglich beispielhaft und es können auch weitere Abwandlungen und Erweiterungen von Transformationen verwendet werden. Erfindungswesentlich ist jedoch, dass die Transformationen durch eine hierarchische Struktur beschrieben werden, wobei ein Kindknoten eine Teilmenge der Gruppe von Transformationen des entsprechenden Elternknotens darstellt. Auf diese Weise wird eine effiziente Lokalisation von Fehlern im Quellcode der Hardware-Beschreibungssprache erreicht. In Fig. 1 und der Tabelle 1 wird zwischen lokalen Transformationen und globalen Transformationen unterschieden. Alle Transformationen, die von dem Knoten NDET ausgehen, sind lokale Transformationen, was bedeutet, dass diese Transformationen immer nur lokal auf einen Quellcodeabschnitt und nicht gleichzeitig auf mehrere unterschiedliche Quellcodeabschnitte im Quellcode angewendet werden. Im Unterschied zu lokalen Transformationen werden globale generische Transformationen, bei denen es sich um die Transformationen NRPL und NXTD der Fig. 1 handelt, auf mehrere Quellcodeabschnitte im Quellcode angewendet. Es wird dabei zwischen verschiedenen Arten von Quellcodeabschnitten unterschieden, die im Folgenden erläutert werden.

[0036] Ein Quellcodeabschnitt in der Form einer syntaktischen Aussage bezeichnet eine vorab festgelegte Art von Aussage im Quellcode und bezieht sich z.B. auf eine abgeschlossene Programmzeile, welche üblicherweise durch Semikolon beendet wird. Ein Satz von syntaktischen Aussagen umfasst dabei mehrere solcher Aussagen. Spezialfälle von syntaktischen Aussagen sind Ausdrücke in der Form von Variablenzuweisungen (z.B. a = b), wobei unter Variablenzuweisung auch eine entsprechende Bedingung zu verstehen ist, welche

das Argument einer If-Anweisung darstellt. In dem Pseudocode in der dritten und vierten Spalte der Tabelle 1 werden eine Mehrzahl von syntaktischen Aussagen eines Quellcodeblocks mit stmt_block bezeichnet. Eine einzelne Aussage wird mit stmt bzw. stmt1 bzw. stmt2 bezeichnet.

[0037] In der Tabelle 1 wird ferner zwischen deterministischen generischen Transformationen und nicht-deterministischen generischen Transformationen unterschieden. Eine deterministische generische Transformation ist dabei durch eine deterministische Funktion gegeben, welche in Tabelle 1 mit DET bezeichnet ist und von einem oder mehreren Parametern der integrierten Schaltung abhängt, insbesondere vom Inhalt des Quellcodeabschnitts, der durch die Transformation modifiziert wird und/oder vom aktuellen oder von vergangenen Zuständen bzw. Eingaben der simulierten integrierten Schaltung. Bei ansonsten gleichen Parametern beschreibt eine deterministische generische Transformation immer eine Teilmenge der Transformationen einer nicht-deterministischen generischen Transformation, d.h. eine nicht-deterministische Transformationsvorschrift umfasst Transformationen von allen entsprechenden deterministischen Transformationsvorschriften. Allgemein bezeichnet in der vorliegenden Anmeldung der Begriff „deterministisch“ eine Abhängigkeit von einen oder mehreren Parametern der integrierten Schaltung, wohingegen „nicht-deterministisch“ die Unabhängigkeit von Parametern der integrierten Schaltung repräsentiert. Die Syntax der dritten und vierten Spalte der Tabelle 1 ist für den Fachmann verständlich und wird nicht im Detail erläutert. Der Ausdruck a = VAR ? d:b + c bedeutet dabei, dass der Variablen a der Wert von (b+c) zugewiesen wird, falls VAR den Wert „false“ oder „0“ enthält. Sonst wird der Variablen a der Wert von der Variablen d zugewiesen. Der Ausdruck if(VAR) bedeutet, dass die darauffolgende Transformation nur durchgeführt wird, wenn VAR einen bestimmte Werte annimmt (z.B. 1, wenn VAR ein Bit darstellt).

Tabelle 1:

Name	verfeinert	wird angewendet auf	Transformierter Quellcodeabschnitt	Beschreibung
NDET		stmt_ block;	Nicht-deterministisch	Ersetze einen Satz von Aussagen in dem Quellcodeblock stmt_block durch einen nicht- deterministischen Quellcode block.
NEXP	NDET	a = b + c;	a = NEWVAR;	Ersetze einen zugewiesenen Wert einer Zuweisung bzw. einer Bedingung (d.h die rechte Seite einer Gleichung aus dem Quellcode) nicht-deterministisch durch einen neuen Wert in der Form der Variablen NEWVAR.
DET	NEXP	a = b + c;	a = DET(b, c);	Ersetze einen zugewiesenen Wert einer Zuweisung durch einen neuen deterministisch bestimmten Wert DET(b, c), der von den Variablen b und c der ursprünglichen Zuweisung ab hängt.
OP_op	DET	a = b + c;	a = b op c, wobei op z.B. –, *, ... ist	Ersetze einen Operator in einer Zuweisung durch einen neuen Operator op unter Verwendung der gleichen Variablen der ur sprünglichen Zuweisung, wobei op jeder Operator sein kann, der in dem Kontext erlaubt ist.
NCDET	NEXP	a = b + c;	a = NEWVAR ? DET (b, c):b + c;	Ersetze einen zugewiesenen Wert einer Zuweisung durch einen neuen deterministisch bestimmten Wert DET(b, c) in Abhängigkeit von dem Wert einer nicht-deterministisch zu gewiesenen Variablen NEWVAR.

CDET	NCDET	a = b + c;	a = (DET(state) ? DET (b, c):b + c);	Ersetze einen zugewiesenen Wert einer Zuweisung durch einen neuen deterministisch bestimmten Wert DET(b, c) in Abhängigkeit von dem aktuellen Zustand (state) und der aktuellen Eingabe der integrierten Schal tung, wobei der aktuelle Zustand den Gesamtzustand der Schal tung oder auch eine Untermenge der Zustandbits oder internen Signale umfassen kann, welche zur Festlegung der Bedingung ausgewählt werden.
TRCDET	NCDET	a = b + c;	a = (DET(trace) ? DET (b, c):b + c);	Ersetze einen zugewiesenen Wert einer Zuweisung durch einen neuen deterministisch bestimmten Wert DET(b, c) in Abhängigkeit von der Historie (trace) umfassend den aktuellen Zustand und die aktuelle Einga be sowie einen oder mehrere vergangene Zustände und ver gangene Eingaben der integrier ten Schaltung (diese Transfor mation ermöglicht die Änderung des Zustandsraums der integrier ten Schaltung).
NASS	NDET	stmt;	nicht-deterministisch	Ersetze alle syntaktischen Aus sagen in einem Quellcodeblock durch neue nicht- deterministische Aussagen.
NCONDIS	NASS	stmt;	if(NEWVAR) stmt;	Deaktiviere eine Aussage in Abhängigkeit von einer Bedin gung, welche von dem Wert einer nicht-deterministisch zu gewiesenen Variablen NEWVAR abhängt.
DCONDIS	NCONDIS	a = b;	if(DET(state)) stmt;	Deaktiviere eine Aussage in Abhängigkeit von einer Bedin gung, welche von dem aktuellen Zustand und der aktuellen Ein gabe der integrierten Schaltung abhängt.
TRDCONDIS	NCONDIS	a = b;	if(DET(trace))stmt;	Deaktiviere eine Aussage in Abhängigkeit von einer Bedin gung, welche von der Historie (trace) umfassend den aktuellen Zustand und die aktuelle Einga be sowie einen oder mehrere vergangene Zustände und ver gangene Eingaben der integrier ten Schaltung abhängt.
EXCONDIS	NCONDIS	a = b;	if(OP(cond))stmt;	Deaktiviere eine Aussage in Abhängigkeit von einer bereits existierenden Bedingung (OP bedeutet „cond= = true“ oder „cond = = false“).
NSTMT	NDET	stmt1; stmt2;	stmt1; NEWSTMT, stmt2;	Füge eine oder mehrere Aussagen NEWSTMT hinzu, die nicht-deterministische Funktio nen realisieren können.

NCONSTMT	NSTMT	stmt1; stmt2;	stmt1; if (NEWVAR) NEWBLOCK; stmt2;	Füge einen neuen Block an Aus sagen hinzu, wobei der Block in Abhängigkeit von einer nicht- deterministisch zugewiesenen Bedingung aktiviert wird.
NCONCP	NCONSTMT	stmt1; stmt2;	stmt1; if (NEWVAR) COPIED-BLOCK; stmt2;	Kopiere einen Block an Aussa- gen, wobei der Block in Abhän gigkeit von einer nicht- deterministisch zugewiesenen Bedingung aktiviert wird.
CP	NCONSTMT	stmt1; stmt2;	stmt1; COPIED-BLOCK; stmt2;	Kopiere einen Block von Aussagen (unabhängig von einer Be dingung).
CONCP	NCONSTMT	stmt1; stmt2;	stmt1; if (DET(state)) COPIEDBLOCK; stmt2;	Kopiere einen Block von Aussa gen, wobei der Block in Abhän gigkeit von dem aktuellen Zu stand und der aktuellen Eingabe der integrierten Schaltung akti viert wird, wobei der aktuelle Zustand den Gesamtzustand der Schaltung oder auch eine Un termenge der Zustandbits oder internen Signale umfassen kann, welche zur Festlegung der Be dingung ausgewählt werden.
TRCONCP	NCONSTMT	stmt1; stmt2;	stmt1; if (DET(trace)) COPIEDBLOCK; stmt2	Kopiere einen Block von Aussa gen, wobei der Block in Abhängigkeit von der Historie (trace) umfassend den aktuellen Zu stand und die aktuelle Eingabe sowie einen oder mehrere ver gangene Zustände und vergan gene Eingaben der integrierten Schaltung aktiviert wird (diese Transformation ermöglicht die Änderung des Zustandsraums der integrierten Schaltung).
EXCONCP	NCONSTMT	stmt1; stmt2;	stmt1; if (OP(cond)) COPIEDBLOCK; stmt2	Kopiere einen Block von Aussa gen, wobei der Block in Abhän gigkeit von einer bereits existie renden Bedingung aktiviert wird (OP bedeutet „cond= = true“ oder „cond = = false“).
NRPL		a = b; c = a + b, a = d; e = c + a;	a = NEWVAR1; c = a + b; a = NEWVAR2; e = c + a;	Ersetze jeden Schreibzugriff auf eine Variable durch eine neue nichtdeterministische Zuwei sung.
NXTD	NRPL	struct {int a, b;};	struct {int a, b; SOME-TYPE NEWVAR;}	Erweitere eine Datenstruktur durch eine neue Mitgliedsvariab le, wobei alle Berechnungen einer Variablen, welche diese Datenstruktur instantiiert, von der neuen Mitgliedsvariablen abhängen können.

[0038] Im Folgenden wird beispielhaft anhand des nachfolgenden Pseudocodes ein Ablauf einer Variante des erfindungsgemäßen Verfahrens erläutert, mit der in dem Hierarchiebaum der Fig. 1 diejenigen generischen Transformationen ermittelt werden, welche zu einer korrekten Ausgabe einer entsprechenden Spezifikation in der Form einer Eingabesequenz für eine mit der Hardware-Beschreibungssprache entworfenen integrierten Schaltung führen. Die Implementierung der Erfindung ist dabei innerhalb der Funktionsschleife function ent-

halten, welche die Funktion „debug“ definiert, die vom Design D der integrierten Schaltung, der Spezifikation S der Eingabesequenz und dem verfeinernden Korrekturmodells CM abhängt. Mittels des Pseudocodes wird dabei ein Graph G aus denjenigen Knoten des Hierarchiebaums der Fig. 1 aufgebaut, welche Transformationsvorschriften spezifizieren, mit denen basierend auf der Spezifikation S eine korrekte Ausgabe erhalten wird. In dem nachfolgenden Pseudocode wird eine Schleife, welche für mehrere Größen wiederholt wird, mit foreach/done bezeichnet. Ferner wird eine Codeausführung, welche an eine Bedingung geknüpft ist, mit if/fi bezeichnet. Der Pseudocode lautet wie folgt:

```
1  function debug (Design D, Spezifikation S, Korrekturmodell
CM)
2      Queue R = alle generischen Transformationen, welche kei-
       ne anderen generischen Transformationen verfeinern
3          Queue Q = leere Schlange
4          Graph G = leerer Graph, der „result“ speichert;
5          foreach T1∈R do
6              foreach Lokation L, wo T1 anwendbar ist do
7                  Q.append (T1, L);
8          done
9          foreach C=(T1, L) ∈Q do
10             D' = Anwenden von C auf D;
11             result= Validiere/Verifiziere (D', S);
12             if (result == valid)
13                 G.addNodeAndEdges(C);
14                 foreach T2, welche T1 verfeinert do
15                     Q.append(T2,L);
16                 done
17             fi
18         done
19         return G;
20   end function;
```

[0039] In dem obigen Pseudocode wird eine Schlange (engl. Queue) R in Zeile 2 gebildet, welche alle generischen Transformationen umfasst, die in dem Korrekturmodell CM keine anderen generischen Transformationen verfeinern, d.h. die Schlange R besteht aus den Transformationen in der obersten Hierarchieebene, welche die Hierarchieebene H1 des Hierarchiebaums der Fig. 1 ist. Gemäß Zeilen 3 bis 8 des Pseudocodes werden alle erlaubten tatsächlichen Transformationen aus der Schlange R für den fehlerhaften Quellcode gemäß dem Design D ermittelt. Eine tatsächliche Transformation bezeichnet dabei die entsprechende Transformationsvorschrift T1 mit denjenigen Lokationen L im Quellcode, auf welche die Transformationsvorschrift angewendet werden kann. Diese tatsächlichen Transformationen werden durch die Funktion Q.append der Schlange Q hinzugefügt. Schließlich werden gemäß Zeilen 9 bis 18 alle tatsächlichen Transformationen C = (T1, L) aus der Schlange Q auf den Quellcode angewendet, wodurch sich ein modifiziertes Design D‘ ergibt. Dabei wird in Zeile 12 validiert, ob gemäß der Eingabesequenz S nunmehr ein gültiges Ergebnis „result“ erhalten wird, wobei das Ergebnis dann gültig (engl. valid) ist, wenn das Design D‘ gemäß dem transformierten Quellcode eine korrekte Ausgabe liefert. Ist dies der Fall, wird die entsprechende Transformationsvorschrift als Knoten mit zugehöriger Kante in den ursprünglich leeren Graphen G aufgenommen, der auch die Verfeinerungsinformation enthält (Zeile 13). Anschließend werden dann alle verfeinernden generischen Transformationen T2 der entsprechenden Transformationsvorschrift T1, d.h. die entsprechenden Kindknoten der ursprünglichen generischen Transformation T1, zur Schlange Q mit den entsprechenden Lokationen hinzugefügt, auf welche die Transformationen T2 angewendet werden können (Zeilen 14 bis 16 des Pseudocodes). Die Schleife terminiert

erst, wenn die Schlange Q vollständig abgearbeitet wurde, also leer ist. Dann wurde der gesamte durch das CM definierte Suchraum möglicher Transformationen bearbeitet. Als Endergebnis erhält man schließlich einen Graphen G, der alle tatsächlichen Transformationen enthält.

[0040] Nachfolgend wird zur Verdeutlichung ein Beispiel des Ablaufs einer Ausführungsform des erfindungsgemäßen Verfahrens basierend auf der Hardware-Beschreibungssprache Verilog erläutert. Es wird dabei der folgende Verilog-Quellcode betrachtet:

```
1    module example (rst, clk, op, a, b,
2                                c_low, c_high);
3
4    parameter IW=2;
5    parameter DW=16;
6
7    input rst, clk;
8    input [IW-1:0] op;
9    input [DW-1:0] a, b;
10   output [DW-1:0] c_low, c_high;
11   output oflow;
12
13   reg [DW+DW:0] tmp;
14   reg [DW-1:0] c_low, c_high;
15   reg oflow;
16
17   always @ (posedge clk or posedge rst)
18   begin
19        if (rst) begin
20             tmp <= 0;
21             oflow <= 0;
22        end else
23             case (op)
24                  0: tmp = a + b;
25                  //bug -- wrong operator
26                  // 1: tmp = a - b;
27                  1: tmp = a + b;
28                  2: tmp = a * b;
29             endcase
30        {oflow, c_high, c_low} = tmp;
31   end
32
33   endmodule
```

[0041] Da Verilog als Hardware-Beschreibungssprache an sich bekannt ist, werden die obigen Befehle des Quellcodes nicht im Detail erläutert. Relevant ist lediglich, dass der Quellcode einen Fehler in Zeile 27 enthält.

Dort soll eine Subtraktion berechnet werden, wie durch die Kommentarzeilen 25 und 26 angegeben wird. Tatsächlich wird jedoch eine Addition, nämlich tmp = a + b, berechnet. Im Rahmen der Simulation der integrierten Schaltung basierend auf dem obigen Quellcode werden drei Eingabesequenzen A, B und C betrachtet, welche in nachfolgender Tabelle wiedergegeben sind:

ID	op	a	b	Ist	Soll
A	0	7	5	12	12 / ok
B	1	7	5	12	2 / Fehler
C	2	7	5	35	35 / ok

[0042] In obiger Tabelle bezeichnet die erste Spalte die Identität der Eingabesequenz und die zweite bis vierte Spalte geben die Werte der Operanden op, a und b wieder. Ferner ist in der fünften Spalte der tatsächliche Ist-Ausgabewert von tmp und in der sechsten Spalte der (korrekte) Soll-Ausgabewert von tmp angegeben. In der Eingabesequenz A nimmt der Operand op gemäß dem obigen Verilog-Quellcode den Wert 0 an, wohingegen der Operand a den Wert 7 und der Operand b den Wert 5 aufweist. Da op = 0 gilt, führt dies gemäß der Case-Anweisung in Zeile 23 dazu, dass tmp = a + b berechnet wird, so dass in Zeile 30 für {oflow, c_high, c_low} = tmp schließlich der Wert 12 ausgegeben wird. Die Eingangssequenz A führt somit zu einer korrekten Ausgabe (Istwert = Sollwert), was in der sechsten Spalte durch „ok“ angezeigt wird. Demgegenüber führt die Eingabesequenz B zum Ausgabewert 12 von tmp, was jedoch nicht der korrekten Ausgabe von tmp = 2 entspricht. Die Eingabesequenz C führt wiederum zu einem korrekten Ausgabewert.

[0043] Ausgehend von diesem Verilog-Quellcode und den entsprechenden Eingabesequenzen werden nunmehr Transformationen aus den obigen Transformationsvorschriften gemäß Fig. 1 angewandt. Beispielhaft wird dabei von den drei generischen Transformationen in NEXP, DET und OP_op ausgegangen, wobei OP_op gemäß Fig. 1 die Transformationsvorschrift DET verfeinert und die Transformationsvorschrift DET die Transformationsvorschrift NEXP verfeinert. Jede dieser generischen Transformationen ersetzt eine Zuweisung, d.h. die rechte Seite einer Gleichung, wie folgt: NDET ersetzt die Zuweisung durch eine neue, nicht-deterministisch zugewiesene Variable. Durch das Anwenden dieser generischen Transformation auf Zeile 30 des obigen Quellcodes wird eine tatsächliche Transformation erhalten, welche zu folgendem Code führt:
{oflow, c_high, c_low} = NEWVAR;

[0044] Die Transformationsvorschrift DET ersetzt eine Zuweisung durch eine deterministische Funktion, welche von den Variablen im Ausdruck der Zuweisung abhängt. Durch Anwenden dieser Transformationsvorschrift auf Zeile 30 des obigen Quellcodes wird eine tatsächliche Transformation erhalten, welche die Zeile 30 wie folgt modifiziert:
{oflow, c_high, c_low} = DET(tmp);

[0045] Die Transformationsvorschrift OP_op ersetzt einen Operator auf der rechten Seite einer Zuweisung mit einem anderen Operator. In hier beschriebenen Beispiel wird durch Anwenden dieser Transformationsvorschrift auf Zeile 30 eine Negation anstatt der Identitätsfunktion durchgeführt, was zu einer tatsächlichen Transformation führt, welche folgendem Code in Zeile 30 entspricht:
{oflow, c_high, c_low} = OP(!, tmp);

[0046] Im Rahmen der hier beschriebenen Ausführungsform wird nunmehr zunächst die Transformationsvorschrift NDET auf Zeile 30 angewendet, und anschließend werden die verfeinernden Transformationen durchgeführt. Durch Anwenden von NDET auf die Zeile 30 des obigen Quellcodes ist es möglich, eine korrekte Ausgabe dadurch zu erzeugen, dass der erwartete Soll-Ausgabewert der neuen Variablen NEWVAR zugewiesen wird. Demzufolge wird nunmehr die verfeinernde Transformationsvorschrift DET auf Zeile 30 des Quellcodes angewendet. Für die Eingabesequenz A und die Eingabesequenz B weist jedoch das interne Signal tmp den Wert 12 auf, wohingegen die Ausgabe in beiden Fällen unterschiedlich sein sollte. Dies ist mit einer deterministischen Funktion nicht zu realisieren. Somit kann über die Transformationsvorschrift DET, welche auf die Zeile 30 angewendet wird, der Fehler im Quellcode nicht behoben werden. Nichtsdestotrotz war die weniger verfeinernde Transformation NDET in der höheren Hierarchieebene erfolgreich. Demzufolge stellt NDET, welches auf Zeile 30 angewendet wird, eine Korrektur C_1 = (NDET, 29) für den fehlerhaften Quellcode dar.

[0047] Das Verfahren kann nunmehr dadurch fortgesetzt werden, dass die Transformationsvorschrift NDET auf Zeile 27 des obigen Quellcodes angewendet wird, was zu folgender Modifikation führt:
1: tmp = NEWVAR;

[0048] Diese Transformation ermöglicht wiederum die Generierung einer korrekten Ausgabe für alle Eingabesequenzen. Demzufolge wird anschließend die verfeinernde Transformationsvorschrift DET auf Zeile 27 angewendet, was zu folgender modifizierten Programmzeile führt:
1: tmp = DET (a, b);

[0049] Auch mit diesen Transformationen kann für alle Eingabesequenzen a bis c eine korrekte Ausgabe generiert werden. Wird nunmehr die Transformationsvorschrift OP_op auf Zeile 27 angewendet, wobei der Operator „+“ durch den Operator „–“ ersetzt wird, ergibt sich die folgende modifizierte Programmzeile:
1: tmp = op(–, a, b);

[0050] Auch dieser transformierte Verilog-Quellcode erzeugt für alle Eingabesequenzen den korrekten Ausgabewert. Da es keine weiteren verfeinernden Transformationen für OP_op gibt, ist eine weitere mögliche Korrektur C_2 = (OP_op, 26). Gemäß der Verfeinerungs-Relation basierend auf dem Hierarchiebaum der Fig. 1 ist die Transformationsvorschrift OP_op die am stärksten verfeinernde Transformationsvorschrift, da sie in einer tieferen Hierarchieebene als die Transformationsvorschriften DET bzw. NDET angeordnet ist. Es wird demzufolge eine Ausgabe generiert, bei der die Korrektur C_2 = (OP_op, 26) mit einer höheren Priorität zur Überprüfung durch den Designer der Schaltung eingestuft ist, als die Korrektur C_1 = (NDET, 29).

[0051] Erfindungsgemäß kann somit in geeigneter Weise eine priorisierte Ausgabe von möglichen Korrekturen für einen fehlerhaften Quellcode einer Hardware-Beschreibungssprache geschaffen werden, so dass der Designer der integrierten Schaltung hierdurch Informationen erhält, mit denen er auf schnelle Weise den Fehler im Quellcode auffinden kann.

[0052] Die erfindungsgemäße Ermittlung entsprechender Transformationsvorschriften kann auf verschiedene Arten implementiert werden, wobei nachfolgend unterschiedliche Implementierungsvarianten kurz umrissen werden. Die Umsetzung einer entsprechenden Implementierung liegt dabei im Rahmen von fachmännischem Handeln und wird deshalb nicht im Detail beschrieben.

[0053] Zur Implementierung des Verfahrens werden zwei ressourcen-intensive Schritte benötigt. Zum einen müssen die Transformationsvorschriften auf die verschiedenen Quellcodeabschnitte angewendet werden und zum anderen muss die Gültigkeit des transformierten Quellcodes in Bezug auf die entsprechend erwartete Ausgabe überprüft werden. Diese zwei Schritte können unabhängig voneinander durchgeführt werden oder ineinander verwoben werden. Folgende Techniken können zur Implementierung dieser Schritte eingesetzt werden:

- explizite simulations-basierte Ansätze, z.B.
- explizites Anwenden von Transformationen gemäß den Transformationsvorschriften und anschließendes Simulieren der integrierten Schaltung basierend auf dem transformierten Quellcode, um hierdurch die Korrektheit der Ausgabe zu überprüfen;
- eine Simulation, welche unbekannte Werte berücksichtigt;
- Path-Tracing-Verfahren
- formale Techniken, wie z.B.
- eine symbolische Simulation;
- Term-Rewriting und in And-Inverter-Graphen;
- Beweiser (engl. Reasoning Engine) zum Ableiten von Schlussfolgerungen (Boolean Satisfiability (SAT), Satisfiabliltiy Modulo Theories (SMT), Binary Decision Diagrams (BDD), Automatic Test Pattern Generation (ATPG));
- hybride Techniken, wie z.B.
- Concolic Simulation
- Symbolic Trajectory Evaluation

[0054] All diese Verfahren erfordern einige Modifikationen, um die tatsächlichen Transformationen durchzuführen und zu überprüfen. Dies wird im Folgenden für simulationsbasierte Ansätze, formale Techniken und hybride Techniken erläutert.

[0055] Im Rahmen eines simulationsbasierten Ansatzes wird die tatsächliche Transformation aus den entsprechenden Transformationsvorschriften auf den Quellcode angewendet, was zu einer modifizierten Beschreibung führt, auf deren Basis die integrierte Schaltung simuliert werden kann. Die Überprüfung der Gültigkeit der Ausgabe der simulierten Schaltung ist für solche tatsächlichen Transformationen, welche direkt einen logischen Abschnitt des Quellcodes durch eine neue Logik ersetzen, problemlos möglich. In diesem Fall wird eine Standard-Simulation durchgeführt, um das Verhalten der integrierten Schaltung basierend auf dem transformierten

Design zu überprüfen. Falls keine Logik eingefügt wird, z.B. durch die generische Transformation NDET oder DET aus Fig. 1, muss die Simulation modifiziert werden. Für NDET werden mögliche Parameter für diese Transformation aufgelistet und die Ausgabe basierend auf diesen Parametern über eine Simulation überprüft. Für DET werden zusätzlich partielle Wahrheits-Tabellen generiert, um zu überprüfen, ob die Parameter der Transformation deterministisch aus anderen Signalen in dem Schaltungs-Design generiert werden können.

[0056] Alternativ kann das Simulations-basierte Verfahren auf einer Netzliste durchgeführt werden. Dazu wird der Quellcode in eine Netzliste synthetisiert. Während dieses Synthese-Schrittes werden die Beziehungen zwischen Quellcode und Elementen in der Netzliste erhalten. Dadurch können mögliche Korrekturen auf der Netzliste auf den Quellcode abgebildet werden. Dann werden eine oder mehrere Transformationsvorschriften direkt auf die Netzliste angewandt und per Simulation wird wie oben das Ergebnis geprüft.

[0057] Im Rahmen der Verwendung von formalen Techniken kann eine tatsächliche Transformation derart angewendet werden, dass der Quellcode direkt geändert wird und ein formales Modell aus dem transformierten Quellcode generiert wird, welches anschließend von einem Beweiser verarbeitet wird. Alternativ können mehrere tatsächliche Transformationen symbolisch zu dem formalen Modell hinzugefügt werden, welches aus dem ursprünglichen Quellcode erzeugt wurde. Hierdurch wird eine Vielzahl von tatsächlichen Transformationen in einem einzelnen Durchlauf des Beweisers analysiert. Für einfache Logik-Ersetzungen kann die Transformation wiederum direkt in den Quellcode eingefügt werden. Für die generische Transformation NDET werden neue symbolische Variablen eingeführt, wobei Wertezuweisungen vom Beweiser übernommen werden. Für die generische Transformation DET werden zusätzliche Beschränkungen zu dem formalen Modell hinzugefügt, um das deterministische Verhalten zu garantieren. Einige Beweiser stellen direkte Mechanismen zur Verfügung, um solche Beschränkungen zu formulieren, bspw. modellieren SMT-Solver sog. „uninterpreted functions“.

[0058] Die Überprüfung der Gültigkeit eines transformierten Designs kann bei formalen Techniken durch Vergleich der Spezifikation einer entsprechenden Eingabesequenz mit der transformierten integrierten Schaltung erreicht werden. Der Beweiser findet dabei Widersprüche zwischen der Spezifikation und dem transformierten Schaltungs-Design. Ist die Spezifikation bspw. durch erwartete Werte für Eingabesequenzen gegeben, welche zu einer fehlerhaften Ausgabe führen, können eine einzelne Eingabesequenz oder mehrere Eingabesequenzen gleichzeitig durch ein einzelnes formales Modell verarbeitet werden.

[0059] Im Rahmen des erfindungsgemäßen Verfahrens können auch hybride Techniken eingesetzt werden, welche sowohl auf Simulationen als auch auf formalen Techniken basieren. Diese Techniken reduzieren typischerweise die Mächtigkeit des Beweisverfahrens im Vergleich zu einem formalen Beweiser, was zu weniger Rechenkomplexität führt. Welche Teile des Problems durch formale Methoden und welche durch simulationsbasierte Methoden verarbeitet werden, kann im Wesentlichen über Heuristiken angepasst werden.

[0060] Im schlechtesten Fall muss jede generische Transformation auf jeden möglichen Quellcodeabschnitt in dem Schaltungs-Design angewendet werden. Die Anzahl an tatsächlichen Transformationen wird somit sehr groß. Es können deshalb ggf. weitere Optimierungs-Techniken eingesetzt werden, um diese Anzahl zu reduzieren. Dabei sind strukturelle Ansätze bekannt, welche die Struktur von Kontroll- und Daten-Fluss-Graphen analysieren. Hierdurch kann die Anzahl der Quellcodeabschnitte reduziert werden, für welche z.B. NDET angewendet werden muss.

[0061] Das im Vorangegangenen beschriebene erfindungsgemäße Verfahren weist eine Reihe von Vorteilen auf. Insbesondere ermöglicht das Verfahren die Analyse von fehlerhaftem Quellcode einer Hardware-Beschreibungssprache dahingehend, dass mögliche Korrekturen unter Verwendung eines hierarchisch aufgebauten Korrekturmodells aus einer Vielzahl von Transformationsvorschriften bestimmt werden. Die hierarchische Struktur der Transformationsvorschriften ist dabei derart aufgebaut, dass eine Transformationsvorschrift, welche der Kindknoten einer Transformationsvorschrift einer höheren Hierarchieebene ist, eine Teilmenge der Transformationen gemäß der Transformationsvorschrift der höheren Hierarchieebene darstellt. D. h., die Fehlerursache wird durch die Verwendung der hierarchischen Struktur in geeigneter Weise immer mehr eingeschränkt. Durch eine priorisierte Ausgabe möglicher Korrekturen, bei der Korrekturen aus tieferen Hierarchieebenen eine höhere Priorität erhalten, wird dem Designer der integrierten Schaltung auch vermittelt, welche Transformationsvorschriften und damit verknüpfte Korrekturen besonders prägnant in dem Sinne waren, dass sie bereits kleine Teilmengen von Transformationen spezifizieren.

ZITATE ENTHALTEN IN DER BESCHREIBUNG

Diese Liste der vom Anmelder aufgeführten Dokumente wurde automatisiert erzeugt und ist ausschließlich zur besseren Information des Lesers aufgenommen. Die Liste ist nicht Bestandteil der deutschen Patent- bzw. Gebrauchsmusteranmeldung. Das DPMA übernimmt keinerlei Haftung für etwaige Fehler oder Auslassungen.

Zitierte Patentliteratur

- US 5862361 A [0003]
- DE 10243598 A1 [0004]
- US 2009/0125766 A1 [0005]

Patentansprüche

1. Verfahren zur rechnergestützten Analyse von fehlerhaftem Quellcode in einer Hardware-Beschreibungssprache, wobei mit der Hardware-Beschreibungssprache der Aufbau und die Operation einer integrierten Schaltung beschrieben wird und der fehlerhafte Quellcode zu einer fehlerhaften Ausgabe der integrierten Schaltung führt, bei dem:
– ein Korrekturmodell (CM) vorgegeben wird, welches eine hierarchische Struktur von in mehreren Hierarchieebenen (H1, H2, H3, H4) angeordneten Knoten in der Form von Transformationsvorschriften umfasst, wobei eine Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) eine Gruppe von Transformationen beschreibt, welche auf zumindest einen Typ eines Quellcodeabschnitts anzuwenden sind und hierdurch den Quellcodeabschnitt verändern und wobei eine Transformationsvorschrift (NDET, NRPL, ..., EXCONCP), welche ein Kindknoten (NDET, NRPL, ..., EXCONCP) einer anderen Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) ist, eine Teilmenge der Gruppe von Transformationen der anderen Transformationsvorschrift (NDET, NRPL, ..., EXCONCP) darstellt;
– die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) aus der hierarchischen Struktur auf den fehlerhaften Quellcode angewendet werden und diejenigen Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) ermittelt werden, welche den Quellcode derart verändern, dass der veränderte Quellcode zu einer korrekten Ausgabe der integrierten Schaltung führt, wobei als Korrekturen zumindest ein Teil der ermittelten Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) zusammen mit dem oder den zugehörigen Quellcodeabschnitten ausgegeben werden, auf welche die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) angewendet wurden.

2. Verfahren nach Anspruch 1, bei dem die Korrekturen mit zugeordneten Prioritäten ausgegeben werden, wobei die Korrekturen für solche ermittelten Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) eine höhere Priorität haben, an welche sich keine ermittelten Transformationsvorschriften als Kindknoten anschließen.

3. Verfahren nach einem der vorhergehenden Ansprüche, bei dem im Rahmen der Ermittlung der Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) aufeinander folgend die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) von den höheren zu den tieferen Hierarchieebenen (H1, H2, H3, H4) auf den fehlerhaften Quellcode angewendet werden, wobei nach dem Anwenden einer Transformationsvorschrift (H1, H2, H3, H4) verifiziert wird, ob der hierdurch veränderte Quellcode zu einer korrekten Ausgabe der integrierten Schaltung führt, wobei nur bei einer korrekten Ausgabe Transformationsvorschriften, welche Kindknoten der angewendeten Transformationsvorschrift bilden, auf den fehlerhaften Quellcode angewendet werden.

4. Verfahren nach einem der vorhergehenden Ansprüche, bei dem die Transformationsvorschriften (NDET, NRPL, ..., EXCONCP) deterministische und/oder nicht-deterministische Transformationsvorschriften umfassen, wobei eine deterministische Transformationsvorschrift durch eine deterministische Funktion gegeben ist, welche von einem oder mehreren Parametern der integrierten Schaltung und insbesondere vom Inhalt des Quellcodeabschnitts abhängt, auf dem die deterministische Transformationsvorschrift angewendet wird, und wobei die nicht-deterministische Transformationsvorschrift unabhängig von einer deterministischen Funktion ist, welche von einem oder mehreren Parametern der integrierten Schaltung abhängt.

5. Verfahren nach einem der vorhergehenden Ansprüche, bei dem die hierarchische Struktur lokale Transformationsvorschriften umfasst, deren Transformationen immer nur auf einen einzelnen Quellcodeabschnitt im Quellcode angewendet werden.

6. Verfahren nach einem der vorhergehenden Ansprüche, bei dem die hierarchische Struktur globale Transformationsvorschriften umfasst, deren Transformationen auf mehrere Quellcodeabschnitte im Quellcode gleichzeitig angewendet werden.

7. Verfahren nach einem der vorhergehenden Ansprüche in Kombination mit Anspruch 4, bei dem die hierarchische Struktur in der obersten Hierarchieebene (H1) eine nicht-deterministische Transformationsvorschrift (NDET) umfasst, welche einen Quellcodeabschnitt durch einen neuen Quellcodeabschnitt ersetzt.

8. Verfahren nach Anspruch 7, bei dem die nicht-deterministische Transformationsvorschrift (NDET) in der obersten Hierarchieebene (H1) zumindest eine der folgenden nicht-deterministischen Transformationsvorschriften (NEXP, NASS, NSTMT) als Kindknoten umfasst:
– eine nicht-deterministische Einzel-Ersetzungs-Transformationsvorschrift (NEXP), welche einen zugewiesenen Wert einer einzelnen Zuweisung im Quellcode durch einen neuen Wert ersetzt;

– eine nicht-deterministische Mehrfach-Ersetzungs-Transformationsvorschrift (NASS), welche alle syntaktischen Aussagen in einem Quellcodeabschnitt durch neue Aussagen ersetzt;
– eine nicht-deterministische Ergänzungs-Transformationsvorschrift (NSTMT), welche eine oder mehrere syntaktische Aussagen einem Quellcodeabschnitt hinzufügt.

9. Verfahren nach Anspruch 8, bei dem die nicht-deterministische Einzel-Ersetzungs-Transformationsvorschrift (NEXP) zumindest eine der folgenden Transformationsvorschriften (DET, NCDET) als Kindknoten umfasst:
– eine deterministische Einzel-Ersetzungs-Transformationsvorschrift (DET), welche einen zugewiesenen Wert einer einzelnen Zuweisung im Quellcode durch einen neuen Wert ersetzt;
– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift (NCDET), welche einen zugewiesenen Wert einer einzelnen Zuweisung im Quellcode durch einen neuen Wert unter Berücksichtigung einer Bedingung ersetzt, welche von einem nicht-deterministisch bestimmten Wert abhängt.

10. Verfahren nach Anspruch 9, bei dem die deterministische Einzel-Ersetzungs-Transformationsvorschrift (DET) die folgende Transformationsvorschrift (OP_op) als Kindknoten umfasst:
– eine deterministische Einzel-Operator-Ersetzungs-Transformationsvorschrift (OP_op), welche einen Operator im zugewiesenen Wert einer einzelnen Zuweisung durch einen neuen Operator ersetzt.

11. Verfahren nach Anspruch 9 oder 10, bei dem die deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift (NCDET) zumindest eine der folgenden Transformationsvorschriften (CDNET, TRCDNET) als Kindknoten umfasst:
– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift (CDET), welche einen zugewiesenen Wert einer einzelnen Zuweisung im Quellcode durch einen neuen Wert unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe der integrierten Schaltung abhängt;
– eine deterministische bedingte Einzel-Ersetzungs-Transformationsvorschrift (TRCDET), welche eine zugewiesene Variable einer einzelnen Zuweisung unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehreren vergangenen Eingaben der integrierten Schaltung abhängt.

12. Verfahren nach einem der Ansprüche 8 bis 11, bei dem die nicht-deterministische Mehrfach-Ersetzungs-Transformationsvorschrift (NASS) die folgende Transformationsvorschrift (NCONDIS) als Kindknoten umfasst:
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift (NCONDIS), welche eine syntaktische Aussage in einem Quellcodeabschnitt unter Berücksichtigung einer Bedingung ersetzt, welche von einem nicht-deterministisch bestimmten Wert abhängt.

13. Verfahren nach Anspruch 12, bei dem die nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift (NCONDIS) zumindest eine der folgenden Transformationsvorschriften (DCONDIS, TRDCONDIS, EXCONDIS) als Kindknoten umfasst:
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift (DCONDIS), welche eine syntaktische Aussage unter Berücksichtigung einer Bedingung ersetzt, welche vom aktuellen Zustand und der aktuellen Eingabe der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift (TRDCONDIS), welche eine syntaktische Aussage unter Berücksichtigung einer Bedingung ersetzt, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehrerer vergangener Eingaben der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Deaktivierungs-Transformationsvorschrift (EXCONDIS), welche eine syntaktische Aussage in Abhängigkeit von einer vorgegebenen Bedingung ersetzt.

14. Verfahren nach einem der Ansprüche 8 bis 13, bei dem die nicht-deterministische Ergänzungs-Transformationsvorschrift (NSTMT) die folgende Transformationsvorschrift (NCONSTMT) als Kindknoten umfasst:
– eine nicht-deterministische bedingte Ergänzungs-Transformationsvorschrift (NCONSTMT), welche eine oder mehrere syntaktische Aussagen einem Quellcodeabschnitt hinzufügt und die hinzugefügten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von einem nicht-deterministisch bestimmten Wert abhängt.

15. Verfahren nach Anspruch 14, bei dem nicht-deterministische bedingte Ergänzungs-Transformationsvorschrift (NCONSTMT) zumindest eine der folgenden Transformationsvorschriften (NCONCP, CP, CONCP, TRCONCP, EXCONCP) als Kindknoten umfasst:

– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift (NCONCP), welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von einem nicht-deterministisch bestimmten Wert abhängt;
– eine nicht-deterministische Kopier-Transformationsvorschrift (CP), welche eine oder mehrere syntaktische Aussagen kopiert;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift (CONCP), welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von dem aktuellen Zustand und der aktuellen Eingabe der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift (TRCONCP), welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer Bedingung aktiviert, welche von dem aktuellen Zustand und der aktuellen Eingabe und einem oder mehreren vergangenen Zuständen und einer oder mehrerer vergangener Eingaben der integrierten Schaltung abhängt;
– eine nicht-deterministische bedingte Kopier-Transformationsvorschrift (EXCONCP), welche eine oder mehrere syntaktische Aussagen kopiert und die kopierten syntaktischen Aussagen unter Berücksichtigung einer vorgegebenen Bedingung aktiviert.

16. Verfahren nach einem der vorhergehenden Ansprüche, bei dem Hardware-Beschreibungssprache Verilog und/oder VHDL und/oder SystemC umfasst.

17. Computerprogrammprodukt mit einem auf einem maschinenlesbaren Träger gespeicherten Programmcode zur Durchführung eines Verfahrens nach einem der vorhergehenden Ansprüche, wenn der Programmcode auf einem Computer ausgeführt wird.

18. Computerprogramm mit einem Programmcode zur Durchführung eines Verfahrens nach einem der Ansprüche 1 bis 16, wenn der Programmcode auf einem Computer ausgeführt wird.

Es folgt ein Blatt Zeichnungen

Anhängende Zeichnungen

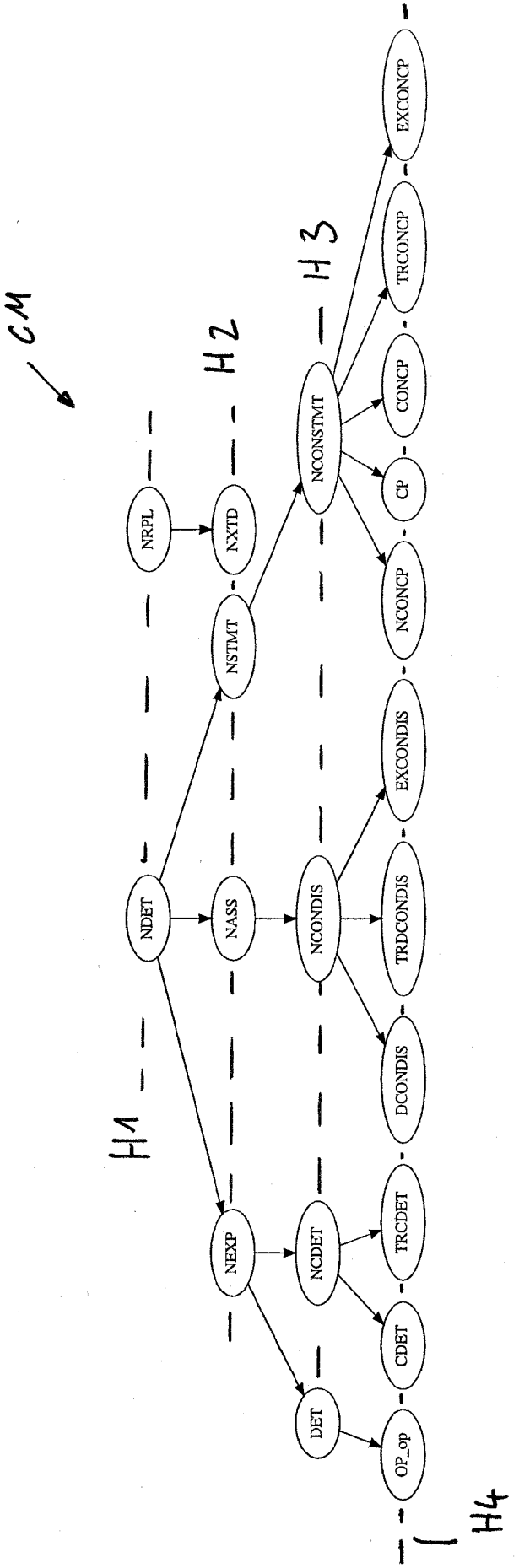


Fig. 1