(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 698 35 106 T2** 2006.12.07

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 233 341 B1**
(21) Deutsches Aktenzeichen: **698 35 106.1**
(96) Europäisches Aktenzeichen: **02 009 367.0**
(96) Europäischer Anmeldetag: **18.11.1998**
(97) Erstveröffentlichung durch das EPA: **21.08.2002**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **28.06.2006**
(47) Veröffentlichungstag im Patentblatt: **07.12.2006**

(51) Int Cl.[8]: ***G06F 11/25*** (2006.01)
G01R 31/3177 (2006.01)
G01R 31/3185 (2006.01)

(30) Unionspriorität:

65602 P	**18.11.1997**	**US**
186607	**06.11.1998**	**US**

(84) Benannte Vertragsstaaten:
DE, FR, GB, IT, NL

(73) Patentinhaber:
Altera Corp., San Jose, Calif., US

(74) Vertreter:
Benedum, U., Dipl.-Chem.Univ.Dr.rer.nat., Pat.-Anw., 80333 München

(72) Erfinder:
Veenstra, Kerry, San Jose, California 95132, US;
Rangasayee,Krishna, Pleasanton, CA 94566, US;
Herrmann, Alan L., Sunnyvale, California 94087, US

(54) Bezeichnung: **Eingebetteter logischer Analysator**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

GEBIET DER ERFINDUNG

[0001] Die Erfindung betrifft allgemein die Untersuchung einer Hardwarevorrichtung in Verbindung mit einem Computersystem. Sie betrifft insbesondere einen Logikanalysator, der zum Zweck der Fehlersuche automatisch in eine Hardwarevorrichtung eingebettet wird.

HINTERGRUND DER ERFINDUNG

[0002] Auf dem Gebiet der Elektronik sind verschiedene elektronische Werkzeuge zur Entwurfsautomatisierung (EDA, EDA = Electronic Design Automation) nützlich für das Automatisieren des Vorgangs, durch den integrierte Schaltungen, Multi-Chip-Module, Platinen usw. entworfen und gefertigt werden. Elektronische Werkzeuge zur Entwurfsautomatisierung eignen sich ganz besonders für den Entwurf von integrierten Standardschaltungen, kundenspezifischen integrierten Schaltungen (z.B. ASICs) und den Entwurf von kundenspezifischen Anordnungen für programmierbare integrierte Schaltungen. Integrierte Schaltungen, die vom Kunden programmierbar sind, damit er seinen eigenen Entwurf herstellen kann, enthalten programmierbare Logikvorrichtungen (PLDs, PLD = Programmable Logic Device). Der Ausdruck programmierbare Logikvorrichtungen bezeichnet beliebige integrierte Schaltungen, die man programmieren kann, damit sie eine gewünschte Funktion ausführen, und die programmierbare Logikanordnungen (PLAs, PLA = Programmable Logic Array), PALs (PAL = Programmable Array Logic), feldprogrammierbare Gatearrays (FPGAs, FPGA = Field Programmable Gate Array), komplexe programmierbare Logikvorrichtungen (CPLDs, CPLD = Complex Programmable Logic Device) sowie zahlreiche unterschiedliche Logikvorrichtungen und Speichervorrichtungen enthalten, die man programmieren kann. Derartige PLDs werden häufig von einem Entwurfsingenieur mit Hilfe eines elektronischen Werkzeugs zur Entwurfsautomatisierung entworfen und programmiert, das die Form eines Softwarepakets hat.

[0003] Im Zuge des Erzeugens eines PLD-Entwurfs, des Programmierens des PLD und des Prüfens seiner Funktionalität auf der Platine oder in dem System, für das es gedacht ist, ist es wichtig, dass man die Fehler im PLD finden kann, da ein Entwurf im ersten Anlauf selten perfekt ist. Bevor ein PLD tatsächlich mit einem elektronischen Entwurf programmiert wird, kann man eine Simulation und/oder eine Timing-Untersuchung dazu verwenden, Fehler im elektronischen Entwurf zu finden. Ist das PLD einmal programmiert und arbeitet es in einem laufenden System, so ist es ebenfalls wichtig, dass man Fehler im PLD in dieser realen Umgebung aufspüren kann.

[0004] Obgleich man eine Simulation dafür einsetzen kann, zahlreiche Aspekte eines PLD zu überprüfen, ist es nahezu unmöglich, eine Simulation zu erzeugen, die präzise alle Eigenschaften des PLD auf einer tatsächlichen Platine durchspielt, die in einem komplizierten System arbeitet. Beispielsweise kann eine Simulation nicht in der Lage sein, die Timing-Merkmale zu liefern, die denen gleichen, die in einem laufenden System tatsächlich auf das PLD einwirken, d.h. die Simulations-Timingsignale können mehr oder weniger von den Signalen abweichen, die in einem realen System tatsächlich auf das PLD einwirken.

[0005] Zusätzlich zu den Schwierigkeiten beim Erzeugen einer umfassenden Simulation verursachen weitere Platinenvariablen wie Temperaturänderungen, Kapazitäten, Rauschen und weitere Faktoren aussetzende Fehler in einem PLD, die nur dann in Erscheinung treten, wenn das PLD in einem laufenden System arbeitet. Zudem kann es schwierig sein, ausreichend unterschiedliche Testvektoren zu erzeugen, die den PLD-Entwurf bis an den Punkt belasten, an dem man vermutlich die meisten Fehler beobachtet. Beispielsweise kann eine Fehlfunktion eines PLD auftreten, wenn an das PLD Eingangssignale angelegt werden, die der Entwickler nicht erwartet hat und daher während des Entwurfs und der Simulation des PLD nicht in Betracht gezogen hat. Derartige Fehlfunktionen sind schwer vorherzusehen und müssen im Zusammenhang mit dem vollständigen System beseitigt werden. Daher ist die Simulation eines elektronischen Entwurfs nützlich, kann jedoch in der Regel nicht alle Fehler in einem PLD finden.

[0006] Ein Ansatz zum Aufspüren von Fehlern in einer Hardwarevorrichtung innerhalb eines laufenden Systems besteht darin, ein eigenes Hardwaregerät zu verwenden, das Logikanalysator heißt und der Analyse von Signalen dient, die an den Anschlussstiften einer Hardwarevorrichtung auftreten. (Ein Beispiel hierfür ist der HP1670A Series Logic Analyzer der Firma Hewlett-Packard.) In der Regel verbindet man von Hand eine Anzahl Fühlerdrähte des Logikanalysators mit interessierenden Anschlussstiften der Hardwarevorrichtung, damit man die Signale an diesen Anschlussstiften überwachen kann. Der Logikanalysator erfasst und speichert diese Signale. Der Gebrauch eines externen Logikanalysators zum Überwachen der Anschlussstifte einer Hardwarevorrichtung unterliegt jedoch gewissen Einschränkungen, wenn es um die Fehlersuche in einer derartigen Vorrichtung geht. Beispielsweise kann ein solcher externer Logikanalysator nur an die externen Anschlussstifte der Hardwarevorrichtung angeschlossen werden und diese überwachen. Auf interne Signale der Hardwarevorrichtung kann man nicht zugreifen, und man kann sie nicht überwachen. Unglücklicherweise wäre es beim Programmieren einer Hardwarevorrichtung in Form eines PLD jedoch sehr nütz-

lich, wenn man einige dieser internen Signale überwachen könnte, um Fehler im PLD zu finden.

[0007] Gewisse kundenspezifische Hardwarevorrichtungen können einer gewissen internen Fehlersuche-Hardware ausgeliefert werden. Diese Fehlersuche-Hardware ist jedoch in aller Regel fest verdrahtet und führt bestimmte interne Signale, die sich nicht leicht durch einen Ingenieur verändern lassen, der andere Signale sehen will. Mit einer derartigen eingebauten Fehlersuche ist es zudem nicht möglich, ein beliebiges vom Ingenieur gewünschtes Signal zu wählen, und der Ingenieur kann auch die Triggersignale und die Triggerbedingungen nicht verändern. Da ein PLD seiner ureigensten Natur nach eine programmierbare Vorrichtung ist, die ein Ingenieur zu programmieren versucht, damit sie eine bestimmte Funktion ausführt, ist es für den Ingenieur wichtig, dass er überwachte Signale, Triggersignale und Triggerbedingungen individuell zuschneiden kann, so dass er in jeder beliebigen Vorrichtung wirksam Fehler suchen kann. Weiterhin ist das Erzeugen eines elektronischen Entwurfs für ein PLD ein iterativer Vorgang, der eine kreative Fehlersuche von einem Ingenieur fordert, der vielleicht nahezu jedes interne Signal sehen will, und der im Verlauf des Fehlersuchvorgangs bei einem PLD innerhalb eines Systems seine Absichten recht häufig ändert. Bekannte externe und interne Logikanalysatoren bieten diese Flexibilität nicht.

[0008] Ein weiterer Nachteil beim Gebrauch eines externen Logikanalysators oder einer fest verdrahteten vorbestimmten Fehlersuch-Hardware innerhalb eines individuellen Chips besteht darin, dass die Anzahl interner Signale, die ein Ingenieur zu überwachen wünscht, häufig größer ist als die Anzahl der verfügbaren Anschlussstifte an der Vorrichtung. Wünscht der Ingenieur beispielsweise sechzehn interne Signale an einer Vorrichtung zu überwachen, so kann er dies mit Hilfe eines externen Logikanalysators nicht tun, falls die Vorrichtung nur vier Anschlussstifte für die Fehlersuche zur Verfügung hat.

[0009] In manchen Fällen ist es einem Ingenieur möglich, einen herkömmlichen Logikanalysator zum Untersuchen eines internen Signals eines PLD zu verwenden. Dies kann zum Beispiel dadurch erreicht werden, dass ein Ingenieur seinen Entwurf so abwandelt, dass ein normalerweise internes Signal zeitweilig mit einem Ausgabestift des PLD verbunden wird. Der Entwurf wird daraufhin neu übersetzt. Der Ingenieur befestigt dann einen Fühler an diesem Ausgabestift, damit er das "interne" Signal überwachen kann. Dabei ist es unangenehm, dass der Ingenieur seinen Entwurf neu übersetzen muss und das PLD neu programmieren muss, damit er dieses interne Signal sehen kann. Nach dem Abschluss der Fehlersuche muss der Ingenieur seinen Entwurf erneut überarbeiten, um das interne Signal von dem Ausgabestift zu entfernen, den Entwurf neu übersetzen und schließlich das PLD nochmals neu programmieren. Dies kann ein ermüdender Vorgang sein.

[0010] Selbst wenn es einem Ingenieur gelingt, ein internes Signal an einen Ausgabestift eines PLD zu leiten, kann es bei einigen Gehäusen für integrierte Schaltungen äußerst schwierig sein, einen externen Logikanalysator anzubringen. Bei einer integrierten Schaltung mit Dual-Inline-Gehäuse kann es ein relativ einfacher Vorgang sein, die Sonden des Logikanalysators oben auf dem Gehäuse zu befestigen, solange sich das Gehäuse an einer einfach zugänglichen Stelle auf der Platine befindet. Befindet sich das Gehäuse jedoch wegen einer überfrachteten Platine an einer schwer zugänglichen Stelle, kann es schwierig werden, die Sonden des Logikanalysators mechanisch an bestimmten interessierenden Ausgabestiften zu befestigen. Noch mehr Schwierigkeiten bereiten integrierte Schaltungen mit Reihen von Miniaturkontakten, die sich auf der Gehäuseoberseite befinden (z.B. "Flip-Chips"). Bei diesem Gehäusetyp ist es schwierig, die Fühler eines Logikanalysators an bestimmten interessierenden Ausgängen zu befestigen. Manche integrierte Schaltungen sind in ein Perlenrastergehäuse eingeschlossen, wobei sich die Kontakte an der Gehäuseunterseite gegenüber der Platine befinden. Bei diesen Gehäusen kann es nahezu unmöglich sein, die Fühler eines Logikanalysators an diesen kleinen Kontakten zu befestigen, die sich an der Unterseite des Gehäuses befinden. Damit weist der Gebrauch eines externen Logikanalysators auch dann Unzulänglichkeiten auf, wenn man ein internes Signal mit einem Anschlussstift einer Vorrichtung verbinden kann.

[0011] Gewisse Logikanalysatoren erlauben es einem Benutzer beispielsweise, eine Triggerbedingung und einen Satz Triggersignale festzulegen, die die Triggerbedingung erfüllen müssen, bevor der Logikanalysator für die Datenerfassung eingeschaltet wird. Derartige Logikanalysatoren sind nützlich, wenn man wünscht, Signaldaten zu erfassen und zu analysieren, die unmittelbar nach einer bestimmten Triggerbedingung auftreten (beispielsweise nach dem Versagen der Vorrichtung). Oft wünscht man jedoch, Signale für die spätere Untersuchung zu erfassen, die vor der Triggerbedingung auftreten. In den meisten Fällen sind diejenigen Logikanalysatoren, die mit der Datenerfassung nach dem Erfüllen einer Triggerbedingung beginnen, nicht in der Lage, erfasste Signale vor der Triggerbedingung zu liefern, da der Logikanalysator lediglich dafür entworfen ist, mit der Erfassung bei einem Fehler oder Versagen oder einer anderen Triggerbedingung zu beginnen. Da diese Fehler und/oder Versagensfälle unvorhersehbar erfolgen, können diese Logikanalysatortypen die Triggerbedingung nicht vorhersehen und sie können damit nicht mit der Datenerfassung beginnen, bevor die Triggerbedingung auftritt.

[0012] In gewissen Fehlersuchsituationen kann es äußerst vorteilhaft sein, Signale zu erfassen, die auftreten, bevor die Triggerbedingung eintritt. Werden beispielsweise Fehler in einer PCI-Bus-Schnittstelle gesucht, so kann ein Zustand auftreten, in dem die Schnittstelle in einen unerlaubten Status geht. Herkömmliche Logikanalysatoren sind in der Lage, diesen unerlaubten Status zu erkennen und sofort mit der Datenerfassung für die spätere Analyse zu beginnen. Es ist jedoch äußerst erwünscht, mit dem Erfassen der Signaldaten zu beginnen, bevor die Busschnittstelle in den unerlaubten Status eintritt, damit man feststellen kann, warum der Bus in diesen unerlaubten Status gelangt. In einem anderen Beispiel kann es beim Auftreten eines Interrupts äußerst erwünscht sein, die Vorgeschichte gewisser Register zu kennen, bevor der Interrupt erfolgt. Anders ausgedrückt kann nach dem Empfang des Interrupts die Datenerfassung beginnen, die Register können sich jedoch bereits in einem inkorrekten Zustand befinden. Es ist äußerst erwünscht, dass man die Signaldaten erfassen und analysieren kann, bevor der Interrupt erfolgt, damit man feststellen kann, warum gewisse Register in einem inkorrekten Zustand sind, wenn der Interrupt auftritt. Weitere Situationen, in denen es erwünscht ist, Signaldaten vor einer bestimmten Triggerbedingung zu erfassen, sind auch möglich.

[0013] Diverse bisher unternommene Versuche liefern Teillösungen, haben jedoch jeweils ihre Nachteile. Beispielsweise erlauben es die externen Logikanalysatoren, die bei der Firma Hewlett-Packard erhältlich sind, Signaldaten zu erfassen, bevor eine Triggerbedingung (oder ein Haltepunkt) auftritt. Unangenehmerweise leiden diese externen Logikanalysatoren unter zahlreichen Nachteilen, die wie beschrieben den externen Logikanalysatoren anhaften. Die Actel Corporation in Sunnyvale, Kalifornien, bietet zwei Fühler innerhalb einer programmierbaren Logikvorrichtung an, die zwei unterschiedliche Signale überwachen können. Diese Signale muss der Benutzer jedoch vorab festlegen, und sie können nicht flexibel anderen Signalen zugewiesen werden. Zudem erlauben die Fühler von Actel eine konstante Überwachung bestimmter Signale; sie erlauben jedoch nicht das Erfassen von einschlägigen Signaldaten bezüglich eines festgelegten Haltepunkts.

[0014] Man wünscht sich daher, eine Einrichtung und eine Vorgehensweise zu besitzen, die es einem in eine programmierbare Logikvorrichtung eingebetteten Logikanalysator erlauben, flexibel interne Signale zu erfassen, und zwar vor und nach einem festgelegten Haltepunkt.

[0015] Zudem wünscht man sich, eine Einrichtung und eine Vorgehensweise zu besitzen, die wirksam und flexibel einen Logikanalysator kontrolliert, der in eine programmierbare Logikvorrichtung eingebettet ist. Im weiteren wird erklärt, dass zwar verschiedene Optionen zum Kontrollieren eines solchen eingebetteten Logikanalysators verfügbar sind, jedoch keine der herkömmlichen Vorgehensweisen optimal ist. Um etwas Hintergrundinformation zu liefern werden zunächst der Entwurf und die Herstellungsphasen eines PLD und einer Platine kurz erklärt.

[0016] Wie in diesem Abschnitt bereits beschrieben entwirft und programmiert ein Entwicklungsingenieur ein PLD mit Hilfe automatischer Werkzeuge für den elektronischen Entwurf. Im Zug dieser Entwurfsphase kann der Entwicklungsingenieur zahlreiche Fehlersuchiterationen im Entwurfsprogramm vornehmen, bevor der Entwurf abgeschlossen ist und das PLD für die Massenfertigung bereit ist. Der Entwicklungsingenieur verwendet häufig eine Simulation und/oder Timing-Untersuchung zur Unterstützung bei der Fehlersuche im elektronischen Entwurf des PLD. Es ist auch denkbar, dass der Entwicklungsingenieur einen eingebetteten Logikanalysator verwendet (siehe die Offenbarung in dem US-Patent Nr. 08/958,435), um die Fehler im Entwurf zu beseitigen. Ist der Entwurf des PLD beendet und stellt er den Entwicklungsingenieur zufrieden, so wird der Entwurf für die Herstellungsphase an einen Fertigungsingenieur übergeben.

[0017] In der Herstellungsphase entwirft ein Fertigungsingenieur einen Herstellungsablauf für die Massenproduktion einer elektronischen Platine oder einer elektronischen Vorrichtung, die ein oder mehrere PLDs enthält. Während der Herstellungsphase ist es erforderlich, die Platine selbst zu prüfen, und es kann auch erforderlich sein, das PLD erneut zu testen. Zu Beginn der Herstellungsphase werden eine beliebige Anzahl und beliebige Typen von Hardwarekomponenten und eine beliebige Anzahl von PLDs auf eine Platine gelötet. Befindet sich das PLD auf der Platine, so wird es in den meisten Fällen mit Hilfe eines JTAG-Ports, der sich auf dem PLD befindet, programmiert (oder konfiguriert). Es ist auch möglich, dass ein bestimmtes PLD für sich programmiert wird, bevor es auf der Platine angeordnet wird, und zwar über einen besonderen Sockel und eine Programmiereinheit.

[0018] Nun kann eine Prüfung der bestückten Platine vorgenommen werden, um die Leiterbahnen, Lötverbindungen und weitere mechanische Verbindungen zwischen den Komponenten auf der Platine zu untersuchen. Es sei darauf hingewiesen, dass eine Platinenprüfung auch erfolgen kann, bevor irgendwelche Bauteile auf der Platine programmiert oder konfiguriert werden. Es ist üblich, einen JTAG-Port eines PLD oder eine andere Vorrichtung zu verwenden, um die Leiterbahnen und Lötverbindungen einer Platine während der Platinenprüfung zu testen. Nach dem Prüfen der mechanischen Verbindungen wird eine vollständige Funktionsprüfung der Platine vor-

genommen, damit die gesamte Funktionalität der Platine getestet wird (d.h., um sicherzustellen, dass bestimmte Eingaben die erwarteten Ausgangssignale erzeugen). Wird an dieser Stelle ein Versagen festgestellt, so kann es nötig sein, ein bestimmtes PLD im eingebauten Zustand auf der Platine nach Fehlern zu untersuchen. Für schwieriger zurückzuverfolgende Fehler kann es auch erforderlich sein, ein PLD, in dem Fehler gesucht werden, von der Platine abzunehmen. Unter diesen Umständen wünscht man, wie bereits erklärt, über einen eingebetteten Logikanalysator in dem PLD zu verfügen, damit man die Fehler leichter suchen kann. Während eines jeglichen Fehlersuchvorgangs im PLD ist es in irgendeiner Weise erforderlich, den eingebetteten Logikanalysator zu steuern, d.h. ihn mit Befehlen und Daten zu versorgen und die erfassten Daten und den Status vom Logikanalysator zu empfangen. Obwohl verschiedene Optionen verfügbar sind, ist derzeit keine völlig zufriedenstellend.

[0019] Man kann beispielsweise vorhandene Ein/Ausgabe-Anschlussstifte einer Vorrichtung dazu verwenden, eine Steuerungsschnittstelle bereitzustellen. Dummerweise kann es sein, dass bei einem bestimmten Entwurf nicht genügend zusätzliche Ein/Ausgabe-Anschlussstifte verfügbar sind, mit denen man eine Schnittstelle herstellen kann, die dem Steuern eines eingebetteten Logikanalysators dient. Es kann erwünscht sein, dass man von einem Kunden, der ein PLD kauft, fordert, dass er eine bestimmte Anzahl von Ein/Ausgabe-Anschlussstiften nicht verwendet, und zwar aus dem einfachen Grund, dass das PLD vielleicht nicht korrekt entworfen ist und an einer gewissen Stelle auf Fehler geprüft werden muss.

[0020] Die Firma Intel in Santa Clara, Kalifornien, verwendet einen JTAG-Port zum Steuern des Zugriffs auf bestimmte Fehlersuchregister, die den Fehlersuchvorgang in einer Zentraleinheit (CPU) unterstützen. Da eine CPU einen bekannten Entwurf hat, weiß man vorher genau, wie viele Fehlersuchregister man benötigt, und die Steuerung wird einfacher. Bei einem PLD ist jeder vom Benutzer implementierte Entwurf ein Individuum. Man weiß im voraus nicht, wie der Entwurf aussieht und wieviele Fehlersuchregister man wohl benötigen wird. Unterschiedliche Entwürfe erfordern unterschiedliche Fehlersuchregister. Damit ist die einfache von Intel verwendete Vorgehensweise mit dem bekannten Entwurf einer CPU nicht auf ein PLD übertragbar.

[0021] DE 4042262 offenbart ein Verfahren zum Analysieren der Funktion digitaler Schaltkreise mit Hilfe abgetasteter Werte, bei dem die Funktionen der digitalen Schaltungen dadurch untersucht werden, dass man Werte von einigen Datenautobahnen zu definierten Zeitpunkten in einen Speicher schreibt und sie auf einem Bildschirm darstellt. Man kann gesteuert durch eine Triggereinheit so viele Ergebnisse wie nötig in den Speicher schreiben, und zwar mit Vorlaufaufzeichnung oder Vorlaufaufzeichnung und Aufzeichnung ab dem Triggerpunkt. Man kann jede beliebige Länge der Vorlaufaufzeichnung und/oder Aufzeichnung ab dem Triggerpunkt verwenden. Die Ergebnisse können über eine digitale Verzögerungsschaltung an den Speicher übertragen werden.

[0022] US-4,873,459 offenbart eine vom Benutzer programmierbare Verbindungsarchitektur, die für Logikanordnungen beim digitalen und analogen Systementwurf eingesetzt werden kann. Zahlreiche Logikzellen oder Module in einer Matrix werden über vertikale und horizontale Verdrahtungskanäle angeschlossen. Die Verdrahtungskanäle kann der Benutzer so programmieren, dass sie die verschiedenen Logikzellen verbinden und dadurch die geforderte logische Funktion implementieren. Erfassungsschaltungen und Verdrahtungen können enthalten sein, damit eine hundertprozentige Beobachtbarkeit der inneren Schaltungsknoten, beispielsweise Modulausgaben, von einer externen Anschlussfleck-Schnittstelle gegeben ist.

[0023] US-5,425,036 offenbart ein Automatisierungssystem für den elektronischen Entwurf (EDA), in dem feldprogrammierbare Gatearrays (FPGA) zum Emulieren von Prototyp-Schaltungsentwürfen verwendet werden. Eine Schaltungs-Netzlistendatei wird in die FPGAs geladen, damit die FPGAs so konfiguriert werden, dass sie die Funktionsdarstellung der Prototypschaltung emulieren. Um zu prüfen, ob die Schaltungs-Netzliste korrekt implementiert ist, wird die Funktion der FPGAs dadurch geprüft, dass man Eingabevektoren daran anlegt und die entstehenden Ausgangssignale der FPGAs mit Ausgabevektoren vergleicht, die aus früheren Simulationen stammen. Fallen die FPGAs bei einem derartigen Vektorvergleich durch, so werden Fehler in den FPGAs dadurch gesucht, dass "Read-Back-Triggerbefehle" in die Eingabevektoren eingefügt werden, die bevorzugt den Fehlerpunkten im verwendeten Vektorstrom entsprechen. Das Modifizieren der Eingabevektoren mit solchen Read-Back-Signalen bewirkt, dass die internen Zustände der Latches und Flipflops in jedem FPGA erfasst werden, wenn die Funktionsprüfung wiederholt wird. Diese interne Statusinformation ist für die Fehlersuche in den FPGAs nützlich und besonders bequem, da man die Schaltungs-Netzliste nicht neu zu übersetzen braucht. Ein ähnlicher Ansatz, bei dem ebenfalls die Read-Back-Merkmale der FPGAs verwendet werden, dient dazu, Fehler in FPGAs zu suchen, die mit einem Zielsystem verbunden sind, das während der Emulationsläufe zu versagen scheint. Dieses Dokument offenbart auch eine Fehlersuchlogik, die einen Logikanalysator und einen JTAG-Port enthält, der zum Herunterladen von Daten auf den Logikanalysator dient.

[0024] Daher wünscht man weiterhin eine Einrichtung und eine Vorgehensweise, die eine einfache, wirksame und flexible Steuerung eines eingebetteten Logikanalysators liefert. Man wünscht zudem, dass eine derartige Steuereinrichtung und Vorgehensweise das Prüfen eines PLD auf einer Platine in einer realen Umgebung zulässt.

ZUSAMMENFASSUNG DER ERFINDUNG

[0025] Um die genannten Ziele zu erreichen und gemäß dem Zweck der Erfindung wird eine Vorgehensweise zum Einbetten eines Logikanalysators in eine programmierbare Logikvorrichtung offenbart, die das Steuern des eingebetteten Logikanalysators über einen JTAG-Port erlaubt. Diese Aufgaben werden im Grunde durch das Verwenden der Merkmale erzielt, die in den unabhängigen Ansprüchen 1 und 9 dargelegt sind. Weitere Verbesserungen sind in den abhängigen Ansprüchen angegeben.

[0026] Häufig wird ein JTAG-Port sowohl zum Programmieren eines PLD als auch zum Unterstützen des Prüfvorgangs einer Platine benutzt, auf der sich PLDs befinden. Vorteilhafterweise hat man erkannt, dass JTAG-Ports bisher während des Entwurfs eines bestimmten PLD und bei der Fehlersuche nicht verwendet wurde. Dadurch hat man ferner erkannt, dass JTAG-Ports auf einem PLD nicht vollständig ausgelastet sind und dass man sie während der Fehlersuche in einem PLD als Mittel zur Kommunikation mit einem eingebetteten Logikanalysator der Erfindung und zum Steuern des Analysators einsetzen kann. Der Standard-JTAG-Port wird vorteilhaft dazu verwendet, die Fehlersuche in einer programmierbaren Logikvorrichtung zu vereinfachen, die einen eingebetteten Logikanalysator enthält. Durch den Einsatz eines JTAG-Ports ist es nicht mehr erforderlich, besondere Steueranschlussstifte für die Fehlersuche zuzufügen. In einer Ausführungsform zum Steuern des eingebetteten Logikanalysators über einen JTAG-Port werden Eingänge und Ausgänge des Logikanalysators mit nicht verbundenen und JTAG-freigegebenen Ein/Ausgabe-Zellen verbunden. Zellen, die Steuersignale liefern, wird vorgespiegelt, dass sie im INTEST-Modus arbeiten, so dass man Signale eingeben kann. Der restliche Teil der Vorrichtung arbeitet aber unter realen Bedingungen. In einer anderen Ausführungsform liefert ein vom Benutzer implementiertes Testdatenregister eine JTAG-artige Kette logischer Elemente, über die Steuer- und Ausgangsinformation geschoben wird. Anregungszellen liefern Steuerinformation an den Logikanalysator, und Erfassungszellen holen Daten vom Logikanalysator.

[0027] Die Erfindung bietet gegenüber dem Stand der Technik zahlreiche Vorteile. Der Gebrauch eines eingebetteten Logikanalysators in einem PLD erlaubt die Fehlersuche in der Vorrichtung in dem System, in dem sie arbeitet, und unter den tatsächlichen Bedingungen, die zu einer Fehlfunktion des PLD führen können. Die Vorgehensweise der Erfindung bettet einen Logikanalysator automatisch in ein PLD ein. Dadurch kann ein Ingenieur jede Logikfunktion in der Vorrichtung überprüfen. Der eingebettete Logikanalysator kann jedes beliebige interne Signal erfassen, das der Ingenieur festlegt. Ein Haltepunkt kann also beliebig festgelegte interne Signale enthalten. Durch den Gebrauch von Speicher innerhalb des eingebetteten Logikanalysators und einer Schnittstelle zum Computer kann man jede beliebige Anzahl und Tiefe von Signalen innerhalb der Vorrichtung überwachen und für eine spätere Analyse an den Computer übertragen. In einer Ausführungsform der Erfindung wird ein JTAG-Port dazu verwendet, den eingebetteten Logikanalysator zu programmieren und die erfasste Signalinformation an den Computer zu übertragen.

[0028] Im Verlauf der Fehlersuche in einem PLD-Entwurf in einem System kann der Ingenieur vorteilhaft die EDA-Werkzeuge zum Festlegen neuer zu überwachender Signale und/oder neuer Haltepunkte verwenden. Der Ingenieur kann daraufhin die Vorrichtung mit einer abgewandelten Logikanalysatorschaltung sehr rasch umprogrammieren, während sie sich im dafür vorgesehenen System befindet, um Fehler in einem anderen Abschnitt der Vorrichtung zu suchen oder die Triggerbedingungen zu verändern. Diese Fähigkeit, einen eingebetteten Logikanalysator ad hoc umzuprogrammieren, weist zahlreiche Vorteile gegenüber eingebauter Fehlersuch-Hardware auf kundenspezifischen Chips auf, die man nicht dynamisch umprogrammieren kann. Diese Umprogrammierfähigkeit weist auch Vorteile gegenüber externen Logikanalysatoren auf, die nur die äußeren Anschlussstifte der Hardwarevorrichtung überwachen können. Hat ein Ingenieur die Fehlersuche in der Vorrichtung mit dem eingebetteten Logikanalysator beendet, so kann er die EDA-Werkzeuge dazu verwenden, eine Ausgabedatei mit der endgültigen Anordnung ohne den Logikanalysator zu erzeugen, die den endgültigen Arbeitsentwurf des Ingenieurs darstellt. Damit braucht der Logikanalysator im endgültigen Entwurf nicht enthalten sein, und er verbraucht keinen Platz auf dem PLD.

[0029] Die Erfindung ist auf einen breiten Bereich von Hardwarevorrichtungen anwendbar, und insbesondere auf PLDs. Ein PLD kann im Einzelnen mit sehr unterschiedlichen Technologien implementiert werden, einschließlich der SRAM-Technologie und der EEPROM-Technologie. Auf der SRAM-Technologie beruhende PLDs sind besonders vorteilhaft, weil sie zusätzlichen eingebetteten Speicher umfassen können, den der eingebettete Logikanalysator dazu verwenden kann, eine große Anzahl von Signalen und eine größere Signaltiefe zu erfassen. Zudem bedeutet ein eingebetteter Logikanalysator, der mit einem EDA-Werkzeug automatisch entworfen und eingefügt wird, dass der Ingenieur keinen externen Logi-

kanalysator als eigenes Ausrüstungsteil benötigt. Ferner kann der Ingenieur den Computer, auf dem er einen Entwurf für das PLD anfertigt, auch dazu verwenden, den eingebetteten Logikanalysator zu steuern und zu konfigurieren und dessen Ergebnisse zu sichten.

[0030] In einer Ausführungsform der Erfindung sind eine Anzahl Anschlussstifte auf dem PLD als Schnittstellenstifte für die Kommunikation mit dem Benutzercomputer zugewiesen. Da die Anschlussstifte der Schnittstelle zugewiesen und vorab bekannt sind, kann man sie zu einer leicht zugänglichen Stelle oder einem Port auf einer Platine führen, und dadurch sehr einfach Fehlersuch-Schnittstellenkabel vom Benutzercomputer mit diesen Anschlussstiften verbinden. Diese Vorgehensweise ist dann besonders vorteilhaft, wenn Anschlussstifte oder Kontakte einer bestimmten integrierten Schaltung in einem Gehäuse nur schwierig oder nahezu nicht erreichbar sind. Da der eingebettete Logikanalysator der Erfindung so konfiguriert werden kann, dass er jedes beliebige interne oder externe Signal des PLD überwacht, sind alle diese überwachten Signale für die Analyse über diese Schnittstellenstifte verfügbar. Anders formuliert ist es nicht nötig, einen Fühler mechanisch mit einem bestimmten externen Anschlussstift von Interesse zu verbinden, da man jedes Signal innerhalb des PLD über diese zugewiesenen Schnittstellenstifte überwachen kann, es im Speicher des eingebetteten Logikanalysators ablegen kann und es danach zur Untersuchung auf den Benutzercomputer hochladen kann.

[0031] Zusätzlich kann man einen eingebetteten Logikanalysator mit PLDs verwenden, die bis nahe an die Kapazitätsgrenze ausgelastet sind. Ein Ingenieur kann vorübergehend einen Teil des Entwurfs entfernen, der mit dem untersuchten Problem nichts zu tun hat, eine Logikanalysatorschaltung einbetten und daraufhin die Fehler im PLD suchen. Nach der Fehlersuche im PLD kann der Ingenieur den eingebetteten Logikanalysator entfernen und den Abschnitt des Entwurfs wieder einsetzen, den er vorübergehend entfernt hat.

KURZE BESCHREIBUNG DER ZEICHNUNGEN

[0032] Man versteht die Erfindung und ihre weiteren Vorteile anhand der folgenden Beschreibung in Verbindung mit den beiliegenden Zeichnungen am besten.

[0033] Es zeigt:

[0034] Fig. 1 ein Blockdiagramm eines programmierbaren Logikentwicklungssystems gemäß einer Ausführungsform der Erfindung;

[0035] Fig. 2 ein Flussdiagramm eines Entwurfsverfahrens, das zum Entwerten einer programmierbaren Logikvorrichtung gemäß einer Ausführungsform der Erfindung verwendet wird;

[0036] Fig. 3A und Fig. 3B ein Flussdiagramm, das eine Vorgehensweise beschreibt, mit der ein Logikanalysator programmiert und in eine Vorrichtung eingebettet wird, und das beschreibt, wie der Logikanalysator Daten erfasst und sie für die Sichtung durch einen Benutzer ausgibt;

[0037] Fig. 4 ein Flussdiagramm, das eine Vorgehensweise beschreibt, mit der man einen Logikanalysator zusammen mit einem Benutzerentwurf übersetzt, um den Logikanalysator in eine Hardwarevorrichtung einzubetten;

[0038] Fig. 5 eine andere Darstellung des Blockdiagramms in Fig. 1, die eine programmierbare Logikvorrichtung zeigt, die einen eingebetteten Logikanalysator innerhalb eines elektronischen Systems aufweist;

[0039] Fig. 6 ein ausführlicheres Blockdiagramm einer programmierbaren Logikvorrichtung, die einen eingebetteten Logikanalysator aufweist;

[0040] Fig. 7 einen eingebetteten Logikanalysator, wobei seine Eingänge und Ausgänge gemäß einer Ausführungsform der Erfindung dargestellt sind;

[0041] Fig. 8 ein Blockdiagramm einer eingebetteten Logikanalysatorschaltung gemäß einer Ausführungsform der Erfindung;

[0042] Fig. 9 eine symbolische Darstellung der Arbeitsweise der Zustandssteuer-Maschine des eingebetteten Logikanalysators;

[0043] Fig. 10 eine Tabelle, die die Zustände und entsprechenden Zustandsausgaben der Zustandssteuer-Maschine gemäß einer Ausführungsform angibt;

[0044] Fig. 11 eine erste Ausführungsform, über die ein JTAG-Port einen eingebetteten Logikanalysator mit Hilfe unverbundener Ein/Ausgabe-Zellen steuert;

[0045] Fig. 12 eine herkömmliche für JTAG freigegebene Ein/Ausgabe-Zelle;

[0046] Fig. 13 eine für JTAG freigegebene Ein/Ausgabe-Zelle gemäß der ersten Ausführungsform in Fig. 11;

[0047] Fig. 14 eine zweite Ausführungsform, über die ein JTAG-Port einen eingebetteten Logikanalysator mit Hilfe eines Testdatenregisters steuert;

[0048] Fig. 15 eine Anregungszelle, die ein Element

des Testdatenregisters in Fig. 14 darstellt;

[0049] Fig. 16 eine Erfassungszelle, die ein Element des Testdatenregisters in Fig. 14 darstellt;

[0050] Fig. 17A und Fig. 17B eine andere Ausführungsform, in der eine beliebige Anzahl von Logikanalysatoren, die in eine Vorrichtung eingebettet sind, mit Hilfe eines JTAG-Ports gesteuert werden; und

[0051] Fig. 18 ein Blockdiagramm eines üblichen Computersystems, das sich zum Implementieren einer Ausführungsform der Erfindung eignet.

AUSFÜHRLICHE BESCHREIBUNG DER ERFINDUNG

[0052] Zum Entwickeln eines Entwurfs für die Programmierung eines elektronischen Entwurfs, beispielsweise einer programmierbaren Logikvorrichtung (PLD), wird ein programmierbares Logikentwicklungssystem verwendet. So wie er hier verwendet wird, bezieht sich der Begriff "elektronischer Entwurf" auf Platinen und Systeme, die zahlreiche elektronische Vorrichtungen und Multi-Chip-Module sowie integrierte Schaltungen enthalten. Zum Vereinfachen der Darstellung ist in der folgenden Besprechung allgemein von "integrierten Schaltungen" oder "PLDs" die Rede, obwohl die Erfindung nicht hierauf eingeschränkt ist.

PROGRAMMIERBARES LOGIKENTWICKLUNGSSYSTEM

[0053] Fig. 1 zeigt ein Blockdiagramm einer Ausführungsform eines programmierbaren Logikentwicklungssystems **10**, das ein Computernetz **12**, eine Programmiereinheit **14** und eine programmierbare Logikvorrichtung **16** enthält, die programmiert werden soll. Das Computernetz **12** enthält eine beliebige Anzahl von Computern, die in einem Netz verbunden sind, beispielsweise ein Computersystem A **18**, ein Computersystem B **20**, ein Computersystem C **22** und einen Computersystem-Dateiserver **23**, die alle über eine Netzverbindung **24** miteinander gekoppelt sind. Das Computernetz **12** ist über ein Kabel **26** mit der Programmiereinheit **14** verbunden. Diese ist ihrerseits über ein Programmierkabel **28** mit dem PLD **16** verbunden. Wahlweise kann nur ein Computersystem direkt mit der Programmiereinheit **14** verbunden sein. Zudem muss das Computernetz **12** nicht ständig mit der Programmiereinheit **14** verbunden sein, beispielsweise dann, wenn ein Entwurf entwickelt wird, sondern es braucht nur dann angeschlossen werden, wenn das PLD **16** zu programmieren ist.

[0054] Die Programmiereinheit **14** kann jede beliebige geeignete Hardware-Programmiereinheit sein, die zum Programmieren des PLD **16** Programmbefehle vom Computernetz **12** aufnimmt. Beispielsweise kann die Programmiereinheit **14** eine Zusatz-Logikprogrammierkarte für einen Computer enthalten sowie eine Masterprogrammiereinheit, die beispielsweise von der Altera Corporation in San Jose, Kalifornien, erhältlich ist. Das PLD **16** kann in einem System oder in einer Programmierstation vorhanden sein. Bei Betrieb nutzt eine beliebige Anzahl von Ingenieuren das Computernetz **12** für die Entwicklung von Programmierbefehlen, wobei sie ein elektronisches Softwarewerkzeug für die Entwurfsautomatisierung nutzen. Haben die Ingenieure einen Entwurf entwickelt und eingegeben, so wird der Entwurf übersetzt und verifiziert, bevor er in die Programmiereinheit heruntergeladen wird. Die Programmiereinheit **14** kann nun den heruntergeladenen Entwurf dazu verwenden, das PLD **16** zu programmieren.

[0055] Zum Zweck der Fehlersuche in einem PLD gemäß einer Ausführungsform der Erfindung kann irgendeiner der dargestellten Computer oder ein anderer Computer dazu genutzt werden, eine Logikanalysatorschaltung zu spezifizieren und diese Schaltung zusammen mit dem Entwurf eines Benutzers zu übersetzen. Zudem kann man ein Programmierkabel **28** dazu verwenden, den Logikanalysator zu steuern und Daten von ihm zu empfangen, oder man kann ein eigenes Fehlersuchkabel dazu verwenden, einen Computer direkt mit der Vorrichtung **16** zu verbinden.

[0056] Ein derartiges Logikentwicklungssystem dient zum Erzeugen eines elektronischen Entwurfs. Die Entwurfseingabe und Verarbeitung erfolgt im Rahmen eines "Projekts". Ein Projekt umfasst eine Projektdatei, Entwurfsdateien, Zuordnungsdateien und Simulationsdateien zusammen mit Hierarchieinformation, Systemeinstellungen und Ausgabedateien, zu denen Programmierdateien und Berichtsdateien gehören. Es kann auch eine Projektdatenbank vorhanden sein, die zwischenzeitlich anfallende Datenstrukturen und Versionsinformationen enthält.

[0057] Ein Projekt umfasst eine oder mehrere Hierarchien von Entwurfsentitäten. Jede Entwurfshierarchie besitzt eine Ausgangsentität, die die alleroberste Entwurfsentität in diesem Hierarchiebaum darstellt (den allerobersten Funktionsblock). Die anderen Entwurfsentitäten in dem Entwurfshierarchiebaum werden Kinderentitäten genannt. Eine Entwurfshierarchie kann Entitäten enthalten, für die keine entsprechende Entwurfsdatei vorhanden ist, beispielsweise bei einer Top-Down-Verfahrensweise. Der Teil der Hierarchie, der solche noch nicht implementierten Entitäten enthält, wird solange nicht übersetzt oder simuliert, bis eine Entwurfsdatei für jede Entität vorliegt. In diesem Fall werden Quelldatei-Schablonen automatisch erzeugt, die definierte Schnittstellen und leere Körper haben, um das Implementieren dieser Teile eines Projekts zu unterstützen. Ein Benutzer erzeugt einen Entwurf, indem er Funktionsblöcke spezifiziert und implementiert. Dies wird nun im Zusam-

menhang mit einer beispielhaften Entwurfsverfahrensweise beschrieben.

ENTWURFSVERFAHREN

[0058] Fig. 2 zeigt ein Entwurfsverfahren **50**, in dem eine Systementwurfsspezifikation dazu verwendet wird, einen Entwurf zu entwickeln, mit dem ein PLD programmiert wird. Man beachte, dass die Erfindung im Zusammenhang mit sehr vielen unterschiedlichen Entwurfsverfahren angewendet werden kann. Als Beispiel sei genannt, dass die Arbeitsgruppen-Computertechniken und Systeme der Erfindung im Rahmen der Verfahrensweise nach Fig. 2 gut mit einem elektronischen Software-Werkzeug zur Entwurfsautomatisierung (EDA) zusammenarbeiten.

[0059] Im Schritt **52** erhält man eine Systemspezifikation für das zu programmierende PLD. Diese Spezifikation ist ein externes Dokument oder eine externe Datei, die beispielsweise die Namen der Anschlussstifte der Vorrichtung beschreibt, die Funktionen eines jeden Anschlussstifts, die gewünschte Systemfunktionalität, die Verfügbarkeit von Timing und Ressourcen usw. Die zahlreichen Ingenieure in einer Arbeitsgruppe verwenden diese Systemspezifikation dazu, mit den EDA-Werkzeugen einen Entwurf zu erzeugen, der danach zum Programmieren eines PLD verwendet wird.

[0060] Liegt die Systemspezifikation vor, so wird mit dem Erzeugen eines Entwurfs mit Hilfe von Funktionsblockdiagrammen begonnen. Im Schritt **54** wird ein Blockdiagramm auf oberster Ebene erzeugt, in dem die Verbindungen zwischen Entwurfsblöcken auf niedrigeren Ebenen festgelegt werden. In diesem Block können die Zielvorrichtung, die Geschwindigkeitsklasse, und die grundlegenden Timinganforderungen festgelegt werden. Fachleuten ist geläufig, dass dieser Block auf oberster Ebene auch Blöcke enthalten kann, die bereits entwickelt oder implementiert sind oder die man von dritter Seite erhalten hat. Dieser Block auf oberster Ebene kann auch in eine HDL-Datei oder eine ähnliche Datei umgewandelt werden, die in anderen verwandten Entwurfswerkzeugen verwendet wird, etwa in einem externen Simulator.

[0061] Der Schritt **56** umfasst das Erzeugen von Entwurfsdatei-Schablonen mit dem EDA-Werkzeug für alle Blöcke, die im Blockdiagramm auf oberster Ebene im Schritt **54** vorhanden sind. Nachdem der Entwickler einen Block erzeugt hat, der noch nicht implementiert ist, kann das System eine Entwurfsdatei-Schablone erzeugen. Solche Schablonen können einen Block in einem Fensterformat darstellen, der beispielsweise einen Titel, ein Datum usw. entlang der Grenzen enthält. Er kann auch gewisse Einzelheiten des Funktionsinhalts enthalten, der im Fenster dargestellt ist. Der Entwurf der Entwurfsdatei-Schablonen kann in jedem beliebigen spezifizierten Entwurfsformat vorliegen, beispielsweise VHDL, AHDL, Verilog, Blockdiagramm, Skizze oder einem ähnlichen Format. Im Fall eines VHDL-Blocks kann die Schablone einen wesentlichen Teil der Formatierung und erforderlichen Syntax für beliebige VHDL-Blöcke enthalten. Der Benutzer braucht die Schablone nur zu nehmen und die geringe Menge VHDL-Syntax einzutragen, die zum Implementieren seiner Funktion erforderlich ist. Beispielsweise braucht der Benutzer lediglich Syntax zufügen, die eine bestimmte UND-Gatter-Funktion bestimmt. Übliche Entwürfe wie VHDL oder andere IEEE-Standards erfordern große Textmengen um den Entwurfsblock adäquat aufzubauen.

[0062] Fachleute sind damit vertraut, dass derartige Entwurfsdatei-Schablonen als Ausgangspunkte für den Entwurf der strukturellen oder funktionalen Einheiten dienen können, die für den Entwurf erforderlich sind. Somit kann eine Entwurfsdatei-Schablone als mehrfach verwendbares Objekt für unterschiedliche Fälle eines Blocks in einem oder mehreren Entwürfen dienen. Wichtiger ist, dass man Entwurfsdatei-Schablonen dafür einsetzt, den Arbeitsaufwand zu reduzieren, den der Entwickler zum Erzeugen der Logik in den Blöcken aufbringen muss. In einer Ausführungsform wird das Erzeugen der Entwurfsdatei-Schablonen so vorgenommen, dass man die Schablonen später aktualisieren kann, falls sich das Blockdiagramm auf oberster Ebene ändert.

[0063] Nun wird im Schritt **58** jeder Block des Blocks auf oberster Ebene mit dem EDA-Werkzeug implementiert. Man beachte, dass bei komplizierteren Entwürfen zusätzliche Blockdiagrammebenen vorhanden sein können (d.h. Blöcke innerhalb von Blöcken). Sind Veränderungen auf oberster Ebene erforderlich, so wird das Blockdiagramm auf oberster Ebene aktualisiert, und die darunter liegenden Entwürfe werden ebenfalls – bevorzugt automatisch – aktualisiert.

[0064] Zudem kann ein Block in eine Anpassstufe für eine bestimmte integrierte Schaltungsform übersetzt werden, damit man Information über die Ressourcennutzung, das Zeitverhalten usw. erlangt, die für einen gegebenen Entwurf nötig sind. Damit wird auch in Betracht gezogen, dass während des Schritts **58** eine gewisse Optimierung des Zeitverhaltens erfolgen kann. Diese Abfolge erläutert einen Entwurfsstil, bei dem ein Ingenieur zuerst entwirft, dann übersetzt und simuliert und anschließend zum Entwurf zurückkehrt, falls die Simulationsergebnisse nicht zufriedenstellend sind. Bei einem anderen Entwurfsstil kann ein Ingenieur durch eine Anzahl von Entwürfen iterieren, worauf Simulationsschleifen folgen, bevor schließlich der vollständige Entwurf übersetzt wird.

[0065] Hinsichtlich der Reihenfolge der Blockimplementierung kann man einen oder mehrere der folgen-

den Faktoren zum Bestimmen der Implementierungsreihenfolge verwenden: (1) die Komplexität des Blocks; (2) die Ungewissheit bzw. das Risiko, die bzw. das mit dem Block verbunden ist; und/oder (3) wie weit oben oder unten in einem gegebenen Datenpfad der Block angesiedelt ist. Jeder der Schritte **60**, **62**, **64**, **68** und **70** kann auch zu diesem Blockimplementierungsschritt zurückführen, wenn durch spätere Änderungen im Entwurf zusätzliche Implementierungen nötig werden.

[0066] Im Schritt **60** wird ein Block funktional auf der Quellebene simuliert, wobei ein Verhaltenssimulator verwendet wird sowie Vektoren, die beispielsweise mit Hilfe einer VHDL- oder Verilog-Prüfbank erzeugt werden. Die Simulationsergebnisse können nun angezeigt oder anderweitig dargestellt bzw. aufgezeichnet werden, und zwar als Kurven, als Text oder den Quelldateien als Kommentar beigegeben werden. Der Entwickler kann auch zum Schritt **58** zurückkehren und einen Block neu implementieren. An dieser Stelle kann ein Block auch übersetzt werden oder man kann das Zeitverhalten untersuchen.

[0067] Ist der Entwickler mit den Simulationsergebnissen zufrieden, so wird der Block im Schritt **62** mit den anderen Blöcken verbunden, und die entstehende Gruppe wird gemeinsam simuliert. In manchen Fällen kann es nützlich sein, eine vollständige Übersetzung abzuschließen, damit man kritische Ressourcen- und Timinginformation erhält. Zudem können die ausgegebenen Simulationsvektoren eines Blocks zu den Eingabe-Simulationsvektoren des folgenden Blocks werden. Der Entwickler kann auch zum Schritt **54** zurückkehren und den Block auf oberster Ebene modifizieren oder zum Schritt **58** und den Block erneut implementieren.

[0068] Nun wird im Schritt **64** der Gesamtentwurf mit Hilfe eines Verhaltenssimulators auf Quellebene funktional simuliert. Bevorzugt wird das Blockdiagramm auf oberster Ebene vor der Simulation vollständig spezifiziert, und es zeigt die vollständigen Entwurfsverbindungen. Vektoren kann man mit Hilfe einer VHDL- oder Verilog-Prüfbank erzeugen. Wiederum kann man die Simulationsergebnisse entweder als Kurven darstellen oder den Quelldateien als Kommentar anhängen. Der Entwickler kann auch zum Schritt **54** zurückkehren und den Block auf oberster Ebene modifizieren oder zum Schritt **58** und einen Block neu implementieren. Im Schritt **66** wird der gesamte Entwurf in eine Datei übersetzt, die die Information enthält, die zum Programmieren eines PLD nötig ist, das den Entwurf des Benutzers implementiert, beispielsweise in eine "Programmier-Ausgabedatei".

[0069] Abhängig davon welche Entwurfsart erzeugt wird kann man eine breite Vielfalt von Übersetzungstechniken verwenden. Als Beispiele werden im Weiteren einige Möglichkeiten der Übersetzung angegeben. Für ein PLD umfasst die Übersetzung die Schritte der Synthese, der Platzierung und Verbindung, des Erzeugens von Programmierdateien und der Simulation. Für einen herkömmlichen integrierten Schaltungsentwurf mit einem gebräuchlichen Layout umfasst das Übersetzen einen Layout-Versionsplan, eine Entwurfsregelprüfung und Simulationen. Für den Entwurf integrierter Schaltungen mit einem hochentwickelten Entwurfswerkzeug umfasst das Übersetzen die Synthese aus einer Sprache wie VHDL oder Verilog, das automatische Platzieren und Verbinden und Simulationen. Für gedruckte Platinen umfasst das Übersetzen das automatische Verbinden, die Entwurtsregelprüfung, die Entnahme verteilter Parameter und die Simulation. Natürlich sind andere Übersetzungsarten und Variationen der obigen Vorgehensweisen möglich.

[0070] Im Zusammenhang der Erfindung kann jede der genannten Übersetzungstechniken abgewandelt werden, um einen eingebetteten Logikanalysator zu erzeugen. Anhand von Fig. 4 wird im Weiteren ausführlicher besprochen, wie das Übersetzen eines PLD modifiziert wird, um einen Logikanalysator in einen Benutzerentwurf einzufügen.

[0071] Nach dem Übersetzen im Schritt **66** wird im Schritt **68** der Timingprüfer innerhalb des Übersetzers dazu verwendet, festzustellen, ob die Verhaltensziele für den Entwurf erreicht werden. Timingsimulationen werden auch dazu verwendet, Einzelheiten des Verhaltens zu prüfen. Zusätzlich können weitere Analysewerkzeuge wie Entwurtsprofiler und/oder Layouteditor dazu verwendet werden, das Verhalten des Entwurfs weiter zu optimieren. Bevorzugt erfolgt die Optimierung nicht vor dem Schritt **68**, da man in der Regel eine vollständige Übersetzung benötigt, um den Ort eines oder mehrerer kritischer Pfade im Entwurf festzustellen. Der Entwickler kann auch zum Schritt **54** zurückkehren und den Block auf oberster Ebene verändern oder zum Schritt **58** um einen Block neu zu implementieren.

[0072] Nun wird im Schritt **70** die Vorrichtung mit Hilfe der Programmiereinheit **14** programmiert bzw. konfiguriert und im System geprüft. Wiederum kann der Entwickler auch zum Schritt **54** zurückkehren und den Block auf oberster Ebene verändern oder zum Schritt **58** um einen Block neu zu implementieren. Die Verfahrensweise **50** stellt einen Top-Down-Entwurfsablauf vor. Man kann sie aber auch dazu verwenden, eine Bottom-Up-Verfahrensweise zu unterstützen. Da nun die allgemeine Entwurfsverfahrensweise beschrieben ist, mit der ein Ingenieur einen Entwurf für ein PLD entwickeln kann, wird nun ein Vorgehen zum Einbetten eines Logikanalysators in ein PLD beschrieben.

EINGEBETTETER LOGIKANALYSATOR

[0073] Das Flussdiagramm in Fig. 3A und Fig. 3B beschreibt eine mögliche Vorgehensweise, mit der ein Benutzer einen Logikanalysator in ein PLD einbetten kann, um gewünschte Signale zu erfassen und die Ergebnisse auf einem Computer zu sehen. Im Schritt **102** erzeugt ein Benutzer einen Entwurf für eine Vorrichtung und übersetzt den Entwurf in eine Ausgabedatei. Man kann eine breite Vielfalt von EDA-Werkzeugen dazu verwenden, einen Entwurf für ein PLD zu erzeugen und zu übersetzen. Als Beispiel kann die Vorgehensweise verwendet werden, die im US-Patent Nr. 60/029,277 offenbart ist.

[0074] Im Schritt **104** wird die übersetzte Ausgabedatei zum Programmieren der Vorrichtung verwendet. Die Vorrichtung wird unter Einsatzbedingungen betrieben, beispielsweise auf einer gedruckten Platine oder innerhalb eines geeigneten elektronischen Systems. In diesem Schritt ist es möglich, dass ein Benutzer Fehlfunktionen der Vorrichtung feststellt. Trifft dies zu, so wird im Schritt **106** ein Hardware-Fehlersuchmerkmal des EDA-Werkzeugs aktiviert. Diese Aktivierung erlaubt es dem EDA-Werkzeug, die Netzliste zu ergänzen. D.h., der Benutzerentwurf in den verschiedenen Entwurfsdateien wird um einen Logikanalysator gemäß einer Ausführungsform der Erfindung erweitert; dies wird in Fig. 4 ausführlicher beschrieben. Dieses Fehlersuchmerkmal erlaubt es auch, die kombinierte Netzliste zu verarbeiten und in die Vorrichtung zu programmieren.

[0075] Nun kann der Benutzer den Logikanalysator programmieren, um Fehler in der Vorrichtung in jeder beliebigen Weise zu suchen, die der Benutzer auswählt. Ein Beispiel für eine Logikanalysatorschaltung ist in Fig. 8 dargestellt. Der Entwurf des Logikanalysators kann bereits in dem EDA-Werkzeug vorhanden sein oder er kann zu jedem beliebigen Zeitpunkt erzeugt werden. Im Schritt **108** werden die interessierenden Signale der Vorrichtung spezifiziert, die überwacht werden sollen. Es handelt sich dabei um die Signale, die ein Benutzer zu sehen wünscht, um die Ursache der Fehlfunktion aufzuspüren. Bei den Signalen kann es sich um Signale handeln, die an Anschlussstiften der Vorrichtung anliegen oder um beliebige interne Signale oder Punkte der Vorrichtung. Die Art der im Schritt **104** beobachteten Fehlfunktion liefert oft einen Anhaltspunkt bezüglich der Signale, die vermutlich weitere Information über das Problem beibringen. Steht beispielsweise die Fehlfunktion im Zusammenhang mit der Datenausgabe an einem bestimmten Anschlussstift, so können die zu überwachenden Signale von der Logik stammen, die dem Anschlussstift vorgeschaltet ist.

[0076] Diese zu überwachenden Signale können auf viele verschiedene Arten spezifiziert werden. Beispielsweise kann man einen hierarchischen Pfadnamen für jedes Signal festlegen, oder man kann eine graphische Benutzerschnittstelle dazu verwenden, eine bestimmte Entwurfsdatei einzusehen und ein Signal oder einen Punkt innerhalb dieser Datei zu wählen, das bzw. der überwacht werden soll. In einer anderen Ausführungsform kann der Benutzer auch festlegen, welche Anschlussstifte der Vorrichtung als Schnittstelle zum Benutzercomputer verwendet werden sollen, d.h. diejenigen Anschlussstifte, die zum Senden von Steuerinformation an den eingebetteten Logikanalysator im PLD und zum Hochladen von erfasster Information vom Logikanalysator zum Benutzercomputer verwendet werden sollen. Bevorzugt kennt man jedoch die Anschlussstifte bereits, die als Schnittstelle verwendet werden sollen, beispielsweise einen JTAG-Port der Vorrichtung.

[0077] Im Schritt **110** wird die Gesamtanzahl der Abtastwerte festgelegt, die erfasst werden sollen. Anders ausgedrückt wird die Tiefe des Abtastspeichers bestimmt. Dies gibt seinerseits die Anzahl der Taktimpulse an, bei denen der Logikanalysator Daten erfasst. Die Gesamtanzahl der zu erfassenden Abtastwerte enthält alle zu erfassenden Abtastwerte, und zwar vor oder nach einem festgelegten Haltepunkt. Anders formuliert bezeichnet die festgelegte Anzahl die Breite eines Datenfensters, das erfasst werden soll. Das Fenster kann den Haltepunkt umschließen, es kann vollständig vor dem Haltepunkt liegen oder es kann vollständig nach dem Haltepunkt liegen.

[0078] In einer Ausführungsform eignet sich ein PLD, das eingebettete Speicherblöcke enthält (beispielsweise irgendein PLD aus der FLEX 10K-Vorrichtungsfamilie, die von der Altera Corporation erhältlich sind), zum Implementieren der Erfindung. Die eingebetteten Speicherblöcke lassen sich leicht so programmieren, dass sie relativ große Puffer (als Teil der Logikanalysatorschaltung) zum Speichern der erfassten Information bilden. Zum Puffern der erfassten Information sind jedoch keine eingebetteten Speichervorrichtungen erforderlich. Man kann auch Vorrichtungen ohne eingebetteten Speicher verwenden; sie lassen sich jedoch nicht so leicht zum erzeugen relativ großer Puffer verwenden. In Vorrichtungen ohne eingebetteten Speicher kann man Puffer über zahlreiche Zellen implementieren und verfügbaren Speicher aus jeder Zelle verwenden.

[0079] Im Schritt **112** wird ein Systemtaktsignal für den Gebrauch durch den Logikanalysator festgelegt. Man kann jedes Signal von zahlreichen verschiedenen in der Vorrichtung verfügbaren Signalen als Systemtaktsignal definieren. Normalerweise wählt man ein Vorrichtungstaktsignal, das sich auf die überwachten Signale bezieht, als Systemtaktsignal. Man kann auch ein Vielfaches eines Vorrichtungstaktsignals wählen, damit die Signale häufiger abgetastet werden.

[0080] Im Schritt **114** wird ein Haltepunkt festgelegt. Ein Haltepunkt kann eine beliebige Anzahl zu überwachender Signale enthalten sowie die Logikpegel, die diese Signale haben müssen, um den Logikanalysator auszulösen. D.h., der Haltepunkt beschreibt einen bestimmten Status der Vorrichtung. Ein Haltepunkt kann ganz einfach die Statusänderung eines Signals sein. Es kann auch ein kompliziertes Muster aus Signalen sein oder eine Folge von Mustern, die auftritt, bevor der Logikanalysator ausgelöst wird. Ein Haltepunkt muss nicht in jedem Fall festgelegt werden. Geschieht dies nicht, so beginnt der Logikanalysator unmittelbar nach dem Anlauf mit der Datenerfassung. Der Benutzer kann den Haltepunkt vorteilhafterweise zu jedem beliebigen Zeitpunkt mit Hilfe des EDA-Werkzeugs ändern. Man kann einen neuen Haltepunkt auf den eingebetteten Logikanalysator in der Vorrichtung herunterladen, ohne dass man alle Entwurfsdateien der Vorrichtung neu übersetzen muss. Da es möglich ist, Haltepunkte für eine Vorrichtung in einem System sehr rasch zu verändern, ist die Fehlersuche sehr viel effizienter. Vorteilhaft ist, dass man die Daten nicht nur nach dem Haltepunkt sondern auch vor dem Auftreten des Haltepunkts erfassen kann.

[0081] Im Schritt **115** legt der Benutzer die Anzahl der Datenabtastungen fest, die vor dem Haltepunkt zu erfassen sind. Vorteilhaft ist, dass der Benutzer jede beliebige Anzahl an Abtastungen festlegen kann, die vor dem Auftreten des Haltepunkts zu erfassen ist. Damit kann man diese vorhergehenden Signale später untersuchen und damit das Erkennen der Ursache des Versagens, des Fehlers oder einer Bedingung unterstützen. Wie im Folgenden anhand von Fig. 8 bis Fig. 10 erklärt wird, erlaubt es die Implementierung des eingebetteten Logikanalysators, Abtastwerte fortlaufend zu speichern. Dies liefert einem Benutzer jede beliebige Anzahl von Abtastwerten, die er vor dem Haltepunkt benötigt. Wahlweise kann ein Benutzer die Anzahl der Abtastwerte festlegen, die er nach dem Haltepunkt benötigt. Da die Gesamtanzahl der zu erfassenden Abtastwerte im Schritt **110** spezifiziert wird, ist es sehr einfach, die benötigten vorhergehenden Abtastwerte aus den benötigten nachfolgenden Abtastwerten zu berechnen oder umgekehrt. Der Benutzer kann die vor dem Haltepunkt benötigten Abtastwerte und die nach dem Haltepunkt benötigten Abtastwerte angeben. Die Gesamtanzahl der zu erfassenden Abtastwerte kann dann automatisch berechnet werden.

[0082] Zusätzlich zur Möglichkeit, die Anzahl der zu erfassenden Abtastwerte im Schritt **110** festzulegen sowie die Anzahl der vor dem Haltepunkt benötigten Abtastwerte im Schritt **115**, können diese Werte auch spezifiziert werden, nachdem der Benutzerentwurf und der Logikanalysator übersetzt sind. Anders ausgedrückt können in den Schritten **110** und **115** gewisse Werte vor dem Übersetzen des Entwurfs bestimmt werden. Diese Werte können aber auch in den Logikanalysator eingegeben werden, wenn das PLD bereits programmiert ist oder selbst dann, wenn der Logikanalysator bereits läuft. Beispielsweise kann das Register **310** in Fig. 8 jederzeit auf den Wert Delay[6:0] gesetzt werden, um die Anzahl der Abtastwerte anzuzeigen, die nach dem Auftreten des Haltepunkts zu erfassen sind. Die Gesamtanzahl der zu erfassenden Abtastwerte ist gleich der Anzahl der Wörter im Abtastspeicher **324**. Weitere vor dem Übersetzen spezifizierte Werte können in ähnlicher Weise nach dem Übersetzen festgelegt werden, beispielsweise das Triggerregister **304**.

[0083] Hat der Benutzer festgelegt, wie der eingebettete Logikanalysator funktionieren soll, so wird der vollständige Entwurf übersetzt. Im Schritt **116** gibt der Benutzer einen Übersetzungsbefehl, damit der Vorrichtungsentwurf des Benutzers zusammen mit dem festgelegten Logikanalysatorentwurf übersetzt wird. In einer bevorzugten Ausführungsform der Erfindung werden die Entwurfsdateien während dieses Vorgangs nicht verändert. Der Logikanalysatorentwurf wird in die erzeugten Ausgabedateien eingebaut. In einer bestimmten Ausführungsform kann der in Fig. 4 dargestellte Ablauf zum Implementieren des Schritts **116** verwendet werden. Natürlich lassen sich weitere ähnliche Vorgehensweisen einsetzen.

[0084] Das Ergebnis dieses Schritts ist eine neue Ausgabedatei, die den Benutzerentwurf mit einem eingebetteten Logikanalysator enthält. Eine Technik, mit der ein EDA-Werkzeug einen maßgeschneiderten Logikanalysator in einen Entwurf einfügt, wird im Weiteren anhand von Fig. 4 ausführlicher erklärt. Nach dem Erzeugen der neuen Ausgabedatei wird die Vorrichtung im Schritt **118** innerhalb ihres Systems neu programmiert, wobei die neue Ausgabedatei verwendet wird.

[0085] Im Schritt **120** verbindet der Benutzer die Vorrichtung und den Benutzercomputer mit einem Fehlersuch-Schnittstellenkabel. Das Schnittstellenkabel kann das gleiche Kabel sein, das zum Programmieren der Vorrichtung verwendet wird, oder es kann ein extra der Fehlersuche zugeordnetes Kabel sein. In einer Ausführungsform der Erfindung ist das Fehlersuchkabel mit Anschlussstiften verbunden, die der Benutzer im Schritt **108** festgelegt hat und die der Logikanalysatorschaltung zugewiesen sind. Hat der Benutzer – anders formuliert – die Anschlussstifte definiert, an die das Fehlersuchkabel anzuschließen ist, so sollte das Kabel auch an diese Anschlussstifte angeschlossen werden. In einer anderen Ausführungsform braucht der Benutzer keine "Fehlersuchstifte" zu spezifizieren, sondern das System legt sie automatisch fest. In einer weiteren Ausführungsform kann ein zugeordneter JTAG-Port der Vorrichtung Verwendung finden.

[0086] Das Kabel kann direkt an diese Anschlussstifte angeschlossen werden, oder die Signale von diesen Anschlussstiften können an einen leicht zugänglichen Ort oder Port auf der Platine geführt werden, an dem man das Fehlersuchkabel leicht anschließen kann. Man kann das Kabel dazu verwenden, Befehle vom Computer an den eingebetteten Logikanalysator zu übertragen sowie zum Hochladen der erfassten Information vom Logikanalysator zum Computer. Im Weiteren wird erklärt, dass Fig. 5 ein PLD zeigt, das sowohl einen Benutzerentwurf als auch einen eingebetteten Logikanalysator innerhalb eines elektronischen Systems enthält. Dargestellt ist ein Kabel **28**, das das elektronische System mit einem externen Computer verbindet.

[0087] Im Schritt **122** fordert der Benutzer über das EDA-Werkzeug den eingebetteten Logikanalysator mit einem geeigneten Befehl auf, mit der Arbeit zu beginnen. Fängt der Logikanalysator an zu laufen, so beginnt er im Schritt **124**, kontinuierlich Daten von den Signalen zu erfassen, die für die Überwachung festgelegt sind. Bevorzugt manipuliert der Benutzer nun das System so, dass es die vorhergehenden Fehlfunktionen wiederholt, die der Benutzer untersuchen will. Die erfassten Daten werden im Speicher des PLD abgelegt, und sie werden bevorzugt im zugewiesenen Speicher innerhalb des eingebetteten Logikanalysators selbst abgelegt. Im Schritt **126** wird geprüft, ob ein Haltepunkt aufgetreten ist. Anders beschrieben stellt der Logikanalysator fest, ob der Status der für die Überwachung spezifizierten Signale gleich dem Haltepunkt ist, den der Benutzer angegeben hat. Trifft dies nicht zu, so erfasst der Logikanalysator weiterhin Daten. Trifft dies zu, so wird im Schritt **128** festgestellt, ob nach dem Haltepunkt weitere Abtastwerte erfasst und gespeichert werden sollen. Der Schritt **128** kann implementiert werden, indem man die Gesamtanzahl der gewünschten Abtastwerte mit der Anzahl der Abtastwerte vergleicht, die bereits vor dem Erreichen des Haltepunkts gespeichert wurden. Sind weitere Abtastwerte zu speichern, so fährt der Logikanalysator im Schritt **130** nach dem Haltepunkt mit dem Speichern der gewünschten Anzahl Abtastwerte fort.

[0088] Wurde die vom Benutzer gewünschte Gesamtanzahl an Abtastwerten erfasst und gespeichert, so werden im Schritt **132** die abgelegten Daten aus dem Abtastspeicher des Logikanalysators auf dem Benutzercomputer übertragen. Der Logikanalysator lädt die gespeicherte Information bevorzugt über das Schnittstellenkabel auf den Benutzercomputer hoch. Im Schritt **134** kann der Benutzer diese vom Logikanalysator empfangenen Signale graphisch ansehen. In einer Ausführungsform werden die Signale in einer Kurvendarstellung angeboten, die mit den Signalnamen bezeichnet ist. Durch das Betrachten dieser interessierenden Signale auf einem Computer kann der Benutzer wirksam Fehler in einer Hardwarevorrichtung nahezu so suchen, als wäre er in der Lage, einen externen Logikanalysator an diese Signale anzuschließen.

[0089] Fig. 4 zeigt ein Flussdiagramm **200**, das eine Vorgehensweise beschreibt, mit der man eine Logikanalysatorschaltung automatisch in einen Benutzerentwurf einfügen und automatisch mit diesem Entwurf übersetzen kann. Die Vorgehensweise in Fig. 4 ist eine geeignete Technik, mit der man den Schritt **116** in Fig. 3B implementieren kann (im Zusammenhang mit dem Übersetzen des PLD). Natürlich kann man mit den verschiedenen EDA-Werkzeugen mehrere Übersetzungstechniken verwenden, und zwar sowohl für den PLD-Entwurf als auch für andere Arten von integrierten Schaltungen. Die Ausführungsform in Fig. 4 erläutert, wie man die Erfindung zusammen mit einer solchen Übersetzungstechnik verwenden kann, obwohl in Betracht gezogen wird, dass die Erfindung auf eine breite Vielfalt von Übersetzungstechniken anwendbar ist.

[0090] Im Schritt **202** nimmt ein EDA-Werkzeug die Benutzerentwurfsdateien an, die zum Beschreiben eines elektronischen Entwurfs für eine Vorrichtung, beispielsweise ein PLD, erforderlich sind. Diese Entwurfsdateien können häufig die Entwurfselemente für den Entwurf festlegen. In manchen Entwurfsumgebungen sind die Entwurfselemente hierarchisch angeordnet, d.h. von der Einheit auf oberster Ebene bis zu den Einheiten auf unterster Ebene. Wird in derartigen Fällen im Entwurf ein bestimmtes Register an zahlreichen Stellen innerhalb des Entwurfs verwendet, so kann es sein, dass nur eine Datei vorhanden ist, die die Implementierung dieses Entwurfs enthält, und der komplette Entwurf kann sich an zahlreichen Stellen auf diese eine Datei beziehen. Beispiele für derartige elektronische Entwurfsdateien sind weiter vorne anhand von Fig. 1 und Fig. 2 erklärt.

[0091] Im Schritt **204** werden diese Benutzerentwurfsdateien genommen, und es wird eine eingeebnete Netzlistendarstellung des Benutzerentwurfs erzeugt. Sollten die Entwurfsdateien – anders ausgedrückt – eine Hierarchie enthalten, so wird diese Hierarchie "eingeebnet", d.h. jede im Entwurf angesprochene Einheit wird so oft dupliziert wie sie benötigt wird. Wird im obigen Beispiel ein bestimmtes Register in der Entwurfshierarchie zwei Mal verwendet, so existiert jedoch nur eine Datei, die dieses Register beschreibt. In diesem Schritt wird der Entwurf eingeebnet, indem zwei derartige Dateien für das Register erzeugt werden. Die Darstellung des Benutzerentwurfs ist in diesem Schritt, wie Fachleuten bekannt ist, bevorzugt eine synthetisierte technologiegestützte Datenbank. An dieser Stelle des Übersetzungsablaufs wird eine synthetisierte Netzliste des Benutzerentwurfs in einem eingeebneten Zustand erzeugt. In der Regel bezeichnet man das Erstellen einer derartigen Netzliste als "Syntheseschritt" im Überset-

zungsablauf, und zwar nachdem die Bearbeitung erfolgt ist.

[0092] Im Schritt **206** wird eine Darstellung der Logikanalysatorschaltung auf Gatterebene erstellt. Die Logikanalysatorschaltung kann sehr unterschiedliche Formen annehmen. Als Beispiel kann die Logikanalysatorschaltung **260** in Fig. 8 verwendet werden. Man darf jedoch nicht übersehen, dass man eine Logikanalysatorschaltung auf zahlreiche unterschiedliche Weisen implementieren kann, die nach wie vor die Funktionalität der in Fig. 8 dargestellten Schaltung besitzen. In einer bevorzugten Ausführungsform der Erfindung ist eine Logikanalysatorschaltung in dem EDA-Werkzeug enthalten, so dass eine Darstellung auf Gatterebene automatisch erzeugt werden kann. Wahlweise kann man es dem Benutzer gestatten, eine eigenständige Schaltung festzulegen. Diese Darstellung auf Gatterebene bezieht jedes Logikelement der Schaltung ein und zwar zusammen mit der Anzahl und den Namen der zu überwachenden Signale, die der Benutzer im Schritt **108** festgelegt hat, der Anzahl der zu erfassenden Abtastwerte, die im Schritt **110** festgelegt werden, der Anzahl der vor dem Haltepunkt benötigten Abtastwerte, die im Schritt **115** festgelegt wird, und dem Haltepunkt, der im Schritt **114** festgelegt wird. Fachleuten ist der Ablauf bekannt, mit dem eine Darstellung auf Gatterebene für eine bestimmte Schaltung erzeugt wird.

[0093] Die tatsächliche Darstellung auf Gatterebene für eine bestimmte Logikanalysatorschaltung hängt von der besonderen Vorrichtung ab, in die der Logikanalysator eingebettet wird. Beispielsweise kann die Hardwarevorrichtung, in die der Logikanalysator einzubetten ist, irgendeine der PLD-Vorrichtungen enthalten, die von der Altera Corporation erhältlich sind. Besonders gut geeignet sind sämtliche Typen FLEX 10K, FLEX 8000, MAX 9000 oder MAX 7000. Jede dieser besonderen Vorrichtungen kann unterschiedliche Merkmale aufweisen, die darauf Einfluss nehmen, wie eine Darstellung auf Gatterebene für einen Logikanalysator erstellt wird. Beispielsweise ist für eine Vorrichtung FLEX 10K mit relativ großen eingebetteten Speicherabschnitten dieser eingebettete Speicher besonders gut zum Implementieren eines großen FIFO-Speichers (First-In, First-Out) für den Logikanalysator geeignet. Bei einer Vorrichtung ohne eingebetteten Speicher, beispielsweise der FLEX 8000, können die Speicherelemente (etwa SRAM-Flipflops) der Logikzellen als Speicher des Logikanalysators verwendet werden. Es kann jedoch sein, dass man den FIFO-Puffer über zahlreiche Zellen verteilen muss, falls der Speicher in einer einzigen Zelle zum Aufnehmen des Puffers nicht groß genug ist. In ähnlicher Weise kann man bei einer auf der EEPROM-Technologie beruhenden Vorrichtung ebenfalls eine oder mehrere Logikzellen für den Puffer des Logikanalysators verwenden. Eine Vorrichtung mit einem großen eingebetteten Speicher eignet sich besonders gut, da die Kapazität für die Signalspeicherung größer ist. Damit entsteht im Schritt **206** eine Darstellung für eine Logikanalysatorschaltung, die mit dem Benutzerentwurf zu verbinden ist.

[0094] Im Schritt **208** wird die Darstellung der Logikanalysatorschaltung auf Gatterebene aus dem Schritt **206** mit der eingeebneten Darstellung des Benutzerentwurfs im Schritt **204** verbunden. Dieser Schritt stellt die logischen Verbindungen von den Eingängen des Logikanalysators (Haltepunktsignale, Systemtakt, zu überwachende Signale) zu den Leitungen her, die die tatsächlichen Signale führen, die im Benutzerentwurf festgelegt sind. Da diese Signale bereits vorher in den Schritten **108** bis **115** festgelegt worden sind, kann das EDA-Werkzeug die jeweiligen Signalleitungen in seiner Datenbankdarstellung des elektronischen Entwurfs finden und die passende Verbindung zu einem Eingang des Logikanalysators herstellen.

[0095] Zudem werden in diesem Schritt geeignete Verbindungen vom Logikanalysator zu einer Schnittstelle zum Computer des Benutzers hergestellt. In der Ausführungsform in Fig. 6, die im Weiteren beschrieben wird, enthält diese Schnittstelle Testdatenregister **274**, die mit dem JTAG-Port **272** verbunden sind. Damit werden Eingabe- und Ausgabeverbindungen zum und vom Logikanalysator, die eine Schnittstelle zum Benutzercomputer liefern, mit dem passenden Testdatenregister verbunden. Beispielsweise werden, siehe Fig. 7 und Fig. 8, die Eingangssignale (vom Benutzercomputer zum Logikanalysator) Trigger Signals, Trigger Register, Set Delay, Delay[6:0], NextReq, StopReq, RunReq und Clear mit geeigneten Testregistern verbunden. Ausgangssignale (vom Logikanalysator zum Benutzercomputer) DataOut[15:0], NumSamples[7:0], Triggered und Run werden ebenfalls mit dem geeigneten Testregister verbunden. Testregister werden bevorzugt aus programmierbarer Logik aufgebaut und sind Fachleuten geläufig.

[0096] Natürlich kann man in einer anderen Ausführungsform einer Logikanalysatorschaltung andere Signale und/oder eine größere oder geringere Anzahl von Schnittstellensignalen verwenden. In einer bevorzugten Ausführungsform der Erfindung werden diese Schnittstellensignale zum und vom Logikanalysator mit zugewiesenen Anschlussstiften auf dem PLD verbunden, die für diesen Zweck reserviert sind. Damit weiß ein Benutzer, an welchen Anschlussstiften das Fehlersuchkabel befestigt werden muss. Wie erwähnt steuern diese Anschlussstifte nicht nur den eingebetteten Logikanalysator, sondern sie empfangen auch Daten vom Logikanalysator. In anderen Ausführungsformen kann man diese zugewiesenen Anschlussstifte auch mit einem anderen Teil der Platine verbinden, damit man ein Kabel leicht anbringen kann. Auf diese Weise wird die Logik für die Logika-

nalysatorschaltung, die im Schritt **206** erzeugt wurde, für die Kommunikation mit dem Benutzercomputer sowohl mit dem Benutzerentwurf als auch den Schnittstellenstiften des PLD verbunden.

[0097] Im Schritt **210** wird der vollständige im Schritt **208** erzeugte Entwurf in einer Weise platziert und verbunden, mit der Fachleute vertraut sind. Die Ausgabe des Platzierungs- und Verbindungsschritts wird daraufhin in den Schritt **212** eingegeben, in dem die Ausgabedatei zusammengestellt wird. Diese Ausgabedatei kann nun in ein PLD heruntergeladen werden, um es zu programmieren. Nach dem Programmieren eines PLD mit dieser Datei kann der Benutzer mit dem Gebrauch des eingebetteten Logikanalysators beginnen, um in der Vorrichtung Fehler zu suchen.

[0098] Fig. 5 zeigt eine weitere Ansicht des programmierbaren Logikentwicklungssystems **10** in Fig. 1. Dargestellt ist eine programmierbare Logikvorrichtung, die einen eingebetteten Logikanalysator innerhalb eines elektronischen Systems aufweist. Das System **10** zeigt ein elektronisches System **252**, das mit einem Computersystem A **18** über ein Kabel **28** oder eine andere Verbindungsvorrichtung verbunden ist. Das elektronische System **252** enthält ein PLD **16**, das eine Komponente des elektronischen Systems darstellt. Das PLD **16** teilt sich möglicherweise eine oder mehrere elektronische Verbindungen **254** mit den anderen Komponenten und Elementen, die das elektronische System bilden. Das PLD **16** ist mit einem Benutzerlogikentwurf **256** und einem eingebetteten Logikanalysator **260** konfiguriert. Die Benutzerlogik **256** ist mit einem Entwurf gemäß der in Fig. 2 beschriebenen Verfahrensweise konfiguriert oder mit irgendeiner anderen geeigneten Entwurfsverfahrensweise. Der eingebettete Logikanalysator **260** ist in das PLD **16** gemäß einer Ausführungsform der Erfindung integriert, die in Fig. 3A und Fig. 3B beschrieben ist.

[0099] Die Logikverbindungen **262** erlauben es, Signale von der Benutzerlogik **256** an den Logikanalysator **260** zu übertragen. Diese Signale können einen Systemtakt, Triggersignale, zu überwachende Signale usw. enthalten. Die Anschlussstifte des PLD **16** dienen zum Verbinden der Schnittstellensignale **264** vom Logikanalysator mit zugehörigen Anschlüssen **266** im elektronischen System **252**. Das Kabel **28** dient dazu, diese Schnittstellensignale mit dem Computer **18** zu verbinden. Wahlweise kann der Computer **18** direkt mit dem PLD **16** verbunden werden und Schnittstellensignale **264** an das PLD übertragen. Auf diese Weise überträgt der Computer **18** Befehle und andere Informationen an den eingebetteten Logikanalysator **260**, und er empfängt Information vom Logikanalysator, ohne den funktionalen Betrieb des elektronischen Systems **252** direkt zu unterbrechen oder zu beeinträchtigen. Das PLD **16** ist somit dafür konfiguriert, sowohl die Funktion der Benutzerlogik **256** als auch des eingebetteten Logikanalysators **260** auszuführen.

[0100] Fachleuten ist bekannt, dass die tatsächlich verwendete Schnittstelle zwischen dem Logikanalysator **260** und einem externen Computersystem mehrere unterschiedliche Formen annehmen kann. Der hier beschriebene eingebettete Logikanalysator kann von einem außen angeordneten Computer über jede beliebige geeignete Schnittstelle auf dem PLD gesteuert werden. Zudem kann sich die exakte Anzahl der Anschlussstifte auf dem PLD **16**, die zum Steuern des Logikanalysators **260** und zum Empfangen von Daten vom Logikanalysator verwendet werden, abhängig von der programmierten Vorrichtung, dem Gesamtplatinenentwurf usw. unterscheiden. Natürlich kann man die verwendeten Anschlussstifte flexibel zuweisen oder man kann die zugeordneten Schnittstellenstifte verwenden.

[0101] Man kann auch andere Techniken zum Steuern eines eingebetteten Logikanalysators und zum Empfangen von Ausgabedaten verwenden.

[0102] Fig. 6 zeigt eine weitere Ansicht des PLD **16** und stellt eine bevorzugte Ausführungsform zum Steuern eines Logikanalysators über den JTAG-Port der Vorrichtung dar, in die der Logikanalysator eingebettet ist. Zur Vereinfachung ist die Benutzerlogik **256** innerhalb des PLD **16** nicht dargestellt. In dieser bevorzugten Ausführungsform werden die Schnittstellensignale **264** mit Hilfe eines JTAG-Ports **272** zusammen mit der Steuerlogik **274** und Signalen **275** implementiert. Ein JTAG-Port **272** (JTAG = Joint Test Action Group) wird gemäß dem IEEE-Standard 1149.1 implementiert und ist Fachleuten bekannt. Die Steuerlogik **274** liefert die Pufferung zwischen dem Logikanalysator **260** und dem JTAG-Port **272** für bestimmte Signale, die im Folgenden in Fig. 7 beschrieben werden. Im Einzelnen liefert die Steuerlogik **274** Steuersignale an den Logikanalysator **260** und unterstützt das Holen von Daten und Statusangaben vom Logikanalysator.

[0103] In dieser Ausführungsform enthält der JTAG-Port **272** die Signale TCLK, TMS, TDI und TDO. Das Signal TCLK ist ein Taktsignal, das die serielle Eingangs- und Ausgangsdatenrate des JTAG-Ports **272** steuert. Das Signal TMS ist ein Modusauswahlsignal, das irgendeinen der sechzehn Zustände des JTAG-Ports auswählt. Die Signale TDI und TDO bedeuten serielle Eingabedaten bzw. serielle Ausgabedaten.

[0104] Normalerweise dient ein JTAG-Port entweder dazu, ein PLD zu programmieren oder das Prüfen einer Platine zu unterstützen, auf der sich PLDs befinden. Vorteilhafterweise hat man erkannt, dass der JTAG-Port bisher während des Entwurfs eines be-

stimmten PLD und bei der Fehlersuche darin nicht verwendet worden ist. Somit hat man ferner erkannt, dass der JTAG-Port auf einem PLD nicht vollständig ausgelastet ist und man ihn während der Fehlersuche in einem PLD als Kommunikationsmittel mit dem eingebetteten Logikanalysator der Erfindung und zum Steuern des Logikanalysators verwenden kann. In vorteilhafter Weise wird ein Standard-JTAG-Port dazu verwendet, die Fehlersuche in einer programmierbaren Logikvorrichtung zu vereinfachen, die einen eingebetteten Logikanalysator enthält. Zwei besondere Ausführungsformen zum Implementieren der Steuerlogik **274**, die das Steuern eines eingebetteten Logikanalysators über einen JTAG-Port vereinfacht, werden im Weiteren anhand von Fig. 11–Fig. 13 und Fig. 14–17 beschrieben.

[0105] Fig. 7 erläutert die Eingabe- und Ausgabesignale für den eingebetteten Logikanalysator **260** gemäß einer Ausführungsform der Erfindung. Die Signale DataIn **280** sind die Signale, die der Benutzer im Schritt **108** festgelegt hat und die zum Zweck der Fehlersuche im PLD zu überwachen sind. Das Signal SetDelay **281** ist eine Steuerleitung, die bewirkt, dass das Register **310** mit dem Wert des Signals Delay **282** geladen wird, das die Anzahl der Abtastwerte angibt, die nach einem Haltepunkt zu erfassen sind. Das Signal Delay **282** bezeichnet die Anzahl der Abtastwerte, die nach einem Haltepunkt zu erfassen sind, und es wird vom Computersystem **18** empfangen, nachdem es der Benutzer spezifiziert hat. Das Signal Breakpoint **283** zeigt dem Logikanalysator an, dass ein Haltepunkt aufgetreten ist. Dieses Signal erzeugt der Triggerkomparator **306** innerhalb des Logikanalysators, oder es kann in der vom Benutzer entworfenen Logik erzeugt werden.

[0106] Das Signal NextReq **284** wird vom Computersystem **18** empfangen und erlaubt das Holen von gespeicherten Abtastdaten, und zwar einen Abtastwert je Zeiteinheit, und es gibt an, dass der folgende Abtastwert auf das Computersystem **18** hochgeladen werden sollte. Das Signal StopReq **285** wird vom Computersystem **18** empfangen und weist den Logikanalysator an, in seinen Haltstatus zu gehen und das Erfassen von Signalabtastwerten zu beenden. Das Signal RunReq **286** wird vom Computersystem **18** empfangen und weist den Logikanalysator an, mit der Arbeit zu beginnen und Abtastdaten zu empfangen. Das Signal DoneDump **287** weist den Logikanalysator an, die Ausgabe der Daten aus seinem Speicher auf das Computersystem zu unterbrechen und in einen Haltstatus zu gehen. Dieses Signal kann aus dem Logikanalysator stammen oder vom Benutzer. Das Signal Clock **288** ist das im Schritt **112** spezifizierte Systemtaktsignal. Das Signal Clear **289** ist ein Rücksetzsignal, das die Zustandssteuer-Maschine **302**, den Abtastspeicher **324** und den Zähler **314** zurücksetzt.

[0107] Die Signale DataOut **290** sind die Ausgangssignale des Abtastspeichers **324**, die die erfassten Signale über die Schnittstelle **264** vom Logikanalysator **260** zum Computersystem **18** übertragen, und zwar ein Wort je Zeiteinheit. Das Signal NumSamples **291** bezeichnet die Anzahl der gültigen Abtastwerte, die der Logikanalysator **260** erfasst hat. Da die tatsächliche Anzahl gültiger Abtastwerte, die der Logikanalysator erfasst hat, geringer sein kann als die Gesamtanzahl an Abtastwerten, die der Benutzer gefordert hat, unterstützt dieses Signal den Benutzer beim Feststellen, welche der im Speicher abgelegten Signale die gültigen sind. Ein Benutzer mag beispielsweise wünschen, insgesamt 128 Abtastwerte zu erfassen; er wünscht jedoch, nach dem Auftreten eines Haltepunkts keinerlei Abtastwerte zu speichern. Tritt ein Haltepunkt auf, nachdem erst vierundsechzig Abtastwerte erfasst sind, so hat das Signal NumSamples **291** einen Wert von vierundsechzig. Dies zeigt an, dass nur vierundsechzig im Logikanalysator gespeicherte Abtastwerte gültige Werte sind. Jegliche hinter den vierundsechzig Werten gespeicherte Abtastwerte sind keine gültigen Abtastwerte. Es ist möglich, dass sie von früheren Datenerfassungen stammen. Das Signal Triggered **292** ist ein Ausgangssignal für den Benutzer, das anzeigt, dass ein Haltepunkt aufgetreten ist. Das Signal Run **293** zeigt dem Benutzer an, dass das Signal RunReq **286** empfangen worden ist und dass der Logikanalysator läuft und Daten erfasst.

[0108] Fig. 8 zeigt den eingebetteten Logikanalysator **260** gemäß einer Ausführungsform der Erfindung. Der Logikanalysator, der in ein PLD eingebettet werden soll, kann abhängig von der Art des PLD, der Art und Anzahl der zu überwachenden Signale, der gewünschten Datentiefe, dem verfügbaren Speicher, den Steuersignalen vom Benutzercomputer, den Vorlieben des Entwicklungsingenieurs usw. auf viele verschiedene Weisen implementiert werden. Um ein Beispiel anzugeben stellt der Logikanalysator **260** ein besonders Beispiel dafür vor, wie man einen derartigen Logikanalysator implementieren kann. Der Benutzer steuert den eingebetteten Logikanalysator vom einem Computer außerhalb des PLD. Der Logikanalysator arbeitet so, dass er beliebige Signale einer Reihe interner Signale erfasst, die der Benutzer wünscht. In dieser Ausführungsform der Erfindung enthält der Logikanalysator **260** eine Zustandssteuer-Maschine **302**, ein Triggerregister **304**, einen Triggerkomparator **306**, Register **308** und **310**, Zähler **312**–**316**, Komparatoren **320**, **322** und Abtastspeicher **324**.

[0109] Nach dem Programmieren des Logikanalysators **260** in ein PLD empfängt er verschiedene Eingangssignale aus dem Inneren des PLD. Die Triggersignale **305** sind diejenigen im Schritt **114** festgelegten Signale, die der Benutzer mit einer Triggerbedingung zu vergleichen wünscht, die im Triggerregister

304 abgelegt ist und ebenfalls im Schritt **114** bestimmt wird. Gemeinsam wirken die Triggersignale **305**, die die Triggerbedingung erfüllen, als Haltepunkt. Der Triggerkomparator **306** vergleicht die Signale **305** mit der Triggerbedingung und erzeugt ein Haltepunktsignal, wenn die Bedingung erfüllt ist.

[0110] Die Zustandssteuer-Maschine **302** kann irgendeine geeignete Kontrollstruktur zum Steuern des eingebetteten Logikanalysators sein; sie wird in Fig. 9 ausführlicher beschrieben. Bevorzugt wird die Zustandsmaschine **302** in programmierbarer Logik entworfen, wobei nach Belieben Tabellen, eingebettete Speicherblöcke, ROM-Register usw. verwendet werden können. Das Eingangssignal DelayDone wird vom Komparator **320** empfangen und gibt an, dass die Gesamtanzahl der vom Benutzer geforderten Abtastwerte erfasst ist. Die Signale NextReq, StopReq, RunReq, DoneDump, Clock und Clear sind bereits beschrieben worden. Das Eingangssignal Breakpoint für die Zustandsmaschine **302** wird vom Triggerkomparator **306** über das Register **308** empfangen.

[0111] Das Register **308** ist ein Synchronisierregister, das dazu dient, die Zustandsmaschine **302** von einem asynchronen Haltepunktsignal abzuschirmen. Das Register **308** erlaubt es, das Eingangssignal Haltepunkt in einem Taktzyklus einzugeben.

[0112] Das Ausgangssignal Stopped ist aktiv, wenn sich die Zustandsmaschine **302** im angehaltenen Zustand befindet. Dieses Signal setzt die Zähler **312** und **316** zurück und bereitet den Logikanalysator darauf vor, einen neuen Lauf zu beginnen. Es erlaubt es auch, Daten aus dem Register **310** in den Zähler **314** zu laden. Das Ausgangssignal Next gibt im aktiven Zustand den Zähler **312** frei, damit eine Adresse im Abtastspeicher **324** erhöht wird. Die Adressen werden erhöht, während der Logikanalysator läuft und Daten erfasst und während der Abtastspeicher **324** seine Adressen durchläuft und Abtastdaten an das Computersystem **18** ausgibt. Das Ausgangssignal Triggered dient zum Freigeben des Zählers **314**. Das Signal Run wird im Gatter **340** mit dem Signal PrevDataFull verknüpft, um den Zähler **316** freizugeben.

[0113] Das Register **310** empfängt das Signal Delay zum Speichern der Anzahl der Abtastwerte nach einem Haltepunkt, die ein Benutzer erfassen will. Sein Takt ist das Eingangssignal SetDelay.

[0114] Der Zähler **312** erhöht während des Abtastens und Datenerfassens die Adressen für den Abtastspeicher **324**, und er inkrementiert Adressen während des Auslesens von Daten zu einem Computersystem **18**. Der Zähler **314** ist ein Abwärtszähler, der eine Verzögerung ab einem erkannten Haltepunkt erzeugt, damit der Logikanalysator mit dem Datenerfassen fortfahren kann, bis der letzte vom Benutzer gewünschte Abtastwert gespeichert ist. Der Zähler **314** wird aus dem Register **310** mit dem Signal Delay geladen, das die Anzahl der Abtastwerte nach dem Haltepunkt angibt, die gespeichert werden sollen. Zählt der Zähler **314** herunter und erreicht er einen Wert von Null, so führt der Komparator **320** einen erfolgreichen Vergleich mit einem festverdrahteten Wert Null aus und setzt das Signal DelayDone. Das gesetzte Signal DelayDone weist die Zustandssteuer-Maschine **302** an, vom Run-Status in den Datenausgabestatus überzugehen.

[0115] Der Zähler **316** zählt die Anzahl der gültigen Abtastwerte, die im Abtastspeicher **324** abgelegt sind. Wie bereits besprochen kann eine Situation auftreten, in der die Anzahl der erfassten gültigen Abtastwerte kleiner ist als die ursprünglich spezifizierte Anzahl der Abtastwerte, die der Benutzer wünscht. In diesem erläuternden Beispiel hat der Abtastspeicher **324** eine Kapazität von 128 Wörtern. Erreicht der Zähler **316** den Wert 128, so nimmt der Komparator **322** einen erfolgreichen Vergleich mit einem festverdrahteten Wert 128 vor. Dadurch wird das Ausgangssignal PrevDataFull gesetzt, und sein invertiertes Signal wird in das Gatter **340** eingegeben. Die Inverse eines gesetzten Signals PrevDataFull sperrt den Zähler **316** und zeigt dadurch an, dass die Gesamtanzahl von gültigen Abtastwerten im Abtastspeicher **324** abgelegt ist. Der Zähler **316** gibt kontinuierlich das Signal NumSamples aus, das die Gesamtanzahl von gültigen Abtastwerten angibt, die zu diesem Zeitpunkt bereits erfasst sind.

[0116] Der Abtastspeicher **324** ist ein Speicher innerhalb des PLD **16**, der auf jede beliebige geeignete Weise implementiert werden kann. Der Speicher **324** kann beispielsweise mit Hilfe von Registersätzen oder mit eingebetteten Speicherblöcken innerhalb des PLD implementiert werden. In einer besonderen Ausführungsform wird eingebetteter SRAM-Speicher zum Implementieren des Speichers **324** verwendet. Natürlich lässt sich der Speicher **324** in zahlreichen PLD-Typen implementieren, die keinen zusätzlichen eingebetteten Speicher enthalten. Der Abtastspeicher **324** wird bevorzugt als Ringpuffer implementiert, so dass eingehende Abtastdaten kontinuierlich im Speicher **324** abgelegt werden, solange der Logikanalysator läuft. Ist der Speicher **324** voll, so kehrt er an seinen Anfang zurück, und die ältesten gespeicherten Daten werden von neu ankommenden Daten überschrieben. Zu Erläuterungszwecken ist der Abtastspeicher **324** als RAM-Speicher von 128 Wörtern mit je 16 Bit implementiert dargestellt. Natürlich kann man jede beliebige Speichergröße verwenden. Man kann die Breite des Speichers erhöhen, um mehr Datensignale zu erfassen, und seine Tiefe erhöhen, um länger zurückliegende Signalwerte zu speichern.

[0117] Fig. 9 zeigt ein Zustandsdiagramm der Zustandssteuer-Maschine **302**. Fig. 10 zeigt eine Tabelle, die den Wert von Ausgangssignalen erläutert, die

jedem der fünf Zustände der Zustandsmaschine **302** zugeordnet sind. Fachleuten ist geläufig, dass man die Zustandssteuer-Maschine **302** mit jeder beliebigen Logik implementieren kann, und dass sie nicht notwendig als Zustandsmaschine implementiert werden muss. Wird sie als Zustandsmaschine implementiert, so kann man verschiedene Zustandsmaschinen verwenden, beispielsweise eine Moore- oder eine Mealy-Zustandsmaschine. In einer bevorzugten Ausführungsform der Erfindung wird die Zustandsmaschine **302** als Moore-Zustandsmaschine implementiert. Dies wird nun anhand von Fig. 9 und Fig. 10 beschrieben.

[0118] Die Zustandsmaschine **302** umfasst die Zustände Run **402**, Delay **404**, Stop **406**, DataDump **408** und Next **410**. Im Stop-Zustand wird das Ausgangssignal Stopped gesetzt, und die anderen Ausgangssignale Run, Triggered und Next befinden sich im Zustand "bedeutungslos". Der Zustand Stop wird beibehalten, bis ein Eingangssignal RunReq gesetzt wird, das einen Übergang in den Run-Zustand bewirkt.

[0119] Im Run-Zustand sind die Ausgangssignale Run und Next gesetzt, und das Ausgangssignal Triggered hat den Wert Null, weil noch kein Haltepunkt aufgetreten ist. Solange die beiden Eingangssignale Breakpoint und StopReq nicht gesetzt werden, wird der Run-Zustand beibehalten. Wird ein StopReq-Signal empfangen, so geht die Zustandsmaschine in den Stop-Zustand. Wird ein Breakpoint-Signal empfangen und bleibt das Signal DelayDone ungesetzt (dies zeigt an, dass noch weitere Abtastwerte zu erfassen sind), so geht das System in den Delay-Zustand. Wird dagegen ein Breakpoint-Signal bei gesetztem DelayDone-Signal empfangen, so zeigt dies nicht nur an, dass ein Haltepunkt aufgetreten ist, sondern dass keine Signale nach dem Haltepunkt erfasst werden müssen. In dieser Situation geht die Zustandsmaschine in den DataDump-Zustand.

[0120] Im Delay-Zustand sind die Signale Run, Triggered und Next gesetzt, und das Ausgangssignal Stopped ist nicht gesetzt. Der Delay-Zustand zeigt an, dass der Logikanalysator noch läuft und Daten erfasst, nachdem eine Triggerbedingung den Haltepunkt gesetzt hat. Die Zustandsmaschine bleibt in diesem Zustand, bis alle erforderlichen Abtastwerte nach dem Haltepunkt erfasst sind. Somit wird der Delay-Zustand beibehalten, solange das DelayDone-Signal nicht gesetzt ist. Sobald der Zähler **314** heruntergezählt hat und anzeigt, dass alle geforderten Abtastwerte nach dem Haltepunkt erfasst sind, wird das DelayDone-Signal gesetzt, und die Zustandsmaschine geht in den DataDump-Zustand.

[0121] Im DataDump-Zustand sind alle Ausgangssignale ungesetzt. In diesem Zustand werden Daten aus dem Abtastspeicher **324** an das Computersystem **18** übertragen. Wird das Signal DoneDump gesetzt, so hat der Logikanalysator das Hochladen von Daten auf das Computersystem **18** beendet. Die Zustandsmaschine geht aus dem DataDump-Zustand in den Stop-Zustand über. Die Zustandsmaschine bleibt im DataDump-Zustand, solange die Signale NextReq und DoneDump beide nicht gesetzt sind. Wie bereits früher beschrieben kann das Computersystem **18** ein Hochladen eines Worts in einer Zeiteinheit aus dem Abtastspeicher **324** anfordern. In dieser Situation setzt das Computersystem **18** das Eingangssignal NextReq und veranlasst dadurch die Zustandsmaschine, in den Next-Zustand überzugehen.

[0122] Im Next-Zustand sind alle Ausgangssignale mit Ausnahme des Ausgangssignals Next nicht gesetzt. Solange der Logikanalysator weiterhin ein gesetztes Signal NextReq empfängt, lädt er ein Wort je Zeiteinheit auf das Computersystem **18** hoch. Sobald das Signal NextReq zurückgesetzt wird, kehrt die Zustandsmaschine in den DataDump-Zustand zurück.

ÜBERBLICK ÜBER DIE STEUERUNG DES JTAG-PORTS

[0123] Wie bereits anhand von Fig. 6 beschrieben wird in einer bevorzugten Ausführungsform der Erfindung der JTAG-Port **272** zusammen mit der Steuerlogik **274** und den Signalen **275** zum Steuern des Logikanalysators **260** verwendet. Es wurde erkannt, dass es vorteilhaft ist, einen JTAG-Port zum Steuern eines Logikanalysators zu verwenden, weil ein JTAG-Port häufig bereits auf einem PLD vorhanden ist. Verwendet man einen JTAG-Port, so sind zudem zusätzliche zugeordnete Fehlersuch-Steuerstifte überflüssig. Zudem besitzen zahlreiche PLD-Hersteller bereits Einrichtungen für den Anschluss an einen JTAG-Port eines PLD und die Kommunikation darüber. Die Altera Corporation in San Jose, Kalifornien, verwendet beispielsweise ein internes Produkt mit dem Namen "Byte Blaster" zum Programmieren eines PLD über einen JTAG-Port. Aus diesen und anderen Gründen wurde festgestellt, dass der Gebrauch eines JTAG-Ports zum Steuern eines eingebetteten Logikanalysators vorteilhaft ist. Trotzdem ist aus unterschiedlichen Gründen nicht unmittelbar zu sehen, wie man eine solche Steuerung mit einem JTAG-Port implementieren kann.

[0124] Um Hintergrundwissen bereitzustellen werden nun weitere Einzelheiten über den Gebrauch eines JTAG-Ports beschrieben. Eine JTAG-Prüfvorrichtung prüft eine Hardwarevorrichtung, die einen JTAG-Port aufweist, indem sie im Grunde die Kontrolle des Anschlussfleck-Rings der Vorrichtung übernimmt. Anders ausgedrückt übernimmt die JTAG-Prüfvorrichtung die Kontrolle über die Treiber für jeden Anschlussstift und isoliert damit effektiv den Kern der Vorrichtung gegen die Außenwelt. Mit Hilfe

des JTAG-Ports der Vorrichtung kann die JTAG-Prüfvorrichtung nun jeden Anschlussstift in einen von drei Zuständen versetzen, nämlich Ansteuern, Erfassen oder Tri-State. Die JTAG-Prüfvorrichtung wird hauptsächlich in einem EXTEST-Modus verwendet, um einen vollständigen Platinentest der mechanischen Verbindungen zwischen den Vorrichtungen auf der Platine vorzunehmen. In diesem Modus kann die JTAG-Prüfvorrichtung, indem sie einen Anschlussstift an einer Vorrichtung so ansteuert, dass er ein Signal ausgibt, und eine Eingabe an einer anderen Vorrichtung auf der Platine erfasst, die mechanische Verbindung zwischen den Vorrichtungen im eingebauten Zustand auf der Platine prüfen. Als solcher ist der EXTEST-Modus zum Steuern eines eingebetteten Logikanalysators nicht geeignet. Der INTEST-Modus wird seltener verwendet; er dient zur internen Prüfung einer Vorrichtung. Wie oben übernimmt die JTAG-Prüfvorrichtung die Kontrolle über die Treiber für jeden Anschlussstift-Treiber und isoliert den Kern. Nun kann man Prüfsignale in den Kern einspeisen und Ausgangssignale abtasten und ihre Genauigkeit ermitteln.

[0125] Da der INTEST-Modus den Kern der Vorrichtung von der Außenwelt abtrennt, wird das PLD ungünstigerweise nicht in einer realen Umgebung auf der Platine geprüft. Wie bereits erklärt ist es häufig erforderlich, ein PLD in einer realen Umgebung auf der arbeitenden Platine zu prüfen, um versteckte Fehlfunktionen aufzuspüren. Zudem ist ein JTAG-Port nur ein serieller Port mit 10 MHz und kann damit nicht die großen Datenmengen liefern, die in einer realen Umgebung mit hoher Geschwindigkeit anfallen können. Somit sind tatsächliche Betriebsbedingungen bei hoher Geschwindigkeit erwünscht. Zudem liefert der Ingenieur während einer JTAG-Prüfung erfundene Prüfvektoren, die möglicherweise nicht repräsentativ für reale Signale sind, die das PLD in einer echten Arbeitsumgebung empfängt. Aus diesen und anderen Gründen ist es nicht besonders wünschenswert, zu versuchen, den eingebetteten Logikanalysator eines PLD mit Hilfe des INTEST-Modus des JTAG-Ports zu steuern. Trotzdem stellt man fest, dass der Gebrauch eines JTAG-Ports in einer bestimmten Weise zum Steuern einer eingebetteten Logikvorrichtung erwünscht ist. Die Erfindung zieht in vorteilhafter Weise zwei Ausführungsformen in Betracht, durch die der JTAG-Port **272** den eingebetteten Logikanalysator **260** eines PLD **16** steuert. Der JTAG-Port wird vorteilhaft dazu verwendet, den eingebetteten Logikanalysator zu steuern, wogegen das PLD, in dem der Logikanalysator eingebettet ist, auf der Platine unter realen Bedingungen arbeiten kann. Diese beiden Ausführungsformen werden im Weiteren anhand von Fig. 11–Fig. 13 bzw. Fig. 14–17 dargestellt.

ERSTE AUSFÜHRUNGSFORM DER JTAG-STEUERUNG

[0126] Fig. 11 zeigt eine erste Ausführungsform, über die ein JTAG-Port **272** einen eingebetteten Logikanalysator **260** eines PLD **16** über Gruppen von unverbundenen Ein/Ausgabe-Zellen **504** und **506** steuert. Der Logikanalysator **260** ist in den Kern **502** des PLD **16** eingebettet und besitzt einen Systemtakt **288**. Die Zellen **504** liefern Signale **514** an den Logikanalysator, und die Zellen **506** empfangen Signale **516** vom Logikanalysator. Die Signale **275** stellen Signale vom JTAG-Port **272** zu den Ein/Ausgabe-Zellen **504** und Signale von den Ein/Ausgabe-Zellen **506** dar. Hierzu gehören: das Signal TDI, das den Anschluss an eine serielle Dateneingabe (SDI) der ersten Eingabezelle **504** herstellt; das Signal TDO, das den Anschluss an eine serielle Datenausgabe (SDO) der letzten Ausgabezelle **506** herstellt; und Steuersignale wie Shift **680**, Clock **682**, Update **684** und Mode **686**, die jeder Zelle nach Bedarf geliefert werden. In dieser Ausführungsform dienen die für JTAG freigegebenen Ein/Ausgabe-Zellen **504** zum Steuern des Logikanalysators **260** über Eingangssignale **514**. Die Ausgabedaten und Statusinformationssignale **516** vom Logikanalysator **260** werden an die für JTAG freigegebenen Ein/Ausgabe-Zellen **506** angelegt.

[0127] Zum Implementieren dieser Ausführungsform wird die Netzliste des PLD erweitert, so dass jede Eingabe und jede Ausgabe des Logikanalysators **260** an eine unverbundene oder anderweitig unbenutzte Ein/Ausgabe-Zelle **504** bzw. **506** gelegt wird. Für jede Ein/Ausgabe-Zelle wird ein besonderes "debug RAM bit" gesetzt, das dazu dient, ein Steuersignal in den Logikanalysator einzugeben. Dieses Bit erlaubt es, ein Steuersignal mit Hilfe des JTAG-Ports zu treiben, obwohl sich die Vorrichtung nicht im INTEST-Modus befindet. Zum Erzeugen eines Steuersignals für die Eingabe in den Logikanalysator **260** wird der bekannte JTAG-Sample/Preload-Befehl eingescannt, die Steuerinformation wird eingescannt, und anschließend wird in den Update/Data-Recovery-Status des JTAG-Ports **272** gegangen. Weitere Einzelheiten werden im Folgenden angegeben. Um die Vorteile dieser Ausführungsform besser zu erläutern wird der bekannte JTAG-INTEST-Modus kurz beschrieben.

[0128] Fig. 12 zeigt eine herkömmliche für JTAG freigegebene Ein/Ausgabe-Zelle **600**, die eine geeignete Grundlage für die Beschreibung dieser Ausführungsform bietet. Die Zelle **600** ist mit einem externen PLD-Anschlussstift **602** verbunden. Über den Anschlussstift **602** wird das Eingangssignal **604** an den Kern **502** des PLD **16** geliefert. In ähnlicher Weise entstehen die Signale Output **606** und Output Enable **608** im Kern **502** und dienen dazu, ein Ausgangssignal am Anschlussstift **602** zu erzeugen. Die Multiple-

xer **610**, **612** und **614** wählen Daten, die in die Erfassungsregister **620** bzw. **622** bzw. **624** geladen werden müssen. Die Erfassungsregister werden dazu verwendet, Daten zunächst vom JTAG-Port **272** durch die Ein/Ausgabe-Zellen der Vorrichtung einzulesen. Die Aktualisierungsregister **630**, **632** und **634** empfangen Daten von den Erfassungsregistern und dienen zum Ausführen eines parallelen Ladevorgangs in den Kern der Vorrichtung. Die Multiplexer **640**, **642** und **644** wählen Daten entweder vom Anschlussstift Input **602** oder Output Enable **608** oder Output **606** oder von einem der Aktualisierungsregister, um ein passendes Signal zu erzeugen. Der Multiplexer **640** erzeugt ein Eingangssignal **604**. Der Multiplexer **642** erzeugt ein Tri-State-Signal für den Treiber **650**, und der Multiplexer **644** erzeugt ein Datensignal für den Treiber **650**, der ein Ausgangssignal am Anschlussstift **602** erzeugt, wenn er freigegeben ist.

[0129] Die Arbeitsweise der JTAG-freigegebenen Zelle **600** ist Fachleuten bekannt. Serial-Data-In **672** ist ein Signal, das von der vorhergehenden Ein/Ausgabe-Zelle (oder vom TDI-Signal am JTAG-Port **272**, falls es sich um die erste Zelle handelt) empfangen wird. Serial-Data-Out **674** ist mit dem Serial-Data-In-Signal der folgenden Ein/Ausgabe-Zelle verbunden (oder mit dem TDO-Signal des JTAG-Ports **272**, falls es sich um die letzte Zelle handelt). Die Steuersignale Shift **680**, Clock **682**, Update **684** und Mode **686** werden von jeder Zelle über Steuerleitungen **275** vom JTAG-Port **272** empfangen. Shift **680** wirkt so, dass es das Serial-Data-In-Signal **672** durch die Zelle **600** schiebt. Clock **682** taktet die Erfassungsregister, damit sie die seriellen Daten erfassen, und Update **684** taktet die Aktualisierungsregister, damit ein paralleler Ladevorgang möglich wird. Mode **686** erlaubt es der Zelle **600**, dass sie sich entweder in einem Normalmodus (Mode = 0) befindet oder im INTEST- oder EXTEST-Modus (Mode = 1). Versetzt Mode **686** im normalen JTAG-Betrieb die Vorrichtung in den INTEST- oder EXTEST-Modus, so werden die Signale Output **606**, Output Enable **608** und Input **604** effektiv vom Anschlussstift **602** getrennt, und der Kern **502** ist damit von der Umgebung abgetrennt. Wie bereits beschrieben ist es unerwünscht, die Vorrichtung vollständig in den INTEST-Modus zu versetzen, wenn man die Vorrichtung unter realistischen Umgebungsbedingungen prüfen will. Versetzt beispielsweise Mode **686** die Vorrichtung in den INTEST-Modus, so sind alle Anschlussstifte der Vorrichtung vom Kern der Vorrichtung abgetrennt und ein normaler Betrieb der Vorrichtung ist nicht möglich.

[0130] Im Standard-JTAG-INTEST-Modus geht der JTAG-Port **272** zuerst in einen seriellen Schiebemodus, in dem über die Leitung TDI empfangene serielle Daten durch alle Ein/Ausgabe-Zellen der Vorrichtung geschoben werden, bis die passenden Daten in der passenden Eingabezelle angekommen sind. Daraufhin tritt der JTAG-Port **272** in einen parallelen Lademodus ein, in dem alle gerade in die Ein/Ausgabe-Zellen geschobenen Daten nun parallel in den Kern **502** geladen werden, damit die Eingangssignale zugeführt werden, die für einen bestimmten Prüfvorgang nötig sind. Gleichzeitig oder nachfolgend kann der JTAG-Port **272** in einem parallelen Erfassungsmodus für diverse Ein/Ausgabe-Zellen sein, die die Ausgangssignale des Prüfvorgangs empfangen. Nachdem diese Prüfdaten empfangen sind, geht der JTAG-Port **272** erneut in den seriellen Schiebemodus, und die Daten werden seriell aus den Ein/Ausgabe-Zellen geschoben, damit sie der JTAG-Port **272** über die Leitung TDO ausgeben kann. Auf diese Weise werden im INTEST-Modus des JTAG-Ports **272** die JTAG-freigegebenen Ein/Ausgabe-Zellen dazu verwendet, dem Kern **502** erfundene Prüfdaten zu liefern und Ausgangssignale aus dieser Prüfung zu erhalten.

[0131] Wie bereits beschrieben kann es jedoch nachteilig sein, den INTEST-Modus zu verwenden, da der INTEST-Modus alle Anschlussstifte des PLD **16** vom Kern **502** abtrennt. Damit arbeitet das PLD **16** nicht unter realistischen Bedingungen, und es kann sein, dass der Einsatz des Logikanalysators **260** in diesem Modus nicht die richtigen Testergebnisse liefert. Die Erfindung macht vorteilhafterweise von unverbundenen Ein/Ausgabe-Zellen **504** Gebrauch und spiegelt einigen dieser Zellen vor, dass sie sich im INTEST-Modus befinden, damit der Logikanalysator **260** über den JTAG-Port **272** gesteuert werden kann.

[0132] Man hat erkannt, dass zahlreiche PLDs zusätzliche Ein/Ausgabe-Zellen **504** und **506** aufweisen, die nicht mit einem Anschlussstift des PLD verbunden sind, jedoch elektrisch an den Kern **502** angeschlossen sind. Vorteilhafterweise hat man in der Erfindung erkannt, dass man diese unverbundenen Ein/Ausgabe-Zellen **504** und **506** nicht nur zum Zuführen von Steuer- und Dateninformation zum Logikanalysator **260** verwenden kann, sondern dass man sie auch dazu verwenden kann, Status- und Ausgabeinformation vom Logikanalysator aufzunehmen. Zudem wird in dieser Ausführungsform nicht der tatsächliche INTEST-Modus verwendet. Damit kann das PLD **16** unter realistischen Umgebungsbedingungen betrieben werden, und der Logikanalysator **260** kann echte Testdaten erfassen. Um nach wie vor Steuer- und Eingabeinformation über unverbundene Ein/Ausgabe-Zellen **504** und die Leitungen **514** zu liefern, spiegelt man diesen Zellen vor, dass sie sich tatsächlich im INTEST-Modus befinden, so dass sie die Steuersignale des Logikanalysators von der Zelle zum Kern **502** weiterleiten. Diese nach innen weitergeleiteten Signale kann man dann dazu verwenden, Steuer- und Eingabedaten für den Logikanalysator **260** zu liefern. Es ist vorteilhaft, dass man zum Implementieren dieses Steuerverfahrens keine zusätzlich Logik benötigt.

[0133] Verwendet man diese Technik, so leiten nur diejenigen Ein/Ausgabe-Zellen **504** nach innen weiter, denen vorgespiegelt wird, dass sie sich im INTEST-Modus befinden. Auf diese Weise werden dem Logikanalysator **260** Steuer- und Datensignale geliefert, wobei man unverbundene Ein/Ausgabe-Zellen **504** verwendet, die mit keinem Anschlussstift des PLD **16** verbunden sind. Da Zellen, die zu nicht benutzten Anschlussstiften des PLD **16** gehören, für die Ein/Ausgabe von Daten und die Steuerung des Logikanalysators **260** verwendet werden, kommunizieren die verbleibenden Anschlussstifte des PLD **16** mit dem Kern **502** und der Außenwelt in einer realistischen Betriebsumgebung und werden durch dieses Steuerverfahren nicht beeinträchtigt.

[0134] Fig. 13 erläutert eine unverbundene Ein/Ausgabe-Zelle **504** gemäß der ersten Ausführungsform der JTAG-Steuerung. In der Zelle **504** ist zusätzlich das Gatter **702** und das Debug-RAM-Bit **704** enthalten. In dieser Ausführungsform versetzt Mode **686** das PLD **16** in seinen normalen Betriebsmodus, so dass der Logikanalysator **260** realistische Daten erfassen kann. In diesem Modus sind die Anschlussstifte des PLD nicht vom Kern **502** abgetrennt. Für unverbundene Ein/Ausgabe-Zellen **504** ist es jedoch nach wie vor erwünscht, dass man den JTAG-Port **272** zum Liefern von Steuersignalen an den eingebetteten Logikanalysator **260** verwenden kann. Hierzu sind das Gatter **702** und das Debug-RAM-Bit **704** bereitgestellt. Das Bit **704** ist stets gesetzt. Daher gibt das Gatter **702** eine logische "1" aus, die den Multiplexer **640** anweist, seine Ausgabedaten immer aus dem Aktualisierungsregister **630** zu holen. Die Daten aus dem Register **630** sind vorher aus dem Erfassungsregister **620** geladen worden, das seine Daten ursprünglich über Seriell-Data-In **672** empfangen hat (nachdem der JTAG-Port **272** veranlasst hat, dass Daten durch die Zellen geschoben werden). Auf diese Weise werden serielle Daten, die der JTAG-Port **272** liefert, schließlich vom Multiplexer **640** ausgegeben und dienen als Eingangssignal **604** für den Kern **502**. Das Eingangssignal **604** kann man dazu verwenden, entweder ein Steuersignal oder ein Dateneingabesignal für den Logikanalysator **260** bereitzustellen. Vorteilhafterweise kann die Vorrichtung im normalen Modus betrieben werden, und alle Anschlussstifte, die mit Ein/Ausgabe-Zellen verbunden sind, bleiben an den Kern **502** angeschlossen. Zudem kann man den Logikanalysator **260** über den JTAG-Port **272** steuern, wobei die JTAG-Sample/Preload-Befehle verwendet werden, die Steuerinformation in den Erfassungsregistern **620**–**624** ablegen.

ZWEITE AUSFÜHRUNGSFORM DER JTAG-STEUERUNG

[0135] Fig. 14 erläutert eine zweite Ausführungsform, über die der JTAG-Port **272** den eingebetteten Logikanalysator **260** mit Hilfe eines Testdatenregisters **802** steuert. In dieser Ausführungsform wird ein vom Benutzer implementiertes Testdatenregister **802** dazu verwendet, dem Logikanalysator **260** Steuersignale zu liefern und Daten und Statusinformation von ihm zu erhalten. Diese Ausführungsform ist insbesondere dann nützlich, wenn keine unverbundenen Ein/Ausgabe-Zellen verfügbar sind. Sie beruht auf zusätzlicher vom Benutzer gelieferter Logik im Testdatenregister **802** und verwendet keine unverbundenen Ein/Ausgabe-Zellen. Zudem stellt diese Ausführungsform ein zusätzliches Signal Runtest(Benutzer) bereit, durch das der Logikanalysator **260** weiß, dass die JTAG-Zustandsmaschine in den Runtest-Status gegangen ist. Das Register **802** enthält jede beliebige Anzahl Anregungszellen **804**, die zum Steuern des Logikanalysators **260** verwendet werden, und jede beliebige Anzahl von Erfassungszellen **805**, die zum Empfangen von Daten- und Statussignalen vom Logikanalysator verwendet werden. Die Steuersignale **806** umfassen das Signal TDI(Benutzer), das an die erste Anregungszelle angelegt und dann durch alle Zellen geschoben wird. Enthalten sind auch die Steuersignale Shift(Benutzer), Clock(Benutzer), Update(Benutzer) und Runtest(Benutzer). Diese Signale werden global an alle Zellen **804** oder **805** angelegt. Das Signal TDO(Benutzer) **807** wird von der letzten Erfassungszelle **805** empfangen und an den JTAG-Port **272** angelegt, damit es zum Signal TDO wird.

[0136] In dieser Ausführungsform gleichen die Steuersignale TDI(Benutzer), Shift(Benutzer), Clock(Benutzer) und Update(Benutzer) den Signalen **672**, **680**, **682** und **684** der in Fig. 13 dargestellten Ausführungsform; diese Steuersignale werden jedoch in der zweiten Ausführungsform in den Kern **502** geleitet und nicht an Ein/Ausgabe-Zellen angelegt. Mit Hilfe dieses ungewöhnlichen Ansatzes, mit dem JTAG-Signale direkt in den Kern geleitet werden, erreicht man in vorteilhafter Weise die Steuerung eines eingebetteten Logikanalysators, ohne dass zusätzliche Anschlussstifte oder Ein/Ausgabe-Zellen des PLD (neben den Anschlussstiften des JTAG-Ports) verwendet werden. Das Signal TDO(Benutzer) entspricht dem Signal **674** der Ausführungsform in Fig. 13; das Signal TDO(Benutzer) stammt jedoch aus dem Kern **502** und nicht aus einer Ein/Ausgabe-Zelle **504**.

[0137] Zum Implementieren dieser Ausführungsform wird die Netzliste des PLD erweitert, um das vom Benutzer implementierte Testdatenregister **802** zuzufügen. Zusätzlich wird jede Eingabe in den und jede Ausgabe aus dem Logikanalysator **260** zu einem Element **804** bzw. **805** geführt. Zum Erzeugen eines Steuersignals, das an den Logikanalysator **260** geliefert wird, wird ein privater "Usertest-Befehl" eingelesen. Anschließend wird die Steuerinformation eingelesen und geladen. Zuletzt werden die Ausga-

bedaten aus dem Logikanalysator ausgelesen. Weitere Einzelheiten werden im Folgenden angegeben.

[0138] Fig. 15 zeigt eine Anregungszelle **804**, die ein Element des Testdatenregisters **802** ist. Die Zelle **804** enthält das Erfassungsregister **820** und das Aktualisierungsregister **822**. Das Scan-In-Signal **824** wird von einer vorhergehenden vergleichbaren Zelle oder vom JTAG-Port **272** empfangen, falls es sich um die erste Anregungszelle handelt. Das Scan-Out-Signal **826** wird an die folgende Anregungszelle übertragen oder an die erste Erfassungszelle **805**, falls es sich um die letzte Anregungszelle handelt. Während eine serielle Verschiebung von Information durch Elemente des Datenregisters **802** erfolgt, kommt die Information über Scan-In **824** an der Zelle **804** an, wird im Register **820** erfasst und über Scan-Out **826** hinausgeschoben. Erfolgt gesteuert durch den JTAG-Port **272** ein paralleler Ladevorgang, so überträgt das Aktualisierungsregister **822** das im Register **820** gespeicherte Bit an den Logikanalysator **260**. Dieses übertragene Bit kann nun als Steuersignal für den Logikanalysator eingesetzt werden.

[0139] Fig. 16 zeigt eine Erfassungszelle **805**, die ein Element des Testdatenregisters **802** ist. Die Zelle **805** enthält einen Multiplexer **830** und ein Erfassungsregister **832**. Das Scan-In-Signal **834** wird von einer vorhergehenden Erfassungszelle oder von der letzten Anregungszelle **804** empfangen, falls es sich um die erste Erfassungszelle handelt. Das Scan-Out-Signal **836** wird an die folgende Erfassungszelle **805** übertragen oder an den JTAG-Port **272**, falls es sich um die letzte Erfassungszelle handelt. Während des seriellen Einlesens von Information über das Testdatenregister **802** hat das Signal Load(Benutzer) den Wert Null. Eingelesene Bits kommen über Scan-In **834** an, werden mit Hilfe des Registers **832** zwischengespeichert und über Scan-Out **836** hinausgeschoben. Während eines parallelen Ladevorgangs (oder eines Erfassungsvorgangs) hat das Signal Load(Benutzer) den Wert Eins. Die Daten und/oder der Status kommen über den Multiplexer **830** an und werden im Register **832** erfasst. Ist nach einem parallelen Ladevorgang eine beliebige Anzahl Bits von den Zellen **805** erfasst, so werden die erfassten Bits mit Hilfe des seriellen Schiebemodus durch den JTAG-Port **272** an das Computersystem **18** hinausgeschoben, damit sie dort untersucht werden. Auf diese Weise setzt man die Erfassungszellen **805** dafür ein, Daten und/oder Statussignale vom Logikanalysator **260** zu holen und dem Benutzer die Information zur Untersuchung darzubieten.

WEITERE AUSFÜHRUNGSFORMEN

[0140] Fig. 17A und Fig. 17B zeigen eine andere Ausführungsform, in der eine beliebige Anzahl von Logikanalysatoren, die in eine Vorrichtung eingebettet sind, mit Hilfe eines JTAG-Ports gesteuert werden. Da PLDs immer umfangreicher werden, kann es sein, dass jede Großfunktion innerhalb der Vorrichtung ihren eigenen eingebetteten Logikanalysator enthalten kann. Es ist erwünscht, dass man jede beliebige Anzahl von eingebetteten Logikanalysatoren über einen JTAG-Port steuern kann, wobei irgendeine der besprochenen Ausführungsformen eingesetzt wird. In einer bestimmten Implementierung arbeitet die in Fig. 14 bis Fig. 16 beschriebene zweite Ausführungsform sehr gut.

[0141] Schweifen wir kurz ab und rufen uns in Erinnerung, dass man die Steuerung eines von zwei eingebetteten Logikanalysatoren erreichen kann, indem man ein Select-Signal verwendet, das der JTAG-Port **272** erzeugt. Bekanntlich kann man private Benutzerbefehle in den JTAG-Port laden. In dieser Ausführungsform kann man einen Benutzer-A-Befehl und einen Benutzer-B-Befehl bereitstellen. Steuerinformation, die für einen ersten Logikanalysator bestimmt ist, wird in den Benutzer-A-Befehl geladen. Steuerinformation, die für einen zweiten Logikanalysator bestimmt ist, wird in den Benutzer-B-Befehl geladen. Wird Benutzer-A geladen, so geht das Signal Select auf high, und wird Benutzer-B geladen, so geht das Signal Select auf low. Das Signal Select wird nun mit den Steuersignalen vom JTAG-Port kombiniert und bewertet diese, so dass sie entweder an ein erstes oder an ein zweites Testdatenregister geleitet werden, das jeweils den ersten bzw. den zweiten eingebetteten Logikanalysator steuert. Bekanntlich kann ein einziges Signal (beispielsweise Select) ein Steuersignal für einen Logikanalysator sperren oder freigeben, und zwar mit einer einfachen Kombination aus UND-Gattern, Invertern usw. Hat beispielsweise Select den Wert logisch "1", so werden die Steuersignale zum ersten Logikanalysator geleitet, und Ausgangssignale werden von ihm empfangen. Der zweite Logikanalysator wird gewählt, wenn Select den Wert logisch "0" hat. Sollen mehr als zwei Logikanalysatoren gesteuert werden, so verwendet man zweckmäßigerweise eine Ausführungsform, die nun beschrieben wird.

[0142] Fig. 17A zeigt eine Ausführungsform, in der die in einem einzigen PLD vorhandenen Logikanalysatoren **260a**, **260b**, **260c** und **260d** über einen JTAG-Port des PLD gesteuert werden. Obwohl Fig. 17A ein vereinfachtes Beispiel darstellt, in dem lediglich vier Logikanalysatoren gesteuert werden, die äußerst einfache Eingänge und Ausgänge haben, sehen Fachleute bei der Lektüre dieser Offenlegung sofort, wie man die Steuerung einer beliebigen Anzahl von Logikanalysatoren implementiert und auch die Steuerung von Logikanalysatoren, die kompliziertere Eingänge und Ausgänge aufweisen.

[0143] Jeder Logikanalysator **260a**–**260d** wird von einem entsprechenden Testdatenregister **860**–**866**

gesteuert. Der Logikanalysator **260a** wird beispielsweise vom Testdatenregister **860** gesteuert. Elemente **860a** und **860b** des Registers **860** liefern Steuereingangssignale an den Logikanalysator **260a**, und Elemente **860c** und **860d** des Registers **860** empfangen Ausgabedaten vom Logikanalysator **260a**. Die anderen Logikanalysatoren werden in vergleichbarer Weise gesteuert.

[0144] In dieser Ausführungsform werden zwei private Benutzerbefehle innerhalb des JTAG-Ports dazu verwendet, eine beliebige Anzahl Logikanalysatoren zu steuern. Der Benutzer-A-Befehl enthält eine Adresse, die sich dafür eignet, auszuwählen, welcher Logikanalysator gesteuert wird und vom welchem Logikanalysator zu einem bestimmten Zeitpunkt Daten empfangen werden. Der zweite private Befehl mit dem Namen Benutzer-B liefert die tatsächliche Steuerinformation und empfängt Daten von dem Logikanalysator, der mit Hilfe des Benutzer-A-Befehls gewählt wird. Das Select-Signal wird auch in dieser Ausführungsform verwendet, und es ist abhängig davon aktiv, ob auf Information im Benutzer-A-Befehl oder im Benutzer-B-Befehl zugegriffen wird.

[0145] Das Testdatenregister **870** enthält zwei Elemente A1 und A0, über die ein Benutzer-A-Befehl hineingeschoben bzw. hinausgeschoben wird. Nicht dargestellt ist ein Signal Clock(Benutzer-A), das an jedes Element angelegt wird. Das Signal TDO(Benutzer-A) **892** führt zum JTAG-Port **272** zurück. Auf diese Weise liefert ein Benutzer-A-Befehl durch das Register **870** Eingaben an einen Decoder **872** und einen Multiplexer **874**. Der Decoder **872** decodiert die beiden Bits des Registers **870** und liefert vier Ausgangssignale D3–D0, die zum Steuern der UND-Gatter **882**–**888** dienen. Die beiden Bits des Registers **870** werden auch in den Multiplexer **874** eingegeben und wählen einen der vier Logikanalysatoren für die Ausgabe. In diesem vereinfachten Beispiel besitzt jeder Logikanalysator nur zwei Eingänge und zwei Ausgänge, und das Register **870** ist nur zwei Byte breit. Natürlich kann jedes dieser Register jede beliebige Größe aufweisen, und es können auch mehr Logikanalysatoren vorhanden sein.

[0146] Wie bereits erwähnt dient ein zweiter privater Benutzer-B-Befehl dazu, den mit dem Benutzer-A-Befehl gewählten Logikanalysator die tatsächliche Steuerinformation zu liefern. Ähnlich wie in der zweiten bereits beschriebenen Ausführungsform wird das Signal TDI(Benutzer-B) **876** seriell jedem der Testregister **860**–**866** zugeführt, die jeweils die Logikanalysatoren **260a**–**260d** steuern. Die mit dem Signal **876** empfangene serielle Information wird durch das passende Testdatenregister geschoben und schließlich durch den Multiplexer **874** hinausgeschoben, damit sie zum Signal TDO(Benutzer-B) **890** wird, das zum JTAG-Port zurückgeführt wird. Soll beispielsweise der Logikanalysator **260d** gewählt werden, so wird Steuerinformation über das Signal **876** zum Element **866a** geschoben. Anschließend wird sie zum Element **866b** geschoben und dann zum Element **866c** geschoben. Zuletzt wird sie zum Element **866d** geschoben. In ähnlicher Weise werden die vom Logikanalysator **260d** empfangenen Ausgabedaten geholt, indem man einen parallelen Ladevorgang in die Elemente **866c** und **866d** verwendet und anschließend zum Multiplexer **874** hinausschiebt.

[0147] Wie bereits beschrieben empfangen die Elemente in einem bestimmten Testdatenregister zahlreiche Steuersignale, etwa Shift(Benutzer), Clock(Benutzer), Update(Benutzer) und Runtest(Benutzer). Diese Steuersignale werden mit Hilfe des Decoders **872** und der UND-Gatter **882**–**888** zum gewählten Logikanalysator geleitet. Wie erwähnt kann man die Signale Clock(Benutzer-B) **880** und Clock(Benutzer-A) (nicht dargestellt) aus dem Signal Clock(Benutzer) mit Hilfe von UND-Gattern und dem Signal Select erzeugen. Ist Select aktiv, so taktet Clock(Benutzer-A), und Clock(Benutzer-B) tut dies nicht. Ein inaktives Select erzeugt die umgekehrte Wirkung. Wird auf diese Weise Information vom privaten Benutzer-A-Befehl zugeführt, so ist Select aktiv und gibt die zugehörigen Steuersignale frei.

[0148] In diesem vereinfachten Beispiel ist nur das Signal Clock(Benutzer-B) **880** mit dem gewählten Logikanalysator verbunden dargestellt. Fachleuten ist jedoch geläufig, wie man die anderen Steuersignale in ähnlicher Weise ebenfalls an den gewählten Logikanalysator führen kann. Das Ausgangssignal des Decoders **872** gibt zu einem Zeitpunkt nur einen von vier Ausgängen D3–D0 frei, damit eines der UND-Gatter **882**–**888** gewählt wird. Damit wird das Signal **880** zu einem gegebenen Zeitpunkt nur an das Testdatenregister für den gewählten Logikanalysator geleitet. Enthält das Register **870** beispielsweise den Wert "11", so ist der Ausgang D3 des Decoders **872** aktiv, und die anderen Ausgangssignale haben den Wert low. Damit lässt nur das Gatter **888** das Signal **880** zum Testdatenregister **860** durch, damit der Logikanalysator **260a** gesteuert wird. Die anderen Gatter **882**–**886** lassen das Signal **880** nicht durch. Man kann ähnliche Formen der Auswahl und Steuerung dazu verwenden, die Steuersignale zum gewählten Logikanalysator zu leiten.

[0149] Fig. 17B erläutert eine Vorgehensweise, mit der Signale TDO(Benutzer-A) **892** und TDO(Benutzer-B) **890** aus dem Register **870** und dem Multiplexer **874** zum JTAG-Port **272** zurückgeführt werden. Die Signale **890** und **892** werden in den Multiplexer **894** eingegeben; eines der Signale wird mit Hilfe des Signals Select **896** gewählt. Wie bereits erwähnt kann man Select dazu verwenden, Steuersignale für einen Logikanalysator oder Signale von einem Logikanalysator zu bewerten und/oder auszuwählen. Wird in diesem Beispiel Information durch das Regis-

ter **870** geschoben, so ist Select aktiv (dadurch wird auch Clock(Benutzer-A) freigegeben). Das Signal **892** läuft durch den Multiplexer **894** zum Port **272**. Das Signal **890** wird gewählt, falls Select den Wert low hat. In dieser Weise wird das passende Ausgangssignal der Ausführungsform **850** an den JTAG-Port **272** geliefert, und es wird schließlich zum einzigen Signal TDO des JTAG-Ports.

[0150] Man kann auch vergleichbare Ausführungsformen zum Steuern zahlreicher Logikanalysatoren verwenden. Anstatt unterschiedliche Testregister (beispielsweise die Register **860**–**866**) für jeden Logikanalysator zu verwenden kann nur ein Testdatenregister vorhanden sein, das alle Logikanalysatoren bedient. Die verwendet Auswahllogik (beispielsweise die UND-Gatter und die Multiplexer in Fig. 17A) wird dann zwischen den Logikanalysatoren und dem einzigen Testdatenregister angeordnet und nicht außerhalb der Logikanalysatoren, wie dies in der Ausführungsform in Fig. 17A dargestellt ist.

[0151] Beispielsweise werden Eingabesteuersignale (wie Clock(Benutzer), Shift(Benutzer) usw.) vom JTAG-Port direkt mit den Elementen des einzigen Testdatenregisters verbunden. Die Daten des Benutzer-B-Befehls werden in dieses Testdatenregister geschoben, damit sie einen der Logikanalysatoren steuern. Jedes Eingabeelement des Testdatenregisters wird zu jedem Logikanalysator weitergeleitet und dort mit einem UND-Gatter oder einer ähnlichen Auswahlschaltung bewertet. Sind beispielsweise drei Bit Steuerinformation von dem einzigen Testdatenregister an vier verschiedene Logikanalysatoren zu liefern, so sind vier UND-Gatter für jedes Bit Steuerinformation vorhanden (die jeweils zu einem Logikanalysator gehören), so dass man insgesamt zwölf UND-Gatter (oder eine ähnliche Auswahllogik) benötigt. Um ein Bit Steuerinformation an einen gewählten Logikanalysator zu liefern kann man eine Auswahllogik verwenden, die dem Register **870** und dem Decoder **872** gleicht. Anders ausgedrückt kann man den privaten Benutzer-A-Befehl dann dazu verwenden, auszuwählen, welcher Logikanalysator gesteuert wird. Die decodierte Information aus diesem Benutzer-A-Befehl kann man dann dazu verwenden, die UND-Gatter für jedes Bit Steuerinformation auszuwählen und/oder zu sperren. Auf diese Weise wird es einem einzigen Bit an Steuerinformation ermöglicht, einen ausgewählten Logikanalysator zu erreichen, die anderen Logikanalysatoren jedoch nicht.

[0152] In ähnlicher Weise kann man Multiplexer dazu verwenden, zu wählen, von welchen Logikanalysatoren ein Ausgangssignal für die Eingabe in das einzige Testdatenregister akzeptiert wird. Beispielsweise kann man einen einzigen Multiplexer (oder eine ähnliche Auswahllogik) für jedes Element des Testdatenregisters verwenden, das Ausgabedaten vom Logikanalysator empfängt. An diesen Multiplexer werden die entsprechenden Datensignale eines jeden Logikanalysators geführt. Beispielsweise wird das geringstwertige Datenbit von jedem Logikanalysator an einen einzigen Multiplexer geführt, der das Ausgangssignal für ein einziges Element des Testdatenregisters liefert. Der Multiplexer wird über Bits aus dem Benutzer-A-Testdatenregister in ähnlicher Weise wie in Fig. 17A dargestellt gesteuert. Ein Multiplexer wird für jedes Ausgabeelement des Testdatenregisters vorgesehen und jeweils mit der gleichen Auswahllogik gesteuert.

[0153] Weitere Ausführungsformen sind möglich. Man kann die oben in Fig. 11–Fig. 13 vorgestellte erste Ausführungsform oder die in Fig. 14–Fig. 16 vorgestellte zweite Ausführungsform ausschließlich allein zum Steuern eines Logikanalysators verwenden, oder man kann sie für die Steuerung kombinieren. Ist eine ausreichende Anzahl zusätzlicher unverbundener Ein/Ausgabe-Zellen verfügbar, so kann es erwünscht sein, ausschließlich die erste Ausführungsform einzusetzen. Dies trifft insbesondere dann zu, wenn es schwierig wäre, zusätzliche Logik in die Vorrichtung einzufügen. Sind zu wenig Ein/Ausgabe-Zellen verfügbar, so kann es erwünscht sein, die zweite Ausführungsform einzusetzen, solange das Zufügen der zusätzlichen Logik, die für das Testdatenregister **802** erforderlich ist, keine Schwierigkeit bereitet. Verwendet man ausschließlich die erste Ausführungsform, so kann man dem eingebetteten Logikanalysator ein Clock-Signal liefern, indem man das Eingangssignal **604** nutzt. Zum Bereitstellen dieses Clock-Signals werden abwechselnde Einsen und Nullen in eine bestimmte Ein/Ausgabe-Zelle geschoben. Anschließend wird ein Wert je Zeiteinheit geladen, um einen abwechselnden Impuls bereitzustellen. Jedes neue Bit muss somit durch einen gesamten Registersatz eingelesen werden, bevor es als Clock-Signal dienen kann. Aus diesem Grund ist die Vorgehensweise des Bereitstellens eines Clock-Signals für den eingebetteten Logikanalysator mit Hilfe der ersten Ausführungsform nicht besonders effizient.

[0154] Für eine bessere Lösung verwendet man eine Kombination der ersten und der zweiten Ausführungsform. In dieser Lösung wird ein in der zweiten Ausführungsform verfügbares zusätzliches Signal Runtest(Benutzer) dazu verwendet, ein Clock-Signal für den eingebetteten Logikanalysator bereitzustellen. Beim Übergang dieses Clock-Signals wird der Logikanalysator angewiesen, die Steuersignale zu beachten, die von den Eingangssignalen **604** der verschiedenen Ein/Ausgabe-Zellen **504** ankommen, die mit Hilfe der ersten Ausführungsform implementiert werden. Dass das Signal Runtest(Benutzer) Taktimpulse liefert, kann man einfach dadurch erreichen, dass man den JTAG-Port **272** abwechselnd in diesen Status versetzt und ihn anschließend verlässt. Mit Hilfe dieser Vorgehensweise stellt man eine wirksa-

mere Steuerung bereit; zusätzliche unverbundene Ein/Ausgabe-Zellen können trotzdem dazu verwendet werden, dem Logikanalysator die tatsächliche Steuerinformation zu liefern.

AUSFÜHRUNGSFORM DES COMPUTERSYSTEMS

[0155] Fig. 18 zeigt ein Computersystem **900**.

[0156] Das Computersystem **900** enthält eine beliebige Anzahl Prozessoren **902** (die auch als Zentraleinheiten oder CPUs bezeichnet werden), die mit Speichervorrichtungen verbunden sind. Hierzu gehört der Hauptspeicher **906** (beispielsweise ein Speicher mit wahlfreiem Zugriff, RAM) und der Hauptspeicher **904** (beispielsweise ein Festwertspeicher, ROM). Bekanntlich dient der Hauptspeicher **904** dazu, Daten und Befehle in einer Richtung an die CPU zu übertragen. Der Hauptspeicher **906** dient in der Regel dazu, Daten und Befehle in zwei Richtungen zu übertragen. Jede dieser Hauptspeichervorrichtungen kann irgendein beliebiges computerlesbares Medium enthalten, das unten beschrieben wird. Eine Massenspeichervorrichtung **908** ist ebenfalls bidirektional mit der CPU **902** verbunden und liefert eine zusätzliche Datenspeicherkapazität. Sie kann ebenfalls irgendeines der unten beschriebenen computerlesbaren Medien enthalten. Man kann die Massenspeichervorrichtung **908** zum Speichern von Programmen, Daten usw. verwenden. Sie ist in der Regel ein Hilfsspeichermedium (beispielsweise eine Festplatte), das langsamer ist als der Hauptspeicher. Natürlich kann die in der Massenspeichervorrichtung **908** gehaltene Information in geeigneten Fällen in üblicher Weise als virtueller Speicher in den Hauptspeicher **906** aufgenommen werden. Eine besondere Massenspeichervorrichtung, beispielsweise die CD-ROM **914**, leitet Daten unidirektional zur CPU.

[0157] Die CPU **902** ist auch mit einer Schnittstelle **910** verbunden, die eine oder mehrere Ein/Ausgabe-Vorrichtungen enthält, beispielsweise Videomonitore, Rollkugeln, Mäuse, Tastaturen, Mikrophone, Berührbildschirme, Aufnehmer-Kartenleser, Magnetband- oder Lochstreifenleser, Tablets, Stifte, Sprach- oder Handschrifterkenner, Biometrie-Leser oder weitere Computer. Die CPU **902** kann wahlweise mit einem weiteren Computer oder einem Telekommunikationsnetz verbunden sein, und zwar über einen Netzanschluss, der allgemein bei **912** dargestellt ist. Mit Hilfe eines solchen Netzanschlusses wird in Betracht gezogen, dass die CPU im Verlauf des Ausführens der oben beschriebenen Verfahrensschritte Information aus dem Netz empfangen kann oder dass sie Information in das Netz ausgeben kann.

[0158] Zudem beziehen sich Ausführungsformen ferner auf Computerspeicherprodukte mit einem computerlesbaren Medium, auf dem sich Programmcode befindet, der verschiedene computerimplementierte Operationen ausführen kann. Das Medium und der Programmcode können speziell für die Zwecke der Erfindung entworfen und konstruiert sein. Es kann sich auch um Medien und Programmcodes handeln, die Computersoftware-Fachleuten bekannt und greifbar sind. Beispiele für computerlesbare Medien sind (ohne Einschränkung hierauf): magnetische Medien, etwa Festplatten, Floppy-Disks und Magnetband; optische Medien, etwa CD-ROM-Disks; magneto-optische Medien, etwa Floptical-Disks; und Hardwarevorrichtungen, die besonders für das Speichern und Ausführen von Programmcode konfiguriert sind, beispielsweise ROM- und RAM-Vorrichtungen. Beispiele für Programmcode umschließen sowohl Maschinencode, den beispielsweise ein Übersetzer erzeugt, als auch Dateien mit Code auf höheren Sprachebenen, die der Computer mit Hilfe eines Interpreters ausführen kann.

[0159] Die obige Erfindung wurde zur Förderung des Verständnisses einigermaßen detailliert beschrieben. Selbstverständlich kann man innerhalb des Bereichs der beigefügten Ansprüche gewisse Änderungen und Abwandlungen vornehmen. Man kann beispielsweise einen Logikanalysator in irgendeine geeignete Vorrichtung oder Schaltungsplatine einbetten, die sich programmieren lässt. Zudem ist die Erfindung auf jede Art von EDA-Werkzeugen anwendbar, die einen Benutzerentwurf übersetzen kann. Obwohl nur ein Beispiel für die Übersetzung eines Logikanalysators vorgestellt wird, kann man abhängig von der Vorrichtung, für die der Entwurf übersetzt wird, Änderungen an der Übersetzungstechnik vornehmen und nach wie vor Vorteile aus der Erfindung ziehen. Zudem dient der dargestellte besondere Logikanalysator nur als Beispiel; man kann auch andere Schaltungen zum Implementieren eines Logikanalysators verwenden. Eine Schnittstelle von einem Computer zu dem Logikanalysator kann jede beliebige Anzahl Anschlussstifte und jede beliebige Protokollart verwenden, beispielsweise seriell, parallel usw. Ein JTAG-Port kann einen oder mehrere eingebettete Logikanalysatoren steuern und dabei entweder die beschriebene erste oder die zweite Ausführungsform oder eine Kombination der Ausführungsformen verwenden. Daher sollten die beschriebenen Ausführungsformen als erläuternd und nicht als einschränkend angesehen werden. Die Erfindung sollte nicht auf die hier angegebenen Einzelheiten eingeschränkt werden, sondern sie wird durch die folgenden Ansprüche und deren vollständigen Gleichwertigkeitsbereich bestimmt.

Patentansprüche

1. Programmierbare Logikvorrichtung PLD (**16**), umfassend:
einen PLD-Schaltkreis, der eine Iteration eines elektronischen Aufbaus in einem Entwurfsprozess zur

Schaffung einer endgültigen PLD darstellt;
einen Logikanalysator-Schaltkreis (**260**), der so in den PLD-Schaltkreis integriert ist, dass ein Teil des PLD-Schaltkreises mit der Logikanalysatorschaltung (**260**) verbunden ist,
gekennzeichnet durch
einen JTAG-Port (**272**), der so eingerichtet ist, dass er Logikanalysatorbefehle von außerhalb der PLD empfängt; und
eine Kontrollvorrichtung (**274**) für die Logikanalysatorschaltung (**260**), die den JTAG-Port (**272**) der PLD (**16**) verwendet, wobei die Logikanalysatorschaltung (**260**) die Befehle von der PLD (**16**) empfängt und demgemäß arbeitet.

2. PLD (**16**) nach Anspruch 1, wobei die Vorrichtung (**274**) enthält:
nicht gebondete Ein/Ausgabe-Zellen (**504**, **506**), über die Signale zwischen dem JTAG-Port (**272**) und der Logikanalysatorschaltung (**260**) laufen.

3. PLD (**16**) nach Anspruch 1, wobei die Vorrichtung (**274**) enthält:
ein Testdatenregister (**802**), das im Kern der PLD (**16**) implementiert ist, und über das Signale zwischen dem JTAG-Port (**272**) und der Logikanalysatorschaltung (**260**) laufen.

4. PLD (**16**) nach Anspruch 1, zudem umfassend:
zahlreiche Logikanalysatoren (**260**); und
ein Mittel zum Auswählen eines der Logikanalysatoren (**260**), wobei die Kontrollvorrichtung (**274**) den gewählten Logikanalysator (**260**) steuert und der gewählte Logikanalysator (**260**) die Befehle von außerhalb der PLD (**16**) empfängt und demgemäß arbeitet.

5. PLD (**16**) nach Anspruch 1, wobei die Kontrollvorrichtung (**274**) umfasst:
zahlreiche erste Ein/Ausgabe-Zellen (**504**), die so angeordnet sind, dass sie serielle Daten von dem JTAG-Port (**272**) empfangen und diese seriellen Daten an die Logikanalysatorschaltung (**260**) liefern; und
zahlreiche zweite Ein/Ausgabe-Zellen (**506**), die so angeordnet sind, dass sie erfasste Daten von der Logikanalysatorschaltung (**260**) empfangen und diese erfassten Daten seriell an den JTAG-Port (**272**) liefern, wobei die Logikanalysatorschaltung (**260**) die Befehle von außerhalb der PLD (**16**) empfängt und die erfassten Daten an den JTAG-Port liefert.

6. PLD (**16**) nach Anspruch 5, wobei die ersten Ein/Ausgabe-Zellen (**504**) so angeordnet sind, dass sie die seriellen Daten in den Kern der PLD (**16**) treiben, wenn sich die PLD in einer normalen Arbeitsumgebung befindet.

7. PLD (**16**) nach Anspruch 1, wobei die Kontrollvorrichtung (**274**) umfasst:
ein Testdatenregister (**802**), das im Kern der PLD (**16**) so implementiert ist, dass es serielle Daten vom JTAG-Port (**272**) an die Logikanalysatorschaltung (**260**) liefert, und dass es von der Logikanalysatorschaltung (**260**) erfasste Daten seriell an den JTAG-Port (**272**) liefert, wobei die Logikanalysatorschaltung (**260**) die Befehle von außerhalb der PLD (**16**) empfängt und die erfassten Daten an den JTAG-Port (**272**) liefert.

8. PLD (**16**) nach Anspruch 7, wobei das Testdatenregister (**802**) enthält:
zahlreiche Anregungszellen (**804**), die die seriellen Daten an die Logikanalysatorschaltung (**260**) liefern; und
zahlreiche Erfassungszellen (**805**), die die erfassten Daten von der Logikanalysatorschaltung (**260**) empfangen.

9. Verfahren zur Fehlersuche in einer programmierbaren Logikvorrichtung (PLD), umfassend:
das Zusammenstellen eines elektronischen Aufbaus und das Einsetzen eines Logikanalysators, damit eine vollständige Entwurfsdatei erzeugt wird;
das Programmieren der PLD mit der vollständigen Entwurfsdatei, wobei der Logikanalysator in die PLD eingebettet ist,
gekennzeichnet durch
das Verbinden eines JTAG-Ports der PLD mit dem Logikanalysator; und
das Steuern des eingebetteten Logikanalysators mit Hilfe des JTAG-Ports, wobei Fehler in der PLD gesucht werden können.

10. Verfahren nach Anspruch 9, ferner umfassend:
das Liefern serieller Daten vom JTAG-Port zu ersten Ein/Ausgabe-Zellen der PLD, wobei die ersten Ein/Ausgabe-Zellen so angeordnet sind, dass sie die seriellen Daten in den Logikanalysator laden; und
das Empfangen erfasster Daten von dem Logikanalysator in zweite Ein/Ausgabe-Zellen, wobei die zweiten Ein/Ausgabe-Zellen so angeordnet sind, dass sie die erfassten Daten an den JTAG-Port liefern.

11. Verfahren nach Anspruch 9, weiterhin umfassend:
das Ausbilden eines Testdatenregisters im Kern der PLD, über das Signale zwischen dem JTAG-Port und dem Logikanalysator seriell laufen können.

Es folgen 15 Blatt Zeichnungen

Anhängende Zeichnungen

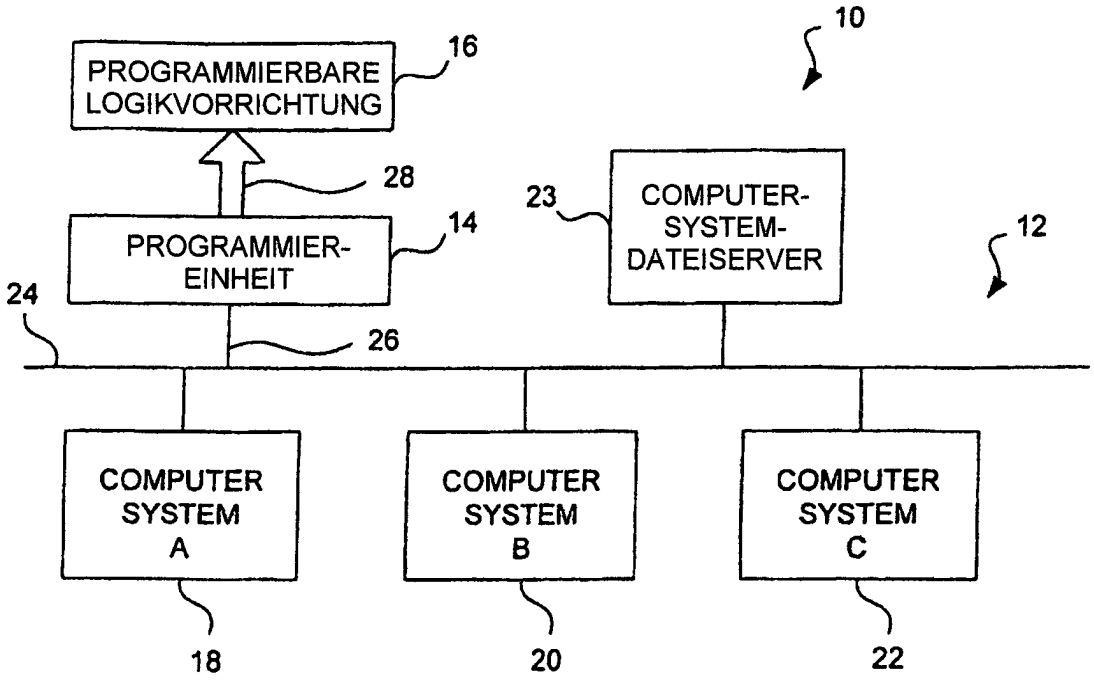

FIG. 1

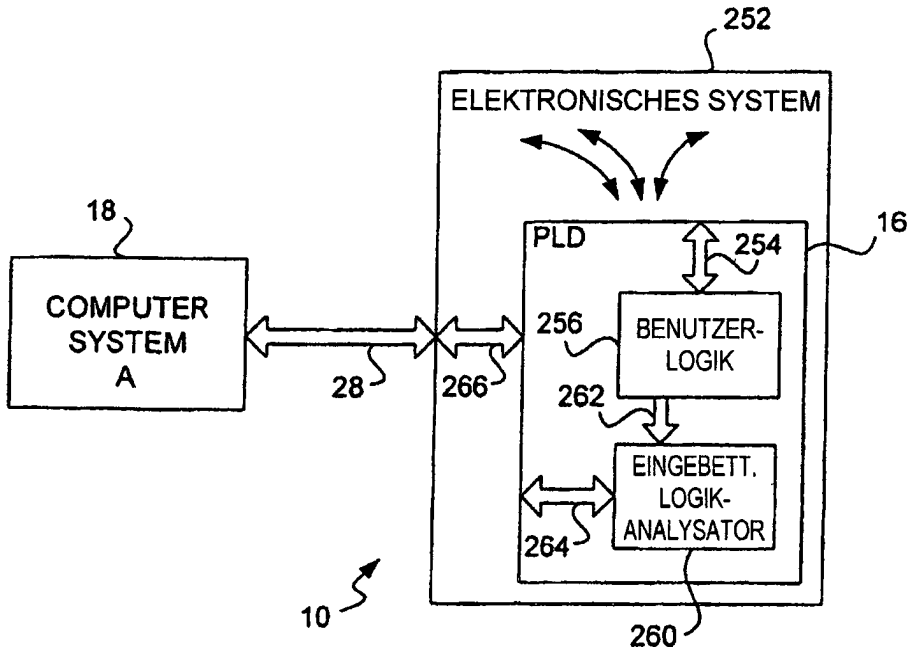

FIG. 5

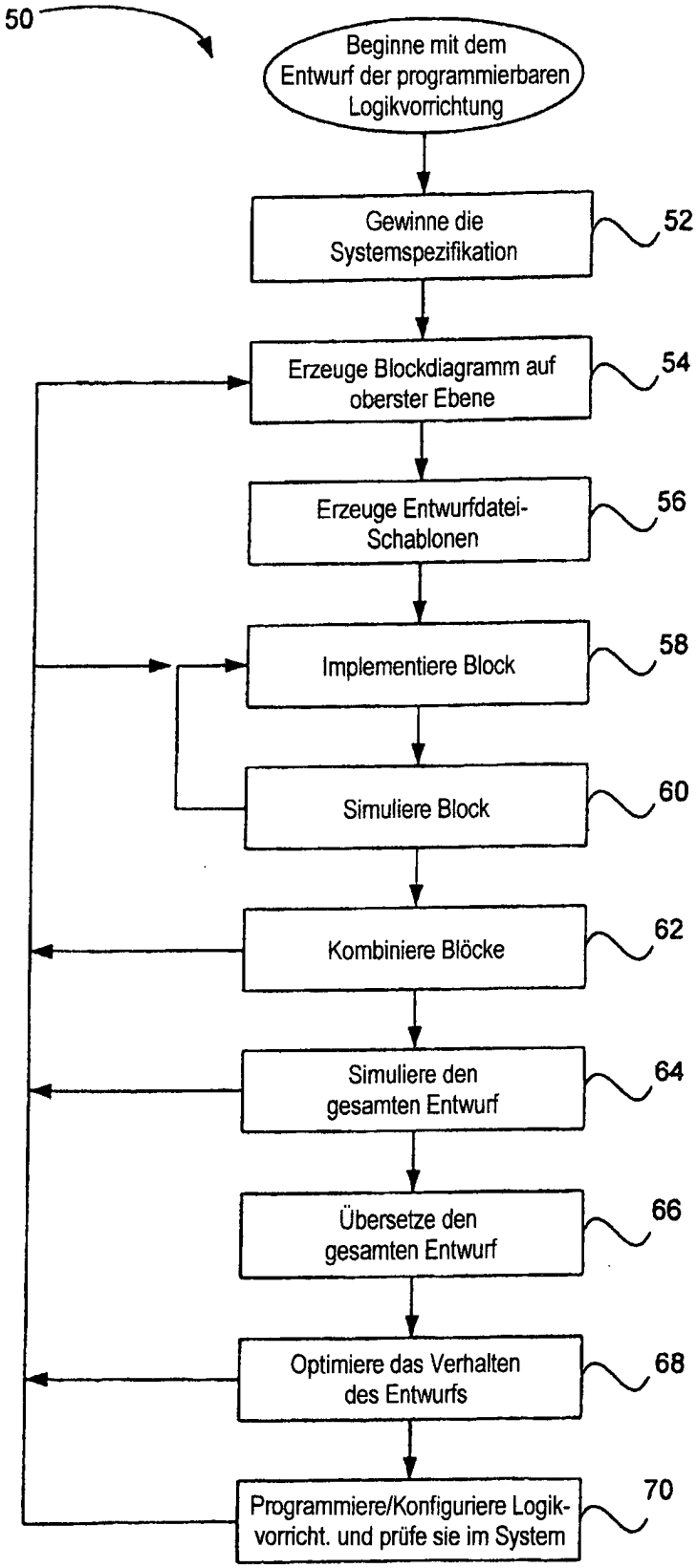
50
Beginne mit dem Entwurf der programmierbaren Logikvorrichtung
Gewinne die Systemspezifikation
52
Erzeuge Blockdiagramm auf oberster Ebene
54
Erzeuge Entwurfdatei-Schablonen
56
Implementiere Block
58
Simuliere Block
60
Kombiniere Blöcke
62
Simuliere den gesamten Entwurf
64
Übersetze den gesamten Entwurf
66
Optimiere das Verhalten des Entwurfs
68
Programmiere/Konfiguriere Logik-vorricht. und prüfe sie im System
70

FIG. 2

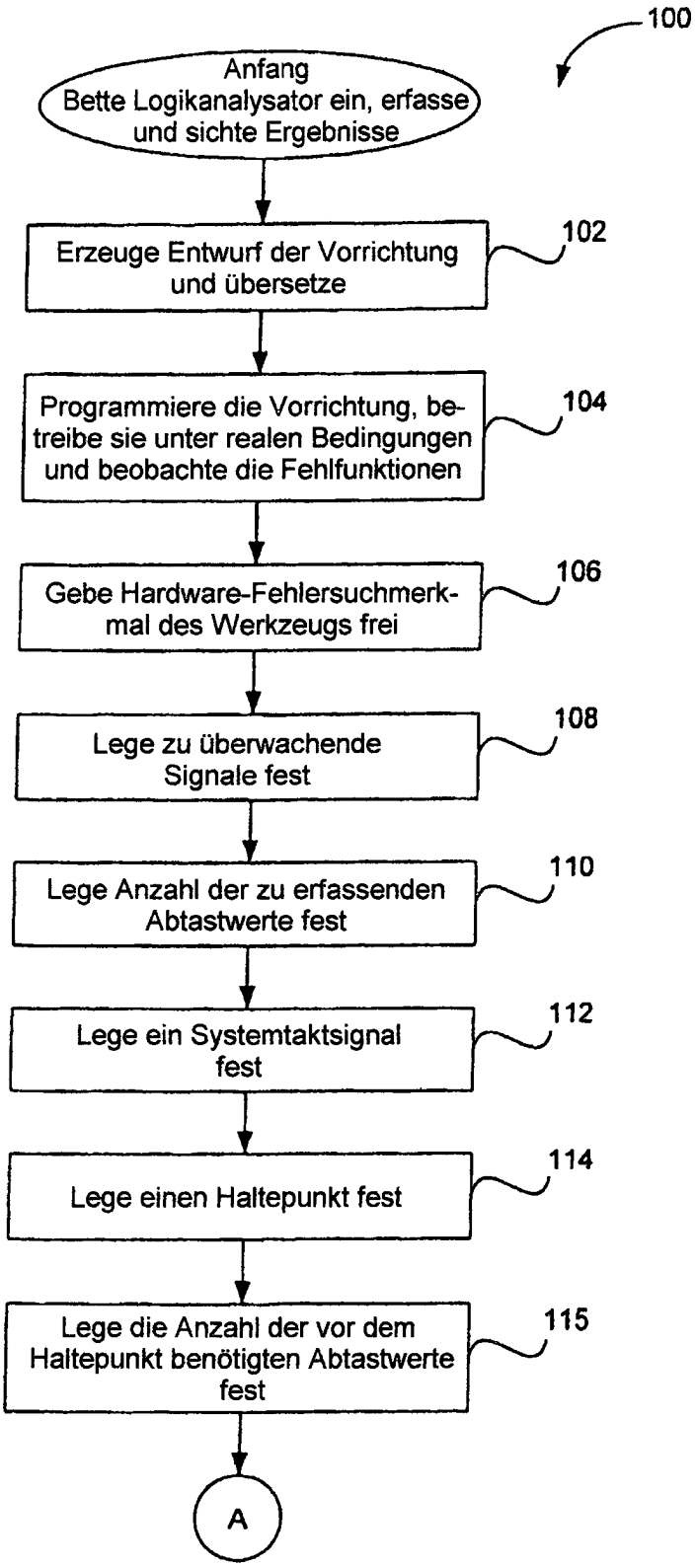
100
Anfang
Bette Logikanalysator ein, erfasse und sichte Ergebnisse
Erzeuge Entwurf der Vorrichtung und übersetze
102
Programmiere die Vorrichtung, betreibe sie unter realen Bedingungen und beobachte die Fehlfunktionen
104
Gebe Hardware-Fehlersuchmerkmal des Werkzeugs frei
106
Lege zu überwachende Signale fest
108
Lege Anzahl der zu erfassenden Abtastwerte fest
110
Lege ein Systemtaktsignal fest
112
Lege einen Haltepunkt fest
114
Lege die Anzahl der vor dem Haltepunkt benötigten Abtastwerte fest
115
A

FIG. 3A

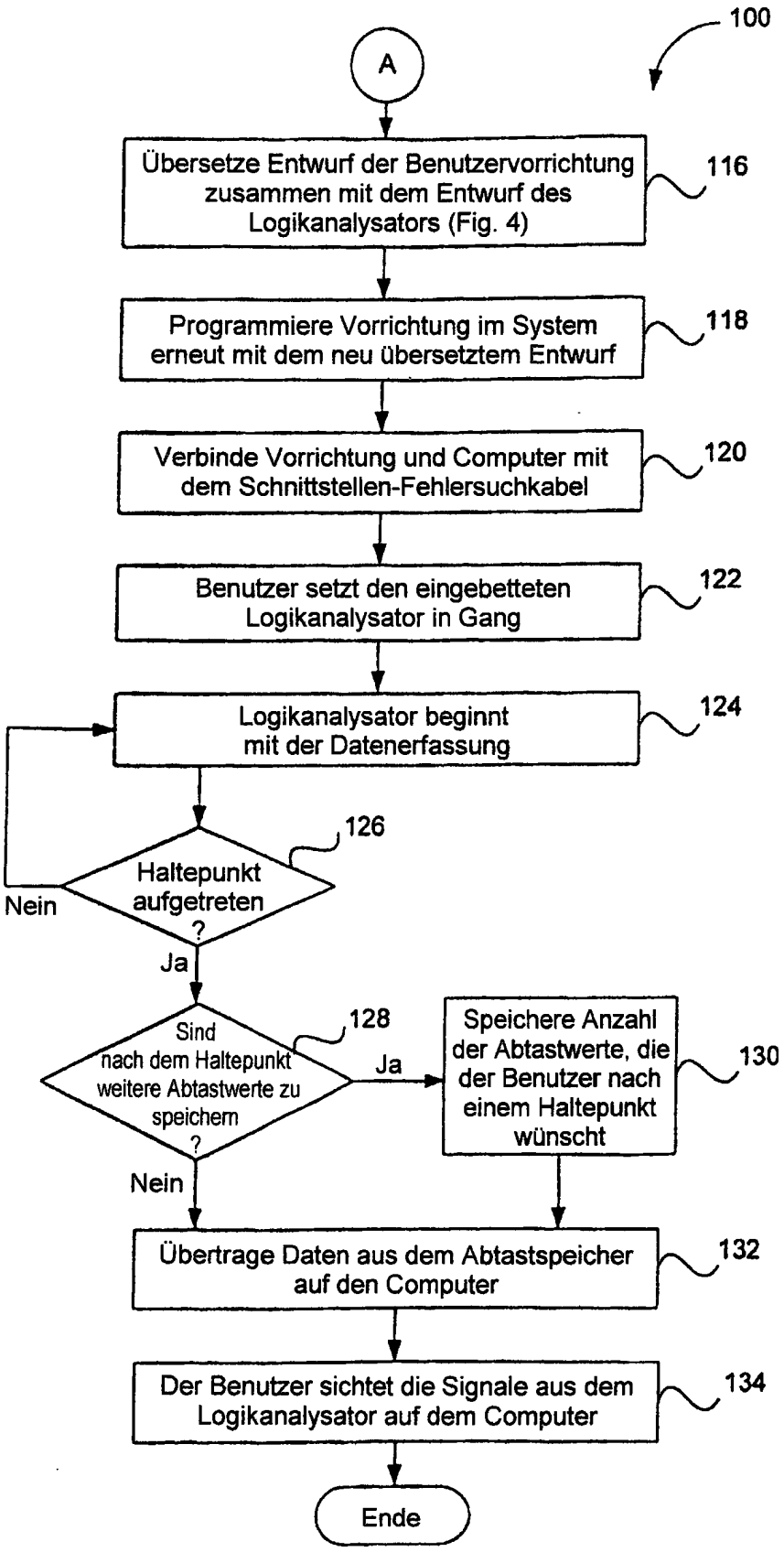
100
A
Übersetze Entwurf der Benutzervorrichtung zusammen mit dem Entwurf des Logikanalysators (Fig. 4)
116
Programmiere Vorrichtung im System erneut mit dem neu übersetztem Entwurf
118
Verbinde Vorrichtung und Computer mit dem Schnittstellen-Fehlersuchkabel
120
Benutzer setzt den eingebetteten Logikanalysator in Gang
122
Logikanalysator beginnt mit der Datenerfassung
124
Haltepunkt aufgetreten ?
126
Nein
Ja
Sind nach dem Haltepunkt weitere Abtastwerte zu speichern ?
128
Ja
Speichere Anzahl der Abtastwerte, die der Benutzer nach einem Haltepunkt wünscht
130
Nein
Übertrage Daten aus dem Abtastspeicher auf den Computer
132
Der Benutzer sichtet die Signale aus dem Logikanalysator auf dem Computer
134
Ende

FIG. 3B

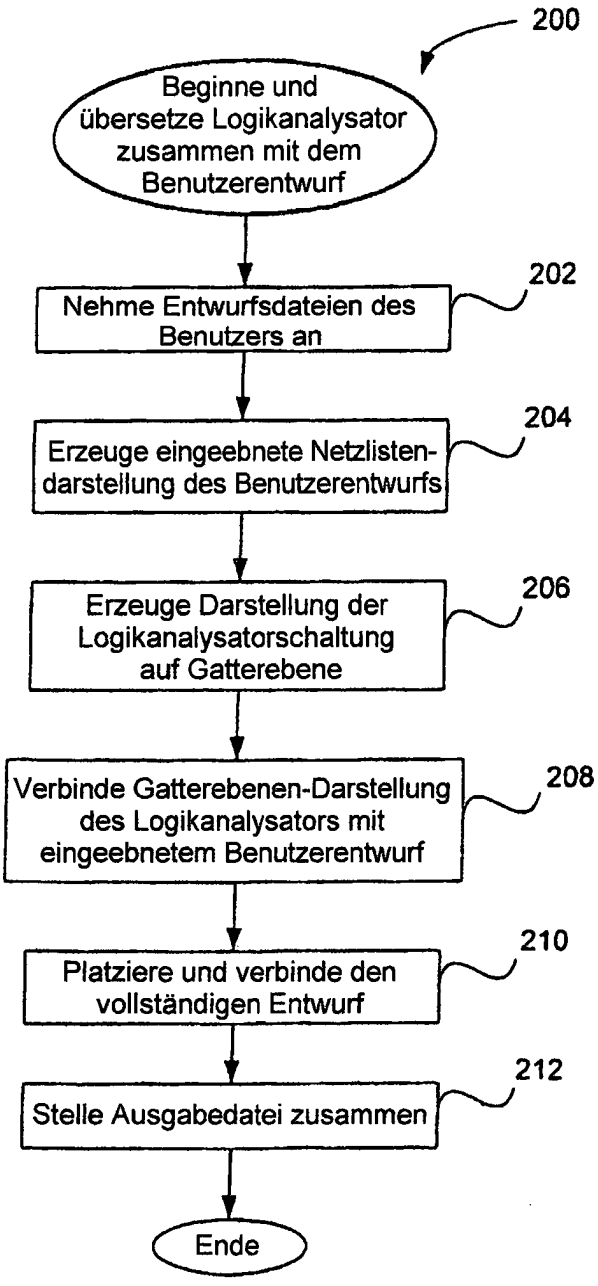
200
Beginne und übersetze Logikanalysator zusammen mit dem Benutzerentwurf
202
Nehme Entwurfsdateien des Benutzers an
204
Erzeuge eingeebnete Netzlisten-darstellung des Benutzerentwurfs
206
Erzeuge Darstellung der Logikanalysatorschaltung auf Gatterebene
208
Verbinde Gatterebenen-Darstellung des Logikanalysators mit eingeebnetem Benutzerentwurf
210
Platziere und verbinde den vollständigen Entwurf
212
Stelle Ausgabedatei zusammen
Ende

FIG. 4

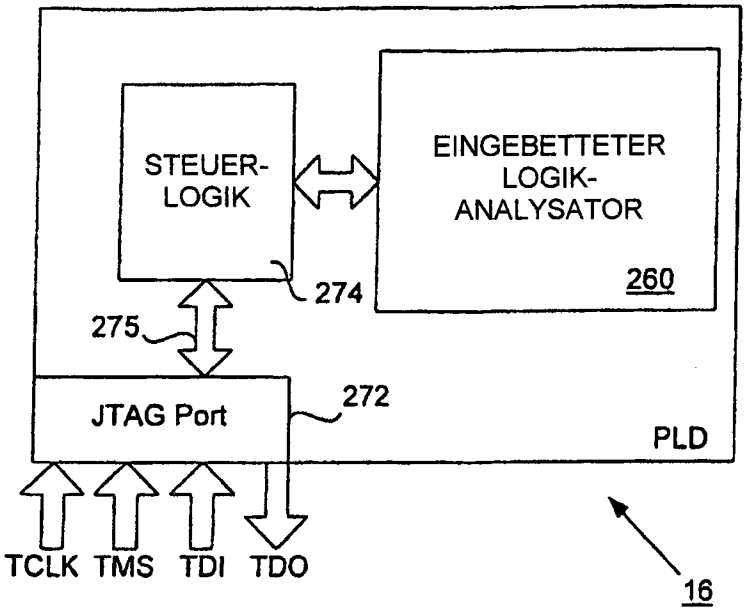
STEUER-
LOGIK
EINGEBETTETER
LOGIK-
ANALYSATOR
260
274
275
JTAG Port
272
PLD
TCLK
TMS
TDI
TDO
16

FIG. 6

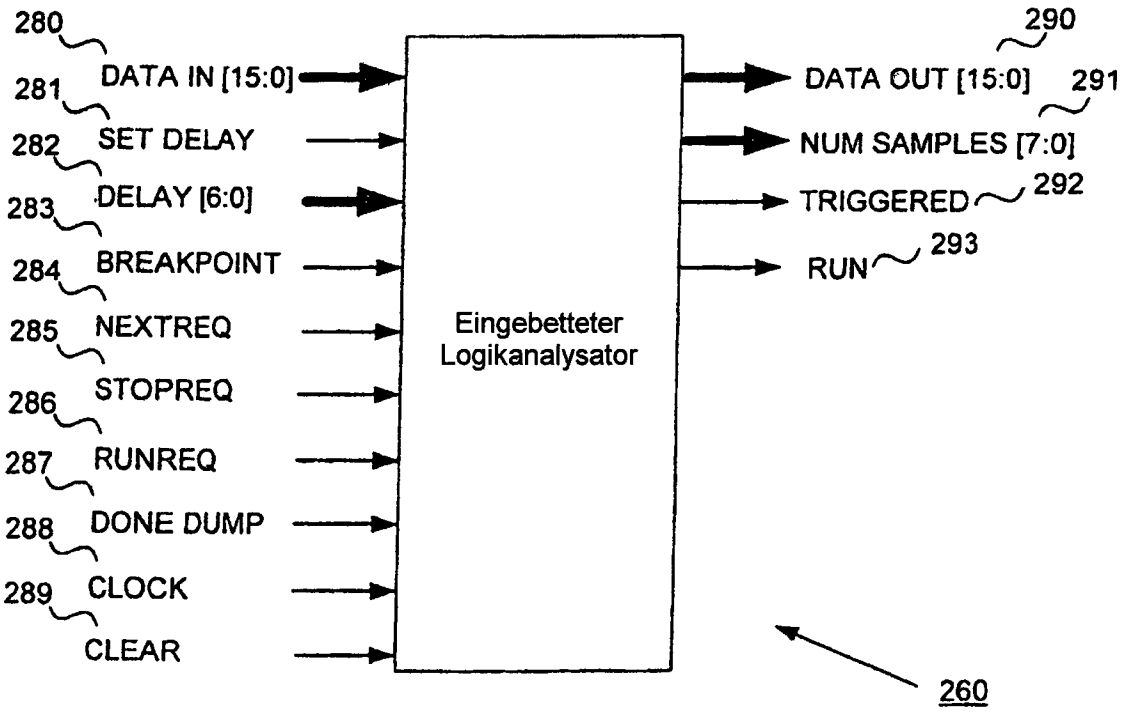
280
DATA IN [15:0]
281
SET DELAY
282
DELAY [6:0]
283
BREAKPOINT
284
NEXTREQ
285
STOPREQ
286
RUNREQ
287
DONE DUMP
288
CLOCK
289
CLEAR
Eingebetteter
Logikanalysator
290
DATA OUT [15:0]
291
NUM SAMPLES [7:0]
TRIGGERED
292
RUN
293
260

FIG. 7

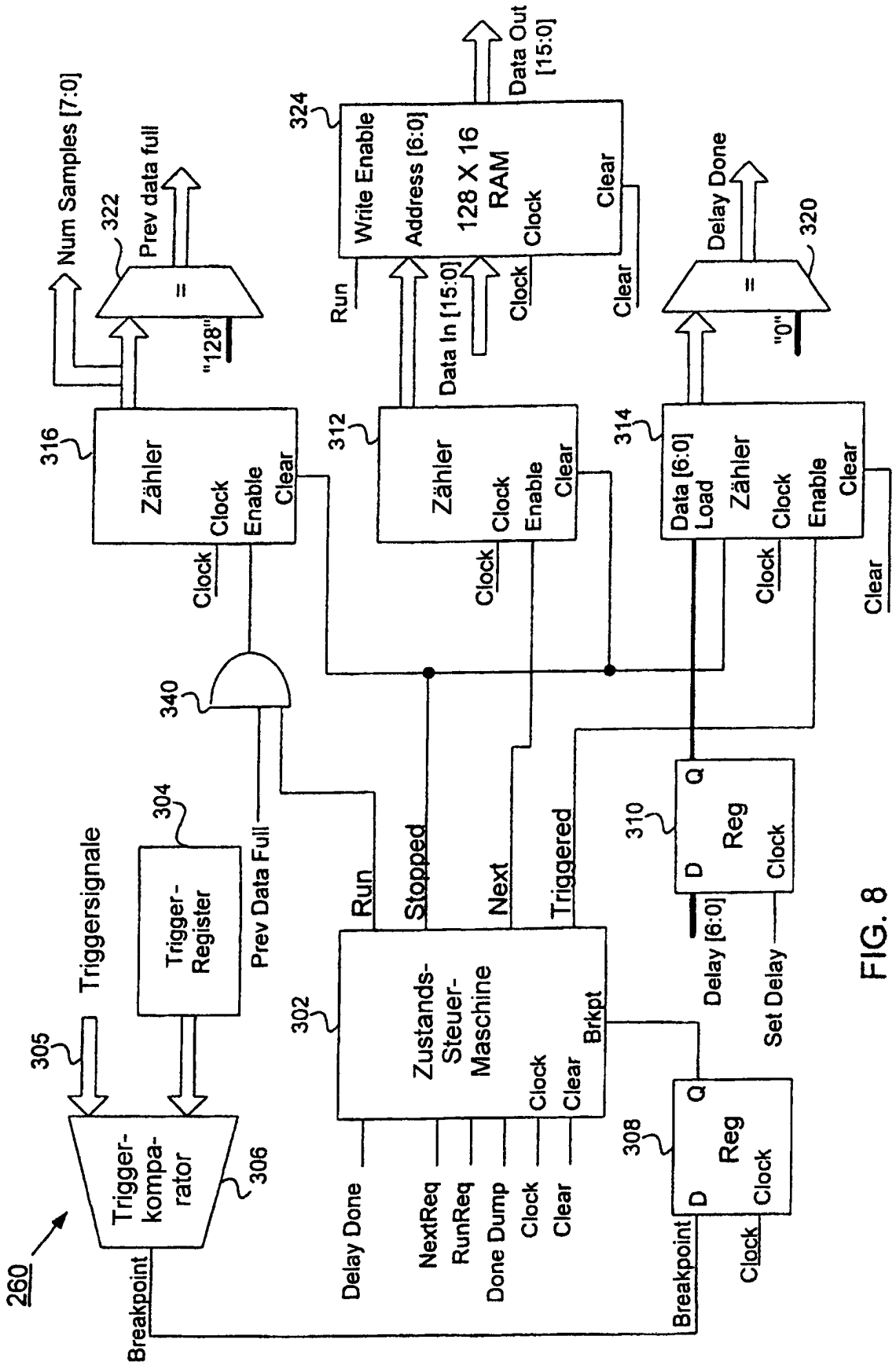
260
Triggersignale
305
Trigger-kompa-rator
306
Breakpoint
304
Trigger-Register
Prev Data Full
340
316
Zähler
Clock
Enable
Clear
Num Samples [7:0]
322
Prev data full
=
"128"
312
Zähler
Clock
Enable
Clear
324
Write Enable
Address [6:0]
128 X 16 RAM
Clock
Clear
Data Out [15:0]
Run
Data In [15:0]
Clock
Clear
314
Data [6:0]
Load
Zähler
Clock
Enable
Clear
Delay Done
320
=
"0"
Clock
Clear
302
Zustands-Steuer-Maschine
Run
Stopped
Next
Triggered
Brkpt
Clock
Clear
Delay Done
NextReq
RunReq
Done Dump
Clock
Clear
310
Reg
D
Q
Clock
Delay [6:0]
Set Delay
308
Reg
D
Q
Clock
Breakpoint
Clock

FIG. 8

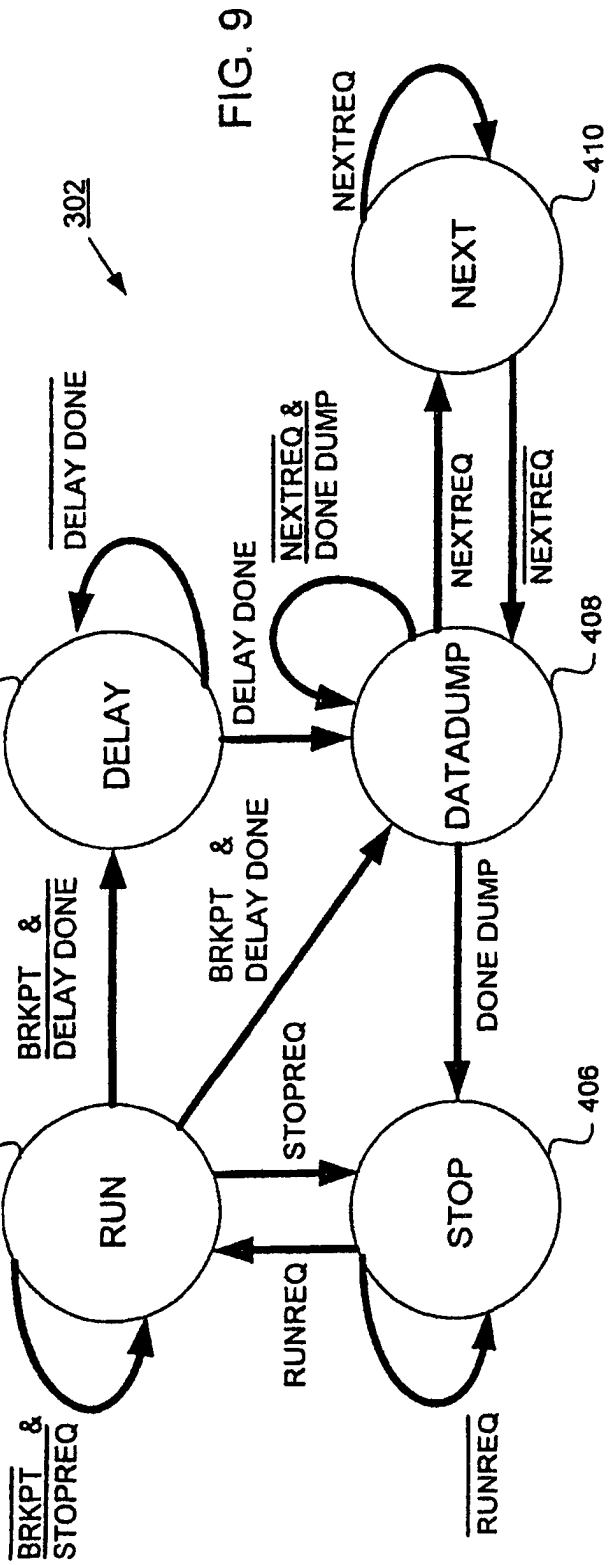

FIG. 9

	ZUSTANDS-AUSGABESIGNALE				
ZUSTÄNDE	RUN	STOPPED	TRIGGERED	NEXT
RUN	1	0	0	1
DELAY	1	0	1	1
STOP	(X)	1	(X)	(X)
DATADUMP	0	0	0	0
NEXT	0	0	0	1

FIG. 10

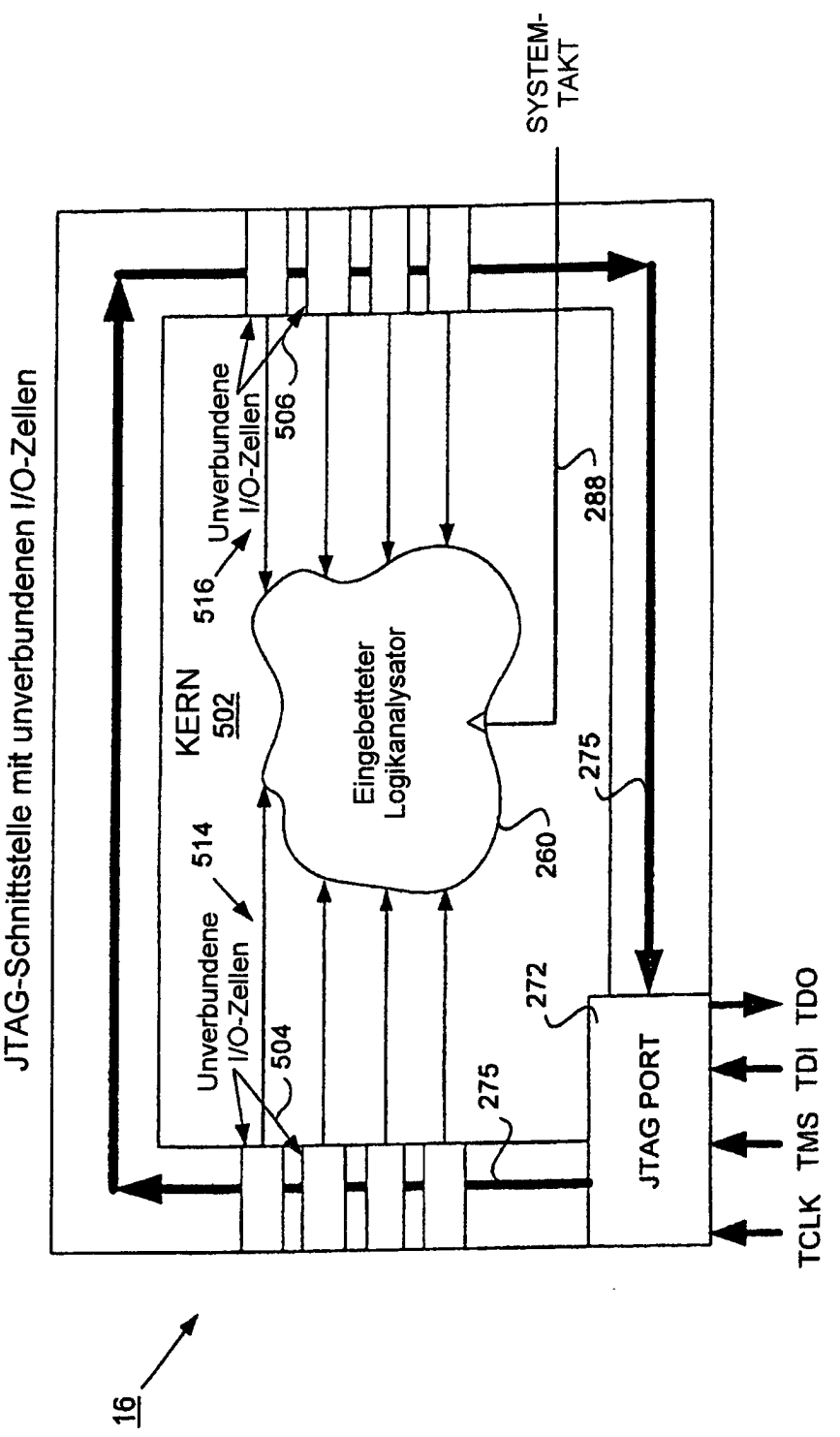
JTAG-Schnittstelle mit unverbundenen I/O-Zellen
SYSTEM-TAKT
KERN 502
Unverbundene I/O-Zellen 506
Unverbundene I/O-Zellen 516
Unverbundene I/O-Zellen 514
Unverbundene I/O-Zellen 504
Eingebetteter Logikanalysator
260
275
288
272
JTAG PORT
TCLK TMS TDI TDO
16

FIG. 11

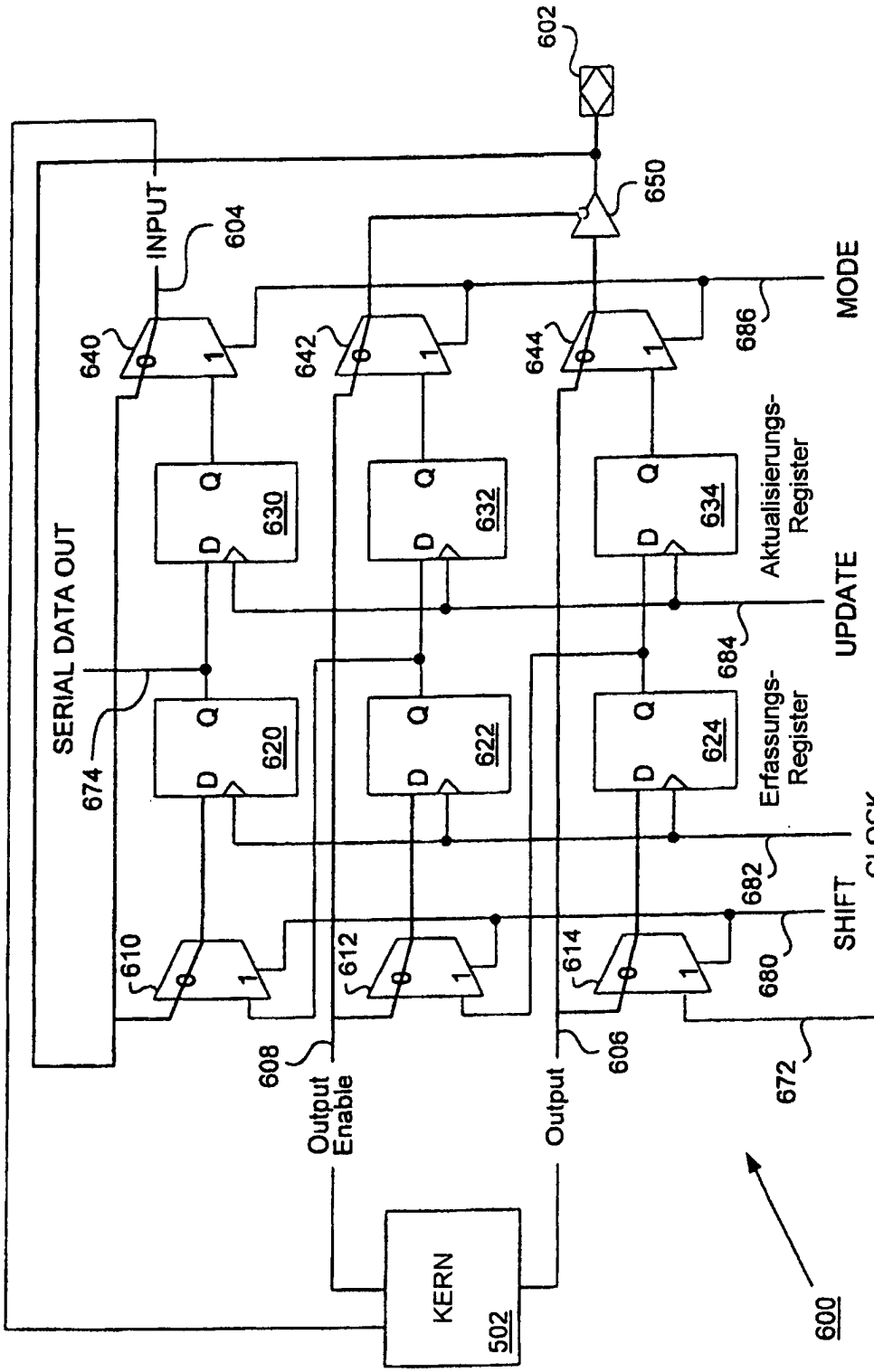

FIG. 12
(STAND DER TECHNIK)

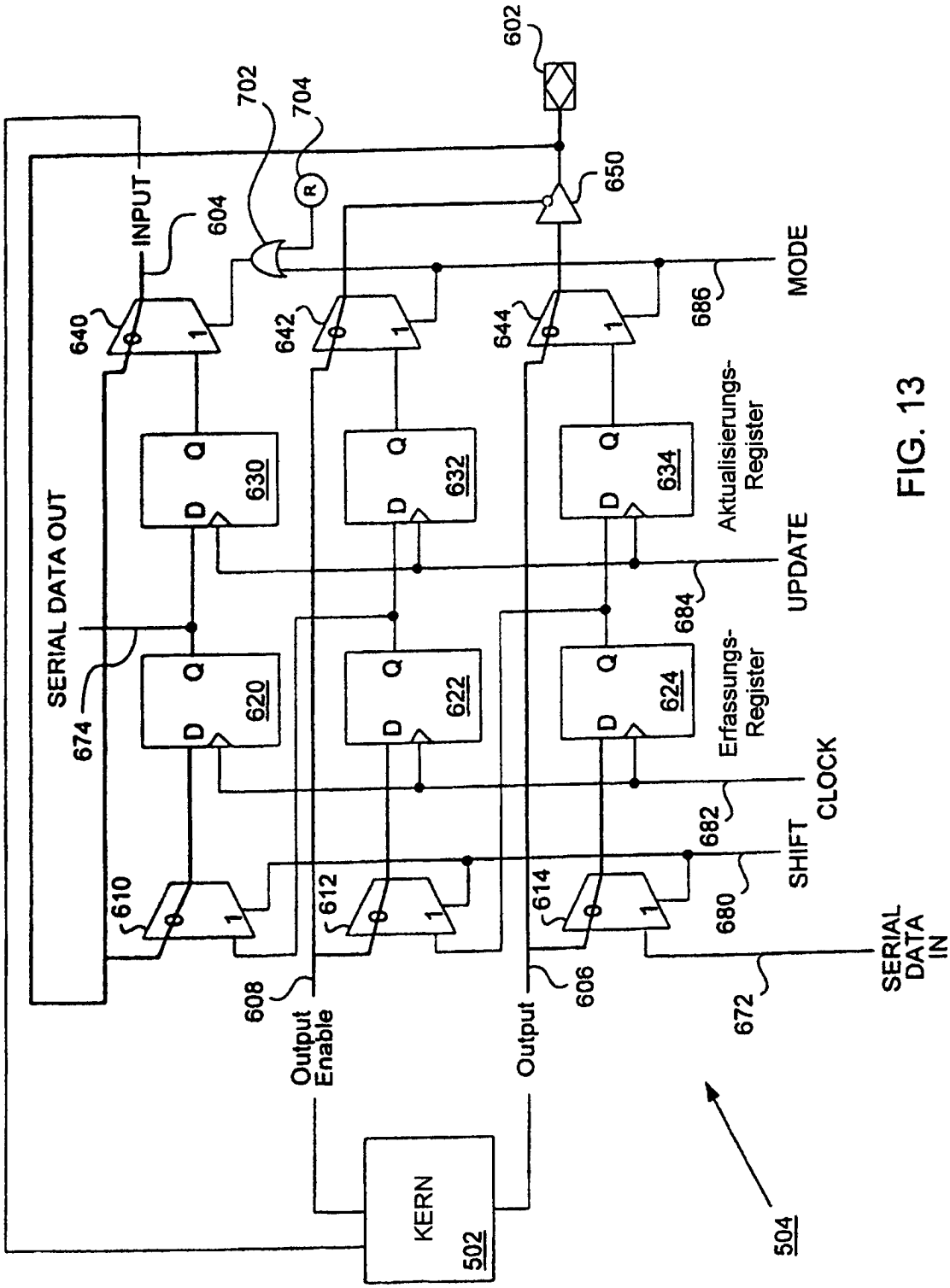
602
702
704
650
INPUT
604
640
642
644
686
MODE
630
632
634
Aktualisierungs-
Register
SERIAL DATA OUT
674
620
622
624
684
UPDATE
Erfassungs-
Register
682
CLOCK
SHIFT
610
612
614
680
SERIAL
DATA
IN
608
606
672
Output
Enable
Output
KERN
502
504

FIG. 13

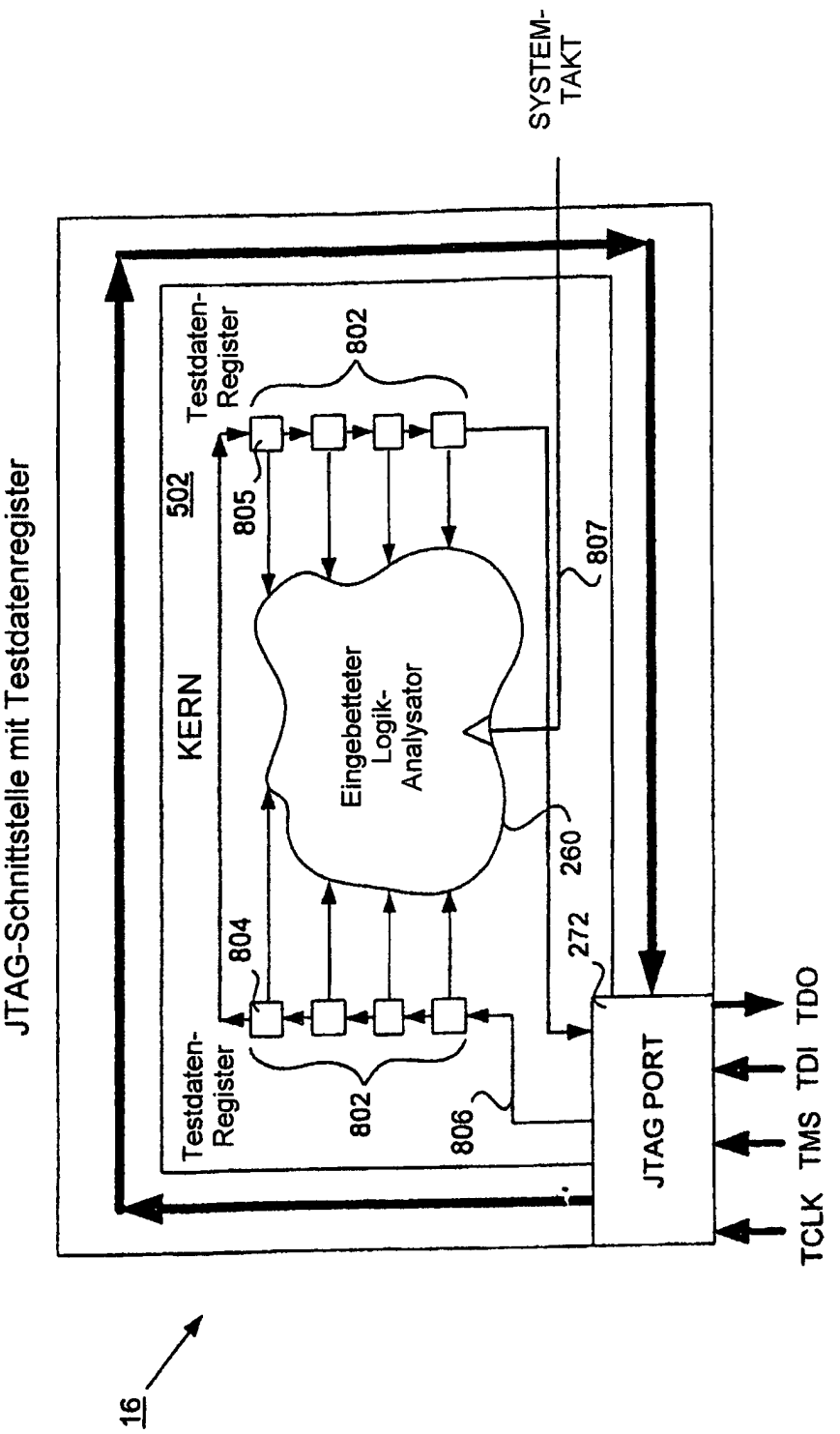
JTAG-Schnittstelle mit Testdatenregister
SYSTEM-TAKT
KERN
502
Testdaten-Register
802
805
807
Eingebetteter Logik-Analysator
260
804
272
Testdaten-Register
802
806
JTAG PORT
TCLK
TMS
TDI
TDO
16

FIG. 14

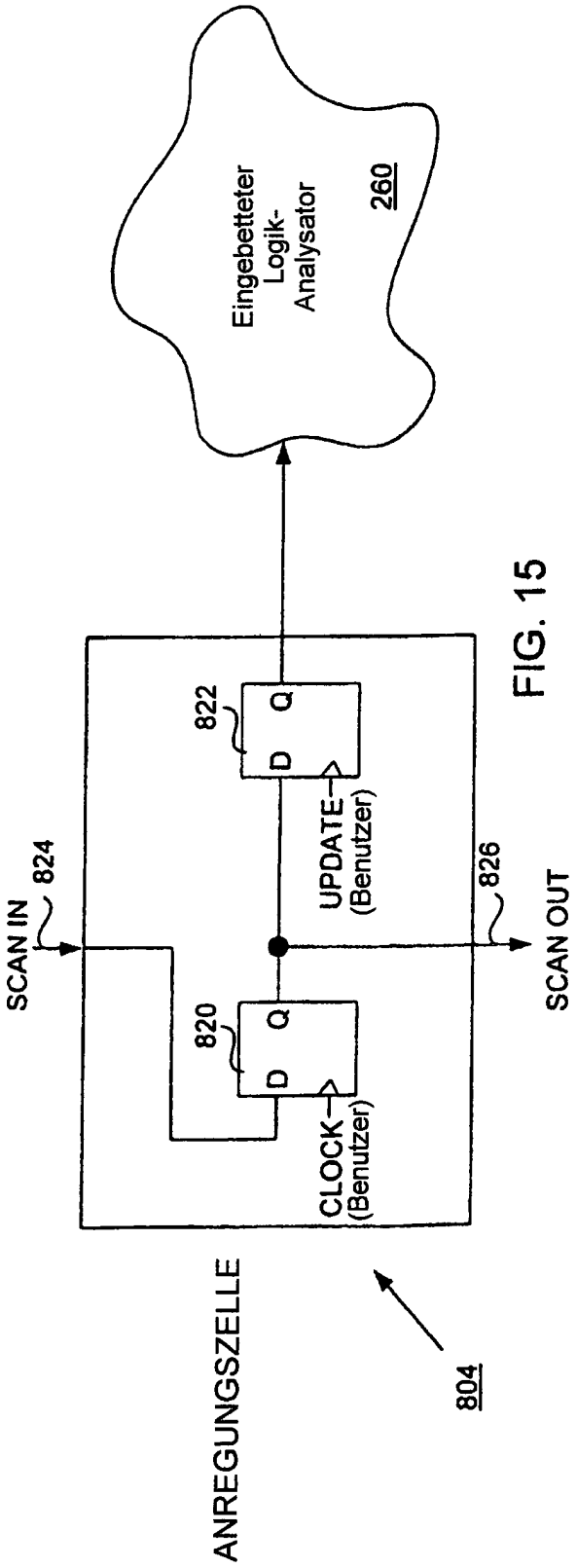
Eingebetteter Logik-Analysator
260
822
Q
D
UPDATE (Benutzer)
SCAN IN
824
826
SCAN OUT
820
Q
D
CLOCK (Benutzer)
ANREGUNGSZELLE
804

FIG. 15

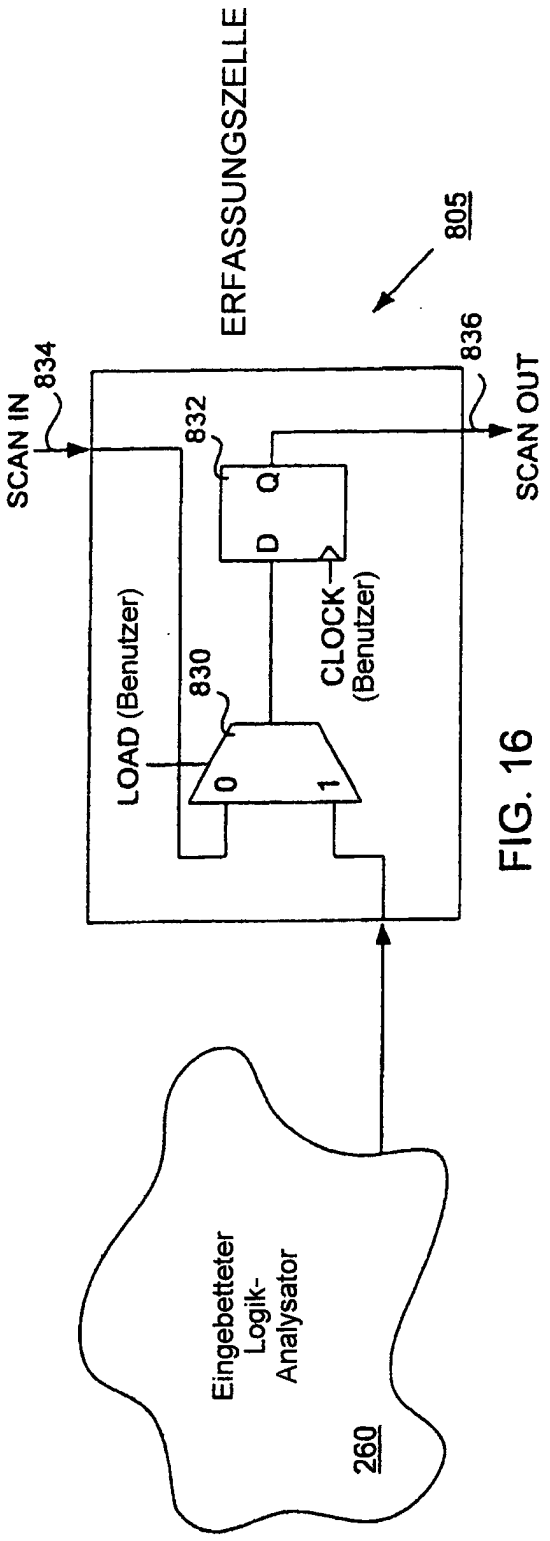
ERFASSUNGSZELLE
805
SCAN IN
834
836
SCAN OUT
832
Q
D
CLOCK (Benutzer)
LOAD (Benutzer)
830
0
1
Eingebetteter Logik-Analysator
260

FIG. 16

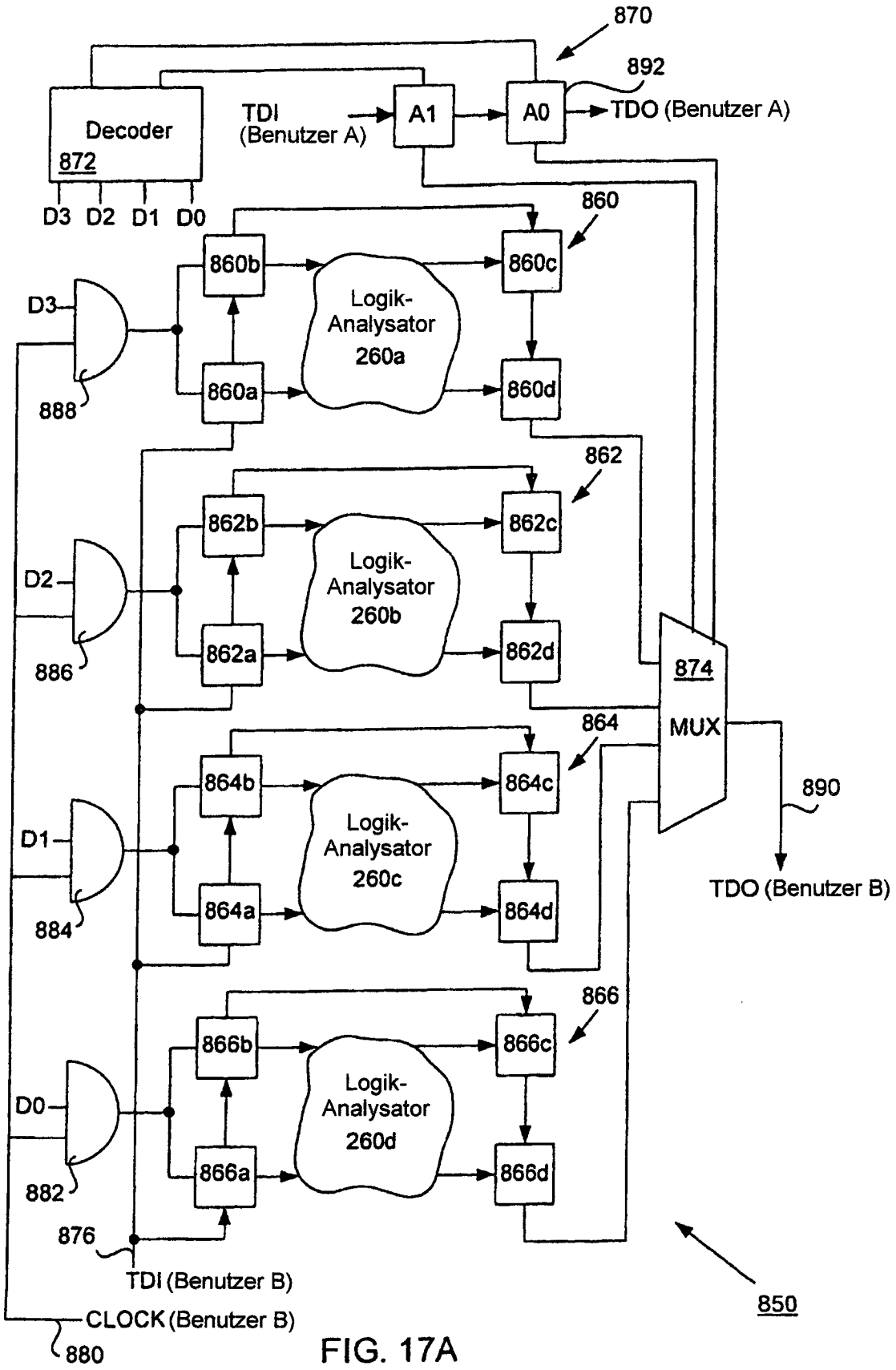
870
892
Decoder
872
TDI
(Benutzer A)
A1
A0
TDO (Benutzer A)
D3 D2 D1 D0
860
860b
860c
Logik-
Analysator
260a
D3
888
860a
860d
862
862b
862c
Logik-
Analysator
260b
D2
886
862a
862d
874
MUX
864
864b
864c
Logik-
Analysator
260c
890
D1
884
864a
864d
TDO (Benutzer B)
866
866b
866c
Logik-
Analysator
260d
D0
882
866a
866d
876
TDI (Benutzer B)
CLOCK (Benutzer B)
880
850

FIG. 17A

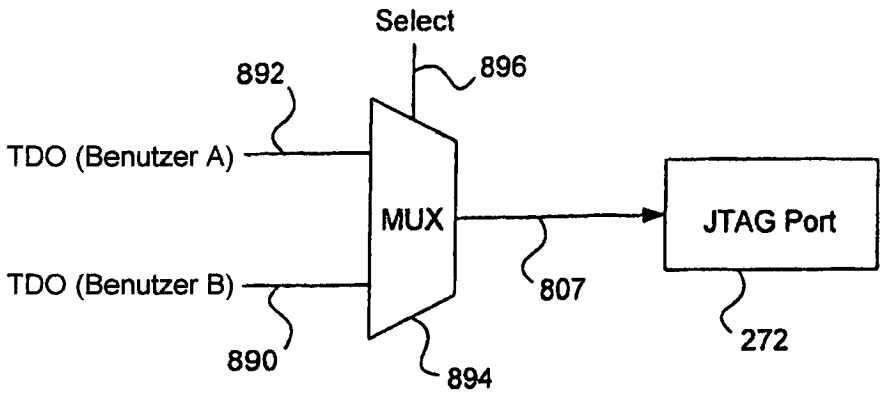

FIG. 17B

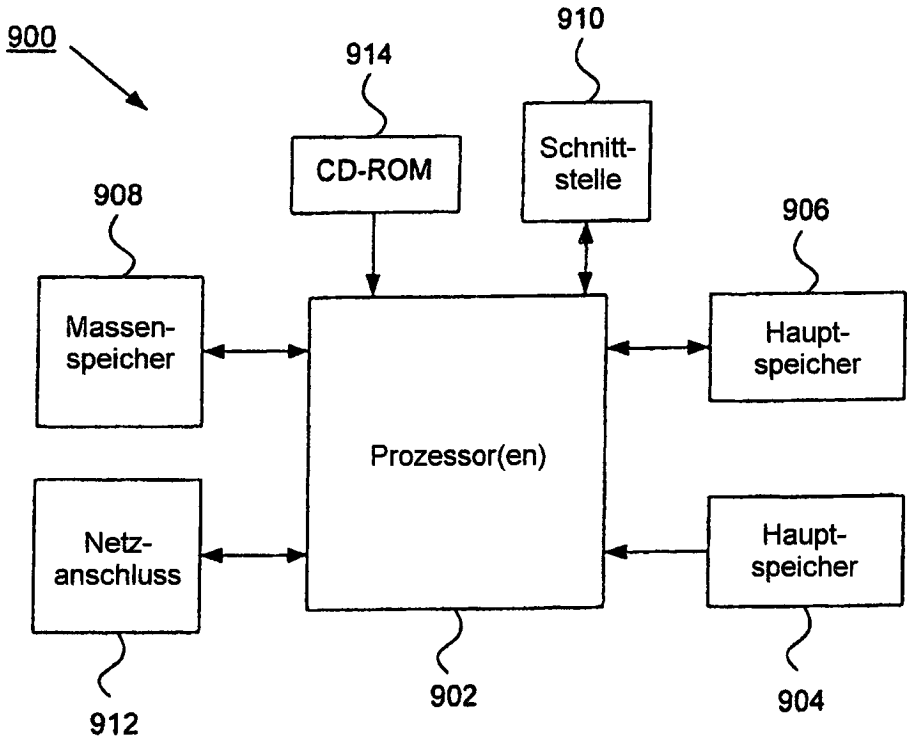

FIG. 18