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

(10) **DE 698 36 407 T2** 2007.07.05

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 919 822 B1**
(21) Deutsches Aktenzeichen: **698 36 407.4**
(96) Europäisches Aktenzeichen: **98 117 373.5**
(96) Europäischer Anmeldetag: **14.09.1998**
(97) Erstveröffentlichung durch das EPA: **02.06.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **15.11.2006**
(47) Veröffentlichungstag im Patentblatt: **05.07.2007**

(51) Int Cl.[8]: ***G01R 31/319*** (2006.01)
G06F 11/273 (2006.01)
G01R 31/28 (2006.01)
G01R 35/00 (2006.01)

(30) Unionspriorität:
979511 **26.11.1997** **US**

(73) Patentinhaber:
Avago Technologies General IP (Singapore) Pte. Ltd., Singapur/Singapore, SG

(74) Vertreter:
Schoppe, Zimmermann, Stöckeler & Zinkler, 82049 Pullach

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

(72) Erfinder:
Stickler, Heather L., Ft. Collins, CO 80526, US

(54) Bezeichnung: **Anordnung zur Überprüfung der Signalspannungspegelgenauigkeit in einer digitalen Testeinrichtung**

Beschreibung

HINTERGRUND DER ERFINDUNG

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf ein System zum Testen automatischer Testmaschinen, und insbesondere auf ein Verfahren und eine Vorrichtung zum Testen der Genauigkeit von Signalspannungspegeln an einer digitalen Testvorrichtung.

Erläuterung der verwandten Technik

[0002] Eine Vielfalt automatischer Testausrüstung (ATE; ATE = automatic test equipment) ist seit langem zum Testen von elektronischen Schaltungen, Bauelementen und anderen Halbleiter- und Elektronikprodukten bekannt. Allgemein ist automatische Testausrüstung in zwei breite Kategorien unterteilt, analoge Tester und digitale Tester. Wie die Namen implizieren, sind analoge Tester allgemein zum Testen analoger Schaltungsvorrichtungen entworfen, während digitale Tester zum Testen digitaler Schaltungsvorrichtungen entworfen sind. Digitale Tester umfassen allgemein, wie bekannt ist, eine Testvorrichtung, die eine Anzahl interner Schaltungskarten oder Kanäle aufweist, die programmierbar gesteuerte Testsignale zum Testen und Auswerten einer zu testenden Vorrichtung (DUT; DUT = device unter test) erzeugen. Insbesondere ist ATE programmierbar gesteuert, um zum Testen einer Vielzahl von Vorrichtungen in einer Vielzahl von Weisen angepasst oder konfiguriert zu sein. Dies wird durch ein Programmieren von Ausgangssignalen, um ein bestimmtes Signal (oder einen Signalübergang) auf einen bestimmten Anschlussstift oder eine Signalleitung auf einer DUT anzulegen, erzielt. Diesbezüglich umfasst ein digitaler Tester allgemein einen Testkopf, durch den elektrische Signale in den Tester eingegeben und aus demselben ausgegeben werden.

[0003] Der Testkopf weist eine Anzahl von Verbindern auf, wobei jeder derselben einen Kanal definiert, die über ein Kabel oder anderweitig mit einer zu testenden Vorrichtung verbunden sein können. Die Elektronik innerhalb des digitalen Testers könnte dann Signale über den Testkopf in eine DUT eingeben und aus derselben ausgeben.

[0004] Durch eine extrem einfache Darstellung wird ein digitaler Tester betrachtet, der konfiguriert ist, um einen Wafer zu testen, der unter anderem ein Zwei-Eingangs-UND-Gatter beinhaltet. Der digitale Tester könnte konfiguriert sein, um eine logische Eins an die beiden Signalleitungen anzulegen, die den Eingängen des UND-Gatters entsprechen, dann das Signal auf der Signalleitung empfangen, die dem Ausgang entspricht, um sicherzustellen, dass dasselbe auf eine logische Eins getrieben wird. Der Tester könnte dann konfiguriert sein, um abwechselnd logische Null-Signale an jede der beiden Signalleitungen anzulegen, die den UND-Gatter-Eingängen entsprechen, um zu verifizieren, dass die Ausgabe des UND-Gatters ansprechend darauf von einer logischen Eins zu einer logischen Null übergeht. Obwohl ein derartiger Test den Funktionsbetrieb des UND-Gatters verifiziert, müssen zusätzliche Tests ausgeführt werden, um eine Zeitgebung und andere Aspekte des UND-Gatters zu verifizieren.

[0005] Es wird zum Beispiel angenommen, dass die beiden Eingangssignale in das UND-Gatter eine logische Eins und eine logische Null sind, die Ausgabe also eine logische Null ist. Wenn jedoch die zweite Eingabe von einer logischen Null zu einer logischen Eins übergeht (wodurch beide Eingaben auf einer logischen Eins sind), geht die Ausgabe des UND-Gatters von einer logischen Null zu einer logischen Eins über. Es ist jedoch wichtig, dass die Ausgabe vollständig in einen Hochzustand übergeht. Insbesondere und wie bekannt ist, arbeiten digitale Logikvorrichtungen allgemein in einem Bereich van null bis fünf Volt, wobei null Volt eine logische Null ist und fünf Volt eine logische Eins ist. Wenn Bauelemente belastet werden, gelingt es ihnen jedoch oft nicht, ein Ausgangssignal vollständig auf fünf Volt zu treiben. Entsprechend wird angenommen, dass ein Bereich eine logische Eins ist. Zum Beispiel könnte eine beliebige Ausgabe über 2,4 Volt (abhängig von Bauelementspezifizierungen) als eine logische Eins behandelt werden. So sollte, mit dem UND-Gatter-Beispiel fortfahrend, wenn beide Eingaben über 2,4 Volt sind, der Ausgangspegel auch auf zumindest 2,4 Volt treiben. Es wird ein automatischer Tester angenommen, der fälschlicherweise eine 2,3-Volt-Ausgangsspannung als 2,4 Volt interpretiert hat. Die getestete Komponente könnte als funktionierend erachtet werden, wenn sie tatsächlich außerhalb einer Toleranz von Spezifizierungen des Herstellers war. Es ist zu erkennen, dass das oben gegebene Beispiel ein extrem einfaches Beispiel ist und lediglich zu Darstellungszwecken vorgelegt wird, wobei, wie für Fachleute auf dem Gebiet zu erkennen ist, digitale Tester viel höher entwickelt und in der Lage sind, viel höher entwickelte und komplexere Testroutinen durchzuführen.

[0006] Entsprechend ist ein extrem wichtiger Aspekt von ATE, einschließlich digitaler Tester, dass die Testausrüstung extrem genaue Toleranzen beibehält. Andernfalls ist nicht klar, ob gemessene Werte einer DUT Fehler oder Diskrepanzen innerhalb der DUT widerspiegeln, oder ob Fehler oder Diskrepanzen aus Komponenten- oder anderen Toleranzvariationen in den ATE-Komponenten resultieren. Diesbezüglich liefern die Hersteller automatischer Testausrüstung allgemein einen Satz von eingebauten Test- (BIT-; BIT = built in test) Routinen für die automatische Testausrüstung bereit. Wie jedoch weiter unten beschrieben ist, hat sich gezeigt, dass diese BIT-Routinen, die vom Hersteller bereitgestellt werden, allgemein unzulänglich auf dem heutigen Markt von Hochgeschwindigkeitskomponenten und Serienproduktions-Herstellungseinrichtungen sind.

[0007] Ein digitaler Tester weist allgemein eine Anzahl ähnlicher (wenn nicht identischer) Kanäle auf, die jeweils eine Anzahl von Treibern und Empfängern zum Treiben von Ausgangssignalen oder Empfangen von Eingangssignalen (Ausgabe aus dem Tester oder Eingabe in den Tester) aufweisen. Diese Kanäle (die allgemein auf einer einzelnen Schaltungsplatine beinhaltet sind) umfassen oft eine Parametermesseinheit (PMU; PMU = parametric measuring unit) oder einen bestimmten anderen Typ von Messvorrichtung zum Erfassen und Messen der Größe eines Signals. Wie bekannt ist, sind PMUs zum Kalibrieren platineninterner Komponenten bekannt. So könnte, wenn ein bestimmter Treiber programmiert ist, um ein Signal einer bestimmten Amplitude zu erzeugen, die platineninterne PMU dieses Signal überwachen und messen, um zu bestimmen, ob die Amplitude dieses Signals der richtige Wert ist.

[0008] Dieser Typ eines weiteren Testens könnte durch ein Verwenden genauer externer Testausrüstung in einer manuellen Weise erzielt werden. Voltmeter zum Beispiel könnten zum Testen von Spannungspegeln verwendet werden und Oszilloskope könnten zum Auswerten von Zeitgebungsaspekten verwendet werden. Es ist jedoch zu erkennen, dass ein externes Testen einer ATE extrem zeitaufwändig ist und deshalb Produktionsmengen nachteilig beeinflusst. Ein bestimmtes Problem, das in den letzten Jahren bemerkt wurde, bezieht sich insbesondere auf die Fähigkeit eines digitalen Testers, sich selbst auf Genauigkeit zu testen. Als ein Ergebnis des allgemeinen Versagens von Testvorrichtungen diesbezüglich weiß man, dass Schaltungsentwerfer Tests für spezifische Tester entwerfen. Dies bedeutet, dass ein Schaltungsentwerfer einen Test mit Testparametern derart entwerfen könnte, dass der Test die Testanforderungen einer bestimmten automatischen Testvorrichtung erfüllt, während diese gleichen Tests bei anderen Testern nicht bestanden werden. Es wurden Tabellen erzeugt, die bestimmte Charakteristika verschiedener automatischer Testausrüstung verfolgen. Schaltungsentwerfer haben die innerhalb dieser Tabellen beinhalteten Informationen verwendet, um Testparameter zu entwerfen, um den Test eines bestimmten Testers zu bestehen. Klar beeinflusst dieser Ansatz jedoch den Gesamtertrag eines entworfenen Produkts negativ, sowie einen Materialfluss durch die Serienproduktions-Herstellungsumgebung, und wird nicht bevorzugt.

[0009] Die US-A-5539305 offenbart einen elektronischen Schaltungstester zum Messen des Ansprechens von elektrischen Signalen, die an eine zu testende elektronische Schaltung angelegt werden, wobei der elektronische Schaltungstester mit einer Kalibrierungsplatine zum Bewirken einer Kalibrierung des Testers auf der Ebene von Verbindern, die in einen Testkopf des Testers eingebaut sind, versehen ist. Gemäß einem Ausführungsbeispiel eines elektronischen Schaltungstesters zum Messen elektrischer Hochfrequenzsignale umfasst die Kalibrierungsplatine ein dielektrisches Substrat, Kalibrierungsstandards, wie zum Beispiel Unterbrechungen, Kurzschlüsse und Lasten, die in jeweiligen Regionen des Substrats an dem Substrat befestigt sind, Verbinder zum elektrischen Verbinden des Kalibrierungsstandards mit den Verbindern des Testkopfs, wenn die Kalibrierungsplatine selektiv anstelle einer Halterungsplatine eingesetzt wird, eine Struktur zum Befestigen der Kalibrierungsplatine an dem Testkopf, so dass eine elektrische Verbindung zwischen der Kalibrierungsplatine und dem Testkopf eingerichtet wird, und Indizien auf der Kalibrierungsplatine und dem Testkopf zum Anzeigen einer Folge von Drehpositionen der Kalibrierungsplatine zur Kalibrierung des Testers unter Verwendung der Kalibrierungsstandards. Die Kalibrierungsplatine wird nachfolgend in die angezeigten Drehpositionen gedreht, so dass ein Tester auf die Ebene der Verbinder des Testkopfs kalibriert werden kann. Die Kalibrierungsplatine kann ferner zumindest einen Verifizierungsstandard, der an dem Substrat befestigt ist, umfassen. Eine weitere Kalibrierungsplatine umfasst Durchgangsverbindungen. In einer Kalibrierungsprozedur installiert der Benutzer zuerst die erste Kalibrierungsplatine an den jeweiligen Drehpositionen, gefolgt durch ein Einbauen der zweiten Kalibrierungsplatine mit den Durchgangsverbindungen. Nach erfolgter Kalibrierung des elektronischen Schaltungstesters verwendet der elektronische Schaltungstester die Messungen in Bezug auf die Verifizierungsstandards, um zu bestimmen, ob die korrigierten Messungen an dem Verifizierungsstandard innerhalb einer akzeptablen Toleranz, die eine akzeptable Kalibrierung anzeigt, liegen oder nicht.

[0010] So besteht die Aufgabe der vorliegenden Erfindung darin, ein System zum Verbessern der Genauigkeit einer automatischen Testausrüstung, insbesondere digitaler Tester, bereitzustellen, indem das Verhalten der digitalen Tester mit einem hohen Maß an Genauigkeit verifiziert wird.

[0011] Diese Aufgabe wird durch ein System zum Testen einer Komponententoleranz einer Vorrichtung gemäß den Ansprüchen 1 und 2 und ein Verfahren zum Diagnostizieren von Komponentenvarianzen gemäß Anspruch 6 gelöst.

[0012] Bestimmte Aufgaben, Vorteile und neuartige Merkmale der Erfindung sind teilweise in der folgenden Beschreibung dargelegt und werden teilweise für Fachleute auf dem Gebiet nach einer Durchsicht von Folgendem zu erkennen sein oder können aus der Anwendung der Erfindung erlernt werden. Die Aufgaben und Vorteile der Erfindung könnten mittels der Einrichtungen und Kombinationen, die in den beigefügten Ansprüchen besonders herausgestellt sind, realisiert und erhalten werden.

[0013] Um die Vorteile und neuartigen Merkmale zu erzielen, richtet sich die vorliegende Erfindung allgemein auf ein System und ein Verfahren zum Testen von Komponententoleranzen einer Vorrichtung zum Testen integrierter Schaltungen, wobei die Vorrichtung eine Mehrzahl von Testverbindern aufweist, die an einem Testkopf angeordnet sind, wobei jeder Testverbinder elektrische Leiter aufweist, die elektrische Signale für einen Testkanal tragen, wobei jeder Testkanal einer Schaltungsplatine entspricht, die zumindest einen Treiber und einen Empfänger umfasst. Gemäß den breiten Konzepten und Lehren der Erfindung funktionieren eine Vorrichtung und ein Verfahren, um Treiber- und Empfängerkomponenten zu vergleichen, indem Treiber von einer ersten Platine mit Empfängern einer zweiten Platine verbunden werden und umgekehrt. Da eingebaute Testprozeduren, die durch den Testerhersteller bereitgestellt werden, typischerweise die Treiber und Empfänger einer einzelnen Platine mit einer Messeinheit, die auf dieser gleichen Platine vorgesehen ist, kalibrieren, ist es möglich, dass die Treiber und Empfänger dieser Platine ohne Erfassung außerhalb einer Spezifizierung geraten, wenn die Messeinheit dieser Platine außerhalb einer Spezifizierung gerät. Entsprechend stellt die Überprüfung auf Übereinstimmung, die durch die vorliegende Erfindung bereitgestellt wird, ein stark verbessertes System und ein derartiges Verfahren zum Verifizieren der Genauigkeit des Betriebs von Testerkomponenten bereit.

[0014] Gemäß einem Aspekt der Erfindung wird eine Vorrichtung bereitgestellt, die eine Kurzschlusseinrichtung zum Einrichten einer elektrischen Niederimpedanzverbindung zwischen elektrischen Leitern eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber von einer ersten Schaltungsplatine elektrisch über einen Niederimpedanzpfad mit einer Spannungsmesseinheit auf einer zweiten Schaltungsplatine verbunden ist, umfasst. Vorzugsweise ist diese „Kurzschlusseinrichtung" durch eine gedruckte Schaltungsplatine implementiert, die Metallleiterbahnen beinhaltet, die zwischen Anschlussstiften zweier Testverbinder kurzschließt oder eine direkte elektrische Verbindung zwischen denselben einrichtet. Eine Steuerung ist konfiguriert, um den ersten Treiber so zu steuern, "dass der erste Treiber ein elektrisches Signal eines vorbestimmten anvisierten Werts ausgibt. Um jedoch zu testen, ob dieser anvisierte Wert genau ist, ist die Messeinheit angeordnet, um den empfangenen Signalspannungspegel zu empfangen und zu messen. Eine Verifizierungseinrichtung ist konfiguriert, um zu verifizieren, dass der an der Messeinheit empfangene Signalpegel innerhalb einer vorbestimmten Toleranz des vorbestimmten anvisierten Signalpegels ist.

[0015] Gemäß einem weiteren Aspekt der Erfindung wird eine Vorrichtung bereitgestellt, die eine Kurzschlusseinrichtung zum Einrichten einer elektrischen Niederimpedanzverbindung zwischen elektrischen Leitern eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber von einer ersten Schaltungsplatine elektrisch über einen Niederimpedanzpfad mit einem ersten Empfänger auf einer zweiten Schaltungsplatine verbunden ist, umfasst. Eine Steuerung ist konfiguriert, um den ersten Treiber so zu steuern, dass der erste Treiber ein elektrisches Signal eines vorbestimmten Werts ausgibt. Es wird angenommen, dass dieser Ausgangswert genau ist, da der erste Treiber zuvor auf Genauigkeit getestet wurde (wie oben beschrieben ist). Deshalb wertet dieser Aspekt des Testens eine Empfängergenauigkeit aus. Entsprechend ist eine Auswertungseinrichtung entworfen, um den an dem ersten Empfänger empfangenen Signalpegel auszuwerten, und eine Verifizierungseinrichtung ist konfiguriert, um zu verifizieren, dass der an dem ersten Empfänger interpretierte Signalpegel innerhalb einer vorbestimmten Toleranz des vorbestimmten Signalpegels, der aus dem ersten Treiber ausgegeben wird, ist.

[0016] Gemäß einem weiteren Aspekt der vorliegenden Erfindung wird ein Verfahren zum Diagnostizieren von Komponentenvarianzen in einer Testvorrichtung bereitgestellt, wobei die Testvorrichtung vorzugsweise eine Mehrzahl von Testverbindern aufweist, die an einem Testkopf angeordnet sind, wobei jeder Testverbinder elektrische Leiter aufweist, die elektrische Signale für einen Testkanal tragen, wobei jeder Testkanal einer Schaltungsplatine entspricht, die zumindest einen Treiber und einen Empfänger umfasst. Das Verfahren umfasst den Schritt eines Einrichtens einer elektrischen Verbindung zwischen elektrischen Leitern eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber von einer ersten Schaltungsplatine elektrisch über einen Niederimpedanzpfad mit einem ersten Empfänger auf einer zweiten Schaltungsplatine verbunden

ist, wobei man weiß, dass der erste Treiber innerhalb einer spezifizierten Toleranz ist.

[0017] Das Verfahren steuert dann die Ausgabe des ersten Treibers, um ein elektrisches Signal auf einen vorbestimmten Wert zu treiben, und wertet den an dem ersten Empfänger interpretierten Signalpegel aus. Schließlich verifiziert das Verfahren, dass der durch den ersten Empfänger interpretierte Signalpegel innerhalb einer vorbestimmten Toleranz des vorbestimmten Signalpegels, der aus dem ersten Treiber ausgegeben wird, ist. Falls dies der Fall ist, erkennt das System die getesteten Komponenten als innerhalb spezifizierter Toleranzen liegend. Falls dies nicht der Fall ist, führt das System eine Flag-Markierung des Fehlers durch und könnte eine weitere geeignete Korrekturhandlung unternehmen. Eine derartige Korrekturhandlung könnte ein Initiieren eines weiteren Testens oder ein Benachrichtigen (über E-Mail oder anderweitig) relevanten Testpersonals, ein weiteres manuelles Testen auszuführen, aufweisen.

BESCHREIBUNG DER ZEICHNUNGEN

[0018] Die beiliegenden Zeichnungen, die in der Beschreibung beinhaltet und einen Teil derselben bilden, stellen mehrere Aspekte der vorliegenden Erfindung dar und dienen gemeinsam mit der Beschreibung dazu, die Prinzipien der Erfindung zu erklären. In den Zeichnungen zeigen:

[0019] Fig. 1 ein Blockdiagramm einer automatisierten Testvorrichtung zum Testen eines digitalen elektronischen Schaltungsaufbaus;

[0020] Fig. 2 ein Blockdiagramm, das eine automatisierte Testvorrichtung darstellt;

[0021] Fig. 3 eine detailliertere Ansicht eines Testkopfs einer automatisierten Testvorrichtung;

[0022] Fig. 4 ein Blockdiagramm, das Hauptkomponenten einer Schaltungsplatine, die eine Mehrzahl von Treibern und Empfängern beinhalten, in einer Testvorrichtung darstellt;

[0023] Fig. 5 ein Blockdiagramm, das die Zwischenverbindung zwischen Anschlussstiften zweier Testverbinder des Testkopfs der Testvorrichtung darstellt;

[0024] Fig. 6A ein Blockdiagramm, das die Konfiguration zweier Schaltungsplatinen des in Fig. 4 dargestellten Typs, konfiguriert, um einen Treiberfehler zu testen und zu messen, darstellt;

[0025] Fig. 6B ein Flussdiagramm, das die Konfiguration zweier Schaltungsplatinen des in Fig. 4 dargestellten Typs, konfiguriert, um einen Empfängerfehler zu testen und zu messen, darstellt;

[0026] Fig. 7 ein Betriebsflussdiagramm, das die Hauptschritte beim Testen und Messen eines Treiberfehlers darstellt; und

[0027] Fig. 8 ein Betriebsflussdiagramm, das die Hauptschritte beim Testen und Messen eines Empfängerfehlers darstellt.

DETAILLIERTE BESCHREIBUNG DES BEVORZUGTEN AUSFÜHRUNGSBEISPIELS

[0028] Nach Zusammenfassung verschiedener Aspekte der vorliegenden Erfindung wird nun detailliert Bezug auf die Beschreibung der Erfindung genommen, wie in den Zeichnungen dargestellt. Während die Erfindung in Verbindung mit diesen Zeichnungen beschrieben ist, besteht keine Absicht, diese auf das oder die Ausführungsbeispiele, die hierin offenbart sind, einzuschränken. Im Gegenteil ist es die Absicht, alle Alternativen, Modifizierungen und Äquivalente, die innerhalb des Schutzbereichs der Erfindung beinhaltet sind, wie durch die beigefügten Ansprüche definiert ist, abzudecken.

[0029] Bezug nehmend auf Fig. 1 ist ein Blockdiagramm eines „Platinentyp"-ATE-Systems (Testers) **100** gezeigt. Obwohl das bevorzugte Ausführungsbeispiel der vorliegenden Erfindung auf Wafertester gerichtet ist, sind die Konzepte und Lehren nicht so eng eingeschränkt. Entsprechend ist Fig. 1 lediglich gezeigt, um eine Plattform des Stands der Technik darzustellen, auf die die Konzepte und Lehren der vorliegenden Erfindung angewendet werden könnten. Das ATE-System **100** umfasst einen Testerzeuger **102** und eine Teststeuerung **104**. Der Testerzeuger **102** erzeugt einen schaltungsinternen Test für jede zu testende Vorrichtung auf einer Platine. Ein generischer Testplan **106** schafft eine Überwachungssteuerung über das Testen. Dies umfasst ein Sequenzieren der Tests, ein Protokollieren der Ergebnisse, ein Steuern einer Platine/Halterung-Schnittstellen-

verbindung, ein Steuern der Testleistungsversorgungen und ein Bereitstellen einer Benutzerschnittstelle. Die Kombination der einzelnen schaltungsinternen Tests und des Testplans **106** bilden eine Testspezifizierung.

[0030] Eine Vorrichtungsmodellbibliothek **108**, eine physische Datenbank **110** und eine elektrische Datenbank **112** könnten die Daten bereitstellen, die für den Testerzeuger **102** erforderlich sind, um die schaltungsinternen Tests zu erzeugen. Die elektrische Datenbank **112** könnte eine Liste der Vorrichtungen auf der DUT-Platine **116**, eine elektrische Beschreibung für jede Vorrichtung und elektrische Zwischenverbindungsinformationen beinhalten. Die physische Datenbank **110** könnte eine topologische Beschreibung der Platine beinhalten, die durch die Teststeuerung **104** verwendet werden kann, um eine DUT **116** zu lokalisieren und zu testen. Die physische Datenbank **110** und die elektrische Datenbank **112** werden typischerweise durch ein CAD/CAM-System (CAD = Computer aided design = computergestützter Entwurf; CAM = Computer aided manufacturing = computergestützte Fertigung) während des Entwurfs einer Platine erzeugt.

[0031] Die Vorrichtungsmodellbibliothek **108** beinhaltet eine Mehrzahl zuvor erzeugter generischer Modelle für häufig verwendete digitale integrierte Schaltungsschips (Ics; IC = integrated circuit). Im Wesentlichen ist jedes Modell eine Testroutine, die in die Platinentestspezifizierung eingeführt (d.h. editiert) werden soll. Jedes Vorrichtungsmodell könnte für eine spezifische Vorrichtung Außenanschlussinformationen (d.h. welche Anschlussstifte Eingänge, Ausgänge, bidirektional oder nicht verwendet sind), ein Testprogramm, ein Verfahren zum Vorkonditionieren (unten beschrieben) jeder Ausgabe der Vorrichtung und vorrichtungsspezifische Informationen, die eine Teststrukturrate und erforderliche Signalpegel umfassen, bereitstellen.

[0032] Die Teststeuerung **104** führt die schaltungsinternen Tests, die durch den Testerzeuger **102** erzeugt werden, aus. Ein Treibermodul **114** wird verwendet, um die Testsignale an eine DUT **116** anzulegen, und ein Sensor-(Empfänger-)Modul **118** wird verwendet, um die Antwort der DUT **116** auf die Testsignale zu empfangen. Die Kombination des Treibermoduls **114** und des Sensormoduls **118** ist als ein Testerkanal bekannt. Der HP83000 Integrated Circuit-Tester ist ein Beispiel eines ATE-Systems und ist bei der Hewlett-Packard Company erhältlich.

[0033] Nun wird Bezug auf Fig. 2 genommen, die eine Umgebung darstellt, in der der Tester **100** arbeiten könnte. Ein Host-Computer **202**, auf dem ein Anwendungsprogramm läuft, könnte mit einer Test-Hardware **208** gekoppelt sein. Bei einem Ausführungsbeispiel könnte der Host-Computer **202** mit der Test-Hardware **208** über ein lokales Netz (LAN; LAN = Local Area Network) **204** gekoppelt sein. Die Test-Hardware **208** umfasst typischerweise einen Testkopf **205**, der den Schnittstelleneingang und -ausgang zu einer DUT **116** bereitstellt. Die Test-Hardware **208** könnte Vorrichtungen, wie zum Beispiel Treiber und Empfänger, umfassen, die zur Durchführung eines Testens an der DUT **116** verwendet werden können. Ein Anwendungsprogramm in dem Host-Computer **202** könnte mit einem Interpretierer kommunizieren, der Dynamikverbindungsbibliothek-(DLL-; DLL = dynamic link library) Rufe durchführt, die den entfernten Testkopf **205** anweisen, eine bestimmte Funktion durchzuführen. Die Test-Hardware **208** könnte Instruktionen von dem Host-Computer **202** empfangen. Diese Instruktionen können dann die verschiedenen Tests, die auf der DUT **116** laufen, steuern.

[0034] Fig. 3 stellt die Test-Hardware **208** detaillierter dar. Bei einem Ausführungsbeispiel werden sechzehn Instrumententore **302**, die jeweils eine Hoch-Leitung **303** und eine Niedrig-Leitung **305** aufweisen, in einen Instrumenten-MUX **304** eingegeben. Der Instrumenten-MUX **305** gibt Signale auf vier Instrumentenbussen **306** aus. Der eingebettete Computer **206** weist den Instrumenten-MUX **304** über eine Steuerleitung **314** an, welche Instrumentsignale auf den Instrumentbussen **306** platziert werden sollen. Die Instrumentbusse **306** werden dann zu einer Mehrzahl von Kanälen über einen Kanal-MUX **308** multiplexiert.

[0035] Der eingebettete Computer **206** steuert, welche Signale aus dem Kanal-MUX **308** ausgegeben werden, über eine Steuerleitung **316**. Die Ausgänge des Kanal-MUX **308** weisen DUT-Hoch-Leitungen **311** und DUT-Niedrig-Leitungen **310** auf. Diese Hoch- und Niedrig-Leitungen werden durch ein Instrument **308** erzeugt. Die Hoch- und Niedrig-Leitungen werden während des Testens als Eingänge in den Testkanal **312**, was in Fig. 4 detaillierter gezeigt ist, verwendet. Wenn eine bestimmte Schaltungsplatine getestet wird, d.h. eine DUT, ist die Anzahl von Kanälen, die durch den eingebetteten Computer **206** verwendet werden, im Allgemeinen gleich der Anzahl zu testender Knoten auf der Schaltungsplatine.

[0036] Nach dieser allgemeinen Beschreibung von Architektur und Umgebung, die vielen Testern gemein ist, wird nun Bezug auf Fig. 4 genommen, die ein Blockdiagramm bestimmter Komponenten einer Schaltungsplatine **400** eines HP83000-Testers ist, der der bevorzugte Plattform-Tester ist, für den das bevorzugte Ausführungsbeispiel der vorliegenden Erfindung entworfen wurde. Wie dargestellt ist, umfasst die Schaltungsplatine **400** eine Mehrzahl von Treibern **402** und Empfängern **404**. Wie bekannt ist, sind die Treiber **402** konfiguriert,

um Signale an Anschlussstifte eines Testverbinders auszugeben. Auf der HP83000 gibt es sechzehn Treiber und sechzehn Empfänger auf einem einzelnen Platine/Kanal-Element. Eine einzelne Parametermesseinheit PMU **406** jedoch ist auf der Platine **400** vorgesehen. Wie bekannt ist, ist die PMU **406** konfiguriert, um Signale zu quantifizieren oder zu messen. In Betrieb schafft die PMU **406** eine Kalibrierung für die Treiber **402** und die Empfänger **404**.

[0037] Eine Anzahl von Relais ist in der Zeichnung ebenso dargestellt und diese sorgen zusammen für eine dynamische Platinenkonfiguration. Insbesondere verbindet und trennt ein Relais **408** selektiv die Treiber **402** und Empfänger **404** bezüglich einer Kontinuität mit dem Testverbinder. Ein weiteres Relais **410** richtet eine selektive Verbindung zu dem Testverbinder und einem weiteren Pfad auf der Platine **400** ein; zum Beispiel mit der PMU. Es ist ferner zu erkennen, dass Abschnitte der vorliegenden Erfindung die Relaissteuersignale so steuern, um die Platine **400** dynamisch in der geeigneten Weise zum Ausführen der Diagnosefunktionen der vorliegenden Erfindung zu konfigurieren. Ferner ist, obwohl nur eines jedes Relais in der Zeichnung dargestellt ist, zu erkennen, dass es mehrere Relais geben könnte. Es gibt zum Beispiel ein Relais **408** für jedes der sechzehn Treiber/Empfänger-**402**/**404**-Paare.

[0038] Bei Normalbetrieb ist der Testverbinder über eine Verkabelung oder anderweitig mit einer zu testenden Vorrichtung (DUT) verbunden. So treiben durch die Treiber **402** erzeugte Signale letztendlich Eingänge von Komponenten auf einer DUT-Platine. Ähnlich werden Ausgangssignale von Komponenten auf einer DUT-Platine über den Testverbinder an die Empfänger **404** geliefert. Durch ein steuerbares Variieren des Ausgangspegels der Treiber **402** und ein Überwachen der Eingangspegel der Empfänger **404** könnte ein Funktionsbetrieb einer DUT getestet werden. Zusätzlich zu einem Verifizieren des Funktionsbetriebs einer DUT könnten Tester auch verifizieren, ob die Komponenten auf der DUT gemäß spezifizierten Toleranzwerten oder -bereichen arbeiten. Wie zuvor beschrieben wurde, kann dieser Typ Testen jedoch falsche oder ungenaue Ergebnisse erzeugen, wenn die Komponenten auf dem Tester ausfallen oder anderweitig außerhalb der spezifizierten Toleranz geraten.

[0039] Wie zuvor erläutert wurde, kennt man ein Verfahren zum Messen einer Pegelgenauigkeit, das die Verwendung externer Messausrüstung, wie zum Beispiel eines Voltmeters, um Genauigkeitsdaten zu sammeln, beinhaltet. Während dieses Verfahren einen relativ hohen Grad an Genauigkeit erzielt, macht es erforderlich, dass ein Benutzer manuell jeden Testerkanal für sieben unterschiedliche Messsätze sondiert. In Serienproduktions-Herstellungsumgebungen ist dieses externe Messvorrichtungsverfahren inakzeptabel.

[0040] Entsprechend ist die vorliegende Erfindung zum Testen und Bestimmen, ob bestimmte Testkomponenten außerhalb einer Toleranz liegen, vorgesehen. Gemäß der vorliegenden Erfindung wird der Tester verwendet, um seine eigene Genauigkeit zu messen, und erfordert eine viel einfachere DUT-Platine und sehr wenig Benutzerinteraktion zur Erfassung von Daten. Wie zu erkennen sein wird, ist dieses Verfahren eine viel bessere Wahl, da es eine Beeinflussung der Herstellungsumgebung minimiert und dennoch eine ausreichende Messgenauigkeit bereitstellt.

[0041] Kurz Bezug nehmend auf Fig. 5 wird eine speziell konfigurierte DUT-Platine **500** in Verbindung mit den Eigentests, die durch das bevorzugte Ausführungsbeispiel der vorliegenden Erfindung durchgeführt werden, verwendet. Insbesondere schließt diese spezialisierte DUT-Platine **400** Anschlussstifte von Testverbindern **502** und **504** miteinander kurz. Wie zuvor beschrieben wurde, umfasst der Tester einen Testkopf **205**, durch den alle elektrischen Verbindungen zu dem Tester eingerichtet werden. Der Testkopf **205** umfasst eine Mehrzahl von Verbindern, wie zum Beispiel Testverbinder **502** und **504**. Kanäle 01 bis 16 sind auf jedem dieser Testverbinder **502** und **504** dargestellt. Der Kanal 01 entspricht einem ersten Treiber/Emffänger-**402**/**404**-Paar auf der Schaltungsplatine **400**. Diese speziell konfigurierte DUT-Platine **500** des bevorzugten Ausführungsbeispiels beinhaltet leitfähige Pfade (z.B. **506**), die Kanalleitungen miteinander kurzschließen, zwischen benachbarten Testverbindern **502** und **504**. Auf diese Weise könnte ein Treiber von einer ersten Schaltungskarte **400** (zum Beispiel einer Karte, die dem Testverbinder **502** entspricht) mit einem Empfänger einer zweiten Schaltungskarte (zum Beispiel einer Karte, die dem Testverbinder **504** entspricht) verbunden sein. Natürlich steuert das bevorzugte Ausführungsbeispiel die Relaissteuersignale auf den jeweiligen Schaltungsplatinen **400**, um so zu bewirken, dass die geeigneten Signalpfade diese Verbindung einrichten. Bei dem bevorzugten Ausführungsbeispiel haben alle leitfähigen Pfade (z.B. **506**) auf der DUT-Platine **500** die gleiche Länge.

[0042] Das Metrologiesteuertool des bevorzugten Ausführungsbeispiels misst eine Treiber- und Empfangs-Spannungspegelgenauigkeit. Das bevorzugte Verfahren zum Verwenden eines Testers zur Messung seiner eigenen Treiberpegelgenauigkeit besteht darin, einen spezifizierten Treiberpegel für alle Kanäle zu programmieren. Der tatsächliche Treiberpegel, der durch jeden Kanal bereitgestellt wird, wird dann mit der plati-

neninternen Spannungsmessvorrichtung des Testers (d.h. der PMU **406**) gemessen. Der Unterschied zwischen dem programmierten Pegel und dem gemessenen Pegel ist der Treiberpegelfehler. Dieser Test ist vorzugsweise auf dem Tester durch ein Konfigurieren einer Hälfte der Testerkanäle als Eingänge und der anderen Hälfte als Ausgänge implementiert. Die DUT-Platine **500** ist die Schnittstelle zwischen Eingängen und Ausgängen. Die Eingänge sind programmiert, um einen Pegel zu treiben, und eine Digitalspannungsmess- (DVM-; DVM = digital voltage measurement) Testfunktion, die die PMU **406** der Platine **400** als ein Voltmeter verwendet, misst die tatsächlichen Pegel. Zum Messen von Treiberpegelfehlern für die zweite Hälfte von Testerkanälen werden Kanalkonfigurationen ausgetauscht und der Test wird wiederholt.

[0043] Insbesondere stellt Fig. 6A die Platinenkonfigurationen zweier Schaltungsplatinen **400a** und **400b** dar. Wie gezeigt ist, sind die verschiedenen Relais konfiguriert, um die Treiber **402a** der Platine **400a** mit der PMU **406b** der Platine **400b** zu verbinden. Wie zuvor erwähnt wurde, ist es, da die PMU **406a** verwendet wird, um die Treiber **402a** auf dieser gleichen Platine **400a** zu kalibrieren, wesentlich, dass die vorliegende Erfindung den Tester durch ein Verifizieren von Signalpegeln von den Treibern **402a** mit der PMU **406b** einer anderen Platine testet.

[0044] Entsprechend wird die Schaltungskonfiguration aus Fig. 6A verwendet, um die Genauigkeit von Treiberkomponenten zu verifizieren. Diesbezüglich wird ein Treiber **402a** gesteuert, um einen vorbestimmten „Ziel"-Wert auszugeben. Der tatsächliche Wert, der durch die PMU **406b** empfangen und gemessen wird, bestimmt, oder der Treiber **402a** außerhalb einer Spezifizierung ist. Beispielhaft und unter Verwendung hypothetischer Werte könnte ein Steuersignal an den Treiber **402a** angelegt werden, um denselben bei einem Zielpegel von 6500mV zu treiben. Wenn die PMU **406b** einen tatsächlichen Wert von 6300mV empfängt und misst, wird bestimmt, dass der Treiber **402a** um 200mV versetzt ist. Wenn die ATE-Spezifizierungen für den Treiber es erfordern, dass die Ausgabe zumindest 6400mV beträgt, würde bestimmt werden, dass der Treiber **402** um 100mV außerhalb der Spezifizierung liegt.

[0045] Gemäß dem bevorzugten Ausführungsbeispiel könnten die Treiber **402a** einer Platine einzeln (einer zu einer Zeit) die PMU **406b** einer anderen Platine treiben. Dann könnten die Relais **408**, **410** umgedreht werden, so dass die Treiber **402b** der zweiten Platine mit der PMU **406a** der ersten Platine verbunden und getestet werden könnten. Auf diese Weise könnten alle Treiber von beiden Platinen getestet werden. Dieser Treiberverifizierungsschritt wird durchgeführt, um die Genauigkeit der Treiber sicherzustellen, bevor die Genauigkeit der Empfänger getestet wird, wie unten beschrieben ist.

[0046] Ein Messen der Genauigkeit des Empfängers impliziert einen komplexeren Test, da Tester nicht konfiguriert sind, um ihre eigenen Empfangspegel direkt zu messen. Um eine Empfangspegelgenauigkeit an dem Tester des bevorzugten Ausführungsbeispiels zu messen, ist eine Hälfte der Testerkanäle als Eingänge konfiguriert und die andere Hälfte als Ausgänge konfiguriert. Die DUT-Platine **500** wird als die Schnittstelle zwischen Eingängen und Ausgängen verwendet. Eingänge werden zum Treiben eines Pegels programmiert und Empfangspegel werden um diesen Treiberpegel herum variiert. Diesbezüglich weist der Empfänger **404** einen Komparator auf. Anstelle eines Erzeugens eines Ausgangs-„Werts" ist die Komparatorausgabe effektiv binär (was anzeigt, dass die Komparatoreingabe – oder der empfangene Wert – entweder oberhalb oder unterhalb eines Referenzwerts ist). Der Referenzwert, der an dem Referenzeingang an den Komparator geliefert wird, könnte variiert werden, bis die Komparatorausgabe einen Zustand verändert, wobei an diesem Punkt der Wert der Referenzeingabe im Wesentlichen gleich dem Wert der empfangenen Eingabe ist. Es wird zum Beispiel angenommen, dass der Empfänger bei einer bekannten Treiberspannung von 500mV getestet wird. Die Komparatorreferenzspannung könnte steuerbar zwischen 450mV und 550mV variiert werden. Während die Referenzspannung kleiner ist als diejenige der getriebenen/empfangenen Spannung, könnte die Ausgabe des Komparators in einem ersten Zustand sein (z.B. null). Sobald jedoch die Referenzspannung die getriebene/empfangene Spannung überschreitet, könnte die Ausgabe des Komparators zu einem zweiten Zustand getrieben werden (z.B. Sättigung oder logische Eins). Wenn diese Zustandsveränderung erfasst wird, ist die Referenzspannung im Wesentlichen gleich der empfangenen Spannung.

[0047] In der vorstehenden Weise könnte man sagen, dass der Empfänger **404** den Wert des empfangenen Signals „interpretiert".

[0048] Für jede Empfangspegeleinstellung wird ein Funktionstest ausgeführt. Empfangspegelfehler werden für jeden Kanal durch ein Nehmen der Differenz zwischen dem tatsächlichen Treiberpegel und dem Empfangspegel (interpretiert), an dem der Funktionstest nicht mehr bestanden wurde (oder ab dem er bestanden wurde, abhängig von dem Test), berechnet. Zum Messen von Empfangspegelfehlern für die zweite Hälfte von Testerkanälen werden Kanalkonfigurationen ausgetauscht und der Test wird wiederholt.

[0049] Nun wird Bezug auf Fig. 6B genommen, dort sind die Relais/Platine-Konfigurationen zweier Schaltungsplatinen **400a** und **400b**, die zum Testen einer Empfängergenauigkeit konfiguriert sind, dargestellt. Insbesondere sind die Relais konfiguriert, um die Treiber **402c** mit den Empfängern **404d** zu verbinden. Ein Initiieren dieses Tests nimmt an, dass die Treiber **402c** zuvor auf Genauigkeit getestet wurden, entweder durch den oben beschriebenen Treibertest oder durch ein Verwenden einer externen Testausrüstung zum Verifizieren der Genauigkeit der Treiberpegel der Treiber **402c**. Danach werden die Treiber **402c** konfiguriert, um ein Signal auf einem vorbestimmten Treiberpegel auszugeben. Der Empfänger **404d** wird dann ausgewertet, ob der von demselben empfangene und ausgegebene Wert innerhalb einer spezifizierten Toleranz des getriebenen Signals ist.

[0050] Vorzugsweise werden die oben beschriebenen Treiber- und Empfangspegelgenauigkeitstests bei Pegeln von 500mV und 6500mV geprüft, da diese Pegel die Einstellungen eines ungünstigsten Falls für den bevorzugten Tester darstellen, die noch in die HP83000-Pegelspezifizierungen fallen. Die gleichen Messungen werden bei 500mV und 6500mV für sowohl niedrige als auch hohe Pegel genommen, um mögliche Unterschiede zwischen einem Treiben und einem Empfangen eines Pegels als ein „hoch" und ein „niedrig" zu verfolgen. Zusätzlich wird für alle Messungen ein Ausgangshub (unten erläutert) auf 500mV gesetzt.

[0051] Zur Untersuchung der Robustheit und Nützlichkeit des oben beschriebenen Metrologiesteuertools und um mehr über das Treiber/Empfänger-Verhalten des HP83000 zu lernen, wurden mehrere Experimente aufgebaut, die die HP83000-Pegelspezifizierungen prüften und herausforderten. Zuerst wurde die DUT-Platine **500**, oben beschrieben, untersucht, um zu bestimmen, ob Entwurfs- oder Herstellungsanomalien vorlagen, die ausreichend wesentlich waren, um Treiber- und Empfangspegelgenauigkeitsmessungen zu beeinflussen. Das Experiment beinhaltete ein Nehmen von Treiberpegelfehlermessungen, ein darauf folgendes Entfernen der DUT-Platine **500** und ein erneutes Durchführen der gleichen Messungen. Die ohne die DUT-Platine durchgeführten Messungen verwendeten die PMUs **406** auf der Schaltungsplatine **400** als die Kanäle, deren Treiberpegel gerade gemessen wurden, um Treiberpegel zu messen. Ergebnisse zeigten eine Differenz von weniger als 1mV zwischen den beiden Messverfahren, so dass die DUT-Platine als geeignete betrachtet wurde.

[0052] Eine Einstellzeit und ihre Wirkungen auf eine Treiberpegelgenauigkeit bei Messung mit dem oben beschriebenen Verfahren wurden ebenso untersucht. Für Einstellzeiten von 0ms bis 500ms wurden Treiberpegel variiert und gemessen. Ergebnisse zeigten eine Differenz von weniger als 1mV zwischen gemessenen Treiberpegeln für Einstellzeiten von 0ms und 500ms. Deshalb wurde bestimmt, dass die Einstellzeit bei dem ausgewählten Verfahren zum Messen der Treiberpegelgenauigkeit kein Faktor war.

[0053] Auf Grund ihrer möglichen Auswirkung auf Empfangspegelgenauigkeitsmessungen wurde eine Treiberpegelwiederholbarkeit untersucht. Fünf Sätze von Treiberpegelmessungen wurden innerhalb eines Zeitraums von fünf Minuten genommen (die geschätzte Zeit, die erforderlich war, um Treiber- und Empfangspegelgenauigkeitsmessungen in der letzten Produktionsversion des Tools durchzuführen). Ergebnisse zeigten, dass Treiberpegel innerhalb 1mV wiederholbar waren, was verglichen mit den erwarteten Empfangspegelfehlern von +/–25mV und +/–50mV unwesentlich ist.

[0054] Die HP83000 „Output Swing"- bzw. „Ausgangshub"-Spezifizierung wurde untersucht, um die Spezifizierung zu verifizieren und ihre Wirkungen auf das Treiberpegelgenauigkeitsmessverfahren zu verstehen. „Ausgangshub" ist als die Entfernung zwischen der Niedrigpegeleinstellung und der Hochpegeleinstellung definiert. Die HP83000-Ausgangshub-Spezifizierung beträgt 500mV bis 6000mV. Dies bedeutet, dass, damit alle anderen Treiberspezifizierungen garantiert sind, die Differenz zwischen Niedrig- und Hoch-Treiberpegeleinstellungen nicht weniger als 500mV und nicht mehr als 6000mV betragen sollte. Es wird jedoch angemerkt, dass ein Ausgangshub auf zwischen 100mV und 7000mV programmiert sein kann. Das Experiment bestand darin, den Ausgangshub von 100mV bis 6500mV zu variieren und Treiberpegelfehler für jede Ausgangshubeinstellung zu messen. Ergebnisse zeigten, dass Treiberpegelfehler minimiert waren, als der Ausgangshub 500mV oder mehr betrug, was die HP83000-Spezifizierung bestätigte. Ausgangshübe von mehr als 6000mV schienen keine Wirkung auf die Treiberpegelgenauigkeit zu haben.

[0055] Da der niedrigste Treiberpegel, der durch das Pegelmetrologiesteuerwerkzeug bzw. -tool gemessen wurde, 500mV betrug, wurde ein Experiment durchgeführt, um eine Genauigkeit für Treiberpegel von weniger als 500mV zu überprüfen. Treiberpegel wurden von 500mV herunter bis auf –1500mV programmiert und ihre Fehler gemessen. Ergebnisse zeigten, dass Fehler für 500mV-Treiberpegel innerhalb von 1 bis 2mV von Fehlern für Treiberpegel von weniger als 500mV waren.

[0056] Ein weiteres Experiment untersuchte, wie der Treiber/Empfangs-**402**/**404**-Schaltungsaufbau, der sich

auf der Messseite der Treiberpegelgenauigkeitsschaltung befindet, Treiberpegelfehler beeinflusste. Um den unnötigen Treiber/Empfangs-**402**/**404**-Schaltungsaufbau zu isolieren, wurde das AC-Relais **408** (siehe Fig. 4) auf der Messseite der Schaltung geöffnet und Treiberpegelfehler wurden gemessen. Diese Daten wurden mit Treiberpegelfehlern verglichen, die mit geschlossenem AC-Relais gemessen wurden, was die Vorgabeposition für das Relais während der DVM-Testfunktion ist. Ergebnisse zeigten, dass der Treiber/Empfangs-Schaltungsaufbau, der sich auf der Messseite der Treiberpegelgenauigkeitsschaltung befand, zusätzliche 5–10mV Fehler für Treiberpegel, die auf 6500mV programmiert waren, beitrug. Dies ist angesichts des erwarteten Fehlers von +/–50mV für einen 6500mV-Treiberpegel beträchtlich. Obwohl das Treiberpegelgenauigkeitsmessverfahren, das oben beschrieben ist, für beide Relaispositionen gut funktioniert, liefert ein Öffnen des Relais **408** auf der Messseite der Schaltung eine bessere Genauigkeit. Die letztendliche Version des Pegelmetrologiesteuertools nimmt Treiberpegelfehlerdaten mit offenen Relais **408**.

[0057] Nun wird kurz Bezug auf Fig. 7 genommen, die ein Betriebsflussdiagramm der Hauptschritte zeigt, die bei dem Verfahren zum Verifizieren der Treiberoperation beinhaltet sind. Insbesondere führt nach einem Hochfahren das Verfahren eine Initialisierungsroutine **702** durch, die ein Initialisieren von Variablen und Durchführen einer Vielzahl von Initialkonfigurationsaufgaben umfasst, die zwangsläufig von Tester zu Tester und Implementierung zu Implementierung variieren und deshalb hierin nicht erläutert werden müssen. Nach einer Initialisierung bestimmt und konfiguriert das Verfahren die verschiedenen Testkanäle zur Durchführung des Treibertests (Schritt **706**). Dann führt es den Test für V_{IL} durch (Schritt **708**). Das Verfahren führt dann den Test für V_{IH} durch (Schritt **710**).

[0058] Schließlich wertet bei Schritt **712** das Verfahren die Ergebnisse aus, um zu bestimmen, ob die Treiber **402** innerhalb einer spezifizierten Toleranz sind oder nicht. Es ist jedoch zu erkennen, dass in Verbindung mit diesem Test verschiedene Korrekturhandlungsmaßnahmen unternommen werden könnten. Zum Beispiel könnte geeignetes Personal über die durchgefallenen Ergebnisse benachrichtigt werden, durch E-Mail (wenn der Tester mit einem LAN verbunden ist) oder anderweitig, so dass geeignetes Personal Korrekturhandlungen durchführen kann.

[0059] Nun wird kurz Bezug auf Fig. 8 genommen, die ein Betriebsflussdiagramm der Hauptschritte zeigt, die bei dem Verfahren zum Verifizieren des Empfängerbetriebs beinhaltet sind. Insbesondere könnte das Verfahren nach einem Hochfahren eine Initialisierungsroutine **802** durchführen, die ein Initialisieren von Variablen und Durchführen einer Vielzahl von Initialkonfigurationsaufgaben umfasst, die zwangsläufig von Tester zu Tester und Implementierung zu Implementierung variieren und deshalb hierin nicht erläutert werden müssen. Es ist jedoch zu erkennen, dass dieser Schritt ausgelassen werden könnte, wenn bereits eine Initialisierung durchgeführt wurde, wie durch Schritt **702** (Fig. 7). Nach einer Initialisierung bestimmt und konfiguriert das Verfahren die verschiedenen Testerkanäle zur Durchführung des Empfängertests (Schritt **806**). Es führt dann den Test für V_{OL} durch (Schritt **808**). Das Verfahren führt dann den Test für V_{OH} durch (Schritt **810**).

[0060] Bei dem bevorzugten Ausführungsbeispiel ist eine Routine vorgesehen, um eine graphische Ausgabe zu erzeugen, die diese Testergebnisse darstellt (Schritt **811**). Schließlich wertet das Verfahren bei Schritt **812** die Ergebnisse aus, um zu bestimmen, ob die Empfänger **404** innerhalb einer spezifizierten Toleranz sind oder nicht. Es ist zu erkennen, dass in Verbindung mit diesem Test die verschiedenen Korrekturhandlungsmaßnahmen, die in Verbindung mit Fig. 7 erwähnt wurden, auch hier unternommen werden können.

[0061] Die vorstehende-Beschreibung wurde zu Darstellungs- und Beschreibungszwecken vorgelegt. Sie soll nicht ausschließlich sein oder die Erfindung auf die genauen offenbarten Formen einschränken. Offensichtliche Modifizierungen oder Variationen sind angesichts der obigen Lehren möglich. Das oder die Ausführungsbeispiele, die erläutert wurden, wurden ausgewählt und beschrieben, um die beste Darstellung der Prinzipien der Erfindung und ihrer praktischen Anwendung bereitzustellen, wodurch es einem durchschnittlichen Fachmann auf dem Gebiet ermöglicht wird, die Erfindung in verschiedenen Ausführungsbeispielen und mit verschiedenen Modifizierungen einzusetzen, wie es für die bestimmte anvisierte Verwendung geeignet ist. Alle derartigen Modifizierungen und Variationen sind innerhalb des Schutzbereichs der Erfindung, wie durch die beigefügten Ansprüche bestimmt wurde, bei Interpretation gemäß der Breite, auf die diese klar und rechtmäßig Anspruch haben.

[0062] Nach einer Beschreibung verschiedener Aspekte der vorliegenden Erfindung sind direkt unten die codierten Instruktionen angeführt, die verwendet werden, um Abschnitte des gegenwärtig bevorzugten Ausführungsbeispiels der Erfindung auszuführen.

```
/*******************************************************
*              Funktion: dr_lev – misst Treiberpegelfehler          *
*******************************************************/
void dr_lev()
{
int error;
TF_PIN_RESULT pin_result[1025];
double tf_result[4];
long task_len;
char task[1024];
long answer_len=1024;
long ret_state;
long tf_state;
int i;
int k;
int channel;
int lev;
char low_level_string[100];
char high_level_string[100];
char answer[1024];
```

```
int failed_dr_level[1025];
int failed_dr_hi_low[1025];
float failed_result[1025];
int failed_channel[1025];
int g;
total_p=1025;
strcpy(device_path,"/fedisk2/cal/");

/*******************************************************/
        /* Durchführen von Vil-Messungen mit DUT-Platine. */
/*******************************************************/
for (k=1; k<=2; k++)
  {
  if (k==1)
    {
    /* Laden eines Datensatzes für ersten Satz von Anschlussstiften --> 10101, 10301, etc. */
    DataManager(dm_set, dm_load, device_path, "dr_levels_a.mpc", 0, &tf_state);
    }
  else
    {
    /* Laden eines Datensatzes für zweiten Satz von Anschlussstiften --> 10201, 10401, etc. */
    DataManager(dm_set, dm_load, device_path, "dr_levels_b.mpc", 0, &tf_state);
    }
if (tf_state != 0) /* Wenn Datensatz-Laden fehlschlägt ... */
    {
    printf(„FEHLER! DataManager-Laden des Datensatzes fehlgeschlagen.\n\n“);
    if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
    HpTerm();
    exit(CI_CALL_ERROR);
    }
for (lev=500; lev<=6500; lev=lev+6000)
    {
     /* Setzen von Pegeln */
    strcpy(task, "DRLV 1,");
    sprintf(low_level_string,"%d",lev);
    sprintf(high_level_string,"%d",lev+500);
    strcat(task,low_level_string);
    strcat(task,",");
    strcat(task,high_level_string);
    strcat(task,", (all_inputs);");
    task_len = strlen(task);
    HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
```

```
answer[1023] = '\0';
if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt ... */
  {
  printf(„\n\nDRLV-Firmwarebefehl fehlgeschlagen!!!\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
/* Firmware-Befehlssequenz zum Öffnen von AC-Relais auf Empfangs-ckt. */
if (error=connect()!=0)
    {
    printf(„FEHLER!! Ruf nach Verbindungsfunktion fehlgeschlagen.\n\n“);
  HpTerm();
    exit(CI_CALL_ERROR);
    }
/* Öffnen des AC-Relais */
strcpy(task,"RLYC IDLE,PMU,(all_outputs);");
task_len = strlen(task);
HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
answer[1023] = '\0';
if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt ... */
  {
  printf(„\n\n erster RLYC-Firmwarebefehl fehlgeschlagen!!! \n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
/* Ausführen einer DRLV-Pegelgenauigkeitsprüftestfunktion. */

ExecTfParam("dvm;all_outputs;V;0;V;7;ms;10;DVM ($P)",tf_result,&tf_state);
if (tf_state == -1 ) /* Wenn Testfunktionsausführung fehlschlägt .. */
  {
  printf(„FEHLER! Ausführung der Testfunktion fehlgeschlagen.\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
GetPinResults(pin_result, &total_p, &tf_state);
if (tf_state == 2)
  {
```

```
for ( i=0; i<total_p; i++ )
  {
   if (lev==500 && k==1)
     {
     /* Speichern der Treiberkanalnummer */
     channel = atoi(pin_result[i].pinname);
       if (channel >13100 && channel < 13300) drlv_channel_a[i]=channel-1;
     else drlv_channel_a[i] = channel - 100;
       /* Speichern des Ergebnis */
       vil_500a[i] = pin_result[i].meas[0].value;
     }
   else if (lev==500 && k==2)
     {
      /* Speichern der Treiberkanalzahl */
       channel = atoi(pin_result[i].pinname);
       if (channel >13100 && channel < 13300) drlv_channel_b[i]=channel+1;
     else drlv_channel_b[i] = channel + 100;
       /* Speichern des Ergebnis */
       vil_500b[i] = pin_result[i].meas[0].value;
     }
     else if (lev==6500 && k==1) vil_6500a[i] = pin_result[i].meas[0].value;
   else if (lev==6500 && k==2) vil_6500b[i] = pin_result[i].meas[0].value;
   }
  } /* schließt, wenn tf_state = 2 */
 /* Ausführen der DRLV-Pegelgenauigkeitsprüftestfunktion. Diese Testfunktion misst
 DRLVs mit einer eigenen PMU jedes Treibers. Gleicher Test wie der Keine-DUT-Pla-
 tinentest, jedoch mit einer DUT-Platine auf dem Testkopf. Je mehr Daten desto besser,
 oder? */
ExecTfParam("dvm;all_inputs;V;0;V;7;ms;10;DVM ($P)",tf_result,&tf_state);
if (tf_state == -1 )  /* Wenn Testfunktionsausführung fehlschlägt ... */
  {
  printf(„FEHLER! Ausführung der Testfunktion fehlgeschlagen.\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
GetPinResults(pin_result, &total_p, &tf_state);
if (tf_state == 2)
  {
  for ( i=0; i<total_p; i++ )
    {
     if (lev==500 && k==1)
```

```
            {
              /* Speichern des Ergebnis */
              vil_500a3[i] = pin_result[i].meas[0].value;
            }
          else if (lev==500 && k==2)
            {
              /* Speichern des Ergebnis */
              vil_500b3[i] = pin_result[i].meas[0].value;
            }
            else if (lev==6500 && k==1) vil_6500a3[i] = pin_result[i].meas[0].value;
          else if (lev==6500 && k==2) vil_6500b3[i] = pin_result[i].meas[0].value;
          }
        }/* schließt, wenn tf_state = 2 */
      } /* schließt Zweites für Schleife. */
    } /* schließt Erstes für Schleife. */
/*****************************************************/
          /* Durchführen von Vih-Messungen mit DUT-Platine. */
/*****************************************************/
for (k=1; k<=2; k++)
  {
  if (k==1)
    {
    /* Laden eines Datensatzes für ersten Satz von Anschlussstiften --> 10101, 10301, etc. */
    DataManager(dm_set, dm_load, device_path, "dr_levels_a.mpc", 0, &tf_state);
    if (tf_state != 0)  /* Wenn Datensatz-Laden fehlschlägt ... */
      {
      printf(„FEHLER! DataManager-Laden des Datensatzes fehlgeschlagen.\n\n“);
      if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
      HpTerm();
      exit(CI_CALL_ERROR);
      }
    /* Laden von Vih_Zeitgebung. */
    DataManager(dm_timing, dm_load, device_path, "timinga1.dr_levels_mpc", 0,
&tf_state);
    }
  else
    {
    /* Laden eines Datensatzes für zweiten Satz von Anschlussstiften --> 10201, 10401, etc. */
    DataManager(dm_set, dm_load, device_path, "dr_levels_b.mpc", 0, &tf_state);
    if (tf_state != 0) /* Wenn Datensatz-Laden fehlschlägt, ... */
      {
      printf(„FEHLER! DataManager-Laden des Datensatzes fehlgeschlagen.\n\n“);
```

```
        if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
        HpTerm();
        exit(CI_CALL_ERROR);
        }
      /* Laden von Vih_Zeitgebung. */
      DataManager(dm_timing, dm_load, device_path, "timingb1.dr_levels_mpc", 0,
&tf_state);
      }
    if (tf_state != 0)  /* Wenn Zeitgebungsladen fehlschlägt ... */
      {
      printf(„FEHLER! DataManager-Laden der Zeitgebung fehlgeschlagen.\n\n“);
      if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
      HpTerm();
      exit(CI_CALL_ERROR);
      }
    for (lev=500; lev<=6500; lev=lev+6000)
      {
      strcpy(task, "DRLV 1,");
      sprintf(low_level_string,"%d",lev-500);
      sprintf(high_level_string,"%d",lev);
      strcat(task,low_level_string);
      strcat(task,",");
      strcat(task,high_level_string);
      strcat(task,", (all_inputs);");
      task_len = strlen(task);
      HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
      answer[1023] = '\0';
      if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt ... */
        {
        printf(„\n\nDRLV-Firmwarebefehl fehlgeschlagen!!!\n\n“);
        if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
        HpTerm();
        exit(CI_CALL_ERROR);
        }
      /* Firmware-Befehlssequenz zum Öffnen von AC-Relais auf Empfangs-ckt. */
      if (error=connect()!=0)
          {
          printf(„FEHLER!! Ruf nach Verbindungsfunktion fehlgeschlagen.\n\n“);
        HpTerm();
          exit(CI_CALL_ERROR);
```

```
        }
    /* Öffnen von AC-Relais */
    strcpy(task,"RLYC IDLE,PMU,(all_outputs);");
    task_len = strlen(task);
    HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
    answer[1023] = '\0';
    if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt ... */
      {
      printf("\n\n erster RLYC-Firmwarebefehl fehlgeschlagen!!! \n\n");
      if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
      HpTerm();
      exit(CI_CALL_ERROR);
      }
    /* Ausführen von DRLV-Pegelgenauigkeitsprüftestfunktion. */
    ExecTfParam("dvm;all_outputs;V;0;V;7;ms;10;DVM ($P)",tf_result,&tf_state);
    if (tf_state == -1 ) /* Wenn die Testfunktionsausführung fehlschlägt ... */
      {
      printf("FEHLER! Ausführung der Testfunktion fehlgeschlagen.\n");
      if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
      HpTerm();
      exit(CI_CALL_ERROR);
      }
    GetPinResults(pin_result, &total_p, &tf_state);
    if (tf_state == 2)
      {
      for ( i=0; i<total_p; i++ )
        {
        if (lev==500 && k==1) vih_500a[i] = pin_result[i].meas[0].value;
          else if (lev==500 && k==2) vih_500b[i] = pin_result[i].meas[0].value;
          else if (lev==6500 && k==1) vih_6500a[i] = pin_result[i].meas[0].value;
          else if (lev==6500 && k==2) vih_6500b[i] = pin_result[i].meas[0].value;
        } /* schließt Drittes für Schleife. */
      } /* schließt, wenn tf_state-Schleife. */
    /* Ausführen der DRLV-Pegelgenauigkeitsprüftestfunktion. Diese Testfunktion misst
    DRLVs mit einer eigenen PMU jedes Treibers. Gleicher Test wie der Keine-DUT-Platinentest, jedoch mit einer DUT-Platine auf dem Testkopf. Je mehr Daten desto besser,
    oder? */
    ExecTfParam("dvm;all_inputs;V;0;V;7;ms;10;DVM ($P)",tf_result,&tf_state);
    if (tf_state == -1 )  /* Wenn Testfunktionsausführung fehlschlägt... */
      {
      printf("FEHLER! Ausführung der Testfunktion fehlgeschlagen.\n");
```

```
        if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
        HpTerm();
        exit(CI_CALL_ERROR);
        }
      GetPinResults(pin_result, &total_p, &tf_state);
      if (tf_state == 2)
        {
        for ( i=0; i<total_p; i++ )
          {
          if (lev==500 && k==1) vih_500a3[i] = pin_result[i].meas[0].value;
            else if (lev==500 && k==2) vih_500b3[i] = pin_result[i].meas[0].value;
            else if (lev==6500 && k==1) vih_6500a3[i] = pin_result[i].meas[0].value;
            else if (lev==6500 && k==2) vih_6500b3[i] = pin_result[i].meas[0].value;
          }
        }
        } /* schließt Zweites für Schleife. */
      } /* schließt Erstes für Schleife. */
    if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
    }
    /************************************************************
    * Funktion: rc_lev – misst Empfangspegel-Bestehen/Durchfallen-Punkte   *
    ************************************************************/
    void rc_lev (void)
    {
    int shmoo_2d=0; /* 0 = kein Shmoo 1 = Ausgangsshmoo */
    TF_PIN_RESULT pin_result[525];
    double tf_result[4];
    int error;
    char task[1024];
    char low_level_string[100];
    char high_level_string[100];
    char answer[1024];
    long task_len;
    long answer_len=1024;
    long ret_state;
    long tf_state;
    int i;
    int j;
    int cnt = 0;
    int vol_500a[525][150];
    int vol_500b[525][150];
```

```
int vol_6500a[525][150];
int vol_6500b[525][150];
int voh_500a[525][150];
int voh_500b[525][150];
int voh_6500a[525][150];
int voh_6500b[525][150];
int k;
int l;
int flag=0;
float first_pass;
float last_pass;
int low_lev;
int high_lev;
int rclv_vol_500a[525];
int rclv_vol_500b[525];
int rclv_vol_6500a[525];
int rclv_vol_6500b[525];
int rclv_voh_500a[525];
int rclv_voh_500b[525];
int rclv_voh_6500a[525];
int rclv_voh_6500b[525];
char set_compare[50];
double E;
float d_i;
total_p=1025;
strcpy(device_path,"/fedisk2/cal/");
/***************** Beginn Pegelgenauigkeitsmessungen ****************/
printf(„--> Empfangspegeldaten nehmen...\n\n“);
fflush(stdout);
/*********************************************/
/*                  Vol-Messungen            */
/*********************************************/
for (k=0; k<=3; k++)
  {
  /********** Datensätze Laden *********/
  if( (k==0) || (k==1) )
    {
    DataManager(dm_set, dm_load, device_path, "rc_levels_a.mpc", 0, &tf_state);
    }
  else if ( (k==2) || (k==3) )
    {
    DataManager(dm_set, dm_load, device_path, "rc_levels_b.mpc", 0, &tf_state);
    }
```

```
if (tf_state != 0)    /* Wenn Datensatz-Laden fehlschlägt... */
  {
  printf(„FEHLER! DataManager-Laden des Datensatzes fehlgeschlagen.\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
/******** Setzen von Treiberpegeln **********/
if (k==0 || k==2)
  {
  low_lev=449;
  strcpy(task, "DRLV 1,500,1000, (all_inputs);");
  }
else if (k==1 || k==3)
  {
  low_lev=6449;
  strcpy(task, "DRLV 1,6500,7000, (all_inputs);");
  }
task_len = strlen(task);
HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
answer[1023] = '\0';
if (ret_state != 0)   /* Wenn der Firmware-Befehl fehlschlägt... */
  {
  printf(„\n\nDRLV-Firmwarebefehl fehlgeschlagen!!!\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
cnt = 0;
/********** Setzen von Empfangspegeln*******/
for (i=low_lev; i<=low_lev+100; i=i+1)
  {
  if (low_lev==500 || low_lev==6500) low_lev++;; /* Vol = 500mV und 6500mV
     sind keine gültigen Testpunkte. */
    /* Setzen von Empfangspegel */
  strcpy(task, "RCLV 1,");
  sprintf(low_level_string,"%d",i);
  sprintf(high_level_string,"%d",i+100);
  strcat(task,low_level_string);
  strcat(task,",");
  strcat(task,high_level_string);
```

```
strcat(task,", (all_outputs);");
  task_len = strlen(task);
HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
answer[1023] = '\0';
if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt... */
  {
  printf(„\n\nRCLV-Firmwarebefehl fehlgeschlagen!!!\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }

/* Veränderung Vergleich Flankenzeitgebung */
strcpy(task,"ETIM 1,1,");
d_i=i;
if( (k==0) || (k==2) ) E=240/fabs(d_i-500)+.1;
if( (k==1) || (k==3) ) E=240/fabs(d_i-6500)+.1;
sprintf(set_compare,"%f",E);
strcat(task,set_compare);
strcat(task,",0,0,0,0,0,(all_outputs);");
task_len = strlen(task);
HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
answer[1023] = '\0';
if (ret_state != 0)
  {
  printf(„\n\nETIM-Firmwarebefehl fehlgeschlagen!!!\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
/* Ausführen einer Testfunktion */
ExecTfParam("functional;",tf_result,&tf_state);
if (tf_state == -1 )  /* Wenn Testfunktionsausführung fehlschlägt... */
  {
  printf(„FEHLER! Ausführung der DRLV-Testfunktion fehlgeschlagen.\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
GetPinResults(pin_result, &total_p, &tf_state);
```

```
    if (tf_state == 1)
      {
      for ( j=0; j<total_p; j++ )
        {
        if (k==0)
            {
          /* Speichern von Bestehen-Durchfallen-Ergebnissen */
            vol_500a[j][cnt] = pin_result[j].meas[0].result;
          /* Speichern entsprechender Pegel */
            rclv_vol_500a[cnt] = i;
            rclv_channel_a[j]=atoi(pin_result[j].pinname);
            }
        else if (k==1)
            {
            vol_6500a[j][cnt] = pin_result[j].meas[0].result;
            rclv_vol_6500a[cnt] = i;
          }
        else if (k==2)
            {
            vol_500b[j][cnt] = pin_result[j].meas[0].result;
            rclv_vol_500b[cnt] = i;
            rclv_channel_b[j]=atoi(pin_result[j].pinname);
          }
        else if (k==3)
            {
            vol_6500b[j][cnt] = pin_result[j].meas[0].result;
            rclv_vol_6500b[cnt] = i;
          }
        } /* schließt für Schleife */
      } /* schließt, wenn tf_state = 1 */
    else
      {
        printf(„GetPinResults für Vol-Messungen fehlgeschlagen. tf_state = %d\n“,
tf_state);
        printf("total_p = %d\n",total_p);
      if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
      HpTerm();
        exit(CI_CALL_ERROR);
        }
    cnt++;
    } /* schließt Zweites für Schleife */
  } /* schließt Erstes für Schleife */
```

```
/*************************************************/
/*                    Voh-Messungen                */
/*************************************************/
for (k=0; k<=3; k++)
  {
  /******** Laden von Datensätzen ***********/
  if( (k==0) || (k==1) )
    {
    DataManager(dm_set, dm_load, device_path, "rc_levels_a.mpc", 0, &tf_state);
    }
  else if ( (k==2) || (k==3) )
    {
    DataManager(dm_set, dm_load, device_path, "rc_levels_b.mpc", 0, &tf_state);
    }
  if (tf_state != 0) /* Wenn Datensatzladen fehlschlägt... */
    {
    printf("FEHLER! DataManager-Laden des Datensatzes fehlgeschlagen.\n\n");
    if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
    HpTerm();
    exit(CI_CALL_ERROR);
    }
  /********* Verändern des Vektoretiketts auf „eins"**********/
  strcpy(task, "SQSL \"one\";");
  task_len = strlen(task);
  HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
  answer[1023] = '\0';
  if (ret_state != 0) /* Wenn der Firmware-Befehl fehlschlägt... */
    {
    printf("\n\nSQSL-Frmwarebefehl fehlgeschlagen!!!\n\n");
    if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
    HpTerm();
    exit(CI_CALL_ERROR);
    }

  /******** Setzen von Treiberpegeln *********/
  if (k==0 || k==2)
    {
    high_lev=449;
    strcpy(task, "DRLV 1,0,500, (all_inputs);");
    }
```

```
    else if (k==1 || k==3)
      {
      high_lev=6449;
      strcpy(task, "DRLV 1,6000,6500, (all_inputs);");
      }
    task_len = strlen(task);
    HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
    answer[1023] = '\0';
    if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt... */
      {
      printf("\n\nDRLV-Firmwarebefehl fehlgeschlagen!!!\n\n");
      if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
      HpTerm();
      exit(CI_CALL_ERROR);
      }
    cnt = 0;
    /******** Setzen von Empfangspegeln *********/
    for (i=high_lev; i<=high_lev+100; i=i+1)
      {

      if (high_lev==500 || high_lev==6500) high_lev++;;  /* voh = 500mV und 6500mV
                sind keine gültigen Testpunkte. */
        /* Setzen von Empfangspegel */
      strcpy(task, "RCLV 1,");
      sprintf(low_level_string,"%d",i-100);
      sprintf(high_level_string,"%d",i);
      strcat(task,low_level_string);
      strcat(task,",");
      strcat(task,high_level_string);
      strcat(task,", (all_outputs);");
       task_len = strlen(task);
      HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
      answer[1023] = '\0';
      if (ret_state != 0)  /* Wenn der Firmware-Befehl fehlschlägt... */
        {
        printf("\n\nRCLV-Firmwarebefehl fehlgeschlagen!!!\n\n");
        if(error=disconnect()!=0) printf("WARNUNG!! Die Trennfunktion wurde nicht erfolg-
reich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n");
        HpTerm();
        exit(CI_CALL_ERROR);
        }
      /* Veränderung Vergleich Flankenzeitgebung */
```

```
strcpy(task,"ETIM 1,1,");
d_i=i;
if( (k==0) || (k==2) ) E=240/fabs(d_i-500)+.1;
if( (k==1) || (k==3) ) E=240/fabs(d_i-6500)+.1;
sprintf(set_compare,"%f",E);
strcat(task,set_compare);
strcat(task,",0,0,0,0,0,(all_outputs);");
task_len = strlen(task);
HpFwTask(task, &task_len, answer, &answer_len, &ret_state);
answer[1023] = '\0';
if (ret_state != 0)
  {
  printf(„\n\nETIM-Firmwarebefehl fehlgeschlagen!!!\n\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
/* Ausführen einer Testfunktion */
ExecTfParam("functional;",tf_result,&tf_state);
if (tf_state == -1 )  /* Wenn Testfunktionsausführung fehlschlägt... */
  {
  printf(„FEHLER! Ausführung der RCLV-Testfunktion fehlgeschlagen.\n“);
  if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
  HpTerm();
  exit(CI_CALL_ERROR);
  }
GetPinResults(pin_result, &total_p, &tf_state);
if (tf_state == 1)
  {
  for ( j=0; j<total_p; j++ )
    {
    if (k==0)
      {
      /* Speichern des Bestehen/Durchfallen-Ergebnis */
        voh_500a[j][cnt] = pin_result[j].meas[0].result;
      /* Speichern des entsprechenden Pegels */
        rclv_voh_500a[cnt] = i;
        }
    else if (k==1)
        {
        voh_6500a[j][cnt] = pin_result[j].meas[0].result;
```

```
                rclv_voh_6500a[cnt] = i;
              }
            else if (k==2)
              {
                voh_500b[j][cnt] = pin_result[j].meas[0].result;
                rclv_voh_500b[cnt] = i;
              }
            else if (k==3)
                {
                voh_6500b[j][cnt] = pin_result[j].meas[0].result;
                rclv_voh_6500b[cnt] = i;
              }
            }
          }
        else
          {
          printf(„GetPinResults für Voh-Messungen fehlgeschlagen. tf_state = %d\n“, tf_state);
            printf("total_p = %d\n",total_p);
          if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);
          HpTerm();
            exit(CI_CALL_ERROR);
            }
          cnt++;
        }
      }
    if(error=disconnect()!=0) printf(„WARNUNG!! Die Trennfunktion wurde nicht erfolgreich ausgeführt. Tester könnte noch in verbundenem Zustand sein. \n\n“);

    /*************************************************/
    /*                Ausgabeergebnisse als 2D Shmoo                */
    /*                        und/oder                              */
    /*   Lokalisieren von Bestehen/Durchfallen-Punkten              */
    /*************************************************/
    for ( k = 0; k < total_p; k++ )
      {
      /********** vol_500_a **********/
      if (shmoo_2d == 1) printf(„\n\n Vol-Messungen...\n“);
      flag = 0; /* Dies markiert ein erstes Bestehen/Durchfallen mit Flag */
      if (shmoo_2d == 1) printf("%d\t",rclv_channel_a[k]);
      for ( l = 1; l< cnt; l++ )
        {
        if (vol_500a[k][l] == 0)
```

```
        {
        if (shmoo_2d == 1) printf("*");
        if (flag==0) /* Wenn dies das erste „*“ nach einem „.“ ist, dann
                        Setzen eines ersten Bestehens. */
          {
          flag = 1;
          first_pass = rclv_vol_500a[l];
          vol_500a_error[k] = fabs(vil_500a[total_p-k-1] - rclv_vol_500a[l]);
          }
      }
    else
        {
        if (shmoo_2d == 1) printf(".");
        flag = 0;  /* Rücksetzen des Flags auf null in dem Fall, dass ein „.“ zwischen
                      einigen „*“ auftrat. */
        }
    }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", first_pass, vol_500a_error[k]);

/********** vol_500_b **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_b[k]);
for ( l = 1; l< cnt; l++ )
    {
    if (vol_500b[k][l] == 0)
        {
        if (shmoo_2d == 1) printf("*");
        if (flag==0) /* wenn dies das erste „*“ nach einem „.“ ist, Setzen eines ersten Bestehens. */
          {
          flag = 1;
          first_pass = rclv_vol_500b[l];
          vol_500b_error[k] = fabs(vil_500b[total_p-k-1] - rclv_vol_500b[l]);
          }
      }
    else
        {
        if (shmoo_2d == 1) printf(".");
        flag = 0;  /* Rücksetzen eines Flags auf null in dem Fall, dass ein „.“ zwischen
                      einigen „*“ auftrat. */
        }
    }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", first_pass, vol_500b_error[k]);
```

```
/********** vol_6500_a **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_a[k]);
for ( l = 1; l< cnt; l++ )
  {
  if (vol_6500a[k][l] == 0)
       {
       if (shmoo_2d == 1) printf("*");
       if (flag==0) /* wenn dies das erste „*“ nach einem „.“ ist, Setzen
                       eines ersten Bestehens. */
         {
         flag = 1;
         first_pass = rclv_vol_6500a[l];
         vol_6500a_error[k] = fabs(vil_6500a[total_p-k-1] - rclv_vol_6500a[l]);
         }
    }
  else
       {
       if (shmoo_2d == 1) printf(".");
       flag = 0; /* Rücksetzen des Flags auf null in dem Fall, dass ein „.“
                    zwischen einigen „*“ auftrat. */
       }
  }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", first_pass, vol_6500a_error[k]);

/********** vol_6500_b **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_b[k]);
for ( l = 1; l< cnt; l++ )
  {
  if (vol_6500b[k][l] == 0)
       {
       if (shmoo_2d == 1) printf("*");
       if (flag==0) /* wenn dies das erste „*“ nach einem „.“ ist, Setzen eines ersten Bestehens. */
         {
         flag = 1;
         first_pass = rclv_vol_6500b[l];
         vol_6500b_error[k] = fabs(vil_6500b[total_p-k-1] - rclv_vol_6500b[l]);
         }
    }
  else
       {
       if (shmoo_2d == 1) printf(".");
```

```
        flag = 0; /* Rücksetzen des Flags auf null in dem Fall, dass ein „.“ zwischen
                     einigen „*“ auftrat. */
        }
    }
 if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", first_pass, vol_6500b_error[k]);

/********** voh_500_a **********/
if (shmoo_2d == 1) printf(„\n\n Voh-Messungen...\n“);
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_a[k]);
for ( l = 1; l< cnt; l++ )
   {
   if (voh_500a[k][l] == 1)
        {
        if (shmoo_2d == 1) printf(".");
        if (flag==0) /* wenn dies das erste „.“ (erstes Durchfallen) ist, Setzen
                     eines letzten Bestehens. */
          {
          flag = 1;
          last_pass = rclv_voh_500a[l-1];
          voh_500a_error[k] = fabs(vih_500a[total_p-k-1] - rclv_voh_500a[l]);
          }
     }
   else
        {
        if (shmoo_2d == 1) printf("*");
        }
   }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", last_pass, voh_500a_error[k]);

/********** voh_500_b **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_b[k]);
for ( l = 1; l< cnt; l++ )
   {
   if (voh_500b[k][l] == 1)
        {
        if (shmoo_2d == 1) printf(".");
        if (flag==0) /* wenn dies das erste „.“ (erstes Durchfallen) ist, Setzen
                     eines letzten Bestehens. */
          {
          flag = 1;
          last_pass = rclv_voh_500b[l-1];
```

```
            voh_500b_error[k] = fabs(vih_500b[total_p-k-1] - rclv_voh_500b[l]);
            }
        }
      else
          {
          if (shmoo_2d == 1) printf("*");
          }
      }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", last_pass, voh_500b_error[k]);

/********** voh_6500_a **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_a[k]);
for ( l = 1; l< cnt; l++ )
  {
  if (voh_6500a[k][l] == 1)
          {
          if (shmoo_2d == 1) printf(".");
          if (flag==0) /* wenn das der erste „." (erstes Durchfallen) ist, Setzen
                      eines letzten Bestehens. */
            {
            flag = 1;
            last_pass = rclv_voh_6500a[l-1];
            voh_6500a_error[k] = fabs(vih_6500a[total_p-k-1] - rclv_voh_6500a[l]);
            }
      }
  else
          {
          if (shmoo_2d == 1) printf("*");
          }
  }
if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", last_pass, voh_6500a_error[k]);

/********** voh_6500_b **********/
flag = 0;
if (shmoo_2d == 1) printf("%d\t",rclv_channel_b[k]);
for ( l = 1; l< cnt; l++ )
  {
  if (voh_6500b[k][l] == 1)
          {
          if (shmoo_2d == 1) printf(".");
          if (flag==0) /* wenn das der erste „." (erstes Durchfallen) ist, Setzen
                      eines letzten Bestehens. */

            {
            flag = 1;
            last_pass = rclv_voh_6500b[l-1];
            voh_6500b_error[k] = fabs(vih_6500b[total_p-k-1] - rclv_voh_6500b[l]);
            }
      }
    else
          {
          if (shmoo_2d == 1) printf("*");
          }
    }
  if (shmoo_2d == 1) printf("\t %6.2f %6.2f\n", last_pass, voh_6500b_error[k]);
  }
}  /* Schließen der rc_lev-Funktion */
```

Patentansprüche

1. Ein System zum Testen einer Komponententoleranz einer Vorrichtung (**100**) zum Testen integrierter Schaltungen, wobei die Vorrichtung (**100**) eine Mehrzahl von Testverbindern aufweist, die an einem Testkopf (**205**) angeordnet sind, wobei jeder Testverbinder elektrische Leiter (**506**) aufweist, die elektrische Signale für einen Testkanal tragen, wobei das System folgende Merkmale aufweist:
eine Kurzschlusseinrichtung (**500**) zum Einrichten einer elektrischen Niederimpedanzverbindung zwischen elektrischen Leitern eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber (**402a**) eines ersten Kanals elektrisch über einen Niederimpedanzpfad mit einer Messeinheit (**406b**) eines zweiten Kanals verbunden ist;
eine Steuerung (**206**), die eine Einrichtung zum Steuern des ersten Treibers (**402a**), um ein elektrisches Signal eines vorbestimmten Zielwerts auszugeben, aufweist; und
eine Verifizierungseinrichtung (**206**, **712**), die eine Einrichtung zum Verifizieren, dass der durch die Messeinheit empfangene Spannungspegel innerhalb einer vorbestimmten Menge des vorbestimmten Zielwerts ist, aufweist;
dadurch gekennzeichnet, dass jeder Testkanal einer Schaltungsplatine (**400**) entspricht, die zumindest einen Treiber (**402**) und einen Empfänger (**404**) umfasst, derart, dass das Einrichten der elektrischen Niederimpedanzverbindung zwischen den elektrischen Leitern des ersten und des zweiten Testverbinders dazu führt, dass der erste Treiber von einer ersten Schaltungsplatine elektrisch über den Niederimpedanzpfad mit der Messeinheit auf einer zweiten Schaltungsplatine verbunden ist, und dadurch, dass das System ferner folgendes Merkmal aufweist:
eine Einrichtung zur Flag-Markierung, wenn der durch die Messeinheit empfangene Spannungspegel nicht innerhalb der vorbestimmten Menge ist, eines Fehlers, dass der erste Treiber nicht innerhalb einer spezifizierten Toleranz ist.

2. Ein System zum Testen einer Komponententoleranz einer Vorrichtung (**100**) zum Testen integrierter Schaltungen, wobei die Vorrichtung (**100**) eine Mehrzahl von Testverbindern aufweist, die an einem Testkopf (**205**) angeordnet sind, wobei jeder Testverbinder elektrische Leiter aufweist, die elektrische Signale für einen Testkanal tragen, wobei das System folgende Merkmale aufweist:
eine Kurzschlusseinrichtung (**500**) zum Einrichten einer elektrischen Niederimpedanzverbindung zwischen elektrischen Leitern eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber (**402a**) eines ersten Kanals elektrisch über einen Niederimpedanzpfad mit einem ersten Empfänger (**404b**) eines zweiten Kanals verbunden ist;
eine Steuerung (**206**), die eine Einrichtung zum Steuern des ersten Treibers (**402a**), um ein elektrisches Signal eines vorbestimmten Werts auszugeben, aufweist;
eine Auswertungseinrichtung (**206**, **808**, **810**) zum Auswerten des Signalspannungspegels, der an dem ersten Empfängern interpretiert wird; und
eine Verifizierungseinrichtung (**206**, **812**), die eine Einrichtung zum Verifizieren, dass das interpretierte Signal innerhalb einer spezifizierten Menge des vorbestimmten Werts ist, aufweist;
dadurch gekennzeichnet, dass jeder Testkanal einer Schaltungsplatine (**400**) entspricht, die zumindest einen Treiber (**402**) und einen Empfänger (**404**) umfasst, derart, dass das Einrichten der elektrischen Niederimpedanzverbindung zwischen den elektrischen Leitern des ersten und des zweiten Testverbinders dazu führt, dass der erste Treiber von einer ersten Schaltungsplatine elektrisch über den Niederimpedanzpfad mit dem ersten Empfänger auf einer zweiten Schaltungsplatine verbunden ist, und dadurch, dass das System ferner folgendes Merkmal aufweist:
eine Einrichtung zur Flag-Markierung, wenn das interpretierte Signal nicht innerhalb einer spezifizierten Menge des vorbestimmten Werts ist, eines Fehlers, dass der erste Empfänger nicht innerhalb einer spezifizierten Toleranz ist.

3. Die Vorrichtung gemäß Anspruch 2, bei der die Kurzschlusseinrichtung (**500**) eine gedruckte Schaltungsplatine umfasst, die Metallschichten beinhaltet, die direkte leitfähige Pfade zwischen Anschlussstiften auf dem ersten Testverbinder zu Anschlussstiften auf dem zweiten Testverbinder einrichten.

4. Die Vorrichtung gemäß Anspruch 2, bei der die Auswertungseinrichtung (**206**, **808**, **810**) und die Verifizierungseinrichtung (**206**, **812**) durch einen Programmcode realisiert sind, der den Betrieb einer Verarbeitungseinheit steuert.

5. Die Vorrichtung gemäß Anspruch 4, bei der die Verarbeitungseinheit ein Bauelement ist, das aus der Gruppe ausgewählt ist, die einen Mikroprozessor, eine Mikrosteuerung, eine eingebettete Steuerung, eine anwendungsspezifische integrierte Schaltung, und eine zweckgebundene Hardware, die speziell zur Ausführung

von Verarbeitungs- und Steuerfunktionen entworfen ist, umfasst.

6. Verfahren zum Diagnostizieren von Komponentenvarianzen, wobei das Verfahren in einer Testvorrichtung (**100**) zum Testen integrierter Schaltungen ausgeführt wird, die eine Mehrzahl von Testverbindern aufweist, die an einem Testkopf (**205**) angeordnet sind, wobei jeder Testverbinder elektrische Leiter (**506**) aufweist, die elektrische Signal für einen Testkanal tragen, wobei jeder Testkanal einer Schaltungsplatine (**400**) entspricht, die zumindest einen Treiber (**402**) und einen Empfänger (**404**) umfasst, wobei das Verfahren zum Diagnostizieren von Komponentenvarianzen die folgenden Schritte aufweist:
Einrichten einer elektrischen Verbindung zwischen elektrischen Leitern (**506**) eines ersten und eines zweiten Testverbinders, derart, dass ein erster Treiber (**402a**) von einer ersten Schaltungsplatine (**400a**) elektrisch über einen Niederimpedanzpfad mit einem ersten Empfänger (**404b**) einer zweiten Schaltungsplatine (**400b**) verbunden ist, wobei man von dem ersten Treiber (**402a**) weiß, dass er innerhalb einer spezifizierten Toleranz ist;
Steuern der Ausgabe des ersten Treibers (**402a**), um ein elektrisches Signal auf einen vorbestimmten Wert zu treiben;
Auswerten des an dem ersten Empfänger (**404b**) interpretierten Signalpegels;
Verifizieren, dass das interpretierte Signal innerhalb einer spezifizierten Menge des vorbestimmten Werts ist; und
Flag-Markieren, wenn das interpretierte Signal nicht innerhalb einer spezifizierten Menge des vorbestimmten Werts ist, eines Fehlers, dass der erste Empfänger nicht innerhalb einer spezifizierten Toleranz ist.

7. Das Verfahren gemäß Anspruch 6, das ferner folgende Schritte umfasst:
Einrichten einer elektrischen Verbindung zwischen elektrischen Leitern (**506**) eines ersten und eines zweiten Testverbinders, derart, dass ein zweiter Treiber von der ersten Schaltungsplatine (**400a**) elektrisch über einen Niederimpedanzpfad mit einem zweiten Empfänger auf einer zweiten Schaltungsplatine (**400b**) verbunden ist, wobei man von dem zweiten Treiber weiß, dass er innerhalb einer spezifizierten Toleranz ist;
Steuern der Ausgabe des zweiten Treibers, um ein elektrisches Signal auf einen vorbestimmten Wert zu treiben;
Auswerten des an dem zweiten Empfänger interpretierten Signalpegels; und
Verifizieren, dass der an dem zweiten Empfänger interpretierte Signalpegel innerhalb einer spezifizierten Menge des vorbestimmten Werts, der aus dem zweiten Treiber ausgegeben wird, liegt.

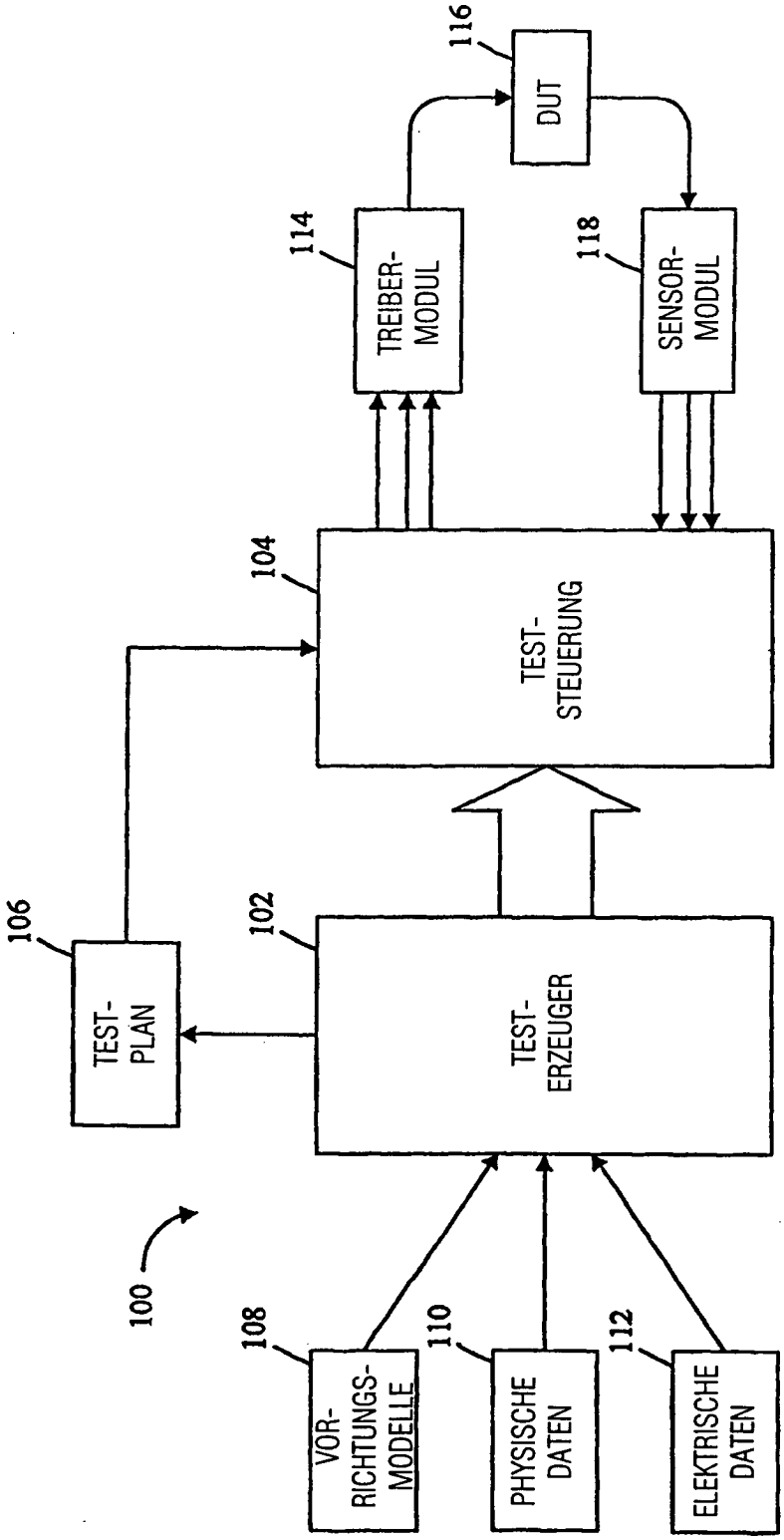
8. Das Verfahren gemäß Anspruch 6, bei dem der Schritt des Einrichtens einer elektrischen Verbindung ferner ein Einrichten einer elektrischen Verbindung zwischen elektrischen Leitern (**506**) eines ersten und eines zweiten Testverbinders, derart, dass eine Mehrzahl von Treibern von einer ersten Schaltungsplatine (**400a**) elektrisch über Niederimpedanzpfade mit einer Mehrzahl von Empfängern auf einer zweiten Schaltungsplatine (**400b**) verbunden ist und eine Mehrzahl von Treibern von einer zweiten Schaltungsplatine (**400b**) elektrisch über Niederimpedanzpfade mit einer Mehrzahl von Empfängern auf einer ersten Schaltungsplatine (**400a**) verbunden ist, wobei man von jedem der Mehrzahl von Treibern weiß, dass er innerhalb einer spezifizierten Toleranz liegt, aufweist.

9. Das Verfahren gemäß Anspruch 6, bei dem der Schritt des Steuerns der Ausgabe des ersten Treibers (**402a**) den Schritt eines Steuerns der Ausgabe des ersten Treibers (**402a**) auf einen ersten Pegel, der einen spezifizierten geringen Spielraum für ein Ausgangs-Hoch-Signal darstellt, umfasst, wobei der Verifizierungsschritt sicherstellt, dass der durch den ersten Empfänger (**404b**) empfangene Wert zumindest dieser Wert ist.

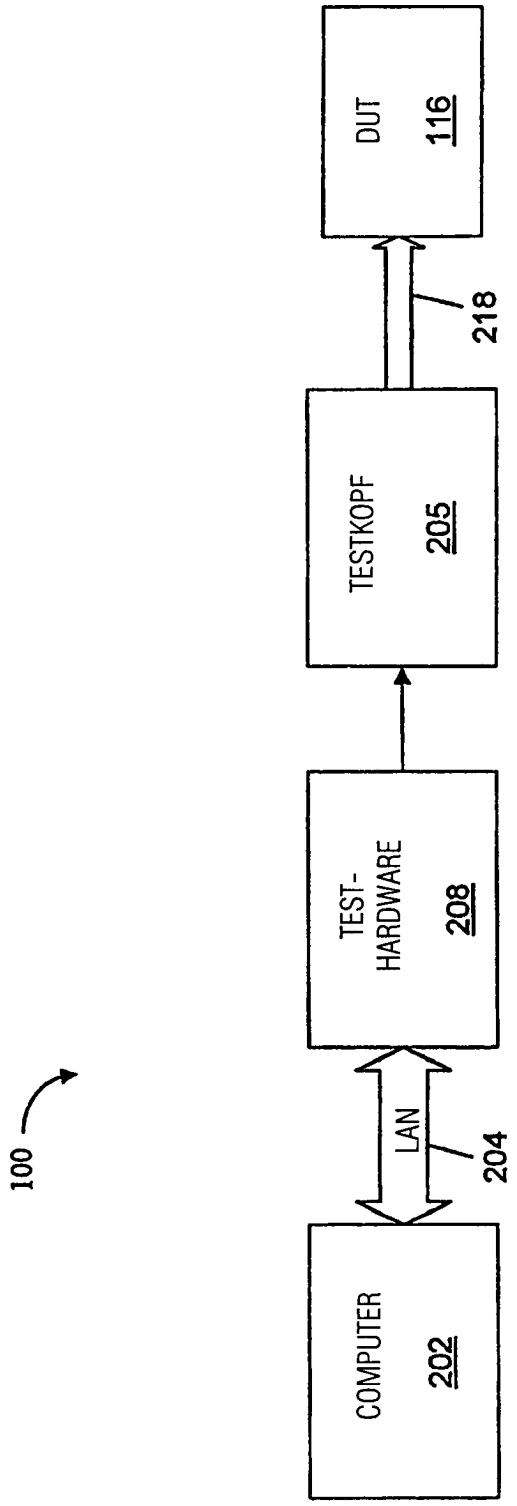
10. Das Verfahren gemäß Anspruch 6, bei dem der Schritt des Steuerns der Ausgabe des ersten Treibers (**402a**) den Schritt eines Steuerns der Ausgabe des ersten Treibers (**402a**) auf einen ersten Pegel, der einen spezifizierten großen Spielraum für ein Ausgangs-Hoch-Signal darstellt, umfasst, wobei der Verifizierungsschritt sicherstellt, dass der durch den ersten Empfänger (**404b**) empfangene Wert nicht größer als dieser Wert ist.

Es folgen 9 Blatt Zeichnungen

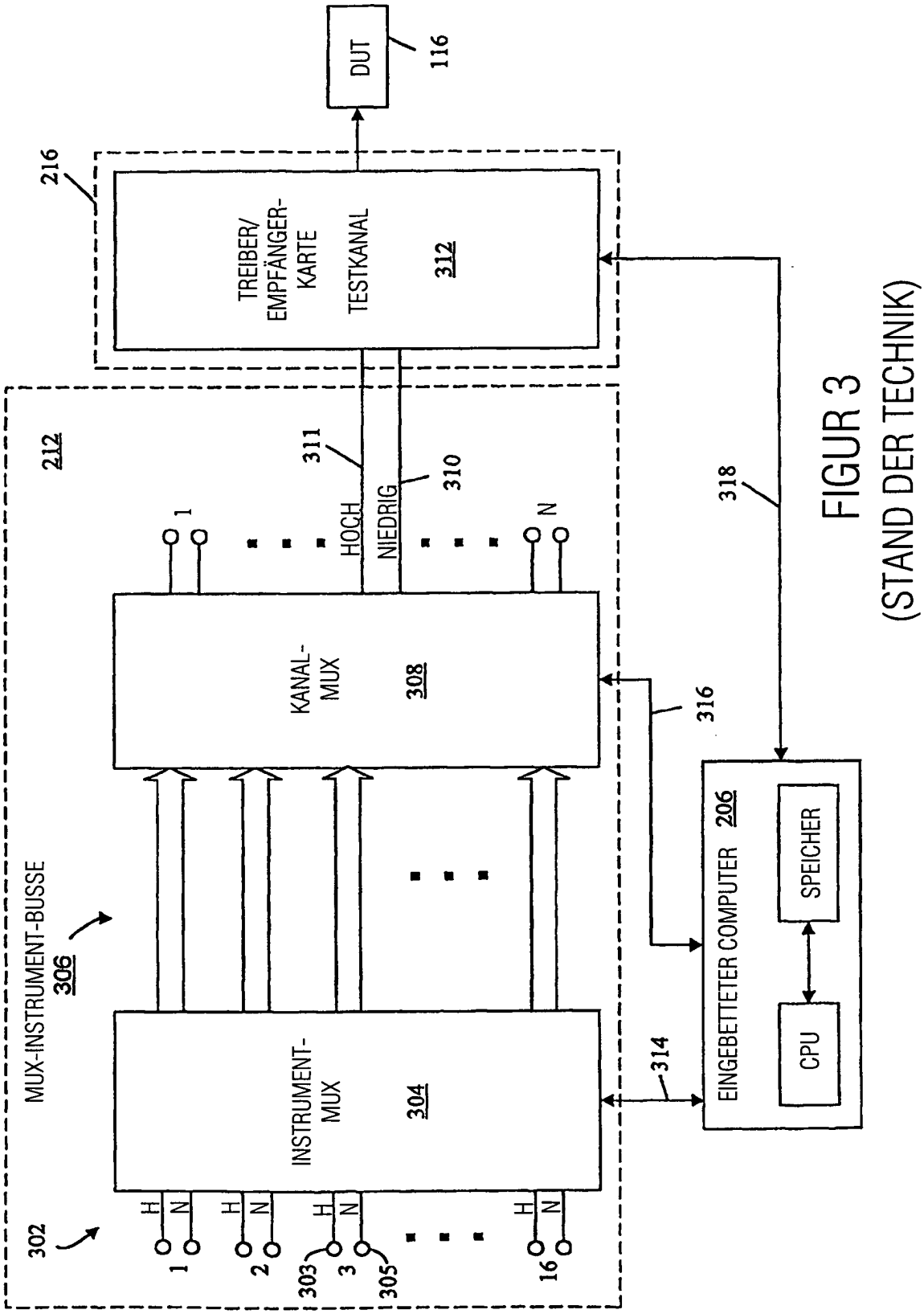
Anhängende Zeichnungen



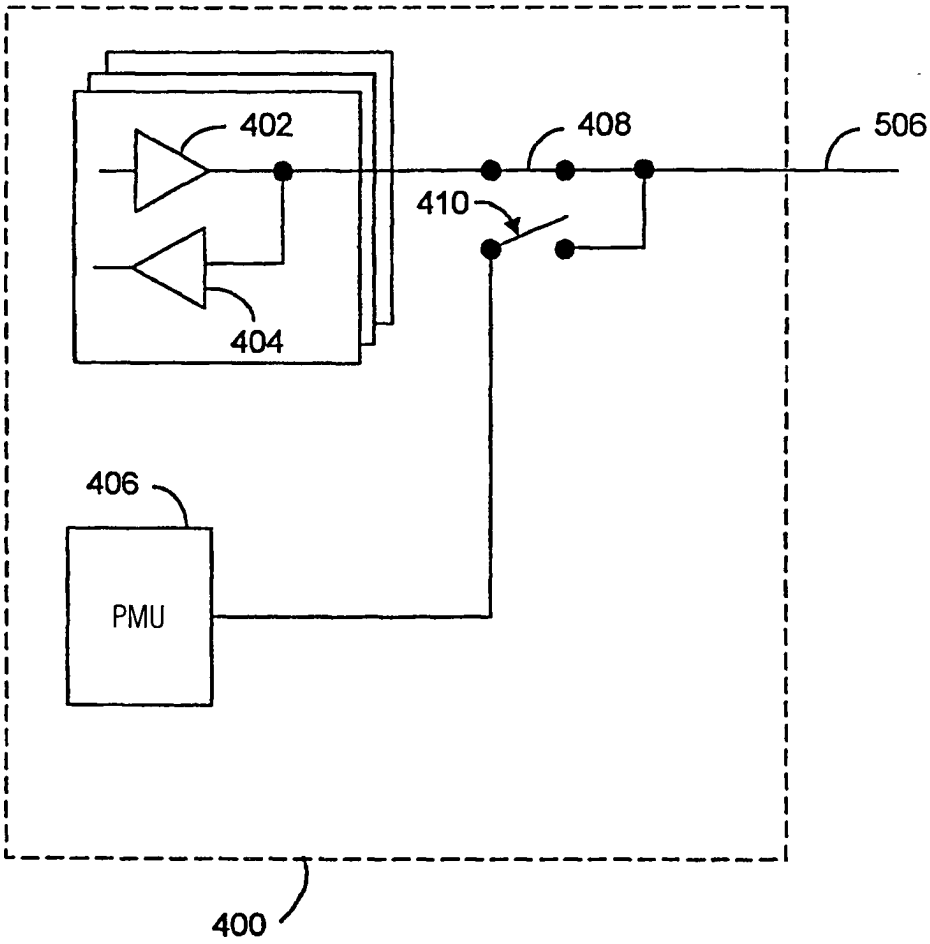
FIGUR 1
(STAND DER TECHNIK)



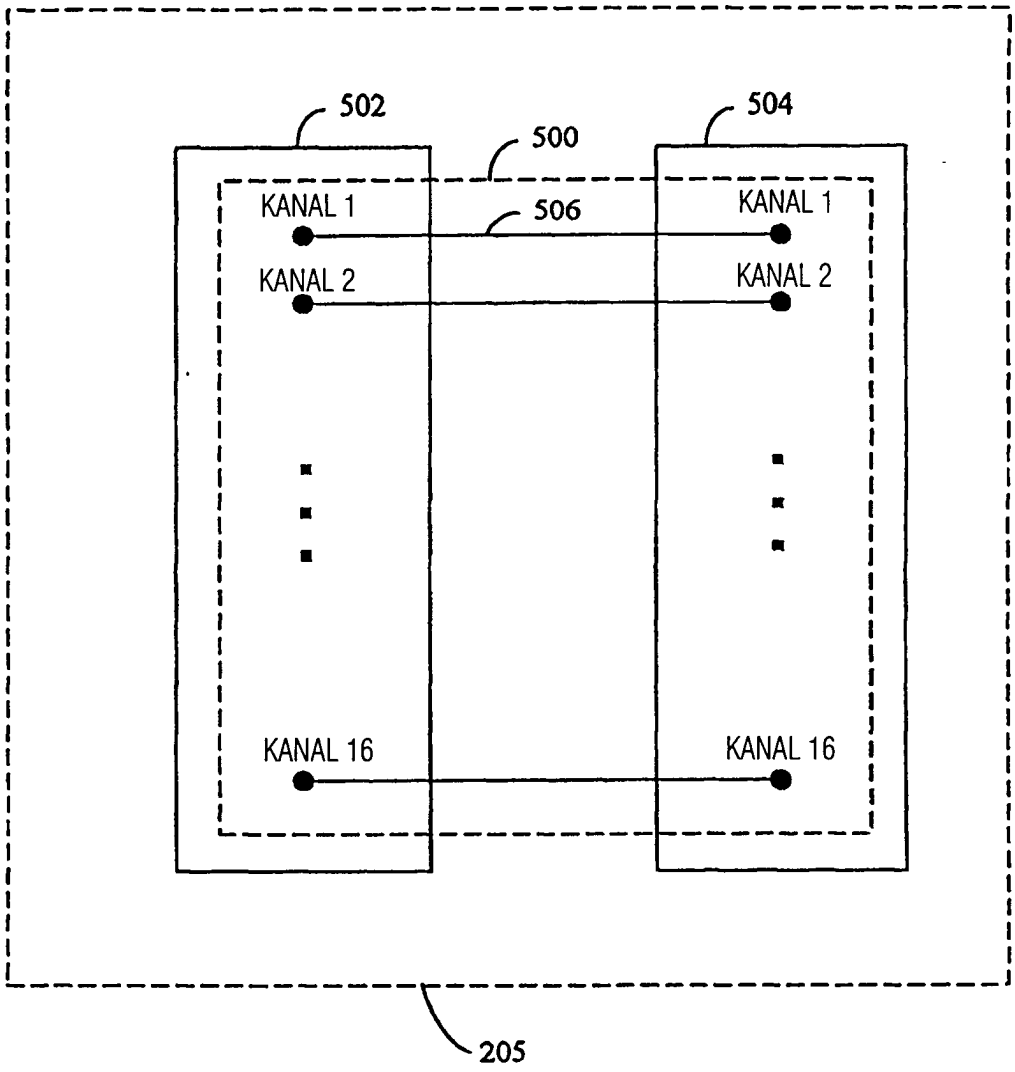
FIGUR 2
(STAND DER TECHNIK)



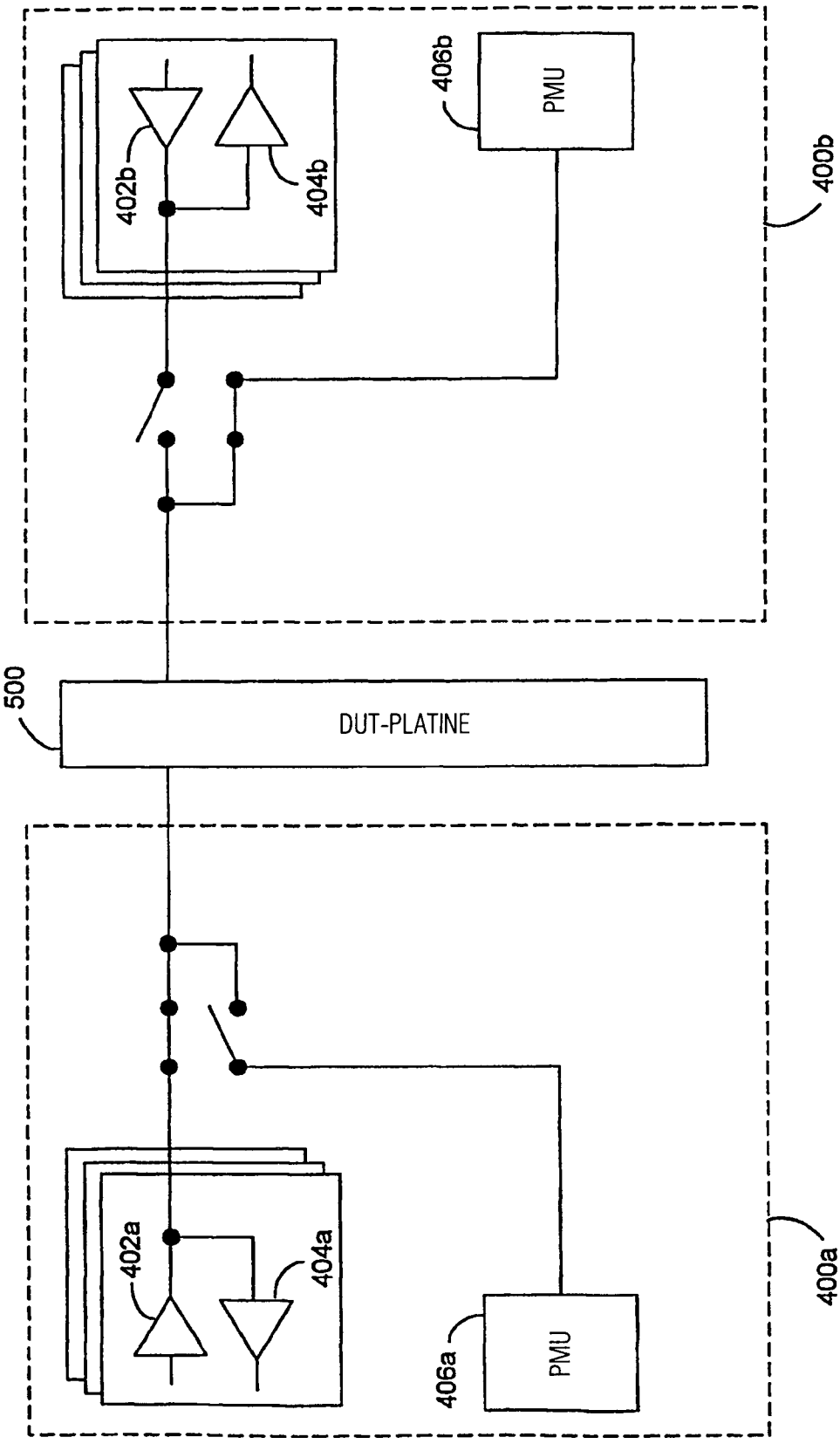
FIGUR 3
(STAND DER TECHNIK)



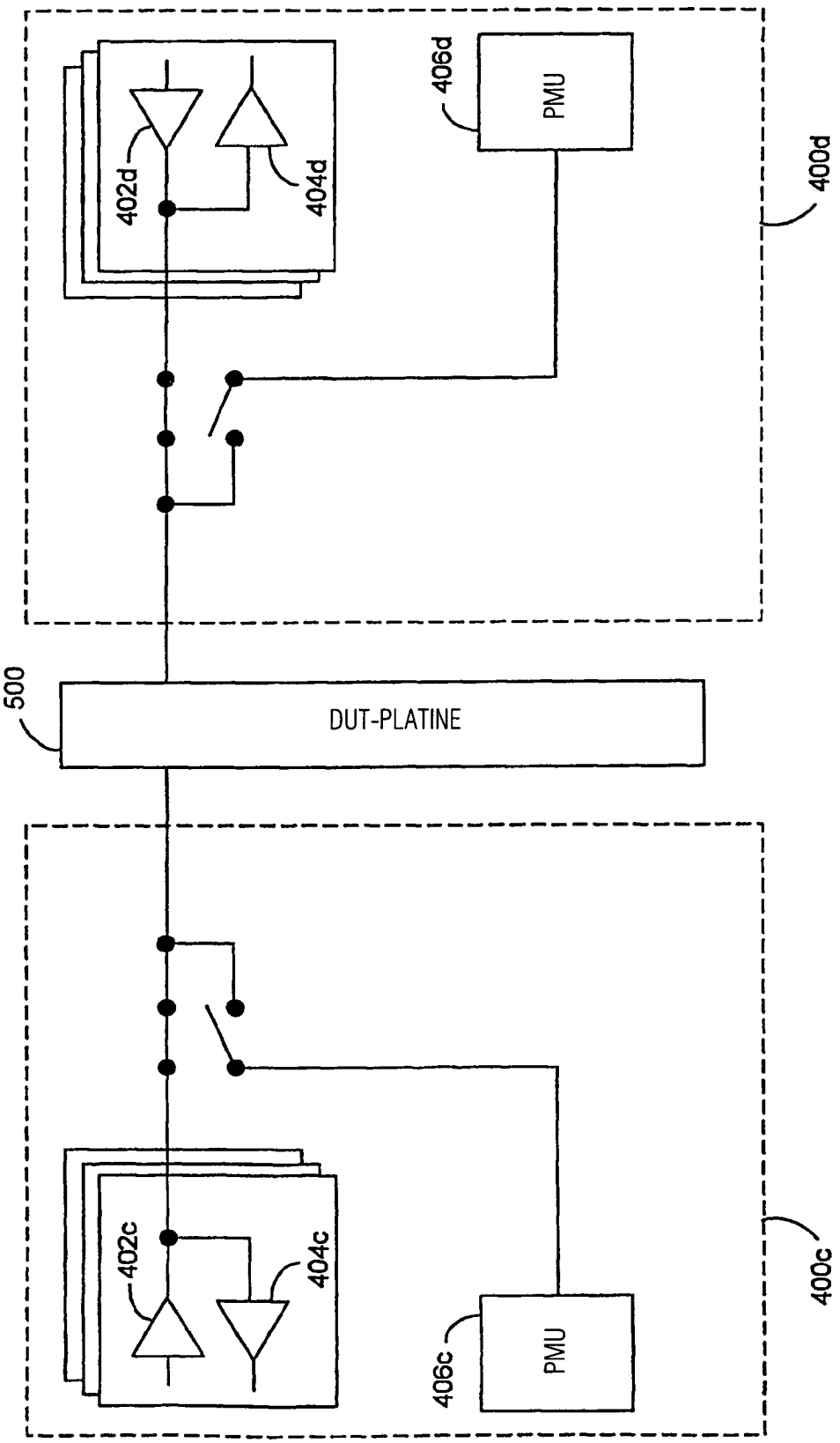
FIGUR 4


502
504
500
KANAL 1
506
KANAL 1
KANAL 2
KANAL 2
KANAL 16
KANAL 16
205

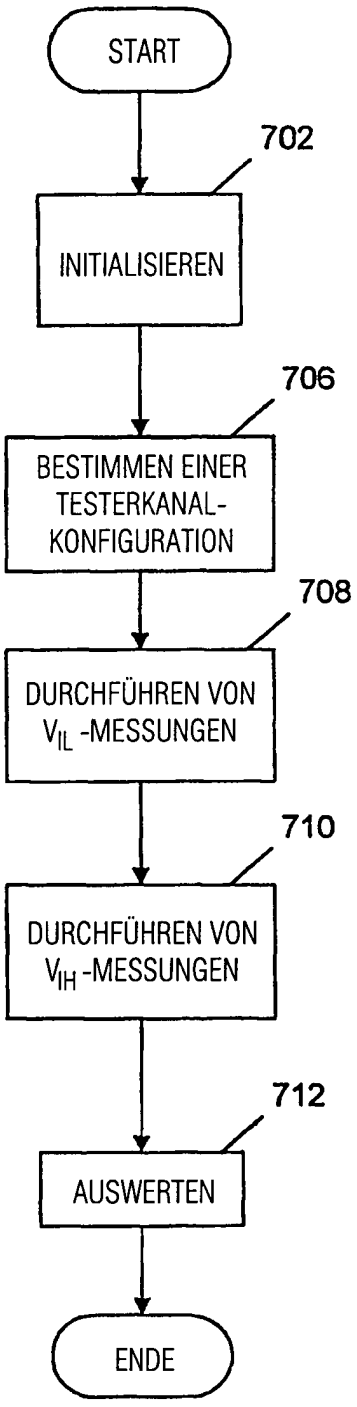
FIGUR 5


400b
406b
PMU
402b
404b
500
DUT-PLATINE
400a
406a
PMU
402a
404a

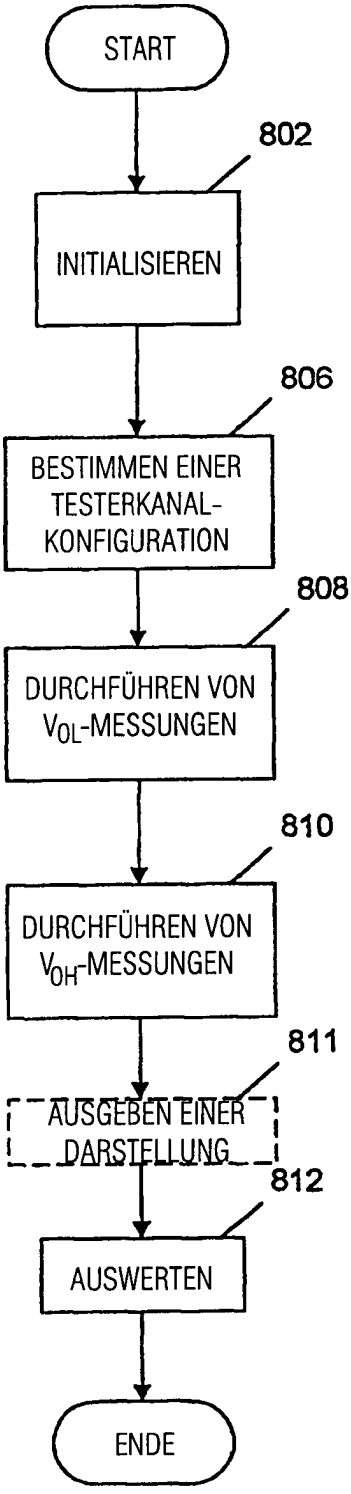
FIGUR 6A


400d
406d
PMU
402d
404d
500
DUT-PLATINE
400c
406c
PMU
402c
404c

FIGUR 6B


START
702
INITIALISIEREN
706
BESTIMMEN EINER TESTERKANAL-KONFIGURATION
708
DURCHFÜHREN VON V_{IL} -MESSUNGEN
710
DURCHFÜHREN VON V_{IH} -MESSUNGEN
712
AUSWERTEN
ENDE

FIGUR 7


START
802
INITIALISIEREN
806
BESTIMMEN EINER TESTERKANAL-KONFIGURATION
808
DURCHFÜHREN VON V_{OL}-MESSUNGEN
810
DURCHFÜHREN VON V_{OH}-MESSUNGEN
811
AUSGEBEN EINER DARSTELLUNG
812
AUSWERTEN
ENDE

FIGUR 8