



(19)
Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 699 10 941 T2** 2004.07.15

(12)

Übersetzung der europäischen Patentschrift

(97) **EP 0 995 999 B1**

(21) Deutsches Aktenzeichen: **699 10 941.8**

(96) Europäisches Aktenzeichen: **99 203 471.0**

(96) Europäischer Anmeldetag: **22.10.1999**

(97) Erstveröffentlichung durch das EPA: **26.04.2000**

(97) Veröffentlichungstag

der Patenterteilung beim EPA: **03.09.2003**

(47) Veröffentlichungstag im Patentblatt: **15.07.2004**

(51) Int Cl.⁷: **G01R 31/3167**
G01R 31/28

(30) Unionspriorität:

105267 P 22.10.1998 US

(73) Patentinhaber:

Texas Instruments Inc., Dallas, Tex., US

(74) Vertreter:

Prinz und Partner GbR, 81241 München

(84) Benannte Vertragsstaaten:

**AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT,
LI, LU, MC, NL, PT, SE**

(72) Erfinder:

Colby, David D., Anna, Texas 75409, US

(54) Bezeichnung: **Generator für beliebige Wellenformen**

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

Hintergrund und Zusammenfassung der Erfindung

[0001] Die vorliegende Erfindung bezieht sich auf das Gebiet des Entwurfs und des Testens elektronischer Bauelemente, und insbesondere auf das Gebiet der Erzeugung analoger Testsignale für das Testen von Bauelementen.

Hintergrund: Lineare und Mischsignal-Bauelemente

[0002] Mischsignal-Bauelemente benötigen synchronisierte analoge und digitale Signale, die zur zu testenden Vorrichtung (sog. „DUT“) für einen umfassenden Test eingespeist werden. Lineare DUTs benötigen auch analoge Signale, die für einen umfassenden Test als Eingangssignal eingespeist werden. Diese vermischten analogen und digitalen Signale nennt man Teststimuli. Beide Arten von Bauelementen benötigen extrem schwer zu erzeugende Eingangs-Signalförmn. Das Problem wird noch schwieriger, wenn man versucht, solche Vorrichtungen mit einem digitalen VLSI-Tester zu testen.

[0003] Üblicherweise werden die analogen und die Mischsignal-Anteile (digitale Muster, die in Verbindung mit analogen Signalen wirken) der Testprogramme von einem Testingenieur erzeugt. Der Testingenieur benutzt hauptsächlich manuelle Verfahren für die Erzeugung und die Messung von Sinuskurven, Strom, Gleichspannungen und digitalen Mustern, um die analogen Anteile und die Mischsignal-Anteile des DUT durchzuprüfen. Aus diesem Grund ist die Erzeugung von Analog- und Mischsignal-Testprogrammen noch immer ein arbeitsintensives, manuelles Verfahren.

Hintergrund: Generatoren zum Erzeugen beliebiger Signalförmn

[0004] Die herkömmliche Lösung zur Erzeugung der Signalförmn, die als analoge Anteile eines Testprogramms geeignet sind, besteht darin, eine digitale Testeinheit mit einem teuren Generator zum Erzeugen beliebiger Signalförmn (sog.

[0005] „AWG“) zu ergänzen, wie beispielsweise der HP E1441A von Hewlett-Packard. Üblicherweise kann ein AWG wie der HP E1441A Standard-Signalförmn erzeugen: Sinus, Quadrat, Dreieck, Rampe und Rauschen. Mit Hilfe des AWGs können auch benutzerdefinierte Signalförmn erzeugt werden. Jede erzeugte Signalförmn kann in AM, FM, FSK oder anderen Modulationsarten moduliert werden. Auch ein Frequenzdurchlaufwobbeln ist normalerweise verfügbar. Die Anzahl von in der Signalförmn definierten Datenpunkten (oder „die Auflösung der Signalförmn“) kann auch vom Anwender des AWGs bestimmt werden.

[0006] **Fig. 2** zeigt die Verbindung einer digitalen Testeinheit sowohl mit einem AWG als auch mit einem DUT. Der digitale VLSI-Tester initiiert und steuert die Erzeugung von Signalförmn, die zum Testen des DUT benötigt werden. Die Kommunikation des digitalen VLSI-Testers mit dem DUT hängt von der Bauart des DUT ab. In der Regel wird eine elektronische Zwischenverbindung, wie z. B. der Testhead **102**, als Schnittstelle zwischen den Ausgangsanschlüssen des digitalen VLSI-Testers und den Eingangsanschlüssen des DUT verwendet. Die Schnittstelle kann weitere Hardware enthalten, was das Ausgangssignal der digitalen Testeinheit verändert. Zur Verbindung des Testheads **102** mit dem DUT wird ein Vorrichtung-Schnittstellen-Board **104** benutzt. Der Steuercomputer der Testeinheit steuert sowohl den AWG als auch den digitalen VLSI-Tester während ein Testprogramm ausgeführt wird. Der AWG muss auch mit dem digitalen VLSI-Tester kommunizieren. Der digitale VLSI-Tester und der AWG sind verbunden, um das Ausgangssignal des AWG (Eingangssignal für das DUT) mit dem digitalen VLSI-Tester zu synchronisieren. Wie **Fig. 2** zeigt, liegt eine Problemquelle für den Testingenieur bei der Verwendung eines AWG in der Verkabelung. Bei der Vorbereitung des Tests eines Bauelementtyps, der AWG Ausgangssignale verwendet, muss der Testingenieur den Testhead, den digitalen VLSI-Tester und den Steuercomputer der Testeinheit verkabeln. Bei der Vorbereitung des Tests eines Bauelementtyps, der den AWG nicht benutzt, müssen die Kabel entfernt werden.

[0007] Abgesehen von der Schnittstellenproblematik, ist ein anderer Nachteil von AWGs der Kostenfaktor. Die Kosten für einen AWG richten sich in der Regel nach der Auflösung der Signalförmn (Abtastpunkte pro Sekunde), die er verarbeiten kann. Der Anstieg der Kosten eines AWGs ist oft überproportional zur Auflösung der erzeugten Signalförmn. Das Ergebnis wäre zum Beispiel eine Kombination aus digitalem Tester/AWG mit einem Preis von 1 Million Dollar im Jahre **1998**, die an ein DUT für zwei Dollar angeschlossen ist.

Generator für pseudobeliebige Signalförmn

[0008] Die vorliegende Anmeldung offenbart eine Methode, die es erlaubt, mit Hilfe eines kostengünstigen Schaltkreises von Widerständen und Kondensatoren die Ausgangssignale von digitalen Testertreibern in eine begrenzte Menge beliebiger Signalförmn umzuwandeln. Das Erzeugen beliebiger Signalförmn findet ohne

einen AWG statt.

[0009] Ein Softwareprogramm empfängt eine Menge von Datenpunkten, die eine für ein Testprogramm benötigte Signalform definiert. Die Menge von Datenpunkten kann in einem Testsimulator wie z. B. SPICE erzeugt werden, oder durch Digitalisierung der für das Testprogramm benötigten beliebigen Signalform. Basierend auf der Menge von Datenpunkten bestimmt die Software, wie viele Widerstände einer Generatorschaltung für pseudobeliebige Signalformen (PAWG) auf Hochpegel angesteuert werden müssen, um die Signalform zu reproduzieren, die als Eingangssignal für das DUT benötigt wird. Die Software bestimmt ferner die Auflösung der Signalform, z. B. 5 ns. Sobald diese Berechnungen vollständig sind, wird eine Menge von digitalen Vektoren erzeugt, die dem digitalen VLSI-Tester beschreibt, wie und wann die Widerstände anzusteuern sind. Die erzeugte Menge von Vektoren muss in einer Sprache vorliegen, die vom angepeilten VLSI-Tester akzeptiert wird.

[0010] Während die Menge von Vektoren verarbeitet wird, führen die Widerstände des PAWG-Schaltkreises zu einer Modulation der Ausgangsspannung. Ein an die Widerstände angeschlossener Kondensator glättet die Modulation und erzeugt dadurch eine Signalform, die im wesentlichen der von der Menge von Datenpunkten beschriebenen ähnelt. Das Ausgangssignal des Schaltkreises (der Teststimulus) wird dann an das DUT angelegt.

Kurze Beschreibung der Zeichnungen

[0011] Die offenbarten Erfindungen werden unter Bezugnahme auf die beigelegten Zeichnungen beschrieben, die exemplarisch wichtige Ausführungsbeispiele der Erfindung aufzeigen und daher als Referenz hierfür in der Beschreibung folgendermaßen mit einbezogen werden:

[0012] **Fig. 1** zeigt den PAWG-Schaltkreis.

[0013] **Fig. 2** zeigt die Verbindung eines AWG mit einem digitalen VLSI-Tester und einem DUT.

[0014] **Fig. 3A** und **3B** zeigen ein Ablaufdiagramm des PAWG.

Detaillierte Beschreibung der bevorzugten Ausführungsbeispiele

[0015] Die zahlreichen innovativen Lehren der vorliegenden Anmeldung werden mit besonderer Bezugnahme auf das derzeit bevorzugte Ausführungsbeispiel ausgeführt. Es wird jedoch darauf hingewiesen, dass diese Gruppe von Ausführungsbeispielen nur einige Beispiele für die vielen vorteilhaften Einsatzgebiete der vorliegenden innovativen Lehren darstellt. Im allgemeinen beschränken die Angaben, die in der Beschreibung der vorliegenden Anmeldung gemacht werden, nicht unbedingt eine der verschiedenen beanspruchten Erfindungen. Außerdem können einige Angaben sich nur auf manche erfinderische Merkmale beziehen und nicht auf andere Merkmale.

Software für den Generator für pseudobeliebige Signalformen

[0016] Im derzeit bevorzugten Ausführungsbeispiel wird die Erzeugung von pseudobeliebigen Signalformen durch die Verwendung eines Softwareprogramms, einer Menge von Vektoren, eines digitalen VLSI-Testers und eines Schaltkreises zur Umsetzung von diskreter Zeit in stetige Zeit ermöglicht. Es gibt heute viele verschiedene digitale Testeinheiten. V-Series oder Polaris Testeinheiten können z. B. erworben und mit dem beschriebenen Generator für pseudobeliebige Signalformen (PAWG) verwendet werden. In Hinsicht auf den PAWG unterscheiden sich die digitalen Tester lediglich in der Menge an Vektoren, die benötigt werden, um den Widerstandsteiler des PAWG-Schaltkreises richtig anzusteuern.

[0017] Ein digitaler VLSI-Tester besitzt in der Regel einen Mustergenerator, der festgelegte Daten verarbeitet und neue Daten in einer genau festgelegten Geschwindigkeit, z. B. alle 5 ns, ausgibt. Bei einer solchen Geschwindigkeit können logische Daten, die an einen beliebigen Eingang R_1 - R_n (die Widerstände des Widerstandsteilers des PAWG-Schaltkreises) zugeführt werden, eine vorprogrammierte hohe Spannung, z. B. 1 beim derzeit bevorzugten Ausführungsbeispiel, in eine vorprogrammierte niedrige Spannung, z. B. -1 beim derzeit bevorzugten Ausführungsbeispiel (oder umgekehrt) alle 5 ns ändern. Der Generator für pseudobeliebige Signalformen führt dazu, dass analoge Teststimuli vereinfacht erzeugt werden können – dies beinhaltet auch ungewöhnlich aussehende Signalformen – und zwar mit relativ wenigen Bauteilen und zu reduzierten Kosten.

[0018] Für die Erzeugung der Menge von Vektoren wird ein Softwareprogramm verwendet. Im Vorfeld muss der digitale Tester bekannt sein, der für das PAWG-Verfahren benutzt wird, damit die Menge von Vektoren exakt erzeugt werden kann. Der Softwarecode, der verwendet wird, um alle Informationen zu erzeugen, die für die Erzeugung der Menge von Vektoren benötigt werden, ist generisch. Nur die Routine der Menge von Vektoren muss für jeden digitalen VLSI-Tester abgeändert werden.

[0019] **Fig. 3A** und **3B** zeigen ein Ablaufdiagramm des PAWG-Softwareprogramms. Im derzeit bevorzugten

Ausführungsbeispiel liest die PAWG-Software eine Datei **302** mit Zeit/Spannungspaaren, die eine Signalforn in diskreten Zeitabschnitten festlegen (Schritt **304**). Ein Simulator, z. B. SPICE oder eine digitalisierte Signalforn, kann für die Erzeugung einer solchen Datei verwendet werden. Bei der tatsächlichen Testdurchführung kann eine analoge Signalforn erzeugt, digitalisiert und dann im Testprogramm verwendet werden. Während der Datenpunkt (Zeit/Spannungspaar) gelesen wird, wird dieser in einem Datenfeldpaar (Zeit und Spannung) gespeichert (Schritt **306**).

[0020] Sobald mindestens zwei Datenpunkte gelesen wurden, kann die Steigung der Kurve, die von den Datenpunkten beschrieben wird, bestimmt werden (Schritt **308**). Die Datenpunkte werden verarbeitet, bis zwei Endpunkte der RC-Kurve gefunden werden (die Steigung der Kurve verändert sich). Der Anfang der Kurve wird im Softwarecode als FNR bezeichnet. Wenn sich die Steigung der Kurve verändert hat oder wenn die Spannung abgenommen hat und jetzt zunimmt oder umgekehrt, wird das als Endpunkt der aktuellen Kurve gekennzeichnet. Die Kurve (oder RC-Kurve) ist die kapazitive Ladungs- oder Abnahmekurve und stellt einen Bogen dar, wenn Spannung gegenüber Zeit grafisch dargestellt wird. Die Form (oder der Winkel) der RC-Kurve hängt von vielen Faktoren ab, inklusive der RC-Konstanten der Software:

RCa ist die Widerstands-/Kapazitätszeitkonstante für R_1 - R_n und C_1 in **Fig. 1**.

RCb ist die Widerstands-/Kapazitätszeitkonstante für R_s und C_s in **Fig. 1**.

res ist die Auflösung der Signalforn, d. h. die Mindestzeitspanne zwischen Änderungen in der Signalforn. Im derzeit bevorzugten Ausführungsbeispiel beträgt die Auflösung der Signalforn 5 ns.

RPV ist die Auflösung pro Vektor. In einer Zeitspanne von 10 ns beträgt die RPV 2, d. h. es gibt 2 Änderungen pro Zeitspanne.

[0021] Ziel dieses Abschnitts der PAWG-Software ist es, die Datenpunkte des Bogens ausfindig zu machen, die die Endpunkte der Kurve darstellen, die als analoge Teststimuli erzeugt wird. Sobald Endpunkte der Kurve gefunden sind, wird eine Kurven-Anpassungs-Routine aufgerufen (Schritt 310).

Kurven-Anpassungs-Routine

[0022] Wenn die Kurven-Anpassungs-Routine auf Grund eines Richtungswechsels aufgerufen wird, erhält die Routine eine 0. Wenn die Kurven-Anpassungs-Routine auf Grund einer Änderung der Steigung aufgerufen wird, erhält die Routine eine 1.

[0023] Die Routine berechnet zuerst die Werte der Veränderung der Spannung (ΔV) und der Veränderung der Zeit (ΔT). Die Werte basieren auf den Datenpunkten, die als die zwei Endpunkte der RC-Kurve gefunden wurden (Schritt **312**). ΔV ist die Differenz zwischen der angelegten Spannung am Beginn der Kurve und der gewünschten Spannung am Ende der Kurve. ΔT ist die Zeitdifferenz von Anfang der Kurve bis zum Ende der Kurve, gerundet auf den nächsten Zeitabschnitt der Auflösung (5 ns beim derzeit bevorzugten Ausführungsbeispiel). Das heißt $\Delta T = \text{res} \cdot \text{Zeit}$.

[0024] Die Kurven-Anpassungs-Funktion berechnet eine ideale Spannung V_{app} , die die gewünschte Spannung am Endpunkt erreicht (Schritt **314**). V_{Aend} ist die Spannung, die am Endpunkt der Kurve angelegt werden sollte. V_{app} ist demnach eine Funktion der Spannung am Widerstandsteiler (R_1 - R_n) verglichen mit RCa. Das heißt, die Funktion berechnet die Ausgangsspannung, die an R_1 - R_n benötigt wird, um ΔV innerhalb ΔT mit einer Kapazität von C_1 zu erreichen. Die im derzeit bevorzugten Ausführungsbeispiel verwendete Formel lautet:

$$V_{app} = \Delta V / (1 - \exp(-1 \cdot \Delta T / R_{Ca})) + V_{Anow}$$

wobei: V_{app} die Spannung ist, die an R_1 - R_n des Widerstandsteilers anzulegen ist, und V_{Anow} die Spannung ist, die sich an C_1 ergeben würde.

[0025] V_{Anow} wird beobachtet, während sich die Zeit- und Spannungswerte an den Datenpunkten der Kurve verändern. Das Ergebnis der Berechnung bestimmt, wie viele der Widerstände R_1 - R_n auf Hochpegel (+1v) und wie viele auf Niederpegel (-1v) angesteuert werden müssen (Schritt 316).

[0026] Die Spannung, die man erhalten würde, indem man den Widerstandsteiler auf diese Art ansteuert, wird als V_{Anew} berechnet, und die Spannung, die am Ende der Kurve erwartet wird, wird auf der Basis der vorhergehenden Kalkulationen vorhergesagt (die Anzahl der Widerstände, die auf Hoch- oder Niederpegel angesteuert werden, verglichen mit der Spannung am Anfang der Kurve) (Schritte **318** und **320**). Wenn V_{Anew} ungleich V_{Aend} ist, muss die Anzahl der auf Hochpegel angesteuerten Widerstände angepasst werden (Schritt **322**). Wenn V_{Anew} kleiner als die ideale Spannung V_{app} ist, werden die Musterdaten (Daten, die angeben, wie viele Widerstände auf Hochpegel bzw. Niederpegel anzusteuern sind) so angepasst, dass mehr Widerstände auf Hochpegel angesteuert werden (Schritt **324**). Wenn die Spannung, die man erhalten würde, größer als die ideale Spannung V_{Aend} ist, werden die Musterdaten so angepasst, dass mehr Widerstände auf Niederpegel angesteuert werden (Schritt **324**). Diese Anpassung wird lediglich bei einigen Datenpunkten am Anfang der Kurve vorgenommen (Schritt **324**). Dieses kleine Anpassungsfenster ergibt sich aus einem Beschleunigungsfaktor an C_1 . Das heißt, dass sich größere Änderungen am Widerstandsteiler anfangs nur gering an C_1

auswirken. Wenn man jedoch außerordentlich hohe oder niedrige Spannungen am Anfang der Kurve anlegt, kann die Ausgangssignalform am Ende der Kurve wesentlich übertrieben erscheinen. Daher wird im derzeit bevorzugten Ausführungsbeispiel die Auflösung der Kurve lediglich während der ersten 20 ns angepasst. Diese Zeit reicht aus, um die Kurve hervorzurufen, ohne die Signalform zu verzerren.

[0027] Die Anpassung von Vapp ist in der Software ein iterativer Prozess. Der iterative Prozess ist dann abgeschlossen, wenn die Anpassungen den Wert von VAnew über VAend hervorbringen (für den Fall, in dem die erhaltene Spannung niedriger als die ideale Spannung Vapp ist) oder unter VAend bringen (für den Fall, in dem die erhaltene Spannung höher als die ideale Spannung Vapp ist) (Schritt **322**).

[0028] Sobald eine angepasste und eine unangepasste Kurve vorliegen, werden diese verglichen, um zu sehen, welche Kurve eine bessere Passgenauigkeit aufweist, und zwar basierend auf den einlaufenden Datenpunkten, der ursprünglichen Vorhersage oder der Vorhersage, die durch den iterativen Prozess wie oben beschrieben angepasst wurde (Schritt **326**). Die Informationen (eine Menge von Mustern), die die bessere Passgenauigkeit beschreiben (Anzahl der auf Hochpegel anzusteuernenden Widerstände und Ansteuerungsdauer), werden dann in einem Datenfeld gespeichert.

Mustererzeugungs-Routine

[0029] Nachdem alle Datenpunkte, die die Signalform einer bestimmten Kurve beschreiben, gelesen wurden und die Kurven-Anpassungs-Funktion für jede Veränderung der Richtung oder Steigung abgeschlossen ist, wird eine Datei erstellt (die Datei „Expected“ (= erwartet) **336**), die die von der Kurven-Anpassungs-Funktion errechneten Zeit- und Spannungsdaten enthält. Die geschriebenen Daten entsprechen der Auflösung des Softwaretreibers. Die erwartete Datei dient als Genauigkeitsüberprüfung, indem man eine grafische Darstellung der eingehenden Zeit/Spannungspaare über eine grafische Darstellung der erwarteten Zeit/Spannungspaare legt.

[0030] Das von der Kurven-Anpassungs-Funktion bestimmte Ergebnis des Musterdatenfeldes wird in den Spannungswert umgerechnet, der an den Widerstandsteiler bei jedem Schritt der zeitlichen Auflösung der vorliegenden Kurve (Schritt **328**) angelegt werden muss. Der Zeitwert und der Spannungswert werden in die Datei „Expected“ geschrieben (Schritt **330**). Der Spannungswert zeigt die neue Spannung an, die bei jedem Schritt der zeitlichen Auflösung angelegt werden muss.

[0031] Die Vektordaten, die mit dem Eingangssignal übereinstimmen, das für den jeweiligen einzusetzenden VLSI-Tester benötigt wird, müssen nun erstellt werden. Die Vektordaten werden auf Basis der Muster bestimmt, die sich aus der Kurven-Anpassungs-Routine ergeben haben. Im derzeit bevorzugten Ausführungsbeispiel treibt die Software einen digitalen Multi-Level Tone 3 (mlt3) Signalform VLSI-Tester. mlt3 ist eine Signalform, die für Ethernet-Schnittstellen am Computer verwendet werden. Bei mlt3-Tests ist die verwendete Signalform so verzerrt, als ob sie 100 Meter durch 10/100T Kabel zurückgelegt hätte. Das DUT wird untersucht, um sicherzustellen, dass es das Signal richtig lesen kann.

[0032] Dieser digitale Tester verwendet einen V-Series-MZDI-Schaltkreis, der auf Vektorbasis programmiert werden muss. Die Vektordauer für den V-Series-MZDI beträgt 10 ns. Folglich passen 2 Schritte (von jeweils 5 ns Dauer) in jeden Vektor. Die Programmierdaten für die Vektordaten basieren auf den aktuellen Daten (Anzahl an auf Hochpegel anzusteuernenden Widerständen) und auf den vorhergehenden Daten (Schritt **332**). Auf dieser Basis wird eine Mengenzahl zugeteilt, die in einer zweiten Datei gespeichert wird (die Musterdatei **338**) (Schritt **334**). Der MZDI-Schaltkreis benötigt für das Programmieren auch eine weitere Datei, die konstant ist (sie ändert sich trotz der Mengen an Eingangsvektoren nicht). Die Eingangsmengen sind speziell für den verwendeten Testgenerator bestimmt. Es wurde auch Softwarecode zur Ansteuerung eines digitalen Polaris-VLSI-Testers erzeugt.

Erzeugung von Signalformen

[0033] **Abb. 1** zeigt den PAWG-Schaltkreis. Der analoge Teststimulus wird vom PAWG-Schaltkreis erstellt, der von einem digitalen VLSI-Tester das Eingangssignal erhält. Die von der PAWG-Software erstellte Musterdatei wird als Eingangssignal für den digitalen VLSI-Tester verwendet. Die in der Musterdatei beschriebenen Vektoren werden vom digitalen VLSI-Tester gelesen. Der digitale Tester wiederum leitet die Eingangssignale des Widerstandsteilers (R_1 - R_n) an den Schaltkreis. Im derzeit bevorzugten Ausführungsbeispiel besteht der Widerstandsteiler aus 8 Widerständen. Eine Änderung der Anzahl an Widerständen im Widerstandsteiler kann die Spannungsauflösung der daraus resultierenden Kurve erhöhen oder vermindern. Die Software wurde für die Herstellung von Vektoren entwickelt, die für einen Widerstandsteiler mit 8 Widerständen bestimmt sind. Deshalb muss die Software abgeändert werden, wenn eine unterschiedliche Anzahl von Widerständen im Widerstandsteiler verwendet werden soll.

[0034] Kondensator C_1 ist mit dem Ausgang des Widerstandsteilers verbunden und geerdet. Die Geschwindigkeit des Ansteigens oder des Abfallens der Spannung wird von C_1 (oder der RCa Konstante) bestimmt. Kon-

densator C_1 verhindert einen schnellen Spannungswechsel im Transistor T_1 . Transistor T_1 puffert den RC-Schaltkreis des DUT, um zu verhindern, dass die Eingangsimpedanz des DUT seine Zeitkonstante beeinflusst. R_S und C_S werden verwendet, wenn eine zweite, niedrigere Frequenzsteigung erwünscht ist. Eine zweite, niedrigere Frequenzsteigung wird üblicherweise bei der Erzeugung von mlt3 Signalformen verwendet. Bei mlt3 Signalformen verursacht ein Übertragungsleitungseffekt, der so genannte „Grundlinie-Wanderungs-Effekt“, die Modulation eines Niederfrequenz-Dreieckssignals mit dem Hochfrequenzsignal der Datenübertragung. Als Folge dieser Modulation steigt die mittlere Referenzlinie, oder Grundlinie, des schon verzerrten Hochfrequenzsignals, das vom DUT empfangen wird, langsam an und fällt dann langsam ab. Um sicherzustellen, dass das DUT den „Grundlinie-Wanderungs-Effekt“ kompensieren kann, werden R_S und C_S verwendet, um die Grundlinie des Hochfrequenzsignals nach oben ansteigen zu lassen und somit einen halben Takt der „Grundlinie-Wanderung“ zu simulieren. In einem alternativen Ausführungsbeispiel, wenn z. B. ein voller Takt der „Grundlinie-Wanderung“ simuliert werden soll, würde C_S anstelle der Erdung an einem Testeinheitstreiber terminiert werden, und der Treiber würde jedes Mal, wenn eine Änderung der Steigung gewünscht ist, von einer hohen auf eine niedrige Spannung und umgekehrt geschaltet werden. Kondensator C_1 dient dazu, die diskreten Zeiteingangssignale (Datenpunkte der Ursprungsdatei und Eingangssignal der Widerstände im Widerstandsteiler) zu integrieren, wodurch die simulierte Signalform von diskreter Zeit auf stetige Zeit umgesetzt wird. Das daraus resultierende Ausgangssignal des Schaltkreises ist ein analoges Ausgangssignal, das um die Auflösung des DUT korrigiert wurde.

Programmcode

[0035] Der folgende Programmcode stellt eine exemplarische Implementierung der offenbarten Technik dar. Natürlich sind alternative Implementierungen für einen Fachmann sehr einfach abzuleiten, und dieser Programmcode dient daher nur als Beispiel dafür, wie diese Funktionen zu Stande gebracht werden können, und soll in keiner Weise den Umfang oder die Ausführungen der offenbarten Technik eingrenzen.

V-Series Mustergenerator für mlt3 Signalformen

- liest Zeit/Spannungspaardaten, die die gewünschte mlt3 Signalform darstellen

und erzeugt eine Menge von Mustern für die V-Series Testeinheit, um den

PAWG zu steuern.

- erzeugt auch Zeit/Spannungspaardaten der erwarteten PAWG Signalform

#

BEGIN { # dieser Block wird einmalig am Beginn des Programms ausgeführt

setze Flags, Zähler und Ausgangsspannung fest

CheckSlope = 0; ProcessLine = 0; x = 1; p = 0;

DrvHi = 1; DrvLo = -1;

Vnow = 0; VAnow = 0; MinRatio = 1; Vnew = 0;

setze RC Wert fest, der aus den gemessenen Steigungen berechnet wurde

RCa = 34,09e-9; RCb = 4,88488e-6;

setze Auflösung (res) auf die Hälfte des Testzeitraums fest (Auflösung auf

V-Series)

res = 5e-9;

```
#    setze Auflösungsschritte pro Vektor

    RPV = 2;

}

#

# Alle Zeilen von hier bis „END“ werden ausgeführt, sobald eine neue Zeile
# der Eingabedatei gelesen wird, wenn nicht anderweitig gekennzeichnet
#

{ # lese einen Datenpunkt

    t[x] = $1; # speichere Zeitmesswert in Datenfeld t

    v[x] = $2; # speichere Spannungsmesswert in Datenfeld v

    if (FNR == 1) {

        FirstT = t[x];

        FirstV = v[x];

        SaveT= t[x]}

#

# bestimme ob Spannung zunimmt oder abnimmt.

#

if (x > 1) {

    if (v[x] > v[x-1]) {rising = 1} else {rising = 0}}

#
```

```
# wenn die Spannung nun zunimmt, davor aber abgenommen hatte,
# dann befindet sich der vorher gelesene Punkt am Ende einer RC Kurve.
```

```
#
```

```
if ((ProcessLine == 0) && (x > 2)) {
    if ((rising == 1) && (v[x-1] < v[x-2])) {
        EndPoint = x - 1;
        SaveT = t[x];
        SaveV = v[x];
        ProcessLine = 1;
        CurveFit(0)} # die Funktion CurveFit befindet sich am Ende dieser Datei
```

```
#
```

```
# wenn die Spannung nun abnimmt, davor aber zugenommen hatte,
# dann befindet sich der vorher gelesene Punkt am Ende einer RC Kurve.
```

```
#
```

```
if ((ProcessLine == 0) && (x > 2)) {
    if ((rising == 0) && (v[x-1] > v[x-2])) {
        EndPoint = x - 1;
        SaveT = t[x];
        SaveV = v[x];
        ProcessLine = 1;
```

CurveFit(0)}} # die Funktion CurveFit befindet sich am Ende dieser Datei

#

wenn |Steigung| am aktuellen Punkt größer ist als |Steigung| am vorherigen

Punkt, dann befindet sich der vorherige Punkt am Ende einer RC Kurve.

die Formeln hängen von den stetigen Zeitschritten in der Eingabedatei ab.

nicht während der ersten 5 ns jeder Kurve prüfen, da die nawk exp

Funktionsauflösung einen Division-durch-Null-Fehler später in der Routine

hervorruft.

#

if ((ProcessLine == 0) && ((t[x] - SaveT) > 5e-9)) {

 d1 = v[x-1] - v[x-2];

 d2 = v[x] - v[x-1];

 if (d1 != 0) {Ratio = d2/d1} else {Ratio = 0}

 if (CheckSlope) {

 if (Ratio > OldRatio) {UpSlope = 1} # am Eckpunkt wenn Ratio >

OldRatio

 else {

 if ((UpSlope) && (Ratio > MinRatio)) { # ein Schritt nach Eckpunkt

 EndPoint = x - 1;

 SaveT = t[x];

SaveV = v[x];

ProcessLine = 1;

CurveFit(1)}} # die Funktion CurveFit befindet sich am Ende dieser

Datei

UpSlope = 0}}

Old Ratio = Ratio;

CheckSlope = 1}

#

wenn nicht am Ende der Kurve, Datenfeldindex erhöhen und nächsten

Datenpunkt lesen.

#

if (ProcessLine == 0) {x++}

#

wenn am Endpunkt der Kurve, dann verarbeite die Kurve.

#

if (ProcessLine == 1) {

ProcessLine = 0}}

#

#

„END“ Block wird einmal ausgeführt, wenn der letzte Datenpunkt gelesen

wurde

#

END { # Ende des Hauptteils, beginne Muster-Erzeugungsroutine

#

#

#

MUSTER-ERZEUGUNGSRoutine

#

#

verarbeite letzten Zeilenabschnitt.

#

EndPoint = x - 1;

CurveFit (0);

VAnow=0;

#

leite Ausgabedateinamen von Eingabedateiname ab.

#

F = FILENAME;

F=substr(F,1,match(F,"[.]")-1);

PatFile = F „.vvs“;

```
ExpectFile = F „exp“;

#

# drucke ersten Datenpunkt in ExpectFile

#

print (FirstT, FirstV) >ExpectFile;

#

# schleife durch Muster Datenfelder

#

for (x=1;x<=p;x++) {

#

# berechne Ausgabespannung, Vnew, bei jedem Schritt der zeitlichen Auflösung.

# drucke Zeit/Spannung in ExpectFile.

#

if (pattern[x] == 0) {Vapp = DrvLo}

if (pattern[x] == 1) {Vapp = (0,75*DrvLo)}

if (pattern[x] == 2) {Vapp = (0,5*DrvLo)}

if (pattern[x] == 3) {Vapp = (0,25*DrvLo)}

if (pattern[x] == 4) {Vapp = 0}

if (pattern[x] == 5) {Vapp = (0,25*DrvHi)}
```

```
if (pattern[x] == 6) {Vapp = (0,5*DrvHi)}
```

```
if (pattern[x] == 7) {Vapp = (0,75*DrvHi)}
```

```
if (pattern[x] == 8) {Vapp = DrvHi}
```

```
VCa = (Vapp - Vanow) * (1 - exp(-1 * res/RCa));
```

```
VAnew = VCa + VAnow;
```

```
VCb = VAnow * (1 - exp(-1 * res/RCb));
```

```
Vnew = Vnew + VCa - Vcb;
```

```
print (x * res + FirstT, Vnew) >ExpectFile;
```

```
VAnew = VAnew;
```

```
#
```

```
# stelle fest, ob sich Muster-Datenfeldindex am Anfang eines neuen Vektors
```

```
# befindet.
```

```
# (teile den Index -1 durch die Auflösungsschritte pro Vektor, RPV.)
```

```
# (if das reale Ergebnis ungefähr mit dem ganzzahligen Teil des)
```

```
# Ergebnisses übereinstimmt, then befindet der Index sich am Anfang des
```

```
# nächsten Vektors.  )
```

```
#
```

```
Real = (x - 1)/RPV;
```

```
Max = int(Real) + 0,05;
```

```
Min = int(Real) - 0,05;
```

```
if ((Real <= Max) && (Real >= Min)) {NewVec =1}
```

```
else {NewVec = 0}
```

```
#
```

```
# wenn NewVec, dann erzeuge Vektordaten
```

```
#
```

```
if (NewVec == 1) {
```

```
    if (pattern[x] == 0) {Data = „0000000000“}
```

```
    if (pattern[x] == 1) {Data = „1000000000“}
```

```
    if (pattern[x] == 2) {Data = „1100000000“}
```

```
    if (pattern[x] == 3) {Data = „1110000000“}
```

```
    if (pattern[x] == 4) {Data = „1111000000“}
```

```
    if (pattern[x] == 5) {Data = „1111100000“}
```

```
    if (pattern[x] == 6) {Data = „1111110000“}
```

```
    if (pattern[x] == 7) {Data = „1111111000“}
```

```
    if (pattern[x] == 8) {Data = „1111111100“}}
```

```
#
```

```
# wenn nicht NewVec, dann stelle fest, ob sich die Daten im nächsten res Schritt
```

```
# ändern sollen
```

```
# und setze V-Series MZDI Mengenzahl fest basierend auf den aktuellen und
```

```
# vorherigen Daten.
```

#

```

if (NewVec == 0) {

    if (x == 1) {LastHigh = pattern[1]}

    if (LastHigh == pattern[x]) {MZDI = „ ( 1, 1) [“}

    else {

        if ((pattern[x] == 0) && LastHigh == 1)) {MZDI = „ ( 1, 2) [“}

        if ((pattern[x] == 1) && LastHigh == 2)) {MZDI = „ ( 1, 3) [“}

        if ((pattern[x] == 2) && LastHigh == 3)) {MZDI = „ ( 1, 4) [“}

        if ((pattern[x] == 3) && LastHigh == 4)) {MZDI = „ ( 1, 5) [“}

        if ((pattern[x] == 4) && LastHigh == 5)) {MZDI = „ ( 1, 6) [“}

        if ((pattern[x] == 5) && LastHigh == 6)) {MZDI = „ ( 1, 7) [“}

        if ((pattern[x] == 6) && LastHigh == 7)) {MZDI = „ ( 1, 8) [“}

        if ((pattern[x] == 7) && LastHigh == 8)) {MZDI = „ ( 1, 9) [“}

        if ((pattern[x] == 1) && LastHigh == 0)) {MZDI = „ ( 1, 2) [“}

        if ((pattern[x] == 2) && LastHigh == 1)) {MZDI = „ ( 1, 3) [“}

        if ((pattern[x] == 3) && LastHigh == 2)) {MZDI = „ ( 1, 4) [“}

        if ((pattern[x] == 4) && LastHigh == 3)) {MZDI = „ ( 1, 5) [“}

        if ((pattern[x] == 5) && LastHigh == 4)) {MZDI = „ ( 1, 6) [“}

        if ((pattern[x] == 6) && LastHigh == 5)) {MZDI = „ ( 1, 7) [“}

        if ((pattern[x] == 7) && LastHigh == 6)) {MZDI = „ ( 1, 8) [“}
    }
}

```

```

if ((pattern[x] == 8) && LastHigh == 7)) {MZDI = „ ( 1, 9) [“}

if ((pattern[x] == 0) && LastHigh == 2)) {MZDI = „ ( 1, 10) [“}

if ((pattern[x] == 1) && LastHigh == 3)) {MZDI = „ ( 1, 11) [“}

if ((pattern[x] == 2) && LastHigh == 4)) {MZDI = „ ( 1, 12) [“}

if ((pattern[x] == 3) && LastHigh == 5)) {MZDI = „ ( 1, 13) [“}

if ((pattern[x] == 4) && LastHigh == 6)) {MZDI = „ ( 1, 14) [“}

if ((pattern[x] == 5) && LastHigh == 7)) {MZDI = „ ( 1, 15) [“}

if ((pattern[x] == 6) && LastHigh == 8)) {MZDI = „ ( 1, 16) [“}

if ((pattern[x] == 2) && LastHigh == 0)) {MZDI = „ ( 1, 10) [“}

if ((pattern[x] == 3) && LastHigh == 1)) {MZDI = „ ( 1, 11) [“}

if ((pattern[x] == 4) && LastHigh == 2)) {MZDI = „ ( 1, 12) [“}

if ((pattern[x] == 5) && LastHigh == 3)) {MZDI = „ ( 1, 13) [“}

if ((pattern[x] == 6) && LastHigh == 4)) {MZDI = „ ( 1, 14) [“}

if ((pattern[x] == 7) && LastHigh == 5)) {MZDI = „ ( 1, 15) [“}

if ((pattern[x] == 8) && LastHigh == 6)) {MZDI = „ ( 1, 16) [“}

if ((pattern[x] == 0) && LastHigh == 3)) {MZDI = „ ( 1, 17) [“}

if ((pattern[x] == 1) && LastHigh == 4)) {MZDI = „ ( 1, 18) [“}

if ((pattern[x] == 2) && LastHigh == 5)) {MZDI = „ ( 1, 19) [“}

if ((pattern[x] == 3) && LastHigh == 6)) {MZDI = „ ( 1, 20) [“}

if ((pattern[x] == 4) && LastHigh == 7)) {MZDI = „ ( 1, 21) [“}

```

```

if ((pattern[x] == 5) && LastHigh == 8)) {MZDI = ,, ( 1, 22) [{"}
if ((pattern[x] == 3) && LastHigh == 0)) {MZDI = ,, ( 1, 17) [{"}
if ((pattern[x] == 4) && LastHigh == 1)) {MZDI = ,, ( 1, 18) [{"}
if ((pattern[x] == 5) && LastHigh == 2)) {MZDI = ,, ( 1, 19) [{"}
if ((pattern[x] == 6) && LastHigh == 3)) {MZDI = ,, ( 1, 20) [{"}
if ((pattern[x] == 7) && LastHigh == 4)) {MZDI = ,, ( 1, 21) [{"}
if ((pattern[x] == 8) && LastHigh == 5)) {MZDI = ,, ( 1, 22) [{"}
if ((pattern[x] == 0) && LastHigh == 4)) {MZDI = ,, ( 1, 23) [{"}
if ((pattern[x] == 1) && LastHigh == 5)) {MZDI = ,, ( 1, 24) [{"}
if ((pattern[x] == 2) && LastHigh == 6)) {MZDI = ,, ( 1, 25) [{"}
if ((pattern[x] == 3) && LastHigh == 7)) {MZDI = ,, ( 1, 26) [{"}
if ((pattern[x] == 4) && LastHigh == 8)) {MZDI = ,, ( 1, 27) [{"}
if ((pattern[x] == 4) && LastHigh == 0)) {MZDI = ,, ( 1, 23) [{"}
if ((pattern[x] == 5) && LastHigh == 1)) {MZDI = ,, ( 1, 24) [{"}
if ((pattern[x] == 6) && LastHigh == 2)) {MZDI = ,, ( 1, 25) [{"}
if ((pattern[x] == 7) && LastHigh == 3)) {MZDI = ,, ( 1, 26) [{"}
if ((pattern[x] == 8) && LastHigh == 4)) {MZDI = ,, ( 1, 27) [{"}
if ((pattern[x] == 0) && LastHigh == 5)) {MZDI = ,, ( 1, 28) [{"}
if ((pattern[x] == 1) && LastHigh == 6)) {MZDI = ,, ( 1, 29) [{"}
if ((pattern[x] == 2) && LastHigh == 7)) {MZDI = ,, ( 1, 30) [{"}

```

```

if ((pattern[x] == 3) && LastHigh == 8)) {MZDI = „ ( 1, 31) [“}

if ((pattern[x] == 5) && LastHigh == 0)) {MZDI = „ ( 1, 28) [“}

if ((pattern[x] == 6) && LastHigh == 1)) {MZDI = „ ( 1, 29) [“}

if ((pattern[x] == 7) && LastHigh == 2)) {MZDI = „ ( 1, 30) [“}

if ((pattern[x] == 8) && LastHigh == 3)) {MZDI = „ ( 1, 31) [“}

if ((pattern[x] == 0) && LastHigh == 6)) {MZDI = „ ( 1, 32) [“}

if ((pattern[x] == 1) && LastHigh == 7)) {MZDI = „ ( 1, 33) [“}

if ((pattern[x] == 2) && LastHigh == 8)) {MZDI = „ ( 1, 34) [“}

if ((pattern[x] == 6) && LastHigh == 0)) {MZDI = „ ( 1, 32) [“}

if ((pattern[x] == 7) && LastHigh == 1)) {MZDI = „ ( 1, 33) [“}

if ((pattern[x] == 8) && LastHigh == 2)) {MZDI = „ ( 1, 34) [“}

if ((pattern[x] == 0) && LastHigh == 7)) {MZDI = „ ( 1, 35) [“}

if ((pattern[x] == 1) && LastHigh == 8)) {MZDI = „ ( 1, 36) [“}

if ((pattern[x] == 7) && LastHigh == 0)) {MZDI = „ ( 1, 35) [“}

if ((pattern[x] == 8) && LastHigh == 1)) {MZDI = „ ( 1, 36) [“}

if ((pattern[x] == 0) && LastHigh == 8)) {MZDI = „ ( 1, 37) [“}

if ((pattern[x] == 8) && LastHigh == 0)) {MZDI = „ ( 1, 37) [“}

```

```
#
```

```
# drucke den Vektor in die Musterdatei
```

```
#
```

```
print MZDI Daten “];“ >PatFile}

LastHigh = pattern[x]}}

# Ende der Muster-Erzeugungsroutine.

#

#

#

#           KURVEN-ANPASSUNGS-ROUTINE

#

#

#

# die folgende Funktion findet die angelegte Spannung für jeden

# Auflösungsschritt heraus

# um der Ausgabe die beste Anpassung an gewünschte Kurve zu geben.

#

function CurveFit (SlopeChange) {

#

# berechne den Spannungsunterschied, dV, und den Zeitunterschied, dT.

# passe dT an Auflösung der Testeinheit an.

#

CheckSlope = 0;
```

```

VAend = v[EndPoint];

dV = VAend - VA now;

dT = t [EndPoint] - t[1];

Zeit = int(dT / res);

d1 = dT - (time * res);

d2 = (res * (time + 1)) - dT;

if (d1 < 0) {d1 = -1 * d1}

if (d2 < 0) {d2 = -1 * d2}

if (d1 > d2) { time ++}

dT = res * time;

#

# berechne ideale angelegte Spannung, Vapp, passe dann an

# Auflösung des Spannungsteilers an. setze NumHi auf Anzahl hoher

# Eingangssignale fest

# um gewünschte Spannung zu erhalten. kopiere NumHi in Muster-Datenfeld.

#

if (dT == 0) {print $1,$2}

Vapp = dV / (1 - exp(-1 * dT/RCa)) + VAnow;

if (Vapp <= (0,875*DrvLo))                {Vapp = DrvLo;

NumHi

```

= 0}

if ((Vapp > (0,875*DrvLo)) && (Vapp <= (0,625*DrvLo)))

{Vapp =

(0,75*DrvLo); NumHi

= 1}

if ((Vapp > (0,625*DrvLo)) && (Vapp <= (0,375*DrvLo)))

{Vapp = (0,5*DrvLo);

NumHi

= 2}

if ((Vapp > (0,375*DrvLo)) && (Vapp <= (0,125*DrvLo)))

{Vapp = (0,25*DrvLo);

NumHi

= 3}

if ((Vapp > (0,125*DrvLo)) && (Vapp <= (0,125*DrvHi)))

{Vapp = 0;

NumHi

= 4}

if ((Vapp > 0,125* DrvHi)) && (Vapp <= (0,375* DrvHi)))

{Vapp = (0,25* DrvHi);

NumHi

= 5}

```
if ((Vapp > (0,375* DrvHi)) && (Vapp <= (0,625* DrvHi)))
```

```
{Vapp = (0,5* DrvHi);
```

```
NumHi
```

```
= 6}
```

```
if ((Vapp > (0,625* DrvHi)) && (Vapp <= (0,875* DrvHi)))
```

```
{Vapp = (0,75* DrvHi);
```

```
NumHi
```

```
= 7}
```

```
if ((Vapp > (0,875* DrvHi))
```

```
{Vapp = DrvHi;
```

```
NumHi
```

```
= 8}
```

```
for (y=1;y<=Zeit;y++) {Muster[p + y] = NumHi}
```

```
#
```

```
# berechne die neue Spannung während der ersten Phase, VAnew, die mit Vapp
```

```
# nach dT erwartet wird.
```

```
#
```

```
VCa = (Vapp - VAnow) * (1 - exp(-1 * dT/RCa));
```

```
VAnew = VCa + VAnow;
```

```
#
```

```

# wenn VAnew niedriger als die benötigte Spannung VAend ist, dann versuche

# Anpassung der Musterdaten zu verbessern.

#

increment = 0;

if ((VAnew < NAend) && (NumHi < 8)) { increment = 1}

if (increment) {

y = 1;

NotDone = 1;

#

# nur die ersten 20 ns können angepasst werden (verhindert Unebenheiten in

# Kurve)

#

while (((y*res)<=20e-9) && (NotDone)) {

V1 = (Vapp + 0,25 - VAnow) * (1 - exp(-1 * (y * res)/RCa));

V2 = (Vapp - (VAnow + V1)) * (1 - exp(-1 * ((time-y)*res)/RCa));

#

# wenn immer noch niedrig nach vorhergesagter Anpassung, dann passe weiter an

# und wiederhole Prozess

#

if ((V1 + V2 + VAnow) < VAend) {pattern[p + y]++; VAnew = V1 + V2 +
VAnow}

```

#

sonst stelle fest, ob angepasst oder unangepasst bessere Form hat

#

else {

NotDone = 0;

d1 = VAend - VAnew;

d2 = (V1 + V2 + VAnow) - VAend;

if (d1 < 0) {d1 = -1 * d1}

if (d2 < 0) {d2 = -1 * d2}

if (d1 > d2) {pattern[p + y]++;VAnew = V1 + V2 + VAnow}}

y++}}

#

wenn VAnew größer als die benötigte Spannung VAend ist, dann versuche

Anpassung der Musterdaten zu verbessern.

#

decrement = 0;

if ((VAnew > VAend) && (NumHi > 0)) { decrement = 1 }

if ((decrement == 1) && (increment == 0)) {

y = 1;

NotDone = 1;

#

nur die ersten 20 ns können angepasst werden (verhindert Unebenheiten in

Kurve)

#

while (((y*res)<=20e-9) && (NotDone)) {

V1 = (Vapp - 0,25 - VAnow) * (1 - exp(-1 * (y * res)/RCa));

V2 = (Vapp - (VAnow + V1)) * (1 - exp(-1 * ((time-y)*res)/RCa));

#

wenn immer noch hoch nach vorhergesagter Anpassung, dann passe weiter an

und wiederhole Prozess

#

if ((V1 + V2 + VAnow) > VAend) {pattern[p + y]--; VAnew = V1 + V2 + VAnow}

#

sonst stelle fest, ob angepasst oder unangepasst bessere Form hat

#

else {

NotDone = 0;

d1 = VAnew - VAend;

d2 = VAend - (V1 + V2 + VAnow);

if (d1 < 0) {d1 = -1 * d1}

```
if (d2 < 0) {d2 = -1 * d2}
```

```
if (d1 > d2) {pattern[p + y]--;VANew = V1 + V2 + VANow}}
```

```
y++}}
```

```
#
```

```
# setze Variablen für das Lesen und Verarbeiten der nächsten Kurve fest.
```

```
#
```

```
p = p + time;
```

```
t[1] = SaveT;
```

```
v[1] = SaveV;
```

```
x = 2;
```

```
VANow = VANew} # Ende der CurveFit Funktion.
```

[0036] Gemäß einer offenbaren Klasse von innovativen Ausführungsbeispielen wird folgendes bereit gestellt: ein Verfahren zum Erzeugen beliebiger Signalformen, das die folgenden Aktionen umfasst: Empfangen einer Menge von Datenpunkten, die eine zu erzeugende Signalform beschreibt; Erzeugen einer Menge von Vektoren aus der Menge von Datenpunkten, die eine digitale Testeinheit ansteuert; und Ansteuern der digitalen Testeinheit mit der Menge von Vektoren; wobei die digitale Testeinheit die Widerstände eines Widerstandsteilers in Übereinstimmung mit der Menge von Vektoren ansteuert, um einen Teststimulus zu erzeugen, der der Signalform im wesentlichen ähnlich ist und in eine zu testende Vorrichtung eingegeben wird.

[0037] Gemäß einer anderen offenbaren Klasse von innovativen Ausführungsbeispielen wird folgendes bereit gestellt: ein Verfahren zum Erzeugen beliebiger Signalformen, das die folgenden Aktionen umfasst: Empfangen einer Menge von Datenpunkten, die eine zu erzeugende Signalform beschreibt; Identifizieren von Steigungs- oder Spannungsänderungen in der Menge von Datenpunkten als Endpunkte; Berechnen der Spannungen, die sich an den Endpunkten ergeben; Bestimmen der Anzahl von Widerständen in einer Generatorschaltung für pseudobeliebige Signalformen, um auf der Grundlage der Ergebnisse des Berechnungsschrittes auf Hochpegel anzusteuern; Umsetzen des Ergebnisses des Bestimmungsschrittes in Spannungswerte, die an die Widerstände angelegt werden sollen; und Erzeugen einer Menge von Vektoren aus dem Ergebnis des Umsetzungsschrittes, die eine digitale Testeinheit ansteuern kann; wobei die digitale Testeinheit die Widerstände der Schaltung in Übereinstimmung mit der Menge von Vektoren ansteuert, um einen Teststimulus zu erzeugen, der der Signalform im wesentlichen ähnlich ist und in eine zu testende Vorrichtung eingegeben wird.

[0038] Gemäß einer anderen offenbaren Klasse von innovativen Ausführungsbeispielen wird folgendes bereit gestellt: ein System zum Erzeugen beliebiger Signalformen, das umfasst: eine Menge von Datenpunkten, die eine zu erzeugende Signalform beschreibt; ein Computersystem, das so konfiguriert ist, dass es die Menge von Datenpunkten empfängt und aus der Menge von Datenpunkten eine Menge von Vektoren erzeugt, die eine digitale Testeinheit ansteuert; eine digitale Testeinheit, die so angeschlossen ist, dass sie die Menge von Vektoren empfängt; und eine Schaltung, die einen Widerstandsteiler umfasst und so angeschlossen ist, dass sie durch den Ausgang der digitalen Testeinheit angesteuert wird und einen Teststimulus erzeugt, der der Signalform im wesentlichen ähnlich ist und in eine zu testende Vorrichtung eingegeben wird.

Modifikationen und Variationen

[0039] Der Fachmann wird schnell erkennen, dass die innovativen Konzepte, die in der vorliegenden Anmeldung beschrieben sind, modifiziert und in einem enormen Umfang von Anwendungsmöglichkeiten abgeändert

werden können. Dementsprechend wird der Umfang des patentierten Gegenstandes durch keine der spezifischen beispielhaften Lehren beschränkt, sondern er wird ausschließlich durch die erteilten Ansprüche definiert.

[0040] Das derzeit bevorzugte Ausführungsbeispiel wird als Software beschrieben, die in der Lage ist, zu steuern, und ein Schaltkreis mit einem Widerstandsteiler mit 8 Widerständen. Der Widerstandsteiler kann jedoch mit einer höheren oder niedrigeren Anzahl von Widerständen aufgebaut sein und die Software kann dementsprechend umgeschrieben werden.

[0041] Das derzeit bevorzugte Ausführungsbeispiele wird als Software beschrieben, die die Widerstände eines Widerstandsteilers mit 0,1 Volt ansteuert. Die Hoch- und Niedrigspannungen können jedoch an die Art der gewünschten Teststimuli angepasst werden.

[0042] Das derzeit bevorzugte Ausführungsbeispiel beschreibt Softwareroutinen für die Erstellung von Treibern für Schaltkreise sowohl für V-Series also auch Polaris Testgeneratoren. Die Software ist jedoch für beide ähnlich und der Code kann leicht modifiziert werden, um Treiberdateien für andere Ausführungen von Testeinheiten zu erzeugen, die wiederum den beschriebenen Schaltkreis ansteuern können. Jede VLSI-Testeinheit, die keinen eigenständigen Hochgeschwindigkeits-AWG hat, inklusive solcher Testeinheiten wie die Teradyne J-97x Serie, kann mit dem beschriebenen PAWG benutzt werden.

Patentansprüche

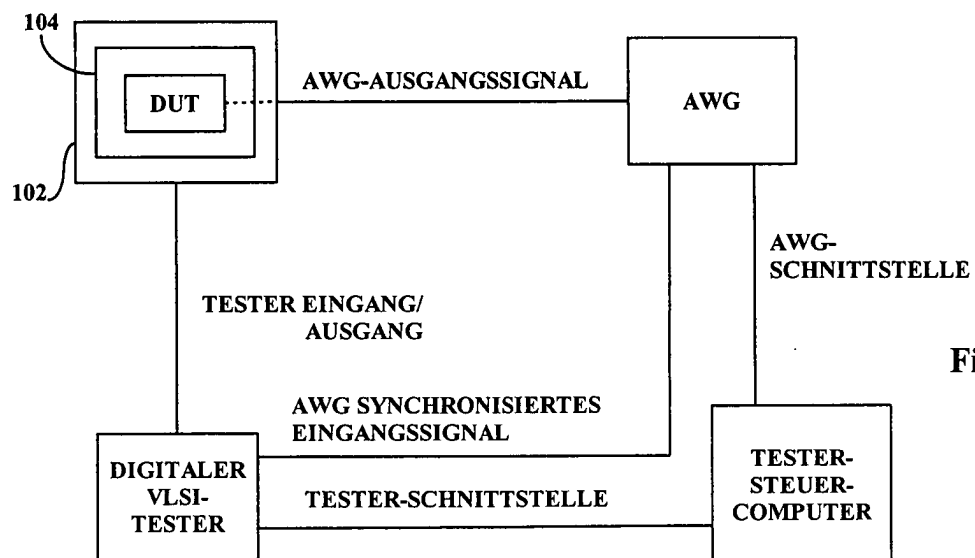
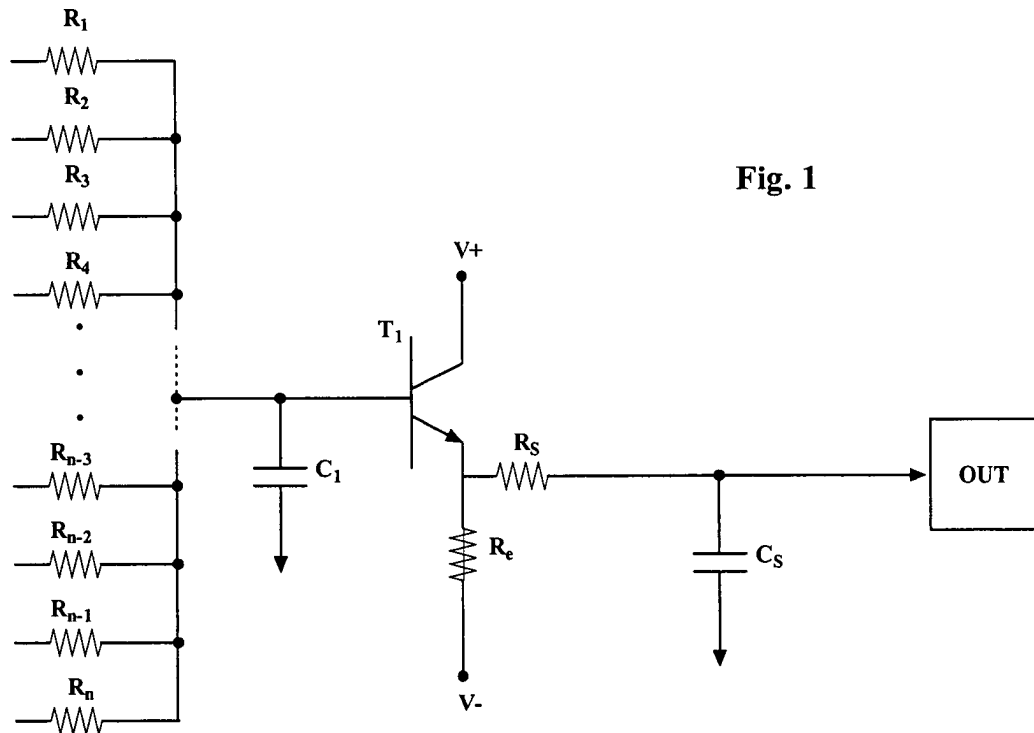
1. Verfahren zum Erzeugen beliebiger Signalformen, das die folgenden Aktionen umfaßt:
Empfangen einer Menge von Datenpunkten, die eine zu erzeugende Signalform beschreibt;
Erzeugen einer Menge von Vektoren aus der Menge von Datenpunkten, die eine digitale Testeinheit ansteuert;
und
Ansteuern der digitalen Testeinheit mit der Menge von Vektoren;
wobei die digitale Testeinheit die Widerstände eines Widerstandsteilers (R_1 - R_n) in Übereinstimmung mit der Menge von Vektoren ansteuert, um einen Teststimulus zu erzeugen, der der Signalform im wesentlichen ähnlich ist und in eine zu testende Vorrichtung eingegeben wird.
2. Verfahren nach Anspruch 1, bei dem die Menge von Datenpunkten durch Digitalisieren einer gewünschten Signalform erzeugt wird.
3. Verfahren nach Anspruch 1 oder Anspruch 2, bei dem die Menge von Vektoren von der Auflösung des Teststimulus abhängt.
4. Verfahren nach Anspruch 1, Anspruch 2 oder Anspruch 3, bei dem ein mit dem Widerstandsteiler verbundener Kondensator die diskrete Zeitausgabe des Widerstandsteilers in eine kontinuierliche Zeitausgabe umsetzt.
5. Verfahren nach einem vorhergehenden Anspruch, das die folgenden Aktionen umfaßt:
Identifizieren von Steigungs- oder Spannungsänderungen in der Menge von Datenpunkten als Endpunkte;
Berechnen der Spannungen, die sich an den Endpunkten ergeben;
Bestimmen der Anzahl von Widerständen einer Generatorschaltung für pseudobeliebige Signalformen, um auf der Grundlage der Ergebnisse des Berechnungsschrittes auf Hochpegel anzusteuern; und
Umsetzen des Ergebnisses des Bestimmungsschrittes in Spannungswerte, die an die Widerstände angelegt werden sollen.
6. Verfahren nach Anspruch 5, bei dem der Bestimmungsschritt ferner die folgenden Schritte umfaßt:
Berechnen der erwarteten Spannung anhand der Ansteuerung der bestimmten Anzahl von Widerständen auf Hochpegel, Vorhersagen einer erwarteten Spannung nach einer gegebenen Zeit, Einstellen der Anzahl der Widerstände auf der Grundlage der vorhergesagten erwarteten Spannung, um sie auf Hochpegel anzusteuern, Einstellen der Anzahl von Widerständen, die auf Hochpegel angesteuert werden sollen, und Wiederholen der weiteren Schritte, bis eine gewünschte Genauigkeit erzielt ist.
7. System zum Erzeugen beliebiger Signalformen, das umfaßt:
Mittel, die eine Menge von Datenpunkten empfangen, die eine zu erzeugende Signalform beschreibt;
ein Computersystem, das so konfiguriert ist, daß es die Menge von Datenpunkten empfängt und aus der Menge von Datenpunkten eine Menge von Vektoren erzeugt, die eine digitale Testeinheit ansteuert;
eine digitale Testeinheit, die so angeschlossen ist, daß sie die Menge von Vektoren empfängt; und
eine Schaltung, die einen Widerstandsteiler (R_1 - R_n) umfaßt und so angeschlossen ist, daß sie durch den Ausgang der digitalen Testeinheit angesteuert wird und einen Teststimulus erzeugt, der der Signalform im wesent-

lichen ähnlich ist und in eine zu testende Vorrichtung eingegeben wird.

8. System nach Anspruch 7, bei dem ein mit dem Widerstandsteiler verbundener Kondensator die diskrete Zeitausgabe des Widerstandsteilers in eine kontinuierliche Zeitausgabe umsetzt.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen



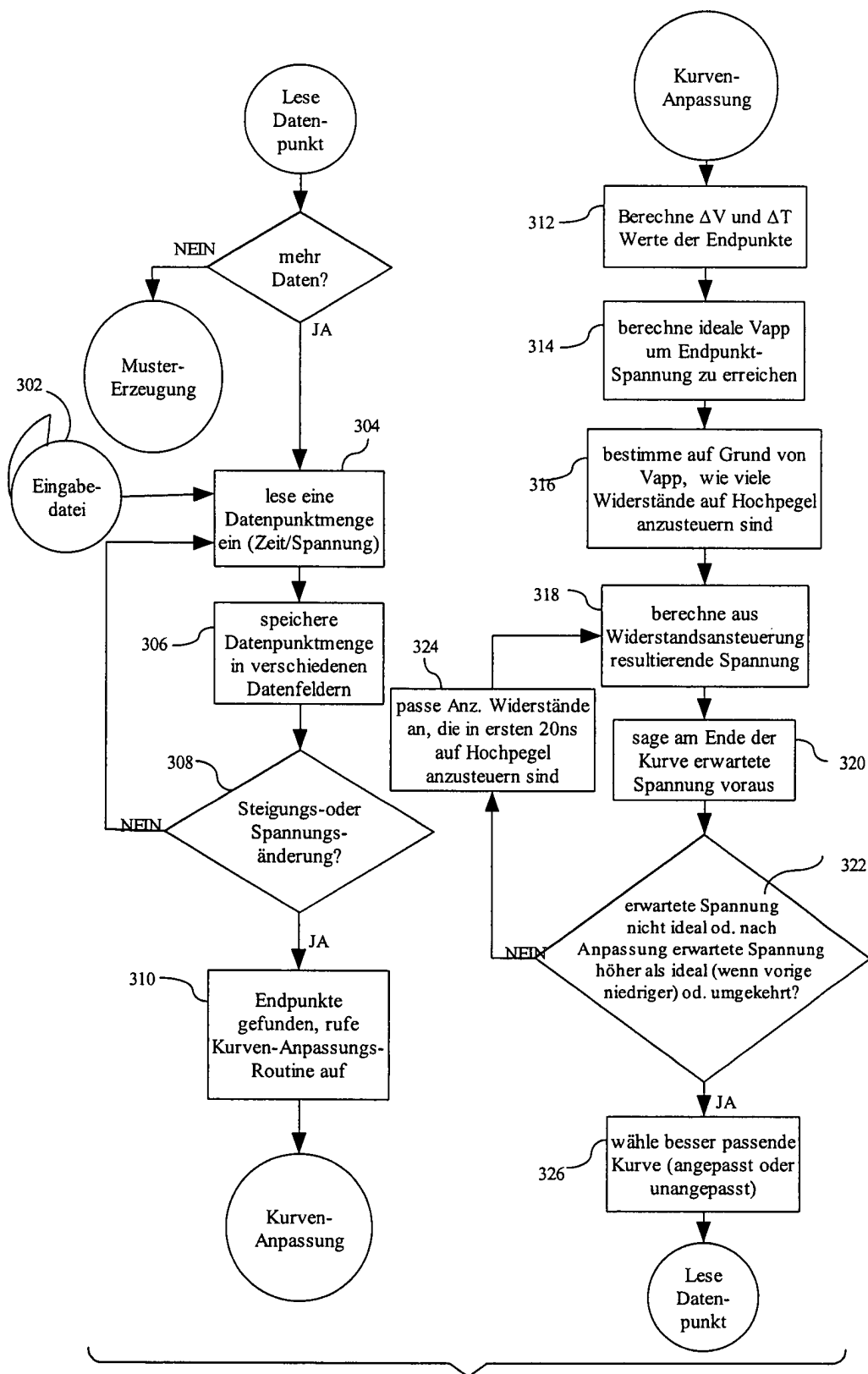


Fig. 3A

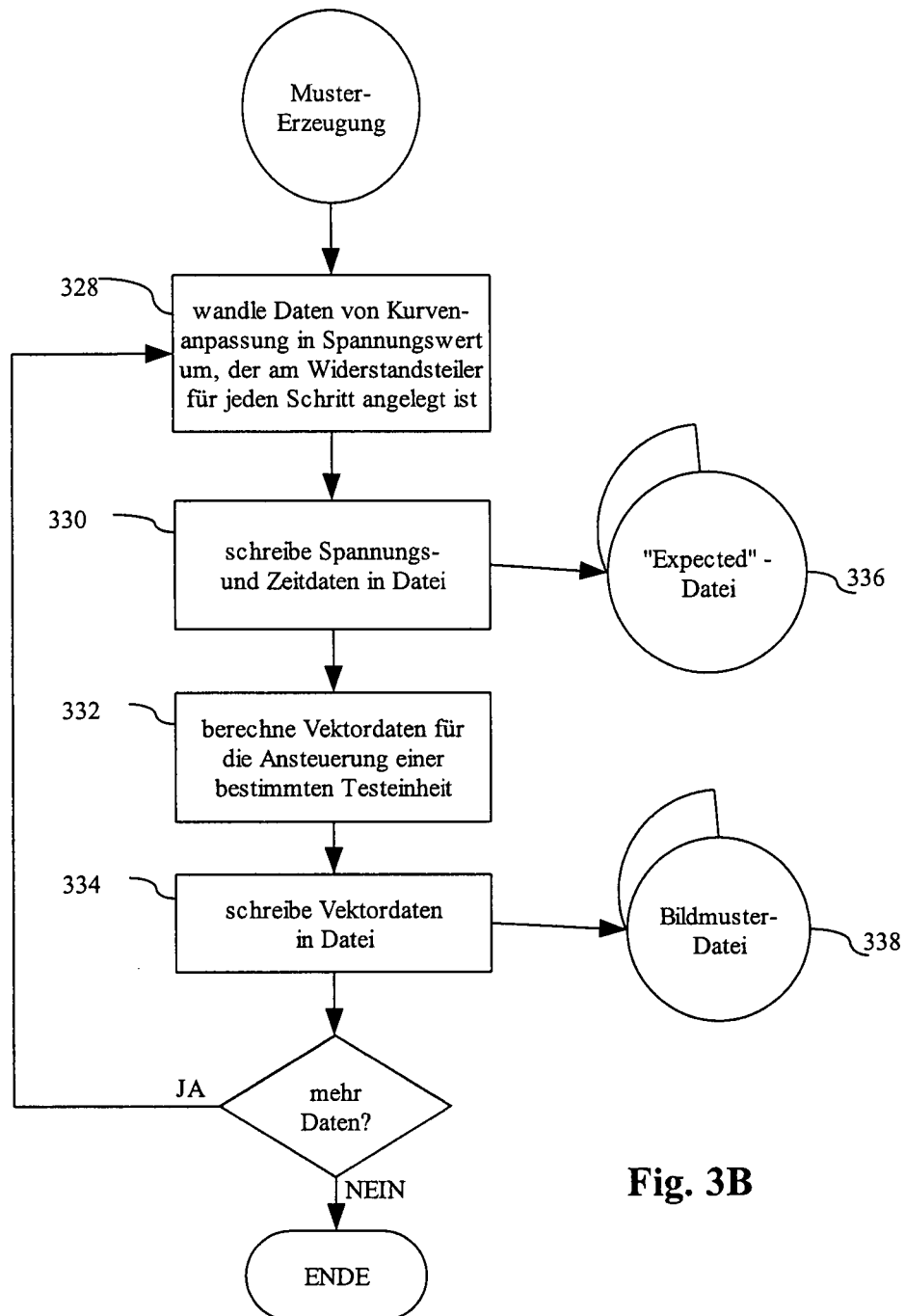


Fig. 3B