(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 60 2004 011 320 T2** 2009.02.05

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 592 976 B1**
(21) Deutsches Aktenzeichen: **60 2004 011 320.4**
(86) PCT-Aktenzeichen: **PCT/JP2004/001649**
(96) Europäisches Aktenzeichen: **04 711 471.5**
(87) PCT-Veröffentlichungs-Nr.: **WO 2004/072670**
(86) PCT-Anmeldetag: **16.02.2004**
(87) Veröffentlichungstag der PCT-Anmeldung: **26.08.2004**
(97) Erstveröffentlichung durch das EPA: **09.11.2005**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **16.01.2008**
(47) Veröffentlichungstag im Patentblatt: **05.02.2009**

(51) Int Cl.[8]: ***G01R 31/319*** (2006.01)
G01R 31/3183 (2006.01)
G06F 11/263 (2006.01)

(30) Unionspriorität:

447839 P	**14.02.2003**	**US**
449622 P	**24.02.2003**	**US**
403817	**31.03.2003**	**US**
404002	**31.03.2003**	**US**

(84) Benannte Vertragsstaaten:
AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LI, LU, MC, NL, PT, RO, SE, SI, SK, TR

(73) Patentinhaber:
Advantest Corp., Tokio/Tokyo, JP

(74) Vertreter:
PFENNING MEINIG & PARTNER GbR, 10719 Berlin

(72) Erfinder:
KRISHNASWAMY, Ramachandran, Nerima-ku, Tokyo 179-0071, JP; SINGH, Harsanjeet, Nerima-ku, Tokyo 179-0071, JP; PRAMANICK, Ankan, Nerima-ku, Tokyo 179-0071, JP; ELSTON, Mark, Nerima-ku, Tokyo 179-0071, JP; CHEN, Leon, Nerima-ku, Tokyo 179-0071, JP; ADACHI, Toshiaki, Nerima-ku, Tokyo 179-0071, JP; TAHARA, Yoshihumi, Nerima-ku, Tokyo 179-0071, JP

(54) Bezeichnung: **VERFAHREN UND STRUKTUR ZUR ENTWICKLUNG EINES TESTPROGRAMMS FÜR INTEGRIERTE HALBLEITERSCHALTUNGEN**

Beschreibung

QUERVERWEIS AUF ZUGEORDNETE ANMELDUNG

[0001] Diese Anmeldung erhebt Anspruch auf den Nutzen der am 14. Februar 2003 eingereichten US-Anmeldung Nr. 60/447 839 „Verfahren und Struktur zur Entwicklung eines Testprogramms für Halbleiterschaltkreise"; der am 14. Februar 2003 eingereichten US-Anmeldung Nr. 60/449 622 „Verfahren und Gerät zum Testen von Schaltkreisen"; der am 31. März 2003 eingereichten US-Anmeldung Nr. 10/404 002 „Prüfemulator, Prüfmodulemulator und Aufzeichnungsmedium, das Programme darin speichert"; sowie der am 31. März 2003 eingereichten US-Anmeldung Nr. 10/403 817 „Prüfgerät und Prüfverfahren". Diese Anmeldung bezieht sich auch auf die gleichzeitig hiermit eingereichte US-Anmeldung Nr. 10/772 327 „Verfahren und Gerät zum Prüfen von Schaltkreisen", die den Anspruch auf den Nutzen der am 24. Februar 2003 eingereichten US-Anmeldung Nr. 60/449 622 „Verfahren und Gerät zum Prüfen von Schaltkreisen" erhebt.

HINTERGRUND DER ERFINDUNG

Gebiet der Erfindung

[0002] Die vorliegende Erfindung betrifft das Testen von integrierten Schaltkreisen (IC) und spezieller die Entwicklung eines Testprogramms für automatische Halbleiter-Prüfanlage (ATE).

Beschreibung des Standes der Technik

[0003] Heutige Prüfgerätehersteller verwenden ihre eigenen gesetzlich geschützten Unternehmenssprachen, um Testprogramme für Halbleiter-Testsysteme (Prüfgeräte) zu entwickeln. Zum Beispiel nutzen von Advantest Corporation hergestellte Maschinen die Sprache zur Testbeschreibung (TDL), und Credence Systems bietet seine eigene Sprache zur Wellenformerzeugung (WGL) an. Um diesen Spezialisierungsgrad zu überwinden, haben Hersteller von Schaltkreisprüfgeräten versucht, eine gemeinsame Grundlage durch Entwicklung der IEEE-Norm 1450, die Standard-Test-Interface-Language (Sprache für normale Testschnittstellen, STIL) zu finden. STIL ist jedoch eine hoch spezialisierte Sprache zur Definition persönlicher Identifikationsziffern, Testbefehlen, zeitlicher Steuerung, usw.. Außerdem muss ein mit STIL arbeitender Prüfingenieur trotzdem noch STIL in die vom Prüfgerät geforderte, gesetzlich geschützte herstellerspezifische Sprache übersetzen. Somit dient STIL lediglich als eine Zwischensprache, die dennoch hoch spezialisiert ist und Programmierern nicht generell bekannt ist.

[0004] Daher ist es erwünscht, ein Verfahren zu entwickeln, durch welches ein Testprogramm in einer Universalsprache geschrieben werden kann. Außerdem sollte dieses Verfahren die leichte Entwicklung von Testprogrammen für Testsysteme mit offener Architektur zulassen.

[0005] Die Druckschrift US-A-2002/0073375 zeigt ein Verfahren zur Entwicklung eines Testprogramms in Universal-C/C++-Konstrukten. Der zugeordnete Stand der Technik kann auch in den Druckschriften US-A-5 488 573, US-B1-6 195 774 und US-A-2003/0005375 gefunden werden.

ABRISS DER ERFINDUNG

[0006] Diese Anmeldung beschreibt die Entwicklung eines Testprogramms unter Verwendung von objektorientierten Konstrukten, zum Beispiel C++-Objekte und -klassen. Insbesondere ist dieses Verfahren geeignet zur Entwicklung von Testprogrammen für ein Prüfgerät offener Architektur wie beispielsweise das in den US-Anmeldungen Serien-Nr. 60/449 662, 10/404 002 und 10/403 817 beschriebene, das dem Rechtsnachfolger der vorliegenden Erfindung erteilt ist. Eine Ausführung der vorliegenden Erfindung stellt ein Verfahren zur Entwicklung eines Testprogramms dadurch bereit, dass Testsystemressourcen, Testsystemkonfiguration, Modulkonfiguration, Testsequenz, Testplan, Testbedingung, Testmuster und Informationen der zeitlichen Steuerung in objektorientierten Universalkonstrukten, z. B. C++ Konstrukte zum Testen eines Prüfobjekts, z. B. ein IC auf einem Halbleitertestsystem wie eine automatische Prüfeinrichtung (ATE), beschrieben werden. Daten, die diese Beschreibungen enthalten, werden in einem Speicher, d. h. ein computerlesbares Medium gespeichert, die dem Testsystem oder einer zugeordneten Ausrüstung, die diese Daten nutzt, zugänglich sind.

[0007] Das Beschreiben von Testsystemressourcen kann die Spezifizierung eines Ressourcentyps umfassen, wobei der Ressourcentyp mit zumindest einem Testmodul zur Anwendung eines Tests an dem IC assoziiert ist, indem ein mit dem Ressourcentyp verbundener Parametertyp und ein Parameter des Parametertyps

bestimmt werden.

[0008] Das Beschreiben der Konfiguration des Testsystems kann die Spezifizierung eines Site-Controllers zum Kontrollieren von mindestens einem Testmodul umfassen, wobei jedes Testmodul an dem IC einen Test anwendet und einen Eingangsport eines Modulverbindungs-Enablers festlegt. Das Testsystem koppelt den Site-Controller am festgelegten Eingangsport an den Modulverbindungs-Enabler, und der Modulverbindungs-Enabler koppelt den Site-Controller an ein Testmodul. Der Modulverbindungs-Enabler kann als eine Switchmatrix implementiert werden.

[0009] Das Beschreiben der Modulkonfiguration kann die Spezifizierung eines Modulidentifizierers zum Spezifizieren eines Modultyps umfassen, der einen ausführbaren Code zum Steuern eines Testmoduls des Modultyps, der durch den Modulidentifizierer bestimmt ist, und einen mit dem Testmodul assoziierten Ressourcentyp festlegt. Der ausführbare Code kann die Form einer Datei für Betriebssystemroutinen (DLL) annehmen.

[0010] Das Beschreiben der Modulkonfiguration kann ferner den Anwender einbeziehen, der einen Slot-Identifizierer zum Spezifizieren eines Ausgangsports des Modulverbindungs-Enablers bestimmt, wobei das Testsystem das Testmodul an den Modulverbindungs-Enabler am Ausgangsport koppelt und der Modulverbindungs-Enabler das Testmodul an einen korrespondierenden Site-Controller koppelt. Der Anwender kann außerdem einen Hersteller-Identifizierer zum Identifizieren des Bereitstellers des Testmoduls und einen Identifizierer der maximalen Anzahl von in Verbindung mit dem Ressourcentyp verfügbaren Ressourceneinheiten festlegen. Der Ressourcentyp kann zum Beispiel digitale Tester-Pins und die Testerkanäle der Ressourceneinheiten sein. Alternativ dazu können die Testerkanal-Ressourceneinheiten auch Ressourcentypen wie zum Beispiel analoge Testerpins, Hochfrequenz-Testerpins, Stromversorgungspins, Digitalisiereinrichtungspins und willkürliche Wellenformerzeugungspins entsprechen. Es kann auch ein Anzeigeelement vorgesehen werden, das darauf verweist, welche Ressourceneinheiten arbeitsunfähig sind. Die als arbeitsunfähig angezeigten Ressourceneinheiten können fehlerhafte Ressourceneinheiten des Testmoduls darstellen.

[0011] Das Beschreiben von Testbedingungen kann das Festlegen von wenigstens einer Testbedingungsgruppe, das Festlegen eines Spezifizierungssets einschließlich zumindest einer Variablen, und das Festlegen eines Selektors zum Auswählen eines mit der Variablen zu verbindenden Ausdrucks umfassen. Eine Verknüpfung der Testbedingungsgruppe mit einem Selektor für den Spezifizierungsset definiert eine Testbedingung.

[0012] Das Beschreiben einer Testsequenz kann das Festlegen der Reihenfolge (oder Ablauf) umfassen, bei der verschiedene Tests angewandt werden können.

[0013] Das Beschreiben von Testmustern kann das Festlegen der Testmuster, damit verbundener Spannungs- und Strompegel, Übergänge in Signalwerten, entsprechender Anstiegs- und Abfallzeiten und einer zugeordneten zeitlichen Steuerung umfassen.

[0014] Eine Ausführung der vorliegenden Erfindung umfasst auch die Verwendung von Preheader-Dateien. Eine Preheader-Datei wird kompiliert, um eine Kopfdatei für eine einem Testgrundelement zugeordnete Klasse zu erzeugen. Der Preheader enthält einen Parameterblock zum Spezifizieren von Parametern, um wenigstens eine Eigenschaft des Testgrundelements zu setzen, und einen Dokumentvorlageblock zum Spezifizieren eines Quellencodes, der durch einen Kompilierer in die Kopfdatei für die Testgrundelementklasse eingesetzt wird. Die Kopfdatei kann eine C++-Kopfdatei sein. Das Testgrundelement kann zum Beispiel ein Test sein, und die Testgrundelementklasse kann eine Testklasse sein. Die Parameter können sich zum Beispiel auf Strukturlisten und Testbedingungen beziehen.

[0015] Ein Strukturkompilierer nach einer Ausführung der Erfindung umfasst zumindest einen modulspezifischen Strukturkompilierer und einen Objektdatei-Manager zum Leiten jedes modulspezifischen Kompilierers, um sowohl einen entsprechenden modulspezifischen Abschnitt einer Strukturquellendatei als auch einen gemeinsamen Abschnitt der Strukturquellendatei zu kompilieren. Der gemeinsame Abschnitt enthält Informationen, die für alle modulspezifischen Kompilierer zugänglich sind. Eine Ausgabe des Kompilierers enthält zumindest einen modulspezifischen Strukturdatenabschnitt. Modulspezifische Strukturladeprogramme laden zur Ausführung entsprechende modulspezifische Strukturdaten der Testmodule von entsprechenden modulspezifischen Strukturdatenabschnitten.

KURZE BESCHREIBUNG DER ZEICHNUNGEN

[0016] Fig. 1 stellt die Architektur eines normalen Testers dar;

[0017] Fig. 2 stellt die Tester-Architektur nach einer Ausführung der vorliegenden Erfindung dar;

[0018] Fig. 3 stellt die Architektur einer Tester-Software nach einer Ausführung der vorliegenden Erfindung dar;

[0019] Fig. 4 stellt einen Testprogramm-Kompilierer nach einer Ausführung der vorliegenden Erfindung dar;

[0020] Fig. 5 veranschaulicht, wie unterschiedliche Testbeispiele aus einer einzelnen Testklasse nach einer Ausführung der vorliegenden Erfindung abgeleitet werden können;

[0021] Fig. 6 stellt einen Strukturkompilierer nach einer Ausführung der vorliegenden Erfindung dar;

[0022] Fig. 7 stellt das Beispiel eines geordneten Strukturbaums nach einer Ausführung der vorliegenden Erfindung dar;

[0023] Fig. 8 stellt ein weiteres Beispiel eines geordneten Strukturbaums nach einer Ausführung der vorliegenden Erfindung dar;

[0024] Fig. 9 veranschaulicht die Beziehung zwischen Datensätzen, die durch ein Testprogramm nach einer Ausführung der vorliegenden Erfindung benötigt werden;

[0025] Fig. 10 stellt eine Wellenformerzeugung nach einer Ausführung der vorliegenden Erfindung dar;

[0026] Fig. 11 veranschaulicht eine Übersicht, die zur zeitlichen Steuerung nach einer Ausführung der vorliegenden Erfindung verwendet wird;

[0027] Fig. 12 veranschaulicht eine weitere, zur zeitlichen Steuerung verwendete Übersicht nach einer Ausführung der vorliegenden Erfindung.

AUSFÜHRLICHE BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGEN

[0028] Die vorliegende Erfindung wird im Allgemeinen hinsichtlich des Testsystems mit offener Architektur beschrieben, wie es in den US-Anmeldungen-Nr. 60/449 622,10/404 002 und 10/403 817 durch den gleichen Rechtsnachfolger offenbart ist. Der Fachmann wird jedoch erkennen, dass Ausführungen des Testprogramm-Entwicklungssystems und Verfahren nach der vorliegenden Erfindung nicht nur auf einen Tester mit offener Architektur sondern außerdem auch auf einen Tester mit festen Architekturen anwendbar sind.

[0029] Eine Beschreibung des Testsystems mit offener Architektur kann in der US-Anmeldung Nr. 10/772 327 "Verfahren und Gerät zum Testen von Schaltkreisen" gefunden werden, die gleichzeitig hiermit eingereicht wird und die Anspruch auf den Nutzen der US-Anmeldung Nr. 60/449 622 durch den gleichen Rechtsnachfolger erhebt.

[0030] Fig. 1 veranschaulicht die verallgemeinerte Architektur eines herkömmlichen Testers, die zeigt, wie ein Signal erzeugt und auf ein Prüfobjekt (DUT) angewandt wird. Jeder DUT-Eingangsgin ist mit einem Treiber **2** verbunden, der Testdaten anwendet, während jeder DUT-Ausgangspin mit einem Vergleicher **4** verbunden ist. In den meisten Fällen werden Dreifach-Treiber-Vergleicher verwendet, so dass jeder Tester-Pin (Kanal) entweder als ein Eingangspin oder als ein Ausgangspin wirksam sein kann. Die einem einzelnen DUT zugeordneten Tester-Pins bilden zusammen einen Messplatz, der mit einem verbundenen Zeitgebergenerator **6**, Wellenformgenerator **8**, Musterspeicher **10**, Zeitsteuerungsdatenspeicher **12**, Wellenformenspeicherdaten **14** und Block **16**, die die Datenrate definieren, arbeitet.

[0031] Fig. 2 veranschaulicht eine Systemarchitektur **100** nach einer Ausführung der vorliegenden Erfindung. Die Systemsteuereinheit (SysC) **102** wird an mehrere Site-Controller (SiteCs) **104** gekoppelt. Die Systemsteuereinheit kann außerdem an ein Netzwerk gekoppelt werden, um auf Datensätze zuzugreifen. Durch einen Modulverbindungs-Enabler **106** wird jeder Site-Controller gekoppelt, um ein oder mehrere am Messplatz **110** befindliche Testmodule **108** zu kontrollieren. Der Modulverbindungs-Enabler **106** lässt Umstrukturierung von angeschlossenen Hardwaremodulen **108** zu und dient außerdem als Signalleitung zum Datentransfer (zum Laden von Musterdaten, zum Ansammeln von Reaktionsdaten, zum Bereitstellen einer Steuerung, usw.). Mögliche Hardware-Ausführungen umfassen Festverbindungen, Schalteranschlüsse, Signalleitungsanschlüsse, Anrufverbindungen und Sternschaltungen. Der Modulverbindungs-Enabler **106** kann zum Beispiel durch eine

Switchmatrix implementiert werden. Jeder Messplatz **110** wird einem DUT **112** zugeordnet, der durch eine Ladungsplatine **114** mit den Modulen des entsprechenden Platzes verbunden ist. In einer Ausführung kann ein einziger Site-Controller mit mehreren DUT-Plätzen verbunden werden.

[0032] Die Systemsteuereinheit **102** dient als Gesamtsystem-Manager. Er koordiniert die Aktivitäten des Site-Controllers, leitet der Systemebene parallele Teststrategien und sieht zusätzlich Kontrollen von Handhabungsprogrammen/Prüfsonden sowie eine Systemebenen-Datenerfassung und Unterstützung bei Fehlerbearbeitung vor. In Abhängigkeit von der Funktionseinstellung kann die Systemsteuereinheit **102** an einer CPU eingesetzt werden, die getrennt von der Funktion der Site-Controllers **104** ist. Alternativ dazu können sich die Systemsteuereinheit **102** und die Site-Controller **104** eine gemeinsame CPU teilen. Ähnlich kann jeder Site-Controller **104** an seiner eigenen zugeordneten CPU (zentrale Verarbeitungseinheit) oder als ein getrennter Prozess oder Gruppe kleiner Programmbausteine innerhalb der gleichen CPU eingesetzt werden.

[0033] Die System-Architektur kann man sich konzeptionell als das in Fig. 2 dargestellte verteilte System mit dem Verständnis vorstellen, dass die einzelnen Systemkomponenten auch als logische Komponenten eines integrierten monolithischen Systems und nicht zwangsläufig als physikalische Komponenten eines verteilten Systems betrachtet werden könnten.

[0034] Fig. 3 veranschaulicht eine Software-Architektur **200** nach einer Ausführung der vorliegenden Erfindung. Die Software-Architektur **200** stellt ein verteiltes Rechnerbetriebssystem dar mit Elementen für die Systemsteuereinheit **220**, mindestens einem Site-Controller **240** und mindestens einem Modul **260** in Übereinstimmung mit zugeordneten Elementen **102**, **104**, **108** des Hardwaresystems. Zusätzlich zu dem Modul **260** umfasst die Architektur **200** ein entsprechendes Element zur Modulemulation **280** in software.

[0035] Als eine beispielhafte Wahl kann die Entwicklungsumgebung für diese Plattform auf Microsoft Windows basiert werden. Die Verwendung dieser Architektur besitzt Nebennutzen bei Programm und Support-Übertragbarkeit von Unterstützung (z. B. könnte ein Kundendienstingenieur einen Laptop anschließen, der auf dem Rechnerbetriebssystem des Testers läuft, um weiterentwickelte Diagnose durchzuführen). Jedoch kann für große rechenintensive Operationen (als wenn Testmuster kompiliert), die relevante Software als eine unabhängige Entität hergestellt werden, die in der Lage ist, unabhängig zu laufen, um eine Jobdisponierung über verteilten Plattformen zuzulassen. So sind zugeordnete Softwaretools für Stapeljobs in der Lage, auf mehreren Plattformtypen zu laufen.

[0036] Als eine beispielhafte Wahl kann der ANSI/ISO-Standard C++ als Muttersprache für die Software genommen werden. Natürlich gibt es eine große Zahl von verfügbaren Optionen (zum Bereitstellen einer Schicht über den nominellen C++-Schnittstellen), die es einem dritten Teilnehmer erlauben, sich mit einer alternativen Sprache seiner eigenen Wahl in das System zu integrieren.

[0037] Fig. 3 veranschaulicht eine Vignettierung von Elementen entsprechend ihrer Organisierung durch Nennwertquelle (oder kollektive Entwicklung als ein Subsystem) einschließlich des Rechenbetriebssystems des Testers, Anwenderkomponenten **292** (z. B. durch einen Anwender für Testzwecke geliefert), Systemkomponenten **294** (z. B. geliefert als Softwareinfrastruktur für grundlegende Vernetzungsfähigkeit und Kommunikation), Modulentwicklungskomponenten **296** (z. B. von einem Modulentwickler geliefert) und externe Komponenten **298** (z. B. durch externe Ressourcen außer Modulentwickler geliefert).

[0038] Aus der Perspektive einer quellenbasierten Organisierung umfasst die Rechenbetriebssystem-Schnittstelle **290** des Testers (TOS): Systemsteuereinheit für Site-Controller-Schnittstellen **222**, Rahmenklassen **224**, Site-Controller für Modulschnittstellen **245**, Rahmenklassen **246**, Schnittstellen mit vorgegebenem Modulpegel, Backplane-Nachrichtenbibliothek **249**, Grundplatten-Slot IF (Schnittstelle) **262**, Lademodul-Hardware-Schnittstelle **264**, Backplane-Simulationsschnittstelle **283**, Lademodul-Simulationsschnittstelle **285**, DUT-Simulationsschnittstelle **287**, Verilog PLI (Programmiersprachen-Schnittstelle) **288** für Verilog-Modell des DUT und C/C++-Sprachenunterstützung **289** für C/C++-Modell des DUT.

[0039] Anwenderkomponenten **292** umfassen: einen Anwender-Testplan **242**, Anwender-Testklassen **243**, Hardware-Lademodul **265** und DUT **266**, ein DUT-Verilogmodell **293** und ein DUT-C/C++-Modell **291**.

[0040] Systemkomponenten **294** umfassen: Systemtools **226**, Nachrichtenbibliothek **230**, Testklassen **244**, einen Backplane-Treiber **250**, HW-Backplane **261**, Simulationsrahmen **281**, Backplane-Emulation **282** und Lademodulsimulation **286**.

[0041] Modulentwicklungskomponenten **296** umfassen: Modulbefehlsausführung **248**, Modul-Hardware **263** und Modulemulation **284**.

[0042] Externe Komponenten **298** enthalten externe Tools **225**.

[0043] Die Systemsteuereinheit **220** umfasst Schnittstellen **222** für Site-Controller, Rahmenklassen **224**, System-Tools **226**, externe Tools **225** und eine Nachrichtenbibliothek **230**. Die Software der Systemsteuereinheit ist der primäre Interaktionspunkt für den Anwender. Sie stellt den Netzkoppler für die Site-Controller der Erfindung und Synchronisation der Site-Controller in einer Mehrstellen-DUT-Umgebung bereit, wie es durch den gleichen Rechtsnachfolger in der US-Anmeldung Nr. 60/449 622 beschrieben ist. Auf der Systemsteuereinheit laufen Anwendungen und Tools für Anwender, eine grafische Anwenderschnittstelle(GUI)-basiert oder anders. Die Systemsteuereinheit kann auch als der Verwahrungsort für alle auf den Testplan bezogenen Informationen einschließlich Datensätze für Testpläne, Testmuster und Testparameter wirksam sein. Der diese Datensätze speichernde Speicher kann für die Systemsteuereinheit lokal oder offline sein, z. B. durch ein Netzwerk mit der Systemsteuereinheit verbunden sein. Ein Testparameterdatensatz enthält Parametrierungsdaten für eine Testklasse in der objektorientierten Umgebung einer Ausführung der Erfindung.

[0044] Dritte Entwickler können Tools zusätzlich zu den normalen Systemtools **226** (oder als Ersatz dazu) bereitstellen. Die normalen Schnittstellen **222** an der Systemsteuereinheit **220** enthalten Schnittstellen, die die Tools nutzen, um auf den Tester und Testobjekte zuzugreifen. Die Tools (Anwendungen) **225**, **226** ermöglichen interaktive und Gruppenfolgesteuerung von Tester und Testerobjekten. Die Tools umfassen Anwendungen zur Bereitstellung von Automatisierungsfähigkeiten (zum Beispiel durch die Verwendung SECS/TSEM, usw.).

[0045] Die in der Systemsteuereinheit **220** liegende Nachrichtenbibliothek **230** bewirkt den Mechanismus zum Kommunizieren mit dem Site-Controller **240** in einer für Nutzeranwendungen und Testprogramme transparenten Weise.

[0046] Die Schnittstellen **222**, die in dem Speicher resident und der Systemsteuereinheit **220** zugeordnet sind, bewirken offene Schnittstellen für die Rahmenobjekte, die auf der Systemsteuereinheit ausführen. Enthalten sind Schnittstellen, die der dem Site-Controller zugrunde liegenden Modulsoftware es erlauben, auf Strukturdaten zuzugreifen und diese wieder zu finden. Außerdem enthalten sind Schnittstellen, die Anwendungen und Tools nutzen, um auf Tester und Testobjekte zuzugreifen, sowie Skript-Schnittstellen, die die Möglichkeit zum Zugreifen und Manipulieren von Tester und Testkomponenten durch eine Scriptmaschine bewirken. Dies erlaubt einen gemeinsamen Mechanismus für interaktive Stapel- und Fernanwendungen, um ihre Funktionen auszuführen.

[0047] Die der Systemsteuereinheit **220** zugeordneten Rahmenklassen **224** bewirken einen Mechanismus zur Interaktion mit diesen oben erwähnten Objekten, indem eine Referenzausführung der normalen Schnittstelle zur Verfügung gestellt wird. Zum Beispiel erzeugt der Site-Controller **240** der Erfindung ein funktionales Testobjekt. Die Rahmenklassen der Systemsteuereinheit können eine entsprechende funktionale Testvollmacht als Ersatz auf Basis einer fernen Systemsteuereinheit des funktionalen Testobjekts bereitstellen. So wird die normale funktionale Testschnittstelle den Tools auf der Systemsteuereinheit **220** verfügbar gemacht. Die Rahmenklassen bewirken ein mit der Host-Systemsteuereinheit effektiv verknüpftes Rechnerbetriebssystem. Sie bilden außerdem die Softwareelemente, die den Netzkoppler für die Site-Controller bewirken und stellen eine Synchronisierung der Site-Controller in einer Mehrstellen/DUT-Umgebung bereit. Diese Schicht bewirkt so in einer Ausführung der Erfindung ein Objektmodell, das zum Manipulieren und Zugreifen der Site-Controller geeignet ist, ohne sich direkt mit den Nachrichtenschichten beschäftigen zu müssen.

[0048] Der Site-Controller **240** richtet einen Anwendertestplan **242**, Anwender-Testklassen **243**, Standardtestklassen **244**, Standardschnittstellen **245**, Site-Controller-Rahmenklassen **246**, Schnittstellen für Module mit H-Pegel-Befehlen (d. h. Schnittstellen **247** mit vorgegebenem Modulpegel), Ausführung **248** von Modulbefehlen, Backplane-Nachrichtenbibliothek **249** und einen Backplane-Treiber **250** aus. Vorzugsweise wird die Testfunktionalität meistens durch die Site-Controller **104**/**240** gehandhabt, was somit eine unabhängige Arbeitsweise der Messplätze **110** ermöglicht.

[0049] Ein Testplan **242** wird vom Anwender geschrieben. Der Plan kann direkt in einer, objektorientierte Konstrukte nutzenden, gültigen Computersprache wie C++ geschrieben werden, oder in einer Testprogrammiersprachen höheren Niveaus beschrieben werden, um einen C++-Code zu erzeugen, der anschließend in das ausführbare Testprogramm kompiliert werden kann. Zur Entwicklung von Testprogrammen nutzt eine Ausführung der Erfindung einen erfinderischen Testprogrammsprachen-Kompilierer (TPL) des Rechtsnachfolgers.

Mit Bezug auf Fig. 4 wirkt der Testprogramm-Kompilierer **400** in einem Teil als ein Codegenerator einschließlich eines Übersetzerprofils **402** zum Übersetzen von Quellenfiles **404** des Testprogrammentwicklers, die Tests und zugeordnete Parameter in objektorientierte Konstrukte, wie beispielsweise C++ Code, beschreiben. Ein Kompiliererprofil **406** kompiliert und verbindet seinerseits den Code zu ladefähigen Dateien, z. B. Dateien für Betriebssystemroutinen, um das Testprogramm zu erzeugen, welches von dem Testersystem ausgeführt werden kann. Obwohl die Anwendung des TPL-Code-Generators/Übersetzers auf Testsysteme neuartig ist, ist bitte davon Kenntnis zu nehmen, dass Codegeneratoren an sich bekannt sind. Auch das Kompiliererprofil kann ein an sich bekannter normaler C++ Kompilierer sein.

[0050] Der Testplan erzeugt Testobjekte durch Nutzung der Rahmenklassen **246** und/oder normaler oder dem Anwender gelieferter Testklassen **244**, die mit den Site-Controllern verknüpft sind, konfiguriert die Hardware unter Verwendung der Standardschnittstellen **245** und definiert den Testplanablauf. Er stellt außerdem jede zusätzliche Logik bereit, die während einer Ausführung des Testplans benötigt wird. Der Testplan unterstützt einige Basisdienste und erzeugt eine Schnittstelle zu den Diensten von zugrunde liegenden Objekten wie Fehlersuchdienste (z. B. Zwischenstoppen) und den Zugriff auf zugrunde liegende Rahmen- und Standardklassen.

[0051] Die Eingabe des Quellencodes in den Testprogramm-Kompilierer **400** umfasst einen Testplan-Beschreibungsdatensatz, der die in einem Testplan verwendeten Objekte und ihre Beziehungen zueinander festlegt. Dieser Datensatz wird in einen C++ Code übersetzt, der auf dem Site-Controller in Form einer Implementierung einer Standardschnittstelle ausgeführt wird, der ITestPlan bezeichnet werden kann. Dieser Code wird in eine Datei für Betriebssystemroutinen von Windows (DLL) gepackt, die in den Site-Controller geladen werden kann. Es wird die Testprogramm-DLL erzeugt, um normale bekannte Einsprungstellen zu erhalten, die die Site-Controller-Software verwenden kann, um das Testplanobjekt, das sie enthält, zu erzeugen und zurück zu setzen. Die Site-Controller-Software lädt die Testprogramm-DLL in ihren Prozessraum und nutzt eine der Einsprungstellen, um ein Beispiel des Testplanobjekts zu erzeugen. Sobald das Testplanobjekt erzeugt worden ist, kann die Site-Controller-Software anschließend den Testplan ausführen.

[0052] Die mit den Site-Controllern verknüpften Rahmenklassen **246** sind ein Satz von Klassen und Verfahren, die gemeinsame testbezogene Operationen durchführen. Der Rahmen auf der Ebene des Site-Controllers enthält zum Beispiel Klassen für Stromversorgung und Fortschaltung von Pin-Elektronik, Setzen von Niveaus und Bedingungen der zeitlichen Zuordnung, Erlangung von Messungen und das Steuern von Testabläufen. Der Rahmen umfasst auch Verfahren für Laufzeitdienste und zur Fehlerbeseitigung. Die Rahmenobjekte können durch Ausführung der Standardschnittstellen arbeiten. Zum Beispiel ist die Ausführung der Rahmenklasse Tester-Pin standardisiert, um eine allgemeine Tester-Pin-Schnittstelle zu implementieren, die Testklassen verwenden können, um mit Hardware-Modulpins in Wechselwirkung zu treten.

[0053] Bestimmte Rahmenobjekte können implementiert werden, so dass sie mit Hilfe der Modulebenen-Schnittstellen **247** arbeiten, um mit den Modulen zu kommunizieren. Die Rahmenklassen der Site-Controller funktionieren effektiv wie ein lokales Rechnerbetriebssystem, das jeden Site-Controller unterstützt.

[0054] Im Allgemeinen sind mehr als 90% des Programmcodes Daten für den Bausteintest, und die verbleibenden 10% des Codes realisieren die Testmethodik. Die Daten des Bausteintests sind vom Prüfobjekt (DUT) (z. B. Stromversorgungsbedingungen, Signalspannungsbedingungen, Zeitsteuerungsbedingungen, usw.) abhängig. Der Testcode besteht aus Verfahren zum Laden der speziellen Bausteinbedingungen in ATE-Hardware und außerdem denjenigen, die zum Realisieren von anwenderspezifischen Aufgaben (wie beispielsweise Datenerfassung) benötigt werden. Der Rahmen nach einer Ausführung der Erfindung stellt einen von Hardware unabhängigen Test und ein Tester-Objektmodell bereit, das es dem Anwender ermöglicht, die Aufgabe einer DUT-Testprogrammierung durchzuführen.

[0055] Um die Wiederverwendbarkeit eines Testcodes zu erhöhen, kann ein solcher Code unabhängig von allen bausteinspezifischen Daten (z. B. Pinname, Auslöseimpulsdaten, usw.) oder von für den Bausteintest spezifischen Daten (z. B. Bedingungen für Gleichstromeinheiten, Messkontakte, Anzahl von Zielkontakten, Name des Strukturdatenfiles, Adressen von Strukturprogrammen) gemacht werden. Wenn ein Code für einen Test mit Daten dieser Typen kompiliert wird, würde die Wiederverwendbarkeit des Testcodes abnehmen. Deshalb können gemäß einer Ausführung der Erfindung beliebige, für den Baustein spezifische Daten oder für den Bausteintest spezifische Daten dem Testcode als Eingaben während einer Codeausführungszeit extern verfügbar gemacht werden.

[0056] In einer Ausführung der Erfindung realisiert eine Testklasse, die eine Ausführung einer hier als ITest bezeichneten normalen Testschnittstelle ist, die Trennung von Testdaten und Code (und damit die Code-Wie-

derverwendbarkeit) für einen speziellen Testtyp. Eine solche Testklasse kann als „Dokumentvorlage" für getrennte Fälle von sich selbst betrachtet werden, die nur auf der Basis von für den Baustein spezifischen und/oder für den Bausteintest spezifischen Daten voneinander abweichen. Die Klassen werden in dem Testplan-Datensatz spezifiziert. Jede Testklasse führt typischerweise einen speziellen Typ eines Bausteintests oder Setup zum Bausteintest aus. Zum Beispiel kann eine Ausführung der Erfindung eine spezielle Ausführung der ITest-Schnittstelle, z. B. FunctionalTest, als Basisklasse für alle Betriebsprüfungen für DUTs bereitstellen. Sie bewirkt die grundlegende Funktionalität zum Einstellen von Testbedingungen, Ausführen von Mustern und Bestimmen des Status des Testes basierend auf dem Vorhandensein von missglückten Impulsen. Andere Arten von Ausführungen können AC- und DC-Testklassen umfassen, die hier als ACParametricTests und DCParametricTests bezeichnet sind.

[0057] Alle Testtypen können Standardausführungen von einigen virtuellen Verfahren (z. B. init(), preExec() und postExec()) bereitstellen. Diese Verfahren werden die Einsprungstellen des Prüfingenieurs zum Übersteuern von standardmäßigem Verhalten und zum Einstellen von beliebigen testspezifischen Parametern. Jedoch können in den Testplänen auch kundenspezifische Testklassen verwendet werden.

[0058] Testklassen erlauben es dem Anwender, Klassenverhalten durch Bereitstellung von Parametern zu konfigurieren, die verwendet werden, um die Optionen für einen speziellen Fall dieses Tests zu spezifizieren. Zum Beispiel kann eine Betriebsprüfung die Parameter PList und TestConditions hernehmen, um die Strukturliste zum Ausführen und die Bedingungen von Ebenen bzw. Taktungen für den Test zu spezifizieren. Die Festlegung unterschiedlicher Werte für diese Parameter (durch Verwendung von unterschiedlichen "Testblocks" in einer Testplan-Beschreibungsdatei) ermöglicht es dem Anwender, unterschiedliche Fälle einer Betriebsprüfung zu erzeugen. **Fig. 5** veranschaulicht, wie unterschiedliche Testfälle aus einer einzigen Testklasse abgeleitet werden können. Diese Klassen können direkt in objektorientierten Konstrukten, wie beispielsweise C++-Code, programmiert werden oder ausgelegt werden, um es einem Testprogramm-Kompilierer zu ermöglichen, die Beschreibung der Tests und ihrer Parameter aus einem Testplan-Datensatz zu entnehmen und einen entsprechenden C++-Code zu erzeugen, der kompiliert und verknüpft werden kann, um das Testprogramm zu generieren. Eine Dokumentvorlagen-Bibliothek kann als die Universalbibliothek von den generischen Algorithmen und Datenstrukturen genutzt werden. Diese Bibliothek kann einem Anwender des Testers sichtbar gemacht werden, so dass der Anwender zum Beispiel die Ausführung einer Testklasse modifizieren kann, um eine anwenderdefinierte Testklasse zu erzeugen.

[0059] Hinsichtlich der für Anwender entwickelten Testklassen unterstützt eine Ausführung des Systems die Integration solcher Testklassen in den Rahmen dadurch, dass sich alle Testklassen von einer einzelnen Testschnittstelle, z. B. ITest, ableiten, so dass der Rahmen sie in der gleichen Weise wie den Standardsatz von System-Testklassen manipulieren kann. Anwendern steht es frei, eine zusätzliche Funktionalität in ihre Testklassen mit dem Verständnis einzubeziehen, dass sie in ihren Testprogrammen einen kundenspezifischen Code nutzen müssen, um Vorteil aus diesen zusätzlichen Systemeinrichtungen zu ziehen.

[0060] Jeder Messplatz **110** ist dem Testen eines oder mehrerer DUTs **106** zugeordnet und funktioniert durch eine konfigurierbare Sammlung von Testmodulen **112**. Jedes Testmodul **112** ist eine Einheit, die eine bestimmte Testaufgabe ausführt. Zum Beispiel könnte das Testmodul **112** eine Stromversorgung des DUT, eine Pinkarte eine Analogkarte, usw. sein. Diese Modullösung bewirkt einen hohen Grad an Flexibilität und Konfigurierbarkeit.

[0061] Die Ausführungsklassen **248** für Modulbefehle können von Modulhardware-Herstellern zur Verfügung gestellt werden und entweder Modulebenen-Schnittstellen für Hardwaremodule implementieren oder modulspezifische Ausführungen von Standardschnittstellen in Abhängigkeit von dem durch einen Hersteller ausgewählten Ausführungsverfahren für Befehle bereitstellen. Die externen Schnittstellen dieser Klassen werden durch vorgegebene Schnittstellenanforderungen für Modulebenen und Anforderungen an Backplane-Nachrichtenbibliotheken definiert. Diese Schicht sorgt außerdem für eine Erweiterung der Standardgröße von Testbefehlen, was das Hinzufügen von Verfahren (Funktionen) und Datenelementen ermöglicht.

[0062] Die Backplane-Nachrichtenbibliothek **249** erzeugt die Schnittstelle für standardmäßige Kommunikationen über die Backplane, wodurch die Funktionen bereitgestellt werden, die zum Kommunizieren mit den am Messplatz angeschlossenen Modulen notwendig sind. Dies erlaubt es, dass herstellerspezifische Modulsoftware einen Backplane-Treiber **250** zum Kommunizieren mit den entsprechenden Hardwaremodulen verwendet. Das Backplane-Nachrichtenprotokoll kann ein paketbasiertes Format nutzen.

[0063] Objekte des Tester-Pins stellen physikalische Tester-Kanäle dar und leiten sich von einer Tes-

ter-Pin-Schnittstelle ab, die hier als ITesterPin bezeichnet ist. Das Software-Entwicklungspaket (SDK) nach einer Ausführung der Erfindung bewirkt eine Standardausführung von ITesterPin, die TesterPin genannt werden kann, die in Form eines I-Kanals mit vorgegebener Modulebenen-Schnittstelle ausgeführt ist. Herstellern steht es frei, von TesterPin Gebrauch zu machen, wenn sie ihre Modulfunktionalität in Form von I-Kanal ausführen können, sonst müssen sie für eine Ausführung von ITesterPin sorgen, um mit ihrem Modul zu arbeiten.

[0064] Die hier als IModul bezeichnete Modulstandardschnittstelle, die von dem Tester-System der Erfindung bereitgestellt wird, stellt generisch das Hardwaremodul eines Herstellers dar. Die von Herstellern gelieferte modulspezifische Software für das System kann in Form von ladefähigen Dateien, wie beispielsweise Dateien für Betriebssystemroutinen (DLL), zur Verfügung gestellt werden. In einer einzelnen DLL kann Software von einem Hersteller für jeden Modultyp eingeschlossen sein. Jedes derartige Softwaremodul ist verantwortlich zur Bereitstellung von herstellerspezifischen Ausführungen für die Befehle der Modulschnittstellen, die die API (Anwendungsprogrammierschnittstelle) für Modulsoftware-Entwicklung umfassen.

[0065] Es gibt zwei Ausführungen der Modulschnittstellen-Befehle: Erstens dienen sie als Schnittstelle für Anwender zum Kommunizieren (indirekt) mit einem speziellen Hardwaremodul in dem System und zweitens stellen sie die Schnittstellen zur Verfügung, von denen dritte Entwickler Nutzen ziehen können, um ihre eigenen Module in den Site-Controller-Ebenenrahmen zu integrieren. So werden die durch den Rahmen zur Verfügung gestellten Modulschnittstellen-Befehle in zwei Typen eingeteilt:
Die ersten und deutlichsten sind diejenigen "Befehle", die dem Anwender durch die Rahmenschnittstellen gezeigt werden. So bewirkt eine Tester-Pin-Schnittstelle (ITesterPin) Verfahren, um Ebenen- und Taktungswerte zu bekommen und einzusetzen, während eine Stromversorgungs-Schnittstelle (IPowerSupply) Verfahren zum Beispiel zum Netzeinschalten und Netzausschalten bewirkt.

[0066] Außerdem stellt der Rahmen die spezielle Kategorie der vorgegebenen Modulebenen-Schnittstellen zur Verfügung, die zum Kommunizieren mit den Modulen genutzt werden kann. Diese sind die durch Rahmenklassen verwendeten Schnittstellen (d. h. „standardmäßige" Ausführungen von Rahmenschnittstellen) zum Kommunizieren mit Herstellermodulen.

[0067] Die Nutzung der zweiten Ausführung, die Modulebenen-Schnittstellen, ist jedoch optional. Der Vorteil, so vorzugehen besteht darin, dass Hersteller dann aus den Ausführungen der Klassen wie ITester-Pin und IPowerSupply, usw. Nutzen ziehen können, während sich auf den Inhalt spezifischer Meldungen konzentriert wird, die durch Implementierung der Modulebenen-Schnittstellen an ihre Hardware gesendet werden. Wenn diese Schnittstellen für den Hersteller ungeeignet sind, können sie jedoch wählen, um ihre kundenspezifischen Ausführungen der Rahmenschnittstellen (z. B. Hersteller-Ausführungen von ITester-Pin, IPower-Supply, usw.) zur Verfügung zu stellen. Diese würden dann die kundenspezifische Funktionalität bewirken, die für ihre Hardware angemessen ist.

[0068] Mit dieser offenen Architektur als Hintergrund wird das Testprogramm-Entwicklungssystem der vorliegenden Erfindung weiter wie folgt beschrieben. Abschnitt A beschreibt Regeln zum Beschreiben der Testumgebung, in der ein Testprogramm verwendet wird; Abschnitt B beschreibt das Verfahren und Regeln zur Testprogramm-Entwicklung; Abschnitt C spezifiziert das Verfahren und Regeln zum Entwickeln eines Testprogramms und wie die Hauptstruktur des Testprogramms zu definieren ist; Abschnitt D beschreibt, wie ein Testprogramm auf einem Testsystem offener Architektur abzuarbeiten ist; Abschnitt E beschreibt ein Verfahren und Regeln für Testmuster; Abschnitt F beschreibt Regeln zum Beschreiben der zeitlichen Steuerung der Testmuster; und Abschnitt G beschreibt Regeln für die gesamte Arbeitsweise des Prüfgerätes.

A. Komponenten

[0069] Die Testumgebung umfasst einen Satz von Dateien, welche die notwendigen Bedingungen zum Hochfahren des Prüfgerätes im Einzelnen festlegen und um es vorzubereiten, damit eine Menge von Tests abgearbeitet wird. Die Testumgebung umfasst vorzugsweise Datensätze für:

1. Tester-Ressourcendefinition: zur Spezifizierung der Typen von Testerbauelementen und unterstützten Parametern für solche Bauelemente, die in dem Testsystem mit offener Architektur verfügbar sind.
2. Tester-Konfiguration: zur Spezifizierung von Site-Controllers, Standorte und entsprechenden Abbildungen.
3. Modul-Konfiguration: zur Spezifizierung des Hardwaremoduls in jedes Standorts.
4. Pin-Beschreibungen: zur Benennung von Prüfobjekt-Pins wie beispielsweise Signal-Pins, Stromversorgungen und zum Beschreiben von Pin-Gruppen.
5. Socket: zur Spezifizierung von Zuweisungen DUT-Pin-zu-Tester-Pin.

6. Pin-Optionen: zur Spezifizierung von speziellen Optionen oder Betriebsarten für Pins.
7. Strukturlisten: zur Spezifizierung von Testmustern und ihrer Abfolge.
8. Strukturen: zur Spezifizierung von Testvektoren.

[0070] Aus dem Oben genannten werden durch ICF (Installations- und Konfigurationsdateien) mit Informationen aus einer CMD (Konfigurationsmanagement-Datenbank) die Entitäten 1 bis 3 erzeugt und an einer bekannten Stelle verfügbar gemacht, während die Entitäten 4 bis 8 anwenderspezifisch sind. Dieser Abschnitt liefert Beschreibungen für die oben genannten Entitäten 1 bis 6, wobei die Entitäten 7 bis 8 ausführlicher im Abschnitt E beschrieben werden. Vorzugsweise werden spezielle Methoden und Regeln verwendet, um jede dieser Komponenten zu entwickeln. Diese Verfahren und Regeln werden mit Beispielen in diesem Abschnitt beschrieben.

A1. Die Ressourcendefinition

[0071] Jedes Hardwaremodul stellt einen oder mehrere Typen von Hardwareressourcen (der Kürze halber Ressourcen) zur Nutzung durch das Testsystem bereit. Die Ressourcendefinition des Prüfgerätes wird vorzugsweise verwendet, um einen Satz von Ressourcennamen für die verfügbaren Ressourcentypen und einen Satz von Parameternamen und -typen, die jedem speziellen Ressourcentyp zugeordnet sind, anzugeben. Zum Beispiel wird der Ressourcenname dpin genutzt, um auf digitale Prüfgerät-Pins zu verweisen. Diese Ressourcen besitzen Parameter wie beispielsweise VIL (für die eingegebene Niederspannung), VIH (für die eingegebene Hochspannung), VOL (für die ausgegebene Niederspannung), VOH (für die ausgegebene Hochspannung), usw.. Eine Ressourcendefinitionsdatei wird die Erweiterung ".rsc" aufweisen. Nachstehend gezeigt ist eine beispielhafte Ressourcendefinition, die einige Prüfgerätressourcen enthält:

```
#
#File Resources.rsc
#
Version 0.1.2;
ResourceDefs
{
   #Digitalpins
    dpin
   {
     # Niedrige und hohe Spannungen für Eingangs-Pins
     Spannung VIL, VIH;
     # Niedrige und hohe Spannungen für Ausgangs-Pins
     Spannung VOL, VOH;
}
# Stromversorgungen
dps
```

```
{
#
# PRE_WAIT bestimmt die Zeit zu warten, nachdem Span-
nung
#          ihren Endwert erreicht hat, um Musterer-
zeugung
#          zu starten. Die tatsächliche Zeit, die das
System
#          warten wird, ist ein kleiner systemspezi-
fischer
#          Bereich:
#          PRE_WAIT-delta <= actual <= PRE_WAIT+delta
#
#          PRE_WAIT_MIN ist eine Mindestgröße zu war-
ten,
#          nachdem Spannung ihren Endwert erreicht
hat, um
#          Mustererzeugung zu starten.
#          Es ist ein systemspezifischer Bereich:
#          PRE_WAIT_MIN <= actual <=
PRE_WAIT_MIN+delta
#
# POST_WAIT bestimmt die Zeit zu warten, nachdem
#          Mustererzeugung endet, um den Strom abzu-
schalten.
#          Die tatsächliche Zeit, die das System war-
ten wird,
#          ist ein kleiner systemdefinierter Bereich:
  #          POST_WAIT-delta <= actual <=
POST_WAIT+delta
  #
  # POST_WAIT_MIN bestimmt die Zeit zu warten, nach-
dem
  #          Mustererzeugung endet, um den Strom abzu-
schalten.
```

```
  #-       Die tatsächliche Zeit, die das System war-
ten wird,
  #        ist ein kleiner systemdefinierter Bereich:
  # POST_WAIT_MIN <= actual <=
POST_WAIT_MIN+delta
  #
  Zeit PRE_WAIT,
  Zeit PRE_WAIT_MIN,
  Zeit POST_WAIT,
  Zeit POST_WAIT_MIN,
  # Die Spannung.
  Spannung VCC;
  }
}
```

[0072] Anzumerken ist, dass der Typ eines Ressourcenparameters (wie Spannung oder Zeit) vorzugsweise eine normale technische Maßeinheit ist. Hersteller, die Betriebsmittel für spezielle Zwecke liefern, die die Spezifikation unterschiedlicher Parameter bevorzugen, sollten ihre eigenen Dateien für Ressourcendefinition erzeugen.

Struktur für die Ressourcendefinition

[0073] Nachstehend ist eine Struktur für die Ressourcendefinition gemäß einer bevorzugten Ausführung der vorliegenden Erfindung gegeben:

```
    resource-file:
       version-info resource-defs
    version-info:
       Version version-identifier ;
    resource-defs:
       ResourceDefs {resource-def-list}
resource-def-list:
       resource-def
       resource-def-list resource-def
```

```
resource-def:
        resource-name {resource-params-
decl-list}

resource-params-decl-list:
    resource-params-decl
    resource-params-decl-list resource-
params-decl

resource-params-decl:
    elementary-type-name resource-params-
  decl-list;

resource-params-decl-list:
    resource-param-name
    resource-param-name-list, resource-
  param-name
```

[0074] Oben erwähnte, unbestimmte Nicht-Eingänge sind nachstehend festgelegt:

1. version-identifier (Versionskennzeichnung): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z]. Sie stellt eine Versionsnummer dar.
2. resource-name (Quellenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen einer Quelle wie beispielsweise dpin oder dps dar.
3. elementarg-type-name (Elementarer Typenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen eines grundlegenden Typs wie beispielsweise Spannung dar (vgl.).
4. resource-param-name (Quellenparametername): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen eines Ressourcenparameters wie beispielsweise VIL dar.

A2. Tester-Konfiguration

[0075] Die Tester-Konfiguration ist ein Satz von Regeln, der vorzugsweise genutzt wird, um die Site-Controller in einer speziellen Systemkonfiguration und die Verbindung der Site-Controller mit den Switchmatrix-Eingabeanschlüssen aufzuführen. In der Architektur einer Ausführung der Erfindung kann ein einzelner Site-Controller mit einem einzelnen Switchmatrix-Eingangsport verbunden werden. Folglich dienen in diesem Zusammenhang die Switchmatrix-Verbindungen als implizite Kennzeichnungen für die Site-Controller im System (andere Konfigurationen sind möglich). Das Folgende ist ein Beispiel einer typischen Tester-Konfiguration:

```
#
# Tester-Konfiguration, Sys.cfg
#
Version 1.2.5
SysConfig
{
  #
  # das erste Feld ist der Zentralrech-
nername der
  # Site-Controller-Maschine;
  # er kann entweder als eine durch
Punkte
  # voneinander getrennte dezimale In-
ternet-
  # Protokoll-Adresse oder einen domä-
ne-
  # qualifizierten Zentralrechnernamen
bestimmt
  # werden.
  #
  # Das zweite Feld ist die Zahl des
  # Switchmatrix-Eingangsports, die im-
plizit
  # als die Kennzeichnung für den mit
ihr
  # verbundenen Site-Controller dient.
  #
  zeus.olympus.deities.org      2;
  127.0.0.2                          4;
  127.0.0.0                          1;#
                                     SI-
                                     TEC
                                     -1
  127.0.0.3                          3;
}
```

[0076] Die Systemkonfiguration für ein spezielles Prüfstandsystem ist Teil des Systemprofils und wird als Systemkonfigurationsdatei Sys.cfg verfügbar gemacht. Es ist anzumerken, dass in einer Ausführung der mit dem Anschluss 1 („127.0.0.0" in dem oben erwähnten Beispiel) verbundene Site-Controller einen besonderen Status besitzen kann, in dem er allein die Switchmatrix konfiguriert. Dieser „besondere" Site-Controller wird als SITEC-1 bezeichnet. Es ist außerdem anzumerken, dass die Site-Controller-Adresse in diesem Beispiel eine IP Adresse ist, weil die Site-Controller durch ein internes Netzwerk mit der Systemsteuereinheit verbunden sein können. Umgekehrt kann die Systemsteuereinheit mit einem externen Netzwerk verbunden werden, um auf

Dateien wie beispielsweise Strukturdaten zuzugreifen.

Struktur für die Tester-Konfiguration

[0077] Nachstehend ist eine Struktur für die Systemkonfigurationsdatei entsprechend einer Ausführung der vorliegenden Erfindung gegeben:

```
    system-config-file:
        version-info-system-config

    version-info:
        Version version-identifier;

    system-config:
        SysConfig {site-controller-connection-list}

  site-controller-connection-list:
    site-controller-connection
    site-controller-connection-list site-controller-
connection

  site-controller-connection:
       site-controller-hostname input-port;

  site-controller-hostname:
      ip-address
      domain-qualified-hostname

  ip-address:
      octet.octet.octet.octet

  domain-qualified-hostname:
     name
     domain-qualified-hostname.name
```

[0078] Oben erwähnte, unbestimmte Nicht-Anschlüsse sind nachstehend festgelegt:

1. version-identifier: Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z]. Sie stellt eine Versionsnummer dar.
2. octet (Achtbit-Zeichen): Eine nicht negative, ganze Zahl von 0 bis 255 (in Dezimaldarstellung).
3. name: Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt ein Namenssegment in einem domainqualifizierten Zentralrechnernamen dar.
5. input-port (Eingangsport): Eine nicht negative, ganze Zahl in Dezimaldarstellung.

A3. Die Modulkonfiguration

[0079] Die Modulkonfiguration ermöglicht die Spezifizierung der physikalischen Konfiguration des Testers, z. B. der physikalische Speicherplatz und Typ jedes Moduls in einem SYSTEM-Chassis. Dies wird durch die dynamische Beschaffenheit der Tester-Buskonfiguration verlangt, die eine Zuordnung der Tester-Busadresse zu dem physikalischen Slot-Speicherplatz zulässt. Diese Informationen ermöglichen es, dass ein Hardware-Feststellvorgang, der zum Zeitpunkt des Hochladens des Systems auftritt, die SYSTEM-Konfiguration gültig macht. Jeder Ausgangsport der Switchmatrix definiert einen physikalischen Slot, der vorzugsweise von einem einzelnen Hardwaremodul eingenommen wird. Nachstehend ist ein Beispiel einer Modulkonfiguration gezeigt, die in der Datei Modules.cfg entsprechend einer Ausführung der Erfindung festgelegt ist:

```
#
# Modulkonfigurationsdatei, Modules.cfg
#
Version 0.0.1;
ModulConfig
 {
    #
    # Eine Konfigurationsdefinition, die Infor-
mationen
    # über den Modultyp zur Verfügung stellt,
der mit
    # den Slots 1-12 und 32-48 verknüpft ist.
    # Zu beachten ist, dass ein Modul mehr als
einen
    # einzelnen Ressourcentyp erzeugen kann.
    #
    Slot 1-12, 32-48
                          # Switchmatrix-
Ausgangsports,
                          # die die nachstehend
  definierte
                          # Konfiguration verwen-
  den.
      {
      HerstellerID 1;     # definierter Herstel-
  lercode
      ModulID 1;          # herstellerdefinierter
  ID-Code
      ModulTreiber mod1.dll;     # Modulsoftware

      #
```

```
      # mit dpin benannte Ressource spezifiziert
Kanäle
      # für digitale Daten. Der Name dpin ist kein
      # Schlüsselwort. Er ist einfach der Name ei-
ner
      # Hardware-Ressource und wird aus der
      # Ressourcendefinitionsdatei erhalten.
      #
      Ressource dpin
      {
         MaxVerfügbar 32;    # Ressourceneinheiten
1 .. 32.
      }
      Ressource analog
      {
      MaxVerfügbar 16      # Ressourceneinheiten 1
.. 16.
      arbeitsunfähig  1-8  # arbeitsunfähige Quel-
len 1 .. 8.
                           # somit sind aktivierte
9 .. 16.
     }
   }
   #
   # Konfigurationsdefinition, die Informationen
über den
   # Modultyp liefert, der mit Slots 16-30, 50 und
61-64
   # verknüpft ist.
   Slot 16-30, 50, 61-64
   {
      Ressource dpin
      {
        MaxVerfügbar 32;         # max. verfügbare
                                   Ressourcenein-
heiten
```

```
        arbeitsunfähig 3, 30-32 # arbeitsunfähige
Quellen.
      }
      ModulTreiber    „module two.dll";

      HerstellerID            2;
      ModulID                2;
}
#
# Konfigurationsdefinition, die Informationen über
# den Modultyp liefert, der mit Slots 65-66 ver-
knüpft ist.
#
Slot 65-66
{
      ModulID          4;       #DPS Modul mit 8
Zuleitungen
      ModulTreiber     mod4.dll;
      HerstellerID      1;
      #
      # Ressourcentyp dps, der Ressourceneinheiten
für eine
      # Baustein-Stromversorgung spezifiziert
      #
      Ressource dps
      {
          MaxVerfügbar 4;
          Arbeitsunfähig  1;
      }
    }
  }
```

[0080] Wie vorher erwähnt, bezieht sich in einer Ausführung ein Slot auf eine Steckverbindung, durch die ein Hardwaremodul wie beispielsweise ein Ausgangsport der Switchmatrix angeschlossen werden kann. Jede Konfigurationsdefinition liefert Informationen über das Modul, das mit einem oder mehreren Slots verknüpft werden kann. Die in einer Konfigurationsdefinition festgelegte HerstellerID ist eine einem Hersteller zugeordnete, einmalige ID. Die ModulID bezieht sich auf einen von diesem Hersteller zur Verfügung gestellten Modultyp. In einer Tester-Konfiguration können mehrere Beispiele der gleichen ModulID vorhanden sein. Der Modultreiber bezieht sich auf eine vom Hersteller gelieferte DLL zum Bedienen des Moduls. Schließlich bezieht sich die Ressource auf die durch dieses Modul bedienten Einheiten und erzeugt einen Namen für den Ressourcentyp, wobei der Ressourcenname aus der Ressourcendefinitionsdatei erhalten wird.

[0081] Das oben erwähnte Beispiel beschreibt drei Konfigurationsblöcke in einer Modulkonfigurationsdatei. In der einen Implementierung werden der erste Konfigurationsblock, Slots 1–12 und 32–48 durch einen vom Hersteller 1 hergestellten Modul bedient. Dieser Hersteller stellt das Modul, die Kennzeichnung „1", um auf die-

sen Modultyp zu verweisen, und die Modultreiber-Bibliothek zur Steuerung des Moduls bereit. Dieses Modul kann zwei Typen von Ressourceneinheiten erzeugen, wobei die eine durch den Quellennamen „dpin" mit einer Gesamtzahl von vorzugsweise 32 Ressourceneinheiten (d. h. „Kanälen") bezeichnet wird, von denen alle verfügbar sind, und die andere durch den Ressourcennamen „analog" mit einer Gesamtzahl von 16 Ressourceneinheiten bezeichnet wird, von denen nur 9 bis 16 verfügbar sind. Der zweite und dritte Konfigurationsblock werden in einer der ersten Konfiguration ähnlichen Art und Weise spezifiziert.

[0082] Es ist anzumerken, dass die Einrichtung, die es Kanälen erlaubt, als „arbeitsunfähig" bezeichnet zu werden, die Identifizierung von fehlerhaften Ressourceneinheiten an Modulen ermöglichen soll, die doch sonst funktionsfähig sind. Es ist auch anzumerken, dass ein Konfigurationsblock eine oder mehrere Slot-Kennzeichnungen aufweisen kann. Wenn ein Block mehr als eine einzelne Slot-Kennzeichnung besitzt, dann wird gesagt, dass die identifizierten Slots geklont sind.

[0083] Die Modulkonfigurationsdatei, Modules.cfg, wird als Teil des Systemprofils durch das ICM (Installationskonfigurations-Verwaltungssystem) erzeugt (mit vom Anwender zur Verfügung gestellten prüfstandspezifischen Informationen) und an einem bekannten Speicherplatz verfügbar gemacht. Das ICM ist ein Dienstprogramm, welches für das Testsystem lokal sein kann, z. B. in der Systemsteuereinheit oder irgendwo im Netzwerk, mit dem die Systemsteuereinheit verbunden ist, liegen kann. Das ICM verwaltet die CMD (Konfigurationsverwaltungs-Datenbank) und wird typischerweise bei Änderungen der Hardware für die Systemkonfiguration aktualisiert. Das ICM ermöglicht es dem Anwender, das System, z. B. Site-Controller und Module, zu konfigurieren. Die CMD ist eine Datenbank, welche die Konfigurationen speichert. Für eine tatsächliche Tester-Konfiguration/Operation erzeugt das ICM die Konfigurationsdateien, z. B. Modulkonfiguration und andere Dateien, und kopiert sie und zugeordnete Dateien wie beispielsweise spezielle Modul-Testprogramme auf den Tester.

Struktur für Modulkonfiguration

[0084] Nachstehend ist die Struktur der Modulkonfiguration entsprechend der bevorzugten Ausführung:

file-contents:
version-info module-config-def

version-info:
Version *version-identifier;*

module-config-def:
ModuleConfig {*slot-entry-list*}

slot-entry-list:
slot-entry
slot-entry-list slot entry

slot-entry:
Slot *positive-integer-list* {*slot-info*}

slot-info:
required-config-list

required-config-list:
required-config
required-config-list required-config

required-config:
HerstellerID *id-code;*
ModulID *id-code;*
ModulTreiber *file-name;*
Ressource *resource-name* {*max-spec disabled-spec*$_{opt}$}

max-spec:

```
          MaxVerfügbar positive integer;

     disabled-spec:
          Arbeitsunfähig positive-integer-list;

     positive-integer-list:
          positive-integer-list-entry
          positive-integer-list, positive-integer-
list-entry

     positive-integer-list-entry:
          positive-integer
          positive-integer-number-range

     positive-integer-number-range:
         positive-integer-pos integer
```

[0085] Oben erwähnte, unbestimmte Nicht-Anschlüsse werden nachstehend beschrieben:

1. version-identifier (Versionskennzeichnung): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z], bei der das erste Zeichen aus der Menge [0-9] sein muss.
2. positive-integer (positive ganze Zahl): Eine Folge von einem oder mehreren Zeichen [0-9], die nicht mit einer 0 beginnt.
3. id-code (Identifikationscode): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z].
4. resource-name (Ressourcenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z], bei der das erste Zeichen aus der Menge [a-zA-Z] sein muss.

[0086] Kommentare werden unterstützt; Kommentare beginnen mit dem #-Zeichen und erstrecken sich bis zum Ende der Zeile.

A4. Pin-Beschreibungen

[0087] Die Pin-Beschreibungen eines Prüfobjektes (DUT) werden unter Verwendung einer Pin-Beschreibungsdatei beschrieben. Der Anwender macht eine Beschreibung der DUT-Pins in einer Pin-Beschreibungsdatei, die die Erweiterung .pin aufweist, verfügbar. Diese Klartextdatei enthält zumindest folgendes: Eine Aufstellung der DUT Pinnamen und Anfangsdefinitionen von nominierten Pingruppen, die von den definierten DUT Pinnamen Gebrauch machen („Anfang", weil sie anschließend modifiziert oder programmatisch hinzugefügt werden können, usw.).

[0088] Die Trennung dieser Datenspezifikation von der Testplan-Beschreibung erlaubt eine generelle Wiederverwendung der DUT Pindefinitionen und ermöglicht es dem Strukturkompilierer, Pinnamen (benötigt zum Auflösen von Verweisen auf in Vektorspezifikationen verwendeten Pinnamen) aus der Pin-Beschreibungsdatei abzuleiten, ohne den Prozess an einen spezifischen Testplan binden zu müssen.

[0089] Nachstehend ist eine beispielhafte Pin-Beschreibungsdatei gezeigt:

```
#
# Pin-Beschreibungsdatei, myDUT.pin.
#
# Es wird angemerkt, dass diese die
# Ressourcenkonfigurationsdatei Resour-
ces.rsc
  importiert.
```

```
#
Version 1.1.3a;
PinBeschreibung
{
  Ressource dpin
  {
    A0;
    A1;
    A2;
    A3;
    A4;
    # Diese Syntax erweitert auf die
Namen
      „ABUS[1]“ und „ABUS[2]“
    ABUS[1:2];
    A5;
    BBUS[1:8];
    DIR;
    CLK;

    Gruppe Grp1
    {
        DIR, CLK, A0, A1, A2, A3, A4,
BBUS[1:4]
    }
    Gruppe Grp2
    {
       A5,
       #
       # Die folgende Zeile wird erwei-
tern auf
       # „DIR, A1, A2, A4, A5, BBUS[2]“
       #
       Grp1 - CLK-A0-A3 -BBUS[1]-
BBUS[3:4] + A5,
       BBUS[5:8]
```

```
    }
  }

  Ressource dps
  {
     vcc1;
     vcc2;
     vcc3;

   Gruppe PSG
   {
     vcc1, vcc2
   }
 }
}
```

[0090] Zu beachten ist, dass die DUT-Pin-Definition und Pingruppendefinition innerhalb von Ressourcentypblöcken eingebettet sind, um es dem Kompilierer zu ermöglichen, Definitionen von Pin und Pingruppen mit den zulässigen Parametereinstellungen für Ebenen, usw. zu korrelieren.

[0091] Es sollen die folgenden Punkte über Pin-Beschreibungen beachtet werden:

1. Pingruppen und Pins teilen sich den gleichen Namensraum und besitzen globalen Umfang (d. h. Testplan). Eine der Konsequenzen der globalen Umfangsbildung dieser Namen ist, dass Pins und Pingruppen keine kopierten Namen verwenden können, auch wenn sie in unterschiedlichen Ressourcenblöcken bekannt gemacht sind.
2. Mindestens eine Ressourcendefinition wird in der Pinbeschreibungsdatei benötigt.
3. Mindestens ein Pinname sollte in jeder Ressource definiert sein.
4. Es ist erforderlich, dass Pin und Gruppennamen innerhalb der Ressourcengrenzen einmalig sind.
5. Für zwei oder mehrere Ressourcen kann der gleiche Pin oder Gruppenname definiert werden. Jedoch werden Duplikate innerhalb der gleichen Ressource ignoriert.
6. Alle Namen und Gruppennamen, die in einer Gruppendefinition erscheinen, sollten bereits innerhalb dieser Ressource definiert worden sein.
7. Gruppendefinitionen, falls gegeben, sollten zumindest einen Pinnamen oder Gruppennamen besitzen (d. h. eine Gruppendefinition kann nicht leer sein).
8. Eine Pingruppendefinition kann einen Bezug auf eine zuvor definierte Pingruppe enthalten.
9. Eine Pingruppendefinition kann Mengenoperationen wie beispielsweise Addition und Subtraktion von zuvor definierten Pins und/oder Pingruppen enthalten.

Struktur für die Pin-Beschreibungen

[0092] Nachstehend ist die Struktur für die Pin-Beschreibungen entsprechend der bevorzugten Ausführung der vorliegenden Erfindung gegeben:

```
pin-description-file:
   version-info pin-description

version-info:
    Version version-identifier;

pin-description:
    Pinbeschreibung {resource-pins-def-list}

        resource-pins-def-list:
           resource-pins-def
           resource-pins-def-list resource-pins-def

        resource-pins-def:
          Ressource resource-name {pin-or-pin-group-
def-list}

        pin-or-pin-group-def-list:
            pin-or-pin-group-def
            pin-or-pin-group-def-list pin-or-pin-
group-def

        pindef-or-pin-groupdef:
           pin-def;
           pin-group-def

        pin-def:
            pin-name
           pin-name [index : index]

        pin-group-def:
            Gruppe pin-group-name {pin-group-def-
item-list}

        pin-group-def-item-list:
            pin-def
            pin-group-def-item-list, pin-def
```

[0093] Oben erwähnte, unbestimmte Nicht-Anschlüsse sind nachstehend festgelegt:

1. version-identifier (Versionskennzeichnung): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z]. Sie stellt eine Versionsnummer dar.
2. resource-name (Ressourcenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen einer Quelle wie beispielsweise dpin

oder dps dar.

3. pin-name (Pinname): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen eines Pins A0 dar.

4. pin-group-name (Pingruppenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], die nicht mit einer Ziffer beginnt. Sie stellt den Namen einer Pingruppe ABUS dar.

5. index (Index): eine nicht negative ganze Zahl. Sie stellt die Untergrenze oder eine Obergrenze an einer Gruppe von zugeordneten Pins dar.

A5. Der Socket

[0094] Der Socket legt die Zuordnung zwischen DUT-Pinnamen und Pin (Kanal) Zuweisungen physikalischer Tester fest (die Kanalnummern physikalischer Tester sind in der Modulkonfigurationsdatei definiert). Zu beachten ist, dass unterschiedliche Sockets verwendet werden können, um unterschiedliche DUT-Pakete und unterschiedliche Lademodul-Konfigurationen, usw. zu unterstützen. Bei einem Multi-DUT-System können die Socket-Definitionen für DUT/Kanalzuweisungen das "Klonen" eines grundlegenden Sockets zu mehrfachen Standorten unterstützen. Jedoch sollten unterschiedliche Sockets (d. h. unterschiedliche physikalische Zuordnungen für die gleichen logischen Pins) Site-Modul-Speicherblöcke respektieren. Folglich definiert der Socket, zusätzlich zur Bereitstellung von Zuweisungen DUT-Pin-zu-Testerkanal, auch effektiv die Standort-Einteilung. Ein Socket-File könnte somit Definitionen für mehrere einzelne Standort-Sockets enthalten. Nachstehend gezeigt ist der drei DUT-Standorte definierende Abtast-Socket-File:

```
          Version 1.1.3
          SocketDef
          {
            DUTType Pentium3
            {
              PinBeschreibung dutP3.pin;
              #Die Pin-Beschreibungsdatei für Pentium3
              DUT2 # Verwendet die Vollspezifikations-
Syntax
              {
                SiteController 1;
                # Switchmatrix-Eingangsport

        Ressource dpin
        {
          #
          # Der CLK Pin wird Ressource dpin zugeord-
net,
          # Slot 2, Ressourceneinheit (Kanal) 13.
          #
          CLK            2.13;
          #
          # Der DIR-Pin wird Ressource dpin zugeord-
net,
          # Slot 5, Ressourceneinheit 15.
          DIR   5.15;
          #
          # Die folgende Anweisung wird erweitert zu
          # BBUS[7] 5.4
          # BBUS[6] 5.5
          # BBUS[5] 5.6
```

```
        #

        # So wird zum Beispiel die Pinfolge BBUS[7],
BBUS[6],
        # BBUS[5] dem gleichen Slot 5 und
        # Ressourceneinheiten 4, 5 bzw. 6 zugewiesen
        #
        BBUS[7:5]    5.[4:6];
        BBUS [1:4]   7. [21:18];
        BBUS[8]      9.16;
    }

    Ressource dps
    {
      #
      # Der Pin V1 wird Quelle dps zugewiesen,
      # Slot 1, Ressourceneinheit (Kanal) 1.
      #
      VCC1 1.1;
      #
      # Der Pin VCC2 wird Ressource dps zugeordnet,
      # Slot 1, Ressourceneinheit (Kanal) 2.
      #
      VCC2 1.2;
    }
  } #EndDUT2

  DUT1 # Dieses wird von DUT2 oben „geklont".
  {
    Site-Controller 1; # gleicher Site-Controller
wie für
                       DUT2
  Ressource dpin
  {
     SlotOffset 1;    # Offsetwert für Slots
  }
```

```
    Ressource dps
    {
       SlotOffset 10; # Offsetwert für Slots
    }
    #
    # Die oben erwähnte Offset-Syntax gibt an, dass
die
    # Zuordnungen Slot/Ressourceneinheit aus dem ers-
ten DUT
    # „geklont" sind, der für diesen DUTTyp, d.h. DUT
2, mit
    # den durch die SlotOffset-Werte aufgerechneten
Slots
    # definiert ist.
    #
    # Die Definition von dpin Ressourceneinheiten für
    # DUT2 ansehen, CLK ist an Slot 2 gebunden. Daher
ist für
    # das vorhandene DUT CLK an Slot 2 + 1 = 3 gebun-
den.
    #
    # Einige der neuen Bindungen, im Wesentlichen in-
folge der
    # aufgerechneten Zuordnungen sind in der Tabelle
unten
    # dargestellt:
    # ----------------------------
    #   Pin     Ressourcee   R-Einheit   Slot
    # ----------------------------
    #   CLK    dpin     13     2 + 1
= 3
    #   DIR    dpin     15     5 + 1
= 6
    # BBUS[8] dpin  16   9 +  1 = 10
    # VCC1    dps    1   1 + 10 = 11
    # VCC2    dps    2   1 + 10 = 11
```

```
    #
  } # EndDUT 1
}# EndDUTType Pentium 3

    DUTType 74LS245
    {

    PinBeschreibung dutLS.pi;

    DUT3 arbeitsunfähig
    # Diese DUT-Position ist arbeitsunfähig und wird
      ignoriert
   {
    ...
   }
  }#End DUTTyp 74LS245
}#End SocketDef
```

[0095] Es sollten die folgenden Punkte über einen Socket-File beachtet werden:

1. Der Socket-File nutzt Informationen sowohl aus der Modul-Konfigurationsdatei als auch den Pin-Beschreibungsdateien des Anwenders für die gegebenen DUT-Typen (siehe Spezifikation für Pin-Description in dem oben erwähnten Beispiel). Die Modul-Konfigurationsinformationen werden dem Socket-File-Kompilierer implizit verfügbar gemacht. Der Socket-File-Kompilierer ist ein Unterteil des Strukturkompilierers, der den DUT-Namen des Sockets liest und auf Zuordnung des Testerkanals sowie die Modulkonfiguration und Pin-Beschreibungsdateien analysiert, um die Zuordnung der Testerpins zu DUT-Pins, die von dem Strukturkompilierer genutzt werden, einzurichten.
2. Mindestens eine Definition des DUT-Standortes je DUT-Typ wird benötigt, die im Gegensatz zur SlotOffset-Syntax die Vollspezifikations-Syntax verwenden muss. Falls mehr als eine DUT-Standortinformation für den gleichen DUT-Typ vorgesehen ist, muss die erste die Vollspezifikations-Syntax verwenden.
3. Jede folgende DUT-Standortdefinition (für den gleichen DUT-Typ) kann entweder die Syntax Vollspezifikation oder die Syntax SlotOffset, jedoch nicht beide, verwenden. Dies ermöglicht es, dass einzelne Positionen von einem Standardmuster abweichen (aufgrund von beispielsweise nicht in Betrieb befindlichen Kanälen).
4. Die von der SlotOffset-Syntax abgeleiteten Bindungen werden im Verhältnis zu der für diesen DUT-Typ (der die volle Spezifikation-Syntax nutzt) definierten ersten Position definiert.
5. DUT-Standorte müssen nicht in der tatsächlichen physikalischen Reihenfolge angemeldet werden. Dies ermöglicht den Fall, bei denen der erste (physikalische) Standort von dem Muster abweicht.
6. Die IDs der DUT-Standorte müssen über den gesamten Socket (d. h. über alle darin definierten DUT-Typen) eindeutig sein.
7. Pro DUT-Standortdefinition wird mindestens eine Ressourcendefinition benötigt.
8. Die Standortdefinitionen müssen in Verbindung mit der Modulkonfiguration verwendet werden, um zu bestimmen, ob die Testkonfiguration Einzelstandort/Einzel-DUT oder Einzelstandort/Mehrfach-DUT ist.
9. In allen Fällen sollte der Socket-File eine Menge von DUT-Kanalzuordnungen spezifizieren, die mit der Pin-Beschreibungsdatei und der Modul-Konfigurationsdatei in Einklang stehen.
10. In einigen Fällen wird es erwünscht sein, die Socket-Definition so spezifizieren zu können, dass einer oder mehrere DUT-Kanäle von dem Tester getrennt werden (zum Beispiel dadurch, dass der zugewiesene physikalische Kanal als einer mit der speziellen ID"0.0" bezeichnet wird). In diesem Fall können diese DUT-Kanäle im Zusammenhang mit dem Testprogramm genutzt und berücksichtigt werden. Operationen an solchen Kanälen werden zu Systemwarnungen (jedoch nicht Fehler) führen. Zum Ladezeitpunkt werden Strukturdaten für getrennte Kanäle gelöscht.

Struktur für den Socket

[0096] Das Folgende ist die Struktur für die Modulkonfiguration nach einer bevorzugten Ausführung der vorliegenden Erfindung:

```
socket-file:
    version-info socket-def

version-info:
   Version version-identifier;

socket-def:
   SocketDef {device-specific-Socket-def-list}

device-specific-Socket-def-list:
    device-specific-socket-def
device-specific-socket-def-list device-specific-
socket-def

device-specific-socket-def:
```

```
            DUTType DUT-type-name {pin-description-
file dut-
                    info-list}

    pin-description-file:
        PinDesc pin-description-file-name;

    dut-info-list:
        dut-info
        dut-info-list dut-info

    dut-info:
        DUT dut-id {site-controller-input-port re-
source-info-
            list}

    site-controller-input-port :
         SiteController switch-matrix-input-port-
number;

    resource-info-list:
        resource-info
        resource-info-list resource-info

    resource-info:
        Resource resource-name {resource-item-unit-
assignment-
                list}

    resource-item-unit-assignment-list:
        resource-item-unit-assignment
        resource-item-unit-assignment-list resource-
item-unit-
        assignment

    resource-item-unit-assignment:
```

```
        resource-item-name slot number .resource-unit;
        resource-item-name [resource-item-index] slot-
number
        .resource-unit-index;
        resource-item-name [resource-item-index-range]
            slot-number .[resource-unit-index-range];

    resource-item-index-range:
        resource-item-index: resource-item-index

    resource-unit-index-range:
         resource-unit-index: resource-unit-index
```

[0097] Oben erwähnte, unbestimmte Nicht-Eingänge sind nachstehend festgelegt:

1. version-identifier (Versionskennzeichnung): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z]. Sie stellt eine Versionsnummer dar.
2. DUT-type-name (DUT-Typenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z], in der das erste Zeichen nicht aus der Menge [0-9] sein muss. Sie stellt einen DUT-Typ, wie beispielsweise Pentium 3, dar.
3. pin-description-file-name (Pinbeschreibungs-Dateiname): Der einfache Name einer Datei, der ihren Verzeichnisnamen nicht enthält, jedoch alle Erweiterungen einschließt. Der Dateiname ist von der Syntax, die vom Betriebssystem des Zentralrechners erkannt wird, und erlaubt Leerstellen und andere Zeichen, wenn sie in Zitaten eingeschlossen sind.
4. switch-matrix-input-port-number (Switchmatrix-Eingangskanalnummer): Eine nicht negative ganze Zahl in Dezimalschreibweise zum Darstellen der Kanalnummer des mit dem Site-Controller verbundenen Eingangskanals.
5. dut-id: Eine nicht negative ganze Zahl in Dezimalschreibweise, um ein Beispiel eines DUT zu identifizieren.
6. resource-name (Ressourcenname): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z], in der das erste Zeichen keine Ziffer sein muss. Sie stellt den Namen einer in einem Ressourcenfile definierten Ressource dar.
7. resource-item-name (Ressourcendatenelementname): Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9a-zA-Z], in der das erste Zeichen keine Ziffer sein muss. Sie stellt den Namen einer Ressourceneinheit wie beispielsweise ein Pin oder eine Pingruppe dar.
8. resource-item-index (Ressourcendatenelementindex): Eine nicht negative ganze Zahl in Dezimalschreibweise, die ein spezielles Element einer Gruppe von Ressourcendatenelementen darstellt. Im Zusammenhang mit einem Indexbereich eines Ressourcendatenelements stellt er die Unter- oder Obergrenze einer zusammenhängenden Folge einer Ressourcen-Datenelementgruppe dar.
9. resource-unit-index (Ressourceneinheitsindex): Eine nicht negative ganze Zahl in Dezimalschreibweise, die ein spezielles Element einer Gruppe von Ressourceneinheiten (Kanälen) darstellt. Im Zusammenhang mit einem Indexbereich eines Ressourcendatenelements stellt er die Unter- oder Obergrenze einer zusammenhängenden Folge einer Ressourceneinheitsgruppe dar.

A6. Pins

[0098] Es ist anzumerken, dass zusätzlich zum logischen Pinnamen für physikalische Kanalzuordnungen (wie durch den Socket bewirkt), zum Spezifizieren der Tester-Ressourcen verschiedene Attribute genutzt werden können. Zum Beispiel könnten Optionen verwendet werden, um spezielle Hardwarekonfigurationen für Kanäle zu definieren, die testspezifisch, herstellerspezifisch und/oder testsystemspezifisch sein können. Diese werden durch die Pin-Modus-Optionen beschrieben und über eine Pin-Modus-Optionen-Datei verfügbar gemacht.

[0099] Eine Definition der Pin-Modus-Option würde die Konfiguration spezieller Optionen oder Modi für einen Testerkanal unterstützen. Diese könnte zum Beispiel verwendet werden, um eine Mehrfachschaltung von Kanälen auszuwählen und zu konfigurieren. Bevorzugt wird, das die Pin-Modus-Optionen nur als Teil eines Test-

plan-Initialisierungsablaufes genutzt werden, weil sie eine signifikante Kanalkonfiguration erforderlich machen könnten. Die Syntax der Pin-Option unterstützt vom Hersteller definierte Optionen. Nachstehend ist ein Beispiel gezeigt:

```
PinModusOptionen
{
   Taktimpuls     EIN     doppelt;
   a0             AUS        einfach;
   ...
};
```

Konfiguration der Testumgebung

[0100] Wie früher dargelegt, werden die Ressourcendefinitionsdatei (Resources.rsc), die Systemkonfigurationsdatei (Sys.cfg) und die Modulkonfigurationsdatei (Modules.cfg) vorzugsweise an einer „bekannten" Örtlichkeit verfügbar gemacht. Diese „bekannte" Örtlichkeit ist das Verzeichnis, das durch den Wert der Systemumgebungsvariablen Tester_ACTIVE_CONFIGS bestimmt ist. Wenn zum Beispiel der Wert von Tester-ACTIVE_CONFIGS das Verzeichnis F:\TesterSYS\configs ist, wird das System erwarten, dass die folgenden Dateien vorhanden sind:
F:\Tester_SYS\configs\Resources.rsc
F:\Tester_SYS\configs\Sys.cfg
F:\Tester_SYS\configs\Modules.cfg

[0101] Während einer Installation wird das auf dem Zentralrechner liegende Installations- und Konfigurationsverwaltungssystem (ICM) vorzugsweise den Wert von Tester_ACTIVE:_CONFIGS setzen. Jedes Mal, wenn das ICM eine neue Version von einer der oben erwähnten Dateien erzeugt, wird es die neue Version in den Speicherplatz, auf den durch Tester_ACTIVE_CONFIGS verwiesen wird, legen. Anzumerken ist, dass zusätzlich zu den oben erwähnten drei Dateien andere Systemkonfigurationsdateien wie beispielsweise die Simulationskonfigurationsdatei auch in dem Speicherplatz abgelegt werden, auf den durch Tester_ACTIVE_CONFIGS verwiesen wird.

B. Regeln zur Testprogrammentwicklung

[0102] Eine der zwei prinzipiellen auf Endbenutzer orientierten Komponenten des Testersystems ist die Testumgebung. Die andere Komponente ist die Programmiereinrichtung, die der Tester für den Endbenutzer (d. h. Prüfingenieur und Entwickler von Testklassen) verfügbar macht.

[0103] Die Hautkomponente der Programmierumgebung ist der Testplan. Der Testplan nutzt Testklassen (die unterschiedliche Implementierungen einer mit Tester bezeichneten Testschnittstelle sind), die die Trennung von Testdaten und Code für spezielle Typen von Tests realisieren.

[0104] Der Plan kann direkt als ein C++-Testprogramm geschrieben oder in eine Testplan-Beschreibungsdatei eingeschrieben werden, die durch einen Testprogrammgenerator (Übersetzungsprogramm **402**) verarbeitet wird, um einen Objekt-orientierten Code wie beispielsweise C++-Code, zu erzeugen. Der generierte C++-Code kann anschließend zu dem ausführbaren Testprogramm kompiliert werden. Die zur Besetzung eines Testklassenbeispiels wie Ebenen, zeitliche Verläufe, usw. benötigten Daten werden von dem Anwender in der Testplan-Beschreibungsdatei festgelegt.

[0105] Ein Testprogramm enthält einen Satz von vom Anwender geschriebenen Dateien, die Details zum Führen eines Tests an einem Baustein festlegen. Eine Ausführung der Erfindung umfasst Sätze von Regeln, die es einem Anwender erlauben, diese Dateien unter Verwendung von C++-Konstrukten zu schreiben.

[0106] Eine der Anforderungen gemäß der Ausführung der Erfindung ist, der Austauschbarkeit des Testsystems mit offener Architektur zu folgen. Eine Modulentwicklung erlaubt Anwendern, einzelne Komponenten zu schreiben, die sich mit unterschiedlichen Aspekten des Tests befassen, und erlaubt anschließend, dass diese Komponenten gemischt und auf verschiedenen Wegen angeglichen werden, damit sie ein komplettes Testprogramm ergeben. Ein Testprogramm entsprechend der bevorzugten Ausführung der vorliegenden Erfindung umfasst einen Satz von Files wie folgt:

files *.usrv für Benutzervariable und Konstanten;
files *.spec für Spezifikationsgruppen;
files *.lvl für Ebenen;
files *.tim für zeitliche Steuerungen;
files *.tcg für Testbedingungsgruppen;
files *.bdefs für Binärdateidefinitionen;
files *.ph für einen Preheader, Dateien für kundenspezifische Funktionen und Testklassen.
files *.ctyp für kundenspezifische Typen;
files *.cvar für kundenspezifische Variable; und
files *.tpl für Testpläne.

[0107] Die obigen Filenamenerweiterungen sind eine empfohlene, die Vereinbarung erleichternde Aufschlüsselung von Files. Ein einzelnes Testprogramm wird vorzugsweise ein einzelnes Testplan-File und die Files, die es importiert, aufweisen. Ein „Import" bezieht sich auf andere Files mit Daten, auf die entweder direkt durch den Importer (der File, der den Import bestimmt) verwiesen wird, oder die durch einen gewissen anderen File, auf den durch den Importer direkt verwiesen wird, importiert werden. Der Testplan-File könnte Globals, Abläufe und andere solche Ziele innerhalb desselben definieren, oder er könnte diese Informationen von anderen Files importieren. Diese Regeln ermöglichen jeder der oben erwähnten Komponenten, sich entweder in ihren eigenen individuellen Files oder direkt in einem Testplan-File mitlaufend zu befinden. Anzumerken ist, dass der Testplan im Konzept einer C-Sprachen-Haupt()-Funktion ähnlich ist.

Eigenschaften von Testprogrammen

Benutzervariable und Konstanten,
Spezifizierungsset,
Ebenen,
Zeitabläufe,
Testbedingungen,
Definition der Binärdatei,
Preheader,
Kundenspezifische Typen,
Kundenspezifische Variable,
Testplan

[0108] Testprogramm-Identifizierer starten vorzugsweise mit einem großbuchstabigen oder kleinbuchstabigen alphabetischen Zeichen und können anschließend eine beliebige Anzahl alphabetischer, numerischer Zeichen oder Unterstrichzeichen ( ) aufweisen. Es besitzt mehrere Kennworte, die in der nachstehend gegebenen Beschreibung vorgesehen sind. Diese Kennworte werden visuell im Code in dieser Druckschrift codiert identifiziert, indem eine steile Schriftart wie beispielsweise Version genutzt wird. Kennworte sind reserviert und werden vorzugsweise nicht als Identifizierer verwendet. Es gibt mehrere spezielle Symbole wie {,}, (,),: und andere, die nachstehend beschrieben sind.

Entwicklung von Testobjekten

[0109] Der Import einer Testbeschreibungsdatei ermöglicht es der importierenden Datei, auf Namen von Objekten zu verweisen, die durch die importierte Datei verfügbar gemacht werden. Dies ermöglicht es der importierenden Datei, sich auf die Objekte zu beziehen, die durch die importierte Datei nominiert sind. Betrachtet wird ein Socket-File aaa.soc, der eine Pinbeschreibungsdatei xxx.pin importiert. Es könnte eine andere Datei bbb.soc geben, die ebenfalls xxx.pin importiert. Jedoch zwingt keiner dieser Importe die durch xxx.pin beschriebenen Objekte, existent zu werden. Sie beziehen sich lediglich auf Objekte, von denen angenommen wird, dass sie bereits existieren.

[0110] Die Frage erhebt sich: Wann werden solche Objekte existent? Dies geschieht dann, wenn der Testplan-File grundsätzlich anders ist. In Analogie zu C wäre es ein File mit einer Haupt-()routine darin. Eine "Import" Anweisung im Testplan-File wird diese Objekte entwickeln, d. h. diese Objekte zwingen, existent zu werden. Der unten gezeigte Testplan mickey.tpl zwingt die Objekte in xxx.pin und aaa.soc entwickelt zu werden:

```
    #File für Mickey's Testplan
    Version 3.4.5;

    #
    # Diese Importanweisungen werden die Objekte
    # tatsächlich zwingen, existent zu werden:
    #
    Import xxx.pin; #Entwickelt Pin- und Pingruppen-
Objekte
    Import aaa.soc; # Entwickelt Site-Socket-
Zuweisungsobjekte

# andere Importe als notwendig
...
Ablauf Flow1
{
  ...
    }
```

[0111] Ein Import von xxx.pin in den Testplan bewirkt, dass alle in xxx.pin vereinbarten Pin- und Pingruppen-Objekte entwickelt werden. Dies wird wie folgt beschrieben: „der File xxx.pin wird entwickelt". Es ist nicht notwendig, dass ein Testplan alle die Files direkt importiert, die entwickelt werden müssen. Der File x wird durch einen File y importiert, wenn jede der zwei nachstehenden Anweisungen wahr ist:

1. y besitzt eine wichtige Anweisung, die x benennt; oder
2. x wird durch z importiert, und y besitzt eine z benennende wichtige Anweisung.

[0112] Wenn ein Testprogramm kompiliert ist, wird es alle Objekte in den Files entwickeln, die durch den Testplan importiert sind. Der Satz von durch einen Testplan importierten Files wird topologisch sortiert, um eine Reihenfolge zu liefern, in der die Files entwickelt sind. Der von einem Testplan importierte Satz von Files bezieht sich auf den Importabschluss des Testplans. Wenn der Importabschluss eines Testplans nicht topologisch sortiert werden kann, dann muss es einen Importzyklus geben. Eine solche Situation ist unrichtig und wird durch den Kompilierer zurückgewiesen.

Benutzervariable und Konstanten

[0113] Globale Variable und Konstanten werden unter Verwendung der Benutzervariablen und Konstanten definiert.

[0114] Konstanten sind Objekte, deren Wert an Übersetzungszeit gebunden ist und nicht geändert werden kann. Der maximale ganzzahlige Wert würde zum Beispiel eine Konstante sein. Andererseits kann sich die Ausdrucksbindung an Variable bei Laufzeit über eine API (Anwendungsprogrammierschnittstelle) ändern.
ganze Zahl,
vorzeichenlose Ganzzahl,
Fließkommzahl doppelter Genauigkeit,
Sequenz,
Spannung in Volt (V),
Spannungsanstieg in Volt pro Sekunde (VPS),
Strom in Amp (A),
Leistung in Watt (W),
Zeit in Sekunden (s),
Länge in Metern (m),
Frequenz in Hertz (Hz),
Widerstand in Ohm (Ohm), und

Kapazität in Farad (F).

[0115] Die Typen Ganze Zahl, Vorzeichenlose Ganzzahl, Fließkommazahl doppelter Genauigkeit und Sequenz sind auf Grundtypen bezogen. Die Grundtypen besitzen keine Maßeinheiten. Die Basistypen, die keine Grundtypen sind, sind eine Fließkommazahl doppelter Genauigkeit mit einer zugeordneten Maßeinheit und einer Maßeinteilung. Die Skalierungssymbole sind normale Maßeinteilungssymbole der Technik:
p (pico) für 10^{-12}, wie in pF (Picofarad)
n (nano) für 10^{-9}, wie in ns (Nanosekunde)
µ (micro) für 10^{-6}, wie in µs (Mikrosekunde)
m (milli) für 10^{-3}, wie in mV (Millivolt)
k (kilo) für 10^{3}, wie in kO (Kiloohm)
M (mega) für 10^{6}, wie in MHz (Megahertz)
G (giga) für 10^{9}, wie in GHz (Gigahertz)

[0116] Ein getrennter File mit Benutzervariablen und Konstanten wird die Erweiterung .usrv. haben. Das Folgende ist das Beispiel eines Files mit einigen globalen Konstanten. Das Beispiel eines Files mit einigen Variablen wird später gegeben.

```
                # ----------------------
                # Filegrenzen.usrv
                # ----------------------

                Version 1.0.0;

          #
          # Diese UserVars-Sammelvereinbarung
          # vereinbart eine Menge von global
          # verfügbaren Variablen und Konstanten.
          #
          UserVars
          {
            # Einige konstante ganzzahlige Globals,
die an
            # verschiedenen Stellen verwendet werden.
            Const Integer  MaxInteger =  2147483647;
            Const Integer  MinInteger = -2147483648;

            # Kleinster Wert, so dass 1,0+Epsilon ! =
1,0
            Const Double Epsilon =
2,2204460492503131e-016;

            # Einige Konstanten, die auf Fließkomma-
zahl
              doppelter Genauigkeit bezogen sind
```

```
            Const Double MaxDouble =
1,79769313486231 58e+308;
            Const Double MinDouble = -MaxDouble;
            Const Double ZeroPlus =
2,22507385850720.14e-308;
            Const Double ZeroMinus = -ZeroPlus ;


          }
```

[0117] Die Menge von oben vereinbarten UserVars sind in Betracht gezogene Definitionen der Variablen links von ‚='. Folglich wird einzelnes Auftreten der Definition einer Variablen oder Konstanten bevorzugt, und sie sollte initialisiert werden.

[0118] Wie früher erwähnt, sollten Konstanten nicht geändert werden, wenn sie einmal definiert sind. Die Ausdrucksbindung an eine Konstante kann vorher definierte Konstanten und direkte Werte umfassen. Andererseits können Variable über eine API (Anwendungsprogrammierschnittstelle) geändert werden. Die Ausdrucksbindung an eine Variable kann vorher definierte Variable, Konstanten und direkte Werte einschließen.

[0119] Jede Variable ist an ein Ausdrucksobjekt gebunden, das zur Laufzeit beibehalten wird. Dies bewirkt die Fähigkeit, den einer Variablen bei Laufzeit zugeordneten Ausdruck zu ändern und anschließend alle Variablen wieder zu bewerten. Das Ausdrucksobjekt ist eine analysierte Form der rechten Seite einer Variablen oder Konstantendefinition. In einer Ausführung ist kein Leistungsmerkmal für die Änderung von Konstanten zur Laufzeit vorgesehen. Ihr Wert ist vorzugsweise zur Übersetzungszeit fixiert.

[0120] In dem Importabschluss eines Testplans kann eine beliebige Anzahl solcher Files mit Globals vorhanden sein. Während der oben erwähnte Global-File eine Menge von numerischen Grenzwerten ist, ist er hier eine Menge von technischen Globals, die technische Maßeinheiten nutzen, und zufälligen Benutzervariablen:

```
    # -------------------------
    # File myvars.usrv
    # -------------------------

    Version 0.1;

    #
    # Diese vereinbart eine UserVars Sammlung von
einigen
    # technischen Globals.
    #

    UserVars MyVars
    {
      # technische Größen
      Const Spannung VInLow  = 0,0;          # 0
Volt
      Const Spannung VInHigh = 5,0;          # 5
Volt
      Const Spannung VOutLow = 400,0 mV;      # 400
Millivolt
      Const Spannung VOutHigh = 5,1;          # 5,1
Volt
      Const Zeit DeltaT = 2,0E-9;           # 2 Na-
nosekunden
      Const Zeit ClkTick = 1,0 ns           # 1 Na-
nosekunde
      Const Widerstand R10 = 10,0 kO;       # 10 Ki-
loohm
```

```
        # Einige Variable sind nachstehend vereinbart.
        Strom ILow = 1,0 mA;          # 1 Milliampere
        Strom IHigh = 2,0 mA          # 2 Milliampere
        Leistung PLow = ILow* VInLow; # niedriger
Leistungswert
        Leistung PHigh = IHigh*VInHigh;  # hoher Leis-
tungswert


      #
      # Eine Matrix niedriger Wert für alle A-Bus-
Pins.
      # Die Vertikalinjektionslogik wird für A0 in
ABusVil[0],
      # für A1 in ABusVil[1], und so weiter sein.
      #
       Spannung ABusVil[8] = {1,0, 1,2, andere =
1,5};
    }
```

[0121] Der Kompilierer prüft vorzugsweise, dass sich Einheiten und Typen nach oben angleichen. Es ist zu beachten, dass die Gleichungen für oben erwähnte PLow und PHigh kompilieren werden, da Spannung mal Strom Leistung ergibt. Jedoch ist typisch, dass eine Anweisung wie die folgende nicht kompilieren wird:

```
        #
        # Kompiliert nicht, weil Strom und Spannung
nicht
        # addiert werden können, damit sich Leistung
ergibt.
        #
        Leistung Pxxx = IHigh + VInHigh;
```

[0122] Der Kompilierer wird bestimmte automatische Typkonvertierungen erlauben:

```
        Power Pxxx = 2;      # Leistung auf 2,0 Watt
setzen
        Integer Y 0 3,6;     # Y bekommt 3 zugeordnet
        Power Pyyy = Y;      # Pyyy bekommt 3,0 Watt
zugeordnet
        Double Z = Pyyy;     # Pyyy wird zu einer ein-
heitslosen
                               Fließkommazahl doppelter
                               Genauigkeit umgewandelt
```

[0123] Es ist auch explizite Typkonvertierung zu einer Fließkommazahl doppelter Genauigkeit, vorzeichenloser Ganzzahl und ganzer Zahl gestattet:

```
Power Pxxx = 3,5;

# Explizite Typkonvertierung ist erlaubt, wird
aber
  nicht benötigt.
# X wird 3,5
Double X = Double (Pxxx);            # wird 3,5
Integer Y = Integer (Pxxx);          # Y wird 3
```

[0124] Konvertierung zwischen beziehungslosen Typen ist auch möglich durch Konvertieren zu einem dazwischen liegenden Basistyp:

```
Power Pxxx = 3,5;

# Explizite Typkonvertierung wird benötigt.

Länge L = Fließkommazahl doppelter Genauigkeit
(Pxxx);
# L wird 3,5 Meter

Spannung V = ganze Zahl (Pxxx); # V wird 3,0
Volt.
```

[0125] Das Testplan-Objekt stellt eine UserVars-Klasse bereit, die eine die Namen und ihre zugeordneten Ausdrücke, Werte und Typen enthaltende Sammlung ist. Benutzervariable können in eine Variablensammlung für Standardbenutzer oder in eine Variablensammlung für nominierte Benutzer gehen. Die UserVars-Vereinbarungen in dem oben erwähnten Beispiel, die keinen festgelegten Namen haben, gehen in die Standard-Sammlung. Es ist jedoch möglich, eine Sammlung wie folgt explizit zu benennen:

```
      # Vereinbare X und Y in der Sammlung MyVars U-
serVars.
      UserVars MyVars
      {
         Integer X = 2,0;

         #
         # Verweist auf das oben erwähnte X und auf
die
         # global verfügbare MaxInteger aus der
         # Standardbenutzer-Variablensammlung.
         #
         Integer Y = MaxInteger - X;
      }
      # Vereinbare  X, Y1 und Y2 in der Sammlung Y-
ourVars
        UserVars.
      UserVars YourVars
      {
         Integer X = 3,0;
         # verweist auf das X aus MyVars.
         Integer Y1 = MaxInteger - MyVars.X;

         # verweist auf das oben vereinbarte X
         Integer Y2 = MaxInteger - X;
      }
      # Mehr Variable, die zu der MyVars-Sammlung ad-
diert
        werden
      UserVars MyVars
      {
        #
        # Verweist auf X und Y aus der früheren
        # Vereinbarung von MyVars.
        #
        Integer Z = X + Y;
     }
```

[0126] Namensauflösung innerhalb einer UserVars-Sammlung geht wie folgt vonstatten:
Wenn ein Name qualifiziert ist – d. h., ein Name umfasst zwei durch einen Punkt getrennte Segmente – dann kommt die Variable aus einer Variablensammlung nominierter Benutzer, die durch das Segment bestimmt sind, das dem Punkt vorausgeht. So bezieht sich oben erwähntes MyVars.X auf das X in der MyVars-Sammlung. Der Name „UserVars" kann genutzt werden, um die Variablensammlung der Standardbenutzer explizit zu be-

zeichnen.

[0127] Wenn der Name nicht qualifiziert ist und es eine Konstante oder Variable des gleichen Namens in der vorhandenen Sammlung gibt, dann löst sich der Name auf diese Konstante oder Variable auf.

[0128] Andererseits löst sich der Name auf eine Konstante oder Variable in der Variablensammlung der Standardbenutzer auf.

[0129] Man kann sich vorstellen, dass die Bewertung eines Blocks von Definitionen in einer UserVars-Sammlung aufeinander folgend von der ersten Definition bis zur letzten stattfindet. Dies kann erforderlich machen, jede Variable zu definieren, bevor sie verwendet wird.

[0130] Außerdem könnte es mehrere Blöcke von Definitionen für eine UserVars-Sammlung geben, wobei jede derselben mehrere Variable definiert. Man kann sich vorstellen, dass alle diese Blöcke von Definitionen in einer Vereinbarungsreihenfolge in dem Testplan bewertet werden und anschließend die Variablen jedes Blocks ebenfalls in Vereinbarungsreihenfolge geprüft werden.

[0131] Schließlich könnte es mehrere UserVars-Sammlungen geben, von denen jede Variable über mehreren Blöcken von Definitionen definieren. Man kann sich vorstellen, dass wiederum alle Variablen in Vereinbarungsreihenfolge initialisiert werden. So würde im oben erwähnten Beispiel die Bewertungsreihenfolge: MyVars.X, MyVars.Y, YourVars.X, YourVars.Y1, YourVars.Y2, MyVars.Z sein.

[0132] Wenn eine UserVars-Sammlung eine Variable aus einer anderen Sammlung nutzt, verwendet sie vorzugsweise nur den groben Wert der Variablen. Zwischen Sammlungen werden keine Abhängigkeitsinformationen aufrechterhalten. So kann eine auf Abhängigkeit basierte Wiederbewertung auf eine einzelne Sammlung begrenzt werden.

[0133] Jede Benutzer-Variablensammlung bezieht sich auf einen Fall einer C++-UserVars-Klasse. Das Standardobjekt der C++-UserVars-Klasse wird „_UserVars" genannt. Die Variablen in einer UserVars-Vereinbarung, die namenlos ist, sind aus der Standardbenutzer-Variablensammlung und werden zu diesem Standardobjekt addiert. Die Variablen in einer Variablensammlung nominierter Benutzer werden zu einem Objekt der diesen Namen aufweisenden C++UserVars-Klasse addiert. In dem oben erwähnten Beispiel wird das „MyVars" C++-Objekt mit den Variablen X, Y und Z abschließen.

C++ für Benutzervariable

[0134] Benutzervariable werden als eine Sammlung von n-Tupel mit der Namen-Sequenz, einer const/var-Aussagenlogik, dem Typ als einem spezifizierten Wert und dem Ausdruck als Ausdrucksbaum implementiert. Der Ausdruck eines Namens kann durch einen Aufruf gesetzt werden:

```
            enum ElementaryType {UnsignedIntegerT,
IntegerT,
                                 DoubleT, VoltageT,
...};
            Status setExpression(const String&name,
                                 const bool isConst,
                                 const elementaryTy-
pe,
                                 const Expressi-
on&expression);
```

[0135] Der Typ Ausdruck ist ein Typ, der eine analysierte Form des der rechten Seite einer Zuweisung entsprechenden Textes ist. Es wird einen global nutzbaren Fall von UserVars geben. Zum Beispiel wird die Menge von Benutzervariablen in limits.usrv (vgl. Seite) durch die Menge von nachstehend dargestellten Aufrufen implementiert:

```
          _UserVars.setExpression(„MaxInteger", true,
 IntegerT,
                                  Expressi-
 on(2147483647));
          _UserVars.setExpression(„MinInteger", true,
 IntegerT,
                                  Expression(-
 2147483648));
          _UserVars.setExpression(„Epsilon", true,
 DoubleT,
                          Expressi-
 on(2.2204460492503131e-016));
          _UserVars.setExpression(„MaxDouble", true,
 DoubleT,
                          Expressi-
 on(1.7976931348623158e+308));
          _UserVars.setExpression(„MinDouble", true,
 DoubleT,
                          Expression(„-MaxDouble"));
          _UserVars.setExpression(„ZeroPlus", true,
 DoubleT,
                          Expressi-
 on(2.2250738585072014e-308));
          _UserVars.setExpression(„ZeroMinus", true,
 DoubleT,
                          Expression(„-ZeroPlus"));
```

[0136] Nachstehend sind die C++-Anweisungen, die für die in myvars.usrv vereinbarten Variablen ausgeführt werden würden:

```
          myVars.setExpression(„VInLow", true, Volt-
ageT,
```

```
                                Expression(0.0));
        myVars.setExpression(„VInHigh", true, Volt-
ageT,
                                Expression(5.0));
        myVars.setExpression(„DeltaT", true, TimeT,
                                Expression(2.0E-9));
        myVars.setExpression(„ClkTick", true, TimeT,
                                Expression(1.0E-9));
        myVars.setExpression(„R10, true, Resistan-
ceT,
                                Expression(10.0E+3));
        myVars.setExpression(„ILow", false, Cur-
rentT,
                                Expression(1.0E-3));
        myVars.setExpression(„IHigh", false, Cur-
rentT,
                                Expression(2.0E-3));
        myVars.setExpression(„PLow", false, PowerT,
                                Expres-
sion("ILow*VInLow"));
        myVars.setExpression(„PHigh", false, PowerT,
                                Expression("IHigh *
VInHigh"));
        myVars.setExpression(„ABusVil[0]", false,
VoltageT,
                                Expression(1.0));
        myVars.setExpression(„ABusVil[1]", false,
VoltageT,
                                Expression(1.2));
        myVars.setExpression(„ABusVil[2]", false,
VoltageT,
                                Expression(1.5));
        myVars.setExpression(„ABusVil[3]", false,
VoltageT,
                                Expression(1.5));
```

```
        myVars.setExpression(„ABusVil[4]", false,
VoltageT,
                             Expression(1.5));
        myVars.setExpression(„ABusVil[5]", false,
VoltageT,
                             Expression(1.5));
        myVars.setExpression(„ABusVil[6]", false,
VoltageT,
                             Expression(1.5));
        myVars.setExpression(„ABusVil[7]", false,
VoltageT,
                             Expression(1.5));
```

[0137] Im oben genannten Code besitzt die Ausdrucksklasse vorzugsweise Konstruktoren, die die analysierte Form des Ausdrucks darstellen. Ein Ausdruck weist mehrere Konstruktoren auf, einschließlich eines, der eine Sequenz wortgetreu nimmt und sie analysiert, und eines anderen, der eine Sequenz wortgetreu nimmt, um sie nur als eine Sequenz wortgetreu zu verwenden. Diese werden durch zusätzliche Parameter unterschieden, die oben wegen der Lesbarkeit nicht spezifiziert sind.

[0138] Benutzervariable in der Variablensammlung der Standardbenutzer werden durch das UserVars-Objekt von Klasse UserVars verwaltet. Benutzervariable in einer Variablensammlung Xxx von nominierten Benutzern werden als ein Xxx genanntes UserVars-Objekt verwaltet.

Laufzeit einer API für UserVars

[0139] Die C++-UserVars-Klasse, die diese Namen und Ausdrücke enthält, exportiert eine Anwendungsprogrammierschnittstelle (API), um diese Werte zur Laufzeit zu bewerten und zu modifizieren. Eine Modifikation der UserVars zugeordneten Ausdrücke widmet sich außerdem der Frage, wann die UserVars neu bewertet werden und was die Auswirkungen der Evaluierung sein werden.

[0140] Zuerst wird die Frage betrachtet, wann die Neuevaluierung von UserVars als Ergebnis einer Änderung ausgelöst werden sollte. Wenn sie unmittelbar bei Herstellung einer Änderung an dem Ausdruck ausgelöst wird, dann wäre der Anwender nicht in der Lage, einer Reihe von entsprechenden Änderungen vor einer Auslösung der Neuevaluierung zu machen. Folglich wird eine Neuevaluierung durch einen expliziten Aufruf vom Anwender ausgelöst.

[0141] Als Nächstes können die Auswirkungen einer Neuevaluierung betrachtet werden. Es gibt drei Arten von Neuevaluierung, die entsprechend der bevorzugten Ausführung verfügbar sind.

[0142] UserVars Collection Re-evaluation ist eine auf eine einzelne UserVars-Sammlung begrenzte Neuevaluierung. Die Semantik dieser Operation ist, alle Variablen dieser Sammlung noch einmal neu zu bewerten.

[0143] UserVars Targeted Re-evaluation ist eine Neuevaluierung, die auf eine Änderung an der Ausdrucksbindung für einen einzelnen Namen begrenzt ist. Dies würde es dem Benutzer ermöglichen, den Ausdruck eines einzelnen Namens zu ändern und zu bewirken, dass die Neuevaluierung der Sammlung stattfindet, indem nur diese spezielle Änderung in Betracht gezogen wird.

[0144] UserVars Global Re-evaluation ist Neuevaluierung von allen UserVars-Sammlungen. Diese löst im Grunde genommen eine Neuevaluierung aller UserVars-Sammlungen in Vereinbarungsreihenfolge aus und ist sehr kostspielig.

[0145] Alle oben erwähnten Neuevaluierungen werden abhängige Objekte wie beispielsweise Ebenen, zeitliche Verläufe, usw. nach Neuevaluierung der UserVars neu bewerten. Abhängige Objekte werden ein Dirty Bit aufweisen, welches darstellt, dass eine Neuevaluierung benötigt wird. Jedes Mal, wenn eine UserVars-Sammlung programmatisch geändert wird, wird sie auch das Dirty Bit auf alle abhängigen Objekte setzen. Dies wird

eine Neuevaluierung der abhängigen Objekte auslösen.

[0146] Zusammenfassend unterstützen nominierte UserVars-Sammlungen, das Problem der Auswirkungen einer Neuevaluierung zu beherrschen. Neuevaluierung ist normalerweise auf eine einzelne Sammlung begrenzt. Eine einfache Möglichkeit der Verwendung von UserVars wäre es, nur die Standardsammlung von UserVars zu verwenden. Auf diese Weise kann der Welligkeitseffekt bei Vornahme einer Änderung allen UserVars passieren. Dieser Welligkeitseffekt kann dadurch begrenzt werden, dass mehrere nominierte UserVars-Sammlungen zugelassen werden.

[0147] Mehrfache Sammlungen können sich auf Variable voneinander beziehen, jedoch sind die an die Variablen gebundenen Werte an den Zeitpunkt der Verwendung gebunden. Zwischen UserVars-Sammlungen wird keine Abhängigkeit beibehalten.

[0148] Für jeden elementaren Typ Xxx (vorzeichenlose ganze Zahl, Strom, Spannung, usw.) ist eine Methode zur Erhaltung des Wertes:
Status getXxxValue(const String& name, Xxx&value) const;

[0149] Zu beachten ist, dass es keine Methode gibt, einen Wert direkt zu setzen, was durch Aufruf zum Setzen des Ausdrucks gemacht wird, dem ein Aufruf zum Neubewerten der Sammlung() folgt.

[0150] Methoden, um den Ausdruck zu bekommen und zu setzen. Das setExpression() kann auch genutzt werden, um eine neue Variable zu definieren, die bisher nicht definiert war.

```
        enum elementaryType
        {
           UnsignedIntegerT, IntegerT, DoubleT, Vol-
tageT, ...
        };
        Status getExpression(const String& name,
                             Expression& expres-
sion) const;
        Status setExpression(const String& name
                             const bool isConst,
                             const elementaryType,
                             const Expression& ex-
pression);
```

[0151] Der Aufruf setExpression() kann misslingen, wenn der Ausdruck zu einer periodisch wiederkehrenden Abhängigkeit führt. Wenn zum Beispiel die folgenden zwei Aufrufe vorgenommen werden würden, würde der zweite Aufruf mit einer Störung periodisch wiederkehrender Abhängigkeit misslingen
setExpression(„X", true, IntegerT, Expression („Y + 1"));
setExpression(„Y", true, IntegerT, Expression („X + 1"));

[0152] Das liegt daran, dass die an Namen gebundenen Werte Gleichungen und keine Zuordnungen sind. Wenn der Wert einer Variablen geändert wird, wird ein Verfahren zur Verfügung gestellt, um alle direkt und indirekt abhängigen Namen neu zu bewerten. Gleichungen wie beispielsweise das oben erwähnte Paar führen zu einer wiederkehrenden Abhängigkeit, die nicht erlaubt ist.

[0153] Es ist anzumerken, dass diese API eine unaufgeforderte Neuevaluierung nicht ausgesprochen unterstützt. Ein Aufruf zu setExpression(0) darf nicht automatisch bewirken, die Variable und alle anderen Variablen, die von ihr abhängig sind, neu zu bewerten. Die an alle Variablen gebundenen Werte werden unverändert bleiben, bis ein Aufruf zu reevaluateCollection() (nachstehend) vorkommt.

[0154] Ein Verfahren zum Bestimmen, wenn der spezielle Name eine Konstante ist:

Status getIsConst(const String& name, bool& isConst);

[0155] Ein Verfahren, um zu dem Typ zu gelangen:

```
enum ElementaryType
{
   UnsignedIntegerT, IntegerT, DoubleT, Volt-
ageT, ...
};
      Status getType(const String& name,
ElementaryType& elementaryType)const;
```

Das Neuevaluierungsverfahren UserVars Collection.

```
Status reevaluateCollection();
```

[0156] Die Klasse wird Gleichungen, die auf alle Variablen bezogen sind, und ihre Abhängigkeiten beibehalten. Wenn dieses Verfahren aufgerufen ist, werden alle Variablen neubewertet erhalten.

[0157] Das auf UserVars gerichtete Neuevaluierungsverfahren.
Status reevaluateTargeted(const String& var);

[0158] Die Klasse wird Gleichungen, die auf alle Variablen bezogen sind, und ihre Abhängigkeiten beibehalten. Wenn dieses Verfahren aufgerufen ist, werden die genannte Variable und alle ihre Abhängigen neubewertet erhalten.
Das Neuevaluierungsverfahren UserVars Global.
static Status reevaluateAllCollections();

[0159] Die Klasse wird Gleichungen, die auf alle Variablen bezogen sind, und ihre Abhängigkeiten beibehalten. Wenn dieses Verfahren aufgerufen ist, wird reevaluateCollection() von allen UserVars-Sammlungen in einer nicht spezifizierten Reihenfolge gefordert.

Verfahren zum Bestimmen, ob ein spezieller Name definiert ist:

```
Status getIsDefined(const String& name, bool& isDe-
fined) const;
```

Verfahren zum Bestimmen aller gegenwärtig definierten Benutzervariablen:

```
Status getNames(StringList&
names) const;
```

Verfahren zum Streichen einer gegenwärtig definierten Variablen:

```
Status deleteName(const
String& name);
```

[0160] Dieser Operation wird misslingen, wenn der Name in Ausdrücken genutzt wird, die andere Variable

enthalten.

[0161] Ein Verfahren zur Erhaltung der Liste von Variablen und Konstanten die von einer gegebenen Variablen oder Konstanten abhängig sind:
Status getDependents(const String& name, StringList& dependents);

Spezifizierungssets

[0162] Der Spezifizierungsset wird verwendet, um eine Sammlung von Variablen zu liefern, die Werte annehmen können, die auf einem Auswähler basieren. Zum Beispiel wird folgender Spezifizierungsset betrachtet, der die Auswähler Minnie, Mickey, Goofy und Daisy verwendet:

```
# ----------------------------------------
# File Aaa.spec
# ----------------------------------------

Version 1.0;

Import Limits.usrv;

SpecificationSet Aaa(Minnie, Mickey, Goofy, Daisy)
    {
      Double xxx = 1,0, 2,0, 3,0, 4,0;
      Integer yyy = 10, 20, 30, 40;
      Integer zzz = MaxInteger - xxx,
                    MaxInteger - xxx - 1,
                    MaxInteger - xxx - 2,
                    MaxInteger - xxx;

      # Die folgende Vereinbarung assoziiert einen
einzelnen
      # Wert, der ohne Rücksicht auf den Auswähler ge-
wählt
      # werden wird. Er entspricht:
      #  Integer www=yyy + zzz, yyy + zzz, yyy + zzz,
yyy + zzz
      Integer www = yyy + zzz;
    }
```

[0163] Der oben genannte Spezifizierungsset mit dem Auswähler Goofy wird die folgenden Assoziationen herstellen:
xxx = 3,0;
yyy = 30;
zzz = MaxInteger – xxx – 2;
www = yyy + zzz;

[0164] Die Operation zum Einstellen des Auswählers auf einen Spezifizierungsset wird später erörtert, wenn Tests beschrieben werden.

[0165] Syntaktisch ist ein Spezifizierungsset eine Liste von Auswählern (im oben genannten Beispiel Minnie, Mickey, Goofy und Daisy) zusammen mit einer Liste von variablen Definitionen (im oben genannten Beispiel xxx, yyy, zzz und www). Die Definition einer Variablen schließt eine Liste von Ausdrücken ein, die entweder so lang ist wie die Liste von Auswählern oder einen einzelnen Ausdruck aufweist.

[0166] Konzeptionell kann man sich einen Spezifizierungsset als eine Matrix von Ausdrücken vorstellen, deren Spalten die Auswähler sind, deren Zeilen die Variablen und deren Eingaben Ausdrücke sind. Ein spezieller Auswähler (Spalte) bindet jede Variable (Zeile) an einen spezifischen Ausdruck (Eingabe). Wenn die Liste einen einzelnen Ausdruck aufweist, stellt sie eine Zeile mit dem Ausdruck dar, der so viele Male reproduziert wird wie Auswähler vorhanden sind.

[0167] Spezifizierungssets können in zwei getrennten Zusammenhängen erscheinen. Sie könnten getrennt in einer .spec-Datei vereinbart werden, wobei sie in diesem Fall wie oben gezeigt erscheinen. Diese sind nominierte Spezifizierungssets. Andererseits können lokale Spezifizierungssets innerhalb einer Testbedingungsgruppe vereinbart werden. In einer solchen Vereinbarung wird der Spezifizierungsset nicht mit einem Namen versehen sein. Er wird ein lokaler Spezifizierungsset sein, der nur für die umfassende Testbedingungsgruppe Bedeutung hat.

[0168] Nominierte Spezifizierungssets können nach der Sammlung nominierter Benutzervariablen gestaltet werden. Der oben genannte Spezifizierungsset kann als eine Aaa genannte UserVars-Sammlung gestaltet werden, die Ausdrücke für xxx [Minnie], xxx [Mickey], xxx [Goofy], xxx [Daisy], yyy [Minnie] und so weiter aufweisen wird. Wenn ein spezieller Auswähler (sagen wir Mickey) im Zusammenhang mit einem Test gewählt ist, werden die Werte von xxx, yyy und zzz aus dem Namen der Variablen und dem Namen des Spezifizierungssets erhalten.

[0169] Eine Testbedingungsgruppe kann höchstens einen Spezifizierungsset besitzen, der entweder ein lokaler Spezifizierungsset oder ein Verweis auf einen nominierten Spezifizierungsset ist. Lokale Spezifizierungssets erscheinen nur im Zusammenhang mit einer Testbedingungsgruppe und haben keinen explizit festgelegten Namen. Ein solcher Spezifizierungsset besitzt einen impliziten Namen, der durch den Namen der umfassenden Testbedingungsgruppe definiert ist. Um einen Namen in einer Testbedingungsgruppe an einem Punkt aufzulösen, an dem mehrere Spezifizierungssets und mehrere UserVars-Sammlungen sichtbar sind, werden die folgenden Regeln angewandt:

1. Wenn der Name qualifiziert ist, muss er in einer Sammlung nominierter Benutzervariablen aufgelöst werden.
2. Wenn der Name nicht qualifiziert ist, wird der Name entweder in einem lokalen Spezifizierungsset, wenn es in der Testbedingungsgruppe vereinbart ist, oder in dem nominierten Spezifizierungsset aufgelöst, wenn auf einen in der Testbedingungsgruppe hingewiesen ist.
3. Wenn der Name nicht durch die früheren Regeln aufgelöst ist, wird er in der Sammlung vorgegebener Benutzervariablen aufgelöst.

[0170] Um diese Regeln zur veranschaulichen, betrachten wir das folgende Beispiel, das Testbedingungsgruppen (die später beschrieben werden) verwendet.

```
            Version 1.2.3;
            Import limits.usrv; # Nimmt die oben ge-
nannte Datei
                                    Grenzen UserVars
auf.
```

```
            Import aaa.spec;     # Nimmt den oben ge-
nannten
                                 Spezifizierungsset AAA
auf.

            TestConditioningGroup TCG1
            {
              SpecificationSet(Min, Max, Typ)
              {
                  vcc = 4,9, 5,1, 5,0;
              }
              # Regel 1: Auflösung in einer Sammlung
              # nominierter Benutzervariablen.
              # Ein Verweis auf MyVars.VInLow bezieht
sich auf
              # VInLow von MyVars.

              # Regel 2: Auflösung in einem lokalen
              # Spezifizierungsset.
              # Ein Verweis auf „vcc" wird sich hier
im
              # Zusammenhang des oben genannten loka-
len
              # Spezifizierungssets auflösen.

              # Regel 3: Auflösung in Sammlung vorge-
gebener
              # Benutzervariablen.
              # Ein Verweis auf „MaxInteger" wird sich
hier zu
              # limits.usrv auflösen.

              # Fehler: Auflösung von xxx
              # Ein Verweis auf xxx löst sich nicht
auf, weil
```

```
            # er sich weder im lokalen Spezifizie-
rungsset
            # noch in limits.usrv befindet.

            # Fehler: Auflösung von Aaa.xxx
            # Sieht nach einer nominierten UserVars-
Sammlung,
            # die mit Aaa benannt ist. Der benannte
            # Spezifizierungsset qualifiziert sich
nicht.
         }
```

Testbedingungsgruppe TCG2

```
   {
      SpecificationSet Aaa; # Verweist auf den impor-
tierten
                            # Spezifizierungsset

      # Regel 1: Auflösung in einer Sammlung nomi-
nierter
      # Benutzervariablen.
      # Ein Verweis auf MyVars.VInLow bezieht sich
auf VInLow
      # von MyVars.

      # Regel 2: Auflösung in einem nominierten
      # Spezifizierungsset.
      # Ein Verweis auf „xxx" wird sich hier im Zu-
sammenhang
      # mit dem oben genannten lokalen Spezifizie-
rungsset Aaa
      # auflösen.

      # Regel 3: Auflösung in der Sammlung vorgegebe-
ner
```

```
      # Benutzervariablen.
      # Ein Verweis auf „MaxInteger" wird sich hier
zu
      # limits.usrv auflösen.

      # Fehler: Auflösung von vcc.
      # Ein Verweis auf vcc löst sich nicht auf, weil
er sich
      # weder im nominierten Spezifizierungsset Aaa
noch in
      # limits.usrv. befindet.
      # Fehler: Auflösung von Aaa.xxx
      # Sieht nach einer Sammlung nominierter User-
Vars, die
      # Aaa genannt wird. Der nominierte Spezifizie-
rungsset
      # qualifiziert sich nicht.

   }
```

[0171] Die Auflösung eines Namens in einem Spezifizierungsset (Regel oben) erfordert, dass ein Auswähler des Satzes zu dem Zeitpunkt ermöglicht wird, wo die Namensauflösung benötigt wird. Dies wird durch die Tatsache verstärkt, dass auf die Testbedingungsgruppe in einem Test durch Spezifizieren eines Auswählers verwiesen wird.

C++- für Spezifizierungssets

[0172] Unter Nutzung oben genannter Regeln können Spezifizierungssets durch die C++-Spezifizierungsset-Klasse implementiert werden. Die Spezifizierungsset-Klasse besitzt im Wesentlichen die gleiche API wie die UserVars-Klasse, abgesehen von einem zusätzlichen Sequenzparameter für den Auswähler. Folglich wird diese API nicht ausführlich beschrieben.

[0173] Alle nominierten Spezifizierungssets werden vorzugsweise mit einem C++-Objekt dieses Namens verknüpft. Ein lokaler Spezifizierungsset wird im Kontext mit einer Testbedingungsgruppe einen Namen besitzen, der für diese Testbedingungsgruppe eindeutig ist. Es ist verboten, auf eine Variable eines lokalen Spezifizierungssets außerhalb des Kontextes der Testbedingungsgruppe, in der sie definiert ist, zu verweisen.

Ebenen

[0174] Die Ebenen werden genutzt, um Parameter von Pins und Pingruppen zu spezifizieren. Es ist eine Sammlung von Vereinbarungen der Form:

```
<pin-or-pin-group-name>
{
   <pin-param-1> = xxx;
   <pin-param-2> = yyy;
  ...
 }
```

[0175] Eine solche Vereinbarung legt die Einstellung der verschiedenen Parameter des nominierten Pins oder der nominierten Pingruppe fest. Zum Beispiel könnte eine solche Anweisung genutzt werden, um die VIL-Werte für alle Pins in der InputPins-Gruppe zu setzen, wie es im nachstehenden Beispiel dargestellt ist:

```
# -------------------------
# File pentiumlevels.lvl
# -------------------------

Version 1.0;
```

```
   Import pentium3resources.rsc;
   Import pentium3pins.pin

   Ebenen Pentium3Levels
   {
       # Spezifiziert Pinparameter für
       # verschiedene Pins und Pin-
gruppen unter
       # Verwendung von Globals und
Werten aus
       # dem Spezifizierungsset.
       #
       # Die Spezifizierungsreihenfol-
ge ist
         signifikant.
       # Pinparameter werden in einer
       # Reihenfolge vom ersten zum
letzten in
       # diesem Ebenenabschnitt und
vom ersten
       # zum letzten für jeden Pin o-
der jeden
       # Pingruppen-Unterabschnitt ge-
setzt.
       #
       # Aus der importierten Pin-
beschreibungs-
       # datei pentium3pins.pin ist
die InPins-
       # Gruppe in der „dpin"-
Ressource. Aus der
       # importierten Ressourcendefi-
nitionsdatei
       # pentium3resources.rsc besitzt
die
```

```
        # „dps"-Ressource VIL und VIH
benannte
        # Parameter.
        #
        InPins {VIL = v_il; VIH = v_ih
+ 1,0;}

        # Die folgende Anweisung erfor-
dert eine
        # Verzögerung von 10µs nach dem
Aufruf,
        # die Ebene InPins zu setzen.
Die
        # wirkliche Verzögerung wird
ein kleiner
        # systemdefinierter Bereich um
10,0E-6
        # sein:
        Delay 10,0E-6;

        #
        # Für die OutPins werden die
Ebenen für
        # die Parameter VOL und VOH
spezifiziert.
        #
        OutPins {VOL = v_ol/2,0; VOH =
v_oh;}

        # Der Clock Pin wird spezielle
Werte
        # besitzen.
        Clock {VOL = 0,0; VOH =
v_ih/2,0;}
```

```
        # Verzögerung von 10 µs nach
dem Aufruf,
          Taktebenen zu setzen.
        # Dies ist eine Mindestverzöge-
rung, die
        # für mindestens 10,0 µs garan-
tiert ist,
        # obwohl sie etwas mehr sein
kann:
        #  10,0E-6 <= wirklich <=
10,0E-6 + Delta
        MinDelay 10,0 µs;

        #
        # Die PowerPins-Gruppe liegt in
der
        # „dps"-Ressource. Pins dieser
Pingruppe
        # weisen spezielle Parameter
auf:
      # PRE_WAIT bestimmt die Zeit zu
warten,
      # nachdem Spannung ihren Endwert
erreicht
      # hat, um Strukturerzeugung zu
starten.
      # Die aktuelle Wartezeit wird
ein kleiner
      # systemdefinierter Bereich um
PRE_WAIT
      # sein (siehe da)
      # POST_WAIT bestimmt die Zeit zu
warten,
      # nachdem Strukturerzeugung en-
det, um den
```

```
                # Strom abzuschalten. Tatsächli-
            che
                # Wartezeit wird ein kleiner
            system-
                # definierter Bereich um
            PRE_WAIT sein
                  (siehe da).
                #
                PowerPins
                {
                  PRE_WAIT = 10,0 ms;
                  POST_WAIT = 10,0 ms;

                  # VCC erreicht seinen Endwert
von 2,0 V
                  # aus seinem gegenwärtigen
Wert in
                  # einem Anstiegsvorgang mit
einer
                  # Spannungsanstiegs-
geschwindigkeit von
                  # ±0,01 Volt je Sekunde.
                  VCC = Anstieg(0,01, 2,0 V);
                }
            }

            Ebenen Pentium4Levels
            {
              # ...
            }
```

[0176] Wie oben ersichtlich ist, wird jeder Ebenenblock vorzugsweise aus einer Anzahl von Ebenen-Datenelementen zusammengesetzt, von denen jedes die Parameter für einen Pin oder eine Pingruppe bestimmt. Jedes Ebenen-Datenelement kann eine Anzahl von Ressourcen-Parametern bestimmen. Die Laufzeit-Semantik zum Setzen dieser Ebenenwerte ist wie folgt:
Die Ebenen-Datenelemente des Ebenenblocks werden in einer Vereinbarungsreihenfolge verarbeitet. Jeder Pin, der in mehr als einem Ebenen-Datenelement auftritt, wird erreichen, mehrere Male verarbeitet zu werden. Eine Mehrfachspezifikation von Werten für einen einzelnen Parameter sollte beibehalten und in Spezifizierungsreihenfolge angewendet werden.

[0177] Die Ressourcen-Parameter in einem Ebenen-Datenelement werden in der Reihenfolge verarbeitet wie sie bestimmt wurden.

[0178] Die Delay-Anweisungen bewirken, dass der Prozess zum Setzen von Ebenen ungefähr die angegebene Dauer lang unterbrochen wird, bevor die nächste Gruppe von Ebenen gesetzt wird. Die tatsächliche War-

tezeit kann in einem kleinen systemdefinierten Bereich um die bestimmte Verzögerung herum liegen. So würde die tatsächliche Verzögerung, wenn die Verzögerung t Sekunden wäre,

$t - \Delta t <= \text{actual-wait} <= t + \Delta t$

erfüllen.

[0179] Die Delay-Anweisungen teilen die Ebenen-Spezifikation in eine Anzahl von Teilfolgen auf, von denen jede zum Verarbeiten getrennte Einstellungen von Test Condition Memory erforderlich machen wird.

[0180] Die MinDelay-Anweisungen bewirken, dass der Prozess zum Setzen von Ebenen mindestens die bestimmte Dauer lang unterbrochen wird, bevor die nächste Gruppe von Ebenen gesetzt wird. Die tatsächliche Wartezeit kann in einem kleinen systemdefinierten Bereich mit einem Mindestwert der bestimmten Mindestverzögerung liegen. So würde die tatsächliche Verzögerung, wenn die Verzögerung t Sekunden wäre,

$t <= \text{actual-wait} <= t + \Delta t$

erfüllen.

[0181] Die MinDelay-Anweisungen teilen die Ebenen-Spezifikation in eine Anzahl von Teilfolgen auf, von denen jede zum Verarbeiten getrennte Einstellungen von Test Condition Memory erforderlich machen wird.

[0182] Jeder Pinname oder Pingruppenname ist in exakt einer Ressource in einer Pinbeschreibungsdatei (suffix.pin) festgelegt und besitzt deshalb eine bestimmte Menge von existenzfähigen Ressourcen-Parametern, die in der Ressourcen-Datei (suffix.rsc) festgelegt sind. Alle nominierten Parameter müssen unter dieser Menge existenzfähiger Ressourcen-Parameter sein, und müssen vom gleichen elementaren Typ wie der zum Setzen ihres Wertes verwendete Ausdruck sein. Informationen über die Namen und Typen von Ressourcen-Parametern kommen aus der Ressourcen-Datei.

[0183] Die Ressourcen-Datei Resources.rsc wird impliziert importiert, indem der Tester mit den Namen und Typen für Parameter von Standardressourcen wie beispielsweise dpin und dps versehen wird.

[0184] Ressourcen-Parameter sind zugewiesene Ausdrücke, die UserVars verwenden können, und Werte nominierter Spezifizierungssets oder ein gegenwärtig sichtbarer lokaler Spezifizierungsset.

[0185] Dps Pin-Ressourcen besitzen spezielle Parameter PRE_WAIT und POST_WAIT. Der PRE_WAIT Parameter legt die Zeit fest, die von dem Zeitpunkt verstreichen muss, an dem der Leistungspin seine Zielspannung erreicht, bis zu dem Zeitpunkt, an dem die Strukturerzeugung beginnen kann. Der POST_WAIT Parameter legt die Zeit fest, die von dem Zeitpunkt verstreichen muss, an dem die Strukturerzeugung unterbrochen ist, bis zu dem Zeitpunkt, an dem der Leistungspin abschaltet.

[0186] Dps Pins bestimmen außerdem, wie der Spannungsparameter seinen endgültigen Wert erreicht. Sie könnten ihn einfach durch eine Gleichung wie alle anderen Pinparameter bestimmen. In dem Fall wird der Wert erreicht werden, wie es die Hardware erlaubt. Sie könnten ihn auch festlegen, indem eine Anstiegsanweisung verwendet wird. Eine Anstiegsanweisung bestimmt, dass die Spannung der Stromversorgung ihren endgültigen Wert von dem Anfangswert in einem Anstiegsvorgang mit einer bestimmten absoluten Spannungsanstiegsgeschwindigkeit erreicht.

C++ für Ebenen

[0187] Mit den oben erwähnten Regeln kann eine C++ Ebenen-Objekt geschrieben werden, welches die folgenden Operationen unterstützt:

```
Es gibt eine Operation

        Status setParameter (const String& pinOrP-
inGroup Name,
                                const String& parame-
terName,
                                ElementaryType ele-
mentaryType,
                                Const Expression& Ex-
pression);
```

[0188] Diese Operation bindet einen Ausdruck an einen Parameter eines Pins oder einer Pingruppe. Zum Beispiel wird der dpin.InPins VIH Wert gesetzt durch:

```
setParameter („InPins", „VIH", Spannung T,
                Ausdruck(„v_ih + 1,0");
```

[0189] Diese Operation wird mehrere Male für alle Vereinbarungen in dem Ebenenobjekt aufgerufen.

```
Es gibt eine Operation

                Status assignLevels(const String&
selector),
```

die durchlaufen wird und alle vorgegebenen Modulebenen-Schnittstellen ausgeben wird, um alle Ebenen von Parametern in Spezifizierungsreihenfolge zuzuweisen wie es früher beschrieben wurde. Der Auswählerparameter wird verwendet, um Namen in den Ausdrücken entsprechend den früher festgelegten Regeln aufzulösen.

Testbedingungsgruppen

[0190] Die Testbedingungsgruppen-Untersprache packt die Beschreibung von Spezifikationen, Taktungen und Ebenen zusammen. Oft werden Taktungsobjekte unter Verwendung von Parametern bestimmt. Parameter können in Taktspannungen verwendet werden, um die Vorderflanke und Hinterflanke von verschiedenen Impulsen zu bestimmen. Ebenso können Ebenen parametriert werden, indem maximale, minimale und typische Werte von verschiedenen Spannungspegeln im Einzelnen festgelegt werden. Ein Testbedingungsgruppenobjekt (TCG) fasst die Spezifikationen und die konkrete Darstellung von zeitlichen Zuordnungen und Ebenen zusammen, die auf diesen Spezifikationen basieren.

[0191] Eine TestConditionGroup-Vereinbarung enthält einen optionalen Spezifizierungsset. Die Spezifizierungsset-Vereinbarung kann ein mitlaufender (und nicht nominierter) lokaler Spezifizierungsset oder ein Bezug auf einen nominierten Spezifizierungsset sein, der anderswo vereinbart wurde. Die optionale Spezifizierungsset-Vereinbarung in einer TCG Vereinbarung schließt sich mindestens an eine Ebenen- oder Taktungsvereinbarung an. Sie kann sowohl Ebenen als auch zeitliche Zuordnungen in beliebiger Reihenfolge besitzen. Ihr ist es jedoch nicht erlaubt, mehr als eine Ebenen- und Taktungsvereinbarung zu besitzen. Diese Einschränkungen werden syntaktisch verstärkt.

[0192] Eine Vereinbarung des Spezifizierungssets in einer TCG ist identisch mit dem getrennt vereinbarten Spezifizierungsset mit der Ausnahme, dass er keinen Namen besitzt. Sein Name ist implizit der Name der einschließenden TCG. Die Vereinbarung der zeitlichen Zuordnungen umfasst eine einzelne Vereinbarung eines Objektes zeitlicher Zuordnungen von einer bestimmten Taktungsdatei. Hier ist das Beispiel einer Datei mit einer Testbedingungsgruppe:

```
# --------------------------------
# File myTestConditionGroups.tcg
# --------------------------------

Version 0,1;

Import pentiumlevels.lvl;
```

```
Import edges.spec;
Import timing1.tim;
Import timing2.tim;

TestConditionGroup TCG1
{
  # Dieser lokale Spezifizierungsset verwendet
  # anwenderdefinierte Auswähler „min", „max" und
  # „typ". Eine beliebige Anzahl von Auswählern mit
  # beliebigen anwenderdefinierten Namen ist
  # erlaubt.
  #
  # Der Spezifizierungsset bestimmt eine Tabelle,
  # die Werte für Variable angibt, die in Ausdrücken
  # zum Initialisieren von zeitlichen Zuordnungen
  # und Ebenen verwendet werden kann. Der unten
  # stehende Spezifizierungsset definiert Werte für
  # Variable wie nach der folgenden Tabelle:
  #   min      max     typ
  # v_cc  2,9    3,1     3,0
  # v_ih vInHigh + 0,0  vInHigh + 0,2 vInHigh + 0,1
  # v_i1 vInLow + 0,0  vInLow + 0,2  vInLow + 0,1
  # ...
```

```
   # Ein Bezug wie "vInHigh" muss vorher in
einem
   # Block von UserVars definiert werden.
   #
   # Folglich würde, falls in einem Funkti-
onstest der
   # „max" Auswähler ausgewählt wäre die
„max" Spalte
   # von Werten an die Variablen gebunden
sein, indem
   # v_cc auf 3,1, v_ih auf vInHigh+2,0 und
so weiter
   # gesetzt wird.
   #
   # Zu beachten ist, dass dies ein lokaler
   # Spezifizierungsset ist, der keinen Na-
men
   # besitzt.
   SpecificationSet(min, max, typ)
   {
     # Minimale, maximale und typische
     # Spezifikationen für Spannungen.
     Spannung v_cc = 2,9, 3,1, 3,0;
     Spannung v_ih = vInHigh + 0,2
                     vInHigh + 0,1;
     Spannung v_il = vInLow + 0,0
                     vInLow + 0,2
                     vnLow + 0,1;

     # Minimale, maximale und typische
     # Spezifikationen für Vorderflanke und
     # Hinterflanke zeitlicher Zuordnungen.
Der
     # Basiswert von 1,0E-6 uS entspricht 1
     # Picosekunde und ist als ein Beispiel
der
```

```
                # Nutzung von wissenschaftlicher
Schreibweise
                # für Zahlen zusammen mit Einheiten
gegeben.
                Zeit t_1e = 1,0E-6 uS,
                            1,0E-6 uS + 4,0 * ΔT,
                            1,0E-6 uS + 2,0 * ΔT;
                Zeit t_te = 30 ns,
                            30 ns + 4,0 * ΔT,
                            30 ns + 2,0 * ΔT;

            }

            # Verweist auf die früher importierten
            # Pentium3Level. Es ist eines von mögli-
cherweise
            # vielen Ebenenobjekten, die aus der oben
erwähnten
            # Datei importiert wurden.
            Ebenen Pentium3Levels;

            # Verweist auf Datei timing1.tim, welche
die
            # einzelne zeitliche Zuordnung Timing1
enthält. Der
            # Dateiname sollte quotiert werden, falls
er in
            # sich Lückenzeichen aufweist.
            Zeitliche Zuordnungen Timing1;
      }

      # Eine weitere Testbedingungsgruppe
      TestConditioningGroup TCG2
      {
       # ClockAndDataEdgesSpecs ist ein Spezifizie-
rungsset,
```

```
      # der in der Datei edges.specs vorhanden ist.
      # Es wird angenommen, dass die folgende Verein-
barung
      # aufweist:
      # SpecificationSet ClockAndDataEdgesSpecs(min,
max, typ)
      #  {
      #      Time clock_le = 10,00 uS, 10.02 uS,
10,01 uS;
      #      Time clock_te = 20,00 uS, 20,02 uS,
20,01 uS;
      #      Time data_le = 10,0 uS, 10.2 uS, 10,1
uS;
      #      Time data_te = 30,0 uS, 30,2 uS, 30,1
uS;
      #  }
      # Nachstehend ein Bezug des Spezifizierungssets
auf
      # diesen nominierten Satz:
      SpecificationSet ClockAndDataEdgesSpecs;

      # eine mitlaufende Ebenenvereinbarung. Weil der
      # zugeordnete Spezifizierungsset (oben) keine
Variablen
      # wie beispielsweise VInLow, VInHigh, VOutLow
und
      # VOutHigh aufweist, müssen sie sich in der
vorgegebenen
      # Sammlung UserVars auflösen.
      Ebenen
      {
         InPins { VIL = VInLow; VIH = VInHigh + 1,0;}
         OutPins { VOL = VOutLow / 2,0; VOH =
VOutHigh; }
      }
```

```
      # Diese zeitliche Zuordnung ist aus der Datei
      # „timing2.tim". Die zeitlichen Zuordnungen
werden die
      # Vorderflanke und Hinterflanke zeitlicher Zu-
ordnungen
      # für Taktimpuls und Daten, wie in dem oben er-
wähnten
      # Spezifizierungsset festgelegt, benötigen.
      Zeitliche Zuordnungen Timing2;
   }
```

[0193] Im oben erwähnten Beispiel beschreibt die Testbedingungsgruppe TCG1 einen Spezifizierungsset mit drei Auswählern, die „min", „typ" und „max". benannt sind. Es kann eine beliebige Anzahl von charakteristischen Auswählern vorhanden sein. Im Hauptteil des Spezifizierungssets werden die Variablen v_il, v_ih, t_le und t_te mit dem Dreifachen von Werten, die den Auswählern entsprechen, initialisiert. So wird im oben erwähnten Beispiel ein Fall von TCG1 mit dem Auswähler „min" die Variable v_il mit dem ersten Zahlenwert (vInputLow + 0,0) binden. Er bringt die Wiederholung hervor, dass die Auswähler für einen Spezifizierungsset anwenderdefiniert sind und eine beliebige Anzahl von ihnen erlaubt ist. Die einzige Forderung ist, dass:
Die Auswähler eines Spezifizierungssets eindeutig bestimmte Identifizierer sind.

[0194] Jeder in dem Spezifizierungsset festgelegte Wert ist mit einer Gruppe von Werten verknüpft, die exakt die gleiche Anzahl von Elementen wie der Satz von Auswählern ist. Den i-ten Auswähler aufzunehmen wird bewirken, dass jeder Wert an den i-ten Wert seines zugeordneten Wertevektors gebunden ist.

[0195] Im Anschluss an den Spezifizierungsset in der TCG könnte es eine Ebenenvereinbarung oder eine Taktungsvereinbarung oder beides geben. Die Ebenenvereinbarung wird verwendet, um Ebenen für verschiedene Pinparameter zu setzen. Die in dem Spezifizierungsset identifizierten Variablen werden verwendet, um diese Ebenen zu setzen, was die dynamische Bindung unterschiedlicher aktueller Werte für Pinparameter auf der Basis des zum Initialisieren der TCG genutzten Auswählers erlaubt.

[0196] Um dies zu veranschaulichen, betrachten wir einen Test, der den Auswähler „min" aktiviert. Mit Bezug auf den auf der Seite angegebenen Spezifizierungsset Pentium3Levels werden der Pinparameter „VIH" für Pins in der InPins Gruppe durch die Vereinbarung:

```
            InPins { VIL = v_il; VIH = v_ih + 1,0 ;
}


zu dem Ausdruck (v_ih + 1,0) vorbereitet.
```

[0197] Dieser löst sich auf zu (VInHigh + 0,0 + 1,0), wenn der Auswähler „min" aktiviert ist. Ebenso kann das Taktungsobjekt basierend auf den ausgewählten Werten der Variablen des Spezifizierungssets ausgewählt werden. Es ist nicht nötig, sowohl eine Taktungsvereinbarung als auch eine Ebenenvereinbarung zu haben. Jede kann durch sich selbst oder beide in einer beliebigen Reihenfolge vorhanden sein wie es durch das folgende Beispiel dargestellt ist:

```
# ------------------------------------
# Datei LevelsOnlyAndTimingsOnly.tcg
# ------------------------------------
```

```
            Version 0.1;

            # Eine Testbedingungsgruppe „Nur Ebenen".
            TestConditionGroup LevelsOnlyTCG
            {
               SpecificationSet(Min, Max, Typ)
               {
                  Spannung v_il = 0,0, 0,2, 0,1;
                  Spannung v_ih = 3,9, 4,1, 4,0;
               }

           # Eine mitlaufende Ebenenvereinbarung.
Weil der
           # verknüpfte Spezifizierungsset (oben)
keine
           # Variablen wie VInLow, VInHigh, VOutLow
und
           # VOutHigh besitzt, müssen sie sich in der
           # vorgegebenen Sammlung UserVars auflösen.
           Ebenen
           {
              InPins { VIL = v_il; VIH = v_ih + 1,0;
}
              OutPins { VOL = v_il / 2,0; VOH =
v_ih ; }
           }
         }
         # Eine Testbedingungsgruppe Nur Taktungen
         TestConditionGroup TimingsOnlyTCG
         {
            SpecificationSet (Min, Max, Typ)
            {
               Zeit t_le = 0,9E-3, 1,1E-3, 1,0E-3;
            }

            Zeitliche Zuordnungen Timing2

}
```

[0198] Es ist jedoch zu beachten, dass in einer TCG nicht mehr als eine zeitliche Zuordnung und mehr als eine Ebene vorhanden sein sollte. So sollten insgesamt von zeitlichen Zuordnungen oder Ebenen mindestens eine und höchstens eine von jeder vorhanden sein.

Testbedingungen

[0199] Ein Testbedingungs-Objekt bindet eine TCG an einen spezifischen Auswähler. Sobald eine TCG, wie oben gezeigt, vereinbart wurde, ist es möglich, Testbedingungs-Objekte wie nachstehend gezeigt zu vereinbaren:

```
TestCondition TCMin
{
     TestConditionGroup = TCG1;
     Auswähler = min;
}

TestCondition TCTyp
{
     TestConditionGroup = TCG1;
     Auswähler = Typ;
}

TestCondition TCMax
{
     TestConditionGroup = TCG1;
     Auswähler = max;
}
```

[0200] Diese Testbedingungen würden in einem Testplan konkret wie folgt dargestellt werden:

```
#
# Vereinbare einen Funktionstest
              # „MyFunctionalTest", der sich auf drei
Fälle
              # von Testbedingungsgruppen bezieht.
              #
              # Test Funktionstest MyFunctionalTest
              {
                # Bestimme die Strukturliste
                PList = pat1Alist;
                # Es kann eine beliebige Anzahl von
                # Testbedingungen bestimmt werden:
                TestCondition = TCMin;
                TestCondition = TCMax;
                TestCondition = TCTyp;
               }
```

Namensauflösung in TCG (Testbedingungsgruppen)

[0201] Die Auflösung von Namen in einer Testbedingungsgruppe wurde früher erörtert. Jedoch bringen diese Regeln Wiederholung hervor und sind nachstehend wiederum angegeben:

1. Wenn der Name qualifiziert ist (siehe Seite), muss er in einer Sammlung nominierter Benutzervariablen aufgelöst sein.
2. Wenn der Name nicht qualifiziert ist, wird der Name aufgelöst entweder in einem lokalen Spezifizierungsset, wenn er in der Testbedingungsgruppe vereinbart ist, oder in dem nominierten Spezifizierungsset, wenn auf einen in der Testbedingungsgruppe verwiesen wird.
3. Falls der Name nicht durch die früheren Regeln aufgelöst ist, wird er in der Sammlung vorgegebener Benutzervariablen aufgelöst.

TCG Laufzeit

[0202] Testbedingungsgruppen weisen die folgende Laufzeitsemantik auf:
Ein Test (wie ein Funktionstest) wird sich auf eine TCG mit einem speziellen Auswähler aus einem Spezifizierungsset beziehen, indem eine konkret dargestellte Testbedingung verwendet wird. Dieser Auswähler wird jede Variable in dem Spezifizierungsset an ihren mit dem gewählten Auswähler verknüpften Wert binden. Diese Bindung von Variablen an ihre Werte wird dann genutzt, um Ebenen und zeitliche Zuordnungen zu bestimmen.

[0203] Parameter-Ebenen in einer Testbedingungsgruppe werden vorzugsweise aufeinander folgend, in der Darstellungsreihenfolge in den Ebenenblöcken gesetzt. So ist im Block Pentium3Level die Reihenfolge, in der Parameterebenen gesetzt werden würden, wie folgt
(Schreibweise: <resource-name>.<resource-parameter>):
InputPins.VIL,
InputPins.VIH
OutputPins.VIL,
OutputPins.VIH,
Clock.VOL,
Clock.VOH.

[0204] Diese Reihenfolge ermöglicht dem Testschreiber, die explizite Leistungsfolgesteuerung von Stromversorgungen zu regeln. Wenn eine Ebene zweimal auftritt, welche die gleichen Pinparameter für einen Pin benennt, dann kommt außerdem dieser Pinparameter dazu, zweimal gesetzt zu werden. Dies kann auch programmatisch passieren.

[0205] Wenn ein Parameter durch eine Anstiegsanweisung wie

VCC = Anstieg (0,01, 2,0 V);

gesetzt wird, bedeutet dies, dass VCC seinen Endwert von 2,0 Volt aus seinem gegenwärtigen Wert in einem Anstiegsvorgang mit einer Spannungsanstiegsgeschwindigkeit von ±0,01 Volt pro Sekunde erreichen wird.

[0206] Variable des Spezifizierungssets können auch in ein Taktungsobjekt in der TCG weitergegeben werden. Das Taktungsobjekt wird anschließend auf der Basis der ausgewählten Variablen vorbereitet. Ein solcher Mechanismus könnte genutzt werden, um ein Taktungsobjekt für einen bestimmten Anwendungsfall wie zum Beispiel dadurch auszulegen, dass Vorder- und Hinterflanke von Wellenformen im Einzelnen festgelegt werden.

C++ für TCGs

[0207] Mit den oben erwähnten Regeln kann die Testbedingungsgruppe in einer C++-Klasse von Testbedingungsgruppen vereinbart werden und ihre Vorbereitung ist wie folgt:
Es wird ein Aufruf an die Elementfunktion der Testbedingungsgruppe vorgenommen
Status setSpecificationSet(SpecificationSet *pSpecificationSet);
der den Spezifizierungsset für die Testbedingungsgruppe setzen wird. Dieser kann entweder ein lokaler Spezifizierungsset oder ein nominierter Spezifizierungsset oder Null (wenn es keinen gibt) sein.

[0208] Es wird ein Aufruf an die Elementfunktion der Testbedingungsgruppe vorgenommen

Status setLevels(Levels *pLevels);
der das Ebenenobjekt für die Testbedingungsgruppe setzen wird. Dieser kann entweder ein lokal vereinbartes Ebenenobjekt oder ein extern vereinbartes Ebenenobjekt oder Null (wenn es keines gibt) sein.

[0209] Es wird ein Aufruf an die Elementfunktion der Testbedingungsgruppe vorgenommen
Status setTimings(Timings *pTimings);
der das Ebenenobjekt für die Testbedingungsgruppe setzen wird. Dieser kann entweder ein extern vereinbartes Ebenenobjekt oder Null (wenn es keines gibt) sein.

Binärdateidefinitionen

[0210] Die Klasse Binärdateidefinitionen definiert Binärdateien, eine Sammlung von Zählern, die die Ergebnisse der Prüfung vieler DUT (Prüfobjekte) zusammenfasst. Im Verlauf der Prüfung eines DUT kann das DUT auf eine beliebige Binärdatei gesetzt werden, um z. B. das Ergebnis eines speziellen Tests anzuzeigen. Wenn die Prüfung fortschreitet, kann das DUT auf eine andere Binärdatei gesetzt werden. Die Binärdatei, auf die das DUT schließlich gesetzt wird, ist eine letzte solche Einstellung am Ende des Tests. Der Zähler für diese letzte Binärdatei wird am Ende des Tests dieses DUT erhöht. Eine getrennte Datei mit Binärdateidefinitionen sollte die Nachsilbe .bdefs haben.

[0211] Binärdatei-Definitionen sind vorzugsweise hierarchisch. Auf einer äußersten Ebene können zum Beispiel die PassFailBins mit zwei Binärdateien vorhanden sein, die Pass und Fail genannt werden. Dann könnten mehrere HardBins vorhanden sein, von denen sich einige auf die Binärdatei Pass abbilden, und andere, die sich auf die Binärdatei Fail abbilden. Es heißt, die HardBins seien eine Verfeinerung der PassFailBins. Schließlich könnte eine große Anzahl von SoftBins, eine Verfeinerung von HardBins vorhanden sein, von denen sich viele auf die gleiche Hard-Binärdatei abbilden. Nachstehend ist ein Beispiel, das die Hierarchie von Binärdateien darstellt:

```
# ------------------------------
# File pentiumbins.bdef
# ------------------------------

Version 1.2.3;

BinDefs
{
  # Die HardBins sind eine äußerste Ebene von
  # Binärdateien. Sie sind keine Verfeinerung
  # irgendwelcher anderer Binärdateien.
  BinGroup HardBins
```

```
        {
          „3GHzPass“:  „DUT“ bestehen bei 3 GHz“,
          „2,8GHzPass“: „DUT bestehen bei 2,8 GHz“,
          „3GHzFail“:   „DUT versagen bei 3 GHz“,
          „2,8 GHzFail“: „DUT versagen bei 2,8 GHz“,
           LeakageFail: „DUT versagen bei Streuver-
lust“,
        }

        # Die SoftBins sind ein nächstes Verfeine-
rungsniveau.
        # SoftBins sind eine Verfeinerung von Hard-
Bins.
        # BinGroup SoftBins: HardBins
        {
          „3GHzAllPass“:
                „Gute DUT bei 3 GHz“, „3GHzPass“,
          „3GHzCacheFail“:
                „Cachespeicher versagt bei 3 GHz“,
„3GHzFail“,
          „3GHzSBFTFail“:
                   „Funktionstest Soft-Binärdatei
versagt bei
                                              3 GHz“,
„3GHzFail“,
          „3GHzLeakage“:
                  „Streuverluste bei 3 GHz“, Leakage-
Fail,
          „2,8GHzAllPass“:
                  „Gute DUTs bei 2,8 GHz“,
„2,8GHzPass“,
          „2,8GHzCacheFail“,
            „Cachespeicher versagt bei 2,8 GHz“,
„2,8GHzFail“,
          „2,8GHzSBFTFail“:
```

```
                „ Funktionstest Soft-Binärdatei ver-
sagt bei
                                          2,8 GHz",
„2,8GHzFail",
          „2,8GHzLeakage",
               „Streuverluste bei 2,8 GHz", Leakage-
Fail,

        }
      }
```

[0212] Im oben erwähnten Beispiel sind die meisten Basiswert-Binärdateien Binärdateigruppen-HardBins. Es heißt, eine Binärdateigruppe X sei eine Gruppe von Basiswert-Binärdateien, wenn eine bestimmte andere Binärdateigruppe eine Verfeinerung von X ist. Folglich sind Binärdateigruppen-HardBins eine Gruppe von Basiswert-Binärdateien, weil die Binärdateigruppe SoftBins eine Verfeinerung von HardBins ist. Die Binärdateien werden als Knoten-Binärdateien bezeichnet. Es heißt, eine Binärdateigruppe Y ist eine Gruppe von Knoten-Binärdateien, falls keine andere Binärdateigruppe eine Verfeinerung von Y ist.

[0213] Der entartete Fall eines BinDefs Blockes mit einer einzelnen Binärdateigruppe Z darin wird sein, dass Z eine Gruppe von den meisten Basis-Binärdateien sowie eine Gruppe von Knoten-Binärdateien ist. Namen von Binärdateigruppen sind im Umfang global. Es kann eine beliebige Anzahl von BinDefs Blöcken vorhanden sein, jedoch müssen die vereinbarten Binärdateigruppen eindeutig bestimmt sein. Eine Binärdateigruppe aus einem BinDefs Block ist es gestattet, eine Verfeinerung einer Binärdateigruppe aus einem anderen BinDefs Block zu sein. So könnten im oben erwähnten Beispiel SoftBins in einem von HardBins getrennten BinDefs Block sein. Es wird jedoch nachdrücklich empfohlen, dass man einen einzelnen BinDefs Block mit allen Binärdateigruppen besitzt, die der Lesbarkeit halber definiert sind.

[0214] Die oben erwähnte Hierarchie kann jetzt erweitert werden, um zu zählen wie viele DUTs (Prüfobjekte) bestanden und nicht bestanden haben, indem eine weitere Binärdateigruppe hinzugefügt wird.

```
# ----------------------------------
# File pentiumbins.bdefs
# ----------------------------------


Version 1.2.3;

BinDefs
{
    # Die Binärdateien PassFail sind eine
äußerste
    # Ebene von Binärdateien. Sie sind keine
    # Verfeinerung von irgendwelchen anderen
    # Binärdateien.

    Binärdateigruppe PassFailBins
    {
        Durchlauf: „Zählung von durchlaufen-
den DUT“,
        Ausfall: „Zählung von ausfallenden
DUT“,
    }

    # HardBins sind eine nächste Verfeine-
rungsstufe.
    # HardBins sind eine Verfeinerung der
```

```
            # PassFailBins,
            # wie durch „HardBins: PassFailBins an-
gegeben".
            Binärdateigruppe HardBins: PassFailBins
            {
              „3GHzPass": „DUT laufen bei 3 GHz
durch",

Durchlauf
              „2,8GHzPass": „DUT laufen bei 2,8 GHz
durch",

Durchlauf,
              „3GHzFail": „DUT versagen bei 3 GHz",
Ausfall,
              „2,8GHzFail": „DUT versagen bei 2,8
GHz",

Ausfall,
              LeakageFail: „DUT versagen bei Streu-
verlust",

Ausfall,
            }

            # Die SoftBins sind eine nächste
            # Verfeinerungsstufe.
            # SoftBins sind eine Verfeinerung von
HardBins.
            Binärdateigruppe SoftBins: HardBins
            {
               „3GHzAllPass":
                  „Gute DUT bei 3 GHz", „3GHzPass",
               „3GHzCacheFail":
                  „Cachespeicher versagt bei 3 GHz",
```

```
    „3GHzFail",
                „3GHzSBFTFail":
                    „Funktionstest Soft-Binärdatei
versagt bei
                                              3 GHz",
„3GHzFail",
                „3GHzLeakage":
                     „Streuverlust bei 3 GHz",  Lea-
kageFail,
                „2,8GHzAllPass":
                     „Gute DUT bei 2,8 GHz",
„2,8GHzPass",
                „2,8GHzCacheFail":
                     „Cachespeicher versagt bei 2,8
GHz",

„2,8GHzFail",
                „2,8GHzSBFTFail":
                     „Funktionstest Soft-Binärdatei
versagt
                                   bei 2,8 GHz",
„2,8GHzFail",
                „2,8GHzLeakage":
                    „Streuverluste bei 2,8 GHz", Lea-
kageFail,
              {
            }
```

[0215] Dieses Mal sind die meisten Basiswert-Binärdateien die Binärdateigruppe PassFailBins. Typisch ist, dass sie keine Verfeinerung von irgendwelchen Binärdateien sind. Die Binärdateigruppe HardBins ist eine Verfeinerung der PassFailBins und sind außerdem Basiswert-Binärdateien. SoftBins sind eine Verfeinerung der HardBins und eine Gruppe von Knoten-Binärdateien. Das oben erwähnte Beispiel hatte in der Hierarchie nur drei Bindateigruppen. Das Folgende ist eine kompliziertere Hierarchie:

```
BinDefs
{
  # Eine Gruppe der größten Basiswert-
Binärdateien
    BinGroup A {...}

  # Eine Gruppe von Basiswert-
Binärdateien, die
     # eine Verfeinerung von A ist
    BinGroup Ax: A {...}

     # Eine Gruppe von Knoten-
Binärdateien, die
     # eine Verfeinerung von Ax ist
    BinGroup Axx: Ax {...}

     # Eine Gruppe von Basiswert-
Binärdateien, die
     # eine Verfeinerung von A ist
    BinGroup Ay: A {...}

     # Eine Gruppe von Knoten-
Binärdateien, die
     # eine Verfeinerung von Ay ist
    BinGroup Ayy: Ay {...}

     # Eine Gruppe von größten Basiswert-
     # Binärdateien
    BinGroup B {...}

     # Eine Gruppe von Knoten-
Binärdateien, die
    # eine Verfeinerung von B ist
   BinGroup Bx: B {...}
}
```

[0216] In diesem Beispiel sind Ax und Ay Verfeinerungen von A, Axx ist eine Verfeinerung von Ax und Ayy ist eine Verfeinerung von Ay. Dieses Beispiel stellt außerdem die Binärdateigruppen B und Bx zur Verfügung, wobei Bx eine Verfeinerung von B ist. Die oben erwähnte Vereinbarung BinDefs mit den PassFailBins, HardBins und SoftBins genannten Binärdateigruppen werden in diesem Abschnitt als ein anhaltendes Beispiel verwendet.

[0217] Jede Binärdatei in einer Binärdateigruppe besitzt:

1. einen Namen, der entweder ein Identifizierung oder eine unmittelbare Datenfolge ist;
2. eine Beschreibung, die beschreibt, was diese Binärdatei zusammenfasst;
3. und falls sich diese Binärdatei in einer Verfeinerungs-Binärdateigruppe befindet, den Namen der Binärdatei, die eine Verfeinerung, auch als die Basiswert-Binärdatei bekannt, davon ist.

[0218] Die zwei Binärdateien in PassFailBins werden „Pass" und „Fail" genannt. Die fünf Binärdateien in HardBins werden „3GHzPass", „2,8GHzPass", „3GHzFail", „2,8GHzFail", „LeakageFail" genannt. Binärdateinamen können eine unmittelbare Datenfolge oder ein Identifizierer sein. Binärdateinamen müssen in einer Binärdateigruppe eindeutig sein, können aber über Binärdateigruppen dupliziert werden. Namen von Binärdateigruppen sind jedoch im Umfang global und müssen über einen Testplan eindeutig sein.

[0219] Von den fünf HardBins bilden sich die Binärdateien „3GHzPass" und „2,8GHzPass" beide auf die Binärdatei „Pass" der PassFailBins ab. Der Rest der HardBins bildet sich auf die Binärdateien „Fail" der PassFailBins ab.

[0220] Schließlich gibt es acht SoftBins. Die zwei Ausfälle bei 3 GHz für SBFT (Funktionstest Soft-Binärdatei) und Cachespeicher bilden auf die Hard-Binärdatei „3GHzFail" ab. Ebenso bilden die zwei Ausfälle bei 2,8 GHz für SBFT und Cachespeicher auf die Hard-Binärdatei „2,8GHzFail" ab. Beide Ausfälle infolge von Streuverlust bilden auf die gleiche Hard-Binärdatei „LeakageFail" ohne Rücksicht auf die Geschwindigkeit ab, bei der sie aufgetreten sind. Zum Beispiel ist der einfachste Test (in der äußersten Ebene), ob ein DUT einen Test besteht oder nicht besteht. Eine Verfeinerung ist zum Beispiel, ob das DUT einen Test bei einer speziellen Frequenz, z. B. 3 GHz, usw. besteht oder nicht besteht.

[0221] Binärdateien werden DUT in einem Testplan-Ablaufdatenelement, das nachstehend beschrieben wird, zugewiesen. Ein Testplan-Ablaufdatenelement besitzt eine Ergebnisklausel, in welcher der Testplan die Maßnahmen und den Übergang beschreibt, die als Ergebnis dessen stattfinden, dass ein spezielles Ergebnis von der Ausführung eines Tests zurückerhalten wird. An diesem Punkt ist es so, dass eine Anweisung SetBin auftreten kann:

```
            # eine Ablaufdatenelement-
Ergebnisklausel. Sie wird später beschrieben.
            Ergebnis 0
            {
              # vorzunehmende Maßnahme beim Zurück-
erhalten
   # einer 0 vom Ausführen eines Tests.

   # Setzen der Binärdatei auf SoftBin.
   # „3GHzPass“ drückt aus, dass das DUT
   # ausgezeichnet war.
   SetBin SoftBins.“3GHzPass“,
}
```

[0222] Viele SetBin Anweisungen könnten im Verlauf eines Testlaufs an einem DUT ausführen. Wenn der Tests schließlich beendet ist, wird die Laufzeit Zähler für die endgültige Binärdatei, die für dieses DUT gesetzt ist und für alle ihre Verfeinerungen erhöhen. Wir betrachten ein DUT, das die folgenden, während des Verlaufs seines Tests ausgeführten Anweisungen SetBin hatte:
SetBin SoftBins."3GHzSBFTFail",
SetBin SoftBins. "2,8GHzAllPass",

[0223] Dieses DUT hat den Test 3GHz-Cachespeicher und den Test Streuverlust bestanden, bestand jedoch nicht den SBFT-Test und wurde somit der Binärdatei „3GHzSBFTFail" zugewiesen. Es wurde anschließend bei 2,8 GHz getestet und bestand alle Tests. Somit ist die Zuweisung der endgültigen Binärdatei auf die Binärdatei

„2,8GHzAllPass", die sich in dem Satz von Soft-Binärdateien befindet. Diese endgültige Zuweisung wird die Zähler der folgenden Binärdateien erhöhen:

1. SoftBins. "2,8GHzAllPass";
2. was eine Verfeinerung von HardBins."2.8GHzPass" ist;
3. was eine Verfeinerung von PassFailBins."Pass" ist.

[0224] Wenn der Test abgeschlossen ist, wird die Laufzeit den Zähler der Zuweisung der endgültigen Binärdatei des DUT erhöhen, wobei es für alle anderen Binärdateien eine Verfeinerung davon ist.

[0225] Eine Anweisung SetBin ist nur an einer Knoten-Binärdatei erlaubt. Es ist verboten, eine Basiswert-Binärdatei zu setzen. Die oben erwähnte den Zähler erhöhende Semantik gewährleistet, dass:

1. Wenn die Binärdatei eine Knoten-Binärdatei ist, sie die Anzahl ist, wie oft eine Anweisung SetBin für diese Binärdatei am Ende der Prüfung eines DUT ausgeführt wurde.
2. Wenn die Binärdatei eine Basiswert-Binärdatei ist, sie die Summe der Zähler der Binärdateien ist, von denen sie eine Verfeinerung ist.

[0226] Folglich sind im oben erwähnten Beispiel in einer Anweisung SetBin nur SoftBins erlaubt. Für HardBins. "LeakageFail" ist der Zähler die Summe der Zähler für SoftBins."3GHzLeakageFail" und SoftBins."2,8GHzLeakageFail". Das Folgende sind einige Regeln, die Definitionen von Binärdateien berücksichtigen:

1. Eine Vereinbarung BinDefinitions besteht aus mehreren Binärdateigruppen-Vereinbarungen.
2. Jede Binärdateigruppen-Vereinbarung besitzt einen Namen, einen optionalen Binärdateigruppen-Namen, der eine Verfeinerung davon ist, der sich ein Block von Binärdatei-Vereinbarungen anschließt.
3. Binardatei-Vereinbarungen umfassen einen Namen, dem sich eine Beschreibung anschließt, der optional der Name der Basiswert-Binärdatei, dass diese Binärdatei eine Verfeinerung davon ist, folgt.
4. Binärdateinamen können eine unmittelbare Folge oder ein Kennungscode (ID) sein. Die zeichenlose Folge sollte kein gültiger Binärdateiname sein. Binärdateinamen sollten eindeutig unter Namen in der Binärdateigruppen-Vereinbarung sein, jedoch könnte der gleiche Name in anderen Binärdateigruppen-Vereinbarungen verwendet werden.
5. Wenn eine Binärdateigruppen-Vereinbarung Xxx eine Verfeinerung einer anderen Binärdateigruppen-Vereinbarung Yyy ist, dann müssen alle Binärdateivereinbarungen in Xxx den Namen einer Basiswert-Binärdatei aus Yyy vereinbaren. Somit ist jede der Binärdatei-Vereinbarungen in Soft-Binärdateien eine Verfeinerung einer Binärdatei von Hard-Binärdateien, weil die Soft-Binärdateien als eine Verfeinerung von Hard-Binärdateien vereinbart sind.
6. Eine Binärdateigruppen-Vereinbarung, die keine Verfeinerung einer anderen Binärdateigruppen-Vereinbarung wie beispielsweise PassFailBins ist, wird vorzugsweise Binärdatei-Vereinbarungen aufweisen, die keine Basiswert-Binärdateien vereinbaren.

[0227] Eine Binärdatei Bbb besitzt einen Satz von Basiswerten, welche der gesamte Satz von Binärdateien ist, von dem Bbb eine Verfeinerung davon ist. Sie ist formal wie folgt definiert:

1. Wenn Aaa die Basiswert-Binärdatei von Bbb ist, dann befindet sich Aaa in dem Basiswertesatz von Bbb.
2. Jeder Basiswert von Aaa befindet sich auch in dem Satz von Basiswerten von Bbb.

[0228] Binärdateigruppennamen sind in einem Testplan global.

[0229] Binärdateinamen sind zu einer Binärdateigruppe lokal.

[0230] Eine Anweisung SetBin ist nur für eine Knoten-Binärdatei erlaubt.

C++ für Binärdatei-Definitionen

[0231] Mit den oben erwähnten Regeln kann eine Binärdateigruppe vom Objekttyp für jede der Binärdateigruppen-Vereinbarungen in der Vereinbarung BinDefs konstruiert werden. Die Klasse Binärdateigruppe wird eine Unterklasse LeafBinGroup aufweisen. Die Operationen dieser zwei Klassen sind die gleichen mit der Ausnahme, dass BinGroup::incrementBin eine C++ geschützte Operation ist, während Leaf-BinGroup::incrementBin eine allgemein zugängliche Operation ist.

[0232] Das Folgende ist ein Standardkonstruktor, der eine BinGroup oder eine LeafBinGroup aufbaut, die keine Verfeinerung einer beliebigen Binärdateigruppe ist.

Konstruktoren:

[0233] BinGroup(BinGroup&baseBinGroup);
LeafBinGroup(BinGroup& baseBinGroup),
diese baut eine Binärdateigruppe auf, die eine Verfeinerung der gegebenen Basiswert-Binärdateigruppe ist.

[0234] Ein Verfahren
Status addBin(const String& binName,
const String& description,
const String& baseBinName),
zum Definieren einer Binärdatei und ihrer Beschreibung. Wenn sie eine größte Basiswert-Binärdatei ist, muss der Parameter des Basiswert-Binärdateinamens die zeichenlose Folge sein.

[0235] Verfahren zum Erhöhen von Binärdateizählern:
Status incrementBin(const String& binName);
Diese Operation wird den Zählre für diese Binärdatei und für alle Binärdateien, die Basiswerte dieser Binärdatei sind, erhöhen. Die Operation wird in der Klasse BinGroup geschützt und ist in der Klasse LeafBinGroup allgemein zugänglich.

[0236] Verfahren zum Rücksetzen von Binärdatei-Zählern
Status resetBin(const String& binName),
Diese Operation wird den Zähler für diese Binärdatei und für alle Binärdateien rücksetzen, die die Basiswerte dieser Binärdatei sind.

[0237] Verfahren zur Gewinnung von Informationen über eine Binärdatei:
Status getBinDescription(const String& binName,
String& description),
Status getBaseBin(const String& binName,
BinGroup* pBaseBinGroup,
String& baseBinName),
Status getBinValue(const String& binName,
unsigned int& value),
Iteratoren werden vorgesehen, um alle gegenwärtig definierten Binärdateinamen zu gewinnen.

[0238] Der Testplanzustand wird eine Anzahl von Binärdateigruppenelementen, eins für jede Vereinbarung von Binärdateigruppen, umfassen. Das C++ für oben genannte Binärdateidefinitionen würde wie folgt sein:

```
//Testplankonstruktor
TestPlan::TestPlan0
:m_PassFailBins0, // Standardkonstruktor
 m_HardBins(&m_PassFailBins),
 m_SoftBins(&m_HardBins)
{}

 // Initialisierungen von Binärdateien
 m_PassFailBins.addBin(„Pass“, „Count of passing
DUTS.”,“”),
 m_HardBins.addBin(“3GHzPass”, “Duts passing 3GHz”,
“Pass”),
 ...
```

[0239] Der Zustand für einen Testplan umfasst eine m_pCurrent-Binärdateigruppe, die zu der unbestimmten Binärdateigruppe (NULL) und dem m_currentBin unbestimmten Binärdateinamen (die zeichenlose Folge) initialisiert wird. Jedes Mal, wenn eine SetBin-Anweisung ausgeführt wird, wird durch einen Aufruf die m_pCurrent-Binärdateigruppe zu der angegebenen nominierten Binärdateigruppe und die m_current Binärda-

tei zu der nominierten Binärdatei in der Gruppe umgewandelt:
//Umsetzung von: SetBin SoftBins."3GHzAllPass", pTestplan->setBin(„SoftBins", „3GHzAllPass"),

[0240] Wenn der Testplan die Ausführung beendet hat, wird er m_pCurrentBinGroup->increment-Bin(m_currentBin) aufrufen, was bewirkt, dass die Zähler dieser Binärdatei und aller ihrer Basiswert-Binärdateien erhöht werden.

[0241] Die Zähler der Binärdateigruppen werden zurückgesetzt, wenn der Testplan entwickelt ist, werden jedoch nicht jedes Mal erneut initialisiert, wenn ein Test läuft. Die Zähler können durch einen expliziten Aufruf an Binärdateigruppe::resetBin zurückgesetzt werden.

C. Der Testplan

[0242] Den Testplan kann man sich als Hauptstruktur des Testprogramms vorstellen. Der Testplan kann sowohl Dateien importieren als auch ähnliche Konstrukte mitlaufend definieren. Somit ist es sowohl möglich, eine Datei mit gegebenen Definitionen von einigen Globalen zu importieren als auch zusätzliche Globale mitlaufend zu vereinbaren.

C1. Testplanabläufe und Ablaufelemente

[0243] Eines der entscheidenden Elemente des Testplans ist der Ablauf. Ein Ablauf schließt einen Endlichzustandsautomaten ein. Er weist mehrere Ablaufelemente auf, die ein IFlowable-Objekt abarbeiten und dann zu einem anderen Ablaufelement übergehen. Das Abarbeiten einer IFlowable umfasst das Abarbeiten eines Objektes, das die Schnittstelle IFlowable implementiert.

[0244] Typische Objekte, die die Schnittstelle IFlowable implementieren, sind Tests und Abläufe selbst.

[0245] Somit besitzt ein Ablauf Ablaufelemente, die Tests und andere Abläufe abarbeiten und anschließend zu einem anderen Ablaufelement übergehen. Er bewirkt außerdem die Möglichkeit, anwenderspezifische Routinen hinsichtlich verschiedener Rücksetzergebnisse vom Abarbeiten einer IFlowable aufzurufen. Typisch ist, dass ein Ablauf somit die folgende Form besitzt:

```
        #
        # Ablauftest1 implementiert einen
        # Endlichzustandsautomaten für die Sorten Min,
Typ und
        # Max von MyFunctionalTest1Test1. Bei Erfolg
testet er
        # Test1Min, Test1Typ, Test1Max und setzt an-
schließend
        # auf seinen Aufrufer mit 0 als einem erfolg-
reichen
        # Status zurück. Bei Fehlschlag setzt er 1 als
einen
        # Fehlerstatus zurück.
        #
        # Es wird angenommen, dass die Tests
        # MyFunctionalTest1min, ... alle ein Ergebnis
von 0
        # (Abnahme), 1 und 2 (ein paar Fehlerebenen)
        # rücksetzen.
        #             Ergebnis 0    Ergebnis 1      Er-
gebnis 2
        # Test1Min    Test1Typ      rücksetzen 1
rücksetzen 1
        # Test1Typ    Test1Max      rücksetzen 1
rücksetzen 1
```

```
        # Test1Max   rücksetzen 0  rücksetzen 1
rücksetzen 1
        #
        Ablauf Ablauftest1
        {
            Ablaufelement Ablauf-
Test1_MinMyFunctionalTest1Min
            {
               Ergebnis 0
               {
                 Merkmal PassFail = „Pass",
                 Inkrementierungszähler PassCount,
                 GeheZu FlowTest1_Typ,
               }

               Ergebnis 1
               {
                  Merkmal PassFail = "Fail",
                  Inkrementierungszähler FailCount,
                  Rücksetzen 1,
               }
               # Dieser Block wird ausgeführt, falls
               # MyFunctionalTest1Min eine von 2, 5,
6, 7, -6,
               # -5 oder -4 zurücksetzt
               Ergebnis 2, 5:7, -6:4
               {
                  Merkmal PassFail = „Fail",
                  Inkrementierungszähler FailCount,
                  Rücksetzen 1,
               }
            }

            Ablaufelement FlowTest1_Typ {...}
            Ablaufelement FlowTest1_Max {...}
         }
```

[0246] Die Rechenoperation des Ablaufs Ablauftest1 ist wie folgt:

1. Inbetriebnahme mit Ausführen von Ablaufelement FlowTest1_Min.
2. FlowTest1_Min arbeitet Funktionstest MyFunctionalTest1Min ab. Einzelheiten dieses Tests werden bereitgestellt, wenn der gesamte Testplan nachstehend dargestellt ist.
3. Es wird erwartet, dass neun Ergebnisse diesen Test 0, 1, 2, 5, 6, 7, –6, –5, oder –4 abarbeiten. Die ersten

zwei Ergebnisklauseln verarbeiten jeweils 0 und 1, und die dritte verarbeitet den gesamten Rest der Ergebniswerte.

4. Falls das Ergebnis „0" (Abnahme) auftritt, dann wird FlowTest1_Min den Zähler PassCounter erhöhen. Es wird anschließend zu einem neuen Ablaufelement FlowTest1_Typ übergehen.

5. Falls Ergebnis „1" oder Ergebnis „2" auftritt, dann wird FlowTest1_Min den Zähler FailCounter erhöhen und von dem Ablauf zurücksetzen.

6. FlowTest1_Typ wird in der gleichen Weise und bei Erfolgsaufruf FlowTest1_Max verarbeiten.

7. FlowTest1_Max wird in der gleichen Weise und bei Erfolgsrücksprung von FlowTest1 mit einem erfolgreichen Ergebnis („0") verarbeiten.

[0247] Folglich wird FlowTest1 bei einem erfolgreichen Lauf ein Bauelement durch die Versionen Minimal, Typisch und Maximal von Test1 abarbeiten und anschließend rücksetzen. FlowTest2 wird in gleicher Weise verarbeiten.

[0248] Ein wie oben beschriebener Ablauf beschreibt grundsätzlich einen Endlichzustandsautomaten mit Zuständen und Übergängen

[0249] Die Ablaufelemente sind grundsätzlich Zustände, die das Folgende machen werden:

1. Ausführen einer IFlowable (es könnte ein zuvor definierter Ablauf oder ein Test oder ein anwenderdefinierter Ablauf sein, der in C++ mit den oben erwähnten Regeln implementiert werden kann).

2. Ausführung der IFlowable setzt ein numerisches Ergebnis zurück. Basierend auf dem Ergebnis treten bestimmte Maßnahmen ein (Aktualisieren einiger Zähler), und dann passiert eines von zwei Dingen:

a) Der Ablauf kehrt zum Aufrufer mit einem numerischen Ergebnis zurück.

b) Der Ablauf setzt sich fort, indem zu einem anderen Zustand übergegangen wird (Ablaufelement).

[0250] So besitzt ein Ablaufelement die folgenden Komponenten:
Ein Ablaufelement besitzt einen Namen.

[0251] Ein Ablaufelement besitzt eine auszuführende IFlowable.

[0252] Ein Ablaufelement besitzt eine Anzahl oder Ergebnisklauseln.

[0253] Jede Ergebnisklausel eines Ablaufelements bewirkt Maßnahmen und endet mit einem Übergang und wird einem oder mehreren Ergebniswerten verknüpft.

[0254] Diese Elemente sind in einem Ablaufelement syntaktisch wie folgt.

```
        Ablaufelement <Name><auszuführende IFlo-
wable>
        {
            Ergebnis <eine oder mehrere Ergebniswer-
te>
            {
               <Maßnahmen für diese Ergebniswerte>
               <Übergang für diese Ergebniswerte>
            }

            Ergebnis <ein oder mehrere andere Ergeb-
niswerte>
            {
               ...
            }
            ...
          }
```

[0255] Die auszuführende IFlowable könnte entweder ein Test oder eine anwenderdefinierte IFlowable oder ein Ablauf sein. Die Maßnahmen für ein Ergebnis könnten eine der folgenden sein:
Eine Merkmalsmaßnahme zum Setzen von mit Folgewerten versehenen Entitäten, die durch GUI Tools genutzt werden, um auf Ergebnisse zurückzuführen. Dies wird ersichtlich in dem oben erwähnten FlowTest1-Beispiel mit:
Merkmal PassFail = „Pass",

[0256] Merkmale sind im Grunde mit Folgewerten oder ganzzahligen Werten versehene, nominierte Entitäten, die mit einer Ergebnisklausel verknüpft sind. Es kann eine Anzahl von ihnen vorhanden sein, und sie werden vorzugsweise durch Tools wie beispielsweise GUI (graphische Benutzeroberflächen) verwendet, die ein Anwender nutzen würde, um mit diesem Ergebnis verknüpfte Informationen anzuzeigen. Sie haben keine Auswirkung auf das tatsächliche Ergebnis des Tests oder den Ablauf des Tests.

[0257] Eine Zählermaßnahme zum Erhöhen einer gewissen Anzahl von Zählern. Dies ist im oben erwähnten Beispiel ersichtlich mit:
Inkrementierungszähler PassCount

[0258] Eine Routinenaufrufmaßnahme zum Aufrufen einer beliebigen oder Anwenderroutine. Diese wird später erörtert.

[0259] Schließlich weist ein Ablaufelement einen Übergang auf, der entweder eine GoTo Anweisung sein könnte, um eine Kontrolle auf ein anderes Ablaufelement zu übertragen oder eine Rücksetzanweisung sein, um eine Kontrolle zurück auf den Aufrufer zu übertragen (entweder ein Aufrufablauf oder die Systemroutine, die den Testplan initiiert hat).

Vorbestimmte Abläufe

[0260] Der typische Gebrauch von Ablaufobjekten ist, eine Folge von Tests zu definieren. Diese Folge wird dann als Ergebnis eines Ereignisses ausgeführt, das in einem Testplan-Server (TPS), d. h. dem Testplan-Ausführereignis, stattfindet. Ein Testplan-Server an jedem Site-Controller führt den Testplan des Benutzers aus. Jedoch werden Ablaufobjekte auch als Reaktion auf andere Ereignisse ausgeführt. Der Name in Klammern ist der Name, der genutzt wird, um diesen Ereignissen Abläufe zuzuweisen.

1. Systemladeablauf (SysLoadFlow). Dieser Ablauf wird an der Systemsteuereinheit ausgeführt, wenn ein Testplan auf einen oder mehrere Site-Controller geladen wird. Er wird vor dem eigentlichen Laden des Test-

plans auf einen beliebigen Site-Controller ausgeführt. Dieser Ablauf ermöglicht es dem Entwickler des Testplans, Maßnahmen zu definieren, die aus der Systemsteuereinheit stammen sollten. Solche Maßnahmen umfassen das Senden einer Ladung von Strukturdatenfiles, Kalibrierungsmaßnahmen, usw.
2. Site-Load Ablauf (SiteLoadFlow). Dieser Ablauf wird auf dem Site-Controller ausgeführt, nachdem auf den Standort ein Testplan geladen und initialisiert wurde. Dies ermöglicht es, dass eine beliebige sitespezifische Initialisierung auftritt.
3. Abläufe von Stichprobenstart/Ende (LotStartFlow/LotEndFlow). Diese Abläufe arbeiten auf den Site-Controller ab, wenn der Testplanserver über den Start einer neuen Stichprobe benachrichtigt wird. Typisch ist, dass dieser in Produktionsumgebungen genutzt wird, um Datenerfassungsströme mit Anmerkungen stichprobenspezifischer Informationen zu versehen.
4. DUT Änderungsablauf (DutChangeFlow). Dieser Ablauf arbeitet auf dem Site-Controller ab, wenn sich seine DUT Informationen ändern. Typisch ist, dass dieser in Produktionsumgebungen zum Aktualisieren von Datenerfassungsströmen genutzt wird.
5. Abläufe von Testplan-Start/Ende (TestPlanStartFlow/TestPlanEndFlow).
Diese Abläufe arbeiten auf dem Site-Controller ab,
wenn der Testplanserver instruiert wird, die Ausführung des aktuellen Testablaufs zu
starten und
wenn dieser Ablauf seine Ausführung beendet.
6. Abläufe von Test-Start/Ende (TestStartFlow/TestEndFlow). Diese Abläufe arbeiten auf dem Site-Controller ab, wenn der Testablauf beginnt, einen neuen Test abzuarbeiten und wenn dieser Test seine Ausführung beendet.
7. Testablauf (TestFlow). Dieser Ablauf ist das Hauptablaufobjekt, das ausgeführt wird, wenn der Testplanserver die Nachricht „Testplan ausführen" empfängt.

[0261] Anzumerken ist, dass, wenn ein Benutzer einen Ablauf im Testplan des Benutzers definiert, der nicht der Testablauf oder einer der anderen vorbestimmten Abläufe ist, dann die bevorzugte Weise ihn ausführen zu lassen ist, dass er in den Übergangszuständen von einem dieser vorbestimmten Abläufe enthalten ist.

Beispiel eines Testplans

[0262] In dem nachstehenden Beispiel sind Abläufe zusammen mit Kommentaren gegeben, die den durch den Ablauf implementierten Endlichzustandsautomaten beschreiben. Der Endlichzustandsautomat ist als eine Übergangsmatrix gegeben. Zeilen der Matrix entsprechen Ablaufelementen und Spalten dem Ergebnis. Die Eingaben einer Zeile der Matrix geben das Ablaufelement an, auf das von dem Ablaufelement der Zeile übergegangen wird, wenn das rückgesetzte Ergebnis der in der Spalte bestimmte Wert ist.

[0263] Nachstehend ist ein Testplan mit den drei Abläufen FlowTest1, FlowTest2 und FlowMain dargestellt. FlowTest1 wird wie oben beschrieben arbeiten. Er wird einen mit MyFunctionalTest1 bezeichneten Test jeweils in den Konfigurationen „min", „typ" und „max" abarbeiten. Ebenso wird FlowTest2 in jeder dieser Konfigurationen MyFunctionalTest2 abarbeiten. Schließlich wird FlowMain FlowTest1 und FlowTest2 abarbeiten. Die Übergangsmatrix des Endlichzustandsautomaten wird in Kommentaren beim Start jeder dieser Abläufe bereitgestellt.

```
        # ----------------------------
        # File mySimpleTestPlan.tpl
        # ----------------------------

        Version 0.1

        Importiere xxx.pin, # Pins
        # Konstante und Variable, die begrenzende Wer-
te geben.
        Importiere limits.usrv,

        # Testbedingungsgruppen Importtest
        Importiere myTestConditionGroups.tcg,

        # Importiere einige Binärdateidefinitionen
        Importiere bins.bdefs,

        # ----------------------------
        # Start des Testplans
        # ----------------------------
```

```
Testplan Abtastwert

# Dieser Block definiert Strukturlistenamen,
die in
# einer Datei qualifiziert sind, und
# Strukturlistenvariable, die in Testvereinba-
rungen
# verwendet werden. Strukturlistenvariable
werden
# zurückgestellt, bis eine kundenspezifische
Auslegung
# geprüft ist.
PListDefs
{
  # Dateiqualifizierte Strukturlistennamen
  pl1A.plist:pat1Alist,
  pl2A.plist:pat2AList
}

# Die Steckstelle für diese Tests in diesem
Testplan
 (diese wird nicht importiert, jedoch aufgelöst
bei
  Aktivierungszeit):
SocketDef = mytest.soc,

# Einige Benutzervariable mitlaufend vereinba-
ren
UserVars
{
  # Sequenzname für aktuellen Test
  Sequenz CurrentTest = „MyTest“,
}

Testbedingung TC1Min
{
```

```
    Testbedingungsgruppe = TCG1,
    Auswähler = min,
}

Testbedingung TC1Typ
{
   Testbedingungsgruppe = TCG1,
   Auswähler = typ,
}

Testbedingung TC1Max
{
   Testbedingungsgruppe = TCG1,
   Auswähler = max,
}

# Ebenso für TC2Min, TC2Typ, TC2Max,


#
# Vereinbare einen Funktionstest. „Funktions-
test"
# verweist auf eine C++-Testklasse, die den
Test
# abarbeitet und eine 0, 1 oder 2 als ein Er-
gebnis
# rücksetzt. Die Testbedingungsgruppe TCG1
wird mit dem
# Auswähler „min" ausgewählt, indem zur Test-
bedingung
# TC1Min Bezug hergestellt wird.
#
Teste Funktionstest MyFunctionalTest1Min
{
  PListParam = pat1AList,
  TestConditionParam = TC1Min,
```

```
}

# Ein weiterer Funktionstest, der TCG1 mit
"Typ"
  auswählt
Teste Funktionstest MyFunctionalTest1Typ
{
  PListParam = pat1AList,
  TestConditionParam = TC1Typ,
}

# Ein weiterer Funktionstest, der TCG1 mit
„Max"
  auswählt
Teste Funktionstest MyFunctionalTest1Max
{
  PListParam = pat1AList,
  TestConditionParam = TC1Max,
}

# Wähle jetzt mit "Min" TCG2 aus
Teste Funktionstest MyFunctionalTest2Min
{
   PListParam = pat2AList,
   TestConditionParam = TC2Min,
}

# Ebenso für TCG2 mit „Typ" und TCG2 mit „Max"
Teste Funktionstest MyFunctionalTest2Typ
{
   PListParam = pat1AList,
   TestConditionParam = TC2Typ,
}

Teste Funktionstest MyFunctionalTest2Max
{
```

```
    PListParam = pat1AList,
    TestConditionParam = TC2Max,
 }

 #
 # Zu diesem Zeitpunkt sind die folgenden Test-
objekte definiert worden
 # MyFunctionalTest1Min
 # MyFunctionalTest1Typ
 # MyFunctionalTest1Max
 # MyFunctionalTest2Min
 # MyFunctionalTest2Typ
 # MyFunctionalTest2Max
 #
 #
 # Zähler sind Variable, die während der Aus-
führung
 # eines Tests erhöht werden. Sie sind vorzei-
chenlose
 # ganze Zahlen, die zu Null initialisiert wer-
den.
 #
 Zähler {Durchlaufzählung, Ausfallzählung}

 #
 # Es können jetzt Abläufe dargestellt werden.
Ein
 # Ablauf ist ein Objekt, das im Wesentlichen
einen
 # Endlichzustandsautomaten darstellt, der Flo-
wables
 # ausführen und in andere Flowables umwandeln
kann, was
 # auf dem Ergebnis basiert, das beim Ausführen
einer
```

```
 # Flowable rückgesetzt wird. Ein Ablauf kann
auch einen
 # anderen Ablauf aufrufen.
 #
 # Ein Ablauf besteht aus einer Anzahl von
 # Ablaufelementen und Übergängen zwischen die-
sen.
 # Ablaufelemente weisen Namen auf, die in dem
 # umfassenden Ablauf, ein „Flowable" Objekt
ausführen,
 # eindeutig bestimmt sind und dann zu einem
anderen
 # Ablaufelement im gleichen umfassenden Ablauf
 # umwandeln.
 #
 # Flowable Objekte enthalten Tests und andere
Abläufe.
 # Wenn ein Flowable Objekt abarbeitet, setzt
es ein
 # numerisches Ergebnis zurück, welches durch
das
 # Ablaufelement genutzt wird, um zu einem an-
deren
 # Ablaufelement überzugehen. Als Folge davon
enden
 # beide Tests und Abläufe, indem ein numeri-
scher
 # Ergebniswert rückgesetzt wird.
 #
 # FlowTest1 führt einen Endlichzustandsautoma-
ten für
 # die Sorten Min, Typ und Max des MyFunctio-
nalTest1
 # aus. Bei Erfolg testet Test1Min, Test1Typ,
Test1Max
```

```
 # und setzt anschließend auf seinen Aufrufer
mit 0 als
 # einem erfolgreichen Ergebnis zurück. Bei
Ausfall
 # setzt er 1 als ein fehlerbehaftetes Ergebnis
zurück.
 #
 # Es wird vorausgesetzt, dass die Tests
 # MyFunctionalTest1Min, ... alle ein Ergebnis
von 0
 # (Durchlauf), 1 und 2 (für mehrere Ausfall-
ebenen)
 # rücksetzen. Die Übergangsmatrix des durch
FlowTest1
 # implementierten Endlichzustandsautomaten
ist:

 # ----------------------------
 #            Ergebnis 0      Ergebnis 1
Ergebnis 2
 # ----------------------------
 # FlowTest1_Min     FlowTest1_Typ    return 1
return 1
 # FlowTest1_Typ     FlowTest1_Max    return 1
return 1
 # FlowTest1_Max         return 0     return 1
return 1
 #
 # wobei die durch jedes Ablaufelement abgear-
beiteten
   IFlowables sind:
 # FlowItem           IFlowable, die abgearbei-
tet hat
 # FlowTest1_Min      MyFunctionalTest1Min
 # FlowTest1_Typ      MyFunctionalTest1Typ
 # FlowTest1_Max      MyFunctionalTest1Max
```

```
#
Ablauf FlowTest1
{
    Ablaufelement FlowTest1_Min MyFunctional-
Test1Min
    {
        Ergebnis 0
        {
           Merkmal PassFail = „Pass",
           Inkrementierungszähler PassCount
           GoTo FlowTest1_Typ
        }
        Ergebnis 1,2
        {
           Merkmal PassFail = „Fail",
           Inkrementierungszähler FailCount,
           Rücksetzen 1,
        }
     }

     Ablaufelement FlowTest1_Typ MyFunctional-
Test1Typ
     {
        Ergebnis 0
        {
          Merkmal PassFail = „Pass",
          Inkrementierungszähler PassCount,
          GeheZu FlowTest1_Max,
        }

        Ergebnis 1,2
        {
           Merkmal PassFail = „Fail",
           Inkrementierungszähler FailCount,
           Rücksetzen 1,
        }
```

```
        }

        # Ebenso für FlowTest1_Max
        Ablaufelement FlowTest1_Max MyFunctio-
nalTest1Max
        {
            Ergebnis 0
            {
               Merkmal PassFail = „Pass“,
               Inkrementierungszähler Pass-
Count,
               Rücksetzen 0
            }

            Ergebnis 1,2
            {
               Merkmal PassFail = „Fail“,
               Inkrementierungszähler Fail-
Count,
               Rücksetzen 1,
             }
          }
         }

        #
        # Ablauftest 2 ist Ablauftest 1 ähn-
lich. Er
        # implementiert einen Endlichzustandau-
tomaten
        # für die Sorten min, Typen und Max von
        # MyFunctionalTest2. Bei Erfolg testet
er
        # Test2Min, Test2Typ, Test2Max und
setzt dann
        # mit 0 als einem erfolgreichen Ergeb-
nis zurück
```

```
        # zu seinem Aufrufer. Bei Ausfall setzt
er 1 als
        # ein fehlerbehaftetes Ergebnis zurück.
        # Vorausgesetzt wird, dass die Tests
        # MyFunctionalTest2Min, ... alle ein
Ergebnis
        # von 0 (Durchlauf), 1 und 2 (für meh-
rere
        # Ausfallgrößen) rücksetzen. Die Über-
gangsmatrix
        # des durch FlowTest2 implementierten
        # Endlichzustandsautomaten ist:



# -------------------------------
#                   Ergebnis 0   Ergebnis 1
Ergebnis 2
# -------------------------------
# FlowTest2_Min FlowTest2_Typ  rücksetzen 1
rücksetzen 1
# FlowTest2_Typ FlowTest2_Max  rücksetzen 1
rücksetzen 1
# FlowTest2_Max rücksetzen 0  rücksetzen 1
rücksetzen 1
#
# wobei die durch jedes Ablaufelement abgear-
beiteten
# IFlowables sind:
#   Ablaufelement    IFlowable, die abgearbei-
tet ist
# FlowTest2_Min   MyFunctionalTest2Min
# FlowTest2_Typ   MyFunctionalTest2Typ
# FlowTest2_Max       MyFunctionalTest2Max
#
Ablauf FlowTest2
```

```
{
  # ...
}


#
# Es kann jetzt FlowMain, der Haupttestablauf,
# dargestellt werden. Er implementiert einen
# Endlichzustandsautomaten, der FlowTest1 und
FlowTest2
# wie unten aufruft:
# ----------------------
#          Ergebnis 0    Ergebnis 1
# ----------------------
# FlowMain_1   FlowMain_2   rücksetzen 1
# FlowMain_2   rücksetzen 0 rücksetzen 1
#
# wobei die durch jedes Ablaufelement abgear-
beiteten IFlowables sind:
# Ablaufelement   IFlowable, die abgearbeitet
ist
# FlowMain_1      FlowTest1
# FlowMain_2      FlowTest2
Ablauf FlowMain
{
  # Der erste vereinbarte Ablauf ist der auszu-
führende
  # Anfangsablauf. Er geht bei Erfolg zu Flow-
Main_2 über
  # und setzt 1 bei Ausfall zurück.
  Ablaufelement FlowMain_1 FlowTest1
  {
      Ergebnis 0
      {
         Merkmal PassFail = „Pass“,
         InkrementierungsZähler PassCount,
```

```
      GoTo FlowMain_2
    }

    Ergebnis 1
    }
      # Bedaure, ... FlowTest1 hat versagt
      Merkmal PassFail = „Fail",
      InkrementierungsZähler FailCount,

      # Addiere zur rechten Soft-Binärdatei
      SetBin SoftBins."3GHzSBFTFail",

      Rücksetzen 1,
     }
   }
    Ablaufelement FlowMain_2 FlowTest2
    {
      Ergebnis 0
      {
        # Alle bestanden!
        Merkmal PassFail = „Pass",
        InkrementierungsZähler PassCount,

        # Addiere zur rechten Soft-
Binärdatei
        SetBin SoftBins."3GHzAllPass",

        Rücksetzen 0
      }

      Ergebnis 1
      {
        # FlowTest1 bestanden, jedoch Flow-
Test2 nicht
        # bestanden
        Merkmal PassFail = „Fail",
```

```
            InkrementierungsZähler FailCount,

            # Addiere zur rechten Soft-
Binärdatei
            SetBin SoftBins."3GHzCacheFail",

            Rücksetzen 1,
          }
        }
      }
```

Testablauf = FlowMain

[0264] Der oben erwähnte Testplan ist wie folgt in einer bevorzugten Reihenfolge strukturiert:

1. Zuerst wird eine Versionsnummer bereitgestellt. Diese Nummer wird verwendet, um Kompatibilität mit der Kompiliererversion zu gewährleisten.
2. Anschließend wird eine Anzahl von Importen vereinbart. Diese sind verschiedene Dateien mit Vereinbarungen, die benötigt werden, um im Testplan verwendete Namen aufzulösen.
3. Als Nächstes wird der Testplan vereinbart, nach dem die mitlaufenden Vereinbarungen des Testplans kommen.
4. Als Nächstes wird eine Größe von PListDefs vereinbart. Diese enthalten dateiqualifizierte Namen, die aus den benannten Dateien GlobalPLists nominieren. Sie enthalten auch Strukturlistenvariable. Strukturlistenvariable sind Variable, die zur Ausführungszeit zu kundenspezifischen Flowables erstellt werden können. Sie bewirken ein Mittel zur Verzögerung von Bindungstests an tatsächliche Strukturlisten bis zur Laufzeit.
5. Als Nächstes wird eine Größe von UserVars vereinbart. Diese enthalten eine Sequenz.
6. Anschließend werden einige Zähler vereinbart, um die Anzahl von Tests zu bestimmen, die bestanden und die nicht bestanden wurden. Zähler sind einfach Variable, die zu 0 erstellt und bei Anweisungen IncrementCounter inkrementiert werden. Sie sind unterschiedlich zu früher beschriebenen Binardateien, die eine Semantik besitzen, dass nur die zurzeit gesetzte Binärdatei am Ende des Tests eines Probestücks (DUT) inkrementiert wird.
7. Als Nächstes wird eine Reihe von Testbedingungen vereinbart. Jede von diesen bestimmt eine Testbedingungsgruppe und einen Auswähler. In diesem Beispiel stammen die Testbedingungsgruppen von mytestconditionsgroup.tcg. Sie könnten jedoch in dem Testplan mitlaufend gewesen sein.
8. Als Nächstes wird eine Reihe von Flowables oder Tests vereinbart. Jeder von ihnen ist der bekannte Test FunctionalTest, der eine Strukturliste und eine Testbedingung auswählt. So wählt zum Beispiel MyFunctionalTest1Max die Testbedingung TC1Max und eine Strukturliste aus.
9. Anschließend an diese werden drei Abläufe Flow Test1, Flow Test2 und FlowMain vereinbart. Abläufe arbeiten Flowables ab. Flowables enthalten Tests (wie beispielsweise MyFunctionalTest1Max) und andere Abläufe (wie FlowTest1 und FlowTest2). Jeder von FlowTest1 und FlowTest2 arbeitet sich durch die Version Minimal, Typisch und Maximal von Test1 bzw. Test2. Der Ablauf FlowMain ruft die früher vereinbarten Abläufe FlowTest1 und anschließend FlowTest2 auf.
10. Schließlich wird das Testablaufereignis dem Ablauf MainFlow zugewiesen. Somit ist der Ablauf FlowMain der eine, der durch diesen Testplan ausgeführt werden wird, wenn ein Benutzer wählt, diesen Plan auszuführen.

C++ für Abläufe

[0265] Mit den oben erwähnten Regeln kann eine C++ Implementierung für die meisten der Elemente mit Ausnahme der Abläufe selbst vorgenommen werden.

C++ für Ablaufelemente

[0266] Die C++-Klasse zum Darstellen eines Ablaufelements kann die folgende Schnittstelle besitzen:
Eine Rechenoperation
Status setFlowable(IFlowable* pIFlowable),

die die IFlowable setzen wird, wird für dieses Ablaufelement ausgeführt werden.

[0267] Sobald das Ablaufelement aus der Menge von Aufrufen, die zum Ausführen dieser IFlowable benötigt werden, rücksetzt, wird es eine Liste von Zählern in Abhängigkeit von dem Ergebniswert inkrementieren müssen. Zu diesem Zweck muss das Ablaufelement einen Vektor von Zählern haben, die es implementieren soll. Dies wird durch einen Aufruf erstellt:
Staus setCounterRefs(unsigned int result,
CounterRefList counterRefs);

[0268] Das Aufrufen desselben erstellt einen Vektor von Verweisen auf Zähler in das Ablaufelement, so dass es sie inkrementieren kann, sobald die IFlowable die Ausführung beendet. Zum Beispiel würde die Anweisung InkrementierungsZähler A, B, C
vorzugsweise den oben erwähnten Aufruf wie folgt nutzen:

```
//Irgendwo früher
CounterRefList counters,
...
// Code für Ergebnisklausel
// Ergebnis 2, 3 {...}
// von Ablaufobjekt.
counters.reset(),
counters.add(&A),
counters.add(&B),
counters.add(&C),
flowObject.setCounterRefs(2,counters),
flowObject.setCounterRefs(3, counters),
```

[0269] Es wird ein Zähler genanntes, temporäres Objekt CounterRefList genutzt. Am Anfang wird counters.reset() aufgerufen, dem sich eine Anzahl von Aufrufen counters.add() anschließt, um die Zählerliste zu erstellen. Diese wird dann verwendet, um den Vektor von zu aktualisierenden Zähleradressen für Ergebniswerte 2 und 3 zu erstellen.

[0270] Anschließend kann das Ablaufelement benötigt werden, um zu einem anderen Ablaufelement an einem speziellen Ergebnis überzugehen:
Status setTransition(unsigned int result,
FlowItem*pFlowItem),

[0271] Mehrere solcher Aufrufe werden natürlich in dem Fall vorgenommen werden müssen, dass sich eine bestimmte Ergebnisklausel mit vielen Ergebniswerten befasst.

[0272] Das Ablaufelement kann ein Ergebnis rücksetzen müssen. Dies wird vorgenommen durch:
Status setReturnResult(unsigned int result,
unsigned int returnResult),

[0273] Zum Beispiel würde für das Ablaufelement FirstFlowItem in dem vorherigen Beispiel das oben Genannte mit dem Wert "2" für "Ergebnis" und "1" für "Ergebnis rücksetzen" aufgerufen werden.

[0274] Schließlich benötigt das Ablaufelement eine Rechenoperation, um auszuführen:
Status execute(unsigned int& result, FlowItem* pNextFlowItem);

[0275] Diese Rechenoperation wird die IFlowable ausführen, dann die angegebenen Zähler aktualisieren und anschließend entweder ein Ergebnis oder einen Zeiger auf das nächste Ablaufelement rücksetzen. Wenn dieser Zeiger NULL ist, dann ist das Ergebnis der zurückgesetzte Wert.

[0276] Der Code, der für das Ablaufelement FlowMain_1 generiert werden würde, ist wie folgt:

```
      FlowItemMain_1,
      FlowItemMain_2,

      Zähler CounterRefList

      FlowMain_1.setFlowable(FlowTest1),

      //Ergebnis 0
      counters.reset(),
      counters.add(&PassCount),
      FlowMain_1.setCounterRefs(0, counters),
      FlowMain_1.setTransition(0, &FlowMain_2),

      //Ergebnis 1
      counters.reset(),
      counters.add(&FailCount),
      FlowMain_1.setCounterRefs(1, counters),

      //Der folgende Aufruf vom ITestPlan wird
die
      # aktuelle Binärdateigruppe und den aktu-
ellen
      # Binärdateinamen setzen.
      pTestPlan->setBin(„SoftBins",
„3GHzSBFTFail"),
      FlowMain_1.setReturnResult(1, 1),
```

[0277] Der oben generierte Code erstellt FlowMain_1, um die IFlowable "FlowTest1" abzuarbeiten und erstellt sie anschließend, um die entsprechende Liste von Zählern für jedes Ergebnis zu inkrementieren, und um schließlich die notwendigen Maßnahmen zu ergreifen. Die notwendige Maßnahme im Fall von Ergebnis „0" ist ein Übergang zu FlowMain_1, und im Fall von Ergebnis „1" eine Rückführung.

C2. Zählerunterstützung in einem Testplan

[0278] Zähler sind Variable, die zu Null initialisiert werden und durch eine Anweisung IncrementCounter an verschiedenen Punkten während eines Testlaufs inkrementiert werden können. Sie sind unterschiedlich zu Binärdateien, die nur am Ende des Tests implementiert werden können. Darüber hinaus sind Binärdateien hierarchisch, während Zähler einfache Variable sind. Somit sind Zähler eine Systemeinrichtung, die viel einfacher und begrenzter als Binärdateien ist.

[0279] Zähler können in einem Testplan unterstützt werden über ein Element einer Zählerklasse, das eine Menge von nominierten Zählern hält, die vorzeichenlose ganze Zahlen sind. Objekte werden in dieser Klasse über eine Zählervereinbarung definiert. Zähler werden nicht automatisch zurückgesetzt, wenn ein Tests startet, womit es dem Testplan ermöglicht wird, Zählungen über das Testen vieler DUT zu sammeln. Es werden Verfahren bereitgestellt, um den Wert eines Zählers zurück zu setzen, zu inkrementieren und abzufragen. Dies ermöglicht eine Alternative zur Kategorieeinstufung, um Zählungen als Ergebnis des Laufs eines Tests zu bestimmen.

[0280] Der Test enthält vorzugsweise eine Elementvariable, m_modifiedCounters, die die Menge von Zählern ist, die durch den Lauf des Tests an einem DUT modifiziert werden. Diese Größe wird zu der leeren Größe beim Start des Testes initialisiert. An jeder Stelle, an der ein Aufruf von Inkrementierungszählern vorgenommen wird, wird ein Code generiert werden, um die nominierten Zähler zu dem Element m_modifiedCounters hinzuzufügen. So sammelt dieses Element alle diejenigen Zähler zusammen, die während der Ausführung eines Tests an einem DUT modifiziert waren.

C++ für das Ablaufobjekt

[0281] Sobald alle Ablaufelemente erzeugt worden sind, kann das Ablaufobjekt als ein C++ Objekt, wie nachstehend dargestellt, erzeugt werden:
Eine Rechenoperation zum Addieren eines Ablaufelements
Status addFlowItem(FlowItem*pFlowItem, bool isInitialFlowItem)
wird das angegebene Ablaufelement zu dem Ablauf addieren. Die Boolesche Algebra wird auf „Wahr" gesetzt, wenn dieses das anfängliche Ablaufelement des Ablaufs ist.

[0282] Eine Rechenoperation zum Ausführen des Ablaufs
Status executedFlow(unsigned int& result),

[0283] Diese wird vorzugsweise rückstellen, wenn der Ablauf zurückspringt, mit dem Ergebnis, dass der Ablauf ausgeführt wird. Die Wirkung dieser ist es, das Ausführen des Ablaufes mit dem anfänglichen Ablaufelement zu starten. Sie wird das Ausführen von Ablaufelementen solange beibehalten, wie das aktuelle Ablaufelement auf ein nächstes Ablaufelement zum Ausführen zurückspringt. Wenn das aktuelle Ablaufelement ein Ergebnis zurück überträgt, dann endet diese Rechenoperation mit diesem Ergebnis.

[0284] Daher besitzt der für einen Ablauf generierte C++-Code mehrere wiederholte Aufrufe an addFlowItem(), um addFlowItems zu dem Ablauf zu addieren. Die Rechenoperation executeFlow() wird stattfinden, wenn dieser Ablauf in dem Testplan zur Ausführung ausgewählt ist.

C3. Testklassen

[0285] Im Allgemeinen ist der Programmcode mehrheitlich Daten für einen Bausteintest und der Rest ist der Code des Testprogramms, der die Testmethodik realisiert. Die Daten sind DUT abhängig (z. B. Stromversorgungsbedingungen, Signalspannungsbedingungen, zeitliche Steuerungsbedingungen, usw.). Der Testcode besteht aus Verfahren zum Laden der bestimmten Bausteinbedingungen in Hardware von Testlaborgeräten und auch denjenigen, die benötigt werden, um die benutzerspezifischen Aufgaben (wie Datenerfassung, usw.) zu realisieren.

[0286] Wie es oben erläutert ist, sollte ein Testcode zur Erhöhung seiner Wiederverwendbarkeit unabhängig sein von irgendwelchen bausteinspezifischen Daten (z. B. Pinnamen, Ansteuerdaten, usw.) oder für den Bausteintest spezifischen Daten (z. B. Bedingungen für Gleichstromeinheiten, Messkontakte, Anzahl von Zielkontakten, Strukturfilename, Adressen von Strukturprogrammen, usw.). Wenn ein Code für einen Test mit Daten dieser Typen kompiliert wird, würde die Wiederverwendbarkeit des Testcodes abnehmen. Deshalb sollten dem Testcode beliebige bausteinspezifische Daten oder dem Bausteintest spezifische Daten, wie Eingaben während der Codeausführungszeit, extern zugänglich gemacht werden.

[0287] In dem Testsystem mit offener Architektur realisiert eine Testklasse, die eine Implementierung der ITest-Schnittstelle ist, die Trennung von Testdaten und Codes (und daher die Wiederverwendbarkeit des Codes) für einen speziellen Testtyp. Eine solche Testklasse könnte als „Schablone" für getrennte Fälle davon betrachtet werden, die voneinander nur auf der Basis von bausteinspezifischen und/oder für den Bausteintest spezifischen Daten abweichen. Die Testklassen sind in der Testplandatei spezifiziert. Jede Testklasse implementiert typischerweise einen speziellen Typ von Bausteintest oder Rechnerschaltung für Bausteintest. Zum Beispiel werden Parametertests für Funktion, Wechselstrom und Gleichstrom vorzugsweise durch getrennte Testklassen implementiert. Jedoch können in den Testplänen auch kundenspezifische Testklassen verwendet werden.

[0288] Testklassen ermöglichen dem Anwender, das Klassenverhalten zu konfigurieren, indem Parameter bereitgestellt werden, die genutzt werden, um die Optionen für einen speziellen Fall dieses Tests zu spezifizieren. Zum Beispiel wird ein Funktionstest jeweils die Parameter PList und TestConditions nehmen, um die Strukturliste zum Ausführen sowie die Ebenen- und Taktungsbedingungen für den Test zu spezifizieren. Das Be-

stimmen von unterschiedlichen Werten für diese Parameter (durch die Verwendung unterschiedlicher „Testblocks" in der Beschreibungsdatei des Testplans) ermöglichen es dem Benutzer, unterschiedliche Fälle eines Funktionstests zu erzeugen. Fig. 5 zeigt, wie unterschiedliche Testfälle **502** aus einer einzelnen Testklasse **504** abgeleitet werden würden.

[0289] Diese Klassen sollten so ausgeführt werden, dass sie es dem Kompilierer **400** ermöglichen, die Beschreibung der Tests und ihrer Parameter aus der Tesplandatei zu nehmen und einen genauen C++ Code zu erzeugen, der kompiliert und verknüpft werden kann, um das Testprogramm zu generieren. Testklassenfälle können Objekten hinzugefügt werden, die einen Testablauf zur Schaffung einer komplexen Ausführungsfolge von Bausteintests beschreiben.

C4. Ableitung aus ITest und IFlowable

[0290] Wie oben erwähnt, leiten sich Testklassen vom ITest ab. Diese können mit den oben erwähnten Regeln in C++-Klassen implementiert werden, die die ITest Schnittstelle implementieren. Zusätzlich zu den für die ITest Schnittstelle spezifizierten Verfahren stellen diese Klassen die testspezifische Intelligenz und Logik bereit, die zur Ausführung spezifischer Klassen von Bausteintests benötigt werden. Diese Klassen führen außerdem die IFlowable-Schnittstelle aus. Als Konsequenz davon können Fälle von Testklassen in den Ablaufelementen zum Abarbeiten von Tests verwendet werden.

Kundenwunschauslegung

[0291] Mechanismen der Kundenwunschauslegung werden bereitgestellt, um es Benutzern zu ermöglichen, C-Funktionen aufzurufen und ihre die Schnittstellen ITest und IFlowable implementierenden eigenen Klassen zu entwickeln.

Selbstbeobachtungsvermögen

[0292] Wenn ein Objekt einer Testklasse hinsichtlich seiner Verfahren und Signaturen abgefragt werden könnte, dann könnte verifiziert werden, dass die entsprechenden Parameter zur Einbeziehung in den generierten Quellencode verfügbar sind. Ein solches Merkmal würde sehr gut zur Fehlerprüfung und Gültigkeitserklärung während der Übersetzungsphase verwendbar sein. Wenn der Prüfingenieur einen Fehler hinsichtlich der Parameter oder der Anzahl (oder möglicherweise der Typen) von Argumenten zu diesen Parametern gemacht hat, könnte ihn die Übersetzungsphase auffangen und zum Zeitpunkt der Übersetzung eine bedeutsame Fehlermeldung zur Verfügung stellen, anstatt auf eine Kompilierzeit-Fehlermeldung vom C++-Kompilierer zu warten. Dies wäre für den Prüfingenieur nützlicher.

[0293] Selbstbeobachtung verweist auf die Fähigkeit, ein Objekt zu bitten, in sich hinein zu schauen und Informationen hinsichtlich seiner Attribute und Verfahren zurückzugeben. Einige Sprachen wie Java bewirken diese Fähigkeit als Teil der Sprache. Andere Sprachen, wie beispielsweise VisualBasic legen eine solche Anforderung Objekten auf, die mit ihr verwendet werden sollen. C++ trifft für dieses Merkmal keine Vorkehrungen.

[0294] Dieses Verfahren eignet sich außerdem gut dazu, sowohl vorgegebene Parameterwerte als auch Angaben von optionalen Parametern zur Verfügung zu stellen. Außerdem könnten dann, wenn diese Fähigkeit als ein Teil der Implementierung aller Testklassen vorgesehen ist, die Anwendungen der grafischen Benutzeroberfläche (GUI) diese Informationen auch nutzen, um Dialoge und andere Elemente der Anwenderschnittstelle aufzubauen, die den Ingenieuren helfen, einen effektiven Gebrauch dieser Klassen zu machen.

[0295] Diese Komplexitäten werden in einer Ausführung der Erfindung durch einen Mechanismus kompensiert, der anstelle einer vollständigen Innenschau ein Verfahren zur Verfügung stellt, das es dem Entwickler von Testklassen erlaubt, in einem einzelnen textbasierten Quellenfile (pro Testklasse) die allgemein zugänglichen Verfahren/Attribute der Testklasse festzulegen, die der Entwickler als diejenigen bezeichnet hat, die zum Parametrieren der Klasse erforderlich sind.

[0296] Es wird eine einzelne Quelle bevorzugt: Man würde nicht die Beschreibung der Parameterschnittstelle einer Testklasse in einem File und die C++ Schnittstellenbeschreibung in einem anderen unabhängigen File (Nachrichtenvorsatz) haben wollen und anschließend mit der Notwendigkeit belastet sein, beide Quellen synchronisiert zu halten. Zu diesem Zweck wird die auf „Text basierende" Beschreibung in einen Preheader-File für die Testklasse eingebettet, die durch den Kompilierer sowohl zur begrenzten Innenschau als auch zur Erzeugung des C++ Nachrichtenvorsatzes für die Testklasse genutzt wird. Der generierte C++ Header-File ist

der, der genutzt wird, um schließlich den Testklassen-C++ Code zu kompilieren.

Die Preheader

[0297] Die Verwendung von Headern in C++ ist bekannt. Weil C++ schwer grammatisch zu definieren und zu lesen ist, definiert eine Ausführung der Erfindung jedoch eine Syntax, die es einem Kompilierer erlaubt, eine C++-Ausgabe zu erzeugen, die von einem Entwickler für Testklassen als ein Nachrichtenvorsatz genutzt werden kann. Nach dieser Ausführung schreibt der Testklassen-Entwickler einen Preheader, der durch den Kompilierer **400** als ein Header-File ausgegeben wird, der Sichtbarkeit in die entsprechenden Testklassen oder andere Testentitäten erlaubt.

[0298] Das folgende Beispiel stellt das Konzept des Preheader-Files für eine Testklasse entsprechend der bevorzugten Ausführung der vorliegenden Erfindung dar. Betrachtet wird der folgende Auszug aus einem Quellenfile mit einem Test FuncTest1:

```
...
Testbedingung TC1
{
  Testbedingungsgruppe = TCG1, # zuvor
  definierte TCG für Ebenen
  Auswähler = min,
}

 Testbedingung TC2
 {
   Testbedingungsgruppe = TCG2, # Zuvor
   definierte TCG zur zeitlichen Steue-
rung
   Auswähler = min,
 }
 ...
 Teste Funktionstest FuncTest1
 {
   PListParam = patList1, # Zuvor defi-
nierte
   Strukturliste
   TestConditionParam = TC1,
   TestConditionParam = TC2,
 }
```

[0299] Der Kompilierer muss wissen, was ein Funktionstest erforderlich macht, um zu bestimmen, ob die Vereinbarung von FuncTest1 oben erlaubt ist. Anstatt die Kenntnis eines Funktionstests in den Kompilierer einzubauen, kann die Definition dessen, was ein Funktionstest erfordert, in dem Preheader spezifiziert werden.

[0300] Es wird angenommen, dass ein Funktionstest eine C++-Klasse mit den Basisklassen Test1 und Test2 sowie Elementen ist, die eine PList und eine Matrix von Testbedingungen sind. Der Kompilierer muss etwas über die Typen der Elemente von Funktionstest verstehen, um zu erkennen, dass die oben erwähnte Vereinbarung von FuncTest1 zulässig ist.

[0301] Außerdem muss ein C++ Header für die Klasse Funktionstest konstruiert werden, um eine C++ Ob-

jektvereinbarung für FuncTest1 zu generieren. Dies erfordert, dass der Kompilierer außerdem etwas über die Basisklassen der Funktion Testklasse, die Namen ihrer Elemente und andere derartige Informationen versteht.

[0302] Die Preheader-Untersprache nach einer Ausführung der Erfindung versorgt den Kompilierer mit den Informationen, die er benötigt, um sowohl die Legalität von Vereinbarungen zu erkennen als auch C++-Header und Objektvereinbarungen zu generieren, die einer Vereinbarung entsprechen.

[0303] Zu beachten ist, dass ein Funktionstest ein einfacher Typ ist (sofern es Parametrierung betrifft), und folglich eine ganz einfache Beschreibung zur Parametrierung verwenden würde. So könnte man einen Preheader, FunctionalTest.ph, schreiben, der die oben erwähnte Parametrierung wie folgt unterstützt (vorausgesetzt, dass die Preheader für die Basistestklassen Test1 und Test2 verfügbar sind):

```
    1 Version 1,0;
    2 #
    3 # Preheader der Parametrierungsspezifikation
für
        Funktionstest
    4 #
    5 Import Test1.ph,  # Für Basisklasse Test1
    6 Import Test2.ph,  # Für Basisklasse Test2
    7 Testklasse = Funktionstest, # Der Name dieser
Testklasse
    8 Allgemein zugängliche Basen = Test1, Test2, #
Liste von
      allgemein zugänglichen Basisklassen
    9 # Die Parameterliste oder „Parameterblock"
   10 Parameter
   11 {
   12   # Die folgende Vereinbarung bestimmt, dass
ein
        # Funktionstest aufweist
   13   # - einen Parameter vom Typ PList
   14   # - [dargestellt durch C++ Typ-
Tester::Strukturbaum]
   15   # - gespeichert in einem m_pPatList bezeich-
neten
        # Element
   16   # - eine Funktion, um sie setPatternTree be-
nannt zu
           setzen.

   17   # - eine Parameterbeschreibung für die GUI
zur
        # Verwendung als Toolspitze
   18   PList PListParam
```

```
19  {
20     Kardinalität = 1,
21     Attribut = m_pPatList,
22     Funktion setzen = setPatternTree,
23     Beschreibung = „Der Parameter PList für
    einen
       Funktionstest",
24  }
25  #
26  # Die folgende Vereinbarung bestimmt, dass
    ein Funktionstest aufweist
27  # - 1 oder mehrere Parameter vom Typ TestCon-
    dition
28  # -[durch C++ Typ-Tester:: TestCondition dar-
    gestellt]
29  # - in einem m_testCondnsArray benannten Ele-
    ment gespeichert
30  # - eine Funktion, um sie addTestCondition
    benannt zu setzen.
31  # - eine Parameterbeschreibung für die GUI
    zur Verwendung als Toolspitze
32  # Die [Implementierungs-]klausel zwingt die
    Übersetzungsphase dazu,
33  # eine gegebene Implementierung dieser Funk-
    tion zu generieren.
34  #
35  TestBedingung TestConditionParam
36  {
37    Kardinalität = 1-n,
38    Attribut = m_testCondnsArray,
39    Funktion setzen = addTestCondition [Imple-
    mentieren],
40    Beschreibung = „Der Parameter Testbedingung
    für
      einen Funktionstest",
41  }
```

```
42  }
43  #
44  # Der nachstehende Abschnitt ist Teil des
    Preheader,
45  der eine Umschaltung in den C++ Code ist.
    Dies wird als „Template-Block" bezeichnet.
46  #
47  # Alles in diesem Abschnitt wird in der gene-
    rierten
48  # Header Datei wortwörtlich reproduziert bis
    auf
49  # „$Class", „$Inc", „$ParamAryTypes", „$Para-
    mAttrs",
    # „$ParamFns" und „$ParamImpuls".
50  #
51  # Zu beachten ist, dass keine mit dem „#"
    Zeichen
52  # beginnende Kommentare innerhalb des folgen-
    den
    # Abschnitts unterstützt werden.
53  #
54  CPlusPlusBegin
55  $Inc
56  Namensraum
57  {
58  Klasse $Class
59  {
60  //Matrixtypen für Parameterspeicherung:
61  $ParamAryTypes
62  allgemein zugänglich:
63  virtuelle Lücke preExec(),
64  virtuelle Lücke exec(),
65  virtuelle Lücke postExec(),
66  $ParamFns
67  ...
68  betriebsintern:
```

```
69    Fließkommazahl doppelter Genauigkeit
      m_someVar,
70    $ParamAttrs

71    ...

72    },

73    ...
74    $ParamImpuls
75    }// Namensraum beenden

76    CPlusPlusEnd
```

C++ für parametrierte Testklassen

[0304] Wenn der Kompilierer eine Preheader Datei verarbeitet, baut er die Werte der Kompiliervariablen wie beispielsweise $Inc, $Class, $ParamAryTypes und andere auf. Dies aktiviert ihn dann, den folgenden C++ Header zu erzeugen, indem der oben erwähnte C++ Code wortwörtlich generiert und in den Werten der Kompiliervariablen $Inc, $Class, usw. an den angegebenen Stellen erweitert wird. Für den Funktionstest.ph erzeugt er den folgenden C++-Header-File Functional-Test.h für die Funktionstestklasse.

```
1 # Zeile 7 „./Funktionstest.ph"
2 # schließe ein <ITest.h>
3 # Zeile 5 „./Funktionstest.ph"
4 # schließe ein <Test1.h>
5 # Zeile 6 „./Funktionstest.ph"
6 # schließe ein <Test2.h>
7 # Zeile 55 „./Funktionstest.ph"
8 # schließe ein <Vektor>
9 # Zeile 55 „./Funktionstest.ph"
```

```
 10 # schließe ein <Level.h>
 11 # Zeile 55 „./Funktionstest.ph"
 12 # schließe ein <TestCondnGrp.h>
 13 ...
 14 # Zeile 56 „./Funktionstest.ph"
 15 Namensraum
 16 {
 17 # Zeile 7 „./Funktionstest.ph"
 18 Klasse Funktionstest: allgemein zugänglicher I-
Test,
 19 # Zeile 8 „Funktionstest.ph"
 20                              allgemein zugängli-
cher Test1,
 21 # Zeile 8 „./Funktionstest.ph"
 22                              allgemein zugängli-
cher Test2,
 23 # Zeile 59 „./Funktionstest.ph
 24 {
 25 // Matrixtypen für Parameterspeicherung:
 26 # Zeile 61 „./Funktionstest.ph"
 27 allgemein zugänglich:
 28 # Zeile 37 „./Funktionstest.ph"
 29    typedef std::vector<#Tester::TestCondition*>
    TestConditionPtrsAry_t;
 30 # Zeile 62 „./Funktionstest.ph"
 31 allgemein zugänglich:
 32  virtuelle Lücke preEcec(),
 33  virtuelle Lücke exec(),
 34  virtuelle Lücke postExec(),
 35  allgemein zugänglich :
 36 # Zeile 7 „./Funktionstest.ph"
 37   Lücke setName(OFCString &name); # Automatisch
für alle
      Tests
 38 # Zeile 22 "./Funktionstest.ph"
 39   Lücke set PatternTree(PatternTree *),
```

```
 40 # Zeile 23 „./Funktionstest.ph"
 41     String getPListParamDescription()const,
 42 # Zeile 39 „./Funktionstest.ph"
 43   Lücke addTestCondition(TestCondition) *),
 44 # Zeile 40 „./Funktionstest.ph"
 45   Lücke getTestConditionParamDescription()
const,
 46 # Zeile 67 „./Funktionstest.ph"
 47   ...
 48  betriebsintern:
 49  Fließkommazahl doppelter Genauigkeit m_some
Var,
 50  # Zeile 70 „./Funktionstest.ph"
 51  betriebsintern:
 52  # Zeile 7 „./Funktionstest.ph"
 53      OFCString m_name, # Automatisch für al-
le Tests
 54  # Zeile 21 „./Funktionstest.ph
 55      Tester::PatternTree *m_pPatList,
 56  # Zeile 38 „./Funktionstest.ph
 57      TestConditionPtrsAry_t
m_testCondnsArray,
 58  # Zeile 71 „./Funktionstest.ph
 59    ...
 60  },
 61  ...
 62  # Zeile 7 „./Funktionstest.ph"
 63  mitlaufende Lücke
 64  # Zeile 7 „./Funktionstest.ph"
 65  Funktionstest::setName(OFCString &name)
 66  # Zeile 74 „./Funktionstest.h"
 67  {
 68     m_name = name,
 69     Rücksprung,
 70  }
 71  # Zeile 39 „./Funktionstest.ph"
```

```
  72  mitlaufende Lücke
  73  # Zeile 39 „./Funktionstest.ph"
  74  Funktions-
test::addTestCondition(TestCondition *arg)
  75  # Zeile 74 „./Funktionstest.ph"
  76  {
  77        m_testCondnsArray-push_back(arg),
  78        Rücksprung,
  79  }
  80  # Zeile 23 „./Funktionstest.ph"
  81  mitlaufende Lücke
  82  Tester:: Sequenz Funktions-
      test::getPListParamSecription()
  83  {
  84    „Den Parameter PList für einen Funktions-
      test" rücksetzen,
  85  }
  86  # Zeile 40 „./Funktionstest.ph"
  87  mitlaufende Lücke
  88  Tester::Sequenz Funktions-
      test::getTestConditionParamDescription()
  89  {
  90  „Den Parameter TestCondition für einen
      Funktionstest" rücksetzen,
  91  }
  92  # Zeile 75 "./Funktionstest.ph"
  93  } // Namensraum beenden
```

[0305] Wie früher beschrieben, aktiviert dieser Preheader, dass der Kompilierer die Gültigkeit einer Funktionstestvereinbarung prüft, einen Code für sie generiert und einen C++ Header generiert, der von ihr benötigt werden würde.

[0306] Als ein Beispiel betrachten wir die früher gegebene Funktionstestvereinbarung, die der Bequemlichkeit halber unten reproduziert ist:

```
  Test Funktionstest FuncTest1
  {
    PListParam = patList1, # zuvor definierte
Strukturliste
          TestConditionParam = TC1,
          TestConditionParam = TC2,
        }
```

[0307] Der C++ Header, der dafür durch den Kompilierer generiert werden würde, ist oben gegeben. Der Kompilierer würde den folgenden Code für das oben erwähnte Funktionstest-Konstrukt generieren:
Funktionstest FuncTest1,
Funktionstest1.setName(„FuncTest1"),
Funktionstest1.setPatternTree (6patList1),
Funktionstest1.addTestCondition(&TC1),
Funktionstest1.addTestCondition(&TC2),

[0308] Zu beachten ist auch der Name, der für die Beschreibungsfunktion generiert wird. Jeder Xxx bezeichnete Parameter wird mit einer Elementfunktion verknüpft:
Status getXxxDescription() const,
die die Sequenz mit einer Beschreibung für die Toolspitze, die die GUI verwenden kann, rücksetzt.

Andere Preheader-Merkmale

[0309] Der Preheader unterstützt einige andere anwenderdefinierte Verzeichisse als einen zusätzlichen Typ. Dieser ermöglicht der GUI, eine Drop-down-Liste von möglichen Auslesen zur Verfügung zu stellen, die zum Setzen des Wertes eines speziellen Parameters verwendet werden könnte. Außerdem bewirkt der Preheader ein Merkmal zum Verknüpfen einer Anzahl von Parametern, die man sich als eine Tabelle vorstellen kann. Zum Beispiel kann es angebracht sein, eine Matrix von „Eigenschaften" als einen verknüpften Satz einer Matrix von Sequenzen für die Namen und eine Matrix von ganzen Zahlen für die Werte zu implementieren. Eine leichte Möglichkeit zum Implementieren dieser Eigenschaft ist, eine Matrix von kundenspezifischen Typen (später erörtert) zu nutzen. Dies erfordert jedoch, dass der Anwender einen kundenspezifischen Preheader für den Gebrauch schreibt. Beide dieser Merkmale sind in dem folgenden Beispiel dargestellt:

```
  # ------------------------------
  # File FooBarTest.ph
  #
  # Parametrierung des Spezifizierungs-Preheader
für
    Foobar-Test kundenspezifischer Testklassen
  # ------------------------------


  Version 1.0,

  Import Test1.ph            # Für Basisklassen-
Test1
  TestClass = FoobarTest,    # Der Name dieser
Testklasse
  PublicBases = Test1,       # Liste von gemein-
schaftlich
```

```
                                nutzbaren Basis-
klassen

  # Die Parameterliste:
  Parameter
  {
     # Ein spezifizierter Typ
     Enum wässrig = Ja, Vielleicht, Möglicherwei-
se, Mag
                    sein, Mag nicht sein, Nein

     # Definiere einen wässrigen Parameter.
     Wässrig WW
     {
        Kardinalität = 1,
        Attribut = m_ww,
        setze Funktion = setWw,
        Beschreibung = „Der WW Parameter für ei-
nen Test
                       Foobar",
     }
     # Diese Klasse besitzt eine Matrix von Paa-
ren von
     # Namen-Zahlen, die in der Klasse wiederge-
geben wird.
     ParamGroup
     {
         Kardinalität = 0-n,

         # Das Namensfeld in dieser Matrix ist:
         # - vom Typ Sequenz
         # - [dargestellt durch C++ Typ Tes-
ter::String]
         # - gespeichert in einem mit m_NameArray
benannten
         # - Element
```

```
        # - Funktion, um es benannten addName zu
setzen.
        # - Parameterbeschreibung für die GUI
zur
            Verwendung als eine Toolspitze
        Sequenz Name
        {
           Attribut = m_NameArray,
           setze Funktion = addName,
           Beschreibung = "Ein Name mit einem
Wert",
        }

        # Das Zahlenfeld in dieser Matrix ist:
        # - vom Typ einer ganzen Zahl,
        # - [dargestellt durch C++ Typ int]
        # - gespeichert in einem mit
m_NumberArray
            benannten Element
        # - Funktion, um es benannte addNumber
zu setzen.
        # - Parameterbeschreibung für die GUI
zur
            Verwendung als eine Toolspitze

        Ganzzahlige Nummer
        {
           Attribut = m_NumberArray,
           setze Funktion = addNumber,
           Beschreibung = „Der Wert des Namens",
        }
    }

    # Die folgende Vereinbarung legt fest, dass
ein
```

```
    # Funktionstest einen Parameter vom Typ PList
besitzt
    # - [dargestellt durch C++ Typ Tes-
ter::PatternTree]
    # - gespeichert in einem mit m_pPatlist be-
nannten
    #   Element
    # - eine Funktion, um ihn benannten setPat-
ternTree zu
        setzen.
    # - Parameterbeschreibung für die GUI zur
Verwendung
    #   als eine Toolspitze
    PList PListParam
    {
       Kardinalität = 1,
       Attribut = m_pPatList,
       setze Funktion = setPatternTree,
       Beschreibung = „Der Parameter PList für
einen
                      Funktionstest",
     }

     # Die folgende Vereinbarung bestimmt, dass
ein
     # Funktionstest
     # - 1 oder mehrere Parameter vom Typ
TestCondition
     #   besitzt
     # - [dargestellt durch C++ type Tes-
ter::TestCondition]
     # - in einem mit m_testCondnsArray benannten
Element
     #   gespeichert ist
     # - eine Funktion besitzt, um ihn benannten
     #   addTestCondition zu setzen.
```

```
    # Die [Implementierungs]klausel bewirkt die
    # Umsetzungsphase davon, um eine
    # Vorgabeimplementierung dieser Funktion zu
erzeugen.
    #
    Testbedingung TestConditionParam
    {
      Kardinalität = 1-n,
      Attribut = m_testCondnsArray,
      setze Funktion = addTestCondition  [Imple-
mentierung]
      Beschreibung = „Der Parameter TestConditi-
on für
      einen Funktionstest“,
    }
  }

  CPlusPlusBegin
  $Inc
  Namensraum
  {
  Klasse $Class
  {
  // Matrixtypen für Parameterspeicherung:
  $PAramAryTypes
  gemeinschaftlich nutzbar:
        virtuelle Lücke preExec(),
        virtuelle Lücke(),
        virtuelle Lücke postExec(),
        $ParamFns
        //...

  betriebsintern:
       Double m_someVar,
       $ParamAttrs
       //...
```

```
 },
 //...
 $ParamImpls
}// Namensraum beenden
CPlusPlusEnd
```

[0310] Es muss beachtet werden, dass Namen-Nummer-Paare von kundenspezifischem Typ vereinbart worden sein könnten und ein einzelner Matrixparameter dieses kundenspezifischen Typs verwendet worden sein könnte, um den gleichen Effekt wie die oben erwähnte ParamGroup von Parametern zu haben. Das oben dargestellte Verfahren ist ein Vorteil, der die Notwendigkeit vermeidet, einen kundenspezifischen Typ zu vereinbaren.

C5. Kundenspezifische Funktionsvereinbarungen

[0311] Dies ermöglicht dem Anwender, kundenspezifische Funktionen aufzurufen, wenn ein Ablaufübergang stattfindet. Kundenspezifische Funktionen werden durch Preheader wie folgt vereinbart:

```
     # ----------------------------
     # File MyFunctions.ph
     #
     # Parametrierung des Spezifikations-
Preheader für
       MyFunctions
     # ----------------------------

     Version 1.0,

     Funktionen = MyFunctions,  # Der Name dieser
Gruppe
                                  von Funktionen
```

```
     # Vereinbare die folgende C++ Funktion im
Namensraum
     # MyFunction, um das Minimum von zwei Werten
zu
     # bestimmen.
     #  // Setze das Minimums von x, y zurück
     #  Fließkommazahl doppelter Genauigkeit
        MyRoutines::Min
     #             (ITestPlan* pITestPlan, int&
x, int& y),
     Ganze Zahl Min(Integer x, Integer y),

     # Vereinbare die folgende C++ Funktion im
Namensraum
     # UserRoutines, um den Mittelwert einer Mat-
rix zurück
     # zu setzen.
     #  // Setze den Mittelwert der Matrix zurück
     #  Fließkommazahl doppelter Genauigkeit
        MyRoutines::Avg
     #     (ITestPlan* pITestPlan, double* a,
const int
            a_size),
     # Die C++ Funktion wird mit a und a'Length
aufgerufen
     # Fließkommazahl doppelter Genauigkeit
Avg(Double
       a[]),

     # Vereinbare die folgende C++ Funktion im
Namensraum
     # UserRoutines, um den Kennungscode des DUT
     # und eine Meldung zu drucken
     #  // Setze den Mittelwert der Matrix zurück
     #  Fließkommazahl MyRoutines::Print
```

```
     #          (ITestPlan* pITestPlan, String*
msg,
                unsigned int&dutId),
     # Die C++ Funktion wird mit a und a'Length
aufgerufen
     # Lücke Print(String msg, UnsignedInteger
dutId),
```

[0312] Typisch ist, dass für die oben erwähnten Vereinbarungen ein C++ Abschnitt zur Verfügung gestellt werden muss, weil der Kompilierer diese Vereinbarungen in üblicher Weise erweitern wird. Der Anwender ist natürlich verantwortlich für die C++ Implementierung dieser Funktionen. Zu beachten ist, dass alle oben erwähnten Funktionen wahrscheinlich eine Hinweismarke ITestPlan als impliziten ersten Parameter nehmen werden. Diese Hinweismarke sorgt für den Zugriff des Funktionsschreibers auf den Zustand S im Testplan. Zum Beispiel könnte der Funktionsschreiber die Schnittstelle ITestPlan nutzen, um auf den aktuellen Ablauf, das aktuelle Ablaufelement in dem Ablauf, die aktuelle Ergebnisklausel, Werte von UserVars und andere solche Informationen zuzugreifen. Bestimmte vom Tester definierte Funktionen sind zum Gebrauch in dem File Functions.ph verfügbar.

```
          Version 1.2.3;

          #
          # File Functions.ph
          #
          Funktionen = Functions, # Name dieser
Gruppe von
                                  Funktionen

          # Vereinbare die folgende C++ Funktion
im
          # Namensraum Funktionen

          # Setze den Kennungscode des aktuellen
DUT, der
          # durch den Aufrufer getestet wird,
zurück.
          # vorzeichenlose Ganzzahl GetDUTID(),
```

C++ für kundenspezifische Funktionsvereinbarungen

[0313] Der C++ Code, der durch den Kompilierer für MyFunctions oben generiert werden würde, soll einfach einige Funktionen im Namensraum MyFunctions vereinbaren:

```
      Namensraum MyFunctions
      {
         double Min(ITestPlan* pITestPlan, int&
x, int& y),
         double Avg(ITestPlan* pITestPlan, doub-
le* a, const
         int a_size),
         void Print(ITestPlan* pITestPlan, char*
Msg,
         unsigned int dutID),
      }
```

[0314] Diese Funktionen werden aus einem Ablauf aufrufbar sein.

C6. Kundenspezifische Flowables

[0315] Es ist außerdem möglich, einen Preheader zu erzeugen, indem die den Preheader nutzende C++ Flowable-Schnittstelle implementiert wird. Dies ermöglicht einem Benutzer, kundenspezifische Flowables zu definieren, die in einem Ablaufelement arbeiten können. Nachstehend gezeigt ist ein Preheader für die benutzerdefinierte Flowable MyFlowable:

```
        # --------------------
        # File MyFlowable.ph
        #
        # Parametrierung des Spezifizierungs-
Preheaders
          für MyFlowable
        # ----------------------------


        Version 1.2.4;

        FlowableKlasse = MyFlowable, # Der Name
dieser
                                     kundenspezifi-
schen Klasse

        # Die Parameterliste:
        Parameter
        {
          # Die folgende Vereinbarung bestimmt,
dass
          # eine MyFlowable besitzt
          # - 1 optionalen Parameter Int1 vom
Typ Ganzzahl
          # - [dargestellt durch C++ Typ int]
          # - gespeichert in einem m_int1Val be-
nannten
```

```
            Element
          # - eine Funktion, um sie benannte se-
tInt1Val
            zu setzen.
          Ganzzahl Int1
          {
            Kardinalität = 0-1,
            Attribut = m_int1Val
            Setze Funktion = setInt1Val
          }
          # Die folgende Vereinbarung bestimmt,
dass eine
          # MyFlowable besitzt
          # - 1 verbindlichen Parameter Int2 vom
Typ
            Ganzzahl
          # - [dargestellt durch C++ Typ int]
          # - gespeichert in einem m_int2Val be-
nannten
            Element
          # - eine Funktion, um sie benannte se-
tInt2Val
            zu setzen.
          Ganzzahl Int2
          {
            Kardinalität = 1
            Attribut = m_int2Val,
            Setze Funktion = setInt2Val,
          }
          # Die folgende Vereinbarung bestimmt,
dass eine
          # MyFlowable besitzt
          # - einen oder mehrere Parameter vom
Typ Sequenz
          # - [dargestellt durch C++ Typ Tes-
ter::String]
```

```
            # - gespeichert in einem
m_stringArrVal
                benannten Element
            # - eine Funktion, um sie benannte
addStringVal
                zu setzen.
            Sequenz StringItem
            {
                Kardinalität = 1-n,
                Attribut = m_stringArrVal,
                Setze Funktion = addStringVal,
            }

            # Die folgende Vereinbarung bestimmt,
dass eine
            # MyFlowable besitzt
            # - einen einzelnen Parameter PList
            # - [dargestellt durch C++ Typ Tes-
ter::PList]
            # - gespeichert in einem m_plist be-
nannten
                Element
            # - eine Funktion, um sie benannte
setPListParam
                zu setzen.
            PList PListParam
            {
                Kardinalität = 1,
                Attribut = m_plist,
                Setze Funktion = setPListParam,
            }
        }

        #
        # Der Abschnitt unten ist ein Teil des
Preheaders,
```

```
          der eine Umschaltung in den C++ Code
ist.
         #
         # Alles in diesem Abschnitt wird wort-
wörtlich im
         # Headerfile reproduziert mit Ausnahme
von
         # „$Class", „$Inc", „$ParamAryTypes",
         # „$ParamAttrs", „$ParamFns" und „$Para-
mImpls".
         #
         # Zu beachten ist, dass innerhalb des
folgenden
         # Abschnitts keine mit dem ‚#' Zeichen
beginnenden
         # Kommentare unterstützt werden.
         #
         CPlusPlusBegin
         $Inc
         Namensraum
         {
         Klasse $Class
         {
         // Matrixtypen zur Speicherung von Para-
metern:
         $ParamAryTypes
         gemeinschaftlich nutzbar:
               virtuelle Lücke preExec(),
               virtuelle Lücke exec(),
               virtuelle Lücke postExec(),
               $ParamFns
               //...
            betriebsintern :
               Double m-someVar,
               $ParamAttrs
               //...
```

```
},
//...
$ParamImpls
}// Namensraum beenden
CPlusPlusEnd
```

[0316] Es gibt mehrere Klassen, die die IFlowable-Schnittstelle implementieren. Diese umfassen:

1. Abläufe zum Programmladen, die prüfen werden, ob ein Testplan innerhalb der aktuellen Testerkonfiguration ausgeführt werden kann.
2. Abläufe zum Strukturladen, die spezifische Strukturen und Strukturlisten laden werden.
3. Abläufe zur Initialisierung, die Hardware und Software in einen bekannten Zustand stellen, globale Variable laden und andere Initialisierungs- und Validierungsfunktionen vornehmen werden.
4. Andere allgemein nützliche Testabläufe.

C7. Kundenspezifische Typen

[0317] Die frühere Erörterung über Parametrierung von Testklassen berücksichtigte nur Testklassenparameter von bekannten Typen, nämlich Basistypen und testerdefinierte Typen wie beispielsweise PLists und TestConditions. Zur Flexibilität durch Benutzer ist es wichtig, eine Erweiterungsmöglichkeit von Typen bereitzustellen, wodurch Typen (die von vorn herein dem Kompilierer unbekannt sind) erzeugt und genutzt werden können. Kundenspezifische Typen (CT) werden in den Custom Types definiert werden. Diese können genutzt werden, um Typen zu definieren, die den Structs der Rechner-Programmiersprache C entsprechen (auch als Plain Old Data Typen oder POD bezeichnet, die völlig anders als ihre Namensvetter in C++ sind) sowie Typen, die Typedefs der Rechner-Programmiersprache C für Funktionssignaturen entsprechen, zu definieren. Ein getrennter File mit Benutzertypen wird die Erweiterung .ctyp besitzen. Hier ist ein Beispiel einer Vereinbarung der Benutzertypen gemäß der bevorzugten Ausführung der vorliegenden Erfindung:

```
            # -------------------------------
            # File MyCustomTypes.ctyp
            # -------------------------------

     Version 1.0.0,

     Kundenspezifische Typen
     {
        # Ein strukturierter Typ Plain-Old-Data
        Pod Foo (Variable)
        {
           Sequenz  S1, # Sequenz ist ein Stan-
dardtyp
           Ganze Zahl I1,   # ... und so ist gan-
ze Zahl
           Sequenz  S2,
        }

        # Ein weiterer strukturierter Typ, der
Foo nutzt
        Pod Bar
        {
           Foo         Foo1,
           Sequenz     S1,
           Foo         Foo2,
        }
```

```
        #
        # Eine Hinweismarke auf eine Funktion.
        #       Typ rücksetzen: ganze Zahl
        #       Parameter. Ganze Zahl, ganze Zahl
        #
        Routine BinaryOp(Integer, Integer) setzt
Integer
        zurück,

        #
        # Weitere Hinweismarke auf eine Funktion.
        #       Typ rücksetzen: Lücke
        #       Parameter: ganze Zahl
        #
        Routine Unary()p(Integer) setzt Lücke zu-
rück,

        #
        # Eine Hinweismarke auf eine Funktion,
die keine
        # Parameter annimmt und keinen Wert rück-
setzt.
        #
        Routine NilOp() setzt Lücke zurück;
      }
```

C++ für kundenspezifische Typen

[0318] Die oben dargestellte Vereinbarung CustomTypes wird durch den Kompilierer in den folgenden C++ Code übersetzt:

```
Namensraum CustomTypes
{
    struct Foo
```

```
   {
      Tester::Sequenz    S1,
      Int                I1,
      Tester::Sequenz    S2
   },
   struct Bar
   {
      Foo                Foo1,
      Tester::Sequenz    S1
      Foo                Foo2,
   }
   typedef int(*BinaryOp) (int&,
int&),
   typedef void(*UnaryOp)(int),
   typedef void(*NullaryOp)(),

}
```

[0319] Objekte dieses Typs können an Testklassen als Parameter weitergegeben werden, wie es als nächstes gezeigt wird.

Verwendung von kundenspezifischen Typen als Testklassenparameter

[0320] Es wird der Fall betrachtet, bei dem der Benutzer eine Erweiterung auf einen Test besitzt, die, zusätzlich zu Strukturlisten und Testbedingungen, mit anderen Klassenobjekten sowie willkürlichen Objekten (d. h. benutzerdefinierten) Objekten, die innerhalb eines CustomTypes enthaltenden Files (d. h. einen .ctyp-File) definiert sind, initialisiert werden müssen. Zum Beispiel wird angenommen, dass der Anwender die im File MyTestCTs.ctyp definierten kundenspezifischen Typen (CT) verwenden will:

```
# File MyTestCTs.ctyp
Version 1.0,

Kundenspezifische Typen
{
  Pod Foo
  {
     Sequenz Name,
     PList  patternList,
  }

  Pod Bar
  {
     Foo   someFoo,
   Fließkommazahl doppelter Genau-
igkeit dVal,
  }
  Routine BinaryOp(Integer, Inte-
ger)
  return Integer
}
```

[0321] Alles, was der Benutzer zur Nutzung der oben erwähnten Typen tun muss um die oben erwähnten Typen zu nutzen, ist, den oben erwähnten File in seinem Testklassen-Preheader zu importieren. Weil der Kompilierer kundenspezifische Typen (CT) übersetzt, die so definiert sind, sind für ihn deshalb die Definitionen für Foo und Bar verfügbar, wenn er den Testklassen-Preheader bearbeitet. Außerdem definiert der Kompilierer zwei Structs der Rechner-Programmiersprache C, Struct Foo und Struct Bar, die jeweils den oben erwähnten Typen Foo und Bar entsprechen, deren Definitionen in den File myTestCTs.h gesetzt werden. Die Anweisung Import für myTestCTs.ctt bewirkt, dass der File myTestCTs.h in den generierten C++ Header der Testklasse # einbezogen wird. Das folgende Beispiel veranschaulicht diesen Prozess. Zuerst wird die Vereinbarung für den Test in dem Testplan betrachtet (die Vereinbarungen für Strukturlisten und Testbedingungen wurden der Deutlichkeit halber weggelassen):

```
            ...
            Import MyFunctions.ph,
            Import MyCustomTypes.ctyp,
            ...
            # Der Block CustomVars definiert die
Variablen
            # der früher definierten kundenspezi-
fischen
            # Typen.
            CustomVars
            {
              ...
              Bar bar1 =
              {
               {„Dies ist eine Foo", somePat-
List}, #someFoo
                3,14159
#dVal
              }
              #
              # Funktionsobjekt, das ein Binär-
operator ist.
              # Der Name auf der rechten Seite
der
              # Zuordnung ist eine in MyFunctions
              # vereinbarte Routine, für die der
Benutzer
              # natürlich eine Implementierung
vorsehen

   # muss.
   BinaryOp bop1 = MyFunctions.Min,
}
...
Test MyFancyTestMyTest1
{
   ...
   BarParam = bar1,
   BinaryOpParam = bop1,
}
...
```

[0322] Im oben genannten Beispiel ist in einem Testplan ein Block CustomVars enthalten. Ein getrennter File mit Variablen individueller Fertigung wird die Erweiterung .cvar haben. Der Anwender würde einen Preheader für MyFancyTest schreiben, der die oben erwähnte Parametrierung (die Parametrierungsvereinbarungen für Strukturlisten und Testbedingungen wurden der Deutlichkeit halber weggelassen) wie folgt unterstützt:

```
      # --------------------------
      # Datei MyFancyTest.ph
      #
      # Parametrierung des Spezifizierungs-
Preheader
      # für MyFancyTest
      # --------------------------

      Version 1.0.2;

      Import MyCustomTypes.ctyp, # Für in My-
FancyTest
                                        verwendete
CTs
```

```
      Import FunctionalTest.ph,  # Für Basisklas-
se
                                   Funktions-
test
      Testklasse = MyFancyTest,      # Der Name
dieser
                                   Testklasse
      Gemeinschaftlich nutzbare    # Liste von
      Basen = FunctionalTest,        gemein-
schaftlich
                                   nutzbaren
                                   Basisklas-
sen
      # Die Parameterliste:
      Parameter
      {
         # Die folgende Vereinbarung bestimmt,
dass ein
         # MyFancyTest aufweist
         # - optionale Matrix von Parametern des
         #   kundenspezifischen Typs Bar
         # -[dargestellt durch C++ Typ kundenspezifi-
sche
         #   Typen::Bar]
         # - gespeichert in einem m_barsArray be-
zeichneten
         #   Element
         # - eine Funktion, um sie addBarParam be-
nannt zu
             setzen.
         # Eine Implementierung wird für addBarParam
           generiert.
         BarBarParam
         {
              Kardinalität = 0-n,
              Attribut = m_barsArray,
```

```
            Setze Funktion = addBarParam [Imple-
ment],
        }

        # Die folgende Vereinbarung bestimmt, dass
ein
        # MyFancyTest aufweist
        # - einen optionalen Parameter vom kunden-
spezifischen
          Typ BinaryOp
        # - [dargestellt durch C++ Typ kundenspezi-
fische
            Typen::BinaryOp]
        # - gespeichert in einem m_binaryOp bezeich-
neten
        #   Element
        # - eine Funktion, um sie setBinaryOpParam
benannt zu
            setzen.
        # Eine Implementierung wird für setBinaryOp-
Param
          generiert werden.
        BinaryOp BinaryOpParam
        {
          Kardinalität = 0-1,
          Attribut = m_binaryOp,
          Setze Funktion = setBinaryOpParam [Imple-
ment],
        }
      }

      CPlusPlusBegin

      $Inc
      Namensraum
      {
```

```
        Klasse $Class
        {
        $ParamAryTypes
        gemeinschaftlich nutzbar:
                virtuelle Lücke preExec(),
                virtuelle Lücke exec(),
                virtuelle Lücke postExec(),
                $ParamFns
                //...

        betriebsintern :
             Fließkommazahl doppelter Genauigkeit
m_someVar,
             $ParamAttrs
             //...
        },
             //...
             $ParamImpls
             }// Namensraum beenden
             CPlusPlusEnd
```

C++ für kundenspezifische Testklassen unter Verwendung von kundenspezifischen Typen

[0323] Sobald der Kompilierer diesen Preheader-File verarbeitet hat, wird er schließlich den folgenden C++ Header-File für die MyFancyTest-Klasse, MyFancyTest.h, erzeugen:

```
# einschließen <MyCustomTypes.h>
# einschließen <Test.h>
# einschließen <FunctionalTest.h>
...
Namensraum
```

```
        {
        Klasse MyFancyTest: gemeinschaftlich
nutzbar ITest,
                    gemeinschaftlich nutzbar
Funktionstest
        {
        gemeinschaftlich nutzbar:
           typedef std::vector<CustomTypes::Bar
*>BarAry_t,

        gemeinschaftlich nutzbar:
           virtuelle Lücke preExec(),
           virtuelle Lücke exec(),
           virtuelle Lücke postExec(),

        gemeinschaftlich nutzbar :
           Lücke setName(OFCString &name), # Au-
tomatisch
                                              für
alle Tests
           Lücke setPatternTree(PatternTree *),
           Lücke addTestCondition(TestCondition
*),
           Lücke addBarParam(CustomTypes::Bar *),
           Lücke setBinaryOpPa-
ram(CustomTypes::BinaryOp *),

           ...

        betriebsintern:
           Fließkommazahl doppelter Genauigkeit
m_someVar,

        betriebsintern:
           OFCString m_name, # Automatisch für
alle Tests
```

```
            PatternTree *m_pPatList,
            TestConditionPtrsAry_t
m_testCondnsArray,
            BarAry_t m_barsArray,
            BinaryOp *m_binaryOp,

            ...
         }, // Klasse MyFancyTest beenden

         ...

         mitlaufende Lücke
         MyFancyTest::addBarParam(CustomTypes::Bar
*arg)
         {
            m_barsArray.push-back(arg),
            Rücksprung
         }
         mitlaufende Lücke
         My-
FancyTest::setBinaryOpParam(CustomTypes::BinaryOp
         *arg)

         {
             m_binaryOp = arg,
             Rücksprung,
         }
         }// Namensraum beenden
```

C8. Parametrierung

[0324] Wie oben ersichtlich ist, bietet ein Preheader für eine Testklasse, eine kundenspezifische Flowable-Klasse oder kundenspezifische Funktionsdefinitionen eine begrenzte Innenschau in Klasse/Funktionen durch einen Spezifizierungsabschnitt der Parametrierung. Der Kompilierer nutzt diesen Abschnitt, um die Parametrierungsschnittstelle für Klasse/Funktion zu generieren (und den Klasse/Funktion-Header selbst zu generieren). Für Testklassen und Flowable-Klassen verwendet er auch diesen Abschnitt, um anschließend die Aufrufe in dem Testplancode zum Initialisieren eines Falles dieser Klasse zu generieren. Es sollten die folgenden Punkte, die die Preheader und eine entsprechende Vereinbarung betreffen, beachtet werden:

1. Jede Definition von Testklassen oder kundenspezifischen Flowable-Klassen wird vorzugsweise in einem Preheader bestimmt. Der Parameterblock im Preheader ist vorzugsweise die einzige Stelle, an der die Parameterliste für eine solche Klasse spezifiziert werden kann. (Folglich müssen als logische Folge die „Standardparameter" für einen Test, wie beispielsweise Spezifikationen von Strukturliste
und Testbedingungen, ebenfalls in dem Parameterblock des Preheaders einbezogen werden, was es ermöglicht, dass alle Parameter, Standardtests und kundenspezifischen Tests einheitlich behandelt werden).
2. Alle als nicht optional definierten Parameter (d. h. mit einer von Null verschiedenen Kardinalität) im Preheader für eine Testklasse oder Flowable-Klasse sollten in der Vereinbarung von Testblock oder Flowab-

le-Block für einen Fall dieser Klasse initialisiert werden.
3. Die zur Initialisierung von Parametern im Test/Flowable-Block verwendeten Objekte sollten vorher definiert worden sein.
4. Austauschanzeigeelemente $Class, $Inc, $ParamAryTypes, $ParamFns, $ParamAttrs und $ParamImpls müssen an den genauen Stellen innerhalb des Benutzercodeabschnitts des Preheaders erscheinen, an denen der Benutzer beabsichtigt, den entsprechenden generierten Code im generierten Klassen-Header-File einzusetzen. Diese sollte exakt einmal erscheinen, weil für jede ein spezifischer Code generiert wird.
5. Der Name einer Parameterspezifizierung im Parameterblock des Preheader (wie beispielsweise in den oben genannten Beispielen PListParam, TestConditionParam oder BarParam) ist der Name des in der Vereinbarung eines Falles dieser Klasse zu verwendenden Parameters.
6. Das Folgende ist die Semantik der in einer Parameterspezifizierung verwendeten Deskriptoren:
a. Kardinalität: diese zeigt die Anzahl von Parametern dieses Typs an, der unterstützt werden wird. Das Folgende sind die möglichen Werte in einer Ausführung:
i 1: Dieser Parameter ist verbindlich und sollte exakt einmal spezifiziert werden. Dieser Parameter wird als ein Zeiger für ein Objekt des Typs des Parameters beibehalten.
ii 0-1: Dieser Parameter ist optional; wenn er festgelegt ist, muss er nur einmal spezifiziert werden. Dieser Parameter wird als ein Zeiger für ein Objekt des Typs des Parameters beibehalten.
iii 1-n: Dieser Parameter ist verbindlich. Außerdem können für diesen mehrere Werte bestimmt werden. Die Werte werden in der Spezifizierungsreihenfolge gespeichert.
iv 0-n: Dieser Parameter ist optional. Für diesen können mehrere Werte bestimmt werden. Die Werte werden in der Spezifizierungsreihenfolge gespeichert.
Zu beachten ist, dass für oben erwähnte () und () alle festgelegten Werte in einem STL Vektor<> gespeichert werden, mit Schablone auf einen Zeiger auf den Typ des Parameters versehen. Der Typ dieses Sektors wird definiert und an dem durch $ParamAryTypes angegebenen Punkt eingesetzt werden. Die Zugangsebene für diese Typendefinitionen ist immer gemeinschaftlich nutzbar.
b. Attribut: der Name der C++ Variablen zur Verwendung als Speicher für Parameterwert(e) dieses Typs. Der Name wird wortwörtlich als ein betriebsinternes Datenelement der C++-Klasse reproduziert und muss den Anforderungen für einen C++ Identifizierer entsprechen. Zu beachten ist, dass der Typ dieses Attributs ist:
i. Ein Zeiger für den Typ des Parameters, wenn nur einzelne Werte erlaubt sind;
ii. Ein STL-Vektor<>, mit Schablone auf einen Zeiger auf den Typ des Parameters versehen, wenn mehrere Werte erlaubt sind (siehe () oben).

[0325] Zu beachten ist, dass die Attribute Bezüge zu Objekten festhalten, die durch den Testplan erzeugt und besetzt sind und diese Objekte nicht besitzen. Die Lebensdauer der Objekte wird immer durch den Testplan selbst verwaltet.

[0326] SetFunction: Der Name der Funktion zur Verwendung, um einen Wert für diesen Parameter zu setzen. Die folgenden Punkte sollten beachtet werden:
i. Der Name wird wortwörtlich reproduziert und muss daher den Anforderungen an die C++ Sprache entsprechen.
ii. Die Zugangsebene der Funktion ist immer gemeinschaftlich nutzbar.
iii. Der Rücksprungstyp ist immer Lücke.
iv. Die Funktion nimmt immer nur ein einziges Argument der Art eines Zeiger-Parameter-Typs an.

[0327] Zu beachten ist, dass ein Wert immer einzeln gesetzt wird, d. h. für Parameter, wie eine Spezifizierung von mehreren Werten erlauben, wobei der generierte Code in dem Testplan diese Funktionen wiederholt aufrufen wird, einmal für jeden spezifizierten Wert, von dem jeder zu einem STL-Vektor (wie oben beschrieben) addiert werden wird.

[0328] Das sich an den Funktionsnamen anschließende optionale Schlüsselwort „[Implementierung]" gibt an, dass eine triviale Implementierung für diese Funktion als ein mitlaufendes Verfahren im Klassen-Header verfügbar gemacht werden wird (der an dem durch $ParamImpls angegebenen Punkt eingesetzt wird). Andererseits ist der Anwender zur Bereitstellung einer Implementierung der Funktion verantwortlich.

[0329] d. Beschreibung: ein Folgeliteral, das eine Tool-Spitze ist, die von einem GUI-Tool verwendet werden wird, um eine Unterstützung während einer Laufzeitmodifizierung dieses Parameters zu bewirken. Die C++ Elementfunktion, die in der kundenspezifischen Klasse für einen mit Xxx benannten Parameter erzeugt wird, wird sein
Sequenz getXxxDescription () const,

[0330] Die Funktion wird die bestimmte Sequenz zurücksetzen.

Beispiel eines Testplans mit Kundenwunschauslegung

[0331] Unten ist das Beispiel eines Testplans gezeigt, der mit einer bestimmten Kundenwunschauslegung verziert ist:

```
# ------------------------------
# File MyCustomizedTestPlan.tpl
# ------------------------------

Version 0.1,


#
```

```
        # Importe wie vorher ...
        # Der folgende Import ist implizit, kann
jedoch
        # explizit vorgesehen werden.
        # Import FunctionalTest.ph,

        # Import für MyFlowables, MyFunctions und
Functions
        Import MyFlowables.ph,
        Import MyFunctions.ph,
        Import Functions.ph,



        # ------------------------------
        # Start des Testplans
        # ------------------------------
        TestPlan Abtastwert,

        # Dieser Block definiert nach Strukturlis-
tenfile
        # qualifizierte Namen und Strukturlisten-
variable,
        # die in Testvereinbarungen verwendet wer-
den.
        # Die file-qualifizierten Namen beziehen
sich auf
        # Strukturlisten in den benannten Files.
Die
        # Variablen beziehen sich auf Sequenzvari-
able, die
        # die Strukturlistennamen zur Laufzeit
halten
        # werden. Benutzerdefinierte Flowable-
Objekte
        # könnten die Werte dieser Variablen durch
eine API
```

```
        # (Anwendungsprogrammierschnittstelle)
setzen.
        PListDefs
        {
           # File-qualifizierte Strukturlistenna-
men
           pl1A.plist:pat1Alist,
           pl2A.plist:pat2AList,

           # Strukturlistenvariable
           plistXxx,
           plistXxx,
           plistZzz
        }
           # SocketDef, UserVars Vereinbarung wie
vorher ...

           # Vereinbarungen der Testbedingungen
TC1Min,
           # TC1Typ, TC1Max, TC2min, TC2Typ,
TC2Max wie
           # vorher ...




           #
           # Vereinbare einen Funktionstest.
"Funktionstest"
           # bezieht sich auf eine C++ Testklasse,
die den
           # Test abarbeitet und eine 0,1 oder 2
als ein
           # Ergebnis rücksetzt. Die Testbedin-
gungsgruppe
```

```
          # TCG1 wird mit dem Auswähler „min"
durch Bezug
          # auf die Testbedingung TC1Min ausge-
wählt.
          #
          # Zu beachten ist, dass der Kompilierer
diese
          # wegen des importierten Files Functio-
nalTest.ph
          # kompilieren kann.
          #
          Test Funktionstest MyFunctionalTest1Min
          {
            PListParam = pat1AList,
            TestConditionParam = TC1Min,

          }

          #
          # zusätzliche Funktionstestvereinbarun-
gen für die
          # folgenden wie vorher
          # MyFunctionalTest1Typ
          # MyFunctionalTest1Max
          # MyFunctionalTest2Min
          # MyFunctionalTest2Typ
          # MyFunctionalTest2Max
          #

# Hier ist eine Vereinbarung von MyFlowable. Sie
nutzt eine # Strukturlistenvariable plistXxx, die
durch die Flowable
# vor einer Verwendung hier initialisiert wird.
#
# Kompilierer kann diese wegen des importierten
Files MyFlowables.ph kompilieren:
```

```
Flowable MyFlowable MyFlowable1

{
   Int1 = 10,
   Int2 = 20,
   Sequenzelement = „Hello World“,
   PlistParam = plistXxx,
}

# Zähler für PassCount und FailCount wie vorher
...

# Abläufe wie vorher. Abläufe FlowTest1 und Flow-
Test2 sind # unverändert zu dem vorherigen Bei-
spiel.
Ablauf AblaufTest1
{
  # ...
}

Ablauf Flowtest2
{
  # ...
}
#
# Jetzt kann FlowMain, ein Hauptablauf, darge-
stellt werden. # Er implementiert einen Endlich-
zustandsautomaten, der
# FlowTest1 und FlowTest2 wie nachstehend auf-
ruft:
# ---------------------------
#             Ergebnis 0    Ergebnis 1
# ---------------------------
# FlowMain_1  FlowMain_2    Rücksprung 1
# FlowMain_2  FlowMain_3    Rücksprung 1
# FlowMain_3  FlowMain_4    Rücksprung 1
```

```
# FlowMain_4  FlowMain_5    Rücksprung 1
# FlowMain_5  Rücksprung 0  Rücksprung 1
#
# Dort, wo die IFlowables durch jedes Flowelement
abgearbeitet werden, sind:
# --------------------------------
#       Ablaufelement    IFlowable, die abgear-
beitet ist
# --------------------------------
#       FlowMain_1       MyFlowable1
#       FlowMain_2       DatalogStartFlow
#       FlowMain_3       FlowTest1
#       FlowMain_4       FlowTest2
#       FlowMain_5       DatalogStopFlow
#
Ablauf FlowMain
{
   #
   # Der erste vereinbarte Ablauf ist der auszu-
führende
   # Anfangsablauf. Er geht zu Flow-
Main_InitializationFlow
   # bei Erfolg und setzt 1 bei Ausfall zurück.
   #
   Ablaufelement FlowMain_1 MyFlowable1
   {
        Ergebnis 0
        {
           Merkmal PassFail = „Pass“,
           Inkrementierungszähler PassCount,
           # Ein Benutzerfunktionsaufruf
           MyFunctions.Print(„Passed MyFlo-
wable1“,
                                Functi-
ons.GetDUTID()),
           GeheZu FlowMain_2,
```

```
        }

        Ergebnis 1
        {
           Merkmal PassFail = „Fail“,
           Inkrementierungszähler FailCount,
           # Ein Benutzerfunktionsaufruf
           # MyFunctions.Print(„Failed MyFlo-
wable1“,
                                Functi-
ons.GetDUTID()),
           SetBin SoftBins.“3GHzLeakage“,
           Rücksprung 1,
         }
       }

       #
       # Geht bei Erfolg zu FlowMain_3 und setzt
1 bei
       # Ausfall zurück.
       #
       Ablaufelement FlowMain_2 DatalogStartFlow
       {
          Ergebnis 0
          {
            Merkmal PassFail = „Pass“,
            Inkrementierungszähler PassCount
            # Ein Benutzerfunktionsaufruf
            MyFuncti-
ons.Print(„PassedDatalogStartFlow“,
                                     Functi-
ons.GetDUTID()),
            GeheZu FlowMain_3,
          }

          Ergebnis 1
```

```
        {
            Merkmal PassFail = „Fail“,
            Inkrementierungszähler FailCount
            MyFunctions.Print(“Failed Datalog-
StartFlow“,
                                        Func-
tions.GetDUTID()),
            Rücksprung 1,
        }
    }

    # Dieses Ablaufelement ruft den vorher
definierten
    # FlowTest1 auf.
    Ablaufelement FlowMain_3 FlowTest1

    {
        Ergebnis 0
        {
         Merkmal PassFail = „Pass“,
         Inkrementierungszähler PassCount,
         # Ein Benutzerfunktionsaufruf
         MyFunctions.Print(„Passed FlowTest1“,
                           Functi-
ons.GetDUTID(),
         GeheZu FlowMain_4,
        }

        Ergebnis 1
        {
          Merkmal PassFail = „Fail“,
          Inkrementierungszähler FailCount,
          # Ein Benutzerfunktionsaufruf
          MyFunctions.Print(„FailedFlowTest1“,
                           Functi-
ons.GetDUTID()),
```

```
            SetBin SoftBins."3GHzCacheFail",
            Rücksprung 1,
          }
        }

        # Dieses Ablaufelement ruft den vorher
definierten
        # AblaufTest2 auf
        Ablaufelement FlowMain_4 FlowTest2
        {
          Ergebnis 0
          {
            Merkmal PassFail = „Pass",
            Inkrementierungszähler PassCount,
            # Ein Benutzerfunktionsaufruf
            MyFunctions.Print(„Passed Flow-
Test2"),
                                Func-
tions.GetDUTID()),
            GeheZu FlowMain_5,
        }

        Ergebnis 1
        {
          #FlowTest1 bestanden, FlowTest2 jedoch
nicht
                                          be-
standen
          Merkmal PassFail = „Fail",
          Inkrementierungszähler FailCount,
          # Ein Benutzerfunktionsaufruf
          MyFunctions.Print(„Failed FlowTest2",
                          Functi-
ons.GetDUTID(),
          SetBin SoftBins."3GHzSBFTFail",
          Rücksprung 1,
```

```
        }
      }

      Ablaufelement FlowMain_5 DatalogStopFlow
      {
        Ergebnis 0
        {
          # Alle bestanden !
          Merkmal PassFail = „Pass“,
          Inkrementierungszähler PassCount,
          # Ein Benutzerfunktionsaufruf
          MyFunctions.Print(„FPassed all!“,
                                Functi-
ons.GetDUTID(),
          SetBin SoftBins.“3GHzAllPass“,
          Rücksprung 0,
        }

        Ergebnis 1
        {
           # FlowTest1 und FlowTest2 bestanden,
           # jedoch DatalogStopFlow nicht be-
standen
           Merkmal PassFail = „Fail“,
           Inkrementierungszähler FailCount,
           # Ein Benutzerfunktionsaufruf
           MyFunctions.Print(„Failed Data-
logStopFlow“,
                                Functi-
ons.GetDUTID(),
           Rücksprung 1,
         }
       }
     }
```

[0332] Über den oben genannten Code müssen die folgenden Punkte besonders erwähnt werden:

1. Der Abschnitt PListDefs weist hier einige PList-Namen und auch einige PList-Variable auf. Die PList-Namen sind Namen, die in Tests direkt verwendet werden können. Die PList-Variablen sind Variable, die in Tests genutzt werden können und deren Wert zur Laufzeit an aktuelle PLists durch Code in einer kundenspezifischen Flowable gebunden ist.
2. Der Abschnitt PListDefs ist optional. Falls er nicht vorhanden ist, wird sein Inhalt durch einen Kompilierer aus den verschiedenen Testvereinbarungen abgeleitet. Wenn er vorhanden ist, muss er alle der verwendeten PList-Parameter von Tests vereinbaren, obwohl er mehr vereinbaren kann.

3. Eine Laufzeit-Anwendungsprogrammierschnittstelle (API) wird verfügbar sein, um den PList-Variablen Werte zuzuordnen. Die Testplanklasse wird eine Funktion besitzen:

```
     Status SetPListVariable(const
Tester::String& varName,
              const Tester::String& fileQualified-
PListName),

     Die Flowable wird in der Lage sein, die oben
erwähnte
     Funktion zu nutzen, um eine PList-Variable
an eine
     spezielle PList zu binden.
```

4. Benutzerfunktionen und Funktionen können in Ablaufelementen direkt vor einem Übergang, der entweder eine Steuerübertragung auf ein anderes Ablaufelement oder ein Rücksprung ist, aufgerufen werden.

C++ für Benutzerfunktionsaufrufe

[0333] Bis auf das Zitieren kundenspezifischer Funktionsaufrufe in Abläufen, wurde für die früher dargestellten verschiedenen Verfahren der Kundenwunschauslegung ein C++-Code gezeigt, der durch den Kompilierer erzeugt werden würde. Aufrufe der Benutzerfunktion in einem Ablaufelement werden vorzugsweise durch ein Element IUserCalls von jedem Ablauf abgearbeitet. Jeder Ablauf hat vorzugsweise ein Element der Schnittstelle IUserCalls, das eine einzelne virtuelle Elementfunktion, wie nachstehend gezeigt, exportiert:

```
     Klasse IUsersCalls
     {
     gemeinschaftlich nutzbar:
       virtuelle Lücke exec(const String& flowI-
temName,
                            unsigned int result)
= 0
     },
```

[0334] Wenn man auf einen Ablauf mit Benutzerfunktionsaufrufen trifft, lässt sich der Ablauf mit einem Fall einer Klasse besetzen, die die oben erwähnte Schnittstelle implementiert. Beispielsweise wird in dem Beispiel FlowMain der Ablauf mit einem Fall der folgenden Klasse besetzt werden:

```
Klasse FlowMain_UserCalls: public IUserCalls
{
gemeinschaftlich nutzbar:
    virtuelle Lücke exec(const String& flo-
wItemName,
                         unsigned int re-
sult)
    {
       if(flowItemName == „FlowMain_1“)
       {
          //...
       } else if(flowItemName == “Flow-
Main_2”)
       {
          //...
       } else if(flowItemName == “Flow-
Main_3”)
       {
         schalten (Ergebnis)
         {
         Fall 0:
             MyFunctions::Print(“Passed
FlowTest1”,
                                Func-
tions::GetDUTID(),
             Rücksprung,
         Fall 1:
             MyFunctions::Print(“Failed
FlowTest1”,
                                Func-
tions::GetDUTID(),
             Rücksprung,
         Vorgabewert:
```

```
                Rücksprung,
              }
            }
            else if (flowItemName == "Flow-
Main_4")
            {
              //...
            }
            else if (flowItemName == "Flow-
Main_5")
            {
              //...
            }
          }
        },
```

[0335] Die Rechenoperation FlowItem::execute() kennt den Namen des Ablaufelements. Bevor sie mit dem Zeiger zu dem nächsten Ablauf zurück springt, wird sie IUserCalls::exec() für den umgebenden Ablauf aufrufen, indem ihr eigener Name des Ablaufelements und der Wert des aktuellen Ergebnisses weitergegeben wird. Dies wird bewirken, dass der oben erwähnte Code ausgeführt wird, indem die benötigten benutzerdefinierten Funktionen aufgerufen werden.

C9. Testprogramm-Kompilierung

[0336] Wie oben erläutert, bestimmt der Testplan-Beschreibungsfile die in einem Testplan verwendeten Objekte und ihre Beziehungen zueinander. In einer Ausführung wird dieser File in den C++-Code übersetzt, der auf dem Site-Controller in Form einer Implementierung einer Standardschnittstelle ITestPlan ausgeführt werden wird. Dieser Code kann in eine Datei für Betriebssystemroutinen von Windows (DLL) gepackt werden, die in den Site-Controller geladen wird. Das Testprogramm DLL wird generiert, um normale bekannte Eingabepunkte zu haben, die die Site-Controller-Software nutzen kann, um das Testplanobjekt, das sie enthält, zu generieren und zurück zu setzen.

Konstruktionen aus einer Testplanbeschreibung

[0337] Der Umsetzungsprozess von einer Testplanbeschreibung zu einer Implementierung von ITestPlan wird durch den Testprogrammkompilierer **400** ausgeführt. Dieser Prozess tritt in zwei Phasen auf: Umsetzung und Kompilierung.

[0338] In der Umsetzungsphase **402** verarbeitet der Kompilierer **400** sowohl einen Testplanfile (und die verschiedenen anderen Files, die er importiert) als auch die Preheader für alle Testtypen, die in dem Testplan verwendet werden. In dieser Phase erzeugt er den C++-Code für das Testplanobjekt und die C++ Header für die gefundenen Testtypen zusammen mit allen anderen Unterstützungsfiles wie beispielsweise MSVC++ (Microsoft Visual C++) Arbeitsbereich und Projektfiles, DLL „Textbausteincode", usw.. Der Kompilierer **400** setzt in den generierten Code File- und Zeilendirektiven ein, um zu gewährleisten, dass Kompilierzeit-Fehlermeldungen auf die kompetente Stelle im Beschreibungsfile zurückverweisen, anstatt in den generierten Code zu zeigen.

[0339] In der Kompilierungsphase, die auftritt, nachdem der Kompilierer die notwendigen Files erzeugt hat, wird ein Standardkompilierer **404** wie beispielsweise ein MSVC++ Kompilierer aufgerufen, um die Files zu kombinieren und sie in einer DLL zu verknüpfen.

[0340] Der Kompilierer nimmt als Eingabe einen gültigen Testplanfile (und alle darauf bezogenen Files) und erzeugt, wenn nötig, einen Testplanfile und alle anderen Files, die in dem Testplanfile durch „Importdirektiven"

dargestellt sind. Außerdem erzeugt er eine MSVC++ „Lösung", um die Testplan-DLL zu konstruieren. Wenn zum Beispiel die Stammdatei (MyTestPlan.tpl) Timing1.tim enthielt, um Informationen der zeitlichen Steuerung einzubeziehen, dann würde der Kompilierer (unter anderem) die folgenden Files erzeugen:
MyTestPlan.h
MyTestPlan.cpp
Timing1.cpp
MyTestPlan.sln(MSVC++ „Solution" file)
MyTestPlan.vcproj (MSVC++ "Project" file)

[0341] Nachdem alle Files erzeugt (oder aktualisiert) sind, ruft der Kompilierer die MSVC++ Anwendung auf, die bestimmt, dass die „Lösung" erzeugt ist, und konstruiert die Datei für Betriebssystemroutinen (DLL). Irgendwelche Fehler und/oder Warnungen würden dem Anwender gezeigt werden.

[0342] Wenn der Anwender nach Konstruktion des Testplans einen Wechsel zu Timing1.tim vorgenommen hat, würde er dann den Kompilierer aufrufen, indem MyTestPlan.tpl an ihn weiter gegeben wird. Der Kompilierer würde (durch Zeitmarkierungsinformationen) erkennen, dass die Testplan-Stammdatei unverändert ist, so dass MyTestPlan.h/.cpp nicht erneut erzeugt werden würde. Jedoch würde er während der Verarbeitung der Testplan-Stammdatei sehen, dass sich die Datei Timing.tim verändert hat. Deshalb würde er die Datei Timing1.cpp erneut erzeugen und die MSVC++ Anwendung aufrufen, um die DLL zu rekonstruieren. Diese vermeidet erneutes Kompilieren von MyTestPlan.cpp, kompiliert nur Timing1.cpp und verknüpft die DLL erneut. Dieser Lösungsweg wird besonders brauchbar sein bei Verringerung der Zeiten für erneutes Kompilieren und erneutes Verknüpfen für große Testpläne, die eine bedeutende Menge an Zeit zum Kompilieren benötigen.

D. Abarbeiten des Testprogramms

[0343] Die Software des Site-Controllers lädt die Testprogramm-DLL in ihren Prozessraum und ruft innerhalb der DLL eine „Betriebsfunktion" auf, um einen Fall des Testplanobjekts zu erzeugen. Sobald das Testplanobjekt erzeugt worden ist, kann dann die Site-Controller-Software den Testplan ausführen oder mit ihm in einer anderen notwendigen Art und Weise in Dialogverkehr treten.

Nicht wechselwirkende Formen

[0344] Für die meisten C++ Softwareentwickler, die eine Anwendung (oder eine DLL oder Bibliothek) in der Windows-Umgebung konstruieren, bedeutet Einführen einer Entwicklungsumgebung (MS Visual C++, Borland C++ oder ähnliches) das Editieren eines Codes und (oft) Drücken eines Knopfes, um das Produkt zu konstruieren.

[0345] Die Testumgebung nach einer Ausführung der Erfindung wird eine ähnliche Menge von Aktivitäten aufweisen. Die Testplanentwickler werden einen Code editieren und ihre Testpläne konstruieren müssen. Jedoch werden Tester von dem Testplanentwickler nicht verlangen, eine C++ Entwicklungsumgebung einzuführen, um die sich ergebende Testplan-DLL herzustellen.

[0346] Um dies auszuführen, nutzt die vorliegende Erfindung das Konzept einer nicht wechselwirkenden Form. Eine nicht wechselwirkende Form ist als eine Bauart definiert, die in einem nicht-interaktiven Modus MS Visual C++ verwendet. Es ist zu beachten, dass es diese dennoch erlaubt, andere Tools interaktiv zu nutzen, um eine solche Form zu verwalten. Die einzige Implikation ist, dass Visual C++ nicht-interaktiv genutzt wird.

Vorausgesetzte Umgebung

[0347] Zur Anwenderkonfiguration werden bestimmte Annahmen getroffen. Die Annahmen sind:

1. Der Testplan-Entwickler wird seinen Testplan nach den oben erwähnten Verfahren und Regeln entwickeln.
2. Der Testplan-Entwickler hat vielleicht keine Kenntnis von C++ auf Sachverständigenniveau.
3. Der Testplan-Entwickler wird zu Kommandozeilen-Tools oder GUI-Tools Zugriff haben, um Datei(n) in eine Testplan-DLL zu konvertieren.

Konstruieren von Anwendungen ohne Berührungsfelder

[0348] Nicht-interaktives Arbeiten mit Microsoft® Visual Studio erfordert einen oder zwei Lösungswege. Der erste (und einfachste) ist, die Kommandozeilen-Schnittstelle zu verwenden. Der zweite (und flexiblere) ist, die

Automatisierungsschnittstelle zu nutzen. Dieser Abschnitt beschreibt beide Lösungswege.

Schaffung des Projektes

[0349] Um Visual Studio nicht-interaktiv zu nutzen, sollte man mit einer Arbeitslösung beginnen, die ein oder mehrere gültige Projekte enthält. Leider ist dies die einzige Aufgabe, die weder von einer Lösung mit Kommandozeile oder einer Automatisierungslösung ausgeführt werden kann. Kein Verfahren stellt einen Mechanismus zur Erzeugung eines Projektes bereitet. Jedoch können Projekte und Lösungen für Visual Studio von einer Schablone erzeugt werden. Deshalb können wir, vorausgesetzt einen Projektnamen und eine Schablone, von der aus gestartet werden kann, eine Lösung/ein Projekt für Visual Studio erzeugen.

Besetzung des Projektes

[0350] Das Hinzufügen von neuen Dateien zu dem erzeugten Projekt nutzt das Automatisierungsmodell von Visual Studio, weil die Kommandozeile dies nicht unterstützt. Wir erzeugen zwei Makros von Visual Studio, um zu einem Projekt neue und vorhandene Dateien hinzuzufügen. Ein ähnlicher Code könnte durch einen externen Script unter Verwendung einer ActiveScript-Software (wie VBScript, JScript, ActivePerl, ActivePython, usw.) genutzt werden, um die gleichen Aufgaben auszuführen. Deshalb könnten unsere Tools der Codeerzeugung neue Dateien schaffen und sie durch das Automatisierungsmodell zu dem vorhandenen Visual Studio Projekt hinzufügen. Nachdem die Dateien erzeugt sind, können sie wenn nötig durch die Tools aktualisiert werden.

Konstruieren des Projektes

[0351] Sobald wir Lösung und Projekt am Ort haben, gibt es mehrere Optionen Visual Studio nicht-interaktiv zu nutzen, um den TestPlan zu konstruieren. Die einfachste Option ist, ihn von der Kommandozeile aufzurufen. Eine solche Kommandozeile würde aussehen wie:
devenv solutionFile/build solutionCfg
wobei solutionFile eine Visual Studio-Lösungsdatei und solutionCfg eine spezifische Konfiguration ist, die auf die Projekte innerhalb der Lösung anwendbar sind. Eine andere Lösung ist es, das Objektmodell von Visual Studio zur Automatisierung zu nutzen. Dies lässt ein feineres Gefüge der Steuerung über den Konstruktions- und Konfigurationsprozess zu. Wie es oben erwähnt ist, enthält sie eine Protokollierung eines Perl-Script zum Konstruieren eines Projektes aus der Kommandozeile. Dieses Programm liest eine Konfigurationsdatei, die Projekte und Konfigurationen zum Konstruieren (sowie andere Informationen über die Projekte) festlegt und konstruiert sie alle unter Verwendung des Automatisierungsmodells. Man sieht sich die Verwendungen des Objektes $msdev in diesem Script für Beispiele an, wie Automatisierungsobjekte in einem Script zu nutzen sind.

Testhilfeprogramm-Unterstützung

[0352] Damit Entwickler von Testklassen ihre Arbeit verifizieren und Fehler beseitigen, müssen sie auf ein Testhilfeprogramm Zugriff haben, das es ihnen ermöglicht, den Site-Controller zu öffnen und durch ihren Code zu schreiten. Weil der von dem Kompilierer erzeugte Code C++ ist, der durch MSVC++ kompiliert wird, verwenden wir das MSVC++ Testhilfeprogramm um Testklassen-Implementierungen zu korrigieren. Es ist zu beachten, dass dieses Merkmal nur für Entwickler von Testklassen oder andere gemeint ist, die direkt in C++ arbeiten. Testingenieuren, die Fehler zu beseitigen oder durch die Rechenoperation eines Testprogramms zu schreiten wünschen, ohne direkt Bezug auf den erzeugten C++ Code zu nehmen, werden andere Mechanismen zur Verfügung gestellt.

Umgebung der Systemsoftware

[0353] Dieser Abschnitt beschreibt die allgemeine Softwareumgebung für den Tester: die Stellen für die Dateien, die durch Benutzertestpläne, benötigt werden, Mechanismen zum Bestimmen von abwechselnden Stellen für solchen Dateien und die Verfahren zum Festlegen der Stellen der Testpläne und Modulsteuersoftware.

Von Testplänen benötigte Umgebung

[0354] Systemstandardstellen sowie die Laufzeitkonfiguration der Suchpfade für von einem Testplan benötigten

1. Strukturlisten,

2. Strukturen,
3. Zeitsteuerungsdaten, und
4. Testklassen-DLL

können durch „Umgebungsvariable" wie durch Umgebungskonfigurationsdateien festgelegt, konfiguriert werden. Diese sind Textdateien mit einer einfachen Syntax wie:
Tester PATOBJ PATH = „patterns\data;D:\projects\SC23\patterns\data"

[0355] Der Vorteil, solche in Textdateien definierte "Umgebungen" anstelle von systemeigenen Umgebungsvariablen zu haben, die durch das Rechnersystem unterstützt werden, ist, dass die Implementierung dann durch die gemeinsamen Einschränkungen, die rechnersystemunterstützte Umgebungsvariable besitzen wie beispielsweise maximale Sequenzlängen, usw., nicht beschränkt ist. Die folgenden „Umgebungsvariablen (Setup) werden für die oben angeführten Entitäten verwendet werden:
Strukturlisten: Tester_PATLIST_PATH.
Strukturobjektdateien: Tester_PATOBJ_PATH.
StukturQuellenfiles: Tester_PATSRC_PATH (dies ist optional, siehe bitte).
Zeitsteuerungsdatenfiles: Tester_TIMING_PATH.
Testklassen-DLL: Tester TEST_CLASS_LIBPATH.

[0356] Um spezielle Fälle zu unterstützen, während nützliches vorgegebenes Verhalten beibehalten wird, stellen wird drei Konfigurationsebenen bereit. Diese werden in ansteigender Rangordnung beschrieben:
Zuerst wird ein Systemumgebungs-Einstellfile, $Tester_INSTALLATION_ROOT\cfg\setups\Setup.env, die vorgegebenen Werte von „Umgebungsvariablen" festlegen. Wenn kein anderer Konfigurationsmechanismus verfügbar ist, wird dieser File benötigt werden. Im Allgemeinen wird er für alle auf dem System laufenden Testpläne verfügbar sein. Dieser File wird während einer Installation durch das Installations- und Konfigurationsverwaltungssystem (ICM) mit Eingabe vom Installierer erzeugt, um die vorgegebenen Werte für die drei oben erwähnten Variablen zuzuordnen. (Zu beachten ist, dass dieser File neben den Systemvorgaben für die oben erwähnten drei Variablen auch die Systemvorgaben für bestimmte andere Variable der Testerumgebung enthalten wird, wie es in dem folgenden Unterabschnitt beschrieben wird).

[0357] Zweitens kann durch den Anwender ein Umgebungs-Einstellfile als Laufzeitargument für den Testplan festgelegt werden. Die Variablen in dieser Laufzeitkonfiguration werden Vorrang gegenüber Vorgabedefinitionen haben.

[0358] Schließlich kann ein Testplan einen speziellen Block zum Festlegen der in seiner Ausführung zu verwendenden Umgebungsvariablen nutzen. Im Testplan definierte Variable werden Vorrang gegenüber denen in dem Vorgabesystemfile oder dem anwenderdefinierten File haben.

[0359] Im Allgemeinen sollten alle notwendigen Variablen durch einen der oben beschriebenen Mechanismen definiert werden. Wenn eine Variable nicht definiert ist, wird ein Laufzeitfehler auftreten.

Andere Umgebungseinstellungen

[0360] Neben den „Umgebungsvariablen", die von den Benutzertestplänen benötigt werden, werden durch die Testumgebung die folgenden zwei „Umgebungsvariablen" benötigt:

1. Tester TEST_PLAN_LIBPATH: Diese legt den Suchpfad fest, den die Systemsteuereinheit nutzen wird, um eine DLL des Anwendertestplans zu finden, die geladen werden soll. Zu beachten ist, dass der gleiche Suchpfad auch genutzt wird zum Finden von Anwender-Pinbeschreibungs- und Socket-Files. Der Vorgabewert für diese Variable, der während einer Installationszeit für das Installationskonfigurations-Verwaltungssystem (ICM) festgelegt ist, wird durch das ICM in der Datei $Tester_INSTALLATION_ROOT\cfg\setups\Setup.env gespeichert.
2. Tester_MODULE_LIBPATH: Diese legt den Suchpfad fest, den das System verwenden wird, um die DLL von vom Hersteller gelieferte Hardwaremodul-Steuersoftware zu laden. Diese aus der Konfigurationsverwaltungs-Datenbank (CMD) gezogenen Informationen werden in der Datei $Tester_INSTALLATION_ROOT\cfg\setups\Setup.env durch das ICM auch gespeichert.

[0361] Während ein Anwender den in der Datei Setup.env gegebenen Wert für die Variable Tester_TEST_PLAN_LIBPATH übersteuern kann, ist zu beachten, dass der in der Datei Setup.env gegebene Wert für den Tester_MODULE_LIBPATH durch den Anwender nicht geändert werden sollte, es sei denn, dass der Anwender den Suchpfad für die DLL von vom Hersteller gelieferte Hardwaremodul-Steuersoftware aus-

drücklich ändern will.

Spezifizierungssemantik für Suchpfade

[0362] Hinsichtlich der „Umgebungsvariablen", die Suchpfade festlegen, sollten die folgenden Punkte beachtet werden:

1. Jede sollte eine durch Semikolon („;") getrennte Liste von Verzeichnisnamen sein, die das System suchen wird, um eine zugeordnete Datei eines speziellen Typs zu finden.
2. Nach dem Suchen des Wertes einer solchen „Umgebungsvariablen" durch das Ausgangssystem werden beliebige vom Anwender vorgenommene Änderungen an ihrem Wert (zum Beispiel durch Editieren einer Umgebungskonfigurationsdatei) nur durch das System registriert werden, wenn der Anwender das System ausdrücklich über die Notwendigkeit, dies zu tun, „informiert".
3. Relative Pfadnamen in den Suchpfaden werden so interpretiert, als seien sie von einer speziellen Einstellung einer zugeordneten Umgebungsvariablen (die die Funktionalität, eine Wurzel zu definieren, bewirkt) abhängig, weil Pfade im Verhältnis zu dem „gegenwärtig arbeitenden Verzeichnis" (CWD) zu zweideutigen Ergebnissen führen könnten, da der Begriff eines CWD in einer verteilten Umgebung, wie dasjenige, in dem der Tester arbeitet, nicht das sein könnte, was der Anwender intuitiv davon erwartet. Diese zugeordnete Umgebungsvariable, die die Wurzel kennzeichnet, dass von allen relativen Pfadnamen in den Suchpfaden angenommen wird, relativ dazu zu sein, ist die Variable „Tester_INSTALLATION_ROOT", die die Stelle des Verzeichnisses der obersten Ebene (d. h. "Wurzel") der Testerinstallation auf einem Anwendersystem angibt.
4. Die Verzeichniseingaben können nicht die Zeichen in der Menge [v:*?"<>|;] enthalten; es ist zu beachten, dass mit Ausnahme des Semikolons (";") alle anderen Zeichen in dieser Menge in Dateinamen von Windows unerlaubt sind. Das Semikolon (";") sollte in Suchpfadeingaben nicht verwendet werden, weil es genutzt wird, um Eingaben im Suchpfad abzugrenzen. Zu beachten ist, dass Pfadnamen eingebettete Lücken haben können, jedoch alle unmittelbar vor und nach einem Pfadnamen auftretenden Lücken (d. h. vor dem ersten und nach dem letzten Nicht-Lücken-Zeichen im Pfadnamen) nicht als Teil des Pfadnamens berücksichtigt werden und ignoriert werden.
5. Die Suchpfadverzeichnisse werden in der Reihenfolge aufgesucht, wie man auf sie in der Definition trifft. Das erste Auftreten einer Datei wird die gewählte sein.

E. Testmuster

[0363] Die effiziente Verwaltung, Handhabung und das Laden einer sehr großen Menge von Testmusterdateien ist ein wichtiger architektonischer Aspekt des Rahmens einer Ausführung der Erfindung. Die Idee von hierarchischen Strukturlisten wird als effektives Tool bei der Bereitstellung einer fügsamen begrifflichen Erfassung und Erleichterung der Verwendung des Systems für den Endbenutzer betrachtet.

[0364] Der Anreiz für ein Prüfobjekt (DUT) wird dem Testsystem durch Testvektoren verfügbar gemacht. Vektoren können allgemein als sequenzielle (oder lineare), von Abtastung oder Algorithmischem Strukturgenerator (APG) abgeleitete zugeordnet werden. In dem System nach einer Ausführung der Erfindung sind Testvektoren unter dem Aspekt von Strukturen organisiert, die zum Testzeitpunkt an dem DUT angewandt werden. Eine Struktur wird durch ein Strukturobjekt im Benutzertestprogramm dargestellt. In dem System sind Strukturen in Strukturlisten organisiert, die durch Strukturlistenobjekte programmatisch dargestellt werden. Ein Strukturlistenobjekt stellt eine geordnete Liste von Strukturen oder andere Strukturlisten dar. Die Ordnung ist implizit in der Vereinbarungsreihenfolge der Listenkomponenten. Zu beachten ist, dass wenn nur eine einzelne Struktur benötigt wird, es erforderlich ist, in einer Liste durch sich selbst eingeschlossen zu werden.

[0365] Ein Strukturlistenobjekt im Testprogramm des Benutzers wird mit einer Strukturlistendatei auf Platte verknüpft, die die aktuelle Definition der Strukturliste enthält. Die Inhalte einer Strukturliste werden somit dynamisch durch die Inhalte der verknüpften Plattendatei bestimmt (mehr darüber wird später gesagt).

[0366] Die Definition einer Strukturliste stellt einen expliziten Namen für die Strukturliste bereit und identifiziert eine geordnete Liste von Strukturen und/oder andere Strukturlisten durch Verknüpfungen von Dateinamen. Sie sieht auch die Spezifizierung von Ausführungsoptionen vor, die ausführlich beschrieben werden, nachdem die Strukturobjekte beschrieben worden sind, weil die Optionen sowohl auf Strukturlisten als auch auf Strukturen angewandt werden können. Die Strukturliste sollte die folgenden Regeln einhalten:

file-contents:
version-info global-pattern-list-definitions

version-info:
Version *version-identifier,*

global-pattern-list-definitions:
global-pattern-list-definition
global-pattern-list-definitions global-pattern-list-definition
global-pattern-list-definition:
global-pattern-list-declaration {list-block}

global-pattern-list-declaration:
GlobalPList *pattern-list-name options*$_{opt}$

list-block:

```
        list-entry
        list-block list-entry
    list-entry:
      pattern-entry,
      pattern-list-entry,
      global-pattern-list-definition,
      local-pattern-list-definition,

    pattern-entry:
      Pat pattern-name options_opt
    pattern-list-entry:
      PList pattern-list-reference option-
s_opt
    pattern-list-reference:
      pattern-list-qualified-name
      file-name ,:' pattern-list-qualified-
name
    pattern-list-qualified-name:
      pattern-list-name
      pattern-list-qualified-
name'.'pattern-list-name
    local-pattern-list-definition:
      local-pattern-list-declaration {list-
block}
    local-pattern-list-declaration:
      LocalPList pattern-list-name option-
s_opt
    options:
      option
      options option
    option:
      [option-definition]
    option-definition:
      option-name option-parameters_opt

    option-parameters:
option-parameter
option-parameters','option-parameter
```

[0367] Das Folgende sind die Beschreibungen von oben verwendeten undefinierten Nicht-Eingängen:

1. version-identifier: Eine Sequenz von einem oder mehreren Zeichen aus der Menge [0-9], wobei das erste

Zeichen eine Ziffer sein muss.
2. name: Eine Sequenz von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], wobei das erste Zeichen aus der Menge [a-zA-Z] sein muss.
3. Pattern-list-name: Eine Sequenz von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], wobei das erste Zeichen aus der Menge [a-zA-Z_] sein muss.
4. file-name: Ein gültiger Windows-Dateiname (muss in doppelten Anführungszeichen eingeschlossen sein, wenn in dem Dateinamen irgendwelche Lücken enthalten sind). Zu beachten ist, dass dies ein einfacher Dateiname sein soll, d. h. er sollte keine Verzeichniskomponente besitzen. Ein Struktur-Listen-Bezug kann entweder ein interner Bezug auf einer Strukturliste in der gleichen Datei oder ein externer Bezug auf eine in einer anderen Datei sein. Externe Bezüge müssen durch einen Dateinamen qualifiziert sein.
5. Option-name: Eine Sequenz von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], wobei das erste Zeichen aus der Menge [a-zA-Z_] sein muss.
6. Option-parameter: Eine Sequenz von einem oder mehreren Zeichen aus der Menge[a-zA-Z_0-9].

[0368] Strukturlistendateien unterstützen Kommentare, die dazu bestimmt sind, durch einen Analysealgorithmus der Strukturlistendateien ignoriert zu werden. Kommentare starten mit dem ‚#' Zeichen und erstrecken sich bis zum Ende der Zeile.

E1. Regeln für Strukturliste

[0369] Die statischen Regeln oder Kompilierzeitregeln für Strukturlisten bestimmen die Vereinbarung und Auflösung von Namen. Namen in der Strukturlisten-Sprache werden durch Global-Struktur-Listen-Definitionen und Lokal-Struktur-Listen-Definitionen vereinbart. Auf sie wird durch Struktur-Listen-Bezüge verwiesen. Nachstehend sind einige Regeln, die diese Vereinbarungen und Bezüge bestimmen.

1. Eine Global-Struktur-Listen-Definition oder eine Lokal-Struktur-Listen-Definition vereinbart den Namen einer Strukturliste. Ein Struktur-Listen-Bezug verweist auf den Namen einer vereinbarten Strukturliste. Die Namen von globalen Strukturlisten sind umfassend bekannt. Die Namen von lokalen Strukturlisten sind nur in dem Listenblock bekannt, in dem sie vereinbart werden. Auf sie kann sich ohne Qualifizierung direkt in diesem Listenblock bezogen werden. In einer tiefer verschachtelten Vereinbarung wird durch einen qualifizierten Namen auf eine lokale Strukturliste Bezug genommen werden müssen.
2. Namen von lokalen Strukturlisten sind innerhalb des Umfangs einer umfassenden Strukturliste und Namen von globalen Strukturlisten innerhalb des Umfangs des Systems bekannt.
Zum Beispiel:

```
GlobalPList G1
{
   LocalPList L1
   {
      LocalPList L2
      {
         ...
      }

      GlobalPList G2
      {
         ...
      }

      Plist L2     #OK. Name L2 ist
in diesem
                           Umfang
bekannt
      Plist G2     #OK. Name G2 ist
global
   }

   PListL2,        #Fehler. Name L2
ist hier
```

```
                                        nicht
 bekannt.
                PList L1.L2,    #OK. Name L1 ist
 hier
                                     bekannt. L2
 ist
                                     bekannt
 durch
                                #    Qualifizie-
 rung.
                PList G1.L1.L2, #OK. Qualifi-
 zierung durch
                                  G1 wird nicht
 benötigt,
                                # ist jedoch
 erlaubt.
                PList G2,       #OK. Name G2
 ist global
             }
```

3. Globale Strukturlisten können an einer äußersten Ebene in einer Strukturlistendatei definiert werden oder können als innerhalb einer umfassenden Strukturliste verschachtelt definiert werden. Die Verschachtelung ist jedoch nur ein Nutzen. Sie sind konzeptionell als globale Strukturlisten an der äußersten Ebene in der Datei definiert. Eine verschachtelte globale Strukturliste ist semantisch einer äußersten (unverschachtelten) globalen Strukturliste des gleichen Namens äquivalent. So zum Beispiel:

```
GlobalPList G1
{
   GlobalPList G2 ...
    }
ist semantisch äquivalent zu:
    GlobalPList G1
    {
      PList G2, # Bezüge G2
    }

    GlobalPList G2 ...
```

4. Alle globalen Strukturlisten sind eindeutig benannt.

```
            GlobalPList G1
            {
              # Zu beachten ist, dass dies so
ist, als
                wäre es an der äußersten Ebene
mit einem
                Bezug darauf direkt hier ver-
einbart.
              GlobalPList G2
              {
                ...
              }
            }

            # Diese Vereinbarung wird in dieser
oder
            # einer beliebigen anderen Datei
ein Fehler
            # sein, weil der Name G2 bereits
genommen
            # wurde.
            # GlobalPList G2 #Fehler. Globaler
Name G2
                                        ist ge-
nommen.
            {
              ...
            }
```

5. Lokale Strukturlisten lassen immer ihre Definitionen innerhalb einer umfassenden Strukturliste, die auch den Umfang des Namens der lokalen Strukturliste bestimmt, verschachteln. Lokale Strukturlisten sind in ihrer umfassenden Strukturliste eindeutig benannt. Den lokalen Strukturlisten ist es syntaktisch nicht erlaubt, an der äußersten Ebene in einer Strukturlistendatei vorzukommen.

```
GlobalPList G1
{
   LocalPList L1
   {
   }

   LocalPList L2
   {
     LocalPList L1  #OK. Kein
lokaler Name
     L1 ist
direkt
        # in dem durch L2 defi-
nierten
          umfassenden Umfang
vereinbart.
     {
     }
     PList L1, #OK. Verweist
auf in L2
                 vereinbarten L1
     PList G1.L1, #OK. Verweist
auf in G1
                    vereinbarten
L1.
   }

   # Fehler. Namen L1 wieder
vereinbaren,
   # wenn der durch G1 defi-
nierte
   # umfassende Umfang bereits
einen in
   # ihm vereinbarten L1 be-
sitzt.
   LocalPList L1,
   {
   }

 }
```

6. Jede Strukturlistendatei enthält die Definition für eine oder mehrere globale Strukturlisten. Diese folgt direkt aus der Syntax. Die äußerste Ebene ist eine Global-Struktur-Listen-Definition, von denen zumindest eine vorhanden sein muss.
7. Der Strukturname ist der Bezug auf eine Struktur, die sich dem Schlüsselwort Pat anschließt. Er bezieht sich auf eine Struktur, die sich in einer Strukturdatei befindet, deren Namen erhalten wird, indem ein Suffix .pat mit dem Strukturnamen verknüpft wird. Die Datei bezeichnet eine Datei, die längs eines für Strukturen definierten Suchepfades erhalten wird.
8. Ein Struktur-Listen-Bezug ist der Bezug auf eine sich dem Schlüsselwort PList anschließende Strukturliste. Der Bezug besteht aus einem optionalen Dateinamen, dem sich ein qualifizierter Strukturlistennamen anschließt, der nur eine Liste von durch Punkte getrennten Namen ist. So könnte zum Beispiel das Folgende ein Struktur-Listen-Bezug:
PList foo.plist:G1.L1.L2.L3,
sein, der sich auf eine lokale Strukturliste L3 bezieht, die verschachtelt ist in L2, der in einem in einer globalen Strukturliste G1 verschachtelten L1 verschachtelt ist, d. h. in einer Datei foo.plist. Das höchstwertige Namenssegment in dem oben erwähnten Namen ist G1.

[0370] Das höchstwertige Namenssegment muss sich entweder zu einer globalen Strukturliste oder auch zu einer lokalen Strukturliste, die vom Bezugspunkt sichtbar ist, auflösen.

[0371] Namensauflösung eines Struktur-Listen-Bezugs geht wie folgt vonstatten:
1. Jedes Namenssegment löst sich zu einem im Zusammenhang mit der vorgesetzten Kennung vor diesem vereinbarten Namen auf.
2. Gibt es eine Dateiqualifizierung, dann löst sich das höchstwertige Namenssegment zu einer in der benannten Datei vereinbarten globalen Struktur auf.
3. Gibt es keine Dateiqualifizierung, dann könnte sich der höchstwertige Namen zu einer lokalen Strukturliste innerhalb des umfassenden Umfangs, und falls dies versagt, dann des nächsten umfassenden Umfangs und so weiter, bis zu einem umfassenden globalen Umfang auflösen.
4. Eine Beschränkung der Suche von Umfängen auf den am nächsten liegenden, umfassenden globalen Umfang wird benötigt, um die Semantik von globalen Umfängen zu schützen, als wären sie an der äußersten Ebene in der Strukturlistendatei vereinbart. Falls der verschachtelte globale Umfang (möglicherweise) wortgetreu an der äußersten Ebene vereinbart wäre, würde die Suche der Namensauflösung enden, nachdem ihr Umfang geprüft ist.
5. Wenn der Bezug durch die vorherigen Schritte nicht aufgelöst worden ist, dann kann das höchstwertige Namenssegment zu einer globalen Strukturliste innerhalb dieser gleichen Datei aufgelöst werden.
6. Wenn der Bezug durch die vorherigen Schritte nicht aufgelöst worden ist, dann kann das höchstwertige Namenssegment zu einer in der Datei benannten globalen Strukturliste aufgelöst werden, indem zu dem höchstwertigen Namenssegment die nachgesetzte Kennung .plist hinzugefügt wird.
7. Wenn der Bezug durch die vorherigen Schritte nicht aufgelöst worden ist, dann ist der Bezug ein Fehler.

[0372] Wie früher erwähnt, schreiben die oben erwähnten Regeln vor, dass das höchstwertige Namenssegment sich entweder zu einer lokalen Strukturliste, die von dem Bezugspunkt sichtbar ist, oder auch zu einer globalen Strukturliste auflöst.

[0373] Das folgende Beispiel veranschaulicht einige dieser Ideen.

```
GlobalPlist G1
{
  PList G3, #OK. Verweist auf eine spätere
            Strukturliste in dieser Datei.
   PList G4, #OK. Verweist auf eine Struk-
turliste
             in Datei „G4.plist"
   #OK. Verweist auf G1 in der Datei
        „my_plists.plist".
   PList my-plists.plist:G1,
```

```
        #OK. Verweist auf eine Strukturliste in
Datei
             "my_plists.plist".
      Der
        # qualifizierte Name verweist auf eine
mit L2
          benannte lokale Strukturlist, die
      vereinbart ist
        # in dem Umfang einer mit L1 benannten
lokalen
          Strukturliste, die in
      dem
        # Umfang einer mit G1 benannten globalen
        Strukturliste vereinbart ist.
        PList my_plists.plist:G1.L1.L2,

        LocalPList L1
        {
          LocalPList L2
          {
          }
        }

        PList L1, #OK. Verweist auf in dem umfas-
  senden
                     Umfang von G1 vereinbarten
  L1
    }

    GlobalPlist G2
    {
      LocalPList L2
      {
      }

      GlobalPList G3
```

```
      {

    LocalPList L3
 {
 }

 PList L1, #Fehler. Kein L1 in diesem oder ir-
gendeinem
           # umfassenden Umfang,

 # Fehler. Der Name L2 ist in diesem Umfang
nicht
 # vereinbart. Obwohl in dem umfassenden Umfang
L2
 # vereinbart ist, ist dieser Umfang global und
somit
 # wird kein weiterer umfassender Umfang ge-
prüft.
 #
 # Kontrast mit Bezug auf Name L2 in LocalPList
L3 unten.
 PList L2,

 PList G1.L1, # OK. Verweist auf L1 in G1.
 # Fehler. G3 ist nicht wirklich innerhalb von
G1
 # verschachtelt. Weil G3 global ist, ist er
wirklich an
 # einer äußersten Ebene vereinbart und so ist
G1.G3
 # bedeutungslos.
 PList G3.G3.L3
}

LocalPList L3
{
```

```
    #OK. Verweist auf G2.L2. Der umfassende glo-
  bale Umfang
    # ist G2 und der Name L2 ist in G2 verein-
  bart.
    PList L2,
  }
}
```

[0374] Alle Namen von Strukturlistendateien und Strukturdateien werden benötigt, damit sie über den sie nutzenden Testplan eindeutig sind.

[0375] Ein Bezug auf Strukturlisten kann auf eine Strukturliste verweisen, die entweder vor oder nach dem Bezug in der gleichen Datei definiert ist.

[0376] Rekursive und wechselseitig rekursive Definitionen von Strukturlisten sind nicht erlaubt. Während in der Syntax der Strukturlistendatei nichts vorhanden ist, was verhindert, dass der Anwender solche Definitionen erzeugt, wird der Analysealgorithmus einen Fehler kennzeichnen, wenn er solche Bedingungen detektiert. Zu beachten ist, dass es einige Kosten gibt, die mit der Erkennung solcher Bedingungen verbunden sind. Der Anwender wird in der Lage sein, die Prüfung abzuschalten, wenn sie/er die Verantwortlichkeit übernehmen kann, zu garantieren, dass der Eingaberaum von wechselseitig rekursiven Definitionen frei ist.

```
GlobalPList G1
{
  LocalPList L2
  {
    LocalPList L
    {
```

```
            #Fehler. L2 arbeitet L3 ab, der L2
abarbeitet.
            # Dies ist ein rekursiver Bezug auf
L2.

            PList L2,

            PList G2,
          }
        }
      }

      GlobalPList G2
      {
        # Fehler. G1.L2 arbeitet L3 ab, der G2
abarbeitet,
        der G1.L2 abarbeitet.
        Dies ist ein wechselseitiger rekursiver
Bezug auf
        G1.L2.
        PList G1.L2,
      }
```

[0377] Die syntaktische Beschreibung von Strukturen und Strukturlisten ermöglicht es, Optionen auf diese zu bestimmen. Im Allgemeinen sind Optionen herstellerspezifisch. Die Syntax ermöglicht es einer beliebigen Struktur oder Strukturliste eine Anzahl von Optionen zu besitzen, die jeweils mit einer Anzahl von Parametern speziell festgelegt ist. Wir beschreiben hier einige unterstützte Optionen, die durch die meisten Hersteller erkannt werden.

[0378] Die dynamische (d. h. Ausführung) Semantik von baumförmigen Strukturen wird hier nach Definition einer Strukturausführungssequenz beschrieben.

E2. Strukturen

[0379] Fig. 6 stellt einen Strukturkompilierer **602** und einen Strukturlader **604** nach einer Ausführung der vorliegenden Erfindung dar. Der anwenderdefinierte Inhalt einer Struktur ist in einem Strukturquellenfile **606** verfügbar, der eine Klartextdatei ist. Ein Strukturkompilierer wird für das Kompilieren eines Quellenfiles in ein modulspezifisches Format, das zum Laden auf die Tester-Hardware geeignet ist, verantwortlich sein, wobei diese letztere Datei als die Strukturobjektdatei bezeichnet werden wird. Das folgende sind die allgemeinen Attribute:

1. Ein Strukturobjekt kann durch den Anwender nicht erzeugt werden, vielmehr ist der Anwender immer mit Strukturlisten befasst, die Sammlungen von anderen Strukturlisten und/oder Strukturen sind. Ein Strukturlistenobjekt erzeugt die Strukturobjekte, die in ihm enthalten sind, besitzt sie und behält sie bei, während sie bei Bedarf dem Anwender zugänglich gemacht werden.
2. Eine Struktur wird innerhalb eines Testplans eindeutig benannt, d. h. zwei Strukturen innerhalb des Testplans können nicht den gleichen Namen besitzen. Der Name einer Struktur ist unterschiedlich zu dem Namen der ihn enthaltenden Datei. Der Strukturdateiname ist der einzige in der Strukturlistendatei verwendete, der auf eine Struktur verweist, während der tatsächliche Name der Struktur in der Strukturdatei definiert ist.

[0380] In einer Ausführung der Erfindung könnte im Allgemeinen ein einzelnes DUT (Prüfobjekt) an Testermodule von unterschiedlichen Herstellern angeschlossen sein. Diese weist Implikationen für die gesamte Kette des Kompilierens, Ladens, Ausführens der Struktur auf. Die Hauptsächlichen werden in diesem Abschnitt beschrieben.

E3. Strukturkompilierung

[0381] Ein Strukturkompilierer **602** muss somit eine spezifische Standortkonfiguration (unter dem Aspekt der verwendeten herstellerspezifischen digitalen Module) treffen. Für den Rest dieser Erörterung wird der Begriff „Modul" verwendet, um als Beispiel auf ein digitales Modul zu verweisen. Um die Integration von Modulen **608** von unterschiedlichen Herstellern in das System zu ermöglichen, werden die folgenden Verfahren bevorzugt:

1. Jeder Modulhersteller wird dafür verantwortlich sein, seinen eigenen modulspezifischen Strukturkompilierer **610** in Form einer dynamisch ladefähigen Bibliothek oder getrennten ladefähigen Datei bereitzustellen. Diese Kompiliererbibliothek/ladefähige Datei wird allermindestens eine bekannte Kompilierfunktion () bereitstellen, die als Argumente nimmt
   a. eine Matrix von (einem oder mehreren) Pfadnamen der Strukturquellenfiles,
   b. den Dateinamen von Pinbeschreibungen,
   c. den Namen des Socket-File,
   d. einen optionalen Verzeichnis-Pfadnamen, der das Ziel des kompilierten Objektes bestimmt,
   e. eine optionale Matrix von Sequenzname/Wertpaaren, die die Spezifizierung von beliebigen herstellerspezifischen Parametern (die durch andere Hersteller ignoriert werden können) ermöglicht.
2. Der Strukturquellenfile wird zwei unterschiedliche Typen von Abschnitten aufnehmen:
   a. einen „gemeinsamen" Abschnitt, der Informationen enthalten wird, die für alle Kompilierer zugänglich sind (von diesen jedoch nicht zwangsläufig verwendet werden), und
   b. einen oder mehrere jeweils durch eindeutige Herstellercodes identifizierte, herstellerspezifische Abschnitte, verwendbar zur Information durch spezifische Kompilierer von Herstellern.
3. Ein Kompilierer des Herstellers wird nicht direkt eine Strukturobjektdatei erzeugen. Stattdessen wird der Tester ein Strukturobjekt „Bilddatei" **612** bereitstellen, das von einem Objektdateiverarbeitungsprogramm (OFM) **614**, das Teil des Strukturkompilierers ist, verwaltet wird. Der Strukturkompilierer kann auf dem Computer als die Systemsteuereinheit wirksam sein oder vom Netz getrennt angeordnet sein, z. B. auf einem Netzwerk, mit dem die Systemsteuereinheit verbunden ist. Die „Strukturobjektdatei", auf die in abstrakten Begriffen insoweit hingewiesen wurde, ist tatsächlich diese Objektbilddatei. Die Objektbilddatei wird genauso benannt wie der Strukturquellenfile, wobei die Erweiterung des Quellenfiles durch die Erweiterung der Objektdatei ersetzt ist. Das OFM wird eine Anwendungsprogrammierschnittstelle (API) zur Verfügung stellen, um diese Datei zu lesen und zu schreiben. In der Objektbilddatei sind Vorkehrungen getroffen zum Speichern von
   a. gemeinsamen Headerinformationen,
   b. modulspezifischen Headerinformationen einschließlich Informationen, die das entsprechende Modul und die Stelle von Strukturdaten für das Modul erkennen,
   c. modulspezifischen Strukturdaten, die wie durch den Modulhersteller benötigt organisiert und in der Lage sind, durch den Modulhersteller interpretiert zu werden.

[0382] Die API des OFM wird dem Kompilierer eines Modulherstellers erlauben, modulspezifische Headerinformationen und Daten in die Objektbilddatei zu schreiben. Zu beachten ist, dass es dieses Layout der Objektbilddatei ermöglicht, die Strukturdaten auf der Basis je Modul auch in dem Fall, wenn zwei oder mehrere Module an dem getroffenen Standort identisch sind, zu organisieren.

[0383] Zu beachten ist, dass von Strukturkompilierern zusätzliche von dem Hersteller gelieferte Konfigurationsinformationen benötigt werden könnten, um die Erzeugung von modulspezifische Hardware ladenden Informationen zu erleichtern, die aus effizienter Datenkommunikation wie direkter Speicherzugriff (DMA) Vorteil ziehen können.

E4. Strukturladen für ein Modul

[0384] Jeder Modulhersteller wird dafür verantwortlich sein, seinen eigenen Strukturlademechanismus **615** vorzusehen, dem die allgemeine Prozedur folgt. Die Strukturobjekt-Bilddatei **612** eines Moduls **608** speichert modulspezifische Daten in unterschiedlichen Abschnitten **616**. Die Implementierung des Herstellers wird die API des OFM nutzen, um auf relevante modulspezifische Abschnitte aus der Strukturobjekt-Bilddatei zuzugreifen. Der Testerrahmen wird verantwortlich dafür sein, jedes Ladeverfahren des Moduls aufzurufen, um wiederum modulspezifische Daten für ein Modul aus dem entsprechenden Abschnitt der Bilddatei zu laden.

E5. Strukturdateien

[0385] Es ist möglich, jeden Hersteller von Kompilierern völlig unterschiedliche Klartextformate für Strukturen bestimmen zu lassen, die genau gesagt, tatsächlich in den meisten Fällen notwendig sein könnten. Jedoch ist im Allgemeinen für eine Testumgebung auf Zyklusbasis, bei der eine kohärente und eine identische Semantik mitten durch Module für jeden Vektor notwendig sind, eine gemeinsam genutzte verallgemeinerte Syntax für die Strukturdatei nicht nur erwünscht, sondern kann notwendig sein. Diese gemeinsam genutzte Syntax ist das, was für den „gemeinsamen" Abschnitt im Strukturquellenfile spezifiziert werden wird. Genau gesagt, für die Mehrheit von Fällen stellt man sich vor, dass der „gemeinsame" Abschnitt der einzige Abschnitt ist (neben Kopfinformationen), der in der Strukturdatei benötigt wird, und jeder Kompilierer des Herstellers nur mit diesem Abschnitt arbeiten wird. Dieser Abschnitt stellt Regeln für die Strukturdatei dar, die alle Kompilierer interpretieren können sollten. Die Strukturdatei wird wie folgt organisiert werden:

```
    file_contents       :
                              versi-
                on_info_pattern_definitions
    version_info        :
                     Version version-identifier
';'

    pattern_definitions :
                pattern_definition
                pattern_definitions pat-
  tern_definition
  pattern_defintion
                main_header'{'main_section'}'
                main_header'{'main_section ven-
  dor_sections'}'
                subr_header'{'subr_section'}'
                subr_header'{'subr_section ven-
  dor_sections'}'
  main_header           :
                MainPattern identifier
  main_section          :
                CommonSection '{'common_contents
  main_section_domains'}'
  common_contents       :
                  timing_reference tim-
  ing_map_reference
```

```
timing_reference      :
                Timing file-name';'
timing_map_reference     :
                TimingMap file-name';'
main_section_domains     :
                main_section_domains
main_section_domain
                main_section_domain
main_section_domain
            Domain do-
main_name'{'main_section_contents'}'
domain_name              :
                identifier
main_section_contents :
                main_section_contents
main_section_content
                main_section_content
main_section_content :
                label_spec main_pattern_spec
                main_pattern_spec
label_spec      :
                label';'
label           :
                identifier
main_pattern_spec    :
                main_operation cap-
ture_mask_flag'{'
vectors_and_waveforms'}'
main_operation        :/*empty*/
                common_operation
                jal_op
                jsr_op
                jsrc_op
                jsc_op
                exit_op
common_operation    :
```

```
                idxi_op
                idxin_op
                jec_op
                jech_op
                jff_op
                jffi_op
                jni_op
                ldin_op
                nop_op
                pause_op
                sndc_op
                sndt_op
                stfi_op
                sti_op
                stps_op
                wait_op
/*
* Für MAIN Strukturen spezifische Anweisungen
*/
jsr_op          :
                JSR Identifizierer
jsrc_op         :
                JSRC Identifizierer
jsc_op          :
                JSC Identifizierer
jal_op          :
                JAL Identifizierer
exit_op         :
                EXIT
/*
 * Anweisungen, die für MAIN und SUBR Struktu-
ren gemeinsam sind
*/
idxi_op         :
                IDXI 24-bit Zahl
idxin_op        :
```

```
                IDXIn Indexregister
jec_op          :
                JEC Identifizierer
jech-op         :
                JECH Identifizierer
jff_op          :
                JFF Identifizierer
jffi_op         :
                JFFI Identifizierer
jni_op          :
                JNI Identifizierer
ldin_op         :
                LDIN Indexregister
nop_op          :
                NOP
pause_op        :
                PAUSE
sndc_op         :
                SNDC 8-bit Zahl
sndt_op         :
                SNDT 8-bit Zahl
stfi_op         :
                STFI 24-bit Zahl
sti_op          :
                STI 24-bit Zahl
stps_op         :
                STPS
wait_op         :
                WARTEN
capture_mask_flag   :/*empty*/
                    capture_mask_flag CTV
                    capture_mask_flag MTV
                    capture_mask_flag MATCH

vectors_and_waveforms : /*epmty */
                    vectors_and_waveforms vector
```

```
                vectors_and_waveforms_waveform
vector        :
              vector_declaration '{'vec-
tor_data'}'
vector-declaration  :
              Vektor
              V
vector_data     :
              Vector_datum
              vector_data vector_datum
vector-datum    :
              pin_name '=' Vektor-Wert ';'
              pin_name '=' Identifizierer ';'
waveform       :
               waveform_declaration '{'wave-
form_data'}'
waveform_declaration  :
              Wellenform
              W
waveform_data          :
            waveform_datum
            waveform_data waveform_datum
waveform_datum          :
          waveform-table-pin-group-name'='
identifier ,;'
pin_name      :
            Identifizierer
Hersteller_Abschnitte  :
            Hersteller_Abschnitte Hersteller-
Abschnitt {}
            Hersteller_Abschnitt {}
Hersteller_Abschnitt   :
      HerstellerAbschnitt '{'Hersteller-
Abschnitt_Inhalt'}'
Subr_header
      SubrPattern
```

```
subr_section :
      GemeinsamerAbschnitt
source_selection_table subr_section_domains'}'
      GemeinsamerAbschnitt '{' common_contents
subr_section_domains '}'
subr_section_domains  :
               subr_section_domains
subr_section_domain
               subr_section_domain
subr_section _domain :
      Domain domain_name
'{'subr_section_contents'}'
source_selection_table :
 Quellenauswahlta-
belle'{'source_selector_definitions '}'
source_selector_definitions :
                source_selector_definitions
source_selector_definition
 source_selector_definition

source_selector_definition:
    Quellenauswähler source-selector_name '{'
source_mappings '}'
source_selector_name  :
         Identifier
source_mappings  :
        source_mappings source_mapping
        source_mapping
source_mapping   :
       pin_name '=' source ';'
source    :
          MAIN
    INVERT_MAIN
    SUBR
    INVERT_SUBR
subr_section_contents :
```

```
                subr_section_contents
        subr_section_content
                subr_section_content
subr_section_content    :
                label_spec subr_pattern_spec
                subr_pattern_spec
subr-pattern_spec    :
               subr_operation cap-
ture_mask_flag'{'
vectors_and_waveforms'}'
subr_operation    : /*empty */
               common-operation
               rtn_op
               stss_op
/*
 * Für SUBR Strukturen spezifische Anweisungen
 */
rtn_op    :
               RTN
stss_op        :
               STSS Identifizierer
```

[0386] Das Folgende sind die Beschreibungen von oben verwendeten, undefinierten Nicht-Eingängen:

1. version-identifier: Eine Folge von einem oder mehreren Zeichen aus der Menge [0-9], in der das erste Zeichen eine Ziffer sein muss.
2. identifier: Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], in der das erste Zeichen aus der Menge [a-zA-Z_] sein muss.
3. vendor-section-content: Beliebiger Text, der nur für einen herstellerspezifischen Kompilierer bedeutungsvoll ist.
4. file-name: Ein gültiger Windows-Dateiname (muss in doppelten Anführungszeichen umschlossen sein, falls irgendwelche Lücken in dem Dateinamen enthalten sind). Zu beachten ist, dass dieser ein einfacher Dateiname sein soll, d. h. er sollte keine Verzeichniskomponente besitzen.
5. waveform-table-pin-group-name: Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], in der das erste Zeichen aus der Menge [a-zA-Z_] sein muss. Diese Variable ist irgendwo vereinbart und hält den Namen der Wellenform-Tabelle, die einer Gruppe von Pins gemeinsam ist.
6. 24-bit Zahl: Eine gültige Dezimalzahl bis zu einem Maximum von 16777215.
7. 8-bit Zahl: Eine gültige Dezimalzahl bis zu einem Maximum von 256.
8. index-register: Eine gültige Dezimalzahl. In einer Ausführung eines Moduls kann diese einen Wert [1-8] besitzen.
9. vector: Dieser ist der Vektoranweisung in STIL ähnlich. Zu beachten ist, dass dieser auf Signalnamen und Signalgruppennamen verweist, die es notwendig machen, dass der Kompilierer Zugriff auf die Pinbeschreibungsdatei hat.
10. waveform-time-reference: Eine Folge von einem oder mehreren Zeichen aus der Menge [a-zA-Z_0-9], in der das erste Zeichen aus der Menge [a-zA-Z_] sein muss.

[0387] Strukturdateien werden Kommentare, die dazu bestimmt sind von einem Strukturdateikompilierer ignoriert zu werden, unterstützen. Kommentare werden mit dem Zeichen '#' beginnen und sich bis zum Ende der Zeile erstrecken.

[0388] Die folgenden Punkte sollten mit Bezug auf die Konstrukte in dem Dateiheader der Strukturdateien und

„gemeinsamen" Abschnitten beachtet werden.

1. Das Strukturnamen-Element bestimmt den Namen, der mit dem Strukturobjekt, für das die Strukturdatei die Daten enthält, verknüpft werden wird. Dieses wird zu dem Dateiheader in der entsprechenden Strukturobjekt-Bilddatei übertragen.
2. Der Wellenformen-Zeit-Bezug ist der Name für eine spezielle Definition von Wellenform-und-Zeitsteuerung, die extern zur Strukturdatei in der Zeitsteuerungsdatei definiert werden würde. Die Spezifikation eines Wellenform-Zeit-Bezuges in der Strukturdatei würde diesen speziellen Namen (für eine Wellenform-und-Zeitsteuerung) an alle nachfolgenden Vektoren binden, bis man auf einen anderen Wellenform-Zeit-Bezug stoßen würde.
3. Der Operand für einen Subroutinen-Aufruf (z. B. JSR und JSRC) ist eine Datenfolge, die entweder ein Pattern-spec-Kennsatz, auf den man zuvor in der gleichen Strukturdatei stößt, oder ein Pattern-spec-Kennsatz in einer extern definierten Subroutinenstruktur sein sollte. Dieser Operand wird letzten Endes zum Zweck des Ladens/Verarbeitens von Subroutinen aufgelöst. Es ist nötig, dass die Kennsätze für Subroutinenaufruf-Operanden über das System eindeutig sind.

[0389] Während Namen von Wellenform-Zeit-Bezug irgend etwas sein könnte, das syntaktisch korrekt ist, ist zu beachten, dass auf Grund von spezifischen Hardwareimplikationen die Namen von Wellenform-Zeit-Bezug auf eine vorher bekannte, genau festgelegte Menge (die zur ergänzten Lesbarkeit durch den Anwender optional zu anwendergewählten Namen abgebildet werden können, wobei die Auflistung in einer Wahldatei dargestellt ist) eingeschränkt werden könnten.

[0390] Außerdem ist zu beachten, dass die Strukturdatei und der Quellenfile Wellenformen-Zeit-Bezug anfängliche Konfigurationsdaten für alle DUT-Kanäle zur Verfügung stellen sollten, die Verbindungen zu physikalischen Testerkanälen besitzen. Falls nachfolgende Daten für einen beliebigen DUT-Kanal übergangen werden, wird der Struktur-Kompilierer die Strukturdaten „aufblähen", um eine Ausgabe von der Anfangsebene einzuhalten.

Beispiel einer Strukturdatei

[0391] Das einfache Beispiel eines Struktur-Quellenfiles MAIN wird helfen, die Verwendung zu veranschaulichen.

```
#
# Filename: good1.pat
#
Version 1.0;
# ----------------------------------------
# Definition der Hauptstruktur
```

```
      # ------------------------------------
      Hauptstruktur good1
      {
      Gemeinsamer Abschnitt
   {
   MacroDef     defaultData Val   (XXXXXXXX)
   MacroDef     nopInstr        (NOP
   MacroDef     label1          (Label1:)
   MacroDef     jniInst         (JNI)

   # ---------------------------------------
   # Zeitsteuerungsbedingungen
   #----------------------------------------
     Zeitsteuerung "productionTiming.tim";
     Zeitsteuerungsplan "productionTimingO-
penSTARMap.tmap";

   #----------------------------------------
   # Zyklen der Vorgabedomäne
   #----------------------------------------
   Domänevorgabe
   {
   #-----------------------------------------------
-
   #Kennsatz: Anweisung {Vektor/Wellenform Daten}
   #-----------------------------------------------
-
       NOP          {V{DATA = $defaultDataVal; CLK
= 1;} W
{DATA = wfs1; CLK = wfs1;}}
     JAL  myAPG  {V {DATA = 00000000;}}
     JSC  mySCAN  {V {DATA = 10101010;}}
     JSRC mySubroutine {V {DATA = 01010101; }}
     JSR myAPG   {V {DATA = 00110011;}}
     STI 100    {}
labZero: NOP      {V {DATA = 00000011;}}
```

```
     JNI labZero {V {DATA = 11111100;}}
     IDXI 3000   {V {DATA = 10101010;}}
     IDXIn 3    {V {DATA = 01010101;}}
$label1 NOP      {V {DATA = $defaultDataVal;}}
        IDXI 2000 {V{DATA = 10101010;}}
        NOP      {}
        EXIT     {V{DATA = LLHHLLHH;}}
 }
  }
}
```

Ein weiteres, einen Strukturquellenfile SUBROUTINE veranschaulichendes Beispiel ist nachstehend dargestellt.

```
#-------------------------------------------
# Definition Subroutinenstruktur mySubrPat1:
#-------------------------------------------
SubrPattern mySubrPat1
{

 gemeinsamer Abschnitt
 {
  #-----------------------------------------
  # Zeitsteuerungsbedingungen
  #-----------------------------------------
  zeitliche Steuerung „productionTiming.tim";
  Zeitsteuerungsplan „productionTimingOpensSTAR-
Map.tmap";

  #-----------------------------------------
  # Quellenauswahlbedingungen
  #-----------------------------------------
    Quellenauswahltabelle
    {
    Quellenauswähler SrcDef
```

```
      {
          DATA = SUBR; CLK = SUBR; DATA = SUBR;
        }
        Quellenauswähler SrcSelOne
        {
          DATA = MAIN; CLK = MAIN;
        }
       }
  #-----------------------------------
  # Vorgabedomänezyklen
  #-----------------------------------
  Domänenvorgabe
  {
  #-----------------------------------
  #Kennsatz: Anweisung  {Aufstellung von Vektor
und
                                    Wellenformda-
ten}
  #-----------------------------------
          STI  100 {Vektor {DATA = 00000000;}}
          IDXI 3000 {Vektor {DATA = 00001111;}}
          IDXIn 3  {Vektor {DATA = 00110011;}}
        $label1 NOP {Vektor {DATA = LLHHLLHH;}}
            NOP    {Vector {DATA = LLXXXXXX;}}
            NOP    {Vektor {DATA = LLHHXXXX;}}
            JNI Label1 {Vektor {DATA =
LLHHLLHH;}}
            STSS SrcSelOne {Vektor {DATA =
LHLHLHLH;}}
            RTN      {Vektor {DATA = LLXXXXXX;}}
        }
       }
 }
```

[0392] Zusammenfassende Informationen aus dem Hauptkopfsatz und dem gemeinsamen Abschnitt im Strukturquellenfile werden in dem Hauptkopfsatz in der Objektbilddatei gespeichert. Die Zusammenfassung besteht aus Informationen, die typischerweise zur schnellen Extraktion benötigt werden, um das vorherige Laden einer Auflösung von Adressen, usw. zu unterstützen oder bei der Datenerfassung zu unterstützen. Weil die Semantik des gemeinsamen Abschnitts exakt die gleiche für alle Kompilierer ist, wird jeder Kompilierer in der Lage sein, die gleichen zusammenfassenden Informationen bereitzustellen, wobei der erste die Bilddatei schreibende Kompilierer diese Informationen speichern wird. Das folgende sind die Informationen, die gespeichert werden:

1. Der Name des Strukturquellenfiles.
2. Der Typ der Struktur wie im Quellenfile vereinbart.

3. Die Versionsinformationen von dem Quellenfile.
4. Eine Liste aller Wellenformen und Zeitsteuerungsnamen, die im gemeinsamen Abschnitt des Strukturquellenfiles verwendet werden.
5. Ein Plan aller Subroutinen-Bezüge auf (relative) Vektoradressen im gemeinsamen Abschnitt des Strukturquellenfiles.
6. Ein Plan aller Kennsatzbezüge auf (relative) Vektoradressen im gemeinsamen Abschnitt des Strukturquellenfiles.
7. Allgemeine Buchhaltungsinformationen: Vektorzählung, Anweisungszählung, usw..

[0393] Das Testsystem mit offener Architektur erfordert sowohl Strukturdateien als auch Strukturlistendateien, um explizite und unterschiedliche Erweiterungen zu haben. Für Strukturdateien gilt dies sowohl für Klartextquelle als auch kompilierte Objektdateien. Dies wird als Erleichterung für den Anwender angesehen, um den Dateityp visuell in einer Verzeichnisaufstellung, usw. schnell zu identifizieren sowie Verknüpfungen auf der Basis von Erweiterungen herstellen zu können. Der Strukturlistendatei-Syntaxanalysator wird Dateinamen mit diesen Erweiterungen erwarten:

```
Klartext-Strukturquellenfile: .pat
kompilierte Strukturobjekt-Bilddatei: .pobj
Strukturlistendatei:          .plst
```

[0394] Der Anwender kann diese Vorgabewerte, z. B. durch Variable der Testerumgebung oder Einstellungsoptionen aufheben.

[0395] Der Tester wird die Definition der folgenden "Umgebungsvariablen" für Dateisuchpfade in zumindest einer der hier beschriebenen Umgebungskonfigurationsdateien benötigen:
Tester_PATLIST_PATH: Für Strukturlistendateien.
Tester_PATSRC_PATH: Für Strukturquellenfiles (optional).
Tester_PATOBJ_PATH: Für Strukturobjekt-Bilddateien.

[0396] Zu beachten ist, dass, wenn die optionale Umgebungs-Einstellungsvariable Tester_PATSRC_PATH nicht definiert ist, sie als die gleiche wie Tester_PATOBJ_PATH vorausgesetzt wird. Allgemein wäre es effizienter, Tester_PATSRC_PATH nicht zu definieren als sie mit dem gleichen Wert wie Tester_PATOBJ_PATH zu definieren.

E6. Softwaredarstellung

[0397] Ein Strukturobjekt wird nicht durch den Anwender erzeugt, vielmehr befasst sich der Anwender immer mit Strukturlistenobjekten, die Sammlungen von anderen Strukturlisten und/oder Strukturen sind. Ein Strukturlistenobjekt erzeugt die in ihm enthaltenen Strukturobjekte, besitzt sie und behält sie bei, während sie dem Anwender zugänglich gemacht werden. Ein Strukturlistenobjekt im Anwendertestprogramm ist mit einer Strukturlistendatei auf Festplatte, die die aktuelle Definition der Strukturliste enthält, verknüpft. Die Definition einer Strukturliste stellt einen expliziten Namen für die Strukturliste bereit und identifiziert eine geordnete Liste von Strukturen und/oder anderen Strukturlisten durch Verknüpfungen von Dateinamen. Dieser Abschnitt beschreibt die Softwaredarstellung von Strukturlisten und Strukturen als eine Einleitung zum Verständnis dessen, wie sie in dem Testerrahmen gehandhabt werden.

Verknüpfungen von Strukturlisten

[0398] Ein einzelner Messplatz in dem Testsystem (und durch Erweiterung die in ihm befindlichen Prüfpläne) kann mit mehreren Strukturlisten oberster Ebene verknüpft werden. Jedoch gibt es zu einem beliebigen Zeitpunkt nur einen einzelnen Abarbeitungskontext für Testpläne. Weil eine Strukturliste oberster Ebene eine Abarbeitungsfolge für die Strukturen definiert, auf die durch sie (hierarchisch) verwiesen wird, ist der aktive Abarbeitungskontext der, der der gegenwärtig ausgewählten Strukturliste oberster Ebene entspricht. Zu beachten ist, dass dies nicht darauf hinausläuft, dass nur die in einer einzelnen Strukturliste enthaltenen Strukturen gleichzeitig in die Hardware geladen werden können, vielmehr muss die Menge von Strukturen, die in die Hardware geladen werden müssen, um eine Folge von Abarbeitungen vorzunehmen, immer eine Teilmenge aller gegenwärtig geladenen Strukturen sein.

Strukturbäume

[0399] Intuitiv fühlt man, dass eine Möglichkeit der Darstellung einer Strukturliste oberster Ebene durch eine bestimmte Art einer baumförmigen Datenstruktur erfolgt. Fig. 7 stellt eine Ausführung eines geordneten Strukturbaums nach der Erfindung dar, in der vorausgesetzt wird, dass die Strukturliste A die Strukturliste oberster Ebene ist.

Informationsinhalt von Strukturbäumen

[0400] Die folgenden Informationen werden an jedem Knoten des Strukturbaums gespeichert:

1. Der Name der mit diesem Knoten verknüpften Entität (Strukturliste oder Struktur).
2. Der Typ der Definitionsquelle. Für einen Knoten (Strukturknoten) wird dieser immer eine Strukturdatei sein; für einen Zwischenknoten (Strukturliste) könnte dieser entweder eine „Datei höchster Ebene" (für Strukturlistendefinitionen höchster Ebene) oder „in eine Datei eingebettet" sein (für verschachtelte Strukturlistendefinitionen).
3. Die letzte Modifizierungs-Zeitmarkierung der Datei auf der Platte, mit der der Knoten verknüpft ist.

[0401] Die folgenden zusätzlichen Informationen werden nur in Zwischenknoten (Strukturliste) gespeichert:

1. Abarbeitungsoptionen (falls vorhanden), die auf das durch diesen Knoten dargestellte Strukturlistenobjekt gesetzt sind, d. h. seine Objektoptionen.
2. Die Abarbeitungsoptionen (falls vorhanden), die auf jeden Tochterbezug innerhalb der durch diesen Knoten dargestellten Strukturlistendefinition gesetzt sind, d. h. die Bezugsoptionen für jeden seiner Tochterknoten.

[0402] Die Sammlung von Knoten, denen man auf dem eindeutigen Pfad von der Wurzel zu einem Zwischenknoten begegnet, und die Sequenz, der sie begegnen, enthalten so alle Informationen, die notwendig sind, um die kombinierten durch diesen Knoten dargestellten, effektiven Ausführungsoptionen festzulegen. Die Ausführungsoptionen einer Struktur werden festgelegt durch die effektiven Ausführungsoptionen ihrer unmittelbaren Mutter, kombiniert mit den Bezugsoptionen, die ihre unmittelbare Mutter für sie haben könnte.

[0403] Während sich der Strukturlisten-Syntaxanalysator im Prozess der Erzeugung des Strukturbaums befindet, soll hier beachtet werden, dass bestimmte Ausführungsoptionen eine anfängliche Speicherung von Werten einfach als Sequenzen erfordern könnten, weil der Kontext ihrer Verwendung nicht erst später aufgelöst werden könnte. Beispiel einer solchen Option ist eine „Maskenoption", die PIN-Maskeninformationen festlegt: Strukturlisten sind nicht mit Socket-Informationen verknüpft, und folglich werden Pinmaskenoptionen (Pin- und Gruppennamen) als Sequenzen gespeichert, um vor dem Laden aufgelöst zu werden.

[0404] Die folgenden zusätzlichen Informationen werden nur in blattartigen Knoten (Struktur) gespeichert:

1. Alle (vielleicht transitiven) Bezüge auf durch diese Struktur aufgerufene Unterprogramme, sowohl externe als auch interne, die als ein Abarbeitungsbaum organisiert sind.

[0405] Natürlich werden alle Strukturknoten außerdem Zugriff auf alle zusammenfassenden Informationen von Strukturdateien, die im gemeinsamen Objektbilddatei-Dateiheader verfügbar sind, haben und könnten wählen, um im Cache abzuspeichern.

Handhabung von Strukturlisten-Modifizierungen

[0406] Änderungen, die am Inhalt einer Strukturliste vorgenommen werden, beeinflussen konzeptionell alle Bezüge auf diese Strukturliste. Die folgenden Regeln, die sowohl auf Strukturobjekte als auch Strukturlistenobjekte geeignet anwendbar sind, werden genutzt, um solche Änderungen zu verwalten:

1. Eine am Inhalt einer Strukturlistendatei vorgenommene Änderung auf Platte wird durch das Testsystem nur bei einem Ladebefehl () verbreitet werden, der auf dieser Strukturliste (oder auf einer beliebigen anderen Strukturliste, die sich auf diese eine bezieht) ausgeführt wird. Mit anderen Worten, die Hierarchie von Strukturlisten in Software wird immer die eine gegenwärtig auf die Hardware geladene widerspiegeln.
2. Der Anwender wird in der Lage sein, einen Modus zu setzen, der die Prüfungen ablehnen wird, die während einer Ladezeit zum Synchronisieren von Strukturlisten mit ihren Plattendateiquellen gemacht wurden, was eine schnellere/sicherere Rechenoperation im Herstellungsmodus ermöglichen wird.

Strukturbaum-Navigation

[0407] Die mit einem Messplatz verknüpften Strukturlisten oberster Ebene (und durch Erweiterung mit einem Testplan für diesen Platz) haben einen gemeinschaftlich nutzbaren (globalen) Umfang. Das System stellt Anwendungsprogrammierschnittstellen (API) zur Verfügung, um in dem eine Strukturliste oberster Ebene darstellenden Strukturbaum zu navigieren, so dass die Anwender Zugriff auf einzelne Knoten und untergeordnete Bäume erhalten können.

E7. Strukturlisten-Dynamik

[0408] Vorher wurden die statischen Regeln von Strukturlisten beschrieben. Jetzt wird eine Beschreibung der dynamischen Regeln (Abarbeitung) von Strukturlisten dargestellt.

[0409] Der Strukturbaum ist für das allgemeine Strukturmanagement wichtig. Zum Beispiel ist der Anfangspunkt für eine Strukturenladesequenz ein Aufruf an das Ladeverfahren () auf dem gegenwärtig mit dem Standort oder dem Testplan verknüpften Strukturbaum. Ein Strukturbaum arbeitet jedoch nicht für sich betrachtet. Es wird ein völlig initialisierter Strukturbaum verwendet, um die folgenden zwei Rahmenobjekte zu schaffen:

1. Eine Strukturliste oberster Ebene definiert eine Strukturabarbeitungssequenz für die Strukturen. Sie beschreibt, wie eine solche Abarbeitungssequenz von dem Strukturbaum, der dieser Strukturliste oberster Ebene entspricht, abgeleitet werden kann. Zum Beispiel ist die Strukturabarbeitungssequenz, die dem in Fig. 7 dargestellten Strukturbaum A entspricht, {q, s, t, q, r, q, u, u, v}. Die Strukturabarbeitungssequenz ist konzeptionell eine geordnete Liste, die die durch den Strukturbaum beschriebene Abarbeitungssequenz widerspiegelt. Der Rahmen baut beliebige notwendige Navigationsverbindungen zwischen Strukturbaumknoten und entsprechenden Eingaben in die Strukturabarbeitungssequenz auf und behält diese bei.
2. Die Strukturgröße, die einfach eine Liste aller eindeutigen Strukturen (einschließlich Unterprogramme) im Strukturbaum ist. Diese ist folglich die Liste, die verwendet wird, um die einzelnen Strukturen zu bestimmen, die auf die Hardware geladen werden sollen. Der Rahmen baut beliebige notwendige Navigationsverbindungen zwischen Strukturbaumknoten und entsprechenden Eingaben in der Strukturmenge auf und behält diese bei. Die Strukturmenge für den Strukturbaum von Fig. 7 ist {q, s, t, r, u, v} (vorausgesetzt wird, dass keine der Strukturen in der Strukturliste A irgendwelche Aufrufe für Unterprogramme enthält):

Zu beachten ist, dass sowohl die Strukturabarbeitungssequenz als auch die Strukturgröße immer vom Strukturbaum abgeleitet werden kann, wobei es jedoch oft Sinn machen würde, sie nach einer anfänglichen Konstruktion solange im Cache zu speichern wie sie lebensfähig wäre.

Abarbeitungsoptionen für Strukturliste

[0410] Wie oben gezeigt ist, kann jeder Strukturlistenvereinbarung (die ihrer Definition vorangeht) oder Strukturliste/Strukturbezugseingabe eine Anzahl von Abarbeitungsoptionen folgen. Strukturlisten-Abarbeitungsoptionen modifizieren die Laufzeitausführung von Strukturlisten. Um zukünftige Erweiterungen zu erlauben, werden die Namen (und optionale Werte) für diese Optionen durch den Strukturlistendatei-Syntaxanalysator des Strukturkompilierers einfach als Zeichenfolgen behandelt, um durch eine spezielle Version als geeignet interpretiert zu werden. Tester schreibt einen Satz von Optionen und ihre Interpretationen vor, die nachstehend beschrieben werden. Jedoch können Hersteller den Satz von Optionen erweitern. Um eine Validierung der Syntaxanalysenzeit von wahlfreier Angabe zu ermöglichen, könnte der Strukturlistendatei-Syntaxanalysator eine Informationsdatei für eine spezielle Version lesen.

[0411] Eine solche Informationsdatei könnte auch genutzt werden, um zu bestimmen, ob eine spezielle Version die Spezifizierung von Abarbeitungsoptionen überhaupt unterstützt.

[0412] Für Versionen, die einen Satz von Abarbeitungsoptionen unterstützen, werden die folgenden allgemeinen Regeln ihre Verwendung verwalten. Um diese Regeln zu verstehen, ist es nützlich, die hierarchische Sammlung von Strukturlisten/Strukturen als einen geordneten Baum sichtbar darzustellen.

1. Eingeprägte Optionen, die auf Strukturlistendefinitionen gesetzt sind (d. h. in den Produktionen „local-Pattern-list-declaration, global-Pattern-list-declaration" in der Datei) sind eigentlich direkte Optionseinstellungen an dem entsprechenden Strukturlisten-Objekt im Testprogramm des Anwenders. Sie lassen sich somit auf alle Bezüge auf dieses Strukturlistenobjekt anwenden und werden als Objektoptionen bezeichnet.
2. Referenzoptionen, die auf Bezüge für Strukturlisten/Strukturen (d. h. in den Produktionen „Pattern-entry" und „Pattern-list-entry") in der Datei gesetzt sind, begrenzen den Umfang der Optionen auf einen spezifischen Pfad in der Hierarchie, den Pfad (durch die Vereinbarungsreihenfolge von Strukturlisten/Strukturen

aufgestellt), der von der Wurzel des Baumes zu dem in Betracht kommenden Bezug führt. Diese sind somit Optionen auf spezifische Objektbezüge (und nicht auf die Objekte selbst) und werden als Referenzoptionen bezeichnet.

3. Die effektiven Optionseinstellungen für eine beliebige Liste/Struktur in der Sammlungshierarchie (durch die Vereinbarungsreihenfolge von Strukturlisten/Strukturen aufgestellt), sind eine Kombination von Objekt- und Bezugsoptionen, auf die man entlang des Pfades von der Wurzel des Baumes zu dieser Liste/Struktur trifft. Der spezifische Kombinationsmechanismus (z. B. NOR-Funktion setzen, Kreuzungsstelle setzen oder ein beliebiger anderer Konfliktauflösungsalgorithmus) ist eine Eigenschaft der Option selbst.

[0413] Es ist zu beachten, dass eine Konsequenz der oben erwähnten Regeln und die Tatsache, dass es keine Systemeinrichtung gibt, um Abarbeitungsoptionen auf eine Strukturdefinition in einer Strukturdatei zu setzen, ist, dass es keine direkte Regel zum Setzen von Optionen gibt, die sich auf alle Bezüge für eine Struktur anwenden lassen. Der Mechanismus, dies zu erreichen ist, eine Einzelstruktur-Strukturliste zu verwenden.

[0414] Der Tester legt einen bestimmten Satz von Strukturlisten-Abarbeitungsoptionen, die sein Zeichengruppenverhalten modifizieren und die seine Abarbeitungssequenz modifizieren, fest.

[0415] Wenn die Hardware einer Abarbeitungssequenz für eine Strukturliste unterzogen wird, erzeugt die Hardware einen Burst. Ein Burst ist die Abarbeitung einer Sequenz von Strukturen direkt durch die Hardware, ohne einen Eingriff von der Soft-Ware. Eine Burst-Unstetigkeit ist eine Position in einer Abarbeitungssequenz, in der ein vorausgehender Burst beendet ist und ein neuer Burst gestartet wird.

[0416] Eines der Entwurfsziele der Strukturverwaltungssoftware ist es, die Hardware mit den Abarbeitungssequenzen zu versehen, die sie benötigt, um daran einen Burst zu erzeugen. Durch Vorgabe liefert ein Strukturbaum eine Abarbeitungssequenz, die zu einem einzelnen Burst führen wird, wenn sie der Hardware ausgesetzt ist. Dieses Verhalten kann jedoch durch die Nutzung von Optionen an der Strukturliste modifiziert werden. So kann die Verwendung von Optionen zu Burstunstetigkeiten führen.

[0417] Außerdem werden Anwender manchmal eine Prolog- oder Epilogstruktur benötigen, die vor oder nach jeder Struktur oder jedem Burst abgearbeitet wird. Dies modifiziert die der Hardware auszusetzende Abarbeitungssequenz.

[0418] Während der Erzeugung oder Modifizierung der Abarbeitungssequenz des Strukturobjekts besitzt das System alle Informationen, die notwendig sind, um Unterbrechungen in Strukturbursts zu bestimmen und wenn nötig darüber zu berichten, die sich aus der Kombination von festgelegten Abarbeitungsoptionen und der durch den Strukturbaum verkörperten speziellen Abarbeitungssequenz ergeben. Während so vorgegangen wird, könnte es die Hardwarefähigkeiten der Module in dem System untersuchen müssen. Zum Beispiel ermöglicht eine Hardwareimplementierung vier gespeicherte Konfigurationen für Pin-Masken, von denen zwei (0 und 3) zur vorgegebenen maskierten Rechenoperation (um Mask This Vector, MTV, zu unterstützen) und unmaskierten Rechenoperation genutzt werden. Dem Anwender sind somit zwei unterschiedliche globale Pinmaskenkonfigurationen erlaubt, ohne den Burst-Modus zu unterbrechen.

[0419] Wenn ein Modulhersteller Strukturlistenimplementierungen in Hardware nicht unterstützt, ist zu beachten, dass die Verarbeitung der Strukturabarbeitungssequenz durch den Hersteller zu einer individuellen Abarbeitung aller Strukturen in der Abarbeitungssequenz führen würde. Sowohl in dem standortkompatiblen System als auch in dem standortheterogenen System würde die Burst-Fähigkeit von Standorten durch den „kleinsten gemeinsamen Nenner" begrenzt sein. Der Tester sorgt für eine bestimmte Vorgabemenge von Optionen, wobei ihre Parameter nachstehend beschrieben werden. Jede Option wird festgelegt, indem angegeben wird: Ob sie „unbezogen" (d. h. verknüpft mit einer Definition mit dem Schlüsselwort Global oder Local) oder „referenziell" (d. h. verknüpft mit einem Bezug mit dem Schlüsselwort Pat oder PList) ist. Eigenoptionen lassen sich am Definitionspunkt und an jedem Bezug anwenden, jedoch lassen sich Referenzoptionen nur an dem Bezug anwenden, mit dem sie verknüpft sind. Außerdem soll eine Option durch Töchter ererbt werden, wenn vorausgesetzt ist, dass sich die Option rekursiv auf alle statistisch (syntaktisch) oder dynamisch (semantisch, indem darauf Bezug genommen wird) verschachtelten Strukturen oder Strukturlisten anwenden lässt.

[0420] Nachstehend ist eine Liste von Optionen. Jeder taugliche Hersteller wird diese Optionen wie festgelegt interpretieren.

1. Maske <pin/pin group>

Unbezogen bei Anwendung auf GlobalPList, LocalPList.

Referenziell bei Anwendung auf PList, Pat. Vererbt durch Töchter.

Diese Strukturliste wird immer die Kreise der Pins vergleichen lassen, auf die durch die angegebene Pin oder deaktivierte Pin-Gruppe verwiesen wird. Manchmal können Einschränkungen der Hardware zu Burst-Diskontinuitäten führen.

2. BurstOff

Unbezogen bei Anwendung auf GlobalPList, LocalPList.

Referenziell bei Anwendung auf PList, Pat. Nicht durch Töchter vererbt.

Die Strukturliste wird immer in dem Non-Burst-Modus ausführen. Diese Option wird nicht durch Töchter vererbt, jedoch wird die Option BurstOffDeep (unten) durch Töchter vererbt.

3. BurstOffDeep

Unbezogen bei Anwendung auf GlobalPList, LocalPList.

Referenziell bei Anwendung auf PList, Pat. Durch Töchter vererbt.

Diese Strukturliste wird immer in dem Non-Burst-Modus ausführen. Diese Option wird durch Töchter vererbt, jedoch wird die Optian BurstOff (oben) nicht durch Töchter vererbt. Zu beachten ist, dass die Option BurstOff nicht durch eine Tochter abgeschaltet werden kann.

4. PreBurst <pattern>

Unbezogen bei Anwendung auf GlobalPList, LocalPList.

Nur durch Tochterknoten vererbt, die keine festgelegten Burst-Optionen besitzen. Die angegebene Struktur ist allen Bursts innerhalb dieser Strukturliste voranzustellen. Die Struktur PreBurst tritt direkt vor jedem Burst auf, der infolge dieses Strukturlistenknotens gestartet wird. Die Option wird nicht angewandt, wenn sie sich bereits innerhalb eines Bursts befindet, der eine Option PreBurst besitzt, die die gleiche Struktur ist.

5. PostBurst <pattern>

Unbezogen bei Anwendung auf GlobalPList, LocalPList.

Nur durch Tochterknoten vererbt, die keine festgelegten Burst-Optionen besitzen. Die angegebene Struktur ist allen Bursts innerhalb dieser Strukturliste anzufügen. Die Struktur PostBurst tritt direkt nach jedem Burst auf, der infolge dieses Strukturlistenknotens gestartet wird. Die Option wird nicht angewandt, wenn sie sich bereits innerhalb eines Bursts befindet, der eine Option PostBurst besitzt, die die gleiche Struktur ist.

6. PrePattern <pattern>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

Die angegebene Struktur ist allen Strukturen innerhalb dieser Strukturliste voranzustellen.

7. PostPattern <pattern>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

Die angegebene Struktur ist allen Strukturen innerhalb dieser Strukturliste anzufügen.

8. Alpg <alpg object name>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

das benannte ALPG Objekt gespeicherte relevante Informationen wie beispielsweise langsame APG Registereinstellungen, Lese-Operationszeit, Sofortdatenregister, Adressenverwürfelung, Datenumkehrung, Datengenerierer, usw.

9. StartPattern <pattern>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

Die Strukturliste wird Abarbeitung beim ersten
Auftreten von StartPattern in ihrer Abarbeitungsfolge
beginnen.

10. StopPattern <pattern>

Unbezogen bei Anwendung auf GlobaLPList, LocalPList

Nicht durch Töchter vererbt.

Die Strukturliste wird Abarbeitung beim ersten Auftreten von StopPattern in ihrer Abarbeitungsfolge beenden.

11. StartAddr <vector Offset or label>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

Diese muss durch eine Option StartPattern begleitet sein. Die Strukturliste wird Abarbeitung an StartAddr beim ersten Auftreten von StartPattern in ihrer Abarbeitungsfolge beginnen.

12. StopAddr <vector Offset or label>

Unbezogen bei Anwendung auf GlobalPList, LocalPList

Nicht durch Töchter vererbt.

Diese muss durch eine Option StopPattern begleitet sein. Die Strukturliste wird Abarbeitung an StartAddr

beim ersten Auftreten von StopPattern in ihrer Abarbeitungsfolge beenden.
13. EnableCompare_StartPattern <pattern>
Unbezogen bei Anwendung auf GlobalPList, LocalPList
Nicht durch Töchter vererbt.
Strukturvergleich wird beim ersten Auftreten der
angegebenen Struktur beginnen.
14. EnableCompare_StartAddr, EnableCompare_StartCycle
Unbezogen bei Anwendung auf GlobalPList, LocalPList
Nicht durch Töchter vererbt.
Diese muss mit EnableCompare_StartPattern begleitet
sein. Gibt Adresse oder Zyklus innerhalb der Struktur
an, in der Strukturvergleich zu starten ist.
15. EnableCompare_StopPattern <pattern>
Unbezogen bei Anwendung auf GlobalPList, LocalPList
Nicht durch Töchter vererbt.
Strukturvergleich wird beim ersten Auftreten der
angegebenen Struktur abschließen.
16. EnableCompare_StopAddr, EnableCompare_StopCycle
Unbezogen bei Anwendung auf GlobalPList, LocalPList
Nicht durch Töchter vererbt.
diese muss mit EnableCompare_StopPattern begleitet
sein. Gibt Adresse oder Zyklus innerhalb der Struktur
an, in der Strukturvergleich abzuschließen ist.
17. Überspringen
Referenziell bei Anwendung auf PList, Pat.
Nicht durch Töchter vererbt.
Bewirkt, dass eine durch eine Strukturliste dominierte
Struktur oder die gesamte Teilfolge übersprungen wird.
Dies wird außerdem das Überspringen aller Optionen an
der Wurzel dieses untergeordneten Strukturlistenbaums
bewirken. Es ist, als wäre dieser untergeordnete
Strukturbaum nicht für Abarbeitungszwecke vorhanden.

Fehlerbündelsteuerung von Strukturlisten

[0421] Wie vorher beschrieben, erzeugt die Hardware, wenn sie einer Abarbeitungsfolge für eine Strukturliste unterzogen wird, einen Burst einer Folge von Strukturen, ohne irgendeine Beteiligung von der Software. Eine Burst-Unstetigkeit ist eine Position in einer Abarbeitungsfolge, bei der ein vorheriger Burst beendet ist und ein neuer Burst gestartet wird. Die Optionen PreBurst, PostBurst, BurstOff und BurstOffDeep kontrollieren, wo die Burst-Unstetigkeiten auftreten, wie es in der Optionsliste oben beschrieben ist. Die Optionen PreBurst und PostBurst bestimmen Burst-Unstetigkeiten, die bestimmten zusätzlichen Regeln, die nachstehend beschrieben werden, abhängig sind:

1. Wenn eine Ausgangsliste die Optionen PreBurst und PostBurst aufweist und die verschachtelte Liste die gleichen entsprechenden Optionen besitzt, dann gibt es keine Burst-Unstetigkeit und die Optionen PreBurst und PostBurst der verschachtelten Liste lassen sich nicht anwenden. Es gibt nur einen einzelnen Burst, der PreBurst und PostBurst der Ausgangsliste anwendet.
2. Es ist zu beachten, dass wenn die verschachtelte Liste keine Burst-Optionen aufweist, es gleichbedeutend ist, als Ausgangsliste die gleichen Optionen PreBurst und PostBurst durch die Beschreibung dieser Optionen zu haben. Folglich führen verschachtelte Listen ohne Burst-Optionen nicht zu einer Burst-Unstetigkeit.
3. Wenn sich die oben erwähnte Regel 1 nicht anwenden lässt und es einen Beitrag zur Strukturabarbeitungsfolge vom Start der Ausgangsliste zum Start der verschachtelten Liste gibt, dann ist beim Start der verschachtelten Liste eine Burst-Unstetigkeit vorhanden. In diesem Fall lassen sich Pre-Burst und Post-Burst der Ausgangsliste auf diesen Beitrag zur Strukturabarbeitungsfolge von der Ausgangsliste anwenden. PreBurst und PostBurst der verschachtelten Liste lassen sich auf die verschachtelte Liste anwenden.
4. Wenn sich die oben erwähnte Regel 1 nicht anwenden lässt und es einen Beitrag zur Strukturabarbeitungsfolge vom Ende der verschachtelten Liste zum Ende der Ausgangsliste gibt, dann ist am Ende der verschachtelten Liste eine Burst-Diskontinuität vorhanden. In diesem Fall lassen sich PreBurst und PostBurst der Ausgangsliste auf diesen Beitrag zur Strukturabarbeitungsfolge von der Ausgangsliste anwenden. PreBurst und PostBurst der verschachtelten Liste lassen sich auf die verschachtelte Liste anwenden.

5. Wenn sich Regel 1 nicht anwenden lässt und es keinen Beitrag zur Strukturabarbeitungsfolge von der Ausgangsliste außer von der verschachtelten Liste gibt, dann lassen sich PreBurst und PostBurst der Ausgangsliste nicht anwenden. Es ist nur ein einzelner Burst vorhanden, der PreBurst und PostBurst der verschachtelten Liste anwendet.

[0422] Nachstehend sind einige Beispiele, die die Wirkung von Optionen der Abarbeitungsfolge veranschaulichen. Zur Vereinfachung wird vorausgesetzt, dass alle Strukturlisten in einer einzigen Datei festgelegt sind.

Beispiel 1: Verwendung von BurstOff

[0423] Dieses Beispiel stellt BurstOff und PreBurst dar. Von besonderem Gewicht ist, dass BurstOff Strukturen bewirkt, die allein in Bursts ablaufen, die eine Struktur lang sind. Daher lässt sich die Option PreBurst immer noch anwenden. Die eingegebenen Strukturlisten sind wie nachstehend:

```
Global A [BurstOff] [PreBurst pat_z]
{
    Pat    q;
    PList  B;
    Pat    r;
    Pat    s;

    Global C
```

```
    {
      Pat   t;
      PList D;
    };

    PList   D;
    PList   E;
};

Global B
{
  Pat a;
  Pat b;
};

Global D [BurstOff]
{
  Pat c;
  Pat d;
};

Global E
{
   Pat e;
};
```

[0424] Der bei A mit Wurzel versehene Baum kann in Fig. 8 dargestellt werden.

[0425] Die Abarbeitungsfolge für diese Struktur ist unten erwähnt. Das Zeichen | gibt eine Burst-Unterbrechung an. Diese Strukturliste arbeitet in 10 Bursts ab, wobei der erste mit Strukturen z und q und der letzte mit Struktur e ist:
zq|ab|zr|zs|t|c|d|c|d|e

[0426] Über diese Abarbeitungsfolge ist folgendes zu beachten:

1. Weil die Option BurstOff auf A durch B nicht vererbt wird, arbeiten die Strukturen a und b in B wie ein Burst.
2. Weil die Option PreBurst auf A durch B nicht vererbt wird, wird a und b in dem Burst durch B kein z vorgesetzt.
3. Der Namenszusatz durch z findet nur für Strukturen statt, die aufgrund dessen abgearbeitet werden, dass sie direkte Töchter nämlich von Strukturen q, r und s sind. Diese Strukturen werden einzeln wie in einem Burst abgearbeitet, der aufgrund dessen, dass A die Option BurstOff besitzt, nur eine Struktur lang ist. BurstOff erfordert es, Strukturen individuell in eine Struktur langen Bursts abzuarbeiten. Folglich lassen sich die Optionen PreBurst und PostBurst immer noch anwenden.
4. Strukturliste D besitzt eine unbezogene BurstOff-Option, die bewirkt, dass ihre Töchter c und d einzeln abgearbeitet werden. Sie vererben PreBurst z nicht von A.

Beispiel 2: Verwendung von BurstOffDeep

[0427] Dieses Beispiel veranschaulicht die Option BurstOffDeep. BurstOffDeep bewirkt während einer Strukturlistendefinition verschachtelte Definitionen und darauf bezogene Listen. Jedoch werden die Optionen PreBurst und PostBurst nicht durch verschachtelte und bezogene Listen vererbt. Das Beispiel nutzt die gleichen Strukturen A, B, C, D, E wie in Beipiel 1, wobei die Optionen jedoch unterschiedlich sind:

5. Optionen auf Definition von A: [BurstOffDeep], [PreBurst z], [PostBurst y]
6. Keine anderen Optionen auf irgendeinen anderen Knoten.

[0428] Die Abarbeitungsfolge ist wie nachstehend erwähnt. Wie vorher, gibt das Zeichen eine Burst-Unterbrechung an.
zqy|a|b|zry|zsy|t|c|d|c|d|e

[0429] Über diese Abarbeitungsfolge ist folgendes zu beachten:

1. PreBurst und PostBurst werden nicht durch B, C, D, E vererbt.
2. BurstOffDeep wird durch B, C, D und E vererbt.

Beispiel 3: Unterbindung von PreBurst und PostBurst

[0430] Angenommen, dass jetzt der Strukturlistenbaum von Beispiel 1 betrachtet wird, in dem die Optionen:

1. Optionen auf Definition von A: [PreBurst x] [PostBurst y]
2. Optionen auf Definition von C: [PreBurst x] [PostBurst z]
3. Keine weiteren Optionen auf einen beliebigen anderen Knoten

sind, wäre die Abarbeitungsfolge:
x g a b r s t c d c d e y

[0431] Die Gründe, weshalb die Teilfolge „t c d" nicht „x t c d z" ist, sind folgende:

1. Das erste x wird unterbunden, da es der Preburst-Option x entspricht, die eigentlich dem aktuellen Burst zugeordnet ist.
2. Das letzte z wird unterbunden, da PostBurst z nicht auf D vererbt wird und es keine Struktur gibt, die von C, an das z angefügt werden kann, generiert wird.

Beispiel 4: Verwendung von Überspringen

[0432] Dieses Beispiel veranschaulicht die Wirkung der Option Überspringen auf verschachtelte Definitionen und bezogene Listen. Das Beispiel verwendet die gleichen Strukturen A, B, C, D, E wie im Beispiel 1, jedoch sind die Optionen anders:

1. Optionen auf Definition von A: [Überspringen], [PreBurst z], [PostBurst y]
2. Optionen auf Bezug zu r: [Überspringen]
3. Optionen auf Definition von C: [Überspringen]

[0433] Die Abarbeitungsfolge ist ein einzelner Burst ohne Unterbrechungen wie unten:
z q a b s c d e y

[0434] Über diese Abarbeitungsfolge ist folgendes zu beachten:

1. Die Knoten für r und C werden übersprungen.
2. Es gibt überhaupt keine Burst-Unterbrechungen.

Beispiel 5: Maskenverwendung

[0435] Dieses Beispiel veranschaulicht die Wirkung der Maskenoption und ihre Auswirkungen auf Stukturdefinitionen und Strukturlistendefinitionen sowie Bezüge. Das Beispiel verwendet die gleichen Strukturen A, B, C, D, E wie in Beispiel 1, jedoch sind die Optionen anders:

1. Optionen auf Definition von A: [mask pin1_pin2], [PreBurst z]
2. Optionen auf Bezug von B: [mask pin3]
3. Optionen auf Definition von B: [mask pin4]
4. Optionen auf Bezug von e: [mask pin5]
5. Keine weiteren Optionen auf irgendwelche Knoten.

[0436] Der Name „pin1_pin2" legt eine Gruppe fest, die Pin1 und Pin2 maskiert. Die Namen „pin3", „pin4" und

„pin5" legen jeweils das Maskieren von Pin3, Pin4 und Pin5 fest. Die Abarbeitungsfolge ist nachstehend vorgesehen, wobei die Burst-Unterbrechung angibt. Die Zahlen unter jeder Struktur geben die Pins an, die während dieser Strukturabarbeitung maskiert werden müssen.

z	q	a	b	z	r	z	s	t	c	d	c	d	| e
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2
			3	3									5
			4	4									

[0437] Über diese Abarbeitungsfolge ist folgendes zu beachten:

1. Die Hersteller-Hardware kann nur 2 Maskenblöcke ohne eine Burst-Unterbrechung aufnehmen. Erst wenn e abgearbeitet ist, sind die zwei Maskenblöcke Pins {1, 2} und Pins {1, 2, 3, 4}. Wenn ein Muster e mit einem unterschiedlichen Maskenblock von Pins {1, 2, 5} erscheint, verlangt die Hardware eine Burst-Unterbrechung.

Beispiel 6: Verwendung von vererbten Optionen und Bezügen

[0438] Dieses Beispiel veranschaulicht, dass sich eine vererbte Option an einer Definition nicht anwenden lässt, wenn auf die Definition Bezug genommen ist. Wir betrachten das folgende Beispiel:

```
Gobal A
{
   Global B [BurstOffDeep]
   {
   Global C
   {
      ...
   };
   ...
   };
...

PList C;
   };

   Global D
   {
     PList C;
   };
```

[0439] Die Option BurstOffDeep wird durch C an ihrem Definitionspunkt vererbt. Sie ist jedoch keine unbezogene Option und wird somit nicht auf C an ihren beiden Bezugspunkten angewendet.

Beispiel 7: PreBurst und PostBurst mit verschachtelten Listen

[0440] Es wird das folgende Beispiel betrachtet:

```
GlobalPList A [PreBurst x] [PostBurst y]
{
   Pat p1;

   LocalPList B [PreBurst x] [PostBurst y]
   {
      Pat p2;
   }

         LocalPList C
         {
            Pat p3;
         }

         LocalPList D [PreBurst x] [Post-
Burst z]
         {
           Pat p4;
         }

         LocalPList E [PreBurst w] [Post-
Burst y]
         {
           Pat p5;
         }
           Pat p6
      }
```

[0441] Die Abarbeitungsfolge ist:
x p1 p2 p3 y | x p4 z | w p5 y | x p6 y

1. Struktur p2 ist der gleiche Burst wie p1, weil die Optionen PreBurst und PostBurst der verschachtelten Listen genauso festgelegt sind wie die mütterlichen. Struktur p3 befindet sich ebenfalls in dem gleichen Burst, weil diese Optionen genauso wie die mütterlichen vererbt werden. Diese Optionen weisen zumindest ein unterschiedliches Element in den verbleibenden verschachtelten Listen auf, das Burst-Unstetigkeiten hervorruft.

Zeitliche Steuerung

[0442] Der Anwender tritt mit dem System in erster Linie dadurch in Wechselwirkung, dass die Strukturdateien verwendenden Testanordnungen definiert werden. Die Zeitsteuerungsdatei wird genutzt, um die zeitliche Steuerung dieser Strukturen zu beschreiben. Diese Datei erfordert andere Systemdateien (z. B. Pin, SpecSelector), um zugrunde liegende Definitionen aufzulösen. Des Weiteren sind die Definitionen Spec-Selectors und Global, die zum Auflösen von verschiedenen in der Zeitsteuerungsdefinition genutzten Variablen verwendet werden, in ein Verbundobjekt Testbedingungsgruppe eingebettet. Dateien höherer Ebenen wie die Testplandatei nutzen wiederum dieses Beispiel Testbedingungsgruppe.

[0443] Die Testplandatei enthält Bezüge auf das Testbedingungsgruppenobjekt. Der Strukturquellenfile stellt

Bezüge zu den Komponenten von Wellenformselektor innerhalb eines Zeitsteuerungsverzeichnisobjekts her. Die Zeitsteuerungsobjekte selbst verweisen auf die Pinobjekte. Optional könnte sich das Zeitsteuerungsobjekt auch auf eine Variable beziehen, die durch ein Objekt SpecSelector moduliert ist. Diese Beziehungen sind in Fig. 9 dargestellt.

[0444] Das Strukturobjekt innerhalb der Strukturliste legt den Namen des Objekts Wellenformselektor zur Verwendung für eine Menge von Strukturzeichen fest. Zu beachten ist außerdem, dass die Datei Zeitsteuerungsverzeichnis in der Struktur festgelegt ist. Strukturen müssen nicht kompiliert werden, wenn dieses Verzeichnis nicht verändert wird.

```
Version 1.0;
Hauptmuster
{

  GemeinsamerAbschnitt
  {
  ...
  Zeitsteuerung = myGalxy.tim;
  TimingMap = myGalxyMap.tmap;
  ...
  Domänenvorgabe
  {
    NOP  V {SIG = 1; CLK = 1; DATA = L;} W
  {SIG = wfs1;
            FASTCLK = wfs1;}
    NOP  W {SIG = wfs2;}
      NOP  V {SIG = L;}
    NOP  V {SIG = 0;}

  }
 }
}
```

[0445] Die Objekte TestConditionGroupFile importieren das Zeitsteuerungsobjekt zur Verwendung und das Objekt TimingMap zur Verwendung. Jeder Test nutzt einen TimingCondition-Fall, der aus dem Objekt TestConditionGroup für diesen Fall abgeleitet ist. So können mehrere Zeitsteuerungsobjekte, die den gleichen Satz von Wellenformtabellen unterstützen, im Tester-System gespeichert und nach Bedarf ausgetauscht werden. Ebenso können mehrere Testplandateien ein gemeinsames Objekt TestConditionGroup teilen.

[0446] Ein Beispiel einer Testplan-Beschreibungsdatei stellt die Verwendung des unten erwähnten Zeitsteuerungsobjektes dar.

```
Import patlist1.plist;
Import tim1.tim;
Import tim2.tim;
Import tmap1.tmap;
TestConditionGroup tim1_prod
{
   SpecSet = prodTmgSpec(min, max,
typ)
   {
     Periodendauer = 10 ns, 15 ns,
12 ns;
   }
 zeitliche Steuerungen
   {
     Timing = tim1;
     TimingMap = tmap1;
   }
 }
 TestConditionGroup tim2_prod
 {
   SpecSet = prodTmgSpec(min, max,
typ)
   {
```

```
                Periodendauer = 10 ns, 15 ns,
12 ns;
              }
              zeitliche Steuerungen
              {
              Timing = tim2;
                 TimingMap  = tmap1;
              }
            }
            TestCondition tim1_prod_typ
            {
               TestConditionGroup = tim1_prod;
             Selector = typ;
            }
            TestCondition tim2_prod_max
            {
               TestConditionGroup = tim2_prod;
             Selector = max;
           }
           Test FunctionalTest MyFunctional-
TestSlow
           {
              PlistParam = patlist1;
              TestConditionParam =
tim1_prod_typ;
           }

           Test FunctionalTest MyFunctional-
TestFast
           {
               PlistParam = patList1;
               TestConditionParam =
tim2_prod_max;
           }
```

[0447] "tim1" und "tim2" sind zwei Tests in einem Testplan, die früher definierte unterschiedliche Zeitsteuerungsobjekte nutzen. Das Zeitsteuerungsobjekt definiert verschiedene Wellenformen auf Pro-Pin-Basis. Die in der Zeitsteuerungsdatei und der Zeitroutinenverzeichnisdatei verwendeten Pins müssen in der Pin-Definitionsdatei geeignet definiert sein.

[0448] Das Zeitsteuerungsobjekt kann SpecificationSet Objekte zum Definieren von Werten innerhalb der Wellenformobjekte nutzen. Obwohl das Zeitsteuerungsobjekt hartcodierte Werte für verschiedene Attribute umfassen kann, ist es normalerweise der Fall, dass Anwender verschiedenen Attributen Werte zuordnen las-

sen, indem Variable genutzt werden. Diese Variablen können wiederum von Objekten SpecificationSet abhängig sein. Ein Beispiel dieser Verwendung ist unten dargestellt.

```
                Version 1.0;
                Zeitsteuerung basic_functional
                {
                 ...
                Pin SIG
                {
                   Wellenformtabelle wfs1
                   {
                     {1{U@t_le; D@t_te D;
Z@45ns;}}
                   };
                 };

                 Pin CLK
                 {
                    Wellenformtabelle wfs1
    {
       {0{U@20ns; D@40ns; }};
    };
  };
}
```

[0449] Die Variable U@t_le, welche die Kantenlage definiert, wird anderswo definiert und ist von SpecificationSet abhängig. Der SpecSelector ist wie unten dargestellt definiert.

```
     Spezifizierungsset prodTmgSpec(min, max,
typ)
     {
        t_le = 10 ns, 14 ns, 12 ns;
        t_le = 30 ns, 34 ns, 32 ns;
        ...
     }
```

[0450] Die Änderung der zeitlichen Steuerung, die durch Änderung von spec genutzt wird, ist in dem Beispiel unten dargestellt.

```
Testbedingung prodTmp_typ
{
    Testbedingungsgruppe = prodTmgSpec;
 SpecSelector = typ;
 }

 Testbedingungsgruppe prodTmp_max
 {
     Testbedingungsgruppe = prodTmgSpec;
  SpecSelector = max;
 };
```

F2. Auflistung auf die Zeitsteuerungskomponenten eines Testers

[0451] Die zeitlichen Steuerungen „typ" und „max" verwenden die typische/maximale Spezifizierung in SpecSelector. Zwei Komponenten eines Testermoduls sind mit der Erzeugung von Wellenformen und ihren zugeordneten zeitlichen Steuerungen direkt verbunden. Die zwei Module sind der Patterngenerator (PG) und die Bildverarbeitungseinheit (FP). In Fig. 10 ist ein vereinfachtes Blockdiagramm dargestellt, das die Formatierung von Wellenformen und Erzeugung der Zeitsteuerung durch die Bildverarbeitungseinheit innerhalb des Testsystems offener Architektur veranschaulicht. Nachstehend wird eine kurze Beschreibung der Erzeugung von Wellenformen gegeben.

[0452] Der Patterngenerator **1002** erzeugt eine Zeitsteuerungsgröße, die für alle Pins in dem Modul gemeinsam ist. Die Zeitsteuerungsgröße wird die Globale Zeitsteuerungsgröße (GTS) genannt. Es gibt drei Modi, in denen der Patterngenerator aufgebaut werden kann. Diese drei Modi beeinflussen die Anzahl von Bits, die verwendet werden können, um die GTS zu beschreiben. Außerdem wirken sich diese Einstellungen auch auf die Anzahl der zum Auswählen einer Datenbank verwendeten Bits aus und darauf, ob die Bits „Erfasse diesen Vektor" (CTV) und „Blende diesen Vektor aus" (MTV) gesetzt sind oder nicht. Um den Tester anzuweisen, die Ergebnisse dieses Vektors zu erfassen, nutzt der Anwender das CTV Flag in der Strukturdatei. Ähnlich nutzt der Anwender das MTV Flag in der Struktur, um den Tester anzuweisen, die Ergebnisse des aktuellen Vektors auszublenden. Dies ist in der Tabelle 1 unten dargestellt.

[0453] Der Patterngenerator **1002** ist außerdem für die Erzeugung von wellenförmigen Zeichen (WFC) verantwortlich. WFC werden auf einer Pro-Pin-Basis erzeugt. Das Testermodul nutzt eine feststehende Anzahl von Bits zum Beschreiben der WFC.

GTS Bits	GTS in einer Datenbank	GTS Datenbank	CTV	MTV
8 Bits	256	4	NEIN	NEIN
7 Bits	128	8	JA	NEIN
6 Bits	64	16	JA	JA

Tabelle 1

[0454] Das Testermodul stellt die Bildverarbeitungseinheit **1004** pro Pin bereit. Jede Bildverarbeitungseinheit enthält einen Zeitsteuerungseinstellvermischer (TSS) **1006**, der in diesem Beispiel eine Gesamttiefe von bis zu **1024** besitzt. Der TSS **1006** kann in Abhängigkeit vom Modus des Patterngenerators in eine Anzahl von Datenbanken **1008** eingeteilt werden, wie es früher beschrieben und in Fig. 10 dargestellt ist, wo 16 Datenbanken von 64 Eingaben pro Datenbank verwendet werden. Der TTS ist vorgesehen, um bei der Fähigkeit, Wellenformtabellen für jeden Pin zu definieren, mehr Flexibilität zuzulassen. Im Modus „FP" gibt der TSS eine

2 Bits nutzende Zeitsteuerungseinstellung aus. Somit wird der TSS eine Gesamtmenge von vier charakteristischen physikalischen Zeitsteuerungseinstellungen pro Pin erzeugen. Diese Zeitsteuerungseinstellungen werden als lokale Zeitsteuerungseinstellungen (LTS) bezeichnet.

[0455] Die Bildverarbeitungseinheit **1004** kombiniert LTS und WFC und erzeugt einen Index **1010** in den Wellenformspeicher **1012** und Zeitsteuerungsspeicher **1014**. In dem Modus „FP" wird der 5-Bit Wert aufgeteilt mit 2 Bits, die durch die LTS erzeugt werden, und 3 Bits, die durch das WFC erzeugt werden. Somit ist die Tiefe des physikalischen Wellenformspeichers und Zeitsteuerungsspeichers 32 tief pro Pin, obwohl ein Maximum von 4 physikalischen Zeitsteuerungseinstellungen verwendet werden kann. Der Wellenformspeicher enthält die möglich gemachten Zeitsteuerungsflanken, die die Wellenformen bilden. Die Zeitsteuerungswerte für die möglich gemachten Flanken werden aus dem Zeitsteuerungsspeicher erhalten. Somit formatiert die Bildverarbeitungseinheit Wellenformen.

Abbildungsmethodik

[0456] Die Methodik besteht darin, alle Wellenform-Tabellenblöcke auf einer Pro-Pin-Basis zu LTS in dem Tester abzubilden. Wenn Tester-Hardware 4 lokale Zeitsteuerungseinstellungen LTS unterstützt, kann der Anwender ein Maximum von 4 Wellenform-Tabellenblöcken definieren. Jeder Wellenform-Tabellenblock kann ein Maximum von n Wellenform-Definitionen für das digitale Testermodul besitzen.

[0457] Die Zeitsteuerungsabbildungsdatei bewirkt eine Abbildung von in dem Zeitsteuerungsabbildungsblock definierten logischen Wellenformselektoren, auf die Wellenformtabelle für das Modul im Testsystem offener Architektur. In diesem Fall unterstützt der Tester bis zu 256 logische Wellenformselektoren. Im Testsystem offener Architektur bilden die logischen Wellenformselektoren direkt auf die GTS ab. Der Strukturkompilierer ist sowohl von dem Zeitsteuerungsabbildungsblock als auch dem Zeitsteuerungsblock abhängig, um die Strukturdateien kompilieren zu können. Wenn jedoch die Wellenformzeichen in den Wellenformtabellen des Zeitsteuerungsblocks unverändert sind oder die Abbildungen des Wellenformselektors in dem Zeitsteuerungsabbildungsblock unverändert sind, dann besteht keine Notwendigkeit, das Muster erneut zu kompilieren.

Ein diese Abbildungsmethodik nutzendes Beispiel

[0458] Um die Abbildung in ein digitales Testermodul darzustellen, werden folgende Annahmen gemacht: die Bildverarbeitungseinheit wird in den FP-Modus gesetzt sowie CTV- und MTV-Bits so gesetzt, dass die gesamte Anzahl von GTS-Bits 6 und die gesamte Anzahl von Zeitsteuerungs-Datenbank-Selektor-Bits 4 ist.

[0459] Jede im Zeitsteuerungsblock definierte Wellenformtabelle wird zu einer bestimmten LTS in der Zeitsteuerungsdatei abgebildet. Dies wird auf einer Pro-Pin-Basis vorgenommen. So wird Wellenformtabelle seq1 zu LTS1 abgebildet. Im Fall des „SIG-Pins" werden alle 8 möglichen Wellenformeingaben verbraucht. Jedoch erfordert der Pin „CLK" eine einzelne Wellenformeingabe und verbraucht somit eine einzelne Zeile in dem Wellenformspeicher (WFT) und dem Wellenform-Zeitsteuerungsspeicher (WTM).

[0460] Die Abbildung der ersten 2 physikalischen Wellenformen des Pins „SIG" ist in **Fig. 11** dargestellt. Wie diese Wellenformtabelle zwei Wellenformzeichen abbildet, die getrennte Konfigurationen der Flanken benötigen, schließen wir das Zuordnen zweier Eingaben in den Wellenformspeicher (WFT) **1112** und den Wellenform-Zeitsteuerungsspeicher (WTM) **1114** ab. Die Gestalt der Wellenform wird in dem WFM und die zeitliche Steuerung für Einzelheiten im WTM gespeichert. Eine Ausführung des Moduls weist eine Gesamtmenge von 6 Zeitsteuerungsflanken T1, T2, T3, T4, T5 und T6 auf. Diese bilden direkt auf die in den Wellenformen innerhalb eines Flankenressourcenabschnitts des Zeitsteuerungsblocks definierten Ereignisse E1, E2, ... ab. Wenn mehr als 6 Ereignisse in dem Zeitsteuerungsblock definiert sind und dieser mit dem oben erwähnten Modul genutzt wird, wird das zu einem Fehler führen. Im Beispiel von **Fig. 11** nutzt das erste Wellenformzeichen „0" Zeitsteuerungsflanke T1, um das Ereignis „Force Down" oder „D" zu programmieren, das zur Zeit 10 ns in dem Zyklus auftritt. Außerdem wird Zeitsteuerungsflanke T2 genutzt, um Ereignis „Force Down" oder „D" zur Zeit 30 ns zu generieren. Schließlich wird Zeitsteuerungsflanke T3 genutzt, um Ereignis „Force Off" oder „Z" zur Zeit 45 ns zu generieren.

[0461] Das zweite Wellenformzeichen „1" nutzt Zeitsteuerungsflanke T1, um das Ereignis „Force Up" oder „U" zu programmieren, das zur Zeit 10 ns in dem Zyklus auftritt. Außerdem wird Zeitsteuerungsflanke T2 genutzt, um ein Ereignis „Force Down" oder „D" zur Zeit 30 ns zu generieren. Schließlich wird Zeitsteuerungsflanke T3 genutzt, um ein Ereignis „Force Off" oder „Z" zur Zeit 45 ns zu generieren.

[0462] Auf diese Weise werden die WFC in den WFM-Speicher und den WTM-Speicher der Bildverarbeitungseinheit abgebildet. Die endgültige Anordnung des Wellenformspeichers WFM von LTS1 für Pin „SIG" ist unten in Tabelle 2 dargestellt.

Index	(WFC)	T1Set	T1ReSet	T2Set	T2ReSet	T2Drel	T2Dret	EXPH	EXPHZ	T3Set	T3ReSet	T3Drel	T3Dret	T4Drel	T4Dret	EXPL	[illegible]
0	0		1		1								1				
1	1	1			1								1				
2	d		1	1									1				
3	u	1			1								1				
4	L															1	
5	H							1									
6	m															1	
7	n							1									

Tabelle 2

[0463] Die endgültige Anordnung des Wellenform-Zeitsteuerungsspeichers WTM von LTS1 für Pin „SIG" ist unten in Tabelle 3 dargestellt.

Index	(WFC)	T1	T2	EXPH	T3	T4	EXPL

0	0	10ns	30ns		45ns		
1	1	10ns	30ns		45ns		
2	d	12ns	32ns		42ns		
3	u	12ns	32ns		42ns		
4	L						17ns
5	H			17ns			
6	m						15ns
7	n			15ns			

Tabelle 3

[0464] Der Pin „CLK" verbraucht eine einzelne Wellenform, und so sind WFM und WFT für diesen Pin sehr einfach. Die endgültige Anordnung des Wellenformspeichers WFM von LTS1 für den Pin „CLK" ist unten in Tabelle 4 dargestellt.

Index	(WFC)	T1Set	T1ReSet	T2Set	T2ReSet	T2Drel	T2Dret	EXPH	EXPHZ	T3Set	T3ReSet	T3Drel	T3Dret	T4Drel	T4Dret	EXPL	EXPHZ
0	1	1			1												
1																	
2																	
3																	
4																	
5																	
6																	
7																	

Tabelle 4

[0465] Die endgültige Anordnung des Wellenform-Zeitsteuerungsspeichers WTM von LTS2 ist unten in Tabelle 5 dargestellt.

Index	(WFC)	T1	T2	EXPH	T3	T4	EXPL
0	1	20ns	40ns				
1							
2							
3							
4							
5							
6							
7							

Tabelle 5

[0466] Der Block Zeitsteuerungsabbildung arbeitet explizit die Wellenformselektoren zu den Wellenformtabellen des Zeitsteuerungsblocks aus. Für ein Testersystem verdichtet sich dies auf das Vorbereiten des Speichers Zeitsteuerungseinstellvermischer (TSS). Der TSS enthält im Grunde eine Abbildung von der GTS auf die LTS, die die Einstellungen hält. Die TSS-Anordnung für unser Beispiel für Pin SIG wird so aussehen wie Tabelle 6 unten.

GTS	LTS
0 (wfs1)	1
1 (wfs2)	1
2 (wfs3)	2
3 (wfs4)	1
4 (wfs5)	3
5 (wfs6)	1
.	
N (wfs1)	1
.	
255	

Tabelle 6

[0467] Nachdem die Anordnungsabbildungen TSS und LTS aufgelöst sind, kann der Patternkompilierer diese Informationen schließlich nutzen, um die Struktur mit der korrekten Wellenformtabelle (LTS) und dem korrekten Wellenformzeichen zur Verwendung zu programmieren. So ist unsere, nur Pin „SIG" berücksichtigende, beispielhafte Pseudostruktur in Fig. 11 dargestellt. Zu beachten ist, dass diese Kompilierung keine Abhängigkeit vom Block Zeitsteuerung hat sondern nur vom Block Zeitsteuerungsabbildung abhängig ist.

G. Tester-Bedienung

[0468] Dieser Abschnitt beschreibt die prinzipielle Bedienung des Tester-Betriebssystems (TOS). Die in diesem Abschnitt betrachteten Aktivitäten sind:
Systeminitialisierung
Testplan laden
Struktur laden
Einen Testplan abarbeiten

Einen individuellen Test abarbeiten

Systeminitialisierung

[0469] Um das System in einer Ausführung zu initialisieren, müssen bestimmte Voraussetzungen erfüllt sein und bestimmte Bedingungen eingehalten werden. Der folgende Unterabschnitt führt diese auf.

Vorbedingungen

[0470] Kopien der relevanten Komponenten der Systemsoftware weisen einen zentralen Speicher auf, dessen Position der Systemsteuereinheit bekannt ist. Diese kann an der Systemsteuereinheit selbst oder auf einem anderen System mit netzmontiertem Verzeichnis sein (oder dem SYSC über einen anderen Mechanismus bekannt sein) und, mit welchem Mechanismus auch immer, muss die gesamte Software der Systemsteuereinheit zur Verwendung verfügbar gemacht werden, bevor das System funktionieren kann. Diese Software enthält:
Hersteller-Hardwaresteuerung (d. h. Modulsoftware)
von DLL,
Standard- oder Anwender-Testklassen DLL, und
Anwender-Testplan DLL.

[0471] Die Modulkonfigurationsdatei des Systems ist in der Systemsteuereinheit verfügbar. Abrufen, dass diese Datei es dem Nutzer erlaubt, die physikalische Konfiguration des Testers, z. B. der physikalische Platz und Typ jedes Moduls in dem Leiterplattenträger des Systems, sowie die Namen der DLL der Modulsoftware festzulegen.

[0472] Die Systemkonfigurationsdatei ist in der Systemsteuereinheit verfügbar. Abrufen, dass diese Datei die Liste von Site-Controller in dem System sowie eine Abbildung von Hostnamen des Site-Controllers auf Eingangsport-Adressen der Switchmatrix enthält.

[0473] Site-Controller besitzen einen Service, der den Standort-Konfigurationsmanager abarbeiten genannt wird. Dieser Service ist verantwortlich zur Bestimmung, welche Hardware durch einen "Feststellung von Hardware" bezeichneten Prozess in jedem Slot installiert ist. Er ist außerdem verantwortlich für die Teilnahme am Initialisierungsprozess des Systems mit der Systemsteuereinheit. Zu beachten ist, dass das Betriebsprotokoll der Switchmatrix in einer Ausführung vorschreibt, dass der SCM auf einem einzelnen Site-Controller mit Eingangsport-Verbindungsadresse 1 der Switchmatrix immer verwendet werden sollte, um die Switchmatrix-Verbindungen mit den Modulen zu konfigurieren. Abrufen, dass dieser „spezielle" Standort als SITEC-1 bezeichnet ist.

[0474] Die Systemsteuereinheit ist dafür verantwortlich, jeden SCM des Site-Controllers mit seiner Switchmatrix-Verbindungsadresse zu versehen.

[0475] Jeder SCM des Site-Controllers ist in der Lage, einen Prozess, Testplanserver (TPS) genannt, zu starten. Der Testplanserver auf jedem Site-Controller ist letzten Endes dafür verantwortlich, den Testplan des Anwenders (oder Testpläne in dem Fall, wo ein einzelner Site-Controller Tests an mehreren DUT abarbeitet) aufzunehmen und auszuführen.

Initialisierungsphase I: Systemvalidierung

[0476] Sobald die oben erwähnten Voraussetzungen und Vorbedingungen erfüllt worden sind, läuft die Systeminitialisierung zuerst mit einem Systemvalidierungsschritt wie folgt ab:

1. Die Systemsteuereinheit liest die System- und Modulkonfigurationsdateien, um die anwenderbestimmte Ansicht des Systems zu initialisieren.
2. Unter Verwendung der festgelegten Systemkonfigurationsinformationen weist die Systemsteuereinheit nach, dass die festgelegten Site-Controller im Gange, erreichbar und bereit sind (d. h lassen SCM laufen). Irgendein Fehler während dieses Bestätigungsschrittes wird bewirken, dass ein Systemfehler hervorgerufen und eine Initialisierung abzubrechen ist.
3. Die Systemsteuereinheit weist anschließend den SCM Dienst auf SITEC-1 an, die Switchmatrix zu konfigurieren, um zu allen Hardwaremodulen Zugriff zu haben und fordert ihn auf, eine Feststellung von Hardware durchzuführen.
4. Der SCM Service an dem SITEC-1 fragt alle verfügbaren Modulslots (bekannte Hardwareplätze) für {Her-

steller, Hardware} Tupel zyklisch ab und erzeugt eine Abbildung von {Hersteller, Hardware} Tupel auf Slots. Beim Abschluss hat diese Abfrage somit die gesamte Menge von {Hersteller, Hardware, Slot} Bindungen, die in dem kompletten System vorhanden sind, identifiziert. Die Ergebnisse dieser Abfrage werden an die Systemsteuereinheit gesendet.

5. Die Systemsteuereinheit bestätigt, dass die Ergebnisse des oben erwähnten Hardwarefeststellungsschrittes mit der anwenderspezifischen Konfiguration in der Modulkonfigurationsdatei übereinstimmen. Ein beliebiger Fehler während dieses Bestätigungsschrittes wird verursachen, dass ein Systemfehler hervorgerufen wird und eine Initilisierung abzubrechen ist.

6. Die Systemsteuereinheit lädt dann eine vorgegebene Umgebung (wie beispielsweise Suchpfade für Modul-DLL, Strukturlisten, Strukturen, Testplan-DLL, Testklassen-DLL, usw.) aus der (den) Umgebungseinstelldatei(en) an einem bekannten Platz (Plätzen).

7. Die Systemsteuereinheit gewährleistet, dass alle identifizierten Modulsoftware-DLLs vorhanden sind. Wenn eine in der Systemsteuereinheit nicht verfügbar ist, wird sie aus dem zentralen Speicher möglichst wieder gewonnen, sonst wird ein Systemfehler hervorgerufen und eine Initialisierung abgebrochen.

Initialisierungsphase II: Standortauslegung (optional)

[0477] Standortauslegung oder Standorteinteilung schließt die Zuordnung von Softwareebenen der verfügbaren Hardwaremodule des Systems zu unterschiedlichen Standorten (d. h., um mehrere DUT zu warten) ein. Abrufen, dass in einem Socket-File Standorteinteilungsinformationen bereitgestellt werden.

[0478] Das Testersystem ermöglicht es, Standorteinteilung (erneute (Einteilung) sowohl als Teil einer Testplanladung (da jeder Testplan mit einem speziellen Socket verknüpft ist) als auch als einen unabhängigen, von dem Anwender aufrufbaren Schritt durchzuführen. Im letzteren Fall leitet der Anwender die Standorteinteilung ein, indem ein Socket-File bereitgestellt wird, das ausschließlich zum Einteilen des Systems genutzt wird. Dies ist speziell während einer Systeminitialisierung im Falle von Mehrfachprüfung von DUT nutzbar, bei der jeder Standort einen unterschiedlichen DUT-Typ testet. Dieser Schritt ist jedoch während der Initialisierungsstufe optional, und der Anwender kann wählen, ihn nicht ausführen zu lassen, indem er sich stattdessen entscheidet, einer Testplanladung zu erlauben, das System geeignet einzuteilen.

[0479] Was auch immer die Mittel sind, die gewählt werden, um Standorteinteilung (durch einen unabhängigen Aufruf oder implizit durch eine Testplanladung) zu bewirken, der Mechanismus ist der gleiche. Dieser Mechanismus wird nachstehend beschrieben.

1. Den Socket vorausgesetzt, legt die Systemsteuereinheit zuerst fest, ob die jetzt vorhandene Systemeinteilung mit dem Socket kompatibel ist oder ob eine erneute Einteilung notwendig ist. Die vorgegebene Einteilung während einer Initialisierung ist eine, in der alle verfügbaren Module mit SITEC-1 verbunden sind. Die übrig bleibenden Schritte unten werden nur ausgeführt, wenn eine erneute Einteilung benötigt wird.

2. Die Systemsteuereinheit sendet an jeden Site-Controller SCM eine Konfigurationsmeldung, um sich mit der Anzahl und Identitäten von DUT Standorten, die dafür unter dem neuen Socket möglich gemacht werden, erneut zu konfigurieren. Zu beachten ist, dass dies ein allgemeines Verfahren ist und den Fall verarbeitet, bei dem die Anzahl von DUT-Standorten, die durch einen Site-Controller kontrolliert werden, Eins ist. Die neuen Socket-Informationen werden ebenfalls an die SCM übertragen.

3. Jeder SCM stoppt den laufenden TPS, falls überhaupt, und startet einen neuen, der ihn mit dem neuen Socket, und der Anzahl und den Identitäten von DUT-Standorten initialisiert, die für ihn unter dem neuen Socket möglich gemacht sind.

4. Die Systemsteuereinheit legt fest, welche Standorte welche Untermengen der erforderlichen Systemmodule benötigen. Während so vorgegangen wird, erarbeitet sie außerdem Hardware-Slotinformationen für die Standorte. Das Nettoergebnis ist für jeden Standort eine Liste von Slots im Vergleich zu diesem Standort zugeordneten Modul-DLLs. Die standortspezifische Liste wird als die Standortmodul-DLL-Slotliste (SITE-MDSL) bezeichnet werden.

5. Die Systemsteuereinheit stellt jedem SCM sowohl die geeignete SITE-MDSL als auch die notwendigen Modul-DLLs bereit. Jeder SCM macht diese Informationen dann wieder dem neu gestarteten TPS verfügbar.

6. Die Systemsteuereinheit fordert anschließend SITEC-1 auf, die Switchmatrix für die zweckmäßigen Site-zu-Slot-Verbindungen, das heißt für Standort-eingeteilten Betrieb zu konfigurieren.

7. Die TPSs an den Standorten 1 bis n laden die in ihren SITE-MDSL festgelegten DLLs. Jede dieser DLLs besitzt eine Initialisieren() genannte Funktion, die eine Matrix von Slot-Zahlen annimmt. Der TPS ruft auf Initialisieren() mit den passenden Slot-Listen für diesen Modultyp. Bei irgendwelchen Fehlfunktionen an diesem Punkt wird ein Systemfehler hervorgerufen und eine Initialisierung abgebrochen. Das Initialisierungs()-Verfahren macht folgendes:

a. Erzeugt konkrete Klassen basierend auf einem Standardschnittstellen-IXXX-Modul. Zum Beispiel wird eine mit einem digitalen Modul verknüpfte DLL ein einzelnes IPinModul-basiertes Objekt erzeugen, um jeden Slot zu bedienen, mit dem sie verknüpft ist.
b. Erzeugt konkrete Klassen basierend auf Schnittstelle IRessource, eine für jede „Ressourceneinheit" in dem Modul. Für ein digitales Modul wird jedes Objekt auf Basis von IPinModul wiederum Objekte auf Basis von ITesterPin für alle Pins in der Sammlung von Slots, die durch Digitalmodule eingenommen werden, erzeugen.
8. Die TPSs an Standorten 1 bis n rufen anschließend
getXXXModul() an jedem geladenen Modul DLL auf, um Modulinhaltsinformationen wiederzugewinnen.
9. Jeder Aufruf an getXXXModul() setzt ein Klassenobjekt <VendorHWType>Module zurück als ein IModul Zeiger (z. B. AdvantestPinModule). Jeder dieser IModul Zeiger wird durch den TPS im Cache abgespeichert, der diese für den Rahmen/Anwendercode verfügbar macht. Zu beachten ist, dass die Sammlung von IModulen, IRessourcen, usw. nachhaltig ist (zumindest für die Lebendsdauer des TPS).
10. Sobald die oben erwähnten Schritte beendet sind, startet der TPS, um seinen zugewiesenen (bekannten) Kanal anzuhören(). Dieser signalisiert der Systemsteuereinheit, dass der TPS „bereit" ist, Normalbetrieb (d. h. standorteingeteilt) zu beginnen.

Laden von Testplänen

[0480] Dieser Abschnitt beschreibt die Schritte, durch die eine TestPlan DLL des Anwenders in einen Site-Controller geladen wird (zum Prüfen von einzelnen oder mehreren DUT).

[0481] Sobald eine Systeminitialisierung (und optional anfängliche Standorteinteilung) beendet worden ist, können Testpläne des Anwenders geladen werden. Das Laden eines Anwender-Testplans in einen Site-Controller geht wie folgt vor sich:
1. Die Systemsteuereinheit lädt zuerst die Testplan-DLL in ihren eigenen Prozessraum, indem sie ihren zugeordneten Socket-File und ihren DUT-Typ-Identifizierer abfragt. Diese Informationen werden genutzt, um den Standort (die Standorte) zu bestimmen, auf denen dieser Testplan läuft, und daher für den (die) Site-Controller, dass dieser Testplan geladen werden würde.
2. Die Systemsteuereinheit verwendet anschließend die mit dem Testplan verknüpften Socket-Informationen, um den Wiedereinteilungsprozess, wie oben in groben Zügen dargestellt, einzuleiten.
3. Die Systemsteuereinheit zieht die Liste von durch den Testplan verwendeten Testklassen DLLs aus der Testplan DLL heraus und sendet, sobald die Systemsteuereinheit geprüft hat, dass der TPS bereit ist, Normalbetrieb zu beginnen (d. h. standorteingeteilt), die Testklassen DLLs und schließlich die Testplan DLL selbst an den entsprechenden TPS.
4. Der TPS ruft LoadLibrary() auf, um sie in seinen Prozessraum zu laden. Er ruft eine bekannte Funktion in der DLL auf, um so viele Testplanobjekte wie die Anzahl von Standorten (d. h. Prüfobjekte [DUT]) zu erzeugen, wie sie abarbeitet.
5. Der TPS initialisiert das (die) Testplanobjekt(e) mit den notwendigen Rahmenobjekten des Testers. Während einer Initialisierung lädt der TPS die geeigneten DLLs für die durch das (die) Testplanobjekt(e) verwendeten Testklassen in den Prozessraum und erzeugt die Testklassenfälle.
6. Der TPS baut den Übertragungskanal zu der/von der Systemsteuereinheit zu dem (den) Testplanobjekt(en) auf.
7. Die Systemsteuereinheit kommuniziert mit dem TPS und errichtet seine Proxy-Server für das Testplanobjekt (die Testplanobjekte).

[0482] Dies beendet das erfolgreiche Laden des Anwender-Testplans in einen Site-Controller.

Abarbeiten eines Testplans

[0483] Das Verfahren zum Ausführen aller Tests in einem Testplan entsprechend der vorgegebenen Ablauflogik ist wie folgt:
1. Die Anwendung des Benutzers überträgt die Mitteilung RunTestPlan zu dem TPS. Der TPS sendet die Mitteilung ExecutingTestPlan an alle geschalteten Anwendungen. Der TPS ruft anschließend Ausführen() im Testplan auf.
2. Das Prüfen mehrerer DUT mit einem einzelnen Site-Controller wird durchgeführt, indem mehrere Gruppen kleiner Programmbausteine auf diesem Site-Controller, einen pro DUT, verwendet werden. Jede Gruppe kleiner Programmbausteine arbeitet einen unterschiedlichen unabhängigen Fall des gleichen Testplanobjekts ab. Weil in diesem Fall die Modulsteuer-Software-DLLs über DUTs teilnehmen könnten, werden die Modulbefehle zur Hardwarekommunikation benötigt, um einen DUT-Identifiziererparameter anzunehmen.

3. Das Testplanobjekt iteriert über jeden Test in seiner Sammlung (teilt alternativ dazu seinem Ablaufobjekt mit, jeden Test gemäß der Ablauflogik zu bearbeiten), indem preExec(), execute() und postExec() aufgerufen wird.
4. Wenn jeder Test ausführt, werden Zustandsmeldungen zurück an alle angeschlossenen Anwendungen gesendet.

Ausführen eines einzelnen Tests

[0484] Ein Anwender kann wünschen, anstelle von allen Tests einen einzelnen Test in einem Testplan auszuführen. Für die Ausführung eines einzelnen Tests ist das Verfahren wie folgt.
1. Benutzeranwendung überträgt die Meldung Run-Test zu dem TPS; der TPS sendet die Meldung ExecutingTest an alle angeschlossenen Anwendungen. Der TPS ruft anschließend executeTest() im Testplan auf, womit festgelegt wird, den Test abzuarbeiten.
2. Das Testplanobjekt führt den festgelegten Test aus, indem preExec(), execute() und postExec() an diesem Testobjekt aufgerufen wird.
3. Wenn der Test ausgeführt wird, sendet er an alle angeschlossenen Anwendungen Zustandsmeldungen zurück.

[0485] Obwohl die Erfindung in Verbindung mit speziellen Ausführungen beschrieben worden ist, wird sich erschließen, dass vom Fachmann verschiedene Modifizierungen und Änderungen vorgenommen werden können. Deshalb ist die Erfindung nicht durch die vorhergehenden erläuternden Einzelheiten zu beschränken sondern vielmehr entsprechend dem Umfang der Patentansprüche zu interpretieren.

Patentansprüche

1. Verfahren zur Entwicklung eines Testprogramms mittels Universal-C/C++-Konstrukten, wobei das Testprogramm zum Testen eines integrierten Halbleiterschaltkreises, IC, in einem Halbleitertestsystem dient, wobei das Verfahren umfasst:
Beschreiben von Testsystemressourcen, Testsystemkonfiguration und Modulkonfiguration mittels Universal-C/C++-Konstrukten für die Entwicklung eines Testprogrammes zum Testen des IC auf dem Halbleitertestsystem, wobei das Beschreiben der Testsystemkonfiguration die Spezifizierung eines Site-Controllers (**104**) zum Kontrollieren wenigstens eines Testmoduls (**108**) umfasst und jedes Testmodul (**108**) herstellerbereitgestellte Hardware- und Software-Module zum Anwenden mindestens eines Tests auf den integrierten Halbleiterschaltkreis umfasst, wobei jedes vom Hersteller bereitgestellte Software-Modul (**606**) einen modulspezifischen Compiler zum Generieren von Testmusterobjekten umfasst, wobei der besagte Site-Controller (**104**) an einen Systemcontroller (**102**), welcher die Site-Controller-Aktivitäten wenigstens eines Site-Controllers (**104**) koordiniert, gekoppelt wird;
Beschreiben einer Testsequenz in Universal-C/C++-Konstrukten zur Entwicklung des Testprogramms zum Testen des IC auf dem Halbleitertestsystem;
Beschreiben eines Testplans in Universal-C/C++-Konstrukten zur Entwicklung des Testprogramms zum Testen des IC auf dem Halbleitertestsystem;
Beschreiben von Testbedingungen in Universal-C/C++-Konstrukten zur Entwicklung des Testprogramms zum Testen des IC auf dem Halbleitertestsystem;
Beschreiben von Testmustern in Universal-C/C++-Konstrukten zur Entwicklung des Testprogramms zum Testen des IC auf dem Halbleitertestsystem; und
Beschreiben einer zeitlichen Steuerung der Testmuster in Universal-C/C++-Konstrukten zur Entwicklung des Testprogramms zum Testen des IC auf dem Halbleitertestsystem.

2. Verfahren nach Anspruch 1, wobei das Beschreiben der Testsystemressourcen umfasst:
Spezifizieren eines Ressourcentyps, wobei der Ressourcentyp mit wenigstens einem Testmodul (**108**) zum Anwenden eines Tests auf den IC assoziiert ist;
Spezifizieren eines mit dem Ressourcentyp assoziierten Parametertyps, und
Spezifizieren eines Parameters des Parametertyps.

3. Verfahren nach Anspruch 1, wobei das Beschreiben der Testsystemkonfiguration zusätzlich umfasst:
Spezifizieren eines Eingangsports eines Modulverbindungs-Enablers (**106**),
wobei das Testsystem den Site-Controller (**104**) am Eingangsport an den Modulverbindungs-Enabler (**106**) koppelt und der Modulverbindungs-Enabler (**106**) den Site-Controller (**104**) an das mindestens eine Testmodul (**108**) koppelt.

4. Verfahren nach Anspruch 3, wobei der Modulverbindungs-Enabler (**106**) eine Switchmatrix ist.

5. Verfahren nach Anspruch 1, wobei das Beschreiben der Modulkonfiguration umfasst:
Spezifizieren eines Modulidentifizierers zum Spezifizieren eines Modultyps;
Spezifizieren von ausführbarem Code zum Steuern eines Testmoduls (**108**) des durch den Modulidentifizierer spezifizierten Modultyps, wobei das Testmodul (**108**) zum Anwenden eines Tests auf den IC dient; und
Spezifizieren eines mit dem Testmodul (**108**) verbundenen Ressourcentyps.

6. Verfahren nach Anspruch 5, wobei das Verfahren zusätzlich umfasst:
Beschreiben eines Slotidentifizierers zum Spezifizieren eines Ausgangsports eines Modulverbindungs-Enablers (**106**), wobei das Testsystem das Testmodul (**108**) über den Ausgangsport an den Modulverbindungs-Enabler (**106**) koppelt und der Modulverbindungs-Enabler (**106**) das Testmodul (**108**) an einen korrespondierenden Site-Controller (**104**) koppelt.

7. Verfahren nach Anspruch 6, wobei der Modulverbindungs-Enabler (**106**) eine Switchmatrix ist.

8. Verfahren nach Anspruch 5, wobei der ausführbare Code eine dynamisch verlinkte Bibliothek ist.

9. Verfahren nach Anspruch 5, zusätzlich umfassend das Spezifizieren eines Hersteller-Identifizierers zum Identifizieren des Bereitstellers des Testmoduls (**108**).

10. Verfahren nach Anspruch 5, zusätzlich umfassend das Spezifizieren eines Identifizierers, welcher die maximal verfügbare Anzahl an Ressourceneinheiten in Verbindung mit einem Ressourcentyp identifiziert.

11. Verfahren nach Anspruch 5, wobei der Ressourcentyp Digital-Prüfanschlüsse und die Ressourceneinheiten Prüfkanäle sind.

12. Verfahren nach Anspruch 5, wobei der Ressourcentyp Analog-Prüfanschlüsse und die Ressourceneinheiten Prüfkanäle sind.

13. Verfahren nach Anspruch 5, wobei der Ressourcentyp Radiofrequenz-Prüfanschlüsse und die Ressourceneinheiten Prüfkanäle sind.

14. Verfahren nach Anspruch 5, wobei der Ressourcentyp Stromversorgungsanschlüsse und die Ressourceneinheiten Prüfkanäle sind.

15. Verfahren nach Anspruch 5, wobei der Ressourcentyp Digitalisiereranschlüsse und die Ressourceneinheiten Prüfkanäle sind.

16. Verfahren nach Anspruch 5, wobei der Ressourcentyp beliebige Funktionsgeneratoren-Anschlüsse und die Ressourceneinheiten Prüfkanäle sind.

17. Verfahren nach Anspruch 5, wobei der Ressourcentyp mit Ressourceneinheiten assoziiert ist und weiterhin ein Indikator spezifiziert wird, welcher auf die arbeitsunfähigen Ressourceneinheiten Bezug nimmt.

18. Verfahren nach Anspruch 17, wobei als arbeitsunfähig indizierte Ressourceneinheiten fehlerhafte Ressourceneinheiten des Testmoduls (**108**) repräsentieren.

19. Verfahren nach Anspruch 1, wobei das Beschreiben der Testbedingungen umfasst: Spezifizieren mindestens einer Testbedingungsgruppe.

20. Verfahren nach Anspruch 19, wobei das Beschreiben der Testbedingungen zusätzlich umfasst:
Spezifizieren wenigstens eines Spezifizierungssets, welches mindestens eine Variable enthält; und
Spezifizieren eines Auswählers zum Auswählen eines Ausdrucks, welcher mit der Variablen verbunden wird.

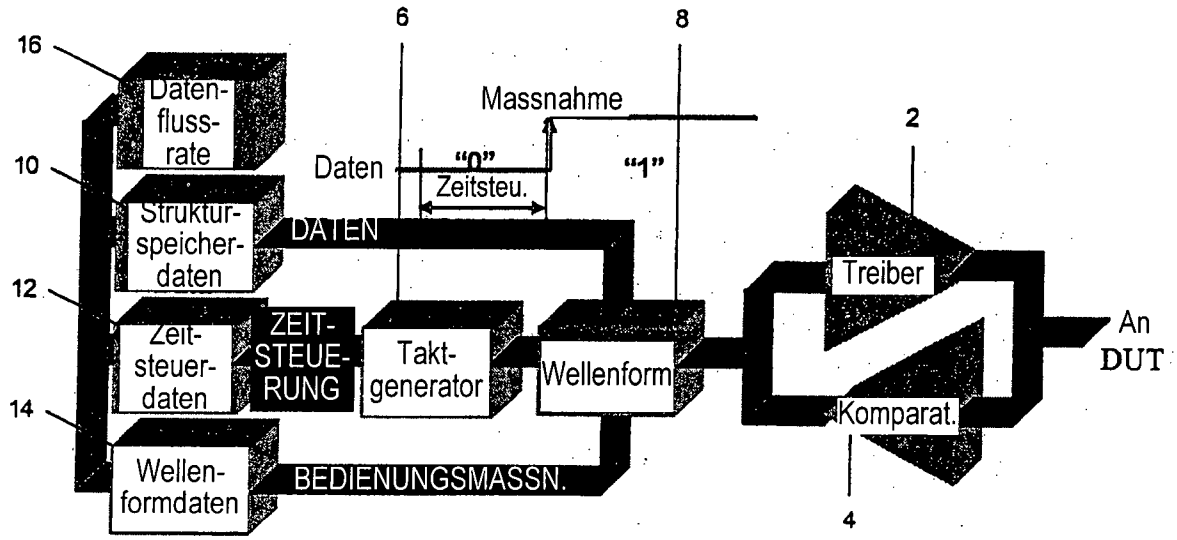
21. Verfahren nach Anspruch 20, wobei die Assoziierung der Testbedingungsgruppe mit einem Auswähler für das mindestens eine Spezifizierungsset eine Testbedingung definiert.

22. Verfahren nach Anspruch 21, wobei die Testbedingung ein Objekt ist.

23. Verfahren nach Anspruch 1, wobei das Beschreiben einer Testsequenz umfasst:
Spezifizieren eines Ergebnisses der Durchführung eines Flusses oder Tests;
Spezifizieren einer Handlung ausgehend von dem Ergebnis; und
Spezifizieren eines Übergangs zu einem anderen Fluss oder Test basierend auf dem Ergebnis.

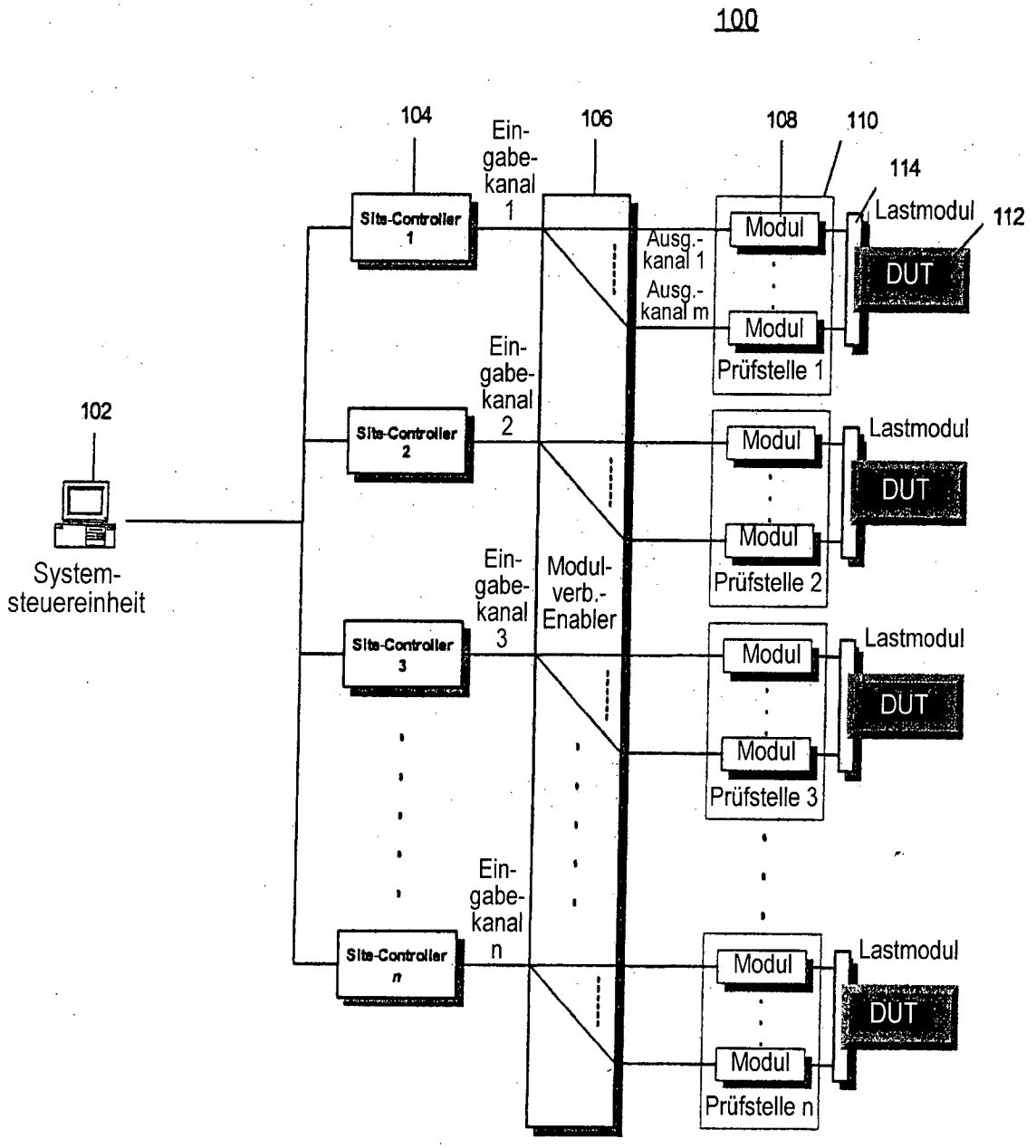
Es folgen 12 Blatt Zeichnungen

Anhängende Zeichnungen

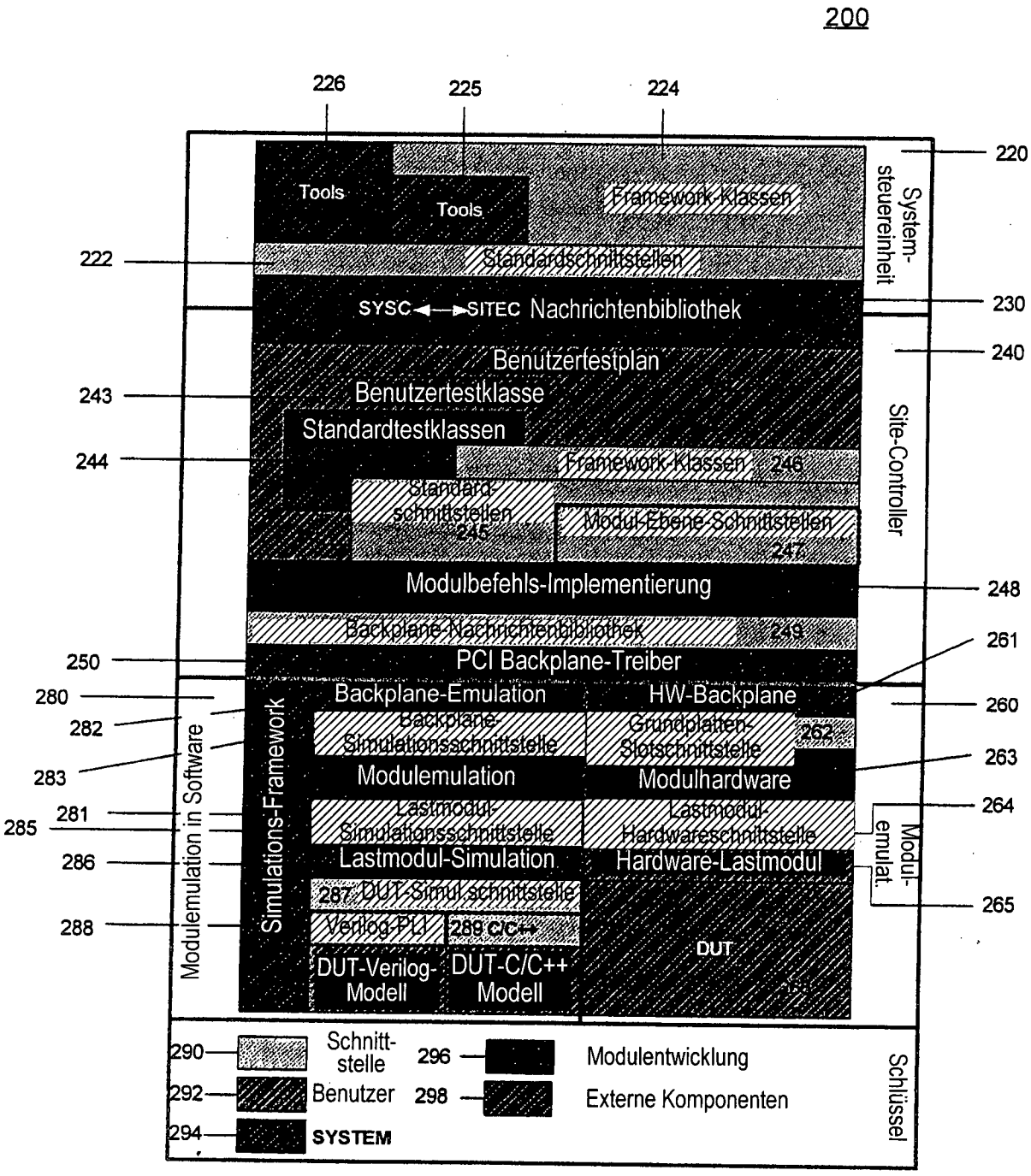


Bedienungsmassnahme = Strukturdaten * Zeitsteuerdaten (begrenzt durch Zeiteinstellung) * Wellensatz * Ansteuerung

Figur 1


100
104
106
108
110
114
112
Ein-
gabe-
kanal
1
Site-Controller
1
Ausg.-
kanal 1
Ausg.-
kanal m
Modul
Modul
Prüfstelle 1
Lastmodul
DUT
102
System-
steuereinheit
Ein-
gabe-
kanal
2
Site-Controller
2
Modul
Modul
Prüfstelle 2
Lastmodul
DUT
Ein-
gabe-
kanal
3
Modul-
verb.-
Enabler
Site-Controller
3
Modul
Modul
Prüfstelle 3
Lastmodul
DUT
Ein-
gabe-
kanal
n
Site-Controller
n
Modul
Modul
Prüfstelle n
Lastmodul
DUT

Figur 2


200
226
225
224
220
Tools
Tools
Framework-Klassen
System-steuereinheit
222
Standardschnittstellen
SYSC ◄—► SITEC Nachrichtenbibliothek
230
Benutzertestplan
240
243
Benutzertestklasse
Standardtestklassen
244
Framework-Klassen 245
Standard-schnittstellen 246
Modul-Ebene-Schnittstellen 247
Site-Controller
Modulbefehls-Implementierung
248
Backplane-Nachrichtenbibliothek 249
261
250
PCI Backplane-Treiber
280
Backplane-Emulation
HW-Backplane
260
282
Backplane-Simulationsschnittstelle
Grundplatten-Slotschnittstelle 262
283
Modulemulation
Modulhardware
263
281
Lastmodul-Simulationsschnittstelle
Lastmodul-Hardwareschnittstelle
264
285
Modul-emulat.
286
Lastmodul-Simulation
Hardware-Lastmodul
287 DUT-Simul.schnittstelle
265
288
Verilog-PLI
289 C/C++
DUT
DUT-Verilog-Modell
DUT-C/C++ Modell
Simulations-Framework
Modulemulation in Software
290
Schnitt-stelle
296
Modulentwicklung
292
Benutzer
298
Externe Komponenten
294
SYSTEM
Schlüssel

Figur 3

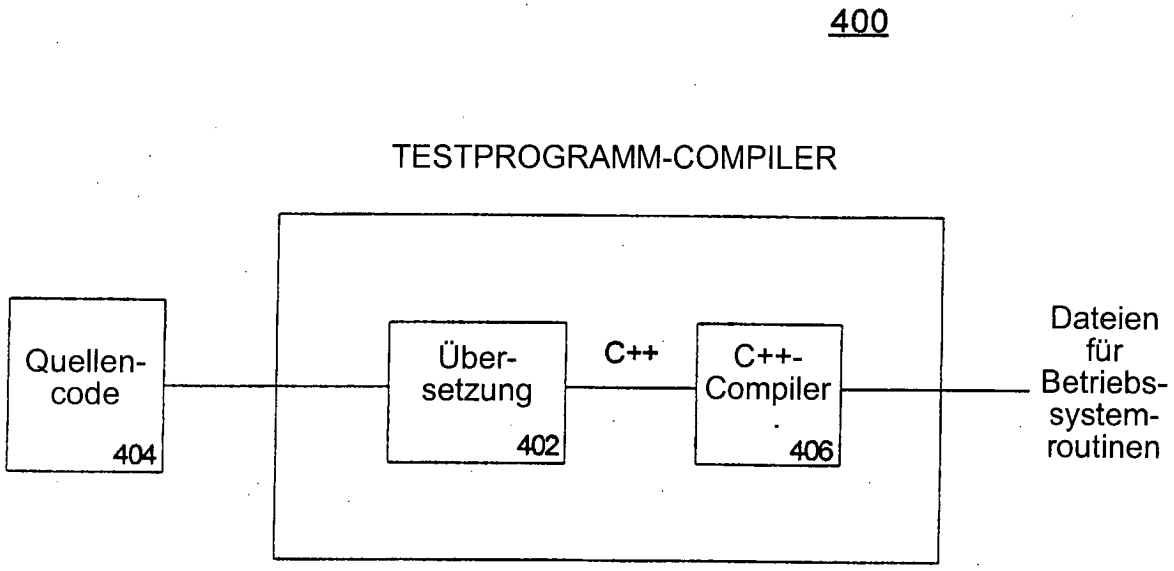
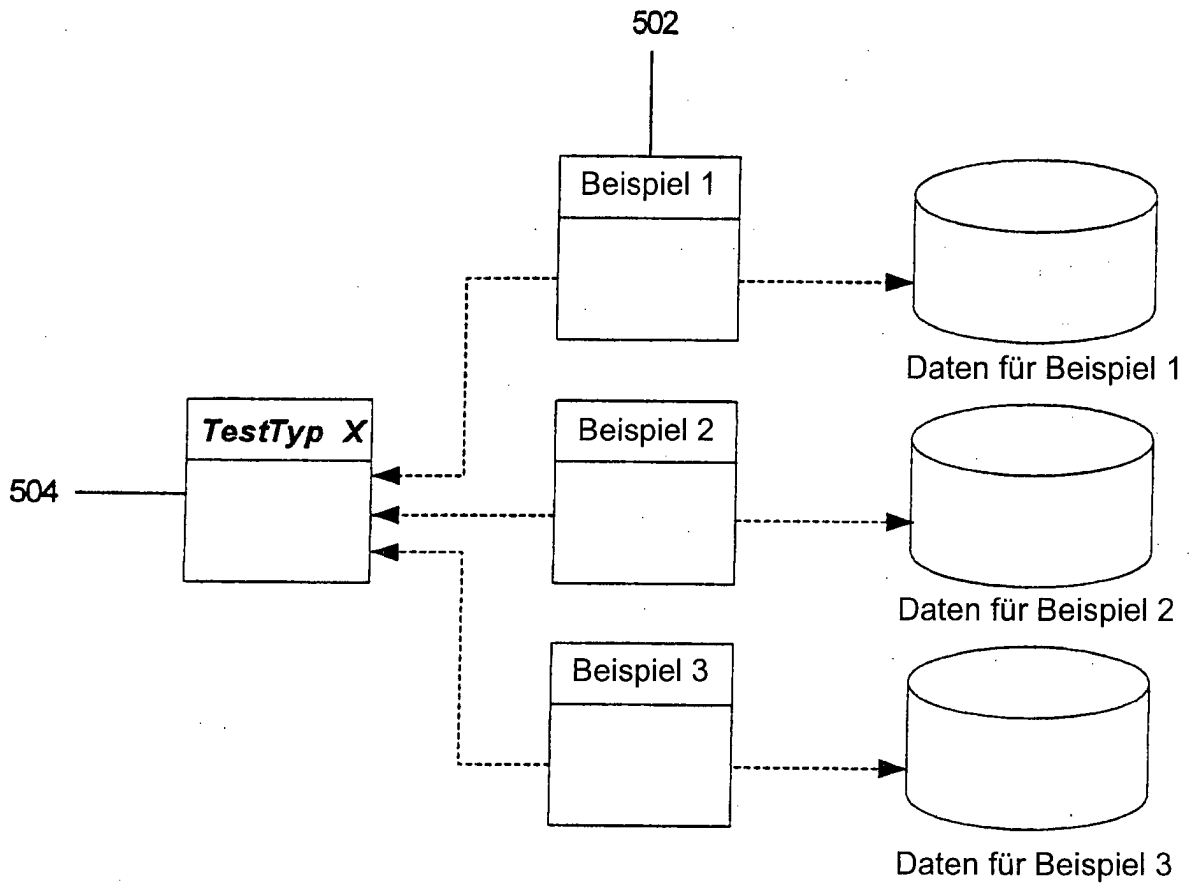
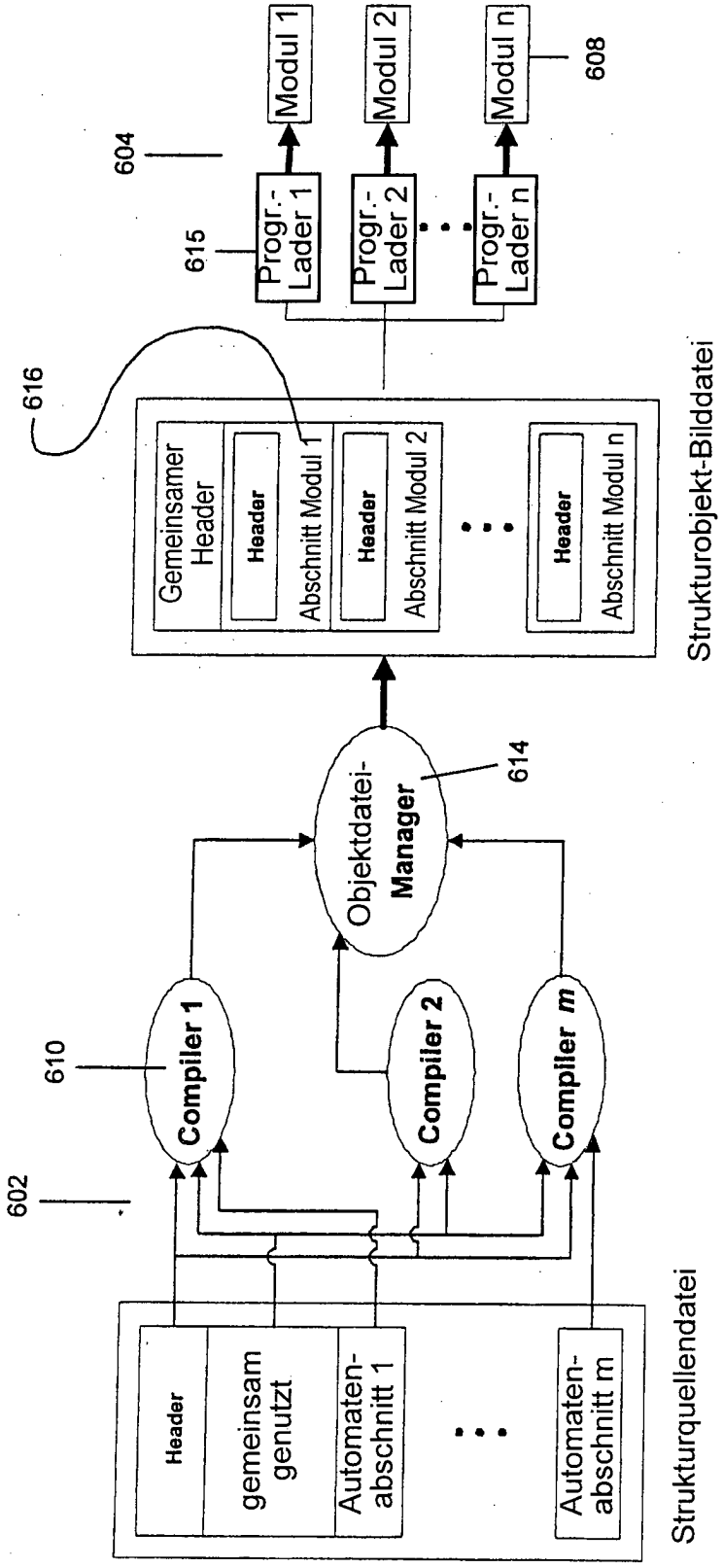

400
TESTPROGRAMM-COMPILER
Quellen-
code
404
Über-
setzung
402
C++
C++-
Compiler
406
Dateien
für
Betriebs-
system-
routinen

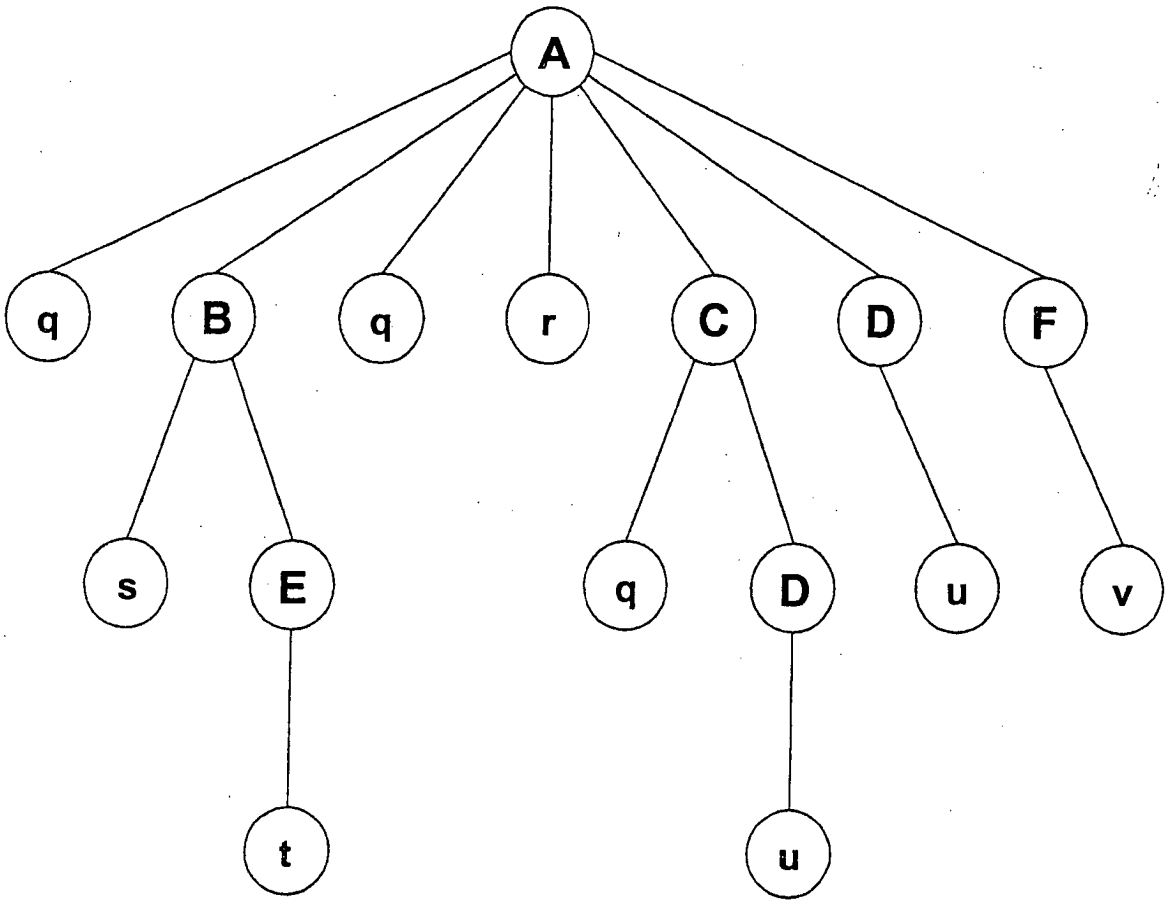
Figure 4


502
Beispiel 1
Daten für Beispiel 1
TestTyp X
504
Beispiel 2
Daten für Beispiel 2
Beispiel 3
Daten für Beispiel 3

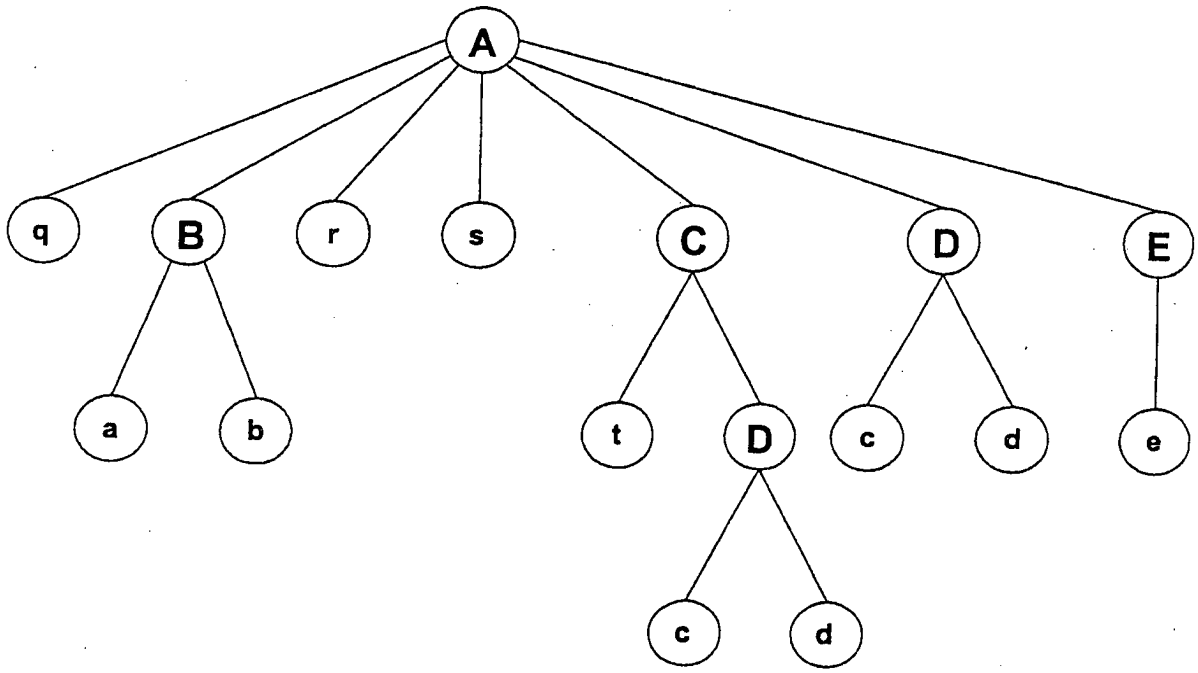
Figur 5


Modul 1
Modul 2
Modul n
608
604
615
Progr.-Lader 1
Progr.-Lader 2
Progr.-Lader n
616
Gemeinsamer Header
Header
Abschnitt Modul 1
Header
Abschnitt Modul 2
Header
Abschnitt Modul n
Strukturobjekt-Bilddatei
612
Objektdatei-Manager
614
610
Compiler 1
Compiler 2
Compiler m
602
Header
gemeinsam genutzt
Automaten-abschnitt 1
Automaten-abschnitt m
Strukturquellendatei
606

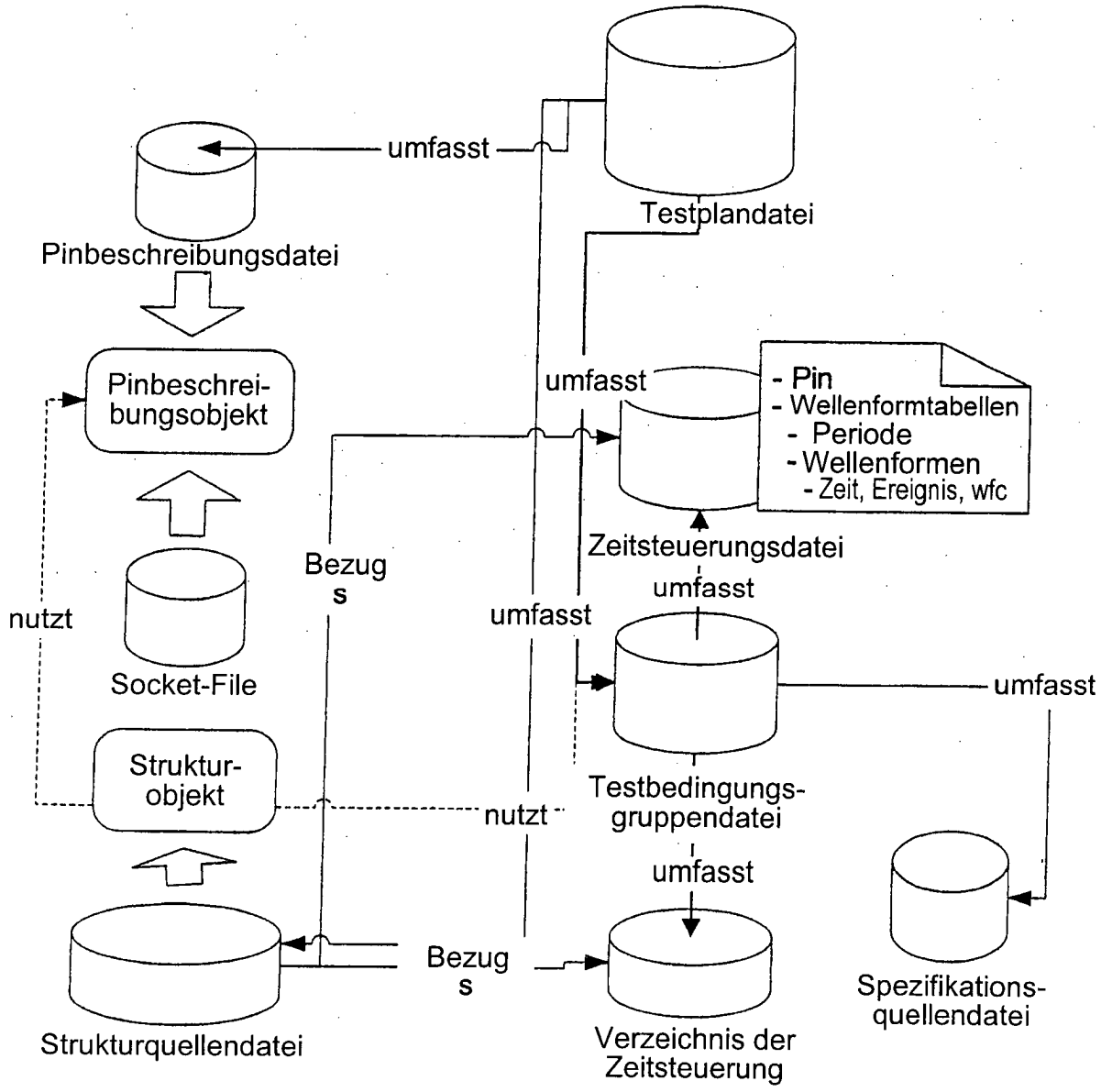
Figur 6



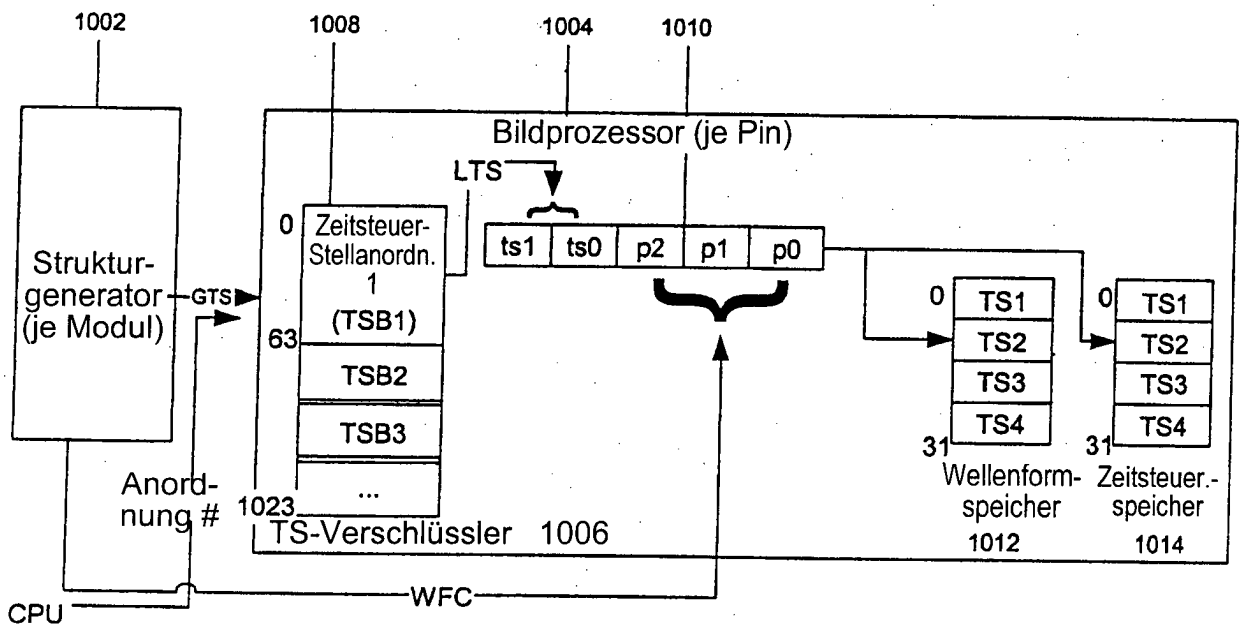
Figur 7



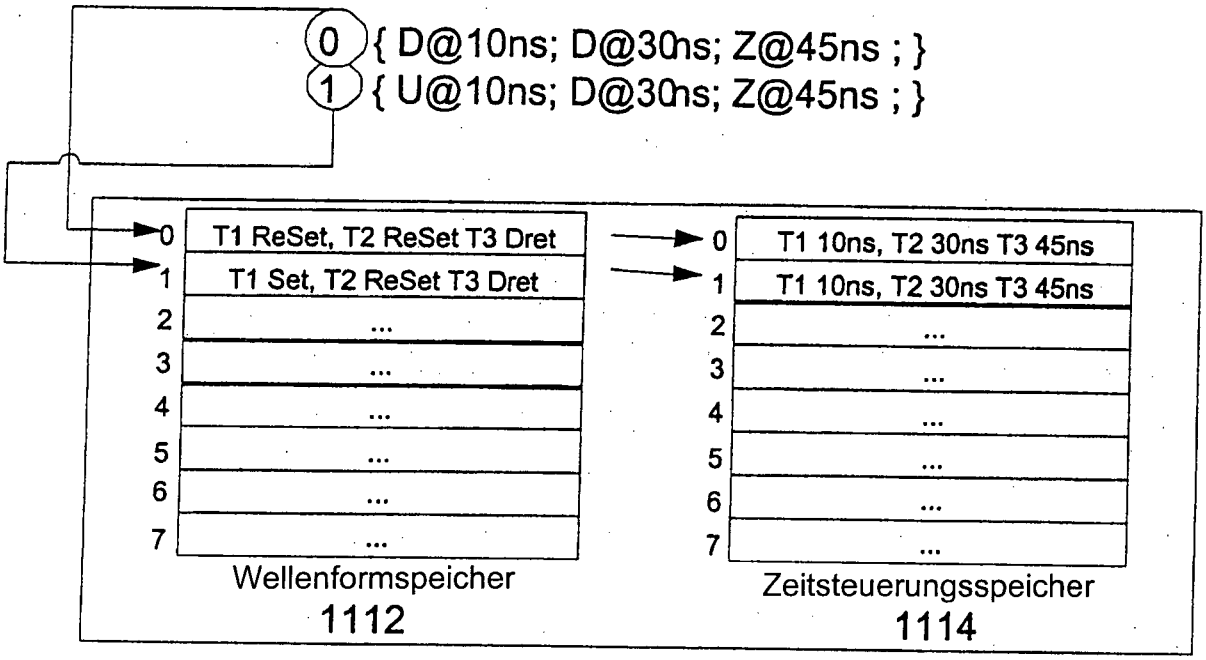
Figur 8


umfasst
Testplandatei
Pinbeschreibungsdatei
Pinbeschrei-
bungsobjekt
umfasst
- Pin
- Wellenformtabellen
- Periode
- Wellenformen
- Zeit, Ereignis, wfc
Zeitsteuerungsdatei
Bezug
s
umfasst
umfasst
nutzt
Socket-File
umfasst
Struktur-
objekt
nutzt
Testbedingungs-
gruppendatei
umfasst
Bezug
s
Spezifikations-
quellendatei
Strukturquellendatei
Verzeichnis der
Zeitsteuerung

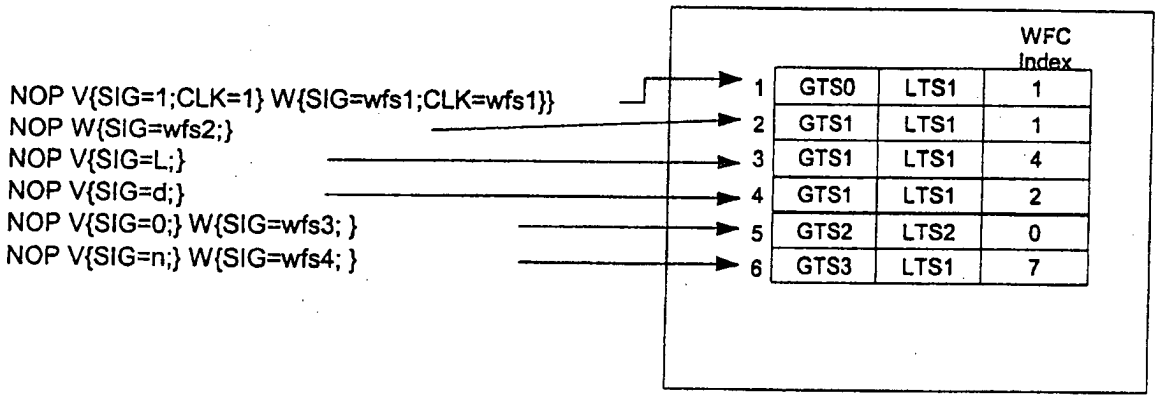
Figur 9


1002
1008
1004
1010
Bildprozessor (je Pin)
LTS
Struktur-
generator
(je Modul)
GTS
0
Zeitsteuer-
Stellanordn.
1
(TSB1)
63
TSB2
TSB3
...
1023
ts1
ts0
p2
p1
p0
0
TS1
TS2
TS3
TS4
31
Wellenform-
speicher
1012
0
TS1
TS2
TS3
TS4
31
Zeitsteuer.-
speicher
1014
Anord-
nung #
TS-Verschlüssler 1006
CPU
WFC

Figur 10


0 { D@10ns; D@30ns; Z@45ns ; }
1 { U@10ns; D@30ns; Z@45ns ; }
0 T1 ReSet, T2 ReSet T3 Dret
1 T1 Set, T2 ReSet T3 Dret
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...
Wellenformspeicher
1112
0 T1 10ns, T2 30ns T3 45ns
1 T1 10ns, T2 30ns T3 45ns
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...
Zeitsteuerungsspeicher
1114

Figur 11


NOP V{SIG=1;CLK=1} W{SIG=wfs1;CLK=wfs1}}
NOP W{SIG=wfs2;}
NOP V{SIG=L;}
NOP V{SIG=d;}
NOP V{SIG=0;} W{SIG=wfs3; }
NOP V{SIG=n;} W{SIG=wfs4; }
WFC Index
1 GTS0 LTS1 1
2 GTS1 LTS1 1
3 GTS1 LTS1 4
4 GTS1 LTS1 2
5 GTS2 LTS2 0
6 GTS3 LTS1 7

Figur 12