(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 10 2006 040 794 A1** 2007.05.31

(12) **Offenlegungsschrift**

(21) Aktenzeichen: **10 2006 040 794.6**
(22) Anmeldetag: **31.08.2006**
(43) Offenlegungstag: **31.05.2007**

(51) Int Cl.[8]: ***G06F 9/45*** (2006.01)

(30) Unionspriorität:
11/287,819 28.11.2005 US

(71) Anmelder:
Agilent Technologies, Inc. (n.d.Ges.d.Staates Delaware), Palo Alto, Calif., US

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

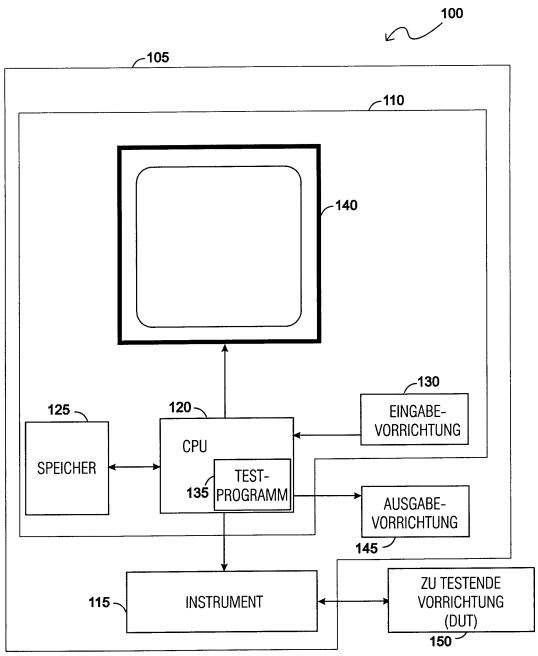
(72) Erfinder:
Davis, Alan Howard, Loveland, Col., US

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Softwareprogramm mit alternativen Funktionsbibliotheken**

(57) Zusammenfassung: Ein Softwareprogramm. Das Softwareprogramm umfasst einen Funktionsruf. Wenn das Programm in einem ersten Modus verknüpft ist, ist eine erste kompilierte Routine aus einer ersten Bibliothek mit dem Funktionsruf gekoppelt. Ansonsten ist eine zweite kompilierte Routine aus einer zweiten Bibliothek mit dem Funktionsruf gekoppelt. Die erste kompilierte Routine ist aus einer Quellcoderoutine kompiliert, wobei ein Parameter in der Quellcoderoutine auf einen ersten Wert eingestellt ist, und die zweite kompilierte Routine ist aus der Quellcoderoutine kompiliert, wobei der Parameter in der Quellcoderoutine auf einen zweiten Wert eingestellt ist. Die erste kompilierte Routine weist dieselbe Funktionalität auf wie die zweite kompilierte Routine plus eine zusätzliche Funktionalität.



Beschreibung

[0001] Um das Verhalten von modernen Produkten im Hinblick auf Übereinstimmung mit ihren Spezifikationen einzuschätzen, wurden Test und Messung ein wichtiger Teil von Produktentwicklungs- und Herstellungslebenszyklen. Testmaschinen, die diese Tests ausführen, können automatisiert sein, häufig unter Computersteuerung, um einen oder mehrere von verschiedenen Tests an einer Vielzahl von Systemen und/oder Komponenten auszuführen.

[0002] Ein Benutzer muss möglicherweise Softwareprogramme schreiben, nach Bedarf, zum Ausführen solcher Tests unter Verwendung dieser Testmaschinen. Um die Anstrengungen der Softwareprogrammierer zu erleichtern, stellt der Hersteller der Testmaschine häufig eine Bibliothek aus allgemeinen Funktionen bereit, die in dem Softwareprogramm verwendet werden können. Wie bei allen Softwareprogrammen muss das Programm selbst getestet und nach Bedarf modifiziert werden, um sicherzustellen, dass es gemäß Entwurf funktioniert. Das Finden und Korrigieren von Fehlern, die möglicherweise in dem Programm existieren, wird als das Korrigieren des Programms bezeichnet. Bei diesem Versuch wird das Programm häufig in einem Korrektur- bzw. Fehlerbeseitigungs-Modus kompiliert, wobei zusätzliche Programmierungsschritte zu dem Programm hinzugefügt werden, die Informationen zu dem Softwareprogrammierer liefern zur Verwendung beim Identifizieren des Typs und der Programmposition jeglicher Fehler. Dies kann ein zeitaufwändiges und kostspieliges Unternehmen sein und ist es auch häufig.

[0003] Sobald der Softwareprogrammierer zufrieden ist, dass das Programm korrekt funktioniert und bereit zur Verwendung ist, wird das Programm ohne die zusätzlichen Programmierschritte neu kompiliert, die zum Korrigieren verwendet werden, so dass die Geschwindigkeit, mit der das Programm seine Operationen ausführt, erhöht wird. Häufig prüft der Softwareprogrammierer nicht im Hinblick auf verschiedene potentielle Programmprobleme, wie z. B. Stapelspeicherüberlauf oder Fehler, die dem Softwareprogramm durch die Bibliotheksfunktionen berichtet werden. Somit, während der Programmierer möglicherweise glaubt, dass das Programm korrekt funktioniert, ist dies unter bestimmten Umständen möglicherweise nicht der Fall. Ferner kann ein Problem mit einer der Bibliotheksfunktionen nachfolgend durch den Hersteller erfasst und in einer späteren Softwareausgabe korrigiert werden. Die Verwendung dieser neuen Version einer Bibliotheksfunktion kann dann ermöglichen, dass ein Programmproblem an die Oberfläche tritt, das vorangehend durch den Softwareprogrammierer oder den Benutzer nicht erkannt wurde.

[0004] Es ist die Aufgabe der vorliegenden Erfindung, ein Softwareprogramm, ein Paar aus Bibliotheken und ein Verfahren zum Erzeugen eines Softwareprogramms mit verbesserten Charakteristika zu schaffen.

[0005] Diese Aufgabe wird durch ein Softwareprogramm gemäß Anspruch 1, ein Paar aus Bibliotheken gemäß Anspruch 7 und ein Verfahren gemäß Anspruch 13 gelöst.

[0006] Bei repräsentativen Ausführungsbeispielen wird ein Softwareprogramm offenbart, das einen Funktionsruf aufweist. Wenn das Programm in einen ersten Modus verknüpft ist, ist eine erste, kompilierte Routine aus einer ersten Bibliothek mit dem Funktionsruf gekoppelt. Ansonsten ist eine zweite, kompilierte Routine aus einer zweiten Bibliothek mit dem Funktionsruf gekoppelt. Die erste, kompilierte Routine ist aus einer Quellcoderoutine kompiliert, wobei ein Parameter bei der Quellcoderoutine auf einen ersten Wert eingestellt ist, und die zweite kompilierte Routine ist aus der Quellcoderoutine kompiliert, wobei der Parameter bei der Quellcoderoutine auf einen zweiten Wert eingestellt ist. Die erste kompilierte Routine weist die selbe Funktionalität auf wie die zweite kompilierte Routine plus eine zusätzliche Funktionalität.

[0007] Bei einem anderen darstellenden Ausführungsbeispiel ist ein Bibliothekenpaar offenbart, das eine erste Bibliothek und eine zweite Bibliothek aufweist. Die erste Bibliothek weist eine erste kompilierte Routine auf, und die zweite Bibliothek weist eine zweite kompilierte Routine auf. Die erste kompilierte Routine ist aus einer Quellcoderoutine kompiliert, wobei ein Parameter in der Quellcoderoutine auf einen ersten Wert eingestellt ist, und die zweite kompilierte Routine ist aus der Quellcoderoutine kompiliert, wobei der Parameter in der Quellcoderoutine auf einen zweiten Wert eingestellt ist. Die erste kompilierte Routine weist die selbe Funktionalität auf wie die zweite kompilierte Routine plus zusätzliche Funktionalität.

[0008] Und bei einem anderen darstellenden Ausführungsbeispiel wird ein Verfahren zum Erzeugen eines Softwareprogramms offenbart. Das Verfahren weist das Spezifizieren eines Modus auf, in dem das Programm verknüpft ist, wobei der Quellcode für das Programm einen Funktionsruf aufweist. Wenn ein erster Modus bei dem Schritt spezifiziert wird, der den Modus spezifiziert, wird eine ausführbare Form des Programms erzeugt durch Koppeln eines Objektcodes für eine erste kompilierte Routine aus einer ersten Bibliothek mit dem Funktionsruf in einer vorangehend kompilierten Form des Programms. Ansonsten wird die ausführbare Form des Programms erzeugt durch Koppeln des Objektcodes für eine zweite kompilierte Routine aus einer zweiten Bibliothek mit

dem Funktionsruf in der vorangehend kompilierten Form des Programms. Die erste kompilierte Routine hat die selbe Funktionalität wie die zweite kompilierte Routine plus zusätzliche Funktionalität.

[0009] Andere Aspekte und Vorteile der darstellenden Ausführungsbeispiele, die hierin präsentiert werden, werden aus der nachfolgenden detaillierten Beschreibung offensichtlich, in Verbindung mit den beiliegenden Zeichnungen.

[0010] Die beiliegenden Zeichnungen liefern visuelle Darstellungen, die verwendet werden, um verschiedene darstellende Ausführungsbeispiele umfassender zu beschreiben, und die durch Fachleute auf dem Gebiet verwendet werden können, um dieselben und ihre inhärenten Vorteile besser zu verstehen. In diesen Zeichnungen identifizieren gleiche Bezugszeichen entsprechende Elemente.

[0011] Bevorzugte Ausführungsbeispiele der vorliegenden Erfindung werden nachfolgend Bezug nehmend auf die beiliegenden Zeichnungen näher erläutert. Es zeigen:

[0012] Fig. 1 eine Zeichnung eines Blockdiagramms eines Testsystems, wie es in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist.

[0013] Fig. 2A eine Zeichnung eines Blockdiagramms von Komponenten und Prozessen, die beim Umwandeln von Programmquellcodekomponenten in das ausführbare Softwareprogramm verwendet werden, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist;

[0014] Fig. 2B eine Zeichnung eines Blockdiagramms von Komponenten und Prozessen, die durch das Softwareprogramm beim Aufrufen von Routinen aus einer von zwei Bibliotheken verwendet werden, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist;

[0015] Fig. 3 eine Auflistung einer Quellcoderoutine für die erste und die zweite kompilierte Routine, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist; und

[0016] Fig. 4 ein Flussdiagramm eines Verfahrens zum Verknüpfen und Betreiben eines Programms unter Verwendung alternativer Bibliotheken, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist.

[0017] Wie in den Zeichnungen zu Zwecken der Darstellung gezeigt ist, sind hierin neue Techniken für ein Softwareprogramm offenbart, das Rufe zu Funktionen in alternativen Bibliotheken aufweist, die sowohl in einem Fehlerbeseitigungsmodus als auch in einem Operationsmodus kompiliert sein können. Die im Fehlerbeseitigungsmodus kompilierten Funktionen sind in einer Fehlerbeseitigungsbibliothek bereitgestellt und die im Operationsmodus kompilierten Funktionen sind in einer separaten Bibliothek bereitgestellt. Das Softwareprogramm kann dann nach Bedarf verknüpft oder neu verknüpft werden, entweder mit den Fehlerbeseitigungsbibliotheksfunktionen oder den Operationsmodusbibliotheksfunktionen, je nach Bedarf, ohne ein Neukompilieren des Softwareprogramms. Erfolg oder Misserfolg der Rufe zu den Bibliotheksfunktionen kann überwacht werden, ohne Ändern und Neukompilieren des Softwareprogramms, das der Benutzer entwickelt hat. Frühere Techniken für eine solche Fehlerbeseitigung basierten auf dem Aktivieren und/oder Hinzufügen eines Fehlerbeseitigungscodes zu dem vom Benutzer entwickelten Softwareprogramm mit einer nachfolgenden Neukompilierung des Softwareprogramms und einer Verknüpfung.

[0018] In der nachfolgenden, detaillierten Beschreibung und in den verschiedenen Figuren der Zeichnungen sind gleiche Elemente durch gleiche Bezugszeichen identifiziert.

[0019] Fig. 1 ist eine Zeichnung eines Blockdiagramms eines Testsystems **100**, wie es in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist. In Fig. 1 weist das Testsystem **100** eine Testmaschine **105** auf, die mit einer zu testenden Vorrichtung (DUT; DUT = device under test) **150** verbunden ist. Die zu testende Vorrichtung (DUT) könnte jegliche von verschiedenen Typen von elektronischen Komponenten **150** oder jegliche eines anderen Typs einer Komponente **150** sein, wie z. B. eine mechanische Komponente **150**. Die Testmaschine **105** weist einen Computer **110** und ein Instrument **115** auf. Ebenfalls in Fig. 1 ist eine optionale, externe Ausgabevorrichtung **145** gezeigt, die extern mit dem Computer **110** verbunden ist. Die externe Ausgabevorrichtung **145** könnte z. B. ein Drucker **145** sein. Der Computer **110** ist verbunden mit und steuert die Operation des Instruments **115** durch Einstellen der Bedingungen, unter denen die Testmessung ausgeführt wird, durch Auslösen der Messung und durch Sammeln der Ergebnisse der Messung. Das Instrument **115**, das in Fig. 1 gezeigt ist, kann entweder ein einzelnes Instrument darstellen, wie z. B. ein Voltmeter, einen Strommesser oder ähnliches, oder kann eine Kombination aus individuellen Instrumenten darstellen, die zu einer Vielzahl von Messungen und Messungstypen in der Lage sind.

[0020] Der Computer weist eine zentrale Verarbeitungseinheit (CPU; central processing unit) **120**, einen Speicher **125**, eine Eingabevorrichtung **130** und eine interne Ausgabevorrichtung **140** auf. Die Eingabevorrichtung **130** könnte eine Tastatur **130** sein oder eine Zeigevorrichtung, wie z. B. ein Stift, eine Maus,

eine Steuerungseinrichtung oder ein Berührungsbildschirm, der geeignet ist für eine Cursor-Manipulation oder ähnliches. Die interne Ausgabevorrichtung **140** könnte ein Monitor **140** sein. Der Monitor **140** kann schwarz-weiß oder farbig sein. Der Computer **110** kann in der Lage sein, ein oder mehrere handelsüblich erhältliche Betriebssysteme zu betreiben, wie z. B. DOS, verschiedene Versionen von Microsoft Windows, (Windows 95, 98, Me, 2000, NT, XP, oder ähnliche), MAC OS X, UNIX, Linux von Apple oder andere geeignete Betriebssysteme. In Betrieb kann ein ausführbares Softwaretestprogramm **135**, das auch bezeichnet als ein Softwaretestprogramm **135** und als ein Testprogramm **135**, das im allgemeinen Fall hierin auch als ein Softwareprogramm **135** und als ein Programm **135** bezeichnet wird, in die CPU **120** aus dem Speicher **125** oder aus einer externen Quelle geladen werden, die in den Figuren nicht gezeigt ist.

[0021] Bei anderen Implementierungen kann das Testsystem **110** mehrere CPUs **120** aufweisen, wobei nicht alle derselben möglicherweise die selbe Testfunktion ausführen, und die an andere Eingabe-/Ausgabe-Vorrichtungen angebracht sein können, und die möglicherweise nicht an die verschiedenen Eingabe-/Ausgabe-Vorrichtungen angebracht sind, die in Fig. 1 gezeigt sind.

[0022] Fig. 2A ist eine Zeichnung eines Blockdiagramms aus Komponenten und Prozessen **200**, die beim Umwandeln von Programmquellcodekomponenten **205** in das ausführbare Softwareprogramm **135** verwendet werden, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist. In Fig. 2A kompiliert ein Kompilierer **215** Programmquellcodekomponenten **205** in Programmobjektcodemodule **220**. Die Programmquellcodekomponenten **205** umfassen Testerfunktionsrufe **210**, die hierin in dem allgemeinen Fall auch als Funktionsrufe **210** bezeichnet werden, die Rufe zu Funktionen sind, die standardmäßig bei der Testmaschine **105** sind und die sich in einer Bibliothek solcher Funktionen befinden. Die Bibliothek aus solchen Funktionen wird üblicherweise durch den Hersteller der Testmaschine **105** bereitgestellt, wohingegen die Programmquellcoderoutinen durch den Benutzer der Testmaschine **105** für den bestimmten Zweck des Benutzers erzeugt werden. Ein Verknüpfer **225** kombiniert oder verknüpft die Programmobjektcodemodule **220** mit einer Programmrohlings- bzw. Stub-Bibliothek **250**, die, wie in Fig. 2B gezeigt ist, verwendet werden kann, um einen Objektcode für Bibliotheksroutinen zu koppeln. Andere Techniken als die Verwendung der Stub-Bibliothek **250**, wie z. B. ein direktes Koppeln der kompilierten Testerroutinen **240a**, **240b** mit den Funktionsrufen **210** in dem Programm **135**, können beim Erfüllen der Funktionsrufe **210** verwendet werden.

[0023] Fig. 2B ist eine Zeichnung eines Blockdiagramms von Komponenten und Prozessen, die durch das Softwareprogramm **135** beim Aufrufen von Routinen aus einer von zwei Bibliotheken **235a**, **235b** verwendet werden, wie in verschiedenen repräsentativen Ausführungsbeispielen beschrieben ist. Bezug nehmend sowohl auf Fig. 2A als auch Fig. 2B erzeugt der Verknüpfer **225** das ausführbare Softwaretestprogramm **135** in einer von zwei Formen, die hierin als Modi bezeichnet werden. Wenn das ausführbare Softwaretestprogramm **135** in einen Fehlerbeseitigungsmodus verknüpft ist, der hierin im Allgemeinen auch als ein erster Modus bezeichnet wird, sind die Testerfunktionsrufe **210** mit ersten kompilierten Testerroutinen **240a** aus einer Fehlerbeseitigungsbibliothek **235a** gekoppelt. Erste kompilierte Testerroutinen **240a** werden hierin in dem allgemeinen Fall als erste kompilierte Routinen **240a** bezeichnet, und die Fehlerbeseitigungsbibliothek **235a** wird hierin in dem allgemeinen Fall als die erste Bibliothek **235a** bezeichnet. Ansonsten, wenn das ausführbare Softwaretestprogramm **135** in einem Operationsmodus verknüpft ist, der hierin im Allgemeinen auch als ein zweiter Modus bezeichnet wird, werden Testerfunktionsrufe **210** durch zweite kompilierte Testerroutinen **240b** aus einer Betriebs- bzw. Operations-Bibliothek **235b** ersetzt. Zweite kompilierte Testerroutinen **240b** werden hierin in dem allgemeinen Fall als zweite kompilierte Routinen **240b** bezeichnet, und die Operationsbibliothek **235b** wird hierin im Allgemeinen als die zweite Bibliothek **235b** bezeichnet. Somit sind abhängig von dem ausgewählten Modus, wenn das Softwaretestprogramm **135** verknüpft wird, kompilierte Testerroutinen **240** aus einer von zwei Bibliotheken **235a**, **235b** mit dem Softwaretestprogramm **135** gekoppelt.

[0024] In der Praxis wird ein Ruf zu dem Testerfunktionsruf **210** über eine Stub-Bibliothek **250** erfüllt, die üblicherweise in einem Betriebssystem **260** des Computers **110** angeordnete ist und die einen ersten Stub (Programmrohling) **251** verwendet, um die Adresse der ersten kompilierten Testerroutine **240a** in der ersten Bibliothek **235a** aus einer Adresstabelle **255** zu erhalten, oder verwendet einen zweiten Stub **252**, um die Adresse der zweiten kompilierten Testerroutine **240b** in der zweiten Bibliothek **235b** aus der Adresstabelle **255** zu erhalten, abhängig von dem Modus, der zur Verknüpfungszeit durch den Benutzer ausgewählt wird. Die erste kompilierte Testerroutine **240a** weist eine Funktionalität zusätzlich zu der Funktionalität der zweiten kompilierten Testerroutine **240b** auf. Bei darstellenden Ausführungsbeispielen liefert die zusätzliche Funktonalität der ersten kompilierten Testerroutine **240a** Informationen für eine Fehlerbeseitigung bei dem Softwaretestprogramm **135**. Die erste und zweite kompilierte Routine **240a**, **240b** tätigen Rufe zu internen Routinen **265**, die in dem Betriebssystem **260** angeordnet sind. Hierin offenbarte Techniken sind an Softwareprogramme **135** mit meh-

reren Funktionsrufen **210** verschiedener Typen anwendbar. Andere Techniken, wie z. B. das Verknüpfen von einer der kompilierten Testerroutinen **240a**, **240b** mit den Programmobjektcodemodulen **220** zur Verknüpfungszeit, könnten ebenfalls verwendet werden anstelle der Verwendung der Stub-Bibliothek **250**. Andere Techniken als die Verwendung der Stub-Bibliothek **250** und der Adresstabelle **255** können beim Erfüllen der Funktionsrufe **210** verwendet werden. Die Stub-Bibliothek **250** jedoch mit einem Funktionsadressennachschlagen in der Adresstabelle **255** ermöglicht Rufe zu Routinen unabhängig von der Revision bzw. Korrektur des verwendeten Betriebssystems.

[0025] Fig. 3 ist eine Auflistung einer Quellcoderoutine **300** für die erste und die zweite kompilierte Routine **240a**, **240b**, wie in verschiedenen repräsentativen Ausführungsbeispielen beschrieben ist. Die zweite kompilierte Testerroutine **240b** ist aus der Quellcoderoutine **300** mit einem Parameter **310** kompiliert, der bei diesem darstellenden Beispiel „DEBUG" (Fehlerbeseitigung) in der Quellcoderoutine **300** ist, eingestellt auf einen zweiten Wert, der bei diesem darstellenden Beispiel NULL ist. Die erste kompilierte Testerroutine **240a** ist aus der Quellcoderoutine **300** kompiliert, wobei der Parameter **310** bei der Quellcoderoutine **300** auf einen ersten Wert eingestellt ist, der bei der ersten kompilierten Testerroutine **240a** resultiert, die eine Funktionalität zusätzlich zur Funktionalität der zweiten kompilierten Testerroutine **240b** aufweist. Wiederum ist bei diesem darstellenden Beispiel der Parameter **310** „DEBUG", aber für eine Kompilierung der ersten kompilierten Testerroutine **240a** weist der Wert von DEBUG einen Wert ungleich NULL auf.

[0026] Die Funktionalität bei der ersten kompilierten Testerroutine **240a** zusätzlich zu der, die sich in der zweiten kompilierten Testerroutine **240b** befindet, umfasst eine Prüfung des Stapels bei der Initiierung der Routine über den Ruf zu „SCheckStack()" und eine Prüfung des Rücksendewerts der Routine „PETable()" innerhalb der ersten kompilierten Testerroutine **240a** über den Ruf zu „SCeckAtm()". Im Gegensatz dazu umfasst die Funktionalität bei der zweiten kompilierten Testerroutine **240b** die Prüfung des Stapels bei der Initiierung der Routine über den Ruf „SCheckStack()" nicht, da „CHECK_STACK" auf NULL kompiliert und nicht auf „SCheckStack()", wie für die erste kompilierte Testerroutine **240a**. Ferner umfasst die Funktionalität bei der zweiten kompilierten Testerroutine **240b** keine Prüfung des Rücksendewerts der Routine „PETable()", die innerhalb der ersten kompilierten Testerroutine **240a** ist über den Ruf zu „SCheckAtm()", da „CHECK_ATM" zu NULL kompiliert und nicht zu „SCeckAtm()", wie für die erste kompilierte Testerroutine **240a**.

[0027] Fig. 4 ist ein Flussdiagramm eines Verfahrens **400** zum Verknüpfen und Betreiben eines Programms **135** unter Verwendung alternativer Bibliotheken **235a**, **235b**, wie in verschiedenen darstellenden Ausführungsbeispielen beschrieben ist. Bei Block **403**, wenn das Programm **135** verknüpft oder neu verknüpft werden soll, überträgt Block **403** die Steuerung zu Block **405**. Ansonsten überträgt Block **403** die Steuerung zu Block **425**.

[0028] Bei Block **405** wird der Modus zum Verknüpfen von Programmquellcodekomponenten **205** entweder mit der Fehlerbeseitigungsbibliothek (der ersten Bibliothek) **235a** oder der Operationsbibliothek (der zweiten Bibliothek) durch den Benutzer eingestellt. Block **405** überträgt dann die Steuerung zu Block **410**.

[0029] Wenn der Fehlerbeseitigungsmodus bei Block **405** ausgewählt wurde, überträgt Block **410** die Steuerung zu Block **415**. Ansonsten überträgt Block **405** die Steuerung zu Block **420**.

[0030] Bei Block **415** wird das Testprogramm **135** mit den entsprechenden ersten kompilierten Testerroutinen **240a** in der Fehlerbeseitigungsbibliothek **235a** verknüpft, um das ausführbare Softwaretestprogramm **135** zu erzeugen. Block **415** überträgt dann die Steuerung zu Block **425**.

[0031] Bei Block **420** wird das Testprogramm **135** mit den entsprechenden zweiten kompilierten Testerroutinen **240b** in der Operationsbibliothek **235b** verknüpft, um das ausführbare Softwaretestprogramm **135** zu erzeugen. Block **420** überträgt dann die Steuerung zu Block **425**.

[0032] Bei Block **425** wird das ausführbare Softwaretestprogramm **135**, das entweder bei Block **415** oder bei Block **420** erzeugt wird, in die CPU **120** des Computers **110** der Testmaschine **105** geladen. Block **425** überträgt dann die Steuerung zu Block **430**.

[0033] Bei Block **430** wird die zu testende Vorrichtung **150** mit der Testmaschine **105** verbunden. Block **430** überträgt dann die Steuerung zu Block **435**.

[0034] Bei Block **435** wird das ausführbare Softwaretestprogramm **135** in der CPU **120** des Computers **110** der Testmaschine **105** aktiviert, um den Test auf der zu testenden Vorrichtung **150** zu betreiben. Block **435** überträgt dann die Steuerung zu Block **440**.

[0035] Wenn der Fehlerbeseitigungsmodus bei Block **405** ausgewählt wurde, überträgt Block **440** die Steuerung zu Block **445**. Ansonsten überträgt Block **440** die Steuerung zu Block **450**.

[0036] Wenn einer oder mehrere Programmfehler

während des Betreibens des Programms **135** bei Block **435** gefunden wurden, überträgt Block **445** die Steuerung zu Block **455**. Ansonsten überträgt Block **445** die Steuerung zu Block **460**.

[0037] Wenn bei Block **450** bestimmt wurde, dass mehrere DUTs **150** zum Testen vorhanden sind, überträgt Block **450** die Steuerung zu Block **430**. Ansonsten beendet Block **450** den Prozess.

[0038] Bei Block **455** werden Fehler in den Programmquellcodekomponenten **205** korrigiert. Block **455** überträgt dann die Steuerung zu Block **405**.

[0039] Wenn bei Block **460** bestimmt wird, dass weitere DUTs **150** zum Testen vorliegen, überträgt Block **460** die Steuerung zu Block **430**. Ansonsten beendet Block **460** die Steuerung bei Block **405**.

[0040] Sobald keine weiteren Fehler beim Betreiben des Testprogramms **135** gefunden werden und das Programm **135** erneut im Operationsmodus verknüpft ist, können Blöcke **425**, **430** und **435** nach Bedarf ausgeführt werden, ohne in der Schleife durch die anderen Schritte in Fig. 4 zu wandern. Die Entscheidung jedoch, die bei Block **403** getroffen wird, führt den Benutzer zu den Blöcken **425**, **430** und **435**, und die Entscheidungen, die bei Blöcken **440** und **450** getroffen werden, führen den Benutzer aus den und zurück zu den Blöcken **425**, **430** und **435**, nach Bedarf.

[0041] Bei repräsentativen Ausführungsbeispielen ist ein Softwareprogramm **135** offenbart, das einen Funktionsruf **210** aufweist. Wenn ein Programm **135** in einen ersten Modus verknüpft ist, ist eine erste kompilierte Routine **240a** aus einer ersten Bibliothek **235a** mit dem Funktionsruf **210** gekoppelt. Ansonsten ist eine zweite kompilierte Routine **240b** aus einer zweiten Bibliothek **235b** mit dem Funktionsruf **210** gekoppelt. Die erste kompilierte Routine **240a** ist aus einer Quellcoderoutine **300** kompiliert, wobei ein Parameter **310** in der Quellcoderoutine **300** auf einen ersten Wert eingestellt ist, und die zweite kompilierte Routine **240b** ist aus der Quellcoderoutine **300** kompiliert, wobei der Parameter **310** bei der Quellcoderoutine **300** auf einen zweiten Wert eingestellt ist. Die erste kompilierte Routine **240a** weist die selbe Funktionalität auf wie die zweite kompilierte Routine **240b** plus eine zusätzliche Funktionalität.

[0042] Bei einem anderen repräsentativen Ausführungsbeispiel ist ein Paar aus Bibliotheken **235a**, **235b** offenbart, das eine erste Bibliothek **235a** und eine zweite Bibliothek **235b** aufweist. Die erste Bibliothek **235a** weist eine erste kompilierte Routine **240a** auf, und die zweite Bibliothek **235b** weist eine zweite kompilierte Routine **240b** auf. Die erste kompilierte Routine **240a** ist aus einer Quellcoderoutine **300** kompiliert, wobei ein Parameter **310** bei der Quellcoderoutine **300** auf einen ersten Wert eingestellt ist, und die zweite kompilierte Routine **240b** ist aus der Quellcoderoutine **300** kompiliert, wobei der Parameter **310** bei der Quellcoderoutine **300** auf einen zweiten Wert eingestellt ist. Die erste kompilierte Routine **240a** weist die selbe Funktionalität auf wie die zweite kompilierte Routine **240b** plus eine zusätzliche Funktionalität.

[0043] Und bei einem anderen repräsentativen Ausführungsbeispiel ist ein Verfahren **400** zum Erzeugen eines Softwareprogramms **135** offenbart. Das Verfahren weist das Spezifizieren eines Modus auf, in dem das Programm **135** verknüpft werden soll, wobei der Quellcode für das Programm **135** einen Funktionsruf **210** aufweist. Wenn ein erster Modus bei dem Schritt spezifiziert wird, der den Modus spezifiziert, wird eine ausführbare Form des Programms **135** erzeugt durch Koppeln eines Objektcodes für eine erste kompilierte Routine **240a** aus einer ersten Bibliothek **235a** mit dem Funktionsruf **210** in einer vorangehend kompilierten Form des Programms **135**. Ansonsten wird die ausführbare Form des Programms **135** erzeugt durch Koppeln des Objektcodes für eine zweite kompilierte Routine **240b** aus einer zweiten Bibliothek **235b** mit dem Funktionsruf **210** in der vorangehend kompilierten Form des Programms **135**. Die erste kompilierte Routine **240a** weist die selbe Funktionalität auf wie die zweite kompilierte Routine **240b** plus eine zusätzliche Funktionalität.

[0044] Wie es der Fall ist, können bei vielen Datenverarbeitungsprodukten die Systeme, die oben beschrieben sind, als eine Kombination aus Hardware- und Software-Komponenten implementiert sein. Ferner kann die Funktionalität, die zur Verwendung der repräsentativen Ausführungsbeispiele erforderlich ist, in einem computerlesbaren Medium verkörpert sein (wie z. B. Disketten, herkömmliche Festplatten, DVDs, CD-ROMs, Flash-ROMs, nicht-flüchtiger ROM und RAM), die beim Programmieren des Computers **110** einer Testmaschine **105** verwendet werden sollen.

[0045] Der Ausdruck „Programmspeichermedium" ist hierin umfassend definiert, um jegliche Art von Computerspeicher zu umfassen, wie z. B. aber nicht ausschließlich Disketten, herkömmliche Festplatten, DVDs, CD-ROMs, Flash-ROMs, nicht-flüchtigen ROM und RAM.

[0046] Bei einem repräsentativen Ausführungsbeispiel der Softwareprogramme **135** und Verfahren, die hierin offenbart sind, schreibt der Benutzer ein Programm, bei dem Rufe zu Bibliotheksroutinen getätigt werden. Diese Rufe können aus kompilierten Testerroutinen erfolgen, die sich in einer von zwei alternativen Bibliotheken **235a**, **235b** befinden. Die Routinen in der ersten **235a** dieser Bibliotheken liefern dem Benutzer zusätzliche Informationen im Hinblick auf

die Operation der Routinen und werden üblicherweise zur Fehlerbeseitigung bei einem Softwareprogramm **135** verwendet, das z. B. ein Softwaretestprogramm **135** sein könnte. Die Routinen in der zweiten **235b** dieser Bibliotheken sind dieselben wie die Routinen in der ersten Bibliothek **235a** ohne diese zusätzlichen Informationen. In der Tat ist der Quellcode für beide Versionen der kompilierten Routinen **240a**, **240b** derselbe abgesehen von dem Wert eines Parameters **310**. Der Wert des Parameters **310** wird als eine Steuerung während der Kompilierung der ersten und zweiten kompilierten Routine **240a**, **240b** verwendet, wodurch ein ausgewähltes, operationsmäßiges Prüfen und andere Funktionen ermöglicht werden, wie das Berichten von Werten an Zwischenpunkten beim Betrieb des Programms **135** durch die kompilierten Versionen der ersten Bibliotheksroutinen **235a**. Programmbestätigung und Fehlerbeseitigung bei Testprogrammen **135** mit neu freigegebenen Bibliotheksfunktionen wird vereinfacht, da der Quellcode für das Softwaretestprogramm **135** nicht neu kompiliert werden muss. Der Quellcode für das Softwaretestprogramm **135** muss nur mit der Fehlerbeseitigungsbibliothek (der ersten Bibliothek) **235a** während der Fehlerbeseitigung verknüpft sein und dann mit der Operationsbibliothek (der zweiten Bibliothek) **235b** neu verknüpft werden, sobald eine entsprechende Operation des Softwareprogramms **135** bestätigt wurde.

[0047] Während die Erörterung hierin im Hinblick auf Softwaretestprogramme **135** zum Testen elektronischer Komponenten **150** erfolgte, werden Fachleute auf dem Gebiet erkennen, dass die hierin offenbarten Techniken nicht auf solche Softwareprogramme **135** beschränkt sind. Die selben Funktionsrufe werden durch das Softwareprogramm **135** getätigt, unabhängig davon, welche Bibliothek **235a**, **235b** zum Erfüllen der Rufe verwendet wird. Zusätzlich dazu, dadurch, dass der Code nicht neu kompiliert wird, den der Benutzer schreibt, sondern nur der, der durch den Hersteller der Testmaschine **105** geliefert wird, ist der kompilierte Code, der verändert wird, auf den beschränkt, der durch den Hersteller geliefert wird, wodurch eine Verwirrung im Hinblick auf die Quelle jeglichen Problems reduziert wird und die Möglichkeit reduziert wird, dass Codeänderungen durch den Benutzer zusätzliche Probleme einbringen.

[0048] Die darstellenden Ausführungsbeispiele, die hierin detailliert beschrieben wurden, wurden beispielhaft und nicht als Einschränkung vorgelegt. Fachleute auf dem Gebiet werden erkennen, dass verschiedene Änderungen an Form und Details der beschriebenen Ausführungsbeispiele ausgeführt werden können, was zu entsprechenden Ausführungsbeispielen führt, die innerhalb des Schutzbereichs der beiliegenden Ansprüche verbleiben.

Patentansprüche

1. Softwareprogramm, das folgende Merkmale aufweist:
einen Funktionsruf (**210**),
wobei,
wenn das Programm (**135**) in einem ersten Modus verknüpft ist: eine erste kompilierte Routine (**240a**) aus einer ersten Bibliothek (**235a**) mit dem Funktionsruf (**210**) gekoppelt ist,
ansonsten: eine zweite kompilierte Routine (**240b**) aus einer zweiten Bibliothek (**235b**) mit dem Funktionsruf (**210**) gekoppelt ist,
– wobei die erste kompilierte Routine (**240a**) aus einer Quellcoderoutine (**300**) kompiliert ist, wobei ein Parameter (**310**) in der Quellcoderoutine (**300**) auf einen ersten Wert eingestellt ist,
wobei die zweite kompilierte Routine (**240b**) aus der Quellcoderoutine (**300**) kompiliert ist, wobei der Parameter (**310**) in der Quellcoderoutine (**300**) auf einen zweiten Wert eingestellt ist, und
wobei die erste kompilierte Routine (**240a**) die selbe Funktionalität aufweist wie die zweite kompilierte Routine (**240b**) plus eine zusätzliche Funktionalität.

2. Softwareprogramm (**135**) gemäß Anspruch 1, bei dem die zusätzliche Funktionalität der ersten kompilierten Routine (**240a**) konfiguriert ist, um Informationen für eine Fehlerbeseitigung bei dem Programm (**135**) zu liefern.

3. Softwareprogramm (**135**) gemäß Anspruch 1 oder 2, bei dem das Programm (**135**) konfiguriert ist, um auf einer Testmaschine (**105**) zu laufen.

4. Softwareprogramm (**135**) gemäß einem der Ansprüche 1 bis 3, bei dem das Programm (**135**) konfiguriert ist, um eine Komponente (**150**) zu testen.

5. Softwareprogramm (**135**) gemäß einem der Ansprüche 1 bis 4, bei dem das Programm konfiguriert ist, um auf einer Testmaschine zu laufen, und konfiguriert ist, um eine Komponente zu testen.

6. Softwareprogramm (**135**) gemäß einem der Ansprüche 1 bis 5,
bei dem das Programm in dem ersten Modus verknüpft ist und, wenn die ausführbare Form des Programms ausgeführt wird:
ein Ruf zu der ersten kompilierten Routine folgendes aufweist:
einen Ruf zu einem ersten Programmrohling in einer Programmrohling-Bibliothek, wobei der erste Programmrohling die Adresse der ersten kompilierten Routine in einer Adresstabelle nachschlägt und nachfolgend den Ruf zu der ersten kompilierten Routine zu dieser Adresse umleitet;
ansonsten, wenn das Programm in dem zweiten Modus verknüpft ist und wenn die ausführbare Form des Programms ausgeführt wird:

ein Ruf zu der zweiten kompilierten Routine folgendes aufweist:
einen Ruf zu einem zweiten Programmrohling in der Programmrohling-Bibliothek, wobei der zweite Programmrohling die Adresse der zweiten kompilierten Routine in der Adresstabelle nachschlägt und nachfolgend den Ruf zu der zweiten kompilierten Routine zu dieser Adresse umleitet.

7. Paar aus Bibliotheken, das folgende Merkmale aufweist:
eine erste Bibliothek, die eine erste kompilierte Routine aufweist;
eine zweite Bibliothek, die eine zweite kompilierte Routine aufweist, wobei die erste kompilierte Routine aus einer Quellcoderoutine kompiliert ist, wobei ein Parameter in der Quellcoderoutine auf einen ersten Wert eingestellt ist, wobei die zweite kompilierte Routine aus der Quellcoderoutine kompiliert ist, wobei der Parameter in der Quellcoderoutine auf einen zweiten Wert eingestellt ist, und wobei die erste kompilierte Routine (**240a**) die selbe Funktionalität aufweist wie die zweite kompilierte Routine (**240b**) plus eine zusätzliche Funktionalität.

8. Paar aus Bibliotheken (**235a**, **235b**) gemäß Anspruch 7, bei dem die zusätzliche Funktionalität der ersten kompilierten Routine (**240a**) konfiguriert ist, um Informationen zum Fehlerbeseitigen bei einem Programm (**135**) zu liefern, das die erste kompilierte Routine (**240a**) aufruft.

9. Paar aus Bibliotheken gemäß Anspruch 8, bei dem das Programm konfiguriert ist, um eine Komponente zu testen.

10. Paar aus Bibliotheken (**235a**, **235b**) gemäß einem der Ansprüche 7 bis 9, bei dem die erste kompilierte Routine und die zweite kompilierte Routine konfiguriert sind, um den Test einer Komponente zu unterstützen.

11. Paar aus Bibliotheken (**235a**, **235b**) gemäß einem der Ansprüche 7 bis 10, bei dem die erste kompilierte Routine und die zweite kompilierte Routine konfiguriert sind, um die Ausführung eines Programms zum Testen einer Komponente auf einer Testmaschine zu unterstützen.

12. Paar aus Bibliotheken (**235a**, **235b**) gemäß einem der Ansprüche 7 bis 11, bei dem die erste kompilierte Routine und die zweite kompilierte Routine konfiguriert sind, um den Test einer Komponente zu unterstützen, und konfiguriert sind, um die Ausführung eines Programms zum Testen der Komponente auf einer Testmaschine zu unterstützen.

13. Verfahren zum Erzeugen eines Softwareprogramms (**135**), das folgende Schritte aufweist:
Spezifizieren eines Modus, in dem das Programm (**135**) verknüpft wird, wobei der Quellcode für das Programm (**135**) einen Funktionsruf (**210**) aufweist; und
wenn ein erster Modus bei dem Schritt spezifiziert wird, der den Modus spezifiziert:
Erzeugen einer ausführbaren Form des Programms (**135**) durch Koppeln eines Objektcodes für eine erste kompilierte Routine (**240a**) aus einer ersten Bibliothek (**235a**) mit dem Funktionsruf (**210**) in einer vorangehend kompilierten Form des Programms (**135**),
ansonsten:
Erzeugen der ausführbaren Form des Programms (**135**) durch Koppeln des Objektcodes für eine zweite kompilierte Routine (**240b**) aus einer zweiten Bibliothek (**235b**) mit dem Funktionsruf (**210**) in der vorangehend kompilierten Form des Programms (**135**), wobei die erste kompilierte Routine (**240a**) die selbe Funktionalität aufweist wie die zweite kompilierte Routine (**240b**) plus eine zusätzliche Funktionalität.

14. Verfahren gemäß Anspruch 13, bei dem die zusätzliche Funktionalität der ersten kompilierten Routine konfiguriert ist, um Informationen zum Fehlerbeseitigen bei dem Programm zu liefern.

15. Verfahren gemäß Anspruch 13 oder 14, bei dem das Programm konfiguriert ist, um auf einer Testmaschine zu laufen.

16. Verfahren gemäß einem der Ansprüche 13 bis 15, bei dem das Programm konfiguriert ist, um eine Komponente zu testen.

17. Verfahren gemäß einem der Ansprüche 13 bis 16, bei dem das Programm konfiguriert ist, um auf einer Testmaschine zu laufen und eine Komponente zu testen.

18. Verfahren gemäß einem der Ansprüche 13 bis 17, das ferner folgende Schritte aufweist:
Laden der ausführbaren Form des Programms (**135**) in einen Computer (**110**); und
Betreiben der ausführbaren Form des Programms (**135**).

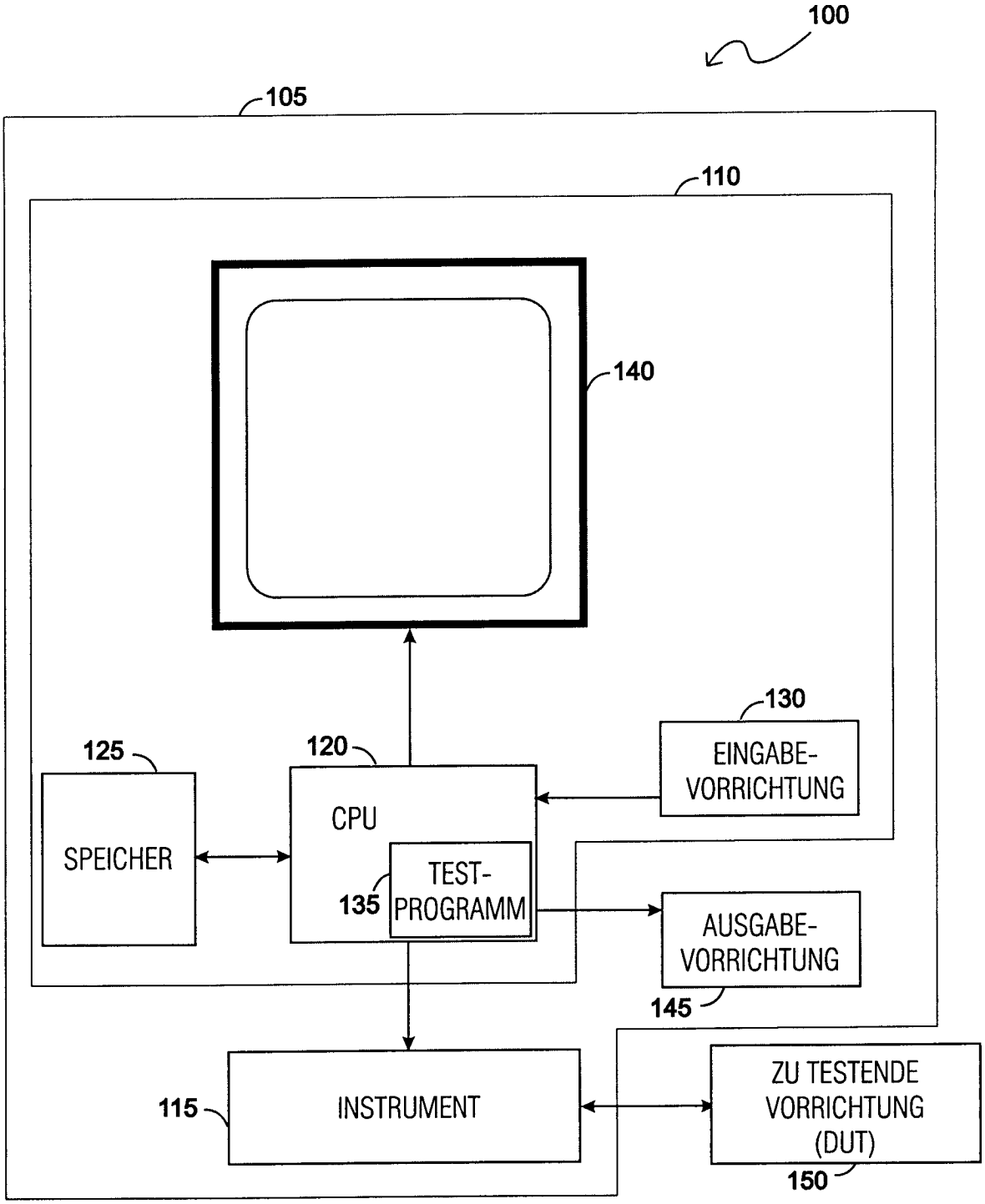
19. Verfahren gemäß Anspruch 18, das ferner folgenden Schritt aufweist:
vor dem Schritt des Betreibens des Programms, Verbinden einer Komponente mit einer Testmaschine, wobei die Testmaschine den Computer aufweist.

20. Verfahren gemäß Anspruch 18 oder 19,
bei dem, wenn das Programm in dem ersten Modus verknüpft ist, der Schritt des Betreibens der ausführbaren Form des Programms folgendes aufweist:
Rufen der ersten kompilierten Routine, wobei das Rufen der ersten kompilierten Routine folgenden Schritt aufweist:
Rufen eines ersten Programmrohlings in einer Programmrohling-Bibliothek, wobei das Rufen des ers-

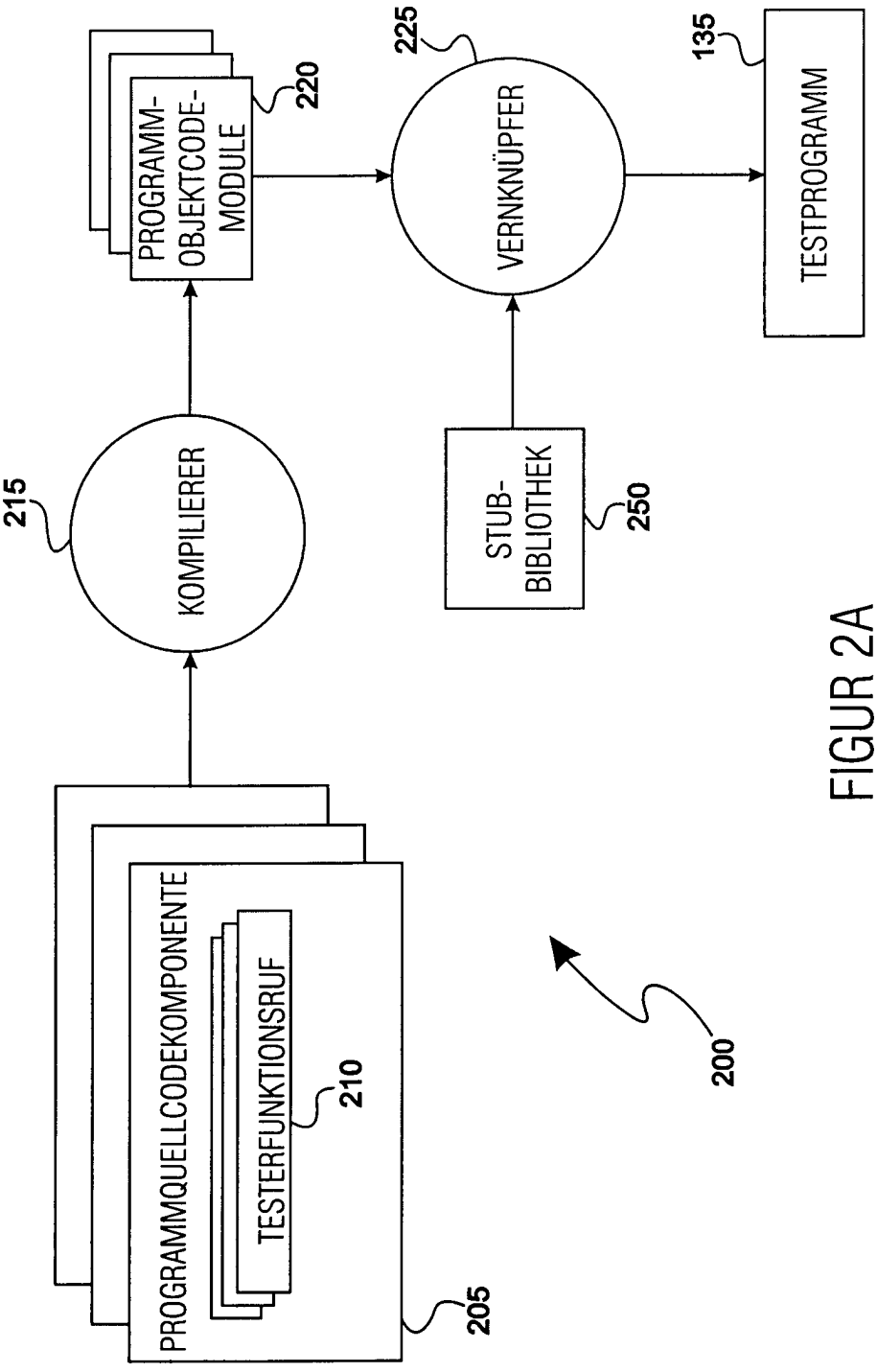
ten Programmrohlings folgende Schritte aufweist:
Nachschlagen der Adresse der ersten kompilierten Routine in einer Adresstabelle und
Rufen der ersten kompilierten Routine in der ersten Bibliothek;
ansonsten, wenn das Programm in dem zweiten Modus verknüpft ist, der Schritt des Betreibens der ausführbaren Form des Programms folgendes aufweist:
Rufen der zweiten kompilierten Routine, wobei das Rufen der zweiten kompilierten Routine folgendes aufweist:
Rufen eines zweiten Programmrohlings in der Programmrohling-Bibliothek, wobei das Rufen des zweiten Programmrohlings folgende Schritte aufweist:
Nachschlagen der Adresse der zweiten kompilierten Routine in der Adresstabelle und
Rufen der zweiten kompilierten Routine in der zweiten Bibliothek.

Es folgen 5 Blatt Zeichnungen

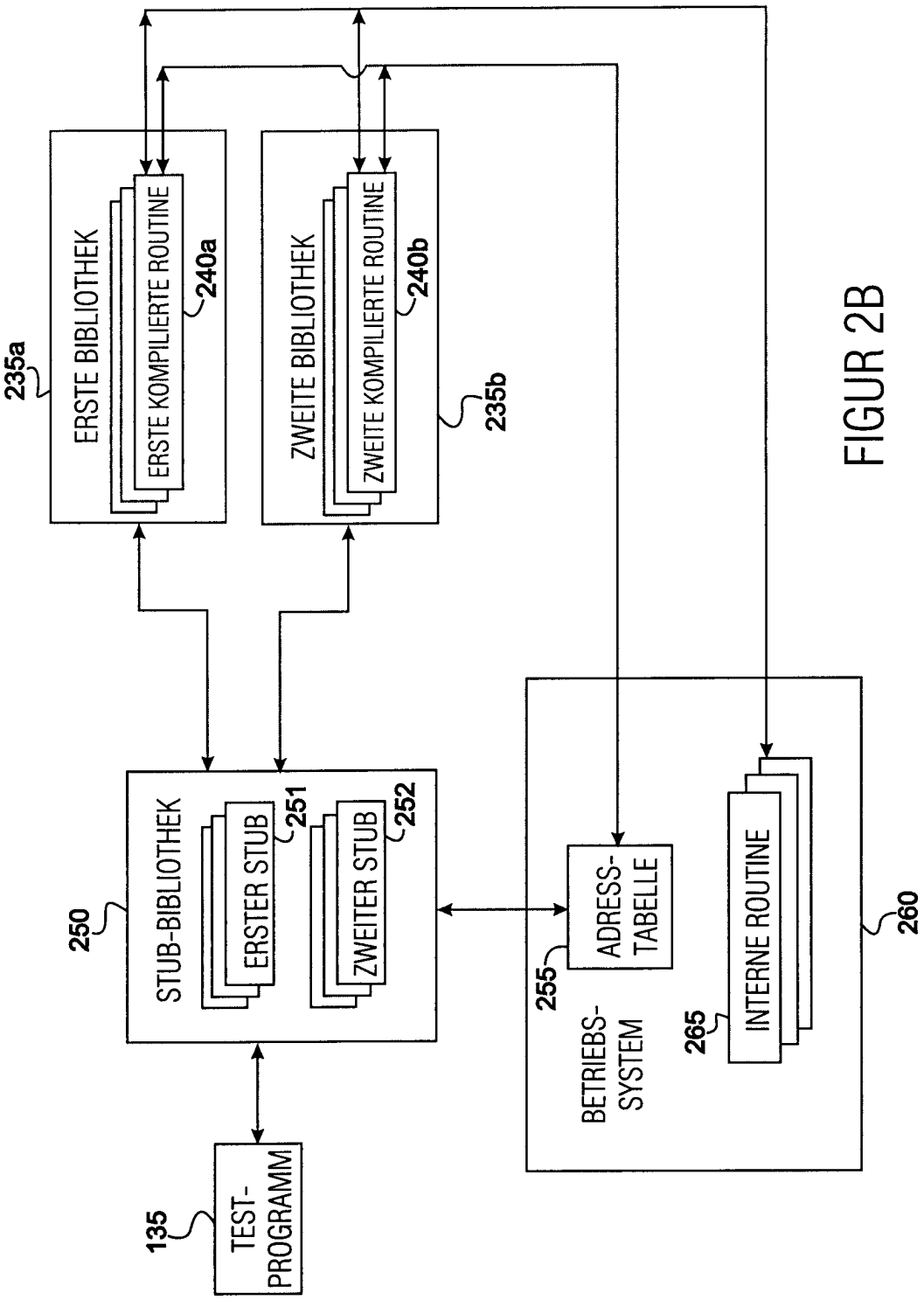
Anhängende Zeichnungen



FIGUR 1


PROGRAMMQUELLCODEKOMPONENTE
TESTERFUNKTIONSRUF
210
205
KOMPILIERER
215
PROGRAMM-OBJEKTCODE-MODULE
220
STUB-BIBLIOTHEK
250
VERNKNÜPFER
225
TESTPROGRAMM
135
200

FIGUR 2A


ERSTE BIBLIOTHEK
ERSTE KOMPILIERTE ROUTINE
240a
235a
ZWEITE BIBLIOTHEK
ZWEITE KOMPILIERTE ROUTINE
240b
235b
STUB-BIBLIOTHEK
ERSTER STUB
251
ZWEITER STUB
252
250
ADRESS-TABELLE
255
BETRIEBS-SYSTEM
INTERNE ROUTINE
265
260
TEST-PROGRAMM
135

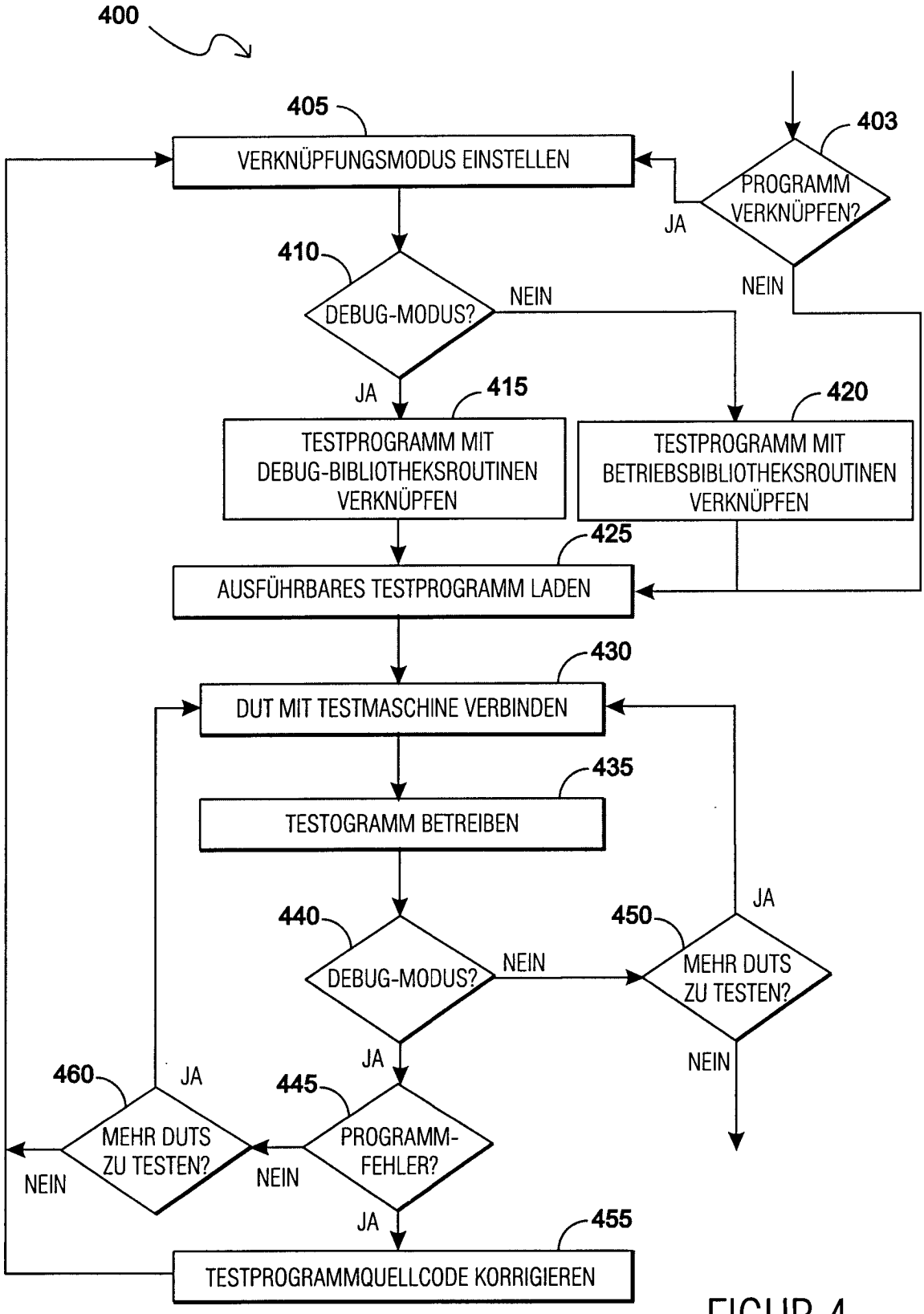
FIGUR 2B

310

```
#ifdef DEBUG
#define CHECK_STACK SCheckStack()
#define CHECK_ATM(x,y) SCheckAtm(x,y)
#else
#define CHECK_STACK
#define CHECK_ATM(x,y)
#endif

int assign_dut_channel(cpinlist_t arg1, int arg2)
{
   CHECK_STACK;
   int result = (PETable->assign_dut_channel)(arg1, arg2);
   CHECK_ATM(result, "assign_dut_channel");
   return result;
}
```

FIGUR 3


400
405
VERKNÜPFUNGSMODUS EINSTELLEN
403
PROGRAMM VERKNÜPFEN?
JA
NEIN
410
DEBUG-MODUS?
NEIN
JA
415
TESTPROGRAMM MIT DEBUG-BIBLIOTHEKSROUTINEN VERKNÜPFEN
420
TESTPROGRAMM MIT BETRIEBSBIBLIOTHEKSROUTINEN VERKNÜPFEN
425
AUSFÜHRBARES TESTPROGRAMM LADEN
430
DUT MIT TESTMASCHINE VERBINDEN
435
TESTOGRAMM BETREIBEN
440
DEBUG-MODUS?
NEIN
450
MEHR DUTS ZU TESTEN?
JA
NEIN
JA
460
MEHR DUTS ZU TESTEN?
JA
NEIN
445
PROGRAMM-FEHLER?
NEIN
JA
455
TESTPROGRAMMQUELLCODE KORRIGIEREN

FIGUR 4