(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5022262号
(P5022262)

(45)発行日　平成24年9月12日(2012.9.12)　(24)登録日　平成24年6月22日(2012.6.22)

(51)Int.Cl.　F I

G06F 11/28 (2006.01)	G06F	11/28	A
G06F 11/22 (2006.01)	G06F	11/28	315A
G01R 31/28 (2006.01)	G06F	11/22	310A
	G01R	31/28	H

請求項の数 7　(全 18 頁)

(21)出願番号　特願2008-30180(P2008-30180)
(22)出願日　平成20年2月12日(2008.2.12)
(65)公開番号　特開2009-193109(P2009-193109A)
(43)公開日　平成21年8月27日(2009.8.27)
審査請求日　平成22年12月1日(2010.12.1)

(73)特許権者　390005175
株式会社アドバンテスト
東京都練馬区旭町１丁目３２番１号
(74)代理人　100079108
弁理士　稲葉　良幸
(74)代理人　100093861
弁理士　大賀　眞司
(74)代理人　100109346
弁理士　大貫　敏史
(74)代理人　100117189
弁理士　江口　昭彦
(74)代理人　100134120
弁理士　内藤　和彦

最終頁に続く

(54)【発明の名称】デバッグ中にツールを使用可能な試験システム及び方法

(57)【特許請求の範囲】
【請求項１】
デバイスの試験を行うための試験プロセスを実行可能な試験システムであって、
前記試験プロセスは、
前記デバイスの試験方法を記述したテストクラスを実行するためのテストクラス・スレッドと、
前記デバイスの試験に利用可能な関数を含むツールを実行するためのツール・スレッドと、を含み、
前記テストクラスのデバッグ中に、前記テストクラス・スレッドを停止状態に制御するとともに、前記ツール・スレッドを動作状態に制御することのできる試験システム。
【請求項２】
デバイスの試験を行うための試験プロセスを実行可能な試験システムであって、
前記試験プロセスは、
前記デバイスの試験方法を記述したテストクラスを実行するためのテストクラス・スレッドと、
前記デバイスの試験に利用可能な関数を含むツールを実行するためのツール・スレッドと、を含み、
前記テストクラスのデバッグ中の状態として、前記テストクラス・スレッドと前記ツール・スレッドとを共に動作させる通常実行状態と、前記テストクラス・スレッドと前記ツール・スレッドを共に停止させるデバッギング状態と、前記テストクラス・スレッドを停

止させて且つ前記ツール・スレッドを動作させるツール使用可能状態と備える試験システム。
【請求項３】
ユーザの操作に連動して、前記デバッギング状態と前記ツール使用可能状態とを切り替えることを特徴とする請求項２記載の試験システム。
【請求項４】
デバッガ用のウィンドウがアクティブの場合には、前記デバッギング状態に切り替え、ツールを操作するためのウィンドウがアクティブの場合には、前記ツール使用可能状態に切り替えることを特徴とする請求項２記載の試験システム。
【請求項５】
デバイスの試験を行うための試験プロセスを実行可能な試験システムにおいて、前記デバイスの試験方法を記述したテストクラスのデバッグ中にツールを使用可能にする方法であって、
前記試験プロセスに含まれる全てのスレッドを停止させるステップと、
前記テストクラスを実行するためのスレッドを停止させた状態で凍結するとともに、前記デバイスの試験に利用可能な関数を含むツールを実行するためのスレッドを動作させるステップと、
を含む方法。
【請求項６】
請求項５に記載のツールを使用可能にする方法をコンピュータに実行させるためのプログラム。
【請求項７】
請求項６に記載のプログラムを記録したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は自動試験装置の技術分野に関する。特に、本発明は、自動試験装置に組み込む試験プログラムのデバッグ支援処理技術に関する。
【背景技術】
【０００２】
従来、自動試験装置（以下「ＡＴＥ」という。）は、ＡＴＥメーカー各社各様の仕様で提供されていたため、ピン構成や測定ユニット等の構成の自由度が低く、また、試験プログラム資産の再利用が困難であった。このような背景から、デバイスの機能に応じて最適な構成にシステムを変更できるようなスケーラブルで柔軟なＡＴＥを実現すべく、標準化されたインタフェースを持つオープン・アーキテクチャＡＴＥが提案されている。
【０００３】
例えば、ＯＰＥＮＳＴＡＲ（登録商標）は、このようなオープン・アーキテクチャＡＴＥの規格の一つである（非特許文献１参照）。そして、Ｔ２０００テスト・システムは、ＯＰＥＮＳＴＡＲ規格のオープン・アーキテクチャを採用した試験システムである。
【０００４】
このＴ２０００テスト・システムでは、デバイス試験のアルゴリズムを、Ｃ＋＋言語のクラスとして記述する（以下、このクラスを「テストクラス」という。）。一般に、テストクラスの作成には、Ｍｉｃｒｏｓｏｆｔ Ｖｉｓｕａｌ Ｓｔｕｄｉｏ（登録商標）等のソフトウェア開発環境を使用し、テストクラスを言語レベルでデバッグするためには、Ｍｉｃｒｏｓｏｆｔ Ｖｉｓｕａｌ Ｓｔｕｄｉｏデバッガ（ＶＳ Ｄｅｂｕｇｇｅｒ）等のデバッガを使用する。
【非特許文献１】「Ｓｅｍｉｃｏｎｄｕｃｔｏｒ Ｔｅｓｔ Ｃｏｎｓｏｒｔｉｕｍ」、［online］、［平成２０年２月１２日検索］、インターネット＜ＵＲＬ：http://www.semitest.org/jp/home＞
【発明の開示】

【発明が解決しようとする課題】
【０００５】
ところで、デバッガは、プログラムの実行を特定の位置で中断するための「ブレークポイント」機能等を備えている。この機能を使用することにより、ユーザは、自分の書いたプログラムの実行を、任意の位置（ブレークポイント）で停止し、行単位で細かくデバッグすることができる。しかし、従来のＡＴＥでは、テストクラスのデバッグ中に、ツールを使用できないという問題がある。なお、ツールとは、デバイスの試験に関わる作業で利用される種々の処理の関数（ＡＰＩ）であり、例えば、デバイスに所定形状の波形を入力するためのツールや、デバイスからの出力を分析するためのツール等を含む。
【０００６】
図７は、この問題点を説明するためのものであり、従来のＡＴＥにおけるテストクラスのデバッグ処理の概略を示す図である。ユーザが作成したテストクラス３３０は、試験プロセス３２０内で動作し、このテストクラス３３０のコードをデバッグするために、デバッガ３４０が使用される。同図は、デバッガ３４０にアタッチされたテストクラスのソースコード３４１のうち、「ＤＵＴＩＤ＿ｔ　ｄｕｔＩＤ＝１；」の部分にユーザがブレークポイント３４２を設定した場合を例示している。
【０００７】
従来のＡＴＥでは、デバッガ３４０でテストクラス３３０のデバッグを行うとき、ブレークポイント３４２が設定されたラインで処理を中断してブレーク状態に遷移するが、このとき、デバッガ３４０は、ＡＴＥ上で実行される全ての処理を停止する。なお、プロセスに複数のスレッドが存在する場合、従来のデバッガのブレーク状態では、全てのスレッドの処理が停止される。
【０００８】
このブレーク状態では、あらゆる要求に対する対応も停止するため、ツール３１０を使用することもできない。ツール３１０を使用するのであれば、デバッガ３４０によるプロセスの停止（ブレーク）を解除する必要がある。ここで、ツールを使用するために、少しの間だけ停止を解除してツールを使用した後、すぐに停止状態に戻ってくることも考えられる。しかし、この場合、停止解除したわずかの間に、テストクラスの実行も行われ、ユーザのコードの実行場所も進んでしまう。このため、目的場所のユーザコードのデバッグを継続することはできない。
【０００９】
また、デバッガの中には、サスペンド機能を備えるものもある。このサスペンド機能とは、テストクラスが複数のテストユニットに分けられる場合に、あるテストユニットによるテストと次のテストユニットによるテストとの間で処理をサスペンド（一時中断）し、サスペンド中であればツールを使用することができるというものである。しかし、サスペンド機能では、予め定められたテストユニット間でのみツールを使うことができるものの、テストクラスの任意のポイントでツールを使用できるわけではない。
【００１０】
しかし、本来、デバッガもツールも、テストクラスの動作に関する情報を集めるためのものである。そのため、これらを同時に使用して、テストクラスのデバッグ中にツールを使用できるようになれば、ユーザにとって大きなメリットになる。
【００１１】
本発明は、かかる事情に鑑みてなされたものであり、デバッガにてテストクラスのデバッグをするときに、テストクラスの実行を停止したままの状態でツールを使用できるようにするためのデバッグ支援機能を備えた試験システムを提供することを目的とする。
【００１２】
さらに、ユーザの操作に連動して、テストクラスをデバッグ可能な状態とツールを使用可能な状態との間の状態遷移を自動的に行うことのできる試験システム提供することを目的とする。
【課題を解決するための手段】

【００１３】
上記課題を解決するため、本発明に係る試験システムの幾つかの態様の一つは、デバイスの試験を行うための試験プロセスを実行可能な試験システムであって、この試験プロセスは、二以上のスレッドを含んで構成される。二以上のスレッドのうち一つは、デバイスの試験方法を記述したプログラムを実行するための第一のスレッドである。そして、この試験システムは、プログラムのデバッグ中に、第一のスレッドのみを停止状態に制御することができる。
【００１４】
また、本発明に係る試験システムの幾つかの態様の一つは、デバイスの試験を行うための試験プロセスを実行可能な試験システムであって、この試験プロセスは、デバイスの試験方法を記述したテストクラスを実行するためのテストクラス・スレッドと、デバイスの試験に利用可能な関数を含むツールを実行するためのツール・スレッドと、を含む。そして、この試験システムは、テストクラスのデバッグ中に、テストクラス・スレッドを停止状態に制御するとともに、ツール・スレッドを動作状態に制御することができる。
【００１５】
さらに、本発明に係る試験システムの幾つかの態様の一つは、デバイスの試験を行うための試験プロセスを実行可能な試験システムであって、この試験プロセスは、デバイスの試験方法を記述したテストクラスを実行するためのテストクラス・スレッドと、デバイスの試験に利用可能な関数を含むツールを実行するためのツール・スレッドと、を含む。そして、この試験システムは、テストクラスのデバッグ中の状態として、テストクラス・スレッドとツール・スレッドとを共に動作させる通常実行状態と、テストクラス・スレッドとツール・スレッドを共に停止させるデバッギング状態と、テストクラス・スレッドを停止させて且つツール・スレッドを動作させるツール使用可能状態と備える。
【００１６】
ここで、試験システムは、ユーザの操作に連動して、デバッギング状態とツール使用可能状態とを切り替えることが好ましい。また、試験システムは、デバッガ用のウィンドウがアクティブの場合には、デバッギング状態に切り替え、ツールを操作するためのウィンドウがアクティブの場合には、ツール使用可能状態に切り替えることが好ましい。
【００１７】
また、本発明に係るツールを使用可能にする方法の幾つかの態様の一つは、デバイスの試験を行うための試験プロセスを実行可能な試験システムにおいて、デバイスの試験方法を記述したテストクラスのデバッグ中にツールを使用可能にする方法であって、試験プロセスに含まれる全てのスレッドを停止させるステップと、テストクラスを実行するためのスレッドを停止させた状態で凍結するとともに、デバイスの試験に利用可能な関数を含むツールを実行するためのスレッドを動作させるステップと、を含む。
【００１８】
本発明のプログラムは、本発明に係るツールを使用可能にする方法の各処理ステップをコンピュータに実行させることを特徴とする。本発明のプログラムは、ＣＤ－ＲＯＭ等の光学ディスク、磁気ディスク、半導体メモリなどの各種の記録媒体を通じて、又は通信ネットワークなどを介してダウンロードすることにより、コンピュータにインストール又はロードすることができる。
【００１９】
なお、本明細書等において、「手段」又は「部」とは、単に物理的手段を意味するものではなく、その手段又は部が有する機能をソフトウェアによって実現する場合も含む。また、１つの手段又は部が有する機能が２つ以上の物理的手段により実現されても、２つ以上の手段又は部の機能が１つの物理的手段により実現されてもよい。
【００２０】
また、本明細書等において、「スレッド」とは、プロセスの中で、プログラムを実行している単位である。１つのプロセスの中に、複数のスレッドが存在することができる。また、同じプロセスに属するスレッドはメモリなどのリソースを共有する。

【００２１】
さらに、本明細書等において、ベンダ等が提供する計測機器のハードウェアを「モジュール」といい、そのハードウェアを対象とするベンダ等のソフトウェアを「モジュール・ソフトウェア」と呼ぶ。
【発明の効果】
【００２２】
本発明によると、デバッガにてテストクラスのデバッグをするときに、テストクラスの実行を停止したままの状態でツールを使用することができるようになる。これにより、ユーザは、テストクラスのデバッグをより効率的に行うことができるようになる。
【００２３】
また、本発明によると、ユーザの操作に連動して、デバッグ可能な状態とツール使用可能な状態との間の状態遷移を自動的に行うことが可能にある。これにより、ユーザは、デバッグ中にスレッドの状態や動作を意識することなく、簡単にツールを使用することができるようになる。
【発明を実施するための最良の形態】
【００２４】
以下、本発明の実施の形態について詳細に説明する。なお、同一の要素には同一の符号を付し、重複する説明を省略する。また、以下の実施の形態は、本発明を説明するための例示であり、本発明をその実施の形態のみに限定する趣旨ではない。さらに、本発明は、その要旨を逸脱しない限り、さまざまな変形及び応用が可能である。
【００２５】
図１は、本発明の一実施形態による試験システム１００のシステムアーキテクチャを示す。試験システム１００は、試験信号を生成して被試験デバイス（Ｄｅｖｉｃｅ ｕｎｄｅｒ Ｔｅｓｔ。以下「ＤＵＴ」という。）１１２に供給し、ＤＵＴ１１２が試験信号に基づいて動作した結果出力する結果信号が期待値と一致するか否かに基づいてＤＵＴ１１２の良否を判断する。なお、本実施形態に係る試験システム１００は、オープン・アーキテクチャにより実現されるものとして説明するが、本発明は、オープン・アーキテクチャの試験システムに限定されるものではない。
【００２６】
本実施形態では、システムコントローラ（ＳｙｓＣ）１０２がネットワークを介して複数のサイトコントローラ（ＳｉｔｅＣ）１０４に接続される。システムコントローラ１０２は、エンド・ユーザが通常作業を行うホストコンピュータであり、試験システム１００全体を管理する役割を果たす。また、システムコントローラ１０２は、サイトコントローラ１０４に対する処理の要求の発行や、サイトコントローラ１０４間の処理の調停を行う。ユーザのアプリケーションや標準のＧＵＩツールは、システムコントローラ１０２上で動作し、サイトコントローラ１０４と通信を行って機能を実現する。
【００２７】
また、システムコントローラ１０２は、サイトコントローラ１０４上でモジュール１０８の制御を行うモジュール・ソフトウェアや、ユーザの試験プログラムやパターン・プログラムなどを保管するストレージを備える。これらは、必要に応じてサイトコントローラ１０４に送られ、実行される。
【００２８】
各サイトコントローラ１０４は、試験サイト１１０に配置される１つ又は複数のモジュール１０８を制御して試験を実行するために、モジュール接続イネーブラ１０６を通して、モジュール１０８に接続される。この試験は、ユーザの作成する試験プログラムに基づいて実行される。なお、サイトコントローラ１０４は、一つのＤＵＴ１１２を試験する試験サイト１１０を複数個、同時に制御するようにしてもよい。
【００２９】
サイトコントローラ１０４の主な役割として、次の３つが挙げられる。まず、試験プログラムが指定する試験サイト１１０の構成に従い、モジュール接続イネーブラ１０６を構

成し、サイトコントローラ１０４とモジュール１０８間のバス１０７の接続を確立する。また、試験サイト１１０内のモジュール１０８の制御を行うモジュール・ソフトウェアを実行する。さらに、試験プログラムを実行し、各試験サイト１１０のＤＵＴ１１２の試験を実行する。

【００３０】

なお、動作環境によっては、システムコントローラ１０２は、サイトコントローラ１０４の動作とは別のＣＰＵ（中央演算装置）上に配置することができる。別法では、システムコントローラ１０２及びサイトコントローラ１０４は、共通のＣＰＵを共有することができる。同様に、各サイトコントローラ１０４は、その専用のＣＰＵ上に配置することができるし、また、同じＣＰＵ内の別個のプロセス又はスレッドとして配置することもできる。

【００３１】

モジュール接続イネーブラ１０６は、サイトコントローラ１０４とモジュール１０８とのバス１０７の接続を任意に構成することのできるスイッチである。モジュール接続イネーブラ１０６は、接続されるハードウェアモジュール１０８の構成を変更できるようにするとともに、データ転送用（パターンデータのロード用、応答データの収集用、制御用等）のバスとしての役割も果たす。実現可能なハードウェアの実装形態は、専用接続、交換接続、バス接続、リング接続及びスター接続を含む。モジュール接続イネーブラ１０６は、たとえばスイッチマトリクスとして実装することができる。

【００３２】

モジュール１０８は、ＤＵＴ１１２に試験信号を供給する計測機器のハードウェアである。本実施形態においては、モジュール１０８として、オープン・アーキテクチャに基づく各種のモジュールを用いることができる。また、モジュール・ハードウェア１０８を動作させるためのソフトウェアをモジュール・ソフトウェアという。モジュール・ソフトウェアは、デバイス測定時にモジュール１０８の制御をつかさどるモジュール・ドライバ、モジュール１０８の校正と診断を行う校正診断ソフトウェア、モジュール１０８の動作をソフトウェアでエミュレートするエミュレーション・ソフトウェア、モジュール１０８固有のパターン・コンパイラ、およびＧＵＩツールなどを含む。

【００３３】

各試験サイト１１０は、それぞれ一つのＤＵＴ１１２に関連付けられる。ＤＵＴ１１２は、ロードボード１１４を通して、対応する試験サイト１１０のモジュール１０８に接続される。

【００３４】

図２は、試験サイト１１０及びロードボード１１４のハードウェアの概略構成と各種設定用ファイルとの関係の一例を示す図である。図２に示すように、モジュール１０８は、試験システム１００のテストヘッド１３２内のモジュール・スロットに差し込まれる。

【００３５】

試験サイト１１０の構成は、テキスト形式で記述されるソケットファイル１１８で指定される。このソケットファイル１１８には、ＤＵＴ１１２ごとに、ＤＵＴソケット１２０のＤＵＴピン１２２とロードボード１１４上のコネクタピン１２４との接続が記述される。コネクタピン１２４は、ブロック番号とコネクタ番号が付けられており、例えば、「１２．３」のように、「（ブロック番号）．（コネクタ番号）」の形式で表記される。ロードボード１１４のコネクタピン１２４は、テスタ・インタフェース・ユニット（Ｔｅｓｔｅｒ　Ｉｎｔｅｒｆａｃｅ　Ｕｎｉｔ。以下「ＴＩＵ」という。）１２６を介してモジュール１０８のピン１２８と接続される。

【００３６】

一方、ロードボード１１４は試験対象となるデバイスによって異なる場合が多い。そのため、モジュール・ソフトウェアが個々のロードボード１１４の実装に依存しないようにするためには、試験システム１００において、モジュール１０８が実装するピン１２８そのものを定義できるようにし、それをもとにモジュール１０８の制御ができるようにしな

ければならない。本試験システム１００では、これはモジュール設定ファイル（Ｍｏｄｕｌｅ Ｃｏｎｆｉｇｕｒａｔｉｏｎ Ｆｉｌｅ。以下「ＭＣＦ」という。）１３０によって達成される。

【００３７】

ＭＣＦ１３０には、モジュール１０８のリソースとロードボード１１４のコネクタピン１２４との接続関係が指定される。これにより、論理的なピンの表現であるリソースと、モジュール１０８の物理ピン１２８、さらにはそれが接続されているロードボード１１４のコネクタピン１２４との関係が定められる。ＭＣＦ１３０は、システムだけが設定可能であることが望ましい。

【００３８】

なお、本実施形態において、モジュール１０８の機能は、主にモジュール１０８の全体的な制御に関わる機能を提供するモジュール・オブジェクトと、モジュールのピン毎の機能を提供するリソース・オブジェクトによって表現される。本実施形態に係る試験システム１００では、これらは全て、ＭＣＦ１３０に記述される内容に従って、実際のハードウェアのエンティティと対称に結び付けられる。なお、これらのソフトウェアのオブジェクトを提供するモジュール・ソフトウェアを、ここではモジュール・ドライバと呼ぶ。

【００３９】

また、本実施形態において、リソースとは、試験システム１００のモジュール１０８が備える物理的なピン１２８とその機能とを、オブジェクトで抽象化して表現したものである。モジュール１０８の一つの物理ピン１２８は、一つのリソースとして表現されてもよいし、機能的に分割された複数のリソースとして表現されてもよい。

【００４０】

リソースの機能的な分類はリソース・タイプと呼ばれ、その機能の定義はモジュール１０８のベンダが提供するリソース定義ファイルによって示される。論理的なピンを指すリソースは、リソース・タイプと、モジュール１０８内でピン１２８を特定する番号（リソースＩＤ）と、試験システム１００内でモジュール１０８を特定するバス１０７のポート番号との３つ組で表現される。リソース・タイプとリソースＩＤの組と、それに対応するモジュール１０８内の物理ピン１２８の関係は固定的であるため、リソースが与えられたときに、それを物理的なピン１２８に対応付けることが可能となる。なお、リソースが物理ピン１２８と対応付けられることはアーキテクチャ上必須ではなく、モジュール１０８の持つ機能を仮想的なピンに見立てて表現し、制御することも可能である。

【００４１】

図３は、試験システム１００のソフトウェア・アーキテクチャ２００の一例を示す。ソフトウェア・アーキテクチャ２００は分散オペレーティングシステムを表しており、関連するハードウェアシステム要素１０２、１０４、１０８に対応する、システムコントローラ・ソフトウェア２２０（以下、単に「システムコントローラ２２０」ともいう。）、少なくとも１つのサイトコントローラ・ソフトウェア２４０（以下、単に「サイトコントローラ２４０」ともいう。）及び少なくとも１つのモジュールを含むテストヘッド・ソフトウェア２６０（以下、単に「テストヘッド２６０」ともいう。）のための構成要素を有する。

【００４２】

一つの例示的な選択として、試験システム１００は、ソフトウェアのプラットフォームとしてＭｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ（登録商標）を使用し、実装言語としてＡＮＳＩ／ＩＳＯ標準Ｃ＋＋を使用することができる。プラットフォームの全てのシステム・インタフェースもまた、Ｃ＋＋のクラスまたはインタフェースとして提供され、ユーザの試験プログラムやモジュール・ソフトウェアはＣ＋＋で実装することができる。

【００４３】

システムコントローラ２２０は、ユーザのための一次的なインタラクションポイントであり、サイトコントローラ２４０へのゲートウェイと、マルチサイト／ＤＵＴ環境におけるサイトコントローラ２４０の同期とを提供する。ユーザアプリケーション及びツールは

、グラフィカルユーザインターフェース（ＧＵＩ）に基づくものでも、そうでないものでも、システムコントローラ２２０上で実行される。

【００４４】

また、システムコントローラ２２０は、テストプラン、試験パターン及び試験パラメータファイルを含む、全てのテストプラン関連情報のためのリポジトリとしての役割も果たす。これらのファイルを記憶するメモリは、システムコントローラ２２０に対してローカルに存在することができるか、又は、ネットワークを介してシステムコントローラ２２０に接続されることができる。

【００４５】

さらに、システムコントローラ２２０は、フレームワーククラス２２１と、システムツール２２２と、外部ツール２２３と、サイトコントローラへの標準インタフェース２２４と、通信ライブラリ２３０とを含む。

【００４６】

システムコントローラ２２０に関連するフレームワーククラス２２１は、オブジェクトと対話するための仕組みを提供し、標準インタフェース２２４の参照インプリメンテーションを提供する。また、サイトコントローラ２４０へのゲートウェイを提供し、且つマルチサイト／ＤＵＴ環境におけるサイトコントローラ２４０の同期を提供するソフトウェア構成要素を構成する。実効的には、フレームワーククラス２２１は、システムコントローラ２２０をサポートするＯＳとしての役割を果たすことができる。

【００４７】

第三者の開発者は、標準システムツール２２２に加えて、又は、標準システムツール２２２に換えて、一以上のツール２２３を提供することができる。システムコントローラ２２０上の標準インタフェース２２４は、テスタ及び試験オブジェクトにアクセスするためにそれらのツールが用いるインタフェースを含む。ツール（アプリケーション）２２２、２２３によって、テスタ及び試験オブジェクトをインタラクティブに、且つバッチで制御できる。また、標準インタフェース２２４は、システムコントローラ２２０上で実行されるフレームワークオブジェクトへのオープンインタフェースや、サイトコントローラ２４０に基づくモジュール・ソフトウェアがパターンデータにアクセスし、それらを検索できるようにするインタフェース等を含む。

【００４８】

システムコントローラ２２０上に存在する通信ライブラリ２３０は、ユーザアプリケーション及び試験プログラムに対してトランスペアレントであるように、サイトコントローラ２４０と通信するための仕組みを提供する。

【００４９】

サイトコントローラ２４０は、試験機能の大部分が提供される。サイトコントローラ２４０は、テストプラン（試験プログラム）２４１と、試験クラス２４２と、サイトコントローラフレームワーククラス２４３と、特定モジュール拡張インタフェース２４４と、標準インタフェース２４５と、モジュール・ソフトウェア・インプリメンテーション２４６と、バックプレーン通信ライブラリ２４７と、バックプレーンドライバ２４８とを含む。

【００５０】

テストプラン２４１はユーザによって記述される。テストプランプログラムは、Ｃ＋＋のようなオブジェクト指向コンポーネントを用いて、標準的なコンピュータ言語において直に記述するか、又は、Ｃ＋＋コードを生成するための、さらに高いレベルの試験プログラミング言語において記述することができ、後に実行可能な試験プログラムにコンパイルすることができる。

【００５１】

テストプラン２４１は、標準的な又はユーザによって供給される試験クラス２４２及び／又はそのサイトコントローラ２４０に関連するフレームワーククラス２４３を用いることにより試験オブジェクトを作成し、標準インタフェース２４５を用いてハードウェアを構成し、テストプランフローを規定する。テストプランはいくつかの基本的なサービスを

サポートし、デバッグサービス（たとえば、ブレークポイント生成）のような、下層のオブジェクトのサービスへのインタフェースを提供し、下層のフレームワーク及び標準クラスにアクセスできるようにする。

【００５２】

試験クラス２４２は、標準試験インタフェースの１つのインプリメンテーションであり、テストプランプログラムにおいて指定される。各試験クラスは通常、特定のタイプのデバイス試験、又はデバイス試験のための設定を実装する。

【００５３】

フレームワーククラス２４３は、共通の試験関連動作を実装する１組のクラス及びメソッドである。フレームワーククラス２４３は、実効的には、各サイトコントローラ２４０をサポートするローカルＯＳとしての役割を果たすことができる。

【００５４】

モジュール・ソフトウェアは、テストの実行時に必要に応じて動的に試験システム１００のプロセスにロードできるように、ＤＬＬ（Ｄｙｎａｍｉｃ　Ｌｉｎｋ　Ｌｉｂｒａｒｙ）形式であることが望ましい。これは、テストプラン２４１が指定する試験サイト１１０の構成に応じて、試験システム１００が実行時に動的に制御対象のモジュール２６１を決定するからである。また、全てのモジュール・ソフトウェアは、モジュール２６１の機能に応じて、システムＯＳが規定するモジュール・ソフトウェアの標準インタフェース２４５を実装することが求められる。

【００５５】

特定モジュール拡張インタフェース２４４は、必要に応じてモジュール固有のより複雑で専門的な機能を持つインタフェース階層をモジュール・ソフトウェアに付加して実装する。例えば、Ｃ＋＋では、標準インタフェース２４５を継承するインタフェースクラスをモジュール・ソフトウェアで定義することによって、インタフェースを拡張することができる。

【００５６】

標準インタフェース２４５は、システムフレームワークが規定する必要最小限の、一般的で普遍的に適用可能なインタフェースである。標準インタフェース２４５を用いて、全てのオブジェクトを画一的に扱うことができる。これにより、システムＯＳを変更することなく、新しいモジュールのソフトウェアをシームレスに、プラグ・アンド・プレイ方式でシステムに導入することが可能となる。

【００５７】

一例として、標準インタフェース２４５は、Ｃ＋＋の純粋仮想インタフェースクラスとして定義される。このとき、ユーザ向けの機能を定義するサブクラスと、システムだけが使用する機能を定義するサブクラスと、リソースの機能を定義するサブクラスとによって構成されることが好ましい。また、標準インタフェース２４５の階層は、モジュールの最も基本的な機能を定義する階層、パターン・プログラムを実行する機能を持つモジュールを表現する階層、複数のピン間で共有されるテスト周期という概念を導入する階層、デジタル・モジュール特有の機能を加える階層、の４階層により構成されることが好ましい。このとき、各モジュール・ドライバは、モジュールの機能に応じて、４階層のインタフェースのいずれかを実装する。しかし、標準インタフェース２４５の構成は、これらの構成に限定されるものではない。

【００５８】

バックプレーン通信ライブラリ２４７は、バックプレーンにわたる標準的な通信のためのインタフェースを提供し、それによりテストヘッド２６０内のモジュール１０８と通信するために必要な機能を提供する。これにより、ベンダ固有のモジュール・ソフトウェアが、バックプレーンドライバ２４８を用いて、対応するモジュール１０８と通信できる。バックプレーン通信プロトコルはパケットに基づくフォーマットを用いることができる。

【００５９】

テストヘッド２６０は、デバイスに対する測定機能が提供される。テストヘッド２６０

は、モジュール２６１と、ＴＩＵ２６２と、ロードボード２６３と、ＤＵＴ２６４とを含む。

【００６０】

また、ソフトウェア・アーキテクチャ２００は、ソフトウェアの名目的な供給元に基づいて、システムフレームワーク２９０、ユーザコンポーネント２９２、テスタオペレーティングシステム２９４、モジュール・ハードウェア・ベンダコンポーネント２９６、及び、ツール・ソフトウェア・ベンダコンポーネント２９８に分類することができる。

【００６１】

システムフレームワーク２９０は、試験システム１００の開発ベンダにより供給され、フレームワーククラス２２１と、標準インタフェース２２４と、フレームワーククラス２４３と、標準インタフェース２４５と、バックプレーン通信ライブラリ２４７とを含む。

【００６２】

ユーザコンポーネント２９２は、試験を実施するユーザによって供給され、テストプラン２４１と、テストクラス２４２と、ロードボード２６３と、ＤＵＴ２６４とを含む。

【００６３】

テスタオペレーティングシステム２９４は、基本的な接続性及び通信のためのソフトウェアインフラストラクチャとして供給され、システムツール２２２と、通信ライブラリ２３０と、バックプレーンドライバ２４８とを含む。

【００６４】

モジュール・ハードウェア・ベンダコンポーネント２９６は、モジュール１０８の開発元によって供給され、特定モジュール拡張インタフェース２４４と、モジュール・ソフトウェア・インプリメンテーション２４６と、モジュール２６１と、ＴＩＵ２６２を含む。

【００６５】

ツール・ソフトウェア・ベンダコンポーネント２９８は、外部ツールの開発元によって供給され、外部ツール２２３を含む。

【００６６】

次に、以上のように構成される試験システム１００における、テストクラスのデバッグ支援機能について説明する。

【００６７】

図４は、デバッグ処理の観点からみて、試験システム１００のソフトウェア構成を大きく３つに分けて示したブロック図である。同図に示すように、試験システム１００のソフトウェアの構成は、ツール（ＧＵＩツール）１０、試験プロセス２０、テストクラス３０に分けることができる。

【００６８】

ツール１０は、デバイス１１２の試験に関わる作業で利用される種々の処理の関数（ＡＰＩ）である。ユーザは、このツール１０を通じて、試験システム１００をコントロールする。また、デバイス試験の条件や結果を、ＧＵＩ上で確認することもできる。

【００６９】

試験プロセス２０は、試験システム１００のシステム・ソフトウェアの中核となるプロセスである。試験プロセス２０は、サイトコントローラ１０４上で動作し、デバイス試験全体の動作をまとめ、また、モジュール１０８等のデバイス試験のためのハードウェアを動作させる。

【００７０】

テストクラス３０は、試験システム１００のユーザが作成する試験プログラムであり、デバイス試験の方法、アルゴリズム等を、Ｃ＋＋言語のような所定のプログラミング言語のクラスで記述する。ユーザが作成したテストクラス３０は、コンパイルされた後、試験プロセス内にロードされ、試験プロセス内で動作する。

【００７１】

図５は、サイトコントローラ１０４等の上で動作する試験プロセス２０の構成を示す図である。図５に示すように、本発明における試験プロセス２０は、少なくとも、次の２つ

のスレッド１１，３１を備える。１つは、ツール１０による処理を行うためのスレッド（以下「ツール・スレッド」という。）１１であり、もう１つは、テストクラス３０を実行するためのスレッド（以下「テストクラス・スレッド」という。）３１である。このように、本試験プロセス２０では、ツール１０による処理の実行と、テストクラス３０の実行とが、それぞれ別のスレッド１１，３１で行われるようになっている。

【００７２】

そして、本発明においては、２つのスレッド１１，３１の動作を個別に細かく制御することによって、試験プロセス２０を、少なくとも、通常実行状態４１、デバッギング状態４２、及び、ツール使用可能状態４３の３つの状態で動作させるようにしている。

【００７３】

通常実行状態４１は、ツール・スレッド１１とテストクラス・スレッド３１の両スレッドが共に動作するよう制御される状態である。この状態では、スレッド１１，３１の両方が動作するため、ツール１０の要求処理も、テストクラス３０の実行も、ともに行われる。すなわち、従来のデバッガによる通常のプロセス実行状態に相当する。

【００７４】

デバッギング状態４２は、ツール・スレッド１１とテストクラス・スレッド３１の両スレッドが共に停止するよう制御される状態である。この状態では、スレッド１１，３１の両方とも停止してしまうため、ツール１０の要求処理は実行されず、またテストクラス３０の実行も停止してしまう。すなわち、従来のデバッガによるブレーク状態に相当する。

【００７５】

ツール使用可能状態４３は、本発明に特有の状態であり、テストクラス・スレッド３１のみを停止状態で凍結させ、ツール・スレッド１１を動作させるよう制御される状態である。これにより、テストクラス３０の実行を停止したままの状態で、ツール１０を動作させることが可能になる。つまり、ツール使用可能状態４３では擬似的なブレーク状態が実現され、一見ブレーク状態のように見えるが、ツール１０を使用することができるようになる。

【００７６】

本実施形態では、デバッガ４０の提供するプロセスの停止及び再開機能とフリーズ機能とを用いて、通常実行状態４１、デバッギング状態４２、及び、ツール使用可能状態４３の３つの状態の遷移を実現している。ここで、プロセスの停止及び再開機能とは、プロセスの動作を停止及び再開、すなわち、複数の全スレッドの動作を停止及び再開させる処理を行うものである。例えば、ブレークポイント機能は、このプロセスの停止及び再開機能に含まれる。また、フリーズ機能とは、複数のスレッドのうち所定のスレッドの状態及び動作のみを個別に一時的に凍結させる処理を行うものである。そして、フリーズを解除することにより、スレッドの動作が再開される。つまり、凍結したスレッドはスレッドの状態及び動作がフリーズ前の状態で凍結され、フリーズが解除されるまで、当該スレッドの処理が止まる。また、フリーズ機能によるスレッドの凍結状態は、スレッドの停止及び再開機能よりも優先される。このため、フリーズ機能によって停止状態で凍結されたスレッドは、フリーズが解除されるまで、たとえプロセスの停止及び再開機能が利用されたとしても、当該スレッドは停止したままの状態が維持される。なお、プロセスの停止及び再開機能は、複数のスレッドの状態を全て実行又は停止させることはできるが、各スレッドの状態を個別に制御することはできない。この点でも、フリーズ機能とプロセスの停止及び再開機能とは相違する。

【００７７】

まず、通常実行状態４１とデバッギング状態４２との間の遷移は、プロセスの停止及び再開機能を利用して、ブレーク状態（全スレッドが停止している状態）への移行及び解除を行う。例えば、デバッガ４０は、ユーザの設定したブレークポイントで、試験プロセス２０をブレーク状態に遷移、すなわち、通常実行状態４１からデバッギング状態４２に遷移する。また、ユーザがプロセス再開機能によりブレーク状態を解除すれば、デバッギング状態４２から通常実行状態４１に遷移する。ユーザは、デバッギング状態４２でのみ、

デバッガ４０を用いてソースコードを行単位でデバッグすることができる。しかし、デバッギング状態４２では、全てのスレッドが停止しており、ツール・スレッド１１も停止しているため、ツール１０を使用することはできない。

【００７８】

テストクラス３０の実行が停止している状態でツール１０を使用するためには、デバッギング状態４２からツール使用可能状態４３に遷移する必要がある。本実施形態では、ツール使用可能状態４３を実現するため、すなわち、複数のスレッドのうち、少なくともテストクラス・スレッド３１のみを停止させるために、デバッガ４０が提供するフリーズ機能を利用する。

【００７９】

本実施形態において、デバッギング状態４２からツール使用可能状態４３に遷移するとき、デバッガ４０は、フリーズ機能によりテストクラス・スレッド３１を停止状態で凍結してから、プロセス再開機能により、試験プロセス２０の停止を解除する。これにより、フリーズしているテストクラス・スレッド３１以外のスレッドは動作する状態になるが、テストクラス・スレッド３１は停止状態で凍結されるため、停止状態が維持されたままになる。つまり、フリーズ機能を用いない場合であれば、通常実行状態４１に遷移するところ、テストクラス・スレッド３１はフリーズ機能によって凍結しているので、ツール使用可能状態４３に遷移する。

【００８０】

このツール使用可能状態４３では、ツール１０は実行可能であるが、テストクラス３０は停止したままであり、ユーザがデバッグしているコードの位置も保存される。

【００８１】

また、ツール使用可能状態４３から、デバッギング状態４２に遷移するとき、デバッガ４０は、プロセス停止機能によって試験プロセス２０を停止する。このとき、テストクラス・スレッド３１は凍結状態が維持されたままの状態であるが、テストクラス・スレッド３１以外の全てのスレッドの動作は停止する。その後、テストクラス・スレッド３１のフリーズを解除する。これにより、テストクラス・スレッド３１は凍結状態が解除され、通常の停止状態に戻る。これにより、ツール・スレッド１１及びテストクラス・スレッド３１を含む全てのスレッドが停止状態になる。こうして、ツール使用可能状態４３からデバッギング状態４２に戻ることにより、ユーザがデバッグしているコードの位置を保存したまま（すなわち、テストクラスの実行を停止したまま）、デバッガ４０によるテストクラス３０のデバッグを再開できる。

【００８２】

図６は、ユーザの操作に連動してデバッグの状態を自動的に切り替える処理を示す図である。本実施形態では、ユーザが作成したテストクラスを、システムコントローラ１０２等のデバッガを用いてデバッグする。このテストクラスのデバッグ用の表示画面において、デバッガ４０用のウィンドウ（以下「デバッガ・ウィンドウ」という。）５１とツール１０を操作するためのウィンドウ（以下「ツール・ウィンドウ」という。）５２間でウィンドウのフォーカスが移動するのに合わせて、デバッギング状態４２とツール使用可能状態４３との間で状態を遷移させる。

【００８３】

本実施形態では、デバッガ４０が実行されるシステムコントローラ１０２等の制御により、図６に示すように、デバッガ・ウィンドウ５１がアクティブになったときは、デバッギング状態４２に状態遷移され、デバッガ４０の使用が可能になる。また、ツール・ウィンドウ５２がアクティブになったときは、ツール使用可能状態４３に状態遷移され、ツール１０の使用が可能になる。

【００８４】

例えば、まず、試験プロセス２０がブレークポイントでブレーク等すると、通常実行状態４１からデバッギング状態４２に遷移する。このとき、試験プロセス２０はブレーク状態になるが、システムコントローラ１０２等の制御により、デバッガ・ウィンドウ５１が

アクティブになる。そして、ユーザは必要に応じてデバッグ処理を行う。

【００８５】

デバッグ処理中に、ユーザがツール・ウィンドウ５２をアクティブにすると、デバッギング状態４２からツール使用可能状態４３に自動的に状態遷移し、テストクラス３０の実行が停止した状態で、ユーザはツール１０を使用できるようになる。さらに、ツールを使用した後などに、ユーザが、デバッガ・ウィンドウ５１をアクティブにするとツール使用可能状態４３からデバッギング状態４２に自動的に状態遷移し、ブレーク状態になる。このとき、ユーザはツール１０を使用することはできないが、テストクラス３０のデバッグ処理を行うことができる。

【００８６】

このように、ユーザがフォーカスしたウィンドウ（アクティブ・ウィンドウ）によって、自動的にプロセスの状態を変更させることによって、ユーザは、デバッグ中の状態遷移を意識することなく、テストクラス３０のデバッグ及びデバッグ中のツール１０の使用を直感的に操作することができるようになる。

【００８７】

このように、本発明では、試験プロセス２０内で、ツール１０の処理を実行するスレッド１１と、テストクラス３０を実行するスレッド３１を、それぞれ別スレッドとして構成している。そして、各スレッドの状態を個別に制御することにより、試験プロセス２０の実行状態として、通常実行状態４１、デバッギング状態４２、及び、ツール使用可能状態４３の３つの状態を実現している。特に、ツール使用可能状態４３では、テストクラス・スレッド３１のみを停止させて、ツール・スレッド１１を動作させている。これにより、テストクラス３０のデバッグ中に、ツール１０を利用できるようになる。

【００８８】

また、デバッグ中のユーザの操作に連動して、自動的にデバッギング状態４２とツール使用可能状態４３との間の状態を変更するデバッグ支援機能を備えている。これにより、ユーザはデバッグの状態を意識することなく、デバッグ処理に集中できるようになる。

【００８９】

さらに、本発明では、ツール１０の処理の実行とテストクラス３０の実行とをそれぞれ別スレッドとして構成して、個別にスレッド制御を行うことにより、ツール使用可能状態４３を実現している。このため、ユーザの作成するテストクラスには、何ら変更を要しないし、ツール１０自体の変更も不要である。したがって、本発明には、過去のプログラム資産やツールを、特段変更することなく、そのまま活用可能である。

【００９０】

なお、本発明は、上記した実施の形態に限定されるものではなく、本発明の要旨を逸脱しない範囲内において、他の様々な形で実施することができる。このため、上記実施形態はあらゆる点で単なる例示にすぎず、限定的に解釈されるものではない。例えば、上述の各処理ステップは処理内容に矛盾を生じない範囲で任意に順番を変更して又は並列に実行することができる。

【図面の簡単な説明】

【００９１】

【図１】本発明の一実施形態による試験システム１００のシステムアーキテクチャを示す。

【図２】試験サイト１１０及びロードボード１１４のハードウェアの概略構成と各種設定用ファイルとの関係の一例を示す図である。

【図３】試験システム１００のソフトウェア・アーキテクチャ２００の一例を示す。

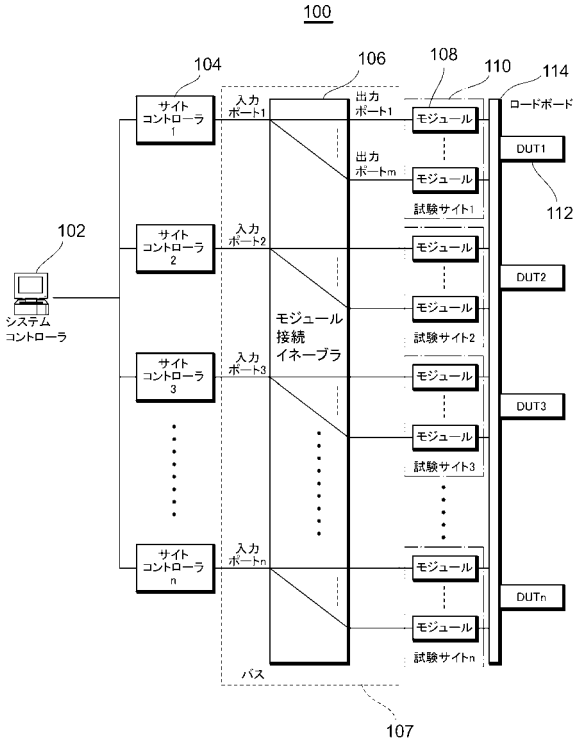
【図４】デバッグ処理の観点からみて、試験システム１００のソフトウェア構成を大きく３つに分けて示したブロック図である。

【図５】サイトコントローラ１０４等の上で動作する試験プロセス２０の構成を示す図である。

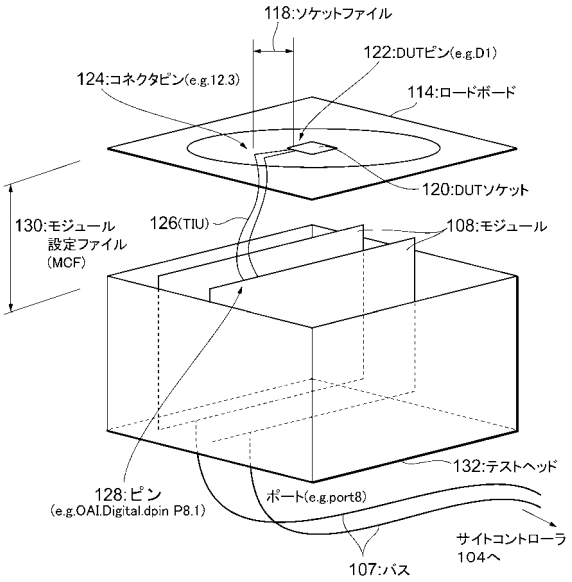
【図６】ユーザの操作に連動してデバッグの状態を自動的に切り替える処理を示す図であ

る。
【図７】従来のＡＴＥにおけるテストクラスのデバッグ処理の概略を示す図である。
【符号の説明】
【００９２】
１０　ツール
１１　ツール・スレッド
２０　試験プロセス
３０　テストクラス
３１　テストクラス・スレッド
４０　デバッガ
４１　通常実行状態
４２　デバッギング状態
４３　ツール使用可能状態
５１　デバッガ・ウィンドウ
５２　ツール・ウィンドウ
１００ 試験システム
１０２ システムコントローラ
１０４ サイトコントローラ
１０６ モジュール接続イネーブラ
１０７ バス
１０８ モジュール
１１０ 試験サイト
１１２ 被試験デバイス（ＤＵＴ）
１１４ ロードボード
１１８ ソケットファイル
１２０ ソケット
１２２ ピン
１２４ コネクタピン
１２６ テスタ・インタフェース・ユニット（ＴＩＵ）
１２８ ピン
１３０ モジュール設定ファイル（ＭＣＦ）
１３２ テストヘッド

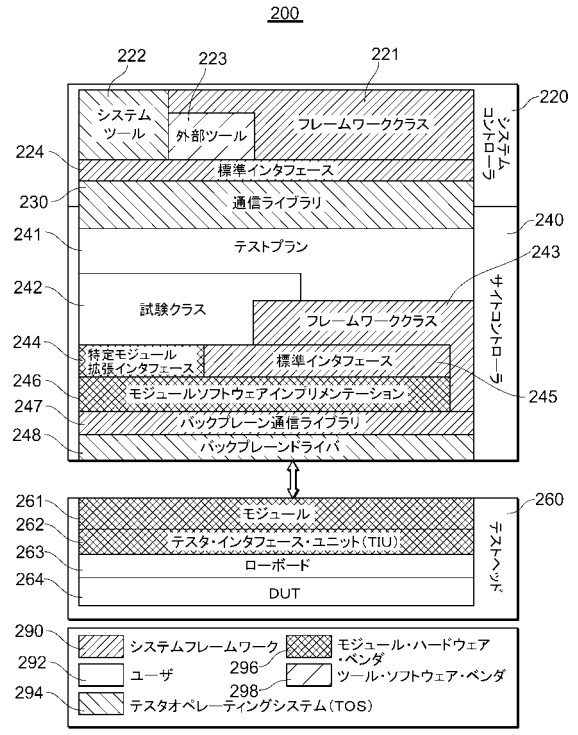
【図１】


100
104
106
108
110
114
サイト
コントローラ
1
入力
ポート1
出力
ポート1
モジュール
ロードボード
出力
ポートm
DUT1
112
試験サイト1
102
サイト
コントローラ
2
入力
ポート2
DUT2
システム
コントローラ
モジュール
接続
イネーブラ
試験サイト2
サイト
コントローラ
3
入力
ポート3
DUT3
試験サイト3
サイト
コントローラ
n
入力
ポートn
DUTn
試験サイトn
バス
107

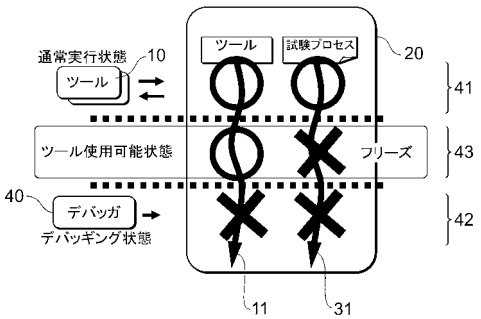
【図２】


118:ソケットファイル
122:DUTピン(e.g.D1)
124:コネクタピン(e.g.12.3)
114:ロードボード
120:DUTソケット
130:モジュール
設定ファイル
(MCF)
126(TIU)
108:モジュール
132:テストヘッド
128:ピン
(e.g.OAI.Digital.dpin P8.1)
ポート(e.g.port8)
サイトコントローラ
104へ
107:バス

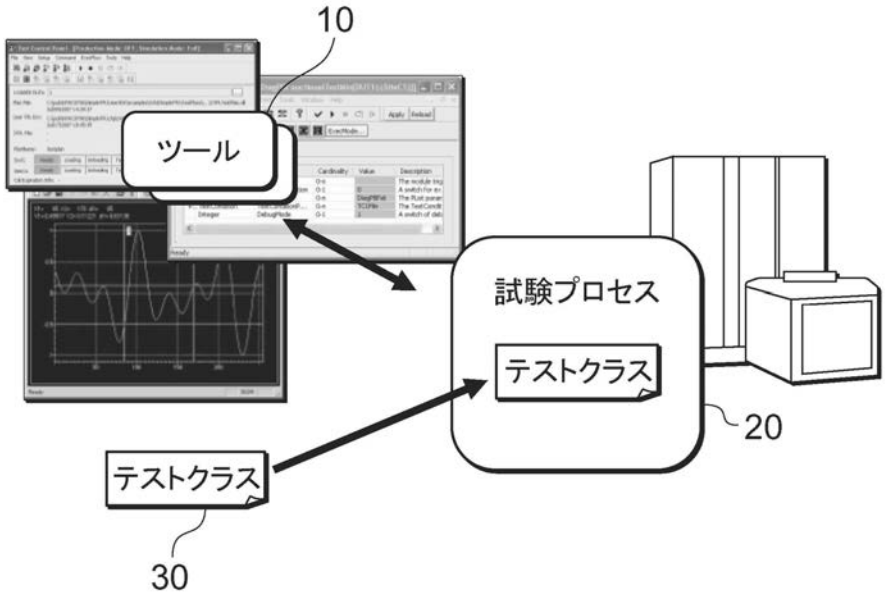
【図３】


200
222
223
221
220
システム
ツール
外部ツール
フレームワーククラス
システム
コントローラ
224
標準インタフェース
230
通信ライブラリ
240
241
テストプラン
243
242
試験クラス
フレームワーククラス
サイトコントローラ
244
特定モジュール
拡張インタフェース
標準インタフェース
246
モジュールソフトウェアインプリメンテーション
245
247
バックプレーン通信ライブラリ
248
バックプレーンドライバ
261
モジュール
260
262
テスタ・インタフェース・ユニット(TIU)
263
ローボード
テストヘッド
264
DUT
290
システムフレームワーク
モジュール・ハードウェア
・ベンダ
292
ユーザ
296
ツール・ソフトウェア・ベンダ
298
294
テスタオペレーティングシステム(TOS)

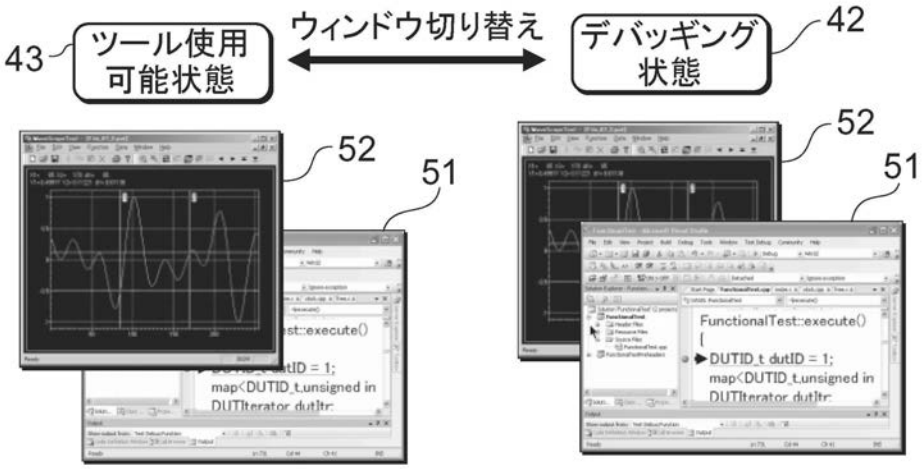
【図５】


通常実行状態
10
ツール
ツール
試験プロセス
20
41
ツール使用可能状態
フリーズ
43
40
デバッガ
デバッギング状態
42
11
31

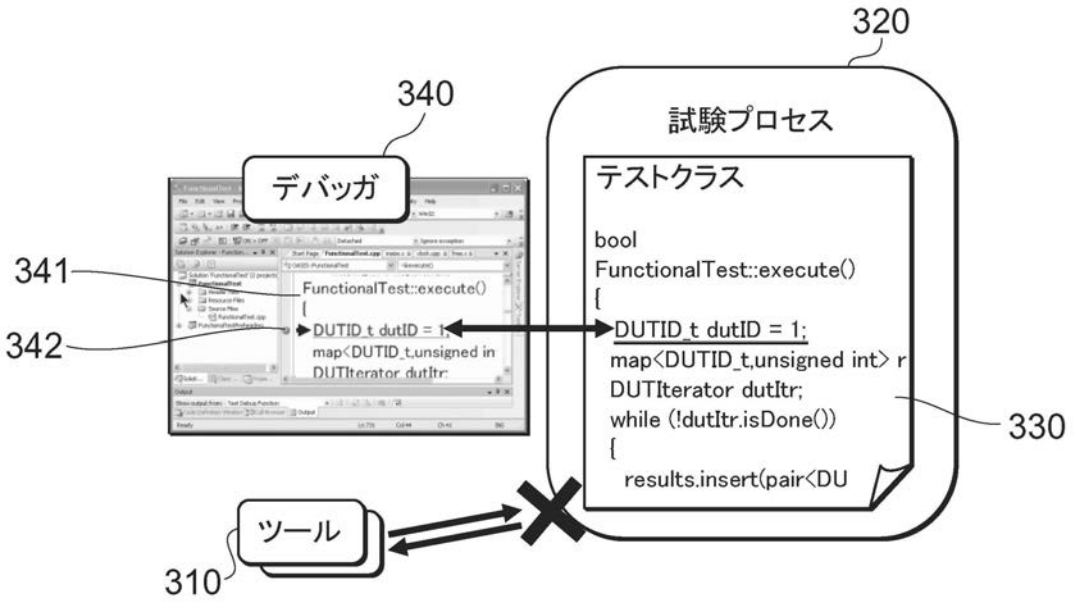
【図４】


10
ツール
試験プロセス
テストクラス
20
テストクラス
30

【図６】


43
ツール使用
可能状態
ウィンドウ切り替え
デバッギング
状態
42
52
51
52
51
FunctionalTest::execute()
DUTID_t dutID = 1;
map<DUTID_t,unsigned in
DUTIterator dutItr;

【図７】


320
340
試験プロセス
デバッガ
テストクラス
341
342
FunctionalTest::execute()
{
DUTID_t dutID = 1;
map<DUTID_t,unsigned in
DUTIterator dutItr;
bool
FunctionalTest::execute()
{
DUTID_t dutID = 1;
map<DUTID_t,unsigned int> r
DUTIterator dutItr;
while (!dutItr.isDone())
{
results.insert(pair<DU
330
ツール
310

フロントページの続き

(72)発明者　志村　卓美
東京都練馬区旭町１－３２－１　株式会社アドバンテスト内

審査官　多賀　実

(56)参考文献　特表２００６－５２０９４７（ＪＰ，Ａ）
特開２００４－３１８６５８（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ１１／２２
Ｇ０６Ｆ１１／２８－１１／３４
Ｇ０１Ｒ３１／２８