【公報種別】特許法第１７条の２の規定による補正の掲載
【部門区分】第６部門第１区分
【発行日】平成18年9月21日(2006.9.21)

【公表番号】特表2006-518460(P2006-518460A)
【公表日】平成18年8月10日(2006.8.10)
【年通号数】公開・登録公報2006-031
【出願番号】特願2006-502669(P2006-502669)
【国際特許分類】
Ｇ０１Ｒ　31/28　　(2006.01)
【ＦＩ】
Ｇ０１Ｒ　31/28　　Ｈ

【誤訳訂正書】
【提出日】平成18年6月8日(2006.6.8)
【誤訳訂正１】
【訂正対象書類名】特許請求の範囲
【訂正対象項目名】全文
【訂正方法】変更
【訂正の内容】
【特許請求の範囲】
【請求項１】
少なくとも一つのテスト対象装置（DUT）をテストする半導体テストシステムのための分散オペレーティングシステムであって、
システムコントローラによる少なくとも一つのサイトコントローラの制御を可能にするホストオペレーティングシステムと、
各サイトコントローラと関連しており、関連するサイトコントローラによる少なくとも一つのテストモジュールの制御を可能にする少なくとも一つのローカルオペレーティングシステムとを備えており、
少なくとも一つのテストモジュールが対応するDUT上でテストを行う分散オペレーティングシステム。
【請求項２】
前記ホストオペレーティングシステムは、前記少なくとも一つのサイトコントローラの動作を同期させる請求項１に記載の分散オペレーティングシステム。
【請求項３】
前記ホストオペレーティングシステムは、前記システムコントローラと前記少なくとも一つのサイトコントローラとの間の通信を仲裁する請求項１に記載の分散オペレーティングシステム。
【請求項４】
前記システムコントローラは、前記少なくとも一つのサイトコントローラの動作をモニタする請求項１に記載の分散オペレーティングシステム。
【請求項５】
サイトコントローラは、当該サイトコントローラに関連する前記少なくとも一つのテストモジュールの動作をモニタする請求項1に記載の分散オペレーティングシステム。
【請求項６】
前記ホストオペレーティングシステムは、前記少なくとも一つのサイトコントローラと通信するための少なくとも一つのホストインタフェースを備えている請求項１に記載の分散オペレーティングシステム。
【請求項７】
前記少なくとも一つのホストインタフェースは、サイトコントローラに関連している少

なくとも一つのテストモジュールと通信するためのものである請求項６に記載の分散オペレーティングシステム。
【請求項８】
前記分散オペレーティングシステムは、サイトコントローラを第一のテストモジュールにインタフェース接続するためのテストモジュール機能を規定するテストモジュールインタフェースをさらに備えており、前記テストモジュールインタフェースは、前記サイトコントローラを第二のテストモジュールにインタフェース接続するように拡張可能であり、前記拡張されていないテストモジュールインタフェースは、前記サイトコントローラを前記第二のテストモジュールにインタフェース接続するには不十分である、請求項１に記載の分散オペレーティングシステム。
【請求項９】
前記ホストオペレーティングシステムは、少なくとも一つのホストフレームワーククラスを有している請求項１に記載の分散オペレーティングシステム。
【請求項１０】
前記少なくとも一つのフレームワーククラスは、前記少なくとも一つのサイトコントローラを制御するためのアプリケーション特有のクラスをユーザが開発することを可能にするように標準的なコンピュータ言語で開発される請求項１に記載の分散オペレーティングシステム。
【請求項１１】
前記標準的なコンピュータ言語はCまたはC++である請求項１０に記載の分散オペレーティングシステム。
【請求項１２】
各ローカルオペレーティングシステムは、少なくとも一つのローカルフレームワーククラスを有している請求項１に記載の分散オペレーティングシステム。
【請求項１３】
前記少なくとも一つのローカルフレームワーククラスは、前記少なくとも一つのテストモジュールを制御するためのアプリケーション特有のクラスをユーザが開発することを可能にするように標準的なコンピュータ言語で開発される請求項１２に記載の分散オペレーティングシステム。
【請求項１４】
前記標準的なコンピュータ言語はCまたはC++である請求項１３に記載の分散オペレーティングシステム。
【請求項１５】
各サイトコントローラによって制御されるモジュールの数は拡張可能である請求項１に記載の分散オペレーティングシステム。
【請求項１６】
対応するサイトコントローラに関連している前記ローカルオペレーティングシステムは、前記サイトコントローラによって制御されるテストモジュールのタイプの再構成を可能にする請求項１に記載の分散オペレーティングシステム。
【請求項１７】
前記ホストオペレーティングシステムは、前記システムコントローラによって制御されるサイトコントローラの数を拡張可能とする請求項１に記載の分散オペレーティングシステム。
【請求項１８】
前記ホストオペレーティングシステムは、前記テストシステムによってテストされるDUTの数を拡張可能とする請求項１に記載の分散オペレーティングシステム。
【請求項１９】
前記少なくとも一つのテストモジュールは、ハードウェアおよび／またはソフトウェアを備えている請求項１に記載の分散オペレーティングシステム。
【請求項２０】

対象テストモジュールを前記テストシステムとともに使用することをシミュレートして、前記対象モジュールが前記テストシステムと互換性があることを検証するエミュレータをさらに備えている請求項１に記載の分散オペレーティングシステム。
【請求項２１】
第一のテストサイトにおける第一のセットのモジュールは、第二のテストサイトにおける第二のセットのモジュールとは異なって構成されている請求項１に記載の分散オペレーティングシステム。
【請求項２２】
第一のテストサイトは第一のDUTをテストするための第一の構成を有しており、第二のテストサイトは第二のDUTをテストするための第二の構成を有しており、前記第一および第二のテストサイトは、代わりに第三のDUTをテストするための第三のテストサイトをともに形成するように再構成可能である請求項１に記載の分散オペレーティングシステム。
【請求項２３】
第一のテストサイトにおける第一のモジュールは、第二のテストサイトにおける第二のモジュールにアクセスすることができる請求項１に記載の分散オペレーティングシステム。
【請求項２４】
テストモジュールとともに使用するための所定のセットの関数およびインタフェースを有している通信ライブラリをさらに備えている請求項１に記載の分散オペレーティングシステム。

【誤訳訂正２】
【訂正対象書類名】明細書
【訂正対象項目名】全文
【訂正方法】変更
【訂正の内容】
【発明の詳細な説明】
【発明の名称】集積回路をテストする方法および装置
【技術分野】
【０００１】
本出願は、2003年2月24日に出願された出願第60/449,622号「集積回路をテストする方法および装置」、2003年2月14日に出願された出願第60/447,839号「半導体集積回路用のテストプログラムを開発する方法および構造」、2003年3月31日に出願された米国出願第10/404,002号「テストエミュレータ、テストモジュールエミュレータおよびプログラムを記憶する記録媒体」、および2003年3月31日に出願された米国出願第10/403,817号「テスト装置およびテスト方法」の恩恵を受けることを主張する。これらの出願は全体としてここに援用される。また本出願は、2003年2月14日に出願された出願第60/447,839号「半導体集積回路用のテストプログラムを開発する方法および構成」の恩恵を受けることを主張する同時に出願された米国出願第＿＿＿＿＿号「半導体集積回路用のテストプログラムを開発する方法および構成」も全体的に援用する。
【０００２】
本発明は、集積回路（IC）のテストに関連する。特に本発明は、一つ以上のICをテストする自動テスト機器（ATE）に関連する。
【背景技術】
【０００３】
システムオンチップ（SOC）装置がより複雑になりチップのテストのコストを削減する要求が出てくるにつれて、IC製造者およびテスタのベンダはともに、どのようにICのテストを行うべきかを考え直すことを余儀なくされている。産業の研究によると、リエンジニアリングをしなければ、テスタの予想されるコストは、しばらくは劇的に上昇し続けるであろう。

【０００４】
テスト機器のコストが高い主な理由には、従来のテスタアーキテクチャの専門化した性質があげられる。それぞれのテスタ製造業者は、アドバンテスト、テラダイン、アジリエントといった会社間で互換性がないだけではなく、アドバンテストによって製造されたT3300、T5500およびT6600シリーズのテスタのようなプラットフォーム間でも互換性のないテスタプラットフォームを数多く有している。このように互換性がないために、各テスタは、他のテスタでは用いることができないそれ自身の特化したハードウェアおよびソフトウェアコンポーネントを必要とする。また、一つのテスタから他のテスタにテストプログラムを移植するのに多大な変換の手間が必要であり、サードパーティソリューションを開発するのは困難である。一つのプラットフォームから他のプラットフォームへの変換プロセスは、一般的に、複雑で間違いを起こしやすく、その結果よけいな手間、時間がかかり、テストのコストが増加することになる。
【０００５】
オペレーティングシステムおよびテスト解析ツール／アプリケーションのようなテスタのソフトウェアはホストコンピュータ上で動く。そのアーキテクチャの専用的な性質のために、すべてのハードウェアおよびソフトウェアは、任意のテスタに対して固定された構成のままである。ICをテストするためには、テスタの能力のいくらか、あるいはすべてを用いる専用のテストプログラムを開発して、テストデータ、信号、波形、ならびに電流および電圧レベルを定義するとともに、DUT応答を集めたり、DUTの合格／不合格を決めたりする。
【発明の開示】
【発明が解決しようとする課題】
【０００６】
テスタは幅広いICをテストすることができなければならないので、ハードウェアおよびソフトウェアコンポーネントは両方とも、幅広い動作範囲で動作するように設計される。したがって、テスタは多くのテストの状況下で用いられることのないリソースを多く含んでいる。同時に、任意のICに対して、テスタは、そのICに適した最も望ましいリソースを提供し得ないかもしれない。例えば、内蔵のマイクロコントローラ、大きな内蔵DRAMおよびフラッシュ、ならびにPCIやUSB等のようなさまざまな他のコアを含む複雑なSoC Aをテストするのに適しているロジックテスタは、内蔵マイクロコントローラおよび大きな内蔵DRAM／フラッシュを有してはいないが、DACおよびシグマーデルタコンバータを有しているASIC Bには不適切であることが判明するかもしれない。ASIC Bをテストするためには、それにふさわしいテスタは、内蔵メモリのテストのための広範囲なサポートよりも、むしろアナログおよび混合された信号のテストユニットを必要とするであろう。
【０００７】
したがって、テストの要件に応じて再構成可能であるテスタを提供することが望ましい。また、ATEと関連して他のベンダの機器と接続し、それを用いることが望ましい。しかしながら、従来のテストシステムの特化した性質と、各ベンダの機器のデータフォーマットの独自仕様というような性質とのために、他のベンダからの機器を差し込んで用いることはしばしば不可能である。
【課題を解決するための手段】
【０００８】
発明の一実施形態によるオープンアーキテクチャのテストシステムは、サードパーティモジュールの使用を許容する。このテストシステムのハードウェアおよびソフトウェアフレームワークは、異なるベンダからのモジュールがプラグアンドプレイの要領で相互作用し得る標準的なインタフェースを有する。モジュールは、機能ユニット、デジタルピンカード、アナログカード、あるいは装置電源のようなハードウェア、または、テスト実行ツール、システムモニタあるいはライセンスツール、ユニットレベルコントローラ（例えばベース機器、GPIB制御）、データベース、あるいは他の機器の制御のためのソフトウェアのようなツールまたはユーティリティのようなソフトウェアであってもよい。

【０００９】
一実施形態において、アーキテクチャは、マイクロソフトウィンドウズオペレーティングシステム下の分散オブジェクト環境である。テスタは、モジュールとモジュールとの通信だけでなく、コントローラとモジュールとの通信をも可能にするモジュール制御ソフトウェアとバックプレーン通信ライブラリとを有するモジュール化されたシステムである。モジュールは、例えば、デジタルモジュール、装置電源（DPS）モジュール、任意波形発生器（AWG）モジュール、デジタイザモジュールおよびアプリケーション特有のソフトウェアを含む。
【００１０】
一実施形態において、モジュール接続イネーブラは、マルチモジュール接続および同期メカニズムを提供するスイッチマトリクスネットワークを備えている。同じタイプの複数のDUTをテストするとき、このスイッチマトリクスネットワークは、複数のコントローラおよびテストサイトの間で共通のテストデータおよびリソースの共有も可能にする。
【００１１】
サイトごとに独立したサイトコントローラのおかげで、全てのテストサイトは非同期で動作することができる。これは結局、複数のDUTテストを容易にする。このモジュール化および複数サイト構成はまた、システムの拡張性も提供する。システムの一実施形態においては、単一のコントローラを、複数のDUTを制御、テストするように構成することができる。
【００１２】
プラグアンドプレイまたは交換可能なモジュールの考え方は、ハードウェア、ソフトウェア両方のレベルで標準的なインタフェースを用いることで助長される。ソフトウェアにおいては、モジュールを有効にし、アクティブにし、制御し、モニタするためにフレームワーククラスを用いる。フレームワークは、共通のテスト関連動作をインプリメントするクラスおよび方法のセットである。これは、電力供給およびピンエレクトロニクスの順序付け、電流／電圧レベルおよびタイミング条件の設定、測定値取得、テストフロー制御等のためのクラスを含んでいる。またフレームワークは、ランタイムサービスおよびデバッグのための方法も提供する。フレームワークオブジェクトは、発明の一実施形態による標準的なインタフェースをインプリメントすることを通して動作する。フレームワーククラスのC++ベースのレファレンスインプリメンテーションが提供される。ユーザは、自身に特有のフレームワーククラスを開発することもできる。
【００１３】
ハードウェアとソフトウェアとのインタフェース接続および通信は、バックプレーン通信ライブラリを通じて実現される。C++言語ベースのテストプログラムとC++より上のGUIテストプログラミングレイヤとを介してアクセスされるオープンバックプレーン通信ライブラリは、テストシステムのための一般化されたユーザインタフェースを提供する。C/C++コンストラクトを用いたテストプログラムを生成する方法は、米国出願第60/447,839号に開示されている。通信ライブラリは、ユーザアプリケーションおよびテストプログラムに見えないやり方でサイトコントローラと通信するメカニズムを提供する。基本的に、バックプレーン通信ライブラリは、テスタバックプレーン（この文脈では「バックプレーン」は抽象的なものであり、必ずしも物理的なハードウェアバックプレーンボードでなくてもよい）をまたいでの通信のために意図されたインタフェースを提供し、それによって、特定のサイトに接続されたモジュールとの通信に必要な機能を提供する。このライブラリの使用により、モジュールベンダは彼ら自身のドライバ（MS－ウィンドウズレベルのドライバ等）を作る必要がなくなる。これにより、ベンダ特有のモジュールソフトウェアは、対応するハードウェアモジュールと通信するために標準的なバックプレーンドライバを用いることが可能である。バックプレーン通信プロトコルは、一実施形態においてパケットベースのフォーマットを用いる。
【００１４】
オープンアーキテクチャの一つの利点は、これは全体的なテスタの使用法を単純にする

ということである。これは、サードパーティソリューションを開発し、これらのソリューションを重大な再作業を必要とすることなく再利用するためのメカニズムを提供する。任意のテストサイトに関して、所望する通りの適切なモジュールを選択し、用いることができる。モジュールは交換可能であるので、各テストサイトをDUTの最適テストを実現するように再構成することができる。またそれは、プラットフォーム間の非互換性の問題も単純にする。これらの単純化は全て、手間の減少、ターンアラウンド時間をはやくすることにつながり、結果としてテストコストの減少をもたらす。

【００１５】

本発明は、少なくとも一つのテスト対象装置（DUT）をテストするシステムを提供する。このシステムは、少なくとも一つのDUTに少なくとも一つのテスト（これはテストプランの一部であってもよい）を適用するように少なくとも一つのテストモジュールを制御する少なくとも一つのサイトコントローラを含んでいる。システムコントローラは、この少なくとも一つのサイトコントローラを制御する。

【００１６】

テストモジュールインタフェースは、サイトコントローラを第一のテストモジュールにインタフェース接続するテストモジュール機能を規定し、ここでテストモジュールインタフェースは、そのサイトコントローラを第二のテストモジュールにインタフェース接続するように拡張可能であり、拡張されていないテストモジュールインタフェースは、そのサイトコントローラを第二のテストモジュールにインタフェース接続するには不十分である。

【００１７】

このシステムはさらに、ユーザによって定義可能なテストクラスのような、DUT特有の特性からは独立している拡張可能なテスト機能を有している。テストは、拡張可能なテスト機能のインプリメンテーションである。

【００１８】

テストモジュールは、テスタピンインタフェースを用いてDUTと通信してもよく、これはDUT特有の特性からは独立していてもよい。テストモジュールインタフェースは、テストモジュールインタフェースクラスを備えていてもよく、テスタピンインタフェースはテスタピンインタフェースクラスを備えていてもよい。

【００１９】

発明の一実施形態による分散オペレーティングシステムは、システムコントローラによる少なくとも一つのサイトコントローラの制御を可能にするホストオペレーティングシステムと、各サイトコントローラに関連しており、関連するサイトコントローラによる少なくとも一つのテストモジュールの制御を可能にする少なくとも一つのローカルオペレーティングシステムとを備えている。少なくとも一つのテストモジュールが、対応するDUTに関するテストを行う。

【００２０】

ホストオペレーティングシステムは、少なくとも一つのサイトコントローラの動作を同期させてもよく、システムコントローラと少なくとも一つのサイトコントローラとの間の通信を仲裁してもよく、また少なくとも一つのサイトコントローラの動作をモニタしてもよい。サイトコントローラは、そのサイトコントローラに関連している少なくとも一つのテストモジュールの動作をモニタしてもよい。

【００２１】

ホストオペレーティングシステムは、少なくとも一つのサイトコントローラと通信するための少なくとも一つのホストインタフェースを備えている。テストモジュールインタフェースは、サイトコントローラを第一のテストモジュールにインタフェース接続するためのテストモジュール機能を規定し、テストモジュールインタフェースはそのサイトコントローラを第二のテストモジュールにインタフェース接続するように拡張可能であり、拡張されていないテストモジュールインタフェースはそのサイトコントローラを第二のテストモジュールにインタフェース接続するには不十分である。

【００２２】
ホストオペレーティングシステムは、少なくとも一つのホストフレームワーククラスを含んでいてもよく、これは、少なくとも一つのサイトコントローラを制御するためのアプリケーション特有のクラスをユーザが開発することを可能にするために、標準的なコンピュータ言語（例えばC/C++）で開発されてもよい。
【００２３】
各ローカルオペレーティングシステムは、少なくとも一つのローカルフレームワーククラスを有していてもよく、これは、少なくとも一つのサイトコントローラを制御するためのアプリケーション特有のクラスをユーザが開発することを可能にするために、標準的なコンピュータ言語（例えばC/C++）で開発されてもよい。
【００２４】
各サイトコントローラによって制御されるモジュールの数は拡張可能である。対応するサイトコントローラに関連しているローカルオペレーティングシステムは、そのサイトコントローラによって制御されるタイプのテストモジュールが再構成されることを可能にする。ホストオペレーティングシステムは、システムコントローラによって制御されるサイトコントローラの数を拡張可能とすることができ、テストシステムによってテストされるDUTの数を拡張可能とすることができる。
【００２５】
エミュレータは、対象テストモジュールをテストシステムとともに使用することをシミュレートして、対象モジュールがテストシステムと互換性を有していることを検証してもよい。
【００２６】
なお、上記の発明の概要は、本発明の必要な特徴の全てを列挙したものではなく、これらの特徴群のサブコンビネーションもまた、発明となりうる。
【発明を実施するための最良の形態】
【００２７】
以下、発明の実施の形態を通じて本発明を説明するが、以下の実施形態は特許請求の範囲にかかる発明を限定するものではなく、また実施形態の中で説明されている特徴の組み合わせの全てが発明の解決手段に必須であるとは限らない。
【００２８】
図１は、従来のテスタの一般化されたアーキテクチャを示しており、どのように信号が生み出されてテスト対象装置（DUT）に与えられるかを図示している。それぞれのDUT入力ピンは、テストデータを与えるドライバ２に接続されており、各DUT出力ピンはコンパレータ4に接続されている。多くの場合、各テスタピン（チャネル）が入力ピンまたは出力ピンのどちらかとして動作することができるように、３つの状態を有するドライバーコンパレータを用いる。単一のDUT専用のテスタピンは、関連するタイミング生成器６、波長生成器８、パターンメモリ１０、タイミングデータメモリ１２、波長メモリデータ１４、およびデータレートを規定するブロック１６とともに動作するテストサイトを共同で構成する。
【００２９】
図２は、本発明の一実施形態によるシステムアーキテクチャ１００を示している。システムコントローラ（SysC）１０２は複数のサイトコントローラ（SiteC）１０４に連結されている。またシステムコントローラは、関連するファイルにアクセスするようにネットワークにもつながれている。モジュール接続イネーブラ１０６を通じて、各サイトコントローラは、テストサイト１１０にある一つ以上のモジュール１０８を制御するように連結されている。モジュール接続イネーブラ１０６は、接続されたハードウェアモジュール１０８の再構成を可能にし、また（パターンデータをロードする、応答データを集める、制御を提供する等のための）データ転送用のバスとしても機能する。また、モジュール接続イネーブラを通じて、あるサイトのモジュールが他のサイトのモジュールにアクセスすることができる。モジュール接続イネーブラ１０６は、異なるテストサイトが同一または異

なるモジュール構成を有することを可能にする。言い換えれば、各テストサイトは、異なる数・タイプのモジュールを使用してもよい。考えられるハードウェアのインプリメンテーションには、専用の接続、スイッチ接続、バス接続、リング接続、およびスター接続が含まれる。モジュール接続イネーブラ１０６は、例えばスイッチマトリクスによってインプリメントされてもよい。各テストサイト１１０は、DUT１１２と関連づけられており、これはロードボード１１４を通じて対応するサイトのモジュールに接続されている。ある実施例においては、単一のコントローラを複数のDUTサイトに接続してもよい。

【００３０】

システムコントローラ１０２は、総合的なシステムマネージャとして機能する。これは、サイトコントローラの活動を統合し、システムレベルでの並列試験の計画を管理し、さらにハンドラ／プローブ制御を提供するとともにシステムレベルでのデータロギングおよびエラー処理サポートを提供する。動作設定に応じて、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上に配置されてもよい。あるいは、システムコントローラ１０２とサイトコントローラ１０４とで共通のCPUを共有してもよい。同様に、各サイトコントローラ１０４を、自身の専用CPU（中央演算処理装置）上に、あるいは同じCPU内の異なるプロセスまたはスレッドとして開発することもできる。

【００３１】

個々のシステムのコンポーネントを集積されたモノリシックなシステムの論理コンポーネントとして見なすことができ、必ずしも分散システムの物理的な構成要素として見なされなくてもよいという理解のもとに、システムアーキテクチャを、図２に示す分散システムとして概念的に描くことができる。

【００３２】

図３は、本発明の一実施形態によるソフトウェアアーキテクチャ２００を示している。ソフトウェアアーキテクチャ２００は、関連するハードウェアシステムの要素１０２、１０４、１０８と対応して、システムコントローラ２００、少なくとも一つのサイトコントローラ２４０、および少なくとも一つのモジュール２６０のための要素を有している分散オペレーティングシステムを表している。モジュール２６０に加えて、アーキテクチャ２００は、ソフトウェアでのモジュールエミュレーションのための対応する要素２８０を含んでいる。

【００３３】

例示的な選択として、このプラットフォームの用の開発環境はマイクロソフトのウィンドウズに基づいていてもよい。このアーキテクチャの使用は、プログラムおよびサポートの携帯性において副次的な利点（例えばフィールドサービスエンジニアは高度な診断を行うためのテスタオペレーティングシステムを動作させるラップトップコンピュータを接続することができるであろう）を有している。しかし、大規模なコンピュータ集約型の動作（テストパターンのコンパイル等）については、関連するソフトウェアは、独立して動作して分散されたプラットフォームを横断してのジョブスケジューリングを可能にすることができる独立した構成要素とされ得る。したがって、バッチジョブに関連するソフトウェアツールは、複数のプラットフォームタイプ上で動作することができる。

【００３４】

例示的な選択として、ANSI/ISO標準のC++をソフトウェア用のネイティブ言語とすることができる。当然のことながら、サードパーティが自身の選択した代わりの言語をシステムにまとめることを可能にする、（名目上のC++インタフェース上のレイヤを提供するための）使用可能な複数の選択肢がある。

【００３５】

図３は、名目上のソースによる組織化（あるいはサブシステムとしての集合的な開発）にしたがって、テスタオペレーティングシステムインタフェース２９０、ユーザコンポーネント２９２（例えば、テスト目的のためにユーザによって供給される）、システムコンポーネント２９４（例えば、基本的な接続性および通信のためのソフトウェアインフラとして提供される）、モジュール開発コンポーネント２９６（例えば、モジュールディベロ

ッパによって提供される）、および外部コンポーネント２９８（例えばモジュールディベロッパ以外の外部ソースによって提供される）を含む要素を陰付きで示している。

【００３６】

ソースベースの組織化の観点から、テスタオペレーティングシステム（TOS）インタフェース２９０は、システムコントローラ－サイトコントローラインタフェース２２２、フレームワーククラス２２４、サイトコントローラ－モジュールインタフェース２４５、フレームワーククラス２４６、所定のモジュールレベルインタフェース２４７、バックプレーン通信ライブラリ２４９、シャーシスロットIF（インタフェース）２６２、ロードボードハードウェアIF２６４、バックプレーンシミュレーションIF２８３、ロードボードシミュレーションIF２８５、DUTシミュレーションIF２８７、DUTのVerilogモデル用のVerilog PLI（プログラミング言語インタフェース）２８８、およびDUTのC/C++モデル用のC/C++言語サポート２８９を含んでいる。

【００３７】

ユーザコンポーネント２９２は、ユーザテストプラン２４２、ユーザテストクラス２４３、ハードウェアロードボード２６５、DUT２６６、DUT Verilogモデル２９３およびDUT C/C++モデル２９１を含んでいる。

【００３８】

システムコンポーネント２９４は、システムツール２２６、通信ライブラリ２３０、テストクラス２４４、バックプレーンドライバ２５０、HWバックプレーン２６１、シミュレーションフレームワーク２８１、バックプレーンエミュレーション２８２およびロードボードシミュレーション２８６を含んでいる。

【００３９】

モデル開発コンポーネント２９６は、モジュールコマンドインプリメンテーション２４８、モジュールハードウェア２６３およびモジュールエミュレーション２８４を含んでいる。

【００４０】

外部コンポーネント２９８は外部ツール２２５を含んでいる。

【００４１】

システムコントローラ２２０は、サイトコントローラに対するインタフェース２２２、フレームワーククラス２２４、システムツール２２６、外部ツール２２５および通信ライブラリ２３０を含んでいる。システムコントローラソフトウェアは、ユーザに対する相互作用の主要な点である。これは、発明のサイトコントローラへのゲートウェイと、同一譲受人による米国出願第60/449,622号に述べられているマルチサイト／DUT環境におけるサイトコントローラの同期化とを提供する。ユーザアプリケーションおよびツールは、グラフィカルユーザインタフェース（GUI）ベースかそれ以外のものであり、システムコントローラ上で動作する。システムコントローラは、テストプラン、テストパターンおよびテストパラメータファイルを含むすべてのテストプラン関連の情報の収納庫としても機能する。テストパラメータファイルは、発明の一実施形態のオブジェクト指向の環境におけるテストクラス用のパラメータ化されたデータを含んでいる。

【００４２】

サードパーティのディベロッパは、標準的なシステムツール２２６に加えて（あるいはその代わりとして）ツールを提供することができる。システムコントローラ２２０上の標準的なインタフェース２２２は、ツールがテスタおよびテストオブジェクトにアクセスするために用いるインタフェースを有している。ツール（アプリケーション）２２５、２２６は、テストおよびテスタオブジェクトの相互的なバッチ制御を可能にする。このツールは、（例えばSECS/TSEM等の使用を通じて）自動化能力を提供するためのアプリケーションを含んでいる。

【００４３】

システムコントローラ２２０上にある通信ライブラリ２３０は、ユーザアプリケーションおよびテストプログラムにトランスペアレントな形でサイトコントローラ２４０と通信

するメカニズムを提供する。
【００４４】
インタフェース２２２は、システムコントローラ２２０と関連したメモリに常駐しており、システムコントローラ上で実行するフレームワークオブジェクトに対するオープンインタフェースを提供する。サイトコントローラベースのモジュールソフトウェアがパターンデータにアクセス、取得することを可能にするインタフェースが含まれる。また、アプリケーションおよびツールがテスタおよびテストオブジェクトにアクセスするために用いるインタフェース、ならびに、スクリプトエンジンを通じてテスタおよびテストコンポーネントにアクセスして操作することができる能力を提供するスクリプトインタフェースも含まれる。これにより、インタラクティブな、バッチおよびリモートアプリケーションのための共通のメカニズムがそれらの機能を行うことが可能となる。
【００４５】
システムコントローラ２２０に関連しているフレームワーククラス２２４は、これらの上述したオブジェクトと相互に作用するメカニズムを提供し、これは標準的なインタフェースのリファレンスインプリメンテーションを提供する。例えば、発明のサイトコントローラ２４０は機能テストオブジェクトを提供する。システムコントローラフレームワーククラスは、この機能テストオブジェクトのリモートシステムコントローラベースの代理として、対応する機能テストプロキシを提供してもよい。したがって、標準的な機能テストインタフェースは、システムコントローラ２２０上のツールに対して利用可能とされる。システム、モジュール開発コンポーネントおよびインタフェースコンポーネント２９４、２９６および２９０はそれぞれ、システムコントローラとサイトコントローラとの間で分散されたオペレーティングシステムであると考えてもよい。フレームワーククラスは、ホストシステムコントローラに関連するオペレーティングシステムインタフェースを実質的に提供する。これらはまた、サイトコントローラに対するゲートウェイを提供するソフトウェア要素も構成し、マルチサイト／DUT環境におけるサイトコントローラの同期を提供する。したがってこのレイヤは、コミュニケーションレイヤを直接扱う必要なくサイトコントローラを操作し、それにアクセスするのに適している、発明の一実施形態におけるオブジェクトモデルを提供する。
【００４６】
サイトコントローラ２４０は、ユーザテストプラン２４２、ユーザテストクラス２４３、標準テストクラス２４４、標準インタフェース２４５、サイトコントローラフレームワーククラス２４６、モジュールハイレベルコマンドインタフェース（例えば所定のモジュールレベルのインタフェース）２４７、モジュールコマンドインプリメンテーション２４８、バックプレーン通信ライブラリ２４９、およびバックプレーンドライバ２５０のホストとなる。好ましくは、テストの機能の大半をサイトコントローラ１０４／２４０が扱い、それによってテストサイト１１０の独立した動作が可能である。
【００４７】
テストプラン２４２はユーザによって書かれる。このプランは、C++のような標準的なコンピュータ言語で直接記述されてもよいし、実行可能なテストプログラムへとコンパイル可能であるC++コードを生成するような、より高レベルのテストプログラミング言語で記述されてもよい。
【００４８】
このテストプランは、フレームワーククラス２４６および／または、サイトコントローラに関連する標準あるいはユーザによって供給されるテストクラス２４４を用いて、テストオブジェクトを作り出し、標準インタフェース２４５を用いてハードウェアを構成し、テストプランのフローを定義する。また、テストプランの実行中に必要とされる追加的なロジックも提供する。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（例えばブレークポイント）等のその下にあるオブジェクトのサービスに対するインタフェースと、その下にあるフレームワークおよび標準クラスへのアクセスとを提供する。

【００４９】
サイトコントローラに関連するフレームワーククラス２４６は、共通のテスト関連動作をインプリメントするクラスおよび方法のセットである。サイトコントローラレベルフレームワークは、例えば、電力供給およびピンエレクトロニクスの順番付け、レベルおよびタイミング条件の設定、測定値取得、テストフロー制御のためのクラスを含んでいる。フレームワークオブジェクトは、標準インタフェースをインプリメントすることを通じて動作してもよい。例えば、テスタピンフレームワーククラスのインプリメンテーションは、テストクラスがハードウェアモジュールピンと相互に作用するために用いるであろう汎用のテスタピンインタフェースをインプリメントするように統一される。
【００５０】
あるフレームワークオブジェクトは、モジュールと通信するためにモジュールレベルインタフェース２４７の助けを借りて動作するようにインプリメントされてもよい。サイトコントローラフレームワーククラスは、実質的に、各サイトコントローラをサポートするローカルオペレーティングシステムとして機能する。
【００５１】
一般的に、プログラムコードの９０％以上は装置テスト用のデータであり、残りの１０％のコードがテスト方法を実現する。装置テストデータはDUT依存のデータ（例えば電力供給条件、信号電圧条件、タイミング条件等）である。テストコードは、指定された装置条件をATEハードウェア上にロードする方法からなり、またユーザが指定した目的（データロギング等）を実現するのに必要である方法からも構成される。発明の一実施形態のフレームワークは、ハードウェア依存性のテストと、ユーザがDUTテストプログラミングのタスクを行うことを可能にするテスタオブジェクトモデルとを提供する。
【００５２】
テストコードの再利用性を高めるために、このようなコードは、装置特有のデータ（例えばピンの名前、刺激データ等）、あるいは装置テストに特有のデータ（例えばDCユニットの条件、測定ピン、ターゲットピンの数、パターンファイルの名前、パターンプログラムのアドレス）のいずれに対しても独立とされてもよい。もしテスト用のコードをこれらのタイプのデータとともにコンパイルすれば、テストコードの再利用性は低下する。したがって、発明の一実施形態によれば、いかなる装置特有のデータあるいは装置テストに特有のデータも、コード実行期間中の入力として、外部からテストコードに役立てられてもよい。
【００５３】
発明の一実施形態においては、標準テストインタフェースのインプリメンテーションであるテストクラスは、ここでたITestと記載するが、特定のタイプのテストに関してテストデータとコードとの分離（したがってコードの再利用性）を実現する。このようなテストクラスは、装置特有および／あるいは装置テスト特有のデータにおいてのみ異なるような別々のテストクラスの「テンプレート」とみなしてもよい。テストクラスはテストプランファイルにおいて指定される。各テストクラスは、典型的には、具体的なタイプの装置テストあるいは装置テスト用のセットアップをインプリメントする。例えば、発明の一実施形態は、DUTに関するすべての機能テストの基本となるクラスとして、ITestインタフェースの具体的なインプリメンテーション、例えばFunctionalTestを提供する。それは、テスト条件の設定、パターンの実行および、失敗したストローブの存在に基づくテスト状況の判定という基本的な機能を提供する。他のタイプのインプリメンテーションは、ここではACParametricTestおよびDCParametricTestとして表記されるACおよびDCテストクラスを含んでいてもよい。
【００５４】
全てのテストタイプは、いくつかの仮想的な方法のデフォルトのインプリメンテーション（例えば、init( )、preExec( )およびpostExec( )）を提供してもよい。これらの方法は、デフォルトの動作を乗り越えてテスト特有のパラメータを設定するためのテストエンジニアのエントリポイントとなる。しかしながら、カスタムテストクラスもテストプラン

において用いることができる。
【００５５】
テストクラスは、そのテストの特定の場合に関するオプションを指定するために用いられるパラメータを提供することによって、ユーザがクラスの動作を構成することを可能にする。例えば、機能テストは、実行すべきパターンリストとテスト用のレベルおよびタイミング条件とを指定するために、パラメータPlistおよびTestConditionと採用してもよい。（テストプラン記述ファイルにおける異なる「テスト」ブロックの使用を通して）これらのパラメータについて異なる値を指定することにより、ユーザは機能テストの異なる例を作り出すことが可能である。図４は、どのようにして単一のテストクラスから異なるテスト例が導き出されるかを示している。テンプレートライブラリは、一般的なアルゴリズムおよびデータ構造の汎用ライブラリとして採用されてもよい。このライブラリはテスタのユーザに見えてもよく、ユーザは、例えば、ユーザ定義のテストクラスを作り出すようにテストクラスのインプリメンテーションを改変してもよい。
【００５６】
ユーザによって開発されるテストクラスに関して、システムの一実施形態は、このようなテストクラスを、全てのテストクラスが単一のテストインタフェース、例えばITestから得られるようなフレームワークに統合することをサポートし、その結果、そのフレームワークはシステムテストクラスの標準的なセットと同じようなやり方でそれらを処理することができる。ユーザは、追加のファシリティを生かすためには自分達のテストプログラムにおいてカスタムコードを用いなければならないという理解のもとで、自分達のテストクラスに追加の機能を自由に追加することができる。
【００５７】
各テストサイト１１０は、一つ以上のDUT１０６のテスト専用のものであり、テストモジュール１１２の構成可能な集合体を通じて機能する。各テストモジュール１１２は特定のテストタスクを行う主体である。例えば、テストモジュール１１２は、DUTの電源、ピンカード、アナログカード等であり得る。モジュールによるこのアプローチは、高いフレキシビリティと構成可能性を提供する。
【００５８】
モジュールコマンドインプリメンテーションクラス２４８は、モジュールハードウェアベンダによって提供されてもよく、ベンダによって選択されるコマンド実行方法に応じて、ハードウェアモジュールに関するモジュールレベルインタフェースをインプリメントするか、あるいは標準的なインタフェースのモジュール特有のインプリメンテーションを提供する。これらのクラスの外部インタフェースは、所定のモジュールレベルインタフェース要件およびバックプレーン通信ライブラリ要件によって規定される。またこのレイヤは、標準的なセットのテストコマンドの拡張も提供し、それにより方法（機能）およびデータ要素の追加が可能となる。
【００５９】
バックプレーン通信ライブラリ２４９は、バックブレーンをまたいでの標準的な通信のためのインタフェースを提供し、それによってテストサイトに接続されたモジュールとの通信に必要な機能を提供する。これにより、ベンダに特有のモジュールソフトウェアが対応するハードウェアモジュールとの通信にバックプレーンドライバ２５０を用いることが可能である。バックプレーン通信プロトコルはパケットベースのフォーマットである。
【００６０】
テスタピンオブジェクトは、物理的なテスタチャネルを表しており、ここではITesterPinで示されるテスタピンインタフェースから得られる。発明の一実施形態によるソフトウェア開発キット（SDK）は、TesterPinと呼ばれることもあるITesterPinのデフォルトのインプリメンテーションを提供し、これは所定のモジュールレベルインタフェースIChannelに関してインプリメントされる。ベンダは、IChannelに関して彼らのモジュールの機能をインプリメントすることができるのであればTesterPinを自由に使うことができるが、そうでなければ、彼らのモジュールとともに動作するITesterPinのインプリメンテーション

を提供しなければならない。

【００６１】

発明のテスタシステムによって提供される標準的なモジュールインタフェースは、ここではIModuleと表記するが、これは一般的には、ベンダのハードウェアモジュールを表している。ベンダによって供給される、システム用のモジュール特有のソフトウェアは、ダイナミックリンクライブラリ（DLL）のような実行可能な形態で提供されてもよい。ベンダからの各モジュールタイプ用のソフトウェアは、単一のDLLにカプセル化されていてもよい。このようなソフトウェアモジュールのそれぞれは、モジュールソフトウェア開発のためのAPIを備えている、モジュールインタフェースコマンド用のベンダに特有なインプリメンテーションを提供することを担っている。

【００６２】

モジュールインタフェースコマンドには２つの局面がある。それらは、第一に、ユーザがシステムにおける特定のハードウェアモジュールと（間接的に）通信するためのインタフェースとして機能し、第二に、サードパーティディベロッパが彼ら自身のモジュールをサイトコントローラレベルのフレームワークに統合するために活用することができるインタフェースを提供する。したがって、フレームワークによって提供されるモジュールインタフェースコマンドは、2つのタイプに分けられる。

【００６３】

一つ目は、最も疑う余地のないものであるが、フレームワークインタフェースを通じてユーザに対してあらわになる「コマンド」である。したがって、例えば、テスタピンインタフェース（ITesterPin）は、レベルおよびタイミングの値を取得、設定するための方法を提供し、一方で電源インタフェース（IPowerSupply）は電力を上げたり下げたりする方法を提供する。

【００６４】

また、フレームワークは、モジュールとの通信に用いられることができる、所定のモジュールレベルインタフェースの特別なカテゴリを提供する。これらは、ベンダのモジュールとの通信のためにフレームワーククラスによって用いられるインタフェース（すなわち、フレームワークインタフェースの「標準的な」インプリメンテーション）である。

【００６５】

しかしながら、第二の局面、モジュールレベルインタフェースの使用は、任意のものである。それをすることの利点は、ベンダは、モジュールレベルインタフェースをインプリメントすることによって彼らのハードウェアに対して送られる具体的なメッセージの内容を注視しつつ、ITesterPinおよびIPowerSupplyのようなクラスのインプリメンテーションを活用し得るということである。しかし、もしこれらのインタフェースがベンダに不適切であれば、それらはフレームワークインタフェースのそれらのカスタムインプリメンテーション（例えばITesterPin、IPowerSupply等のベンダインプリメンテーション）を提供することを選択してもよい。そうすればこれらは、それらのハードウェアに対して適切であるカスタム機能を提供するであろう。

【００６６】

したがって、モジュールに特有なベンダソフトウェアの統合は、2つの異なった手段、すなわち、関連するフレームワーククラスおよびインタフェースのカスタムインプリメンテーション、あるいはモジュールレベルインタフェースの特別なカテゴリのカスタムインプリメンテーションを通じて実現され得る。

【００６７】

次に、両方の方法の例示的な応用を図５の助けを借りて説明する。図５は、発明の一実施形態によるテスタシステムとベンダによって供給されるモジュールとの相互作用を示すユニバーサルモデリング言語（UML）クラスダイアグラムである。

【００６８】

新しいデジタルモジュールのベンダであるサードパーティA（TPA）は、そのハードウェアモジュールと通信するためのソフトウェアモジュールを提供する。このソフトウェアモ

ジュールは、標準的なインタフェースIModuleをインプリメントする。このモジュールオブジェクトをTPAPinModuleと呼ぶことにしよう。ベンダTPAは、そのモジュールにおいて、関連する所定のモジュールレベルインタフェース、この場合にはIChannelをインプリメントすることによって、ここではTesterPinとして表されている、ITesterPinインタフェースの標準的なシステムインプリメンテーションを利用することができる。これは、TesterPinがモジュールと通信するためにIChannelのような標準的な所定のモジュールレベルインタフェースを用いるという事実によって可能とされる。したがって、TPAPinModuleは、TesterPinオブジェクトを単に作り出してあらわにすることでピンを提供する。

【００６９】

ここで、IChannelインタフェースは自分達のハードウェアとともにはうまく動作しないと判断する、異なるベンダであるサードパーティB（TPB）を考える。したがって、TPBは、彼ら自身のIModuleインプリメンテーション（TPBPinModule）だけではなく、ITesterPinインタフェースのインプリメンテーションTPBTesterPinも提供することが必要となる。

【００７０】

このアプローチは、サードパーティディベロッパがどのようにして自分達のハードウェアを開発するかの選択およびソフトウェアのサポートにおいて、多大なフレキシビリティをサードパーティディベロッパに与える。彼らはIModuleインタフェースをインプリメントすることを求められながら、モジュールレベルインタフェースをインプリメントすることか、適合するとわかればTesterPinのようなオブジェクトをインプリメントすることかを選択し得る。

【００７１】

実際に、ベンダは、ITesterPinインタフェースにおいてはサポートされていない拡張を提供するためにTesterPinをインプリメントすることを選択してもよい。フレームワークはユーザに、特定のインタフェースを取り出すためのメカニズムあるいはオブジェクトへのインプリメンテーションポインタを提供する。これは、ユーザコードがITesterPinポインタを有している場合に、フレームワークはそれが必要であるときにいわゆるTPBTesterPinオブジェクトをポイントしているかどうかを判断することができるということを意味している。（この特徴は標準的なC++ランタイムタイプ識別（RTTI）を介して提供されてもよいことに留意されたい。）言い換えると、テストプランがITesterPinインタフェースを要求するときには、インタフェースは、TesterPinクラスのベンダのテスタピンのインプリメンテーションを直接呼び出し、これがモジュールに特有の情報（例えば、特定のDUT刺激を与えるように設定されるべきレジスタのアドレス）を内蔵している。

【００７２】

まとめると、フレームワークコードが常にITesterPinインタフェースを使用している間、ユーザは、必要なときにモジュールベンダによって提供される具体的な特徴および拡張を自由に使うことができる。言い換えると、モジュールベンダは、例えば、クラスの標準的なシステムインプリメンテーションに方法（機能）を付加することができる。ユーザに対するトレードオフは、具体的なベンダの拡張を活用することが他のベンダのモジュールに対するテストコードの有用性を低下させるということである。

【００７３】

モジュールのレベルでは、システム１００は、名目上2つの動作モードを有している。動作のオンラインモードでは、モジュールエレメント２６０（例えばハードウェアエレメント）が用いられ、動作のオフラインモードではソフトウェアにおけるモジュールエミュレーション２８０が用いられる。

【００７４】

動作のオンラインモードについて、モジュールエレメント２６０は、HW（ハードウェア）バックプレーン２６１、シャーシスロットIF（インタフェース）２６２、モジュールハードウェア２６３、ロードボードハードウェアIF２６４、ハードウェアロードボード２６５およびDUT２６６を有している。

【００７５】

動作のオフラインモードについて、ソフトウェアでのモジュールエミュレーション２８０は、シミュレーションフレームワーク２８１、バックプレーンエミュレーション２８２、バックプレーンシミュレーションIF２８３、モジュールエミュレーション２８４、ロードボードシミュレーションIF２８５、ロードボードシミュレーション２８６およびDUTシミュレーションIF２８７を有している。2つのモデルをDUTシミュレーションに関して示す。Verilogを用いるモデルは、Verilog PLI（プログラミング言語インタフェース）２８８とDUT Verilogモデル２９３とを有している。C/C++を用いるモデルは、C/C++言語サポート２８９とDUT C/C++モデル２９１とを有している。シミュレーションは、PC等のいかなるコンピュータ上でも行うことができることに留意されたい。

【００７６】

オンラインモードでは、モジュールベンダは、デジタルテスタチャネル、DUT電源、あるいはDC測定ユニットといった、テストをサポートするための物理的なハードウェアコンポーネントを提供する。モジュールは、シャーシスロットIF２６２を通じてHWバックプレーン２６１にインタフェース接続される。

【００７７】

オフラインでの作業については、システムコントローラと等価なものを動かすPCベースあるいは他の環境が、付加的に、サイトコントローラレベルのフレームワークと、ソフトウェアのより低いレイヤのランタイム環境とを提供するとともにハードウェアをエミュレートするための全ての任務を引き受ける。

【００７８】

バックプレーンエミュレーション２８２は、物理的なバックプレーン２６１のためのソフトウェアによる代理を提供する。これは、バックプレーンシミュレーションインタフェース２８３を通して（ベンダが供給する）モジュールエミュレーションソフトウェア２８４と通信する。

【００７９】

モジュールエミュレーションソフトウェア２８４は、好ましくはモジュールベンダによって提供され、典型的にはモジュール２６３の特定のベンダインプリメンテーションと密接に結びついている。したがって、モジュールエミュレーションソフトウェアは、典型的には、異なるベンダによって供給されるモジュール間で詳細で異なっている。この場合、モジュールシミュレーションにより、ベンダは、ソフトウェアモデル（例えばモジュールエミュレーションソフトウェア２８４）を通してハードウェアの機能をあらわにし、シミュレートされるロードボード２８６に対して刺激信号を送り、DUTシミュレーションIF２８７を介してDUTモデリングソフトウェア２９１、２９３に接続されているシミュレートされるロードボード２８６からのDUT応答信号を受け取って処理することが可能になる。モジュールの単純な機能シミュレーションを提供してモジュールファームウェアのエミュレーションを迂回することが有利であるとベンダが考える場合もある。モジュールエミュレーションソフトウェアは、シミュレートされたモジュール刺激信号に対するシミュレートされたDUTの応答を、既知の良好なDUT応答と比較する。この比較に基づいて、ソフトウェアは、そのモジュールによって実行されているテストが所望のとおりDUTをテストするという目標に適合しているかを判断し、ユーザがオンラインの実際のテスタ上のIC（実際のDUT）上でそれを用いるのに先立って、モジュールのデバッグを行うことを助ける。

【００８０】

ロードボードシミュレーションインタフェース２８５は、モジュールエミュレーションレイヤおよびシミュレートされるロードボード２８６への、およびこれらからの信号のためのルートとして機能する。ロードボードシミュレーションコンポーネント２８６は、デバイスソケットマッピングと、DUTシミュレーションIF２８７への、およびそれからの信号伝達とをサポートする。

【００８１】

DUTシミュレーションは、ネイティブコード（すなわちC/C++）シミュレーション２９１、または目的のテスト対象装置２９３の機能モデルに対するVerilogプログラミング言語

インタフェース（PLI）であってもよい。このモデルは、DUTシミュレーションインタフェース２８７を通じて、シミュレートされるロードボードとインタフェース接続する。

【００８２】

これらのレイヤの全体の制御はシミュレーションフレームワーク２８１によって提供されることに留意されたい。シミュレーションフレームワークは、既知の刺激信号に対するシミュレートされたDUT応答を測定する。システムエミュレーションの方法は、米国出願第10/403,817号に開示されている。

通信および制御

【００８３】

通信および制御は、関連するソフトウェアオブジェクトの管理を通じて実現される。好ましくは、通信のメカニズムは、システムコントローラ上のオブジェクトモデルの後ろに隠れている。このオブジェクトモデルは、サイトコントローラ上に見られるクラスおよびオブジェクトに対してプロキシを提供し、それによって、アプリケーションの開発（例えばIC装置のテスト）のための便利なプログラミングモデルを提供する。これにより、アプリケーションの開発者（例えばATEシステムのユーザ）は、アプリケーションとサイト／システムコントローラとの間の通信の具体的な情報に関連する不要な詳細を避けることが可能である。

【００８４】

図６は、サイトコントローラソフトウェア２４０内にサイトコントローラ１０４によって維持されているときのサイトコントローラオブジェクトの具体的な実施形態を示している。サイトコントローラオブジェクトは、CmdDispatcher６０２、FunctionalTestMsgHandler６０４およびFunctionalTest６０６を有している。インタフェースは、IMsgHandler６０８およびITest６１０を有している。

【００８５】

好ましくはサイトコントローラ２４０は、アプリケーションがアクセスのために必要とする機能クラスの全てを含んでいる。これらのクラスは、例えば、テスト、モジュール、ピン等を含む。ユーザのテストおよびソフトウェアツールは典型的には異なるコンピュータ上に存在するので、メッセージは、システムコントローラ上のツールからサイトコントローラ上のサーバに送られる。このサーバは、コマンド発送オブジェクトに関する方法を必要とする。

【００８６】

コマンド発送オブジェクト（CmdDispatcher）６０２は、IMsgHandlerインタフェース６０８をインプリメントするメッセージハンドラオブジェクトのマップを保持する。図６は、IMsgHandlerの具体的なインプリメンテーションFunctionalTestMsgHandler６０４を示している。DmdDispatcherオブジェクト６０２によって受信されたメッセージは通信すべきオブジェクトの識別子を含んでいる。この識別子は、内部のマップにおいて見られ、具体的なインプリメンテーション、この場合には図示されているFunctionalTestMsgHandlerオブジェクト６０４に帰着する。

【００８７】

本例では、IMsgHandler６０８は、単一の方法handleMessage()からなる。この方法は、好ましくは単一のインプリメンテーションクラスとしてインプリメントされる。図示されている場合においては、FunctionalTestMsgHandler６０４は、入ってくるメッセージの正確な性質に応じて、６つの方法のうちの１つにメッセージを送る。入ってくるメッセージのヘッダは、メッセージハンドラがどのようにメッセージを解釈し、どこにメッセージを送るかを決定することを可能にするメッセージIDを含んでいる。

【００８８】

システムコントローラ１０２における対応する通信環境は、システムコントローラソフトウェア２２０のツール２２５、２２６セクションに関連する。図７は、システムコントローラソフトウェア２２０においてシステムコントローラ１０２上に保持されるツールオ

ブジェクト（あるいはシステムコントローラオブジェクト）の一実施形態を、図6に示したサイトコントローラオブジェクトと対応するように示している。ツールオブジェクトは、オブジェクトCmdDispatcher７０２、FunctionalTestMsgHandler７０４およびFunctionalTestProxy７０６を含んでいる。インタフェースは、IMsgHandler７０８、ITestClient７１０、およびIDispatch７１２を含んでいる。またユーティリティアプリケーション７１４も含まれる。

【００８９】

この例に関して、クラスCmdDispatcher７０２、IMsgHandler７０８、およびFunctionalTestMsgHandler７０４は図６に示したものと同じである。しかしながら、FunctionalTest６０６（あるいは他のいかなるサイトコントローラクラス）のインスタンス化は用いられない。代わりに、ツールオブジェクトは、サイトコントローラ１０４上の各オブジェクトと通信するためのプロキシクラスを有している。したがって例えば、ツールオブジェクトはFunctionalTest６０６に代えてクラスFunctionalTestProxy７０６を含んでいる。同様に、ツールオブジェクトにおけるITestClient７１０は、サイトコントローラオブジェクトにおけるITest６１０と同じではない。一般的に、サイトコントローラ１０２上で動作するアプリケーションは、サイトコントローラ１０４上に設けられているものそのもののようなインタフェースを用いない。この場合、ITest６１０の3つの方法（すなわち、preExec()、execute()およびpostExec()）はITestClient７１０における単一の方法（すなわちrunTest()）に置き換えられる。またITestClient７１０は、好ましくはデュアルインタフェース、すなわち、IDispatch７１２を受け継ぐものであり、マイクロソフトコンポーネントオブジェクトモデル（COM）としてインプリメントされる。それは、そのインタフェースをインプリメントするオブジェクトへのスクリプトエンジンのアクセスを可能にするようなインタフェースを提供する。これによって、システムをマイクロソフトウィンドウズプラットフォーム上で記述することが可能となる。

【００９０】

図６～７に示す実施形態の動作の一例として、（例えば、ツールセクション２２６、２２８のうちの一つにおいて）システムコントローラ１０２上で動作するアプリケーションは、テストプラン２４２が一つ以上のFunctionalTestオブジェクト６０６を有しているようなサイトコントローラ１０４と通信してもよい。サイトコントローラ１０４上でのテストプラン２４２の初期化中に、対応するテストプランオブジェクトはサイトコントローラ１０４上にロードされ、TestPlanMessageHandlerオブジェクトを構成し、それをCmdDispatcherオブジェクト６０２とともに登録する。これがメッセージハンドラに独自のIDを割り当てる。同様な動作は、テストプラン２４２を構成する他のTestPlanオブジェクトでも起こる。

【００９１】

システムコントローラ１０３上の（例えばツール２２６、２２８における）アプリケーションは、通信ライブラリ２３０を初期化し、通信チャネルを介してサイトコントローラ１０４に接続し、TestPlanオブジェクトのためのIDを取得する。この初期化中に、プロキシオブジェクトは、それがテストをいくつ含んでいるかと、それらのタイプおよびIDとを決定する。それはタイプごとに（この場合には一つだけのタイプ）適切なDLLをロードし、それらに関するプロキシオブジェクトを構成し、それらをID値を用いて初期化する。

【００９２】

TestProxyオブジェクトも初期化する。これをするために、それらは、それらの名前を（それらのID値を用いて）取得するための適切なメッセージを構成してサイトコントローラ１０４の通信サーバに送信する。通信サーバは、メッセージをCmdDispatcher６０２に渡す。このオブジェクトは、その内部マップにおいて宛先IDを調べて、FunctionalTestMsgHandler６０４のhandleMessage()方法にメッセージを送る。例えばもしメッセージがテスト名取得の要求であれば、これらのオブジェクトは、それぞれのテスト名を取得し、適切なネーム列でアプリケーションのTestProxyオブジェクトに応答する。

【００９３】

初期化が完了すると、アプリケーションは、TestPlanオブジェクトへのリモートアクセスと、それを通じて両方のTestオブジェクトへのリモートアクセスを有する。ユーザはここで、例えば、アプリケーション上の「テストプラン起動」のボタンを押す。その結果、アプリケーションはTestPlanProxyオブジェクト上のRunTestPlan()方法を呼び出す。この方法は、TestPlanオブジェクトの宛先IDでRunTestPlanメッセージを構成し、RPCプロキシ上でsendMessage()機能を呼び出す。この機能がサイトコントローラにメッセージを送信する。

【００９４】

サイトコントローラ１０４上の通信サーバは、CmdDispatcherオブジェクト６０２上のhandleMessage()方法を呼び出し、TestPlanオブジェクトのIDをそれに渡す。CmdDispatcherオブジェクト６０２はその内部マップでこのIDを調べて、TestPlanオブジェクト用のメッセージハンドラを見つけて、このオブジェクト上のhandleMessage()方法を呼び出し、これがTestPlanオブジェクト上のRunTestPlan()方法を呼び出す。同じようなやり方で、アプリケーションは名前とTestオブジェクトの最近の動作状況とを取得することができる。

通信ライブラリを用いる方法

【００９５】

通信ライブラリ２３０を用いる例を以下に述べる。

【００９６】

通信ライブラリ２３０は好ましくは静的なライブラリである。アプリケーションは、CommLibrary.hファイルを通してこの通信ライブラリを使用することができる。通信ライブラリクラスをエクスポートする必要があるアプリケーションは、上記インクルードファイルを含むことに加えて、定義されたプリプロセッサ定義COMMLIBRARY_EXPORTS、COMMLIBRARY_FORCE_LINKAGEを有していなければならない。通信ライブラリをインポートするアプリケーションは、プリプロセッサ定義を何も定義する必要はない。通信ライブラリがサーバとして用いられるときには、アプリケーションは、CcmdDispatcherの次の静的な関数を呼び出さなければならない：InitializeServerunsigned long portNo）。

【００９７】

このportNoは、サーバが要求を待たなければならないポート番号である。サーバに対応するコマンドディスパッチャは、静的な関数getServerCmdDispatcherをCcmdDispatcherクラス上に呼び出すことによって読み出される。

【００９８】

通信ライブラリがクライアントとして用いられるときには、アプリケーションはCcmdDispatcherの以下の静的な関数を呼び出さなければならない。

```
InitializeClientconst OFCString serverAddress,
unsigned long serverPortNo,
CcmdDispatcher **pCmdDispatcher,
OFCString serverId)
```

【００９９】

このserverAddressおよびServerPortNoは、クライアントが接続しなければならないものである。この関数は、クライアント用のコマンドディスパッチャポインタおよびそれが接続するサーバIDを初期化する。また、後の時点で、クライアントは、静的な関数getClientCmdDispatcherを呼び出すことによってサーバIDに対応するコマンドディスパッチャを取り出すことができる。

【０１００】

通信ライブラリがコンパイルされるときには、ファイルClientInterface.idlおよびServerInterface.idl上ではビルドは排除される。好ましい実施形態は、これらのインタフェース定義ファイルに関して既に生成されたスタブおよびプロキシファイルを適用して、プロキシおよびスタブインプリメンテーションファイルを同じライブラリにリンクする。し

たがって、サーバおよびクライアントは、同じアドレス空間内にインスタンス化される。インタフェース定義ファイルおよびスタブファイルにおける以下の変更は、好ましくは、通信ライブラリをサーバおよびクライアントとして同じアドレス空間内で動作させるために行われる。

インタフェース定義ファイルにおける変更

【０１０１】

以下のネームスペースの宣言は、好ましくは、インタフェース定義ファイルのそれぞれにおいて付加される。これは、プロキシインプリメンテーション機能とインタフェース機能の我々自身のインプリメンテーションとの名前の衝突を避けるためである。以下のネームスペースの宣言は、serverInterface.idlにおいて付加される。

```
cpp_quote("#ifdef __cplusplus")
cpp_quote("namespace COMM_SERVER")
cpp_quote("{")
cpp_quote("#endif")
cpp_quote("}")
```

【０１０２】

スタブインプリメンテーションファイルにおける関数は、インタフェースにおいて宣言された機能のための我々自身のインプリメンテーション関数を呼び出すように変更される。すなわち、我々は、インタフェースにおいて宣言された機能のそれぞれに対応する異なる名前の関数をもつことになる。

【０１０３】

関数呼び出しにおける競合を避けるために、インプリメンテーション関数の名前を「COMM_」列で始まる名前とすることが好ましい。そうすればスタブ関数におけるコードは、「functionName」に代えて「COMM_functionName」を呼び出すように変更される。

【０１０４】

この方法が動作するためには、存在する全ての機能クラスは、対応するメッセージハンドラオブジェクトおよびプロキシクラスも有していなければならない。全てのメッセージハンドラオブジェクトは、通信ライブラリによって提供されるIMsgHandlerクラスから得られなければならない。IMsgHandlerクラスは抽象的なクラスである。メッセージハンドラのインプリメンタの任務は、handleMessage、setObject、handleErrorの定義を提供することが好ましい。全てのメッセージタイプは、１から始まらなければならない（ゼロはhandleErrorのためにとっておく）。機能クラスは好ましくは、そのメンバが可変であるように対応するメッセージハンドラを有する。機能クラスのコンストラクタにおいて、機能クラスは、そのメッセージハンドラによって提供される関数を呼び出すことによって、メッセージハンドラとともに自身を登録させる。次にメッセージハンドラオブジェクトは、addMsgHandler関数をコマンドディスパッチャ上にパラメータとしてのメッセージハンドラとともに呼び出すことによって、コマンドディスパッチャとともに登録されなければならない。addMsgHandler関数は、メッセージハンドラおよび機能クラスにIDを割り当てる。機能クラスのデストラクタは、パラメータとしての機能クラス識別子を送ることによって、コマンドディスパッチャ上にremoveMsgHandler関数を呼び出さなければならない。またプロキシクラスも、機能クラスに関して説明されたように、同じ登録手順に従わなければならない。

【０１０５】

以下のCTestPlanクラスは、典型的な機能クラスがどのように本発明の好ましい実施形態に従うかを示している。

```
File: - TestPlan.h
Class CTestPlan
```

```
{
private:
    unsigned long m_Id;
    CTestPlanMsgHandler m_tplMsgHandler;
}
File: - TestPlan.cpp
extern CcmdDispatcher *g_pCmdDispatcher;
CTestPlan::CTestPlan
{
    m_tplMsgHandler.setTestPlan(this);
    g_pCmdDispatcher.AddMsgHandler(&m_tplMsgHandler)
}
CTestPlan::~CTestPlan
{
    g_pCmdDispatcher.removeMsgHandler(m_Id)
}
```

【０１０６】

この g_pCmdDispatcher object は、通信ＤＬＬ’ｓによりエクスポーズされるgetCmdDispatcher()を呼び出すことで初期化される。以下のCTestPlanMsgHandler クラスは、典型的なメッセージハンドラがどのようなものかを示す。

```
File: - TestPlanMsgHandler.h
Class CtestPlanMsgHandler : public IMsgHandler
{
public:
    setTestPlan(CTestPlan *pTestPlan);
    setTestPlanProxy(CTestPlanProxy *pTestPlanProxy);
    void handleMessage(unsigned long msgType,
                       unsigned long senderId,
                       unsigned long senderMsgLen,
                       byte *pSenderMsg)
    void handleSetName(unsigned long senderId,
                       unsigned long senderMsgLen,
                       byte *pSenderMsg);
    void handleGetName(unsigned long senderId,
                       unsigned long senderMsgLen,
                       byte *pSenderMsg);
private:
    CTestPlan m_pTestPlan;
    CTestPlanProxy m_pTestPlanProxy;
    typedef void (CFuncTestMsgHandler::*handlerFn)(unsigned long, unsigned long, byte*);
    std::map<int, handlerFn> m_handlers;
}
File: - TestPlanMsgHandler.cpp
CTestPlanMsgHandler::CtestPlanMsgHandler
{
    m_handlers[HandleError] = handleError;
    m_handlers[GetName] = handleGetName;
    m_handlers[SetName] = handleSetName;
}
```

```
void
CTestPlanMsgHandler::handleMessage(unsigned long msgType,
                unsigned long senderId,
                unsigned long senderMsgLen,
                byte *pSenderMsg)
{
    if (msgType == 0)
    {
        handleError(senderId, senderMsgLen, pSenderMsg);
    }
    else
    {
        handlerFn fn = NULL;
        hIter_t fIter;
        fIter = m_handlers.find(msgType);
        if (fIter == m_handlers.end())
        {
            return;
        }
        fn = fIter→second;
        if (NULL != fn)
        {
            (this→*fn)(senderId, senderMsgLen, pSenderMsg);
        }
    }
}
void
CTestPlanMsgHandler::handleSetName(unsigned long senderId,
                        unsigned long senderMsgLen,
                        byte *pSenderMsg)
{
    if (m_pTestPlanProxy != NULL)
    {
        OFCString tplName = ByteToString(senderMsgLen, pSenderMsg)
        m_pTplProxy→setName(tplName);
    }
}
void
CTestPlanMsgHandler::handleGetName(unsigned long senderId,
                        unsigned long senderMsgLen,
                        byte *pSenderMsg)
{
    OFCString testName;
    if (m_pTestPlan != NULL)
    {
        unsigned long l_destId
        unsigned long l_msgType;
        unsigned long l_senderId;
        unsigned l_senderMsgLen;
        byte *l_senderMsg = NULL;
```

```
            if (m_pTestPlan→getName(testName) != true)
            {
                // If a failure has occurred Send error message
                char *errorString = "Error retrieving name";
            l_destId = senderId;
            l_msgType = HandleError;
            l_senderId = m_Id;
            l_senderMsgLen = strlen(errorString);
            l_senderMsg = StringToByte(errorString);
            sendMsg(l_destId,
                    l_msgType,
                    l_senderId,
                    l_senderMsgLen,
                    l_senderMsg);
                return;
            }
            l_destId = senderId;
            l_msgType = SetName;
            long l_senderId = m_Id;
            l_senderMsgLen = testName.length();
            l_senderMsg = NULL;
            StringToByte(testName, &l_senderMsg);
            sendMsg(l_destId,
                    l_msgType,
                    l_senderId,
                    l_senderMsgLen,
                    l_senderMsg);
            DELETE_BYTE(l_senderMsg);
        }
    }
    void
    CTestPlanMsgHandler::handleError(unsigned long senderId,
                        unsigned long senderMsgLen,
                        byte *pSenderMsg)
    {
        OFCString errorString;
        ByteToString(senderMsgLen, pSenderMsg, errorString);
        m_pTestPlanProxy→setError(errorString);
    }
```

The following CTestPlanProxy class shows how a typical Proxy class will look like.

```
    File: - TestPlanProxy.h
    Class CTestPlanProxy
    {
        CTestPlanProxy(unsigned long serverId);
        ~CTestPlanProxy();
    private:
```

```
        CTestPlanProxy();
        unsigned long m_Id;
        unsigned long m_serverId;
        CTestPlanMsgHandler m_tplMsgHandler;
    }
    File: - TestPlanProxy.cpp
    extern CcmdDispatcher *g_pCmdDispatcher;
    CTestPlanProxy::CTestPlanProxy(unsigned long serverId)
    {
        m_serverId = serverId;
        m_tplMsgHandler.setTestPlanProxy(this);
        g_pCmdDispatcher.AddMsgHandler(&m_tplMsgHandler)

        /// initialize the proxy with its name.
        unsigned long msgType;
        unsigned long senderMsgLen;
        byte *pSenderMsg = NULL;
        msgType = GetName;
        senderMsgLen = 0;
        pSenderMsg = NULL;
        sendMsg(m_clientId,
                msgType,
                m_Id,
                senderMsgLen,

                    pSenderMsg);
        // Check if the error string has been set by the message handler.
        if (m_errorString.length() != 0)
        {
            OFCString errorString = m_errorString;
            m_errorString = "";
            throw exception(errorString.c_str());
        }
    }
    CTestPlanProxy::~CTestPlanProxy
    {
        g_pCmdDispatcher.removeMsgHandler(m_Id)
    }
```

The g_pCmdDispatcher object should be initialized by calling getCmdDispatcher().

システム構成とテスト

【０１０７】

図８は、本発明の一実施形態による名目上のテストシーケンス８００を示している。テストシーケンス８００は、テスト環境８０４におけるモジュールの設置８０２を含んでおり、これはテスト準備８０６とシステムテスト８０８とを包含している。初めに新しいモジュール（ハードウェアまたはソフトウェアまたはこれらの組み合わせ）８１０が（ベンダの品質管理に基づいているかもしれないいくつかの外部手順によって）認証８１２され

る。設置８０２はまず、オフラインシミュレーション８１０のためのハードウェアモジュールエミュレーションの設定、テストプログラム開発８１４のためのモジュールリソースファイルおよびインタフェースの設定、ならびにパターンコンパイル８１４のためのモジュール特有のパターンコンパイラの設定を含むテスト準備８０６を必要とする。次にシステムテスト８０８が、較正８１６、診断８１８および構成８２０からの入力を用いて実行される。そして新しいモジュールに対して、（１）インタフェース制御、（２）同期、順序付けおよび再現性、（３）エラー／アラーム対応、（４）マルチサイト制御、ならびに（５）マルチインストゥルメントモジュール制御を含むシステムテスト８０８が行われる。

【０１０８】

以上、本発明を実施の形態を用いて説明したが、本発明の技術的範囲は上記実施の形態に記載の範囲には限定されない。上記実施の形態に、多様な変更または改良を加えることが可能であることが当業者に明らかである。その様な変更または改良を加えた形態も本発明の技術的範囲に含まれ得ることが、特許請求の範囲の記載から明らかである。

【図面の簡単な説明】

【０１０９】

【図１】図１は従来のテスタアーキテクチャを示している。

【図２】図２は本発明の一実施形態によるシステムアーキテクチャを示している。

【図３】図３は本発明の一実施形態によるソフトウェアアーキテクチャを示している。

【図４】図４は発明の一実施形態によるテストクラスの使用を示している。

【図５】図５は発明の一実施形態による、テスタシステムと異なるベンダが提供するモジュールリソースとの相互作用を示す統一モデリング言語（UML）ダイアグラムである。

【図６】図６は、サイトコントローラによって保持されているときのユーザのテストを管理するためのサイトコントローラオブジェクトの一実施形態を示している。

【図７】図７は、図６に示されているサイトコントローラオブジェクトを表しているシステムコントローラ側のオブジェクトの代理の一実施形態を示している。

【図８】図８は発明の一実施形態によるテスト環境を示している。

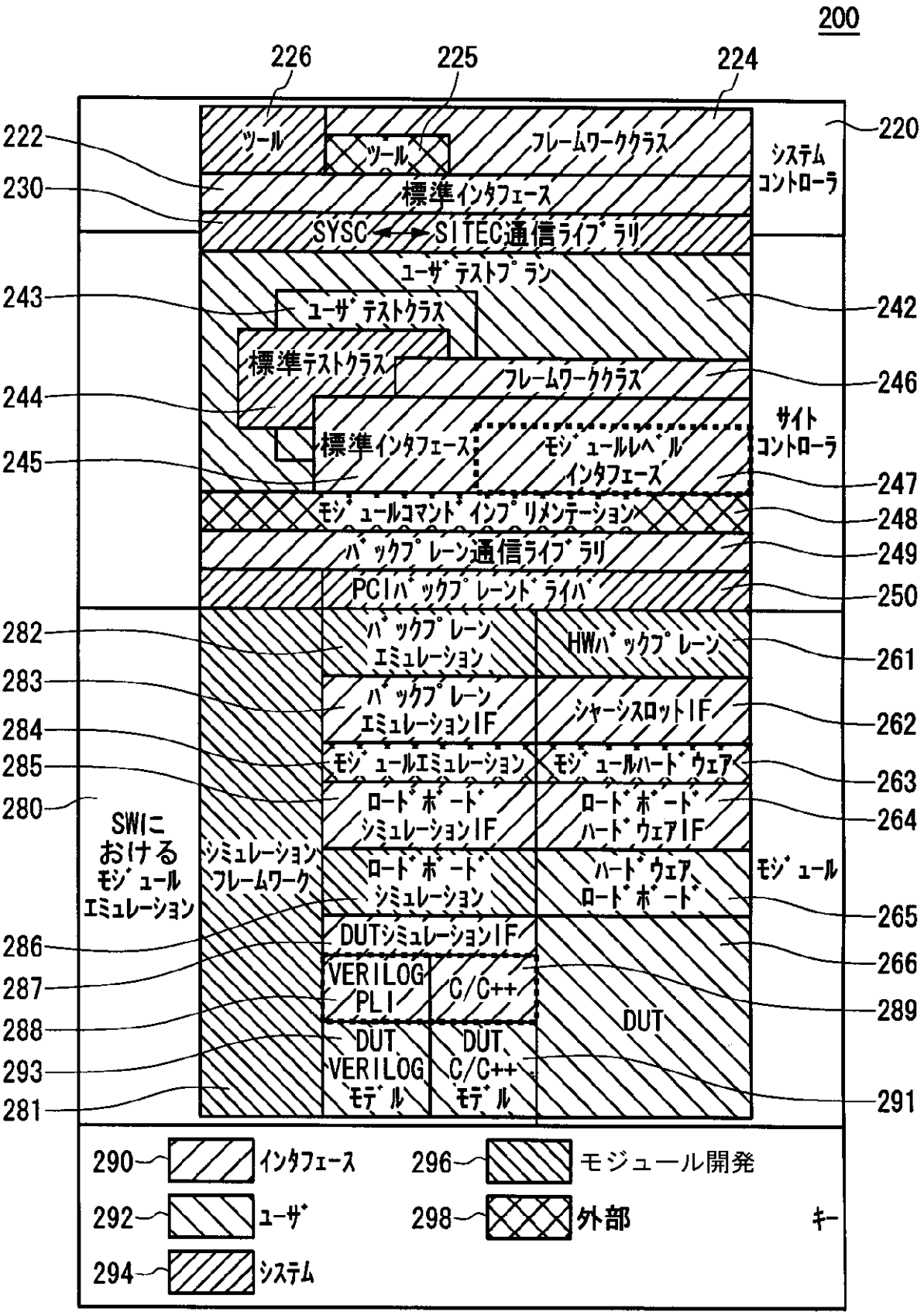
【符号の説明】

【０１１０】

１００　システムアーキテクチャ
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
１０８　モジュール
１１０　テストサイト
２００　ソフトウェアアーキテクチャ
２４０　サイトコントローラソフトウェア
２４２　ユーザテストプラン
２４３　ユーザテストクラス
２４４　標準テストクラス
２４６　フレームワーククラス
２４５　標準インタフェース
２４７　モジュールレベルインタフェース
２４８　モジュールコマンドインプリメンテーション
２４９　バックプレーン通信ライブラリ
２５０　ＰＣＩバックプレーンドライバ
２６０　モジュール
２６１　ＨＷバックプレーン
２６２　シャーシスロットＩＦ
２６３　モジュールハードウェア

２６４　ロードボードハードウェアＩＦ
２６５　ハードウェアロードボード
２８０　ＳＷにおけるモジュールエミュレーション
２８１　シミュレーションフレームワーク
２８２　バックプレーンエミュレーション
２８３　バックプレーンシミュレーションＩＦ
２８４　モジュールエミュレーション
２８５　ロードボードシミュレーションＩＦ
２８６　ロードボードシミュレーション
２８７　ＤＵＴシミュレーションＩＦ
２９０　オペレーティングシステムインタフェース
２９１　ＤＵＴ　C/C++モデル
２９２　ユーザコンポーネント
２９３　ＤＵＴ　Verilogモデル
２９４　システムコンポーネント
２９６　モジュール開発コンポーネント
２９８　外部コンポーネント
８０４　テスト環境
８１７　較正
８１８　診断
８２０　構成
８０６　テスト準備
８１４　テストプログラム開発
８０８　テストシステム
８１０　新しいＨＷまたはＳＷモジュール
８１２　認証
【誤訳訂正３】
【訂正対象書類名】図面
【訂正対象項目名】図３
【訂正方法】変更
【訂正の内容】

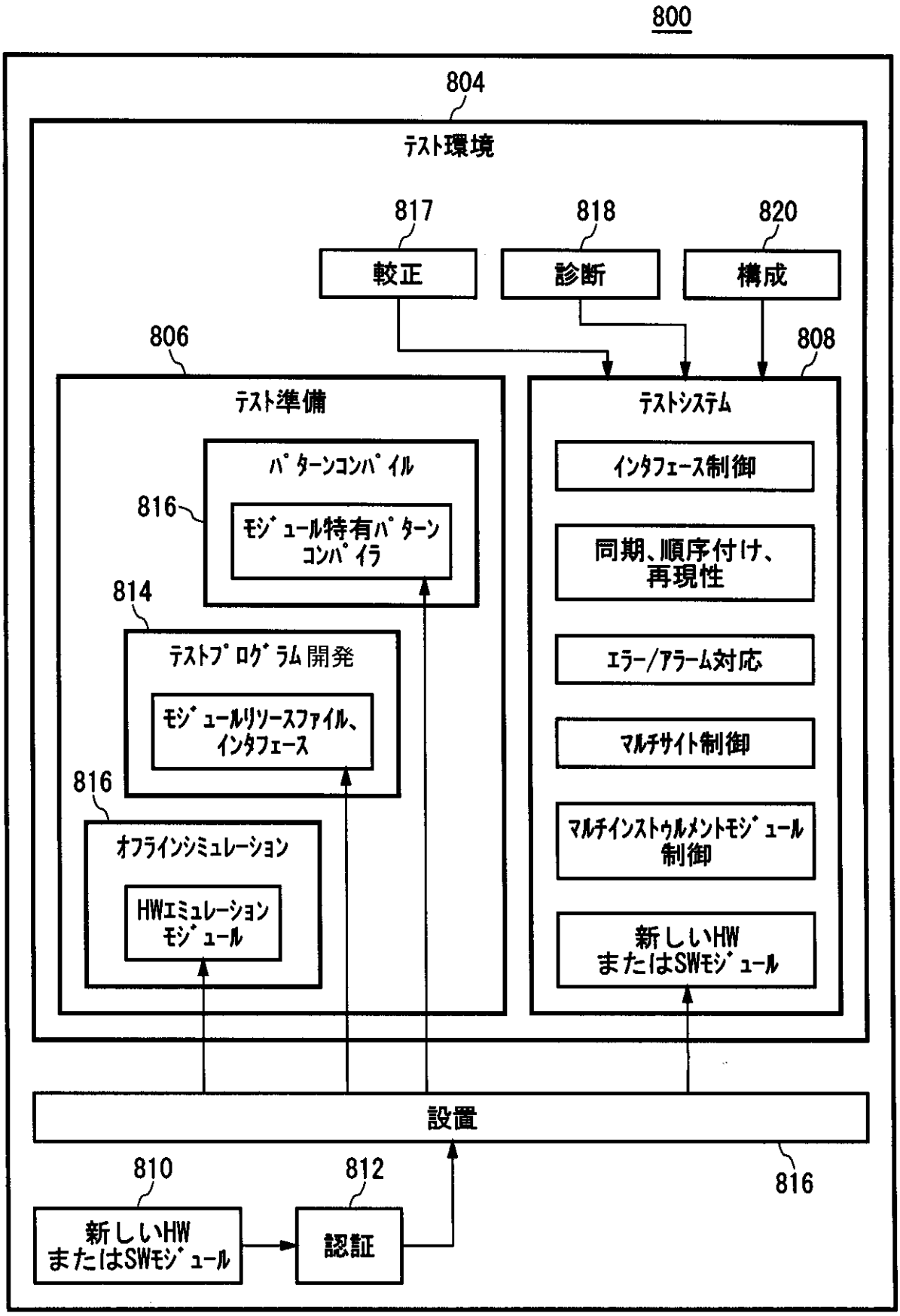
【図３】



【誤訳訂正４】
【訂正対象書類名】図面

【訂正対象項目名】図８
【訂正方法】変更
【訂正の内容】

【図８】


800
804
テスト環境
817
818
820
較正
診断
構成
806
808
テスト準備
テストシステム
パターンコンパイル
816
モジュール特有パターンコンパイラ
814
テストプログラム開発
モジュールリソースファイル、インタフェース
816
オフラインシミュレーション
HWエミュレーションモジュール
インタフェース制御
同期、順序付け、再現性
エラー/アラーム対応
マルチサイト制御
マルチインストゥルメントモジュール制御
新しいHWまたはSWモジュール
設置
816
810
812
新しいHWまたはSWモジュール
認証