(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3939336号
(P3939336)
(45)発行日　平成19年7月4日(2007.7.4)　(24)登録日　平成19年4月6日(2007.4.6)

(51)Int.Cl.		F I		
G06F 11/22	(2006.01)	G06F	11/22	310B
G01R 31/28	(2006.01)	G06F	11/22	330B
		G01R	31/28	H

請求項の数 12　(全 160 頁)

(21)出願番号	特願2006-502670 (P2006-502670)
(86)(22)出願日	平成16年2月16日 (2004.2.16)
(65)公表番号	特表2006-520947 (P2006-520947A)
(43)公表日	平成18年9月14日 (2006.9.14)
(86)国際出願番号	PCT/JP2004/001649
(87)国際公開番号	WO2004/072670
(87)国際公開日	平成16年8月26日 (2004.8.26)
審査請求日	平成18年2月8日 (2006.2.8)
(31)優先権主張番号	60/447,839
(32)優先日	平成15年2月14日 (2003.2.14)
(33)優先権主張国	米国 (US)
(31)優先権主張番号	60/449,622
(32)優先日	平成15年2月24日 (2003.2.24)
(33)優先権主張国	米国 (US)

(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　100104156
弁理士　龍華　明裕
(72)発明者　クリシュナスワミー　ラマチャンドラン
東京都練馬区旭町１丁目３２番１号　株式会社アドバンテスト内
(72)発明者　シング　ハルサンジート
東京都練馬区旭町１丁目３２番１号　株式会社アドバンテスト内
(72)発明者　プラマニック　アンカン
東京都練馬区旭町１丁目３２番１号　株式会社アドバンテスト内

最終頁に続く

(54)【発明の名称】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

(57)【特許請求の範囲】

【請求項１】

テスト対象装置（ＤＵＴ）をテストする半導体テストシステムであって、

前記半導体テストシステムを管理するシステムコントローラと、

それぞれが前記システムコントローラに連結され、それぞれのサイトコントローラを含むテストサイトに関連付けられた少なくとも１つの前記ＤＵＴのテストをそれぞれ制御する複数のサイトコントローラと、

それぞれが前記ＤＵＴに対して信号を与え、前記ＤＵＴが出力する信号を受け取って前記ＤＵＴをテストする複数のハードウェアモジュールと

を備え、

前記システムコントローラとして機能するコンピュータは、

前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行することにより、前記複数のハードウェアモジュールのそれぞれに関連するパターンソースファイルをコンパイルして、当該ハードウェアモジュールに特有のフォーマットを有するパターンデータを生成する

半導体テストシステム。

【請求項２】

前記コンピュータは、前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行することにより、前記複数のハードウェアモジュールについて生成した複数の前記パターンデータを、オブジェクトメタファイルに格納する請求項１に記載の半

導体テストシステム。
【請求項３】
前記コンピュータは、前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行することにより、前記複数のパターンデータのそれぞれに対応付けて、当該パターンデータが対応する前記ハードウェアモジュールを特定するヘッダ情報を更に前記オブジェクトメタファイルに格納する請求項２に記載の半導体テストシステム。
【請求項４】
前記サイトコントローラは、前記ＤＵＴのテストに先立って、前記オブジェクトメタファイルから前記複数のハードウェアモジュールのそれぞれに特有の前記パターンデータを読み出して、対応する前記ハードウェアモジュールにロードする請求項３に記載の半導体テストシステム。
【請求項５】
前記サイトコントローラは、複数の前記パターンデータの順序をツリー構造により定義したパターンリストファイルに基づいて、前記複数のパターンデータの実行シーケンスを生成し、
前記複数のハードウェアモジュールは、前記実行シーケンスに応じた順序で前記複数のパターンデータによるテストを実行する請求項４に記載の半導体テストシステム。
【請求項６】
前記複数のサイトコントローラおよび前記複数のハードウェアモジュールを接続するモジュール接続イネーブラを更に備え、
前記コンピュータは、前記複数のサイトコントローラのそれぞれが接続された前記モジュール接続イネーブラの入力ポートを記述したテストシステムコンフィギュレーションファイルと、前記複数のハードウェアモジュールのそれぞれが接続された前記モジュール接続イネーブラの出力ポートを指定するスロット識別子を記述したモジュールコンフィギュレーションファイルとに基づいて、前記サイトコントローラを、当該サイトコントローラに対応するテストサイトに関連付けられた少なくとも１つの前記ハードウェアモジュールに接続させるべく前記モジュール接続イネーブラの接続を構成する
請求項３に記載の半導体テストシステム。
【請求項７】
前記コンピュータは、それぞれのテストサイトに対応する、複数の前記ＤＵＴのそれぞれと、当該ＤＵＴをテストする前記ハードウェアモジュールが接続された前記モジュール接続イネーブラの出力ポートを指定するスロット識別子を記述したソケットファイルに基づいて、それぞれのテストサイトに含まれる前記ハードウェアモジュールを判断する請求項６に記載の半導体テストシステム。
【請求項８】
前記コンピュータは、前記モジュール接続イネーブラの構成を制御する一の前記サイトコントローラに対して前記モジュール接続イネーブラの接続の構成を要求することにより、当該一のサイトコントローラを介して前記モジュール接続イネーブラの接続を構成する請求項６に記載の半導体テストシステム。
【請求項９】
前記モジュール接続イネーブラはスイッチマトリクスである請求項６に記載の半導体テストシステム。
【請求項１０】
テスト対象装置（ＤＵＴ）をテストする半導体テストシステム用のプログラムであって、
当該プログラムは、前記半導体テストシステムを、
前記半導体テストシステムを管理するシステムコントローラと、
それぞれが前記システムコントローラに連結され、それぞれのサイトコントローラを含むテストサイトに関連付けられた少なくとも１つの前記ＤＵＴのテストをそれぞれ制御する複数のサイトコントローラと、

それぞれが前記ＤＵＴに対して信号を与え、前記ＤＵＴが出力する信号を受け取って前記ＤＵＴをテストする複数のハードウェアモジュールと
して機能させ、
前記システムコントローラとして機能するコンピュータにより、前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行させることにより、前記複数のハードウェアモジュールのそれぞれに関連するパターンソースファイルをコンパイルさせて、当該ハードウェアモジュールに特有のフォーマットを有するパターンデータを生成させる
プログラム。

【請求項１１】
前記コンピュータにより、前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行させることにより、前記複数のハードウェアモジュールについて生成した複数の前記パターンデータを、オブジェクトメタファイルに格納させる請求項１０に記載のプログラム。

【請求項１２】
前記コンピュータにより、前記複数のハードウェアモジュールのそれぞれに特有のパターンコンパイラを実行させることにより、前記複数のパターンデータのそれぞれに対応付けて、当該パターンデータが対応する前記ハードウェアモジュールを特定するヘッダ情報を更に前記オブジェクトメタファイルに格納させる請求項１１に記載のプログラム。

【発明の詳細な説明】
【技術分野】
【０００１】
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造」、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している記録媒体」、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。
【０００２】
本発明は、集積回路（IC）をテストすることに関し、特に、自動半導体テスト機器（ATE）用のテストプログラムを開発することに関する。
【背景技術】
【０００３】
今日、テスタの製造者は、半導体テストシステム（テスタ）用のテストプログラムを開発するのに、彼らの独自仕様の言語を用いている。例えば、アドバンテストコーポレーションによって製造されたマシンは、テスト記述言語（TDL）を使用しており、クレデンスシステムは、自身の波形生成言語（WGL）を提供している。この専門化の度合いを克服するために、ICおよびテスタ製造者は、IEEE標準規格1450、標準テストインタフェース言語（STIL）を開発することによって共通の領域を見つけ出そうとしている。しかしながら、STILは、ピン、テストコマンド、タイミング等を定義するのに高度に特化した言語である。また、それにもかかわらずSTILを作動させるテストエンジニアは、いまだに、STILを、テスタが必要とする独自仕様の製造者特有の言語に変換する必要がある。したがって、STILは、それにもかかわらず高度に専門化され、プログラマに一般的に知られていない中間言語としてしか機能していない。
【０００４】
したがって、テストプログラムを汎用的な言語で書くことができる方法を開発することが望まれている。また、この方法は、オープンアーキテクチャテストシステム用のテストプログラムを簡単に開発することができるようにしなければならない。
【発明の開示】
【発明が解決しようとする課題】

【０００５】
本発明の目的は、テストプログラムを汎用的な言語で書くことができる方法を提供することである。本発明の他の目的は、上記方法で、オープンアーキテクチャテストシステム用のテストプログラムを簡単に開発することができるようにすることである。
【課題を解決するための手段】
【０００６】
本出願は、オブジェクト指向コンストラクト、例えばC++オブジェクトおよびクラスを用いたテストプログラム開発を記載している。特に、この方法は、本発明の譲受人に譲受された米国出願第60/449,622号、10/404,002号および10/403,817号に記載されているテスタのようなオープンアーキテクチャテスタ用のテストプログラムを開発するのに適している。
【０００７】
本発明の一実施形態は、自動テスト機器（ATE）のような半導体テストシステム上でテスト対象装置、例えばICをテストするために、テストシステムリソース、テストシステムコンフィギュレーション、モジュールコンフィギュレーション、テストシーケンス、テストプラン、テスト条件、テストパターンおよびタイミング情報を汎用のオブジェクト指向の、例えばC/C++のコンストラクトで記載することによってテストプログラムを開発する方法を提供する。これらの記述を含むファイルは、そのファイルを用いるテストシステムまたは関連する機器がアクセス可能であるメモリ、すなわちコンピュータで読み取り可能である媒体に記憶されている。
【０００８】
テストリソースを記述することは、ICにテストを提供する少なくとも一つのテストモジュールに関連しているリソースタイプを指定すること、リソースタイプに関連したパラメータタイプを指定すること、ならびにそのパラメータタイプのパラメータを指定することを包含していてもよい。
【０００９】
テストシステムコンフィギュレーションを記述することは、少なくとも一つのテストモジュールであってそれぞれがICに対してテストを適用するようなテストモジュールを制御するサイトコントローラを指定することと、モジュール接続イネーブラの入力ポートを指定することとを包含している。テストシステムは、その特定された入力ポートでサイトコントローラをモジュール接続イネーブラにつないで、モジュール接続イネーブラはサイトコントローラをテストモジュールにつなぐ。モジュール接続イネーブラは、スイッチマトリクスとしてインプリメントされてもよい。
【００１０】
モジュールコンフィギュレーションを記述することは、モジュールタイプを指定するモジュール識別子を指定することと、そのモジュール識別子によって指定されたモジュールタイプのテストモジュールを制御する実行可能なコードを指定することと、そのテストモジュールに関連したリソースタイプを指定することとを包含している。実行可能なコードは、ダイナミックリンクライブラリの形をとってもよい。
【００１１】
モジュールコンフィギュレーションを記述することはさらに、モジュール接続イネーブラの出力ポートを指定するスロット識別子をユーザが指定することを含んでいてもよく、テストシステムは、その出力ポートでモジュール接続イネーブラにテストモジュールを繋ぎ、モジュール接続イネーブラはテストモジュールを対応するサイトコントローラにつなぐ。またユーザは、テストモジュールの提供者を識別するためのベンダ識別子と、リソースタイプとともに使用可能である最大数のリソースユニットの識別子とを指定してもよい。例えば、リソースタイプは、デジタルテスタピンおよびリソースユニットテスタチャネルであってもよい。あるいは、テスタチャネルリソースユニットは、例えばアナログテスタピン、RFテスタピン、電源ピン、デジタイザピン、および任意波形発生ピンのようなリソースタイプに対応してもよい。どのリソースユニットが無効であるに関連するインジケ

ータも提供されてもよい。無効であると示されたリソースユニットは、テストモジュールの不良リソースユニットを表してもよい。

【００１２】

テスト条件を記述することは、少なくとも一つのテスト条件グループを指定することと、少なくとも一つの変数を含む仕様セットを指定することと、変数に結びつけられている式を選択するセレクタを指定することとを包含していてもよい。テスト条件グループと仕様セットのセレクタとの関連付けがテスト条件を定義する。

【００１３】

テストシーケンスを記述することは、さまざまなテストが適用され得る順序（あるいはフロー）を指定することを包含していてもよい。

【００１４】

テストパターンを記述することは、電圧レベルおよび電流レベル、信号値変化、ならびに対応する立ち上がり・立下り時間および関連するタイミングに関連して、テストパターンを指定することを包含していてもよい。

【００１５】

また、本発明の一実施形態は、プリヘッダファイルの使用を含んでいてもよい。プリヘッダファイルは、テスト対象（entiry）に関連したクラスのためのヘッダファイルを生成するようにコンパイルされる。プリヘッダは、テスト対象の少なくとも一つの属性を設定するためのパラメータを指定するパラメータブロックと、コンパイラによってテスト対象クラスのヘッダファイルに挿入されるソースコードを指定するテンプレートブロックとを含んでいる。ヘッダファイルは、C++ヘッダファイルであってもよい。例えば、テスト対象はテストであってもよく、テスト対象クラスはテストクラスであってもよい。パラメータは、例えばパターンリストおよびテスト条件に関連してもよい。

【００１６】

発明の一実施形態のパターンコンパイラは、少なくとも一つのモジュール特有パターンコンパイラと、各モジュール特有コンパイラにパターンソースファイルの対応するモジュール特有セクションとパターンソースファイルの共通セクションとの両方をコンパイルさせるオブジェクトファイルマネージャとを含んでいる。共通セクションは、モジュール特有コンパイラの全てがアクセス可能である情報を含んでいる。コンパイラの出力は、少なくとも一つのモジュール特有パターンデータセクションを含んでいる。モジュール特有パターンローダは、実行のために、対応するテストモジュールに、対応するモジュール特有パターンデータセクションからモジュール特有パターンデータをロードする。

【００１７】

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】

【００１８】

本発明は、同一の譲受人による米国出願第60/449,622号、10/404,002号および10/403,817号に開示されたオープンアーキテクチャテストシステムについて概略を説明される。しかしながら当業者は、本発明のテストプログラム開発システムおよび方法の実施形態は、オープンアーキテクチャにだけではなく、同様に固定されたテスタアーキテクチャにも適用可能であることを理解されたい。

【００１９】

オープンアーキテクチャのテストシステムの説明は、同一の譲受人による米国出願第60/449,622号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10/772,327号「集積回路をテストするための方法および装置」において見られる。

【００２０】

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。

【００２１】

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。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。ある実施形態においては、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。

【００２２】

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展開することもできる。

【００２３】

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。

【００２４】

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２２０、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。

【００２５】

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。

【００２６】

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。

【００２７】

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展開）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、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。

【００２８】

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。

【００２９】

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。

【００３０】

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。

【００３１】

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。

【００３２】

外部コンポーネント２９８は外部ツール２２５を含んでいる。

【００３３】

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。また、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。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にローカルであってもよく、あるいはオフライン、ネットワークを通じてシステムコントローラに接続されていてもよい。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。

【００３４】

サードパーティディベロッパは、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。

【００３５】

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見えないよう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。

【００３６】

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。

【００３７】

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上のツールに役立てられる。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。

【００３８】

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。

【００３９】

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。図４を参照すると、テストプログラム４００は、テストおよび関連するパラメータを記述するテストプログラム開発者ソースファイル４０４を、C++コードのようなオブジェクト指向コンストラクトに変換するためのトランスレータセクション４０２を含むコードジェネレータとして部分的に機能する。コンパイラセクション４０６は、コードを実行可能なもの、例えばDLLにコンパイル、リンクして、テスタシステムによって実行され得るテストプログラムを生成する。テストシステムに対するTPLコードジェネレータ／トランスレータを適用することは新規ではあるが、コードジェネレータは当分野で知られていることに留意されたい。また、コンパイラセクションも、標準的なC++コンパイラとして当分野で知られているかもしれない。

【００４０】

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。

【００４１】

テストプログラム４００へ入力されるソースコードは、テストプランにおいて用いられるオブジェクトを指定し、これらの間の関係を指定するテストプラン記述ファイルを含んでいる。このファイルは、ITestPlanとして表される標準インタフェースのインプリメンテーションの形でサイトコントローラ上で実行されるC++コードに変換される。このコードは、ウィンドウズダイナミックリンクライブラリ（DLL）にパッケージ化されて、サイトコントローラ上にロードされ得る。テストプログラムDLLは、サイトコントローラソフトウェアが、それが含むTestPlanオブジェクトを生成して戻すために用いることができる標準的な公知のエントリポイントを有するように生成される。サイトコントローラソフトウェアは、テストプログラムDLLをその処理空間にロードし、テストプランオブジェクトのインスタンスを生成するためにエントリポイントの一つを用いる。一旦テストプランオブジェクトが生成されたら、サイトコントローラソフトウェアはそのテストプランを実行することができる。

【００４２】

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。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ギングの方法を提供してもよい。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。

【００４３】

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。

【００４４】

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。

【００４５】

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。

【００４６】

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。

【００４７】

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。

【００４８】

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インスタンスを作り出すことが可能である。図５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インスタンスが導き出されるかを示している。これらのテストは、C++コードのようなオブジェクト指向コンストラクトで直接プログラムされてもよいし、テストプログラムコンパイラがテストプランファイルからのテストおよびパラメータの記述をとりあげて、対応するC++コードを生成することを可能にするように設計されてもよい。生成されたC++コードはテストプログラムを生成すべくコンパイル、リンクされ得る。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変更してもよい。

【００４９】

ユーザによって展開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。

【００５０】

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対象物である。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。

【００５１】

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対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。

【００５２】

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。

【００５３】

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。

【００５４】

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展開のための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。

【００５５】
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。
【００５６】
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。
【００５７】
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。
【００５８】
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。
【００５９】
このオープンアーキテクチャを背景に用いて、本発明のテストプログラム開発システムを以下でさらに説明する。下記セクションＡはテストプログラムが用いられるテスト環境を記述するルールを説明し、セクションＢはテストプログラム開発用の方法およびルールを説明し、セクションＣは、テストプランを開発する方法およびルールと、どのようにしてテストプログラムの主要構造を定義するかとを指定し、セクションＤはオープンアーキテクチャテストシステム上でどのようにテストプログラムを動作させるかを説明し、セクションＥはテストパターン用の方法およびルールを説明し、セクションＦはテストパターンのタイミングを説明し、セクションＧは全体的なテスト動作のルールを説明する。

Ａ．コンポーネント
【００６０】
テスト環境は、テスタを持ち出して、それをテストのセットを動作させるように準備するために必要な条件を指定するファイルのセットを備えている。好ましくは、テスト環境は以下のファイルを備えている。
１．テスタリソース定義：　オープンアーキテクチャテストシステムで利用可能であるテスタリソースのタイプ――およびこのようなりソースについてのサポートされるパラメータ――の指定のため
２．テスタコンフィギュレーション：　サイトコントローラ、サイトおよび対応するマッピングの指定のため
３．モジュールコンフィギュレーション：　各サイトにおけるハードウェアモジュールの指定のため

４．ピン記述：　信号ピン、電源等のDUTピンのネーミングのため、ならびにピングループを記述するため
５．ソケット：　DUTピン－テスタピンの割り当ての指定のため
６．ピンオプション：　ピンについての特別なオプション、またはモードの指定のため
７．パターンリスト：　テストパターンおよびそのシーケンスの指定のため
８．パターン：　テストベクトルの指定のため

【００６１】

以上のうち、項目１～３は、ICF（設定およびコンフィギュレーションファイル）によってCMD（コンフィギュレーションマネジメントデータベース）からの情報を用いて生成され、既知の配置において利用可能とされ、項目４～８はユーザによって指定される。このセクションは、上記項目１～６を説明し、項目７～８はセクションEにおいてより詳細に説明する。好ましくは、具体的な方法およびルールは、これらのコンポーネントのそれぞれを開発するために用いられ、これらの方法およびルールは、このセクションにおいて例とともに説明される。

Ａ１．　リソース定義

【００６２】

各ハードウェアモジュールは、テストシステムが使用するための一つ以上のハードウェアリソース（短く、リソースとよぶ）を提供する。好ましくは、テスタのリソース定義は、使用可能であるリソースタイプのリソース名のセットと、各特定のリソースタイプに関連するパラメータ名およびタイプのセットとを宣言するために用いられる。例えば、リソース名dpinはデジタルテスタピンを指すために用いられる。これらのリソースは、VIL（低い入力電圧用）、VIH（高い入力電圧用）、VOL（低い出力電圧用）、VOH（高い出力電圧用）等のようなパラメータを有する。リソース定義ファイルは、拡張子「.rsc」を有する。以下に、いくつかのテストリソースを含むリソース定義の例を示す。

```
#
# File Resources.rsc
#
Version 0.1.2;
ResourceDefs
{
    # digital pins
    dpin
    {
        # Low and High voltages for input pins
        Voltage VIL, VIH;
        # Low and High voltages for output pins
        Voltage VOL, VOH;
    }
    # power supplies
    dps
    {
        #
        # PRE_WAIT specifies the time to wait after voltage
        #        reached its final value to start pattern
        #        generation.  The actual time that the system
        #        will wait is a small system specified range:
        #         PRE_WAIT-delta <= actual <= PRE_WAIT+delta
        #
        # PRE_WAIT_MIN is a minimum amount to wait after voltage
```

```
        #       reached its final value to start pattern generation.
        #       It is a system specified range:
        #        PRE_WAIT_MIN <= actual <= PRE_WAIT_MIN+delta
        #
        # POST_WAIT specifies the time to wait after pattern
        #       generation ends to shut down the power.  The actual
        #        time that the system will wait is a small system
        #        defined range:
        #       POST_WAIT-delta <= actual <= POST_WAIT+delta
        #
        # POST_WAIT_MIN specifies the time to wait after pattern
        #       generation ends to shut down the power.  The actual
        #        time that the system will wait is a small system
        #        defined range:
        #       POST_WAIT_MIN <= actual <= POST_WAIT_MIN+delta
        #
        Time PRE_WAIT;
        Time PRE_WAIT_MIN;
        Time POST_WAIT;
        Time POST_WAIT_MIN;
        # The voltage.
        Voltage VCC;
    }
}
```

Ａ２．　テスタコンフィギュレーション

【００６３】

テスタコンフィギュレーションは、好ましくは特定のシステムコンフィギュレーションにおいてサイトコントローラとスイッチマトリクス入力ポートへのサイトコントローラの接続とをリストするために用いられるルールのセットである。発明の一実施形態によるアーキテクチャにおいては、単一のサイトコントローラを単一のスイッチマトリクス入力ポートに接続することができる。したがって、この文脈では、スイッチマトリクス接続は、システムにおいてサイトコントローラのための暗黙の識別子として働く（他の構成も可能である）。典型的なテスタコンフィギュレーションの例を以下に示す。

```
#
#  Tester Configuration, Sys.cfg
#
Version 1.2.5;
SysConfig
{
    #
    # The first field is the hostname of the Site Controller machine;
    # it can be specified as either a dotted-decimal IP address or a
    # domain-qualified hostname.
    #
    # The second field is the switch matrix input port number, which
    # implicitly serves as the identifier for the Site Controller
    # connected to it.
    #
    zeus.olympus.deities.org        2;
```

```
    127.0.0.2                       4;
    127.0.0.0                       1; # SITEC-1
    127.0.0.3                       3;
}
```

【００６４】

特定のテストフロアシステム用のシステムコンフィギュレーションは、システムプロファイルの一部であり、システムコンフィギュレーションファイルSys.cfgとして利用可能とされる。ある実施形態においてはポート１（上記例では127.0.0.0）に接続されたサイトコントローラは、単独でスイッチマトリクスを構成するという特別な状態を享受してもよい。この「特別な」サイトコントローラは、SITEC-1と称される。また、サイトコントローラはインターネットネットワークによってシステムコントローラに接続され得るので、この例におけるサイトコントローラのアドレスはIPアドレスであることに留意されたい。逆に、システムコントローラは、パターンデータのようなファイルにアクセスするために外部のネットワークに接続されてもよい。

テスタコンフィギュレーションのための構造

【００６５】

以下に、本発明の一実施形態によるシステムコンフィギュレーションファイルの構造を示す。

```
    version-info system-config
version-info:
    Version version-identifer ;
system-config:
    SysConfig { site-controller-connection-list }
site-controller-connection-list:
    site-controller-connection
    site-controller-connection-list site-controller-connection
site-controller-connection:
    site-controller-hostname input-port ;
site-controller-hostname:
    ip-address
    domain-qualified-hostname
ip-address:
    octet . octet . octet . octet
domain-qualified-hostname:
    name
    domain-qualified-hostname . name
```

上で定義されていないターミナル以外のものは以下の通り指定される：

【００６６】

１．　version-identifier：セット[0-9a-zA-Z.]からの一つ以上の文字の列。バージョン番号を表す。

２．　octet：0から255までの負でない整数（１０進法で）

【００６７】

３．　name：セット[a-zA-A_09]からの一つ以上の文字の列。数字では始まらない。ドメイン限定のホストネームにおける名前セグメントを表している。

【００６８】

４．　input-port：10進法で、負でない整数。

Ａ３．　モジュールコンフィギュレーション

【００６９】

モジュールコンフィギュレーションにより、テスタの物理的なコンフィギュレーション、例えばSYSTEMシャーシにおける各モジュールの実際の配置およびタイプを指定することができる。これは、テスタバスコンフィギュレーションの動的な性質によって必要とされ、テスタバスアドレスの物理的なスロット配置へのマッピングを可能にする。この情報により、システム起動時間に起こるハードウェアディスカバリプロセスがSYSTEMコンフィギュレーションを認証することができる。スイッチマトリクスの各出力ポートは物理的なスロットを規定し、好ましくはこれは単一のハードウェアモジュールが占有される。発明の一実施形態による、ファイルModules.cfgにおいて指定されるモジュールコンフィギュレーションの一例を以下に示す。

```
#
# Module Configuration File, Modules.cfg
#
Version 0.0.1;
ModuleConfig
{
    #
    # A configuration definition which provides information about
    # the module type that is attached to slots 1-12 and 32-48.
    # Note that a  module might provide more than
    # a single type of resource.
    #
    Slot 1-12, 32-48                    # Switch matrix output ports
                                        # which use the configuration
                                        # defined below.
    {
        VendorID    1;                      # defined vendor code.
        ModuleID    1;                     # Vendor-defined id code.
        ModuleDriver mod1.dll;      # Module software.
        #
        # Resource named dpin specifies channels
        # for digital data.  The name dpin is not
        # a keyword.  It is simply the name of a hardware
        # resource, and is obtained from the resource
        # definition file.
        #
        Resource dpin
        {
            MaxAvailable    32;            # Resource units 1 .. 32.
        }
        Resource analog
        {
            MaxAvailable    16;            # Resource units 1 .. 16.
            Disabled        1-8;        # Disabled resources 1 .. 8.
                                        # So, enabled ones are 9 .. 16.
        }
    }
    #
    # A configuration definition which provides information about
```

```
    # the module type that is attached to slots 16-30, 50, and 61-64.
    #
    Slot 16-30, 50, 61-64
    {
        Resource dpin
        {
            MaxAvailable    32;                # Max available resource units.
            Disabled        3, 30-32;        # Disabled resources.
        }
        ModuleDriver        "module two.dll";
        VendorID            2;
        ModuleID            2;
    }
    #
    # A configuration definition, which provides information about
    # the module type that is attached to slots 65-66.
    #
    Slot 65-66
    {
        ModuleID            4;                # DPS module with 8 supplies.
        ModuleDriver        mod4.dll;
        VendorID            1;
        #
        # Resource type dps specifying resource units for a
        # Device Power Supply
        #
        Resource dps
        {
            MaxAvailable    4;
            Disabled        1;
        }
    }
}
```

【００７０】

先に述べたように、ある実施形態においては、スロットは、スイッチマトリクスの出力ポートのような、それを通じてハードウェアモジュールを接続することが可能であるコネクタのことを指す。各コンフィギュレーション定義は、一つ以上のスロットに取り付けられ得るモジュールについての情報を提供する。コンフィギュレーション定義において指定されたベンダID（VederID）は、ベンダに関連付けられた固有のIDである。モジュールID（ModuleID）は、このベンダによって提供されるモジュールのタイプを指す。テスタコンフィギュレーションにおいては同じモジュールIDの例がいくつかあり得る。モジュールドライバ（ModuleDriver）はモジュールを使用可能にするためのベンダによって供給されるDLLを指す。最後に、リソース（Resource）は、このモジュールによって使用可能にされるユニットを指しており、リソースタイプの名前を提供し、リソース名はリソース定義ファイルから得られる。

【００７１】

上記例は、モジュールコンフィギュレーションファイルにおける３つのコンフィギュレーションブロックを述べている。あるインプリメンテーションにおいては、最初のコンフィギュレーションブロック、スロット１～１２および３２～４８は、ベンダ１によって製造されたモジュールによって使用可能にされる。このベンダは、モジュールタイプを指す

識別子が「１」であるモジュールと、そのモジュールを制御するモジュールドライバライブラリとを提供する。このモジュールは、2つのタイプのリソースユニットを提供する。一つはリソース名「dpin」で表され、好ましくは全部で32個のリソースユニット（すなわちチャネル）を有し、これら全てが利用可能である。もう一つはリソース名「analog」で表され、全部で16個のリソースユニットを有しており、このうち9から16だけが利用可能である。第二および第三のコンフィギュレーションブロックは、第一のコンフィギュレーションと同様にして指定される。

【００７２】

チャネルが「無効である」と表されることを可能にするということは、別の点ではまだ機能する不良リソースユニットまたはモジュールを示すことができるということであるということに留意されたい。また、コンフィギュレーションブロックは一つ以上のスロット識別子を有し得ることにも留意されたい。ブロックが一つよりも多いスロット識別子を有するときには、識別されたスロットは複製されていると考えられる。

【００７３】

モジュールコンフィギュレーションファイルModule.cfgは、システムプロファイルの一部としてICM（設定コンフィギュレーションマネジメントシステム）によって（ユーザによって提供されるテストフロア特有の情報を用いて）作製され、既知の配置で利用可能とされる。ICMは、テストシステムに対してローカルである、例えばシステムコントローラ上、あるいはシステムコントローラが接続されているネットワーク上のどこかに存在しているユーティリティである。ICMは、CMD（コンフィギュレーションマネジメントデータベース）を管理し、典型的にはハードウェア上でシステムコンフィギュレーションに対する変化をアップデートした。ICMは、ユーザがシステム、例えばサイトコントローラおよびモジュールを構成することを可能にする。CMDはコンフィギュレーションを記憶するデータベースである。実際のテスタコンフィギュレーション／動作に関して、ICMは、例えばモジュールコンフィギュレーションのようなコンフィギュレーションファイルと他のファイルとを生成し、それらと、特定のモジュールDLLのような関連するファイルとをテスタ上にコピーする。

モジュールコンフィギュレーションの構造

【００７４】

好ましい実施形態によるモジュールコンフィギュレーションの構造を以下に示す。

```
file-contents:
    version-info  module-config-def
version-info:
    Version version-identifier ;
module-config-def:
    ModuleConfig { slot-entry-list }
slot-entry-list:
    slot-entry
    slot-entry-list slot-entry
slot-entry:
    Slot positive-integer-list { slot-info }
slot-info:
    required-config-list
required-config-list:
    required-config
    required-config-list required-config
required-config:
    VendorID  id-code ;
    ModuleID  id-code ;
```

```
    ModuleDriver  file-name ;
    Resource  resource-name  { max-spec disabled-spec_opt }
max-spec:
    MaxAvailable  positive-integer ;
disabled-spec:
    Disabled  positive-integer-list ;
positive-integer-list:
    positive-integer-list-entry
    positive-integer-list , positive-integer-list-entry
positive-integer-list-entry:
    positive-integer
    positive-integer-number-range
positive-integer-number-range:
    positive-integer - pos-integer
```

【００７５】

上で定義されていない非ターミナルを以下に述べる。

【００７６】

１．　version-identifie：セット[0-9a-zA-Z]からの一つ以上の文字の列。最初の文字はセット[0-9]からのものでなければならない。

【００７７】

２．　positive-integer：セット[0-9]からの一つ以上の文字の列。０で始まらない。

【００７８】

３．　id-code：セット[a-zA-A_0-9]からの一つ以上の文字の列。

【００７９】

４．　resource-name：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z]からのものでなければならない。コメントがサポートされる。コメントは「#」の文字で始まり、行末まで伸びている。

Ａ４．　ピン記述

【００８０】

DUTピン記述は、ピン記述ファイルを用いて記述される。ユーザは、ピン記述ファイルにおけるDUTピンの記述を提供し、これは「.pin」の拡張子を有している。このプレインテキストファイルは、少なくとも以下のものを含んでいる：DUTピン名のリスト、および名前を付けられたピングループの初期の定義。後者は定義されたDUTピン名を使用する（それらを、例えばプログラム的に後で改変、追加等することができるので「初期」である）。

【００８１】

このデータの指定をテストプランの記述とは別にすることによって、DUTピン定義の一般的な再利用が可能であり、パターンコントローラがピン名（ベクトル指定において用いられるピン名への言及を説明するのに必要とされる）を、プロセスを具体的なテストプランに結び付けることなく、プロセスピン記述ファイルから得ることが可能である。

【００８２】

ピン記述ファイルの一例を以下に示す。

```
#
# Pin description file, myDUT.pin.
#
# Note that this implicitly imports the resource
# configuration file,Resources.rsc.
#
```

```
Version 1.1.3a;
PinDescription
{
    Resource dpin
    {
        A0;
        A1;
        A2;
        A3;
        A4;
        # This syntax expands to the names "ABUS[1]" and "ABUS[2]"
        ABUS[1:2];
        A5;
        BBUS[1:8];
        DIR;
        CLK;
        Group Grp1
        {
            DIR, CLK, A0, A1, A2, A3, A4, BBUS[1:4]
        }
        Group Grp2
        {
            A5,
            #
            # The following line will expand to
            #"DIR, A1, A2, A4, A5, BBUS[2] ":
            #
            Grp1 - CLK - A0 - A3 - BBUS[1] - BBUS[3:4] + A5,
            BBUS[5:8]
        }
    }
    Resource dps
    {
        vcc1;
        vcc2;
        vcc3;
        Group PSG
        {
            vcc1, vcc2
        }
    }
}
```

【００８３】

DUTピンおよびピングループの定義は、コンパイラがピンおよびピングループの定義をレベル等の許されるパラメータ設定に相関させることを可能にするように、リソースタイプブロック内にカプセル化されることに留意されたい。

【００８４】

ピン記述について、以下の点に留意しなければならない。

【００８５】

１．　ピングループとピンとは、同一のネームスペースを共有しており、グローバルな

（例えばテストプラン）範囲を有している。これらの名前がグローバルな範囲を有することによる効果の一つは、ピンおよびピングループは、異なるリソースブロック内で宣言される場合であっても、重複する名前を使うことがないということである。

【００８６】

２．　少なくとも一つのリソース定義がピン記述ファイルにおいて必要である。

【００８７】

３．　少なくとも一つのピン名が各リソースにおいて定義されなければならない。

【００８８】

４．　ピン名およびグループ名はリソース境界内で固有のものであることが要求される。

【００８９】

５．　同一のピン名またはグループ名が、2つ以上のリソースについて定義され得る。しかし、同一のリソース内での重複は無視される。

【００９０】

６．　グループ定義に現れる全てのピン名およびグループ名は、そのリソース内で既に定義されたものでなければならない。

【００９１】

７．　もしあれば、グループ定義は、少なくとも一つのピン名あるいはグループ名を有していなければならない（すなわち、グループ定義は空であってはならない）。

【００９２】

８．　ピングループ定義は、前に定義されたピングループへの言及を含むことができる。

【００９３】

９．　ピングループ定義は、前に定義されたピンおよび／またはピングループの加算および減算のような演算セットを含むことができる。

ピン記述の構造

【００９４】

本発明の好ましい実施形態による、ピン記述のための構造を以下に示す。

```
pin-description-file:
    version-info pin-description
version-info:
    Version version-identifer ;
pin-description:
    PinDescription { resource-pins-def-list }
resource-pins-def-list:
    resource-pins-def
    resource-pins-def-list resource-pins-def
resource-pins-def:
    Resource resource-name { pin-or-pin-group-def-list }
pin-or-pin-group-def-list:
    pin-or-pin-group-def
    pin-or-pin-group-def-list  pin-or-pin-group-def
pindef-or-pin-groupdef:
    pin-def ;
    pin-group-def
pin-def:
    pin-name
    pin-name [ index : index ]
pin-group-def:
```

```
    Group pin-group-name { pin-group-def-item-list }
pin-group-def-item-list:
    pin-def
    pin-group-def-item-list , pin-def
```

【００９５】
上で定義されていない非ターミナルは、以下のように指定される。
【００９６】
１．　version-identifier：セット[0-9a-zA-Z]からの一つ以上の文字の列であり、バージョン番号を表す。
【００９７】
２．　resource-name：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、数字で始まらない。dpinあるいはdpsのようなリソース名を表す。
【００９８】
３．　pin-name：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、数字では始まらない。ピンA0の名前を表す。
【００９９】
４．　pin-group-name：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、数字では始まらない。ピングループABUSの名前を表す。
【０１００】
５．　index：関連するピンのグループの下方の限界あるいは上方の限界を表す。

Ａ５．　ソケット
【０１０１】
ソケットは、DUTピン名と物理的なテスタピン（チャネル）割り当てとの間のマッピングを指定する（物理的なテスタチャネル番号はモジュールコンフィギュレーションファイルで定義される）。異なるソケットは、異なるDUTパッケージおよび異なるロードボードコンフィギュレーション等をサポートするのに用いられ得ることに留意されたい。マルチDUTシステムについては、DUT／チャネル割り当てに関するソケット定義は、基本となるソケットの複数サイトへの「複製」をサポートすることができる。しかしながら、異なるソケット（すなわち、同じ論理ピンについての異なる物理的なマッピング）は、モジュールパーティションを尊重しなければならない。したがって、テスタチャネルに対するDUTピンの割り当てを提供することに加えて、ソケットは、サイトをパーティションで区切ることも実質的に定義する。ソケットファイルはしたがって、いくつかの個別のサイトソケットについての定義を含み得る。以下に、３つのDUTサイトを定義する一例のソケットファイルを示す。

```
Version 1.1.3
SocketDef
{
    DUTType Pentium3
    {
        PinDescription dutP3.pin; # The pin description file for Pentium3
        DUT 2 # Uses the full-specification syntax
        {
            SiteController 1; # Switch Matrix input port
            Resource dpin
            {
                #
                # The CLK pin is assigned to resource dpin,
```

```
        # slot 2, resource unit (channel) 13.
        #
        CLK        2.13;
        #
        # The DIR pin is assigned to resource dpin,
        # slot 5, resource unit 15.
        DIR        5.15;
        #
        # The following statement will be expanded to
        #        BBUS[7]    5.4
        #        BBUS[6]    5.5
        #        BBUS[5]    5.6
        #
        # So for example, the pin sequence BBUS[7], BBUS[6],
        # BBUS[5] is assigned to the same slot 5, and to
        # resource units 4, 5 and 6 respectively.
        #
        BBUS[7:5]        5.[4:6];
        BBUS[1:4]        7.[21:18];
        BBUS[8]        9.16;
    }
    Resource dps
    {
        #
        # The V1 pin is assigned to resource dps,
        # slot 1, resource unit (channel) 1.
        #
        VCC1    1.1;
        #
        # The VCC2 pin is assigned to resource dps,
        # slot 1, resource unit (channel) 2.
        #
        VCC2    1.2;
    }
} # End DUT 2
DUT 1 # This is "cloned" from DUT 2 above
{
    SiteController 1; # Same Site Controller as for DUT 2
    Resource dpin
    {
        SlotOffset 1;        # Offset value for slots
    }
    Resource dps
    {
        SlotOffset 10;    # Offset value for slots
    }
    #
    # The offset syntax above indicates that the slot/resource
    # unit assignments are "cloned" from the first DUT defined
    # for this DUTType, i.e., DUT 2, with the slots offset by
```

```
            # the SlotOffset values.
            #
            # Looking at the definition of dpin resource units for
            # DUT 2, CLK is bound to slot 2.  Hence, for the present
            # DUT, CLK is bound to slot 2 + 1 = 3.
            #
            # Some of the new bindings in effect due to the offset
            # assignments are shown in the table below:
            #
            # ------------------------------------------------
            #        Pin            Resource        RUnit    Slot
            # ------------------------------------------------
            #        CLK            dpin            13        2 +  1 =  3
            #        DIR            dpin            15        5 +  1 =  6
            #        BBUS[8]    dpin            16        9 +  1 = 10
            #        VCC1        dps                1        1 + 10 = 11
            #        VCC2        dps                2        1 + 10 = 11
            #
        } # End DUT 1
    } # End DUTType Pentium3
    DUTType 74LS245
    {
        PinDescription dutLS.pin;
        DUT 3 disabled # This DUT site is disabled, and will be ignored
        {
            ...
        }
    } # End DUTType 74LS245
} # End SocketDef
```

【０１０２】

ソケットファイルについては以下の点に留意すべきである。

【０１０３】

１．　ソケットファイルは、モジュールコンフィギュレーションファイルと、与えられたDUTタイプについてのユーザピン記述ファイルとの両方からの情報を用いる（上記例のピン記述の指定を参照）。モジュールコンフィギュレーション情報は、ソケットファイルコンパイラに暗黙のうちに役立てられる。ソケットファイルコンパイラは、パターンコンパイラによって用いられるDUTピンに対するテスタピンのマッピングを設定するためにソケットDUT名対テスタチャネルマッピングとモジュールコンフィギュレーションファイルおよびピン記述ファイルとを読み出し、解析するパターンコンパイラのサブパートである。

【０１０４】

２．　DUTタイプごとに少なくとも一つのDUT定義が必要とされ、それはSlotOffsetシンタックス（syntax）ではなく、全指定シンタックスを用いなければならない。もし２つ以上のDUTサイト定義が同一のDUTタイプについて提供されるのなら、最初のものが完全指定シンタックスを用いていなければならない。

【０１０５】

３．　その後のDUTサイト定義（同一のDUTタイプについての）のそれぞれは、完全指定シンタックスかSlotOffsetシンタックスを用いてもよいが、両方を用いてはならない。これにより、個々のサイトが（例えば動作不能のチャネルのせいで）標準的なパターンからはずれることが可能である。

【０１０６】
４．　SlotOffsetシンタックスから得られる結合は　、そのDUTタイプについて定義された第一のサイト（完全指定シンタックスを用いている）に関連して定義される。
【０１０７】
５．　DUTサイトは、実際の物理的な順番で宣言される必要はない。これにより、第一の（物理的な）サイトがパターンから外れる場合が可能である。
【０１０８】
６．　DUTサイトのIDは、ソケット全体にわたって（すなわちそこで定義される全てのDUTタイプにわたって）固有のものであることが要求される。
【０１０９】
７．　少なくとも一つの定義がDUTサイト定義ごとに必要とされる。
【０１１０】
８．　サイト定義は、テストコンフィギュレーションが単一サイト／単一DUTか、単一サイト／マルチDUTかを判断するために、モジュールコンフィギュレーションと関連して用いられなければならない。
【０１１１】
９．　全ての場合において、ソケットファイルは、ピン記述ファイルとモジュールコンフィギュレーションファイルとに一致したDUTチャネルマッピングのセットを指定しなければならない。
【０１１２】
１０．いくつかの場合において、ソケット定義が、一つ以上のDUTチャネルがテスタから切断されていることを（例えば割り当てられる物理的なチャネルを特別なID「0.0」を有するものとすることによって）指定することを可能にすることが望ましい。この場合、これらのDUTチャネルは、テストプログラムの関連で用いられ、参照される。このようなチャネル上での動作は、システム警告という結果をもたらす（エラーではない）。ロード時には、切断されたチャネルについてのパターンデータは捨てられる。

ソケットの構造
【０１１３】
本発明の好ましい実施形態によるモジュールコンフィギュレーションの構造を以下に示す。

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

version-info:
    Version version-identifer ;

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

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

device-specific-socket-def:
    DUTType DUT-type-name { pin-description-file dut-info-list }

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

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

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

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

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

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

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

resource-item-unit-assignment:
    resource-item-name slot-number . resource-unit ;
    resource-item-name [ resource-item-index ] slot-number . resource-unit-index
 ;
    resource-item-name [ resource-item-index-range ]    \
        slot-number . [ resource-unit-index-range ] ;

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

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

【０１１４】

上で定義されていない非ターミナルは以下の通り指定される。

【０１１５】

１．　version-identifier：セット[0-9a-zA-Z]からの一つ以上の文字の列であり、バージョン番号を表す。

【０１１６】

２．　DUT-type-name：セット[0-9a-zA-Z]からの一つ以上の文字の列であり、最初の文字はセット[0-9]からのものではない。これは、例えばペンティアム３のようなDUTのタイプを表す。

【０１１７】

３．　pin-description-file-name：ファイルの単なる名前であり、そのディレクトリ名は含まないが、全ての拡張子を含んでいる。ファイル名は、ホストオペレーティングシステムによって認識されるシンタックスを有しており、引用符で挟まれていれば空白や他の文字も許される。

【０１１８】

４．　switch-matrix-input-port-number：十進法の負でない整数であり、サイトコン

トローラに接続された入力ポートのポート番号を表す。

【０１１９】

５．　dut-id：DUTのインスタンスを識別する十進法の負でない整数である。

【０１２０】

６．　resource-name：セット[0-9a-zA-Z]からの一つ以上の文字の列であり、最初の文字は数字であってはならない。リソースファイルにおいて定義されるリソース名を表す。

【０１２１】

７．　resource-item-name：セット[0-9a-zA-Z]からの一つ以上の文字の列であり、最初の文字は数字であってはならない。ピン、あるいはピングループといったリソースユニットの名前を表す。

【０１２２】

８．　resource-item-index：リソースアイテムのグループの特定のメンバを表す十進法の負でない整数である。リソースアイテムインデックス範囲に関しては、これはリソースアイテムグループの連続的な列の下方の境界あるいは上方の境界を表す。

【０１２３】

９．　resource-unit-index：リソースユニット（チャネル）のグループの特定のメンバを表す十進法の負でない整数である。リソースユニットインデックス範囲に関しては、これはリソースユニットグループの連続的な列の下方の境界あるいは上方の境界を表す。

Ａ６．　ピン

【０１２４】

論理ピン名対物理チャネルのマッピング（ソケットによって提供される）に加えて、テスタリソースを指定するためにいくつかの属性を用いることができることに留意されたい。例えば、チャネルの特定のハードウェアコンフィギュレーションを定義するためにオプションを用いてもよい。これはテスト特有、ベンダ特有、および／あるいはテストシステムに特有のものであってもよい。これらは、ピンモードオプションを用いて記述され、ピンモードオプションファイルを介して利用可能とされる。

【０１２５】

ピンモードオプションの定義は、テスタチャネルについて特定のオプションまたはモードのコンフィギュレーションをサポートするであろう。これは、例えば、チャネル多重化を選択し、構成するために用いられることもできる。ピンモードオプションのみをテストプラン初期化フローの一部として用いることが好ましい。なぜなら、重要なチャネルコンフィギュレーションを必要とし得るからである。ピンモードオプションシンタックスは、ベンダ定義のオプションをサポートする。一例を以下に示す。

```
PinModeOptions
{
    clock        IN        double;
    a0            OUT        single;
    ...
};
```

テスト環境コンフィギュレーション

【０１２６】

先に指摘したように、リソース定義ファイル（Resource.rsc）、システムコンフィギュレーションファイル（Sys.cfg）およびモジュールコンフィギュレーションファイル（Modules.cfg）は好ましくは、「既知の」位置で利用可能とされる。この「既知の」位置は、システム環境変数Tester_ACTIVE_CONFIGSの値によって指定されるディレクトリである。

例えば、もしTester_ACTIVE_CONFIGの値がディレクトリF:?Tester_SYS?configs?であれば、システムは以下のファイルが存在するであろうと予測することができる。

F:\Tester_SYS\configs\Resources.rsc
F:\Tester_SYS\configs\Sys.cfg
F:\Tester_SYS\configs\Modules.cfg

【０１２７】

インストールの間、ホストコンピュータ上にあるインストール・コンフィギュレーション管理システム（ICM）は好ましくは、Tester_ACTIVE_CONFIGSの値を設定する。ICMは上記ファイルの一つの新しいバージョンを生成するたびに、Tester_ACTIVE_CONFIGSが指す位置に新しいバージョンを置く。上記３つのファイルに加えて、シミュレーションコンフィギュレーションファイルといった他のシステムコンフィギュレーションファイルもまたTester_ACTIVE_CONFIGSが指す位置に置かれる。

Ｂ．　テストプログラム開発のためのルール

【０１２８】

テスタシステムの２つの主なエンドユーザ指向のコンポーネントのうちの一つは、テスト環境である。もう一つのコンポーネントは、テスタがエンドユーザ（すなわちテストエンジニアおよびテストクラスディベロッパ）に供するプログラミング機構である。

【０１２９】

プログラミング環境の主要なコンポーネントはテストプランである。テストプランは、テストクラス（これらはTestで表されるテストインタフェースの異なるインプリメンテーションである）を用いており、これらは特定のタイプのテストのためのテストデータおよびコードの分離を実現する。

【０１３０】

プランは、C++テストプログラムとして直接書かれてもよいし、テストプログラムジェネレータ（トランスレータ４０２）によってC++コードのようなオブジェクト指向のコードを作成するように処理されるテストプラン記述ファイルで記述されてもよい。作製されたC++コードは、実行可能なテストプログラムにコンパイル可能である。レベル、タイミング等のようなテストクラスインスタンスに関連することが要求されるデータは、テストプラン記述ファイルにおいてユーザによって指定される。

【０１３１】

テストプログラムは、装置上でテストを動作させるための詳細を指定するユーザ作成のファイルのセットを含んでいる。発明の一実施形態は、ユーザがこれらのファイルをC++コンストラクトを用いて書くことを許容するルールのセットを含んでいる。

【０１３２】

発明のこの実施形態による要件の一つは、オープンアーキテクチャテストシステムのモジュール方式に従うことである。モジュール式の開発は、ユーザがテストの異なる局面を扱う個々のコンポーネントを記載することを許容し、そして、完全なテストプログラムを生み出すためにこれらのコンポーネントがさまざまなやり方で混合され、適合されることを許容する。本発明の好ましい実施形態によるテストプログラムは、以下のファイルのセットを備えている。

files *.usrv for user variables and constants;
files *.spec for specification sets;
files *.lvl for levels;
files *.tim for timings;
files *.tcg for test condition groups;
files *.bdefs for bin definitions;
files *.ph for a pre-header, files for custom functions and test classes.
files *.ctyp for custom types;

files *.cvar for custom variables; and
files *.tpl for test plans.
【０１３３】
上記ファイルの拡張子は、ファイルのカテゴリー化をスムーズにする、推奨される取り決めである。単一のテストプログラムは、好ましくは、単一のテストプランファイルとそれがインポートするファイルとを備えている。「インポート」とは、インポータ（インポートを指定するファイル）によって直接参照されるか、インポータによって直接参照される何か他のファイルによってインポートされるデータを有する他のファイルのことをいう。テストプランファイルは、グローバル、フローならびに、その中の他のこのようなオブジェクトを定義することができ、あるいは他のファイルからのこの情報をインポートすることができる。これらのルールは、上記コンポーネントのいずれかが、自身の個々のファイルにあるか、あるいはテストプランファイルに直接インラインされるかを可能にする。なお、テストプランは、C言語のmain()関数と概念的に似ている。

テストプログラムの特徴

ユーザ変数および定数
仕様セット
レベル
タイミング
テスト条件
ビン定義
プリヘッダ
カスタムタイプ
カスタム変数
テストプラン
【０１３４】
テストプログラム識別子は、好ましくは、大文字または小文字のアルファベットの文字で始まり、その後にアルファベット、数字あるいはアンダースコア（_）といった文字をいくつでも有することができる。以下に示される記述において提供されるものは、いくつかのキーワードを有している。これらのキーワードは、例えばVersionのような太字を用いてこの文書ではコードで視覚的に識別される。キーワードは取っておかれ、好ましくは識別子としては用いられない。{, }, (,),のような特別な記号がいくつかあり、以下に示す他のものもある。

テストオブジェクトの詳細
【０１３５】
テスト記述ファイルのインポートは、インポートするファイルが、インポートされるファイルによって利用可能とされているオブジェクトの名前を参照することを可能にする。これにより、インポートするファイルは、インポートされるファイルによって名付けられたオブジェクトを参照することができる。ピン記述ファイルxxx.pinをインポートするソケットファイルaaa.socを考えることにする。xxx.pinをインポートする別のbbb.socファイルがあるかもしれない。しかし、これらのインポートはいずれも、xxx.pinで述べられるオブジェクトを発生させない。これらは単に、既に存在していると仮定されるオブジェクトを参照するにすぎない。
【０１３６】
ここで疑問が生じる。このようなオブジェクトはいつ発生するのか？テストプランファイルが根本的に異なるのはここである。Cとの相似において、それはmain()ルーチンを中に有しているファイルである。テストプランファイルにおける「インポート」命令文は、これらのオブジェクトを詳述する。すなわち、これらのオブジェクトを出現させる。以下

に示すテストプランmickey.tplは、xxx.pinおよびaaa.socにおけるオブジェクトを詳述させる。

```
# File for Mickey's TestPlan
Version 3.4.5;

#
# These import statements will actually cause the
# objects to come into existence:
#
Import xxx.pin;    # Elaborates pin and pin-group objects
Import aaa.soc;    # Elaborates site socket map objects
# Other imports as necessary
...
Flow Flow1
{
    ...
}
```

【０１３７】

テストプランにおけるxxx.pinのインポートは、xxx.pinにおいて宣言されている全てのピンおよびピングループオブジェクトを詳述させる。これを次のように説明する。「ファイルxxx.pinは詳述される。」テストプランは詳述される必要がある全てのファイルを直接インポートする必要はない。もし以下の2つの命令文が真であれば、ファイルxはファイルyによってインポートされる。

１．yはxの名前を挙げるインポート命令文を有している、あるいは

【０１３８】

２、xはzによってインポートされ、yはzの名前を挙げるインポート命令文を有している。

【０１３９】

テストプログラムがコンパイルされるときには、それは、テストプランによってインポートされるファイルにおける全てのオブジェクトを詳述する。テストプランによってインポートされるファイルのセットは、ファイルが詳述される順番を作り出すように、位相的にソートされる。テストプランによってインポートされるファイルは、テストプランのインポートクロージャ（closure）と呼ばれる。もしあるテストプランのインポートクロージャを位相的にソートすることができなければ、インポートサイクルがあるはずである。このような状況はエラーであり、コンパイラによって拒絶される。

ユーザ変数および定数

【０１４０】

グローバルな変数および定数が、ユーザ変数および定数を用いて定義される。定数は、その値がコンパイル時間に結び付けられているようなオブジェクトであり、変化することはない。例えば、最大の整数値は一定である。一方、変数に結び付けられている式は、APIを介してランタイムで変化し得る。

整数
符号なし整数
倍（Double）
文字列（String）
ボルト（V）での電圧
1秒あたりのボルト（VPS）での電圧スルー

アンペア（A）での電流
ワット（W）での電力
秒（S）での時間
メートル（M）での長さ
ヘルツ（H）での周波数
オーム（Ohm）での抵抗値
ファラド（F）での容量値

【０１４１】

整数、符号なし整数、倍および文字列のタイプはベーシックタイプと呼ばれる。ベーシックタイプは測定単位をもたない。ベーシックタイプではないエレメンタリタイプは倍であり、関連する測定単位と尺度（スケール）とを有する。スケーリングの記号は、一般的な工学のスケーリングの記号である。

【０１４２】

10-12についてはｐ（ピコ）で、pF（ピコファラド）のようである。

【０１４３】

10-9についてはｎ（ナノ）で、ns（ナノ秒）のようである。

【０１４４】

10-6についてはμ（マイクロ）で、μS（マイクロ秒）のようである。

【０１４５】

10-3についてはｍ（ミリ）で、mV（ミリボルト）のようである。

【０１４６】

10+3についてはｋ（キロ）で、kOhm（キロオーム）のようである。

【０１４７】

10+6についてはM（メガ）で、MHz（メガヘルツ）のようである。

【０１４８】

10+9についてはG（ギガ）で、GHz（ギガヘルツ）のようである。

【０１４９】

ユーザ変数および定数を有する別個のファイルは、拡張子「.usrv」を有する。いくつかのグローバルな定数を有するファイルの一例を以下に示す。いくつかの変数を有するファイルは、後で示す。

```
# ------------------------------------------------------
# File limits.usrv
# ------------------------------------------------------

Version 1.0.0;

#
# This UserVars collection declaration declares a set of
# globally available variables and constants.
#
UserVars
{
    # Some constant Integer globals used in various places.
    Const Integer        MaxInteger = 2147483647;
    Const Integer        MinInteger = -2147483648;

    # Smallest value such that 1.0 + Epsilon  != 1.0
    Const Double    Epsilon = 2.2204460492503131e-016;
```

```
    # Some important constants related to Double
    Const Double MaxDouble = 1.7976931348623158e+308;
    Const Double MinDouble = - MaxDouble;
    Const Double ZeroPlus = 2.2250738585072014e-308;
    Const Double ZeroMinus = - ZeroPlus;

}
```

【０１５０】

上で宣言されたユーザ変数のセットは、「＝」の左の変数の定義と考えられる。その結果、変数または定数の定義が一度出現することが好ましく、それは初期化されなければならない。

【０１５１】

先に述べたように、定数は一度定義されたら変わるべきではない。定数に結び付けられている式は、先に定義された定数とリテラル（literal）な値とを含み得る。一方、変数はAPIを介して変えることができる。変数に結び付けられた式は、先に定義された変数、定数およびリテラルな値を含み得る。

【０１５２】

各変数は、ランタイムで維持される式オブジェクトに結び付けられている。これは、ランタイムでの変数に関連している式を変更し、全ての変数を再評価する可能性を提供する。式オブジェクトは、変数または定数の定義の右辺の分解された形である。ある実施形態において、ランタイムでの定数を変更することについては何の機構も提供されない。定数の値は、好ましくは、コンパイル時間において固定されている。

【０１５３】

グローバルを有するこのようなファイルはいくつでも、テストプランのインポートクロージャ中に存在することができる。上記グローバルのファイルが数字の制限のセットであるとき、工学測定単位と、いくつかのランダムなユーザ変数を用いた工学のグローバルのセットがこれである。

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

Version 0.1;

#
# This declares a UserVars collection of some engineering
# globals.
#
UserVars MyVars
{
    # Engineering quantities.
    Const Voltage VInLow = 0.0;           # 0 Volts
    Const Voltage VInHigh = 5.0;           # 5 Volts
    Const Voltage VOutLow = 400.0 mV;       # 400 milliVolts
    Const Voltage VOutHigh = 5.1;           # 5.1 Volts
    Const Time DeltaT = 2.0E-9;           # 2 nanoseconds
    Const Time ClkTick = 1.0ns;           # 1 nanosecond
    Const Resistance R10 = 10.0 kOhms;    # 10 kilo Ohms

    # Some variables are declared below.
```

```
    Current ILow = 1.0 mA;                    # 1 milliAmp
    Current IHigh = 2.0 mA;                   # 2 milliAmp
    Power PLow = ILow * VInLow;             # Low power value
    Power PHigh = IHigh * VInHigh;        # High power value

    #
    # An array of low values for all A bus pins.
    # The vil for A0 will be in ABusVil[0], for A1
    # in ABusVil[1], and so on.
    #
    Voltage ABusVil[8] = {1.0, 1.2, Others = 1.5};
}
```

【０１５４】

好ましくはコンパイラは、単位とタイプとが合っているかをチェックする。電圧×電流が電力となるので、上記PLowとPHighについての式をコンパイルする。しかし、以下のような命令文は典型的にはコンパイルしない。

```
#
# Does not compile because a Current and a Voltage cannot be added
# to yield a Power.
#
Power Pxxx = IHigh + VInHigh;
```

【０１５５】

コンパイラは以下の特定の自動型変換を許容する。

```
Power Pxxx = 2;    # Set the power to 2.0 watts
Integer Y = 3.6;    # Y gets assigned 3
Power Pyyy = Y;    # Pyyy gets assigned 3.0 watts
Double Z = Pyyy;    # Pyyy gets converted to a unitless Double
```

【０１５６】

ダブル、符号なし整数および整数への明示的な変換もまた許容される。

```
Power Pxxx = 3.5;

# Explicit type conversion is allowed, but not required.
# X becomes 3.5
Double X = Double(Pxxx);        # X becomes 3.5
Integer Y = Integer(Pxxx);        # Y becomes 3
```

【０１５７】

無関係のタイプ間の変換も、中間のベーシックタイプへと変換することによって、可能である。

```
Power Pxxx = 3.5;

# Explicit type conversion is required.
Length L = Double(Pxxx);    # L becomes 3.5 meters
Voltage V = Integer(Pxxx);    # V becomes 3.0 Volts.
```

【０１５８】

テストプランオブジェクトは、名前と関連する式、値およびタイプを含む集合体であるUserVar）クラスを提供する。ユーザ変数は、デフォルトユーザ変数コレクション（Defau

lt User Variables Collection）、あるいは名前付きユーザ変数コレクション（Named User Variables Collection）に入ることができる。上記例でのユーザ変数（UserVars）の宣言は、特定の名前は有してはいないが、デフォルトのコレクションに入る。しかしながら、以下のように、コレクションを明示的に名付けることもできる。

```
# Declare X and Y in the MyVars UserVars collection.
UserVars MyVars
{
    Integer X = 2.0;
    #
    # Refers to the above X, and to the globally
    # available MaxInteger from the default
    # UserVars collection.
    #
    Integer Y = MaxInteger - X;
}
# Declare X, Y1 and Y2 in the YourVars UserVars collection.
UserVars YourVars
{
    Integer X = 3.0;
    # Refers to the X from MyVars.
    Integer Y1 = MaxInteger - MyVars.X;
    # Refers to the X declared above.
    Integer Y2 = MaxInteger - X;
}
# More variables being added to the MyVars collection
UserVars MyVars
{
    #
    # Refers to X and Y from the earlier declaration
    # of MyVars.
    #
    Integer Z = X + Y;
}
```

【０１５９】

UserVarsのコレクション内の名前の解決は以下のように進行する。

【０１６０】

もし名前が限定されていれば――すなわち、名前がドットで区切られた２つのセグメントを有していれば――変数は、ドットの前のセグメントが示される名前つきのユーザ変数コレクションからもたらされる。つまり、上のMyVars.Xは、MyVarsコレクションにおけるXを指している。「_UserVars」という名前は、デフォルトユーザ変数コレクションを明示的に示すために用いられ得る。

【０１６１】

もし名前が限定されておらず、現コレクション内に同じ名前の定数あるいは変数があれば、その名前はその定数あるいは変数になる。

【０１６２】

そうでなければ、その名前は、デフォルトユーザ変数コレクション内の定数あるいは変数となる。

【０１６３】

ユーザ変数コレクションにおける定義のブロックの評価は、最初の定義から最後のもの

まで順に起こるものと考えることができる。これは、各変数が用いられる間に定義されていることを必要とするであろう。

【０１６４】

さらに、ユーザ変数コレクションのための定義のブロックがいくつかあり、これらのそれぞれがいくつかの変数を定義していることもある。これらの定義ブロックの全ては、テストプランにおける宣言順に評価されるものと考えることができ、各ブロックの変数もまた宣言順でチェックされる。

【０１６５】

最後に、いくつかのユーザ変数コレクションがあり、これらのそれぞれがいくつかの定義ブロックに対して変数を定義していることがある。ここでも変数の全ては宣言順に初期化されるものと考えることができる。したがって、上記例では、評価順は、MyVars.X、MyVars.Y、YourVars.X、YourVars.Y1、YourVars.Y2、Myvars.Zである。

【０１６６】

ユーザ変数コレクションが他のコレクションからの変数を用いるときには、それは、好ましくは変数の生のデータだけを用いる。コレクションの間には、何の依存度（dependency）に関する情報も維持されていない。したがって、再評価に基づく依存度（dependency）は、単一のコレクションに限定することができる。

【０１６７】

各ユーザ変数のコレクションは、C++UserVarsクラスのインスタンスを参照する。C++UserVarsクラスのデフォルトオブジェクトは、「_UserVars」と名付けられる。名付けられていないUserVars宣言における変数は、デフォルトのユーザ変数コレクションからのものであり、このデフォルトオブジェクトに追加される。名前付きユーザ変数コレクションにおける変数は、その名前を有するC++UserVarsクラスのオブジェクトに追加される。上記例では、「Myvars」C++オブジェクトは、結局は変数X、YおよびZを有する。

ユーザ変数のためのC++

【０１６８】

ユーザ変数は、名前の文字列、const/varのブール代数、列挙された値としてのタイプ、および拡張子ツリーとしての式（expression）を有するｎタプルのコレクションとしてインプリメントされる。名前の式は、コール（call）によって設定することができる。

```
enum ElemenaryType {UnsignedIntegerT, IntegerT,
                    DoubleT, VoltageT, ...};
Status setExpression(const String& name,
                     const bool isConst,
                     const elementaryType,
                     const Expression& expression);
```

【０１６９】

タイプの式は、割り当ての右辺に対応するテキストの解析された形であるタイプである。UserVarsのグローバルに利用可能なインスタンスがある。例えば、limits.usrvにおけるユーザ変数のセット（ページ参照）は、以下に示すコールのセットによってインプリメントされる。

```
_UserVars.setExpression("MaxInteger", true, IntegerT,
                        Expression(2147483647));
_UserVars.setExpression("MinInteger", true, IntegerT,
                        Expression(-2147483648));
_UserVars.setExpression("Epsilon", true, DoubleT,
                        Expression(2.2204460492503131e-016));
_UserVars.setExpression("MaxDouble", true, DoubleT,
```

```
                        Expression(1.7976931348623158e+308));
_UserVars.setExpression("MinDouble", true, DoubleT,
                        Expression("- MaxDouble"));
_UserVars.setExpression("ZeroPlus", true, DoubleT,
                        Expression(2.2250738585072014e-308));
_UserVars.setExpression("ZeroMinus", true, DoubleT,
                        Expression("- ZeroPlus"));
```

【０１７０】

myvars.usrvにおいて宣言される変数について実行されるであろうC++命令文は以下の通りである。

```
myVars.setExpression("VInLow", true, VoltageT,
                     Expression(0.0));
myVars.setExpression("VInHigh", true, VoltageT,
                     Expression(5.0));
myVars.setExpression("DeltaT", true, TimeT,
                     Expression(2.0E-9));
myVars.setExpression("ClkTick", true, TimeT,
                     Expression(1.0E-9));
myVars.setExpression("R10", true, ResistanceT,
                     Expression(10.0E+3));
myVars.setExpression("ILow", false, CurrentT,
                     Expression(1.0E-3));
myVars.setExpression("IHigh", false, CurrentT,
                     Expression(2.0E-3));
myVars.setExpression("PLow", false, PowerT,
                      Expression("ILow * VInLow"));
myVars.setExpression("PHigh", false, PowerT,
                      Expression("IHigh * VInHigh"));
myVars.setExpression("ABusVil[0]", false, VoltageT,
                     Expression(1.0));
myVars.setExpression("ABusVil[1]", false, VoltageT,
                     Expression(1.2));
myVars.setExpression("ABusVil[2]", false, VoltageT,
                     Expression(1.5));
myVars.setExpression("ABusVil[3]", false, VoltageT,
                     Expression(1.5));
myVars.setExpression("ABusVil[4]", false, VoltageT,
                     Expression(1.5));
myVars.setExpression("ABusVil[5]", false, VoltageT,
                     Expression(1.5));
myVars.setExpression("ABusVil[6]", false, VoltageT,
                     Expression(1.5));
myVars.setExpression("ABusVil[7]", false, VoltageT,
                     Expression(1.5));
```

【０１７１】

上記コードにおいて、式クラスは、好ましくは、式の解析された形をあらわすコンストラクタ（constructor）を有する。式は、リテラルな文字列をとって分解するコンストラクタと、リテラルな文字列としてだけ用いるためにリテラルな文字列を取り込む別のコンストラクタとを含むいくつかのコンストラクタを有している。これらは、読みやすさのた

めに上で指定していない追加のパラメータによって区別される。

【０１７２】

デフォルトのユーザ変数コレクションにおけるユーザ変数は、UserVarsクラスの_UuserVarsオブジェクトによって管理される。名前付きユーザ変数コレクションXxxにおけるユーザ変数は、Xxxの名前がついたUserVarsオブジェクトによって管理される。

UserVarsのためのランタイムAPI

【０１７３】

名前および式を含むC++UserVarsクラスは、これらの値をランタイムで評価し、改変するために、アプリケーションプログラミングインタフェース（API）をエクスポートする。また、UserVarsに関連している式の改変は、いつUserVarsが再評価されるか、および評価の影響はどのようなものであるかという問題をも扱う。

【０１７４】

変化の結果としてのUserVarsの再評価がいつトリガされるべきであるかという問題を最初に考える。もしそれが式に対して変更がなされたときに直ちにトリガされれば、ユーザは、再評価のトリガよりも前に一連の関連する変更を行うことができないであろう。したがって、再評価は、ユーザによる明示的なコールによってトリガされる。

【０１７５】

再評価の影響を次に考える。好ましい実施形態によると利用可能である再評価は三種類ある。

【０１７６】

UserVarsコレクションの再評価は、単一のUserVarsコレクションに限定された再評価である。この処理のセマンティクス（semantics）は、このコレクションの全ての変数をもう一度再評価することである。

【０１７７】

UserVarsターゲットの再評価は、単一の名前に結び付けられた式に対する変化に限定された再評価である。これは、ユーザが単一の名前の式を変更することを可能にし、この特定の変更のみを考慮に入れた、コレクションの再評価を起こす。

【０１７８】

UserVarsグローバル再評価は、全てのUserVarsコレクションの再評価である。これは基本的には、宣言の順番でUserVarsコレクションの全ての再評価のトリガとなり、極めてコストがかかる。

【０１７９】

上記再評価の全ては、UserVarsを再評価した後に、レベル、タイミング等の従属オブジェクトを再評価する。従属オブジェクトは、それが再評価を必要とすることを表している「汚い（dirty）」ビットを有している。UserVarsコレクションがプログラム的に変更されるといつでも、それは、全ての従属オブジェクト上で汚いビットを設定する。これが、従属オブジェクトの再評価のトリガとなる。

【０１８０】

まとめると、名前つきのUserVarsコレクションは、再評価の影響の問題を含むことを助ける。再評価は通常は単一のコレクションに限定される。UserVarsを用いる単純な方法は、デフォルトUserVarsコレクションを用いることだけであり得る。その方法、変更することの波状の効果が全てのUserVarsに起こる。この波状の効果は、いくつかの名前付きUserVarsコレクションを有することによって制限される。

【０１８１】

複数のコレクションは、互いから変数を参照することができるが、変数に結びつけられた値は、使用の時間に結び付けられている。UserVarsコレクション間には従属関係（dependency）は何も維持されていない。

各エレメンタリタイプXxx（符号なし整数、電流、電圧等）について、値を得る方法は、

```
    Status getXxxValue(Const String& name, Xxx& value) const;
```

【０１８２】
である。
【０１８３】
値を直接設定する方法はなく、それは、式を設定するようにコールを通じて行われ、reevaluateCollection()へのコールが後に続くことに留意されたい。
【０１８４】
式を取得、設定する方法である。setExpression()のコールは、これまでに定義されていなかった新しい変数を定義するためにも用いられる。

```
    enum elementaryType
    {
        UnsignedIntegerT, IntegerT, DoubleT, VoltageT, ...
    };
    Status getExpression(const String& name,
                         Expression& expression) const;
    Status setExpression(const String& name,
                         const bool isConst,
                         const elementaryType,
                         const Expression& expression);
```

【０１８５】
setExpression()コールは、式が循環的な従属関係をもたらすのであれば、失敗する。例えば、以下の2つのコールが行われると、二番目のコールは、循環的な従属関係という不具合で失敗する。

```
    setExpression("X", true, IntegerT, Expression("Y+1"));
    setExpression("Y", true, IntegerT, Expression("X+1"))
```

【０１８６】
これは、名前に結び付けられた値は、等式（equation）であって、割当（assignment）ではないからである。変数の値が変わると、直接的、間接的に従属する名前の全てを再評価するように方法が提供される。上記ペアのような等式は、許容されない循環的な従属関係という結果に終わる。
【０１８７】
このAPIは、典型的には、頼まれもしない再評価はサポートしないことを留意されたい。setExpression()へのコールは、自動的に変数、およびそれに依存する全てのほかの変数の再評価を引き起こすのではない。全ての変数に結び付けられた値は、reevaluateCollection()へのコール（以下）が起こるまでは、変化しないままである。
特定の名前が定数であるかどうかを判断する方法は、

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

であり、タイプを取得する方法は、

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

UserVarsコレクションの再評価方法は

```
static Status reevaluateAllCollections();
```

【０１８８】
である。
【０１８９】
クラスは、全ての変数に関連している等式、ならびに従属関係を維持する。この方法が呼び出されると、変数の全ては再評価されるであろう。
UserVarsターゲットの再評価方法は、

```
Status reevaluateTargeted(const String& var);
```

【０１９０】
クラスは、全ての変数に関連する式、ならびにそれらの従属関係を維持する。この方法が呼び出されると、名前つきの変数、およびそれに依存するものの全てが再評価される。
UserVarsグローバル再評価方法は、

```
static Status reevaluateAllCollections();
```

【０１９１】
クラスは、全ての変数に関連する式、ならびにそれらの従属関係を維持する。この方法が呼び出されると、reevaluateCollection()が全てのUserVarsコレクション上に指定されていない順序で呼び出される。
特定の名前が定義されているかどうかを判断する方法は、

```
Status getIsDefined(const String& name, bool& isDefined) const;
```

【０１９２】
である。
現在定義されているユーザ変数の全てを判断する方法は、

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

【０１９３】
である。
現在定義されている変数を削除する方法は、

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

【０１９４】
である。
任意の変数または定数に依存する変数および定数のリストを取得する方法は、

```
Status getDependents(const String& name, StringList& dependents);
```

【０１９５】
である。

仕様セット
【０１９６】
仕様（Specification）セットは、セレクタ（Selector）に基づく値をとる変数のコレクションを供給するために用いられる。例えば、ミニー、ミッキー、グーフィーおよびデイジーというセレクタを用いる以下の仕様セットを考える。

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

Version 1.0;
```

```
Import Limits.usrv;

SpecificationSet Aaa(Minnie, Mickey, Goofy, Daisy)
{
    Double xxx = 1.0, 2.0, 3.0, 4.0;
    Integer yyy = 10, 20, 30, 40;
    Integer zzz = MaxInteger - xxx,
                  MaxInteger - xxx - 1,
                  MaxInteger - xxx - 2,
                  MaxInteger - xxx;

    # The following declaration associates a single
    # value, which will be chosen regardless of the
    # selector.  It is equivalent to:
    #    Integer www = yyy + zzz, yyy + zzz, yyy + zzz, yyy + zzz
    Integer www = yyy + zzz;
}
```

【０１９７】
セレクタグーフィーを有する上記仕様セットは、以下の関連付けを行う。

```
xxx = 3.0;
yyy = 30;
zzz = MaxInteger - xxx - 2;
www = yyy + zzz;
```

【０１９８】
仕様セット上でのセレクタの設定の処理は、後で、テストを説明するときに述べる。

【０１９９】
構文的には、仕様セットは、変数の定義（上記例ではxxx、yyy、zzzおよびwww）のリストと同様に、セレクタ（上記例ではミニー、ミッキー、グーフィーおよびデイジー）のリストである。変数の定義は、セレクタのリストであるか、単一の式を備えているかのどちらかである式のリストを含む。

【０２００】
概念的には、仕様セットは、式のマトリクスであると考えることができ、その列がセレクタ、その行が変数、エントリが式である。特定のセレクタ（列）は、各変数（行）を特定の式（エントリ）に結びつける。もしリストが単一の式を有していれば、それはセレクタがある回数だけ反復される式を有する行を表している。

【０２０１】
仕様セットは、2つの別個の状況において登場する。それらは、「.spec」ファイルにおいて別々に宣言され得、いずれかの場合においてそれらは上記のように登場する。これらは名前つきの使用セット（named specification set）である。そうでなければ、ローカルな仕様セットをテスト条件グループ内で宣言することができる。このような宣言において、仕様セットは、名前を与えられない。それは、ローカルな仕様セットであり、それを取り巻くテスト条件グループにとってのみ重要性を有する。

【０２０２】
名前つき仕様セットは、名前つきユーザ変数コレクションの模倣とされてもよい。上記仕様セットは、Aaaという名前のUserVarsコレクションとして模倣され得、これはxxx[ミニー]、xxx[ミッキー]、xxx[グーフィー]、xxx[デイジー]、yyy[ミニー]等と続く式を有する。特定のセレクタ（ミッキーとしよう）がテストの状況において選ばれると、xxx、yyyおよびzzzの値が変数名および仕様セット名から得られる。

【０２０３】
テスト状況グループは、多くても一つの仕様セットを有することができ、これはローカルな仕様セットか、名前つき仕様セットへの参照であるかである。ローカルな仕様セットは、テスト条件グループの状況においてのみ現れ、明示的に指定された名前はもたない。このような仕様セットは、取り囲むテスト条件グループの名前によって定義される暗黙の名前を有する。いくつかの仕様セットおよびいくつかのUserVarsコレクションが目に見えるような点でテスト状況グループにおいて名前を解決するために、以下のルールが適用される。
【０２０４】
１．名前が限定されていれば、それは名前つきユーザ変数コレクションで解決されなければならない。
【０２０５】
２．名前が限定されていなければ、その名前は、テスト条件グループにおいて宣言されていればローカルな仕様セットであり、テスト条件グループで参照されていれば名前つき仕様セットである。
【０２０６】
３．名前が先のルールで解決されなければ、それはデフォルトユーザ変数コレクションに決定される。
【０２０７】
これらのルールを示すために、テスト条件グループ（後述する）を用いている以下の例を考える。

```
Version 1.2.3;
Import limits.usrv;    # Picks up the limits UserVars file above.
Import aaa.spec;     # Picks up the Specification Set AAA above.

TestConditionGroup TCG1
{
    SpecificationSet(Min, Max, Typ)
    {
        vcc = 4.9, 5.1, 5.0;
    }

    # Rule 1: Resolution in a named user variables collection.
    # A reference to MyVars.VInLow refers to VInLow from MyVars.

    # Rule 2: Resolution in a local specification set.
    # A reference to "vcc" here will resolve in the context
    # of the local specification set above.

    # Rule 3: Resolution in default user variables collection.
    # A reference to "MaxInteger" here will resolve to limits.usrv.

    # Error: Resolution of xxx
    # A reference to xxx does not resolve because it is neither in
    # the local specification set, nor in limits.usrv.

    # Error: Resolution of Aaa.xxx
    # Looks for a named UserVars collection named Aaa.  The named
    # specification set does not qualify.
```

```
}

TestConditionGroup TCG2
{
    SpecificationSet Aaa;    # References the imported specification set

    # Rule 1: Resolution in a named user variables collection.
    # A reference to MyVars.VInLow refers to VInLow from MyVars.

    # Rule 2: Resolution in a named specification set.
    # A reference to "xxx" here will resolve in the context
    # of the local specification set Aaa above.

    # Rule 3: Resolution in default user variables collection.
    # A reference to "MaxInteger" here will resolve to limits.usrv.

    # Error: Resolution of vcc
    # A reference to vcc does not resolve because it is neither in
    # the named specification set Aaa, nor in limits.usrv.

    # Error: Resolution of Aaa.xxx
    # Looks for a named UserVars collection named Aaa.  The named
    # specification set does not qualify.
}
```

【０２０８】

仕様セットにおける名前の解決（上のルール）は、そのセットのセレクタが名前の解決が必要とされるときに有効とされていることを要する。これは、テスト条件グループは、セレクタを指定することによってテストにおいて参照されるという事実によって強制される。

仕様セットのためのC++

【０２０９】

上記ルールを用いて、仕様セットをC++SpecificationSetクラスによってインプリメントすることができる。SpecificationSetクラスは、セレクタ用の余分な文字列（String）パラメータをのぞいては、UserVarsクラスと本質的に同じAPIを有している。したがって、このAPIは詳細には説明しない。

【０２１０】

全ての名前つきの仕様セットは、好ましくは、その名前のC++オブジェクトに関連付けられている。テスト条件グループの状況では、ローカルな仕様セットは、そのテスト条件グループに固有の名前を有する。それが定義されているテスト条件グループの状況外でローカルな仕様セットの変数を参照することは不正である。

レベル

【０２１１】

ピンおよびピングループのパラメータを指定するためにレベル（Level）を用いる。それは以下の形態の宣言のコレクションである。

```
<pin-or-pin-group-name>
{
```

```
    <pin-param-1> = xxx;
    <pin-param-2> = yyy;
    ...
}
```

【０２１２】

このような宣言は、名前つきのピンまたはピングループのさまざまなパラメータの設定を指定する。例えば、このような命令文は、以下の例に示すように、InputPinsグループにおける全てのピンのためのVIL値を設定するために用いられ得る。

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

Version 1.0;

Import pentium3resources.rsc;
Import pentium3pins.pin;

Levels Pentium3Levels
{
    #
    # Specifies pin-parameters for various pins and
    # pin groups using globals and values from
    # the specification set.
    #
    # The order of specification is significant.
    # Pin parameters will be set in order from
    # first to last in this Levels section, and
    # from first to last for each pin or pin-group
    # subsection.
    #
    # From the imported pin description file pentium3pins.pin,
    # the InPins group is in the "dpin" resource.  From the
    # imported resource definition file pentium3resources.rsc,
    # the "dps" resource has parameters named VIL and VIH.
    #
    InPins { VIL = v_il; VIH = v_ih + 1.0; }


    # The following statement requires a delay of 10 uS after
    # the call to set the InPins levels.  Actual delay will be
    # a small system defined range around 10.0E-6:
    #       10.0E-6 - delta <= actual <= 10.0E-6 + delta
    Delay 10.0E-6;

    #
    # For the OutPins, the levels for the parameters
    # VOL and VOH are specified.
    #
    OutPins { VOL = v_ol / 2.0; VOH = v_oh; }
```

```
    # The clock pin will have special values.
    Clock { VOL = 0.0; VOH = v_ih / 2.0; }

    # A Delay of 10 uS after the call to set Clock levels.
    # This is a minimum delay, that is guaranteed to be for
    # at least 10.0 uS, though it may be a little more:
    #       10.0E-6 <= actual <= 10.0E-6 + delta
    MinDelay 10.0 uS;

    #
    # The PowerPins group is in the "dps" resource.  Pins of this
    # pin group have special parameters:
    #    PRE_WAIT specifies the time to wait after voltage
    #        reached its final value to start pattern
    #        generation.  Actual wait time will be a small
    #        system defined range around PRE_WAIT (see)
    #    POST_WAIT specifies the time to wait after pattern
    #        generation ends to shut down the power.  Actual
    #        wait time will be a small system defined range
    #        around PRE_WAIT (see).
    #
    PowerPins
    {
        PRE_WAIT = 10.0 ms;
        POST_WAIT = 10.0 ms;

        # VCC reaches its final value of 2.0 V from its
        # present value in a ramp with a Voltage Slew Rate
        # of ±.01 Volts per Second.
        VCC = Slew(0.01, 2.0 V);
    }
}


Levels Pentium4Levels
{
    # ...
}
```

【０２１３】

上でわかるように、各レベルブロックは、好ましくは、多くのレベルアイテムから構成され、それらのそれぞれは、ピンまたはピングループのためのパラメータを指定する。各レベルアイテムは、リソースパラメータの数を指定することができる。これらのレベルの値を設定するためのランタイムセマンティクス（semantics）は、以下の通りである。

【０２１４】

レベルブロックのレベルアイテムは、宣言順に処理される。2つ以上のレベルアイテムで起こるピンはいずれも、複数回処理される。単一のパラメータについて値の複数の仕様が仕様順に保持され、適用されなければならない。

【０２１５】

レベルアイテムにおけるリソースパラメータは、それらが指定される順番で処理される

。

【０２１６】

Delay命令文は、次のレベルグループの設定に先立って、レベル設定の処理をほぼ指示された期間の間休止させる。実際の待機時間は、小さなシステムでは、指定された遅延の付近の定義された範囲である。したがって遅延がｔ秒であれば、実際の遅延は以下を満足する。

ｔ - Δｔ <= 実際の待機 <= ｔ + Δｔ

【０２１７】

Delayの命令文は、Levelsの仕様をいくつものサブシーケンスに分割し、それらのそれぞれは、処理のための別々のテスト条件メモリ設定を必要とする。

【０２１８】

MinDelay命令文は、レベルの次のグループの設定に先立って、レベル設定の処理を少なくとも指定された期間休止させる。実際に待機時間は、小さなシステムにおいては、M指定された最小の遅延の最小値を有する定義された範囲である。したがって、最小の遅延をｔ秒とすると、実際の遅延は以下を満足する。

ｔ <= 実際の待機 <= ｔ + Δｔ

【０２１９】

MinDelay命令文は、Levels仕様をいくつものサブシーケンスに分割し、それらのそれぞれは、処理のための別々のテスト条件メモリ設定を必要とする。

【０２２０】

それぞれのピン名またはピングループ名は、ピン記述ファイル（接尾辞 .pin）において正確に一つのリソースにおいて指定され、したがって、リソースファイル（接尾辞 .rsc）において指定される実行可能なリソースパラメータのあるセットを有している。名前つきの全てのパラメータは、このセットの実行可能なリソースパラメータの間からのものでなければならず、それらの値を設定するのに用いられる式と同じエレメンタリタイプのものでなければならない。リソースパラメータの名前およびタイプについての情報は、リソースファイルからもたらされる。

【０２２１】

リソースファイルResource.rscは、暗黙のうちにインポートされ、dpinおよびdpsのような標準的なリソースのパラメータの名前およびタイプをテスタに与える。

【０２２２】

リソースパラメータは、UserVarsを用いることのできる割り当てられた式、および名前つきの仕様セットあるいは現在見えているローカルな仕様セットからの値である。

【０２２３】

Dpsピンは、特別なパラメータPRE_WAITおよびPOST_WAITを有する。PRE_WAITパラメータは、電力ピンがその到達先の電圧に達した時間からパターン生成がスタートすることができる時間までに過ぎる必要がある時間を指定する。POST_WAITパラメータは、パターン生成が停止した時間から電源ピン遮断される時間までに過ぎる必要がある時間を指定する。

【０２２４】

また、Dpsピンは、どのようにして電圧パラメータがその最終値に到達するかも指定する。それらは、他のピンパラメータのように、それを単に等式によって指定する。その場合、ハードウェアがそれを許すときに最終値に到達する。また、それらは、スルー（Slew）命令文を用いてもそれを指定する。スルー命令文は、電源電圧がその初期値から指定された絶対電圧スルーレートを有する傾斜で最終値に到達することを指定する。

レベルのためのC++

【０２２５】

上記ルールを用いて、以下の動作をサポートするC++レベルオブジェクトを書くことができる。

```
Status setParameter(const String& pinOrPinGroupName,
                    const String& parameterName,
                    ElementaryType elementaryType,
                    const Expression& Expression);
```

【０２２６】
という動作がある。
【０２２７】
この動作は、式をピンまたはピングループのパラメータに結びつける。例えば、dpin.InPins VIH値は、以下によって設定される。

```
setParameter("InPins", "VIH", VoltageT,
             Expression("v_ih + 1.0");
```

【０２２８】
この動作は、レベルオブジェクトにおける全ての宣言について何回か呼び出される。

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

which will go through and issue all the pre-determined module level interfaces to assign all the levels of parameters in specification order, as described earlier. The selector parameter is used to resolve names in the expressions according to the rules specified earlier.

【０２２９】
という動作がある。
【０２３０】
これは、先に述べたように仕様順でパラメータのレベルの全てを割り当てるために、所定のモジュールレベルインタフェースの全てを通り抜け、発行する。セレクタパラメータは、先に指定されたルールに従って、式において名前を解決するために用いられる。

テスト条件グループ
【０２３１】
テスト条件グループサブ言語は、仕様、タイミングおよびレベルの記述を一緒にパッケージ化する。タイミングオブジェクトはしばしばパラメータを用いて指定される。さまざまなパルスの立ち上がり・立ち下がりエッジを指定するために、タイミングにおいてパラメータを用いることができる。同じように、さまざまな電圧レベルの最大、最小および典型的な値を指定することによってレベルをパラメータ化することができる。テスト条件グループ（TCG）オブジェクトは、これらの仕様に基づいて、タイミングおよびレベルの指定とインスタンス化とを一括して扱う。
【０２３２】
TestConditionGroupの宣言は、オプションであるSpecificationSetを含んでいる。SpecificationSetの宣言は、インラインされた（かつ名前がつけられていない）ローカルなSpecificationSetであってもよく、あるいは、どこかで宣言された名前つきのSpecificationSetへの参照であってもよい。TCGの宣言におけるオプション的なSpecificationSetの宣言の後には、少なくとも一つのレベルまたはタイミングの宣言が続く。それは、どのような順番でもレベルおよびタイミングの両方を有することができる。しかしながら、２つ以上のレベルおよびタイミングの宣言を有することは認められていない。これらの制限事項は、構文的に強化される。
【０２３３】
TCGにおける仕様セットの宣言は、名前をもたない点以外は、別に宣言された仕様セットと同一である。その名前は、暗黙のうちには取り囲むTCGの名前である。タイミングの宣言は、指定されたタイミングファイルからのタイミングオブジェクトの単一の宣言を包含している。テスト条件グループを有するファイルの一例がこれである。

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

Version 0.1;

Import pentiumlevels.lvl;
Import edges.spec;
Import timing1.tim;
Import timing2.tim;

TestConditionGroup TCG1
{
    # This Local SpecificationSet uses user-defined selectors
    # "min", "max" and "typ".  Any number of selectors with any
    # user defined names is allowed.
    #
    # The specification set specifies a table giving values for
    # variables that can be used in expressions to initialize
    # timings and levels.  The specification set below defines
    # values for variables as per the following table:
    #             min            max          typ
    #    v_cc       2.9            3.1          3.0
    #    v_ih  vInHigh + 0.0     vInHigh + 0.2 vInHigh + 0.1
    #    v_il  vInLow + 0.0   vInLow + 0.2  vInLow + 0.1
    #  ...
    # A reference such as "vInHigh" must be previously defined
    # in a block of UserVars.
    #
    # Thus, if the "max" selector was selected in a functional
    # test, then the "max" column of values would be bound to
    # the variables, setting v_cc to 3.1, v_ih to vInHigh+2.0
    # and so on.
    #
    # Note that this is a local specification set, and has no
    # name.
    SpecificationSet(min, max, typ)
    {
        # Minimum, Maximum and Typical specifications for
        # voltages.
        Voltage v_cc = 2.9, 3.1, 3.0;
        Voltage v_ih = vInHigh + 0.0,
                         vInHigh + 0.2,
                          vInHigh + 0.1;
        Voltage v_il = vInLow + 0.0,
                        vInLow + 0.2,
                        vInLow + 0.1;

        # Minimum, Maximum and Typical specifications for
```

```
        # leading and trailing timing edges.  The base
        # value of 1.0E-6 uS corresponds to 1 picosecond,
        # and is given as an example of using scientific
        # notation for numbers along with units.
        Time t_le = 1.0E-6 uS,
                    1.0E-6 uS + 4.0 * DeltaT,
                    1.0E-6 uS + 2.0 * DeltaT;
        Time t_te = 30ns,
                    30ns + 4.0 * DeltaT,
                    30ns + 2.0 * DeltaT;
    }

    # Refers to the Pentium3Levels imported earlier.  It
    # is one of possibly many levels objects that have been
    # imported from the above file.
    Levels Pentium3Levels;

    # Refers to file timing1.tim containing the single
    # timing Timing1.  The filename should be quoted if
    # it has whitespace characters in it.
    Timings Timing1;
}

# Another test condition group
TestConditionGroup TCG2
{
    # ClockAndDataEdgesSpecs is a specification set which
    # is available in the edges.specs file.  Assume it has
    # the following declaration:
    #    SpecificationSet ClockAndDataEdgesSpecs(min, max, typ)
    #    {
    #        Time clock_le = 10.00 uS, 10.02 uS, 10.01 uS;
    #        Time clock_te = 20.00 uS, 20.02 uS, 20.01 uS;
    #        Time data_le  = 10.0 uS, 10.2 uS, 10.1 uS;
    #        Time data_te  = 30.0 uS, 30.2 uS, 30.1 uS;
    #    }
    # A SpecificationSet reference to this named set is below:
    SpecificationSet ClockAndDataEdgesSpecs;

    # An inlined levels declaration.  Since the associated
    # specification set (above) does not have variables such
    # as VInLow, VInHigh, VOutLow and VOutHigh, they must
    # resolve in the default UserVars collection.
    Levels
    {
        InPins { VIL = VInLow; VIH = VInHigh + 1.0; }
        OutPins { VOL = VOutLow / 2.0; VOH = VOutHigh; }
    }

    # This Timing is from the file "timing2.tim".  The timings
```

```
    # will need the leading and trailing edge timings for clock
    # and data as specified in the above specification set.
    Timings Timing2;
}
```

【０２３４】

上記例においては、テスト条件グループTCG１は、「min」、「typ」および「max」という名前の３つのセレクタを有する仕様セットを記述している。異なったセレクタがいくつあってもよい。仕様セットのボディの中には、変数v_il、v_ih、t_leおよびt_teが、セレクタに対応する、値の３倍で初期化されている。したがって上記例では、セレクタ「min」を有するTCGIのインスタンスは、変数v_ilを第一の数値（vInputLow+0.0）に結びつける。これは、仕様セット用のセレクタがユーザ定義のものであり、どのような数のセレクタも許されるということの繰り返しを行う。ただ一つの要件は、以下の通りである。

【０２３５】

仕様セット用のセレクタは、固有の識別子でなければならない。

【０２３６】

仕様セットで指定されるそれぞれの値は、セレクタのセットときっちり同じ数の要素の値のアレイに関連付けられる。ｉ番目のセレクタを取り出すことによって、各値は、関連する値のベクトルのうちのｉ番目の値に結び付けられる。

【０２３７】

TCGにおける仕様セットの後に、レベル宣言あるいはタイミング宣言あるいはその両方があり得る。レベル宣言はさまざまなピンパラメータについてレベルを設定するために用いられる。仕様セットにおいて特定された変数はこれらのレベルを設定するために用いられ、TCGを初期化するために用いられるセレクタに基づいて、ピンパラメータについて異なる実際の値を動的に結びつけることを許容する。

【０２３８】

これを例示するために、セレクタ「min」を有効にするテストを考える。ページに示された仕様セットPentium3Levelsを参照すると、InPinsグループのピンに関するピンパラメータ「VIH」は、宣言によって式(v_ih + 1.0)に初期化される。

```
    InPins { VIL = v_il; VIH = v_ih + 1.0; }
```

【０２３９】

これは、セレクタ「min」が有効にされると、(VinHigh + 0.0 + 1.0)を解く。同様にタイミングオブジェクトを、仕様セット変数の選択された値に基づいて初期化することができる。タイミング宣言とレベル宣言との両方を有する必要はない。以下の例で示すように、いずれかが単独で存在することもできるし、両方がどんな順番で存在することもできる。

```
# ------------------------------------------------------
# File LevelsOnlyAndTimingsOnly.tcg
# ------------------------------------------------------

Version 0.1;

# A Levels-only Test Condition Group.
TestConditionGroup LevelsOnlyTCG
{
    SpecificationSet(Min, Max, Typ)
    {
        Voltage v_il = 0.0, 0.2, 0.1;
        Voltage v_ih = 3.9, 4.1, 4.0;
```

```
    }

    # An inlined levels declaration.  Since the associated
    # specification set (above) does not have variables such
    # as VInLow, VInHigh, VOutLow and VOutHigh, they must
    # resolve in the default UserVars collection.
    Levels
    {
        InPins { VIL = v_il; VIH = v_ih + 1.0; }
        OutPins { VOL = v_il / 2.0; VOH = v_ih; }
    }
}


# A Timings-only Test Condition Group
TestConditionGroup TimingsOnlyTCG
{
    SpecificationSet(Min, Max, Typ)
    {
        Time t_le = 0.9E-3, 1.1E-3, 1.0E-3;
    }

    Timings Timing2;
}
```

【０２４０】
しかしながら、TCGにおいては２つ以上のタイミングおよび２つ以上のレベルがあってはならないことに留意されたい。したがって、まとめると、タイミングまたはレベルの少なくとも一つがなければならず、それぞれは多くても一つでなければならない。

テスト条件

【０２４１】
テスト条件（TestCondition）オブジェクトは、TCGを具体的なセレクタに結びつける。上で示すようにTCGが一旦宣言されると、以下に示すようにテスト条件（TestCOndition）オブジェクトを宣言することができる。

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

TestCondition TCTyp
{
    TestConditionGroup = TCG1;
    Selector = typ;
}

TestCondition TCMax
{
    TestConditionGroup = TCG1;
```

```
    Selector = max;
}
```

【０２４２】

テスト条件は以下のテーブルで宣言される。

```
#
# Declare a FunctionalTest "MyFunctionalTest" that refers to three
# Test Condition Group instances.
#
Test FunctionalTest MyFunctionalTest
{
    # Specify the Pattern List
    PList = pat1Alist;
    # Any number of TestConditions can be specified:
    TestCondition = TCMin;
    TestCondition = TCMax;
    TestCondition = TCTyp;
}
```

TCG（テスト条件グループ）における名前の解決

【０２４３】

テスト条件グループにおける名前の解決は先に論じた。しかし、これらのルールを反復し、以下にもう一度示す。

【０２４４】

１．もし名前が限定されていれば（ページ参照）、それは名前つきユーザ変数コレクションにおいて解決されなければならない。

【０２４５】

２．もし名前が限定されていなければ、その名前は、テスト条件グループで宣言されていればローカルな仕様セットで解決され、それがテスト条件グループで参照されていれば名前つき仕様セットで解決される。

【０２４６】

３．もし名前が上記ルールで解決されなければ、デフォルトユーザ変数コレクションで解決される。

TCGランタイム

【０２４７】

テスト条件グループは、以下のランタイムセマンティクスを有する。

【０２４８】

テスト（機能テスト（FunctionalTest）等）は、インスタンス化されたテスト条件（TestCondition）を用いて、その仕様セット（SpecificationSet）からの特定のセレクタを有するTCGを参照する。このセレクタは、仕様セット（SpecificationSet）中の各変数を、選ばれたセレクタに関連する値に結びつける。そしてこの変数の値への結びつけは、レベルおよびタイミングを決定するのに用いられる。

【０２４９】

テスト条件グループ（TestConditionGroup）におけるパラメータレベルは、好ましくは、レベルブロックにおいて提示される順番で、順次設定される。したがって、Pentium3Levelsブロックにおいては、パラメータレベルが設定される順番は次のようになる（表記の仕方：＜リソース名＞．＜リソースパラメータ＞）。

```
InputPins.VIL
InputPins.VIH
```

OutputPins.VIL
OutputPins.VIH
Clock.VOL
Clock.VOH
【０２５０】
このシーケンス順は、テストライターが電源の明示的な電力シーケンスを制御することを可能にする。さらに、もしレベルアイテムが、ピンについての同じピンパラメータの名前を挙げながら二度登場すれば、そのピンパラメータは二度設定される。これはプログラム的にも起こることができる。
もしパラメータがスルー（Slew）命令文

VCC = Slew(0.01, 2.0 V);
【０２５１】
によって設定されれば、それは、VCCが現在の値から、1秒あたり±0.01の電圧スルーレートを有する傾斜で最終値2.0ボルトに達するということを意味する。
【０２５２】
また、仕様セット変数は、TCGにおけるタイミングオブジェクトにも渡される。そしてタイミングオブジェクトは、選択された変数に基づいて初期化される。このようなメカニズムは、例えば、波形の立ち上がり・立ち下がりエッジを指定することによって、タイミングオブジェクトをカスタマイズするために用いられることができる。

TCGのためのC++
【０２５３】
上記ルールを用いて、テスト条件グループをC++TestConditionGroupクラスにおいて宣言することができ、次のようにそれを初期化することができる。
TestConditionGroupメンバ関数

Status setSpecificationSet(SpecificationSet *pSpecificationSet);
【０２５４】
に対してコールが行われる。これは、TestConditionGroupについて仕様セットを設定するものである。これは、ローカル仕様セットか、名前つきの仕様セットか、あるいはヌル（何もなければ）であり得る。
TestConditionGroupメンバ関数

Status setLevels(Levels *pLevels);
【０２５５】
に対してコールが行われる。これは、TestConditionGroupについてレベルオブジェトを設定するものである。これは、ローカルに宣言されたレベルオブジェクトか、外部で宣言されたレベルオブジェクト化、あるいはヌル（何もなければ）であり得る。
TestConditionGroupメンバ関数

Status setTimings(Timings *pTimings);
【０２５６】
に対してコールが行われる。これは、TestConditionGroupについてタイミングオブジェクトを設定するものであり、外部で宣言されたタイミングオブジェクトか、ヌル（何もなければ）であり得る。

ビン（Bin）定義
【０２５７】
ビン（Bin）定義クラスは、ビン（bin）、多くのDUTをテストした結果を要約するカウ

ンタのコレクションを定義する。あるDUTのテスト中に、そのDUTは、例えば特定のテストの結果を示すために、いずれかのビンに設定される。テストが進行するにつれて、そのDUTは他のビンに設定されてもよい。DUTが最終的に設定されるビンは、そのテストの最後でこのように設定される最後のものである。この最終のビンについてのカウンタは、このDUTのテストの最後にインクリメントされる。ビン定義を有する別個のファイルは、接尾辞「 .bdefs」を有していなければならない。

【０２５８】

ビン定義は好ましくは、階層的である。例えば、最外層のレベルでは、合格および不合格と名付けられた２つのビンを有するPassFailBinであり得る。そして、いくつかのHardBinsが存在し得、この中には合格ビンへマップされるものも、不合格ビンへマップされるものもある。HardBinはPassFailBinのリファインメントであると考えることができる。最後に、多くの数のSoftBin、HardBinsのリファインメントが存在し得、これらの多くは同じハードビンへマップされる。ビンの階層を示す例を以下に示す。

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

Version 1.2.3;

BinDefs
{
    # The HardBins are an outermost level of
    # bins.  They are not a refinement of any other
    # bins.
    BinGroup HardBins
    {
        "3GHzPass":        "DUTs passing 3GHz";
        "2.8GHzPass":    "DUTs passing 2.8GHz";
        "3GHzFail":        "DUTs failing 3GHz";
        "2.8GHzFail":    "DUTs failing 2.8GHz";
        LeakageFail:    "DUTs failing leakage";
    }

    # The SoftBins are a next level of refinement.
    # SoftBins are a refinement of HardBins.
    BinGroup SoftBins : HardBins
    {
        "3GHzAllPass":
            "Good DUTs at 3GHz",    "3GHzPass";
        "3GHzCacheFail":
            "Cache Fails at 3GHz", "3GHzFail";
        "3GHzSBFTFail":
            "SBFT Fails at 3GHz",    "3GHzFail";
        "3GHzLeakage":
            "Leakages at 3GHz",        LeakageFail;
        "2.8GHzAllPass":
            "Good DUTs at 2.8GHz",    "2.8GHzPass";
        "2.8GHzCacheFail":
            "Cache Fails at 2.8GHz","2.8GHzFail";
```

```
        "2.8GHzSBFTFail":
            "SBFT Fails at 2.8GHz",    "2.8GHzFail";
        "2.8GHzLeakage":
            "Leakages at 2.8GHz",    LeakageFail;
    }
}
```

【０２５９】

上記例では、大半の基本ビンは、BinGroup HardBinsである。BinGroup Xは、何か他のBinGroupがXのリファインメントであれば、基本ビンのグループであると考えることができる。したがって、BinGroup HardBinsは、BinGroup SoftBinsがHardBinsのリファインメントであるので、基本ビンのグループである。SoftBinsのビンをリーフ（leaf）ビンと呼ぶ。BinGroup Yは、他のBinGroupがYのリファインメントでなければリーフビンのグループであると考えることができる。

【０２６０】

単一のBinGroup Zを有するBinDefsブロックの悪化した場合は、Zを大半の基本ビンのグループであるとともにリーフビンのグループとする。BinGroupの名前は、範囲においてはグローバルである。何個のBinDefsブロックがあってもよいが、宣言されたBinGroupは異なっていなければならない。一つのBinDefsブロックからのBinGroupは、他のBinDefsブロックのBinGroupのリファインメントであることが可能である。したがって、上記例では、SoftBinsは、HardBinsからの別個のBinDefsブロック中に存在し得る。しかしながら、読みやすさのために定義される全てのBinGroupを単一のBinDefsブロックにもたせることが強く推奨される。

【０２６１】

ここで、上記階層を、他のBinGroupを追加することによって、何個のDUTが合格、不合格であったかをカウントするように拡張することができる。

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

Version 1.2.3;

BinDefs
{
    # The PassFailBins are an outermost level of
    # bins.  They are not a refinement of any other
    # bins.
    BinGroup PassFailBins
    {
        Pass: "Count of passing DUTS.";
        Fail: "Count of failing DUTS.";
    }

    # The HardBins are a next level of refinement.
    # HardBins are a refinement of the PassFailBins,
    # as indicated by "HardBins : PassFailBins".
    BinGroup HardBins : PassFailBins
    {
        "3GHzPass":        "DUTs passing 3GHz",    Pass;
        "2.8GHzPass":    "DUTs passing 2.8GHz",    Pass;
```

```
        "3GHzFail":        "DUTs failing 3GHz",    Fail;
        "2.8GHzFail":    "DUTs failing 2.8GHz",    Fail;
        LeakageFail:    "DUTs failing leakage",    Fail;
    }

    # The SoftBins are a next level of refinement.
    # SoftBins are a refinement of HardBins.
    BinGroup SoftBins : HardBins
    {
        "3GHzAllPass":
            "Good DUTs at 3GHz",    "3GHzPass";
        "3GHzCacheFail":
            "Cache Fails at 3GHz", "3GHzFail";
        "3GHzSBFTFail":
            "SBFT Fails at 3GHz",    "3GHzFail";
        "3GHzLeakage":
            "Leakages at 3GHz",        LeakageFail;
        "2.8GHzAllPass":
            "Good DUTs at 2.8GHz",    "2.8GHzPass";
        "2.8GHzCacheFail":
            "Cache Fails at 2.8GHz","2.8GHzFail";
        "2.8GHzSBFTFail":
            "SBFT Fails at 2.8GHz",    "2.8GHzFail";
        "2.8GHzLeakage":
            "Leakages at 2.8GHz",    LeakageFail;
    }
}
```

【０２６２】

このとき、大半の基本ビンはBinGroup PassFailBinsである。これらは典型的には、どのビンのリファインメントでもない。BinGroup HardBinsはPassFailBinsのリファインメントであり、基本ビンでもある。SoftBinsはHardBinsのリファインメントであり、リーフビンのグループである。上記例は、階層中に３つのBinGroupのみを有していた。以下は、より複雑な階層である。

```
BinDefs
{
    # A group of most base bins
    BinGroup A { ... }

    # A group of base bins that is a refinement of A
    BinGroup Ax : A { ... }

    # A group of leaf bins that is a refinement of Ax
    BinGroup Axx : Ax { ... }

    # A group of base bins that is a refinement of A
    BinGroup Ay : A { ...}

    # A group of leaf bins that is a refinement of Ay
    BinGroup Ayy : Ay { ... }
```

```
    # A group of most base bins
    BinGroup B { ... }

    # A group of leaf bins that is a refinement of B
    BinGroup Bx : B { ... }
}
```

【０２６３】

この例では、AxとAyとがAのリファインメントであり、AxxがAxのリファインメントであり、AyyはAyのリファインメントである。またこの例は、BinGroup BおよびBxを与え、BxはBのリファインメントである。PassFailBins、HardBinsおよびSoftBinsと名づけられたBinGroupを有する上記BinDefsの宣言は、このセクションにおいて引き続いて述べられる例として用いられる。

BinGroupにおける各ビンは、

１．　識別子あるいはリテラルな文字列である名前と、

２．　このビンが要約しているものが何であるかを述べる記述と、

３．　このビンがリファインメントBinGroupの中のものであれば、それがリファインメントである、基本ビンとしても知られるビンの名前と

【０２６４】

を有している。

【０２６５】

PassFailBinsにおける２個のビンは、「Pass」および「Fail」と名付けられている。HardBinsにおける５個のビンは、「3GhzPass」、「2.8GhzPass」、「3GhzFail」、「2.8GhzFail」「LeakageFail」と名付けられている。ビン名はリテラルな文字列であっても識別子であってもよい.ビン名はBinGroup内で固有のものでなければならないが、BinGroup間で重複してもよい。しかしBinGroup名は、範囲においてはグローバルであり、テストプランにわたって固有のものでなければならない。

【０２６６】

５個のHardBinsのうち、ビン「3GhzPass」および「2.8GhzPass」は、ともにPassFailBinsの「Pass」ビンへマップされる。HardBinsの残りは、PassFailBinsの「Fail」ビンにマップされる。

【０２６７】

最後に、８個のSoftBinsがある。SBFT（ソフトビン機能テスト）およびCacheに関して3GHzでの２つの不合格は、「3GhzFail」のHardBinへマップされる。同様に、STFTおよびCacheに関して2.8GHzでの2つの不合格は、「2.8GhzFail」HardBinへマップされる。漏れによる不合格はともに、それらが起こる速度に関わらず、同じ「LeakageFail」HardBinへマップされる。例えば、最も粗いテスト（最外レベルでの）は、DUTがテストに合格するか、不合格であるかである。リファインメントは、例えば、DUTが特定の周波数、例えば3GHz等でテストに合格するか、不合格であるかである。

【０２６８】

ビンは、下で述べるように、TestPlan FlowItemにおいてDUTに割り当てられる。TestPlan FlowItemは、テストを実行することから特定の結果を出すことの結果として起こるアクションおよび変化をテストプランが記述している結果節を有している。SetBin命令文が起こり得るのはこの時点である。

```
# A FlowItem Result clause.  It is described later.
Result 0
{
    # Action to be taken on getting a 0 back from
    # executing a test.
```

```
    # Set the bin to SoftBin."3GHZPass" expressing that the
    # DUT was excellent.
    SetBin SoftBins."3GHzPass";
}
```

【０２６９】

多くのSetBin命令文がDUTについてのテストを動作させている間に実行され得る。テストが最終的に完了すると、ランタイムはそのDUTについて、ならびに全てのそのリファインメントについて設定される最終ビンについてのカウンタをインクリメントする。テスト中に実行される以下のSetBin命令文を有したDUTを考える。

```
SetBin SoftBins."3GHzSBFTFail";
SetBin SoftBins."2.8GHzAllPass";
```

【０２７０】

このDUTは、3Ghz Cacheテストおよび漏れ（Leakage）テストに合格したが、SBFTテストには不合格であり、したがって、「3GHzSBFTFail」ビンに割り当てられる。そして2.8GHzでテストされ、全てのテストに合格した。この最終的な割り当ては、以下のビンのカウンタをインクリメントすることになる。

１．　SoftBins「2.8GhzAllPass」

２．　HardBins「2.8GhzPass」のリファインメントであるもの

３．　PassFailBins「Pass」のリファインメントであるもの

【０２７１】

テストが完了すると、ランタイムはDUTの最終的なビン割り当てのカウンタをインクリメントし、それがリファインメントであるような全てのほかのビンについてもインクリメントする。

【０２７２】

SetBin命令文は、リーフビン上においてのみ許される。基本ビンを設定することは不正である。上記カウンタをインクリメントするセマンティクスは、

１．　もしそのビンがリーフビンであれば、それは、DUTのテストの最後にこのビンについてSetBins命令文が実行された回数である

２．　もしそのビンが基本ビンであれば、それは、そのビンがリファインメントであるようなビンのカウンタの合計である

【０２７３】

ということを確実にする。

【０２７４】

したがって、上記例では、SoftBinsのみがSetBin命令文において許容される。HardBins「LeakageFail」についてのカウンタは、SoftBins「3GhzLeakageFial」およびSoftBins「2.8GhzLeakageFail」についてのカウンタの合計である。ビン定義に関するいくつかのルールは以下の通りである。

【０２７５】

１．　BinDefinitions宣言は、いくつかのBinGroup宣言からなる。

【０２７６】

２．　各BinGroup宣言は、名前、それがリファインメントであるオプション的なBinGroup名を有しており、それにビン宣言のブロックが続く。

【０２７７】

３．　ビン宣言は、名前と、その後に続く宣言とを備えており、オプション的にその後に、このビンがリファインメントであるところの基本ビンの名前が続く。

【０２７８】

４．　ビン名は、リテラルな文字列であってもよいし、識別子であってもよい。空の文字列は有効なビン名ではありえない。ビン名は、BinGroup宣言における名前の間では固有

のものでなければならないが、同じ名前は他のBinGroup宣言で用いられ得る。
【０２７９】
５．　もしBinGroup宣言Xxxが他のBinGroup宣言Yyyのリファインメントであれば、Xxxにおけるビン宣言の全ては、Yyyからの基本ビンの名前を宣言しなければならない。したがって、SoftBinsはHardBinsのリファインメントであると宣言されているので、SoftBinsにおけるビン宣言のそれぞれは、HardBinsのビンのリファインメントである。
【０２８０】
６．　PassFailBinsのような、他のBinGroup宣言のリファインメントではないというBinGroup宣言は、好ましくは、基本ビンを宣言していないビン宣言を有する。
【０２８１】
ビンBbbは、Bbbがリファインメントであるようなビンの完全なセットであるベース（base）のセットを有する。それは正式には次のように定義される。
【０２８２】
１．　AaaがBbbの基本ビンであれば、AaaはBbbのベースのセット中にある。
【０２８３】
２．　Aaaのどのベースも、Bbbのベースのセット中にもある。
【０２８４】
BinGroup名はTestPlanにおいてはグローバルである。
【０２８５】
ビン名は、BinGroupにローカルなものである。
【０２８６】
SetBin命令文は、リーフビンについてのみ許される。

ビン定義のためのC++
【０２８７】
上記ルールを用いて、オブジェクトタイプBinGroupをBinDefs宣言におけるBinGroup宣言のそれぞれについて構築することができる。クラスBinGroupは、サブクラスLeafBinGroupを有する。これらの２つのクラスの動作は、BinGroup::incrementBinがC++で保護動作であるのに対して、LeafBinGroup::incrementBinはC++パブリック動作であること以外は同じである。
【０２８８】
以下は、他のどのBinGroupのリファインメントではないBinGroupあるいはLeafBinGroupを構築するデフォルトコンストラクタである。
コンストラクタ：

```
BinGroup(BinGroup& baseBinGroup);
LeafBinGroup(BinGroup& baseBinGroup);
```

that builds a BinGroup that is a refinement of the given baseBinGroup.
【０２８９】
これは任意のbaseBinGroupのリファインメントであるBinGroupを構築する。

```
Status addBin(const String& binName,
              const String& description,
              const String& baseBinName);
```

【０２９０】
という方法は、ビンおよびその記述を定義するためのものである。もしそれが最も基本のビンであれば、baseBinNameパラメータは空の文字列でなければならない。
ビンカウンタをインクリメントするための方法は

```
Status resetBin(const String& binName);
```

【０２９１】
である。この動作は、このビンについて、ならびにこのビンのベースである全てのビンについてのカウンタをインクリメントする。この動作はクラスBinGroupにおいては保護されており、クラスLeafBinGroupにおいてはパブリックである。
ビンカウンタをリセットする方法は、

```
Status incrementBin(const String& binName);
```

【０２９２】
である。この動作は、このビン、ならびにこのビンのベースであるすべてのビンについてのカウンタをリセットする。
ビンについての情報を取得する方法は、

```
Status getBinDescription(const String& binName,
                         String& description);
Status getBaseBin(const String& binName,
                  BinGroup* pBaseBinGroup,
                  String& baseBinName);
Status getBinValue(const String& binName,
                   unsigned int& value);
```

【０２９３】
である。
【０２９４】
現在定義されている全てのビン名を取得するためには、反復子が与えられる。
【０２９５】
TestPlanの状態は、いくつかのBinGroupメンバを含み、それぞれは各BinGroup宣言についてのものである。上記BinDefinitionsについてのC++は次のようになる。

```
// TestPlan constructor
TestPlan::TestPlan()
: m_PassFailBins(), // Default Constructor
  m_HardBins(&m_PassFailBins),
  m_SoftBins(&m_HardBins)
{}

// Bin initializations
m_PassFailBins.addBin("Pass", "Count of passing DUTS.","");
m_PassFailBins.addBin("Fail", "Count of failing DUTS.","");
m_HardBins.addBin("3GHzPass", "Duts passing 3GHz", "Pass");
...
```

【０２９６】
TestPlanについての状態は、未定義のBinGroup（NULL）とm_currentBin未定義ビン名（空の文字列）とに初期化されるm_pCurrentBinGroupを含んでいる。SetBin命令文が実行される都度、m_pCurrentBinGroupは、コールによって示されている名前付きのBinGroupへと変わり、m_CurrentBinはそのグループにおける名前付きのビンに変わる。

```
// Translation of: SetBin SoftBins."3GHzAllPass";
pTestPlan→setBin("SoftBins", "3GHzAllPass");
```

【０２９７】
テストプランが実行されると以下の呼び出しがなされる。

```
m_pCurrentBinGroup→incrementBin(m_currentBin);
```

【０２９８】
これによって、このビンおよびその全てのベースビンは、それらのカウンタをインクリメントさせる。
【０２９９】
BinGroupカウンタは、テストプランが詳述されるときにリセットされるが、テストが動作されるたびに再度初期化されるわけではない。カウンタは、BinGroup::resetBinへの明示的なコールによってリセットすることができる。

Ｃ．　テストプラン
【０３００】
テストプランは、テストプログラムの主要な構造であると考えることができる。テストプランはファイルをインポートすることができるとともに、同様のコンストラクトインラインを定義することができる。したがって、ファイルにいくつかのグローバルの任意の定義をインポートするとともに、追加のグローバルのインラインを宣言することができる。

Ｃ１．　テストプランフローおよびフローアイテム
【０３０１】
テストプランの重要な要素の一つはフローである。フローは、有限状態マシンをカプセル化する。これは、IFlowableオブジェクトを動作させ、それから他のFlowItemへと移行するいくつかのFlowItemを備えている。IFlowableを動作させることには、IFlowableインタフェースをインプリメントするオブジェクトを動作させることが含まれる。IFlowableインタフェースをインプリメントする典型的なオブジェクトは、テストおよびフローである。
【０３０２】
したがって、フローはテストおよび他のフローを動作させ、そして他のFlowItemへと移行するFlowItemを有している。また、これは、ユーザがカスタマイズしたルーチンを、IFlowableを動作させることからのさまざまなリターン結果上に呼び出す機会を提供する。したがって、典型的にはフローは以下の形態を有する。

```
#
# FlowTest1 implements a finite state machine for the
# Min, Typ and Max flavors of MyFunctionalTest1.  On
# success it tests Test1Min, Test1Typ, Test1Max
# and then returns to its caller with 0 as a successful
# status.  On failure, it returns 1 as a failing status.
#
# Assume that the tests MyFunctionalTest1Min, ... all
# return a Result of 0 (Pass), 1 and 2 (for a couple
# of levels of failure).
#                   Result 0        Result 1        Result 2
#    Test1Min       Test1Typ        return 1        return 1
#    Test1Typ       Test1Max        return 1        return 1
#    Test1Max       return 0        return 1        return 1
#
Flow FlowTest1
{
    FlowItem FlowTest1_Min MyFunctionalTest1Min
    {
        Result 0
        {
```

```
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            GoTo FlowTest1_Typ;
        }

        Result 1
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            Return 1;
        }

        # This result block will be executed if
        # MyFunctionalTest1Min returns any of
        # 2, 5, 6, 7, -6, -5 or -4
        Result 2, 5:7, -6:-4
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            Return 1;
        }
    }

    FlowItem FlowTest1_Typ { ... }
    FlowItem FlowTest1_Max { ... }
}
```

【０３０３】

フローFlowTest1の動作は次の通りである。

【０３０４】

１．　FlowItem FlowTest1_Minを実行することで始まる。

【０３０５】

２．　FlowTest1_Minは機能テストMyFunctionalTestMinを動作させる。このテストの詳細は、完全なテストプランを以下で提示するときに、与えられる。

【０３０６】

３．　このテストを動作させることで９個の結果、０、１、２、５、６、７、－６、－５または－4が期待される。最初の２つの結果節がそれぞれ０と１とを扱い、３つ目は結果値の残りの全てを扱う。

【０３０７】

４．　もし結果「０」（合格）が出現すれば、FlowTest1_MinはカウンタPassCounterをインクリメントする。そして新しいFlowItem FlowTest1_Typに移行する。

【０３０８】

５．　もし結果「１」または結果「２」が出現すれば、FlowTest1_MinはカウンタFailCounterをインクリメントし、フローから戻る。

【０３０９】

６．　FlowTest1_Typは同じように動作し、続けてFlowTest1_Maxを呼び出す。

【０３１０】

７．　FlowTest1_Maxが同じように動作し、続けて、正常な結果（「０」）でFlowTest1から戻る。

【０３１１】

したがって、FlowTest1は、正常な動作で、Test1の最小、典型および最大バージョンを

通じて装置を動作させ、そして戻る。FlowTest2は同じようにして動作する。
【０３１２】
上述したフローは、基本的には、状態および遷移を有する有限状態マシンを記述している。FlowItemは基本的には状態であり、以下のことを行う。
【０３１３】
１．　IFlowable（これは先に定義されたフロー、あるいはテスト、あるいは上記ルールでC++でインプリメントされ得るユーザ定義のフローであり得る）を実行する。
【０３１４】
２．　IFlowableの実行は数字の結果を返す。この結果に基づいて、あるアクションが起こり（いくつかのカウンタを更新し）、そして２つのことのうちの一方が起こる。
【０３１５】
ａ．フローは数字の結果で呼び出し元に答える。
【０３１６】
ｂ．フローは、他の状態（FlowItem）に遷移することで続行する。
【０３１７】
したがって、FlowItemは以下のコンポーネントを有する。
【０３１８】
FlowItemは名前を有する。
【０３１９】
FlowItemは実行されるべきIFlowableを有する。
【０３２０】
FlowItemは数あるいは結果節を有する。
【０３２１】
FlowItemの各結果節は、アクションを提供し、遷移で終了し、一つ以上の結果値に関連付けられている。
【０３２２】
これらのアイテムは、構文的にはFlowItemにおいて以下の通りとなる。

```
FlowItem <name> <IFlowable to be executed>
{
    Result <one or more result values>
    {
        <actions for these result values>
        <transition for these result values>
    }

    Result <one or more other result values>
    {
        ...
    }
    ...
}
```

【０３２３】
実行されるべきIFlowableは、テスト、あるいはユーザ定義のIFlowable、あるいはフローであり得る。結果についてのアクションは、次のいずれかである。
【０３２４】
GUIツールによって用いられる文字列の重要な構成要素を属性結果に設定するプロパティアクション。これは上記FlowTest１の例において見られ、

```
Property PassFail = "Pass";
```

【０３２５】
プロパティは基本的には、結果節と関連付けられた、名前付きの文字列あるいは整数の重要な構成要素である。これらは何個あってもよく、好ましくは、ユーザがこの結果に関連付けられた情報を表示するのに用いるGUIのようなツールによって用いられる。それらは、テストの実際の結果、あるいはテストのフローには何の影響も及ぼさない。
【０３２６】
いくつかのカウンタをインクリメントするカウンタアクション。これは上記例において

```
IncrementCounters PassCount;
```

【０３２７】
で見られる。
【０３２８】
任意のあるいはユーザルーチンを呼び出すルーチンコールアクション。これは後で説明する。
【０３２９】
最後に、FlowItemは、他のFlowItemへの移動制御のためのGoTo命令文、あるいは呼び出し元（呼び出しフロー、あるいはテストプランを開始するシステムルーチン）への移動制御のためのReturm命令文であり得る遷移（Transition）を有する。

予め定義されたフロー
【０３３０】
フローオブジェクトの典型的な使用は、テストのシーケンスを定義することである。このシーケンスはその後、テストプランサーバ（TPS）において起こるイベント、すなわち実行テストプランイベント（Execute Test Plan event）の結果として実行される。各サイトコントローラ上のテストプランサーバは、ユーザのテストプランを実行する。しかしながら、フローオブジェクトは、他のイベントにも応答して実行される。括弧内の名前は、フローをこれらのイベントに割り当てる際に用いられる名前である。
【０３３１】
１．　システムロードフロー（SysLoadFlow）。このフローは、テストプランが一つ以上のサイトコントローラ上にロードされるときにシステムコントローラ上で実行される。これは、どのサイトコントローラ上でのテストプランの実際のロードよりも前に実行される。このフローによって、テストプランディベロッパは、システムコントローラに由来すべきであるアクションを定義することが可能である。このようなアクションは、パターンファイル、較正アクションなどのブロードキャストロードを含む。
【０３３２】
２．　サイトロードフロー（SiteLoadFlow）。このフローは、テストプランがサイト上にロードされ、初期化された後にサイトコントローラ上で実行される。これにより、サイト特有の初期化が起こすことができる。
【０３３３】
３．　ロット開始／終了フロー（LotStartFlow/LotEndFlow）。これらのフローは、テストプランサーバが新しいロットの開始を知らされたときにサイトコントローラ上で実行される。これは典型的には、データログストリームにロット特有の情報を用いて注釈をつけるために製造環境で用いられる。
【０３３４】
４．　DUT変更フロー（DutChangeFlow）。このフローは、DUT情報が変化するとサイトコントローラ上で実行される。また、これは典型的にはデータログストリームを更新するために製造環境で用いられる。
【０３３５】
５．　テストプラン開始／終了フロー（TestPlanStartFLow/TestPlanEndFlow）。これらのフローは、テストプランサーバが現在のテストフローの実行を始めるように指示され

たとき、およびそのフローが実行を終えたときにサイトコントローラ上で実行される。
【０３３６】
６．　テスト開始／終了フロー（TestStartFlow/TestEndFlow）。これらのフローは、テストフローが新しいテストを動作させ始めたとき、およびそのテストが実行終了したときにサイトコントローラ上で実行される。
【０３３７】
７．　テストフロー（TestFlow）。このフローは、テストプランサーバが「Execute Test Plan」メッセージを受け取ったときに実行される主要なフローオブジェクトである。
【０３３８】
もしユーザが、TestFlowあるいは他の予め定義されたフローの一つではないユーザテストプランにおいてフローを定義すれば、それを実行させる好ましい方法は、それをこれらの予め定義されたフローの一つの遷移状態に含めることである。

テストプラン例
【０３３９】
以下の例では、フローは、フローによってインプリメントされる有限状態マシンを記述するコメントに沿って与えられる。有限状態マシンは、遷移マトリクスとして与えられる。マトリクスの行はFlowItemに対応し、列は結果に対応する。マトリクスの行のエントリは、戻された結果が列で指定された値であるときにその行のFlowItemからの遷移先のFlowItemを示す。
【０３４０】
3つのフローFlowTest1、FlowTest2およびFlowMainを有するテストプランを以下に示す。FlowTest1は、上述したように動作する。これは、「min」、「typ」および「max」のコンフィギュレーションのそれぞれにおいてMyFunctionalTest1と名付けられたテストを動作させる。同様に、FlowTest2は、これらのコンフィギュレーションのそれぞれにおいてMyFunctionalTest2を動作させる。最後に、FlowMainはFlowTest1とFlowTest2とを動作させる。有限状態マシンの遷移マトリクスは、これらのフローのそれぞれの開始時のコメントにおいて提供される。

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

Version 0.1;

Import xxx.pin;        # Pins

# Constants and variables giving limiting values.
Import limits.usrv;

# Import test condition groups
Import myTestConditionGroups.tcg;

# Import some bin definitions.
Import bins.bdefs;

#-------------------------------------------------------
# Start of the test plan
#-------------------------------------------------------
```

```
TestPlan Sample;

# This block defines Pattern Lists file-qualified names and
# Pattern List variables that are used in Test declarations.
# Pattern list variables are deferred till customization is
# examined.
PListDefs
{
    # File qualified pattern list names
    pl1A.plist:pat1Alist,
    pl2A.plist:pat2AList
}

# The socket for the tests in this test plan (this is not imported,
# but resolved at activation time):
SocketDef = mytest.soc;

# Declare some user variables inline
UserVars
{
    # String name for current test
    String CurrentTest = "MyTest";
}


TestCondition TC1Min
{
    TestConditionGroup = TCG1;
    Selector = min;
}

TestCondition TC1Typ
{
    TestConditionGroup = TCG1;
    Selector = typ;
}

TestCondition TC1Max
{
    TestConditionGroup = TCG1;
    Selector = max;
}

# Likewise for TC2Min, TC2Typ, TC2Max ...


#
# Declare a FunctionalTest.  "FunctionalTest" refers to a C++
# test class that runs the test, and returns a 0, 1 or 2 as
# a Result.  The Test Condition Group TCG1 is selected with
```

```
# the "min" selector by referring to the TC1Min TestCondition.
#
Test FunctionalTest MyFunctionalTest1Min
{
    PListParam = pat1AList;
    TestConditionParam = TC1Min;
}

# Another FunctionalTest selecting TCG1 with "typ"
Test FunctionalTest MyFunctionalTest1Typ
{
    PListParam = pat1AList;
    TestConditionParam = TC1Typ;
}

# Another FunctionalTest selecting TCG1 with "max"
Test FunctionalTest MyFunctionalTest1Max
{
    PListParam = pat1AList;
    TestConditionParam = TC1Max;
}

# Now select TCG2 with "min"
Test FunctionalTest MyFunctionalTest2Min
{
    PListParam = pat2AList;
    TestConditionParam = TC2Min;
}

# Likewise for TCG2 with "typ" and TCG2 with "max"
Test FunctionalTest MyFunctionalTest2Typ
{
    PListParam = pat1AList;
    TestConditionParam = TC2Typ;
}

Test FunctionalTest MyFunctionalTest2Max
{
    PListParam = pat1AList;
    TestConditionParam = TC2Max;
}

#
# At this time the following Test objects have been defined
#        MyFunctionalTest1Min
#        MyFunctionalTest1Typ
#        MyFunctionalTest1Max
#        MyFunctionalTest2Min
#        MyFunctionalTest2Typ
#        MyFunctionalTest2Max
```

```
#
#
# Counters are variables that are incremented during the
# execution of a test.  They are UnsignedIntegers that are
# initialized to zero.
#
Counters {PassCount, FailCount}

#
# Flows can now be presented.  A Flow is an object that
# essentially represents a finite state machine which
# can execute "Flowables", and transition to other flowables based
# on the Result returned from executing a Flowable.  A Flow can also
# call another flow.
#
# A Flow consists of a number of FlowItems and transitions
# between them.  FlowItems have names which are unique in
# the enclosing Flow, execute a "Flowable" object, and then
# transition to another FlowItem in the same enclosing Flow.
#
# Flowable objects include Tests and other Flows.  When
# a Flowable object executes, it returns a numeric Result
# which is used by the FlowItem to transition to another
# FlowItem.  As a result of this, both Tests and Flows
# terminate by returning a numeric Result value.
#
# FlowTest1 implements a finite state machine for the
# Min, Typ and Max flavors of MyFunctionalTest1.  On
# success it tests Test1Min, Test1Typ, Test1Max
# and then returns to its caller with 0 as a successful
# Result.  On failure, it returns 1 as a failing Result.
#
# Assume that the tests MyFunctionalTest1Min, ... all
# return a Result of 0 (Pass), 1 and 2 (for a couple
# of levels of failure).  The Transition Matrix of the
# finite state machine implemented by FlowTest1 is:
# ------------------------------------------------------------
#                       Result 0           Result 1    Result 2
# ------------------------------------------------------------
#    FlowTest1_Min        FlowTest1_Typ        return 1    return 1
#    FlowTest1_Typ        FlowTest1_Max        return 1    return 1
#    FlowTest1_Max        return 0            return 1    return 1
#
# where the IFlowables run by each FlowItem are:
#        FlowItem            IFlowable that is run
#        FlowTest1_Min        MyFunctionalTest1Min
#        FlowTest1_Typ        MyFunctionalTest1Typ
#        FlowTest1_Max        MyFunctionalTest1Max
#
Flow FlowTest1
```

```
{
    FlowItem FlowTest1_Min MyFunctionalTest1Min
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            GoTo FlowTest1_Typ;
        }
        Result 1,2
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            Return 1;
        }
    }

    FlowItem FlowTest1_Typ MyFunctionalTest1Typ
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            GoTo FlowTest1_Max;
        }

        Result 1,2
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            Return 1;
        }
    }

    # Likewise for FlowTest1_Max
    FlowItem FlowTest1_Max MyFunctionalTest1Max
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            Return 0;
        }

        Result 1,2
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            Return 1;
        }
```

```
    }
}


#
# FlowTest2 is similar to FlowTest1.  It implements a
# finite state machine for the Min, Typ and Max flavors
# of MyFunctionalTest2.  On success it tests Test2Min,
# Test2Typ, Test2Max and then returns to its caller with
# 0 as a successful Result.  On failure, it returns 1 as
# a failing Result.
#
# Assume that the tests MyFunctionalTest2Min, ... all
# return a Result of 0 (Pass), 1 and 2 (for a couple
# of levels of failure).  The Transition Matrix of the
# finite state machine implemented by FlowTest2 is:
# ------------------------------------------------------------
#                       Result 0           Result 1    Result 2
# ------------------------------------------------------------
#    FlowTest2_Min       FlowTest2_Typ       return 1    return 1
#    FlowTest2_Typ       FlowTest2_Max       return 1    return 1
#    FlowTest2_Max       return 0           return 1    return 1
#
# Where the IFlowables run by each FlowItem are:
#       FlowItem           IFlowable that is run
#       FlowTest2_Min       MyFunctionalTest2Min
#       FlowTest2_Typ       MyFunctionalTest2Typ
#       FlowTest2_Max       MyFunctionalTest2Max
#
Flow FlowTest2
{
    # ...
}


#
# Now the FlowMain, the main test flow, can be presented.  It
# implements a finite state machine that calls FlowTest1
# and FlowTest2 as below:
# -----------------------------------
#                   Result 0       Result 1
# -----------------------------------
#    FlowMain_1    FlowMain_2    return 1
#    FlowMain_2    return 0       return 1
#
# Where the IFlowables run by each FlowItem are:
#       FlowItem       IFlowable that is run
#       FlowMain_1       FlowTest1
#       FlowMain_2       FlowTest2
Flow FlowMain
```

```
{
    # The first declared flow is the initial flow to be
    # executed.  It goes to FlowMain_2 on success, and
    # returns 1 on failure.
    FlowItem FlowMain_1 FlowTest1
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            GoTo FlowMain_2;
        }

        Result 1
        {
            # Sorry ... FlowTest1 failed
            Property PassFail = "Fail";
            IncrementCounters FailCount;

            # Add to the right soft bin
            SetBin SoftBins."3GHzSBFTFail";

            Return 1;
        }
    }
    FlowItem FlowMain_2 FlowTest2
    {
        Result 0
        {
            # All passed!
            Property PassFail = "Pass";
            IncrementCounters PassCount;

            # Add to the right soft bin
            SetBin SoftBins."3GHzAllPass";

            Return 0;
        }

        Result 1
        {
            # FlowTest1 passed, but FlowTest2 failed
            Property PassFail = "Fail";
            IncrementCounters FailCount;

            # Add to the right soft bin
            SetBin SoftBins."3GHzCacheFail";

            Return 1;
        }
```

```
    }
}

TestFlow = FlowMain;
```

【０３４１】

上記テストプランは、好ましい順番では以下のように構成される。

【０３４２】

１．　第一に、バージョン番号が与えられる。この番号は、コンパイラのバージョンと互換性を確保するために用いられる。

【０３４３】

２．　そして、いくつものインポートが宣言される。これらは、テストプランで用いられる名前を解決するために必要とされる宣言を有するさまざまなファイルである。

【０３４４】

３．　次に、テストプラン名が宣言され、その後に、テストプランのインライン宣言がくる。

【０３４５】

４．　次にPListDefsのセットが宣言される。これらは、名前つきのファイルからのGlobalPListsの名前を挙げる全限定の（full-qualified）名前を含む。またこれらは、パターンリスト変数も含んでいる。パターンリスト変数は、実行時にカスタムフローアブル（custom flowable）において初期化することができる変数である。それらは、実際のパターンリストへの連結テストをランタイムまで遅らせる手段を提供する。

【０３４６】

５．　次に、UserVarsのセットが宣言される。これらは文字列を含む。

【０３４７】

６．　合格したテスト、不合格のテストの数を決定するために、いくつかのカウンタが宣言される。カウンタは、ゼロに初期化され、IncrementCounter命令文でインクリメントされる変数にすぎない。これらは、現在設定されているビンのみがDUTのテストの終了時にインクリメントされるというセマンティクスを持っている先に述べたビンとは異なっている。

【０３４８】

７．　次に、一連のテスト条件が宣言される。これらのそれぞれは、テスト条件グループおよびセレクタを指定する。この例においては、テスト条件グループは、mytestconditionsgroups.tcgからもたらされる。しかしこれらは、テストプランにおけるインラインであることもできる。

【０３４９】

８．　次に、一連のフローアブル（Flowables）あるいはテストが宣言される。それぞれは、パターンリストとテスト条件とを選択する既知のテストFunctionalTestである。したがって例えば、MyFunctionalTest1Maxはテスト条件TC1Maxとパターンリストとを選択する。

【０３５０】

９．　これに続いて、３つのフローFlowTest1、FlowTest2およびFlowMainが宣言される。フローはフローアブル（Flowables）を動作させる。フローアブルは、テスト（MyFunctionalTest1Max等）および他のフロー（FlowTest1およびFlowTest2等）を含む。FlowTest1およびFlowTest2のそれぞれは、それぞれTest1およびTest2の最小、典型、最大のバージョンを通じて動作する。フローFlowMainは先に宣言されたフローFlowTest1を呼び出し、それからFlowTest2を呼び出す。

【０３５１】

１０．最後に、TestFlowイベントがFlowMainフローに割り当てられる。したがって、フローFlowMainは、ユーザがこのプランの実行を選んだときにこのテストプランによって実行されるフローである。

フローのためのC++

【０３５２】

上記ルールを用いて、フロー自体をのぞく要素の大半についてC++インプリメンテーションを行うことができる。

FlowItemのためのC++

【０３５３】

FlowItemを表すためのC++は、以下のインタフェースを有していてもよい。

```
Status setFlowable(IFlowable* pIFlowable);
```

【０３５４】

という動作は、このFlowItemについて実行されるIFlowbleを設定する。

【０３５５】

一旦、FlowItemがこのIFlowableを実行するのに必要とされるコールのセットから戻ると、結果値に応じて、カウンタのリストをインクリメントする必要がある。このために、FlowItemはインクリメントされるべきカウンタのベクトルをもつ必要がある。これは、次のコールによって初期化される。

```
Status setCounterRefs(unsigned int result,
                      CounterRefList counterRefs);
```

【０３５６】

これを呼び出すことがカウンタへの参照のベクトルをFlowItemに設定し、その結果、一旦IFlowableが実行を完了するとカウンタをインクリメントすることができる。例えば、命令文

```
IncrementCountes A, B, C;
```

【０３５７】

は好ましくは、以下のようにして、上記コールを用いる。

```
// Somewhere earlier
CounterRefList counters;
...
// Code for Result clause
//    Result 2, 3 {...}
// of flowObject.
counters.reset();
counters.add(&A);
counters.add(&B);
counters.add(&C);
flowObject.setCounterRefs(2, counters);
flowObject.setCounterRefs(3, counters);
```

【０３５８】

カウンタと名付けられたテンポラリCounterRefListオブジェクトが用いられる。初めにcounters.reset()が呼び出され、カウンタリストを構成するいくつものcounters.add()コールが続く。そして、これがカウンタアドレスのベクトルを構成するために用いられ、結果値２および３に関して更新される。

【０３５９】

そしてFlowItemは、特定の結果で別のFlowItemへと移行する必要があり得る。

```
IncrementCounters A, B, C;
```

【０３６０】
ある結果節が多くの結果値を扱う場合には、当然のことながら、何回かののこのようなコールがなされる必要がある。
【０３６１】
FlowItemは、結果を戻す必要があり得る。これは、

```
Status setTransition(unsigned int result, FlowItem* pFlowItem);
```

【０３６２】
によって行われる。
【０３６３】
例えば、前の例におけるFlowItem FirstFlowItemについて、上記は、「Result」の値「２」と「ReturnResult」の値「１」とで呼び出される。
最後に、FlowItemは、

```
Status execute(unsigned int& result, FlowItem*pNextFlowItem);
```

【０３６４】
を実行するための動作を必要とする。
【０３６５】
この動作は、IFlowableを実行し、その後示されているカウンタを更新し、その後結果を戻す、あるいは次のFlowItemへのポインタを戻す。もしこのポインタがNULLであれば、結果は戻された値となる。
【０３６６】
FlowItemFlowMain_1について生成されるコードは以下の通りである。

```
FlowItem FlowMain_1;
FlowItem FlowMain_2;

CounterRefList counters;

FlowMain_1.setFlowable(FlowTest1);

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

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

// The following call from ITestPlan will set the
// current bin group and bin name.
pTestPlan→setBin("SoftBins", "3GHzSBFTFail");
FlowMain_1.setReturnResult(1, 1);
```

【０３６７】
上で生成されたコードは、IFlowable「FlowTest1」を動作させるためにFlowMain_1をセットアップし、それから各結果に関してカウンタの適切なリストをインクリメントするようにそれをセットアップし、最後に必要なアクションを起こす。結果「０」の場合に必要なアクションは、FlowMain_1への移行であり、結果「１」の場合には戻ることである。

Ｃ２．　TestPlanにおけるカウンタサポート

【０３６８】

カウンタは、ゼロに初期化され、テスト動作中のいろいろな時点でIncrementCounter命令文によってインクリメントすることができる変数である。これらは、テスト終了時にのみインクリメントされるビンとは異なる。さらに、ビンは階層的であるのに対して、カウンタは単純な変数である。したがって、カウンタは、ビンよりも単純で、より限定されたファシリティ（facility）である。

【０３６９】

符号なしの整数である名前つきのカウンタのセットを保持するCounterクラスのメンバを介して、TestPlanにおいてカウンタをサポートすることができる。オブジェクトは、Counters宣言を介してこのクラスで定義される。カウンタは、テスト開始時に自動的にリセットされず、したがって、TestPlanが多くのDUTのテストに関してカウントを集めることを可能にする。カウンタの値をリセット、インクリメント、および問い合わせる方法が提供される。これは、テストを動作させた結果としてのカウントを判断するための、代替物のビンニングを可能にする。

【０３７０】

好ましくは、TestPlanは、メンバ変数m_modifiedCoutnersを含み、これはDUT上でテストを動作させることによって変更されるカウンタのセットである。このセットは、テスト開始時には空のセットに初期化される。IncrementCountersのコールがなされるそれぞれのところで、名前つきのカウンタをm_modifiedCountersメンバに追加するためにコードが生成される。したがって、このメンバは、DUT上でのテストの実行中に変更されたカウンタの全てを集める。

フローオブジェクトのためのC++

【０３７１】

一旦全てのFlowItemが生成されると、フローオブジェクトを、以下に示すように、C++オブジェクトとして生成することができる。

FlowItemを追加するための動作は

```
Status addFlowItem(FlowItem* pFlowItem, bool isInitalFlowItem);
```

【０３７２】

であり、これは示されているFlowItemをフローに追加する。ブール代数は、これがフローの最初のFlowItemであれば真に設定される。

フローを実行する動作は

```
Status executeFlow(unsigned int& result);
```

【０３７３】

である。

【０３７４】

これは、好ましくは、フローが戻るときに、フローを実行した結果を返す。これのアクションは、最初のFlowItemでフローを実行することを開始することである。それは、現在のFlowItemが実行すべき次のFlowItemを返す限り、FlowItemを実行しつづける。現在のFlowItemが結果を返すと、この動作はその結果でもって終了する。

【０３７５】

したがって、フローについて生成されたC++コードは、FlowItemをそのフローに追加するためにaddFlowItem()への何回も繰り返されるコールを有する。executeFlow()の動作は、テストプランにおけるこのフローが実行のために選択されると起こる。

Ｃ３．　テストクラス

【０３７６】

一般的に、プログラムコードの大半は装置テスト用のデータであり、残りはテストプログラムのコードであり、これがテスト方法を実現する。このデータは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。

【０３７７】

上で説明したように、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なければならない。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、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なければならない。

【０３７８】

オープンアーキテクチャテストシステムにおいては、ITest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、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基づいてのみ異なるような、それの別々のインスタンスの「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、機能テスト、ACパラメータのテスト、DCパラメータのテストが好ましくは別々のテストクラスによってインプリメントされる。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られてもよい。

【０３７９】

テストクラスは、そのテストの特定のインスタンスについてのオプションを指定するのに用いられるパラメータを提供することによって、ユーザがクラスの振る舞いを構築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をとってもよい。（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インスタンスを作り出すことが可能である。図５は、どのようにして単一のテストクラス５０４から異なるテストインスタンス５０２が導き出されるかを示す。

【０３８０】

これらのクラスは、コンパイラ４００がテストおよびそのパラメータの記述をテストプランファイルから取り出し、正しいC++コードを生成することが可能であるように設計されなければならない。C++コードはテストプログラムを生成するようにコンパイルされ、リンクされ得る。テストクラスのインスタンスは、装置テストの複雑な実行シーケンスを作り出すためにテストフローを記述しているオブジェクトに加えられてもよい。

Ｃ４．　ITestおよびIFlowableからのずれ

【０３８１】

上述したように、テストクラスはITestから導かれる。上記ルールを用いて、これらをITestインタフェースをインプリメントするC++クラスでインプリメントすることができる。ITestインタフェースについて指定された方法に加えて、これらのクラスは、装置テストの具体的なテストを行うのに必要なテスト特有のインテリジェンスおよびロジックを提供する。またテストクラスは、IFlowableインタフェースもインプリメントする。これの

結果、テストクラスインスタンスは、テストを動作させるためのFlowItemにおいて用いることができる。

カスタム化

【０３８２】

カスタム化メカニズムは、ユーザがC関数を呼び出して、ITestインタフェースおよびIFlowableインタフェースをインプリメントする彼ら自身のクラスを開発することを可能にするように提供される。

イントロスペクション（introspection）能力

【０３８３】

テストクラスのオブジェクトを、その方法およびシグネチャ（signature）に関して質問することができれば、適切なパラメータが生成されたソースコードに含まれるように利用可能であることを検証することができる。このような特徴は、変換段階においてエラーをチェックし、検証するのに非常に有用である。テストエンジニアがパラメータの名前を間違えたり、あるいはこれらのパラメータの引数の個数（あるいはタイプもあり得る）を間違えると、変換段階はそれを捕えて、C++コンパイラからのコンパイルタイムエラーメッセージを待つ代わりに、意味のあるエラーメッセージを変換時に提供する。これはテストエンジニアにとってはより有用である。

【０３８４】

イントロスペクションは、オブジェクトにその内部を見てその属性および方法に関する情報を戻すように依頼する能力を指す。Javaのようないくつかの言語は、この能力を言語の一部として提供している。Visual Basicのような他の言語は、このような要件をそれとともに用いられることが意図されるオブジェクトに課す。C++はこの特徴については何も規定しない。

【０３８５】

またこの方法は、デフォルトパラメータ値を提供すること、ならびにオプション的なパラメータを示すことにもよく役立つ。加えて、この能力が全てのテストクラスのインプリメンテーションの一部として提供されれば、GUIアプリケーションは、エンジニアがこれらのクラスを有効に利用することを助けるように、ダイアログおよび他のユーザインタフェース要素を動的に構築するのにこの情報も用いることができる。

【０３８６】

これらの複雑さは、発明の一実施形態においては、完全なイントロスペクションの代わりに、テストクラスディベロッパがクラスをパラメータ化するのに必要であるものとして指定したテストクラスのパブリックな方法／属性を、単一のテキストベースのソースファイル（テストクラスごとに）において、テストクラスディベロッパが完全に指定することを可能にする方法を提供するメカニズムを通じて相殺される。

【０３８７】

単一のソースが好まれる。あるものは、あるファイル内にテストクラスのパラメータ化されたインタフェースの記述を、別の独立した（ヘッダ）ファイル内にC++インタフェースの記述を持ちたくないであろうし、両方のソースを同期させつづける必要に苦しむであろう。このため、「テキストベースの」記述がテストクラスのためのプリヘッダファイルに埋め込まれ、これが限定されたイントロスペクションのために、ならびにテストクラスのためのC++ヘッダを生成するために、コンパイラによって用いられる。生成されたC++ヘッダファイルは、テストクラスC++コードを最終的にコンパイルするために用いられるファイルである。

プリヘッダ

【０３８８】

C++におけるヘッダの使用がよく知られている。しかし、C++は解析することも読むこと

も難しいので、発明の一実施形態は、C++の出力をコンパイラが生成することを可能にするシンタックス（syntax）を規定する。このC++出力をテストクラスディベロッパがヘッダとして用いることができる。この実施形態によると、テストクラスディベロッパはプリヘッダを書き、これがコンパイラ４００によってヘッダファイルとして出力され、対応するテストクラスあるいは他のテスト対象を目に見えるようにする。

【０３８９】

次の例は、本発明の好ましい実施形態による、テストクラスのためのプリヘッダファイルの概念を示すものである。テストFuncTest1とともにソースファイルからの以下の抜粋を考える。

```
...
TestCondition TC1
{
    TestConditionGroup = TCG1;    # Previously defined TCG for Levels
    Selector = min;
}

TestCondition TC2
{
    TestConditionGroup = TCG2;    # Previously defined TCG for Timing
    Selector = min;
}
...
Test FunctionalTest FuncTest1
{
    PListParam = patList1;            # Previously defined pattern list
    TestConditionParam = TC1;
    TestConditionParam = TC2;
}
```

【０３９０】

コンパイラは、上記FuncTest1の宣言が正当なものであるかを判断するために、FunctionalTestが何を引き起こすかを知る必要がある。コンパイラへのFunctionalTestの知識における構築ではなく、FunctionalTestが何を引き起こすかの定義がPre-Headerにおいて記述される。

【０３９１】

FunctionalTestがTest1およびTest2を有し、PListであるメンバならびにTestConditionのアレイを有するするC++クラスであると仮定する。コンパイラは、FuncTest1の上記宣言が正当であると認識するために、FunctionalTestのメンバのタイプについて知る必要がある。

【０３９２】

さらに、FuncTest1についてのC++オブジェクト宣言を生成するために、クラスFunctionalTestについてのC++ヘッダを構築する必要がある。これは、コンパイラがFunctionalTestクラスのベースクラス、そのメンバの名前および他のこのような情報についても知ることを要求する。

【０３９３】

発明の一実施形態のプリヘッダサブ言語は、コンパイラに、それが宣言の正当性を認識するため、およびC++ヘッダおよび宣言に対応するオブジェクト宣言を生成するための両方に必要である情報を提供する。

【０３９４】

FunctionalTestは、（パラメータ化に関する限り）単純なタイプであり、したがって、

パラメータ化について極めて単純な記述を用いるであろうということに留意されたい。したがって、上記パラメータ化をサポートするプリヘッダFunctionalTest.phを以下のように書くことができるであろう（ベーステストクラスTest1およびTest2についてプリヘッダが利用可能であると仮定する）。

```
Version 1.0;
#
#  Parameterization specification pre-header for FunctionalTest
#
Import Test1.ph;                    # For base class Test1
Import Test2.ph;                    # For base class Test2
TestClass = FunctionalTest;    # The name of this test class
PublicBases = Test1, Test2;    # List of public base classes
# The parameters list or "parameter block":
Parameters
{
    # The following declaration specifies that a FunctionalTest has
    #     - a parameter of  type PList
    #    - [represented by C++ type Tester::PatternTree]
    #    - stored in a member named m_pPatList
    #    - a function to set it named setPatternTree.
    #    - a parameter description for the GUI to use as a tool tip
    PList PListParam
    {
        Cardinality = 1;
        Attribute = m_pPatList;
        SetFunction = setPatternTree;
        Description = "The PList parameter for a FunctionalTest";
    }
    #
    # The following declaration specifies that a FunctionalTest has
    #    - 1 or more parameters of  type TestCondition
    #    - [represented by C++ type Tester::TestCondition]
    #    - stored in a member named m_testCondnsArray
    #    - a function to set it named addTestCondition.
    #    - a parameter description for the GUI to use as a tool tip
    # The [implement] clause causes the translation phase of  to
    # generate a default implementation of this function.
    #
    TestCondition TestConditionParam
    {
        Cardinality = 1-n;
        Attribute = m_testCondnsArray;
        SetFunction = addTestCondition [Implement];
        Description = "The TestCondition parameter for a FunctionalTest";
    }
}
#
# The section below is part of the Pre-Header which is an escape
# into C++ code.  This will be referred to as a "template block."
```

```
#
# Everything in this section will be reproduced verbatim in the
# generated header file, except for "$Class", "$Inc",
# "$ParamAryTypes", "$ParamAttrs", "$ParamFns" and "$ParamImpls".
#
# Note that no comments beginning with the '#' character are supported
# within the following section.
#
CPlusPlusBegin
$Inc
namespace
{
class $Class
{
// Array types for parameters storage:
$ParamAryTypes
public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
    $ParamFns
    ...
private:
    double m_someVar;
    $ParamAttrs
    ...
};
...
$ParamImpls
} // End namespace
CPlusPlusEnd
```

パラメータ化されたテストクラスのためのC++

【０３９５】

コンパイラがプリヘッダファイルを処理する際に、コンパイラは、$Inc、$Class、$ParamAryTypesおよび他のもののようなコンパイラ変数の値を構築する。このことによって、コンパイラはその後に、上記C++コードを逐語的に生成し、指示された場所でコンパイラ変数$Inc、$Class等の値において展開することによって以下のC++ヘッダを作り出すことを可能にする。FunctionalTest.phに関してコンパイラは、FunctionalTestクラスのために以下のC++ヘッダファイルFunctionalTest.hを作り出す。

```
1 #line 7 " ./FunctionalTest.ph"
 #include <ITest.h>
 #line 5 " ./FunctionalTest.ph"
 #include <Test1.h>
 #line 6 " ./FunctionalTest.ph"
 #include <Test2.h>
 #line 55 " ./FunctionalTest.ph"
 #include <vector>
 #line 55 " ./FunctionalTest.ph"
```

```
#include <Levels.h>
#line 55 " ./FunctionalTest.ph"
#include <TestCondnGrp.h>
...
#line 56 " ./FunctionalTest.ph"
namespace
{
#line 7 " ./FunctionalTest.ph"
class FunctionalTest :    public ITest,
#line 8 " ./FunctionalTest.ph"
                          public Test1,
#line 8 " ./FunctionalTest.ph"
                          public Test2
#line 59 " ./FunctionalTest.ph"
{
// Array types for parameters storage:
#line 61 " ./FunctionalTest.ph"
public:
#line 37 " ./FunctionalTest.ph"
    typedef std::vector<Tester::TestCondition *> TestConditionPtrsAry_t;
#line 62 " ./FunctionalTest.ph"
public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
public:
#line 7 " ./FunctionalTest.ph"
    void setName(OFCString &name); # Automatic for all tests
#line 22 " ./FunctionalTest.ph"
    void setPatternTree(PatternTree *);
#line 23 " ./FunctionalTest.ph"
    String getPListParamDescription() const;
#line 39 " ./FunctionalTest.ph"
    void addTestCondition(TestCondition *);
#line 40 " ./FunctionalTest.ph"
    void getTestConditionParamDescription() const;
#line 67 " ./FunctionalTest.ph"
    ...
private:
    double m_someVar;
#line 70 " ./FunctionalTest.ph"
private:
#line 7 " ./FunctionalTest.ph"
    OFCString m_name; # Automatic for all tests
#line 21 " ./FunctionalTest.ph"
    Tester::PatternTree *m_pPatList;
#line 38 " ./FunctionalTest.ph"
    TestConditionPtrsAry_t m_testCondnsArray;
#line 71 " ./FunctionalTest.ph"
    ...
```

```
};
...
#line 7 " ./FunctionalTest.ph"
inline void
#line 7 " ./FunctionalTest.ph"
FunctionalTest::setName(OFCString &name)
#line 74 " ./FunctionalTest.h"
{
    m_name = name;
    return;
}
#line 39 " ./FunctionalTest.ph"
inline void
#line 39 " ./FunctionalTest.ph"
FunctionalTest::addTestCondition(TestCondition *arg)
#line 74 " ./FunctionalTest.ph"
{
    m_testCondnsArray.push_back(arg);
    return;
}
#line 23 " ./FunctionalTest.ph"
inline void
Tester::String FunctionalTest::getPListParamDescription()
{
    return "The PList parameter for a FunctionalTest";
}
#line 40 " ./FunctionalTest.ph"
inline void
Tester::String FunctionalTest::getTestConditionParamDescription()
{
    return "The TestCondition parameter for a FunctionalTest";
}
#line 75 " ./FunctionalTest.ph"
} // End namespace
```

【０３９６】

先に述べたように、このプリヘッダは、コンパイラがFunctionalTest宣言の妥当性をチェックすること、それのためのコードを生成すること、およびそれによって必要とされるであろうC++ヘッダを生成することを可能にする。

【０３９７】

一例として、先に与えられたFunctionalTest宣言を考える。便宜のために以下に再掲する。

```
Test FunctionalTest FuncTest1
{
    PListParam = patList1;            # Previously defined pattern list
    TestConditionParam = TC1;
    TestConditionParam = TC2;
}
```

【０３９８】

これのためにコンパイラによって生成されるC++ヘッダが上に示される。コンパイラは

、上記FunctionalTestコンストラクトのために以下のコードを生成する。

```
FunctionalTest FuncTest1;
FuncTest1.setName(" FuncTest1 " );
FuncTest1.setPatternTree(&patList1);
FuncTest1.addTestCondition(&TC1);
FuncTest1.addTestCondition(&TC2);
```

【０３９９】

記述関数のために生成された名前にも注目されたい。Xxxと名付けられたそれぞれのパラメータは、メンバ関数

```
Status getXxxDescription() const;
```

【０４００】

に関連付けられており、GUIが用いることができるツールチップ（tool tip）についての記述を有する文字列を返す。

他のプリヘッダの特徴

【０４０１】

プリヘッダは、追加のタイプとして、他のユーザ定義の列挙をサポートする。これにより、GUIが、特定のパラメータの値を設定するために用いることのできる可能な選択肢のドロップダウンリストを提供することが可能である。さらに、プリヘッダは、テーブルと考えることができる数多くのパラメータを関連付けるための特徴を提供する。例えば、名前用の文字列のアレイの関連しているセットとしての「プロパティ」のアレイ、ならびに値のための整数のアレイをインプリメントすることは便利であるかもしれない。この特徴をインプリメントする一つの簡単な方法は、カスタムタイプ（後述する）のアレイを用いることである。しかしながら、それは、ユーザに対して、使うためのカスタムタイププリヘッダを書くことを要求する。これらの特徴は両方とも、以下の例に示されている。

```
# ----------------------------------------------------------
# File FooBarTest.ph
#
#  Parameterization specification pre-header for
# custom test class FoobarTest
# ----------------------------------------------------------


Version 1.0;

Import Test1.ph;                    # For base class Test1
TestClass = FoobarTest;    # The name of this test class
PublicBases = Test1;    # List of public base classes

# The parameters list:
Parameters
{
    # An enumerated type
    Enum WishyWashy = Yes, Perhaps, Possibly, Maybe, MaybeNot, No;

    # Define a WishyWashy parameter.
    WishyWashy WW
```

```
{
    Cardinality = 1;
    Attribute = m_ww;
    SetFunction = setWw;
    Description = "The WW parameter for a Foobar Test";
}

# This class has an array of name-number pairs that is
# interpreted in the class.
ParamGroup
{
    Cardinality = 0-n;

    # The Name field in this array is:
    #    - of  type String
    #    - [represented by C++ type Tester::String]
    #    - stored in a member named m_NameArray
    #    - a function to set it named addName.
    #    - a parameter description for the GUI to use as a tool tip
    String Name
    {
        Attribute = m_NameArray;
        SetFunction = addName;
        Description = "A Name with a Value";
    }

    # The Number field in this array is:
    #    - of  type Integer
    #    - [represented by C++ type int]
    #    - stored in a member named m_NumberArray
    #    - a function to set it named addNumber.
    #    - a parameter description for the GUI to use as a tool tip
    Integer Number
    {
        Attribute = m_NumberArray;
        SetFunction = addNumber;
        Description = "The value of the Name";
    }
}

# The following declaration specifies that a FunctionalTest has
#     - a parameter of  type PList
#    - [represented by C++ type Tester::PatternTree]
#    - stored in a member named m_pPatList
#    - a function to set it named setPatternTree.
#    - a parameter description for the GUI to use as a tool tip
PList PListParam
{
    Cardinality = 1;
    Attribute = m_pPatList;
```

```
        SetFunction = setPatternTree;
        Description = "The PList parameter for a FunctionalTest";
    }

    #
    # The following declaration specifies that a FunctionalTest has
    #    - 1 or more parameters of  type TestCondition
    #    - [represented by C++ type Tester::TestCondition]
    #    - stored in a member named m_testCondnsArray
    #    - a function to set it named addTestCondition.
    # The [implement] clause causes the translation phase of to
    # generate a default implementation of this function.
    #
    TestCondition TestConditionParam
    {
        Cardinality = 1-n;
        Attribute = m_testCondnsArray;
        SetFunction = addTestCondition [Implement];
        Description = "The TestCondition parameter for a FunctionalTest";
    }
}

   CPlusPlusBegin
   $Inc
   namespace
   {
   class $Class
   {
   // Array types for parameters storage:
   $ParamAryTypes
   public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
    $ParamFns
       // ...
   private:
    double m_someVar;
    $ParamAttrs
    // ...
   };
   // ...
   $ParamImpls
   } // End namespace
   CPlusPlusEnd
```

【０４０２】

カスタムタイプの名前と数のペアは宣言された可能性があり、そのカスタムタイプの単一のパラメータは、パラメータの上記ParamGroupと同じ効果を持つように用いられた可能性があることに留意しなければならない。上で提示した手法は、カスタムタイプを宣言する必要を避ける便利さである。

Ｃ５．　カスタム関数宣言

【０４０３】

これは、フローの移行が起こるときにユーザがカスタム関数を呼び出すことを可能にする。カスタム関数は、プリヘッダを通して以下のように宣言される。

```
# ------------------------------------------------------
# File MyFunctions.ph
#
#  Parameterization specification pre-header for MyFunctions
# ------------------------------------------------------

Version 1.0;

Functions = MyFunctions;     # The name of this group of functions

# Declare the following C++ function in the
# MyFunctions namespace to determine the minimum
# of two values.
#    // Return the minimum of x, y
#    double MyRoutines::Min
#                (ITestPlan* pITestPlan,int& x, int& y);
Integer Min(Integer x, Integer y);

# Declare the following C++ function in the
# UserRoutines namespace to return the average of
# an array.
#    // Return the average of the array
#    double MyRoutines::Avg
#                (ITestPlan* pITestPlan, double* a, const int a_size);
# The C++ function will be called with a and a'Length
Double Avg(Double a[]);

# Declare the following C++ function in the
# UserRoutines namespace to print the dut id
# and a message
#    // Return the average of the array
#    double MyRoutines::Print
#                (ITestPlan* pITestPlan, String* msg, unsigned int& dutId);
# The C++ function will be called with a and a'Length
Void Print(String msg, UnsignedInteger dutId);
```

【０４０４】

典型的には、C++セクションは、コンパイラが上記宣言を標準的なやり方で拡張する際に、これらの宣言を与えられる必要がある。当然のことながら、ユーザはこれらの関数のC++インプリメンテーションを担う。上記関数の全てがおそらく暗黙のうちの第一のパラメータとしてITestPlanポインタをとるということに留意されたい。このポインタは、関数の書き手にTestPlanにおけるsteteSへのアクセスを与える。例えば、関数の書き手は、現在のフロー、そのフローにおける現在のFlowItem、現在の結果節、UserVarsの値、および他のこのような情報にアクセスするのにITestPlanインタフェースを用いることができ

る。あるテスタ定義の関数はファイルFunctions.phにおいて使用されるために利用可能である。

```
Version 1.2.3;

#
# File Functions.ph
#
Functions = Functions;     # The name of this group of functions

# Declare the following C++ function in the
# Functions namespace

# Returns the ID of the current DUT being tested by the
# caller.
UnsignedInteger GetDUTID();
```

カスタム関数宣言のためのC++

【０４０５】

上記MyFunctionsについてコンパイラによって生成されるC++コードは、MyFunctions名前空間においていくつかの関数を単に宣言するためである。

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

【０４０６】

これらの関数は、フローから呼び出し可能である。

Ｃ６．　カスタムフローアブル

【０４０７】

プリヘッダを用いてC++IFlowableインタフェースをインプリメントするプリヘッダを生成することも可能である。これは、FlowItemで動作させることができるカスタムフローアブルをユーザが定義することを可能にする。

```
# ----------------------------------------------------------
# File MyFlowable.ph
#
# Parameterization specification pre-header for MyFlowable
# ----------------------------------------------------------

Version 1.2.4;

FlowableClass = MyFlowable;        # The name of this custom class

# The parameters list:
Parameters
{
```

```
    # The following declaration specifies that a MyFlowable has
    #     - 1 optional parameter Int1 of  type Integer
    #    - [represented by C++ type int]
    #    - stored in a member named m_int1Val
    #    - a function to set it named setInt1Val.
    Integer Int1
    {
        Cardinality = 0-1;
        Attribute = m_int1Val;
        SetFunction = setInt1Val;
    }

    # The following declaration specifies that a MyFlowable has
    #     - 1 mandatory parameter Int2 of  type Integer
    #    - [represented by C++ type int]
    #    - stored in a member named m_int2Val
    #    - a function to set it named setInt2Val.
    Integer Int2
    {
        Cardinality = 1;
        Attribute = m_int2Val;
        SetFunction = setInt2Val;
    }
    # The following declaration specifies that a MyFlowable has
    #    - one or more parameters of type String
    #    - [represented by C++ type Tester::String]
    #    - stored in a member named m_stringArrVal
    #    - a function to set it named addStringVal.
    String StringItem
    {
        Cardinality = 1-n;
        Attribute = m_stringArrVal;
        SetFunction = addStringVal;
    }

    # The following declaration specifies that a MyFlowable has
    #    - A single PList parameter
    #    - [represented by the C++ type Tester::PList]
    #    - stored in a member named m_plist
    #    - a function to set it named setPListParam
    PList PListParam
    {
        Cardinality = 1;
        Attribute = m_plist;
        SetFunction = setPListParam;
    }
}

#
# The section below is part of the Pre-Header which is an escape
```

```
# into C++ code.
#
# Everything in this section will be reproduced verbatim in the
# generated header file, except for "$Class", "$Inc",
# "$ParamAryTypes", "$ParamAttrs", "$ParamFns" and "$ParamImpls".
#
# Note that no comments beginning with the '#' character are supported
# within the following section.
#
CPlusPlusBegin
$Inc
namespace
{
class $Class
{
// Array types for parameters storage:
$ParamAryTypes
public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
    $ParamFns
    // ...
private:
    double m_someVar;
    $ParamAttrs
    // ...
};
// ...
$ParamImpls
} // End namespace
CPlusPlusEnd
```

【０４０８】

IFlowableインタフェースをインプリメントするいくつかのクラスがある。これらは、

１．　テストプランを現在のテスタコンフィギュレーション内で実行することができるかどうかをチェックする、プログラムローディングのためのフロー、

２．　具体的なパターンおよびパターンリストをロードする、パターンローディングのためのフロー、

３．　ハードウェアおよびソフトウェアを既知の状態にし、グローバル変数をロードし、他の初期化および検証機能を行う、初期化のためのフロー、ならびに

４．　テストフローに一般的に有用である他のもの

【０４０９】

を含む。

Ｃ７．　カスタムタイプ

【０４１０】

先のテストクラスのパラメータ化についての議論は、テストクラスのパラメータが既知のタイプのもの、すなわちPListsおよびTestConditionsエレメンタリタイプおよびであることを可能にするのみであった。ユーザの柔軟性のために、タイプ拡張性を提供することは重要であり、それによってタイプ（コンピュータにとって未知の先天的なものである）

を作り出し、用いることができる。カスタムタイプ（CT）はCustom Typesにおいて定義される。これらを、C言語のストラクト（プレインオールドデータタイプ、つまりPODとも呼ばれ、C++における同名のものとは全く異なる）に対応するタイプを定義するために、ならびに、ファンクションシグネチャについてのC言語typedefsに対応するタイプについて用いることができる。ユーザタイプを有する別個のファイルは拡張子「.ctyp」を有する。本発明の好ましい実施形態による、ユーザタイプ宣言の例がこれである。

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

Version 1.0.0;

CustomTypes
{
    # A structured Plain-Old-Data type
    Pod Foo
    {
        String        S1;        # String is a standard  type
        Integer    I1;        # ... and so is Integer
        String        S2;
    }

    # Another structured type using Foo
    Pod Bar
    {
        Foo            Foo1;
        String        S1;
        Foo            Foo2;
    }

    #
    # A pointer to a function.
    #        Return type:    Integer
    #        Parameters:    Integer, Integer
    #
    Routine BinaryOp(Integer, Integer) Returns Integer;

    #
    # Another pointer to a function.
    #        Return type:    Void
    #        Parameter:    Integer
    #
    Routine UnaryOp(Integer) Returns Void;

    #
    # A pointer to a function that takes
    # no parameters and does not return a value.
    #
    Routine NullaryOp() Returns Void;
```

```
}
```

カスタムタイプのためのC++

【０４１１】

上で提示したCustomTypes宣言は、コンパイラによって以下のC++コードに変換される。

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

【０４１２】

これらのタイプのオブジェクトは、次に示すようにパラメータとしてのテストクラスへと渡され得る。

テストクラスパラメータとしてのカスタムタイプの使用

【０４１３】

ユーザがテストの拡張を有する場合を考える。このテストの拡張は、パターンリストおよびテスト条件に加えて、他のクラスオブジェクトならびに、カスタムタイプを含むファイル（すなわち.ctypファイル）内で定義される任意の（すなわちユーザ定義の）オブジェクトで初期化される必要がある。例えば、ユーザがファイルMyTestCTs.ctypで定義されるCTを用いることを望んでいるとする。

```
# File MyTesetCTs.ctyp
Version 1.0;

CustomTypes
{
    Pod Foo
    {
        String name;
        PList    patternList;
    }

    Pod Bar
    {
        Foo        someFoo;
```

```
        Double    dVal;
    }

    Routine BinaryOp(Integer, Integer) return Integer;
}
```

【０４１４】

ユーザが上記タイプを用いるためにする必要があるのは、ユーザのテストクラスプリヘッダに上記ファイルをインポートすることだけである。コンパイラはそのように定義されたCTを解釈するので、FooおよびBarの定義は、それがテストクラスプリヘッダを処理しているときに、それに役立つ。また、コンパイラは、それぞれ上記タイプFooおよびタイプBarに対応する２つのC言語のストラクト、ストラクトFooおよびストラクトBarを定義する。それらの定義は、ファイルmyTestCts.hに置かれる。myTestCts.csttへのImport命令文は、ファイルmyTestCts.hを生成されたテストクラスC++ヘッダに#include-dさせる。以下の例は、このプロセスを示すものである。まず、テストプランにおけるテストの定義を考える（パターンリストおよびテスト条件の定義は明瞭にするために省略されている）。

```
...
Import MyFunctions.ph;
Import MyCustomTypes.ctyp;
...
# The CustomVars block defines variables of the Custom
# types defined earlier.
CustomVars
{
    ...
    Bar bar1 =
    {
        { "This is a Foo", somePatList },        # someFoo
        3.14159                                  # dVal
    }
    #
    # A function object that is a binary operator.
    # The name on the right hand side of the assignment
    # is a routine declared in MyFunctions, for which,
    # of course, the user has to provide an implementation.
    #
    BinaryOp bop1 = MyFunctions.Min;
}
...
Test MyFancyTest MyTest1
{
    ...
    BarParam = bar1;
    BinaryOpParam = bop1;
}
...
```

【０４１５】

上記例では、CustomVarsブロックがテストプランに含まれる。カスタム化された変数を有する別個のファイルは、拡張子「.cvar」を有する。ユーザは、上記パラメータ化をサポートするMyFancyTestのプリヘッダを以下のように書く（パターンリストおよびテスト

条件についてのパラメータ化の宣言は明瞭にするために省略されている）。

```
# ------------------------------------------------------
# File MyFancyTest.ph
#
#  Parameterization specification pre-header for MyFancyTest
# ------------------------------------------------------

Version 1.0.2;

Import MyCustomTypes.ctyp;        # For CTs used in MyFancyTest
Import FunctionalTest.ph;        # For base class FunctionalTest
TestClass = MyFancyTest;        # The name of this test class
PublicBases = FunctionalTest;    # List of public base classes

# The parameters list:
Parameters
{
    # The following declaration specifies that a MyFancyTest has
    #     - an optional array of parameters of custom  type Bar
    #    - [represented by C++ type CustomTypes::Bar]
    #    - stored in a member named m_barsArray
    #    - a function to set it named addBarParam.
    # An implementation will be generated for addBarParam.
    Bar BarParam
    {
        Cardinality = 0-n;
        Attribute = m_barsArray;
        SetFunction = addBarParam [Implement];
    }

    # The following declaration specifies that a MyFancyTest has
    #     - an optional parameter of custom  type BinaryOp
    #    - [represented by C++ type CustomTypes::BinaryOp]
    #    - stored in a member named m_binaryOp
    #    - a function to set it named setBinaryOpParam.
    # An implementation will be generated for setBinaryOpParam.
    BinaryOp BinaryOpParam
    {
        Cardinality = 0-1;
        Attribute = m_binaryOp;
        SetFunction = setBinaryOpParam [Implement];
    }
}


CPlusPlusBegin

$Inc
namespace
```

```
{

class $Class
{
$ParamAryTypes
public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
    $ParamFns
    // ...
private:
    double m_someVar;
    $ParamAttrs
    // ...
};

// ...
$ParamImpls
} // End namespace
CPlusPlusEnd
```

カスタムタイプを用いるカスタムテストクラスのためのC++

【０４１６】

最後に、一旦コンパイラがこのプリヘッダファイルを処理すると、コンパイラは、MyFancyTestクラスのための以下のC++ヘッダファイルMyFancyTest.hを作成する。

```
#include <MyCustomTypes.h>
#include <ITest.h>
#include <FunctionalTest.h>
...
namespace
{
class MyFancyTest : public ITest,
                    public FunctionalTest
{
public:
    typedef std::vector<CustomTypes::Bar *> BarAry_t;
public:
    virtual void preExec();
    virtual void exec();
    virtual void postExec();
public:
    void setName(OFCString &name); # Automatic for all tests
    void setPatternTree(PatternTree *);
    void addTestCondition(TestCondition *);
    void addBarParam(CustomTypes::Bar *);
    void setBinaryOpParam(CustomTypes::BinaryOp *);
    ...
private:
```

```
    double m_someVar;
private:
    OFCString m_name; # Automatic for all tests
    PatternTree *m_pPatList;
    TestConditionPtrsAry_t m_testCondnsArray;
    BarAry_t m_barsArray;
    BinaryOp *m_binaryOp;
    ...
}; // End class MyFancyTest
...
inline void
MyFancyTest::addBarParam(CustomTypes::Bar *arg)
{
    m_barsArray.push_back(arg);
    return;
}
inline void
MyFancyTest::setBinaryOpParam(CustomTypes::BinaryOp *arg)
{
    m_binaryOp = arg;
    return;
}
} // End namespace
```

Ｃ８．　パラメータ化

【０４１７】

上でわかるように、テストクラス、カスタムフローアブルクラス、あるいはカスタム関数定義のためのプリヘッダは、パラメータ化された仕様セクションを通じて、クラス／関数に限定されたイントロスペクションを提供する。コンパイラは、クラス／関数のためのパラメータ化されたインタフェースを生成する（ならびにクラス／関数ヘッダ自体を生成する）ために、このセクションを用いる。テストクラスおよびフローアブルクラスについては、コンパイラは、テストプランコードにおいてコールをその後に生成してそのクラスのインスタンスを初期化するためにも、このセクションを用いる。プリヘッダおよび対応する宣言に関連する以下の点に留意しなければならない。

【０４１８】

１．　どのテストまたはカスタムフローアブルクラス定義も、好ましくは、プリヘッダにおいて定義されれる。プリヘッダにおけるパラメータブロックは、好ましくは、このようなクラスのパラメータリストを指定することができる唯一の場所である。（したがって、必然の結果として、パターンリストおよびテスト条件の指定のような、テストのための「標準的な」パラメータも、プリヘッダのパラメータブロックに含まれる必要がある。これにより全てのパラメータ、標準およびＣＴが一様に扱われることが可能になる。）

【０４１９】

２．　テストまたはフローアブルクラスについてのプリヘッダにおいて、オプショナルではない（すなわち、ゼロでない濃度（cardinality）を有する）と定義されたパラメータの全ては、そのクラスのインスタンスについてのテストブロックまたはフローアブルブロック宣言において初期化されなければならない。

【０４２０】

３．　テスト／フローアブルブロックにおけるパラメータの初期化に用いられるオブジェクトは、先に定義されていなければならない。

【０４２１】

４．　置換インジケータ$Class、$Inc、$ParamAryTypes、$ParamFns、$ParamAttrsおよび$ParamImplsは、対応する生成されたコードが生成されたクラスヘッダファイル内で挿入されることをユーザが意図する、プリヘッダのユーザコードセクション内の正確な位置に現れなければならない。これらは、具体的なコードがそれぞれについて生成されるので、正確に一度現れる。

【０４２２】

５．　プリヘッダのパラメータブロックにおけるパラメータ指定の名前（上記例におけるPListParam、TestConditionParamあるいはBarParam）は、そのクラスのインスタンスの宣言において用いられるべきパラメータの名前である。

【０４２３】

６．　以下は、パラメータ指定において用いられる記述子のセマンティクスである。

【０４２４】

ａ．Cardinality：これはサポートされるこのタイプのパラメータの数を示す。以下は、一実施形態において可能である数である。

【０４２５】

ｉ．　１：このパラメータは必須であり、正確に一度指定されなければならない。このパラメータはパラメータのタイプのオブジェクトへのポインタとして保持される。

【０４２６】

ii．　０～１：このパラメータはオプション的であり、指定されるのなら、一度だけ指定されなければならない。このパラメータはパラメータのタイプのオブジェクトへのポインタとして保持される。

【０４２７】

iii．　１～ｎ：このパラメータは必須である。また、これに対して複数の値を指定することができる。値は指定順で記憶される。

【０４２８】

iv．　０～ｎ：このパラメータはオプション的である。これに対して複数の値を指定することができる。値は、指定順で記憶される。

【０４２９】

上の（）および（）について、全ての指定された値は、STLベクトル<>で記憶され、パラメータのタイプへのポインタ上でテンプレート化されることに留意されたい。このベクトルのタイプは、$ParamAryTypesによって示された時点で定義、挿入される。これらのタイプ定義へのアクセスレベルは常にパブリックである。

【０４３０】

ｂ．Attribute：このタイプのパラメータ値について記憶として用いられるC++変数の名前である。この名前は、C++クラスのプライベートデータメンバとして、逐語的に再現され、C++識別子の要件に合致していな　ければならない。この属性のタイプは

ｉ．　もし単一の値が許容されるのであれば、パラメータのタイプへのポインタ、

ii．　もし複数の値が許容されるのであれば、パラメータのタイプへのポインタ上でテンプレート化された、STLベクトル<>（上の（）を参照）

【０４３１】

であることに留意されたい。

【０４３２】

なお、Attributeはテストプランによって作成され、占められるオブジェクトへの参照を保持し、これらのオブジェクトを所有しない。オブジェクトの寿命は常にテストプラン自体によって管理される。

【０４３３】

ｃ．SetFunction：このパラメータについての値を設定するのに用いる関数の名前である。以下の点に留意しなければならない。

【０４３４】

ｉ．　名前は逐語的に再現され、したがって、C++言語の要件に合致していなければなら

ない。
【０４３５】
ii.　関数へのアクセスレベルは常にパブリックである。
【０４３６】
iii.　リターンタイプは常に無効（void）である。
【０４３７】
iv.　関数は常に一つだけの、ポインタ対ポインタタイプの引数をとる。
【０４３８】
値は常に一つで設定されることに留意されたい。すなわち、複数個の値の指定を許すパラメータについて、テストプランにおける生成されたコードは、この関数を繰り返し呼び出し、一旦どの値も指定されると、それらのそれぞれは、STLベクトルに追加される（上述したように）。
【０４３９】
関数名に続くオプションのキーワード「[implement]」は、この関数についての自明な（trivial）インプリメンテーションが、（$ParamImplsによって示される時点で挿入される）クラスヘッダにおけるインライン方法として利用可能とされることを示している。そうでなければ、ユーザはその関数のインプリメンテーションを提供する責任がある。
【０４４０】
ｄ．Description：GUIツールによって、このパラメータのランタイム変更（modification）中に助けを与えるために用いられるツールチップである、リテラルな文字列である。Xxxと名付けられたパラメータについてカスタムクラスにおいて生成されたC++メンバ関数は、

```
String getXxxDescription () const;
```

【０４４１】
となり、この関数は指定された文字列を返す。

カスタム化を有するテストプラン例
【０４４２】
いくつかのカスタム化で装飾されたテストプラン例を以下に示す。

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

Version 0.1;

#
# Imports as before ...

# The following import is implicit, but can be explicitly
# provided.
Import FunctionalTest.ph;

# Import for MyFlowables, MyFunctions and Functions
Import MyFlowables.ph;
Import MyFunctions.ph;
Import Functions.ph;

#------------------------------------------------------------
# Start of the test plan
```

```
#-------------------------------------------------------
TestPlan Sample;

# This block defines Pattern Lists file-qualified names and
# Pattern List variables that are used in Test declarations.
# The file-qualified names refer to pattern lists in the named
# files.  The variables refer to String variables which will
# hold the pattern list names at run time.  User defined Flowable
# objects could set the values of these variables through an
# API.
PListDefs
{
    # File qualified pattern list names
    pl1A.plist:pat1Alist,
    pl2A.plist:pat2AList,

    # Pattern list variables
    plistXxx,
    plistYyy,
    plistZzz
}


# SocketDef, UserVars declaration as before ...

# Declarations of TestConditions TC1Min, TC1Typ, TC1Max,
# TC2Min, TC2Typ, TC2Max as before ...

#
# Declare a FunctionalTest.  "FunctionalTest" refers to a C++
# test class that runs the test, and returns a 0, 1 or 2 as
# a Result.  The Test Condition Group TCG1 is selected with
# the "min" selector by referring to the TC1Min TestCondition.
#
# Note that compiler can compile this because of the imported
# FunctionalTest.ph file.
#
Test FunctionalTest MyFunctionalTest1Min
{
    PListParam = pat1AList;
    TestConditionParam = TC1Min;
}

#
# Additional FunctionalTest declarations for the following, as before
#        MyFunctionalTest1Typ
#        MyFunctionalTest1Max
#        MyFunctionalTest2Min
#        MyFunctionalTest2Typ
#        MyFunctionalTest2Max
```

```
#

# Here is a declaration of MyFlowable.  It uses a PatternList variable
# plistXxx which is initialized by the flowable prior to use here.
#
# Compiler can compile this because of the imported MyFlowables.ph file:
Flowable MyFlowable MyFlowable1
{
    Int1 = 10;
    Int2 = 20;
    StringItem = "Hello World";
    PListParam = plistXxx;
}

# Counters for PassCount and FailCount as before ...

# Flows as before.  Flows FlowTest1 and FlowTest2 are
# unchanged from the previous example.
Flow FlowTest1
{
    # ...
}

Flow FlowTest2
{
    # ...
}
#
# Now FlowMain, a main flow, can be presented.  It
# implements a finite state machine that calls FlowTest1
# and FlowTest2 as below:
# -----------------------------------
#                    Result 0        Result 1
# -----------------------------------
#    FlowMain_1    FlowMain_2    return 1
#    FlowMain_2    FlowMain_3    return 1
#    FlowMain_3    FlowMain_4    return 1
#    FlowMain_4    FlowMain_5    return 1
#    FlowMain_5    return 0        return 1
#
# Where the IFlowables run by each FlowItem are:
# ----------------------------------------
#        FlowItem            IFlowable that is run
# ----------------------------------------
#        FlowMain_1        MyFlowable1
#        FlowMain_2        DatalogStartFlow
#        FlowMain_3        FlowTest1
#        FlowMain_4        FlowTest2
#        FlowMain_5        DatalogStopFlow
#
```

```
Flow FlowMain
{
    #
    # The first declared flow is the initial flow to be
    # executed.  It goes to FlowMain_InitializationFlow
    # on success, and returns 1 on failure.
    #
    FlowItem FlowMain_1 MyFlowable1
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            # A user function call
            MyFunctions.Print ("Passed MyFlowable1",
                               Functions.GetDUTID());
            GoTo FlowMain_2;
        }

        Result 1
        {
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            # A user function call
            MyFunctions.Print("Failed MyFlowable1",
                              Functions.GetDUTID());
            SetBin SoftBins."3GHzLeakage";
            Return 1;
        }
    }

    #
    # Goes to FlowMain_3 on success
    # and returns 1 on failure.
    #
    FlowItem FlowMain_2 DatalogStartFlow
    {
        Result 0
        {
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            # A user function call
            MyFunctions.Print("Passed DatalogStartFlow",
                              Functions.GetDUTID());
            GoTo FlowMain_3;
        }

        Result 1
        {
            Property PassFail = "Fail";
```

```
        IncrementCounters FailCount;
        MyFunctions.Print("Failed DatalogStartFlow",
                          Functions.GetDUTID());
        Return 1;
    }
}

# This FlowItem calls the previously defined FlowTest1
FlowItem FlowMain_3 FlowTest1
{
    Result 0
    {
        Property PassFail = "Pass";
        IncrementCounters PassCount;
        # A user function call
        MyFunctions.Print("Passed FlowTest1",
                          Functions.GetDUTID());
        GoTo FlowMain_4;
    }

    Result 1
    {
        Property PassFail = "Fail";
        IncrementCounters FailCount;
        # A user function call
        MyFunctions.Print("Failed FlowTest1",
                          Functions.GetDUTID());
        SetBin SoftBins."3GHzCacheFail";
        Return 1;
    }
}

# This FlowItem calls the previously defined FlowTest2
FlowItem FlowMain_4 FlowTest2
{
    Result 0
    {
        Property PassFail = "Pass";
        IncrementCounters PassCount;
        # A user function call
        MyFunctions.Print("Passed FlowTest2",
                          Functions.GetDUTID());
        GoTo FlowMain_5;
    }

    Result 1
    {
        # FlowTest1 passed, but FlowTest2 failed
        Property PassFail = "Fail";
        IncrementCounters FailCount;
```

```
            # A user function call
            MyFunctions.Print("Failed FlowTest2",
                              Functions.GetDUTID());
            SetBin SoftBins."3GHzSBFTFail";
            Return 1;
        }
    }

    FlowItem FlowMain_5 DatalogStopFlow
    {
        Result 0
        {
            # All Passed!
            Property PassFail = "Pass";
            IncrementCounters PassCount;
            # A user function call
            MyFunctions.Print("Passed all!",
                              Functions.GetDUTID());
            SetBin SoftBins."3GHzAllPass";
            Return 0;
        }

        Result 1
        {
            # FlowTest1 and FlowTest2 passed,
            # but DatalogStopFlow failed
            Property PassFail = "Fail";
            IncrementCounters FailCount;
            # A user function call
            MyFunctions.Print("Failed DatalogStopFlow",
                              Functions.GetDUTID());
            Return 1;
        }
    }
}
```

【０４４３】

上記コードについて、以下の点に留意する必要がある。

【０４４４】

１．　ここでPListDefsセクションは、PList名をいくつか有し、またPList変数もいくつか有している。PList名は、テストにおいて直接用いることができる名前である。PList変数は、テストにおいて用いることができる変数であって、その値は、ランタイムでは、カスタマイズされたフローアブルにおけるコードによって実際のPListに結び付けられている。

【０４４５】

２．　PListDefsはオプション的である。もし存在しなければ、その中身は、さまざまなテスト宣言からのコンパイラによって推測される。もし存在すれば、それは、もっと多くを宣言しているかもしれないが、テストの使用されているPListパラメータの全てを宣言しなければならない。

【０４４６】

３．　ランタイムAPIはPList変数に対して値を割り当てるために利用可能である。テス

トプランクラスは関数

```
    Status SetPListVariable(const tester::String& varName,
                    const Tester::String& fiileQualifiedPListName);
```

【０４４７】

を有する。フローアブルは、PListVariableを特定のPListに結びつけるために上記関数を用いることができる。

【０４４８】

４．　移行のすぐ前に、FlowItemにおいてユーザ関数および関数を呼び出すことができる。この移行は、他のFlowItemへの移動制御か、あるいはリターンである。

ユーザ関数コールのためのC++

【０４４９】

フローにおいてカスタム関数コールを含むことをのぞいて、コンパイラによって生成されるであろうC++コードは、先に提示したさまざまなカスタム化手法について示されている。FlowItemにおけるユーザ関数コールは、好ましくは、それぞれのフローのIUserCallsメンバによって扱われる。それぞれのフローは好ましくは、以下に示すように、単一のバーチャルメンバ関数をエクスポートするインタフェースIUserCallsのメンバを有する。

```
class IUserCalls
{
public:
    virtual void exec(const String& flowItemName,
                        unsigned int result) = 0;
};
```

【０４５０】

ユーザ関数コールを有するFlowに出会うと、Flowは上記インタフェースをインプリメントするクラスのインスタンスに占められる。例えば、フローの例におけるFlowMainにおいて、以下のクラスのインスタンスで占められる。

```
class FlowMain_UserCalls : public IUserCalls
{
public:
    virtual void exec(const String& flowItemName,
                        unsigned int result)
    {
        if (flowItemName == "FlowMain_1")
        {
            // ...
        } else if (flowItemName == "FlowMain_2")
        {
            // ...
        } else if (flowItemName == "FlowMain_3")
        {
            switch (result)
            {
            case 0:
            MyFunctions::Print("Passed FlowTest1",
                                Functions::GetDUTID());
                return;
            case 1:
```

```
            MyFunctions::Print("Failed FlowTest1",
                                Functions::GetDUTID());
                return;
            default:
                return;
            }
        }
        else if (flowItemName == "FlowMain_4")
        {
            // ...
        }
        else if (flowItemName == "FlowMain_5")
        {
            // ...
        }
    }

};
```

【０４５１】

FlowItem::execute()動作は、フローアイテムの名前を知っている。それがポインタで次のフローに戻る前に、取り囲んでいるフローについてIUserCalls::exec()を呼び出し、それ自身のフローアイテム名および現在の結果の値を渡す。これにより、必要とされるユーザ定義関数を含む上記コードが実行される。

Ｃ９．　テストプログラムのコンパイル

【０４５２】

上で説明したように、Test Plan記述ファイルはテストプランにおいて用いられるオブジェクトとそれらの互いの関係とを指定する。ある実施形態においては、このファイルはC++コードに変換され、標準的なインタフェースITestPlanのインプリメンテーションの形態でサイトコントローラ上で実行される。このコードは、サイトコントローラ上にロード可能であるウィンドウズダイナミックリンクライブラリ（DLL）にパッケージ化されることができる。テストプログラムDLLは、サイトコントローラソフトウェアがそれに含まれるTestPlanオブジェクトを生成し、戻すために用いることができる標準的な既知のエントリポイントを有するように生成される。

Test Plan記述からのコンストラクション

【０４５３】

テストプラン記述からITestPlanのインプリメンテーションへの変換プロセスは、テストプログラムコンパイラ４００によって達成される。このプロセスは２つの段階、変換およびコンパイルで起こる。

【０４５４】

変換段階４０２において、コンパイラ４００は、テストプランファイル（およびそれがインポートするさまざまなほかのファイル）、ならびにそのテストプランにおいて用いられる全てのテストタイプについてのプリヘッダを処理する。この段階において、コンパイラは、Test PlanオブジェクトについてのC++コードと、出会うテストタイプについてのC++ヘッダとを、MSVC++（マイクロソフトVisual C++）ワークスペースおよびプロジェクトファイル、DLL「ボイラープレート（boilerplate）」コード等のような全ての他のサポートファイルとともに作成する。コンパイラ４００は、コンパイラタイムエラーメッセージが、生成されたコードの代わりに、記述ファイル内の適切な場所を参照し返すことを確実にするようにファイルおよびライン指令文（directive）を生成されたコードに挿入する

。
【０４５５】
コンパイラが必要なファイルを生成した後に起こるコンパイル段階において、MSVC++コンパイラのような標準的なコンパイラ４０４は、ファイルをコンパイルしてそれらをDLLにリンクするように呼び出される。
【０４５６】
コンパイラは、入力として、有効なテストプランファイル（および全ての関連するファイル）を取り出して、必要なときに、TestPlanファイルと、テストプランファイルにおいて「Import」指令文によって表される全ての他のファイルとを生成する。また、コンパイラは、Test Plan DLLをビルドするためにMSVC++「ソリューション」を生成する。例えば、メインファイル（MyTestPlan.tpl）がタイミング情報を組み込むためにTiming1.timを含んでいれば、コンパイラは以下のファイルを作成（他のものの間で）することになる。

1. MyTestPlan.h
2. MyTestPlan.cpp
3. Timing1.cpp
4. MyTestPlan.sln (MSVC++ "Solution" file)
5. MyTestPlan.vcproj (MSVC++ "Project" file)

【０４５７】
全てのファイルが作成された（あるいは更新された）後に、コンパイラは、それが作成した「ソリューション」を指定するMSVC++アプリケーションを呼び出し、DLLをビルドする。いかなるエラーおよび／あるいは警告もユーザに対して示される。
【０４５８】
もし、Test Planをビルドした後に、ユーザがTiming1.timに対して変更を行ったら、ユーザはコンパイラを呼び出して、それにMyTestPlan.tplを渡す。コンパイラは、メインテストファイルは変わっていないことを（タイムスタンプ情報によって）認識し、MyTestPlan.h/.cppは再作成されない。しかし、メインテストプランファイルを処理している間に、コンパイラは、Timing.timファイルが変わったことを見つける。したがって、コンパイラはTiming1.cppファイルを再作成し、DLLを再ビルドするようにMSVC++アプリケーションを呼び出す。これはMyTestPlan.cppを再コンパイルすることを避け、Timing1.cppをコンパイルして、DLLに再リンクするだけである。このアプローチは、コンパイルにかなりの時間をとるような大きなテストプランについて、再コンパイル、再リンクの回数を削減するのに特に有用である。

Ｄ．　テストプログラムを動作させる
【０４５９】
サイトコントローラソフトウェアは、テストプログラムDLLをその処理空間にロードし、Test Planオブジェクトのインスタンスを作成するためにDLL内に「factory」関数をコールする。一旦Test Planオブジェクトが作成されると、サイトコントローラソフトウェアはテストプランを実行、あるいは他の必要な方法でそれに相互作用することができる。

非インタラクティブビルド
【０４６０】
ウィンドウズ環境の大半のC++ソフトウェア開発者は、アプリケーション（あるいはDLL、あるいはライブラリ等）を作ることは、開発環境（MS Visual C++、Borland C++、あるいは類似したもの）を立ち上げ、コードを編集し、そして（しばしば）製品を作るためにボタンを押すことを意味する。
【０４６１】
発明の一実施形態のテスト環境は、同様のアクティビティのセットを有する。テストプランの開発者は、コードを編集して、彼らのテストプランを構成する必要がある。しかし

ながら、テスタは、結果としてのTest Plan DLLを生成するためにC++開発環境を立ち上げることをテストプラン開発者には要求しない。

【０４６２】

これを実現するために、本発明は、非インタラクティブなビルドの概念を採用する。非インタラクティブなビルドは、非インタラクティブなモードにおいてMS Visual C++を使うビルドとして定義される。これは依然として、このようなビルドを管理するために他のツールをインタラクティブに用いることを可能にするということに留意されたい。唯一の暗黙の意味は、Visual C++を非インタラクティブに用いるということである。

仮定された環境

【０４６３】

ユーザの環境についてある仮定をする。この仮定とは、

【０４６４】

１．テストプラン開発者は、上記方法およびルールに従ってテストプランを開発している。

【０４６５】

２．テストプラン開発者は、C++の専門化レベルの知識をもっていないかもしれない。

【０４６６】

３．テストプラン開発者は、ファイルをTest Plan DLLに変換するためにコマンドラインツールあるいはGUIツールへのアクセスを有している。

ボタンなしでアプリケーションを作る

【０４６７】

非インタラクティブにMS Visual Studioで作業をすることは２つのアプローチのうちの１つを必要とする。一番目（そして最も単純なもの）は、コマンドラインインタフェースを用いることである。二番目（そしてより柔軟性のあるもの）はオートメーションインタフェースを用いることである。このセクションでは両方のアプローチを説明する。

プロジェクトを作成する

【０４６８】

非インタラクティブにVisual Studioを用いるためには、一つ以上の有効なプロジェクトを含む作業ソリューション（working Solution）で始めないとならない。不運なことに、これは、コマンドラインのアプローチでもオートメーションのアプローチでも達成することができないタスクである。方法もプロジェクト作成のためのメカニズムも提供しない。しかし、Visual Studio用のプロジェクトおよびソリューションは、テンプレートから作成することができる。したがって、開始するために、プロジェクト名およびテンプレートが与えられると、Visual Studio用のソリューション／プロジェクトを作成することができる。

プロジェクトを占有する

【０４６９】

生成されたプロジェクトに新しいファイルを追加することは、Visual Studioのオートメーションモデルを用いる。コマンドラインはこれをサポートしていないからである。我々は、プロジェクトに新しいファイルおよび既存のファイルを追加するために２つのVisual Studioマクロを提供する。類似したコードは、同じタスクを行うためにActiveScriptエンジンを用いる外部のスクリプト（VBScript、Jscript、ActivePerl、ActivePython等）によって用いられ得る。したがって、我々のコード生成ツールは、新しいファイルを作成し、オートメーションモデルを用いて、それらを既存のVisual Studioプロジェクトに追加することができるであろう。ファイルが生成された後に、それらをツールによって必要なときに更新することができる。

プロジェクトを作る
【０４７０】
一旦ソリューションおよびプロジェクトの準備ができると、テストプランを作るために非インタラクティブにVisual Studioを用いることに対していくつかの選択肢がある。最も単純な選択肢は、それをコマンドラインから呼び出すことである。このようなコマンドラインは、以下のようであり、

derenv solutionFile /build solutionCfg

【０４７１】
solutionFileは、Visual Studioソリューションファイルであり、solutionCfgはそのソリューション内のプロジェクトに適用可能である具体的なコンフィギュレーションである。他のソリューションは、オートメーションについてのVisual Studio Object Modelを用いることである。これにより、ビルドプロセスおよびコンフィギュレーションプロセスに対してよりきめの細かい制御が可能である。上述したように、それはコマンドラインからプロジェクトを作るためにPerlスクリプトのリストを含んでいる。このプログラムは、作るべきプロジェクトおよびコンフィギュレーション（ならびにプロジェクトについてのほかの情報）を指定するコンフィギュレーションファイルを読んで、それらをオートメーションモデルを用いて全て作る。スクリプトにおいてオートメーションオブジェクトをどのように用いるかの例について、このスクリプトにおける$msdvオブジェクトの使用を参照されたい。

デバッガサポート
【０４７２】
テストクラスの開発者が彼らの作業を検証してデバッグするために、彼らは、彼らがサイトコントローラに入り込んでコードをたどることを可能にするデバッガへのアクセスを必要とする。コンパイラによって生成されたコードは、MSVC++によってコンパイルされるC++であるので、我々は、テストクラスインプリメンテーションのデバッグのためにMSVC++デバッガを用いる。この特徴は、テストクラス開発者についてのみ意味をもつのではなく、直接C++で作業する者についても重要であるということに留意されたい。他のメカニズムが、生成されたC++コードを直接参照することなく、テストプログラムの動作デバッグしたい、あるいはたどりたいと願うテストエンジニアに対して提供される。

システムソフトウェア環境
【０４７３】
このセクションは、テスタのための一般的なソフトウェア環境、すなわち、ユーザテストプランによって要求されるファイルの場所、このようなファイルの代わりの場所を指定するためのメカニズム、およびテストプランおよびモジュール制御ソフトウェアの配置を指定する方法を説明する。

テストプランによって要求される環境

テストプランによって要求されるシステム標準の場所、ならびに
１．　パターンリスト
２．　パターン
３．　タイミングデータ
４．　テストクラスDLL

【０４７４】
についてのサーチパスのランタイムコンフィギュレーションは、環境コンフィギュレーションファイルによって指定されるように「環境」変数によって構成されてもよい。これら

は、例えば以下のような単純な構文を有するテキストファイルである。

```
Tester_PATOBJ_PATH = "patterns\data;D:\projects\SC23\patterns\data"
```

【０４７５】

ネイティブなOSサポートの環境変数を通じてではなく、テキストファイルに定義されたこのような「環境」を有することの利点は、OSサポート環境変数が有する、最大文字列長等の一般的な限定によってインプリメンテーションが制限されないことである。以下の「環境」（セットアップ）変数は、上に列挙した構成要素について用いられる。

```
\
Pattern lists: Tester_PATLIST_PATH.
Pattern object files: Tester_PATOBJ_PATH.
Pattern source files: Tester_PATSRC_PATH (this is optional; please see).
Timing data files: Tester_TIMING_PATH.
Test class DLLs: Tester_TEST_CLASS_LIBPATH.
```

【０４７６】

具体的な事例をサポートするために、有用なデフォルトの振る舞いを維持しつつ、我々はコンフィギュレーションの３つのレベルを提供する。これらは、優先度が増えていく順に説明する。

【０４７７】

まずシステム環境セットアップファイル?Tester_INSTALLATION_ROOT?cfg?setups?Setup.envは、「環境」変数のデフォルトの値を指定する。他のコンフィギュレーションメカニズムが利用可能でなければ、このファイルが要求される。一般的に、それはシステム上で動作する全てのテストプランについて利用可能である。このファイルは、上述した３つの変数についてのデフォルトの値を割り当てるために、インストーラからの入力とともに、インストール中にインストールおよびコンフィギュレーション管理（ICM）システムによって作成される。（なお、上記３つの変数についてのシステムデフォルトに加えて、このファイルは、次のサブセクションで説明する他のテスタ［環境］変数についてのシステムデフォルトも含んでいる。）

【０４７８】

第二に、環境セットアップファイルは、テストプランへのランタイム引数としてユーザによって指定されてもよい。このランタイムコンフィギュレーションにおける変数は、デフォルトの定義に優先する。

【０４７９】

最後に、テストプランは、その実行において用いられるべき環境変数を指定するために特別なブロックを用いてもよい。テストプランにおいて定義された変数は、デフォルトシステムファイルあるいはユーザ定義ファイルにおける変数に優先する。

【０４８０】

一般的に、全ての必要な変数は、上述したメカニズムのうちの一つを通じて定義されるべきである。もし変数が定義されなければ、ランタイムエラーが生じる。

他の環境セットアップ

【０４８１】

ユーザテストプランによって要求される［環境］変数に加えて、以下の２つの［環境］変数がテスト環境によって要求される。

【０４８２】

１．Tester_TEST_PLAN_LIBPATH：これは、システムコントローラがロードされるべきユーザテストプランDLLを見つけるために用いるサーチパスを指定する。同じサーチパスが、ユーザピン記述ファイルおよびソケットファイルを見つけ出すためにも用いられることに留意されたい。この変数のデフォルト値は、インストール中にICMに指定され、ICMによってファイル$Tester_INSTALLATION_ROOT?cfg?setups?Setup.envに記憶される。

【０４８３】
２．Tester_MODULE_LIBPATH：これは、ベンダ提供のハードウェアモジュール制御ソフトウェアDLLをロードするためにシステムが用いるサーチパスを指定する。この情報は、コンフィギュレーション管理データベース（CMD）から抽出され、ファイル$Tester_INSTALLATION_ROOT?cfg?setups?Setup.envにICMによって記憶される。
【０４８４】
ユーザはTester_TEST_PLAN_LIBPATH変数についてSetup.envファイルにおいて与えられる値を無効にすることができるが、Setup.envファイルにおいてTester_MODULE_LIBPATHについて与えられる値は、ユーザがベンダ提供のハードウェアモジュール制御ソフトウェアDLLについてのサーチパスを明示的に変更することを望まなければ、ユーザによって変更されるべきではないことに留意されたい。

サーチパス指定セマンティクス
【０４８５】
サーチパスを指定する「環境」変数について以下の点に留意しなければならない。
【０４８６】
１．　それぞれは、システムが特定のタイプの参照されるファイルを見つけ出すためにサーチを行うディレクトリ名のセミコロンで区切られたリストでなければならない。
【０４８７】
２．　このような「環境」変数の値をシステムが最初に参照した後は、その値に対するユーザによるいかなる変更（例えば環境コンフィギュレーションファイルを編集することによる）も、ユーザがそれをする必要があることをシステムに明示的に「知らせる」ときに、システムによって登録されるのみである。
【０４８８】
３．　分散環境における「現在の作業ディレクトリ」CWDの注釈――テスタが作業する環境等――は、ユーザが直感的にそれが予想するものではないかもしれないために（CWD）に関連するパスがあいまいな結果をもたらし得るときに、サーチパスにおける関連するパス名は、関連する環境変数（ルートを規定する機能を提供する）の特定の設定に関連するものとして解釈される。この関連する環境変数は、サーチパスにおける全ての関連パス名が関連を有すると仮定されるルートを指定するものであるが、これは「Tester_INSTALLATION_ROOT」変数であり、ユーザのシステム上へのテスタのインストールの最上位レベル（すなわち「ルート」ディレクトリの場所を与える。
【０４８９】
４．　ディレクトリエントリは、セット[V:*?"<>|;]の文字を含むことができない。セミコロン（;）をのぞいてこのセットの他の全ての文字は、ウィンドウズのファイル名では不正であることに留意されたい。セミコロンは、サーチパスエントリでは用いるべきではない。なぜならこれは、サーチパスにおけるエントリの境界を定めるために用いられるからである。なお、パス名は埋め込まれた空白を有することができるが、パス名の直前、直後（すなわちパス名における最初の空白ではない文字の前および最後の空白ではない文字の後）にくる全ての空白は、パス名の一部とは考えられず、無視される。
【０４９０】
５．　サーチパスディレクトリは、それらが定義に登場する順番でサーチされる。ファイルの最初の登場が選択されるものである。

Ｅ．　テストパターン
【０４９１】
テストパターンファイルの非常に大きなセットの効率的に管理し、扱い、ロードすることが発明の一実施形態のフレームワークの重要な構造上の局面である。階層的なパターンリストの考えは、扱いやすい概念化ならびにシステムの使用の容易さをエンドユーザに提供する有効なツールであるとみなされる。

【０４９２】
DUTへの刺激は、テストベクトルを通じてテストシステムに役立てられる。ベクトルは一般的に、シーケンシャル（あるいはリニア）、スキャン、あるいはアルゴリズムパターン生成器（APG）由来のものに分類可能である。発明の一実施形態のシステムにおいては、テストベクトルを、テストタイムにDUTに適用されるパターンについて組織化する。パターンは、ユーザのテストプログラムにおけるパターンオブジェクトによって表される。そのシステムにおいては、パターンは、パターンリストにおいて組織化され、パターンリストオブジェクトによってプログラム的に表される。パターンリストオブジェクトは、順序付けられたパターンのリストあるいは他のパターンリストを表す。順序付けは、潜在的にはリストのコンポーネントの宣言の順である。なお、もし一つのパターンだけが必要とされれば、それは単独でリストにカプセル化される必要がある。

【０４９３】
ユーザのテストプログラムにおけるパターンリストオブジェクトは、ディスク上のパターンリストファイルに関連付けられており、それはパターンリストの実際の定義を含んでいる。したがってパターンリストの内容は、関連付けられているディスクファイルの内容によって動的に決定される（これについては後でより多くを説明する）。

【０４９４】
パターンリストの定義は、パターンリストについて明示的な名前を提供し、ファイル名の関連付けを通じて、順序付けられたパターンのリストおよび／あるいは他のパターンリストを特定する。またそれは、実行オプションの指定も提供する。これについては、このオプションはパターンリストおよびパターンの両方に適用可能であるので、パターンオブジェクトを述べた後に詳細に説明する。パターンリストは以下のルールに従わなければならない。

```
file-contents :
    version-info global-pattern-list-definitions
version-info :
    Version version-identifier ;
global-pattern-list-definitions :
    global-pattern-list-definition
    global-pattern-list-definitions global-pattern-list-definition
global-pattern-list-definition :
    global-pattern-list-declaration { list-block }
global-pattern-list-declaration :
    GlobalPList pattern-list-name options_opt
list-block :
    list-entry
    list-block list-entry
list-entry :
    pattern-entry ;
    pattern-list-entry ;
    global-pattern-list-definition ;
    local-pattern-list-definition ;
pattern-entry :
    Pat pattern-name options_opt
pattern-list-entry :
    PList pattern-list-reference options_opt
pattern-list-reference :
    pattern-list-qualified-name
    file-name ':' pattern-list-qualified-name
```

```
pattern-list-qualified-name :
    pattern-list-name
    pattern-list-qualified-name '.' pattern-list-name
local-pattern-list-definition :
    local-pattern-list-declaration { list-block }
local-pattern-list-declaration :
    LocalPList pattern-list-name optionsopt
options :
    option
    options option
option :
    [ option-definition ]
option-definition :
    option-name option-parameteropt
option-parameters :
    option-parameter
    option-parameters ',' option-parameter
```

【０４９５】

以下は、上で用いた未定義の非ターミナルの説明である。

１．　version-identifier：　セット[0-9]からの一つ以上の文字の列であり、最初の文字は数字でなければならない

【０４９６】

２．　name：　セット[a-zA-Z_0-9]からの一つ以上の文字の列でり、最初の文字はセット[a-zA-Z]からの文字でなければならない。

【０４９７】

３．　pattern-list-name：　セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z_]からの文字でなければならない。

【０４９８】

４．　file-name：　有効なウィンドウズファイル名（このファイル名に空白が含まれるのであれば二重引用符で挟まれなければならない）。これは単純なファイル名でなければならない、すなわちディレクトリコンポーネントを有すべきではないことに留意されたい。pattern-list-referenceは、同じファイル内のパターンリストへの内部参照か、あるいは他のファイルのパターンリストへの外部参照のどちらかであり得る。外部参照は、file-nameによって制限される必要がある。

【０４９９】

５．　option-name：　セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z_]からの文字でなければならない。

【０５００】

６．　option-parameter：　セット[a-zA-Z_0-9]からの一つ以上の文字列である。

【０５０１】

パターンリストファイルはコメントをサポートする。コメントはパターンリストファイル解析部によって無視されることを意味する。コメントは、「#」の文字で始まり、行末まで続く。

Ｅ１．　パターンリストのルール

【０５０２】

パターンリストについての静的な、あるいはコンパイルタイムルールは名前の宣言と解決とを支配する。パターンリスト言語における名前は、グローバルパターンリスト定義およびローカルパターンリスト定義によって宣言される。それらは、パターンリスト参照によって参照される。これらの宣言および参照を支配するルールのいくつかを以下に示す。

【０５０３】
１．　グローバルパターンリスト定義またはローカルパターンリスト定義は、パターンリストの名前を宣言する。パターンリスト参照は、宣言されたパターンリストの名前を参照する。グローバルパターンリストの名前はグローバルに知られる。ローカルパターンリストの名前は、それらが宣言されたリストブロック内でのみ知られる。それらは、そのリストブロック内では直接制限なしで参照されることができる。より深く繰り込まれた宣言においては、ローカルパターンリストは制限された名前によって参照される必要があるであろう。
【０５０４】
２．　ローカルパターンリスト名は、周りのパターンリストの範囲内で知られ、グローバルパターンリスト名はシステムの範囲内で知られる。例えば、

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

        GlobalPList G2
        {
            ...
        }

        PList L2;         # OK.  Name L2 is known in this scope
        PList G2         # OK.  Name G2 is global
    }

    PList L2;             # Error.  Name L2 is not known here.
    PList L1.L2;         # OK. Name L1 is known here.  L2 is known by
                         #        qualification.
    PList G1.L1.L2;     # OK.  Qualification by G1 is not needed but
                         # is allowed.
    PList G2;             # OK. Name G2 is global
}
```

【０５０５】
３．　グローバルパターンリストは、パターンリストファイルにおける最外レベルで定義されるか、あるいは周りのパターンリスト内で繰り込まれたものとして定義されてもよい。しかしながら、繰り込みは便宜のためにすぎない。それらは概念的には、ファイルの最外レベルのグローバルパターンリストとして定義される。繰り込まれたグローバルパターンリストは、同名の最外の（繰り込まれていない）グローバルパターンリストと意味的には等価である。したがって例えば

```
    GlobalPList G1
    {
        GlobalPList G2 ...
    }
```

is semantically equivalent to:

```
GlobalPList G1
{
    PList G2; # References G2
}

GlobalPList G2 ...
```

【０５０６】

４．　全てのグローバルパターンリストは、固有の名前を付けられる。

```
GlobalPList G1
{
    # Note that this is as if declared at the outermost level
    # with a reference to it right here.
    GlobalPList G2
    {
        ...
    }
}

# This declaration will be an error in this or any other file,
# as the name G2 is already taken.
GlobalPList G2   # Error.  Global name G2 is taken.
{
    ...
}
```

【０５０７】

５．　ローカルパターンリストは常に、ローカルパターンリストの名前の範囲を決定する周りのパターンリスト内に繰り込まれた定義を有している。ローカルパターンリストは、周りのパターンリスト内で固有の名前をつけられる。ローカルパターンリストは、パターンリストファイルの最外レベルに登場することを構文的には認められない。

```
GlobalPList G1
{
    LocalPList L1
    {
    }

    LocalPList L2
    {
        LocalPList L1  # OK. No local name L1 is declared directly
                       # in the enclosing scope defined by L2.
        {
        }

        PList L1;    # OK. Refers to L1 declared in L2
        PList G1.L1; # OK. Refers to L1 declared in G1.
    }

    # Error. Redeclaring name L1 when the enclosing scope
    # defined by G1 already has an L1 declared in it.
```

```
        LocalPList L1;
        {
        }

    }
```

【０５０８】

６．　各パターンリストファイルは、一つ以上のグローバルパターンリストについての定義を含んでいる。これは、直接構文から得られる。最外レベルはグローバルパターンリスト定義であり、それらの少なくとも一つがなければならない。

【０５０９】

７．　パターン名はパターンへの参照であり、Patというキーワードの後に続く。それは、その名前がパターン名に接尾辞.patを連結させることによって得られるようなパターンファイル内にあるパターンを参照する。このファイルは、パターンについて定義されたサーチパスに沿って得られるファイルを表している。

【０５１０】

８．　パターンリスト参照は、PListキーワードの後に続くパターンリストへの参照である。この参照は、オプションのファイル名からなり、その後に、ドットによって区切られる名前の単なるリストにすぎない制限されたパターンリスト名が続く。したがって、例えば、ファイルfoo.plistにあるグローバルパターンリストG1に繰り込まれているL1に繰り込まれているL2に繰り込まれているローカルパターンリストL3を参照するパターンリスト参照は以下のようであり得る。

```
PList foo.plist:G1.L1.L2.L3;
```

【０５１１】

上の名前で一番左の名前セグメントはG1である。

【０５１２】

一番左の名前セグメントは、グローバルパターンリストか、あるいは参照点から見えるローカルパターンリストのいずれかにならなければならない。

【０５１３】

パターンリスト参照の名前の解決は以下のように進行する。

【０５１４】

１．　各名前部分は、その前の接頭辞の中身で宣言される名前になる。

【０５１５】

２．　もしファイル制限があれば、一番左の名前セグメントは名付けられたファイルにおいて宣言されたグローバルパターンリストになる。

【０５１６】

３．　もしファイル制限がなければ、一番左の名前は、取り囲む範囲内のローカルパターンリストになり得、もしそれが失敗すれば、次の取り囲む範囲、その次の範囲と、取り囲むグローバルな範囲まで続く。

【０５１７】

４．　範囲のサーチを最も近い取り囲むグローバルな範囲に限定することは、それらがパターンリストファイル内の最外レベルで宣言されたかのようにグローバルな範囲のセマンティクスに従うために必要とされる。もし繰り込まれたグローバルな範囲が最外レベルで（等価的に）原文どおりに宣言されれば、名前の解決のサーチは、その範囲を調べた後に終了する。

【０５１８】

５．　もし参照が前述のステップによって解決しなければ、一番左の名前セグメントは、この同じファイル内のグローバルパターンリストになるとすることができる。

【０５１９】

６．　もし前述のステップによって参照が解決しなければ、一番左の名前セグメントは

、「.plist」という接尾辞を一番左の名前セグメントに付加することによってファイルで名付けられたグローバルパターンリストとなるとすることができる。

【０５２０】

７．　もし参照が前述のステップによって解決されなければ、参照は間違っている。

【０５２１】

先に述べたように、上記ルールは、一番左の名前部分は、参照点から見えるローカルパターンリストであるか、グローバルパターンリストになることを決定する。

【０５２２】

次の例はこれらの考えのいくつかを示す。

```
GlobalPlist G1
{
    PList G3;  # OK. Refers to a pattern list later in this file.

    PList G4;  # OK. Refers to a pattern list in file "G4.plist"

    # OK.  Refers to G1 in the file "my_plists.plist".
    PList my_plists.plist:G1;

    # OK.  Refers to a pattern list in file "my_plists.plist".  The
    # qualified name refers to a local pattern list named L2 declared
    # in the scope of a local pattern list named L1 declared in the
    # scope of a global pattern list named G1.
    PList my_plists.plist:G1.L1.L2;

    LocalPList L1
    {
        LocalPList L2
        {
        }
    }

    PList L1;  # OK. Refers to L1 declared in the
               # enclosing scope of G1

}

GlobalPlist G2
{
    LocalPList L2
    {
    }

    GlobalPList G3
    {
      LocalPList L3
        {
        }

        PList L1; # Error.  No L1 declared in this or any enclosing
```

```
                # scope;

            # Error.  The name L2 is not declared in this scope.  Also
            # though L2 is declared in the enclosing scope, this scope
            # is global, and so no further enclosing scope is examined.
            #
            # Contrast with reference to name L2 in LocalPList L3 below.
            PList L2;

            PList G1.L1;   # OK.  Refers to L1 in G1.

            # Error.  G3 is not really nested inside G1.  Since G3
            # is global, it is really declared at an outermost level,
            # and so G1.G3 is meaningless.
            PList G2.G3.L3;
        }

        LocalPList L3
        {
            # OK.  Refers to G2.L2.  The enclosing global scope is G2
            # and the name L2 is declared in G2.
            PList L2;
        }
    }
```

【０５２３】

全てのパターンリストファイル名およびパターンファイル名は、それらを用いるテストプランにわたって固有のものであることが要求される。

【０５２４】

パターンリスト参照は、同じファイルにおける参照の前または後に定義されるパターンリストを参照することができる。

【０５２５】

再帰パターンリスト定義および相互に再帰的なパターンリスト定義は許容されない。このような定義をユーザが作成することを妨げるものは、パターンリストファイル構文には何もなく、解析部はこのような状況を検出するとエラーのフラグを立てる。このような状況の検出に関連していくらかのコストがかかることに留意されたい。ユーザは、もしユーザが入力空間が互いに再帰的な定義を有していないことを保証する責任を負うことができるかどうかのチェックをオフに切り替えることができる。

```
    GlobalPList G1
    {
        LocalPList L2
        {
            LocalPList L3
            {
                 # Error.  L2 runs L3 which runs L2.
                 # This is a recursive reference to L2
                 PList L2;

                 PList G2;
            }
```

```
        }
    }

    GlobalPList G2
    {
        # Error.  G1.L2 runs L3 which runs G2 which runs G1.L2.
        # This is a mutually recursive reference to G1.L2.
        PList G1.L2;
    }
```

【０５２６】

パターンおよびパターンリストの構文的な記述は、オプションがそれらにおいて指定されることを可能にする。一般的にオプションはベンダに特有のものである。構文は、いかなるパターンまたはパターンリストが指定されたいくつものオプションを有することを可能にする。オプションのそれぞれはいくつものパラメータを有する。大半のベンダによって認識されるいくつかのサポートされたオプションを説明する。

【０５２７】

パターンツリーの動的な（すなわち実行）セマンティクスを、パターン実行シーケンスを定義した後に説明する。

Ｅ２．　パターン

【０５２８】

図６は、本発明の一実施形態によるパターンコンパイラ６０２およびパターンローダ６０４を示す。パターンのユーザによって定義された中身は、プレインテキストファイルであるパターンソースファイル６０６において利用可能である。パターンコンパイラは、ソースファイルを、テスタハードウェア上へのロードに適したモジュール特有のフォーマットにコンパイルする任務をもつ。後者のファイルは、パターンオブジェクトファイルとして参照される。一般的な属性は以下の通りである。

【０５２９】

１．　パターンオジェクトはユーザによって作成可能ではなく、むしろ、ユーザは常にパターンリストを扱い、これは他のパターンリストおよび／あるいはパターンの集合体である。パターンリストオブジェクトは、その中に含まれるパターンオブジェクトを、必要に応じてユーザに対してアクセス可能としつつ、作成し、所有し、保持する。

【０５３０】

２．　パターンは、テストプラン内で固有の名前を付けられている。すなわち、テストプラン内で２つのパターンが同一の名前をもつことはできない。パターンの名前は、それを含むファイルの名前とは別個のものである。パターンファイル名は、パターンを参照するためにパターンリストファイルにおいて用いられる名前であり、パターンの実際の名前はパターンファイルにおいて定義される。

【０５３１】

発明のある実施形態においては、一般的に、単一のDUT（テスト対象装置）が異なるベンダからのテスタモジュールに接続される可能性がある。これは、パターンコンパイル－ロード－実行のチェーン全体に影響を与える。主な影響をこのセクションで述べる。

Ｅ３．　パターンコンパイル

【０５３２】

したがって、パターンコンパイラ６０２は、（用いられるベンダ特有のデジタルモジュールに関して）特定のサイトコンフィギュレーションを目標とする必要がある。この議論の残りについて、「モジュール」という用語は、一例としてのデジタルモジュールを指すために用いられるものとする。異なるベンダからのモジュール６０８をシステムに統合することを可能にするために、以下の手順が好まれる。

【０５３３】
１．　各モジュールベンダは、自身のモジュールに特有なパターンコンパイラ６１０を、動的にロード可能なライブラリまたは別々の実行ファイル（executable）の形態で提供する責任がある。このコンパイラライブラリ／実行ファイルは、最低限でも、引数として
ａ．　（一つ以上の）パターンソースファイルパス名のアレイ
ｂ．　ピン記述ファイル名
ｃ．　ソケットファイル名
ｄ．　コンパイルされたオブジェクトの行先を指定するオプション的なディレクトリパス名
ｅ．　いかなるベンダ特有パラメータの指定も可能にする文字列名／値のペアのオプション的なアレイ（他のベンダは無視することができる）
【０５３４】
をとるよく知られたcompile()関数を提供する。
【０５３５】
２．　パターンソースファイルは、２つの異なるタイプのセクションを収容する。
ａ．　全てのコンパイラがアクセス可能である（しかし必ずしも用いられない）情報を含む「共通」セクション
【０５３６】
ｂ．　一つ以上のオプション的なベンダ特有のセクション。それぞれ、固有のベンダコードで識別され、特定のベンダのコンパイラによって使用可能な情報のためのものである。
【０５３７】
３．　ベンダのコンパイラはパターンオブジェクトファイルを直接作成しない。代わりに、テスタが、パターンコンパイラの一部であるオブジェクトファイルマネージャ（OFM）６１４によって管理されるパターンオブジェクト「メタファイル（metafile）」６１２を提供する。パターンコンパイラは、システムコントローラとして機能するコンピュータ上に配置されてもよいし、オフライン、例えばシステムコントローラが接続されるネットワーク上に配置されてもよい。ここまで抽象的な用語で言及している「パターンオブジェクトファイル」は、実際にはこのオブジェクトメタファイルである。オブジェクトメタファイルは、パターンソースファイルと同じ名前を付けられ、ソースファイルの拡張子はオブジェクトファイルの拡張子に置き換えられる。OFMはこのファイルを読み書きするためのアプリケーションプログラミングインタフェース（API）を提供する。オブジェクトメタファイルは、
ａ．　共通ヘッダ情報
ｂ．　対応するモジュールおよびそのモジュールについてのパターンデータの場所を特定する情報を含む、モジュール特有のヘッダ情報
【０５３８】
ｃ．　モジュールベンダによって要求されるように組織化されたモジュール特有のパターンデータ。モジュールベンダが解釈可能である。
【０５３９】
を記憶するための規定を有している。
【０５４０】
OFM APIは、モジュールベンダのコンパイラが、モジュールに特有なヘッダ情報およびデータをオブジェクトメタファイルに書き込むことを可能にする。なお、オブジェクトメタファイルのこのレイアウトによって、目標とされるサイトにおける２つ以上のモジュールが同一のものである場合であっても、パターンデータをプレモジュールベースで組織化することが可能である。
【０５４１】
ダイレクトメモリアクセス（DMA）のような効率的なデータ通信を活用するモジュール特有ハードウェアローディング情報の生成を容易にするために、パターンコンパイラは、

追加のベンダ供給コンフィギュレーション情報を必要とするかもしれないということに留意されたい。

Ｅ４．　モジュールのためのパターンロード

【０５４２】

各モジュールベンダは、一般的な手順の後に続く、自身のパターンローディングメカニズム６１５を提供する責任がある。モジュール６０８のパターンオブジェクトメタファイル６１２は、異なるセクション６１６にモジュール特有データを記憶する。ベンダインプリメンテーションは、パターンオブジェクトメタファイルからの関連するモジュール特有セクションにアクセスするためにOFM APIを用いる。テスタフレームワークは、メタファイルの適切なセクションからモジュール特有データをモジュールにロードするために、各モジュールのロード方法を呼び出す責任がある。

Ｅ５．　パターンファイル

【０５４３】

各コンパイラベンダに、パターンに関する完全に異なるプレインテキストフォーマットを指定させることが可能であり、これは実際に、大半の場合に必要とされる可能性がある。しかしながら、一般的に、モジュールにわたって整合的で同一のセマンティクスがどのベクトルについても必要とされるようなサイクルベースのテスト環境については、パターンファイルについての共有され、一般化された構文は望ましいだけでなく、必要であるかもしれない。この共有された構文は、パターンソースファイルの「共通」セクションについて何が指定されているかということである。実際、大多数の場合について、「共通」セクションがパターンファイルにおいて要求される唯一のセクション（ヘッダ情報以外では）であるということが想定され、どのベンダのコンパイラも、そのセクションだけとともに動作する。このセクションは、全てのコンパイラが解釈することができなければならないパターンファイルについてのルールを提示している。パターンファイルは以下の通り組織化される。

```
file_contents           :
                            version_info pattern_definitions
version_info            :
                            Version version-identifier ';'

pattern_definitions           :
                            pattern_definition
                    pattern_definitions pattern_definition
pattern_definition            :
                            main_header '{' main_section '}'
                            main_header '{' main_section vendor_sections '}'
                            subr_header '{' subr_section '}'
                            subr_header '{' subr_section  vendor_sections '}'
main_header             :
                            MainPattern identifier
main_section            :
                            CommonSection '{' common_contents main_section_domai
ns '}'
common_contents         :
                            timing_reference timing_map_reference
timing_reference        :
                            Timing file-name ';'
```

```
timing_map_reference    :
                          TimingMap file-name ';'
main_section_domains    :
                          main_section_domains main_section_domain
                          main_section_domain
main_section_domain     :
                          Domain domain_name '{' main_section_contents '}'
domain_name             :
                          identifier
main_section_contents   :                                                  10
                         main_section_contents main_section_content
                          main_section_content
main_section_content    :
                         label_spec main_pattern_spec
                          main_pattern_spec
label_spec              :
                          label ':'
label                   :
                          identifier
main_pattern_spec       :                                                  20
                          main_operation capture_mask_flag '{' vectors_and_wav
eforms '}'
main_operation          :   /* empty */
                          common_operation
                          jal_op
                          jsr_op
                          jsrc_op
                          jsc_op
                          exit_op
common_operation        :                                                  30
                          idxi_op
                          idxin_op
                          jec_op
                          jech_op
                          jff_op
                          jffi_op
                          jni_op
                          ldin_op
                          nop_op
                         pause_op                                          40
                         sndc_op
                         sndt_op
                         stfi_op
                         sti_op
                         stps_op
                         wait_op
/*
 *   Instructions specific to the MAIN Patterns
 */
jsr_op                  :                                                  50
```

```
                        JSR identifier
jsrc_op             :
                        JSRC identifier
jsc_op              :
                        JSC identifier
jal_op              :
                        JAL identifier
exit_op             :
                        EXIT
/*
 *   Instructions common to both MAIN and SUBR Patterns
 */
idxi_op             :
                        IDXI 24-bit number
idxin_op            :
                        IDXIn index-register
jec_op              :
                        JEC identifier
jech_op             :
                        JECH identifier
jff_op              :
                        JFF identifier
jffi_op             :
                        JFFI identifier
jni_op              :
                        JNI identifier
ldin_op             :
                        LDIN index-register
nop_op              :
                        NOP
pause_op            :
                        PAUSE
sndc_op             :
                        SNDC 8-bit number
sndt_op             :
                        SNDT 8-bit number
stfi_op             :
                        STFI 24-bit number
sti_op              :
                        STI 24-bit number
stps_op             :
                        STPS
wait_op             :
                        WAIT
capture_mask_flag   :   /* empty */
                        capture_mask_flag  CTV
                        capture_mask_flag   MTV
                        capture_mask_flag   MATCH

vectors_and_waveforms   :   /* empty */
```

```
                        vectors_and_waveforms vector
                        vectors_and_waveforms waveform
vector                  :
                        vector_declaration '{' vector_data '}'
vector_declaration      :
                        Vector
                       V
vector_data             :
                        vector_datum
                         vector_data vector_datum
vector_datum             :
                        pin_name '=' vector-value ';'
                         pin_name '=' identifier ';'
waveform                :
                        waveform_declaration '{' waveform_data '}'

waveform_declaration     :
                        Waveform
                       W
waveform_data            :
                        waveform_datum
                         waveform_data waveform_datum
waveform_datum           :
                        waveform-table-pin-group-name '=' identifier ';'
pin_name                :
                        identifier
vendor_sections         :
                        vendor_sections vendor_section {}
                         vendor_section {}
vendor_section          :
                        VendorSection '{' vendor_section_contents '}'
subr_header             :
                        SubrPattern
subr_section            :
                        CommonSection '{' common_contents source_selection_t
able subr_section_domains '}'
                        CommonSection '{' common_contents subr_section_domai
ns '}'
subr_section_domains    :
                        subr_section_domains subr_section_domain
                         subr_section_domain
subr_section_domain     :
                        Domain domain_name '{' subr_section_contents '}'
source_selection_table  :
                        SourceSelectionTable '{' source_selector_definitions
 '}'
source_selector_definitions:
                        source_selector_definitions source_selector_definiti
on
                         source_selector_definition
```

```
source_selector_definition:
                        SourceSelector source_selector_name '{' source_mappi
ngs '}'
source_selector_name    :
                        identifier
source_mappings         :
                        source_mappings source_mapping
                         source_mapping
source_mapping          :                                                    10
                        pin_name '=' source ';'
source                  :
                         MAIN
             INVERT_MAIN
             SUBR
             INVERT_SUBR
subr_section_contents   :
                        subr_section_contents subr_section_content
                         subr_section_content
subr_section_content    :                                                    20
                        label_spec subr_pattern_spec
                         subr_pattern_spec
subr_pattern_spec       :
                        subr_operation capture_mask_flag '{' vectors_and_wav
eforms '}'
subr_operation          :   /* empty */
                         common_operation
                         rtn_op
                         stss_op
/*                                                                           30
 *   Instructions specific to the SUBR Patterns
 */
rtn_op                :
                        RTN
stss_op                :
                        STSS identifier
```

【０５４４】

上で用いられた未定義の非ターミナルの説明は以下の通りである。

【０５４５】

１．　version-identifier：セット[0-9]からの一つ以上の文字の列であり、最初の文字は数字でなければならない。

【０５４６】

２．　identifier：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z_]からのものでなければならない。

【０５４７】

３．　vendor-section-contents：ベンダ特有のコンパイラにとってのみ意味のある任意のテキスト。

【０５４８】

４．　file-name：有効なウィンドウズのファイル名（ファイル名に空白が含まれる時には二重引用符で挟まなければならない）。なお、これは単なるファイル名である、つま

りディレクトリコンポーネントをもつべきではない。
【０５４９】
５．　waveform-table-pin-group0name：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z_]からのものでなければならない。この変数は、どこかで宣言され、ピンのグループに共通の波形テーブルの名前を保持する。
【０５５０】
６．　24-bit number：最大16777215までの有効な十進数。
【０５５１】
７．　8-bit number：最大256までの有効な十進数。
【０５５２】
８．　index-register：有効な十進数。モジュールの一実施形態では、これは[1-8]の値を有することができる。
【０５５３】
９．　vector：これは、STILにおけるVector命令文と類似している。これは信号名および信号グループ名を参照しており、コンパイラがピン記述ファイルへのアクセスを有することを必要とすることに留意されたい。
【０５５４】
１０．waveform-time-reference：セット[a-zA-Z_0-9]からの一つ以上の文字の列であり、最初の文字はセット[a-zA-Z_]からのものでなければならない。
【０５５５】
パターンファイルはコメントをサポートする。コメントはパターンファイルコンパイラによって無視されることを意味する。コメントは「#」の文字で始まり、行末まで続く。
【０５５６】
パターンファイルのヘッダおよび［共通］セクションにおけるコンストラクトを参照して、以下の点に留意すべきである。
【０５５７】
１．　pattern-nameの項目は、パターンファイルがそれに関するデータを含んでいるようなパターンオブジェクトに関連付けられる名前を指定する。これは、対応するパターンオブジェクトメタファイルにおいてヘッダに繰り越される。
【０５５８】
２．　wavaform-time-referenceは、タイミングファイルにおいて、パターンファイルの外部で定義される特定の波形およびタイミング定義についての名前である。パターンファイルにおけるwavaform-time-referenceの指定は、その特定の名前を、他のwavaform-time-referenceが現れるまで、全てのそれ以降のベクトルに結びつける。
【０５５９】
３．　サブルーチンコール（例えばJSRおよびJSRC）のオペランドは、同一のパターンファイル内で前に登場したpattern-spec labelであるか、外部定義のサブルーチンパターンにおけるpattern-spec labelである。このオペランドは、最後にはサブルーチンをロードする／扱う目的で解かれる。サブルーチンコールのオペランドについてのラベルは、システムにわたって固有であることが要求される。
【０５６０】
wavaform-time-reference名は構文的に正しいどのようなものであってもよいが、具体的なハードウェアの影響のために、waveform-time-reference名は、先に知られているよく定義されたセット（追加された読み易さのために、オプション的に、ユーザによってユーザが選んだ名前にマップされることができ、マッピングがオプションファイルに提示される）に制限される必要がある。
【０５６１】
また、パターンおよびwaveform-time-referenceソースファイルは、物理的なテスタチャネルへの接続を有する全てのDUTチャネルについて初期コンフィギュレーションデータを提供すべきであることにも留意されたい。もし後のデータがいずれかのDUTチャネルに

ついて省略されると、パターンコンパイラは、初期レベルからの出力を保持するためにパターンデータを「詰める」。

パターンファイル例

【０５６２】

MAINパターンソースファイルの単純な例は、使用法を示す助けとなるであろう。

```
#
# Filename : good1.pat
#
Version 1.0 ;
#-----------------------------------------------------------------
# Main Pattern definition:
#-----------------------------------------------------------------
MainPattern good1
{
  CommonSection
  {
    MacroDef        defaultDataVal      (XXXXXXXX)
    MacroDef        nopInstr            (NOP)
    MacroDef        label1              (Label1 : )
    MacroDef        jniInst             (JNI)

    #-----------------------------------------------------------------

    #    Timing Specifications
    #-----------------------------------------------------------------
        Timing "productionTiming.tim";
        TimingMap "productionTimingOpenSTARMap.tmap";

    #-----------------------------------------------------------------
    #    Default Domain Cycles
    #-----------------------------------------------------------------
    Domain default
    {
      #-----------------------------------------------------------------
      #label: instruction  {Vector/Waveform Data}
      #-----------------------------------------------------------------
              NOP                 { V { DATA = $defaultDataVal; CLK = 1;} W { D
ATA = wfs1; CLK = wfs1; } }
              JAL     myAPG       { V { DATA = 00000000; } }
              JSC     mySCAN      { V { DATA = 10101010; } }
              JSRC    mySubroutine { V { DATA = 01010101; } }
              JSR     myAPG       { V { DATA = 00110011; } }
              STI     100         { }
    labZero:  NOP                 { V { DATA = 00000011; } }
              JNI     labZero     { V { DATA = 11111100; } }
              IDXI    3000        { V { DATA = 10101010; } }
              IDXIn   3           { V { DATA = 01010101; } }
    $label1   NOP                 { V { DATA = $defaultDataVal; } }
```

```
                IDXI    2000    { V { DATA = 10101010; } }
                NOP             { }
                EXIT            { V { DATA = LLHHLLHH; } }
    }
      }
  }
```

【０５６３】

SUBROUTINEパターンソースファイルを示す他の例が以下に示される。

```
#-----------------------------------------------------------------
# Subroutine Pattern mySubrPat1 definition:
#-----------------------------------------------------------------
SubrPattern mySubrPat1
{

  CommonSection
  {
    #-------------------------------------------------------------
    #    Timing Specifications
    #-------------------------------------------------------------
    Timing "productionTiming.tim";
    TimingMap "productionTimingOpenSTARMap.tmap";

    #-------------------------------------------------------------
    #    Source Selection Specifications
    #-------------------------------------------------------------
      SourceSelectionTable
     {
      SourceSelector SrcSelDef
       {
            DATA=SUBR; CLK=SUBR; DATA=SUBR;
        }
        SourceSelector SrcSelOne
        {
            DATA=MAIN; CLK=MAIN;
        }
     }

    #-------------------------------------------------------------
    #    Default Domain Cycles
    #-------------------------------------------------------------
    Domain default
    {
    #-------------------------------------------------------------
    #label :    instruction    { Vector and Waveform Data setups }
    #-------------------------------------------------------------
                STI     100     { Vector { DATA = 00000000; } }
                IDXI    3000    { Vector { DATA = 00001111; } }
                IDXIn   3       { Vector { DATA = 00110011; } }
        $label1 NOP             { Vector { DATA = LLHHLLHH; } }
```

```
                NOP             { Vector { DATA = LLXXXXXX; } }
                NOP             { Vector { DATA = LLHHXXXX; } }
                JNI Label1      { Vector { DATA = LLHHLLHH; } }
                STSS SrcSelOne  { Vector { DATA = LHLHLHLH; } }
                RTN             { Vector { DATA = LLXXXXXX; } }
    }
  }
}
```

【０５６４】

パターンソースファイルにおけるメインヘッダおよび共通セクションからの要約の情報は、オブジェクトメタファイルのメインヘッダに記憶される。要約は、アドレス等の予めロードされた解決を助けるための、あるいはデータロギングを助けるための素早い抽出のために典型的には必要とされる情報からなり、共通セクションのセマンティクスは、全てのコンパイラについて正確に同じであるので、どのコンパイラも同じ要約情報を提供することができ、メタファイルを書く第一のコンパイラがこの情報を記憶する。以下は、記憶される情報である。

１．　パターンソースファイル名
２．　ソースファイルで宣言されたパターンのタイプ
３．　ソースファイルからのバージョン情報
４．　パターンソースファイルの共通セクションにおいてもちいられる波形およびタイミング名の全てのリスト
５．　パターンソースファイルの共通セクションにおける（関連の）ベクトルアドレスへのサブルーチン参照の全てのマップ
６．　パターンソースファイルの共通セクションにおける（関連の）ベクトルアドレスへのラベル参照の全てのマップ
７．　一般的なブックキーピング（bookkeeping）情報：ベクトルカウント、インストラクションカウント等

【０５６５】

オープンアーキテクチャのテストシステムは、明示的で異なる拡張子を有するためにパターンファイルおよびパターンリストファイルの両方を必要とする。パターンファイルについて、これは、プレインテキストソースファイルおよびコンパイルされたオブジェクトファイルの両方に適合する。これは、ディレクトリの一覧等において視覚的にファイルタイプを素早く識別するために、ならびに拡張子に基づいての関連付けを可能にするためにユーザにとっては便利なものとしてみられる。パターンリストファイル解析部は、これらの拡張子を有するファイル名を待つ。

プレインテキストパターンソースファイル：　.pat
コンパイルされたパターンオブジェクトメタファイル：　.pobj
パターンリストファイル：　.plst

【０５６６】

ユーザはこれらのデフォルトの値を、例えばテスタ環境変数あるいはセットアップオプションを通じて無効にすることができる。

テスタは、以下の［環境］変数の定義を

```
Tester_PATLIST_PATH：    パターンリストファイルについて
Tester_PATSRC_PATH：        パターンソースファイルについて（オプション）
Tester_PATOBJ_PATH：     パターンオブジェクトメタファイルについて
```

【０５６７】

において記述される環境コンフィギュレーションファイルの少なくとも一つにおけるファイルサーチパスについて必要とする。

【０５６８】
もしオプション的な環境／セットアップ変数Tester_PATSRC_PATHが定義されなければ、それはTester_PATOBJ_PATHと同じであると仮定される。一般的に、Tester_PATSRC_PATHを定義しないことは、それをTester_PATOBJ_PATHと同じ値で定義するよりもより効率的である。

Ｅ６．　ソフトウェア表現
【０５６９】
パターンオブジェクトはユーザによって作成されるのではなく、ユーザは常に、他のパターンリストおよび／あるいはパターンの集合体であるパターンリストオブジェクトを扱う。パターンリストオブジェクトは、それに含まれるパターンオブジェクトを、ユーザがアクセス可能な状態にしつつ、作成し、所有し、保持する。ユーザのテストプログラムにおけるパターンリストオブジェクトは、パターンリストの実際の定義を含んでいる、ディスク上のパターンリストファイルに関連付けられている。パターンリストの定義は、パターンリストに明示的な名前を与え、パターンの順序付けられたリストおよび／あるいは他のパターンリストをファイル名の関連付けを通じて識別する。このセクションは、パターンリストおよびパターンのソフトウェア表現を、それらがどのようにテスタフレームワークにおいて操作されるかについての理解に対する除外として、説明する。

パターンリストの関連付け
【０５７０】
テストシステムにおける単一のテストサイト（および、拡張によって、その中のテストプラン）は、複数の最上位レベルのパターンリストに関連付けられる。しかし、任意の時点ではテストプランについて単一の実行コンテキスト（context）のみがある。最上位レベルのパターンリストは、それによって参照される（階層的に）パターンについての実行シーケンスを定義するので、アクティブな実行コンテキストは、現在選択されている最上位レベルのパターンリストに対応するものである。なお、これは単一のパターンリストに含まれるパターンのみが一度にハードウェア上にロードされ得るということを暗に示しているのではない。むしろ実行シーケンスを実行可能にするためにハードウェア上にロードされることが要求されるパターンのセットは常に、現在ロードされているパターンの全てのサブセットでなければならない。

パターンツリー
【０５７１】
直感的に、最上位レベルのパターンリストを表現するための方法は、何かしらのツリーデータ構造によってであると感じられる。図７は、発明の順番付けられたパターンツリーの一実施形態を示しており、パターンリストＡが最上位レベルのパターンリストであると仮定している。

パターンツリー情報の内容
【０５７２】
以下の情報が、パターンツリーのどのノードにも記憶される。
【０５７３】
１．　そのノードに関連付けられている構成要素（パターンリストあるいはパターン）の名前。
【０５７４】
２．　定義ソースのタイプ。リーフ（パターンノード）については、これは常にパターンファイルであり、中間（パターンリスト）ノードについては、「最上位レベルファイル」（最上位レベルパターンリスト定義に関して）か、あるいは「ファイルに埋め込まれている」（繰り込まれたパターンリスト定義に関して）であり得る。

【０５７５】
３．　ノードが関連付けられているディスク上のファイルのタイムスタンプの最後の修正。
【０５７６】
以下の追加的な情報は、中間（パターンリスト）ノードにおいてのみ記憶される。
【０５７７】
１．　そのノードによって表されるパターンリストオブジェクト上に設定された実行オプション（もしあれば）――すなわち、そのオブジェクトオプション。
【０５７８】
２．　そのノードによって表されるパターンリスト内部のそれぞれの子参照上に設定された実行オプション（もしあれば）――すなわち、その子のそれぞれについての参照オプション。
【０５７９】
したがって、ルートからある中間ノードまでの固有のパス上に出て来るノードの集合体、ならびにそれらが出て来るシーケンスは、そのノードによって表される組み合わされた、有効な実行オプションを決定するために必要な全ての情報を含んでいる。パターンの実行オプションは、その中間の親がそれについて有しているかもしれない参照オプションと組み合わされて、その中間の親の有効な実行オプションによって決定される。
【０５８０】
ここで、パターンリスト解析部がパターンツリーを作成するプロセスにある間、ある実行オプションは単に文字列として値の初期記憶を要求するかもしれない。なぜなら、それらの使用のコンテキストは、後で解決されないかもしれないからである。このようなオプションの一例は、「mask」オプションであり、これは、パターンリストがソケット情報に関連付けられていないピンマスク情報を指定し、したがって、ピンマスクオプション（ピンおよびグループ名）は、ロードされるのに先立って解決されるべく、文字列として記憶される。
【０５８１】
以下の追加的な情報は、リーフ（パターン）ノードにおいてのみ記憶される。
【０５８２】
１．そのパターンによって呼び出される全ての（おそらく過渡的な）参照。外部参照、内部参照の両方ともであり、実行ツリーとして組織化されている。
【０５８３】
当然のことながら、全てのパターンノードは、オブジェクトメタファイル共通ヘッダにおいて利用可能である全てのパターンファイル要約情報に対するアクセスをさらに有しており、それをキャッシュに格納することを選択する可能性もある。

パターンリストの修正の扱い
【０５８４】
パターンリストの中身への変更は、概念的には、そのパターンリストへの全ての参照に影響する。次のルールがこのような変更を管理するために用いられる。これらのルールは、パターンオブジェクトならびにパターンリストオブジェクトにたいして適切なものとして適合する。
【０５８５】
１．　ディスク上のパターンリストファイルの中身に対する変更は、そのパターンリスト上（あるいはそれを参照している他のパターンリスト上）で実行されているload()コマンド上でのみテストシステムを通じて伝播する。言い換えると、ソフトウェアにおけるパターンリストの階層は常に、ハードウェア上に現在ロードされているそれを反映する。
【０５８６】
２．　ユーザは、ディスクファイルソースにパターンリストを同期させるために、ロード時間中になされたチェックを無効にするモードを設定することができる。これにより、

プロダクションモードにおいてより素早く安全な動作が可能である。

パターンツリーナビゲーション

【０５８７】

テストサイトに（および、拡張によって、そのサイトに関するテストプランに）関連付けられている最上位レベルのパターンリストは、パブリック（グローバル）な範囲を有する。システムは、ユーザが個々のノードおよびサブツリーにアクセスすることができるように、最上位レベルのパターンリストを表すパターンツリーをナビゲートするようにAPIを提供する。

Ｅ７．　パターンリストダイナミクス

【０５８８】

先に、パターンリストの静的なルールを説明した。ここではパターンリストの動的な（実行）ルールを説明する。

【０５８９】

パターンツリーは、一般的なパターン管理に必要不可欠である。例えば、パターンロードシーケンスの開始点は、サイトあるいはテストプランに現在関連付けられているパターンツリー上へのload()方法のコールである。しかしながら、パターンツリーは孤立しては動作しない。完全に初期化されたパターンツリーは、以下の２つのフレームワークオブジェクトを作成するように用いられる。

【０５９０】

１．　最上位レベルのパターンリストは、パターンについてパターン実行シーケンス（Pattern Execution Sequence）を定義する。これは、どのようにこのような実行シーケンスがその最上位レベルのパターンリストに対応するパターンツリーから得られるかを述べるものである。例えば、図７に示すパターンツリーＡに対応するパターン実行シーケンスは、{q，s，t，q，r，q，u，u，v}である。パターン実行シーケンスは概念的には、パターンツリーを通じて述べられた実効シーケンスを反映する順序付けられたリストである。フレームワークは、パターンツリーノードとパターン実行シーケンス内の対応するエントリとの間のいかなる必要なナビゲーションリンクをも、確立して保持する。

【０５９１】

２．　パターンセット、これは単に、パターンツリー内の固有のパターン（サブルーチンを含む）の全てのリストである。したがって、これは、ハードウェア上にロードされるべき個々のパターンを決定するために用いられるリストである。フレームワークは、パターンツリーノードとパターンセットにおける対応するエントリとの間のいかなる必要なナビゲーションリンクをも確立して保持する。図７のパターンツリーについてのパターンセットは、(q，s，t，r，u，v)である（パターンリストＡにおけるパターンのどれも、サブルーチンコールを含まないものと仮定する）。

【０５９２】

パターン実行シーケンスおよびパターンセットは両方とも、常にパターンツリーから得ることができるということに留意されたい。しかし、初期の構成の後に、実行可能である限り、これらをキャッシュに格納することはしばしば意味がある。

パターンリスト実行オプション

【０５９３】

上で示したように、各パターンリスト宣言（その定義に先立つ）あるいはパターンリスト／パターン参照エントリの後に、いくつかの実行オプションを続けることができる。パターンリスト実行オプションは、パターンリストのランタイム実行を修正する。将来の拡張を許すために、これらのオプションの名前（およびオプション的な値）は、適切である特定のバージョンによって解釈されるべく、パターンコンパイラのパターンリストファイル解析部によっては単に文字列として扱われる。テスタは、以下に述べるように、オプシ

ョンのセットおよびそれらの解釈を指示する。しかしながら、ベンダは、そのオプションセットを拡張することができる。オプション構文の解析時の妥当性確認を可能にするために、パターンリスト解析部は、特定のバージョンについての情報ファイルを読むことができるであろう。また、このような情報ファイルは、特定のバージョンがそもそも実行オプションの仕様をサポートしているかどうかを指定するためにも用いられる。

【０５９４】

実行オプションのセットをサポートしているバージョンに関して、以下の一般的なルールがそれらの使用を支配する。これらのルールを理解するためには、順序付けられたツリーとしてパターンリスト／パターンの階層的な集合体を視覚化することが有用である。

【０５９５】

１．　パターンリスト定義上（すなわちファイル内の「ローカルパターンリスト宣言、グローバルパターンリスト宣言」の生成物内に設定されたイントリンシックオプションは、事実上、ユーザのテストプログラムにおける対応するパターンリストオブジェクト上での直接的なオプション設定である。したがって、これらは、そのパターンリストオブジェクトへの全ての参照に適合し、オブジェクトオプションとして参照される。

【０５９６】

２．　パターンリスト／パターンへの参照上（すなわち、ファイルにおける「パターンエントリ」および「パターンリストエントリ」生成物内）に設定されたリファレンシャルオプションは、このオプションの範囲を階層中の具体的なパス、つまりツリーのルートから検討中の参照まで至る（パターンリスト／パターンの宣言順によって確立される）パスに限定する。したがって、これらは具体的なオブジェクト参照上（そしてオブジェクト自体の上ではない）のオプションであり、参照オプションとして参照される。

【０５９７】

３．　集合体の階層におけるいかなるリスト／パターンについての有効オプション設定（パターンリスト／パターンの宣言順によって確立される）は、ツリーのルートからそのリスト／パターンまでのパスに沿って出て来るオブジェクトオプションと参照オプションとの組み合わせである。具体的な組み合わせメカニズム（例えば、セット合併、セット交差あるいは、他のいかなる競合解決アルゴリズム）はそのオプション自体のプロパティである。

【０５９８】

上記ルールの結果、ならびに、パターンファイル内のパターン定義上に実行オプションを設定するための機構はないという事実は、パターンへの全ての参照に適合するオプションを設定するための直接的なルールはないということである。これを実現するためのメカニズムは、単一パターンパターンリストを用いることである。

【０５９９】

テスタは、そのバースト動作（burst behavior）を修正し、その実行シーケンスを修正するパターンリスト実行オプションのあるセットを指定する。

【０６００】

パターンリストについての実行シーケンスがハードウェアに提出されると、ハードウェアはバーストを作り出す。バーストは、ソフトウェアからのどのような関わりなしでの、直接ハードウェアによってのパターンのシーケンスの実行である。バーストの不連続は実行シーケンスにおける、前のバーストが終わる位置であり、新しいバーストが始まる。

【０６０１】

パターン管理ソフトウェアの目的の一つは、ハードウェアに、それがバーストを作り出すことを必要とするような実行シーケンスを提供することである。デフォルトによって、パターンツリーは、実行シーケンスを生み出し、もしこれがハードウェアに提出されれば、単一のバーストを引き起こす。しかしながらこの動作は、パターンリスト上でオプションを使用することによって修正され得る。したがって、オプション結果の使用は、バーストの不連続性という結果になる。

【０６０２】

さらに、ユーザは、どのパターンあるいはどのバーストの前にも、あるいは後にも、プロローグパターンまたはエピローグパターンを動作させることを時折要求する。これは、ハードウェアに提出されるべき実行シーケンスを修正する。

【０６０３】

パターン実行シーケンスオブジェクトの作成あるいは修正の間、フレームワークは、指定された実行オプションとパターンツリーによって具体化された特定の実行シーケンスとの組み合わせから得られるパターンバーストにおける中断を判断し、必要であれば報告するために必要な情報の全てを有している。これをしている間、それは、システムにおけるモジュールのハードウェア能力を調査する必要があるかもしれない。例えば、あるハードウェアインプリメンテーションは、ピンマスクについて４つの記憶されたコンフィギュレーションを可能にし、そのうちの２つ（０および３）はデフォルトのマスクされた（Mask This Vecor, MTVをサポートするために）、およびマスクされていない動作に対して用いられる。したがって、ユーザは、バーストモードを中断することなく、２つの異なるグローバルピンマスクコンフィギュレーションを許される。

【０６０４】

なお、もしモジュールベンダがハードウェアにおけるパターンリストインプリメンテーションをサポートしていなければ、パターン実行シーケンスのベンダによる処理は、実行シーケンスにおける全てのパターンの個別の実行という結果になるであろう。サイト互換のシステムおよびサイト－ヘテロジニアスなシステムの両方において、サイトのバースト能力は、「最小公分母」によって制限される。テスタはオプションのあるデフォルトセットを提供し、それらのパラメータは以下に説明される。それぞれのオプションは、以下を述べることによって指定される。

【０６０５】

それがイントリンシックである（すなわち、グローバルあるいはローカルなキーワードを有する定義に関連付けられている）か、あるいはリファレンシャルである（すなわち、PatあるいはPListキーワードを有する参照に関連付け　られている）か。イントリンシックなオプションは定義の時点およびどの参照でも当てはまるが、リファレンシャルオプションはそれらが関連付けられている参照でしか適合しない。

【０６０６】

さらに、オプションは、そのオプションが全て静的に（構文的に）あるいは動的に（参照されることによって意味的に）、繰り込まれたパターンあるいはパターンリストに再帰的に適合すると仮定されれば、子（Children）によって受け継がれると考えられる。

【０６０７】

以下は、オプションのリストである。どの対応（コンプライアント）ベンダも、指定されたこれらのオプションを解釈する。

１．　Mask<pin/pin group>

【０６０８】

GlobalPList、LocalPListに適合したときにはイントリンシック。

【０６０９】

PList、Patに適合したときにはレファレンシャル。

【０６１０】

子によって受け継がれる。

【０６１１】

このパターンリストは常に、無効とされている示されたピンあるいはピングループによって参照されるピンの比較回路を有している。ハードウェアの制限がバーストの不連続という結果をもたらすことがある。

２．　BurstOff

【０６１２】

GlobalPList、LocalPListに適合したときにはイントリンシック。

【０６１３】

PList、Patに適合したときにはレファレンシャル。
【０６１４】
子によって受け継がれない。
【０６１５】
このパターンリストは常に非バーストモードで実行される。このオプションは子によって受け継がれないが、BurstOffDeepオプション（下記）は子によって受け継がれる。
３．　BurstOffDeep
【０６１６】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６１７】
PList、Patに適合したときにはレファレンシャル。
【０６１８】
子によって受け継がれる。
【０６１９】
このパターンリストは常に非バーストモードで実行される。このオプションは子によって受け継がれるが、BurstOffオプション（上記）は子によって受け継がれない。なお、BurstOffDeepオプションは子によってオフにされることができない。
４．　PreBurst<pattern>
【０６２０】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６２１】
指定されたバーストオプションを有していない子ノードによってのみ受け継がれる。
【０６２２】
示されたパターンは、このパターンリスト内の全てのバーストの前につけられるべきである。PreBurstパターンは、このパターンリストノードのせいで開始されるどのバーストの直前でも起こる。このオプションは、同じパターンであるPreBurstオプションを有するバースト内に既にあるときには適用されない。
５．　PostBurst<pattern>
【０６２３】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６２４】
指定されたバーストオプションを有していない子ノードによってのみ受け継がれる。
【０６２５】
示されたパターンは、このパターンリスト内の全てのバーストの後につけられるべきである。PostBurstパターンは、このパターンリストノードのせいで開始されるどのバーストの直後でも起こる。このオプションは、同じパターンであるPostBurstオプションを有するバースト内に既にあるときには適用されない。
６．　PrePattern<pattern>
【０６２６】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６２７】
子によって受け継がれない。
【０６２８】
示されたパターンは、このパターンリスト内の全てのパターンの前につけられるべきである。
７．　PostPattern<pattern>
【０６２９】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６３０】
子によって受け継がれない。

【０６３１】
示されたパターンは、このパターンリスト内の全てのパターンの後につけられるべきである。
８．　Alpg<alpg object name>
【０６３２】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６３３】
子によって受け継がれない。
【０６３４】
名前つきのALPGオブジェクトは低速APGレジスタ設定、読み出し待ち時間（latency）、即値データ（immediate data）レジスタ、アドレススクランブル、データ反転、データジェネレータ等の関連情報を記憶する。
９．　StartPattern<pattern>
【０６３５】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６３６】
子によって受け継がれない。
【０６３７】
パターンリストは、その実行シーケンスにおいてStartPatternが最初に出現したときに実行開始する。
１０．StopPattern<pattern>
【０６３８】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６３９】
子によって受け継がれない。
【０６４０】
パターンリストは、その実行シーケンスにおいてStopPatternが最初に出現したときに実行を停止する。
１１．StartAddr<vector offset or label>
【０６４１】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６４２】
子によって受け継がれない。
【０６４３】
これはStartPatternオプションを伴わなければならない。パターンリストは、その実行シーケンスにおいてStartPatternの最初の出現におけるStartAddrのときに実行開始する。
１２．StopAddr<vector offset or label>
【０６４４】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６４５】
子によって受け継がれない。
【０６４６】
これはStopPatternオプションを伴わなければならない。パターンリストは、その実行シーケンスにおいてStopPatternの最初の出現におけるStopAddrのときに実行を停止する。
１３．EnableCompare_StartPattern<pattern>
【０６４７】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６４８】

子によって受け継がれない。
【０６４９】
示されたパターンの最初の出現時にパターン比較が始まる。
１４．EnableCompare_StartAddr, EnableCompare_StartCycle
【０６５０】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６５１】
子によって受け継がれない。
【０６５２】
これはEnableCompare_StartPatternを伴わなければならない。パターン比較が始まるべき、パターン内のアドレスまたはサイクルを示す。
１５．EnableCompare_StopPattern<pattern>
【０６５３】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６５４】
子によって受け継がれない。
【０６５５】
示されたパターンの最初の出現時にパターン比較が終わる。
１６．EnableCompare_StopAddr, EnableCompare_StopCycle
【０６５６】
GlobalPList、LocalPListに適合したときにはイントリンシック。
【０６５７】
子によって受け継がれない。
【０６５８】
これはEnableCompare_StopPatternを伴わなければならない。パターン比較が終わるべき、パターン内のアドレスまたはサイクルを示す。
１７．Skip
【０６５９】
PList、Patに適合したときにはレファレンシャル。
【０６６０】
子によって受け継がれない。
【０６６１】
パターンリストによって支配されるパターンあるいはシーケンス全体をスキップさせる。またこれは、このパターンリストサブツリーのルートにおけるすべてのオプションもスキップさせる。実行目的ではこのパターンサブツリーがそこにないかのようである。

パターンリストバースト制御
【０６６２】
先に述べたように、パターンリストのための実行シーケンスがハードウェアに提出されるときに、ハードウェアは、ソフトウェアの関与なしに、パターンシーケンスのバーストを作り出す。バーストの不連続は、前のバーストが終わる実行シーケンスにおける位置であり、新しいバーストが始まる。PreBurst、PostBurst、BurstOffおよびBurstOffDeepオプションは、上記オプションリストで説明したように、どこでバーストの不連続が起こるかを制御する。PreBurstおよびPostBurstオプションは、以下に説明するある追加のルールに従ってバーストの不連続を決定する。
【０６６３】
１．　パターンリストがPreBurstおよびPostBurstオプションを有し、繰り込まれたリストが同じ対応するオプションを有するときには、バーストの不連続はなく、繰り込まれたリストのPreBurstおよびPostBurstオプションは当てはまらない。パターンリストのPreBurstおよびPostBurstを適用するたった一つのバーストがあるのみである。

【０６６４】
２．　なお、繰り込まれたリストがバーストオプションを有していなければ、それは、、これらのオプションの記述によって親リストと同じPreBurstおよびPostBurstオプションを有していることと等価である。したがって、バーストオプションをもたない繰り込まれたリストは、バーストの不連続をもたらさない。
【０６６５】
３．　もし上のルール１が当てはまらず、親リストの開始から繰り込まれたリストの開始までのパターン実行シーケンスに対する寄与があれば、繰り込まれたリストの開始時にバーストの不連続がある。この場合、親リストのPreBurstおよびPostBurstは、親リストからのパターン実行シーケンスへのこの寄与に当てはまる。繰り込まれたリストのPreBurstおよびPostBurstは繰り込まれたリストに適合する。
【０６６６】
４．　もし上のルール１が当てはまらず、繰り込まれたリストの終わりから親リストの終わりまでのパターン実行シーケンスへの寄与があれば、繰り込まれたリストの終わりにバーストの不連続がある。この場合、親リストのPreBurstおよびPostBurstは親リストからのパターン実行シーケンスへのこの寄与に当てはまる。繰り込まれたPreBurstおよびPostBurstは繰り込まれたリストに適合する。
【０６６７】
５．　もし上のルール１が当てはまらず、繰り込まれたリスト以外の親リストからのパターン実行シーケンスへの寄与がなければ、親リストのPreBurstおよびPostBurstは当てはまらない。繰り込まれたPreBurstおよびPostBurstを適用するたった一つのバーストがあるだけである。
【０６６８】
以下に、実行シーケンスへのオプションの影響を示す二、三の例を示す。簡略化のために、全てのパターンリストは単一のファイルにおいて指定されるものと仮定する。

例１：　BurstOffの使用
【０６６９】
この例はBurstOffおよびPreBurstを示している。特に強調されるのは、BurstOffによって、パターンは、１パターンの長さのバーストにおいて単独で動作することである。したがって、PreBurstオプションもまだ当てはまる。入力パターンリストは以下の通りである。

```
Global    A    [BurstOff] [PreBurst pat_z]
{
    Pat        q;
    PList    B;
    Pat        r;
    Pat        s;

    Global C
    {
        Pat        t;
        PList    D;
    };

    PList    D;
    PList    E;
};
```

```
Global B
{
    Pat a;
    Pat b;
};


Global D [BurstOff]
{
    Pat c;
    Pat d;
};


Global E
{
    Pat e;
};
```

【０６７０】
ルートをＡに置くツリーは図８に表され得る。
【０６７１】
このパターンの実行シーケンスは以下の通りである。「｜」の文字はバースト中断を示している。このパターンリストは10個のバーストを実行し、最初のバーストはパターンｚおよびｑを有し、最後のバーストはパターンｅを有する。

ｚｑ｜ａｂ｜ｚｒ｜ｚｓ｜ｔ｜ｃ｜ｄ｜ｃ｜ｄ｜ｅ

【０６７２】
この実行シーケンスについて、以下のことに留意されたい。
【０６７３】
１．　ＡについてのBurstoffオプションはＢに受け継がれないので、Ｂにおけるパターンａおよびｂはバーストとして動作する。
【０６７４】
２．　ＡについてのPreBurstオプションはBに受け継がれないので、Ｂによるバーストにおけるａおよびｂはｚによって前には付けられない。
【０６７５】
３．　ｚによるプリフィクスは、ａの直接の子であることによって実行されるパターン、すなわちパターンｑ、ｒおよびｓについてのみ起こる。これらのパターンは、ＡがBurstOffオプションを有するために、バーストにおいて１パターンの長さであるかのように単独で実行される。BurstOffは、パターンが１パターンの長さのバーストにおいて個々に動作されることを要求する。したがって、PreBurstおよびPostBurstオプションは依然として当てはまる。
【０６７６】
４．　パターンリストＩＤは、その子ｃおよびｄを単独で実行させるイントリンシックバーストオフオプションを有している。それらはＡからPreBurstｚを引き継がない。

例２：　BurstOffDeepの使用
【０６７７】
この例はBurstOffDeepオプションを示している。パターンリスト定義中のBurstOffDeepは繰り込まれた定義および参照リストに影響する。しかしながら、PreBurstおよびPostBu

rstオプションは繰り込まれ、参照されているリストによっては受け継がれない。この例は例１におけるのと同じパターンＡ、Ｂ、Ｃ、Ｄ、Ｅを用いているが、オプションは異なる。

５．　Ａの定義についてのオプション：[BurstOffDeep]、[PreBurst z]、[PostBurst y]

６．　他のどのノードについても他のオプションはない

【０６７８】

実行シーケンスは以下の通りである。先に述べたように、「｜」の文字はバースト中断である。

ｚｑｙ｜ａ｜ｂ｜ｚｒｙ｜ｚｓｙ｜ｔ｜ｃ｜ｄ｜ｃ｜ｄ｜ｅ

【０６７９】

この実行シーケンスについては以下の点に留意されたい。

【０６８０】

１．　PreBurstおよびPostBurstはB、C、D、Eによって受け継がれない。

【０６８１】

２．　BurstOffDeepはB、C、DおよびEによって受け継がれる。

例３：　PreBurstおよびPostBurst抑制

【０６８２】

ここで例１のパターンリストツリーを考えることにする。オプションは

１．　Ａの定義についてのオプション：[PreBurst x]、[PostBurst y]

２．　Ｃの定義についてのオプション：[PreBurst x]、[PostBurst z]

３．　他のどのノードについても他のオプションはない

実行シーケンスは

ｘｑａｂｒｓｔｃｄｃｄｅｙ

【０６８３】

となる。

【０６８４】

「ｔｃｄ」サブシーケンスが「ｘｔｃｄｚ」ではない理由は以下の通りである。

【０６８５】

１．　最初のｘは、実施されている現在のバーストに関連付けられているプレバーストオプションｘに等しいので、抑制される。

２．　最後のｚは、PostBurstｚはDには受け継がれなず、かつｚが加えられ得るCから生成されるパターンはないので、抑制される

例４：　Skipの使用

【０６８６】

この例は、繰り込まれた定義および参照されるリストへのSkipオプションの影響を示す。この例は、例１と同じパターンＡ、Ｂ、Ｃ、Ｄ、Ｅを用いているが、オプションは異なる。

１．　Ａの定義についてのオプション：[Skip]、[PreBurst z]、[PostBurst y]

２．　ｒへの参照についてのオプション：[Skip]

３．　Cの定義についてのオプション：[Skip]

【０６８７】

実行シーケンスは、以下に示すような中断のない単一のバーストである。

ｚｑａｂｓｃｄｅｙ

【０６８８】

この実行シーケンスについて以下の点に留意されたい。
【０６８９】
１．　ｒおよびＣについてのノードはスキップされる。
【０６９０】
２．　バーストの中断は全くない。

例５：　Maskの使用
【０６９１】
この例は、Maskオプションの影響とパターンおよびパターンリスト定義および参照へのその影響とを示す。この例は、例１と同じパターンＡ、Ｂ、Ｃ、Ｄ、Ｅを用いているが、オプションは異なる。
１．　Ａの定義についてのオプション：[mask pin1_pin2]、[PreBurst z]
２．　Bの定義についてのオプション：[mask pin3]
３．　Bの定義についてのオプション：[mask pin4]
４．　ｅへの参照についてのオプション：[mask pin5]
【０６９２】
５．　他のどのノードについても他のオプションはない。
【０６９３】
「pin1_pin2」という名前は、Pin1およびPin2をマスクするグループを指定している。「pin3」、「pin4」および「pin5」という名前はそれぞれPin3、Pin4およびPin5をマスクすることを指定している。実行シーケンスは以下に与えられるが、「｜」はバーストの中断を示す。各パターンの下の数字は、そのパターン実行中にマスクされるピンを示している。

```
z  q  a  b  z  r  z  s  t  c  d  c  d  |  e
1  1  1  1  1  1 1  1 1 1  1  1  1     1
2  2  2  2  2  2 2  2 2 2  2  2  2     2
            3  3                                5
            4  4
```

【０６９４】
この実行シーケンスについて以下の点に留意されたい。
【０６９５】
１．　ベンダのハードウェアは、バーストの中断なしで２個のマスクブロックを収容することができるのみである。ｅが実行されるまでは、２個のマスクブロックはピン｛１，２｝およびピン｛１，２，３，４｝である。パターンｅが異なるマスクブロックであるピン｛１，２，５｝とともに到着すると、ハードウェアはバーストの中断を要求する。

例６：　受け継がれたオプションおよび参照の使用
【０６９６】
この例は、定義において受け継がれたオプションは、その定義が参照されるときには当てはまらないということを示す。以下の例を考える。

```
Global A
{
    Global B [BurstOffDeep]
    {
        Global C
        {
            ...
        };
```

```
        ...
    };
   ...

   PList C;
};

Global D
{
    PList C;
};
```

【０６９７】
BurstOffDeepオプションは、定義の時点ではＣによって受け継がれる。しかしそれはイントリンシックなオプションではなく、したがってその参照時点ではＣには適用されない。

例７：　繰り込まれたリストを有するPreBurstおよびPostBurst

【０６９８】
以下の例を考える。

```
GlobalPList A [PreBurst x] [PostBurst y]
{
    Pat p1;

    LocalPList B [PreBurst x] [PostBurst y]
    {
        Pat p2;
    }

                    LocalPList C
                    {
                        Pat p3;
                    }

                    LocalPList D [PreBurst x] [PostBurst z]
                    {
                        Pat p4;
                    }

                    LocalPList E [PreBurst w] [PostBurst y]
                    {
                        Pat p5;
                    }

                    Pat p6;
}
```

実行シーケンスは

```
x  p1  p2  p3  y  |  x  p4  z  |  w  p5  y  |  x  p6  y
```

【０６９９】
である。
【０７００】
１．　パターンp2はp1と同じバースト内にある。なぜなら繰り込まれたリストのPreBurstおよびPostBurstオプションが親と同じに指定されているからである。パターンp3もまた同じバースト内にある。なぜならこれらのオプションは親と同様に引き継がれるからである。これらのオプションは、残りの繰り込まれたリストにおける少なくとも一つの異なるメンバを有し、バーストの不連続を生じさせる。

タイミング
【０７０１】
ユーザは、主として、パターンファイルを用いてテストセットアップを定義することによってシステムとやり取りを行う。タイミングファイルは、これらのパターンのタイミングを記述するために用いられる。このファイルは、定義の根底にある他のシステムファイル（例えばPin、SpecSelector）が解決されることを必要とする。さらに、タイミング定義において用いられるさまざまな変数を解決するのに用いられるSpec-Slectorおよびグローバル定義は、コンポジットTestConditionGroupオブジェクトにカプセル化される。テストプランファイルのようなより上位レベルのファイルは、このTestConditionGroupインスタンスを用いる。
【０７０２】
テストプランファイルは、TestConditionGroupオブジェクトへの参照を含んでいる。パターンソースファイルは、TimingMapオブジェクト内のWaveformSlectorコンポーネントを参照する。Timingオブジェクト自体はPinオブジェクトを参照する。オプションとして、Timingオブジェクトは、SpecSlectorオブジェクトによって調節される変数も参照し得る。これらの関係を図９に示す。
【０７０３】
Pattern-List内のPatternオブジェクトは、パターンキャラクタのセットについて用いるためのWaveformSelectorの名前を指定する。また、タイミングマップファイルはパターンにおいて指定されることに留意されたい。パターンは、このマップが変更されなければコンパイルされる必要はない。

```
Version 1.0;
MainPattern
{
    CommonSection
    {
        ...
        Timing = myGalxy.tim;
        TimingMap = myGalxyMap.tmap;
        ...
        Domain default
        {
        NOP    V    {SIG=1; CLK=1; DATA=L; } W {SIG=wfs1; FASTCLK=wfs1; }
                NOP    W    {SIG=wfs2;}
                    NOP    V    {SIG=L; }
                NOP    V    {SIG=0; }

        }
    }
}
```

【０７０４】
TestConditionGroup Fileオブジェクトは、使うべきTimingオブジェクトと使うべきTimingMapオブジェクトとをインポートする。各テストは、TimingConditionインスタンスのためのTestConditionGroupオブジェクトから得られたそのインスタンスを用いる。したがって、波形テーブルの同じセットをサポートする複数のTimingオブジェクトをテスタフレームワークに記憶することができ、必要なときにスワップすることができる。同様に複数のテストプランファイルは、共通のTestConditionGroupオブジェクトを共有することができる。

【０７０５】
テストプラン記述ファイルの一例は、タイミングオブジェクトの使用を以下に示す。

```
Import patlist1.plist;
Import tim1.tim;
Import tim2.tim;
Import tmap1.tmap;
TestConditionGroup tim1_prod
{
    SpecSet = prodTmgSpec(min, max, typ)
    {
        period = 10ns, 15ns, 12ns;
    }
   Timings
    {
        Timing = tim1;
        TimingMap = tmap1;
    }
}
TestConditionGroup tim2_prod
{
    SpecSet = prodTmgSpec(min, max, typ)
    {
        period = 10ns, 15ns, 12ns;
    }
    Timings
    {
       Timing = tim2;
        TimingMap = tmap1;
    }
}
TestCondition tim1_prod_typ
{
    TestConditionGroup = tim1_prod;
   Selector = typ;
}
TestCondition tim2_prod_max
{
    TestConditionGroup = tim2_prod;
   Selector = max;
```

```
}

Test FunctionalTest MyFunctionalTestSlow
{
    PListParam  = patlist1;
    TestConditionParam = tim1_prod_typ;
}

Test FunctionalTest MyFunctionalTestFast
{
    PListParam  = patList1;
    TestConditionParam = tim2_prod_max;
}
```

【０７０６】

「tim1」および「tim2」は、先に定義された異なるタイミングオブジェクトを用いるテストプラン内の２つのテストである。Timingオブジェクトは、ピン一つ一つにさまざまな波形を定義する。タイミングファイルおよびタイミングマップファイルで用いられるピンは、ピン記述ファイルにおいて適切に定義される必要がある。

【０７０７】

Timingオブジェクトは、waveformオブジェクト内で値を定義するためにSpecificationSetオブジェクトを用いることができる。Timingオブジェクトは、さまざまな属性についてハードコードされた値を含むことができるが、ユーザがさまざまな属性を変数を用いる値に割り当てさせるのが通常である。これらの変数はSpecificationSetオブジェクトに依存し得る。この使用法の一例を以下に示す。

```
Version 1.0;
Timing basic_functional
{
    . . .
    Pin SIG
    {
        WaveformTable wfs1
        {
                { 1 { U@t_le ; D@t_te D; Z@45ns;} }
        };
    };
    Pin CLK
    {
        WaveformTable wfs1
        {
                { 0 { U@20ns; D@40ns; }};
        };
    };
}
```

【０７０８】

変数U@t_leは、エッジ配置（placement）を定義するものであるが、これはどこかで定義され、SpecificationSetに依存している。SpecSelectorは以下に示すように定義される。

```
SpecificationSet prodTmgSpec( min, max, typ)
```

```
{
    t_le = 10ns, 14ns, 12ns;
    t_te = 30ns, 34ns, 32ns;
    ...
}
```

【０７０９】

specを変更することによって用いられるタイミングを変更することは、以下の例に示されている。

```
TestCondition prodTmp_typ
{
    TestConditionGroup = prodTmgSpec;
   SpecSelector = typ;
}

TestConditionGroup prodTmp_max
{
    TestConditionGroup = prodTmgSpec;
   SpecSelector = max;
};
```

Ｆ２．　テスタのタイミングコンポーネントへのマッピング

【０７１０】

タイミング「typ」および「max」は、SpecSelectorにおける典型的な／最大の仕様を用いる。テスタモジュールの２つのコンポーネントは、波形およびそれらに関連するタイミングの生成に直接関与する。２つのモジュールは、パターンジェネレータ（PG）およびフレームプロセッサ（FP）である。図１０に、オープンアーキテクチャのテストシステムアーキテクチャ内のフレームプロセッサによる波形フォーマットとタイミング生成とを示す単純化したブロック図が示されている。以下、波形生成を簡単に説明する。

【０７１１】

パターンジェネレータ１００２は、モジュール内の全てのピンに共通のタイミングセットを生成する。このタイミングセットはグローバルタイミングセット（GTS）と呼ばれる。パターンジェネレータをセットアップすることができるモードは３つある。これらの３つのモードは、GTSを記述するために用いることができるビットの数に影響する。また、これらのセッティングも、バンクを選択し、かつCapture This Vector（CTV）ビットおよびMask This Vector（MTV）ビットがセットされているかどうかを選択するために用いることができるビットの数に影響する。テスタにこのベクトルの結果を取り込むように指示するために、ユーザはパターンファイルにおいてCTVフラグを用いる。同様に、ユーザは、現在のベクトルの結果をマスクするようにテスタに指示するために、パターンにおいてMTVフラグを用いる。これを以下の表１に示す。パターンジェネレータ１００２は、波形キャラクタ（WFC）の生成する役目も有している。WFCはピン一つずつに生成される。テスタモジュールは、WFCを記述するために固定数のビットを用いる。

【表１】

GTS bits	GTS in a Bank	GTS Bank	CTV	MTV
8bits	256	4	NO	NO
7bits	128	8	YES	NO
6bits	64	16	YES	YES

【０７１２】
テスタモジュールは、ピンごとにフレームプロセッサ１００４を提供する。各フレームプロセッサは、タイミングセットスクランブラ（TSS）１００６を含んでおり、これがこの例においては1024までのトータルの深度を有している。TSS１００６は、先に述べ、そして図１０に示したように、パターンジェネレータのモードに依存していくつものバンク１００８をパーティションで区切り、バンクごとに６４個のエントリの１６個のバンクが用いられる。TSSは、各ピンについて波形テーブルを定義する能力においてより柔軟性を与えるように設けられる。「FP」モードにおいてはTSSは2ビットを用いてタイミングセットを出力する。したがってTSSは、ピンごとに合計で４つの物理的なタイミングセットを生成する。これらのタイミングセットは、ローカルタイミングセット（LTS）と呼ばれる。

【０７１３】
フレームプロセッサ１００４はLTSとWFCとを組み合わせて、波形メモリ１０１２およびタイミングメモリ１０１４にインデックス１０１０を作成する。「FP」モードにおいて、5ビットの値は、LTSによって生成される２ビットとWFCによって生成される３ビットとに分割さえる。したがって、物理的な波形メモリおよびタイミングメモリの深度は、最大４つの物理的タイミングセットが用いられ得るが、ピンごとに32 deepである。波形メモリは、波形を形成するイネーブルにされたタイミングエッジを含んでいる。イネーブルのエッジについてのタイミングの値は、タイミングメモリから得られる。したがって、フレームプロセッサは波形をフォーマットする。

マッピング方法
【０７１４】
この方法は、全てのWaveformTableブロックをピンごとにテスタ内のLTSにマップすることである。もしテスタのハードウェアが４つのLTSをサポートしていれば、ユーザは最大４つのWaveformTableブロックを定義することができる。各WaveformTableブロックは、テスタデジタルモジュールについて最大ｎ個の波形定義を有することができる。

【０７１５】
Timing-Mapファイルは、オープンアーキテクチャテストシステムにおけるモジュール用のWaveformTableへのTiming-Mapブロックにおいて定義されたLogical WaveformSelectorのマッピングを提供する。この場合、テスタは、256個までのLogical WaveformSelectorをサポートする。オープンアーキテクチャのテストシステムにおいては、Logical WaveformSelectorは直接GTSにマップされる。パターンコンパイラは、Timing-MapおよびTimingブロックの両方に依存し、パターンファイルをコンパイルすることができる。しかしながTimingブロックのWaveformTableにおける波形キャラクタが変更されなければ、またはTiming-MapブロックにおけるWaveformSelectorのマッピングが変更されなければ、パターンをコンパイルしなおす必要はない。

このマッピング方法を用いる例

【０７１６】

テスタのデジタルモジュールへのマッピングを示すために、以下を仮定する。フレームプロセッサはFPモードにセットされ、CTVおよびMTVビットは、GTSビットの総数が６となり、タイミングバンクセレクタのビットの総数が４となるようにセットされる。

【０７１７】

Timingブロックにおいて定義された各WaveformTableは、Timingファイル内の別個のLTSにマップされる。これはピンごとに行われる。したがって、WaveformTable seq1はLTS1にマップされる。「SIG」ピンの場合には、８個の可能な波形エントリの全てが使い切られる。しかし「CLK」ピンは単一の波形を必要とし、したがって波形メモリ（WFT）および波形タイミングメモリ（WTM）における単一の行を使い切る。

【０７１８】

「SIG」ピンの最初の２つの物理的な波形のマッピングを図１１に示す。このWaveformTableはエッジの別々のコンフィギュレーションを必要とする２つの波形キャラクタをマップするので、我々は、波形メモリ（WFT）１１１２および波形タイミングメモリ（WTM）１１１４における２つのエントリを割り当てることで終わることになる。波形の形は、WFMに記憶され、タイミングの詳細はWTMに記憶される。モジュールの一実施形態は、合計で６個のタイミングエッジT1、T2、T3、T4、T5およびT6を有する。これらは、タイミングブロックのエッジリソースセクション内の波形において定義されるイベントE1、E2、・・・に直接マップされる。６個より多いイベントがタイミングブロック内で定義され、かつこれが上記モジュールとともに用いられれば、それは結果としてエラーをもたらす。

【０７１９】

図１１の例において、最初の波形キャラクタ「０」は、周期的に10nsで起こる「Force Down」あるいは「D」イベントをプログラムするためにタイミングエッジT1を用いる。タイミングエッジT2も30nsで「Force Down」または「D」イベントを生成するために用いられる。最後にタイミングエッジT3は45nsで「Force Off」あるいは「Z」イベントを生成するために用いられる。

【０７２０】

第二の波形キャラクタ「１」は、周期的に10nsで起こる「Force Up」あるいは「U」イベントをプログラムするためにタイミングエッジT1を用いる。タイミングエッジT2も30nsで「Force Down」または「D」イベントを生成するために用いられる。最後にタイミングエッジT3は45nsで「Force Off」あるいは「Z」イベントを生成するために用いられる。

【０７２１】

このようなやり方で、WFCはフレームプロセッサのWFMメモリおよびWTMメモリにマップされる。ピン「SIG」についてのLTS1の波形メモリWFMの最終セットアップを下の表２に示す。

【表２】

Index	(WFC)	T1Set	T1ReSet	T2Set	T2ReSet	T2Drel	T2Dret	EXPH	EXPHZ	T3Set	T3ReSet	T3Drel	T3Dret	T4Drel	T4Dret	EXPL	EXPHZ
0	0		1		1								1				
1	1	1			1								1				
2	d		1	1									1				
3	u	1			1								1				
4	L															1	
5	H							1									
6	m															1	
7	n							1									

【０７２２】

ピン「SIG」についてのLTS1の波形タイミングメモリWTMの最終セットアップを下の表３に示す。

【表３】

Index	(WFC)	T1	T2	EXPH	T3	T4	EXPL
0	0	10ns	30ns		45ns		
1	1	10ns	30ns		45ns		
2	d	12ns	32ns		42ns		
3	u	12ns	32ns		42ns		
4	L						17ns
5	H			17ns			
6	m						15ns
7	n			15ns			

【０７２３】

「CLK」ピンは単一の波形を使いきり、したがってこのピンについてのWFMおよびWFTは非常に単純である。「CLK」ピンについてのLTS1の波形メモリWFMの最終セットアップを下の表４に示す。

【表４】

Index	(WFC)	T1Set	T1ReSet	T2Set	T2ReSet	T2Drel	T2Dret	EXPH	EXPHZ	T3Set	T3ReSet	T3Drel	T3Dret	T4Drel	T4Dret	EXPL	EXPHZ
0	1	1			1												
1																	
2																	
3																	
4																	
5																	
6																	
7																	

【０７２４】

LTS2の波形タイミングメモリWTMの最終セットアップを下の表５に示す。

【表５】

Index	(WFC)	T1	T2	EXPH	T3	T4	EXPL
0	1	20ns	40ns				
1							
2							
3							
4							
5							
6							
7							

【０７２５】

TimingMapブロックは、Timingブロックの波形テーブルにWaveformSelectorを明示的にマップする。テスタシステムについて、これはタイミングセットスクランブラ（TSS）メモリのセットアップを縮める。TSSは基本的には、セッティングを保持するGTSからLTSへのマッピングを含んでいる。ピンSIGについての我々の例に関するTSSのセットアップは下の表６のようになる。

【表６】

GTS	LTS
0 (wfs1)	1
1 (wfs2)	1
2 (wfs3)	2
3 (wfs4)	1
4 (wfs5)	3
5 (wfs6)	1
.	
N (wfs1)	1
.	
255	

【０７２６】
TSSおよびLTSのセットアップマッピングが解決した後に、パターンコンパイラは、正し

い波形テーブル（LTS）および用いるべき正しい波形キャラクタとともにパターンをプログラムするためにこの情報を用いることができる。したがって、ピン「SIG」だけを考える我々の例の擬似パターンが図１１に示される。このコンパイルはタイミングブロックとは依存関係をもたず、Timing-Mapブロックに依存しているのみであることに留意されたい。

Ｇ．　テスタ動作

【０７２７】

このセクションは、テスタオペレーティングシステム（TOS）の基本動作を説明する。このセクションで検討されるアクティビティは

システム初期化
テストプランをローディング
パターンローディング
テストプランを動作させること
個々のテストを動作させること

【０７２８】

である。

システム初期化

【０７２９】

一実施形態においてシステムを初期化するために、ある仮定が満足されなければならず、かつある条件が満たされなければならない。以下のサブセクションはこれらを列挙する。

前提条件

【０７３０】

関連するシステムソフトウェアコンポーネントのコピーは中央記憶を有しており、それの場所はシステムコントローラには知られている。これは、システムコントローラ自体の上であってもよく、あるいは、システムが機能することができる前に、全てのソフトウェアがシステムコントローラにとって利用可能とされなければならないようなどのようなメカニズムであっても、ネットワークマウントディレクトリを有する他のシステム（あるいは他のメカニズムを介してSYSCに知られている上であってもよく）上であってもよい。このソフトウェアは以下を含む。

ベンダハードウェア制御（すなわちモジュールソフトウェア）DLL、
標準あるいはユーザテストクラスDLL、および
ユーザテストプランDLL

【０７３１】

システムモジュールコンフィギュレーションファイルはシステムコントローラ上で利用可能である。このファイルはユーザがテスタの物理的なコンフィギュレーション、例えばシステムシャーシにおける各モジュールの物理的な位置およびタイプ、ならびにモジュールソフトウェアDLLの名前を指定することを可能にすることを思い出してほしい。

【０７３２】

システムコンフィギュレーションファイルは、システムコントローラ上で利用可能である。このファイルは、システムにおけるサイトコントローラのリスト、ならびにスイッチマトリクス入力ポートアドレスに対するサイトコントローラホスト名のマップを含んでいることを思い出してほしい。

【０７３３】

サイトコントローラは、サイトコンフィギュレーションマネージャ（SCM）と呼ばれるサービスを動作させる。このサービスは、「ハードウェアディスカバリ」という用語で呼ばれるプロセスによって、各スロット内にインストールされるハードウェアが何であるか

を判断する任務を有している。また、システムコントローラとともにシステム初期化プロセスに参加する任務も有している。なお、一実施形態においては、スイッチマトリクスオペレーションプロトコルは、単一のサイトコントローラ上のSCMは常に、スイッチマトリクス入力ポート接続アドレス１とともに、モジュールへのスイッチマトリクス接続を構成するために用いられるべきであるということを指示する。この「特別な」サイトがSITEC-1と呼ばれることを思い出してほしい。

【０７３４】

システムコントローラは、各サイトコントローラのSCMにスイッチマトリクス接続アドレスを与えることを担っている。

【０７３５】

各サイトコントローラのSCMは、テストプランサーバ（TPS）と呼ばれるプロセスを開始することができる。各サイトコントローラ上のテストプランサーバは、最後には、ユーザのテストプラン（あるいは、単一のサイトコントローラが複数のDUT上でテストを作動させている場合には複数のテストプラン）を含み、実行する責任がある。

初期化段階Ｉ：　システムの検証

【０７３６】

上記仮定および前提条件が一旦満足されると、システムの初期化はまず、以下のようにシステム検証ステップに進む。

【０７３７】

１．　システムコントローラは、システムのユーザ指定ビューを初期化するために、システムおよびモジュールコンフィギュレーションファイルを読み出す。

【０７３８】

２．　指定されたシステムコンフィギュレーション情報を用いて、システムコントローラは、指定されたサイトコントローラが生きており、到達可能であり、準備ができていること（すなわちSCMが動作していること）を検証する。この検証ステップ中のどのようなエラーも、システムエラーを生じさせ、初期化を途中停止する。

【０７３９】

３．　その後システムコントローラは、SITRC-1のSCMサービスに、全てのハードウェアモジュールへのアクセスを有するようにスイッチマトリクスを構成するよう指示し、それにハードウェアディスカバリを行うことを要求する。

【０７４０】

４．　SITEC-1におけるSCMサービスは、｛ベンダ、ハードウェア｝のタプルについて全ての利用可能であるモジュールスロット（既知のハードウェア位置）をポールし、スロットに対する｛ベンダ、ハードウェア｝のタプルのマップを生成する。最後に、このポールはこのようにして、完成したシステムに存在する｛ベンダ、ハードウェア、スロット｝の結合のセット全体を特定する。このポールの結果はシステムコントローラに送られる。

【０７４１】

５．　システムコントローラは上記ハードウェアディスカバリステップの結果がモジュールコンフィギュレーションファイルにおけるユーザ指定コンフィギュレーションに合致することを検証する。この検証ステップ中のどのようなエラーもシステムエラーを生じさせ、初期化を途中停止する。

【０７４２】

６．　その後システムコントローラは、既知の場所で環境セットアップファイルからデフォルトの環境（モジュールDLL、パターンリスト、パターン、テストプランDLL、テストクラスDLLについてのサーチパス等）をロードする。

【０７４３】

７．　システムコントローラは、全ての識別されたモジュールソフトウェアDLLが存在することを保証する。もしシステムコントローラ上で利用可能でなければ、それは可能であれば中央記憶から取り出され、そうでなければシステムエラーが生じ、初期化を途中停

止する。

初期化段階II：サイトコンフィギュレーション（オプション）

【０７４４】

サイトコンフィギュレーション、あるいはサイトを区切ることは、利用可能なシステムハードウェアモジュールを異なるサイトに（すなわち複数のDUTにサービスを提供するために）ソフトウェアレベルで割り当てることを含んでいる。サイトを区切るための情報はソケットファイルにおいて提供されることを思い出してほしい。

【０７４５】

テスタシステムは、サイトの（再）区切りが、テストプランロードの一部として（なぜなら各テストプランは特定のソケットに関連付けられているので）、および独立したユーザが呼び出し可能なステップとしての両方で行われることを可能にする。後の場合には、ユーザは、システムを区切るために単独で用いられるソケットファイルを提供することによってサイト区切りを始動させる。これは、各サイトが異なるDUTタイプをテストする複数のDUTテストの場合におけるシステムの初期化中に特に有用である。しかしながら、このステップは、初期化のステージではオプション的であり、ユーザはそれを行わせないことを選び、代わりにテストプランロードがシステムを適切に区切ることを可能にすることを選ぶこともできる。

【０７４６】

（独立したコールあるいはテストプランロードを通じて暗黙のうちに）サイト区切りを達成するためにどのような手段が選ばれようと、このメカニズムを以下説明する。

【０７４７】

１．　ソケットが与えられると、システムコントローラはまず、現在存在するシステムパーティションはソケットと互換性があるか、あるいは再区切りが必要かを判断する。初期化中のデフォルトのパーティションは、全ての利用可能なモジュールがSITEC-1に接続されているようなパーティションである。以下の残りのステップは、もしパーティションを切りなおす必要がある場合のみ行われる。

【０７４８】

２．　システムコントローラは、新しいソケットの元でそれについて有効にされているDUTサイトの数および識別で自身を再構成するために各サイトコントローラのSCMにコンフィギュレーションメッセージを送る。なおこれは一般的な手順であり、サイトコントローラによって制御されるDUTサイトの数が一つである場合を扱う。新しいソケット情報もSCMに運ばれる。

【０７４９】

３．　各SCMは、もしあれば動作しているTPSを停止し、それを新しいものを開始して、それを新しいソケット、ならびにその新しいソケットの元でそれに対して有効にされているDUTサイトの数および識別で初期化する。

【０７５０】

４．　システムコントローラは、どのサイトが、要求されているシステムモジュールのどのサブセットを必要としているかを判断する。これをしている間、サイトについてのハードウェアスロット情報の準備もする。最終結果は、各サイトについて、スロット対そのサイトに割り当てられたモジュールDLLのリストである。このサイト特有のリストは、サイトモジュールDLLスロットリスト（SITE-MDSL）と表される。

【０７５１】

５．　システムコントローラは適切なSITE-MDSL、ならびに必要なモジュールDLLを各SCMに提供する。各SCMはこの情報を新しく開始されたTPSに役立つようにする。

【０７５２】

６．　その後システムコントローラは、STRC-１に、適切なサイト－スロット接続について、すなわちサイトが区切られた動作についてのスイッチマトリクスを構成するように要求する。

【０７５３】
７．　サイト１～ｎについてのTPSはそれらのSITE-MDSLにおいて指定されるDLLをロードする。これらのDLLのそれぞれは、スロットメンバのアレイを取り出すinitialize()という名前の関数を有している。TPSはそのモジュールタイプについての適切なスロットリストとともにinitialize()を呼び出す。この時点で何らかの不具合があれば、システムエラーが生じ、初期化を途中で終わる。このinitialize()方法は以下を行う。
【０７５４】
ａ．　標準インタフェースIXXXModuleに基づいて具体的なクラスを作成する。例えば、デジタルモジュールに関連付けられているDLLは、それに関連付けられている各スロットにサービスを提供するように単一のIPinModuleベースのオブジェクトを作成する。
【０７５５】
ｂ．　インタフェースIResourceに基づいて具体的なクラスを、モジュールにおける各「リソースユニット」について一つ作成する。また、デジタルモジュールについて、各IPinModuleベースオブジェクトは、デジタルモジュールによって占められるスロットの集合体における全てのピンについてITesterPinベースオブジェクトを作成する。
【０７５６】
８．　サイト１～ｎについてのTPSはその後、モジュールコンテンツ情報を取得するために各ロードされたモジュールDLL上にgetXXXModule()を呼び出す。
【０７５７】
９．　getXXXModule()へのそれぞれのコールは、IModuleポインタ（例えばAdvantestPinModule）として<VendorHWType>Moduleクラスオブジェクトを返す。このようなIModuleポインタのそれぞれはTPSによってキャッシュに記憶され、これらをフレームワーク／ユーザコードにとって利用可能にする。IModule、IResource等の集合体は持続する（少なくともTPSの寿命の間）ということに留意されたい。
【０７５８】
１０．上記ステップが一旦完了すると、TPSはその割り当てられた（既知の）ポート上でlisten()を開始する。これはシステムコントローラに、TPSが通常の（すなわちサイトが区切られた）動作を開始する「用意ができて」いることを伝える。

５．テストプランロード
【０７５９】
このセクションは、そのステップによってユーザテストプランDLLがサイトコントローラ上にロードされるようなステップを（単一あるいは複数のDUTテストについて）説明する。
【０７６０】
システムの初期化（およびオプションとして最初のサイト区切り）が一旦完了すると、ユーザテストプランをロードすることができる。サイトコントローラ上へのユーザテストプランのロードは次のようにして進行する。
【０７６１】
１．　システムコントローラはまず、それ自身の処理空間にテストプランDLLをロードし、それの関連付けられているソケットファイルおよびDUTタイプ識別子について質問を行う。この情報は、このテストプランを動作させるサイト、ならびに、したがってこのテストプランがロードされるサイトコントローラを識別するのに用いられる。
【０７６２】
２．　その後システムコントローラは、上で概略を説明したように再区切りプロセスを始動させるためにテストプランに関連付けられたソケット情報を用いる。
【０７６３】
３．　システムコントローラは、テストプランによって用いられるテストクラスDLLのリストをテストプランDLLから抽出し、一旦システムコントローラがTPSは通常の（すなわちサイトが区切られた）動作を始める準備ができていると検証すると、適切なTPSに対し

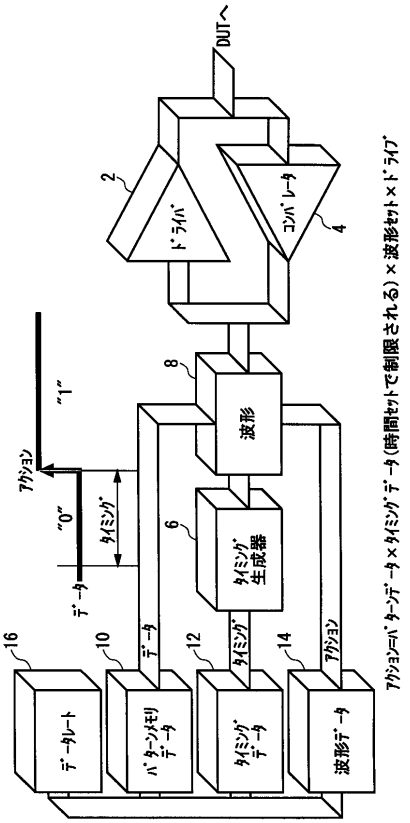
て、テストクラスDLLを送り、最後にテストプランDLL自体を送る。
【０７６４】
４．　TPSは、その処理空間にそれをロードするためのLoatLibrary()を呼び出す。それは、それがサービスを提供しているサイト（すなわちDUT）の数と同じ数のTestPlanオブジェクトを作成するためにDLL上に既知の関数を呼び出す。
【０７６５】
５．　TPSは、必要なテスタフレームワークオブジェクトでTestPlanオブジェクトを初期化する。初期化中に、TPSは処理空間内に、TestPlanオブジェクトによって用いられるテストクラスのための適切なDLLをロードし、テストクラスインスタンスを作成する。
【０７６６】
６．　TPSは、システムコントローラへ／からの通信チャネルをTestPlanオブジェクトにセットアップする。
【０７６７】
７．　システムコントローラはTPSと通信し、TestPlanオブジェクトのためのそのプロキシを作る。
【０７６８】
これで、サイトコントローラ上へのユーザのテストプランの正常なロードを終了する。

テストプランを動作させる
【０７６９】
先に定義したフローロジックにしたがってテストプランにおける全てのテストを実行する方法は以下の通りである。
【０７７０】
１．　ユーザのアプリケーションがTPSにRunTestPlanメッセージを送信する。TPSは全ての接続されているアプリケーションにExecutingTestPlanメッセージを送る。そしてTPSはテストプラン上にexecute()を呼び出す。
【０７７１】
２．　単一のサイトコントローラで複数のDUTをテストすることは、そのサイトコントローラ上で、一つのDUTに一つの複数のスレッドを用いて、行われる。各スレッドは、同一のTestPlanオブジェクトの異なる独立したインスタンスを動作させる。この場合、モジュール制御ソフトウェアDLLはDUT間で共有されているかもしれないので、ハードウェア通信についてのモジュールコマンドは、DUT識別子パラメータをとることが要求される。
【０７７２】
３．　TestPlanオブジェクトは、各テストをその集合体において反復し（あるいは、そのFlowオブジェクトにフローロジックにしたがって各テストを処理するように言い）、preExec()、execute()およびpostexec()を呼び出す。
【０７７３】
４．　各テストの実行の際に、ステータスメッセージが全ての接続されたアプリケーションに対して送り返される。

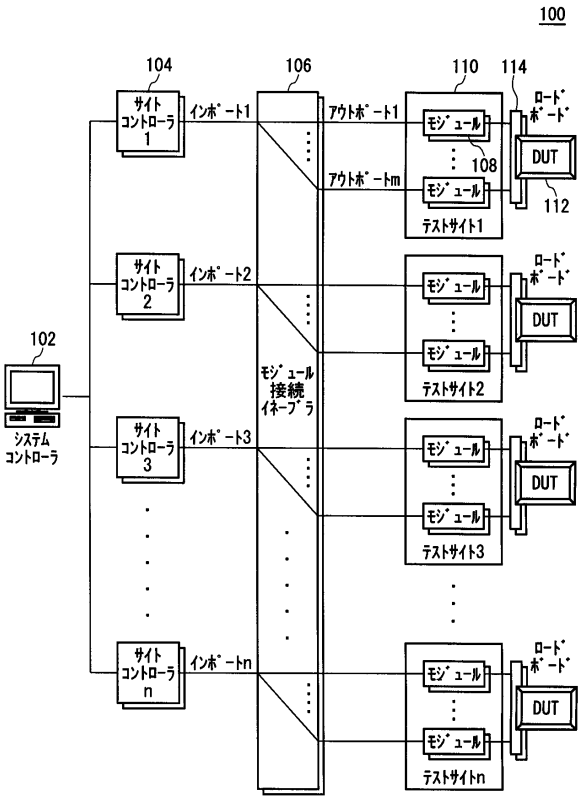
単一のテストを実行する
【０７７４】
ユーザは、全てのテストの代わりに、テストプランにおける単一のテストを実行することを望むかもしれない。単一のテストの実行については、方法は以下の通りである。
【０７７５】
１．　ユーザアプリケーションがTPSにRunTestメッセージを送信し、TPSは全ての接続されたアプリケーションにExecutingTestメッセージを送る。TPSはその後テストプラン上にexecute()を呼び出し、動作すべきテストプランを指定する。
【０７７６】
２．　テストプランオブジェクトは、そのテストオブジェクト上にpreExec()、execute

()およびpostExec()を呼び出すことによって指定されたテストを実行する。
【０７７７】
３．　テスト実行時に、それは全ての接続されたアプリケーションにステータスメッセージを送り返す。
【０７７８】
発明を特定の実施形態に関連して説明したが、当業者は発明の精神および範囲から逸脱することなくさまざまな改変および修正を行い得ることが理解されるであろう。発明は、前述の例示的な詳細に限定されるべきではなく、クレームの範囲にしたがって解釈されるべきである。
【図面の簡単な説明】
【０７７９】
【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テスタ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テスタソフトウェアアーキテクチャを示している。
【図４】図４は本発明の一実施形態によるテストプログラムコンパイラを示している。
【図５】図５は、本発明の一実施形態による、どのようにして単一のテストクラスから異なるテストインスタンスが得られるかを示している。
【図６】図６は本発明の一実施形態によるパターンコンパイラを示している。
【図７】図７は本発明の一実施形態による順序決めされたパターンツリー例を示している。
【図８】図８は本発明の一実施形態による他の順序決めされたパターンツリー例を示している。
【図９】図９は本発明の一実施形態による、テストプログラムによって必要とされるファイル間の関係を示している。
【図１０】図１０は本発明の一実施形態による波形生成を示している。
【図１１】図１１は本発明の一実施形態によるタイミングのために用いられるマッピングを示している。
【図１２】図１２は本発明の一実施形態によるタイミングのために用いられる他のマッピングを示している。
【符号の説明】
【０７８０】
６０２　　パターンコンパイラ
６０８　　モジュール
６１０　　パターンコンパイラ
６１２　　パターンオブジェクトメタファイル
６１４　　オブジェクトファイルマネージャ
１００２　　パターンジェネレータ
１００４　　フレームプロセッサ
１０１０　　インデックス
１０１２　　波形メモリ
１０１４　　タイミングメモリ
１１１２　　波形メモリ
１１１４　　波形タイミングメモリ

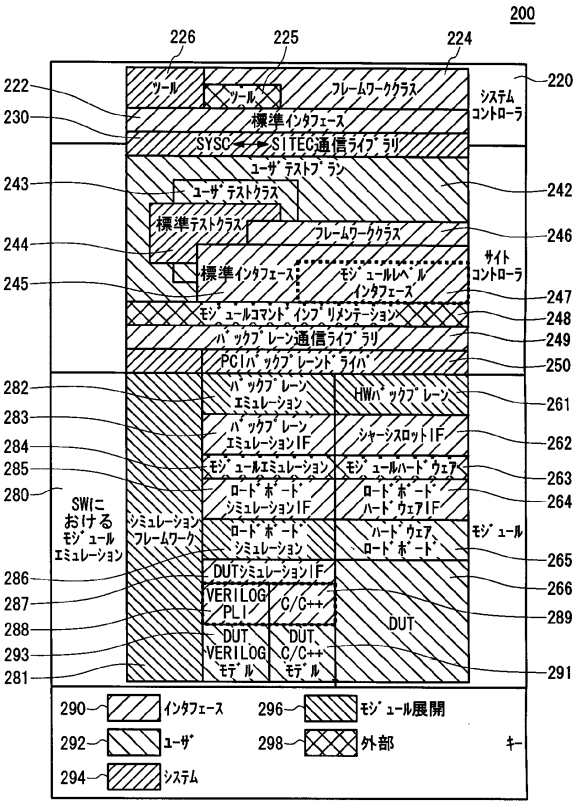
【図１】


DUTへ
ドライバ
コンパレータ
2
4
8
波形
6
タイミング生成器
"1"
"0"
アクション
タイミング
データ
16
10
12
14
データレート
パターンメモリデータ
タイミングデータ
波形データ
アクション=パターンデータ×タイミングデータ(時間セットで制限される)×波形セット×ドライブ

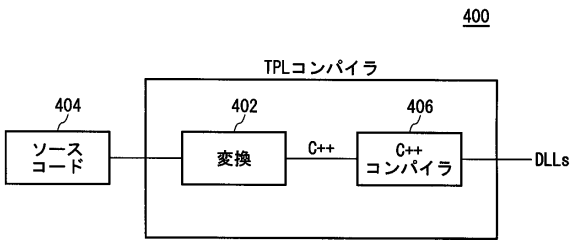
【図２】


100
104
106
110
114
サイトコントローラ1
サイトコントローラ2
サイトコントローラ3
サイトコントローラn
インポート1
インポート2
インポート3
インポートn
アウトポート1
アウトポートm
モジュール
108
DUT
112
ロードボード
テストサイト1
テストサイト2
テストサイト3
テストサイトn
102
システムコントローラ
モジュール接続イネーブラ

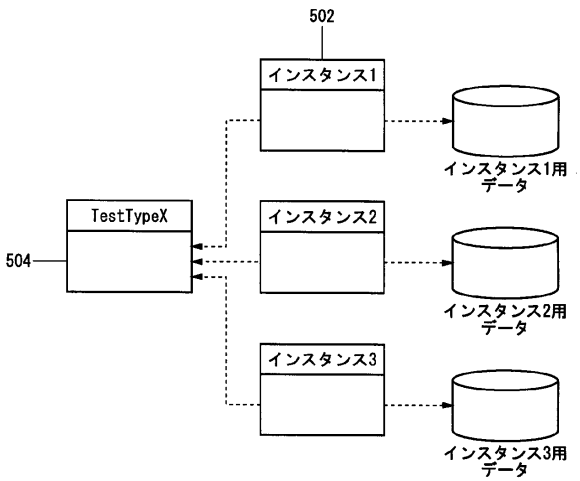
【図３】


200
226
225
224
220
222
230
243
242
244
246
245
247
248
249
250
282
261
283
262
284
263
285
264
280
265
286
266
287
289
288
293
291
281
ツール
フレームワーククラス
システムコントローラ
標準インタフェース
SYSC→SITEC通信ライブラリ
ユーザテストプラン
ユーザテストクラス
標準テストクラス
フレームワーククラス
標準インタフェース
モジュールレベルインタフェース
モジュールコマンドインプリメンテーション
バックプレーン通信ライブラリ
PCIバックプレーンドライバ
バックプレーンエミュレーション
HWバックプレーン
バックプレーンエミュレーションIF
シャーシスロットIF
モジュールエミュレーション
モジュールハードウェア
ロードボードシミュレーションIF
ロードボードハードウェアIF
ロードボードシミュレーション
ハードウェアロードボード
DUTシミュレーションIF
VERILOG PLI
C/C++
DUT
DUT VERILOG モデル
DUT C/C++ モデル
SWにおけるモジュールエミュレーション
シミュレーションフレームワーク
モジュール
290 インタフェース
292 ユーザ
294 システム
296 モジュール展開
298 外部
キー

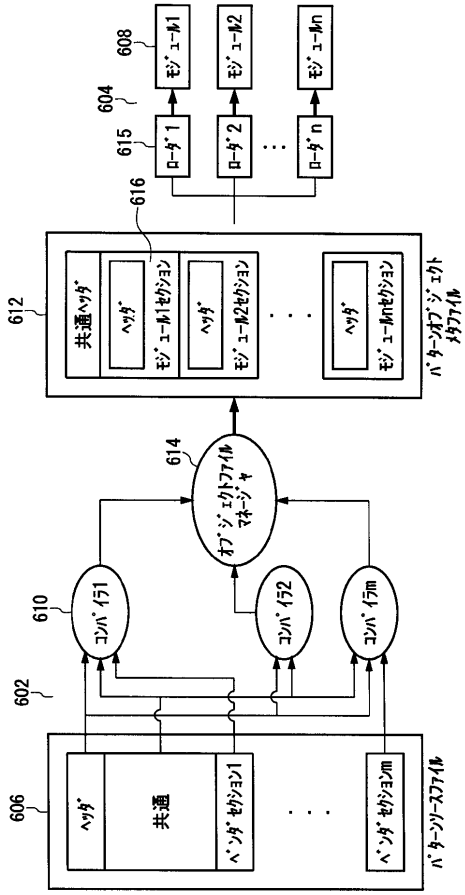
【図４】


400
TPLコンパイラ
404
402
406
ソースコード
変換
C++
C++コンパイラ
DLLs

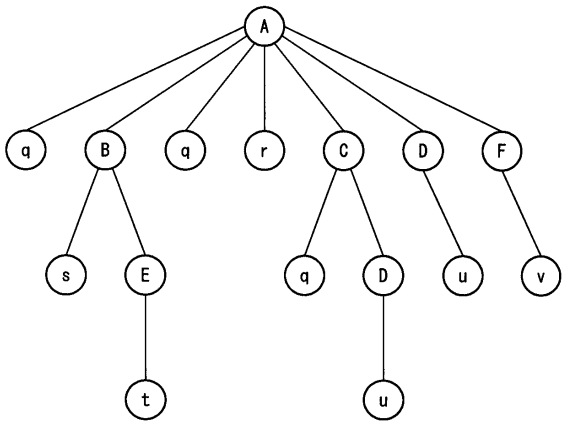
【図５】


502
インスタンス1
インスタンス1用データ
TestTypeX
504
インスタンス2
インスタンス2用データ
インスタンス3
インスタンス3用データ

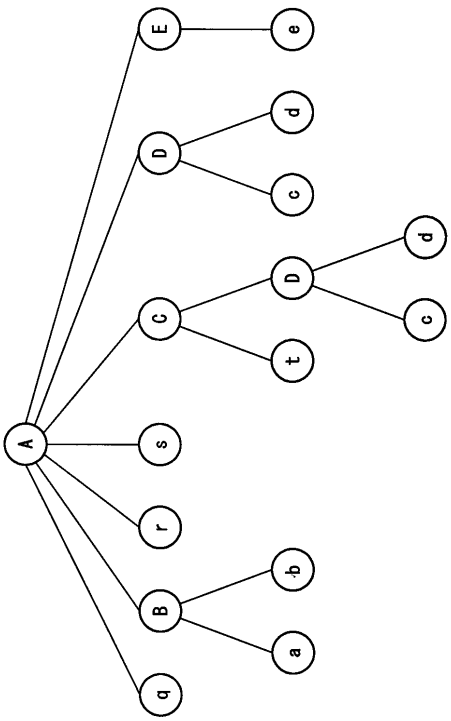
【図６】


608
モジュール1
モジュール2
モジュールn
604
615
ローダ1
ローダ2
ローダn
616
612
共通ヘッダ
ヘッダ
モジュール1セクション
ヘッダ
モジュール2セクション
ヘッダ
モジュールnセクション
パターンオブジェクトメタファイル
614
オブジェクトファイルマネージャ
610
コンパイラ1
コンパイラ2
コンパイラm
602
606
ヘッダ
共通
パンダセクション1
パンダセクションm
パターンソースファイル

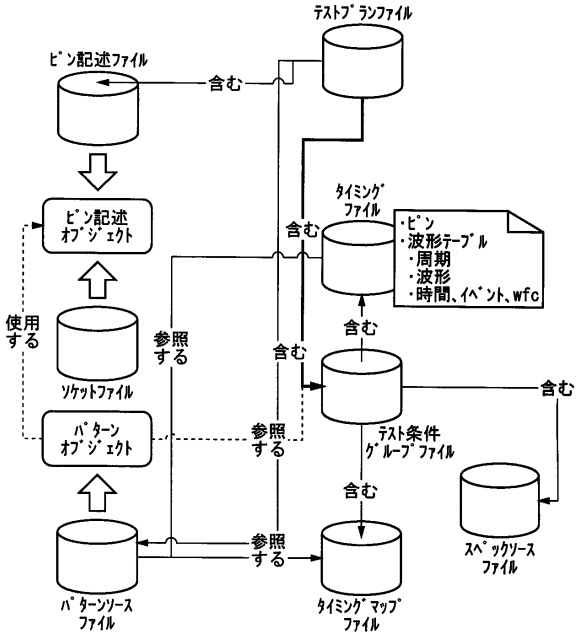
【図７】


A
q
B
q
r
C
D
F
s
E
q
D
u
v
t
u

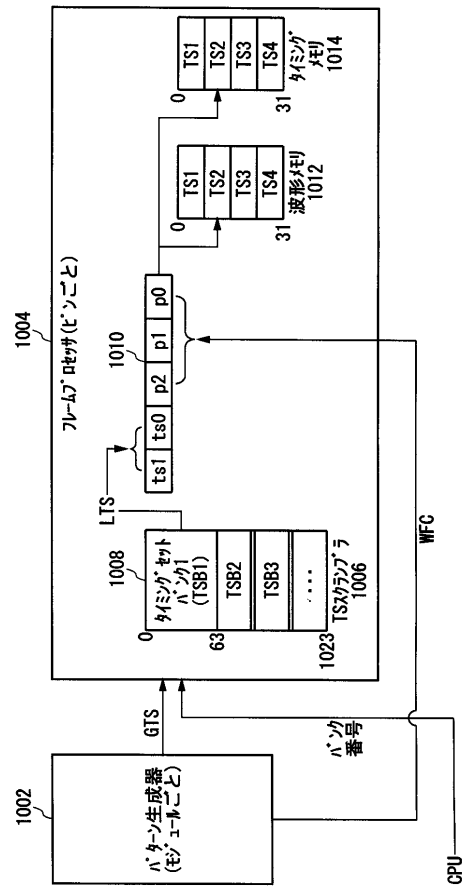
【図８】


E
e
D
d
c
C
D
d
c
t
A
s
r
B
b
a
q

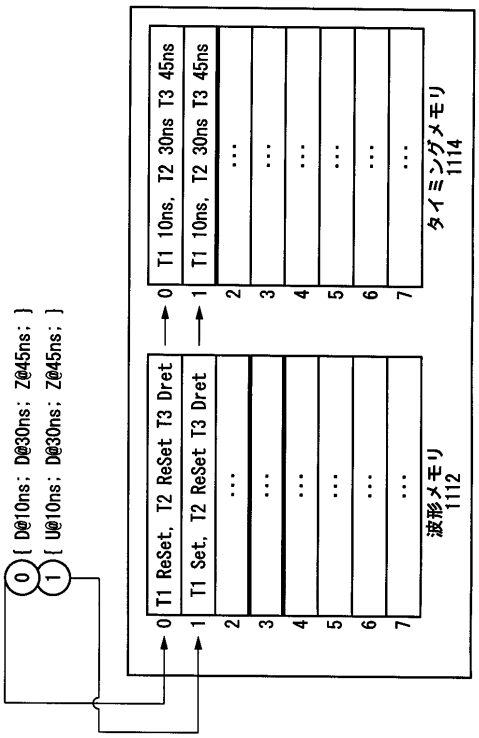
【図９】


テストプランファイル
ピン記述ファイル
含む
タイミング
ファイル
含む
・ピン
・波形テーブル
・周期
・波形
・時間、イベント、wfc
ピン記述
オブジェクト
使用する
参照する
含む
含む
ソケットファイル
含む
パターン
オブジェクト
参照する
テスト条件
グループファイル
含む
参照する
スペックソース
ファイル
パターンソース
ファイル
タイミングマップ
ファイル

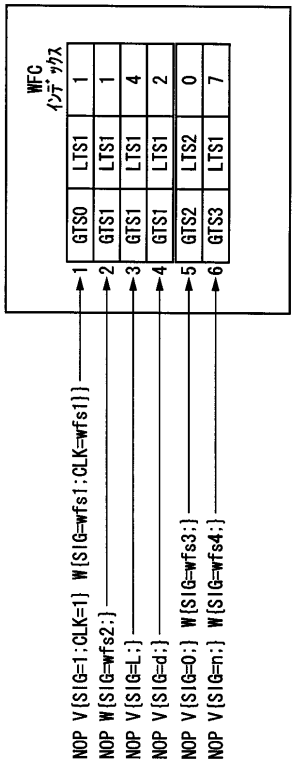
【図１０】


1004
フレームプロセッサ(ピンごと)
1010
p0
p1
p2
ts0
ts1
LTS
1008
タイミングセット
パンク1
(TSB1)
TSB2
TSB3
TSメカニックテーブル
1006
0
63
1023
波形メモリ
1012
タイミング
メモリ
1014
TS1
TS2
TS3
TS4
31
WFC
GTS
パンク
番号
1002
パターン生成器
(モジュールごと)
CPU

【図１１】


0 { D@10ns; D@30ns; Z@45ns; }
1 { U@10ns; D@30ns; Z@45ns; }
T1 ReSet, T2 ReSet T3 Dret
T1 Set, T2 ReSet T3 Dret
T1 10ns, T2 30ns T3 45ns
T1 10ns, T2 30ns T3 45ns
波形メモリ
1112
タイミングメモリ
1114

【図１２】


WFC
インデックス
GTS0 LTS1 1
GTS1 LTS1 1
GTS1 LTS1 4
GTS1 LTS1 2
GTS2 LTS2 0
GTS3 LTS1 7
NOP V{SIG=1;CLK=1} W{SIG=wfs1;CLK=wfs1}}
NOP W{SIG=wfs2;}
NOP V{SIG=L;}
NOP V{SIG=d;}
NOP V{SIG=0;} W{SIG=wfs3;}
NOP V{SIG=n;} W{SIG=wfs4;}

フロントページの続き

(31)優先権主張番号 10/403,817
(32)優先日 平成15年3月31日(2003.3.31)
(33)優先権主張国 米国(US)
(31)優先権主張番号 10/404,002
(32)優先日 平成15年3月31日(2003.3.31)
(33)優先権主張国 米国(US)

早期審査対象出願

(72)発明者 エルストン マーク
東京都練馬区旭町１丁目３２番１号 株式会社アドバンテスト内
(72)発明者 チェン リーオン
東京都練馬区旭町１丁目３２番１号 株式会社アドバンテスト内
(72)発明者 足立 敏明
東京都練馬区旭町１丁目３２番１号 株式会社アドバンテスト内
(72)発明者 田原 善文
東京都練馬区旭町１丁目３２番１号 株式会社アドバンテスト内

審査官 久保 正典

(56)参考文献 米国特許出願公開第２００２／００７３３７５（ＵＳ，Ａ１）
米国特許第０５４８８５７３（ＵＳ，Ａ）
米国特許第０６１９５７７４（ＵＳ，Ｂ１）
米国特許出願公開第２００３／０００５３７５（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F11/22-11/277
G01R31/28-31/319