

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第4332200号
(P4332200)

(45) 発行日 平成21年9月16日 (2009. 9. 16)

(24) 登録日 平成21年6月26日 (2009. 6. 26)

(51) Int. Cl.

F I

G O 1 R 31/3183 (2006.01)

G O 1 R 31/28

Q

請求項の数 26 外国語出願 (全 166 頁)

(21) 出願番号	特願2008-180843 (P2008-180843)	(73) 特許権者	390005175
(22) 出願日	平成20年7月11日 (2008. 7. 11)		株式会社アドバンテスト
(62) 分割の表示	特願2006-519573 (P2006-519573) の分割		東京都練馬区旭町 1 丁目 3 2 番 1 号
原出願日	平成17年5月23日 (2005. 5. 23)	(74) 代理人	100104156
(65) 公開番号	特開2009-8683 (P2009-8683A)		弁理士 龍華 明裕
(43) 公開日	平成21年1月15日 (2009. 1. 15)	(74) 代理人	100118005
審査請求日	平成20年8月8日 (2008. 8. 8)		弁理士 飯山 和俊
(31) 優先権主張番号	60/573, 577	(74) 代理人	100143502
(32) 優先日	平成16年5月22日 (2004. 5. 22)		弁理士 明石 英也
(33) 優先権主張国	米国 (US)	(74) 代理人	100138128
(31) 優先権主張番号	10/918, 513		弁理士 東山 忠義
(32) 優先日	平成16年8月13日 (2004. 8. 13)	(74) 代理人	100112520
(33) 優先権主張国	米国 (US)		弁理士 林 茂則
		(74) 代理人	100156591
			弁理士 高田 学

最終頁に続く

(54) 【発明の名称】 モジュール式試験システム内のパターンオブジェクトファイルを管理するための方法およびモジュール式試験システム

(57) 【特許請求の範囲】

【請求項 1】

モジュール式試験システム内のパターンオブジェクトファイルを管理するための方法であって、

モジュール式試験システムを配設することであって、該モジュール式試験システムは少なくとも1つのサイトコントローラを制御するためのシステムコントローラを備え、該少なくとも1つのサイトコントローラは少なくとも1つの試験モジュールおよびその対応する被試験デバイス (DUT) を制御する、モジュール式試験システムを配設すること、

複数のベンダ供給パターンコンパイラと前記モジュール式試験システムとの間に標準インターフェースを確立するためのオブジェクトファイル管理 (OFM) フレームワークを作成すること、

複数のタイプのパターンブロックを含むパターンソースファイルを受信すること、

各ベンダ供給パターンコンパイラから得られた情報から共通のタイプのパターンブロックを決定することであって、前記共通のタイプのパターンブロックは全てのベンダ供給パターンコンパイラにより利用することができること、

前記OFMフレームワークを用いて、前記パターンソースファイルに基づいてパターンオブジェクトメタファイルを作成することであって、前記共通のタイプのパターンブロックを、複数の同じインスタンスを許容しないように共通セクションに集約することを含む、
パターンオブジェクトメタファイルを作成すること、及び

前記パターンオブジェクトメタファイルを用いて前記試験モジュールを通して前記DUT

10

20

Tを試験すること

を含む、モジュール式試験システム内のパターンオブジェクトファイルを管理するための方法。

【請求項 2】

前記 OFM フレームワークは、ベンダ供給パターンデータを前記モジュール式試験システムに組み込むことをサポートするための標準インターフェースクラスを提供する、請求項 1 に記載の方法。

【請求項 3】

前記 OFM フレームワークはさらに、モジュール特有のパターンデータを種々のベンダから前記パターンオブジェクトメタファイルに転送するように構成される、請求項 1 に記載の方法。

【請求項 4】

前記 OFM フレームワークを作成することは、

前記ベンダ供給パターンコンパイラと前記試験モジュールを接続するためのベンダパターンコンパイラクラスを作成すること、

パターンオブジェクトメタファイルクラスにアクセスするための OFM モジュールエージェントクラスを作成することであって、該パターンオブジェクトメタファイルクラスは、コンパイルされる前記パターンオブジェクトメタファイルをカプセル化する、OFM モジュールエージェントクラスを作成すること、及び

パターンコンパイル中に用いられる他の補助パターンオブジェクトメタファイルの特定の部分にアクセスするための OFM ラベルリーダークラスを作成することを含む、請求項 1 に記載の方法。

【請求項 5】

前記ベンダパターンコンパイラクラスを作成することは、

前記パターンソースファイルのパターンコンパイルをサポートすること、

前記パターンオブジェクトメタファイルのモジュール特有のセクションを更新すること

、
前記パターンオブジェクトメタファイルの前記共通セクションを更新すること、及びコンパイルエラーを一纏めにすること

を含む、請求項 4 に記載の方法。

【請求項 6】

前記 OFM モジュールエージェントクラスを作成することは、

前記共通セクションからの前記ベンダ供給パターンコンパイラの読出し動作をサポートすること、

ベンダ特有のセクションからの前記ベンダ供給パターンコンパイラの読出し動作をサポートすること、及び

前記ベンダ供給パターンコンパイラから前記パターンオブジェクトメタファイルのモジュール特有のセクションへの書込み動作をサポートすること

を含む、請求項 4 に記載の方法。

【請求項 7】

前記 OFM ラベルリーダークラスを作成することは、

補助パターンオブジェクトメタファイルのラベルにアクセスすること、及び

前記補助パターンオブジェクトメタファイルのラベルオフセットにアクセスすることを含む、請求項 4 に記載の方法。

【請求項 8】

前記パターンオブジェクトメタファイルは、

前記パターンソースファイル内の非サイクルベースのパターンブロックを表すための 1 つまたは複数のパターンブロックと、

前記パターンソースファイル内のサイクルベースのパターンブロックを表すための多くても 1 つのサイクライズパターンブロックと

を含む、請求項 1 に記載の方法。

【請求項 9】

前記パターンオブジェクトメタファイルを作成することは、
パターンのコンパイルを制御するために OFM マネージャを初期化すること、
各ベンダモジュールが前記パターンオブジェクトメタファイルによって利用されるべく
ベンダパターンコンパイラインターフェースオブジェクトをインスタンス化することであ
って、該ベンダパターンコンパイラインターフェースオブジェクトは多数のタイプのパタ
ーンブロックをサポートする、インスタンス化すること、
ベンダパターンコンパイラインターフェースオブジェクトに従って前記試験モジュール
によってサポートされるシステム資源のリストを獲得すること、

10

パターンコンパイラを用いて前記パターンソースファイル内の 1 以上の共通のタイプの
パターンブロックを含む資源タイプのリストをコンパイルすることであって、それによ
って、共通セクションデータを生成する、コンパイルすること、

パターンオブジェクトメタファイルクラスとして共通セクションデータを格納すること
であって、該共通セクションデータは、前記ベンダ供給パターンコンパイラがアクセスで
きる情報をパターンオブジェクトメタファイルに含む、共通セクションデータを格納する
こと、

並びに

コンパイルエラーおよび警告のリストを生成すること
を含む、請求項 1 に記載の方法。

20

【請求項 10】

前記パターンコンパイラは、
少なくとも 1 つのモジュール特有のパターンコンパイラと、
各モジュール特有のコンパイラに対し、前記対応するパターンソースファイルのモジュ
ール特有のセクションおよび共通セクションの両方をコンパイルするように指示するた
めのオブジェクトファイルマネージャとを含む、請求項 9 に記載の方法。

【請求項 11】

前記ベンダパターンコンパイラインターフェースオブジェクトはサイクルベースパター
ンコンパイラインターフェースオブジェクトを含む、請求項 9 に記載の方法。

【請求項 12】

前記ベンダパターンコンパイラインターフェースオブジェクトは非サイクルベースパタ
ーンコンパイラインターフェースオブジェクトを含む、請求項 9 に記載の方法。

30

【請求項 13】

前記多数のタイプのパターンブロックは、サイクルベースパターンおよび非サイクルベ
ースパターンからなるグループから選択される 1 つまたは複数の項目を含む、請求項 9 に
記載の方法。

【請求項 14】

モジュール式試験システムであって、
システムコントローラと、
前記システムコントローラに接続される少なくとも 1 つのサイトコントローラと、
少なくとも 1 つの試験モジュールおよびその対応する被試験デバイス (DUT) であ
って、該試験モジュールは前記サイトコントローラによって制御される、少なくとも 1 つの
試験モジュールおよびその対応する被試験デバイスと、

40

複数のベンダ供給パターンコンパイラと該モジュール式試験システムとの間に標準イン
ターフェースを確立するためのオブジェクトファイル管理 (OFM) フレームワークと、
複数のタイプのパターンブロックを含むパターンソースファイルを受信するための手段
と、

各ベンダ供給パターンコンパイラから得られた情報から共通のタイプのパターンブロッ
クを決定する手段であって、前記共通のタイプのパターンブロックは全てのベンダ供給パ
ターンコンパイラにより利用することができる手段、

50

前記OFMフレームワークを用いて、前記パターンソースファイルに基づいてパターンオブジェクトメタファイルを作成するための手段であって、前記共通のタイプのパターンブロックを、複数の同じインスタンスを許容しないように共通セクションに集約することを含む、パターンオブジェクトメタファイルを作成する手段、と、

前記パターンオブジェクトメタファイルを用いて、前記試験モジュールを通して前記DUTを試験するための手段と
を備える、モジュール式試験システム。

【請求項15】

前記OFMフレームワークは、ベンダ供給パターンデータを前記モジュール式試験システムに組み込むことをサポートするための標準インターフェースクラスを提供する、請求項14に記載のシステム。

【請求項16】

前記OFMフレームワークは、モジュール特有のパターンデータを種々のベンダから前記パターンオブジェクトメタファイルに転送するように構成される、請求項14に記載のシステム。

【請求項17】

前記OFMフレームワークは、

前記ベンダ供給パターンコンパイラと前記試験モジュールを接続するためのベンダパターンコンパイラクラスを作成するための手段と、

パターンオブジェクトメタファイルクラスにアクセスするためのOFMモジュールエージェントクラスを作成するための手段であって、該パターンオブジェクトメタファイルクラスは、コンパイルされる前記パターンオブジェクトメタファイルをカプセル化する、作成するための手段と、

パターンコンパイル中に用いられる他の補助パターンオブジェクトメタファイルの特定の部分にアクセスするためのOFMラベルリーダクラスを作成するための手段と
をさらに備える、請求項14に記載のシステム。

【請求項18】

前記ベンダパターンコンパイラクラスを作成するための前記手段は、

前記パターンソースファイルのパターンコンパイルをサポートするための手段と、

前記パターンオブジェクトメタファイルのモジュール特有のセクションを更新するための手段と、

前記パターンオブジェクトメタファイルの前記共通セクションを更新するための手段と、

コンパイルエラーを一纏めにするための手段と
を備える、請求項17に記載のシステム。

【請求項19】

前記OFMモジュールエージェントクラスを作成するための前記手段は、

前記共通セクションからの前記ベンダ供給パターンコンパイラの読出し動作をサポートするための手段と、

モジュール特有のセクションからの前記ベンダ供給パターンコンパイラの読出し動作をサポートするための手段と、

前記ベンダ供給パターンコンパイラから前記パターンオブジェクトメタファイルの前記モジュール特有のセクションへの書込み動作をサポートするための手段と
を備える、請求項17に記載のシステム。

【請求項20】

前記OFMラベルリーダクラスを作成するための前記手段は、

補助パターンオブジェクトメタファイルのラベルにアクセスするための手段と、

前記補助パターンオブジェクトメタファイルのラベルオフセットにアクセスするための手段と

を備える、請求項17に記載のシステム。

10

20

30

40

50

【請求項 2 1】

前記パターンオブジェクトメタファイルは、

前記パターンソースファイル内の非サイクルベースパターンブロックを表すための 1 つまたは複数のパターンブロックと、

前記パターンソースファイル内のサイクルベースパターンブロックを表すための多くても 1 つのサイクライズパターンブロックと

を含む、請求項 1 4 に記載のシステム。

【請求項 2 2】

前記パターンオブジェクトメタファイルを作成するための前記手段は、

パターンのコンパイルを制御するために OFM マネージャを初期化するための手段と、

各ベンダモジュールが前記パターンオブジェクトメタファイルによって利用されるべくベンダパターンコンパイラインターフェースオブジェクトをインスタンス化するための手段であって、該ベンダパターンコンパイラインターフェースオブジェクトは多数のタイプのパターンブロックをサポートする、インスタンス化するための手段と、

ベンダパターンコンパイラインターフェースオブジェクトに従って前記試験モジュールによってサポートされるシステム資源のリストを獲得するための手段と、

パターンコンパイラを用いて前記パターンソースファイル内の 1 以上の共通のタイプのパターンブロックを含む資源タイプ のリストをコンパイルするための手段であって、それによって、共通セクションデータを生成する、コンパイルするための手段と、

パターンオブジェクトメタファイルクラスとして共通セクションデータを格納する手段であって、該共通セクションデータは、前記ベンダ供給パターンコンパイラがアクセスできる情報をパターンオブジェクトメタファイルに含む、格納する手段と、 コンパイルエラーおよび警告のリストを生成するための手段と

を備える、請求項 1 4 に記載のシステム。

【請求項 2 3】

前記パターンコンパイラは、

少なくとも 1 つのモジュール特有のパターンコンパイラと、

各モジュール特有のコンパイラに対し、前記対応するパターンソースファイルのモジュール特有のセクションおよび共通セクションの両方をコンパイルするように指示するためのオブジェクトファイルマネージャと

を備える、請求項 2 2 に記載のシステム。

【請求項 2 4】

前記ベンダパターンコンパイラインターフェースオブジェクトはサイクルベースパターンコンパイラインターフェースオブジェクトを含む、請求項 2 2 に記載のシステム。

【請求項 2 5】

前記ベンダパターンコンパイラインターフェースオブジェクトは非サイクルベースパターンコンパイラインターフェースオブジェクトを含む、請求項 2 2 に記載のシステム。

【請求項 2 6】

前記多数のタイプのパターンブロックは、サイクルベースパターンおよび非サイクルベースパターンからなるグループから選択される 1 つまたは複数の項目を含む、請求項 2 2 に記載のシステム。

【発明の詳細な説明】

【技術分野】

【0001】

本特許出願は、2004 年 2 月 6 日に出願の同時係属の米国特許出願第 10 / 772, 434 号「Method and Structure to Develop a Test Program for Semiconductor Integrated Circuits」の一部継続出願であり、同特許出願の恩典を主張し、同特許出願は、2003 年 2 月 14 日に米国特許出願第 60 / 447, 839 号「Method and Structure to Develop a Test Program for Semiconductor Integrated Circuits」の恩典を主張する。また本特許出願は、2004 年 5 月 22 日に米国仮特許出願第 60 / 57

10

20

30

40

50

3, 577号「Software Development in an Open Architecture Test System」の恩典も主張する。それらの特許出願は全て、アドバンテスト社に譲渡され、その全体を参照することにより本明細書に援用される。

【0002】

本発明は半導体検査のための自動試験装置（ATE）の分野に関する。詳細には、本発明は、モジュール式試験システム内のパターンオブジェクトファイルを管理するための方法およびシステムに関する。

【背景技術】

【0003】

従来のATEシステムでは、パターンオブジェクトファイルの内容は、ベンダに特有の自社開発のハードウェアに密接に関連する。1つの試験システムにおいてパターンオブジェクトファイルを結合して、種々のベンダの自社開発のハードウェアで正常に機能するようにするための標準規格は存在しない。さらに、モジュールのベンダは、自社開発のフォーマットを公開して共有することを望まないため、汎用のパターンオブジェクトフォーマットを開発するのは難しい。さらに、共通のフォーマットを強要することは、余分なオーバーヘッドを生み出す可能性があり、パターンの使用効率を低下させることに繋がる可能性がある。結果として、種々のモジュールベンダの間でパターンデータが用いられるようにするための共通のオブジェクトファイルフォーマットは存在しない。モジュールベンダは、パターンソースファイルをコンパイルして、パターンオブジェクトファイルを生成するために、自社のパターンコンパイラを提供しなければならない。これにより、或るベンダが異なるベンダの試験装置をインストールすることに決める度に、パターンファイルを翻訳するという一連の面倒な作業を行ってきた。これらの不都合な点を克服することが望ましい。

【発明の開示】

【発明が解決しようとする課題】

【0004】

それゆえ、オープンアーキテクチャのモジュール式試験システムが必要とされている。詳細には、モジュール式試験システム内のパターンオブジェクトファイルを管理するための仕組みが必要とされている。

【課題を解決するための手段】

【0005】

本特許出願は、オブジェクト指向構成体、たとえばC++オブジェクトおよびクラスを用いる試験プログラム開発を記述する。詳細には、この方法は、本発明の譲受人に譲渡される米国特許出願第60/449,622号、第10/404,002号および第10/403,817号に記載されるような、オープンアーキテクチャテストのための試験プログラムを開発するのに適している。

【0006】

本発明の一実施の形態は、汎用のオブジェクト指向、たとえばC/C++構成体において、試験システム資源、試験システム構成、モジュール構成、試験シーケンス、試験計画、試験条件、試験パターンおよびタイミング情報を記述することによって、自動試験装置（ATE）のような半導体検査システムにおいて、被試験デバイス、たとえばICを試験するための試験プログラムを開発するための方法を提供する。これらの記述を含むファイルは、それらのファイルを使用する試験システムまたは関連する装置がアクセスできるメモリ、すなわちコンピュータ読取り可能媒体に記憶される。

【0007】

試験システム資源を記述することは、資源タイプを指定することであって、その資源タイプは、ICに対して1つの試験を実施するための少なくとも1つの試験モジュールに関連付けられる、資源タイプを指定することと、その資源タイプに関連付けられるパラメータタイプを指定することと、及び、そのパラメータタイプのパラメータを指定することとを含んでもよい。

【0008】

試験システム構成を記述することは、少なくとも1つの試験モジュールを制御するためのサイトコントローラを指定することであって、各試験モジュールはICに対して1つの試験を実施する、サイトコントローラを指定することと、及び、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含んでもよい。その試験システムは、指定された入力ポートにおいて、サイトコントローラをモジュール接続イネーブラに接続し、モジュール接続イネーブラは、サイトコントローラを1つの試験モジュールに接続する。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実装してもよい。

【0009】

モジュール構成を記述することは、モジュールタイプを指定するためのモジュール識別子を指定することと、モジュール識別子によって指定されるモジュールタイプの試験モジュールを制御するための実行可能コードを指定することと、及び、試験モジュールに関連付けられる資源タイプを指定することとを含んでもよい。実行可能コードは、ダイナミックリンクライブラリの形をとることができる。

10

【0010】

モジュール構成を記述することはさらに、ユーザが、モジュール接続イネーブラの出力ポートを指定するためのスロット識別子を指定することを含んでもよく、試験システムは、その出力ポートにおいて、試験モジュールをモジュール接続イネーブラに接続し、モジュール接続イネーブラは試験モジュールを対応するサイトコントローラに接続する。またユーザは、試験モジュールの供給元を特定するためのベンダ識別子と、資源タイプに関連して利用することができる資源ユニットの最大数の識別子とを指定してもよい。資源タイプは、たとえばデジタルテストピンおよび資源ユニットテストチャネルであってもよい。別法では、テストチャネル資源ユニットはまた、たとえば、アナログテストピン、RFテストピン、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資源タイプに対応してもよい。どの資源ユニットが使用禁止にされるかに関する指標も与えることができる。使用禁止として指示された資源ユニットは、試験モジュールの欠陥のある資源ユニットを表し得る。

20

【0011】

試験条件を記述することは、少なくとも1つの試験条件グループを指定することと、少なくとも1つの変数を含む仕様セットを指定することと、及び、変数に結び付けられるべき式を選択するためのセレクトラを指定することとを含んでもよい。試験条件グループをその仕様セットのためのセレクトラに関連付けることにより、試験条件が規定される。

30

【0012】

試験シーケンスを記述することは、種々の試験を実施することができる順序（またはフロー）を指定することを含んでもよい。

【0013】

試験パターンを記述することは、試験パターン、関連する電圧および電流レベル、信号値の遷移、対応する立ち上がりおよび立ち下がり時間、ならびに関連するタイミングを指定することを含んでもよい。

【0014】

本発明の一実施の形態は、プリヘッダファイルを使用することを含む。プリヘッダファイルはコンパイルされて、試験実体に関連するクラスのためのヘッダファイルを作成する。プリヘッダは、試験実体の少なくとも1つの属性を設定するためのパラメータを指定するためのパラメータブロックと、コンパイラによって試験実体クラスのため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ためのテンプレートブロックとを含む。ヘッダファイルはC++ヘッダファイルであってもよい。たとえば、試験実体は1つの試験であってもよく、試験実体クラスは1つの試験クラスであってもよい。パラメータは、たとえばパターンリストおよび試験条件に関連してもよい。

40

【0015】

一実施の形態では、モジュール式試験システム内のパターンオブジェクトファイルを管

50

理するための方法は、モジュール式試験システムを配設することを含み、モジュール式試験システムは少なくとも1つのサイトコントローラを制御するためのシステムコントローラを備え、少なくとも1つのサイトコントローラは少なくとも1つの試験モジュールおよびその対応する被試験デバイス（DUT）を制御する。本方法はさらに、ベンダ供給パターンコンパイラとモジュール式試験システムとの間に標準インターフェースを確立するためのオブジェクトファイル管理フレームワークを作成すること、サイトコントローラにおいてパターンソースファイルを受信すること、オブジェクトファイル管理フレームワークを用いて、パターンソースファイルに基づいてパターンオブジェクトメタファイルを作成すること、及び、パターンオブジェクトメタファイルを用いて試験モジュールを通して被試験デバイスを試験することを含む。一実施の形態では、パターンコンパイラであって、少なくとも1つのモジュール特有のパターンコンパイラと、各モジュール特有のコンパイラに対し、パターンソースファイルの対応するモジュール特有のセクションおよび該パターンソースファイルの共通セクションの両方をコンパイルするように指示するためのオブジェクトファイルマネージャと、パターンオブジェクトメタファイルであって、共通ヘッダセクションと、少なくとも1つのモジュール特有のパターンデータセクションとを含むパターンオブジェクトメタファイルとを含む、パターンコンパイラを提供する。共通セクションは、全てのモジュール特有のコンパイラにアクセスできる情報を含んでよい。パターンコンパイラは、モジュール特有のパターンデータを、実行するための対応する試験モジュールにロードするための少なくとも1つのモジュール特有のパターンローダをさらに備えてよい。パターンオブジェクトメタファイルは、パターンソースファイル内の非サイクルベースパターンブロックを表すための1つまたは複数のパターンブロックと、パターンソースファイル内のサイクルベースパターンブロックを表すための多くても1つのサイクルライズパターンブロックとをさらに含んでよい。

【0016】

本発明の上記の特徴および利点、ならびに本発明のさらに別の特徴および利点は、添付の図面とともに取り上げられるときに、本発明の実施形態の詳細な説明の結果として、以下においてさらに明確に理解されるであろう。

【発明を実施するための最良の形態】

【0017】

モジュール式試験システム内のパターンオブジェクトファイルを管理するための方法およびシステムが提供される。以下の説明は、当業者の誰もが本発明を実施し、使用できるようにするために提示される。特定の実施形態および応用形態の説明は、例としてのみ提供される。本明細書において説明される例の種々の変更および組み合わせが当業者には容易に明らかになるはずであり、本発明の精神および範囲から逸脱することなく、本明細書において規定される一般原理は、他の例および応用形態にも適用することができる。したがって、本発明は説明および図示される例に限定されることを意図するのではなく、本明細書に開示される原理および特徴と合致する最も広い範囲を与えられるべきである。

【0018】

本発明は包括的には、同じ譲受人による米国特許出願第60/449,622号、第10/404,002号および第10/403,817号に開示されるようなオープンアーキテクチャ試験システムの観点から説明される。しかしながら、本発明の試験プログラム開発システムおよび方法の実施形態が、オープンテストアーキテクチャだけでなく、固定テストアーキテクチャにも同様に当てはまることは当業者には認識されよう。

【0019】

オープンアーキテクチャ試験システムに関する記述は、米国特許出願第10/772,372号「Method and Apparatus for Testing Integrated Circuits」において見ることができ、その特許出願は、同じ譲受人による米国特許出願第60/449,622号の恩典を主張する。

【0020】

図1は、如何に信号が生成され、被試験デバイス（DUT）に加えられるかを示す、従

10

20

30

40

50

来のテストの一般化されたアーキテクチャを示す。各DUT入力ピンが、試験データを加えるドライバ2に接続され、一方、各DUT出力ピンはコンパレータ4に接続される。大抵の場合に、トライステートドライバ-コンパレータが用いられ、各テストピン（チャンネル）が入力ピン、出力ピンのいずれかとして動作できるようにする。単一のDUTのために専用に設けられるテストピンは合わせて試験サイトを形成し、その試験サイトは、関連するタイミング発生器6、波形発生器8、パターンメモリ10、タイミングデータメモリ12、波形メモリデータ14およびデータ速度を規定するブロック16によって動作する。

【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を示す。システムコントローラ（SysC）102が多数のサイトコントローラ（SiteC）104に接続される。システムコントローラは、ネットワークにも接続され、ファイルにアクセスすることができる。試験サイト110に配置される1つまたは複数の試験モジュール108を制御するために、各サイトコントローラは、モジュール接続イネーブラ106を通して、試験モジュール108に接続される。モジュール接続イネーブラ106は、接続されるハードウェアモジュール108の構成を変更できるようにするとともに、データ転送用（パターンデータのロード用、応答データの収集用、制御用など）のバスとしての役割も果たす。実現可能なハードウェアの実装形態は、専用接続、交換接続、バス接続、リング接続およびスター接続を含む。モジュール接続イネーブラ106は、たとえばスイッチマトリクスとして実装することができる。各試験サイト110は1つのDUT112に関連付けられ、DUT112は、ロードボード114を通して、対応するサイトのモジュールに接続される。1つの実施形態では、単一のサイトコントローラを多数のDUTサイトに接続することができる。

【0022】

システムコントローラ102は、システム全体のマネージャとしての役割を果たす。システムコントローラ102は、サイトコントローラの動作状態を調整し、システムレベルの並列試験方法を管理し、さらに、ハンドラ/プローブを制御し、システムレベルのデータロギングおよび誤り処理をサポートする。動作環境によっては、システムコントローラ102は、サイトコントローラ104の動作とは別のCPU上に配置することができる。別法では、システムコントローラ102およびサイトコントローラ104は、共通のCPUを共有することができる。同様に、各サイトコントローラ104は、その専用のCPU（中央演算装置）上に配置することができるか、あるいは同じCPU内の別個のプロセスまたはスレッドとして配置することができる。

【0023】

個々のシステムコンポーネントを、集積されたモノリシックシステムの論理的なコンポーネントを見なすことができ、必ずしも1つの分散処理システムの物理的なコンポーネントではないことを前提とすると、図2に示されるシステムアーキテクチャは概念的には分散処理システムと考えることができる。

・別の実施形態では、システムコントローラおよびサイトコントローラは1つまたは複数のコンピューティングデバイスで実装することができる。各コンピューティングデバイスは、

・種々の基本システムサービスを取り扱うための手順、およびハードウェアに依存するタスクを実行するための手順を含むオペレーティングシステム；

・オペレーティングシステムと、パターンオブジェクトファイルを管理するためのアプリケーションプログラムのような、試験システムの他のアプリケーションプログラムとの間のインターフェースを構成するためのアプリケーション層；

・オペレーティングシステムおよびアプリケーションプログラムからの命令を実行するためのプロセッサユニット；および

・試験システムのデータを記憶するためのメモリユニットを含むことができ、メモリユニットは、ランダムアクセスメモリ、不揮発性メモリまたは大容量記憶装置を含むことがで

10

20

30

40

50

きる。

【0024】

アプリケーションプログラムは、実行可能手順、サブモジュール、テーブルおよび他のデータ構造を含むことができる。他の実施形態では、付加的な、または異なるモジュールおよびデータ構造を用いることができ、先に記載されたモジュールおよび／またはデータ構造のうちのいくつかが用いられない場合もある。アプリケーションプログラムはソフトウェアおよび／またはハードウェアにおいて実装することができる。ハードウェアで実装するとき、それらのアプリケーションプログラムは、特定用途向け集積回路（ASIC）で、またはフィールドプログラマブルゲートアレイ（FPGA）で実装することができる。

10

【0025】

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を示す。ソフトウェアアーキテクチャ200は分散オペレーティングシステムを表しており、関連するハードウェアシステム要素102、104、108に対応する、システムコントローラ220、少なくとも1つのサイトコントローラ240および少なくとも1つのモジュール260のためのコンポーネントを有する。モジュール260に加えて、アーキテクチャ200は、ソフトウェア内のモジュールエミュレーション280のための対応するコンポーネントを含む。

【0026】

1つの例示的な選択として、このプラットフォームのための開発環境はマイクロソフトウィンドウズを基にすることができる。このアーキテクチャを用いることによって、プログラムおよびサポートの可搬性に関して副次的な利点がもたらされる（たとえば、フィールドサービスエンジニアが、テストオペレーティングシステムを実行するラップトップを接続して、高度な診断を実行することができる）。しかしながら、大量のコンピュータ集約的な作業（たとえば、テストパターンコンパイル）を行うために、関連するソフトウェアは、分散プラットフォーム上でジョブをスケジューリングできるようにするために個別に実行することができる個別の実体として構成することができる。したがって、バッチジョブのための関連するソフトウェアツールは、多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。

20

【0027】

1つの例示的な選択として、ANSI/ISO標準C++を、ソフトウェアのためのネイティブ言語とすることができる。当然、第三者が自ら選択した別の言語を用いるシステムに組み込むことができるようにする多数のオプションが（名目的なC++インターフェース上に層を設けるために）利用可能である。

30

【0028】

図3は、テストオペレーティングシステム、ユーザコンポーネント292（たとえば、試験を実施する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（たとえば、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ストラクチャとして供給される）、モジュール開発コンポーネント296（たとえば、モジュール開発者によって供給される）および外部コンポーネント298（たとえば、モジュール開発者以外の外部供給元によって供給される）を含む、名目的な供給元によって（またはサブシステムとしてまとめて開発するものとして）編成された構成に基づくコンポーネントの陰影を示す。

40

【0029】

供給元に基づいて編成されたという見方をすると、テストオペレーティングシステム（TOS）インターフェース290は、システムコントローラ／サイトコントローラインターフェース222と、フレームワーククラス224と、サイトコントローラ／モジュールインターフェース245と、フレームワーククラス246と、既定モジュールレベルインターフェースと、バックプレーン通信ライブラリ249と、シャースロットIF（インターフェース）262と、ロードボードハードウェアIF264と、バックプレーンシミ

50

ュレーション I F 2 8 3 と、ロードボードシミュレーション I F 2 8 5 と、D U T シミュレーション I F 2 8 7 と、D U T の Verilog モデルのための Verilog PLI（プログラミング言語インターフェース）2 8 8 と、D U T の C/C++ モデルのための C/C++ 言語サポート 2 8 9 とを含む。

【0 0 3 0】

ユーザコンポーネント 2 9 2 は、ユーザ試験計画 2 4 2 と、ユーザ試験クラス 2 4 3 と、ハードウェアロードボード 2 6 5 と、D U T 2 6 6 と、D U T Verilog モデル 2 9 3 と、D U T C/C++ モデル 2 9 1 とを含む。

【0 0 3 1】

システムコンポーネント 2 9 4 は、システムツール 2 2 6 と、通信ライブラリ 2 3 0 と、試験クラス 2 4 4 と、バックプレーンドライバ 2 5 0 と、HW バックプレーン 2 6 1 と、シミュレーションフレームワーク 2 8 1 と、バックプレーンエミュレーション 2 8 2 と、ロードボードシミュレーション 2 8 6 とを含む。

【0 0 3 2】

モジュール開発コンポーネント 2 9 6 は、モジュールコマンドインプリメンテーション 2 4 8 と、モジュールハードウェア 2 6 3 と、モジュールエミュレーション 2 8 4 とを含む。

【0 0 3 3】

外部コンポーネント 2 9 8 は外部ツール 2 2 5 を含む。

【0 0 3 4】

システムコントローラ 2 2 0 は、サイトコントローラへのインターフェース 2 2 2 と、フレームワーククラス 2 2 4 と、システムツール 2 2 6 と、外部ツール 2 2 5 と、通信ライブラリ 2 3 0 とを含む。システムコントローラソフトウェアはユーザのための一次的なインタラクションポイントである。それは、本発明のサイトコントローラへのゲートウェイと、同じ譲受人による米国特許第 6 0 / 4 4 9 , 6 2 2 号に記載されるようなマルチサイト/D U T 環境におけるサイトコントローラの同期とを提供する。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ーフェース（G U I）に基づくものでも、そうでないものでも、システムコントローラ上で実行される。システムコントローラは、試験計画、試験パターンおよび試験パラメータファイルを含む、全ての試験計画関連情報のためのリポジトリとしての役割も果たす。これらのファイルを記憶するメモリは、システムコントローラに対してローカルに存在することができるか、またはオフライン、すなわち或るネットワークを通してシステムコントローラに接続されることができる。試験パラメータファイルは、本発明の一実施形態のオブジェクト指向環境内にある試験クラスのためのパラメータ化データを含む。

【0 0 3 5】

第三者の開発者は、標準システムツール 2 2 6 に加えて（またはその代わりに）、複数のツールを提供することができる。システムコントローラ 2 2 0 上の標準インターフェース 2 2 2 は、テストおよび試験オブジェクトにアクセスするためにそれらのツールが用いるインターフェースを含む。ツール（アプリケーション）2 2 5、2 2 6 によって、試験およびテストオブジェクトをインタラクティブに、かつバッチで制御できるようになる。それらのツールは、自動能力（たとえば、S E C S / T S E M を用いること等による）を提供するためのアプリケーションを含む。

【0 0 3 6】

システムコントローラ 2 2 0 上に存在する通信ライブラリ 2 3 0 は、ユーザアプリケーションおよび試験プログラムに対してトランスペアレントであるように、サイトコントローラ 2 4 0 と通信するための仕組みを提供する。

【0 0 3 7】

システムコントローラ 2 2 0 に関連するメモリ内に存在するインターフェース 2 2 2 は、システムコントローラ上で実行されるフレームワークオブジェクトへのオープンインターフェースを提供する。サイトコントローラに基づくモジュールソフトウェアがパターン

10

20

30

40

50

データにアクセスし、それらを検索できるようにするインターフェースが含まれる。また、テストおよび試験オブジェクトにアクセスするためにアプリケーションおよびツールが用いるインターフェース、ならびにスクリプト用エンジンを通してテストおよび試験コンポーネントにアクセスし、それらを操作する能力を提供するスクリプト用インターフェースも含まれる。これにより、インタラクティブ、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仕組みが、それらの機能を実行できるようになる。

【0038】

システムコントローラ220に関連付けられるフレームワーククラス224は、これらの上記のオブジェクトと対話するための仕組みを提供し、標準インターフェースの参照インプリメンテーションを提供する。たとえば、本発明のサイトコントローラ240は機能試験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、機能試験オブジェクトのリモートシステムコントローラに基づく代理として、対応する機能試験プロキシを提供することができる。こうして、システムコントローラ220上のツールが、標準機能試験インターフェースを利用できるようになる。フレームワーククラスは実効的には、ホストシステムコントローラに関連付けられるオペレーティングシステムを提供する。また、それらのフレームワーククラスは、サイトコントローラへのゲートウェイを提供し、かつマルチサイト/DUT環境におけるサイトコントローラの同期を提供するソフトウェアコンポーネントを構成する。こうして、この層は、通信層で直に取り扱うことを必要とすることなく、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た、本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。

【0039】

サイトコントローラ240は、ユーザ試験計画242、ユーザ試験クラス243、標準試験クラス244、標準インターフェース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ーフェース（すなわち、既定モジュールレベルインターフェース）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のホストとして機能する。試験機能の大部分がサイトコントローラ104/240によって取り扱われ、それにより試験サイト110が個別に動作できるようにすることが好ましい。

【0040】

試験計画242はユーザによって記述される。その計画は、C++のようなオブジェクト指向構成体を用いて、標準的なコンピュータ言語において直に記述することができるか、または、C++コードを生成するための、さらに高いレベルの試験プログラミング言語において記述することができ、後に実行可能な試験プログラムにコンパイルすることができる。試験プログラムを開発するために、本発明の1つの実施形態は、譲受人の発明による試験プログラム言語（TPL）コンパイラを利用する。図4を参照すると、試験プログラムコンパイラ400が部分的には、翻訳プログラムセクション402を含むコード生成プログラムとしての役割を果たし、試験および関連するパラメータを記述する、試験プログラム開発者のソースファイル404を、C++コードのようなオブジェクト指向構成体に翻訳する。さらに、コンパイラセクション406が、コードをコンパイルして、そのコードを実行可能ファイル、たとえばDLLにリンクし、テストシステムが実行することができる試験プログラムを生成する。TPLコード生成プログラム/翻訳プログラムを試験システムに適用することには新規性があるが、コード生成プログラムは当該技術分野において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には、当該技術分野において知られている標準的なC++コンパイラを用いることができる。

【0041】

試験計画は、その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および/または標準的な、またはユーザによって供給される試験クラス244を用いることにより試験オブジェクトを作成し、標準インターフェース245を用いてハードウェアを構成し、試験計画フローを規定する。また試験計画は、試験計画の実行中に必要とされる任意の付加的なロジックを提供する。試験計画はいくつかの基本的なサービスをサポートし、デ

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

【0042】

試験プログラムコンパイラ400に入力されるソースコードは、1つの試験計画において用いられるオブジェクト、およびその互いに対する関係を指定する試験計画記述ファイルを含む。このファイルは、標準インター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++コードに翻訳され、それはITestPlanで表すことができる。このコードは、ウインドウズ・ダイナミックリンクライブラリ（DLL）にパッケージングされ、サイトコントローラにロードされることができる。サイトコントローラソフトウェアが試験計画オブジェクトを生成し、それが含む試験計画オブジェクトを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、試験プログラムDLLが生成される。サイトコントローラソフトウェアは、その試験プログラムDLLを、その処理空間内にロードし、それらのエントリポイントのうちの1つを用いて、試験計画オブジェクトのインスタンスを作成する。一旦、試験計画オブジェクトが作成されたなら、その後、サイトコントローラソフトウェアはその試験計画を実行することができる。

10

【0043】

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、共通の試験関連動作を実装する1組のクラスおよびメソッドである。サイトコントローラレベルフレームワークは、たとえば、電源およびピンエレクトロニクスを一定の順序で配列するためのクラス、レベルおよびタイミング条件を設定するためのクラス、測定値を求めるためのクラス、および試験フローを制御するためのクラスを含む。またフレームワークは、ランタイムサービスおよびデバッグのためのメソッドも提供する。フレームワークオブジェクトは、標準インターフェースを実装することによって機能することができる。たとえば、TesterPinフレームワーククラスのインプリメンテーションは、試験クラスがハードウェアモジュールピンと対話するために用いることができる汎用のテストピンインターフェースを実装するように標準化される。

20

【0044】

特定のフレームワークオブジェクトは、モジュールレベルインターフェース247の助けを借りて動作し、モジュールと通信するように実装することができる。サイトコントローラフレームワーククラスは実効的には、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の役割を果たすことができる。

30

【0045】

一般的には、プログラムコードの90パーセント以上がデバイス試験のためのデータであり、コードの残りの10パーセントは試験方法を実現する。デバイス試験データは、DUTに依存する（たとえば、電源条件、信号電圧条件、タイミング条件など）。試験コードは、指定されたデバイス条件をATEハードウェアにロードするための方法、およびユーザによって指定された目的（データロギングなど）を実現するために必要な方法からなる。本発明の一実施形態のフレームワークは、ハードウェアに依存しない試験、およびユーザがDUT試験プログラミングのタスクを実行できるようにするテストオブジェクトモデルを提供する。

40

【0046】

試験コードの再利用性を高めるために、そのようなコードは、任意のデバイス固有のデータ（たとえば、ピン名、刺激データなど）、またはデバイス試験固有のデータ（たとえば、DCユニットのための条件、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）とは無関係に作成することができる。1つの試験のためのコードがこれらのタイプのデータとコンパイルされる場合には、試験コードの再利用性が減少するであろう。それゆえ、本発明の一実施形態によれば、任意のデバイス固有のデータまたはデバイス試験固有のデータを、コード実行時間中の入力として、外部の試

50

験コードが利用できるようにする。

【0047】

本発明の一実施形態では、試験クラスは、標準試験インターフェースの1つのインプリメンテーションであり、ここではITestとして表され、特定のタイプの試験のための試験データおよびコードの分離（それゆえ、コードの再利用）を実現する。そのような試験クラスは、それ自体の別個のインスタンスのための「テンプレート」と見なすことができ、それらのインスタンスは、デバイス固有データおよび／またはデバイス試験固有データだけが互いとは異なる。試験クラスは、試験計画ファイルにおいて指定される。各試験クラスは通常、特定のタイプのデバイス試験、またはデバイス試験のための設定を実装する。たとえば、本発明の一実施形態は、DUTのための全ての機能試験のための基本クラスとして、ITestインターフェースの特定のインプリメンテーション、たとえばFunctionalTestを提供することができる。それは、試験条件を設定する機能、パターンを実行する機能、およびストローブに失敗した場合に試験のステータスを判定する機能からなる基本機能を提供する。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として表される、ACおよびDC試験クラスを含むことができる。

10

【0048】

全試験タイプは、いくつかの仮想的なメソッド（たとえば、init()、preExec()およびpostExec()）のデフォルトインプリメンテーションを提供することができる。これらのメソッドは、試験技師がデフォルトの挙動を無効にし、任意の試験固有のパラメータを設定するためのエントリポイントになる。しかしながら、試験計画において、カスタム試験クラスを用いることもできる。

20

【0049】

試験クラスによって、ユーザは、その試験の特定のインスタンスのためのオプションを指定するために用いられるパラメータを与えることにより、クラス挙動を構成できるようになる。たとえば、機能試験は、パラメータPリストおよびTestConditionを受け取り、それぞれ、実行すべきパターンリスト、およびその試験のためのレベルおよびタイミング条件を指定することができる。これらのパラメータのために種々の値を指定することによって（試験計画記述ファイルにおいて種々の「試験」ブロックを用いることによる）、ユーザは機能試験の種々のインスタンスを作成できるようになる。図5は、単一の試験クラスから種々の試験インスタンスを如何に導出することができるかを示す。これらのクラスは、C++コードのような、オブジェクト指向構成体において直にプログラミングすることができるか、または試験プログラムコンパイラが試験計画ファイルから試験の記述およびそのパラメータを取り込み、対応するC++コードを生成できるようにし、その後、対応するC++コードをコンパイルし、リンクして、試験プログラムを生成することができるように設計することができる。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のライブラリとして用いることができる。このライブラリはテストのユーザから見ることで、ユーザは、たとえば、ある試験クラスのインプリメンテーションを変更し、ユーザ定義の試験クラスを作成することができる。

30

【0050】

ユーザによって開発された試験クラスに関しては、そのシステムの一実施形態は、全ての試験クラスが単一の試験インターフェース、たとえばITestから派生し、結果として、標準的な1組のシステム試験クラスと同じようにして、フレームワークがそれらの試験クラスを操作できるようになるという点で、そのような試験クラスをフレームワークに組み入れることをサポートする。これらの付加的なファシリティを利用するために試験プログラムにおいてカスタムコードを使用しなければならないことを前提とした上で、ユーザは付加機能をその試験クラスに自由に取り入れることができる。

40

【0051】

各試験サイト110は、1つまたは複数のDUT106を専用に試験し、構成変更可能な一群の試験モジュール112を通して機能する。各試験モジュール112は、特定の試験タスクを実行する実体である。たとえば、試験モジュール112には、DCT電源、ピ

50

ンカード、アナログカードなどを用いることができる。このモジュールによる手法は、高い自由度と構成可能性とを提供する。

【0052】

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が提供することができ、ベンダによって選択されるコマンドインプリメンテーション方法に応じて、ハードウェアモジュールのためのモジュールレベルインターフェース、または標準インターフェースのモジュール固有のインプリメンテーションを実装する。これらのクラスの外部インターフェースは、既定モジュールレベルインターフェース要件、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。この層は、標準的な1組の試験コマンドの拡張も可能にし、メソッド（関数）およびデータ要素を追加できるようにする。

10

【0053】

バックプレーン通信ライブラリ249は、バックプレーンにわたる標準的な通信のためのインターフェースを提供し、それにより試験サイトに接続されるモジュールと通信するために必要な機能を提供する。これにより、ベンダ固有のモジュールソフトウェアが、バックプレーンドライバ250を用いて、対応するハードウェアモジュールと通信できるようになる。バックプレーン通信プロトコルはパケットに基づくフォーマットを用いることができる。

【0054】

テストピンオブジェクトは、物理的なテストチャネルを表し、ここではITesterPinとして表される、テストピンインターフェースから派生する。本発明の一実施形態のソフトウェア開発キット（SDK）は、TesterPinとも呼ばれる場合がある、ITesterPinのデフォルトインプリメンテーションを提供し、それは既定モジュールレベルインターフェース、IChannelという形で実装される。ベンダは、IChannelという形でモジュールの機能を実装することができる場合には、自由にTesterPinを利用することができる。そうでない場合には、ベンダは、そのモジュールで正常に機能するための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。

20

【0055】

本発明のテストシステムによって提供される、ここではIModuleとして表される標準モジュールインターフェースは包括的には、ベンダのハードウェアモジュールを表す。種々のベンダが種々のモジュールを提供することができる。また、1つのベンダが複数の異なるモジュールを提供する場合もある。ベンダ供給の、そのシステムのためのモジュール固有のソフトウェアは、ダイナミックリンクライブラリ（DLL）のような実行可能ファイルの形で提供されることができる。1つのベンダからのモジュールタイプ毎のソフトウェアは、単一のDLL内にカプセル化することができる。そのような各ソフトウェアモジュールは、モジュールソフトウェア開発のためのAPIを含む、モジュールインターフェースコマンドのためのベンダ固有のインプリメンテーションを提供するための役割を担う。

30

【0056】

モジュールインターフェースコマンドには2つの側面がある。第一に、それらのコマンドは、ユーザがシステム内の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ーフェースとしての役割を果たし、第二に、それらのコマンドは、第三者の開発者が、自らのモジュールをサイトコントローラレベルフレームワークに組み込むために利用することができるインター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ーフェースコマンドは、2つのタイプに分類される。

40

【0057】

最初の、そして最も明らかなコマンドは、フレームワークインターフェースを通してユーザが見ることができる「コマンド」である。こうして、テストピンインターフェース（ITesterPin）は、レベルおよびタイミング値を入手し、設定するための方法を提供し、一方、電源インターフェース（IPowerSupply）は、たとえばパワーアップおよびパワーダウンするための方法を提供する。

50

【0058】

さらに、フレームワークは既定モジュールレベルインターフェースの特殊なカテゴリを提供し、それは、モジュールと通信するために用いることができる。これらは、ベンダのモジュールと通信する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ーフェース（すなわち、フレームワークインターフェースの「標準的な」インプリメンテーション）である。

【0059】

しかしながら、第2の側面、すなわちモジュールレベルインターフェースを用いることはオプションである。それを用いることの利点は、モジュールレベルインターフェースを実装することによってハードウェアに送出される特定のメッセージの内容に重点を置きながら、ベンダがITesterPinおよびIPowerSupplyなどのクラスのインプリメンテーションを利用できることである。しかしながら、これらのインターフェースがそのベンダに適していない場合には、それらのベンダは、フレームワークインターフェースのカスタムインプリメンテーション（たとえば、ITesterPin、IPowerSupplyなど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することもできる。その際、これらのベンダは、そのハードウェアに適したカスタム機能を提供するであろう。

【0060】

バックグラウンドとしてこのオープンアーキテクチャを用いて、本発明の試験プログラム開発システムが以下にさらに説明される。以下のセクションAは、試験プログラムが用いられることになる試験環境を記述するための規則を説明する。セクションBは、試験プログラム開発のための方法および規則を説明する。セクションCは、試験計画を開発するための方法および規則、ならびに試験プログラムの主要構造を定義する方法を説明する。セクションDは、オープンアーキテクチャ試験システム上で試験プログラムを実行する方法を説明する。セクションEは、試験パターンのための方法および規則を説明する。セクションFは、試験パターンのタイミングを記述するための規則を説明する。セクションGは、テスト動作全体のための規則を説明する。

【0061】

[A. コンポーネント]

試験環境は、テストをインストールし、それを1組の試験を実行するために準備するための必要な条件を指定する1組のファイルを含む。試験環境は、以下に記載されるもののためのファイルを含むことが好ましい。

1. テスタ資源定義：そのオープンアーキテクチャ試験システムにおいて利用することができるテスト資源タイプ、およびそのような資源のためにサポートされるパラメータを指定するためのファイル。

2. テスタ構成：サイトコントローラ、サイトおよび対応するマッピングを指定するためのファイル。

3. モジュール構成：各サイト内のハードウェアモジュールを指定するためのファイル。

4. ピン記述：信号ピン、電源のようなDUTピンを命名し、ピングループを記述するためのファイル。

5. ソケット：DUTピン - テスタピンの割当てを指定するためのファイル。

6. ピンオプション：ピンのための特殊なオプション又はモードを指定するためのファイル。

7. パターンリスト：試験パターンおよびそのシーケンスを指定するためのファイル。

8. パターン：試験ベクトルを指定するためのファイル。

【0062】

上記のファイルのうち、項目1～3はCMD（構成管理データベース）からの情報を用いてICF（インストールおよび構成ファイル）によって作成され、既知の場所において入手可能であるのに対して、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、上記の項目1～6のための説明を提供する。項目7および8はセクションEにおいてさ

らに詳細に説明される。これらのコンポーネントをそれぞれ開発するために、特定の方法および規則が用いられることが好ましい。これらの方法および規則は、例とともにこのセクションにおいて説明されるであろう。

【0063】

[A1. 資源定義]

各ハードウェアモジュールは、試験システムによって用いるための1つまたは複数のタイプのハードウェア資源（略して資源）を提供する。テスト資源定義は、利用可能な資源タイプのための1組の資源名、並びにそれぞれの特定の資源タイプに関連する1組のパラメータ名およびタイプを宣言するために用いられることが好ましい。たとえば、資源名 `dpin` は、デジタルテストピンを指すために用いられる。これらの資源は、`VIL`（入力低電圧用）、`VIH`（入力高電圧用）、`VOL`（出力低電圧用）、`VOH`（出力高電圧用）などのパラメータを有する。資源定義ファイルは、「. rsc」の拡張子を有するであろう。以下に示されるのは、いくつかのテスト資源を含む、資源定義の一例である。

【0064】

```
#
# 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;
}
}

```

【0065】

資源パラメータのタイプ（VoltageまたはTimeなど）は標準的な工学単位であることが好ましいことに留意されたい。異なるパラメータを指定することが望ましい特定用途の資源を供給するベンダは、自らの資源定義ファイルを提供すべきである。

【0066】

[資源定義のための構造]

以下に与えられるのは、本発明の好ましい実施形態による資源定義ファイルのための構造である。

【0067】

```

resource-file:
  version-info resource-defs
version-info:
  Version version-identifer;
resource-defs:
  ResourceDefs {resource-def-list}
resource-def-list:
  resource-def
  resource-def-list resource-def
resource-def:
  resource-name {resource-params-decl-list}
resource-params-decl-list:
  resource-params-decl
  resource-params-decl-list resource-params-decl
resource-params-decl:
  elementary-type-name resource-param-name-list;
resource-param-name-list:
  resource-param-name
  resource-param-name-list , resource-param-name

```

【0068】

上記の未定義の非終端記号は以下のように指定される。

1. version-identifer: 集合 [0 - 9 a - z A - Z .] からの1つまたは複数の文字からなる文字列。それはバージョン番号を表す。

2. resource-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列であり、数字で始まらない。それはdpinまたはdpsのような資源の名前を表す。

3. elementary-type-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列であり、数字で始まらない。それは、Voltage(cf.)のような基本タイプ

の名前を表す。

4. resource-param-name: 集合 [a-z A-Z _ 0-9] からの1つまたは複数の文字空なる文字列であり、数字で始まらない。それは、V I Lのような資源パラメータの名前を表す。

【0069】

[A2. テスタ構成]

テスタ構成は、特定のシステム構成内のサイトコントローラ、およびサイトコントローラのスイッチマトリクス入力ポートへの接続を記載するために用いられることが好ましい1組の規則である。本発明の一実施形態のアーキテクチャでは、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、この状況では、スイッチマトリクス接続は、システム内のサイトコントローラのための暗黙の識別子としての役割を果たす（他の構成も可能である）。以下は一般的なテスタ構成の一例である。

【0070】

```
#
# 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;
}
```

【0071】

特定の試験フロアシステムのためのシステム構成はシステムプロファイルの一部であり、システム構成ファイルSys.cfgとして入手することができる。1つの実施形態では、ポート1（上記の例では「127.0.0.0」）に接続されるサイトコントローラは特別なステータスを享受することができ、そのステータスでは、そのサイトコントローラが単独でスイッチマトリクスを構成することに留意されたい。この「特殊な」サイトコントローラはSITEC-1と呼ばれるであろう。また、そのサイトコントローラは内部ネットワークによってシステムコントローラに接続することができるので、この例におけるサイトコントローラアドレスはIPアドレスであることにも留意されたい。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ネットワークに接続することができる。

【0072】

[テスタ構成のための構造]

以下に与えられるのは、本発明の一実施形態によるシステム構成ファイルのための構造である。

【0073】

10

20

30

40

50

```

system-config-file:
    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

```

10

【0074】

20

上記の未定義の非終端記号は以下のように指定される。

1. version-identifer: 集合 [0 - 9 a - z A - Z .] からの 1 つまたは複数の文字からなる文字列。それはバージョン番号を表す。
2. octet: 0 ~ 255 の負でない整数 (10 進表記)。
3. name: 集合 [a - z A - Z _ 0 - 9] からの 1 つまたは複数の文字からなる文字列であり、数字で始まらない。それは、ドメイン修飾されたホスト名内の名前部分を表す。
4. input-port: 10 進表記の負でない整数。

【0075】

[A3. モジュール構成]

モジュール構成によって、テストの物理的な構成、たとえばシステムシャーシ内の各モジュールの物理的な場所およびタイプを指定できるようになる。これは、動的なテストバス構成によって必要とされ、そのテストバス構成によって、テストバスアドレスを物理的なスロットの場所にマッピングできるようになる。この情報は、システム構成の妥当性を検査するためにシステムブートアップ時に行われるハードウェア発見プロセスを可能にする。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定義し、そのスロットは単一のハードウェアモジュールによって占有されることが好ましい。以下に示されるのは、本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構成の一例である。

30

【0076】

```

#
# Module Configuration File, Modules.cfg
#
Version0.0.1;
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.

```

40

50

```

#
Slot1-12, 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                                10
    # 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
    {
        MaxAvailable32;                        # Resource units 1 .. 32.
    }
    Resource analog                                20
    {
        MaxAvailable16;                        # 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 slots16-30, 50, and 61-64.
#
# Slot 16-30, 50, 61-64                                30
{
    Resource dpin
    {
        MaxAvailable32;                        # Max available resource units.
        Disabled 3,30-32;                      # Disabled resources.
    }
    ModuleDriver "module two.dll";
    VendorID 2;
    ModuleID 2;                                40
}
#
# 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;                                50
}

```

```

#
# Resource type dps specifying resource units for a
# Device Power Supply
#
Resource dps
{
    MaxAvailable 4;
    Disabled 1;
}
}

```

10

【0077】

先に記載されたように、1つの実施形態では、スロットは、スイッチマトリクス出力ポートのような、1つのハードウェアモジュールを接続することができるコネクタを指している。各構成定義は、1つまたは複数のスロットに取り付けられる場合があるモジュールについての情報を提供する。構成定義において指定されるVendorIDは、1つのベンダに関連する固有のIDである。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している。1つのテスト構成内に同じModuleIDのいくつかのインスタンスが存在する場合もある。ModuleDriverは、そのモジュールをサービスするためにベンダ供給DLLを指している。最後に、Resourceは、このモジュールによってサービスされるユニットを指しており、資源タイプのための名前を与える。資源名は資源定義ファイルから得られる。

20

【0078】

上記の例は、モジュール構成ファイル内の3つの構成ブロックを記述する。1つの実施形態では、第1の構成ブロック、スロット1～12および32～48はベンダ1によって製造されるモジュールによってサービスされる。このベンダは、そのモジュール、このモジュールタイプを指すための識別子「1」、およびモジュールを制御するためのモジュールドライバライブラリを提供する。このモジュールは2つのタイプの資源ユニットを提供することができ、一方のタイプは資源名「dpin」によって表され、好ましくは全数が32資源ユニット（すなわち「チャンネル」）であり、その全てが利用可能であり、他方のタイプは資源名「analog」によって表され、全数が16資源ユニットであり、9から16までのみ利用可能である。第2および第3の構成ブロックは第1の構成と同じようにして指定される。

30

【0079】

チャンネルが「使用禁止」として指示されるようにする措置は、他の点では依然として機能するモジュールにおいて欠陥のある資源ユニットを特定できるようにすることであることに留意されたい。また、構成ブロックは1つまたは複数のスロット識別子を有することができることに留意されたい。1つのブロックが2つ以上のスロット識別子を有するとき、特定されたスロットはクローンであると言われる。

【0080】

40

モジュール構成ファイル、Module.cfgは、ICM（インストール構成管理システム）によってシステムプロファイルの一部として作成され（ユーザによって提供される試験フロア固有の情報を含む）、周知の場所において入手することができる。ICMは、試験システム、たとえばシステムコントローラ上にローカルに存在することができるか、またはシステムコントローラが接続されるネットワーク上のいずれかの場所に存在することができるユーティリティである。ICMはCMD（構成管理データベース）を管理し、通常は、システム構成に対するハードウェア変更時に更新される。ICMによって、ユーザは、システム、たとえばサイトコントローラおよびモジュールを構成できるようになる。CMDは、それらの構成を記憶するデータベースである。実際のテストの場合、構成／動作ICMが構成ファイル、たとえばモジュール構成、および他のファイルを生成し、それらのフ

50

ファイル、および特定のモジュールDLLのような関連するファイルをテスト上にコピーする。

【0081】

[モジュール構成のための構造]

以下は、好ましい実施形態によるモジュール構成構造である。

【0082】

```

file-contents:
    version-info module-config-def
version-info:
    Version version-identifier; 10
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: 20
    required-config
    required-config-list required-config
required-coiifig:
    VendorID id-code ;
    ModuleID id-code ;
    ModuleDriver file-name ;
    Resource resource-name {max-spec disabled specopt}
max-spec:
    MaxAvailable positive-integer ;
disabled-spec: 30
    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

```

【0083】

上記の未定義の非終端記号は以下のように指定される。

1. version-identifier: 集合 [0 - 9 a - z A - Z .] からの1つまたは複数の文字からなる文字列。ただし、最初の文字は集合 [0 - 9] から選択されなければならない。
2. positive-integer: 集合 [0 - 9] からの1つまたは複数の文字からなる文字列であり、0で始まらない。
3. id-code: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列。
4. resource-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列。ただし、第1の文字は集合 [a - z A - Z] から選択されなければならない。

【0084】

コメントがサポートされる。コメントは「#」文字で始まり、行の終わりまで続く。

【0085】

[A4. ピン記述]

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、ピン記述ファイル内のDUTピンの記述を入手できるようになり、その記述は拡張子 `.pin` を有する。このプレーンテキストファイルは、少なくとも、DUTピン名のリストと、定義されたDUTピン名を利用する、命名されたピングループの初期定義とを含む（それらの定義は、たとえばプログラミングによって、後に変更または追加することができるので「初期」である）。

10

【0086】

このデータ指定を試験計画記述から分離することにより、DUTピン定義を広く再利用できるようになり、そのプロセスを特定の試験計画に関連付けることなく、パターンコンパイラが、ピン記述ファイルからピン名（ベクトル指定において用いられるピン名への参照を決定するために必要とされる）を導出できるようになる。

【0087】

以下に示されるのは、ピン記述ファイルの一例である。

【0088】

```
#
# 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]":
```

20

30

40

50

```

#
Grp1 - CLK - A0 - A3 - BBUS [1] - BBUS [3:4] + A5,
BBUS[5:8]
}
}
Resource dps
{
    vcc1;
    vcc2;
    vcc3;
    Group PSG
    {
        vcc1, vcc2
    }
}
}

```

10

【0089】

コンパイラがピンおよびピングループ定義をLevelなどのため許容パラメータ設定と関連付けられるようにするために、DUTピンおよびピングループ定義は資源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。

20

【0090】

ピン記述についての以下の点に留意されたい。

1. ピングループおよびピンは同じ名前空間を共有し、グローバル（すなわち試験計画）範囲を有する。これらの名前をグローバル範囲にする結果の1つは、異なる資源ブロックにおいて宣言されるときでも、ピンおよびピングループが重複した名前を使用することができないことである。

2. ピン記述ファイルにおいて少なくとも1つの資源定義が必要とされる。

3. 各資源において少なくとも1つのピン名が定義されなければならない。

4. ピンおよびピングループ名は資源境界内で固有であることが要求される。

5. 2つ以上の資源のために同じピンまたはグループ名を定義することができる。しかしながら、同じ資源内に同じものがある場合には無視される。

30

6. 1つのグループ定義内に現れる全てのピン名およびグループ名はその資源内で既に定義されていなければならない。

7. もし与えられる場合には、グループ定義は少なくとも1つのピン名またはグループ名を持たなければならない（すなわち、グループ定義は空であることはできない）。

8. ピングループ定義は、予め定義されたピングループへの参照を含むことができる。

9. ピングループ定義は、予め定義されたピンおよび／またはピングループの追加および削除のような集合演算を含むことができる。

【0091】

[ピン記述のための構造]

40

以下に与えられるのは、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である。

【0092】

```

pin-description-file:
    version-info pin-description
version-info:
    Version version-identifier;
pin-description:
    PinDescription {resource-pins-def-list}
resource-pins-def-list:

```

50

```

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

```

10

【0093】

上記の未定義の非末端部は以下のように指定される。

20

1. version-identifier: 集合 [0 - 9 a - z A - Z .] からの1つまたは複数の文字からなる文字列。それはバージョン番号を表す。

2. resource-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列であり、数字で始まらない。それは d p i n または d p s のような資源の名前を表す。

3. pin-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列であり、数字で始まらない。それは、ピン A 0 の名前を表す。

4. pin-gourp-name: 集合 [a - z A - Z _ 0 - 9] からの1つまたは複数の文字からなる文字列であり、数字で始まらない。それは、ピングループ A B U S の名前を表す。

5. index: 負でない整数。それは、関連するピンのグループの下限または上限を表す。

30

【0094】

[A 5 . ソケット]

ソケットは、D U T ピン名と物理的なテストピン（チャンネル）割当てとの間のマッピングを指定する（物理的なテストチャンネル番号はモジュール構成ファイルにおいて定義される）。異なるソケットを用いて、異なる D U T パッケージおよび異なるロードボード構成などをサポートすることができることに留意されたい。マルチ D U T システムの場合、D U T / チャンネル割当てのためのソケット定義は、基本ソケットの多数のサイトへの「クローニング」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、同じ論理ピンのための異なる物理的なマッピング）はサイトモジュール区分を考慮すべきである。したがって、D U T ピンをテストチャンネル割当てに与えるに加えて、ソケットは実効的にはサイト区分も定義する。こうして、ソケットファイルは、いくつかの個別のサイトソケットのための定義を含むことができる。以下に示されるのは、3つの D U T サイトを定義するソケットファイルのサンプルである。

40

【0095】

```

Version 1.1.3
SocketDef
{
  DUTType CHIP3
  {

```

50

```

PinDescription dutP3.pin; # The pin description file for CHIP3
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
    {
        SIotOffset 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 CHIP3
DUTType 74LS245
{
    PinDescription dutLS.pin;
    DUT 3 disabled # This DUT site is disabled, and will be ignored
    {
        . . .
    }
} # End DUTType 74LS245
} # End SocketDef

```

【0096】

ソケットファイルについて以下の点に留意されたい。

1. ソケットファイルは、モジュール構成ファイル、および所与のDUTタイプのためのユーザのピン記述ファイルの両方からの情報を用いる（上記の例のピン記述の場合の指定を参照されたい）。モジュール構成情報は、ソケットファイルコンパイラが暗黙のうちに利用することができる。ソケットファイルコンパイラは、パターンコンパイラの一部を構成し、ソケットDUT名からテストチャネルへのマッピング、モジュール構成およびピン記述ファイルを読み出し、分析して、テストピンのパターンコンパイラによって用いられるDUTピンへのマッピングを設定する。

2. DUTタイプ当たり少なくとも1つのDUTサイト定義が必要とされ、それは、SlotOffset構文ではなく、フルスペック構文を用いなければならない。同じDUTタイプに

対して2つ以上のDUTサイト定義が与えられる場合には、第1のサイト定義はフルスペック構文を用いなければならない。

3. 後続の各DUTサイト定義(同じDUTタイプ)は、フルスペック構文、SlotOffset構文のいずれかを用いることができるが、両方は用いることができない。これにより、個々のサイトが、標準パターンから逸脱できるようになる(たとえば、動作不能のチャンネルに起因する)。

4. SlotOffset構文から導出される結合は、そのDUTタイプのために定義される第1のサイト(フルスペック構文を用いる)に対して定義される。

5. DUTサイトは、実際の物理的な順序で宣言される必要はない。これにより、第1の(物理的な)サイトがそのパターンから逸脱する状況が許される。

6. DUTサイトIDは、ソケット全体にわたって(すなわち、その中で定義される全てのDUTタイプにわたって)固有であることが要求される。

7. DUTサイト定義毎に少なくとも1つの資源定義が必要とされる。

8. サイト定義を、モジュール構成とともに用いて、試験構成が単一サイト/単一DUTであるか、単一サイト/多数DUTであるかを判定しなければならない。

9. 全ての場合に、ソケットファイルは、ピン記述ファイルおよびモジュール構成ファイルと一致する1組のDUTチャンネルマッピングを指定しなければならない。

10. 場合によっては、1つまたは複数のDUTチャンネルがテストから切断されることをソケットが指定できるようにすることが望ましいであろう(たとえば、割り当てられた物理チャンネルを特殊なID「0.0」を有するチャンネルと指定することによる)。この場合、試験プログラムの関連で、これらのDUTチャンネルを使用し、参照することができる。そのようなチャンネル上での動作の結果として、システム警告が生成されるであろう(ただし、エラーではない)。ロード時に、切断されたチャンネルのためのパターンデータは破棄されるであろう。

【0097】

[ソケットのための構造]

以下は、本発明の好ましい実施形態によるモジュール構成のための構造である。

【0098】

socket-file:

version-info socket-def

version-info:

Version version-identifier;

socket-def:

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

device-specific-socket-def-list:

device-specific-socket-def

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

device-specific-socket-def:

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

pin-description-file:

PinDesc pin-description-file-name ;

dut-info-list:

dut-info

dut-info-list dut-info

dut-info:

DUT dut-id {site-controller-input-port 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

```

10

【0 0 9 9】

上記の未定義の非終端記号は以下のように指定される。

1. version-identifier: 集合 [0 - 9 a - z A - Z .] からの1つまたは複数の文字からなる文字列。それはバージョン番号を表す。

20

2. Dut-type-name: 集合 [0 - 9 a - z A - Z .] からの1つまたは複数の文字からなる文字列。ただし、最初の文字は集合 [0 - 9] から選択されてはならない。それは、CHIP 3のようなDUTのタイプを表す。

3. pin-description-file-name: ファイルの簡単な名前であり、そのディレクトリ名は含まないが、全ての拡張子を含む。そのファイル名は、ホストオペレーティングシステムによって認識される構文からなり、引用符に囲まれる場合には、空白および他の文字が許される。

4. switch-matrix-input-port-number: 10進表記の負でない整数であり、サイトコントローラに接続される入力ポートのポート数を表す。

5. dut-id: 10進表記の負でない整数であり、DUTのインスタンスを特定する。

30

6. resource-name: 集合 [0 - 9 a - z A - Z] からの1つまたは複数の文字からなる文字列であり、最初の文字は数字であってはならない。それは資源ファイルにおいて定義される資源の名前を表す。

7. resource-item-name: 集合 [0 - 9 a - z A - Z] からの1つまたは複数の文字からなる文字列であり、最初の文字は数字であってはならない。それは、ピンまたはピングループのような資源ユニットの名前を表す。

8. resource-item-index: 10進表記の負でない整数であり、資源項目のグループのうち特定のメンバを表す。resource-item-index-rangeの関連では、それは、資源項目グループの連続した文字列の下限または上限を表す。

9. resource-unit-index: 10進表記の負でない整数であり、資源ユニット（チャンネル）のグループのうち特定のメンバを表す。resource-unit-index-rangeの関連では、それは、資源ユニットグループの連続した文字列の下限または上限を表す。

40

【0 1 0 0】

[A 6. ピン]

論理的なピン名を物理チャンネルにマッピングすること（たとえばソケットによって提供される）に加えて、テスト資源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。たとえば、試験固有、ベンダ固有、および／または試験システム固有の場合がある、チャンネルのための特定のハードウェア構成を定義するために、複数のオプションを用いることができる。これらのオプションは、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通して入手できるようになるであろう。

50

【0101】

ピンモードオプション定義は、テストチャネルのための特殊なオプションまたはモードの構成をサポートするであろう。たとえば、これを用いて、チャネルを選択し、チャネル多重化を構成することができる。ピンモードオプションはチャネル構成をかなり変更することが必要となる場合もあるので、試験計画初期フローの一部としてのみ用いられることが好ましい。ピンオプション構文は、ベンダによって定義されるオプションをサポートする。一例が以下に示される。

【0102】

```
PinModeOptions
```

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

10

【0103】

[試験環境構成]

先に指摘されたように、資源定義ファイル (Resource.rsc)、システム構成ファイル (Sys.cfg) およびモジュール構成ファイル (Module.cfg) は、「周知の」場所において入手できることが好ましい。この「周知の」場所は、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。たとえば、Tester_ACTIVE_CONFIGS の値がディレクトリ F:\Tester_SYS\configs である場合には、システムは、以下のファイルが存在するものと予想するであろう。

20

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

【0104】

インストール中に、ホストコンピュータ上に存在するインストールおよび構成管理システム (ICM) が、Tester_ACTIVE_CONFIGS の値を設定することが好ましいであろう。ICM が上記のファイルのうちの 1 つの新たなバージョンを作成する度に、その新たなバージョンを、Tester_ACTIVE_CONFIGS によって指示される場所に置くであろう。上記の 3 つのファイルに加えて、シミュレーション構成ファイルのような他のシステム構成ファイルも、Tester_ACTIVE_CONFIGS によって指示される場所に置かれることに留意されたい。

30

【0105】

[B. 試験プログラム開発のための規則]

テストシステムの 2 つの主なエンドユーザ向けコンポーネントのうちの 1 つが試験環境である。他のコンポーネントは、テストがエンドユーザ (すなわち、試験技師および試験クラス開発者) に提供するプログラミングファシリティである。

【0106】

プログラミング環境の主なコンポーネントは試験計画である。試験計画は試験クラス (すなわち Test で表される試験インターフェースの種々のインプリメンテーションである) を使用し、試験クラスは、特定のタイプの試験のための試験データおよびコードの分離を実現する。

40

【0107】

その計画は C++ 試験プログラムとして直に書くことができるか、または試験計画記述ファイル内に記述することができ、そのファイルは試験プログラム発生プログラム (翻訳プログラム 402) によって処理され、C++ コードのようなオブジェクト指向コードが生成される。その後、生成された C++ コードをコンパイルして、実行可能試験プログラムを作成することができる。レベル、タイミングなどの試験クラスインスタンスを構成するために必要とされるデータは、試験計画記述ファイルにおいてユーザによって指定される。

【0108】

50

試験プログラムは、1つのデバイス上で試験を実行するための詳細を指定する、1組のユーザ書込みファイルを含む。本発明の一実施形態は、ユーザがC++構成体を用いてこれらのファイルを書くことができるようにする複数組の規則を含む。

【0109】

本発明の実施形態による要件のうちの1つは、オープンアーキテクチャ試験システムのモジュール性を追求することである。モジュール式の開発によって、ユーザが試験の種々の側面に関係する個々のコンポーネントを書くことができるようになり、それにより、これらのコンポーネントを種々の態様で様々に組み合わせて、1つの完全な試験プログラムを生成できるようになる。本発明の好ましい実施形態による試験プログラムは以下のような1組のファイルを含む。

ユーザ変数および定数のためのファイル*.usrv

仕様セットのためのファイル*.spec

レベルのためのファイル*.lvl

タイミングのためのファイル*.tim

試験条件グループのためのファイル*.tcg

ピン定義のためのファイル*.ddefs

カスタム関数および試験クラス用のファイルプリヘッダのためのファイル*.ph

カスタムタイプのためのファイル*.ctyp

カスタム変数のためのファイル*.cvar

試験計画のためのファイル*.tpl

【0110】

上記のファイル拡張子は、ファイルのカテゴリ化を容易にする、推奨される決まりである。単一の試験プログラムは、単一の試験計画ファイルと、それがインポートするファイルとを含むことが好ましいであろう。「インポート」は、インポータ（インポートを指定するファイル）によって直に参照されるか、またはインポータによって直に参照されるいくつかの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を指している。試験計画ファイルは、グローバル、フロー、およびそのファイル内の他のそのようなオブジェクトを定義することができるか、またはそのファイルは、他のファイルからこの情報をインポートすることができる。これらの規則によって、上記のコンポーネントのうちの任意のものが、その自らの個々のファイル内に存在できるようになるか、または試験計画ファイル内に直に並べられるようになる。試験計画は、C言語main()関数に概念的に類似であることに留意されたい。

【0111】

[試験プログラム主要項目]

- ・ユーザ変数および定数
- ・仕様セット
- ・レベル
- ・タイミング
- ・試験条件
- ・ピン定義
- ・プリヘッダ
- ・カスタムタイプ
- ・カスタム変数
- ・試験計画

【0112】

試験プログラム識別子は、アルファベットの大文字または小文字で始まり、それに続いて、任意の数のアルファベット文字、数字またはアンダーバー（_）を有することができる。それは、以下に与えられる説明において提供されるいくつかのキーワードを有する。これらのキーワードは、Versionのように、太字を用いて、本明細書のコード内で視覚的に特定される。キーワードは予約され、識別子として用いられない

ことが好ましい。{, }、(,)などのいくつかの特殊な記号、および他の記号があり、それらの記号が以下に説明される。

【0113】

〔試験オブジェクトのエラボレーション〕

試験記述ファイルをインポートすることにより、インポート用ファイルが、インポートされるファイルによって利用可能であるオブジェクトの名前を参照できるようになる。これにより、インポート用ファイルは、インポートされるファイルによって命名されるオブジェクトを参照できるようになる。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について考える。同じくxxx.pinをインポートする別のbbb.socファイルも存在することができる。しかしながら、これらのインポートのいずれによっても、xxx.pinによって記述されるオブジェクトは発生しない。それらは、既に存在するものと仮定されるオブジェクトを参照するだけである。

10

【0114】

そのようなオブジェクトがいつ発生するかという疑問が生じる。これは、試験計画ファイルが根本的に異なる場合である。Cとの類似性から、その中にmain()ルーチンを有するファイルが存在するであろう。試験計画ファイル内の「Import」ステートメントは、これらのオブジェクトをエラボレートするであろう。すなわち、「Import」ステートメントによって、これらのオブジェクトが発生する。以下に示される試験計画mickey.tplによって、xxx.pinおよびaaa.soc内のオブジェクトがエラボレートされる。

20

【0115】

```
# File for Mickey's Test Plan
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 Flow 1
{
...
}
```

30

【0116】

試験計画にxxx.pinをインポートすることにより、xxx.pinにおいて宣言される全てのピンおよびピングループオブジェクトがエラボレートされる。これは、「ファイルxxx.pinがエラボレートされる」のように記述される。試験計画が、エラボレートされる必要がある全てのファイルを直にインポートする必要はない。以下の2つのステートメントのうちのいずれかが正しい場合には、ファイルxがファイルyによってインポートされる。

40

1. yがxを命名するインポートステートメントを有する。または、
2. xがzによってインポートされ、yがzを命名するインポートステートメントを有する。

【0117】

試験プログラムがコンパイルされるとき、それは、試験計画によってインポートされるファイル内の全てのオブジェクトをエラボレートするであろう。1つの試験計画によってインポートされる1組のファイルは、それらのファイルがエラボレートされる順序をもたらすようにトポロジカルソートされる。1つの試験計画によってインポートされる1組のファイルは、その試験計画のインポート閉包と呼ばれる。1つの試験計画のインポート閉

50

包がトポロジカルソートできない場合には、インポートサイクルが存在しなければならない。そのような状況は誤っており、コンパイラによって拒否されるであろう。

【0118】

[ユーザ変数および定数]

ユーザ変数および定数を用いて、グローバル変数および定数が定義されるであろう。これらの定数は、その値がコンパイル時に固定され、変更することができないオブジェクトである。たとえば、最大整数値は定数になるであろう。一方、変数に結び付けられる式は、APIを介して、実行時に変更することができる。

- ・整数
- ・符号なし整数
- ・ダブル
- ・ストリング
- ・ボルト単位の電圧 (V)
- ・ボルト／秒単位の電圧スルー (V P S)
- ・アンペア単位の電流 (A)
- ・ワット単位の電力 (W)
- ・秒単位の時間 (S)
- ・メートル単位の長さ (M)
- ・ヘルツ単位の周波数 (H z)
- ・オーム単位の抵抗 (O h m)
- ・ファラド単位の静電容量 (F)

10

20

【0119】

タイプ整数、符号なし整数、ダブルおよびストリングは基本タイプと呼ばれる。基本タイプは測定単位を持たない。基本タイプでない基本的なタイプはダブルであり、関連する測定単位およびスケールを有する。指数記号は、共通の工学指数記号である。

- ・p F (ピコファラド) の場合のような、10−12の場合のp (ピコ)
- ・n S (ナノ秒) の場合のような、10−9の場合のn (ナノ)
- ・u S (マイクロ秒) の場合のような、10−6の場合のu (マイクロ)
- ・m V (ミリアンペア) の場合のような、10−3の場合のm (ミリ)
- ・k O h m (キロオーム) の場合のような、10+3の場合のk (キロ)
- ・M H z (メガヘルツ) の場合のような、10+6の場合のM (メガ)
- ・G H z (ギガヘルツ) の場合のような、10+9の場合のG (ギガ)

30

【0120】

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を有するであろう。以下は、いくつかのグローバル定数を有するファイルの一例である。いくつかの変数を有するファイルの一例は後に与えられる。

【0121】

```
# -----
# 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.
```

40

```
    Const Integer MaxInteger = 2147483647;
```

50

```

Const Integer MinInteger = -2147483648;
# Smallest value such that 1.0 + Epsilon!= 1.0
Const Double Epsilon =2.2204460492503131 e-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;
}

```

【0122】

10

先に宣言された1組のUserVarsは「=」の左側に変数の定義を有するものと見なされる。結果として、1つの変数または定数の定義は一度だけ行われることが好ましく、初期化されるべきである。

【0123】

先に述べられたように、定数は、一旦定義されたなら、変更されないであろう。1つの定数に結び付けられる式は、予め定義された定数と、リテラル値とを含むことができる。一方、変数はAPIを介して変更することができる。変数に結び付けられる式は、予め定義された変数と、定数と、リテラル値とを含むことができる。

【0124】

各変数は実行時に保持される式オブジェクトに結び付けられる。これは、実行時に変数に関連する式を変更し、その後、全ての変数を再評価する能力を与える。式オブジェクトは、変数または定数定義の右辺のパーズされた形である。1つの実施形態では、実行時に定数を変更するためのファシリティは提供されない。それらの値は、コンパイル時に固定されることが好ましい。

20

【0125】

グローバルを有する任意の数のそのようなファイルが、1つの試験計画のインポート閉包内に存在することができる。上記のグローバルファイルは1組の数値限界であるが、ここでは、工学測定単位を用いる1組の工学グローバルおよびいくつかのランダムユーザ変数である。

【0126】

30

```

# -----
# File myvars.usrv
# -----
Version 0.1;
#
# This declares a UserVars collection of some engineering
# globals.
#
UserVars MyVars
{

```

40

```

# 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 TimeClkTick =1.0ns;    # 1 nanosecond
Const Resistance R10 = 10.0 kOhms;    # 10 kilo Ohms

```

Some variables are declared below.

50

```

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], A1
# in ABusVil [1], so on.
#
Voltage ABusVil[8] = {1.0, 1.2, Others = 1.5};
}

```

10

【0 1 2 7】

コンパイラは、単位およびタイプが一致することを確認することが好ましい。電圧×電圧は電力になるので、上記のPLowおよびPHighのための式はコンパイルされることに留意されたい。しかしながら、以下に記載されるようなステートメントは典型的にはコンパイルされないであろう。

【0 1 2 8】

```

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

```

20

【0 1 2 9】

コンパイラは、ある特定の自動タイプ変換を可能にするであろう。

【0 1 3 0】

```

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

```

30

【0 1 3 1】

ダブル、符号なし整数および整数への明示タイプ変換も許される。

【0 1 3 2】

```

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

```

【0 1 3 3】

中間の基本タイプに変換することにより、非関連タイプ間の変換も可能である。

40

【0 1 3 4】

```

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.

```

【0 1 3 5】

試験計画オブジェクトは、名前及びその関連する式、値並びにタイプを含むコレクションであるUserVars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

50

に、または命名ユーザ変数コレクションに入ることができる。上記の例のUserVars宣言は、指定された名前はなく、デフォルト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命名することができる。

【0136】

```
# 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;
}
```

10

20

30

【0137】

UserVarsコレクション内の名前分解は以下のように進められる。

名前が修飾されている、すなわち名前が点によって分離される2つの部分を含む場合には、変数は、その点の前にある部分によって名前を付けられる、名前付きユーザ変数コレクションからもたらされる。したがって、上記のMyVars.XはMyVarsコレクション内のXを指している。デフォルトユーザ変数コレクションを明示するために、名前「_UserVars」を用いることができる。

40

【0138】

その名前が修飾されず、かつそのコレクション内に同じ名前の定数または変数が存在する場合には、名前はその定数または変数に分解する。

【0139】

そうでない場合には、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または変数に分解する。

【0140】

UserVarsコレクション内の定義のブロックの評価は、最初の定義から最後の定義まで順番に生じるものと考えられる。これには、各変数が用いられる前に、その変数が定義される必要がある。

50

【0141】

さらに、UserVarsコレクションのためにいくつかの定義ブロックが存在することができ、各ブロックがいくつかの変数を定義している。これらの定義ブロックは全て、試験計画において宣言された順序で評価されるものと考えることができ、その後、各ブロックの変数も宣言された順序で検査される。

【0142】

最後に、いくつかのUserVarsコレクションが存在することができ、各UserVarsコレクションがいくつかの定義ブロックにわたって変数を定義する。再び、それらの変数は全て、宣言された順序で初期化されるものと考えることができる。こうして、上記の例では、評価順序は、MyVars.X、MyVars.Y、YoursVars.X、YoursVars.Y1、YoursVars.Y2、MyVars.Z 10
になるであろう。

【0143】

UserVarsコレクションが別のコレクションからの変数を用いるとき、UserVarsコレクションはその変数の生データだけを用いることが好ましい。コレクション間では従属性情報は保持されない。それゆえ、従属性に基づく再評価は、ただ1つのコレクションに限定することができる。

【0144】

各ユーザ変数コレクションはC++ UserVarsクラスのインスタンスを指している。C++ UserVarsクラスのデフォルトオブジェクトは「_UserVars」と名前を付けられる。名前付きでないUserVars宣言内の変数は、デフォルトユーザ変数コレクションからのものであり、20
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内の変数は、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。上記の例では、「MyVars」C++オブジェクトは、変数X、YおよびZを有することになるであろう。

【0145】

[ユーザ変数のためのC++]

ユーザ変数は、nameストリング、const/varブーリアン、列挙された値としてのtypeおよび式木としてのexpressionを有するn個組のコレクションとして実装される。1つの名前の式は、以下のコールによってセットすることができる。

【0146】

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

【0147】

タイプExpressionは、アサインメントの右辺に対応するテキストのパーズ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存在するであろう。たとえば、limits.usrv内の1組のユーザ変数の集合（pageを参照）は、以下に示される1組のコールによって実装される。 40

【0148】

```
_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") );

```

【0149】

以下は、myvars.usrvにおいて宣言される変数のために実行されることになるC++ステートメントである。

【0150】

```

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(I .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));

```

【0151】

上記のコードでは、Expressionクラスは式のパースされた形を表すコンストラクタを有することが好ましい。Expressionはいくつかのコンストラクタを有し、その中には、ストリングリテラルを受け取り、それをパースするコンストラクタ、およびストリングリテラルを受け取り、ストリングリテラルとしてだけ用いる別のコンストラクタが含まれる。これらは、読みやすくするためにこれまでは指定されていない付加的なパラメータによって区別される。

【0152】

デフォルトユーザ変数コレクション内のユーザ変数は、クラスUserVarsの_UserVarsオブジェクトによって管理されるであろう。名前付きユーザ変数コレクションXxx内のユーザ変数は、Xxxと名前を付けられたUserVarsオブジェクトによって管理されるであろう。

【0153】

[UserVarsのためのランタイムAPI]

これらの名前および式を含むC++ UserVarsクラスは、アプリケーションプログラムインターフェース (API) をエクスポートし、実行時にこれらの値を評価し、変更する。UserVarsに関連する式の変更は、UserVarsがいつ再評価されるか、およびその評価の影響が何であるかという問題にも対処する。

10

【0154】

最初に、1つの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の問題を考える。それが、式に変更を加えるときに直ちにトリガされる場合には、ユーザは、再評価をトリガする前に、一連の関連する変更を加える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。

【0155】

次に、再評価の影響を考えることができる。好ましい実施形態によれば、3種類の再評価が利用可能である。

【0156】

UserVars Collection Re-evaluationは、ただ1つのUserVarsコレクションに限定される再評価である。この演算の意味は、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。

20

【0157】

UserVars Targeted Re-evaluationは、ただ1つの名前に結び付けられる式に対する変更限定される再評価である。これにより、ユーザは、ただ1つの名前の式を変更できるようになり、この特定の変更だけを考慮に入れて、コレクションの再評価が行われるであろう。

【0158】

UserVars Global Re-evaluation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、全てのUserVarsコレクションの再評価を、宣言された順序でトリガするので、非常にコストがかかる。

30

【0159】

上記の再評価は全て、UserVarsを再評価した後に、Level、Timingなどの従属オブジェクトを再評価するであろう。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すダーティビットを有するであろう。UserVarsコレクションがプログラムによって変更されるときはいつでも、それは、全ての従属オブジェクトにおいてダーティビットもセットするであろう。これは、従属オブジェクトの再評価をトリガするであろう。

【0160】

要約すると、名前付きUserVarsコレクションは、再評価の影響の問題を封じ込めるのを助ける。再評価は通常、ただ1つのコレクションに限定される。UserVarsを用いる1つの単純な方法は、デフォルトUserVarsコレクションだけを用いることであろう。そのように、変更を加えることの波及効果は、全てのUserVarsに起こり得る。この波及効果は、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することができる。

40

【0161】

多数のコレクションが互いからの変数を参照することができるが、それらの変数に結び付けられる値は使用時に固定される。UserVarsコレクション間の従属性は保持されない。

【0162】

基本タイプXxx（符号なし整数、電流、電圧など）毎に、値をゲットするためのメソッドは、以下の通りである。

【0163】

50

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

【0164】

値を直にセットするためのメソッドは存在しないので、それは、式をセットするためのコールと、その後のreevaluateCollection()へのコールとを通して行われることに留意されたい。

【0165】

式をゲットし、セットするためのメソッド。setExpression()コールを用いて、これまでに定義されていなかった新たな変数を定義することもできる。

【0166】

```
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);
```

【0167】

setExpression()コールは、その式の結果として循環的な従属性が生じる場合には、失敗する可能性がある。たとえば、以下の2つのコールが行われる場合には、第2のコールは、循環的な従属性による失敗によって失敗するであろう。

【0168】

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

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

【0169】

これは、名前に結び付けられる値が式であり、アサインメントでないためである。変数の値が変更されるとき、直接のおよび間接的に従属する名前を全て再評価するためのメソッドが提供される。上記の一对の式のような式の結果として、循環的な従属性が生じ、それは許容されない。

【0170】

このAPIは典型的には要求されていない再評価をサポートしないことに留意されたい。setExpression()へのコールでは、変数、およびその変数に従属する全ての他の変数が自動的に再評価されない場合がある。全ての変数に結び付けられる値は、reevaluateCollection()（以下）へのコールが生じるまで、変更されないままである。

【0171】

特定の名前が定数であるか否かを判定するためのメソッド。

【0172】

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

【0173】

タイプをゲットするためのメソッド。

【0174】

```
enum ElementaryType
```

```
{
```

```
    UnsignedIntegerT, IntegerT, DoubleT, VoltageT, ...
```

```
};
```

```
Status getType(const String& name,
```

```
                ElementaryType& elementaryType) const;
```

【0175】

UserVars Collection Re-evaluationメソッド。

【0176】

```
Status reevaluateCollection();
```

【0177】

そのクラスは、全ての変数およびそれらの従属する変数に関連する式を保持するであろう。このメソッドがコールされるとき、全ての変数が再評価されるようになるであろう。

【0178】

UserVars Targeted Re-evaluationメソッド。

【0179】

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

10

【0180】

そのクラスは、全ての変数およびそれらの従属する変数に関連する式を保持するであろう。このメソッドがコールされるとき、名前付き変数、およびその従属変数の全てが再評価されるようになるであろう。

・UserVars Global Re-evaluationメソッド。

【0181】

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

【0182】

そのクラスは、全ての変数およびそれらの従属する変数に関連する式を保持するであろう。このメソッドがコールされるとき、全てのUserVarsコレクション上で、reevaluateCollection()が不特定の順序でコールされる。

20

【0183】

特定の名前が定義されるか否かを判定するためのメソッド。

【0184】

```
StatusgetIsDefined(const String& name, bool& isDefined) const;
```

【0185】

現在定義されている全てのユーザ変数を判定するためのメソッド。

【0186】

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

【0187】

30

現在定義されている変数を削除するためのメソッド。

【0188】

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

【0189】

この演算は、その名前が他の変数を含む式において用いられる場合には、失敗するであろう。

・所与の変数または定数に従属する変数および定数のリストをゲットするためのメソッド。

【0190】

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

40

【0191】

[仕様セット]

仕様セットは、セレクトラに基づいて値を取り込むことができる変数のコレクションを供給するために用いられる。たとえば、セレクトラMinnie、Mickey、GoofyおよびDaisyを用いる以下のSpecification Setについて考える。

【0192】

```
# -----
# File Aaa. spec
# -----
Version 1.0;
```

50

```

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; }
}

```

10

【0193】

セレクトGoofyを有する上記の仕様セットは以下のように連想するであろう。

【0194】

```

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

```

20

【0195】

Testが記述されるときに、仕様セットにおいてセレクトをセットする演算が以下に説明されるであろう。

【0196】

構文的には、仕様セットaは、セレクトのリスト（上記の例では、Minnie、Mickey、GoofyおよびDaisy）と、変数定義のリスト（上記の例では、xxx、yyy、zzzおよびwww）である。変数の定義は、セレクトのリストと同じ長さの式のリストを含むか、またはただ1つの式を含む。

30

【0197】

概念的には、仕様セットは式の行列と考えることができ、その列がセレクトであり、その行が変数であり、そのエントリが式である。特定のセレクト（列）が各変数（行）を特定の式（エントリ）に結び付ける。そのリストがただ1つの式を有する場合には、そのリストは、セレクトが存在するのと同じ数だけ式が繰り返される行を表す。

【0198】

仕様セットは2つの別個の状況において現れることができる。仕様セットは、.specファイルにおいて別個に宣言されることができ、その場合には、それらの仕様セットは先に示されたように現れる。これらは名前付き仕様セットである。別の状況では、ローカル仕様セットをTest Codition Group内で宣言することができる。そのような宣言では、仕様セットは名前を与えられないであろう。取り囲んでいる試験条件グループに対してだけ意味のあるローカルな仕様セットが存在するであろう。

40

【0199】

名前付きユーザ変数コレクションの後に、名前付き仕様セットをモデル化することができる。上記の仕様セットは、Aaaと名前を付けられるUserVarsコレクションとしてモデル化されることができ、それはxxx[Minnie]、xxx[Mickey]、xxx[Goofy]、xxx[Daisy]、yyy[Minnie]などのための式を有するであろう。ある試験の状況において特定のセレクト（たとえばMickey）が選択されるとき、xxx、yyyおよびzzzの値は変数名および仕様セット名から得られる。

50

【0200】

試験条件グループは、ローカル仕様セットか、名前付き仕様セットへの参照かのいずれかである、多くても1つの仕様セットを有することができる。ローカル仕様セットは、試験条件グループの状況においてのみ現われ、明示される名前を持たない。そのような仕様セットは、取り囲んでいる試験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コレクションが現われる時点で、1つの試験条件グループ内の1つの名前を分解するために、以下の規則が適用される。

1. その名前が修飾されている場合には、その名前は、名前付きユーザ変数コレクションにおいて分解されなければならない。
2. その名前が修飾されていない場合には、その名前は、その名前が試験条件グループにおいて宣言される場合にはローカル仕様セットにおいて、それが試験条件グループにおいて参照される場合には名前付き仕様セットにおいて分解される。
3. その名前が上記の規則によって分解されない場合には、その名前はデフォルトユーザ変数コレクションにおいて分解される。

10

【0201】

これらの規則を例示するために、Test Conditions Group（後に説明されることになる）を用いる以下の例について考える。

【0202】

```
Version 1.2.3;
```

20

```
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.
```

30

```
    # 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
```

40

```
    # 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
```

50

```

# 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.

```

10

【0203】

20

仕様セット内の名前の分解（上記の規則）では、そのセットのセレクトが、名前分解が要求される時点で使用可能にされることが要求される。これは、試験条件グループが、セレクトを指定することによって1つのTestにおいて参照されることになるという事実によって強要されるであろう。

【0204】

[仕様セットのためのC++]

上記の規則を用いるとき、仕様セットは、C++ SpecificationSetクラスによって実装されることができる。SpecificationSetクラスは基本的には、UserVarsクラスと同じAPIを有するが、セレクトのための余分なストリングパラメータがあることが異なる。したがって、このAPIは詳細には説明されない。

30

【0205】

全ての名前付き仕様セットは、その名前のC++オブジェクトに関連付けられることが好ましい。ある試験条件グループの状況におけるローカル仕様セットは、その試験条件グループに固有の名前を有するであろう。そのローカル仕様セットが定義される試験条件グループの状況以外では、ローカル仕様セットの変数を参照することは違法である。

【0206】

[レベル]

Levelは、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用いられる。それは、以下の形の宣言のコレクションである。

【0207】

40

```

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

```

【0208】

そのような宣言は、名前付きピンまたはピングループの種々のパラメータの設定を指定する。たとえば、以下の例に示されるように、そのようなステートメントを用いて、InputPinsグループ内の全てのピンのためのVIL値をセットすることができる。

50

【 0 2 0 9 】

```
# -----
# File CHIPlevels.lvl
# -----
```

```
Version 1.0;
```

```
Import CHIP3resources.rsc;
Import CHIP3pins.pin;
```

10

```
Levels CHIP3Levels
{
```

```
    #
    # 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.
```

20

```
    #
    # From the imported pin description fileCHIP3pins.pin,
    # the InPins group is in the "dpin" resource. From the
    # imported resource definition file CHIP3resources.rsc,
    # the "dps" resource has parameters named VIL and VIH.
```

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

30

```
    # 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.
```

40

```
    #
    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;
```

50

```

#
# 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 CHIP4Levels
{
    # ...
}

```

【0 2 1 0】

上記のように、各Levelブロックは、多数のレベル項目から構成されることが好ましく、各レベル項目は1つの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。各レベル項目は多数の資源パラメータを指定することができる。これらのレベル値の設定のための実行時の意味は以下の通りである。

【0 2 1 1】

Levelsブロックのレベル項目は宣言された順序で処理される。2つ以上のレベル項目において生じるピンがあれば、何度でも処理されることになる。ただ1つのパラメータのための値の多数の仕様は、指定された順序で保持され、適用されるべきである。

【0 2 1 2】

1つのレベル項目内の資源パラメータは、それらが指定された順序で処理される。

【0 2 1 3】

次のレベルグループをセットする前に、Delayステートメントによって、レベルをセットするプロセスが概ね指示された持続時間だけ中断する。実際の待ち時間は、指定された遅延を中心にして、小システムによって規定される範囲内にあることができる。したがって、遅延がt秒であった場合には、実際の遅延は以下の式を満たすであろう。

【0 2 1 4】

$$t - \Delta t \leq \text{actual-wait} \leq t + \Delta t$$

【0 2 1 5】

Delayステートメントは、Levels指定を多数のサブシーケンスに分割し、各サブシーケンスは、処理するための別個のTest Condition Memory設定を必要とするであろう。

【0 2 1 6】

次のレベルグループをセットする前に、MinDelayステートメントによって、レベルをセ

ットするプロセスが少なくとも指定された持続時間だけ中断する。実際の待ち時間は、小システムによって規定される範囲内にあり、指定された最小遅延の最小値を有することができる。したがって、最小遅延が t 秒であった場合には、実際の遅延は以下の式を満たすであろう。

【0217】

$$t \leq \text{actual-wait} \leq t + \Delta t$$

【0218】

MinDelayステートメントは、Levels指定を多数のサブシーケンスに分割し、各サブシーケンスは、処理するための別個のTest Condition Memory設定を必要とするであろう。

【0219】

各ピンまたはピングループ名はピン記述ファイル（接尾部.pin）内の厳密に1つの資源において指定され、それゆえ、資源ファイル（接尾部.rsc）において指定される、ある1組の実用的な資源パラメータを有する。名前を付けられる全てのパラメータはこの1組の実用的な資源パラメータの中からもたらされるべきであり、それらの値をセットするために用いられる式と同じ基本タイプを有するべきである。資源パラメータの名前およびタイプについての情報は資源ファイルからもたらされる。

【0220】

資源ファイルResource.rscは暗黙のうちにインポートされ、テストにdpinおよびdpsのような標準的な資源のパラメータのための名前およびタイプを与える。

【0221】

資源パラメータは、UserVarsを用いることができる式、および名前付きの仕様セットまたは現時点で見ることができるローカル仕様セットからの値を割り当てられる。

【0222】

dpsピン資源は、特殊なパラメータPRE_WAITおよびPOST_WAITを有する。PRE_WAITパラメータは、電源ピンがその目標電圧に達する時刻から、パターン生成を開始することができる時刻まで経過するのにかかる時間を指定する。POST_WAITパラメータは、パターン生成が停止した時刻から、電源ピンが遮断する時刻まで経過するのにかかる時間を指定する。

【0223】

またdpsピンは、電圧パラメータがその最終値に如何に達するかも指定する。それらのピンは、全ての他のピンパラメータのように、単に1つの式によって電圧パラメータを指定することができる。その場合、その値は、ハードウェアがそれを可能にするときに、達成されるであろう。またそれらのピンは、Slewステートメントを用いて、それを指定することもできる。Slewステートメントは、電源電圧が、指定された絶対Voltage Slew Rateを有する傾斜で、その初期値から最終値に達することを指定する。

【0224】

[LevelsのためのC++]

上記の規則を用いて、以下の演算をサポートするC++ Levelsオブジェクトを書くことができる。

・以下の演算が行われる。

【0225】

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

【0226】

この演算は、1つの式を、1つのピンまたはピングループのパラメータに結び付ける。たとえば、dpin.InPins VIH値は以下の式によってセットされる。

【0227】

```
setParameter ("InPins", "VIH", VoltageT,
```

10

20

30

40

50

```
Expression("v_ih + 1.0");
```

【0228】

この演算は、Levelsオブジェクト内の全ての宣言のために何度もコールされるであろう。

・以下の演算が行われ、その演算は、先に説明されたように、全ての所定のモジュールレベルインターフェースをくまなく調べて、発行し、パラメータの全てのレベルを、指定された順序で割り当ててであろう。セクタパラメータを用いて、先に指定された規則に従って、それらの式内の名前が分解される。

【0229】

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

10

【0230】

[試験条件グループ]

試験条件グループ部分言語は、仕様、タイミングおよびレベルの記述と一緒にパッケージ化する。Timingオブジェクトは多くの場合に、複数のパラメータを用いて指定される。複数のパラメータを複数のタイミングにおいて用いて、種々のパルスの前縁および後縁を指定することができる。同様に、Levelsは種々の電圧レベルの最大値、最小値および典型値を指定することによりパラメータ化することができる。試験条件グループ（TCG）オブジェクトは、仕様と、これらの仕様に基づくTimingsおよびLevelsのインスタネーションとを一纏めにする。

【0231】

20

TestConditionGroup宣言はオプションのSpecificationSetを含む。SpecificationSet宣言には、インライン化された（そして名前付きでない）ローカルSpecificationSetを用いることができるか、または他の場所で宣言された名前付きSpecificationSetへの参照を用いることができる。TCG宣言内のオプションのSpecificationSet宣言には、少なくとも1つのLevelsまたはTimings宣言が続く。それは、複数のLevelおよび複数のTimingの両方を任意の順序で有することができる。しかしながら、それは、2つ以上のLevelsおよびTimings宣言を有することは許されない。これらの制約は構文的に強制される。

【0232】

TCG内の仕様セット宣言は、それが名前を持たないことを除いて、別個に宣言される仕様セットと同じである。その名前は暗黙のうちに、取り囲んでいるTCGの名前になる。Timings宣言は、指定されたタイミングファイルからのTimingsオブジェクトのただ1つの宣言を含む。ここに示されるのは、試験条件グループを有するファイルの一例である。

30

【0233】

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

```
Version 0.1;
```

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

40

```
TestConditionGroup TCG1
{
```

```
    # This Local SpecificationSet uses user-defined selectors
    # "min", "max" and "typ". Any number of selectors with any
    # user defined names is allowed.
    #
```

50

```

# 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 CHIP3Levels imported earlier. It
# is one of possibly many levels objects that have been
# imported from the above file.
Levels CHIP3Levels;

# Refers to filetiming1.tim containing the single
# timing Timing1. The filename should be quoted if
# it has whitespace characters in it.

```

```

    Timings Timing 1 ;
}

# 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:
    # Specification Set 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;
    #   Timedata_le = 10.0 uS, 10.2 uS, 10.1 uS;
    #   Time data_te = 30.0 uS, 30.2 uS, 30.1 uS;
    # }
    # A Specification Set 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;
}

```

【0 2 3 4】

上記の例では、試験条件グループTCG1は、「min」、「typ」および「max」の名前を付けられた3つのセレクトを有する仕様セットを記述する。任意の数の別個のセレクトが存在することができる。仕様セットの本体内で、それらのセレクトに対応する3個1組の値で、変数v_il、v_ih、t_leおよびt_teが初期化される。したがって、上記の例では、セレクト「min」を有するTCG1のインスタンスが、変数v_ilを第1の数値と結合するであろう(vInputLow+0.0)。再び、1つの仕様セットのための複数のセレクトがユーザーによって定義され、任意の数のセレクトが許される。唯一の要件は以下の通りである。

【0 2 3 5】

1つの仕様セットのセレクトは固有の識別子である。

【0 2 3 6】

その仕様セットにおいて指定される各値は、その1組のセレクトと厳密に同じ数の要素を有する値の配列に関連付けられる。i番目のセレクトを選ぶことにより、各値が、それらの値の関連するベクトルのi番目の値に結び付けられるであろう。

【0 2 3 7】

TCG内の仕様セットに続いて、Levels宣言またはTimings宣言、あるいはその両方が存在することができる。Levels宣言を用いて、種々のピンパラメータのためのレベルがセ

ットされる。仕様セットにおいて特定される変数を用いて、これらのレベルがセットされることになり、それにより、TCGを初期化するために用いられるセクタに基づいて、ピンパラメータのための種々の実際の値を動的に結合できるようになる。

【0238】

これを例示するために、セクタ「min」を使用可能にするTestを考える。ページ上に与えられる仕様セットCHIP3Levelsを参照すると、InPinsグループ内のピンのためのピンパラメータ「VIH」は、以下の宣言によって式(v_{ih}+1.0)に初期化されるであろう。

【0239】

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

【0240】

セクタ「min」が使用可能にされるとき、これは(VInHigh+0.0+1.0)に分解する。同様に、Timingsオブジェクトは、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。TimingsおよびLevels宣言の両方を有する必要はない。以下の例によって例示されるように、いずれかがそれだけで存在することができるか、または両方が任意の順序で存在することができる。

【0241】

```
# -----
# 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;
```

```
        Voltagev_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;
```

```
}
```

10

20

30

40

50

【0242】

しかしながら、1つのTCG内に2つ以上のTimingsおよび2つ以上のLevelsが存在すべきでないことに留意されたい。したがって、要約すると、TimingsまたはLevelsの少なくとも一方が存在すべきであり、かつ多くてもそれぞれの1つが存在すべきである。

【0243】

[試験条件]

TestConditionオブジェクトは1つのTCGを特定のセクタに結び付ける。一旦、TCGが先に示されるように宣言されたなら、以下に示されるように、TestConditionオブジェクトを宣言することができる。

【0244】

```
TestCondition TCMin
{
    TestConditionGroup = TCG1;
    Selector = min;
}
TestCondition TCTyp .
{
    TestConditionGroup = TCG1;
    Selector = typ;
}
TestCondition TCMax
{
    TestConditionGroup = TCG1;
    Selector = max;
}
```

10

20

【0245】

これらのTest Conditionは、以下のように、1つのTest Planにおいてインスタンス化されるであろう。

【0246】

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

30

40

【0247】

[TCG (試験条件グループ) 内の名前分解]

試験条件グループ内の名前の分解は先に説明された。しかしながら、これらの規則が繰返し述べられ、再び以下に与えられる。

1. その名前が修飾されている場合には（ページを参照されたい）、その名前は、名前付きユーザ変数コレクションにおいて分解されなければならない。
2. その名前が修飾されていない場合には、その名前は、その名前が試験条件グループにおいて宣言される場合にはローカル仕様セットにおいて、それが試験条件グループにおい

50

て参照される場合には名前付き仕様セットにおいて分解される。

3. その名前が上記の規則によって分解されない場合には、その名前はデフォルトユーザ変数コレクションにおいて分解される。

【0248】

[TCGランタイム]

試験条件グループは以下の実行時の意味を有する。

【0249】

Test (FunctionalTestなど) は、インスタンス化されたTestConditionを用いて、そのSpecificationSetからの特定のセレクタを有するTCGを参照するであろう。このセレクタは、SpecificationSet内の各変数を、選択されたセレクタに関連付けられる値に結び付けるであろう。その後、変数とそれらの値とのこの結び付きを用いて、LevelsおよびTimingsが判定されるであろう。

10

【0250】

TestConditionGroup内のパラメータLevelsは、Levelsブロックの提示順に、順番にセットされることが好ましい。したがって、CHIP3Levelsブロックでは、パラメータレベルがセットされることになる順序は以下の通りである（表記法：<resource-name>.<resource-parameter>）。

- ・InputPins.VIL
- ・InputPins.VIH
- ・OutputPins.VIL
- ・OutputPins.VIH
- ・Clock.VOL
- ・Clock.VOH

20

【0251】

このようなシーケンシングの順序によって、試験ライタは、電源の明示的なパワーシーケンシングを制御できるようになる。さらに、1つのレベル項目が2回生じ、1つのピンのために同じピンパラメータの名前を付ける場合には、そのピンパラメータは2回セットされるようになる。これは、プログラムによって生じることでもある。

【0252】

1つのパラメータが以下のようなSlewステートメントによってセットされる場合には、それは、VCCが、毎秒±0.01ボルトの電圧スルーレートの傾斜で、その現在の値から2.0ボルトの最終値に達することを意味する。

30

【0253】

VCC = Slew(0.01,2.0 V);

【0254】

仕様セット変数はTCG内のTimingsオブジェクトにも渡すことができる。その後、Timings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であろう。そのような仕組みを用いて、たとえば、波形の前縁および後縁を指定することにより、Timingsオブジェクトをカスタマイズすることができる。

【0255】

40

[TCGのためのC++]

上記の規則を用いて、Test Condition GroupがC++ TestConditionGroupクラスにおいて宣言されることができ、それを以下のように初期化することができる。

【0256】

以下のTestConditionGroupメンバ関数へのコールが行われる。

【0257】

Status setSpecificationSet(SpecificationSet *pSpecificationSet);

【0258】

その関数はTestConditionGroupのための仕様セットをセットするであろう。これは、ローカル仕様セット、または名前付き仕様セット、またはヌル（存在しない場合）のいずれ

50

かにすることができる。

【0259】

以下のTestConditionGroupメンバ関数へのコールが行われる。

【0260】

```
Status setLevels(Levels *pLevels);
```

【0261】

その関数はTestConditionGroupのためのLevelsオブジェクトをセットするであろう。これは、ローカルに宣言されるレベルオブジェクト、または外部から宣言されるレベルオブジェクト、またはヌル（存在しない場合）のいずれかにすることができる。

【0262】

以下のTestConditionGroupメンバ関数へのコールが行われる。

【0263】

```
Status setTimings(Timings *pTimings);
```

【0264】

その関数はTestConditionGroupのためのTimingsオブジェクトをセットするであろう。これは、外部から宣言されるTimingsオブジェクト、またはヌル（存在しない場合）のいずれかにすることができる。

【0265】

[ビン定義]

Bin Definitionクラスはビン、すなわち、数多くのDUTを試験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。1つのDUTを試験する過程において、そのDUTは、たとえば特定の試験の結果を指示するために、任意のビンにセットされることができる。試験が進められるとき、そのDUTは別のビンにセットされることができる。DUTが最終的にセットされるビンは、その試験の終了時の最後のそのような設定である。この最終的なビンのためのカウンタは、このDUTの試験の終了時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部.bdefsを有するべきある。

【0266】

ビン定義は階層的であることが好ましい。たとえば、最も外側にあるレベルでは、PassおよびFailと名前を付けられる2つのビンに有するPassFailBinが存在することができる。その際、いくつかのHardBinが存在することができ、そのうちのいくつかはPassビンにマッピングし、他のものはFailビンにマッピングする。HardBinは、PassFailBinのリファインメントと言われる。最後に、多数のSoftBin、すなわちHardBinのリファインメントが存在することができ、それらの多くが同じHardBinにマッピングする。以下はビンの階層を示す一例である。

【0267】

```
# -----
# File CHIPbins.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";
```

10

20

30

40

50

```

    "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;
}
}

```

【0268】

上記の例では、複数の最も基本的なベースピンはBinGroup HardBinsである。いくつかの他のBinGroupがXのリファインメントである場合には、BinGroupXは、ベースピンのグループであると言われる。したがって、BinGroup SoftBinsは複数のHardBinsのリファインメントであるので、BinGroup HardBinsはベースピンのグループである。複数のSoftBinsのピンはリーフピンと呼ばれる。他のBinGroupがYのリファインメントでない場合には、BinGroupYは、リーフピンのグループであると言われる。

【0269】

その中にただ1つのBinGroupZを有するBinDefsブロックの悪化した事例は、最も基本的な複数のベースピンのグループ、ならびにリーフピンのグループであるようなZを有するであろう。BinGroup名は、その範囲がグローバルである。任意の数のBinDefsブロックが存在することができるが、宣言された複数のBinGroupsは識別できなければならない。1つのBinDefsブロックからのBinGroupは、別のBinDefsブロックからのBinGroupのリファインメントであることを許される。したがって、上記の例では、複数のSoftBinsは、複数のHardBinsからの別個のBinDefsブロック内に存在することができる。しかしながら、読みやすくするために、全てのBinGroupsが定義されているただ1つのBinDefsブロックを有することが強く推奨される。

【0270】

ここで、合格したDUTの数および不合格のDUTの数をカウントするために、別のBinGroupを追加することにより、上記の階層を拡張することができる。

【0271】

```

# -----
File CHIPbins.bdefs
# -----

```

10

20

30

40

50

```

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
    {
        "3 GHzAllPass" :
            "Good DUTs at 3GHz", "3GHzPass";
        "3 GHzCacheFail":
            "Cache Fails at 3GHz", "3GHzFail";
        "3 GHzSBFTFail":
            "SBFT Fails at 3GHz", "3GHzFail";
        "3GHzLeakage":
            "Leakages at 3GHz", LeakageFail;
        "2.8GHzAllPass":
            "Good DUTs at 2.8GHz", "2.8GHzPass";
        "2.8GHzCacheFail":
            "Cache Fails at2.8GHz", "2.8GHzFail";
        "2.8GHzSBFTFail":
            "SBFT Fails at 2.8GHz", "2.8GHzFail";
        "2.8GHzLeakage":
            "Leakages at 2.8GHz", LeakageFail;
    }
}

```

【 0 2 7 2 】

今度は、複数の最も基本的なベースピンがBinGroup PassFailBinsである。それらは典型的には任意のピンのリファインメントではない。BinGroup HardBinsは複数のPassFailBinsのリファインメントであり、ベースピンでもある。複数のSoftBinsは複数のHardBinsのリファインメントであり、リーフピンのグループである。上記の例は、その階層内に 3

つのBinGroupしか持たない。以下は、さらに複雑な階層である。

【0273】

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 { ... }
}
```

10

20

【0274】

この例では、AxおよびAyはAのリファインメントであり、Axxは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。この例は、BinGroupBおよびBxも提供し、BxはBのリファインメントである。PassFailBins、HardBinsおよびSoftBinsと名前を付けられたBinGroupを有する上記のBinDefs宣言は、このセクションにおいて、継続的な例として用いられるであろう。

【0275】

1つのBinGroup内の各ピンは、

1. 識別子またはストリングリテラルのいずれかである名前と、
2. このピンが何を要約するかを記述する記述と、
3. このピンがリファインメントBinGroup内にある場合には、それがリファインメントであり、ベースピンとしても知られているピンの名前とを有する。

30

【0276】

複数のPassFailBin内の2つのピンは「Pass」および「Fail」と名前を付けられる。複数のHardBin内の5つのピンは、「3GHzPass」、「2.8GHzPass」、「3GHzFail」、「2.8GHzFail」、「LeakageFail」と名前を付けられる。ピン名は、リテラルストリング、または識別子にすることができる。ピン名は1つのBinGroup内で固有でなければならないが、複数のBinGroupにわたって繰り返されることができる。しかしながら、BinGroup名は、範囲がグローバルであり、1つの試験計画にわたって固有でなければならない。

【0277】

5つのHardBinのうち、ピン「3GHzPass」および「2.8GHzPass」はいずれも、PassFailBinの「Pass」ピンにマッピングする。複数のHardBinのうちの残りはPassFailBinのうちの「Fail」ピンにマッピングする。

40

【0278】

最後に、8つのSoftBinが存在する。SBFT（ソフトピン機能試験）およびCacheの場合の3GHzにおける2つの不合格は「3GHzFail」HardBinにマッピングする。同様に、SBFTおよびCacheの場合の2.8GHzにおける2つの不合格は「2.8GHzFail」HardBinにマッピングする。Leakageに起因する不合格はいずれも、それらが生じた速度にかかわらず、同じ「LeakageFail」HardBinにマッピングする。たとえば、最も粗い（最も外側のレベルでの）試験は、1つのDUTが試験に合格するか、不合格であるかである。たとえば、リフ

50

ァインメントは、その D U T が特定の周波数、たとえば 3 G H z など、試験に合格するか、不合格であるかである。

【0279】

ピンは以下に述べるように Test Plan FlowItem 内の D U T に割り当てられる。Test Plan FlowItem は Result Clause を有し、その中で、試験計画は、1 つの試験を実施することから特定の結果が戻される結果として行われるべき動作および遷移を記述する。この時点で、SetBin ステートメントが生じることができる。

【0280】

```
# 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";
```

```
}
```

【0281】

多くの SetBin ステートメントは、1 つの D U T における試験を実行している過程において実行することができる。その試験が最終的に完了するとき、ランタイムは、その D U T のためにセットされた最終的なピン、および全てのそのリファインメントのためのカウンタをインクリメントするであろう。その試験の過程において実行される以下の SetBin ステートメントを有する D U T について考える。

【0282】

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

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

【0283】

この D U T は 3GHz Cache および Leakage 試験に合格したが、S B F T 試験には不合格であったので、「3GHzSBFTFail」ピンに割り当てられる。その後、2.8GHz において試験され、全ての試験に合格した。したがって、最終的なピン割当ては、「2.8GHzAllPass」ピンに対してなされ、それは、1 組の SoftBin の中にある。この最終的な割当ては、以下のピンのカウンタをインクリメントするであろう。

1. 複数の SoftBin 「2.8GHzAllPass」
2. 複数の HardBin のリファインメント 「2.8GHzPass」
3. 複数の PassFailBin のリファインメント 「Pass」

【0284】

その試験が完了するとき、ランタイムは、D U T の最終的なピン割当て、およびそれがリファインメントである全ての他のピンのためのカウンタをインクリメントするであろう。

【0285】

SetBin ステートメントはリーフピンにおいてのみ許される。ベースピンをセットすることは違法である。上記のカウンタインクリメントの意味は、以下のことを仮定する。

1. ピンがリーフピンである場合には、それは、1 つの D U T の試験の終了時にこのピンのために SetBin ステートメントが実行された回数である。
2. ピンがベースピンである場合には、それは、それがリファインメントであるピンのカウンタの和である。

【0286】

こうして、上記の例では、1 つの SetBin ステートメントにおいて、SoftBin だけが許される。複数の HardBin 「LeakageFail」 のためのカウンタは、複数の SoftBin 「3GHzLeakage

10

20

30

40

50

Fail」および複数のSoftBin「2.8GHzLeakageFail」のためのカウンタの和である。以下はビン定義に関するいくつかの規則である。

1. BinDefinitions宣言はいくつかのBinGroup宣言から構成される。
2. 各BinGroup宣言は、名前と、それがリファインメントであるオプションのBinGroup名と、それに続くビン宣言のブロックとを有する。
3. ビン宣言は、名前と、それに続く記述と、オプションでそれに続く、このビンがリファインメントであるベースビンの名前とを含む。
4. ビン名はストリングリテラルまたはIdにすることができる。空ストリングは有効なビン名にすべきではない。ビン名は、BinGroup宣言内の名前の中で固有であるべきであるが、他のBinGroup宣言において同じ名前を用いることができる。
5. BinGroup宣言Xxxが別のBinGroup宣言Yyyのリファインメントである場合には、Xxx内の全てのビン宣言は、Yyyからのベースビンの名前を宣言しなければならない。したがって、複数のSoftBinは複数のHardBinのリファインメントであることを宣言されるので、複数のSoftBin内の各ビン宣言は複数のHardBinのビンのリファインメントである。
6. PassFailBinのような、別のBinGroup宣言のリファインメントでないBinGroup宣言は、ベースビン宣言しないBin宣言を有することが好ましい。

10

【0287】

ビンBbbは、Bbbがそのリファインメントである完全な1組のビンである1組のベースを有する。正式には以下のように定義される。

1. AaaがBbbのベースビンである場合には、AaaはBbbの1組のベース内にある。
2. Aaaの任意のベースもBbbの1組のベース内にある。

20

【0288】

BinGroup名は1つのTestPlanにおいてグローバルである。

【0289】

ビン名は1つのBinGroupに対してローカルである。

【0290】

SetBinステートメントはリーフビンの場合にのみ許される。

【0291】

[ビン定義のためのC++]

上記の規則を用いて、1つのオブジェクトタイプBinGroupを、BinDefs宣言内のBinGroup宣言毎に構成することができる。クラスBinGroupはサブクラスLeafBinGroupを有するであろう。これらの2つのクラスの演算は同じであるが、BinGroup::incrementBinがC++保護演算であるのに対して、LeafBinGroup::incrementBinがC++公開演算であることを除く。

30

【0292】

以下は、他のどのBinGroupのリファインメントでもないBinGroupまたはLeafBinGroupを構築するデフォルトコンストラクタである。

【0293】

以下のコンストラクタは、所与のベースBinGroupのリファインメントであるBinGroupを構築する。

40

【0294】

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

【0295】

以下のメソッドは1つのビンおよびその記述を定義する。

【0296】

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

【0297】

50

それが最も基本的なベースビンである場合には、ベースBinNameパラメータは空ストリングでなければならない。

【0298】

ビンカウンタをインクリメントするためのメソッド。

【0299】

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

【0300】

この演算は、このビン、およびこのビンのベースである全てのビンのためのカウンタをインクリメントするであろう。その演算はクラスBinGroupにおいて保護され、クラスLeafBinGroupにおいて公開される。

10

【0301】

ビンカウンタをリセットするためのメソッド。

【0302】

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

【0303】

この演算は、このビン、およびこのビンのベースである全てのビンのためのカウンタをリセットするであろう。

【0304】

1つのビンについての情報をゲットするためのメソッド。

【0305】

20

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

【0306】

現在定義されている全てのビン名を知るために繰り返し子が与えられるであろう。

【0307】

30

TestPlan状態は、BinGroup宣言毎に1つの、多数のBinGroupメンバを含むであろう。上記のBinDefinitionのためのC++は以下になるであろう。

【0308】

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

40

```
// 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");
...
```

【0309】

1つのTestPlanのための状態は、未定義のBinGroup(NULL)に初期化されるm_pCurrentBinGroupと未定義のビン名（空ストリング）に初期化されるm_currentBinとを含む。SetBinステートメントが実行される度に、1つのコールによって、m_pCurrentBinGroupが指示された名前付きBinGroupに変更され、m_currentBinがグループ内の名前付きビンに変更され

50

る。

【0310】

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

【0311】

試験計画が実行を完了するとき、その試験計画は以下をコールして、このビンおよび全てのそのベースビンのカウンタがインクリメントされるようにする。

```
m_pCurrentBinGroup->incrementBin(m_currentBin);
```

【0312】

BinGroupカウンタは、試験計画がエラボレートされるときにリセットされるが、試験が実行される度に再初期化されない。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。

【0313】

[C. 試験計画]

試験計画は、試験プログラムの主要構造と考えることができる。試験計画はファイルをインポートし、類似のコンストラクツインラインを定義することができる。こうして、いくつかのグローバルの定義を与えられたファイルをインポートし、付加的なグローバルズインラインを宣言することができる。

【0314】

[C1. Test Plan FlowおよびFlowItem]

試験計画の重要な要素のうちの1つはFlowである。Flowは、有限状態機械をカプセル化する。それはいくつかのFlowItemを含み、それらのFlowItemはIFlowableオブジェクトを実行し、その後、別のフロー項目に遷移する。IFlowableを実行することは、IFlowableインターフェースをインプリメントするオブジェクトを実行することを含む。IFlowableインターフェースをインプリメントする一般的なオブジェクトはTestおよびFlowそのものである。

【0315】

こうして、1つのFlowは、Testおよび他のFlowを実行し、その後、別のFlowItemへの遷移を実行するFlowItemを有する。それは、1つのIFlowableを実行することに起因する種々のリターン結果において、ユーザがカスタマイズしたルーチンをコールするための機会も提供する。典型的には、こうして、Flowは以下の形式を有する。

【0316】

```
#
# FlowTest1 implements a finite state machine for the
# Min, Typ and Max flavors of MyFunctionalTest1. On
# success it testsTest1Min, 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 testsMyFunctionalTest1Min, ... 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
```

10

20

30

40

50

```

{
  Result 0
  {
    PropertyPassFail = "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 { ... }
}

```

【0317】

フローFlowTest1の演算は以下の通りである。

1. FlowItem FlowTest1_Minを実行することで開始する。
2. FlowTest1_Minは機能試験、MyFunctionalTest1Minを実行する。この試験の詳細は、後に完全な試験計画が提示されるときに与えられる。
3. この試験を実行することから、9つの結果、0、1、2、5、6、7、-6、-5または-4が予想される。最初の2つのResult句はそれぞれ0および1を処理し、第3の句は結果値の残り全てを処理する。
4. 結果「0」（合格）が生じる場合には、FlowTest1_MinはカウンタPassCounterをインクリメントするであろう。その後、それは新たなFlowItem FlowTest1_Typに遷移するであろう。
5. 結果「1」または結果「2」が生じる場合には、FlowTest1_MinはカウンタFailCounterをインクリメントし、そのフローから戻るであろう。
6. FlowTest1_Typは同じようにして動作し、成功時にFlowTest1_Maxをコールするであろう。
7. FlowTest1_Maxは同じようにして動作し、成功時に、成功した結果（「0」）でFlowTest1から戻るであろう。

【0318】

こうして、FlowTest1は、実行の成功時に、Test1の最小、典型および最大バージョンを通してデバイスを動かし、その後、戻るであろう。FlowTest2も同じようにして動作するであろう。

【0319】

上記のようなFlowは基本的には、状態および遷移を有する有限状態機械を記述する。FlowItemは基本的には状態であり、それは以下のことを果たすであろう。

10

20

30

40

50

1. 1つのIFlowableを実行する（それは、以前に定義されたFlow、またはTest、または上記の規則によってC++においてインプリメントされることができるユーザ定義のFlowを用いることができる）。

2. IFlowableの実行は数値結果を返す。その結果に基づいて、ある特定の動作が行われ（いくつかのカウンタを更新する）、その後、2つの事柄のうちの1つが起こる。

a. Flowは数値結果とともにコール元に戻る。

b. Flowは、別の状態(FlowItem)に遷移することにより継続する。

【0320】

こうして、1つのFlowItemは以下のコンポーネントを有する。

- ・ 1つのFlowItemは1つの名前を有する。 10
- ・ 1つのFlowItemは実行されるべき1つのFlowableを有する。
- ・ 1つのFlowItemは数またはResult句を有する。
- ・ 1つのFlowItemの各Result句は動作を提供し、1つの遷移で終了し、1つまたは複数の結果値に関連付けられる。

【0321】

これらの項目はFlowItem内で構文的に以下のようになる。

【0322】

```
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、Test、またはユーザ定義IFlowable、またはFlowのいずれかにすることができる。1つの結果のための動作は、以下に記載される動作のうちの任意のものにすることができる。

・ GUIツールによって用いられるストリング値を有する実体を属性結果にセットするためのプロパティ動作。これは上記のFlowTest1の例において見ることができる。

【0324】

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

【0325】

プロパティは基本的には、1つの結果句に関連付けられる名前付きストリングまたは整数値を有する実体である。任意の数のプロパティが存在することができ、プロパティは、この結果に関連する情報を表示するためにユーザが用いることになる、GUIのようなツールによって用いられることが好ましい。プロパティは、試験の実際の結果、すなわち試験のフローに影響を及ぼさない。

【0326】

いくつかの数のカウンタをインクリメントするためのカウンタ動作。これは上記の例において見ることができる。

【0327】

```
IncrementCounters PassCount;
```

- ・ 任意の、またはユーザルーチンをコールするためのルーチンコール動作。これは後に 50

説明される。

【0328】

最後に、FlowItemは、別のFlowItemに制御を渡すためのGoToステートメント、またはコール元（コール処理フロー、または試験計画を開始したシステムルーチンのいずれか）に制御を戻すためのReturnステートメントのいずれかにすることができる遷移を有する。

【0329】

[予め定義されたFlow]

Flowオブジェクトの典型的な使用法は、Testのシーケンスを定義することである。その後、このシーケンスは、試験計画サーバ（TPS）において生じるイベント、すなわちExecute Test Planイベントの結果として実行される。各サイトコントローラ上にある試験計画サーバが、ユーザの試験計画を実行する。しかしながら、Flowオブジェクトは、他のイベントに応答しても実行される。括弧内の名前は、Flow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。

1. System Load Flow(SysLoadFlow)。このFlowは、1つの試験計画が1つまたは複数のサイトコントローラ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。それは、任意のサイトコントローラ上に試験計画を実際にロードする前に実行される。このフローによって、試験計画開発者は、システムコントローラを起源とすべきである動作を定義できるようになる。そのような動作は、パターンファイルのブロードキャストロード、較正動作などを含む。

2. Site Load Flow(SiteLoadFlow)。このFlowは、1つの試験計画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、任意のサイト特有の初期化を行うことができる。

3. Lot Start/End Flow(LotStartFlow/LotEndFlow)。これらのFlowは、試験計画サーバが新たなロットの開始を通知されるときに、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で注釈を付すために製造環境において用いられる。

4. DUT Change Flow(DutChangeFlow)。このFlowは、そのDUT情報が変化するとき、サイトコントローラ上で実行される。再び、これは典型的には、アナログストリームを更新するために製造環境において用いられる。

5. TestPlan Start/End Flow(TestPlanStartFlow/TestPlanEndFlow)。これらのFlowは、試験計画サーバが、現在のTest Flowを実行し始めるように指示されるとき、およびそのフローがその実行を終了するとき、サイトコントローラ上で実行される。

6. Test Start/End Flow(TestStartFlow/TestEndFlow)。これらのFlowは、Test Flowが新たなTestを実行し始めているとき、およびそのTestがその実行を終了するとき、サイトコントローラ上で実行される。

7. Test Flow(TestFlow)。このFlowは、試験計画サーバが「Execute Test Plan」メッセージを受信するときに実行される主要Flowオブジェクトであることに留意されたい。

【0330】

ユーザが、TestFlowでないか、または他の所定のフローのうちの1つでない、ユーザの試験計画内の1つのFlowを定義する場合には、それを実行させるための好ましい方法は、これらの所定のフローのうちの1つの遷移状態内にそれを含むことであることに留意されたい。

【0331】

[試験計画例]

以下の例では、フローによってインプリメントされる有限状態機械を記述するコメントとともにFlowが与えられる。その有限状態機械は、遷移行列として与えられる。その行列の行はFlowItemに対応し、列は結果に対応する。その行列の1つの行のエントリは、返された結果が列において指定された値であるときに、その行のFlowItemから遷移されるFlowItemを指示する。

【0332】

3つのフロー、FlowTest1、FlowTest2およびFlowMainを有する試験計画が以下に示される。FlowTest1は先に説明されたように動作するであろう。それは、「min」、「typ」および「max」の各構成において、MyFunctionalTest1と名前を付けられた試験を実行するであろう。同様に、FlowTest2は、これらの各構成において、MyFunctionalTest2を実行するであろう。最後に、FlowMainは、FlowTest1およびFlowTest2を実行するであろう。これらの各フローの開始時に、コメントにおいて、有限状態機械遷移行列が与えられる。

【0 3 3 3】

```
# -----
# File mySimpleTestPlan.tpl
# -----
10

Version 0.1;

Import xxx.pin;    # Pins

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

# Import test condition groups
Import myTestConditionGroups.tcg;
20

# 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
    pllA.plist:pat1Alist,
    pl2A.plist:pat2Alist
}
30

# The socket for the tests in this test plan (this is not imported,
# but resolved at activation time):
SocketDef = mytest. soc;
40

# Declare some user variables inline
UserVars
{
    # String name for current test
    String CurrentTest = "MyTest";
}
50
```

```

TestConditionTC1Min
{
    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 theTC1Min TestCondition.
#
Test FunctionalTest MyFunctionalTest1Min
{
    PListParam = pat1AList;
    TestConditionParam =TC1Min;
}

# Another FunctionalTest selecting TCG1 with "typ"
Test FunctionalTestMyFunctionalTest1Typ
{
    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;
}

```

10

20

30

40

50

```

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

Test FunctionalTest MyFunctionalTest2Max
{
    PListParam patlAList;
    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 areUnsignedIntegers 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 numberof 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

```

10

20

30

40

50

```

# Result. On failure, it returns 1 as a failing Result.
#
# Assume that the testsMyFunctionalTest1Min, ... 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      10
# 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      20
{
    FlowItem FlowTest1_Min MyFunctionalTest1Min
    {
        Result 0
        {
            PropertyPassFail = "Pass";
            IncrementCounters PassCount;
            GoTo FlowTest1_Typ;
        }
        Result 1,2      30
        {
            PropertyPassFail = "Fail";
            IncrementCounters Fail Count;
            Return 1;
        }
    }
    FlowItem FlowTest1_Typ MyFunctionalTest1Typ
    {
        Result 0
        {
            PropertyPassFail = "Pass";
            IncrementCounters PassCount;
            GoToFlowTest1_Max;
        }
        Result 1,2
        {
            PropertyPassFail ="Fail";
            IncrementCountersFailCount;
            Return 1;
        }
    }
}

```

```

}
# Likewise for FlowTest1_Max
FlowItem FlowTest1_Max MyFunctionalTest1Max
{
    Result 0
    {
        PropertyPassFail = "Pass";
        IncrementCounters PassCount;
        Return 0;
    }
    Result 1,2
    {
        PropertyPassFail = "Fail";
        IncrementCountersFailCount;
        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 testsMyFunctionalTest2Min, ... 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 eachFlowItem 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
        {
            PropertyPassFail "Pass";
            IncrementCounters PassCount;
            GoTo FlowMain_2;
        }

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

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

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

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

            Return 0;
        }
    }
}

```

```

    }

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

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

        Return 1;
    }
}

```

TestFlow = FlowMain;

【0 3 3 4】

上記の試験計画は、好ましい順序で以下のように構成される。

1. 最初に、バージョン番号が与えられる。この番号を用いて、コンパイラバージョンの互換性が確保される。
2. その後、多数のインポートが宣言される。これらは、その試験計画において用いられる名前を分解するために必要とされる宣言を有する種々のファイルである。
3. 次に、試験計画名が宣言され、その後、試験計画のインライン宣言を行う。
4. 次に、1組のPListDefsが宣言される。これらは、名前付きファイルからGlobalPListsの名前を付ける、ファイル修飾付きの名前を含む。それらは、パターンリスト変数も含む。パターンリスト変数は、実行時にcustom flowableにおいて初期化されることができ変数である。それらは、ランタイムまで、試験と実際のパターンリストとの結合を遅らせる手段を提供する。
5. 次に、1組のUserVarsが宣言される。これらはストリングを含む。
6. その後、合格した試験および不合格の試験の数を判定するために、いくつかのカウンタが宣言される。カウンタは変数に過ぎず、初期化されて0になり、IncrementCounterステートメントにおいてインクリメントされる。それらは、1つのDUTの試験の終了時に現在セットされているピンだけがインクリメントされるという意味を有する、先に説明されたBinとは異なる。
7. 次に、一連のTest Conditionが宣言される。これらはそれぞれ、Test Condition Groupおよびセクタを指定する。この例では、Test Condition Groupは、mytestconditionsgroup.tcgからもたらされる。しかしながら、それらは試験計画においてインラインにすることができる。
8. 次に、一連のFlowableまたはTestが宣言される。これは、パターンリストおよび試験条件を選択する既知のTest FunctionalTestである。こうして、たとえば、MyFunctionalTest1Maxは、試験条件TC1Maxおよびパターンリストを選択する。
9. この後に、3つのフロー、FlowTest1、FlowTest2およびFlowMainが宣言される。FlowはFlowableを実行する。Flowableは、Test (MyFunctionalTest1Maxなど) および他のフロー (FlowTest1およびFlowTest2など) を含む。FlowTest1およびFlowTest2はそれぞれ、Test1およびTest2の最小、典型および最大バージョンを通る。フローFlowMainは、先に宣言されているフロー、FlowTest1およびFlowTest2をコールする。
10. 最後に、TestFlowイベントがFlowMain Flowに割り当てられる。したがって、フローFlowMainは、ユーザがこの計画を実行しようとするときに、この試験計画によって実行されることになるフローである。

【0335】

[FlowのためのC++]

上記の規則を用いて、Flowそのものを除いて、要素の大部分のためのC++インプリメンテーションを果たすことができる。

【0336】

[FlowItemのためのC++]

FlowItemを表すためのC++クラスは以下のインターフェースを有することができる。

・このFlowItemのために実行されることになるIFlowableをセットすることになる演算。

【0337】

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

10

【0338】

一旦、このIFlowableを実行するための必要とされる1組のコールからFlowItemが戻ったなら、それは、Result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。これを実現するために、FlowItemは、インクリメントされることになるカウンタのベクトルを有する必要がある。これは以下のコールによって初期化される。

【0339】

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

【0340】

これをコールすることにより、カウンタへの参照のベクトルがFlowItem内に設定され、結果として、一旦、IFlowableが実行を完了したなら、それはそれらのカウンタをインクリメントできるようになる。たとえば、ステートメントIncrementCounters A, B, Cは、以下のように上記のコールを用いることが好ましいであろう。

20

【0341】

```
// 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) ;
```

30

【0342】

カウンタで名前を付けられた一時的なCounterRefListオブジェクトが用いられる。最初に、counters.reset()がコールされ、その後、多数のcounters.add()をコールして、カウンタリストが設定される。その後、これを用いて、結果値2および3のために更新されることになるカウンタアドレスのベクトルが設定される。

40

【0343】

その後、FlowItemは、特定の結果において別のFlowItemに遷移する必要があると得る。

【0344】

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

【0345】

ある特定のResult句が多数の結果値を取り扱う場合には、必然的にいくつかのそのようなコールが行われる必要があると得る。

【0346】

50

FlowItemは結果を返す必要があり得る。これは以下のコールによって果たされる。

【0347】

```
Status setResult(unsigned int result,
                  unsigned int returnResult);
```

【0348】

たとえば、以前の例におけるFlowItem FirstFlowItemの場合、上記のコールは、「result」の場合には値「2」で、「returnResult」の場合には「1」でコールされるであろう。

・最後に、FlowItemは演算を実行する必要がある。

【0349】

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

【0350】

この演算は、IFlowableを実行し、その後、指示されたカウンタを更新し、その後、Result、または次のFlowItemへのポインタを返すであろう。このポインタがヌルである場合には、その結果は返された値である。

【0351】

FlowItem FlowMain_1のために生成されることになるコードは以下の通りである。

【0352】

```
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) ;
```

【0353】

上で生成されたコードはFlowMain_1を設定し、IFlowable「FlowTest1」を実行し、その後、それを設定して、結果毎にカウンタの適当なリストをインクリメントし、最後に、必要な動作を行う。結果「0」の場合の必要な動作は、FlowMain_1への遷移であり、結果「1」の場合には、リターンである。

【0354】

[C2. TestPlanにおけるカウンタサポート]

カウンタは0に初期化される変数であり、試験実行中の種々の時点において、IncrementCounterステートメントによってインクリメントされることができる。これは、試験の終了時にのみインクリメントされるBinとは異なる。さらに、ピンは階層的であるのに対して、カウンタは変数に過ぎない。したがって、カウンタはピンよりもはるかに簡単であり、より制限されたファシリティである。

10

20

30

40

50

【0355】

カウンタは、符号なし整数である1組の名前付きカウンタを保持するCounterクラスのメンバを介して、TestPlanにおいてサポートされることができる。Counter宣言を介して、このクラスにおいてオブジェクトが定義されるであろう。カウンタは試験開始時に自動的にリセットされないの、TestPlanが、数多くのDUTを試験し終わるまでカウントを収集できるようになる。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ためのメソッドが与えられるであろう。これにより、順番にピンに入れることに対する代替の手段が、試験を実行する結果としてのカウントを判定できるようになる。

【0356】

TestPlanはメンバ変数、m_modifiedCountersを含むことが好ましく、そのメンバ変数は、1つのDUTにおいて試験を実行することによって変更される1組のカウンタである。この1組のカウンタは試験の開始時に空集合に初期化される。各場所において、IncrementCountersコールが行われ、コードが生成されて、名前付きカウンタをm_modifiedCounterメンバに追加するであろう。こうして、このメンバは、1つのDUTで試験を実行中に変更された全てのカウンタを集める。

10

【0357】

[FlowオブジェクトのためのC++]

一旦、全てのFlowItemが作成されたなら、Flowオブジェクトは、以下に示されるように、C++オブジェクトとして作成されることができる。

・FlowItemを追加するための演算。

20

【0358】

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

【0359】

その演算は、指示されたFlowItemをそのFlowに追加するであろう。これがFlowの初期FlowItemである場合には、ブーリアンが真にセットされる。

・Flowを実行するための演算。

【0360】

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

【0361】

これは、そのフローを実行する結果として、Flowが戻るときに返ることが好ましい。この動作は、初期FlowItemでそのフローを実行し始めることである。それは、現在のFlowItemが実行すべき次のFlowItemを返す限り、FlowItemを実行し続けるであろう。現在のFlowItemが結果を返すとき、この演算は、その結果で完了する。

30

【0362】

それゆえ、1つのFlowのために生成されるC++コードは、そのFlowにFlowItemを追加するために、addFlowItem()へのコールを何度か繰り返す。executeFlow()演算は、TestPlan内のこのFlowが実行するために選択されるときに生じるであろう。

【0363】

[C3. Testクラス]

一般に、プログラムコードの大部分はデバイス試験のためのデータであり、残りは、試験方法を実現する試験プログラムのコードである。このデータはDUTに依存する（たとえば、電源条件、信号電圧条件、タイミング条件など）。試験コードは、ATEハードウェアに指定されたデバイス条件をロードするためのメソッドと、ユーザによって指定された目的（データロギングなど）を実現するために必要とされるメソッドから構成される。

40

【0364】

先に説明されたように、試験コードの再利用性を高めるために、そのようなコードは任意のデバイス固有データ（たとえば、ピン名、刺激データなど）、またはデバイス試験固有のデータ（たとえば、DCユニットのための条件、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など）とは無関係であるべきである。1つの試験のためのコードがこれらのタイプのデータとコンパイルされる場合には、そ

50

の試験コードの再利用性は低下するであろう。それゆえ、任意のデバイス固有のデータまたはデバイス試験固有のデータは、コード実行時の入力として、その試験コードが外部から入手できるようにすべきである。

【0365】

オープンアーキテクチャ試験システムでは、ITestインターフェースの1つのインプリメンテーションであるTestクラスが、特定のタイプの試験のための試験データとコードとの分離を（それゆえ、コードの再利用性を）実現する。そのような試験クラスは、その別個のインスタンスのための「テンプレート」と見なすことができ、デバイス固有のデータおよび／またはデバイス試験固有のデータに基づいてのみ互いとは異なる。試験クラスは、試験計画ファイルにおいて指定される。各試験クラスは典型的には、ある特定のタイプのデバイス試験またはデバイス試験のための設定をインプリメントする。たとえば、機能的なACおよびDCパラメータ試験は、別個の試験クラスによってインプリメントされることが好ましい。しかしながら、カスタム試験クラスも試験計画において用いることができる。

10

【0366】

試験クラスによって、ユーザは、その試験の特定のインスタンスのためのオプションを指定するために用いられるパラメータを与えることにより、クラス挙動を構成できるようになる。たとえば、1つの機能試験は、パラメータPリストおよびTestConditionsを取り込み、実行すべきパターンリスト、ならびにその試験のためのレベルおよびタイミング条件をそれぞれ指定するであろう。これらのパラメータのために異なる値を指定することにより（試験計画記述ファイル内の異なる「Test」ブロックを用いることによる）、ユーザは、1つの機能試験の種々のインスタンスを作成できるようになる。図5は、ただ1つの試験クラス504から異なる試験インスタンス502が如何に導出されることになるかを示す。

20

【0367】

これらの試験クラスは、コンパイラ400が試験および試験のパラメータの記述を試験計画ファイルから取り込み、正確なC++コードを生成できるようになり、そのC++コードがコンパイルおよびリンクされて、試験プログラムを生成することができるようにより設計されるべきである。試験クラスインスタンスを、試験フローを記述するオブジェクトに追加して、デバイス試験の複雑な実行シーケンスを作成することができる。

30

【0368】

[C4. ITestおよびIFlowableからの導出]

先に述べられたように、試験クラスはITestから派生する。上記の規則を用いて、これらは、ITestインターフェースをインプリメントするC++クラスにおいてインプリメントされることができる。ITestインターフェースのための指定されるメソッドに加えて、これらのクラスは、デバイス試験の特定のクラスを実行するために必要とされるTest固有のインテリジェンスおよびロジックを提供する。また試験クラスはIFlowableインターフェースもインプリメントする。この結果として、Testクラスのインスタンスは、試験を実行するためにFlowItemにおいて用いることができる。

40

【0369】

[カスタマイゼーション]

ユーザがC関数をコールできるようにし、ITestおよびIFlowableインターフェースをインプリメントする自らのクラスを開発できるようにするために、カスタマイゼーションの仕組みが提供される。

【0370】

[イントロスペクション能力]

1つの試験クラスのオブジェクトが、そのメソッドおよびシグネチャに関して問合せを受けることができる場合には、そのオブジェクトは、生成されるソースコード内に包含するのに適したパラメータが入手できることを検証されることができる。そのような特徴は、翻訳段階における誤り検査および妥当性検査のために極めて有用であろう。試験技師が

50

パラメータの名前、またはこれらのパラメータへの引き数の数（またはおそらくタイプ）を間違った場合には、C++コンパイラからのコンパイル時エラーメッセージを待つ代わりに、翻訳段階において、それが見つけられて、翻訳時に意味のあるエラーメッセージを与えることができる。これは、試験技師にとってさらに有用であろう。

【0371】

イントロスペクションは、オブジェクトに自らの内部を探索し、その属性およびメソッドに関する情報を返すように求める能力を指している。Javaのようないくつかの言語は、その言語の一部としてこの能力を提供する。ビジュアルベーシックのような他の言語は、それとともに用いられることになるオブジェクトにそのような要件を課す。C++は、この機能を提供しない。

10

【0372】

この方法は、デフォルトパラメータ値、およびオプションのパラメータの指示を提供するのにも非常に役に立つ。さらに、この能力が全てのTest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る場合には、GUIアプリケーションがこの情報を用いて、ダイアログおよび他のユーザインターフェース要素を動的に構築し、技師がこれらのクラスを効率的に使用することを助けることもできる。

【0373】

このように複雑であることは、十分なイントロスペクションの代わりに、Testクラス開発者がそのクラスをパラメータ化するために必要とされるものとして設計したTestクラスの公開メソッド／属性を、ただ1つ（Testクラス当たり）のテキスト系ソースファイルにおいて完全に指定できるようにする方法を提供する仕組みを用いる本発明の1つの実施形態では相殺される。

20

【0374】

ただ1つのソースが好ましい。1つのファイル内に1つのTestクラスのパラメータ化インターフェースの記述と、別の個別の（ヘッダ）ファイル内にC++インターフェース記述とを有し、その後、両方のソースを同期させておくための要求を負担することは望ましくないであろう。このような目的で、その試験クラスのためのプリヘッダファイル内に「テキスト系」の記述が埋め込まれ、それがイントロスペクションを制限し、かつその試験クラスのための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、Test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。

30

【0375】

[プリヘッダ]

C++においてヘッダを用いることはよく知られている。しかしながら、C++はパースし、読み出すのが難しいので、本発明の一実施形態は、コンパイラが、試験クラス開発者がヘッダとして用いることができるC++出力を作成できるようにする構文を定義する。この実施形態によれば、試験クラス開発者はプリヘッダファイルを書き、それがプリヘッダファイルとしてコンパイラ400によって出力され、それにより、対応する試験クラスまたは他の試験実体において見るようにすることができるようにする。

40

【0376】

以下の例は、本発明の好ましい実施形態による、1つのTestクラスのためのプリヘッダファイルの概念を例示する。試験FuncTest1について、ソースファイルから抜粋された以下の部分を考える。

【0377】

```
...
TestCondition TC 1
{
    TestConditionGroup = TCG1;    # Previously defined TCG for Levels
    Selector = min;
}
```

50

```

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

```

【0378】

上記のFuncTest1の宣言が正当であるか否かを判定するために、コンパイラは、FunctionalTestが何を引き起こすかを知る必要がある。FunctionalTestの知識をコンパイラに組み込むのではなく、FunctionalTestが何を引き起こすかの定義を、プリヘッダにおいて指定することができる。

【0379】

FunctionalTestが、ベースクラスTest1およびTest2を有し、かつPListであるメンバおよびTestConditonのアレを有するC++クラスであると仮定する。コンパイラは、FuncTest1の上記の宣言が正当であることを認識するために、FunctionalTestのメンバのタイプについて知る必要がある。

【0380】

さらに、FuncTest1のためのC++オブジェクト宣言を生成するために、クラスFunctionalTestのためのC++ヘッダが構成される必要がある。これは、コンパイラに対して、FunctionalTestクラスのベースクラス、そのメンバの名前および他のそのような情報について知ることも要求する。

【0381】

本発明の一実施形態のプリヘッダ部分言語は、コンパイラに、宣言の正当性を認識し、かつ宣言に対応するC++ヘッダおよびオブジェクト宣言を生成するために必要とされる情報を提供する。

【0382】

FunctionalTestは1つの単純なタイプ（パラメータ化が関係している限り）であり、それゆえ、パラメータ化のためにかなり単純な記述を使用するであろうことに留意されたい。こうして、以下のように、上記のパラメータ化をサポートするプリヘッダ、FunctionalTest.phを書くことができる（プリヘッダはベース試験クラスTest1及びTest2について利用可能であると仮定する）。

【0383】

```

Version 1.0;
#
# Parameterization specification pre-header for FunctionalTest
#
ImportTest1.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 voidexec();
    virtual voidpostExec();
    $ParamFns
    ...
private:
    double m_someVar;
    $ParamAttrs
    ...
};
...
$ParamImpls
} // End namespace
CPlusPlusEnd

```

10

【0384】

[パラメータ化された試験クラスのためのC++]

コンパイラがプリヘッダファイルを処理するとき、コンパイラは、\$Inc、\$Class、\$ParamAryTypesなどのコンパイラ変数の値を蓄積する。これにより、その後、コンパイラは上記と全く同じようにC++コードを生成し、指示された場所においてコンパイラ変数\$Inc、\$Classなどの値に拡張することにより、以下のC++ヘッダを作成できるようになる。FunctionalTest.phの場合、コンパイラは、FunctionalTestクラスのための以下のC++ヘッダファイルFunctionalTest.hを作成する。

20

【0385】

```

#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"

```

30

40

50

```

public:
#line 37 "./FunctionalTest.ph"
    typedef std::vector<Tester::TestCondition *>
TestConditionPtrsAry_t;
#line 62 "./FunctionalTest.ph"
public:
    virtual void preExec();
    virtual voidexec();
    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"
    StringgetPListParamDescription0 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_tm 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_testConduArray.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::StringFunctionalTest::getTestConditionParamDescription()
{
    return "The TestCondition parameter for a FunctionalTest";
}
#line 75"./FunctionalTest.ph"
} // End namespace

```

10

【0386】

先に説明されたように、このプリヘッダによって、コンパイラはFunctionalTest宣言の妥当性を検査し、そのためのコードを生成し、それによって必要とされることになるC++ヘッダを生成できるようになる。

20

【0387】

一例として、便宜上、以下に再現される、先に与えられたFunctionalTest宣言について考える。

【0388】

```

Test FunctionalTestFuncTest1
{
    PListParam = patList1;          # Previously defined pattern list
    TestConditionParam = TC1;
    TestConditionParam = TC2;
}

```

30

【0389】

このためにコンパイラによって生成されることになるC++ヘッダは先に与えられている。コンパイラは、上記のFunctionalTest構成体のための以下のコードを生成するのであろう。

【0390】

```

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

```

40

【0391】

Description関数のために生成される名前にも注目されたい。Xxxと名前を付けられる各パラメータは以下のメンバ関数に関連し、それは、GUIが利用できるツールチップのための記述を有するストリングを返す。

【0392】

```

Status getXxxDescription() const;

```

【0393】

[他のプリヘッダの特徴]

50

プリヘッダは、付加的なタイプとして、いくつかの他のユーザ定義列挙法をサポートする。これにより、GUIは、ある特定のパラメータの値をセットするために用いることができる、取り得る選択肢のドロップダウンリストを提供できるようになる。さらに、プリヘッダは、テーブルと考えることができる多数のパラメータを関連付けるための機能を提供する。たとえば、プリヘッダは、「プロパティ」のアレイを、関連する1組の、名前のためのストリングのアレイ、および値のための整数のアレイとしてインプリメントするのに都合良く用いることができる。この機能をインプリメントする1つの簡単な方法は、カスタムタイプ（後に説明される）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、使用するべきカスタムタイププリヘッダを書くことを要求する。これらの機能がいずれも、以下の例において示される。

10

【0394】

```
# -----
# 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
```

20

30

40

50

```

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 setPattemTree.
# - 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

```

10

20

30

40

50

```

$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

```

【0395】

カスタムタイプ名前 - 数値対が宣言されることもでき、そのカスタムタイプのただ1つのアレイパラメータを用いて、パラメータの上記のParamGroupと同じ効果をもたらすこともできることに留意されたい。上に提示された技法は、カスタムタイプを宣言する必要性を避けるのに好都合である。

【0396】

[C5. カスタム関数宣言]

これにより、フロー遷移が行われるときに、ユーザはカスタム関数をコールできるようになる。カスタム関数は、以下のように、プリヘッダを通して宣言される。

【0397】

```

# -----
# 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);

```

10

20

30

40

50

```
# 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);
```

10

【0398】

典型的には、コンパイラが標準的な方法でこれらの宣言を拡張するとき、上記の宣言のためにC++セクションが与えられる必要がある。当然、ユーザはこれらの関数のC++インプリメンテーションに対して責任を持つ。上記の関数は全て、おそらく、暗示的な第1のパラメータとしてITestPlanポインタを取り込むことに留意されたい。このポインタは、関数ライタがTestPlan内の状態Sにアクセスできるようにする。たとえば、関数ライタは、現在のFlow、そのフロー内の現在のFlowItem、現在のResult句、UserVarsの値および他のそのような情報にアクセスするために、ITestPlanインターフェースを用いることができる。ファイルFunctions.phにおいて用いるために、ある特定のテストによって定義される関数を利用することができる。

20

【0399】

```
Version 1.2.3;
```

30

```
#
# 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.
```

40

```
UnsignedIntegerGetDUTID();
```

【0400】

[カスタム関数宣言のためのC++]

上記のMyFunctionsのためにコンパイラによって生成されることになるC++コードは、MyFunctions名前空間において単純にいくつかの関数を宣言することである。

【0401】

```
namespace MyFunctions
{
    double Min(ITestPlan* pITestPlan, int& x, int& y);
    double Avg(ITestPlan* pITestPlan, double* a, const int a_size);
```

50

```
void Print(ITestPlan* pITestPlan, char* Msg, unsigned int dutID);
}
```

【0402】

これらの関数は1つのフローからコールすることができるであろう。

【0403】

[C6. カスタムFlowables]

プリヘッダを用いて、C++ IFlowableインターフェースをインプリメントするプリヘッダを作成することもできる。これにより、ユーザは、FlowItem内で実行することができるカスタムフローアブルを定義できるようになる。以下に示されるのは、ユーザ定義のFlowable MyFlowableのためのプリヘッダである。

10

【0404】

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

Version 1.2.4;

```
FlowableClass = MyFlowable;      # The name of this custom class
```

20

```
# 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
```

30

```
{
```

```
    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
```

40

```
#   - 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]
```

50

```

# - stored in a member named m_stringArrVal
# - a function to set it named addStr.ingVal.
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
```

【0405】

IFlowableインターフェースをインプリメントするいくつかのクラスが存在する。これらは以下のものを含む。

1. 現在のテスト構成内で試験計画を実行することができるか否かを検査することになるプログラムローディングのためのFlow。

2. 特定のパターンおよびパターンリストをロードすることになるパターンローディングのためのFlow。

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、他の初期化および妥当性検査機能を実施することになる初期化のためのFlow。

4. 他の一般的に有用な試験フロー。

【0406】

[C7. カスタムタイプ]

先のTestクラスパラメータ化に関する説明は、既知のタイプ、すなわち基本タイプ並びにPListsおよびTestConditionsのようなテスト定義のタイプを有するような試験クラスパラメータの場合のみ許される。ユーザの自由度を高めるために、タイプ拡張性を与え、それにより、複数のタイプ（コンパイラには事前にはわかっていない）を作成し、使用できるようにすることが重要である。カスタムタイプ（CT）はCustom Typeにおいて定義されるであろう。これらのカスタムタイプを用いて、C言語ストラクトに対応するタイプ（Plain Old Dataタイプ、すなわちPODとも呼ばれ、C++の同じ名前のもとは大きく異なる）および関数シグネチャのためのC言語タイプ定義に対応するタイプを定義することができる。ユーザタイプを含む別個のファイルは拡張子.ctypを有するであろう。ここに示されるのは、本発明の好ましい実施形態によるユーザタイプ宣言の一例である。

【0407】

```
# -----
# 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;
    }
}
```

10

20

30

40

50

```
#
# 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 () Void;
```

```
}
```

【0408】

[カスタムタイプのためのC++]

先に提示されたCustomType宣言は、コンパイラによって以下のC++コードに翻訳されるであろう。

【0409】

```
namespace CustomTypes
{
```

```
    struct Foo
    {
        Tester:: String S1;
        int I1;
        Tester:: String S2
    };
    struct Bar
```

```
    {
        Foo Foo 1;
        Tester:: String S1;
        Foo Foo2;
    };
    typedef int (*BinaryOp) (int&, int&);
    typedef void (*UnaryOp)(int);
    typedef void (*NullaryOp)();
```

```
}
```

【0410】

これらのタイプのオブジェクトは、次に示されるように、パラメータとしてTestクラスに渡すことができる。

【0411】

[試験クラスパラメータとしてのカスタムタイプの使用]

ユーザがある試験を拡張する場合について考えると、その試験は、パターンリストおよび試験条件に加えて、他のクラスオブジェクト、ならびにCustom Type (すなわち、.ctyp

10

20

30

40

50

ファイル)を含む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。たとえば、ユーザがファイルMyTestCTs.ctypにおいて定義されるC Tを用いることを望むものと仮定する。

【0412】

```
# File MyTesetCTs.ctyp
Version 1.0;
```

```
CustomTypes
```

```
{
```

```
    Pod Foo
```

```
{
```

```
    String name;
```

```
    PList patternList;
```

```
}
```

```
    Pod Bar
```

```
{
```

```
    Foo someFoo;
```

```
    Double dVal;
```

```
}
```

```
    RoutineBinaryOp(Integer, Integer) return Integer;
```

```
}
```

【0413】

上記のタイプを用いるためにユーザがする必要がある全てのことが、ユーザの試験クラスプリヘッダにおいて上記のファイルをインポートされる。コンパイラが、そのように定義されたC Tを解釈するので、コンパイラが試験クラスプリヘッダを処理しているときに、FooおよびBarのための定義を入手することができる。さらに、コンパイラは、上記のタイプFooおよびBarにそれぞれ対応する2つのC言語ストラクト、すなわちストラクトFooおよびストラクトBarを定義し、それらの定義がファイルmyTestCTs.h内に置かれる。myTestCTs.cttのためのImportステートメントによって、ファイルmyTestCTs.hは、生成された試験クラスC++ヘッダ内に含まれるようになる。以下の例はこのプロセスを例示する。最初に、試験計画内の試験のための宣言を考える(パターンリストおよび試験条件のための宣言は、明確にするために省略されている)。

【0414】

```
...
```

```
Import MyFunctions.ph;
```

```
Import MyCustomTypes.ctyp;
```

```
...
```

```
# TheCustomVars block defines variables of the Custom
```

```
# types defined earlier.
```

```
CustomVars
```

```
{
```

```
    ...
```

```
    Bar bar 1 =
```

```
{
```

```
    {"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;
}
...

```

10

【0415】

上記の例では、試験計画にCustomVarsブロックが含まれる。カスタム化変数を含む別個のファイルは拡張子.cvarを有するであろう。ユーザは、以下のようにして、上記のパラメータ化をサポートするMyFancyTestのためのプリヘッダを書くであろう（パターンリストおよび試験条件のためのパラメータ化宣言は、明確にするために省略されている）。

【0416】

20

```

# -----
# 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 mBarsArray
    # - a function to set it named addBarParam.
    # An implementation. will be generated for addBarParam.
    Bar BarParam
    {
        Cardinality = 0-n;
        Attribute =mBarsArray;
        SetFunction = addBarParam [Implement];
    }
    # The following declaration specifies that a MyFancyTest has
    # - an optional parameter of custom type BinaryOp

```

30

40

50

```

# - [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];
}

```

10

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 .

```

20

30

【0417】

[カスタムタイプを用いるカスタム試験クラスのためのC++]

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したなら、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、すなわちMyFancyTest.hを作成するであろう。

40

【0418】

```

#include <MyCustomTypes.h>
#include <ITest.h>
#include <FunctionalTest.h>
...
namespace
{
class MyFancyTest : public ITest,
                    public Functional Test

```

50

```

{
public:
    typedef std::vector<CustomTypes::Bar *> Bar Ary_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

```

【0419】

[C8. パラメータ化]

先に示されたように、Testクラス、カスタムFlowableクラス、またはカスタム関数定義のためのプリヘッダは、パラメータ化仕様セクションを通して、クラス／関数へのイントロスペクションを制限する。コンパイラはこのセクションを用いて、クラス／関数のためのパラメータ化インターフェースを生成する（そして、クラス／関数ヘッダそのものを生成する）。TestクラスおよびFlowableクラスの場合、コンパイラはまた、このセクションを用いて、Test Planコード内のコールを後に生成し、そのクラスのインスタンスを初期化する。プリヘッダおよび対応する宣言に関する以下の点に留意されたい。

1. 全てのTestまたはカスタムFlowableクラス定義は、プリヘッダ内で指定されることが好ましい。プリヘッダ内のParametersブロックは、そのようなクラスのためのパラメタリストを指定することができる唯一の場所であることが好ましい（したがって、結果と

して、パターンリストおよび試験条件仕様のようなTestのための「標準的な」パラメータも、プリヘッダのParameters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C Tが一樣に処理されるようになる）。

2. TestまたはFlowableクラスのためのプリヘッダにおいて非オプションとして定義される（すなわち、0以外の基数を有する）全てのパラメータが、そのクラスのインスタンスのためのTestブロックまたはFlowableブロック宣言の中で初期化されるべきである。

3. Test/Flowableブロックにおいてパラメータの初期化のために用いられるオブジェクトは予め定義されているべきである。

4. 置換指示子\$Class、\$Inc、\$ParamAryTypes、\$ParamFns、\$ParamAttrsおよび\$ParamImplsが、ユーザが対応する生成されたコードを生成されたクラスヘッダファイル内に挿入させるつもりである、プリヘッダのユーザコードセクション内の正確な場所に現われなければならない。置換指示子毎に特定のコードが生成されるので、これらの置換指示子は厳密に一度だけ現われるべきである。

10

5. プリヘッダのParametersブロック内のパラメータ仕様の名前（上記の例のPListParam、TestConditionParamまたはBarParamのような）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ことになるパラメータの名前である。

6. 以下はパラメータ仕様において用いられる記述子の意味である。

a. Cardinality：これはサポートされることになるこのタイプのパラメータの数を指示する。以下は一実施形態において取り得る値である。

i. 1：このパラメータは強制的であり、厳密に一度だけ指定されるべきである。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であろう。

20

i i. 0 – 1：このパラメータはオプションである。指定される場合には、それは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であろう。

i i i. 1 – n：このパラメータは強制的である。さらに、このパラメータのために多数の値を指定することができる。この値は指定された順序で記憶されるであろう。

i v. 0 – n：このパラメータはオプションである。このパラメータのために多数の値を指定することができる。この値は指定された順序で記憶されるであろう。

上記の（）および（）の場合に、全ての指定された値が、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、S T Lベクトル<>に記憶されることになることに留意されたい。このベクトルのタイプは定義され、\$ P a r a m A r y T y p e sによって指示される場所に挿入されるであろう。これらのタイプ定義のためのアクセスレベルは常に公開である。

30

b. Attribute：このタイプのパラメータ値（複数可）のための記憶として用いるためのC++変数の名前。その名前は、C++クラスの私用データメンバとして全く同じ語で再現されることになり、C++識別子のための要件に準拠しなければならない。この属性のタイプに関して、以下のことに留意されたい。

i. ただ1つの値だけが許される場合には、そのパラメータのタイプへのポインタである。

40

i i. 多数の値が許される場合には、そのパラメータのタイプへのポインタ上にテンプレート化されるS T Lベクトル<>である（上記の（）を参照）。

それらの属性はTest Planによって作成され、ポピュレートさ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ことに留意されたい。オブジェクトの寿命は常に、Test Planそのものによって管理される。

c. SetFunction：このパラメータのための値をセットするために用いるための関数の名前。以下の点に留意されたい。

i. その名前は全く同じ語で再現されることになり、それゆえ、C++言語要件に準拠しなければならない。

i i. その関数のアクセスレベルは常に公開である。

50

i i i. リターンタイプは常に空である。

i v. その関数は常に、タイプポインタ／パラメータタイプのただ1つの引き数をとる。

【0420】

1つの値が常に1つずつセットされる；すなわち、多数の値を指定できるようにするパラメータの場合に、Test Plan内の生成されたコードが、指定された値毎に一度、この関数を繰り返しコールし、各値がS T Lベクトル（先に説明されたような）に追加されることになることに留意されたい。

【0421】

関数名に続くオプションのキーワード [インプリメント] は、この関数のための普通のインプリメンテーションが、クラスヘッダ (\$ParamImplsによって指示される場所において挿入される) においてインラインメソッドとして利用できるようになることを指示する。そうでない場合には、ユーザがその関数のインプリメンテーションを提供する役割を担う。

d. Description: このパラメータのランタイム変更中にヘルプを提供するために、G U I ツールによって用いられることになるツールチップであるストリングリテラル。Xx xと名前を付けられたパラメータのためのカスタムクラスにおいて生成されるC++メンバ関数は以下の通りであり、その関数は指定されたストリングを返すであろう。

【0422】

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

【0423】

[カスタム化を用いる試験計画例]

以下に示されるのは、いくつかのカスタム化を用いて説明される試験計画例である。

【0424】

```
# -----
# 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
```

10

20

30

40

50

```

# 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 theTC1Min 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 eachFlowItem 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 ("PassedMyFlowable1",
                    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_5DatalogStopFlow
{
    Result 0
    {

```

```

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

```

10

```

Result 1
{
    # FlowTest1 and FlowTest2 passed,
    # but DatalogStopFlow failed
    Property PassFail = "Fail";
    IncrementCounters FailCount;
    # A user function call
    MyFunctions.Print("Failed DatalogStopFlow",
                      Functions.GetDUTID());
    Return1;
}
}
}

```

20

【0425】

上記のコードにおいて、以下の点について留意されたい。

1. ここでPListDefsセクションは、いくつかのPList名とともに、いくつかのPList変数も有する。PList名は、試験において直に用いることができる名前である。PList変数は試験において用いることができる変数であり、その値は、ランタイムにおいて、カスタム化されたFlowable内のコードによって実際のPListに結び付けられる。

2. PListDefsセクションはオプションである。存在しない場合には、その内容は種々のTest宣言からコンパイラによって推測されるであろう。存在する場合には、それは、用いられるTestのPListパラメータの全てを宣言しなければならないが、それはさらに多くのパラメータを宣言することもできる。

30

3. 値をPList変数に割り当てるために、ランタイムAPIを利用できるであろう。TestPlanクラスは以下の関数を有するであろう。そのFlowableはPListVariableを特定のPListに結び付けるためにその関数を用いることができるであろう。

【0426】

```

Status SetPListVariable(const Tester::String& varName,
                        const Tester::String& fileQualifiedPListName);

```

【0427】

40

4. 別のFlowableに制御を渡すことか、リターンかのいずれかである遷移の直前に、FlowItemにおいてユーザ関数および関数をコールすることができる。

【0428】

[ユーザ関数コールのためのC++]

複数のフローにおいてカスタム関数コールを呼び出すことを除いて、コンパイラによって生成されることになるC++コードが、先に提示された種々のカスタム化技法のために示されている。FlowItem内のユーザ関数コールは、各FlowのIUserCallsメンバによって取り扱われることが好ましい。各Flowは、以下に示されるように、単一の仮想メンバ関数をエクスポートするインターフェースIUserCallsのメンバを有することが好ましい。

【0429】

50

```

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

```

【0430】

ユーザ関数コールを有するFlowに直面するとき、そのFlowは、上記のインターフェースをインプリメントするクラスのインスタンスでポピュレートされるようになる。たとえば、その例内のFlowMainでは、フローは以下のクラスのインスタンスでポピュレートされるであろう。

10

【0431】

```

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")
        {
            // ...
        }
    }
};

```

20

30

40

50

【0432】

FlowItem::execute()演算はフロー項目の名前を知っている。それは次のフローへのポインタとともに戻る前に、それは取り囲んでいるフローのためのIUserCalls::exec()をコールし、その自らのフロー項目名、および現在の結果の値を渡すであろう。これにより、上記のコードが実行されることになり、必要とされるユーザ定義関数が呼び出される。

【0433】

[C9. 試験プログラムコンパイル]

先に説明されたように、Test Plan記述ファイルは、1つの試験計画において用いられる複数のオブジェクトと、それらの互いに対する関係とを指定する。一実施形態では、このファイルはC++コードに翻訳され、それは、標準インターフェースITestPlanのインプリメンテーションの形でサイトコントローラにおいて実行されることになる。このコードは、サイトコントローラ上にロードされることができ、ウインドウズ・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、Test Program DLLは、サイトコントローラソフトウェアが試験計画オブジェクトを生成し、それが含む試験計画オブジェクトを返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。

【0434】

[試験計画記述からの構成]

1つの試験計画記述からITestPlanのインプリメンテーションへの変換のプロセスは、試験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段階、すなわち翻訳およびコンパイルにおいて行われる。

【0435】

翻訳段階402では、コンパイラ400は試験計画ファイル（およびそれがインポートする種々の他のファイル）と、試験計画において用いられる全ての試験タイプのためのプリヘッダとを処理する。この段階では、コンパイラ400は、MSVC++（マイクロソフト・ビジュアル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」コードなどの全ての他のサポート用ファイルとともに、Test PlanオブジェクトのためのC++コード、および出合った試験タイプのためのC++ヘッダを作成する。コンパイラ400は、生成されたコードにファイルおよび行指示文を挿入し、コンパイル時エラーメッセージが、生成されたコードの中を指示する代わりに、記述ファイル内の適当な場所に戻って参照できるようにする。

【0436】

コンパイラが必要なファイルを作成した後に行われるコンパイル段階では、MSVC++コンパイラのような標準的なコンパイラ404が呼び出され、ファイルをコンパイルし、それらのファイルをDLLにリンクする。

【0437】

コンパイラは、入力として、有効な試験計画ファイル（および全ての関連するファイル）を取り込み、必要に応じて、試験計画ファイル内の「インポート」指示文によって表されるTestPlanファイルおよび全ての他のファイルを生成する。さらに、コンパイラは、MSVC++「ソリューション」を生成し、Test Plan DLLを構築する。たとえば、メインファイル(MyTestPlan.tpl)が、タイミング情報を組み入れるためのTiming1.timを含んでいた場合には、コンパイラは、（中でも）以下のファイルを作成するであろう。

【0438】

MyTestPlan.h
MyTestPlan.cpp
Timing1.cpp
MyTestPlan.sln (MSVC++ "Solution" file)
MyTestPlan.vcproj (MSVC++ "Project" file)

【0439】

全てのファイルが作成（または更新）された後に、コンパイラはMSVC++アプリケーション

10

20

30

40

50

ンを呼び出し、そのアプリケーションが作成する「Solution」を指定し、DLLを構築する。エラーおよび／または警告があれば、ユーザに対して示されるであろう。

【0440】

Test Planを構築した後に、ユーザがTiming1.timに変更を加えた場合には、ユーザはコンパイラを呼び出し、コンパイラにMyTestPlan.tplを渡す。コンパイラは（タイムスタンプ情報によって）メイン試験計画ファイルが変更されていないことを認めるので、MyTestPlan.h/.cppは作成し直されないであろう。しかしながら、メイン試験計画ファイルを処理している間に、Timing1.timファイルが変更されていることがわかるであろう。それゆえ、コンパイラはTiming1.cppファイルを作成し直し、MSVC++アプリケーションを呼び出し、DLLを構築し直すであろう。これにより、MyTestPlan.cppをコンパイルし直すのが避けられ、Timing1.cppのみがコンパイルされ、DDLがリンクし直される。この手法は、コンパイルするのにかなりの時間を要する大きな試験計画の場合に、コンパイルおよびリンクし直す時間を切り詰める際に特に有用であろう。

10

【0441】

[D. 試験プログラムの実行]

サイトコントローラソフトウェアは、その処理空間内にTest Program DLLをロードし、DLL内の「ファクトリ」関数をコールして、Test Planオブジェクトのインスタンスを作成する。一旦Test Planオブジェクトが作成されたなら、サイトコントローラソフトウェアは、その試験計画を実行するか、または必要とされる任意の他の方法で試験計画とやりとりすることができる。

20

【0442】

[非インタラクティブビルド]

ウインドウズ環境の大部分のC++ソフトウェア開発者にとってアプリケーション（またはDLL、ライブラリなど）を構築することは、開発環境（MSビジュアルC++、ボーランドC++など）を構築し、コードを編集し、そして（多くの場合に）ボタンを押してプロダクトを構築することを意味する。

【0443】

本発明の1つの実施形態の試験環境は、類似の1組の機能を有するであろう。Test Plan開発者は、コードを編集し、それらのTest Planを構築する必要があるであろう。しかしながら、テストは、結果として生成されるTest Plan DLLを生成するために、Test Plan開発者がC++開発環境を構築することを要求しないであろう。

30

【0444】

これを果たすために、本発明は非インタラクティブビルドの概念を利用する。非インタラクティブビルドは、非インタラクティブモードにおいてMSビジュアルC++を用いるビルドと定義される。この場合でも依然として、他のツールがインタラクティブに用いられ、そのようなビルドを管理できるようになることに留意されたい。唯一意味することは、ビジュアルC++が非インタラクティブに用いられることである。

【0445】

[想定される環境]

ユーザの環境について、ある特定の想定がなされる。それらの想定は以下の通りである。

40

1. Test Plan開発者は、上記の方法および規則に従って、自らのTest Planを開発しているであろう。

2. Test Plan開発者はC++の熟練者レベルの知識を持たない場合もある。

3. Test Plan開発者は、ファイル（複数可）をTest Plan DLLに変換するために、コマンドラインツールまたはGUIツールを利用できるであろう。

【0446】

[ボタンを使用しないアプリケーションの構築]

MSビジュアルスタジオで非インタラクティブに作業するには、2つの手法のうちの1つが必要になる。第1の（そして最も簡単な）手法はコマンドラインインターフェースを

50

用いることである。第2の（そしてより自由度のある）手法は自動インターフェースを用いることである。このセクションは両方の手法を説明する。

【0447】

〔プロジェクトの作成〕

ビジュアルスタジオを非インタラクティブに使用するために、1つまたは複数の有効なプロジェクトを含むワーキングソリューションで開始すべきである。残念なことに、これは、コマンドライン手法又は自動手法のいずれからでも達成することができないタスクである。いずれの方法もプロジェクト作成のための仕組みを提供しない。しかしながら、ビジュアルスタジオのため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。それゆえ、プロジェクト名およびテンプレートを与えて開始することにより、ビジュアルスタジオのためのソリューション／プロジェクトを作成することができる。

10

【0448】

〔プロジェクトのポピュレーション〕

コマンドラインがサポートしないので、生成されたプロジェクトに新たなファイルを追加するために、ビジュアルスタジオ自動モデルが用いられる。プロジェクトに新たなファイルおよび既存のファイルを追加するために、2つのビジュアルスタジオマクロが提供される。ActiveScriptエンジン（VBScript、JScript、ActivePerl、ActivePythonなどのような）を用いて同じタスクを実行するために、外部スクリプトが類似のコードを用いることができる。それゆえ、本発明のコード生成ツールは新たなファイルを作成し、自動モデルを用いて、それらのファイルを既存のビジュアルスタジオプロジェクトに追加することができる。ファイルが作成された後に、それらのファイルは、必要に応じて、ツールによって更新することができる。

20

【0449】

〔プロジェクトの構築〕

一旦、適当なソリューションおよびプロジェクトが得られたなら、ビジュアルスタジオを非インタラクティブに用いて、Test Planを構築することに対していくつかのオプションがある。最も単純なオプションは、コマンドラインからそれら呼び出すことである。そのようなコマンドラインは以下になるであろう。

【0450】

```
devenv solutionFile /build solutionCfg
```

30

【0451】

ただし、solutionFileはビジュアルスタジオ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することができる特定の構成である。別のソリューションは、自動のためのビジュアルスタジオオブジェクトモデルを用いることである。これにより、そのビルドおよび構成プロセスを、さらに細かく制御できるようになる。上記のように、それは、コマンドラインからプロジェクトを構築するために、Perlスクリプトのリストを含む。このプログラムは、構築するためのプロジェクトおよび構成を指定する構成ファイル（およびそれらのプロジェクトについての他の情報）を読み出し、自動モデルを用いてそれらを全て構築する。1つのスクリプトにおいて自動オブジェクトを用いる方法を例示するために、このスクリプトにおける\$msdevオブジェクトの使用を考える。

40

【0452】

〔デバッグサポート〕

Testクラスの開発者がその作業を検査し、デバックするために、サイトコントローラ内でブレークし、そのコードの中を逐次的に進むことができるようにするデバッグを利用する必要がある。コンパイラによって生成されるコードはMSVC++によってコンパイルされるC++であるので、MSVC++デバッグを用いて、Testクラスインプリメンテーションがデバックされる。この特徴は、C++において直に作業するTestクラス開発者などだけを対象にしていることに留意されたい。生成されたC++コードを直に参照することなくTestプ

50

プログラムの動作をデバッグするか、または逐次的に進めることを望む試験技師に対して他の仕組みが提供されるであろう。

【0453】

〔システムソフトウェア環境〕

このセクションは、テストのための一般的なソフトウェア環境、すなわちユーザ試験計画によって必要とされるファイルのための場所、そのようなファイルのための代替の場所を指定するための仕組み、ならびに試験計画およびモジュール制御ソフトウェアの場所を指定するための方法を説明する。

【0454】

〔試験計画によって必要とされる環境〕

10

システムの標準的な場所、および1つの試験計画によって必要とされる

1. パターンリスト
2. パターン
3. タイミングデータ

4. 試験クラスDLL、のための探索経路のランタイム構成を、環境構成ファイルによって指定されるような、「環境」変数によって構成することができる。これらはテキストファイルであり、以下のような単純な構文を用いる。

【0455】

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

【0456】

20

ネイティブOSによってサポートされる環境変数を用いる代わりに、テキストファイルにおいて、そのような「環境」を定義する利点は、そのインプリメンテーションが後に、OSによってサポートされる環境変数が最大ストリング長を有するなどの共通の制約によって制限されないことである。以下の「環境」（設定）変数は、先に記載されたエンティティのために用いられるであろう。

- ・パターンリスト：Tester_PATLIST_PATH
- ・パターンオブジェクトファイル：Tester_PATOBJ_PATH
- ・パターンソースファイル：Tester_PATSRC_PATH（これはオプションである；以下を参照）

- ・Timingデータファイル：Tester_TIMING_PATH

30

- ・TestクラスDLL：Tester_TEST_CLASS_LIBPATH

【0457】

特殊な事例をサポートするために、有用なデフォルト挙動を保持しながら、3つの構成レベルが提供される。これは、優先順位が高くなる順に説明される。

【0458】

第一に、システム環境設定ファイル、

\$Tester_INSTALLATIONROOT\cfg\setups\Setup.envが、「環境」変数のデフォルト値を指定するであろう。他の構成の仕組みが利用できない場合には、このファイルが必要とされるであろう。一般的に、それは、システム上で実行される全ての試験計画の場合に利用可能であろう。このファイルはインストール中にインストールおよび構成管理（ICM）システムによって作成され、インストーラからの入力に述べられた3つの変数のためのデフォルト値を割り当てる（上記の3つの変数のためのシステムデフォルトに加えて、このファイルは、以下のサブセクションにおいて説明されるような、ある特定の他のテスト「環境」変数のためのシステムデフォルトも含むことに留意されたい）。

40

【0459】

第二に、環境設定ファイルを、試験計画に対するランタイム引き数としてユーザが指定することができる。このランタイム構成の変数は、デフォルト定義よりも高い優先順位を取るであろう。

【0460】

最後に、試験計画が特殊なブロックを用いて、その実行時に用いられることになる環境

50

変数を指定することができる。試験計画において定義される変数は、デフォルトシステムファイルまたはユーザ定義ファイル内の変数よりも高い優先順位を取るであろう。

【0461】

一般的に、全ての必要な変数は、上記の仕組みのうちの1つを通して定義されるべきである。変数が定義されない場合には、ランタイムエラーが生じることになる。

【0462】

〔他の環境設定〕

ユーザ試験計画によって必要とされる「環境」変数に加えて、試験環境によって、以下の2つの「環境」変数が必要とされる。

1. Tester_TEST_PLAN_LIBPATH: これは、ロードされるべきであるユーザ試験計画DLLを見つけるためにシステムコントローラが用いることになる探索経路を指定する。ユーザピン記述およびソケットファイルを見つけるためにも同じ探索経路が用いられることに留意されたい。ICMへのインストール時間中に指定される、この変数のためのデフォルト値は、ICMによって、ファイル\$Tester_INSTALLATION_ROOT\cfg\setups\Setup.envに格納される。

2. Tester_MODULE_LIBPATH: これは、ベンダ提供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ことになる探索経路を指定する。構成管理データベース(CMD)から引き出されるこの情報も、ICMによってファイル\$Tester_INSTALLATION_ROOT\cfg\setups\Setup.envに格納される。

【0463】

ユーザはTester_TEST_PLAN_LIBPATH変数のためのSetup.envファイル内に与えられる値を無効にすることができるが、ユーザがベンダ提供ハードウェアモジュール制御ソフトウェアDLLのための探索経路を明示的に変更することを望まない限り、Tester_MODULE_LIBPATHのためのSetup.envファイル内に与えられる値は変更されるべきでないことに留意されたい。

【0464】

〔探索経路指定の意味〕

探索経路を指定する「環境」変数について以下の点に留意されたい。

1. 各変数は、ある特定のタイプの参照されるファイルを見つけるためにシステムが探索することになるディレクトリ名の、セミコロン(「;」)によって分離されたリストにすべきである。

2. システムがそのような「環境」変数の値を最初に調べた後に、その値に対してユーザによって行われた任意の変更(たとえば、環境構成ファイルを編集することによる)は、それを行うための必要性をユーザが明示的にシステムに「通知する」ときにのみ、システムによって登録されるであろう。

3. テスタが正常に動作する環境のような分散環境内の「カレントワーキングディレクトリ」(CWD)の表記法は、ユーザが直観的に予想するものではないかもしれないので、CWDに対する経路が曖昧な結果に繋がる可能性があるとき、探索経路内の相対経路名は、関連する環境変数(ルートを定義する機能を提供する)の特定の設定に関連するように解釈されるであろう。探索経路内の全ての相対経路名が基づくことになると想定されるルートを示す、この関連する環境変数は、「Tester_INSTALLATION_ROOT」変数であり、それはユーザのシステムへのテストインストールのトップレベル(すなわち「ルート」)ディレクトリの場所を与える。

4. ディレクトリエントリは集合[V:*?<>|;]内の文字を含むことができない。セミコロン(「;」)を除いて、この集合内の全ての他の文字はウィンドウズファイル名において違法であることに留意されたい。セミコロン(「;」)は探索経路内のエントリを区切るために用いられるので、セミコロンは探索経路エントリにおいて用いられるべきでない。経路名には空白を埋め込むことができるが、経路名の直前および直後に生じる(すなわち、経路名内の空白でない最初の文字の前および最後の文字の後の)全ての空白は経路名の一部であるとは見なされず、全て無視されるであろうことに留意されたい。

5. 探索経路ディレクトリは、定義の中に現われた順序に探索されるであろう。最初に現われたファイルが選択されるであろう。

【0465】

[E. 試験パターン]

非常に大きな1組の試験パターンファイルの効率的な管理、取り扱いおよびローディングは、本発明の1つの実施形態のフレームワークのアーキテクチャに関する重要な側面である。階層的なパターンリストの概念は、扱いやすい概念化を提供し、エンドユーザがシステムを使いやすくする際の効率的な手段であると見なされる。

【0466】

DUTへの刺激は、試験ベクトルを通して試験システムが入手できるようになる。ベクトルは一般的に、シーケンシャル（または線形）、スキャンあるいはアルゴリズムパターン発生器（APG）導出として分類することができる。本発明の1つの実施形態のシステムでは、試験ベクトルは、試験時にDUTに加えられるパターンの観点から編成される。パターンは、ユーザの試験プログラム内のパターンオブジェクトによって表される。そのシステムでは、パターンは、パターンリストオブジェクトによってプログラムに基づいて表されるパターンリストとして編成される。パターンリストオブジェクトは、パターンの順序付きリストまたは他のパターンリストを表す。その順序は、リストコンポーネントの宣言の順序で暗示される。ただ1つのパターンしか必要とされない場合には、そのパターンはそれだけでリスト内にカプセル化される必要があることに留意されたい。

【0467】

ユーザの試験プログラム内の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、そのファイルはパターンリストの実際の定義を含む。こうして、パターンリストの内容は、関連するディスクファイルの内容によって動的に判定される（これについては後にさらに説明されるであろう）。

【0468】

パターンリストの定義は、そのパターンリストのための明示的な名前を与え、ファイル名の連想を通して、パターンの順序付きリストおよび／または他のパターンリストを特定する。それは、実行オプションの仕様も提供し、それら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されることができるので、パターンオブジェクトが説明された後にさらに詳細に説明されるであろう。パターンリストは以下の規則に従うべきである。

【0469】

```
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 optionsopt
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 optionsopt
pattern-list-entry :
    PList pattern-list-reference optionsopt
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-parametersopt
option-parameters:
    option-parameter
    option-parameters ',' option-parameter

```

【0470】

以下は、先に用いられた未定義の非終端記号の記述である。

1. version-identifier: 集合 [0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は数字でなければならない。

2. name: 集合 [a-zA-Z_0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-zA-Z_] から選択されなければならない。

3. pattern-list-name: 集合 [a-zA-Z_0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-zA-Z_] から選択されなければならない。

4. file-name: 有効なウィンドウズファイル名（ファイル名内に任意の空白が含まれる場合には、それは二重引用符で囲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持つべきで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を内部参照することができるか、または別のファイル内のパターンリストを外部参照することができる。外部参照はファイル名によって修飾される必要がある。

5. oprion-name: 集合 [a-zA-Z_0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-zA-Z_] から選択されなければならない。

6. option-parameter: 集合 [a-zA-Z_0-9] からの1つまたは複数の文字からなる文字列。

【0471】

パターンリストファイルはコメントをサポートし、コメントはパターンリストファイルパーサによって無視されなければならない。コメントは「#」文字で開始し、その行の終わりまで続く。

【0472】

[E1. パターンリストのための規則]

パターンリストのための静的規則またはコンパイル時の規則は宣言および名前の分解を

規定する。パターンリスト言語内の名前は、global-pattern-list-definitionおよびlocal-pattern-list-definitionによって宣言される。それらの名前は、pattern-list-referenceによって参照される。以下は、これらの宣言および参照を規定するいくつかの規則である。

1. global-pattern-list-definitionまたはlocal-pattern-list-definitionはパターンリストの名前を宣言する。pattern-list-reference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ている。ローカルパターンリストの名前は、それらの宣言されたリストブロック内でのみ知られている。それらの名前は、修飾を用いることなく、そのリストブロック内で直に参照されることができる。さらに深くネストされた宣言では、ローカルパターンリストは、修飾された名前によって参照される必要があるであろう。

10

2. ローカルパターンリスト名は、包含パターンリストの範囲内で知られており、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ている。たとえば、以下を参照されたい。

【0473】

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.

PListL1.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

}

20

30

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の最も外側のレベルにおいて定義されることができるか、または包含パターンリスト内でネストされるように定義されることができる。しかしながら、そのネスティングは単に便宜的なものである。それらは、ファイル内の最も外側のレベルにおいてグローバルパターンリストとして概念的に定義される。ネストされたグローバルパターンリストは、同じ名前の最も外側の（ネストされない）グローバルパターンリストと意味的に同等である。したがって、たとえば、以下を参照されたい。

40

【0474】

GlobalPList G1

{

GlobalPList G2

}

is semantically equivalent to:

50

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

4. 全てのグローバルパターンリストは固有に名前を付けられる。

【0475】

```
GlobalPList G1
{
    # Note that this is as if declared at the outermost level          10
    # 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.                      20
{
    ...
}
```

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も決定する、包含パターンリスト内でネストされた定義を有する。ローカルパターンリストは、その包含パターンリスト内で固有に名前を付けられる。ローカルパターンリストは構文的に、パターンリストファイル内の最も外側のレベルにおいて生じることを許されない。

【0476】

```
GlobalPList G1
{
    LocalPList L
    {
    }

    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
        PListG1.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;
    {
    }
}
```

50

```

    }
}

```

6. 各パターンリストファイルは、1つまたは複数のグローバルパターンリストのための定義を含む。これは構文から直に生じる。最も外側のレベルはglobal-pattern-list-definitionであり、それらのうちの少なくとも1つが存在しなければならない。

7. Pattern-nameは、P a tキーワードに続く、1つのパターンへの参照である。それは、その名前が接尾語、p a tをパターン名に連結することによって得られるパターンファイル内に存在するパターンを参照する。そのファイルは、パターンのために定義された1つの探索経路に沿って得られることになる1つのファイルを示す。

8. pattern-list-referenceは、PListキーワードに続くパターンリストへの参照である。その参照は、オプションのファイル名、およびそれに続く、修飾されたパターンリスト名からなり、そのパターンリスト名は単に、ドットによって分離される名前のリストである。したがって、たとえば、以下は、ファイルfoo.plist内にあるグローバルパターンリストG 1内にネストされるL 1内にネストされるL 2内にネストされるローカルパターンリストL 3を参照するpattern-list-referenceとして用いることができる。その名前の最も左側の名前部分はG 1である。

【0 4 7 7】

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

【0 4 7 8】

最も左側の名前部分は、グローバルパターンリストに分解しなければならないか、または参照点から見ることができるローカルパターンリストに分解しなければならない。

【0 4 7 9】

pattern-list-referenceの名前分解は以下のように進められる。

1. 各名前部分は、その前にある接頭部に照らして宣言される名前に分解する。
2. ファイル修飾がある場合には、最も左側の名前部分は、名前を付けられたファイルにおいて宣言されるグローバルパターンリストに分解する。

3. ファイル修飾がない場合には、最も左側の名前は、包含範囲内のローカルパターンリストに分解することができ、それが失敗する場合には、次の包含範囲内のローカルパターンリストに分解することができ、それが包含グローバル範囲まで続けられる。

4. グローバル範囲がパターンリストファイル内の最も外側のレベルにおいて宣言されたかのように、グローバル範囲の意味を保持するために、最も近くにある包含グローバル範囲への範囲の探索を制限する必要がある。ネストされたグローバル範囲が最も外側のレベルにおいて（同等に）テキスト形式で宣言された場合には、名前分解探索は、その範囲を検査した後を終了するであろう。

5. その参照が以前のステップによって分解されていなかった場合には、最も左側の名前部分が、この同じファイル内のグローバルパターンリストに分解されることができる。

6. その参照が以前のステップによって分解されていなかった場合には、最も左側の名前部分が、.plist接尾部を最も左側の名前部分に追加することにより、そのファイル内で名前を付けられるグローバルパターンリストに分解されることができる。

7. その参照が以前のステップによって分解されていなかった場合には、その参照はエラー状態である。

【0 4 8 0】

先に述べられたように、上記の規則は、最も左側の名前部分が、参照点から見ることができるローカルパターンリスト、またはグローバルパターンリストのいずれかに分解することを指示する。

【0 4 8 1】

以下の例はこれらの概念のいくつかを例示する。

【0 4 8 2】

```
GlobalPlist G1
{

```

10

20

30

40

50


```

    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;
}

```

10

【0483】

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、それらを用いる試験計画の中で固有であることが要求される。

【0484】

パターンリスト参照は、同じファイル内の参照の前後いずれかにおいて定義されるパターンリストを参照することができる。

【0485】

再帰的および相互に再帰的なパターンリスト定義は許されない。ユーザがそのような定義を作成するのを防ぐために、パターンリストファイル構文内には何も存在しないが、パーサがそのような条件を検出するとき、パーサはエラーフラグを立てるであろう。そのような条件を検出することに関連して或るコストがかかることに留意されたい。ユーザは、入力空間が相互に再帰的な定義を免れることを保証する責任を負うことができるか否かの検査のスイッチを切ることができるであろう。

20

【0486】

```

GlobalPList G1
{
    LocalPList L2
    {
        LocalPList L3
        {

```

30

```

            # Error. L2 runs L3 which runs L2.
            # This is a recursive reference to L2
            PList L2;
            PList G2;
        }
    }
}

```

```

GlobalPList G2
{

```

40

```

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

```

【0487】

パターンおよびパターンリストの構文的な記述は、それらの記述においてオプションを指定できるようにする。一般的に、オプションはベンダ特有である。構文は、任意のパターンまたはパターンリストが、それぞれ複数のパラメータで指定される複数のオプションを有することができるようにする。大部分のベンダによって認識されることになるいくつかのサポートされるオプションが記述される。

50

【0488】

パターン実行シーケンスを定義した後に、パターン木の動的な（すなわち実行時の）意味が記述される。

【0489】

[E2. パターン]

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。パターンのユーザ定義の内容は、プレーンテキストファイルであるパターンソースファイル606において入手することができる。パターンコンパイラは、ソースファイルを、テストハードウェア上にロードするの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責任を担うであろう。この後者のファイルはパターンオブジェクトファイルと呼ばれるであろう。以下はその一般的な属性である。

1. パターンオブジェクトは、ユーザが作成することはできない。むしろ、ユーザは、他のパターンリストおよび／またはパターンのコレクションであるパターンリストを常に取り扱う。パターンリストオブジェクトは、必要に応じてユーザがアクセスできるようにしながら、その中に含まれるパターンオブジェクトを作成し、所有し、保持する。

2. パターンは1つの試験計画内で固有に名前を付けられる。すなわち、その試験計画内では、同じ名前を有する2つのパターンは存在することはできない。パターンの名前は、それを含むファイルの名前とは異なる。パターンファイル名は、1つの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、一方、その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。

【0490】

本発明の一実施形態では、一般的に、ただ1つのDUT（被試験デバイス）が、種々のベンダからのテストモジュールに接続されることもある。これは、全パターンのコンパイル-ロード-実行連鎖にとって複数の意味を有する。主なものがこのセクションにおいて説明される。

【0491】

[E3. パターンコンパイル]

こうして、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照らして）特定のサイト構成をターゲットにする必要がある。この説明の残りの部分において、用語「モジュール」は、一例として、デジタルモジュールを指すために用いられるであろう。種々のベンダからのモジュール608をシステムに組み込むことができるようにするために、以下の手順が好ましい。

1. 各モジュールベンダは、動的にロード可能なライブラリまたは個別の実行可能プログラムの形で、自ら所有するモジュール特有のパターンコンパイラ610を提供する責任を担うであろう。このコンパイラライブラリ／実行可能プログラムは、最低でも、compile()メソッドを提供するであろう。

2. パターンソースファイルは、それが含むパターンブロックのタイプ毎に2つの異なるタイプのセクションを収容するであろう。

a. 全てのコンパイラがアクセス可能である情報を含む「共通」セクション（ただし、必ずしも用いられる必要はない）。

b. 固有ベンダコードによってそれぞれ特定され、特定のベンダコンパイラによって使用可能な情報のための1つまたは複数のオプションのベンダ特有セクション。

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に作成しないであろう。代わりに、テストは、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「metafile」612を提供するであろう。パターンコンパイラは、システムコントローラとして動作するコンピュータ上に、またはオフラインで、たと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することができる。これまで抽象語において参照された「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ることになり、ソースファイル拡張

子がオブジェクトファイル拡張子によって置き換えられる。OFMは、このファイルの読出しおよび書込みを行うためのアプリケーションプログラミングインターフェース（API）を提供するであろう。オブジェクトメタファイルは、以下のものを格納するための手段を有するであろう。

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のため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

c. モジュールベンダの要求に応じて編成され、モジュールベンダによって解釈されることができる、モジュール特有のパターンデータ

【0492】

OFM APIによって、モジュールベンダのコンパイラは、モジュール特有の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ができるようになるであろう。オブジェクトメタファイルのこのレイアウトによって、ターゲットにされたサイトにおいて2つ以上のモジュールが同じ場合であっても、パターンデータがモジュール毎に編成されるようになることに留意されたい。

【0493】

直接メモリアクセス（DMA）のような効率的なデータ通信を利用することができる、モジュール特有の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によって、さらに多くのベンダ供給構成情報が必要とされる場合もあることに留意されたい。

【0494】

[E4. オブジェクトファイルマネージャ（OFM）フレームワーク]

オープンアーキテクチャ試験システムは、拡張性があり、かつ自由度の高いソリューションを提供する。本発明の一実施形態のオープンアーキテクチャ試験システムでは、単一の被試験デバイス（DUT）を種々のベンダからのテストモジュールに接続することができる。この文脈における試験モジュールは、オープンアーキテクチャ試験システム内の1つの機能ユニットを指しており、ハードウェアおよびソフトウェア両方のコンポーネントを含むことができる。この実施形態の分散形で、構成変更可能なモジュール式オープンアーキテクチャ試験システムは、本明細書では、試験システムまたはシステムとも呼ばれる。

【0495】

このパターンを種々のハードウェアベンダによって開発された多数のテストモジュールとコンパイルできるようにするための知識をできる限り用いることなく、試験システムのユーザは、DUTを試験するためのパターンを書く。これは、これらの各モジュールがそのパターンにおいて指定される機能を提供することができること、およびベンダに特有のパターンフォーマットがそのパターンファイル内に含まれることができることを前提とする。

【0496】

DUTへの刺激は、試験ベクトルを通して試験システムが入手できるようになる。試験システムフレームワーク内の試験ベクトルは、試験中のDUTに加えられるパターンに照らして編成される。パターンはユーザの試験プログラム内のパターンオブジェクトによって表される。以下の段落は、試験システムのOFMフレームワークの重要な用語のうちのいくつかを導入する。

【0497】

パターンブロックオブジェクトは、所与のタイプのパターンデータ（サイクライズまたは非サイクライズ）のコレクションを表しており、パターンブロック内のパターンデータを記述するための共通の構文を共有する。一実施形態では、サポートされるパターンブロックのタイプは、パターンソース内のタイプ（たとえば、pattern[type]）によって特定される、サイクライズパターンブロック（たとえば、MainPattern、SubrPattern、algorithmic pattern generator pattern(ALPG Pattern)、および非サイクライズパターンタイ

10

20

30

40

50

プである。SubrパターンブロックおよびALPG Patternブロックは、補助パターンオブジェクトメタファイルの例示的なブロックである。パターンブロックは以下の2つの異なるタイプのセクションを収容することができる：1) 全てのコンパイラがアクセス可能な（ただし、必ずしも用いられない）情報を含む共通セクション、および2) ベンダコンパイラが利用できる情報を格納するための、それぞれが固有のベンダコードによって特定される1つまたは複数のオプションのベンダ特有のセクション。パターンオブジェクトは、パターンオブジェクト内の2つのブロックが同じタイプから成らないという制約を有する、パターンブロックのコレクションを表す。サイクライズパターンブロックは、シーケンシャルに繰り返されるデジタルテストにおいて用いられる試験ベクトルのコレクションを表しており、それは、ベンダが従うべき標準的な構文を提供する。パターンソースファイルは、1つまたは複数のパターンブロックを収容することができる。

10

【0498】

図13は、本発明の一実施形態によるモジュール式試験システムにおいてパターンオブジェクトファイルを管理するためのOFMフレームワークを示す。OFMフレームワークは、ベンダコンパイラインターフェース (IVendorCompiler) 1302と、ベンダパターンコンパイラインターフェース (IVendorPatCompiler) 1303と、ベンダサイクライズパターンコンパイラインターフェース (IVendorCyclizedPatCompiler) 1304とを備える。モジュールベンダは、自ら所有するパターンコンパイラを提供する責任があり、そのパターンコンパイラは、IVendorPatCompilerインターフェース1303またはIVendorCyclizedPatCompilerインターフェース1304、あるいはその両方をインプリメントする。OFMフレームワークはさらに、OFMモジュールエージェント (OFModuleAgent) クラス1306と、OFMサイクライズパターンモジュールエージェント (POFCyclizedModuleAgent) クラス1308と、OFMラベルリーダー (OFMLabelReader) クラス1310と、オブジェクトファイルマネージャ (OFMManager) クラス1312と、パターンオブジェクトメタファイル (PatternObjMetaFile) クラス1314とを備える。非サイクライズパターンブロックを取り扱うための第1のベンダパターンコンパイラ (ATRFPatCom) クラス1316、およびサイクライズパターンブロックを取り扱うための第2のベンダパターンコンパイラ (ATDM250PatCom) クラス1318はそれぞれ、IVendorPatCompilerインターフェース1303およびIVendorCyclizedPatCompilerインターフェース1304をインプリメントするベンダ提供のクラスの例である。それらのインターフェースおよびクラスの記述が以下に提供される。

20

30

【0499】

[IVendorPatCompilerインターフェース]

IVendorPatCompilerインターフェース1303は、2つの関数グループをサポートする。第1の関数グループは、パターンソースファイル内の非サイクライズパターンブロックのコンパイルと、モジュールデータによるパターンオブジェクトメタファイルの更新とをサポートする。第2の関数グループは、パターンオブジェクトメタファイルの共通ブロックセクションの更新をサポートする。

【0500】

ベンダコンパイラがモジュール構成ファイルにおいてセットされるモジュールタイプIDと一致し、そのブロックタイプをサポートする場合には、OFMはパターンコンパイルメソッドのグループを呼び出す。パターンコンパイルメソッドが呼び出され、ベンダコンパイラが、コンパイルされたパターンオブジェクトデータを、対応するモジュールパターンローダのためのメタファイルに格納する。パターンコンパイルメソッドの第1のグループが以下に説明される。

40

・compile(): compile()メソッドは、パターンソースファイルをコンパイルするためにOFMによって呼び出される。OFMはPOFCyclizedModuleAgentオブジェクトをベンダコンパイラに渡す。このオブジェクトは、保護/アクセス制御を提供し、ベンダコンパイラがパターンソースファイルの共通セクションおよびモジュール特有セクションへの読出しアクセス、並びにベンダモジュール特有セクションへの書込みアクセスを共に許可される

50

ようにする。ピンと資源との間のマッピングについての情報を提供するために、OFMResourcePinMapperインスタンスも引き数としてベンダコンパイラに渡される。

- ・needsPinNames():needsPinNames()メソッドは、compile()メソッドが呼び出されるときに、モジュールコンパイラが提供されることになるピン名のリストを必要とするか否かを問い合わせるために、OFMによって呼び出される。

- ・getSupportedResources():getSupportedResources()メソッドは、それがサポートする資源の名前についてモジュールコンパイラに問い合わせるために、OFMによって呼び出される。これは、ベンダコンパイラが資源レベル毎にパターンコンパイルプロセスのためのピンリストを必要とするときに、ピンのリストを作成するために用いられる。

- ・release():release()メソッドは、所望のIVendorPatCompilerオブジェクトが削除される前に、ベンダパターンコンパイラがクリーンアップ動作を実行できるようにするために、OFMによって呼び出される。

- ・getErrorList():getErrorList()メソッドは、OFCStatusオブジェクトの配列を検索するために、OFMによって呼び出される。この配列は、パターンコンパイラがパターンソースのコンパイル中に出会う場合があるエラーおよび警告の両方を含む。OFCStatusは、エラーについての情報を含むエンティティである。それには、特定のエラーコードを表す簡単な形の整数値を用いることができる。OFCStatusを、エラーが生じた場合のエラーコード、およびエラーの特定のインスタンスに関連するいくつかの特有のデータを含む情報を有するオブジェクトとすることが有用である。

【0501】

OFMは、IVendorPatCompilerインターフェース1303の「getter」メソッドを用いて、ベンダコンパイラに共通セクションの種々の設定を問い合わせる。コンパイルされるパターンソースファイルからの共通セクションの設定に関する詳細を提供することは、ベンダコンパイラの責任である。パターンソースファイルをコンパイルするために用いられる共通セクションメソッドが以下に説明される。

- ・getPinRef():オブジェクトファイルマネージャがベンダパターンコンパイラにパターンソースファイルをパースし、OFMにピン参照設定ファイルの名前を提供することを要求する場合には、オブジェクトファイルマネージャはgetPinRef()メソッドを呼び出す。

- ・getSubReferences():オブジェクトファイルマネージャがベンダパターンコンパイラにパターンソースファイルをパースし、OFMにこのパターンソースファイルによって参照されるSubroutineの配列を提供することを要求する場合には、オブジェクトファイルマネージャはgetSubReferences()メソッドを呼び出す。

- ・getSynchronizedNameArray():OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMにこのパターンソースファイル内に存在する同期したブロックの配列を提供することを要求する場合には、OFMはgetSynchronizedNameArray()メソッドを呼び出す。OFMフレームワークはこれらの名前（ピン記述ファイルにおいて宣言される）を用いて、これらのブロックの実行を調整する。

【0502】

[IVendorCyclizedPatCompilerインターフェース]

IVendorCyclizedPatCompilerインターフェース1304は、IVendorCompilerインターフェース1302から継承する。またそれは、モジュールベンダに2つの関数グループをサポートすることを要求する。第1の関数グループは、パターンソースファイル内のサイクリズパターンブロックのコンパイルをサポートし、パターンオブジェクトメタファイルを更新する。第2の関数グループは、パターンオブジェクトメタファイルの共通ブロックセクションの更新をサポートする。

【0503】

ベンダコンパイラがモジュール構成ファイルにおいてセットされるモジュールタイプIDと一致し、そのブロックタイプをサポートする場合には、OFMはパターンコンパイルメソッドを呼び出す。パターンコンパイルメソッドが呼び出され、ベンダコンパイラが、

コンパイルされたパターンオブジェクトデータを、対応するモジュールパターンローダのためのメタファイルに格納する。パターンコンパイルメソッドの第1のグループが以下に説明される。

【0504】

compile()メソッドは、パターンソースブロックをコンパイルするためにOFMによって呼び出される。OFMはPOFCyclizedModuleAgentオブジェクトをベンダコンパイラに渡す。このオブジェクトは、保護／アクセス制御を提供し、ベンダコンパイラが共通セクションおよびモジュールセクションへの読出しアクセス、並びにモジュールセクションへの書込みアクセスを許可されるようにする。ピンと資源との間のマッピングについての情報を提供するために、OFMResourcePinMapperインスタンスも引き数としてベンダコンパイラに渡される。

10

【0505】

OFMは、IVendorCyclizedPatCompilerインターフェース1304の「getter」メソッドを用いて、ベンダコンパイラに共通セクションの種々の（サイクライズテスト特有の）設定を問い合わせる。コンパイルされるパターンソースファイルからの詳細を提供することは、ベンダコンパイラの実務である。パターンソースファイルをコンパイルするために用いられる共通セクションメソッドが以下に説明される。

- ・getALPGReferences(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMにパターンソースファイル内で参照されるALPGパターンのリストを提供することを要求する場合には、OFMはgetALPGReferences()メソッドを呼び出す。

20

- ・getDomainList(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMにパターンによって用いられるDomainのリストを提供することを要求する場合には、OFMはgetDomainList()メソッドを呼び出す。1つのパターンがデバイスの全てのドメインのために用いられない場合もあるので、このドメインのリストには、ピン記述ファイルにおいて指定されるドメインのリストのサブセットを用いることができることに留意されたい。

- ・getLabelList(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMに所与のドメインのためのパターンソースにおいて用いられるLabelのリストを提供することを要求する場合には、OFMはgetLabelList()メソッドを呼び出す。ラベルは1つのドメインにおいて或る範囲を有し、同じラベルを多数のドメインにおいて用いることができることに留意されたい。

30

- ・getLabelOpcodeOffset(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMに所与のドメインのラベルのためのOpcodeオフセットを提供することを要求する場合には、OFMはgetLabelOpcodeOffset()メソッドを呼び出す。

- ・getLabelOperandOffset(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMに所与のドメインのラベルのためのOperandオフセットを提供することを要求する場合には、OFMはgetLabelOperandOffset()メソッドを呼び出す。

- ・getPXRSetupFileRef(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMに、インポートされるPattern Cross Reference Setupファイルの名前を提供することを要求する場合には、OFMはgetPXRSetupFileRef()メソッドを呼び出す。

40

- ・getCyclizedPatternType(): OFMがベンダパターンコンパイラにパターンソースファイルをパースし、OFMに指定されるサイクライズパターンブロックのタイプ（SQPG、ALPG、SUBR、SCAN、ESCAN）を提供することを要求する場合には、OFMはgetCyclizedPatternType()メソッドを呼び出す。

- ・getAMAX(): OFMがベンダパターンコンパイラにソースファイルをパースし、OFMに、共通セクションに属する最大アドレスサイズデータを提供することを要求する場合には、OFMはgetAMAX()メソッドを呼び出す。

【0506】

50

[OFModuleAgent]

OFModuleAgent 1 3 0 6 は、POFCyclizedModuleAgent クラスによって拡張することができるベースクラスである。このクラスは、各モジュールのバイナリオペークデータにアクセスするために必要とされるメソッドグループのインプリメンテーションを提供する。OFModuleAgent クラス 1 3 0 6 は以下のメソッドを含む。

- ・ flush() : バッファされたバイナリデータがフラッシュされる必要がある場合には、ベンダコンパイラは flush() メソッドを呼び出す。
- ・ position() : 1 つのパターンブロック内のバイナリデータの現在位置を判定するために、ベンダコンパイラは position() メソッドを呼び出す。
- ・ preAllocate() : ベンダコンパイラが OFM に、効率を改善するために所与のサイズの pattern - object - metafile 内の 1 つのブロックを予め割り当てることを要求する場合には、ベンダコンパイラは preAllocate() メソッドを呼び出す。バイナリブロックのために割り当てられるデフォルトサイズは 0 x 4 0 0 0 である。
- ・ getSize() : ベンダコンパイラが OFM に、モジュールのためのバイナリデータのサイズを判定するためにメタファイルに問い合わせることを要求する場合には、ベンダコンパイラは getSize() メソッドを呼び出す。
- ・ read() : ベンダコンパイラが OFM に、所与のサイズのデータを、そのモジュールのためのメタファイルからの所与のオフセットにおいて読み出すことを要求する場合には、ベンダコンパイラは read() メソッドを呼び出す。オフセットおよびサイズはそのモジュールのためのブロックに対するものであり、全メタファイルに対するものではない。
- ・ verify() : ベンダコンパイラが OFM に、所与のサイズの有効データが、そのモジュールのためのこのメタファイルの所与のオフセットにおいて存在することを検査することを要求する場合には、ベンダコンパイラは verify() メソッドを呼び出す。オフセットおよびサイズはそのモジュールのためのブロックに対するものであり、全メタファイルに対するものではない。
- ・ write() : ベンダコンパイラが OFM に、所与のサイズのデータを、所与のオフセットにおいてそのモジュールのためのメタファイルに書き込むことを要求する場合には、ベンダコンパイラは write() メソッドを呼び出す。オフセットおよびサイズはそのモジュールのためのブロックに対するものであり、全メタファイルに対するものではない。
- ・ getSocketRef() : ベンダコンパイラが OFM に、タイプ SocketInfo のオブジェクトを作成することを要求する場合には、ベンダコンパイラは getSocketRef() メソッドを呼び出す。このクラスは、ファイルをパースすることを必要とすることなく、コンパイラがシステムソケットファイルの設定にアクセスできるようにする。
- ・ getSourceRef() : ベンダコンパイラが OFM に、パターンソースファイルの名前を提供することを要求する場合には、ベンダコンパイラは getSourceRef() メソッドを呼び出す。
- ・ getSourceVersion() : ベンダコンパイラが OFM に、パターンソースファイル言語バージョンを表すバージョンストリングを供給することを要求する場合には、ベンダコンパイラは getSourceVersion() メソッドを呼び出す。

【 0 5 0 7 】

[POFCyclizedModuleAgent]

POFCyclizedModuleAgent クラス 1 3 0 8 は、OFM が或るパターンをコンパイルし始めるときに、OFM によって開かれる PatternObjMetaFile クラス 1 3 1 4 へのシャローハンドルである。それは、サイクライズテストモジュールに特有の付加的な方法で OFModuleAgent を拡張する。それは、ベンダパターンコンパイラが共通セクションおよびモジュールセクションの両方から読出しを行うことができるようにすることにより、ベンダパターンコンパイラがパターンオブジェクトメタファイルにアクセスするのを制限する。また、それはベンダコンパイラのモジュールセクションへの書き込みアクセスも提供する。このインターフェースは、OFM が compile() メソッドを呼び出すときに、パターンのコンパイル中にベンダパターンコンパイラによって用いられる。POFCyclizedModuleAgent クラス 1 3

0 8 は以下のメソッドを含む。

・getALPGRefList(): ベンダコンパイラが O F M に、このパターンソース内で参照される A L P G パターンのアレイを提供することを要求する場合には、ベンダコンパイラは getALPGRefList() メソッドを呼び出す。

・getDomainList(): ベンダコンパイラが O F M に、このパターンによって用いられるドメインのアレイを提供することを要求する場合には、ベンダコンパイラは getDomainList() メソッドを呼び出す。1 つのパターンが 1 つのデバイスの全てのドメインのために用いられない場合もあるので、このドメインのアレイは、ピン記述ファイルにおいて指定されるドメインのアレイのサブセットの場合もあることに留意されたい。

・getLabelList(): ベンダコンパイラが O F M に、所与のドメインのためのパターンソースにおいて用いられる Label のアレイを提供することを要求する場合には、ベンダコンパイラは getLabelList() メソッドを呼び出す。ラベルはドメイン内の或る範囲を有し、同じラベルを多数のドメインにおいて用いることができることに留意されたい。

・getLabelOpcodeOffset(): ベンダコンパイラが O F M に、所与のドメイン内のラベルのための Opcode オフセットを提供することを要求する場合には、ベンダコンパイラは getLabelOpcodeOffset() メソッドを呼び出す。

・getLabelOperandOffset(): ベンダコンパイラが O F M に、所与のドメイン内のラベルのための Operand オフセットを提供することを要求する場合には、ベンダコンパイラは getLabelOperandOffset() メソッドを呼び出す。

・getPXRSetupFileRef(): ベンダコンパイラが O F M に、インポートされる Pattern Cross Reference Setup ファイルの名前を提供することを要求する場合には、ベンダコンパイラは getPXRSetupFileRef() メソッドを呼び出す。

・getTimingRef(): ベンダコンパイラが O F M に、ITimFile インターフェースをインプリメントするオブジェクトを提供することを要求する場合には、ベンダコンパイラは getTimingRef() メソッドを呼び出す。このクラスは、このファイルをパースすることを必要とすることなく、コンパイラがシステム試験プログラム言語タイミングファイルの設定にアクセスできるようにする。

・getTimingMapRef(): ベンダコンパイラが O F M に、ITMapFile インターフェースをインプリメントするオブジェクトを提供することを要求する場合には、ベンダコンパイラは getTimingMapRef() メソッドを呼び出す。このクラスは、このファイルをパースすること

を必要とすることなく、コンパイラが試験システムタイミングマップファイルの設定にアクセスできるようにする。

・getCyclizedPatternType(): ベンダコンパイラが O F M に、指定されるパターンのタイプ (SQPG、ALPG、SUBR、SCAN、ESCAN) を提供することを要求する場合には、ベンダコンパイラは getCyclizedPatternType() メソッドを呼び出す。

・getTMAPResolutionChecksFlag(): ベンダコンパイラは、試験システム変数 TMAP_RESOLUTION_CHECKS がシステム環境構成ファイル内でセットされるか否かを問い合わせるために、getTMAPResolutionChecksFlag() メソッドを呼び出す。

【0 5 0 8】

[OFMLabelReader]

ベンダパターンコンパイラは、サイクライズパターンブロックをコンパイルするときに、サブルーチンパターンオブジェクトメタファイルおよび A L P G パターンオブジェクトメタファイルのラベルおよびオフセットにアクセスする必要がある。このアクセスは、OFMLabelReader クラス 1 3 1 0 によって提供される。このクラスは、POFCyclizedModuleAgent クラスと同じように機能し、ドメイン毎にメタファイルのサブルーチンおよび ALPG ラベル／オフセットデータへの読み出し専用アクセスを提供する。ベンダコンパイラは、参照されるサブルーチン／ALPG パターン毎にこのオブジェクトをインスタンス化し、そのタスクが完了するときに、そのオブジェクトを削除することができる。この実施形態では、これは、サイクライズデジタルモジュールパターンコンパイラに特有である。OFMLabelReader クラス 1 3 1 0 は以下のメソッドを含む。

・OFMLabelReader()：ベンダコンパイラは、指定されたサブルーチンまたはALPGパターンオブジェクトメタファイルのためのOFMLabelReaderオブジェクトを構成するために、OFMLabelReader()メソッドを呼び出す。

・open()：ベンダコンパイラは、OFMLabelReaderオブジェクトがファイルを開くために、open()メソッドを呼び出す。

・close()：ベンダコンパイラは、OFMLabelReaderオブジェクトがファイルを閉じるために、close()メソッドを呼び出す。

・getLabelsList()：ベンダコンパイラがOFMに、OFMLabelReaderインスタンスの所与のドメインのためのラベルの配列をゲットすることを要求する場合には、ベンダコンパイラはgetLabelsList()メソッドを呼び出す。

・getLabelOpcodeOffset()：ベンダコンパイラがOFMに、OFMLabelReaderインスタンスの所与のドメイン内のラベルのためのOpcodeオフセットを読み出すことを要求する場合には、ベンダコンパイラはgetLabelOpcodeOffset()メソッドを呼び出す。

・getLabelOperandOffset()：ベンダコンパイラがOFMに、OFMLabelReaderインスタンスの所与のドメイン内のラベルのためのOperandオフセットを読み出すことを要求する場合には、ベンダコンパイラはgetLabelOperandOffset()メソッドを呼び出す。

・getSourceVersion()：ベンダコンパイラがOFMに、このOFMLabelReaderインスタンスによって表されるサブルーチンパターンオブジェクトファイルを生成するために用いられるソースファイルのVersionを読み出すことを要求する場合には、ベンダコンパイラはgetSourceVersion()メソッドを呼び出す。

【0509】

一実施形態では、パターンブロックは、Pattern Blockのタイプをカプセル化する最も上位のオブジェクトである。1つのパターンソースファイル内に、パターンブロックのうちの1つまたは複数が存在することができる。各パターンブロックは、ただ1つのCommon Sectionおよび0または1つ以上のVendorSection（モジュールセクションとも呼ばれる）を含む。OFMフレームワークは、以下のタイプのパターンブロックをサポートする：MainPattern、SubrPattern、ALPGPatternおよびPattern。MainPattern、SubrPatternおよびALPGPatternは、サイクライズパターンを取り扱うための特有のブロックである。汎用のパターンタイプは、OFMフレームワークが他のテストパターン言語をトランスペアレントに使用できるように拡張可能である。

【0510】

以下はパターンソースファイルの一例である。それは、サイクライズデジタルモジュールおよび仮想汎用パターン（タイプF00）のためのMainPatternを含む。同じソースファイル内にサイクライズパターンおよびF00タイプパターンを入れる理由は、サイクライズMainPatternブロックおよびF00パターンブロックが、共にただ1つのエンティティとして用いることができるほど十分に密接に関連しているためであることに留意されたい。この例では、Pattern F00のCommonSection内の構文は、試験システム内のタイプF00の全てのパターンために標準化されている構文であるものと仮定される。標準的な構文が存在しない場合には、CommonSectionは空であってもよい。

【0511】

```
#
# Filename : main1. pat (produces object file main1.pobj)
#
Version 1.0 ;
#
```

```
-----
# Main Pattern definition:
#
-----
```

10

20

30

40

50

```

-----
MainPattern
{
    CommonSection
    {
        $ define    ALL_H    (H*)
        $ define    ALL_L    (L*)
        $ define    ALL_0    (0*)
        $ define    ALL_1    (1*)
        # ----- 10
    }
}
-----
# Timing Specifications
# -----
-----
Timing          "productionTiming.tim:ProductionTiming";

# -----
-----
# Pin Description file Specifications
# ----- 20
-----
PinDescription   "Pin.pin";

# -----
-----
# Setup file Specifications
# -----
-----
Pxr   "setup.pxr";

# ----- 30
-----
# Default Domain Cycles
# -----
-----
Domain default
{
    # -----
}
-----
# label:instruction {Vector/Waveform Data}
# ----- 40
-----
NOP      { V { DIR=1; APins=$ (ALL_1) ;
            BPins=$ (ALL_H); OE_=0; } W {AllPins=wfs1; } }
NOP      { V { APins=${ALL_0}; BPins=${ALL_L}; } }
NOP      { V { APins=10000000; BPins=HLLLLLLL; } }
NOP      { V { APins=${ALL_1}; BPins=${ALL_H}; } }
EXIT     { V { APins=${ALL_0}; BPins=${ALL_L}; } }
}
}
----- 50

```

```

}
# -----
-----
# F00 Pattern definition;
# -----
-----
Pattern F00
{
    CommonSection
    {
        NumberOfData = 192;
        DataFormat
        {
            float, int;
        }
        Data
        {
            0.0E+00, 1;
            5.21E+00, 0;
            . . . . ;
        }
    }
    VendorSection 1
    {
        Compression = ENABLED;
    }
}

```

【0 5 1 2】

同じタイプの多数のインスタンスは許されない。これは、MainPattern、SubrPatternまたはALPGPatternブロックのうちの1つだけが同じサイクライズパターンタイプとして取り扱われることを意味する。以下の例は、同じタイプ（サイクライズ）の2つのパターンブロックを含むときにエラーを生成する。

【0 5 1 3】

```

#
# -----
-----
# File twoCyclizedTypes.pat
#
# -----
-----

```

Version 1.0;

```

#
# This is the main cyclized pattern block
#
MainPattern
{
    CommonSection
    {
        . . .
    }
}

```

```

}
#
# This is the subroutine cyclized pattern block
#
SubrPattern
{
    CommonSection
    {
        . . .
    }
}

```

10

【0 5 1 4】

パターンの場合、宣言されたタイプのただ1つのインスタンスが許される。他の異なる、宣言されたタイプの場合、パターンの多数のインスタンスが許される場合もある。以下の例は、PatternF001およびPatternF002の2つのインスタンス、ならびにタイプMainPatternのPattern Blockインスタンスを含む。

【0 5 1 5】

```

#

```

```

-----
-----
# File threeTypes.pat
#
-----
-----

```

20

```

Version 1.0;

```

```

#
# This is the main cyclized pattern block
#

```

30

```

MainPattern

```

```

{
    CommonSection
    {
        . . .
    }
}

```

```

# This is ,a pattern block of generic type F001
#

```

```

Pattern F001

```

40

```

{
    CommonSection
    {
        . . .
    }
}

```

```

#
# This is a pattern block of generic type F002
#

```

```

Pattern F002

```

50

```

{
    CommonSection
    {
        . . .
    }
}

```

【0516】

CommonSectionブロックは、試験システムのために標準化されるパターンタイプの記述を含む。このセクションの内容によって、この標準的な構文が、種々のコンパイラによって同じように解釈されるようになる。そのような標準的な構文の一例は、CyclizedデジタルモジュールのためのMainPatternのために定義される構文である。試験システム内のこのタイプのための標準的な構文が存在しない場合には、空のCommonSectionが定義される。

10

【0517】

OFMフレームワークによって、各モジュールコンパイラが、各Patternブロック内の対応するモジュールセクションの内容を判定できるようになる。他のベンダコンパイラは、ベンダ特有のPatternブロックにおいて確立される内容にアクセスすることを許されない。

【0518】

[パターンオブジェクトメタファイル]

20

要約情報および他の共通の構成情報がパターンオブジェクトメタファイルのメイン共通ヘッダに格納される。その要約は、パターンオブジェクト内の種々のパターンブロックにアクセスするために必要な情報からなる。以下の情報がメイン共通ヘッダに格納される。

1. パターンソースファイル名。
2. ソースファイルからのバージョン情報。
3. パターンコンパイル中に用いられることになる、Socketファイル、Utility Configuration FileおよびModule Configuration Fileへの参照。
4. パターンオブジェクトメタファイル内に収容されるパターンブロックのタイプのコレクション。

【0519】

30

各パターンブロックはヘッダセクションを有し、ヘッダセクションは、アドレスのプリロード分解などを処理するため、またはデータロギングを処理するために必要とされる情報を含む。各パターンブロックの共通セクションの意味は、特定のブロックタイプの場合と概ね同じであるので、各コンパイラは同じ要約情報を提供することができる。こうして、メタファイルを書く第1のコンパイラがこの情報を格納する。以下の情報は、非サイクライズブロックの共通セクションヘッダに格納される。

1. ソースファイルにおいて宣言されるようなパターンのタイプ。
2. パターンソースファイルの共通セクション内のサブルーチン参照のマップ。
3. このタイプのパターンブロックのために必要とされる場合には、Pin設定ファイルへの参照。
4. このパターンにおいて定義される同期したブロックのマップ。これは、システムピン記述ファイル内のkeyword Synchronizedで定義される名前に対応する。

40

【0520】

以下の情報は、サイクライズブロックの共通セクションヘッダに格納される。

1. ソースファイルにおいて宣言されるようなパターンのタイプ。
2. パターンソースファイルの共通セクション内のサブルーチン参照のマップ。
3. このパターンにおいて定義されるドメインブロックのマップ。これは、システムピン記述ファイル内のkeyword Domainで定義される名前に対応する。これらの名前は、非サイクライズパターンのためのSynchronizedブロックと同じようにしてOFMフレームワークによって処理される。

50

4. パターンソースファイルの共通セクションにおいて用いられる波形およびタイミング名のリスト。

5. パターンソースファイルの共通セクション内のベクトルオペコードおよびオペランドアドレスへのラベル参照のマップ。

6. パターンソースファイルにおいて指定されるPin、Timing、Timing Mapおよび他の設定への参照。

【0521】

図14は、パターンオブジェクトメタファイル (PatternObjMetaFile) とそのサポート用クラスとの間のインタラクションを示す。PatternObjMetaFileクラス1402は、サイクライズパターンオブジェクトブロック (POFCyclizedCommonSection) クラス1416の1つのインスタンスと、非サイクライズパターンオブジェクトブロック (POFCommonSection) クラス1408の0または1つ以上のインスタンスとを含む。パターンオブジェクトメタファイル (PatternObjMetaFile) クラス1402は、パターンオブジェクトメタファイルの機能をカプセル化し、OFMフレームワークに、メタファイルとやりとりするためのAPIを提供する。

【0522】

POFCommonSectionクラス1408は、メタファイル内の非サイクライズパターンオブジェクトの機能をカプセル化する。このクラスは、POF共通セクションインターフェース (IPOFCommonSection) 1410をインプリメントする。ObjMetaFileクラス1405は、パターンオブジェクトメタファイルのプロプライエタリベンダセクションへのクライアントアクセスを提供する。プロプライエタリベンダモジュールセクションは、POFModuleオブジェクト1412およびそれらの対応するPOFBLOBオブジェクト1414内にカプセル化される。各POFModuleオブジェクトは、ベンダの特定のモジュールに関連する。1つのベンダが2つ以上のモジュールを提供することができ、それゆえ、1つのパターンオブジェクトメタファイル内に2つ以上のモジュールセクションを有することができる。

【0523】

POFCyclizedCommonSectionクラス1416は、サイクルベースのデジタルパターンオブジェクトブロックの機能をカプセル化する。このクラスは、POFサイクライズ共通セクションインターフェース (IPOFCyclizedCommonSection) 1418をインプリメントする。それは、全てのパターンブロックタイプのために提供される、サイクルベースの共通セクションデータ、および非サイクルベースの共通データの両方へのアクセスを提供するので、POFCommonSectionクラスとは異なる。

【0524】

[OFMフレームワークとベンダモジュールコンパイラとの間のインタラクション]

一実施形態では、試験システムパターンコンパイラは、特定のサイト構成 (用いられるベンダ特有のデジタルモジュールに照らして) をターゲットにする。種々のベンダからのモジュールの試験システムへの組み込みをサポートするために、以下の手法がインプリメントされる。

1. ベンダコンパイラはパターンオブジェクトファイルを直に作成しない。代わりに、試験システムは、OFM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を提供する。用語「パターンオブジェクトメタファイル」は、オブジェクトメタファイルとも呼ばれる。OFMフレームワークは、共通パターンデータのメタファイルへの格納を管理するが、種々のモジュールベンダからのプラグインに基づいて、ベンダ特有のデータのコンパイルを呼び出す。パターンコンパイルプロセスのためのモデルは、図6に示されるように視覚化することができる。パターン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有するが、ソースファイル拡張子がパターンオブジェクトファイル拡張子によって置き換えられる。OFMは、これらのファイルに対する読出しおよび書込みのためのアプリケーションプログラミングインターフェース (API) を提供する。パターンオブジェクトメタファイルは、以下のものを格納するための手段を有する。

a. 共通ヘッダ情報

b. モジュール特有ヘッダ情報

c. モジュールベンダによって編成され、モジュールベンダによって解釈されることができるモジュール特有パターンデータ

2. 各モジュールベンダは、動的にロード可能なライブラリの形で、自ら所有するパターンコンパイラを提供する責任がある。このライブラリは、IVendorPatCompilerまたはIVendorCyclizedPatCompiler、あるいはその両方のような、OFMフレームワークによって定義される標準インターフェースの完全なインプリメンテーションを検索する能力を与える。IVendorPatCompilerインターフェースを提供する機能は以下のライブラリによって与えられる。

【0525】

```
OFCStatus create VendorPatCompiler (
    IVendorPatCompiler*&pCompiler,
    const OFCString& type);
```

【0526】

さらに、IVendorCyclizedPatCompilerインターフェースを提供する機能はベンダによって与えられる。

【0527】

```
OFCStatus create VendorCyclizedPatCompiler (
    IVendorCyclizedPatCompiler*& pCompiler);
```

【0528】

ベンダがCyclizedデジタルパターンまたは特定のタイプの汎用パターンのためのコンパイラをサポートしない場合には、上記の機能は予め定義されたOFCStatusを返し、要求されたコンパイラタイプがサポートされないことをOFMフレームワークに通知する。

3. OFMフレームワークは、パターンコンパイルのタスクをサポートするために必要とされる、ベンダコンパイラが共通データにアクセスできるようにするための1組のインターフェースを含む。これは、DUTピン、DUTソケットおよびDUTタイミングの記述をカプセル化するランタイムオブジェクトへのアクセスを含む。またそれは、パターンサブルーチンのような、他の参照されるパターンメタファイルにアクセスするための能力も含む。

4. OFMフレームワークは、モジュールベンダのコンパイラがモジュール特有のヘッダ情報およびデータをパターンオブジェクトメタファイルに書き込むことができるようにするための1組のインターフェースも含む。オブジェクトメタファイルのこのレイアウトによって、ターゲットとされるサイト内の2つ以上のモジュールが同じ場合であっても、パターンデータがモジュール毎に編成されるようになることに留意されたい。

【0529】

上記の手法によって、多数のベンダが、自らの独自開発のフォーマットを使用し続けることができるようになり、かつ多数のベンダからのモジュールがオープンアーキテクチャ試験システム内で共存できるようになる。メタファイルのベンダ特有セクションは「ブラックボックス」として取り扱われ、OFMフレームワークはそれについての特定の知識を有する必要はなく、他のベンダはそれにアクセスできない。

【0530】

別の実施形態では、パターンコンパイル中のVendor Module CompilerとOFMフレームワークとのインタラクションは、以下に記述されるステップのシーケンスにおいて例示される。

1. OFMフレームワークはOFMManagerのインスタンスをインスタンス化し、SYSTEMパターンコンパイラのコマンドラインにおいて指定される1つまたは複数のパターンのコンパイルを管理する。

2. OFMManagerは初期化されるようになる。これは、システムユーティリティズ構成ファイルを用いて、指定された（ベンダ毎およびモジュール毎）全てのパターンコンパイラ・ダイナミックリンクライブラリ（DLL）を登録することを含む。システムユーティリ

10

20

30

40

50

ティズ構成ファイルは、或る特定のシステム構成のユーティリティ設定を指定する。このファイルは、パターンコンパイラDLL（複数の場合もあり）の名前（複数の場合もあり）と、ベンダのモジュール（複数の場合もあり）毎に登録される共有されるライブラリを含む。この情報は、種々のベンダコンパイラを呼び出して、パターンソースファイルの対応するセクションをコンパイルするために、OFMフレームワークによって用いられる。登録の一部として、それがサポートするパターンブロックのタイプを見つけ出すために、各DLLが問い合わせを受ける。

3. その後、OFMManagerはPatternObjectMetaFileクラスを用いて、全パターンソースファイルに共通のデータをセーブする。このステップの一部として、フレームワークはパターンソースファイルを読み出し、このパターンソースファイルを含むパターンブロックのタイプを検索する。見つけられたパターンブロックの種々のタイプが共通セクションに格納される。フレームワークがパターンブロックの2つの広義のタイプ、CyclizedとRegular（非Cyclized）とを区別することに留意されたい。通常のパターンブロックは、それらのタイプを定義する任意の名前を有することができる。OFMフレームワークは、これらの名前を予め通知される必要はないので、システムはオープンアーキテクチャを有することができる。

4. コンパイルされるパターンソース内に含まれるパターンブロックのタイプ毎に、その後、OFMManagerはベンダパターンコンパイラDLLを問い合わせ、IVendorPatCompilerインターフェースをインプリメントするオブジェクトをインスタンス化する。サイクライズパターンブロックがコンパイルされている場合、OFMManagerはDLLを問い合わせ、代わりに、ICyclizedVendorPatCompilerインターフェース（それはIVendorCompilerインターフェースのサブタイプである）をインプリメントするオブジェクトをインスタンス化する。

5. 上記のステップ4において得られたIVendorCompiler派生オブジェクト毎に、OFMManagerはgetSupportedResources()メソッドをコールし、このモジュールによってサポートされるシステム資源のリストをゲットする。システム資源のリストを用いて（SocketファイルおよびシステムModule Configuration File(MCF)とともに）、コンパイルされる必要がある資源のコレクションが作成される。パターンコンパイルのためにベンダコンパイラがそのようなリストを提供される必要がある場合には、同じくシステム資源のリストを用いて、被試験デバイス（DUT）のためのピンのリストが作成される。コンパイルされているパターンがピン向きには記述されず、それゆえコンパイラがピンリストを必要としない場合もあるので、OFMフレームワークはこれを強制しないことに留意されたい。その場合には、IVendorPatCompilerインターフェースが、needsPinNames()メソッドをサポートし、OFMManagerが、特定のベンダコンパイラがDUTピンのリストを必要とするか否かを見いだすことができるようにする。

6. OFMManagerは、コンパイルされているパターンブロックのタイプをサポートする各IVendorPatCompilerインターフェースのcompile()メソッドを呼び出す。それは、このメソッドに以下の引き数を与える。

a. パターンソースにおいて参照される資源タイプのリスト、および（オプションで）そのような各資源タイプのピン名のリスト。ピンが必要とされない場合には、ピンのリストは空である。

b. ベンダパターンコンパイラが、パターンオブジェクトメタファイルに読出しアクセスおよび書き込みアクセスできるようにするためのOFModuleAgent派生オブジェクト（またはサイクライズパターンの場合のPOFCyclizedModuleAgent）のインスタンス。

7. パターンソースの各ブロックをコンパイルした後に、OFMManagerは、最初に用いられたコンパイラに、共通セクションデータを問い合わせ、その後、それを用いて、パターンオブジェクトメタファイルが更新される。

8. その後、OFMManagerは各ベンダパターンコンパイラに、出合ったエラー／警告のリストを問い合わせる。このエラー／警告のリストは一纏めにされて、ユーザに提供される。

10

20

30

40

50

9. コンパイルされるパターンソースファイル毎にステップ3～8が繰り返されて、対応するパターンオブジェクトメタファイルが生成される。

10. 最後に、OFMManagerは、このステップシーケンスのステップ2において登録された全てのベンダパターンコンパイラを公開する。これにより、種々のベンダコンパイラが、コンパイルプロセスが終了する前に、クリーンアップできるようになる。

【0531】

[パターンデータのローディング]

本発明の一実施形態の試験システムでは、各モジュールベンダが、自ら所有する特有のパターンローディング機構を提供する責任を担う。先に説明されたように、パターンオブジェクトメタファイルは、種々のセクションにモジュール特有データを格納する。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関連するセクションにアクセスするために、システムアプリケーションプログラミングインターフェース (API) を用いる。さらにOFMフレームワークは、各モジュールのロードパターンメソッドをコールし、モジュール特有のデータをメタファイルの適当なセクションからモジュールにロードする責任を担う。

【0532】

パターンローディングのためのOFMフレームワークとベンダモジュールとの間のインタラクションを例示するステップのシーケンスが以下に説明される。

1. PatternMgrオブジェクトが、パターン実行シーケンスを含むパターンのコレクションをロードするための責任を担う。このオブジェクトは、ロードされるパターン毎にPatternLoaderインスタンスをインスタンス化する。

2. PatternLoaderは、ロードされるパターンのためのPatternObjMetaFileクラスのインスタンスをインスタンス化し、それに、パターンオブジェクトメタファイル内のパターンブロックのタイプのリストを問い合わせる。PatternLoaderはさらに、PatternObjMetaFileクラスに、各パターンブロックタイプ内に含まれるモジュールのタイプを問い合わせる。

3. その後、PatternLoaderは、ベンダによって提供される対応する派生IModuleクラスのloadPattern()メソッドを呼び出し、ブロックおよびモジュールのタイプ毎にパターンデータをロードする。

4. 対応するIModuleインターフェースのloadPattern()メソッドは、PatternLoaderクラスに渡された参照を用いる。その後、それはPatternLoaderクラスのload()メソッドを用いて、PatternObjMetaFileクラスにアクセスし、それからモジュール特有のデータを読み出し、それによりパターンのローディングを達成する。

【0533】

[E5. パターンファイル]

各コンパイラベンダは複数のパターンのための種々のプレーンテキストフォーマットを完全に指定することができ、それは実際には大部分の場合に必要なであろう。しかしながら、一般的に、全てのベクトルのために複数のモジュールにわたる統一性および同一の意味が必要になる、サイクルベースの試験環境の場合、パターンファイルのための、共有され、一般化された構文が望ましいだけでなく、必要になるであろう。この共有された構文は、パターンソースファイル内の「共通」セクションのために指定されることになる構文である。実際には、大抵の場合に、「共通」セクションはパターンファイルにおいて必要とされる唯一のセクション（ヘッダ情報を除く）であり、全てのベンダのコンパイラはそのセクションにおいてのみ正常に動作することが想定される。このセクションは、全てのコンパイラが解釈できるべきである、パターンファイルのための規則を表す。パターンファイルは以下のように編成されるであろう。

【0534】

```
file_contents      :
                    version_info pattern_definitions
version_info      :
```

10

20

30

40

50

```

        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_domains '{'
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    :
        main_section_contents main_section_content
        main_section_content
main_section_content    :
        label_spec main_pattern_spec
        main_pattern_spec
label_spec    :
        label ':'
label    :
        identifier
main_pattern_spec    :
        main_operation capture_mask_flag '{'
vectors_and_waveforms '{'
main_operation    : /* empty */
        common_operation
        jal_op
        jsr_op
        jsrc_op
        jsc_op
        exit_op
common_operation    :
        idxi_op
        idxin_op

```

```

        jec_op
        jech_op
        jff_op
        jffi_op
        jni_op
        ldin_op
        nop_op
        pause_op
        sndc_op
        sndt_op
        stfi_op
        sti_op
        stps_op
        wait_op
/*
 * Instructions specific to the MAIN Patterns
 */
jsr_op      :
              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

```

10

20

30

40

50

```

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    :

```

10

20

30

40

50

```

CommonSection '{' common_contents
source_selection_table subr_section_domains '{'
CommonSection '{' common_contents
subr_section_domains '{'
subr_section_domains :
    subr_section_domains subr_section_domain
    subr_section_domain
subr_section_domain :
    Domain domain_name '{' subr_section_contents '{'
source_selection_table :
    SourceSelectionTable '{' source_selector_definitions '{'
source_selector_definitions :
    source_selector_definitions source_selector_definition
    source_selector_definition
source_selector_definition :
    SourceSelector source_selector_name '{'
source_mappings '{'
source_selector_name :
    identifier
source_mappings :
    source_mappings source_mapping
    source_mapping
source_mapping :
    pin_name '=' source ';'
source :
    MAIN
    INVERT_MAIN
    SUBR
    INVERT_SUBR
subr_section_contents :
    subr_section_contents subr_section_content
    subr_section_content
subr_section_content :
    label_spec_subr_pattern_spec
    subr_pattern_spec
subr_pattern_spec :
    subr_operation capture_mask_flag '{'
vectors_and_waveforms '{'
subr_operation : /* empty */
    common_operation
    rtn_op
    stss_op
/*
 * Instructions specific to the SUBR Patterns
 */
rtn_op :
    RTN
stss_op :
    STSS identifier

```

以下は、先に用いられた未定義の非終端記号の記述である。

1. version-identifier: 集合 [0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は数字でなければならない。

2. identifier: 集合 [a-z A-Z _ 0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-z A-Z _] から選択されなければならない。

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

4. file-name: 有効なウィンドウズファイル名 (ファイル名内に任意の空白が含まれる場合には、それは二重引用符で囲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持つべきでないことに留意されたい。

10

5. waveform-table-pin-group-name: 集合 [a-z A-Z _ 0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-z A-Z _] から選択されなければならない。この変数は他の場所で宣言され、ピンのグループに共通である波形テーブルの名前を保持する。

6. 24bit number: 最大16777215までの有効な10進数。

7. 8bit number: 最大256までの有効な10進数。

8. index-register: 有効な10進数。1つのモジュールの実施形態では、これは値 [1-8] を有することができる。

9. vector: これは、S T I L 内の V e c t o r ステートメントに類似である。これは、信号名および信号グループ名を参照し、コンパイラがPin Descriptionファイルにアクセスできるようにするために必要になることに留意されたい。

20

10. waveform-time-reference: 集合 [a-z A-Z _ 0-9] からの1つまたは複数の文字からなる文字列であり、最初の文字は [a-z A-Z _] から選択されなければならない。

【0536】

パターンファイルはコメントをサポートし、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になっている。コメントは「#」文字で始まり、その行の終わりまで続く。

【0537】

パターンファイルのヘッダおよび「共通」セクションの構成体を参照して、以下の点に留意されたい。

30

1. pattern-name項目は、パターンファイルがそのためのデータを含む P a t t e r n オブジェクトに関連付けられることになる名前を指定する。これは、対応するパターンオブジェクトメタファイル内のヘッダに引き継がれるようになる。

2. waveform-time-referenceは、Timingファイル内の、パターンファイルに対して外部から定義されることとなる特定のwaveform-and-timing定義のための名前である。パターンファイルにおいてwaveform-time-referenceを指定することにより、別のwaveform-time-referenceが現われるまで、(waveform-and-timingのための) 特定の名前が全ての後続のベクトルに結合されるであろう。

40

3. サブルーチンコールのためのオペランド (たとえば、J S R および J S R C) は、同じパターンファイルにおいて以前に現われたpattern-specラベル、または外部から定義されたサブルーチンパターン内のpattern-specラベルであるストリングである。このオペランドは最終的には、サブルーチンをロード/処理するために分解されるであろう。サブルーチンコールオペランドのためのラベルは、システムにわたって固有である必要がある。

【0538】

waveform-time-reference名は、構文的に正確であるなら何でも用いることができるが、特定のハードウェア含意に起因して、waveform-time-reference名は予めわかっている明確な集合に限定される必要があり得る (さらに読みやすくするために、それはオプショ

50

ンでは、ユーザ選択の名前にユーザによってマッピングされることができ、そのマッピングはオプションのファイルにおいて提示される）ことに留意されたい。

【0539】

また、パターンおよびwaveform-time-referenceソースファイルは、物理的なテストチャンネルに接続される全てのDUTチャンネルのための初期構成データを提供すべきであることにも留意されたい。任意のDUTチャンネルの場合に後続のデータが省略される場合には、パターンコンパイラは、パターンデータを「パッド」して、初期レベルからの出力を保持するであろう。

【0540】

[パターンファイル例]

10

MAIN Patternソースファイルの簡単な例が、その使用法を例示するのを助けるであろう。

【0541】

```
#
# Filename : goodl.pat
#
Version 1.0 ;
# -----
# Main Pattern definition:
# -----
MainPattern goodl
{
    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
{DATA = wfs1; CLK = wfs1; } }
            JAL    myAPG    { V { DATA = 00000000; } }
            JSC    mySCAN   { V { DATA = 10101010; } }
            JSRC   mySubroutine { V { DATA = 01010101; } }
            JSR    myAPG    { V { DATA = 00110011; } }
            STI    100      { }
```

20

30

40

50

```

labZero:  NOP    { V { DATA = 00000011; } }
          JNl     labZero { V { DATA = 11111100; } }
          IDXI    3000   { V { DATA = 10101010; } }
          IDXIIn  3      { V { DATA = 01010101; } }
$label1   NOP    { V { DATA = $defaultDataVal; } }
          IDXI    2000   { V { DATA = 10101010; } }
          NOP     { }
          EXIT     { V { DATA =LLHHLHH; } }

      }
  }
}

```

10

【0542】

SUBROUTINEパターンソースファイルを例示する別の例が以下に示される。

【0543】

```

# -----
# Subroutine Pattern mySubrPat1 definition:
# -----
SubrPattern mySubrPat1
{
    CommonSection
    {
        # -----
        #   Timing Specifications
        # -----
        Timing "productionTiming.tim";
        TimingMap "productionTimingOpenSTARMap.tmap";

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

        # -----
        #   Default Domain Cycles
        # -----
        Domain default
        {
            # -----
            # label :   instruction   { Vector and Waveform Data setups }
            # -----

```

20

30

40

50

```

STI    100    { Vector { DATA =00000000; } }
IDXI   3000   { Vector { DATA = 00001111; } }
IDXIn   3     { Vector { DATA = 00110011; } }
$label1 NOP    { Vector { DATA =LLHHLH; } }
        NOP    { Vector { DATA =LLXXXXXX; } }
        NOP    { Vector { DATA =LLHHXXXX; } }
        JNI Label1 { Vector { DATA =LLHHLH; } }
        STSS SrcSelOne { Vector { DATA = LHLHLHLH; } }
        RTN    { Vector { DATA = LLXXXXXX; } }
    }
}
}

```

10

【0544】

パターンソースファイル内のメインヘッダおよび共通セクションからの要約情報は、オブジェクトメタファイル内のメインヘッダに格納される。その要約は、アドレスなどのプリロード分解を助けるために、またはデータロギングにおいて助けるために、迅速に抽出するために通常必要とされる情報からなる。共通セクションの意味は全てのコンパイラの場合と厳密に同じであるので、全てのコンパイラが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1のコンパイラがこの情報を格納するであろう。以下は、格納されることになる情報である。

20

1. パターンソースファイル名
2. ソースファイルにおいて宣言されるようなパターンのタイプ
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用いられる全てのwaveform-and-timing名のリスト
5. パターンソースファイルの共通セクション内の（相対的な）ベクトルアドレスへの全てのサブルーチン参照のマップ
6. パターンソースファイルの共通セクション内の（相対的な）ベクトルアドレスへの全てのラベル参照のマップ
7. 一般的なブックキーピング情報：ベクトルカウント、命令カウントなど。

30

【0545】

オープンアーキテクチャ試験システムは、明示的で、かつ異なる拡張子を有するために、パターンおよびパターンリストファイルの両方を必要とする。パターンファイルの場合、これは、プレーンテキストソース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当てはまる。これは、ディレクトリリスティング等において視覚的にファイルタイプを迅速に特定するために、および拡張子に基づいて連想がなされるようにするために、ユーザにとって都合であると考えられる。パターンリストファイルパーサは、これらの拡張子を用いてファイル名を予想するであろう。

プレー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

40

【0546】

ユーザは、たとえば、テスト環境変数または設定オプションを通して、これらのデフォルト値を無効にすることができる。

【0547】

テストは、以下に記述される環境構成ファイルのうちの少なくとも1つにおいて、ファイル探索経路のための以下の「環境」変数の定義を必要とするであろう。

Tester_PATLIST_PATH：パターンリストファイルの場合。

Tester_PATSRC_PATH：パターンソースファイル（オプション）の場合。

Tester_PATOBJ_PATH：パターンオブジェクトメタファイルの場合。

50

【0548】

オプションの環境／設定変数Tester_PATSRC_PATHが定義されない場合には、Tester_PATOBJ_PATHと同じであると仮定されることに留意されたい。一般的に、Tester_PATOBJ_PATHと同じ値である場合、Tester_PATSRC_PATHを定義するよりも、定義しないほうが、より効率的であろう。

【0549】

[E6. ソフトウェア表現]

パターンオブジェクトはユーザによって作成されない。むしろ、ユーザは常に、他のパターンリストおよび／またはパターンのコレクションであるパターンリストオブジェクトを処理する。ユーザがアクセスできるようにしながら、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作成し、所有し、保持する。ユーザの試験プログラム内の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む、ディスク上のパターンリストファイルに関連する。パターンリストの定義は、そのパターンリストのための明示的な名前を提供し、ファイル名連想を通して、パターンの順序付きリストおよび／または他のパターンリストを特定する。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がテストフレームワークにおいて如何に操作されるかを理解するための前置きとして、それらのソフトウェア表現を記述する。

【0550】

[パターンリスト連想]

試験システム内の1つの試験サイト（および拡大解釈して、その中にある試験計画）は、多数のトップレベルパターンリストに関連付けることができる。しかしながら、複数の試験計画に対して、常に1つの実行文脈だけが存在する。トップレベルパターンリストは、それによって（階層的に）参照されるパターンのための実行シーケンスを定義するので、アクティブな実行文脈は、現在選択されているトップレベルパターンリストに対応するものである。これは、ある時点において、1つのパターンリスト内に含まれるパターンだけがハードウェア上にロードされることができるとを意味しないことに留意されたい。むしろ、1つの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必要がある1組のパターンは常に、現在ロードされている全てのパターンのサブセットでなければならない。

【0551】

[パターン木]

直観的には、トップレベルパターンリストを表現するための1つの方法は、ある種の木データ構造によると考えられる。図7は、本発明の順序付きのパターン木の1つの実施形態を示しており、パターンリストAがトップレベルパターンリストであると仮定する。

【0552】

[パターン木情報内容]

以下の情報がパターン木の全てのノードにおいて格納されるであろう。

1. そのノードに関連するエンティティ（パターンリストまたはパターン）の名前。
2. 定義ソースのタイプ。葉（パターンノード）の場合、これは常にパターンファイルになるであろう。中間（パターンリスト）ノードの場合、これは「トップレベルファイル」（トップレベルパターンリスト定義用）、または「エンベデッド・イン・ファイル」（ネストされたパターンリスト定義用）のいずれかにすることができる。
3. そのノードが関連するディスク上のファイルの最後の変更タイムスタンプ。

【0553】

以下の付加情報は、中間（パターンリスト）ノードにおいてのみ格納されるであろう。

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（もしあるなら）セットされる実行オプション、すなわちそのオブジェクトオプション。
2. その子毎に、そのノードによって表されるパターンリスト定義内の各子参照上に（もしあるなら）セットされる実行オプション、すなわち参照オプション。

【0554】

こうして、ルートから中間ノードへの固有の経路において現われるノードのコレクション、およびそれらに出合ったシーケンスは、そのノードによって表される、組み合わせられた有効な実行オプションを決定するために必要とされる全ての情報を含む。1つのパターンの実行オプションは、その中間の親の有効な実行オプションと、それと組み合わせた、その中間の親がそのために有することができる参照オプションによって決定される。

【0555】

ここで、パターンリストパーサがパターン木を作成している過程にあるときに、その使用の文脈が後の時点まで分解されない場合もあるので、ある特定の実行オプションが単にストリングとして値を最初に格納する必要があるかもしれないことに留意されたい。そのようなオプションの1つの例は「マスク」オプションであり、それは、ピンマスク情報を指定する。パターンリストはSocket情報に関連しないので、ピンマスクオプション（ピンおよびグループ名）は、ローディング前に分解されるべきストリングとして格納される。

10

【0556】

以下の付加情報は葉（パターン）ノードにおいてのみ格納されるであろう。

1. 実行木として編成される、外部および内部両方のパターンによってコールされるサブルーチンへの全ての（おそらく遷移的な）参照。

【0557】

当然、全てのパターンノードはさらに、オブジェクトメタファイル共通ヘッダにおいて入手可能な全てのパターンファイル要約情報にアクセスでき、そしてそれらをキャッシュすることができるであろう。

20

【0558】

[パターンリスト変更の処理]

パターンリストの内容に対してなされる変更は概念的には、そのパターンリストへの全ての参照に影響を及ぼす。必要に応じてパターンオブジェクトおよびパターンリストオブジェクトに適用される以下の規則が、そのような変更を管理するために用いられるであろう。

1. ディスク上のパターンリストファイルの内容に対してなされる変更は、そのパターンリスト上（またはそれを参照する任意の他のパターンリスト上）で実行されるload() コマンドにおいてのみ、試験システムの中に伝えられるであろう。言い換えると、ソフトウェア内のパターンリスト階層は常に、ハードウェア上に現在ロードされているパターンリストを反映するであろう。

30

2. ユーザは、パターンリストをそれらのディスクファイルソースと同期させるために、ロード時間中に行われる検査を無効にするモードをセットすることができるであろう。これにより、生成モードの動作を、より迅速かつ安全にすることができるであろう。

【0559】

[パターン木ナビゲーション]

試験サイト（および拡大解釈して、そのサイトのための試験計画）に関連するトップレベルパターンリストは公開（グローバル）範囲を有する。そのシステムは、ユーザが個々のノードおよび副木にアクセスできるように、トップレベルパターンリストを表すパターン木をナビゲートするためのAPIを提供する。

40

【0560】

[E7. パターンリスト動力学]

パターンリストの静的な規則は先に説明された。パターンリストの動的な（実行）規則の記述がここで提示される。

【0561】

パターン木は包括的なパターン管理のために不可欠である。たとえば、1つのパターンロードシーケンスのための開始点は、サイトまたは試験計画に現在関連しているパターン木上でのload()メソッドへのコールである。しかしながら、1つのパターン木は孤立して動作しない。完全に初期化されたパターン木を用いて、以下の2つのフレームワークオブ

50

ジェクトが作成されるであろう。

1. トップレベルパターンリストは、複数のパターンのための1つのPattern Execution Sequenceを定義する。それは、そのような実行シーケンスをそのトップレベルパターンリストに対応するパターン木から如何に導出することができるかを記述する。たとえば、図7に示されるパターン木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。Pattern Execution Sequenceは概念的には、そのパターン木を通して記述される実行シーケンスを反映する順序木である。フレームワークは、パターン木ノードと、Pattern Execution Sequence内の対応するエン트리との間の任意の必要なナビゲーションリンクを確立し、保持する。

2. Pattern Setは、そのパターン木内の全ての固有のパターン（サブルーチンを含む）のリストに過ぎない。したがって、これは、ハードウェア上にロードされるべきである個々のパターンを決定するために用いられることになるリストである。フレームワークは、パターン木ノードと、Pattern Set内の対応するエン트리との間の任意の必要なナビゲーションリンクを確立し、保持する。図7のパターン木のためのPattern Setは（q、s、t、r、u、v）である（パターンリストA内のパターンはいずれも如何なるサブルーチンコールも含まないものと仮定される）。

【0562】

Pattern Execution SequenceおよびPattern Setはいずれも、常に、パターン木から導出することができることに留意されたい。しかしながら、多くの場合に、実用的である限り、初期構成後に、それらをキャッシュことが理にかなっているであろう。

【0563】

[パターンリスト実行オプション]

上記のように、各パターンリスト宣言（その定義に先行する）またはパターンリスト／パターン参照エントリの後に、多数の実行オプションが続くことができる。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変更する。さらに拡張できるようにするために、これらのオプションのための名前（およびオプション値）がパターンコンパイラのパターンリストファイルパーサによって単にストリングとして処理され、必要に応じて特定のバージョンによって解釈されることになる。テストは、以下に記述される、1組のオプションおよびそれらの解釈を規定する。しかしながら、ベンダはその1組のオプションを拡張することができる。オプション構文をパース時に妥当性検査できるようにするために、パターンリストファイルパーサは、ある特定のバージョンのための情報ファイルを読み出すことができる。そのような情報ファイルを用いて、ある特定のバージョンがそもそも実行オプションの指定をサポートするか否かを指定することもできる。

【0564】

1組の実行オプションをサポートするバージョンの場合、以下の一般的な規則がそれらの使用法を決定するであろう。これらの規則を理解するために、パターンリスト／パターンの階層的なコレクションを順序木として視覚化することが有用である。

1. パターンリスト定義上（すなわち、ファイル内の「local-pattern-list-declaration、global-pattern-list-declaration」生成物内）でセットされるIntrinsicオプションは、事実上、ユーザの試験プログラム内の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、これは、そのパターンリストオブジェクトへの全ての参照に当てはまり、オブジェクトオプションと呼ばれる。

2. パターンリスト／パターンへの参照上（すなわち、ファイル内の「pattern-entry」および「pattern-list-entry」生成物内）でセットされるReferentialオプションは、オプションの範囲を、その階層内の特有の経路、すなわち木のルートから考慮中の参照に導く経路（パターンリスト／パターンの宣言順序によって確立される）に制限する。したがって、これらは、特定のオブジェクト参照上のオプションであり（且つ、オブジェクトそのものにおけるオプションではない）、参照オプションと呼ばれる。

3. コレクション階層（パターンリスト／パターンの宣言順序によって確立される）内の任意のリスト／パターンのための有効オプション設定は、木のルートからそのリスト／

10

20

30

40

50

パターンまでの経路に沿って出合うオブジェクトオプションおよび参照オプションの組み合わせである。その特定の組み合わせの仕組み（たとえば、集合の結び、集合の交わり、または任意の他の競合解消アルゴリズム）は、オプションそのもののプロパティである。

【0565】

上記の規則の結果、そして1つのパターンファイル内のパターン定義上で実行オプションをセットするためのファシリティがないという事実は、1つのパターンへの全ての参照に当てはまるオプションをセットするための直接的な規則が存在しないということであることに留意されたい。これを果たすための仕組みは、単一パターンのパターンリストを用いることである。

【0566】

テストは、そのバースト挙動を変更し、その実行シーケンスを変更する、ある1組のパターンリスト実行オプションを指定する。

【0567】

1つの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示されるとき、ハードウェアはバーストを生成する。Burstは、ソフトウェアが何も関与することなく、ハードウェアによって直にパターンのシーケンスを実行することである。Burst Discontinuityは、先行するバーストが終了し、新たなバーストが開始される、実行シーケンス内の場所である。

【0568】

パターン管理ソフトウェアの目的の1つは、バーストを生成するために必要とされる実行シーケンスをハードウェアに与えることである。デフォルトによって、1つのパターン木は1つの実行シーケンスを生成し、それがハードウェアに提示される場合には、結果として1つのバーストが生成されるであろう。しかしながら、この挙動は、パターンリスト上でオプションを用いることにより変更することができる。したがって、オプション結果を用いる結果として、バースト不連続部が生じ得る。

【0569】

さらに、ユーザは多くの場合に、全てのパターンまたは全てのバーストの前または後に実行されることになるプロローグまたはエピローグパターンを必要とするであろう。これは、ハードウェアに提示されることになる実行シーケンスを変更する。

【0570】

Pattern Execution Sequenceオブジェクトの作成または変更中に、フレームワークは、指定された実行オプションと、パターン木によって具現される特定の実行シーケンスとの組み合わせから生じるパターンバースト内のブレイクを判定するために必要な全ての情報を有し、必要に応じて報告する。これを果たすときに、システム内のモジュールのハードウェア能力を調査することが必要な場合もある。たとえば、1つのハードウェアインプリメンテーションによれば、ピンマスクのために4つの構成を格納できるようになり、そのうちの2つ（0および3）は（Mask This Vector、MTVをサポートするために）デフォルトマスクおよびアンマスク演算のために用いられる。こうして、ユーザは、バーストモードをブレイクすることなく、2つの異なるグローバルピンマスク構成を許される。

【0571】

或るモジュールベンダがハードウェアにおいてパターンリストインプリメンテーションをサポートしない場合には、Pattern Execution Sequenceのベンダの処理の結果として、実行シーケンス内の全てのパターンが個別に実行されることになることに留意されたい。Site-CompatibleおよびSite-Heterogeneousの両方のシステムにおいて、サイトのバースト能力は、「最小共通分母」によって制限されるであろう。テストは、ある特定のデフォルトの1組のオプションを提供し、それらのパラメータが以下に記述される。各オプションは以下のことを述べることにより指定される。

・それがIntrinsicである（すなわち、GlobalまたはLocalキーワードを伴う定義に関連する）か、Referentialである（すなわち、PatまたはPListキーワードを伴う参照に関連する）か。Intrinsicオプションは、定義の場所および全ての参照において当てはまるが

10

20

30

40

50

、Referentialオプションは、それらが関連する参照においてのみ当てはまる。

・さらに、オプションが全ての静的に（構文的に）または動的に（参照されることにより意味的に）ネストされたパターンまたはパターンリストに再帰的に当てはまるものと仮定される場合には、そのオプションは「子によって継承される」と言われる。

【0572】

以下はオプションのリストである。全ての準拠するベンダは、これらのオプションを指定されたとおりに解釈するであろう。

1. Mask<pin/pin group>

GlobalPList、LocalPListに適用されるときにIntrinsic。

PList、Patに適用されるときにReferential。

子によって継承される。

このパターンリストは常に、使用禁止の指示されたピンまた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するであろう。多くの場合に、ハードウェアの制限の結果として、バースト不連続部が生じることになる。

2. BurstOff

GlobalPList、LocalPListに適用されるときにIntrinsic。

PList、Patに適用されるときにReferential。

子によって継承されない。

このパターンリストは常に、非バーストモードにおいて実行されるであろう。このオプションは子によって継承されないが、BurstOffDeepオプション（下記）は子によって

3. BurstOffDeep

GlobalPList、LocalPListに適用されるときにIntrinsic。

PList、Patに適用されるときにReferential。

子によって継承される。

このパターンリストは常に、非バーストモードにおいて実行されるであろう。このオプションは子によって継承されるが、BurstOffオプション（上記）は子によって継承されない。BurstOffDeepオプションは子によって消されないことに留意されたい。

4. PreBurst<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

指定されたバーストオプションを有しない子ノードによってのみ継承される。

指示されるパターンは、このパターンリスト内の全てのバーストに前置されるべきである。PreBurstパターンは、このパターンリストノードに起因して開始される全てのバーストの直前に生じる。そのオプションは、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>

GlobalPList、LocalPListに適用されるときにIntrinsic。

指定されたバーストオプションを有しない子ノードによってのみ継承される。

指示されるパターンは、このパターンリスト内の全てのバーストに後置されるべきである。PostBurstパターンは、このパターンリストノードに起因して開始される全てのバーストの直後に生じる。そのオプションは、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>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

指示されるパターンは、このパターンリスト内の全てのパターンに前置されるべきである。

7. PostPattern<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

指示されるパターンは、このパターンリスト内の全てのパターンに後置されるべきである。

8. Alpg<alpg object name>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

名前付きALPGオブジェクトは、低速APGレジスタ設定、読出し待ち時間、中間データレジスタ、アドレススクランブル、データ反転、データ発生器などの関連する情報を格納する。

9. StartPattern<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

そのパターンリストは、その実行シーケンスにおいてStartPatternが最初に現われるときに実行し始めるであろう。

10. StopPattern<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

そのパターンリストは、その実行シーケンスにおいてStopPatternが最初に現れるときに実行するのを止めるであろう。

11. StartAddr<vector offset or label>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

これはStartPatternオプションを伴わなければならない。そのパターンリストは、その実行シーケンスにおいてStartPatternが最初に現われるときのStartAddrにおいて実行し始めなければならない。

12. StopAddr<vector offset or label>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

これはStopPatternオプションを伴わなければならない。そのパターンリストは、その実行シーケンスにおいてStopPatternが最初に現われるときにStopAddrにおいて実行するのを止めなければならない。

13. EnableCompare_StartPattern<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

パターン比較は、指示されるパターンが最初に現われるときに開始するであろう。

14. EnableCompare_StartAddr、EnableCompare_StartCycle

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

これは、EnableCompare_StartPatternを伴わなければならない。パターン比較が開始することになるパターン内のアドレスまたはサイクルを指示する。

15. EnableCompare_StopPattern<pattern>

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

パターン比較は、指示されるパターンが最初に現われるときに終了するであろう。

16. EnableCompare_StopAddr、EnableCompare_StopCycle

GlobalPList、LocalPListに適用されるときにIntrinsic。

子によって継承されない。

これは、EnableCompare_StopPatternを伴わなければならない。パターン比較が終了することになるパターン内のアドレスまたはサイクルを指示する。

17. Skip

PList、Patに適用されるときにReferential。

子によって継承されない。

パターンリストによって支配されるパターンまたはサブシーケンス全体がスキップされるようにする。これは、このパターンリスト副木のルートにおいて全てのオプションもスキップさせるであろう。それは、このパターン副木が実行するために存在しなかったかのようにする。

【0573】

[パターンリストバースト制御]

先に説明されたように、或る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示されるとき、ハードウェアは、ソフトウェアが関与することなく、パターンのシーケンスのバーストを生成する。バースト不連続部は、先行するバーストが終了し、新たなバーストが開始される、実行シーケンス内の場所である。上記のオプションリストに示されるように、PreBurst、PostBurst、BurstOffおよびBurstOffDeepオプションは、バースト不連続部が生じる場所を制御する。PreBurstおよびPostBurstオプションは、以下に記述されるある特定の付加的な規則に対するバースト不連続部サブジェクトを決定する。

1. 親リストがPreBurstおよびPostBurstオプションを有し、ネストされたリストが同じ対応するオプションを有するとき、バースト不連続部は存在せず、ネストされたリストのPreBurstおよびPostBurstオプションは当てはまらない。親リストのPreBurstおよびPostBurstを適用するただ1つのバーストだけが存在する。

2. ネストされたリストがバーストオプションを持たないとき、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オプションを有することと同じであることに留意されたい。結果として、バーストオプションを持たないネストされたリストはバースト不連続部を生じない。

3. 上記の規則1が当てはまらず、かつ親リストの開始からネストされたリストの開始までパターン実行シーケンスへの寄与がある場合には、ネストされたリストの開始時にバースト不連続部が存在する。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。ネストされたリストのPreBurstおよびPostBurstは、ネストされたリストに当てはまる。

4. 上記の規則1が当てはまらず、かつネストされたリストの終了から親リストの終了までのパターン実行シーケンスへの寄与がある場合には、ネストされたリストの終了時にバースト不連続部が存在する。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。ネストされたリストのPreBurstおよびPostBurstは、ネストされたリストに当てはまる。

5. 上記の規則1が当てはまらず、かつネストされたリストから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い場合には、親リストのPreBurstおよびPostBurstは当てはまらない。ネストされたリストのPreBurstおよびPostBurstを適用するただ1つのバーストだけが存在する。

【0574】

以下は、実行シーケンスへのオプションの影響を示す数例である。簡略化するために、全てのパターンリストがただ1つのファイルにおいて指定されるものと仮定する。

【0575】

[例1：BurstOffの使用]

この例はBurstOffおよびPreBurstを例示する。特に強調されるのは、BurstOffによって、複数のパターンが、1パターン長である複数のバースト内で1つずつ実行されることである。それゆえ、PreBurstオプションは依然として当てはまる。入力パターンリストは以下のとおりである。

【0576】

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

10

20

30

40

50

```

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;
};

```

【0577】
Aをルートとする木は図8に表されるであろう。

【0578】
このパターンのための実行シーケンスが以下に示される。|文字はバーストブレイクを示す。このパターンリストは10バーストにおいて実行され、第1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。
z q | a b | z r | z s | t | c | d | c | d | e

【0579】
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。

1. A上のBurstOffオプションはBによって継承されないので、B内のパターンaおよびbはバーストとして動作する。
2. A上のPreBurstオプションはBによって継承されないので、Bによるバースト内のaおよびbはzによって前置されない。
3. zによる前置は、aの直接の子であることに起因して実行されるパターン、すなわちパターンq、rおよびsの場合にのみ起こる。これらのパターンは、BurstOffオプションを有するAに起因して1パターン長だけであるバースト内にあるかのように1つずつ実行される。BurstOffは、パターンに、1パターン長バーストにおいて個別に実行されることを要求する。それゆえ、PreBurstおよびPostBurstオプションは依然として当てはまる。
4. パターンリストDは、その子cおよびdが1つずつ実行される固有バーストオフオプションを有する。それは、AからのPreBurst zを継承しない。

【0580】

10

20

30

40

50

[例 2 : BurstOffDeepの使用]

この例はBurstOffBurstOffオプションを例示する。パターンリスト定義中のBurstOffDeepは、ネストされる定義および参照されるリストに影響を及ぼす。しかしながら、PreBurstおよびPostBurstオプションは、ネストされたリストおよび参照されたリストによって継承されない。その例は、例 1 の場合と同じパターン A、B、C、D、E を用いるが、それらのオプションが異なる。

5. A の定義上のオプション : [BurstOffDeep]、[PreBurst z]、[PostBurst y]

6. 任意の他のノード上の他のオプションはない。

【0581】

実行シーケンスは以下のとおりである。先に説明されたように、| 文字はバーストブレイクを示す。

z q y | a | b | z r y | z s y | t | c | d | c | d | e

【0582】

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。

1. PreBurst および PostBurst は B、C、D、E によって継承されない。

2. BurstOffDeep は B、C、D、E によって継承される。

【0583】

[例 3 : PreBurst および PostBurst 禁止]

ここで、例 1 のパターンリスト木について考えるものとする。ただし、オプションは以下のとおりである。

1. A の定義上のオプション : [PreBurst x]、[PostBurst y]

2. C の定義上のオプション : [PreBurst x]、[PostBurst z]

3. 任意の他のノード上の他のオプションはない。

【0584】

実行シーケンスは以下のようになるであろう。

x q a b r s t c d c d e y

【0585】

「t c d」サブシーケンスが「x t c d z」でない理由は以下のとおりである。

1. 最初の x は、実際には現在のバーストに関連するプレバーストオプション x に等しいので、禁止される。

2. 最後の z は、PostBurst z が D に継承されず、z が付加されることができない C から生成されるパターンが存在しないので、禁止される。

【0586】

[例 4 : Skipの使用]

この例は、ネストさ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例示する。その例は、例 1 の場合と同じパターン A、B、C、D、E を用いるが、それらのオプションが異なる。

1. A の定義上のオプション : [Skip]、[PreBurst z]、[PostBurst y]

2. r への参照上のオプション : [Skip]

3. C の定義上のオプション : [Skip]

【0587】

その実行シーケンスは、以下のような、ブレイクのない単一のバーストである。

z q a b s c d e y

【0588】

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。

1. r および C のためのノードはスキップされる。

2. バーストブレイクは全く存在しない。

【0589】

[例 5 : Maskの使用]

この例はMaskオプションの影響と、パターンならびにパターンリスト定義および参照へ

10

20

30

40

50

のその影響とを例示する。その例は、例 1 の場合と同じパターン A、B、C、D、E を用いるが、それらのオプションが異なる。

1. A の定義上のオプション： [mask pin1_pin2] 、 [PreBurst z]
2. B の参照上のオプション： [mask pin3]
3. B の定義上のオプション： [mask pin4]
4. e の参照上のオプション： [mask pin5]
5. 任意のノード上の他のオプションはない。

【0590】

名前「pin1_pin2」はPin1およびPin2をマスクするグループを指定する。名前「pin3」、「pin4」および「pin5」はそれぞれPin3、Pin4およびPin5をマスクすることを指定する。実行シーケンスが以下に与えられ、| はバーストブレイクを示している。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す。

【0591】

```
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
2 2 2 2 2 2 2 2 2 2 2 2 2
   3 3                               5
   4 4
```

【0592】

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。

1. ベンダのハードウェアは、バーストブレイクを生じることなく、2つのマスクブロックだけしか収容することができない。e が実行されるまで、2つ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 がピン {1、2、5} の異なるマスクブロックで到着するとき、ハードウェアはバーストブレイクを要求する。

【0593】

[例 6：継承されるオプションおよび参照の使用]

この例は、或る定義において継承されるオプションが、その定義が参照されるときには当てはまらないことを例示する。以下の例について考える。

【0594】

```
Global A
{
    Global B [BurstOffDeep]
    {
        Global C
        {
            ...
        };
        ...
    };
    ...
```

```
PList C;
};
```

```
Global D
{
    PList C;
};
```

【0595】

BurstOffDeep オプションは、その定義の時点において C によって継承される。しかしな

がら、それは固有オプションではないので、その参照のいずれの時点においても C に適用されない。

【0596】

[例7：ネストされたリストを有するPreBurstおよびPostBurst]

以下の例について考える。

【0597】

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;

}

【0598】

実行シーケンスは以下のとおりである。

x p1 p2 p3 y | x p4 z | w p5 y | x p6 y

【0599】

1. ネストされたリストのPreBurstおよびPostBurstオプションは親と同じように指定されるので、パターン p 2 は p 1 と同じバーストの中にある。これらのオプションは親と同じように継承されるので、パターン p 3 も同じバーストの中にある。これらのオプションは、残りのネストされたリスト内に少なくとも1つの異なるメンバを有し、バースト不連続部を生じる。

【0600】

[タイミング]

ユーザは主に、パターンファイルを用いて試験設定を定義することにより、システムとやりとりする。Timing Fileは、これらのパターンのTimingを記述するために用いられる。このファイルは、分解されるべき基礎的な定義のための他のシステムファイル（たとえば、Pin、SpecSelector）を必要とする。さらに、Timing定義において用いられる種々の変数を分解するために用いられるSpec-SelectorおよびGlobal定義は、複合TestCondition Groupオブジェクトにカプセル化される。Test Planファイルのような、さらに上位のファイルはさらに、このTestConditionGroupインスタンスを使用する。

【0601】

Test Plan Fileは、TestConditionGroupオブジェクトへの参照を含む。Pattern Source Fileは、TimingMapオブジェクト内のWaveformSelectorコンポーネントを参照する。TimingオブジェクトそのものはPinオブジェクトを参照する。オプションでは、Timingオブジェクトは、SpecSelectorオブジェクトによって変更される変数も参照する場合がある。

これらの関係が図9に示される。

【0602】

Pattern-List内のPatternオブジェクトは、1組のパターン文字のために用いるためのWaveformSelectorオブジェクトの名前を指定する。Timing Mapファイルがそのパターンにおいて指定されることにも留意されたい。このマップが変更されない場合には、パターンはコンパイルされる必要はない。

【0603】

```
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;}
        }
    }
}

*Here we define the
WaveformSelector
component of the TimingMap
to use for the pattern
characters.
```

【0604】

TestConditionGroup Fileオブジェクトは、使用するためのTimingオブジェクトおよび使用するためのTimingMapオブジェクトをインポートする。各Testは、そのインスタンスのためのTestConditionGroupオブジェクトから導出されるTimingConditionインスタンスを使用する。こうして、同じ1組の波形テーブルをサポートする多数のTimingオブジェクトがテストフレームワーク内に格納されることができ、必要に応じてスワップされることができる。同様に、多数のTest Plan Fileが、共通のTestConditionGroupオブジェクトを共有することができる。

【0605】

Test Plan記述ファイルの一例が、以下のTimingオブジェクトの使用を例示する。

【0606】

```
Import patlist1.plist;
Importtim1.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;
}
TestFunctionalTest MyFunctionalTestFast
{
    PListParam = patList1;
    TestConditionParam = tim2_prod_max;
}

```

**Two Tests within a Test Plan
 using different timing objects
 defined earlier

【 0 6 0 7 】

Timingオブジェクトはピン毎に種々の波形を定義する。TimingファイルおよびTiming Mapファイルにおいて用いられるピンは、P i n定義ファイルにおいて適当に定義される必要がある。

【 0 6 0 8 】

TimingオブジェクトはSpecificationSetオブジェクトを用いて、波形オブジェクト内の値を定義することができる。Timingオブジェクトは種々の属性のためのハードコード化された値を含むことができるが、通常は、ユーザが、変数を用いて種々の属性に値を設定するというのが実情である。これらの変数はさらに、SpecificationSetオブジェクトにも依存することができる。この使用法の一例が以下に示される。

【0609】

```
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; } } };
        };
    };
}

**This variable, which defines
the edge placement, is
defined elsewhere and is
dependent on a
SpecificationSet.
```

【0610】

SpecSelectorは以下に示されるように定義される。

【0611】

```
SpecificationSet prodTmgSpec( min, max, typ)
{
    t_le = 10ns, 14ns, 12ns;
    t_te = 30ns, 34ns, 32ns;
    ...
}
```

【0612】

specを変更することによって用いられるタイミングの変化が以下の例に示される。

【0613】

```
TestCondition prodTmp_typ
{
    TestConditionGroup = prodTmgSpec;
    SpecSelector = typ;
}
```

```
TestConditionGroup prodTmp_max
```

```

{
  TestConditionGroup = prodTmgSpec;
  SpecSelector = max;
};

```

*This timing uses the
typical specification in the
SpecSelector.

**This timing uses the
max specification in the
SpecSelector.

10

【0614】

[F2. テスタのタイミングコンポーネントへのマッピング]

テストモジュールの2つのコンポーネントが、波形の生成およびその関連するタイミングに直に関与する。2つのモジュールは、パターン発生器(PG)およびフレームプロセッサ(FP)である。オープンアーキテクチャ試験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ティングおよびタイミング発生を例示する簡略化されたブロック図が図10に示される。波形の発生の簡単な説明が以下に与えられる。

【0615】

20

パターン発生器1002は、モジュール内の全てのピンに共通であるタイミングセットを生成する。このタイミングセットはGlobal Timing Set(GTS)と呼ばれる。パターン発生器を設定することができる3つのモードがある。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を及ぼす。さらに、これらの設定は、バンクを選択するために用いられるビットの数、およびCapture This Vector(CTV)およびMask This Vector(MTV)がセットされるか否かにも影響を及ぼす。テストにこのベクトルの結果を収集する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、テストに現在のベクトルの結果をマスクするように指示するために、ユーザはそのパターンにおいてMTVフラグを用いる。これが以下の表1に示される。パターン発生器1002は、Waveform Character(WFC)を生成する責任も担う。WFCはピン毎に生成される。テストモジュールは一定のビット数を用いて、WFCを記述する。

30

【0616】

【表1】

GTSビット	バンクにおける GTS	GTSバンク	CTV	MTV
8ビット	256	4	無	無
7ビット	128	8	有	無
6ビット	64	16	有	有

40

【0617】

テストモジュールはピン毎に1つのフレームプロセッサ1004を設ける。各フレームプロセッサは、Timing Set Scrambler(TSS)1006を含み、それは、この例では、1024までの全深を有する。TSS1006は、先に説明され、図10に示されるパターン発生器のモードに応じて、多数のバンク1008に分割されることができる。ここでは、バンク当たり64エントリの16バンクが用いられている。TSSは、ピン毎にWaveform Tableを定義する能力において、より高い自由度を与えることができるように設けられる。「FP」モードでは、TSSは、2ビットを用いてTiming Setを出力する。こうして、TSSは、ピン当たり全部で4つの個別の物理的なTiming Setを生成するであろう。これ

50

らのTiming Setは、Local Timing Set(LTS)と呼ばれる。

【0618】

フレームプロセッサ1004はLTSおよびWFCを組み合わせて、Waveform Memory 1012およびTiming Memory 1014へのインデックス1010を作成する。「FP」モードでは、5ビット値が、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ビットに分割される。こうして、最大4つの物理的なTiming Setを用いることができるが、物理的なWaveform MemoryおよびTiming Memoryの深さはピン当たり32の深さである。Waveform Memoryは、波形を形成する使用可能なタイミングエッジを含む。使用可能なエッジのためのタイミング値はTiming Memoryから得られる。こうして、フレームプロセッサは波形をフォーマットする。

10

【0619】

[マッピング方法]

その方法は、ピン毎の全てのWaveformTableブロックをテスト内のLTSにマッピングすることである。テストハードウェアが4LTSをサポートする場合には、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、テストデジタルモジュールのために最大n個の波形定義を有することができる。

【0620】

Timing-Mapファイルは、Timing-Mapブロックにおいて定義されるLogical WaveformSelectorの、オープンアーキテクチャ試験システム内のモジュールのためのWaveformTableへのマッピングを提供する。この場合、テストは256までのLogical WaveformSelectorをサポートする。オープンアーキテクチャ試験システムでは、Logical WaveformSelectorはGTSに直にマッピングする。パターンコンパイラは、Timing-MapおよびTimingブロックの両方に基づいて、パターンファイルをコンパイルすることができる。しかしながら、TimingブロックのWaveformTable内の波形文字が変更されないか、またはTiming-Mapブロック内のWaveformSelectorマッピングが変更されない場合には、そのパターンをコンパイルし直す必要はない。

20

【0621】

[このマッピング方法を用いる例]

テストDigital Moduleへのマッピングを例示するために、以下のことが仮定される。フレームプロセッサはFPモードにセットされる。CTVおよびMTVビットは、GTSビットの全数が6であり、Timing Bank Selectorビットの総数が4であるようにセットされる。

30

【0622】

Timingブロックにおいて定義される各WaveformTableは、Timingファイル内の個別のLTSにマッピングされる。これはピン毎に行われる。こうして、WaveformTable seq1はLTS1にマッピングされる。「SIG」ピンの場合には、全部で8個の取り得る波形エントリを使い果たす。しかしながら、「CLK」ピンは、ただ1つの波形エントリを必要とし、それゆえ、Waveformメモリ(WFT)およびWaveform Timingメモリ(WTM)内のただ1つの行を使い果たす。

【0623】

「SIG」ピンの最初の2つの物理的な波形のマッピングが図11に示される。このWaveformTableはエッジの個別の構成を必要とする2つの波形文字をマッピングするので、Waveformメモリ(WFT)1112およびWaveform Timingメモリ(WTM)1114において2つのエントリを割り当てることにする。波形の形状はWFMに格納され、タイミングの詳細はWTMに格納される。そのモジュールの一実施形態は、T1、T2、T3、T4、T5およびT6の全部で6個のタイミングエッジを有する。これらは、TimingブロックのEdge Resourceセクション内の波形において定義されるEvent E1、E2、...に直にマッピングする。7個以上のイベントがTimingブロックにおいて定義され、上記のモジュールとともに用いられる場合には、エラーが生じるであろう。

40

【0624】

50

図 11 の例では、第 1 の波形文字「0」はTiming Edge T1を用いて、「Force Down」または「D」イベントをプログラミングし、それはそのサイクル内の10 nsにおいて生じる。Timing Edge T2も、時刻30 nsにおいて「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、Timing Edge T3は、時刻45 nsにおいて「Force Off」または「Z」イベントを生成するために用いられる。

【0625】

第2の波形文字「1」はTiming Edge T1を用いて、「Force Up」または「U」イベントをプログラミングし、それはそのサイクル内の10 nsにおいて生じる。Timing Edge T2も、時刻30 nsにおいて「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、Timing Edge T3は、時刻45 nsにおいて「Force Off」または「Z」イ

10

【0626】

このようにして、WFCは、フレームプロセッサのWFMメモリおよびWTMメモリにマッピングされる。ピン「SIG」のためのLTS1のWaveform Memory WFMの最後の設定が以下の表2に示される。

【0627】

【表2】

I n d e x	(W F C)	T1 S e t	T1 R e S e t	T2 S e t	T2 R e S e t	T2 D r e l	T2 D r e t	E X P H	E X P H Z	T3 S e t	T3 R e S e t	T3 D r e l	T3 D r e t	T4 D r e l	T4 D r e t	E X P L	E X P H Z
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									

20

【0628】

ピン「SIG」のためのLTS1のWaveform Timing Memory WTMの最後の設定が以下の表3に示される。

30

【0629】

【表3】

I n d e x	(W F C)	T 1	T 2	E X P H	T 3	T 4	E X P L
0	0	1 0 n s	3 0 n s		4 5 n s		
1	1	1 0 n s	3 0 n s		4 5 n s		
2	d	1 2 n s	3 2 n s		4 2 n s		
3	u	1 2 n s	3 2 n s		4 2 n s		
4	L						1 7 n s
5	H			1 7 n s			
6	m						1 5 n s
7	n			1 5 n s			

40

【0630】

「CLK」ピンはただ1つの波形を使い果たすので、このピンのためのWFMおよびWFTは非常に簡単である。「CLK」ピンのためのLTS1のWaveform Memory WFMの最後の設定が以下の表4に示される。

【0631】

【表 4】

I n d e x	(W F C)	T1 S e t	T1 R e S e t	T2 S e t	T2 R e S e t	T2 D r e l	T2 D r e t	EX PH	EX PH Z	T3 S e t	T3 R e S e t	T3 D r e l	T3 D r e t	T4 D r e l	T4 D r e t	EX PL	EX PH Z
0	1	1			1												
1																	
2																	
3																	
4																	
5																	
6																	
7																	

10

【0 6 3 2】

L T S 2 のWaveform Timing Memory WTMの最後の設定が以下の表 5 に示される。

【0 6 3 3】

【表 5】

I n d e x	(W F C)	T1	T2	EX PH	T3	T4	EX PL
0	1	2 0 n s	4 0 n s				
1							
2							
3							
4							
5							
6							
7							

20

【0 6 3 4】

TimingMapブロックは、WaveformSelectorをTimingブロックのWaveformテーブルに明示的にマッピングする。或るテストシステムの場合、これは要するに、Timing Set Scrambler(TSS)メモリを設定することになる。T S Sは基本的には、それらの設定を保持する、G T SからL T Sへのマッピングを含む。ピンS I Gのための本発明の例のためのT S S設定は、以下の表 6 のようになるであろう。

【0 6 3 5】

【表 6】

G T S	L T S
0 (w f s 1)	1
1 (w f s 2)	1
2 (w f s 3)	2
3 (w f s 4)	1
4 (w f s 5)	3
5 (w f s 6)	1
N (w f s 1)	1
2 5 5	

40

【0 6 3 6】

最後に、T S SおよびL T S設定マッピングが分解された後に、Pattern Compilerはこの情報を用いて、用いるための正確な波形テーブル（L T S）および正確な波形文字を用いてパターンをプログラミングすることができる。こうして、本発明の例による、ピン「S I G」だけを考慮した擬似パターンが図 1 1 に示される。このコンパイルはTimingブロックに依存せず、Timing-Mapブロックだけに依存することに留意されたい。

【0 6 3 7】

[G. テスタ動作]

50

このセクションはテストオペレーティングシステム（T O S）の基本動作を記述する。
このセクションにおいて考えられる動作は以下のものである。

- ・システム初期化
- ・Test Planローディング
- ・パターンローディング
- ・Test Planの実行
- ・個々のTestの実行

【0638】

〔システム初期化〕

一実施形態ではシステムを初期化するために、ある特定の仮定が満たされなければならない。以下のサブセクションがこれらを記載する。

10

【0639】

〔事前条件〕

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が中央に格納され、その場所がシステムコントローラに知られている。これは、システムコントローラそのものに存在することができるか、またはネットワーク実装ディレクトリを有する（または別の仕組みを介してSYSCに知られている）他のシステム上に存在することができ、どのような仕組みであっても、システムが機能することができる前に、システムコントローラが使用するために、全てのソフトウェアが入手できなければならない。このソフトウェアは以下のものを含む。

20

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

標準またはユーザ試験クラスDLL

ユーザ試験計画DLL

【0640】

システムモジュール構成ファイルはシステムコントローラ上で入手することができる。このファイルによって、ユーザがテストの物理的な構成、たとえばシステムシャーシ内の各モジュールの物理的な場所およびタイプ、ならびにモジュールソフトウェアDLLの名前を指定できるようになることを思い起こされたい。

【0641】

30

システム構成ファイルはシステムコントローラ上で入手することができる。このファイルはシステム内のサイトコントローラのリスト、およびサイトコントローラホスト名のSwitch Matrix入力ポートアドレスへのマップを含むことを思い起こされたい。

【0642】

サイトコントローラは、Site Configuration Manager(SCM)と呼ばれるサービスを実行する。このサービスは、「ハードウェア発見」と呼ばれるプロセスによって、どんなハードウェアが各スロットにインストールされるかを判定する責任を担う。またそれは、システムコントローラでのシステム初期化プロセスに関与する責任も担う。一実施形態では、Switch Matrix動作プロトコルは、Switch Matrix入力ポート接続アドレス1とともに、ただ1つのSite Controller上のSCMを常に用いて、モジュールに対するSwitch Matrix接続を構成すべきであることに留意されたい。この「特別な」サイトはSITEC-1として表されることを思い起こされたい。

40

【0643】

システムコントローラは、各サイトコントローラのSCMに、そのSwitch Matrix接続アドレスを提供する責任を担う。

【0644】

各サイトコントローラのSCMは、Test Plan Server(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Test Plan Serverは最終的に、ユーザの1つの試験計画（または、1つのサイトコントローラが多数のDUT上で試験を実行している場合には、複数の試験計画）を保持し、実行する責任を担う。

50

【0645】

〔初期化段階Ⅰ：システム妥当性検査〕

一旦、上記の仮定および事前条件が満たされたなら、システム初期化は最初に、以下のようなシステム妥当性検査ステップを開始する。

1. システムコントローラが、システムおよびモジュール構成ファイルを読み出し、システムのユーザ指定の意図を初期化する。

2. 指定されたシステム構成情報を用いて、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動作中であり、接触可能であり、しかも準備できている（すなわち、SCMを実行している）ことを確認する。この確認ステップ中にエラーがあれば、システムエラーが引き起こされて、初期化は中止されるであろう。

10

3. その後、システムコントローラは、SITE C-1にあるSCMサービスに、スイッチマトリックスを構成して、全てのハードウェアモジュールにアクセスできるようにすることを指示し、ハードウェア発見を実行するように要求する。

4. SITE C-1におけるSCMサービスは、{vendor, hardware}タプルのための全ての利用可能なモジュールスロット（既知のハードウェアの場所）をポーリングし、{vendor, hardware}タプルのスロットへのマッピングを生成する。したがって、終了時に、このポーリングは、完全なシステム内に存在する完全な1組の{vendor, hardware, slot}結合を特定している。このポーリングの結果はシステムコントローラに送られる。

5. システムコントローラは、上記のハードウェア発見ステップの結果が、モジュール構成ファイル内のユーザ指定の構成と一致することを確認する。この確認ステップ中に何らかのエラーがあれば、システムエラーが引き起こされ、初期化は中止されるであろう。

20

6. その後、システムコントローラは、周知の場所（複数の場合もあり）において環境設定ファイル（複数の場合もあり）からデフォルト環境（モジュールDLLを探すための探索経路、パターンリスト、パターン、試験計画DLL、試験クラスDLLなど）をロードする。

7. システムコントローラは、全ての特定されたモジュールソフトウェアDLLが存在することを確実にする。システムコントローラ上で入手できない場合には、可能であるなら、中央記憶装置から検索される。そうでない場合には、システムエラーが引き起こされ、初期化が中止される。

30

【0646】

〔初期化段階Ⅱ：サイト構成（オプション）〕

サイト構成、すなわちサイト分割は、利用可能なシステムハードウェアモジュールを種々のサイトに（すなわち、多数のDUTにサービスするために）ソフトウェアレベルで割り当てることを含む。サイト分割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起こされたい。

【0647】

テストシステムによって、試験計画ロードの一部として（各試験計画は特定のソケットに関連するため）、かつ個別のユーザコール可能なステップとして、サイト（再）分割が実行されるようになる。後者の場合、ユーザは、システムを分割するためにのみ用いられるソケットファイルを提供することにより、サイト分割を開始する。これは、各サイトが異なるDUTタイプを試験するマルチDUT試験の場合のシステム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、初期化段階ではオプションであり、ユーザはそれを実行しないように選択し、代わりに、試験計画ロードがシステムを適当に分割できるようにすることを選択することができる。

40

【0648】

サイト分割を達成するためにどの手段が選択されるにしても（個別のコールによるか、または試験計画ロードを通して暗黙的に）、その仕組みは同じである。この仕組みが以下に記載される。

1. ソケットが与えられるとき、システムコントローラは最初に、現時点で存在しているシステム分割がソケットに適合するか否か、または再分割が必要であるか否かを判定す

50

る。初期化中のデフォルト分割は、全ての利用可能なモジュールがSITEC-1に接続される分割である。以下の残りのステップは、再分割が必要とされる場合のみ実行される。

2. システムコントローラは各サイトコントローラSCMに構成メッセージを送り、新たなソケットの下でそのために使用可能にされるDUTサイトの数および識別でそのものを再構成する。これは一般的な手順であり、1つの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1つである場合を取り扱うことに留意されたい。新たなソケット情報はSCMにも伝達される。

3. 各SCMは、もしあれば、実行中のTPSを停止して、新たなTPSを開始し、新たなソケットと、新たなソケットの下でそのために使用可能にされるDUTサイトの数および識別とを用いてそれを初期化する。

10

4. システムコントローラは、どのサイトが、必要とされるシステムモジュールのどのサブセットを必要とするかを判定する。これを行いながら、システムコントローラはサイトのためのハードウェアスロット情報も準備する。サイト毎の最終的な結果は、そのサイトに割り当てられるモジュールDLLに対するスロットのリストである。このサイト特有のリストは、Site Module DLL Slot List (SITE-MDSL) として表されるであろう。

5. システムコントローラは適当なSITE-MDSL、および必要なモジュールDLLを各SCMに提供する。各SCMはさらに、新たに開始されたTPSがこの情報を入手できるようにする。

6. その後、システムコントローラは、適当なサイト-スロット間接続のために、すなわち、サイト分割作業のために、SITEC-1にSwitch Matrixを構成するように要求する。

20

7. サイト1～n上のTPSは、そのSITE-MDSLにおいて指定されるDLLをロードする。これらの各DLLは、initialize()という名前の関数を有し、その関数はスロット番号のアレイを取得する。TPSは、そのモジュールタイプのための適当なスロットリストでinitialize()をコールする。この時点で誤動作があれば、システムエラーが引き起こされて、初期化が中止される。initialize()メソッドは以下のことを行う。

a. 標準インターフェースIXXXModuleに基づいて具体的なクラスを作成する。たとえば、或るデジタルモジュールに関連するDLLは、それが関連する各スロットにサービスするために、1つのIPinModuleベースオブジェクトを作成するであろう。

b. インターフェースIResourceに基づいて、そのモジュール内の「資源ユニット」毎に1つの複数の具体的なクラスを作成する。再び、或るデジタルモジュールの場合、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有されるスロットのコレクション内の全てのピンのためのITesterPinベースオブジェクトを作成するであろう。

30

8. その後、サイト1～n上のTPSは、ロードされた各モジュールDLLにおいてgetXXXModule()をコールし、モジュール内容情報を検索する。

9. getXXXModule()へのコールの度に、IModuleポインタとして<VendorHWType>Moduleクラスオブジェクトが返される（たとえば、AdvantestPinModule）。そのような各IModuleポインタはTPSによってキャッシュされ、TPSは、フレームワーク/ユーザコードがこれらを手に入れるようにする。IModule、IResourceなどのコレクションは永続的である（少なくともTPSの寿命にわたって）ことに留意されたい。

40

10. 一旦、上記のステップが完了したなら、TPSは、その割り当てられた（既知の）ポートにおいて、listen()を開始する。これは、TPSが標準（すなわちサイト分割された）動作を開始する「準備ができています」ことを、システムコントローラに伝える。

【0649】

〔試験計画ロード〕

このセクションは、ユーザTest Plan DLLがサイトコントローラ上にロードされる（1つまたは複数のDUT試験のために）ステップを記述する。

【0650】

一旦、システム初期化（およびオプションで初期サイト分割）が完了したなら、ユーザ試験計画をロードすることができる。ユーザ試験計画のサイトコントローラ上へのローデ

50

ィングは、以下のように進められる。

1. システムコントローラは最初に、試験計画DLLをその自らの処理空間内にロードし、それに、関連するソケットファイルおよびDUTタイプ識別子を問い合わせる。この情報を用いて、この試験計画を実行するサイト（複数の場合もあり）、それゆえこの試験計画がロードされることになるサイトコントローラ（複数の場合もあり）が特定される。

2. その後、システムコントローラは、その試験計画に関連するソケット情報を用いて、先に略述されたような再分割プロセスを開始する。

3. システムコントローラは、試験計画DLLから、試験計画によって用いられる試験クラスDLLのリストを抽出し、一旦、システムコントローラが、TPSが標準（すなわち、サイト分割された）動作を開始する準備ができていることを確認したなら、試験クラスDLLを、そして最後に、試験Plan DLLそのものを適当なTPSに送出する。

4. TPSはLoadLibrary()をコールして、それを処理空間にロードする。それは、DLLにおいて周知の関数をコールして、それがサービスするサイト（すなわちDUT）の数と同じ数の試験計画オブジェクトを作成する。

5. TPSは、必要なテストフレームワークオブジェクトで試験計画オブジェクト（複数の場合もあり）を初期化する。初期化中に、TPSは、試験計画オブジェクト（複数の場合もあり）によって用いられる試験クラスに適したDLLを処理空間にロードし、試験クラスインスタンスを作成する。

6. TPSはシステムコントローラに対する通信チャネルを試験計画オブジェクト（複数の場合もあり）に設定する。

7. システムコントローラはTPSと通信し、試験計画オブジェクト（複数の場合もあり）のためのプロキシを構築する。

【0651】

その結果として、サイトコントローラ上のユーザの試験計画のロードに成功する。

【0652】

〔試験計画の実行〕

予め定義されたフローロジックにしたがって試験計画内の全ての試験を実行するための方法は以下のとおりである。

1. ユーザのアプリケーションがRunTestPlanメッセージをTPSに送信する。TPSはExecutingTestPlanメッセージを全ての接続されたアプリケーションに送る。その後、TPSはTest Plan上でexecute()をコールする。

2. 1つのサイトコントローラで多数のDUTを試験することは、そのサイトコントローラ上で、DUT当たり1つの多数のスレッドを用いて実行される。各スレッドは、同じ試験計画オブジェクトの異なる個別のインスタンスを実行する。この場合には、モジュール制御ソフトウェアDLLを複数のDUTにわたって共有することができるので、DUT識別子パラメータを取得するために、ハードウェア通信のためのモジュールコマンドが必要とされる。

3. 試験計画オブジェクトは、そのコレクション内の各試験にわたって繰り返され（別法では、そのFlowオブジェクトに、フローロジックにしたがって各試験を処理するように指示し）、preExec()、execute()およびpostExec()をコールする。

4. 各試験が実行されるとき、全ての接続されるアプリケーションに対してステータスメッセージが返送される。

【0653】

〔単一の試験の実行〕

ユーザは、全ての試験ではなく、1つの試験計画内のただ1つの試験を実行したい場合もある。ただ1つの試験を実行する場合、その方法は以下のとおりである。

1. ユーザアプリケーションがRunTestメッセージをTPSに送信する。TPSはExecutingTestメッセージを全ての接続されるアプリケーションに送る。その後、TPSは、Test PlanにおいてexecuteTest()をコールし、実行するための試験を指定する。

2. Test Planオブジェクトは、その試験オブジェクトにおいてpreExec()、execute()

10

20

30

40

50

およびpostExec()をコールすることにより、指定された試験を実行する。

3. その試験が実行されるとき、それは全ての接続されるアプリケーションにステータスメッセージを返送する。

【0654】

本発明が特定の実施形態とともに説明されてきたが、本発明の精神および範囲から逸脱することなく、当業者によって種々の変更形態および代替形態が実施できることは理解されたい。本発明は上記の例示的な詳細によって制限されるべきでなく、特許請求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】

【0655】

10

【図1】従来のテストアーキテクチャを示す図である。

【図2】本発明の一実施形態による、テストアーキテクチャを示す図である。

【図3】本発明の一実施形態による、テストソフトウェアアーキテクチャを示す図である。

。

【図4】本発明の一実施形態による、試験プログラムコンパイラを示す図である。

【図5】本発明の一実施形態による、単一の試験クラスから種々の試験インスタンスを如何に導出することができるかを示す図である。

【図6】本発明の一実施形態による、パターンコンパイラを示す図である。

【図7】本発明の一実施形態による、順序パターン木の例を示す図である。

【図8】本発明の一実施形態による、別の順序パターン木の例を示す図である。

20

【図9】本発明の一実施形態による、試験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す図である。

【図10】本発明の一実施形態による、波形発生を示す図である。

【図11】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す図である。

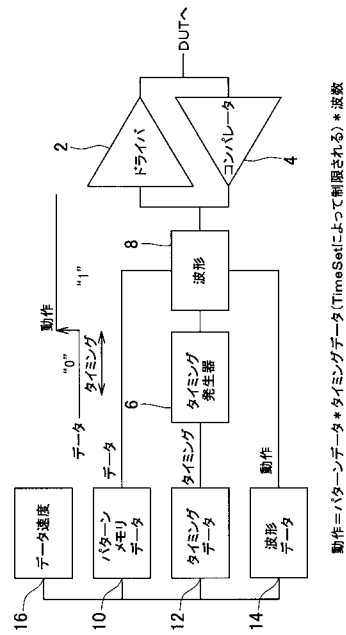
【図12】本発明の一実施形態による、タイミングのために用いられる別のマッピングを示す図である。

【図13】本発明の一実施形態による、モジュール式試験システムにおいてパターンオブジェクトファイルを管理するためのOFMフレームワークを示す図である。

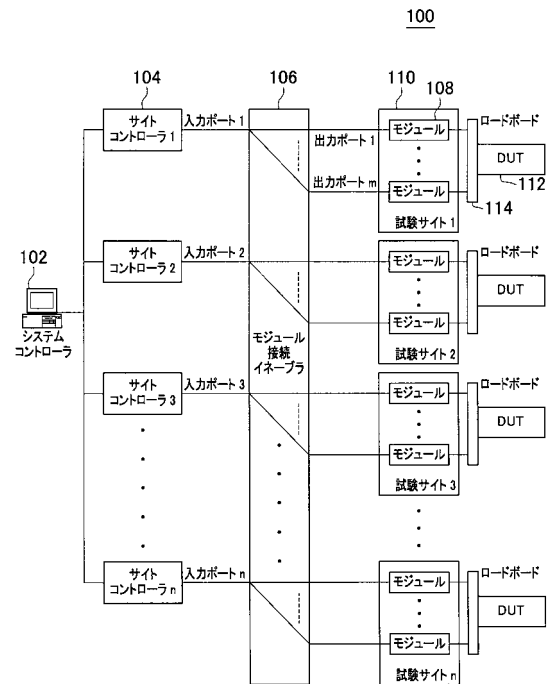
【図14】パターンオブジェクトメタファイルとそのサポート用クラスとの間のインタラクションを示す図である。

30

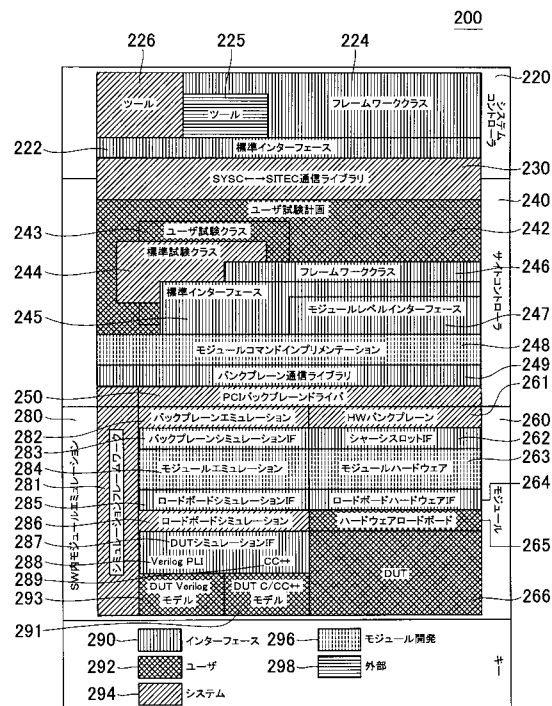
【図 1】



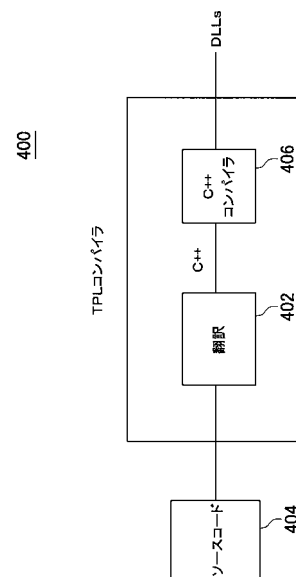
【図 2】



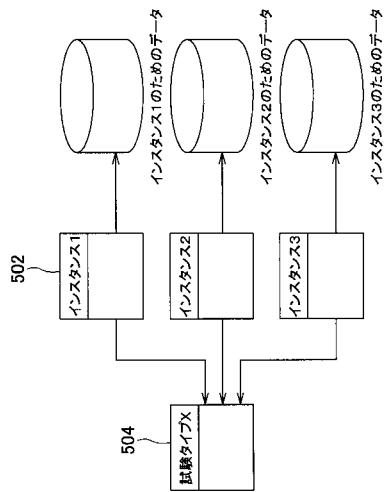
【図 3】



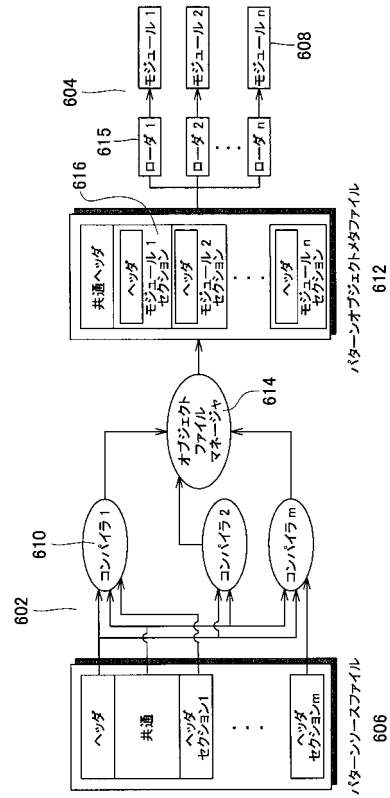
【図 4】



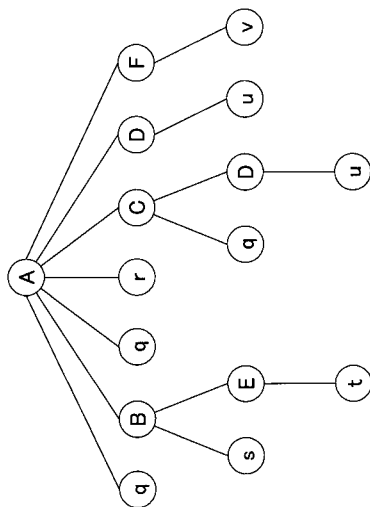
【図 5】



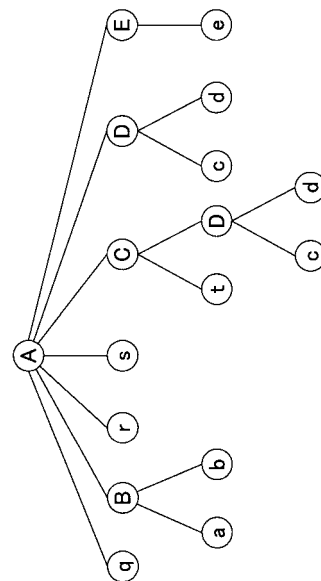
【図 6】



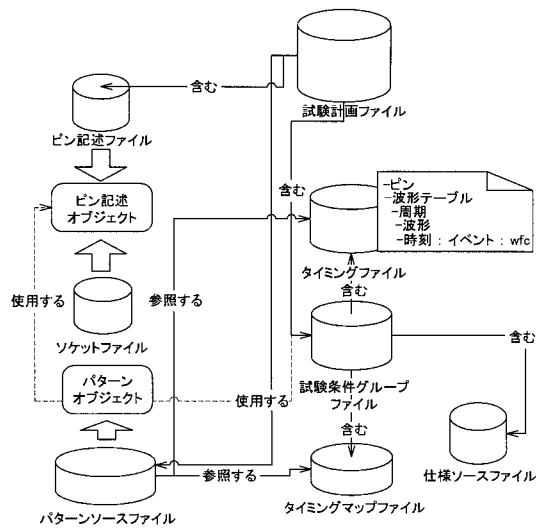
【図 7】



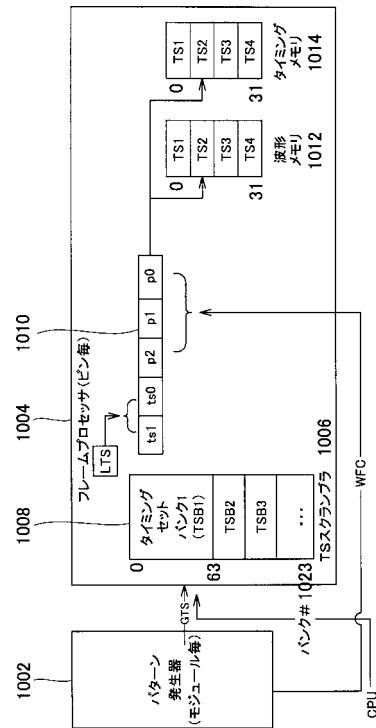
【図 8】



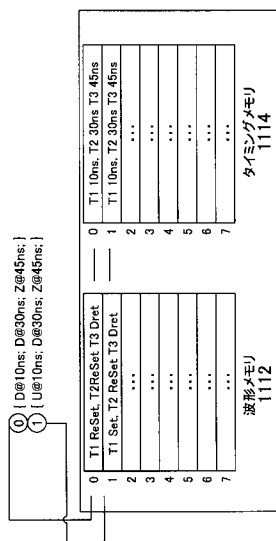
【図 9】



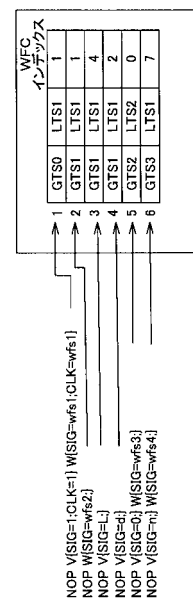
【図 10】



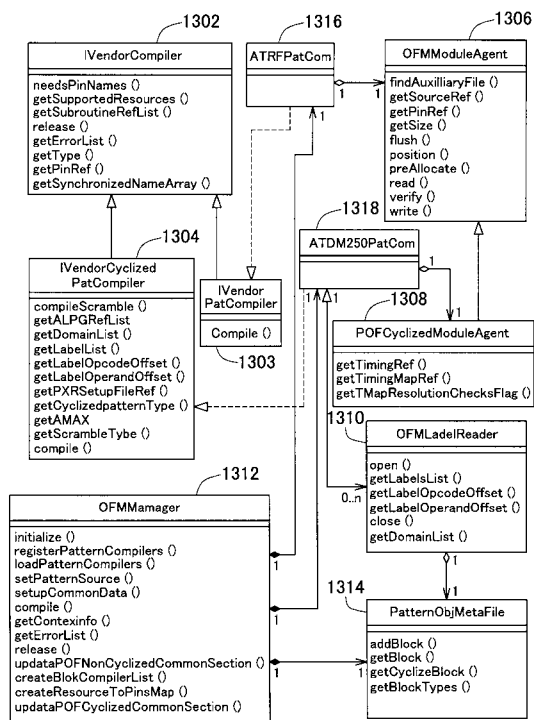
【図 1 1】



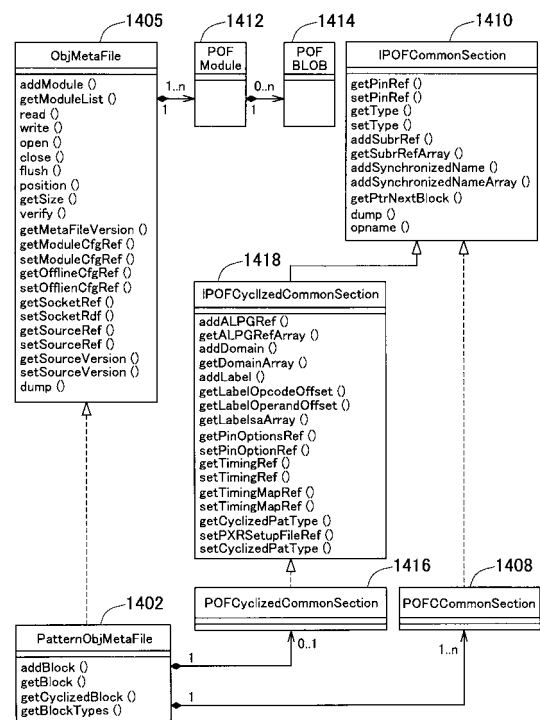
【图 1 2】



【図 13】



【図 14】



フロントページの続き

- (72)発明者 シング、ハーサンジート
アメリカ合衆国、94526 カリフォルニア州、ダンビル、バルセタ コート 100
- (72)発明者 プラマニック、アンカン
アメリカ合衆国、95318 カリフォルニア州、サンノゼ、カプリアーノ ドライブ 5839
- (72)発明者 エルストン、マーク
アメリカ合衆国、93906 カリフォルニア州、サリーナス、メルロット ウェイ 1777
- (72)発明者 田原 善文
アメリカ合衆国、94087 カリフォルニア州、サニーバール、サニーマウント アベニュー 587
- (72)発明者 足立 敏明
アメリカ合衆国、95120 カリフォルニア州、サンノゼ、ニキット ウェイ 1196

審査官 神谷 健一

- (56)参考文献 特開平4-180127 (JP, A)
特開平5-61657 (JP, A)
特開平9-26873 (JP, A)
特開平9-160943 (JP, A)
特開平11-134198 (JP, A)
特開2001-67395 (JP, A)
特開2003-15876 (JP, A)
米国特許出願公開第2003/0217343 (US, A1)
PEREZ S M, FURUKAWA Y, Open architecture test system: the new frontier, IEEE/CPMT/SEMI 28th International Electronics Manufacturing Technology Symposium, 2003. IEMT 2003., 米国, IEEE, 2003年 7月16日, 211 - 214, ISBN: 0-7803-7933-0
RAJSUMAN R, An overview of the open architecture test system, International Workshop on Electronic Design, Test and Applications, 2004. DELTA 2004. Second IEEE, 米国, IEEE, 2004年 1月28日, 341 - 346, ISBN: 0-7695-2081-2
SIVARAM A.T, FAN D, PRYCE J, Xml and java for open ate programming environment, International Test Conference, 2003. Proceedings. ITC 2003., 米国, IEEE, 2003年 9月30日, Vol. 1, 793 - 801, ISBN: 0-7803-8106-8
Pramanick, A. Krishnaswamy, R. Elston, M. Adachi, T. Harsanjeet Singh Parnas, B. Chen, L, Test programming environment in a modular, open architecture test system, Test Conference, 2004. Proceedings. ITC 2004. International, 米国, IEEE, 2004年10月26日, 413 - 422, ISBN: 0-7803-8580-2
PARNAS B R, PRAMANICK A, ELSTON M, ADACHI T, Software development for an open architecture test system, Ninth IEEE European Test Symposium (ETS 2004), 米国, IEEE, 2004年 5月23日, ISBN: 0-7695-2119-3

- (58)調査した分野(Int.Cl., DB名)
G01R 31/28-31/3193
G06F 17/50
IEEE Xplore