(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5039130号
(P5039130)
(45)発行日　平成24年10月3日(2012.10.3)　(24)登録日　平成24年7月13日(2012.7.13)

(51)Int.Cl.　F I
G06F 17/50 (2006.01)　G06F 17/50 672Z
G06F 17/50 664Z

請求項の数 6　(全 25 頁)

(21)出願番号　特願2009-508341 (P2009-508341)
(86)(22)出願日　平成19年5月2日(2007.5.2)
(65)公表番号　特表2009-535726 (P2009-535726A)
(43)公表日　平成21年10月1日(2009.10.1)
(86)国際出願番号　PCT/EP2007/054255
(87)国際公開番号　WO2007/128753
(87)国際公開日　平成19年11月15日(2007.11.15)
審査請求日　平成22年2月18日(2010.2.18)
(31)優先権主張番号　11/381,437
(32)優先日　平成18年5月3日(2006.5.3)
(33)優先権主張国　米国(US)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】シミュレーション結果を表示するための信号の仕様をサポートする方法、システム、およびプログラム製品

(57)【特許請求の範囲】
【請求項１】
データ処理システムにおけるデータ処理の方法であって、データ処理システムが、
信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットを入力として受け取るステップと、
前記データ・セットの受け取りに応答して、
前記データ・セット内の前記少なくとも１つのエントリを処理して、前記信号グループ名を識別するステップと、
前記識別された信号グループ名に対応する信号グループに含まれる信号の信号名を、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グループ情報から決定するステップと、
前記複数の信号のインスタンスに関連付けられた前記イベント・トレース・ファイルからの１以上のシミュレーション結果を、プレゼンテーションに含めるステップと、
リネームから関連する信号名が保護されることになる特定の信号を識別する保護指示を受け取るステップと、
前記保護指示に応答して、前記プレゼンテーションにおいて前記関連する信号名によって前記信号を識別するステップと
を実行することを含む、前記方法。
【請求項２】
前記信号グループが第１の信号グループを備えており、

前記第１の信号グループが、前記複数の信号をメンバとして集合的に有する第２および第３の信号グループを備えている、
請求項１に記載の方法。
【請求項３】
前記データ・セットは、信号インスタンス化識別子、信号グループ・インスタンス化識別子、及び、参照有効範囲コマンドを含み、
前記データ処理システムが、
前記データ・セット内の参照有効範囲コマンドを解釈して、前記インスタンス化識別子又は信号グループ・インスタンス化識別子の中からユーザに提示する識別子を選択するステップ
をさらに実行することを含む、請求項１又は２に記載の方法。
【請求項４】
データ処理システムに、請求項１～３のいずれか一項に記載の方法の各ステップを実行させるコンピュータ・プログラム。
【請求項５】
データ処理システムに、請求項１～３のいずれか一項に記載の方法の各ステップを実行させるコンピュータ・プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項６】
データ処理システムであって、
プロセッサと、
前記プロセッサに結合されたデータ・ストレージと
を備えており、
前記データ・ストレージが、
前記データ処理システムに、請求項１～３のいずれか一項に記載の方法の各ステップを実行させるプログラム・コード
を含む、前記データ処理システム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般に、デジタル・デバイス、モジュール、およびシステムのシミュレートに関し、具体的には、デジタル・デバイス、モジュール、およびシステムのコンピュータ・シミュレーションに関する。
【背景技術】
【０００２】
デジタル設計の論理的正確さの検証、および、必要であれば設計のデバッグは、ほとんどのデジタル設計プロセスにおいて非常に重要なステップである。論理ネットワークは、実際にネットワークを構築すること、またはコンピュータ上でネットワークをシミュレートすることの、いずれかによってテストされる。論理ネットワークが非常に複雑になるにつれて、設計が実際に構築される前に設計をシミュレートすることが必要になってきている。これは特に、集積回路の製造にかなりの時間を必要とし、ミスの訂正にかなりのコストがかかることから、設計が集積回路として実装される場合に当てはまる。デジタル設計シミュレーションの目的は、設計の論理的正確さの検証である。
【０００３】
従来の電子コンピュータ支援設計（ＥＣＡＤ）システムによってサポートされる典型的な自動設計プロセスでは、設計者は、ＶＨＤＬなどのハードウェア記述言語（ＨＤＬ）を使用して高水準の記述を入力し、様々な回路ブロックおよびそれらの相互接続の表現を生成する。ＥＣＡＤシステムは、設計記述を、シミュレーションに最適なフォーマットにコンパイルする。次にシミュレータを使用して、回路レイアウトを展開するのに先立って設計の論理的正確さを検証する。
【０００４】

シミュレータとは、典型的には、デジタル表現、または回路のシミュレーション・モデル、およびデジタル・システムの入力を表現する入力刺激（input stimuli）のリスト（すなわちテストケース）で動作する、ソフトウェア・ツールである。シミュレータは、回路の応答の数値表現を生成し、これが、値のリストとしてディスプレイ画面上に表示されるか、または、しばしば別のソフトウェア・プログラムによってさらに解釈され、グラフの形でディスプレイ画面上に提示される可能性がある。シミュレータは、シミュレーション用に特別に設計された、汎用コンピュータまたは通常は汎用コンピュータに接続された他の電子装置上で、実行可能である。全体として汎用コンピュータ上のソフトウェア内で実行するシミュレータを、以下では「ソフトウェア・シミュレータ」と呼ぶ。特別に設計された電子装置の支援を受けて実行されるシミュレータを、以下では「ハードウェア・シミュレータ」と呼ぶ。

【０００５】

通常、ソフトウェア・シミュレータは非常に多くの計算を実行し、ユーザの観点からゆっくりと動作する。性能を最適化するために、シミュレーション・モデルのフォーマットは、シミュレータが非常に効率良く使用するように設計される。ハードウェア・シミュレータは、本質的に、回路記述を備えるシミュレーション・モデルが特別に設計されたフォーマットで通信されることを必要とする。いずれの場合も、以下ではシミュレーション実行可能モデルと呼ばれる、ＨＤＬ記述からシミュレーション・フォーマットへの変換が必要である。

【０００６】

シミュレータによるシミュレーション実行可能モデルへのテストケースの適用の結果は、本明細書では「全イベント・トレース」（ＡＥＴ）と呼ばれる。ＡＥＴは、シミュレーション実行可能モデル内の信号あるいはストレージ要素またはその両方の論理値を含む。ＡＥＴビューアは、再検討および分析のためにＡＥＴのコンテンツによってユーザに提示するために使用することができる。

【０００７】

大規模なシミュレーション実行可能モデルの場合、莫大な量のデータがＡＥＴに存在することになり、それらのすべてがユーザに関係する訳ではないことを理解されよう。したがって従来のＡＥＴビューアでは、ユーザは、ユーザが表示したいシミュレーション実行可能モデル内の信号を指定する入力／出力（Ｉ／Ｏ）リストを入力することができる。これに応答して、従来のＡＥＴビューアは、Ｉ／Ｏリスト内で識別されたシミュレーション実行可能モデル内の信号のみをユーザに提示する。

【発明の開示】

【発明が解決しようとする課題】

【０００８】

本発明は、特に複雑なシミュレーション実行可能モデルの場合、Ｉ／Ｏリストの（たとえばキーボードを利用した）ユーザ入力が退屈で時間のかかるものであることを理解している。したがって本発明は、シミュレーション処理のための方法、システム、およびプログラム製品を提供する。

【課題を解決するための手段】

【０００９】

本発明は、データ処理システムにおけるデータ処理の方法であって、データ処理システムが、

信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットを入力として受け取るステップと、

前記データ・セットの受け取りに応答して、

前記データ・セット内の前記少なくとも１つのエントリを処理して、前記信号グループ名を識別するステップと、

前記識別された信号グループ名に対応する信号グループに含まれる信号の信号名を、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グルー

プ情報から決定するステップと、
前記複数の信号のインスタンスに関連付けられた前記イベント・トレース・ファイルからの１以上のシミュレーション結果を、プレゼンテーションに含めるステップと、
リネームから関連する信号名が保護されることになる特定の信号を識別する保護指示を受け取るステップと、
前記保護指示に応答して、前記プレゼンテーションにおいて前記関連する信号名によって前記信号を識別するステップと
を実行することを含む、前記方法を提供する。

【００１０】

例示的方法によれば、所定の信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットが、データ処理システムによって受け取られる。このデータ・セットの受け取りに応答して、信号グループ名を識別するためにデータ・セット内のエントリが処理される。信号グループのメンバである複数の信号の信号名を特定するために、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グループ情報にアクセスする。その後、当該複数の信号のインスタンスに関連付けられたイベント・トレース・ファイルからのシミュレーション結果が、シミュレーション結果のプレゼンテーションに含められる。

本発明のすべての目的、特徴、および利点は、以下の詳細な説明で明らかになろう。

【００１１】

本発明の新規な機能であるとみなされる特徴は、添付の特許請求の範囲に示される。しかしながら、本発明それ自体、ならびに好ましい使用モード、それらの他の目的および利点は、添付の図面に関して例示的実施形態の以下の詳細な説明を参照することによって、最も良く理解されるであろう。

【発明を実施するための最良の形態】

【００１２】

次に図面を参照すると、また特に図１を参照すると、これによって本発明が有利に使用できる、データ処理システム１０の絵画図が示されている。図に示されるように、データ処理システム１０は、１つまたは複数のノード１３が接続されたワークステーション１２を備える。ワークステーション１２は、好ましくは、ニューヨーク州アーモンクのインターナショナル・ビジネス・マシーンズ（ＩＢＭ）（登録商標）コーポレーションから入手可能なＰＯＷＥＲライン・コンピュータ・システムのうちの１つなどの、高性能マルチプロセッサ・コンピュータを備える。ワークステーション１２は、好ましくは、本発明の方法およびシステムに従ってデジタル回路設計を開発および検証するために使用可能なＥＣＡＤシステムを備えるソフトウェア・アプリケーションを格納するための、不揮発性および揮発性の内部ストレージを含む。図に示されるように、ノード１３は、ディスプレイ・デバイス１４、キーボード１６、およびマウス２０を含む。ワークステーション１２内で実行されるＥＣＡＤソフトウェア・アプリケーションは、好ましくは、デジタル回路設計者がキーボード１６およびマウス２０を使用して対話することが可能なグラフィック・ユーザ・インターフェース（ＧＵＩ）を、ディスプレイ・デバイス１４のディスプレイ画面２２内に表示する。したがって、キーボード１６およびマウス２０を利用して適切な入力を実行することによって、デジタル回路設計者は、以下でより詳細に説明する方法に従ってデジタル回路設計を開発および検証することができる。

【００１３】

図２は、データ処理システム１０を示すより詳細なブロック図である。図に示されるように、データ処理システム１０は、従来のマイクロプロセッサなどの１つまたは複数の中央処理ユニット（ＣＰＵ）、および、システム相互接続２６を介して相互接続された、いくつかの他の構成要素を含む。図２には示されていないが、ＣＰＵ　２４などのＣＰＵは、通常、コンピュータ・メモリ内のデータおよびプログラム・ストレージを編成し、このデータおよび他の情報をコンピュータ・システムの様々な部分間で転送する、制御ユニットを含む。ＣＰＵは一般に、加算、比較、乗算などの算術的および論理的演算を実行する

、１つまたは複数の演算論理ユニットも含む。

【００１４】

データ処理システム１０は、ランダム・アクセス・メモリ（ＲＡＭ）２８、読み取り専用メモリ（ＲＯＭ）３０、ディスプレイ・デバイス１４の接続をサポートするディスプレイ・アダプタ３２、および周辺デバイス（たとえばディスクおよびテープ・ドライブ３３）を接続するためのＩ／Ｏアダプタ３４を、さらに含む。データ処理システム１０は、データ処理システム１０を通信ネットワークに接続するための通信アダプタ４２と、キーボード１６、マウス２０、スピーカ３８、マイクロフォン４０、あるいは他のユーザ・インターフェース・デバイスまたはそれらすべてを、システム相互接続２６に接続するための、ユーザ・インターフェース・アダプタ３６とを、さらに含む。

【００１５】

当業者であれば理解されるように、データ処理システム１０は、ＲＡＭ　２８、ＲＯＭ　３０、磁気ディスク、磁気テープ、または光ディスク（最後の３つはディスクおよびテープ・ドライブ３３内に配置される）などの、任意の好適なコンピュータ読み取り可能媒体に常駐可能な、オペレーティング・システム（たとえばＡＩＸ）および１つまたは複数の他のプログラムの制御の下で動作する。

【００１６】

シミュレートされたデジタル回路設計モデルは、以下で設計エンティティと呼ばれる、少なくとも１つ、通常は多くの、サブユニットからなる。図３は、その内部に本発明の方法およびシステムが実装可能な、例示的設計エンティティ３００のブロック図である。設計エンティティ３００は、エンティティ名、エンティティ・ポート、および設計エンティティ３００によって実行される機能の表現という、いくつかの構成要素によって定義される。所与のモデル内の各エンティティは、各エンティティのＨＤＬ記述で宣言された固有名（図３には明示的に図示せず）を有する。さらに各エンティティは、通常、ポートと呼ばれる、エンティティ外部の信号へのいくつかの信号相互接続を含む。これらの外部信号は、全体設計内の他のエンティティに接続している全体設計または信号の主入力／出力（Ｉ／Ｏ）とすることができる。

【００１７】

通常、ポートは、入力ポート、出力ポート、および双方向ポートという、３つの別個のタイプのうちの１つに属するものと分類される。設計エンティティ３００は、設計エンティティ３００内に信号を搬送するいくつかの入力ポート３０３を有するものとして示される。入力ポート３０３は入力信号３０１に接続される。加えて、設計エンティティ３００は、設計エンティティ３００外部へ信号を搬送するいくつかの出力ポート３０６を含む。出力ポート３０６は出力信号３０４のセットに接続される。双方向ポート３０５は、設計ポート３００の内部および外部へ信号を搬送するために使用される。双方向ポート３０５は、双方向信号３０９のセットに接続される。設計エンティティ３００などのエンティティは、３つすべてのタイプのポートを含む必要がなく、悪くすると、まったくポートを含まない。エンティティ・ポートの外部信号への接続を実施するために、「ポート・マップ」と呼ばれるマッピング技法が使用される。ポート・マップ（図３には明示的に図示せず）は、エンティティ・ポート名とエンティティが接続された外部信号との間の指定された対応関係からなる。シミュレーション・モデルを構築する場合、ＥＣＡＤソフトウェアを使用し、ポート・マップ指定に従って、外部信号をエンティティの適切なポートに接続する。

【００１８】

最後に、設計エンティティ３００は、設計エンティティ３００によって実行される１つまたは複数の機能を記述した本体部分３０８を含む。デジタル設計の場合、本体部分３０８は、他のエンティティのインスタンス化に加えて、論理ゲート、ストレージ要素などの相互接続を含む。他のエンティティ内にエンティティをインスタンス化することによって、全体設計の階層記述が達成される。たとえばマイクロプロセッサは、同一機能ユニットの複数インスタンスを含む場合がある。したがって、マイクロプロセッサそれ自体は、し

ばしば単一エンティティとしてモデル化されることになる。マイクロプロセッサ・エンティティ内では、任意の重複する機能エンティティの複数のインスタンス化が存在することになる。

【００１９】

次に図４を参照すると、本発明の好ましい実施形態で使用可能な例示的シミュレーション・モデル３２９の概略図が示されている。シミュレーション・モデル３２９は、複数の階層的な設計エンティティを含む。視覚的に単純かつ明瞭にするために、シミュレーション・モデル３２９内のエンティティを相互接続するポートおよび信号の多くは明示的に示されていない。いかなるモデルにおいても、唯一のエンティティがいわゆる「最上位エンティティ」である。最上位エンティティ３２０は、シミュレーション・モデル３２９内のすべての他のエンティティを包含するエンティティである。すなわち、最上位エンティティ３２０は、設計内のすべての下位（descendant）エンティティを直接または間接的にインスタンス化する。シミュレーション・モデル３２９は、ＦＸＵエンティティ３２１の２つのインスタンスである３２１ａおよび３２１ｂを直接インスタンス化する、最上位エンティティ３２０からなる。各インスタンス化は、エンティティ名および固有のインスタンス化名を含む関連記述を有する。最上位エンティティ３２０の場合、記述３１０は「ＴＯＰ：ＴＯＰ」とラベル表示される。記述３１０は、コロンに先行する「ＴＯＰ」としてラベル表示されたエンティティ名３１２を含み、コロンに続く「ＴＯＰ」としてラベル表示されたインスタンス化名３１４も含む。

【００２０】

ＦＸＵエンティティ３２１のインスタンス化３２１ａおよび３２１ｂで示されるように、特定のエンティティを複数回インスタンス化することが可能である。インスタンス化３２１ａおよび３２１ｂは、それぞれ、インスタンス化名ＦＸＵ０およびＦＸＵ１を備えた、ＦＸＵエンティティ３２１の別個のインスタンス化である。最上位エンティティ３２０は、シミュレーション・モデル３２９の階層内の最高レベルにある。下位エンティティをインスタンス化するエンティティは、以下では、下位エンティティの「上位（ancestor）」と呼ばれる。したがって、最上位エンティティ３２０は、ＦＸＵエンティティ・インスタンス化３２１ａおよび３２１ｂを直接インスタンス化する上位である。シミュレーション・モデル階層の任意の所与のレベルで、すべてのインスタンス化のインスタンス化名が固有でなければならない。

【００２１】

ＦＸＵエンティティ３２１のインスタンス化３２１ａ内では、エンティティＡ　３２５およびエンティティＢ　３２６の単一インスタンス・エンティティ３２５ａおよび３２６ａが、それぞれ直接インスタンス化される。同様に、同じＦＸＵエンティティのインスタンス化３２１ｂは、それぞれエンティティＡ　３２５およびエンティティＢ　３２６のインスタンス化３２５ｂおよび３２６ｂを含む。同様に、インスタンス化３２６ａおよびインスタンス化３２６ｂはそれぞれ、エンティティＣ　３２７の単一インスタンスを、それぞれエンティティ３２７ａおよび３２７ｂとして直接インスタンス化する。

【００２２】

インスタンス化されたすべてのエンティティが、別個または複合的にかかわらず固有のエンティティ名を有し、任意の所与のレベルの階層のインスタンス化名が互いに固有であると想定すると、他のエンティティ内にエンティティをネストすることは、任意レベルの複雑さまで続行することができる。各エンティティは、エンティティを記述するために必要な情報を含む１つまたは複数のＨＤＬファイルから構築される。

【００２３】

各エンティティのインスタンス化との関連付けが、いわゆる「インスタンス化識別子」である。所与のインスタンス化のインスタンス化識別子は、最上位エンティティ・インスタンス化名から始まるエンクロージング・エンティティ・インスタンス化名からなる文字列である。たとえば、ＦＸＵエンティティ３２１のインスタンス化３２１ａ内にあるエンティティＣ　３２７のインスタンス化３２７ａのインスタンス化識別子は、「ＴＯＰ．Ｆ

ＸＵ０．Ｂ．Ｃ」である。この識別子は、シミュレーション・モデル内の各インスタンス化を固有に識別する働きをする。

【００２４】

例示的シミュレーション・モデル３２９内では、様々な信号がインスタンス化される（たとえば信号Ｅ、Ｆ０、Ｆ１、Ｇ、Ｈ０、Ｈ１、Ｌ、Ｍ、Ｎ、Ｐ、およびＱ）。各信号は、関連する信号名（たとえば「Ｍ」）と、好ましい実施形態では最上位レベル・エンティティ・インスタンス化名から始まり信号名で終わる、エンクロージング・エンティティ・インスタンス化名からなる文字列である、信号インスタンス化識別子とを有する。したがって、ＦＸＵエンティティ３２１のインスタンス化３２１ａ内にある信号Ｍのインスタンス化識別子は「ＴＯＰ．ＦＸＵ０．Ａ．Ｍ」である。このインスタンス化識別子は、シミュレーション・モデル内の各信号インスタンス化を固有に識別する働きをする。たとえば信号Ｐ（０．．４）などの信号は、マルチビット信号ベクトルとすることができることに留意されたい。また、いくつかの信号（たとえば信号ＴＯＰ．ＦＸＵ０．Ｅ、ＴＯＰ．ＦＸＵ１．Ｅ、ＴＯＰ．ＦＸＵ０．Ｇ、ＴＯＰ．ＦＸＵ１．Ｇ）は、設計エンティティの境界線を越えると（それぞれ、信号ＴＯＰ．ＦＸＵ０．Ｆ０、ＴＯＰ．ＦＸＵ１．Ｆ１、ＴＯＰ．ＦＸＵ０．Ｈ０、ＴＯＰ．ＦＸＵ１．Ｈ１として）リネームされることにも留意されたい。

【００２５】

次に図５を参照すると、本発明の好ましい実施形態で実装可能なモデル構築プロセスの流れ図が示されている。このプロセスは、１つまたは複数の設計エンティティＨＤＬソース・コード・ファイル３４０、および、潜在的には、ＨＤＬコンパイラ３４２の以前のランから入手可能な、以下では「プロト・ファイル」３４５と呼ばれる１つまたは複数の設計エンティティ中間フォーマット・ファイル３４５で始まる。ＨＤＬコンパイラ３４２は、シミュレーション・モデルの最上位エンティティで始まり、完全なシミュレーション・モデルを記述するすべてのＨＤＬまたはプロト・ファイルを介して再帰的に進行する、ＨＤＬファイル３４０を処理する。

【００２６】

コンパイル・プロセス中の各ＨＤＬファイル３４０について、ＨＤＬコンパイラ３４２は、以前にコンパイルされたプロト・ファイルが使用可能であり整合性を持つかどうかを判別するために、プロト・ファイル３４５を検査する。こうしたファイルが使用可能であり整合性を持つ場合、ＨＤＬコンパイラ３４２はその特定のファイルを再コンパイルせず、むしろ現存のプロト・ファイルを参照することになる。使用可能なこうしたプロト・ファイルがなく、プロト・ファイルが整合性を持たない場合、ＨＤＬコンパイラ３４２は当該のＨＤＬファイル３４０を明示的に再コンパイルし、その後のコンパイルで使用するためにプロト・ファイル３４４を作成する。こうしたプロセスは、以下では「増分コンパイル」と呼ばれ、シミュレーション実行可能モデル３４８を作成するプロセスを大幅にスピードアップすることができる。ＨＤＬコンパイラ３４２によっていったん作成されると、プロト・ファイル３４４は、その後のコンパイルでプロト・ファイル３４５として働くように使用可能である。

【００２７】

プロト・ファイル３４４に加えて、ＨＤＬコンパイラ３４２は、コンピュータ・システム１０のメモリ４４内に、設計エンティティ・プロト・データ構造３４１および設計エンティティ・インスタンス・データ構造３４３という、２つのデータ構造セットも作成する。設計エンティティ・プロト・データ構造３４１および設計エンティティ・インスタンス・データ構造３４３は、シミュレーション実行可能モデル３４８のコンテンツのメモリ・イメージとして働く。データ構造３４１および３４３は、メモリ４４を介して、データ構造３４１および３４３をシミュレーション実行可能モデル３４８に処理するモデル構築ツール３４６へと渡される。

【００２８】

以下では、各エンティティが単一のＨＤＬファイルによって記述されるものと想定され

る。本発明が実施される規則または特定のＨＤＬに応じて、この制約が必要な可能性がある。しかしながら、特定の環境では、または特定のＨＤＬについては、複数のＨＤＬファイルを使用することによってエンティティを記述することができる。当業者であれば、エントリが複数のＨＤＬファイルによって記述できる場合に、本発明を実施するために必要な拡張を理解されよう。さらに、各エンティティについて、エンティティ名と、エンティティを表すＨＤＬファイルの名前およびそのエンティティのプロト・ファイルの名前の両方との間に、直接の対応関係があるものと想定される。

【００２９】

以下の説明では、所与のエンティティに対応するＨＤＬソース・コード・ファイルは、エンティティ名の後に「．ｖｈｄｌ」を付けて呼ばれることになる。たとえば、最上位エンティティ３２０を記述するＨＤＬソース・コード・ファイルは、ＴＯＰ．ｖｈｄｌと呼ばれることになる。このラベル表示規則は、単なる表記上の便宜として働くものであり、本発明の適用可能性をＶＨＤＬ以外のＨＤＬに限定するものと解釈すべきではない。

【００３０】

図４に戻ると、各エンティティが、１つまたは複数の他のエンティティを直接または間接的にインスタンス化できることがわかる。たとえば、ＦＸＵエンティティは、Ａエンティティ３２５およびＢエンティティ３２６を直接インスタンス化する。さらにＢエンティティ３２６はＣエンティティ３２７を直接インスタンス化する。したがって、ＦＸＵエンティティ３２１は、Ａエンティティ３２５、Ｂエンティティ３２６、およびＣエンティティ３２７を、直接または間接的にインスタンス化する。他のエンティティによって直接または間接的にインスタンス化されるそうしたエンティティを、以下では「下位」と呼ぶ。最上位エンティティ３２０の下位は、ＦＸＵエンティティ３２１、Ａエンティティ３２５、Ｂエンティティ３２６、およびＣエンティティ３２７である。各エンティティが下位の固有セットを有すること、ならびに、エンティティがインスタンス化されるごとに、エンティティの固有インスタンスおよびその下位が作成されることがわかる。シミュレーション・モデル３２９内では、ＦＸＵエンティティ３２１は、最上位エンティティ３２０によって、ＦＸＵ：ＦＸＵ０　３２１ａおよびＦＸＵ：ＦＸＵ１　３２１ｂの２回インスタンス化される。ＦＸＵエンティティ３２１の各インスタンス化は、ＦＸＵ、Ａ、Ｂ、およびＣエンティティのインスタンスの固有セットを作成する。

【００３１】

各エンティティについて、「部品表」またはＢＯＭと呼ばれるものを定義することができる。ＢＯＭは、エンティティそれ自体およびエンティティの下位の日付および時刻スタンプを有する、ＨＤＬファイルのリストである。再度図５を参照すると、エンティティのコンパイル後、エンティティのＢＯＭはプロト・ファイル３４４に格納される。したがって、ＨＤＬコンパイラ３４２がＨＤＬファイル３４０の中の特定のＨＤＬソース・コード・ファイルをコンパイルする場合、もしもあれば、エンティティおよびエンティティの下位を構成するＨＤＬファイル３４０をリスト表示するＢＯＭを含むプロト・ファイル３４４が生成される。ＢＯＭは、ＨＤＬファイルがコンパイルされていたときにコンピュータ・システム１０のディスク／テープ３３上に出現するたびに参照された、ＨＤＬファイルのそれぞれについての日付および時刻スタンプも含む。

【００３２】

エンティティまたはエンティティの下位を構成するいずれかのＨＤＬファイルが後に変更された場合、プロト・ファイル３４４は不整合としてフラグが立てられ、以下でより詳細に説明するように、ＨＤＬコンパイラ３４２は後続の再コンパイルでＨＤＬファイル３４０を再コンパイルすることになる。たとえば図４に戻ると、ＦＸＵエンティティ３２１のＢＯＭによって参照されるＨＤＬファイルは、それぞれ適切な日付および時刻スタンプを備えた、ＦＸＵ．ｖｈｄｌ、Ａ．ｖｈｄｌ、Ｂ．ｖｈｄｌ、およびＣ．ｖｈｄｌである。最上位エンティティ３２０のＢＯＭによって参照されるファイルは、適切な日付および時刻スタンプを備えた、ＴＯＰ．ｖｈｄｌ、ＦＸＵ．ｖｈｄｌ、Ａ．ｖｈｄｌ、Ｂ．ｖｈｄｌ、Ｃ．ｖｈｄｌ、およびＦＰＵ．ｖｈｄｌである。

【００３３】
図５に戻ると、ＨＤＬコンパイラ３４２は、コンピュータ・システム１０のメイン・メモリ４４内にシミュレーション・モデルの構造のイメージを作成する。このメモリ・イメージは、「プロト」データ構造３４１および「インスタンス」データ構造３４３という構成要素からなる。プロトとは、モデル内の各エンティティについて、エンティティのポートに関する情報、エンティティの本体コンテンツ、およびエンティティによって直接インスタンス化された他のエンティティへの参照のリストを含む、データ構造のことである（以下では、「プロト」という用語は、前述のメモリ内データ構造を言い表すために利用され、「プロト・ファイル」という用語は、中間フォーマット・ファイル３４４を記述するために利用される）。したがってプロト・ファイル３４４は、ＨＤＬコンパイラ３４２によって生成されるメモリ内プロト・データ構造のディスク上表現である。

【００３４】
インスタンス・データ構造とは、モデル内のエンティティの各インスタンスについて、インスタンスのインスタンス名、インスタンスが参照するエンティティの名前、および、エンティティを外部信号と相互接続するために必要なポート・マップ情報を含む、データ構造のことである。コンパイル時に、各エンティティはプロト・データ構造を１つだけ有することになるが、エンティティの複数のインスタンス化の場合、各エンティティは１つまたは複数のインスタンス・データ構造を有することができる。

【００３５】
モデルを効率よく増分的にコンパイルするために、ＨＤＬコンパイラ３４２は、プロト・ファイル３４５が使用可能であり、それらのエンティティおよびそれらの下位を構成するＨＤＬソース・ファイルと整合性を持つ場合、モデルの連続エンティティがこうしたファイルから考慮およびロードされる、再帰的コンパイル方法に従う。既存のプロト・ファイル３４５からロードできない各エンティティの場合、ＨＤＬコンパイラ３４２はエンティティの下位を再帰的に検査し、プロト・ファイル３４５から使用可能なそれらの下位エンティティをロードし、プロト・ファイル３４５と不整合なそれらの下位について、必要に応じてプロト・ファイル３４４を作成する。ＨＤＬコンパイラ３４２のメイン制御ループに関する擬似コードが、以下に示される（擬似コードの右側のライン番号は擬似コードの一部ではなく、単に表記上の便宜としての働きをする）。

【数１】

```
process_HDL_file(file)                                          5
{                                                               10
        if (NOT proto_loaded(file)) {                           15
                if (exists_proto_file(file) AND check_bom(file)) {  20
                        load_proto(file);                       25
                } else {                                        30
                        parse_HDL_file(file)                    35
                        for (all instances in file) {           40
                                process_HDL_file(instance);     45
                        }                                       50
                        create_proto(file);                     55
                        write_proto_file(file);                 60
                }                                               65
        }                                                       70
        create_instance(file):                                  75
}                                                               80
```

【００３６】
コンパイラ３４２が初期に呼び出された場合、プロト・データ構造３４１またはインスタンス・データ構造３４３がコンピュータ・システム１０のメモリ４４内に存在する。メイン制御ループである、ルーチンｐｒｏｃｅｓｓ＿ＨＤＬ＿ｆｉｌｅ（）（ライン５）が呼び出され、パラメータ「ｆｉｌｅ」によって、最上位エンティティの名前が渡される。アルゴリズムは第１に、現在のエンティティに関するプロト・データ構造がメモリ４４内に存在するかどうかを、ルーチンｐｒｏｔｏ＿ｌｏａｄｅｄ（）（ライン１５）を使用して判別する。いかなるプロト・データ構造もメモリ４４にロードされることなくプロセス

が開始されるため、最上位エンティティに関するプロト・データ構造は、決してメモリ内に存在しないことに留意されたい。メモリ４４内に合致するプロト・データ構造が存在する場合、現在のエンティティおよび現在のエンティティの下位に関するインスタンス・データ構造があれば、必要に応じて、ルーチンｃｒｅａｔｅ＿ｉｎｓｔａｎｃｅ（）（ライン７５）によって、メモリ４４内に作成される。

【００３７】

しかしながら、合致するプロト・データ構造がメモリ４４内に存在しない場合、制御はライン２０に渡され、ここでルーチンｅｘｉｓｔｓ＿ｐｒｏｔｏ＿ｆｉｌｅ（）は、エンティティに関するプロト・ファイルが存在するかどうかを判別するために、プロト・ファイル３４５を検査する。合致するプロト・ファイルが存在する場合、および存在する場合にのみ、プロト・ファイル３４５が整合性を持つかどうかを判別するために、ルーチンｃｈｅｃｋ＿ｂｏｍ（）が呼び出される。プロト・ファイルが整合性を持つかどうかを判別するために、プロト・ファイルに関するＢＯＭが検査される。ルーチンｃｈｅｃｋ＿ｂｏｍ（）は、ＨＤＬソース・コード・ファイルに関する日付または時刻スタンプが変更されたかどうか、あるいは、ＨＤＬソース・コード・ファイルが削除されたかどうかを判別するために、ＢＯＭ内にリスト表示された各ＨＤＬソース・コード・ファイルを検査する。ＢＯＭ内の任意のファイルに対していずれかの状態が発声した場合、プロト・ファイルは不整合であり、ルーチンｃｈｅｃｋ＿ｂｏｍ（）は失敗する。しかしながら、ｃｈｅｃｋ＿ｂｏｍ（）が成功した場合、制御はライン２５に渡され、ここでルーチンｌｏａｄ＿ｐｒｏｔｏ（）はプロト・ファイルおよび任意の下位プロト・ファイルをメモリ４４内にロードし、現在のエンティティおよび、あれば現在のエンティティの下位に関するプロト・データ構造３４１を作成する。ｐｒｏｃｅｓｓ＿ＨＤＬ＿ｆｉｌｅ（）の構造は、いったんプロト・ファイルが整合性を持つとして検証されると、その下位プロト・ファイルもすべて整合性を持つことを保証する。

【００３８】

プロト・ファイルが実在しないか、または不整合である場合、制御はライン３５に渡され、ここでルーチンｐａｒｓｅ＿ＨＤＬ＿ｆｉｌｅ（）は、現在のエンティティに関するＨＤＬソース・コード・ファイルをロードする。ルーチンｐａｒｓｅ＿ＨＤＬ＿ｆｉｌｅ（）（ライン３５）は、構文上の正確さについてＨＤＬソース・コード・ファイルを検査し、もしもあれば、どちらの下位エンティティが現在のエンティティによってインスタンス化されるかを決定する。ライン４０、４５、および５０は、現在のエンティティによって呼び出される下位エンティティを処理するために、ルーチンｐｒｏｃｅｓｓ＿ＨＤＬ＿ｆｉｌｅ（）が再帰的に呼び出されるループを構成する。このプロセスでは、現在のエンティティのすべての下位のプロト・データ構造３４１およびプロト・データ・ファイル３４４を作成する縦型様式（depth-first fashion）で、現在のエンティティのすべての下位のトラバースを再帰的に繰り返す。下位エンティティが処理されると、制御はライン５５に渡され、ここでルーチンｃｒｅａｔｅ＿ｐｒｏｔｏ（）によって、メモリ４４内に現在のエンティティに対して新しいプロト・データ構造が作成される。次に、制御はライン６０に渡され、ここで、関連するＢＯＭを含む新しいプロト・ファイル３４４がルーチンｗｒｉｔｅ＿ｐｒｏｔｏ＿ｆｉｌｅ（）によってディスク３３に書き込まれる。最終的に、制御はライン７５に渡され、ここでルーチンｃｒｅａｔｅ＿ｉｎｓｔａｎｃｅ（）は、現在のエンティティおよび必要に応じて任意の下位エンティティに対してインスタンス・データ構造３４３を作成する。このようにして、ｐｒｏｃｅｓｓ＿ＨＤＬ＿ｆｉｌｅ（）（ライン５）は、プロト・データ構造３４１およびインスタンス・データ構造３４３からなるモデルのメモリ内イメージを作成するシミュレーション・モデル全体を、再帰的に処理する。

【００３９】

さらに図５に示されるように、さらに本発明は、シミュレーション実行可能モデル３４８のシミュレートの結果を表示する際に注目される可能性が高い特定の信号を識別する、１つまたは複数の信号グループ指示３５０を、設計者が設計エンティティＨＤＬファイル

３４０内に含められるようにする。信号グループ指示３５０に対する例示的なセマンティクスについて、図１２～図１３を参照しながら以下で説明する。前述の処理に加えて、ＨＤＬコンパイラ３４２は、任意の便利なデータ構造（たとえばリンク付きリスト、テーブルなど）を使用する注目される信号の信号インスタンス化識別子を表す信号グループ情報（ＳＧＩ）４００を生成するために、好ましくは信号グループ指示３５０を処理する。モデル構築ツール３４６は、次に、オプションで何らかのフォーマットの追加変換を使用して、信号グループ情報（ＳＧＩ）４００をシミュレーション実行可能モデル３４８内に配置する。

【００４０】

ここで図６を参照すると、本発明の好ましい実施形態で実装可能な、コンパイル済みデータ構造を表すブロック図が示されている。メモリ４４は、シミュレーション・モデル３２９内で参照されるエンティティそれぞれについて１つの、プロト・データ構造３６１を含む。加えて、シミュレーション・モデル３２９におけるインスタンス化はインスタンス・データ構造３６２によって示される。インスタンス・データ構造３６２は、シミュレーション・モデル３２９内のエンティティのインスタンス化の階層的性質を示すポインタによって接続される。最終的に、メモリ４４はＳＧＩ　４００を含む。図５のモード構築ツール３４６は、シミュレーション実行可能モデル３４８を生成するために、メモリ４４のコンテンツをメモリ・データ構造に処理する。

【００４１】

次に図７を参照すると、本発明に従って設計をシミュレートし、シミュレーション結果を表示するためのプロセスの流れ図が示されている。図に示されるように、図５のプロセスによってシミュレーション実行可能モデル３４８が取得されると、デジタル設計の動作をシミュレートするためのテストケース４０２を利用してシミュレーション実行可能モデル３４８をシミュレートするために、ソフトウェアあるいはハードウェアまたはその両方のシミュレータ４０４が使用される。シミュレーション時には、全イベント・トレース（ＡＥＴ）ファイル４０６が、テストケース４０２に対するシミュレーション実行可能モデル３４８の応答を表すデータを記録する。ＡＥＴファイル４０６内のデータは、経時的にシミュレーション実行可能モデル３４８ならびにＳＧＩ４００内の様々な信号あるいはストレージ要素またはその両方の値を含む。

【００４２】

ＡＥＴファイル４０６のコンテンツを再検討するために、ユーザは一般に、本明細書ではＡＥＴビューア４１０と呼ばれる、別個のまたは一体化されたビューア・プログラムを採用する。たとえばユーザは、ＡＥＴファイル４０６からのデータを、ディスプレイ画面２２内にグラフィカル・フォーマットで、またはハードコピー・フォーマットで提示するように、ＡＥＴビューア４１０に要求することができる。前述のようにユーザは、データ・セット（本明細書ではＩ／Ｏリスト４０８と呼ばれる）内に注目する信号を指定することによって、ＡＥＴビューア４１０によるデータのプレゼンテーションを、注目する特定の信号に有利に限定することができる。

【００４３】

図８に示されるように、従来技術に従った従来のＩ／Ｏリスト５００は、それぞれが注目する信号のうちの１つの信号インスタンス化識別子を示す大量のエントリを含むリストであり、この場合は図４のＦＸＵインスタンス化３２１ａ内のすべての信号を備える。したがって、図８で与えられた簡略化された例から理解されるように、従来技術では、ユーザは、潜在的に大量の信号インスタンス化識別子のそれぞれを、キーボード１６を使用して入力しなければならない。信号インスタンス化識別子を手動でキー入力するこの従来の技法は、退屈でエラーの起こりやすいものである。

【００４４】

加えて、一部のシミュレータ４０４は、設計エンティティの境界線をより高水準の設計エンティティへと越境する下位設計エンティティ内の信号の信号名を保護しない。その代わりに、ＡＥＴファイルにおける信号の重複をなくすために、こうしたシミュレータ４０

４は、その内部に出現する最高位設計エンティティ内のその信号名によってのみ、ＡＥＴファイル内の信号を識別する。したがって、信号名を保護しないシミュレータ４０４が採用される場合、ユーザは、信号が出現する最高水準設計エンティティからの信号の信号名を採用する信号インスタンス化識別子を利用して、Ｉ／Ｏリスト５００内に信号を指定しなければならない。たとえば、図８のエントリ５０２と図９のＩ／Ｏリスト５００’の対応するエントリ５０４とを比較することでわかるように、信号名を保護しないシミュレータ４０４のユーザは、参照番号５０２で示されるような信号インスタンス化識別子ＦＸＵ．Ｅではなく、参照番号５０４で示されるような信号インスタンス化識別子Ｆ０を利用しなければならない。理解されるように、作業が主に下位設計エンティティに関連するＡＥＴビューア４１０のユーザは、ユーザが精通している低水準設計エンティティをインスタンス化する、より高水準の設計エンティティ内で使用される信号名を決定すること、または容易に想起することが、困難な可能性がある。

【００４５】

従来のＩ／Ｏリスト５００または５００’の代わりに、本発明は、ＡＥＴビューア４１０のユーザが、図１０に従って、１つまたは複数の改良されたＩ／Ｏリスト４０８を利用してＡＥＴファイル４０６からのデータのプレゼンテーションを代わりにフィルタリングできるようにする。図１０に示されるように、Ｉ／Ｏリスト４０８は、１つまたは複数のエントリを含むリストである。図８または図９に示されるような従来の信号インスタンス化識別子を含むゼロまたはそれ以上のエントリに加えて、エントリ５１０などのＩ／Ｏリスト４０８のエントリは、ＳＧＩ４００内の情報に対応する信号グループ・インスタンス化識別子によって、注目する１つまたは複数の信号のグループを識別することができる。

【００４６】

図に示されるように、信号グループ・インスタンス化識別子は、信号インスタンス化識別子と同様に形成され、最高位エンティティ・インスタンス化名から始まり、括弧付きコンテンツが別の信号グループ名前空間のメンバであることを示す１対の山括弧（「＜」および「＞」）によって囲まれた信号グループ名で終わる、エンクロージング・エンティティ・インスタンス化名の文字列からなる。したがって、ＦＸＵエンティティ・インスタンス化３２１ａ内で注目する６つの信号は、信号インスタンス化識別子をＩ／Ｏリスト４０８内に個々に入力するのではなく、単一エントリ「ＦＸＵ０．＜ＦＸＵ＿Ｇｒｏｕｐ＞」によって簡単に識別することができる。前述のように、信号グループＦＸＵ＿Ｇｒｏｕｐを備える個々の信号は、設計エンティティＨＤＬファイル３４０内の信号グループ指示３５０を利用して指定される。

【００４７】

次に、図１２を参照すると、本発明に従った設計エンティティＨＤＬファイル３４０ａの例示的実施形態が示される。当業者であれば理解されるように、設計エンティティＨＤＬファイル３４０ａは、この場合は設計エンティティＦＸＵ　３２１である、設計エンティティを記述する従来のＨＤＬソース・コードを含む。従来のＨＤＬソース・コードは、ポート・マップ６００および信号割り当てステートメント６０２を含む。加えて、設計エンティティＨＤＬファイル３４０ａは、本発明に従った信号グループ指示３５０（図５）を含む従来のＨＤＬコメントを含む。

【００４８】

設計エンティティＨＤＬファイル３４０ａでは、信号グループ指示３５０は、信号グループ宣言６１０および信号保護指示６２０という、２つの異なるタイプの信号グループ指示を含む。信号グループ宣言６１０は「－－！！Ｓｉｇｎａｌ　Ｇｒｏｕｐ　ｓｉｇｎａｌ＿ｇｒｏｕｐ＿ｎａｍｅ；」の形のＨＤＬコメントで始まり、「－－！！ＥＮＤ　Ｓｉｇｎａｌ　Ｇｒｏｕｐ　ｓｉｇｎａｌ＿ｇｒｏｕｐ＿ｎａｍｅ；」の形のＨＤＬコメントで終わるが、ここでｓｉｇｎａｌ＿ｇｒｏｕｐ＿ｎａｍｅは所与のターゲット設計エンティティに対して固有の信号グループ名（この例ではＦＸＵ＿Ｇｒｏｕｐ）である。信号グループ宣言６１０の最初と最後の間に、注目する１つまたは複数の信号の信号名が、ＡＥＴビューア４１０によって所望のプレゼンテーション順にリスト表示される。この実施形

態では、信号名はターゲット設計エンティティ（たとえばＦＸＵ設計エンティティ３２１）に関して指定される。本発明の少なくとも一部の実施形態は、設計エンティティ階層内で、次に高位の階層を示すために、慣用構文「．．＼」を利用して、ターゲット設計エンティティに関してより上位の信号名を指定することができる。

【００４９】

ユーザは、好ましくは、信号グループ宣言６１０内の信号のプレゼンテーションに関して、追加の属性をさらに指定することができる。たとえば、ユーザは、信号の所望の色、波形またはバイナリ信号表現のデフォルト、非整列ビット・ベクトルの所望の位置合わせなどを指定することができる。したがって、信号グループ宣言６１０のステートメント６１２では、ユーザは５ビット信号ベクトルＢ．Ｃ．Ｐ（０．．４）の左位置合わせを指定している。

【００５０】

さらに図１３の設計エンティティＨＤＬファイル３４０ｂに示されるように、信号グループ宣言６３０などの信号グループ宣言も、好ましくは、ユーザが任意の正当な深さまでネストされた信号グループを指定できるようにする。より大規模な信号グループの一部を含むネストされた信号グループを指定するために、ユーザは、山括弧（すなわち「＜」および「＞」）に囲まれた信号グループを備えたネストされた信号グループのインスタンス化識別子を参照する信号グループ宣言内に、ステートメントを含めるだけである。山括弧を使用することで、ＨＤＬコンパイラ３４２は、信号の名前空間と信号グループ名との間を区別することができる。

【００５１】

再度図１２を参照すると、信号保護指示６２０を利用して、特定のリネームされた信号（たとえば信号Ｇ）を保存するために、デフォルトでは、リネームされた信号より低位の信号名を保存しないことをシミュレータ４０４に指示する。したがって、シミュレータ４０４がデフォルトではリネームされた信号の低位の名前を保護しないものと想定すると、ＡＥＴファイル４０６は、信号インスタンス化識別子ＴＯＰ．ＦＸＵ０．ＧおよびＴＯＰ．ＦＸＵ１．Ｇに関するデータを含むことになる。図９に関して上記で述べたように、精通した信号名を保護するための機能により、ユーザが、保護された信号Ｇの信号データを表示するために、精通していない可能性のある信号インスタンス化識別子ＴＯＰ．ＦＸＵ１．Ｈ０およびＴＯＰ．ＦＸＵ１．Ｈ１をＩ／Ｏリストに入力する必要がなくなる。

【００５２】

次に図１４～図１６を参照すると、本発明に従った、ＡＥＴビューア４１０がＩ／Ｏリスト４０８を処理する際に使用する例示的プロセスの高水準論理流れ図が示される。論理流れ図として、動作は順番ではなく論理的に示され、示された動作の多くは並行して、または代替順で実行することができる。

【００５３】

図に示されるように、プロセスは図１４のブロック７００で始まり、その後、ユーザがＩ／Ｏリスト４０８の参照有効範囲、すなわち、参照によりすべての他のＩ／Ｏリスト・エントリが解析されるシミュレーション実行可能モデル３４８の有効範囲に入ったか否かの判別を示す、ブロック７０２へと進む。デフォルトでは、参照有効範囲は、シミュレーション実行可能モデル３４８内の最上位設計エンティティ・インスタンス、たとえば図４の最上位設計エンティティ・インスタンスである。一実施形態では、ユーザは「Ｓｃｏｐｅ　ｌｉｍｉｔ：ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０．［ｄｅｓｉｇｎ＿ｅｎｔｉｔｙ］．ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ１」の形の参照有効範囲をさらに限定するコマンドを入力することが可能であり、ここでｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０およびｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ１はオプションの設計エンティティ・インスタンス文字列であり（ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０は最上位設計エンティティ・インスタンスで始まる）、角括弧で囲まれたｄｅｓｉｇｎ＿ｅｎｔｉｔｙはオプションの設計エンティティ名である。正当な参照有効範囲コマンドは、３つのオプション・フィールド、ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０、［ｄｅｓｉｇｎ＿ｅｎｔｉｔｙ］、およびｉｎｓｔａｎｃｅ＿ｓｔｒ

ｉｎｇ１のうちの少なくとも１つを含む。参照有効範囲コマンドは、たとえばコマンド行を介してＡＥＴビューア４１０に、またはＩ／Ｏリスト４０８内に送ることができる。Ｉ／Ｏリスト４０８に挿入された場合、参照有効範囲コマンドは、好ましくはそのＩ／Ｏリスト４０８内のすべてのエントリに適用される。

【００５４】

ブロック７０２で示されるように、ユーザが異なる参照有効範囲を入力していない場合、ブロック７０４で示されるように、ＡＥＴビューア４１０はデフォルトで、参照有効範囲をシミュレーション実行可能モデル３４８の最上位設計エンティティ・インスタンスに設定する。その後、プロセスはページ連結子Ａを通って図１５へと進む。ＡＥＴビューア４１０がブロック７０２で、ユーザが異なる参照有効範囲を入力したものと判別した場合、ＡＥＴビューア４１０は、ブロック７１０で、参照有効範囲コマンドが括弧付き構文（たとえば［ｄｅｓｉｇｎ＿ｅｎｔｉｔｙ］）を含むか否かを判別するために参照有効範囲コマンドをさらに解析する。含まない場合、プロセスは、参照有効範囲コマンドによって指定された設計エンティティ・インスタンスがシミュレーション実行可能モデル３４８内に存在するか否かを、ＡＥＴビューア４１０が判別することを示す、ブロック７１３に進む。存在しない場合、プロセスはブロック７１５でエラー終了する。存在する場合、プロセスはブロック７１３からブロック７２０へと渡され、ここでＡＥＴビューア４１０は、参照有効範囲を、参照有効範囲コマンドのｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０フィールド内に指定された特定の設計エンティティ・インスタンスに設定する。たとえば、図４のシミュレーション実行可能モデル３２９を想定すると、参照有効範囲コマンド「Ｓｃｏｐｅ　ｌｉｍｉｔ：ＴＯＰ．ＦＸＵ０」は、参照有効範囲をＦＸＵ設計エンティティ３２１のインスタンス３２１ａに設定する。この参照有効範囲の場合、図１０のＩ／Ｏリスト４０８のエントリ５１０は＜ＦＸＵ＿Ｇｒｏｕｐ＞として簡略化することができる。その後、プロセスはページ連結子Ａを通って図１５へと進む。

【００５５】

ブロック７１０に戻ると、参照有効範囲コマンドが括弧付き構文を採用しているとの決定に応答して、ＡＥＴビューア４１０は次に、ブロック７１２で、名前付き設計エンティティがシミュレーション実行可能モデル３４８内に存在するか否かを判別する。存在しない場合、プロセスはブロック７１４でエラー終了する。しかしながら、指定された設計エンティティがシミュレーション実行可能モデル３４８内に存在する場合、ＡＥＴビューア４１０は、存在する場合はｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ０で定義されるシミュレーション実行可能モデル３４８の一部分に存在する設計エンティティのすべてのインスタンスを識別するために、シミュレーション実行可能モデル３４８を再帰的に検索する。ブロック７１８で示されるように、指定された設計エンティティのインスタンスが指定された有効範囲内に存在しない場合、プロセスはブロック７１４でエラー終了する。ＡＥＴビューア４１０がブロック７１８で、指定された有効範囲内に、指定された設計エンティティの少なくとも１つのインスタンスの位置が突き止められたものと判別した場合、さらにＡＥＴビューア４１０は、ブロック７２２で、指定された有効範囲内に設計エンティティの単一のインスタンスのみが見つけられたかどうかを判別する。見つけられた場合、プロセスはブロック７２０へと渡され、ここでＡＥＴビューア４１０は、設計エンティティの単一のインスタンスを参照有効範囲として設定する。その後、プロセスはページ連結子Ａを通って図１５へと進む。

【００５６】

ＡＥＴビューア４１０が、ブロック７２２で、指定された有効範囲内に指定された設計エンティティの複数のインスタンスが見つけられたものと判別した場合、ＡＥＴビューア４１０は、たとえばディスプレイ画面２２内に表示されるグラフィカル・メニューを介して選択するために、設計エンティティ・インスタンスのリストをユーザに提示する（ブロック７２４）。その後ＡＥＴビューア４１０は、ブロック７２６で示されるように、所望の参照範囲を定義する複数の設計エンティティ・インスタンスのうちの１つを指定するユーザ入力を受け取る。たとえば、図４のシミュレーション実行可能モデル３２９および図

１１のＩ／Ｏリスト４０８’を想定すると、図１１の参照番号５１２で与えられた参照有効範囲コマンド「Ｓｃｏｐｅ　ｌｉｍｉｔ：［ＦＸＵ］」は、ＡＥＴビューア４１０に、シミュレーション実行可能モデル３２９内でＦＸＵインスタンス３２１ａおよび３２１ｂの位置を突き止めさせ、インスタンス３２１ａおよび３２１ｂのインスタンス識別子を選択のためにユーザに提示させる。有利なことに、ＦＸＵインスタンス３２１ａおよび３２１ｂのどちらが注目される、ユーザによって選択されるかに関わらず、信号グループは、図１１のＩ／Ｏリスト４０８’のエントリ５１２に示されるように＜ＦＸＵ＿Ｇｒｏｕｐ＞として指定することができる。その後プロセスは、これまでに説明したブロック７２０に渡される。

【００５７】

次に図１５を参照すると、プロセスはページ連結子Ａで始まり、その後ブロック７３０へと進んで、ＡＥＴファイル４０６のプレゼンテーションを構築するために、ＡＥＴビューア４１０が１つまたは複数のＩ／Ｏリスト４０８内の各エントリを処理する、処理ループを提示する。ブロック７３０でＡＥＴビューア４１０が、Ｉ／Ｏリスト内のすべてのエントリが処理されたものと決定した場合、プロセスは以下で説明するブロック７３２へと渡される。しかしながら、Ｉ／Ｏリスト４０８内に、処理されるべき少なくとも１つのエントリが残っている場合、プロセスはブロック７４０へと進み、ここでＡＥＴビューア４１０は第１または次の信号識別エントリへと移動して、エントリの作業有効範囲を参照有効範囲へと初期化する。Ｉ／Ｏリスト４０８内の信号識別エントリは、参照有効範囲に関して処理され（すなわち参照有効範囲によって暗に限定され）、以下の形を取り、

ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ２．［ｄｅｓｉｇｎ＿ｅｎｔｉｔｙ＊］．ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ３．ｓｉｇｎａｌｓ

上式で、ｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ２およびｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ３はオプションの設計エンティティ・インスタンス文字列であり、

角括弧（すなわち「［」および「］」）で囲まれたｄｅｓｉｇｎ＿ｅｎｔｉｔｙはオプションの設計エンティティ名であり、

＊は、指定された有効範囲内のすべてのｄｅｓｉｇｎ＿ｅｎｔｉｔｙインスタンスを示すオプションのユニバーサル演算子であり、

ｓｉｇｎａｌｓは、信号名、山括弧（すなわち「＜」および「＞」）で囲まれた信号グループ名、または指定された有効範囲内のすべての信号グループを示す空の山括弧を指定する、必須パラメータである。

【００５８】

図１５に示されるプロセスは、ブロック７４０からブロック７４２へと進み、ここでＡＥＴビューア４１０は、現在の信号識別エントリが先行設計エンティティ・インスタンス修飾子（たとえばｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ２）を有するかどうかを判別する。有さない場合、プロセスは以下で説明するブロック７５２へと渡される。有する場合、プロセスはブロック７４２からブロック７４４へと進み、ここでＡＥＴビューア４１０は、シミュレーション実行可能モデル３４８の参照有効範囲内に指定された設計エンティティ・インスタンスが存在するか否かを判別する。存在しない場合、プロセスはブロック７４６でエラー終了する。存在する場合、プロセスはブロック７５０へ進む。ブロック７５０は、現在の有効範囲を、設計エンティティ・インスタンス修飾子によって定義された有効範囲を参照有効範囲に追加することによって形成された有効範囲に設定する。次に、ＡＥＴビューア４１０は、ブロック７５２で、Ｉ／Ｏリスト４０８の信号識別エントリ内の次のフィールドが、角括弧に囲まれたｄｅｓｉｇｎ＿ｅｎｔｉｔｙ修飾子であるか否かを判別する。括弧付き修飾子でない場合、プロセスはページ連結子Ｂを通って図１６へと進む。括弧付き修飾子である場合、信号識別エントリの処理はブロック７５４へと続く。

【００５９】

ブロック７５４は、ＡＥＴビューア４１０が、指定された設計エンティティ名に合致するエンティティ名を有する現在の有効範囲内で、すべての設計エンティティ・インスタンスの位置を突き止めるために、シミュレーション実行可能モデル３４８を再帰的に検索す

ることを示す。その後ＡＥＴビューア４１０は、ブロック７５６で、何らかのこうした設計エンティティ・インスタンスが存在するか否かを判別する。存在しない場合、処理はブロック７４６でエラー状態で終了する。存在する場合、さらにＡＥＴビューア４１０は、ブロック７６０で、括弧付き構文が、指定された有効範囲内へのすべての設計エンティティ・インスタンスの包含を示すアスタリスクを含むか否かを判別する。含む場合、ＡＥＴビューア４１０は、エントリの作業有効範囲を、ブロック７５４で突き止められた１つまたは複数の設計エンティティ・インスタンスまで狭める（ブロック７６１）。その後プロセスは、ページ連結子Ｂを通って図１６へと進む。

【００６０】

ブロック７６０に戻ると、否定の決定に応答して、プロセスはブロック７６２へと進み、ここでＡＥＴビューア４１０は、単一の設計エンティティ・インスタンスがブロック７５４で発見されたかどうかを判別する。発見された場合、ブロック７６４で、作業有効範囲は単一の設計エンティティ・インスタンスに設定される。その後プロセスは、ページ連結子Ｂを通って図１６へと進む。これに対して、ブロック７６２でＡＥＴビューア４１０が、ブロック７５４で複数の設計エンティティ・インスタンスが発見されたものと決定した場合、ＡＥＴビューア４１０は、たとえばディスプレイ画面２２内に表示されるグラフィカル・メニューを介して選択するために、設計エンティティ・インスタンスのリストをユーザに提示する（ブロック７７０）。その後、ＡＥＴビューア４１０は、ブロック７７２で示されるように、所望の作業有効範囲を定義する複数の設計エンティティ・インスタンスのうちの１つまたは複数を指定するユーザ入力を受け取る。その後、プロセスはブロック７６４に渡され、ここでＡＥＴビューア４１０は、ユーザの選択に従って１つまたは複数の作業有効範囲を確立する。その後プロセスは、ページ連結子Ｂを通って図１６へと進む。

【００６１】

次に図１６を参照すると、プロセスはページ連結子Ｂで開始され、その後ブロック７７４へ進み、ここでさらにＡＥＴビューア４１０は、エントリが作業有効範囲を制限するための他のインスタンス修飾子（たとえばｉｎｓｔａｎｃｅ＿ｓｔｒｉｎｇ３）を含むかどうかを判別するために、信号識別エントリを解析する。含まない場合、プロセスは以下で説明するブロック７８０へ渡される。含む場合、ＡＥＴビューア４１０は、ブロック７７６で、指定された設計エンティティ・インスタンスが存在するか否かを判別する。存在しない場合、プロセスはブロック７８６でエラー終了する。指定された設計エンティティ・インスタンスが存在する場合、ＡＥＴビューアは、現在の信号識別エントリの作業有効範囲を、第２のインスタンス修飾子によって示された設計エンティティ・インスタンスまで狭める（ブロック７７８）。その後、プロセスはブロック７８０に渡される。

【００６２】

ブロック７８０で、さらにＡＥＴビューア４１０は、エントリの終端ｓｉｇｎａｌｓフィールドが単一の信号名または信号グループ名を含むかどうかを判別するために、Ｉ／Ｏリスト４０８の信号識別エントリを解析する。ｓｉｇｎａｌｓフィールドが信号名を含む場合、次にＡＥＴビューア４１０は、ブロック７８１で、指定された信号が作業有効範囲内に存在するかどうかを判別する。存在しない場合、プロセスはブロック７８６でエラー状態で終了する。存在する場合、ＡＥＴビューア４１０は、指定された信号をＡＥＴファイル４０６のプレゼンテーションに追加する（ブロック７８２）。その後プロセスは、ページ連結子Ｃを介して図１５のブロック７３０に戻る。

【００６３】

再度ブロック７８０を参照すると、信号識別エントリのｓｉｇｎａｌｓフィールドが信号名を含まないとの決定に応答して、プロセスはブロック７８３へ渡され、ここでＡＥＴビューア４１０は、ｓｉｇｎａｌｓフィールドが空の山括弧を含むかどうかを判別する。含む場合、ＡＥＴビューア４１０は、ブロック７８４に示されるように、作業有効範囲内にある設計エンティティ・インスタンス内のすべての信号グループ・インスタンスの位置を再帰的に突き止める。ＡＥＴビューアがブロック７８５で、何も存在しないと決定した

場合、プロセスはブロック７８６でエラー終了する。これに対して、ＡＥＴビューア４１０がブロック７８５で、１つまたは複数の信号グループが存在すると決定した場合、ＡＥＴビューア４１０は、個々の信号がＳＧＩ　４００からの信号グループを備えることを決定し、ブロック７９０で、すべてのこうした信号をプレゼンテーションに追加する。その後プロセスは、ページ連結子Ｃを通って図１５に戻る。

【００６４】

ブロック７８３に戻り、ＡＥＴビューア４１０が、Ｉ／Ｏリスト４０８の信号識別エントリのｓｉｇｎａｌｓフィールドが空の山括弧を含まないが、その代わりに山括弧内に信号グループ名を指定するものと決定した場合、プロセスはブロック７８７に渡される。ブロック７８７で、ＡＥＴビューア４１０は、作業有効範囲内にある設計エンティティ・インスタンス内の指定された信号グループ・インスタンスのインスタンスの位置を再帰的に突き止める。ブロック７８８によって表されるように、信号グループ・インスタンスの位置が突き止められない場合、プロセスはブロック７８６でエラー状態で終了する。別の方法として、名前付き信号グループの少なくとも１つのインスタンスの位置が突き止められた場合、プロセスはブロック７９０へと進み、以前に説明したブロックへと続く。

【００６５】

以上説明してきたように、本発明は、シミュレーション結果をプレゼンテーション用に処理するための方法、システム、およびプログラム製品を提供する。本発明によれば、シミュレーション結果のプレゼンテーションをフィルタリングするために必要なユーザ入力の量は、所定の信号グループの使用、およびオプションで有効範囲コマンドの使用によって、大幅に削減される。加えて、信号リネームが存在する中で信号名が保護されることになる信号を設計者が指定できるようにする信号保護指示のサポートを通じて、シミュレーション結果の理解しやすさが向上する。

【００６６】

以上、本発明について、好ましい実施形態を参照しながら具体的に説明してきたが、当業者であれば、本発明の趣旨および範囲を逸脱することなく、形式および細部における様々な変更が実行可能であることを理解されよう。たとえば、本発明の諸実施形態のうちの１つは、図１および図２で説明されたように全体として構成される１つまたは複数のコンピュータ・システムの、ランダム・アクセス・メモリ２８または不揮発性ストレージ内に常駐するプログラム・コードを使用して、実装可能である。プログラム・コードのセットは、ユーザによって所望された場合、コンピュータ・システム１０によって要求されるまで、ディスク・ドライブ３３またはＣＤ－ＲＯＭなどの他のコンピュータ読み取り可能ストレージ・デバイス内、あるいは他のコンピュータのデータ・ストレージ内に格納すること、および、ローカル・エリア・ネットワークまたはインターネットなどのワイド・エリア・ネットワークを介して伝送することが可能である。コンピュータ使用可能媒体内で具体化されたプログラム・コードは、コンピュータ・プログラム製品と呼ぶことができる。

【００６７】

（１）データ処理システムにおけるデータ処理の方法であって、

所定の信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットを入力として受け取るステップと、

前記データ・セットの受け取りに応答して、

前記信号グループ名を識別するために、前記データ・セット内の前記エントリを処理するステップと、

前記信号グループのメンバである複数の信号の信号名を決定するために、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グループ情報にアクセスするステップと、

前記複数の信号のインスタンスに関連付けられた前記イベント・トレース・ファイルからのそれらのシミュレーション結果を、プレゼンテーションに含めるステップと、

を有する、方法。

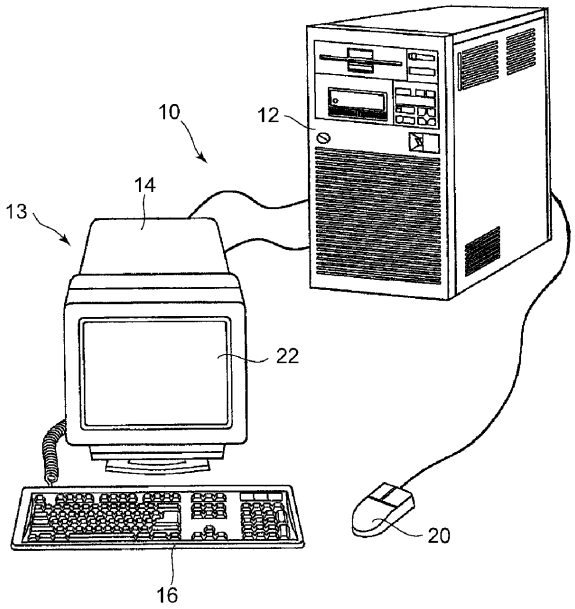
（２）シミュレーション実行可能モデル内でインスタンス化された設計エンティティを記

述するＨＤＬソース・コード・ファイルを参照して前記信号グループ内の前記信号名のメンバシップを確立するステップをさらに有する、（１）に記載の方法。
（３）前記信号グループが第１の信号グループを備え、前記第１の信号グループが、前記複数の信号をメンバとして集合的に有する第２および第３の信号グループを備える、（１）または（２）に記載の方法。
（４）リネームから関連する信号名が保護されることになる特定の信号を識別する保護指示を受け取るステップと、
前記保持指示に応答して、プレゼンテーションにおいて前記関連する信号名によって前記信号を識別するステップと、
をさらに有する、（１）、（２）、または（３）に記載の方法。
（５）前記シミュレーション実行可能モデル内の設計エンティティ・インスタンスに関して、参照有効範囲を示す有効範囲コマンドを入力として受け取るステップと、
前記有効範囲コマンドの受け取りに応答して、前記参照有効範囲に関する前記データ・セット内の前記少なくとも１つのエントリを解釈するステップと、
をさらに有する、（１）から（４）のいずれか一つに記載の方法。
（６）前記シミュレーション実行可能モデル内に複数の設計エンティティ・インスタンスを有する設計エンティティ名を、エントリ内で識別するために、前記データ・セット内の前記エントリを処理するステップと、
前記設計エンティティ名の識別に応答して、ユーザの選択のために、前記複数の設計エンティティ・インスタンスのインスタンス識別子を提示するステップと、
前記インスタンス識別子のうちの１つのユーザ選択に応答して、前記信号グループに属する信号のインスタンスが位置する有効範囲を狭めるステップと、
をさらに有する、（１）から（５）のいずれか一つに記載の方法。
（７）コンピュータ読み取り可能媒体と、
前記コンピュータ読み取り可能媒体内のプログラム・コードと、を備えるプログラムであって、前記プログラム・コードがデータ処理システムに、
所定の信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットを入力として受け取るステップと、
前記データ・セットの受け取りに応答して、
前記信号グループ名を識別するために、前記データ・セット内の前記エントリを処理するステップと、
前記信号グループのメンバである複数の信号の信号名を決定するために、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グループ情報にアクセスするステップと、
前記複数の信号のインスタンスに関連付けられた前記イベント・トレース・ファイルからのそれらのシミュレーション結果を、プレゼンテーションに含めるステップと、
を含む方法を実行させる、プログラム。
（８）前記方法が、シミュレーション実行可能モデル内でインスタンス化された設計エンティティを記述するＨＤＬソース・コード・ファイルを参照して前記信号グループ内の前記信号名のメンバシップを確立するステップをさらに含む、（７に記載のプログラム。
（９）前記信号グループが第１の信号グループを備え、前記第１の信号グループが、前記複数の信号をメンバとして集合的に有する第２および第３の信号グループを備える、（７）または（８）に記載のプログラム。
（１０）前記方法が、
リネームから関連する信号名が保護されることになる特定の信号を識別する保護指示を受け取るステップと、
前記保持指示に応答して、プレゼンテーションにおいて前記関連する信号名によって前記信号を識別するステップと、
をさらに含む、（７）、（８）、または（９）に記載のプログラム。
（１１）前記方法が、

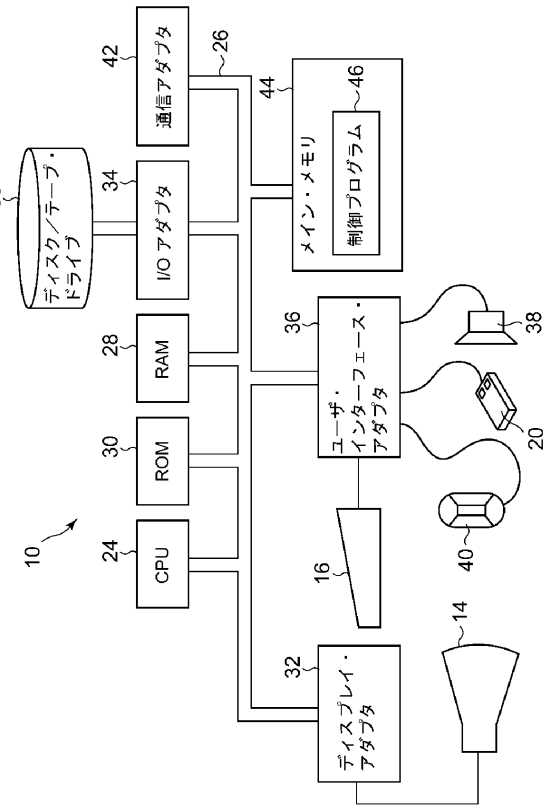
前記シミュレーション実行可能モデル内の設計エンティティ・インスタンスに関して、参照有効範囲を示す有効範囲コマンドを入力として受け取るステップと、
前記有効範囲コマンドの受け取りに応答して、前記参照有効範囲に関する前記データ・セット内の前記少なくとも１つのエントリを解釈するステップと、
をさらに含む、（７）から（１０）のいずれか一つに記載のプログラム。
（１２）前記方法が、
前記シミュレーション実行可能モデル内に複数の設計エンティティ・インスタンスを有する設計エンティティ名を、エントリ内で識別するために、前記データ・セット内の前記エントリを処理するステップと、
前記設計エンティティ名の識別に応答して、ユーザの選択のために、前記複数の設計エンティティ・インスタンスのインスタンス識別子を提示するステップと、
前記インスタンス識別子のうちの１つのユーザ選択に応答して、前記信号グループに属する信号のインスタンスが位置する有効範囲を狭めるステップと、
をさらに含む、（７）から（１１）のいずれか一つに記載のプログラム。
（１３）プロセッサと、
前記プロセッサに結合されたデータ・ストレージと、を備える、データ処理システムであって、前記データ・ストレージが、前記データ処理システムに、
所定の信号グループ名によって信号グループを指定する少なくとも１つのエントリを含むデータ・セットを入力として受け取るステップと、
前記データ・セットの受け取りに応答して、
前記信号グループ名を識別するために、前記データ・セット内の前記エントリを処理するステップと、
前記信号グループのメンバである複数の信号の信号名を決定するために、シミュレーション結果を含むイベント・トレース・ファイルに関連付けられた信号グループ情報にアクセスするステップと、
前記複数の信号のインスタンスに関連付けられた前記イベント・トレース・ファイルからのそれらのシミュレーション結果を、プレゼンテーションに含めるステップと、
を含む方法を実行させる、プログラム・コードを含む、データ処理システム。
（１４）前記方法が、シミュレーション実行可能モデル内でインスタンス化された設計エンティティを記述するＨＤＬソース・コード・ファイルを参照して前記信号グループ内の前記信号名のメンバシップを確立するステップをさらに含む、（１３）に記載のデータ処理システム。
（１５）前記信号グループが第１の信号グループを備え、前記第１の信号グループが、前記複数の信号をメンバとして集合的に有する第２および第３の信号グループを備える、（１３）または（１４）に記載のデータ処理システム。
（１６）前記方法が、
リネームから関連する信号名が保護されることになる特定の信号を識別する保護指示を受け取るステップと、
前記保持指示に応答して、プレゼンテーションにおいて前記関連する信号名によって前記信号を識別するステップと、
をさらに含む、（１３）、（１４）、または（１５）に記載のデータ処理システム。
（１７）前記方法が、
前記シミュレーション実行可能モデル内の設計エンティティ・インスタンスに関して、参照有効範囲を示す有効範囲コマンドを入力として受け取るステップと、
前記有効範囲コマンドの受け取りに応答して、前記参照有効範囲に関する前記データ・セット内の前記少なくとも１つのエントリを解釈するステップと、
をさらに含む、（１３）から（１６）のいずれか一つに記載のデータ処理システム。
（１８）前記方法が、
前記シミュレーション実行可能モデル内に複数の設計エンティティ・インスタンスを有する設計エンティティ名を、エントリ内で識別するために、前記データ・セット内の前記

エントリを処理するステップと、
前記設計エンティティ名の識別に応答して、ユーザの選択のために、前記複数の設計エンティティ・インスタンスのインスタンス識別子を提示するステップと、
前記インスタンス識別子のうちの１つのユーザ選択に応答して、前記信号グループに属する信号のインスタンスが位置する有効範囲を狭めるステップと、
をさらに含む、（１３）から（１７）のいずれか一つに記載のデータ処理システム。
【図面の簡単な説明】
【００６８】
【図１】本発明に従ったデータ処理システムを示す絵画図である。
【図２】図１に示されたデータ処理システムの代表的なハードウェア環境を示す図である。
【図３】本発明の教示に従ったデジタル設計エンティティを示す簡易ブロック図である。
【図４】本発明の教示に従ったシミュレーション・モデルを示す概略図である。
【図５】本発明の教示に従ったモデル構築プロセスを示す流れ図である。
【図６】本発明の教示に従った設計を表すシミュレーション・モデル・データ構造を示すブロック図である。
【図７】シミュレーション実行可能モデルのシミュレーションおよびシミュレーションの結果をユーザに提示することを示す流れ図である。
【図８】従来技術に従った第１の従来のＩ／Ｏリストを示す図である。
【図９】従来技術に従った第２の従来のＩ／Ｏリストを示す図である。
【図１０】本発明に従った例示的Ｉ／Ｏリストを示す図である。
【図１１】本発明に従った例示的Ｉ／Ｏリストを示す図である。
【図１２】本発明に従った、信号グループ記述子を含む例示的設計エンティティＨＤＬファイルを示す図である。
【図１３】本発明に従った、ネストされた信号グループ記述子を含む例示的設計エンティティＨＤＬファイルを示す図である。
【図１４】本発明に従った、ＡＥＴビューアがＡＥＴファイルのプレゼンテーションを生成するためにＩ／Ｏリストを処理する際に使用する例示的プロセスを示す、高水準論理流れ図の一部である。
【図１５】本発明に従った、ＡＥＴビューアがＡＥＴファイルのプレゼンテーションを生成するためにＩ／Ｏリストを処理する際に使用する例示的プロセスを示す、高水準論理流れ図の一部である。
【図１６】本発明に従った、ＡＥＴビューアがＡＥＴファイルのプレゼンテーションを生成するためにＩ／Ｏリストを処理する際に使用する例示的プロセスを示す、高水準論理流れ図の一部である。

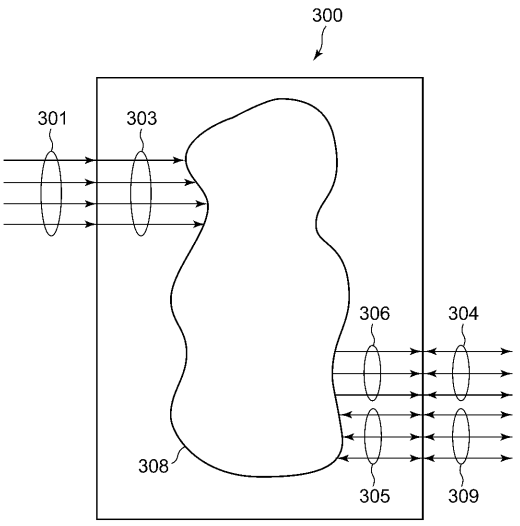
【図１】


10
12
13
14
16
20
22

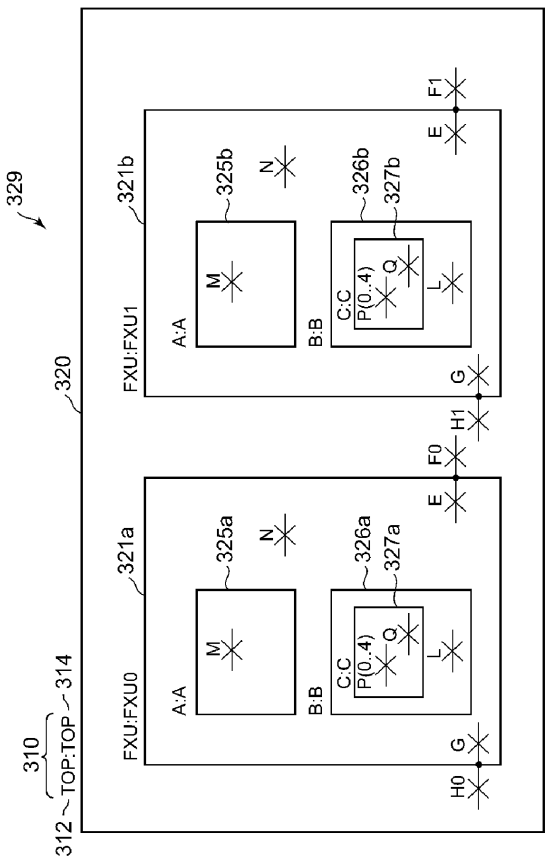
【図２】


10
16
20
24
CPU
26
28
RAM
30
ROM
32
ディスプレイ・アダプタ
33
ディスク／テープ・ドライブ
34
I/O アダプタ
36
ユーザ・インターフェース・アダプタ
38
40
42
通信アダプタ
44
メイン・メモリ
46
制御プログラム

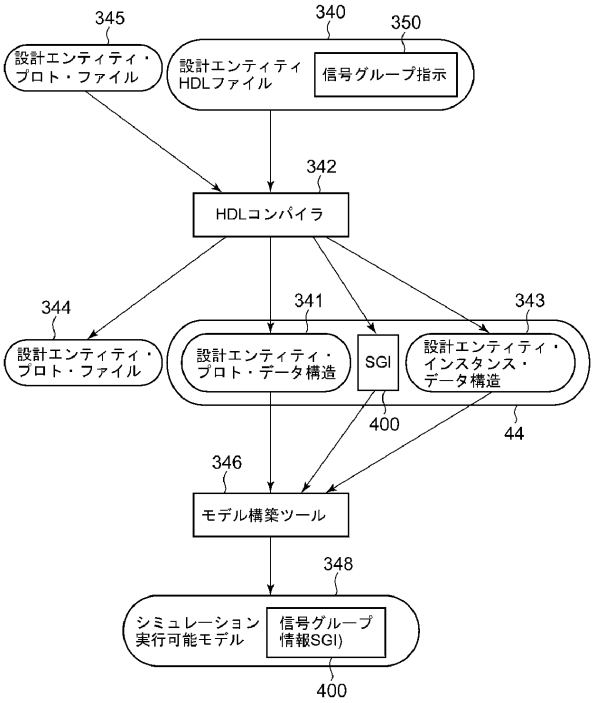
【図３】


300
301
303
304
305
306
308
309

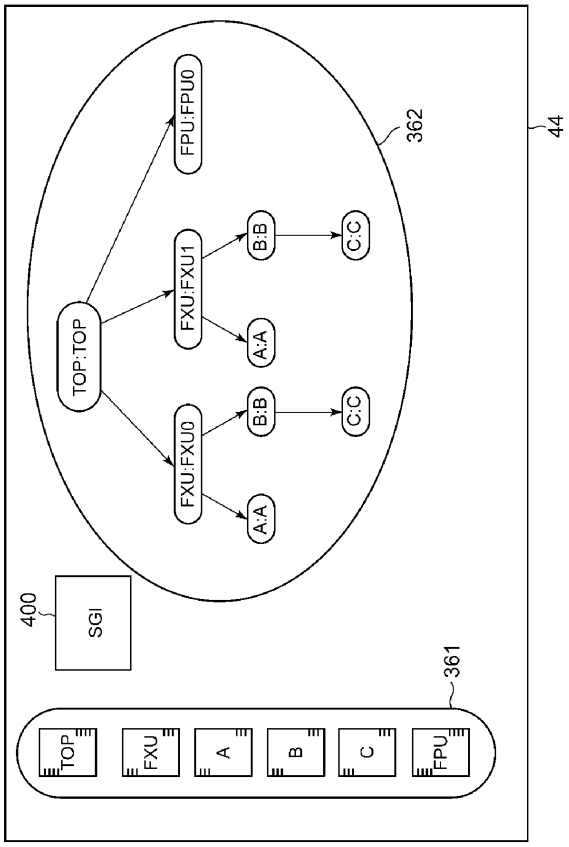
【図４】


310
312
TOP:TOP
314
320
329
FXU:FXU0
321a
A:A
B:B
C:C
P(0:4)
325a
326a
327a
M
N
Q
L
E
F0
G
H0
FXU:FXU1
321b
325b
326b
327b
F1
H1

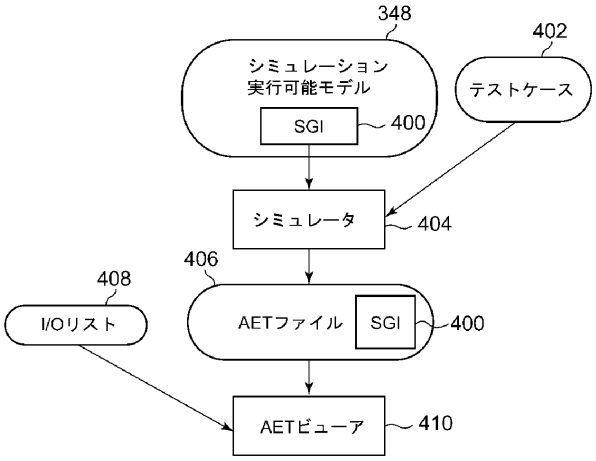
【図５】


345
340
350
設計エンティティ・プロト・ファイル
設計エンティティ HDLファイル
信号グループ指示
342
HDLコンパイラ
344
341
343
設計エンティティ・プロト・ファイル
設計エンティティ・プロト・データ構造
SGI
設計エンティティ・インスタンス・データ構造
400
44
346
モデル構築ツール
348
シミュレーション実行可能モデル
信号グループ情報SGI)
400

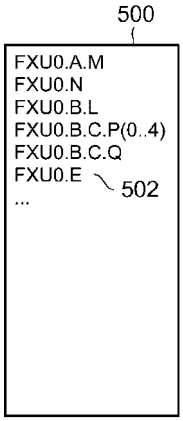
【図６】


362
44
FPU:FPU0
TOP:TOP
FXU:FXU1
B:B
C:C
A:A
FXU:FXU0
B:B
C:C
A:A
400
SGI
361
TOP
FXU
A
B
C
FPU

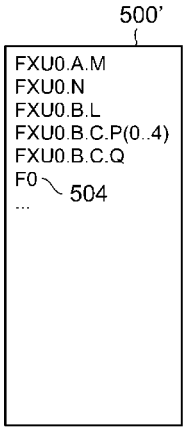
【図７】


348
シミュレーション実行可能モデル
SGI
400
402
テストケース
シミュレータ
404
406
408
I/Oリスト
AETファイル
SGI
400
AETビューア
410

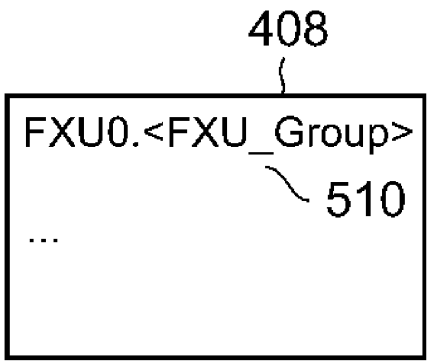
【図８】


500
FXU0.A.M
FXU0.N
FXU0.B.L
FXU0.B.C.P(0..4)
FXU0.B.C.Q
FXU0.E
...
502

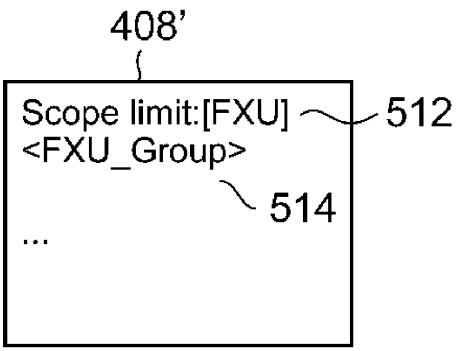
【図９】



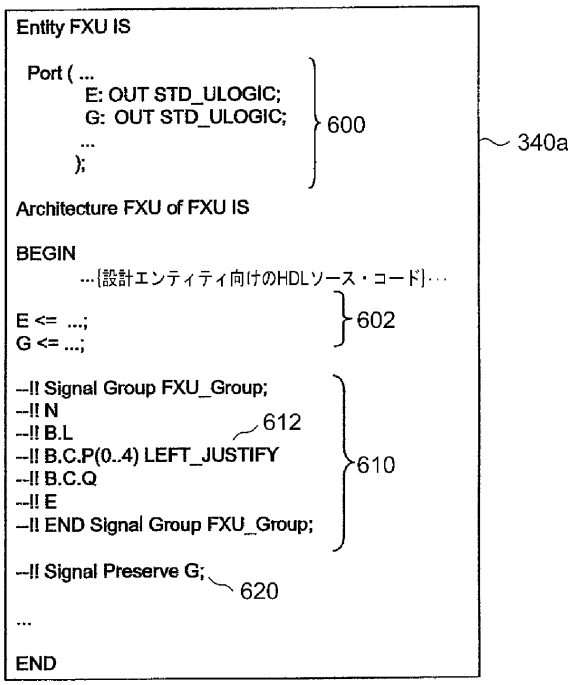
【図１０】



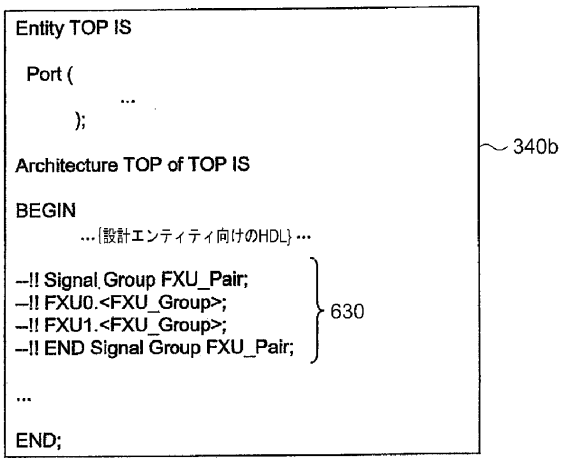
【図１１】



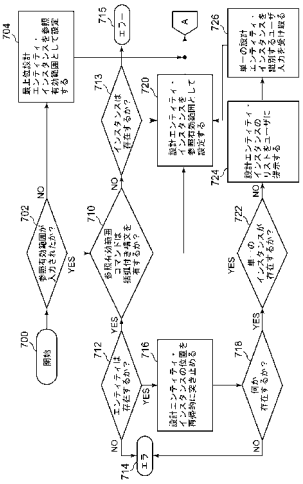
【図１２】



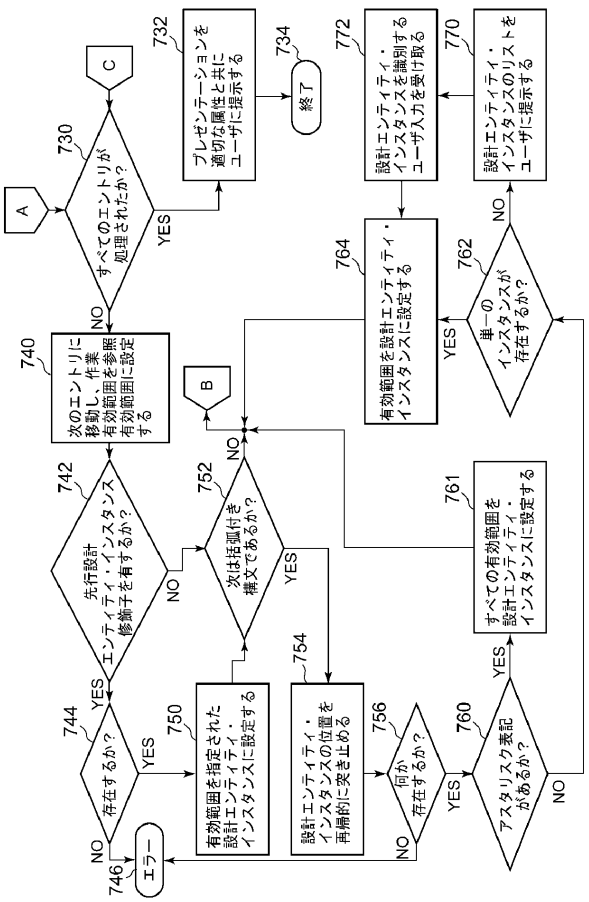
【図１３】



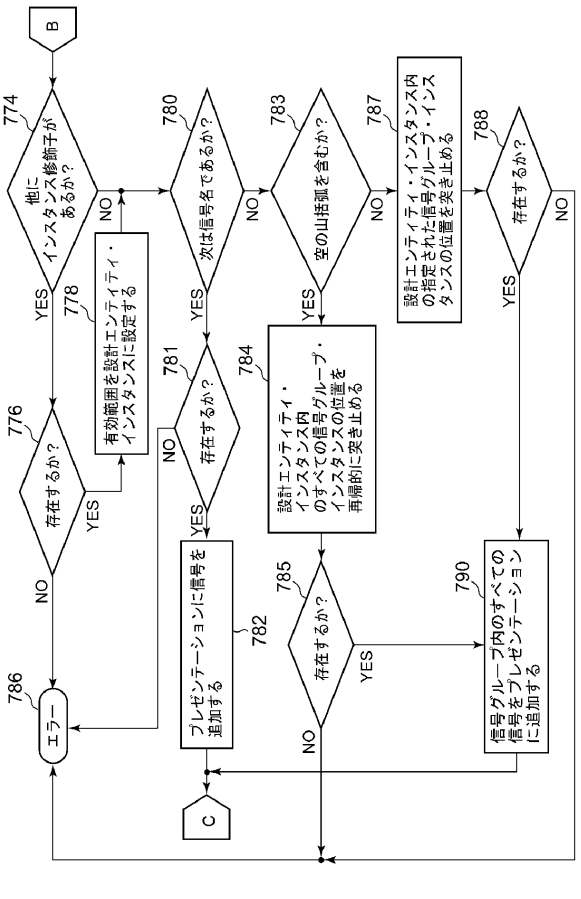
【図１４】



【図１５】



【図１６】



フロントページの続き

(74)代理人 100086243
弁理士 坂口 博
(72)発明者 ボボク、ガボール
アメリカ合衆国１２３０９ ニューヨーク州ニスカユナ ホーソーン・ロード １３４５
(72)発明者 レスナー、ウォルフガング
アメリカ合衆国７８７２６ テキサス州オースティン チェストナット・リッジ・ロード １０７１７
(72)発明者 ウィリアムズ、デレク、エドワード
アメリカ合衆国７８７１７ テキサス州オースティン スレイト・クリーク・トレイル ９４０６

審査官 松浦 功

(56)参考文献 特開平１１－０６６１１４（ＪＰ，Ａ）
特開平０３－１１６２７６（ＪＰ，Ａ）
特開平０５－１５１３０４（ＪＰ，Ａ）
特開平０５－３４２２９６（ＪＰ，Ａ）
小林優，Ｖｅｒｉｌｏｇ－HDLによるＦＰＧＡ回路設計の実際，Ｉｎｔｅｒｆａｃｅ，CQ出版株式会社，１９９６年 5月 1日，第２２巻，第4号，ｐｐ．１２８－１３７
藤谷つぐみ，パソコン上で動作するＶｅｒｉｌｏｇ HDL － ＳＩＬＯＳIII，インターフェース，CQ出版株式会社，１９９３年１２月 1日，第１９巻，第１２号，ｐｐ．１８４－１８８

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 17/50
Google Scholar