(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2011-22702
(P2011-22702A)
(43)公開日　平成23年2月3日(2011.2.3)

(51)Int.Cl.		F I		テーマコード（参考）
G06F 11/28	(2006.01)	G06F 11/28	340C	5B042

審査請求 有　請求項の数 18　ＯＬ　（全 25 頁）

(21)出願番号　特願2009-165698 (P2009-165698)
(22)出願日　平成21年7月14日 (2009.7.14)

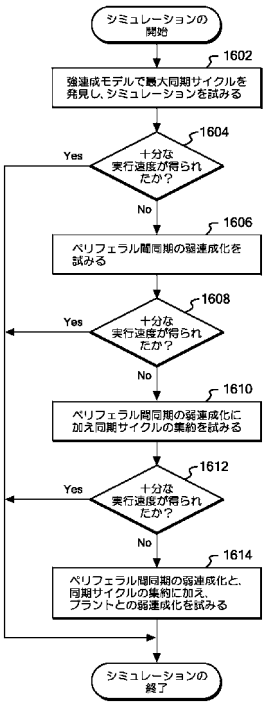
(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】シミュレーション方法、システム及びプログラム

(57)【要約】
【課題】 シミュレーションの速度を向上させること

【解決手段】 互いに通信する複数のペリフェラルをもつシミュレーション・システムにおいて、ペリフェラルをノードとし、接続路をエッジとし、通信時間を重みとする重みつきグラフを構成し、そのループにおける通信時間のうちの最小時間を第１の同期タイミングとし、許容可能な遅延を入れたタイミングを第２のタイミングとし、ユーザが指定したそれより長いタイミングを第３の同期タイミングとして、第３の同期タイミングのタイミングが使えるところではそれを使うことにより、なるべく長い同期タイミングで、ペリフェラルを同期させる。
【選択図】 図１６



【特許請求の範囲】
【請求項１】
コンピュータにより、複数の機能ブロックをもつシステムを動作させる方法であって、
前記複数の機能ブロックの接続関係と、該接続路を介しての通信時間とから、前記機能ブロックをノード、前記接続路を前記通信時間を重みとしてもつエッジとするグラフを構成するステップと、
前記グラフの閉路を見出すステップと、
前記見出された閉路のうち最小の重みをもつ閉路を決定するステップと、
前記最小の重みをもつ閉路の重みを、第１の同期タイミングとして保存するステップと、
前記第１の同期タイミングで前記複数の機能ブロックを同期させるステップを有する、
システムの動作方法。
【請求項２】
前期グラフの通信時間に対して許容可能な遅延が付け加わったグラフに対して，第２の同期タイミングを決定するステップと、
前記第２の同期タイミングで前記複数の機能ブロックを同期させるステップをさらに有する、
請求項１に記載の動作方法。
【請求項３】
前記複数の機能ブロックと外部インターフェースとの間で、第１、第２の同期タイミングより大きい、ユーザーに指定された第３の同期タイミングで同期が許容されるかを判断するステップと、
前期グラフの前期複数の機能ブロックと外部インターフェースとの間で前記第３の同期タイミングが許容可能と示された箇所に対して、前期機能ブロックと外部インターフェースとの間で前記第３の同期タイミングによって同期を行わせるステップをさらに有する、
請求項２に記載の動作方法。
【請求項４】
前記許容可能な遅延は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値により決定される、請求項２に記載の動作方法。
【請求項５】
前記同期が許容されるかの判断は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値に基づき行われる、請求項３に記載の動作方法。
【請求項６】
前記第１の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断するステップと、
前記第１の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第２の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断するステップと、
前記第２の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第３の同期タイミングで前記システムを動作させるステップをさらに有する、
請求項３に記載の動作方法。
【請求項７】
コンピュータにより、複数の機能ブロックをもつシステムを動作させるプログラムであって、
前記コンピュータをして、
前記複数の機能ブロックの接続関係と、該接続路を介しての通信時間とから、前記機能ブロックをノード、前記接続路を前記通信時間を重みとしてもつエッジとするグラフを構成するステップと、
前記グラフの閉路を見出すステップと、

前記見出された閉路のうち最小の重みをもつ閉路を決定するステップと、
前記最小の重みをもつ閉路の重みを、第１の同期タイミングとして保存するステップと、
前記第１の同期タイミングで前記複数の機能ブロックを同期させるステップを実行させる、
プログラム。
【請求項８】
前記コンピュータをして、
前期グラフの通信時間に対して許容可能な遅延が付け加わったグラフに対して，第２の同期タイミングを決定するステップと、
前記第２の同期タイミングで前記複数の機能ブロックを同期させるステップをさらに実行させる、
請求項７に記載のプログラム。
【請求項９】
前記コンピュータをして、
前記複数の機能ブロックと外部インターフェースとの間で、第１、第２の同期タイミングより大きい、ユーザーに指定された第３の同期タイミングで同期が許容されるかを判断するステップと、
前期グラフの前期複数の機能ブロックと外部インターフェースとの間で前記第３の同期タイミングが許容可能と示された箇所に対して、前期機能ブロックと外部インターフェースとの間で前記第３の同期タイミングによって同期を行わせるステップをさらに実行させる、
請求項８に記載のプログラム。
【請求項１０】
前記許容可能な遅延は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値により決定される、請求項８に記載のプログラム。
【請求項１１】
前記同期が許容されるかの判断は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値に基づき行われる、請求項９に記載のプログラム。
【請求項１２】
前記コンピュータをして、
前記第１の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断するステップと、
前記第１の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第２の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断するステップと、
前記第２の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第３の同期タイミングで前記システムを動作させるステップをさらに実行させる、
請求項９に記載のプログラム。
【請求項１３】
コンピュータにより、複数の機能ブロックを動作させるシステムであって、
前記複数の機能ブロックの接続関係と、該接続路を介しての通信時間とから、前記機能ブロックをノード、前記接続路を前記通信時間を重みとしてもつエッジとするグラフを構成する手段と、
前記グラフの閉路を見出す手段と、
前記見出された閉路のうち最小の重みをもつ閉路を決定する手段と、
前記最小の重みをもつ閉路の重みを、第１の同期タイミングとして保存する手段と、
前記第１の同期タイミングで前記複数の機能ブロックを同期させる手段を有する、
システム。

【請求項１４】
前期グラフの通信時間に対して許容可能な遅延が付け加わったグラフに対して，第２の同期タイミングを決定する手段と、
前記第２の同期タイミングで前記複数の機能ブロックを同期させる手段をさらに有する、
請求項１３に記載のシステム。
【請求項１５】
前記複数の機能ブロックと外部インターフェースとの間で、第１、第２の同期タイミングより大きい、ユーザーに指定された第３の同期タイミングで同期が許容されるかを判断する手段と、
前期グラフの前期複数の機能ブロックと外部インターフェースとの間で前記第３の同期タイミングが許容可能と示された箇所に対して、前期機能ブロックと外部インターフェースとの間で前記第３の同期タイミングによって同期を行わせる手段をさらに有する、
請求項１４に記載のシステム。
【請求項１６】
前記許容可能な遅延は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値により決定される、請求項１４に記載のシステム。
【請求項１７】
前記同期が許容されるかの判断は、前記コンピュータの記憶手段に、予め所定の機能ブロックに設定された値に基づき行われる、請求項１５に記載のシステム。
【請求項１８】
前記第１の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断する手段と、
前記第１の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第２の同期タイミングで前記システムを動作させて、所望の動作速度が得られたかどうか判断する手段と、
前記第２の同期タイミングで前記システムが動作した際に所望の動作速度が得られない場合に、前記第３の同期タイミングで前記システムを動作させる手段をさらに実行させる、
請求項１５に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、自動車などの物理システムのシミュレーションに関し、より詳しくは、ソフトウェア・ベースでのシミュレーション・システムに関するものである。
【背景技術】
【０００２】
自動車は、その初期の時代の２０世紀初頭は、動力としてのエンジンと、ブレーキ、アクセル、ハンドル、トランスミッション、サスペンジョンを含む、機構部品からなっていたが、エンジンのプラグの点火、ヘッドライト以外は、電気的な仕組みはほとんど利用していなかった。
【０００３】
ところが、１９７０年代頃から、大気汚染、石油危機などに備えて、エンジンを効率的に制御する必要性が生じ、このためエンジンの制御に、ＥＣＵが使用されるようになってきた。ＥＣＵは、一般的に、センサからの入力信号を、例えばＡ／Ｄ変換する入力インターフェースと、決められた論理に従ってディジタル入力信号を処理する論理演算部（マイクロコンピュータ）と、その処理結果を、アクチュエータ作動信号に変換する出力インターフェースとから構成される。
【０００４】
いまや、エンジンやトランスミッションなどの制御システム、Anti-lock Breaking Sys

tem （ＡＢＳ）、Electronic Stability Control （ＥＳＣ）、パワーステアリングだけでなく、ワイパー制御やセキュリティ・モニタリング・システムなどに至るまで、最近の自動車では、機構部品だけでなく、エレクトロニクス部品やソフトウエアが重要な比率を占める。後者に関する開発費は全体の２５％とも４０％とも言われ、ハイブリッド型の自動車では７０％を占める。

【０００５】

電子制御は、ＥＣＵを複数、配置して行われる。ＥＣＵ間は車載ネットワーク、例えば、Controller Area Network （ＣＡＮ） で相互に接続される。また、制御の対象である、エンジンやトランスミッションなどには、それぞれのＥＣＵから直接ワイヤ接続する。

【０００６】

ＥＣＵは、小さなコンピュータであり、センサ入力などからの割り込みに応じて動作する。一方、エンジンなどは連続的に機械的動作を行っている。すなわち、コンピュータ系のディジタル・システムと、機械系の物理システムが、自動車という単一システムにおいて、並列に協調動作を行っている。当然、これを支えるソフトウエアは複雑さがますます増大しており、ＥＣＵ単体で動作を検証するだけでなく、複数を同時に検証する仕組みの実現が要望されている。

【０００７】

一方、ＥＣＵの出力信号によって駆動されるアクチュエータには、電磁ソレノイド及びモータ等がある。ソレノイドは例えば、エンジンのインジェクタ、トランスミッションのシフト・コントロール、ブレーキのバルブ制御、ドアロックなどに使用される。

【０００８】

このようなテストにために従来行われている技法として、ＨＩＬＳ(Hardware In the Loop Simulation)がある。特に、自動車全体のＥＣＵをテストする環境は、ホールビークルＨＩＬＳ(Whole Vehicle Hardware In the Loop Simulation)と呼ばれる。ホールビークルＨＩＬＳにおいては、実験室内で、本物のＥＣＵが、エンジン、トランスミッション機構などをエミュレーションする専用のハードウェア装置に接続され、所定のシナリオに従って、テストが行われる。ＥＣＵからの出力は、監視用のコンピュータに入力され、さらにはディスプレイに表示されて、テスト担当者がディスプレイを眺めながら、異常動作がないかどうか、チェックする。

【０００９】

しかし、ＨＩＬＳは、専用のハードウェア装置を使い、それと本物のＥＣＵの間を物理的に配線しなくてはならないので、準備が大変である。また、別のＥＣＵに取り替えてのテストも、物理的に接続し直さなくてはならないので、手間がかかる。さらに、本物のＥＣＵを用いたテストであるため、テストに実時間を要する。従って、多くのシナリオをテストすると、膨大な時間がかかる。また、ＨＩＬＳのエミュレーション用のハードウェア装置は、一般に、非常に高価である。

【００１０】

そこで近年、高価なエミュレーション用ハードウェア装置を使うことなく、ソフトウェアで構成する手法が存在する。この手法は、ＳＩＬＳ(Software In the Loop Simulation)と呼ばれ、ＥＣＵに搭載されるマイクロコンピュータ、入出力回路、制御のシナリオなどを全て、ソフトウェア・シミュレータで構成する技法である。これによれば、ＥＣＵのハードウェアが存在しなくても、テストを実行可能である。

【００１１】

ところで、自動車用シミュレーション・システムは、連続系シミュレータと、離散系シミュレータを有する。連続系シミュレータの例として、エンジンの機械系部分をシミュレートするシミュレータがある。離散イベント系シミュレータの例として、エンジン回転のパルスのタイミングで動作し、燃料噴射や点火のタイミングを制御するＥＣＵのシミュレータがある。

【００１２】

４ＷＤのシミュレーションをする場合においては、連続系シミュレータの例として、各

タイヤへのトルク配分から車の動作を繰り返し計算するシミュレータがあり、離散イベント系シミュレータの例として、10ミリ秒ごとの定期パルス信号で動作し、車のヨーレートなどのセンサ入力から各タイヤへのトルク配分を決定するＥＣＵをシミュレートするシミュレータがある。

【００１３】

さらに、離散系シミュレータは、連続系シミュレータのタイムスライスとは非同期に、パルス信号入力以外に、Ｉ／Ｏポートを通じてデータの読み書きを行う。典型的には、センサからのデータを読み込み、更新する。

【００１４】

典型的な離散系シミュレータである、ＥＣＵエミュレータは、ＣＰＵエミュレータと、ＣＰＵエミュレータと連続系シミュレータの間をインターフェースするペリフェラルとからなる。

【００１５】

このようなシミュレータで、特に問題となるのは、ペリフェラルをシミュレートすることである。というのは、特にペリフェラルは種類が多く、その動作タイミングもまちまちなので、正確に動作するようにシミュレーション・システムを構成することが困難である。

【００１６】

１つの解決策は、ＥＣＵベンダが提供するハードウェア記述をそのまま利用して、忠実にステップ毎にペリフェラル・エミュレータを実行させることである。

【００１７】

TLM 2.0 user manual, June 2008の3.3.2章には、Loosely-timed coding style and temporal decouplingに関する記述がある。

【００１８】

特開２００１－１０１１５６号公報は、最適シミュレーションパラメータ自動推定システムにおいて、生データとシミュレーションデータとの一致性を客観的に判断できるようにするために、出発値、可変ステップ幅、終了値を設定し、パラメータ可変用テーブルを作成するパラメータ可変用設定ダイヤログウインドウと、シミュレーション計算結果と比較データとの類似度を評価するデータ比較モジュールとを備え、パラメータ可変用テーブルの条件全てについて、シミュレーション計算を行って計算結果と比較データとの類似度を評価して最適シミュレーションパラメータを推定することを開示する。

【００１９】

特開２００２－３１８９９０号公報は、シミュレーション・システムにおいて、ｊステップ目まで既知の状態で、処理が実行され、ｊ＋１ステップ目の予測子及び修正子を計算し、誤差の推定値が誤差の許容値より小さいか判定し、その判定が“Ｙｅｓ”であれば、次のステップ幅又は解を計算し、処理を１ステップ進め、その判定が“Ｎｏ”であれば、ステップ幅を縮小するか、又は、処理を１ステップ戻すことを開示する。

【先行技術文献】

【特許文献】

【００２０】

【特許文献１】特開２００１－１０１１５６号公報

【特許文献２】特開２００２－３１８９９０号公報

【非特許文献】

【００２１】

【非特許文献１】TLM 2.0 user manual, June 2008

【発明の概要】

【発明が解決しようとする課題】

【００２２】

上記従来技術には、予め既定よりも緩いタイミングでシミュレーション・システムを動作させること、及び、必要に応じて、その動作条件を変化させることは開示されているが

、ペリフェラルなどの動作条件の異なる複数の機能ブロックの間の動作タイミングを適合させることは開示も示唆もされていない。

【００２３】

従って、この発明の目的は、ＥＣＵのペリフェラルなどの動作条件の異なる複数の論理ブロックをもつシミュレーション・システムにおいて、互いに接続される論理ブロック間の同期タイミングを適合させることによって、動作速度を向上させることにある。

【課題を解決するための手段】

【００２４】

この発明の前提としてまず、典型的にはＥＣＵのペリフェラル・モジュールである、論理ブロックのハードウェア仕様が、SystemCなどの記述言語により記述される。

【００２５】

そのように記述されたSystemCのハードウェア仕様が、所定のスキャンツールによってスキャンされ、これによって、ペリフェラル・モジュールをノードとし、ペリフェラル・モジュール間の通信時間を重みとしてもつエッジからなるグラフが、好適にはコンピュータのメモリ上に作成される。

【００２６】

このようにして作成されたグラフが、コンピュータ上で探索され、ループが抽出される。抽出されたループについて、エッジの重みに基づきパス長が計算され、こうして計算されたパス長のうちの最小のものが、ΔT_periとして、コンピュータのメモリに記憶される。この発明では、ΔT_periでの同期サイクルでの動作を、強連成モデルでの動作と呼ぶ。

【００２７】

次に、同様にして抽出されたループについて、仕様や経験に基づきユーザが設定した許容される遅延により、より緩められた同期の値が求められ、そのうちの最小のものが、ΔT_peri'として、コンピュータのメモリに記憶される。この発明では、ΔT_peri'での同期サイクルでの動作を、弱連成モデルでの動作と呼ぶ。

【００２８】

本発明によれば次に、あるループ内にあるペリフェラルと、別のループにあるペリフェラルとの間で、ΔT_periやΔT_peri'よりも大きいΔT_majorで通信することが可能かどうかがチェックされ、もしそうなら、ΔT_majorで通信される。この発明では、あるループ内にあるペリフェラルと、別のループにあるペリフェラルとの間で、許容された大きい同期サイクルで動作することを、ペリフェラル間の同期サイクルの集約モデルでの動作と呼ぶ。

【００２９】

本発明によればさらに、あるペリフェラルとプラント間の通信が、ある集約された同期サイクルΔT_majorで通信することが可能かどうかがチェックされ、もしそうなら、ΔT_majorで通信される。この発明では、あるペリフェラルと、プラントの間で、許容された大きい同期サイクルで動作することを、ペリフェラルとプラントの同期サイクルの集約モデルでの動作と呼ぶ。

【００３０】

このようなスキームが用意されると、シミュレーション・システムではまず、強連成モデルでの動作が試みられ、それで所望の動作速度が得られるなら、強連成モデルでシミュレーション・システムが実行される。

【００３１】

そうでないなら次に、弱連成モデルでの動作が試みられ、それで所望の動作速度が得られるなら、弱連成モデルでシミュレーション・システムが実行される。

【００３２】

そうでないなら次に、ペリフェラル間の同期サイクルの集約モデルでの動作が試みられ、それで所望の動作速度が得られるなら、ペリフェラル間の同期サイクルの集約モデルでシミュレーション・システムが実行される。

【００３３】

そうでないなら、ペリフェラルとプラントの同期サイクルの集約モデルで、シミュレーション・システムが実行される。

【発明の効果】

【００３４】

この発明によれば、シミュレーション・システムにおいて、仕様記述から同期タイミングを次第に緩和していくようにして、動作の正確さを維持しつつ、可能な最大の動作速度を達成することが可能となる。

【図面の簡単な説明】

【００３５】

【図１】ＥＣＵの典型的な制御である、フィードバック閉ループ系の例を示す図である。

【図２】応答関数による記述の例である。

【図３】本発明を実施するためのハードウェア構成のブロック図である。

【図４】本発明を実行するためのデータ、モジュール、及び実行環境の機能ブロック図である。

【図５】ＥＣＵエミュレータとプラント・シミュレータの同期実行を示す図である。

【図６】プラント・シミュレータとペリフェラル・エミュレータの間の通信、及びペリフェラル・エミュレータ同士の通信に限定して図式的に示す図である。

【図７】ΔT_periでの同期サイクルでの動作を、強連成モデルでの同期サイクルΔT_periを求める処理のフローチャートを示す図である。

【図８】最大同期サイクルを説明するための図である。

【図９】強連成モデルから得られた同期サイクルによる実行の処理を示すフローチャートであり、特に実行時スケジューラの場合を示す。

【図１０】強連成モデルから得られた同期サイクルによる実行の処理を示すフローチャートであり、特に静的スケジューラの場合、すなわち、あらかじめ実行の順番などを決めてペリフェラル実行コードをディスパッチするコードを予め生成し，それを実行する方式の場合を示す。

【図１１】弱連成化のための緩和された同期サイクルを説明するための図である。

【図１２】同期サイクルの集約について説明する図である。

【図１３】実行時スケジューラの場合の処理を示すフローチャートである。

【図１４】静的スケジューラの場合の処理を示すフローチャートである。

【図１５】プラントとの同期の弱連成化を説明する図である。。

【図１６】処理の全体のシナリオを説明する図である。

【発明を実施するための形態】

【００３６】

以下、図面を参照して、本発明の一実施例の構成及び処理を説明する。以下の記述では、特に断わらない限り、図面に亘って、同一の要素は同一の符号で参照されるものとする。なお、ここで説明する構成と処理は、一実施例として説明するものであり、本発明の技術的範囲をこの実施例に限定して解釈する意図はないことを理解されたい。

【００３７】

本発明を実現するための構成を説明する前に、その前提として、ＥＣＵについて説明する。ＥＣＵは、一般的に、センサからの入力信号を、例えばＡ／Ｄ変換する入力インターフェースと、決められた論理に従ってディジタル入力信号を処理する論理演算部（マイクロコンピュータ）と、その処理結果を、アクチュエータ作動信号に変換する出力インターフェースとから構成されるものである。

【００３８】

この発明は、説明の便宜上、以下では、自動車のＥＣＵに関連して説明するが、それには限定されず、航空機、ロボットなどその他のＥＣＵをもつメカトロニクス機構全般に適用可能であることを理解されたい。

【００３９】

ＥＣＵは、周辺や環境状態、エンジンなどの駆動機構の状態、及び人間による指示操作

の内容をセンサで検出して、信号として入力する。具体的には、水温センサ、吸気温センサ、過給圧センサ、ポンプ角センサ、クランク角センサ、車速センサ、アクセル位置センサ、Ａ／Ｔシフト・ポジション、スタータ・スイッチ、エアコンＥＣＵなどからの信号がある。

【００４０】

ＥＣＵは、これらの信号を入力して、電磁スピル弁、フュエル・カット・ソレノイド、タイミング・コントロール・バルブ、吸気絞りＶＳＶ、グロー・プラグ・リレー、タコメータ及びエアコン・リレーなどを駆動する信号を出力しする。

【００４１】

１つのＥＣＵが複数の異なる機構を制御するための駆動信号を出力するようにすることは不可能ではないが、例えば、エンジンとエアコンのように、応答性やその制御の厳密性が異なるものを単一のＥＣＵで制御することは合理的でなく、従って、一般的に自動車にＥＣＵは複数個設けられる。

【００４２】

図１は、ＥＣＵの典型的な制御である、フィードバック閉ループ系の例を示す図である。すなわち、図１において、ある目標の信号が、ＥＣＵであるコントローラ１０２に入力され、ＥＣＵは、目標の信号を内部処理することによって、駆動信号を出力し、制御対象モデルである、エンジンなどのプラント１０４を駆動し、プラント１０４の出力は、センサ１０６を介して、コントローラ１０２の入力にフィードバックされる。

【００４３】

ここで目標信号として与えられるのは、例えば、スロットル開度、アイドル・コントロール、ブレーキ力、シフト、スタータＯＮ・ＯＦＦ、バッテリ電圧、インジェクション通電時間、インジェクション通電回数、デポジット、ドウェル角、進角値、吸気完了フラグ、点火完了フラグ、大気圧、車両重量、転がり抵抗係数、道路勾配、粘着係数、吸気温、などのパラメータである。

【００４４】

また、センサ信号としてフィードバックされるのは、スロットル開度、吸気圧力、吸入空気量、シフト、エンジン回転数、車速、排気温、O_2、冷却水温、空燃比、ノック、点火異常、などである。

【００４５】

ＥＣＵが制御する対象は、ニュートンの力学方程式で解かれる、機構系システムであったり、電気回路の応答方程式で解かれる、電気駆動回路であったり、それらの組み合わせであったりする。これらは、基本的に微分方程式であり、制御工学によれば、ラプラス変換によって応答関数に変換されて、記述することができる。

【００４６】

図２は、そのような応答関数による記述の例である。図４で破線２０２で囲った箇所が、図１のコントローラ１０２に対応し、破線２０４で囲った箇所が、図１の制御対象モデル１０４に対応し、センサ１０６が、ブロック２０６に対応する。なお、図２は、応答関数による表現の一例であって、特に本発明を限定する意図はないことを理解されたい。

【００４７】

次に、図３を参照して、本発明を実施するために使用されるコンピュータのハードウェアについて説明する。図３において、ホスト・バス３０２には、複数のＣＰＵ０　３０４ａ、ＣＰＵ１　３０４ｂ、ＣＰＵ２　３０４ｃ、ＣＰＵ３　３０４ｄが接続されている。ホスト・バス３０２にはさらに、ＣＰＵ０　３０４ａ、ＣＰＵ１　３０４ｂ、ＣＰＵ２　３０４ｃ、ＣＰＵ３　３０４ｄの演算処理のためのメイン・メモリ３０６が接続されている。

【００４８】

一方、Ｉ／Ｏバス３０８には、キーボード３１０、マウス３１２、ディスプレイ３１４及びハードティスク・ドライブ３１６が接続されている。Ｉ／Ｏバス３０８は、Ｉ／Ｏブリッジ３１８を介して、ホスト・バス３０２に接続されている。キーボード３１０及びマ

ウス３１２は、オペレータが、コマンドを打ち込んだり、メニューをクリックするなどして、操作するために使用される。ディスプレイ３１４は、必要に応じて、後述する本発明に係るプログラムをＧＵＩで操作するためのメニューを表示するために使用される。

【００４９】

この目的のために使用される好適なコンピュータ・システムのハードウェアとして、ＩＢＭ（Ｒ）Ｓｙｓｔｅｍ Ｘがある。その際、ＣＰＵ０ ３０４ａ、ＣＰＵ１ ３０４ｂ、ＣＰＵ２ ３０４ｃ、ＣＰＵ３ ３０４ｄは、例えば、インテル（Ｒ）Ｃｏｒｅ ２ ＤＵＯであり、オペレーティング・システムは、Ｗｉｎｄｏｗｓ（商標）Ｓｅｒｖｅｒ ２００３である。オペレーティング・システムは、ハードディスク・ドライブ３１６に格納され、コンピュータ・システムの起動時に、ハードディスク・ドライブ３１６からメイン・メモリ３０６に読み込まれる。

【００５０】

ここで図示されているＣＰＵの個数は４個であるが、これに限定されず、シングル・プロセッサのシステムであってもよく、あるいは、任意の個数のマルチコア、またはマルチプロセッサのシステムであってもよい。

【００５１】

なお、本発明を実施するために使用可能なコンピュータ・システムのハードウェアは、ＩＢＭ（Ｒ）Ｓｙｓｔｅｍ Ｘに限定されず、本発明のシミュレーション・プログラムを走らせることができるものであれば、任意のコンピュータ・システムを使用することができる。オペレーティング・システムも、Ｗｉｎｄｏｗｓ（Ｒ）に限定されず、Ｌｉｎｕｘ（Ｒ）、Ｍａｃ ＯＳ（Ｒ）など、任意のオペレーティング・システムを使用することができる。さらに、ＥＣＵエミュレータ・プログラム、プラント・シミュレータなどの論理プロセスを高速で動作させるために、ＰＯＷＥＲ（商標）６ベースで、オペレーティング・システムがＡＩＸ（商標）のＩＢＭ（Ｒ）Ｓｙｓｔｅｍ Ｐなどのコンピュータ・システムを使用してもよい。

【００５２】

ハードディスク・ドライブ３１６にはさらに、ＥＣＵエミュレータ、プラント・シミュレータなどの複数の論理プロセス、及び、複数の論理プロセスを協働して動作させるためのプログラムが格納され、キーボード３１０及びマウス３１２によって起動操作可能である。

【００５３】

好適には、フルビークルＳＩＬＳを実現するために、１台の自動車で使われるすべてのＥＣＵエミュレータ・プログラムと、ＥＣＵエミュレータ・プログラムとプラント・シミュレータとのインターフェースを与えるペリフゥラル・エミュレータが、ハードディスク・ドライブ３１６に保存されている。

【００５４】

ハードディスク・ドライブ３１６にはさらに、図４に関連して後述するデータ及びプログラム・モジュール、及びプラント・シミュレータも格納されている。

【００５５】

なお、ここでの「エミュレータ」と、「シミュレータ」の用語の使い分けであるが、もともとの、別のプロセッサで動くことを想定して書かれていたＥＣＵのコードを、ＣＰＵ０～ＣＰＵ３などをターゲットとして動くようにすることを、エミュレーションと呼び、それを行うプログラムを、エミュレータと呼ぶ。一方、エンジンなどの物理的システムの動作を仮想計算するシステムを、シミュレータと呼ぶ。

【００５６】

図４は、本発明を実行するためのデータ、モジュール、及び実行環境の機能ブロック図である。図４において、ハードウェア仕様４０２とは、コンピュータ可読にハードディスク・ドライブ３１６などに保存されたファイルであって、例えば、SystemCで記述されている。以下に、SystemCの記述例を示す。

【００５７】

下記で、通信にかかる遅延時間は、peripheral_a.cpp 16行目に記述:
sc_time* time = new sc_time(50, SC_NS); // communication delay is 50nsに記述
【００５８】
AからBへの通信は、peripheral_a.cpp 24行目に記述:
i_socket->b_transport(*trans, *time);
【００５９】
BからAへの通信は、peripheral_a.cppのb_transportからのreturnに相当する
【００６０】
Bの処理時間と通信にかかる遅延時間は、peripheral_b.cpp 12行目に記述：
sc_time* delay = new sc_time(100, SC_NS); // processing time and communication delay are 100ns
【００６１】

```
// peripheral_a.h
#ifndef PERIPHERAL_A_H_
#define PERIPHERAL_A_H_
#include <systemc>
#include <tlm.h>
using namespace sc_core;
using namespace sc_dt;
using namespace std;
using namespace tlm;
SC_MODULE(PERIPHERAL_A), tlm_bw_transport_if<>
{
public:
  tlm_initiator_socket<32> i_socket;
  SC_HAS_PROCESS(PERIPHERAL_A);
  PERIPHERAL_A(sc_module_name);
protected:
  virtual tlm_sync_enum nb_transport_bw(tlm_generic_payload&, tlm_phase&, sc_tim
e&);
  virtual void invalidate_direct_mem_ptr(uint64, uint64);
  void thread0(void);
};
// end of peripheral_a.h
#endif
```

【００６２】

```
// peripheral_a.cpp
#include "peripheral_a.h"
PERIPHERAL_A::PERIPHERAL_A(sc_module_name name)
  : sc_module(name)
  , i_socket("i_socket")
{
  i_socket.bind(*this);

  SC_THREAD(thread0);
}
void PERIPHERAL_A::thread0(void)
{
  tlm_generic_payload* trans = new tlm_generic_payload;
  int write_value = 100;
```

```
  sc_time* time = new sc_time(50, SC_NS); // communication delay is 50ns
  trans->set_address(0x0);
  trans->set_data_ptr(RCAST<unsigned char *>(&write_value));
  trans->set_data_length(4);
  trans->set_write();
  // communication start
  i_socket->b_transport(*trans, *time);

  // the value of "time" is 150ns here
  cout << time->to_string() << endl;
  if (trans->is_response_error()) {
    //error processing
  }
  delete trans;
  delete time;
}
tlm_sync_enum PERIPHERAL_A::nb_transport_bw(tlm_generic_payload& trans, tlm_phas
e& phase, sc_time& t)
{
  trans.set_response_status(TLM_GENERIC_ERROR_RESPONSE);
  return TLM_COMPLETED;
}
void PERIPHERAL_A::invalidate_direct_mem_ptr(uint64 a, uint64 b)
{
}
// end of peripheral_a.cpp
```

【００６３】

```
// peripheral_b.h
#ifndef PERIPHERAL_B_H
#define PERIPHERAL_B_H
#include <systemc>
#include <tlm.h>
using namespace sc_core;
using namespace sc_dt;
using namespace std;
using namespace tlm;
SC_MODULE(PERIPHERAL_B), tlm_fw_transport_if<>
{
public:
  tlm_target_socket<32> t_socket;
  SC_HAS_PROCESS(PERIPHERAL_B);
  PERIPHERAL_B(sc_module_name);
protected:
  virtual void b_transport(tlm_generic_payload&, sc_time&);
  virtual tlm_sync_enum nb_transport_fw(tlm_generic_payload&, tlm_phase&, sc_tim
e&);
  virtual unsigned int transport_dbg(tlm_generic_payload&);
  virtual bool get_direct_mem_ptr(tlm_generic_payload&, tlm_dmi&);
};
#endif
```

```
// end of peripheral_b.h
```
【００６４】
```
// peripheral_b.cpp
#include "peripheral_b.h"
PERIPHERAL_B::PERIPHERAL_B(sc_module_name name)
  : sc_module(name)
  , t_socket("t_socket")
{
  t_socket.bind(*this);
}
void PERIPHERAL_B::b_transport(tlm_generic_payload& trans, sc_time& t)
{
  sc_time* delay = new sc_time(100, SC_NS); // processing time and communication
 delay are 100ns
  // process descriptions of peripheral_b
  t = t + *delay;
  delete delay;
}
tlm_sync_enum PERIPHERAL_B::nb_transport_fw(tlm_generic_payload& trans, tlm_phas
e& phase, sc_time& t)
{
  return TLM_COMPLETED;
}
unsigned int PERIPHERAL_B::transport_dbg(tlm_generic_payload& trans)
{
  return 0;
}
bool PERIPHERAL_B::get_direct_mem_ptr(tlm_generic_payload& trans, tlm_dmi& dmi)
{
  return false;
}
// end of peripheral_b.cpp
```
【００６５】
```
// main.cpp
#include "peripheral_a.h"
#include "peripheral_b.h"
#define deltaT 10.0
#define ATUII_NUM 5
int sc_main(int argc, char **argv)
{
  PERIPHERAL_A *peri_a;
  PERIPHERAL_B *peri_b;
  peri_a = new PERIPHERAL_A("A");
  peri_b = new PERIPHERAL_B("B");
  // connect from peri_a to peri_b
  peri_a->i_socket(peri_b->t_socket);
  sc_start(200, SC_NS);
  delete peri_a;
  delete peri_b;
  return 0;
```

```
}
// end of main.cpp
```

【００６６】

モジュール間遅延データ４０４は、ユーザが設定した、所定のペリフェラル間、あるいは所定のペリフェラルと所定のプラントの間の許容可能な遅延時間を含むデータの集まりであり、好適には、ＣＳＶ、あるいは適当なデータベースのフォーマットでハードディスク・ドライブ３１６に保存され、後で、ペリフェラルのＩＤ、プラントのＩＤなどで、設定された許容可能な遅延時間を検索可能である。

【００６７】

同期タイミング計算モジュール４０６は、ハードディスク・ドライブ３１６に保存されたハードウェア仕様４０２及びモジュール間遅延データ４０４を参照して、後述する、強連成モデル、弱練成モデル、ペリフェラル間の同期サイクルの集約モデル、または、ペリフェラルとプラントの同期サイクルの集約モデルにおける同期タイミングを計算する。

【００６８】

ハードディスク・ドライブ３１６にはさらに、ＥＣＵエミュレータ４０８と、プラント・シミュレータ４１４が保存されており、コンピュータの起動時に、メイン・メモリ３０６にロードされて、オペレーティング・システムにより実行される。このような実行環境を、参照符号４１６で示す。

【００６９】

ＥＣＵエミュレータ４０８は、実際は複数のＥＣＵエミュレータからなっており、電磁スピル弁、フュエル・カット・ソレノイド、タイミング・コントロール・バルブ、吸気絞りＶＳＶ、グロー・プラグ・リレー、タコメータ及びエアコン・リレーなどに個々に対応する。個々のＥＣＵエミュレータは、ＣＰＵエミュレータ４１０と、ペリフゥラル・エミュレータ４１２からなり、ペリフゥラル・エミュレータ４１２がＣＰＵエミュレータ４１０とプラント・シミュレータ４１４の間のインターフェースを与える。

【００７０】

プラント・シミュレータ４１４もまた、実際は複数のプラント・シミュレータからなっており、エンジン・シミュレータ、トランスミッション、エアコン、ブレーキ、ワイパーなどに個々に対応する。

【００７１】

図示されているように、実行環境４１６は、同期タイミング計算モジュール４０６を参照して、ペリフゥラル・エミュレータ間、またはペリフゥラル・エミュレータとプラント・シミュレータの間の同期タイミングを設定しつつ、シミュレーション・システムを動作させる。

【００７２】

図５は、ＥＣＵエミュレータとプラント・シミュレータの同期実行を示す図である。図５において、ＥＣＵエミュレータ５０２及び５０４は、図４で、ＥＣＵエミュレータ４０８と示したものと同一であり、例えば、ＥＣＵエミュレータ５０２は、ＣＰＵエミュレータ５０２ａと、ペリフェラル・エミュレータ５０２ｂとからなる。

【００７３】

図５には、ＥＣＵエミュレータは２個しか示されていないが、実際のホールビークルＳＩＬＳを構成する際には、４０個以上接続されることもある。

【００７４】

ＣＰＵエミュレータ５０２ａは、プラント・シミュレータ５０６と、ペリフェラル・エミュレータ５０２ｂを介して通信する。プラント・シミュレータ５０６と、ペリフェラル・エミュレータ５０２ｂの間は、メイン・メモリ３０６の指定された共有メモリ領域を使うか、またはＣＡＮエミュレータ（図示しない）を介して行なわれる。

【００７５】

図６は、プラント・シミュレータとペリフェラル・エミュレータの間の通信、及びペリフェラル・エミュレータ同士の通信に限定して図式的に示す図である。ここには、プラン

ト・シミュレータ６０２と、ペリフェラル・エミュレータ６０４～６１０が示されている。

【００７６】

図６において、プラント・シミュレータ６０２は、1μs毎にペリフェラル・エミュレータＡ６０４と通信するものとする。そこで、実機における1秒分のシミュレーションを考慮すると、5ns毎に各ペリフェラル・エミュレータが動作するとすると、実機における1秒分に対応して、シミュレーションに20.40sの時間がかかる。

【００７７】

そこで、ペリフェラル・エミュレータ間の同期コストを削減することにより、100ns毎に各ペリフェラル・エミュレータが動作するとすると、実機における1秒分に対応して、シミュレーションにかかる時間は、5.85sまで減る。

【００７８】

そこでさらに、ペリフェラル・エミュレータＡ６０４からプラント・シミュレータ６０２への通信を、1μs毎に１回にすることにより、シミュレーションにかかる時間は、1.60sまでに減り、これは実機の実時間に迫る速度となる。本発明は、このようなシミュレーション速度の向上を意図するものである。

【００７９】

図７は、ΔT_periでの同期サイクルでの動作を、強連成モデルでの同期サイクルΔT_periを求める処理のフローチャートを示す図である。この処理は、図４に示す同期タイミング計算モジュール４０６によって実行される。

【００８０】

図７のステップ７０２において、同期タイミング計算モジュール４０６は、ハードディスク・ドライブ３１６に保存されている、好適にはSystemCのソースコードであるハードウェア仕様４０２から、ペリフェラルの接続情報を抽出する。

【００８１】

次のステップ７０４で、同期タイミング計算モジュール４０６は、抽出した情報に基づき、ペリフェラルをノード、ペリフェラル間の接続をエッジとするグラフを生成して、そのデータ構造を含む情報を、メイン・メモリ３０６に保存する。このとき、エッジには、仕様に基づく、ペリフェラル間の処理時間が重みとして関連付けられ、従って、グラフは重み付きグラフである。

【００８２】

次のステップ７０６では、同期タイミング計算モジュール４０６が、ステップ７０４で生成されたグラフからループを抽出する。

【００８３】

すなわち、求めたいグラフ上の経路を$(p_0, \ldots, p_k)$とする。そして、ステップ７０４で生成されたグラフ$G \equiv (V,E)$　ここで、VはグラフGのノードの集合、EはグラフGのエッジの集合とすると、このようなループは、下記の条件を満たすものとして抽出される。

$p_i \in V \ (0 <= i <= k)$

$(p_j p_{j+1}) \in E \ (0 <= j <= k)$

$p_0 = p_k$

【００８４】

このようにして求められたループ内の仕様の処理時間を加えた値に基づき、ステップ７０８で、ΔT_periが求められる。ΔT_periの求め方については、図８を参照して、より詳細に説明する。

【００８５】

ステップ７１０では、個々のループで計算したΔT_periのうちの最小のものが、改めてΔT_periとして、後のシミュレーション実行処理のために、メイン・メモリ３０６、またはハードディスク・ドライブ３１６に保存される。

【００８６】

次に、図８を参照して、具体的にΔT_periを求める方法について説明する。先ず、図８

(a)のように、ペリフェラル１とペリフェラル２からなるループが見つかったとする。そして、ペリフェラル１からペリフェラル２への通信の仕様上の時間が100nsで、ペリフェラル２からペリフェラル１への通信の仕様上の時間も100nsであるとする。

【００８７】

すると、ペリフェラル１からペリフェラル２へ通信され、それに応答して、ペリフェラル１からペリフェラル２へ通信されることのトータルで、少なくとも200nsという時間がかかる。

【００８８】

そこで、同期サイクルを200nsより大きくとるとする。というのは、同期サイクルを大きくとるほど、シミュレーション速度を向上することができるからである。

【００８９】

しかし、200nsより大きい同期サイクルには、ペリフェラル１からの送信という事象と、ペリフェラル１のペリフェラル２からの受信という事象がともに含まれ得る。ところが、１つの同期サイクル内で起こる事象は、前後関係を確認することができないので、１つの同期サイクル内で同一のペリフェラルに関する異なる２つ以上の事象が複数含まれると、それらの間の因果関係を保証できなくなる。すなわち、図８(b)にあるような「不正確な同期サイクル」になってしまう。

【００９０】

そこで、同期サイクルを縮めて、200nsより短くすると、図８(b)に示すように、ペリフェラル１からの送信という事象と、ペリフェラル１のペリフェラル２からの受信という事象が別の同期サイクルに収まるようになって、因果関係が維持されるようになる。このような同期サイクルは、前述のように、最大同期サイクルΔT_periと呼ばれる。

【００９１】

図９は、強連成モデルから得られた同期サイクルによる実行の処理を示すフローチャートであり、特に実行時スケジューラの場合を示す。言い換えると、ペリフェラル実行コードを、グリーンスレッド・ライブラリなどの実行時にディスパッチする方式である。ここで、グリーンスレッドとは、オペレーティング・システムのサポートによらないユーザー空間のスレッドのことをいい、グリーンスレッドを動かすための実行時ライブラリをグリーンスレッド・ライブラリと呼ぶ。以下の処理は、図４の実行環境４１６によって実行される。

【００９２】

さて、図９において、ステップ９０２では、ペリフェラルが１つ選ばれる。次のステップ９０４では、選ばれたペリフェラルが実行される。

【００９３】

ステップ９０６では、選ばれたペリフェラルが、同期サイクル分ΔT_peri実行したかどうかが判断される。ここでの実行単位は、図８(b)でのクロック１つ分である。こうして、ペリフェラルが、同期サイクル分ΔT_peri実行されると、ステップ９０８では、すべてのペリフェラルを今回の同期サイクル分、すなわち、ΔT_peri分実行させたかどうかが判断され、もしそうでなければ、ステップ９０２に戻って次のペリフェラルを選ぶ。

【００９４】

もしそうであるなら、ステップ９１０に行って次の同期サイクルに進む。すなわち、同期サイクル分、シミュレーションの時間を進める。

【００９５】

図１０は、強連成モデルから得られた同期サイクルによる実行の処理を示すフローチャートであり、特に静的スケジューラの場合、すなわち、あらかじめ実行の順番などを決めてペリフェラル実行コードをディスパッチするコードを予め生成し，それを実行する方式の場合を示す。

【００９６】

図１０において、ステップ１００２では、ペリフェラル１を実行する。ステップ１００４では、同期サイクル分実行したかどうかが判断され、そうでないならステップ１００２

に戻ってペリフェラル１の実行を続け、ペリフェラル１が同期サイクル分実行されたと判断されると、次にペリフェラル２についての同様の処理へと進む。

【００９７】

こうして、ペリフェラル１、ペリフェラル２、・・・ペリフェラルＮと順次進み、ステップ１００６でペリフェラルＮを実行する。ステップ１００８では、同期サイクル分実行したかどうかが判断され、そうでないならステップ１００６に戻ってペリフェラルＮの実行を続け、ペリフェラルＮが同期サイクル分実行されたと判断されると、次にステップ１０１０に進んで、次の同期サイクル、例えば、弱連成化同期サイクルへと進む。

【００９８】

次に、ペリフェラル間同期の弱連成化について説明する。あるサイクルで同期するシミュレーションでは、ペリフェラル間の発生時刻が遅れるが、そのことが分かっているユーザが予め、特定のペリフェラル間の許容可能な遅延を、図４のモジュール間遅延データ４０４に記述しておく。

【００９９】

すると、同期タイミング計算モジュール４０６は、図７のフローチャートの処理と同様の処理で見出されたループにおけるペリフェラル間の許容可能な遅延をルックアップし、その許容可能な遅延も含めて、緩和した同期サイクルΔT_peri'を計算する。

【０１００】

先ず、図１１(a)に示すように、ペリフェラル１とペリフェラル２からなるループが見つかったとする。そして、ペリフェラル１からペリフェラル２への通信の仕様上の時間が100nsで、ペリフェラル２からペリフェラル１への通信の仕様上の時間も100nsであるとする。さらにまた、ペリフェラル１からペリフェラル２への通信の、ユーザにより指定された許容可能な遅延が500ns、ペリフェラル２からペリフェラル１への通信の、ユーザにより指定された許容可能な遅延も500nsであるとする。

【０１０１】

すると、図１１(b)のタイミング図に示すように、ペリフェラル１での遅れP1と、ペリフェラル２での遅れP2が、ユーザにより指定された許容可能な遅延に収まるように、且つ、因果関係を維持するように、緩和した同期サイクルΔT_peri'が決定される。この処理の擬似コードを以下に示す。これは、現実にはどのプログラミング言語にも対応しないが、Ｃ言語の文法をベースとしているので、この分野の当業者なら、理解することに困難はないと信じる。

【０１０２】

ペリフェラル$p_0,\cdots,p_m$からループが形成されていると仮定
function path_len(i, j): p_iからp_jへ至るパスの長さを得る関数
function delay_allowed(i): p_iに許容される遅延を得る関数
function roundup(u, d): n*u<dでd<=(n+1)*uのとき(n+1)*uを返す関数
ΔTp = ΔT_major // ΔT_majorはシミュレーション環境から得られる最大の同期サイクル
while (delay_allowed(0) < ΔTp) { // 一番最初のペリフェラルp_0に関して，許容範囲内に収まるようにΔTpを小さくする
 ΔTp /= 2;
}
ステップ１：
prev_time = 0; // prev_timeはp_{i-1}の弱連成シミュレーションでの時刻
for (i=1; i<m; i++) { // 2番目以降のすべてのペリフェラルに関して
 time = prev_time + roundup(ΔTp, path_len(i-1, i)); timeはp_iの弱連成シミュレーションでの時刻
 delay = time - path_len(0, i); // 実世界で起こるタイミングからのずれ
 if(delay_allowed(i) < delay) { // 遅延は許容範囲内か?
  ΔTp /= 2; // 許容範囲を超えていた場合はΔTpを半分にして再試行

```
    goto ステップ１
  }
}
ΔTpを答えとする // 全てのペリフェラルが許容範囲に収まったらそのときのΔTpが答え
```

このようにして決定されたΔTpの値が、ΔT_peri'として、シミュレーション・システムが参照可能となるように、メイン・メモリ３０６またはハードディスク・ドライブ３１６に格納される。

【０１０３】

次に、図１２を参照して、同期サイクルの集約について説明する。図１２(a)に示すように、それぞれがループをなす、グループ１内のペリフェラルと、グループ２内のペリフェラルとが、互いに通信する、ということがありえる。

【０１０４】

そこで、本発明の１つの知見によれば、因果律はグループ内で閉じているので、別のグループに属するペリフェラル同士は、シミュレーション環境から得られる最大の同期サイクルである、ΔT_majorにより動作させても、シミュレーションが正しく実行されるということが分かった。すると、ユーザは、そのことを、モジュール間遅延データ４０４に書き込んでおく。

【０１０５】

その様子を、図１２(b)のタイミング図に示す。ここでΔT_minorというのは、前述のΔT_periまたはΔT_peri'である。

【０１０６】

次に、ペリフェラルを、同期サイクルの集約で、すなわち、ΔT_minorとΔT_majorで動作させる場合の処理について説明する。

【０１０７】

図１３は、実行時スケジューラの場合の処理を示すフローチャートである。ステップ１３０２において、実行環境４１６はまず、minorペリフェラルを１つ選ぶ。minorペリフェラルとは、ΔT_minorの同期サイクルで動作するペリフェラルであり、ΔT_minorとは、前述のΔT_periまたはΔT_peri'である。なおここで、ペリフェラルがminorペリフェラルであることは、同期タイミング計算モジュール４０６からの情報により識別できる。

【０１０８】

ステップ１３０４では、そのように選ばれたペリフェラルが実行される。ステップ１３０６では、選ばれたペリフェラルがΔT_minor実行したかどうかが判断され、そうでなければ、ステップ１３０４が継続される。

【０１０９】

ステップ１３０８では、全てのペリフェラルが今回のΔT_minorだけ実行されたかどうかが判断され、そうでなければ、ステップ１３０２に戻って、次のminorペリフェラルが１つ選ばれる。minorペリフェラルとは、ΔT_minorで動作するペリフェラルのことである。

【０１１０】

ステップ１３０８で、すべてのペリフェラルが今回のΔT_minorだけ実行されたと判断されると、ステップ１３１０では、次のΔT_minorへ進められる。すなわち、ΔT_minorだけシミュレーションの時間が進められる。

【０１１１】

ステップ１３１２では、全てのペリフェラルを今回のΔT_major実行させたかどうかが判断され、そうでなければ、ステップ１３０２に戻って、次のminorペリフェラルが１つ選ばれる。ここで、ΔT_majorとは、はシミュレーション環境から得られる最大の同期サイクルであり、例えば、ユーザによって、モジュール間遅延データ４０４に予め設定されている。

【０１１２】

ステップ１３１２で、全てのペリフェラルを今回のΔT_major実行させたと判断される

と、ステップ１３１４に進んで、そこで、majorペリフェラルが１つ選ばれる。majorペリフェラルとは、ΔT_majorで動作するペリフェラルのことである。なおここで、ペリフェラルがmajorペリフェラルであることは、同期タイミング計算モジュール４０６からの情報により識別できる。

【０１１３】

ステップ１３１６では、選ばれたmajorペリフェラルが実行される。ステップ１３１８では、選ばれたペリフェラルがΔT_major実行したかどうかが判断され、そうでなければ、ステップ１３１６が継続される。

【０１１４】

ステップ１３１８では、選ばれたペリフェラルがΔT_major実行したと判断されると、ステップ１３２０で、ΔT_majorだけシミュレーションの時間が進められ、ステップ１３０２に戻る。

【０１１５】

図１４は、静的スケジューラの場合の処理を示すフローチャートである。図１４の処理においては、ステップ１４０２で、minorペリフェラル１が実行され、ステップ１４０４ではminorペリフェラル１について、ΔT_minorだけ実行したかどうか判断され、すなわち、ΔT_minor経過するまで、minorペリフェラル１が実行される。なおここで、ペリフェラルがminorペリフェラルであることは、同期タイミング計算モジュール４０６からの情報により識別できる。

【０１１６】

これと同じ処理が、minorペリフェラル２、minorペリフェラル３・・・に適用されて、ステップ１４０６に示すようにminorペリフェラルNまで実行され、ステップ１４０８でminorペリフェラルNがΔT_minorだけ実行したと判断されると、ステップ１４１０でΔT_minorだけ時間が進められる。

【０１１７】

次にステップ１４１２で、全てのペリフェラルを今回のΔT_major実行させたと判断されると、ステップ１４１４に進んでそこで、majorペリフェラル１が実行され、ステップ１４１６ではmajorペリフェラル１について、ΔT_majorだけ実行したかどうか判断され、すなわち、ΔT_major経過するまで、majorペリフェラル１が実行される。なおここで、ペリフェラルがmajorペリフェラルであることは、同期タイミング計算モジュール４０６からの情報により識別できる。

【０１１８】

これと同じ処理が、majorペリフェラル２、majorペリフェラル３・・・に適用されて、ステップ１４１８に示すようにmajorペリフェラルNまで実行され、ステップ１４２０でminorペリフェラルNがΔT_majorだけ実行したと判断されると、ステップ１４２２でΔT_majorだけ時間が進められる。こうして処理は、ステップ１４０２に戻る。

【０１１９】

次に、図１５を参照して、プラントとの同期の弱連成化について説明する。ユーザは予め、あるペリフェラルとプラントの通信が間が集約された同期サイクルΔT_majorまで遅延を許可するかどうか、予め知っている情報を、モジュール間遅延データ４０４に記録しておく。

【０１２０】

図１５(a)は、ペリフェラル２と、プラント・グループの間が、ΔT_majorで同期可能であることを示す。

【０１２１】

図１５(b)は、ペリフェラルとプラントとの、サイクルΔT_minorでの同期において、ペリフェラルとプラントの間の、ΔT_major内の途中の通信が削除（省略）可能であることを示す。すなわち、途中の通信が省略可能であることにより、ペリフェラルとプラントとがΔT_majorで通信可能であることが分かる。

【０１２２】

図１６は、本発明の処理の全体のシナリオを説明する図である。このシナリオは、同期タイミング計算モジュール４０６と通信しつつ、実行環境４１６によって実行される。その際、同期タイミング計算モジュール４０６は、必要に応じて、モジュール間遅延データ４０４を参照する。

【０１２３】

図１６において、ステップ１６０２では、実行環境４１６は、図７の処理により強連成モデルで最大同期サイクルを発見し、図９または図１０の処理によりシミュレーションを試みる。

【０１２４】

ステップ１６０４では、十分な実行速度が得られたかどうかが判断され、もしそうなら、シミュレーションは終了する。

【０１２５】

ステップ１６０４で、十分な実行速度が得られないと判断されると、ステップ１６０６に進んで、ペリフェラル間の弱連成化が試みられる。このために、前述の擬似コードで示した処理により、弱連成化のためのΔT_peri'が計算され、このΔT_peri'で以って図９または図１０の処理によりシミュレーションを試みる。

【０１２６】

ステップ１６０８では、十分な実行速度が得られたかどうかが判断され、もしそうなら、シミュレーションは終了する。

【０１２７】

ステップ１６０８で、十分な実行速度が得られないと判断されると、ステップ１６１０に進んで、ペリフェラル間の弱連成化に加え、図１３または図１４の処理により同期サイクルの集約が試みられる。

【０１２８】

ステップ１６１２では、十分な実行速度が得られたかどうかが判断され、もしそうなら、シミュレーションは終了する。

【０１２９】

ステップ１６１２で、十分な実行速度が得られないと判断されると、ステップ１６１４に進んで、ペリフェラル間の弱連成化と、同期サイクルの集約に加え、図１５に示すプラントとの同期の弱連成化を試み、シミュレーションは終了する。

【０１３０】

以上、ＥＣＵエミュレータに関するペリフェラル間の通信を例にして本発明を説明してきたが、本発明は、互いに同期通信する任意の機能ブロックについて適用可能である。

【０１３１】

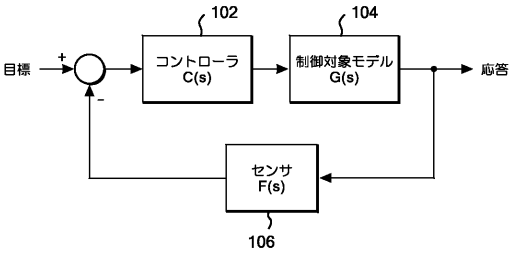
また、自動車用の複数のシミュレーション・システムに関連して、本発明の特定の実施例を説明してきたが、本発明はこのような特定の実施例に限定されず、飛行機用のシミュレーション・システムなど、一般的な電子機械制御系システムのシミュレーション・システムに適用可能であることを、この分野の当業者であるなら、理解するであろう。

【符号の説明】

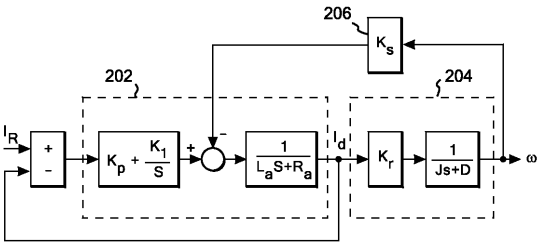
【０１３２】

４０４　モジュール間遅延データ
４０６　同期タイミング計算モジュール
４０８　エミュレータ
４１４　プラント・シミュレータ
４１２　ペリフェラル・エミュレータ
４１６　実行環境
５０６　プラント・シミュレータ
６０２　プラント・シミュレータ
６０４　ペリフェラル・エミュレータ

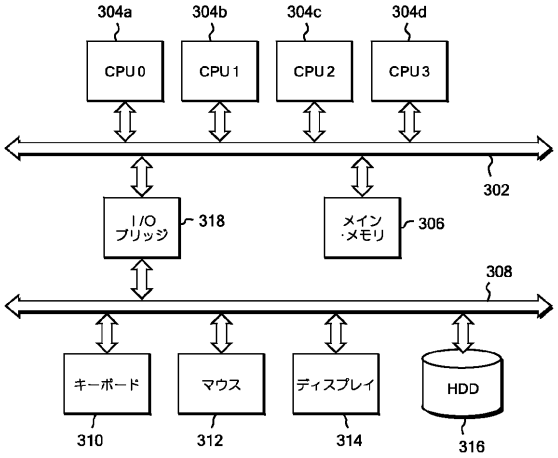
【図１】



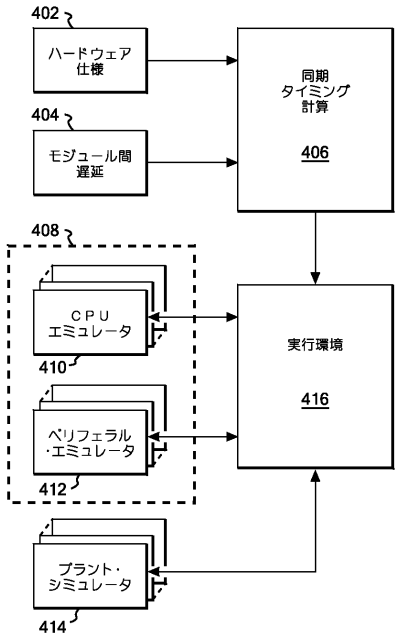
【図２】



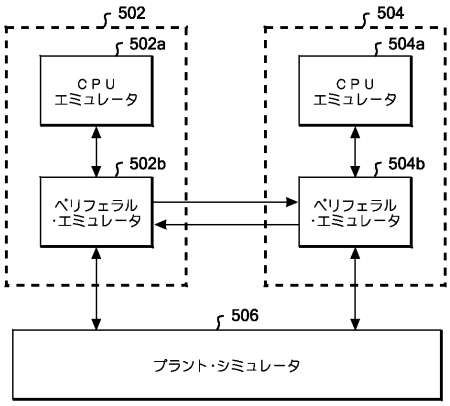
【図３】



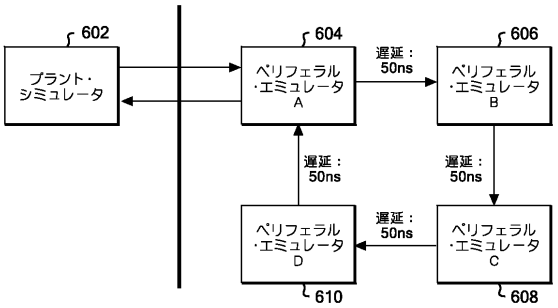
【図４】



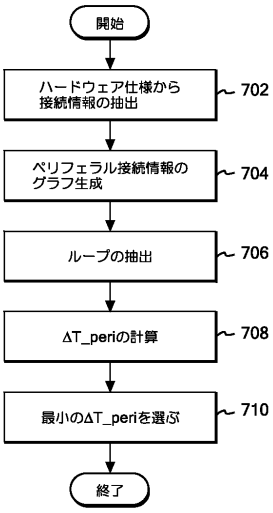
【図５】



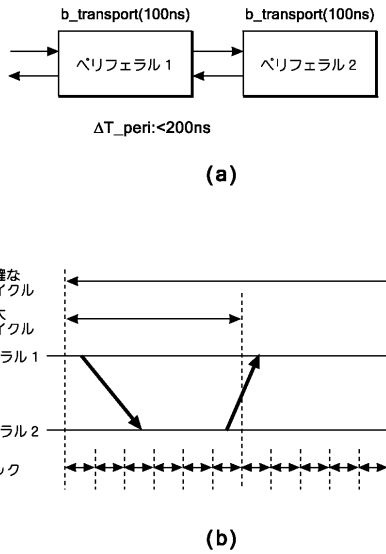
【図６】



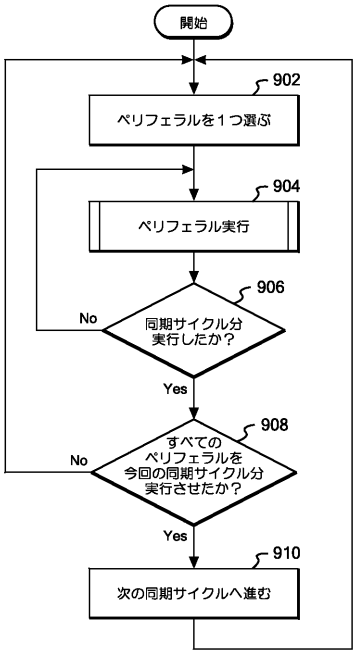
【図７】


開始
ハードウェア仕様から
接続情報の抽出
702
ペリフェラル接続情報の
グラフ生成
704
ループの抽出
706
ΔT_periの計算
708
最小のΔT_periを選ぶ
710
終了

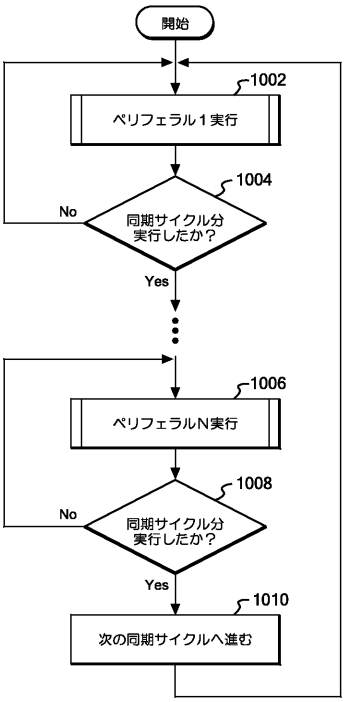
【図８】


b_transport(100ns)
b_transport(100ns)
ペリフェラル１
ペリフェラル２
ΔT_peri:<200ns
(a)
不正確な
同期サイクル
最大
同期サイクル
ペリフェラル１
ペリフェラル２
クロック
(b)

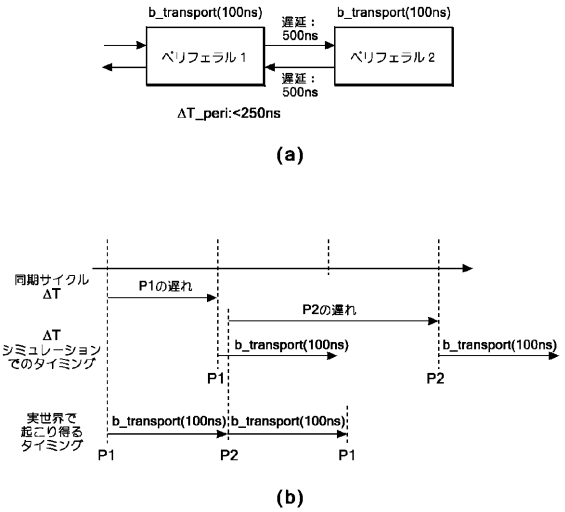
【図９】


開始
902
ペリフェラルを１つ選ぶ
904
ペリフェラル実行
906
No
同期サイクル分
実行したか？
Yes
908
No
すべての
ペリフェラルを
今回の同期サイクル分
実行させたか？
Yes
910
次の同期サイクルへ進む

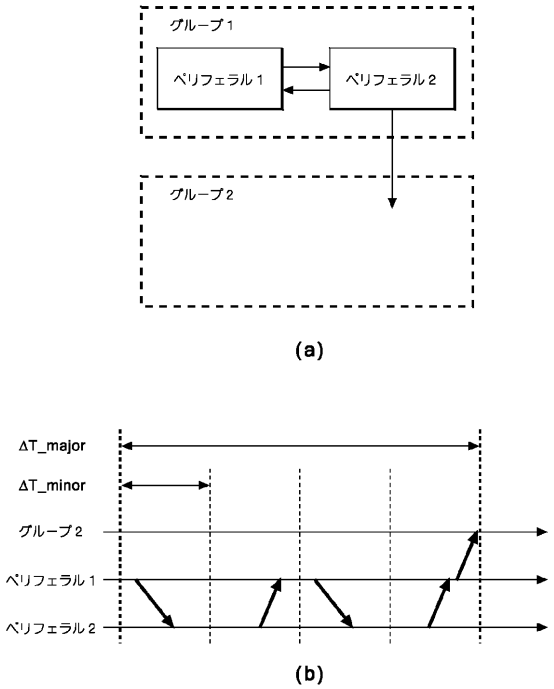
【図１０】


開始
1002
ペリフェラル１実行
1004
No
同期サイクル分
実行したか？
Yes
1006
ペリフェラルN実行
1008
No
同期サイクル分
実行したか？
Yes
1010
次の同期サイクルへ進む

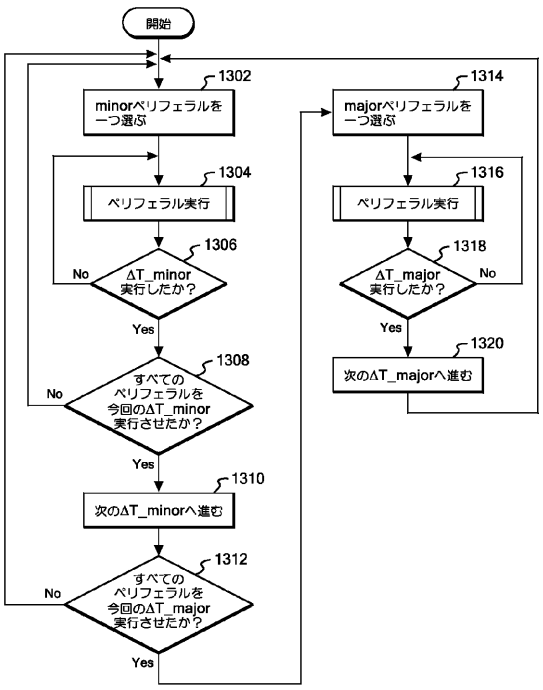
【図１１】



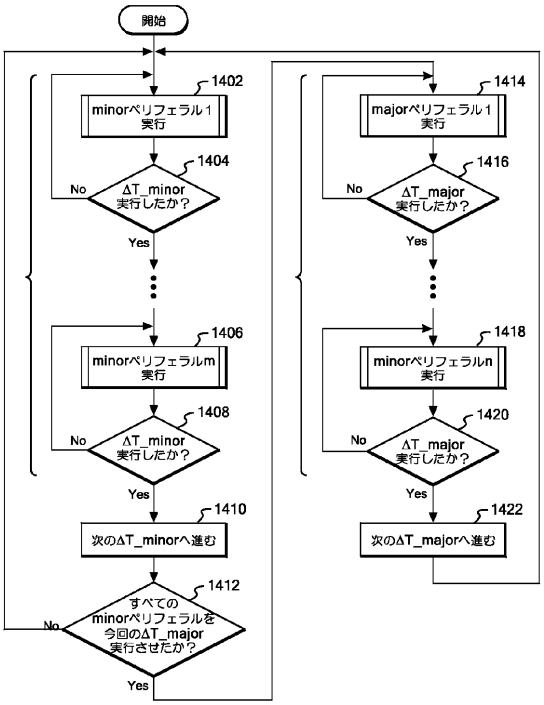
【図１２】



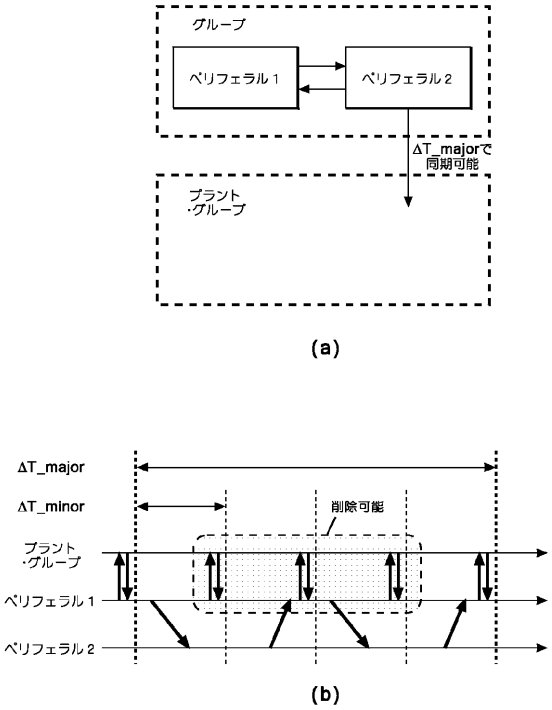
【図１３】



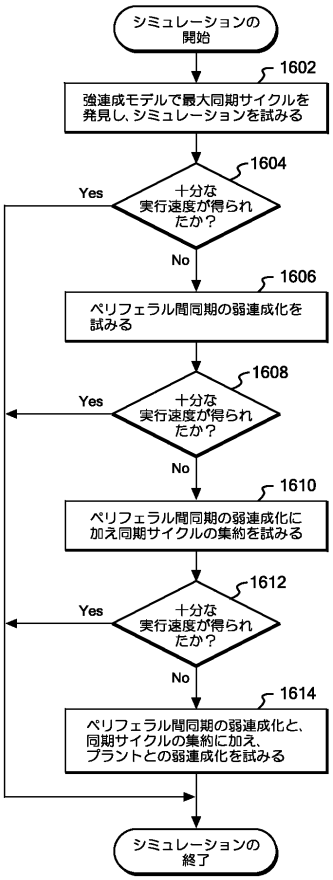
【図１４】



【図１５】


グループ
ペリフェラル１
ペリフェラル２
ΔT_majorで
同期可能
プラント
・グループ
(a)
ΔT_major
ΔT_minor
削除可能
プラント
・グループ
ペリフェラル１
ペリフェラル２
(b)

【図１６】


シミュレーションの
開始
1602
強連成モデルで最大同期サイクルを
発見し、シミュレーションを試みる
1604
十分な
実行速度が得られ
たか？
Yes
No
1606
ペリフェラル間同期の弱連成化を
試みる
1608
十分な
実行速度が得られ
たか？
Yes
No
1610
ペリフェラル間同期の弱連成化に
加え同期サイクルの集約を試みる
1612
十分な
実行速度が得られ
たか？
Yes
No
1614
ペリフェラル間同期の弱連成化と、
同期サイクルの集約に加え、
プラントとの弱連成化を試みる
シミュレーションの
終了

フロントページの続き

(72)発明者　近藤　豪
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内
(72)発明者　片岡　正樹
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内
(72)発明者　小松　秀昭
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内
(72)発明者　大澤　史朋
神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　大和事業所内

Ｆターム(参考) 5B042 GB08　HH06　HH07