(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3951925号
(P3951925)
(45)発行日　平成19年8月1日(2007.8.1)　(24)登録日　平成19年5月11日(2007.5.11)

(51)Int.Cl.		F I		
G06F 17/50	(2006.01)	G06F	17/50	664A
G06F 11/28	(2006.01)	G06F	17/50	664J
		G06F	11/28	L

請求項の数 4　(全 16 頁)

(21)出願番号　特願2003-24706(P2003-24706)
(22)出願日　平成15年1月31日(2003.1.31)
(65)公開番号　特開2004-234528(P2004-234528A)
(43)公開日　平成16年8月19日(2004.8.19)
審査請求日　平成15年1月31日(2003.1.31)

(73)特許権者　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号
(74)代理人　100099759
弁理士　青木　篤
(74)代理人　100092624
弁理士　鶴田　準一
(74)代理人　100102819
弁理士　島田　哲郎
(74)代理人　100141254
弁理士　榎原　正巳
(74)代理人　100108383
弁理士　下道　晶久

最終頁に続く

(54)【発明の名称】ハードウェア／ソフトウェア協調検証方法

(57)【特許請求の範囲】
【請求項１】
ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、
(a) 該ホストＣＰＵが、入力された検証モデルとしてのＣベース言語記述のTimedソフトウェア部品をコンパイルし、入力された検証モデルとしてのＣベース言語記述のハードウェア部品をコンパイルし、並びに該コンパイルされたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、
(b) 該ホストＣＰＵが、入力されたテストベンチをコンパイルするステップと、
(c) 該ホストＣＰＵが、ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、
(d) 該ホストＣＰＵが、ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、
(e) 該ホストＣＰＵが、ステップ(d)によるシミュレーションの結果を出力するステップと、
を具備するとともに、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品を予め作成すべく、更に、
該ホストＣＰＵが、入力されたＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、
該ホストＣＰＵが、該制御点を挿入されたUntimedソフトウェア部品をコンパイルして

ターゲットＣＰＵ用バイナリ・コードを生成するステップと、
該ホストＣＰＵが、該生成されたターゲットＣＰＵ用バイナリ・コードについて制御点間の実行時間を算出するステップと、
該ホストＣＰＵが、該算出された実行時間に応じて、該制御点を挿入されたUntimedソフトウェア部品の各制御点へ実行時間挿入文を挿入し、Ｃベース言語記述のTimedソフトウェア部品として出力するステップと、
を具備するハードウェア／ソフトウェア協調検証方法。
【請求項２】
ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、
(a) 該ホストＣＰＵが、入力された検証モデルとしてのＣベース言語記述のハードウェア部品をコンパイルし、並びに入力された検証モデルとしてのホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、
(b) 該ホストＣＰＵが、入力されたテストベンチをコンパイルするステップと、
(c) 該ホストＣＰＵが、ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、
(d) 該ホストＣＰＵが、ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、
(e) 該ホストＣＰＵが、ステップ(d)によるシミュレーションの結果を出力するステップと、
を具備するとともに、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品を予め作成すべく、更に、
該ホストＣＰＵが、入力されたターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品を、ホストＣＰＵ用バイナリ・コードのソフトウェア部品へと変換するステップと、
該ホストＣＰＵが、該ホストＣＰＵ用バイナリ・コードのソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、
該ホストＣＰＵが、該制御点を挿入されたソフトウェア部品について制御点間の実行時間を算出するステップと、
該ホストＣＰＵが、該算出された実行時間に応じて、該制御点を挿入されたソフトウェア部品の各制御点へ実行時間挿入文同等機能のバイナリ・コードを挿入し、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品として出力するステップと、
を具備するハードウェア／ソフトウェア協調検証方法。
【請求項３】
ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、
(a) 該ホストＣＰＵが、入力された検証モデルとしてのＣベース言語記述のTimedソフトウェア部品をコンパイルし、入力された検証モデルとしてのＣベース言語記述のハードウェア部品をコンパイルし、並びに、該コンパイルされたTimedソフトウェア部品と、入力された検証モデルとしてのホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品と、該コンパイルされたハードウェア部品と、をリンクするステップと、
(b) 該ホストＣＰＵが、入力されたテストベンチをコンパイルするステップと、
(c) 該ホストＣＰＵが、ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、
(d) 該ホストＣＰＵが、ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、
(e) 該ホストＣＰＵが、ステップ(d)によるシミュレーションの結果を出力するステップと、

を具備するとともに、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品を予め作成すべく、更に、

該ホストＣＰＵが、入力されたＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、

該ホストＣＰＵが、該制御点を挿入されたUntimedソフトウェア部品をコンパイルしてターゲットＣＰＵ用バイナリ・コードを生成するステップと、

該ホストＣＰＵが、該生成されたターゲットＣＰＵ用バイナリ・コードについて制御点間の実行時間を算出するステップと、

該ホストＣＰＵが、該算出された実行時間に応じて、該制御点を挿入されたUntimedソフトウェア部品の各制御点へ実行時間挿入文を挿入し、Ｃベース言語記述のTimedソフトウェア部品として出力するステップと、

を具備するハードウェア／ソフトウェア協調検証方法。

【請求項４】

ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、

(a) 該ホストＣＰＵが、入力された検証モデルとしてのＣベース言語記述のTimedソフトウェア部品をコンパイルし、入力された検証モデルとしてのＣベース言語記述のハードウェア部品をコンパイルし、並びに該コンパイルされたTimedソフトウェア部品と、入力された検証モデルとしてのホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品と、該コンパイルされたハードウェア部品と、をリンクするステップと、

(b) 該ホストＣＰＵが、入力されたテストベンチをコンパイルするステップと、

(c) 該ホストＣＰＵが、ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、

(d) 該ホストＣＰＵが、ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、

(e) 該ホストＣＰＵが、ステップ(d)によるシミュレーションの結果を出力するステップと、

を具備するとともに、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品を予め作成すべく、更に、

該ホストＣＰＵが、入力されたターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品を、ホストＣＰＵ用バイナリ・コードのソフトウェア部品へと変換するステップと、

該ホストＣＰＵが、該ホストＣＰＵ用バイナリ・コードのソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、

該ホストＣＰＵが、該制御点を挿入されたソフトウェア部品について制御点間の実行時間を算出するステップと、

該ホストＣＰＵが、該算出された実行時間に応じて、該制御点を挿入されたソフトウェア部品の各制御点へ実行時間挿入文同等機能のバイナリ・コードを挿入し、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品として出力するステップと、

を具備するハードウェア／ソフトウェア協調検証方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、半導体装置に搭載されるハードウェア及びソフトウェアを協調検証（Co-verification）する方法に関する。

【０００２】

【従来の技術】

近年、ＳｏＣを搭載した機器が広く普及してきている。ＳｏＣとは、System on a Chipの略で、コンピュータの主要機能を一つのチップ（半導体装置）に詰め込む技術、あるい

は、当該技術によりコンピュータの主要機能を搭載したチップをいう。

【０００３】

図１は、かかるＳｏＣの上流設計フローを示す図である。同図に示されるように、システム・レベルの設計が完了した後、アーキテクチャ・レベルの設計に移行する。アーキテクチャ・レベルの設計では、ＣＰＵ（Central Processing Unit）、ＯＳ（Operating System）、バス（Bus）等の基本部品の選択、ハードウェア及びソフトウェアへの機能分割、並びにハードウェア設計及びソフトウェア設計が行われる。そして、アーキテクチャ・レベルの設計により得られた基本部品、ハードウェア部品及びソフトウェア部品に対し、それらの検証モデルに基づくハードウェア／ソフトウェア協調検証が実行される。

【０００４】

一般に、かかるハードウェア／ソフトウェア協調検証においては、ＣＰＵの検証モデルとして、下記特許文献１～３に開示されるように、命令レベルでシミュレーションを行うＩＳＳ（Instruction Set Simulator，命令セット・シミュレータ）が使用されている。このＩＳＳは、Ｃベース言語で記述され、市販のシミュレータとしてＳｙｓｔｅｍ－Ｇ（ガイオ・テクノロジー社製）等が知られている。なお、本明細書において、「Ｃベース言語」とは、ＡＮＳＩ－Ｃ／Ｃ＋＋の各種拡張言語、ＳｐｅｃＣ、ＳｙｓｔｅｍＣのいずれかの言語を意味する。

【０００５】

又、ＣＰＵ専用メモリ、バス等のハードウェア部品についても、Ｃベース言語で記述された検証モデルが使用される。更に、その他のハードウェア部品についても、Ｃベース言語を用いたビヘイビア（Behavior）記述又はＲＴＬ（Register Transfer Level）記述による検証モデルが使用される。なお、ビヘイビア記述とは、回路のふるまいを記述したものであり、一方、ＲＴＬ記述とは、レジスタの値が遷移していく様子を記述したものである。

【０００６】

一方、ＯＳ、ミドルウェア、割込みハンドラ、デバイス・ドライバ、タスク等のソフトウェア部品の検証モデルは、実論理即ちターゲットＣＰＵのバイナリ・コード（命令コード）自身で構成される。なお、本明細書において、「ターゲットＣＰＵ」とは、検証対象のＳｏＣ等の半導体装置に搭載されているＣＰＵ（例えば、ＡＲＭプロセッサ）を意味する。

【０００７】

以上のような検証モデル（ハードウェア部品、ソフトウェア部品）にテストベンチ及びＣベース・シミュレータを加えることで、図２に示される如き従来の協調検証システム（ソフトウェア構成）が構成される。テストベンチは、テスト・データの入力、テスト・データの出力と期待値との比較、等を実行し、Ｃベース・シミュレータは、シミュレーション全体の動作を制御する。又、ＩＳＳは、ハードウェア部品からのマスク可能割込みＩＮＴ（maskable INTerrupt）やテストベンチからのマスク不能割込みＮＭＩ（NonMaskable Interrupt）を受付けつつ動作する。

【０００８】

このＩＳＳの機能は、命令レベル・シミュレーション、メモリ・アクセス（ＬＯＡＤ／ＳＴＯＲＥ命令）、Ｉ／Ｏアクセス、及び割込み処理である。命令レベル・シミュレーション機能では、ターゲットＣＰＵのバイナリ・コード・レベルでのシミュレーションを実行する。又、メモリ・アクセス機能では、バスへのＲＥＡＤ／ＷＲＩＴＥアクセスを実行する。なお、バスがメモリ・アクセスを制御する。Ｉ／Ｏアクセス機能では、ＬＯＡＤ／ＳＴＯＲＥ命令（一回あたりのデータ転送量が小さい）を用いた、バスへのＲＥＡＤ／ＷＲＩＴＥアクセスを実行する。なお、バスがＩ／Ｏ（ハードウェア部品）アクセスを制御する。又、割込み処理機能では、割込み（ＩＮＴｉ(i=1, …,n)、ＮＭＩ）の受付け、割込みハンドラの起動制御、実行中の処理の中断、中断した処理の再開、等を実行する。

【０００９】

ＩＳＳを使用した協調検証に関する先行技術文献としては、下記特許文献１～３の他に、

下記の非特許文献１及び２が存在する。なお、下記の非特許文献３は、後述する説明において引用される“Basic Block”に関するものであり、非特許文献４～６は、後述する説明において引用される“Fixed I/O Behaviorモデル”に関するものであり、非特許文献７～９は、Ｃ言語ベースの設計及び検証の技術動向に関するものである。

【００１０】

【特許文献１】

特開2000－259445号公報

【特許文献２】

特開2001－256072号公報

【特許文献３】

特開2002－175344号公報

【非特許文献１】

若林一敏：Ｃ言語によるＬＳＩ設計－動作合成とＨＷ／ＳＷ協調検証の実際、NE Embedded Symposium 2002.

【非特許文献２】

黒川、池上、大坪、浅尾、桐ヶ谷、三栖、高橋、川津、新田、笠、若林、友部、高橋、向山、竹中：Ｃ言語ベースの動作合成を利用したシステムＬＳＩ設計手法の効果分析と考察、電子情報通信学会第１５回軽井沢ワークショップ、pp. 131-142、Apr. 2002.

【非特許文献３】

T. Sherwood, E. Perelman and B. Calder, “Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications”, in International Conference on Parallel Architectures and Compilation Techniques, Sept. 2001.

【非特許文献４】

D. W. Knapp, T. Ly, D. MacMillen and R. Miller, “Behavioral Synthesis Methodology for HDL-Based Specification and Validation”, Proc. Design Automation Conf., June 1995.

【非特許文献５】

T. Ly, D. W. Knapp, R. Miller and D. MacMillen, “Scheduling using Behavioral Templates”, Proc. Design Automation Conf., June 1995.

【非特許文献６】

D. W. Knapp, “Behavioral Synthesis: Digital System Design using the Synopsys Behavioral Compiler”, Prentice Hall PTR.

【非特許文献７】

L. Gauthier, S. Yoo and A. A. Jerraya, “Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software”, Proc. Design Automation and Test in Europe, Mar. 2001.

【非特許文献８】

D. Lyonnard, S. Yoo, A. Baghdadi and A. A. Jerraya, “Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip”, Proc. Design Automation Conf., June 2001.

【非特許文献９】

S. Yoo, G. Nicolescu, L. Gauthier and A. A. Jerraya, “Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC”, Proc. Design Automation and Test in Europe, Mar. 2002.

【００１１】

【発明が解決しようとする課題】

上述のような、従来技術に係る、ＩＳＳを使用したハードウェア／ソフトウェア協調検証方法では、シミュレーションが命令レベルで実行され、即ち一命令ごとにその命令内容を解釈しながら実行されるとともに、メモリ・アクセスが必要となるため、シミュレーショ

ン時間即ち検証時間が大きくなるという問題がある。

【００１２】

本発明は、上述した問題点に鑑みてなされたものであり、その目的は、タイミングについてのシミュレーション精度を低下させることなく、一命令ごとの解釈と実行を不要とするＣベースのネイティブ・コード・シミュレーションを実現することにより、シミュレーションの高速化を実現したハードウェア／ソフトウェア協調検証方法を提供することにある。

【００１３】

なお、以下の説明において、「Untimedソフトウェア部品」とは、その全ての処理をＡＮＳＩ－Ｃで記述したソフトウェア部品、又は、その全ての処理を実行するバイナリ・コードのソフトウェア部品を意味する。一方、「Timedソフトウェア部品」とは、その全ての処理を複数の単位処理に分割し、各単位処理を記述したＡＮＳＩ－Ｃ記述の後に、当該単位処理の実行時間を記述したＣベース言語記述（実行時間挿入文）を追加したソフトウェア部品、又は、その全ての処理を複数の単位処理に分割し、各単位処理を実行するバイナリ・コードの後に、当該単位処理の実行時間を記述したＣベース言語記述（実行時間挿入文）同等機能のバイナリ・コードを追加したソフトウェア部品を意味する。又、「ホストＣＰＵ」とは、協調検証を実行するパーソナル・コンピュータ（ＰＣ）又はワーク・ステーション（ＷＳ）に搭載されているＣＰＵ（例えば、Pentium(登録商標)プロセッサ）を意味する。

【００１４】

【課題を解決するための手段】

上記目的を達成するために、本発明の第一の面(Aspect)によれば、ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしてのＣベース言語記述のTimedソフトウェア部品を入力してコンパイルし、検証モデルとしてのＣベース言語記述のハードウェア部品を入力してコンパイルし、並びに該コンパイルされたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ(d)によるシミュレーションの結果を出力するステップと、を具備するハードウェア／ソフトウェア協調検証方法が提供される。

【００１５】

又、本発明の第二の面によれば、ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしてのホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品を入力し、検証モデルとしてのＣベース言語記述のハードウェア部品を入力してコンパイルし、並びに該入力されたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ(d)によるシミュレーションの結果を出力するステップと、を具備するハードウェア／ソフトウェア協調検証方法が提供される。

【００１６】

又、本発明の第三の面によれば、ホストＣＰＵを使用して、一つのターゲットＣＰＵ及び一つのＯＳが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしてのＣベース言語記述のTimedソフトウェア部品を入力してコンパイルし、検証モデルとしてのホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品を入力し、検証モデルとしてのＣベース言語記述のハードウェア部品

を入力してコンパイルし、並びに該コンパイルされ又は該入力されたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ(d)によるシミュレーションの結果を出力するステップと、を具備するハードウェア／ソフトウェア協調検証方法が提供される。

【００１７】

又、本発明の第四の面によれば、前記本発明の第一又は第三の面による方法において、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品を予め作成すべく、更に、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品を入力して、Basic Block を認識し、制御点を挿入するステップと、該制御点を挿入されたUntimedソフトウェア部品をコンパイルしてターゲットＣＰＵ用バイナリ・コードを生成するステップと、該生成されたターゲットＣＰＵ用バイナリ・コードについて制御点間の実行時間を算出するステップと、該算出された実行時間に応じて、該制御点を挿入されたUntimedソフトウェア部品の各制御点へ実行時間挿入文を挿入し、Ｃベース言語記述のTimedソフトウェア部品として出力するステップと、が具備される。

【００１８】

又、本発明の第五の面によれば、前記本発明の第二又は第三の面による方法において、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品を予め作成すべく、更に、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品を入力して、ホストＣＰＵ用バイナリ・コードのソフトウェア部品へと変換するステップと、該ホストＣＰＵ用バイナリ・コードのソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、該制御点を挿入されたソフトウェア部品について制御点間の実行時間を算出するステップと、該算出された実行時間に応じて、該制御点を挿入されたソフトウェア部品の各制御点へ実行時間挿入文同等機能のバイナリ・コードを挿入し、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品として出力するステップと、が具備される。

【００１９】

【発明の実施の形態】

以下、添付図面を参照して本発明の実施形態について説明する。

【００２０】

図３は、本発明に係るハードウェア／ソフトウェア協調検証方法を実施するためのハードウェア環境を説明するための図である。同図に例示されるように、本発明に係る協調検証は、中央処理装置（ＣＰＵ）１２及び主記憶装置（ＭＳ）１４を有するコンピュータ本体１０、ディスプレイ２０、キーボード２２、マウス２４、ハードディスク装置等からなる外部記憶装置３０を備える通常のパーソナル・コンピュータ（ＰＣ）又はワーク・ステーション（ＷＳ）上で実行可能である。

【００２１】

ＣＰＵ１２は、協調検証を実行するホストＣＰＵとして動作するものであり、例えば、Pentium(登録商標)プロセッサである。以下に説明される協調検証のためのプログラムは、ＣＰＵ１２によって実行される。又、各種のデータ、ファイル等は、外部記憶装置３０から主記憶装置（ＭＳ）１４にロードされて処理される。

【００２２】

次に、ハードウェア／ソフトウェア協調検証の対象となるＳｏＣ（半導体装置）に搭載される各種部品のうち、ソフトウェア部品の検証モデルの作成方法について、図４を用いて説明する。同図に示される例では、ソフトウェア部品のうち、ＯＳ／ミドルウェア及びタスクは、ＡＮＳＩ－Ｃのみで論理設計されているが、割込みハンドラ及びデバイス・ドライバについては、それらの設計論理にＡＮＳＩ－Ｃによる記述に加えてアセンブリ言語による記述が含まれる。そのため、まず、そのアセンブリ言語による記述部分がＡＮＳＩ－

Ｃによる記述へと変換される。この変換は、人手によりなされる。
【００２３】
更に、デバイス・ドライバについては、バスへの直接ＲＥＡＤ／ＷＲＩＴＥアクセスを実行するように、人手により変換される。即ち、従来のＩＳＳでは、一回あたりのデータ転送量が小さいＬＯＡＤ／ＳＴＯＲＥ命令を用いてＩ／Ｏアクセスがなされていたため低速であったが、本実施形態においては、Ｉ／Ｏアクセスの高速化が図られる。
【００２４】
最後に、全てＡＮＳＩ－Ｃにより記述されたＯＳ／ミドルウェア、割込みハンドラ、デバイス・ドライバ及びタスクについて、Untimedソフトウェア部品（ＡＮＳＩ－Ｃ記述）からTimedソフトウェア部品（Ｃベース言語記述）への自動的な変換が、次に説明されるプログラムにより、実施される。
【００２５】
図５は、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。又、図６は、その変換処理の内容について説明するための図である。
【００２６】
まず、図５のステップ１１０では、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品５０を入力して、Basic Block を認識し、制御点を挿入することにより、制御点を挿入されたUntimedソフトウェア部品（ＡＮＳＩ－Ｃ記述）５２を出力する。このBasic Blockは、プログラムがストレートに走行する部分を指すものであり、その詳細は、上記非特許文献３に説明されている。そして、その認識された Basic Block の前後に制御点が挿入される。
【００２７】
即ち、図６（Ａ）に示されるUntimedソフトウェア部品（ＡＮＳＩ－Ｃ記述）に対しては、ノードａ及びノードｂが一つの Basic Block と認識され、ノードｃ、ノードｄ及びノードｅが一つの Basic Block と認識され、ノードｃ及びノードｆが一つの Basic Block と認識され、ノードｇが一つの Basic Block と認識される。そのため、図６（Ｂ）に示されるように、ノードｂとノードｃとの間、ノードｅとノードｇとの間、ノードｆとノードｇとの間、及びノードｇの後に、それぞれ制御点が挿入される。
【００２８】
次いで、ステップ１２０では、かかる制御点を挿入されたUntimedソフトウェア部品（ＡＮＳＩ－Ｃ記述）５２をコンパイルすることにより、ターゲットＣＰＵ用のバイナリ・コード５４を生成する。
【００２９】
次いで、ステップ１３０では、上述のコンパイルの結果、生成されたターゲットＣＰＵ用バイナリ・コード（命令コード）５４に基づいて、制御点間の実行時間を算出する。その算出は、
ｋΣ［各命令のサイクル数］
なる演算式に基づいて行われる。ここで、係数ｋは、キャッシュ・メモリのミスヒットに起因するオーバヘッド係数であり、本実施形態においてはキャッシュ・メモリ・モデルを設けないことから、統計的処理を可能とするために導入されたものである。
【００３０】
最後のステップ１４０では、制御点を挿入されたUntimedソフトウェア部品５２の各制御点へ、ステップ１３０で算出された実行時間に応じた実行時間挿入文を挿入し、Ｃベース言語記述のTimedソフトウェア部品５６として出力する。
【００３１】
例えば、ステップ１３０において、図６（Ｂ）に示されるノードａ及びノードｂの実行時間がｔ１と算出され、ノードｃ、ノードｄ及びノードｅ の実行時間がｔ２と算出され、ノードｃ及びノードｆ の実行時間がｔ３と算出され、ノードｇ の実行時間がｔ４と算出されたとする。その場合には、図６（Ｃ）に示されるように、ノードｂとノードｃとの間の制御点には、実行時間挿入文として waitfor(t1) が挿入される。同様に、ノードｅと

ノードｇとの間の制御点には実行時間挿入文として waitfor(t2) が挿入され、ノードｆとノードｇとの間の制御点には実行時間挿入文として waitfor(t3) が挿入され、ノードｇの後の制御点には実行時間挿入文として waitfor(t4) が挿入されることとなる。

【００３２】

図７は、このようにして作成されたＣベース言語記述のTimedソフトウェア部品５６に基づくＣベース・シミュレーションについて説明するための図である。Ｃベース言語記述のTimedソフトウェア部品５６をコンパイルしてホストＣＰＵ用バイナリ・コードを生成することにより、ホストＣＰＵ１２（図３）におけるネイティブ・コード・シミュレーション（Native Code Simulation）が可能となる。

【００３３】

即ち、Ｃベース言語記述のTimedソフトウェア部品５６をコンパイルして生成されたホストＣＰＵ用バイナリ・コードをホストＣＰＵで実行させることにより、一命令ごとにその命令内容を解釈してシミュレーションを実行するＩＳＳの場合に比較して、１００～１００倍程度の高速化を図ることができる。

【００３４】

一方、シミュレーションの過程において、 waitfor 文が出現したところで、シミュレータがその内容を解釈することにより、命令実行時間の管理が可能となるため、タイミングについてのシミュレーション精度も維持することができる。即ち、ノードｂに相当する命令コードの実行後には、 waitfor(t1) の命令コードを解釈することで累積処理時間Ｔ１＝ｔ１が求められる。同様に、ノードｅに相当する命令コードの実行後には累積処理時間Ｔ２＝ｔ１＋ｔ２が求められる一方、ノードｆに相当する命令コードの実行後には累積処理時間Ｔ２＝ｔ１＋ｔ３が求められる。

【００３５】

そして、ノードｇに相当する命令コードの実行後には、プログラムがノードｃ、ノードｄ及びノードｅ のルートを走行した場合には累積処理時間Ｔ３＝ｔ１＋ｔ２＋ｔ４が求められる一方、プログラムがノードｃ及びノードｆのルートを走行した場合には累積処理時間Ｔ３＝ｔ１＋ｔ３＋ｔ４が求められることとなる。

【００３６】

ところで、ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品を作成する図５のソフトウェア部品検証モデル作成方法では、Untimedソフトウェア部品のソース・コード（ＡＮＳＩ－Ｃ）が必要となるが、大半のＯＳやミドルウェアでは、その入手が困難である。そこで、本実施形態においては、ターゲットＣＰＵ用バイナリ・コードによるUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードによるTimedソフトウェア部品への変換が行われる。

【００３７】

図８は、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。まず、ステップ２１０では、ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品６０を入力して、ホストＣＰＵ用バイナリ・コードのソフトウェア部品６２へと変換する。この変換は、Ｄｙｎａｍｉｃ（Transitive Technologies社製ツール）等を利用して実現することができる。

【００３８】

次いで、ステップ２２０では、ホストＣＰＵ用バイナリ・コードのソフトウェア部品６２について、前述したステップ１１０（図５）と同様に、Basic Block を認識することにより、制御点を挿入されたソフトウェア部品６４を出力する。

【００３９】

次いで、ステップ２３０では、前述したステップ１３０（図５）と同様に、制御点を挿入されたソフトウェア部品６４について制御点間の実行時間を算出する。その算出は、

ｋΣ［各命令のサイクル数］

なる演算式に基づいて行われる。ここで、係数ｋは、キャッシュ・メモリのミスヒットに

起因するオーバヘッド係数である。

【００４０】

最後のステップ２４０では、制御点を挿入されたソフトウェア部品６４の各制御点へ、算出された実行時間に応じた実行時間挿入文同等機能のバイナリ・コードを挿入し、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品６６として出力する。

【００４１】

次に、ハードウェア／ソフトウェア協調検証の対象となるＳｏＣ（半導体装置）に搭載される各種部品のうち、ハードウェア部品の検証モデルの作成方法について説明する。前述のように、従来のハードウェア／ソフトウェア協調検証においては、ＣＰＵ及びＣＰＵ専用メモリの検証モデルが準備されたが、本実施形態においては、それらの検証モデルは作成されず、その代わりに、ＩＳＳ（Instruction Set Simulator）における割込み処理部を独立化させたＩＲＳ（Interrupt Routine Scheduler）が新たに導入される。このＩＲＳは、Ｃベース言語で記述される。又、バスの検証モデルは、従来と同様に、Ｃベース言語で新規に作成される。

【００４２】

又、Ｃベース言語を用いて論理設計されたビヘイビア（Behavior）記述のハードウェア部品は、動作合成ツールの拡張機能を利用して、Ｃベース言語記述からFixed I/O Behaviorモデルへと自動的に変換されることにより、その検証モデル（Ｃベース言語）が生成される。このFixed I/O Behaviorモデルは、Basic Block を利用した Timed ハードウェア・モデルと同等のものであり、その詳細は、上記の非特許文献４～６に説明されている。

【００４３】

又、Verilog／ＶＨＤＬを用いて論理設計されたＲＴＬ（Register Transfer Level）記述のハードウェア部品は、ＨＤＬ Import（CoWare社製ツール）等を利用して、ＲＴＬ記述からＲＴＬ－Ｃベース言語モデルへと自動的に変換されることにより、その検証モデル（Ｃベース言語）が生成される。このＲＴＬ－Ｃベース言語モデルは、ＦＳＭ（Finite State Machine,有限状態機械）の１ステートが１クロックの動作を表現するものである。

【００４４】

以上のようにして作成されたソフトウェア部品及びハードウェア部品の検証モデルにテストベンチ及びＣベース・シミュレータを加えることで、本実施形態に係る協調検証システム（ソフトウェア構成）が図９に示されるように構成される。なお、前述のように、テストベンチは、テスト・データの入力、テスト・データの出力と期待値との比較、等を実行し、Ｃベース・シミュレータは、シミュレーション全体の動作を制御する。ＩＲＳは、ＣＰＵ割込み回路相当の機能を実現するものである。又ＩＮＴは、マスク可能割込み（maskable INTerrupt）、ＮＭＩは、マスク不能割込み（NonMaskable Interrupt）である。

【００４５】

図１０は、図３に示されるハードウェア環境及び図９に示されるソフトウェア構成の下で実行される、ＳｏＣに対するハードウェア／ソフトウェア協調検証の手順を説明するためのフローチャートである。まず、ステップ３１０では、検証モデルの入力、必要なコンパイル、及びリンクを実行する。

【００４６】

即ち、検証モデルとして、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品、Ｃベース言語記述のTimedソフトウェア部品、Ｃベース言語記述のバス、Ｃベース言語記述の一般ハードウェア部品、及びＣベース言語記述のＩＲＳを入力する。なお、Timedソフトウェア部品としては、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品のみが入力される場合、Ｃベース言語記述のTimedソフトウェア部品のみが入力される場合、及びそれらの双方が入力される場合がある。なお、ソフトウェア部品には、前述のように、ＯＳ／ミドルウェア、割込みハンドラ、デバイス・ドライバ、タスク等が含まれる。又、一般ハードウェア部品は、ＣＰＵ、ＣＰＵ専用メモリ及びバスを除くハードウェア部品である。

【００４７】

そして、Ｃベース言語記述のTimedソフトウェア部品、Ｃベース言語記述のバス、Ｃベース言語記述の一般ハードウェア部品、及びＣベース言語記述のＩＲＳについては、そのコンパイルを行う一方、ホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品についてはコンパイルの必要はない。かかるコンパイルを実施した後、全てのソフトウェア部品及びハードウェア部品をリンクする。

【００４８】

次いで、ステップ３２０では、テストベンチを入力してコンパイルする。更に、ステップ３３０では、ステップ３１０において処理された各部品即ち検証モデルと、ステップ３２０において処理されたテストベンチとをリンクする。次いで、ステップ３４０では、Ｃベース・シミュレータによる制御の下、ステップ３３０で生成された実行プログラムにより、図７で説明されたシミュレーションが実行される。最後のステップ３５０では、そのシミュレーションの結果がディスプレイ２０等に出力されて、協調検証が終了する。

【００４９】

最後に、本実施形態における部品間の通信方式について、図１１に基づき説明すると、ソフトウェア部品とソフトウェア部品との間の通信方式としては、ＡＮＳＩ－Ｃ通信方式（Semaphore,MailBox,EventFlag）が採用される一方、ハードウェア部品とハードウェア部品との間の通信方式としては、ＳｙｓｔｅｍＣ通信方式（sc_event）が採用される。又、ソフトウェア部品とハードウェア部品との間の通信方式としては、ハードウェア起動時にあってはＳｙｓｔｅｍＣ通信方式（sc_event）が採用される一方、ハードウェア動作終了時にあっては割込み方式が採用される。

【００５０】

一方、非特許文献７～９に記載されているＣ言語ベースの設計及び検証技術の実施形態における部品間の通信方式について説明すると、ソフトウェア部品とソフトウェア部品との間の通信方式及びハードウェア部品とハードウェア部品との間の通信方式としては、ＳｙｓｔｅｍＣ通信方式（sc_event）が採用されている。又、ソフトウェア部品とハードウェア部品との間の通信方式としては、ハードウェア起動時にあってはＳｙｓｔｅｍＣ通信方式（sc_event）が採用されている一方、ハードウェア動作終了時にあってはポーリング方式が採用されている。

【００５１】

以上、本発明の実施形態について述べてきたが、もちろん本発明はこれに限定されるものではなく、例えば、図１２に示されるような、マルチＣＰＵシステムの検証モデルにも、本発明は適用可能である。

【００５２】

【発明の効果】

以上説明したように、本発明によれば、Ｃベースのネイティブ・コード・シミュレーションが実現されるため、従来のＩＳＳを使用した方法に比較して、シミュレーション性能（命令数／秒）が従来の10^2～10^3倍程度向上し、ハードウェア／ソフトウェア協調検証におけるシミュレーションの高速化が図られる。しかも、Timedソフトウェア部品に基づく時間管理が行われることで、タイミング検証の精度も維持される。したがって、本発明は、ハードウェア／ソフトウェア協調検証（特に、ソフトウェアの検証）における工数の低減に大きく寄与するものである。

【図面の簡単な説明】

【図１】ＳｏＣの上流設計フローを示す図である。

【図２】従来のハードウェア／ソフトウェア協調検証システムの構成（ソフトウェア構成）を示すブロック図である。

【図３】本発明に係るハードウェア／ソフトウェア協調検証方法を実施するためのハードウェア環境を例示するブロック図である。

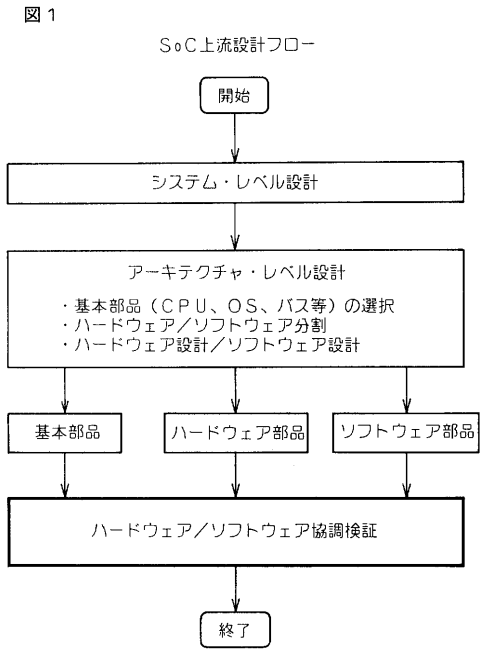
【図４】ソフトウェア部品の検証モデルの作成方法について説明するための図である。

【図５】ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである

。
【図６】ＡＮＳＩ－Ｃ記述のUntimedソフトウェア部品からＣベース言語記述のTimedソフトウェア部品への変換処理の内容について説明するための図である。
【図７】Ｃベース言語記述のTimedソフトウェア部品に基づくＣベース・シミュレーションについて説明するための図である。
【図８】ターゲットＣＰＵ用バイナリ・コードのUntimedソフトウェア部品からホストＣＰＵ用バイナリ・コードのTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。
【図９】本実施形態に係るハードウェア／ソフトウェア協調検証システムの構成（ソフトウェア構成）を示すブロック図である。
【図１０】本実施形態に係るハードウェア／ソフトウェア協調検証の手順を説明するためのフローチャートである。
【図１１】部品間の通信方式について説明するための図である。
【図１２】マルチＣＰＵシステムの検証モデルを例示する図である。
【符号の説明】
１０…コンピュータ（ＰＣ又はＷＳ）本体
１２…中央処理装置（ＣＰＵ）
１４…主記憶装置（ＭＳ）
２０…ディスプレイ
２２…キーボード
２４…マウス
３０…外部記憶装置（ハードディスク装置）

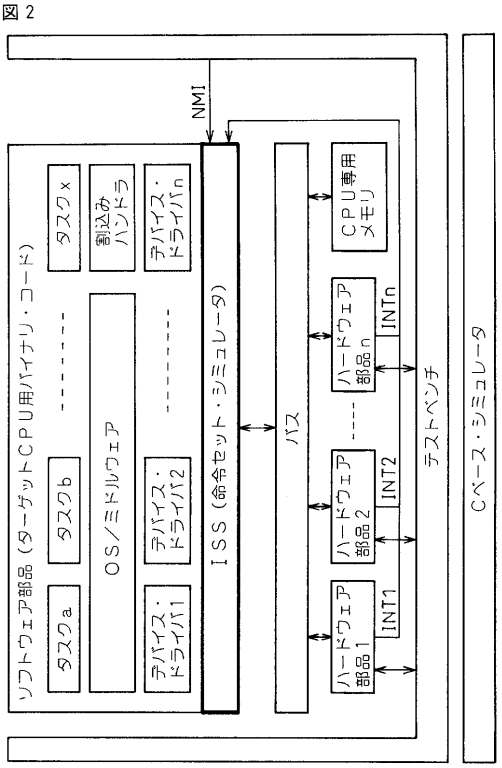
【図１】

図１



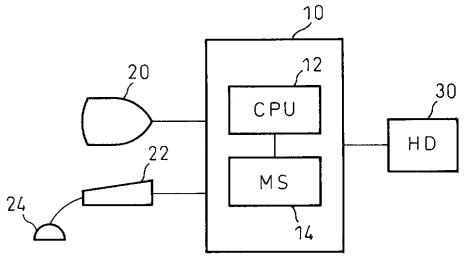
【図２】

図２



【図３】

図３



【図４】

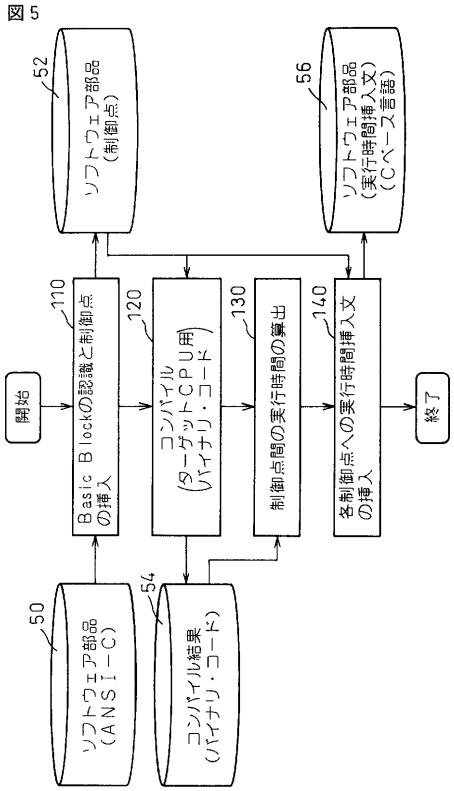
図４

ソフトウェア部品	設計論理の記述言語	検証モデルの作成方法			検証モデルの記述言語
		変換１*1	変更*2	変換２*3	
ＯＳ／ミドルウェア	ＡＮＳＩ－Ｃ	－	－	○	Ｃベース言語
割込みハンドラ	ＡＮＳＩ－Ｃ及びアセンブリ言語	○	－	○	Ｃベース言語
デバイス・ドライバ	ＡＮＳＩ－Ｃ及びアセンブリ言語	○	○	○	Ｃベース言語
タスク	ＡＮＳＩ－Ｃ	－	－	○	Ｃベース言語

*1：アセンブリ言語記述部分のＡＮＳＩ－Ｃ記述への変換（人手）
*2：「バスへの直接ＲＥＡＤ／ＷＲＩＴＥアクセス」に関する変更（人手）
*3：Ｕｎｔｉｍｅｄソフトウェア部品からＴｉｍｅｄソフトウェア部品への変換（自動）

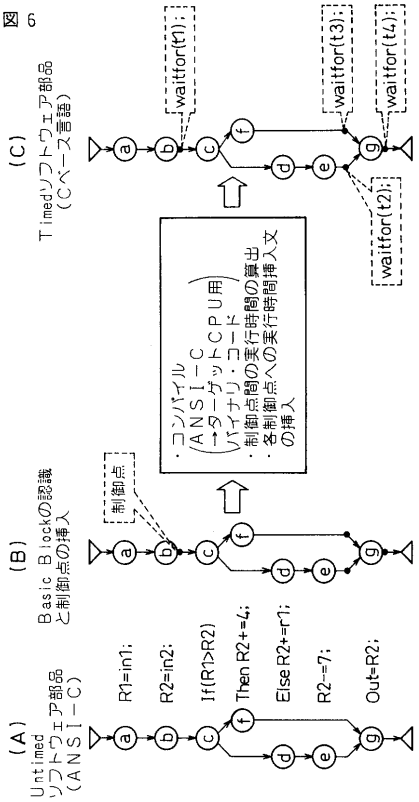
【図５】

図５



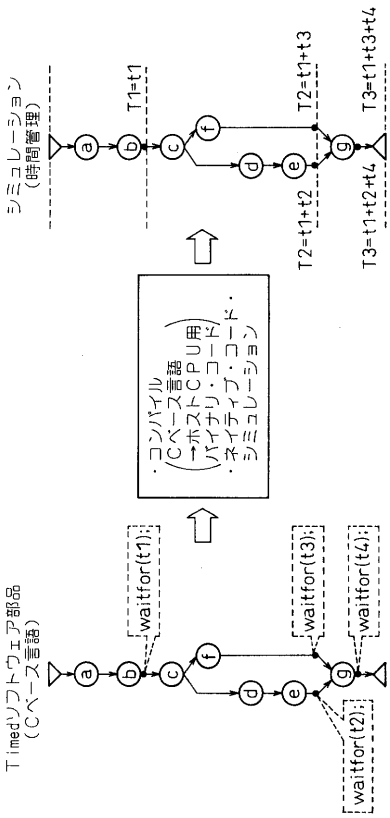
【図６】

図６



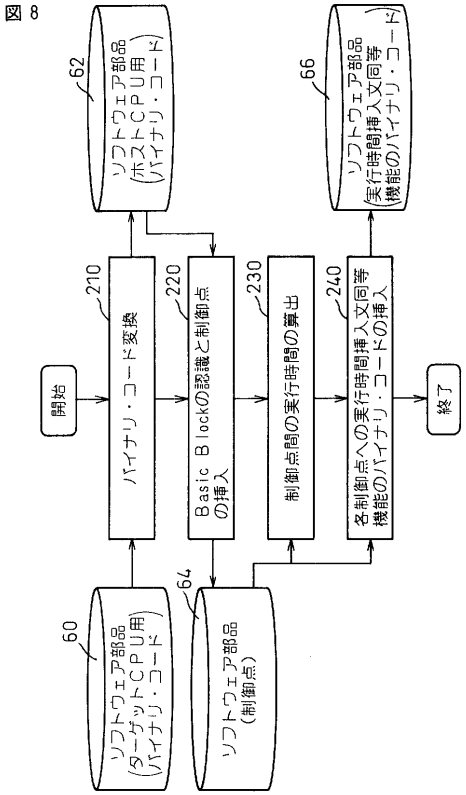
【図７】

図７



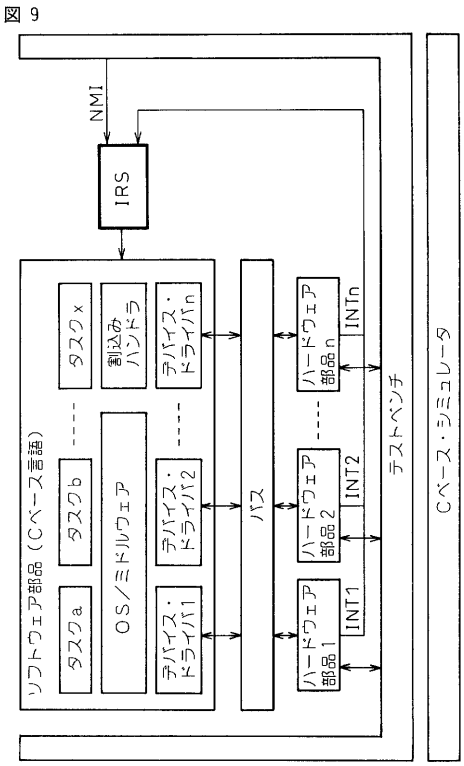
【図８】

図８



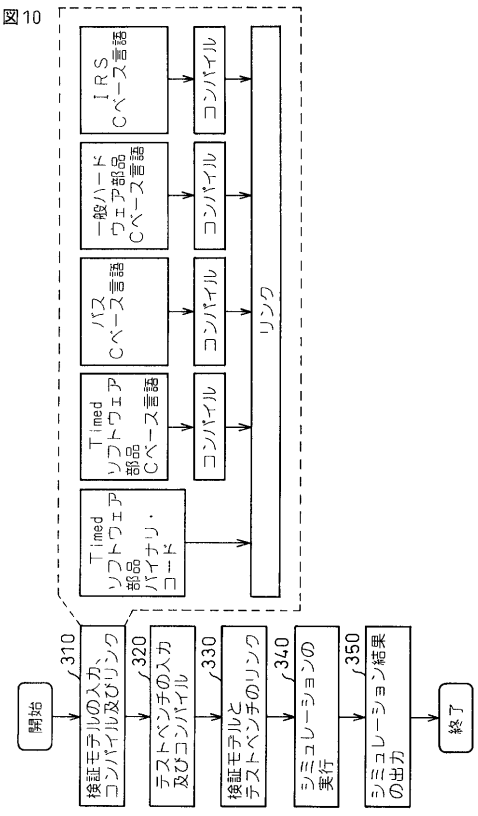
【図９】

図９



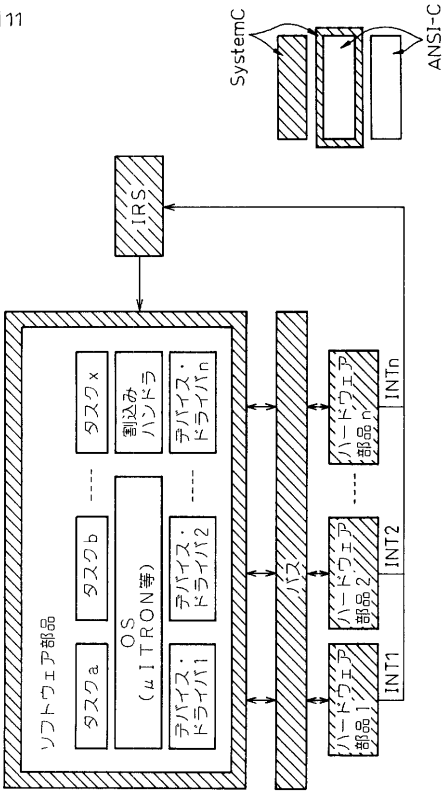
【図１０】

図10



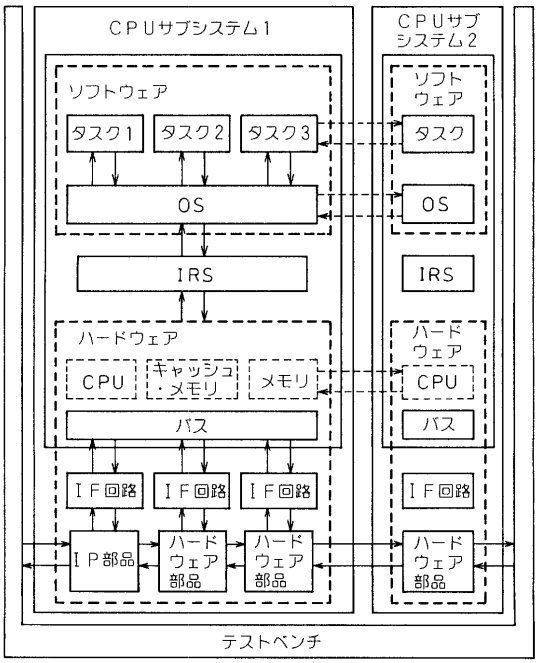
【図１１】

図11



【図１２】

図12



フロントページの続き

(72)発明者 山下 博行
神奈川県横浜市港北区大豆戸町１２１０－１ ＴＯＰ新横浜４０９号
(72)発明者 新舎 隆夫
神奈川県横浜市青葉区あざみ野３－３－２－４０４
(72)発明者 藤懸 英昭
福岡県福岡市西区内浜１－６－１－２０５
(72)発明者 古渡 俊明
東京都青梅市新町１－２－３ アーバンヒルズ河辺第２－１０８
(72)発明者 平尾 智也
東京都町田市小川２－２８－１２ ヴィラアイコウ２０４号
(72)発明者 大熊 敦史
千葉県柏市新富町２－３－１６
(72)発明者 西 宏晃
神奈川県横浜市鶴見区北寺尾７－２０－１９
(72)発明者 村岡 道明
東京都町田市金井町２３０７－４

審査官 木方 庸輔

(56)参考文献 特開平１０－１４９３８２（ＪＰ，Ａ）
安田光宏、他1名，コンポーネント論理バスアーキテクチャに基づくトップダウン コ・シミュレーション手法の提案，ＤＡシンポジウム’９８，日本，社団法人情報処理学会，１９９８年７月１６日，Ｖｏｌ．９８，Ｎｏ．９，ｐ．７－１２

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 17/50
G06F 11/28
JSTPlus(JDream2)