計算機アーキテクチャ 110－2
設 計 自 動 化 73－2
（1995. 1. 19）

大規模論理装置に対する高速設計検証システム

高山浩一郎　広瀬文保　下郡慎太郎
庄司稔　岩下洋哲

(株) 富士通研究所

従来の論理シミュレーションをベースとした設計検証システムにおいては、設計に対する入力系列の生産性や網羅性が低く、検証漏れにより設計バグを見落とす可能性があった。また、論理シミュレーションとその結果の確認の速度が遅く、検証のTATが長くなるといった問題があった。

本文では、従来の枠組みを踏襲しつつ検証能力を飛躍的に高める論理検証手法について述べる。入力系列の質と生産性を高めるために、形式的検証のアプローチを応用して検証用入力系列を自動生成する。また、構造可変なデータ並列パイプラインアーキテクチャを採用したCADアクセラレータTP5000上で高位レベルのVHDLシミュレーションと期待値比較を高速に実行することでTATを短縮する。

A High Speed Logic Verification System of Large Scale Digital Designs

Koichiro Takayama, Fumiyasu Hirose, Shintaro Shimogori,
Minoru Shoji and Hiroaki Iwashita

Fujitsu Laboratories Ltd.
1015 Kamikodanaka, Nakahara-ku, Kawasaki, 211 Japan

This paper proposes innovative strategy which drastically increases both quality and efficiency of the simulation-based verification system. In order to verify the design according with specification in short time, compact but effective input sequence is generated automatically from the specification, and high level VHDL simulation and confirming result is accelerated by a hardware. Formal verification techniques are employed to generating input sequence. Thread Processor TP5000, which is a reconfigurable processor for data parallel pipeline, is employed to accelerate both simulation and comfirming results.

1　はじめに

近年のマイクロプロセッサの性能は、年率35%向上しており[1]、回路設計規模も同率で増大する傾向にある。CADツールは一般に設計回路規模の2乗に比例した処理時間を要し、またフォンノイマンボトルネックのために、設計コストの増大がEWSの性能向上を上回ってしまう。しかし、設計者には、システムの高性能化と設計期間の短縮という相反する要求が与えられ、これに対処するには、まず、開発期間の4割を占めると言われる設計検証期間[2]の短縮が不可欠である。

形式的検証[3]は与えられた2つの論理回路の等価性判定を数学的に完全に行う。しかし、大規模な設計を検証する場合には、設計を人手により分割、抽象化を行う必要があり、完全自動化には至っていない。

設計時に装置全体を一括して検証する手段として、一般的には検証用入力系列を用いた論理シミュレーションが行われている。この検証の流れを図1－1に示す。まず、仕様をもとに検証用入力系列を作成する。検証用入力系列は、仕様から検証すべき項目を抽出し、それを実現するための入力信号列として作成される。次に、この入力系列を設計に対する外部入力として使用し、論理シミュレーションを行う。一方、同じ入力系列を正解値モデルに適用し、期待値を作成する。正解値モデルは、仕様を忠実に実現したモデルであり、汎用プログラム言語で記述されEWS上で高速に動作して検証用入力系列に対する期待値を出力する。最終的に、シミュレーション結果と期待値とを比較することで、設計が仕様を満足しているかどうかを検証する。もし、シミュレーション結果が期待値と一致しなかった場合には、設計の誤りを追跡、修正した後、再び論理シミュレーションを行う。また、仕様の修正が起こった場合には、入力系列が作り直され、この流れを繰り返す。

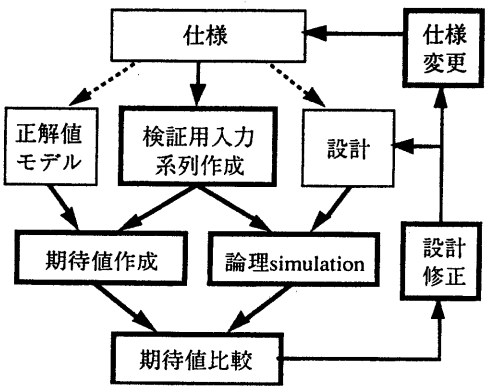


図1－1．論理シミュレーションに基づく設計検証の流れ

従来のシステムには、(1)検証用入力系列の生産性が低く網羅性の保証がない。系列長が長い。(2)論理シミュレーションの速度が遅い。(3)正解値モデルの観測性が低く、期待値の比較コストが高い。といった問題があるが、本文では、従来の論理シミュレーションをベースとした検証システムの枠組みは保存しつつ、これらの問題点に対処して全体性能を向上させるための手法を提案する。

第2章では従来システムの上記の問題点を解析して改良のポイントを明確にする。第3章では検証用入力系列の自動生成手法について述べる。形式的検証のアプローチを応用して入力系列の質と生産性を高める。第4章では高位レベルで記述された設計に対する論理シミュレーションの高速化手法について述べる。構造可変なデータ並列パイプラインアーキテクチャを採用した汎用CADアクセラレータTP5000上に、VHDLシミュレータを実装した。第5章では期待値比較を効率的に行う手法について述べる。正解値モデルの観測性を向上し、また、TP5000上に期待値比較機構を実装して高速処理を行う。第6章では、今後の課題を述べ、本報告のまとめを行う。

2　従来システムの問題点と改良のポイント

(1) 検証用入力系列の生産性が低く網羅性の保証がない。系列長が長い。

従来、検証用入力系列は人手で作成しているため、仕様の複雑化にともない開発工数が増大する。さらに、仕様に対して試験項目を網羅的に選択したかどうかを判定する手段がないため、見落とされた項目に関する設計誤りが発見できない可能性がある。そこで、一部で乱数を使用して網羅率を確率的に向上させているため、系列長が増大する。また、人手では仕様の変更に柔軟に追随することが困難であるため、特に設計の初期段階において効率の良い入力系列を使用した検証が十分にできないという問題もある。これらを解決する

ためには、検証項目を自動抽出することで100%の網羅性を保証し、各項目を検証する効率の良い入力系列を自動生成することがポイントとなる。

(2) 論理シミュレーションの速度が遅い。

近年の論理合成技術の進歩により、設計レベルが従来のゲートレベルからレジスタ転送レベル（RTL）に上がり、またVHDLのようなゲートレベルから高位レベルまで混在した設計記述を可能にした言語が世界標準となる[4]など、設計の生産性が飛躍的に向上した。一方で、論理シミュレーションをハードウエアにより高速化する研究は古くから行われている[5]が、これらはゲートレベルを対象としており、設計の初期段階において高位レベルで記述された設計の動作を検証するためには、依然としてソフトウエアのシミュレータに頼らざるを得ない。また、設計の初期には、仕様変更や誤りの修正により、同じ入力系列を用いて論理シミュレーションを繰返し実行する頻度が高く、シミュレータの速度がネックとなり検証のTATが悪くなる。これを解決するポイントとして、ハードウエアにより高位レベルシミュレーションをアクセラレートする。

(3) 正解値モデルの観測性が低く、期待値の比較コストが高い。

正解値モデルは、仕様を忠実に実現しているが、クロックなど詳細な設計情報は反映されていない。したがって、正解値モデルの出力と実設計をシミュレーションした結果は時々刻々の対応が取れていないことがある。そこで、あるまとまった入力系列を走行した後で、メモリやレジスタの内容を一旦ダンプして全ての値を比較するといった手段が取られることが多い。このとき読み出されるログデータの量が膨大になる。また、結果が期待値と合わなかった場合に、設計中の誤りの箇所を特定することが困難になる。これを解決するには、比較するデータ量を最小限に抑え、時系列に沿った期待値比較もハードウエアによりアクセラレートすることがポイントとなる。

3　検証用入力系列の自動生成手法

3．1　我々のアプローチ

大規模な論理装置を設計する際には、制御部とデータパス部に分けて設計することが、設計の誤りを防ぐためにも、後の検証や修正の手間を低くするためにも重要な手法となっている。データパスの設計は既存の設計を流用したり、小規模なモジュールが規則的に並んだ構造をとるといった特徴を持つ。それに対して、制御部の設計はほとんど新規であり、不規則で論理深度が深い順序回路となる特徴を持つ。

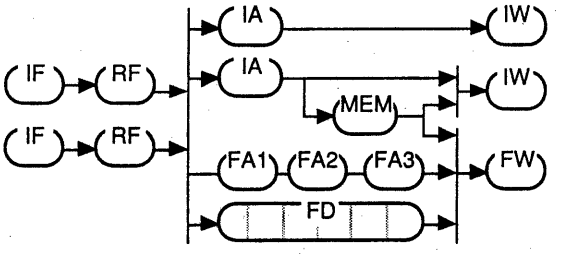
我々の提案する検証用入力系列自動生成手法は、設計の初期段階で誤りが混入しやすい制御部を検証の対象とする。本手法は制御部の仕様と検証項目を入力として受け取ると、制御部の動作モデルとして決定的有限状態機械（FSM）を作成する[6]。ここで、FSMの動作は、外部入力系列による制御部の内部状態の遷移に対応している。検証項目は、制御部の設計が仕様通りに動作するかどうかが疑わしい状況であり、すなわち、FSM上で次に正しい遷移が行われるかどうかが疑わしい状態（危険な状態）の集合として表現される。次にFSM上で、初期状態から到達可能な状態を全て求める。このとき、危険な状態の中で初期状態から到達不可能なものは検証項目から除去することにより100%の網羅性を保証する。最後に、到達可能な危険な状態から初期状態に戻るパスを逆にトレースすることで、検証用入力系列を生成する。

形式的検証により仕様と設計の等価性を検証する場合、BDD[7]の技術により、その適用規模が拡大しているにもかかわらず、内部状態数が大きな設計を処理するときにBDDのノード数が爆発する可能性があり、設計を人手で分割、抽象化する必要がある。我々の手法においては、制御部の仕様のみをモデル化すればよく、設計には手をつけず、また、状態集合への到達可能性問題のみを扱うので、状態数が爆発する危険性は大幅に小さくなり、検証のための前処理が不要となる。

3．2　プロセッサのパイプライン制御への適用

我々は、すでに、マイクロプロセッサのパイプラインの制御論理に対して前節で示した手法を適用し、パイプラインハザードが起こったときの制御の動作を検証する入力系列（検証プログラム）を自動生成することに成功している[6]。

システムの入力（パイプラインの仕様）として与えられるのは、図3－1に示すようなパイプラインを構成する資源（ユニット）およびその接続の情報と各命令が時刻を追って専有する資源の情報である。システムは、まず、各ユニットに命令が存在している状況を考え、これを内部状態とするFSMを構築する。FSMの遷移は、あるユニット



(a) パイプライン構造の仕様

	1	2	3	4	5	6	7	8	9
NOP	IF	RF							
ALU	IF	RF	IA	IW					
FMA	IF	RF	FA1	FA2	FA3				
FDiv	IF	RF	FD	FD	FD	FD	FD	FD	FW
LD	IF	RF	IA	MEM	IW				
FLD	IF	RF	IA	MEM	FW				
ST	IF	RF	IA	MEM					

(b) 命令の仕様

図３－１．パイプラインの仕様

上の命令が次時刻にどのユニットに移動するかを制御することに対応する。パイプライン中に何も命令が入っていない状態を初期状態とする。

次に、検証項目となるパイプラインハザードが発生する状況の洗い出しを行う。構造ハザードは、１つのユニットに対して２つ以上のユニットから同時刻に命令が入ろうとするときに発生する。図３－１の例においては、FWユニットに３方向から命令が入ってくる可能性があり、ここに構造ハザードが発生する。構造ハザードが発生した時の解消方法は仕様として与えられる。得られた各々のハザードが発生する状況は、先に求めたFSM中の状態の集合に対応付けられる。

次に、初期状態から到達可能なハザード状態を列挙する。その後、遷移パスを初期状態に向かって逆に辿ることにより、最終的に初期状態から全ての可能なハザード状態に至るための検証プログラムが生成される。

３．３　実験結果

本手法を図３－１に示す中規模クラスのマイクロプロセッサのパイプラインに適用して実験を行った。結果を図３－２に示す。440通りのハザード状態を検証するプログラムを5334ステップで実現した。生成に要したCPU時間は698秒

ステージ数	19
命令グループ数	7
FSM状態数	3.0 e9
到達可能なFSM状態数	3.6 e7
到達可能なケース	440
到達不可能なケース	2,775
生成時間（CPU秒）	698
検証プログラム長	5,334

図３－２．実験結果(1)

（Sun4/10）である。同じ作業を人手で行った場合、約２週間要すると見積もられるので、約400倍の速度を達成している。また、乱数により生成した命令列が同じ検証項目を網羅するために必要なステップ数と比較すると本手法により生成されたプログラム長は2000分の１以下であった（図３－３）。

別の検証項目に適用した例として、パイプライン中により多くの命令が詰まっている状態での動作を検証するためのプログラムを生成した。FDの６つのユニットは時分割されていないので、最大14命令までが同時にパイプライン中に詰まることができる可能性がある。図３－４に示すように、13命令が詰まった状態を発生する18ステップのプログラムを282CPU秒で生成した。この目標状態に到達するまでに２回のハザードが起こっている。また同時に、パイプラインに14命令詰まった状態を発生するプログラムは生成できないことを証明した。

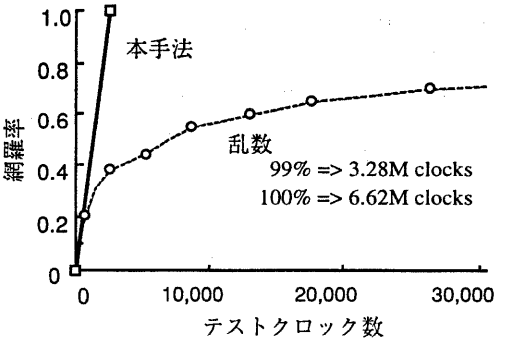


図３－３．入力系列を乱数で生成した場合との比較

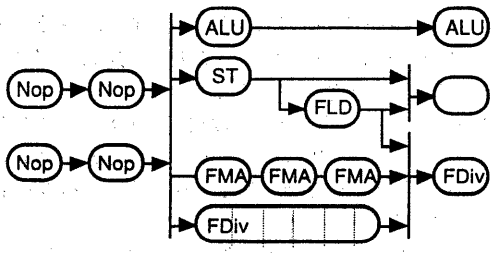


図３－４．実験結果(2)

4　VHDLシミュレータ

4．1　汎用CADアクセラレータ

CADの問題の特徴は、大量のデータに対して同じ処理を繰返し実行することであると言える。従来の専用マシンでは、大量のデータをパイプライン処理することで高速化を達成している。そのために、まず、アルゴリズムをデータの流れを基に分割する。分割された処理単位をここではスレッドと呼ぶ。次に、各スレッドでデータを並列にアクセスし、また、スレッド内部でのデータの加工やスレッド間のデータ転送をパイプライン処理するようにハードウエアを構成する。多くのCADアルゴリズムにおいては、スレッド内部のデータの処理手順は複雑であるが、スレッド間のデータの流れはほぼ一方向であり、いくつかのスレッドが大きな処理ループを形成して処理が繰り返されるといった特徴がある。

従来のほとんどの専用マシンは、処理パイプラインの構造が固定であるがゆえに単機能であった。今後、さらに設計期間を短縮するためには、より広範囲にわたって処理をアクセラレートしなければならないが、各設計段階向けに単機能の専用マシンを開発するのは非常にコストがかかる。

これまでに、プログラマブルなアクセラレータとしてMARS[8]が開発されている。MARSの要素プロセッサ（PE）は、レジスタや演算器を３本のバスで接続し、これらの動作をマイクロプログラムで制御する。また、PE間をクロスバスイッチで接続してクラスタを構成し、クラスタ間はハイパーキューブネットワークで接続されてPE間のデータ転送が行われる。PE内部のデータ処理および転送はパイプライン処理されている。しかし、バス接続はクロスバ接続に比べて自由度が小さく、前記したスレッドの特徴から考えて、PE内部のバスの本数の制約により全てのファシリティを効率良く使用したパイプラインを常に構成できるとは限らない。

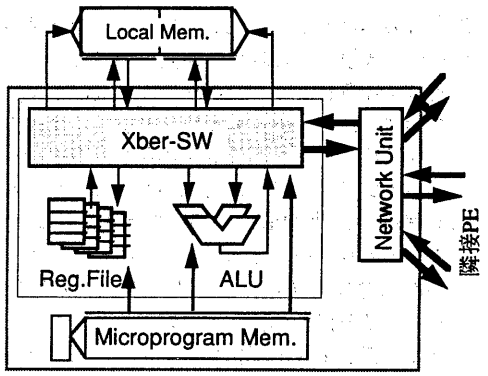


図４－１．PEのアーキテクチャ

そこで、我々は、構造可変なデータ並列パイプラインアーキテクチャを採用したThread Processor TP5000を開発した[9]。要素プロセッサ（PE）は、ローカルメモリやレジスタファイル、演算器、ネットワークの入出力ポートといった複数の機能ユニット間をクロスバスイッチで接続し、細粒度でのパイプライン処理を実現する（図４－１）。各ユニットおよびスイッチはマイクロプログラムにより制御される。各PEには４対の入出力ポートを持ったネットワークユニットがあり、１対がクロスバスイッチ、３対が隣接PEと接続してデータ転送を行う。TP5000では、10個のPEをまとめてProcessor Group（PG）を構成する。PGは、構成自由な階層ネットワークToF-Net[10]により最大512PGまで接続できる。

ToF-Net上には、PGと同列にインタフェースボードを接続することが可能で、ホストとバス接続する。TP5000はアプリケーション走行中であっても、各PEがマイクロプログラムの制御下でToF-Netを介してホストとの間でデータ転送を行うことができる。

4．2　TP5000応用VHDLシミュレータ

我々は、近年の設計の主流である高位レベル記述に対するシミュレーションをアクセラレートするために、TP5000上にイベント駆動VHDLシミュレータを実装した。本シミュレータでは、ゼロ、デルタ、ユニット、３レベルの遅延を実現している。ゼロ遅延とデルタ遅延は、それぞれ、VHDLにおけるバリアブルとシグナルへの代入に要する遅延に対応している。VHDLにはこの上位に時間の概念があるが、我々のシステムではユニット遅

延を時刻単位として実現している。本システムの目的は大規模設計の論理を検証することであるので遅延は考慮しない。図４－２にTP5000上に実装したVHDLシミュレータのパイプライン構成を示す。各処理ブロックは以下のような動作を行う。このパイプラインは、QUEとFOUTにそれぞれ２PEを使用し、計10PE（１PG）で実現される。

IN：ローカルメモリ上にあらかじめロードされた外部入力に対するイベントパケットが時刻を追って読み出され、QUEに転送される。パケットは、素子番号と変化するピンの情報（位置と値）からなる。シミュレーションを終了する時刻をあらかじめ指定できる。

QUE：３つの遅延レベルに対応したイベントキューを各２本づつ持つ。遅延の処理の優先度は、ゼロ遅延＞デルタ遅延＞ユニット遅延である。一方のキューからより優先度の高い遅延レベルにあるイベントを読み出して出力する。また、そのイベントが処理された結果生じる新しいイベントを受け取り、その遅延レベルに対応した他方のキューに登録される。

NIN：素子の入力値をローカルメモリに保持する。イベントパケットを受け取ると、古い入力値を新しい値で更新して、新しい入力値をEVに転送する。

TYP：素子のタイプをローカルメモリから読み出し、EVに転送する。

EV：受け取った入力値をタイプにより、出力値を計算する。演算結果はNOUTに転送される。評価単位となる素子は、４値モデルで16入力、16出力である。

NOUT：演算結果と、ローカルメモリに有する素子の古い出力値とを比較する。もし、異なっていたら、自身の値を更新し新しいイベントをFOUTおよびOUTに転送する。また、信号変化をトレースする必要がある素子に対してプローブフラグを立てておくことができる。

FOUT：イベントが起こった素子のファンアウトを取り出しQUEに転送する。このとき、ファンアウト素子が評価されるべき遅延のレベルも併せて転送される。

OUT：プローブフラグが立ったイベントパケットを受け取ったとき、イベントの情報を順次ローカルメモリに書き込む。シミュレーション終了後、結果がホストに読み出され解析される。

４．３　順次処理部のモデル化

従来のイベント駆動ゲートレベルシミュレータでは、ゲート間は信号線で接続され、あるゲートの出力に値の変化が起こると、そのファンアウトゲートに対して新しい値とともに次時刻に評価されることをイベントとして伝播する。つまり、値の変化と評価命令が組になって信号線上を伝播していくことでシミュレーションが行われる。

VHDLにおいて変数に対する値の代入文には同時代入文と順次代入文の２つがある。同時代入文は、右辺の式で使用される変数の値が変化したときに式が演算（評価）され、結果が左辺の変数に代入される。値が変化していたならば、次にその変数を使用するすべての同時代入文が同時に演算される。一方、順次代入文は、式を評価する順序があらかじめ決められており、その順番になると、右辺にある変数の値が変化しているかどうかにかかわらず式の演算が行われる。

VHDLシミュレータを実現する場合には、同時代入文では、右辺の演算結果が変化したとき左辺の変数を使用しているすべての同時代入文に対して値の変化と評価命令を組合わせて伝播すること

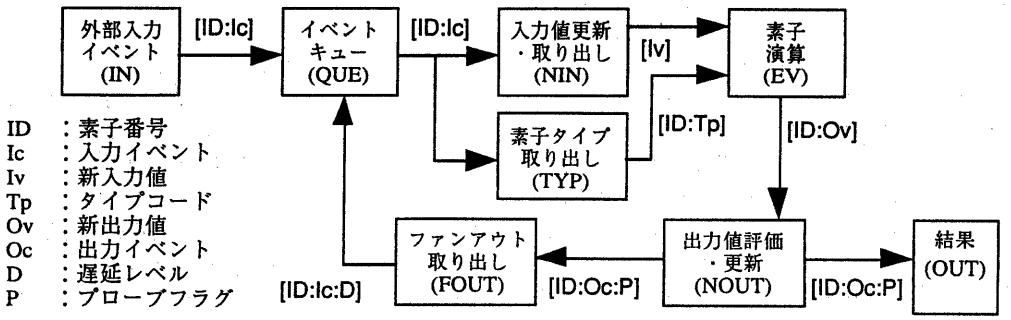


図４－２．VHDLシミュレータの処理パイプライン構成

で式の演算順序を制御する。これは、従来のゲートレベルのシミュレーションモデルと同様である。しかし、順次代入文では値の変化の伝播と式の演算順序の制御とを独立して行う必要があり、従来のモデルではこの制御を行うことができない。本シミュレータでは、値の変化の伝播を行うイベント（更新イベント）と式の演算順序を制御するイベント（評価イベント）を分離して扱うことで、順次代入文を直接モデル化[11]してシミュレーションする。シミュレータ上では、1つのプロセス文内部の順次代入文は直列に処理されるが、複数のプロセス文の処理は並列に行われるので、順次代入文を直接的にモデル化することによる性能の低下は小さい。

4．4　性能評価

我々は、これまでに8PG構成のTP5000を試作し、VHDLシミュレータを実装して大規模なゲートレベル記述に対する性能評価を行った。評価に使用した回路は、TP5000のPG間通信用のVLSIで、ゲート規模は90KBC（1BCは2入力NAND相当）である。結果を図4－4に示す。また、Sun4/10（88MB）上で走行する市販のソフトウエアシミュレータとの速度比較を行った。横軸は、並列動作させたLSIの接続個数を示す。縦軸は対象回路の動作周波数（Hz＝処理サイクル数／秒）を表す。ソフトウエアシミュレータでは、8個を越える規模については、メモリ量の制約によりページフォールトが頻繁に発生し、CPU時間に対する経過時間の比が大きくなり実用時間内で処理が終了しなかったため見積りを行った。

本シミュレータは、ソフトウエアの360倍の速度を達成しており、TP5000が汎用的なプログラマブルハードウエアでありながら、従来の単機能アクセラレータと同等の性能[12]を達成していることがわかる。

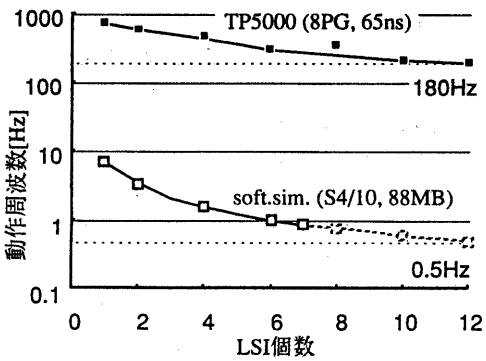


図4－4．性能評価

現在、高位レベル記述に対する性能を評価中であり、ここでは速度の見積りを行う。20万行のRTL記述をコンパイルした後のシミュレーションプリミティブ数が10万になるとする。1サイクルあたりのイベント率を35％とし、8PG構成のTP5000（65ns）を使用したときの並列効果を0.6とする。また、本シミュレータは2クロックで1イベントを処理するので、動作周波数は、

1 / (100K * 0.35 * 2 * 65e-9 / (8 * 0.6))
= 1054 [Hz]

となる。市販のシミュレータに比べて100倍程度、従来のゲートレベルのアクセラレータと比較しても10倍程度の高速化が達成できる。

5　期待値の自動比較の高速化手法

クロックの概念を持たない動作モデルが出力する期待値と、設計をシミュレーションして得られる結果とは次々刻々の比較ができるとは限らない。しかし、ほとんどの場合、比較すべき値の順序は両者間で一致している。したがって、他の信号の値や変化を条件として比較するタイミングを取ることができれば時刻の枠を超えた期待値比較が可能となる。そこで我々は、図5－1に示すようにイベント駆動期待値比較手法を提案する。

比較対象となる信号を、比較するタイミングが同一となるような部分集合に分割する。各集合ごとに2本のキューを用意し、一方に期待値（eQ）を、他方にシミュレーション結果（sQ）をイベント列として格納する。sQには、比較される信号（tS）の結果に加えて、比較条件を判定するために使用される信号（cS）の結果を格納する。sQから取り出した信号がtSのイベントである場合には現在の値を更新し、cSのイベントである場合には条件判定部へ送られる。比較するタイミング条件が成立したとき、eQから期待値を取り出し、現在の値と一致しているかどうか確認する。一致しなかったときはエラーを出力する。そうでなければ、以上の処理を繰り返す。

ところで、従来はシミュレーション結果を読み出してホスト上で期待値比較を行っているが、読み出されるデータには期待値を比較するには不要なイベントも含まれているので、扱うデータ量が大きくなる。また、シミュレーション後に比較を

行うので処理が直列となる。そこで、TP5000上にイベント駆動期待値比較手法を実装して論理シミュレーションと並列に処理を行う。まず、eQをPEのローカルメモリ上に実現する。動作モデルは各時点での期待値の差分をイベントとして出力し、これをシミュレーション開始前にeQに転送する。sQの代わりに、並列して動作する論理シミュレータからの対象信号のイベントを動的に入力して比較タイミング条件の観測と現在値の更新を行う。タイミング条件が成立したとき、現在値と期待値とを比較する。一致していたならば、次の期待値イベントを読み出して期待値を更新しておく。一致していなければ、処理を停止してホストに通知する。このとき、期待値が一致しなかった時点の回路内部の状態が保持されているので、設計誤りの追跡が容易になる。また、ホストとTP5000間の転送データ量が最小になるとともに、論理シミュレーションと期待値比較を並列処理することで、検証のTATが短縮できる。

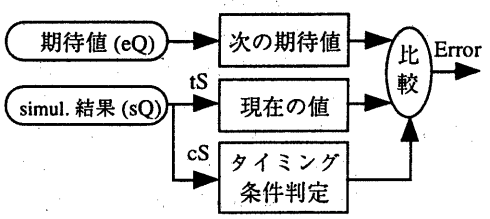


図５－１．イベント駆動期待値比較

6　おわりに

論理シミュレーションをベースとした設計検証システムの枠組みの中で、検証用入力系列を仕様から自動生成し、論理シミュレーションと期待値比較をCADアクセラレータにより高速化することで、設計検証の質を向上し、かつTATを短縮する手法について述べた。

入力系列の自動生成については、プロセサのパイプライン制御部に適用し、人手の400倍の速度で乱数の2000分の１の短さの検証プログラムを自動生成した。現在、仕様記述フォーマットの一般化を行っており、今後は、プロセサの割り込み制御やキャッシュ制御への適用を行う。

再構成可能なデータ並列パイプラインマシンであるTP5000上にVHDLシミュレータを実装し、高位レベル記述を直接扱うことで従来のゲートレベルのシミュレーション専用マシンに比べて、10倍の高速化を達成した。さらに、検証のTATを短縮するために、従来シミュレーション終了後にホスト上で実行していた期待値比較をTP5000上で論理シミュレーションと並行してイベント駆動で行う手法を提案した。今後は、本手法をTP5000上に実装してシステム全体の性能評価を行う。

参考文献

[1] J. L. Hennessy et al.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 1990.

[2] R. Peck et al.: Design methodology for a MIPS compatible embedded control processor. Proc. of IEEE Int. Conf. on Comput. Design, pp.324-328, 1991.

[3] 平石, 藤田 (編): 論理設計の形式的検証. 情報処理, Vol. 35, No. 8, pp.708-750, 1994.

[4] IEEE Standard VHDL Language Reference Manual. IEEE Std. 1076-1987, IEEE, 1988.

[5] T. Blank: A survey of hardware accelerators used computer-aided design. IEEE Design and Test of Computers, Vol. 1, pp.21-39, Aug. 1984.

[6] H. Iwashita et al.: Automatic Test Program Generation for Pipelined Processors. Proc. of IEEE Int. Conf. of Comput. Aided Design, pp.580-583, 1994.

[7] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Comput., Vol. C-35, No. 8, pp.677-691, 1986.

[8] P. Agrawal et al.: MARS: A Multiprocessor-Based Programmable Accelerator. IEEE Design and Test of Comput. pp.28-36, Oct. 1987.

[9] 下郡 他: CAD高速化のためのThread Processor TP5000. FGCS'94 ワークショップ「並列／分散処理によるLSI-CAD」, pp.17-24, 1994.

[10] H. Matsuoka et al.: Topologically Flexible and Highspeed Network (ToF-Net). Proc. of IEEE Int. ASIC Conf. and Exhibit., pp.300-303, 1994.

[11] M. Shoji et al.: VHDL compiler of behavioral descriptions for ultrahigh-speed simulation. Proc. of 2nd Asian Pacific Conf. on Hardware Description Languages, pp.85-88, 1994.

[12] F. Hirose: Performance Evaluation of an Event-Driven Simulation Machine. Proc of 29th Design Automation Conf., pp.428-431, 1992.