時間優先評価アルゴリズムによる論理シミュレーションの高速化†

石浦 菜岐佐†† 安浦 寛人†† 矢島 脩三††

論理シミュレーションは論理回路の設計検証の一手法としてきわめて広く用いられている．近年の半導体技術の進歩による実現可能な回路の大規模化は，論理シミュレーションの計算時間を急速に増加させており，処理の高速化に対する要望はますます高まっている．ゲート・レベル・シミュレーションに対して従来から用いられてきたタイム・マッピング・イベント法は，時刻を逐次進めながら各時刻における全回路のイベントを処理していく「空間優先評価」の計算方法であるといえる．これに対し本論文では，回路中の各ゲートに対し，そのゲートに関して処理可能なイベントをまとめて処理してゆく「時間優先評価」の計算方法（Tアルゴリズム）を新しく提案する．本方法はとくに組合せ回路のシミュレーションに有効であり，実際に試作したシミュレータを用いた比較実験により，従来の方法に比べて 7〜8 倍高速であるという結果が得られた．また，本方法は本質的にデータフロー型のアルゴリズムであるため計算の並列性が高く，専用ハードウェアによるさらに高速な計算への応用も期待できる．

1. まえがき

近年の半導体技術の進歩に伴い，実現可能な論理回路の規模はますます大きくなっている．このため，回路設計を誤りなく，しかも短期間に行うことは，人手では困難になり，計算機を利用した設計が用いられるようになっている．論理シミュレーションは，計算機上で対象回路の論理的な動作を模擬する設計検証の一手法である．大規模な集積回路（LSI/VLSI）が実用される今日では，回路の設計誤りは LSI/VLSI の再製作につながり多大なコストと長い期間を要するため，実物を作成する以前に十分な設計検証を行っておくことが不可欠となっている．仕様と設計の形式的な記述から論理設計に誤りのないことを導く検証理論の研究は盛んではあるが，一般的な実用にはほど遠く，論理回路の設計検証はもっぱら論理シミュレーションの助けを借りて行われているのが現状である[1),2)]．

論理シミュレーションに要する計算時間は，おおざっぱにいって回路の規模と回路の入力となるテストパタンの長さに比例する．論理回路の大規模化は必然的にテストパタンの増大をも招き，シミュレーションの計算時間を急速に増大させており，論理シミュレーションの高速化に対する要望はいっそう高まっている．

論理シミュレーションは，論理ゲートをシミュレーションの単位とするゲート・レベル・シミュレーションと，それよりも複雑な機能をもった機能ブロックを単位とする機能レベル・シミュレーションに大別できる[1)]．ゲート・レベル・シミュレーションに対しては従来から，イベント（信号値の変化）に注目するタイム・マッピング・イベント法が広く用いられてきた[1)-3)]．これは，時刻を逐次進めながら各時刻における全回路中のイベントを処理してゆく「空間優先評価」の計算方法（Sアルゴリズム）であるといえる．これに対し本論文では，回路中の各ゲートに対し，そのゲートに関して評価可能なイベントをまとめて処理してゆく「時間優先評価」の方法（Tアルゴリズム）を新しく提案する[5)]．実際にこの方法に基づく論理シミュレータを試作し実験を行った結果，とくに組合せ回路のシミュレーションに有効であり，従来の方法に基づいて作成したシミュレータに比べて 7〜8 倍高速であることがわかった．また本方法は，本質的にデータ駆動型のアルゴリズムであるため計算の並列性が高く，専用ハードウェアによるさらに効率のよい計算への応用が期待できる．

本論文では，まず2章で論理回路のモデル化と論理シミュレーションに関する基本事項について述べ，3章で T アルゴリズムによる論理シミュレーションについて述べる．4章では T アルゴリズムの組合せ論理回路のシミュレーションへの適用について述べ，その実験結果に基づいて処理速度，使用記憶量等の評価を行う．順序回路のシミュレーションやシミュレータのハードウェア化等アルゴリズムの応用に関する考察は5章で行う．

2. 論理回路のモデルと論理シミュレーション

論理回路は外部入力，外部出力，ゲートおよびその

† High-speed Logic Simulation by Time First Evaluation Algorithm by NAGISA ISHIURA, HIROTO YASUURA and SHUUZO YAJIMA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

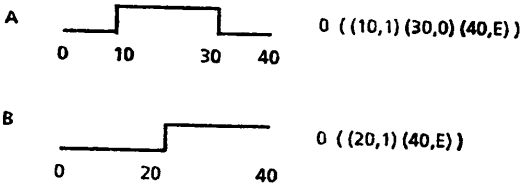

図 1 信号値の表現例
Fig. 1 Expression of signal values.

間の結線関係によって定義される．ゲートはn本の入力線（nは自然数）と1本の出力線をもち，定められたn変数論理関数を実現する．各ゲートでの論理関数の計算にはゲートごとに定められた遅延を伴うものとする．ゲートのファイン数とは，そのゲートの入力線の数であり，ファンアウト数は，そのゲートの出力線につながる他のゲートの入力線および外部出力の数をいう．論理シミュレーションは，論理回路の回路記述と，外部入力に対する各離散時刻ごとの信号値から，外部出力（さらには各ゲートの出力線）の各離散時刻における信号値を計算するものである．

ゲートの遅延を考慮した論理シミュレーションでは，計算の効率化のため，ゲートの入力線の信号値の変化（イベントという）に対してのみゲートの論理関数の計算を行うイベント法が用いられる．これに対応して，信号線の信号値は図1に示すように初期値とイベントのリストで表現される（Eはこの時刻で信号値が終了していることを表す）．また，回路情報は図2に示すような表（テーブル）の形式で記憶され，計算の必要に応じてテーブル参照が行われる．

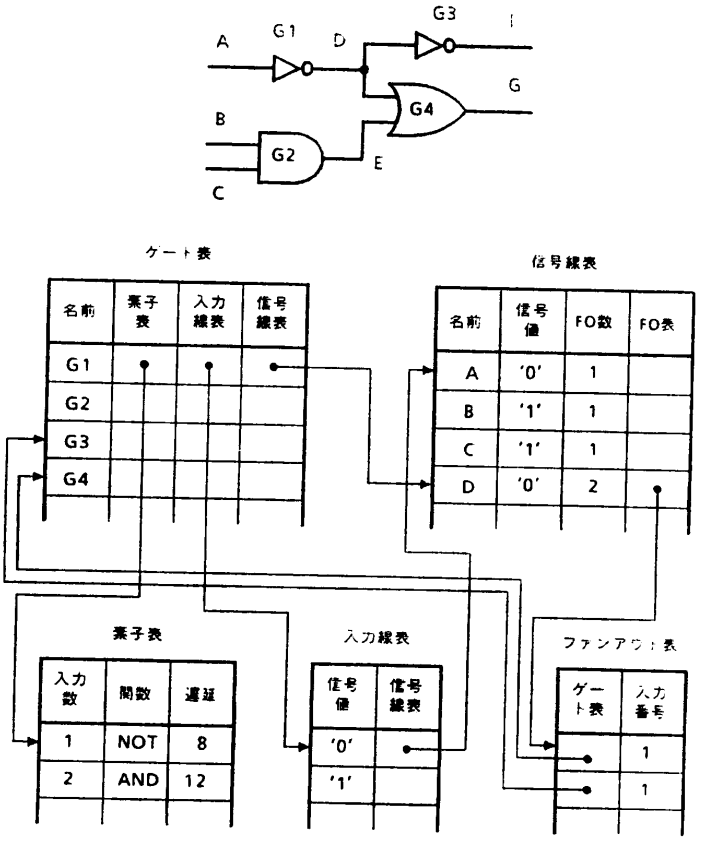

図 2 回路モデルと表
Fig. 2 Circuit model and tables.

3. 時間優先評価アルゴリズム—Tアルゴリズム

3.1 概　　要

ゲート・レベル・シミュレーションにおいて広く用いられているイベント駆動方式では一般に，

（1） 既知のイベントを取り出す．
（2） そのイベントの影響を評価する．
（3） その結果新しいイベントが発生すれば登録する．

というステップを繰り返してシミュレーションを進めていく．この三つの操作の処理効率はシミュレータの性能を大きく左右する[3]．

タイム・マッピング・イベント法はタイム・ホイルの導入によってイベント・リスト法における(3)の処理効率を改善したものといえる[1),3)]．ここで，イベントの処理順序に注目するといずれの方法も，発生時刻が回路全体で最小となっているイベントから順に処理していく．これは，イベントの因果的順序関係を誤らずにシミュレーションを進めていく最も確実な方法といえる．

しかし，論理ゲートは入力線の影響によってのみ出力を変化させるのであるから，すべての入力線に対するイベントの系列が確定していれば，他の要因とは無関係に出力線のイベントを計算できる．たとえば図3において信号線A，B，Cに発生するイベントがそれぞれ時刻 200，150，100 まで確定しているとする．ゲートの遅延をdとすれば，ゲート G1 の出力線上のイベントは $(150+d)$ 時刻まですべて計算でき，その結果ゲート G2 の出力も $(100+d)$ 時刻まで計算可能となる．このように，出力が計算可能なゲートに注目して，

表1 変数一覧
Table 1 List of variables.

変数名	意味
n	ゲートの入力数
delay	ゲートの遅延
inlist (i)	ゲートの i 番目の入力となる信号線のイベント・リスト
initiv (i)	(initial input value) ゲートの i 番目の入力となる信号線の初期値
outlist	ゲートの出力線のイベント・リスト
initov	(initial output value) ゲートの出力線の初期値
nextevent (i)	ゲートの i 番目の入力線に次に発生するイベントで，nextevent (i). time はその発生時刻を nextevent (i). value はその新しい信号値を表す
lastevent (i)	inlist (i) の最後の要素（イベント）で，lastevent (i). time はその発生時刻を表す
eventpointer (i)	inlist (i) において次に発生するイベント nextevent (i) をさすポインタ
currenttime	ゲートがどの時刻まで処理されたかを示す
currentiv (i)	(current input value) ゲートの i 番目の入力線の currenttime における値
currentov	(current output value) ゲートの出力線の currenttime における値
egset	(SET of Evaluatable Gate) 評価可能ゲートの集合
igset	(SET of Initializable Gate) 初期化可能ゲートの集合

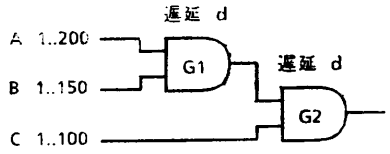

図3 回路例
Fig. 3 An example.

そのゲートに関する全時間の計算を行っていく「時間優先評価」の方法（Tアルゴリズム）は，従来の方法において1イベントの処理ごとに必要であった空間位置に関する情報（ゲートの種類，遅延，結線関係等）のテーブル参照が，1ゲートの処理に1回で済み，(2)の処理効率が大幅に改善される．

以下，このTアルゴリズムの詳細を述べる．まず各ゲートについて入力線のイベントの系列から出力線のイベント系列を計算する手続きについて，次に，どの順にゲートを処理するかを制御する手続きについて，最後にこれに先立って必要となる初期化の手続きについて述べる．各手続きで使用する変数とその意味の一覧を表1に示す．

3.2 各ゲートの出力の計算方法

ゲートの出力線のイベントは，全入力線に発生するイベントを時刻の順に取り出してその影響を評価することにより求めてゆく．手続き「ゲートの出力計算」を次に示す．ただし，初期条件は3.4節で述べる初期化手続きの実行後成立する条件である．

初期条件

$i=1\sim n$ の入力について

eventpointer (i) は inlist (i) の先頭の要素をさしている．

currentiv (i) には i 番目の入力線の初期値がセットされている．

currentov には入力線の初期値より計算した出力値がセットされている．

outlist の最後の要素は (t, E) である（i はある時刻）．

手続き「ゲートの出力計算」

step 0　outlist の最後の要素 (t, E) を削除する．

step 1　すべての入力 i ($i=1\sim n$) のなかで nextevent (i). time の最小値を newtime とし，これを与える入力（複数存在しうる）を変化入力 i_k ($k=1\sim m$) とする．

step 2　currenttime: =newtime とする．もし，少なくとも一つの変化入力 i_k ($k=1\sim m$) に対して nextevent (i_k). value=E であれば step 6 に行く．

step 3　すべての変化入力 i_k について currentiv (i_k): =nextevent (i_k). value とし，currentiv の値から新しい出力値を計算し newov とする．

step 4　もし newov が currentov と異なっていれば currentov: =newov とし，(currenttime+delay, newov) を outlist の最後に付加する．

step 5　すべての変化入力 i_k について eventpointer (i_k) を一つずつ進め，step 1 に戻る．

step 6　outlist の最後に (currenttime+delay, E) を付加して終了する．

たとえば図4の場合計算は次のように行われる．

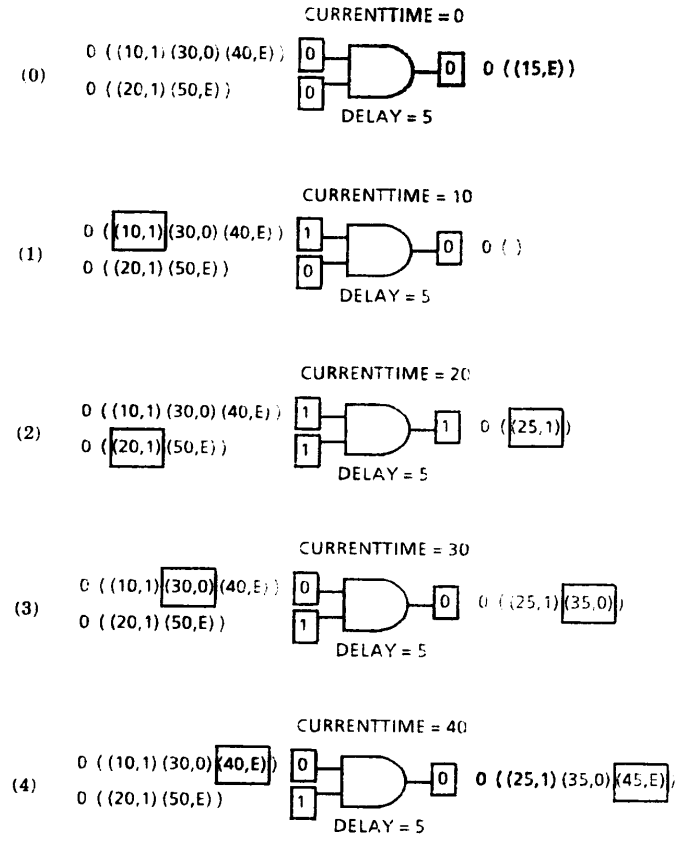

図 4 ゲートの出力計算例
Fig. 4 Calculation of gate output.

(0) 初期状態である.

(1) 出力線のイベント・リストから (15, E) を削除する. 入力線のイベントのうち時刻が最も小さい (10, 1) を取り出し currenttime を 10 まで進める. 新しい入力から出力を計算する. 出力が 0 のまま変化しないので, 新しいイベントを登録する必要はない.

(2) 次のイベント (20, 1) を取り出す. 出力が 0 から 1 に変化するのでゲートの遅延を考慮したイベント (25, 1) を登録する.

(3) 次のイベント (30, 0) を取り出す. 新しい出力は 0 なのでイベント (35, 0) を登録する.

(4) 次のイベント (40, E) を取り出す. 信号値が E なので, イベント (45, E) を登録して終了する.

3.3 ゲートの処理順序の制御

ゲートの処理は, イベントがすべての入力線にそろったゲート (評価可能ゲート) から行っていく. 手続き「ゲートの処理順序制御」を次に示す. ここで評価可能ゲートとは, すべての入力 i (i=1〜n) に対して

currenttime<lastevent (i). time

を満たすゲートをいう.

手続き「ゲートの処理順序制御」

step 0 初期化の結果評価可能となったゲートをすべて egset の要素とする.

step 1 egset よりゲートを取り出し, ゲートの出力計算を行う.

step 2 このゲートの出力を入力としているゲートが評価可能になっていれば egset に加える.

step 3 egset が空ならば終了する. さもなければ step 1 に戻る.

3.4 初 期 化

3.2, 3.3 節の手続きに先立って必要な初期条件を設定する手続き「初期化」を次に示す. ただし, ゲートの出力線がフィード・バック線かどうかは手続き「フィード・バック線の判定」で判定する. また初期化可能ゲートとは, すべての入力 i (i=1〜n) に対して

inlist (i)≠( )

を満たすゲートをいう.

手続き「初期化」

step 1 外部入力の初期値とイベント・リストを読み込む.

step 2 すべてのゲートについて

currenttime=$-\infty$;

currentov=X; (X は不定値を表す)

outlist=( );

とする.

出力線がフィード・バック線であるゲートに限り

outlist=((∞, E))

とする.

step 3 初期化可能なゲートを igset の要素とする.

step 4 igset からゲートを取り出し initiv の値からゲートの出力を計算し newov とする. nextevent (i). time の i=1〜n にわたる最小値を nexttime とする.

step 5 もし currentov≠newov または currenttime≠nexttime であれば,

currentov : =newov;

currenttime : =newtime;

outlist : =((currenttime+delay, E));

とする.

step 6　このゲートの出力を入力とするゲートが初期化可能になっていれば igset に加える.

step 7　igset が空であれば終了.

さもなければ step 4 に戻る.

手続き「フィード・バック線の判定」

step 1　すべてのゲート g について

visited (g): ='no';

ascendant (g): ='no';

とする.

step 2　回路の外部出力につながるゲートを g_{xj} (j=1～m) とする.

j=1～m について g_{xj} をパラメータとして手続き「ループの探索」を呼び出す.

手続き「ループの探索」 g：パラメータ

step 1　visited (g)='yes' かつ ascendant (g)='no' なら何もしないで終了する.

visited (g)='yes' かつ ascendant (g)='yes' ならこのゲートの出力線がフィード・バック線であることを記録し，終了する.

visited (g)='no' ならば，step 2～step 4 を実行する.

step 2　visited (g): ='yes'; ascendant (g): ='yes'; とする.

step 3　ゲート g の入力 i (i=1～n) に信号値を供給するゲートを g_i とする.

i=1～n について g_i をパラメータとして手続き「ループ探索」を再帰的に呼び出す.

step 4　ascendant (g): ='no' とする.

4. 組合せ回路のシミュレーションへの応用

4.1 処理速度

前章で述べたTアルゴリズムを，実際に組合せ回路のシミュレーションに適用し，その処理速度，使用記憶量の特性に関する実験を行った．**表2**は，Tアルゴリズムと従来のタイム・マッピング・イベント法に基づくシミュレータの処理速度をいくつかの組合せ回路について測定したものである．この表より，Tアルゴリズムに基づくシミュレータは組合せ回路のシミュレーションにおいては従来のものより7倍程度高速であることがわかるが，その理由としては次が考えられる.

(1)　従来のアルゴリズムにくらべてテーブルの参照回数が少ない.

表2　処理速度の比較
Table 2　Comparison of simulation speed.

回　路	ゲート数	従来のアルゴリズム	本論文のアルゴリズム
8ビット加算器1	48	6,602	49,948
8ビット加算器2	72	7,565	56,737
8ビット乗算器1	400	7,641	48,969
8ビット乗算器2	568	8,667	57,918

単位　　イベント/CPU 秒
入力パタン　18,000 シミュレーション・ステップ
使用計算機　HITAC M-240 H (2.3 MIPS)

(2)　イベントの登録の操作が単純である.

タイム・マッピング・イベント法では，1イベントの処理ごとに，

- イベントの発生した出力線の影響を受けるゲートを捜し（2回のテーブル参照），これらのゲートに対し，
- 入力線の値を準備する（入力線の数だけ）.
- ゲートの種類を調べる（1回）.
- 出力の値を計算する（1回）.
- 出力信号値の変化をチェックし（1回），変化していれば遅延時間（1回）後のイベントとして登録する.

という処理が必要になり，合計，

[2+ファンアウト数×(入力数+4)] 回

のテーブル参照が必要となる．これに対しTアルゴリズムでは，出力の計算以外のテーブル参照は1ゲートの処理に対して1回ずつでよく，一度に処理されるイベント系列が長くなればなるほどテーブル参照によるオーバヘッドの比率は小さくなる.

また，イベント登録については，従来のアルゴリズムでは，(現時刻+遅延) 時刻のタイム・ホイル上での相対位置を求め，その位置にあるリストを操作しなければならないが，Tアルゴリズムでは，そのゲートの出力線のイベント・リストの最後に要素を付加するだけでイベント登録が完了する.

ゲートの1入力の信号値変化に対する処理に必要な機械語命令数 (HITAC M-240 H) を調べてみると，

従来のアルゴリズム　$120+16F_i+38F_o$

Tアルゴリズム　$29+7F_i$

(ただし，F_i, F_o はそれぞれファンイン数，ファンアウト数の平均値) となっている.

4.2 使用記憶量

Tアルゴリズムによる論理シミュレーションは従来の方法に比べてイベント・リストのための記憶域が

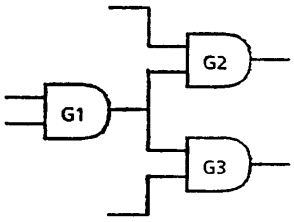

図 5 ファンアウトのある回路
Fig. 5 A circuit with fanout.

余分に必要となる．図5に示す回路において，G1の出力イベント・リストは，いちいちG2, G3に転送していたのでは時間的にも記憶量の面でも効率が悪いので，ポインタを用いてG2, G3から共有的にアクセスし，すべてのファンアウト先のゲートの計算が終了した時点で記憶域を解放する．この際，イベント・リストのために必要な記憶量（すなわち計算の各段階において保持すべきリスト長の合計の最大値）ができる限り小さくなるように，ゲートの処理順序を決定することが重要となる．しかしこの問題は，すべてのリストの長さが一定であると仮定し，保持すべきリストの数の最大値を最小にするという問題に簡単化した場合でさえ Register Sufficiency Problem[4] の変形と考えられ，NP 困難である．したがって，ここではヒューリスティックな手法により，ゲートの処理順序の決定を行うことを考える．

いま，シミュレーションのために，保持しなければならないリストの数に注目すると以下のことがわかる．

（1） ファンアウト数が1の出力線を入力するゲート（タイプ1と呼ぶ）を処理した場合，リストの数は

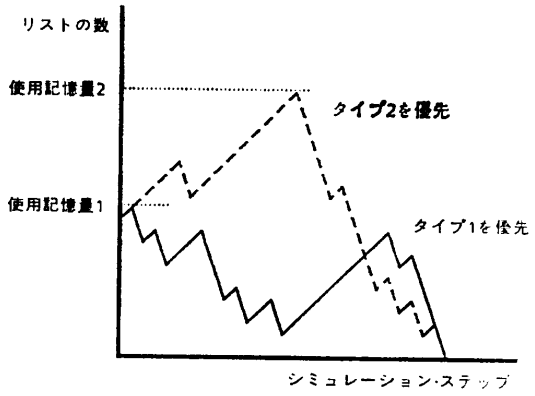

図 6 使用記憶量
Fig. 6 Memory size.

1増えてから（出力リストの分），入力数だけ減る（入力リストが解放される分）．

（2） ファンアウト数が2以上の出力線を入力とするゲート（タイプ2と呼ぶ）を処理した場合，出力リストを計算した後も，すぐには入力リストの解放が行えないことがある．

したがって，図6からもわかるように，タイプ2のゲートよりもタイプ1のゲートを優先して処理したほうが，保持すべきリストの数の最大値を小さくできる．実際にいくつかの組合せ回路に関する実験から，タイプ分けをしない場合に比べてリストのための使用記憶量が 50〜60% におさえられるという結果が得られた．さらにこの方法は，3.3節で述べた egset を処理するスタックをタイプ別に2本用意することで容易に実現でき，スケジューリング手法として有効であると結論できる．

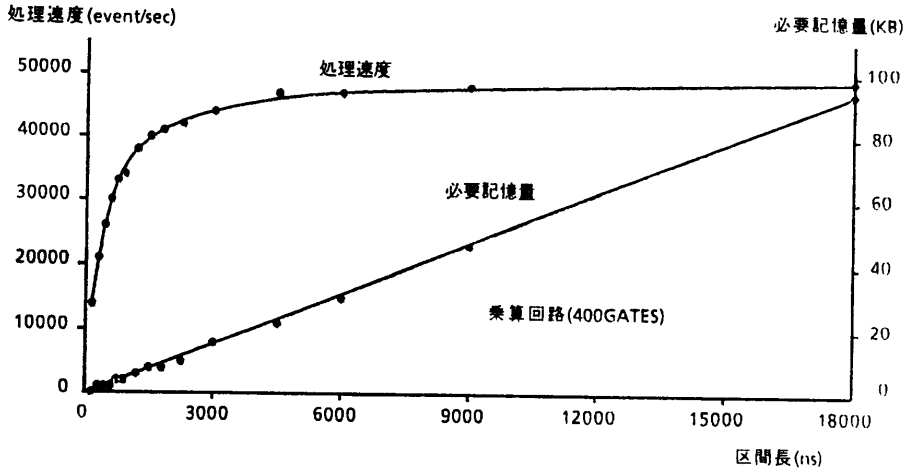

図 7 処理速度と必要記憶量
Fig. 7 Simulation speed and memory size.

4.3 処理速度と使用記憶量のトレード・オフ

4.1 節の(1)より，一度に処理されるイベントの数が多いほど，1イベントあたりのテーブル参照のオーバヘッドが減り，処理効率は上がるといえるが，それにしたがって，イベント・リストのために必要な記憶量も増大する．**図7**はTアルゴリズムに基づくシミュレータにおいて 18,000 のシミュレーション・ステップをいくつかの区間に分割し，何回かに分けて処理した場合の処理速度と使用記憶量の関係を示している．この区間長をさまざまに変えて実験を行い次の結果を得た．

(1) イベント・リストのための記憶量は区間長に比例する．

(2) 処理時間＝区間数×オーバヘッド（1ゲートの処理に必要なテーブル参照等の時間)＋出力イベントの計算時間

したがって一定ステップのシミュレーションを行う場合，グラフからも読み取れるように，区間長は少し長くとるだけでオーバヘッドが著しく削減され，極端に記憶量を大きくとらなくても最大値に近い処理速度を得ることができる．

5. Tアルゴリズムの応用

5.1 ゲート・遅延のモデル

本論文では，ゲートのモデルとして多入力1出力の記憶を持たないものを取り上げたが，一般にはフリップ・フロップ等多出力で記憶をもった機能素子も同様に扱うことができる．また，遅延に関しては標準遅延をもつゲートのシミュレーション方法だけを述べたが，従来のアルゴリズムに適用可能なすべての遅延モデルを扱うことができる．立上り／立下り遅延，最大／最小遅延，慣性遅延といった複雑な遅延モデルを扱う場合，イベントの取消し（特定のゲートに関して一度登録したイベントをすべて取り消す）という操作が必要になり，従来これが効率を落とす原因となっていた．Tアルゴリズムでは，一つのゲートに関するイベントはすべて一つのリストにつながれているため，簡単なリスト操作でイベントの取消しができ，処理効率を落とすことなく複雑な遅延モデルを扱うことができる．また，Tアルゴリズムは零遅延シミュレーションに対しても効率よく適用することができる．

5.2 フィード・バック・ループの問題

Tアルゴリズムに基づいて，非同期回路も含めた一般の論理回路を扱えるシミュレータを作成し，実験を行った結果，短いフィード・バック・ループを含む非同期回路の場合は，組合せ回路の 1/10～1/100 の処理効率しか出ないことがわかった．これは，回路中にループが存在すると，4.3 節で述べた区間長がそのループ長（遅延時間）でおさえられ，1イベントあたりのテーブル参照のオーバヘッドが大きくなるためである．とくに**図8(a)** の SR フリップ・フロップのようにループ長が極端に短い場合には，全ゲートがループの遅延時間ごとに評価されるため，著しく処理効率が悪くなる．この問題は，短いループをもつ部分回路を，記憶をもった一つの機能ブロックで置き換える（図8(b)）ことにより解決することができる．

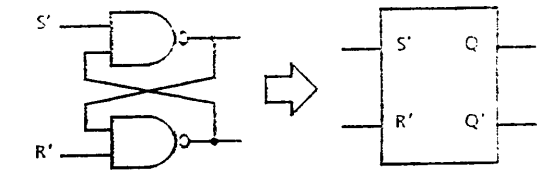

図8(a) 短いループを含む回路
Fig. 8(a) Short loop.

図8(b) 機能ブロック化
Fig. 8(b) Replacement with a function block.

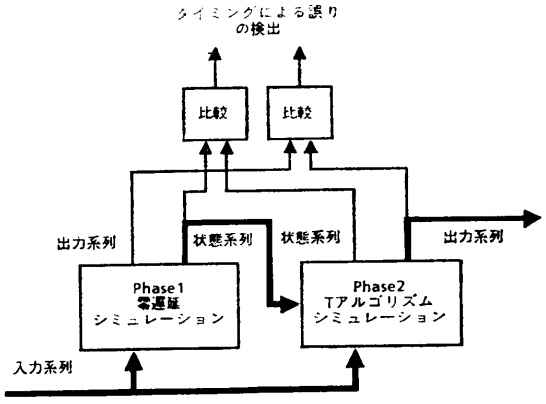

図9 同期式順序回路のシミュレーション
Fig. 9 Simulation of a synchronous sequential circuit.

5.3 同期式順序回路のシミュレーション

同期式順序回路のシミュレーションを行う場合，5.2 節の議論からも明らかなように，シミュレーションの区間長はクロック周期より長くはできない．シミュレーションの効率は，組合せ回路部分の複雑さや，クロック周期により異なってくる．**図9**に示す方法は，組合せ回路部分が単純，あるいはクロック周期が短い場合にも十分な処理効率を保証するものである．すなわち，まず零遅延シミュレーションによって回路の内部記憶のとる信号値（状態値）の系列を求め，次にこれを用いてTアルゴリズムによる遅延を

考慮したシミュレーションを行う．内部記憶のとる信号値系列をあらかじめ計算しておくことにより，Tアルゴリズムのシミュレーションの区間長がクロック周期の任意倍にとれ，4.5 節で述べた処理速度の最大値が得られる．零遅延シミュレーションは遅延を考慮したシミュレーションにくらべて数分の1の時間で処理でき，この方法による同期式順序回路のシミュレーションは，組合せ回路の場合の2倍以下の時間で処理できる．

また，この方法では，零遅延シミュレーションの結果得られた内部記憶，外部出力の信号値系列を，遅延を考慮したシミュレーションの結果と比較することにより，タイミングに関する誤りを自動的にチェックでき，タイミングの検証手法としても有効である．

5.4 入力制約の監視

入力制約とは，入力線に対して許容される信号の値，フリップ・フロップのセットアップ／ホールドタイム，パルスとして認識される最小時間幅のように，回路が正常に動作するために入力信号値が満たすべき制約条件である．われわれは，会話型論理設計・検証支援システム ISS において，回路あるいは部分回路ごとに入力制約を記述し，シミュレーションの実行と同時にこの制約が守られていることを監視することが，設計誤りの早期発見に有効であることを示してきた[6),7)]．入力制約の監視には，単純なシミュレーションに比べ，さらに多くのテーブル参照や入力線の信号値の記憶／参照が必要となる．しかし，入力制約の監視もシミュレーションと同様に「時間優先」で計算することにより，これらのテーブル参照の回数も大幅に減り，計算時間を削減することができる．

5.5 ハードウェア化

3.3節で述べた egset に含まれるゲートはそれぞれ他のゲートとは独立に処理してゆくことができる．しかも，処理の順序制御が本質的にデータフローであるため，高度な並列計算が可能である．

一般に，複数のプロセッサに回路の部分を割り当てて処理を行う場合，あるプロセッサにおける計算結果の一部を他のプロセッサに転送する必要が生じる．転送はプロセッサ間の交換網を介して行われるが，この転送量および転送回数はシステム全体の効率に大きく影響する．従来のハードウェア・シミュレータ[8),9)]では，転送の単位が1イベントであったのに対し，Tアルゴリズムを適用する場合，転送の単位はイベント系列であり，一度に多数のイベントをまとめて転送することができる．このため，転送回数およびそれに伴うオーバヘッドが激減し，高い並列性を生かした効率のよいシミュレーションが期待できる．

6. むすび

時間優先評価アルゴリズムによるゲート・レベル・シミュレーションについて述べた．Tアルゴリズムは，組合せ回路と同期式順序回路を高速にシミュレーションでき，従来のアルゴリズムで用いられてきた種種の信号値／遅延のモデルも効率よく扱える．また，Tアルゴリズムは本質的にデータフロー型の計算方法であるため計算の並列性が高く，専用ハードウェアによるさらに高速な計算への応用も期待できる．この場合，ハードウェアの並列性を効果的に利用したハードウェア・アルゴリズムとして本アルゴリズムを改良することが今後の課題となる．

謝辞 ご討論いただいた本学平石裕実講師はじめ矢島研究室の諸氏に感謝します．なお，本研究は一部文部省科学研究費による．

参考文献

1) 村井真一：ゲート・レベル論理シミュレーション，情報処理，Vol. 22, No. 8, pp. 762-769 (1980).
2) Breuer, M. A. and Friedman, A. D.: *Diagnosis & Reliable Design of Digital Systems*, p. 308, Computer Science Press, Woodland Hills, Calif. (1976).
3) Javor, A.: An Adaptive Time Advance Algorithm for Descrete Simulation, *IPL*, Vol. 6, No. 6, pp. 83-86 (1977).
4) Garey, M. R. and Johnson, D. S.: *Computers and Intractability—A Guide to the Theory of NP-Completeness*, p. 340, W. H. Freeman and Company, San Francisco (1979).
5) 石浦菜岐佐，安浦寛人，矢島脩三：組合せ論理回路の高速タイミング・シミュレーション・アルゴリズムについて，情報処理学会第 28 回全国大会，3P-7, pp. 1443-1444, (1984).
6) 矢島研究室：会話型論理設計・検証支援システム ISS 利用者マニュアル第2版 (1984).
7) 安浦寛人，蚊野 浩，大井 康，木村晋二，石浦菜岐佐，矢島脩三：入力制約監視機能を持つ会話型シミュレーション・システム ISS，情報処理学会論文誌，Vol. 25, No. 2, pp. 285-292 (1984).
8) Denneau, M., Kronstadt, E. and Pfister, G.: Design and Implementation of a Software Simulation Engine, *CAD*, Vol. 15, No. 3, pp. 123-130 (1983).
9) 大森健児，小池誠彦，佐々木徹：超高速論理シミュレータ，設計自動化研究会資料，18-2 (1983).

(昭和59年7月26日受付)
(昭和59年10月18日採録)