新しい素子集合方式に基づく高速ハードウェア論理シミュレータについて

髙 﨑 茂†

本論文は，従来とは異なる新しい素子集合方式に基づいた高速性と大規模収容性に優れたブロックレベルのハードウェアシミュレータを提案している．本方式のブロック演算は，ブロック内素子の回路構成に着目し，素子属性と接続関係を分離して，素子の評価を前段との接続関係を連続的に読み出して処理する方法で，従来のように標準素子を用いることなく，高い並列性と高速性を実現している．シミュレータ全体として見れば，ブロック群の並列処理とブロック内素子群の並列処理を同時に実行できるので，非常に並列性の高いシステムとなる．さらに，シミュレータ全体の処理時間を見たとき，ブロックの演算時間の割合が大きいことから，ブロック内素子間の接続データ量を削減する稠密処理を提案して，接続データ量を大幅に削減し，高速化を図っている．実際の LSI 回路例の評価では，データ量を平均で半分以下にしている．本提案のハードウェア論理シミュレータは最大 64 プロセッサよりなるシステムとして実現されている．性能的には内製ソフトウェアシミュレータの数百倍の高速性を達成するとともに，ハードウェアのゲートレベル方式よりも数倍向上している．本シミュレータは数百万ゲート以上の大規模装置のシミュレーションに適用され，今まで実行が難しかった大量のテストデータによる論理確認を実現して，LSI 設計品質の大幅な向上と開発期間の短縮に大きく寄与することができた．

On the High-speed Hardware Logic Simulator Based on a New Element Dense Set Method

Shigeru Takasaki†

This paper describes a new block-level simulation method. The new method is quite different from the original one adopted in HAL. The first feature of the newly developed block-method is its easiness for implementing parallel processes to evaluate elements (gates) in individual blocks without using conventional standard gates (primitives). As the result of that, the proposed method can easily execute the total block evaluations in a short time. The next feature is to adopt a heuristic algorithm, called an element signal dense set algorithm, to accelerate block computations. This algorithm aims at shortening signal line distances among elements. This can be achieved by new element depth labeling. The proposed method has been implemented in HAL II system and used successfully in simulating large digital systems with more than several million gates.

1. はじめに

LSI の高密度化・高集積化はますます進み，LSI 内に収容される機能は急激に増加してきている．特に，情報技術（IT）に関係する製品は機能を LSI 内に格納しているため，LSI 化された装置ということができる．この LSI 化装置は小型化をはじめ多くの利点を持っているが，一番注意しなければならないのが，LSI の設計品質である．ここに問題があると，LSI の再作成による開発費の増加と製品提供の遅れによる機会損失という大きなリスクをかかえることになるからである．

したがって，設計品質の確保には十分注意していく必要があるが，一方で製品の開発期間も短縮する必要があり，これを満たしながらいかに信頼性の高い製品を提供していくかが大きな課題となっている．これは，特に汎用のサーバ（コンピュータ）やスーパコンピュータのような大規模なシステムを開発する際に顕著になる．これは新規で大規模なシステムを開発すると，新たな問題を混入させてしまう確率が高くなるからである．

このような論理検証の要求に対して，今まで様々なタイプの論理シミュレータが開発されてきた．1 つの方向は，VLSI 設計が高級言語で行われるようになっ

† 理化学研究所
RIKEN (The Institute of Physical and Chemical Research)

たことから，それに対応できるシミュレータであるが，一方で高性能を追求するトランジスタレベルの設計も幅広く行われており，ゲートレベルの詳細さで確認を行いたいという根強い要求もある．今までは多くのものがソフトウェア（SW）により実現されていた．また，近年，FPGA に機能をマッピングして，高速に多量データを実行できるハードウェア（HW）化されたツールも提供されている．しかしながら，方式的には，従来のゲートレベルの方式を元にして，回路テクノロジの進歩（速度向上やメモリ容量拡大）に沿って，提供しているようである．このため対象規模も限られ，マルチシステムのような大規模なものは処理が難しくなるとともに，HW 化されたツールは標準素子（プリミティブ）を採用していることにより，標準素子で直接置換できない回路は複数の標準素子により等価置換されることから対象回路規模が増大することや，論理問題発生時に元の回路（ゲート）との対応関係がとり難くなる問題が起きる．さらに，非常に高価であるという難点がある[2),5),7),9)〜11),13),14)]．

本論文は，従来とは異なる新しい方法に基づいたブロックレベル方式のハードウェア論理シミュレータを提案している．ブロックとはゲートやメモリの集合体として定義されるものである[6)]．本方式のブロック演算は，ブロック内素子の回路構成に着目し，素子属性と接続関係を分離して，素子の評価を前段との接続関係を連続的に読み出して処理する方法で，従来のように標準素子を用いることなく，高い並列性と高速性を実現している．さらに，個々のブロックの演算時間を短縮するために，素子間接続を稠密処理（後述）する方法を提案して，高速化を図っている．ここで提案した方式は論理シミュレータ（最大 64 プロセッサ：HAL II）として実装し，大規模なマルチシステムやスーパコンピュータのシミュレーションに効果的に活用された[8)]．

2. 新しいブロック処理方式

本論文で考察している論理シミュレータは同期式設計における論理（機能）確認用のものである．実際の装置開発においては，この論理確認の占める比率が非常に大きく，かつきわめて重要なためである（論理回路の信号伝播時間（遅延時間）については論理そのものとは別に確認する方法が多くとられている[11)]）．

2.1 従来方式の問題点

2.1.1 ゲートレベル方式の問題点

ゲートレベルの HW シミュレータとして実用化に至ったものはいくつか報告されているが，方式的には元の論理回路を標準素子（たとえば，2 入力 1 出力素子等）に変換して，ゲートレベルでシミュレーションしていくものである[5),7),9),11),13),14)]．実際の回路では，標準素子以外のゲート（たとえば，2 入力以上の多入力を持つゲート等）も多数存在するが，これらは基本的に標準素子に変換（入力数の大きいものは複数の標準素子に分解）されてシミュレーションされるようになる．この方法は HW 実現の容易性等からすると作りやすい方法であるが，この方式で大規模装置（たとえば，数百万ゲート以上のシステム）を扱おうとすると重大な問題が出てくる．それは，標準素子に変換することによりシミュレーション対象回路規模が増大して，それにほぼ比例する形でモデル作成時間も増え，結果的にシミュレーションのための準備時間や実行時間が増大していくことである．また，FPGA を用いたエミュレータも標準素子方式をとっているので同様の規模対応問題があるのに加えて，非常に高価であることや，シミュレータのように内部観測が容易でないこと等があり，さらにこれを用いる場合は，周りの実マシン環境（I/O 装置等）を準備しなければ動かないという問題もある[10),11),13)]．

実際の適用場面で考えると，標準素子の規模増加問題は切実な問題となる．たとえば，元のゲートを標準素子に置き換えた場合，その増加率が 1.5 倍程度に拡大すると仮定すると，装置モデルが 800 万ゲート規模であったとしても，約 1,200 万ゲート相当の収容性を持つシミュレータを準備しておかなければならないことになる．これは費用面の負担とともにシミュレーションの実行可否を左右する重大問題になる．

また最近では，BDD を用いたサイクルベースシミュレータをエンジン化しようとする研究も報告されており，評価が進められていると思われる[12)]．

2.1.2 従来のブロックレベル方式の問題点

従来のゲートレベル処理方式では，ゲート間の転送処理等が多くなり，全体的な処理時間がかかるということから，ゲート群をまとめて扱うブロックレベル方式が提案された[3),4)]．本方式は，ゲート群（組合せ回路）をまとめて処理することで，ゲート演算やゲート間の信号伝播といった処理を効率化することを狙ったものであった．そこで用いられた方法はダイナミックロジックアレイ（DLA）方式と呼ばれた．

詳細は文献 1) に記述されているが，DLA 方式の基本的な考え方は，組合せ論理の真理値表を活用していくものである．すなわち，組合せ論理の真理値表をデコードパターンと呼び，これをメモリに書き込んで，入力値（メモリのアドレスに対応）を入れて，出力値

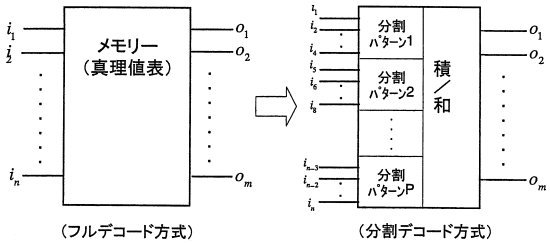

図 1 DLA 方式
Fig. 1 DLA method.

（メモリの読み出し値）を得るのが基本であるが，入力信号数の増大にともなって，ブロックの組合せ論理全体を 1 個の真理値表として表すのはメモリ容量として実質不可能になってくる．そこで DLA 方式は入力信号を分割して，それぞれに対応した分割真理値表（分割デコードパターン）を作り，それを積和の形でまとめることで，容量拡大を抑えつつ論理を実現するものである．この模様を概念的に示したものが**図 1** である．

この方式は，ブロック内のゲート間のオーバヘッドをなくす効果的な方法であったが，一方でこの方式はブロックの入力と出力の真理値関係のみを見ているので，このブロックの回路構成（素子とその接続関係）を活かす工夫はいっさいできなかった．さらにこの方式は，論理規模として，数百ゲートでも数百 K ビットのメモリを必要とする場合があり，LSI 等より生成される規模の大きい数千ゲート位の論理ブロックを収容することはきわめて難しくなってくるということが分かってきた．

DLA 方式で 1 つの論理ブロック（組合せ回路の集合）を表すのに必要なメモリは次のように示されている[1)]．

$$2^m \times \left(\left[\frac{M}{m} \right] \times P + \left[\frac{P}{m} \right] \times N \right) \qquad (1)$$

ここで，

M：入力信号線，N：出力信号線，P：積項数，

m：束ねる数，$[M/m]$：M/m を切り上げた整数

である．

たとえば，次のような属性（$M = 80$，$N = 80$，$P = 30$，$m = 16$）を考えてみると，必要メモリは約 20,316K ビットとなる．したがって，本方式はブロックの規模が大きくなったとき，多数の LSI より作られる総ブロックを収容するのが難しくなる（モデルの収容容量が膨大になる）とともに，メモリアクセスの関係からブロックの処理時間（性能）も顕著にかかってしまうという問題があった．

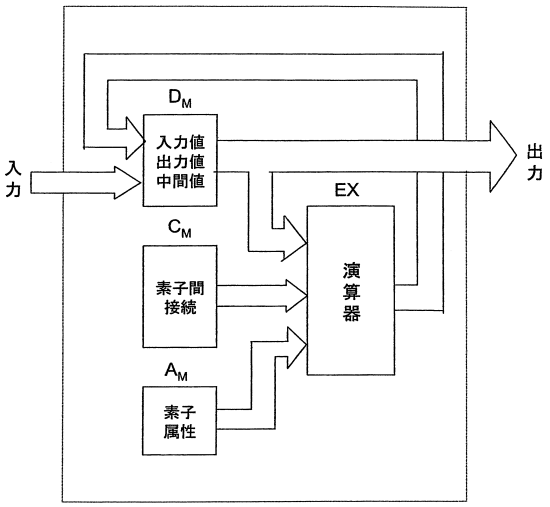

図 2 新方式のブロック図
Fig. 2 The new method block diagram.

2.2 新しいブロック処理方式

ブロックレベル方式は，基本的にゲートの集合体（たとえば，数千ゲート）を 1 つの論理的な固まりとして処理する方式である．したがって，この方式の良い点は，ブロックとして一度作成しておくと，装置全体をシミュレーションする際，ブロック内素子の接続関係を展開することなく，ブロック間の接続のみでシミュレーションすることができることである（後述 4.1 節参照）．本論文はこのブロックのシミュレーションを効果的に行う新しい方式を提案している．この方式は，ブロック内の回路構成を活かし，簡潔にシミュレーションすることを狙ったものである．具体的には，ブロックの回路構成（素子（ゲート）の属性（AND，OR，EXOR 等）やそれらの間の接続関係）の特徴を活かし，素子属性と接続を分離し，前段との接続関係を連続的に読み出しながら，演算機構を用いてブロック機能を実現しようとするものである．この方式により実現されるブロックの機能構成を**図 2** に示す．

本方式のシミュレーション実現法は，非常に簡潔な構成であるが，今までこのような形で報告されているものは見当たらない[2)~7),9)~14)]．本方式の特徴としては，ブロックの規模拡大に対して容易に対応できるとともに，高速にブロックの演算を実現できることであるが，この実行法ついては後のシミュレーション動作で説明されている．DLA 方式は，ブロックの論理構成や規模増大にともなって必要メモリが急激（たとえば，指数関数的）に拡大していくが，新しい方式では規模増加に対してほぼ線形に近い形で増大していく

（ブロック規模数千ゲートクラスの実際例）．図 2 に示される機能構成図の各部分の機能およびその動作は以下のようである．

【機能構成図内各部の機能】

C_M： 素子（ゲート）間の接続情報を格納しているメモリである．

A_M： 素子（ゲート）の属性（AND，OR，EXOR，出力の正/否等）が入るメモリである．

D_M： ブロック内素子演算の途中状態および最終結果が格納されるメモリである．

E_X： C_M，A_M，D_M を入力して，ブロック内素子の演算（シミュレーション）を実行する部分である．

新方式に基づくブロックシミュレーションの動作は次のようである．

【新方式によるブロックのシミュレーション動作】

S-1：（1-1，1-2，1-3 は同時動作）

1-1： データメモリ D_M よりブロックの入力値を読み出す．

1-2： 接続メモリ C_M より前段との接続関係を読み出す．

1-3： 素子属性メモリ A_M より素子の属性（AND，OR，EXOR 等）を読み出す．

S-2： 上記 1-1〜1-3 の値をもとに，素子の演算（シミュレーション）を行う．

S-3： S-2 の結果はデータメモリ D_M に格納され，該素子から接続されている素子をシミュレーションするときに読み出される．これらの動作が繰り返し行われる．

この方式は，ブロックのシミュレーションを効果的に実現するのに適した方法である．しかしながら，この方法で装置全体をシミュレーションすることは効果的な方法とは考えられない．それは，この方式で素子を装置全体まで拡大すると，各素子の前段からの接続関係が多岐にわたり，結果的に遠く離れた素子との接続を表す接続データ量が増えて，これに必要なメモリ容量が増大し，処理のオーバヘッドが増えていくためである．したがって，この方式は，ブロック（LSI 内部のゲート集合体で規模目安としては数千ゲート）を高速に処理するのに適した方法である．

【新しいブロック処理方式と従来方式の相違点】

- DLA 方式との相違点は以下のようである．
  - ブロックの機能実現法（演算法）が明確に異なる．
  - ブロックサイズの拡充：基本的に前段との接続関係を格納する方式であるから，ブロックの入力信号線や内部の論理段数の制約を受けず，任意サイズのブロックを作ることができる．
  - ブロックのデータ量予測が容易：DLA 方式はブロックの論理構成により必要なメモリ量が急激に変動（たとえば，指数関数的）し，シミュレーションすべき装置全体のブロックを格納するのに必要なデータ量の見積りは難しかったが，新方式ではブロックを格納するのに必要なデータ量はほぼ素子の接続関係に準ずる形で見積もることができる．
  - 詳細さの保証と素子入力数の制約解除：ブロック内は元の素子（ゲート）からそのまま作られているので，ゲートレベルでの詳細確認が可能である．また，従来のような標準素子を用いる方式を採用していないため，素子入力数に対する制約はない．
- 従来報告されている HW シミュレータやエミュレータに対する方式上の相違点は次のようである[2),5),7),9),10),13),14)]．
  - 本提案のブロック処理方式は，シミュレーションすべきモデル全体をブロックに分割して，ブロック処理を基本とする方式である（後述 4.1 節参照）．したがって，従来のようにモデル全体を標準素子に変換してシミュレーションする方式とは異なる．
  - 本提案方式における素子の評価は，素子の属性と接続を明確に分離して，前段との接続関係を連続的に読み出しながら，専用の演算機構（HW）で素子機能を実現する方法であるが，このような柔軟な構成（図 2，図 3）をとっているものは報告されていない．従来のものは入力数に変動のない標準素子を用いているので，素子の評価は入力値に対する出力値を返すテーブル方式をとっている[5),7),9),10),14)]．
  - 本方式は，素子の入力数に対する制約はない．従来のゲートレベル方式は標準素子を用いているため，入力数の変動に柔軟に対応することは難しかった．従来は，実ゲートの入力数が標準素子より少ない場合はクランプ処理等で，また実ゲートの入力数が標準素子より多い場合は複数の標準素子を用いてモデル化する必要があり，モデルを生成する際に手間を要することや，対象回路規模の増大を招くことに加えて，元の回路との対応づけが難しくなる問題があった[5),7),9),10),14)]．これに対して本方式は，該当素子への入力があるかどうかを，前段との接続関係があるかどうか（読み

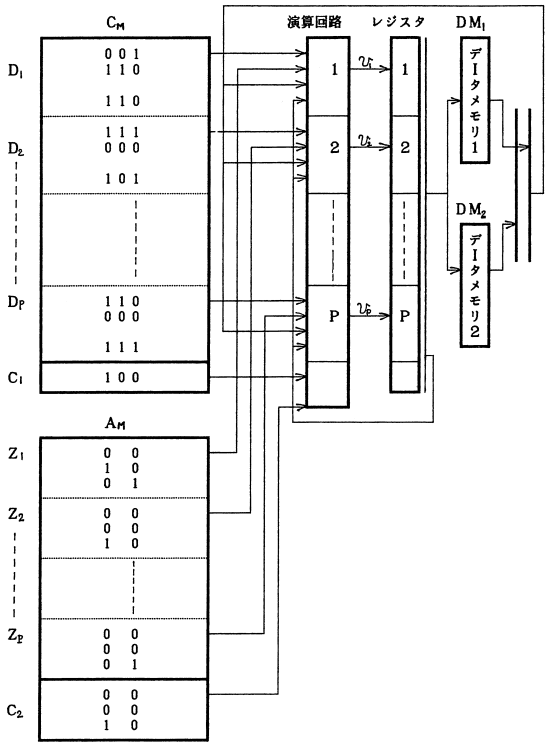

図 3 P 個の素子を並列に実行するブロック図
Fig. 3 The block diagram with P element evaluations.

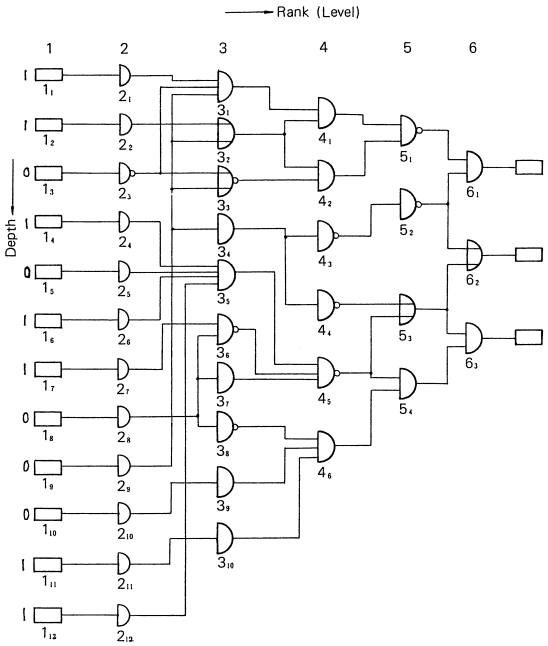

図 4 レベル（ランク）と深さが設定されたブロック回路図
Fig. 4 Original block circuit with level (rank) and element.

出し終了フラグの有無）で判断しているので，入力数に制限を設けずに元の回路に対応した形で実行できる（後述 2.3.1 項および 4.2 節参照，素子あたり最大 256 千入力数まで処理可能）．また，本方式の素子間接続関係は，前段素子との接続有無という非常に簡単な形で表しているため，連続的な読み出しが可能で，高速化に適した特徴を備えている．

新方式は，同一レベル（詳細は後述）の素子を並列に実行（たとえば，P 個同時実行）することも容易に実現できる．このブロック図を**図 3** に示す（実装された具体的構成は後述 4.2 節参照）．

【ブロック図の説明】

C_M： 素子の前段との接続有無（1：有，0：無）が入る．P 個の素子を並列実行（D_1〜D_P）．C_1：読み出し終了．

A_M： 素子属性（AND：“00”，OR：“01”，EXOR：“10”，出力正否（1：正，0：否））が入る．P 個の素子を並列実行（Z_1〜Z_P）．C_2 は制御データ．

演算回路： C_M，A_M，$DM_{1,2}$ を入力して演算（シミュレーション）が実行される．P 個の素子が並列に実行される．この部分は専用 HW で実現できる．

レジスタ： 演算した結果を保持する．前段との接続関係が完結しないものは，フィードバック信号線を通して再度演算される．

$DM_{1,2}$： 各レベルのシミュレーション結果を格納するメモリ．DM_1 と DM_2 は交互にデータが入れ替わって格納される．これは，前段からの読み出しデータ（前レベルでの結果）と現レベルでの結果の格納を交互に行うためである．

C_M，A_M，DM は素子単位に独立しているので，同時読み出しが可能であるとともに，並列化しても他の素子データと競合するようなことはない．本例の場合，P 個の素子が並列に実行されても競合するところはない．また，ブロック全体は各々のプロセッサに割り付けるので，ブロック間の競合も無いようにしている．

2.3 新しいブロック処理方式の課題

2.3.1 新しいブロック処理方式における回路処理例

本提案のシミュレータでは，LSI は組合せ回路のブロックに分割されるが，このブロックの一回路例を示したものが，**図 4** である．

ブロック内は基本素子（ゲート）から構成されるものとし，シミュレーション方法としては，ブロック内の各素子に対して，入力側から出力側に向かってレベル付けがなされ，レベル単位に処理が行われる．

本提案のシミュレーション方式は，ブロック内の回

路構成と密接に関係しているため，諸々の記号やデータについて以下に説明する．

【ブロック内の回路についての準備】

- 図 4 において，左端がブロックの入力端子側で，右端がブロックの出力端子側となる．
- シミュレーションの準備として，まず入力端子の一番上から下に向かって深さ（depth）をつける．さらに，ブロックの入力端子から出力端子に向かってレベル（ランク：rank と同意）をつける．図 4 の各素子に対してこれらを施すと，最大の深さは 12 となり，レベルは 6 となる．出力端子そのものにはレベルを付加しない．素子に付加された記号 r_i において，r はレベル順位，i は素子の深さを表す．

新しいブロック処理方式では，C_M と A_M のデータ格納構造が処理効率や高速化に重要となる．このため，ブロック内の各レベルの素子が 4 個並列に実行される場合を例にして C_M や A_M への格納方式を説明する．

接続メモリ C_M は，4 素子単位に前段の素子との接続有無（1：有，0：無）を見ていく．たとえば，レベル 2 の素子 2_1 は前段の 4 素子 1_1〜1_4 を考えたとき，1_1 のみと接続があり，他とはない．したがって，“1000” が C_M の対応するところに格納される．同様に素子 2_2 は “0100” となる．レベル 3 の素子 3_1 は前段の素子 2_1，2_3，および 2_9 より接続されている．したがって，“101000001000” となる．同様に 3_2 は “010000001000” となり，4 ビット単位に C_M に格納される．前段との接続が継続するかどうかは読み出し完了信号（1：完了，0：継続）で分かるようにしておく．この模様を**図 5** に示す．

次に素子属性メモリ A_M は，シミュレーションすべき素子がどういう属性（たとえば，AND，OR，EXOR およびその出力の正否）であるかを示すとともに，ブロック内シミュレーションの制御用データが格納される．これは前述の図 3 各ブロックで説明された形で入る．したがって，素子 2_1 は AND であるので，“000”，2_3 は NAND であるので “001”，3_2 は OR であるので，“010”，3_3 は NOR であるので，“011” となる．多種類の素子もビットを拡張すれば対応可能である．制御信号としては，ブロック内のレベル処理の終了（“1”）や，ブロック全体の終了（“1”）を示すとともに，前記 C_M で前段との接続がない場合には，スキップして無駄な読み出しをしないようにすることもできる．これはブロックの回路構成を見て，その特徴を活かして処理を効率化したものである．これらの模様を

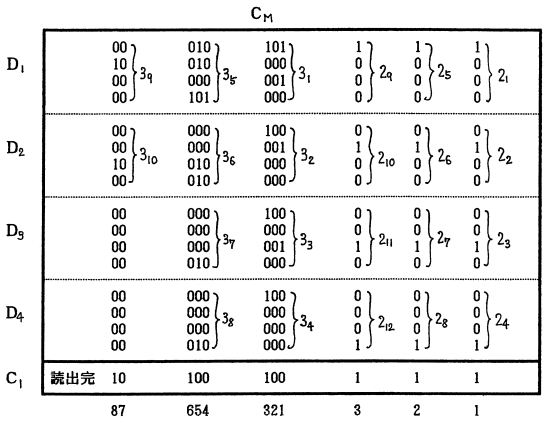

図 5 素子接続データの格納イメージ
Fig. 5 Element connection data configuration.

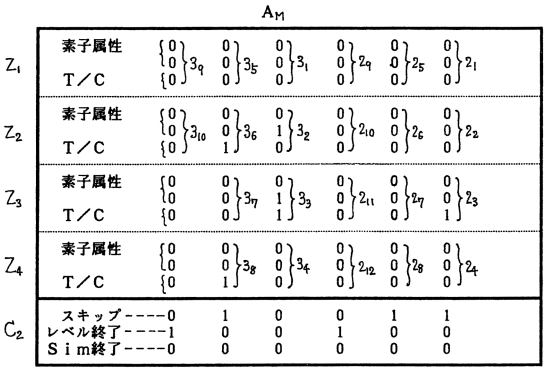

図 6 素子属性データ格納イメージ
Fig. 6 Element attribute data configuration.

図 6 に示す．

図 4 に示されるブロックが，実際にどのようにシミュレーションされるかについて，図 3，図 5，図 6 を用いて示す．基本的には，前述の【新方式によるシミュレーション動作】の手順に沿って行われる．まず，実行に先立ち，ブロックの入力端子 1_1〜1_{12} は，入力データ “110101100011” が設定されたものとし，これらは図 3 の DM_1 に格納されているものとする．シミュレーションは 4 素子ごとに行われ，最終レベルの素子が完了すると終了する．簡単化のため，レベル 2 と 3 の実行過程を具体的に説明する．

【レベル 2 のシミュレーション過程】

S-1： 最初の 4 素子 2_1〜2_4 のデータ（DM_1，C_M，A_M）が読み出される．

S-2： これらは，演算回路で実行され，結果 “1111” はレジスタに入る．

S-3： 図 5 の C_M より，前記 4 素子の前段からの接続が完了していることが分かるので，レジスタの値は DM_2 に入る．

同様に 2_5〜2_8 が行われ，結果は DM_2 に入る．最後に 2_9〜2_{12} が行われるが，ここでレベル 2 は終了（A_M のデータより）するので，この結果を DM_2 に入れる．

レベル 3 も基本的にはレベル 2 と同様に行われる．

【レベル 3 のシミュレーション過程】

S-1： 最初の 4 素子 3_1〜3_4 のデータが DM_2，C_M，A_M より読み出される．

S-2： 1 回目の演算実行が行われ，その結果はレジスタに入る．ただ，本 4 素子は前段との接続が完了していないので，再度 S-1 の読み出しが行われる．

S-1： 次の 4 素子分 2_5〜2_8 からの接続分が読まれる．この段階でも前段との接続は未完了である．

S-2： 2 回目の演算実行が行われるが，このときはレジスタに入っている前回の結果も加味して演算される．これは 1 列目の結果を次列に伝えることを意味している．

同様に 3 列目も行われるが，ここで前段からの接続は完了（C_M の C_1：1）するので，3_1〜3_4 の演算結果 “0100” は DM_1 に格納される（DM_1 と DM_2 はレベルごとに交換して使用される）．同様に次の 4 素子 3_5〜3_8，さらに次素子 3_9，3_{10} が行われると，レベルの終了となり，レベル 3 素子のシミュレーションは完了する．このような形で最終レベル 6 まで進むと，ブロック全体が終了（A_M の Sim 終了：1）することが分かるので，ブロック全体のシミュレーションが終了する．このブロックの出力結果は，DM_2 に格納される．

2.3.2 新しいブロック処理方式における性能予測と改善点

新しいブロックレベル方式の処理時間 B_P は各レベルの処理時間とレベル数により予測することができる．まず，ブロック内の i レベル処理時間 L_{Pi} は次のように求めることができる．

$$L_{pi} = \frac{\max\{\mathrm{r}\times \mathrm{C_M}, \mathrm{A_M}, \mathrm{D_{Mr}}\} + \mathrm{r}\times \mathrm{E} + \mathrm{D_{Ms}}}{m} \tag{2}$$

ここで，

C_M：接続データアクセス時間，

A_M：素子属性データアクセス時間，

D_{Mr}：データメモリアクセス時間，

E：演算実行およびレジスタ設定時間，

D_{MS}：データメモリ設定時間，

r：前段素子に対する繰返し読み出し数，

m：並列実行素子数．

したがって，ブロックの処理時間 B_P は次のようになる．

$$B_p = \sum_{i=1}^{R} L_{pi} \tag{3}$$

ここで，

R：レベル（ランク）数．

式 (2)，(3) より推測できるように，ブロックの処理時間を短縮するには，メモリアクセス時間の短縮，特に繰返し読み出しの多い接続データを短縮することが一番大きな影響を持ってくる．たとえば，図 5 の回路事例において，レベル 3 の素子すべてを実行するのに，A_M や D_{Mr} は 3 列を読み出せば済むが，C_M は合計 8 列（3＋3＋2）読み出す必要がある．したがって，この読み出し量を論理を変えることなく減らすことができれば，ブロック処理の性能向上を図ることができる．接続データは前レベルの素子の深さが深いほど遠くからの接続が多くなり，データ量が増える傾向にあるので，この繰返しをいかに減らせるかがさらなる性能向上への鍵となる．

3. 稠密処理方式に基づくブロック処理方式について

ここでは，前記の課題を克服する新たな提案（素子間接続の稠密処理）を行い，その効果について述べる．

3.1 素子間接続の稠密処理

素子間の接続関係を密にする処理（稠密処理と呼称）を施して，ブロック全体の論理を変えることなく，前段との接続データ量を削減し，さらに高速化する方法を検討した．

【素子間接続の稠密処理】

S-1： 最小レベル（ランク）で，最小深さの素子を選び，ファンアウトトレース（以降，FO トレースと略す）の元素子とする．

S-2： 該素子より FO トレースを実行し前記素子に接続している素子を探し，該ランクの上から下に向かって深さの新ラベルをつけていく．ただし，FO トレースすべき素子が最大レベルに達し，全素子の処理が完了して，すべてに新ラベルがふり直されたら，稠密処理は完了する．

S-3： 前記で得られた各素子のファンイン元を調べ，前記ファンアウト信号線以外の入力があればファンイントレース（以降 FI トレースと略す）の素子に設定して S-4 へ行く．そのほかは，それらを FO トレース素子へ設定し，S-2 へ行く．

表 1 各種稠密処理手法によるデータ削減効果
Table 1 Various signal dense set algorithm effectiveness.

	block 1	block 2	block 3	block 4	block 5	block 6	block 7	block 8
稠密無	1	1	1	1	1	1	1	1
本手法	0.54	0.67	0.45	0.49	0.4	0.61	0.5	0.65
FI手法	0.69	0.9	0.8	0.75	0.52	1.09	0.6	1.23
FO手法	0.55	0.65	0.49	0.54	0.4	0.77	0.5	0.86

S-4： FI トレース素子より FI トレースを実施し，現在のレベルより前レベルの素子を決定する．次にその素子に新規の深さを設定する．次にその素子を FI トレース素子に設定し，同様のプロセスを実行する．この処理は最小レベルの素子まで繰り返される．最小レベルの新規の深さが設定（ラベルづけ）されたら，該素子から接続されているファンアウト側の素子で，ラベルが付加されていない素子を探す．もしそのような素子があったなら，新規のレベルを付け，その素子を FO トレースの素子へ設定し，S-2 へ行く．

このヒューリステックな信号線間の稠密処理は，素子間の信号線の距離を短くすることを狙ったもので，これを上記のように素子の深さのレベルをふり直すことにより実現している．今までゲートのグルーピング等を行っている方法も報告されているが，本提案の稠密処理とは異なるものである[15]．稠密処理の方法としては，最小レベルから FO トレースのみを行って素子を集める方法や，最大レベルから FI トレースのみを行う方法等を検討した．**表 1** は各手法を使ったときにデータ量の削減がどのように行われたかを示したものである．削減効果は回路構成により変動するが，本提案の手法が一番効果的だったのでこの方法を採用した．

本提案の稠密処理を図 4 のブロック内回路に適用した結果を**図 7** に示す．図 7 の素子に付けられた r_{i-j} において，r はレベル数，i は旧の深さ，j は新の深さを示している．図 7 の回路は，ブロックの論理を変えることなく，稠密処理によって素子を入れ替え，素子間の信号線距離を短くしたものである．本処理を施した後の回路に対して，素子接続データを作り直したものを**図 8** に示す．

図 5 と図 8 の C_M を比べてみると，元の回路では，素子 3_1〜3_{10} を処理するのに合計 8 列読み出しが必要だったものが，稠密処理を施すことによって，3_1〜3_{10} は 4 列に削減されていることが分かる．これは素子の接続関係を格納するメモリ量を半減させるとともに，レベル 3 の素子を実行するアクセス回数が半減させたことを意味している．したがって，ブロック処理の高速化に大きく寄与するものである．

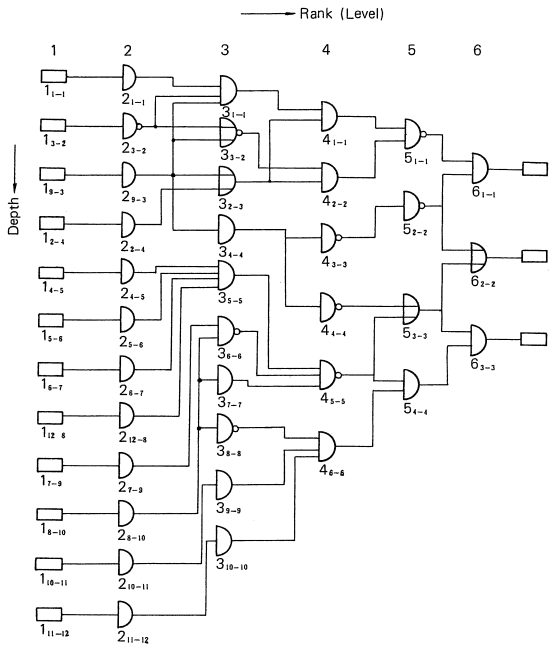

図 7 稠密処理後のブロック回路図
Fig. 7 Modified block circuit after element signal dense set algorithm execution.

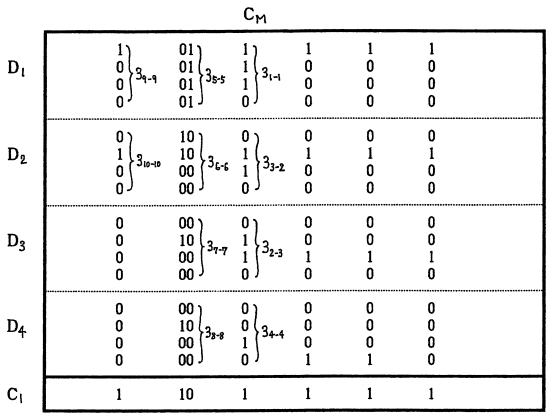

図 8 素子接続データ
Fig. 8 Element connection data configuration.

3.2 実回路による削減効果

本稠密処理を実際の演算処理系の LSI より作られたブロックに適用し，元のブロックとのデータ量を対比したものを**表 2** に示す．

表 2 より分かるように，1 千〜4 千ゲート規模のブロックに対して，稠密処理を行うことによって，素子間の接続データ量（メモリ量）が平均で半分以下になっていることが分かる．本データのアクセス時間がブロック処理時間に占める比率が高いので，本処理により全体性能が大きく向上することが見積もれる．

表 2 稠密処理の実回路例
Table 2 Practical circuit examples after element signal dense set algorithm execution.

	入力数	出力数	最大ランク	最大深さ	ゲート規模	稠密無	*稠密有
1	62	69	15	165	1676	1	0.56
2	73	28	17	233	2320	1	0.49
3	54	69	11	163	1186	1	0.32
4	93	44	11	311	1954	1	0.32
5	60	67	12	205	1573	1	0.32
6	85	36	14	353	2218	1	0.54
7	92	45	16	245	2214	1	0.39
8	81	36	14	261	2242	1	0.52
9	35	61	11	111	952	1	0.35
10	108	43	11	300	1880	1	0.46
11	23	27	6	82	337	1	0.48
12	124	32	7	316	1397	1	0.62
13	123	30	12	282	2006	1	0.58
14	127	31	15	380	2870	1	0.45
15	63	58	20	354	4319	1	0.44
16	70	13	19	287	1728	1	0.59
平均							0.46

*:稠密処理無(元の状態)を1とした時の比率

4. 新ブロック処理方式の HW シミュレータ実装について

4.1 新しいブロック処理方式に基づくシミュレーション全体処理

ブロック方式が大規模装置のシミュレーションとして実行されるまでの過程を示すと**図 9** のようになる．

シミュレーションすべき元の装置（たとえば，コンピュータ）は主に LSI より構成される LSI 集合体と見なすことができる．次に，これらの LSI は組合せ回路よりなるブロックに分割される．したがって，この時点で，元の被シミュレーションモデルはブロックの集合体となる．さらに，これらのブロックは入力端子から出力端子に向かってレベル（ランクとも呼ぶ）づけが行われる．この模様は図 9 の一番下に示されている．シミュレーション全体としては，レベル 1 からレベル n に向かって行われるレベル処理方式であるとともに，ブロックのイベントがある場合のみ実行されるイベント方式である．遅延は零である．したがって，同一レベルのブロック群は同時に実行できる．シミュレーション論理値としては，高速性を重視して 2 値 (0,1) としている．

実回路では，メモリ素子も含まれるが，これはメモリブロックとして，F/F と同一レベルに位置づけられてシミュレーションされる．また，ブロック間で信号線が戻る帰還回路も想定されるが，このような場合は，現在実行されているレベルを帰還先に戻して再度実行することで処理することができる．

以上の全体処理から推測できるように，シミュレーション全体の高速化を図るためには，並列に実行できるブロック数を増やすとともに，ブロック処理自体を高速にしていくことが重要となる．

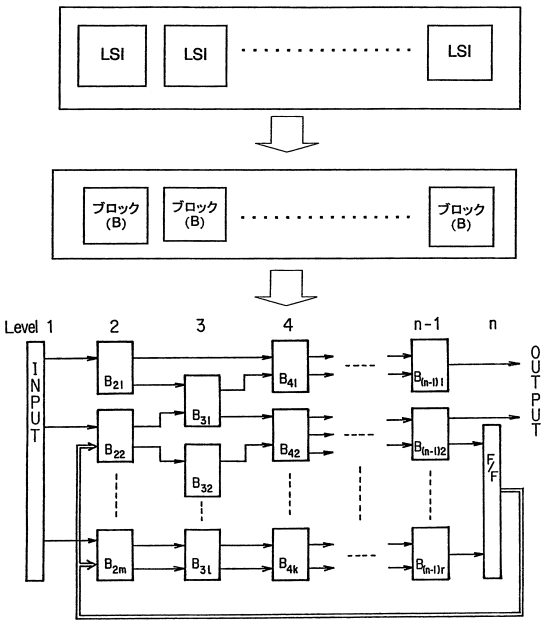

図 9 ブロックレベルシミュレーション全体の流れ
Fig. 9 Block-level simulation flow.

表 3 HW シミュレーターのシステム容量
Table 3 HW simulator system capacity.

	論理	メモリー
収容規模	1,160 万ゲート	8M バイト
ブロック数	59,392	4,096
ブロック種類数	7,424	512

4.2 新ブロック処理方式のハードウェア実装について

本論文で提案したブロック処理方式として，1 プロセッサにおけるブロック内の素子演算が 8 並列実行できるものを HW シミュレータ（64 プロセッサシステム）に組み込んで実装した[8]．演算器の構成としては，DM が 8 bit 幅 × 1 kw × 2 個，C_M が 8 bit 幅 × 256 kw × 8 並列，A_M が 3 bit 幅 × 128 kw × 8 並列，演算回路 8 個，レジスタ 8 bit となっている．HW シミュレータの収容性等は**表 3** に示すような形になる．

従来の汎用コンピュータ（30 MIPS マシン相当）上で走行する内製の RTL レベル SW シミュレータでは，実行できないシミュレーションモデルも多いので，一概に性能を比較することは難しいが，可能な領域で SW シミュレータに対して，ゲートレベル，ブロックレベル（稠密有無）の相対性能を示したものが**表 4** である．モデルとして，それぞれ約 26 万ゲート（上段），

表 4 様々なシミュレーション方式での相対性能比較
Table 4 Relative performance comparison in various simulation methods.

HW シミュレータ			SW シミュレータ（汎用コンピュータ）
ゲート方式	ブロック方式		
	稠密無	稠密有	
$\frac{1}{86}$	$\frac{1}{369}$	$\frac{1}{589}$	1
$\frac{1}{72}$	$\frac{1}{348}$	$\frac{1}{512}$	1
$\frac{1}{68}$	$\frac{1}{305}$	$\frac{1}{443}$	1

約 120 万ゲート（中段）および約 135 万ゲート（下段）の各装置を 100 万クロック走行させて，クロックあたりの処理時間を相対比較したものである．ゲート方式とブロック方式は同一の HW シミュレータ（32 プロセッサ）で実行している．ゲート方式は，ブロック内を前記のゲート集合体ではなく単純にゲートのみでモデル化して実行しているため，取扱い回路の規模制約はあるが，相対的な性能比は判断することができる．

高速化の観点からは，HW 化した利点が出ている．さらに HW 方式で見るとブロック方式が数倍優れていることが分かる．これは，ブロック方式の方がプロッサー間をわたる信号伝播のオーバヘッドが少ないとともに，ブロック内処理が並列化等で高速に実行ができるためと考えられる．シミュレーション実行前に行うモデルの生成時間もゲート方式に比べブロック方式の方が約 7〜10 倍高速化されていることが分かった．これはブロック方式がブロック間の接続処理でモデル生成できるのに対して，ゲート方式は毎回ゲート間の接続を作る必要があることによっている．どのぐらいの素子を同時並列に実行できるかの並列化指標としては，本モデル実行例の場合，30 プロセッサが論理ブロックを処理しているので，同一レベルのものは同時実行可能（図 9 参照）となり，30 ブロックが並列に実行できることになる．さらに個々のプロセッサでは 8 素子が並列に実行できるので，最大 240（30×8）素子が並列に実行できることになる（最大性能：2.4 Ggates/sec）．モデルを構成しているブロック群は並列実行できるように個々のプロセッサに割り付けるので，ブロック間で競合することはない．また稠密処理により，さらなる高速化（約 5 割前後の改善）を図れることが分かる．

表 4 の下段の回路を PC（Pentium4 2.8 GHz, 512 MB，Linux 8.0 実装の市販機種）上で市販のゲートレベルシミュレータ（VCS：シノプシス社）で実行したものを比べると，本提案のブロック方式は約 8 倍程度高速であるが，これは実現されている HW 実装形態の差異による要素が大きく，もし本方式が最新の実装技術で構築されたとすると数百倍の高速性を実現できることが予測される．

並列処理の場合，各処理における粒度（本例の場合ブロックの規模（ゲート量））をどの程度にしておくかは，大事な点であるが，本例ではシミュレーション全体の高速性とブロック種の版数管理を考慮して，約 5 千ゲートを上限の目安にした．性能的にはもう少し小規模なものの方が高速化しやすいと思われたが，管理面の煩雑さや全体オーバヘッドより上記の値を目安にした．

5. むすび

本論文は，新しいブロック方式のシミュレーション方法およびさらにこれを改善して高速化した稠密処理方式について述べた．本 HW シミュレータは最大 64 台のプロセッサよりなるシステムとして実装し，数百万ゲート規模の大規模装置シミュレーションに使われ，今まで実行が難しかった大量のテストデータによる論理確認を可能にした．その結果，新規装置の設計品質を大幅に改善するとともに，開発期間の短縮と品質向上に大きく寄与することができた．

謝辞 日頃ご指導いただく法政大学大森健児教授，小池誠彦教授，日本電気（株）野水宣良氏，石倉浩氏，北陸日本電気（株）大窪一郎氏，東京電機大学山田昭彦教授，元日本電気（株）佐々木徹氏に深謝いたします．

参考文献

1) 大森健児，小池誠彦，佐々木徹：論理の動的変更が可能なダイナミックロジックアレイについて，信学会研究会資料 EC82-12，pp.13–22 (Sep. 1982).
2) Pfister, G.F.: The Yorktown Simulation Engine: Introduction, *Proc. 19th DA Conf.*, pp.51–54 (1982).
3) Koike, N., Ohmori, K., Kondo, H. and Sasaki, T.: A High Speed Logic Simulation Machine, *Compcon83 Spring*, pp.446–451 (1983).
4) Sasaki,T., Koike, N., Ohmori, K. and Tomita, K.: HAL; Block Level Hardware Logic Simulation, *Proc. 20th DA Conf.*, pp.150–156 (June 1983).
5) Blank, T.: A Survey of Hardware Accelerators Used in Computer Aided Design, *IEEE Design & Test*, Vol.1, No.4, pp.21–39 (Aug. 1984).
6) Takasaki, S., Sasaki, T., Nomizu, N., Koike, N. and Ohmori, K.: Block-Level Hardware Logic Simulation Machine, *IEEE Trans. CAD*, Vol.CAD-6, No.1, pp.46–54 (Jan. 1987).

7) Hirose, F., Ishii, M., Niitsuma, J., Shindo, T., Kawato, N., Hamamura, H., Uchida, K. and Yamada, H.: Simulation Prosessor SP, *ICCAD-87*, pp.484–487 (Nov. 1987).
8) Takasaki, S., Hirose, F. and Yamada, A.: Logic Simulation Engines in Japan, *IEEE Design & Test*, Vol.6, No.5, pp.40–49 (Oct. 1989).
9) Beece, B., et al.: IBM enjineering verification engine, *Proc. 25th DA Conf.*, pp.218–224 (1988).
10) Babb, J., Tessier, R., et al.: Logic Emulation With Virtual Wires, *IEEE Trans. CAD*, Vol.16. No.6, pp.609–626 (June 1997).
11) Darringer, J., et al.: EDA in IBM: Past, Present, and Future, *IEEE Trans. CAD*, Vol.19, No.12, pp.1476–1497 (Dec. 2000).
12) 伊勢野総，井口幸洋，笹尾　勤，松浦宗寛：決定図に基づくサイクルベース・シミュレーションエンジン，信学技法 VLD98-138 (Mar. 1999).
13) http://www.cadence.co.jp/products/
14) Hamamura, H. and Komatsu, H.: SP2: A Very Large-Scale Event Driven Logic Simulation Hardware, *IEICE Trans. Fundamentals*, Vol.E85-A, No.12, pp.2737–2745 (Dec. 2002).
15) 石浦菜岐佐，安浦寛人，矢島脩三：ベクトル計算機による高速論理シミュレーション，情報処理学会論文誌，Vol.27, No.5, pp.510–517 (1986).

(平成 16 年 3 月 15 日受付)
(平成 17 年 1 月 7 日採録)

髙﨑　茂（正会員）

1975 年工学院大学工学部電子工学科卒業，1977 年同大学院修士課程修了，同年日本電気（株）入社．以来論理回路の試験容易化設計，テスト設計，シミュレーション，CAD 専用エンジン，CAD システム，IT システムの研究開発に従事．最近では，分子生物学，ゲノム情報科学等の研究に興味を持つ．工学博士．現在，理化学研究所ゲノム総合科学研究センターに勤務．生化学学会，IEEE 各会員．