設 計 自 動 化 85－13
（1997. 10. 29）

連想メモリを搭載したハードウェアエンジンによる故障回路並列故障シミュレーションの高速化手法

福山 誠一郎　戸川 望　佐藤 政生　大附 辰夫

早稲田大学理工学部電子・情報通信学科
〒169 東京都新宿区大久保 3-4-1
E-mail: seiichi@sato.comm.waseda.ac.jp

あらまし

連想メモリ (CAM: Content Addressable Memory) を搭載したハードウェアエンジンを用いて故障回路並列の故障シミュレーションを実行することにより，シミュレーション時間が逐次型計算機を利用した場合に比べて短縮されることが知られている．これは，連想メモリの一致検索機能や並列書き込み機能等を利用することで，全故障回路を連想メモリ上で並列にシミュレーションできるためである．しかしながら，対象とする回路が大規模化すると，全故障回路を連想メモリ上に同時に記憶できないことがある．従来，全故障回路を連想メモリ上に一度に記憶可能な数毎に分割して処理していたが，その際に必要となる連想メモリとホストコンピュータとの通信がボトルネックとなり，シミュレーション速度向上の妨げとなっていた．本稿では，連想メモリを搭載したハードウェアエンジン上に通信用 RAM を設けることで，ホストコンピュータとの通信による遅延を削減する手法を提案し計算機による提案手法の評価結果を報告する．

キーワード　連想メモリ，単一論理縮退故障，並列故障シミュレーション，ホストコンピュータ，通信時間

A Fast Parallel-Fault Simulation Algorithm Using a CAM-Based Hardware Engine

Seiichiro FUKUYAMA　Nozomu TOGAWA　Masao SATO　Tatsuo OHTSUKI

Dept. of Electronics, Information and Communication Engineering
Waseda University
3-4-1 Okubo, Shinjuku, Tokyo 169, Japan
E-mail: seiichi@sato.comm.waseda.ac.jp

Abstract

CAM (Content Addressable Memory) can operate word-parallel equivalence search and word-paralle writing. Fault simulation time can be reduced by using a CAM-based hardware engine compared with using a serial computer, since fault circuits can be simulated in parallel on CAM. However, if the size of circuits is larger, we cannot simulate all fault circuits because of limited CAM capacity. In such a case, we can divide the parallel-fault simulation. This simulation needs communication between the hardware engine and host computer, which is a bottleneck of the parallel-fault simulation and decreases fault simulation speed. In this paper, we propose a fast parallel-fault simulation algorithm using a CAM-based hardware engine which has communication RAM. Communication RAM reduces a delay caused by communication between the hardware engine and host computer. Experimental results demonstrate its efficiency and effectiveness.

Key Words *CAM, single stuck at fault, parallel fualt simulation, host computer, communication time*

1 はじめに

故障シミュレーションは回路に故障を想定した場合の論理シミュレーションであり，故障検査系列の故障検出率の評価，故障検査系列の生成，故障辞書の作成などに用いられ，大規模集積回路の信頼性を担う故障検査に不可欠のものとなっている [1]．しかし，こうした用途を担うには，想定され得る多くの故障に対してシミュレーションする必要がある．故障のモデルを単一縮退故障とした場合，すべての信号線に縮退故障が発生する可能性があるので，ゲート数 n の場合，その計算量は逐次計算機上では1テストベクトルあたり $O(n^2)$ となり，回路規模の増大に伴う計算量の増大が深刻な問題となっている．

このような背景から，計算時間短縮を図るため，逐次計算機上のアルゴリズムの改良のみならず，並列処理ハードウェアを用いた故障シミュレーションの大並列化というアプローチが試みられるようになった．石浦らは連想メモリの並列処理機能を利用した故障回路並列故障シミュレーション手法を提案した [2]．連想メモリはメモリとしての機能に加え，ワード並列の一致検索機能，ワード並列書き込み機能，および隣接ワード間での1ビットデータの双方向通信機能を持ち，SIMD 型の演算器として用いることが可能である．文献 [2] では，連想メモリを用いた故障回路並列故障シミュレーション手法により，逐次計算機上では $O(n^2)$ であった計算量が $O(n)$ に削減され得ることが示されている．さらに文献 [3] では，実際に連想メモリを搭載したハードウェアエンジン CHARGE II を用い，連想メモリを用いた故障回路並列故障シミュレーション速度を逐次計算機上でのソフトウェアによるシミュレーション速度と比較した結果，ISCAS'85 ベンチマーク回路では1テストベクトルあたり 2.56〜5.4 倍高速であることが確かめられた．

一方，文献 [2] の手法では，回路の1信号線に対し連想メモリの1ビットを割り当て，故障回路を並列に記憶させているので，1テストベクトルあたりのシミュレーションに必要な記憶量は $O(n^2)$ である．そのため，大規模回路のシミュレーションでは，必要な回路情報を連想メモリ上に一度に記憶できないという問題が生じる．文献 [3] では，全故障回路のシミュレーションを連想メモリ上に一度に記憶可能な数毎に分割して処理し，各分割処理が終了する毎に結果をホストコンピュータに伝達し，次の処理に必要な情報をホストコンピュータから受け取ることで，全故障回路のシミュレーションを可能にしている．しかし，全実行時間の大部分がホストコンピュータとの通信によって占められているため，ソフトウェアによるシミュレーションに比べて実行時間は短縮されたものの，ホストコンピュータとの通信による時間のオーバヘッドが，連想メモリを用いた並列故障シミュレーションの実用化の妨げとなっていた．

ホストコンピュータとの通信回数を削減し，実行時間の短縮を図るには，連想メモリに一度に記憶可能な故障回路の数を増やす必要があり，連想メモリ自体の大容量化と必要記憶量の削減が不可欠な課題とされてきた．連想メモリの大容量化は，画像処理や ATM スイッチ等の他のアプリケーションからの要望もあり多数の報告がなされているが [4],[5],[6]，大容量化するためにはコンパレータ数の削減といった機能面での犠牲が多く性能の劣化を考えると大規模な大容量化は難しい [7]．故障シミュレーションに要する記憶量の削減に関しては，連想メモリ内ビットの再利用によって記憶量を削減し高速化を実現した手法が報告されている [8]．しかし，連想メモリの大容量化が困難であること，および今後の LSI 技術の進歩によるシミュレーション対象回路の大規模化・高性能化を考えると，文献 [8] の手法を用いたとしても，回路の大規模化に伴いホストコンピュータとの通信回数が増加するので実用化には十分とはいえない．

本稿では，連想メモリを搭載したハードウェアエンジン CHARGE II 上に通信用 RAM を設け，ホストコンピュータの代わりに通信用 RAM と通信することにより，ホストコンピュータとの通信による遅延時間を削減する手法を提案する．

本稿は以下のように構成される．2章で我々が提案しているハードウェアエンジン CHARGE II について説明し，3章で連想メモリを用いた故障回路並列故障シミュレーションの基本手法を説明する．4章では並列故障シミュレーションの高速化手法を提案し，5章で評価結果を示す．6章で本稿をまとめる．

2 CHARGE II

本章では，我々が提案している連想メモリを搭載したハードウェアエンジン CHARGE II について説明する．CHARGE II は VME バスを通じてホストマシンであるワークステーション SUN4 に接続されており，ホスト計算機上で CHARGE II を制御することが可能な連想プロセッサシステムが構築されている [3]．まず最初に連想メモリの基本構造と機能，続いて連想プロセッサシステムを説明する．

2.1 連想メモリ

図 1に連想メモリの基本構造を示す．メモリセルアレイの各ワードには RAM と同様，固有のアドレスが与えられており，外部からアクセスすることが可能である．以下に連想メモリの基本機能を示す．

2.1.1 一致検索機能

連想メモリは，外部から与えたデータと一致するワードを並列に検索する一致検索機能を持ち (図 2)，検索結果と

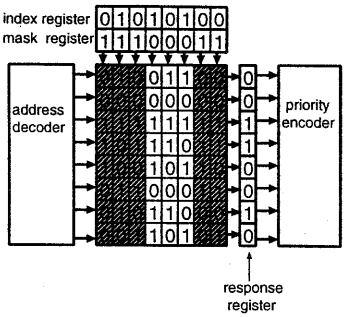

図 1: 連想メモリの基本構造

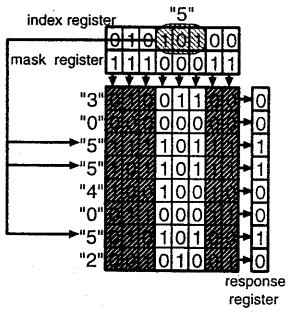

図 3: 並列書き込み

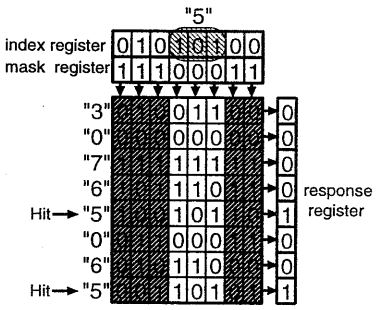

図 2: 一致検索

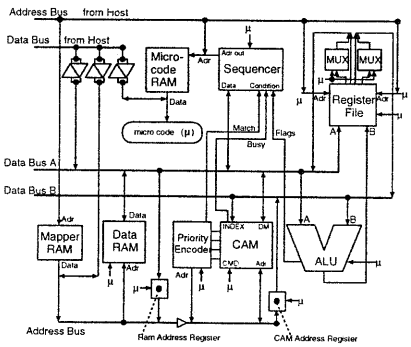

図 4: CHARGE II ブロック図

して得られたワードに対し同時にアクセスすることができる．

検索結果を格納するために，各ワードには 1 ビットのレスポンスレジスタ (RR) が付属している．一致検索では，インデクスレジスタに検索データを格納し，マスクレジスタで検索対象となるビットを指定する．命令を実行すると，指定された部分が検索データと一致する全てのワードの RR が同時に 1 か 0 に設定される．この検索に要する時間はメモリアレイのワード数によらず一定である．

一致検索により検索されたワードの中から 1 ワードのみを選択したい場合，プライオリティエンコーダ (PE: Priority Encoder) を用いることにより，その中から 1 ワードを選択しアクセスすることが可能である．この場合，PE の出力するアドレスを用いてアクセスすればよい．

2.1.2 並列書き込み

連想メモリは，RR で指定したワードに対し，インデクスレジスタの値を並列に書き込む並列書き込み機能を持つ (図 3)．並列書き込みでは，並列書き込みするワードを RR で指定し，値を書き込む部分のビットをマスクレジスタで選択する．並列書き込み命令を実行すると，RR で指定されたワードのマスクレジスタで選択された部分に，インデクスレジスタの値が並列に書き込まれる．データを書き込むワードを指定する場合，$RR = 1$ あるいは $RR = 0$ のワードや全ワードを指定することも可能である．

2.1.3 シフト機能

連想メモリには，レスポンスレジスタの値を上位もしくは下位に 1 ビットずつシフトする機能がある．このシフトは，全ワードのレスポンスレジスタに対して同時に実行される．

2.2 連想プロセッサシステム

連想メモリを搭載したハードウェアエンジン CHARGE II のアーキテクチャを図 4に示す．CHARGE II はプロセッサとしてシーケンサ LSI を持ち，マイクロコード RAM に書かれた 80 ビットのマイクロコードに従って自走する VLIW 型のアーキテクチャをとっている．

CHARGE II は，ホストコンピュータである SUN4 の内部バス (VME バス) に専用インターフェースを通して接続されている．ホストコンピュータには，CHARGE II

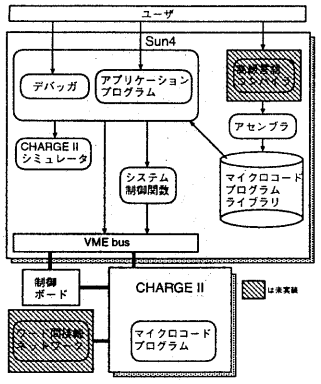

図 5: 連想プロセッサシステム

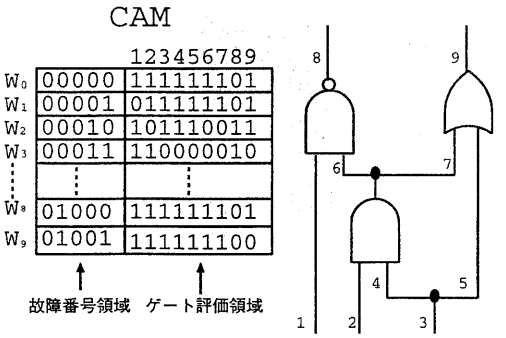

図 6: 故障回路の記憶方法

が持つ RAM や CAM 等のモジュールがマッピングされ，ホストコンピュータにより主記憶と同様にアクセスが可能である．ホストコンピュータは CHARGE II を起動させる際に，実行命令を CHARGE II のマイクロコード RAM に，実行に必要なデータをデータ RAM に，変数や定数をレジスタファイルに，連想メモリ上で処理すべきデータを CAM に書き込む．ホストコンピュータ上で動作するアプリケーションプログラムは，必要に応じて CHARGE II を起動し，結果を読み出して処理することが可能である (図 5)．

3 故障並列故障シミュレーション

故障並列故障シミュレーションとは，テストパタンを 1 つ与え，複数の故障回路を並列にシミュレーションするものである．本章では，連想メモリを用いた故障並列故障シミュレーションの基本的手法 [2] を紹介する．ここで扱う故障は単一論理縮退故障で，対象とする回路は組み合わせ回路と同期式順序回路である．

連想メモリ上に故障回路を並列に記憶させ，シミュレーションを実行するには，ゲート評価や故障の挿入といった故障シミュレーションに必要な操作を連想メモリ上で処理しなければならない．以下にこれらの処理方法を説明する．

3.1 故障回路の記憶

図 6に示すように故障回路を記憶することを考える．図 6 において，W_0は正常回路である．W_1から W_9は，その番号と対応する信号線[1]に縮退故障が存在している故障回路である．故障番号領域には，各故障回路番号が 2 進数で表示されている．ゲート評価領域には回路の信号線の値が記憶されている．ゲート評価領域中の各ビットには信号線番号が割り当てられており，その数字と対応する信号線の値を記憶している．すなわち番号 b が割り当てられているビットには，全ての故障回路に共通して，信号線 b に記憶されるべき値が格納されている．

3.2 ゲート評価

ゲートはレベルソート順に評価される．図 6では，AND ゲート，NAND ゲート，OR ゲートの順に評価される．以下に，連想メモリで 1 つのゲートを評価する方法を説明する．但し，連想メモリの 1 ワードが m ビットで構成されている場合，m ビット全てが 1 である 2 進数を $all_m(1)$，m ビット全てが 0 である 2 進数を $all_m(0)$ と書くことにする．

1. AND, NAND ゲートを評価する場合，インデクスレジスタに $all_m(1)$ を，OR, NOR ゲートの場合は $all_m(0)$ を記憶する．
2. ゲートの入力信号線に対応しているビットのみのマスクを外し一致検索することで，一致したワードの RR に 1 を不一致のものには 0 を格納する．
3. AND, NOR ゲートの場合はインデクスレジスタに $all_m(1)$ を，OR, NAND ゲート場合はインデクスレジスタに $all_m(0)$ を記憶する．
4. ゲートの出力信号線に対応しているビットのみのマスクを外す．
5. $RR = 1$ のワードに対し，インデクスレジスタ値を並列に書き込む．
6. AND, NOR ゲートの場合はインデクスレジスタに $all_m(0)$ を，OR, NAND ゲート場合はインデクスレジスタに $all_m(1)$ を記憶する．
7. マスク情報は変えずに，$RR = 0$ のワードに対し並列書き込みする．

[1] 信号線とは分岐のない結線要素であり，k本のファンアウトを持つネットは $k+1$ 本の信号線として扱うものとする．信号線番号はレベルソート順に決定されている．

以上がゲート評価の方法である．EXORは，以上の操作を組み合わせることで評価する．

3.3 故障の挿入

故障回路番号jの故障回路は，信号線jに単一縮退故障が存在している回路である．故障回路jに故障を挿入する場合は，信号線jを入力信号線とするゲートが評価される直前に以下の処理をし，縮退故障を所定のビットに挿入する．

1. インデクスレジスタに，故障番号領域の値jを2進数で表示したデータf_jを記憶する．
2. 故障番号領域のみのマスクを外し，一致検索する．一致したもののRRを1，不一致のものを0にする．
3. 縮退故障が0縮退故障の場合は$all_m(0)$を，1縮退故障の場合は$all_m(1)$をインデクスレジスタに記憶させる．
4. ゲート評価領域中のjビット目のみのマスクを外し，$RR=1$のワードに対し並列書き込みする．

以上により，故障番号jの故障回路を記憶しているCAMワード中の，ゲート評価領域がj番目のビットに縮退故障が挿入される．すべてのゲートの評価が終了した時点で，外部出力信号線値が正常回路のものと異なる故障回路が検出可能な回路である．

4 並列故障シミュレーションの高速化手法

本章では通信用RAMをCHARGE II内に設けることで故障シミュレーションを高速化する手法を提案する．まず最初に従来手法において故障シミュレーションの高速化の妨げになっていた要因を考え，続いて故障シミュレーションの高速化手法を提案する．

4.1 従来手法

あるテストベクトルを入力とし，回路中の全てのゲートに対し，ゲート評価と故障の挿入を繰り返すことで，並列故障シミュレーションは実行される．全ゲート評価が終了した時点で，外部出力信号線の値が正常回路の値と違う故障回路が，そのテストベクトルで検出可能なものとなる．回路中の信号線数をpとした場合，故障回路の数はpであるので，逐次型計算機では正常回路と合わせて$(p+1)$回路のシミュレーションを実行しなければならなかった．ゆえに，ゲート数nの場合，各回路に対しn回のゲート評価が必要であるので，計算量は$O(n^2)$であった．これに対し，連想メモリを用いた場合は，故障回路を並列にシミュレーションできるので，1回路分のゲート評価のみでよく，$O(n)$の計算量で実行可能となる．扱う回路規模が大きくなると，一故障回路の全情報を連想メモリ中の一ワード中に記憶できない場合が生じる．この場合の対処策として，文献[2]では，一故障回路の情報を複数ワードに分割して記憶させ，それを一単位として扱うことによりシミュレーションを可能としている．

一故障回路を複数ワードにわたって記憶させることにより，ある程度まで規模の大きい回路の全故障に対してもシミュレーション可能となるが，連想メモリの容量に制限があるので，一定規模を越えると，全故障回路を連想メモリ中に記憶できないという問題が生じる．CHARGE IIは，1ワードが36ビットで構成され，ワード数が2048であるので，ISCAS'85ベンチマーク回路c7552のような大規模回路では，全故障回路を一度に記憶できない．文献[3]では，全故障回路を一度に記憶可能な数毎に分割して処理している．各分割処理が終了する毎に，得られた結果をCHARGE IIからホストコンピュータに転送し，さらに次の処理に必要なデータをホストコンピュータからCHARGE IIへと送ることにより，全故障回路のシミュレーションを可能としている．しかし，CHARGE IIとホストコンピュータとの通信に平均して1000倍近くの時間を要している．

回路規模の増大に伴い，一定容量の連想メモリで同時に処理できる故障回路数(並列度)が低下しホストコンピュータとの通信回数が増えること，ホストコンピュータとの通信を含めた1テストベクトルあたりの処理時間が，ISCAS'85ベンチマーク回路を対象とした場合，CHARGE IIの処理速度がソフトウェア(Sun Sparc Staion 2使用)に対して2.56～5.4倍程度であると文献[3]で報告されていることを考えると，ホストコンピュータとの通信によるボトルネックが深刻な問題となっている．

文献[8]は，必要記憶量を削減すればホストコンピュータとの通信回数が減るという立場から，後のゲート評価で不要となった信号値を格納しているビットを再利用することでメモリを有効利用する手法を提案している．しかし，この方法を用いても回路規模の増大に伴い並列度が減少し，通信回数が増加する．

4.2 通信用RAMを用いた並列故障シミュレーションの高速化

大規模回路の故障シミュレーションにおいて全故障回路を一度に処理できる数毎に分割してシミュレーションする場合，従来はそれぞれを独立した処理として取り扱っていた．つまり，各処理毎にCHARGE IIからホストコンピュータに結果を転送し，ホストコンピュータから次の処理に必要なデータを送るという処理をただ単に繰り返していた．この様子を図7に示す．これにより，全故障回路を分割した数だけホストコンピュータとの通信が必要となっていた．分割処理する際にホストコンピュータとの通信が必要であった理由を示す．

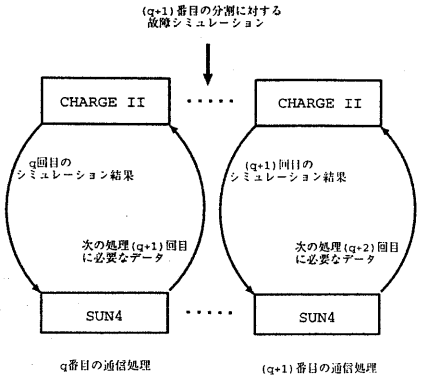

図 7: ホストコンピュータと通信する手法

1. 故障の検出が可能であったか否かを外部に読み出さなければならない.

2. 連想メモリ内の故障番号領域には，各故障回路の番号が記憶されているが，各分割処理毎に記憶する故障回路が全て入れ替わるので，故障番号領域の値を書き換えなければならない.

3. シミュレーションを実行する前に，入力信号値を記憶すべきビットに対して入力信号値を前もって記憶させておかなければならないが，入力信号線が故障している故障回路では，その信号線の値が他の故障回路と異なっている場合があるので，各分割処理毎にその値を書き直さなければならない.

以上のように，CAM のデータを読み出すことや CAM にデータを書き込むことが必要であることから，各分割処理毎にホストコンピュータとの通信が必要であると考えられていた．しかし，連想メモリの容量に限界がなければ，本来は同時にシミュレーションできるものを分割して処理しているだけなので，以上に示した問題点を CHARGE II 内で解決できれば，各分割処理毎に必要であったホストコンピュータとの通信は不要となる.

問題点 1 から 3 までの解決策を以下で順に提案する．ゲート評価や故障の挿入といった他の操作方法は，分割処理をしない場合と同じであるので，これらの問題点を CHARGE II 内で処理すれば，分割処理毎にホストコンピュータとの間で通信をしなくてもすむ.

4.2.1 結果の読み出し方法

従来，ホストコンピュータに記憶していた各分割処理毎のシミュレーション結果を，CHARGE II 内に設けた通信用 RAM に記憶する方法を以下に示す．通信用 RAM とは，回路の信号線数を p とした場合，シミュレーション結果を記憶させるために確保された，データ RAM 上のアドレス 1 から p までの領域のことである.

1. 全てのゲート評価が終了したら正常回路を記憶しているワードをアドレス (0 番地) を用いて読み出し，それをインデクスレジスタに記憶させる．次に，外部出力信号線値を記憶しているビットのマスクのみを外し一致検索することで，一致したワードの RR を 0 に，不一致のものを 1 にする．この様子を図 8(a) に示す.

2. $RR = 1$ のワードをプライオリティエンコーダ (PE) を用いて読み出し，ALU においてシフト演算を実行する．ゲート評価領域が g ビットの場合，下位へ g ビットシフトする．ここで，1 ビットシフトする毎に最上位ビットを 0 にする．この様子を図 8(b) に示す．この演算により得られる結果は，そのワードに割り当てられていた故障回路番号の 2 進数表示となる．故障回路番号が i $(1 \leq i \leq p)$ である場合，これを $A(W_i)$ と書くことにする.

3. シミュレーション結果を次のように通信用 RAM に記憶する．故障回路番号 i のシミュレーション結果は通信用 RAM 上のアドレス $A(W_i)$ に記憶する (図 8(c))．このとき，まずアドレス $A(W_i)$ に記憶されている値 R_i を読み出し，以下に示す演算を実行することによって結果を記憶する.

   いま，l を通信用 RAM のワード内のビット幅と定義する．h 番目のテストベクトルのシミュレーション結果を通信用 RAM の h ビット目に記憶する方法を示す．変数 t を l ビットの 2 進数で，h 番目のテストベクトルのシミュレーションでは，h ビット目のみが 1 で他はすべて 0 である数とする．h 番目のテストベクトルにおけるシミュレーション結果は，$R_i = R_i + t$ という演算を ALU で実行し，R_i を再び通信用 RAM に記憶させることにより格納される．全故障回路に対して h 番目のテストベクトルのシミュレーションが終了したとき，t を 1 ビット上位にシフトする．これを，$(h+1)$ 番目のテストベクトルに対して利用する．図 8(c) に示されている例は，1 番目のテストベクトルに対する結果を格納している.

この方法により，RAM の 1 ワード中のビット数が l の場合，l 通りのテストベクトルに対し，ホストコンピュータとの通信をせずに CHARGE II 上でシミュレーションできる.

4.2.2 故障番号の書き換え

各分割処理毎に変化する故障番号領域の値を CHARGE II 上で書き換える方法を以下に示す．連想メモリ上に一度に記憶できる故障回路の数を u とし，故障回路番号 v から $(v+u-1)$ までが記憶されているものとすると，次の処

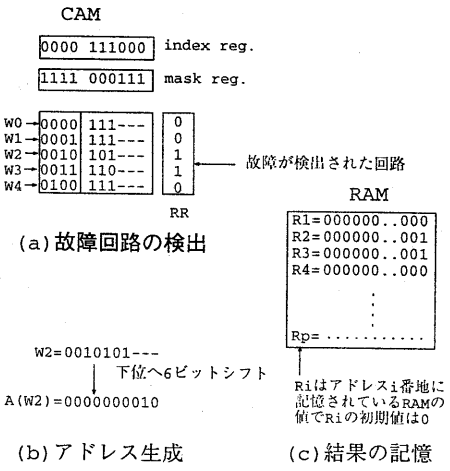

図 8: 結果の読み出し方法

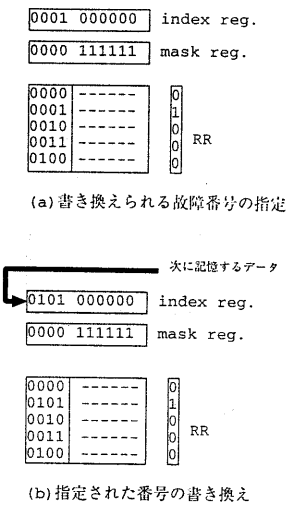

図 9: 故障番号の書き換え

理では，故障回路番号 $(v+u)$ から $(v+2u-1)$ まで記憶することになる．

1. インデクスレジスタに $preI$ を記憶する．ここで，$preI$ とは故障番号領域の値が v を 2 進数表示したものであり，ゲート評価領域の値がすべて 0 である数を示している．

2. 故障番号領域のみのマスクを外し，一致検索することで，一致したワードの RR のみを 1 にする．

3. インデクスレジスタに $nexI$ を記憶する．ここで，$nexI$ とは故障番号領域の値が $(v+u)$ を 2 進数表示したものであり，ゲート評価領域の値がすべて 0 である数を示している．

4. 故障番号領域のみのマスクを外し，インデクスレジスタ値を $RR=1$ のワードに対し並列書き込みする．

5. 故障番号領域の値が 1 を 2 進数表示したものであり (故障番号 1 に相当)，ゲート評価領域の値がすべて 0 である数を $preI$ と $nexI$ 両方に加え，新たに $preI$, $nexI$ とする．この加算は ALU で実行される．1 に戻る．

以上の書き換え操作を連想メモリ中に記憶されている全ての故障番号に対して繰り返すことにより，ホストコンピュータと通信する必要がなくなる (図 9)．回路の信号線の数を p とすると，書き換え操作の回数は p であるので，ゲート数を n としたとき，$O(n)$ の時間でシミュレーションが可能となる．

4.2.3 入力信号値の書き直し

外部入力信号線に縮退故障が存在する回路のシミュレーション後に，外部入力信号線が正常である回路をシミュレーションする場合，初期入力信号値を書き直さなければならない．各シミュレーションを実行する前に，初期入力信号値は，以下に示す方法で修正する．

1. インデクスレジスタに検索データとして，初期入力信号値を記憶しているビットが初期入力信号値であるデータを与える (図 10)．

2. 初期入力信号値を記憶しているビットのみのマスクを外し，一致検索する．一致したワードのレスポンスレジスタ値は 0 に，一致しないものは 1 となる．

3. $RR=1$ のワードに対し，初期入力信号値をマスク付き並列書き込みする．

以上の操作により，修正作業は終了する．

4.3 CHARGE II とホストコンピュータ間の通信回数

CHARGE II を起動させてシミュレーションする場合，シミュレーション開始時は，回路情報，CHARGE II の命令，レジスタや RAM, CAM に記憶するデータをホストコンピュータから CHARGE II に，シミュレーション終了時は，シミュレーション結果を CHARGE II からホストコンピュータに通信しなければならない．従来手法では，それ以外にも分割処理毎にホストコンピュータとの通信を必要としていたが，提案手法では CHARGE II 内に設けた通信用 RAM を利用することにより，l 通りのテストベ

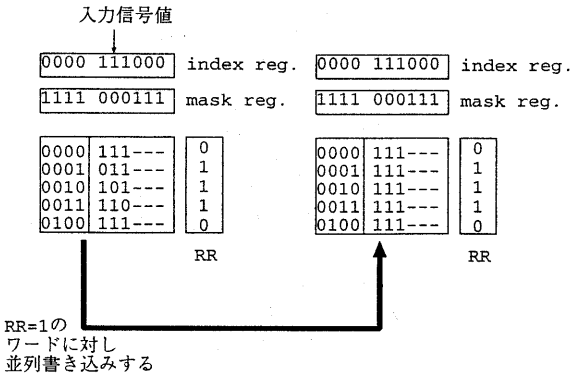

図 10: 初期入力信号値の修正

表 1: 実行時間の比較

回路	パス回数 [回]	文献 [3][s]	提案手法 [s]
c17	1	1.640	1.640
c432	1	14.884	14.884
c499	1	17.085	17.085
c880	1	33.519	33.519
c1355	1	56.013	56.013
c1908	2	152.459	76.249
c2670	3	341.959	114.039
c3540	3	465.770	155.330
c5315	5	1056.664	211.544
c6288	6	825.486	137.886
c7552	12	1614.084	135.354

クトルの全故障回路に対するシミュレーションの最初と最後のみに CHARGE II とホストコンピュータ間で通信すればよいことになる.

5 評価

1 テストベクトル当たりの CHARGE II による実行時間を, 本手法を用いた場合および文献 [3] の手法を直接用いた場合で測定した結果を表 1に示す. 用いた回路は, ISCAS'85 ベンチマーク回路である. 本シミュレーションで対象としている故障は全ネットに対してではなく, FOS (Fanout Stem)[2]における故障のみを対象としている.

表中のパス回数とは, 対象回路を全て処理するのに必要となるシミュレーション回数のことで, パス回数が w である回路は, w回に分けてシミュレーションされている. 表 1から提案手法は文献 [3] 手法に比べ, 約 w倍高速であることが示された. これは, CHARGE II とホストコンピュータ間の通信回数が, 提案手法では文献 [3] 手法に比べ $(w-1)$ 回削減され, w回から 1 回となったからである.

c1355 より小さい回路では, 文献 [3] の手法と同じデータが得られているが, これは c1355 より小さい回路のパス数が 1 であり分割処理がないためである.

6 むすび

本稿では, 故障回路並列故障シミュレーションにおいて, 連想メモリを搭載したハードウェアエンジン上に通信用 RAM を設けることでホストコンピュータとの通信による遅延を削減する手法を提案し, その有効性を示した. これにより, 従来は, 分割処理毎に必要であった CHARGE II とホストコンピュータ間の通信なしに全故障回路を CHARGE II 内でシミュレーションできるので, 対象回路の規模が増大し分割処理回数が増えても CHARGE II とホストコンピュータ間の通信回数は増加しない. 提案手法により, 並列故障シミュレーションの実用化の妨げとなっていたホストコンピュータとの通信による時間のオーバヘッドが解決されたことになる.

[2]FOS とは, 回路を FFR (Fanout Free Region: 分岐のない信号線で結ばれたゲートの集合) に分割した際, FFR の先頭ネットのことである. FOS における故障のみを対象とした方法では, 全故障を扱う必要がなく, 仮にある FOS 上で故障が検出された場合, その FFR 内での故障を検索すればよく, したがって FOS 上で故障が検出されなければ, それに対応する FFR 内に故障はなく無視することができる [3].

参考文献

[1] 藤原秀雄, コンピュータの設計とテスト, 工学図書, 1990.

[2] N. Ishiura and S. Yajima, "Linear time fault simulation algorithm using a content addressable memory," *IEICE Trans. Fundamentals*, vol .E 75-A, no. 3, pp. 41–48, Mar. 1992.

[3] 前田志門, 松下章, 大野慎介, 大附辰夫, "CAM 搭載ハードウェアエンジンとアプリケーション実装環境," 情処研報, ARC110-3, DA73-3, Jan. 1995.

[4] I. Okabayasi, H. Kotani, and H. Kadota, "A proposed structure of a 4Mbit content-addressable and sorting memory," *Proc. Symp. VLSI Circuits*, pp. 109–110. 1990.

[5] K. J. Schultz and P. G. Gulak, "Fully parallel integrated CAM/RAM using preclassification to enable large capacities," *IEEE J. Solid-Stat Circuits*, Vol. 31, No. 5, pp. 689–699, May 1996.

[6] M. Motomura, J. Toyoura, K. Hirata, H. Ooka, H. Yamada, and T. Enomoto, "A 1.2-million transistor, 33-MHz, 20-b dictionary search processor (DISP) ULSI with a 160kb CAM," *IEEE J. Solid-State Circuits*, SC-25. pp. 1158–1165. 1990.

[7] K. J. Schuitz and P. G. Gulak, "Architectures for large capacity CAMs," *Integration, The VLSI Journal*, vol. 18, pp. 151–172. June 1995.

[8] 大野慎介, 前田志門, 松下章, 大附辰夫, "連想メモリを用いた高並列故障シミュレーション手法," 情処研報, ARC110-4, DA73-4, Jan. 1995.