DA 32－3
(1977. 3. 17)

機能素子を含む論理回路の並列故障シミュレーション手法

山田昭彦　若槻信夫　大関和正　富田恭次
(日本電気株式会社)

1. まえがき

近年の集積回路技術の進歩は著しく，論理回路の高集積化は驚くべき速度で進行し，非常に多彩な機能を有する集積回路が市場に出回っている。これらの集積回路は，コンピュータを始めとする論理回路の設計者に，設計作業の簡単化という多大のメリットを与え，設計回路をゲート・レベルで考える煩わしさから解放している。しかも，これらの集積回路の中には，すでにゲート・レベルで考えることができない，あるいは考えることが無意味であるものも多数存在している。一方これらの集積回路，あるいはこれらの集積回路から構築される論理回路に対する試験は，装置全体の信頼性の向上，出荷検査期間の短縮の意味あいからも，その重要性を増している。

論理回路の集積および実装密度の向上は，将来にわたって継続するであろう。この現実に対処するために，この分野のＤＡプログラムに求められる手段の１つは，あるまとまった機能を有する論理回路を機能ブロックとして処理することであり，これは最も自然であり，有効な方法であると考えられる。機能シミュレーション法は，次のような点でゲート・レベル・シミュレーション法より優れている。

(1) 論理回路のモデリングがコンパクトになる。このことは，有限であるメモリーを有効に使用することにより，少ないメモリー容量で大きな論理回路を表現することを可能とする。
(2) トランザクション量（イベント数）が減少することにより，シミュレーションの高速化を可能とする。
(3) 繁雑なゲート・レベル展開処理を省略できる。

機能シミュレーション法は，上記のような優れた特性を有するが，ゲート・レベル・シミュレーションに比較してタイミング問題に若干厳密さを欠くことは周知のごとくである。しかも，機能シミュレーション法が適用できないランダム・ロジックが数多く存在するのも事実である。ゆえに，機能シミュレーション法をすべての論理回路に適用するのは無理であり，適用するか否かの判断基準は次のようなものになろう。

(1) 汎用性があること。論理回路中，広範に使用され，その使用頻度が高いもの。
(2) 論理回路のゲート・レベル・モデリングに比較して，ゲート数の削減効率の高いもの。
(3) 論理回路の有する機能が，まとまっていてシンプルであること。
(4) ゲート・レベルで等価回路が表現できないもの。

さて，故障シミュレータに関して，これまで種々の考察がなされて来た。古典的な並列法故障シミュレータ，デダクティブ法故障シミュレータそして最新のコンカレント法故障シミュレータがある。しかし，いずれの方法もゲート・レベルに限定されており，機能シミュレーション法についての考察は，並列故障シ

ミュレータについて〔7〕で，デダクティブ法故障シミュレータについて〔6〕でなされているが，実用化に関する詳細な報告は，いずれの場合も報告されていない。本故障シミュレータについては，既に〔8〕でその詳細が報告されており，本論文において既発表の論文内容と重複するところがあるが，それ以降の研究成果であるＲＡＭ，ＲＯＭ，Register File に対する簡単でしかも有効であるシミュレーション手法，とくにこれらの機能素子の並列演算手法を中心に話を進めて行く。なお，ここで述べられる並列演算法は，並列故障シミュレーション環境において開発され，使用されているものであるが，他の応用例として，並列演算を必要とする論理シミュレータに対しても十分適用できる手法である。

2. シミュレータの概要と外部環境

本故障シミュレータは，並列方式の故障シミュレータであり最大９６０の単一故障回路と１つの正常回路を同時にシミュレートでき，ＡＣＯＳ７７シリーズ・コンピュータをはじめとする種々の論理装置のメモリー専用パッケージと非論理回路パッケージを除く広範囲な論理パッケージ（Printed Circuit Board）の故障診断ツールとして使用されている。　本故障シミュレータの採用している基本的シミュレーション手法と外部環境について説明する。

2.1　セレクティブ・トレース＆タイム・マッピング法

世間一般の論理シミュレータが採用しているこれらの手法を本故障シミュレータも採用している。　これらの手法を採用することにより処理対象となる論理パッケージの回路上の制約はない。

2.2　基本素子

Table 1 は，本故障シミュレータで解釈可能な基本素子の一覧である。　フリップ・フロップは，エッジトリガーおよびレベルトリガーのものを含む。　また基本素子にＲＡＭ，ＲＯＭなどの機能素子を加えたことによりシミュレーション対象論理パッケージは拡大した。　さらに，これらの機能素子の演算を，並列演算の特性を生かすブーリアン方程式記述の演算式に展開できたことにより，これらの機能素子を含む論理パッケージをも効率良く処理できる。

No.	Primitive Element Type
1	N-inputs AND
2	N-inputs NAND
3	N-inputs AND/NAND
4	N-inputs OR
5	N-inputs NOR
6	N-inputs Exclusive OR
7	N-inputs Exclusive NOR
8	Amp.
9	Amp. Inverter
10	Input Connector
11	Output Connector
12	Test Pad
13	Wired AND
14	Wired OR
15	D type flip-flop
16	J-K flip-flop
17	T type flip-flop
18	R-S flip-flop
19	One-shot
20	Random Access Memory
21	Read Only Memory
22	Content Addressable Memory
23	Register File

Table 1　Available primitive elements

2.3　２値＆３値演算

最大９６０の故障回路はそれぞれ１ビット

で，そして正常回路は6ビットで状態表上に表現される。　故障回路は0，1の2値で，正常回路は0，1，Xの3値で演算される。　第3の値Xは，信号の過渡状態を表わすのではなくて，回路の未知状態を表わすのに用いられ，回路の初期状態の設定に使用されている。　初期状態設定の完了後，故障回路および正常回路について一様にブーリアン演算が適用される。

$$A = B \cdot C = \{ b \cdot c , b_1 \cdot c_1 , b_2 \cdot c_2 , \cdots , b_n \cdot c_n \}$$

$$A = B + C = \{ b + c , b_1 + c_1 , b_2 + c_2 , \cdots , b_n + c_n \}$$

$$\bar{A} = \{ \bar{a} , \bar{a_1} , \bar{a_2} , \cdots \bar{a_n} \}$$

fault-free circuit

fault circuits

where
$$A = \{ \boxed{a} , \boxed{a_1 , a_2 , \cdots , a_n} \}$$
$$B = \{ \boxed{b} , \boxed{b_1 , b_2 , \cdots , b_n} \}$$
$$C = \{ \boxed{c} , \boxed{c_1 , c_2 , \cdots , c_n} \}$$

2.4　故障の仮定と検出

仮定故障は，単一固定0，1故障に限定している。　故障は，論理回路を構成している基本素子の入出力ピン上に仮定され，等価および冗長故障の削減が可能な範囲で行なわれる。

故障の検出は，テスト・ポイントで正常回路の状態値と故障回路の状態値が相違するかどうかで決定される極めて簡単な方法で行なわれる。

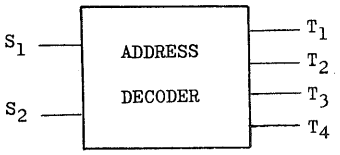
3.　機能素子の並列演算手法

集積回路技術の進歩による素子の高密度化と，集積回路の実装効率の進歩によるシミュレーション単位当りの素子数の増大に対して機能シミュレーション法がゲート・レベル・シミュレーション法に比して有効であることは先に述べた通りである。　本故障シミュレータは，さらに一層の改善と拡張が必要であるが，数種の機能素子について機能シミュレーション法を適用し良い結果を得た。　ここでは，これらの機能素子に適用された並列シミュレーション法，特に並列処理の高速化につながるブーリアン演算式への展開方法について述べる。

3.1　並列アドレス・デコーディング手法

各種のデコーダは，論理回路中頻繁に使用される汎用性の高い回路である。後で述べるROM，RAM中にも必要とされるワード選択用として内蔵されている。　その他マルチプレクサ，セレクターとして使用されており，本手法をこれらの機能素子に適用することは，極めて順当なところである。

Figure 1Aに2入力4出力のアドレス・デコーダとその真理値表が，示されている。　出力Tj（j＝1～4）は，入力Si（i＝1，2）の2進数表記に依存していることがわかる。　この関係に注目してプログラミングしたものがFigure 1Bで



TRUTH TABLE

S_2	S_1	T_1	T_2	T_3	T_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figure 1A Address decoder and its truth table

```
m = 2
B = 0
DO j = 1 TO 2^m
    Tj = all 1
    DO  k = 1 TO m
        IF kth bit of B = 0
            THEN Tj = Tj · S̄k
            ELSE Tj = Tj · Sk
    END
    B = B + 1
END
```

Figure 1B Parallel evaluation routine for address decoder

ある。 手続き中の変数mはデコーダへのアドレス入力数を表わし，変数Bは2進表記を表わすのに使用されている。 Si およびTj は，すべてベクトルである。 変数Bのビット構成により，この手続きは，(1)式を導き，Figure 1A に示されている真理値表と等価であることがわかるであろう。

$$\left.\begin{array}{lll} \text{When} \quad B = 0\ 0 & , & T_1 = \overline{S}_2 \cdot \overline{S}_1 \\ 0\ 1 & & T_2 = \overline{S}_2 \cdot S_1 \\ 1\ 0 & & T_3 = S_2 \cdot \overline{S}_1 \\ 1\ 1 & & T_4 = S_2 \cdot S_1 \end{array}\right\} \quad (1)$$

Figure 1C は，Figure 1A で示されたアドレス・デコーダの並列故障シミュレーション例である。 Gは正常回路であり，F_1，F_2，F_3 は，S_1，S_2 に故障の影響を与えている故障回路である。 正解値と故障回路値は，同時にTj (j=1～4)上に求められている。

	G	F_1	F_2	F_3
S_1	0	0	1	1
S_2	0	1	1	0
T_1	1	0	0	0
T_2	0	0	0	1
T_3	0	1	0	0
T_4	0	0	1	0

Figure 1C An example of parallel fault simulation for address decoder with two BCD inputs and four outputs

3.2 Read Only Memory

シミュレーションに先立ち，該当するＲＯＭのメモリー内容がビット・モードで記述されたＲＯＭファイルが読み込まれ，ＲＯＭ専用テーブルに同じくビット・モードで書き込まれる。　ＲＯＭ専用テーブルは，状態表と形式において全く異っており，正常回路の状態値しか記述されない。　なぜならＲＯＭは，動作時にメモリー内容を変化させることはなく，故障回路と正常回路の影響から完全に独立していると考えられるからである。

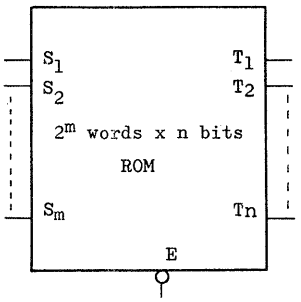
一般にＲＯＭは，m個のアドレス入力をデコードするアドレス・デコーダと，2^mワード×nビットのメモリー・アレイから構成されていると考えられるからＲＯＭの演算処理は，アドレス選択動作とメモリー読出し動作の並列処理を実現すればよいことになる。アドレス選択動作は，3.1で述べた手法がそのまま適用できる。

Figure 2にＲＯＭの並列演算手続きが示されている。

すべてのワードがアクセスされ出力$T_1 \sim Tn$上に出力結果が，パラレルに求められる。

Bj，kは，j番目のワードのk番目のビットを意味する。

Aには，選択するワードをマスクするマスク・ベクトルが，アドレス・デコーディング手法を利用して作成される。



Where $S_1 \sim S_m$ = Address Inputs,

E = Chip Select Inputs,

$T_1 \sim T_n$ = Data Outputs.

```
DO  j = 1 TO n
|   Tj = all 0
END
```

INITIALIZATION

```
B = 0
DO j = 1 TO 2^m
|   A = all 1
|   DO  k = 1 TO m
|   |   IF kth bit of B = 0
|   |     THEN  A = A · S̄k
|   |     ELSE  A = A · Sk
|   END
|   DO  k = 1 TO n
|   |   Tk  = Tk + A · Bj,k
|   END
|   B = B + 1
END
```

ADDRESS DECODING OPERATION

READ OPERATION

```
DO  j = 1 TO n
|   Tj = E + Tj
END
```

OUTPUT ADJUSTMENT OPERATION

Figure 2　Simplified evaluation routine for 2^m words x n bits ROM

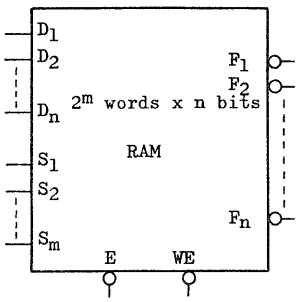
3.3 Random Access Memory

ＲＡＭの演算手続きは，ＲＯＭのRead動作にWrite動作が加わるだけでＲＯＭと大差ない。 ただしWrite動作のために ビット数分だけのメモリー・エリアが，状態表上に確保されることが，ＲＯＭと異なる点である。 すなわち６４ビットＲＡＭなら６４個，２５６ビットＲＡＭなら２５６個のメモリー・エリアが必要となる。 こうすることにより故障回路と正常回路のメモリーに対する影響を並列に処理することが可能となる。

Figure 3は，ＲＡＭの並列演算手続きを示している。

3.4 Register File

Figure 4にRegister Fileの並列演算手続きが示されている。 使用されている手法は，すでに述べられたものの延長である。



Where $D_1 \sim D_n$ = Data Inputs,
$S_1 \sim S_m$ = Address Inputs,
E = Chip Select Input,
WE = Write Enable Input,
$F_1 \sim F_n$ = Data Outputs.

```
DO j = 1 TO n
|  Fj = all 0
END
```

INITIALIZATION

```
B = 0
W = Ē · WE
DO j = 1 TO 2^m
|  A = all 1
|  DO k = 1 TO m
|  |  IF kth bit of B = 0
|  |    THEN A = A · S̄k
|  |    ELSE A = A · Sk
|  END
|  DO k = 1 TO n
|  |  Bj,k = W·A·Dk + W·A·Bj,k
|  |  Fk = Fk + A·Bj,k
|  END
|  B = B + 1
END
```

ADDRESS DECODING OPERATION

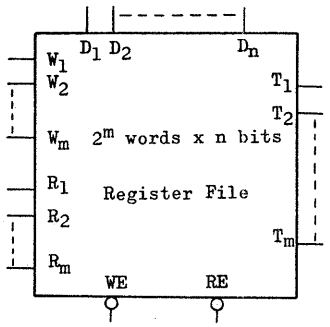
WRITE OPERATION

READ OPERATION

```
R = Ē · WE
DO  j = 1 TO n
|  Fj = R̄ · Fj
END
```

OUTPUT ADJUSTMENT OPERATION

Figure 3 Simplified evaluation routine for 2^m words x n bits RAM



Where $W_1 \sim W_m$ = Write Address Inputs,

$R_1 \sim R_m$ = Read Address Inputs,

$D_1 \sim D_n$ = Data Inputs,

WE = Write Enable Input,

RE = Read Enable Input,

$T_1 \sim T_n$ = Data Outputs.

```
DO  j = 1 TO 2^m
|   A = address mask bits pattern is
|       generated from the write
|       address inputs using address
|       decoding technique explained.
|   A = WE · A
|   DO  k = 1 TO n
|   |   B_j,k = A·D_j + Ā·B_j,k
|   END
END
```

WRITE OPERATION

```
DO  j = 1 TO 2^m
|   A = mask bits pattern is generated
|       from the read address inputs
|       in the same manner used in
|       WRITE OPERATION.
|   DO  k = 1 TO n
|   |   T_k = T_k + A · B_j,k
|   END
END
```

READ OPERATION

```
DO  j = 1 TO n
|   Tj = RE + T_j
END
```

OUTPUT ADJUSTMENT OPERATION

Figure 5 Simplified evaluation routine for 2^m words x n bits Register File

4. シミュレーション結果と考察

各種 Flip-Flop, One-Shot, Content Addressable Memoryの機能シミュレーション手法は，すでに〔8〕で紹介されたので本論文では割愛した。

本故障シミュレータは，PL/I-likeコンパイラー言語とアッセンブラー言語を用いてインプリメントされ，５２４Ｋキャラクター・メモリーを有するNEAC２２００シリーズ・コンピュータ・モデル７００で現在稼動している。

Table 2 に機能素子を含む論理パッケージの故障シミュレーション結果が示されている。 回路Ａは，機能素子を含まないが非常に複雑な非同期順序回路である。 ３０個の4 by 4 Register File と制御ゲートから構成されている回路Ｆについて，機能シミュレーション法の有効性を調査するための実験を行った。

F-aは，Register File に機能シミュレーション法が適用された結果であり，F-bは，各Register File をゲートと１６個のＴタイプFlip-Flopに展開して故障シミュレーションを行った結果である。 ＣＰＵタイムで約10倍，シミュレーション効率で約２倍，機能シミュレーション法が優れていることが実証された。この傾向は，機能素子の占める比率の高い論理パッケージほど，顕著になるであろう。

なお機能シミュレーション法を推進するにあたり，機能素子の内部故障が，試験に及ぼす影響を十分に把握しなくてはならない。 なぜなら故障の設定が機能素子を含む基本素子の入出力ピン上にしかなされないので，入出力ピン上の故障で代表できない内部故障については，シミュレーション対象から除外されているからである。 この問題についての今後の研究を筆者らは，切望している。

		*1 Gates (No.)	Flip-Flops (No.)	Functional elements (No.)	Faults simulated (No.)	Vectors simulated (No.)	Faults detected (%)	CPU time (sec.)	*2 Efficiency
A		541	0	0	1307	647	91	200	4.23
B		1156	32	0	3471	778	82	705	3.83
C		434	12	1K ROM:3	1175	2431	90	975	2.93
D		586	32	1K ROM:3 4×4 CAM:3	1646	2160	84	3043	1.17
E		446	11	16×4 RAM:9	1428	676	94	840	1.15
F	a	527	12	4×4 Register file : 30	1897	372	90	572	1.23
	b	2099	492	0	8425		98	5340	0.59
G		161	0	16×4 RAM:30	803	842	92	1524	0.44

*1 Number of gates excluding flip-flops and functional elements.

*2 $\text{Efficiency} = \frac{\text{Faults} \times \text{Vectors}}{\text{CPU Time (ms.)}}$

Table 2 Simulation results of this fault simulator

≪ 謝　辞 ≫

本故障シミュレータの開発に当り，多大な御援助と御指導を戴きました宮城本部長，小林部長，北村部長，秋野課長，長束課長に深く感謝致します。

≪ 参考文献 ≫

[1] D. B. Armstrong: "A Deductive Method for Simulating Faults in Logic Circuits", IEEE Transactions on Computers, May, 1972.

[2] H. Y. Chang, S. G. Chappell, C. H. Elemendorf and L. D. Schmidt: "Comparison of Parallel and Deductive Fault Simulation Methods", IEEE Transactions on Computers, November, 1974.

[3] E. G. Ulrich: "Time-Sequenced Logical Simulation Based on Circuit Delay and Selective Tracing of Active Network Paths", ACM 20th National Conference, 1965.

[4] E. G. Ulrich: "Exclusive Simulation of Activity in Digital Networks", Communication of ACM, February, 1969.

[5] D. M. Schuler, E. G. Ulrich, T. E. Baker and S. P. Bryant: "Random Test Generation Using Concurrent Logic Simulator", Proceedings of 12th DA Workshop, June, 1975.

[6] S. G. Chappell, P. R. Menon and J. F. Pellegrin: "Functional Simulation in the LAMP System", Proceedings of 13th DA Workshop, June, 1976.

[7] S.A. Szygenda and F.W. Thompson: "Digital Logic Simulation in a Time-Based, Table-Driven Environment", COMPUTER, March, 1975.

[8] 山田昭彦，若槻信夫，富田恭次：「非同期イベント方式パラレル故障シミュレータ」，「情報処理」第17巻7号，昭和51年7月