順序回路に対する消費電力削減のためのテストベクトル変更法

樋上喜信† 小林真也† 高松雄三†

携帯用機器に搭載されるような低消費電力のVLSIでは，テスト時においても消費電力を考慮したテストベクトルの印加が必要である．本論文では，CMOS順序回路のテスト時の消費電力を削減するようにテストベクトルを変更する手法を提案する．提案法では，回路の消費電力を信号値が遷移するゲート数で評価し，与えられたテスト系列に対して，元の縮退故障検出率を維持したまま，信号値が遷移するゲート数ができるだけ少なくなるように変更する．まず，与えられたテスト系列を複数の部分系列に分割し，その後各部分系列を，(1) 回路の状態を遷移させる部分系列と，(2) 故障を活性化し故障の影響を外部出力に伝搬する部分系列，の2種類に分類する．テストベクトル変更については，外部入力値を1ビットずつ反転させる手法を用いる．元の縮退故障検出率を保証するためには，(1) の部分系列については論理シミュレーションを，(2) の部分系列に対しては故障シミュレーションを用いる．最後に，提案法をプログラム化しISCAS'89ベンチマーク回路に適用した結果によって，提案法の有効性を確認する．

Modifying Test Vectors to Reduce Power Dissipation for Sequential Circuits

Yoshinobu Higami,† Shin-ya Kobayashi† and Yuzo Takamatsu†

In testing of VLSIs designed for low power dissipation, test vectors that avoid excessive power dissipation should be applied. In this paper, we propose a method to modify test vectors for reducing power dissipation in CMOS sequential circuits. Since the power dissipation is proportional to the number of gates with signal value transitions, we modify test vectors so that they bring less number of gates with signal value transitions. First, we partition a given test sequence into several subsequences, and classify them into (1) subsequences that transfer a circuit to a specific state, and (2) subsequences that activate faults and propagate effects of faults to primary outputs. Next we modify test vectors by changing the value at a primary input one by one. The original stuck-at fault coverage is guaranteed by logic simulation for subsequences in (1), and by fault simulation for subsequences in (2). The proposed method is implemented by C language, and its effectiveness is shown by experimental results for ISCAS'89 benchmark circuits.

1. はじめに

近年，携帯電話に代表されるような小型の電子機器が急速に進歩・普及しているが，そのような小型の電子機器に搭載されているVLSIは，動作時の消費電力が低減するように設計・製造されている．このような低消費電力VLSIのテストにおいて，通常動作時に用いられないようなテスト系列を印加した場合には，消費電力が設計時の見積り以上に増大し，その結果誤ったテスト結果をもたらしたり，回路を破壊する可能性がある．したがって，低消費電力に設計されたVLSIのテストにおいては，消費電力を考慮したテスト系列を印加する必要がある．

テスト時の低消費電力化に関する研究については，近年特にさかんになりつつあり，代表的なのもとして以下のような研究が報告されている．たとえば，低消費電力のBIST法[5),6),10),13)]，テスト時の低消費電力化のためのATPG[3),11)]，組合せ回路のテスト時における低消費電力化[4),11)]，スキャン回路のテスト時における低消費電力化[4),8),12)]などの研究が報告されている．しかしながら，スキャン設計を行わない順序回路のテスト時の低消費電力化に関する研究はほとんど報告されていない．

低消費電力を実現するテスト系列を得るための手法としては，次の2つの手法が考えられる．

† 愛媛大学工学部情報工学科
Faculty of Engineering, Ehime University

(1) 決定論的なテスト生成アルゴリズムを用いて，高故障検出率と低消費電力を同時に満たすようなテスト系列を生成する手法
(2) 高故障検出率を達成するテスト系列に対して，元の故障検出率を保証しつつ，消費電力を削減するようにテスト系列を変更する手法

(1) の手法によって，高検出率かつ低消費電力なテスト系列を短い計算時間で得ることができるかもしれないが，欠点として，アルゴリズムが複雑になり実装が困難となることが予想される．(2) の手法の利点は，テスト系列生成法に独立であるため，どのようなテスト系列に対しても，適用可能であるという点にある．したがって，(1) の手法によって得られたテスト系列でさえ，(2) の手法を適用することによってさらに消費電力を削減することが可能となる．

本論文では，上記 (2) の手法の 1 つとして，与えられた CMOS 順序回路のテスト系列に対して，元の縮退故障検出率を維持したまま，消費電力を削減するようにテストベクトルを変更する手法を提案する．CMOS 回路の消費電力は信号値が変化するゲート数に比例するので，提案法では，信号値が遷移するゲート（**遷移ゲート**と呼ぶ）数をできるだけ減少させるようにテストベクトルを変更する．まず与えられたテスト系列に対して故障シミュレーションを行い，各故障がどの時刻のテストベクトルで検出されるかを調べる．この情報を元に，テスト系列を複数の部分系列に分割し，各部分系列を，(1) 回路を特定の状態に遷移させる部分系列と，(2) 故障を活性化し故障の影響を外部出力に伝搬する部分系列，の 2 種類に分類する．テストベクトル変更においては，外部入力値を 1 ビットずつ変更し，遷移ゲート数が減少するかどうかを調べる．遷移ゲート数の評価においては，1 時刻での最大遷移ゲート数と平均遷移ゲート数を用い，最大遷移ゲート数を増加させずに，平均遷移ゲート数が減少するように，テストベクトルを変更する．また，元の縮退故障検出率を保証するためには，(1) の部分系列に対しては論理シミュレーションを行い，(2) の部分系列に対しては故障シミュレーションを行う．提案法を C 言語を用いてプログラム化し，ISCAS'89 ベンチマーク回路に対して行った実験の結果を示し，提案法の有効性を示す．

本論文の以下の構成は次のようである．2 章では，消費電力と遷移ゲート数の関係を述べる．3 章では，テスト系列を分割・分類する手法を説明する．4，5 章では，2 種類の部分系列に対するテストベクトル変更法を説明する．6 章では，ISCAS'89 ベンチマーク回路に対して行った実験の結果を示す．最後に 7 章で本論文のまとめを述べる．

2. 消費電力と遷移ゲート数の関係

CMOS 回路の消費電力は，静的消費電力と動的消費電力に分けて考えることができる．静的消費電力としてはリーク電流があり，動的消費電力としては，負荷容量の充放電によるものと，VDD から GND への貫通電流によるものがある．一般的には，これらの成分のうち負荷容量の充放電によるものが支配的であり，結果として CMOS 回路の組合せ回路部分における消費電力は，

$$P = \frac{1}{2} \cdot f \cdot V_{DD}^2 \sum_g C_g \cdot N_g(v_1, v_2)$$

で表される[1)~4),9)]．ここで，f は動作周波数，V_{DD} は電源電圧，C_g はゲート g の負荷容量を表す．また，$N_g(v_1, v_2)$ は，入力ベクトル v_1，v_2 を印加したときのゲート g における信号値の遷移回数を表す．過去の研究と同様，すべてのゲートの負荷容量が同一[2)]で，ゲートの遅延がゼロ[4),6)]であると仮定すると，消費電力を最小化する問題は，$\sum_g N_g(v_1, v_2)$ が最小となるような入力ベクトル v_1，v_2 を求める問題となる．遷移ゲート数を求める際，ゲートの遅延を考慮することによって，より正確な値を求めることが可能であるが，反面，(1) シミュレーション時間が増大する，(2) モデルによっては過大評価となる[4)]，という問題がある．そこで本研究では遷移ゲート数を求める際には，ゼロ遅延モデルのシミュレーションを用い，遷移ゲート数ができるだけ少なくなるようにテストベクトルを変更する．

3. テスト系列の分割・分類

提案法では，与えられたテスト系列に対して，故障シミュレーションを行い，対象となる縮退故障の 1 つ 1 つに対して，初めて検出される時刻と故障の影響がフリップフロップに存在する時刻を調べ記憶する．次にこの情報に基づいて，テスト系列を複数の部分系列に分割し，それらを 2 種類に分類する．消費電力を削減するためにテストベクトルを変更する際には，2 種類の部分系列に対して，異なる手法を用いて元の故障検出率を保証する．

以下では，テスト系列を部分系列に分割・分類するため，定義 1，2，3 で状態に関する用語を，定義 4，5，6 で，テストベクトルの種類を，定義 7，8 で部分系列の種類をそれぞれ定義する．提案法では，定義 7，8 に基づいて，与えられたテスト系列を分割・分類す

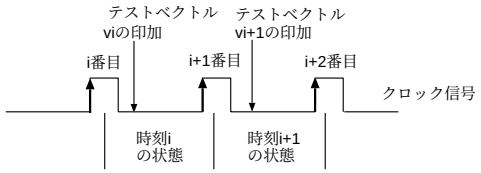

図 1 状態の定義
Fig. 1 Definition of state.

表 1 故障伝搬の例
Table 1 Example of fault propagation.

time	1	2	3	4	5	6	7	8
v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
s	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
fault f_1	P	D						
fault f_2		P	D					
fault f_3						P	P	D

P：故障の影響がフリップフロップの入力に伝搬
D：故障の影響が外部出力に伝搬

る．

［定義 1］ i 番目のクロックの立上りエッジから $i+1$ 番目のクロックの立上りエッジまでの間を**時刻** i と呼び，i 番目のクロックによって決定するフリップフロップの値の集合を時刻 i の**状態**と呼ぶ（**図 1** 参照）．

［定義 2］時刻 i に印加するテストベクトル v_i に対して，時刻 i の状態を v_i の**現状態**と呼ぶ（図 1 参照）．

［定義 3］時刻 i に印加するテストベクトル v_i が終端である部分系列 T_{sub} に対して，時刻 $i+1$ の状態を T_{sub} の**終状態**と呼ぶ（図 1 参照）．

［定義 4］時刻 i において故障の影響が初めて外部出力に伝搬する故障が少なくとも 1 つ存在するとき，時刻 i の入力ベクトルを**故障検出ベクトル**と呼ぶ．

［定義 5］次の条件をすべて満たす時刻 i のテストベクトルを**故障伝搬ベクトル**と呼ぶ．

条件 1：時刻 i のテストベクトルが故障検出ベクトルでない．

条件 2：$i < d$ の任意の時刻 d で初めて検出される故障の影響が，時刻 i から時刻 $d-1$ のすべての時刻において，少なくとも 1 つのフリップフロップに伝搬している．

［定義 6］故障検出ベクトルでも故障伝搬ベクトルでもないテストベクトルを**状態遷移ベクトル**と呼ぶ．

［定義 7］時刻 i から時刻 j のすべてのテストベクトルが状態遷移ベクトルのとき，部分系列 $v_i - v_{i+1} - \cdots - v_j$（$v_k$：時刻 k のテストベクトル）を**状態遷移系列**と呼ぶ．

［定義 8］時刻 i から時刻 j のすべてのテストベクトルが故障検出ベクトルまたは故障伝搬ベクトルのとき，部分系列 $v_i - v_{i+1} - \cdots - v_j$（$v_k$：時刻 k のテストベクトル）を**故障検出系列**と呼ぶ．

［**テスト系列の分割・分類の例**］

テスト系列 $v_1 - v_2 - v_3 - \cdots - v_8$ と時刻 1, 2, ..., 8 の状態 $s_1, s_2, \ldots, s_8$ が与えられ，故障 f_1，f_2，f_3 に対する故障の影響の伝搬が，**表 1** のようであったとする．ここで，time，v，s の各行は，時刻，テストベクトル，テストベクトル v_i の現状態をそれぞれ表す．また，fault f_1，fault f_2，fault f_3 の各行は，故障 f_1，f_2，f_3 の影響がフリップフロップの入力または外部出力に伝搬したかどうかを表し，‘P’ は故障の影響がフリップフロップの入力に，‘D’ は故障の影響が外部出力に伝搬したことを表す．このとき，時刻 2, 3, 8 のテストベクトル v_2，v_3，v_8 が故障検出ベクトル，時刻 1, 6, 7 のテストベクトル v_1，v_6，v_7 が故障伝搬ベクトル，時刻 4, 5 のテストベクトルが状態遷移ベクトルとなる．さらに，部分系列 $v_1 - v_2 - v_3$ と $v_6 - v_7 - v_8$ が故障検出系列，部分系列 $v_4 - v_5$ が状態遷移系列となる．

4. 状態遷移系列のテストベクトル変更法

テストベクトル変更は，外部入力値を 1 ビットずつ反転させることによって行う．変更を行った際，消費電力削減という点から次の 2 つの条件を調べる．

- 条件 1：どのテストベクトルについても変更後の遷移ゲート数が元の最大遷移ゲート数を超えない．
- 条件 2：1 つのテストベクトルの平均遷移ゲート数が元より減少する（提案アルゴリズム中では遷移ゲート数の和が減少する）．

また故障検出率が低下しないことを保証するためには，次の条件を調べる．

- 条件 3：状態遷移系列の終状態が元と同一である．

上記のすべての条件を満足する場合，変更した外部入力値を採用しその値に確定する．

ここで，故障検出率が低下しないことを完全に保証するならば，条件 3 にある終状態は，各故障に対して故障の影響の伝搬を考慮した状態を調べる必要があり，これを行うには故障シミュレーションが必要である．しかしながら，もし故障の影響がマスクされる（故障の影響の伝搬が打ち消しあう）ことが起こらないならば，故障の影響の伝搬について考慮する必要がなく，正常回路について条件 3 を調べるだけで十分となる．故障の影響のマスクについては，6 章の実験結果で示すように，現実には起こる可能性がきわめて低い．また，もしすべての故障について，故障シミュレーションを行い条件 3 を調べたならば，計算時間が非常に長

Algorithm: *Modification of test vectors in a state-transition subsequence*
/* T : a set including times when target test vectors are applied */
/* $T = \{t_{start}, t_{start+1}, ..., t_{end}\}$ */
/* t_{start} : time when the first test vector in the target subsequence is applied */
/* t_{end} : time when the last test vector in the target subsequence is applied */
/* t_L : time when the last test vector in the original test sequence is applied */
/* $tr\_gate(x)$: the number of transition gates in time x */
/* s_x : present state in time x */
/* max_tr_gate : the maximum number in $tr\_gate(x)$ */
/* PI_x : primary input at the x-th position */
/* v_x : test vector applied in time x */
1 : *while* ($T \neq \phi$)
2 : Select a time t among T such that $tr\_gate(t)$ is the maximum;
3 : $T = T - \{t\}$;
4 : $i = 1$; $n\_fail = 0$;
5 : *while* ($n\_fail \neq$ the number of primary inputs)
6 : Invert the value at PI_i of v_t
7 : Make the circuit state s_{t-1};
8 : Logic simulation with application of v_{t-1};
9 : *for* ($j = t; j \leq t_{end} + 1$ && $j \leq t_L; j++$)
10 : Capture values at input lines of flip-flops as a present state;
11 : Logic simulation with application of v_j;
12 : Calculate the number of transition gates and store it in $tr\_gate\_new(j)$;
13 : *if* ($tr\_gate\_new(j) > max\_tr\_gate$)
14 : *goto* FAILED;
15 : *else if* ($\sum_{k=t}^{j} tr\_gate\_new(k) < \sum_{k=t}^{j} tr\_gate(k)$ && the present state is equivalent to s_j)
16 : $n\_fail = 0$;
17 : Update s_k and $tr\_gate(k)$ for $t \leq k \leq j$;
18 : *goto* NEXT;
19 : **FAILED** :
20 : n_fail ++;
21 : Invert the value at PI_i in v_t;
22 : **NEXT** :
23 : i ++ ;
24 : *if* ($i >$ the number of primary inputs)
25 : $i = 1$;

図 2 状態遷移系列中のテストベクトルの変更アルゴリズム
Fig. 2 Algorithm to modify test vectors in a state-transition subsequence.

くなることが予想される．そこで提案法では，論理シミュレーションによって，正常回路についてのみ条件3を満足するかどうかを調べる．

図 2 に状態遷移系列中のテストベクトル変更アルゴリズムを示す．アルゴリズムが開始する時点では，以下の情報が既知であるとする．これらの情報は，テスト系列分割・分類のための故障シミュレーションにおいて得ることができる．

- T：変更の対象となる部分系列中のテストベクトルを印加する時刻の集合
- t_{start}：対象となる部分系列中の最初のテストベクトルを印加する時刻
- t_{end}：対象となる部分系列中の最後のテストベクトルを印加する時刻
- t_L：与えられたテスト系列中の最後のテストベクトルを印加する時刻
- $tr\_gate(x)$：時刻 x における遷移ゲート数
- s_x：時刻 x の状態
- max_tr_gate：$tr\_gate(x)$ の中での最大値
- PI_x：x 番目の外部入力線
- v_x：時刻 x で印加するテストベクトル

またアルゴリズム内部では，以下の変数が用いられる．

- t：変更の対象となるテストベクトルを印加する時刻
- i：値を変更する外部入力線の番号
- j：論理シミュレーションの対象となっている時刻
- n_fail：外部入力値の変更が失敗した回数
- $tr\_gate\_new(x)$：変更した部分系列の時刻 x の遷移ゲート数

図 2 のアルゴリズムを以下に説明する．ここでは，集合 T に含まれる時刻のテストベクトル 1 つ 1 つについて，順に処理を行う．処理するテストベクトルの順序は，2 行目に示されるように，遷移ゲート数が大きいものから順に行う．1 つのテストベクトルについての処理（5 行目から 25 行目）は，変数 n_fail が外部入力線数と同じになるまで続けられる．変数 n_fail は，外部入力値の変更が失敗したとき，インクリメントされ，成功したときリセットされる．すなわち，外部入力値の変更が外部入力線数と同数連続して失敗するまで繰り返される．論理シミュレーションは，まず，対象の部分系列の最初の時刻の 1 つ前の時刻（$t-1$）

について行われる（7，8行目）．これは時刻 t での遷移ゲート数を計算するために，時刻 $t-1$ の信号線の値が必要となるためである．その後，時刻 t から時刻 $t_{end}+1$（ただし t_L を超えない）について1時刻ずつ，論理シミュレーションを行い，そのときの遷移ゲート数を計算する（9行目から18行目）．9行目からのループ処理が終了する場合は次の3通りである．

(1) 時刻 j の遷移ゲート数が最大遷移ゲート数を超えた場合（13行目の条件）．このとき，この外部入力値の変更は失敗であったとして，19行目からの失敗した場合の処理に進む．

(2) 時刻 t から時刻 j の遷移ゲート数の和が，変更前より減少し，かつ時刻 j の状態が変更前の状態（s_j）と同一であった場合（15行目の条件）．このとき，外部入力値の変更は成功であったとして，*ループを抜ける*．ループを抜ける前には，失敗した回数を表す n_fail のリセット（16行目）と，時刻 $k(t \leq k \leq j)$ の状態 s_k と遷移ゲート数 $tr\_gate(k)$ の更新（17行目）を行う．

(3) 時刻 $t_{end}+1$ まで論理シミュレーションを行った場合（9行目の条件）．このとき15行目の条件を1度も満足しなかったことから，この外部入力値の変更は失敗であったとして19行目からの処理に進む．

外部入力値の変更が失敗のときの処理としては，失敗回数（変数 n_fail）のインクリメント（20行目）と外部入力値を元に戻す（21行目）操作を行う．変数 i は，変更する外部入力線の番号を示すが，これをインクリメントすることで，変更する外部入力線を順に変更する（23行目）．ただし，変更する外部入力線番号が外部入力線数を超えた場合に，それを1に戻す操作を行う（24，25行目）．

［**状態遷移系列のテストベクトル変更の例**］

表2 に示されるようなテスト系列が与えられたとする．v，s，tr_gate，$type$ の各行は，各時刻でのテストベクトル，テストベクトル v_i の現状態，遷移ゲート数，テストベクトルの種類を表す．この系列中の時刻4から時刻7の状態遷移系列 $v_4-v_5-v_6-v_7$ に，図2のアルゴリズムを適用した場合について説明する．まず対象となるテストベクトルを印加する時刻の集合 $T=\{4,5,6,7\}$ を得る．次に T の中から，$tr\_gate(t)$ が最大である，$t=5$ を選択する（2行目）．次に，集合 T から $\{5\}$ を除外し（3行目），テストベクトル v_5 の1番目の外部入力値を反転させる（6行目）．その後，回路の状態を s_4 とし，v_4 を印加して論理シミュレーションを行う（7，8行目）．9行目からの *for* ループでは，時刻5から8について，v_5，v_6，v_7，v_8 を順に印加し，論理シミュレーションを行う．たとえば $j=6$ のときの処理について説明する．まずフリップフロップの入力線の値をフリップフロップに取り込んだ後（10行目）．時刻6のテストベクトル v_6 を印加し論理シミュレーションを行い（11行目），その時の遷移ゲート数を $tr\_gate\_new(6)$ に記憶する（12行目）．13行目では，遷移ゲート数 $tr\_gate\_new(6)$ が最大遷移ゲート数 $max\_tr\_gate=15$（時刻3の遷移ゲート数）より大きいかどうか調べ，大きい場合には，この変更は失敗であったとして，*for* ループを抜ける．また，15行目では，時刻5から時刻6の遷移ゲート数の和が変更前より減少し，かつ時刻6の状態が変更前の状態（s_6）と同一かどうかを調べる．もし両方の条件を満足すれば，この変更は成功であったとし，失敗回数 n_fail のリセットと，時刻 t〜j の状態と遷移ゲート数の更新（16，17行目）を行い，22行目に進む．もし15行目の条件を満たさない場合，10行目からの $j=7$ の処理に進む．変更が失敗であった場合，失敗回数 n_fail のインクリメント（20行目）と，1番目の外部入力値を元に戻す（21行目）操作を行う．その後，外部入力線の番号を2にした後（23行目），6行目に戻り，2番目の外部入力線の値を変更し，先と同様の操作を行う．外部入力線の変更が外部入力線数と同数回連続して失敗した場合，5行目からの *while* ループを終了する．その後再び，T の中から遷移ゲート数が最大である時刻を選択し，3行目以降の処理を続ける．

表2 テスト系列の例
Table 2 Example of a test sequence.

time	1	2	3	4	5	6	7	8	9
v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
s	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
tr_gate	-	8	15	4	11	9	7	8	3
type	P	D	D	S	S	S	S	P	D

D：故障検出ベクトル，P：故障伝搬ベクトル，S：状態遷移ベクトル

5. 故障検出系列のテストベクトル変更法

故障検出系列中のテストベクトル変更についても，状態遷移系列中のテストベクトルに対する手法と同様に，外部入力値を1ビットずつ変更し，最大遷移ゲート数を増加させずに，平均遷移ゲート数が減少するかどうかを調べる．状態遷移系列に対する手法と異なる点は，故障検出率を保証するために故障シミュレーションを用いることである．対象となる故障検出系列で検出される故障が，同じ部分系列内ですべて検出さ

Function: *Fault simulation for a fault-detecting subsequence*
1 : Collect faults that are detected at t_{start} to t_{end} for the first time, and include them in F;
2 : *if* (the cardinality of F is larger than MAX_FAULT)
3 : Return $FAIL$;
4 : Make the circuit state $s_{t_{start}}$;
5 : *for* ($j = t_{start}$; $j \leq t_{end}$; $j++$)
6 : Fault simulation with application of test vector v_j ;
7 : Drop detected faults from F ;
8 : *if* ($F == \phi$)
9 : Return $SUCCESS$;
10 : Return $FAIL$;

図 3 状態検出系列に対する故障シミュレーション
Fig. 3 Fault simulation for a fault-detecting subsequence.

れない場合には，外部入力値の変更は失敗であったとする．また，対象の部分系列以降の部分系列の故障検出を保証するためには，対象の部分系列の終状態が元と一致することを確認する．

故障検出系列中のテストベクトル変更するアルゴリズムは，図 2 の 15 行目と 16 行目の間に次の処理を追加したものである．

if (FAIL == fault_simulation())
goto FAILED;

ここで呼び出されている関数 fault_simulation() は，**図 3** に示される．この関数では，変更の対象となる故障検出系列で初めて検出される故障を対象に，それらが，変更後の部分系列で検出されるかどうかを調べる．1 行目に示されるように，変数 F は，時刻 t_{start} から t_{end} で初めて検出される故障を含む集合である．この集合に含まれるすべての故障が検出された場合，「成功」を表す $SUCCESS$ を返す．そうでない場合，「失敗」を表す $FAIL$ を返す．変数 MAX_FAULT は，F に含まれる故障数の上限を表している．もし 1 つの故障検出系列で検出される故障数が MAX_FAULT を超える場合には，無条件に「失敗」を表す $FAIL$ を返す（3 行目）ことによって，大規模回路における計算時間の増大を防いでいる．

6. 実験結果

提案法を C 言語を用いてプログラム化し，コンピュータ（Pentium III 800MHz）上において，ISCAS'89 ベンチマーク回路に対して行った実験の結果を示す．元の系列として，HITEC[7] によって生成されたテスト系列を用いた．

表 3 に，元のテスト系列中のテストベクトルを分類した結果を示す．表中には左から順に，回路名（circuit），総テストベクトル数（total），故障検出ベクトル数（D），故障伝搬ベクトル数（P），故障検出ベクトルと故障伝搬ベクトルの和（D+P），状態遷移ベクトル数（S），状態遷移ベクトル数の総テストベクトル数に対する割合（rate）を示す．

次に**表 4** に，平均遷移ゲート数と計算時間に関する結果を示す．この実験における検出故障系列の故障数の上限（変数 MAX_FAULT の値）は 1,000 とした．表中では左の列から順に，回路名（circuit），元の系列での平均遷移ゲート数（original），状態遷移系列を変更した結果減少した平均遷移ゲート数（state-seq），故障検出系列を変更した結果減少した平均遷移ゲート数（detect-seq），減少した平均遷移ゲート数の総数（sum），“original” の列の結果に対する “sum” の列の結果の割合（rate），計算時間（cpu）を示す．実験の結果，最大で約 35%（s1423），最小で約 4%（s382）遷移ゲート数が減少した．また計算時間は最大で約 4,200 秒（s35932）であった．s1423 の状態遷移系列を変更した結果が 0 であるのは，表 3 より s1423 では状態遷移系列が 0 であったためである．また，s1196，s1238 で状態遷移系列を変更した結果の減少が少ないのも，表 3 より，状態遷移系列が比較的少なかったためと分かる．

故障の影響のマスクについて，現実に起こっているかどうかを調べるために，得られたテスト系列について故障シミュレーションを行った．その結果，すべての回路において，提案法で得られたテスト系列によって，元のテスト系列と同数の故障が検出されており，故障の影響のマスクによって故障検出率の低下が起こっていないことが確認できた．

次に最大遷移ゲート数についての結果を**表 5** に示す．表中には，左から順に，回路名（circuit），元の系列での最大遷移ゲート数（original），状態遷移系列を変更した結果減少した最大遷移ゲート数（state-seq），故障検出系列を変更した結果減少した最大遷移ゲート数（detect-seq），減少した最大遷移ゲート数の総数（sum），“original” の列の結果に対する “sum” の列の結果の割合を示す．実験の結果，最大で約 23%（s713）減少した．しかしながら，いくつかの回路ではまったく減少しなかった．

表 3 テストベクトルの分類
Table 3 Classification of test vectors.

circuit	total	D	P	D+P	S	rate(%)
s344	127	43	25	68	59	46.5
s349	134	46	25	71	63	47.0
s382	2074	52	271	323	1751	84.4
s386	286	93	28	121	165	57.7
s400	2214	55	423	478	1736	78.4
s420	166	31	46	77	89	53.6
s444	2240	57	264	321	1919	85.7
s526	2258	47	1365	1412	846	37.5
s641	209	94	31	125	84	40.2
s713	173	87	22	109	64	37.0
s820	1115	189	88	277	838	75.2
s832	1137	192	108	300	837	73.6
s1196	435	246	69	315	120	27.6
s1238	475	269	109	378	97	20.4
s1423	150	67	83	150	0	0
s1488	1170	187	61	248	922	78.8
s1494	1245	186	63	249	996	80.0
s5378	912	199	557	756	156	17.1
s35932	496	182	150	332	164	33.1

D：故障検出ベクトル，P：故障伝搬ベクトル，S：状態遷移ベクトル

表 4 平均遷移ゲート数
Table 4 Average number of transition gates.

circuit	original	state-seq	detect-seq	sum	rate(%)	cpu(s)
s344	44.17	3.39	3.18	6.57	14.9	0.23
s349	42.74	4.06	3.15	7.21	16.9	0.30
s382	17.61	0.19	0.46	0.65	3.7	17.40
s386	44.85	5.62	2.33	7.95	17.7	0.32
s400	17.62	0.22	0.58	0.80	4.5	24.50
s420	24.58	2.03	1.81	3.84	15.6	0.60
s444	22.03	0.39	0.67	1.06	4.8	14.51
s641	90.62	10.12	19.32	29.44	32.5	4.17
s713	94.38	7.86	17.76	25.62	27.1	3.54
s820	66.89	13.63	3.53	17.16	25.7	7.46
s832	67.23	13.02	3.64	16.66	24.8	6.68
s1196	137.36	0.28	24.59	24.87	18.1	7.59
s1238	127.35	0.12	31.42	31.54	24.8	62.77
s1423	170.95	0	59.23	59.23	34.6	103.50
s1488	222.99	18.72	3.43	22.15	9.9	7.26
s1494	221.02	18.48	2.78	21.26	9.6	8.18
s5378	555.70	39.46	0.22	39.68	7.1	81.53
s35932	5705.35	117.23	324.17	441.40	7.7	4176.86

表 5 最大遷移ゲート数
Table 5 Maximum number of transition gates.

circuit	original	state-seq	detect-seq	sum	rate(%)
s344	91	0	13	13	14.3
s349	80	0	0	0	0
s382	69	0	0	0	0
s386	73	0	7	7	9.6
s400	73	0	5	5	6.8
s420	40	0	0	0	0
s444	76	0	1	1	1.3
s641	165	9	8	17	10.3
s713	188	30	13	43	22.9
s820	135	1	17	18	13.3
s832	127	5	0	5	3.9
s1196	209	0	1	1	0.5
s1238	185	0	2	2	1.1
s1423	295	0	13	13	4.4
s1488	307	20	8	28	9.1
s1494	296	14	8	22	7.4
s5378	859	0	0	0	0
s35932	9416	0	0	0	0

表 6 検出故障数の影響
Table 6 Effect of the limit of the number of detected faults.

circuit	MAX_FAULT	original	state-seq	detect-seq	sum	rate(%)	cpu(s)
s35932	1000	5705.35	117.23	324.17	441.40	7.7	4176.86
s35932	10000	5705.35	117.23	434.70	551.93	9.7	7346.40

表 6 には，s35932 回路において，故障検出系列が検出する故障数の上限（変数 MAX_FAUL の値）を変えた場合の結果について示す．左から 2 列目が故障数の上限を表している．故障数の上限が 1,000 の場合の結果は，表 4 と同一である．実験の結果では，故障数の上限を 10,000 とすることで平均遷移ゲート数が約 2%さらに減少したが，計算時間が約 1.8 倍に増大した．また，最大遷移ゲート数については，故障数の上限を 10,000 としても結果に変化はなかった．

7. おわりに

本論文では，CMOS 順序回路のテスト時の消費電力を削減するためのテストベクトル変更法を提案した．提案法では，まず与えられたテスト系列を部分系列に分割し，それらを状態遷移系列と故障検出系列に分類した．テストベクトル変更においては，外部入力値を 1 ビットずつ変更する手法を用い，最大遷移ゲート数を増加させずに，平均遷移ゲート数が減少する場合のみ，外部入力値を変更した．また，元の故障検出率を保証するために，状態遷移系列に対しては論理シミュレーションを行い，故障検出系列に対しては故障シミュレーションを行った．これらのシミュレーションにおいては，すべての故障についてその影響の伝搬を考慮することは行っていないので，故障の影響のマスクが起こるような場合には，故障検出率が低下する場合がある．しかしながら，実験の結果においては，すべての回路について故障の影響のマスクによる故障検出率の低下は起こっていないことが確認された．また，消費電力に関する実験の結果では，平均遷移ゲート数を最大で約 35%，最大遷移ゲート数を最大で約 23%削減することができた．

提案法では，平均消費電力と最大消費電力の両方を削減することを目指したが，平均消費電力または最大消費電力のどちらか一方を，重点的に削減する要求も考えられる．そのような要求に対しても，提案法を拡張することで対応可能であると考えられ，これについては今後，有効な手法を開発する予定である．また，外部入力値を複数ビット同時に変更する手法や，複数のテストベクトルを同時に変更する手法，さらには，縮退故障以外の故障についても元の故障検出率を保証するような，テストベクトル変更法を開発することが，今後の課題である．

参考文献

1) 張　凱，篠木　剛，林　照峰：組合せ回路における同時スイッチングゲート数の上界評価の一手法，信学会論文誌 A, Vol.J83-A, pp.387–395 (2000).
2) 上田祐彰，樹下行三：遺伝的アルゴリズムによる CMOS 論理回路の最大変化ゲート数の評価，信学会論文誌 D-I，Vol.J78-D-I, pp.367–375 (1995).
3) Corno, F., Prinetto, P., Rebaudengo, M. and Reorda, M.S.: A Test Pattern Generation Methodology for Low Power Consumption, *Proc. VLSI Test Symp.*, pp.453–457 (1999).
4) Dabholkar, V., Chakravarty, S., Pomeranz, I. and Reddy, S.M.: Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application, *IEEE Trans. Computer-Aided Design*, Vol.17, pp.1325–1333 (1998).
5) Gerstendorfer, S. and Wunderlich, H.-J.: Minimized Power Consumption for Scan-based BIST, *Proc. Int. Test Conf.*, pp.77–84 (1999).
6) Girard, P., Guiller, L., Landrault, C. and Pravossoudovitch, S.: A Test Vector Inhibiting Technique for Low Energy BIST Design, *Proc. VLSI Test Symp.*, pp.407–412 (Apr. 1999).
7) Niermann, T.M. and Patel, J.H.: HITEC: A Test Generation Package for Sequential Circuits, *Proc. European Conf. on Design Automation*, pp.214–218 (Feb. 1991).
8) Sankaralingam, R., Oruganti, R.R. and Touba, N.A.: Static Compaction Techniques to Control Scan Vector Power Dissipation, *Proc. VLSI Test Symp.*, pp.35–40 (Apr. 2000).
9) Vai, M.M.: *VLSI Design*, CRC Press (2001).
10) Wang, S. and Gupta, S.: LT-RTPG: A New Test-Per-Scan BIT TPG for Low Heat Dissipation, *Proc. Int. Test Conf.*, pp.85–94 (1999).
11) Wang, S. and Gupta, S.K.: ATPG for Heat Dissipation Minimization during Test Application, *Proc. Int. Test Conf.*, pp.250–258 (Oct. 1994).
12) Wang, S. and Gupta, S.K.: ATPG for Heat Dissipation Minimization for Scan Testing, *Proc. Design Automation Conf.*, pp.614–619 (1997).
13) Wang, S. and Gupta, S.K.: DS-LFSR: A New BIST TPG for Low Heat Dissipation, *Proc. Int. Test Conf.*, pp.848–857 (Nov. 1997).

(平成 14 年 9 月 13 日受付)
(平成 14 年 3 月 14 日採録)

樋上　喜信（正会員）

平成 8 年大阪大学大学院工学研究科応用物理学専攻博士後期課程修了．同年日本学術振興会特別研究員採用．平成 10 年より愛媛大学工学部助手．現在同学部講師．論理回路に対するテスト生成およびテスト容易化設計に関する研究に従事．電子情報通信学会，IEEE 各会員．博士（工学）．

小林　真也（正会員）

昭和 60 年大阪大学工学部情報工学科卒業．平成 3 年大阪大学大学院博士課程修了．工学博士．同年金沢大学工学部電気・情報工学科助手．その後，同講師，助教授，同大学大学院自然科学研究科助教授を経て，平成 11 年愛媛大学工学部情報工学科助教授．その間平成 8 年にカリフォルニア大学アーバイン校客員教官．並列処理システム，分散処理システムの研究に従事．電子情報通信学会，日本ソフトウェア科学会，電気学会，IEEE，ACM 各会員．訳書「計算機設計技法」（トッパン），著書「基礎から学ぶ UNIX ワークステーショ ン」（トッパン）．

高松　雄三（正会員）

昭和 41 年愛媛大学工学部電気工学科卒業．佐賀大学理工学部電子工学科助教授を経て，昭和 62 年 10 月より愛媛大学工学部情報工学科教授，現在に至る．論理回路の診断・テストに関する研究に従事．工学博士．著書「論理設計入門」（共著，日新出版）等．電子情報通信学会正員，IEEE Senior Member．