ドントケア故障シミュレーションを用いた
組合せ回路の動的テスト圧縮に関する研究

日大生産工（学部）○秋山 祐介　　日大生産工　細川 利典
日大生産工（院）　大森 悠翔　　明大　山崎 浩二

1. はじめに

近年，LSI の大規模化に伴い，それにかかるテストコストの増大が問題となっている．テストコストとテストパターン数は比例関係にあるためテストパターン数を削減することにより，テストコストの削減が期待できる．

本稿では従来のバッファを用いる動的圧縮の手法[1]にドントケアを含んだままのテストパターンに対する故障シミュレーションを実装し，テスト生成の対象となる故障数を削減することにより更なる総テストパターン数の削減を試みる．

2. テスト圧縮について

テストパターンを圧縮する手法には、テストパターンを生成しながら圧縮していく動的圧縮[2]と，テストパターン生成後に圧縮する静的圧縮[2]の 2 種類がある．

動的圧縮はテスト生成に時間がかかるが圧縮の効果は大きく，静的圧縮は動的に比べて時間はかからないが圧縮の効果が小さい．

(定義：圧縮可能)

外部入力(PI_1 , PI_2 , ･･･ , PI_n)を持つ組合せ回路に対する 2 つのテストパターンT_1，T_2を考える（nは外部入力数）．T_1，T_2の外部入力PI_iの値をそれぞれ$T_1(PI_i)$ ，$T_2(PI_i)$と表記する（$1 \leqq i \leqq n$）．$T_1(PI_i)$ ，$T_2(PI_i) \in \{0, 1, X\}$である．

表 1 に示す圧縮演算$\cap_T$を用いると，圧縮演算結果Tは任意のi($1 \leqq i \leqq n$)について式 1 で表すことができる．

$$T(PI_i) = T_1(PI_i) \cap_T T_2(PI_i) \quad (式\ 1)$$

任意i($1 \leqq i \leqq n$)について，$T(PI_i) \in \{0,1,X\}$となるときT1 とT2 は圧縮可能であるとし，このときの演算結果Tを圧縮したテストパターンという．また$T(PI_i) = \phi$ が 1 つ以上あるとき，T_1とT_2は圧縮不可能であるとする．

例として以下の外部入力数 3 であるテストパターン T 1，T2，T3 に対して考える．

$T_1 = (0,1,X)$
$T_2 = (1,X,1)$
$T_3 = (X,1,0)$

圧縮の可否は各々のテストパターンの$T(PI_i)$について表 1 の圧縮演算$\cap_T$を行い，演算結果にϕが含まれなければ圧縮が可能である．

例に対する演算結果は以下の通りになる．

$T_1 \cap_T T_2 = (\phi, 1, 1)$
$T_1 \cap_T T_3 = (0, 1, 0)$
$T_2 \cap_T T_3 = (1, 1, \phi)$

以上の結果より，T_1とT_3のみ演算の結果にϕを含まなかったので圧縮可能となることがわかる．

表 1．圧縮演算規則表

$\cap_T$	0	1	X
0	0	ϕ	0
1	ϕ	1	1
X	0	1	X

A Method of Dynamic Test Compaction using Don't Care Fault Simulation Technique

Yusuke AKIYAMA, Toshinori HOSOKAWA, Yusyo Omori
and Koji YAMAZAKI

3. ドントケア故障シミュレーション

（定義：ドントケア故障シミュレーション）

0,1,X の 3 値から構成されるテストパターンに対して，故障シミュレーションを実行し，該当するテストパターンで検出が可能である故障の情報を得る操作をドントケア故障シミュレーションとする．

圧縮バッファを用いた動的圧縮は，未検出故障を 1，2 個程度しか検出できないテストパターンを生成してしまうことが頻繁に起こりうる．これはテスト生成の対象となる故障が多数であることに起因する．

例えば，初期に故障f1，f4，f6 を検出できるテストパターンT_Aが生成され，圧縮バッファに格納されたとする．従来の手法では故障シミュレーションを実行して故障テーブルを更新する機会はそのテストパターンが全体のテストセットに加えられるときだけである．よってTaがバッファ内にある時点で，f4 やf6 を検出するテストパターンを生成してしまい，結果としてT_Aはf1 しか検出できない効果の薄いテストパターンとなってしまう恐れがある．

生成されたばかりのテストパターンや，バッファ内で圧縮したテストパターンを対象にドントケア故障シミュレーションを実行することで，つまりは従来の手法に比べて早く，適時に各々のテストパターンで検出できる故障の状況を把握することでこのような状況は回避することが可能である．これについて次章の 4.1 にて詳細を記す．

4. テスト圧縮を含んだテスト生成アルゴリズム

4.1 故障テーブル

故障シミュレーションを実装する際，故障テーブルを用意する．故障テーブルは故障の検出状況を把握し，テスト生成の対象となる故障を選択するためのテーブルである．

従来の故障テーブルは各々の故障に対して検出フラグを持ったテーブルである．検出フラグとは故障が検出されたかを判断するためのフラグであり初期の値は 0 が入っている．テストパターンに故障シミュレーションを実行した際，そのテストパターンによって検出が可能である故障に該当するフラグの値を 0 から 1 に更新する．値が 1 の故障は，未検出故障ではないと判断することができるので以降のテスト生成の対象から外すことが可能となる（図 1）．

本手法では新たに ATPG フラグ（テスト生成フラグ）をテーブルに追加する．

ATPG フラグも検出フラグと同様に初期の値は 0 を入れておき，ドントケア故障シミュレーションを実行することによりドントケアを含んだままでも検出が可能な故障に対して 0 から 1 に値を更新することでテスト生成の必要性を判断する．

先ほどの例ではドントケア故障シミュレーションを用いることで，テストパターン生成直後に f1,f4,f6 のATPGフラグには1が入り，f4，f6 に対するテスト生成を回避することが可能となる（図2）．

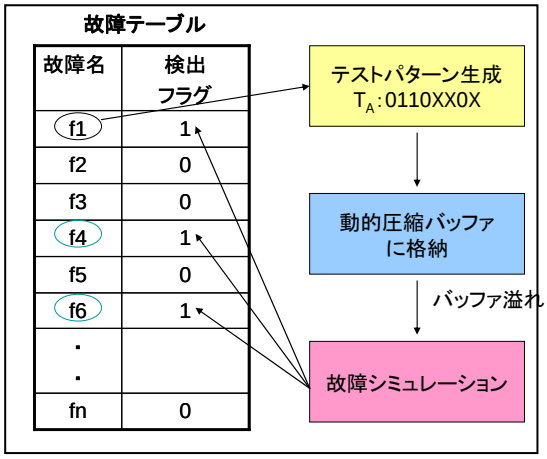

図1. 故障テーブルの更新（従来法）

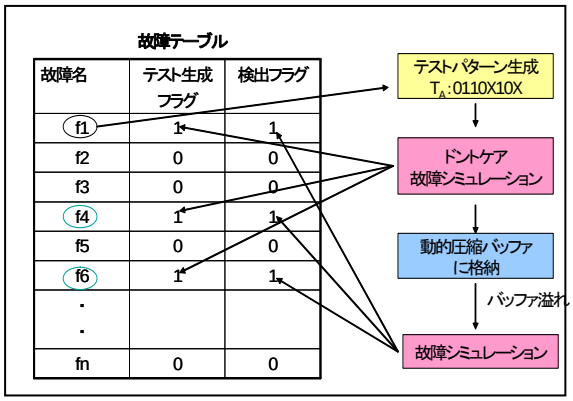

図2. 故障テーブルの更新（本手法）

4.2 テスト生成アルゴリズム

テスト生成のアルゴリズムを以下に示す．

まず故障数が tab_size である故障テーブル f_table 中の ATPG フラグが未チェックである先頭の故障 f に対してテストパターンが生成される．生成されたパターン t にドントケア故障シミュレーションを実行して，このパターンで同時に検出できるすべての故障の ATPG フラグを１にする．次に生成されたパターンを動的圧縮バッファ buf に格納する．

もしバッファ内の既存のパターン buf[i]と先ほど生成されたパターンの圧縮が可能なら圧縮する（図３）．圧縮されたパターンに対してドントケア故障シミュレーションを実行し，ドントケア数で降順にバッファ内のパターンがソートされる．もし圧縮されない場合は，バッファ内に新しくパターンが格納される（図４）．

また，バッファ内のパターン数がバッファサイズ buf_size を上回った場合はバッファが溢れたとして，パターンをドントケア数で降順にソートした後，溢れたパターンのドントケアをすべてランダムで１か０のどちらかに埋め，故障シミュレーションを実行する．これにより，そのパターンで検出可能な故障の検出フラグに１が入る．

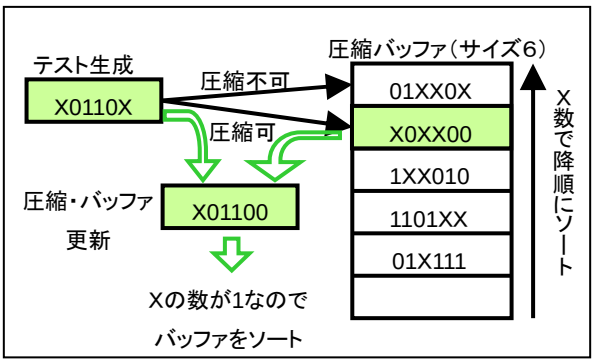

図３．動的圧縮アルゴリズム例１

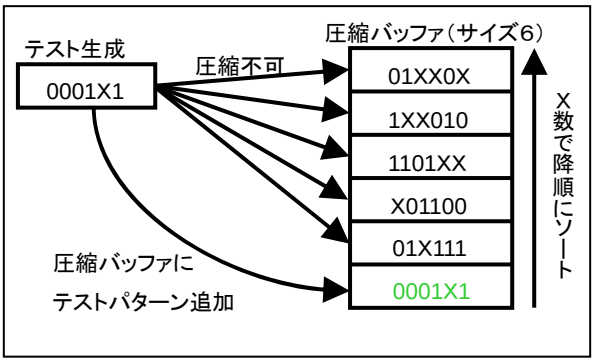

図４．動的圧縮アルゴリズム例２

```
1  ATPG_ALG(f_tab, tab_size){
2    for( i=0; i<tab_size; i++){
3      if(故障fiのATPGフラグ＝0){
4        故障fiに対してテストパターンti生成
5        DY_COMP(ti);
6      }
7    }
8    for(i=0; i<bus_size; i++){
9      buf[i]に故障シミュレーション
10   }
11 }
12 DY_COMP(ti){
13   for(i=0; i<buf_size; i++){
14     if( tiとbuf[i]が圧縮可能){
15       buf[i]=tiとbuf[i]を圧縮したテストパターン
16       buf[i]にドントケア故障シミュレーション
17       X数でbufをソート
18     }
19     else{
20       tiをbufに追加
21       tiにドントケア故障シミュレーション
22       X数でbufをソート
23       if(バッファ溢れ){
24         溢れたテストパターンに故障シミュレーション
25         X数でbufをソート
26       }
27     }
28   }
29 }
```

図５．テスト生成アルゴリズム擬似コード

これらの一連の処理をすべての未検出故障に対して実行したらテスト生成を終了とする．最後にバッファ内に残ったパターンすべてに故障シミュレーションを実行する．この時，未検出故障を検出できないテストパターンは破棄される．

テスト生成アルゴリズムの擬似コードを図５に示す．

5. 実験結果

実験は ISCAS’85 ベンチマーク回路に対して行った．動的圧縮を用いないケースを OFF，ドントケア故障シミュレーションを実装していない従来の手法を prev，本手法を ON として結果を比較した．結果は以下の表２～４の通りである．また prev と ON の OFF に対するテストパターン数の削減率を表５に示す．

表２．総テストパターン数

circuit	OFF	prev	ON
c880	87	74	68
c1355	116	96	107
c1908	154	149	146
c2670	143	153	135
c3540	190	194	168
c5315	180	176	160
c6288	33	32	32
c7552	276	335	241

表３．テスト生成回数

circuit	OFF	prev	ON
c880	87	217	154
c1355	116	629	127
c1908	154	472	203
c2670	143	775	303
c3540	190	1301	488
c5315	180	1685	591
c6288	33	51	33
c7552	276	1971	613

表４．CPU 時間（秒）

circuit	OFF	prev	ON
c880	1.27	2.16	2.22
c1355	15.41	17.19	17.88
c1908	8.70	11.66	15.83
c2670	22.38	34.48	34.38
c3540	43.20	70.59	78.64
c5315	50.25	80.52	105.81
c6288	27.56	30.05	31.14
c7552	215.17	321.22	385.11

表５．テストパターン削減率(%)

circuit	prev	ON
c880	85	78
c1355	83	92
c1908	97	95
c2670	107	94
c3540	102	88
c5315	98	89
c6288	96	96
c7552	121	87

表６．故障検出数の少ないパターン数

	prep				ON			
	det0	det1	det2	det3	det0	det1	det2	det3
c880	22	18	10	5	11	12	9	10
c1355	135	3	28	36	20	1	18	45
c1908	63	44	51	19	23	38	31	15
c2670	77	20	34	16	26	19	33	7
c3540	156	41	31	18	42	25	16	16
c5315	202	57	20	21	38	36	18	14
c6288	9	0	2	0	0	0	2	0
c7552	320	98	42	21	91	55	35	16

c1355 と c6288 を除く全ての回路で総テストパターン数の減少に成功していることがわかる. c6288 は回路の特性上、総テストパターン数に変化がなかったものと推測される．c1355 のみが prev に対してテストパターン数が多くなってしまったが，原因確認は現時点では難しい.

テスト生成回数は今回の手法を用いることで prev に対して大幅な削減に成功した. また，検出可能な故障が０～３の少数であるテストパターンが顕著に削減されていることがわかった（表６）.

CPU 時間は多くの回路で on は prev に対して増加している．これはドントケア故障シミュレーションを実行しているため処理の時間が付加されたためだと考えられる.

テストパターンの削減率を考察する．prev においては OFF よりテストパターン数が増加してしまうケースが存在する．これは動的圧縮バッファ内に格納されたまま，結果的に検出できる故障が少なくなったテストパターンが多数生成されてしまったからだと考えられる．これに対して ON では全ての回路でテストパターン数削減に成功した.

6. おわりに

本稿では，バッファを用いる動的圧縮アルゴリズムの提案と，一部の回路に対しての評価を行った.

本手法の実装により，テスト生成回数および，少数の未検出故障しか検出ができないテストパターンの生成回数を削減することに成功した.

今後の課題としてバッファ内の解析をすることで，更なる圧縮の効果の向上を目指したい.

参考文献

[1]日本大学生産工学部数理情報工学科 2005 年度卒業研究，飯野友里奈 “組合せ回路の動的パターン圧縮アルゴリズム”

[2]Niraj jha and Sadeep Gupta,
“Testing of Digital Systems”,
Cambridge university Press,2002.