極大圧縮法を用いた動的テスト圧縮法に関する研究

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

1. はじめに

近年，半導体微細化技術の進歩に伴い，大規模集積回路（Large Scale Integration: LSI）の規模が増大しており，それに伴いテストコストの増大が問題となっている[1]．テストコストはテストパターン数と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．

小規模の回路では，ほぼ最小のテスト集合を得るアルゴリズム[2]が提案されている．しかしながら，大規模回路においては計算量が多く，現実的な計算時間での適用は困難である．そこで大規模回路に適応可能な手法として，ドントケア故障シミュレーションとドントケアに基づく動的テスト圧縮を組み合わせた圧縮手法[3]が提案されている．本稿では，極大圧縮法[4]を用いて，テスト生成によって生成されたテストパターン中のドントケア率を増加させ，テストパターン圧縮の向上を目指す．さらに本稿では，動的テスト圧縮アルゴリズムに極小解圧縮[5]を適用する．一般にテストパターンのドントケアに基づく静的圧縮における貪欲解と厳密解がほぼ一致することが報告されている[4]．動的テスト圧縮を行う時に，いくらかのテストパターンをバッファに保存し，極小解圧縮を適用することで，テストパターン圧縮の更なる向上を図る．

2. ドントケア故障シミュレーションを用いた動的テスト圧縮

2.1 故障テーブル

図 1 に文献[3]で提案されている動的テスト圧縮とテスト生成で用いる故障表を示す．故障表は各障に対してテスト生成フラグと検出フラグを備えている．故障表から未検出故障を 1 つ取り出し，それに対するテストパターンの生成を試みる．また，一度テスト生成を試みた故障に対しては再度テスト生成されることはないものとする．

テスト生成フラグは，ある故障に対してテスト生成を試みたか否かを示すフラグである．未検出故障であってもテスト生成フラグがチェックされていると，その故障に対してテスト生成は行われない．一方，検出フラグとは故障が検出されたか否かを判断するためのフラグであり，生成されたテストパターンで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される故障の検出フラグにチェックをする．検出フラグにチェックがついた故障は，未検出故障ではないと判断し，以降のテスト生成や故障シミュレーションの対象から外す．

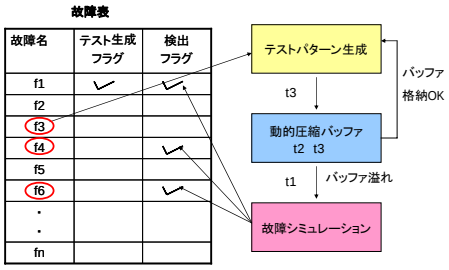

図 1.故障表の更新

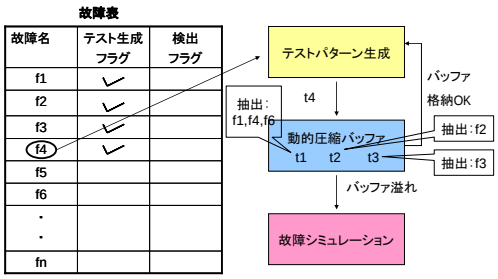

図 2.故障表の問題点

図 1 において，まず未検出故障 $f1$ に対してテストパターン $t1$ が生成される．$t1$ を動的圧縮バッファ[2]に格納させる時に，動的圧縮バッファが他のテストパターンで満たされていて，$t1$ が動的圧縮バッファに格納されずに溢れたとき，$t1$ で故障シミュレーションを実行する．ここで未検出故障 $f1$，$f4$，$f6$ を検出できたとす

On the Improvement of a Dynamic Test Compaction Method Using Maximal Compaction Technique

Tatsuya YAMAZAKI, Yusuke AKIYAMA
Toshinori HOSOKAWA, and Koji YAMAZAKI

ると，この情報からその検出フラグをチェックし，故障テーブルを更新する．

図 2 において，$t1$ は故障 $f1$，$f4$，$f6$ を検出できるがバッファに格納されたままでは $f4$ や $f6$ に対するテストパターンが生成されてしまうことがありうる．このことを避けるために，文献[3]では，ドントケア故障シミュレーションが提案された．

2.2 ドントケア故障シミュレーション

テストパターンが生成され，動的圧縮バッファに格納された時点で，そのテストパターン(ドントケアを含む)を用いてドントケア故障シミュレーション[3]を実行する．ドントケア故障シミュレーションを実行後，ドントケアを含んだテストパターンでも検出が可能な故障のテスト生成フラグをチェックする．

ドントケア故障シミュレーションを実行することでテストパターンが検出できる故障の情報を得ることが可能となり，結果的に生成されたテストパターンが動的圧縮バッファ内に格納されたままでも，そのテストパターンで検出が可能な未検出故障に対するテスト生成を回避できる．この手法によりテスト生成回数を削減し，テスト生成時間を高速化し，かつテストパターンの圧縮効率を高める．図 3 おいて，$t1$ が圧縮バッファに格納された時点でドントケア故障シミュレーションを実行することで $f4$，$f6$ のテスト生成フラグがチェックされ，t1 がバッファに格納されたままでも $f4$，$f6$ に対するテスト生成は行われない．

3. 極大圧縮法

極大圧縮法[4]とは，自動テストパターン生成(Auto Test Pattern Generatior:ATPG)が生成したテストパターンに対して目標とする故障の検出に必要のない割り当てを，ドントケアに変更するアルゴリズムである．これによりテストパターン中のドントケア数を増加させることで，圧縮効率の増加が期待できる．

図 4 に極大圧縮法アルゴリズムの擬似コードを示す．まず極大圧縮法の対象となるテストパターンの論理値を 1 ビットずつ反転させ，その 1 ビット反転したテストパターンそれぞれに対し故障シミュレーションを実行する．もし，その 1 ビット反転したテストパターンで目標故障の検出が可能ならば，入力反転テーブルの反転フラグをチェックする．入力反転テーブルについては後述する．入力反転テーブルで反転フラグがチェックされたビットの論理値を全て反転させたテストパターンで故障シミュレーションを実行する．そして，そのテストパターンで目標故障の検出が可能ならば，対象のテストパターンを入力反転テーブルで更新されたビットの論理値を全てドントケアに変換する．目標故障の検出ができなければ，テストパターンを変換しない．

図 5 に入力反転テーブルの例を示す．反転フラグがチェックされている外部入力は，その論理値を反転させても，目標故障の検出が可能であることを示す．図 5 の例では，目標故障を検出できるテストパターン(a,b,c,d,e)=(1,1,1,1,1)が生成されたとき，テストパターン(1,1,1,0,1)と(1,1,1,1,0)で，目標故障が検出可能であるということを示す．

図 6 に，信号線 b の 1 縮退故障を検出するテストパターンに対して極大圧縮法を実行した例を示す．この故障に対してテストパターン(a,b,c,d,e)=(0,0,1,0,0)が生成されたと仮定する．まず外部入力 a の論理値を反転させたテストパターン(1,0,1,0,0)で信号線 b の 1 縮

```
maximal_compaction(テストパターン){
        for(i=0;i<外部入力数;i++){
                fault_SIM(外部入力[i]の論理値を反転させたテストパターン);
                if(外部入力[i]の論理値を反転させたテストパターンで目標故障を検出)
                        反転フラグをチェック;
        }
        fault_SIM(反転テーブルでチェックされたビットを全て反転させたテストパターン);

        if(入力反転テーブルで更新されたビットを全て反転させたテストパターンで目標故障を検出)
                return(反転テーブルでチェックされたビットを全てドントケアに変換したテストパターン);
        else
                return(テストパターン);
}
```

図 4.大圧縮法擬似コード

外部入力	a	b	c	d	e
反転フラグ				✓	✓

図 5.入力反転テーブル例

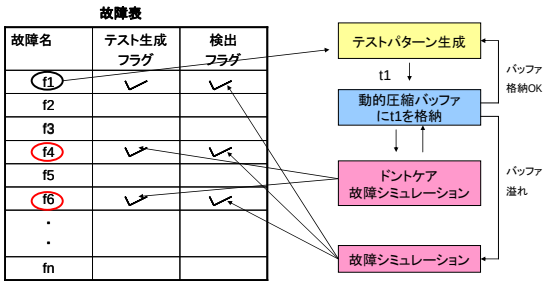

図 3.ドントケア故障シミュレーションの故障表の更新

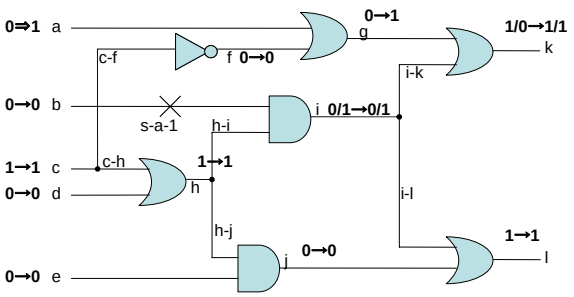

図 6.最大圧縮法実行例

退故障が検出できるかを試みる．しかしながら，このテストパターンでは故障の検出が不可能なので入力反転テーブルの外部入力 a には反転フラグをチェックしない．次に外部入力 b の論理値を反転させたテストパターン(0,1,1,0,0)で検出可能かを試みる．このテストパターンでも故障検出は不可能である．同様に，外部入力 c の論理値を反転させた(0,0,0,0,0)でも故障の検出は不可能である．よって，入力反転テーブルの外部入力bと外部入力cの反転フラグはチェックされない．次に外部入力 d の論理値を反転させたテストパターン(0,0,1,1,0)と，外部入力 e の論理値を反転させたテストパターン(0,0,1,0,1)でそれぞれ故障の検出が可能かを試みる．この 2 つのテストパターンは故障の検出が可能なので，外部入力 d と外部入力 e に反転フラグをチェックする．そして，各外部入力に対して反転させたテストパターンで故障の検出が可能かを試み終えたので，入力反転テーブルの反転フラグがチェックされている外部入力に対して全ての論理値を反転させたテストパターン(0,0,1,1,1)で信号線 b の 1 縮退故障の検出が可能か否かを確認する．このテストパターンは信号線 b の 1 縮退故障の検出が可能である．信号線 b の 1 縮退故障はテストパターン(0,0,1,0,0), (0,0,1,1,1), (0,0,1,1,0), (0,0,1,0,1)で検出が可能である．つまりテストパターン(0,0,1,X,X)で故障の検出が可能である．よって生成されたテストパターン(0,0,1,0,0)を(0,0,1,X,X)に置き換えることができる．また，入力反転テーブルで反転フラグがチェックされている全ての外部入力に対して反転させたテストパターンで目標の故障が検出できなかった場合，初めに生成されたテストパターンを返す.

4. テスト生成アルゴリズムの改善

4.1 極小解圧縮

ドントケアに基づくテストパターンの静的圧縮の貪欲解と厳密解がほぼ一致することが報告されている[5]. 極小解圧縮は, 貪欲法でテストパターンを圧縮し, 極小解を計算することで，最小個のテストパターン数ほぼ一致する値を得るとともに，圧縮速度の向上を図る圧縮法である．本稿ではテスト生成中に動的に極小解圧縮を行う方法を提案する．

テスト生成中に動的に極小解圧縮を実行するには，動的圧縮バッファを用意し，そこに ATPG で生成され極大圧縮したテストパターンを格納し，動的圧縮バッファ内に一定の閾値を設けて，その値までテストパターンを格納すると，極小解圧縮を開始する．多数のテストパターンを極小解圧縮用バッファに格納することにより，テストパターン生成中に静的圧縮に近い環境で極小解圧縮を実行でき，厳密解にほぼ一致するテストパターン圧縮解を得ることができる.

4.2 圧縮アルゴリズム

本稿では頂点彩色問題[5]のアルゴリズムをテストパターン圧縮のアルゴリズムとして採用している．テストパターン圧縮での頂点彩色問題を説明する. まず, 頂点をテストパターン，圧縮可能なテストパターンを辺で結ぶことによりテストパターンの圧縮可能性を無向グラフとして表現することができる．頂点彩色問題はそのグラフに対し補グラフを取り，グラフの隣接する頂点が異なる色になるように，頂点に色を塗るために必要な最小の色数を求める問題である．本稿では頂点彩色問題のアルゴリズムの Dsatur[6]を採用する.

5. 動的圧縮テスト生成システム

図 7 にテスト生成システム全体のアルゴリズムを示す.

(step1)

故障テーブル内の未検出故障または未処理故障が存在しなくなったか否かを判断する．存在しないときは処理を終了し，それ以外のときは step2 へ進む.

(step2)

故障テーブルからテスト生成の対象となる未処理でかつ未検出故障を 1 つ取り出す.

(step3)

テストパターンを生成する.

(step4)

step3 で生成したテストパターンに対し極大圧縮法を実行する.

(step5)

動的圧縮バッファに生成されたテストパターンを格納する.

(step6)

step5 のテストパターンでドントケア故障シミュレ

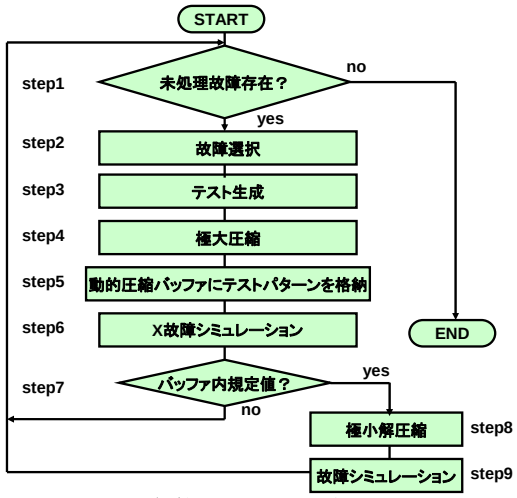

図 7.全体アルゴリズム

ーションを実行し，故障テーブルを更新する．

(step7)

一定の閾値を極小解圧縮バッファに設け，動的圧縮バッファ中のテストパターンがその閾値と一致しなければ step1 へ戻る．閾値とテストパターン数が一致したら step8 に進む．

(step8)

極小解圧縮を実行する．

(step9)

動的圧縮バッファ内で圧縮された全てのテストパターンに対して故障シミュレーションを行い，故障テーブルを更新する．また，故障シミュレーションを実行する際のテストパターン中のドントケアにはランダムに 0 又は 1 を設定する．

6. 実験結果

極大圧縮法を実装し，評価実験を行った．実験では ITC'99 ベンチマーク回路を用いた．表 1 は極大圧縮法の効果を示すための実験結果である．表 1 において「circuit」は極大圧縮法の評価を行う対象の回路名である．「off」はドントケア故障シミュレーションを用いた動的圧縮[3]を実行時に生成されたテストパターン数，「on」はドントケア故障シミュレーションを用いた動的圧縮に極大圧縮法を適用時に生成されたテストパターン数である．「削減数」は「off」で生成されたテストパターン数と「on」で生成されたテストパターン数との差分である．「mc 回数」は生成されたテストパターンに対して極大圧縮され，ドントケア数が増えたテストパターン数を示している．表 2 において「circuit」は極大圧縮法の評価を行う対象の回路名である．「off」はドントケア故障シミュレーションを用いた動的圧縮を実行時のテスト生成時間，「on」はドントケア故障シミュレーションを用いた動的圧縮に極大圧縮法を適用時のテスト生成時間である．「比率」は「off」のテスト生成時間と「on」のテスト生成時間の比率である．表 2 はそのテスト生成の時間の比較を示した実験結果である．今回の実験では，極小解圧縮の圧縮アルゴリズムである Dsatur[6]の実装が間に合わなかったため，文献[3]で提案された動的圧縮アルゴリズムを適用している．

表 1 から，極大圧縮法で生成されたテストパターン集合のドントケア数の増加に成功し，それに伴い，テストパターン数が削減されており，極大圧縮法が有効であることがわかる．しかし，表 2 から極大圧縮法を適用すると，適用しないときより約 1.5 倍テスト生成時間が増加しているのがわかる．また，必ずしも極大圧縮した回数とテストパターン削減数が比例しないことがわかる．このことより，極大圧縮の入力反転テーブルにフラグが立った入力を全て反転させると，目標故障を検出できない場合が多いことがわかる．

表 1. テストパターン数の比較

circuit	off	on	削減数	mc_回数
b_14	1239	1226	13	1
b_15	790	779	11	5
b_20	1822	1812	10	1
b_21	1925	1910	15	1
b_22	1920	1909	11	13

表 2. テスト生成時間の比較

circuit	off(s)	on(s)	比率(%)
b_14	1842	2804	125.2
b_15	3847	4456	115.8
b_20	6567	11753	178.9
b_21	7052	12535	177.7
b_22	14461	28763	198.9

7. おわりに

本稿では極大圧縮法と極小解圧縮をテスト生成中に動的に実行するアルゴリズムを提案した．また極大圧縮法の評価を行った．その結果，ドントケア数が増加しながら，テストパターン数の削減に成功した．しかし，テスト生成時間が 1 倍～2 倍に増加するという課題を残した．

今後の課題として，テスト生成時間の高速化と，本稿の提案手法である極小解圧縮アルゴリズムの評価することが挙げられる．

参考文献

[1] Y.Sato, T.Ikeda, M.Nakao, and T.Nagumo, "Abist approach for very deepsub-micron (vdsm) defect, "Proc. International Test Conference, pp. 283291, 2000.

[2] Y.Matsunaga, "MINT-An exact algorithm for finding minimum test set", IEICE Trans. Fundamentals vol.E76-A, pp1652-1658 (1993)

[3] 秋山祐介, "ドントケア故障シミュレーションを用いた動的テスト圧縮の効率化", 平成 18 年度日本大学生産工学部数理情報工学科卒業研究概要集 (2006)

[4] Irith Pomeranz, "COMPACTEST: A METHOD TO GENERATE COMPACT TEST SETS FOR COMBINATIONAL CIRCUITS", IEEE(1991)

[5] 八木澤圭, "テストパターンの静的圧縮における厳密解と貪欲解の比較" IEICE Technical Report DC2 007-79 (2008)

[6] D.Brelaz, "New methods to color the vertices of a graph",Communications of the ACM, 22, pp.251-256(1979)