ISSN 2186-5647

非順序式バックトラックを用いたテスト生成アルゴリズムの高速化

日大生産工　○鈴木　悠介
日大生産工　細川利典　九大　吉村正義

1　はじめに

近年，大規模集積回路(Large Scale Integrated circuits:LSI)の大規模化・複雑化に伴い，テストパターンの生成時間の増加や打ち切り故障の増加による故障検出効率の低下が問題となっている．

これらの問題を解決するためには，スキャン設計のようなテスト容易な設計技法で論理設計を行うとともに，さらに効率の良いテスト生成アルゴリズム[1]を考案することが必要である．

効果的なテスト生成アルゴリズムの一つとしてSPOP[2]が挙げられる．しかしながら，このアルゴリズムを用いても，現実的な時間で 100%の故障検出効率を得るテストパターンを生成することは困難である．

そのため，効率よくテスト生成を行うためのアルゴリズムを実装し，テストパターンの生成を容易にする必要がある．テスト生成を高速化するためには，主に含意操作の高速化[3]とバックトラック回数の削減[4-5]の二点が挙げられる．

含意操作の高速化の代表例として，一意活性化や学習機能を用いることで一意に値を決定できる信号線を増加させ，矛盾の早期発見や解の探索空間を削減する手法が挙げられる．

テスト生成時において，バックトラック回数が増加すると，含意操作処理の増加が発生し，テスト生成時間の増加につながる．そのため，効率よくバックトラックを行う必要がある．

本研究では，バックトラック回数の削減に着目し，含意グラフ[5]を用いた非順序式バックトラック手法と従来手法の順序式バックトラック手法のバックトラック回数，故障検出率，故障検出効率，テスト生成時間の比較を行う．非順序式バックトラック手法は故障伝搬アルゴリズムに SPOP，正当化アルゴリズムに後方追跡[6]に基づく含意ノード[7]割当てアルゴリズムを用いたテスト生成アルゴリズムに対して実装される．

2　テスト生成

テスト生成は，主に以下の 3 つの処理から成り立っている．

・ある信号線への値割当て処理
・値割当てに伴う含意操作の処理
・値割当ての結果，値の矛盾が発生せずテストパターン生成ができたかを判定する処理

テスト生成時の値割当ては決定木[8]を用いて信号線の値の選択を行う．

2.1 決定木を用いたテスト生成

決定木とは，ノードは変数，エッジはその変数がとり得る値や選択肢をもつ．

テスト生成における決定木は，選択処理による値割当てと値割当てによって一意に決定される信号線の値情報を保持している．図 1 はテスト生成の正当化処理に対する決定木を示す．最初の選択処理で信号線 a=0，次に信号線 b=1 と選択して，テストパターン生成に成功した例を示している．

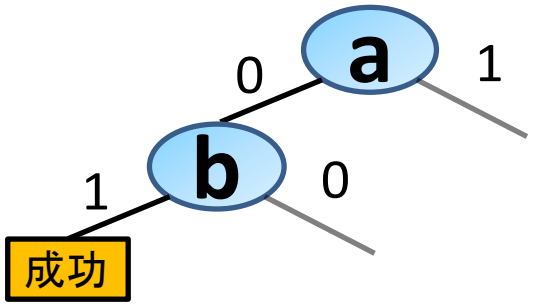

図 1．決定木を用いたテスト生成

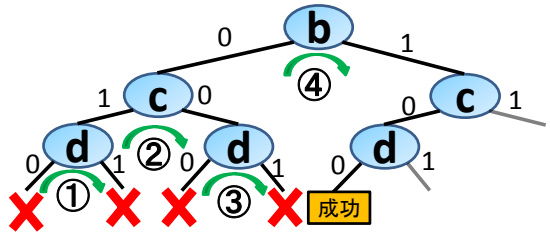

図 2．順序式バックトラック手法

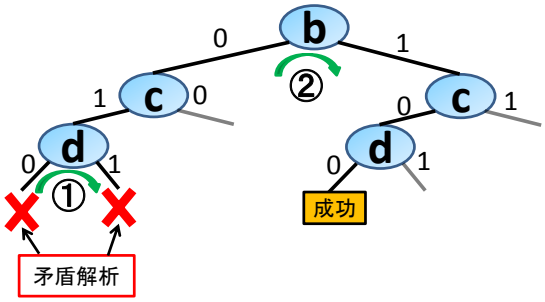

図 3．非順序式バックトラック手法

2.2 順序式バックトラック手法

含意操作の処理において，信号線の値の矛盾が発生した場合，割当てた値を戻して別の値を割当てる処理をバックトラックという．従来の順序式バックトラック手法は値の矛盾が発生した場合，決定木に積まれている最新の割当てた値とその割当てによる含意操作で決定された値を無効にして，逆の値を割当てる．もし逆の値を試しているならば，次に新しく決定木に積まれている値に対して逆の値を割当てる手法である．

図 2 にテスト生成の正当化処理時において，値の矛盾が発生した場合の決定木の例を示す．選択処理で b=0→c=1→d=0 の順で値割当が行われる．この時，d=0 の割当てによる含意操作時に値の矛盾が発生した場合，バックトラック処理により信号線 d の値と含意操作された信号線の値を全て無効にし，d=1 が割当てられる．また d=1 の含意操作時にも値の矛盾が発生した場合，すでに逆の値である d=0 を試しているので次に新しい割当てである c のノードに戻り，c=1 の割当てを無効にし，

On the Accleration of Test Generation Algorithm Using Non-chronological Backtrack Techniques

Yusuke SUZUKI and Toshinori HOSOKAWA

パターンが生成されるまで処理が繰り返し行われる．図 2 ではバックトラック処理を 4 回繰り返して b=1→c=0→d=0 で矛盾なくテストパターン生成に成功する．

順序式バックトラック手法の問題点として図 2 のように決定木の根である a の割当てた値が矛盾の原因だった場合，バックトラック回数が増加し，テスト生成時間の増加につながってしまう欠点がある．

2.3 非順序式バックトラック手法

非順序式バックトラック手法では，最も最近割当てた値の逆の値を同じ信号線に割当てる．もしすでに試されている場合は，値の矛盾解析をし，矛盾に関係のある最も最近割当てた値の信号線まで値を無効にし，バックトラックを行う．

図 3 では図 2 と同じ信号線の故障を対象としてテスト生成を行う例を示している．図 2 と同じ処理を行い，d=1 で衝突が発生し，d の値割当てをすべて試した場合，非順序式バックトラック手法では矛盾の原因を解析する．解析した結果，b=0 の割当てが矛盾に関係のある最も最近の値の割当だった場合，b までの値割当てをすべて無効にし，b=1 の割当てを行う．つまり c=0 の割当てを行わないので，図 3 の例では順序式バックトラック手法より少ないバックトラック回数で故障検出できることがわかる．

バックトラック処理を削減するためには矛盾の原因を解析し，原因となっている値割当てまで直接バックトラックを行う方が効率的だと考えられる．本論文では，効率的なバックトラック手法として，含意グラフを用いた矛盾解析による非順序式バックトラック手法を用いる．

3　含意グラフ

含意グラフとは含意操作時における回路内の信号線の依存関係を有効グラフで示したもので，直接含意と間接含意の 2 種類がある．

3.1 直接含意

直接含意とはある信号線に値割当てを行うと一意に決定される信号線の含意を示す．図 4(左)は OR ゲートの入力に a=1 を割当てた場合を示している．この場合，含意操作により出力の c に 1 が一意に割当てられる．この時，含意グラフでは c は a によって値が一意的に割当てられたので図 3(左)のように a→c の矢印で結ばれる．図 4(右)では OR の入出力が d=0, f=1 となっている場合を考える．この場合，含意操作により e に 1 が一意に割当てられる．この時，含意グラフで e は d と f によって値が一意的に割当てられたので d→e←f の矢印で結ばれる．

3.2 間接含意

図 5 の回路において c=0 が割当てられた時を考える．静的学習などから得られた間接含意情報により d が 1，e が 1 に割当てられる．この時，含意グラフは d←c→e の矢印で結ばれる．

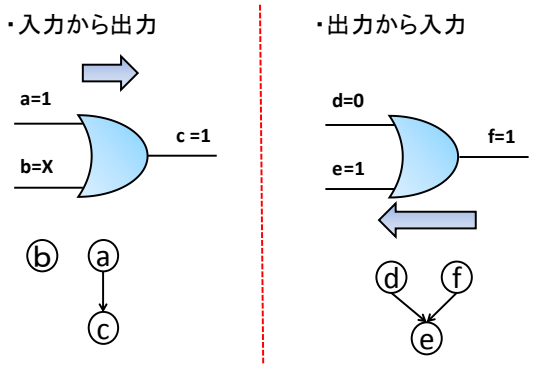

図 4.　直接含意

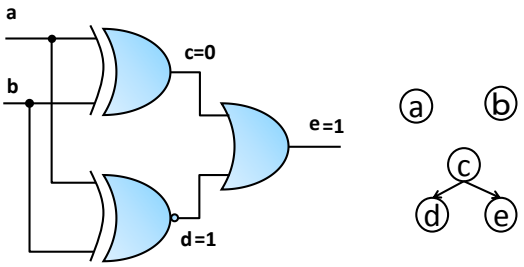

図 5.　間接含意

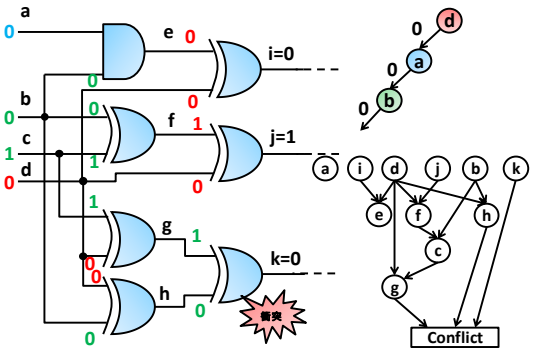

図 6.　含意グラフを用いた矛盾解析(b=0)

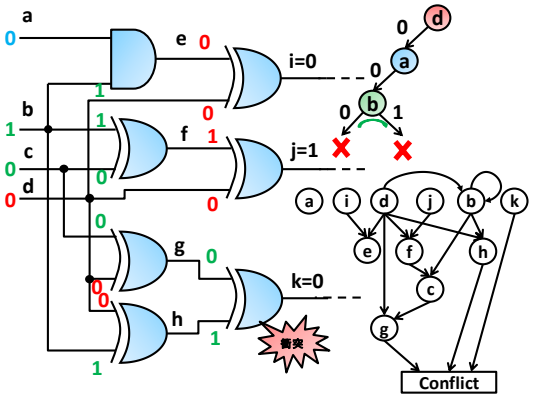

図 7.　含意グラフを用いた矛盾解析(b=1)

3.3 含意グラフを用いた矛盾解析処理

図 6,7 は i=0, j=1, k=0 の正当化を行う場合で，正当化は PODEM ベースで入力信号線のみを決定木のノードに積む場合を例としている．含意グラフを用いた非順序式バックトラック手法はあるノードのすべての選択肢が矛盾した場合に実行される．

まず，i=0 を正当化するために信号線 d に 0 の割当て目標を立て，d=0 の値割当てを行う．d=0 により，e=0, f=1 が一意に割当てられる．次に e=0 を正当化するために，a=0 を割り当てる．f=1 を正当化するために，b=0 を割当て，含意操作により，g=1，h=0 が一意に割当てられる．ここで信号線 k において値の矛盾が発生している．ここではノードの選択肢が残っているので矛盾解析処理を行わず，従来の順序式バックトラックを行う(図 6).

しかしながら，b=1 の割当てを行っても，同様に信号線 k で値の矛盾が発生している．ここでノード b のすべての選択肢を試しているので矛盾解析処理を行う(図 7).

矛盾解析処理では含意グラフの情報を使用する．まず b=0 における含意グラフ(図 6)から，矛盾の原因となった g，h，k がどの信号線の影響を受けているか探索することで原因を解析する．つまり各ノードの矢印を逆に辿っていくと元となった影響信号線がわかる．図 6 から d，j，b，k の値割当てから影響を受けており，これらの信号線が b=0 の時の矛盾の原因になっていると考えられる．同様に，b=1 の時(図 7)も探索すると d，j，b，k の値割当てが原因となっている．このことから，b の値割当ての矛盾の原因は d，j，b，k の値割当てが原因となっていることが分かる．

3.4 矛盾解析による非順序式バックトラック手法

図 7 の例では，従来の順序式バックトラック手法を用いた場合，a=1 の割当てにバックトラックしても b の値割当て時に，同じ原因で値の矛盾が発生することが考えられる．そのため非順序式バックトラックでは矛盾の原因の内，最も最近に割当てられた信号線，つまり d=1 の割当てでバックトラックを行うことで b の値割当て時に再び矛盾が発生するのを防ぐことができる．この手法は決定木において矛盾の原因が根にあるほど有効な手法だと考えられる．

4　実験結果

本実験では従来の順序式バックトラック手法においてバックトラック回数が多い回路(b04b05,b14,b15,b20,b21)を対象とし，それらを 1 時間展開の組合せ回路，故障モデルは縮退故障[1]として実験を行う．しかしながら，すべての故障に対して含意グラフ生成と矛盾解析を用いていたのでは実行時間の増加を招くので，予備実験として各回路のどの故障を対象に非順序式バックトラック手法を用いるかを選別する．また今回は静的学習を行わず，含意グラフは直接含意グラフのみ生成する．

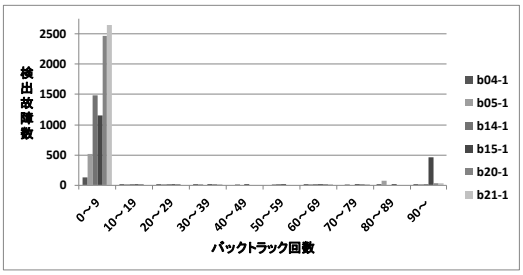

図 8. バックトラック制限=100

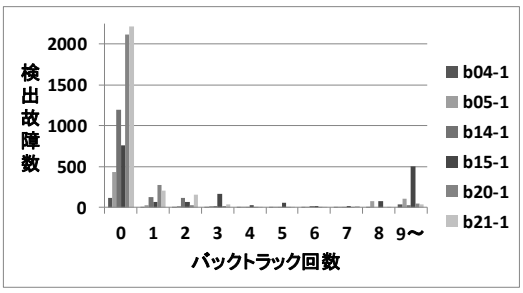

図 9. バックトラック制限=10

4.1 予備実験

図 8 はバックトラック制限=100 とした場合の検出故障数とバックトラック回数の割合を示し，90~は 90 回以上のバックトラック回数を示す.図 8 より 9 割以上の故障が 0~9 以内のバックトラック回数で検出できていることがわかる．図 9 はさらにバックトラック制限=10 とした場合で，9~は 9 回以上のバックトラック回数を示す．図 9 より 9 割以上の故障が 3 回以内のバックトラック処理で故障検出可能だということが分かる．

4.2 実験

本実験では予備実験の結果から最初にバックトラック制限=3 でテスト生成を行い，打ち切られた故障を対象にバックトラック制限=1000 で順序式バックトラック手法と矛盾解析を用いたバックトラック手法のテスト生成をそれぞれ行い，故障検出率，故障検出効率，テスト生成時間の比較を行った．

結果を表 1 に示す．各回路の上段が順序式バックトラック手法，下段が矛盾解析を用いたバックトラック手法の結果を示している．また，バックトラック回数は伝搬フェーズ，正当化フェーズそれぞれ分けて算出している．項目の矛盾解析はあるノードの選択肢をすべて試し，矛盾解析処理を行った回数，非順序式バックトラック実行回数は非順序式バックトラックを行い，選択肢をすべて試さずにバックトラックを行ったノードの数を示している．

4.3 考察

結果からすべての回路において平均バックトラック数を削減できている．これは矛盾解析により解のない探索空間を少ないバックトラック回数で

回避できていると考えられる．また非順序式バックトラック手法を行うことにより打ち切り故障を削減し，故障検出効率，故障検出李がすべて順序式バックトラック手法以上に向上した．テスト生成時間については b04,b05,b14,b20 は削減できたが，b15,b21,17 の回路は実行時間が増加してしまった．この 3 つの回路についてみると，故障数に対して，打ち切り故障数の割合が多い傾向にあった．つまり探索空間が広すぎて，非順序式バックトラックを行っても，解空間にたどり着けなかった故障が多かったと言える．矛盾解析の回数と非順序式バックトラック実行回数の関係についてみると，矛盾解析を行ったからと言って必ずしも非順序式バックトラックを行っていなかった．

5　まとめ

本論文では，テスト生成アルゴリズムに含意グラフを用いた非順序式バックトラック手法を実装し，その評価を行った．結果，バックトラック数鵜の削減や故障検出効率の向上が見られたが，実行時間は探索空間の広い回路では増加してしまった このことから探索空間をより削減していく必要がある．

今後の予定として以下の点が挙げられる．

・間接含意グラフの導入及び静的学習との併用
・より非順序式バックトラックの効果を示す指標の作成
・FANⅢや D アルゴリズムの併用

・参考文献

[1] H. Fujiwara, “Logic Testing and Design for Testability”, The MIT Press, 1985.

[2] M. Henftling, H. C. Wittmann, K. J. Antreich, “A Single-Path-Oriented Fault-Effect Propagation in Digital Circuits”, IEEE/ACM International Conference, pp.304-309, Nov 1995.

[3] MICHAEL H. SCHULZ, ERWIN TRISCHLER, THOMAS M. SAPFERT, “SOCRATES: a highly efficient automatic test pattern generation system”, IEEE Transactions on Computer-Aided Design, Vol.7, No.1, pp.126-137, Jan 1988.

[4] 吉村正義，松永裕介，“非順序式バックトラック手法を用いたテストパタン生成における矛盾の解析”，情報処理学会シンポジウム論文集，pp.211-214, 2008.

[5]Xijiang Lin,”Techniques for improving the efficiency of sequential circuit test generation”,IEEE/ACM international conference on Computer-aided design,p147-151,1999

[6] Fujiwara, H,　Shimono, T, “On the Acceleration of Test Generation Algorithms”,IEEE Transactions,p1137-1144,Dec. 1983

[7] Mituo Teramoto, “A Method for Reducing the Search Space in Test Pattern Generation,” Proc.International Test Conference, pp.429-435, 1993.

[8] M. Abramovici, M. A. Breuer and A. D. Friedman, “Digital systems testing and testable design,” IEEE Press, 1995.

表 1. 実験結果

回路名	故障数	検出数	打切り数	冗長数	バックトラック回数		平均バックトラック数	矛盾解析	非順序式バックトラック実行回数	故障検出効率(%)	故障検出率(%)	テスト生成時間(秒)
					正当化	伝搬						
b04-1	73	63	0	10	2108	381	73.21	–	–	100	86.3	1.42
		63	0	10	698	273	28.56	555	163	100	86.3	0.7
b05-1	231	62	33	136	54227	10585	327.33	–	–	85.71	26.84	50.69
		59	0	172	12896	2719	78.47	12056	1452	100	25.54	15.43
b14-1	52	35	10	7	13460	265	280.1	–	–	80.77	67.31	35.23
		38	7	7	7481	247	157.71	5131	690	86.54	73.08	22.03
b15-1	921	319	360	242	364036	28062	543.07	–	–	60.91	34.64	1559.91
		346	247	328	273622	25631	428.12	175134	57532	73.18	37.57	2219.04
b21-1	82	38	35	9	35834	310	463.38	–	–	57.32	46.34	54.29
		38	34	10	35180	283	454.65	32884	1511	58.54	46.34	65.15
b20-1	59	25	26	8	31766	269	572.05	–	–	55.93	42.37	57.53
		37	10	12	12290	297	228.85	10025	3051	83.05	62.71	36.74
b17-1	2025	621	674	730	709162	89961	480.82	–	–	66.72	30.67	3555.78
		639	416	970	485961	47605	324.55	359417	102307	79.46	31.56	3691.79

上段	順序式バックトラック手法
下段	矛盾解析を用いたバックトラック手法