ISSN 2186-5647

2-59

必須割当てを用いたテスト生成の高速化について

日大生産工(院)　○日下部　建斗　日大生産工(院)　山崎　紘史
日大生産工　細川　利典　京産大　吉村　正義

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 Integrated circuits: LSI）が大規模化・複雑化している．そのため，テスト生成時間が増加している．テスト生成手法には回路構造に基づくテスト生成と，充足可能性(Satisfiability: SAT)を用いたテスト生成[6]がある．回路構造に基づくテスト生成の代表的なアルゴリズムとして，PODEM[1]，FAN[2]，SOCRATES[3]，Dアルゴリズム[4]など多数が提案されている．SATベーステスト生成は，論理回路をCNF式に変換し，すべての変数を1(真)，0(偽)に定めることで，式全体の値を1(テスト生成可能)とする割当てが存在するか否かを判定する手法である．SATベーステスト生成では，単位項[6]を入力することで解探索空間を削減することができるため，CNF式を高速に解くことができる．単位項は回路における固定値や，故障に対する必須割当てに対応している．故障に対する必須割当ては，故障挿入[2]，含意操作[7]，一意活性化[2]などを用いることにより求めることができる．

本論文では各故障の必須割当てを求め，この必須割当てを単位項に変換し，SATベーステスト生成の制約として加えることによりテスト生成を高速化する手法を提案する．ISCAS85，ISCAS89 ベンチマーク回路でその効果を評価する．

2. SATベーステスト生成

SATとはCNF式が与えられたときに，すべての変数の値を1(真)または0(偽)のどちらかに定めることで，式全体の値を1(真)にできる割当てが存在するか否かを判定する．CNF式を1(真)にできる割当てが存在した場合，充足可能といい与えられた CNF 式が正当化可能であることを示す．

SATはCNF式を入力として問題を解決するため，論理回路をCNF式に変換する必要がある．変換するためには各論理ゲートを CNF 式変換規則に従い変換する．表1に各論理ゲートの変換規則を示す．表1の括弧内の論理和を節といい，節の論理積をとった式がCNF式である．例として2入力(X, Y)1出力(Z)のANDゲートを考える．全体の値が1(真)になるのは(X, Y, Z)=(0, 0, 0)，(1, 0, 0)，(0, 1, 0)，(1, 1, 1)の4通りだけである．これによりCNF式が実際の AND ゲートの動作を表現していることがわかる．回路全体のCNF式は各論理ゲートの動作を表現していることがわかる．

次に SAT ベーステスト生成について説明する．初めにテスト対象となる回路に対して，正常動作する回路と，ある1本の信号線に縮退故障を仮定した故障回路を用意する．そして，正常回路と故障回路の入力に共通の外部入力を接続する．外部出力に対しては，故障の影響を励起・伝搬させるために片方の外部出力を真，もう一方を偽にさせる必要があるため，それぞれの外部出力をEXORゲートの入力に接続する．

故障の検出には少なくとも一本の外部出力へ故障の影響が伝搬できればよいため，各外部出力を接続したEXORゲートの出力をORゲートの入力に接続し，このORゲートの出力が常に1(真) となるCNF節を付加する．故障情報は故障信号線に対して，外部入力側に正常値，外部出力側に故障値を設定することで示す．

図1にSATベーステスト生成モデルの例を示す．図1のG1, G2, G3が正常回路であり，G1’，G2’，G3’が故障回路である．NOT ゲートG3’の入力信号線に1縮退故障があると仮定する．このためANDゲートG1’の出力が正常値0となるように設定し，NOT ゲートG3’はG1’からの入力が故障値1となるように設定する．a，b は外部入力であり，これらは正常回路と故障回路の入力に接続している．正常回路の外部出力はc, dであり故障回路の出力はc’,

On The Acceleration of Test Generation Using Necessary Assignments

Kent KUSAKABE , Hiroshi YAMAZAKI , Toshinori HOSOKAWA and Masayoshi YOSHIMURA

d’である．これらの出力 c，c’と d，d’をそれぞれ EXOR ゲートの入力に接続し，EXOR ゲートの出力が e，f である．EXOR ゲートの出力である e，f を OR ゲートの入力に接続し，その OR ゲートの出力が g とし，故障を検出するために 1 を設定する．

表 1．論理ゲートの CNF 式変換規則

ゲートタイプ	入力	出力	CNF
AND	X Y	Z	(¬Z+X)･(¬Z+Y)･(¬X+¬Y+Z)
OR	X Y	Z	(Z+¬X)･(Z+¬Y)･(X+Y+¬Z)
EXOR	X Y	Z	(¬X+Y+Z)･(X+¬Y+Z)･(¬X+¬Y+¬Z)･(X+Y+¬Z)
NOT	X	Y	(X+Y)･(¬X+¬Y)
FOUT	X	Y Z	(¬X+Y)･(X+¬Y)･(¬X+Z)･(X+¬Z)

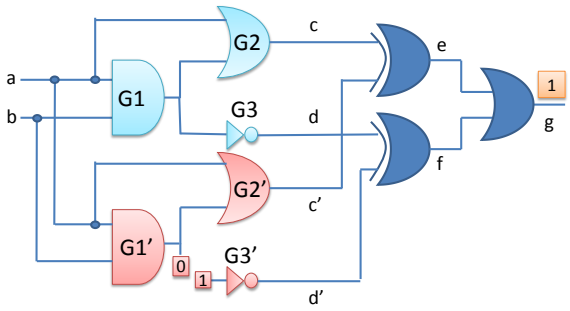


図 1．SAT ベーステスト生成モデル回路例

3. 必須割当て

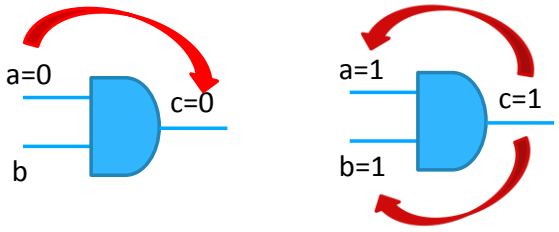
必須割当てとは故障を検出するために必要な信号線の論理値割当てである．本論文では必須割当てを求めるために，故障挿入，含意操作，一意活性化を用いる．以下にこの 3 つの操作を説明する．

3.1 故障挿入

故障挿入とはある信号線に対して故障を仮定した場合に，故障影響を外部出力まで伝搬させるために故障を仮定した信号線に故障の論理値とは逆の論理値を正常値として割当てる操作である．故障が 0 縮退故障の場合は故障を仮定した信号線に正常値として 1 を割当て，1 縮退故障の場合は故障を仮定した信号線に正常値として 0 を割当てる．

3.2 含意操作

含意操作とはある信号線の論理値を決定することにより，一意に決定される他の信号線の論理値を求める操作のことである．含意操作はゲートの入出力の接続関係から論理値を求めることのできる直接含意[7]と，ゲートの入出力の接続関係だけでは論理値を求めることのできない間接含意[3]がある．また直接含意は前方含意と後方含意に分けられる．図 2 に直接含意例を示す．図 2(a)は前方含意の例である．信号線 a に AND ゲートの制御値が割当てられているとき，AND ゲートの接続関係から出力信号線の値を一意に決定することができる．ゲートの入力側から出力側に値を推移していくのが前方含意である．図 2(b)は後方含意の例である．出力信号線 c の値が 1 であるとき AND ゲートの接続関係から入力信号線 a，b の値は 1 と決定できる．ゲートの出力側から入力側に値を推移していくのが後方含意である．図 3 に間接含意例を示す．信号線 d の信号値を 1 とした場合，直接含意では他の信号線は決定できないが，b=1 または c=1 のいずれかが成り立たなければならないため，いずれの場合も a=1 が成り立つ．図 3 のように d=1 ならば a=1 となるのが間接含意である．



(a)前方含意　　(b)後方含意

図 2．直接含意例

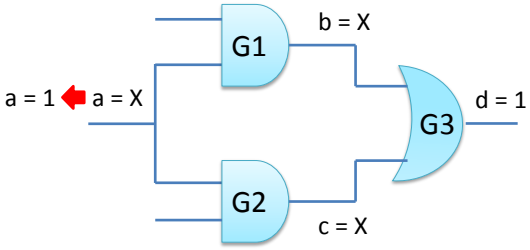


図 3．間接含意例

3.3 一意活性化

一意活性化とは，D フロンティア[1]が唯一のとき，D フロンティアを外部出力に伝搬する際に，必ず通る部分回路を活性化するために部分回路上の各ゲートの故障が到達不可能な信号線であるサイドインプットに非制御値を割当てる操作である．文献[2]では回路構造のみから一意活性化可能な信号線を求める方法が提案されていたが文献[7]では回路構造と現在の信号線に割当てられた値から動的に一意

活性化可能な信号線を求める方法が提案されている．

3.4　必須割当てを求めるアルゴリズム

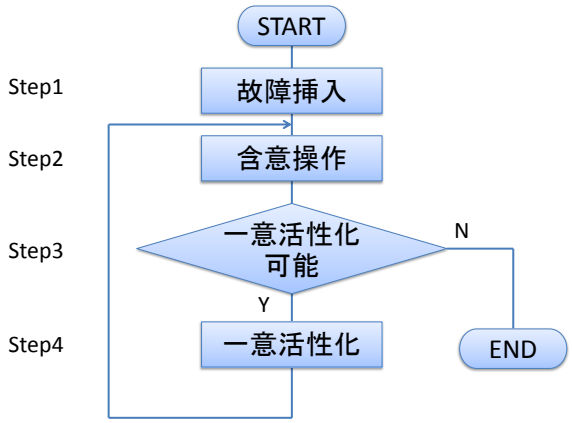


図 4. 必須割当てを求めるアルゴリズム

(Step 1)

テスト対象回路に対して故障挿入を行う．

(Step 2)

Step 1 で励起した論理値及び Step4 の操作で決定した信号線の論理値に対して，含意操作を行う．

(Step 3)

Step2 の含意操作後の回路状態で一意活性化可能か否かを判定する．一意活性化可能である場合は Step4 へ，不可能な場合は終了する．

(Step 4)

D フロンティアを外部出力に伝搬する際に，必ず通る部分回路を活性化するために部分回路上の各ゲートのサイドインプットに非制御値を割当てる．

以下に図 5 の回路を用いた必須割当てを求める例を示す．

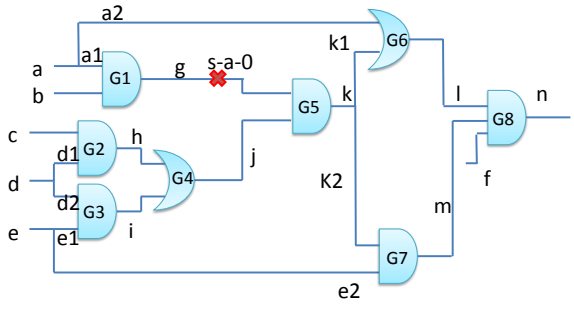


図 5. 必須割当てを求めるための回路例

図 5 の信号線 g に 0 縮退故障を仮定した場合の必須割当てを求める．初めに信号線 g に対して故障挿入を行うと信号線 g の値は 1/0 になる．次に信号線 g の値から含意操作を行う．信号線 g の正常値を 1 にするためには信号線 a1, b が 1 である必要があるため，信号線 a1，b は 1 となる．信号線 a1 が 1 であることから信号線 a，a2 が含意操作により 1 となる．信号線 a2 の値が 1 になることで G6 は OR ゲートのため前方含意により信号線 l の値は 1 となる．次に G5 が唯一の D フロンティアとなるため，一意活性化を行う．まず G5 のサイドインプットである信号線 j に G5 の非制御値である 1 を割当てる．次に G6 は出力が固定されているため，G7 が動的一意活性化[6]の対象となり G7 のサイドインプットである信号線 e2 に G7 の非制御値である 1 を割当てる．最後に G8 が一意活性化の対象となるため G8 のサイドインプットである信号線 f に G8 の非制御値である 1 を割当てる．一意活性化によって割当てた信号線の論理値が増加したため再び含意操作を行う．信号線 k は入力信号線 g と j の値が決定したため 1/0 と 1 より信号線 k は 1/0 となり，信号線 k1，k2 も同様に 1/0 となる．信号線 k2 の値が決定したため k2 と e2 の値より信号線 m の値は 1/0 となる．信号線 l，m，f の値が決定したため信号線 n の値は 1/0 となる．信号線 e2 に値が決定したため信号線 e，e1 の値は 1 となる．信号線 j の値が 1 であることから，間接含意[2]より信号線 d の値は 1 となり，前方含意操作により信号線 d1, d2 の値は 1 となり，信号線 i の値も 1 となる．以上のことから信号線 g の 0 縮退故障の必須割当ては (a,a1,a2,b,c,d,d1,d2,e,e1,e2,f,g,h,i,j,k,l,m,n)= (1,1,1,1,X,1,1,1,1,1,1,1,1,1,X,1,1,1,1,1,1,1,1) となる．

4. 必須割当てを制約としたテスト生成法

必須割当てを制約としたテスト生成法とは，まず各故障の必須割当てを 3．4 節で説明したアルゴリズムによって求める．次に SAT に各故障の必須割当てを制約として与えて，その制約を単位項に変換し CNF 式に加えることによって解探索空間を削減することができる．制約がない場合と制約がある場合の例を図 6，図 7 に示す．変数を a，b，c，d とすると，制約がないとき，最悪の場合探索木をすべて探索する必要があり，図 6 の場合 15 回バックトラックが発生する．

しかしながら a=1, b=1 という制約を与えた場合，解探索空間は最悪の場合でも，図 7 のよ

うな探索木となり，3 回のバックトラックで SAT となる．

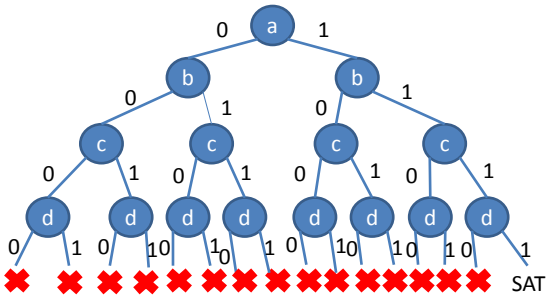


図 6. 制約なしの場合の探索木

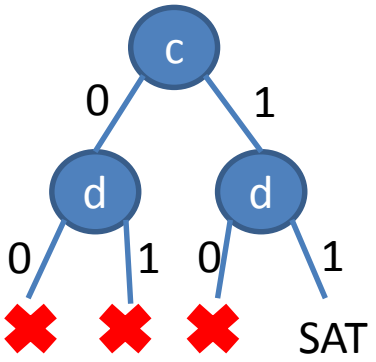


図 7. 制約ありの場合の探索木

5. 実験結果

本論文では SAT に 3.4 節のアルゴリズムにより求めた必須割当てを単位項に変換し，制約として CNF 式に追加した場合と，追加しなかった場合で比較を行った．実験環境は CPU：Intel Corei73．40Ghz，メモリ：8GB，OS：Windows8 である．実験対象は ISCAS85，ISCAS89 ベンチマーク回路を使用した．以下に実験結果を示す．

表 2. ISCAS85 での実験結果

回路名	代表故障数	テスト生成時間(sec)		バックトラック数	
		従来	提案	従来	提案
c1908	1879	61.48	67.83	256402	259058
c2670	2747	67.37	69.01	292460	287824
c3540	3428	550.29	547.02	1208909	1238931
c5315	5350	218.81	200.59	796448	780387
c6288	7744	756.7	693.52	1983201	1983841
c7552	7550	315.06	307.5	733802	737551

表 3. ISCAS89 での実験結果

回路名	代表故障数	テスト生成時間(sec)		バックトラック数	
		従来手法	提案手法	従来手法	提案手法
cs5378	4551	62.48	73.39	64487	64450
cs9234	6927	278.33	272.5	285755	284187
cs13207	9663	364.5	452.77	199833	200149
cs15850	11697	585.41	607.46	299210	294946
cs35932	38518	2372.78	2697.45	236477	235980
cs38417	30744	3090.87	3130.17	875096	874159
cs38584	35995	2349.78	2840.84	243006	242653

6. おわりに

本論文では SAT ベーステスト生成に対して必須割当てを制約としてテスト生成を高速化する手法を提案した．ISCAS85 の回路においては多少テスト生成時間が減少しているが，ISCAS89 の回路においてバックトラック数は減少しているもののテスト生成時間は減少しなかった．

参考文献

[1] Goel, P.：An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits, IEEE Trans. 1981.

[2] H. Fujiwara, T. shimono, ”On the Acceleration of Test Generation Algorithms”, IEEE Transactions on Electronic Computers, Vol. 32, No. 12, Dec, 1983, pp. 1137-1144.

[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, Jan, 1988, pp. 126-137

[4] J. PAUL ROTH, WILLARD G. BOURICIUS, PETERR, Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits, Electronic Computers, IEEE Transactions, Oct. 1967, p567-580

[5] Tracy Larrabee, “Test Pattern Generation Using Boolean Satisfiability”, IEEE TRANSACTION ON COMPUTER-AIDEDDESIGN, VOL. 11, No1, 1992.

[6] 鈴木悠介，山崎紘史，細川利典，吉村正義，山崎浩二，中尾教伸，”テスト生成アルゴリズムのおける検出困難故障の評価” 2013

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