SATを用いた検出困難故障に対する テスト生成・冗長故障判定の高速化

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　　九大　吉村　正義

1 はじめに

近年，LSIの大規模化・複雑化が進み，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が困難となっている．

テスト生成中，バックトラックの制限値を設定し，その制限値内でテストパターンが生成できない検出困難故障に対しては，テストパターンの生成を途中で打ち切っている．しかしながら，このような打切り故障は冗長故障であることが多い．したがって，テスト生成前に回路中の冗長故障を判定し，それらの故障をテスト生成の対象から除いておくことで，テスト生成時間の削減が期待できる．文献[1]で提案されている冗長故障判定アルゴリズムは故障の励起と外部出力までの伝搬操作を基本のアルゴリズムとしている．そのため，励起や含意で割り当てた信号値の組み合わせが正当化できないものであってもその判断ができない．したがって，実際には冗長故障でありながらテスト生成の対象と判定してしまうことがある．この問題を解決するためには，冗長故障の判定に未正当化信号線の信号値の組み合わせが正当化可能であるか否かを判定する必要がある．

一方，近年ではSAT-solverの処理速度の高速化に伴い，回路の等価性検証やテスト生成技術など論理関数を処理する分野で，SAT（充足可能性問題）を利用した手法が提案されている[2][3][4]．

よって，本稿ではSATを利用することで高速な正当化可能判定が実現できることに着目し，文献[1]の冗長故障判定手法に，SATに基づく正当化処理を組み込んだ手法を提案する．本手法は少ないバックトラック制限値のATPG(Automatic Test Pattern Generation：自動テストパターン生成)で打切り故障と判定された検出困難故障を対象とした冗長故障判定能力に優れたテスト生成方法である．

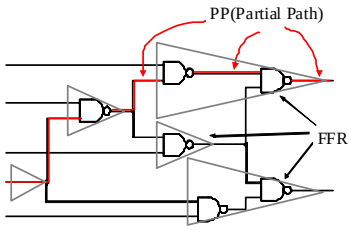

図１．FFR分割

（定義１：検出困難故障）

テスト生成が困難なため，定めた一定のバックトラック制限内ではテスト生成が不可能なため打切りと判定された故障を検出困難故障という．

2 冗長故障判定アルゴリズム

現在，FFR（fan-out free-region）や再収斂ブロックと呼ばれる単一出力回路ブロック内で判定した故障の情報を共用することにより，比較的効率良く高速に冗長故障を判定するREDI[1]と呼ばれるアルゴリズムが提案されている．これらの基本アルゴリズムは，単一故障伝搬経路の活性化に基づくテスト生成アルゴリズムSPOP[5]の故障伝搬処理に基づいたものである．

2.1 基本アルゴリズム

組合せ回路は，FFR（fan-out free-region）や再収斂ブロックと呼ばれる単一出力の回路ブロックに分割できる．図１に回路をFFR分割した例を示す．

図1の三角形で囲まれた部分がFFRである．FFR内には，どの入力からも出力に対してただ１つの部分経路（partial path）が存在する．そのため，FFRの接続によって，回路中の全ての経路が得られる．REDIは，この部分経路の活性化と，含意操作によって冗長故障を判定する．

以下にREDIの基本アルゴリズムを示す．

On the Acceleration of Test Generation and Redundant fault Identification for Hard-to-Detect faults Using Satisfiability
Yusuke AKIYAMA, Toshinori HOSOKAWA,
Koji YAMAZAKI and Masayoshi YOSHIMURA

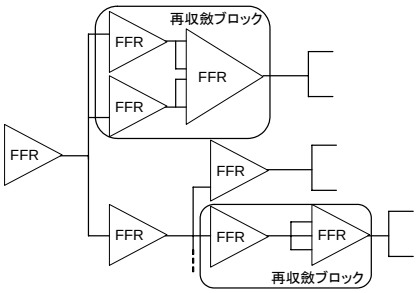

図２．再収斂ブロック分割

（step1）
対象となる全故障に対して冗長判定が行われたたか判断する．まだ未処理の故障がある場合は故障をひとつ取り出す．その故障を励起してstep2へ進む. 未処理の故障がない場合は処理を終了する．

（step2）
故障の伝搬，含意操作を行うことで，step1で取り出した故障が，ブロック出力まで経路を活性化できるか判定する．活性化可能な場合はstep3へ進む. 活性化不可能な場合はバックトラックをして別の経路を探索するが，バックトラック制限値を超えた場合，または全経路で活性化が不可能な場合，対象としている故障は冗長故障と判定する．

（step3）
経路が外部出力に到達しているならば，対象とした故障は冗長故障ではないと判定してstep1へ進む．また，分岐先まで未到達の場合は分岐先のブロックを選択してstep2へ.

テストパターンを生成するには，故障箇所から外部出力までの経路を少なくとも１つ活性化することが必要である．したがって，故障箇所から外部出力まで活性化される経路が１本もなければ，その故障は冗長故障だと判定できる．

一方，再収斂ブロックとは，一度分岐して形成されたFFRブロックが, 再び1つの信号線に収斂するような回路構造を１つのブロックとしたものである. 図2に再収斂ブロック分割の例を示す．複数の故障をブロック単位で処理することによって冗長故障判定は高速化され，そのグループ内に含む故障が多い方が学習できる情報が多くなり，効果的であると報告されている[1][6]．回路を再収斂ブロックに分割することで，ブロック内に含む信号線数はFFRのそれよりも多くなる．よって，回路を再収斂ブロック分割することで，FFR分割に比べてより効果的な処理が可能である[6].

2.2　経路活性化の効率化技法

故障箇所から外部出力へ故障を伝搬するためには，多くの部分経路を活性化しなければならない．そのため，効率的に経路活性化を行う必要がある．以下に経路活性化の効率化技法[1]を記す.

(1) ブロッケージ学習
ブロック内の故障の影響を外部出力へ伝搬するための信号線への値の割り当てを学習する.

(2) 動的経路順序付け
・分岐先のブロック内で冗長故障が判定された場合，そのブロックへの分岐先信号線の選択順位を最下位に変更する.
・故障箇所から外部出力までの活性化が成功した場合，通ってきたブロックへの分岐先信号線の選択順位を最上位に変更する.

(3) 故障のグループ化
・上記の(1)，(2)の処理で得られた故障の解析データをブロック内で共有する.
・ブロック内で冗長故障が判定された場合，ブロック内の同様に冗長故障とみなせる故障は冗長故障と判定する.

2.3　従来手法の問題点

REDIでは，活性化経路が外部出力に到達したとき処理を終了する．REDIでは正当化操作を用いていないため，実際に故障の励起や含意操作で割り当てた信号値の組合せが正当化不可能であっても冗長故障ではないと判定されてしまう．したがって，本手法では従来の手法に対してSATを用いた正当化可能判定処理を追加することで，より精度の高い冗長故障の判定を行う.

3 SAT に基づく正当化処理を用いた冗長故障判定

SAT とは和積標準形論理式（CNF：Conjunctive Normal Form）が与えられたときに，それに含まれる全ての変数の値に1（真）または0（偽）を定めることで，全体の値を真にできる割り当てが存在するか否かを判定する問題である.

SATはCNFを入力として問題を解決するため，提案手法では論理回路をCNFに変換する必要がある．そこで3.1で論理回路のCNF変換について述べ，3.2で全体のアルゴリズムを示す.

3.1　論理回路のCNF変換

表1に各論理ゲートのCNF変換規則を示す．各論理ゲートは表1の規則に従いCNFに変換する．それぞれの括弧でくくられた論理和式を節といい，節を論理積した式が各ゲートのCNF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．全体の値が真になるのは（X, Y, Z）= (0, 0, 0), (0, 1, 0), (1, 0, 0), (1, 1, 1)の４通りだけである. これにより, CNFが実際の論理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は各論理ゲートのCNFを論理積することで表現でき，信号線に値を割り当て

表１．論理ゲートのCNF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)
NAND	X Y	Z	(Z+X)･(Z+Y)･(¬X+¬Y+¬Z)
NOR	X Y	Z	(¬Z+¬X)･(¬Z+¬Y)･(X+Y+Z)
NOT	X	Y	(X+Y)･(¬X+¬Y)

る処理では，節のどれかひとつでも偽であった時点で，その割当の組み合わせでは正当化が不可能であることがわかる．したがって，効率的に早期に矛盾を発見でき，高速な正当化可能判定処理が可能である．

3.2　全体アルゴリズム

図3に全体のアルゴリズムを示す．

(step1)

少ないバックトラック制限値でATPGを実行する．この際に得られたテストパターン集合を保存し，打切り故障を冗長故障判定対象故障リストとして保存する．

(step2)

回路全体のCNFを生成する．回路全体のCNFは回路中のすべてのゲートを対象として，表１の規則に従ってCNFに変換し，それらを論理積でひとつのCNFにしたものである．

(step3)

全信号線に対して静的学習を行う．静的学習は各信号線に値が割り当てられた際の，含意関係を信号線ごとに記憶しておく処理である．

(step4)

step1で打切り故障だと判定された故障に対して冗長故障判定を行う．冗長故障判定を行った結果，非冗長だと判定された故障のテストパターンは，故障の影響が外部出力に伝搬した時点での正当化可能判定で決定された外部入力の値から得られるため，それらをstep1で得たテストパターン集合に追加する．

冗長故障判定の詳細なアルゴリズムは3.3に記す．

3.3　冗長故障判定アルゴリズム

図4に冗長故障判定処理のアルゴリズムを示す．

(step1)

すべての対象故障に対して判定を行ったら処理を終了する．未判定の故障が存在するときは故障を１つ選択してstep2へ進む．

(step2)

step1で取り出した故障を励起するために，故障を活性化するために必要な制約となるCNFを制約CNFとして初期化・保存する．

(step3)

分割ブロックの出力まで故障の影響を伝搬する．含意操作を行い，割り当てた値が矛盾しなければstep4へ進む．その際，活性化経路上のゲートの経路外入力に割当てた非制御値と含意操作で割当てた値を制約CNFとして，制約CNFとする．含意操作で値の衝突が生じた場合はstep5へ進む．

(step4)

全体アルゴリズムのstep2で生成した回路全体のCNFとstep3で生成した制約CNFを論理積で１つのCNFにする．結合したCNFを入力として充足可能性問題を解き，正当化可能判定を行う．正当化可能ならstep6へ進む．正当化不可能ならstep5へ進む．

(step5)

活性化を試みていない経路が存在しない場合は対象の故障を冗長故障と判定してstep1へ進む．またバックトラックの回数が制限値を超えた場合は対象の故障を打切り故障と判定してstep1へ進む．それらの条件を満たしていない場合は，活性化を試みる経路を選択してstep3へ進む．

(step6)

活性化経路が外部出力に到達してない場合は分岐先から次の経路を選択してstep3へ進む．活性化経路が外部出力まで到達した場合は，正当化可能判定処理で得た，外部入力の割り当て値を対象故障のテストパターンとしてテスト集合に追加し，step1へ進む．

例として図4を用いて冗長故障判定を示す．信号線d-eに0縮退故障を仮定する．故障を励起するためにd-eに1を割当てる制約を制約CNFとして保存する．次に故障の影響を伝搬するためにbに１を割当てる制約を制約CNFに追加する．含意操作でも値の割当ては正当であり，故障の影響が分岐先eに到達したため，前処理で生成した回路全体のCNFと制約CNFをひとつのCNFとし，これを入力としてSATを解くことで正当化可能判定を行う．本例では充足可能と判定される．活性化経路が外部出力に到達していないため処理を続行する．次に経路はe-fを選択する．故障の影響を伝搬するためにaに1, g-iに1に割り当てるという制約を制約CNFに追加更新して，SATを解くことで正当化可能判定を行う．ここで充足可能と判定されたとき，外部出力まで活性化経路が到達したため対象の故障は冗長故障ではないと判定する．また正当化判定で割り当てた外部入力の値を信号線d-eの0縮退故障のテストパターンとして保存する．

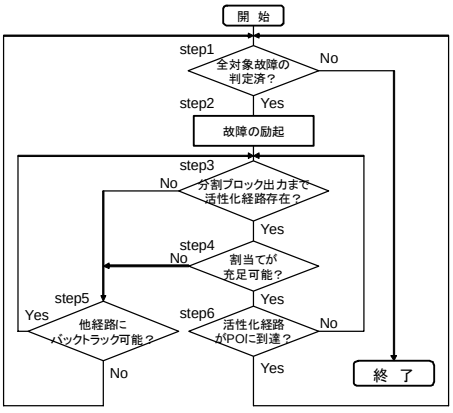

図３．冗長判定処理アルゴリズム

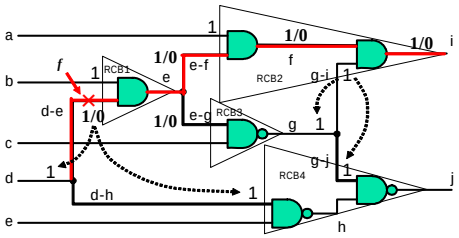

図４．冗長故障判定例

４　実験結果

実験として，提案手法を実装し，提案手法の各処理に要するCPU時間を評価した．また，Synopsys社のテスト生成ツールであるTetra MAXと提案手法で，検出困難故障に対する冗長故障判定結果を比較した．実験環境は，OSはPentium4 ，メモリは２GB ，OSはRedHatである．実験対象としてITC99 ベンチマーク回路のb14，b15，b17，b21，b22を用いた．また，検出困難故障を得るために行うテスト生成はSynopsys社のTetra MAXを用いた．この際のバックトラック制限値は300である．

表２は各処理に要したCPU時間である．列は左から回路名，処理の対象となる検出困難故障の数，ATPG時間，リスト読込みなどの前処理時間，静的学習時間，冗長故障判定時間，全体の処理時間である．括弧内の値は静的学習を用いなかったときに要した処理時間である．表2から対象となる検出困難故障が少ないb15とb17以外の回路では静的学習を用いないほうが高速に処理できることを確認した．

表３は検出困難故障に対するTetra MAXのバックトラック制限値10000000のテスト生成と提案手法の比較である．列は左から回路名，処理の対象となる検出困難故障数，等価故障解析後の検出困難故障数，打切り故障数，CPU時間，CPU時間の削減率である．また要素の左の値は提案の手法結果，右の値はTetraMAXの結果である．

TetraMAXではバックトラック制限値を10000000で設けても，b15以外の回路で打切り故障が残ったが，提案手法では対象の全回路で打切り故障数が０の故障検出効率100%を達成した．またバックトラック制限値10000000のTetraMAXのテスト生成に対して，提案手法を用いることでより多くの故障を判定しながら，全ての対象回路で高速化に成功し，平均して94%の時間の削減に成功した．

５　おわりに

本稿では，SATを用いた検出困難故障に対する冗長故障判定・テスト生成方法の提案，および評価実験を行った．実験結果から本手法を用いることで，ATPGツールのみを用いるより高速に検出困難故障を同定し，故障検出効率100%が達成できたことを確認した．今後は，各処理の高速化や，より大規模な回路での実験・評価を行う予定である．

「参考文献」

[1]Chen Wang,Irith Pomeranz and Sudhakar M.Reddy, ”REDI(An Efficient Fault Oriented Procedure to Identify Redundant Faults in Combinational Logic Circuits)” , IEEE/ACM international conference on Computer-aided design,Nov,2001 ,pp. 370-374

[2]Tracy Larrabee , “Test Pattern Generation Using Boolean Satisfiability”, IEEE TRANSACTION ON COMPUTER-AIDED DESIGN ,VOL.11,No.1,Jan, 1992, pp. 4-15

[3]Paul Stephan , Robert K.Brayton , and Alberto L.Sangiovanni-Vincentelli , “Combinational Test Generation Using Satisfiabillity”, IEEE TRANSACTION ON COMPUTER-AIDED DESIGN, VOL.15,No.9 ,Sep, 1996, pp. 1167-1176

[4]Paul Tafertshofer,Andreas Ganz “SAT Based ATPG Using Fast Justification and Propagation in the Implication Graph” , IEEE/ACM international conference on Computer-aided design,Nov,1999, pp. 139-146

[5]M.Henftling, H.Wittmann and K.Antreich, “A Single-Path-Oriented- Fault-Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization,” Proceedings of ICCAD , Nov.1995 , pp. 304-309

[6]日本大学生産工学部数理情報工学科 2007年度卒業研究，清水永吉，“再収斂ブロック情報を用いた冗長故障判定アルゴリズムの高速化”．

表２．各処理CPU時間

回路名	故障数	ATPG	前処理	静的学習	冗長故障判定	合計
b14	1	5.81	0.35	78.67 (0)	3.88 (1.33)	88.71 (7.49)
b15	181	41.27	0.89	147.49 (0)	64.84 (664.78)	254.49 (706.94)
b17	438	120.40	4.82	1193.95 (0)	342.04 (1896.21)	1661.21 (2021.43)
b21	2	12.83	1.02	347.35 (0)	8.32 (2.67)	369.52 (16.52)
b22	2	18.51	1.88	739.02 (0)	16.18 (4.05)	775.59 (24.44)

表３．TetraMAXと提案手法の比較結果

回路名	#faults	#rep_faults	#not detected	Total CPU time	time reduction(%)
b14	1	1	0 / 1	7.49 / 1850.46	99.81
b15	181	114	0 / 0	254.49 / 3697.88	93.12
b17	438	257	0 / 2	1661.21 / 9423.16	82.37
b21	2	2	0 / 2	8.32 / 2919.99	99.72
b22	2	2	0 / 1	16.18 / 566.25	97.14