ISSN 2186-5647

2-15

並列含意操作を用いた静的学習の高速化

日大生産工（学部）　○秋山　正碩　　日大生産工（院）　山崎　紘史
日大生産工　　細川　利典

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 circuit：LSI)が大規模化・複雑化し，テスト生成時間の増加が問題となっている．そのため，テスト生成(1)の高速化が求められている．テスト生成とは LSI に仮定した故障モデルを検出できるテストパタンを生成することである．

テスト生成を効率化する技術の一つとして，SOCRATES(2)で提案された静的学習(2)(3)がある．静的学習は含意操作の結果に学習規則(2)(3)を適用し，間接含意を求める処理である．テスト生成の前処理段階で静的学習により間接含意を求め，テスト生成中の含意操作で間接含意を利用する．間接含意の利用は矛盾の早期発見につながり，テスト生成処理を高速化することが可能である．

前述のように静的学習はテスト生成の解の探索空間の削減に必要不可欠な技術である．しかしながら，近年の LSI の大規模化に伴い，静的学習の処理時間の増加が問題となり，静的学習の高速化が必要となる．本論文では含意操作を高速化するための技術である並列含意操作(4)(5)を用いる．並列含意操作において，信号線の値は計算機の 1 語の各ビットに対応づけて表現される．計算機の整数型 1 語が 32 ビットである場合，最大で 32 本の信号線の値を同時に保持できる．各ゲートの演算は同じ位置のビットごとに論理演算を行うビット演算で含意操作を一度に行うことが可能である．

本論文では文献(4)(5)で提案された並列含意操作を用いて静的学習を高速化する方法を提案する．静的学習を高速化することで繰り返し静的学習を実行することができ，より多くの間接含意を取得することが可能となる．その結果，テスト生成の高速化が期待できる．

本論文の構成は以下の通りである．第 2 章では含意操作について述べる．第 3 章では静的学習について述べる．第 4 章では並列含意操作について述べる．第 5 章では信号線のグループ化について述べる．第 6 章では実験結果を示し，第 7 章では本論文についてまとめる．

2. 含意操作

含意操作はゲートの入出力の接続関係から値を求めることのできる直接含意(2)(3)と，ゲートの入出力の接続関係だけでは値を求めることのできない間接含意(2)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 1 に 2 入力 1 出力 AND ゲートに対する直接含意の例を示す．図 1(a)は前方含意の例である．信号線 a の値が 0 である場合，AND ゲートの入力に 0 が割当てられたとき出力は 0 となるため，信号線 c の値は 0 と決定できる．このようにゲートの入力側から出力側に値を決定していく処理が前方含意である．図 1(b)は後方含意の例である．信号線 c の値が 1 である場合，AND ゲートの出力を 1 にするには全ての入力が 1 でなくてはならないため，信号線 a,b の値は 1 と決定できる．このように出力側から入力側に値を決定していく処理が後方含意である．

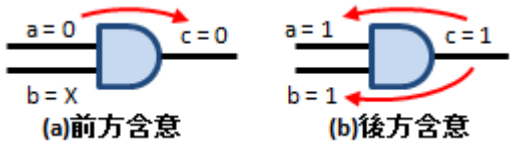

図 1. 直接含意の例

On the Acceleration of Static Learning Using Parallel Implication Techniques

Masahiro AKIYAMA, Hiroshi YAMAZAKI, and Toshinori HOSOKAWA

図 2 に間接含意例を示す．信号線 f の値を 1 とした場合，直接含意では信号線 d または信号線 e のいずれかの値が 1 となり，一意に値を決定できる信号線はない．しかし信号線 d を 1 に決定する場合と信号線 e を 1 に決定する場合のいずれの場合も，信号線 b が 1 でなければならないため，信号線 f の値が 1 ならば信号線 b の値は 1 と決定できる．このように直接含意で決定できない含意操作が間接含意である．

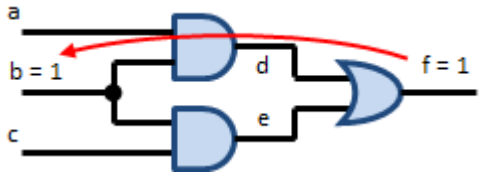

図 2. 間接含意の例

3.静的学習

静的学習とは直接含意の結果から間接含意を求める処理で，繰り返し実行することでより多くの間接含意を得られる．静的学習では対偶規則と固定値学習の 2 つを用いる．

3.1. 対偶規則

対偶規則とはある命題が真のとき，対偶も新であるという規則である．図 3 に対偶規則の例を示す．始めに信号線 b の値を 0 とし前方含意を行うと信号線 f の値が 0 となる．これより”b=0 ならば f=0”という命題が真になる．対偶規則を適用すると”f=1 ならば b=1”という命題が成り立ち，間接含意を得ることができる．

3.2. 固定値学習

固定値学習とは含意操作中に衝突が発生した際に固定の値を決定するものである．図 4 に固定値学習の例を示す．信号線 d の値を 1 とし広報含意を行うと，信号線 b の値が衝突することがわかる．そのため信号線 d の値は 0 に固定される．

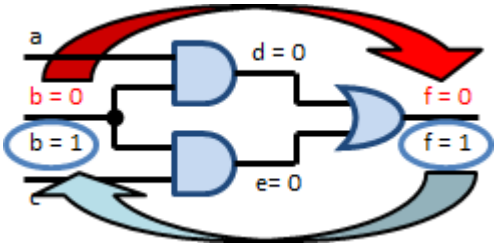

図 3. 対偶規則の例

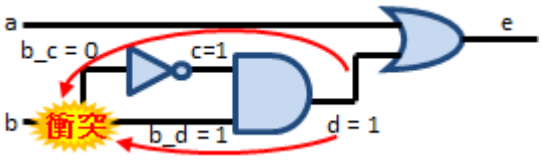

図 4. 固定値学習の例

4.並列含意操作

並列含意操作は含意操作のゲートごとの演算を，ビット演算を用いて並列化することである．静的学習の含意操作は 1,0,X(未割当て)の 3 値を用いて行われる．並列含意操作では，この 3 値を計算機の 2 語の各ビットに対応付けて表す(4)．ある信号線 A に対する 1,0,X のビット表現(A0,A1)をそれぞれ(0,1),(1,0),(0,0)とする．このとき各ゲートの演算を図 5 に基づいて行うことができるので，各ゲートの演算を並列化できる．

含意操作では信号線の論理値を 1 つ保持するのに対し，並列含意操作では信号線の論理値を複数保持することで，信号線の異なる値の含意操作を同時に行う．この同時に含意操作を行う信号線の集合を含意グループと定義する．図 6 の並列含意操作の例では，信号線 a の論理値と信号線 b の論理値を並列に処理して信号線 c の論理値を得ている．並列に処理しない場合は含意操作が 3 回必要だが，並列に処理した場合は含意操作が 1 回となり，含意回数を削減できる．

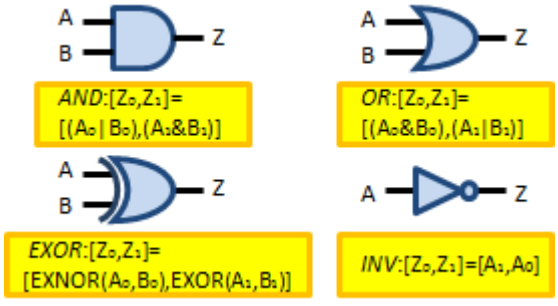

図 5. ゲートごとのビット演算

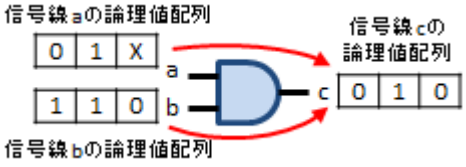

図 6. 並列含意操作のビット演算例

5.信号線のグループ化

本論文では信号線のグループ化を評価する．並列含意操作においては，どの信号線を並列に処理するかが重要である．ある信号線と並列に処理を行う信号線は，故障の影響範囲が同じだとよいと考えられる．図 7 に故障影響範囲の例を示す．図 7 の信号線 1,b の故障を出力まで伝搬する場合，斜線部から out2,3 に故障が出力される．次に out2,3 に影響を与える信号線の範囲は格子部となる．この斜線部と格子部が故障影響範囲である．

本論文では対象回路内のすべての信号線を FFR(Fan-out Free Region)にする．FFR にする理由は FFR と故障影響範囲が等しいからである．図 8 に FFR の例を示す．FFR は外部出力信号線または分岐元信号線(Fan-out stem)から入力側に回路を追跡し，外部入力信号線または分岐先信号線(Fan-out branch)に到達するまでに存在する信号線すべての集合である．図 8 の場合はすべての信号線を 3 つの FFR に分けられる．

次に FFR を深さ優先探索によりグループ化することで含意グループを作成する．深さ優先探索を行う理由は，深さ優先探索による信号線が故障影響範囲内に存在するからである．図 9 に深さ優先探索の例を示す．図 9 の信号線 a は信号線 b の故障影響範囲内に存在しており，並列に処理しやすいと考えられる．しかし故障影響範囲内であってもレベルの差が大きい場合は並列に処理しづらくなる．図 10 にレベルの差が大きい場合の例を示す．図 10 の信号線 a は信号線 b の故障影響範囲内に存在しているが，信号線 a の故障影響範囲は信号線 b の故障影響範囲より大きいことがわかる．故障影響範囲の差が大きい場合，並列に処理しづらくなる．

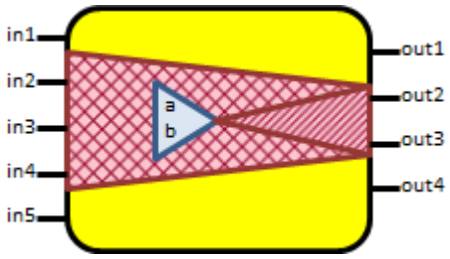

図 7.故障影響範囲の例

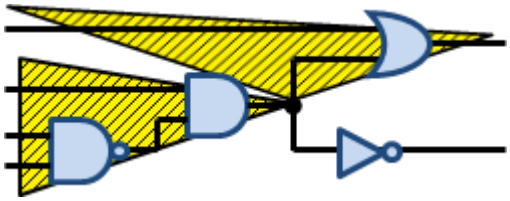

図 8.FFR の例

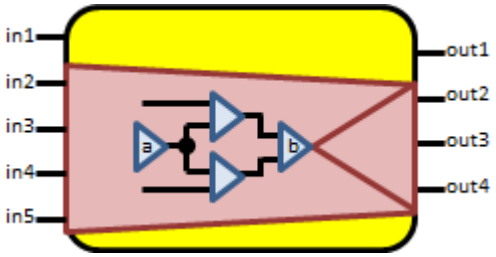

図 9.深さ優先探索の例

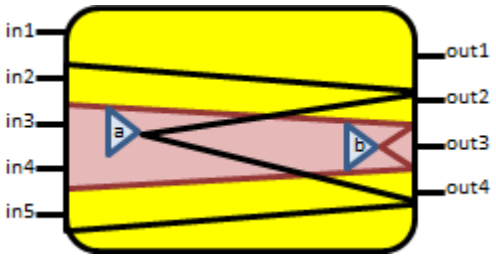

図 10.レベルの差が大きい場合の例

6. 実験結果

FFR を基にした深さ優先の信号線のグループ化を実装し，ISCAS'89 ベンチマーク回路と ITC'99 ベンチマーク回路に対し含意回数の評価を行った．

表 1 は FFR の信号線グループで並列含意操作を行った場合と，提案手法の含意グループで並列含意操作を行った場合の，含意操作回数と並列含意操作を行うグループ数を比較した表である．含意操作回数は含意グループが行った前方含意，広報含意の総数である．この表から提案手法が FFR の信号線グループに対し，含意操作回数が平均 16.88%，含意グループ数が平均 69.35%削減できたことがわかる．

表 2 は FFR の信号線グループで並列含意操作を行った場合と，提案手法の含意グループで並列含意操作を行った場合の各処理時間の比較をした表である．"グループ化"は各グループの作成にかかった時間，"含意操作"は並列含意操作の時間，"CPU"は全体の時間である．表 2 から含意操作が平均 21.33%，CPU が平均 3.02%削減できたことがわかるが，

s38417 と b19 においてグループ化に要する増加時間が，含意操作で削減した時間より大きくなったため，全体の時間が増加したことがわかる．

7. おわりに

本論文では順序回路の静的学習の高速化を行うために，回路情報を基にした新合繊のグループ化を提案し実装した．実験結果から含意操作回数と含意グループ数の削減による並列含意処理の高速化に成功し，ほとんどの回路においてグループ化にかかる時間の増加による，全体の実行時間が増加してしまった．今後はグループ化の時間を削減することによるさらなる高速化を考える

表 1. 含意総数とグループ数の比較

回路		FFR	提案手法	削減率
s13207	含意総数	2021916	1664181	17.69
	グループ数	2764	936	66.14
s15850	含意総数	2255977	1794529	20.45
	グループ数	3352	1096	67.30
s35932	含意総数	39204696	28199454	28.07
	グループ数	11230	2714	75.83
s38417	含意総数	2765029	2543800	8.00
	グループ数	9436	3308	64.94
s38584	含意総数	39135377	35532209	9.21
	グループ数	8724	2968	65.98
b17	含意総数	79980216	64768430	19.02
	グループ数	17662	5290	70.05
b19	含意総数	944475336	685212631	27.45
	グループ数	135234	36808	72.78
b20	含意総数	13398425	11498742	14.18
	グループ数	10124	3094	69.44
b21	含意総数	15686432	13413366	14.49
	グループ数	10070	3098	69.24
average	含意総数			16.88
	グループ数			69.35

表 2. 実行時間の比較

回路		FFR	提案手法	削減率
s13207	グループ化	0.01	0.2	
	並列含意	1.57	1.29	17.94
	CPU	1.91	1.9	0.42
s15850	グループ化	0.02	0.22	
	並列含意	1.74	1.42	18.30
	CPU	2.14	1.98	7.39
s35932	グループ化	0.05	1.72	
	並列含意	33.22	23.7	28.65
	CPU	34.05	26.16	23.18
s38417	グループ化	0.04	1.81	
	並列含意	2.36	2.01	14.76
	CPU	3.26	4.68	-43.38
s38584	グループ化	0.05	1.2	
	並列含意	43.54	31.32	28.07
	CPU	44.44	33.31	25.05
b17	グループ化	0.09	4.35	
	並列含意	76.51	53.26	30.39
	CPU	78.44	59.58	24.05
b19	グループ化	0.64	604.81	
	並列含意	422.97	610.3	30.52
	CPU	428.96	1228.59	-37.59
b20	グループ化	0.052	1.53	
	並列含意	11.11	9.48	14.67
	CPU	12.056	11.87	1.54
b21	グループ化	0.052	1.53	
	並列含意	13.16	11.06	15.96
	CPU	14.174	13.45	5.11
average	並列含意			21.23
	CPU			3.02

参考文献

1) H. Fujiwara , “Logic Testing and Design for Testability”, The MIT Press, (1995).

2) MICHAEL H. SCHULZ, ERWIN TRISCHLER, “SOCRATES : A Highly Efficient Automatic Test Generation System”, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, VOL 7 NO 1, JANUARY (1988) p130.

3) 梶原誠司 and 猿渡慶太郎 “静的学習における効率的な間接含意の発見と保存について” 電子情報通信学会 TECHNICAL REPORT OF IEICE (2009) pp25-28

4) 南山哲郎 高松雄三 “組合せ回路における冗長故障の判定に関する考察” 電子情報通信学会論文誌 (1998) pp1149-1156

5) Mahesh A, Iyer and Miron Abramovici “FIRE ; A Fault-Independent Combinational Redundancy Identification Algorithm” IEEE TRANSACTIONS ON VERY LARGESCALE INTEGRATION (VLSI) SYSTEMS, VOL 4 NO 2 (1996) pp295-301