設 計 自 動 化 83－1
（1997．2．14）

パス遅延故障のテストのためのロバスト依存パスの識別法

梶原 誠司† 樹下 行三†† イリス ポメランツ††† スダーカ M. レディ†††

†九州工業大学 情報工学部 電子情報工学科
††大阪大学 工学研究科 応用物理学専攻
†††アイオワ大学 電気コンピュータ学科

あらまし

製造した回路のタイミング検証の重要性が高まっている一方で，ロバスト依存パスや機能的活性化不能パスのようなパス遅延故障のテストにおいてテスト不要であるパスを多く含む回路が存在することがわかってきた．テスト不要なパスをテスト生成の前に指摘し，テスト生成の対象としないことは，パス遅延故障のテスト生成の効率化に役立つ．本論文は，高速にロバスト依存パスと機能的活性化不能パスを識別する手法を提案する．提案手法は回路の局所的な解析に基づくため，従来多大な処理時間を要していたパス数の多い回路も高速に処理可能である．実験では，本手法は従来手法と比較して短時間で計算でき，テスト不要なパスの判定能力はほぼ同等であることを示す．また，従来手法ではパス数が 10^{20} 以上ある *c6288* を現実的には扱うことができなかったが，本手法により 99%以上のパスはテスト不要であることを示す．

A Method for Identifying Robust Dependent Paths for Path Delay Fault Testing

Seiji Kajihara† Kozo Kinoshita†† Irith Pomeranz††† Sudhakar M. Reddy†††

†Computer Science and Electronics Dept., Kyushu Institute of Technology
††Applied Physics Dept., Osaka University
†††Electrical and Computer Eng. Dept., University of Iowa

Abstract

It has been shown previously that a logic circuit often contains a large number of logical paths that need not be tested to verify the timing behavior of the circuit. For example, robust dependent paths and functionally unsensitizable paths are known as unnecessary-to-test paths. This paper proposes a method for efficiently identifying both types of paths. The proposed procedure uses local circuit analysis to keep the run time relatively low, and relatively independent of the number of paths in the circuit. Experimental results show that the numbers it finds are comparable, and sometimes even higher, than those found by other methods. The procedure can be applied to circuits such as c6288 that cannot be handled by other methods.

1．はじめに

論理システムの高速化・高信頼化の要求により，製造された回路の遅延故障に対するテストの重要性が高まっている．遅延に関する故障モデルとしては，ゲート遅延故障モデル[1]とパス遅延故障モデル[2]が提案されている．これら2つの故障モデルのうち，パス遅延故障モデルは，外部入力の信号が外部出力に至る経路を伝搬する際に蓄積する遅延時間の総和をモデル化しているので，製造された回路がタイミング制約を満たすか否かを確かめることにより適している．

最近の研究で，タイミング制約を満たすか否かを検査するときテストする必要がないパスを多く含む回路があることがわかってきた[3~6]．そのようなパスとして，[5] では，ロバスト依存パス (RDパス: robust dependent path) と呼ばれる概念が導入された．RDパスの特徴は，RDパス以外のパスに遅延故障が無いなら，RDパス上のパス遅延故障は回路の遅延を増加しないことである．従って，RDパス以外のパス遅延故障がロバストにテストされるなら，RDパス上のパス遅延故障はテスト不要である．また，[4] では，機能的活性化不能パス (FUパス: functionally unsensitizable path) の概念が導入された．FUパスは，他のパスのテスト可能性にかかわらずテスト不要であり，RDパスの集合の一部となっている．

FUパスとRDパス集合を見つける手法は，[3~6] で提案されている．[4] の手法は，一つ一つのパスを順々に活性化可能か否かを判定する手法[6]を用いて部分的にFUパスとなるFUセグメントを見つけ，FUセグメントを含むパスを以降の判定の対象としないことにより処理を効率化する．[3] は，[4] の処理手順をRDパスに適用できるよう工夫している．これらの手法は，活性化の可能性の判定に必要割当 (necessary assignments)[8,9,11] を用いるため，識別できるRDパスとFUパスは，実際のRDパスやFUパスの部分集合である．また，一つ一つのパスを順々に活性化可能か調べるため，計算時間は回路内に含まれるパス数とともに増加し，ISCAS'85 のベンチマーク回路[12]の *c6288* のようにパス数が 10^{20} 以上もある回路には現実的に適用不可能である．

本論文では，回路のパス数に依存しない計算時間で FUパスおよびRDパスを判定する手法を提案する．提案手法は，まず，各分岐枝からその分岐枝を含むファンアウトフリー領域の出力までのパスを活性化するための必要割当を求める．次に，この必要割当の解析から，b-f ペアと呼ぶ信号線のペアを求める．b-f ペアを構成する2つの信号線を含むパスは活性化できない，つまり，RD(FU)パスになる．本論文では，得られた b-f ペアからRD(FU)パスを計算する2つのアルゴリズムも提案する．1つは，b-fペアを含むパスの数を正確に数えるアルゴリズムであるが，回路が非常に多くの再収れん構造を持つ場合計算に大量の記憶空間を必要とする．他の一つは，[7] のパス数計算手法に基づいて，b-f ペアから求められるRD(FU)パス数の下界を求める近似アルゴリズムである．計算に必要な記憶空間が回路規模に比例する程度でよいため，パス数が多い回路にも適用可能である．

本手法は，[3], [4], [6] の手法とは異なり，特定の長さの部分パスしか活性化しないため，パスの活性化に要する時間が短くなる利点を持つが，識別できるFUパスやRDパスの数は少なくなる．しかしながら，静的学習 [10] による間接含意を利用して必要割当をより多く求め，識別能力を高めている．実験では，[3], [4] の手法と比較して，計算時間は非常に短く，その識別能力はほぼ同等であることを示す．また，従来は扱うことができなかった *c6288* に対しても，HP735 ワークステーションで約70秒で処理可能であり，99%以上のパスはテスト不要であることを見つけている．

本論文で提案する FUパスの識別手法は，[3] の lemma 2 によりRDパスに適用できるので，本論文は提案手法の説明をFUパスに限定して次のように構成する．2章では，本文中で用いる用語を定義し，本手法の基礎となる考え方を述べる．3章では，FUパスの判定の基礎となる b-fペアの見つけ方を述べる．4章では，b-fペアからFUパス数を数えるアルゴリズムを述べる．5章でベンチマーク回路に対する実験結果を示し，6章で本論文のまとめを行う．

2．準備

2．1　諸定義

物理パス $P=(l_0, l_1, \cdots, l_m)$ は，外部入力 l_0 から外部出力 l_m に至る信号線の系列である．分岐の枝

は，分岐の幹の信号線と別の信号線とする．論理パス Px は，外部入力 I_0 において信号の遷移 $x \in$ {rising, falling} を持つ物理パス P のことである．信号線 I_i ($0 \le i \le m$) は，P のパス上入力 (on-path input) と呼ばれる．パス上入力 I_i が入力しているゲートのファンイン信号線で，パス上入力でない信号線を，Px のパス上入力 I_i に関するパス外入力 (off-path input) という．ゲート入力の制御値とは，そのゲートの出力の値を決めることができるゲート入力の信号値のことである．例えば，信号値0は，AND および NAND ゲートの入力の制御値である．制御値の否定を非制御値という．

論理パス Px のパス上入力 I_i は，遷移 x_i を持つ．信号線 I_0 から I_i までのパス上にある否定系ゲート（NOT, NAND, NOR ゲート）の数が偶数個のとき，$x_i = x$ であり，そうでないとき x_i は x と逆の遷移となる．論理パス Px のパス上入力の最終的な値が非制御値であるどのゲートのパス外入力にも非制御値を割当る入力ベクトルが存在するとき，Px は機能的活性化可能 (functionally sensitizable) なパスであるという．Px を機能的活性化する入力ベクトルが存在しないとき，Px を機能的活性化不能パス(func-tionally unsensitizable path) といい [4]，以後 FUパスと表す．Px を機能的活性化するどの入力ベクトルに対しても，信号線 l の値が $v \in \{0, 1\}$ となるとき，v は Px を機能的活性化するための必要割当と呼ばれる[6]．

本論文で提案する手法は，部分パスと呼ぶパスを用いるが，それは次のように定義される．部分物理パス $PP = (I_s, I_{s+1}, \cdots, I_t)$ は，外部入力または分岐の枝 I_s から外部出力または分岐の幹 I_t に至る信号線の系列であり，その間の信号線 I_i ($s < i < t$) は分岐の枝を含まない．部分物理パス PP は始点の信号線 I_s だけから特定できることを記しておく．部分論理パス PPx は，I_s において遷移 $x \in$ {rising, falling} を持つ部分物理パス PP のことである．部分パスとの区別を明確にするため，外部入力から外部出力までのパスを完全パスと呼ぶ．

部分パスの数は，回路の信号線数や完全パスの数より少ないことは明らかである．また，再収れん構造を多くもつ回路では部分パス数と完全パス数の差は増大する．例えば，図1の点線で囲まれた回路ブロックが n 個直列に接続されている回路を考える．

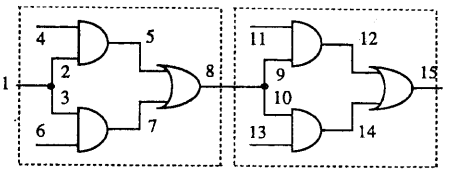

Fig. 1: Example circuit

$n \ge 2$ のとき，回路の信号線数は $8n - 1$，完全物理パス数は $3 \times 2^n - 2$，部分物理パス数は $4n$ である．

[4, 6] における FUパスの検出は，パスを活性化するための必要割当に矛盾が生じるまで，外部入力側から活性化するパスの長さを徐々に延ばしていく．従って，その計算時間は完全パス数に大きく依存する．一方，本論文で提案する手法は，部分パスを活性化する必要割当のみを求めるので，その計算時間は完全パス数には依存しない．従って，本手法は計算時間の点で従来手法を改善できる．以降では，FUパスを見つける新しい手法を述べるが，ロバスト依存パス (RDパス: robust dependent path) を見つける手法に容易に拡張可能である．

2．2　提案手法の概略

以下では，Px を機能的活性化するための必要割当のことを，単に Px の必要割当と呼ぶ．提案手法は，次の補題に基づく．

[lemma 1]　信号線 I_i の論理値0（1）が，I_j から遷移 x で始まる部分論理パス PPx の必要割当であるとする．このとき，I_i での立ち上がり（立ち下がり）と I_j での遷移 x を含むすべての完全論理パスは，FUパスである．

証明：I_j での遷移 x を含むすべての完全パスは，部分論理パス PPx を含むので，I_j での遷移 x を含むどの完全パスを機能的活性化するにも，PPx を機能的活性化しなければならない．PPx を機能的活性化するには，信号線 I_i の論理値を0（1）にしなければならない．しかしながら，I_i の立ち上がり（立ち下がり）の遷移を含むどの完全論理パスを機能的活性化するためにも，I_i の論理値を1（0）にする入力ベクトルが必要である．これは，I_i での必要割当0（1）に矛盾する．よって，lemma 1 は成立する．（証明終り）

今，信号線 I_i から I_j へのパスは存在するが，I_i での遷移 x と I_j での遷移 y を持つどの完全論理パスもFUパスとなるような信号線 I_i と I_j を考える．このと

き，信号線 l_i の遷移 x を b-line (back line) と呼び，信号線 l_j の遷移 y を f-line (forward line) と呼ぶ．各 b-line と f-line の位置と遷移から，FUパスが求められるが，b-line と f-line の組合せを b-fペアと呼ぶ．

lemma 1 から，l_j から外部出力に至るパス上に l_i が存在するならば，l_j の遷移 x を b-line とし，l_i の立ち上がりを f-line とする b-f ペアが得られる．また，l_i が l_j から外部入力に至るパス上にある場合，l_j の遷移 x が f-line に，l_i の立ち上がりが b-line となる．本論文で提案する FUパスの識別手法は，部分論理パスの必要割当から見つけられる b-fペアを用いるため，完全パス数の非常に多い回路にも現実的な時間内で計算可能となる．しかしながら，活性化するパスの長さが短くなるため，識別できる FUパス数は [4, 6] の手法より少なくなる．本手法では，その欠点を補うために静的学習[10]による間接含意を必要割当の計算に用いる．

3．b-fペアの算出

本手法では，b-fペアの計算に部分パスの必要割当を用いるが，得られる b-fペアの数が多ければ多いほど，識別できる FUパス数も多くなる．そこで，より多くの b-fペアを求めるため，lemma 1 を拡張し，更に，2つ以上の部分パスを活性化して b-fペアを見つける手法を述べる．また，識別できる FUパス数は維持したまま，b-fペアを圧縮する手法および不要な b-fペアを削除する手法について述べる．

3．1　必要割当の比較

lemma 1 では部分論理パス PPx とその必要割当（信号線 l_i の論理値 0）から b-fペアを見つけることを考えた．ここで，信号線 l_i の論理値 1 は，信号線 l_i の立ち上がり遷移を含む部分論理パスの必要割当である．従って，2つの部分論理パス（この場合，PPx と信号線 l_i の立ち上がりを含む部分論理パス）が互いに矛盾する必要割当を持てば，それらの部分論理パスを含む完全論理パスは FUパスになる．つまり，次に述べるように，2つの部分論理パスの必要割当の比較から，b-fペアが得られる．

[lemma 2]　部分論理パス PPx と PQy に関して，信号線 l_i の論理値 $v \in \{0, 1\}$ が PPx の必要割当であり，かつ，信号線 l_i の $\overline{v}$ が PQy の必要割当であるとき，PPx と PQy を含むすべての完全論理パスは，FUパスである．

部分論理パス PPx と PQy に対して，lemma 2 の条件が満たされるならば，PPx と PQy の始点の信号線により b-fペアを構成できる．また，lemma 1 により識別できる FUパスは，lemma 2 でもすべて識別できる．しかしながら，lemma 2 で識別されるすべての FUパスを，lemma 1 で識別できるとは限らない．その例を図 2 の回路例で示す．l_a で立ち上がり遷移を持つ部分論理パス $PPx = (l_a, l_b, l_c)$ と l_e で立ち上がり遷移を持つ部分論理パス $PQy = (l_e, l_f, l_g)$ を考える．PPx と PQy の必要割当は図 3 のようになるが，信号線 l_d で必要割当の矛盾が生じるので，lemma 2 から l_a と l_e の b-f ペアを得ることができる．しかしながら，lemma 1 からこの b-fペアを得ることはできない．なぜなら，PQ 上の信号線に PPx の必要割当はなく，また，PP 上の信号線に PQy の必要割当がないからである．結果として，lemma 1 から PPx と PQy を含む完全論理パスが FUパスであると認識できない．このように，lemma 1 を拡張した lemma 2 はより多くの b-f ペア（つまり FUパス）を見つけることができる．

3．2　拡張必要割当

本手法では，b-fペアに基づいて FUパスを識別するため，多くの b-fペアを見つければ見つけるほど，識別できる FUパスの数も増えてくる．そこで，よ

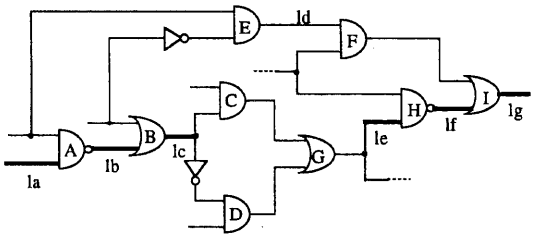

Fig. 2: Example circuit

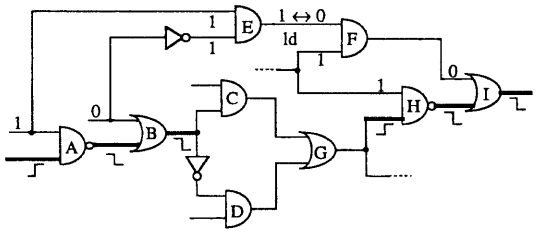

Fig. 3: Necessary assignments for PPx and PQy

り多くのb-fペアを見つけるため，部分パス PPx の必要割当を用いる代わりに，PPx を含むすべての完全パスに共通の必要割当を考える．そのような必要割当を拡張必要割当と呼ぶことにする．

[lemma 3] PPx と PQy を部分論理パスとする．信号線 l の論理値 $v \in \{0, 1\}$ が PPx を含むどの完全パスの必要割当にもなっており，かつ，信号線 l の $\bar{v}$ が PQy の必要割当であるとき，PPx と PQy を含むすべての完全論理パスは，FUパスである．

証明：信号線 l での必要割当の矛盾より，PPx と PQy を含むすべての完全論理パスを機能的活性化する入力ベクトルは存在しないことは明らかである．従って，PPx と PQy を含むすべての完全論理パスはFUパスである．（証明終り）

PPx を含むすべての完全パスに共通の必要割当をすべて求めることは現実的でないので，本手法では PPx に直接接続している部分パスのみを対象に PPx の拡張必要割当を求める．

今，分岐の枝 $br_1, br_2, \ldots, br_n$ を持つ分岐の幹の信号線 l_t が終点となる部分パス $PPx = (l_s, \cdots, l_t)$ と，信号線 l_t における遷移 y を考える．l_t のすべての分岐の枝 br_i に対して，ある信号線 g 上の信号値 v が br_i の遷移 y で始まる部分パスの必要割当であるなら，g の信号値 v は部分パス PPx の拡張必要割当である．同様に，部分パス PPx の始点である信号線 l_s の遷移 x に対して，l_s の分岐の幹の遷移 x が終点となるすべての部分パスに共通の必要割当も，PPx の拡張必要割当となる．もし PPx が信号線 br_i の遷移 y で始まる部分パスと同時に機能的活性化不可能なら，l_s が b-line で br_i が f-line となることは明らかである．

また，PPx が信号線 br_i の遷移 y で始まる部分パスと同時に機能的活性化可能であったとしても，その必要割当が外部出力までのどの論理パスの活性化も阻止するなら，l_s と br_i が b-fペア となる．この考え方は，PODEM[8] におけるXパスチェックと呼ばれる考え方と同様である．更に，PPx を含むすべての完全パスが PP 上にない信号線 l の立ち上がり（立ち下がり）遷移を必ず含むとき，信号線 l の最終的な値が非制御値であるならそのパス外入力に同じ非制御値を割り当てることができる．これは，FAN[9] における一意経路活性化と呼ばれる考え方と同様である．

拡張必要割当を求めるには通常の必要割当より多くの時間を要するため，本手法では，本節で述べた拡張必要割当を求める手法をプログラムのオプションとして用意する．

3．3　b-f pair の圧縮

b-fペアからFUパスを数える前にいくつかのb-fペアを統合し，その数を少なくすることでFUパスの計算時間を圧縮する．例えば，I_1 と I_2 を入力とする2入力ANDゲートがあり，(l_1, l_f) と (l_2, l_f) が同じ遷移のb-fペアであるならば，この2つのb-fペアは，2入力ANDゲートの出力 l_o と l_f で構成されるb-fペアに置き換えることができる．一般にあるゲートのすべての入力が，同じ遷移の同じ信号線とb-fペアを構成するとき，それらのb-fペアは一つに統合できる．同様に，ある分岐のすべての分岐の枝が，同じ遷移で同じ信号線とb-fペアを構成するときにも，それらのb-fペアは一つに統合できる．

4．　パス数計算アルゴリズム

本手法のFUパスの判定能力を他の手法[3][4]と比べるには，識別できたFUパス（RDパス）の数を比較する必要がある．完全論理パス上にb-fペアを構成する信号線を含んでいれば，それはFUパスであることがわかるので，各々のb-fペアから求められるFUパスの数を数えることは容易である．しかしながら，一つのFUパスに複数のb-fペアが含まれる場合には同じFUパスが何度も数えられることになり，回路のFUパスの数を正確にかつ短時間で計算することは，容易ではない．

そこで，b-fペアを含む完全論理パスの数を数えるアルゴリズムを2通り開発した．1つは，b-fペアからFUパスの数を正確に数えることができるが，複雑な分岐再収れん構造を持つ回路には，大きな記憶空間を必要とするアルゴリズムで，他の1つは，FUパスの数の下界しか得られないが，計算時間と記憶空間のどちらも現実的なアルゴリズムである．

パスの数の計算は，外部入力から外部出力までの信号線を順次処理していく[7]の完全物理パス数計算手法に基づく．まず，各外部入力のラベルを1とする．ゲートの出力信号線のラベルは，そのゲートの入力信号線のラベルの総和とし，分岐の枝のラベ

ルは，分岐の幹のラベルと同じとする．最終的に外部出力のラベルの総和が，回路のパス数となる．

4．1　正確な計算アルゴリズム

正確な数え上げ手順は，各信号線 l_i について，外部入力から l_i までの論理パスを以下のグループに分けて数える．それらは，外部入力から l_i までに，

(1) b-fペアを構成する b-line と f-line の両方を含むパス，

(2) b-fペアを構成する b-line のみを含み，その f-line は l_i から外部入力までに現れるようなパス，

(3) f-line が l_i を通る完全パス上に現れるような b-fペアの b-line を外部入力から l_i までに持たない（b-fペアを構成する信号線を含まない）パス，

の3種類である．l_i が b-line のとき，(3) のパスは (2) のパスとなる．また，l_i が f-line のとき，対応する b-line を含む (2) のパスは，(1) のパスとなる．

詳細なアルゴリズムを図4に示す．なお，信号線 l_i は遷移の情報を含むものとする．$np_{nt}(l_i)$ は (1) のパスの数であり，それらはFUパスである．$np_t(l_i)$ は (3) のパス数で，それらパスはFUパスでない．変数 $s_j(l_i)$, $np_{sj}(l_i)$, $S(l_i)$ 等は，(2) のパスを記憶するために用いる．l_i までに現れる b-line の各集合に対して，$s_j(l_i)$ は，対応する f-line を表現する．$np_{sj}(l_i)$ は第 j 番目の f-line の集合に対する b-line を通過するパスの数である．l_i が $s_j(l_i)$ に含まれる f-line ならば，$np_{sj}(l_i)$ の値を $np_{nt}(l_i)$ に加算し，$s_j(l_i)$ を空にする．変数 $S(l_i)$ はすべて $s_j(l_i)$ を要素とする集合であり，要素数を $ns_s(l_i)$ で表す．$np_s(l_i)$ は外部入力から l_i までのパスのうち，$np_{nt}(l_i)$ と $np_t(l_i)$ のどちらにも数えられなかったパス数である．また，$FIN(l_i)$ は l_i のファンイン信号線の集合である．

プログラムの実装において，$s_j(l_i)$ と $S(l_i)$ にはリスト構造を用いる．また，$S(l_i)$ は l_i のファンイン信号線 l_j の $S(l_j)$ の和集合から得られ，l_j と l_i がそれぞれ分岐の枝と幹であるとき，$S(l_i)$ は $S(l_j)$ のコピーとなる．従って，分岐再収れんが複雑で多くの b-fペアが見つかった回路では，$S(l_i)$ が大きくなり，多大な記憶空間を必要とすることになる．

4．2　近似計算アルゴリズム

近似計算アルゴリズムの概要は，図5に示すが，これも外部入力から外部出力に向かって信号線を順次処理する．前節で述べたアルゴリズムとは，各 f-line を独立に扱う点が異なる．結果として，FUパスとして数えられるパスが実際より少なく数えられる場合が生じる．

次に図5で用いられた変数を説明する．$np(l_i)$ は外部入力から l_i までのパスのうち，l_i まではFUパスでないパスの数である．各 f-line l_i に対して，$np_{li}(l_j)$ は外部入力から l_j までのパスのうち l_i の b-line を含まないパスの数である．l_i と l_j で b-fペアを構成するとき，$np_{li}(l_j)$ の値は0となる．そうでなければ，$np_{li}(l_j)$ は l_j の入力 l_k における $np_{li}(l_k)$ の値の総和となる．$np_{li}(l_j)$ の計算において，l_j が l_i に達したら，$np_{li}(l_j)$ の値が $np(l_j)$ となる．回路全体の FUパスの数は外部出力 l_i における $np(l_i)$ の総和となる．

$np(l_i)$ の計算では，外部入力から l_i までのパスに b-line と f-line の両方が含まれるような b-fペアは考慮されない場合がある．そのため，$np(l_i)$ には，FUパスが含まれることがあり，得られる FUパス数が実際より少なくなる場合が生じる．

Procedure: Counting the exact number of FU paths
for every PI l_i,
  set $np_t(l_i)$=0, $np_{nt}(l_i)$=0, $ns_s(l_i)$=0.
  if l_i is a b-line of b-f pairs, then
    set f-lines of the b-f pairs to $s_1(l_i)$,
    set $np_{s1}(l_i)$=1 (i.e. $ns_s(l_i)$=1).
  else if partial path PP_x starting from l_i is FU, then set $np_{nt}(l_i)$=1.
  else set $np_t(l_i)$=1.
for every internal line or PO l_i,
  set $np_t(l_i) = \sum_{lj \in FIN(li)} np_t(lj)$, $np_{nt}(l_i) = \sum_{lj \in FIN(li)} np_{nt}(lj)$, $S(l_i) = \bigcup_{lj \in FIN(li)} S(lj)$.
  if partial path PP_x starting from l_i is FU, then
    for every f-line set $s_j(l_i)$ of l_i,
      set $np_{nt}(l_i)$=$np_{nt}(l_i)$+$np_{sj}(l_i)$,
      remove $s_j(l_i)$ from $S(l_i)$.
  else
    for every f-line set $s_j(l_i)$ of l_i,
      if l_i exists in $s_j(l_i)$of l_i, then
        set $np_{nt}(l_i)$=$np_{nt}(l_i)$+$np_{sj}(l_i)$,
        remove $s_j(l_i)$ from $S(l_i)$.
      else if any line in $s_j(l_i)$ never exists between l_i and POs, then
        set $np_{nt}(l_i)$=$np_{nt}(l_i)$+$np_{sj}(l_i)$,
        remove $s_j(l_i)$ from $S(l_i)$.
    if l_i is a b-line of b-f pairs, then
      for every f-line set $s_j(l_i)$ of l_i,
        add f-lines of the b-f pairs to $s_j(l_i)$.
      if $np_t(l_i)$≠0, then
        set $ns_s(l_i)$=$ns_s(l_i)$+1,
        set f-lines of the b-f pairs to $s_m(l_i)$ where m=$ns_s(l_i)$,
        set $np_{sm}(l_i)$=$np_t(l_i)$,
        set $np_t(l_i)$=0.
The number of FU paths is $\Sigma np_{nt}(l_i)$for every PO l_i.

Fig. 4: Procedure for exact count of FU Paths

Procedure: Counting the approximate number of FU paths
for every PI l_i,
  if partial path PPx starting from l_i is FU, then set $np(l_i)$=0.
  else set $np(l_i)$=1.
for every internal line or PO l_i,
  if l_i is an f-line, then
    for every PI l_j,
      if l_j and l_i are a b-f pair, then set $np_{li}(l_j)$=0.
      else set $np_{li}(l_j)$=$np(l_j)$.
    for every line l_j which exists in paths from PIs to l_i,
      if l_j and l_i are a b-f pair, then set $np_{li}(l_j)$=0.
      else if $np(l_j) > \sum_{lk \in FIN(lj)} np_{li}(lk)$, then set $np_{li}(l_j) = \sum_{lk \in FIN(lj)} np_{li}(lk)$.
      else set $np_{li}(l_j)$=$np(l_j)$.
  else *set* $np(l_i) = \sum_{lj \in FIN(li)} np(lj)$.
The number of FU paths = total paths - $\Sigma np(l_i)$ for every PO l_i.

Fig. 5: Procedure for approximate count of FU Paths

5．実験結果

提案手法を HP735 ワークステーション (256MB) 上にC言語でプログラム化し，ISCAS'85[12] およびフルスキャンを仮定したISCAS'89[13] のベンチマーク回路に適用した．プログラムのオプションとして，3章の拡張必要割当と4章のパス数計算の2つの手順に加え，各外部出力を単一出力回路とみなす回路分割を用意した．回路分割により，部分パスの長さが長くなるものがあり，結果としてより多くの FUパスを識別できる．実験では，これらのオプションを表1の MODE_1 から MODE_5 に示す組合せで適用した．MODE_5 は，記憶容量が不十分で正確な FUパス数計算アルゴリズムを適用できなかった *c6288* にだけ用いた．

FUパスに対する [4] の手法と本手法の実験結果を表2に示す．計算時間にはb-fペアを計算する時間だけでなく，b-fペアから FUパスを数える時間も含まれている．また，[4] の手法の計算時間は，SUN Sparc10 ワークステーションによるものである．約半数の回路で，[4] の手法は本手法よりも多くの FUパスを見つけている．しかし，必要割当の計算に間接含意を用いているため，*c1908* や *s38584* のように本手法の方が勝っている回路もある．また，計算時間については，[4] の手法はパス数の多い回路では多大な時間を要しているが，本手法はそのような傾向がなく高速に処理できていることがわかる．特に，[4] の手法では *c6288* を現実的に処理できないが，本手法は表4に示すように，わずか71秒で 10^{20} 以上あるパスの99% 以上がテスト不要であることを見つけている．

Table 1: Options investigated

	path count	nec. ass.	partition
MODE_1	approximate	simple	no
MODE_2	exact	simple	no
MODE_3	exact	extended	no
MODE_4	exact	extended	yes
MODE_5	approximate	extended	yes

FUパスの計算アルゴリズムの違いは，MODE_1 と MODE_2 の結果を比較することで，また，拡張必要割当の効果と回路分割の効果は，それぞれ MODE_2 と MODE_3 の結果，およびMODE_3 と MODE_4 の結果を比較することでわかるが，識別できる FUパス数が増加する事が確認できる．

本手法は，RD パスの識別にも適用できるが，実験結果を表3に示す．[3] の手法は [4] の手法に基づくため，実験結果の比較では，識別できたパス数，計算時間とも FUパスの場合と同じような傾向が見られる．

6．まとめ

本論文では，パス遅延故障の効率化のための組合せ回路における FU/RD パスの識別法を提案した．提案手法は，部分パスの必要割当に基づく回路の局所的な解析により FU/RD パスが含む信号線のペアを見つけるため，回路のパス数に依存しない計算時間で処理可能となった．一般に識別できる FU/RD パスの数は従来手法と比べて少なくなるが，その差は小さい．その一方で処理時間は大幅に短縮し，*c6288* のようにパス数が多く従来は扱うことができなかった回路であっても，短時間で処理可能になった．本手法は，FU/RD パスのようなテスト不要パスを識別するだけであるため，残りのパスについてテストパターンを生成することが今後の課題である．

References

[1] G. L. Smith, "Model for Delay Faults Based upon Paths," International Test Conf., pp. 342-349, 1985.

[2] Z. Barzilai and B. K. Rosen, "Comparison of AC Self-testing Procedures," International Test Conf., pp. 89-91, 1983.

[3] U. Sparmann, D. Luxenburger, K. -T. Cheng, and S. M. Reddy, "Fast Identification of Robust Dependent Path Delay Faults," Design Automation Conf., pp. 119-125, 1995.

[4] K. -T. Cheng and H. -C. Chen, "Delay Testing for Non-robust

Table 2: Number of FU paths of benchmark circuits

	number of FU paths identified						run time (sec)				
circuit	total paths	[4]	MODE_1	MODE_2	MODE_3	MODE_4	[4]	M_1	M_2	M_3	M_4
c880	17,284	163	163	163	163	163	9*	1	1	1	2
c1355	8,346,432	6,776,160	6,753,120	6,776,160	6,776,160	6,776,160	968*	3	4	6	67
c1908	1,458,114	478,107	496,167	501,955	502,013	505,834	1152*	3	3	5	42
c2670	1,359,920	1,050,737	1,192,015	1,192,940	1,194,077	1,194,077	119*	3	3	5	19
c3540	57,353,342	41,386,885	37,869,721	38,197,445	39,452,156	39,538,787	10115*	9	131	201	233
c5315	2,682,610	2,093,814	1,373,805	1,825,325	1,858,165	1,858,584	1067*	7	7	12	76
c7552	1,452,988	999,357	673,289	877,937	885,324	886,350	400*	21	13	32	220
s5378	27,084	2,955	2,712	2,712	2,769	2,955	22*	5	5	7	30
s9234	489,708	285,921	283,594	283,745	283,940	284,566	649*	16	15	23	121
s13207	2,690,738	1,946,331	1,879,095	1,889,165	1,930,925	1,933,419	1855*	41	44	110	544
s15850	329,476,092	277,285,787	275,470,498	277,177,686	277,178,126	277,199,468	117931*	60	135	184	2408
s35932	394,282	276,137	270,057	276,137	276,137	276,137	4497*	493	399	546	1487
s38417	2,783,158	944,360	861,860	890,000	890,000	905,204	4215*	224	179	248	2560
s38584	2,161,446	1,461,347	1,370,244	1,457,527	1,462,263	1,462,263	8630*	450	432	701	2746

(*) Time is measured on SUN Sparc 10

Table 3: Number of RD paths of benchmark circuits

	number of RD paths identified			run time (sec)		
circuit	[3]	MODE_3	MODE_4	[3]	M_3	M_4
c880	553	309	309	21	1	3
c1355	7,236,128	6,776,160	6,776,160	1199	6	120
c1908	1,094,921	1,092,858	1,097,696	813	7	73
c2670	1,121,767	1,219,306	1,219,306	799	6	20
c3540	54,480,998	53,057,006	53,744,298	36749	259	360
c5315	2,248,151	2,091,832	2,094,419	988	17	141
c7552	1,114,613	1,017,421	1,020,752	1136	36	283
s5378	4,632	3,810	4,073	50	7	35
s9234	419,122	406,051	407,124	222	25	131
s13207	2,162,610	2,080,600	2,082,766	5818	116	503
s15850	315,716,886	310,277,470	311,443,789	182671	191	3085
s35932	335,625	335,625	335,625	629	1399	2566
s38417	1,518,659	1,364,469	1,376,879	3937	266	2841
s38584	1,713,928	1,712,630	1,732,277	2912	1312	3564

Table 4: Number of FU/RD paths of c6288

	number of paths	time
total paths	197886883476589871104	-
MODE_1(FU)	197774771745979470000	71
MODE_5(FU)	197775408336551180000	1342
MODE_1(RD)	197778466998649090000	71
MODE_5(RD)	197779083275152000000	1390

Untestable Circuits," International Test Conf., pp. 954-961, 1993.

[5] W. K. Lam, A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Delay Fault Coverage and Performance Tradeoffs," Design Automation Conf., pp. 446-451, 1993.

[6] K. Fuchs, F. Fink, and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation for Path Delay Faults," IEEE Trans. on CAD, vol. CAD-10, pp. 1323-1335, Oct. 1991.

[7] I. Pomeranz and S. M. Reddy, "An Efficient Non-Enumerative Method to Estimate Path Delay Fault Coverage," International Conf. on Computer-Aided Design, pp. 560-567, Nov. 1992.

[8] P. Goel, "Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Comp., vol. C-30, pp. 215-222, Mar. 1981.

[9] H. Fujiwara, and T. Shimono, "On the Acceleration on Test Generation Algorithms," IEEE Trans. on Comp., vol C-32, pp. 1137-1144, Dec. 1983.

[10] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Trans. on CAD., pp. 126-137, Jan. 1988.

[11] J. Rajski, and H. Cox, "A Method to Calculate Necessary Assignments in Algorithmic Test Pattern Generation," 1990 ITC, pp. 25-34, Sept. 1990.

[12] F. Brglez, and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Design and a Special Translator in Fortran," ISCAS'85, June 1985.

[13] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," ISCAS'89, May 1989.