設計自動化 44－7
（1988. 10. 28）

ＳＰによるテスト系列自動生成システム

高山浩一郎　　広瀬文保　　川戸信明

㈱富士通研究所

被検査回路のテスト系列を自動生成する「テスト生成回路」をホスト上で自動合成し，これを論理シミュレーション専用マシンＳＰで高速にシミュレーションすることによりテスト系列を自動生成するシステムを研究開発中である．テストパターンの生成手法に関しては，これまでの縦型探索アルゴリズムに基づく手法に加え，ランダムパターンを用いたテスト生成手法を実現した．

本文では，複数のプロセッサを用いて，ランダムパターンによるテスト生成手法をＩＳＣＡＳのベンチマーク回路に適用して性能評価を行ったので報告する．また，複数のプロセッサを用いた並列処理による高速化と，局所的に有効な故障のみの挿入による効果について解析を加えた．さらに，ランダムパターンによるテスト生成手法から，アルゴリズミックな手法に変更することによる高検出率の達成の可能性を確認した．

An automatic test pattern generation system using SP

Koichiro TAKAYAMA　Fumiyasu HIROSE　Nobuaki KAWATO

FUJITSU LABORATORIES LTD.
1015, Kamikodanaka, Nakahara-ku, KAWASAKI 211, JAPAN

We are developing an ATPG system using an ultrahigh-speed logic simulator "SP". We synthesize a 'test-generation circuit' of a circuit under test, which automatically generates the tests of the original circuit. The tests are generated by simulating the test-generation circuit using the SP. We implemented the random pattern test mode into our test-generation circuit, where one of the depth-first-type algorithmic mode is already working.

In this paper, we report experimental results of the random test pattern using the ISCAS benchmark circuits. We also analyzed the performance of parallel processing using multiple processors of the SP and the effect of the locally effective fault injection. And we confirmed the ability of high fault coverage by changing the generation mode from the random pattern test to the algorithmic approach.

1.はじめに

ＶＬＳＩなどの製品の品質を保持する手段としてテストパターンを用いた検査が行われている．ところがＶＬＳＩの高集積化に伴い，高品質なテストパターンを高速に求めることが困難になっている．

順序回路のテストパターン生成は，テスト容易化設計の採用により組合せ回路の問題に帰着して問題の簡単化を行っている[1]．そして，組合せ回路のテストパターン生成に対しては，これまでに，いくつかの手法が提案されている[2~6]が，我々は，さらに高速化が期待できるアプローチとして，シミュレーション専用マシンを応用したテスト生成システムの開発に着手した[8,9]．図１に本システムの概要を示す．

まず，被検査回路（組合せ回路）と仮定故障集合からテスト生成回路をホスト計算機上で自動的に合成する．そして，高速な論理シミュレータを用いてテスト生成回路の動作を模擬することにより被検査回路に対するテスト系列を自動生成する．論理シミュレータとしてはＳＰ[7]を使用する．

被検査回路の初期的な入力としてランダムパターンを用いると，短時間に高い検出率を達成することができる．本文では，あらかじめ用意したランダムパターンを用いたテスト生成手法を実現し，ベンチマーク回路に適用して性能を評価した．また，複数のプロセッサを用いて並列処理による効果を測定した．並列処理方式としては，仮定故障の集合を分割して各プロセッサに与え，並列にテスト生成を行う方式を採用した．

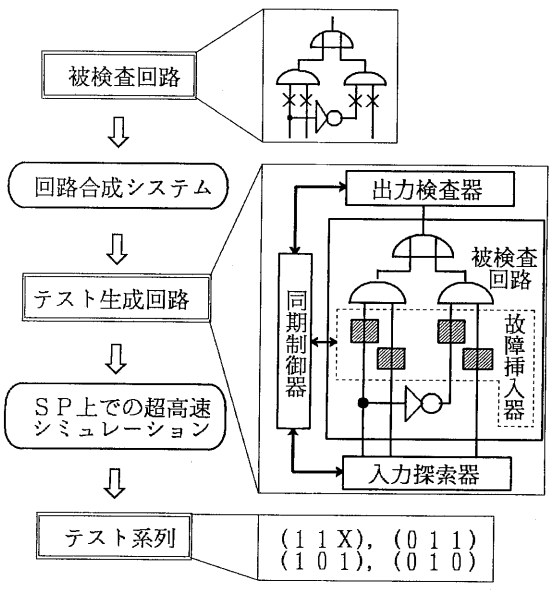

図１．テスト系列自動生成システムの概要

実験の結果，回路によって処理時間が台数以上の加速を示したものがあった．これは，テスト生成回路のインプリメント法によるものである．

ランダムパターンによるテスト生成を行った後，まだ検出率が不十分である場合には，ランダムパターンで検出されなかった故障に対してアルゴリズミックな探索によるテスト生成手法を適用することによりさらに高い検出率を達成する．実験では，ランダムパターンで検出されなかった故障に対して入力先行型[8]を適用することにより，いくつかの回路で良好な結果を得ることができた．

2.テスト生成回路

2.1.概要

現在実現しているテスト生成手法は３つある．

①ランダムパターン型（ＲＰ型）：被検査回路に対してランダムパターンを入力し，検出された故障を検査対象から取り除く．

②入力先行型（ＩＦ型）：被検査回路に対して，未探索空間から取り出した入力パターンの集合Ｉを設定する．Ｉが検出する可能性のある故障が存在する限りＩの範囲を狭めていき，検出された故障を検査対象から取り除く．Ｉのもとで検出される故障が無くなった時点で未探索空間からＩを取り除き，新たなＩを設定する．

③故障先行型（ＦＦ型）：被検査回路中に１つの未検出故障ｆを仮定し，ｆを検出する入力パターンｉを求める．次にｉのもとでの等価故障を検出し検査対象から取り除く．

テスト生成回路は被検査回路に次の４つのモジュールを付加することにより合成される．

①同期制御器：上記したテスト生成手法に従って，各モジュールにクロック（Pclk, Rclk, Fclk）を発行することにより同期制御器を行う．

②入力探索器：被検査回路に対して与える入力パターンを発生する．ＩＦ型またはＦＦ型によりテスト生成を行う場合，Pclkに同期して入力の縦型探索を行う．ＲＰ型によりテスト生成を行う場合，Rclkに同期してランダムパターンを取り替える．

③故障挿入器：被検査回路中の信号線に仮定故障を挿入する．Fclkに同期して故障を取り替える．

④出力検査器：被検査回路の出力を検査して故障が検出されたかどうかを判定する．

2.2.動的な故障挿入

テスト生成回路のシミュレーションを効率よく行うために，故障の動的な挿入を実現している．

本システムで用いている故障挿入器は単一縮退故障を対象としている．仮定故障のうち同一信号線上の0縮退故障と1縮退故障をまとめて一要素として，故障の仮定される素子の入出力線の各々に1つの故障挿入レジスタ（ＶＣ:Value Converter）が挿入される．ＶＣは，故障挿入器を構成する基本レジスタである．各ＶＣは内部に検出フラグを持ち，故障が検出されたときにフラグを立てて検査対象から外れる．また，自分が故障を挿入する信号線の論理値を知ることができるので，自分の持つ検出フラグと比較して，信号線に故障を挿入して影響が現れるときにのみ故障を挿入する

故障シミュレーションはパラレル法を基本としているので，同時に複数個の故障を挿入することができるその数をmとすると，仮定故障の集合をm分割し各部分集合に属するＶＣを，故障挿入権を伝搬するキャリーで接続して計m本のチェイン（ＶＣチェイン）を構成する．ＶＣチェインはFclkに同期して故障を取り替えるが，キャリーにより局所的に有効（未検出で挿入して影響が現れる）な故障のみが選ばれて次々挿入される．このように，故障の動的な挿入を行うことにより故障シミュレーションの回数を削減している．

2.3.テスト生成手法の変更

被検査回路の初期的な入力としてランダムパターンを用いると，短時間に高い検出率を達成することができる．しかし，未検出故障が少なくなってくると，1個の故障を検出するために投入するパターン数が多くなり経過時間に比べて検出率の伸びが小さくなる．そこで，一般に，ランダムパターンを用いてある程度の検出率を達成した後，アルゴリズミックにテストパターンを生成して，さらに検出率を高めるという手法が取られている．我々のシステムでも，まず，被検査回路に対してＲＰ型を適用して高速に検出率を高めた後検出されなかった故障に対してＩＦ型，ＦＦ型を適用して，より高い検出率を持つテストパターンを高速に生成する．本システムでは，同期制御器にテスト生成手法を次々と変更する機構を付加することで，効率良くテスト生成を遂行する方式を可能にしている．テスト生成手法の変更の時期は，同期制御器内部にあらかじめ設定することができる．

3.ランダムパターン型によるテスト生成

ＲＰ型によるテスト生成手法を10個のベンチマーク回路[10]に適用して実験を行った．実験で使用したランダムパターンは，あらかじめホスト上のシステム関数を用いて必要数生成したものをＳＰ内部のメモリにロードしておく．入力探索器はRclkによりカウンタをインクリメントしてメモリからランダムパターンを読み出し，被検査回路の入力とする．もちろん，線形帰還シフトレジスタのような疑似乱数発生器をRclkにより動作させて入力パターンを得るような機構にすることも可能である．適用したパターン数は4096個とした．また，性能を評価するためにパターンを32個ずつ区切ってシミュレーションを行った．

3.1.単プロセッサによるシミュレーション

まず，1台のプロセッサを用いて実験を行った．故障（ＶＣ）は回路記述ファイルに出現する順に2つの集合に振り分け，順番にキャリーで結合することにより2本のＶＣチェインを構成した．各パターンでの検出故障数および故障検査数，32パターンごとのシミュレーションに要した時間および動作クロック数を測定した．故障検査数とは1個のパターンに対して故障挿入器が挿入した故障の数である．所要時間はＳＰがシミュレーションを実行した経過時間で，動作クロック数とは同期制御器に対して与えられる基本クロック（Bclk）数である．表1に実験結果を示す．c2670，c7552を除いて検出率が95%を越えている．また，c432，c499，c1355，c5315，c6288においては，文献(4)に示される検出可能な故障はすべて検出されていることからも，初期パターンとしてランダムパターンを用いることが有効であることがわかる．

図2に適用したパターン数に対する未検出故障数と故障検査数の変化を示す．故障検査数はRclk～Rclk間に各ＶＣチェインに対して発行されたFclk数を合計して求めた．図中，点線は（未検出故障数／2）の値を示す．すべての回路で故障検査数は常に未検出故障数の約半分の値である．この結果から，動的な故障挿入の実現によりテスト生成回路の動作量の削減が効率良く行われていることがわかる．

図3に各測定区間に対して要した時間と動作クロック数の変化を示す．いずれの回路もシミュレーションが進むにつれて1パターン当たりの故障シミュレーションに要する時間および動作クロック数が減少しているが，所要時間が動作クロック数に比べて減少率の小さい回路がある．

テスト生成回路はBclkにより動作しており，

Bclk数＝Fclk数＋Rclk数　　…（式1）

である．Fclkによるテスト生成回路の動作（シミュレーション時のイベント）は，故障の挿入状況が変化した信号線から，その信号線が支配する出力へのパス上の素子に発生する．一方，Rclkでは被検査回路の入力がランダムに変化するので，Rclkによるテスト生成回路の動作量はFclkによる動作量に比べて大きいと予測できる．各区間のシミュレーションにおいて，Fclk数は図2に示すように減少していくが，Rclk数は一定（

表1．単プロセッサによる実験結果（4096パターン）:preliminary

回路名	ｹﾞｰﾄ数	ﾃｽﾄ生成回路規模	全故障数(VC数)	未検出故障数(VC数)	検出率(%)	ﾃｽﾄ数	所要時間(秒)	動作ｸﾛｯｸ数
c432	160	2110	524 (392)	4 (4)	99.24	88	5.896	16368
c499	202	2540	758 (459)	8 (8)	98.94	64	9.464	18556
c880	383	4155	942 (791)	7 (7)	99.26	106	12.14	20344
c1355	546	6180	1574(1283)	8 (8)	99.49	102	25.25	30613
c1908	880	7219	1879(1469)	16 (16)	99.15	178	64.49	77069
c2670	1193	12071	2747(2153)	430 (387)	84.35	111	380.4	418556
c3540	1669	13554	3428(2827)	145 (145)	95.77	244	215.6	166765
c5315	2303	22152	5350(4421)	59 (59)	98.90	229	263.2	118763
c6288	2406	27861	7744(6256)	34 (34)	99.56	55	225.1	23980
c7552	3512	30158	7550(6142)	438 (421)	94.20	266	1523	596461

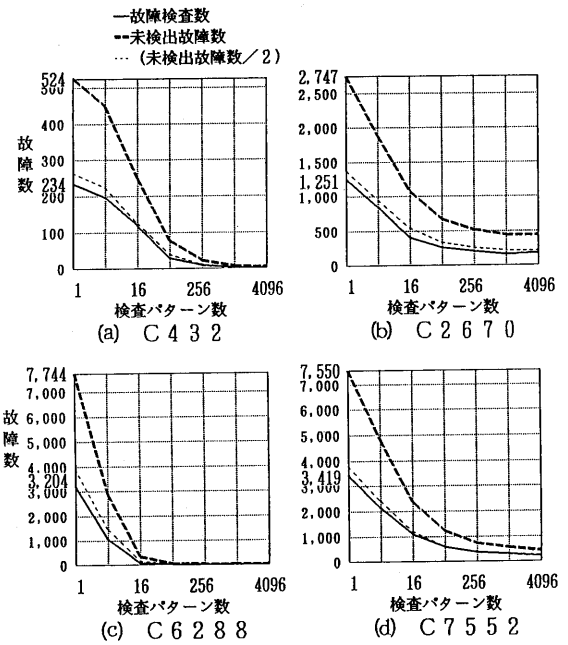

図2．検査パターン数に対する
未検出故障数と故障検査数の変化

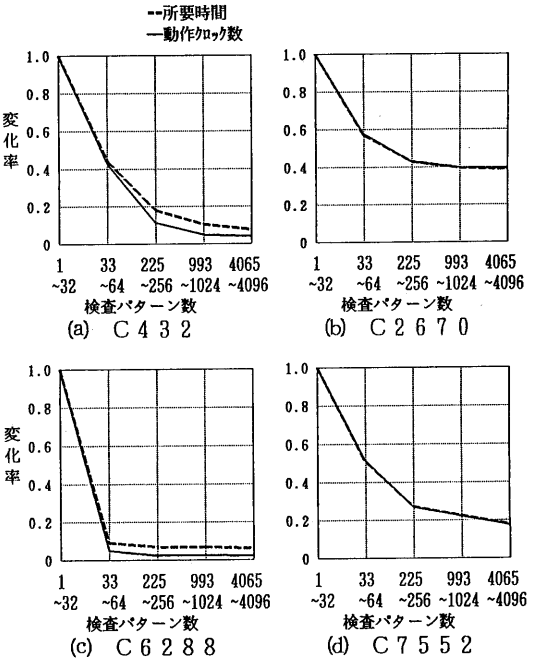

図3．検査パターン数に対する各区間の
所要時間と動作クロック数の変化

32個）である．従って，未検出故障数が少なくなるほどRc1kによる動作量が支配的となり，動作クロック数に比べて所要時間の減少率が小さくなる．c2670，c7552などは最後まで多くの故障が検出されずに残っており，Fc1kによる動作量が依然支配的であるため，所要時間と動作クロック数の減少率はほぼ同じになる．

3.2.複数プロセッサによるシミュレーション

4台までのプロセッサを用いて並列処理による台数効果を測定した.

並列処理を行う場合，問題をどのように分割するか分割した問題を資源にどのように割り当てて処理を行うかを決定することが重要である.

問題を分割するアプローチとして,

①仮定故障の集合を分割し，各部分集合に対してテスト生成を行う．ただし，RP型の場合，各故障集合には同じランダムパターンを適用する．IP型，FP型の場合，各故障集合に対する入力の探索は全空間を対象とする.

②テストパターンの探索空間を分割する．すなわちＲＰ型の場合，適用するランダムパターンの集合を分割する．ＩＦ型，ＦＦ型の場合，探索を行う入力空間を分割する．ただし，分割された入力はそれぞれ全仮定故障を対象とする.

③仮定故障の集合を分割し，さらに，各部分集合に属する故障に対するテストパターンの探索を分割して行う．すなわち，①と②のアプローチを組み合わせてテスト生成を行う.

の３通りがある．①は仮定故障の部分集合に対する探索は独立に行うことができる．②においてフォールトドロップを行うためには，ある入力空間内で故障が検出されたときに，他の空間を探索しているプロセスに対して，検出された故障の情報を伝達する手段が必要となる．③は①または②のアプローチによる並列処理の性能の向上が限界に達した場合に有効な手法である例えば，仮定故障の分割による並列処理の性能の向上が限界に達した場合，それ以上故障の分割は行わず，各部分集合に対する入力の探索を分割することによって，さらに性能を向上させることが可能になる.

本システムにおいては，分割された小問題ごとにテスト生成回路を合成し，ＳＰを用いて各テスト生成回路をシミュレーションする．テスト生成回路をシミュレーションするときのアプローチとしては,

④１台のプロセッサに１個のテスト生成回路をロードしてシミュレーションを行う.

⑤１台のプロセッサに複数個のテスト生成回路をロードしてシミュレーションを行う.

⑥１個のテスト生成回路を複数台のプロセッサに分割してロードしてシミュレーションを行う.

の３通りがある．④は，並列処理を行う手法として一般的なものである．⑤は，合成されたテスト生成回路の規模がプロセッサの容量（64ｋ素子／ＧＰ）に比べ十分小さく，シミュレーションのパイプライン効果が現れない場合に有効である．⑤は，一般に，合成されたテスト生成回路がプロセッサの容量を越えた場合に適用される．また，容量の範囲内であっても，分割してシミュレーションしたほうが効率が良い場合がある．このアプローチでは，負荷分散を考慮してテスト生成回路をどのように分割するかが問題となる.

今回の実験では①と④を組み合わせた．その理由は，テスト生成回路の合成が容易で，しかも，プロセッサ（ＧＰ）間の通信がまったく起きないので，性能評価が簡単になるからである.

各テスト生成回路が受け持つＶＣチェイン数はすべて２本とした．従って，ｎ台のプロセッサを用いる場合には，仮定故障の集合を3.1.節と同様に被検査回路の記述ファイルに出現する順に２ｎ個の部分集合に分割する．それをもとに，２個の故障集合をペアにしてｎ個のテスト生成回路を合成し，ｎ台のプロセッサに１回路ずつロードしてシミュレーションを行った．その結果を表２に示す．また，１台のプロセッサを用いてシミュレーションを行った場合に要した時間および動作クロック数を１としたときの台数効果（加速比）を図４に示す．実線が所要時間，破線が動作クロック数を示す．なお，一点鎖線（---）は，後述するように動作クロック数の台数効果の理論値を示す.

ｐ番目のパターンに対する故障検査数を$F(p, F_0)$とする．F_0は全故障数である．また，全パターンに対するシミュレーション中に検査される故障の合計は$\Sigma_p F(p, F_0)$である．ｎ台のプロセッサを用いる場合，各テスト生成回路に割り当てられる仮定故障数はF_0/nとなる．ｉ番目のテスト生成回路でｐ番目のパターンに対する故障検査数を$F_i(p, F_0/n)$とする．ところで，ｐ番目のパターンに対して検査される故障の総数は分割に関係なく一定であるので,

$$F(p, F_0) = \Sigma_i F_i(p, F_0/n)$$

であり，全パターンに対する故障検査数の合計は,

$$\Sigma_p F(p, F_0) = \Sigma_p \Sigma_i F_i(p, F_0/n)$$

となる．ここで，各プロセッサにロードしているテスト生成回路間でFclk数のばらつきが無いと仮定すると,

$$\Sigma_p F(p, F_0) = n \Sigma_p F_i(p, F_0/n)$$

$$\therefore \Sigma_p F_i(p, F_0/n) = \Sigma_p F(p, F_0)/n \quad \cdots (式2)$$

となる．故障検査数は各ＶＣチェインに対して発行されるFclk数の和に等しい．全Rclk数をRclkすると，故障をｎ分割した場合の動作クロック数Bclk(n) は式１より,

$$Bclk(n) = \Sigma_p F_i(p, F_0/n) + Rclk$$

（分割しない場合はｎ＝１）

となる．従って，分割によるBclk数の比R_bは,

$$R_b = \frac{Bclk(n)}{Bclk(1)} = \frac{1}{n} + \frac{(n-1)\,Rclk}{n \cdot Bclk(1)} \quad \cdots (式3)$$

となる．式３より加速比$1/R_b$を求めたものを図４中

表２．複数プロセッサによる実験結果（4096パターン）:preliminary

プロセッサ数 / 回路名	所要時間（秒）				動作クロック数			
	1	2	3	4	1	2	3	4
c432	5.896	3.357	2.346	1.662	16368	13056	11490	10562
c499	9.464	5.449	3.826	1.675	18556	12808	11322	9189
c880	12.14	6.123	5.313	3.065	20344	14035	16960	11020
c1355	25.25	14.71	11.56	8.364	30613	26625	16417	24363
c1908	64.49	30.10	21.15	17.15	77069	46207	34362	31461
c2670	380.4	127.3	70.20	49.49	418556	223151	158032	117424
c3540	215.6	86.79	58.57	44.52	166765	92223	72502	58014
c5315	263.2	113.7	77.43	62.27	118763	65139	49518	42895
c6288	225.1	169.4	147.3	128.3	23980	16528	14306	13308
c7552	1523	471.2	249.9	168.1	596461	318397	218666	167567

に一点鎖線（-·-）で示す．実際の動作クロック数の加速比は，式３の予測に非常によく一致している．若干の差が生じている理由は，各パターンに対するテスト生成回路間でのFclk数のばらつきや，32パターンずつ区切ってシミュレーションしたことによる各区間での各テスト生成回路の動作量のばらつきのために，性能が落ちるからである．

テスト生成回路中のＶＣ数を♯VC とし，被検査回路中のゲート数をＧとする．１発のRclkに対する，故障をｎ分割したテスト生成回路の動作量（イベント数）$M_R(n)$は，

$$M_R(n) = \alpha(G + \#VC/n) + \beta$$

となる．ここで，αは（被検査回路＋ＶＣ）のイベント率，βはその他の付加回路の動作量である．従ってｎ分割することによるRclkでのテスト生成回路の動作量の比R_Rは，

$$R_R = \frac{M_R(n)}{M_R(1)} = \frac{\alpha(G + \#VC/n) + \beta}{\alpha(G + \#VC) + \beta}$$

となり，Rclkでのテスト生成回路の動作量は分割による台数効果はあまり期待できない．

現在のテスト生成回路のインプリメント法では，FclkがあるＶＣチェインに発行されると，そのチェインに属する全ＶＣが評価される．ゆえに，全Fclkに対する，故障をｎ分割したテスト生成回路の動作量$M_F(n)$は，

$$M_F(n) = \Sigma F_i(p, F_0/n) \cdot (\#FIR/n + \gamma)$$

となる．γは故障の挿入状況の変化にともなう回路の動作量である．従って，ｎ分割することによるFclkでのテスト生成回路の動作量の比R_Fは，式２より，

$$R_F = \frac{M_F(n)}{M_F(1)} = \frac{\#VC/n + \gamma}{n(\#VC + \gamma)}$$

となり，Fclkでのテスト生成回路の動作量は分割による台数効果のオーダは$O(1/n^2)$と台数以上の加速を期待できる．

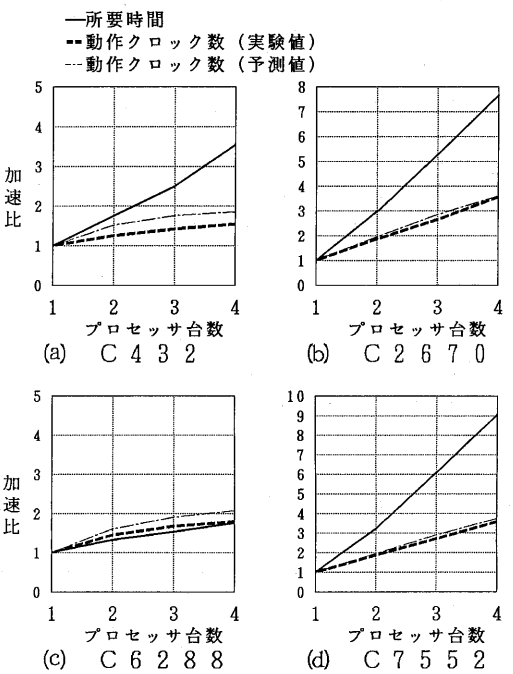

図４．並列処理による加速

表3．テスト生成手法の変更による実験結果・その1:preliminary
（4台のプロセッサを用いて，ＲＰ型により256ﾊﾟﾀｰﾝ 適用した後，ＩＦ型に変更）

回路名	ＲＰ型（256ﾊﾟﾀｰﾝ）			ＩＦ型				全体		
	所要時間（秒）	未検出故障数	ﾃｽﾄ数	所要時間（秒）	ｱﾎﾞｰﾄ故障数	冗長故障数	ﾃｽﾄ数	所要時間（秒）	検出率（%）	ﾃｽﾄ数
c432	0.317	22	76	0.054	4	0	6	0.371	99.24	82
c499	0.333	19	54	0.113	9	0	9	0.446	98.81	63
c880	0.572	47	81	0.145	0	0	26	0.717	100	107
c1355	1.13	102	68	0.367	9	0	34	1.50	99.43	102
c1908	2.29	253	83	1.70	55	0	89	3.99	97.07	172
c2670	3.70	512	86	22.4	299	0	102	26.1	89.12	188
c3540	4.94	306	146	5.51	160	39	77	10.5	94.19	223
c5315	8.09	177	154	1.44	113	0	45	9.53	97.89	199
c6288	11.8	34	55	0.72	34	0	0	12.5	99.56	55
c7552	16.9	720	165	49.0	460	2	102	65.9	93.88	167

並列処理による加速は，R_R とR_F のトレードオフにより決定される．c2670，c7552においては，最後まで多くの故障が検出されずに残っている．すなわち，テスト生成回路の動作量はFclkによる動作量が最後まで支配的となっているので，故障を分割すると，R_F が性能を支配して台数以上の加速を示した．一方，c6288 では，少ないパターン数でほとんどの故障が検出されてしまったため，R_R が性能を支配して故障を分割しても加速しなかった．c432では，ちょうどその中間であり，ほぼ台数通りの加速を示した．以上の考察より，故障の分割数を増やしていくにつれて，R_R の支配が強くなり加速が鈍る地点があることが指摘できる．そのときに，並列度に余裕があれば，③のアプローチに変更することで，さらに処理効率を上げることが可能になるであろう．また，ある程度の検出率が得られた時点でテスト生成手法を変更した方が効率が良いと思われる．

4.テスト生成手法の変更による高速化

全仮定故障に対してＲＰ型を適用した後，検出されなかった故障に対してＩＦ型を適用した．

ＲＰ型からＩＦ型へ切り換える時期の考察は今後の課題として，実験では，簡単のためすべてのベンチマーク回路に対して 256個のランダムパターンを適用した後，テスト生成手法の切り換えを行った．ＩＦ型では，入力の探索は縦型探索アルゴリズムにもとづいて行う．探索において，入力の値を決定する順番は入力が回路の記述ファイルに現れる順番とした．各入力の信号値はX⇨0⇨1⇨Xと変化させる．また，出力検査器の内部に，検出が困難である故障を検査対象から一時的に削除するアボート回路を付加し，探索時間が不必要に長くなるのを抑えた．実験では，故障が検出されることなく64回のバックトラックが行われたとき挿入していた故障をアボートするようにした．

ＩＦ型における縦型探索は，入力信号値をすべてXにリセットして開始される．探索中に再び信号値がすべてXにもどったとき，入力の全空間の探索を終了したことになる．このとき，アボートされずに残っていた未検出故障は冗長故障であることが証明される．

実験では，3.2.節の方法で故障を分割し，4台のプロセッサを用いて並列処理を行った．ＲＰ型の適用後，未検出故障に対してテスト生成回路を再合成してＩＦ型によるテスト生成を行った結果を表3に示す．

以下では，3.2.節において，4台のプロセッサを用いて4096個のランダムパターンによるＲＰ型を適用した実験結果との比較を行う．

c432，c880，c6288 は最高検出率（文献(4)）を，3.2.節の実験に比べて4～10倍高速に達成している．c2670 は約半分の所要時間で検出率が向上しているが，依然90%には達していない．その他の6個の回路では検出率が落ちており，これはＲＰ型からＩＦ型へテスト生成手法を変更する時期が悪かったためと思われる．そこで，その中で，3.2.節の実験で最高検出率を達成していない回路（c1908，c3540，c7552 ）とc2670 に

表4．テスト生成手法の変更による実験結果・その2:preliminary
（4台のプロセッサを用いて，ＲＰ型により4096ﾊﾟﾀｰﾝ適用した後，ＩＦ型に変更）

回路名	ＲＰ型 (4096ﾊﾟﾀｰﾝ)	ＩＦ型				全体
	未検出故障数	所要時間 (秒)	ｱﾎﾞｰﾄ 故障数	冗長 故障数	ﾃｽﾄ 数	検出率 (%)
c1908	16	0.46	13	0	3	99.31
c2670	430	17.0	255	0	80	90.72
c3540	145	2.23	103	38	4	95.89
c7552	438	22.7	260	9	75	96.44

対して，ＲＰ型により4096個のランダムパターンを適用した後テスト生成手法をＩＦ型に変更して実験を行った．結果を表4に示す．その結果，4回路とも数～数十秒のシミュレーションでさらに検出率を向上していることがわかる．

以上の考察から，ＲＰ型からＩＦ型へのテスト生成手法の最適な変更時期を設定することで，より高い検出率を効率良く達成できることがわかる．また，テスト生成手法をＦＦ型に変更することで，さらに高い検出率を達成できる．

4.おわりに

被検査回路の入力としてランダムパターンを適用するテスト生成手法をインプリメントして性能を評価した．また，複数のプロセッサに仮定故障を分割して与え，並列処理による高速化を図った．また，その性能を解析的に予測することを試み，実験値と良く一致した．その結果，仮定故障数が多く，かつ検出率の低い回路ほど並列処理により台数以上の加速を示すことがわかった．これは，テスト生成回路のインプリメント法によるもので，今後，Fclkの発行方式を最適化してFclkによるテスト生成回路の動作量を低減することによりシミュレーションの高速化を図る予定である．また，故障の分割数をさらに増やしたときの性能を測定する予定である．

ＲＰ型からＩＦ型へテスト生成手法を変更した結果手法を変更する時期を適切に設定することにより，より高い検出率をより高速に達成できることがわかった今後，テスト生成手法の変更を行う最適な時期の解析を行う予定である．また，テスト生成手法をＦＦ型に変更することで，さらに高い検出率を達成する．

参考文献

(1)樹下，藤原：ディジタル回路の故障診断（上），工学図書，1983年.

(2) P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trns. on Computers, Vol. C-30, No. 3, pp. 215-222, March 1981.

(3) H. Fujiwara, "FAN : A Fanout-oriented Test Pattern Generation Algorithm," Proc. ISCAS'85, pp. 671-674, June 1985.

(4) M. H. Shulz, A. Elisabeth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," FTCS-18, pp. 30-35, June 1988.

(5)伊藤，石浦，矢島：高速故障シミュレータを用いたテスト生成，情報処理学会第37回全国大会，pp. 1830-1831，1988年9月.

(6) A. Motohara, K. Nishimura, H. Fujiwara and I. Shirakawa, "A Parallel Scheme for Test-pattern Generation," Proc. ICCAD, pp. 156-159, November 1986.

(7) F. Hirose, et al., "Simulation Processor "SP"," Proc. ICCAD, pp. 484-487, November 1987.

(8) F. Hirose, K. Takayama, and N. Kawato, "A Method to Generate Tests for Combinational Logic Circuits using an Ultrahigh-speed Logic Simulator," Proc. ITC, pp. 102-107, September 1988.

(9)高山，広瀬，川戸：シミュレーション専用マシンＳＰによる自動テスト生成，情報処理学会第36回全国大会，pp. 2055-2056，1988年3月.

(10) F. Brglez, H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in FORTRAN," Special Session on ATPG and Fault Simulation, Proc. ISCAS, June 1985.