設計自動化　５０－１５
（1989.12.15）

ＳＰ応用テスト生成システムにおける高検出率化手法の評価

高山　浩一郎　　広瀬　文保
(株)富士通研究所

あらまし　我々は、これまでに論理シミュレーション専用マシンＳＰを応用した自動テスト生成システムを提案し、そのプロトタイプを作成した。本文では、本システムをベンチマーク回路に適用して性能を評価したので報告する。今回、入力の探索を３つの階層に分割する機構と、テスト生成時と故障シミュレーション時との故障の挿入方法を変更する機構を実現して検出率を向上した。その結果、ＳＰの１台のプロセッサを使用した場合でも従来の大形計算機上のソフトウエアによるテスト生成システムに比べ平均２倍の高速化と同等の検出率を達成できた。また、新しく、複数プロセッサを用いた性能向上の研究にも着手した。単純に故障を分割することによる並列処理を行った場合でも、８台のプロセッサを使用してさらに約４倍高速化できることがわかった。

The evaluation of the method of coverage improvement for the ATPG system using the SP

Koichiro TAKAYAMA and Fumiyasu HIROSE
FUJITSU LABORATORIES LTD.
1015, Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

Abstract　We developed an automatic test generation system using an ultrahigh-speed logic simulator SP. This paper describes the performance of the system by applying it to the ISCAS benchmark circuits. We improved the coverage through the three-layered depth first search of input space and the single fault injection for test generation and multiple faults injection for parallel fault simulation. Experimental results show that the system with only one of the SP's processors is twice as fast as the conventional software test generator running on a mainframe. In parallel processing by dividing the faults, the system with 8 processors is four times faster than that with one processor on average.

1. はじめに

論理回路の高集積化にともない、テストパターンの生成に要する時間は回路規模の2乗に比例して増加するといわれている。また、ASICの登場により製品のターンアラウンドタイムは短縮しており、大規模回路に対する高速なテスト生成システムの要求が強くなっている。大規模な回路に対しては、スキャンのようなテスト容易化設計により回路の一部を組合せ回路として切り出してテストパターンを自動生成する[1]。従来の組合せ回路に対するテスト生成システムは大形計算機上で走行するソフトウエアにより実現されており、優秀なアルゴリズムがいくつか提案されている[2-5]。しかし、処理速度や対象回路の規模に関してはマシンの性能によるところが大きく、現場では数時間のCPU時間をかけてテストパターンを生成する例も少なくない。また、並列処理よるアプローチも提案されているが大形計算機の速度を凌駕するには至っていない[6]。そこで、今後大規模化する回路のテスト生成における速度および容量の問題に対処するために、我々は論理シミュレーション専用マシンＳＰ[8]を応用したテスト自動生成システムを提案した[9,10]。

本システムは、図１に示すようにテスト生成の対象となる被検査回路と仮定故障（単一縮退故障）のリストが与えられると、ホスト上で「テスト生成回路」と呼ばれる回路を合成する。テスト生成回路は、ある故障を検出するパターンを入力の探索により求める処理（テスト生成）や、ある入力の元で検出可能な故障を求める処理（故障シミュレーション）を直接実行する回路である。テストパターンは、このテスト生成回路の動作を論理シミュレーション専用マシン上で高速に模擬した結果として生成される。

これまでにプロトタイプを作成しベンチマーク回路に適用した結果、２、３の回路に対して良好な検出率を得ることができなかった。そこで、テスト生成回路の動作を解析した結果、次の２つが原因であることがわかった。

(1)入力の探索順序がテスト生成回路合成時に固定されるため、特に入力数の大きい回路で目標故障によっては無駄な空間を探索してしまう。

(2)テスト生成回路は、４ビットの信号線上で正常回路と２個の故障回路を並列にシミュレーションする符号を採用しているが、解像度の限界により故障がむやみにアボートされた。

本文では、上記２点の問題に対処するためにテスト生成回路に改良を行い検出率を向上することに成功したので報告する。ＳＰの１台のプロセッサのみを用いたシステムをベンチマーク回路に適用した結果、大形計算機上のシステムと比較して平均２倍の高速化と互角の検出率を達成できた。また、８台までのプロセッサを用いて単純に故障を分割することによる並列処理を行った結果、さらに約４倍高速化できることがわかった。さらに、合成されたテスト生成回路の規模は被検査回路の規模の平均10倍であることから本システムは40万ゲートという従来のシステムでは取り扱いが困難な大規模な回路への適用が期待できる。

2. テスト生成回路

テスト生成回路は、被検査回路に４つのモジュールを付加することにより合成される。入力探索器は被検査回路の入力に接続し、ランダムパターンを発生するモードと、入力の縦型探索を行うモードにより入力パターンを発生する。故障挿入器は被検査回路中の信号線に埋め込まれ被検査回路に対して複数個の単一縮退故障を順次並列に挿入する。出力検査器は被検査回路の出力を検査し、故障が検出可能かどうかまたは不明かを判断する。同期制御器はテスト生成アルゴリズムに従って入力探索器や故障挿入器の動作をクロックにより制御する。すなわち、故障挿入器を固定して入力探索器を動作させることにより目標故障に対するテストパターンを生成する。また、入力探索器を固定して故障挿入器を動作させることにより故障シミュレーションを行い等価故障を検出する。

3. テスト生成回路のアルゴリズム

本システムでは、テスト生成回路のアルゴリズムとして次の２つを実現している。

RP_test：被検査回路の入力としてランダムパターンを発生し、その元で故障シミュレーションを行い故障を検出する。あらかじめ設定された数のランダムパターンを発生するか、故障が検出されずにあるパターン数を発生したとき処理を終了する。

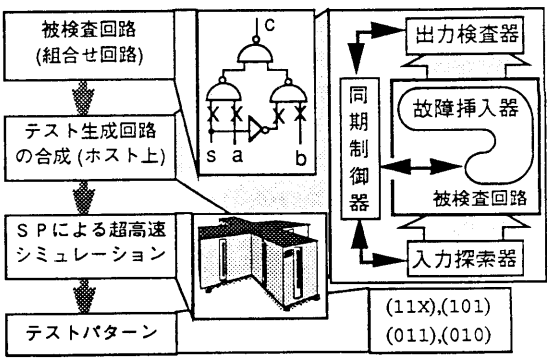


図１．本システムの構成

AL_test：被検査回路中に目標故障を仮定し、入力の縦型探索を行ってテストパターンを生成する。テストパターンが求められたならば、その入力の元で故障シミュレーションを行って等価故障を検出する。テスト生成回路は、入力がバックトラックしたとき故障の検出の可能性が不明となった回数をカウントし、その値が規定数に達したときに探索を打ち切って目標故障をアボートする機構を持つ。また、目標故障が検出されることなく入力の全空間の探索が終了したとき、その目標故障の冗長性が証明される。全ての故障が検出されるか、アボートされるか、冗長性が証明されたとき処理を終了する。

テスト生成の最初の段階としてRP_testを適用すると、アルゴリズミックな手法と比較してより短い時間である程度の検出率を達成することができる。しかし、RP_testでは、未検出故障が冗長であることを証明するには実用的でなく、また、適用パターン数が増加するにつれて検出率の伸びが飽和してくる。そこで、本システムでは、まずRP_testを適用し、適切な時期を見計らってAL_testに変更する（図２）。

4. 入力探索器によるテスト生成

4.1. 入力の縦型探索

本システムでは、被検査回路の入力ベクタの空間を縦型探索することにより目標故障に対するテストパターンを求める（図３）。まず被検査回路の全入力を論理値X(不定値)で初期化する（リセット）。次に、故障の検出の可能性が不明であるかぎり値がXである入力を１つ選びその値を０とする（前進）。故障が検出できないことが判明したとき、最後に０を割り当てた入力の値を１とし、それ以後に１が割り当てられた入力の値をXとする（バックトラック）。

この手法は、被検査回路の入力にバックトラックを発生させるという点でPODEM[3]に似ている。しかし、PODEMが被検査回路中の信号線を監視してバックトラックを発生するのに比べ、本手法は被検査回路の出力のみを監視してバックトラックを発生している。また、PODEMでは、次に値を決定する入力を後方追跡により動的に決定するが、本システムでは入力探索器と被検査回路はテスト生成回路を合成する際に静的に接続されるので被検査回路の入力の値を決定する順番が目標故障によらず固定される。このため、目標故障によっては故障の伝播に関係ない無駄な入力空間を探索してしまいパターンを求めるまでに必要以上の時間を要することがわかった。

この問題に対処するために次節に示すように入力探索器の内部を階層化し、目標故障に応じて探索の順序を変更できるようにした。

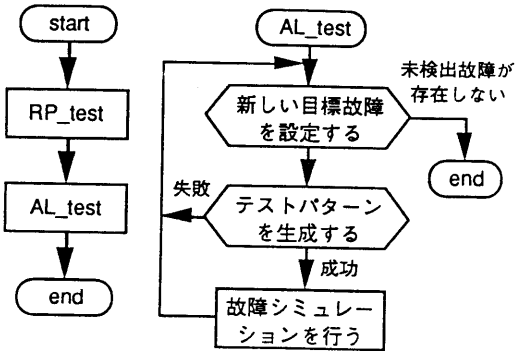
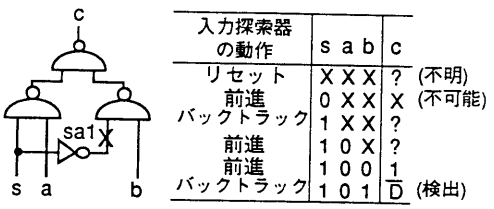


図２．テスト生成回路のアルゴリズム



入力探索器の動作	s	a	b	c	
リセット	X	X	X	?	(不明)
前進	0	X	X	X	(不可能)
バックトラック	1	X	X	?	
前進	1	0	X	?	
前進	1	0	0	1	
バックトラック	1	0	1	D	(検出)

図３．入力の縦型探索

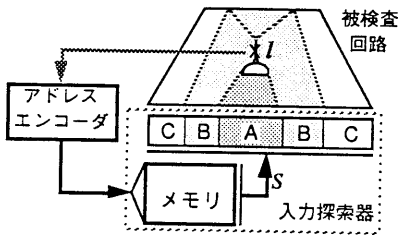


図４．入力の三階層探索

4.2. 入力の三階層探索

ある目標故障に対するテストパターンを入力の探索に基づいて生成する場合、まず故障を活性化するための入力の値を求めた後、故障の影響を出力に伝播するための入力の値を求めることで処理を高速化できると考えた。そこで、入力の探索を階層的に行うために、各故障に対して被検査回路の入力を次のように３つの集合に分割する。

[集合A] 故障を仮定する信号線*l*へのパスが存在する入力の集合。すなわち、目標故障を活性化するための入力の集合である。

[集合B] 信号線*l*から到達可能な出力に対するパスが存在する入力でAに属しないもの。すなわち、目標故障の

影響を出力に伝播するための入力の集合である。
[集合C]A、Bに属しない入力。すなわち、目標故障の伝播には関係のない入力の集合である。

この分割の処理はテスト生成回路を合成する際に行われ、結果は行列として入力探索器内部のメモリにあらかじめロードされる。テスト生成が開始すると、入力探索器はそのメモリから目標故障に対応した入力集合の情報を制御信号Sとして読み出し、探索順序を変更する（図4）。入力探索器内部にはあらかじめ3つの探索の階層が実現されており、制御信号Sに応じてまずAに属している入力の値を決定し、次にBに属している入力の値を決定することによりテストパターンを生成する。Cに属している入力に対しては、故障シミュレーション時に他の故障の検出の可能性が不明となった時に値が設定される。

5. 故障挿入器による改良型並列故障シミュレーション

テスト生成回路における故障シミュレーションは並列法[11]を改良した方式を採用している。ＳＰは4ビットマシンであるのでテスト生成回路の信号線も4ビット幅であるが、従来の2倍の密度で故障を挿入できる符号化法により、テスト生成回路は正常回路と2個の故障回路を並列にシミュレーションする[10]。そこで、仮定故障の集合を2分割し、各集合に属している故障を割り当てられたビットに順に挿入する。フォールトドロップの機能を実現し、また、局所的に有効な故障のみを処理の対象とすることにより、従来法に比べて故障シミュレーションの回数を大幅に削減している。ここで、局所的に有効であるとは、故障を仮定する信号線にその故障を活性化する値が割り当てられていることをいう。

しかし、テスト生成時に並列に挿入された複数個の故障を対象として探索を行った場合、符号の解像度の限界からバックトラックの原因がどの故障に起因したかを判定することが困難な場合があり、このため故障が過剰にアボートしてしまう状況があることがわかった。そこで、故障挿入器に改良を加え、テスト生成時にはただ一つの故障を対象とし、故障シミュレーション時は複数個の故障を並列に挿入するようにした。

6. 論理シミュレーション専用マシンＳＰ

本システムにおいてテスト生成回路の動作を模擬するために使用する論理シミュレーション専用マシンＳＰは、64台のプロセッサが並列に動作し、400万素子の大規模論理回路の動作を大形計算機上のソフトウエアシミュレータに比べて1000倍以上高速に模擬する[8]。

7. 実験結果

7.1. 単プロセッサによる実験

7.1.1. 実験条件

ISCASのベンチマーク回路[7]に本システムを適用して実験を行った。実験では、まず全故障を対象とした1個のテスト生成回路を合成し、ＳＰの1台のプロセッサを用いてその動作を模擬した。テスト生成回路を合成する際、被検査回路の入力と入力探索器との接続順は入力が回路の記述ファイルに出現する順とした。また、故障はファイルに出現する順に番号を付け、奇数と偶数とで2分割し、番号の小さい順に挿入するようにした。

テスト生成回路のアルゴリズムとして、まずRP_testを適用した。RP_testでは、32個のランダムパターンを発生する度にシミュレーションを停止し、この区間で故障が1つも検出されなかったときに処理を終了した。その後、未検出の故障に対してテスト生成回路を再合成しAL_testを適用した。再合成されたテスト生成回路は元の回路に比べて検出した故障のモジュールを被検査回路中から削除しただけであり、被検査回路と入力探索器の接続、および未検出故障の挿入順序は元の回路と同じにした。AL_testでは、入力探索器に256回のバックトラックが発生したときに故障をアボートした。

7.1.2 テスト生成回路の規模

最初に合成を行った際のテスト生成回路の規模（素子数）の被検査回路の規模（論理ゲート数+外部入出力数）に対する増加比を表1に示す。平均で約10倍増加していることがわかる。ＳＰの容量は400万素子であるから、64台の全プロセッサを使用することにより本システムは40万ゲートという従来のシステムでは取り扱いが困難な大規模な回路への適用が期待できる。

7.1.3 実験結果

ＳＰのプロセッサ1台を使用してテスト生成回路のシミュレーションを行った結果を表1に示す。比較のために、本文に示した改良を行う以前の方式により実行した場合の結果を併せて示す。検出率は、検出した故障数を全故障数から冗長であることが証明された故障数を引いた値で割ったものである。本文で提案した手法により検出率が向上していることがわかる。速度については、回路によって増減様々である。これは、本文に示した機構を回路として実現したことによるテスト生成回路の動作量の増加と、以前アボートしていた故障がこの改良の効果で検出されるか冗長性が証明されたことによる探索の手数の減少とのトレードオフによる。さらに、大形計算機上のシステムであるFAN[4]と性能を比較すると、検出率は互角であり、速度は平均して2倍高速であることがわかった。

7.2. 複数プロセッサによる実験

7.2.1. 実験条件

ＳＰは64台のプロセッサを有するが、この内８台までのプロセッサを使用して並列処理性能を評価した。並列処理に対するアプローチについては現在研究中であるが、最も簡単な手法として故障を分割することによる並列処理の効果を評価した。実験では、ｎ台のプロセッサを使用する場合、故障を２ｎ個の部分集合に分割し、それを２個ずつ組にしてｎ個のテスト生成回路を合成した。7.1.1.節と同様の実行条件の元で各テスト生成回路の動作をそれぞれ１台のプロセッサを使用して模擬した。

7.2.2. 実験結果

10回路中ゲート数の大きい５回路について結果を表２に示す。加速比はプロセッサ１台で処理した場合の時間を１としたときの速度比を表す。アルゴリズム別にみると、RP_testの方がAL_testに比べ加速比が大きい。これは、RP_testを開始する際テスト生成回路間で故障を等分割しているのに比べ、AL_testにおいては、RP_testに比べて処理の対象となる故障数が少なく、RP_testの実行後未検出であった故障数がテスト生成回路間でばらつくためであると考える。しかし、全体でみると、８台使用した場合５回路の平均で3.8倍の加速を得ている。今後、より有効な並列処理方式を採用することによりさらに高速化が期待できる。

８．おわりに

論理シミュレーション専用マシンＳＰを応用したテスト生成システムにおいて、テスト生成回路に対して次の２点の改良を行うことにより検出率を向上することに成功した。

(1)テスト生成時に目標故障に応じて入力の探索の順番を変更する機構により、無駄な空間を探索しないようにした。

(2)テスト生成時には１個の故障のみを対象とし、故障シミュレーション時には故障を並列に挿入する機構により、符号の解像度の限界から生じる検出率の低下を防いだ。

本システムをベンチマーク回路に適用した結果、１台のみのプロセッサを使用して、大形計算機上のシステムと比較して互角の検出率を持つテストパターンを平均２倍高速に生成することがわかった。また、故障を分割して並列処理を行った結果、８台のプロセッサを使用して約４倍の加速を達成できることがわかった。さらに、本システムは40万ゲートという従来のシステムでは取り扱いが困難であった大規模な回路への適用が期待できる。

今後の課題は、数万ゲート以上の大規模な回路に適用して性能を評価することと、複数台のプロセッサを有効に利用する並列処理方式を構築することである。

表１．本システムの性能（１プロセッサを使用）

回路名	全故障数	テスト生成回路の素子増加率	改良後		改良前	
			検出率＊	時間 [s]	検出率＊	時間 [s]
c432	524	11.88	99.24%	1.04	99.24%	0.95
c499	758	10.45	98.94%	1.66	98.94%	1.25
c880	942	9.95	100.00%	1.54	100.00%	1.54
c1355	1574	11.25	99.49%	4.01	99.49%	3.96
c1908	1879	8.60	99.52%	9.16	99.10%	10.24
c2670	2747	8.50	96.11%	35.32	95.12%	30.35
c3540	3428	8.71	98.24%	19.36	97.31%	21.57
c5315	5350	9.33	99.66%	17.39	98.75%	21.09
c6288	7744	12.67	99.97%	19.49	99.56%	30.85
c7552	7550	8.76	97.46%	58.04	97.07%	52.76

$$\text{＊検出率} = \frac{\text{検出した故障数}}{\text{全故障数} - \text{冗長性を証明した故障数}}$$

表2．並列処理効果

回路名	プロセッサ数	RP_test		AL_test		合計	
		時間 [s]	加速比	時間 [s]	加速比	時間 [s]	加速比
c2670	1	6.14	1.00	29.18	1.00	35.32	1.00
	2	3.65	1.68	19.52	1.49	23.17	1.52
	4	2.15	2.86	13.26	2.20	15.41	2.29
	8	1.41	4.35	9.01	3.24	10.42	3.39
c3540	1	10.22	1.00	9.14	1.00	19.36	1.00
	2	6.66	1.53	6.12	1.49	12.78	1.51
	4	4.00	2.56	4.29	2.13	8.29	2.34
	8	2.65	3.86	2.82	3.24	5.47	3.54
c5315	1	13.39	1.00	4.00	1.00	17.39	1.00
	2	8.21	1.63	2.71	1.48	10.92	1.59
	4	5.08	2.64	2.28	1.75	7.36	2.36
	8	3.26	4.11	1.56	2.56	4.82	3.61
c6288	1	18.71	1.00	0.78	1.00	19.49	1.00
	2	10.52	1.78	0.75	1.04	11.27	1.73
	4	5.89	3.18	0.45	1.73	6.34	3.07
	8	3.59	5.21	0.41	1.90	4.00	4.87
c7552	1	24.53	1.00	33.51	1.00	58.04	1.00
	2	15.15	1.62	21.23	1.58	36.38	1.60
	4	8.97	2.73	14.17	2.36	23.14	2.51
	8	5.75	4.27	9.74	3.44	15.49	3.75

参考文献

[1] E. J. McCluskey, "Logic Design Principles," Prentis-Hall, pp. 433-447, 1986.

[2] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and Method," IBM J. of Research and Development, vol.10, pp. 278-291, July 1966.

[3] P. Goel, " An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. Comput., Vol. C-30, pp. 215-222, March 1981.

[4] H. Fujiwara, "FAN: a Fanout-oriented Test Pattern Generation Algorithm," Proc. Int. Symp. on Circuits and Systems, pp. 671-674, June 1985.

[5] M. H. Schulz and E. Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," 18th FTCS, pp. 30-35, June 1988.

[6] A. Motohara, K. Nishimura, H. Fujiwara, and I. Shirakawa, "A Parallel Scheme for Test-Pattern Generation," Proc. of Int. Conf. on CAD, pp. 156-159, November 1986.

[7] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," Proc. Int. Symp. on Circuits and Systems, June 1985.

[8] F. Hirose et al., "Simulation Processor SP," Proc. of Int. Conf. on CAD, pp. 484-487, November 1987.

[9] F. Hirose, K. Takayama, and N. Kawato, "A Method to Generate Tests for Combinational Logic Circuits using an Ultrahigh-speed Logic Simulator," Proc. of Int. Test Conf., pp. 102-107, September 1988

[10] 広瀬文保, 高山浩一郎, 川戸信明, "テスト生成回路を用いた高速テスト生成方式," 信学論(D-I), Vol. J72-D-I, No.9, pp. 678-684, 1989年9月.

[11] E. J. Tompson and S. A. Sygenda, "Digital logic simulation in a time-based, table-driven environment, Part2, Parallel Fault Simulation," Computer, Vol. 8, No. 3, pp.38-40, 1975.