ISSN 2186-5647

2-128

規則 G に基づくテスト環境生成アルゴリズムの評価

日大生産工(学部)　○西間木 淳　日大生産工　細川利典

1. はじめに

近年，半導体集積技術の発達に伴い，大規模集積回路(Large Scale Integrated circuits：LSI)の大規模化，複雑化が急速に進んでいる．これに伴い，LSI のテスト生成時間の増大が問題となっている．組合せ回路に対しては，効率の良いテスト生成アルゴリズムが提案されており，大規模な回路に対しても，高い故障検出効率を達成するテスト集合を比較的容易に生成することができる[1]-[5]．しかしながら，順序回路に対するテスト生成は，たとえ小規模な回路であっても，現実的な時間で高い故障検出効率を達成するテスト集合を得ることが困難な場合がある．

現在，LSI のテストでは，テスト容易化設計としてフルスキャン設計[6]を施した回路に対して，自動テストパターン生成ツール(Automatic Test Pattern Generator：ATPG)を用いてテストパターンを生成し，テストを実行する手法が広く用いられている．フルスキャン設計を施すことにより，テスト対象の順序回路を疑似的に組合せ回路とみなして，テスト生成を実行することが可能となる．しかしながら，回路中のフリップフロップをスキャンフリップフロップで構成し，それらをシフトレジスタ状に接続することによる面積オーバーヘッドや，テスト実行時間の増加が問題となっている．それゆえ，非スキャンに基づく効率の良い順序回路のテスト生成技術が求められている．

一方，LSI の大規模化，複雑化に伴い，設計期間の長期化や，設計コストの増大が問題となっている．現在の LSI 設計において主流であるレジスタ転送レベル(Register Transfer Level：RTL)設計では，回路の構造情報を設計者が詳細に定義する必要があるために，設計生産性の観点から，設計が困難となっている．この問題を解決するために，RTL よりもさらに抽象度の高い動作レベルの回路記述から，RTL 回路記述を自動生成する動作合成[7]が提案されている．RTL 回路記述が回路構造情報を含むのに対して，動作記述には回路で実現したい機能動作のみを記述すればよいため，記述量が少なく設計生産性に優れる[7]．

動作合成は，一般にスケジューリング[7]及びバインディング[7]の 2 つのフェーズから構成される．スケジューリングは，動作記述中の各演算操作を実行する時刻を決定するフェーズである．バインディングは，動作記述中の変数，演算操作に対して，それぞれレジスタ及び演算器を割当てるフェーズである．スケジューリングを実行した後のサイクル精度な記述の回路を，機能 RTL 回路と呼び，スケジューリング及びバインディングを実行した後の，変数と演算操作がそれぞれレジスタ及び演算器に割当てられた回路を構造 RTL 回路と呼ぶ．

機能 RTL 回路を対象とした，縮退故障の階層テスト生成法[8]が過去に提案されている[9]．文献[9]では，ADD(Assignment Decision Diagram)[10]と呼ばれるグラフで表現された機能 RTL 回路を対象に，各機能要素のテスト環境を生成し，機能要素に対応するゲートレベル回路の縮退故障テストパターンを，テスト環境に代入することでテスト系列を生成している．文献[9]では，従来のゲートレベルの順序テスト生成と比較して，高速にテスト系列を生成できることが報告されており，設計の上流工程からテスト生成を行うことの有効性が示されている．

本論文では，文献[9]で提案された，9 種類の記号から成る正当化規則及び伝搬規則(以下規則 G と呼ぶ)に基づくテスト環境生成アルゴリズムを用いて，動作合成ベンチマーク回路に対してテスト環境を生成し，その結果について考察する．また，得られたテスト環境からテスト系列を生成し，その故障検出率，テスト系列長をゲートレベルの順序テスト生成と比較評価する．

2. ADD

機能 RTL 回路を表現する ADD について説明する．図 1 に示すように ADD は，読込みノード，書込みノード，演算ノード，割当決定ノード(Assignment Decision Node：ADN)の 4 種のノードから構成される．演算ノードは算術演算及び論理演算を，読込みノードは外部入力，変数，定数を，書込みノードは外部出力及び変数をそれぞれ表現する．

Evaluation of Test Environment Generation Algorithm based on Rule G

Jun NISHIMAKI and Toshinori HOSOKAWA

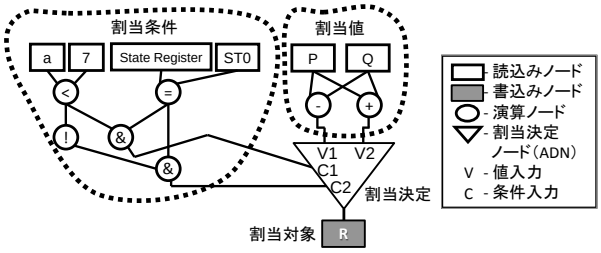

図 1. ADD

ADN では，論理演算により計算される条件入力の値に応じて，どの値入力を出力に割当てるかが決定される．読込みノードから書込みノードまでのデータ転送は 1 クロックサイクルで実行され，書込みノードに対して値が割当てられると，対応する読込みノードの値が更新される．

ADD は本来動作合成における動作記述の内部表現として提案されたものであるが，機能 RTL 回路を表現するために用いることもできる．また，ADD はコントローラとデータパスを一様に表現することが可能であるため，コントローラとデータパスを分離せずにテスト生成を行うモデルとして適している．

3. ADD を用いた階層テスト生成

ADD で表現された機能 RTL 回路に対する階層テスト生成は，テスト環境生成及びテスト系列生成の 2 つのフェーズから構成される．

3. 1. テスト環境生成

ADD を構成する各ノードをテストするためには，外部入力からテスト対象ノードの入力に対して任意の入力値を正当化し，テスト対象ノードの任意の出力応答を外部出力まで伝搬する必要がある．外部入力からテスト対象ノードの入力に，任意の入力値を正当化する経路を正当化経路，テスト対象ノードの任意の出力応答を外部出力まで伝搬する経路を伝搬経路と呼び，正当化経路及び伝搬経路を有効とするための，外部入力の時系列をテスト対象ノードのテスト環境と呼ぶ．文献[9]では，9 種類の記号から成る 9 値代数を用いて，ADD を構成する各ノードのテスト環境を生成する．図 2 にテスト環境の例を示す．

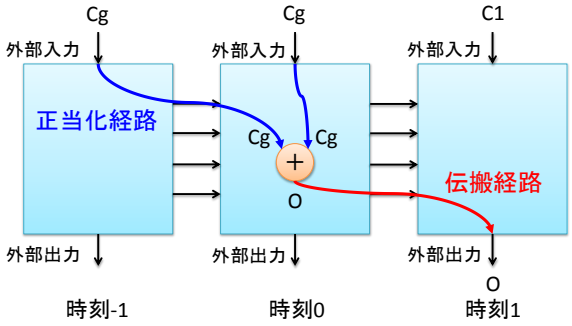

図 2. テスト環境

図 2 は，ある ADD の加算ノードに対して生成されたテスト環境を示している．時刻 0 における加算ノードの正当化経路，及び伝搬経路を有効とする外部入力時系列である，Cg，Cg，C1 が加算ノードのテスト環境に対応する．各記号の持つ意味については，第 4 章で詳細に説明する．

3. 2. テスト系列生成

テスト環境生成後，あらかじめライブラリ化しておいた，各ノードのテストパターンをテスト環境に代入することでノードに対するテスト系列を生成する．ここで用いるテストパターンのライブラリは，ADD を構成するノードの機能ごとに論理合成を行い，得られたゲートレベル回路に対して組合せ回路のテスト生成を実行することで生成されたテスト集合である．但し，機能 RTL 回路を合成することで得られるゲートレベルの回路構造と，ライブラリ生成時に想定していたゲートレベルの回路構造が異なる可能性があるため，最終的に得られたテスト系列を用いてゲートレベルで故障シミュレーションを実行し，故障検出率を算出する必要がある．

4. 規則 G に基づくテスト環境生成

文献[9]で提案されたテスト環境生成規則 G は，以下の 9 種類の記号から構成される．各記号はノードに対して論理値を取り，記号が持つ条件を達成可能な場合には論理値 1 を，達成不可能な場合には論理値 0 を取る．

- Cg：任意の値に制御できること．
- Cq：定数に制御できること．
- C0：値 0 に制御できること．
- C1：値 1 に制御できること．
- Ca1：全てのビットを 1 に制御できること．
- Cz：ハイインピーダンスに制御できること．
- Cs：任意の状態に制御できること．
- O：伝搬する任意の誤りを観測できること．1 ビットの場合は，1 が 0 に誤る故障の影響を観測できること．
- O’：1 ビットのみで定義され，0 が 1 に誤る故障の影響を観測できること．

テスト環境を生成するに当たり，まずテスト対象ノードに対して，テスト環境を生成するための初期目標を設定する．図 3 に加算ノード及び比較ノードに対するテスト環境目標を示す．

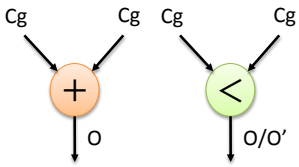

図 3. テスト環境目標

テスト対象ノードの各入力には，ゲートレベルのテスト生成により得られた，任意のテストパターンを入力する必要があるため，ノードの各入力に，任意の値に制御可能であることを意味する記号 Cg を割当てる．また，テスト対象ノードの出力に現れる任意の誤りを観測する必要があるため，ノードの出力に任意の誤りを観測可能であることを意味する記号 O を割当てる．比較ノードの出力に現れる誤りは，テストパターンによって 1 が 0 に誤る故障影響と，0 が 1 に誤る故障影響の 2 通りが考えられる．したがって，比較ノードに対しては，出力に O を割当てる目標と，O'を割当てる目標の 2 通りのテスト環境目標を設定する必要がある．

ノードの種類ごとに定義される記号の正当化規則及び伝搬規則を適用し，テスト対象ノードの各入力に割当てられた Cg を，外部入力まで正当化すること，また，テスト対象ノードの出力に割当てられた O または O'を外部出力まで伝搬することが，テスト環境生成における目標となる．

5. 実験

本論文では，文献[9]で提案された規則 G に基づくテスト環境生成アルゴリズムを用いて，文献[11]に掲載されている動作合成ベンチマーク回路に対して階層テスト生成を行い，ゲートレベルの順序テスト生成との比較評価を行った．ゲートレベル ATPG には Synopsys 社の TetraMax を用いた．実験結果を表 1 に示す．表 1 の第 2 列目は，ADD のテスト対象ノード数に対して，テスト環境が生成されたノード数の割合を意味するテスト環境生成率[12]である．第 3 列目及び第 4 列目は，それぞれ階層テスト生成により得られたテスト系列の故障検出率，テスト系列長を表し，第 5 列目及び第 6 列目は，それぞれ TetraMax により生成されたテスト系列の故障検出率，テスト系列長を表す．

Tseng に関しては，階層テスト生成の方がより高い故障検出率を示している．これに対して Paulin では TetraMax により生成されたテスト系列の方が，より高い故障検出率を達成している．ADD を用いた階層テスト生成では，ノードに対して生成されたテスト環境に，ゲートレベル回路のテストパターンを代入することでテスト系列を生成する．したがって，テスト環境が生成できなければ，そのノードをテストするためのテスト系列も生成することができない．テスト環境生成率の低かった Paulin では階層テスト生成よりも，ゲートレベル ATPG の方が高い故障検出率を示している．Paulin には，片側入力が定数の乗算ノードが存在し，このノードが他のノードのテスト環境生成の妨げとなっていた．

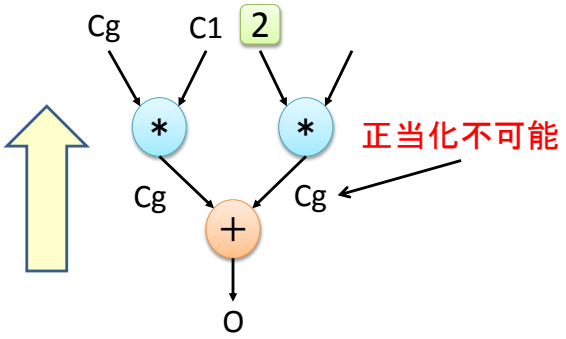

図 4. 正当化失敗例

例えば，図 4 の加算ノードがテスト対象ノードであった場合，加算ノードの両入力にはテスト環境目標として，制御記号 Cg が割当てられる．テスト環境を生成するために，Cg を入力側に正当化することになるが，ここで，加算ノードの右入力に割当てられた Cg は正当化することができない．加算ノードの右入力には，定数 2 を入力に持つ乗算ノードの出力が接続されているため，加算ノードの右入力は，必ず偶数の値を取ることになる．加算ノードの右入力を任意の値に制御することはできないため，Cg の正当化に失敗する．

文献[9]の手法では，構造 RTL 回路におけるマルチプレクサの故障を検出するために，ADN に対してもテスト環境を生成しているが，今回の実験では演算ノードに対してのみテスト環境を生成した．その結果，テスト環境生成率の高い Tseng においては，ほぼ全てのマルチプレクサの故障が検出可能であった．このことから，演算ノードに対して生成されたテスト系列で，その演算ノードが割当てられた演算器周辺のマルチプレクサの故障は，ある程度検出可能であると考えられる．

テスト環境生成に成功したとしても，ゲートレベルの故障シミュレーションで，検出することのできない故障が存在した．これの 1 つの要因としては，バインディング時に行ったリソースの共有化が考えられる．テスト環境生成に成功したノードと，そのノードの正当化経路及び伝搬経路上に存在するノードが，同一のリソースにバインディングされると，構造 RTL 回路における正当化経路及び伝搬経路は，テスト対象のモジュールを複数回通過することになる．このとき，故障によっては，テスト対象モジュールの入力に与えるべきテストパターンのビット列が変化してしまう可能性や，伝搬すべき誤りがマスクされてしまう可能性がある．

今回の実験では，階層テスト生成において，テスト環境を手動で計算したため，計算時間の比較評価は行っていないが，階層テスト生成は，ゲートレベル ATPG よりも高速にテスト系列を生成できることが報告されている[11]．

6. おわりに

本論文では，文献[9]で提案された規則 G に基づくテスト環境生成アルゴリズムを用いて，動作合成ベンチマーク回路に対してテスト環境を生成し，その結果を考察した．また，得られたテスト環境を基にテスト系列を生成し，その故障検出率，テスト系列長をゲートレベルの順序テスト生成と比較評価した．今回，実験対象回路として用いた小規模なベンチマーク回路では，実験結果に大きな差異が認められなかった．今後の課題としては，テスト環境生成，及びテスト系列生成の実装と，更なる大規模回路での実験が挙げられる．

参考文献

[1] M.Schulz, E.Trischler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Trans. on Computer-Aided Design, Vol. 7, No. 1, pp. 126-137, Jan. 1988.

[2] W. Kunz and D. Pradhan,"Recursive Learning: An Attractive Alternative to the Decision Tree for Test Genration in Digital Circuits," Proc. IEEE International Test Conference on Discover the New World of Test and Design, pp.816-825, 1992.

[3] M.Henftling, H.C.Wittmann, K.J.Antreich, "A Single-Path-Oriented Fault- Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization," Proc. 1995 IEEE/ACM international conference on Computer-aided design, pp.304-309, 1995.

[4] C.Wang, S.Reddy, I.Pomeranz, X.Lin, J.Rajski,"Conflict driven techniques for improving deterministic test pattern generation," Proc. IEEE International Conference on Computer-Aided Design, pp.87-93, 2002.

[5] E. Gizdarski and H. Fujiwara, "SPIRIT: a highly robust combinational test generation algorithm," IEEE Trans. on Computer Aided Design for Integrated Circuits and Systems, Vol. 21, No. 12, pp. 1446-1458, Dec. 2002.

[6] H.Fujiwara: Logic Testing and Design for Testability, The MIT Press, 1985

[7] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin, HIGH-LEVEL SYNTHESIS Introduction to Chip and System Design, Kluwer Academic Publisher, 1992.

[8] B. T. Murray and J. P. Hayes, "Hierarchical test generation using pre computed tests for modules," IEEE Trans. Computer-Aided Design, Integrated Circuits & Syst., vol.9, no.6, pp.594-603, Jun. 1990.

[9] I. Ghosh, M. Fujita, "Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams," Proc. of ACM/IEEE Design Automation Conference, pp.43-48, Jun. 2000.

[10] V. Chaiyakul, D. D. Gajski, and L. Ramachandran, "High-Level Transformations for Minimizing Syntactic Variances," in Proc. Design Automation Conference, pp. 413-428, Jun. 1993.

[11] Mike Tien-Chien Lee : High-Level Test Synthesis of Digital VLSI Circuits, Artech House, pp.98

[12] H. Fujiwara, C. Y. Ooi and Y. Shimizu, "Enhancement of test environment generation for assignment decision diagrams," 9th IEEE Workshop on RTL and High Level Testing (WRTLT'08), pp.45-50, Nov. 2008.

表 1. 実験結果

回路名	階層テスト生成			ゲートレベルATPG	
	テスト環境生成率	故障検出率	テスト系列長	故障検出率	テスト系列長
Tseng	87.50%	99.19%	342	98.63%	321
Paulin	33.33%	88.03%	103	97.58%	115