ISSN 2186-5647

テスト環境生成結果を用いた階層テストのためのバインディング法

日大生産工(院)　○西間木　淳　日大生産工　細川　利典
大阪学院大　藤原　秀雄

1. はじめに

近年，半導体集積技術の発達により，設計される大規模集積回路(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．

組合せ回路に関しては，効率の良いテスト生成アルゴリズムが提案され，大規模回路に対しても比較的短時間で高い故障検出率を達成することができる[1]-[5]．これに対して順序回路のテスト生成は，問題の解空間が組合せ回路のテスト生成と比較して大きく，現実的な時間で高い故障検出率を達成することが困難である．

一方，LSI の大規模化，複雑化に伴い，LSI の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．これに伴い，回路設計手法において現在の主流であるレジスタ転送レベル(Register Transfer Level：RTL)設計が困難になってきている．この問題を解決するために，RTL よりも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[6]が提案され，LSI 設計現場で普及が進んでいる．

動作合成は，動作記述中の各演算操作を実行する時刻を決定するスケジューリングと，演算操作及び変数に対して，演算器及びレジスタをそれぞれ割当てるバインディングの 2 つの処理から構成される[6]．スケジューリングが完了し，クロックサイクル毎の動作が決定した RTL 回路を機能 RTL 回路と呼ぶ．スケジューリング及びバインディングが完了し，クロックサイクル毎の動作と回路構造が決定した RTL 回路を構造 RTL 回路と呼ぶ．

効率の良い順序回路のテスト生成手法として，機能 RTL 回路情報を利用した階層テスト生成法が提案されている[7]-[9]．文献[7]-[9]で提案された階層テスト生成法は，機能 RTL 回路を構成する演算操作や変数などの各機能要素を階層の基本単位としてテスト系列を生成する．テスト系列の生成後，階層テスト生成に利用された機能 RTL 回路は動作合成，または論理合成の入力となり，バインディングの処理を経てゲートレベル回路に合成される．階層テスト生成において生成されたテスト系列は，バインディング適用前の機能 RTL 回路情報を利用して生成されたテスト系列であるため，テスト系列の故障検出能力，すなわちそれらのテスト系列でテスト可能な演算器数と最終的に検出可能な故障数はバインディング結果に依存すると考えられる．このことに着目し，階層テスト生成結果を利用して演算器バインディングを行うことで，階層テストにおいてより高い故障検出率を達成するテスト容易化バインディング法が提案されている[10][11]．これらのバインディング法は，演算器バインディングのみでテスト容易性の向上を図るものであるが，演算器バインディングに加えレジスタバインディングにおいても，テスト容易性の向上を指向することで，階層テストにおいてより高い故障検出率を達成できると考えられる．

本論文では，機能 RTL 回路情報を用いた階層テスト生成結果を利用して，演算器及びレジスタバインディングを行うことで，階層テスト可能な演算器数を増加させ，階層テストにおいてより高い故障検出率を達成するテスト容易化バインディング法を提案する．第 2 章で，機能 RTL 回路情報を用いた階層テスト生成について述べ，第 3 章で，提案するテスト容易化バインディング法について述べ，第 4 章で，実験結果を示し，最後に第 5 章で，結論と今後の課題について述べる．

2. 機能 RTL 回路情報を用いた階層テスト生成

機能 RTL 回路情報を用いた階層テスト生成は，テスト対象の機能 RTL 回路をコントロールデータフローグラフ(Control Data Flow Graph：CDFG)[6]や，割当決定図(Assignment Decision Diagram：ADD)[12]と呼ばれるグラフで表現し，これらのグラフ上で階層テスト生成を行う．機能 RTL 回路情報を利用した階層テスト生成では，機能 RTL 回路を構成する演算操作や変数などの各機能要素を階層の基本単位としてテスト系列を生成する．各機能要素に対する階層テスト生成の処理は，テスト環境生成及びテスト系列生成の 2 つの手続きから構成される．

テスト対象機能要素の入力に対して，外部入力から任意の入力値を正当化する経路を正当化経路と呼び，テスト対象機能要素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ぶ．正当化経路及び伝搬経路と，それらの経路を有効にするための外部入力値

A Binding Method for Hierarchical Testing Using Results of Test Environment Generation

Jun NISHIMAKI, Toshinori HOSOKAWA and Hideo FUJIWARA

の時系列をテスト対象機能要素のテスト環境と呼ぶ．テスト対象の全機能要素に対してテスト環境生成を実行し，テスト環境が生成可能か否かの情報も取得する．

テスト系列生成では，各機能要素に対して生成されたテスト環境に対して，機能要素毎にあらかじめ生成しておいた縮退故障テストパターンを代入し，各機能要素をテストするためのテスト系列を生成する．ここで利用される縮退故障テストパターンは，各機能要素を論理合成して得られたゲートレベル回路に対して組合せテスト生成を実行することで得られるものである．

機能 RTL 回路情報を用いた階層テスト生成において各機能要素に対して生成されたテスト系列が，具体的にどのハードウェア要素をテストするテスト系列となるかはバインディング結果に依存する．次章では，階層テスト生成により生成されたテスト系列で，より多くの演算器がテスト可能となるようなテスト容易化バインディング法を提案する．

3. 階層テスト容易化バインディング法

本章では，提案する階層テスト容易化バインディング法について述べる．提案する階層テスト容易化バインディング法は，階層テスト生成結果を利用して演算器及びレジスタバインディングを行うことで，階層テスト生成により生成されたテスト系列で，可能な限り多くの演算器をテストすることを目的とする．

3.1 テスト容易化バインディング法の基本戦略

本論文では，演算器入力に対して任意のテストパターンが伝搬可能であり，かつ演算器の任意の出力応答が外部出力で観測可能である時，その演算器を階層テスト可能演算器と呼ぶ．これに対して，上記の階層テスト可能演算器の性質を満たさない演算器を階層テスト不可能演算器と呼ぶ．テスト環境生成に成功した演算は，その入力に対して任意のテストパターンが伝搬すること，及び任意の出力応答が外部出力まで伝搬することが保証されている．したがって，テスト環境生成成功演算が 1 つ以上マッピングされた演算器は階層テスト可能演算器となる．階層テスト可能演算器に対しては，その演算器をテストするためのテスト系列が階層テスト生成により生成されている．しかしながら，階層テスト不可能演算器に対しては，その演算器をテストするためのテスト系列が生成されていない．それゆえ，バインディングの結果，階層テスト不可能演算器が多数存在すると，階層テストで検出困難な故障数が増加し，結果として故障検出率が低下すると考えられる．したがって，階層テストにおいてより高い故障検出率を達成するためには，階層テスト可能演算器数を可能な限り増加させることが重要であると考えられる．提案するテスト容易化バインディング法は，与えられた制約を満たす範囲内で階層テスト可能演算器数の最大化を目指し，演算器及びレジスタバインディングを行う．

3.2 階層テスト容易化演算器バインディング

階層テスト可能演算器数の向上を目的とした演算器バインディング法が過去に提案されている[10][11]．これらの演算器バインディングは，各演算器に対して可能な限り 1 つ以上のテスト環境生成成功演算がマッピングされるようにバインディングを行う．文献[11]で提案された演算器バインディングは，テスト環境生成成功演算がマッピングされていない演算器に対して，優先的にテスト環境生成成功演算をマッピングすることで，ある演算器に対してテスト環境生成成功演算が集中的にマッピングされることを防いでいる．このような演算器バインディングを行うことで，階層テスト可能演算器数が増加し，テスト環境生成結果を考慮せずにバインディングを行った場合と比較して高い故障検出率を達成することに成功している．しかしながら，これらのテスト容易化演算器バインディングを行ったとしても，階層テスト不可能演算器が合成される場合がある．動作合成において，動作記述に記述された動作を RTL 回路で実現するために必要な演算器数はスケジューリング結果に依存する．この時，必要となる演算器数がテスト環境生成成功演算数よりも多い場合，文献[11]で提案された演算器バインディングを行ったとしても，テスト環境生成成功演算がマッピングされない演算器が残る．このことを図 1 に示すスケジューリング済み CDFG と，図 2 に示す階層テスト容易化バインディングの具体例を用いて説明する．

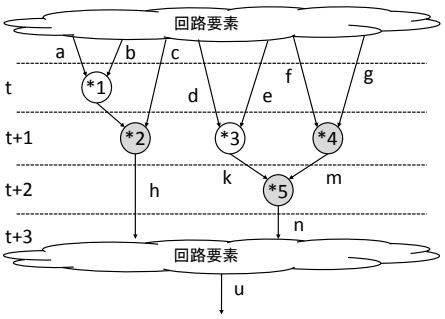

図 1. スケジューリング済み CDFG 例

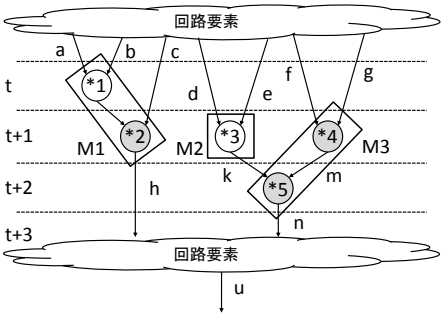

図 2. 階層テスト容易化演算器バインディング例

図 1 に示す CDFG において, 無色の演算ノードはテスト環境生成成功演算を表し，有色の演算ノードはテスト環境生成失敗演算を表す. 図 1 に示す CDFG では時刻 t+1 に 3 つの乗算がスケジューリングされている. 1 つの演算器はある時刻で 2 つ以上の演算を同時に実行することが不可能である．したがって，図 1 に示すスケジューリングのもとでは少なくとも 3 つの乗算器が必要となる．これに対してテスト環境生成成功演算は，*1 と*3 の 2 つであるため，図 2 に示すように可能な限り多くの演算器に対して，テスト環境生成成功演算がマッピングされるようにバインディングを行ったとしても，階層テスト不可能乗算器が合成される．図 2 に示す例では，乗算器 M1 及び M2 に関しては，それぞれテスト環境生成成功演算*1 と*3 がマッピングされているため階層テスト可能演算器であるが，テスト環境生成成功演算が 1 つもマッピングされていない乗算器 M3 は，現時点で階層テスト不可能演算器である．提案するテスト容易化バインディング法では，テスト環境生成成功演算の不足により出現する階層テスト不可能演算器を，後述する階層テスト容易化レジスタバインディングを行うことで，階層テスト可能演算器に換えることを試みる.

3.3 階層テスト容易化レジスタバインディング

提案する階層テスト容易化レジスタバインディングは，テスト環境生成成功演算の不足により現れる階層テスト不可能演算器を，階層テスト可能に換えることを目的としたバインディングを行う．階層テスト不可能演算器を階層テスト可能演算器とするためには，階層テスト不可能演算器の入力に対してその演算器のテストパターンを与えることができ，かつその演算器の出力応答を外部出力で観測できるようになれば良い. 提案する階層テスト容易化レジスタバインディングでは，上記の 2 点が実現可能となるようなレジスタバインディングを行う．本論文では，テスト環境生成成功演算の入力変数をテストパターン変数，テストパターン変数に対して割当てられたレジスタをテストパターンレジスタと呼ぶ．図 2 に示す例では，テスト環境生成成功演算*1 及び*3 の入力変数 a , b 及び d , e がテストパターン変数に対応し，これらの変数に割当てられるレジスタがテストパターンレジスタに対応する．階層テスト生成の結果，テストパターン変数には演算器のテストパターンが書き込まれることが保証されている．これらのテストパターン変数と，階層テスト不可能演算器の入力変数間でレジスタ(テストパターンレジスタ)を共有することで, 階層テスト不可能演算器の入力に対するテストパターンの伝搬を実現する. また, 階層テスト不可能演算器の出力変数と外部出力変数間でレジスタ(外部出力レジスタ)を共有することで，階層テスト不可能演算器の出力応答を外部出力で直接観測可能にする．図 3 に，図 2 の階層テスト容易化演算器バインディング結果に基づいた階層テスト容易化レジスタバインディング例を示す.

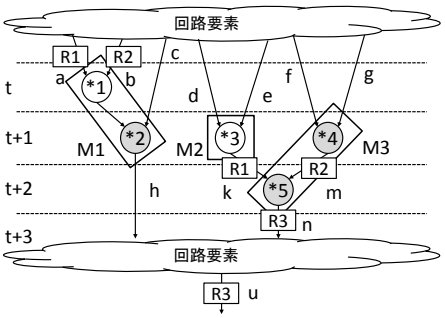

図 3. 階層テスト容易化レジスタバインディング

図 3 に示すレジスタバインディング例では，テストパターン変数 a , b と階層テスト不可能乗算器 M3 にマッピングされた*5 の入力変数 k , m 間でテストパターンレジスタ R1 及び R2 を共有している. これにより，乗算器 M1 をテストするためのテストパターンが R1 及び R2 に格納されると, そのタイミングで乗算器 M3 の入力に対しても, 乗算器テストパターンが伝搬する. 具体的には，時刻 t の間，乗算器 M1 をテストするためのテストパターンが M3 の入力に伝搬する．また，図 3 に示すバインディング例では，外部出力変数 u と*5 の出力変数 n 間で外部出力レジスタを共有しているため，乗算器 M3 の出力応答は外部出力 u で直接観測することが可能である．したがって，時刻 t において M3 に伝搬する乗算器テストパターンに対する出力応答は，時刻 t+1 において，外部出力 u で観測可能となる．これにより，階層テスト不可能乗算器 M3 が階層テスト可能演算器となる.

4. 実験結果

提案する階層テスト容易化バインディング法の有効性を示すために, 2 種類の例題回路に対して, 機能 RTL 回路情報を利用した階層テスト生成を行い，階層テスト容易化バインディングを行う場合と，行わない場合の故障検出率と回路面積を評価する実験を行った．また，階層テスト生成とゲートレベルの順序テスト生成の比較評価を行った．動作合成には内製の動作合成ツール PICTHY，論理合成には Synopsys 社の Design Compiler をそれぞれ用いた．また, 階層テストにおける故障シミュレーション，各機能要素に対する組合せテスト生成，及びゲートレベルの順序テスト生成には Synopsys 社の TetraMAX を用いた．回路合成結果を表 1 に示す．表 1 において「テスト容易化」は，提案する階層テスト容易化バインディングを行ったか否かを表す.「レジスタ数」は合成されたデータパス中のレジスタ数,「MUX 総入力数」はマルチプレクサの総入力数,「演算器数」は演算器数,「階層テスト可能演算

器数」は階層テスト可能演算器数,「回路面積」は論理合成後の回路面積をそれぞれ表す．本実験では階層テスト容易化バインディングを行うことで，example1, example2 共に全ての演算器を階層テスト可能演算器とすることに成功している．

表 2 に, 階層テスト生成及びゲートレベル ATPG のテスト生成結果を示す．階層テスト生成結果に着目すると, 階層テスト容易化バインディングを行う場合に, 故障検出率が平均して約 11.81%向上している．また example1，example2 共に階層テスト容易化バインディングを行う場合の階層テスト生成が本実験において最も高い故障検出率を達成している．階層テスト生成のテスト生成時間は，ゲートレベル ATPG と比較して，平均して約 41.53 倍高速になっている. このことから，機能 RTL 回路情報を利用した階層テスト生成は，ゲートレベルの順序テスト生成と比較して高速なテスト生成が可能であることがわかる．階層テスト容易化バインディングを行う場合の階層テスト生成において，テスト生成時間及び故障検出率の最良の結果が得られた．このことから，機能 RTL 回路情報を利用した階層テスト生成と，提案する階層テスト容易化バインディングを組合せたテスト設計は，テスト生成時間及び故障検出率の両面において有効であると考えられる．

5. おわりに

本論文では，機能 RTL 回路情報を用いた階層テスト生成結果を利用することで，階層テスト可能演算器数を増加させ，より高い故障検出率を達成するテスト容易化バインディング法を提案した．実験の結果，階層テスト容易化バインディングを行うことで，階層テストにおける故障検出率を，平均して約 11.81%向上させることに成功した．また，テスト生成時間をゲートレベルの順序テスト生成と比較して平均約 41.53 倍高速化することに成功した．今後の課題としては，大規模ベンチマーク回路での実験が挙げられる．

参考文献

[1] M. Schulz, E. Trischler, and T. Serfert, “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 Generation 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, and 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, and 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. 1446-1558, Dec. 2002.

[6] 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.

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

[8] L. Zhang, I. Gosh, and M. Hsiao, “Efficient Sequential ATPG for Functional RTL Circuits,” Proc. International Test Conference, pp. 290-298, Sep. 2003.

[9] 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), pp45-50, Nov. 2008.

[10] S. Bhatia, and N. K. Jha, “Integration of Hierarchical Test Generation with Behavioral Synthesis of Controller and Data Path Circuits,” IEEE trans. on Very Large Scale Integration Systems, Vol. 6, No. 4, Dec. 1998.

[11] T. Hosokawa, Hiroaki Fujiwara, R.Inoue, and Hideo Fujiwara, “A Binding Method for Hierarchical Testing Using Results of Test Environment Generation,” 12th IEEE Workshop on RTL and High Level Testing (WRTLT’11), pp. 16-22, Nov. 2011.

[12] V. Chaiyakul, D. D. Gajski, and L. Ramachandran, “High-Level Transformations for Minimizing Syntactic Variances,” proc. Design Automation Conference, pp. 413-428, Jun. 1993.

表 1. 回路合成結果

回路名	テスト容易化	レジスタ数	MUX総入力数	演算器数	階層テスト可能演算器数	回路面積
example1	なし	6	32	5	3	25401
	あり	7	31	5	5	25662
example2	なし	6	39	5	2	25736
	あり	6	37	5	5	25643

表 2. テスト生成結果

回路名	テスト容易化	階層テスト生成			ゲートレベルATPG		
		故障検出率(%)	テスト系列長	テスト生成時間(sec)	故障検出率(%)	テスト系列長	テスト生成時間(sec)
example1	なし	95.86	819	156	96.02	236	6364
	あり	99.95	819	147	97.12	327	4530
example2	なし	79.98	936	260	69.11	207	16384
	あり	99.51	936	195	97.07	438	6136