ISSN 2186-5647

2-1

テスト環境生成結果を用いた階層テストのためのスケジューリング法

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. まえがき

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)の大規模化や複雑化が進んでいる．これに伴い，LSIのテスト生成は益々困難な問題となってきている．特に順序回路に対するテスト生成は問題の解空間が大きく，現実的な時間で高い故障検出効率を達成することが困難である．現在，LSI 設計現場では，フルスキャン設計[1]と組合せ回路の自動テストパターン生成ツール(Automatic Test Pattern Generator : ATPG)を組み合せたテスト設計手法が広く取り入れられている．フルスキャン設計を施すことで，テスト生成時に順序回路を組合せ回路として扱うことが可能になる．したがって，フルスキャン設計された順序回路に対しては，組合せ回路のテスト生成法が適用可能となる．組合せ回路に関しては，効率的なテスト生成法が提案されており[2-4]，現実的な時間で高い故障検出効率を達成することができる．しかしながら，スキャン設計された順序回路は，面積，遅延時間，及び消費電力等のハードウェアオーバーヘッドが増大するという問題点がある．また，テスト時のスキャンシフト動作に伴うテスト実行時間の長大化も，テストコスト削減の観点から問題視されている．これらの問題点を解決するためには，スキャン設計を適用せずにテスト生成を実行する必要があり，その実現のためには，効果的な順序回路のテスト生成法が必要不可欠である．

効果的な順序回路のテスト生成法として，動作レベル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[5]．動作レベル回路情報を利用した階層テスト生成法は，動作合成を利用して設計される回路に対する効果的なテスト生成法である．動作レベル回路情報を利用した階層テスト生成では，動作記述をコントロールデータフローグラフ(Control Data Flow Graph : CDFG)と呼ばれるグラフで表現し，CDFG 内の各演算に対してテスト系列を生成する．階層テスト生成において，ある演算に対して生成されたテスト系列は，バインディングにおいて，その演算に対して割当てられた演算器をテストすることができる．本論文では，階層テスト生成により生成されたテスト系列でテストが可能な演算器を階層テスト可能演算器と呼び，テストが不可能な演算器を階層テスト不可能演算器と呼ぶ．動作レベル回路情報を利用した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，多くの演算器を階層テスト可能演算器とすることが重要である．

階層テスト可能演算器数の向上を指向したテスト容易化バインディング法が提案されている[6]．文献[6]で提案されたバインディング法は，ある演算器に対して生成されたテスト系列を利用して，他の階層テスト不可能演算器をテストできるように演算器及びレジスタバインディングを行うことで，階層テスト可能演算器数の向上を実現している．

動作合成におけるバインディングは，スケジューリングが適用された CDFG(Scheduled CDFG : SCDFG)に対して実行される処理であり，その解空間はスケジューリング結果に依存する．したがって，スケジューリングにおいて，文献[6]で提案されたバインディング法が，より効果的に働く SCDFG の生成を指向することで，階層テスト可能演算器数の向上が期待できる．

本論文では，文献[6]で提案されたバインディング法において，階層テスト可能演算器数を向上させるためのテスト容易化スケジューリング法を提案する．第 2 節で，動作レベル回路情報を利用した階層テスト生成法について述べ，第 3 節で，文献[6]で提案されたテスト容易化バインディング法について述べ，第 4 節で，提案するテスト容易化スケジューリング法について述べる．第 5 節で，実験結果を示し，最後に，第 6 節で，結論と今後の課題について述べる．

2. 動作レベル回路情報を利用した階層テスト生成法

本節では，動作レベル回路情報を利用した階層テスト生成法について述べる．動作レベル回路情報を利用した階層テスト生成は，データパス内の演算器に対するテスト系列を効果的に生成するテスト生成法である．動作レベル回路情報を利用した階層テスト生成では，まず，CDFG 内の各演算に対して，テスト環境と呼ばれる外部入力値の時系列を生成する．テスト環境は，外部入力から演算の入力に対して，任意の値を伝搬し，演算の任意の出力値を，外部出力まで伝搬するための

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

Jun NISHIMAKI, Toshinori HOSOKAWA and Hideo FUJIWARA

外部入力値の時系列である．階層テスト生成では，CDFG 内の全ての演算に対して，テスト環境の生成を試みるが，回路機能上，テスト環境が生成不可能な演算も存在する．テスト環境が生成された演算に対しては，テスト環境に対して，あらかじめ生成された演算器のテストパターンを代入することで，テスト系列が生成される．演算に対して生成されたテスト系列は，バインディングにおいて，その演算に対して割当てられた演算器をテストすることができる．

テスト環境が生成された演算に対しては，テスト系列が生成されているため，テスト環境が生成された演算に対して割当てられた演算器は，階層テスト可能演算器となる．一方，テスト環境が生成されなかった演算に対してのみ割当てられた演算器は，階層テスト不可能演算器となる．

3. 階層テスト容易化バインディング法

本節では，文献[6]で提案されたバインディング法について述べる．動作レベル回路情報を利用した階層テストにおいては，演算器数に対して，テスト環境が生成された演算の数が不足する場合，階層テスト不可能演算器が合成される．図 1 に演算器バインディングが適用された SCDFG の例を示す．図 1 に示す SCDFG において，白色の演算はテスト環境が生成された演算を表し，灰色の演算はテスト環境が生成されなかった演算を表す．乗算*1，加算+1 及び+2 はテスト環境が生成された演算に該当し，乗算*2 及び*3 はテスト環境が生成されなかった演算に該当する．また，変数 o は外部出力変数である．各演算を囲う枠は，演算に対して割当てられた演算器を表す．*1 及び*2 に対しては乗算器 M1 が割当てられており，*3 に対しては乗算器 M2 が割当てられている．加算+1 及び+2 に対しては，加算器 A1 及び A2 がそれぞれ割当てられている．

図 1 に示す例では，M1，A1 及び A2 は，テスト環境が生成された演算に対して割当てられているため，これらの演算器は階層テスト可能演算器である．一方，M2 は，テスト環境が生成されなかった乗算*3 のみが割当てられているため，階層テスト不可能演算器である．図 1 の SCDFG では，時刻 t+1 に 2 個の乗算がスケジュールされているため，2 個の乗算器が必要となる．しかしながら，テスト環境が生成された乗算の数は 1 個であるため，いかなる演算器バインディングを行ったとしても，階層テスト不可能乗算器が 1 個合成される．

文献[6]で提案されたバインディング法は，ある演算器に対して生成されたテスト系列で，階層テスト不可能演算器がテスト可能となるように，レジスタバインディングを行う．ある演算器に対して生成されたテスト系列を利用して，階層テスト不可能演算器をテストすることを，階層テスト不可能演算器の救済と呼ぶ．

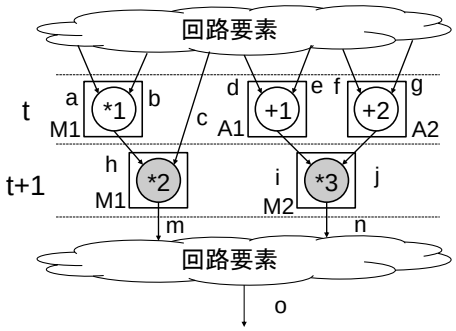

図 1. 演算器バインディング済み SCDFG 例

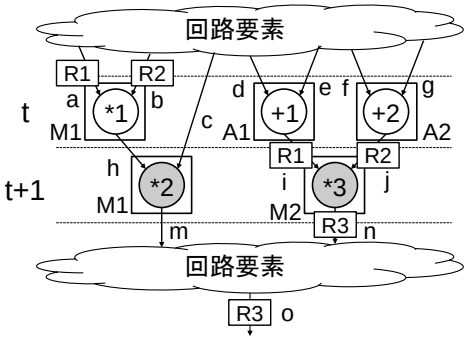

図 2. 階層テスト容易化バインディング例

図 2 に M2 の救済のためのレジスタバインディング例を示す．テスト環境が生成された演算の入力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．テストパターンレジスタには，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．文献[6]で提案されたバインディング法では，テストパターンレジスタを用いて，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．

図 2 に示す例では，テスト環境が生成された乗算*1 の入力変数 a 及び b に対して割当てられたレジスタ R1 及び R2 は乗算器のテストパターンレジスタである．時刻 t において，乗算器のテストパターンが R1 及び R2 に取り込まれる．文献[6]で提案されたバインディング法は，M2 の入力に乗算器のテストパターンを伝搬するために，R1 及び R2 を*3 の入力変数 i 及び j にそれぞれ割当てる．これにより，R1 及び R2 から M2 の入力に乗算器のテストパターンを伝搬するための経路が確保される．また，R1 及び R2 に乗算器のテストパターンが取り込まれる時刻 t には，M2 を用いた乗算がスケジュールされていないため，時刻 t において M2 はアイドル状態にある．したがって，時刻 t において，M2 に対していかなる動作を指定したとしても，機能動作に影響は及ばない．時刻 t において，M2 が R1 及び R2 からのデータを入力するように，RTL コントローラの制御信号を指定することで，M2 の入力に対する乗算器のテストパターンの伝搬が実現する．

また，文献[6]で提案されたバインディング法では，階層テスト不可能演算器のテスト応答の観測を，外部出力レジスタを用いて実現する．図 2 において，R3 は外部出力変数 o に対して割当てられているため，外部出力レジスタである．文献[6]で提案されたバインディング法は，M2 のテスト応答を外部出力で観測するために，*3 の出力変数 n に R3 を割当てる．さらに，時刻 t において，R3 が M2 からのデータを取り込むように，RTL コントローラに制御信号を設定することで，M2 のテスト応答を外部出力で直接観測できるようになる．

3. 階層テスト容易化スケジューリング法

本節では，提案する階層テスト容易化スケジューリング法について述べる．本論文では，文献[6]で提案されたバインディング法において，より多くの階層テスト不可能演算器を救済するための，2 種類のスケジューリング戦略を提案する．動作合成におけるスケジューリングは，合成される RTL 回路のレイテンシと，データパス内の演算器数が決定されるフェーズである．小面積かつ高速動作可能な回路を合成するためには，スケジューリングにおいて，レイテンシ及び演算器数を最適化することが重要である．実行サイクル数の最小性を保証しながら，演算器数の小さいスケジューリングを探索するアルゴリズムとして，フォースディレクティッドスケジューリング(Force Directed Scheduling : FDS)アルゴリズムが提案されている[7]．提案するスケジューリング法は，FDS を実行することで得られるレイテンシと演算器数を制約とする．与えられた制約を充足しながら，後述する 2 種類の戦略を用いて，より多くの階層テスト不可能演算器を救済するためのスケジューリングを実行する．

3. 1. 演算器のアイドル状態のための戦略

文献[6]で提案されたバインディング法において，階層テスト不可能演算器を救済するためには，テスト環境が生成された演算がスケジュールされた時刻において，階層テスト不可能演算器がアイドル状態であることが求められる．したがって，より多くの階層テスト不可能演算器を救済するためには，テスト環境が生成された演算がスケジュールされる時刻において，アイドル状態となる演算器数を増加させることが重要であると考えられる．これを実現するために，提案するスケジューリング法では，テスト環境が生成された演算がスケジュールされる時刻に，同種類の演算がスケジュールされることを可能な限り避けてスケジューリングを行う．これにより，テスト環境が生成された演算がスケジュールされる時刻において，アイドル状態の演算器数が増加し，結果として，救済される階層テスト不可能演算器数が増加すると考えられる．

3. 2. テストパターンレジスタのための戦略

文献[6]で提案されたバインディングでは，階層テスト不可能演算器が割当てられた演算，すなわち，テスト環境が生成されなかった演算の入力変数に対して，テストパターンレジスタを割当てることで，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．より多くの階層テスト不可能演算器を救済するためには，多くのテスト環境が生成されなかった演算の入力変数に対して，テストパターンレジスタが割当て可能であることが望ましい．これを実現するために，提案するスケジューリング法では，テスト環境が生成された演算の入力変数のライフタイムが，可能な限り短くなるようにスケジューリングを行う．これにより，より多くのテスト環境が生成されなかった演算の入力変数に対して，テストパターンレジスタが割当て可能となり，結果として，救済される階層テスト不可能演算器数が増加すると考えられる．

4. 実験結果

提案手法の有効性を示すために，3 種類の回路を用いた評価実験を行った．実験では，FDS により得られた SCDFG と，提案手法により得られた SCDFG に対して，文献[6]で提案されたバインディング法を実行し，回路合成結果と，動作レベル回路情報を利用した階層テストにおける故障検出率を評価した．論理合成には Synopsys 社の Design Compiler を用いた．データパスのビット幅は 32 ビットとして合成し，本実験では，データパスとコントローラに含まれる全故障を評価の対象とした．

表 1 に，動作記述に含まれる演算数とテスト環境生成結果を示す．表中の「Circuit」は回路名を表し，「#ADD」及び「#MUL」は，動作記述中に含まれる加算数及び乗算数をそれぞれ表す．また，「#ADD with TE」及び「#MUL with TE」は，テスト環境が生成された加算数及び乗算数をそれぞれ表す．ARF は，文献[8]に掲載された Auto Regressive Filter である．また，本実験では，大規模な CDFG を用いた評価を行うために，FIR+MPEG と DWT+MPEG の 2 種類の回路を設計した．FIR+MPEG は，文献[8]に掲載された Finite Impulse Response filter(FIR)と，MPEG Motion Vector(MPEG)の外部出力を加算した回路である．また，DWT+MPEG は，文献[8]に掲載された Discrete Wave Transformation(DWT)と MPEG の外部出力を加算した回路である．

表 2 に，回路合成結果及び動作レベル回路情報を利用した階層テストにおける故障検出率を示す．表中の「Latency」はレイテンシを表し，「#FU」はデータパス中の演算器数を表す．「#Testable FU」は階層テスト可能演算器数を表し，「FC」は動作レベル回路情報を用いた階層テストにおける故障検出率を表す．

「#REG」はデータパス中のレジスタ数を表し，「#MUX」はマルチプレクサの総入力数を表し，「Area」は論理合成後の回路面積を表す．

FDS では，全ての回路において，階層テスト不可能演器が合成されている．これに対して，提案手法は，全ての回路において，全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．また，故障検出率に着目すると，提案手法は，FDS と比較して最大で約 16%，平均して約 11%高い故障検出率を達成している．

提案手法は，全ての回路において，マルチプレクサの総入力数が FDS よりも大きい．また，FIR+MPEG 及び DWT+MPEG の 2 回路においては，データパス中のレジスタ数が FDS よりも大きい．しかしながら，論理合成後の回路面積に着目すると，面積増加は平均して約 2%に抑えることができている．

5. むすび

本論文では，文献[6]で提案されたバインディング法において，より多くの階層テスト不可能演算器を救済するためのテスト容易化スケジューリング法を提案した．動作合成ベンチマーク回路を用いた実験では，提案手法と文献[6]で提案されたバインディング法を組み合せることにより，動作レベル回路情報を利用した階層テストにおいて，故障検出率を最大で約 16%，平均して約 11%向上させることに成功した．今後の課題として，更なる大規模ベンチマーク回路を用いた評価実験が挙げられる．

参考文献

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] J. Nishimaki, T. Hosokawa, and H. Fujiwara, "Functional Unit and Register Binding Methods for Hierarchical Testability," 14th IEEE Workshop on RTL and High Level Testing (WRTLT'13), Nov. 2013.

[7] P. G. Paulin and J. P. Khight, "Force-directed scheduling for the behavioral synthesis of ASIC's," IEEE Trans. on Computer-Aided Design, Vol.8, No.6, pp661-679, Jun 1989.

[8] S. P. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, Low-Power High-Level Synthesis for Nanoscale CMOS Circuits, Springer, 2008.

表 1. 演算数及びテスト環境生成結果

Circuit	#ADD	#ADD with TE	#MUL	#MUL with TE
ARF	12	1	16	2
FIR+MPEG	32	2	22	1
DWT+MPEG	26	1	22	2

表 2. 回路合成結果及び故障検出率

Circuit	Latency	#FU	FDS					Proposed				
			#Testable FU	FC(%)	#REG	#MUX	Area	#Testable FU	FC(%)	#REG	#MUX	Area
ARF	10	6	4	84.37	10	1,851	35,491	6	99.96	10	2,082	35,900
FIR+MPEG	12	7	4	91.78	26	3,800	37,859	7	99.96	27	3,832	38,497
DWT+MPEG	13	7	5	91.47	22	3,130	35,617	7	99.96	24	3,384	36,816