ISSN 2186-5647

演算器入出力順序深度削減指向バインディング法

日大生産工（院）　○兒玉　雄佑　　日大生産工　細川　利典

1.はじめに

近年，半導体技術の進展に伴い，大規模集積回路(Large Scale Integrated circuits: LSI)の規模や複雑度が飛躍的に増大している．これにより，設計される LSI が急速に大規模化しており，設計が困難になっている．この問題を解決するためには，LSI の設計生産性を向上させる必要があり，抽象度の高い動作レベルでの設計が提案されてきた[1]．動作記述により設計された回路を，レジスタ転送レベル（Register Transfer Level: RTL）回路に合成する技術として動作合成[2]が存在する．近年，RTL 回路の面積や性能を最適化するための動作合成アルゴリズム[3][4][5][6]が数多く提案され，動作合成による LSI 設計が一部では行われている．

設計製造された LSI は，不良品であるか否かをテストし，良品であると判定された LSI のみを出荷しなければならない．テストを行うために入力する値のことをテストパターンと呼び，自動テストパターン生成ツール(Automatic Test Pattern Generator：ATPG)を用いて生成される．LSI は，論理ゲートを組み合わせて生成された組み合わせ回路と，値を保持するフリップフロップより構成される順序回路である．順序回路のテストパターン生成は，フィードバックループ[7]などの問題により困難とされている[8]．この，順序回路のテスト容易化設計として，広く用いられている技術がフルスキャン設計[9]である．フルスキャン設計を施すことにより，回路内のフリップフロップの値が外部より制御，観測が可能となり，高い故障検出効率を達成するテストパターンを，容易に生成することができる．しかしながら，LSI の大規模化に伴い，フルスキャン設計を施した回路に対するハードウェアオーバヘッドやテストコストが増加している[10]．また，暗号回路などの機密情報を扱う回路に対してスキャン設計を行った場合，スキャン FF やスキャンチェインを用いた攻撃が可能になる問題が指摘されている[11][12]．これにより，非スキャン設計に基づく，テスト容易性を考慮した動作合成が必要とされている[3]．

これまでの研究でテスト容易性を考慮した動作合成アルゴリズムとして，バインディング時に順序深度を削減することにより，テスト容易化を実現する方法が提案されている[13][14][15]．しかしながら，これらの手法は回路のデータパスのみに着目した手法となっており，データパスに対して高い故障検出効率を得るためのテスト生成を行うためには，データパスとコントローラがテスト時に分離されていることが前提となる．テスト時にデータパスとコントローラを分離するために必要な回路面積などのハードウェアオーバヘッドは，設計によっては大きく，実際分離困難な場合もある．それゆえ，コントローラを無視してデータパスのみに対するテスト容易化設計を行っても，データパスとコントローラが分離されない回路に対してテスト容易性の向上が得られないと考えられる．

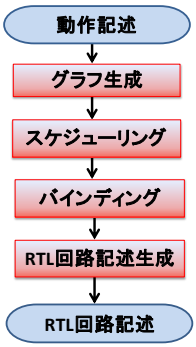

図 1.動作合成の流れ

本論文では，コントローラの動作を含めた回路全体のテスト容易化バインディング手法の前段階の研究として，データパス内の演算器に着目した演算器入出力順序深度という評価尺度を提案し，演算器入出力順序深度削減のためのバインディング法を提案する．

2.動作合成

動作合成とは，C 言語や SystemC[16]などのプログラミング言語でハードウェアの動作を表現した動作記述を，VHDL[17]や Verilog-HDL[18]といったハードウェア記述言語（Hardware Description Language:HDL）によって記述された RTL 回路記述に変換する技術である．

図 1 に動作合成の処理の流れを示す．動作合成の処理内容は大きく 4 つに分けることができる．各処理は，グラフ生成，スケジューリング，バインディング，RTL 回路記述生成である．グラフ生成フェーズでは，与えられた動作記述をグラフで表現する．グラフには，コントロールデータフローグラフ（Control Data Flow Graph:CDFG）[2]や割当て決定グラフ（Assignment Decision Diagram:ADD）[19]などが存在する．本論文では，グラフとして CDFG を用いて説明する．スケジューリングフェーズでは演算操作に時刻を割当てる．バインディングフェーズでは各変数や各演算にそれぞれレジスタや演算器を割当てる．RTL 回路記述生成フェーズでは，結線を実現したデータパスと制御信号を生成するコントローラを生成する．

A Binding Method to Reduce Sequential Depths of Resource Inputs and Outputs

Yusuke Kodama,Toshinori Hosokawa

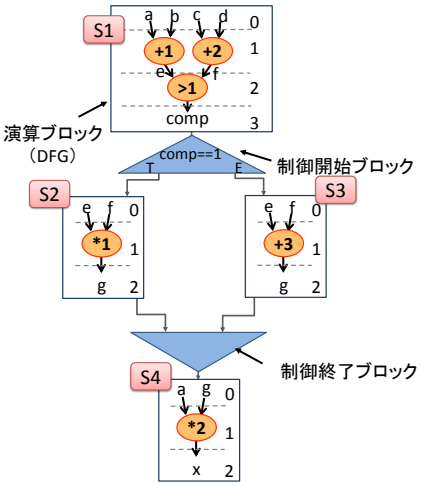

図 2.if 文の CDFG

2.1. グラフ生成

グラフ生成とは与えられた動作記述からグラフを生成することである．本研究では，if や while などの制御文のある C 言語を入力するため，CDFG を用いてグラフ生成を行う．CDFG の表現方法は，コントロールフローグラフ（Control Flow Graph: CFG）とデータフローグラフ（Data Flow Graph: DFG）を分けて表現する方法と，CFG と DFG を 1 つのグラフで表現する方法の 2 種類がある．CFG とは動作記述の制御文の構造を示すグラフであり，DFG とは演算操作の構造を示すグラフである．図 2 に CFG と DFG を 1 つのグラフで表した例を示す．

2.1.1.CFG

CFG は演算ブロック，制御開始，制御終了の 3 つの要素から構成される．演算ブロックとは DFG の部分であり，演算処理のみで構成される部分を表す．図 2 では　S1，S2，S3，S4 が該当する．制御開始とは，制御文の開始部分を表す．if 文や switch 文などの条件分岐や，for 文や while 文のようなループ処理の判定が該当する．図 2 では if 文の条件が真(Then)であった場合に T の S2 の処理が行われ，偽(Else)であった場合は E の S3 の処理が行われる．制御終了とは，条件分岐やループ処理の終了部分を表す．

2.1.2. DFG

DFG は LSI における演算処理を表現するグラフで，入力変数，出力変数，内部変数，演算操作から構成される．

2.2.スケジューリング

スケジューリングとは CFG，DFG に対して，各演算操作をどの時刻で実行するかを割当てる処理である．　CFG に対するスケジューリングでは，各演算ブロックの前後関係を決定する．CFG でのスケジューリングにおいて，同じ状態にスケジューリングされた演算ブロックは同時に実行されることはない．それゆえにこの後バインディング

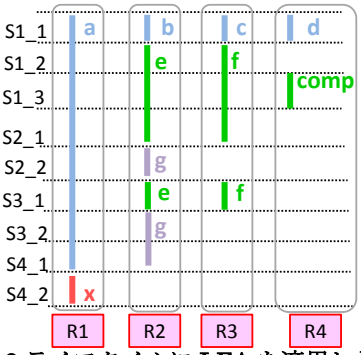

図 3.ライフタイムに LEA を適用した結果

フェーズにおいて，同じレジスタや演算器などのリソースを共有することができる．DFG に対するスケジューリングでは，時刻ごとに，どのような種類の演算器をどれだけ使用するか，また，どれだけの動作サイクルで実行するかを決める必要がある．スケジューリングされた DFG をスケジューリング済み DFG（Scheduled Data Flow Graph：SDFG）という．

2.3.バインディング

バインディングとは，SDFG に存在する演算操作および変数にそれぞれ演算器やレジスタを実際に割当てる処理である．演算器やレジスタを最適に割当てるためには，各演算操作や変数に対して実行時刻やライフタイム[2]を求める必要がある．ライフタイムとは，変数が値を保持し始める時刻から，値を保持しなくなる時刻までの時間である．実行時刻やライフタイムが重ならない演算操作や変数を，同じ演算器やレジスタで共有することができる．

2.3.1 面積最小化バインディング

図 2 の CDFG に対して，面積最小化バインディングを行う．まず，レジスタバインディングを考える．図 2 の CDFG に対してライフタイム抽出を行い，得られたライフタイムに対して面積最小化を目的としたレフトエッジアルゴリズム（Left-Edge Algorithm: LEA）[20]を適用した．その結果を図 3 に示す．図 3 より，変数 a と x はライフタイムが重ならないので，同一のレジスタ R1 に割当てることができる．同様に，変数 b, e, g はレジスタ R2 に，変数 c，f はレジスタ R3 に，変数 d，comp はレジスタ R4 に割当てることができる．

次に演算器バインディングを行う．図 4 に演算実行時刻に対して，共有可能な演算器を割当てた結果を示す．演算の場合は実行時刻が重ならない，同一種類の演算が演算器を共有することができる．図 4 の加算の場合は，演算操作+1 と演算操作+3 が実行時刻が異なるので，同一の演算器 Add1 を共有することができる．乗算と比較の場合も同様に，演算操作*1 と演算操作*2 は実行時刻が異なるので，同一の演算器 Mul1 を共有することができ，演算操作>1 は Grat1 に割当てられる．

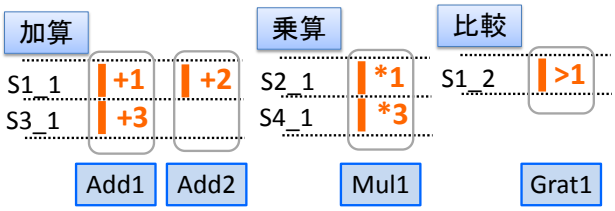

図 4. 演算実行時刻に対して
演算器を割当てた結果

2.4.RTL 回路記述生成

RTL 回路生成では，バインディングされた情報を基に，データパスとコントローラを生成する．データパスとは，SDFG とバインディング情報を基にレジスタと演算器を結線し，回路を実現したものである．コントローラとは，データパスを制御するもので，有限状態機械（Finite State Machine：FSM）として設計される．

3.データパスのテスト容易化バインディング法

本章ではデータパスのテスト容易化バインディング法として，演算器入出力順序深度の削減を行うアルゴリズムを提案，説明をおこなう．

3.1 演算器入出力順序深度

本節では，テスト容易化バインディングの評価尺度として，演算器入出力順序深度を提案する．テスト容易化バインディングの従来法として，順序深度削減バインディング[13]が存在する．この手法はレジスタバインディング時に，レジスタの可制御性および可観測性の強化を行い，外部から直接制御も観測も不可能なレジスタ数を削減することを試みる．問題点としては，順序深度のみ削減するという評価尺度では外部入力から外部出力までのレジスタの段数のみを考慮しており，どの演算器を経路としたレジスタの段数なのかが考慮されていない．それゆえ，順序深度計算時に経路として扱われなかった演算器が存在する可能性がある．その結果，その演算器に関してはテスタビリティが考慮されていないため，順序深度が浅い回路でも，テスト困難な演算器が存在している可能性ができる．その問題を解決するために，各演算器に着目した新たな評価尺度として，演算器入出力順序深度を提案する．演算器入力順序深度は，外部入力から演算器入力への最短経路で通過するレジスタの段数と定義し，演算器出力順序深度は，演算器出力から外部出力への最短経路で通過するレジスタの段数と定義する．

3.2.演算器入出力順序深度削減指向バインディング法

本節では，演算器入出力順序深度削減指向バインディング法を説明する．図 5 に演算器入出力順序深度削減指向バインディングアルゴリズムを示す．まず，可観測性強化レジスタバインディングとして，出力変数が割当てられた出力レジスタに対して，優先的に内部変数を割当てる．その際，ライフタイムの始まる時刻が早い（入力に近い）変数から割当てていく．また，割当てる変数の選択が起きた場合，できる限り多くの種類の演算の出力変数が割当てられるような選択を行う．

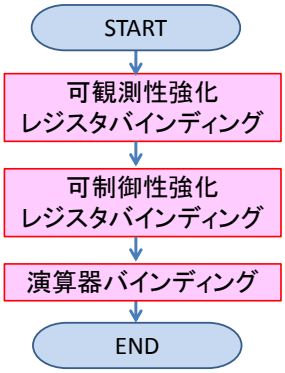

図 5. 演算器入出力順序深度削減指向
バインディングアルゴリズム

次に，出力レジスタに割当てられなかった内部変数に対して可制御性強化レジスタバインディングを行う．このフェーズでは，入力変数が割当てられた入力レジスタに，ライフタイムの終了時刻が遅い（出力に近い）内部変数を割当てる．

レジスタバインディングの最後に，出力レジスタと入力レジスタが共有できるか否かを判定し，共有ができれば，レジスタの共有を行う．また，入力レジスタと出力レジスタに割当てることのできない内部変数が存在した場合，内部変数に新たなレジスタを割当てる．

アルゴリズムの最後に，レジスタバインディングを行った結果を用いて，演算器バインディングを行う．演算器バインディングでは演算器出力（入力）順序深度が浅い演算と，演算器出力（入力）順序深度が深い演算を同じ演算器に割当てる．このような割当てを行うことにより，演算器出力順序深度が深い演算器の個数を削減することができる．

4. 実験結果

本論文では，第 3 章で提案した演算器入出力順序深度削減指向バインディング法と，テスタビリティを考慮しないバインディングアルゴリズムとして，LEA を元にした面積最小化バインディング法を比較する．テスト容易性を比較するために，各手法を適用した RTL 回路を，論理合成ツール design vision を用いて論理合成を行い，単一縮退故障を対象に対してテスト生成を行った．実験を行う回路は，文献[21]に掲載されている回路を参考に Verilog-HDL で実装した．ATPG ツールは Synopsys 社の TetraMax を用いた．なお，スケジューリングアルゴリズムは ALAP を使用した．実験結果を表 1 に示す．なお，TC は故障検出効率（Test Coverage），FC は故障検出率（Fault

表 1. 実験結果

回路名	バインディング手法	TC(%)	FC(%)	回路面積	CPU時間（sec）
intermediate	演算器順序深度削減	98.66	98.09	3414	422.83
	面積最小化	99.07	98.82	3220	541.48

Coverage）を表している．

5. おわりに

今回の実験の結果より，演算器順序深度削減指向バインディングと，面積最小化バインディングには故障検出効率の大きな差は出なかった．この理由として，実験に使用した回路の規模が小さく，レジスタや演算器の割当てに大きな差が出なかったことが考えられる．しかしながら，テスト生成時間である CPU 時間は改良されていることから，テスト容易性の向上の効果はあったと考える．

今後の課題として，他の回路での実験を行うこと，他のテスト容易性向上の手法と今回の提案手法を組合わせることなどが挙げられる．

参考文献

[1].藤原 秀雄, “ディジタルシステムの設計とテスト”, 工学図書株式会社, 2004.

[2].Daniel D, Gajski, “High-Level Synthesis”, Kluwer Academic Pub, 1992.

[3].M.C.McFarland, A.C.Parker, R.Camposano, “The high-level synthesis of digital systems”, Proc. IEEE, pp.301-318, 1990.

[4].R.Camposano and W.H.Wolf, “High-Level VLSI Synthesis”, Kluwer Academic Publishers, 1991.

[5].Z. Peng and K. Kuchcinski, “Automated transformation of algorithms into register-transfer level implementations”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp.150-166, 1994.

[6].P. G. Paulin and J. P. Knight, “Forced-directed scheduling for the behavioral synthesis of ASIC's”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp.661-678, June 1989.

[7].M.Abramovici, M.A.Breuer, and. D.Friedman, “Digital systems testing and testable design”, IEEE Pres,1995.

[8].V.Femandez and P.Shchez,“Partial Scan High-Level Synthesis”,1996 European Design and Test Conference (ED&TC '96),1996.

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

[10].Y. Sato, T. Ikeda, M. Nakao, and T. Nagumo, “A bist approach for very deep sub-micron (vdsm) defect,” Proc. International Test Conference , pp. 283-291, 2000.

[11].Bo Yang, Kaijie Wu, Ramesh Karri, "Scan Based-Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard, " in Proc. International Test Conference 2004, pp.339-344,Oct. 2004.

[12].Bo Yang, Kai j ie Wu, Ramesh Kar ri , "Secure Scan: A Design-for-Test Archi tecture for Crypto Chips, " in Proc. Annual ACM IEEE Design Automation Conference, pp.135-140, June 2005.

[13].Tien-Chien Lee, Wayne H. Wolf, Niraj K. Jha, John M. Acken, “Behav-ioral Synthesis for Easy Testability in Data Path Allocation”, ICCD, pp.29-32, 1992.

[14] .T. Yang and Z. Peng, “An efficient algorithm to integrate scheduling and allocation high-level test synthesis”, In Proceedings of Design, Auto-mation and Test in Europe (DATE-98) Paris, France, pp.74-81, February 1998.

[15].Benmao Cheng, Hong Wang, Shiyuan Yang, Daoheng Niu and Yang Jin, “Register Allocation Algorithm for High-Level Circuit Synthesis for Improved Testability”, TSINGHUA SCIENCE AND TECHNOLOGY, pp.836-842 Volume 13, Number 6, December 2008.

[16].IEEE Standard 1666, “Open SystemC Language Reference Manual”, IEEE, 2005.

[17].IEEE Standard 1076, “VHDL Language Reference Manual”, IEEE, 1987.

[18].IEEE Standard 1076, “Verilog Language Reference Manual”, IEEE, 2001.

[19].Viraphol Chaiyakul, Daniel D.Gajski, “Assignment Decision Diagram for High-Level Synthesis”, Dept. of Information and Computer Science, pp.92-103, December 12, 1992.

[20].F.J.Kurdahi and A.C.Parker: “REAL: A program for register allocation”, In Proc. IEEE Design Automation Conf., pp.210-215, Miami Beach, June 1987.

[21].Mike Tien-Chien Lee “High-Level Test Synthesis of Digital VLSI Circuits”,Artech House Publishers,1997