テスト容易化動作合成法

日大生産工(院)　○長 孝昭　　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 Integration: 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，いくつかの問題が発生している．そのひとつとして，LSI 設計の生産性に関するものが存在する．現在，レジスタ転送レベル（Register Transfer Level: RTL）での LSI 設計が主流であるが，LSI の大規模化により，設計が非常に困難になってきている．LSI 設計が困難になると，LSI の設計生産性の低下や，設計コストの増加につながってしまう．これを回避するため，より高位レベルでの設計が提案されてきた．現在，RTL より高位レベルである動作レベルで設計された回路を RTL に変換する技術である動作合成[1]が注目を浴びている．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[2]が数多く提案され，動作合成による LSI 設計が実用化されるようになった．

LSI 設計に関する問題に解決策が見出されている一方，LSI テストに関する問題も存在する．設計，製造された LSI は，不良品であるか否かのテストを行い，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．現在，LSI のテスト容易化設計では，高い故障検出効率を得るテストパターンを容易に生成することができるフルスキャン設計[3]が主流となっている．しかしながら，フルスキャン設計を施した回路は，スキャンチェインを通して LSI の記憶素子(レジスタ)に直接アクセス可能になってしまうため，外部に漏れてはいけない値が可観測になってしまう等，セキュリティ面に問題が生じてしまう[4]．つまり，RSA[5]や，DES[6]といった暗号回路等へフルスキャン設計の適用は望ましいとはいえない．よって，フルスキャン設計を施さず，テスタビリティを向上させる手法が必要とされている．通常，スキャン設計を施していない LSI に対し，自動テストパターン生成（Automatic Test Pattern Generator: ATPG）を用いてテストパターンを生成し，高い故障検出率を得ることは困難である．フルスキャン設計を施さず，テスタビリティを向上させる方法には，動作合成時に，テスト容易化を考慮に入れたアルゴリズムとして，レジスタの段数（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手法が提案されている[7]．本稿では，[7]の手法をベースに，さらにテスタビリティを強化する手法を提案する．

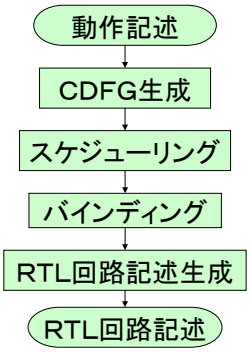

図 2-1.動作合成の流れ

2. 動作合成

動作合成とは，C 言語や SystemC[8]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[9]や Verilog-HDL[10]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ものであり，その処理内容は大きくわけると 4 四つのフェーズにわかれる[1]．各フェーズはそれぞれ，CDFG（Control Data Flow Graph）生成，スケジューリング，バインディング，RTL 回路記述生成となっており（図 2-1），入力として動作記述ファイルおよび使用可能な演算器やレジスタ（リソース）の数を渡す必要がある．CDFG 生成というフェーズでは，与えられた動作記述を変換し，グラフで表現する．グラフにおける頂点は演算を、辺は変数を表す．スケジューリングというフェーズでは，各演算の依存関係を保ちつつ，リソース制約を条件の下，各時刻に演算操作を割り当てる．また，リソースの制約を与えられていない場合，スケジューリングのフェーズで，CDFG に現れる演算を実現するためのリソース数を決める必要がある．バインディングというフェーズでは，スケジューリングされた DFG (Scheduled Data Flow Graph: SDFG) を元に，各演算操作や変数に具体的な演算器やレジスタを割り当てる．RTL 回路記述生成では，結線を実現した RTL データパスと制御信号を生成するコントローラ（Finite State Machine: FSM）を生成する．なお，本稿では，制御文のない動作記述ファイルを入力とするので，CDFG ではなく DFG を取り扱う．

Behavioral Test Synthesis Method

Takaaki CHO, Toshinori HOSOKAWA

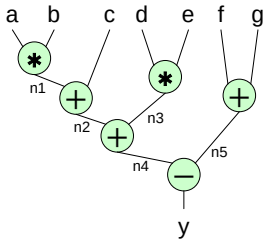

図 2.1-1. DFG

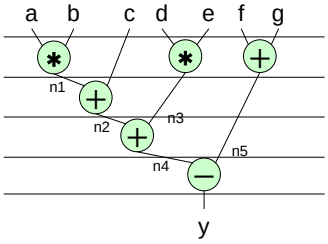

図 2.1-2. ASAP

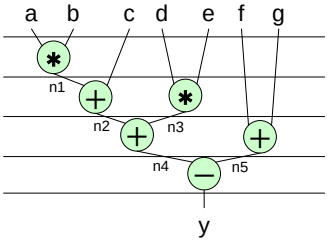

図 2.1-3. ALAP

2.1. スケジューリング

スケジューリングには，可能な限り早い時刻に演算子を割り当てるスケジューリング手法（As Soon As Possible: ASAP）[11]，可能な限り遅い時刻に演算子を割り当てるスケジューリング手法（As Late As Possible: ALAP）[11]が最も基本的な手法として存在している．ASAP および ALAP では，演算子の実行時刻しか考慮していないため，無駄な面積オーバーヘッドが存在している可能性がある．

図 2.1-1 に示す DFG を例にスケジューリング結果を説明する．この DFG に対し ASAP を行った場合，必要リソース数はそれぞれ，乗算器 2，加算器 1，減算器 1，レジスタ 7 となる（図 2.1-2）．ALAP を行った場合，必要リソース数はそれぞれ，乗算器 1，加算器 2，減算器 1，レジスタ 7 となる（図 2.1-3）．しかしながら，この DFG は乗算器 1，加算器 1，減算器 1，レジスタ 7 で実現可能である（図 2.1-4）．このような効率のよい SDFG を得るには，List スケジューリング[11]や FDS（Force Directed Scheduling）[11]といったヒューリスティックなスケジューリングを適用する必要がある．本稿で適用するスケジューリングでは，実行にかかるサイクル数（レイテンシ）を最小に保ちつつ，リソースの最小化を実現するため，FDS を用いる．

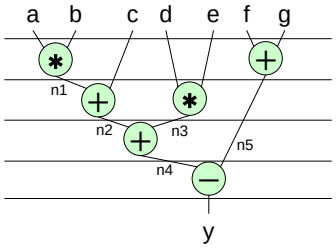

図 2.1-4. FDS

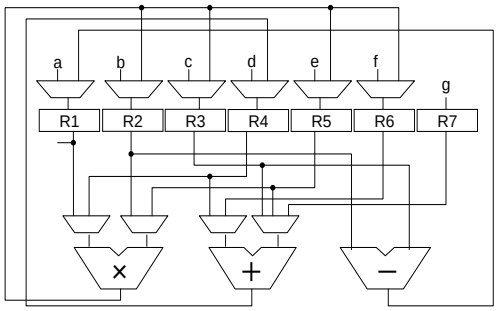

図 2.2-1. ランダムバインディング後のデータパス

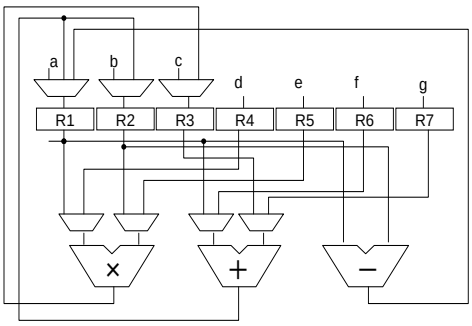

図 2.2-2. 順序深度削減指向バインディング後のデータパス

2.2. バインディング

バインディングは，実際に使用するリソースへ割り当てるフェーズであるため，生成される RTL 回路の面積やテスタビリティに影響を与えやすい．よって，手法次第で順序深度やマルチプレクサ数などを削減し，面積の減少やテスタビリティの向上を図ることが可能である．本稿では，テスタビリティの向上を目的としているため，順序深度に着目し，バインディングを行う必要がある．以下に順序深度削減を指向したバインディングについて述べる．

順序深度の削減を実現するには，入力変数が割り当てられたレジスタ（入力レジスタ）や，出力変数が割り当てられたレジスタ（出力レジスタ）に，可能な限り内部変数を割り当て，入力レジスタから出力レジスタまでの順序深度を削減するという指針でバインディングを行う必要がある[7]．なお，ここで扱う順序深度とは，入力レジスタが出力されるまでに最低限通らなければいけないレジスタの段数のことを言う．図 2.1-1 の DFG に対し，何も考慮せずに

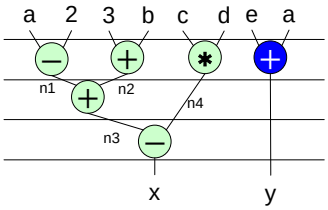

図 3.1-1. SDFG

ランダムでバインディングを行った場合(図 2.2-1)，レジスタに割り当てられた変数はそれぞれ R1＝｛a, y｝，R2＝｛b, n4｝，R3＝｛c, n5｝，R4＝｛d, n2｝，R5＝｛e, n3｝，R6＝｛f, n1｝，R7＝｛g｝となり，順序深度は R1：0，R2：1，R3：1，R4：2，R5：2，R6：2，R7：2 となる．また，順序深度削減指向でバインディング[7]を行った場合（図 2.2-2），レジスタに割り当てられた変数はそれぞれ R1＝｛a, n1, n2, n4, y｝，R2＝｛b, n5｝，R3＝｛c, n3｝，R4＝｛d｝，R5＝｛e｝，R6＝｛f｝，R7＝｛g｝となり，順序深度は R1：0，R2：1，R3：1，R4：1，R5：1，R6：1，R7：1 となる．結果，この例では，順序深度削減指向バインディングを適用したことにより，順序深度が 2 から 1 に削減できた．このように，バインディングによって順序深度を削減することが可能であり，その結果テスタビリティが向上することが報告されている[7].

3.　テスト容易化動作合成法

本稿では，順序深度削減指向バインディング[7]をベースに，さらなるテスタビリティ向上を目的とする．今回は 2 点に着目した改善を行う．戦略 1 として，テスタビリティの評価尺度の改善を行う．戦略 2 としてテスタビリティ強化対象フェーズの拡大を行う．

3.1.　戦略 1

従来は順序深度にのみ着目し，それを削減することをテスタビリティ向上のための指針としてバインディングを行うものであった[7]．しかしながら，順序深度のみ削減するという評価尺度では入力レジスタから出力レジスタまでの経路を考慮していない．そこで，各演算器に着目し，入力レジスタから演算器までのレジスタの段数と演算器から出力までのレジスタの段数の合計値（以下，演算器順序深度）を新たな評価尺度として提案する．

この評価尺度の効果を，図 3.1-1 に示す SDFG を例に説明する．なお，図 3.1-1 に示す SDFG において，e，a を入力とし y を出力とする加算は，扱うビット幅が異なるため，共有はできないものとする．順序深度のみを考慮したバインディング結果，レジスタに割り当てられた変数はそれぞれ R1＝｛a, y｝，R2＝｛e, x｝，R3＝｛d, n2, n3｝，R4＝｛c, n1｝，R5＝｛b, n4｝となり，順序深度および演算器順序深度は R1：0，R2：0，R3：1，R4：2，R5：1，加算器：2, 減算器;1, 乗算器:2, 除算器:1 となる(図 3.1-2)．同様の SDFG に演算器順序深度削減バインディングを適用する．方針として，演算器順序深度の削減を優先し，その後で順序深度の削減を考慮する．

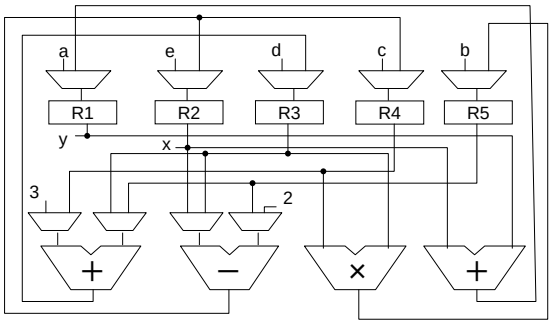

図 3.1-2. 順序深度削減指向

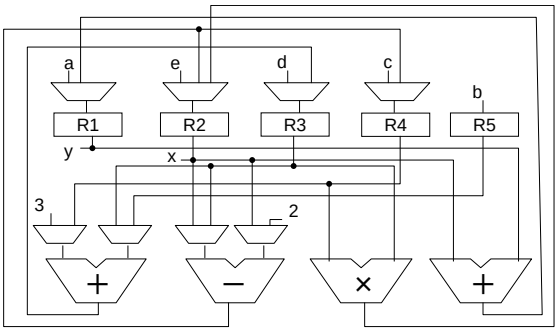

図 3.1-3.　演算器順序深度削減指向

削減する演算器順序深度の優先度は，構造が複雑な演算器を優先とする．結果は，レジスタに割り当てられた変数がそれぞれ R1＝{a, y}, R2＝{e, x, n4}, R3＝｛d, n2, n3｝，R4＝｛c, n1｝，R5＝｛b｝となり，順序深度および演算器順序深度は R1：0，R2：0，R3：1，R4：2，R5：2，加算器：2，減算器；1，乗算器：1，除算器：1 となる（図 3.1-3）．この例では，乗算器の演算器順序深度を削減することができたため，乗算器の故障が検出しやすくなり，テスタビリティの向上につながると考えられる．

3.2.　戦略 2

従来はバインディングのフェーズのみに着目し，テスタビリティ向上を図っていた．しかしながら，バインディングのみでは，テスタビリティを向上させにくい回路も存在する．そこで，スケジューリングのフェーズでもテスタビリティ向上を図ることにする．手法としては，レイテンシ最小およびリソース最小を保った状態で，現在と異なるスケジューリング結果が存在することがわかっている場合，バインディング後，再スケジューリングを行うものである．再スケジューリング前と後とのバインディング結果を比較し，演算器順序深度の少ないほうの結果を選択する．この機能の追加による効果を，戦略 1 と同様の図 3.1-1 に示す SDFG を例に説明する．

この SDFG では，レイテンシ最小およびリソース最小を保った状態で，再スケジューリングを行うことが可能である．制約を保ったまま，再スケジューリングを行った SDFG（図 3.2-1）に対し，演算子順序深度削減バインディングを行う（図 3.2-2）．その結果，レジスタに割り当てられた変数はそれぞれ R1＝｛d, n4, x｝，R2＝｛b, n2, y｝，R3＝｛c, n1｝，R4＝｛a, n3｝，R5＝｛e｝となり，順序深度および演算

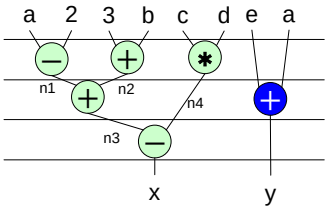

図 3.2-1. 再 SDFG

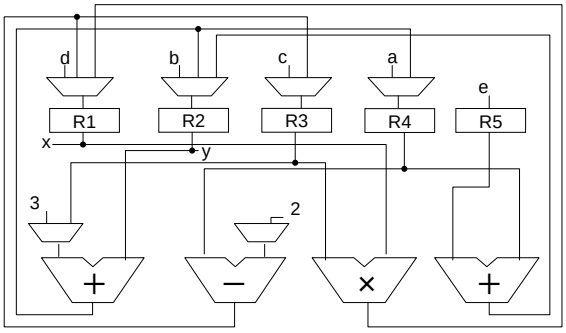

図 3.2-2. 再スケジューリング適用
演算器順序深度削減指向バインディング後の
データパス

器順序深度は R1 : 0，R2 : 0，R3 : 1，R4 : 1， R5 : 1，加算器 : 1，減算器 ; 1，乗算器 : 1，除算器 : 1 となる．この例では，順序深度，演算器順序深度ともに削減できたため，確かなテスタビリティの向上が見込まれる．

4. 実験結果

本稿の 3 章で記載した RTL 回路に対し，8bit, 32it のビット幅でそれぞれ TetraMAX を用い実験を行った（表 4-1）．なお，今回使用したモジュールを単体でテストした結果，故障検出率は 100%であった．

戦略 2 では，従来法である順序深度削減指向バインディングも，戦略 1 である演算器順序深度削減指向バインディングも上回る結果となった．戦略 1 は，従来法と比較し，ビット幅が大きい 32bit 回路では，故障検出効率もテスト生成時間も向上したが，ビット幅の小さい 8bit 回路で故障検出効率は下がってしまった．8bit 回路で検出できなかった故障は，従来法では乗算器，戦略 1 では加算器の内部に存在するものだった．戦略 1 を適用することにより，演算器順序深度の削減ができた乗算器に対しては，故障検出率が向上したのだが，同じ演算器順序深度である加算器には効果が見られなかった．これは，バインディングの方法で回路構造が異なるので，同じ演算器順序深度のモジュールであっても，故障検出率が異なってしまったのだと考えられる．

5. おわりに

本稿では，テスト容易化動作合成法として，バインディング時における新たな評価尺度の提案および，スケジューリングのフェーズへのテスト容易化対象フェーズ拡大を提案した．今回は対象とした回路数が少なく，規模も小さいものであったので，効果の傾向をはっきりと解析できなかった．今後の予定として，対象回路の増加および拡大を実現し，回路による効果の違いを解析し，よりテスト容易化が可能なよう研究を進める．

参考文献

[1] Daniel D. Gajski：High-Level Synthesis, Kluwer Academic Pub, 1992
[2] M.C.McFarland,, A.C.Parker, R.Camposano: The high-level synthesis of digital systems, Proc. IEEE, 301-318, 1990
[3] H.Fujiwara: Logic Testing and Design for Testability, The MIT Press, 1985
[4] Bo Yang, Kaijie Wu, Ramesh Karri, “Secure Scan: A Design-for-Test Architecture for Crypto Chips”, Annual ACM IEEE Design Automation Conference, pp.339-344, 2004
[5] A Method for Obtaining Digital Signature and Public-key Cryptsystems; R.L.Rivest,A.Shamir,and L.Adelman; MIT Laboratory for Computer Science; Thechnical Memo LCS/TM82; April 4,1977
[6] National Bureau of Standards, "Data Encryption Standard", Federal Information Processing Standards Publication 46, 1977.
[7] Tien-Chien Lee, Wayne H. Wolf, Niraj K. Jha, John M. Acken: Behavioral Synthesis for Easy Testability in Data Path Scheduling, ICCD, 1992
[8] 並木 秀明,後閑 哲也,片岡 忠士: SystemC によるシステムデザイン入門，株式会社技術評論社, 2005
[9] 中 幸政: VHDL と CPLD によるロジック設計入門, CQ 出版株式会社, 2005
[10] 並木 秀明,前田 智美,宮尾 正大: ディジタル回路と Verilog-HDL，株式会社技術評論社, 1996
[11] Giovanni De Micheli, SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS, McGraw-Hill,inc, 1994

表 4-1. 実験結果

	ビット幅	打ち切り故障	テスト不能故障	故障検出率	故障検出効率	テスト生成時間(秒)
従来法	8	3	0	99.91	99.91	219.00
	32	90	0	99.65	99.65	3367.02
戦略1	8	6	0	99.81	99.81	198.85
	32	3	0	99.99	99.99	1859.07
戦略2	8	0	16	99.50	100	6.34
	32	0	64	99.75	100	515.45