ISSN 2186-5647

2-62

演算器入出力順序深度削減のための テスト容易化バインディングアルゴリズムの評価

日大生産工　○林 愛美　　日大生産工(院)　西間木 淳
日大生産工(院)　増田 哲也　　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)の大規模化や複雑化が急速に進展している．これにより，LSIの設計コストが増大し，設計が困難になってきている．従来，LSI設計において，ハードウェア記述言語(Hardware Description Language : HDL)でレジスタ転送レベル(Register Transfer Level : RTL)の回路を記述し，論理合成ツールを用いることでネットリストを生成する方法が主流であった．しかしながら，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，LSIの複雑度が増加するにつれて設計が困難となる[1]．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いる[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[1][2]が存在する．動作記述はRTL記述に比べ記述量が少ないため，設計生産性に優れる．その反面，レジスタや演算器の割当てなどを動作合成ツールが決定するため，ツールの性能が合成後の回路の性能に大きく影響する[1].

また，設計された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のみを出荷する必要がある．しかしながら， LSIの大規模化に伴い，高品質かつ低コストなテストが困難となってきている．これにより，テスト容易性を考慮した動作合成が提案されている[1].

これまでに提案された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ある．文献[3]では，任意の外部入力レジスタと外部出力レジスタペア間の順序深度[4]を最小化している．しかしながら，外部入力レジスタと外部出力レジスタ間に存在する演算器が，最小化した順序深度を持つ経路上に存在するとは限らない．そのような演算器の故障に対するテスト生成は，困難である可能性がある．

本論文では，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演算器の入出力の順序深度を削減する方法を新たに提案し，その優位性を実験により評価する．

本論文の構成は以下の通りである．第2章で，順序深度削減に基づくテスト容易化バインディング法について述べ，第3章で，提案するバインディング時に演算器入出力順序深度を削減する方法について述べ，第4章で，実験結果を示し，最後に第5章でまとめと今後の課題について述べる．

2. 順序深度削減に基づくテスト容易化バインディング法

本章では、従来法である順序深度削減に基づくテスト容易化バインディング法について説明する．

2.1. バインディング

一般に動作合成は，グラフ生成，スケジューリング，バインディング，RTL 回路記述生成の 4 つのフェーズによって構成される．バインディングはスケジューリング済み DFG（Scheduled Data Flow Graph : SDFG）に対して実行される．図 1 に SDFG の例を示す．バインディングは SDFG 上に存在する演算操作および変数に演算器やレジスタをそれぞれ割当てる処理である．演算器やレジスタの共有を考慮する場合，各演算操作の実行時間や各変数のライフタイム[1]を求める必要がある．実行時刻が衝突しない演算操作や変数を，それぞれ同じ

An Evaluation of a Binding Algorithm for Testability to Reduce Sequential Depth for Inputs and Outputs of Operational Units

Manami HAYASHI,　Jun NISHIMAKI,
Tetsuya MASUDA and Toshinori HOSOKAWA

演算器やレジスタで共有することが可能である．

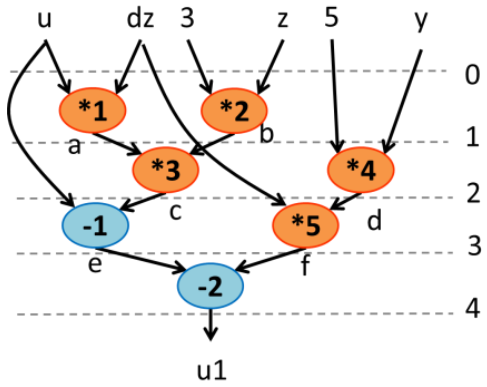

図 1. スケジューリング済み DFG

2.2. 順序深度

外部入力レジスタと外部出力レジスタペア間の任意の経路上に存在するレジスタ数の最小値を外部入力レジスタと外部出力レジスタ間の順序深度という．外部入力変数は{u，dz，z，y}，外部出力変数は{u1}である．図 1 の例では，外部入力変数{u，dz，z，y}を割当てたレジスタを外部入力レジスタという．また，外部出力変数{u1}を割当てたレジスタを外部出力レジスタという．回路にあるすべての外部入力レジスタと外務出力レジスタ間の順序深度を求め，最大のものをデータパスの順序深度とする．このデータパスの順序深度が小さいほど，テスト容易であるといえる．

2.3. 順序深度削減バインディング

順序深度削減バインディングは，各中間変数について外部入力レジスタまたは外部出力レジスタを可能な限り割当てるものである．その結果を図 2 に示す．図 2 において，REG1)に{a, c, f, u1}, REG2 に{b, z}, REG3 に{d, y}，REG4 に{e，u}，REG5 に{dz}の変数が割当てられている．演算において，減算器(SUB1)に{-1，-2}，乗算器(MUL1)に{*1，*4，*5}，乗算器(MUL2)に{*2，*3}が割当てられている．順序深度を計算した結果を表 1 に示す．
表1の結果より，データパスの順序深度が最小に削減されていることがわかる．しかしながら，順序深度削減バインディングは，レジスタのみに着目したバインディングであるため，演算器については考慮されていない．

3. 演算器の入出力順序深度削減に基づくテスト容易化バインディング法

本章では、提案手法である演算器入出力順序深度削減に基づくテスト容易化バインディング法について説明する．

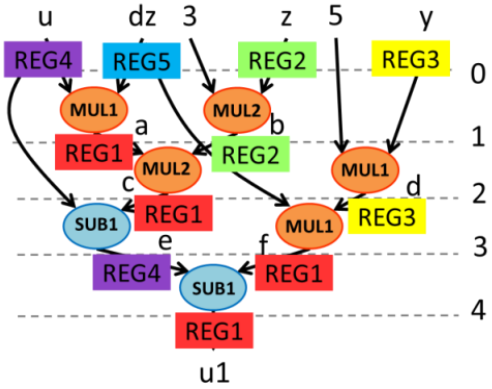

図 2. 順序深度削減バインディング

表 1. 順序深度

（外部入力レジスタ，外部出力レジスタ）ペア	順序深度
（R2，R1）	0
（R3，R1）	0
（R4，R1）	0
（R5，R1）	0

3.1. 演算器入力順序深度

演算器入力順序深度とは，外部入力レジスタと演算器の入力ペア間の任意の経路に存在するレジスタ数の最小値と定義する．図 3 において，演算器 X の左入力順序深度は 0，演算器 B の左入力順序深度は 1 である．

3.2. 演算器出力順序深度

演算器出力順序深度とは，演算器の出力と任意の外部出力レジスタペア間の任意の経路に存在するレジスタ数の最小値と定義する．図 3 において，演算器 B の出力順序深度は 0，演算器 X の出力順序深度は 1 である．

3.3. 順序深度削減バインディングの問題点

2 章の最後にも述べたとおり，順序深度削減バインディングはレジスタのみに着目したものであるため，演算器入出力順序深度は考慮されていない．また，演算器入出力順序深度を計算した結果を表 2 に示す．表 2 より，演算器の入出力順序深度のすべてが最小になっていないことがわかる．テスト容易性の向上を目指すため，演算器入出力順序深度削減指向バインディング

を提案する．

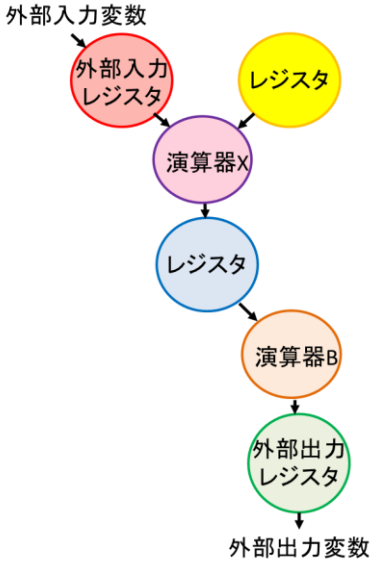

図 3. 演算器 B と X の演算器入出力順序深度

表 2. 演算器入出力順序深度

演算器	左入力	右入力	出力
MUL1	0	0	0
MUL2	1	0	0
SUB1	0	1	0

3.4. 演算器の入出力順序深度削減指向バインディング

演算器入出力順序深度削減指向バインディングを図 1 の SDFG を使用して説明する．

まず，外部入力変数と外部出力変数にレジスタを割当て，外部入力レジスタ(REG1~REG4)と外部出力レジスタ(REG5)を決定する．

次に，演算器バインディングを考える．各外部入力から各演算の入力までの経路間の辺の最大値と各演算の出力から外部出力変数までの経路間の辺の最小値を計算する．辺の数が 1 になるものは，外部入力変数もしくは外部出力変数が演算の入力もしくは出力であることを表す．ただし，入力に定数が与えられている場合，辺の数を∞と定義する．図 1 では，乗算{*1，*2，*3, *4*5}は一意に乗算器の割当てができないため, 辺の数からバインディングを行う. 例えば，乗算{*3}の左入力は，変数{a}を通り外部入力変数{u}に到達するため，辺の数は{a，u}の 2 である．各乗算の入力から外部入力変数へ到達する経路の最小の辺の数，および各演算の出力から外部出力変数へ到達する経路の最小の辺の数を表 3 に示す．

表 3. 乗算の入出力が外部入出力へ到達するまでの辺の数

	左入力	右入力	出力
*1	1	1	4
*2	∞	1	4
*3	2	∞	3
*4	∞	1	3
*5	1	∞	2

まず，入力順序深度削減を考える．演算{*1，*2}と演算{*3，*4}はそれぞれ同じ時刻にスケジュールされているため，同じ演算器に割当てができない．演算{*1，*2，*3，*4，*5}の中で左入力も右入力も辺の数が 1 である演算{*1}は演算器入出力順序深度を削減するために有効である．演算の共有化を行うと，辺の数は小さい方に合わせることができる．入力が 2 である演算{*3}と演算{*1}を同じ演算器に割当てる．また，演算{*2}，{*4}，{*5}を共有化することで，辺の数はすべて 1 とすることができる．そのため，乗算器(MUL1)に演算{*1，*3}，乗算器(MUL2)に{*2，*4，*5}を割当てる．この割当てにより，すべての乗算器の入力に外部入力レジスタを割当てられることができる．乗算の入力は外部入力レジスタにできたため，次に演算器の入出力順序深度を 0 にするため, 減算器(SUB1)の右入力変数{c，f}のいずれかを外部入力レジスタに，乗算器(MUL1)の出力変数{a，c}のいずれかを外部出力レジスタに，乗算器(MUL2)の出力変数{b，d，f}のいずれかを外部出力レジスタに割当てる．バインディングの結果を図 4 に示す．また，演算器の入出力順序深度削減指向バインディング実行後，各演算器の入出力順序深度を表 4 に示す．表 4 から全演算器の演算器入出力順序深度が 0 であることがわかる．

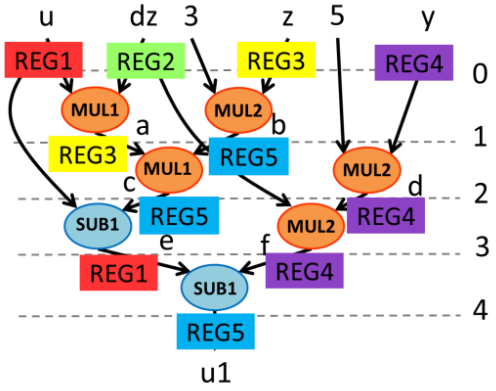

図 4.演算器の入出力順序深度削減指向バインディング

表5. 実験結果

回路名	バインディング方法	レジスタバインディング	演算器バインディング	MUX総入力数	故障検出率［%］	故障検出効率［%］	ATPG時間［s］	テストパターン数
ex2	［4］	R1:a,c,f,u1 R2:z,b R3:y,d R4:u,e R5:dz	(*)*1,*4,*5 (*)*2,*3 (−)−1,−2	117	99.74	99.82	13.77	285
	提案手法	R1:u,e R2:dz R3:z,a R4:d,f R5:b,c,u1	(*)*1,*3 (−)−1,−2 (*)*2,*4,*5	133	99.77	99.90	6.48	211
oven − ctrl	［4］	R1:T1,a R2:T2,c,diff R3:T3 R4:T4 R5:b,d,e,Tavg、Terror R6:Tset	(+)+1,+4 (+)+2,+3,+5 (−)−1 (*)*1	114	99.84	99.84	4.02	178
	提案手法	R1:Tset R2:T4,Tdiff R3:T1,a,Terror,Tavg,e R4:T2 R5:T3,c R6:b	(+)+1,+3 (+)+2,+4,+5 (−)−1 (*)*1	160	99.87	99.87	5.08	221
$\mathrm{DiffE}q_A$	［4］	R1:z,z1 R2:a,c,f,u1 R3:b,d,g,y1 R4:A R5:dz R6:u,e R7:y R8:ctrl	(*)*1,*4,*6 (*)*2,*3,*5 (−)−1,−2 (+)+1,+2	165	99.84	99.87	6.17	216
	提案手法	R1:u,e R2:dz R3:z,z1 R4:y R5:A,c,f R6:b,d,g,y1 R7:ctrl,a R8:u1	(*)*1,*3,*5 (*)*2,*4,*6 (−)−1,−2 (+)+1,+2	189	99.83	99.87	5.72	202
Tseng_A	［4］	R1:v1,b,d,w3 R2:v4,c,p,w2 R3:v3,a R4:v5,f R5:u2	(+)+1,+3 (−)−1 (+)+2 (*)*1,*2	160	99.87	99.87	4.63	269
	提案手法	R1:v1,c,d,w2 R2:v2 R3:v3,a,e,w3 R4:v4,b R5:v5,f	(+)+1,+3 (−)−1 (+)+2 (*)*1,*2	136	99.86	99.86	4.89	299

表 4. 演算器の入出力順序深度

演算器	左入力	右入力	出力
MUL1	0	0	0
MUL2	0	0	0
SUB1	0	0	0

4. 実験結果

本論文では，文献[4]に掲載された回路ex2，oven-ctrl, $DiffEq_A$，$Tseng_A$に対して実験を行い，文献[4]の手法と比較した結果を表5に示す．従来法では，ex2の乗算器の左入力と減算器の右入力の順序深度は0ではなかった．また，oven-ctrl，$DiffEq_A$にも演算器入出力順序深度が0でないものがあった．それに対して提案手法では，演算器入出力順序深度削減指向バインディングを行った結果すべての演算器入出力順序深度を0にできた．そのうちで，故障検出率，故障検出効率が向上した回路はex2，oven-ctrl，テストパターン数が減少したものはex2，$DiffEq_A$となった．しかしながら，$Tseng_A$のような，[4]のバインディングですでに演算器入出力順序深度がすべて0だったものに対しては，より良い結果は出せなかった．

5. まとめ

本論文では，テスト容易化のための演算器入出力順序深度削減指向バインディングのアルゴリズムを提案した．今後の課題として，コントロールフローインテンシブ回路での実験などが挙げられる．

[参考文献]

[1] 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.

[2] 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

[3] Tien-Chien Lee, Wayne H. Wolf, Niraj K. Jha, John M. Acken: “Behavioral Synthesis for Easy Testability in Data Path Allocation”, Int.Conf.Computer Design, pp29-32, 1992.

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