ISSN 2186-5647

2-51

演算器入出力順序深度削減向けバインディング法

日大生産工 ○佐藤 護　日大生産工(院) 増田 哲也　NEC エンジニアリング株式会社
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伴い，製造される大規模集積回路(Large Scale Integrated circuits：LSI)の大規模化，高集積化が急速に進んでいる．これに伴い，LSI の設計コストの増加が問題視されている．現在の LSI 設計において，レジスタ転送レベル(Register Transfer Level：RTL)での回路設計が主流である．RTL での回路設計は，回路構造を設計者の手によって詳細に記述をする必要がある．したがって，LSI の大規模化，高集積化に伴い RTL での回路設計が困難となっている．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る．それゆえ，回路の動作のみを記述した動作記述から，その動作を実現する RTL 回路記述を自動生成する動作合成が提案されている[1][2]．動作記述は，RTL 回路記述に対して記述量が少なく，設計生産性に優れている．その反面，合成される RTL 回路の構造情報は，すべて動作合成ツールが決定するため，ツールの性能が合成される回路の性能に大きく影響する．従来では，動作合成ツールによって合成される RTL 回路の品質が，人手によって設計された RTL 回路と回路面積や性能，テスト容易性の観点から比較して大幅に劣ることが問題視されていた．この問題に対し，近年では，動作合成によって生成される回路の面積や性能を最適化する手法が提案されている．これにより，動作合成による LSI の設計は，実用的なレベルであると報告されている[3].

また，設計製造された LSI は良品，不良品を判別するためのテストが実行され，良品と判定された LSI のみを出荷する必要がある．しかしながら，前述した LSI の大規模化，複雑化の進展に加え，ハードウェアセキュリティの重要性のため，LSI のテストが困難となっている．この問題を解決するために，LSI のテスト容易性を考慮した動作合成法が提案されている．

テスト容易化動作合成手法として，データパスの順序深度削減を考慮した動作合成法が提案されている[4]. 外部入力 i と外部入力 j 間の順序深度とは，外部入力 i から外部出力 j までの経路における，最小のレジスタ段数のことである．しかしながら，文献[4]の手法では，順序深度が削減された経路上に存在しないハードウェア要素におけるテスト容易性の向上は考慮されていない．したがって，そのようなハードウェア要素はテスト生成困難となる可能性がある．

本論文では，演算器入出力順序深度削減を考慮したバインディング法を提案する．本手法はバインディング時にデータパスの演算器入出力順序深度を削減することを目的とする．演算器入力順序深度と演算器出力順序深度とは，それぞれ演算器の入力から外部入力レジスタに到達するまでの経路上に存在する最小のレジスタ段数および，演算器の出力から外部出力レジスタまで到達するまでの経路上に存在する最小のレジスタ段数である．本手法はデータパスの大部分を占める演算器に対して演算器入出力順序深度を削減することで，データパス全体のテスト容易性の向上を図る．ベンチマーク回路に対する実験により，提案手法の有効性を示す.

2. 動作合成

動作合成とは，回路の動作記述から，その動作を実現する RTL 回路記述を自動生成する技術である．図 1 に動作合成の流れを示す.

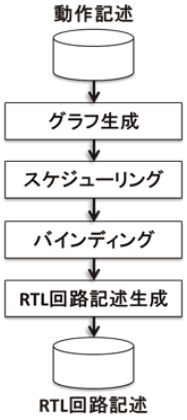

図 1 動作合成の流れ

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 つのフェーズから構成される.

グラフ生成は，与えられた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)に変換するフェーズである．スケジューリングは，動作記述中の各演算の実行時刻を決定するフェーズである．バインディングは，各変数にレジスタ，各演算に演算器を割当てるフェーズである.

最後に CDFG におけるスケジューリング，バインディング結果をもとに，RTL 回路記述が生成される．RTL 回路記述は，データパスとコントローラで構成されている.

2-1. バインディング

バインディングでは，スケジューリングによって実行時刻が決定された演算や，ライフタイム[1]が決定された変数に対して，演算には演算器，変数にはレジスタを割当てる処理である．ライフタイムとは，変数が値を保持し始める

A Binding Method to Reduce Sequential Depth for Inputs and Outputs of Operational Units

Mamoru SATO, Tetsuya MASUDA, Jun NISHIMAKI, Toshinori HOSOKAWA and Hideo FUJIWARA

時刻から，値を保持しなくなるまでの時刻である．バインディングでは演算器数，レジスタ数を削減するためにハードウェア要素の共有化を考える．共有化の結果によって，回路面積やテスト容易性など回路の性能が異なる．また，本論文で提案する演算器入出力順序深度削減を考慮したバインディングでは，共有化を実行しない状態での演算器入出力順序深度を解析するために，始めに 1 個の演算，変数に対し 1 個の演算器，レジスタを割当てる．また，バインディングは演算器バインディングとレジスタバインディングに分けられる．次に，例題とするスケジューリングが終了した CDFG である SDFG(Scheduled Data Flow Graph：SDFG)を図 2 に示す．

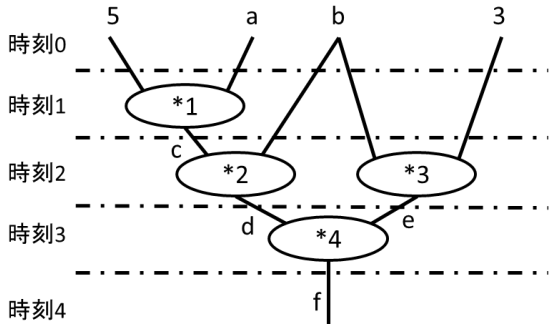

図 2　SDFG　例

図 2 に対して 1 演算に 1 演算器，1 変数に 1 レジスタの割当てを行う．その結果を図 3 に示す．

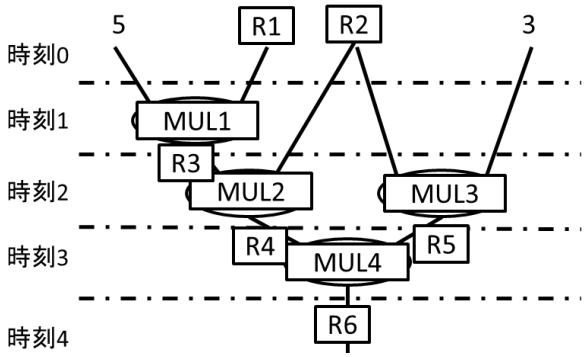

図 3　バインディング済み SDFG　例

次に演算器バインディングを実行する．同時刻に実行される演算器同士は共有化不可能である．レジスタも同様に、同時刻に値を保持するレジスタ同士は共有化することができない．すなわち、ライフタイム時刻が重なっているレジスタ同士は共有化することができない．

3. 演算器順序深度

以下に，本論文に頻出する単語の定義を行う．

1. 順序深度
   ある外部入力レジスタから任意の外部出力レジスタまでの経路に存在する最小のレジスタ数．
2. 演算器 k の演算器入出力順序深度
   演算器 k の入力（出力）から任意の外部入力（出力）レジスタまでの経路に存在する最小のレジスタ数．
3. レジスタ v のレジスタ入出力順序深度
   レジスタ v の入力（出力）から任意の外部入力（出力）レジスタまでの経路に存在する最小のレジスタ数．ただし v は，外部入力レジスタ，外部出力レジスタ以外とする．
4. データパス順序深度
   外部入力レジスタの順序深度の中で，最大の順序深度のこと．

データパスは主に演算器，レジスタ，マルチプレクサなどのハードウェア要素で構成される．各ハードウェア要素のテストを考える場合，その中で多くのテストパターンが必要であると考えられるハードウェア要素は，回路規模が大きい演算器である．あるテスト対象モジュールのテスト生成において，そのモジュールの可制御性，可観測性が高い場合，そのモジュールはテスト生成が容易と考えられる．データパスの順序深度のみの削減を考慮した場合，外部入力から外部出力までのレジスタ段数の小さい経路に存在するモジュールに関して高い可制御性，可観測性は確保されるが，そのレジスタ段数の小さい経路に存在しないモジュールに関して，テスト容易性は考慮されておらず，テストが困難となる可能性がある．その問題を解決するために，本論文では演算器に対して，可制御性，可観測性の向上を図るために演算器入出力順序深度[5]を定義し，演算器入出力順序深度削減向けバインディング法を提案する

図 4 に，RTL データパスの例を示す．

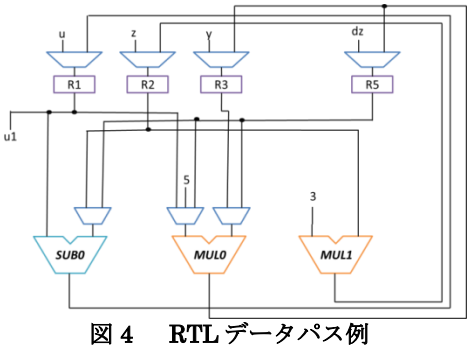

図 4　RTL データパス例

図 4 の演算器における演算器入出力順序深度を解析した結果を表 1 に示す．

表 1 図 4 の例題回路　順序深度表

	出力	左入力	右入力
SUB0	0	0	0
MUL0	1	0	0
MUL1	1	∞	0

定数のみの入力が存在する演算器は任意の値の制御が不可能であるため，定数制御をうける入力側の演算器入力順序深度を∞と定義する．図 4 の MUL1 の左入力は定数 3 のみを入力としているため，MUL1 の演算器左入力順序深度は∞となる．MUL1 の右入力は外部入力レジスタまでの経路が存在し，外部入力レジスタ R2 に到達するまでレジスタ段数は 0 であるため，MUL1 の演算器右入力順序深度は 0 となる．定数入力のみの演算器はテスト生成において非常に大きな問題となる．そのため，本論文で提案するバインディング法では，定数のみを入力とする演算器の削減を優先的に実行する．その後，各演算器の演算器入出力順序深度削減を考慮したバインディングを実行する．以上の処理によりテスト容易性の高いデータパスの生成を実現する．

4. 演算器入出力順序深度削減を考慮したバインディング

本章では，提案手法である演算器入出力順序深度削減を考慮したバインディングのアルゴリズムを説明する．本手法では，演算器バインディングを実行した後にレジスタバインディングを実行する．演算器入出力順序深度の改善を

行う共有化の際に選択が発生した場合，演算器入出力順序深度削減の効果が高い共有化の選択を行う．また，演算器入出力順序深度削減を考慮したバインディングにおいて，演算器入出力順序深度の改善が見込めない共有化の時，演算器の入力となる変数の数の均一化を考慮した共有化を実行する．

4.1. 演算器バインディング

ある演算器の入力が定数のみの場合，その演算器は可制御性の観点からテスト生成困難であるといえる．また，定数制御演算器数の削減は演算器バインディング時に考慮する必要がある．したがって，本手法の演算器バインディングでは定数制御演算器数の削減を優先的に実行する．

本手法の演算器バインディングでは，Step1 において，すべての演算器に対して，制御される入力の種類によって分類を行う．右入力のみ定数の演算器（分類 2），左入力のみ定数の演算器(分類 1)，定数入力が存在しない演算器(分類 0)の 3 種類に分類する． 図 3 の SDFG に対して演算器入出力順序深度解析と分類した結果を示す．

表 2　図 3　解析，分類結果

	出力	左入力	右入力	分類
MUL1	2	∞	0	2
MUL2	1	1	0	0
MUL3	1	0	∞	1
MUL4	0	2	1	0

Step2 では，分類 1,2 に分類された演算器が共有できる演算器において，分類 1,2 の数を均一にする．この処理では，加算器や乗算器に対して，演算器の左入力と右入力を入れ替えることで，分類を 1 から 2 へ，2 から 1 へ変更する．これにより，共有化できる演算器が分類 1 同士，分類 2 同士であることが防げる．よって，定数制御演算器を削減するための共有化の組み合わせを増やすことが可能である．例ではこの処理は不要である．

Step3 では，演算器の共有化を行う．このとき，分類 1 と分類 2 の演算器同士でお互いの定数制御入力を削除することを優先する．分類 0 の演算器に対しては，演算器入力順序深度と演算器出力順序深度いずれかが大きい演算器から順に，演算器入力順序深度及び演算器出力順序深度削減を考慮した共有化を実行する．演算器入力順序深度と演算器出力順序深度の値が同一である場合、演算器入力順序深度の削減を優先する．表 2 の結果から，MUL1 の左入力と MUL3 の右入力を削減する必要があることがわかる．このように削減すべき数値が同一の演算器入力順序深度がある場合，削減すべき個所以外の演算器入出力順序深度を見比べ，合計した数値が大きい演算器を優先的に共有化に使用する．従って，MUL1 から優先的に共有化を実行する．MUL1 は分類 2 なので，分類 1 と優先的に共有化を実行する．なぜなら，1 つの共有化によって定数制御演算器を 2 つ減らせることが可能だからだ．例では MUL1 と MUL3 の共有化を実行し，MUL1 とする．その後，一意的に MUL2 と MUL4 の共有化を実行し，MUL2 とする．図 3 に対して演算器バインディングを行った結果を表 3 に示す．

表 3 図 3 演算器バインディング結果

	出力	左入力	右入力	分類
MUL1	1	0	0	0
MUL2	0	1	0	0

以上の処理を回路に存在する演算器の種類ごとに実行し，可能な限り演算器の共有化を実行した後，演算器バインディングを終了する．演算器バインディング後の SDFG を図 5 に示す．

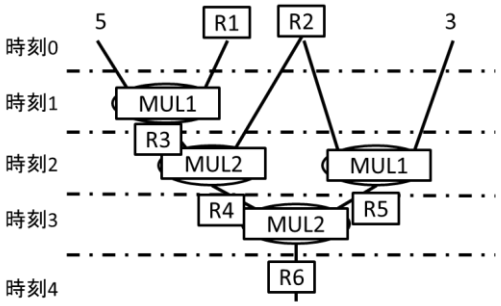

図 5　バインディング済み DFG

4.2. レジスタバインディング

表 3 を見ると，MUL1 の演算器出力順序深度と，MUL2 の左入力順序深度に削減の余地が見られるため，レジスタバインディングによる削減を試みる．演算器バインディングと同様に，削減できる値が同一の場合，演算器出力順序深度より演算器入力順序深度を優先する．よって，MUL2 の左入力から削減を行う．演算器入出力順序深度を可能な限り削減した後はレジスタ数が最小となるように可能な限りレジスタの共有化を実行し，終了する．

Step1 において，演算器の入出力順序深度を考慮したレジスタの共有化を行う．共有化によって削減できる演算器入出力順序深度が同一の選択があった場合，共有化可能レジスタ数という尺度を用いて，それが最も小さいレジスタを選択し，共有化を行う．この尺度は，処理の後半においても各レジスタにおける共有可能なレジスタ数を多く確保し，演算器入出力順序深度を改善可能なレジスタを確保するために用いる．共有可能レジスタ解析した結果を表 4 に示す．

表 4 共有可能レジスタ解析

	共有可能数		
R1	4	R4	4
R2	3	R5	4
R3	4	R6	5

MUL2 の演算器左入力の順序深度を削減するには，MUL2 の左入力を制御するレジスタの入力順序深度を削減する必要がある．例では，レジスタ R3 と R4 が該当する．表 4 より，R3 と R4 どちらも同一の数値なので，時刻が若い R3 を使用し削減を実行する．R3 と共有化可能かつレジスタの入力順序深度が 0 なレジスタは R1 が該当するため，R1 と R3 を共有化し，名前を R1 とする．共有可能レジスタの再解析をかけた結果を表 5 に示す．

表 5　共有可能レジスタ解析

	共有可能数		
R1	3	R4	3
R2	3	R5	3
		R6	4

次に MUL1 の演算器出力順序深度を削減する．MUL1 の演算器出力順序深度を削減するには，MUL1 の出力を格納するレジスタの，レジスタの出力順序深度を削減する必要がある．例では，R1(図では R3 だが，共有化されたため)と R5 が該当する．表 5 より R1 と R3 どちらも同一の数値なので，時刻が若い R1 を使用し削減を実行する．R1 と共有化可能かつレジスタ出力順序深度が 0 なレジスタは

R6 が該当するため，R1 と R6 を共有化し，名前を R1 とする．この処理が終わった時点で，演算器入出力順序深度が全て 0 となるため，レジスタ数が最小となるように共有化を行い、レジスタバインディングを終了する．

5. 実験結果

本論文で提案した手法を用いて，DFG ベースである ARF[6] ，BPF[6]，FFT[6]，EX2[7]などの定数入力の演算が存在する回路に対して，実験を行った．実験では，動作合成ツール PICTHY[8]を用いて，本手法を適用したバインディング済の DFG より RTL 回路を生成し，論理合成によって生成された回路に対してテスト生成を行った．テスト生成はテスト生成ツール STAGY[9]を用いた．比較対象は実験回路に対して PICTHY に実装されている演算器・レジスタ数最小化を指向したバインディングを適用した回路のテスト生成結果である．表 2 に各回路におけるテスト生成結果を示す．

表 2　テスト生成結果

回路名	故障検出効率(%)		テスト生成時間(sec)	
	PICTHY	提案手法	PICTHY	提案手法
ex2	94.54	99.71	489.46	504.94
ARF	68.28	76.03	77033.91	49699.67
FFT	85.24	92.04	8916.61	16619.33
BPF	77.90	85.14	18626.15	21100.56

提案手法を用いることで，実験に使用したベンチマーク回路 4 つにおいて平均して約 6.74%，最大 7.75%故障検出効率が向上した．テスト生成時間は大きく削減できたものと増加したものに分かれている．大きく削減できたものに関しては，回路構造上検出するためのテストパターン生成に時間がかかっていた故障が，回路構造が変わったことにより検出が容易になったと考えられる．増加したものに関しては，単純に検出できる故障の数の増加に伴いテスト生成時間が伸びたと考えられる．

6. おわりに

本論文では，テスト容易な回路を生成するために，動作合成のフェーズの 1 つであるバインディングに着目し，演算器入出力順序深度の削減を考慮したバインディング法を提案した．実験から，故障検出効率が平均で 6.72%向上した．実験結果から演算器入出力順序深度という概念が有用であるとともに，それを削減することで回路がテスト容易になることが明らかになった．今後の課題として，共有化を進める際に，単純に大きな演算器入力順序深度及び演算器出力順序深度から削減するのではなく，乗算器の演算器入力順序深度及び演算器出力順序深度を優先的に削減するために，共有化をする演算器を評価する評価式を導入することによって，よりよい結果を得ることができると推測する．

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]M.C.McFarland, A.C.Parker, R.Camposano: The high-level synthesis of digital systems, Proc. IEEE, 301-318, 1990

[3]Kazutoshi Wakabayashi，CyberWorkBench: Integrated Design Environment Based on C-based Behavior Synthesis and Verification，2005

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

[5] 林 愛美，西間木 淳，増田 哲也，細川 利典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

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

[7]M.T.-C.Lee, “High-Level Test Synthesis of Digital VLSI Circuits”, Artech House Publishers, 1997.

[8]石井英明，細川利典，コントロール／データフローグラフを用いた動作合成システム PICTHY の評価，日本大学　生産工学部　数理情報工学科　学術講演会，2011

[9]Kazuya Sugiki, Toshinori Hosokawa, and Masayoshi Yoshimura, ”A Test Generation Method for Datapath Circuits Using Functional Time Expansion Models ”, The Ninth Workshop on RTL and High Level Testing (WRTLT’08), pp.39 -44, Nov. 2008.