ISSN 2186-5647

2-34

演算器入出力順序深度削減指向スケジューリング法

日大生産工 ○山口 聖也　日大生産工(院) 佐藤 護　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 設計では，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た[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，合成される回路の品質に大きく影響する．したがって，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，消費電力，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．これに対し，近年では，動作合成によって生成される回路の性能，面積，消費電力，テスト容易性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，複雑化が進むにつれて，LSI のテストが困難となっている．そのため，動作合成でも LSI テストの困難性を解消するための研究が進められている[4]．

LSI テストの困難性を解消するための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バインディング法が提案されている[4]．順序深度とは，ある外部入力からある外部出力までのレジスタ段数の最短経路中に存在するレジスタ数のことである．しかしながら，文献[4]の手法では，最大レジスタ段数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．

その問題を解決するために， データパスの大部分を占め，テスト生成が困難であると考えられるハードウェア要素である演算器に対して定義した演算器入出力順序深度を削減する．これによってデータパス全体のテスト容易性の向上を図る．そのための手法として，演算器入力および出力順序深度削減指向テスト容易化バインディング法が提案されている[5]．しかしながら，文献[5]のバインディング手法だけでは，外部入出力変数が少なく，内部変数が多い回路において，演算器入出力順序深度を最小にできない可能性がある．

本論文では，演算器入出力順序深度削減を考慮したスケジューリング法を提案する．演算器入出力順序深度削減指向スケジューリングは，演算器入出力順序深度削減指向バインディングの効果をより高めるためのものである．本手法と文献[5]のバインディング法により，各演算器の演算器入出力順序深度が，小さくなることが見込める．各演算器の演算器入出力順序深度が少なくなることで，各演算器の可制御性，可観測性を向上させることを目的としている．

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 2-1 に動作合成のフローチャートを示す．

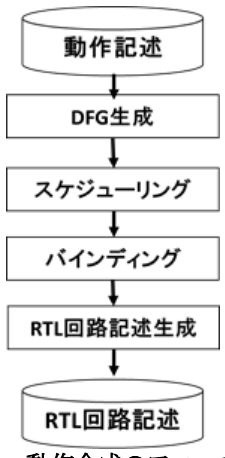

図 2-1.　動作合成のフローチャート

一般に作合成は，DFG 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．

DFG 生成は，与えられた動作記述をデータフローグラフ(Data Flow Graph：DFG)に変換するフェーズである．スケジューリングは，DFG 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 DFG におけるスケジューリング・バインディングの結果をもとに，RTL 回路記述生成が実行され，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. スケジューリング

スケジューリングでは，DFG に対して，どの演算をどの時刻で行うか，すなわち演算操作の実行時刻を決定する．代表的なスケジューリングアルゴリズムとして，可能な限

A Scheduling Method to Reduce Sequential Depth for Inputs and Outputs of Operational Units in Datapath

Seiya YAMAGUCHI,　Mamoru SATOU,　Jun NISHIMAKI,　Toshinori HOSOKAWA and Hideo FUJIWARA

り早い時刻に演算を割当てる ASAP(As Soon As Possible)スケジューリング[6]や，可能な限り遅い時刻に演算を割当てる ALAP(As Late As Possible)スケジューリング[6]がある.他には,面積最小化を考慮したFDS(Force Directed Scheduling)[7][8]や，使用する演算器数を制約として与えるリストスケジューリング[8]などが提案されている．図 2-1-2 に，図 2-1-1 の DFG に対して ALAP を実行した例を示す.

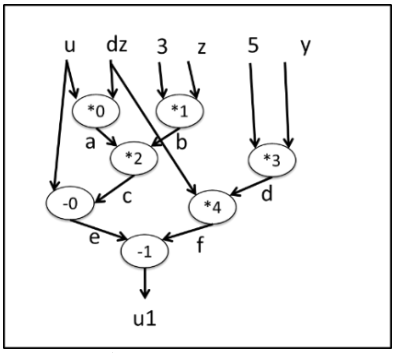

図 2-1-1.　DFG

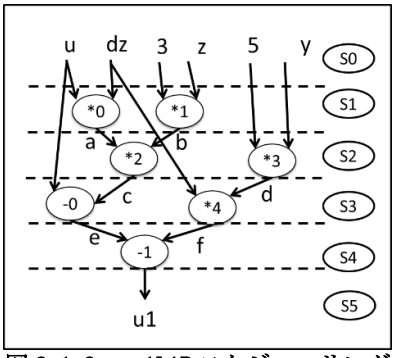

図 2-1-2.　ALAP スケジューリング

図 2-1-1 において，演算の割当て時刻に自由度がある演算は*3 である．ALAP スケジューリングを実行した場合，*3 は時刻 S2 に割当てられる．対して，図 2-1-1 の DFG に対して ASAP スケジューリングを実行した場合，*3 は時刻 S1 に割当てられる．スケジューリング結果が異なると，生成される回路の面積や，テスト容易性がそれぞれ異なる．

2-2.　バインディング

バインディングとは，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．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.バインディングした結果を図 2-2-1 に示す.

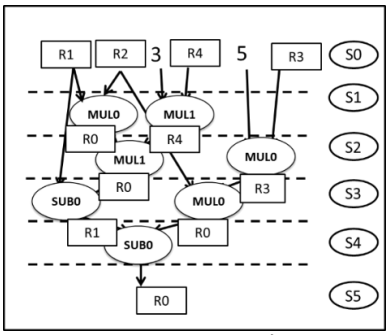

図 2-2-1.　バインディング

図 2-2-1 では，演算器バインディングを行うことによって，演算器 MUL1 には演算*1，*2 が割当てられる．演算器の割当てを共有できる条件として，同じ時刻で演算が実行されていないことと，演算の種類が同じことが挙げられる．レジスタバインディングの処理をすることで，　レジスタ R0 には変数 a，c，f，u1 が割当てられる．レジスタの割当てを共有できる条件として，同じ時刻でレジスタが使用されていないことが挙げられる．レジスタは，同一時刻に 2 つ以上の変数を保持することができないため，変数が値を保持している時間を解析する必要がある．この保持している時間のことをライフタイムと呼ぶ．レジスタバインディングをする際に必要である．

3.　演算器入出力順序深度

演算器入力（出力）順序深度[5]とは，ある演算器の入力（出力）から，任意の外部入出力に到達するまでの各経路における最小のレジスタ数である．

テスト対象モジュールである演算器，レジスタ，マルチプレクサなどのハードウェア要素が存在するが，その中で多くのテストパターンを必要とするハードウェア要素は，演算器である．

演算器入出力順序深度は，データパス順序深度を削減するだけではテスト容易性が保証されない回路が存在する．

図 3-1 に，RTL データパスの例を示す.

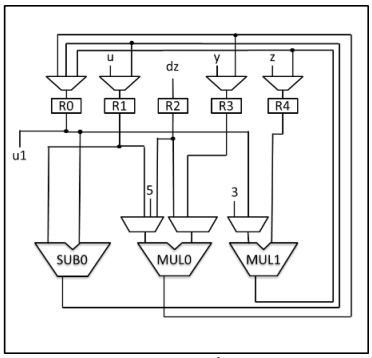

図 3-1.　RTL データパス例

図 3-1. の各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 3-1 に示す.

表 3-1.　演算器入出力順序深度表

	出力	左入力	右入力
MUL0	1	1	1
MUL1	1	2	1
SUB0	1	1	2

定数のみの入力が存在する演算器は外部入力に到達しないため，演算器入力順序深度を∞と定義する．例えば，MUL1 の右入力は外部入力に到達するまでに通る経路上に存在する最小レジスタ数は 1 であるため,MUL1 の演算器右入力順序深度は 1 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．

4.　演算器入出力順序深度削減バインディング

演算器入出力順序深度削減バインディング[5]は，各演算器の可制御性および可観測性の向上を考慮することにより，データパスのテスト容易性の向上を目的としている．文献[5]のバインディングでは最初に 1 変数に 1 レジスタ，1 演算に 1 演算器を割当てる.

図 4-1 に図 2-1-2 のスケジューリング結果に演算器入出力順序深度削減バインディングを実行した場合のバインディング結果を示す.

表 4-1 に図 4-1 のバインディング結果から生成したデー

タパスの演算器入出力順序深度を示す.

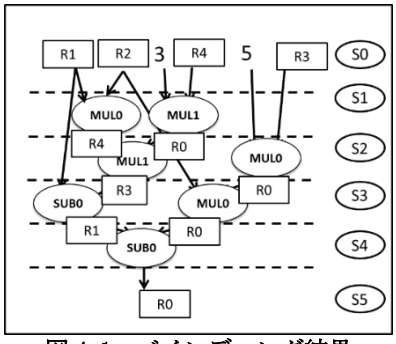

図 4-1. バインディング結果

表 4-1. 演算器入出力順序深度

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

演算器入出力順序深度削減指向バインディングは，演算器入出力順序深度を削減する方法である．バインディングでは，各演算器の入力変数，出力変数にそれぞれ外部入力レジスタ，外部出力レジスタを割当てることを理想としている．演算器入出力順序深度は，表 3-1 と表 4-1 から図 2-2-1 の乗算器 MUL1 の左入力と減算器 SUB0 の右入力の演算器入力順序深度が削減されていることがわかり，可制御性が向上していることがわかる．

しかしながら，大規模回路で，外部入出力変数が少ない回路では，バインディング法のみでは可制御性と可観測性の向上が困難である場合がある．したがって，大規模回路で外部入出力変数が少ない回路において演算器入出力順序深度を削減し，可制御性と可観測性を向上させるためにスケジューリングに着目する．スケジューリングでは，いかに外部入出力レジスタを内部変数に割当てられるかということを考慮することにより，バインディング時に演算器入出力順序深度を削減できる可能性を高める．

5. 演算器入出力順序深度削減指向スケジューリング

演算器入出力順序深度削減指向スケジューリングは，演算器入出力順序深度削減指向バインディングの効果をより高めるためのものである．本手法と文献[5]のバインディング法により，各演算器の演算器入出力順序深度が，小さくなることが見込める．各演算器の演算器入出力順序深度が少さくなることで，演算器の可制御性・可観測性の向上が見込める．

演算器入出力順序深度削減指向スケジューリングでは，演算器数を最小にし，外部入出力変数のライフタイムを最小にすることで，演算器入出力順序深度削減指向バインディングの際に，内部変数に外部入出力レジスタを割当てる確率を高めることが目的である．内部変数に外部入出力レジスタを割当てることができれば演算器入出力順序深度は 1 まで削減され，可制御性と可観測性の向上が期待できる．

図 5-1 に演算器入出力順序深度削減指向スケジューリングアルゴリズムのフローチャートを示す.

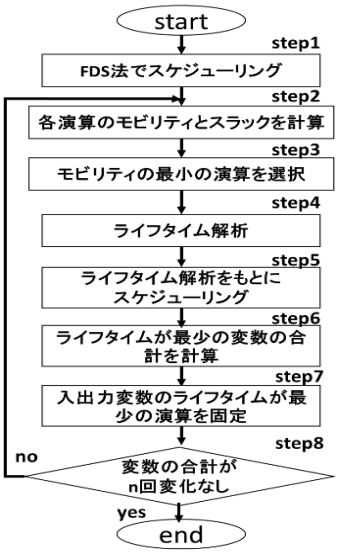

図 5-1. 演算器入出力順序深度削減指向スケジューリングアルゴリズム

step1: 面積最小化を考慮した FDS(Force Directed Scheduling)法でスケジューリング法を行うことで最小演算器数を求める.

step2:各演算に対してモビリティ(移動度)[7][8]とスラック(移動範囲)[7][8]を求める．モビリティとは，演算をスケジューリングする際に，割当てることができる時刻の範囲の大きさである．スラックは演算を時刻に割当てることができる時刻の範囲を表している.

step3:モビリティが最小の演算を選択する.

step4:演算を割当て可能な各時刻に割当てた場合の入出力変数のライフタイムを求める.

step5:時刻毎に求めた変数のライフタイムを比較し，ライフタイムが 1 になる変数の合計が最も多い時刻に演算をスケジューリングする.

step6:スケジューリング完了後に，全変数のライフタイムを求める．次にライフタイムが 1 になっている変数の合計を計算する.

step7:スケジューリング結果毎に，ライフタイムが 1 になっている変数の合計を比較し，合計が最大になるスケジューリング結果を採用する.

step8:step2～step7 を繰り返し，スケジューリング結果に n 回変化がなかった場合にスケジューリングを終了する.

6. 実験結果

本論文で提案したスケジューリング手法を用いて,DFG ベース回路である ARF[12] , BPF[12]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．また，外部入出力変数を極端に少なくし，内部変数を多く設計した回路 SYG，H1 にも同様に実験を行った.実験では,動作合成ツール PICTHY[10]を用いて，本手法を適用したスケジューリング済の DFG より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後，演算器入出力順序深度解析を行った．比較対象は実験回路に対して PICTHY に実装されている面積最小化を指向したスケジューリング(FDS)を適用した回路の各演算器の演算器入出力順序深度である.

表 6-1，表 6-2，表 6-3，表 6-4 はそれぞれ ARF，BPF，SYG，H1 における各演算器の演算器入出力順序深度を示す.

表 6-1. ARF における演算器入出力順序深度

	従来手法(FDS)			提案手法		
	出力	左入力	右入力	出力	左入力	右入力
*0	1	1	1	1	1	1
*1	1	1	1	1	1	1
*2	1	1	1	1	1	1
*3	1	1	1	1	1	1
+0	1	1	1	1	1	1
+1	1	1	1	1	1	1

表 6-2. BPF における演算器入出力順序深度

	従来手法(FDS)			提案手法		
	出力	左入力	右入力	出力	左入力	右入力
*0	1	1	1	1	1	1
*1	1	1	1	1	1	1
+1	1	1	1	1	1	1
+2	1	1	1	1	1	1
-0	1	1	1	1	1	1
-1	1	1	1	1	1	1

表 6-3. SYG における演算器入出力順序深度

	従来手法(FDS)			提案手法		
	出力	左入力	右入力	出力	左入力	右入力
*0	1	1	1	1	1	1
*1	1	1	1	1	1	1
*2	1	1	1	1	1	1
+0	1	1	1	1	1	1
-0	1	1	1	1	1	1
-1	1	1	1	1	1	1

表 6-4. H1 における演算器入出力順序深度

	従来手法(FDS)			提案手法		
	出力	左入力	右入力	出力	左入力	右入力
*0	1	1	1	1	1	1
*1	1	1	1	1	1	1
*2	1	1	1	1	1	1
+0	1	1	1	1	1	1
+1	1	1	2	1	1	1
-0	1	1	1	1	1	1
-1	1	1	1	1	1	1

提案手法を用いることで H1 において，+1 の演算器右入力順序深度を削減した．H1 以外の ARF，BPF，SYG の回路において，演算器入出力順序深度は文献[5]のバインディング法だけでも 1 にすることができた．この要因として回路規模が挙げられる．回路規模が小さいために，演算器入出力順序深度を削減する際に，演算器入出力順序深度削減指向バインディングによって得られる効果が大きく，従来手法と提案手法での差異が生じにくいと考えられる．

7. おわりに

本論文では，演算器入出力順序深度削減指向バインディングの効果をより高め，各演算器の演算器入出力順序深度を削減する，動作合成のフェーズの 1 つであるスケジューリングに着目し，演算器入出力順序深度の削減を考慮したスケジューリング法を提案した．実験結果から H1 に対して本提案手法を適用することで演算器入出力順序深度を削減することができた．この結果からテスト生成時間の面でテスト容易性が向上することが文献[5]で示されている．このことから本提案手法によってテスト容易性向上が期待できることを示した．今後の課題として，テスト容易化機能的時間展開モデルを用いたテスト生成法[15]を用いて故障検出率やテスト生成時間を評価することが挙げられる．

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]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] 佐藤 護, 増田 哲也, 西間木 淳, 細川 利典, 藤原 秀雄, テスト容易化機能的時間展開モデル生成のためのバインディング法, FTC 研究会, 2016

[6]Giovanni De Micheli , “SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS”, McGraw-Hill, inc, 1994.

[7]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.

[8]Giovanni De Micheli , “SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS”, McGraw-Hill, inc, 1994.

[9]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

[10]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.

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

[12]PIERRE G. PAULIN , LOHN P. KNIGHT : “Force-Directed Scheduling for the Behavioral Synthesis of ASIC’s”, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN 1989.

[13]早川鉄平，細川利典，吉村正義，機能的時間展開モデルを用いたデータパス回路のテスト生成法(テスト生成，VLSI 設計とテスト及び一般)， DC2010-65

[14] T. Masuda, J. Nishimaki, T. Hosokawa and H. Fujiwara, "A Test Generation Method for Datapaths Using Easily Testable Functional Time Expansion Models and Controller Augmentation , " IEEE the 24th Asian Test Symposium (ATS'15), pp. 37-42, Nov. 2015.