レジスタ分散型アーキテクチャを対象とするフロアプランとタイミング制約を考慮した高位合成手法

田中 真[†] 内田純平[†] 宮岡祐一郎[†]
戸川 望[†] 柳澤政生[†] 大附辰夫[†]

演算器ごとに専用のローカルレジスタを持たせるレジスタ分散型アーキテクチャを用いると，レジスタ間データ転送を利用することによって配線遅延が回路の性能に与える影響を削減することが可能である．しかし，高位合成のスケジューリングの段階からフロアプラン情報を考慮する必要がある．本論文では，レジスタ分散型をターゲットアーキテクチャとし，(1) スケジューリング，(2) レジスタバインディング，(3) モジュール配置，の工程を繰り返し，(3) から得られたフロアプラン情報を (1), (3) の工程にフィードバックすることによって，解（合成結果）を収束させる高位合成手法を提案する．フロアプラン情報をスケジューリングに反映させるために，フィードバックされた配置情報とタイミング制約に基づいて，レジスタ間データ転送を利用することができるスケジューリング手法を提案する．また，レジスタ分散型に対応したレジスタバインディング手法を提案する．提案バインディング手法では，ローカルレジスタを入力側と出力側で区別し，出力側レジスタで可能な限りデータを保持することにより，総レジスタ数を削減する．提案手法により，フロアプランを考慮したレジスタ間データ転送を用いた回路を解として得ることが可能となる．計算機実験によって，提案手法の有効性を示す．

High-level Synthesis with Floorplaning and Timing Constraint for a Distributed-register Architecture

Akira Tanaka,[†] Jumpei Uchida,[†] Yuichiro Miyaoka,[†]
Nozomu Togawa,[†] Masao Yanagisawa[†] and Tatsuo Ohtsuki[†]

By using a distributed-register architecture, we can synthesize the circuits with register-to-register data transfer, and can reduce influence of interconnect delay. In this paper, we propose a high-level synthesis method targeting a distributed-register architecture. Our method repeats (1) scheduling, (2) register binding, (3) module placement processes, and feeds back floorplan information from (3) to (1) in order to decide which functional units use register-to-register data transfers. Our scheduling algorithm can use register-to-register data transfer based on floorplan and timing constraint. We also propose a register binding algorithm on a distributed-register architecture. We show effectiveness of the proposed methods through experimental results.

1. まえがき

大規模化，複雑化する LSI 設計の生産性を向上させるには，動作レベル記述による回路設計を可能とする高位合成を利用することはきわめて有効な手段である．従来の高位設計手法ではフロアプラニングを高位合成の後処理として扱っていたために，モジュールの配置関係やモジュール間の配線遅延情報を，高位合成の段階で考慮することはできなかった（ここで，モジュールとは回路を構成する要素として，演算器，制御回路，レジスタ，MUX を指すものとする）．近年の LSI 設計プロセスの微細化にともない，ゲート遅延に対して配線遅延が相対的に増加している．今後この傾向は継続すると予想され，配線遅延情報を高位合成の段階で考慮することが，実行速度や面積においてより優れた回路を合成するために必要となると考えられる．また，配線遅延を考慮したスケジューリングを実現できれば，下位工程に進んでからの配線遅延に起因する遅延制約違反を避けることが可能となる．

従来のフロアプランを考慮した高位合成手法の研究[2),8),12)] では，主にモジュール配置工程を工夫することによって，クリティカルパスに含まれる配線遅延の割合を削減してクロック周期を短縮するという手法

† 早稲田大学理工学部コンピュータ・ネットワーク工学科
Department of Computer Science, Waseda University

が用いられていた．文献 13) もレイアウト情報を考慮することによってクロック周期を短縮するような高位合成手法であるが，FPGA を対象とした手法である．文献 13) は，レイアウト情報をスケジューリング工程へフィードバックし，クロック周期制約に基づいてスケジューリングを修正するアプローチを用いている．これらの手法は，従来の高位合成手法で用いられてきた，演算器がレジスタを共有するアーキテクチャモデルを採用している．

配線遅延がボトルネックとなる状況下を想定して，文献 4)，5) では高位合成のターゲットアーキテクチャを抜本的に変えるアプローチであるレジスタ分散型アーキテクチャ（distributed-register architecture）が提案された．

レジスタ分散型アーキテクチャは，従来の高位合成手法で用いられていた演算器どうしでレジスタを共有するモデルとは異なり，演算器ごとに専用のローカルレジスタを配置する．各演算器は隣接した位置に配置されたローカルレジスタとデータをやりとりをするため，レジスタ集中共有型のモデルと比較して配線遅延の影響は小さくなる．また，離れて配置された演算器間のデータ転送にはレジスタ間データ転送を利用できるという大きな特徴を持つ．本論文では，配線遅延の影響が相対的に大きいディープサブミクロンプロセスに適したモデルである，レジスタ分散型をターゲットアーキテクチャとして用いる．

レジスタ分散型をターゲットアーキテクチャとしたフロアプランを考慮する高位合成手法の研究としては，コントロールステップを考慮しないで非同期的にスケジューリングした後，最適なクロック周期を決定する手法[4] や，最初に演算器のバインディングを決定してからスケジューリング，モジュール配置の工程に進む手法[5] が存在する．しかし，それらは分岐処理を含むアプリケーションの合成に対応していない，あるいはクロック周期制約に対応していないという課題を持つ．画像処理や暗号処理など，高位合成の対象となるアプリケーションの多くは分岐処理を含むので，分岐処理への対応は必須だと考えられる．また，高位合成の対象となる回路は通常システム LSI を構成する回路の一部分であるため，供給されるクロックの周波数が決まってしまっていたり，周辺回路にクロック周期を合わせなければならなかったりする場合も存在するはずである．したがって，実用レベルの高位合成では，クロック周期制約が与えられた状態で実行時間や面積を最小化することができなければならないと考えられる．

配置問題を線形計画問題に帰着させる手法[9] もレジスタ分散型をターゲットアーキテクチャとしているが，文献 9) は，レジスタは分散させて配置するが，レジスタ間データ転送を考慮することができない．そのほかにも，文献 1) が RDR（Regular Distributed Register）アーキテクチャというレジスタ分散型アーキテクチャに似たアイデアを採用している．チップを均一の大きさに分割して配置の粒度を下げるため，設計自体は容易化されるが，レジスタ分散型と比べるとアーキテクチャの自由度が少なく，回路性能を犠牲にしたアプローチとなっている．

また，レジスタ分散型を用いた既存研究[4),5),9)] においてはスケジューリングやモジュール配置に関する手法は示されているものの，レジスタのバインディング手法については言及されていない．さらに，モジュール配置の工程には言及しているにもかかわらず，配置のデッドスペースを考慮した評価がなされていないという問題点もある．

そこで，本論文ではレジスタ分散型をターゲットアーキテクチャとし，配置結果をフィードバックすることによってスケジューリングの段階から配置情報を考慮する高位合成手法を提案する．また，フィードバックされたフロアプラン情報とクロック周期制約に基づいて，どの演算器間でレジスタ間データ転送を利用すべきかを判断し，スケジューリング結果に反映させるスケジューリング手法を提案する．提案スケジューリング手法は，分岐処理に対応したスケジューリング手法である CVLS（Condition Vector List Scheduling）[10),11)] をベースとしているが，フィードバックされた配置結果から各演算器間のデータ転送に必要なクロック数を表すテーブルを作成し，それをスケジューリングの制約条件として加えることによって，レジスタ間データ転送を利用することが可能となり，スケジューリング結果のコントロールステップ数を短縮できる．さらに，レジスタ分散型に対応したレジスタバインディング手法を提案する．提案レジスタバインディング手法では，ローカルレジスタを演算器の入力側と出力側で区別し，どのコントロールステップでレジスタ間データ転送を実行するかに着目する．可能な限り出力側レジスタで変数を保持することによって，総レジスタ数の増加を抑えるように，レジスタ間データ転送を実行するコントロールステップを決定する．最後に，計算機実験によって提案手法の有効性を示す．計算機実験では配置のデッドスペースも含めて回路面積を評価する．

2. ターゲットアーキテクチャ

提案手法では，レジスタ分散型アーキテクチャ[4),5)]

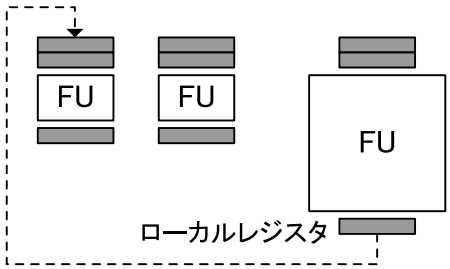

図 1 レジスタ分散型アーキテクチャ
Fig. 1 A distributed register architecture.

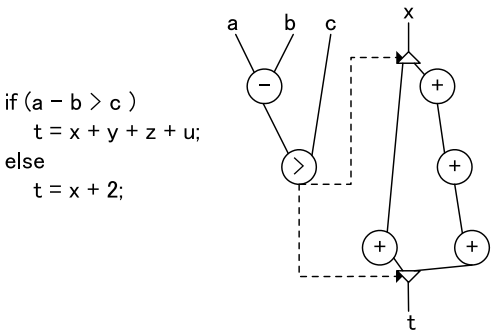

図 2 CDFG の例
Fig. 2 An example of CDFG.

をターゲットアーキテクチャとして用いる．

レジスタ分散型アーキテクチャは，**図 1** に示すように各演算器にローカルなレジスタを分散させて配置するアーキテクチャである．いずれの演算器に対しても，隣接した位置に専用のレジスタが配置されるため，演算器とレジスタ間の配線遅延が小さくなる．したがって，高位合成の過程で，演算器間の配線遅延や演算器とレジスタとの間の配線遅延をスケジューリング時のマージンとして大きく見積もっておく必要がなくなる．レジスタ分散型アーキテクチャの大きな特徴は，離れた位置に配置された演算器どうしでは，ローカルレジスタ間データ転送（図 1 の点線の矢印）を利用して，1 クロック（あるいは複数クロック）周期すべてをデータ転送の時間に割り当てることが可能だということである☆．

本論文の提案手法では，合成される回路の実行時間の最小化を目指すために，レジスタ分散型アーキテクチャを用いる．提案手法ではクロック周期制約を考慮して，どの演算器どうしのデータ転送にレジスタ間データ転送を使う必要があるかを判断し，スケジューリング時に制約条件として与える．

3. 配置情報のフィードバックを用いる合成フロー

文献 3) で取り上げられているような高位合成の基礎的な理論や，文献 10)，11) では，高位合成の段階ではモジュールの配置関係はまったく考慮されていなかった．その後，できるだけ近い演算器どうしでクリティカルパス上のデータ転送が実行されるようにモジュールを配置する手法[2),12)] が提案された．

前述のように，レジスタ分散型アーキテクチャを用いることによって，合成される回路の実行時間を従来手法[2),10)〜12)] よりも短縮させることが可能となると考えられるが，従来手法のようなアプローチはスケジューリングの段階で配線遅延情報を参照できないため，レジスタ分散型アーキテクチャでレジスタ間データ転送を用いる場合には適用するのが難しい．そこで，本論文では配置情報をスケジューリング工程にフィードバックして，どの演算器間でレジスタ間データ転送を用いるかを決定することが可能となる高位合成手法を提案する．

本論文において CDFG（Control Data Flow Graph）は，有向グラフ $G = (V, E)$ で表現され，ノード集合 V は，演算ノード集合 $N = \{n_i | i = 1, 2, \ldots, m\}$ と，分岐制御を表すフォークノード $(\triangle, \bigtriangledown)$ の集合 V_C を含むものとする．CDFG の例を**図 2** に示す．また，利用可能な演算器を表す演算器制約のリストとして，$F = \{f_i | i = 1, 2, \ldots, n\}$ が存在するものとする．各演算器は，面積および演算に要する遅延に関する情報と，何クロックで演算を実行するかというパラメータを持つ．クロック周期制約☆☆とは，任意の演算器 f の遅延が T_f で m クロックで演算を処理するときに，f がレジスタから入力を読み込み，出力をレジスタに格納するまでの時間 T_c が，クロック周期制約 T_{clk} に対して $T_c/m \leq T_{clk}$ の条件を満たさなければならないという制約である．

3.1 問題定義

本論文におけるレジスタ分散型アーキテクチャを対象とする高位合成の問題定義は，レジスタ分散型アーキテクチャを対象とした既存研究[4)] にならい，CDFG を入力とし，制約条件として演算器制約を与え，合成される回路の実行時間の最小化を目的関数とする．ただし，文献 4) ではクロック周期制約条件を考慮していないが，本論文ではクロック周期制約を問題の制約条

☆ レジスタ間データ転送を用いると，データ転送の間も転送元演算器が使用可能となる一種のパイプライン効果が得られる．

☆☆ 本論文では，クロック周期制約とタイミング制約を同一の意味で使う．

件として加える．クロック周期制約に対応することで，1 章でも述べたように，システム LSI の一部分の回路として回路を合成したい場合に，たとえ供給されるクロックの周波数が先に決まっていても，そのクロック周期制約を満足しながら実行時間を最小化した回路が合成できる．

したがって，本論文におけるレジスタ分散型アーキテクチャを対象とする高位合成問題とは，CDFG を入力とし，制約条件として演算器制約，クロック周期制約が与えられたときに，CDFG で表されたアプリケーションプログラムを実行可能な RT レベルのデータパスと制御回路，およびモジュールの配置情報を出力する問題である．目的関数はアプリケーションの実行時間の最小化である．実行時間が同一の場合，配置されたモジュールを囲む最小矩形の面積を最小化する．

3.2 合成フロー

高位合成フローとして，**図 3** に示すように，配線遅延情報をフィードバックする合成フローを提案する．

スケジューリング/FU（Functional Unit）バインディングの工程では，モジュール配置工程からフィードバックされた配線遅延情報を考慮し，レジスタ間データ転送を利用する CVLS ベースのスケジューリングを実行する．ただし，合成フローの初回は配線遅延情報を参照できないので，すべての演算器間の配線遅延が一定値であると仮定する．スケジューリングと同時に FU バインディングも実行される．スケジューリング/FU バインディングのアルゴリズムは 4 章で提案する．フィードバックする配線遅延情報の詳細についても，4 章で記述する．

レジスタバインディングの工程では，スケジューリング済みの CDFG から抽出される変数を，各演算器のローカルレジスタへバインディングする．どのコントロールステップでデータをローカルレジスタ間で転送するかに着目して，総レジスタ数の増加を抑えるようにバインディングを実行する．レジスタバインディングのアルゴリズムは 5 章で提案する．

概略モジュール配置の工程では，データ構造として Sequence-pair [7] を用い，モジュール配置を SA（Simulated Annealing）によって最適化する．SA のコスト関数は，A をデッドスペースを含む回路の総面積，W を各モジュールを結ぶ総配線長，V を各演算器間のデータ転送においてクロック周期制約条件を違反したディレイの総合計としたとき，

$$\alpha A + \beta W + \gamma V$$

と計算する[2]．ただし，α，β，γ は，任意のパラメータである．W と V がコスト関数に加えられることによって，データのやりとりが必要な演算器どうしが近くに配置されるようになり，合成される回路の実行時間が短縮される配置結果に近づいていく．モジュール配置に関しては，既存手法を適用しているので，詳細な説明は省略する☆．

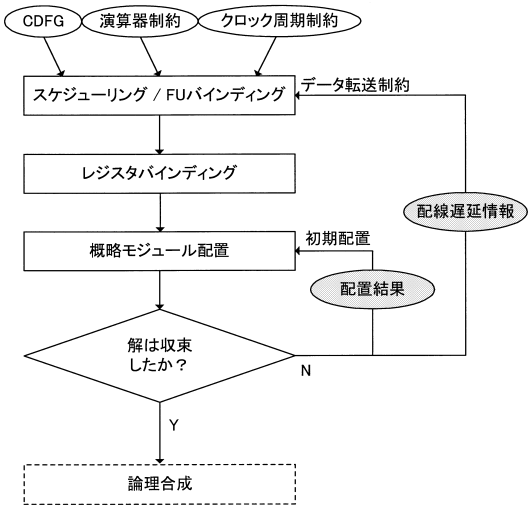

図 3 配置結果をフィードバックする高位合成フロー
Fig. 3 Our high-level synthesis flow.

また，概略モジュール配置工程では，i 回目のイタレーションにおける最終的な配置結果を，$i+1$ 回目のイタレーションのモジュール配置工程における SA の初期配置として用いる．ただし，毎回初期温度が高い状態から SA によるモジュール配置を実行すると，フィードバックされた配置情報が反映されなくなってしまうので，イタレーションごとに SA の初期温度を下げていく．具体的には，合成フローの i 回目のイタレーションのモジュール配置工程における SA の初期温度を T_i とすると，任意のパラメータ $K > 1$ を用いて，

$$T_{i+1} = T_i / K \tag{1}$$

とする（K は，2〜10，あるいは数 10 程度）．このような配置結果のフィードバックと，初期温度の設定手法を用いることによって，イタレーションの回数が少ない段階では，モジュール配置の自由度が高く，イタレーションの回数が増えるにしたがって，モジュール配置が固定されていくという合成過程を経ることができる．したがって，イタレーションの初期では，モジュール配置に関して広い解空間を探索することが可能であり，かつイタレーションの回数を重ねることに

☆ なお，本質的には，概略モジュール配置で異なるデータ構造と最適化手法を用いることも可能である．

よる解の収束性の保証ができる．

モジュール配置の工程が終了した後，回路の面積および実行時間が収束したと見なされなければ，スケジューリング/FU（Functional Unit）バインディングの工程へ配線遅延情報を，概略モジュール配置工程の工程へ配置結果をそれぞれフィードバックする．

4. レジスタ間データ転送を考慮したスケジューリング/FU バインディング

スケジューリングとは，CDFG の演算ノードに対して，エッジで表されたデータ依存関係を満たすように，コントロールステップを対応づけることである．また，FU バインディングとは，CDFG の演算ノードに対して，演算器制約リスト中の演算器を対応づけることである．ただし，同一のコントロールステップで 1 つの演算器が対応づけることのできる演算ノードは 1 つのみである．

2 章でも述べたように，レジスタ分散型アーキテクチャを用いることによって，演算器間でのデータ転送にはレジスタ間データ転送を利用でき，データ転送時にも転送元演算器が次のデータ入力に対する演算処理を開始することが可能となる．これによって，アプリケーションを処理するために必要なコントロールステップ数が減少し，合成される回路の実行時間も短縮される．また，演算器間の配線遅延を仮想的に大きく見積もっておく必要もなくなり，これも実行時間の短縮に寄与する．しかし，レジスタ間データ転送に対応したスケジューリングをするためには，どの演算器間で何クロックのレジスタ間データ転送を用いるかという情報がスケジューリングの段階から必要である．

レジスタ分散型アーキテクチャを用いる文献 4) ではスケジューリングの段階で配置情報を利用するアプローチを採用しているものの，クロック周期制約には対応しておらず，クロック周期は高位合成工程の最後の段階で最適な数値が選択される．クロック周期制約は 1 章でも述べたとおり，システム LSI の一部分としての回路を合成する際には必須事項と考えられる．

そこで，分岐処理に対応し，実用レベルで用いられているスケジューリング手法である CVLS [10),11)] をベースとし，クロック周期制約を与えることが可能で，かつフィードバックされた配線遅延情報に基づいたレジスタ間データ転送を利用可能とするスケジューリング手法を提案する．

4.1 準　　備

本論文における提案スケジューリング手法のベースとなっている CVLS [10),11)] で用いられる CV（Condition Vector）と FUV（Functional unit Utilization Vector）という概念の定義について説明する．

		CV (a)	CV (b)	CV (c)
x = a+b-c+d;	-(1)	[1,0,1]	[1,1,1]	[1,1,1]
if (a != 0)	-(2)	[1,1,1]	[1,1,1]	[1,1,1]
y = x+c;	-(3)	[1,0,0]	[1,0,0]	[1,1,1]
else if (a+b < c)	-(4)	[0,1,1]	[0,1,1]	[1,1,1]
y = c+d;	-(5)	[0,1,0]	[0,1,1]	[1,1,1]
else y = x+d;	-(6)	[0,0,1]	[0,1,1]	[1,1,1]
	a != 0 -(2)	done	done	not
	a+b < c -(4)	done	not	don't care

図 4 CV の例
Fig. 4 An example of CVs.

4.1.1 CV

CV は文献 10) で提案された概念であり，**図 4** のように各演算処理（演算ノード）に与えられるベクトルである．共通のビットに 1 が立っていない演算どうしは互いに排他的であるため，同じコントロールステップでも同じ演算器に割当て可能であるということを表す☆．CV は条件式の結果が利用可能かどうかにより動的に変化する．たとえば，(2) の結果が得られていない場合には CV は (c) の状態であるが，(2) の結果が得られた時点で CV は (b) の状態に遷移する．CV が (b) の状態では (3) と (4)，(3) と (5)，(3) と (6) の演算処理が互いに排他的である．

4.1.2 FUV

$\mathrm{FUV}_f(k)$ は，コントロールステップ k において FU f に割り当てられた演算の CV の合計である．たとえば，コントロールステップ 3 に CV が (a) の状態だったと仮定し，(3) と (5) の加算が演算器 ADD1 に割り当てられたとすると，$\mathrm{FUV}_{ADD1}(3) = [1,1,0]$ である．この状態からさらにコントロールステップ 3 で ADD1 に割り付けられるのは CV$= [0,0,1]$ の加算のみである．

4.2 レジスタ間データ転送に対応した CVLS

モジュールの配置関係を考慮し，レジスタ分散型アーキテクチャにおけるレジスタ間データ転送を利用可能とする CVLS ベースのスケジューリング/FU バインディング手法を提案する．

4.2.1 問 題 定 義

文献 10)，11) の CVLS では，CDFG，演算器制約，クロック周期制約を入力として，スケジューリング結果のコントロールステップ数の最小化を目的関数とするが，これに演算器間の配線遅延情報を入力として加えて，配線遅延情報に基づいたレジスタ間データ転送を用いることができれば，コントロールステップをよ

☆ 具体的な計算方法は文献 10) で詳述されている．

り少なくすることが可能になり，3 章に示した提案高位合成手法の合成フローから出力される回路の実行時間が短縮される．

したがって，本論文におけるスケジューリング/FU バインディング問題を次のように定義する．

CDFG，演算器制約，クロック周期制約，および演算器制約リストに含まれるすべての演算器間の配線遅延が与えられたときに，演算器制約とクロック周期制約，および CDFG のエッジで表されたノード間のデータ依存関係を満たすように CDFG の演算ノードにコントロールステップと演算器を対応づける，コントロールステップ数の最小化問題である．ただし，CV を用いて条件分岐に基づく排他性を利用できる場合，同一のコントロールステップで 1 つの演算器に複数の演算ノードを対応づけることが可能だとする．

4.2.2 スケジューリング/FU バインディングアルゴリズム

提案手法では，データ転送制約テーブル，クリティカルパス長リスト，レイテンシの見積りという概念を用いる．これらは Dynamic critical path scheduling [6] の考え方を応用したものであり，スケジューリングアルゴリズムの中で，演算器間のデータ転送に要するサイクル数を，スケジューリングの制約条件や，演算ノードの優先度に反映させるために必要な概念である．また，演算器の配置関係を考慮した FU バインディングを実現するためにも必要である☆．

4.2.2.1 データ転送制約テーブル

レジスタ間データ転送を利用するために，各演算器間のデータ転送に何クロック要するかを，モジュール配置工程からフィードバックされた配置結果をもとに，計算しておく．

具体的には，下記に示す計算方法で，任意の 2 個の演算器間でのデータ転送にともなう遅延がクロック周期内に収まるかを判定する．

t_{REG} をレジスタの読み出しおよび書き込みに要する時間，t_{CK} をクロック周期とする．演算器 f_i の遅延を c_i，f_i〜f_j 間の配線遅延を $d_{i,j}$ とし，

$$Slack_i = t_{CK} - t_{REG} - c_i$$

と計算するとき，$f_i \rightarrow f_j$ というデータ転送に必要なステップ数 $id_{i,j}$ を，

転送先

転送元	ADD1	SUB1	MUL1	MUL2
ADD1	0	0	0	1
SUB1	0	0	1	0
MUL1	1	1	0	0
MUL2	1	1	0	0

単位：クロック

図 5 データ転送制約テーブル
Fig. 5 A table of data transfer constraints.

$$id_{i,j} = \begin{cases} 0 & (Slack_i \geq d_{i,j}) \\ \lceil (d_{i,j}+t_{REG})/t_{CK} \rceil & (Slack_i < d_{i,j}) \end{cases}$$

と計算する．加算器 1 個，減算器 1 個，乗算器 2 個の場合の配線遅延制約テーブルの例を**図 5** に示す．注意すべきなのは，必ずしも $id_{i,j} = id_{j,i}$ ではないということである．c が小さい演算器ほど，演算と同一のステップでデータ転送を終えることができる範囲が広がる．

4.2.2.2 クリティカルパス長リスト

クリティカルパス長リスト（cp リスト）は，演算ノードを各演算器に割り当てた場合のクリティカルパス長を計算したものである．このクリティカルパス長がリストスケジューリングの優先度として用いられる．通常の演算ノードについては文献 4) で用いられている計算方法に準じた方法を用いることができ，ノード n_i を演算器 f_j に割り当てた場合の $cp_{i,j}$ は次式で計算される．ただし，c_j は f_j の演算処理時間であり，$succ_i$ は n_i のすべての子孫である．

$$cp_{i,j} = c_j + \max_{n_k \in succ_i} \{\min_l (id_{j,l} + cp_{k,l})\}$$

なお，分岐を表すフォークノード (△) については演算器を割り当てるという概念が存在しないうえ，必ずしも子孫のうち最大のクリティカルパス長を持つ演算ノードが実行されるとは限らない．そこで，n_i の隣接した子孫であるフォークノード群を $succ.fork_i$ と定義し，$n_f \in succ.fork_i$ を介して直接的に n_i からデータ転送があるノード群を $succ.fork.op_f$ とする．それぞれのノード $n_k \in succ.fork.op_f$ に分岐する確率を p_k とし，$cpf_{i,j}$ という関数を

$$cpf_{i,j} = \max_{n_f \in succ.fork_i} \left[\sum_{n_k \in succ.fork.op_f} \{p_k \times \min_l (id_{j,l} + cp_{k,l})\} \right]$$

☆ 文献 4) のアプローチと似ているが，文献 4) がスケジューリング後にクロック周期を決定するのに対し，本論文のアプローチはクロック周期制約を考慮してデータ転送制約をサイクル単位でスケジューリング前に決定する点が異なる．

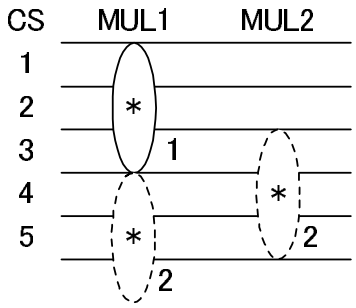

図 6 レイテンシ見積りの必要性
Fig. 6 Necessity of latency estimates.

と計算することにする．ここで，$succ.op_i$ を n_i の子孫のうちの演算ノード群とすると，条件分岐を含む CDFG の $cp_{i,j}$ は次のように一般化される．

$$cp_{i,j} = c_j + \max\left[\max_{n_k \in succ.op_i}\{\min_l(id_{j,l}+cp_{k,l})\},\quad cpf_{i,j}\right]$$

4.2.2.3 レイテンシの見積り

レジスタ間データ転送を用いる場合，同じ種類の演算でも割り当てる演算器によって最終的に必要となるコントロールステップ数が変わってしまう可能性がある．なぜならば，演算器どうしの位置関係によってデータ転送に費やされる時間が異なってくるからである．つまり，同種類の演算器でもまったく同等に扱うということはできない．

たとえば，**図 6** に示すように，コントロールステップ 1 で $*1$ が乗算器 1 にバインディングされたとする．その後，コントロールステップ 3 で $*2$ がレディーリストに入ったとき，通常の CVLS の手順を踏むと，$*2$ は乗算器 2 にバインディングされてしまう．しかし，

$$cp_{*2,\,MUL2} - cp_{*2,\,MUL1} \geq 2$$

である場合，$*2$ は乗算器 2 にバインディングするよりも，コントロールステップ 4 まで待って乗算器 1 にバインディングするべきだと考えられる．

提案アルゴリズムでは，ノード n_i が演算器 f_j にバインディング可能になるコントロールステップ（演算器 f_j に n_i 以外の演算ノードが割り当てられているリソース競合の状態と，f_j へのデータ転送制約を考慮する）を，コントロールステップ k の時点で判断した値を $abl_{i,j,k}$ として，

$$lat_{i,j,k} = abl_{i,j,k} + cp_{i,j}$$

で計算されるレイテンシの見積りの値を用いる．この $lat_{i,j,k}$ をバインディング前に計算し，どの演算器にバインディングするべきかという判断基準として用いる．

4.2.2.4 アルゴリズム

提案スケジューリング手法のアルゴリズムを**図 7** に示す．提案スケジューリング手法により，CDFG の条件分岐，およびクロック周期制約に対応し，かつ適切にレジスタ間転送データ転送を利用したスケジューリングを実現できる．また，スケジューリングと同時に演算器への演算ノードのバインディングも決定されるのも特徴である．

手順 1. 配線遅延テーブルを作成する．配線遅延テーブルを使ってすべての演算ノードに対してクリティカルパス長のリスト（cp リスト）を作成し，各演算ノード n_i に対して $\min_j(cp_{i,j})$ を優先度として設定する．

手順 2. コントロールステップ k を 0 に設定する．

手順 3. すべての演算ノードがスケジューリング済みの場合，アルゴリズムを停止する．k に 1 を加え，レディーリストを作成する．

手順 4. 最も優先度が高いノード n_i を選択し，CV を計算のうえ，cp リストを基にレイテンシの見積り $lat_{j,j,k}$ が最小となるような演算器 f_j を選択する．n_i が f_j にコントロールステップ k でバインディング可能ならば，n_i を f_j にバインディングし，f_j の FUV を更新する．レディーリストから n_i を削除する．

手順 5. レディーリストが空でない場合，手順 4 に戻る．レディーリストが空となった場合，手順 3 に戻る．

図 7 提案スケジューリングアルゴリズム
Fig. 7 The proposed scheduling algorithm.

5. 分散型レジスタバインディング

スケジューリング済みの CDFG のエッジは，アプリケーションプログラムにおける変数に対応している．レジスタバインディングとは，スケジューリング済み CDFG のエッジに対して，レジスタを対応づけることである．ただし，同一のコントロールステップにおいては 1 つのレジスタには 1 つの変数しか割り当てることはできない．

文献 4), 5) では，レジスタ分散型アーキテクチャが提案されているが，レジスタ分散型に対応したレジスタバインディングの手法について言及されていない．そこで，本論文ではレジスタ分散型アーキテクチャに対応するレジスタバインディング手法を提案する．提案バインディング手法によってレジスタ分散型アーキテクチャに対してのレジスタバインディングが可能となり，高位合成の解として，完全な RT レベルの回路記述を得ることができるようになる．また，レジスタの面積も含めた回路面積の見積りおよび評価が可能となる．

5.1 問題定義

分散型レジスタバインディングについて，**図 8** の CDFG を例にとって説明する．図 8 は演算ノードが 5 つの CDFG に関して，加算器，減算器，比較器が

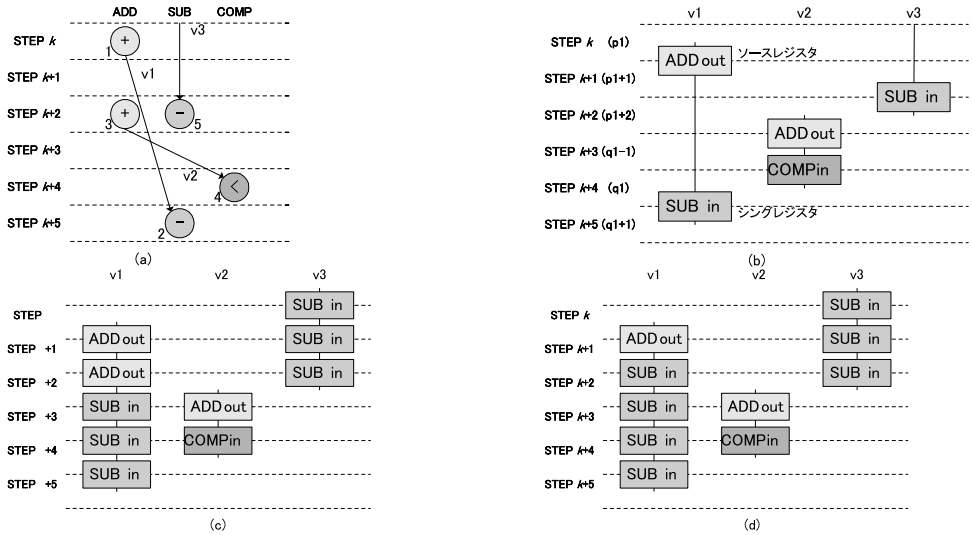

図 8 分散型レジスタバインディング：(a) DFG のスケジューリング結果の例，(b) 初期バインディング結果，(c) 減算器の入力側レジスタが 1 個になる例，(d) 減算器の入力側レジスタが 2 個必要な例

Fig. 8 Distributed register binding: (a) An example of scheduled DFG, (b) The result of Initial binding, (c) The example of binding result that uses one input register for SUB, (d) The example of binding result that uses two input register for SUB.

利用可能な演算器として存在すると考えた場合のスケジューリング結果である（たとえば，ノード 3 の加算は，STEP$k+2$ で加算器（ADD）にバインディングされている）．スケジューリング済み CDFG の 1 つのエッジはアプリケーションの 1 つの変数に対応している．それぞれの変数は開始のコントロールステップと終了のコントロールステップを持つ．すなわち，すべての変数は開始から終了というコントロールステップ単位のライフタイムを持つ．

演算のスケジューリングが終了した時点で，レジスタへのバインディングの一部はすでに確定していると考えることができる．各演算器間のデータ転送に 1 クロック必要だと仮定すると，少なくとも図 8 (b) に示すように変数は開始ステップではソース側の演算器のローカルレジスタ，終了ステップではシンク側のローカルレジスタへ割り当てられていなければ，図 8 (a) のスケジューリング結果を満足することができない．この制約を満足したバインディング結果を初期バインディング結果と呼ぶことにする．ただし，ADD out，SUB in という表記は，それぞれ加算器の出力側ローカルレジスタ，減算器の入力側ローカルレジスタを表現していて，図 8 (b) の例では，変数 v_1 は STEPk〜STEP$k+1$ にかけて加算器の出力側レジスタにバインディングされていることを表している☆．

レジスタ分散型アーキテクチャでは，図 8 (b) の初期バインディング結果を満足したまま，どのコントロールステップでソース側のレジスタからシンク側のレジスタへデータを転送するかを決定するアルゴリズムが必要となる．ここで，データ転送を実行するコントロールステップを決定する際に，CDFG のスケジューリング結果に変更を加えることなく，総レジスタ数が最小であるようなレジスタバインディング結果が得られれば，同一の実行時間のスケジューリング結果に対して，より小さい面積の回路が得られると考えられる．つまり，3 章の提案高位合成手法の合成フローの出力に関して，アプリケーション実行時間が同一の解に対しての面積の最小化という効果が得られる．

したがって，本論文では分散型レジスタバインディング問題を，スケジューリングおよび FU バインディング済みの CDFG が与えられたときに，CDFG のすべてのエッジに対して，初期バインディング結果で開始ステップに割り当てられたレジスタから，終了ステップに割り当てられたレジスタにデータを転送するコントロールステップを決定する，総レジスタ数最小

☆ このように，提案手法では，ローカルレジスタを演算器の入力側ポートと出力側ポートで区別する．

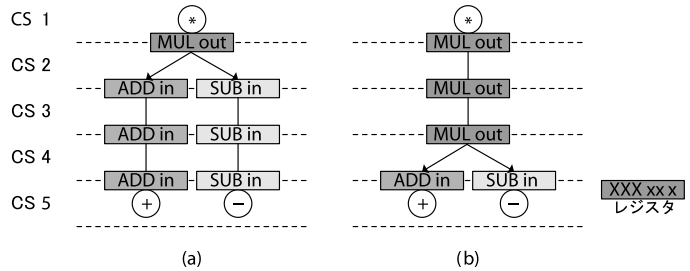

図 9 レジスタバインディング：(a) CS2 でデータ転送した場合，(b) CS4 でデータ転送した場合
Fig. 9 An register binding result: (a) The case where the data is transfered at CS2, (b) The case where the data is transfered at CS4.

化問題であると定義する．

5.2 レジスタバインディングアルゴリズム

図 8 (c) は，STEP$k+2$ において加算器の出力側レジスタから減算器の入力側レジスタへデータを転送した例である．図 8 (d) のように可能な限り早いステップ（STEP$k+1$）でデータ転送を実行するというアプローチもありうるが，STEP$k+1$〜STEP$k+2$ にかけて SUB in が競合してしまい，結果として減算器のローカルレジスタ数を増やす必要が生じる．このように，できるだけ早くデータ転送を実行するよりも，可能な限り出力側（ソース側）のレジスタで変数を保持したほうが，全体のレジスタ数は少なくなると考えられる．

提案レジスタバインディング手法では，最初は入力側も出力側もレジスタの数を固定と考え，可能な限り出力側のレジスタで変数を保持するように変数をバインディングする．しかし，変数の競合のためにレジスタ数を増やす必要が生じる場合には，出力側のローカルレジスタを増やし，再度割当てを試みる．出力側のレジスタ数を増やすのは，入力側のレジスタを増やして割り当てることにすると，**図 9** のように転送元演算器から複数の演算器にデータを転送する場合，データを転送してしまうと同じ変数が複数のレジスタに格納されてしまうので［図 9 (a)］，出力側でより長く保持できるようにするためである［図 9 (b)］．

提案バインディング手法の手順を**図 10** に示す．利用可能な演算器のリスト $F=\{f_1, f_2, \dots\}$ に対して，各 f_i に対応した入力ポート数 k_i および出力ポート数 l_i が存在するものとした．また，変数 v_i のソースとなっている演算器を $f_{so(v_i)}$，シンクとなっている演算器を $f_{si(v_i)}$ と表すことにした．

手順 1. スケジューリング済みの CDFG から，変数のリスト $L=\{v_1, v_2, \dots\}$ を抽出する．すべての演算器 $f_i \in F$ に関して，k_i に f_i の入力ポートの数，l_i に f_i の出力ポートの数を代入する．

手順 2. すべての変数 $v_i \in L$ についてライフタイムを解析し，p_i にライフタイムの開始ステップ，q_i にライフタイムの終了ステップを代入する．すべての変数 $v_i \in L$ について，STEPp_i〜STEPp_i+1 において v_i を $f_{so(v_i)}$ の入力側レジスタに割り当て，STEPq_i〜STEPq_i+1 において，v_i を $f_{si(v_i)}$ の出力側レジスタに割り当てる．

手順 3. 任意の変数 $v_i \in L$ を選択する．もし，$L=\phi$ であれば手順 7 へ進む．$m=1$，$n=1$ とする．

手順 4. STEPp_i+m〜STEPp_i+m+1 において，$f_{so(v_i)}$ の出力側レジスタに割り当てられている変数の数が $l_{so(v_i)}$ 個未満であれば，v_i を $f_{so(v_i)}$ の出力側レジスタに割り当て，m に 1 を加え，手順 4 を繰り返す．割当て不可能ならば $p_i=p_i+m-1$ として，手順 5 へ進む．

手順 5. STEPq_i-n〜STEPq_i-n+1 において，$f_{si(v_i)}$ の入力側レジスタに割り当てられている変数の数が $k_{si(v_i)}$ 個未満であれば，v_i を $f_{si(v_i)}$ の入力側レジスタに割り当て，n に 1 を加える．手順 5 を繰り返す．割当て不可能ならば $q_i=q_i-n+1$ として，手順 6 へ進む．

手順 6. L から v_i を削除する．手順 3 へ戻る．

手順 7. 未割当てのステップを持つ変数のみで，新たに変数のリスト L を作成する．$L=\phi$ であれば（すべての変数の全ステップがローカルレジスタに割り当てられていれば），処理を終了する．そうでなければ，すべての l に f の現在の出力側レジスタの数に 1 加えた数を代入する．手順 3 に戻る．

図 10 分散型レジスタバインディングアルゴリズム
Fig. 10 A distributed register binding algorithm.

6. 計算機実験

提案スケジューリング手法，提案レジスタバインディング手法，および概略モジュール配置の工程を C 言語を用いて計算機上に実装した．計算機実験環境は，OS が VineLinux Version 2.0，CPU が Intel PentiumIII 850 MHz，メモリ容量が 256 MB，C コンパイラが gcc(egcs-1.1.2) である．計算機実験の総面積は，演算器，レジスタ，MUX，制御回路の面積を含む値であり，配置のデッドスペースも含んだ評価となっている．

表 1 計算機実験結果（FIR）
Table 1 Experimental results (FIR).

	演算器制約	総面積 [μm^2]	FU 部 [μm^2]	REG 部 [μm^2]	MUX 部 [μm^2]	制御回路部 [μm^2]	実行時間 [ns]	状態数	CPU 時間 [s]
(1) 従来手法[10),11)]	+2, *3	1,810,056	1,121,362	189,968	237,808	37,321	122.5	49	40.7
	+2, *4	2,112,488	1,478,310	196,096	208,416	31,258	97.5	39	53.1
	+3, *6	3,132,776	2,217,465	245,120	277,888	35,515	72.5	29	161.9
(2) 提案手法	+2, *3	1,735,360	1,121,362	196,096	221,776	40,308	100.0	40	156.1
	+2, *4	2,110,880	1,478,310	232,864	253,840	38,776	92.5	37	261.3
	+3, *6	2,937,510	2,217,465	257,376	278,055	45,321	70.0	28	557.9

表 2 計算機実験結果（DCT）
Table 2 Experimental results (DCT).

	演算器制約	総面積 [μm^2]	FU 部 [μm^2]	REG 部 [μm^2]	MUX 部 [μm^2]	制御回路部 [μm^2]	実行時間 [ns]	状態数	CPU 時間 [s]
(1) 従来手法[10),11)]	+3, *3	1,617,903	1,146,621	147,072	176,352	28,343	70.0	28	45.7
(2) 提案手法	+3, *3	1,638,175	1,146,621	153,200	197,728	26,140	55.0	22	203.0

対象アプリケーションとしては 7 次 FIR フィルタ（節点数 82），および DCT（節点数 47）を用いた．FIR フィルタに関して提案手法と従来手法[10),11)] による合成結果を比較した結果を**表 1** に示す．DCT に関しての比較結果を**表 2** に示す．ターゲットアーキテクチャとしては，レジスタ分散型を用いた．演算器は 16 bit 幅と仮定し，面積/遅延は VDEC ライブラリ（CMOS0.35 μm テクノロジ）☆をもとに，あらかじめ合成して得られた値を用いた．乗算器の面積，遅延をそれぞれ 356,948 [μm^2]，5.71 [ns] とし，加算器の面積，遅延をそれぞれ 25,259 [μm^2]，1.44 [ns] とした．また，1 ビットレジスタの面積，遅延をそれぞれ 383 [μm^2]，0.40 [ns]，2-1 マルチプレクサの面積，遅延をそれぞれ 167 [μm^2]，0.23 [ns] とした．コントローラの面積は Synopsys 社の Design Compiler により実際に論理合成して求めることにした．ただし，配線遅延に関しては配線長の 2 乗に比例すると仮定し，1,000 [μm] あたり 1 [ns] と設定した．各演算器の入出力ポートはモジュールの中心にあると仮定した．クロック周期制約は 2.5 [ns] とした．式 (1) のパラメータは $K = 10$ と設定した．

また，表 1 および 表 2 中の FU 部，REG 部，MUX 部，制御回路部とは，それぞれ演算器，レジスタ，マルチプレクサ，コントローラが回路に占める面積を示している☆☆．実行時間とは，対象アプリケーションの実行に要する時間（クロック周期と，実行に要するコントロールステップ数の積）を表している．

FIR の例題入力に対しては，いずれの演算器制約においても面積が従来手法よりも小さくなり，実行時間が短縮（最大で 19%）された．また，DCT を例題入力とした場合，従来手法に比べて面積は 1%増加したものの，実行時間は 21%短縮された．これは，提案手法により配線遅延情報を考慮して，クロック周期制約に応じた適切なレジスタ間データ転送を利用できるようになった効果によるものである．なお，提案手法の効果が面積の削減として得られる場合があるのが確認できるのは，提案手法が，どのようなモジュール配置結果に対しても，配線遅延制約テーブルをフィードバックして，その配置結果から得られるデータ転送制約条件を満たすようにスケジューリングし直せることに起因している．従来手法では遅延制約条件に違反してしまうような配置結果でも，提案手法では利用可能であるので，よりデッドスペースの小さい配置結果が得られる場合がある．

7. むすび

本論文では，レジスタ分散型アーキテクチャをターゲットとし，フロアプラン情報のフィードバックをともなう合成フローを用いた高位合成手法を提案した．また，フロアプランとタイミング制約に基づいたレジスタ間データ転送を利用可能とするスケジューリング手法，およびレジスタ分散型アーキテクチャに対応したレジスタバインディング手法を提案した．

提案手法により，合成に要する時間は増加するものの，演算器の配置情報を考慮して，タイミング制約に基づいたレジスタ間データ転送を利用したスケジュー

☆ VDEC 日立ライブラリは東京大学大規模集積システム設計教育研究センターを通し株式会社日立製作所および大日本印刷株式会社の協力で作成されたものである．

☆☆ FU 部，REG 部，MUX 部，制御回路部の面積を合計しても総面積の値に満たないのは，総面積はそれらのモジュールの面積だけではなく，デッドスペースの面積も加えられているからである．

リングが可能となる．従来手法と比較して実行速度，あるいは面積に関して優れた回路を合成することができるようになった．また，従来の研究では言及されていなかった，レジスタ分散型アーキテクチャに対応したレジスタバインディングアルゴリズムの提案，および配置のデッドスペースを考慮した実験結果の提示をし，より精度の高い評価によって提案手法の有効性を示すことができた．

今後の課題としては，ローカルレジスタと共有レジスタの併用型をターゲットアーキテクチャとする手法の検討があげられる．

謝辞 本研究を進めるにあたり，多くの有益なご助言，ご討論をいただきました，日本電気株式会社システムデバイス研究所粟島亨氏に深く感謝いたします．本研究の一部は早稲田大学特定課題研究助成費（課題番号：2003A-575）によるものである．

参 考 文 献

1) Cong, J., Fan, Y., Han, G., Yang, X. and Zhang, Z.: Architectural synthesis integrated with global placement for multi-cycle communication, *Proc. ICCAD 2003*, pp.536–543 (2003).
2) Fang, Y.M. and Wong, D.F.: Simultaneous functional-unit binding and floorplanning, *Proc. ICCAD 1994*, pp.317–321 (1994).
3) Gajski, D.D., Dutt, N.D., Wu, A.C.-H. and Lin, S.Y.-L.: *High-level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).
4) Jeon, J., Kim, D., Shin, D. and Choi, K.: High-level synthesis under multi-cycle interconnect delay, *Proc. ASP-DAC 2001*, pp.662–667 (2001).
5) Kim, D., Jung, J., Lee, S., Jeon, J. and Choi, K.: Behavior-to-placed RTL synthesis with performance-driven placement, *Proc. ICCAD 2001*, pp.320–325 (2001).
6) Kwok, Y. and Ahmad, I.: Dynamic critical-path schedule: An effective technique for allocating task graphs to multiprocessors, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.5, pp.506–521 (1996).
7) Murata, H., Fujiyosho, K. and Nakatake, S.: Rectangle-packing-based module placement, *Proc. ICCAD 1995*, pp.472–479 (1995).
8) Tarafdar, S., Leeser, M. and Yin, Z.: Integrating floorplanning in data-transfer based high-level synthesis, *Proc. ICCAD 1998*, pp.412–417 (1998).
9) Um, J., Kim, J. and Kim, T.: Layout-driven resource sharing in high-level synthesis, *Proc. ICCAD 2002*, pp.614–618 (2002).
10) Wakabayashi, K. and Yoshimura, T.: A resource sharing and control synthesis method for conditional branches, *Proc. ICCAD 1989*, pp.62–65 (1989).
11) Wakabayashi, K. and Tanaka, H.: Global scheduling indepandent of control dependencies based on condition vectors, *Proc. 29th ACM/IEEE DAC 1992*, pp.112–115 (1992).
12) Weng, J.P. and Parker, A.C.: 3D scheduling: High-level synthesis with floorplanning, *Proc. 28th ACM/IEEE DAC 1992*, pp.668–673 (1991).
13) Xu, M. and Kurdahi, F.J.: Layout-driven high level synthesis for FPGA based architectures, *Proc. DATE '98*, pp.446–450 (1998).

(平成 16 年 11 月 18 日受付)
(平成 17 年 3 月 1 日採録)

田中 真

2003 年早稲田大学理工学部電子・情報通信学科卒業．2005 年同大学大学院修士課程修了．現在，(株) 野村総合研究所．システム LSI 設計，特に下位工程を考慮したシステム LSI 高位合成に関する研究に従事．

内田 純平

2001 年早稲田大学理工学部電子・情報通信学科卒業．2004 年同大学大学院修士課程修了．現在，同博士後期課程．システム LSI の設計および検証，特に高位合成に関する研究に従事．IEEE 学生会員．

宮岡祐一郎（正会員）

2000 年早稲田大学理工学部電子・情報通信学科卒業．2002 年同大学大学院修士課程修了．2005 年同博士後期課程修了．博士（工学）．現在，(株) 東芝．VLSI 設計，特にアプリケーションに特化したプロセッサの合成に関する研究に従事．IEEE，電子情報通信学会各会員．

戸川　望（正会員）

1992 年早稲田大学理工学部電子通信学科卒業．1994 年同大学大学院修士課程修了．1997 年同博士後期課程修了．博士（工学）．早稲田大学理工学総合研究センター講師，北九州市立大学国際環境工学部情報メディア工学科助教授を経て，現在早稲田大学理工学部コンピュータ・ネットワーク工学科助教授．VLSI 設計，計算幾何学，グラフ理論等の研究に従事．1996 年第 9 回安藤博記念学術奨励賞受賞．1997 年度（第 21 回）丹羽記念賞受賞．IEEE，電子情報通信学会会員．

柳澤　政生（正会員）

1981 年早稲田大学理工学部電子通信学科卒業．1983 年同大学大学院博士前期課程修了．1986 年同博士後期課程修了．博士（工学）．米国カリフォルニア大学バークレー校研究員，拓殖大学工学部情報工学科助教授を経て，現在早稲田大学理工学部コンピュータ・ネットワーク工学科教授．電子回路の設計自動化，ノイズ解析，計算幾何学，グラフ理論，バイオインフォマティクス等の研究に従事．1987 年度丹羽記念賞受賞．1990 年安藤博学術奨励賞受賞．IEEE，ACM，電子情報通信学会，プリント回路学会，日本 OR 学会各会員．

大附　辰夫（正会員）

1963 年早稲田大学理工学部電気通信学科卒業．1965 年同大学大学院修士課程修了．同年日本電気（株）入社．1980 年同退社．現在，早稲田大学理工学部コンピュータ・ネットワーク工学科教授．博士（工学）．システム LSI およびこれに関連した基礎研究に従事．1969 年度電子情報通信学会論文賞受賞．1994 年度第 32 回電子情報通信学会業績賞受賞．IEEE CAS Society より Guillmin-Cauer Prize Award（1974 年），Meritorious Service Award（1995 年），Golden Jubilee Medal（2000 年）受賞．2000 年 IEEE より 3rd Millennium Medal 受賞．共著『VLSI の設計 I』（岩波書店），編共著『Layout Design and Verification』（North-Holland）．IEEE フェロー，電子情報通信学会フェロー，電気学会，プリント回路学会各会員．