社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2004－SLDM－117（34）
2004／12／2

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[†]

† 早稲田大学理工学部コンピュータ・ネットワーク工学科
†† 北九州市立大学国際環境工学部情報メディア工学科
††† 早稲田大学理工学総合研究センター

〒169-8555 東京都新宿区大久保 3-4-1
TEL:03-5286-3396, Fax: 03-3203-9184
E-mail: a.tanaka@ohtsuki.comm.waseda.ac.jp

あらまし　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が，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．本稿では，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．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．また，計算機実験によって，提案手法の有効性を示す．

キーワード　高位合成，フロアプラニング，レジスタ分散型アーキテクチャ，ディープサブミクロンプロセス，配線遅延

A High-level Synthesis Algorithm with Floorplaning for a Distributed-register Architecture

Akira TANAKA[†], Jumpei UCHIDA[†], Yuichiro MIYAOKA[†], Nozomu TOGAWA[††,†††], Masao YANAGISAWA[†], and Tatsuo OHTSUKI[†]

† Department of Computer Science, Waseda University
†† Department of Information and Media Science, The University of Kitakyushu
††† Advanced Research Institute for Science and Engineering, Waseda University

3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan
Tel: +81-3-5286-3396, Fax: +81-3-3203-9184
E-mail: a.tanaka@ohtsuki.comm.waseda.ac.jp

Abstract　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 repeat (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. We show effectiveness of the proposed methods through experimental results.

Key words　High-level Synthesis, Floorplaning, Distributed-register architecture, Deep sub-micron process, Interconnect delay

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極めて有効な手段である．従来型の高位合成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，モジュール間の配置関係や配線遅延情報を，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伴い，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．

従来のフロアプランを考慮した高位合成手法の研究[2][7][11]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が用いられていた．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，[3][4]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では，各演算器は近い位置に配置されたローカルレジスタとデータをやり取りをするため，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するという大きな特徴を持つ．

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[3]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[4]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点を持つ．配置問題を線形計画問題に帰着させる手法[8]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，手法[8]では，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．その他にも，研究[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関わらず，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．

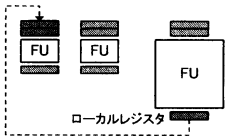

図1　レジスタ分散型アーキテクチャ

そこで，本稿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）[9][10]をベースとしている．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，ローカルレジスタ間のデータ転送を実行するコントロールステップを，総レジスタ数の増加を抑えるように決定する．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[3][4]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．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同士では，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）周期全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(注1)．

本稿の提案手法ではレジスタ分散型アーキテクチャを用いるが，クロック周期制約を考慮して，どの演算器同士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. 合成フロー

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，スケジューリング段階で各演算器間の配線遅延情報が必要となる．従って，従来用いられてきた高位合成手法のように，モジュール配置工程を単純に高位合成の後工程とすることはできない．

(注1)：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．

そこで，本稿では図2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とFUバインディングを同時に実行する．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，全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 / 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 [6] 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．コスト関数に関しては，既存手法を適用しているので，詳細な説明は省略する．

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，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 としたときに，任意のパラメータ $K > 1$ を用いて，

$$T_{i+1} = T_i / K \qquad (1)$$

とする(注2)．このような初期温度設定手法を用いることによって，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，回路の面積及び実行時間が収束したとみなされなければ，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．

(注2)：K は，2〜10．あるいは数10など，比較的大きい値．

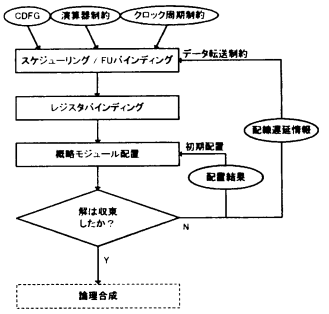

図2 配置結果をフィードバックする高位合成フロー

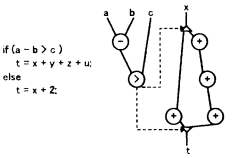

図3 CDFGの例

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

従来のフロアプラン考慮型高位合成手法 [2] [11] では，スケジューリングの段階で配置情報を考慮することはできなかった．レジスタ分散型アーキテクチャを用いる [3] 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．

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 [9] [10] 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．

ただし，本稿において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，分岐制御を表わすフォークノード（△，▽）の集合 V_C を含むものとする．CDFGの例を図3に示す．また，利用可能な演算器のリストとして，$F = \{f_i | i = 1, 2, \ldots, n\}$ が存在するものとする．

4.1 準　　備

本稿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 [9] [10] 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．

a） CV

CVは [9] 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同士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り当て可能であるとい

		CV (a)	(b)	(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

転送元＼転送先	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

うことを表す(注3). 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. 例えば, (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.

b) FUV

$FUV_f(k)$ 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. 例えば, 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, $FUV_{ADD1}(3) = [1,1,0]$ 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手法を提案する. 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 [5] 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.

c) データ転送制約テーブル

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, 下記に示す計算方法で, あらかじめ計算しておく.

t_{REG} をレジスタの読み出し及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,

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

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4).

d) 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 [3] 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 の全ての子孫である.

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

なお, 分岐を表わすフォークノード (△) 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上, 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

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

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.

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

e) 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, 演算器同士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.

例えば, 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,

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

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

(注3) : 具体的な計算方法は [9] で詳述されている.
(注4) : 必ずしも $id_{i,j} = id_{j,i}$ ではない.

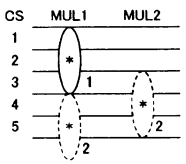

図 6　レイテンシ見積もりの必要性

手順 1. 配線遅延テーブルを使って，全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. 全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 を計算の上，cp リストを基にレイテンシの見積もり $lat_{j,j,k}$ 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無い場合，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

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もりの値を用いる．この $lat_{j,j,k}$ 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．

f)　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，及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．

5.　分散型レジスタバインディング

[3] [4] 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，本稿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．

レジスタバインディングについて，図 8 の DFG を例にとって説明する．図 8 は演算ノードが 5 つの DFG に関して，加算器，減算器，比較器が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（例えば，ノード 3 の加算は，STEP$k+2$ で加算器にバインディングされている）．ここで，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) に示すように変数が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．ただし，ADD out，SUB in と

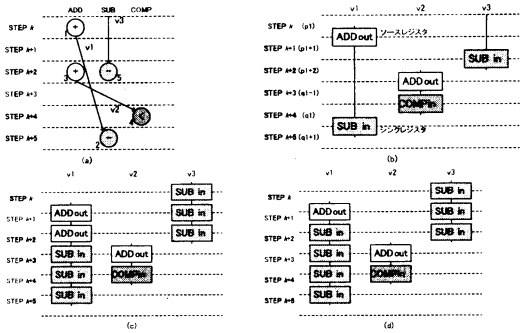

図 8　分散型レジスタバインディング：(a) スケジューリング結果の例，(b) 初期バインディング結果，(c) 減算器の入力側レジスタが 1 個になる例，(d) 減算器の入力側レジスタが 2 個必要な例

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(注5)．

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．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可能だが，STEP$k+1$〜STEP$k+2$ にかけて SUB in が競合してしまい，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，再度割り当てを試みる．

提案バインディング手法の手順を図 9 に示す．利用可能な演算器のリスト $F = \{f_1, f_2, \ldots\}$ に対して，各 f_i に対応した入力ポート数 k_i 及び出力ポート数 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．

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，及び概略モジュール配置の工程のシミュレーションプログラムを 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 850MHz，メモリ容量が 256MB，C コンパイラが gcc(egcs-1.1.2) である．

(注5)：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．

手順 1. スケジューリング済みの CDFG から，変数のリスト $L = \{v_1, v_2, \ldots\}$ を抽出する．全ての演算器 $f_i \in F$ 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. 全ての変数 $v_i \in L$ 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．全ての変数 $v_i \in L$ 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. STEP$p_i + m$～STEP$p_i + m + 1$ 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．割り当て不可能ならば $p_i = p_i + m - 1$ として，手順 5 へ進む．

手順 5. STEP$q_i - n$～STEP$q_i - n + 1$ 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．割り当て不可能ならば $q_i = q_i - n + 1$ 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. 未割り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（全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，全ての 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 9　提案レジスタバインディングアルゴリズム

実験結果の面積は，演算器，レジスタ，MUX，コントローラ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．

対象アプリケーションとして 7 次 FIR フィルタ，および DCT を用い，提案手法と従来手法 [9] [10] による合成結果を比較した [表 1]．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 16bit 幅と仮定し，面積/遅延は VDEC ライブラリ（CMOS0.35μm テクノロジー）(注6)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 356948[μm^2]，5.71[ns] とし，加算器の面積，遅延をそれぞれ 25259[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，1000[μm] 当たり 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．(1) 式のパラメータは $K = 10$ と設定した．また，表 1 中の実行時間とは，クロック周期と，対象アプリケーションの実行に要するコントロールステップ数との積を表わしている．CPU 時間は，シミュレーションプログラムの実行に要した時間，すなわち合成に要する時間を表わす．

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レジスタ間データ転送を利用可能となった効果によるものである．なお，提案手法により面積が削減される場合がある．これは，デッドスペースが少ないが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，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ためである．

(注6)：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．

表 1　シミュレーション実験結果

		(1) 従来手法 [9] [10]			(2) 提案手法		
例題入力	演算器制約	面積 [μm^2]	実行時間 [ns]	CPU 時間 [s]	面積 [μm^2]	実行時間 [ns]	CPU 時間 [s]
FIR	+2, *3	1810056	122.5	40.7	1735360	100.0	156.1
	+2, *4	2112488	97.5	53.1	2110880	92.5	261.3
	+3, *6	3132776	72.5	161.9	2937510	70.0	557.9
DCT	+3, *3	1617903	70.0	45.7	1638175	55.0	203.0

7. むすび

本稿では，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高位合成フロー，スケジューリング手法，及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．今後の課題としては，現実のデバイス上での提案手法の評価や，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挙げられる．

文　献

[1] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architectural synthesis integrated with global placement for multi-cycle communication, " in *Proc. ICCAD 2003*, pp. 536–543, 2003.

[2] Y. M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning, " in *Proc. ICCAD 1994*, pp. 317–321, 1994.

[3] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay, " in *Proc. ASP-DAC 2001*, pp. 662–667, 2001.

[4] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement, " in *Proc. ICCAD 2001*, pp. 320–325, 2001.

[5] Y. Kwok and I. Ahmad, "Dynamic critical-path schedule: an effective technique for allocating task graphs to multiprocessors, " in *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, 1996.

[6] H. Murata, K, Fujiyosho, and S.Nakatake, "Rectangle-packing-based module placement, " in *Proc. ICCAD 1995*, pp. 472–479, 1995.

[7] S. Tarafdar, M. Leeser, and Z. Yin, "Integrating floorplanning in data-transfer based high-level synthesis, " in *Proc. ICCAD 1998*, pp. 412-417, 1998.

[8] J. Um, J. Kim, and T. Kim, "Layout-driven resource sharing in high-level synthesis, " in *Proc. ICCAD 2002*, pp. 614–618, 2002.

[9] K. Wakabayashi and T. Yoshimura, "A resource sharing and control synthesis method for conditional branches, " in *Proc. ICCAD 1989*, pp. 62–65, 1989.

[10] K. Wakabayashi and H. Tanaka, "Global scheduling independent of control dependencies based on condition vectors, " in *Proc. 29th ACM/IEEE DAC 1992*, pp. 112–115, 1992.

[11] J. P. Weng and A. C. Parker, "3D scheduling: High-level synthesis with floorplanning, " in *Proc. 28th ACM/IEEE DAC 1992*, pp. 668–673, 1991.