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

2005－SLDM－122 (13)
2005／12／1

# レジスタ分散・共有アーキテクチャを対象としたフロアプラン指向高位合成手法

大智 輝[†] 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
〒 169-8555 東京都新宿区大久保 3-4-1
TEL:03-5286-3396, Fax:03-3203-9184
E-mail: †o-chi@ohtsuki.comm.waseda.ac.jp

**あらまし** 近年の LSI 設計プロセスの微細化に伴い，配線による遅延の割合がゲート遅延に対し相対的に増加してきており，高位合成の段階においてもフロアプランを考慮する必要がある．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り配線遅延が回路の性能に与える影響を削除することが可能であるが，レジスタ数が増大し面積増加を招いてしまうという問題点が生じる．本稿では，レジスタ分散型とレジスタ共有型を併用するレジスタ分散・共有型を対象とし，(1) スケジューリング，(2) レジスタアロケーション，(3) レジスタバインディング，(4) モジュール配置の工程を繰り返し (4) から得られたフロアプラン情報をフィードバックすることにより，解を収束させる高位合成手法を提案する．この手法はレジスタ分散型アーキテクチャと同等の回路の性能を維持しながら面積を削減す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，フロアプランニング，レジスタ分散・共有アーキテクチャ，配線遅延，ディープサブミクロンプロセス

# A High-level Synthesis Algorithm Based on Floorplans for Distributed/Shared-Register Architectures

Akira OHCHI[†], Nozomu TOGAWA[†], Masao YANAGISAWA[†], and Tatsuo OHTSUKI[†]

† Department of Computer Science, Waseda University
3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan
Tel: +81-3-5286-3396,Fax: +81-3-3203-9184
E-mail: †o-chi@ohtsuki.comm.waseda.ac.jp

**Abstract** As device feature size decreases, interconnection delay becomes the dominating factor of total delay. By using Distributed-Register architectures, we can synthesize the circuits with register-to-register data transfer, and can reduce influence of interconnect delay. However, Distributed-Register architectures have the problem that circuit area increases by the number of registers increasing. In this paper, we propose a high-level synthesis method targeting a Distributed/Shared-Register architectures.Our method repeats (1)scheduling,(2)register allocation,(3)register binding,(4)module placement processes,and feeds back floorplan information from (4). This method can reduce circuit area while maintaining the performance of the circuit equal with Distrubuted-register architectures. We show effectiveness of the proposed methods through experimental results.

**Key words** High-Level Synthesis, Floorplaning, Distributed/Shared-Register Architectures, Deep sub-micron process, Interconnect delay

## 1. はじめに

大規模複雑化する LSI 設計の生産性を向上させるため，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有効な手段である．従来型の高位合成では，フロアプラン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（演算器，レジスタ，コントローラ，MUX 等）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．近年，LSI 設計プロセスの微細化が進むにつれ，配線による遅延がゲート遅延に対し相対的に増加しており，今後もこの傾向は継続すると予想される．そのため，高位合成の段階においても，モジュールの配置関係，配線遅延を考慮する必要があると考えられる．

従来のフロアプランを考慮した高位合成手法の研究 [1], [8] 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言う手法が用いられてきた．これらの手法は，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．このモデルでは，演算器とレジスタとの間に長い配線を引く必要が生じる場合がある．これにより，配線遅延が増大し，回路の性能の低下を招いてしまう．また，[11] ではフロアプランと高位合成を同時に行っているが，これも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．その他のフロアプランを考慮した高位合成手法 [2]〜[4] では，フロアプロンを考慮することにより，配線長を減らし，低消費電力化を図っているが，回路の実行速度について言及されていない．

[6], [7] では，配線遅延がボトルネックとなる状況下を想定して，レジスタ分散型アーキテクチャ（Distributed-Register Architecture）を提案している．このアーキテクチャでは，各演算器ごとにローカルレジスタを配置する．レジスタ・演算器間の配線長が短くなりレジスタ共有型に比べレジスタ・演算器間の配線遅延が少なくなり，クロック周期をほぼ演算器の遅延のみで占められるようになる．そのため，クロック周期を短縮し回路の実行速度を上げることができる．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．この特徴を利用することにより，1 クロックあたりの演算器の利用効率が上がる．しかし，このモデルでは全ての演算器に専用のローカルレジスタを付加するため，レジスタ共有型よりもレジスタ数が増加し面積の増加という問題点が発生する．

その他のモデルとして，[5] では RDR（Regular Distributed Register）アーキテクチャを対象とした高位合成を提案している．このモデルは，チップを均一の大きさに分割し，その 1 つの区間に演算器，レジスタファイル，コントローラーを配置する手法である．このモデルは配線遅延がレジスタ共有型より小さくなり，設計は容易化されるが，回路を一定の大きさに分割するので面積のオーバーヘッドが大きくなると考えられる．

本稿では，レジスタ分散型の性能を維持しながら面積を減らすために，レジスタ分散型アーキテクチャとレジスタ共有型アーキテクチャを併用するレジスタ分散・共有併用アーキテクチャを対象とした高位合成手法を提案する．提案手法は配置情報をフィードバックする合成フローを用いることで，スケジューリング結果に配置情報を反映させることができ，各演算器が適したレジスタを使用できるようになる．また，提案手法は，スケジューリングが CVLS（Condition Vector List Scheduling）をベースとしており，レジスタバインディングで

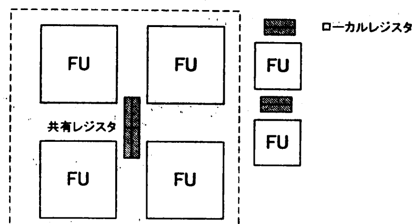


図 1　レジスタ分散・共有アーキテクチャ

は CV（Condition Vector）を使用することにより，分岐処理に対応している．

## 2. レジスタ分散・共有アーキテクチャ

提案手法はレジスタ分散型アーキテクチャとレジスタ共有型アーキテクチャを併用させたレジスタ分散・共有アーキテクチャを対象としている．

レジスタ分散型アーキテクチャは，各演算器に専用のローカルなレジスタを配置するアーキテクチャである．演算器に隣接した位置にレジスタを配置するため，演算器とレジスタ間の配線遅延が小さくなる．また離れた位置に配置された演算器同士は，レジスタ間データ転送を利用し，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であり，このレジスタ間データ転送を行うことにより，1 クロックあたりの演算器リソースの利用効率が上がる．しかし，完全なレジスタ分散型ではすべての演算器にローカルレジスタを付加するためレジスタ数の増大という問題が生じてくる．

図 1 に示すように，レジスタ分散・共有アーキテクチャは 1 つの共有レジスタ群を配置し，演算器の遅延とレジスタ・演算器間の配線遅延の合計がクロック周期制約を違反する共有レジスタから離れている演算器のみがローカルレジスタを持ちそれ以外の演算器は共有レジスタを使用するモデルである．このアーキテクチャを用いることでレジスタ間データ転送の利点を生かしながら，レジスタ分散型よりも面積を削減することができる．

## 3. 合成フロー

本稿における高位合成問題は，入力を CDFG とし，制約条件に演算器制約，クロック周期制約を与え，RT レベルのデータパスと制御回路およびモジュール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．また実行時間が同一の場合面積を最小化する．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 $N=\{n_i|i=1,2,...,m\}$ と，分岐制御を表すフォークノードの集合 $V_C$ を含むものとする．

提案手法では，レジスタ間データ転送を用いるため，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．また，各演算器のレジスタ種類（ローカルか共有）を決定するために，配置情報が必要となる．そのため，図 2 に示すような配置情報，配線遅延情報をフィードバックする合成フローを用

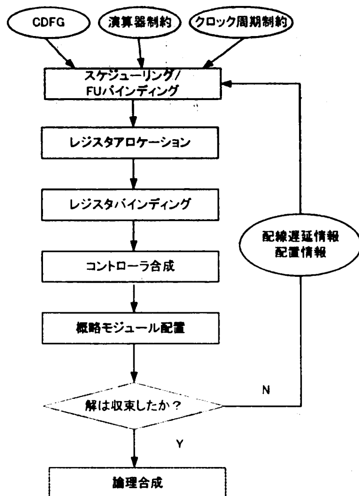


図 2　合成フロー

| | | (a) | (b) | (c) |
|---|---|---|---|---|
| x= a+b−c+d; | −(1) | [101] | [111] | [111] |
| if ( a != 0 ) | −(2) | [111] | [111] | [111] |
| y = x+c; | −(3) | [100] | [100] | [111] |
| else if ( a+b < c ) | −(4) | [011] | [011] | [111] |
| y= c+d; | −(5) | [010] | [011] | [111] |
| else y = x+d; | −(6) | [001] | [011] | [111] |
| Conditionals: | a != 0 −(2) | done | done | not |
| | a+b<c −(4) | done | not | don't care |

図 3　CV　　例

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 リストを元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 $f_i$ を選択する．$n_i$ が $f_i$ 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 $f_i$ にバインディングし，$f_i$ 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 4　スケジューリング/FU バインディング手順

いる．

スケジューリング/FU（Functional Unit）バインディング工程ではフィードバックされた配線遅延情報を参照し，[10] で提案された手法により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行う．

レジスタアロケーションの工程では，モジュール配置工程からフィードバックされた配置情報を参照し，クロック周期制約を満たすよう各演算器が使用するレジスタの種類を決定する．ここで，合成フローの初回には配線遅延情報を参照できないため，すべての演算器は共有レジスタを使用するとする．詳細については 5 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，共有レジスタもしくはローカルレジスタへ総レジスタ数が最小になるようにバインディングする．レジスタバインディングの詳細については 6 章で記述する．

モジュール配置の工程では，データ構造として Sequence-pair [9] 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時

$$\alpha A + \beta W + \gamma V \qquad (1)$$

と計算する．ただし，$\alpha$，$\beta$，$\gamma$ は，任意のパラメータである．合成フローの $i$ 回目のイタレーションの SA の初期温度を $T_i$ としたとき $i+1$ 回目のイタレーションの初期温度を

$$T_{i+1} = T_i / K \qquad (2)$$

とする．ただし $K > 1$ とする．また，モジュール配置の初期配置は前回のイタレーションの解を使用する．イタレーションの回数を重ねることにより SA の初期温度が下がりモジュール配置が固定されていき解が収束していく．

モジュール配置が終了した後，回路の面積，実行時間が収束したとみなされなければ，配線遅延情報，配置結果をそれぞれフィードバックする．

## 4. スケジューリング／FU バインディング

本稿でのスケジューリング/FU バインディング問題を次のように定義する．入力を CDFG，演算器間の配線遅延とし，制約に演算器制約，クロック周期制約を与え，演算ノードに演算器とコントロールステップを対応づける問題である．目的関数はコントロールステップの最小化である．

[10] で提案された CVLS ベースのスケジューリング手法を説明する．この手法は，CVLS をベースとしている．CVLS で用いられる CV と FUV という概念の定義について説明する．また，この手法では，データ転送制約テーブル，クリティカルパス長リスト，レイテンシの見積もりという概念を用いる．これらは，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れるために必要な概念である．

a)CV

CV は図 3 のように各リソース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うことを表している．また，CV は条件式の結果が利用可能かどうかにより変化する．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．

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．

a) 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，計算しておく．$t_{REG}$ 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

$$Slack_i = t_{CK} - t_{REG} - c_i \qquad (3)$$

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，

$$id_{i,j} = \begin{cases} 0 & (Slack_i \geq d_{i,j}) \\ \lceil (d_{i,j} + t_{REG})/t_{CK} \rceil & (Slack_i < d_{i,j}) \end{cases} \qquad (4)$$

と計算する．

b) 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．ノード $n_i$ を演算器 $f_i$ 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．

$$cp_{i,j} = c_j + \max_{n_k \in succ_i} \{\min_l (id_{j,l} + cp_{k,l})\} \qquad (5)$$

c) レイテンシの見積もり

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．ノード $n_i$ が演算器 $f_i$ 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 $f_i$ 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，$f_i$ 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，

$$lat_{i,j,k} = abli, j, k + cpi, j \qquad (6)$$

で計算されるレイテンシの見積もりの値を用いる．この $lat_{i,j,k}$ 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．

d) アルゴリズム

スケジューリングアルゴリズムを図 4 に示す．この手法は，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．

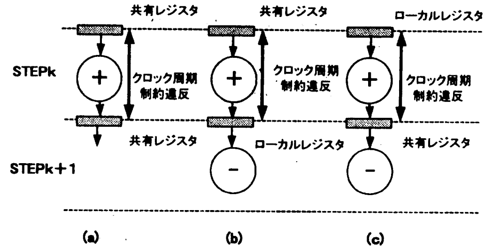


図 5　クロック周期制約違反

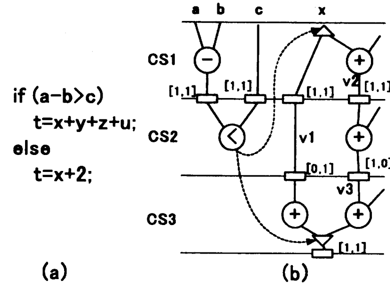


図 6　レジスタの CV

## 5. レジスタアロケーション

本稿でのレジスタアロケーション問題を次のように定義する．入力をスケジューリング済み CDFG，配線遅延情報とし，制約にクロック周期制約を与え，演算器ごとに共有レジスタを使用するか専用のローカルレジスタを追加するかを決定する問題である．目的関数はローカルレジスタ数の最小化である．

レジスタ分散・共有アーキテクチャでは各演算器が専用のローカルレジスタをつけるか共有レジスタを使用するかを決定する必要がある．提案手法では，共有レジスタをなるべく用いてレジスタ数の最小化する．しかし，演算処理時間（転送元レジスタ→演算器→転送先レジスタの総遅延時間）がクロック周期違反（演算器を実行するのに最低限必要なクロック数を超える場合）の時にはローカルレジスタを追加する必要がある．ここで，クロック周期違反は

$$T_{CLOCK} * \lceil T_f / T_{CLOCK} \rceil \qquad (7)$$

で計算できる（$T_{CLOCK}$:クロック周期，$T_f$:演算器の遅延時間）．

クロック周期違反を起こす場合とそのときのローカルレジスタのつけ方は以下の 3 通りがある．

- STEP$k$ で入力側・出力側共に共有レジスタの時にクロック周期制約違反を起こした場合，STEP$k$ の演算器にローカルレジスタをつける [図 5(a)]．
- STEP$k$ で入力側が共有レジスタで出力側にローカルレジスタの時にクロック周期制約違反を起こした場合，STEP$k$ の演算器にローカルレジスタをつける [図 5(b)]．
- STEP$k$ で入力側がローカルレジスタで出力側に共有レジ

手順 1. 全ての変数 $v_i \in V$ 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．

手順 2. 全ての変数 $v_i \in V$ について，STEP$p_i$〜STEP$p_i+1$ において $v_i$ を入力側レジスタに割り当て，STEP$q_i$〜STEP$q_i+1$ において，$v_i$ を出力側レジスタに割り当てる．

手順 3. 任意の変数 $v_i \in V$ を選択する．$m=1$，$n=1$ とする．もし出力側レジスタが共有レジスタならば手順 5 へ進む．

手順 4. STEP$p_i+m$〜STEP$p_i+m+1$ において出力側レジスタがまだ未割り当てならば，$v_i$ を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手順 5 へ進む．

手順 5. STEP$q_i-n$〜STEP$q_i-n+1$ において入力側レジスタがまだ未割当ならば $v_i$ を入力側レジスタに割り当て，$n$ に 1 を加え，手順 5 を繰り返す．割り当て不可能ならば手順 6 へ進む．

手順 6. $p_i+m-1_i-n$ が転送元レジスタ→共有レジスタ→転送先レジスタにかかるクロック数より少なければ転送元の演算器の出力側レジスタ数を 1 増やし手順 2 に進む．それ以外ならば，STEP$p_i+m$〜STEP$q_i-n$ に共有レジスタを割り当てる．

手順 7. V から $v_i$ を削除する．$V=\phi$ ならば終了する．それ以外ならば手順 3 へ戻る．

図 7 レジスタバインディング手順

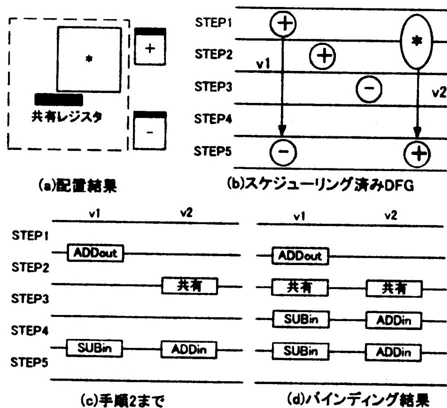


図 8 バインディング例

スタの時にクロック周期制約違反を起こした場合，STEP$k+1$ の演算器にローカルレジスタをつける [図 5(c)]．

また，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を区別しているため，ローカルレジスタを追加するときは，演算器の入力側と出力側にローカルレジスタを追加する．

## 6. レジスタバインディング

本稿でのレジスタバインディング問題を次のように定義する．入力をスケジューリング済み CDFG とし，CDFG の全てのエッジの各コントロールステップにレジスタを割り当てる問題である．目的関数は総レジスタ数の最小化である．

### 6.1 CV

レジスタバインディングでは，分岐処理に対応させるためにスケジューリングと同様に CV を使用する．また，条件式の結果が利用可能かどうかによりレジスタの CV も変化する．

図 6 の例をみると，変数 $v1$ の CS1 と CS2 の間に使用するレジスタはまだ条件判定の結果が出ていないので CV は [1,1] であり CS1 と CS2 の間はこのレジスタは他の変数との共有はできない．CS2 と CS3 の間に使用するレジスタは CS2 で条件判定を行う比較器がスケジューリングされており，条件判定の結果がわかるので CV は [0,1] となる．また変数 $v3$ の CS2 と CS3 の間に使用するレジスタの CV は [1,0] となり，CS2 と CS3 の間には変数 $v1$ と変数 $v3$ は CV に共通のビットに 1 が立っていないので同じレジスタにバインディングする事が可能となる．

### 6.2 アルゴリズム

レジスタ分散・共有併用型でのレジスタバインディング手順を図 7 に示す．この手順はスケジューリング済み CDFG と演算器のレジスタの種類を入力とし，総レジスタ数を最小とするようにレジスタバインディングをおこなう．この手順により変数がローカルレジスタもしくは共有レジスタに割り当てられる．その後共有レジスタに割り当てられた変数をレフトエッジ法でバインディングする．提案アルゴリズムはまず転送元演算器のレジスタに可能な限り変数を保持するようにバインディングを行い，競合が起こった場合は転送先演算器のレジスタにバインディングする．それでも競合が起きる場合，共有レジスタにバインディングする．

レジスタバインディングについて，図 8(b) のスケジューリング済み DFG を例に説明する．また図 8(a) のような配置で，点線で囲まれた演算器が共有レジスタを使用しそれ以外の演算器は専用のローカルレジスタがあると仮定する．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，図 8(b) のスケジューリング結果を満たすためには，少なくとも変数 $v1, v2$ は図 8(c) のように割り当てる必要がある．ただし ADDout，SUBin 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ることを意味している．

変数 $v1$ は STEP2〜3 で加算器のローカルレジスタはすでに使用され競合してしまうので STEP1 から STEP2 まで転送元の加算器のローカルレジスタをバインディングする．次に転送先の減算器のローカルレジスタに割り当てる．STEP2〜3 で減算器のローカルレジスタはすでに使用され競合するので STEP5 から STEP3 まで転送先のローカルレジスタにバインディングする．STEP2〜3 は転送元，転送先に割り当て不可能なので共有レジスタを割り当てる．また変数 $v2$ は乗算器が使用するレジスタは共有レジスタであるので，共有レジスタ数を最小限にするためには，共有レジスタに保持される期間をできるだけ少なくする必要があるため，転送先の加算器のローカルレジスタに長い期間割り当てられるようにする．

## 7. 計算機実験結果

提案手法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．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，提案手法であるレジスタ分散・共有併用型とレジスタ分散型，

表1 実験結果

| 例題入力 | 演算器制約 | 手法 | 面積 $[\mu m^2]$ | 実行時間 [ns] | 総レジスタ数 | ローカルレジスタ数 | 共有レジスタ数 | MUX数 |
|---|---|---|---|---|---|---|---|---|
| DCT | +3,*3 | 提案手法 | 1542560 | 55 | 20 | 18 | 2 | 37 |
| | | 分散型 | 1557504 | 57.5 | 26 | 26 | - | 41 |
| | | 共有型 | 1504230 | 70 | 15 | - | 15 | 31 |
| DCT | +4,*4 | 提案手法 | 1866200 | 42.5 | 25 | 24 | 1 | 45 |
| | | 分散型 | 1913565 | 42.5 | 30 | 30 | - | 45 |
| | | 共有型 | 1957371 | 60 | 15 | - | 15 | 30 |
| FIR | +3,*5 | 提案手法 | 2477204 | 75 | 33 | 31 | 2 | 66 |
| | | 分散型 | 2580600 | 75 | 41 | 41 | - | 80 |
| | | 共有型 | 2570104 | 87.5 | 22 | - | 22 | 57 |
| FIR | +3,*6 | 提案手法 | 3013144 | 75 | 34 | 27 | 7 | 86 |
| | | 分散型 | 2925032 | 75 | 46 | 46 | - | 74 |
| | | 共有型 | 2970864 | 82.5 | 22 | - | 22 | 60 |

レジスタ共有型を比較した [表1].

演算器は16bit幅と仮定し，面積と遅延はVDECライブラリ(注1) (CMOS0.35$\mu m$テクノロジー) 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$[\mu m^2]$，5.71[ns]とし，加算器の面積，遅延を25259$[\mu m^2]$，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$[\mu m^2]$，0.40[ns]，2-1マルチプレクサの面積，遅延をそれぞれ167$[\mu m^2]$，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た．また，配線遅延は配線長の2乗に比例すると仮定し，1000$[\mu m]$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，クロック周期制約は2.5[ns]とした．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．また実験結果の実行時間はアプリケーションの実行時間（クロック周期*ステップ数）を表している．

実験結果より提案手法の実行時間は分散型と同等の実行時間で共有型と比べ平均18.4%最大29.1%短縮された．また面積は分散型と比べDCT（+3,*3）で0.9%，DCT（+4,*4）で2.5%，FIR（+3,*5）で4.0%削減された．これより提案手法は実行時間をレジスタ分散型と同等に維持しながら面積を削減できている．

## 8. おわりに

本稿では，レジスタ分散・共有型アーキテクチャを対象とした高位合成フロー，レジスタバインディング手法を提案した．計算機実験により提案手法はレジスタ分散型と比較し，実行時間が約20%短縮でき，レジスタ分散型と比べ面積が減少する傾向があることが確認できた．今後の課題としては，共有レジスタ群の配置を分散化したアーキテクチャを対象が挙げられる．

**謝辞** 本研究に関し，有用な議論および討論をいただきました，株式会社NEC粟島亨博士，鈴木克青博士に感謝致します．本研究の一部は，日本学術振興会科学研究費補助金 (若手研究B，課題番号15700071)，電気通信普及財団研究調査助成金，早稲田大学特定課題研究助成費 (2005A-872) の援助を受けた．

(注1) 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

### 文献

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

[2] A. Stammermann, D. Helms, M. Schulte, A. Schulz, and W. Nebel, "Binding,allocation and floorplanning in low power high-level synthesis," in *Proc. International Conference on Computer Aided Design*, pp. 544–550,2003.

[3] L. Zhong and N. K. Jha, "Interconnect-aware high-level synthesis for low power," in *Proc. International Conference on Computer Aided Design*, pp. 110–117,2002.

[4] Z. Gu, J. Wang,R. P. Dick and H. Zhou, "Incremental Exploration of the Combined Physical and Behavioral Design Space," in *Proceedings of the 42nd annual conference on Design automation*, pp. 208–213,2005.

[5] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architectual synthesis integrated with global placement for multi-cycle communication," in *Proc. International Conference on Computer Aided Design (ICCAD-03)* , pp. 536–543, 2003.

[6] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," in *Proc. Asia South Pacific Design Automation Conf*,pp . 662–667,2001.

[7] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," in *Proc. International Conference on Computer Aided Design*, pp. 320–325,2001.

[8] Y. M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," in *Proc. International Conference on Computer Aided Design*, pp. 317–321,1994.

[9] H. Murata, K, Fujiyoshi, and S. Nakatake, "Rectangle-packing-based module placement," in *Proc. International Conference on Computer Aided Design*, pp. 472–479,1995.

[10] 田中真，内田純平，宮岡祐一郎，戸川望，柳澤政生，大附辰夫，"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，" 信学技報 VLD2004-82，pp. 127–132，2004.

[11] 大塚正臣，伊藤和人，"フロアプランと高位合成を同時に行うLSI設計手法，" 信学技報 2005-SLDM-119，pp. 25–30，2005.

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