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

2008－SLDM－137（25）
2008 / 11 / 19

レジスタ分散型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のためのマルチプレクサ削減手法

遠藤 哲弥† 大智 輝† 戸川 望† 柳澤 政生† 大附 辰夫†

† 早稲田大学大学院基幹理工学研究科 情報理工学専攻
〒 169–8555 東京都新宿区大久保 3–4–1
Tel: 03–3209–3211(5775), Fax: 03–3208–7439
E-mail: †endou@togawa.cs.waseda.ac.jp

あらまし リソース共有型の高位合成において，バインディング結果として演算器やレジスタの入力側にマルチプレクサが挿入される．マルチプレクサ数の増加は回路の面積増加や性能低下の原因となるため，高位合成の段階で考慮する必要がある．本稿では，レジスタ分散型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のためのマルチプレクサ削減手法を提案する．提案手法は，データ転送回数テーブルを利用したスケジューリング/FU バインディング手法，演算ノードの割当コントロールステップならびに割当 FU の局所変更を行う FU Connect Reduction 手法，モジュール間ポート再割当を行う Port Re-Assignment 手法，の 3 手法によりマルチプレクサ数を削減する．対象とする高位合成に提案手法を組み込む事で，平均で 15.4%のマルチプレクサ数，9.9%の面積が削減でき有効性を確認した．
キーワード マルチプレクサ，高位合成，レジスタ分散型アーキテクチャ，ポート再割当

A Multiplexer Reducing Algorithm in Floorplan-Aware High-level Synthesis for Distributed-Register Architectures

Tetsuya ENDO†, Akira OHCHI†, Nozomu TOGAWA†, Masao YANAGISAWA†, and Tatsuo OHTSUKI†

† Dept. of Computer Science and Engineering, Waseda University
3–4–1 Okubo, Shinjuku, Tokyo 169–8555, Japan
Tel: +81–3–3209–3211(5775), Fax: +81–3–3208–7439
E-mail: †endou@togawa.cs.waseda.ac.jp

Abstract In high level synthesis for resource shared architecture, multiplexers are inserted between registers and functional units as a result of binding. Multiplexer reduction is necessary for area and performance of synthesized circuit. In this paper, we propose multiplexer reducting algorithms in floorplan-aware high-level synthesis for distributed-register architectures. These algorithms can reduce the number of multiplexers for conventional high-level synthesis. We show effectiveness of the proposed algorithm thorough experimental results.
Key words multiplexer, high-level synthesis, distributed-register architecture, port reassignment

1. まえがき

大規模複雑化する LSI 設計の生産性を向上させるため，抽象度の高い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，モジュール（演算器，レジスタ，コントローラ，マルチプレクサ等）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おり，今後もこの傾向は継続すると予想される．そこで，高位合成の段階においても，モジュール配置，配線遅延を考慮することが必要となる．一方で，リソース共有型の高位合成において，バインディングの結果として，演算器やレジスタの入力側に必要に応じてマルチプレクサが挿入される．マルチプレクサは合成結果として出力される回路の面積増加や性能低下の原因となる．特に FPGA のように回路構成を変更できるような集積回路では，マルチプレクサが回路面積や性能，消費電力に与える影響が大きいことが報告されている [3]．結線数の増大，結線構造の複雑化によるマルチプレクサの増大は今後も続くと予想され，

マルチプレクサが回路に与える影響も大きくなると考えられる．

従来のフロアプランを考慮した高位合成の研究 [5], [12] では，モジュール配置を工夫することで，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している．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，回路の性能低下を招いてしまう．

配線遅延がボトルネックとなる状況下を想定して，[6], [7] ではレジスタ分散型アーキテクチャが提案された．このアーキテクチャでは，演算器毎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取りするため，配線遅延の影響は小さくなり，クロック周期をほぼ演算器の遅延で占められるようになる．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特徴を持つ．この特徴により，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，従来のレジスタ共有型アーキテクチャよりもレジスタ数が増加してしまう．レジスタ数の増加に伴い，レジスタ間接続に要する結線数が増加し，結線制御に必要なマルチプレクサ数が増大するという問題点が発生する．そのため，レジスタ分散型アーキテクチャにおけるマルチプレクサの削減が必要がある．

マルチプレクサ削減手法については，多くの既存研究が存在するが，最近では次のような研究がある．[2] では，レジスタの入力ポート割当の変更を行うことでマルチプレクサを削減する手法を提案している．しかしながら，この手法はレジスタ共有型アーキテクチャを対象としているため，レジスタ分散型アーキテクチャには対応していない．[4] では，互いに依存関係にある FU バインディング，レジスタバインディングを同時に行う手法を提案している．[4] ではバインディングは最小費用流問題として解かれ，マルチプレクサ数を削減している．[8] では，結線数削減を目的としたパスベースバインディング手法を提案し，マルチプレクサへの総入力線を削減している．しかしながら，[8] の手法では入力アプリケーションによっては演算器，レジスタ数が増加している．[9] では，適当な FU バインディングおよびレジスタバインディングを初期解として，タブーサーチをベースとした局所改善を反復して行う手法を提案している．[9] では，一定の反復回数ごとに重み付き二部グラフの最大マッチングを用いた FU バインディングおよびレジスタバインディングを行うことで探索の停滞を防止し，収束性を高めている．しかしながらこの手法は回路の遅延を考慮していない．以上の既存研究では全てスケジューリングは終了しているものとし，FU バインディング，またはレジスタバインディングを行っている．そのため，スケジューリング結果がマルチプレクサ数に与える影響を考慮することが困難である．

本稿ではレジスタ分散型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のためのマルチプレクサ削減手法を提案する．提案手法は，[11] で提案された高位合成をベースとしている．[11] は，レジスタ分散型アーキテクチャを対象とし，(1) スケジューリング/FU バインディング，(2) レジスタバインディング，(3) コントローラ合成，(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させることにより解を収束させる高位合成である．提案手法では，[11] に対し，(1-1) データ転送回数テーブルを利用したスケジューリング/FU バインディング手法，(1-2) スケジューリング/FU バインディング解に対し，可動度という概念を用い，演算ノードの割当ステップならびに割当 FU を局所変更する FU Connect Reduction 手法，(2) レジスタバインディング解に対し，モジュール間ポート再割当を行う Port Re-Assignment 手法，の 3 手法を組み込むことで，マルチプレクサ数を削減することを可能とする．

本稿は以下のように構成される．2 章では対象とするレジスタ分散型アーキテクチャを説明する．3 章では提案手法を組み込んだ高位合成手法の概要，合成フロー，各工程を説明する．4 章では，計算機実験を行った結果を報告する．

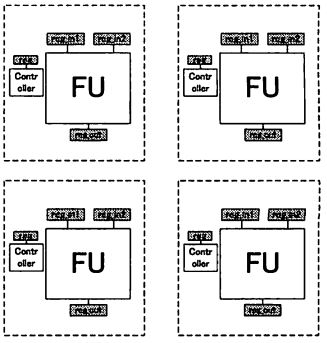

図 1　レジスタ分散型アーキテクチャ．

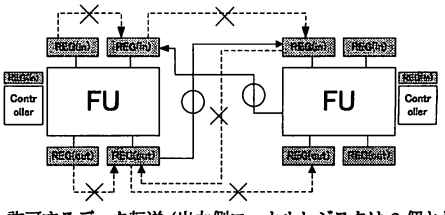

図 2　許可するデータ転送 (出力側ローカルレジスタは 2 個とした)

2. レジスタ分散型アーキテクチャ

図 1 にターゲットアーキテクチャであるレジスタ分散型アーキテクチャ [1], [11] を示す．レジスタ分散型アーキテクチャでは，配線遅延がボトルネックとなる状況を想定して，各演算器が入出力に専用のローカルレジスタを有する構成となっている．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．離れたローカルレジスタとデータをやり取りするため，レジスタ共有型のモデルと比較して配線遅延の影響は小さくなり，クロック周期をほぼ演算器の遅延で占めることが可能となるため回路性能が向上する．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ため，レジスタ間データ転送による複数のサイクルをまたぐ配線遅延を扱うこともできる．またこのモデルでは，各演算器毎に専用のコントローラを有する構成となっている．いずれの演算器に対しても隣接した位置に専用のコントローラが配置されるため，モジュールとコントローラ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では，各演算器に配置する入力側ローカルレジスタ数を 2 個 (固定)，出力側ローカルレジスタ数を可変とし，

- 出力側ローカルレジスタ⇒入力側ローカルレジスタ
- 演算器の出力 (ダイレクト) ⇒入力側ローカルレジスタ

のデータ転送のみ考えることにする (図 2)．

レジスタ分散型アーキテクチャを使用したとき，転送元レジスタ $r1$ から演算器 fu を通って転送先レジスタ $r2$ にデータを格納するまでの遅延時間 T_c は，演算器 fu の遅延を t_{fu}，レジスタのセットアップ時間を t_{reg}，コントローラの遅延を t_{con}，レジスタ $r1$ から演算器 fu の配線遅延を t_{r1fu}，演算器 fu からレジスタ $r2$ への配線遅延を t_{fur2}，コントローラからマルチプレクサへの配線遅延を t_{cm}，マルチプレクサの段数を M，遅延時間を t_{mux} としたとき，

$$t_1 = t_{reg} + t_{r1fu} + t_{fu} + t_{fur2}$$

$$t_2 = t_{con} + t_{cm}$$

$$T_c = \max(t_1, t_2) + M \cdot t_{mux} \quad (1)$$

で算出できる．ここで，コントローラの遅延 t_{con} は無視できるほど小さく，ボトルネックとなることがない．また，演算器とレジスタ，コントローラは隣接して配置されるため，配線遅延 t_{r1fu}，t_{fur2}，t_{cm} は無視できるようになる

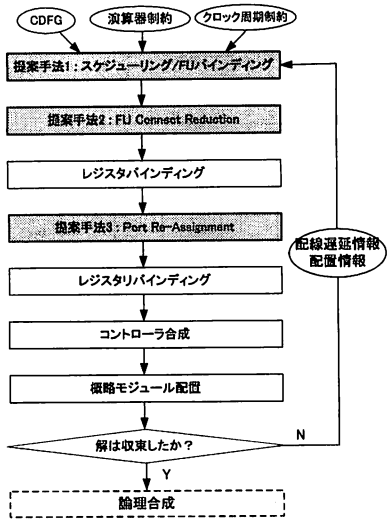

図 3　合成フロー

3. レジスタ分散型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

本章では，既存手法 [11] に提案手法を組み込んだ，レジスタ分散型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の概要，合成フロー，各工程を提案する．

3.1 問題定義

CDFG 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 $V_N=\{n_i \mid i=1,2,\ldots,l\}$ と，分岐制御を表すフォークノードの集合 V_C を含むものと定義する．エッジ集合 E は，データフローエッジ集合 E_d と，コントロールフローエッジ集合 E_c を含むものと定義する．

利用可能な演算器を表すリスト $F=\{f_i \mid i=1,2,\ldots,m\}$ を定義した時，演算器制約とは F 内の演算器のみを用いるという制約である．各演算器は，面積及び演算に要する遅延に関する情報を持つ．クロック周期制約とは，n クロックで演算を処理するときに，クロック周期制約 T_{clk} に対して $T_c/n \leqq T_{clk}$ の条件を満たさなければならないという制約である．

以上のもと，高位合成問題とは，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．第一目的関数は入力アプリケーションの実行時間の最小化，第二目的関数は回路面積の最小化である．総マルチプレクサ数を削減することで回路面積最小化を実現する．

3.2 合成フロー

レジスタ間データ転送を扱うためには，スケジューリングの段階で各演算器間の配線遅延情報が必要になる．[11] では，配置情報，配線遅延情報をフィードバックし，スケジューリング段階で各演算器の配線遅延を考慮する合成フローを提案している．本稿では，[11] を拡張し，新たにマルチプレクサ削減工程を加えた合成フローを提案する (図 3)．

スケジューリング/FU バインディングの工程では，入力を CDFG，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とし，レジスタ間データ転送を利用するスケジューリングとデータ転送回数テーブルを利用する FU バインディングを同時実行する．目的関数はコントロールステップ数の最小化である．

FU Connect Reduction 手法の工程では，入力をスケジューリング/FU バインディング済み CDFG，前工程で計算されたデータ転送回数テーブルとし，可動度という概念を用い演算ノードの割当ステップならびに割当 FU を局所変更する．目的関数は演算器間結線数の最小化である．

レジスタバインディングの工程では，入力をスケジューリング済み CDFG とし，CDFG から抽出される全てのエッジの各コントロールステップにレジスタを対応づける．目的関数は総レジスタ数の最小化である．レジスタバインディングでは，最初は入力側ローカルレジスタ数を 2 個 (固定)，出力側ローカルレジスタ数を 1 個 (可変) 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．

Port Re-Assignment 手法の工程では，入力をスケジューリング/FU バインディング済み CDFG，レジスタバインディング解とし，モジュール間接続に対し，入出力ポートの再割当を行う．目的関数は演算器の入力側ローカルレジスタへの総入力線の最小化である．

レジスタリバインディングの工程では，ポート再割当済み CDFG を入力とし，再度レジスタバインディングを行う．目的関数は総レジスタ数の最小化である．

コントローラ合成の工程では，Synopsys 社 Design Compiler を用いてコントローラを論理合成する．

モジュール配置の工程では，データ構造に Sequence-pair [10] を用い，モジュール配置を SA によって最適化する．SA のコスト関数は，

$$\frac{A_{BB}}{A_{total}}+\alpha\frac{V}{T_{clock}}+\beta\frac{W}{W_{Max}} \tag{2}$$

である．ここで A_{BB} はデッドスペースを含む回路の総面積，A_{total} はモジュール面積の総合計，V は各モジュール間のデータ転送においてクロック周期制約違反した遅延の総合計，T_{clock} はクロック周期，W は各モジュールを結ぶ総配線長，W_{Max} は $(height+width)*W_{num}$(W_{num} は配線本数) である．また，

$$\alpha=\alpha_{base}+(i*\gamma) \tag{3}$$

とする．ここで i は図 3 に示す合成フロー全体のイタレーション回数であり，α_{base},γ は任意のパラメータである．また合成フローの 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 $T_{i+1}=T_i/K$ 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．

図 3 において，レジスタバインディング，レジスタリバインディング，概略モジュール配置の 3 工程に対しては [11] で提案された手法を用いる．さらに，データ転送回数テーブルを利用したスケジューリング/FU バインディング手法 (3.3 節)，FU Connect Reduction 手法 (3.4 節)，Port Re-Assignment 手法 (3.5 節) の 3 手法を提案し，合成フローに組み込むことで，マルチプレクサ数削減を実現する．

3.3 [提案手法 1] データ転送回数テーブルを利用したスケジューリング/FU バインディング手法

本節ではデータ転送回数テーブルを利用したスケジューリング/FU バインディング手法を提案する．

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．入力を CDFG，演算器間の配線遅延情報とし，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付ける問題である．目的はコントロールステップ数の最小化である．

3.3.1 データ転送制約テーブル

前回のイタレーションにおける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元に計算する．下記に示す計算法で，任意の 2 個の演算器間でのデータ転送に伴う遅延がクロック周期内に収まるかを判定している．レジスタのセットアップ時間を t_{reg}，クロック周期を t_{clk} とする．演算器 f_i の遅延を $fd(f_i)$，f_i から f_j の配線遅延を $id(f_i,f_j)$，マルチプレクサの段数を M，遅延を t_{mux} とし，

$$Slack_i = t_{clk} - t_{reg} - fd(f_i) - M \cdot t_{mux} \quad (4)$$

と計算する時，$f_i \rightarrow f_j$ というデータ転送に必要なステップ数 $dts(f_i, f_j)$ を

$$dts(f_i, f_j) = \begin{cases} 0 & (Slack_i \geqq id(f_i, f_j)) \\ \lceil (id(f_i, f_j) + t_{reg})/t_{clk} \rceil & (Slack_i < id(f_i, f_j)) \end{cases} \quad (5)$$

と計算する．

3.3.2 データ転送回数テーブル

データ転送回数テーブルは，現在のイタレーションにおける全演算器間のデータ転送回数を保持するものである．演算ノード n_i を演算器 f_j にバインディングする際，n_i に接続された全親演算ノード集合 $parent(n_i)$ に対し，$\forall n_g \in parent(n_i)$ を満たす演算ノード n_g に割り当てられた演算器 $f(n(g))$ と f_j のデータ転送回数を加えていく．全演算ノードの FU バインディングが終了することで，各演算器間の最終的なデータ転送回数 $DTC(f(n_g), f_j)$ が決定する．ただし，演算器 $f(n_g)$ から f_j へのデータ転送，f_j から $f(n_g)$ へのデータ転送は異なる．

3.3.3 手　　順

最初に各演算ノードに対しクリティカルパス長を求める．クリティカルパス長 $cp(n_i, f_j)$ はノード n_i を演算器 f_j に割り当てたときのエンドノードからノード n_i への最大パス長である．ノード n_e をエンドノードの一つとするとこのノードを演算器 f_j に割り当てたときの $cp(n_e, f_j)$ は f_j の演算処理クロック数 $c(f_j)$ となる．エンドノードからスタートノードに向かって次式によりクリティカルパス長を計算する．

$$cp(n_i, f_j) = c(f_j) + \max_{n_k \in succ(n_i)} \left[\min_{f_l \in FUs} \{ dts(f_j, f_l) + cp(n_k, f_l) \} \right] \quad (6)$$

ここで $succ(n_i)$ はノード n_i の子孫ノードである．

次に，優先度を各ノードごとに設定する．ノード n_i の優先度 $priority(n_i)$ を以下の式で計算する．

$$priority(n_i) = \min_{k \in FUs} cp(n_i, f_k) \quad (7)$$

スケジューリング/FU バインディングの手順は以下のように行う．最初にレディーリストを作成し，その中から $priority(n_i)$ が一番大きいノードを選択する．ノード n_i に対し，各演算器に割り当てたときのレイテンシの見積もり $lat(n_i, f_j)$ を算出する．レイテンシの見積もり $lat(n_i, f_j)$ は演算器 f_j に割り当てたときのスケジュール後の最大コントロールステップの見積もり値であり，以下の式で計算する．

$$lat(n_i, f_j) = cs(n_i, f_j) + cp(n_i, f_j) \quad (8)$$

$cs(n_i, f_j)$ は n_i が演算器 f_j に割り当て可能な最小コントロールステップ数であり，リソース競合状態によって算出される．

レイテンシの見積もり $lat(n_i, f_j)$ を算出した後，レイテンシの見積もり値が最小となる演算器 f_j が対象コントロールステップに割り当て可能ならば割り当てを行う．

ここで，レイテンシの見積もり値が最小となる演算器が複数存在した場合に，最適な演算器を選ぶ判断基準として，データ転送回数テーブルを用いる．データ転送回数テーブルを元に，それまでのバインディング結果としてデータ転送回数テーブルに保持された接続情報を解析し，n_i の任意の親演算ノード n_g に割り当てられた演算器 $f(n_g)$ との接続が既に存在する演算器を選択する．該当する接続がデータ転送回数テーブルに存在しない場合，任意の演算器にバインディングする．

この手法により，クロック周期制約に対応し，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，データ転送回数テーブルを利用することで，演算器間の接続関係を考慮したバインディング実現し，演算器間結線数を削減できる．

3.4 [提案手法 2] FU Connect Reduction 手法

本節では，演算器間結線数最小化を目的とする FU Connect Reduction 手法 (以下 FUCR 手法) を提案する．

FUCR 問題を次のように定義する．入力をスケジューリング/FU バインディング済み CDFG，データ転送回数テーブルとし，スケジューリング結果を初期解としたとき，各演算ノードごとに最適な割当ステップならびに割当 FU を決定する問題である．目的関数は演算器間総結線数の最小化である．

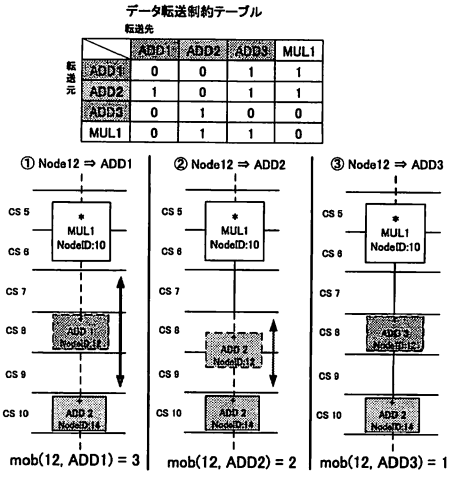

図 4　可動度の例

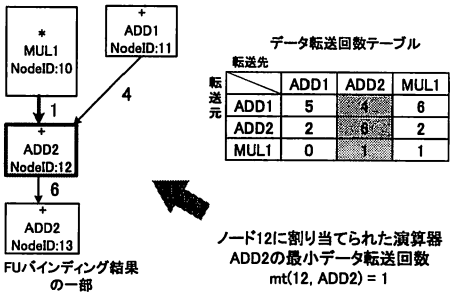

図 5　最小データ転送回数の例

3.4.1 可動度 (mobility)

可動度は，演算ノードを割当可能な各演算器に割り当てた場合の可動ステップ数を表したものである．ノード n_i を演算可能な演算器集合を $bindable(n_i)$ とする．n_i を演算器 $f_j \in bindable(n_i)$ に割り当てた際の最小コントロールステップ $mcs(n_i, f_j)$ を式 (9)，最大コントロールステップ $Mcs(n_i, f_j)$ を式 (10) で計算する．ここで，ノード n_i の可動範囲 $able(n_i, f_j)$ を以下の式 (11)，可動度 $mob(n_i, f_j)$ を以下の式 (12) で計算する．

$$mcs(n_i, f_j) = \max_{n_g \in parent(n_i)} \left[cs(n_g) + c(f(n_g)) + dts\{f(n_g), f_j\} \right] \quad (9)$$

$$Mcs(n_i, f_j) = \min_{n_k \in child(n_i)} \left[cs(n_k) - c(f(n_j)) - dts\{f_j, f(n_k)\} \right] \quad (10)$$

$$able(n_i, f_j) = \{ q \mid mcs(n_i, f_j) \leqq q \leqq Mcs(n_i, f_j) \} \quad (11)$$

$$mob(n_i, f_j) = | able(n_i) | \quad (12)$$

ここで，$parent(n_i)$，$child(n_i)$ はそれぞれノード n_i に接続された親ノード，子ノードの集合であり，$cs(n_g)$, $cs(n_k)$ はそれぞれ前工程のスケジューリングで決定したノード n_g, n_k が演算されるコントロールステップ，$c(f(n_g))$, $c(f(n_k))$ はそれぞれ演算器 $f(n_g)$, $f(n_k)$ の演算サイクルを表す．FUCR では，各演算ノードごとに演算可能な演算器全ての可動度を計算する．n_i を f_j に割り当てる場合，可動範囲 $able(n_i, f_j)$ 内でスケジューリングの変更が行える．可動度の例を図 4 に示す．図 4 では，ノード 12 に異なる加算器 ADD1，ADD2，ADD3 を割り当てた際の可動度を示している．

3.4.2 最小データ転送回数の計算

データ転送回数テーブルを元に，演算ノード n_i を演算可能な各演算器 $f_j \in bindable(n_i)$ に割り当てた際，f_j と n_i に接続された全演算ノードに割り当てられた演算器との間の最小データ転送回数 $mt(n_i, f_j)$ を計算する．$mt(n_i, f_j)$ は以下の式

$$mt(n_i, f_j) = \min \Big[\min_{n_g \in parent(n_i)} \{ DTC(f(n_g), f_j) \}, \min_{n_k \in child(n_i)} \{ DTC(f_j, f(n_k)) \} \Big] \quad (13)$$

Step 1 全ての演算ノード n_i に対し，割当可能な全ての演算器 f_j との可動度 $mob(n_i, f_j)$，最小データ転送回数 $mt(n_i, f_j)$ を計算する．
Step 2 全ての演算ノード n_i に対し $priority(n_i)$ を計算し，優先度リスト L を作成する．
Step 3 最も優先度 $priority(n_i)$ が高いノード $n_i \in L$ を選択し，移動・置き換え操作を判定する．操作可能な場合，n_i のスケジューリング/FU バインディングを変更し，Step5 へ．不可能な場合 Step4 へ．
Step 4 n_i と同一ステップ内の同種の演算ノード n_k に対し交換操作を判定する．操作可能な場合，n_i，n_k のバインディングを変更し Step5 へ．不可能な場合 Step6 へ．
Step 5 可動度，データ転送回数テーブルを更新する．
Step 6 $n_i(n_k)$ を優先度リスト L から削除する．L が空の場合，アルゴリズムを停止する．空でない場合 Step2 へ．

図 6 提案 FUCR 手順

で計算される．FUCR では，各演算ノードごとに演算可能な演算器全ての最小データ転送回数を計算する．

図 5 にスケジューリング/FU バインディング済み CDFG における，ノード 12 に対する最小データ転送回数の例を示す．

3.4.3 移動・置き換え操作

演算ノード n_i に対し，現在演算器 f_j が割り当てられているものとする．ここで，以下の式

$$mt(n_i, f_j) \leqq mt(n_i, f_p) \tag{14}$$

を満たす演算器 $f_p \in bindable(n_i)$ が存在している場合，f_p が利用可能なコントロールステップ $q \in able(n_i, f_p)$ に n_i を移動し，f_p に割り当てる．利用可能なコントロールステップが $able(n_i, f_p)$ 内に存在しない場合，操作を打ち切る．移動・置き換え操作により，対象ノードを他の演算器に割り当てることで，最小データ転送回数が小さい演算器間の接続をなくすことが可能となる．

3.4.4 交換操作

演算ノード n_i に対し，移動・置き換え操作ができない場合，同一コントロールステップ内の演算ノード n_k に割り当てられた演算器 $f_l \in bindable(n_i)$ とのバインディング交換を行う．交換操作において，満たすべき条件は，データ転送制約を違反しないこと，そして以下の 2 式

$$mt(n_i, f_j) \leqq mt(n_i, f_l) \tag{15}$$

$$mt(n_k, f_l) \leqq mt(n_k, f_j) \tag{16}$$

を満たすノード間でのバインディング交換を行うことである．交換操作により，対象の 2 ノードに割り当てられた演算器を交換することで，最小データ転送回数が小さい演算器間の接続をなくすことが可能となる

3.4.5 手順

提案 FUCR 手法の手順を図 6 に示す．最初に各演算ノードごとに可動度，最小データ転送回数を計算する．次に優先度を各ノードごとに決定する．優先度は以下の式

$$priority(n_i) = \min_{f_j \in bindable(n_i)} mt(n_i, f_j) \tag{17}$$

により計算する．優先度 $priority(n_i)$ が同一の演算ノードが複数存在する場合，可動度が大きいノードに高い優先度を与える．優先度が高い演算ノードから順に移動・置き換え操作あるいは交換操作を行うことにより，演算器間の接続数が削減される．

3.5 [提案手法 3] Port Re-Assignment 手法

本節では，モジュール間接続におけるポート割当最適化を目的とする Port Re-Assignment 手法 (以下 PRA 手法) を提案する．

PRA 問題を次のように定義する．任意の演算器 f_i の入出力に配置されたローカルレジスタをそれぞれ $R_{in}(i) = \{r^1_{in}(i), r^2_{in}(i)\}$，$R_{out}(i) = \{r^k_{out}(i), | k = 1, 2, \ldots, n\}$ と表す．入力をスケジューリング/FU バインディング済み CDFG，レジスタバインディング結果とし，任意の演算器 f_h から演算器 f_i へのデータ転送を考える．このとき PRA 問題とは，各データ転送ごとに，f_h の出力，あるいは f_h の k 番目の出力側ローカルレジスタ $r^k_{out}(h)$ の出力を，f_i の入力側ローカルレジスタ $r^1_{in}(i)$，$r^2_{in}(i)$ の一方，もしくは両方の入力側ポートに割当を行うかを決定する問題である．目的関数は演算器の入力側ローカルレジスタへの総入力線の最小化である．

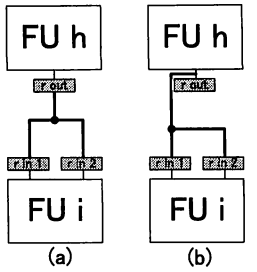

図 7 ポート再割当候補

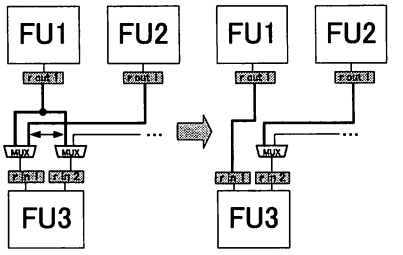

図 8 PRA の例

3.5.1 PRA によるポート割当最適化

PRA では，f_i に配置された 2 つの入力側ローカルレジスタ $r^1_{in}(i)$，$r^2_{in}(i)$ に着眼し，$r^1_{in}(i)$，$r^2_{in}(i)$ の入力側ポートに対しどのように配線が接続されているかを調べる．そこで，f_h から f_i へのデータ転送がある際，転送元演算器である f_h の出力側ポート，あるいは f_h の k 番目の出力側ローカルレジスタの出力側ポート $r^k_{out}(h)$ が，転送先ローカルレジスタ $r^1_{in}(i)$，$r^2_{in}(i)$ の両方の入力側ポートを利用するように配線が接続されている場合，冗長な配線が増えることによってマルチプレクサ数が増加する可能性がある．このため，PRA の工程ではこのような接続に対する割当を最適化することで，冗長な配線が減らしていく．PRA の候補となる接続を

- 転送元ローカルレジスタの出力側ポート⇒転送先ローカルレジスタの入力側ポート (図 7(a))
- 転送元演算器の出力側ポート⇒転送先ローカルレジスタの入力側ポート (図 7(b))

とする．各データ転送ごとにどのポートを用いているかを探索し，PRA の候補となる接続に対し，一方の入力側ローカルレジスタの入力側ポートに接続を割当直せるかどうかを調べ，ポート交換が可能な演算器 (加算器，乗算器，AND 演算) である場合，一方に割当を変更し固定していく．PRA の例を図 8 に示す．図 8(a) では，FU1 の出力側ローカルレジスタ $r^1_{out}(FU1)$ の出力側ポートが，転送先演算器 FU3 の入力側ローカルレジスタ $r^1_{in}(FU3)$，$r^2_{in}(FU3)$ の両方の入力側ポートと接続されているため，図 8(b) で表されるように一方に割当を変更し冗長な結線を削減する．

3.5.2 優先度リストを利用したポート再割当手法

PRA の候補となる接続に対し，どの候補から順番に再割当を行うのか，また f_h から f_i への接続が候補であったとき，$r^1_{in}(i)$，$r^2_{in}(i)$ のどちらに割当を固定するのかを決定するため，優先度リストを利用したポート割当手法を行う．

まず，変数 (各データ転送) を全探索し，PRA の候補となる接続を検出する．そして，そのデータ転送回数を元に優先度を決定する．$r^1_{in}(i)$ の入力側ポートへのデータ転送回数を p[回]，$r^2_{in}(i)$ の入力側ポートのデータ転送回数を q[回] とした時，$p+q$[回] を再割当候補に対する優先度とし，優先度リスト

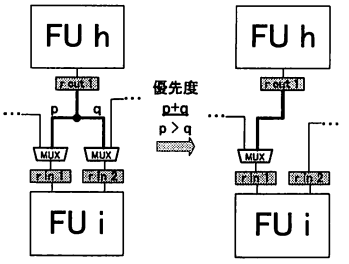

図 9 優先度決定，再割当

Step 1 レジスタバインディング結果より，各演算器，又は各出力側ローカルレジスタの出力側ポートから各入力側ローカルレジスタの入力側ポートへのデータ転送回数を算出する．
Step 2 Step1 を元に，PRA の候補となる接続を抽出する．各候補に対し優先度を算出し，優先度リストを作成する．
Step 3 優先度リストを元に，優先度が高い順に PRA を行う．リストが空の場合 PRA を終了する．ここで，転送先演算器が加算器，乗算器，AND 演算の場合，Step4 へ．それ以外の演算器の場合，該当候補に対する処理を打ち切りリストから削除する．
Step 4 再割当候補に対し，データ転送回数の多いポートに再割当を行う．全ての接続に再割当を行った後リストから削除し Step 3 へ．

図 10 提案 PRA 手順

表 1 演算器の面積と遅延

	面積 (um^2)	遅延 (ns)
加算器	287	1.36
減算器	318	1.37
乗算器	4507	2.93
比較器	148	0.88
AND 演算	68	0.03
シフタ	270	0.48
レジスタ (1bit)	13	0.09
マルチプレクサ (1bit)	7	0.04

を作成する．優先度リストを元に，優先度の高い候補から順番に PRA を行う．ここで，p と q でデータ転送回数の多いポートに割当を固定していく．図 9 に例を示す．

3.5.3 手　順

提案 PRA 手法の手順を図 10 に示す．この手順によりレジスタバインディング結果に変更を加え，ポートを再割当する．その後，再度レジスタバインディングを行うことにより変数が最適なレジスタに割り当てられ，演算器の入力側ローカルレジスタへの入力線の総数が削減される．

4. 計算機実験

提案手法を C++言語を用いて計算機上に実装した．実験環境は OS が Debian/Sarge，CPU が Intel Xeon 3.4GHz，メモリ容量が 4GB である．

演算器は 16bit 幅と仮定し，面積/遅延は Synopsys 社 Design Compiler を用い，セルライブラリには STARC(注1)(CMOS 90[nm]) を用い，あらかじめ合成して得られた値を利用した．実験で用いた演算器の面積，遅延を表 1 に示す．配線遅延は配線長の 2 乗に比例すると仮定し，250[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，クロック周期制約は 1.8[ns] とした．また，

対象としたアプリケーションとして DCT（ノード数 48），7 次 FIR フィルタ（ノード数 75），EWF（ノード数 34），EWF3（ノード数 102），COPY(ノード数 378，分岐有) を用いた．

[11] と提案手法 (1 と 2 の組み合わせ)，提案手法 (1,2,3 全て組み合わせる) を比較した．なお，[11] で提案されたレジスタ分散型アーキテクチャではコントローラを 1 つとしているが，本稿では [11] もコントローラ分散化して実験を行った．実験結果を表 2 に示す．実験結果における面積は，演算器，レジスタ，マルチプレクサ，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．CT はイタレーションを含む合成フロー全体にかかる時間である．実験結果より，提案手法は [11] と比較して，コントロールステップ数，ローカルレジスタ数をほぼ維持しつつ，平均で 15.4%マルチプレクサ数を削減し，平均で 9.9%総面積を削減している．これより提案手法を組み込んだ場合，[11] と同等の性能を維持しながらマルチプクレサ数，総面積を削減できることを確認した．

(注1)：STARC[90nm] 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，株式会社半導体理工学研究センター（STARC）と株式会社先端 SoC 基盤技術開発（ASPLA）の協力で開発されたものである．

表 2 実験結果

App. (演算器制約)	手法	面積 [um^2]	#CS	#Regs.	#Muxs	CT (sec)
DCT (+3*3)	手法 [11]	34894	14	25	75	1302
	提案手法 (1+2)	32906	14	25	66	1007
	提案手法 (1+2+3)	32185	14	25	61	3315
FIR (+3*3)	手法 [11]	28329	33	23	34	1183
	提案手法 (1+2)	26320	33	23	33	3774
	提案手法 (1+2+3)	25560	33	23	31	1182
EWF (+2*1)	手法 [11]	15390	21	12	31	678
	提案手法 (1+2)	15390	21	12	31	837
	提案手法 (1+2+3)	14040	21	12	27	444
EWF3 (+3*2)	手法 [11]	30294	52	20	62	885
	提案手法 (1+2)	28361	52	20	54	757
	提案手法 (1+2+3)	25776	52	20	48	1154
COPY(+3 -1*5AND1 <1shift2)	手法 [11]	94997	141	147	339	8963
	提案手法 (1+2)	88292	135	140	304	15069
	提案手法 (1+2+3)	86944	135	139	292	16142

5. むすび

本稿では，レジスタ分散型アーキテクチャを対象とする高位合成手法のためのマルチプレクサ削減手法を提案した．計算機実験により提案手法はレジスタ分散型アーキテクチャを対象とする従来の高位合成手法 [11] と比較し，コントロールステップ数，ローカルレジスタ数をほぼ維持しつつ，平均で 15.4%マルチプレクサ数を削減でき，平均で 9.9%総面積を削減することを確認できた．今後の課題は，合成フロー全体を通してマルチプレクサ数を削減する手法の構築である．

文　献

[1] 大智輝，戸川望，柳澤政生，大附辰夫，“マルチサイクル配線遅延を考慮したフロアプラン指向高位合成手法，” 情報処理学会 DA シンポジウム'08 論文集，pp. 109–114, 2008.
[2] D. Chen and J. Cong, “Register binding and port assignment for multiplexer optimization,” in *Proc. ASP-DAC 2004*, pp. 68–73, 2004.
[3] D. Chen, J. Cong, and Y. Fan, “Low-power high-level synthesis for FPGA architectures,” in *Proc. International Symposium on Low Power Electronics and Design*, 2003.
[4] J. Cong and J. Xu, “Simultaneous FU and register binding based on network flow method,” in *Proc. DATE 2008*, March 2008.
[5] Y. M. Fang and D. F. Wong, “Simultaneous functional-unit binding and floorplanning,” in *Proc. ICCAD 1994*, pp. 317–321, 1994.
[6] 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.
[7] 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.
[8] T. Kim and X. Liu, “Compatibility path based binding algorithm for interconnect reduction in high level synthesis,” in *Proc. ICCAD 2007*, pp. 435–441, 2007.
[9] 小玉翔，松永祐介，“マルチプレクサ削減を目的としたバインディング改善手法，” 信学技報，vol. 108, no. 22, pp.19–24, 2008.
[10] H. Murata, K. Fujiyosho, and S. Nakatake, “Rectangle-packing-based module placement,” in *Proc. ICCAD 1995*, pp. 472–479, 1995.
[11] 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とタイミング制約を考慮した高位合成手法，” 情報処理学会論文誌，vol. 46, no. 6, pp.1383–1394, 2005.
[12] 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.