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

2009-SLDM-138 (17)
2009 / 1 / 29

フロアプランを考慮した高位合成のための高速なモジュール配置手法

佐藤 亘† 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
E-mail: †satou@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，配線遅延がゲート遅延に対し相対的に増加してきている．そのため，高位合成の段階においてフロアプランを考慮する必要がある．LSI設計プロセスの微細化の一方で，Time to marketの条件が厳しく設計に割ける時間が短くなってきているため，フロアプランを考慮した高位合成を短時間で実行することが望まれる．本稿では，高位合成とフロアプランを繰り返し実行する環境の中で，高位合成の情報を利用した高速なモジュール配置手法を提案する．本手法はイタレーションしている高位合成を対象としてスケジューリング/FUバインディング工程で得られる情報を利用した構築的手法によって高速かつモジュール間の配線遅延を考慮した配置を実行する．計算機実験によって，対象とする高位合成システムに本手法を組み込んだ場合，システム全体の実行時間を平均で98%削減した．

キーワード 高位合成，モジュール配置，レジスタ分散型アーキテクチャ，モジュール間接続情報

Fast Module Placement in Floorplan-aware High-level Synthesis

Wataru SATO†, 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
E-mail: †satou@ohtsuki.comm.waseda.ac.jp

Abstract As device feature size decreases, interconnect delay becomes the dominating factor of total delay. Therefore it is necessary to consider a floorplan in a stage of the high-level synthesis. While device feature size decreases, a condition of the Time to Market is severe, we need to design in a short time. Therefore it is desired to execute the high-level synthesis with floorplan in a short time. In this paper, we propose a high-speed module placement algorithm that used information of the high-level synthesis for the system that execute high-level synthesis and a floorplan repeatedly. This algorithm executes the placement fast that considered interconnect delay between modules by constructive method that used information of a scheduling/FU binding process. We show effectiveness of the proposed algorithm through experimental results.

Key words high level synthesis, floorplan, distributed-register architecture, conected-module infomation

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，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事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ン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，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，配線遅延などのフロアプランを考慮する必要がある．このLSI設計プロセスの微細化の一方でTime to marketの条件が厳しく，回路の設計に割くことのできる時間は短くなってきている．このため高位合成を短時間で実行することが望まれている．

我々はイタレーションを含む高位合成フローを用いることでフロアプランの情報を利用する高位合成システム [9] を提案している．この高位合成システム全体の実行時間は，制御回路合成とモジュール配置工程が実行時間の約 9 割%を占めている．高位合成システムでは制御回路合成とモジュール配置工程の実行時間を短縮することで，高位合成システム全体の実行時間を削減することが出来る．制御回路合成には Design Compiler を用いて合成をしており，見積もり式を算出することで高速化することが可能であると考えられる．モジュール配置工程では Simulated Annealing [4] を用いて最適化する．Simulated Annealing は反復改良法であるため解を得るのに時間がかかる手法である．このためモジュール配置工程の高速化が必要となる．本稿ではモジュール配置工程に着目する．

Simulated Annealing そのものを高速化した手法として，高速 Simulated Annealing 手法が提案されている [1]．しかし Simulated Annealing，高速 Simulated Annealing とも反復改良法であり，より高速に配置するための構築的な配置手法が必要である．さらにこれらの手法は矩形を効率よく詰め込むだけのものであり，高位合成で得られる情報を利用していない．

高速にモジュール配置を行い，高位合成で得られる情報をモジュール配置に活用している研究として [2], [3], [7] がある．[2] は入力が DFG に限定されているため，入力するアプリケーションが限定される．[7] では文献 [8] の配置手法を用いている．これは入力として与えられたスケジューリング済み DFG を各コントロールステップごとに区切り，レジスタ—レジスタ間のサブグラフを作成しこれをもとに配置を行う．この手法では演算器間の接続について考慮されておらず，モジュール間のデータ転送制約違反が多く発生すると考えられる．[3] では，配置に performance-driven placement [6] をもとにした手法を用いている．これはモジュール間の転送に費やすことのできるクロックサイクル数とモジュール間の接続回数を利用した配置手法である．この手法は配置後にスケジューリングを行うため，高位合成フローをイタレーションさせておらず，配置に用いる高位合成の情報はバインディングだけである．このため配置結果はバインディング結果に大きく依存する欠点がある．

[9] 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を考慮した高位合成が提案されている．本稿では，[9] を対象とした高速モジュール配置手法を提案する．本手法はスケジューリング/FU バインディング工程の結果からモジュール間の接続の有無と，モジュール間のデータ転送のスラックを算出する．これらの情報から重み付きグラフを作成し，構築的にモジュールを配置する．本手法により，高速かつモジュール間の配線遅延を考慮した配置が可能となる．

本稿は以下のように構成される．2 章ではレジスタ分散型アーキテクチャを説明する．3 章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システムを説明する．4 章ではモジュール間接続情報グラフを用いた高速なモジュール配置を提案する．5 章では本手法を用いた高位合成手法に対して計算機実験を行った結果を報告する．

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 [9], [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，

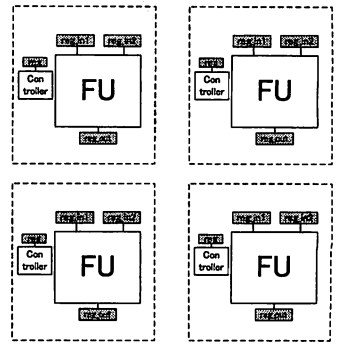

図 1　レジスタ分散型アーキテクチャ．

- 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．

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_{rg}，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_{mx} としたとき，

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

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

$$t_c = \max(t_1, t_2) + m \cdot t_{mx}$$

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．このため，

$$t_c = t_{fu} + t_{rg} + m \cdot t_{mx}$$

となる．

3. レジスタ分散型アーキテクチャを対象とする高位合成 [9]

[9] では分岐処理，クロック周期制約に対応しており，レジスタ分散型をターゲットアーキテクチャ対象とする高位合成手法を確立している．本章では [9] の概要，合成フローを説明する．

[9] 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．目的関数は入力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は面積を最小化する．

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 を含むものとする．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，$F = \{f_i \mid i = 1, 2, \ldots, m\}$ が存在するものとする．各演算器は，面積及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_u の遅延が t_{f_u} で n クロックで演算を処理するときに，f_u がレジスタから入力を読み込み，出力をレ

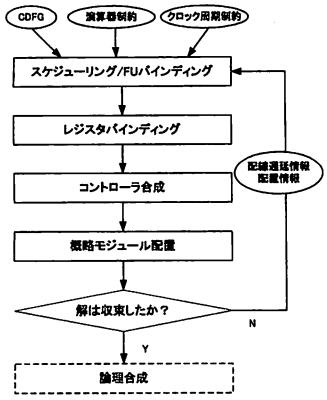

図 2 合成フロー.

ジスタに格納するまでの時間 t_c が，クロック周期 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.

[9] ではレジスタ間データ転送を扱うため，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．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いている.

3.1 スケジューリング/FU バインディング

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.

3.2 レジスタバインディング

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.

3.3 モジュール配置

概略モジュール配置工程では，データ構造に Sequence-pair [5] 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を違反した遅延の総合計としたとき，

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

と計算する．ただし，α, β, γ は任意のパラメータである．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るので，イタレーション毎に SA の初期温度を下げ，イタレーションの回数を重ねるごとにモジュール配置が固定し，解が収束されるようにする．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，

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

とする．モジュール配置が終了した後，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 バインディングの工程へ配線遅延情報，配置結果をそれぞれフィードバックする.

表 1 [9] の各工程の実行時間.

App.	演算器制約		モジュール配置	制御回路合成	その他の工程	合計
DCT	+3*3	時間 [sec]	26.2	413.8	1.1	441.1
		割合 [%]	5.9	93.8	0.2	100
FIR	+3*3	時間 [sec]	40.7	947.8	2.5	991.1
		割合 [%]	4.1	95.6	0.3	100
EWF	+2*1	時間 [sec]	3.2	194.9	56.9	255.0
		割合 [%]	1.2	76.4	22.3	100
EWF3	+3*2	時間 [sec]	22.6	683.2	4.8	710.6
		割合 [%]	3.2	96.1	0.7	100
copy	+3-1<1*5	時間 [sec]	1,178.6	7,873.7	722.8	9,775.1
	AND1 shift2	割合 [%]	12.1	80.5	7.4	100
parker	+2-2	時間 [sec]	26.0	618.1	1.4	645.5
	<1	割合 [%]	4.0	95.8	0.2	100
jacob	+2-1	時間 [sec]	60.0	1,734.5	4.4	1,798.9
	*2 /2	割合 [%]	3.3	96.4	0.2	100

4. モジュール間接続情報を用いた高速なモジュール配置手法

[9] の高位合成システムを C++言語を用いて計算機上に実装し，高位合成システム全体の実行時間を計測した（表 1）(注1)．制御回路の合成(注2)が実行時間の 9 割を占めているが，制御回路の面積，遅延を何らかの方法で見積もることが出来れば，高位合成のイタレーションの度に制御回路を合成をする必要がなくなる．このとき、残る工程のなかでモジュール配置工程が占める時間の割合は約 8 割近くになり，この工程の高速化が必要になる.

[9] のモジュール配置工程では SA を用いて最適化している．SA は反復改良法であるため解を得るのに時間がかかる手法である．Sequence-pair による SA を用いたモジュール配置手法は矩形を効率よく詰め込むだけのものであり，[9] ではコスト関数にクロック周期制約違反した遅延の総合計があるものの，高位合成で得られる情報を十分に配置に利用しているとは言い難い.

本章では，[9] の高位合成システムを対象とした構築的な配置手法を提案する．提案手法はスケジューリング，バインディング結果からモジュール間接続情報グラフを作成し，これをもとにモジュール間のデータ転送を考慮したものである.

4.1 問題定義

モジュール配置を次のように定義する．スケジューリング/バインディング済み CDFG を入力としたとき，クロック周期制約のもと，配置結果を出力する．目的関数はクロック周期制約違反の減少である.

4.2 モジュール間接続情報グラフ

本節ではモジュール間接続情報グラフを定義する．モジュール間接続情報グラフはスケジューリング/FU バインディング済み CDFG を入力とし作成される．モジュール間接続情報グラフは重み付グラフであり，このノード，エッジ，重みはそれぞれモジュール，モジュール間の接続，モジュール間のデータ転送に用いることのできる時間的猶予を表す．任意の 2 モジュール m_i，m_j 間の枝の重みは以下の式で算出する.

$$w(m_i, m_j) = \frac{1}{s(m_i, m_j)}$$

ここで $s(m_i, m_j)$ とはモジュール m_i から m_j へのデータ転送に割りあてることが出来る時間的余裕のことで，以下の式で算出する.

(注1)：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.

(注2)：制御回路の合成は Synopsys 社の Design Compiler を用いて論理合成を行う.

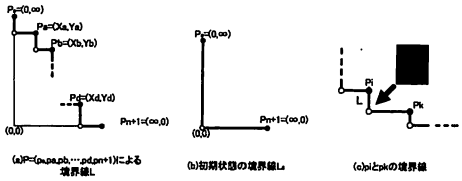

図3 境界線.

$$s(m_i,m_j)=$$

$$\begin{cases}(\lceil\frac{t_{cn}(m_i,m_j)}{T_{clk}}\rceil-\frac{t_{cn}(m_i,m_j)}{T_{clk}})\times T_{clk} & (n_{cs}(m_i,m_j)=0)\\ n_{cs}(m_i,m_j)\times T_{clk} & (n_{cs}(m_i,m_j)\neq 0)\end{cases}$$

ここで $t_{cn}(m_i,m_j)$ は転送元レジスタからモジュール i を通ってモジュール j のレジスタにデータを格納するまでの遅延時間であり，以下の式で算出する．

$$t_{cn}(m_i,m_j)=t_{m_i}+t_{rg}+t_{mx}$$

t_{m_i} はモジュール m_i の演算遅延，t_{rg} はレジスタの読み書き時間，t_{mx} はモジュール m_i から m_j へのデータ転送間に挿入されるマルチプレクサの遅延の合計である．n_{cs} は前回のイタレーションの配置結果からもとめた各モジュール間のデータ転送に必要なクロック数を保持しておくデータ転送制約テーブルを表し，$n_{cs}(m_i,m_j)$ はモジュール m_i から m_j へのデータ転送に必要なクロック数を表す．モジュール接続情報グラフは高位合成の情報をモジュール間の接続，データ転送に必要なクロック数として保持している．

4.3 モジュール配置手法

提案手法では配置手法に境界線法 [10] の一部を用い，各モジュールに code と order を割り当てることで配置を行う．

4.3.1 境界線法

境界線法は各方形に code と order が割り当てられている時，各方形を境界線に沿って配置する手法である．code，order は方形の順列である．code は既配置方形との相対的な配置位置を決定し，order は方形を配置する順序を決定する．方形 r が順列 code（order）において前から i 番目に現れるとき，方形 r の code 値（order 値）を i とし，$\mathrm{code}(r)=i$ ($\mathrm{order}(r)=i$) と表す．境界線 L とは方形の数を n 個としたとき，点系列 $P=<p_0,p_a,p_b,...,p_n,p_{n+1}>$ によって定義される．P 内の各点の座標 $p_0=(0,\infty)$, $p_a=(x_a,y_a)$, $p_b=(x_b,y_b)$,..., $p_n=(x_n,y_n)$, $p_{n+1}=(\infty,0)$ は $0<x_a<x_b<\cdots<x_n<\infty$, $\infty>y_a>y_b>\cdots>y_n>0$ を満たす．このとき境界線 L は図 3(a) のような右下がりの折れ線を表す．

方形の順列 code，order が与えられた時，方形配置を求める手順を説明する．初期状態の境界線 L_0 は $P_0=<p_0,p_{n+1}>$ とする（図 3(b)）．order の先頭の方形 r_m の code 値 j は $j=code(r_m)$ である．次に $i<j<k$ を満たす点系列 P 内の連続する 2 点 p_i, p_k を求め，方形 r_m を境界線 L 上の 2 点 p_i, p_k で定まる境界線に接するように左下詰めに配置する（図 3(c)）．次に境界線の更新手続きを行う．点 $p_j=(x_j,y_j)$ の座標を，配置した方形 r_m の右上の角の座標とし，p_j を点系列 P 内の 2 点 p_i, p_k の間に挿入する．点系列 P 内において $i<j$ を満たす全ての点 $p_i=(x_i,y_i)$ に対し，$y_i\leq y_j$ を満たす場合，点 p_i を P から削除し，点系列 P 内において $k>j$ を満たす全ての点 $p_k=(x_k,y_k)$ に対し，$x_k\leq x_j$ を満たす場合，点 p_k を P から削除する．order の順にすべての方形に対し同じ手続きをとることで方形の配置を行う．

4.3.2 提案モジュール配置手法

提案手法ではモジュール間接続情報グラフから 2 ノードを選択し，選択されたノードに対応するモジュールに code，order を割り当てることで配置を行う．選択の仕方はモジュール間接続情報グラフの最も重みの大きい枝の両端の 2 ノードを選択する．

code，order が割り当てられていないモジュールを単モジュール，code,order が割り当てられているモジュールをモジュール群と呼ぶ．モジュール群は複数個のモジュールからなり，構成するモジュールの相対的な位置関係を表すために各モジュール群はそれぞれ code，order をもつ．モジュール群 G の code を C_G，order を O_G と表す．選択された 2 ノードに対応するモジュールの種類によって，code，order の割り当て方は以下の 3 通りに場合分けできる．

- 単モジュール同士の場合

単モジュール m_i, m_j が選択された場合，この 2 モジュールからなるモジュール群 G_0 を新たに作り，G_0 の code，order を m_i, m_j に割り当てる．選択された 2 モジュール m_i, m_j の横と縦の辺の長さを m_{ix}, m_{iy}, m_{jx}, m_{jy} とする．このとき，

$C_{G_0}=<m_j,m_i>$, $O_{G_0}=<m_i,m_j>$ $(m_{ix}>m_{iy},m_{jx},m_{jy})$

$C_{G_0}=<m_i,m_j>$, $O_{G_0}=<m_i,m_j>$ $(m_{iy}>m_{ix},m_{jx},m_{jy})$

$C_{G_0}=<m_i,m_j>$, $O_{G_0}=<m_j,m_i>$ $(m_{jx}>m_{jy},m_{ix},m_{iy})$

$C_{G_0}=<m_j,m_i>$, $O_{G_0}=<m_j,m_i>$ $(m_{jy}>m_{jx},m_{ix},m_{iy})$

とする．上記のように code，order を割り当てることで，モジュール群 G_0 内での相対的な位置関係は最も長い辺をもつモジュールが左下詰めとなる．

- 単モジュールとモジュール群の場合

選択された 2 モジュール m_i, m_j のうち，m_i がモジュール群 G_i に属しており，m_j が単モジュールとする．このとき，単モジュール m_j をモジュール群 G_i に併合する．$C_{G_i}=<c_1,c_2,...,m_i,...,c_k>$, $O_{G_i}=<o_1,o_2,...,m_i,...,o_k>$ とすると，単モジュールを併合したモジュール群 G_i の order を $O_{G_i}=<o_1,o_2,...,m_i,...,o_k,m_j>$ と更新する．G_i の code は，$C_{G_i1}=<c_1,c_2,...,m_i,m_j,...,c_k>$，もしくは $C_{G_i2}=<c_1,c_2,...,m_j,m_i...,c_k>$ の 2 つを候補とする．選択された 2 モジュールの code 値を近づけることで，選択された 2 モジュールの相対的な位置関係が近くになる．

選択された 2 モジュール m_i, m_j 間のデータ転送に必要な時間 t_w と，C_{G_i1}, C_{G_i2} の比較を行うためのコストを算出するために，O_{G_p} と C_{G_i1}, C_{G_i2} にもとづいて更新されたモジュール群 G_i の仮配置を行う．t_w が以下の式を満たさない場合，クロック周期制約を満たさない．

$$t_w(m_i,m_j)<s(m_i,m_j)$$

ここで $s(m_i,m_j)$ は m_i, m_j 間の転送に用いる事の出来る時間猶予である．クロック周期制約を違反しない C_{G_i1}, C_{G_i2} のうち，最もコスト関数の値の小さい候補を G_i の code とする．コスト関数は以下の式で定める．

$$cost=\frac{a_{BB}}{a_{total}}+\frac{w}{w_{Max}}$$

ここで a_{BB} はモジュール群 G_i のデッドスペースを含む回路の総面積，a_{total} は G_i を構成するモジュールの面積の総合計，w は G_i 内の各モジュールを結ぶ総配線長，w_{Max} は $(height+width)\times w_{num}$ (w_{num} は配線本数) である．

選択された 2 モジュールが C_{G_i1}, C_{G_i2} の 2 つともクロック周期制約を満たさない場合，モジュール m_j とモジュール群 G_i の併合を行わず，データ転送制約テーブル $n_{cs}(m_i,m_j)$ の値とモジュール間接続情報グラフの重みを変更する．データ転送制約テーブルはクロック数であらわされているため，$n_{cs}(m_i,m_j)=0$ の場合，値を 1 に変更する．$n_{cs}(m_i,m_j)\neq 0$ の場合，値を 1 増加する．モジュール間接続情報グラフの重みは，$n_{cs}(m_i,m_j)$ の値を変更したことで $s(m_i,m_j)$ の値も変わるので，$w(m_i,m_j)$ を 再計算する．このことにより，クロック周期制約を満たさないモジュールの配置を回避し，他の制約を満たす可能性のあるモジュールの配置を可能にする．

- モジュール群同士の場合

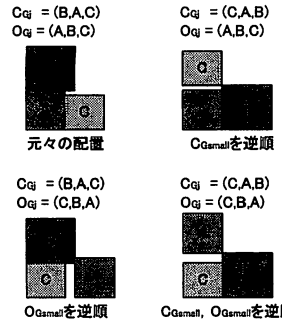

図 4 code, order の逆順によるグループの回転.

選択された 2 モジュール m_i, m_j が異なるモジュール群に属している場合，2 モジュール群のうちデッドスペースを含む回路の面積が大きい方のモジュール群 G_i に，小さいほうのモジュール群 G_j を併合することで，モジュール群の併合を行う．$C_{G_i} = < c_{i1}, ..., c_{il} >$, $O_{G_i} = < o_{i1}, ..., o_{il} >$, $C_{G_j} = < c_{j1}, ..., c_{jm} >$, $O_{G_j} = < o_{j1}, ..., o_{jm} >$ とすると，併合後のモジュール群 G_i の order は $O_{G_i1} =< o_{i1}, ..., o_{il}, o_{j1}, ..., o_{jm} >$, もしくは $O_{G_i2} =< o_{i1}, ..., o_{il}, o_{jm}, ..., o_{j1} >$ を候補とし，G_i の code は

C_{G_i1}=<c_{i1},...,c_{il},c_{j1},...,c_{jm}>, C_{G_i2}=<c_{j1},...,c_{jm},c_{i1},...,c_{il}>,

C_{G_i3}=<c_{i1},...,c_{il},c_{jm},...,c_{j1}>, C_{G_i4}=<c_{jm},...,c_{j1},c_{i1},...,c_{il}>

を候補とする．O_{G_i} の後ろに O_{G_j} を結合することで，モジュール群 G_i の上，もしくは左にモジュール群 G_j を配置することになる．code, order を併合する際，C_{G_j}, O_{G_j} の並びを正順，逆順に変換する事によってモジュール群 G_j 内で回転を行い，複数の配置候補を得る（図 4 参照）．

C_{G_i1}, C_{G_i2}, C_{G_i3}, C_{G_i4} と O_{G_i1}, O_{G_i2} の組み合わせにもとづいて更新されたモジュール群 G_i の仮配置を行い，選択された 2 モジュール間がクロック周期制約を満たしている組み合わせの中で，最もコスト関数の小さい候補を G_i の code, order とする．コスト関数は単モジュールとモジュール群の場合の時と同様である．選択された 2 モジュール間でいずれの組み合わせでもクロック周期制約を満たせない場合，モジュール群 G_i, G_j の併合を行わず，$n_{cs}(m_i, m_j)$ の値とモジュール間接続情報グラフの重みを単モジュールとモジュール群の場合の時と同様に変更する．

4.3.3 グラフの更新

モジュール間接続情報グラフの最も重みの大きい枝の両端の 2 ノードを選択され，対応する 2 モジュールに code, order が割り当てらる，もしくは code, order の併合がなされた場合，選択された 2 ノードを併合する．併合される 2 ノードに接続されている全てのエッジは，併合後もどちらのノードに接続されていたかを情報として保持しておく．モジュール間接続情報グラフのすべてのノードが選択されたとき，すべてのモジュールが 1 つのモジュール群に属している状態にあり，モジュール間接続情報グラフのノードはすべて併合され 1 つになる．

4.4 アルゴリズム

モジュール配置のアルゴリズムを図 5 に示す．モジュール間接続情報グラフの重みの分母はデータ転送に割り当てる事の出来る時間を表したものであり，重みの値が大きいことはデータ転送に時間的猶予がないことを意味する．このことから，モジュール間接続情報グラフの重みの値の大きい 2 モジュールを近い位置に配置していく．データ転送の時間的猶予が少ないモジュールから配置をしていくことで回路全体のクロック周期制約違反が小さくなると考えられる．すべてのモジュールが配置されたら配置工程を終了し，各モジュールの座標を算出する．

モジュール配置を図 6 の例を用いて説明する．図 6(a) のよ

Step 1 スケジューリング/FU バインディング結果をもとにモジュール間接続情報グラフを作成する．
Step 2 モジュール間接続情報グラフから重みの最も大きい枝の両端のノードに対応するモジュールに対し code, order を割り当てる．グラフにノードが 1 つしかない場合モジュール配置を終了する．
Step 3 2 モジュールへの割り当てを行い，その 2 モジュール間でクロック周期制約違反を起こさなければ選択された 2 ノードを併合し **Step 2** へ．クロック周期制約違反を起こす場合 **Step 4** へ．
Step 4 2 モジュールへの割り当てを取り消し，そのモジュール間の n_{cs} の値が 0 の場合 1 に変更，0 でない場合 1 増加し，モジュール間接続情報グラフの枝の重みを再計算し，**Step 2** へ．

図 5 提案配置アルゴリズム.

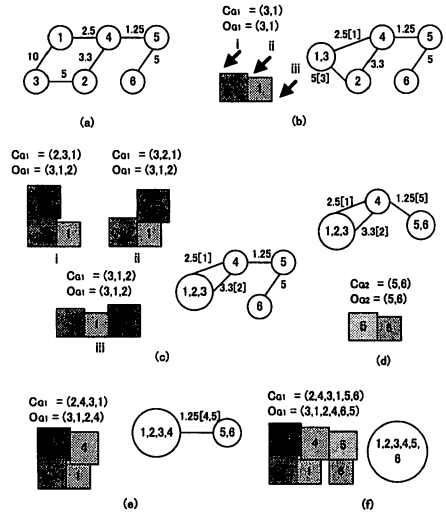

図 6 モジュール配置の具体例.

うなモジュール間接続情報グラフが与えられた時，最初に選択される 2 ノードは最も重みの大きい枝の両端のノードの 1 と 3 であり，このノードに対応する 2 モジュールは単モジュールなので，この 2 モジュールをモジュール群とし，code, order が割り当てられる（図 6(b)）．次に選択される 2 ノードは 2 と 3 で，モジュール 3 は先ほどのモジュール群に属しているので，単モジュールとモジュール群の併合となる．モジュール 2 は図 6(b) の 3 か所の矢印のいずれかに配置する事が出来る．それぞれの場合の配置結果と code, order は図 6(c) である．このとき，モジュール 2 の code がモジュール 3 の code の前後どちらかに割り当てられているのは，6(c) の と である．コストを計算した結果，ここでは 6(c) の が選択されたとする．次にノード 5 と 6 が選択されるが，これは先に述べた単モジュール同士の場合である（図 6(d)）．次にノード 2, 4 が選択され，単モジュール 4 とモジュール群の併合が行われる（図 6(e)）．最後にノード 4 と 5 が選択される．これはモジュール群同士の併合になる（図 6(f)）．

4.5 計算複雑度

提案するモジュール配置手法の計算複雑度を求める．提案手法はモジュール間接続情報グラフを作成した後，配置するモジュールの選択，モジュールの配置，モジュール間接続情報グラフの更新の，3 工程のループに入り，終了する．ここでモジュール数を N，モジュール間接続情報グラフの枝数を E とすると，モジュールの選択に $O(E)$，配置工程では $O(1)$ となっている．図 5 の **Step2** から **Step4** のループの回数は最悪の場合，一回のループに最大で $O(N)$ かかり，それを N 個のモジュールに対して行うため $O(N^2)$ となる．以上のことを踏まえると計算複雑度は $O(N^2E)$ となる．

表 2 実験結果.

App.	手法	コントロールステップ数	面積 [um^2]	イタレーション回数	実行時間 (sec) モジュール配置工程	制御回路合成	その他の工程	合計
DCT	[9]	14	47151	3	26.2	413.8	1.1	441.1
	提案手法	14	35040	3	0.2	415.3	1.1	416.6
FIR	[9]	33	32374	6	40.7	947.8	2.5	991.1
	提案手法	33	28362	4	0.3	649.5	1.7	651.5
EWF	[9]	21	15762	3	3.2	194.9	56.9	255.0
	提案手法	21	15390	3	0.1	209.3	0.8	210.2
EWF3	[9]	52	30453	5	22.6	683.2	4.8	710.6
	提案手法	52	30690	6	0.8	726.7	5.2	732.7
jacobi	[9]	45	41548	7	60.0	1734.5	4.4	1798.9
	提案手法	45	40872	3	0.2	491.7	1.3	493.2
parker	[9]	7	7405	6	26.0	618.1	1.4	645.5
	提案手法	7	10488	2	0.1	229.5	0.5	230.1
copy	[9]	136	143319	24	1178.6	7873.7	722.8	9775.1
	提案手法	152	165923	17	37.5	5576.1	621.3	6234.9

表 3 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

5. 計算機実験結果

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3)(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 3 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 1.7[ns] とした．

対象としたアプリケーションは以下の 7 つである．

- DCT：ノード数　48（加算器 3，乗算器 3）
- 7 次 FIR フィルタ：ノード数　75（加算器 3，乗算器 3）
- EWF：ノード数　34（加算器 2，乗算器 1）
- EWF3：ノード数　102（加算器 3，乗算器 2）
- parker：ノード数　22，分岐あり（加算器 2，減算器 3，比較器 1）
- jacobi：ノード数　48（加算器 2，減算器 1，乗算器 2，除算器 2）
- COPY：ノード数　378，分岐あり（加算器 3，減算器 1，比較器 1，乗算器 5，AND 演算 1，シフタ 2）

実験結果を表 2 に示す．実験結果より，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 [9] と比較し，コントロールステップ数が最大で 12% 面積が最大で 42%増加したがそれ以外では平均して同程度の配置結果を得ながら，モジュール配置工程の実行時間を平均して 98%削減できた．

6. むすび

本稿ではフロアプランを考慮した高位合成のための高速なモジュール配置手法を提案した．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 [9] と比較し，コントロールステップ数が最大で 12% 面積が最大で 42% 増加したがそれ以外のでは平均して同程度の配置結果を得ながら，モジュール配置工程の実行時間を平均して 98% 削減できた．今後の課題は配置結果の面積の削減，コントロールステップ数の増加を抑える改良，制御回路合成に代わる見積もり式の算出による高速化である．

(注3)：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．

文　献

[1] T. C. Chen and Y. W. Chang, "Modern floorplanning based on fast simulated annealing," in *Proc. the 2005 international symposium on Physical design,* pp. 104–112, 2005.

[2] W. E. Dougherty and D. E. Thomas, "Unifying behavioral synthesis and physical design," in *Proc. DAC,* pp. 756–760, 2000.

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

[4] S. Kirkpatric, C. D. Gelatt Jr. and M. P. Vecchi, "Optimization by simulated annealing," *Science* , Volume 220, no. 4598, pp. 671–680, 1983.

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

[6] I. Lin and D. H. C. Du, "Performance-driven constructive placement," in *Proc. DAC,* pp. 103–106, 1990.

[7] V. G. Moshnyaga and K. Tamaru, "A placement driven methodology for high-level synthesis of sub-micron ASIC's," in *Proc. ISCAS,* pp. 572–575, 1996.

[8] Y. Mori, V. G. Moshnyaga, H. Onodera and K. Tamaru, "A performance-driven macro-block placer for architectural evaluation of ASIC designs," in *Proc. Eighth Annual IEEE International ASIC Conference and Exhibit,* pp. 233–236, 1995.

[9] 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.

[10] 長尾明，澤卓，重弘裕二，白川功，神戸尚志，"方形パッキング法の一算法，" 電子情報通信学会論文誌, Vol.J81-A, No.10, pp. 1362–1371, 1998.

[11] 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，" DA シンポジウム'08 論文集, pp. 109–114, 2008.