演算チェイニングの候補列挙・選択アルゴリズムを用いたフロアプラン指向高位合成手法

寺田 晃太朗[1,a)]　柳澤 政生[1]　戸川 望[1]

概要：LSI 設計では高性能かつ低面積である回路を短期間で製造することが求められ，設計の上位工程に相当する高位合成は設計コストを削減する有用な技術である．一方，半導体の微細化に伴い配線遅延がゲート遅延と比較して相対的に増大している．従来の高位合成手法のように配線遅延を明示的に計算しない手法では，配線遅延として過多なタイミングマージン挿入を想定することになり，生成回路の性能向上を妨げる．そこで高位合成でも配線遅延の影響を抑えるためにフロアプランを意識する必要がある．本稿では，RDR アーキテクチャを対象とした高位合成段階でフロアプランを設計する演算チェイニングを用いた高位合成手法を提案する．提案手法は各演算に対して演算チェイニング可能な演算クラスタを列挙し，最適な候補を選択しながらスケジューリング・バインディングを実行することで，レイテンシの削減を目指す．候補の列挙にはクリティカル長に基づく優先度を用いてレイテンシを増大させるような候補を排除することでレイテンシの増加を防ぎ最適な演算チェイニング構成を実現する．計算機実験により提案手法はレジスタ数を削減しながらレイテンシを最大 23.5% 削減する手法であることを確認した．

A Floorplan-Driven High-Level Synthesis Method Based on Operation Chainings Enumeration-and-Selection

Kotaro Terada[1,a)]　Masao Yanagisawa[1]　Togawa Nozomu[1]

Abstract: In deep-submicron era, interconnection delays are not negligible even in high-level synthesis, and RDR (Regular-Distributed-Register) architecture has been proposed to cope with this problem. In this paper, we propose a high-level synthesis algorithm based on enumeration-and-selection of operation chainings. Our algorithm enumerates operation chaining candidates before performing scheduling and binding. We find out optimal ones for RDR architectures while scheduling to minimize latency. Experimental results show that our algorithm reduces the latency by up to 23.5% compared to the conventional algorithm.

1. はじめに

半導体の微細化技術により，高集積 LSI の製造が可能となり，大規模なシステムを LSI 上に実装することが可能となった．LSI 設計では高性能かつ低面積である回路を短期間で製造することが求められる．設計の上位工程に相当し，抽象度の高い C/C++言語等で記述された動作記述からレジスタ転送レベル（RTL）回路を合成する高位合成は設計コストを削減する有用な方法である．

一方、プロセスの微細化が進むにつれ配線遅延がゲート遅延と比較し相対的に増大している．配線遅延を考慮しない高位合成では配線遅延のために過剰なマージンの挿入は避けられず現実的ではない．高位合成段階で配線遅延を考慮することが求められる．レジスタ分散型アーキテクチャを用いることが解決方法の 1 つである．レジスタ分散型アーキテクチャの一種として RDR アーキテクチャ（Regular Distributed Register Architecture）がある [2].

本稿で提案する高位合成手法は RDR アーキテクチャ [2] を対象とする．RDR アーキテクチャを図 1 に示す．RDR アーキテクチャはレジスタ分散型アーキテクチャをとっており，従来のレジスタ集中型アーキテクチャに対し，チップを複数のクラスタに分割し各クラスタに演算器とレジスタを配置することで配線遅延の相対的拡大問題を解決し，チップ上のマルチサイクル通信を実現する．各島はサイズが規格化されているため，RDR アーキテクチャを対象と

[1] 早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻
[a)] kotaro.terada@togawa.cs.waseda.ac.jp

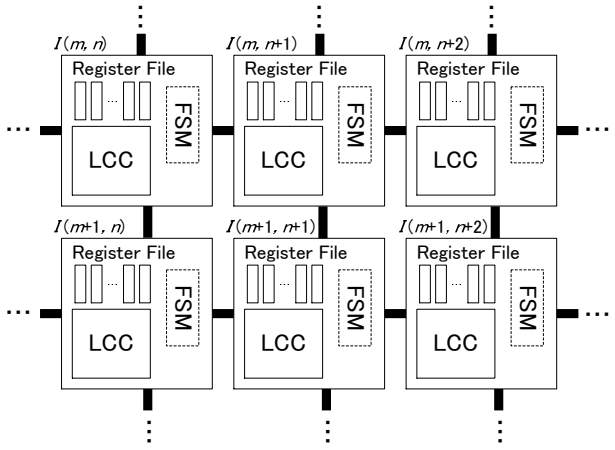


図 1 RDR アーキテクチャ [2].

した高位合成では高位合成段階で配線遅延を見積もることが容易である．各島にレジスタ，演算器，コントローラ（FSM）を配置する．RDR アーキテクチャを対象とした高位合成手法として MCAS [2] が提案されている．

高位合成が対象とするアーキテクチャはデータパスと有限状態機械（FSM）で構成される．FSM はコントローラの役割を果たし，クロック信号で状態遷移する．データパスは FSM の状態遷移に応じてデータ演算する．高位合成におけるスケジューリング問題は入力アプリケーションの演算をコントロールステップ（クロックサイクル）に割り当てる問題，バインディング問題は演算を実行する演算器を決定する問題である．

一般に問題を簡単にするために高位合成は入力で与えられたアプリケーションの演算を 1 コントロールステップ（クロックサイクル）に割り当てる．しかし実際の演算を実行する演算器は演算の種類によって遅延が異なる．この問題を解決する 1 つの方法が演算チェイニングである．演算チェイニングは複数の連続した演算を少ないコントロールステップに割り当てる手法である．演算チェイニングを適用することで回路のレイテンシの削減，レジスタ数およびレジスタに付随するマルチプレクサ（MUX）数の削減が期待できる．演算チェイニングを用いた高位合成手法に関して様々な研究がされているが，いずれもレジスタ集中型アーキテクチャを対象とした手法である [1,4,5].

[3] は FPGA 等のプラットフォームを対象とした RDR アーキテクチャのレジスタをレジスタ・ファイルに置き換えた DRFM アーキテクチャを提案している．[3] には DRFM アーキテクチャを対象とした高位合成手法に演算チェイニングを適用する方法が記述されている．しかし，DRFM アーキテクチャを対象とした高位合成手法は配線遅延を明示的に計算していなく，演算チェイニング適用による効果の有無は実験評価されていない．演算チェイニングを適用したレジスタ分散型アーキテクチャを対象とした高位合成手法は筆者らの知る限り存在しない．

筆者らはこれまで [7,8] で RDR アーキテクチャを対象とした高位合成手法で演算チェイニングを用いた合成手法を提案した．[8] では 2 演算間に MCD と呼ばれる演算チェイニング可能な最大島間距離を与え，それに基づいてスケジューリングすることで演算チェイニングを構築し，レイテンシを削減した．[7] ではパスに基づく候補を列挙しスケジューリング・バインディングすることで，演算チェイニングを構成する演算ノード数を 2 個以上に拡張した．

本稿では，RDR 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を提案する．提案手法は演算クラスタの候補列挙・選択アルゴリズムに基いて演算チェイニングを実現するスケジューリング・バインディングが実行され，提案手法によって合成される回路はレイテンシおよびレジスタ数を削減する．2 章で RDR アーキテクチャを対象とした演算クラスタに基づくチェイニングを用いた高位合成問題を定義する．3 章で演算クラスタの候補列挙・選択アルゴリズムに基づく演算チェイニングを実現する高位合成手法を提案し，4 章で提案アルゴリズムの計算機実験結果を示す．5 章で結論を述べる．

2. 問題の定式化

本章では，コントロールデータフローグラフ，RDR アーキテクチャ，RDR アーキテクチャ上の多段演算チェイニングを定式化し，RDR アーキテクチャを対象とした多段演算チェイニングを用いた高位合成問題を定義する．

2.1 コントロールデータフローグラフ・演算器

高位合成の入力の 1 つは動作記述である．動作記述はコントロールデータフローグラフ（CDFG）またはデータフローグラフ（DFG）で表すことができる．本稿では簡単のために高位合成の入力に DFG を使用する．なお本稿で提案する手法は，CDFG にも容易に拡張できる．

DFG は無閉路有向グラフ（DAG）である．DFG は演算ノードの集合を V，データ依存を表すエッジの集合を E とすると，$G = (V, E)$ と表せる．

演算器 fu の遅延を $d_f(fu)$，レジスタの読み出し・書き込み遅延を d_{reg} とする [*1]．演算が演算器 fu を使用し，データをレジスタから読み出してから再びレジスタへ書き込むまでの演算の合計遅延を $d_{sum}(fu)$ と表す．演算合計遅延 $d_{sum}(fu)$ は

$$d_{sum}(fu) = d_{reg} + d_f(fu) \quad (1)$$

と表される．

演算 v を実行可能な演算器の集合を $EFU(v)$ とする．例えば，v が加算であり，fu_1 が加算器，fu_2 が加減算器，fu_3 が乗算器のとき，$EFU(v) = \{fu_1, fu_2\}$ である．

DFG 上で先行ノードが存在しないノードを PI (Primary Input) と呼び，PI の集合を PI とする．後続ノードを持たないノードを PO (Primary Output) と呼び，PO の集合を PO とする．

DFG 上の演算クラスタを定義する．DFG G 上のサブグラフ G' は n $(n \geq 1)$ 個の演算ノード $v_1, v_2, \cdots, v_n$ で構成されるとする．これらのノードが連結であるとき，G' は DFG G 上の演算クラスタであるという．演算クラスタ C 内での PI を $PI(C)$，PO を $PO(C)$ と表す．

2.2 対象 RDR アーキテクチャ

RDR アーキテクチャ [2] は配線遅延を考慮したマルチサイクル通信を実現するアーキテクチャである．チップを複数の規則的な島に分割し各島にレジスタを分散させる．島サイズが規則的であるため配線遅延の予測が容易である．

各島はレジスタ，演算器，コントローラ（FSM）の要素から構成される．演算器の入出力はレジスタに接続される．レジスタはその島のローカルストレージである．FSM は状

[*1] 演算遅延とレジスタ遅延は前段に付随する MUX 遅延を含む.

態遷移によって演算器とレジスタを制御するコントローラである．各島はグローバル配線で結線される．RDR アーキテクチャは演算器の近くにレジスタを配置するため，レジスタ演算器間の配線遅延を減らすことができる．

各島は正方形とし，チップを $M \times N$ の島に分割する．m 行 n 列の島を $I(m,n)$ と表す．ただし，$1 \leq m \leq M$，$1 \leq n \leq N$ である．島 $i_1 = I(m_1,n_1)$ から島 $i_2 = I(m_2,n_2)$ までのデータ転送時間 $D_c(i_1,i_2)$ は以下の式で定義される．

$$D_c(i_1,i_2) = C_d \cdot (|m_1 - m_2| + |n_1 - n_2|) \qquad (2)$$

配線中にはバッファが挿入され配線遅延は距離に比例すると仮定する．C_d は配線遅延係数である．

演算器 fu_1 が島 $i_1 = I(m_1,n_1)$ に配置されているとする．演算器 fu_1 の出力データを島 $i_x = I(m_x,n_x)$ に配置されている演算器 fu_x へ転送することを考える．クロック周期を T_{clk} とする．

$T_{clk} \geq D_c(i_1,i_x) + d_{sum}(fu_1)$ のとき：
演算器 fu_1 で演算の後，1 ステップ内で島 i_x へ転送することが可能であるから，演算実行と同じステップで島 i_x のレジスタへ転送する．

$T_{clk} < D_c(i_1,i_x) + d_{sum}(fu_1)$ のとき：
演算器 fu_1 で演算の後，同一ステップで転送することが不可能であるから，一旦島 i_1 のレジスタに格納した後 $\lceil D_c(i_1,i_x)/T_{clk} \rceil$ ステップかけて島 i_x のレジスタへ転送する．

RDR アーキテクチャ上の島容量を定義する．各島は容量制限 C を持ち，演算器 fu は面積コスト c_{fu} を持つ．ある島に配置される演算器の面積コスト c_{fu} の総和は C を超えないように演算器を配置する．

2.3 演算クラスタとスケジューリング

DFG G 上のパス P が n 個の演算 $v_1 \to v_2 \to \cdots \to v_n$ で構成され，演算 $v_i \in P$ が演算器 fu_i に割り当てるとするとき，パス P の実行遅延 $d_{path}(P)$ は

$$d_{path}(P) = d_{reg} + d_f(fu_1) + \sum_{v_i \in P'} \{d_f(fu_i) + D_c(i_i, i_{i+1})\} \qquad (3)$$

と計算される．ただし，P' は P から v_1 を取り除いたパス，演算器 fu_i が配置されている島を i_i とする．演算 $v \in V$ から $u \in V$ に至るパスの集合を $PATH(v,u)$ と表す．ただし，$v \neq u$ かつ v はトポロジカルオーダーで u に先行する．

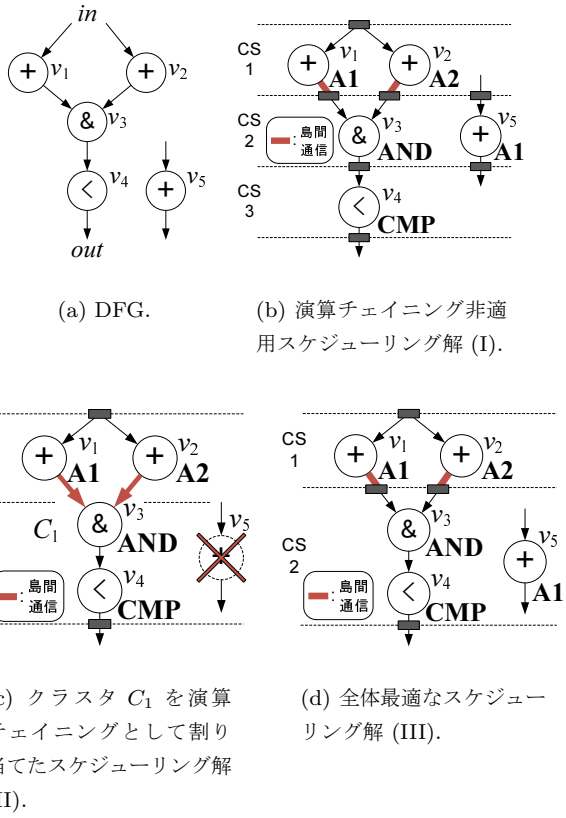
演算 $v_i \in C$ を実行する演算器を fu_i とするとき，演算クラスタ C は

$$s(C) = \max_{p \in PATH(i,o)} \{d_{path}(p)\} \qquad (4)$$

ステップかけて実行される．ただし，$i \in PI(C), o \in PO(C)$ である．$PI(C)$ 内の演算を実行する演算器はその演算器が配置されている島のレジスタからデータが与えられる．$PO(C)$ 内の演算の出力データは 2.2 節と同様に演算器が配置される島のレジスタに保存されるか，後続の演算が利用する演算器が配置される島へ転送される．

以上をもとに演算クラスタのスケジューリング・バインディングを次のように定義する．

定義 1. 演算クラスタ C を $s(C)$ コントロールステップに割り当てる（スケジューリング・バインディングする）ことである． □



(a) DFG.

(b) 演算チェイニング非適用スケジューリング解 (I).

(c) クラスタ C_1 を演算チェイニングとして割り当てたスケジューリング解 (II).

(d) 全体最適なスケジューリング解 (III).

図 2 (a) の DFG のスケジューリング解の比較．(b) は演算チェイニングを用いない場合で 3 ステップ要する．(c) は C_1 を演算チェイニングとして割り当てることでレイテンシ・レジスタ数削減を試みるが v_5 を並列に実行できない．(d) は C_1 の内，v_3 と v_4 をチェイニングし，全体で 2 ステップで実行できる．

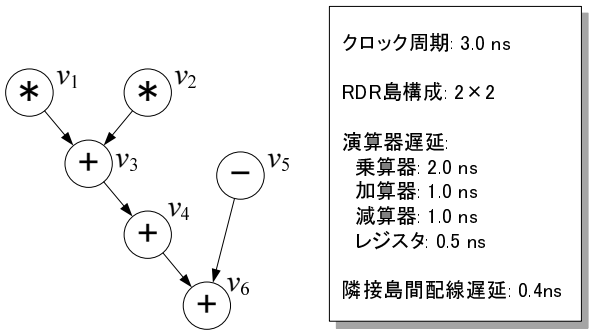
演算クラスタのスケジューリング・バインディングは以下の条件を満たす．

(1) 演算クラスタ内の演算を実行する演算器はその演算クラスタの占有ステップの間は演算器を使用し，他のクラスタ内の演算はその演算器を使用できない．
(2) 演算クラスタにデータを与えるレジスタはその演算クラスタの占有ステップの間はデータを与え続ける．

例 1. 図 2 に例を示す．島 i_1 と島 i_2 があり，島 i_1 には加算器が 2 個（A1, A2），島 i_2 には論理積演算器（AND）と比較器（CMP）が配置されていると仮定する．今，クロック周期は 2.0 ns，演算器遅延は加算器 1.6 ns，論理積演算器 0.5 ns，比較器 0.5 ns とし，隣接島間配線遅延は 0.3 ns とする．図 2(a) に示す DFG 上の演算クラスタ $C_1 = \{v_1, v_2, v_3, v_4\}$ とすると，演算クラスタ C_1 は演算 v_1, v_2, v_3, v_4 がそれぞれ演算器 A1，A2，AND，CMP にバインディングされたとき，2 ステップで実行することが可能である． □

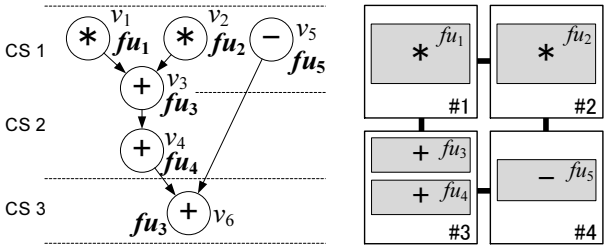
2.4 問題定義

以上をもとに，RDR アーキテクチャを対象とした演算クラスタに基づくチェイニングを用いた高位合成問題を次のように定義する．



(a) DFG.　(b) 制約（クロック周期, RDR 仕様, 演算器情報）.

図 3　演算チェイニングを用いた高位合成問題に対する入力例.



(a) スケジューリング・バインディング済 DFG.　(b) 演算器フロアプラン.

図 4　演算チェイニングを用いた高位合成問題の出力例.

定義 2. RDR アーキテクチャを対象とした演算クラスタに基づくチェイニングを用いた高位合成問題とは，DFG $G = (V, E)$，クロック周期 T_{clk}，RDR 島数，演算器情報を入力とし，スケジューリング・バインディング済み DFG $G' = (V', E')$ および演算器の島へのフロアプランを出力とする．このとき，演算クラスタをチェイニングとしてスケジューリング・バインディングすることを許容して，入力情報の制約のもとでレイテンシを最小化する．□

例 2. 入力と出力を図 3 と図 4 にそれぞれ示す．図 3(a) は入力 DFG であり各ノードの右に示す記号 v_i はその演算ノードを表す文字である．図 3(b) にその他の入力制約を項目別にまとめる．スケジューリング・バインディング済みの出力 DFG を図 4(a) に示す．水平の破線での区切りが 1 ステップを表し，各ノードの横に示す fu_i はその演算ノードを実行する演算器を表す．演算クラスタ $\{v_1, v_2, v_3, v_4\}$ は演算チェイニングとしてステップ 1 とステップ 2 に渡って割り当てられる．同様に $\{v_5, v_6\}$ はチェイニングされてステップ 3 で実行される．チェイニングされた演算間のデータはレジスタに格納されないが，v_1，v_2 の入力データを保持するレジスタはステップ 1 とステップ 2 の間データを与え続ける．図 4(b) は演算器フロアプランである．図 4(a) のスケジューリング・バインディング結果はフロアプランに基づいた島間配線遅延も考慮している．□

3. 提案手法

本章では RDR アーキテクチャを対象とした演算クラスタに基づくチェイニングを用いた高位合成手法を提案する．アルゴリズムの方針を述べた後，全体の合成フローを示し，各ステップを記述する．

3.1 演算クラスタ列挙・選択アルゴリズムの方針

図 2(b)，図 2(c)，図 2(d) は図 2(a) の入力 DFG を異なる演算クラスタに分割したときのスケジューリング解を比較した図である．図 2(b) に示すスケジューリング解 (I) は演算チェイニングを用いない場合，すなわち DFG を $\{v_1\}$，$\{v_2\}$，$\{v_3\}$，$\{v_4\}$，$\{v_5\}$ の 5 つのクラスタに分割した場合である．このとき実行に 3 ステップ，レジスタ数は 3 個（島 i_1 に 1 個と島 i_2 に 2 個）要する．

図 2(c) に示すスケジューリング解 (II) は DFG を $C_1 = \{v_1, v_2, v_3, v_4\}$ と $\{v_5\}$ に分割した場合である．C_1 は例 1 で説明したようにステップ 1 とステップ 2 にまたがって割り当てられ演算チェイニングが構築される．これによってクラスタ C_1 の実行ステップ数と使用レジスタ数は削減されたが，2.3 節の条件 (1) よりクラスタ $\{v_5\}$ 内の加算がステップ 1 でもステップ 2 でも実行できず，全体を実行するステップ数は 3 となり，全体の実行ステップ数はスケジューリング解 (I) と変わらない．

図 2(d) に示すスケジューリング解 (III) は $\{v_1\}$，$\{v_2\}$，$\{v_3, v_4\}$，$\{v_5\}$ の演算クラスタに分割した場合である．演算クラスタ $\{v_3, v_4\}$ はステップ 2 に割り当てられる．スケジューリング解 (I) と比較して，実行ステップ数は 2 ステップに削減されたため，(III) はレイテンシ最小化の目的のときの最適解である．このとき，必要レジスタ数は 3 個であり，(I) と比べて増加しない．

(II) のようにできるだけ多くの演算をクラスタとしてチェイニングすればレイテンシ削減とレジスタ数削減の機会が増加するが演算器占有率の上昇により他に並行して割り当て必要なノードの割り当てを妨げてしまい，結果的にレイテンシが増大する可能性がある．演算クラスタ列挙・選択アルゴリズムは (II) のような解を退け，(III) のような解を発見する必要がある．

3.2 合成フロー

提案手法の合成フローを以下に示す．

Step 0. 入力 DFG をもとに [2] により RDR アーキテクチャの島への演算フロアプランを決定する．

Step 1. DFG 上の全演算ノード $v \in V$ に対して，v を終点とするクリティカルパス長を計算する．

Step 2. 演算クラスタ候補列挙・候補選択アルゴリズムに基づくリストスケジューリング・バインディングを実行する．

スケジューリング・バインディング完了後，レジスタおよび MUX を合成する．

3.3 Step 1: クリティカルパス長の計算

レイテンシを最小化するにはクリティカルパス上の演算を優先的に実行する必要がある．Step 1 では，[2] を参考に，クリティカルパス長 (CPL) を計算し，各演算ノードに対し CPL に基づく優先度を算出する．

演算ノード v が演算器 fu_i に割り当てられたときの v から o までの CPL は，fu_i，fu_j が配置される島をそれぞれ i_i，i_j とすると再帰的に次の式で計算される．

$$cpl(v, fu_i, o) = d_{sum}(fu_i) + \max_{v_k \in S(v)} \left\{ \min_{fu_j \in EFU(v_k)} \{D_c(i_i, i_j) + cpl(v_k, fu_j, o)\} \right\} \quad (5)$$

特に，PO までの CPL は次の式で計算される．

$$cpl(v, fu_i) = \max_{o \in PO} \{cpl(v, fu_i, o)\} \quad (6)$$

演算ノード v の優先度 $pr(v)$ は CPL をもとに次の式で計算される．

$$pr(v) = \min_{fu_i \in EFU(v)} \{cpl(v, fu_i)\} \quad (7)$$

Step 1 では，DFG G 内の全ノード $v \in V$ と，PI 内のノードを除く DFG G 内の全ノード n について，$cpl(v, fu_i, n)$ を計算する．この計算により，DFG 内の任意の演算ノードから任意の演算ノードへの CPL が全て計算される．これは演算クラスタの優先度で用いられる．

3.4 Step 2: スケジューリング・バインディング

演算器制約のもとレイテンシの最小化を目的とするため，リストスケジューリングを用いる．

RL はリストスケジューリングおける割り当て待ち演算ノードのリスト（優先度付きキュー）である．ある演算ノードが PI であるまたはその親ノードがスケジューリング済みの場合 RL に入る対象となる．スケジューリング・バインディングでは，コントロールステップ cs を 1 から増加させ，各コントロールステップにおいて RL を構築する．RL から優先度が高い順に演算 v を取り出して以下を実行する．

(1) v をスタートノードとする演算クラスタの列挙
(2) v をスタートノードとする演算クラスタ候補選択アルゴリズムによるスケジューリング・バインディング．

3.4.1 演算クラスタ列挙アルゴリズム

演算クラスタ列挙アルゴリズムは演算ノード s をスタートノードとするときの演算チェイニングを実現するためのスケジューリング候補を列挙する．

以下の条件を満たす DFG G の部分グラフ C を演算クラスタ候補として全て列挙する．

(1) C は v を含み，未スケジューリングノードだけで構成されるクラスタである．
(2) クラスタ内全演算 $c \in C$ に対して，$pr(c)$ は未スケジューリングの任意ノード u の優先度 $pr(u)$ よりも大きいか等しい．
(3) クラスタ内全演算 $c \in C$ に対して，c を実行する演算器は資源制約を満たす．

(2) はクラスタ内のノードがそれよりも優先度が高い未割り当てのノードに先立ってスケジューリングされることで図 2(c) のスケジューリング解 (II) のような解に陥ることを防ぐ．(3) で列挙するノードの最大値を定めることで計算爆発を抑える．これら演算クラスタの列挙はノード s から先行ノードおよび後続ノード方向に対して幅優先探索を用いることで列挙可能である．

例 3. 図 3 の DFG を入力としたとき，$cs = 1$ において v_1 をスタートノードとしたときの列挙された演算ノードクラスタを図 5 に示す．$C' = \{v_1, v_3\}$ や $C'' = \{v_1, v_2, v_3, v_4, v_5\}$ はそれぞれ (3)，(4) を満たさないため列挙されない． □

3.4.2 演算クラスタ選択アルゴリズム

演算クラスタ選択アルゴリズムは列挙されたクラスタを選択して実際にスケジューリング・バインディングする．各演算クラスタに優先度を割り当てる．

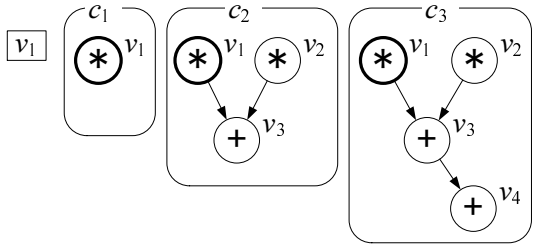


図 5 $cs = 1$ における v_1 をスタートノードとする演算クラスタ列挙．

演算クラスタ C の優先度は

(1) Step 1 で計算した CPL に基づく演算クラスタ C 内における PI から PO までのクリティカルパス遅延
(2) 演算クラスタ C を構成するノード数

と設定する．優先順位は (1)，(2) の順である．できる限り長いクリティカルパスに演算チェイニングを適用させる方がレイテンシおよびレジスタ数削減に効果的であり，同じクリティカルパス長であればできる限り演算ノード数が多い方がレイテンシおよびレジスタ数削減に効果的であるという考えに基づく．

選択アルゴリズムにより演算クラスタ候補がスケジューリング対象として選ばれる，データ転送制約および演算器資源制約を満たす限り演算クラスタ内の各ノードのスケジューリング・バインディングがトポロジカルオーダーに試みられる．各ノードは CPL が最小となる演算器に割り当てられる．クラスタのスケジューリング・バインディングに失敗した場合，次の演算クラスタ候補が試される．

例 4. 図 3 を入力としたとき，$cs = 1$ において $RL = \{v_1, v_2, v_5\}$ である．RL 内で優先度が最大の演算 v_1 が取り出される．v_1 をスタートノードとする演算クラスタ候補は例 3 で図 5 に示した．演算クラスタ候補選択アルゴリズムにより，c_3 がスケジューリング対象として選択される．クラスタ c_3 は演算器制約およびデータ転送制約を満たした上で 2 ステップにわたって割り当て可能である．演算 v_1, v_2, v_3, v_4 はそれぞれ演算器 fu_1, fu_2, fu_3, fu_4 へ割り当てる． □

4. 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．計算機実験環境は，OS が CentOS 5.10，CPU が Intel Xeon CPU E7-4870 2.4 GHz × 40，メモリ容量が 330 GB である．対象ベンチマークアプリケーションとして，ARF（ノード数 : 28），EWF（ノード数 : 34），EWF3（ノード数 : 102）を用いた．クロック周期は 3.4 ns とする [*2]．演算器は 16 bit 幅で，プロセスルールは STARC 90nm とする．島サイズは $90\mu\text{m} \times 90\mu\text{m}$ で，島の容量コストを $C = 2$，配線遅延係数 C_d は [6] に従い 250μm で 1ns 要すると仮定する．実験に用いた演算器の面積コストと遅延時間を表 2 に示す．演算器遅延は Synopsys Design Compiler を用いて算出された値である．メモリユニットはオンチップメモリとの通信ポートと仮定し面積は 0 とする．メモリユニットはいずれか 1 つの島に 1 個のみ配置する．

提案手法の有効性を示すため以下の 4 手法を比較する．

[*2] 3.4 ns は乗算を実行して同じステップに 1 つ隣の島にデータを転送できる最小のクロック周期である（0.1 ns 単位）．

表 1 計算機実験結果.

App.	#nodes	#islands	FUs	Algorithm	T_{clk} [ns]	Control steps	Latency [ns]	#Regs	#chainings	CPU Time [sec]
EWF	34	2 × 4	Add ×6, Mul ×5	MCAS [2]	3.4	17	57.8 (1.00)	13	–	0.35
				[8]		15	51.0 (0.88)	14	10	0.35
				[7] ($K = 2$)		13	44.2 (0.76)	13	14	0.40
				Ours		13	44.2 (0.76)	12	7	0.65
EWF3	102	2 × 4	Add ×6, Mul ×5	MCAS [2]	3.4	49	166.6 (1.00)	17	–	3.66
				[8]		45	153.0 (0.92)	18	32	3.59
				[7] ($K = 2$)		39	132.6 (0.80)	19	43	3.60
				Ours		36	122.4 (0.73)	17	24	5.46
ARF	28	2 × 3	Add ×4, Mul ×4	MCAS [2]	3.4	11	37.4 (1.00)	10	–	0.28
				[8]		10	34.0 (0.91)	10	4	0.39
				[7] ($K = 1$)		10	34.0 (0.91)	10	4	0.28
				Ours		9	30.6 (0.82)	8	8	0.75

表 2 演算器の面積コストと遅延 [6].

演算器	面積コスト	遅延 [ns]
加算器	1	1.44
乗算器	2	2.82
メモリユニット	–	2.82
レジスタ	–	0.23

従来手法 1　RDR アーキテクチャを対象とした演算チェイニングを用いない従来の高位合成手法（MCAS）[2]
従来手法 2　RDR アーキテクチャを対象として 2 演算間の演算チェイニングを実現する高位合成手法 [8]
従来手法 3　パスに基づく多段の演算チェイニングを用いた高位合成手法 [7]
提案手法　本稿で提案した高位合成手法.

島数を 2 × 2 から 3 × 3 までとし，加算器と乗算器の数を変化させて合成した．各ベンチマークアプリケーションに対する実験結果を表 1 に示す．[7] はチェイニングパス探索ステップ制限 K を与えるが，その値をアルゴリズムの欄に示す．‘#Regs’ と ‘#chainings’ で示す欄はそれぞれレジスタ数と構築された演算チェイニング数を表す．CPU 時間は高位合成アルゴリズムの内，論理合成を含まない部分（フロアプラン・スケジューリング・バインディング）の実行時間である．

実験した全ベンチマークアプリケーションに対して提案手法は既存手法と比較してレイテンシを削減した．EWF のとき最大で [2] と比較してレイテンシを 23.5%削減した．EWF と ARF では提案手法は既存手法と比較してレジスタ数を削減した．既存の演算チェイニングを用いた高位合成手法 [7,8] と比較して多くの候補を探索するため実行時間を要するが，ARF のとき最大で 2.7 倍である．

5. おわりに

本稿では演算クラスタの列挙・選択に基づいて演算チェイニングを実現する RDR 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を提案した．提案手法はフロアプラン決定後，リストスケジューリング・バインディング実行時に CPL を利用した列挙・選択アルゴリズムを用いたレイテンシを増加させないような演算クラスタリングに基づいて演算チェイニングを実現する．計算機実験により，提案手法は演算チェイニングを用いない従来手法，演算チェイニングを用いた RDR アーキテクチャを対象とした既存手法と比較してレイテンシおよびレジスタ数を削減する手法であることを確認した．演算チェイニングを用いない従来手法と比較してレイテンシを最大 23.5 削減した．

本稿で提案した合成フローは論理合成や配置配線等の下位設計を実行していない．演算遅延の合計に基づいて島間の演算チェイニングを設計した場合，配線遅延の分のタイミングマージンが減少するため各島に対する論理合成でのタイミング制約が厳しくなることが予想される．下位設計を考慮した高位合成手法への改良は今後の課題である．

謝辞　本研究の一部は NEDO による．

参考文献

[1] L. Chen, M. Ebrahimi, and M. B. Tahoori, “Reliability-aware operation chaining in high level synthesis,” in *Proc. of 2015 20th IEEE European Test Symposium*, pp. 1–6, 2015.

[2] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, “Architecture and synthesis for on-chip multicycle communication,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, 2004.

[3] J. Cong, Y. Fan, and J. Xu, “Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture,” *ACM Trans. on Design Automation Electronic Systems*, vol. 14, no. 3, pp. 35:1–35:31, 2009.

[4] S. Sinha and T. Srikanthan, “Dataflow graph partitioning for area-efficient high-level synthesis with systems perspective,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 20, no. 1, pp. 5:1–5:18, 2014.

[5] M. Tan, S. Dai, U. Gupta, and Z. Zhang, “Mapping-aware constrained scheduling for LUT-based FPGAs,” in *Proc. of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 190–199, 2015.

[6] S. Tanaka, M. Yanagisawa, T. Ohtsuki, and N. Togawa, “A fault-secure high-level synthesis algorithm for RDR architectures,” *IPSJ Trans. on System LSI Design Methodology*, vol. 4, pp. 150–165, 2011.

[7] K. Terada, M. Yanagisawa, and N. Togawa, “A floorplan-driven high-level synthesis algorithm with multiple-operation chainings based on path enumeration,” in *Proc. of 2015 IEEE International Symposium on Circuits and Systems*, pp. 2129–2132, 2015.

[8] K. Terada, M. Yanagisawa, and N. Togawa, “A high-level synthesis algorithm with inter-island distance based operation chainings for RDR architectures,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 7, pp. 1366–1375, 2015.