多段演算チェイニングを利用した配線遅延を考慮した高位合成手法

寺田 晃太朗 †　柳澤 政生 †　戸川 望 †

† 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

半導体の微細化技術に伴い，配線遅延の相対的増大が問題となっている．本稿では，高位レベルで配線遅延を見積もり可能な RDR アーキテクチャを対象に，多段接続された演算チェイニングを構築してレイテンシを削減する高位合成手法を提案する．提案手法は多段接続された演算チェイニングパスの候補を列挙した後，配線遅延を考慮しながらスケジューリング，バインディングを実行する．アルゴリズムを用いて実行可能な候補を列挙し配線遅延を考慮して RDR アーキテクチャに最適なものを選択する．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，2 段接続に制限された演算チェイニングを利用した手法と比較して，レイテンシを削減し，提案手法の有効性を示した．

An Interconnection-Delay-Aware High-Level Synthesis Algorithm with Multiple-Operation Chainings

Kotaro TERADA†　Masao YANAGISAWA†　Nozomu TOGAWA†

† Dept. of Computer Science and Communications Engineering, Waseda University

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 using multiple-operation chainings which consist of two or more operations, and reduce the overall latency targeting RDR architectures. Our algorithm enumerates multiple-operation-chaining path candidates before performing scheduling/binding considering interconnection delays. Based on them, we find out optimal ones for RDR architectures. Experimental results show that our algorithm reduces the latency compared to the approach without operation chainings and the one with chainings of up to two operations.

1　はじめに

近年，半導体の微細化技術により，高集積回路の製造が可能となった．それに伴い抽象度の高い動作記述からレジスタ転送レベル（RTL）回路を自動生成する高位合成が重要な技術となっている．高位合成によって，設計時間や設計コストの削減などを実現できる．高位合成は演算ノードをコントロールステップ（CS）に割り当てるスケジューリング，演算器をライブラリから選択し数を決定するアロケーション，演算ノードや変数を演算器やレジスタに割り当てるバインディングなどから構成される．

プロセスの微細化が進むにつれ，配線遅延がゲート遅延に比べ相対的に大きくなっている．結果として，配線遅延時間がクロック周期を上回り，チップ全体にわたり 1 クロックサイクルで通信するチップは非効率である．高位合成の段階で配線遅延を考慮することが回路のレイテンシを削減するために重要である．

これに対し，RDR アーキテクチャ（Regular-Distributed-Register Architecture）が提案された [2]. 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アーキテクチャである．チップ上を複数の均等な島に分割して各島にレジスタ，演算ユニット（LCC），コントローラ（FSM）を配置する．各島が規則的に配置されているため配線遅延時間を容易に推測できるという利点を持つ．RDR アーキテクチャ向けの高位合成手法として MCAS [2] が提案されている．

また，高位合成では演算の種類によって演算器の遅延時間が異なる点を考慮する必要がある．一般に高位合成では，1 クロック周期すなわち 1 ステップで 1 つの演算を実行する．クロック周期は常に一定であるため，演算によってはクロック周期内のごく短時間で演算が完了し無駄な時間が生じる．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．演算チェイニングを用いてデータ依存のある複数の演算を少ないクロックサイクル内に詰めて実行することにより全体的なステップ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 [3–14, 17–21]．これらの研究は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．我々は [16] で RDR アーキテクチャを対象とした高位合成手法の中に演算チェイニングを用いた合成手法を提案した．[16] ではチェイニングを構成する演算ノードは最大で 2 個だったが，本稿ではこれを n 個 ($n \geq 2$) に拡張する．チェイニングパス探索ステップ制限 K を与え，K ステップまでにチェイニング可能なパスを列挙することでよりスラックタイムが小さくなるようなチェイニングを構成することで，結果的に n 個のノードから構成されるチェイニングを実現する．

本稿では，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手法を提案する．提案手法はチェイニングパス候補を列挙した後，配線遅延を考慮したマルチサイクル通信に適したスケジューリング，バインディングを実行する．チェイニング候補を仮定として与えるのではなく，アルゴリズムを用いて実行可能な候補を列挙し RDR アーキテクチャ

に最適なものを選択する．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の RDR アーキテクチャを対象とした高位合成手法 [2] と比較してレイテンシを最大 35.3%，2 演算のチェイニングに限定した高位合成手法 [16] と比較してレイテンシを最大 26.7% 削減する手法であることを確認した．

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

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．ノード v_1 から v_2 へ向かうエッジを e_{v_1,v_2} と表す．

演算器 fu の遅延時間を d_{fu}，レジスタの読み出し・書き込み時間を d_{reg} とする．演算遅延時間とレジスタ遅延時間は MUX 遅延時間を含む．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時間を $dsum(fu)$ と表す．合計遅延時間 $dsum(fu)$ は，

$$dsum(fu) = d_{reg} + d_{fu} \tag{1}$$

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 上の演算ノード $v \in V$ の隣接する先行ノードの集合を $P(v)$，隣接する後続ノードの集合を $S(v)$ で表す．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 PIs とする．また，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 POs とする．

DFG 上の道（パス）を定義する．ノード v_1 からノード v_n まで $v_2, v_3, \cdots, v_{n-1}$ を経由するパスを $P = v_1 v_2 \cdots v_n$ と表す．ただし，$1 \leq i < n$ について v_i と v_{i+1} はエッジを持つ．パス遅延 $d_{path}(P)$ はパス P に含まれる各演算について，これを実行する演算器の最小遅延の和にレジスタ遅延を加えたものである．$P = v_1 v_2 \cdots v_n$ のとき，パス遅延 $d_{path}(P)$ は，

$$d_{path}(P) = d_{reg} + \sum_{v \in P} \left(\min_{fu \in EFU(v)} d_{fu} \right) \tag{2}$$

と計算できる．

2.2 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．

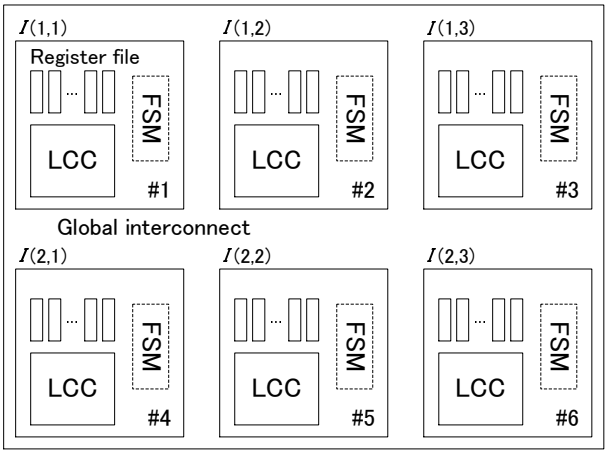


図 1: 2 × 3 RDR アーキテクチャ．

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．チップを規則的に分割することで配線遅延の予測が容易である．

例として 2 × 3 の島から構成される RDR アーキテクチャを図 1 に示す．各島はローカル機能ユニットクラスタ，レジスタ・ファイル，有限状態機械の 3 つの要素から構成される．ローカル機能ユニットクラスタは機能ユニットが集合する回路であり演算を行う．演算器の入出力はレジスタに接続される．レジスタ・ファイルはその島専用のストレージである．島で使用するデータや演算結果は各島のレジスタへ格納される．有限状態機械は状態遷移によって LCC とレジスタの振る舞いを制御するコントローラである．

島の大きさは島内部での演算とレジスタへの演算結果の格納が 1 クロックサイクルで完了するように決定される．各島はグローバル配線で結線される．

RDR アーキテクチャでは，演算器の近くにレジスタが配置されるため，通信時間を減らすことができる．島が規則的に配置されるため，配線遅延時間を見積もるのが容易である．

各島は正方形とし，チップを $N \times M$ の島に分割する．n 行 m 列の島を $I(n, m)$ と表す．ただし，$1 \leq n \leq N$，$1 \leq m \leq M$ である．島 $i_1 = I(n_1, m_1)$ から島 $i_2 = I(n_2, m_2)$ までのデータ転送時間 $D_c(i_1, i_2)$ は，[15] を参考に，以下の式で定義される．

$$D_c(i_1, i_2) = C_d \cdot (|n_1 - n_2| + |m_1 - m_2|)^2 \tag{3}$$

ここで C_d は配線遅延係数である．

演算器 fu_1 が島 $i_1 = I(n_1, m_1)$ に配置されているとする．演算器 fu_1 の出力データを島 $i_x = I(n_x, m_x)$ 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．

$T_{clk} \geq D_c(i_1, i_x) + dsum(fu_1)$ 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．

$T_{clk} < D_c(i_1, i_x) + dsum(fu_1)$ 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．1 つの

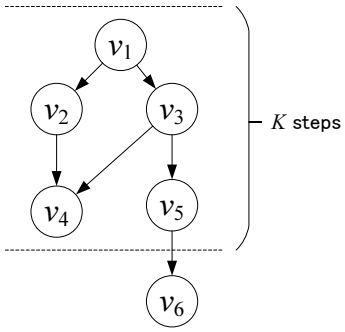


図 2: チェイニングパス.

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 を超えないものとする．

2.3 RDR アーキテクチャ上の多段演算チェイニング

本節では，配線遅延時間を考慮した RDR アーキテクチャ上の多段演算チェイニングを定義する．チェイニングされる演算の最大数は制限しない．ノード $v \in V$ から全ての PO までの全パスの内，チェイニングパス探索ステップ制限として定数 K（正の整数）が与えられたとき，$d_{path}(P) \leq K \cdot T_{clk}$ を満たすパス P の集合を v のチェイニングパスと呼び $CPG(v)$ と表す．今，$v_1 \in V$ に対するチェイニングパスとして $CPG(v_1) = \{P\}, P = v_1 v_2 v_3 \cdots v_n$ とする．実際に演算器フロアプラン・バインディングで演算 v_i が島 i_i 内の演算器 fu_i に割り当てられたとする．このとき，v は最大で

$$d_{reg} + d_{fu_1} + \sum_{v_i \in P'} (d_{fu_i} + D_c(i_i, i_{i+1})) \leq K \cdot T_{clk} \quad (4)$$

を満たすパス P の部分パス P' 内の全てのノードと多段チェイニングを構成することが可能である．

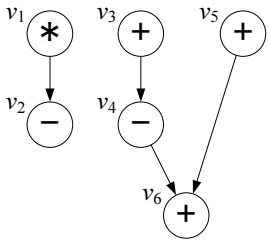
例 1. 図 2 に例を示す．$T_{clk} = 3.0$, 全ての演算遅延は 1.8ns とする．v_1 から全ての PO までのパスは $P_1 = v_1 v_2 v_4$, $P_2 = v_1 v_3 v_4$, $P_3 = v_1 v_3 v_5 v_6$ の 3 つ存在する．$K = 2$ のとき，ノード 3 つまでのパス遅延は $K \cdot T_{clk} = 6.0$ns を上回らない．よって，$CPG(v) = \{v_1 v_2 v_4, v_1 v_3 v_4, v_1 v_3 v_5\}$ となる．

本稿では簡単のため全ての演算器の遅延はクロック周期を上回らないとするが，演算器遅延がクロック周期を上回る場合，チェイニング開始演算が終了する CS から K ステップまでのステップ制限でチェイニングパスを構成することで RDR アーキテクチャ上の多段チェイニングを定義することができる．

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多段演算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．

定義 1. 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とは，DFG $G = (V, E)$，クロック周期 T_{clk}，チェイニングパス探索ステップ制限 K，RDR 島数，演算器情報（遅延時間・面積）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，任意の演算 v から最大 K ステップ後までに実行される演算を多段にチェイニングを構成することを制約として，レイテンシの最小化を目的とする．



(a) DFG.

T_{clk}	3.0 ns
探索ステップ制限K	2
RDR島数	2×2
演算器数	加算器×2 (fu_1, fu_3) 減算器×2 (fu_2, fu_4) 乗算器×2 (fu_5, fu_6)
加/減算器遅延時間	1.5 ns
乗算器遅延時間	2.8 ns
レジスタ遅延時間	0.1 ns
隣接島間通信	0.2 ns

(b) クロック周期，RDR 仕様，演算器情報.

図 3: RDR アーキテクチャを対象とした多段演算チェインニングを用いた高位合成問題に対する入力例.

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 であり各ノードの左に示す記号 v_i はその演算ノードを表す文字である．図 3(b) にその他の入力を項目別に表としてまとめる．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．水平線での区切りが 1 CS を表し，各ノードの右に示す fu_i はその演算ノードを実行する演算器である．演算 v_1，v_2 と演算 v_3，v_4，v_6 はチェイニングされて共に 2 CS 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_3 の入力データを保持するレジスタは 2 CS 間データを保持する．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通信時間も考慮している．

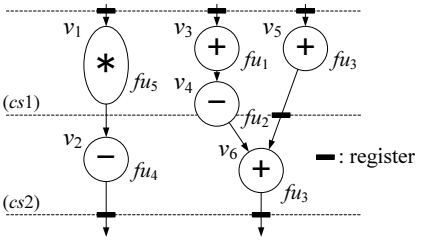
3 提案手法

本章では，RDR アーキテクチャを対象とした多段演算チェイニングを用いた高位合成を提案する．提案手法の合成フローを示し，各ステップを提案する．

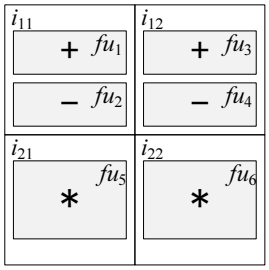
3.1 合成フロー

まず，入力 DFG をもとに RDR アーキテクチャの島への演算器配置を決定する．演算器配置の決定には MCAS [2] を用いる（Step 0）．

多段演算チェイニングを用いたスケジューリング・バインディングは 3 つのステップで構成される．Step 1 では，チェイニングパスを列挙する．このステップ



(a) スケジューリング・バインディング済 DFG.



(b) フロアプラン.

図 4: RDR アーキテクチャを対象とした多段演算チェインニングを用いた高位合成問題の出力例.

では島間のデータ転送時間を考慮しない．Step 2 では，Step 3 でリストスケジューリングを行うために各演算ノードと各チェイニングパスの優先度を設定する．Step 3 では，Step 2 で算出した優先度をもとにチェイニングパスの中から実際にチェイニングを構成する演算を決定しリストスケジューリングとバインディングを同時に行う．このとき，島間データ転送時間を考慮する．DFG 上でスケジューリング・バインディングが完了すると，レジスタ・MUX・FSM を合成する．

Step 1: チェイニングパスの列挙

Step 1 では，DFG 上の各ノードに対しチェイニングパスを求める．DFG 上の全ノード $v \in V$ に対し，v から深さ優先探索でパスを構築しながらノードを探索する．パスに対しパス遅延 d_{path} を計算しその値が $K \cdot T_{clk}$ を超えたところでそれ以上の深さへの探索を打ち切る．$K \cdot T_{clk}$ を超えない最長のパスを $CPG(v)$ に追加し，v から探索可能な全てのパスについて調査し，チェイニングパス $CPG(v)$ を構成する．ここでは配線遅延を考慮せず，最も遅延の小さい演算器を使用しているため，可能な限り最長のパスを列挙することができる．

Step 2: 演算ノードとチェイニングパスの優先度の算出

Step 2 では，Step 3 でリストスケジューリングを実行するための優先度を算出する．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．[2] を参考に，優先度を CPL (Critical Path Length) に基づき計算する．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 PO 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．

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

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

$$pr(v) = \min_{\forall fu_i} \{cpl(v, fu_i)\} \quad (6)$$

Step 3 でスケジューリングを実行するときに，チェイニングパスの優先度が必要である．パス P の優先度を $pr(P)$ で表す．ノード v に対するチェイニングパス $CPG(v)$ の内の各パスは次の方法でソートし，し，ソートされた順にパスの優先度を設定することができる．$CPG(v)$ の要素数が m であるとし，各パスを

$$\begin{cases} P_1 &= v_1^1 v_2^1 \cdots \\ P_2 &= v_1^2 v_2^2 \cdots \\ &\vdots \\ P_m &= v_1^m v_2^m \cdots \end{cases}$$

と表す．パス P_i $(1 \leq i \leq m)$ の j 番目のノードを v_j^i と表す．まず，$j = 1$ とし，$pr(v_1^i)$ をキーとしてパスをソートする．$pr(v_1^i)$ が唯一の値を持つならば P_i の順序が確定する．$pr(v_1^i)$ が互いに等しい値を持つならば，j を 1 増加させ $pr(v_2^i)$ をキーとして同様の手順を繰り返す．ただし，v_j^i が存在しないときその優先度 $pr(v_j^i)$ は 0 とする．全てのパスの順序関係が確定したとき $CPG(v)$ の内の全パスのソートが完了する．

Step 3: スケジューリング・バインディング

Step 3 は，スケジューリングとバインディングである．ここでは，リストスケジューリングを用いる．

リストスケジューリングでは，上式の優先度をもとに未スケジューリング・未バインディングの演算ノードで優先度キュー（レディーリスト）RL を構成する．各 CS 毎にリソースを管理するリソースプール $RP(cs) = \langle fu_1, fu_2, \cdots, fu_n \rangle$ を用意する．fu_i はその CS で未使用であれば 1，使用中であれば 0 をとる．

ある CS cs について，RL から演算器割り当て可能かつ優先度が最大の演算ノード v を取ってくる．

演算ノード v がチェイニングパスを持たないとき：

v は cs にスケジューリングされ，割り当て可能な演算器の中で $cpl(v, fu_i)$ が最小となる演算器 fu_i へバインディングされ，$RP(cs)$ の fu_i を 0 にする．演算器に空きが無い場合，v は RL に戻され後の CS にスケジューリングされる．

演算ノード v がチェイニングパスを持つとき：

v は同様にスケジューリング・バインディングされる．$P \in CPG(v)$ のうち，パスの優先度が大きい順にパス p とチェイニングを試みる．パス P を先頭から走査し，最初に現れる未割り当てノードを u とする．u は割り当て可能な演算器の中で $cpl(u, fu_j)$ が最小最小となる演算器 fu_j へのバインディングが試みられる．ここで u は u の親ノードの演算終了時刻の内最も遅いノードからの島間配線遅延を考慮した CS に割り当てる必要が

ある．v の割り当て CS からこの CS まで K ステップを超える場合，u はスケジューリング・バインディングされない．そうでない場合，スケジューリング・バインディングが成功する．u のスケジューリングされた CS が複数の CS をまたがる場合，その演算チェイニングがまたがる全ての CS で演算器を占有しなければならないため，v のスケジューリングされた CS cs_v から u のスケジューリングされた CS cs_u までの全 cs_i で $RP(cs_i)$ 中の fu_j を 0 にする．

割り当て可能な演算器が無くなると cs を 1 増加させて同様の手順を繰り返す．DFG 上の全てのノードに CS と演算器が割り当てられるとスケジューリング・バインディングが完了する．

例 3. 図 3 を入力とし [2] より図 4(b) に示される演算器フロアプランが得られたとする．Step 1 で $CPG(v_1) = \{v_1v_2\}$, $CPG(v_2) = \{v_2\}$, $CPG(v_3) = \{v_3v_4v_6\}$, $CPG(v_4) = \{v_4v_6\}$, $CPG(v_5) = \{v_5v_6\}$, $CPG(v_6) = \{v_6\}$ を得る．

Step 2 で各ノードの優先度を求めると，$pr(v_1) = 4.7$, $pr(v_2) = 1.6$, $pr(v_3) = 4.8$, $pr(v_4) = 3.2$, $pr(v_5) = 3.2$, $pr(v_6) = 1.6$ である．

Step 3 では，$cs = 1$ へのスケジューリング・バインディングを考える．$RL = \{v_3, v_1, v_5\}$ となる．まず RL から v_3 が取り出され，$cs = 1$，演算器 fu_1 へスケジューリング・バインディングされる．$RP(1) = \langle 0, 1, 1, 1, 1, 1\rangle$ となる．$CPG(v_3) = \{v_3v_4v_6\}$ であるから，v_4 のスケジューリング・バインディングが試みられ $cs = 1$, fu_2 へスケジューリング・バインディングされる．v_4 は CS をまたいでスケジューリングされ，$RP(1) = \langle 0, 0, 1, 1, 1, 1\rangle, RP(2) = \langle 0, 0, 1, 1, 1, 1\rangle$ となる．さらに v_6 のスケジューリング・バインディングが試みられるが，入力データが揃っていないため一旦打ち切られる．次に RL から v_1 が取り出され，$cs = 1$，演算器 fu_5 へスケジューリング・バインディングされる．$RP(1) = \langle 0, 0, 1, 1, 0, 1\rangle, RP(2) = \langle 0, 0, 1, 1, 1, 1\rangle$ となる．$CPG(v_1) = \{v_1v_2\}$ であるから，v_2 のスケジューリング・バインディングが試みられ $cs = 2$，fu_4 へスケジューリング・バインディングされる．$RP(1) = \langle 0, 0, 1, 0, 0, 1\rangle, RP(2) = \langle 0, 0, 1, 0, 1, 1\rangle$ となる．次に RL から v_5 が取り出され，$cs = 1$，fu_3 へスケジューリング・バインディングされる．$RP(1) = \langle 0, 0, 0, 0, 0, 1\rangle, RP(2) = \langle 0, 0, 1, 0, 1, 1\rangle$ となる．$CPG(v_5) = \{v_5v_6\}$ であるから，v_6 のスケジューリング・バインディングが試みられるが v_4 の終了を待つ必要があり $cs = 2$，fu_3 へスケジューリング・バインディングされる．このとき，一旦レジスタにデータを格納することで同じ演算器を使用できる．割り当て可能な演算が無くなり，すべてのノードのスケジューリング・バインディングが完了し，図 4(a) に示すスケジューリング・バインディング結果が得られる．

4 計算機実験結果

提案手法を計算機上に C++ 言語を用いて実装した．計算機実験環境は，OS が CentOS 5.5，CPU が AMD Quad-Core Opteron 2360 SE 2.5 GHz × 2，メモリ容量が 16 GB である．対象アプリケーションは，

表 1: 演算器の面積コストと遅延（MUX 遅延を含む）[1].

Functional unit	Capacity cost	Delay [ns]
加算器	1	1.44
乗算器	2	2.82
メモリユニット	0	2.82
レジスタ	0	0.11

EWF（ノード数 : 34），DCT（ノード数 : 48），EWF3（ノード数 : 102），FIR（ノード数 : 75）の 4 つでいずれも分岐ノードを含まない．クロック周期は 3.0ns とする．演算器は 16bit 幅で，プロセスのデザインルールは STARC 90nm とする．島サイズは $90\mu m \times 90\mu m$ で，島の容量コストを $C = 2$，配線遅延は配線長の 2 乗に比例し $250\mu m \times 250\mu m$ で 1ns 要するとする [15]．実験に用いる演算器の面積コストと遅延時間を表 1 に示す．ただし，メモリユニットはチップ内のいずれか 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，

従来手法 1 RDR アーキテクチャを対象とした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 演算に限定したチェイニングを用いた高位合成手法 [16]
提案手法 本稿で提案した高位合成手法

の 3 つを比較する．提案手法はチェイニングパス探索ステップ制限 $K = 1$ ならびに 2 とする．

島数を 1×2 から 2×3 までとし，加算器と乗算器の数を変化させて合成した．アプリケーションに対する実験結果を表 2 に示す．CPU 時間は合成アルゴリズムの内，Step 1，Step 2，Step 3 の部分のみの実行時間である．

EWF と EWF3 については，提案手法 $K = 2$ のとき最もレイテンシが削減できていることが確認できる．一方，DCT と FIR については，提案手法 $K = 1$ のとき [16] とレイテンシが等しかった．

提案手法は DFG 上の深さを優先して，あるノードからできるだけ先のノードまで詰め込めるだけ詰め込むという操作をしており，ノードが比較的縦に多くつながっている EWF や EWF3 に対しては効果が見られた．

5 おわりに

本稿では，配線遅延の相対的拡大問題に対応した 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手法を提案した．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の RDR アーキテクチャを対象とした高位合成手法 [2] と比較してレイテンシを最大 35.3%，2 演算のチェイニングに限定した高位合成手法 [16] と比較してレイテンシを最大 26.7% 削減する手法であることを確認した．したがって，本稿で提案した手法は，配線遅延の相対的拡大問題に対し，多段演算チェイニングを用いることでレイテンシが削減できたため，有効な手法であると言える．

一方，DFG 上の深さを優先してスケジューリング・バインディングを実行したため，一部のアプリケーションに対しては効果が見られなかった．今後は，チェイ

表 2: 計算機実験結果.

App. #islands	FUs (ADD,MUL,MEM)	Alg.	T_{clk} [ns]	Control steps	Latency [ns]	CPU Time [ms]
EWF 2×2	$(4, 2, 0)$	w/o chainings (MCAS) [2]	3.0	17	51.0	1711
		w/ 2-chain [16]		15	45.0	826
		Ours ($K = 1$)		13	39.0	859
		Ours ($K = 2$)		**11**	**33.0**	920
EWF3 2×3	$(6, 3, 0)$	w/o chainings (MCAS) [2]	3.0	49	147.0	2342
		w/ 2-chain [16]		45	135.0	1135
		Ours ($K = 1$)		42	126.0	1231
		Ours ($K = 2$)		**36**	**108.0**	1439
DCT 2×3	$(6, 3, 0)$	w/o chainings (MCAS) [2]	3.0	11	33.0	2112
		w/ 2-chain [16]		**9**	**27.0**	941
		Ours ($K = 1$)		**9**	**27.0**	969
		Ours ($K = 2$)		10	30.0	971
FIR 2×3	$(4, 4, 1)$	**w/o chainings (MCAS)** [2]	3.0	**16**	**48.0**	2227
		w/ 2-chain [16]		**16**	**48.0**	1079
		Ours ($K = 1$)		**16**	**48.0**	1161
		Ours ($K = 2$)		27	81.0	1231

ニング演算の構成を工夫することでこの問題を解決する．加えて，レジスタ数，MUX 数，コントローラ面積を評価し比較する．

謝辞

本研究は独立行政法人新エネルギー・産業技術総合開発機構（NEDO）の先導的産業技術創出事業の支援を受けて行われた．

参考文献

[1] S. Abe, M. Yanagisawa, and N. Togawa, “An energy-efficient high-level synthesis algorithm for huddle-based distributed-register architectures,” in *Proc. 2012 IEEE International Symposium on Circuits and Systems*, pp. 576–579, 2012.

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

[3] B. Landwehr and P. Marwedel, “Oscar: Optimum simultaneous scheduling, allocation and resource binding based on integer programming,” in *Proc. Conference on European Design Automation*, pp. 90–95, 1994.

[4] T. Ly, D. Knapp, R. Miller, and D. MacMillen, “Scheduling using behavioral templates,” in *Proc. 32nd ACM/IEEE Conference on Design Automation*, pp. 101–106, 1995.

[5] P. Marwedel, B. Landwehr, and R. Dömer, “Build-in chaining: Introducing complex components into architectural synthesis,” in *Proc. Asia South Pacific Design Automation Conference*, pp. 599–605, 1997.

[6] K. Mittal, A. Joshi, and M. Mutyam, “Timing variation-aware scheduling and resource binding in high-level synthesis,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 16, no. 4, article 40, 2011.

[7] M. C. Molina, J. M. Mendías, and R. Hermida, “Bit-level scheduling of heterogeneous behavioural specifications,” in *Proc. 2002 IEEE/ACM International Conference on Computer-Aided Design*, pp. 602–608, 2002.

[8] M. C. Molina, R. Ruiz-Sautua, P. García-Repetto, and J. M. Mendías, “Performance-driven scheduling of behavioural specifications,” *Integration, the VLSI Journal*, vol. 42, issue 3, pp. 294–303, 2009.

[9] S. Park and K. Choi, “Performance-driven scheduling with bit-level chaining,” in *Proc. 36th ACM/IEEE Conference on Design Automation*, pp. 286–291, 1999.

[10] S. Park and K. Choi, “Performance-driven high-level synthesis with bit-level chaining and clock selection,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 199–212, 2001.

[11] M. Rim, R. Jain, and R. D. Leone, “Optimal allocation and binding in high-level synthesis,” in *Proc. 29th ACM/IEEE Conference on Design Automation*, pp. 120–123, 1992.

[12] R. Ruiz-Sautua, M. C. Molina, J. M. Mendías, and R. Hermida, “Behavioural transformation to improve circuit performance in high-level synthesis,” in *Proc. Conference on Design, Automation and Test in Europe*, vol. 2, pp. 1252–1257, 2005.

[13] 貞方毅, 松永裕介, “動作合成におけるチェイニングに関する考察,” 情報処理学会研究報告, 2005-SLDM-122, pp. 67–72, 2005.

[14] T. Sadakata and Y. Matsunaga, “A simultaneous module selection, scheduling, and allocation method considering operation chaining with multi-functional units,” *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E90-A, no. 4, pp. 792–799, 2007.

[15] 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.

[16] 寺田晃太朗, 柳澤政生, 戸川望, “演算チェイニング候補列挙に基づく配線遅延を考慮した高位合成手法,” 第 27 回回路とシステムワークショップ, 2014.

[17] H. Tomiyama, A. Inoue, and H. Yasuura, “Statistical performance-driven module binding in high-level synthesis,” in *Proc. 11th International Symposium on System Synthesis*, pp. 66–71, 1998.

[18] Y. H. Wu, C. J. Yu, and S. D. Wang, “Heuristic algorithm for the resource-constrained scheduling problem during high-level synthesis,” *IET Computers and Digital Techniques*, vol. 3, no. 1, pp. 43–51, 2009.

[19] S. Xydis, I. Triantafyllou, G. Economakos, and K. Pekmestzi, “Flexible datapath synthesis through arithmetically optimized operation chaining,” in *Proc. 2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pp. 407–414, 2009.

[20] Y. Yi, I. Nousias, M. Milward, S. Khawam, T. Arslan, I. Lindsay, “System-level scheduling on instruction cell based reconfigurable systems,” in *Proc. 2006 Design, Automation and Test in Europe*, pp. 1–6, 2006.

[21] D. C. Zaretsky, G. Mittal, R. P. Dick, and P. Banerjee, “Balanced scheduling and operation chaining in high-level synthesis for FPGA designs,” in *Proc. 8th International Symposium on Quality Electronic Design*, pp. 595–601, 2007.