SIMD型プロセッサコアの自動合成のためのパイプライン演算ユニット生成手法

栗原 輝[†] 宮岡 祐一郎[†,☆] 戸川 望[†]
柳澤 政生[†] 大附 辰夫[†]

画像処理アプリケーションなどで頻繁に処理されるSIMD乗算や乗加算は演算実行遅延が大きく，SIMD演算の実行遅延がプロセッサの動作周波数を決定することが多い．SIMD演算を実行可能なSIMD型プロセッサコアの自動合成では，SIMD演算のマルチサイクル化によるプロセッサコアの動作周波数の向上が期待できるが，演算のマルチサイクル化にはプロセッサに付加する演算ユニットを柔軟にパイプライン化する工夫が必要である．本稿では，SIMD型プロセッサコア自動合成のためのパイプライン演算ユニット生成手法を提案する．提案手法は，演算ユニットに割り当てられる命令の集合，パイプライン段数，および面積と遅延の制約値を入力とし，制約値を満たし割り当てられた命令を実行可能なパイプライン演算ユニットを自動で生成する．演算ユニットを複数の部分機能を実現するハードウェアユニットの組合せから構成することにより，面積と遅延値の小さい演算ユニットを高速に生成できる．さらに，それぞれのハードウェアユニットを非常に小さな単位のハードウェアモジュールから構成し，モジュール間にパイプラインレジスタを挿入することによって，与えられた段数に応じた柔軟なパイプライン化が可能となる．計算機実験により本手法の有効性を評価した．

A Pipelined Functional Unit Generation Method for SIMD Processor Synthesis System

Akira Kurihara,[†] Yuichiro Miyaoka,[†,☆] Nozomu Togawa,[†]
Masao Yanagisawa[†] and Tatsuo Ohtsuki[†]

A SIMD processor core has SIMD functional units whose critical path delay is relatively long and it usually determines operating frequency. Pipelining of functional units is quite necessary to increase the operating frequency, and thus we propose a pipelined functional unit generation method which can synthesize functional units with varying numbers of pipeline stages. Given a set of instructions to be executed by a functional unit, the number of pipeline stages, and constraints for area and delay, the proposed algorithm generates more than one architecture candidates for the functional unit. By composing the functional units from the combination of subfunctional units, we can generate the functional units with small area and short delay quickly. Furthermore, because each subfunctional unit is composed of very small hardware units, we can insert pipeline registers between them. Then we can decide a pipeline stage number freely and can obtain the pipelined functional units. We also show the promising experimental results on the algorithm evaluation.

1. はじめに

Packed SIMD型演算（またはSIMD演算）とは，1つの b ビット演算ユニットを用いて，n 個の b/n ビットデータを1命令で実行する演算のことである．SIMD演算を用いることで，画像処理アプリケーションを画素並列に実行することができる．SIMD演算を実行する命令をSIMD命令，SIMD演算を実行する演算ユニットをSIMD演算ユニット，また，SIMD演算ユニットを持ち，SIMD命令を命令セットとして持つプロセッサコアをSIMD型プロセッサコアと呼ぶ．SIMD演算を用いた文献1)，2)，9)などでは，画像処理アプリケーションを高速に実行することが可能である．

SIMD命令は同種の演算であっても，n の値や飽和処理の有無などによって多数の命令が存在する．しかし特定のアプリケーションを実行する場合，ごく一部のSIMD命令のみが必要となるので，SIMD命令を持つプロセッサの設計にはアプリケーションに応じて命令セットを変更するハードウェア/ソフトウェア（ま

† 早稲田大学理工学部コンピュータ・ネットワーク工学科
Department of Computer Science, Waseda University
☆ 現在，株式会社東芝
Presently with Toshiba corporation

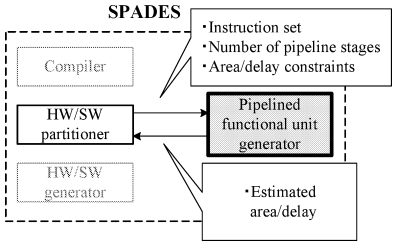

図 1 HW/SW 分割系と演算ユニット生成系
Fig. 1 A HW/SW cosynthesis system and pipelined functional unit generation.

たは HW/SW）協調合成が有効であると考えられる．

我々は，SIMD 型プロセッサコアを対象とした HW/SW 協調合成システム SPADES☆を提案している[3),10),11)]．SPADES は，アプリケーションプログラムと実行時間制約を入力とし，実行時間制約を満たす中で面積最小のプロセッサコア構成と，プロセッサコアの持つ命令セットを合成する．**図 1** に示すように，SPADES はコンパイラ，HW/SW 分割系，演算ユニット生成系，HW/SW 生成系から構成される．プロセッサコア構成を決定するうえで核となるのは，HW/SW 分割系と演算ユニット生成系の相互作用である．まず HW/SW 分割系が，入力されたアプリケーションプログラムから，プロセッサコアの持つ命令を最適化し候補となる命令セットを生成する．次に演算ユニット生成系が，生成された命令セットの実行に必要となる演算ユニットを生成し，演算ユニットの面積と遅延値の見積りを行う．続いて HW/SW 分割系が，生成された演算ユニットが付加されたプロセッサコアを合成し，プロセッサコア面積と実行時間を評価する．以上の手順を繰り返すことによって，要求にあったプロセッサコア構成と，最適化された命令セットが得られる．

SIMD 型プロセッサコアの合成において，SIMD 乗算のように演算実行遅延の大きい演算を含む場合，演算をマルチサイクル化して動作周波数を向上させることにより，少ない面積オーバヘッドでより高速なプロセッサコア構成を得ることが期待できる．ただし，アプリケーションプログラムに潜在するデータ依存や条件分岐などのハザードの影響により，必ずしもパイプライン段数を深くすることがアプリケーション実行時間の短縮につながるとは限らないため，プロセッサコアの自動合成では，与えられた実行時間制約の中で最適なパイプライン構成を探索するのが望ましい．SPADES でパイプライン構成の最適化を考えるには，演算ユニット生成系の機能を拡張し，パイプライン演算ユニットの生成を可能にすることで対応できる．

パイプライン演算ユニットの生成には，以下の 3 点が求められる．(i) パイプライン段数の自由な決定，(ii) 演算ユニット生成の高速性，および (iii) 候補の複数列挙である．(i) について，プロセッサコアのパイプライン構成を自動で決定するためには様々な段数のパイプライン構成を評価する必要があり，演算ユニットも様々な段数にパイプライン化できる必要がある．(ii) について，要求にあった最適なプロセッサコア構成を決定するまでに，多数のプロセッサコア構成に対してプロセッサコアの面積およびアプリケーションの実行時間を見積もるため，演算ユニットの面積と遅延の見積りの高速化は重要な要件である．(iii) について，複数の候補を列挙することで，同じ命令集合に対して，何度も演算ユニットの面積や遅延の見積り値を呼び出すことなくプロセッサコア構成を選択できる．

パイプライン演算ユニット生成に関する既存研究として，演算ユニットのパイプライン化アルゴリズムを提案した報告[5),7)] は数多くされているが，パイプライン演算ユニットを自動生成し，自動生成の高速性までを評価した報告は著者の知る限りではない．演算ユニットの自動生成手法としては，Module Compiler[8)] や，SIMD プロセッサコアを対象としたハードウェアユニット生成系[4)] が提案されている．しかし，Module Compiler は，要件 (ii)，(iii) を満たしていないうえに独自の記述言語での仕様記述が必要である．ハードウェアユニット生成系は要件 (i) に対応していない．

以上の背景から，3 つの要件すべてを満たすパイプライン演算ユニットの生成手法および生成系を提案する．提案する生成系は，演算ユニットで実行される命令の集合，パイプライン段数，および面積と遅延の制約値を入力とし，パイプライン化された演算ユニットを複数構成し，制約を満たす構成の面積と遅延見積り値を出力する．パイプライン演算ユニット生成系は，入力された命令セットから必要な機能を抽出し演算ユニットのアーキテクチャテンプレートを決定する部分機能抽出と，アーキテクチャテンプレートに部分機能ユニットを割り当てるアーキテクチャ構成，および決定されたアーキテクチャへのパイプラインレジスタ挿入により実現される．提案するパイプライン演算ユニット生成アルゴリズムにより，適切な位置にパイプラインレジスタが挿入され，面積と遅延の小さい演算ユニットの構成を高速に複数列挙することが可能である．

☆ SPADES は System for Processor Architecture Design with Estimation-type SIMD の略である．

2. パイプライン演算ユニット生成系

本章では，まず生成されるパイプライン演算ユニットが対象とする命令セットを説明し，続いてパイプライン演算ユニット生成系およびパイプライン演算ユニットの見積り手法を提案する．

2.1 命令セット

生成する演算ユニットに割り当てられる命令セットは，(1) 基本命令，(2) SIMD 命令，および (3) 複合命令の 3 種類によって構成される．基本命令は，市販のディジタル信号処理プロセッサ[6] を基本としている．

SIMD 命令とは，SIMD 演算を実行する命令であり，b ビット演算ユニットを用いて同時に n 個の b/n ビットデータを演算することができる．n は SIMD 命令ごとに異なる値を設定でき，梱包数と呼ぶ．SIMD 演算は算術演算，シフト演算，飽和・拡張演算といった，画像処理アプリケーションなどで頻繁に実行される一連の演算を 1 つの演算として実行できる．これら詳細演算に対するパラメータとして符号および飽和・拡張演算の有無，シフト演算のシフト方向およびシフト量を選択できる．たとえば，梱包数 4，符号なし，右 2 ビットシフト，飽和演算ありの SIMD 乗算命令は MUL_4_ur2s と表される．SIMD 命令を**表 1** に示す．複合命令は，基本命令および SIMD 命令を複数個並列に実行する命令である．基本命令および SIMD 命令のあらゆる組合せを複合命令として持つのではなく，アプリケーションに応じて複合命令となる命令の組合せを決定する．

表 1 SIMD 命令
Table 1 SIMD Instructions.

Arithmetic operation	ADD, SUB, MUL, MAC
Shift operation	SRA, SLA, SLL
Bit extend/extract operation	EXTD, EXTR
Others	EXCH

2.2 SIMD 演算ユニット

SIMD 演算ユニットとは，SIMD 演算を実行する演算ユニットであり，1 つの SIMD 演算ユニットに対して，複数の SIMD 演算が割り当てられる．割り当てられた SIMD 演算を SIMD オプションと呼ぶ．たとえば，2 つの SIMD オプション MUL_2_ur2s と MUL_2_ul4s を持つ非パイプライン SIMD 乗算ユニット mul_0 を仮定すると，mul_0 はあるクロックサイクルで 2 つの SIMD オプションのうちのどちらかの演算を実行可能である．並列に実行される必要のある SIMD オプションは，異なる演算ユニットに割り当てられる．

mul_0 の生成を考えたとき，SIMD オプションごとにアーキテクチャを構成すると，**図 2** (a) のように構成される．ここで 2 つの SIMD オプションは並列に実行されることがないので，16 ビット乗算と飽和演算ユニットを 2 組持つ必要はなく，図 2 (b) のような構成が望ましい．冗長なアーキテクチャを生成しないためには，各 SIMD オプションから必要な部分的な機能を抽出し，各機能ごとに構成する必要がある．SIMD 演算を構成している部分的な機能を部分機能，部分機能を実現するハードウェアユニットを部分機能ユニットと呼ぶ．さらに，演算ユニットを実現するのに必要な部分機能の集合とその接続関係を表したものをアーキテクチャテンプレートと呼ぶ．

SIMD 演算ユニットは，加算，乗算などの演算の種類の違いを除いて，算術演算，演算結果のシフト，飽和・拡張演算の手順は一定である．したがって，命令形式に対応して，算術演算部，固定シフト部，飽和・拡張演算部の 3 つの部分機能に分割し，それを順に接続したものを SIMD 演算ユニット共通のアーキテクチャテンプレートとすることができる．部分機能ユニット

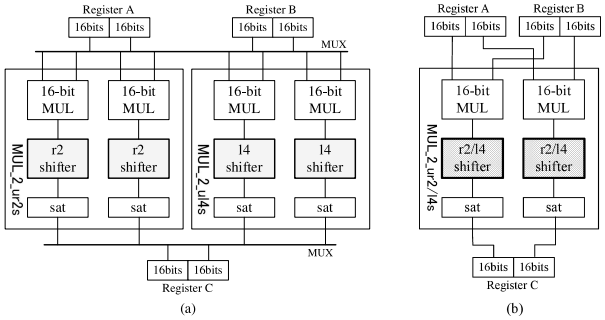

図 2 SIMD 演算ユニット構成：
(a) SIMD オプションごとに生成した mul_0，(b) 部分機能ごとに生成した mul_0
Fig. 2 SIMD functional unit configuration:
(a) mul_0 generated by SIMD options, (b) mul_0 generated by subfunctions.

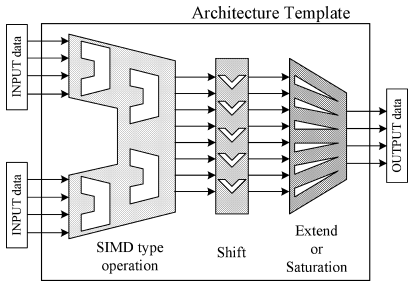

図 3 SIMD 演算ユニットのアーキテクチャテンプレート
Fig. 3 Architecture template of a SIMD functional unit.

は，たとえば 1 梱包演算（A），2 梱包演算（B），4 梱包演算（C）が可能な算術演算部に対して，3 種類の演算の組合せ（A，B，C，AB，BC，CA，ABC）から 7 種類の部分機能ユニットを用意する．ここでたとえば（AB）とは，1 梱包演算と 2 梱包演算が実行可能な組合せを意味する．固定シフト部，飽和・拡張演算部でも 7 種類の部分機能ユニットを用意すると仮定すれば，$7 \times 3 = 21$ 種類の部分機能ユニットを用意することで，$7^3 = 343$ 種類の SIMD 演算ユニットを構成できる．このように，1 つのアーキテクチャテンプレートによって容易に多種類の SIMD 演算ユニットを構成することができる．SIMD 演算ユニットのアーキテクチャテンプレートを**図 3** に示す．

2.3 パイプライン演算ユニット生成系のシステム構成

提案するパイプライン演算ユニット生成系は，**図 4** に示すとおり，与えられた命令セットから部分機能を抽出し，アーキテクチャテンプレートを決定するステージ（部分機能抽出：Subfunction extraction）と，抽出した部分機能を実現する部分機能ユニットを，アーキテクチャテンプレートに割り当てて，演算ユニットのアーキテクチャを構成するステージ（アーキテクチャ構成：Architecture configuration），および構成されたアーキテクチャにパイプラインレジスタを挿入するステージ（パイプラインレジスタ挿入：Pipeline register insertion）からなる．アーキテクチャ構成とパイプラインレジスタ挿入を繰り返すことにより，面積と遅延値の異なるパイプライン演算ユニットが複数構成される．以下，部分機能抽出とアーキテクチャ構成，およびパイプラインレジスタ挿入をそれぞれ説明する．

2.3.1 部分機能抽出

SIMD 命令は，1 つの命令の中に，演算（演算種，梱包数，符号の有無），固定シフト演算（シフト方向，量），飽和・拡張演算（飽和・拡張の有無）といった部分機能の情報を持つ．また SIMD 演算ユニットは，1 つ

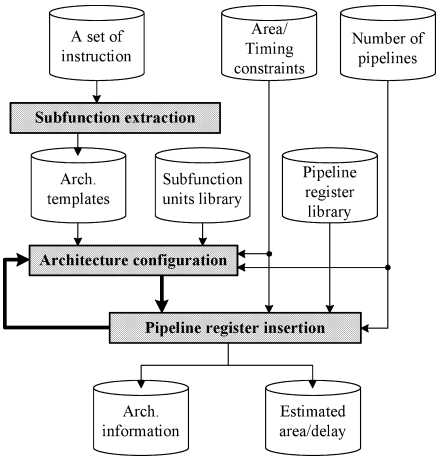

図 4 パイプライン演算ユニット生成系の構成図
Fig. 4 Pipelined functional unit generation system.

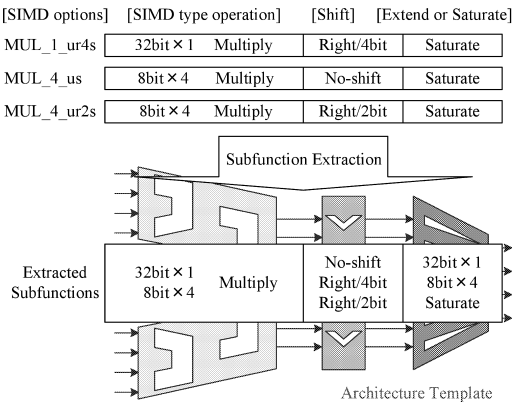

図 5 部分機能抽出とアーキテクチャテンプレートの決定
Fig. 5 Subfunction extraction and determination of architecture template.

の演算ユニットに対して複数の SIMD 命令が SIMD オプションとして割り当てられる．SIMD 演算ユニットの生成には，割り当てられたすべての SIMD オプションから，各部分機能の情報を抽出し，必要最小限の部分機能からなるアーキテクチャテンプレートを決定する．**図 5** に，HW/SW 分割系から与えられた SIMD 命令の集合から部分機能を抽出し，アーキテクチャテンプレートを決定する例を示す．例では，MUL_1_ur4s（1 梱包乗算，符号なし，右 4 ビットシフト，飽和演算），MUL_4_us（4 梱包乗算，符号なし，シフトなし，飽和演算），MUL_4_ur2s（4 梱包乗算，符号なし，右 2 ビットシフト，飽和演算）の 3 つの SIMD オプションから部分機能抽出を行う．3 つの SIMD オプションが持つそれぞれの部分機能から，演算種は乗算で 1 梱包および 4 梱包演算が実行可能，シフト演算はシフトなし，右 4 ビットシフト，および右 2 ビットシフトが

実行可能，飽和・拡張演算は 1 桁包飽和および 4 桁包飽和が実行可能，という必要最小限の部分機能が抽出され，アーキテクチャテンプレートが決定される．

2.3.2 アーキテクチャ構成

前述のとおり，演算ユニットに割り当てられた SIMD オプションから部分機能の抽出を行い，各部分機能に分けて SIMD 演算ユニットを生成することにより，図 2 (b) のように，冗長なハードウェアを含まないアーキテクチャ構成を得られる．アーキテクチャ構成では，部分機能抽出で決定されたアーキテクチャテンプレートに含まれる部分機能に，各部分機能を実現可能な部分機能ユニットを割り当て，SIMD 演算ユニットのアーキテクチャ構成を決定する．ここで，各部分機能について，面積と遅延値の異なる部分機能ユニットを部分機能ユニットライブラリ（図 4）に複数用意しておき，それらの組合せを変化させることで，複数のアーキテクチャ構成を得る．部分機能ユニットの組合せの決定には，3.1.1 項で提案するアーキテクチャ構成アルゴリズムを用いる．アーキテクチャ構成アルゴリズムにより，面積と遅延の小さい部分機能ユニットの組合せを決定し，パイプライン化する前の演算ユニットのアーキテクチャを決定する．

ここで，構成された演算ユニットのアーキテクチャへのパイプラインレジスタ挿入を考えたとき，アーキテクチャテンプレートの部分機能ユニット間にレジスタを挿入したのでは，レジスタが挿入可能な位置が数個に限定されるうえに，各パイプラインステージの遅延値が離散的になり動作周波数向上の妨げになるため効果的でない．また，演算ユニットを論理合成して得られるネットリスト上でゲート間にレジスタを挿入することを考えると，挿入可能な位置が膨大で，遅延値が同程度になる位置を自動で高速に決定するのが難しい．

したがって，パイプラインレジスタの挿入を考慮し，部分機能ユニットを，遅延時間に関してできる限り小さい単位のハードウェアモジュールの集合から構成する手法をとる．このハードウェアモジュールを最小単位モジュールと呼ぶ．最小単位モジュールは，ゲートレベルよりは粒度が粗く部分機能ユニットよりは粒度の細かいハードウェアモジュールで，論理合成の結果から面積と遅延値をあらかじめ見積もることが可能である．最小単位モジュールを縦列結合して部分機能ユニットを構成し，パイプラインレジスタを挿入できる位置をある程度限定することによって，あらかじめ見積もった最小単位モジュールの遅延値をもとに，高速にパイプラインレジスタを挿入することが可能になる．

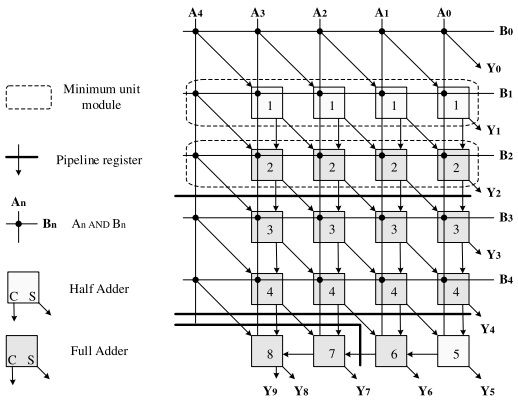

図 6 パイプライン化された乗算ユニットの構成例
Fig. 6 Pipelined carry-save multiplier.

最小単位モジュールの具体例として，Carry-save 型乗算ユニット[5] の構成例を**図 6** に示す．図 6 で，加算器に振られた番号の順でデータが並列に伝播するため，同じ番号の振られた加算器から出力されるデータをパイプラインレジスタに格納することによって，乗算ユニットをパイプライン化することができる．このとき，同じ番号の振られた加算器をまとめたものが最小単位モジュールとなる．このように，演算ユニット内のデータの伝播を意識して同じ構成の内部演算ユニット（図 6 の例では加算器）を並列に並べたものを最小単位モジュールとすることにより，モジュール内部のパスの遅延値が一律に近くなり，結果としてパイプライン化された演算ユニットの各ステージのパスの遅延が一律に近い構成が得られる．

2.3.3 パイプラインレジスタ挿入

パイプラインレジスタ挿入では，アーキテクチャ構成によって決定されたアーキテクチャにパイプラインレジスタを挿入することにより，演算ユニットをパイプライン化する．

アーキテクチャ構成において，アーキテクチャテンプレートに割り当てられる部分機能ユニットを最小単位モジュールから構成することにより，パイプラインレジスタ挿入では，最小単位モジュール間に，各パイプラインステージの遅延値がバランスするようにパイプラインレジスタを挿入することが可能になる．パイプラインレジスタの挿入位置を決定するアルゴリズムは 3.2.1 項で説明する．最小単位モジュールを細かく構成するほど，パイプラインレジスタを挿入できる位置を小刻みに設定できるため，より遅延値が正確にバランスする位置にパイプラインレジスタを挿入できる．

SIMD 演算ユニットについて，たとえば SIMD 乗算ユニットは，図 6 に示す乗算ユニットを梱包数分だ

け並列に並べることにより構成可能である．並列に並べられたそれぞれの乗算ユニットは，図 6 と同様に最小単位モジュールに分割できる．つまり SIMD 演算ユニットは，同じ最小単位モジュールを並列に並べた構成となるため，最小単位モジュールの集合から演算ユニットを構成する本手法は，SIMD 演算ユニットのパイプライン化に特に有効である．

2.4 パイプライン演算ユニットの面積および遅延見積り

上記のとおり，パイプライン演算ユニットはアーキテクチャテンプレートに割り当てられた部分機能ユニットとパイプラインレジスタから構成され，さらに部分機能ユニットは最小単位モジュールから構成される．パイプライン演算ユニットの面積と遅延の見積りには，パイプラインレジスタ，最小単位モジュール，および部分機能ユニットの面積と遅延の見積り値が必要である．以下に，パイプラインレジスタ，最小単位モジュール，部分機能ユニット，およびパイプライン演算ユニットの面積と遅延見積りについて説明する．

パイプラインレジスタについて，挿入される位置によってパイプラインレジスタのビット幅が異なり，面積はパイプラインレジスタのビット幅に依存するため，挿入される位置のビット幅から面積値を見積もる．パイプラインレジスタからのデータの読み出しと書き込みには，挿入される位置によらず一定の遅延がかかり，レジスタを論理合成した結果から遅延を見積もる．最小単位モジュールについて，最小単位モジュールを論理合成し，得られた面積値を面積見積り値，得られた遅延値を遅延見積り値とする．部分機能ユニットについて，最小単位モジュールの縦列結合からなる部分機能ユニットの組合せ回路全体を論理合成し，得られた面積値を面積見積り値，得られた遅延値を遅延見積り値とする．パイプライン演算ユニットについて，パイプライン演算ユニットを構成する部分機能ユニットの面積見積り値とパイプラインレジスタの面積見積り値の和から面積値を見積もる．パイプライン演算ユニットの各ステージの遅延値は，各ステージを構成する最小単位モジュールの遅延見積り値と，パイプラインレジスタの読み出しと書き込みにかかる遅延見積り値の和から見積もる．ただし，与えられたパイプライン段数が 1 段でパイプライン化する必要のない場合は，部分機能ユニットの遅延値の和から見積もる．

パイプライン演算ユニットのパイプラインステージの遅延の見積りについて，最小単位モジュールを縦列結合して得られる組合せ回路全体のクリティカルパス遅延と，最小単位モジュールのクリティカルパス遅延の和をとった遅延見積り値は一般的に異なる．**図 7** に示すように，たとえば 2 つの最小単位モジュール A，B が縦列に接続されており，前段のモジュールの A–M_{A_0} 間，A–M_{A_1} 間の遅延がそれぞれ 5 ns，3 ns であった場合，クリティカルパスは A–M_{A_0} 間で遅延値は 5 ns である．後段のモジュールの M_{B_0}–B 間，M_{B_1}–B 間の遅延がそれぞれ 4 ns，5 ns であった場合，クリティカルパスは M_{B_1}–B 間で遅延値は 5 ns である．最小単位モジュールごとに得られた遅延の和は 5＋5 = 10 ns であるが，結合された 2 つの最小単位モジュールのクリティカルパス経路は A–M_0–B で，クリティカルパス遅延は 5 + 4 = 9 ns である．つまり，最小単位モジュールごとに遅延値を見積もり，見積もられた遅延値の和からパイプラインステージの遅延を見積もる本手法は，パイプラインステージ全体の遅延を大きく見積もる可能性がある．パイプライン段数が 1 段で，部分機能ユニットの遅延見積り値の和からパイプライン演算ユニットの遅延値を見積もる場合も同様である☆．

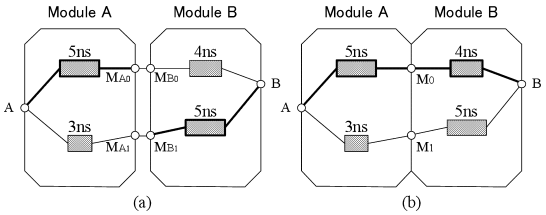

図 7 見積り遅延値の誤差：(a) 最小単位モジュールごとの遅延見積り，(b) 組合せ回路全体の遅延値
Fig. 7 Error of estimated delay: (a) Estimated delay for each minimum unit module, (b) Critical path delay of the entire modules.

3. パイプライン演算ユニット生成アルゴリズム

本章では，図 4 のアーキテクチャ構成とパイプラインレジスタ挿入に着目し，パイプライン演算ユニット生成において問題となるアーキテクチャ構成問題を定義し，パイプライン演算ユニットのアーキテクチャ構成アルゴリズムを提案する．また，構成されたアーキテクチャにパイプラインレジスタを挿入する際に問題となるパイプラインレジスタ挿入問題を定義し，パイプラインレジスタ挿入アルゴリズムを提案する．

3.1 アーキテクチャ構成問題

パイプライン演算ユニット u で実行可能な命令集

☆ 本手法により見積もられたパイプラインステージの遅延値と，最小単位モジュールとパイプラインレジスタを結合し論理合成して得られた遅延値との比較を，4 章の表 4 に示す計算機実験で行っている．その結果，大きな誤差はないことが分かる．

合 I_u を u の命令集合と呼ぶ．面積制約を満たすとは，パイプライン演算ユニット u の面積が制約値 a_{limit} より小さいことであり，遅延制約を満たすとは，パイプライン演算ユニット u の中で最も遅延の大きいパイプラインステージの遅延値が，制約値 d_{limit} より小さいことである．このときパイプライン演算ユニットのアーキテクチャ構成問題とは，入力として，(1) 命令集合 I から決定されるアーキテクチャテンプレート，(2) パイプライン段数 p，(3) 面積制約 a_{limit}，および (4) 遅延制約 d_{limit} を与えられたときに，面積制約と遅延制約を満たし，$I \subseteq I_u$ となるパイプライン演算ユニット u の構成を複数個出力することである．

3.1.1 アーキテクチャ構成アルゴリズム

アーキテクチャ構成問題は，制約を満たす複数の解を高速に列挙する必要があり，得られる個々のアーキテクチャ構成は，面積と遅延の小さいものであることが望ましい．したがって以下のような手法をとる．

まず，部分機能抽出で決定されたアーキテクチャテンプレートに基づいて，非パイプライン演算ユニットを構成する．初期構成として，すべての部分機能に対して，部分機能ユニットの中から遅延が最も小さい部分機能ユニットを選ぶ．構成された非パイプライン演算ユニット u_{np} は面積値 $a(u_{np})$ および遅延値 $d(u_{np})$ を持つ．面積値 $a(u_{np})$ が面積制約を満たさなければ，式 (4) に示す評価値に従って部分機能ユニットの組合せを変更する．

次に，与えられたパイプライン段数から演算ユニット u_{np} にパイプラインレジスタを挿入し，演算ユニット u_{np} をパイプライン化する．パイプライン化された演算ユニットの面積値は，パイプラインレジスタ挿入前の演算ユニットの面積値 $a(u_{np})$ と，挿入されたすべてのパイプラインレジスタの面積値の和から見積もれる．遅延値は，パイプライン化された演算ユニットのすべてのステージの中で最も遅延の大きいステージの遅延値と，パイプラインレジスタへの書き込みにかかる遅延の和から見積もれる．ここで，パイプラインレジスタの挿入位置を決定する前に，与えられたパイプライン段数から，パイプライン演算ユニットの面積の最小値および遅延の最小値を算出する．パイプラインレジスタの面積値は，挿入される位置によって格納されるデータのビット幅が異なるため挿入される位置に依存するが，与えられたパイプライン段数から，挿入されるレジスタ面積の総和の最小値をあらかじめ求められる．たとえば，最大で m 段までパイプライン化できるアーキテクチャ構成（最小単位モジュール数が m）で，各挿入位置のパイプラインレジスタを面積値の小さい順に並べたものを $\{r_1, r_2, \ldots, r_{m-1}\}$，パイプラインレジスタ r の面積値を $a(r)$ とする．与えられたパイプライン段数が p 段のとき，パイプラインレジスタ面積の最小値 $a_{r,min}$ は，

$$a_{r,min} = \sum_{j=1}^{p-1} a(r_j) \tag{1}$$

と見積もれる．パイプラインレジスタの挿入位置を決定する前の最小面積値と最小遅延値を持つ仮想的なパイプライン演算ユニットを u_{min} とすると，式 (1) より u_{min} の最小面積値は，

$$a(u_{min}) = a(u_{np}) + a_{r,min} \tag{2}$$

と見積もれる．最小遅延値は，演算ユニットを p 等分した場合の 1 ステージ分の遅延値であり，パイプラインレジスタの読み出しと書き込みにかかる遅延値を d_{reg} とすれば，

$$d(u_{min}) = \frac{d(u_{np})}{n} + d_{reg} \tag{3}$$

と見積もれる．最小遅延値が遅延制約を満たさなければ解探索を終了する．遅延制約を満たしかつ面積制約を満たさなければ，パイプラインレジスタを挿入せずに式 (4) に示す評価値に従って部分機能ユニットの組合せを変更する．遅延制約と面積制約をともに満たせば，パイプラインレジスタの挿入位置を決定する．

パイプラインレジスタの挿入位置の決定は，3.2.1 項で提案するパイプラインレジスタ挿入アルゴリズム（図 12）に従い，パイプライン化された演算ユニット u の面積値 $a(u)$ および遅延値 $d(u)$ を見積もる．遅延見積り値が遅延制約を満たさなければ解探索を終了する．遅延制約を満たしかつ面積制約を満たさなければ，式 (4) に示す評価値に従って部分機能ユニットの組合せを変更する．遅延制約と面積制約をともに満たせば，見積り値を解の 1 つとして出力する．

続いて，**図 8** に示すように部分機能ユニットの組合せの変更を行う．入力されたアーキテクチャテンプレートの部分機能の集合を $F = \{f_1, f_2, \ldots, f_q\}$ とする．F をもとに，部分機能ユニットライブラリから f_i $(i = 1, \ldots, q)$ を実現可能な部分機能ユニットの集合 S_i $(i = 1, \ldots, q)$ が選択される．部分機能ユニット $s \in S_i$ は面積値 $a(s)$ と遅延値 $d(s)$ を持つ．今，各 f_i を $s_i \in S_i$ によって実現していると仮定する．この部分機能ユニット s_i を，部分機能ユニット集合 S_i の中で現在の部分機能ユニットより面積が小さい中で最も遅延の小さい部分機能ユニット s_i' へ置き換える．このとき置き換えを行った部分機能 f_i の評価値 $c(f_i)$ は

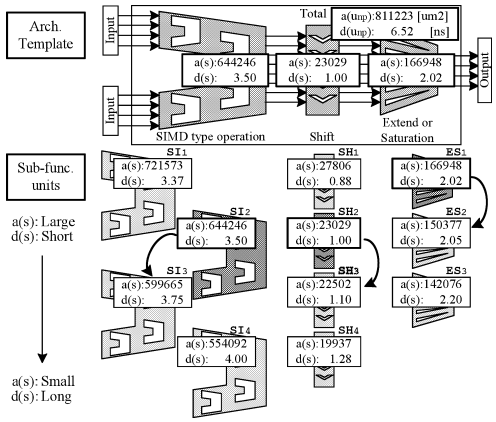

図 8 部分機能ユニットの置き換え
Fig. 8 Subfunctional unit replacing.

$$c(f_i) = \frac{d(s'_i) - d(s_i)}{a(s_i) - a(s'_i)} \tag{4}$$

のように表せる．すべての部分機能 f_i $(i = 1, \ldots, q)$ について同様の置き換えを行い，評価値 c が最も小さい値をとった部分機能 $f_{i_{min}}$ について，$s_{i_{min}}$ から $s'_{i_{min}}$ に部分機能ユニットを変更する．

図 8 に，SIMD 演算ユニットの部分機能ユニット変更の例を示す．図 8 において，アーキテクチャテンプレートは算術演算，シフト演算，および飽和・拡張演算を行う部分機能を持つ．今，算術演算部に面積 644,246 μm^2 遅延 3.50 ns の部分機能ユニット SI_2，シフト演算部に面積 23,029 μm^2 遅延 1.00 ns の部分機能ユニット SH_2，および飽和・拡張演算部に面積 166,948 μm^2 遅延 2.02 ns の部分機能ユニット ES_1 が割り当てられ，全体で面積 834,223 μm^2 遅延 6.52 ns の演算ユニットが構成されていると仮定する．つまり，現在演算ユニットは (SI_2, SH_2, ES_1) で構成されている．

ここで，すべての部分機能について，それぞれの部分機能ユニット集合の中で現在の部分機能ユニットより面積が小さい中で最も遅延の小さい部分機能ユニットへ置き換えを行い，評価値を算出する．まず算術演算部に着目し，$(\underline{SI_2}, SH_2, ES_1)$ を $(\underline{SI_3}, SH_2, ES_1)$ に置き換えることを考える．このとき評価値は，

$$\frac{3.75 - 3.50}{644246 - 599665} = 5.61 \times 10^{-6} \tag{5}$$

となる．次にシフト演算部に着目し，$(SI_2, \underline{SH_2}, ES_1)$ から $(SI_2, \underline{SH_3}, ES_1)$ へ置き換えを考えると評価値は

$$\frac{1.10 - 1.00}{23029 - 22502} = 1.90 \times 10^{-3} \tag{6}$$

となる．続いて飽和・拡張演算部で $(SI_2, SH_2, \underline{ES_1})$ から $(SI_2, SH_2, \underline{ES_2})$ へ置き換えを考えると評価値は

Step 0 アーキテクチャテンプレートに基づいて非パイプライン演算ユニットを構成する．
初期構成として，すべての部分機能について最も遅延の小さい部分機能ユニットを選択する．
Step 1 非パイプライン演算ユニットの面積値 $a(u_{np})$ と遅延値 $d(u_{np})$ を見積もる．
Step 1-1 $a(u_{np}) > a_{limit}$ ならば **Step 5** で部分機能ユニットの組合せを変更する．
Step 2 与えられたパイプライン段数からパイプラインレジスタの最小面積値 $a_{r,min}$ を見積もり，パイプライン演算ユニットの最小面積値 $a(u_{min})$ と最小遅延値 $d(u_{min})$ を算出する．
Step 2-1 $d(u_{min}) > d_{limit}$ ならば解探索を終了する．
Step 2-2 $a(u_{min}) > a_{limit}$ ならば **Step 5** で部分機能ユニットの組合せを変更する．
Step 3 **パイプラインレジスタ挿入アルゴリズム**に従ってパイプラインレジスタを挿入し，パイプライン演算ユニットの面積値 $a(u)$ と遅延値 $d(u)$ を見積もる．
Step 3-1 $d(u) > d_{limit}$ ならば解探索を終了する．
Step 3-2 $a(u) > a_{limit}$ ならば **Step 5** で部分機能ユニットの組合せを変更する．
Step 4 制約を満たす解の 1 つとして，見積り値 $a(u)$，$d(u)$ を出力する．
Step 5 アーキテクチャテンプレートを構成するすべての部分機能について部分機能ユニット変更の評価値 c を算出し，c が最も小さい値をとった構成への変更を行う．変更できる部分機能ユニットがなければ解探索を終了する．
Step 6 **Step 1** から **Step 5** を繰り返す．

図 9 アーキテクチャ構成アルゴリズム
Fig. 9 Architecture configuration algorithm.

$$\frac{2.05 - 2.02}{166948 - 150377} = 1.81 \times 10^{-6} \tag{7}$$

となる．以上から，最も評価値が小さい飽和・拡張演算部の部分機能ユニットの交換が決定される．このとき，それぞれの部分機能ユニットの集合の中ですべての部分機能ユニットの面積と遅延の間にトレードオフの関係が成立しており，評価値 c が負になることはない．

初期構成から，以上の手順を，変更できる部分機能ユニットがなくなるか，遅延制約を満たさなくなるまで続ける．アーキテクチャ構成アルゴリズムを**図 9** に示す．

評価値 c は，部分機能ユニットを変更したときの，演算ユニットの面積削減量に対する遅延増大量の比を示す．したがって評価値の小さい部分機能ユニットを変更することによって，遅延の増大を抑えかつ面積が大きく削減された構成へと変更されるため，面積と遅延値の小さい構成のみを得ることができる．部分機能ユニット変更による面積と遅延値の変化を**図 10** に示す．本アルゴリズムでは，面積最大・遅延最小の構成から，部分機能ユニットの変更を繰り返すことで徐々に面積最小・遅延最大の構成へと遷移する．このとき評価値 c は，図 10 において遷移先の構成への傾きの

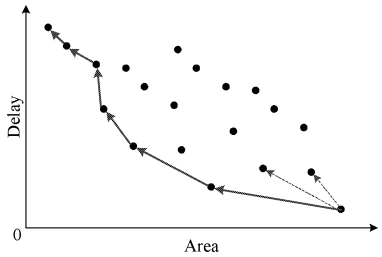

図 10 演算ユニットの面積と遅延値の遷移
Fig. 10 Effect of area reduction to delay increasing.

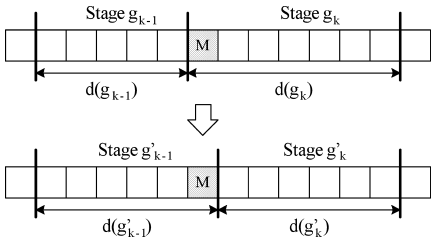

図 11 遅延値のリバランス
Fig. 11 Rebalancing.

絶対値を示す．傾きの絶対値の小さい構成へと遷移させることによって，遅延の増大を小さく面積の削減を大きくできるため，面積と遅延値の小さい構成を得られることが分かる．4 章の図 13 (a)–(d) に示される実験結果でも，図 10 と同様の結果が得られており，評価値 c の効果が確認できる．

次に，アーキテクチャ構成アルゴリズムの時間計算量を見積もる．部分機能ユニットの総数 $\sum|S_i|$ を n とする．同じ部分機能を実現する部分機能ユニットを，遅延の小さい順にそれぞれ整列しておけば，図 9 の **Step 0–1** は $O(1)$ で実行可能である．**Step 2** はパイプラインレジスタを遅延の小さい順に整列するため，現在の演算ユニットを構成する最小単位モジュール数を m とすると，$O(m \log m)$ で実行可能である．**Step 3** のパイプラインレジスタ挿入の時間計算量については 3.2.1 項で解説するが，$O(m^2)$ で実行可能である．**Step 4** は $O(1)$ で実行可能である．すべての部分機能について c を算出するので **Step 5** は $O(q)$ で求められる．**Step 1–5** は最大で部分機能ユニットの総数回繰り返されるので，$m \geq q$，$m > \log m$ より時間計算量は $O(nm^2 + n \log n)$ である．次に空間計算量を見積もる．すべての部分機能ユニットに含まれる最小単位モジュール数を m_{total} とすれば，m_{total} の最小単位モジュールとパイプラインレジスタを保存し，アーキテクチャテンプレート内の m の領域に最小単位モジュールを割り当て，p の領域にパイプラインレジスタを割り当てるため，$m_{total} \geq m$，$m_{total} \geq p$ より，空間計算量は $O(m_{total})$ である．

3.2 パイプラインレジスタ挿入問題

与えられたパイプライン段数が p 段ならば，演算ユニットを p 分割する必要があり，p 分割されてできたパイプラインステージを要素とする集合を $G = \{g_1, g_2, \ldots, g_p\}$ とする．パイプラインステージ $g \in G$ は遅延値 $d(g)$ を持つ．集合 G の中で最も遅延値の大きいステージを g_{max} とする．

このとき，パイプラインレジスタ挿入問題とは，(1) 演算ユニットを構成する最小単位モジュールの遅延値と (2) パイプライン段数を入力とし，$d(g_{max})$ を最小化することである．

3.2.1 パイプラインレジスタ挿入アルゴリズム

$d(g_{max})$ の値を小さくすることと，演算ユニットのアーキテクチャを各パイプラインステージの遅延値が均等になるように分割することは等価である．まず，非パイプライン演算ユニットの遅延見積り値 $d(u_{np})$ を p 等分して遅延目安値 d_{opti} を求め，演算ユニットの入力側から d_{opti} に最も近い値をとる位置にレジスタを順次挿入する．このとき，各ステージの遅延値が遅延目安値 d_{opti} を超えることを許す．

p 段にパイプライン化した後に，最も遅延の大きいステージ g_k の遅延値 $d(g_k)$ に着目し，1 つ前（入力側）のステージ g_{k-1} の遅延値 $d(g_{k-1})$ とのリバランスを行う．**図 11** に示すように，ステージ g_k の先頭の最小単位モジュール 1 つをステージ g_{k-1} に移動し，移動後の遅延値をそれぞれ $d(g'_{k-1})$，$d(g'_k)$ とする．リバランスとは，$d(g'_{k-1}) \leq d(g_k)$ を満たす場合にパイプラインレジスタの挿入位置を変更することである．パイプラインレジスタの挿入位置が変更された場合，同様に再度最も遅延の大きいステージのリバランスを行う．$d(g'_{k-1}) > d(g_k)$ ならばパイプラインレジスタの挿入位置を変更せずに，$d(g_k)$ の値にパイプラインレジスタ転送にかかる遅延 d_{reg} を加えた値を，パイプライン演算ユニットの遅延見積り値 $d(u)$ とする．以上の手順を，すべてのステージのリバランスが行われるか，パイプラインレジスタの挿入位置が変更されなくなるまで繰り返す．パイプラインレジスタ挿入アルゴリズムを**図 12** に示す．

次に，このアルゴリズムの時間計算量を求める．図 12 の **Step 2** は，最小単位モジュールの遅延値と遅延目安値との比較を m 回行うので，$O(m)$ で実行可能である．**Step 3** は，遅延の最も大きいステージの探索に $O(p)$ かかり，最大で p 回繰り返される．ここで，演算ユニットは最大で最小単位モジュールの数だけパイプライン化が可能であるので，パイプライン段数 p の

Step 1 演算ユニットの遅延見積り値 $d(u_{np})$ を p 等分し，遅延目安値 d_{opti} を求める．

Step 2 演算ユニットの入力側から，d_{opti} に最も近い値をとる位置にレジスタを順次挿入する．

Step 3 最も遅延の大きいステージ g_k の遅延値 $d(g_k)$ に着目し，1 つ前（入力側）のステージ g_{k-1} の遅延値 $d(g_{k-1})$ とのリバランスを行う．

Step 3-1 $d(g'_{k-1}) > d(g_k)$ ならばパイプラインレジスタの挿入位置を変更せずに，$d(g_k)$ にパイプラインレジスタの転送遅延 d_{reg} を加えた値を，パイプライン演算ユニットの遅延見積り値 $d(u)$ として出力し，解探索を終了する．

Stes 3-2 $d(g'_{k-1}) \leq d(g_k)$ ならば，パイプラインレジスタの挿入位置を変更する．

Stes 4 **Step 3** をすべてのパイプラインステージについて繰り返す．

図 12 パイプラインレジスタ挿入アルゴリズム
Fig. 12 Pipeline register insertion algorithm.

最大値は m である．以上から，時間計算量は $O(m^2)$ である．次に，空間計算量を求める．アーキテクチャ構成から与えられた m 個の最小単位モジュールに対し，パイプラインレジスタを挿入する位置を決定するので，空間計算量は $O(m)$ である．

4. 実験結果

提案した手法を C 言語を用いて Sun Ultra SPARC（version9，750 MHz，メモリ 256 MB）上に実装し，SIMD 乗算ユニットに適用した．コンパイラは gcc（version 2.9）を使用した．部分機能ユニットおよび最小単位モジュールはあらかじめ VHDL で記述し，Design Compiler を用いて論理合成して面積値および遅延値を得た．セルライブラリには VDEC ライブラリ（CMOS 0.35 [μm] テクノロジー）を用いた☆．比較のため，すべての部分機能ユニットの組合せを列挙し，さらに与えられたパイプライン段数からパイプラインレジスタを挿入できるすべてのパターンで挿入して，パイプライン演算ユニットの面積値および遅延値を見積もった（全列挙手法）．全列挙手法におけるパイプライン演算ユニットの見積りは，2.4 節で説明した提案手法の見積りと同様である．

提案手法と全列挙手法に対し，それぞれに**表 2** に示す SIMD 乗算命令を SIMD オプションとして与え，パイプライン SIMD 乗算ユニットを生成した．与えられた SIMD オプションから，部分機能抽出により必要となるアーキテクチャテンプレートは一意に決定され，演算は 1 梱包乗算と 4 梱包乗算が実行可能，シフト演算はシフトしない，右 4 ビットシフト，および右 6 ビットシフトが実行可能，飽和・拡張演算は飽和処理しない，1 梱包と 4 梱包の飽和演算が実行可能な部分機能を有する．それぞれの部分機能を実現する部分機能ユニットとして，**表 3** に示す部分機能ユニットが選択されアーキテクチャ構成に用いられた．ここで算術演算部について，回路面積縮小のために 8 ビット乗算の結果を利用して 32 ビット乗算を実現する回路構成とした．したがって表 3 では，8 ビット乗算までを行う部分機能と，その結果を利用して 32 ビット乗算の結果を生成する部分機能に分かれている．

全列挙手法との比較シミュレーションにおいて評価すべき点は，(1) 提案アルゴリズムで得られる解の品質，(2) 制約を満たす構成の列挙，(3) パイプライン

☆ 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．

表 2 入力として与えた SIMD 乗算命令
Table 2 Given SIMD instruction set.

mul_1_uw	mul_1_us
mul_4_uw	mul_4_us
mul_4_ur4w	mul_4_ur4s
mul_4h_ur6w	mul_4l_ur6w

表 3 アーキテクチャ構成に用いられる部分機能ユニット
Table 3 Subfunctional units used in the architecture template.

8 ビット乗算部の	最小単位モジュール数：15	
部分機能ユニット	面積 [μm^2]	遅延 [ns]
候補 1	721,573	3.37
候補 2	644,246	3.50
候補 3	599,665	3.75
候補 4	554,092	4.00
候補 5	507,475	4.39
32 ビット乗算結果生成	最小単位モジュール数：49	
部の部分機能ユニット	面積 [μm^2]	遅延 [ns]
候補 1	170,224	2.00
候補 2	122,688	2.40
候補 3	106,976	2.61
候補 4	98,960	3.04
候補 5	79,040	3.58
候補 6	41,024	7.22
シフト部の	最小単位モジュール数：2	
部分機能ユニット	面積 [μm^2]	遅延 [ns]
候補 1	27,806	0.88
候補 2	23,029	1.00
候補 3	22,502	1.10
候補 4	19,937	1.28
候補 5	18,331	1.47
候補 6	8,737	2.17
飽和部の	最小単位モジュール数：6	
部分機能ユニット	面積 [μm^2]	遅延 [ns]
候補 1	166,948	2.02
候補 2	150,377	2.05
候補 3	142,076	2.20
候補 4	118,720	2.35
候補 5	103,574	2.52

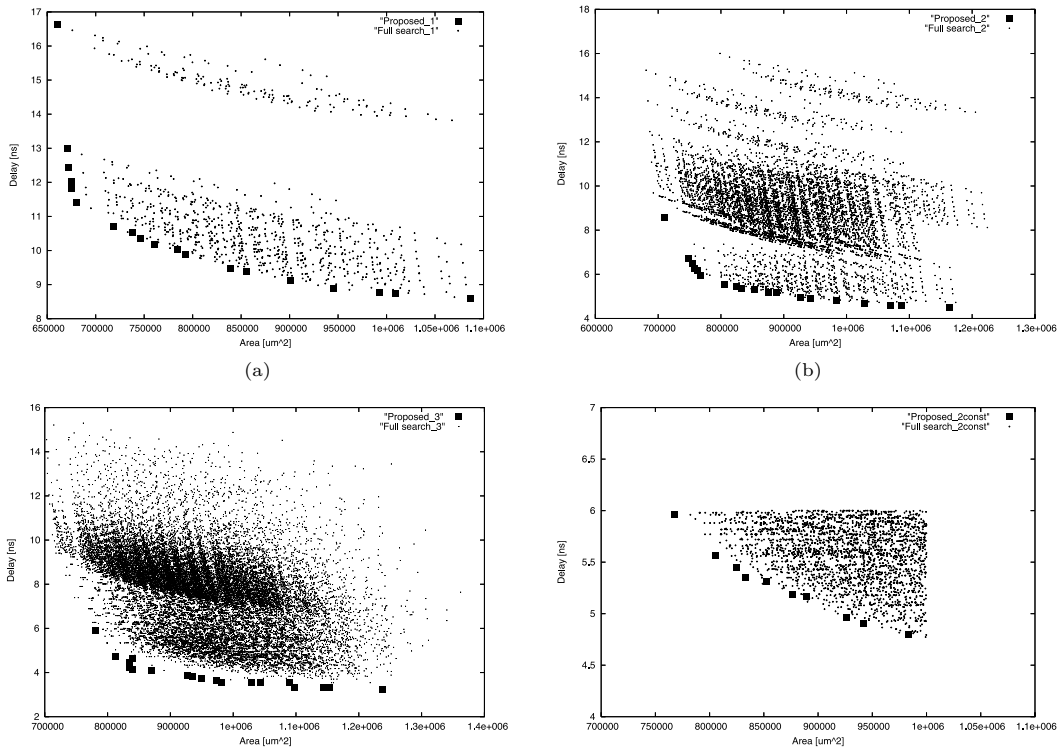

図 13 SIMD 演算ユニット構成：(a) 非パイプライン，制約なし．
(b) 2 段パイプライン，制約なし．(c) 3 段パイプライン，制約なし．
(d) 2 段パイプライン，制約；面積：1,000,000 [μm^2]，遅延：6 [ns]

Fig. 13 SIMD functional unit configuration:
(a) The stage number 1, no constraints,
(b) The stage number 2, no constraints,
(c) The stage number 3, no constraints,
(d) The stage number 2, area/delay constrains: 1,000,000 [μm^2]/6 [ns].

レジスタ挿入位置の妥当性，(4) 見積もられた面積および遅延値の信頼性，および (5) パイプライン演算ユニット生成の高速性の 5 点である．

評価点 (1) に関して，両手法により構成されたパイプライン SIMD 乗算ユニットの面積と遅延値をプロットしたグラフを**図 13** (a)–(d) に示す．図 13 に示した面積および遅延値は，提案手法と全列挙手法のどちらも 2.4 節で解説した見積り手法を用いて見積もった値である．図 13 (a) は，非パイプライン構成，制約を与えなかったときの全列挙手法との比較を示す．以下，図 13 (b) は，2 段パイプライン構成，制約なし，図 13 (c) は，3 段パイプライン構成，制約なし，図 13 (d) は，2 段パイプライン構成，面積制約 1,000,000 μm^2，遅延制約 6 ns をそれぞれ与えたときの面積および遅延見積り値の比較を示す．図 13 で与えた 1〜3 段のすべてのパイプライン段数において，全列挙手法で列挙された構成の中で面積と遅延値の両方が小さく，面積と遅延のトレードオフのとれた有用なパイプライン演算ユニットのみを構成できた．

評価点 (2) に関して，図 13 (d) から，面積および遅延制約を与えたとき，制約を満たす構成のみを複数列挙していることが確認できる．

評価点 (3) に関して，**表 4** に，図 13 (d) の実験で提案手法によって生成されたパイプライン演算ユニットの各ステージの遅延値と遅延目安値 d_{opti} との比較を示す．遅延目安値に対する最も遅延の大きいステージの遅延値は，平均で 1%未満，最大で 1.62%であり，最小単位モジュールが遅延に関して十分小さく，さらにパイプラインレジスタ挿入アルゴリズムによりレジスタが遅延のバランスする位置に挿入されていることが分かる．また，図 13 において，面積と遅延がともに小さい構成が出力される条件は，面積と遅延の小さい部分機能ユニットが選択され，かつ遅延のバランスする位置にパイプラインレジスタが挿入されているこ

表 4 図 13 (d) における面積および遅延の見積り値と論理合成値
Table 4 Estimated values and mesured values for actual circuit in Fig. 13 (d).

提案手法による見積り値						回路全体を論理合成した値			
面積 [μm^2]	遅延 [ns]	1 段目 [ns]	2 段目 [ns]	d_{opti} [ns]	増加率 [%]	面積 [μm^2]	遅延 [ns]	1 段目 [ns]	2 段目 [ns]
983,192	4.46	**4.46**	4.34	4.40	1.36	1,078,823	4.44	**4.44**	4.30
942,211	4.57	**4.57**	4.48	4.53	0.88	1,059,631	4.52	**4.52**	4.45
926,499	4.62	**4.62**	4.52	4.58	0.87	1,022,394	4.69	**4.69**	4.49
889,066	4.83	4.70	**4.83**	4.77	1.26	1,000,054	4.67	4.67	**4.77**
876,173	4.86	**4.86**	4.84	4.85	0.21	981,369	4.82	**4.82**	4.81
852,817	4.98	4.87	**4.98**	4.92	1.02	977,459	4.94	4.85	**4.94**
833,079	5.02	**5.02**	5.00	5.01	0.20	948,449	4.96	**4.96**	4.95
825,063	5.12	**5.12**	5.08	5.10	0.39	926,313	5.05	**5.05**	5.04
805,143	5.23	**5.23**	5.15	5.19	0.58	925,186	5.16	**5.16**	5.09
768,127	5.63	**5.63**	5.45	5.54	1.62	867,549	5.62	**5.62**	5.42

表 5 図 13 (d) における同一面積の構成の遅延見積り値比較
Table 5 Estimated delay of the circuit which has the similar area in Fig. 13 (d).

提案手法による見積り値				回路全体を論理合成した値			
面積 [μm^2]	遅延 [ns]	1 段目	2 段目	面積 [μm^2]	遅延 [ns]	1 段目	2 段目
926,499	4.62	**4.62**	4.52	1,022,394	4.59	**4.59**	4.47
926,639	4.85	4.30	**4.85**	1,021,722	4.80	4.28	**4.80**
926,454	4.91	**4.91**	4.70	1,024,147	4.85	**4.85**	4.67
926,651	5.01	4.67	**5.01**	1,017,891	4.94	4.66	**4.94**
926,493	5.27	**5.27**	4.31	1,025,018	5.23	**5.23**	4.25

とであり，図 13 で面積と遅延がともに小さい構成が出力されていることから，遅延のバランスする位置にレジスタが挿入されているといえる．

評価点（4）に関して，**表 5** に図 13 (d) の実験で得られた構成の中で，ほぼ同じ面積でかつ遅延時間が近いいくつかの構成（全列挙手法で得られたものを含む）の面積および遅延見積り値と，それらと同じ構成の演算ユニット全体を Design Compiler を用いて論理合成して得た回路の面積および遅延値を示す☆．表 5 に示す結果から，面積見積り値の近い 5 つの構成（ばらつきが 0.1%未満）について，それらを改めて論理合成した結果，面積値のばらつきが 0.2%未満で，また遅延値については，表 4 に示した 5 つの構成すべてにおいて，同じ構成の異なるステージに関しても，異なる構成に関しても大小関係が保存されていることが確認できた．すなわち，提案アルゴリズムで列挙された候補回路が，実際の回路においても同一面積の回路の中で最速の構成であるといえる．

さらに，表 4 に図 13 (d) の実験で提案手法によって生成されたすべての構成の面積および遅延見積り値と，同じ構成を持つ演算ユニットの回路全体を論理合成して得られた面積および遅延値を示す．2.4 節で説明したように提案手法の遅延見積りがパイプラインステージの遅延の悲観的な見積りであることから，回路全体の論理合成結果に対して遅延見積り値は増加すると予想されるが，増加率は平均で 0.92%，最大で 1.35%に収まった．要因として，対象となる乗算ユニットの最小単位モジュールが図 6 に示すように全加算器を並列に並べた構成であるため，最小単位モジュールの中を通るパスの遅延値がほぼ一律であったことが考えられる．また表 4 に示す結果についても，遅延見積り値の大小関係と論理合成によって得られた遅延値の大小関係が，同じ構成の異なるステージに関しても，異なる構成に関しても一致していることが確認できる．したがって，提案アルゴリズムで列挙された面積と遅延の小さい構成は，実際の回路においても面積と遅延の小さい構成である．

評価点（5）に関して，**表 6** に図 13 (a)–(d) に示した計算機実験において構成されたパイプライン演算ユニットの構成数と，CPU 時間の比較を示す．表 6 から，すべての実験において提案手法は構成数が圧倒的に少なく，計算コストについても，アーキテクチャ構成の計算コストを表すユーザ時間 T_u の比較から，提案アルゴリズムの方が高速であることが分かる．

続いて，全列挙手法のボトルネックになっている処理の解析から提案手法の有効性を評価する．まず全列挙手法の CPU 時間について，アーキテクチャ構成の計算コストを表すユーザ時間 T_u と，構成の出力の計算コストを表すシステム時間 T_s に分けて計測した．

☆ 部分機能ユニット同様，セルライブラリには VDEC ライブラリ（CMOS 0.35 [μm] テクノロジー）を用いた．

表 6 構成数と CPU 時間
Table 6 The number of enumerated configurations and the CPU time.

段数	構成数		CPU 時間 [s]					
	提案手法	全列挙手法	提案手法		全列挙手法			
			T_u	T_s	T_u	$T_{u,rep}$	$T_{u,pipe}$	T_s
1	19	900	0.012	0.083	0.0533	0.0528	0.00	0.125
2	19	63,000	0.013	0.081	1.45	0.0486	1.40	0.230
2+consts.	10	2,388	0.011	0.082	1.23	0.0621	1.17	0.190
3	19	2,170,800	0.011	0.080	49.6	0.0467	49.5	2.53

さらに，アーキテクチャ構成の計算コスト T_u を，部分機能ユニットの交換にかかるユーザ時間 $T_{u,rep}$ とパイプラインレジスタの挿入にかかるユーザ時間 $T_{u,pipe}$ に分けて計算コストを計測した．

まず，アーキテクチャ構成の計算コストと構成出力の計算コストについて，構成出力の計算コストはアーキテクチャ構成の計算コストよりも十分に小さく，アーキテクチャ構成全体の計算コストは，出力される構成数にほとんど依存していない．また部分機能ユニット交換の計算コスト $T_{u,rep}$ と，パイプラインレジスタ挿入の計算コスト $T_{u,pipe}$ の比から，アーキテクチャ構成の計算コストのほとんどをパイプラインレジスタ挿入が占めており，アーキテクチャ構成全体の計算コストは，パイプラインレジスタ挿入の計算コストに大きく依存している．

この結果から，パイプラインレジスタ挿入を高速化することに大きな意義があり，また提案アルゴリズムのアーキテクチャ構成が高速なのは，提案したパイプラインレジスタ挿入アルゴリズムが高速であることが大きく寄与しているものと考えられる．

また得られた CPU 時間の大きさについて，たとえば SPADES の HW/SW 分割[10] から 3 段パイプライン構成の演算ユニットを構成する要求（図 1）を 1,000〜10,000 回受けたと仮定する．実験結果から，全列挙手法では数十時間〜数百時間の時間を要するが提案手法では数十秒から数分で要求に応えられるため，提案手法によるパイプライン演算ユニット生成は高速である．

SIMD 型プロセッサコアの自動合成の過程を考えたとき，パイプライン演算ユニット生成系から有用な少数の構成が高速に列挙されるため，HW/SW 分割系も面積と遅延の小さいパイプライン演算ユニットを持った優れたプロセッサコア構成を高速に探索することが可能となる．以上から，提案手法の有効性を示せたと考える．

5. おわりに

SIMD 型プロセッサコアの自動合成のためのパイプライン演算ユニット生成手法を提案し，与えられた命令セットとパイプライン段数，面積・遅延の制約値から，面積と遅延値の小さい有用な演算ユニット構成のみを複数列挙することができた．本稿では，論理合成により得られた最小単位モジュールの遅延値に従ってパイプラインレジスタ挿入位置を決定したが，実際に配置配線を行う工程まで考慮すると配線の負荷の大小により必ずしもネットリストの遅延値が実際の遅延値になるとは限らないため，遅延見積りの精度を上げるためには，配線の負荷を考慮する必要がある．また，パイプラインレジスタが電力の消費と密接に関係していることから，消費電力の評価にも取り組みたい．

謝辞 本研究は半導体理工学研究センター（STARC）との共同研究の成果の一部である．また本研究の一部は，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．

参 考 文 献

1) Garcia, C., Lario, R., Prieto, M., Pinuel, L. and Tirado, F.: Vectorization of multigrid codes using SIMD ISA extensions, *Proc. IPDPS 03*, pp.58–65 (2003).
2) Intel: Pentium 4 Processor on 90 nm Technology SSE3 Instructions (2004). http://www.intel.com/technology/itj/2004/volume08issue01/art01_microarchitecture/p06_sse.htm
3) Kawazu. H., Uchida, J., Miyaoka, Y., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: Suboperation parallelism optimization in SIMD processor core synthesis, *IEICE Trans. Fundamentals*, Vol.E88-A, No.4, pp.876–884 (2005).
4) Miyaoka, Y., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: A hardware unit generation algorithm for a hardware/software cosynthesis system of digital signal processor cores with packed SIMD type instructions, *J. Inf. Process. Soc. Jpn.*, Vol.43, No.5, pp.1191–1201 (2002).
5) Myjak, M.J. and Delgado-Frias, J.G.: Pipelined multipliers for reconfigurable hardware,

Proc. International Parallel and Distributed Processing Symposium, pp.26–30 (2004).
6) NEC：信号処理 LSI（DSP/音声）データブック (1996).
7) Gustafsson, O. and Wanhammar, L.: Bit-level pipelinable general and fixed coefficient digit-serial/parallel multipliers based on shift-accumulation, *IEEE Conf. Proc. of Electr. Circuits & Syst.*, Vol.2, pp.493–496 (2002).
8) Synopsys: *Module Compiler Datasheet.* http://www.synopsys.com/products/datapath/module_comp_ds.html
9) Tanabe, J., Taniguchi, Y., Miyamori, T., Miyamoto, Y., Takeda, H., Tarui, M., Nakayama, H., Takeda, N., Maeda, K. and Matsui, M.: Visconti: Multi-VLIW image recognition processor based on configurable processor, *Proc. IEEE Custom Integrated Circuits Conference*, pp.185–188 (2003).
10) Togawa, N., Tachikake, K., Miyaoka, Y., Yanagisawa, M. and Ohtsuki, T.: A hardware/software partitioning algorithm for processor cores with packed SIMD-type instructions, *IEICE Trans. Fundamentals*, Vol.E86-A, No.12, pp.3218–3224 (2003).
11) Togawa, N., Tachikake, K., Miyaoka, Y., Yanagisawa, M. and Ohtsuki, T.: Instruction set and functional unit synthesis for SIMD processor cores, *Proc. ASP-DAC*, pp.743–750 (2004).

(平成 17 年 10 月 21 日受付)
(平成 18 年 4 月 4 日採録)

栗原　輝

2004 年早稲田大学理工学部電子・情報通信学科卒業．現在，同大学大学院修士課程在学．VLSI 設計，特にマイクロプロセッサのハードウェア/ソフトウェア協調設計に関する研究に従事．

宮岡祐一郎（正会員）

2000 年早稲田大学理工学部電子・情報通信学科卒業．2002 年同大学大学院修士課程修了．2005 年同博士後期課程修了．博士（工学）．現在，（株）東芝．VLSI 設計，特にアプリケーションに特化したプロセッサの合成に関する研究に従事．IEEE，電子情報通信学会各会員．

戸川　望（正会員）

1992 年早稲田大学理工学部電子通信学科卒業．1994 年同大学大学院修士課程修了．1997 年同博士後期課程修了．博士（工学）．早稲田大学理工学総合研究センター講師，北九州市立大学国際環境工学部情報メディア工学科助教授を経て，現在，早稲田大学理工学部コンピュータ・ネットワーク工学科助教授．VLSI 設計，計算幾何学，グラフ理論等の研究に従事．1996 年第 9 回安藤博記念学術奨励賞受賞．1997 年度（第 21 回）丹羽記念賞受賞．IEEE，電子情報通信学会各会員．

柳澤　政生（正会員）

1981 年早稲田大学理工学部電子通信学科卒業．1983 年同大学大学院博士前期課程修了．1986 年同後期課程修了．工学博士．現在，早稲田大学理工学部電子・情報通信学科教授．電子回路の設計自動化，ノイズ解析，計算幾何学，グラフ理論等の研究に従事．1987 年度丹羽記念賞受賞．1990 年安藤博学術奨励賞受賞．IEEE，ACM，電子情報通信学会，プリント回路学会，日本 OR 学会各会員．

大附　辰夫（正会員）

1963 年早稲田大学理工学部電気通信学科卒業．1965 年同大学大学院修士課程修了．同年日本電気（株）入社．1980 年同退社．現在，早稲田大学理工学部電子・情報通信学科教授．工学博士．システム LSI およびこれに関連した基礎研究に従事．1969 年度電子情報通信学会論文賞受賞．1994 年度第 32 回電子情報通信学会業績賞受賞．IEEE CAS Society より Guillmin-Cauer Prize Award（1974 年），Meritorious Service Award（1995 年），Golden Jubilee Medal（2000 年）受賞．2000 年 IEEE より 3rd Millennium Medal 受賞．共著『VLSI の設計 I』（岩波書店），編共著『Layout Design and Verification』（North-Holland）．IEEE フェロー，電子情報通信学会フェロー，電気学会，プリント回路学会各会員．