情報処理学会論文誌 Vol. 49 No. 10 3462–3481 (Oct. 2008)

SIMD 型プロセッサコアの面積/遅延見積り

山崎 大輔[†1,*1] 小原 俊逸[†1] 戸川 望[†1]
柳澤 政生[†1] 大附 辰夫[†1]

ASIP（Application Specific Instruction Processor）の自動合成は，対象とするアプリケーションに最適な構成を決定し，プロセッサのハードウェア部分とソフトウェア部分を同時に設計する．最適な構成の探索において，ある時点での構成に対して逐一論理合成を行い最適な構成の判定を行うと探索に多大な時間を要してしまうため，探索の評価指標として面積/遅延の見積り値を用い，論理合成することなく高速な探索を行う必要がある．また，アーキテクチャ探索に使用する見積り値と論理合成値との誤差が大きいと解の探索において適切な解が得られない可能性があるため精度の高い見積りを行うことが重要となる．本稿では，SIMD 演算ユニットおよびアドレッシングユニットの構成の変化に対応した SIMD 型プロセッサコアの面積/遅延時間見積り式を提案する．見積り式はプロセッサコアと付随するハードウェアユニットを部分機能ごとに分けてパラメータ化することによって導出し，これを用いることで論理合成することなく所望のアーキテクチャの面積・遅延値を導出することが可能となる．見積り式により導出されたプロセッサコアの面積値と論理合成値の相対誤差は平均 2.25%，遅延時間の誤差は平均で 0.54 ns となった．

Area/Delay Estimation for SIMD Processor Cores

Daisuke Yamazaki,[†1,*1] Shunitsu Kohara,[†1]
Nozomu Togawa,[†1] Masao Yanagisawa[†1]
and Tatsuo Ohtsuki[†1]

In synthesis of ASIP (Application Specific Instruction Processor), we optimize processor architecture for a target application, and design a hardware part and a software part at the same time. In order to obtain an optimal processor architecture in a short time, we require a fast area/delay estimation without doing logic synthesis in an architecture exploration phase. It is important to estimate them accurately because a large range of errors may lead an inadequate solution. This paper proposes area/delay estimation for SIMD processor cores with configurable SIMD functional units and adressing units. Estimation equation is obtained by partitioning the processor core and hardware units into several functional parts and parameterizing them, and can obtain an estimation value for an architecture. We show the effectiveness of estimation equation by verifying the area/delay values obtained from the estimation equation and the logic synthesis value of processor cores. Relative error of them is 2.25% on the average. Error of delays is 0.54 ns on the average.

1. はじめに

アプリケーションプロセッサのような特定の用途に対して高い演算能力を持ったコアプロセッサの設計は，高性能が要求され実行時間や面積制約の厳しい組み込みシステムにおいて有効な手法であると考えられている．一方で，回路規模の増大による設計期間の長期化が問題となり，大規模な回路をフルカスタムで設計することは，製品の市場滞在時間の短い現代の市場においては難しい状況となってきている．これらの要求を満足するためには，ターゲットとするアプリケーションプログラムから自動で最適なプロセッサを合成することが有効であると考えられる．HW/SW 協調合成手法をプロセッサ最適化に応用した手法が提案されている[4),9)]．これらの手法では，対象とするアプリケーションに応じてプロセッサのハードウェア（HW）部分とソフトウェア（SW）部分を同時に設計することで設計期間を短縮し，両者のトレードオフを考慮して最適化を図る設計手法である．人手で解決してきたプロセッサの HW 部分と SW 部分の最適化を定量的な手法に基づいた自動設計手法で実現することにより，短い設計期間で最適なプロセッサ構成を得ることができる．

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[9),10)]．SIMD 型プロセッサコアとは，SIMD 命令を命令セットとして持ち，SIMD 命令を実行する SIMD 演算ユニットを持つプロセッサコアである．SPADES は，命令形式に VLIW 命令を用い，アプリケーションプログラムとアプリケーション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自動で合成する．システムの核となるプロセスの 1 つに HW/SW 分割がある．HW/SW 分割は，面積最大のプロセッサコア構成から，演算ユニットやレジスタ等の HW ユニットを徐々に削減していき，HW で実現されている部分を SW で代替するこ

†1 早稲田大学大学院基幹理工学研究科情報理工学専攻
Department of Computer Science and Engineering, Waseda University
*1 現在，ソニー株式会社
Presently with SONY Corporation

とで時間制約を満たす間で最小のプロセッサ構成を出力する．このシステムの HW/SW 分割系に組み込まれる要素技術として面積/遅延見積りがある．HW/SW 分割の際，評価値としてアプリケーションの実行時間とプロセッサの HW コストの見積り値が必要になる．見積り値はプロセッサコア構成を決定するパイプライン段数やレジスタ数，命令セット等のパラメータを基にして得ることが可能となる．HW/SW 分割系で用いるプロセッサコア構成パラメータを基に見積もられた面積/遅延値と，最終的に得られた HW 記述を論理合成して得られる論理合成値との誤差が大きいと，HW/SW 分割で適切な解が得られない可能性がある．そのため，見積り値に必要な条件は以下の 2 点である．

(i) アプリケーションに最適なプロセッサ構成を探索するために広範囲なアーキテクチャの見積り値を得ることが可能．
(ii) 面積/遅延見積り値の大小関係と論理合成値の大小関係が一致する．

(i) について，マルチメディアアプリケーション処理に有効な SIMD 演算ユニットおよびアドレッシングユニットの見積りを可能とし，それぞれの内部構成の変化についてもメモリバンク数等のパラメータ化を行い，入力されたアプリケーションに対して必要となる命令セットに対応した見積りを行うことで探索範囲の拡張を可能とする．(ii) について，プロセッサコアを部分機能ごとに分け，それぞれに対してパラメータ化を行って見積もり，精度を上げることで対応する．

プロセッサの面積を見積もる手法に関する報告としては文献 2)，4)，6)，8) がある．文献 2)，4)，6) の手法は SIMD 演算ユニットおよびアドレッシングユニットの見積りは考慮されていない．SIMD 演算ユニット等はプロセッサコアに対して占める面積の割合が大きく，並列度 1，パイプライン段数 5 の RISC プロセッサに付加した場合に SIMD 乗算ユニットは全体の約 38%，アドレッシングユニット（6 種類の間接アドレッシングモード，アドレスレジスタ 4 本）は約 30%を占める．このため，これらの見積りは無視できない．そして，単一の構成ではなく，アプリケーションにとって必要十分な内部構成をとることが望ましく，そのための見積りが求められる．文献 8) ではアドレッシングユニットの見積りに関して言及されているが，内部のレジスタ数の変化のみに対応した見積りとなっており，間接アドレッシングモードやメモリバンク数の変化に対しての対応がされていないため，ユニット内の柔軟性に欠ける．遅延時間を見積もる手法に関して，文献 5)，7)，8) では，SIMD 演算ユニットおよびアドレッシングユニットの遅延見積りには対応しておらず，命令の変更にともなう単一リソース内の構成変化に応じた見積りに対応するものではない．文献 5) においては各ハードウェアモジュールの持つパラメータはそれぞれが独立しており，本稿のようなパラメータの組合せをとることで面積値の見積りは行っていない．

また，アプリケーションに特化した構成をアーキテクチャ候補の中から探索する場合，候補の数が少ない場合にはデータベース化を行い検索する手段をとることが有効であると考えられる．しかしながら，プロセッサコアアーキテクチャにはパラメータが多く，すべてのパラメータの組合せに即したアーキテクチャ候補を用意した場合，10^{20} 通り以上の解候補が考えられ，見積り値も同数の値をデータベースに保持しておく必要がある．メモリ空間の観点から考えてこれほどの値を保持することは困難であると考えられるため，メモリに負荷をかけることなく高速に見積り値を得ることが必要となる．

本稿では SPADES プロセッサコアを対象とした SIMD 演算ユニット，アドレッシングユニットの構成の変化に対応した SIMD 型プロセッサコアの面積/遅延時間見積り手法を提案する[⋆1]．提案手法ではプロセッサコアと付随する HW ユニットを部分機能ごとに分けてパラメータ化することでパイプライン段数や並列度といった構成の変化と，付随する HW ユニット構成の変化に対応した面積/遅延の見積り値が得られ，HW/SW 分割での解候補に対して逐一論理合成することなく最適解の判定を行うことが可能となる．

本稿は以下のように構成される．2 章では見積り対象となるプロセッサアーキテクチャを定義し，SIMD 演算ユニット，アドレッシングユニットについて説明する．3 章では面積見積り手法，4 章では遅延時間見積り手法を提案する．5 章では提案手法を SIMD 型プロセッサコアに適用し，得られる面積/遅延見積り値と論理合成値の精度・忠実度による検証実験を行った結果を報告する．

2. アーキテクチャモデル

本章では，見積りの対象となるプロセッサアーキテクチャモデルを定義する．**図 1** にプロセッサアーキテクチャの例としてパイプライン段数 5 段（IF，ID，EXE，MEM，WB），並列度 2 のモデルを示す．見積り対象のプロセッサコアはハーバードアーキテクチャをとり，外部メモリとして，1 つの命令メモリおよび複数のデータメモリを持つ．また，レジスタファイルに含まれる汎用レジスタ数，サブルーチンコール時のリターンアドレス退避用スタックレジスタ数，アドレッシングユニット内のアドレスレジスタ数も可変であり，入力するアプリケーションと実行時間制約値から最適化される．演算部は，ALU，シフタ，乗算器，SIMD 演算ユニット等を付加することが可能であり，演算器ごとにデータが出力されるた

⋆1 予備稿として文献 11) と文献 12) がある．

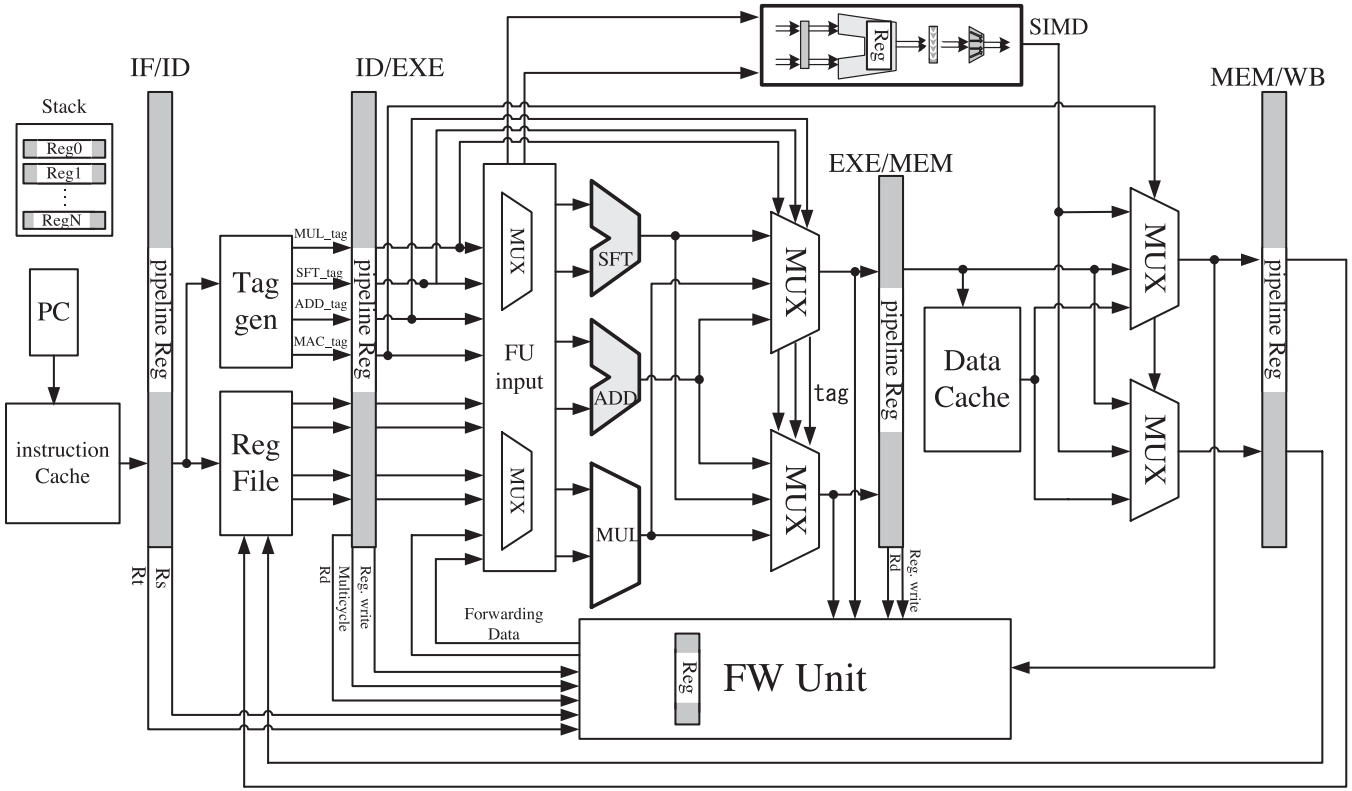


図 1　プロセッサアーキテクチャの例
Fig. 1　An example of processor architecture.

め，セレクタを用いて使用するデータのみを選択する．コンピュータグラフィックスにおける合成のためにアルファブレンドを実行する場合や，ビデオ・コーデックにおいて用いられる離散コサイン変換（DCT）等といったアプリケーションに対して SIMD 処理は高速化に適し，SPADES はこれに対応する．演算処理が複数段にまたがる演算器に対しては図 1 の SIMD 演算ユニットのように内部にレジスタを含んだ機能ユニットとすることで対応する．

対象アーキテクチャは命令レベル並列処理が可能であり，プロセッサコアに付加される演算器はすべてのスロットが任意で利用可能である．ここでスロットとはプロセッサコアで命令を発行する領域のことで，4 命令並列実行可能なプロセッサコアは 4 つのスロットがあるということとする．図 1 は，ID ステージで演算器ごとにタグを発行し，実行ステージでタグを基に演算器入力データおよび後段のパイプラインレジスタへ入力されるデータを選択している．演算の後，実行ステージの後段に付加される出力選択セレクタで加工されたデータをタグを参照することにより本来のスロットに戻す．以上の操作により，スロットとデータの整合性が保たれることとなり，少量の面積オーバヘッドで演算器を効率的に使用することが可能となる．

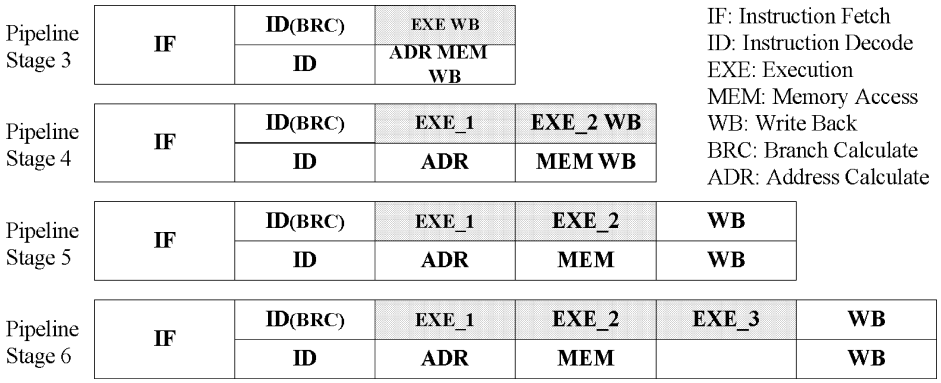


図 2　パイプライン構成モデル
Fig. 2　Pipeline architecture model.

以下ではパイプライン構成モデルを定義し，SIMD 演算ユニットおよびアドレッシングユニットを定義する．

2.1　パイプライン構成モデル

本節では，見積り対象となるプロセッサコアのパイプライン構成をパイプライン構成モデルとして定義する．最小パイプライン段数は 3 段とし，6 段までのモデルを**図 2** に示す．各構成モデルでは，第 1 ステージで命令のフェッチを行い，第 2 ステージで命令のデコードを行う．演算命令の場合，第 3 ステージ以降はパイプライン段数ごとに演算が複数段にまたがって行われる．分岐命令は第 2 ステージで分岐先のアドレス計算が行われる．ロード/ストア命令は第 3 ステージでアドレス計算が行われ，その後キャッシュアクセスおよびレジスタ書き込みを行う．

パイプライン段数 6 段以降の構成については，第 2 ステージまでと最後のステージでレジスタ書き込みを行う点はパイプライン段数 5 段の構成と同様で，パイプライン段数が増えるごとに演算実行ステージが 1 段ずつ増える．なお，今回提案する見積り手法は 32 ビットのワード長を対象としているが，64 ビットや 128 ビットのワード長の場合においても同様にして見積り値を得ることが可能である．

2.2　SIMD 演算ユニット

SIMD（Single Instruction Multiple Data）命令とは，1 個の b ビット演算ユニットを用いて n 個の b/n ビット演算を実現する命令であり，1 命令で複数のデータを処理することを

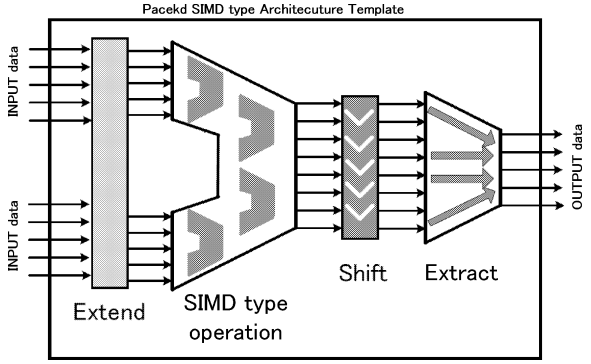


図 3　SIMD 演算ユニットのアーキテクチャテンプレート
Fig. 3　Architecture template of a SIMD functional unit.

意味する．n は SIMD 命令ごとに異なる値を設定でき梱包数と呼ぶ．一般に，画像処理アプリケーション等に代表されるマルチメディアアプリケーションで扱うデータの多くは，8 ビットまたは 16 ビットとサイズが小さいため，1 命令で複数のデータを処理可能な SIMD 命令はこうした処理に適している．SIMD 演算ユニットとは，SIMD 命令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 命令が割り当てられる．割り当てられた SIMD 命令を SIMD オプションと呼ぶ．SIMD 演算ユニットは演算実行ステージに配置され，内部にレジスタを挿入することでパイプライン化が可能となる．SIMD 演算ユニット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，SIMD 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 演算ユニットのアーキテクチャテンプレートを**図 3** に示す．SIMD 演算ユニットは，加算，乗算等の演算の種類の違いを除いて，精度拡張–算術演算–演算結果のシフト–精度縮小（飽和処理または丸め処理）の手順は一定である．したがって，命令形式に対応して，精度拡張演算部，算術演算部，シフト部，精度縮小演算部の 4 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．見積りを行う部分機能ユニットは，1 梱包演算には必ず対応するものとし，たとえば 1 梱包演算，2 梱包演算，4 梱包演算が可能な算術演算部に対して，1 梱包演算に加えて 2 梱包演算または 4 梱包演算またはその両方に対応した部分機能ユニットとなる．精度拡張演算部，固定シフト部，精度縮小演算部でも梱包数に対応した部分機能ユニットを用意することで所

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

望の梱包数演算が可能な SIMD 演算ユニットを構成できる．

以下では見積り対象となる命令セットについて説明し，SIMD 演算ユニットを構成する部分機能ユニットを定義する．

2.2.1　SIMD 命令セット

SIMD 命令は算術演算，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等で頻繁に実行される一連の演算を 1 つの命令として実行できる．これらの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，梱包数 2，符号付き，上位ビット拡張，左 10 ビットシフトの SIMD 加算命令は ADD_2h_sl10 と表される．SIMD 命令を**表 1** に示す．

2.2.2　精度拡張部（Extend）

精度拡張部は低精度のデータを，他のデータの精度にあわせて梱包しなおす際に使用される．入力は梱包数，符号付きの有無，オペランドであり，オペランドを精度拡張した結果を出力する．精度拡張部で行う処理は梱包された符号なしデータに対するゼロ拡張と，符号付きデータに対する符号拡張が存在する．これらの操作は精度拡張後の上位の空きビットに 0 を埋めるゼロ拡張に対し，符号拡張は上位ビットに符号ビットを繰り返しコピーして埋める処理を行う．拡張前の左半分を上位（high）データ，右半分を下位（low）データと呼び，命令の処理対象が上位/下位であるかを区別するときに high，low と呼ぶ．精度拡張の例を**図 4** に示す．

2.2.3　算術演算部（SIMD type operation）

SIMD 命令セットの算術演算命令には，加算，減算，乗算，乗加算が存在する．各算術演算部には，梱包数，符号付きの有無，オペランドを入力し，各オペランドに対して指定された数の梱包を行い，梱包されたデータごとに演算を行って結果を出力する．演算結果は固定小数点数が考慮され，シフトを行い適切な位置に小数点を移動してから精度の拡張/縮小が行われる．

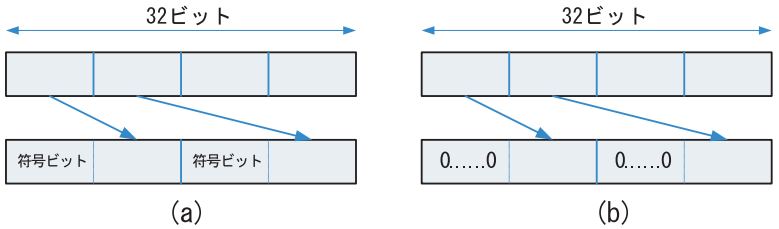


図 4　精度拡張．(a) 上位符号拡張，(b) 上位ゼロ拡張
Fig. 4　Extend (a) sign extended (b) zero extended.

2.2.4　シフト部（Shift）

シフト部は梱包数，符号付きの有無，シフト方向，シフト量を入力し，梱包されたデータごとにシフトされた結果を出力する．シフト方向は左シフト，右シフトの 2 種類が存在し，命令そのものでシフト量を静的に指定する．シフト量は，1 ビットから，対象となるデータの精度 n ビットまでの間で指定される．

2.2.5　精度縮小部（Extract）

精度縮小ユニットは入力として梱包数，符号付きの有無，シフト結果，丸めもしくは飽和演算を指定し，データの精度を縮小した結果を出力する．以下では丸め命令および飽和命令について説明する．

丸め命令（wrap around）　あるデータ型のデータの精度を縮小し，低精度のデータに丸める際に使用される．下位 $n/2$ ビットが保存され，上位 $n/2$ ビットは無視される．たとえば，0000000110000001 で表される 16 ビットの数値を 8 ビットに丸めた場合，結果は下位 8 ビットをとり 10000001 となる．n ビットのデータ型は $n/2$ ビットのデータ型に縮小されるため，1 つのレジスタに 2 倍の個数のデータを梱包することができるようになる．

飽和命令（saturation）　丸め命令と同様に，あるデータ型のデータの精度を縮小する際に使用される．たとえば，16 ビット符号付き整数型である 270 を 8 ビット符号付き整数型に縮小した場合，8 ビット符号付き整数が表せる範囲 $[-128, +127]$ を超えるので最も近い $+127$ に丸められる．b/n ビットのデータ型は $b/2n$ ビットのデータに縮小されるため，基本ビット長 b ビットの中に $2n$ 個のデータを梱包することができるようになる．

2.3　アドレッシングユニット

マルチメディアアプリケーションを処理するプロセッサでは，大量のデータに柔軟に効率良くアクセスすることが求められる．アドレッシングユニットはアドレス計算用のレジスタ

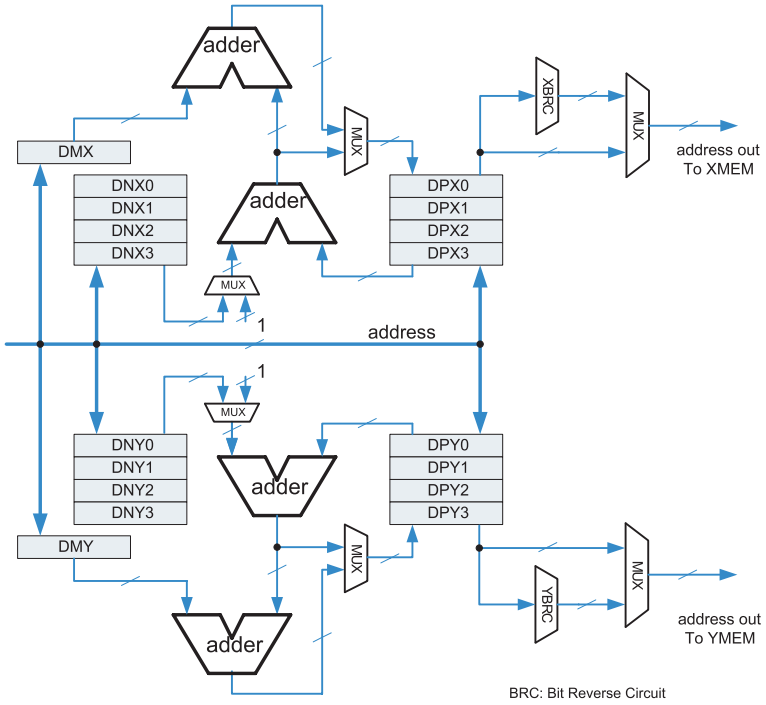


図 5　アドレッシングユニット
Fig. 5　Addressing unit.

および演算機能を持ち，豊富な間接アドレッシングモードを持つため，アドレス計算に通常の演算器を用いる必要がなく，高速かつ柔軟なデータメモリアクセスを実現できる．対象としているアドレッシングモードは間接アドレッシングである．**図 5** にアドレッシングユニットのブロック図を示す．

アドレッシングユニットの各レジスタの機能ブロックについて説明する．

(a) アドレスレジスタ（DPX，DPY）
間接アドレッシングのためのレジスタ．X メモリのアドレスを DPXn（n はレジスタ番号），Y メモリのアドレスを DPYn で指定する．ここで X メモリ，Y メモリはメモリバンクを意味する．

(b) インデクスレジスタ（DNX，DNY）
アドレスレジスタ値を修飾するためのレジスタ．メモリアクセス後，もしくは後段のパイプラインレジスタに入力後にインデクスレジスタ DNXn または DNYn の値で DPXn または DPYn（添え字 n は同一）を修飾する．

(c) モジュロレジスタ（DMX，DMY）
アドレスレジスタ値をリングカウント動作させながら修飾するときにリングカウントの範囲を指定するためのレジスタ．DPXn の範囲を DMX で，DPYn の範囲を DMY で指定する．

(d) アドレス加算器（adder）
アドレスレジスタ値を修飾するための加算器．X メモリ，Y メモリでそれぞれ 2 つずつ持ち，一方が DPXn+DNXn または DPYn+DNYn を行い，もう一方がモジュロ演算を行う．

(e) ビットリバース回路（XBRC，YBRC：X, Y Bit Reverse Circuit）
アドレスレジスタ値の上位と下位を反転したアドレスを出力する回路．最上位ビットの値が最下位ビットに，最下位ビットの値が最上位ビットになるよう反転する．

対象とするアドレッシングユニットで実行可能な間接アドレッシングモードについて説明する．

no change　アドレスレジスタ値でメモリアクセスし，アドレスレジスタ値は保存される．

post increment　アドレスレジスタ値でメモリアクセスし，アドレスレジスタ値をインクリメントする．

post decrement　アドレスレジスタ値でメモリアクセスし，アドレスレジスタ値をデクリメントする．

index add　アドレスレジスタ値でメモリアクセスし，アドレスレジスタ値にインデクスレジスタの値を加算する．

modulo add（モジュロ・インデクス加算）　アドレスレジスタ値でメモリアクセスし，アドレスレジスタ，インデクスレジスタおよびモジュロレジスタを用いてリングカウンタを実現する加算処理である．初期アドレスにアドレスレジスタ，加算値にインデクスレジスタ値を使用し，上限値はモジュロレジスタで与える．

bit reverse　アドレスレジスタの上位–下位ビットを反転した値でメモリアクセスし，アドレスレジスタにインデクスレジスタの値を加算する．加算されるアドレスレジスタ値はビット反転されない．

3. 面積見積り

本章では，図 1 で定義したアーキテクチャに付加される SIMD 演算ユニットおよびアドレッシングユニットの面積の見積り式を提案する．

プロセッサコアの面積 c は，**図 6** のように機能分割を行い，レジスタファイルの面積 c_{reg}，フォワーディングユニットの面積 c_{fw}，機能ユニットの面積 c_{FU}，HW ループユニットの面積 c_{loop}，アドレッシングユニットの面積 c_{ad} およびそれ以外のプロセッサとして動作するための基本的な機能を持つプロセッサカーネルの面積 c_k を加えることで算出され，次式で与えられる．

$$c = c_{reg} + c_{fw} + c_{FU} + c_{loop} + c_{ad} + c_k \tag{1}$$

このように分けて見積もることで以下の特徴を持つ．

(a) 機能分割した各部分ごとに必要なパラメータの数が少ない．
(b) 部分ごとに少ない数でパラメータ化を行うため，見積り式の導出が行いやすく精度の高い見積り値が得られる．

ここで，ALU やシフタといった基本的な演算を行うものは基本機能ユニット，SIMD 演算を行うものは SIMD 演算ユニットから構成され，これらの面積の和を機能ユニットの面積とする．基本機能ユニット u_b の面積は，u_b の集合を機能ユニットライブラリ U_b に入れておき，ライブラリから使用する機能ユニットの値を用いることで見積り値を得ることができる．基本機能ユニットの面積を $f_{base}(u_b)$，SIMD 演算ユニット u_s の面積を $f_s(u_s)$，プロセッサコア内で使用する u_s の集合を U_s として機能ユニットの面積 c_{FU} を以下に示す．

$$c_{FU} = \sum_{u_b \epsilon U_b} f_{base}(u_b) + \sum_{u_s \epsilon U_s} f_s(u_s) \tag{2}$$

式 (1)，(2) において，プロセッサカーネル，レジスタファイル，フォワーディングユニット，基本機能ユニット，HW ループユニットの見積りは文献 11)，12) を用いている．このため，以下では SIMD 演算ユニットおよびアドレッシングユニットの実装を行い面積の見積りを行う．これらの HW ユニットに対してパラメータの異なる様々な構成を論理合成し，合成結果とパラメータの関係の分析を行い，得られたアーキテクチャの性質を基にして見積り式を提案する．提案する見積り式を用いることで特定の HW ユニット内の様々な構成の面積値を逐一論理合成することなく高速に見積もることが可能となる．なお，実装には，HW 記述言語 VHDL を使用し，Synopsys 社の Design Compiler を用いて論理合成を行った．解析の際，セルライブラリには STARC[⋆1]（CMOS 90 nm）の設計ルールを用い，面積の単位は μm^2，遅延時間の単位は ns である．

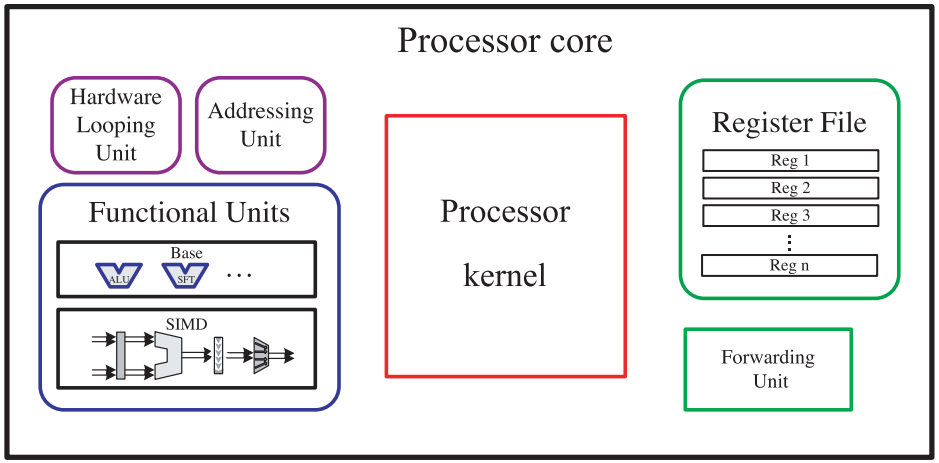


図 6　見積りにおけるプロセッサコアの機能分割
Fig. 6　Functional partition of processor core for estimation.

3.1 SIMD 演算ユニット面積見積り

SIMD 演算ユニット u_s の面積 $f_s(u_s)$ は図 3 のアーキテクチャテンプレートで示される部分機能ユニット（精度拡張部，算術演算部，シフト部，精度縮小部）ごとに見積もり，これらの和をとることで SIMD 演算ユニット全体の見積り値を導出する．精度拡張部の面積を f_{extd}，算術演算部の面積を f_u，シフト部の面積を f_{sf}，精度縮小部の面積を f_{extr} とすると f_s は以下の式となる．

$$f_s(u_s) = f_{extd} + f_u + f_{sf} + f_{extr} \tag{3}$$

3.1.1 精度拡張部見積り

精度拡張命令における選択可能な SIMD オプションは梱包数，符号の有無，上位/下位拡張である．各 SIMD オプションの要素数が増加するほどそのオプションに対応した HW を付加し，選択する構成となるため，面積は要素数に応じて増加する．ここでの要素数とは，各 SIMD オプションにおいて演算器が対応する部分機能の数である．たとえば，MUL_4_ur2s,

⋆1 STARC 90 nm ライブラリは東京大学大規模集積システム設計教育研究センターを通し，株式会社半導体理工学研究センター（STARC）と株式会社先端 SoC 基盤技術開発（ASPLA）の協力で開発されたものである．

表 2　精度拡張部の各パラメータと面積
Table 2　Relation between a parameters and area for Extend.

e_p	e_g	e_l	合計	面積 [μm^2]
1	1	1	3	361
1	1	2	4	431
1	2	1	4	422
2	1	1	4	410
1	2	2	5	479
2	1	2	5	524
2	2	1	5	479
2	2	2	6	624

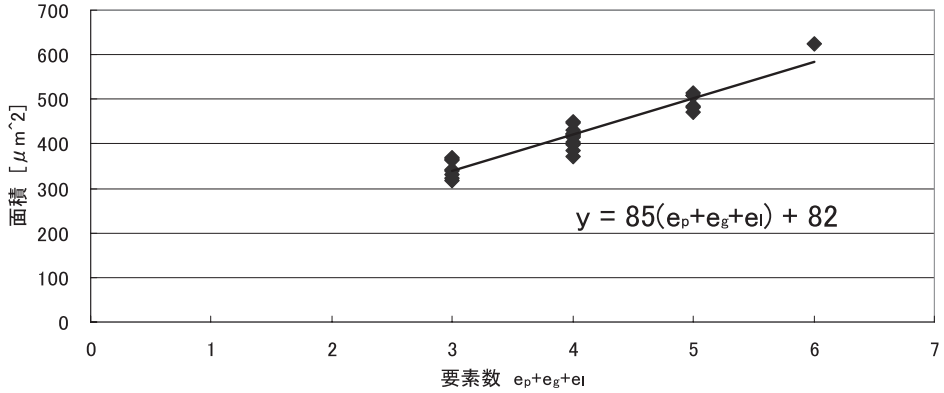


図 7　精度拡張部の要素数と面積の関係
Fig. 7　Relation between a number of elements and area for Extend.

ADD_4_sr2s，ADD_4h_u の 3 つの命令集合が与えられたとき，SIMD 演算ユニット（精度拡張部）が対応する部分機能は 4 梱包，符号付き（sign），符号なし（unsign），上位拡張であるため要素数は 4 となる．ここで，精度拡張において 1 梱包は処理を行う必要がないので要素数には含まない．SIMD オプションの要素数と精度拡張部の面積の関係の表を**表 2**，グラフを**図 7** に示す．図 7 の結果から精度拡張部は以下のような性質を持つことが分かる．

性質 1　精度拡張部の面積は要素数の増加に対して線形に増加する．

また，表 2 から，既存の方式と比較して，以下の点で差異がある．

(1)　SIMD 演算ユニットおよびアドレッシングユニットのパラメータ化による線形近似ではない．

(2)　パラメータの性質を読み取り，各パラメータ単独ではなく組合せによって見積りを行っている．

これは，1 つのパラメータではなく表 2 のようにパラメータの値の合計で決まるという点で既存方式と差異があるといえる．

以上から精度拡張部の面積 f_{extd} は梱包数の要素数 e_p，符号の要素数 e_g および上位/下位拡張の要素数 e_l をパラメータとして以下のように表される．

$$f_{extd} = 85(e_p + e_g + e_l) + 82 \tag{4}$$

$$e_p = \begin{cases} 2 & I_u \text{ に 2 梱包演算と} \\ & \text{4 梱包演算が含まれているとき} \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

$$e_g = \begin{cases} 2 & I_u \text{ に符号付き演算と} \\ & \text{符号なし演算が含まれているとき} \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

$$e_l = \begin{cases} 2 & I_u \text{ に上位拡張と} \\ & \text{下位拡張が含まれているとき} \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

ここで，I_u は SIMD 演算ユニット u_s で実行可能な命令集合を意味する．

3.1.2　算術演算部見積り

対象とする SIMD 命令における算術演算命令は加算，減算，乗算および乗加算が存在する．減算命令は加算器で実行を行うため，以下では加算器，乗算器，乗加算器の見積りに関して説明する．

算術演算部の選択可能な SIMD オプションは加算器が梱包数であり，乗算器と乗加算器が梱包数と符号の有無である．1 梱包で演算を実行する加算器のアーキテクチャモデルを**図 8** に示す．演算器の SIMD 化を行った場合，各梱包数ごとに HW を用意して出力を選択する構成だと特定の SIMD オプションを実行している間に使用されていない回路が多く HW 資源の無駄となる．また，保持しなければならないビット数が多いためにパイプライン化を行った際に用意しなければならないパイプラインレジスタが大きくなり，回路が大きくなる．図 8 で加算器は SIMD 加算を実行するために太線部の回路 p（packing control）を用

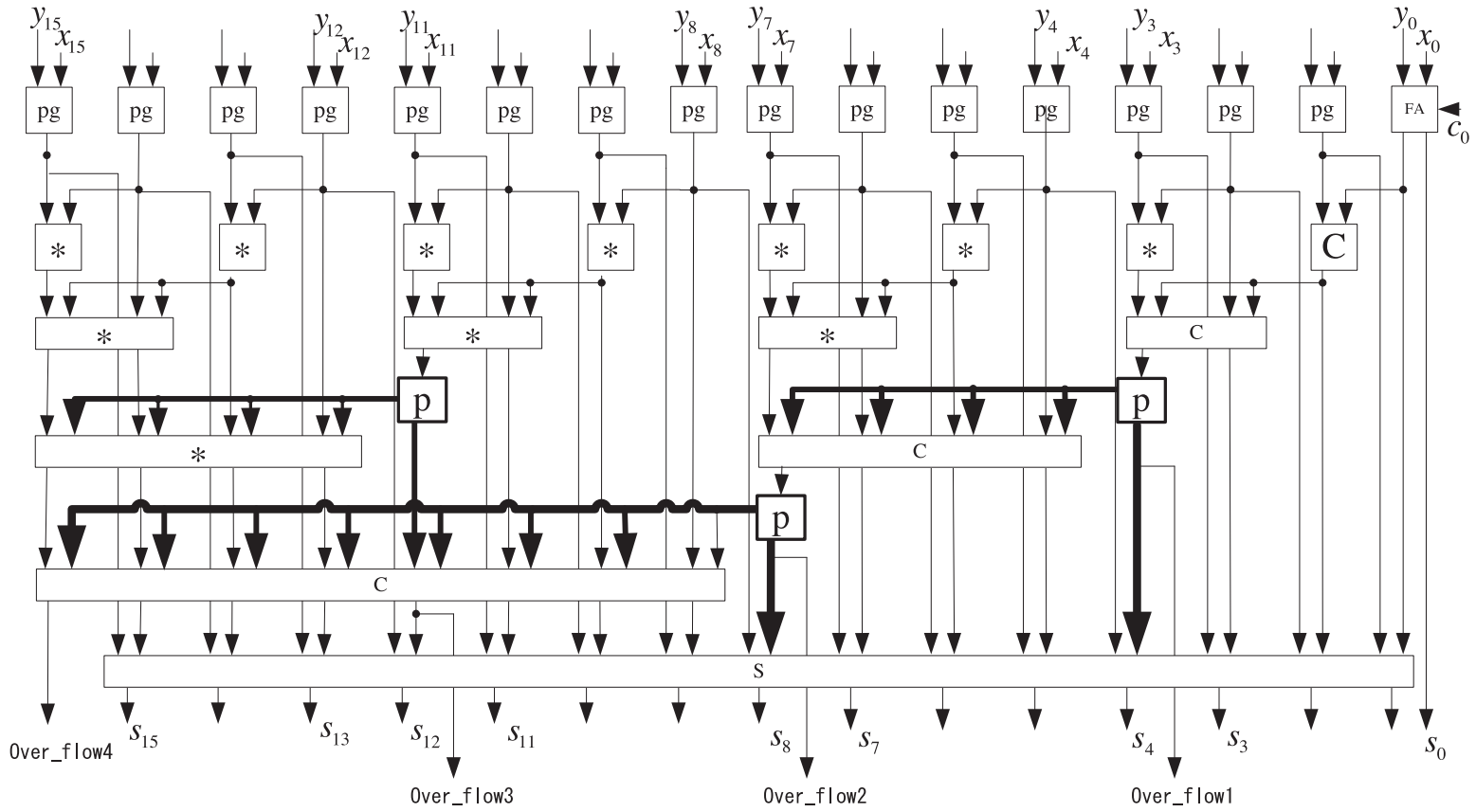


図 8　SIMD 加算器
Fig. 8　SIMD adder.

いて上位桁へのキャリー等の値を梱包数に応じて変化させている．たとえば，4 梱包加算の場合は図 8 の回路 p の出力を 0 にすることで太線部分の値がすべて 0 になり上位の桁への影響を与えない状態を作ることで 4 梱包の加算を可能にする．また，乗算器は Carry-save 型の乗算器を用い，部分積を求める際に被乗数を梱包数に応じて変化させることで SIMD 化に対応する．たとえば 2 梱包乗算を行う場合，部分積を求める際に乗数が 1 梱包目（下位桁）の値の場合は被乗数の上位半分を制御して値を 0 にする．乗数が 2 梱包目（上位桁）の値の場合は被乗数の下位半分を制御して値を 0 にすることで SIMD 乗算を実現する．乗加算器の場合は乗算器の構成に Half adder を加えた構成となる．これにより通常の加算器，乗算器および乗加算器から少量の面積増加で SIMD 化を実現する．**表 3**，**表 4** にシングルサイクル SIMD 加算器，SIMD 乗算器の面積/遅延値を示す．これらパイプライン化されていない構成では梱包数の変化に対して回路構成の変化が少なく，符号演算の有無に対してもプロセッサコア自体の面積が大きいために影響が少ない．ゆえに算術演算部の見積りはパイプライン化された構成が問題となる．

演算器をパイプライン化した場合，パイプラインレジスタの挿入位置ごとにパイプラインレジスタのビット幅が異なり，演算器全体の面積も異なる．しかしながら，パイプライン化はパイプラインレジスタの挿入以外に演算器の内部構成に変更はないので，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 f_i と，挿入されたパイプラインレジスタの面積値の和から見積もられる．演算器内部におけるパイプラインレジスタの挿入位置は文献 3) で定義されている最小単位モジュール間に挿入することを考える．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，この概念を導入することによりパイプラインレジスタを挿入する際に挿入範囲が膨大になることなく，プロセッサコアの動作周波数を決定するうえで適切な間隔の遅延時間で部分機能ユニットを分割することが可能となり，高速な見積りが実現できる．**図 9** は最小単位モジュールの間隔を入力側から数値で表したものである．ここで，パイプラインレジスタのビット幅 k_p の値は，位置情報を基にして図 9 の <1>...<61> にパイプラインレジスタを挿入したときに横切るビット線の数で決まる．

表 3　シングルサイクル SIMD 加算器の面積/遅延時間
Table 3　Area/Delay for single cycle SIMD adder.

pipeline stage	梱包数	面積 [μm^2]	遅延 [ns]
1	1, 2, 4	1,501	1.07
1	1, 2	1,394	0.88
1	1, 4	1,458	1.12

表 4　シングルサイクル SIMD 乗算器の面積/遅延時間
Table 4　Area/Delay for single cycle SIMD multiplier.

梱包数	sign	unsign	面積 [μm^2]	遅延 [ns]
1, 2, 4	○	○	26,662	7.48
1, 2, 4	○	×	26,694	7.38
1, 2, 4	×	○	26,601	7.48
1, 2	○	○	26,389	7.56
1, 2	○	×	26,496	7.65
1, 2	×	○	26,355	7.52
1, 4	○	○	26,408	7.64
1, 4	○	×	26,548	7.71
1, 4	×	○	26,349	7.61

加算器，乗算器，乗加算器に関して，それぞれを構成する最小単位モジュール間にパイプラインレジスタを挿入し，論理合成を行った．これらの面積増加分を導出し，パイプラインレジスタのビット幅 k_p と面積の関係を**図 10** に示す．図 10 の結果から算術演算部は以下のような性質を持つことが分かる．

性質 2　シングルサイクルの SIMD 演算器では，梱包数，符号の変化に対する影響がプロセッサコア全体から見て無視できるほどきわめて小さい．

パイプライン化の際，パイプラインレジスタのビット幅の増加にともないパイプラインレジスタの面積は線形に増える．

以上から算術演算部の面積 f_u 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 f_i，パイプラインレジスタのビット幅 k_p を用いて以下の式で見積もられる．

$$f_u = f_i + 14k_p - 78 \tag{8}$$

3.1.3　シフト部見積り

シフタは構成を決定する SIMD オプションとして梱包数，符号の有無，シフト方向が選択可能である．アプリケーションに最適な構成を考えた場合，システムから合成される SIMD 対応シフタは，特定のビット数のみのシフトが多く，バレルシフタのような任意ビットのシフトが可能なシフタの場合，通常のシフタに加え，梱包数や符号も任意ビットへの対応が要求されるため，不要な構成が多くなる．このため，シフト量を限定し，必要なビットのみシフト可能なシフタが望ましい．したがって，見積りは特定のシフト量を持つシフタの見積りを行う．

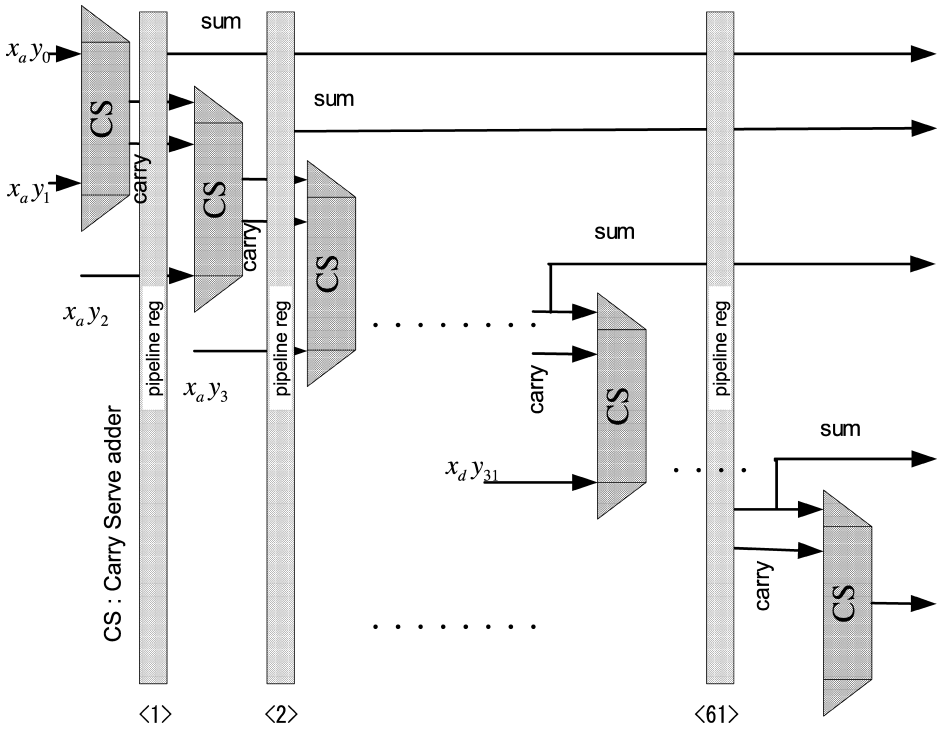


図 9　パイプラインレジスタ挿入位置
Fig. 9　Pipeline register position.

シフタには構成を決定する SIMD オプションが梱包数，符号の有無，シフト方向の 3 種類存在するため，たとえば 32 ビット演算を行った場合，シフト量 1〜31 ビットのそれぞれに対して梱包数，符号，シフト方向が選択でき，さらにこうしたシフトパターンの組合せでシフタが構成される．面積はこれら SIMD オプションの組合せの総数に依存する．この組合せ総数をシフトパターン数と呼ぶ．例として，MUL_2_ur1s，MUL_4_ul2s，MUL_1_sl30s の 3 つの命令集合が与えられたとき，SIMD 演算ユニット（シフト部）が対応する部分機能は『2 梱包–符号なし–1 ビット右シフト』，『4 梱包–符号なし–2 ビット左シフト』および『1 梱包–符号あり–1 ビット左シフト』であるためシフトパターン数を h とすると，$h = 3$ となる．シフトパターン数とシフタの面積の関係を**図 11** に示す．図 11 よりシフタの面積

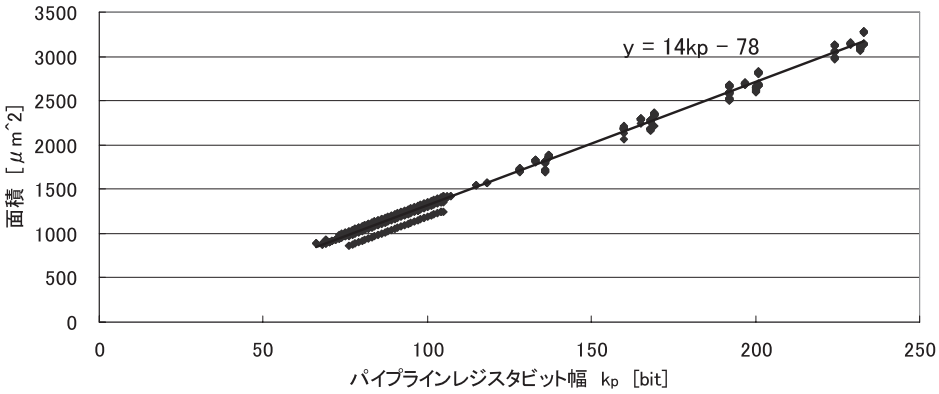


図 10　パイプラインレジスタビット幅と面積の関係
Fig. 10　Relation between pipeline register bit width and area.

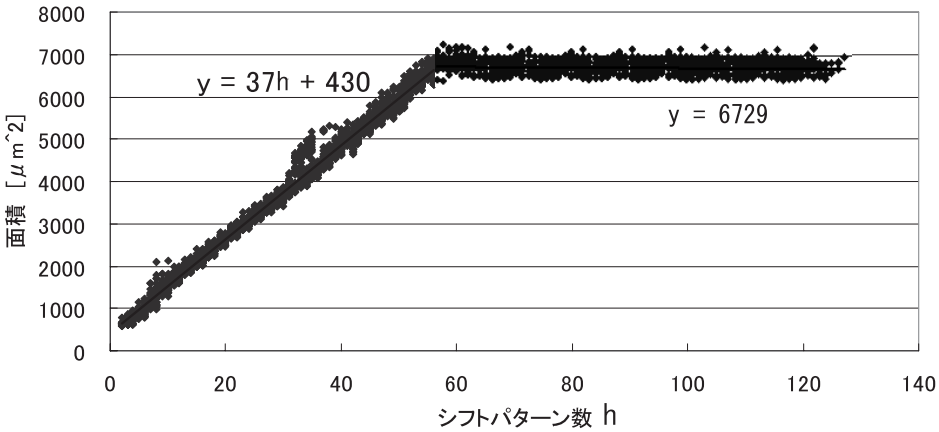


図 11　シフタのシフトパターン数と面積の関係
Fig. 11　Relation between a number of shift pattern and area.

はシフトパターン数が 57 付近までほぼ線形に増加しその後はほぼ一定の値をとっている．このことから，シフト部は以下のような性質を持つことが分かる．

性質 3　シフト部はシフトパターン数の増加にともない面積は線形に増え，ある点を境に一定の値をとる．

シフトパターン数 h を用いて，SIMD 演算ユニットのシフト部 $f_{sf}(h)$ の面積見積り式は以下のようになる．

$$f_{sf}(h) = 37h + 430 \quad (h < 57)$$
$$f_{sf}(h) = 6729 \quad (h \geqq 57) \tag{9}$$

3.1.4　精度縮小部見積り

精度縮小命令における選択可能な SIMD オプションは梱包数，符号の有無，丸め，飽和処理であり，これらの SIMD オプションに応じて面積/遅延時間が変化する．各梱包数，符号の有無における精度縮小部の面積/遅延時間を**表 5** に示す．ここで，演算器は梱包数 1 および丸め処理には必ず対応するものとしている．表 5 では，飽和処理に対応しているものが対応していないものに比べて面積が大きい．これは飽和処理は実行前にオペランドの飽和を調べるための HW が必要となるため，丸め操作に比べて SIMD オプション選択時の面積増加分が大きいことにつながっている．このことから精度縮小部は以下のような性質を持つことが分かる．

性質 4　精度縮小部は飽和処理を行う場合が，行わない場合に比べて他のパラメータに関係なく面積は必ず大きい．

表 5　精度縮小部の面積/遅延時間
Table 5　Area/Delay for Extract.

梱包数	sign	unsign	飽和処理	丸め	面積 [μm^2]	遅延 [ns]
1, 2, 4	○	○	○	○	629	0.97
1, 2, 4	○	×	○	○	448	0.66
1, 2, 4	×	○	○	○	461	0.62
1, 2	○	○	○	○	453	0.88
1, 2	○	×	○	○	325	0.72
1, 2	×	○	○	○	363	0.54
1, 4	○	○	○	○	483	0.80
1, 4	○	×	○	○	368	0.62
1, 4	×	○	○	○	387	0.51
1, 2, 4	○	○	×	○	280	0.46
1, 2, 4	○	×	×	○	246	0.47
1, 2, 4	×	○	×	○	241	0.43
1, 2	○	○	×	○	203	0.43
1, 2	○	×	×	○	196	0.48
1, 2	×	○	×	○	185	0.37
1, 4	○	○	×	○	236	0.47
1, 4	○	×	×	○	212	0.51
1, 4	×	○	×	○	207	0.40

さらに，精度縮小部は演算自体が処理量の小さいものであるため，全体的に面積が小さい．以上から近似を行い，飽和演算の有無で分類し，飽和演算を行う場合は表 5 の飽和演算対応の論理合成値の平均値（433 [μm^2]）を精度縮小部の面積見積り値とし，飽和演算をオプションにとらない場合は表 5 の飽和演算未対応の平均値（222 [μm^2]）を面積見積り値とする．精度縮小部 f_{extr} の面積見積り式を以下に示す．

$$f_{extr} = \begin{cases} 433 & I_u \text{ に飽和演算が含まれているとき} \\ 222 & \text{otherwise} \end{cases} \tag{10}$$

3.2　アドレッシングユニット面積見積り

アドレッシングユニット面積のパラメータは間接アドレッシングモード I_{ad} とメモリバンク数 $\mathbf{m}$，アドレスレジスタ数 k_a から決定される．アプリケーションにとって十分な間接アドレッシングモードやメモリバンク数およびアドレスレジスタ数を選択することで，必要十分な構成をとることが可能となるため，アドレッシングユニットの面積削減が可能となる．ここで，見積りには一般的な DSP に沿って，以下の (1)，(2)，(3) を前提とする．

(1)　間接アドレッシングモードは必ず 2 モード以上使用し，そのうち 1 つは no change である．
(2)　モードによらず加算器を必ず 1 つ以上持つ．
(3)　インデクスレジスタが付加される場合，レジスタ数はアドレスレジスタ数と等しい．

以上をふまえ，パラメータを変化させ，様々な構成をとるアドレッシングユニットを実装した．$k_a = 0$ のときの各モードにおけるアドレッシングユニットの面積/遅延時間を**表 6** に示す．また，アドレスレジスタ数 k_a とアドレッシングユニットの面積増加量の関係を**図 12** に示す．表 6 より，$\mathbf{m} = 1$ においてラベル 1（no change）からの面積増加分に着目すると増加分は大きく以下の 3 つに分けられる．

(i)　modulo add を持つもの（面積 4,232〜4,400：ラベル 6，10，11）．
(ii)　(i) を除いた中でインデクスレジスタを持つもの（面積 520〜593：ラベル 4，5，8，9）．
(iii)　インデクスレジスタを持たないもの（面積 207〜236：ラベル 2，3，7）．

これは，アドレッシングユニットを構成する機能ブロックのうち，(i) はすべてを搭載する，(ii) はモジュロ演算用の加算器とモジュロレジスタを持たない，(iii) はモジュロ演算

表 6　アドレッシングユニットの面積/遅延時間（アドレスレジスタ数 $k_a = 0$）
Table 6　Area/Delay of addressing units (a number of address registers $k_a = 0$).

ラベル	no change	post inc	post dec	idx add	bit reverse	modulo add	メモリバンク数 $\mathbf{m} = 1$ 面積 [μm^2]	1 からの面積増加分	遅延 [ns]	メモリバンク数 $\mathbf{m} = 2$ 面積 [μm^2]	1 からの面積増加分	遅延 [ns]
1	○	×	×	×	×	×	1,486	0	0.63	2,942	0	0.82
2	○	○	×	×	×	×	1,705	219	1.18	3,395	452	1.25
3	○	×	○	×	×	×	1,693	207	1.54	3,401	459	1.45
4	○	×	×	○	×	×	2,006	520	1.44	3,928	986	1.45
5	○	×	×	×	○	×	2,067	580	1.39	4,055	1,113	1.59
6	○	×	×	×	×	○	5,718	4,232	3.33	11,397	8,456	3.32
7	○	○	○	×	×	×	1,722	236	1.42	3,437	495	1.42
8	○	○	×	○	×	×	2,027	541	1.17	3,948	1,006	1.26
9	○	○	×	×	○	×	2,079	593	1.47	4,081	1,139	1.51
10	○	○	×	×	×	○	5,730	4,244	3.28	11,450	8,507	3.31
11	○	○	○	○	○	○	5,886	4,400	3.27	12,014	9,071	3.86

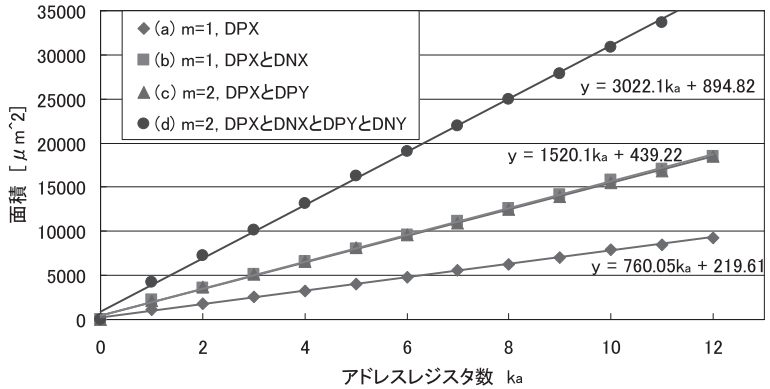


図 12　アドレスレジスタ数と面積増加量の関係
Fig. 12　Relation between a number of address registers and area increasing.

用の加算器，モジュロレジスタとインデクスレジスタを持たないことに起因する．一方で，$\mathbf{m} = 1$ と $\mathbf{m} = 2$ において，とりうる間接アドレッシングモードが等しいものどうしを比較すると $\mathbf{m} = 2$ は $\mathbf{m} = 1$ の 2 倍程度となっていることが分かる．これは，$\mathbf{m} = 1$ と $\mathbf{m} = 2$ ではとりうる間接アドレッシングモードが同じならば機能ユニットの数は 2 倍になるためである．次に，図 12 を見ると，3 種類の見積り式に分類される．レジスタの種類は DPX，DPY，DNX，DNY の 4 種類であり，その中から 1 つをとるもの（(a)：$760k_a + 220$），2 つをとるもの（(b)，(c)：$1520k_a + 439$），4 つすべてをとるもの（(d)：$3022k_a + 895$）となる．(a) は $\mathbf{m} = 1$ でインデクスレジスタを持たないものである，(b) は $\mathbf{m} = 1$ でインデクスレジスタを持つものである，(c) は $\mathbf{m} = 2$ でインデクスレジスタを持たないものである，(d) は $\mathbf{m} = 2$ でインデクスレジスタを持つものである．このことより，(a)（DPX のみ）を基準とすると，式 (b)，(c) は式 (a) の約 2 倍，式 (d) は式 (a) の約 4 倍となっていることが分かり，間接アドレッシングモード，メモリバンク数から使用する式が決定され，アドレスレジスタ数によりアドレッシングユニットのレジスタの面積を導出することができる．このことからアドレッシングユニットは以下のような 3 つの性質を持つことが分かる．

性質 5　アドレッシングユニットは以下の 3 つの性質を持つ．

(1)　間接アドレッシングモードに関して，アドレッシングユニットは modulo add を持つもの，modulo add を持たずインデクスレジスタを持つもの，インデクスレジスタを持たないものの 3 種類に分類される．

(2)　メモリバンク数に関して，間接アドレッシングモード，アドレスレジスタ数が同じならばメモリバンク数の増加にともない線形に面積は増える．

(3)　レジスタに関して，アドレッシングユニット内のレジスタの面積はアドレスレジスタの本数の増加にともない線形に増える．

また，アドレッシングユニットをプロセッサコアに付加したとき，$\mathbf{m} = 2$ 以上の場合，並

Step 1　メモリバンク数 $\mathbf{m}=1$，アドレスレジスタ数 $k_a=0$，no change のみの面積 a_{ini} から開始し，modulo add を使用するならば modulo add 用の機能ユニットの面積 a_{md} を加算し **Step 4** へ，使用しなければ **Step 2** へ進む．

Step 2　index add もしくは bit reverse を使用するならばインデックスレジスタ付加時の制御部分の面積 a_{id} を加算し **Step 4** へ，使用しないならばアドレスレジスタ付加時の制御部分 a_{dp} を加算し **Step 3** へ進む．

Step 3　現在の値にアドレスレジスタの面積 a_{dpreg} を加算し，$\mathbf{m}=1$ ならば終了，$\mathbf{m}=2$ ならば結果を 2 倍し，アドレッシングユニットを付加したときのプロセッサコアの増加分 c_{ad} を加算して終了．

Step 4　アドレスレジスタとインデックスレジスタの面積 $a_{dpreg}+a_{dnreg}$ を加算し，$\mathbf{m}=1$ ならば終了，$\mathbf{m}=2$ ならば現在の値を 2 倍し，アドレッシングユニットを付加したときのプロセッサコアの増加分 c_{ad} を加算して終了．

図 13　アドレッシングユニット面積見積りフロー
Fig. 13　Area estimation flow of addressing units.

列度にかかわらずレジスタファイルへ書き込むポート数が増加する．このため，メモリアクセスおよびレジスタ書き込みに必要な制御線と増加した書き込みポートに格納するデータを保持するパイプラインレジスタが必要となり，面積が増加する．これには，アドレッシングユニット付加時の面積増加分をとることで対応する．増加分 c_{ad} はパイプライン段数 s に依存し，以下で表される．

$$c_{ad}=1022s-639 \tag{11}$$

以上より，アドレッシングユニットの面積見積りフローを**図 13** に示す．Step1，2 の no change からの増加分の加算は増加分の中で最大のものを用いた．

4. 遅延時間見積り

プロセッサコアの遅延時間は，機能ユニット遅延時間，セレクタ遅延時間およびレジスタ書き込み時間を基に見積りを行う．セレクタの遅延時間は，出力セレクタと EXE1 ステージの演算器の直前に配置されるフォワーディングしたデータを選択するセレクタに分けられる．本章では，機能ユニット遅延時間の一部である SIMD 演算ユニットの遅延時間，アドレッシングユニットの遅延時間見積りを提案する．

一般にプロセッサコアでは演算部分の遅延時間が他の部分に対して大きいため，プロセッサコアの遅延時間は演算器を通るパスの遅延となる．このため，遅延時間見積り手法では演算ステージの遅延時間に焦点を置き見積りを行う．並列度 n の場合，各スロット i の演算ステージの遅延時間 T^i_{EXE} の最大値をとったものとなり，以下の式で表される．

$$T_c=\max\{T^1_{EXE},\ldots,T^n_{EXE}\} \tag{12}$$

T_{op1} を EXE1 ステージの機能ユニット遅延時間，T_{op2} を EXE2 ステージの機能ユニット遅延時間，T_{op3} を EXE3 ステージの機能ユニット遅延時間，T_{sel1} を EXE1 ステージのセレクタ遅延時間，T_{sel2} を EXE2 ステージのセレクタ遅延時間，T_{sel3} を EXE3 ステージのセレクタ遅延時間，T_{WB} をレジスタ書き込み遅延時間とすると，T^i_{EXE} は次のように見積もられる．

- パイプライン段数 3 段
  $T^i_{EXE}=T_{op}+T_{sel}+T_{WB}$
- パイプライン段数 4 段
  $T^i_{EXE}=\max\{T_{op1}+T_{sel1},\ T_{op2}+T_{sel2}+T_{WB}\}$
- パイプライン段数 5 段
  $T^i_{EXE}=\max\{T_{op1}+T_{sel1},\ T_{op2}+T_{sel2}\}$
- パイプライン段数 6 段
  $T^i_{EXE}=\max\{T_{op1}+T_{sel1},\ T_{op2}+T_{sel2},\ T_{op3}+T_{sel3}\}$

つまり，パイプライン段数ごとに分け，すべてのスロットの遅延時間の最大をとったものがプロセッサコアの遅延時間 T_c となる．たとえば，EXE1 ステージの遅延は機能ユニット遅延時間にセレクタ遅延時間を加えたものである．

提案する SIMD 演算ユニットとアドレッシングユニット以外の機能ユニット遅延時間，セレクタ遅延時間およびレジスタ書き込み時間の見積りは文献 11)，12) で提案された遅延時間見積り式を用いる．以下では SIMD 演算ユニット遅延見積りとアドレッシングユニット遅延見積りに関して説明する．

4.1 SIMD 演算ユニット遅延見積り

SIMD 演算ユニットの遅延時間は機能ユニット遅延時間として見積りを行う．SIMD 演算ユニットを構成する部分機能ユニットの中で，算術演算部以外の見積りはパラメータの変化に対して遅延時間の変化が小さいため一定の値を用い，ここでは算術演算部に着目する．

SIMD 演算ユニットがシングルサイクルで実行されるとき，遅延時間見積りは表 3，表 4 の結果から SIMD オプションの変化が遅延値に与える影響が小さいため基本機能ユニットと同様にしてユニットごとの遅延値をライブラリから導出する．

パイプライン化された SIMD 演算ユニットの場合，文献 3) で提案されている最小単位モジュールの概念を用いて，最小単位モジュール間にパイプラインレジスタを挿入し，挿入位置に対する遅延値の変化を解析することで見積りを行う．

スロット i に対して，5 段パイプラインプロセッサの場合，演算ステージは図 2 より EXE1，EXE2 の 2 ステージに分けられる．SIMD 演算ユニットがプロセッサコアに付加されてい

る場合，これにともない SIMD 演算ユニットも 2 ステージに分けられ，機能ユニット遅延時間はステージごとに T_{op1}，T_{op2} となる．このとき，パイプラインレジスタの挿入位置は部分機能ユニット間または算術演算部の内部となる．部分機能ユニット間にパイプラインレジスタを挿入した場合，T_{op1}，T_{op2} は EXE1，EXE2 ステージに配置される各部分機能ユニットの遅延時間の和として見積り値を得ることができる．ここでは算術演算部の内部にパイプラインレジスタを挿入した場合（**図 14**）を考える．図 14 で，T_{op1}，T_{op2} は精度拡張部の遅延時間 d_{1_1}，算術演算部の 1 段目の遅延時間 d_{1_2}，算術演算部の 2 段目の遅延時間 d_{2_1}，シフタの遅延時間 d_{2_2}，精度縮小部の遅延時間 d_{2_3} を用いて以下の式で表される．

$$T_{op1} = d_{1\_1} + d_{1\_2} \tag{13}$$

$$T_{op2} = d_{2\_1} + d_{2\_2} + d_{2\_3} \tag{14}$$

例として SIMD 乗算器を最小単位モジュールごとに分割し，間にパイプラインレジスタを挿入して 2 段にした結果を各 SIMD オプションごとに**図 15** に示す．図 15 の，挿入位置は図 9 のように最小単位モジュールの間隔を入力側から数値で表したもので，値が大きいほどパイプラインレジスタが部分機能ユニットの出力側に近くなり．図 15 では挿入位置が出力側へ行くほど入力側遅延が大きくなり，出力側遅延が小さくなる．また，図 15 では，パイプライン段数 1 段目，2 段目ともに挿入位置に対してほぼ線形となっており，パイプラ

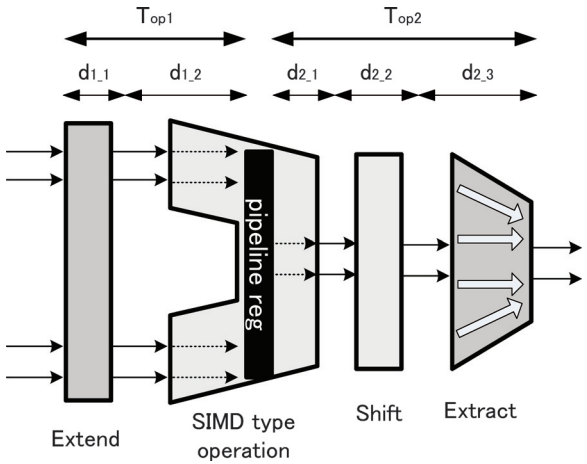


図 14　算術演算部のパイプライン化
Fig. 14　Pipelining of SIMD type operation.

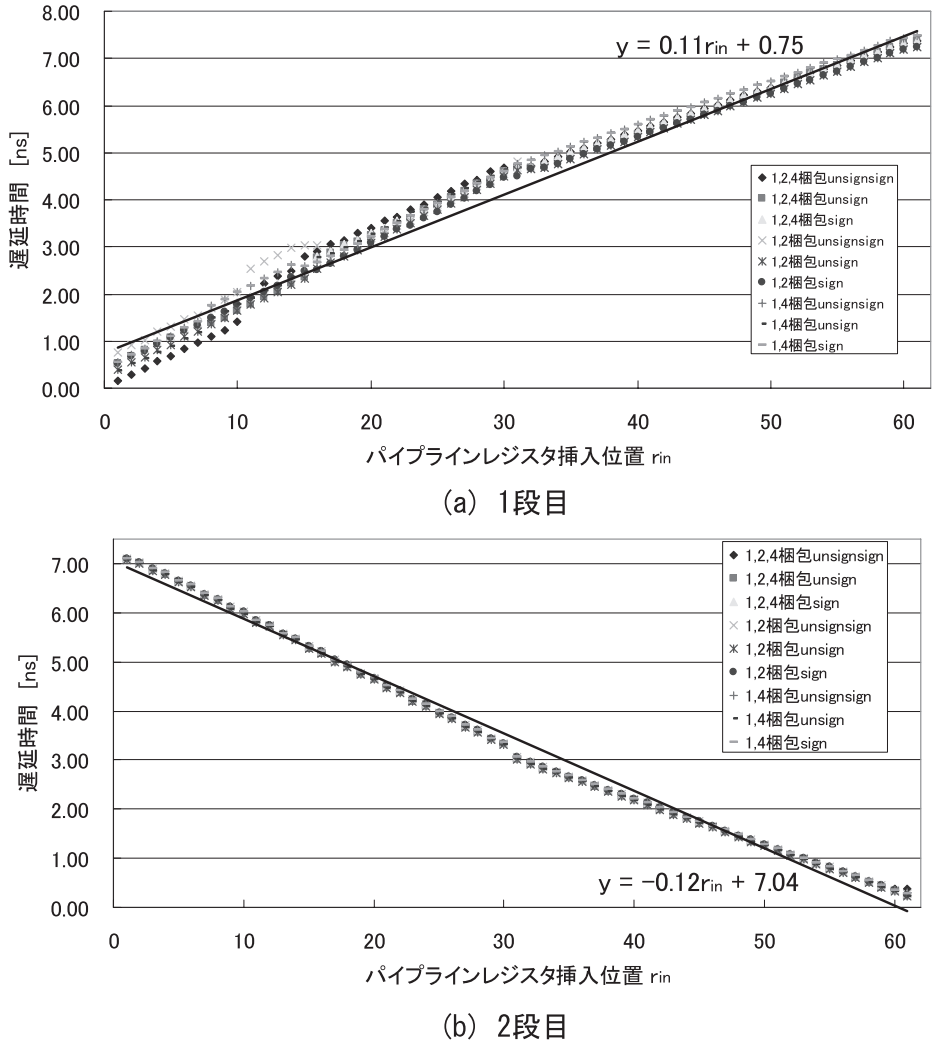


図 15　SIMD 乗算器のパイプラインレジスタ挿入位置と遅延時間の関係．(a) 1 段目の遅延，(b) 2 段目の出力側遅延
Fig. 15　Relation between a pipeline register position and delay for SIMD multiplier. (a) inside delay, (b) outside delay.

インレジスタの挿入位置を変更することで 0.1〜0.15 [ns] 間隔で遅延を操作することができる．このとき，SIMD 演算ユニットは以下のような性質を持つことが分かる．

性質 6　SIMD 演算ユニットの遅延時間はパイプラインレジスタの挿入位置の変化に対して線形に変化する．SIMD オプションが遅延時間に与える影響は微小である．

以上からパイプラインレジスタの挿入位置をパラメータとすることにより見積もることが可能となり，1 段目の遅延が $0.11r_{in} + 0.75$（r_{in}：レジスタ挿入位置），2 段目の遅延が $-0.12r_{in} + 7.04$ となる．3 段以上のパイプライン化の際もこのようにして見積り値を得ることができる．このように最小単位モジュールに基づきパイプラインレジスタを挿入して遅延値を見積もる手法は演算ユニットが最小単位モジュールの集合であるという考えに帰着するため，乗算器によらず加算器，乗加算器においても適用可能である．

4.2　アドレッシングユニット遅延見積り

アドレッシングユニットは ALU やシフタ等と同様にスロット i の中に含まれ EXE1 ステージに付加されるため，得られた遅延時間の見積り値は機能ユニット遅延と同じ扱いとなる．ただし，アドレッシングユニットに入力される値が通るパスは通常の演算パスとは異なりフォワーディングのパスは通らず，デコードステージと EXE1 ステージ間のパイプラインレジスタから出力される命令の制御線からアドレスレジスタ値を演算器により加工して元のレジスタに格納するまでとなる．このため，セレクタ遅延はアドレッシングユニットの遅延パスには含まれず，機能ユニット遅延時間がパスとなる．アドレッシングユニットの遅延時間は表 6 より，間接アドレッシングモードに依存し，modulo add を持つもの（遅延時間 3.86〜3.27：ラベル 6，10，11），modulo add を持たないもの（遅延時間 1.59〜1.17：ラベル 2, . . . , 5, 7, . . . , 9）の 2 つに分類できる．これは，データが加算器を通過する回数の違いから分けられる．modulo add 以外の間接アドレッシングモードは 1 単位時間にインクリメントやインデクス加算の 1 度のみを行うのに対し，modulo add はインデクス加算をした後でモジュロ演算を行うため，加算を 2 度行っているためである．このため，modulo add の有無で分け，それぞれの遅延時間の最大値をとることでアドレッシングユニットの遅延時間とする．以上からアドレッシングユニットは以下のような性質を持つことが分かる．

性質 7　遅延時間見積りに関して，アドレッシングユニットは modulo add を持つもの（3.86 [ns]），modulo add を持たないもの（1.59 [ns]）の 2 種類に分類される．

5. 検証実験結果

本章では，対象となるプロセッサコアとして評価実験用にプロセッサを 17 種類用意し，導出した面積見積り式，遅延時間見積り式によって見積もられた値と論理合成値を比較検証した結果について報告する．

5.1　精 度 検 証

本節では，導出した面積/遅延の見積り式で得られる見積り値の精度の評価を行う．実装した評価実験用のプロセッサのパラメータと面積/遅延見積り値，論理合成値を**表 7**，アドレッシングユニットのパラメータを**表 8**，評価実験用のプロセッサが実行可能な SIMD 命令セットを**表 9** に示す．また，面積/遅延見積り値と論理合成値を比較した図をそれぞれ**図 16**，**図 17** に示す．

面積見積り値の相対誤差は最大で 5.62%，最小で 0.42%であり，平均相対誤差は 2.25%であった．遅延見積り値の誤差は最大で 1.60 ns，最小で 0.01 ns であり，平均誤差は 0.54 ns となった．誤差の原因として，面積見積りでは，機能分割された HW ごとに見積もった際にそれぞれの少量の誤差が集約されたことが考えられる．たとえば，sample I におけるプロセッサカーネルの見積り値は 21,989 [μm^2]，論理合成値は 26,012 [μm^2] であり，約 18[%] の誤差となっている．プロセッサカーネルは機能ユニットやレジスタファイル等を使うための制御を主に担っていることから，面積見積りに関しての誤差は制御構造の見積りが十分でないために引き起こされるものであるといえる．改善提案として制御構造の見積りを詳細に行うことによってより論理合成値に近い値での見積りが可能となる．遅延見積りでは，演算部分以外での推定パスと合成後のパスの不一致が考えられる．遅延時間見積り手法は各部分機能ごとの遅延値の和からプロセッサコアの遅延値を導出しているため，遅延パスの不一致により誤差が大きくなっている．たとえば，sample I における見積りのクリティカルパスは SIMD 演算ユニットを通るパスであるが，SIMD 演算ユニットの遅延時間は各部分機能ユニットの最大の遅延時間をとることで導出している．遅延見積りにおいては，各機能ユニットを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，各機能ユニット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．実際の SIMD 演算ユニット内のパス遅延はシフト部が 0.13 [ns] でありシフト部の遅延見積り値が 0.86 [ns] であることから大きな誤差が生じている．このため，見積り対象となるハードウェア全体のクリティカルパスを探索することによってパスの不一致を解消し，より精度の高い見積りを実現することが可能となる．しかしながら，図 16 を見ると面積見積り値は相対誤差を考えても論理合成値に非常に近い値をとっているといえる．

ここで，面積はハードウェア構成によって検証用プロセッサごとに論理合成値と見積り値の値の差がまったく異なることから相対誤差を用いている．たとえば，表 7 におけるラベ

表 7　見積り式検証用プロセッサの各パラメータ
Table 7　Parameters of processors for verifying estimation equation.

ラベル	並列度 n	パイプライン段数 s	命令長 i	レジスタ数 k	演算器	adrs unit	HW loop	面積 [μm^2] 見積り値	面積 [μm^2] 論理合成値	相対誤差 [%]	遅延 [ns] 見積り値	遅延 [ns] 論理合成値	誤差 [ns]
A	1	3	12	16	ALU1, SFT1 SIMDMAC1	×	0	64,696	65,327	0.97	9.93	9.95	−0.02
B	1	3	10	16	ALU1, SFT1 MAC1 SIMDMUL1	○	nest2	104,756	105,862	1.05	10.18	8.99	1.20
C	1	3	13	32	ALU1, SFT1 SIMDADD1	×	0	49,814	50,387	1.14	2.94	2.43	0.51
D	1	4	10	16	ALU1, SFT1 SIMDMUL1	○	0	77,865	79,670	2.27	6.00	4.95	1.05
E	1	5	12	16	ALU1, SFT1 MAC1	×	0	70,895	73,236	3.20	2.80	2.73	0.07
F	1	6	6	4	ALU1, SFT1 MUL1	×	nest3	70,349	74,540	5.62	2.00	2.01	−0.01
G	2	3	9	16	ALU1, SFT2 MUL1, MAC1	○	0	111,781	116,920	4.40	5.52	5.40	0.12
H	2	3	10	8	ALU1, SFT1	×	0	30,407	30,391	0.05	2.10	1.94	0.16
I	2	3	14	8	ALU1, SFT1 SIMDMUL1	×	0	64,229	67,803	5.27	10.58	8.98	1.60
J	2	4	12	32	ALU1, SFT2 MUL1	○	0	153,963	158,158	2.65	3.86	4.09	−0.23
K	2	5	7	64	ALU2, SFT2 SIMDMAC1	×	nest3	167,232	161,985	3.24	6.01	4.95	1.06
L	2	6	8	4	ALU1, SFT1 MAC1	○	0	85,228	86,531	1.51	3.86	3.45	0.41
M	4	3	6	64	ALU2, SFT2 SIMDADD1 MAC2	×	nest1	269,522	278,752	3.31	5.88	5.64	0.24
N	4	3	12	64	ALU2, SFT2 SIMDMUL1 MAC2	×	nest1	203,164	204,706	0.75	10.05	8.49	1.56
O	4	4	11	16	ALU1, SFT3 MUL1	×	nest3	131,243	129,976	0.98	3.12	2.94	0.18
P	4	5	12	8	ALU2, SFT1 MUL1, MAC1	○	nest2	155,324	155,984	0.42	3.28	2.91	0.37
Q	4	6	8	4	ALU2, SFT2	○	nest1	102,908	104,422	1.45	3.86	3.48	0.38

表 8　アドレッシングユニットのパラメータ
Table 8　Parameters of addressing units.

ラベル	メモリバンク数	レジスタ数	no change	post inc	post dec	idx add	bit reverse	modulo add
B	1	10	○	○	○	×	×	×
C	2	8	○	×	○	×	×	×
F	1	4	○	○	×	×	○	×
H	2	12	○	○	○	○	○	○
J	1	6	○	×	×	×	×	○
M	2	2	○	○	×	×	×	×
N	1	8	○	○	○	○	○	○

表 9　見積り式検証用 SIMD 命令セット
Table 9　SIMD instruction set to verify the estimation equation.

ラベル	命令セット
A	mac_1_sr10s, mac_2_sr24s, mac_2_ur1w mac_2h_ur1, mac_1l_sr10
B	mul_1_sr1-10w, mul_2_sr_5w, mul_1h_u mul_4_sr12-14w_h, mul_4_sl11w, mul_4l_s
C	add_1_ur15w, add_2_ul31w, add_2h_u
D	mul_1_sl31s, mul_2_sl13s, mul_2_ul1-31w mul_1l_sl13, mul_2h_ul5-10
I	mul_4_ul8w, mul_4_ul_4w, mul_1l_u mul_1_sl12w, mul_4_ul4s
K	mac_1_ul2s, mac_4_ul5s
M	add_4_ur27w, add_1_sr31w, add_1h_s add_1_sr30w, add_2_sr_3w
N	mul_1_ur10w, mul_2_ul24w, mul_4_ul8w

ル C の論理合成値と見積り値の絶対誤差をとると 573 [μm^2]（相対誤差：1.14[%]）となるが，ラベル Q の絶対誤差は 1,514 [μm^2]（相対誤差：1.45[%]）となる．2 つの検証用プロセッサにおいて，絶対誤差でみると大きな差があるが相対誤差では近い値をとっていることになる．このことから，面積見積りにおいて高精度な見積りを回路規模に応じて評価するために相対誤差を用いた．遅延に関してはハードウェア構成の規模の大きさと遅延時間に関連が強いわけではないため，各検証用プロセッサどうしの比較が容易なことから絶対誤差での評価を行った．

5.2　忠実度検証

本節では，見積り値の忠実度による評価を行う．見積りの忠実度とは，任意の異なるアーキテクチャ間でプロセッサコア実装後の測定値との大小関係を正しく見積もった割合で定義される[1)]．忠実度が高いほど，その見積り式で得られる見積り値の信頼性が高まり，HW/SW 分割における評価値としての適切なアーキテクチャ探索を行うことが可能となる．以下に見積りの忠実度を求める方法を説明する．

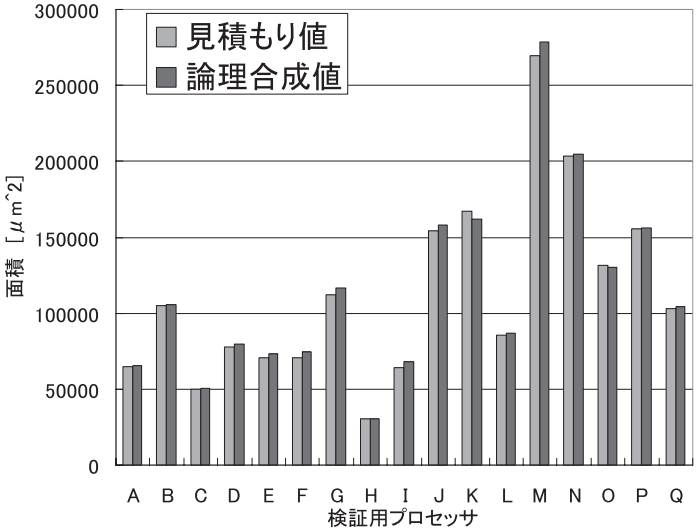


図 16　面積見積り値と論理合成値
Fig. 16　Area of estimation and logic synthesis values.

$D_1, D_2, \ldots, D_n$ を異なるプロセッサコアとし，$1 \leqq i,\ j \leqq n$ かつ $i \neq j$ を満たす $i,\ j$ に対して，μ_{ij} を以下のように定義する．

$$u_{ij} = \begin{cases} 1 & \text{if } E(D_i) > E(D_j) \text{ and } M(D_i) > M(D_j), \text{or} \\ & \quad E(D_i) < E(D_j) \text{ and } M(D_i) < M(D_j), \text{or} \\ & \quad E(D_i) = E(D_j) \text{ and } M(D_i) = M(D_j) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

上式において $E(D_n)$ は見積り値，$M(D_n)$ は論理合成後の測定値を示す．見積りの忠実度 F は論理合成後の測定値の比較関係を正しく見積もった割合で定義され，以下の式で求められる．

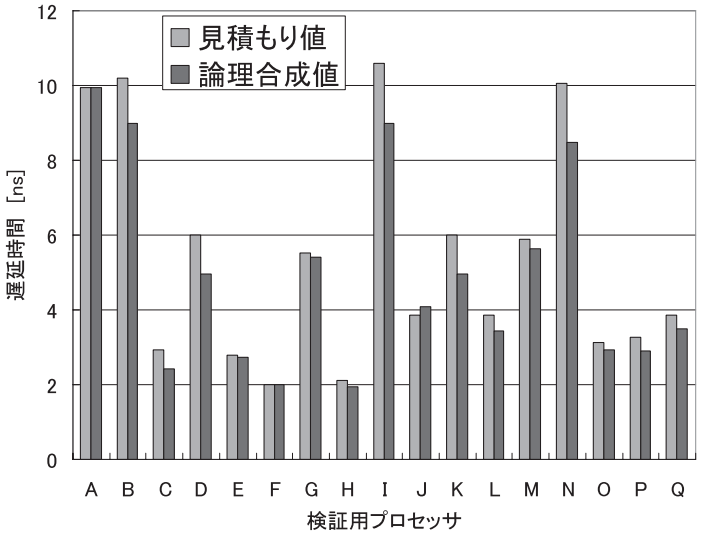


図 17　遅延時間見積り値と論理合成値
Fig. 17　Delay of estimation and logic synthesis values.

$$F = 100 \times \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mu_{ij} \quad [\%] \tag{16}$$

表 7 の検証用プロセッサに対して面積見積り式と遅延時間見積り式における忠実度による評価を行った結果，忠実度は面積見積りで 97.8%，遅延見積りで 90.4%であった．精度検証での見積り値の正確性を考慮すると論理合成で得られる値の大小関係と，見積り式により得られる見積り値の大小関係は高確率で一致するといえ，相対的なアーキテクチャの比較に十分に利用できると考えられる．

6. おわりに

本稿では，SPADES プロセッサコアを対象とした SIMD 演算ユニットおよびアドレッシングユニットの構成の変化に対応した面積/遅延時間見積り手法を提案し，見積り値の精度・忠実度に関する評価を行い提案手法により得られる見積り値は論理合成値に非常に近い値をとり，アーキテクチャの比較に十分に利用できることを示した．

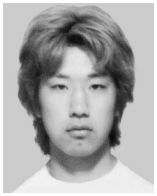
本稿における見積り手法では 90 [nm] のテクノロジをモデルとしているため，使用するテクノロジが変更された場合には対応しておらず，異なるテクノロジで本手法を適用するならば同様のサンプルをとる必要がある．ゆえに，異なるテクノロジへの対応は今後の課題となる．しかし，テクノロジが変更された場合でも各パラメータがプロセッサコアに与える影響には同様の傾向があると考えられ，本手法を系統立てて使用することにより見積り値を導出する時間を短縮することが可能であると考えられる．また，本稿では，論理合成値に対する見積りを行ったが，配置配線を行う工程までを考慮すると誤差は大きくなることが考えられる．このため，今後は配置配線後の見積りへの対応を行い，より広範囲なアーキテクチャの探索に対して精度の高い見積りを行うことが必要となる．

参 考 文 献

1) Gajski, D., Vahid, F. and Gong, S.N.J.: *Specification and Design of Embedded Systems*, PTR Prentice Hall, Inc. (1994).
2) Huang, I.-J. and Despain, A.M.: Synthesis of instruction sets for pipelined microprocessors, *31st DAC*, pp.5–11 (1994).
3) Kohara, S., Kurihara, A., Miyaoka, Y., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: A Pipelined Functional Unit Generation Method in HW/SW Cosynthesis System for SIMD Processor Cores, *Proc. SASIMI '06*, pp.287–294 (2006).
4) 久須裕之，塩見彰睦，伊藤真紀子，今井正治：PEAS-III におけるアーキテクチャレベル見積り手法，DA シンポジウム'01 論文集，pp.211–216 (2001).
5) Morifuji, T., Takeuchi, Y., Sato, J. and Imai, M.: Flexible hardware model database management system: Implementation and effectiveness, *Proc. SASIMI '97*, pp.83–89 (1997).
6) Pitkanen, T., Rantanen, T., Cilio, A.G.M. and Takala, J.: Hardware Cost Estimation for Application-Specific Processor Design, *Embedded Computer Systems: Architectures, Modeling, And Simulation: 5th International Workshop, SAMOS 2005*, Hamalainen, T.D., Pimentel, A.D., Takala, J. and Vassiliadis, S. (Eds.), Samos, Greece, pp.222–231, Springer (2005).
7) 酒田輝昭，木村　勉，武内良典，今井正治：コンフィギュラブル・プロセッサの遅延時間見積もり手法の提案，信学技報 VLD，pp.45–50 (2001).
8) Togawa, N., Kataoka, Y., Miyaoka, Y., Yanagisawa, M. and Ohtsuki, T.: Area and Delay Estimation in Hardware/Software Cosynthesis for Digital Signal Processor Cores, *Trans. IEICE*, Vol.E84-A, No.11, pp.2639–2647 (2001).
9) Togawa, N., Tachikake, K., Miyaoka, Y., Yanagisawa, M. and Ohtsuki, T.: Instruction set and functional unit synthesis for SIMD processor cores, *Asp-Dac '04*, pp.743–750 (2004).

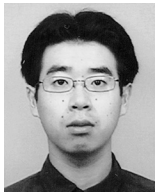
10) Togawa, N., Tachikake, K., Miyaoka, Y., Yanagisawa, M. and Ohtsuki, T.: A hardware/software partitioning algorithm for processor cores with packed SIMD-type instructions, *Trans. IEICE*, Vol.E86-A, No.12, pp.3218–3224 (2003).
11) 山崎大輔，小原俊逸，戸川　望，柳澤政生，大附辰夫：HW/SW 協調合成におけるアプリケーションプロセッサの面積/遅延見積もり手法，信学技法 VLD，pp.1–6 (2006).
12) 山崎大輔，小原俊逸，戸川　望，柳澤政生，大附辰夫：HW/SW 協調合成における ASIP の面積/遅延見積もり手法，DA シンポジウム'07 論文集，pp.31–36 (2007).

(平成 20 年 1 月 9 日受付)
(平成 20 年 7 月 1 日採録)



山崎　大輔

1982 年生．2006 年早稲田大学理工学部電子・情報通信学科卒業．2008 年同大学大学院修士課程修了．現在ソニー（株）．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．



小原　俊逸（正会員）

1979 年生．2001 年早稲田大学理工学部電子・情報通信学科卒業．2003 年同大学大学院修士課程修了．現在同大学院博士後期課程在学．VLSI 設計，特にプロセッサ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）

1970 年生．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 年同大学院博士後期課程修了．博士（工学）．現在，早稲田大学大学院基幹理工学研究科情報理工学専攻准教授．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）

1959 年生．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 年同大学院博士後期課程修了．工学博士．現在，早稲田大学大学院基幹理工学研究科情報理工学専攻教授．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）

1940 年生．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，早稲田大学大学院基幹理工学研究科情報理工学専攻教授．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，電気学会会員，プリント回路学会各会員．