システムＬＳＩ設計技術　101－2
(2001. 5. 17)

ディジタル信号処理向けプロセッサコアの Packed SIMD型ハードウェアユニット生成手法

宮岡祐一郎† 戸川望‡,* 柳澤政生† 大附辰夫†

† 早稲田大学理工学部電子・情報通信学科
‡ 北九州市立大学国際環境工学部情報メディア工学科
* 早稲田大学理工学総合研究センター

〒169-8555 東京都新宿区大久保3-4-1
Tel: 03-5286-3396　E-mail: miyaoka@ohtsuki.comm.waseda.ac.jp

あらまし
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Packed SIMD型命令は画像処理などに有効である．Packed SIMD型命令を持つプロセッサコアをハードウェア/ソフトウェア協調合成システムによって合成するとき，必要な命令が実行できるPacked SIMD型演算ユニットを構成し面積と遅延を高速に見積もることが要求される．そこで，本稿では複数のハードウェアユニットを高速に構成するPakced SIMD型ハードウェアユニット生成手法を提案する．本手法は，1つのハードウェアユニットで実行される命令の集合と，生成されるハードウェアユニットの面積と遅延の制約を入力とし，ハードウェアユニットに必要となる部分機能を抽出して，その部分機能を実現するハードウェアを組み合わせることでハードウェアユニット構成を複数列挙し面積と遅延の見積もり値を出力する．提案手法を計算機上に実装しPacked SIMD型演算ユニットに適用した結果を示しその有効性を評価する．
キーワード　ハードウェア/ソフトウェア協調合成，Packed SIMD型命令，ハードウェアユニット，ハードウェアユニット生成，アーキテクチャ構成

A Hardwareunit Generation Algorithm for Packed SIMD Type Functional Units of Digital Signal Processor Cores

Yuichiro MIYAOKA† Nozomu TOGAWA‡,* Masao YANAGISAWA† Tatsuo OHTSUKI†

† Dept. of Electronics, Information and Communication Engineering, Waseda University
‡ Dept. of Information and Media Sciences, the University of Kitakyushu
* Advanced Research Institute for Science and Engineering, Waseda University

3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan
Tel: 03-5286-3396　E-mail: miyaoka@ohtsuki.comm.waseda.ac.jp

Abstract
Consider to synthesize a processor core with packed SIMD type instructions by a hardware/software cosynthesis system. The system is required to configure functional units executing packed SIMD type instructions and obtain the area and delay of the functional units to evaluate the synthesized processor core. This paper proposes a hardware unit generation algorithm for packed SIMD type functional units. Given a set of instructions to be executed by a hardware unit and constraints for area and delay of the hardware unit, the proposed algorithm extracts a set of subfunctions to be required by the hardware unit and generates more than one architecture candidates for the hardware unit. The algorithm also outputs the estimated area and delay of each of the generated hardware units. The execution time of the proposed algorithm is very short and thus it can be easily incorporated into the processor core synthesis system. Experimental results for packed SIMD type functional units demonstrate effectiveness and efficiency of the algorithm.
Key Words　Hardware/software cosynthesis, packed SIMD type instructions, hardware units, hardware unit generation, architecture configuration

1 まえがき

bビットの演算ユニットを改良すれば，1つのbビット演算ユニットを用いてn個のb/nビット演算を実行させることができる．この演算を，Packed SIMD 型演算と呼ぶ．[2, 6, 9] などで報告されているプロセッサは，Packed SIMD 型演算を用いた命令を持つことで，画像処理アプリケーションを画素並列に処理することができ，高速に実行することが可能である．Packed SIMD 型演算はn個のデータを並列に扱うので，個々のデータの演算に対して特定の処理をすることはできない．そこで，Packed SIMD 型演算は，演算結果のあふれを飽和させる処理や，$n/2$個の演算結果のみをレジスタに書き込むことによって演算の精度を保つ処理を一連の流れとして扱う命令で用いるのが有効である．この命令を Packed SIMD 型命令と呼ぶ．Packed SIMD 型命令は同種の演算（加算・乗算など）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 Packed SIMD 型命令のみが必要となるので，Packed SIMD 型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/ソフトウェア協調設計が有効であると考えられる．

我々はディジタル信号処理向けプロセッサコアのハードウェア/ソフトウェア協調合成システムを提案している [8]．ハードウェア/ソフトウェア協調合成システムの核となるハードウェア/ソフトウェア分割では，プロセッサが持つ命令を最適化し，合成されるプロセッサコアの面積とアプリケーションの実行時間を見積もって，複数の候補の中から要求にあった面積と実行時間を持つプロセッサコアの構成を得る．プロセッサの持つ命令に応じて，プロセッサコアに付加される演算ユニット，アドレッシングユニットやハードウェアループユニットが決定される．演算ユニットに代表される，プロセッサの命令を実行するための機能をハードウェアユニットと呼ぶ．プロセッサコアの面積とアプリケーションの実行時間の見積もりには，ハードウェアユニットの面積と遅延の見積もりが不可欠である．Packed SIMD 型演算ユニットのように，実行できる命令が複数あるハードウェアユニットでは，取りうるすべての構成に対して面積と遅延の見積もり値を得る必要がある．複数の候補の中から要求にあった面積と実行時間を持つプロセッサコアの構成を得るためには，プロセッサが持つ命令を繰り返し再構成することによって何種類ものプロセッサコアの構成を作る必要がある．そのため，ハードウェアユニットの面積と遅延の見積もりは高速に実行されなければならない．

これまでの我々のシステムや [8]，その他のプロセッサレベルのハードウェア/ソフトウェア協調設計やプロセッサ記述の合成に関する研究例 [1, 7, 10] では，必要とするハードウェアユニットを人手で設計して用意している．しかし，Packed SIMD 型演算ユニットは取りうる構成が多数あるため，その全てを用意するのが困難である．[3] などでは，ハードウェアユニットの管理システムとして FHM-DBMS[4] を用いているが，FHM-DBMS はハードウェアユニットのアーキテクチャや演算アルゴリズムを指定して，面積や遅延の見積もりを出力するため，面積や遅延の値を考慮しながら高速に複数の候補を列挙するのには向いていない．

以上の背景から，Packed SIMD 型演算ユニットを対象としたハードウェアユニット生成系を提案する．1つのハードウェアユニットで実行される命令の集合と，生成されるハードウェアユニットの面積と遅延の制約を入力とし，ハードウェアユニットの構成を複数列挙し面積と遅延の見積もり値を出力する．複数の解候補を列挙することで，同じ命令集合に対して，何度もハードウェアユニットの面積や遅延の見積もり値を呼び出すことなくハードウェア構成を選択することが可能である．ハードウェアユニット生成系は，入力された命令から必要な機能を抽出する部分機能抽出と，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部分機能ユニットを割り当てるアーキテクチャ構成により実現される．提案するアーキテクチャ構成アルゴリズムにより，すべてのハードウェアユニットで実現される命令集合の組み合わせに対して，その構成を高速に列挙することを可能とする．

本稿は以下のように構成される．2 章では Packed SIMD 型命令セットを定義する．3 章ではハードウェアユニット生成系を提案する．4 章では高速にハードウェアユニットの面積・遅延を見積もるためのアーキテクチャ構成アルゴリズムを提案する．5 章では提案したハードウェアユニット生成手法を計算機上に実装し，Packed SIMD 型ハードウェアユニットに適用した結果を報告する．

2 Packed SIMD 型命令

本章では，Packed SIMD 型命令 [5] を定義する．Packed SIMD 型命令は，bビットの演算ユニットを用いて，同時にn個のb/nビット演算をする命令である．ここでnを梱包数と呼び，$k = b/n$とする．Packed SIMD 型命令を表 1 に示す．

表 1: Packed SIMD 型命令．

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

算術演算命令　算術演算命令は次の2種類に分けることができる．

n 個の k ビット演算命令（図 1） n 個の k ビットデータが2組与えられて，n 個の k ビット演算が同時に実行される．演算に対して，(i) 符号の有無，(ii) 演算結果のシフトの有無およびシフト量・シフト方向，および (iii) 飽和処理の有無を選択できる．この命令を {#1}_{#2}_{#3}{#4}{#5} という形式で表す．#1 は演算の種類を表し，#2 は梱包数を表す．#3 は符号の有無を s および u で表し，#4 は演算結果のシフトをするとき，その方向を r と l，シフト量を数字で表す．#5 は飽和処理の有無を s および w で表す．例えば，梱包数が4，符号あり，2ビット右シフト，飽和ありの乗算命令を MUL_4_sr2s と表す．

$n/2$ 個のビット拡張 k ビット演算（図 2） n 個の k ビットデータが2組与えられたとき，上位あるいは下位を構成する $n/2$ 個の k ビットデータに対して演算が同時に実行され，結果を $2 \times k$ ビットとして得る．演算に対して (i) 上位と下位のどちらに演算を実行するか，(ii) 符号の有無，(iii) 演算結果のシフトの有無およびシフト量・シフト方向を選択できる．この命令を {#1}_{#2}{#3}_{#4}{#5} という形式で表す．#1 は演算の種類，#2 は梱包数を表す．#3 は上位か下位かを h と l で表し，#4 は符号の有無，#5 はシフト方向と量を表す．例えば，梱包数4のデータの上位2つのデータに対して，符号ありで演算結果を7ビット左にシフトする加算命令を ADD_4h_sl7 と表す．

シフト演算命令　シフト演算も算術演算と同様に，n 個の k ビット演算命令と $n/2$ 個のビット拡張 k ビット演算に分けることができる．2個のデータに対して算術右シフトをする命令を SRA_2，4個のデータの下位2個のデータを左シフトする命令を SLL_4 と表す．

ビット拡張，縮小命令　ビット拡張命令 EXTD は，n 個のデータの上位，あるいは下位を構成する $n/2$ 個のデータに対して，上位ビットを拡張して $2 \times k$ ビットのデータを構成するものである（図 3）．ビット拡張は (i) 符号の有無，(ii) 拡張結果のシフトの有無を指定できる．

ビット縮小命令 EXTR は，n 個の $2 \times k$ ビットデータが与えられて，それぞれのデータの下位 k ビットを抽出することで n 個の k ビットデータを構成する命令である（図 4）．ビット縮小命令は (i) 符号の有無，(ii) 与えられた $2 \times k$ ビットデータのシフトの有無および (iii) 飽和処理の有無を指定できる．

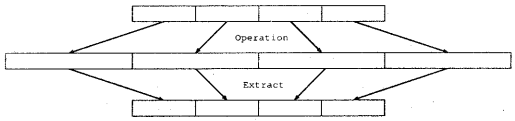

図 1: n 個の k ビット演算．

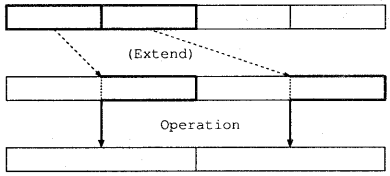

図 2: $n/2$ 個のビット拡張 k ビット演算．

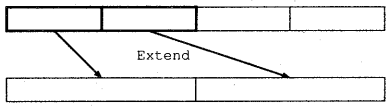

図 3: ビット拡張命令．

図 4: ビット縮小命令．

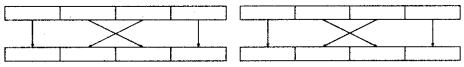

図 5: 交換命令．

交換命令　n 個の k ビットデータが2組与えられたとき，k ビットデータの位置を交換し新たな n 個の k ビットデータを2組構成する命令を EXCH 命令と呼ぶ（図 5）．EXCH 命令では，データが交換される場所は梱包数によってあらかじめ決定されている．

3　ハードウェアユニット生成系

本章では，まず Packed SIMD 型命令セットを持つプロセッサのハードウェア/ソフトウェア協調合成システムの中で，特にハードウェア/ソフトウェア分割とハードウェアユニットの面積・遅延の見積もりとの関係について検討し，その後ハードウェアユニット生成系を提案する．

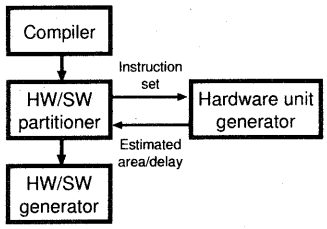

図 6: ハードウェア/ソフトウェア協調合成システムとハードウェアユニット生成.

3.1 ハードウェア/ソフトウェア協調合成システムとハードウェアユニット

ディジタル信号処理向けプロセッサのハードウェア/ソフトウェア協調合成システム [8] は，C 言語で書かれたアプリケーションプログラム，アプリケーションデータおよびアプリケーションの実行時間制約を入力として，プロセッサコアのハードウェア記述，オブジェクトコードおよびソフトウェア環境を出力するものである．システムは主にコンパイラ，ハードウェア/ソフトウェア分割，およびハードウェア/ソフトウェア生成から構成される．

ハードウェア/ソフトウェア分割では，コンパイラの出力したアセンブリコードから決まるプロセッサアーキテクチャを再構成し，最適なハードウェア構成を決定する．変更の対象の中心は，演算ユニットなどに代表されるハードウェアユニットである．そのため，ハードウェア/ソフトウェア分割にはハードウェアユニットの面積や遅延の見積もりが必要となる（図 6）．例えばハードウェアユニットを 1 つ削減すればプロセッサコアの面積が削減され，そのハードウェアユニットがクリティカルパス上に存在してプロセッサコアのクロック周期を決定していれば，クロック周期が短くなる．Packed SIMD 型命令セットを持つプロセッサを対象としたとき，Packed SIMD 型演算ユニットは 1 つの演算ユニットで複数の命令を実行するため，演算ユニットで実行する命令の組み合わせによって，ハードウェアユニットの面積や遅延の値が異なる．命令の再構成は，プロセッサコアの面積の削減やあるいはアプリケーションの実行時間の削減を目的とするので，命令の再構成によって得られる新たな Packed SIMD 型演算ユニットは，命令を再構成する前の Packed SIMD 型演算ユニットより面積やクリティカルパス遅延の値が小さいものでなければならない．ハードウェア/ソフトウェア分割では，命令を再構成するごとにそのときのプロセッサコアの面積やアプリケーションの実行時間を見積もってプロセッサアーキテクチャを決定するので，高速にハードウェアユニットを構成し，面積と遅延を見積もる必要がある．

3.2 ハードウェアユニット生成系

3.1 節で検討した，ハードウェア/ソフトウェア分割で必要となるシステムをもとに，ハードウェアユニット生成系を提案する．

3.2.1 アーキテクチャテンプレート

ハードウェアユニットは，部分的な機能を実現するハードウェア（部分機能ユニットと呼ぶ）を複数組み合わせて実現される．部分機能ユニットが実行する処理を部分機能と呼ぶ．ハードウェア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．ハードウェアユニットの面積は，アーキテクチャテンプレートの部分機能を実現する部分機能ユニットの面積の総和に，アーキテクチャテンプレートにより定まる制御部の面積を足すことで見積もられる．ハードウェアユニットの遅延は，アーキテクチャテンプレートにより定まるクリティカルパス上にある部分機能ユニットのクリティカルパス遅延と制御部の遅延から見積もることができる．

例えば，`mul_4_sr3s` を実行するハードウェアユニットは，梱包数 4 の符号つき Packed SIMD 型乗算，右 3 ビットシフトおよび符号つき飽和の 3 つの部分機能と図 7 に示す接続関係を持つアーキテクチャテンプレートを持つ．図 8 のように，図 7 の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ことで，複数のハードウェアユニット構成を得ることができる．

3.2.2 システムの概要

ハードウェアユニット生成系は 1 つのハードウェアユニットで実行される命令の集合と，生成されるハードウェアユニットの面積と遅延の制約を入力とし，ハードウェアユニットの構成を複数列挙し面積と遅延の見積もり値を出力する．ハードウェアユニット生成系の概要を図 9 に示す．ハードウェアユニット生成系は部分機能抽出とアーキテクチャ構成から成る．部分機能抽出では，入力された命令集合をもとにアーキテクチャテンプレートを決定する．アーキテクチャ構成では，アーキテクチャテンプレートの部分機能に，その部分機能を実現する部分機能ユニットを割り当てることによりハードウェアユニットの構成と面積および遅延の見積もりを得る．

部分機能ユニットを組み合わせてハードウェアユニットを実現することにより，いくつかの部分機能ユニットを用意するだけで入力された命令集合を実現するハードウェアユニットが生成可能である．これにより，ハードウェア/ソフトウェア分割によって実行する命令が変更されたとき，新たなハードウェアユニットを構成するこ

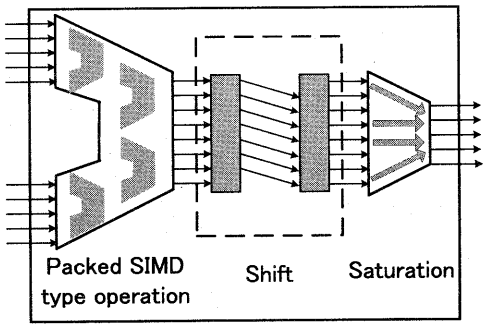

図 7: アーキテクチャテンプレート.

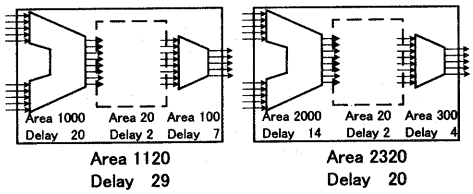

図 8: 部分機能ユニット.

とができ，しかも面積や遅延の制約を与えることにより面積削減や遅延の削減を目的としたハードウェア/ソフトウェア分割に対して，必要なハードウェアユニットの構成のみを出力することができる．複数の解候補を列挙することで，同じ命令集合に対して，何度もハードウェアユニットの面積や遅延の見積もり値を呼び出すことなくハードウェア構成を選択することが可能である．したがって，ハードウェア/ソフトウェア分割を高速に実行することができる．

4 アーキテクチャ構成アルゴリズム

本章では，ハードウェアユニット生成系の中心となるアーキテクチャ構成を問題として定義し，高速にハードウェアユニットを構成するアーキテクチャ構成アルゴリズムを提案する．

4.1 アーキテクチャ構成問題

ハードウェアユニット u で実行可能な命令集合 I_u を u の命令集合と呼ぶ．面積制約を満たすとは，ハードウェアユニット u の面積が許容値 a_{max} より小さいことであり，時間制約を満たすとは，ハードウェアユニット u のクリティカルパス遅延が許容値 d_{max} より小さいことである．このときハードウェアユニットのアーキテクチャ構成問題とは，入力として命令集合 I から決定されるアーキテクチャテンプレート，面積制約 a_{max} および時

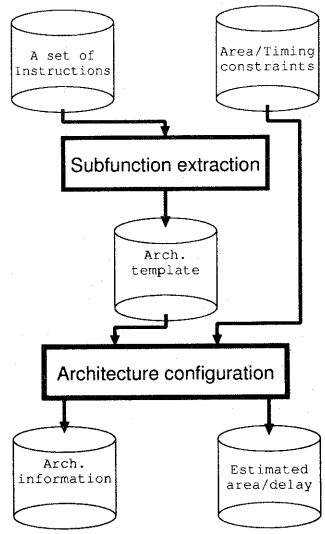

図 9: ハードウェアユニット生成系.

間制約 d_{max} を与えられたときに，面積制約と時間制約を満たし，$I \subseteq I_u$ となるハードウェアユニット u のアーキテクチャを複数個出力することである．

4.2 アーキテクチャ構成アルゴリズム

アーキテクチャテンプレートの部分機能の集合を $F = \{f_1, f_2, \ldots, f_m\}$ とする．入力されるアーキテクチャテンプレートによって f_i $(i = 1, \ldots, m)$ を実現可能な部分機能ユニットの集合 S_i を選択することができる．部分機能ユニット $s \in S_i$ は面積 $a(s)$ と遅延 $d(s)$ を持つ．

アーキテクチャ構成問題は，複数の解を高速に列挙する必要がある．解として得られる個々のハードウェア構成は，同じ遅延を持つハードウェア構成の中で最も面積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，各部分機能 f_i $(i = 1, \ldots, m)$ に対して最も遅延の小さい部分機能ユニットを割り当てることでハードウェアユニットを構成する．得られた構成は解候補の中で最も遅延の小さいハードウェアユニットとなり，時間制約を満たしていることが期待される．

次に，現在の構成から1つの部分機能 f_i に着目し，f_i を実現している部分機能ユニットを，部分機能ユニット集合 S_i から，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に置き換え，そのときのハードウェアユニットの面積と遅延を見積もる．すべて部分機能に対してこの処理を実行し，得られたハードウェアユニット構成の中で，時間制約を満たし最も面積の小さい構成を解の候補とし新たな構成とする．

Step 1 アーキテクチャテンプレートに基づいて，ハードウェアユニットを構成する．このとき，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ものを選ぶ．構成されたハードウェアユニットが時間制約を満たさなければ終了．

Step 2 現在の構成によるハードウェアユニットが面積制約を満たせば，その構成を解の 1 つとして出力する．

Step 3 現在のハードウェアユニットのアーキテクチャから，ある 1 つの部分機能を実現する部分機能ユニットを，現在の部分機能ユニットより面積が小さいユニットの中で最も遅延の小さいユニットに置き換えたときの，ハードウェアユニットの面積とクリティカルパス遅延を調べる．

Step 4 すべての部分機能に対して **Step 3** を実行し，時間制約を満たす中でハードウェアユニットの面積を最小とする構成に変更する．

Step 5 **Step 4** のようなハードウェアユニットが存在する間，**Step 2–4** を繰り返す

図 10: アーキテクチャ構成アルゴリズム．

この操作で得られる新たなハードウェアユニットの構成は m 回のみの部分機能ユニットの交換で得られる．この操作を時間制約を満たさなくなるまで続けることで，徐々に面積の小さい構成を得ることができ，高速に複数の解候補を列挙できる．

最後に得られた解候補の中で，面積制約を満たすものを解とし列挙する．

以上のアルゴリズムを図 10 に示す．次に，この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 10 の **Step 2** は $O(1)$ で実行可能である．すべての部分機能に対して **Step 3** を実行するので **Step 2–4** は $O(m)$ で求められる．**Step 2–4** は最大で，部分機能ユニットの総数回繰り返されるので，時間計算量は $O(mn + n\log n)$ である．次にこのアルゴリズムの空間計算量を見積もる．n 個の部分機能ユニットを整列して保存し，m 個の部分機能を実現する部分機能ユニットを用意することでハードウェアユニットを構成するので，空間計算量は $O(m + n)$ である．

5 計算機実験結果

提案した手法を C 言語を用いて Sun Ultra SPARC (200MHz，メモリ 128MB) 上に実装し，Packed SIMD 型加算ユニットおよび Packed SIMD 型乗算ユニットに適用した．使用したコンパイラは gcc(version 2.95.2)，最適化レベルを O3 とした．部分機能ユニットはあらかじめ VHDL で記述し，Design Compiler を用いて論理合成して面積と遅延の値を算出した．セルライブラリには VDEC ライブラリ（CMOS0.35μm テクノロジー）を用いた[1]．提案するアーキテクチャ構成アルゴリズムと，比較手法としてすべての部分機能ユニットの組み合わせを全列挙するを選び，入力として表 2 に示す命令集合と，面積制約 $100,000\mu m^2$ および時間制約 $8.00ns$ を与えたときの，実験結果を図 11 に示す．入力として表 3 に示す命令集合と，面積制約 $780,000\mu m^2$ および時間制約 $16.0ns$ を与えたときの，実験結果を図 12 に示す．解を得るためのアルゴリズムを 1 万回実行したときの実行時間を表 4 に示す．図 11 と図 12 から，提案手法が，複数のハードウェアユニット構成を列挙できることがわかる．全列挙手法に比べて列挙された構成は Packed SIMD 型加算ユニットで 1 割程度で，Packed SIMD 型乗算ユニットで 3 割程度であるが，全列挙で得られる解の中の，同じ遅延を持つ中で最も面積の小さいハードウェアユニットの多くを列挙できる．したがって，ハードウェア/ソフトウェア分割で有用となるハードウェアユニット構成のみを扱うことができるようになる．しかも，表 4 から，その解が全列挙手法に比べて 1/10 以下の時間で得られるので，高速にハードウェア/ソフトウェア分割をすることができるようになると考えられる．以上からアルゴリズムの有効性を示せたと考える．

表 2: 入力した命令集合（加算）．

add	add_1_us
add_1_ur1w	add_1_ul1s
add_2_uw	add_2_us
add_2_ur1w	add_2_ul1s
add_4_uw	add_4_us
add_4_ur1w	add_4_ul1s
add_2h_u	add_2l_u
add_2h_ul8	add_2l_ul8
add_4h_u	add_4l_u
add_4h_ul8	add_4l_ul8

表 3: 入力した命令集合（乗算）．

mul_2_sw	mul_2_ss
mul_2_sr3w	mul_2_sr3s
mul_2_sr6w	mul_2_sr6

表 4: 1 万回実行したときの実行時間．

	提案手法 [ns]	全列挙 [ns]
加算	6.13	66.3
乗算	3.14	41.3

[1]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．

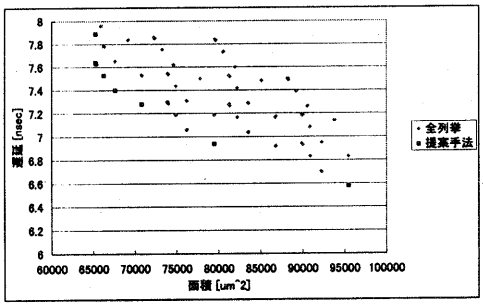

図 11: Packed SIMD 型加算ユニットの実験結果.

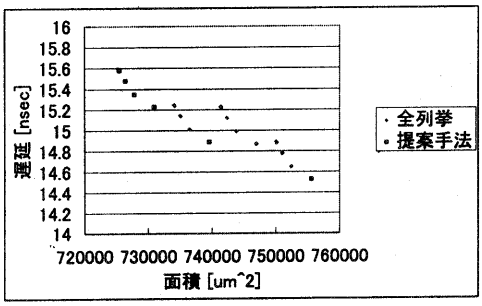

図 12: Packed SIMD 型乗算ユニットの実験結果.

6 むすび

Packed SIMD 型演算ユニットを対象としたハードウェアユニットの生成手法を提案し，高速に，面積・遅延の小さい解を複数列挙することができた．今後は，面積と遅延以外に，消費電力やマルチサイクル演算ユニットを対象としたサイクル数なども考慮したハードウェアユニット生成手法を検討する．

謝辞

本研究に関して，有用な議論，討論をいただいた本学野々垣直浩氏（現東芝）に感謝いたします．本研究の一部は，文部科学省科学研究費補助金 (奨励研究 (A)，課題番号 12750369) の援助を受けた．

参考文献

[1] H. Akaboshi and H. Yasuura, "COACH: A computer aided design tool for computer architectures," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E76-A, no. 10, pp. 1760–1769, 1993.

[2] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "AltiVec extension to PowerPC accelerates media processing," IEEE Micro, vol. 20, no. 2, pp. 85–94, 2000.

[3] M. Itoh, S. Higaki, J. Sato, A. Shiomi, Y. Takeuchi, A. Kitajima, and M. Imai, "PEAS-III: An ASIP design environment," in *Proceedings of the 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 430–436, 2000.

[4] T. Morifuji, Y. Takeuchi, J. Sato, and M. Imai, "Flexible hardware model database management system: implementation and effectiveness," in *Proc. of the Synthesis and System Integration Mixed Technologies(SASIMI'97)*, pp. 83–89, 1997.

[5] 野々垣直浩, 戸川望, 柳澤政生, 大附辰夫, "画像処理を対象とした Packed SIMD 型命令セットを持つプロセッサのハードウェア/ソフトウェア協調合成システムにおける並列化 C コンパイラ," 信学技法, VLD2000-139, ICD2000-215, 2001.

[6] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," IEEE Micro, vol. 16, no. 4, pp. 42–50, 1996.

[7] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai, "PEAS-I: A hardware/software codesign system for ASIP development," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E77-A, no. 3, pp. 483–491, 1994.

[8] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A hardware/software cosynthesis system for digital signal processor cores," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82-A, no. 11, 1999.

[9] M. Tremblay, J. M. O'Connor, V. Narayanan and L. He, "VIS speeds new media processing," IEEE Micro, vol. 16, no. 4, pp. 10–20, 1996.

[10] J.-H. Yang, B.-W. Kim, S.-J. Nam, Y.-S. Kwon, D.-H. Lee, J.-Y. Lee, C.-S. Hwang, Y.-H. Lee, S.-H. Hwang, I.-C. Park, and C.-M. Kyung, "MetaCore: An application-specific programmable DSP development system," *IEEE Trans. on Very Large Scale Integration(VLSI) systems*, vol. 8, no. 2, pp. 173–183, 2000.