社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－133（28）
2008／1／17

アプリケーションプロセッサのカーネル記述自動生成手法

日浦 敏宏† 小原 俊逸† 戸川 望†,†† 柳澤 政生†,††† 大附 辰夫†,††

† 早稲田大学理工学研究科 情報・ネットワーク専攻
†† 早稲田大学基幹理工学部 情報理工学科
††† 早稲田大学基幹理工学部 電子光システム学科
〒 169-8555 東京都新宿区大久保 3-4-1
Tel: 03-3209-3211(5716), Fax: 03-3204-4875
E-mail: †hiura@yanagi.comm.waseda.ac.jp

あらまし 本稿ではアプリケーションプロセッサ向けハードウェア/ソフトウェア (HW/SW) 協調設計システム「SPADES」におけるプロセッサのカーネル記述自動生成手法について提案する．近代の組み込みシステムの中枢を担うアプリケーションプロセッサには低コスト，低面積，高性能でかつ短期間の生産が求められている．アプリケーションプロセッサの性能をあげる主な手段として，SIMD 型演算器や MAC 演算器，ハードウェアループユニットやアドレッシングユニット，Y データメモリといったハードウェアユニットを付加することがあげられ，要求に応じてこれらハードウェアユニットを付加できることが重要になる．提案手法では，プロセッサコアを構成する各々の機能を実現する部分に分割し，ハードウェアユニットに対応可能な個々の機能部分を生成させる．生成された機能部分の各記述を併合することで，プロセッサコアの記述を生成する．本手法により生成された VHDL の平均 9%ほどの XML 記述で，プロセッサコアの VHDL 記述をおおよそ 3 秒以内で生成することができた．
キーワード アプリケーションプロセッサ，HDL 自動生成，HW/SW 協調設計，SPADES

A Processor Kernel Generation Method for Application-specific Processors

Toshihiro HIURA†, Shunitsu KOHARA†, Nozomu TOGAWA†,††, Masao YANAGISAWA†,†††, and Tatsuo OHTSUKI†,††

† Major in Computer Science, Waseda University
†† Dept. of Computer Science and Engineering, Waseda University
††† Dept. of Electronic and Photonic Systems, Waseda University
3-4-1 Okubo, Shinjuku, Tokyo 169-8555, Japan
Tel: +81-3-3209-3211(5716), Fax: +81-3-3204-4875
E-mail: †hiura@yanagi.comm.waseda.ac.jp

Abstract This paper proposes a processor kernel generation method for HW/SW co-design system named SPADES. SPADES is a system to synthesize processor cores specialized in application automatically. Low cost, small area, high performance and high productivity are required for application-specific processors in embedded systems. One of the effective methods to improve the processor performance is to integrate some hardware units such as SIMD functional units, MAC functional units, hardware loop unit, addressing unit, extra data memory, and it is important to select the appropriate hardware units depending on each target application. In our work, we divide the application-specific processor into some functional parts which are customized for additional hardware units, and our method generates and merges them. The description of the processor core is composed of each description of the function parts. In the experimental results, the VHDL descriptions of the processor cores can be generated in 3 seconds.
Key words Application-specific Processor, HDL Generation Method, HW/SW co-design, SPADES

1. まえがき

携帯電話に代表されるような近年の小型電気機器には，高性能な特定のアプリケーション処理に強い SoC(System On a Chip) が組み込まれている．SoC に代表される特定用途向けの組み込みシステムは面積・コストの厳しい設計制約，そして高い処理能力が求められている．その一方で，製品の市場における生存時間はますます短くなってきており，設計者は要求性能を満たすシステム設計を短期間で行うことを余儀なくされてきている．このような設計制約や開発時間の短縮を同時に行うための手段として，ターゲットアプリケーションからの要求仕様を満たすアプリケーションプロセッサを自動で設計・生成することが有効な手法であると考えられる．

開発効率を考慮したアプリケーションプロセッサの開発方式には，プロセッサの仕様記述から生成する方式と，テンプレートを元にプロセッサを要求仕様に合わせて開発する方式の大きく二つに分けられる．プロセッサ仕様記述から生成する方式の例としては LISA [1] や ASIP Meister [2] 等が挙げられる．これらの方式はプロセッサ用に設けられた独自の記述を用いることで，生成対象のプロセッサモデル範囲内において要求に応じた柔軟なプロセッサを設計することができる．またいずれの手法もパイプライン構成やアーキテクチャのタイプなど，高速化・柔軟性に考慮があり，アプリケーションに特化したプロセッサ設計に適している．従来のように HDL を直接記述するより大幅な設計効率の改善を図れるが，仕様記述を学習・記述しなければならない点や，アーキテクチャを探索しなければならない点を考慮すると，結果として設計工数が肥大化する恐れがある．またテンプレートを元に開発する方式の例としては MeP [5] や Xtensa [8], [9] が挙げられる．テンプレートを基にターゲットアプリケーションに適したプロセッサを設計することで大幅な設計時間の短縮が図れ，拡張命令などの記述は必要であるが，前者に比べると記述量は少なくてすむために設計工数が肥大化する可能性は低い．しかしながらテンプレートが決まっているためにアーキテクチャ上の制約は多く，アーキテクチャの探索をしなければならないデメリットも存在する．

どちらの方式も一長一短があるが，いずれにしても設計するにはアーキテクチャ探索が必要となる．アプリケーションプロセッサはハードウェア (HW) とソフトウェア (SW) が独立してではなく，協調して動作をすることから，設計手法には HW と SW の設計を両者のトレードオフを考慮しながらアーキテクチャを探索し，並行して設計を行うプロセッサコア向けのハードウェア/ソフトウェア協調設計 (以下，HW/SW 協調設計) が有効であると考えられる．マルチメディアアプリケーション処理を考慮した場合，乗加算器 (MAC) や SIMD 型演算器のみならず，HW ループユニットやアドレッシングユニットを付帯させることもできるプロセッサカーネルの自動設計が不可欠となる．

そこで本稿ではアプリケーションプロセッサコア向けの HW/SW 協調設計システム SPADES(注1) [7] におけるアプリケーションプロセッサのカーネル記述自動生成手法を提案する．本手法はプロセッサコアを構成する各々の機能を実現する部分にわけ，個々の機能部分を生成させ，生成させた各記述を併合することでプロセッサカーネルの記述を生成する．

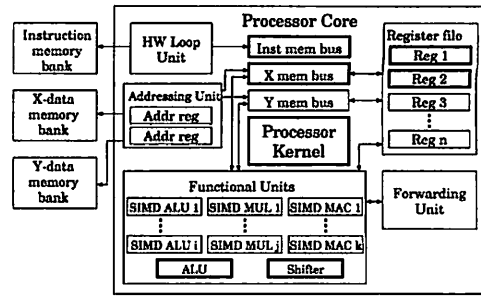

図 1　プロセッサコア内の機能分担

本稿は以下のように構成される．第 2 章では HW/SW 協調設計システム SPADES の概要を述べる．第 3 章ではアプリケーションプロセッサのカーネル生成手法について示し，第 4 章では計算機上で実装したシステムから提案手法の有効性を示す．

2. HW/SW 協調設計システム SPADES

SPADES は Packed SIMD 型演算 (以下，SIMD 型演算) に対応したプロセッサコア向けの HW/SW 協調設計システムである．SIMD 型演算とは，1 つの b ビット演算器を用いて n 個の $b/n(=k)$ ビットデータを 1 命令で実行する演算のことである．SIMD 型演算を用いることで，画素並列で実行することができるため，画像処理アプリケーションを高速で実行可能となる．本章では SPADES のターゲットアーキテクチャと本システムの概要について述べる．

2.1 ターゲットアーキテクチャ

SPADES プロセッサコアはパイプラインプロセッサ方式を取り，命令形式は VLIW 命令を用いる．さらに命令メモリと複数のデータメモリを外部メモリとしてもつハーバードアーキテクチャを採用している．プロセッサコア内の機能分担について，図 1 に示す．SPADES が生成対象とするプロセッサコアのアーキテクチャの一例を図 2 に示す．図 2 は 1 並列 5 段パイプラインのモデルである．

プロセッサコアは中枢部となるカーネル，演算器 (Functional Units)，メモリ用バス，レジスタファイルとその他の HW ユニットで構成される．ここでいう HW ユニットとは，演算器や HW ループユニット，アドレッシングユニットやフォワーディングユニットなどプロセッサコアの動作を補助するユニットを指す．プロセッサとして機能するためには，カーネルと命令メモリ，そしてデータメモリ用バス，ALU，シフタは最低一つ，レジスタファイルは最低二つ必要になる．基本的にプロセッサコアはプロセッサカーネルにいくつか HW ユニッ

(注1)：SPADES とは System for Processor Architecture Design with Estimation - type SIMD の略である．

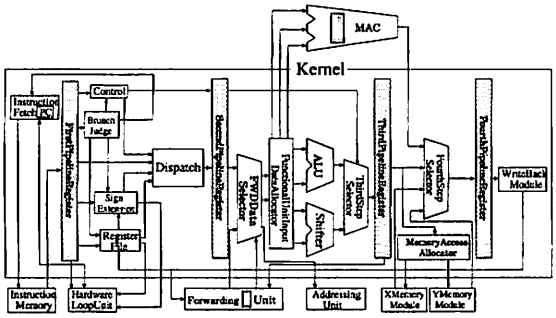

図 2　プロセッサコアアーキテクチャの一例

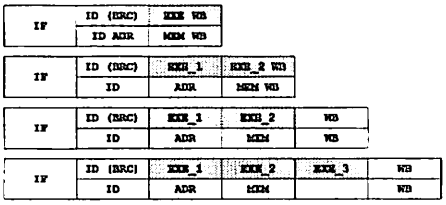

図 3　パイプライン構成モデル

トを付加することで構成される．

2.1.1　プロセッサカーネル

プロセッサカーネルとは，HW ユニットを制御するために必要な HW とする．プロセッサカーネルは複数命令を並列実行可能な VLIW 型で，パイプライン段数，並列度ともにアプリケーションに応じた適切な構成を得ることができる．プロセッサカーネルは，図 2 の kernel の内部に該当し，プロセッサにおける心臓部となる．

2.1.2　HW ユニット

マルチメディアアプリケーション処理にあたっては，SIMD 演算やループ処理，効率よくデータアクセスができることが処理性能を向上させると考えられる．プロセッサコアが持つことが可能な HW ユニットとして，演算器，SIMD 型演算器，Y データメモリバス，レジスタファイル，アドレッシングユニット，HW ループユニットおよびフォワーディングユニットがある．付加される HW ユニットの種類および数は可変であり，アプリケーションに応じて，必要とされる HW 構成に適した HW ユニットをプロセッサカーネルに追加することが可能である．

2.1.3　パイプライン構成モデル

アプリケーションプロセッサのとりうるパイプライン構成をパイプライン構成モデルと定義する．パイプライン段数 6 段までのパイプライン構成モデルを図 3 に示す．最低パイプライン段数は 3 段とする．各パイプライン構成モデルでは，まず第 1 ステージで命令のフェッチを行い，第 2 ステージでは命令デコードを行う．演算命令の場合，第 3 ステージ以降はパイプラインごとに対象になる演算が分割して行われ，最後にレジスタ書き込みが行われる．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及びレジスタ書き込みを行う．また分岐命令は第 3 ステージで分岐先のアドレス計算が行われる．

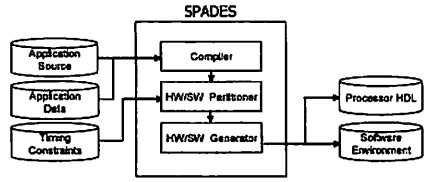

図 4　HW/SW 協調合成システム：SPADES

パイプライン 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 段づつ増える．

2.2　SPADES の概要

SPADES のシステム構成を図 4 に示す．SPADES 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実行時間制約を入力とし，Packed SIMD 型演算の可能なプロセッサコアの HW 記述，プロセッサコア上で動作するオブジェクトコードおよび SW 環境を出力する．SPADES は実行時間制約を満たす範囲で面積最小のアプリケーションプロセッサを合成することを目的としている．

システムは並列化コンパイラ，HW/SW 分割，HW/SW 生成から構成される．現在本システムは CoDaMa フレームワーク [4] を利用しており，それぞれの系を介す言語は XML となっている．アプリケーションプログラムの CDFG(Control Data Flow Graph) を XML で記述したものを Program XML(Prog-XML) と呼び，プロセッサコア構成を XML で記述したものを Architecture XML(Arch-XML) と呼ぶ．

2.2.1　並列化コンパイラ

入力のアプリケーションプログラムとアプリケーション解析によって得られた情報から，必要とされる HW ユニットを全て付加した状態の仮想的なプロセッサコアを考え，その仮想プロセッサコア上で動くアプリケーションプログラムの並列性を最大限に抽出し，最速処理が可能となるようにスケジューリングされた Prog-XML を出力する．

2.2.2　HW/SW 分割

はじめに，並列化コンパイラより出力された Prog-XML を実行することのできるように HW ユニットを付加する．得られるプロセッサコア構成では，アプリケーションを最速処理することが可能であるが，面積は大きくなる．一般的に HW ユニットを削減すると，プロセッサコアの面積は減少するが，アプリケーションの実行時間は増加するため，HW/SW 分割では HW による実現部を徐々にソフトウェアに代替していくことで，プロセッサコアの面積削減を図る．実行時間制約を満たす間この処理を繰り返し行い，時間制約を満たす中で面積最小のプロセッサコア構成を決定し，Prog-XML と Arch-XML を出力する．

2.2.3　HW/SW 生成

HW 生成は Arch-XML からプロセッサコア構成を抽出し，プロセッサコアの HDL を出力する．SW 生成は Prog-XML および Arch-XML からプロセッサコア上で動作するオブジェク

トコード，コンパイラ，アセンブラ，シミュレータを生成する．

3. HW 生成フレームワーク

SPADES の構築要素である HW 生成は，Arch-XML を入力として HW 記述言語の一つである VHDL を生成する．生成される VHDL パイプライン段数や並列度などのプロセッサ構成をアプリケーションと遅延制約に応じて可変な HW 自動生成システムの実装を目標としている．そこで HW 生成方法として HW 生成フレームワークに基づくプロセッサカーネル自動生成手法を提案する (図 5)．

3.1 プロセッサモジュール

基本的なプロセッサコアは命令フェッチ，命令デコード，キャッシュアクセスや演算などの動作を行う必要があり，それは提案プロセッサコアも例外ではない．それぞれの動作は，図 3 のパイプライン構成モデルを示したように，パイプライン化した各ステップ毎に割り振ることもできる．また，例えば命令デコード部では制御やデータ割り当てなどの機能を実現する単位，演算部では ALU やシフタなどの演算機能を実現する単位に分けることもできる．これら ALU やレジスタなど，プロセッサコアを構成する各々の機能を実現する部分をプロセッサモジュールと定義する．プロセッサコアは各々のプロセッサモジュールの連携とも言えるので，提案手法ではプロセッサモジュールに基づいてプロセッサコア生成を行う．

プロセッサモジュールはダイナミックモジュールとスタティックモジュールの二種類に分けることができる．

3.1.1 ダイナミックモジュール

プロセッサモジュールには制御部やパイプラインレジスタなどのようにパイプライン段数や命令発行数などのプロセッサコアの構成をパラメータとして，各々の入出力などの構成が変わるものがある．このようにプロセッサコアの構成によって入出力が変わるモジュールをダイナミックモジュールと定義する．

ダイナミックモジュールは，プロセッサコア構成に左右されるため，各構成のモジュールを用意するより，状況に応じて生成できるほうが望ましい．そこで HW 生成フレームワークでは各モジュールの生成機能を用意する．

3.1.2 スタティックモジュール

演算器などのように，プロセッサモジュールにはパラメータに構成が左右されないものもある．このようにプロセッサコアの構成によって入出力が変わらないモジュールをスタティックモジュールと定義する．

これらはスタティックモジュールは，各々の VHDL 記述のテンプレートを用意し，必要に応じて使用するライブラリとして取り扱うようにする．

3.2 HW ジェネレータ

HW ジェネレータは Arch-XML を読み込み，プロセッサモジュール生成系にモジュールを生成させて，プロセッサコア記述を生成させる．いわば HW 生成フレームワークにおけるコントローラの役割を担う．HW ジェネレータは XML ハンドラ，モジュールハンドラ，HDL マージャの三つから成り立っている．

XML ハンドラ

Arch-XML からプロセッサ構成や使用する演算器などを読み込み，各プロセッサモジュール生成系に渡すパラメータを生成する．

モジュールハンドラ

プロセッサモジュール生成系にパラメータを与え，各プロセッサモジュールを生成させる．また各パラメータから必要となるモジュールを決定する．

HDL マージャ

生成されたプロセッサモジュールとスタティックモジュールの中から，モジュールハンドラが決定したプロセッサモジュールを読み出し，信号線などを併合させてプロセッサ記述を生成する．

3.3 カーネル生成手法

マルチメディアアプリケーション処理には繰り返し演算をすることや，積和演算（乗加算）をすることが多く登場するので，MAC 演算器や SIMD 型演算器，HW ループ等の HW ユニットがあると有効であるといえる．また大量のデータに柔軟に効率よくアクセスすることが求められるのでアドレッシングユニット，そして Y データメモリバスがあると有効であると考えられる．これらはプロセッサカーネルが制御を行うので，プロセッサコアにこれらの HW ユニットを付加させるようにするためには，プロセッサカーネルが要求に応じて入出力をはじめ，変化できることが望ましい．そのため，本提案手法では，プロセッサカーネルの構成をこれらの HW ユニットに対応できるようにする．

3.3.1 MAC 演算器対応

積和演算は，以下のような式で定義される．

$$R3 = R1 \times R2 + R3 \quad (1)$$

演算器は基本的に 2 入力演算を対象としているが，積和演算を行うために，MAC には 3 つの入力が必要になる．プロセッサカーネルは各スロット 2 入力には対応しているので，MAC が存在する場合には入力の信号線をあと 1 つ必要となる．

提案手法ではプロセッサコアに搭載する MAC の数を抽出し，それをパラメータとすることでこの演算が可能となるカーネル構成が出力される．

3.3.2 SIMD 型演算器対応

SIMD 型命令セットは，演算オペレーションコードに加えて，梱包数やシフト方向，シフト量などを考慮しなければならない．SPADES では役割の異なるビットフィールドに分けて，SIMD 命令用のオペコードフォーマットを考えて，それに従い割り当てを行うこととする．オペコードフォーマットを図 6 に示す．提案手法では使用する SIMD 命令を読み込んで，オペコードフォーマットを最小ビットになるように設定する．またデコード部も生成されたオペコードフォーマットに対応可能なカーネル構成が出力される．

3.3.3 HW ループユニット対応

HW ループユニットはループ処理の終了を HW によって検出することで命令の読み込みアドレスを自動的に変更する．この操作により，指定された命令メモリアドレス間の命令をパイプラインハザードを防ぐための NOP 命令なしで一定数回実行

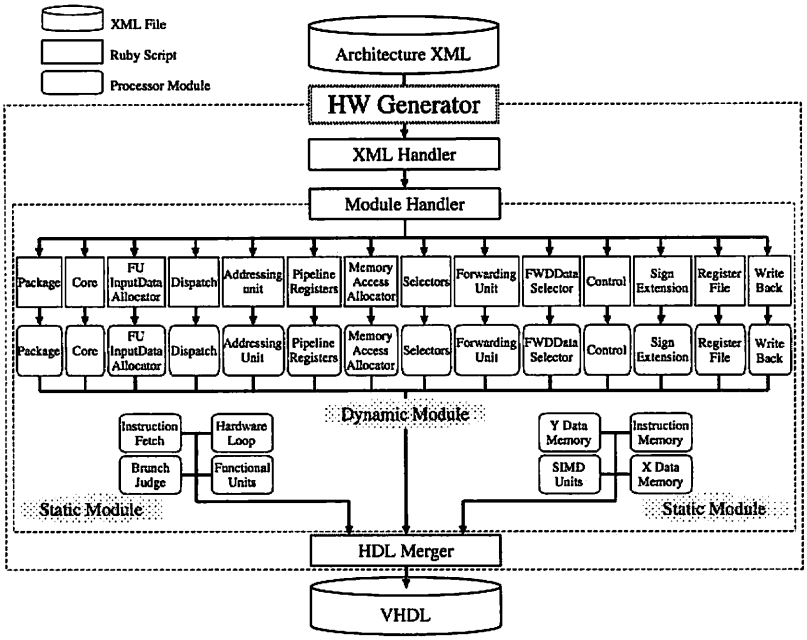

図 5　ハードウェア生成フレームワーク

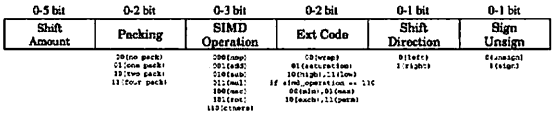

図 6　SIMD 命令用オペコードフォーマット

することが可能である．

HW ループユニットは命令フェッチの段階で仕事を行うので，提案手法では HW ループユニットが必要となる場合，命令フェッチの入出力を変更して，HW ループユニットに対応することが可能である．また多重ループ対応も可能となっている．

3.3.4　アドレッシングユニット対応

アドレッシングユニットはアドレス計算用のレジスタ及び演算機能を持つため，アドレス計算をするために通常の演算ユニットを用いる必要がなく，メモリアクセスのアドレス計算を効率よく行うことができる．様々なアドレッシングモードがあるので，柔軟で効率的なデータアクセスが可能となる．アドレッシングモードについて表 1 に示す．

提案手法では，アドレッシングユニットの有無をパラメータとし，カーネル仕様を対応させる．さらにアドレッシングユニットを使用すると X データメモリと Y データメモリの並列アクセスも可能になる．並列アクセス命令が必要になる場合のカーネル仕様も，提案手法は対応可能である．

3.3.5　Y データメモリ対応

プロセッサコアは X データメモリを持ち，プロセッサカーネル内に X データメモリ用バスを持つ．これに加えてさらに Y データメモリ用バスを持つことができる．Y データメモリを使用する場合，X データメモリと Y データメモリを同時に使用することで，データメモリから並列に 2 個の値をロード・ストアをすることが可能となる．このことにより，データメモリが一つの場合と比較して，高速なメモリアクセスが可能となり，結果としてアプリケーションの実行が高速化される．

提案手法では，Y データメモリが使われる場合，入出力やバスを生成することが可能である．

表 1　アドレッシングモード

アドレッシングモード	説明
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 incriment	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 decriment	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	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．
bit reverse	アドレスレジスタ値の上位-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．

4.　計算機実験

提案手法により，XML から短時間でアプリケーションプロセッサコアの VHDL が正しく生成されるかを確認するため，計算機上で HW 生成フレームワークを実装し，いくつかのプロセッサを設計した．

4.1　実験環境

実験環境は

OS　Debian GNU/Linux sarge

CPU　Intel Pentium4 2.80GHz

Memory　2GB

表 2 プロセッサコア記述生成結果

ラベル	並列度	パイプライン段数	合令長	HW ループ	アドレッシングユニット	汎用レジスタ	演算器	遅延 [ns]	面積 [μm^2]	XML [kbyte]	VHDL [kbyte]	XML /VHDL[%]	生成所要時間 [s]
A	1	3	13	×	×	32	ALU1,SFT1 SIMDALU1	1.92	49340	18.57	99.22	18.7	2.02
B	1	4	10	×	C	16	ALU1,SFT1	1.60	40694	18.77	141.1	13.3	2.39
C	1	5	12	×	×	16	ALU1,SFT1 MAC1	2.73	73244	18.44	130.9	14.1	2.34
D	1	6	6	nest3	×	4	ALU1,SFT1 MUL1	2.02	74572	18.44	159.2	11.6	2.57
E	2	3	9	×	O	16	ALU1,SFT1 MUL1,MAC1	5.63	112586	19.84	219.8	9.03	2.25
F	2	5	7	nest3	×	64	ALU2,SFT2	1.82	129332	18.51	219.8	8.42	2.58
G	2	6	8	×	O	4	ALU1,SFT1 MAC1	3.45	103668	19.34	254.2	7.61	2.52
H	4	4	11	nest3	×	16	ALU1,SFT1 MUL1	2.04	133864	19.01	329.9	5.76	2.58
I	4	5	12	nest2	O	8	ALU2,SFT1 MUL1,MAC1	2.78	134925	20.28	427.2	4.75	2.76
J	4	6	8	nest1	O	4	ALU2,SFT2	3.12	115321	19.59	421.4	4.65	2.75
平均												9.79	2.48

表 3 アドレッシングユニットのパラメータ

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	O	O	O	×	×	×
E	1	4	O	O	×	×	O	×
G	1	6	O	O	×	×	×	O
I	2	2	O	O	×	×	×	×
J	1	8	O	O	O	O	O	O

である．提案手法はオブジェクト指向スクリプト言語 Ruby [6] を用いて実装を行った．また 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(注2)(CMOS90nm) の設計ルールを用いた．計算機上で実験した結果を表 2 に示す．表 2 上の生成時間の値は，3 回生成させてその平均時間を取ったものである．また，アドレッシングユニットのパラメータについて表 3 に示す．

さらにいくつかパラメータを変更させて，プロセッサコア記述を生成したときの XML と VHDL 記述の変更量を示した結果を表 4 に示す．

4.2 提案手法の評価

本手法を用いることで VHDL 記述の平均 9%ほどの Arch-XML からアプリケーションプロセッサ記述を生成することができた．その際に要した時間は 1 つのプロセッサコアにつき平均 2.5 秒であった．プロセッサコアの VHDL 記述はおおよそ数千行に渡るものであるが，その記述を生成可能な Arch-XML の記述は平均 9%であるため，本手法を用いることで，設計者はおおよそ 10 分の 1 の労力でプロセッサコアの記述が可能となり，設計効率の大幅な改善につながるといえる．

文献 [3] の小林らの手法では，少量の仕様記述変更でプロセッサ VHDL 記述の変更ができることを示すために，2 スロットの VLIW プロセッサのシフタ数を変更した場合，25 行の仕様記述変更で約 450 行の VHDL 記述変更が行えることを示している．そこで本提案手法に対しても同様の検証を行った．2 並列 5 段パイプライン，ALU が 2 つ，シフタが 1 つ付加されているプロセッサを基準として，シフタ数，並列度，パイプライン段数，MAC 演算器，HW ループユニットについて変更を行った (表 4)．ここで仕様記述変更量を VHDL 記述変更量で割ったものを記述変更率とすると，小林らの手法の記述変更率は 5.56%となることに対し，提案手法の記述変更率は 0.32%～1.96%となる．この結果から提案手法は，記述変更率の観点から小林らの手法に比べて，VHDL 記述のレベルに多少の差異はあるが，効率的にプロセッサコア VHDL 記述を変更できるといえる．

表 4 パラメータ変更に伴う XML 記述と VHDL 記述の変更量．

パラメータ	XML 記述 (行)	VHDL 記述 (行)	XML 記述 変更量 (行)	VHDL 記述 変更量 (行)
基準プロセッサ	202	4099	-	-
シフタ数 1 → 2	203	4191	1	92
並列度 2 → 4	209	7008	10	2909
パイプライン段数 5 → 6	202	4404	1	305
MAC 演算器 無→有	213	4659	11	560
HW ループ 無→有	203	4784	9	685

5. むすび

本稿では HW/SW 協調設計システム SPADES におけるアプリケーションプロセッサのカーネル記述自動生成手法を提案した．本手法を用いることで，アプリケーションプロセッサの性能をあげる HW ユニットが付加できるプロセッサカーネルの自動生成が可能になり，設計効率が大幅に改善できることを述べた．今後はシミュレータの自動設計が必要となる．

(注2)：STARC90nm 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

文献

[1] A.Hoffmann, O.Schliebusch, A.Nohl, G.Braun, O.Wahlen and H.Meyr, "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA," *in proc. IEEE/ACM ICCAD*, pp. 625–630, 2001.

[2] 今井 正治, 谷口 一徹, 武内 良典, 坂主 圭史 "コンフィギュラブル・プロセッサ開発環境 ASIP Meister, " 信学技報, VLD2006-5, Vol. 106, No. 31, pp. 25-30, 2006.

[3] 小林 悠記, 小林 真輔, 坂主 圭史, 武内 良典, 今井 正治, "コンフィギュラブル VLIW プロセッサの HDL 生成手法," 情報処理学会論文誌, Vol.45, pp. 1311–1321, 2004.

[4] 小原 俊逸, 史 又華, 戸川 望, 柳澤 政生, 大附 辰夫, "XML をベースとした CDFG マニュピレーションフレームワーク:CoDaMa," 信学技法, VLD2006-97, pp. 19–24, 2006.

[5] MeP Microcomputer Toshiba Semiconductor Company, http://www.MePcore.com/

[6] オブジェクト指向スクリプト言語 Ruby, http://www.ruby-lang.org/

[7] 大東 真崇, 小原 俊逸, 戸川 望, 柳澤 政生, 大附 辰夫, "SIMD 型プロセッサコアを対象としたハードウェア/ソフトウェア分割フレームワーク," 信学技法, VLD2006-120, pp. 7–12, 2006.

[8] R.E.Gonzalez, "Xtensa: a configurable and extensible processor," *IEEE Micro*, Vol.20, pp. 60–70, 2000.

[9] Tensilica Inc. Xtensa Microprocessor, http://www.tensilica.com/

[10] 山崎 大輔, 小原 俊逸, 戸川 望, 柳澤 政生, 大附 辰夫, "SIMD プロセッサコアの面積/遅延見積もり手法," ESS-2007, pp. 233–240, 2007.