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

2006－SLDM－127 (30)
2006／11／30

アプリケーションプロセッサのフォワーディングユニット最適化手法

日浦 敏宏† 小原 俊逸† 史 又華† 戸川 望† 柳澤 政生†
大附 辰夫†

† 早稲田大学理工学部コンピュータ・ネットワーク工学科
〒 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 特定用途向けプロセッサは近年小型，低コスト，高性能に加え，設計時間の短縮が求められている．我々の提案しているアプリケーションプロセッサコア向け HW/SW 協調設計システム SPADES は，アプリケーションの実行時間制約を満たす範囲で最小面積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する．SPADES の面積削減手法は，不要な HW ユニットの削減と命令セットの変更をベースとしているが，より面積の小さいプロセッサを合成するには，命令セットの変更を伴わないプロセッサアーキテクチャレベルでの最適化手法が有効であると考えられる．本稿ではフォワーディングユニットについて着目する．一般的なフォワーディングユニットは，プロセッサのパイプライン段数やスロット数が増えると比較対象のデータ数が増え，クリティカルパスになりやすい．そこで従来のフォワーディングユニットよりも 1 ステージ分早い段階で判定を行う先見判定型のフォワーディングユニットが提案されている．一般的な型と先見判定型は面積/遅延でトレードオフの関係にあり，提案手法はプロセッサアーキテクチャパラメータから最適な方式を選択し HDL を自動生成する．それぞれの方式のフォワーディングユニット HDL 記述自動生成システムを実装し，様々なパラメータを与えてフォワーディングユニットの HDL 記述の生成を行い，それぞれの面積/遅延にトレードオフの関係にあることが示された．

キーワード アプリケーションプロセッサ，フォワーディングユニット，HW/SW 協調設計，SPADES

A Forwarding Unit Optimization Method for Application Processors

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

† Dept. of Computer Science, 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 To meet the requirements in application specific processor designs, such as area, cost, performance and design time, we have been developing a HW/SW co-design system, called SPADES, which can generate an application specific processor with minimum area on the constraint of the execution time of an application. In SPADES, we reduce the area by reducing unnecessary HW unit and then change the instruction set. However, to change the instruction set will affect the processor architecture. On the other hand, forwarding unit is easy to become the critical path in processors when the processor architecture becomes complex. Thus in this paper, we focus on the forwarding unit for optimization by making a tradeoff between area and delay while without any changes in the instruction set. We also propose a new forwarding unit architecture, called foresight judgment type forwarding unit, which can be incorporated into SPADES to generate HDL description automatically without any knowledge of our system. Experimental results show that the proposed method is more suitable in HW/SW co-design systems to generate the optimized forwarding unit.

Key words Application Processor, Forwarding unit, HW/SW co-design, SPADES

1. まえがき

小型の電気機器に組み込まれる特定用途向けのシステムは小型で，低コスト，そして高い処理能力と設計時間の短縮が求められている．それを満たす解にはアプリケーションに特化したプロセッサ (以下，アプリケーションプロセッサ) が挙げられる．アプリケーションプロセッサはハードウェアとソフトウェアが独立してではなく，協調して動作をすることから，設計手法の一つに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) が挙げられる．

プロセッサコア向けの協調設計に関する研究は，文献 [2], [5], [8], [11] などに，また，プロセッサの自動合成に関する研究は文献 [1], [4], [9], [12] などに報告がある．いずれの手法も短期間で効率よく高性能なプロセッサ設計を行うことを目的としており，プロセッサの高性能化を図るためにパイプライニングを考慮した研究も多い．パイプライニングは動作周波数の向上が見込め，高性能化には有効な手法であるが，それに伴いパイプラインハザードの問題も発生する．パイプラインハザードはデータハザード・制御ハザード・構造ハザードが挙げられ，それを解決する技術もパイプライニングと同時に求められる．本稿ではその中でもデータハザードに着目する．

データハザードは処理するデータの前後に依存関係があった場合に生じるハザードであり，その回避には発生したら先の演算が終わるまで次の演算を待たせる必要がある．これでは動作周波数向上を図ったパイプライニングの利点を生かしきれない．そこでフォワーディングユニットというハードウェアを追加させることでデータを先送りし，解決する方法が挙げられる．

フォワーディングユニットはデータハザードの原因となりうるレジスタアドレスを入力とし，そのレジスタアドレスがデータハザードを引き起こす場合，必要とされるデータが送られるように制御信号線を送る逐次判定型が挙げられる [10]．しかしこれではアプリケーションプロセッサのパイプライン段数の増加やスロット数の増加に伴いデータの比較対象が増え，結果としてフォワーディングユニットを通るパスがプロセッサのクリティカルパスとなる可能性が考えられる．そこで内部にレジスタを挿入し，前者の方策より一段階前にデータの依存関係を判定し，次クロックで制御信号とフォワーディングのデータを送るという先見判定型の方式が考えられる [13]．このことにより，動作周波数の向上が見込めるが，内部にレジスタを挿入する分，面積が増えることが考えられる．しかし逐次判定型と先見判定型のフォワーディングユニットは命令セットレベルでは影響がないために，置き換えても命令セットの変更に及ぶことはない．

著者の知る限り，既に発表されているプロセッサ生成手法では，フォワーディングユニット生成は行えるが，面積/遅延を最適化したフォワーディングユニット生成に関しては提案されていない．プロセッサ面積をより小さくし，遅延時間をより短くすることは小規模化，高性能化が求められる組み込みシステムにおいて有効な手段である．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], [13] 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おり，その面積削減手法は，不要な HW ユニットの削減と命令セットの変更をベースとしている．しかし，より小面積のアプリケーションプロセッサを合成するには，命令セットの変更を伴わないプロセッサアーキテクチャレベルでの最適化手法が有効であると考えられる．

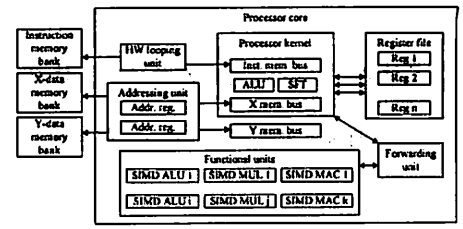

図 1 プロセッサコアアーキテクチャモデル

以上の背景からプロセッサ構成におけるフォワーディングユニットの面積最適化生成手法を提案する．提案手法は SPADES のアプリケーションプロセッサの HDL 記述自動生成をする HW 生成系の一部であり，命令セットを変えることなく，最適なフォワーディングユニットの生成が可能である．

本稿は以下のように構成される．2 章では HW/SW 協調設計システム SPADES の概要を述べる．3 章ではフォワーディングユニットの面積最適化生成手法について示し，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. SPADES

SPADES は Packed 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．本章では SPADES のターゲットアーキテクチャと概要について述べる．

2.1 ターゲットアーキテクチャ

SPADES が生成するプロセッサコアのアーキテクチャモデルを図 1 に示す．SPADES プロセッサコアはプロセッサカーネルにいくつかハードウェアユニットを付加することで構成される．

プロセッサカーネル

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られるスケーラビリティを持つ．プロセッサカーネルはプロセッサとして動作する必要最低限のハードウェアであり，X データメモリ用バス，レジスタファイル，バレルシフタ及び ALU を一つずつ持つ．

ハードウェアユニット

プロセッサコアが持つことが可能なハードウェアユニットとして，(1) SIMD 型演算器 (2) Y データメモリバス (3) レジスタファイル (4) アドレッシングユニットおよび (5) ハードウェアループユニットがある．アプリケーションに応じたハードウェア構成が必要とした分のハードウェアユニットをプロセッ

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

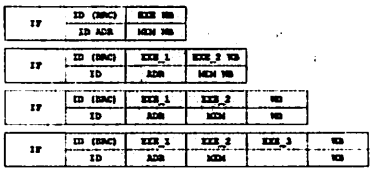

図 2　パイプライン構成モデル

サカーネルに追加できる．

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 2 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．

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 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ハードウェア記述，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ソフトウェア環境を出力する．

Packed SIMD 型演算 (以下，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．このとき n を細包数と呼ぶ．SIMD 型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で高速に実行することができる．SIMD 型演算を実行する命令を Packed SIMD 型命令 (SIMD 命令)，SIMD 型演算を実行する演算ユニットを Packed SIMD 型演算ユニット (SIMD 型演算ユニット) と呼ぶ．SIMD 型演算を用いた [3], [6] 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となっている．SPADES は以下に示す 4 つの系から構成される．

並列化コンパイラ系

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から得られた情報から，必要とされるハードウェアユニットを全て付加した状態の仮想的なプロセッサを考え，その仮想プロセッサ上で動くアプリケーションの持つ最大限の並列性を抽出したアセンブリコードを生成する．

HW/SW 分割系

はじめに，並列化コンパイラより出力されたアセンブリコードを実行することのできるプロセッサコアを構成する．このプロセッサコア構成はアプリケーションの実行時間を短くすることが可能であるが，プロセッサコアの面積は大きくなる．HW/SW 分割ではハードウェア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得る．

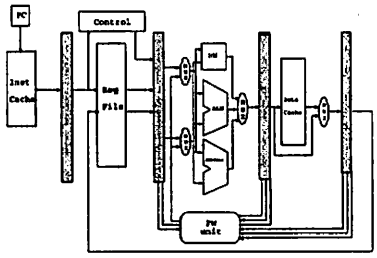

図 3　逐次判定型フォワーディングユニット

SW 生成系

HW/SW 分割系からのプロセッサアーキテクチャ，命令セット，アセンブリコードより，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．

HW 生成系

HW/SW 分割系からのプロセッサアーキテクチャとアーキテクチャテンプレートからプロセッサコア構成を遅延時間制約を超えない範囲で最適化し，プロセッサコアのハードウェア記述を生成する．

3.　フォワーディングユニットの最適化生成

本章では，SPADES が対象としているアプリケーションプロセッサに対応したフォワーディングユニット最適化手法を提案する．フォワーディングユニットのアプリケーションプロセッサにおける役割と動作を説明し，面積最適化とフォワーディングユニット自動生成システムを提案する．

3.1　フォワーディングユニットの動作

パイプラインプロセッサにおいてデータハザードが発生したとき，データを先送りすることでデータハザードを解決することができる．データハザードを解決するにあたってフォワーディングユニットが挙げられる．フォワーディングユニットは一般的には逐次判定型が用いられるが，このフォワーディングユニットを通るパスがクリティカルにならないよう先見判定型も提案されている [13]．

逐次判定型

図 3 に逐次判定型フォワーディングユニットのプロセッサにおけるブロック図を示す．図 3 の例はプロセッサカーネルが 1 並列 5 段パイプラインの場合である．逐次判定型は

（1）　演算ステージ以降のハザードが起こりうるデータ，レジスタ書き込み信号とアドレスを入力

（2）　依存関係を調べる後続のデータのアドレスを入力

（3）　依存関係の判定

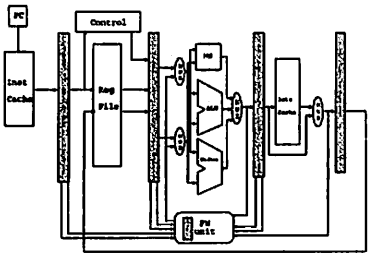

図 4　先見判定型フォワーディングユニット

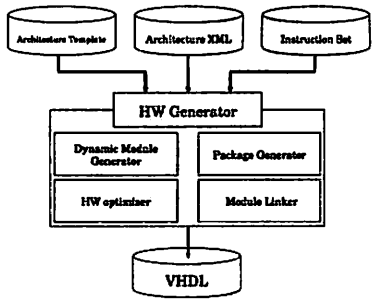

図 5　HW 生成系

（4）　演算ステージのマルチプレクサの制御信号とデータを出力

という一連の動作を行い，データハザードの解消を図るものである．パイプライン段数と並列度が増えた場合，(1) の入力は増え，依存関係の判定を行う (3) での比較対象が増えるために，フォワーディングユニットを通るパスがプロセッサにおけるクリティカルパスになる可能性がある．

先見判定型

図 4 に先見判定型フォワーディングユニットのプロセッサにおけるブロック図を示す．図 4 の例はプロセッサカーネルは 1 並列 5 段パイプラインの場合である．先見判定型は

（1）　演算ステージ以降のハザードが起こりうるデータ，レジスタ書き込み信号とアドレスを入力

（2）　依存関係を調べる後続のデータのアドレスを入力

（3）　依存関係の判定

（4）　値とレジスタアドレスをフォワーディングユニット内部のレジスタに格納

（5）　クロック同期で演算ステージのマルチプレクサの制御信号とデータを出力

という一連の動作を行い，データハザードの解消を図るものである．逐次判定型よりも，(1) (2) の入力が 1 ステージ分早く，また一度レジスタに格納したのち，クロック同期で制御信号とデータを出力することで，フォワーディングユニットを通るパスがプロセッサにおけるクリティカルパスになる可能性をさけるというものである．

しかしながら，内部にレジスタを挿入する分，面積の増加が見込まれる．したがって動作周波数に影響を及ぼさずに面積の小さいフォワーディングユニットを選ぶことが望ましい．

3.2　フォワーディングユニット面積最適化

既に述べたように，先見判定型と逐次判定型は面積と遅延時間においてトレードオフの関係にあることがわかる．本提案では，このトレードオフを利用し，HW/SW 分割系から得られるハードウェア構成において，HDL 記述に要する時間の短縮を図るだけでなく，フォワーディングユニットの面積の小さいほうを選び，HDL 記述の自動生成を行うものである．

HW のアーキテクチャ構成が変わることで，命令セットにも影響がでることが考えられ，アーキテクチャ構成の変更に伴った命令セットの変更が求められることが考えられる．しかし，これら 2 つのフォワーディングユニットはデータの先送りの動作自体が両方同じ動作をするので，ソフトウェアの観点からはまったく同じアーキテクチャ構成であるように捉えることができるため，命令セットの変更が不要になる．

3.2.1　HW 生成系における面積最適化

HW 生成系は HW/SW 分割系からの得られるプロセッサ構成と命令セット，アーキテクチャテンプレートを入力とし，アプリケーションプロセッサの HDL 記述を生成するものである．プロセッサ構成は XML で表現し，HW 生成系はその入力からプロセッサの各モジュール生成とパッケージ生成，面積最適化と上位記述生成を行う．HW 生成系と HW 生成系における面積最適化系（HW optimizer）の概要図を図 5 に示す．

3.2.2　提案する最適化手法

SPADES は時間制約を入力のひとつとしている．そのために HW/SW 分割系からの HW 生成系にわたる情報は動作周波数が条件を満たすものとなっている．そこで分割系においてはフォワーディングユニットは常に先見判定型を選ぶものとし，初期のプロセッサのフォワーディングユニットは面積が逐次判定型よりも大きいものとする．

フォワーディングユニットの遅延時間は，図 2 のパイプライン構成モデルにおける EXE_1 ステージに加えられるため，提案手法では，HW/SW 分割系からのプロセッサの遅延時間と EXE_1 ステージの遅延時間の差が，そのプロセッサ構成に適した逐次判定型フォワーディングユニットの遅延時間より大きければ，HW optimizer が逐次判定型，小さければ先見判定型の VHDL 記述を出力する．

3.3　フォワーディングユニット自動生成システムの概要

フォワーディングユニットの自動生成を行うためには次に挙げるプロセッサ構成の情報が必要になる．

- パイプライン段数 $s(s = 3, 4, 5\ldots)$
- スロット数 $n(n = 1, 2, 3\ldots)$
- 3 入力演算対応のスロット数 $t(t = 1, 2, 3\ldots)$

これに加えて，レジスタアドレスのビット数とデータのビット数に関する情報が必要になる．

パイプライン段数

パイプライン段数が増加すると，その段数分データハザードを判定し，先送りをするべき情報が増える．これらはフォワーディングユニットのポート数やアーキテクチャに大きく関わる重要な情報となる．

入力

パイプライン段数 s，スロット数 n，三入力情報 t
プロセッサのクリティカルパス情報

出力

フォワーディングユニット HDL 記述

Step1[入力] プロセッサのクリティカルパス情報，パイプライン段数 s，スロット数 n，三入力情報 t を読み込む．

Step2 パイプライン段数が 3 以下であれば終了，それ以外なら Step3 へ．

Step3 パイプライン段数 s，スロット数 n，三入力情報 t からフォワーディングユニットのポート数，動作の決定．

Step4 2 種類のフォワーディングユニット記述生成．HW optimizer に渡す．

Step5 プロセッサの遅延時間から全スロットの EXE_1 の中の最大遅延時間との差をとる．

Step6 その差が見積もられた逐次判定型の遅延時間より大きければ逐次判定型，小さければ先見判定型の選択．

Step7 [出力] 選ばれたフォワーディングユニットの VHDL 記述の出力．

図 6　フォワーディングユニット最適化生成フロー

スロット数

スロット数，つまりカーネルの並列度が増加すると，各々のスロットへ判定した情報を送るため先送りをするべき情報と先送り情報の出力先が増える．これもフォワーディングユニットのポート数やアーキテクチャに大きく関わる重要な情報となる．

3 入力演算対応のスロット数

プロセッサカーネルに組み込まれる演算器は基本的に二入力演算を対象としているが，ディジタル信号処理において積和演算（乗加算）はよく現れるので，積和演算ができるハードウェアユニットがあると有効であると考えられる．このことから SPADES は乗加算器（MAC）を付加できる仕様になっている．積和演算を行うためには以下に示す式から

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

と 3 つのデータが必要になる．よって，データハザードを判定すべき情報が二入力よりも一つ増えるためにフォワーディングユニットのポート数やアーキテクチャが変わる．そのために 3 入力演算を行えるスロット数の情報は必要不可欠となる．

自動生成システムはこれら 3 つの情報を読み込み，フォワーディングユニットのポート数を決定，アーキテクチャを生成し，VHDL を出力する．またフォワーディングユニットは，パイプライン段数が 4 段以降のみ有効になる．SPADES プロセッサでは，3 段目から演算を行うことを前提としているために，4 段目の既に処理された値からデータ先送りの必要性が出てくるためである．本提案手法におけるフォワーディングユニット記述自動生成システムは，パイプライン段数，スロット数ともに $s \leqq 3$ 以外に制約を設けていないのでパイプライン段数，スロット数に上限なく HDL 記述自動生成を行える．図 6 にフォワーディングユニット最適化生成手法のフローを示す．

4. 計算機実験結果

本章では，逐次判定型フォワーディングユニット自動生成システムと先見判定型フォワーディングユニット自動生成システムを用い，パイプライン段数 s，スロット数 n，3 入力演算 t の値を変えさまざまなフォワーディングユニットの HDL 記述を生成，それを論理合成した結果から本提案手法の有効性を示す．

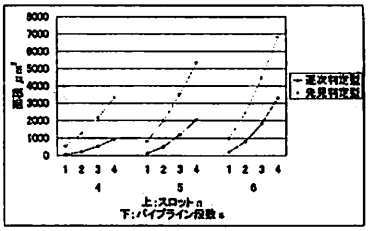

図 7　2 タイプのフォワーディングユニット面積比較

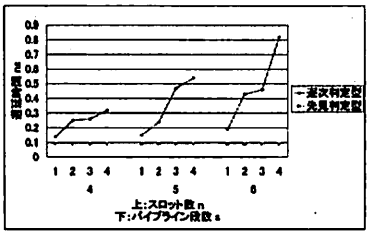

図 8　2 タイプのフォワーディングユニット遅延時間比較

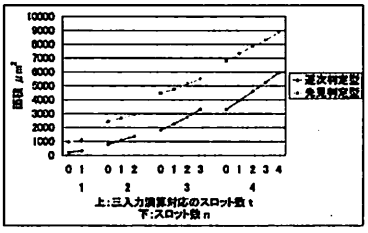

図 9　3 入力演算対応のフォワーディングユニット面積比較

論理合成は Synopsys 社の Design Compiler で行い，解析の際のセルライブラリは STARC (注2)（CMOS90[nm]）の設計ルールを用い，制約はなしで行った．

4.1 合成結果

レジスタアドレスのビット数が 3 である 2 タイプのフォワーディングユニットを，パイプライン段数 s を $s=4$ から $s=6$，スロット数 n を $n=1$ から $n=4$ まで変化させ生成した．その面積/遅延の変化を図 7，8 に示す．またパイプライン段数 $s=6$ の時のみ 3 入力対応スロット数 t をスロット数分変化させた面積/遅延の変化を図 9，10 に示す．面積の単位は $[\mu m^2]$，遅延時間の単位は $[ns]$ である．　逐次判定型フォワーディングユニットの遅延時間は，クリティカルパスの値であり，先見判定型フォワーディングユニットの遅延時間は，レジスタに格納されたデータが出力されるまでの値をとっている．先見判定型はフォワーディングの判定を一つ前の処理の最中に行うためク

(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．

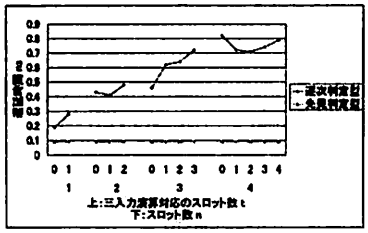

図 10　3 入力演算対応のフォワーディングユニット遅延時間比較

リティカルになる部分は解消されており，出力に要する値のみがクリティカルになる可能性があるためにこの値を得た．

また本自動生成システムにまったく知識のない学生 3 名に協力してもらい，$s=4, n=6, t=4$ に対応するフォワーディングユニットを生成していただいた．レジスタアドレスのビット数は 3 である．自動生成システムの実行の仕方の説明時間を含め，生成するのにかかった平均時間と，記述量を表 1 に示す．

表 1　出力記述量と生成所要時間

タイプ	出力 VHDL[行]	所要平均時間 [min]
逐次判定型	2246	12.22
先見判定型	2713	8.86

4.2　提案手法の適用可能性

提案手法の適用範囲は，図 6 よりクリティカルパスが EXE_1 以外のときであることがわかる．SPADES のターゲットアプリケーションは画像処理やディジタル信号処理などが多いために，乗加算器は非常に多用されると考えられる．MAC などはその演算器が遅延値のクリティカルになることが考えられるためにパイプライン化されることが十分にある．パイプライン化された場合，EXE_2，EXE_3 まで演算が及ぶこともあるため，EXE_1 以外のパイプラインステージがクリティカルパスになることは十分に考えられる．このことから提案手法の適用可能性は十分に高いと考えられる．

4.3　本手法の有効性

本提案手法は，HW/SW 分割系から得られる情報よりフォワーディングユニットの面積を小さくすることを目的としている．図 7，8 より，パイプライン段数，スロット数の増加に伴い，面積/遅延が増加しているのがわかるが，同時に逐次判定型は先見判定型よりも面積が小さく，また先見判定型は逐次判定型よりも遅延時間が小さくなっていることがわかる．また 3 入力を考慮した図 9，10 においても同様の結果が得られていることがわかる．このことより逐次判定型と先見判定型は面積/遅延においてトレードオフの関係が示され，制約時間を満たす限りでフォワーディングユニット面積の最適化を行う本手法が有効であるといえる．

また，2000 行を超えるフォワーディングユニットの VHDL 記述が入力パラメータの与え方，自動生成システムの実行の仕方に知識のない学生が 10 分前後で生成できることから，記述の自動生成によって設計にかかる時間が大幅に短縮できる．

5. むすび

本稿では HW/SW 協調設計システム SPADES について述べ，SPADES が対象とするアプリケーションプロセッサにおけるフォワーディングユニット HDL 記述面積最適化自動生成手法を提案し，逐次判定型と先見判定型が面積/遅延においてトレードオフの関係にあることを示し，手法の有効性を示した．今後は逐次判定型，先見判定型両方のフォワーディングユニットの綿密な面積/遅延の見積もりが求められる．

謝　辞

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 18700049)，早稲田大学特定課題研究助成費，電気通信普及財団研究調査助成金，中島記念国際奨励財団研究助成金の援助を受けた．

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*, 625–630, 2001.

[2] A.kitajima, Y.Takeuchi, A.Shiomi, J.Sato, S.Kobayashi and M.Imai, "Architectural design space exploration of configurable processor using ASIP Meister," *SASIMI*, 181–187, 2003.

[3] C.Garcia, R.Lario, M.Prieto, L.Pinuel and F.Tirado, "Vectorization of multigrid codes using SIMD ISA extensions," *in proc. IPDPS 03*,58–65, 2003.

[4] G.Ezer, "Xtensa with user defined DSP coprocessor microarchitectures," *in proc. Computer Design 2000*, 335–342, 2000.

[5] 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 Transactions*, Vol.8, 173–183, April 2000.

[6] J.Tanabe, Y.Taniguchi, T.Miyamori, Y.Miyamoto, H.Takeda, M.Tarui, H.Nakayama, N.Takeda, K.Maeda and M.Matsui, "Visconti: Multi-VLIW image recognition processor based on configurable processor," *CICC*, 185–188, 2003.

[7] 栗原 輝, 宮岡 祐一郎, 戸川 望, 柳澤 政生, 大附 辰夫, "SIMD 型プロセッサの自動合成におけるパイプライン演算ユニット生成手法," DA シンポジウム, 25–30, 2005.

[8] 伊藤 真紀子, 檜垣 茂明, 塩見 彰睦, 佐藤 淳, 武内 良典, 今井 正治, "パイプライン・ハザードを考慮した合成可能なプロセッサの HDL 記述生成手法の提案," DA シンポジウム'99 論文集, 201–206, 1999.

[9] O.Schliebusch, A.Hoffmann, A.Nohl, G.Braun and H.Meyr, "Architecture implementation using the machine description language LISA," *in proc. ASP-DAC/VLSI Design 2002*, 239–244, 2002.

[10] パターソン ヘネシー, "コンピュータの構成と設計 ハードウェアとソフトウェアのインタフェース 上下 第 3 版," 日経 BP 社, 2006.

[11] P.G.Paulin and M.Santana, "FlexWare: a retargetable, embedded-software development environment," *IEEE Design & Test of Computers*, 59–69, 2002.

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

[13] 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, VLD2006-14, 1–6, 2006.