3J-4

# PISAパイプラインプロセッサの設計とFPGAへの実装

三村 貴志　横田 隆史　大津 金光　馬場 敬信†
宇都宮大学工学部情報工学科‡

## 1　はじめに

我々はプログラム実行中のプロファイリング情報を用いて動的最適化を行うメタレベル最適化計算機システムYAWARA[1]を提案している。YAWARAはTE(Thread Engine)と呼ばれるVLIWプロセッサを複数並べたCMP(Chip Multi Processor)の構成になっており、各TEがプログラムの実行・プロファイリング・最適化の処理を分担して行うことにより、動的最適化を実現する。また、TEは最大で4つのRISC命令を同時実行可能なVLIWプロセッサである。

本稿では、TEを設計するための礎として、パイプラインプロセッサの設計とFPGAへの実装を行った。

## 2　設計方針

まず、命令セットをPISA(Portable Instruction Set Architecture)[2]で定義されている命令セットと同一のものとした。命令長は64bit、データ長は32bitである。命令セットとしてPISAを用いた理由は、PISAがMIPSをベースとしたシンプルなアーキテクチャであるとともに、拡張性に優れているためである。また、単一VLIWプロセッサシミュレーション環境であるCHA-MEN[3]がPISA命令セットを採用しているため、CHA-MENの言語処理系を利用できるという点も挙げられる。これによって、CHA-MEN言語処理系を利用して、動作検証用のアセンブリコードを容易に入手できる。

尚、本稿では、基礎的なプロセッサの設計を目的としているため、簡単な整数系プログラムをFPGAへ実装したプロセッサで動作させることを第一の目標とした。実数系プログラムは対象にしないので、浮動小数点関連命令は実装しない事とした。また、複雑な制御(ファイルを扱う等)を行うプログラムを対象としないため、システムコールも実装していない。

設計の手順は、始めに全命令のデータパスを検討し、主要な機能(レジスタファイル・ALU等)を独立したモジュール(以下では機能モジュールと称す)で設計する。これは、主要機能を独立させ、各モジュールのインターフェースを決定しておく事により機能の変更に柔軟且つ容易に対応できるようにするためである。また、プロセッサ内での作業の流れを考慮した上で独立させる事により、後のパイプライン化が容易に行える。各機能モジュールを設計した後、非パイプラインプロセッサを設計し、検討したデータパスが正しいか、機能モジュールが正しく動作するかをシミュレーションで確認する。その後パイプラインのステージ数を検討、プロセッサをパイプライン化し、正しく動作するかをシミュレーションにて確認する。そして設計したパイプラインプロセッサをFPGAへ実装し、小規模なプログラムを実行して結果を確認することとした。

## 3　パイプラインプロセッサの設計

設計はVerilog-HDLで行い、統合開発環境であるXilinx ISE 5(Xilin社)、論理合成ツールとしてSynplify Pro 7.3(Synplicity社)を用いた。

Design of PISA Pipeline Processor and Its FPGA Implementation
† Takashi Mimura, Takashi Yokota, Kanemitsu Ootsu and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

### 3.1　データパスの検討と機能モジュールの設計

まず始めに、実装する全命令のデータパスを検討し、そこから独立した機能モジュールとして設計する機能を検討した。その結果、機能モジュールとして命令キャッシュ・デコーダ・レジスタファイル・ALU・データキャッシュを独立させることとした。命令キャッシュはアドレスを入力するとそのアドレスに格納されている命令を出力する。今回はキャッシュを設計していないので、複数のレジスタで代用した。デコーダは命令を入力すると制御信号を出力する。レジスタファイルは汎用レジスタを32個と乗除算の結果を格納するHIGH・LOWレジスタを読み書きする。ALUは四則演算・論理演算・シフト演算を行う。データキャッシュは入力されたアドレスに対して、データの読み書きを行う。命令キャッシュと同様にキャッシュの設計は行わず、複数のレジスタで代用した。

### 3.2　非パイプラインプロセッサ

機能モジュールを設計し終えた後、非パイプラインプロセッサの設計を行った。ブロック図を図1に示す。その後、シミュレーションによる動作検証を行った。

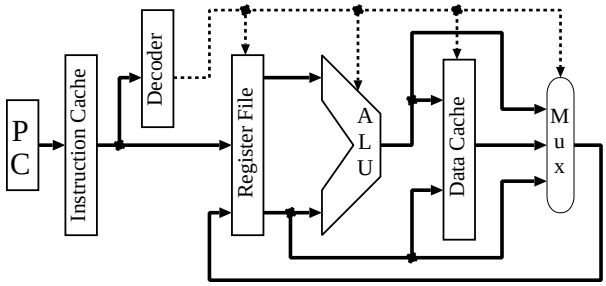


図 1. 非パイプラインプロセッサのブロック図

シミュレーションにはModelSim XE II v5.6e(Mentor Graphics社)を利用した。まず、実装した全命令が正確に動作するかを1命令ずつ確認した。次に四則演算を行うプログラムを動作させ、演算結果が正しいかを確認した。検証の結果、全命令が正確に動作し、四則演算の実行結果も正しい事が確認できた。

### 3.3　プロセッサのパイプライン化

まず始めに、データパスや機能モジュールを考慮した上でパイプラインのステージ数を検討した。プロセッサ内では命令は図1の左から右に流れるように進んでゆく。そこで、図1のPCと命令キャッシュモジュールを1つにまとめ、実行する命令を確定するIF(Instruction Fetch)ステージ、デコーダモジュールを主とした命令の制御信号を生成するID(Instruction Decode)ステージ、レジスタファイルモジュールを主としたレジスタファイルの内容を読みだすRR(Register Read)ステージ、ALUモジュールを主とした演算を実行するEX(EXecution)ステージ、データキャッシュモジュールを主としたデータキャッシュの読み書きを行うMA(Memory Access)ステージ、レジスタファイルへの書込みを行うWB(Write Back)ステージの6ステージで設計する事とした。そして各ステージ間にパイプラインレジスタを挿入した。これにより、プロセッサはパイプライン処理が可能となった。

### 3.4 フォワーディング

プロセッサをパイプライン化しただけでは命令間にRAW(Read After Write)の依存関係が存在する場合に、正しい演算結果が得られないことがある。そこで、フォワーディングを行う必要がある。フォワーディングはMAステージやWBステージで処理中の命令の演算結果を直接EXステージの入力とする処理である。これによりレジスタに書き込まれる前の値をEXステージの演算で扱えるようになり、RAW依存関係による問題が解決した。

### 3.5 ハザードの解決

フォワーディングを行うことにより、RAW依存関係による問題を解決できるが、1つだけフォワーディングでは解決できないRAW依存関係がある。それはload命令で値を書き込むレジスタの値を次の命令で使用する場合である。load命令ではレジスタに書き込む値が確定するのはMAステージでの処理を終えた後である。しかし、その時点ではEXステージでの演算はすでに終了しているため、フォワーディングを行うことが出来ない。そこで、そのような場合にはハザード(障害)が発生したとし、IF・ID・RR・EX間のパイプラインレジスタとPCの更新を停止する。そして、ハザードが解消されたら更新を再開する。load命令によるハザードでは1クロック分停止させる事によりMAステージからのフォワーディングが有効になる。load命令以外には除算命令によるハザードが発生する。除算命令は1クロックで処理する事ができないため、後続命令で演算結果を利用する場合にハザードが発生する。この場合には除算命令の終了と同時に更新を再開させる。

以上のような処理を行うために非パイプラインプロセッサにフォワーディング制御ユニットとハザード制御ユニットを組み込んだ。以上の工程を経て設計されたパイプラインプロセッサのブロック図を図2に示す。

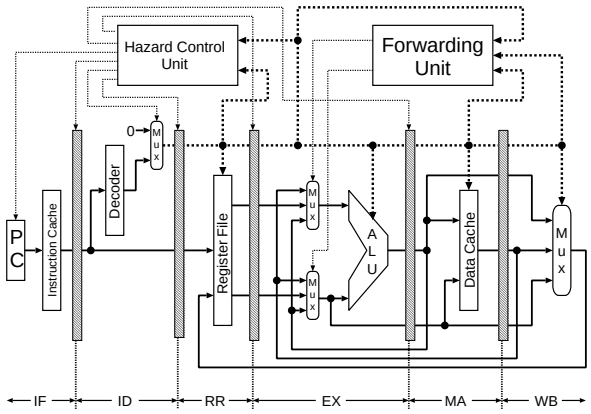


図 2. パイプラインプロセッサのブロック図

### 3.6 パイプラインプロセッサの動作検証

非パイプラインプロセッサの動作検証と同様に、ModelSimを用いてシミュレーションによる動作検証を行った。四則演算プログラムを実行した結果、結果が正しく求められる事を確認した。

## 4 FPGAへの実装

設計したパイプラインプロセッサをFPGAへ実装した。FPGAはVirtex-II XC2v6000(Xilinx社)を使用した。

### 4.1 RAM・UARTの実装

これまでのプロセッサ設計過程における動作検証では実行命令数が20程度と少ない、簡単な四則演算プログラムを用いてきた。そのため命令・データキャッシュモジュールの動作を複数のレジスタを用いるという方法で実現しても、用意するレジスタ数が少なくて済み、実行する命令をレジスタの初期値として与える事でプログラムを容易に実行する事が出来た。また、シミュレーションでは回路規模を考慮する必要は無かった。しかし、命令数が100〜200程度の小規模なプログラム(エラトステネスの篩やNQueen問題)を実行するためには多数のレジスタを使用することになってしまう。そのため、実行する命令をレジスタの初期値として与える事が困難となり、回路規模が大きくなってしまう。そこで、FPGAのエンベデッドRAMを使用した。使用したFPGAには18Kbitのブロック RAMが144個搭載されている。このエンベデッドRAMを使用するために、ISE 5のCORE Generatorという基本的なHW要素(カウンタやRAM、コンパレータ等を自動生成するツールを利用して、デュアルポートRAMを生成し、プロセッサに組み込んだ。これにより、小規模なプログラムを実行出来るようになった。しかし、プログラムの実行結果を確認するための出力機能が無いため、プログラムを実行しても正しく動いている事が確認できない。そこで、UART(Universal Asyncronous Receiver Transmitter)を設計し、シリアル通信によってホストコンピュータに出力を行えるようにした。

### 4.2 小規模プログラムによる動作検証

FPGAに実装したプロセッサ上でエラトステネスの篩を用いての100までの間の素数を求めるプログラムと8-Queen問題・9-Queen問題を実行し、結果の確認を行った。結果は全てのプログラムで正しい結果が得られた。

### 4.3 回路規模

今回設計したプロセッサの回路規模を以下に示す。

- 使用スライス数:5441
- 使用F/F数:2387
- 使用ブロックRAM数:128

また、限界周波数は41.5MHzである。

## 5 おわりに

本稿では、TE設計の礎としてパイプラインプロセッサの設計を行った。そして小規模なプログラムを用いて動作検証を行うことにより、設計したプロセッサが正しく動作することを確認した。

今後の課題として、今回は実装を行わなかったシステムコールや浮動小数点演算機能の実装を行うことや、キャッシュの実装、プロセッサのVLIW化等が挙げられる。

**謝辞** 本研究は、一部日本学術振興会科学研究費補助金(基盤研究(B)14380135、同(C)16500023、若手研究(B)17700047)の援助による。

## 参考文献

[1] Takanobu Baba, et al: “YAWARA: A Meta-Lebel Optimizing Computer System,” IWIA, 2004.

[2] Doug Burger, T.Austin: “The SimpleScalar Tool Set, Version 2.0,” Univ. of Wisconsin-Madison Comuter Sciences Department Technical Repot # 1324(1997.6)

[3] 月川 淳ほか: “メタレベル最適化計算機システムYAWARAのシミュレーション環境 —PISAをベースとしたVLIWアセンブラの開発—”, 情処第67回全国大会, 2005.3