計算機アーキテクチャ 115－20
設 計 自 動 化 78－20
（1995. 12. 15）

ASIP 設計用ワークベンチ PEAS-III の実現方法についての考察 — CPU アーキテクチャの分類とパラメタ化 —

片岡健二† 塩見彰睦† 今井正治†
青山義弘† 佐藤淳‡ 引地信之§

† 豊橋技術科学大学
‡ 鶴岡工業高等専門学校
§ SRA

ASIC のコデザインに関する研究は種々行なわれているが，現状ではコデザインで用いられる CPU アーキテクチャがかなり限定されているのが現状である．しかし実際には対象となるアプリケーションの分野に応じて CPU のアーキテクチャを変化させた方が良い結果が得られると考えられる．本稿では ASIP 設計用ワークベンチである PEAS-III システムのアーキテクチャ記述入力系のプロトタイピングを行なった結果を報告する．特に様々なアーキテクチャに対応できる手法について述べる．また実現の際に得られた所見についても述べる．

Observations on the Implementation of a Codesign Workbench PEAS-III for ASIP Design — Classification and Parameterization of CPU Architectures —

Kenji Kataoka†, Akichika Shiomi†, Masaharu Imai†,
Yoshihiro Aoyama†, Jun Sato‡, Nobuyuki Hikichi§

† Toyohashi University of Technology
‡ Tsuruoka National College of Technology
§ Software Research Associates Inc.

In many HW/SW codesign system can handle very limited architecture class of CPU. However, application specific CPU would have better performance, HW cost, and power consumption trade-off. This paper describes the prototype implementation of PEAS-III, a HW/SW codesign workbench for ASIP (Application Specific Integrated Processor) design.

1 はじめに

プロセッサの開発，特に特定用途向きのプロセッサの開発に対して，その期間短縮が望まれている．一方，プロセッサ・アーキテクチャの複雑さは増大しつつあるので回路規模も増大し，それに伴い設計工数も増大する傾向がある．

そのような特定用途向きのプロセッサのコデザインに関する研究は，CPU 内部まで最適化を行なうような研究も行なわれているものの [1][2][3]，CPU のアーキテクチャはかなり限定されてしまっているのが現状である．しかしアプリケーションによって最適なアーキテクチャも変化すると考えられる．したがって，CPU アーキテクチャのチューニングによって，与えられたアプリケーションに対してより高い性能を持ち，回路規模が小さく，消費電力の少ない CPU が利用可能となる．

このような特定用途向き CPU を設計するためには多様なアーキテクチャに対応可能な CAD ツールが必要になる．そこで筆者らは HW/SW コデザインワークベンチ PEAS-III(Practical Environment for ASIP Development Type III) を提案した [4]．PEAS-III システムはアーキテクチャの入力の際に様々なアーキテクチャ・パラメータを設定でき，設計したプロセッサの性能の評価を設計の早い段階で行なうことができる．本稿では PEAS-III システムのプロトタイピングを通して多様なアーキテクチャに対応する手法について述べる．

2. で PEAS-III システムの概要について簡単に説明し，3. でハードウェアを一括して扱うための方法について示す．4. では HDL 記述の生成方法について説明する．5. で本研究で得られた所見等について述べ，最後に 6. でまとめと今後の課題について述べる．

2 PEAS-III の概要

PEAS-III は ASIP(Application Specific Integrated Processor) を念頭に置いた CPU 設計ワークベンチである．PEAS-III システムの構成を図 1 に示す．

PEAS-III システムは，まず入力された応用プログラムを解析し，プロセッサの設計に役立つ情報をユーザに表示する．この情報と設計条件を考慮して，ユーザがアーキテクチャや命令セットの仕様を入力する．予測系では入力されたアーキテクチャの性能

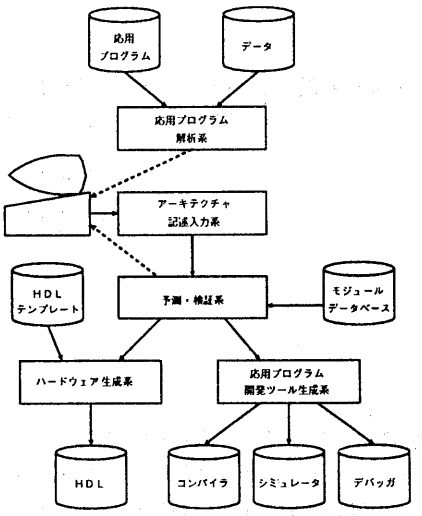

図 1: PEAS-III システムの構成

等を予測し，その結果を再びユーザに表示する．もし所望の特性が得られていなければアーキテクチャを変更する．予測系では，同時に検証も行なうので，矛盾があった場合にはすぐに検出することができる．これらを何度か繰り返して所望の特性の得られるアーキテクチャを決定する．アーキテクチャが決定されると，システムはプロセッサの HDL 記述と，その応用プログラム開発ツール群を自動生成する．

本稿では特に，アーキテクチャ記述入力系，及びハードウェア生成系について述べる．

3 アーキテクチャ記述入力系

通常，プロセッサの設計ではまずハードウェア構成と命令セットを決めるが，両者は密接に関連しているので，それぞれを変更しながら設計を進めていくことが多い．従ってこれらを一括して扱うことのできる環境が必要になる．また，PEAS-III で扱うことのできるハードウェアは多岐にわたるため，様々なハードウェア・パラメータを設定できなければならない．PEAS-III ではこれら部分はアーキテクチャ記述入力系で実現している．

また，アーキテクチャ記述入力系はユーザ・インタフェースの核であり，システムに対する入力はほとんどここで行なわれる．従って入力系はユーザ

にとって分かりやすいようにGUI(Graphical User Interface)を用いる．

以下，アーキテクチャ記述入力系の詳細について説明する．

3.1 アーキテクチャ・ライブラリ

入力しなければならないアーキテクチャ・パラメータは，アーキテクチャの型や割り込みの許可といった，ごく基本的なパラメータと，アーキテクチャの型に依存したパラメータがある．そこで考えられるパラメータを基本パラメータとアーキテクチャの型に依存するパラメータに分類する．

基本的なパラメータは以下の通りである．

1. アーキテクチャの型
2. クロックスキーム
3. レジスタ構成
4. 割り込み
5. I/O

アーキテクチャの型としてはスカラ型，スーパースカラ型，VLIW型などが考えられるが，それぞれに固有のパラメータが存在する．そこでこれらを同一の枠内で扱う手法が必要になる．

解決策として，それぞれの型ごとに設定しなければならないパラメータをデータベースとして登録する方法を採用した．このアーキテクチャ・パラメータが登録されるデータベースをアーキテクチャ・ライブラリと呼ぶ．

このアーキテクチャ・ライブラリには，パラメータの名前と，そのパラメータがとり得る値の型を記述しておく．それぞれのパラメータのとり得る値としては以下の3種類が考えられる．

1. 整数値（integer型）
2. 論理値（boolean型）
3. 選択値（select型）

整数値をとるものとしては，例えばデータのビット幅やパイプライン段数などがある．割り込みやI/Oを持つかどうかといった問題は論理値で表すことができる．また，制御方式はスカラ型，スーパスカラ型，VLIW型などの中から一つを排他的に選択するので選択値に該当する．選択値はC言語などの列挙型に似ている．選択値の場合は選択肢も登録しておく必要がある．

3.2 リソースの設定

実際にプロセッサの内部で使用するリソースを設定する．リソースはあらかじめ用意されているリソースの中から選択することもできるし，ユーザが独自に設計したものを利用することも可能である．

リソースの場合にも，アーキテクチャの制御方式の場合と同様の問題がある．つまり，各リソースがそれぞれ異なる設定パラメータを持っている．そこでアーキテクチャ・パラメータの場合と同様の手法でこれらを管理する．リソースのパラメータ等を登録しておくデータベースをリソース・ライブラリと呼ぶ．

リソースライブラリには以下の項目を登録しておく．

1. ハードウェア・コスト
2. 入出力端子
3. 制御入力端子と制御の種類
4. リソース固有の設定パラメータ

ユーザはリソース・ライブラリに登録されたリソースを選択し，実際に使用するインスタンスとしてインスタンス名とパラメータを設定する．ここで設定されたインスタンス名を使って，次節に述べる命令の仕様を記述する．

3.3 命令仕様の入力

命令の仕様記述は，命令のマイクロ動作を記述することによって行なう．これらを全ての命令について記述したものをマイクロ動作表と呼ぶ．このマイクロ動作表はフェーズ単位で，リソース間のデータの流れと制御情報を最小粒度で記述する．基本的な記述形式は次の2種類である．

1. リソースからリソースへの転送
2. リソースの起動

である．

転送の記述では，あるリソースの出力端子から別のリソースの入力端子へのデータ転送を記述する．従って，リソースごとにどのような入出力端子を持っているかをユーザに示す機能が必要になる．記述されたマイクロ動作から，ハードウェアのHDL記述の生成に必要な接続情報，制御情報を抽出する．マイクロ動作表の記述例を図2に示す．

命令	IF	ID	EX	...
addsi	IM_ABUS <= PC IMEM(READ) IR <= IM_DBUS UCNT.in <= PC UCNT() PC <= UCNT.out	DEC(IR)	BUS1 <= REGF(R1) BUS2 <= REGF(R2) ALU.in1 <= BUS1 ALU.in2 <= BUS2 ALU(ADD) BUS3 <= ALU.out	...

図 2: マイクロ動作表の記述例

また，マイクロ動作表の記述と並行して，命令セットに関するデータを入力する．命令関連のデータには次の項目がある．

1. ニモニック
2. オペコード
3. ビットフィールド
4. オペランドの型

これらの中で，ビットフィールドに関しては，ユーザがビット割当を命令タイプとして定義し，命令ごとにその命令の属す型を決める．

また，オペランドの型はオペランドの型をクラスの中から選択する．選択できるクラスには即値，メモリ，レジスタクラスがある．レジスタクラスはレジスタの集合として定義される．ユーザはレジスタクラスの名前とそのレジスタクラスに属すレジスタを設定できる．マイクロ動作表でレジスタを対象に転送などを行なう場合にはこのレジスタクラスを指定する．

3.4 中間ファイルの生成

アーキテクチャ記述入力系で入力されたアーキテクチャや命令の仕様などの情報はいくつかの中間ファイルに出力する必要がある．中間ファイルには次の種類がある．

1. 制御情報
2. 接続情報
3. リソース情報
4. 命令情報

がある．

制御情報の主な内容は制御に必要なアーキテクチャ・パラメータとマイクロ動作表から抽出された演算器の起動タイミングである．

接続情報は，転送記述に基づいて接続すべき 2 つのリソースの組である．

リソース情報は各リソース・インスタンスのパラメータ・データである．

命令情報は，マイクロ動作表として記述されている各命令に関するデータで，命令コードやその命令のアセンブラのフォーマット，ビットフィールドなどが含まれる．

3.5 プロトタイプ

作成したアーキテクチャ記述入力系の実行時の画面の例を図 3に示す．

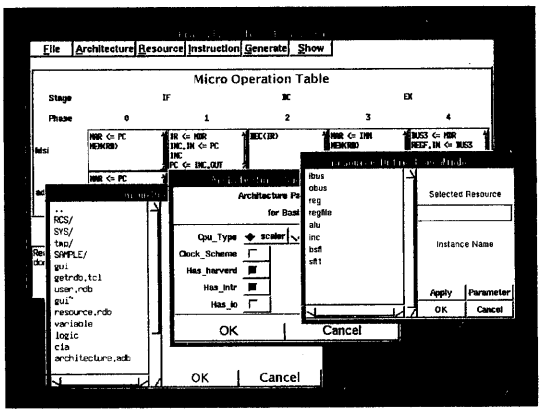

図 3: プロトタイプの実行例

プロトタイプの作成は Tcl/tk を用いてコーディングを行なった．Tcl/tk の主な特徴は GUI の作成に必要なボタンなどの部品をあらかじめ用意してあり，これらを組み合わせていくことで簡単に GUI を作ることが可能であること，PDS であること，習得が容易であることなどである [8].

4 HDL 記述生成系

4.1 概要

HDL 記述生成系は，アーキテクチャ記述入力系で入力されたアーキテクチャ・パラメータのデータや命令仕様に基づいてハードウェアの HDL 記述を生成する．ハードウェア生成に必要な情報としては，マイクロ動作表の他に，アーキテクチャ・データ，各リソースのデータ，命令関連のデータ（命令コードやフォーマットなど）がある．

HDL としては VHDL を採用した．VHDL はエンティティ/アーキテクチャの概念を持っており，実際に HDL 記述の生成を行なおうとした場合に都合

がよい．またVHDLはIEEEの標準ハードウェア記述言語であり，合成ツールなども多数市販されている．

HDL記述生成系は以下の3つのサブシステムから構成されている．

1. ハードウェア・モジュール記述生成系
2. コントローラ記述生成系
3. トップモジュール記述生成系

それぞれの生成系は必要なデータから各モジュールのHDL記述を生成する．

プロトタイプではシミュレーション可能なVHDLのビヘイビア記述を出力することとした．最終的には合成可能なVHDL記述を出力できるようにする予定である．

4.2 リソース生成系

リソース生成系ではパラメタライズされた演算器などのリソースのHDL記述を生成する．リソース生成に必要な情報はリソースデータである．リソースデータの項目にはビット幅や，演算器のパイプライン構成など細かな設定ができるようになっている．

4.3 コントローラ生成系

コントローラ生成系ではコントローラをモジュールとして生成する．

コントローラの生成には，制御に関連するアーキテクチャのデータを読み込む必要があるが，どのようなパラメータがあるかはアーキテクチャ・ライブラリから読み込み，そのデータを入力系から出力されるデータフィルから読み込む．

コントローラはステートマシンであり，状態によって各部への制御信号を生成するようなモジュールである．各リソースへの制御信号は，マイクロ動作表から得られた制御情報から，適切なタイミングで制御信号を出力するように生成される．

4.4 トップモジュール生成系

トップモジュール生成系では主に接続情報を用いて，各リソース，制御部をサブモジュールとするようなトップモジュールの構造記述を生成する．

5 考察

本研究を通して得られた所見などを項目ごとに述べる．

1. アーキテクチャ・パラメータ
   アーキテクチャ・パラメータのうち，データベースに登録されていないパラメータはデータパスの構成に関するパラメータであり，PEAS-IIIシステムにおいてはデータパスの設計はマイクロ動作表を記述することと等価なので登録する必要がない．このように考えると，真に必要なアーキテクチャパラメータはそれほど多くない．
2. 情報の抽出
   HDL記述を生成するには，マイクロ動作表から制御情報，接続情報を抽出する必要がある．制御情報は主にリソースの制御信号であるが，これはリソースの起動を明示的に記述しているため，制御信号を出すタイミングと内容を容易に得ることができる．また，接続情報もリソースの端子間での転送を記述するために，どのような接続になっているかを容易に得られる．

   このように最小粒度で記述を行なうことによってハードウェア生成のための情報を得やすくしている．しかし最も低レベルでの記述となってしまっているので記述量が増大してしまい，デザインが大きい場合には変更による煩雑さも大きくなるという問題点もある．この問題を解決するため，著者らはマクロ機能を持たせることを検討している．いくつかのマイクロ動作を マクロとして定義することで記述量を減少させることができる．
3. ライブラリ化
   アーキテクチャやリソースのパラメータをデータベース化しておくことで，異なったオブジェクトを同一の枠組で扱うことができる．特にリソースの場合には，ユーザ独自の演算器を追加したい場合にも，データベースに必要な項目を登録するだけで既存のリソースのように使用すること可能である．これにより設計の再利用が容易に行なえる．
4. レジスタの扱い
   命令のフォーマットに関しては，マイクロ動作表を記述する際に，基本的にインスタンス名を記述するために汎用レジスタの扱いが問題

となった．レジスタを対象とする命令のマイクロ動作を記述するには，レジスタのインスタンス名で接続を記述しなければならない．しかし複数のレジスタの中から一つを選択し，そのレジスタを対象とする命令を記述しようとした場合，インスタンス名を記述することはできない．この問題を解決するためにレジスタクラスという概念を導入した．これにより複数のレジスタを対象とするような命令の定義もできる．

6 まとめ

アーキテクチャのパラメータを統合化して扱うことのできる手法を提案し，同時にHDL記述の自動生成を行なう手法についても述べた．本稿で提案した手法を用いることによって，様々なタイプのプロセッサの設計を迅速に行なうことが可能となる．

また，アーキテクチャ記述入力系，HDL記述生成系に関する主な今後の課題としては以下の項目が挙げられる．

- マイクロ動作表の記述の簡素化
- 合成可能なHDL記述の出力
- 他のアーキテクチャへの対応

マイクロ動作表の抽象化は，前節の2項で述べたような問題を解決しなければならない．

次に，現在のHDL記述生成系はシミュレーションモデルを出力するようになっているが，これを合成可能な記述を出力するように変更しなければならない．合成可能なVHDLテンプレートライブラリの作成もこれに含まれる．

最後に他のアーキテクチャへの対応であるが，本稿では選択可能な制御方式をスカラ型（パイプライン，ノンパイプライン）のみに限定して述べてきたが，他の制御方式についても同様に扱える必要がある．スーパスカラ型に関しては，データパス部を変更することはほとんどなく，制御部の変更の形で対応できるため，ほぼ問題ないと思われるが，VLIW型を同様に扱えるかどうかは，現在検討中である．

謝辞

最後に，本研究を行なうにあたり討論して頂いた松下電気産業(株)半導体研究所の村岡，松本の両氏，豊田工業高等専門学校の木村 助手，豊橋技術科学大学VLSI設計研究室の諸兄に感謝致します．また，研究環境を提供して頂いている(株)サイエンス・クリエイトに感謝致します．なお，本研究の一部は科学研究費補助金 一般(C)07680353，試験(B)(1)07558038、奨励(A)07780294の研究助成による．

参考文献

[1] J. Wiliberg, R. Camposano, W. Rosestiel, "Design Flow for Hardware/Software Cosynthesis of a Video Compression System" IEEE International Conference on Computer Design, pp428–431, 1994

[2] D. E. Thomas, J. K. Adams, and H. Schmit: "A Model and Methodology for Hardware-Software Codesign," *IEEE, Design & Test of Computers*, Vol. 10, No. 3, pp. 6–15, September 1993.

[3] 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," *Trans. IEICE*, Vol. E77-A, No. 3, pp. 483–491, March 1994.

[4] 塩見彰睦・今井正治・片岡健二・青山義弘・佐藤淳・引地信之, "ASIP設計用コデザインワークベンチPEAS-IIIの提案", 情報処理学会 設計自動化 76-10, pp. 73–80, 情報処理学会, 1995

[5] J. L. Hennessy and D. A. Patterson, "Computer Architecture a Quantitative Approach," Morgan Kaufmann Publishers , 1990.

[6] 古渡聡・岩下洋哲・中田恒夫・広瀬文保, "パイプラインプロセサの制御論理自動生成", 信学技報 VLD94-41, pp.17–24, 電子情報通信学会, 1994

[7] "IEEE:IEEE Standard VHDL Language Reference Manual," Std 1076-1993, IEEE, 1994.

[8] John K. Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley Publishing Company Inc., 1993.