設計自動化 68－7
（1993. 10. 28）

計算機アーキテクチャ設計支援に関する考察

赤星 博輝　安浦 寛人
九州大学 大学院総合理工学研究科 情報システム学専攻
〒816 春日市春日公園 6—1
E-mail: {akaboshi, yasuura} @is.kyushu-u.ac.jp

あらまし 計算機アーキテクチャの設計支援として，今までの論理/レイアウト設計に対する設計支援だけでは十分でなく，性能評価や検証などの支援が必要である．我々は，ハードウェア/ソフトウェアのインターフェイスを中心とした設計について性能評価や検証も含めた設計支援の検討を進めている．ハードウェア/ソフトウェアのインターフェイスを中心とした計算機アーキテクチャ設計に必要とされる，コンパイラ生成やハードウェア記述言語や，さらにハードウェア記述から情報抽出などについて考察を行う．

和文キーワード：計算機アーキテクチャ設計，コンパイラ生成，性能評価

Study on Support for Computer Architecture Design

Hiroki AKABOSHI and Hiroto YASUURA

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan

E-mail:{akaboshi,yasuura} @is.kyushu-u.ac.jp

Abstract For computer architecture design, computer architects need the support for performance evaluation and verification as well as the support for logic/layout design. To design and evaluate a computer architecture, the support at the hardware/software interface level is required. We study the requirements to support the architecture design such as compiler generation, hardware description language, information extraction, etc.

英文 Key Words: Computer architecture design, Compiler generation, Performance evaluation

1 はじめに

狭い意味で計算機アーキテクチャの設計とは,「命令セットや構成方式の設計」といえる．しかし，高性能な計算機アーキテクチャを設計するためには，命令セットや構成方式の設計と同様に「論理設計，レイアウト設計」も同時に考慮した設計を行う必要がある．また，今まで組込み用プロセッサでは，Z80 など既存のプロセッサをコア・プロセッサとして使い，必要とされる機能を付加回路として周辺に組み込むことが行われている．この方法ではコア・プロセッサの内部を変更することができいため，より細かい部分まで変更できなかった．CAD 技術の進歩により論理/レイアウト合成が自動化可能となり，ハードウェア記述言語で記述されたコア・プロセッサを使用することで，設計者が内部まで変更を行うことが可能となる．このような手法により，大幅な変更が可能となり，さまざまな用途に使用できる組込み用プロセッサが作成できると考えている．この環境の中で性能向上をねらうには，命令セットや演算器を追加したり減少させたりすることで，動かそうとするプログラムに応じて最適化を行う必要がある．

高性能な計算機アーキテクチャを作る場合も組込み用プロセッサを作る場合も仕様を満たすためには，今までのような論理/レイアウト設計の支援ツールやシミュレータを用いハードウェアの評価をすると同時に，命令セットや構成方式といったハードウェア/ソフトウェアのインターフェイスとなる部分の評価をプログラムを使って実際に評価する必要がある．

そのために計算機アーキテクチャの設計支援についての考察を行い，現在，開発中の計算機アーキテクチャ設計支援システム "COACH" のプロトタイプでの問題点から，さらに今後必要な支援についての考察を行う．本論文では，2 章で計算機アーキテクチャの設計について述べ，3 章で計算機アーキテクチャ設計支援システムのプロトタイプの概要について説明を行い，4 章でプロトタイプでの問題点についての考察，5 章で今後検討すべき点について述べる．

2 計算機アーキテクチャ設計

2.1 ターゲット

計算機アーキテクチャといってもさまざまなターゲットがある．大きく分けると，ワークステーションなど汎用の計算機で用いる MPU，それに対して組み込み制御用のプロセッサに分けられる．

汎用計算機では，より高速化を行うために，さまざまな構成方法，命令セットなどを考えていく必要がある．

組込み制御を行うプロセッサは，潜在的な需要は大きく，これらの要求に応えられるだけの設計手法の確立が必要である．今までの手法では，図 1のように予め用意されたコア・プロセッサに機能を付加することはできるが，コア・プロセッサ自体を変更することはできなかった．しかし，CAD システムの進歩により，集積回路に対する知識が無くてもハードウェア記述言語によって動作を記述するだけで集積回路が作成できるようになっている．一からプロセッサを作成すると多大な労力を要するが，基本となるプロセッサのハードウェア記述言語による記述を用意することで，設計者は用途に応じてその記述を変更しプロセッサを作成することが可能になる．今までの手法では，ハードウェアは増加する方向にしか変化することはできなかったが，ハードウェア記述言語によるコア・プロセッサではハードウェア/ソフトウェアのどちらも変化するという環境を提供することができる [1].

汎用計算機，組み込み用プロセッサのどちらの計算機アーキテクチャの設計も，図 2のように大きく分けて 1) 設計，2) 検証，3) 性能評価の 3 つに分類できる．それぞれに，いくつかの段階，仕事がありそれぞれについて考察を行う．

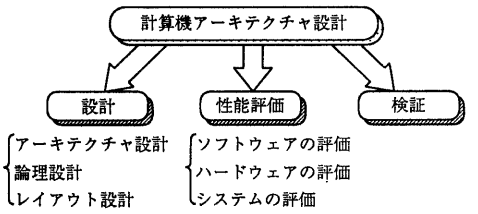

図 2: 計算機アーキテクチャの設計

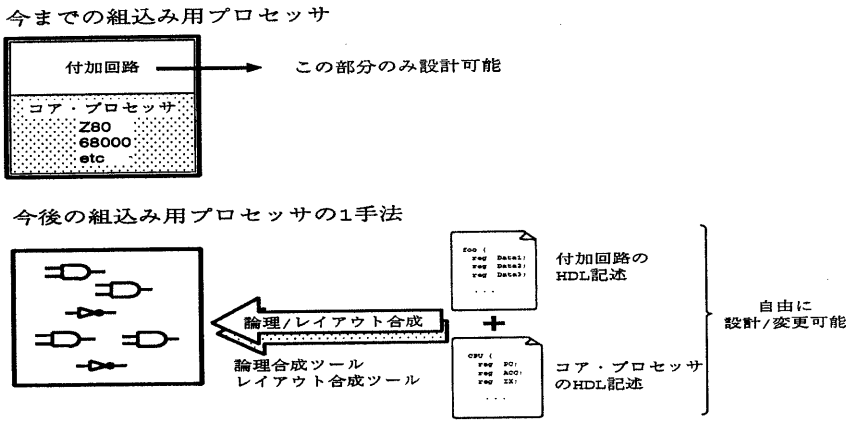

図 1: 組込み用プロセッサ

2.2 設計

設計も命令セットや構成方式を決定するアーキテクチャ設計，そして，インプリメントを行う論理設計，レイアウト設計の3つに分けられる．全体として目標 (プログラムの実行時間を最小，フォルトトレラント，ある時間内で処理が完了など) を小問題に分割し，各段階でその問題を解きながら設計を行っている．

アーキテクチャ設計では，命令セットやパイプラインなどの構成方式を変えることで，プログラムの実行ステップ数の最小化など用途に合わせた最適化を図る．論理設計では論理式の最簡化を行い，レイアウト設計では面積の最小化，クロック周波数の高速化を行う．

2.3 検証

検証も設計の各レベルで必要となるもので，仕様と論理設計，論理設計とレイアウトなどの検証が必要である．近年，計算機アーキテクチャは，高速化のためにさまざまな技術を取り入れている．そのためソフトウェアに対してより多くの処理を要求することが多い．実際に設計を行っているハードウェア上で，ソフトウェアが正しく動作するかという検証が重要となっている．例えば，現在 RISC プロセッサでは遅延分岐が取り入れられていることが多いが，そのためにコンパイラが正しく遅延スロットを埋めているか検証するなどである．

2.4 性能評価

性能評価には，1) ハードウェアの評価，2) ソフトウェアの評価，3) システムの評価がある．ハードウェアの評価項目としては，チップ面積，クロック周波数，消費電力などがある．ソフトウェアの評価項目としては，アルゴリズム，処理時間などがある．システムの評価項目としては，プログラムの実行時間あるいはステップ数，資源の利用状況などがある．計算機アーキテクチャの設計者が評価する場合は，ハードウェアとシステムの評価が重要である．汎用計算機の設計では，限られたハードウェアでプログラムの実行時間をできるだけ短くする必要がある．また，組み込み用プロセッサでは，ある時間内で処理を終了させることや消費電力を一定量以下にするなどの処理が必要である．特に組込み用ではプログラムが固定のものが多いため，そのプログラムだけの最適化が可能となる．最適な設計を行うためには，ハードウェアとシステムの双方の評価を同時に行う必要があるが，今まではハードウェアとシステムを同時に評価することは難しかった．現在，ハードウェア/ソフトウェアの双方の評価を行うシステムがいくつか研究・開発されている [2, 3].

プロセッサレベルの設計では，ソフトウェアに関しては高級言語で書かれたベンチマークプログラムを利用することが多いため，計算機アーキテクチャ設計時にはソフトウェア自体の評価はあまり行われていなかった (今後，並列処理などを取り入れると，ハードウェアとソフトウェアとの相性などか

ら考慮する必要があり，評価法など検討が必要である)．今回は，ハードウェアとシステムの評価についての考察を行う．

2.4.1 ハードウェアの性能評価

ハードウェアの評価を行うためには，

1. 論理/レイアウト合成を行いハードウェアの評価
2. 実際に論理/レイアウト合成を行わずに，何らかの定式化などによりハードウェアの評価

の2つの方法が考えられる．1の方法は今まで多く用いられていた手法で，実際に設計を行い論理/レイアウト設計を行い，チップ面積，クロック周波数などを求めることにより性能評価を行っていた．

それに対して，2の論理/レイアウト合成を行わずに，ハードウェアの評価を行う場合には，かなり予測の部分が多くなるために精度が問題となる．しかし，今後より設計回路の大規模化が進めば，実際に論理/レイアウト合成を行う時間の増大が予想される．迅速な性能予測を行うためには，今後実際に論理/レイアウト合成を行わないで，短時間でハードウェアの評価を行う評価手法も必要となる (現段階では，レイアウトを行わないでディレイなどの評価を行う研究が行われている [4])．

2.4.2 システムの性能評価

ハードウェア上でプログラムを動かすことで評価するものをシステムの性能評価とする．具体的には，実行ステップ数，演算器の使用率などを考えている．

システムの性能評価を行うためには，プログラムとハードウェアの情報をどう利用するか選択肢がある．

プログラム: ベンチマークプログラムが高級言語で与えられることが多いため，高級言語のみを考える．

A 実際にプログラムをコンパイルして評価

A' 既存のマシン用にコンパイル・実行し各命令や動作などの履歴をとっておいて，既存のマシンの命令と対応から評価

B ソースレベルでの評価

ハードウェア :

A ハードウェアの動作をシミュレーションして評価

B シミュレーションを行い評価

アーキテクチャ設計を行うために命令セットレベルでの変化が見える必要があり，Bの方法では細かな部分が見えない．A'では，汎用機のように命令セットレベルのインターフェイスが固定されているものについては有効な方法であるが，新しい命令セットなどを設計する場合には必ずしも有効ではない．

Aの方法では，命令セットや構成方式などを変化させてシステムの評価を行いたいが，いくつかのシステムをいくつかのベンチマークプログラムを使って評価するためにはコンパイラが不可欠となるが，この段階ではコンパイラがない場合が多い．A'の方法では新しい命令セットについての評価がされないために不十分である．そのために，アーキテクチャに応じたコンパイラを自動的に生成するコンパイラ・ジェネレータが必要となる．

また，ハードウェアの動作をシミュレーションするため，どのようにハードウェアを記述するかと言う問題がある (ここでは，ハードウェアを記述する言語をハードウェア記述言語と呼ぶ)．

2.5 コンパイラ・ジェネレータ

コンパイラ・ジェネレータとは，アーキテクチャの情報を与えることで，そのアーキテクチャに応じたコンパイラを生成するものである．コンパイラ・ジェネレータのメリットとしては，

- さまざまなアーキテクチャの検討を実際のプログラムを用いて評価することができる
- ハードウェアとソフトウェアとのインターフェイスを提供する

がある．特に，ハードウェアが変化してもそれに追随してコンパイラが生成されることで，ソフトウェアは再コンパイルするだけで良い．

コンパイラ・ジェネレータに対する要件としては，

- ターゲットアーキテクチャに応じたコンパイラを迅速に生成
- 設計者の負担の軽減

がある．

さまざまなアーキテクチャに対応しているGNU Cコンパイラ(GCC)[1]では，ターゲットマシンに非依存な中間言語を用いることにより，ソースプログラムから中間言語に変換する部分については共用とし，中間言語から実際にマシンコードを生成する部分でそれぞれのアーキテクチャに応じてコンパイルする手法が採られている．

コンパイラ・ジェネレータでは，さまざまな命令セットにどのように対応するのか，さまざまな構成方式にどのように対応するのかという2つの問題ある．

コンパイラ・ジェネレータに与える情報をどのように与えるかという問題がある．これについては，将来行われる設計支援にも関係してくるが，ハードウェア記述言語による記述から，コンパイラ生成に必要な情報を自動的に抽出することが必要である．これらについては，ハードウェア記述言語で述べる．

2.6 ハードウェア記述言語

今までのハードウェア記述言語は，論理/レイアウト合成やシミュレーションを中心に，記述の容易さなどを含めて設計がなされていた．今後のハードウェア記述言語は，今まで考えてきた設計支援を行う必要があり，そのために必要な機能としては，

1 論理/レイアウト合成

2 シミュレーション

3 コンパイラ，OSなどの生成用情報

がある．今までと違うのは，3のコンパイラ，OSなどの生成用の情報が必要と言うことである．

そのためにコンパイラの生成に必要な情報についての考察を行った．コンパイラをフロントエンドとバックエンドに分けて情報をまとめてみると，

フロントエンド：ソースプログラムから中間言語への変換

1 トークンの情報

2 文法の情報

3 中間言語を生成するための規則の情報

バックエンド：中間言語からマシンプログラムへの変換

4 中間言語からマシンコードへの変換規則の情報

5 命令セットの情報(命令の種類・動作，各命令の実行時間など)

6 メモリやレジスタ割り当てのための情報(関数の引数の渡し方，汎用レジスタの構造，スタックの構造，バウンダリなど)

7 最適化の情報(演算器の数，パイプラインの段数など)

高級言語と中間言語を1度決めてしまえば1–3の情報も固定するため，アーキテクチャに応じてコンパイラを生成するには4–7の情報を何らかの形で得る必要がある．ハードウェア記述言語による設計からコンパイラを自動生成するためには，これらの情報を自動抽出する必要がある．そのために，計算機アーキテクチャの設計支援システムのプロトタイプを作成することで，考察を行った．

3 プロトタイプ

計算機アーキテクチャ設計支援システムのプロトタイプは，図3のような構成となっている．実際に論理/レイアウト合成を行いハードウェアの性能評価，また，実際にコンパイラ・ジェネレータを作成しアーキテクチャに応じたコンパイラを生成しプログラムをコンパイルして性能評価を行える環境を作成した[5]．プロトタイプでは，コンパイラ・ジェネレータに与える情報は設計者が与える．対象としているアーキテクチャは，パイプライン・プロセッサやスーパースカラ・プロセッサである．

3.1 論理/レイアウト合成

プロトタイプでは，ハードウェアの評価を行うために定式化などの方法を使わずに，実際に論理/レイアウト合成を行うことで評価を行う．論理/レイアウト合成，シミュレーションを行う部分は，既存のCADツールを利用した．論理合成や動作シミュレーションはPARTHENON[6]やTsutsuji[7]を使用し，レイアウトや論理シミュレーションではSoloやCompassを利用した．これらを組み合わせることで，実際に論理/レイアウト合成を行ってハードウェ

[1]GCCは，コンパイラの自動生成を行うものではなく，マルチターゲットコンパイラである．

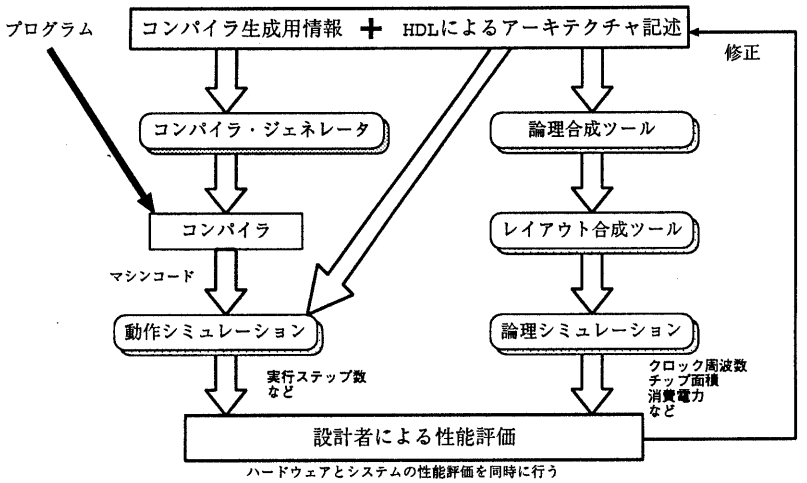

図 3: 計算機アーキテクチャ設計支援システム

アの性能評価を行い，また，動作シミュレーションを行うことでシステムの性能評価を行う．

3.2 コンパイラ・ジェネレータ

システムの性能評価のため，実際にプログラムをコンパイルしてその出力したマシンコードを使用してシミュレーションを行う方法をとる．そのためコンパイラ・ジェネレータのプロトタイプを一部 GCC[8] を利用して作成した．GCC では，一度 C 言語のソースプログラムを *RTL* という中間言語に変換し，その後の部分について情報を与えることで，さまざまなアーキテクチャに応じたコンパイラを生成している．ソース言語からアセンブリプログラムに変換する部分に GCC を利用し，そのあと，リストスケジューリングなどの最適化を行い，マシンコードを生成する．実際に与えるべき情報としては，

4,5 命令セット (RTL との対応，アセンブラを出力する時のニモニック)

6 レジスタの情報 (レジスタの数，割り当て順序など)

6 スタックの情報 (スタックポインタを持つレジスタなど)

6 メモリの情報 (バウンダリ，エンディアンなどのメモリ配置，1 ワードの大きさなど)

7 最適化の情報 (演算器の数，ディレイスロット，パイプラインの段数など)

がある (番号は第 2.6節のコンパイラの生成に必要な情報と対応している)．プロトタイプでは容易に入力できるように，4,5 に関しては実際の命令と仮想的なマシンの命令セットの対応を記述することで GCC で必要となる情報を自動生成を行い，6 についてはテンプレートを用意することで用意に記述できるようにしている．また，7 の情報を用いて最適化を行っている．

3.3 設計のながれ

プロトタイプでは，PARTHENON が採用している SFL というハードウェア記述言語を利用して設計を行うか，Tsutsuji では図形入力を利用して設計を行うとともに，コンパイラ生成用の情報を記述する．ハードウェア記述言語による記述から，既存の CAD を利用して論理/レイアウト合成を自動で行うことで，ハードウェアの評価を行う．それに対して，コンパイラ生成用の情報からコンパイラを生成し，ベンチマークプログラムをコンパイルしてから，シミュレーションすることによってソフトウェアの評価を行う．ハードウェアとシステムの性能評価を同時に行うことで，アーキテクチャに修正を加えていく．

4 問題点

プロトタイプを実現し使用することで，いくつかの問題点が明らかになった．1つは，ハードウェア記述言語による記述からの情報抽出で，もう1つは，コンパイラの生成能力という問題点である．

4.1 情報抽出

プロトタイプではコンパイラを生成するための情報は設計者が与えている．コンパイラを生成するための情報の自動抽出を行うために，既存のハードウェア記述言語によって設計から情報抽出の可能について検討を行った．問題としては，既存のハードウェア記述言語からの情報抽出が困難な点がある．理由の1つは，コンパイラ・ジェネレータのプロトタイプで必要とした情報が明示的に示されてなく，仕様の段階では記述されていた情報などでも実際にハードウェア記述言語の記述になった段階でいくつかの情報は消えている場合がある (例えば，パイプラインの段数など)．他の理由として，設計者がレジスタと記述しても，そのレジスタがどのようなデータを保存するか (例えば，プログラム・カウンタなど) が解っていない場合は，情報抽出が難しい．

これらの問題に対しては，設計者がコンパイラ生成に対する情報をどのように記述するかで2通りの解決法がある．

- コンパイラ生成用の情報を，論理合成やシミュレーション用の記述とは別に (明示的に) 記述する．
- 基本的に論理合成やシミュレーション用の記述に付加情報を加えることで情報を抽出する．

プロトタイプでは別に記述を行ったが，別に記述することの問題点としてはコンパイラ生成用の記述と論理合成やシミュレーション用の記述との間に一貫性が保たれない可能性がある．何らかの形で，2つの記述の間の一貫性を保証する機構が必要である．自動的に一貫性のチェックを行うためには，記述から情報を取り出し比較する必要がある．どちらで行うに場合でも情報を抽出する必要があり，論理合成やシミュレーション用の記述から情報を抽出する技術が必要である．しかし，既存のハードウェア記述言語では，すでに述べたような問題点がある．

我々はこの問題に，既存のハードウェア記述言語に付加情報を記述できるようことで対処する．例えば，レジスタという情報だけでなく，そのレジスタがプログラムカウンタであるという情報を加える．これにより，2つ目の問題点は解消され，それと同時に，ハードウェア記述言語による記述から情報抽出を容易にする．

しかし，既存のハードウェア記述言語の記述体系に新しい記述を加えることは難しいため，実際のインプリメントでは，ハードウェア記述言語でのコメントに付加情報を記述可能とする予定である．

4.2 コンパイラの生成能力

性能評価を正確に行うためには，コンパイラが品質の良いコードを出力する必要がある．そのようなコンパイラを作成するためには，コンパイラ・ジェネレータがアーキテクチャに適したコンパイラを生成する必要がある．

多様な命令セットアーキテクチャに対応してコンパイルすることと，さまざまな構成方式に対応した最適化を行う部分が問題となる．

今回，多様な命令セットに対応するため，仮想的な命令セットとの対応を与えることで実現をした．仮想的な命令セットでコンパイラ生成能力が決定されてため，コンパイラ生成としては問題がある．これに関しては，A.V.Aho らによって提案されている Tree Rewriting[9] という手法を使って対応する予定である．

また，さまざまな構成方式 (パイプライン，スーパースカラ，分岐予測など) に対して，どのような最適化を行うかは難しい問題てある．現段階では，予めあるアーキテクチャに対してはこの最適化手法を用いるなどと指示を与えておく方法を考えているが，自動的にアーキテクチャに応じた最適化手法を選ぶ方法などの研究も必要である．

5 今後の検討課題

現段階では，コンパイラの自動生成を行うコンパイラ・ジェネレータや計算機アーキテクチャ向けのハードウェア記述言語とその記述からの情報抽出について研究を行っている．しかし，これだけでは計算機アーキテクチャの設計支援システムとしては，不十分である．今後，今まで人間が行ってきた部分 (例えば，パイプライン設計など) に対する支援が必要となる．

現在，性能評価を行う部分に関しては，実行の履歴などから設計者がボトルネックを発見している．さまざまな要因が考えられる場合には，ボトルネックの発見が困難な場合がある．そのため，プログラムをコンパイルしてシミュレーションを行う場合などに，その結果を設計者に解りやすい形で出力したり，さらには，ボトルネックとなっている部分の発見などを行う手法などについての研究も必要である．

最近の設計規模の増大，設計回路の複雑化のために，検証に対する支援が重要となっている．アーキテクチャ設計段階での誤りは，以後の設計すべてに影響を与えるために，早期の段階での検証が必要である．特に，ハードウェア/ソフトウェアのインターフェイスの部分に関しての検証を考えていく必要がある．

6 まとめ

本論文では，計算機アーキテクチャの設計について考察を行い，現在必要とされている支援について考察を行った．計算機アーキテクチャの評価では，ハードウェアとシステムの両面からの評価が必要であり，今までの論理/レイアウト合成やシミュレーションだけでなく，コンパイラ・ジェネレータが必要となる．

コンパイラ・ジェネレータによりハードウェア/ソフトウェアのインターフェイスとなるコンパイラを生成し，システムの評価を行う．コンパイラを自動的に生成するためには，コンパイラ生成用の情報を自動抽出する必要がある．既存のハードウェア記述言語で記述する情報は論理/レイアウト合成には十分であっても，コンパイラを生成する情報は不十分である．そのため，既存のハードウェア記述言語による記述だけでなく，付加情報を加える必要がある．

将来的には，情報抽出や性能評価結果から改善点などを示す支援やアーキテクチャレベルでの検証などが必要である．

謝辞

Tsutsuji の提供など御協力頂く YHP の皆様，パルテノン (PARTHENON) の提供など協力頂く NTT コミュニケーション科学研究所の皆様，Compass の提供など御協力頂く (株) ソリトンシステムズの皆様，Solo の提供など御協力頂く (株) テクノアライアンスの皆様に感謝致します．また，日頃から討論頂く村上和彰講師，中村秀一氏，冨山宏之氏，および，安浦研究室の皆様に感謝致します．この研究は一部，京都高度研究所の新産学交流事業 EAGL の支援による．

参考文献

[1] 中村 秀一，安浦 寛人，“専用システムのための HW/SW 協調再設計の一手法，” 情処研報，93–DA–65–13, pp97–104, 1993 年 1 月．

[2] 高橋 瑞樹，山口 雅之，野田 浩明，竹田 信弘，藤本 徹哉，神戸 尚志，“トップダウン設計方式の信号処理 LSI 設計への適用，” *DA* シンポジウム *'93* 論文集，pp105–108, 1993 年 8 月．

[3] 中田 武治，佐藤 淳，塩見 彰睦，今井 正治，引地 信之，“AISP 向けハードウェア/ソフトウェアコデザインシステム PEAS-I のハードウェア生成系，” *DA* シンポジウム *'93* 論文集，pp17–20, 1993 年 8 月．

[4] 芳野 泰成, 清水 嗣雄，“設計初期における概略ディレイ評価の一手法”, *DA* シンポジウム *'93* 論文集，pp.101–104, 1993 年 8 月．

[5] 赤星博輝，安浦寛人，“アーキテクチャ評価用ワークベンチ —コンパイラの自動生成—，” 電子情報通信学会技術研究報告, VLD–92–84, 1993 年 1 月．

[6] “PARTHENON User's Manual,”NTT データ通信株式会社

[7] “ASIC Design System Users's Manual,” 株式会社 YHP システム技術研究所

[8] Richard M.Stallman: “Using and Porting GNU CC for version 1.37.1,”Free Software Foundation,INC, Feb.1990

[9] A. V. Aho, M. Ganapthi, S. W. K. Tjiang, “Code Generation U sing Tree Matching and Dynamic Programming,” *ACM Transactions on Programming Languages and Systems,* Vol.11, No.4, pp.491–516, Oct 1989.