設 計 自 動 化 76−10
（1995．7．20）

ASIP 設計用コデザイン・ワークベンチ PEAS-III の提案

塩見 彰睦† 今井 正治† 片岡 健二†
青山 義弘* 佐藤 淳‡ 引地 信之§

† 豊橋技術科学大学
* 福井工業高等専門学校
‡ 鶴岡工業高等専門学校
§ SRA

あらまし
本稿では ASIP 設計を行うための，HW/SW コデザイン・ワークベンチ PEAS-III を提案する．設計者は，PEAS-III を用いて対話的にアーキテクチャを選択し，リソースのパラメータを設定し，マイクロ動作を記述することによってプロセッサの設計を行う．さらに設計結果の性能見積り・評価および短期試作も可能となる．本稿では，PEAS-III の構想，設計支援の方針，開発すべき要素技術について述べる．

和文キーワード コデザイン，設計支援，ASIP

Proposal of a Codesign Workbench PEAS-III for ASIP Design

Akichika SHIOMI† Masaharu IMAI† Kenji KATAOKA†
Yoshihiro AOYAMA* Jun SATO‡ Nobuyuki HIKICHI§

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

Abstract
This paper proposes a HW/SW codesign workbench PEAS-III to design an ASIP. Using the workbench with the dialog interaction, a designer can select an architecture type, assign resource parameters, and describe micro operations of the target processor. This workbench will enable the designer to estimate the hardware cost, performance of the designed ASIP in a short TAT (Turn Around Time). This paper also describe a idea of PEAS-III, a strategy for design support, and element any technologies to be developed to realize PEAS-III.

Key words Codesign, Design support, ASIP

1 はじめに

今後のVLSI設計者の仕事は最適なアーキテクチャの設計となるであろう．アーキテクチャ設計を効率良く支援するために，これまでコデザイン手法，高位合成，ESDA(Electronic System Design Automation)ツールの研究が進められてきた[1][2][3]．しかし，システムの最適設計を自動的に行う一般的な手法は，現在の所明らかにされていない．結局，システムが複雑な場合には試行錯誤が必要となる．

その一方で，設計者にはシステムが大規模・複雑化するにも関わらず，システムを短時間に誤りなく設計することが要請されている．さらに，設計者は小型化，性能向上，機能強化などの多様な設計目標を達成しなければならない．

筆者らがこれまでに開発したPEAS-Iシステム[4]は，与えられた応用プログラムの解析結果に基づいて最適な命令セットを持つCPUコアのHDL記述を生成する．また，これと同時に，コンパイラおよび命令レベルシミュレータなどの応用プログラム開発ツールも同時に生成する．PEAS-Iでは，CPUのアーキテクチャ・モデルに制約を加えているため，比較的容易に最適化が行えていた．また，応用プログラム開発ツールの自動生成も比較的容易に実現できた．

PEAS-IIIでは，設計の自由度がさらに大きい，システム・アーキテクチャレベルの記述からシステムの機能検証・性能等の予測・自動生成を短時間に効率良く行なうワークベンチを目指す．PEAS-IIIは，アーキテクチャの選択のような上流設計工程を支援することで設計の試行錯誤にかかるTAT(Turn Around Time)を短縮し，生産性や設計効率の向上をはかることを目的とする．

本稿では，設計選択の自由度の大きいシステム・アーキテクチャレベルから効率良く設計を支援するためのASIP (Application Specific Integrated Processor)設計用コデザイン・ワークベンチPEAS-IIIを提案する．本稿の構成は以下の通りである．

まず，2節ではPEAS-IIIの概要について述べ，次に3節ではPEAS-IIIのシステム構成，特にアーキテクチャ記述入力系によるCPUの設計方法を中心に説明する．4節ではPEAS-IIIで必要とされる機能検証，性能等の予測，自動生成の3つの要素技術とそれらが抱える問題点および今後の課題について述べる．

2 PEAS-IIIの概要

PEAS-IIIは，コデザインの主要な目標[5]である(1)Rapid Protoryping, (2)Precise Estimation, (3)Short TATを実現するASIP設計ワークベンチである．PEAS-Iでも，上記の3つの項目を実現していた．しかし，CPUの命令セットの最適化のため，アーキテクチャやパイプラインの段数は固定され，取り扱う命令もC言語の演算子レベルの命令に限られていた．PEAS-IIIでは，アーキテクチャ，パイプラインの段数，命令セットなどに柔軟に対応し，設計者の経験や直観による創意工夫を採り入れたCPUの評価，ハードウェアの自動生成およびソフトウェア開発環境の自動生成を目指している．

設計者によって与えられるCPUのアーキテクチャ・パラメータには以下のものが考えられる[6]．

(1) **アーキテクチャ**
ノンパイプライン，パイプライン，スーパーパイプライン，スーパースカラ，VLIWなど

(2) **パイプラインの構成**
段数，命令キューの有無および大きさ，インタロック機構の有無，パイプライン・バイパス，など

(3) **バスの構成**
外部／内部バスの幅，本数，専用バスの有無など

(4) **レジスタ構成**
汎用レジスタの本数，専用レジスタの種類及び本数，レジスタファイルの構成，レジスタの特殊機能など

(5) **演算器の種類**
ALU，バレルシフタ，乗除算器などの種類および実装個数，演算サイクル数，ユーザ定義の演算器，周辺回路の有無など

(6) **その他**
キャッシュの有無・大きさ・方式，割り込みの制御，IO命令，命令セットのモードなど

このように，設計者が指定／変更できる選択肢はアーキテクチャ，データパス，ハードウェア・リソースなど多岐にわたる．PEAS-IIIでは，GUI (Graphical User Interface)を用いこれらのパラメータをインタラクティブに選択・設定する．その後，機能検

証，評価を繰り返しながら設計を進めることを考えている．

しかし，設計者が，全てのパラメータの値を短時間で，最適に指定することは難しいと思われる．そこで，CPU の設計を容易に行なうために，基本的な CPU のモデルをあらかじめ複数用意する．設計者はこれらの基本モデルの選択肢を変更して，CPU を容易に設計することができる．

既存の汎用 CPU のモデルが用意できれば，それとそれに変更を加えた CPU の性能などを比較することが可能である．これは，システムを設計する上で，既存の汎用 CPU を利用するか，新規に ASIP を開発するべきかを判断するための有益な情報の一つとなる．また，応用プログラム解析系と組み合わせることで，既存の CPU に対して，応用プログラムに合わせた CPU の変更を，容易に行なうことができると考えられる．

さらに，試行錯誤を容易に繰り返すことができるため，設計上のノウハウを短期間で修得することも可能であろう．ハードウェア・リソースや処理方式の違いによる性能の違いを GUI を用いて効果的に示すことができれば，アーキテクチャ教育など教育の分野での利用も期待できる．

3 PEAS-III の構成

PEAS-III のシステム構成を図 1に示す．PEAS-III も PEAS-I と同様，応用プログラムとデータを解析し，最終的にハードウェア記述とコンパイラ等の応用プログラム開発ツールを自動生成する．PEAS-I と異なるのは，応用プログラムとデータの解析結果に基づいてパラメータを設計者が入力し，設計したプロセッサの性能予測を繰り返しながら，アーキテクチャの比較・検討ができる点である．

従って，PEAS-III の核は，アーキテクチャ記述入力系と予測系である．その他の応用プログラム解析系や応用プログラム開発ツール生成系，ハードウェア生成系は，PEAS-I で開発された技術を応用することで一部実現することが可能と考えられる．今後開発すべき要素技術的については 4 節で詳しく述べる．本章では，特に PEAS-III の核となるアーキテクチャ記述入力系に説明の重点を置く．

アーキテクチャ記述入力系への入力は GUI を用い，生成されるプロセッサのアーキテクチャ・タイプ，ハードウェア・リソース，命令の動作などの情報を入力・変更しながら設計を進める．これらの情報

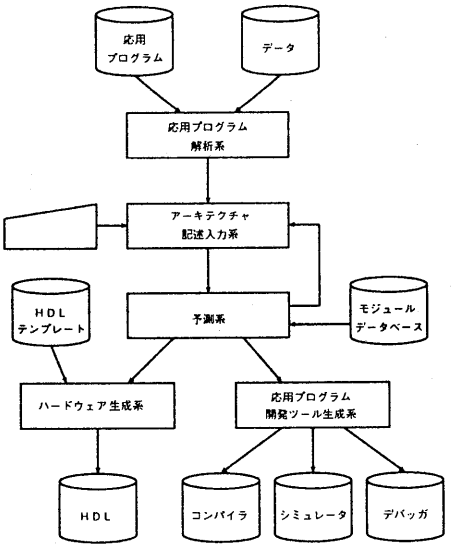

図 1: PEAS-III の構成

はそれぞれ独立に以下の表や図の形で管理される．

(1) ハードウェア・リソース表
使用する演算器，レジスタ，バス等のリソースを選択し，それぞれのリソースの名前を宣言する．ここで宣言したリソース名を用いてマイクロ動作を記述する

(2) マイクロ動作記述表
選択したリソースとあらかじめ定義されている演算子や手続きを用いて，各命令のマイクロ動作を表形式で入力する

(3) 接続情報記述
リソース及びリソース間の転送経路等の接続の情報を入力する

これらの 3 つの情報に関しては次節以降で詳細に述べる．

性能等の予測および HDL の自動生成は，これらの情報とモジュール・データベースから必要な情報を抽出することで可能となる．

PEAS-III の試作では，ハードウェア記述に用いる HDL として VHDL[7] を用いる予定である．演算器の HDL 記述は，ビット幅などをパラメタライズしなければならず，かつそれが論理合成可能な記述でなければならない．VHDL を用いることによ

り，このような記述が比較的容易に実現できると考えられる．

3.1 ハードウェア・リソース表

マイクロ動作記述表で使用するハードウェア・リソースはハードウェア・リソース表で定義する．2節で述べた設計者によって与えられるパラメータの全てが，ここで定義される．設計者はアーキテクチャを選択し，さらに使用するハードウェア・リソースの仕様を決定する．ハードウェア・リソースの仕様には，機能，型，属性，実行サイクル数，ビット幅，使用個数，インスタンス名等がある．

設計者は，ハードウェア・リソースの記述の際に，パイプライン・アーキテクチャを特に意識する必要はない．PEAS-IIIは，パイプライン・アーキテクチャで複製が必要となるリソース・インスタンスを検出し，複製を生成し構造ハザードを回避する．

以下ではハードウェア・リソース表を単にリソース表と呼ぶ．

3.2 マイクロ動作記述表

マイクロ動作記述表は，選択されたリソースとあらかじめ定義されている演算器等を用いて，命令ごとにプロセッサ内部での細かな処理手順を記述されたものである．

マイクロ動作の記述例を図2に示す．マイクロ動作は，はじめに細粒度のフェーズ単位で記述され，同一フェーズ内の記述は，ステートやステージで分割することはできない．ステートは複数のフェーズからなり，外部から見た1クロックでの動作に相当する．ステージは複数のステートの集まりである．

ステートの分割点はフェーズの区切りのどれかであり，ステージの分割点はステートの区切りのどれかになる．これは設計者が自由に設定することができる．これによりパイプラインのステージ分割を自由に行なうことができる．設計者がステートやステージの区切りを指定すると，図3に示すように，マイクロ動作の粒度を変えて表示することができる．

マイクロ動作記述表中に記述する動作の種類は，以下の4種類である．

(1) リソース間のデータ転送

(2) 演算器の起動

(3) メモリアクセス

(4) 条件判断

マイクロ動作記述表の情報は，主に構造ハザードやデータハザードの検出とコンパイラの自動生成のために用いられる．ここで検出されたハザードの状況に応じて，インターロック制御回路やインスタンスを生成し，ハザードを回避する．

3.3 接続情報記述

接続情報記述は，リソース間の転送経路を示す情報である．いわゆるブロック図に相当する．設計者に対して，現在設計中のアーキテクチャを視覚的に表示する必要がある．

接続情報記述はリソース表とマイクロ動作記述表から抽出することも可能である．接続情報記述は2つのアプローチで得られる．

(1) リソース表およびマイクロ動作記述から，高位合成技術を用いて自動抽出する

(2) 接続情報は設計者が入力し，検証時にリソース表およびマイクロ動作記述表との整合性をチェックする

4 PEAS-III を実現するための要素技術

PEAS-Iではアーキテクチャモデルを固定し，命令セットの最適化を行なっていた．しかし，PEAS-IIIでは，アーキテクチャ・モデルまで可変にするため，PEAS-Iでは取り扱わなかった問題が生じる．

本節では，PEAS-IIIを実現するために必要な要素技術と解決すべき課題について述べる．

4.1 機能検証

機能検証は，プロセッサを構成する構成要素間でのインターフェース及び全体での機能検証と設計者が定義した演算器などの構成要素単体の機能検証に大別される．

4.1.1 設計記述の整合性の検証

まず，設計者によって入力されたリソース，マイクロ動作，接続情報の3つの情報に矛盾がないことを検証しなければならない．

Phase	1	2	3	4	...
LD	BUS1 <= PC IMAR <= BUS1 INC(PC)	IMEM(READ) IR <= IMDR	DECODE(IR) REGF(RD,R2)	BUS1 <= REGF.OUT IMAR <= BUS1	...
:	:	:	:	:	

図 2: マイクロ動作の細粒度記述の例

Stage	IF	ID&DF	EX	WB
LD	BUS1 <= PC IMAR <= BUS1 IMEM(READ) IR <= IMDR INC(PC)	DECODE(IR) REGF(RD,R2) BUS1 <= REGF.OUT IMAR <= BUS1	BUS2 <= IMDR DMAR <= BUS2 DMEM(READ)	BUS2 <= DMDR REGF(WR)
:	:	:	:	:

図 3: マイクロ動作のステージ単位での表示例

(1) リソース
リソース表では，定義された演算器のビット幅などの不一致や，排他的に用いられるべき指定の検証が行なわれなければならない．また，リソースのインターフェースと接続情報が一致していなければならない．

(2) マイクロ動作
マイクロ動作記述表で使用されるリソースの競合やビット幅などの不一致，ステージやステートの分割点の正当性の検証が必要となる．また，リソース表で定義されたリソースをマイクロ動作に正しく使用しているかの検証も必要となる．

(3) 接続情報
接続情報では，構成要素間のインタフェースに矛盾がないことを検証しなければならない．また，マイクロ動作記述表で明示されている転送と接続情報に矛盾がないことも検証する必要がある．

このような検証を効率良く洩れなく行なうための管理情報および検証アルゴリズムを確立する必要がある．これらの検証では，検証の条件がリソースやマイクロ動作数の組合せになってしまうことは明白である．条件の依存関係を管理するために論理的なハイパーリンク構造を実現し，これらの関係を効率良く全探索するアルゴリズムが必要となる．

4.1.2 ハザードの検出と回避

マイクロ動作記述表で記述された動作記述中で，リソースの競合が発生する可能性がある．例えば，オペランド計算と演算を同一ステートに割り当てた場合，ALU に競合が生じる．

また，パイプライン・アーキテクチャを選択した場合，マイクロ動作記述表の命令間に構造ハザードやデータ・ハザードが生じプロセッサの性能が低下する．この場合，設計者がマイクロ動作や接続情報を変更することでハザードを軽減することも可能である．しかし，根本的に解消できない場合には，プロセッサの信頼性を向上させるためにインターロック制御回路の自動生成技術が必要となる．

このように，命令のマイクロ動作の設計に問題があることを検出し，設計者にフィードバックし，必要であればプロセッサにインターロック制御回路を付加し，ハザードを回避しなければならない．これらのハザードの検出は，パイプライン・プロセッサにおけるハザードの検出および解消に関する研究成果 [8][9] が利用できると考えられる．

さらに，ハザードの回避方法が複数考えられる時，ハードウェア・コストや性能のトレードオフを考慮して最適な回避方法を選択するアルゴリズムの確立も必要となるであろう．そのためには，ハザードの回避方法のモデル化および問題の定式化が必要となる．

4.1.3 構成要素の記述と検証

演算器では，その入出力ビット幅，実行サイクル数などが設計者の指定により様々に変化する．ハードウェアの自動生成を考えた時，この指定にあった演算器をどのように生成するかが問題となる．自動生成では以下のアプローチが考えられる．

- パラメータに合わせて HDL 記述を生成するジェネレータの実現
- HDL 記述自体をパラメタライズし，パラメータを与えて合成

これらのアプローチのどちらを採用するかは，実現する構成要素の特性に合わせて選択すれば良い．

しかし，最終的に生成された記述の機能検証が必要となる．ここで生成するたびに個々の記述を検証するのでは効率が悪い．できればHDL 記述の生成アルゴリズムの検証，パラメタライズされた記述自体の検証技術を確立し，最終的な生成物の検証を省略したい．いわゆるメタな記述に対する機能検証が今後の大きな課題となる．

4.1.4 設計全体の検証

これまで個々の工程での検証について述べたが，設計全体の機能検証も必要である．全体の検証を効率良く行なうために，ボトムアップまたはトップダウンな統合的な検証環境およびアルゴリズムの開発が必要である．

4.1.5 テスト生成

また，BIST(Built-in Self-test) 回路の自動生成やATPG(Automatic Test Pattern Generator) の実現 [10] も今後の課題となる．

4.2 諸元の予測

設計したプロセッサの諸元としては，PEAS-I でも採用している (1) 応用プログラムの実行サイクル数，(2) ハードウェア・コスト，(3) 消費電力の 3 つがある．これらの評価項目を短時間で精確に見積もることがPEAS-III でも重要な課題となる．

4.2.1 性能

設計したプロセッサの性能は，実行サイクル数とクロック・レートを掛けた，時間で評価する．実行サイクル数は，オブジェクトコードの品質，演算器の有無，ハザードの頻度および回避方法に大きく依存する．本質的にはPEAS-I で用いている演算器で処理される実行サイクル数と実行頻度から予測することが可能であると考えられる．

一方，クロック・レートは，論理合成結果の品質，使用するライブラリ，レイアウトなどに大きく依存する．正確なクロック・レートはレイアウト後の設計で評価しなければならない．しかし，PEAS-III では，一次近似として論理合成の結果からクロック・レートを予測する．正確な予測手法は今後の課題である．

また，PEAS-I ではキャッシュを取り扱っていない．キャッシュがある場合，メモリアクセスにかかるサイクル数が実行サイクル数に大きく影響する．正確な実行サイクル数を予測する場合，応用プログラムとデータ及びキャッシュの大きさと方式からキャッシュのヒット率を予測する必要がある．

4.2.2 ハードウェア・コスト

PEAS-III では，PEAS-I と同様に構成要素それぞれのハードウェア・コストの単純和を用いることを考えている．PEAS-I では，演算器にその制御論理も含まれていたため，高精度で回路規模を予測することができた [11]．制御論理を構成要素に含めて制御論理の構造化および階層化をはかることは，自動生成，ハードウェア・コストの予測，設計の再利用の観点から有効な手法である．

ハードウェア・コストは，ゲートアレイやスタンダードセルといった ASIP の実現方法にも依存する．PEAS-III では，それぞれの実現方法に応じたモジュール・データベースを用意することで対応する．

4.2.3 消費電力

消費電力の予測に関しては，与えられたパラメータ，応用プログラムおよびデータを解析することで消費電力の精確な物理量を予測することが理想であ

る．しかし，消費電力は，使用するライブラリやレイアウト，処理するデータに依存する [12][13].

PEAS-I では，モジュール単体で合成し，論理合成時に得られた消費電力がモジュールデータ・ベースに登録されている．すなわち，レイアウトや処理されるデータについては考慮されていない．また，PEAS-III では，モジュールのビット幅や実行サイクルが可変となるため，全ての組合せについて論理合成をしてデータベースを構築する手法は現実的ではない．ライブラリの情報に基づいて，モジュールのパラメータと処理するデータの特徴から必要な精度で消費電力を推定する手法を確立する必要がある．この方法によっても複数のアーキテクチャの候補の消費電力の相対的な比較が可能であると期待される．厳密な消費電力の評価はレイアウト後に行なう必要がある．

4.3 HDL 記述の自動生成

最終的に設計者が与えたパラメータやマイクロ動作記述表にしたがって合成可能なプロセッサの HDL 記述を自動生成しなければならない．HDL の自動生成のアプローチは 4.1節で述べたように，(1) ジェネレータを実現する方法と (2) パラメタライズした VHDL の記述を用意する方法が考えられる．

(1) の方法では，外部からパラメータだけを与えて合成可能な HDL 記述を生成するアルゴリズムを確立しなければならない．規則性のある記述であれば容易に自動生成できるが，変則的に変化する記述の場合，その自動生成アルゴリズムは発見的アルゴリズムによるところが大きいと考えられる．

また (2) の方法については記述のガイドラインが文献 [14] に示してあるものの，論理合成ツールはベンダによって，受け入れる記述の範囲が異なるため，どこまでパラメタライズした記述が可能かを調査・検討する必要がある．

4.4 応用プログラム開発環境の自動生成

アーキテクチャ設計支援の立場からコンパイラの自動生成に関する研究が様々な角度から行なわれている [15][16][17]. PEAS-I で用いたコンパイラの自動生成方法は，GCC の想定している仮想マシンの命令セットに CPU の命令セットを合わせている [18] ため，PEAS-III で扱うような様々な命令セットを持つプロセッサに柔軟に対応することができない．PEAS-III では，コンパイラはマイクロ動作記述表に記述されている命令の意味を考慮してコード生成を行なわなければならない．赤星らは，HDL 記述からコンパイラに必要な情報を抽出して，コンパイラを自動生成する方法を提案している [15][16]. PEAS-III では，コンパイラの生成に必要な情報はリソース表，マイクロ動作記述表中にあるため，HDL 記述から情報を抽出する必要はない．したがって，PEAS-III では，赤星らの方法よりも容易にコンパイラの自動生成が行なわれると期待される．

しかし，ハードウェアとして実現されない演算は実現されている命令の組合せで実現しなければならない．そして，その組合せはできる限り最適である必要がある．これを実現するためには，与えられた機能を満たす最短命令列を探す関数列生成ツール superopt として実現されている，Granlund らの研究成果 [19] を利用できると考えられる．

5 まとめ

本稿では，ASIP 設計用コデザイン・ワークベンチとしての PEAS-III の構想を提案した．PEAS-III はシステム・アーキテクチャレベルからの設計を支援し，機能検証・性能等の予測・ハードウェアおよびソフトウェア開発環境の自動生成を行い，(1)Rapid Protoryping, (2)Precise Estimation, (3)Short TAT を実現することを示した．

PEAS-III では試行錯誤を容易に繰り返すことができるため，設計上のノウハウを短期間に修得できる効果も期待できる．さらにアーキテクチャなどの違いによる性能差を示すことで，アーキテクチャ教育などの教育の分野での利用も期待できる．

さらに，PEAS-III を支える要素技術と実現に向けての課題について述べた．HDL の自動生成や論理合成など今後の研究成果を待つ必要があるが，その他の課題については，これまで PEAS-I でのアルゴリズムや手法を拡張し適用することで，ある程度解決できると考えられる．

本稿では触れなかったが，最終的にはプロセッサ全体の最適化を行なう必要がある．プロセッサ全体の最適化をシステマティックに行うためには，それぞれのモジュールの複数の実現方法の組合せを考慮してプロセッサ全体の最適化を行わなければならない．これらの問題の多くは，組合せ最適化問題（線形整数計画問題）として定式化出来るが，この種の問題の多くは NP 困難である．この問題を効率良く解くためには，分枝限定法（Branch-and-Bound

法）や動的計画法（Dynamic Programming）に基づく発見的な手法を用いて準最適解を求める方法が有効であると考えられる．

また，これまでの研究では，専用周辺回路のコデザインと専用CPUコアのコデザインのそれぞれが比較的独立して進められてきた．しかし，応用分野の性質は各種各様であり，どちらか一方の構成要素の最適化だけではシステム全体の最適化を達成することは困難であると考えられる．したがって今後は，これら両者の構成要素を統合し同時に最適化する新しい手法が必要になると思われる．これらはいずれも今後の課題である．

謝辞

本研究を進めるにあたり，御助言をして下さった豊田工業高等専門学校の木村勉 助手ならびに豊橋技術科学大学 今井研究室の諸兄に感謝致します．また，研究環境を御提供頂く，(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 の研究助成による．

参考文献

[1] R.K. Gupta, C.N. Coelho, and G. De Micheli: "Program Implementation Schemes for Hardware-Software Systems," *IEEE, Computer*, Vol. 27, No. 1, pp. 48–56, January 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] "特集 100万ゲート ASIC 設計用 EDA", 日経エレクトロニクス, 1994 8-1, pp.53–78, 1994.

[4] 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.

[5] 今井 正治, "ハードウェアの見積りと生成", 情報処理学会誌「ハードウェア／ソフトウェア・コデザイン」小特集, Vol.36, No.7 (掲載予定), 1995.

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

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

[8] 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

[9] Sofiene Tahar and Ramayya Kummar, "Towards a Methodology for the Formal Hierarchical Verification of RISC Processors," Proceedings of 1993 IEEE International Conference on Computer Design: VLSI in Computers & Processors(ICCD '93), pp.58–62. 1993.

[10] 今井正治 編著, "ASIC 技術の基礎と応用", 電子情報通信学会, 1994

[11] A.Y. Alomary, M. Imai, and N. Hikichi: "An ASIP Instruction Set Optimization Algorithm with Functional Module Sharing Constraint," *Trans. IEICE*, Vol. E76-A, No. 10, pp. 1713-1720, October 1993.

[12] 木村勉, 山本俊之, 満田千秋, 塩見彰睦, 今井正治, 引地信之, "VHDL を用いた論理回路の消費電力見積りの方法の提案 ～32 ビットアダー回路の消費電力についての考察～", 情報処理学会研究報告 95-DA-75, pp.31–38, 1995.

[13] P. E. Landman and J. M. Rabaey , "Architectural Power Analysis: The Dual Bit Type Method," IEEE Trans. on VLSI SYSTEMS, Vol.3 No.2 , pp.173–187, IEEE, 1995

[14] E. Villa and A. Debreil, "Synthesis and Formal Proof Language Support," CENELEC TC117, Microelectronics Group of the University of Cantabira, September 1994.

[15] 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 —", 信学技報, VLD92-86, pp.9-16 , 1993.

[16] 冨山 宏之, 赤星博輝, 安浦寛人, "アーキテクチャ評価用コンパイラ・ジェネレータの評価", DA シンポジウム '94 論文集, pp.193–198, 1994.

[17] P. Marwedel, "Tree-Based Mapping of Algorithms to Predefined Structures," IEEE/ACM International Conference on Computer-Aided Design, pp.586-593, 1993

[18] 博多哲也, 佐藤淳, A.Y.Alomary, 今井正治, 引地信之, "ASIC CPU 向きソフトウェア開発環境生成系の実現", 信学技報, VLD92-25, pp.43-48, 1992.

[19] T. Granlund and R. Kenner, "Eliminating Branches using a Superoptimizer and the GNU C Compiler," superopt, ACM SIGPLAN NOTICES, Vol.27 No.7, pp.341–352,1992.