計算機アーキテクチャ　115－14
設　計　自　動　化　78－14
（１９９５．　１２．　１４）

演算資源構成にもとづくアーキテクチャ評価の一手法

山口 雅之　山田 晃久　中岡 敏博　神戸 尚志
{masa, yamada, toshi, kambe}@edag.ptdg.sharp.co.jp

シャープ株式会社 精密技術開発センター
〒632 奈良県天理市櫟本町 2613-1
TEL: 07436-5-2531　FAX: 07436-5-4968

あらまし

本稿では、プログラム組み込み方式LSIのアーキテクチャ設計において、演算資源とその構成 / 制御、並びにそれらをソフトウエアで制御するための命令セット決定を、コスト / 性能 / 消費電力について応用プログラム向きに最適化することを目的とした評価手法を提案する。提案手法では演算資源構成の詳細や命令セットは制約として表現し、演算資源の概略と処理アルゴリズムのみにもとづくモデルを用いて評価を行なう。評価は静的解析と動的解析を組み合わせることで個々の演算 / 転送資源の利用度、ボトルネックを精度高く評価し、カスタムDSPのハードウエア構成及び命令セットの最適化を支援可能である。本稿では、この手法にもとづく処理ステップ数予測によるアーキテクチャの性能評価を中心に考察する。さらに、提案手法を用いたアーキテクチャ設計支援システムを実際の音声処理LSIに適用して、その効果と課題を考察する。

キーワード アーキテクチャ、性能評価、演算資源構成、並列制約

An Architecture Evaluation Method based on the Structure of Datapath Resources

Masayuki YAMAGUCHI　Akihisa YAMADA　Toshihiro NAKAOKA　Takashi KAMBE
{masa, yamada, toshi, kambe}@edag.ptdg.sharp.co.jp

Precision Technology Development Center, SHARP Corporation
2613-1 Ichinomoto, Tenri, Nara 632, Japan
TEL: +81-7436-5-2531　FAX: +81-7436-5-4968

Abstract

In this paper, an evaluation method in architecture level for embedded LSIs is presented. The proposed method aims at the application specific optimization of the structure/control of datapath and the instruction set architecture in terms of cost/performance/power. The evaluation madel is based on only the datapath structure and the application algorithm and represents a detailed structure and/or the instruction set as constraints. This method is capable of estimating the utilization of operation/transfer resources and the bottleneck accurately, and supporting the datapath design and the instruction set design of custom DSPs. In this paper, we focus on the performance evaluation by using the number of execution steps. We apply this method to an actual design of an audio signal processor, compare with the conventional method, and discuss its accuracy of the estimated number of the execution steps.

Key words architecture, performance evaluation, datapath structure, parallel constraint

1 はじめに

システムの多様化や寿命短期化が進む中で、目的に応じてコスト性能比を追求したLSIの短期開発の必要性が高まっている。これには、アーキテクチャ設計段階から設計が要求仕様を満たすかどうか判断してシステムを最適化し、設計の後戻りをなくすことが望まれている。本稿では、アーキテクチャ設計において、演算資源とその構成 / 制御、並びにそれらをソフトウエアで制御するための命令セット決定を、コスト / 性能 / 消費電力について応用プログラム向きに最適化することを目的とした評価の一手法を提案し、特にこの手法にもとづくアーキテクチャの性能評価を中心に考察する。

アーキテクチャ性能評価はアーキテクチャの最適化を目指したハードウエア・ソフトウエア協調設計においても重要な要素技術である。プログラム組み込み方式LSIの協調設計では、要求仕様として与えられる応用プログラムを解析して性能評価を行い、目的に応じた最適化を行なうことが多い[1-9]。従来のアーキテクチャ評価手法としては、命令セットを用いた手法があげられる[1,8]。この手法では命令セットを含むアーキテクチャ情報からコンパイラを生成して応用プログラムを命令レベルで実行し、処理ステップ数を求めることで性能評価を行なう。この手法は正確な評価ができる反面、命令セットを完全に設計しなければ利用できないため、命令セット自身の最適化に利用することは困難である。一方、特に命令セットを用いずに応用プログラム解析を行うシステムも研究されている[4,7,9]。[4, 7] は命令セット合成やハード / ソフト分割の最適化を目的としたシステムであり、[9] は応用プログラム解析により入力データによる処理ステップ数の上界と下界を求める性能解析システムである。これらのシステムは処理ステップ数自身の正確な予測は目的とせず、転送資源に起因するデータ衝突も考慮しない。しかし、DSP等の設計では目的とする処理と最大動作周波数の関係から処理ステップ数が設計目標として与えられる。そのため、正確な処理ステップ数予測は重要であり、転送資源におけるデータ衝突が処理ステップ数を増加させる場合もあるため、データ転送が評価できることが望まれる。

本稿では、演算資源構成にもとづくモデルと、モデル上での応用プログラム解析を用いた性能評価手法を提案する。提案手法では演算資源構成の詳細や命令セットは制約として表現し、演算資源の概略と処理アルゴリズムのみにもとづくモデルを用いて評価を行なう。演算資源構成はアーキテクチャ設計初期段階で決定される設計項目の1つであるが、資源利用とデータの流れを予測し、性能を評価する十分な情報を持つ点に我々は注目している。また、本手法は静的解析だけでなく、動的解析をも組み合わせることで個々の演算 / 転送資源の利用度、ボトルネックを精度高く評価し、これによって、カスタムDSPのハードウエア構成及び命令セットの最適化を支援可能である。さらに、コンパイラのように処理を実行する正確な命令コードを生成する必要はないため、全体の解析は直接応用プログラムを用い、簡略化を用いた解析手法を用いることで評価時間を短縮できる。また、演算資源の並列性の制約情報を付加することで、制御系や命令セット設計を考慮できるという特徴を持つ。本手法はキャッシュや外部RAMは考慮していないが、設計対象としてDSPなどデータ処理が支配的であるLSIには適した手法と思われる。

我々は、提案手法にもとづくアーキテクチャ設計支援システムを開発し、実際に開発された音声信号処理LSIに適用した例を示すことで提案手法の有効性と課題を考察する。

2 モデル

本節では、アーキテクチャ評価に用いるモデルについて説明する。モデルの概略を図1に示す。本手法ではハードウエアとソフトウエアをモデル化し、ハードウエアモデル上でのソフトウエアモデルの実行シミュレーションにより評価を行なう。

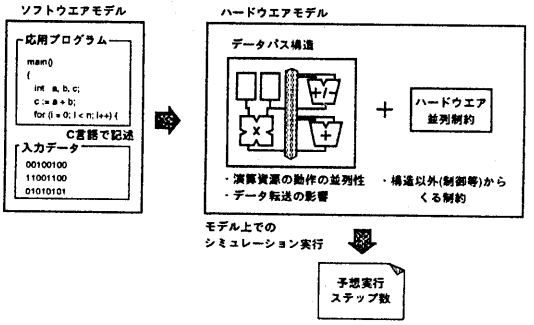

図 1: 性能評価手法のモデル

2.1 ハードウエアモデル

本手法では、以下の(a)、(b)でハードウエアをモデル化する。

(a) データパス構造

(b) ハードウエア並列制約

(a)はデータパス部の演算資源構成である。資源の集合R、資源(のポート)間の接続の集合I、資源RからライブラリLへのマッピング関数$f : R \rightarrow L$の3項組(R, I, f)で表す。Iには処理データの入出力のみが含まれ、クロックや制御信号は含まない。

ライブラリLは5項組(OP, IN, OUT, OE, PI)の集合である。ここで、OPは実行可能な演算種類、INは入力ポートの集合、OUTは出力ポートの集合、OEは実行時間、PIはパイプライン処理間隔である。資源は機能部品である演算器、記憶部品であるメモリ、転送部品であるトライステートバスやマルチプレクサのいずれかである。機能部品はINの値に演算$op \in OP$を行

なった結果を OE 時間後に $o \in OUT$ に出力する。演算は PI で定められた時間間隔で実行可能である。記憶部品や転送部品は書き込みや読み出し、転送という特別の演算を OP として持つ。

(b) は制御回路や命令フィールド共有による制御排他性などのデータパス構成以外の理由から生じる資源の並列性に対する制約である。演算 (資源 R) と転送 (資源の 2 項組 $[R, R]$) の論理式で表わす。各資源はこの論理式を満たさない限り、並列に動作可能である。

提案手法では、演算資源をライブラリから選択して接続した段階では、演算資源とその接続構成を (a) を用いてモデル化して評価する。さらに、制御部や命令セットの設計が行われた段階では、それらの設計による演算資源構成の並列性への制約を (b) として付加して評価する。このようにアーキテクチャ設計で行われる各項目の設計を順に考慮に入れて評価することで、初期段階からのアーキテクチャ性能評価を実現する。

命令セットを用いた評価の場合、演算資源構成の設計後にさらに (1) 命令制御方式 (パイプラインなど)、(2) レジスタの位置と容量の決定、(3) 命令形式の決定 (語長等)、(4) 命令フィールドの割り当て、を設計する必要がある。本手法は、アーキテクチャ設計の項目の中で性能に最も影響の大きい演算資源構成を含むため、上記の情報がない場合でも良い評価が行える。また、(3)、(4) の設計の影響は (b) を用いて考慮可能なため、命令セットレベルでの評価の差は (1)、(2) の影響と考えられる。

2.2 ソフトウエアモデル

応用プログラムは C 言語のサブセットで記述する。応用プログラムはアルゴリズム記述であるが、内部 ROM、RAM に格納された初期データの配置や処理結果データの格納、定数テーブルの参照など、データ転送を評価したい部分については意識した記述を行なう必要がある。例えば、ROM、RAM は配列として記述しなければならない。

応用プログラムは基本ブロックの集合と基本ブロック間の制御関係でモデル化される。制御関係は C 言語の制御構造 (if then 文、for 文など) に対応する。基本ブロックは制御構造を含まない (処理の流れがデータに依存しない) プログラムの連続した一部分である。各基本ブロックは DFG (Data Flow Graph、データフローグラフ) で表す。DFG の頂点は (1) C の基本演算、(2)C 記述で関数定義された演算、(3)ROM/RAM の読み書き、(4) データ転送、(5) 定数読み込み、のいずれかである。

3 性能評価手法

提案する評価手法の概略処理フローを図 2に示す。評価は応用プログラムを用いた静的解析と動的解析を組み合わせて行なう。処理は以下の 5 つのステージに分けられる。静的解析では各基本ブロックの解析をハードウエアモデルから抽出した情報を用いて解析し、実行ステップ数を得る。動的解析ではプログラム実行により各基本ブロックが何回実行されたかを求める。結果集計部で得られたブロック実行の回数で重み付けして総和をとることで全体の処理ステップ数を得る。

なお、本評価手法では未設計の項目に関しては最適設計が行われるという仮定をおき、既設計部分の本質的な性能を評価する方針で行っている。

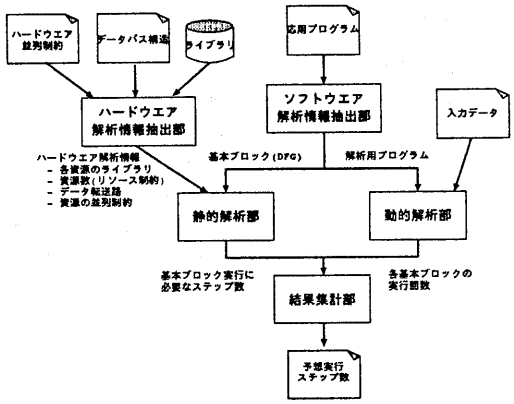

図 2: 性能評価の概略フロー

(1) **ハードウエア解析情報抽出部**

アーキテクチャのハードウエアモデルから静的解析に必要な以下の 4 情報を抽出する。

(A) 各資源のライブラリ
(B) 資源数 (リソース制約)
(C) 資源間のデータ転送路
(D) 資源の並列制約

(A) は資源のマッピング関数から、(D) は (b) から直接求められる。(B) は資源 R の実行可能演算を分類することで得ることができる。また、(a) には資源間の接続情報が含まれるため、任意の資源間のデータ転送路 (C) の抽出が可能である。データ転送路は実際の処理では必要なもののみ静的解析部で算出される。

(2) **ソフトウエア解析情報抽出部**

応用プログラムを構文解析によって分割し、各基本ブロックを DFG でモデル化する。拡張基本ブロック抽出部で生成された DFG にはデータ転送を表す DFG ノードは現れない。静的解析では解析する基本ブロック間の並列性やデータ転送が考慮されないので、ブロック規模が小さ過ぎる場合評価誤差が大きくなる。そのため、本手法では基本ブロック抽出後に、(a) ループ展開を行ない複数回基本ブロックを併合する、(b) トレーススケ

ジューリング[10]を利用したブロック併合を行なう、などの手法を用いて複数の基本ブロックを1つの分岐のないブロックに併合した拡張基本ブロック(以後、これを単に基本ブロックと呼ぶ)を取り扱う。

また、同時に動的解析用プログラムを生成する。解析用プログラムにはプログラム実行時に各基本ブロックの実行回数をカウントし、実行順序を出力するためのコードを追加する。

(3) 静的解析部

ハードウエア解析情報抽出部で抽出された情報を利用して各基本ブロックの資源制約スケジューリングを行い、実行ステップ数を求める。

解析では、転送資源におけるデータ衝突を回避し、かつ、転送にも処理ステップを割り当ててスケジューリングするために、バス等の転送部品も資源とみなし、かつ必要な転送をDFGの頂点として追加し、スケジューリングを行う。そのため、最初に与えられたDFGのスケジューリングを予備的に行なって各DFGノードに演算資源を割り当てる。割り当てられた演算資源をもとにして資源間のデータ転送路に対応する転送ノードをDFGに追加する。転送路はハードウエアモデル中の資源の接続情報から導出する。データ転送が追加されたDFGを再度転送も含めてスケジュールする。転送ノード追加後のスケジューリングは転送部品のリソース制約も考慮して行なう。スケジューリングは双方ともリストベースのスケジューリングを採用している。また、スケジューリング処理において演算や転送を処理ステップに割り当てる際に、ハードウエア並列制約を与える。スケジュールではハードウエア並列制約以外各資源の並列性に制限はないという仮定をおく。

解析では、各資源の入出力ポートには無限容量の暗黙レジスタを仮定する(ただし、暗黙レジスタを介さない転送も可能とする)。暗黙レジスタの仮定により、データ転送は演算実行から次の演算実行までのいずれかのステップにスケジューリングされる。これは、最適な位置に最適な容量のレジスタを配置してレジスタ最適化を行なった場合の本質的な処理能力を評価することに相当する。

(4) 動的解析部

基本ブロック抽出部で生成した解析プログラムに典型的な入力データを与えて実行する。解析用プログラムは各基本ブロックの実行数と実行追跡結果を出力する。

(5) 結果集計部

静的解析で得られた各基本ブロックの実行ステップ数を動的解析で得られたブロック実行の回数で重み付けして総和をとることで全体の処理ステップ数を得る。

上記手続きから明らかなように、提案した性能評価手法は(a)、(b)から抽出されたされた4情報のみをハードウエアモデルから抽出して用いる。以下では、これらの情報から導いた命令セットの実行ステップ数予測の精度評価について実験を行なう。

4 音声処理LSIへの適用実験

我々は提案手法にもとづく性能評価を組み込んだアーキテクチャ設計支援システムのプロトタイプを開発した。本節では、これをすでに開発済みの音声処理LSIに適用した評価結果と課題に対する考察について示す。

4.1 実験1:命令セットレベルとの予測精度

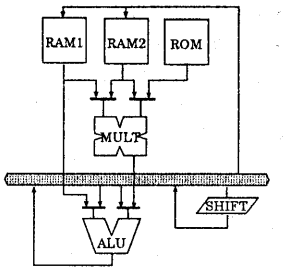

図3: 音声処理LSIのデータパス構成

```
1  int fft(n, ns, adr, s)
2  int     n;
3  int     ns;
4  int     adr;
5  int     s;
6  {
7          int     i, i0, i1, c;
8          int     x1, y1;
9          int     l1, sc, sce;
10         int     sin, cos;
11
12         sce = RO_SC256 + 256;
13         for (i = 0, i0 = adr, sc = RO_SC256; i < n; i++, sc += ns) {
14             if (sc > sce) sc = RO_SC256;
15             cos = rom[sc];
16             sin = rom[sc + 1];
17             i1 =  invert(adr, i0, s);
18             x1 = ram1[i1] * cos - ram2[i1] * sin;
19             y1 = ram2[i1] * cos + ram1[i1] * sin;
20             ram1[i1] = ram1[i0] - x1;
21             ram2[i1] = ram2[i0] - y1;
22             ram1[i0] = ram1[i0] + x1;
23             ram2[i0] = ram2[i0] + y1;
24             i0 = _fft_inc(adr, i1, n, s);
25         }
26 }
```

図4: 応用プログラム記述例(AP1の一部)

実験に用いた音声処理LSIのデータパス構成相当の構成を図3に示す。実験1では、このデータパス構成を用いて音声圧縮伸長処理の一部分を解析し、実際の命令レベルの実行ステップ数と比較した例を表1に示す。命令レベルのプログラムは実際に使用された人手設計のアセンブラプログラムである。使用した応用プログラムはAP1とAP2の2種類である。AP1はFFTを含む計算処理中心の部分で、AP2は音声データにより分岐を多く含む前処理部分である。AP2はデータに依存して処理の実行が変わるため2通りのデータで実験した。図4にAP1の一部(FFT部分)を示す。なお、この音声処理LSIはVLIW命令形式を採用しており、基本的に各演

表 1: 実験 1 の結果

応用プログラム	#LINE	#BB	EVAL	INS	EVAL/INS(%)
AP1	559	66	4,000	4,750	89.9
AP2(1)	636	129	5,693	6,057	93.2
AP2(2)	636	129	17,749	20,653	85.9

* EVAL: 評価値、 INS: 実際の命令ステップ数
#LINE:C プログラム行数、 #BB: 基本ブロック数

算器を独立に制御できるため、実験 1 ではハードウエア並列制約は設定していない。

本手法では、前述の通り、アーキテクチャ設計初期段階からの適用を対象とするために、命令セットレベルの評価に比べて (1) 命令制御方式、 (2) レジスタの位置と容量、を考慮せずに評価している。また、比較対象のアセンブラコードの最適性もソフトウエアによってばらつきがあると考えられる。それにも関わらず、実験 1 では評価結果として −14% ～ −7% 程度の見積もり精度が得られた。これはアーキテクチャ設計早期段階の予測結果としては十分実用的な値と考えることができる。

今回の我々の対象のように複雑な制御により高性能化を図らず、かつ、データ処理が支配的な DSP では命令制御の影響は小さいと思われる。また、レジスタに関しては解析結果からこの評価値を達成するレジスタ位置と容量を見積もることができるため、それをもとに性能向上を目指したレジスタ最適化が可能である。

アセンブラコードの最適性に関しては、実験に用いた LSI の場合でも AP1 と AP2 でコードの品質が異なる。また、解析ではコード最適化はあまり行なっていない。これらの理由により評価精度は当然左右されるが、目標値となりうる評価結果を得ることで十分実用的であると考える。

4.2 実験 2 : ハードウエア並列制約の効果

実験 2 では並列制約を付加した場合の影響について実験を行なった。実験対象とした LSI はもともと VLIW 命令形式を採用し、データパス演算資源の各部の制御が独立に制御できるように設計されているため、今回は命令語長を制限したと仮定した場合にどのフィールドを共有すれば並列性を損なわないかを実験した。実験結果を表 2に示す。

実験において与えた並列制約は以下の通りである。

(i) 並列制約なし

(ii) MULT–ALU 転送と BUS 転送の制約

(iii) ALU 演算と BUS 転送の制約

(iv) RAM1 転送と BUS 転送の制約

与えた制約はすべて BUS の転送制御との排他性であり、(ii) は MULT–ALU 転送、 (iii) は ALU 演算、 (iv) は RAM1 転送との間に制約を設けた。これらは例えば、命令における BUS 転送の制御とそれぞれの制御フィールドを命令語長の制限などの理由から共有した場合にモデルに追加される。

結果、 AP1 の場合には (ii)、 (iii) の制約の影響が大きく、 AP2 では (iv) の影響が大きい。また (ii) の制約は AP2 の実行には影響を与えないことが判明した (これはすなわち MULT–ALU をローカルバス化しなくても AP2 実行の性能は変わらないことを意味している)。従って、例えば AP2 のみを実行するための LSI であれば MUL T–ALU 転送や RAM1 転送と BUS 転送の制御フィールドを共有して命令語長を短くしても問題がないことがわかる。また、 AP1、 AP2 を引続き実行する LSI では (iii) より (iv) の方が処理ステップ数に与える影響を押えることができることがわかる。このように、ハードウエア並列制約はアーキテクチャの改良や命令セット形式の設をする際に用いることができ、その影響を評価することができる。この結果を用いて、設計者は命令セットの最適化やアーキテクチャ最適化を行える。

4.3 評価精度の向上に関する考察

実験 1 では早期段階の予測結果としては十分実用的な見積もり精度が得られた。しかし、ループ部分などで誤差の割合が大きくなっていることが判明した。解析の結果、以下に列挙する 2 項目が誤差要因と思われるため、これらの項目につきさらに精度を向上させるための考察を加える。

(a) 基本ブロックが適切な大きさでなく、基本ブロック間のパイプライニング等の並列性やデータ転送が考慮されていない。

(b) 静的解析における転送経路を考慮したスケジューリングが不十分で効率の悪い割り当てを行なう場合がある。

実験 1 では平均して C 記述で 5 ～ 9 行、 DFG のノード数にして 12 ～ 20 個の基本ブロックを扱っているが、現状では 3 節で述べた場合のみ基本ブロックを併合しているため、ブロック規模が適切ではない場合がある。解析時間とのトレードオフを考慮して適切なブロック規模を拡大する手法を現在検討中である。

また、転送を考慮したスケジューリングでは、転送スケジューリングするために DFG に転送処理を追加することで、最初の資源割り当てに利用した演算の処理ステップへの割り当てと追加後の割り当てがずれ最適な演

表 2: 実験 2 の結果

応用プログラム	ハードウエア並列制約			
	(i)	(ii)	(iii)	(iv)
AP1	4,000(100)	4,936(123)	5,184(129)	4,131(103)
AP2(1)	5,693(100)	5,693(100)	5,843(103)	6,343(111)
AP2(2)	17,749(100)	17,749(100)	18,671(105)	20,407(115)

* 括弧内は制約なしの場合とのステップ数の比 (%) である。

算器割り当てが変わる可能性があるためと思われる。上記プログラムの評価結果では、AP1 では約 10%、AP2 では 3.6% ～ 6.3% の処理時間がデータ転送に費やされている。これは実際のアセンブラ実行結果とも一致した傾向があり、転送を正しく考慮することは重要な課題の 1 つである。現在は、バックトラックの採用と、スケジューリングにおける演算器割り当てのコスト評価の改良を進めている。

5 まとめ

データパス部の演算資源構成にもとづくモデルとそれを用いて処理ステップ数を予測する性能評価手法を提案し、プロトタイプを開発した。また、音声処理 LSI に適用して評価した結果を報告した。演算資源構成はデータパスにどのような演算器をどのように接続して使用するかという情報であり、通常アーキテクチャ設計初期段階に決定される。また、データ転送を考慮して解析するための最低限の情報を持つ。従って、従来の命令セットにもとづく手法より早い設計段階で、データ転送まで考慮した応用プログラム向き命令セット最適化やハード / ソフト分割を行なうことが可能になる。これにより、アーキテクチャ設計の最適化を短期間に効率的に進めることができる。

命令セットを用いた手法に比べた場合、少ない情報量からの評価であるにも関わらず、−14% ～ -7% 程度の誤差で命令セットレベルの実行ステップ数を予測でき、手法の有効性を示すことができた。実験では同時に精度向上の考察を行ない、それにもとづいた手法の改良を進めている。本手法は、データパス部の演算資源構成を中心に評価する手法であるため、特にデータ処理が支配的である DSP などローコストで性能を追求した LSI には適した手法と思われる。

性能評価に関しては、本手法の能力と課題について明確にし、評価を用いた最適化の方向に研究を進める予定である。特に、今回適用した LSI は VLIW 方式のため、ハードウエア並列制約のモデル化能力を評価していない。並列制約の影響がわかるような命令セットを持つアーキテクチャで命令セットレベルの評価を比較し、能力を考察する予定である。また、我々はコストと消費電力に関して同レベルのモデルからの予測手法を考察中である。ハードウエアコストは演算資源構成からデータパス系に関しては直接導かれる。制御系は実行における状態遷移空間の大きさからある程度予測できると考えられる。消費電力は評価過程で得られる情報をどのように利用するかを現在検討中である。

参考文献

[1] 冨山宏之, 赤星博輝, 安浦博人, “アーキテクチャ評価用コンパイラの自動生成,” 信学技報, VLD93-92, pp.57-64 (Dec. 1992).

[2] R.K. Gupta and G. De Micheli, “Hardware-software cosynthesis for digital systems,” IEEE Design and Test of Computers, vol.10, no.3, pp.29 - 41 (Sep. 1993).

[3] M. Horowitz and K. Keutzer, “Hardware - software co-design,” Proc. of SASIMI '93, pp. 5 - 14 (Oct. 1993).

[4] I-J. Hunag, B. Holmer, and A. Despain, “ASIA: Automatic synthesis of instruction-set architectures,” Proc. of the SASIMI '93, pp. 15 - 12 (Oct. 1993).

[5] A. Karavade and E.A. Lee, “A hardware-software codesign methodology for DSP applications,” IEEE Design and Test of Computers, vol.10, no.3, pp.16 - 28 (Sep. 1993).

[6] G. Menez, M. Augin, F. Boéri and C. Carrière, “A partitioning algorithm for system-level synthesis,” Proc. of EDAC'92, pp. 482 - 487 (1992).

[7] J. Henkel and R. Erunst, “A path-based technique for estimating hardware runtime in HW/SW-cosynthesis,” Proc. of the 8th ISSS, pp. 116 - 121 (Sep. 1995).

[8] 中田武治, 佐藤淳, 塩見彰睦, 今井正治, 引地信之 “ASIP 向きハードウエア / ソフトウエア・コデザインシステム PEAS-I におけるハードウエア生成手法,” 信学技報, VLD93-93, pp.65-72, (Dec.1993).

[9] Yau-Tsun S. Li and S. Malik, “Performance analysis of embedded software using implicit path enumeration,” Proc. of 32nd DAC, pp.456 - 461 (Jun. 1995).

[10] J.A. Fisher, “Trace scheduling: A technique for global microcode compaction,” IEEE Trans. on Computer, vol.C-30, no.7, pp.478-490 (Jul. 1981).