計算機アーキテクチャ　121－6
設　計　自　動　化　82－6
（１９９６．　１２．　１２）

システム・オン・シリコン時代の特定用途向けシステム設計手法

井上 昭彦　　冨山 宏之　　エコー ファジャル
大隈 孝憲　　安浦 寛人
{inoino, tomiyama, eko, okuma, yasuura}@c.csce.kyushu-u.ac.jp

九州大学 大学院 システム情報科学研究科 情報工学専攻
〒816 福岡県春日市春日公園 6-1

本稿では，システムが１チップ上に集積される時代に適したシステム設計手法を提案する．機能と構造が可変なマイクロ・プロセッサをコアプロセッサとして集積することにより，用途に最も適したシステムを短期間で設計することが可能である．また，提案手法の要素技術の一つとして，コアプロセッサのデータ語長を可変とした場合の設計上位における見積り手法の提案を行う．実験の結果，プロセッサのデータ語長がシステムの面積と速度に及ぼす影響を確認でき，設計初期段階における見積りの重要性について確認した．

キーワード：ハードウェア/ソフトウェア協調設計，リターゲッタブル・コンパイラ，特定用途向けシステム

A Design Methodology for Application Specific System-on-Silicon

Akihiko INOUE　　Hiroyuki TOMIYAMA　　Eko Fajar
Takanori OKUMA　　Hiroto YASUURA

Department of Computer Science and Communication Engineering, Kyushu University

In this paper, we propose a design methodology suitable for application specific systems on silicon. We use *SoftCore Processor* as a core processor in a system whose architecture can be modified for the target applications. Our design methodology with SoftCore processor yields systems optimized in terms of cost, performance and power consumption in a short period.

We can change the data word width of a SoftCore processor for a target application. We also propose a method to estimate cost and performance of systems in the earlier design stage when the data word width of the core processor is changed. The estimation method is one of the essential technologies in proposed methodology. Experiments show the importance of estimation in earlier design stage.

Key Words : Hardware/Software codesign, Retargettable compiler, Application specific system

1 はじめに

集積化技術の進歩により，組み込み用途向けのシステム全体が1チップ上に集積される時代になりつつある．本稿では，1チップ上にシステムが集積される時代に適したシステム設計手法を提案する．システムは，プロセッサ，メモリ，および，ソフトウェアから構成されるため，それらを統合的に短期間で最適化するハードウェア/ソフトウェア協調設計が必要である．ハードウェア/ソフトウェア協調設計に関して多くの研究がなされている．しかし，文献[1, 2, 3]の手法はコアプロセッサの変更を仮定していない．また，文献[4]の手法において，コアプロセッサの変更は可能であるが，データ語長の変更は不可能である．我々は，データ語長をもプロセッサ設計時の変更パラメータとすることにより，システム設計の自由度を挙げ，最適に近いシステム構築を狙う．

本稿において，我々は(1)ソフトコア・プロセッサを用いた設計手法の提案，(2)データ語長可変プロセッサを搭載したシステムにおける設計上位における見積り手法の提案を行う．**ソフトコア・プロセッサ**とは，用途に応じて機能と構造の変更が可能なコアプロセッサである．また，ソフトコア・プロセッサは再設計が可能であるため，プロセッサの再設計時における設計変更の方向性を設計者に示す必要がある．(2)はそのための見積り方法である．

2章において我々が提案しているシステム設計手法を述べ，3章において提案手法に必要な要素技術および開発中のツールについて述べる．4章ではプロセッサのデータ語長がシステムへ及ぼす影響を解析し，データ語長をパラメータとしたシステムの性能等の見積り式を挙げる．5章においてデータ語長を可変としたシステムの性能等の見積りを行い，システムの面積と速度の関係を議論する．

2 提案するシステム設計手法

2.1 提案する設計手法の特徴

提案するシステム設計手法は以下のような特徴を持つ．

- 設計手法や設計環境を再利用することで，容易に特定用途向きのプロセッサとその基本ソフトウェアを構築することが可能である．
- 設計者は1からシステムを構築するのではない．設計者には可変項目(例えば，命令セットなど)を持つシステムが与えられる．設計者は可変項目を用途に応じて設定することにより，短期間にシステムの構築が可能となる．
- 機能と構造が可変なプロセッサ，ROM, RAM, ソフトウェアなど異なる要素を柔軟に組み合わせながらシステムを最適化する．
- 試作段階から大量生産に至る各工程を貫く統合的な設計開発を可能にし，ライフサイクルの短い製品の各設計開発段階に合わせた実現を可能とする．すなわち試作段階では，既存のプロセッサを用いて主にソフトウェアでシステムの機能を実現し，市場の反応を見ながら仕様を変更する．大量生産へ移行するに連れ，プロセッサを応用に合わせてチューニングし，性能やコストの最適化を図る．

2.2 対象とするシステム

本稿では，システムを「コアプロセッサ(マイクロプロセッサ)，ROM，RAMが1チップに集積されたものとその上で動作するソフトウェア」として定義する．ROMはプログラムメモリとして，RAMはデータメモリとしての機能を実現する．システム全体はコアプロセッサにより制御されるものとする．

システムは性能，価格，消費エネルギーにより評価される．性能はプログラムの平均，または，最悪実行時間により決定され，価格はシステム全体の面積により近似されるものとする．消費エネルギーとはシステム全体の消費電力の時間による積分値とする．

2.3 システム設計フロー

我々が提案するシステム設計フローは以下の通りである(図1参照)．

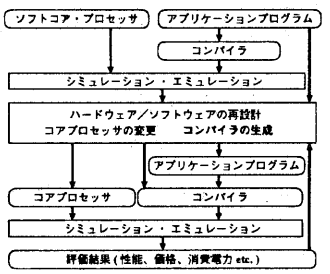

図 1: システム設計フロー

1. システム設計者には，ソフトコア・プロセッサとそれに対するコンパイラが与えられる．システム設計者は最初にアプリケーション・プログラムをC言語などの高級プログラミング言語により記述し，システムのプロトタイピングを行う．

2. 次に，設計者はシミュレータ等により，システムの性能評価を行う．この段階でシステム全体の仕様の決定，性能の見積りを行う．

3. システムの機能や目標性能が十分に固まった段階で，性能，コスト，消費電力などの制約に従い，ハードウェアとソフトウェアの境界の変更を行う．変更は設計上位における性能等の見積り結果を利用し，最適解に近づくように行う．プロセッサの変更を行う際，プログラムメモリおよびデータメモリを考慮に入れて変更する必要がある．プロセッサの変更により，システムのメモリの大きさや消費エネルギーが変化するためである．

4. 再設計されたプロセッサは再合成され，ネットリスト，レイアウトデータなどを得る．プロセッサを変更することにより，プロセッサ上で動作するアプリケーション・プログラムを再コンパイルする必要がある．コンパイラの設計にはリターゲッタブル・コンパイラを用いる (3.3節参照)．生成されたコンパイラを用いてアプリケーション・プログラムをコンパイルし，設計変更されたプロセッサ上で実行されるマシンコードを得る．再びシステムのシミュレーションまたはエミュレーションを行い，性能評価を行う．

5. 設計者は，目的に適合するプロセッサを得るまで，プロセッサの変更と性能評価を繰り返す．最終的に，カスタマイズされたコアプロセッサ，必要であれば周辺回路，プロセッサ上で動作するアプリケーションプログラムからなるシステムを得ることができる．

2.4 システムの可変項目

システム設計者はコアプロセッサのアーキテクチャを自由に変更することが可能である．しかし，設計の自由度が大き過ぎるとシステムを短期間に設計することは困難である．そのため，設計者に与えられる段階では，システムは一部のプロセッサ・パラメータのみが変更可能となっている．このように一部のパラメータが変更可能であるようなプロセッサがソフトコア・プロセッサである．ソフトコア・プロセッサは複数存在し，設計者は目的にあったソフトコア・プロセッサを選択することが可能である．ソフトコア・プロセッサの可変項目として選ばれる可能性のある，主なプロセッサ・パラメータを以下に挙げる．

データ語長: プロセッサの基本演算語長 (汎用レジスタの長さ) のことである．データ語長はシステムの性能，面積，消費エネルギーに大きな影響を及ぼすため，システムの最適化において重要な変更項目の一つである．

命令セット: アプリケーションに応じてコアプロセッサの命令セットを選択することが有効であることが報告されている [4]．速度向上を目的として，システムに専用回路などを付加して新たな命令を定義したり，必要のない命令を削除したりすることで，システムの性能等が大きく変化すると考えられる．

レジスタ数: レジスタファイルはチップ上で比較的大きな面積を占めている．また，データを可能な限りレジスタに格納しておくことはプログラムの速度向上，および，ROM 面積の削減につながる．更に，データメモリとプロセッサ間のデータ転送回数が削減され，消費電力の低減につながる．

その他，可変項目としてデータメモリのアドレス長，プログラムメモリのアドレス長，命令語長等がある．また，設計者はコアプロセッサの変更に応じてメモリのサイズやアドレス可能なビット数なども変更できる．

システム設計者はハードウェアだけでなくソフトウェアも変更することが可能である．ソフトウェアの変更とは，ROM に格納されているプログラムを変更することである．変更方法として以下の 2 点を挙げることができる．

- アプリケーション・プログラムのアルゴリズムを変更する．
- アルゴリズムは一定としてプログラムに適応するコンパイラ技術を変更する．

我々は，システム設計時の目的関数をコアプロセッサ，ROM，および，RAM から成るシステム全体の面積の最小化とする．システム全体の面積を最小化することにより，(i) チップのコスト削減，(ii) 動作周波数の向上，(iii) 消費電力の削減が期待できる．

3 要素技術の開発

提案する設計手法を実現するためには以下の要素技術を開発する必要がある．

1. カスタマイズに適したプロセッサ・アーキテクチャの開発
2. アプリケーション記述言語の開発
3. カスタマイズ可能な基本ソフトウェアの開発

4. 設計上位における性能，コスト，消費電力の見積り技術

5. カスタマイズしたプロセッサの完全な自動合成技術

6. 高速なシステム・シミュレーションおよびエミュレーション技術

現在，我々はこの中で主に，1, 2, 3, 4 の開発を中心に行っている．5については，現在の論理合成，レイアウト合成技術の進歩により市販の CAD ツールを用いることが可能である．また，6については，市販のシミュレーションツールやエミュレーションツールを用いることが可能である．さらに，文献 [5, 6] において提案されている高速シミュレーション手法を用いることも可能である．

3.1 ソフトコア・プロセッサ

設計データが公開され，アプリケーションに応じて機能や構造の変更が可能なコアプロセッサのことをソフトコア・プロセッサという．ソフトコア・プロセッサは主に組み込みシステムのコアプロセッサとして利用される．ソフトコア・プロセッサはレイアウトデータ，ネットリスト，HDL による論理レベルの記述，構造記述，および，動作記述により表される．設計変更は主に HDL 記述を変更することにより行われる．変更された HDL 記述を論理合成，レイアウト合成することによりカスタマイズされたプロセッサを得る．

現在，システムのコアプロセッサしては既設計のプロセッサが用いられる場合が多い．この場合，既設計のプロセッサのレイアウト情報をそのままシステムに組み込むこと，つまり，設計自身の再利用のみが許される．これに対しソフトコア・プロセッサでは，そのレイアウト情報だけでなく，基本ソフトウェアの設計情報およびその設計環境が全て公開される．そのため，システム設計者は用途に応じて比較的容易にシステムのハードウェア設計を行うことが可能となる．我々は文献 [7] においてソフトコア・プロセッサのプロトタイプの開発を行った．

3.2 アプリケーション記述言語

アプリケーション・プログラムは様々な有効ビットを持つ変数を有する．有効ビットとは変数がプログラムの実行において実際に必要とするビット数である．例えば，2 つの状態を保持する変数の有効ビットは 1 ビットである．プログラマは各変数に対する有効ビットは容易に知ることができる．我々が開発しているアプリケーション記述言語 **Valen-C** は，プログラマが変数に対して有効ビット数を指定できるように，C 言語を拡張した言語である．`intm`(m は正の整数) により宣言された変数は，プロセッサのデータ語長によらず，常に m ビットの精度がコンパイラにより保証される．

我々が提案する設計環境では，プロセッサの演算語長を変更し，目標となる性能，面積，消費エネルギーを満たすプロセッサを設計することが可能である．あるプログラム中で有効ビット数が変数により様々である場合，そのプログラムを動作させるプロセッサの演算語長を変化させ，システムの性能等を最適化することができる [8]．ただし，Valen-C を有効に利用するためには処理系が必須である．次節で Valen-C を受理する処理系について述べる．

3.3 リターゲッタブル・コンパイラ

ソフトコア・プロセッサの設計変更の結果，マシンコード・レベルにおける互換性が失われた場合，そのプロセッサに対する新たなコンパイラが必要となる．我々はカスタマイズされたプロセッサに対するコンパイラとして，リターゲッタブル・コンパイラを開発している．設計者はプロセッサ・パラメータ (レジスタ数，命令セットなど) を入力し，高級言語で記述されたリターゲッタブル・コンパイラをコンパイルすることでカスタマイズされたプロセッサのコンパイラを得ることができる．リターゲッタブル・コンパイラの研究はこれまでにもいくつか行われている ([9, 10]) が，これらのコンパイラは全て，プロセッサのデータ語長が 8×2^n(n は整数) であることを仮定している．データ語長はシステムの様々な要素に大きな影響を及ぼすため (4.1章参照)，対応可能なプロセッサのデータ語長を 8×2^n とすることは，プロセッサの最適化にとって得策とはいえない．

我々は新たなリターゲッタブル・コンパイラを開発している [11]．開発中のコンパイラは，プロセッサの命令セット，レジスタ数，データ語長などの変更に柔軟に対応することができる．更に，Valen-C および ANSI C により記述されたプログラムを共にコンパイルすることが可能である．任意のデータ語長を持つプロセッサに対して適応可能である点が，従来のリターゲッタブル・コンパイラと大きく違う点である．

3.4 設計上位における見積り技術

提案する設計手法では，プロセッサの変更とシステム全体の性能評価を交互に行う (最適化サイクルと呼ぶ) が，システムの再設計の後，性能評価を行い，その結果をフィードバックして設計に反映させるのに要する時間は短くない．したがって，システムの再設計毎に最適解に近づくような変更を行い，

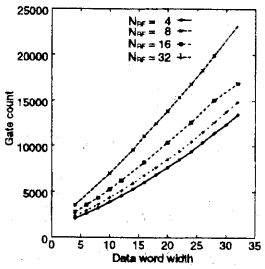

図 2: プロセッサのデータ語長とゲート数の関係

最適化サイクル数を減少させることが短期間設計につながる．我々は，システムの変更後の性能等を定量的に見積ることにより，最適化サイクル数の減少を図る．以降，本稿ではシステムの性能，面積，消費エネルギーの設計上位における見積り手法について説明する．

システムの性能は，プログラムの実行時間 (単位:秒) の逆数と定義する．面積値はコアプロセッサ，ROM，RAM の面積を合計したものであり，単位は mm^2である．システム全体の消費エネルギー (単位:J) は，コアプロセッサ，ROM，RAM それぞれについて，動作時間と動作時消費電力の積，および，待機時間と待機時消費電力の積の和を求め，それらを合計した値と定義する*．待機時間は動作時間により算出可能である．したがって，プロセッサ，ROM，RAM それぞれについて，面積，平均・最悪動作時間，動作時・待機時消費電力を，システムの可変項目をパラメータとした数式により表現すれば，性能等の見積もりを行うことが可能である．

4 データ語長を可変とした設計上位見積り手法

本章では，システムの可変項目のうちプロセッサのデータ語長のみを可変とした場合の性能，および，面積の見積り方法を述べる．ただし，メモリアクセスの最小単位 (ビット) はプロセッサのデータ語長と等しいものとする．また，プログラム中に現れる変数は全て静的にメモリへ割り当てられるものとする．

4.1 データ語長がシステムへ及ぼす影響

プロセッサのデータ語長とゲート数の関係を図 2 に示す．これは，Synopsys 社の論理合成系 [12] を用いて，我々が設計したソフトコア・プロセッサ Bung-DLX [7] を論理合成したものである．Bung-DLX は，(i) 逐次型プロセッサ，(ii) ハーバード・アーキテクチャ，(iii)DLX[13] の命令セットから浮動小数点系命令を除いたものという特徴を持つ．図 2に示す通り，プロセッサのゲート数はデータ語長の一次関数として近似され得る．データ語長が 32 ビットから 8 ビットへ減少すると，プロセッサのゲート数は約 60%削減される．

また，ROM, RAM 面積もデータ語長の変更に伴い変化する [14]．例えば，3 つの変数を格納する RAM を考える．プロセッサのデータ語長が 27 ビットであるとき，RAM の総ビット数は $27 \times 3 = 81$ ビットとなる．プロセッサのデータ語長が 9 ビットになると総ビット数は 72 ビットとなる．これは，未使用領域が減少したためである (図 3参照)．

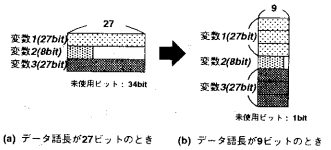

図 3: RAM の面積の変化

データ語長はシステムのあらゆる項目へ影響を及ぼすため，データ語長がシステムへ及ぼす影響を解析することは非常に重要といえる．

4.2 面積の見積り

4.2.1 プロセッサ面積

プロセッサの面積は予めデータ語長を変化させた場合の面積のデータをとる必要がある．そのデータを解析することによりデータ語長とプロセッサの面積の関係式を導出する．例えば，文献 [7] において設計されたソフトコア・プロセッサ Bung-DLX においてレジスタ数が 32 本としたときのゲート数 (G_{total}) は，データ語長 (DW) をパラメータとした以下の式により近似される．

$$G_{total} = -44.9 + 709 \times DW \qquad (1)$$

プロセッサの面積においてはレイアウト後の見積りが重要である．データ語長の変更は主にデータパス系に影響する．大きくレイアウトを変更しない限り，データパス系の規則性を利用することにより比較的容易に見積り式を構築できると思われる．

4.2.2 ROM 面積

ROM の面積は，ROM に格納する命令数に依存する．システムが必要とする ROM の面積は以下の式により近似される．

C_{ROM} × ROM に格納する命令数 × 命令語長

C_{ROM}はビット当たりの単価 (面積) を表しており，使用する ROM に依存する値である．データ語長の

*プロセッサはメモリアクセス時も動作しているものとみなす

削減により単精度の演算が多倍精度になり得る．倍精度演算が単精度の演算に展開された結果必要とするレジスタ数が増加し†，データ転送命令が増加する場合がある．n ビットのデータ語長を持つプロセッサ(以降，n ビットプロセッサと呼ぶ)において，ROM に格納する命令数 ($Ninst_{ROM}(n)$ とする) は以下の式により近似される．

$$Ninst_{ROM}(n) = \sum_{op \in OP} \sum_{k} N_{op}(k) \times (UF_{op}(k,n) + MA_{op}(k,n))$$

ただし，

OP	プログラム中の演算の種類の集合 ($\subseteq$ 命令セット)．
$N_{op}(k)$	デスティネーション・オペランドの有効ビットが k ビットの演算 op の数．
$UF_{op}(k,n)$	デスティネーション・オペランドの有効ビットが k ビットの演算 op を n ビットプロセッサ上で実行する際の命令数．
$MA_{op}(k,n)$	デスティネーション・オペランドの有効ビットが k ビットの演算 op を n ビットプロセッサ上で実行する際に生じるメモリアクセス命令の数．

である．$UF_{op}(k,n)$ は演算の種類毎にデータを用意する必要がある．$MA_{op}(k,n)$ は最悪の場合を見積もる．つまり，多倍精度の演算を単精度に展開した結果必要となるレジスタの増加分だけロード/ストア命令を挿入することにする．x オペランド命令を持つ多倍精度の演算が単精度へ展開されたときの $MA_{op}(k,n)$ は以下の式により近似される．

$$2 \times x \times \left(\left\lceil \frac{k}{n} \right\rceil - 1 \right)$$

ここで，係数 2 はロード命令とストア命令が必ず同数挿入されることを意味する．

4.2.3 RAM 面積

RAM の面積は RAM に格納する変数の有効ビット数に依存する．必要な RAM の面積は以下の式により近似される．

$C_{RAM} \times$ メモリユニット数 $\times$ メモリユニットの大きさ

ここで，メモリユニットとは，メモリのアクセスの最小単位となる記憶領域を指す．n ビットプロセッサにおけるメモリユニット数は，

$$\sum_{k} N_{var}(k) \times \left\lceil \frac{k}{\text{メモリユニットの大きさ}} \right\rceil$$

により近似される．ここで，N_{var} は有効ビットが k ビットの変数の数である．

4.3 動作時間の見積り

プロセッサの動作時間 (単位:秒) は，

$$\frac{\text{実行サイクル数}}{\text{プロセッサの動作周波数 } (Hz)}$$

により，ROM, RAM の動作時間 (単位:秒) は，

$$\text{アクセス時間 (秒)} \times \text{アクセス回数}$$

により近似される．ここで，アクセス時間とはメモリへ読出しの信号が出力されてから所望のデータが得られるまでの時間である．アクセス時間，および，プロセッサの動作周波数は使用する合成系やテクノロジに依存する．本節では，実行サイクル数，アクセス回数を見積る方法を述べる．見積りは基本ブロック単位に行う．プロセッサのデータ語長が n ビットのときのプログラム全体の実行サイクル数，ROM, RAM のアクセス回数は，

$$\sum_{b \in B} F(b,n) \times E(b) \qquad (2)$$

により与えられる．ここで，$F(b,n)$ は n ビットプロセッサ上で基本ブロック b を実行したときの実行サイクル数，ROM のアクセス数，RAM のアクセス数のいずれかに対応する．また，B はプログラム中の全ての基本ブロックの集合を表し，$E(b)$ は，

最悪値を求める場合: プログラムの実行時間がもっとも長くなるようにプログラムを一回実行したときに，基本ブロック b が実行される回数の期待値

平均値を求める場合: プログラムを一回実行したときに，基本ブロック b が実行される回数の期待値

である．以降の節において各基本ブロックの実行サイクル数，ROM, および，RAM のアクセス回数の見積り方法について述べる．

4.3.1 プロセッサの実行サイクル数

式 (2) の $F(b,n)$ を $Cycle(b,n)$ に置き換えることにより，プログラム全体の実行サイクル数を見積もることができる．n ビットプロセッサにおける基本ブロック b の実行サイクル数 $Cycle(b,n)$ は以下の式により近似される．

$$Cycle(b,n) = \sum_{op \in OP} \sum_{k} N_{op}(b,k) \times (UFC(k,n) + MAC(k,n))$$

ここで，

†レジスタの総容量ではなくレジスタ数を一定としているため

$N_{op}(b,k)$　基本ブロック b において，デスティネーション・オペランドの有効ビットが k ビットの演算 op の数．

$UFC_{op}(k,n)$　デスティネーション・オペランドの有効ビットが k ビットの演算 op を n ビットプロセッサ上で実行する際のサイクル数．

$MAC_{op}(k,n)$　デスティネーション・オペランドの有効ビットが k ビットの演算 op を n ビットプロセッサ上で実行する際に増加するメモリアクセスに要するサイクル数．

である．$UFC_{op}(k,n)$ は $UC_{op}(k,n)$ および各命令のサイクル数が分かれば求めることができる．$MAC_{op}(k,n)$ は以下の式により近似される．

$$MA_{op}(k,n) \times C_{mem} + 2 \times \left(\left\lceil \frac{\text{RAM のアドレス長}}{n} \right\rceil \times C_{ldc} \right)$$

ここで，C_{mem}, C_{ldc}はそれぞれメモリアクセスのサイクル数，即値をロードする命令のサイクル数であり，これらは使用するプロセッサに依存して決まる値である．第一項はメモリアクセスのみに必要なサイクル数であり，第二項はアドレスをレジスタに転送するために必要なサイクル数を表す．

4.3.2 ROM のアクセス回数

式 (2) の $F(b,n)$ を $AccROM(b,n)$ に置き換えることにより，プログラム全体の ROM のアクセス回数を求めることができる．ROM のアクセス回数は実行される命令数と考えることができる．n ビットプロセッサにおいて，基本ブロック b が実行されたときの ROM のアクセス回数 $AccROM(b,n)$ は以下の式により近似される．

$$AccROM(b,n) = \sum_{op \in OP} \sum_{k} N_{op}(b,k) \times (UF_{op}(k,n) + MA_{op}(k,n))$$

4.3.3 RAM の動作時間

式 (2) の $F(b,n)$ を $AccRAM(b,n)$ に置き換えることにより，プログラム全体の RAM のアクセス回数を求めることができる．n ビットプロセッサにおいて，基本ブロック b が実行されたときの RAM アクセス回数 $AccRAM(b,n)$ は，基本ブロック b に含まれているメモリアクセス命令の数に等しい．$AccRAM(b,n)$ は以下の式により近似される．

$$AccRAM(b,n) = \sum_{k} N_{mem}(b,k) \times \left\lceil \frac{k}{n} \right\rceil + \sum_{k} MA_{mem}(k,n)$$

ただし，mem はメモリアクセス命令を表している．

5 実験

プロセッサのデータ語長がシステムへ及ぼす影響を調べるために実験を行った．実験に用いたアプリケーションは自作したテレビコントローラである．ソース・プログラムは C 言語により記述した (約 250 行)．プログラム中の変数の分布を表 1に示す．

表 1: 変数の分布

有効ビット数	1	2	3	4	5	6
個数	2	2	2	1	1	9

実験は見積りのみを行い，見積り値と実測値の比較は行っていない．実験は次の手順により行った．(i) プログラムを *SPARCompiler 3.0* を用いてコンパイルし，アセンブラコードを得る．(ii) アセンブラコードを解析することにより，プロセッサ，ROM, RAM の面積を見積る．(iii)10 個の入力データを用意し，トレースをとる．(iv) トレース情報をもとに各基本ブロックの平均の実行回数を計算し，プロセッサ，ROM，RAM の動作サイクル数を見積る．実験では，全ての演算は 1 サイクルで終了するものとする．

表 2，3は実験結果である．表 2において，プロセッサのゲート数は 4.2節の式 (1) を使用して求めた．RAM, および，ROM のゲート数は $0.8\mu m$ の CMOS ゲートアレイ にマッピングしたものであり，文献 [14] のコストモデルに従っている．

結果はサイクル数の削減が必ずしも面積の増加につながらないことを示している．サイクル数はデータ語長が 9 ビットにおいて最小値を取り，それ以上データ語長を伸長しても変化しない．これは，ROM のアドレス長が 9 ビットとなったためである．面積はデータ語長が 6 ビットにおいて最小値を取る．周波数はデータ語長の圧縮に伴い向上するものと思われる．周波数を考慮に入れ，面積と速度の制約条件により，データ語長を決定する必要がある．また，データ語長が 3 ビットより小さくなると RAM のゲート数が増大する傾向にある．メモリ・アドレス長の増加によりアドレスをデコードするための回路が増加したことが原因と考えられる．1 ビットプロセッサにおける ROM のゲート数の激的な増加は，レジスタが不足するために生じるメモリ・アクセス，および，そのアドレス計算命令の増加に起因している．

表 2: 面積に関する見積り (単位:gate†)

データ語長	1	2	3	4	5	6	8	9	16	32
プロセッサ	**664**	1373	2081	2790	3499	4207	5625	6334	11295	22636
ROM	54456	29497	18152	18152	15883	13614	13614	**11345**	**11345**	**11345**
RAM	4303	2544	**1605**	1712	1819	1926	2140	2247	2996	4708
Total	59423	33414	21838	22654	21201	**19747**	21379	*19926*	25636	38689

† 2 入力 NAND ゲート換算

表 3: 動作サイクル数に関する見積り (単位:サイクル)

データ語長	1	2	3	4	5	6	8	9	16	32
プロセッサ	7493	2206	1199	1199	487	421	421	**394**	**394**	**394**
ROM	1660	923	645	645	371	330	330	**309**	**309**	**309**
RAM	833	428	277	277	116	91	91	**84**	**84**	**84**

6 おわりに

本稿では，(i) ソフトコア・プロセッサを用いた設計手法とその要素技術の開発，(ii) データ語長を可変とするプロセッサを搭載したシステムの設計上位における見積り方法について述べた．実験の結果，メモリを含めたシステム全体の設計において，面積，速度の関係を設計の初期段階で見積ることの必要性を確認した．

今後，実用化を念頭に入れツールの開発を行い，本手法を用いたシステム設計を実際に実行する予定である．その後，設計手法および開発すべきツールの再検討や現在のツールの改良を行う．また，消費電力の見積り方法を確立した後，本稿で述べた見積り値の相対的な関係の正確さを確認するために，実際の値と比較する必要がある．

謝辞

本プロジェクトに御協力頂く Hewlett-Packard 研究所の Barry Shackleford 氏，京都高度技術研究所の神原弘之氏，三菱電機 (株) の鈴木文雄氏，安田光宏氏，京都大学大学院工学研究科の清水友人氏，九州システム情報技術研究所の伊達博博士に感謝致します．また，日頃御討論頂く村上和彰助教授，岩井原瑞穂助教授をはじめとする九州大学安浦研究室の諸氏に感謝致します．

本研究は一部，情報処理振興事業協会 (IPA)「独創的情報技術育成事業」の支援による．

参考文献

[1] R. K. Gupta, C. N. Coelho Jr., and G. De Micheli. "Program Implementation Schemes for Hardware-Software Systems". *IEEE Computer*, Vol. 27, No. 1, pp. 48–55, 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] P. H. Chou, R. B. Ortega, and G. Borriello. "The Chinook Hardware/Software Co-Synthesis System ". In *Proc. of 8th Int'l Symp. System Synthesis*, pp. 22–27, 1995.

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

[5] 赤星 博輝, 安浦 寛人. "プロセッサの命令レベルシミュレーションモデルの自 動生成". *IEICE Trans.*, Vol. J78-A, No. 8, pp. 919–928, 1995 年 8 月.

[6] T. Uchida, H. Kiya, and A. Yamada. "Generating a Hierarchical Simulation Model on the Basis of Functional Model of Register Transfers". *IEEE Proc. of the ISCAS96*, Vol. IV, pp. 830–833, 1996.

[7] エコー ファジャル ヌルプラセティヨー, 井上 昭彦, 冨山 宏之, 安浦 寛人. "ハードウェア/ソフトウェア・コデザインのためのソフトコア・プロセッサの検討". 信学技報 VLD96–13, pp. 85–92, 1996 年 6 月.

[8] B. Shackleford, M.Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura. "Satsuki: An Integrated Processor Synthesis and Compiler Generation System". *IEICE Trans. Information and Systems*, Vol. E79–D, No. 10, pp. 1373–1381, October 1996.

[9] H.Tomiyama, H.Akaboshi, and H.Yasuura. "Compiler Generator for Hardware/Software Codesign". In *Proc. of Asia Pacific Conf. on Hardware Description Languages*, pp. 267–270, 1994.

[10] R. M. Stallman. *Using and Porting GNU CC for version 2.6*. Free Software Foundation, Inc., September 1994.

[11] 冨山 宏之, 井上 昭彦, 清水 友人, 神原 弘之, 安浦 寛人. "ハードウェア/ソフトウェア・コデザインのためのビット指定言語 Valen-C とその処理系の開発". 情処研報 DA 研究会, 1996 年 12 月.

[12] Synopsys Inc. Design Compiler ファミリー リファレンス・マニュアル version 3.1, 1994.

[13] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[14] B. Shackleford, M.Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura. "Memory-CPU Size Optimization for Embedded System Designs". Submitted to *34th Design Automation Conf.*