システムＬＳＩ設計技術　95－2
（2000. 2. 9）

データタイプを考慮したASIP消費電力見積り手法の提案

芝原 真一† 武内 良典† 今井 正治† 北嶋 暁† 木村 勉‡ 村岡 道明†

† 大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3
TEL: 06(6850)6626
FAX: 06(6850)6627

‡ 豊田工業高等専門学校 情報工学科
〒471-8525 愛知県豊田市栄生町 2-1
TEL: 0565(36)5869
FAX: 0565(36)5926

E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

あらまし Register Transfer Level (RTL) において，アプリケーションプログラムを実行したときのプロセッサの平均消費電力見積りを高速に行う手法を提案する．本手法では入力として，高級言語で記述されたアプリケーションプログラム，アプリケーションプログラムの入力データ群，プロセッサの HDL 記述，プロセッサ命令毎の各コンポーネントの動作情報およびプロセッサを動作させる周波数を用いる．そして，アプリケーションプログラム中にある変数のデータタイプに基づき，各コンポーネントの消費電力をモデル化した Power Macromodel を用いて消費電力の見積りを行う．RTL シミュレーションを伴わずに見積りを行うため，大規模な回路においても高速に消費電力の見積りを行うことができる．実験結果より，従来の RTL シミュレーションを用いた消費電力見積り手法と比べて高速に見積りが行え，かつ見積り値の精度もゲートレベルで見積りを行った場合と比べて遜色ないことを確認した．

キーワード ASIP, レジスタ転送レベル, 消費電力見積り, Power Macromodeling, Instruction Level Profiling

A Power Estimation Method for ASIP Considering Variables' Datatype in Application Program

Shin-ichi SHIBAHARA† Yoshinori TAKEUCHI† Masaharu IMAI†
Akira KITAJIMA† Tsutomu KIMURA‡ Michiaki MURAOKA†

† Dept. of Informatics and Mathematical Science
Graduate School of Engineering Science
Osaka University
1-3 Machikaneyama-cho, Toyonaka
Osaka, 560-8531 Japan
TEL: +81 6 6850 6626
FAX: +81 6 6850 6627

‡ Dept. of Infomation and Computer Engineering
Toyota College of Technology
2-1 Eisei-cho, Toyota
Aichi, 471-8525 Japan
TEL: +81 565 36 5869
FAX: +81 565 36 5926

E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

Abstract We propose a fast Register Transfer Level (RTL) power estimation technique for an ASIP on which an application program runs. Given the application program, groups of input data for the application program, HDL description of the ASIP, instruction level activity information for each component in the ASIP and clock frequency of the ASIP, power estimation is performed using power macromodels which contain input datatype dependent power consumption values for each component. Since this method is carried out without RTL simulation, it can be applied for large scale circuits. We have confirmed that reasonable estimation results can be obtaind using our method in considerably less time than RTL simulation.

key words ASIP, Register Transfer Level, Power Estimation, Power Macromodeling, Instruction Level Profiling

1. はじめに

ASIP (Application Specific Integrated Processor, 特定用途向き集積化プロセッサ) の設計では，応用分野に応じて適切な命令セットやアーキテクチャを決定する．例えば，携帯機器向け VLSI 設計において，設計候補の中から適切な命令セットやアーキテクチャを短時間で決定するためには，設計の初期段階で平均消費電力を高速に見積ることが重要である．

設計の初期段階における消費電力見積り手法は，多く提案されている [1][2]．提案手法の 1 つとして，設計資産のコンポーネントに対し消費電力の計測を行い，その計測値を Power Macromodel として登録したデータベースを利用して，コンポーネントの消費電力の見積りを行う手法がある．複数の既存のコンポーネントからなる回路に対し，Power Macromodel を用いた見積り手法は，設計の初期段階において精度の高い消費電力見積りができることで知られている．

従来，様々な Power Macromodel のモデリング (Power Macromodeling) 手法が提案されている．文献 [3] では，ルックアップテーブル (Look-up Table, LUT) モデルが提案されている．LUT のパラメータとして，入力ビットの平均信号確率 (average input signal probability) P_{in}，入力ビットの平均遷移濃度 (average input transition density) D_{in}，出力ビットの平均遷移濃度 (average output transition density) D_{out} を持つ．LUT から消費電力の見積り値を得るためには，各パラメータの値を RTL シミュレーションを行うことによって求める．LUT に登録するデータの個数は，LUT の各パラメータ P_{in}, D_{in}, D_{out} がとりうる値の個数をそれぞれ l, m, n とすると，$l \times m \times n$ になり，LUT を作ることに時間を要する．これに対し，文献 [4] では，P_{in} と D_{in} に加えて入力の空間的相関 (spatial correlation) と時間的相関 (temporal correlation) を考慮し，Power Macromodel を 1 つの関数によりモデル化した．これにより，D_{out} を求めるために必要であった RTL シミュレーションを行わずに消費電力の見積りが行え，Power Macromodel が持つデータの量が少なくなる．しかし，見積りの対象となる回路が複数のコンポーネントから構成されている場合には，各コンポーネントの入力系列を得るために RTL シミュレーションを行う．このように，現状の Power Macromodel を用いた消費電力見積りでは，Power Macromodel から消費電力の見積り値を得るために RTL シミュレーションを行っている．しかし，RTL シミュレーションは，回路が大規模化するに伴い，膨大な時間を要する．Power Macromodel を用いた消費電力見積りを行うために必要な時間は，Power Macromodel から消費電力の見積り値を得るために必要となる各パラメータ値を求める作業に要する時間が支配的になる．したがって，大規模な回路における消費電力を見積る場合，非常に時間を要することになる．

本稿では，RTL の ASIP に対し，アプリケーションプログラムを実行したときの平均消費電力の高速な見積り手法を提案する．本手法の特徴は，(i) RTL シミュレーションを伴わずに見積りを行うこと，(ii) Power Macromodel をアプリケーションプログラム中に存在する変数のデータタイプに基づいてモデル化したこと，である．上記 (i) により，大規模な回路においても高速に消費電力の見積りが行える．また，上記 (ii) について，このモデルを用いると，各コンポーネントの入力のデータタイプを調べることで，入力値の動的な特性を用いることなく，かつデータタイプを考慮しない場合と比較して精度良く見積りを行える．また，Power Macromodel に登録すべきデータの個数は，高々データタイプの種類の個数にしかならない．

提案手法での消費電力見積りの手順は以下の通りである．本手法では入力として，高級言語で記述されたアプリケーションプログラム，アプリケーションプログラムの入力データ群，プロセッサの HDL 記述，プロセッサ命令毎の各コンポーネントの動作情報 (動作・待機・停止) およびプロセッサを動作させる周波数を用いる．次に，アプリケーションプログラムおよびアプリケーションプログラムの入力データ群を用いて，命令レベルのシミュレータを実行させることにより，各 Basic Block の実行回数の統計をとる．さらに，HDL 記述より得られる各コンポーネントの接続情報とアプリケーションプログラムの各変数のデータタイプの情報をもとに，アプリケーションプログラムの各命令単位・各 Basic Block 単位での消費電力の見積り値を求め，それらの値を用いてアプリケーションプログラムを実行したときの消費電力の見積り値を算出する．各命令単位の消費電力の見積り値の算出には，まず各コンポーネントの入力のデータタイプを調べて，その結果を用いて Power Macromodel を参照し，各コンポーネントの消費電力の見積り値として用いる．

提案手法の見積り精度と見積り速度を評価する実験を行った．3 つのアプリケーションプログラムに対してそれぞれのアプリケーション対応のプロセッサを設計し，消費電力の見積りを行った結果，ゲートレベルで見積った消費電力と比べて約 10 % の精度で，従来の RTL シミュレーションを用いた消費電力見積り手法と比べて 1000 倍以上高速に消費電力の見積りを行うことができることを確認した．

以下，第 2 章に Power Macromodel の構成方法を示し，第 3 章で提案する消費電力見積り手法について述べ，第 4 章で実験を行った結果の考察を行い，第 5 章で本稿をまとめる．

2. データタイプに基づいた Power Macromodeling

2.1 コンポーネントの入力のデータタイプ

コンポーネントの入力をデータタイプにより区別する．コンポーネントの状態は，クロック以外の入力が変化しないか，クロックは動作しているが，コンポーネントはイネーブル信号により動作していない状態である待機状態，全ての入力が変化しない状態である停止状態，待機・停止状態以外の状態である動作状態で表すことができる．動作状態であるときのデータタイプは，C 言語での integer 型，short integer 型などで表し，停止または待機状態であるときのデータタイプは，それぞれ停止型，待機型とする．なお，以降では，このコンポーネントの状態の情報をコンポーネントの動作情報と呼ぶ．

2.2 Power Macromodel の構成

Power Macromodel はコンポーネント毎に用意され，以下の要素からなる．

1. 入力のデータタイプ $Type$ に対応するコンポーネント C_{type} 本体の消費電力 $P_{comp}(C_{type}, Type)$ (単位：W/MHz)
2. コンポーネント C_{type} のファンアウト $Fanout$ と入力のデータタイプ $Type$ に対応する接続部の消費電力 $P_{conn}(C_{type}, Fanout, Type)$ (単位：W/MHz)
3. コンポーネントの遅延時間 (単位：ns)

コンポーネントの構成を図 1 に示す．ここで，P_{conn} は，コンポーネントの出力の接続における消費電力である．なお，3. は，クロック周期がコンポーネントの遅延時間より小さい場合，Power Macromodel を使用することができないため，使用することができるか確認するために用いる．各コンポーネントの消費電力 P_{macro} は，C_{type}, $Type$, $Fanout$, プロセッサの動作周波数 F_{proc} (単位：MHz) を用いて，次式で表すことができる．

$$P_{macro}(C_{type}, Type, Fanout, F_{proc}) = (P_{comp}(C_{type}, Type) + P_{conn}(C_{type}, Fanout, Type)) \times F_{proc}$$

上式は，各コンポーネントの消費電力は，コンポーネント本体の消費電力と接続部の消費電力からなることを表している．

2.3 Power Macromodel の構築

既存のコンポーネントに対し，Power Macromodel を構築する方法について述べる．Power Macromodel の構築では，以下の条件を仮定する．

- コンポーネントの遅延時間は既知である

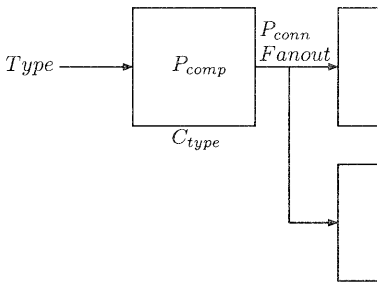

図 1　コンポーネントの構成

- ゲートレベルのネットリストが存在する
- コンポーネント内部の配線遅延・配線容量は既知
- 配線容量モデルが存在する

以上の仮定に基づき，Power Macromodel の要素であるコンポーネント C_{type} 本体の消費電力 $P_{comp}(C_{type}, Type)$ と接続部の消費電力 $P_{conn}(C_{type}, Fanout, Type)$ を計測する方法を次節で述べる．

2.3.1 コンポーネント本体の消費電力

コンポーネント C_{type} 本体の消費電力は，コンポーネント内の配線遅延・配線容量を考慮してゲートレベルシミュレーションした結果をゲートレベルの消費電力見積りツールに入力することで求める．コンポーネント本体の消費電力計測は，データタイプ $Type$ 全てについて行う．

図 2 にゲートレベルシミュレーションにおける各入力信号の与え方の概念図を示す．ゲートレベルシミュレーションにおいて各入力信号線に与える入力値には，変数のデータタイプを考慮した乱数を用いる．ここで，変数のデータタイプを考慮した乱数とは，変数のデータタイプで表現可能な数を範囲として発生させる乱数である．なお，イネーブル信号はそのコンポーネントの動作が可能になるような信号値に固定し，その他の信号はランダムに信号が変化するものとする．入力と制御信号は，コンポーネントの遅延時間以上の間隔で同期して変化し，クロック周期は入力の変化の時間間隔と同じとする．

乱数を用いた消費電力見積りは文献 [6] などで提案されているが，本稿では，変数のデータタイプを考慮した一様乱数を用いて，入力を 10000 回変化させたときの消費電力をコンポーネント本体の消費電力として登録する．

2.3.2 接続部の消費電力

接続部の消費電力 $P_{conn}(C_{type}, Fanout, Type)$ は，配線容量モデルから求められる配線容量と，変数のデータタイプを考慮した乱数の入力に対するコンポーネントの出力における各ビットの遷移濃度 (transition density: クロック周期の中で値が遷移する回数) $D_{out_i}(C_{type}, Type)$ (i :

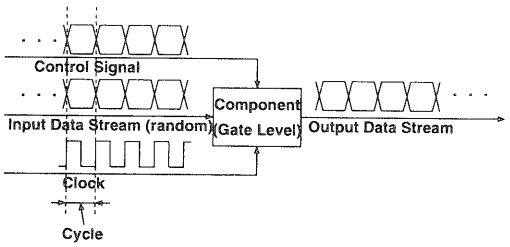

図 2　ゲートレベルシミュレーションにおける
各入力信号の与え方

output bit ID) より求める．後者は，コンポーネントの消費電力を計測する際，出力における各ビットの遷移濃度を計測することで得られる．

接続部分の消費電力は，次式により求められる．

$$P_{conn}(C_{type}, Fanout, Type) = \sum_{i}^{\#output\_bit} \frac{1}{2} \times C_{out_i}(C_{type}, Fanout) \times {V_{DD}}^2 \times D_{out_i}(C_{type}, Type) \times 10^6$$

ここで，$\#output\_bit$ は出力のビット数，C_{out_i} (i: output bit ID) は，出力ビットのファンアウトに対応した配線容量 (単位：F)，V_{DD} はプロセッサの電源電圧 (単位：V) をそれぞれ表す．本稿での配線容量モデルとしては，配線容量がコンポーネントのファンアウトに比例するモデルを用いた．なお，待機・停止時における接続部分の消費電力は 0 とする．

3. 提案する消費電力見積り手法

本章では，アプリケーションプログラムを実行したときのプロセッサの平均消費電力を見積るための手順を述べる．

図 3 に，本手法で行う消費電力見積りの全体図を示す．本手法では，まずアプリケーションプログラムおよびアプリケーションプログラムの入力データ群を用いて，命令レベルのシミュレータを実行させることにより，各 Basic Block の実行回数の統計をとる．次に，HDL 記述，プロセッサ命令毎の各コンポーネントの動作情報，プロセッサを動作させる周波数をもとに，各命令単位・各 Basic Block の消費電力を求める．最後に，アプリケーションプログラムを実行したときの消費電力の見積り値を算出する．各命令単位の消費電力の見積り値の算出には，まず各コンポーネントの入力のデータタイプを調べて，その結果を用いて Power Macromodel を参照し，各コンポーネントの消費電力の見積り値として用いる．

3.1 見積りの対象

本手法における見積りの対象は，以下に挙げる項目を

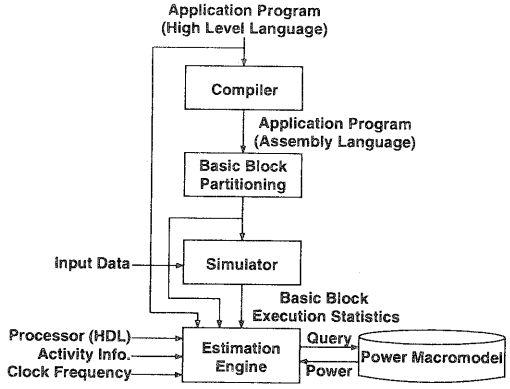

図 3　提案する見積り手法の概要

満たすことにする．

1. プロセッサは，演算器，レジスタなどのコンポーネントとそれらを制御する制御部からなる
2. アプリケーションプログラムは，変数にデータタイプが定義可能な高級言語で記述されている
3. プロセッサ中の各コンポーネントは，各プロセッサ命令に対し，動作・待機・停止の 3 状態のいずれかになる

ここで，3. について，プロセッサ命令は命令セットに定義されている個々の命令のことをいう．

3.2 仮　　定

本手法では，以下に挙げる条件を仮定する．

1. プロセッサ中の各コンポーネントに対して Power Macromodel が存在する
2. プロセッサアーキテクチャおよび命令セットに対応したコンパイラと命令レベルのシミュレータが存在する
3. 制御部の消費電力はプロセッサ全体の消費電力に比べて無視できるほど小さい

制御部の消費電力見積りは文献 [5] などで提案されているが，プロセッサ内ではデータパス中のコンポーネントの回路規模が大きく，制御部の占める割合は小さい．この妥当性は，4.3.1 で考察する．

3.3 入　　力

本手法の入力を以下に挙げる．

1. 高級言語で記述されたアプリケーションプログラム
2. アプリケーションプログラムの入力データ群
3. プロセッサの HDL 記述
4. プロセッサ命令毎の各コンポーネントの動作情報
5. プロセッサを動作させる周波数

本稿では，C 言語で記述されたアプリケーションプログラムを用意する．

3.4 見積り手順

前述の仮定・入力に基づいて，消費電力を見積る具体的な手順を述べる．

1. **アプリケーションプログラムのコンパイル**

   コンパイラにより，アプリケーションプログラムをアセンブリ言語で書かれた記述 (以降，アセンブリプログラムと呼ぶ) に変換する．

2. **アセンブリプログラムの Basic Block への分割**

   Basic Block 単位での消費電力の見積りを行うために，アセンブリプログラムを Basic Block に分割する

3. **アセンブリプログラム中に存在する各 Basic Block の実行回数の統計処理**

   アセンブリプログラムとアプリケーションプログラムの入力データ群を用いて，命令レベルのシミュレータを実行させることにより，アセンブリプログラム中に存在する各 Basic Block の実行回数の統計 $Num_{exe}(BB_i)$ をとる．ただし，アプリケーションプログラムの入力データによって各 Basic Block の実行回数が異なる場合，入力データを変えながらシミュレータを複数回実行することで，実行回数の統計をとる．以降では，この作業を Instruction Level Profiling と呼ぶ．

4. **各コンポーネントの演算器および接続情報の抽出**

   プロセッサの HDL 記述を解析することにより，各コンポーネントの演算器および接続の情報を抽出する．

5. **各コンポーネントのファンアウトの算出**

   各コンポーネント C_{name} に対し，接続情報を解析することでファンアウト $Fanout(C_{name})$ を算出する．コンポーネントの出力ビット毎にファンアウトが異なる場合は，最大のファンアウトをもつ出力ビットのファンアウトをコンポーネントのファンアウトとする．

6. **アプリケーションプログラム中にある変数のデータタイプの抽出と，各アセンブリ命令のオペランドのデータタイプの決定**

   アセンブリ命令とは，アセンブリプログラム中にある個々のプロセッサ命令を指す．変数のデータタイプの抽出は，アプリケーションプログラムの変数宣言より行う．各アセンブリ命令のオペランドのデータタイプの決定は，変数のデータタイプの抽出結果とアセンブリプログラムをもとに行う．

7. **アセンブリ命令毎の各コンポーネントの入力データタイプの決定**

   プロセッサ命令毎の各コンポーネントの動作情報とアセンブリ命令 A_{inst} のオペランドのデータタイプと各コンポーネント C_{name} の接続情報を用いて，プロセッサの入力よりオペランドのデータが流れたときのコンポーネントの入力のデータタイプ $Type(C_{type}(C_{name}), A_{inst})$ を決定する．具体例を以下に示す．図 4 はパイプラインプロセッサ中の EX ステージの構成例を示している．アセンブリ命令 ``ADD a3 a1 a2'' (a3 ← a2 + a1) で，a1, a2 が integer 型として実行される場合，Register-0, Register-1 の入力は integer 型となる．これにより，Adder, Shifter, Selector, Register-2 の入力は integer 型となる．

8. **各アセンブリ命令を実行したときの消費電力の算出**

   プロセッサ中に存在する各コンポーネント C_{name} の演算器の種類 $C_{type}(C_{name})$，各コンポーネントの各アセンブリ命令 A_{inst} に対する入力のデータタイプ $Type(C_{type}(C_{name}), A_{inst})$ およびファンアウト $Fanout(C_{name})$，プロセッサを動作させる周波数 F_{proc} より，アセンブリ命令の消費電力 $P(A_{inst})$ は，次式により求められる．

$$P(A_{inst}) = \sum_{\forall C_{name}} P_{macro}(C_{type}(C_{name}), Type(C_{type}(C_{name}), A_{inst}), Fanout(C_{name}), F_{proc})$$

   ここで，$P_{macro}(\cdot)$ は Power Macromodel を用いて得られる $C_{type}(C_{name})$ の消費電力である．

9. **各 Basic Block を実行したときの消費電力の算出**

   各 Basic Block BB_i (i : Basic Block ID) を実行したときの消費電力 $P(BB_i)$ は，次式により求められる．

$$P(BB_i) = \frac{\sum_{\forall A_{inst} \text{ in } BB_i} P(A_{inst})}{Num_{inst}(BB_i)}$$

   ここで，$Num_{inst}(BB_i)$ は BB_i の中に含まれるアセンブリ命令の数を表す．

10. **アプリケーションプログラムを実行したときの消費電力の算出**

    各 Basic Block の実行回数の統計 $Num_{exe}(BB_i)$ より，アプリケーションプログラムを実行したときの消費電力 $P_{program}$ は，次式により求められる．

$$P_{program} = \frac{\sum_{i}^{\#BB} P(BB_i) \times Num_{exe}(BB_i)}{\sum_{i}^{\#BB} Num_{exe}(BB_i)}$$

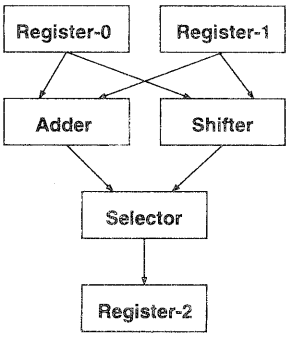

図 4　EX ステージの例

ここで，$\#BB$ はアプリケーションプログラムに含まれる Basic Block の数を表す．

4. 評価実験

提案手法の有効性を確かめるために，アプリケーションプログラムとそのアプリケーションプログラムを実行する目的で定められた命令セットおよびアーキテクチャからなるプロセッサを用いて，提案手法の見積り精度を評価するための実験を行った．

4.1 実験方法

本実験では，以下のアプリケーションプログラムを実行したときのプロセッサの消費電力見積りを行った．

- 200 次元ベクトルの内積 (naiseki)
- 10 × 10 の行列積 (array)
- 20 個のデータによるマージソート (merge)

命令セットの決定では，MIPS 社 R3000 の命令セットをもとに，アプリケーションプログラム毎に必要な命令を抽出した．プロセッサは，ASIP 設計用ワークベンチである PEAS-III [7] を用いて設計した．PEAS-III は，パイプライン段数や遅延スロット数などのアーキテクチャ情報，使用するコンポーネントの宣言，命令セットの定義，各命令の動作をクロック単位で記述したマイクロ動作記述を入力することで，入力に応じたプロセッサの HDL 記述を生成する．また，HDL 記述を生成する過程で，各プロセッサ命令に対する各コンポーネントの動作情報を得ることができる．本稿では，ハーバードアーキテクチャ，パイプライン段数 5 段 (IF, ID, EX, MEM, WB)，遅延スロット数 1 となる 32 bit RISC プロセッサを設計した．表 1 に設計したプロセッサの命令数，面積，遅延を示す．

コンパイラは GCC (GNU C Compiler) を用いた．シミュレータは MIPS R2000/R3000 シミュレータである SPIM を用いた [8]．

表 1　設計したプロセッサの情報

Program	命令数	面積 [Gates]	遅延 [ns]
naiseki	20	31,897	42.48
array	20	31,897	42.48
merge	21	19,628	12.64

表 2　実験環境

Machine	Sun Ultra Enterprise 450
CPU	UltraSPARC 296 MHz × 2
Memory	1.6 GB
Technology	ROHM 0.6μm ($V_{DD} = 5$ V)
Cell Library	EXD Library
Place & Route	Avant! Milkyway, Apollo
RTL Simulation	Synopsys VSS Simulator
Gate Level Simulation	Synopsys VSS Simulator
Gate Level Power Estimation	Synopsys DesignPower
Processor Design	PEAS-III
Compiler	GCC
Simulator	SPIM

見積り精度の評価は，ゲートレベルでの見積り値を真値とし，提案手法で見積った消費電力と比較することで行った．ゲートレベルでの消費電力の見積り値は，具体的には，アプリケーションプログラムを配線遅延・配線容量を考慮してゲートレベルで動作させた結果を入力として，ゲートレベルの消費電力見積りツールで見積りを行うことにより得た．配線遅延・配線容量は，配置配線ツールを用いて配置配線した結果より抽出した．また，プロセッサは 20 MHz で動作させることとした．

見積り速度の評価は，消費電力見積りに必要な情報を得るまでに経過した時間，すなわち，提案手法で用いる Instruction Level Profiling および従来手法で用いる RTL シミュレーションによって，アプリケーションプログラムを動作させたときに必要となる時間を比較することで行った．また，変数のデータタイプを考慮することの有効性の評価は，提案手法の誤差と変数のデータタイプを考慮しないで消費電力を見積ったときの誤差を比較することにより行った．変数のデータタイプを考慮しない消費電力見積りは，Power Macromodel としてコンポーネントの入力ビット数を n とすると，$-2^{n-1} \sim 2^{n-1} - 1$ の範囲で発生する一様乱数を用いたときにおけるコンポーネント本体および接続部分の消費電力を 2.3 に基づいて登録したものを用いて行った．

この実験で用いた環境を表 2 に示す．

4.2 実験結果

表 3 にゲートレベルでアプリケーションプログラムを 20 MHz で実行させたときの消費電力，提案手法で見積っ

表 3　消費電力見積り結果

Program	Gate Level [mW]	Error (Proposed) [%]	Error (考慮なし) [%]
naiseki	617.1696	−6.41	151.98
array	582.7245	−4.92	166.88
merge	515.4261	−5.42	39.71

表 4　消費電力見積りに必要な情報を得るまでに要した時間

Program	Inst. Level Profiling [s]	RTL [s]	Gate Level [s]
naiseki	0.07	511.10	1,681.70
array	0.09	1,575.10	12,827.40
merge	0.05	89.80	747.20

た消費電力と真値との誤差，変数のデータタイプを考慮しないで見積ったときの消費電力と真値との誤差を示す．表 3 より，提案手法でプロセッサ全体の消費電力を見積ったときの誤差は 10 % 以内に収まることが分かった．また，変数のデータタイプを考慮することにより，消費電力の見積り誤差が小さくなることが分かった．

表 4 に Instruction Level Profiling, RTL シミュレーション，ゲートレベルシミュレーションで，アプリケーションプログラムを動作させたときに要した時間を示す．表 4 より，提案手法で必要な情報を得るまでに要した時間は，RTL シミュレーションを用いる手法に比べて 1000 倍以上短くなることが分かった．

以上の結果より，本手法を用いることで，従来の RTL シミュレーションを用いた消費電力見積り手法と比べて高速に行え，かつ見積り値の精度もゲートレベルで見積りを行った場合と比べて遜色ないことが分かった．

4.3 考　　察

本節では，3.2 で述べた仮定の妥当性および見積り誤差が生じた原因，見積り誤差を小さくする方法を考察する．表 5 に，200 次元ベクトルの内積を行うプログラムを 20 MHz で実行させたときの各コンポーネントの消費電力，提案手法で見積った各コンポーネントの消費電力と真値との誤差を示す．ここで，表 5 中の Wire は各コンポーネント間の接続部分における消費電力の総和，Ctrl は制御部の消費電力を表す．

4.3.1 仮定の妥当性

表 5 より，制御部の消費電力はプロセッサ全体の消費電力の約 2 % であることが分かる．これにより，制御部の消費電力はプロセッサ全体の消費電力に比べて無視できるほど小さいとみなすことができる．したがって，提案手法における仮定は満たされている．

4.3.2 見積り誤差が生じた原因

表 5 より，PC (Program Counter) の値が流れる演算器 (ADD0, SEL6) で約 300 %，EXT0 で約 100 %，ALU0 で約 55 %の誤差が生じていることが分かる．図 5 に大きく誤差が生じている演算器の入力データの流れを示す．

PC の値が流れる演算器で大きな誤差を生み出す原因について考察する．PC は，クロック毎の各ビットの変化はほとんど下位ビットでしか行われない．したがって，PC の出力値が流れる演算器は，入力に流れるデータタイプを考慮しても，実際に生じる消費電力は少ない．これが，消費電力を大きく見積った原因と考えられる．この問題を解決するためには，PC の変化まで考慮すればよい．

EXT0 で大きな誤差を生み出す原因について考察する．EXT0 (符号拡張器) のトグルは命令のビットフィールドに依存し，プログラム中のデータタイプには依存しない．しかし，プログラム中のデータタイプに依存するように見積ったために，消費電力を大きく見積ったと考えられる．この問題を解決するためには，命令のビットフィールドまで考慮すればよい．

ALU0 で大きな誤差を生み出す原因について考察する．ALU0 は入力が 0 の場合でも，何らかの演算が行われる．また，プロセッサ命令 NOP が行われる場合，EX ステージで ALU0 に 0 が入る．しかし，NOP で ALU0 が動作しないと予測したために，消費電力を小さく見積った原因となったと考えられる．この問題を解決するためには，各プロセッサ命令による演算器の動作状況をより詳細に分類すればよい．

その他の演算器で誤差を生み出す原因は，ストア命令やロード命令のアドレス計算に必要となるスタックポインタの値のデータタイプやプログラムに依存しないプロセッサ内部の値を考慮しなかったことが考えられる．この問題を解決するためには，プロセッサ内部の値のデータタイプまで考慮すればよい．

以上の誤差が生じた原因となった要素を考慮することで，精度がさらに向上することが期待できる．しかし，見積りに考慮する要素が増えるにつれて，見積りにかける時間が長くなることが予想される．これにより，見積りに要する時間と精度のトレードオフが生じる．

5. むすび

本稿では，RTL の ASIP に対し，アプリケーションプログラムを実行したときの平均消費電力の高速な見積り手法を提案した．本手法は，RTL シミュレーションを伴わずに見積りを行うため，大規模な回路においても高速に消費電力の見積りを行うことができる．

実験結果より，従来の RTL シミュレーションを用いた消費電力見積り手法と比べて高速に，かつ精度を落とすこ

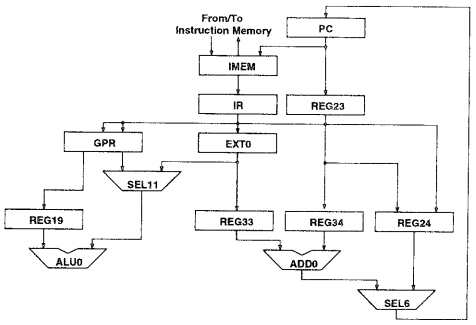

図 5　誤差の生じる原因となる演算器間のデータの流れ

となく消費電力を見積ることができることを確認した．

今後の課題としては，4.3.2 で述べた問題まで考慮したときの見積り精度の考察，パイプラインストールまで考慮した消費電力見積り手法の考案，乱数の種類 (一様乱数，正規乱数，ホワイトノイズ) などをパラメータに持つ Power Macromodel の最適な構築方法の考案が挙げられる．

参考文献

[1] P. Landman, "High-Level Power Estimation," Int. Symp. Low Power Electronics and Design, pp. 29–35, Aug. 1996.

[2] M. Pedram, "Power Simulation and Estimation in VLSI Circuits," The VLSI Handbook, The CRC Press and the IEEE Press, 1999.

[3] S. Gupta, F. N. Najm, "Power Macromodeling for High Level Power Estimation", Proc. 34th Design Automation Conf., pp. 365–370, Jun. 1997.

[4] G. Bernacchia, M. C. Papaefthymiou, "Analytical Macromodeling for High-Level Power Estimation," Proc. ICCAD, pp. 280–283, Nov. 1999.

[5] P. E. Landman, J. M. Rabaey, "Activity-Sensitive Architectual Power Analysis," IEEE Trans. on Computer-Aided Design, pp. 571–587, Jun. 1996.

[6] R. Burch, F. Najm, P. Yang, T. Trick, "A Monte Carlo Approach for Power Estimation," IEEE Trans. on VLSI Systems, Vol. 1, No. 1, pp. 63–71, 1993.

[7] M. Itoh, Y. Takeuchi, M. Imai and A. Shiomi, "Synthesizable HDL Generation for Pipelined Processors from Micro-Operation Description," IEICE Trans. on Fundamentals., Mar. 2000. (to appear)

[8] J. R. Larus, "SPIM S20: A MIPS R2000 Simulator," University of Wisconsin–Madison, 1990.

表 5　消費電力見積り結果 (詳細)
Program: 200 次元ベクトルの内積

C_{name}	Gate Level [mW]	Error [%] (Proposed)
PC	11.0430	−0.39
IMEM	0.0000	0.00
IR	10.2303	1.65
GPR	281.2241	2.83
EXT0	0.0004	100.12
ALU0	24.3573	−54.65
DMEM	2.4320	0.78
SFT0	2.3122	17.40
MUL0	42.7651	−18.58
HI	8.4680	4.38
LO	8.6100	2.66
ADD0	1.3496	293.37
SEL6	0.5894	301.18
SEL7	0.1696	−17.39
SEL8	2.5050	−51.77
SEL9	1.1320	−51.61
SEL10	0.6661	51.60
SEL11	1.0930	−52.02
REG19	8.9500	6.19
REG20	4.4361	10.65
REG21	2.9560	10.68
REG22	2.9554	10.70
REG23	9.1980	25.43
REG24	9.6830	19.97
REG25	15.1702	−37.34
REG26	9.0630	4.29
REG27	9.4522	0.55
REG28	9.3908	1.21
REG29	1.2786	−23.99
REG30	1.1402	−14.76
REG31	22.6794	−7.64
REG32	18.7241	11.87
REG33	8.6180	12.87
REG34	9.2432	24.82
REG35	9.2432	3.12
REG36	10.2983	−7.45
REG37	9.6631	−1.36
CSW	8.4770	3.63
Wire	2.5054	5.87
Ctrl	12.7725	—
Total	617.1696	−6.41