社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＳＬＤＭ－120（５）
2005／5／19

[招待論文]　高位設計における低消費電力技術

宇佐美 公良

芝浦工業大学 工学部 情報工学科　〒337-8570 埼玉県さいたま市見沼区深作 307

E-mail:　usami@sic.shibaura-it.ac.jp

あらまし　システム LSI や SoC の設計フローの中で、アルゴリズム設計やアーキテクチャ設計は、高位設計と呼ばれる。本論文では、マイクロアーキテクチャ・レベルでの低消費電力技術に焦点を当て、パイプラインやキャッシュの電力低減技術、および電力見積り・解析技術を紹介する。また、微細化とともに増大する動作時リーク電力の問題を取り上げ、動作時リーク電力を低減するマイクロアーキテクチャ・レベルの技術についても触れる。

キーワード　低消費電力，高位設計，マイクロアーキテクチャ，リーク電力

Low Power Technology in High-level Design

Kimiyoshi USAMI

Dept. of Information Science and Engineering, Shibaura Institute of Technology

307 Fukasaku, Minuma-ku, Saitama-City, Saitama 337-8570, Japan

E-mail:　usami@sic.shibaura-it.ac.jp

Abstract　In the design flow for system LSI's or SoC's, algorithm and architecture designs are referred to the high-level design. This paper focuses on low-power design techniques at the micro-architecture level and describes power optimization techniques for pipelines and caches. Power estimation and modeling techniques are described as well. This paper also presents approaches to reduce active leakage power at the micro-architecture level.

Keyword　Low power, ,high-level design,　micro-architecture,　leakage power

1. はじめに

LSI の高機能化・高性能化を追求する過程で、消費電力の増大をいかに解決するかが、近年の LSI 設計の大きな課題となっている。CMOS LSI の消費電力の主成分は、ダイナミック電力、貫通電力、リーク電力である[1]。貫通電力は、設計を注意深く行うことにより小さくできることを考慮すると、消費電力は次の式で表される。

$$P = CV_{DD}^{2} f\alpha + I_{leak} V_{DD} \qquad (1)$$

ここで、右辺第１項はダイナミック電力による成分であり、第２項はリーク電力による成分である。また、C は負荷容量、V_{DD} は電源電圧、f は動作周波数、αはスイッチング確率、I_{leak}はリーク電流である。LSI の低消費電力技術は、これら物理量の大きさをできるだけ小さくする技術といえるが、これらの物理パラメータには性能に影響を与えるものも含まれているため、性能とのトレードオフを考慮しつつ、いかに消費電力低減効果を上げられるかが鍵となる。

システム LSI や SoC の設計フローの中で、RTL 設計以降の工程では、ゲーテッド・クロックや低消費電力論理合成といった代表的な技術が開発され、実用レベルで幅広く使われている。これに対し、RTL 設計より上位の設計工程、すなわち高位設計と呼ばれるアルゴリズム設計やアーキテクチャ設計の工程では、応用用途の依存性が大きいため、どの用途でも万能という低消費電力技術は存在しにくい。むしろ、応用用途の性質をうまく利用した低消費電力技術が開発され、実用化されている。

高位設計における低消費電力技術を俯瞰すると、大きく次の４つの技術分野が存在し、発展を続けていることが分かる。

1. マイクロアーキテクチャ・レベルの低電力技術
   CPU をはじめとするプロセッサにおいて、キャッシュやパイプラインの構成や制御方式を工夫することにより低消費電力化を図る技術である。
2. マルチメディア向け低電力アルゴリズム／アーキテクチャ技術
   画像処理や音楽処理等の応用では、人間の視覚や聴覚の特性をうまく利用して計算量を減らし、消費電力低減化を図るアルゴリズムやアーキテクチャ技術が研究開発されている[2]。

3. 高位合成 CAD 技術
アルゴリズム設計やアーキテクチャ設計の段階で、消費電力と性能のトレードオフ解析を行い、高位合成を行なう CAD が開発され、実用化されている。ChipVision 社の ORINOCO[3]等がその例である。
4. DVS（Dynamic Voltage Scaling）技術
処理負荷に応じて電源電圧を動的に切り替える技術である。処理負荷が軽い時には電源電圧を下げて動作させるので、ダイナミック電力が低減する。電圧の最適スケジューリングを行なう OS 技術も必要である。Transmeta 社の Crusoe に搭載されて以来、注目が高まった。

以上述べた 4 つの技術分野の内、本論文では、マイクロアーキテクチャ・レベルの低電力技術に焦点を当てて論ずる。

第 2 章では、マイクロアーキテクチャ・レベルのダイナミック電力低減技術について、代表的な技術や手法を取り上げながら述べる。第 3 章では、マイクロアーキテクチャ・レベルでの電力見積り技術について、消費電力のモデル化や電力見積りツールを取り上げつつ述べる。第 4 章では、最近の話題として、動作時リーク電力を低減するマイクロアーキテクチャ・レベルの技術を取り上げ、紹介する。最後に、本論文のまとめと今後の動向について述べる。

2. マイクロアーキテクチャ・レベルの低消費電力技術

2.1. CPU の消費電力の内訳

低電力化技術の議論に入る前に、まず、CPU 内部で消費電力の大きい部分がどこなのかを知っておくことは重要である。図 1 に、StrongARM SA-110 [4]、ARM920T [5]、PentiumPro [6]、PowerPC [7]の消費電力の内訳を示す。

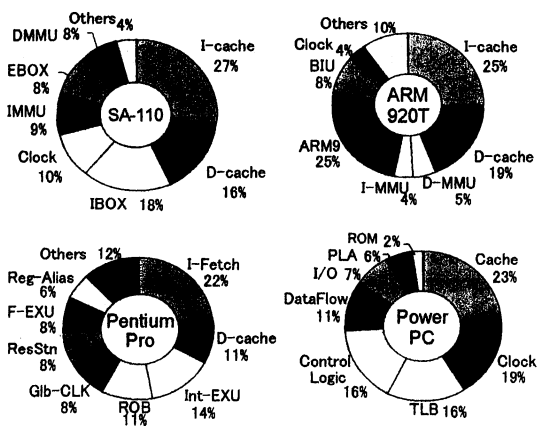


図 1. CPU の消費電力内訳

いずれも、キャッシュの電力の占める割合が最も大きく、チップ全体の消費電力の 23～44%を占めることがわかる。このことから、キャッシュをターゲットにしたマイクロアーキテクチャ・レベルの低電力技術が、数多く研究され報告されている。また、CPU の技術動向を見ると、1 次、2 次キャッシュに加え、メガバイト規模の 3 次キャッシュまで CPU チップ上に搭載したものも出現しており（例:Itanium2）、キャッシュの低電力技術は、今後ますます重要性を増すと考えられる。

2.2. ダイナミック電力低減技術－キャッシュ

キャッシュを低電力化するマイクロアーキテクチャ・レベルの技術には、大別して 2 つの方式がある。

1 つ目は、キャッシュ全面を動作させない方式である。キャッシュを部分的に動作させることでスイッチング時の負荷容量 C を小さくし、ダイナミック電力を低減する。バンク分割、ラインバッファリング、ウェイの選択活性化、ウェイ予測等の技術がこれに当たる。

2 つ目は、小規模のキャッシュを設けてメイン・キャッシュの動作回数を減らす方式である。ループ・キャッシュ、フィルタ・キャッシュ[8]といった技術がこれに該当する。

2.2.1. バンク分割

キャッシュのデータメモリ・アレイを n 個のバンクに分割し、アクセスする際に n 個のうち 1 個のバンクだけを選択活性化する方式である[9]。バンクの選択は、アドレス中のブロック内オフセットを使って行う。データメモリの消費電力が低減するとともに、性能低下がほとんど無い点が利点であるが、タグメモリの電力が減らないという欠点がある。

2.2.2. ラインバッファリング

メモリアクセスには空間的・時間的局所性があることが知られているが、キャッシュにおいても、1 ライン中の語が続けてアクセスされる場合が多い。この性質に着目し、一番最近アクセスしたラインをバッファリングしておく技術がラインバッファリングである[10]。構成を図 2 に示す。ラインバッファがヒットしたときには、タグメモリ、データメモリは動作させず、ラインバッファから読み出す。ラインバッファがミスしたときのみ、タグメモリ、データメモリを動作させる。バンク分割に比べ、タグメモリの消費電力低減も行える点が利点である。一方、この方式では、ラインバッファのミスが判明してからタグメモリ、データメモリのアクセス動作を開始するため、キャッシュアクセスのレイテンシが大きくなる欠点がある。

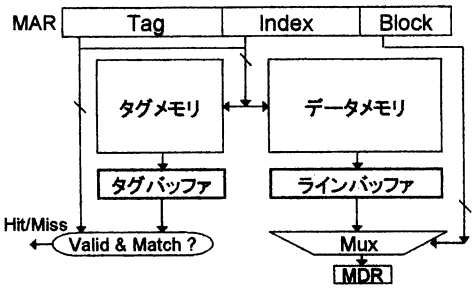


図 2. ラインバッファリングの構成

最近アクセスしたm本のラインをバッファリングしておく方式（マルチ・ラインバッファ）も提案されている[11]。1本のラインバッファで消費電力を 60%程度に低減できるが、8本まで増やすと消費電力を 40%程度まで低減できることが報告されている[11]。

2.2.3. ウェイの選択活性化

n-ウェイのセットアソシアティブ・キャッシュでは、通常n個のウェイが同時に動作する。ところが、必要なデータは1つのウェイだけにしか存在しないので、消費電力的に無駄が多い。必要なデータの存在するウェイのみ選択活性化させることで、キャッシュの消費電力を減らす方式が報告されている[12]。この方式では、キャッシュのアクセス動作を2フェーズに分割する。フェーズ1では、タグの比較を行いデータメモリはアクセスしない。フェーズ2では、ヒットしたウェイに対してのみデータメモリをアクセスする。日立 SH3 に適用した例が報告されており[12]、消費電力が60%低減した結果が示されている。一方、フェーズ 1とフェーズ2のシーケンシャルな動作が必要なため、レイテンシが大きくなる欠点も指摘されている。このため、上記適用例では低動作周波数モードのときのみ使用されている。

2.2.4. ウェイ予測

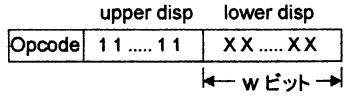
前節で述べた方式のレイテンシ増大の問題点を解決すべく提案された方式が、ウェイ予測である[13]。この方式では、まず、アクセスするウェイを予測し、その予測に従ってキャッシュのアクセス動作を行う。予測が正しければ、正しいデータが読み出され、キャッシュアクセスは1サイクルで完了する。予測が外れた場合には、次のサイクルで、残りのウェイを並列にアクセスし読み出し動作を行う。このため、予測ミスの場合、アクセス開始から2サイクルで正しいデータが読み出されることになる。

ウェイの予測には、MRU（Most Recently Accessed）アルゴリズムを使う方法が提案されており[13]、16KBの4ウェイ・セットアソシアティブキャッシュ（ラインサイズ 16B）で、ウェイの予測ヒット率 90%が得られることが報告されている。その結果、平均アクセスタイムは、従来のキャッシュに比べてほとんど増大しない。また、ウェイ予測を行う方式では、2.2.3 節で述べた方式とほぼ同等の消費エネルギー削減効果が得られることも報告されている。

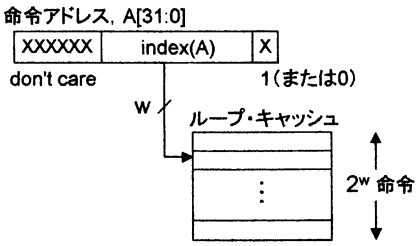
以上述べた技術は、キャッシュ全面を動作させないことで低消費電力化を図るものであった。次に、小規模のキャッシュを設けてメイン・キャッシュの動作回数を減らし、消費電力を低減する技術について述べる。代表的な例として、ループ・キャッシュを取り上げて説明する。

2.2.5. ループ・キャッシュ

組込み制御のプログラムでは、実行時間の大半を小さいループの実行に費やしている。小さいループの実行では、消費電力の大きい命令キャッシュを毎回アクセスするのではなく、命令キャッシュの前段に設けた小規模の命令バッファ（ループ・キャッシュ）から命令をフェッチしたほうが、消費電力を低減できる。モトローラ社の M・CORE にループ・キャッシュを適用した例が報告されている[14]。この例では、まず、short backward branch 命令（sbb 命令）と呼ばれる特殊な分岐命令が、命令セットに追加されている。sbb 命令のフォーマットと、ループ・キャッシュのアクセス方法を図3に示す。



(a). sbb命令のフォーマット



(b). ループ・キャッシュのアクセス方法

図 3. (a) sbb 命令と(b)ループ・キャッシュ

命令ストリーム中でsbb命令が検出され、分岐が成立すると、ループの2回目の繰り返しが始まったとみなし、そこから始まる命令を順にループ・キャッシュに格納していく。3回目の繰り返し以降は、ループ・キャッシュから命令をフェッチする。

sbb命令中で、後方分岐のディスプレースメントとしてwビットの値を指定できる。この値をインデックスとして用いて、ループ・キャッシュ内のエントリをアクセスする。ループ・キャッシュには2^w命令が格納でき、タグは格納しない。ループ・キャッシュから命令をフェッチし続けている間は、メイン・キャッシュは動作を停止するので消費電力が減る。32命令程度の規模のループ・キャッシュで、メイン・キャッシュへのアクセス回数が38%減ることを報告している[14]。

2.3. ダイナミック電力低減技術－パイプライン

パイプラインの構成方法や制御方法を工夫することで低電力化する技術として、パイプライン・ゲーティングと、狭ビット幅オペランドの動的活用、が提案されている。

2.3.1. パイプライン・ゲーティング

スーパースカラープロセッサでは、アウトオブオーダー実行や投機実行を駆使して性能を上げる。分岐予測が当たるという仮定の下で、次々に命令をフェッチしパイプラインに投入しながら処理が進められるが、分岐予測は外れる場合もある。予測がミスした場合には、実行結果は捨てられ、コミットは行われないが、無駄な電力を消費していることになる。パイプライン・ゲーティングは、各分岐予測に対して、予測の信頼度を検知する回路を設け、信頼度の低い予測であると判断した場合には、その予測に基づく命令フェッチを停止（ゲーティング）する[15]。これによって、wrong-pathの命令をできるだけパイプラインに入れないようにしている。この方式の評価は、式(2)に示す指標EW（Extra Work）で行っている。

$$EW = \frac{\text{フェッチした命令数} - \text{コミットした命令数}}{\text{コミットした命令数}} \quad (2)$$

EWが大きいことは、フェッチした命令のうちコミットされたものが少ないことを示し、消費電力の無駄が大きいことを意味する。SPECベンチマークで評価した結果、パイプライン・ゲーティングによってEWが最大30%減少することが報告されている[15]。

2.3.2. 狭ビット幅オペランドの動的活用

ハイエンドのCPUは32ビットから64ビットへシフトしていく傾向にあるが、一方、整数を扱うプログラムでは、64ビットのダイナミックレンジが必要な場合は極めてまれである。64ビットCPUであるAlphaに対し、SPECint95を使って分析した結果、16ビット以下のオペランドが約50%を占めることが報告されている[16]。16ビット以下のオペランドに対し、64ビットの整数ユニットを動作させるのは、消費電力的に無駄が大きい。この視点から、オペランドのビット幅を動的に検出し、狭ビットならファンクションユニットの上位ビットの遷移を停止する方式が提案された。SPECint95、MediaBenchでの評価では、この方式により、それぞれ54%、58%の電力削減が行えることが判明している[16]。

3. マイクロアーキテクチャ・レベルでの電力見積り技術

アーキテクチャ設計での意思決定は、性能および消費電力に非常に大きな影響を与える。例えば、プロセッサの設計で、命令キャッシュやデータキャッシュの容量、連想方式によって、性能と消費電力の間にトレードオフが生ずる。分岐予測の方式やブランチ・ターゲット・バッファのサイズ、ファンクションユニットの個数、命令発行幅といったパラメータも、性能と消費電力に大きく影響する。アーキテクチャ・レベルで消費電力を見積るCADツールが必要となるが、世の中に出回っている消費電力解析ツールは、ほとんどがRTLより下位レベルの設計に対応したものである。

数は少ないながらも、CPUのマイクロアーキテクチャ・レベルで性能と消費電力のトレードオフ解析を行うツールが研究され、報告されている。本節では、その代表的なツールであるWattchについて紹介するとともに、IBMやIntelといった企業で開発されたツールについても述べる。

3.1. Wattch

Wattchは、プリンストン大学で研究開発され、公開されているツールであり[6]、アーキテクチャ・レベルのプロセッサシミュレータSimpleScalarと連動して動作する。Wattchの構造を図4に示す。SimpleScalarも大学で開発され公開されているツールである[17]。Wattchとの連動では、プロセッサのアーキテクチャとしてMIPSライクのPisa、およびAlphaがサポートされている。キャッシュ込みのシミュレーションが行え、消費電力と性能のトレードオフ解析ができる。SimpleScalarは、入力された命令ストリームに対し、CPU内部のハードウェアの動作を、サイクル単位でシミュレートする。これによって、どのサイクルにどのハードウェア部品がアクセスされるかが分かるので、

この情報を元に消費電力を見積る。

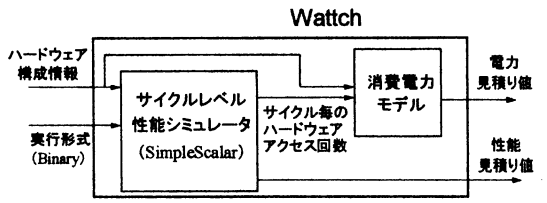


図 4. Wattch の構成

消費電力の見積りでは、以下のような消費電力モデルが使用されている。Wattch では、プロセッサの内部構成要素を 4 種類に分類して、負荷容量 C をモデル化し、ダイナミック電力を計算している。

①アレイ構造

キャッシュやレジスタファイル等がこの分類に入る。ロウ数、カラム数、read/write ポート数をパラメータにして、ワード線やビット線の負荷容量を数式で表現している。

②CAM 構造

命令ウィンドウ、TLB、ロードストア・キューの一部がこの分類に入る。CAM のタグ線やマッチ線の負荷容量を数式で表現している。

③組合せ論理と配線

ALU 等のファンクションユニットや、依存関係のチェックロジックがこの分類に入る。これらのモデル化は非常に難しいため、文献で報告された過去の設計データをスケーリングして用いている。

④クロック

クロックの負荷容量が、1)グローバルクロックの配線、2)グローバル・クロックバッファ、3)パイプラインレジスタ等のゲート容量、から構成されると仮定し、モデル化している。1)では、グローバルクロック信号がチップのすみずみまで行き渡ると仮定し、H-tree をベースにしたモデル化を行っている。一方、2)は過去の設計データを元に見積り、3)はパイプラインレジスタの数から見積る。

Wattch の見積り精度については、PentiumPro、MIPS-R10K、Alpha21264 で評価した結果が示されており、実際の値と Wattch の見積り値との差は 30%程度と報告されている[6]。

3.2. IBM の PowerTimer

将来の PowerPC 開発に向けて IBM で作られた、マイクロアーキテクチャ・レベルの電力解析ツールである[18]。IBM 内製のサイクルレベル・プロセッサシミュレータ Turandot/MET と連動して動く。PowerTimer は Wattch と異なり、消費電力モデルとして、回路レベルの電力シミュレーションデータに基づく empirical model を用いている。

3.3. Intel の ALPS

ALPS (Architectural-Level-Power-Simulator) は、Intel で開発され、Pentium4 の設計で使用された[19]。ファンクションユニットの消費電力は、過去のデータからの外挿で求め、消費電力モデルを構築する。アーキテクチャ・レベルの性能シミュレータと、消費電力モデルを連動させ、消費電力の見積りを行う。

4. 最近の話題

以上述べた低消費電力や電力見積り技術は、いずれもダイナミック電力を対象にしたものである。一方、半導体の微細化技術の進歩とともに、リーク電力の増大が指摘されており、リーク電力の低減技術が注目を浴びつつある。動作時の消費電力に関する ITRS の予測によると、65nm の世代でダイナミック電力とリーク電力が大きさとして肩を並べるようになる[20]。動作時の消費電力を低減するには、ダイナミック電力を減らすだけでは達成できず、動作時のリーク電力を減らす必要がある。

動作時リーク電力を低減するマイクロアーキテクチャ・レベルの技術として、①キャッシュのリーク低減技術、②ファンクションユニットのリーク低減技術が報告されている。

4.1. キャッシュの動作時リーク電力を減らす技術

キャッシュは通常 SRAM で構成されるが、SRAM のメモリセルにパワースイッチを挿入し、動作中に、ライン単位でスリープ(すなわちパワースイッチをオフ)させてリークを低減する。この技術が Gated-Vdd である[21]。一方、Gated-Vdd の応用として、Cache Decay と呼ばれる手法[22]が提案されている。Cache Decay 手法では、キャッシュのラインがアクセスされてから、一定期間（インターバル）の間、次のアクセスがなければ、そのラインは再び使われることなくリプレイスされると判断し、スリープさせる。ラインのスリープには Gated-Vdd を使う。インターバルを短く設定すると、スリープするラインが多くなりリークは減るが、キャッシュミスが増えるため、その分の電力増加がある。インターバルを長くした場合には、逆のことが起こる。これらの事実から、リークエネルギーを最小にする最適インターバルが存在する[22]。

Gated-Vdd ではスリープさせたラインのデータが消失してしまうため、この欠点を補う方式として提案さ

れたのが Drowsy cache である[23]。Drowsy cache では、ラインに2つのモード(awake モードと drowsy モード)を持たせる。awake モードでは、ラインには通常の電源電圧 *VDD* を与え動作させる。一方、drowsy モードでは、低電源電圧 *VDD_low* を与えてデータ保持させる。低電源電圧 *VDD_low* でのデータ保持は、低リークで行える。定期的に全ラインを drowsy モードに入れ、アクセスされたラインだけ wake-up させる。

4.2. ファンクションユニットの動作時リーク低減技術

前節で述べた Gated-Vdd や Cache Decay は、キャッシュ（SRAM）のメモリセルに対してパワーゲーティングを行う方式であった。これに対し、ファンクションユニットに対してパワーゲーティングを行う方式が報告されている[24]。構成を図5に示す。CPU の中の固定小数点ユニット（FXU）と浮動小数点ユニット（FPU）に、それぞれパワースイッチを付け、CPU の走行中に選択的にスリープさせる。スリープを開始する方式として、①ファンクションユニットが一定期間アイドルになっていたらスリープさせる、②分岐予測が外れたらスリープさせる、というアルゴリズムが提案されている。

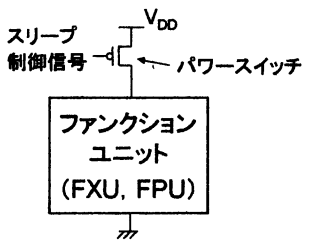


図5. ファンクションユニットのパワーゲーティング

IBM の Power4 を対象にしたシミュレータで評価した結果、この方式により、FPU は全実行時間の 28%、FXU は全実行時間の 12%、スリープ可能であることが報告されている[24]。

一方、筆者らの研究グループでは、さらに細粒度のリーク制御の研究を進めており、CPU 中のデータパス構成要素の動作時リーク低減方式[25]や、状態遷移回路の動作時リーク低減方式[26]について報告している。

5. おわりに

アーキテクチャ・レベルの低消費電力技術は、RTL より下位の技術に比べて設計の自由度が大きいことから、電力削減効果が大きい。本論文では、高位設計における低消費電力技術のなかでも、とくにプロセッサのマイクロアーキテクチャ・レベルでの低消費電力技術に焦点を当てて述べた。今後、ダイナミック電力を低減する技術に加え、動作時リーク電力を低減する技術の必要性が著しく高まっていくと予想される。さらに、動作時リーク電力の低減には、マイクロアーキテクチャ・レベルの技術と回路技術の密接なコラボレーションが必要と考えられる。

参考文献

[1] 桜井編「低消費電力、高速 LSI 技術」, リアライズ社, 1998.

[2] 榎本編「画像 LSI システム設計技術」, コロナ社, 2003.

[3] http://www.chipvision.com/

[4] Montanaro, *JSSC*, Vol.31, No.11, pp.1703-1714, Nov. 1996.

[5] *Microprocessor Report*, January 21, 2003.

[6] Brooks, et al, *ISCA*, pp.83-94, 2000.

[7] Rol, *ISSCC*, pp.208-209, 1994.

[8] Kin, et al, *Int. Symp. on Microarchitecture*, pp.184-193, 1997.

[9] Su, *Hawaii Int. Conf. on Sys. Sci.*, pp.306-315, 1995.

[10] Su, et al, *ISLPD*, pp.63-68, 1995

[11] Ghose, et al, *ISLPED*, pp.70-75, 1999.

[12] Hasegawa, et al, *IEEE Micro*, pp.11-19, Dec. 1995.

[13] Inoue, et al, *ISLPED*, pp.273-275, 1999.

[14] Lee, et al, *ISLPED*, pp.267-269, 1999.

[15] Manne, et al, *ISCA*, pp.132-141, 1998.

[16] Brooks, et al, *HPCA*, pp.13-22, 1999.

[17] http://www.simplescalar.com/

[18] Brooks, et al, *IBM J. Res & Dev.*, pp.653-670, Sep/Nov 2003.

[19] Gunther, et al, *Intel Technology Journal*, Q1, 2001, pp.1-9.

[20] Sung, et al, *IEEE Computer*, pp.68-75, December 2003.

[21] Powell, et al, *ISLPED*, pp.90-95, 2000.

[22] Kaxiras, et al, *ISCA*, pp.240-251, 2001.

[23] Flautner, et al, *ISCA*, pp.148-157, 2002.

[24] Hu, et al, *ISLPED*, pp.32-37, 2004.

[25] 森, 宇佐美, 電子情報通信学会東京支部大会予稿集, 2005 年 3 月.

[26] Usami, Yoshioka, *IEICE Trans. On Fundamentals*, Vol. E87-A, No.12, pp.3116-3123, Dec. 2004.

（なお、*ISLPED* は IEEE International Symposium on Low Power Electronics and Design, *ISCA* は ACM/IEEE International Symposium on Computer Architecture, *HPCA* は IEEE International Symposium on High Performance Computer Architecture を表す。）