マルチタスク組込みアプリケーションの低消費エネルギー化のためのメモリ管理技術

山口, 誠一朗
九州大学大学院システム情報科学府

室山, 真徳
九州大学システムLSI 研究センター

石原, 亨
九州大学システムLSI 研究センター

安浦, 寛人
九州大学大学院システム情報科学研究院

出版情報：電子情報通信学会技術研究報告, VLD2007-74. 107 (334), pp.25-29, 2007-11. 電子情報通信学会
バージョン：
権利関係：

マルチタスク組込みアプリケーションの低消費エネルギー化のためのメモリ管理技術

山口誠一朗† 室山 真徳†† 石原 亨†† 安浦 寛人†††

† 九州大学 大学院システム情報科学府
〒 819–0395 福岡市西区元岡 744 番地
†† 九州大学 システム LSI 研究センター
〒 814–0001 福岡市早良区百道浜 3 丁目 8 番 33 号
††† 九州大学 大学院システム情報科学研究院
〒 819–0395 福岡市西区元岡 744 番地
E-mail: †seiichiro@c.csce.kyushu-u.ac.jp, ††{muroyama,ishihara}@slrc.kyushu-u.ac.jp,
†††yasuura@c.csce.kyushu-u.ac.jp

あらまし 携帯情報機器をはじめとした組込みシステムではメモリ・システムのエネルギー削減が強く求められている．キャッシュ・メモリよりもアクセスに要するエネルギーが小さいスクラッチパッド・メモリを利用した静的コード配置手法などが盛んに研究されている．しかしながら，マルチタスク組込みアプリケーションをターゲットとした場合効果は少ない．ソフトウェアの記述量が増える一方，スクラッチパッド・メモリの容量には限界がある．本稿では，マルチタスク組込みアプリケーションを対象として，消費エネルギー削減に適したメモリ・アーキテクチャを提案する．また，提案したアーキテクチャをターゲットとしたメモリ管理技術についても議論する．

キーワード 組込みシステム，低消費エネルギー，メモリ管理，マルチタスクアプリケーション

A Memory Management Technique for Energy Reduction in Multi-Task Embedded Applications

Seiichiro YAMAGUCHI†, Masanori MUROYAMA††,

Tohru ISHIHARA††, and Hiroto YASUURA†††

† Graduate School of Information Science and Electrical Engineering, Kyushu University
Motooka 744, Nishi-ku, Fukuoka-shi, 819–0395 Japan
†† System LSI Research Center, Kyushu University
Momochihama 3–8–33, Sawara-ku, Fukuoka-shi, 814–0001 Japan
††† Faculty of Information Science and Electrical Engineering, Kyushu University
Motooka 744, Nishi-ku, Fukuoka-shi, 819–0395 Japan
E-mail: †seiichiro@c.csce.kyushu-u.ac.jp, ††{muroyama,ishihara}@slrc.kyushu-u.ac.jp,
†††yasuura@c.csce.kyushu-u.ac.jp

Abstract Memory systems consume a significant amount of the energy in embedded systems. Static code placement techniques using scratchpad memory which dissipate the energy less than cache memory have been proposed. However, these techniques do not have much effectiveness on multi-task embedded applications. This paper presents a novel memory architecture which is suitable for multi-task embedded applications. A memory management technique which targets proposed memory architecture is also discussed.

Key words Embedded Systems, Energy Reduction, Memory Management, Multi-Task Applications

1. はじめに

携帯電話や携帯ゲーム機，携帯音楽プレイヤーなどの携帯情報機器が非常に普及している．現在の携帯電話ではゲーム，音楽再生，写真撮影，テレビ視聴などの機能も一つの携帯電話端末に実装されている．今後も複数の機能が一つの携帯情報機器で実現されると推測できる．携帯情報機器をはじめとした多くの組込みシステムの課題の一つに消費エネルギーの削減が挙げられる．特に，携帯情報機器はバッテリー駆動であるため，消費エネルギーは機器の連続使用時間に大きな影響を与える．近年では，ハードウェア技術とソフトウェア技術で協調して消費エネルギーを削減することが期待されている．

組込みシステムで利用されるマイクロプロセッサでは，オンチップ・メモリの消費電力（消費エネルギーの時間微分）の割合が大きい．ARM920T マイクロプロセッサではキャッシュ・メモリ（以下，キャッシュ）の消費電力が 44%（命令用：25%，データ用：19%）を占めている [17]．また，StrongARM SA-110 マイクロプロセッサでも 43%（命令用：27%，データ用：16%）の電力がキャッシュで消費されている [14]．一方で，オフチップ・メモリへのアクセスで消費されるエネルギーはオンチップ・メモリよりも非常に大きい．したがって，メモリ・システムの低消費エネルギー化を考える場合，オフチップ・メモリも考慮する必要がある．本研究の最終的な目的は，マイクロプロセッサ上で複数のアプリケーションが実行される状況において，オンチップ・メモリのみならずオフチップ・メモリを含めたメモリ・システム全体での消費エネルギーを削減するメモリ管理技術を提案することである．

オンチップ・メモリにはキャッシュ，スクラッチパッド・メモリ（以下，SPM）および用途が限定されたメモリがある．また，それぞれのメモリで命令用とデータ用に分ける場合（ハーバード・アーキテクチャ）と分けない場合（フォン・ノイマン・アーキテクチャ）がある．キャッシュの動作はハードウェアが完全にサポートしており，ソフトウェアのレイヤからは全く意識することなく使用することができる．一方，SPM は消費電力，レイテンシ，面積などに関してキャッシュよりも優れていることが知られているが，キャッシュとは異なりソフトウェアのレイヤからの制御やコード／データ配置の技術が必要である．用途が限定されたメモリの例としては，特別なコードやデータを保存するメモリがある．システム設計者はキャッシュ，SPM および用途が限定されたメモリの特徴を考慮し，実装するアプリケーションに応じて適切なメモリ・アーキテクチャを構築，または適切なメモリ・アーキテクチャが構築されているマイクロプロセッサ IP（Intellectual Property）を選択する．本稿では，ソフトウェア技術と協調して消費エネルギーを削減できるようなメモリ・アーキテクチャを提案する．

本稿の構成は以下の通りである．第 2 章ではオンチップ・メモリの消費エネルギー削減に関する関連研究について述べる．提案するメモリ・アーキテクチャは第 3 章で説明する．第 4 章ではメモリ管理技術の提案に向けての検討を行い，最後に第 5 章で本稿をまとめる．

2. 関連研究

オンチップ・メモリの消費エネルギー削減手法はこれまでにも数多く提案されてきた．キャッシュに関する手法としては，次回アクセスされるウェイを予測して無駄なアクセスを削減する手法がある [6]〜[8], [13]．あるセットで最も最近アクセスされたウェイ “MRU（Most Recently Used）ウェイ” を記憶しておき，次回そのセットにアクセスするときはまず MRU ウェイにのみアクセスし，MRU ウェイでヒットしなかった場合は残りのウェイにアクセスする [6]．セット数 $\times \lceil \log_2$ ウェイ数 $\rceil$ 分の容量のメモリを用意し，キャッシュへアクセスする前にアドレスから MRU ウェイを検索する．文献 [7], [8] ではさらに，予測されたウェイにおいてタグ・アクセスおよびタグ比較を無くすために MRU ウェイのタグの値まで記憶している．しかし，タグの値まで記憶する必要があるため，メモリの容量は大きくなる．全てのセットの情報を記憶するのではなく，アクセスの新しいセットから複数セットの情報のみを記憶することでメモリ容量を削減している．

容量が小さいキャッシュを CPU コアと L1 キャッシュの間に用意する手法もあり，Filter Cache [12]，Block Buffering [4], [20]，S-cache [15]，L-Cache [5] などがある．Filter Cache は先に述べた手法の中では最も単純な手法であり，キャッシュの特徴を何も変えず，容量が小さい，すなわちアクセスに必要なエネルギーが小さいキャッシュを CPU コアと L1 キャッシュの間に挿入するだけである [12]．Filter Cache はアクセスの時間的局所性が高い場合に効果が大きい．Block Buffering はキャッシュへアクセスした際に取得したキャッシュ・ブロックのコード／データおよびそのブロック・アドレスをバッファに保持しておく [4], [20]．次のアクセスのブロック・アドレスがバッファに保持しているブロック・アドレスと等しいとき，取得したいコード／データはバッファ内に存在するためキャッシュを活性化させる必要はない．S-cache は頻繁に実行される基本ブロックのコードを専用の小さなメモリに保持することでキャッシュのタグ比較などが不要になり消費エネルギーを削減している [15]．頻繁に実行される基本ブロックを適切なアドレスに配置する必要があり，S-cache は SPM に似たメモリである．L-Cache は S-cache が対象とする基本ブロックをループ内の基本ブロックに限定した手法である [5]．

命令を圧縮することで一度に読み出す命令数を増やし，アクセス回数を削減する手法がある [2], [9], [26]．圧縮された命令を解凍するためのメモリや回路が必要になる．文献 [26] では，プログラム中の全ての命令を圧縮している．この手法はアプリケーションを後から追加したり更新したりすることはできない．また，プログラムが大規模になれば圧縮率も下がってしまう．この問題点を解決するために，圧縮する命令を頻繁に実行されるいくつかの命令に限定した手法が文献 [2] で提案されている．例えば，頻繁に実行される 256 個の命令を圧縮した場合，1 命令を 8 ビットに圧縮できる．1 ワードを 32 ビットとすると 4 倍の命令を一度に取得でき，メモリへのアクセス回数を削減できる．文献 [9] では，頻繁に実行される連続した命令列を圧縮対

象としている．基本ブロック内の連続した命令列を一つに圧縮することでメモリへのアクセス回数を削減できる．

キャッシュはタグの読み出しおよび比較が必要であり，ウェイ数（連想度）分のコード／データも読み出す必要があるため，SPM に比べアクセスに必要なエネルギーが大きい．したがって，頻繁にアクセスされるコード／データを SPM に配置することでエネルギーを削減できる [1], [10], [19], [23]．文献 [1] は，グローバル変数とスタックのデータを対象として SPM に配置する手法である．SPM に配置する対象をコードとした手法は文献 [23] で，関数，基本ブロックおよび変数を対象とした手法は文献 [10], [19] で提案されている．文献 [10] ではさらに，キャッシュのコンフリクト・ミスを削減するコード配置や，参照の時間的局所性が低いコード／データをキャッシュしない領域へ配置している．

近年ソフトウェアの規模が大きくなり，また複数のアプリケーションを実装している機器も多くなってきた．ソフトウェアのコード量／データ量が増えたことで，頻繁に実装されるコードや頻繁に参照されるデータの量が増えれば，SPM の容量の制限から全てを SPM に配置できず，SPM へのコード／データ配置でのエネルギー削減効果は薄れてしまう．そこで，実行されるタスク単位で頻繁に実行されるコードや頻繁に参照されるデータをあらかじめ定めておき，コンテキスト・スイッチが発生した際に SPM をオーバーレイする動的 SPM 管理手法が提案されている [3], [11], [16], [18], [21], [22], [24], [25]．SPM から追い出されるデータをオフチップ・メモリに書き出し，オーバーレイするコード／データをオフチップ・メモリから読み出す必要がある．

3. マルチタスクに適したメモリ・アーキテクチャ

メモリで消費されるエネルギーを削減するためには，アクセス・エネルギーが小さいメモリへアクセスすることが望ましい．したがって，高頻度でアクセスされるコード／データをアクセス・エネルギーが小さいメモリに配置すればよい．しかし，ソフトウェアの規模の増大や実装されるアプリケーションの種類の増加により，高頻度でアクセスされるコード／データを全て同時に保持することは難しくなった．メモリ容量の限界である．前章で述べた動的 SPM 管理手法は，この問題を解決する手法の一つである．しかし，SPM のオーバーレイの際にオフチップ・メモリへアクセスするため，メモリ・システム全体での消費エネルギー削減を考える場合，オーバーレイの頻度に注意する必要がある．本章では，メモリのエネルギー・モデルを示し，複数のタスクが実行される状況でメモリ・システム全体の消費エネルギーを削減できるメモリ・アーキテクチャを提案する．

3.1 エネルギー・モデル

メモリ・システム全体での消費エネルギーを削減するためには，各々のメモリのエネルギー・モデルを考慮する必要がある．ここでは，キャッシュ，SPM およびオフチップ・メモリのエネルギー・モデルを示す．エネルギー・モデルで使用するパラメータを以下のように定義する．

- E_{CR}：キャッシュ・リードのエネルギー
- E_{CW}：キャッシュ・ライトのエネルギー
- E_{SR}：SPM リードのエネルギー
- E_{SW}：SPM ライトのエネルギー
- E_{OR}：オフチップ・メモリ・リードのエネルギー
- E_{OW}：オフチップ・メモリ・ライトのエネルギー
- N_{way}：キャッシュの連想度
- E_{tagR}：タグ・リードのエネルギー
- E_{tagW}：タグ・ライトのエネルギー
- E_{dataR}：データ・リードのエネルギー
- E_{dataW}：データ・ライトのエネルギー
- R_{missR}：キャッシュ・リード・ミス率
- R_{missW}：キャッシュ・ライト・ミス率
- E_{missR}：キャッシュ・リード・ミスのエネルギー・オーバーヘッド
- E_{missW}：キャッシュ・ライト・ミスのエネルギー・オーバーヘッド
- E_{bus}：オフチップ・バスのエネルギー
- E_{DRAMR}：DRAM リードのエネルギー
- E_{DRAMW}：DRAM ライトのエネルギー

キャッシュのリード／ライト，SPM のリード／ライト，オフチップ・メモリのリード／ライトのアクセスあたりのエネルギーはぞれぞれ式 (1)，(2)，(3)，(4)，(5) および (6) で求めることができる．

$$E_{CR} = N_{way} \cdot (E_{tagR} + E_{dataR}) + R_{missR} \cdot E_{missR} \quad (1)$$

$$E_{CW} = N_{way} \cdot E_{tagR} + E_{tagW} + E_{dataW} + R_{missW} \cdot E_{missW} \quad (2)$$

$$E_{SR} = E_{dataR} \quad (3)$$

$$E_{SW} = E_{dataW} \quad (4)$$

$$E_{OR} = E_{bus} + E_{DRAMR} \quad (5)$$

$$E_{OW} = E_{bus} + E_{DRAMW} \quad (6)$$

3.2 メモリ・アーキテクチャ

提案するマルチタスクに適したメモリ・アーキテクチャでは，頻繁に参照されるコード／データを SPM ではなく，小容量のフル・セット・アソシアティブ・キャッシュを多少変更したメモリに保持する．本稿では，このフル・セット・アソシアティブ・キャッシュを Frequently Accessed Cache（FA-Cache）と呼ぶ．FA-Cache はタスク毎にコンパイル時に決定された特定のコード／データのみ保持できるメモリである．また，FA-Cache に保持するように割り当てられたコード／データでも FA-Cache の利用状況に応じて，L1 キャッシュに保持することができる機構をもつ．

図 1 に提案するメモリ・アーキテクチャのメモリ階層を，図 2 に動的 SPM 管理手法で利用されるメモリ階層を示す．動的

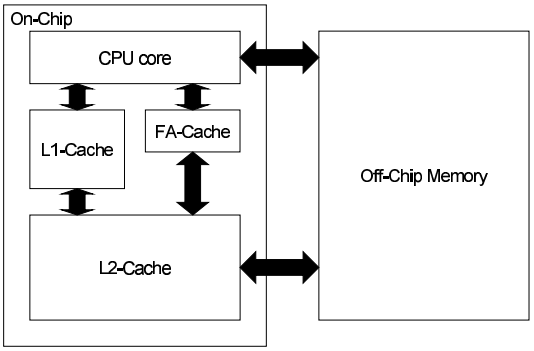

図 1　提案するメモリ・アーキテクチャのメモリ階層
Fig. 1　A memory hierarchy for our memory architecture.

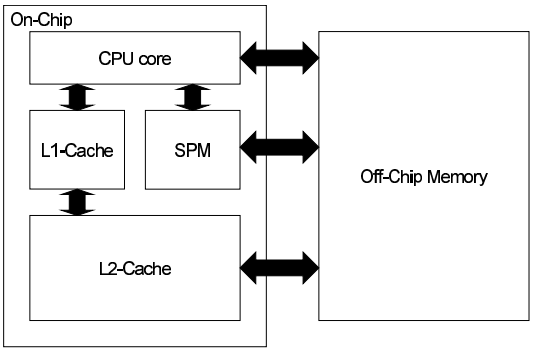

図 2　動的 SPM 管理手法で利用されるメモリ階層
Fig. 2　A memory hierarchy for dynamic scratchpad management.

SPM 管理手法ではオーバーレイの際，必ずオフチップ・メモリへのアクセスが発生する．しかし，FA-Cache では L2 キャッシュに該当コード／データが存在しない場合にのみオフチップ・メモリへのアクセスが発生する．オフチップ・メモリへのアクセスはエネルギーを多く消費するため，L2 キャッシュにコード／データが存在すればエネルギー削減効果は大きい．

動的 SPM 管理手法では頻繁に参照されるコード／データはコンパイル時に SPM のどのアドレスにオーバーレイするか決定されている．ここで，コード／データを同じアドレスにオーバーレイするように割り当てられた二つのタスク，タスク A およびタスク B を考える．タスク A とタスク B との間でコンテキスト・スイッチが頻繁に発生すると SPM のオーバーレイも頻繁に発生することとなり，オフチップ・メモリへのアクセスが増大し，消費エネルギーのみならず性能のオーバーヘッドまで大きくなる．一方，FA-Cache はフル・セット・アソシアティブ・キャッシュであるため，タスク A およびタスク B の頻繁にアクセスされるコード／データが存在するアドレスに関わらず記憶することができる．

FA-Cache の詳細な構成を図 3 に示す．最も注目すべき特徴は全てのタグがラッチで構成されており，CPU コア内に存在することである．SPM はアドレスが固定されているため，パイプラインの命令フェッチのステージの時には既に L1 キャッシュへアクセスすべきか SPM へアクセスすべきか決定している．しかし，FA-Cache が保持しているコード／データのアドレスは固定されていない．そこで，命令フェッチのステージの前に

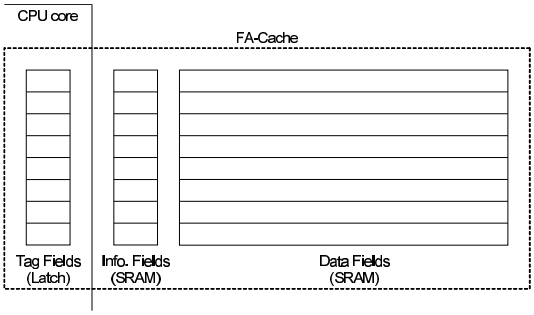

図 3　FA-Cache のアーキテクチャ
Fig. 3　FA-Cache architecture.

FA-Cache の全てのタグとアクセスしようとしているアドレスを比較し，FA-Cache にコード／データが存在するか否かを判定することで，どちらにアクセスすべきか決定する．タスク毎にコンパイル時に決定する頻繁にアクセスされるコード／データは，FA-Cache のデータ・フィールドのブロック・サイズ単位で連続したアドレスへ配置する．タスクによっては FA-Cache を使用しなかったり，複数のブロックを使用してもよい．複数のブロックを使用する場合，FA-Cache の利用状況によっては一部のみ L1 キャッシュに保持される場合もある．FA-Cache の利用状況や使用優先度，フラグなどは情報フィールドに保持する．利用状況にはタスクの状態（強制待ち状態，休止状態，実行可能状態など）を保持することもできる．

4.　メモリ管理技術の検討

前章で提案したメモリ・アーキテクチャを利用してメモリ・システム全体の消費エネルギーを最小化するメモリ管理技術を提案することが最終的な目標であるが，本稿ではその目標に向けての検討を行う．

4.1　メモリ・オブジェクト

FA-Cache に配置するメモリ・オブジェクトの候補としては以下の 4 つがある．

- 関数
- 基本ブロック
- ローカル変数
- ローカル定数

ローカル変数のようなデータは，値が変更された場合は書き戻す必要があるため，管理がやや複雑になる．

4.2　コード配置

シングルタスクの静的コード配置に関する研究は数多く存在する [1], [10], [19], [23]．これらの研究で定義されている問題を改良すれば我々のコード配置問題に拡張できると考える．本稿で提案したメモリ・アーキテクチャにはフル・アソシアティブ・キャッシュである FA-Cache が存在するため，タスク間のキャッシュ・コンフリクトを考える必要がないからである．配置するコード／データ量は FA-Cache のブロック・サイズによって決まる．

4.3　FA-Cache 使用優先度

本研究の最も重要な事項である．FA-Cache に全てコード／

データが保持されているときでも情報フィールドにある利用状況や使用優先度，フラグの状態をさまざま考慮して，FA-Cache の使用を許可するか否かを決定する．また，許可した場合，どのキャッシュ・ブロックを上書きするか決定する必要がある．

5. おわりに

本稿では，オンチップ・メモリのみならずオフチップ・メモリまで考慮したメモリ・システム全体を考え，マルチタスク組込みアプリケーションを対象としたときに消費エネルギーを削減することができるメモリ・アーキテクチャを提案した．フル・アソシアティブ・キャッシュを改良したメモリを使用することで，従来のスクラッチパッド・メモリの動的管理手法に比べ，オフチップ・メモリへのアクセスを削減することを目的としている．またメモリ管理技術の提案に向けての検討を行った．今後の課題は FA-Cache 使用優先度決定問題を定め，またコード配置問題を設定し，メモリ・システムの消費エネルギーモデルを用いた定量的な評価を行うことである．

謝辞 本研究の一部は，科学技術振興事業団（JST）の戦略的創造研究推進事業（CREST）「情報システムの超低消費電力化を目指した技術革新と総合化技術」の支援によるものである．

文　献

[1] O. Avissar, R. Barua and D. Stewart, “An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems,” *IEEE Trans. on Embedded Computing Systems*, Vol. 1, No. 1, pp. 6–26, Nov. 2002.

[2] L. Benini, A. Macii, E. Macii et al., “Selective Instruction Compression for Memory Energy Reduction in Embedded Systems,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 206–211, Aug. 1999.

[3] P. Francesco, P. Marchal, D. Atienza et al., “An Integrated Hardware/Software Approach For Run-Time Scratchpad Manegement,” In *Proc. of Design Automation Conference*, pp. 238–243, Jun. 2004.

[4] K. Ghose and M. B. Kamble, “Analytical Energy Dissipation Models for Low Power Caches,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 143–148, Aug. 1997.

[5] N. B. I. Hajj, G. Stamoulis, N. Bellas et al., “Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 70–75, Aug. 1998.

[6] K. Inoue, T. Ishihara and K. Murakami, “Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 273–275, Aug. 1999.

[7] K. Inoue, V. G. Moshnyaga and K. Murakami, “A Low Energy Set-Associative I-Cache with Extended BTB,” In *Proc. of International Conference on Computer Design*, pp. 148–153, Sep. 2002.

[8] T. Ishihara and F. Fallah, “A Way Memoization Technique for Reducing Power Consumption of Caches in Application Specific Integrated Processors,” In *Proc. of Design, Automation and Test in Europe*, Vol. 1, pp. 358–363, Mar. 2005.

[9] T. Ishihara and H. Yasuura, “A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors,” In *Proc. of Design, Automation and Test in Europe*, pp. 617–623, Mar. 2000.

[10] Y. Ishitobi, T. Ishihara and H. Yasuura, “Code Placement for Reducing the Energy Consumption of Embedded Processors with Scratchpad and Cache Memories,” In *Proc. of IEEE Workshop on Embedded Systems for Real-Time Multimedia*, pp. 13–18, Oct. 2007.

[11] M. Kandemir, J. Ramanujam, M. J. Irwin et al., “Dynamic Management of Scratch-Pad Memory Space,” In *Proc. of Design Automation Conference*, pp. 690–695, Jun. 2001.

[12] J. Kin, M. Gupta and W. H. M.-Smith, “The Filter Cache: An Energy Efficient Memory Structure,” In *Proc. of International Symposium on Microarchitecture*, pp. 184–193, Aug. 1997.

[13] A. Ma, M. Zhang and K. Asanović, “Way Memoization to Reduce Fetch Energy in Instruction Caches,” In *Proc. of Workshop on Complexity Effective Design*, Jul. 2001.

[14] J. Montanaro, R. T. Witek, K. Anne et al., “A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor,” *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 11, pp. 1703–1714, Nov. 1996.

[15] R. Panwar and D. Rennels, “Reducing the frequency of tag compares for low power I-cache design,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 57–62, Aug. 1995.

[16] R. Pyka, C. Faßbach, M. Verma et al., “Operating system integrated energy aware scratchpad allocation strategies for multiprocess applications,” In *Proc. of International Workshop on Software and Compilers for Embedded Systems*, pp. 41–50, Apr. 2007.

[17] S. Seger, “Low-Power Design Techniques for Microprocessors (Tutorial),” *International Solid State Circuits Conference*, Feb. 2001.

[18] S. Steinke, M. Knauer, L. Wehmeyer et al., “Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory,” In *Proc. of International Symposium on System Synthesis*, pp. 213–218, Oct. 2002.

[19] S. Steinke, L. Wehmeyer, B.-S. Lee et al., “Assigning Program and Data Objects to Scratchpad for Energy Reduction,” In *Proc. of Design, Automation and Test in Europe*, pp. 409–415, Mar. 2002.

[20] C.-L. Su and A. M. Despain, “Cache Design Tradeoffs for Power and Performance Optimization: A Case Study,” In *Proc. of International Symposium on Low Power Design*, pp. 63–68, Apr. 1995.

[21] S. Udayakumaran, A. Dominguez and R. Barua, “Dynamic Allocation for Scratch-Pad Memory Using Compile-Time Decisions,” *ACM Trans. on Embedded Computing Systems*, Vol. 5, No. 2, pp. 472–511, May 2006.

[22] M. Verma, K. Petzold, L. Wehmeyer et al., “Scratchpad sharing strategies for multiprocess embedded sytems: A first approach,” In *Proc. of IEEE Workshop on Embedded Systems for Real-Time Multimedia*, pp. 115–120, Sep. 2005.

[23] M. Verma, L. Wehmeyer and P. Marwedel, “Cache-aware Scratchpad Allocation Algorithm,” In *Proc. of Design, Automation and Test in Europe*, Vol. 2, pp. 1264–1269, Feb. 2004.

[24] M. Verma, L. Wehmeyer and P. Marwedel, “Dynamic Overlay of Scratchpad Memory for Energy Minimization,” In *Proc. of International Conference on Hardware/Software Codesign and System Synthesis*, pp. 104–109, Sep. 2004.

[25] L. Wehmeyer, U. Helmig and P. Marwedel, “Compiler-optimized usage of partitioned memories,” In *Proc. of Workshop on Memory Performance Issues*, pp. 114–120, Jun. 2004.

[26] Y. Yoshida, B.-Y. Song, H. Okuhata et al., “An Object Code Compression Approach to Embedded Processors,” In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 265–268, Aug. 1997.