Responsive Multithreaded Processorの設計・実装

伊藤 務 † 内山 真郷 †† 佐藤 純一 †††
薄井 弘之 † 松浦 克彦 † 山崎 信行 †

本論文は，分散リアルタイム処理用プロセッサである *Responsive MultiThreaded*（*RMT*）*Processor*のプロセッシングユニットである *RMT Processing Unit*（*RMT PU*）について述べる．*RMT PU*は 8way の細粒度マルチスレッディングアーキテクチャであり，優先度を用いて計算資源の競合を調停することにより，優先度の高いスレッドを優先的に実行する．さらにチップ内にコンテキスト専用のキャッシュを持ち，ハードウェアでコンテキストスイッチを行うことにより，コンテキストスイッチにかかるオーバヘッドを削減する．これにより，ハードウェアレベルでリアルタイム処理を支援し，より短かい時間粒度でリアルタイム処理の制御を可能とする．

The Design and Implementation of Responsive Multithreaded Processor

Tsutomu Itou ,† Masato Uchiyama ,†† Junichi Satou ,†††
Hiroyuki Usui ,† Katsuhiko Matsuura †
and Nobuyuki Yamasaki †

This paper describes about *RMT Processing Unit*(*RMT PU*), which is the processing unit of *Responsive MultiThreaded (RMT) Processor* for distributed real-time processing. *RMT PU* is the 8-way fine-grained Multithreaded architecture. It arbitrates computation resources with using priority and executes the higher priority threads first. In addition, *RMT PU* has the dedicated context cache and performs context switch with hardware, it reduces the overhead of context switch. Therefore, *RMT PU* supports real-time processing on the hardware level and is able to control real-time processing with the finer grained.

1. はじめに

リアルタイム性[1)]とは，処理の真偽が時間にも依存する性質を言う．狭義には，与えられた時間制約（デッドライン）を守ることを意味する．リアルタイム性にはその時間制約により大きくハードリアルタイム性とソフトリアルタイム性に分かれる．ハードリアルタイム性とは，与えられた時間内に必ず処理が完了しなければならない性質であり，時間内に処理が完了しない場合，価値がただちに 0 になる性質である．一方ソフトリアルタイム性とは，与えられた時間内に処理が完了しなくても価値がただちに 0 にはならないが，結果の品質が時間経過と共に低下する性質である．

リアルタイムシステムにおけるタスクの多くは周期的で，ハードリアルタイム性を持つタスクは周期の時間粒度が 100us から 10ms 程度と短く，CPU の処理時間も短かい．ソフトリアルタイム性を持つタスクの周期は時間粒度が 10ms から 1s 程度と比較的大きい．リアルタイムシステムでは，これらのタスクの時間制約を守るために，スケジューラが各タスクの周期や時間制約の条件によりスケジューリングを行い，各タスクに優先度を付与する．タスクは，ある一定時間間隔毎に優先度の高い順にプロセッサ資源を与えられて実行される．この時，実行するタスクを切り替えるためにコンテキストスイッチが生じる．コンテキストスイッチは現在実行しているタスクのコンテキスト（レジスタセット，プログラムカウンタ，ステータスレジスタ等）をメモリに退避し，次に実行するタスクのコンテキストをプロセッサ内に復帰しなければならないため大きなオーバヘッドとなる．リアルタイムシステムでは，多くのタスクがスケジューリングされて実行されるため，このオーバヘッドは大きな問題となる．そこで本研究ではハードウェアレベルでリアルタイム処理

† 慶應義塾大学
Keio University
†† 東芝
Toshiba
††† NEC

を支援するためのプロセッサを設計・実装する．細粒度マルチスレッドアーキテクチャによりコンテキストスイッチのオーバヘッドを削減し，ハードウェアでタスクの優先度を用いて実行順序を制御することにより，ソフトウェアのスケジューラがより時間粒度の短かい周期でタスクのスケジューリングを行うことを可能とする．

2. 関連研究

細粒度マルチスレッディングは CPU 内に複数のコンテキストを保持し，これらを高速に切り替えながら実行することにより，レイテンシの長い命令が実行された場合に，別のスレッドを実行することによりこのレイテンシを隠蔽し，全体としてのスループットを向上している．

また，Simultaneous MultiThreading（SMT）[2),3)] は細粒度マルチスレッディングとスーパスカラを合わせた特徴を持ち，1 クロックサイクルで複数のスレッドから命令を発行することができる．別々のスレッドから命令を発行することにより，より依存性の小さい命令を発行することができるため，Instruction Level Parallelism（ILP）が向上し，プロセッサ全体のスループットをさらに向上させることができる．リアルタイムシステムでは多くのスレッドを切り替えながら実行するため，複数のスレッドを並列に実行することができるマルチスレッディングアーキテクチャは有効である．

3. Responsive Multithreaded Processor

Responsive MultiThreaded（*RMT*）*Processor*はリアルタイム処理をハードウェアレベルで支援する *RMT Processing Unit*（*RMT PU*）をプロセッシングコアに持ち，リアルタイム通信機構（レスポンシブリンク），コンピュータ用周辺機能（PCI64 I/F，USB2.0 I/F，IEEE1394 I/F，シリアル I/F，DDR SDRAM I/F，DMA コントローラ），各種周辺制御機能（PWM ジェネレータ，パルスカウンタ，クロック制御ユニット）を 1 つのチップに集積した System on a Chip（SoC）である．

図 1 に *RMT Processor* のブロック図を示す．*RMT PU* は，256bit のバスを介して DDR SDRAM I/F と接続している．バンド幅の広いバスを用いてプロセッシングコアとメインメモリを接続することにより，命令フェッチや後に述べるベクトル演算において，メモリアクセスのスループットを改善している．

レスポンシブリンク（Responsive Link），PCI I/F，USB I/F，IEEE1394 I/F，シリアル I/F（UART），DMA コントローラ（DMAC），PWM ジェネレータとパルスカウンタ（PWM/PC），クロック制御ユニット（CLK CTRL），外部バス I/F（EXT I/F）は 32bit バスに接続されている．32bit バスと 256bit バスはゲートウェイ（GW）を介して接続されている．それぞれのバスを流れるデータはゲートウェイにおいてバスサイジングが行われ，もう片方のバスに送られる．また，レスポンシブリンク [4)] のイベントリンクは *RMT PU* のメモリアクセスユニットに直接接続され，プロセッシングコアからはバスを介さず，制御レジスタの一部としてアクセスすることができる．これにより，高速にイベントリンクにアクセスすることが可能である．

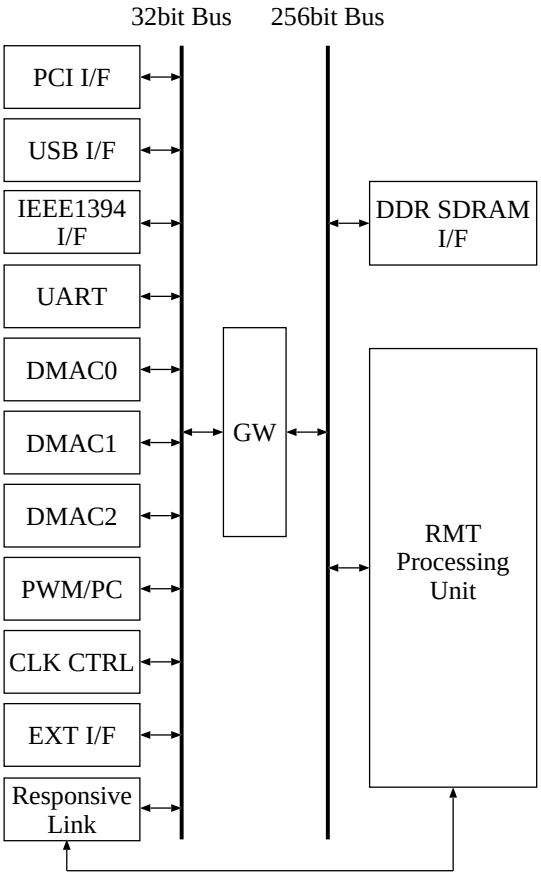

図 1 *RMT Processor* のブロック図
Fig. 1 Block diagram of *RMT Processor*.

3.1 Responsive Multithreaded Processing Unit

RMT PU は 8way の細粒度マルチスレッディングに優先度を用いた制御を行うことにより，ハードウェアで様々なレベルのリアルタイム処理を支援する．マルチスレッドアーキテクチャでは複数のスレッドが並列

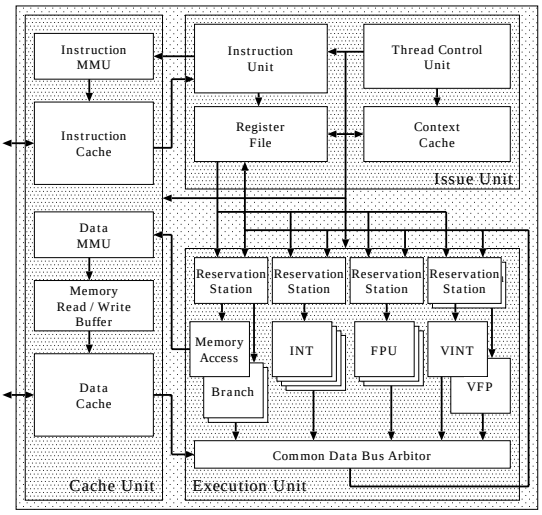

図 2　*RMT PU* のブロック図
Fig. 2　Block diagram of *RMT PU*.

に実行されるため，スレッド間で演算器やキャッシュシステム等の計算資源の競合が起こる．競合が起こった場合，*RMT PU* はスレッド毎に設定された優先度を基に，優先度のより高い命令に対して先に計算資源を割り当てる．これにより並列に実行しているスレッドの中で，優先度の高いスレッドから優先的に実行する．

図 2 に *RMT PU* のブロック図を示す．*RMT PU* は命令発行ユニット（Issue Unit），命令演算ユニット（Execution Unit），キャッシュユニット（Cache Unit）の大きく 3 つに分かれる．命令発行ユニットは各スレッドの実行を制御し，優先度に従って命令演算ユニットに対して各スレッドの命令を送る．命令演算ユニットは命令発行ユニットから送られてきた命令を演算する．キャッシュユニットは命令発行ユニットからの命令フェッチ要求，命令演算ユニットからのデータアクセス要求を処理する．

3.2　命令発行ユニット

命令発行ユニットの役割は各スレッドの実行を制御し，命令演算ユニットに対して命令を発行することである．**表 1** に命令発行ユニットの概要を示す．

アクティブスレッドとはプロセッサ内に保持されているスレッドで，すぐに実行を開始することができる．キャッシュスレッドとは後で述べるコンテキストキャッシュ内に保持されているスレッドを示す．優先度は 8bit を用いて 256level[5] で表し，値が大きいほど優先度は高くなる．

各スレッドの制御はスレッド制御ユニットで行う．アクティブスレッドはスレッド制御ユニット内にあるスレッドテーブルによって管理される．スレッドテーブルのフォーマットを**図 3** に示す．ENABLE フィールドはアクティブスレッドが有効であるかどうかを示す．STATE フィールドはアクティブスレッドの状態を示し，実行中，停止中，後述するコンテキストキャッシュへの退避中等といった状態を示す．KEEP フィールドはアクティブスレッドをプロセッサ内に保持しつづけるかどうかを示す．PRIORITY フィールドはスレッドの優先度を示し，この値が *RMT PU* 全体で使用される．

表 1　命令発行ユニットの概要
Table 1　The outline of the instruction issue unit.

アクティブスレッド数	8
キャッシュスレッド数	32
優先度の指定	256level
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32bit × 32entry × 8set
整数リネームレジスタ数	32bit × 64entry
浮動小数点レジスタ数	64bit × 8entry × 8set
浮動小数点リネームレジスタ数	64bit × 64entry

13	12:9	8	7:0
ENABLE	STATE	KEEP	PRIORITY

図 3　スレッドテーブルのフォーマット
Fig. 3　The format of the thread table.

RMT PU ではスレッドの生成，削除，実行，停止，優先度の設定等のために新たに命令を追加した．スレッド制御ユニットはこれらの命令が発行されると，命令に応じてスレッドテーブルを書き換え，アクティブスレッドの制御を行う．

先に述べた通り，*RMT PU* では 8 つのコンテキストをプロセッサ内に保持して実行することができる．しかしそれ以上のスレッドを実行する場合，コンテキストスイッチが発生する．コンテキストスイッチは現在実行しているスレッドのコンテキストをメモリに退避し，新しく実行するスレッドのコンテキストをメモリから復帰しなければならないため，オーバヘッドが大きくなる．

RMT PU ではコンテキストを格納するための専用キャッシュをオンチップに用意し，レジスタファイルとの間を広いバス（GPR:256bit，FPR:128bit）で接続している．コンテキストキャッシュは 32 個のコンテキストを格納することができ，コンテキストスイッチをハードウェアにより 4 クロックサイクルで行う．これによりコンテキストスイッチにかかるオーバヘッドを大幅に削減する．

表 2　命令演算ユニットの概要
Table 2　The outline of the instruction execution unit.

整数演算器	4 + 1（Divider）
浮動小数点演算器	2 + 1（Divider）
64bit 整数演算器	1
整数ベクトル演算器	1（8IU × 2 line）
浮動小数点ベクトル演算器	1（4FPU × 2 line）
分岐ユニット	2
メモリアクセスユニット	1
同期ユニット	1

アクティブスレッドのコンテキストキャッシュへの退避，キャッシュスレッドのプロセッサ内への復帰，アクティブスレッドとキャッシュスレッドの入れ替えは新たに追加した命令により，スレッド制御ユニットが行う．スレッド制御ユニットはスレッドの退避命令や復帰命令，入れ替え命令を受け取ると，内部に保持しているキャッシュスレッドのテーブルを検索し，コンテキストキャッシュをアクセスするためのアドレスを生成して，コンテキストキャッシュをアクセスする．

命令発行ユニットは命令キャッシュアクセスと命令演算ユニットへの命令発行スロットで，優先度を用いた調停を行う．

3.3　命令演算ユニット

命令演算ユニットの役割は命令発行ユニットから送られてくる命令を演算することである．**表 2** に命令演算ユニットの概要を示す．

*RMT PU*はリザベーションステーションとリオーダバッファを用いて，アウトオブオーダ実行を行う．*RMT PU*では複数のスレッドが並列に実行されているため，各演算器においてスレッド間で競合が起こる．命令演算ユニットではリザベーションステーションにおいて，優先度による制御を行う．リザベーションステーションでは演算に必要なオペランドがそろうまで命令は保持される．演算に必要なオペランドがそろい命令の実行が可能になると，各演算器に対して命令が発行される．*RMT PU*では複数の命令が実行可能になった場合，リザベーションステーションは，各命令の優先度を調べ，優先度の高い命令から先に演算器に発行する．これにより優先度の高いスレッドの命令に対して，先に演算器を割り当てる．

一方，マルチメディア処理のようなソフトリアルタイム処理では多くのデータを繰り返し演算しなければならないため，高い演算性能が要求される．このような処理ではデータの並列性を利用して演算性能を高めることができる．

*RMT PU*ではベクトル演算機構を用いている．ベクトル演算により，少ない命令スロットを有効に活用し，ソフトリアルタイム処理に要求される高い演算性能を実現する．また，ベクトル演算を行うスレッドの数やプログラムによって必要とされるベクトルレジスタの構成は異なってくる．そこで *RMT PU*では整数，浮動小数点共に 512 セットあるベクトルレジスタを，ベクトル長やレジスタの個数等の構成を動的に変更してスレッド間で共有することにより，複数のスレッドで柔軟なベクトル演算を可能としている．

表 3　キャッシュユニットの概要
Table 3　The outline of the cache unit.

TLB エントリ（命令，データ）	64 entry
キャッシュサイズ（命令，データ）	32K byte
victim cache（命令，データ）	512 byte

ベクトル演算は整数演算，浮動小数点演算共に 2 つの演算パイプラインが並列に動作することにより，複数のスレッドで並列してベクトル演算を行うことができる．各演算パイプラインは，整数演算パイプラインで 8 個，浮動小数点演算パイプラインで 4 個の演算器を持つことにより，複数のベクトル要素を並列に演算する．また，プログラマが複合演算を定義し，定義した命令を 1 命令で実行することにより，ベクトル演算器の使用率を向上させ，ベクトル演算の性能を向上させている．

3.4　キャッシュユニット

キャッシュユニットの役割は命令発行ユニットから送られてくる命令フェッチ要求と，命令実行ユニットから送られてくるデータアクセス要求を処理することである．**表 3** にキャッシュユニットの概要を示す．

キャッシュユニットは MMU（Memory Management Unit）を持ち，ハードウェアでアドレス変換を行うため，各スレッドは仮想アドレスを用いてプログラミングを行うことができる．

MMU が置かれる場所により，仮想アドレスでキャッシュをアクセスするか物理アドレスでキャッシュをアクセスするかが決まる．図 2 に示した通り，*RMT PU* では MMU はキャッシュよりも前に置かれ，キャッシュアクセスを行う前にアドレス変換を行う．よってキャッシュは物理アドレスを用いてアクセスされる．キャッシュアクセスの前にアドレス変換を行うため，キャッシュアクセスにかかるレイテンシが増加するが，実行するスレッドがコンテキストスイッチにより切り替わった場合でもキャッシュをフラッシュする必要がなくなる．また，複数のスレッドでメモリ領域を共有する場合，仮想アドレスでキャッシュをアクセスすると同一の物理メモリのデータが複数キャッシュされる問題（synonym）

が起こるが，物理アドレスを用いてキャッシュをアクセスすることによりその問題を回避することができる．

MMU における TLB エントリには仮想ページ番号，物理ページ番号の他に，複数スレッドで共有するための共有情報，コンテキストグループ番号を指定する．*RMT PU* は最大 8 つのスレッドが動作するため，TLB エントリのミス率が高くなることが考えられる．共有情報を用いることにより，複数のスレッドで TLB エントリを共有し，使用する TLB のエントリ数を削減することができる．

共有情報を設定した後に，新しいスレッドを共有情報に追加する場合はコンテキストグループ番号を用いる．TLB を設定する場合，コンテキストグループ番号を指定することにより，コンテキストグループ番号の一致するエントリの共有情報に自身のスレッドを追加する．これにより，使用する TLB のエントリ数を増やすことなく TLB を有効化することができる．

キャッシュは命令キャッシュ，データキャッシュ共に 8way の set-associative 方式，ブロックサイズは 32byte で，キャッシュアクセスはパイプライン化されている．キャッシュミスが起こった場合，入れ換えるブロックの選択方法は LRU と優先度がある．優先度を基に入れ換えるブロックを選択する場合，より優先度の低いスレッドが使用しているブロックから先にキャッシュを追い出される．これにより，優先度の高いスレッドのキャッシュブロックが追い出されることを防ぐ．

victim cache[6] は，キャッシュブロックの入れ換えに伴ないキャッシュを追い出されたブロックを，full associative 方式で保持する．キャッシュミスを起した場合，victim cache にデータが残っていれば，そのブロックをキャッシュに戻すことにより．キャッシュミスによる内部バスへの要求を減らし，メモリアクセスの遅延を減少させる．

キャッスミス等によりバスを介して下位メモリをアクセスする場合にも優先度を用いた制御を行う．メモリアクセスはキャッシュよりも低速なため，待ち行列が発生する．この場合，より優先度の高いスレッドからバスを使用して下位メモリにアクセスする．

4. 評　　価

RMT Processor は現在，実機による評価環境が整っていないため，評価は実チップの設計・実装に用いた RTL によるシミュレーションによって行った．

単純ソートを行うプログラムを用いて優先度による制御の評価を行った．並列に実行するスレッドの数を 1 スレッドから 8 スレッドまで変えた時に，優先度を用いた場合と用いない場合について，処理にかかる時間を測定した．優先度を用いた場合，Thread0 の優先度が最も高く，段階的に優先度が低くなり，Thread7 の優先度が最も低くなるように設定した．

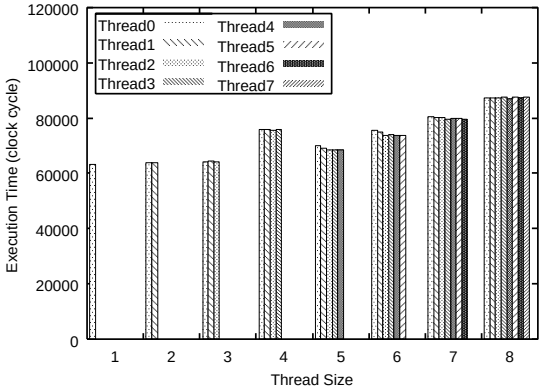

図 4　優先度を用いない場合の実行結果
Fig. 4　Execution time without using priority.

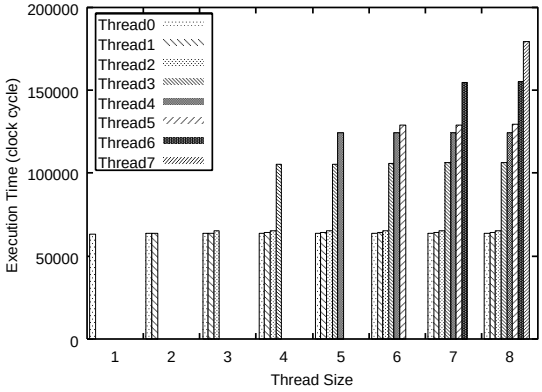

図 5　優先度を用いた場合の実行結果
Fig. 5　Execution time with using priority.

図 4 に優先度を用いない場合の実行結果を示す．8 スレッド並列に実行した場合，1 スレッド単独で実行した場合に比べて 37.7%実行時間が増加した．これは計算資源がラウンドロビンで割り当てられるため，スレッド数が増えるに従って各計算資源で競合が増えたためと考えられる．しかし順番に計算資源が割り当てられていくため，全てのスレッドが同じ実行時間で終了した．

図 5 に優先度を用いた場合の実行結果を示す．8 スレッド並列に実行した場合，優先度が最も高いスレッドは 1 スレッド単独で実行した場合に比べて，0.4%実行時間が増加した．また，優先度が最も低いスレッドは，1 スレッド単独で実行した場合に比べて 182.7%実行時間が増加した．これは優先度による調停により，優先

度の高いスレッドから順に処理が行われ，優先度の低いスレッドは優先度の高いスレッドが終了するまで処理が待たされたためと考えられる．よって，ハードウェアで優先度を用いて計算資源の競合を調停することにより，ソフトウェアでコンテキストスイッチを行いながら実行する通常のリアルタイム実行と同様の効果を得ることができたと言える．このとき，*RMT PU*ではコンテキストスイッチによるオーバヘッドはないため，ソフトウェアのスケジューラが適切にタスクのスケジューリングを行い，各スレッドに優先度を付加し，*RMT PU*で優先度を用いて実行することにより，スケジューリングの時間粒度を小さくすることができる．また，優先度の高いスレッドがメモリアクセスなどの長いレイテンシの命令を実行している場合は，細粒度マルチスレッディングにより，空いている計算資源を優先度の低いスレッドが使用するため，優先度の高いスレッドの実行を妨げることなく全体のスループットを向上することができる．

5. 結　　論

本研究では *Responsive MultiThreaded*（*RMT*）*Processor*のプロセッシングユニットである *RMT Processing Unit*（*RMT PU*）の設計・実装について述べた．

*RMT PU*は細粒度マルチスレッディングに，優先度を用いて計算資源の競合を調停することにより，優先度の高いスレッドから優先的に実行し，ソフトウェアでコンテキストスイッチを行いながら優先度の高いスレッドから実行する，通常のリアルタイム実行と同様の効果を得ることができる．ソフトウェアのスケジューラはこの効果を利用することにより，より短かい時間粒度でスケジューリングを行うことが可能となる．

*RMT Processor*は現在実チップを用いた評価環境を構築中であり，今後は実機を用いて評価を行っていく予定である．

参 考 文 献

1) Stankovic, J. A.: Misconceptions about real-time computing: a serious problem for next-generation systems, *IEEE Computer*, Vol. 21, pp. 10–19 (1988).
2) Eggers, S. J., Emer, J. S., Henry M, L., Lo, J. K., Stamm, R. L. and Dean M, T.: Simultaneous multithreading : A platform for next-generation processors., *IEEE Micro*, Vol. 17, No. 5, pp. 12–19 (1997).
3) Tullsen, D. M., Eggers, S. J., Emer, J. S., Henry M, L., Lo, J. K. and Stamm, R. L.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *Proceedings of the 23rd Annual International Symposium on Computer Architecture* (1996).
4) 山崎信行, 松井俊浩: 並列分散リアルタイム制御用レスポンシブプロセッサ, 日本ロボット学会誌, Vol. 19, No. 3, pp. 68–77 (2001).
5) W.S.Liu, J.: *REAL-TIME SYSTEMS*, Prentice Hall (2000).
6) Jouppi, N. P.: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, *Proc. 17th Annual Int'l Symposium on Computer Architecture*, pp. 364–373 (1990).