実時間制御を支援する組み込み用 RISC コア ライブラリの開発

Development of Embedded RISC Core Library for Supporting Real-Time Processing

田中　清史
Kiyofumi TANAKA

北陸先端科学技術大学院大学　情報科学研究科 情報システム学専攻　（〒923-1292 石川県能美郡辰口町旭台１－１　E-mail: kiyofumi@jaist.ac.jp ）

ABSTRACT. In this project, I introduce the processor architecture that enables real-time processing in embedded systems, and develop the new processor core that implements the architecture. The core is configurable about the hardware elements, for example the size of a cache or the bit width of memory bus, according to demands on speed, size of hardware and consumption of electricity. In addition, the core has fast context-switching mechanisms with multiple contexts for trap processing and explicit cache-control instructions for enabling data prefetching and fast DMA. The core aims to support research and development in embedded systems by opening the source codes written in VHDL.

１．背景

近年 PLD/FPGA や ASIC の集積度が向上し、組み込みシステムにおいて制御用の CPU コアとその周辺回路を同一チップに搭載することが可能となってきた。同一チップに埋め込むことは、処理速度が大きく向上し、電力が小さく抑えられ、かつシステムの実装面積を小さくすることが可能であることを意味する。

組み込みシステムにおいてチップを設計／開発する際に、周辺回路を独自設計し、その制御回路は既存の CPU チップあるいは CPU コアを使用することにより、開発期間を短縮することが可能である。制御用コントローラとして IP ベンダが提供する CPU コアを使用する選択肢があるが、権利やソースコード改変の問題、あるいはインターフェースの適合性の面から問題があり、組み込みシステムの設計において大きな障害となる。そのため、制御回路として CPU コアを組み込んだハードウェアを設計する場合は、柔軟で再利用可能な CPU コアがフリーなオープンソースコードとして使用可能であることが求められている。CPU コアをはじめとし、回路要素の充実した記述ライブラリを構築することにより、組み込みシステムの開発がソフトウェア開発と同様に、ライブラリからコンポーネントを選択し、それらを組み合わせることにより達成される。

２．目的

本プロジェクトでは組み込みシステムに適した柔軟性のある CPU コアを新規に開発し、オープンソースコードとして研究／開発を支援することを目的としている。組み込みシステムにおけるリアルタイム処理を可能とするプロセッサアーキテクチャを提案し、それを実現するプロセッサコアを開発する。パイプライン動作する従来の汎用 RISC プロセッサの方式に加え、割込みに対する高速な応答を可能とするための割込み専用レジスタセットを搭載し、割込み発生時にハードウェアによって高速にコンテクストを切り替える。この方式により、コンテクストの切り替え時に従来のようなメモリに対するロードおよびストア命令によるレジスタ群の退避・復帰の必要性を除去可能である。このような割込み専用レジスタセットを複数用意することにより、多重の割込みに対して、優先度に従って高速な応答を保証する。また、組み込みシステムにおける低速なメモリの使用に起因するメモリアクセスのオーバヘッドを削減するために、安価でかつ高機能なキャッシュメモリモデルを実現する。さらに、命令セットに既存の RISC アーキテクチャのものを採用することにより、これらの機能が容易に利用可能となり、システムプログム開発の期間を短縮することを目的としている。組み込みシステムの用途、あるいは処理能力に対する要求に従って、プロセッサ内の各要素のサイズ／有無を選択可能とすることにより、多目的組み込み用 RISC コアライブラリとして研究開発する。

以上の方針にしたがい、RISC コアをハードウェア記述言語である VHDL で記述することにより、多くの EDA ベンダの設計 CAD システムでの適用が可能となる。また VHDL の使用により、特定のターゲットライブラリ／チップに限定されず、あらゆる FPGA や ASIC ライブラリへの適用が可能である。本 RISC コアは、オープンソースコードの位置付けをとることにより、組み込みシステムにおいて近年注目されている再構成可能な FPGA や ASIC 技法によるシステムオンチップ（SOC）の開発にソースコードレベルで容易に適用可能となる。

３．設計の基本方針

組み込み用 CPU コアの満たすべき条件として、以下のものが挙げられる。

・十分な計算処理能力

・割込みに対する高速な応答
・プログラミング環境の提供
・柔軟なインターフェース
・低消費電力

様々な高速化技法を有する市販の汎用高性能プロセッサは必要なハードウェア量あるいは消費電力の面で組み込みシステムには適していない。また周辺回路の制御用として使用する場合、浮動小数点数の演算ユニットを搭載する必要はない（ただし、アプリケーションによっては必要となるので、FPU のインターフェースを持つことが望ましい）。そこで本 CPU コアは整数命令を基本としている。

組み込み用途で CPU コアを使用する場合、外部のイベント要求に対する処理速度が発生するオーバヘッドの大きな原因となっている。このことから、割込みに対する応答速度がリアルタイム処理の効率を決定づける。本コアは割込み発生時のオーバヘッドを小さくすることを第一とする。

CPU コアの設計で新規の命令セットを採用した場合、プログラムを開発する際に少なくともアセンブラを開発する必要がある。これに対し、既存の命令セットを採用することにより、既存のアセンブラ／コンパイラが利用でき、またコードの再利用性の面からプログラマの負担が小さくなる。特にコンパイラの利用はデバッグ効率の面からプログラム開発の期間を短縮する。本コアでは、命令セットとして SPARC Architecture Version 8[1] の整数命令群を使用する。ただし、制御用コントローラとして効率の良い処理を実現するために、割込み発生時の振舞いなど、内部的な仕様は独自のものである。

通常のプロセッサでは、割込み発生時にプロセッサコンテクストをソフトウェアで退避する必要がある。これは割込みに対する応答速度の面でボトルネックとなり、リアルタイム処理を実現するにあたって大きな障害となる。SPARC アーキテクチャではレジスタウィンドウによりスタックを構成し、割込み発生時にレジスタを退避する必要がないが、オーバーラップしているレジスタ群は変更することができず（あるいはその部分を退避する必要がある）、本来 32 本あるレジスタの使用に制限が加わる。このことは、割込みハンドラの記述の際に既存のコンパイラをそのまま使用することが事実上不可能であることを意味する。また、割込みを受理可能とするために、空きのウィンドウがあることを常に保証しながら使用しなければならない。

本 RISC コアでは割込みルーチン専用のレジスタセットを用意し、割込み発生時に自動的にアクティブなレジスタセットを切り替えることにより、ソフトウェアでレジスタを退避する必要を除去し、高速なコンテクスト切り替えを実現する。また、そのような割込み専用レジスタセットを複数用意することで、優先度ベースで多重の割込みに対応する。ただし本コアは SPARC 特有のレジスタウィンドウを持たないため、ウィンドウを明示的に回転する命令（save/restore）を使用しないことを前提としている。GCC[3] にはレジスタウィンドウを回さないコードを生成するためのオプション（-mflat）を持つバージョンがあり、レジスタウィンドウを持たないことはプログラム開発において支障はない。

安価な組み込みシステムを開発する際、メモリアクセスに関してスプリットフェーズバスのような高価なバス機構を搭載するのは困難であり、インターロック式のメモリバスが使用されることが多い。低速なメモリアクセスによるオーバヘッドを軽減するために、本コアでは多重発行可能なキャッシュプリフェッチ機構[4,5]を搭載する。この多重発行機能は、DMA や他からのメモリアクセスとの競合によってレイテンシコストが変動する場合の性能低下を最小限に抑える効果がある。また、外部のメモリにおいてデータの授受を行うアプリケーションに対応するために、外部メモリとキャッシュ間の一貫性を小規模なハードウェアで効率良く保証するための機構[5]を実現する。

４．アーキテクチャ

図 1 に本 RISC コアの全体構成図を示す。主な構成要素は、インテジャユニット(Integer Unit：IU)、命令キャッシュ（Instruction Cache)、データキャッシュ（Data Cache)、メモリアクセス制御レジスタ（Memory Access Control Register： MACR)、バスインターフェースユニット（Bus Interface Unit： BIU)、割込み要求コントローラ（Interrupt Request Controller：IRC)である。各要素を接続する命令バス（I-BUS）およびデータバス（DI/DO-BUS)は 64 ビット幅であり、IRC に対する外部からの割込み要求信号は 15 チャネルである。コンフィグレーションにより、命令およびデータキャッシュの有無／サイズ、あるいは BIU におけ

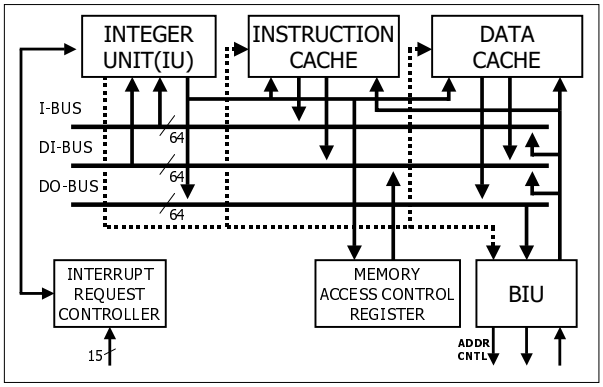

図 1　ブロック図（全体構成）

（1）インテジャユニット

図 2 にインテジャユニットの構成図を示す。主な構成要素として、各制御信号の出力生成および実行パイプラインの制御を行う命令ブロック(INSTRUCTION_BLOCK)、命令アドレスの出力生成を行うアドレスブロック(ADDRESS_BLOCK)、演算部である実行ブロック（EXECUTE_BLOCK）、およびプリフェッチ＆ストアバッファ（Prefetch/Store buffer）である。

命令セットとして SPARC Architecture Version 8[1]の整数命令セットを実行する。ただし、コンパイラによる生成が稀な一部の命令は演算器としてはサポートしない。これらの非サポート命令が実行プログラム中に出現した場合、unimplement instruction trap を発生させ、ハンドラにより命令をエミュレートする。実行に関して特権モードとユーザモードの 2 つのモードを切り替えて動作し、特権命令は特権モードでのみ実行可能であり、ユーザモードでこれを実行した場合は privileged instruction trap を発生する。

全ての命令はフェッチ(F)、デコード(D)、実行(E)、メモリアクセス(M)、レジスタ書き込み(WB)の 5 段パイプラインで実行される。従来の SPARC が持つレジスタウィンドウは無く、32 個の 32 ビットレジスタからなるレジスタセットを複数持ち、その数はコンフィグレーションによ

って、2 セット、4 セット、8 セットのいずれかとなる。ここで、SPARC における 8 つのグローバルレジスタは各レジスタセット毎にそれぞれ用意されている。またプリフェッチ＆ストアバッファの有無／機能（ストア命令のみ挿入、またはストア命令とプリフェッチ命令を挿入）が選択可能である。

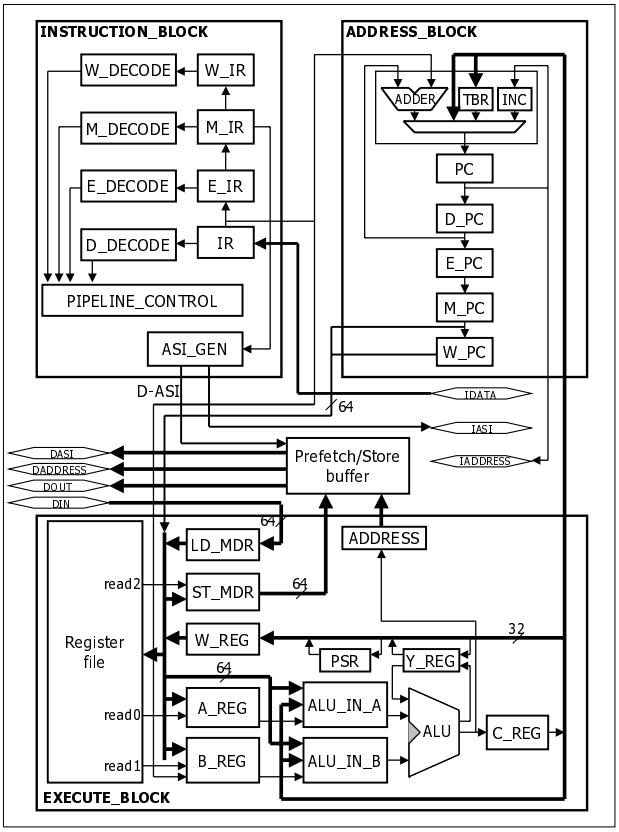

図 2　インテジャユニット

（2）命令／データキャッシュ

命令キャッシュとデータキャッシュを内蔵可能であり、サイズはそれぞれ、0K バイト、4K バイト、8K バイト、16K バイトのいずれかとなる。キャッシュのブロックサイズは 32 バイトである。4K バイトキャッシュの場合がダイレクトマップ方式、8K、16K バイトキャッシュの場合がそれぞれ 2 ウェイセットアソシアティブ、4 ウェイセットアソシアティブ方式であり、セットアソシアティブにおける置換方式は(擬似)LRU に従う。なお、データキャッシュはライトバック方式である。

命令、データキャッシュそれぞれ、割込みへの高速応答のための機構の一つとしてブロックのロック機能を持つ。ロックされたブロックはブロックの置換によって追い出されることはない。（ただし、セット内の全てのブロックがロックされた場合は、ある一つのブロックが追い出される。）ブロックをロックするためには、そのブロックに対応するタグ内のロックビットをセットするか、あるいはメモリアクセス制御レジスタ内の自動ロックビットをセットする。後者の場合、キャッシュに fill されるブロックは自動的にロックされることになる。

その他の機能として、データキャッシュに対する命令ベースの明示的制御機構があり、これについては 6 章で述べる。

（3）メモリアクセス制御レジスタ

メモリアクセス制御レジスタは 64 ビットのレジスタであり、命令、データキャッシュの ON/OFF、キャッシュ自動ロックの ON/OFF、SPARC の alternate space load/store の cacheability、ブートモード、およびプリフェッチ＆ストアバッファの ON/OFF を示すフィールドを持つ。さらに読み出し専用フィールドとして、キャッシュサイズやプリフェッチ＆ストアバッファなどのコンフィグレーション情報を持つ。

（4）バスインターフェースユニット

本コアはバスインターフェースユニットを介してインターロック式のメモリバスに接続される。メモリアクセス要求のデータサイズはロード、ストア命令による 1、2、4、8、およびキャッシュのブロック単位である 32 バイトである。メモリバスのビット幅はコンフィグレーションにより 8、16、32、64 ビットのいずれかをとる。

（5）割込み要求コントローラ

割込み要求コントローラは 15 チャネルの割込み要求入力信号を持ち、各入力信号について連続する 3 クロックサイクルの間アクティブな状態を検出した場合に割込みを受け付ける。

レジスタセットモード割り当てレジスタ（RSMAR）と割込み要求レベル割り当てレジスタ（IRLAR）を持つ。RSMAR は割込み要求の優先度とレジスタセットとの対応を指定するレジスタであり、その優先度を持つ割込み要求の処理は対応するレジスタセットで実行される。一方、IRLAR は外部の割込み要求入力信号と優先度の対応を指定するレジスタである。これらの 2 つのレジスタを設定することにより、システム内の物理的配線が固定された後に、割込み要求の優先度を柔軟に変更することが可能となる。

5．割込み高速応答機構

複数の割込み専用レジスタセットを内蔵し、割込み発生時にアクティブなレジスタセットをハードウェアにより自動的に切り替えて使用することにより、多重の割込みに対して高速な応答を可能にする。これはマルチコンテクストアーキテクチャ[6]を割込み処理のコンテクストに応用したものである。表 1 に 4 つのレジスタセットを内蔵した場合のそれぞれのレジスタセットモードの用途を示す。

表 1　レジスタセットモード

CRSP 値	モード	用途
0	Normal exec.	通常の実行時
1	Exception	内部トラップ(例外)ハンドラの実行時
2, 3	Interrupt	外部割込みハンドラの実行時

現在実行中のスレッドが使用するレジスタセットはプロセッサ状態レジスタ(PSR)内の“ CRSP (current register-set pointer) ”フィールドが指定する。以前の実行スレッドを“ PRSP (previous register-set pointer) ”フィールド

が指定する。CRSP の実体は 1 つであるが、PRSP はレジスタセット毎に独立して存在する。

図 3 にレジスタセット 3 を使用する割込みを処理中に、レジスタセット 2 を使用する優先度のより高い割込みを受けた場合のアクティブなレジスタセットの切り替えの様子を示す。図中の実線が前者の割込み処理時、点線が後者の割込み処理時を表している。後者の割込みを受けたとき、CRSP の値は 3 から 2 に変化し、レジスタセット 2 に対応する PRSP に 3 がセットされる。後者の割込み処理が終了する際には、リターントラップ命令によって PRSP の値(3)が CRSP にセットされ、前者の割込み処理に復帰する。

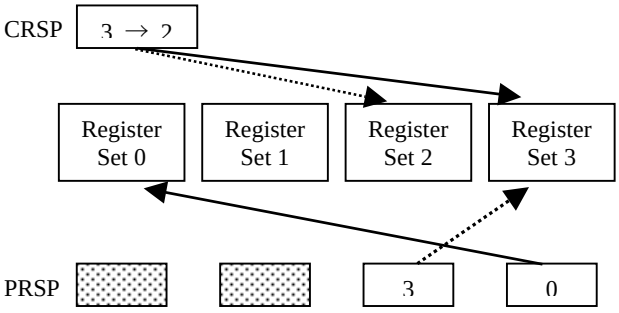

図 3　割込み発生時のレジスタセット切り替え

プロセッサ状態レジスタ(PSR)内の一部のフィールド、プログラムカウンタ(PC)、SPARC の Y レジスタなどはプロセッサコンテクストの一部として扱う必要があり、これらについてレジスタセットと同様に多重化し、CRSP の値にしたがって切り替えて使用する。このような切り替え機構により、同一レジスタセットを使用する割込みを多重に受け付けない方針で使用する限り、コンテクストを退避する必要はない。同一レジスタセットを使用する割込みを多重に処理する場合は、退避／復帰のコンベンションで使用する。

６．データキャッシュの明示的制御機構

低コストであることが重要な組み込みシステムにおいて、複雑な回路を必要とするスプリットフェーズ式のメモリバスは一般的には使用されず、インターロック式のバスが使用される。メモリバスの低スループットに起因するオーバヘッドを緩和するために、本コアはキャッシュへのプリフェッチ機構として、elastic prefetch[4,5]を実現する。このプリフェッチ機構は外部メモリのアクセス競合により引き起こされるレイテンシの変動に耐性を持たせることが可能であるという特徴を持つ。

さらに、小規模のハードウェアによりデータキャッシュと外部メモリの間の一貫性を維持するデータキャッシュブロック強制 fill/update 機構を持つ。この機構により、外部メモリを介して周辺回路とのデータの授受を行うアプリケーションの高速実行を可能とする。

（1）データキャッシュ多重プリフェッチ機構

データキャッシュのプリフェッチ機構を持つ高性能マイクロプロセッサは、ロックアップフリーキャッシュとスプリットフェーズバスの使用により、複数のプリフェッチ要求を多重に発行することが可能である。しかしロックアップフリーキャッシュやスプリットフェーズバスは高コストの要因となるため、組み込みシステムに適用するのは困難である。そこでこれらの代わりにプロセッサ内に FIFO バッファを用意することにより、実行パイプラインをストールさせることなしに多重プリフェッチを可能とする elastic prefetch を実現する。（本コアではハードウェアサイズと複雑さを考慮し、FIFO バッファをプリフェッチ要求とストア要求で共用する。ただしコンフィグレーションによりストア要求のみに使用することが可能である。）FIFO バッファ内の先頭のプリフェッチ要求により、プリフェッチ命令によって指定されたメモリアドレスを含むブロックがキャッシュ内に存在しない場合は、外部メモリ内の該当ブロックがキャッシュ内に fill される

ここで、例として 16 個の整数の総和を求める以下のコードシーケンスを扱う。ここで、キャッシュブロックのサイズを 16 バイトとする（実際のブロックサイズは 32 バイトである）。

```
ld [%g1], %l0
ld [%g1+4], %l1
add %l0, %l1, %l0
ld [%g1+8], %l2
ld [%g1+12], %l3
add %l0, %l2, %l0
add %l0, %l3, %l0
ld [%g1+16], %l2
ld [%g1+20], %l3
add %l0, %l2, %l0
add %l0, %l3, %l0
    •
    •
    •
ld [%g1+120], %l2
ld [%g1+124], %l3
add %l0, %l2, %l0
add %l0, %l3, %l0
st %l0, [%o0]
```

多重プリフェッチ機構とともに実行する場合は、以下の 4 つの命令をコードシーケンスの先頭に付加する。

```
prefetch [%g1]
prefetch [%g1+16]
prefetch [%g1+32]
prefetch [%g1+48]
```

加算命令(add)とストア命令(st)の実行に 1 サイクル要し、プリフェッチ命令(prefetch)により FIFO バッファに要求を挿入するのに同じく 1 サイクルかかる。16 個の整数データは外部メモリ上で初期化されていると仮定する。また、16 バイトのブロックを外部メモリからデータキャッシュに取り込むのに 7 サイクル要するものとする。図 4 に実行に要するサイクル数と外部メモリバスの使用状況を示す。図の(a)のシーケンスはプリフェッチなしの実行であり、(b)のシーケンスは多重プリフェッチを伴う実行である。プリフェッチなしの実行ではシーケンスの実行が終了するまで 56 サイクルかかっている。一方、多重プリフェッチの実行は終了まで 39 サイクルとなっている。これはプリフェッチなしの実行がバスの使用に関して断片的になっているために 4 つのロード命令(ld)がストールしているのに

対し、多重プリフェッチ実行では最初のロード命令でプリフェッチによるキャッシュ fill が間に合っていないために 3 サイクルストールしているのみで、後続するその他のロード命令はストールなしで実行できているためである。

効的に使用することにより、レイテンシが変動する場合にその影響を最小におさえる方式である。

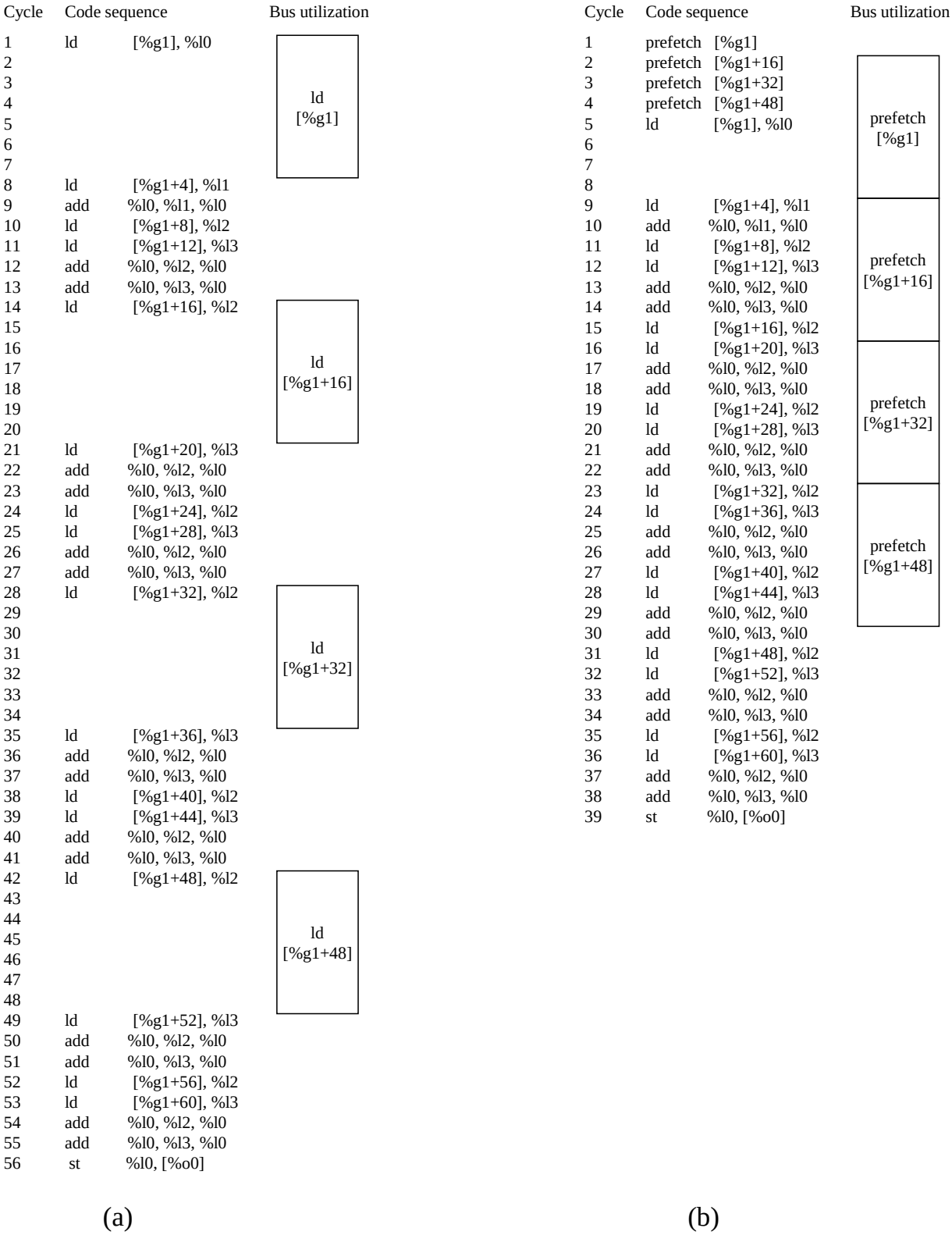

図 4　多重プリフェッチの効果。(a) プリフェッチなしの実行　(b) 多重プリフェッチを伴う実行

（2）データキャッシュブロック強制 fill/update 機構

周辺回路と制御プロセッサ間で DMA 機構により外部メ

モリを介してデータの移動を行う際、コヒーレントキャッシュを使用して効率良く行うことが有効であるが、組み込み用プロセッサではコスト削減のためにコヒーレントキャッシュを持たないことが一般的である。プロセッサがバイト単位やワード単位で外部メモリにデータを生成することが可能であるが、この場合高いパフォーマンスは期待できない。そこで本コアでは命令実行ベースでデータキャッシュと外部メモリとの一貫性を保証する方式[5]を実現する。この方式はキャッシュミス時のブロックの fill と、ブロック置換時の write-back という、キャッシュに本質的に備わる機能を命令実行によって起動するものであり、したがって複雑な制御回路を必要とせず低コストである。

データを周辺回路へ送る場合は、プロセッサ自身が DMA を起動する。したがって、DMA を起動する前にデータキャッシュ内の値で外部メモリを更新するのみでよい。これを実現するために、“ Store with Write-Back (STWB) ”命令を提供する。この命令の実行により、キャッシュ内に指定されたアドレスのブロックが含まれ、かつそのブロックが変更された状態(“ dirty ”)ならばメモリを更新する。

一方、周辺回路や DMA が外部メモリ内に生成したデータをプロセッサが受け取る場合は、周辺回路や DMA はデータが用意できたことをプロセッサに割込みにより通知する。続いて、プロセッサは更新されたメモリブロックをデータキャッシュに fill することになる。このとき、キャッシュのヒット／ミスヒットにかかわらず fill することが必要である。これを実現するために、“ Load with Explicit cache-Fill (LDEF) ”命令を提供する。この命令の実行により、指定されたアドレスのメモリブロックが強制的にキャッシュに fill される。

以上の STWB、LDEF により、効率の良いデータの授受が可能となる。さらに、組み込みシステムにおいてより高速な処理が必要な場合はマルチプロセッサを構成することが可能となる。

7．拡張命令

本コアは SPARC における実装依存の命令を利用することによりいくつかの機能を提供する。実装依存の命令の使用は、既存のコンパイラ／アセンブラを使用することが可能であることを意味する。

（1）IRSI: Inter-Register-Sets Instructions

任意のレジスタセットモードでの実行時に任意のレジスタセット内のレジスタをアクセス可能な命令であり、高速なシステムコールの実行を支援する。3 種類の命令があり、SPARC の load/store from/into alternate space 命令を使用している。

a) IRS-Load

ASI=0x80+*x* を指定した alternate space からのロード命令を使用する。ここで ASI は SPARC における address space identifier であり、*x* はデスティネーションのレジスタセットの番号であり、メモリからロードした値がその中のレジスタに格納される。メモリアドレスは CRSP で指定されるレジスタセット内のレジスタ値を用いて計算される。通常のロード命令がサポートする全てのデータサイズに対して有効である。

b) IRS-Store

ASI=0x80+*x* を指定した alternate space へのストア命令を使用する。*x* はソースのレジスタセットの番号であり、その中のレジスタの値がメモリへストアされる。メモリアドレスは CRSP で指定されるレジスタセット内のレジスタ値を用いて計算される。全てのデータサイズに対して有効である。

c) IRS-Add

ASI=0xC0+*x* を指定した alternate space からのロード命令を使用する。CRSP で指定されるレジスタセット内の 2 つのレジスタの加算値が *x* が示すデスティネーションのレジスタセットのレジスタへ格納される。（加算を行う 2 つのレジスタは、本来ロード命令のアドレス計算のためのレジスタ組であるが、この命令ではメモリアクセスは発生しない。）レジスタ組の一方に 0 番レジスタを指定した場合は、レジスタセット間のレジスタ値の移動命令となる。

（2）CLFI: Cache Line Forced Instructions

前章で示したデータキャッシュブロック強制 fill/update 機構を起動する命令を以下に挙げる。

a) FF-Load

LDEF を実現する命令であり、ASI=0x60—0x6F を指定した alternate space からのロード命令を使用する。この命令の実行により、指定されたアドレスを含む 32 バイトのメモリブロックが強制的にデータキャッシュに fill される。同時に、指定されたアドレスおよびサイズのデータがレジスタにロードされる。

b) FU-Store

STWB を実現する命令であり、ASI=0x60—0x6F を指定した alternate space へのストア命令を使用する。この命令の実行により、指定されたアドレスのブロックがデータキャッシュ内に存在し、かつ変更された状態（“ dirty ”）の場合のみに外部メモリが更新される。実行後のキャッシュ内の当該ブロックの状態は“ clean ”となる。なお、ストア命令としての書き込み値は無視される。

（3）DCPI: Data Cache Prefetch Instructions

データキャッシュへのプリフェッチ機構を拡張命令によって実現する。SPARC において 0 番レジスタの値はゼロに固定であり、0 番レジスタへの 4 バイト以下のサイズの cacheable なロード命令はキャッシュを除くプロセッサコンテクストに影響を及ぼさないため、これをプリフェッチ命令として使用する。また、他の拡張命令との組み合わせを可能とする。例えば、ロード先に 0 番レジスタを指定した FF-Load を発行することにより、後続する実際のロード命令に先行してデータをキャッシュに取り込むことが可能となる。

（4）BTI: Byte Twisting Instructions

エンディアンバイトオーダーを反転させる命令であり、load/store from/into alternate space 命令を使用する[1]。8 ビットの ASI 値において、下位から 5 ビット目を 1 にセットした値を指定した場合、この機能が働く。ロード命令の場合、メモリあるいはキャッシュから読み込んだ値についてエンディアンを反転させたものがレジスタに格納される。ストア命令の場合、ソースレジスタの値についてエンディアンを反転させたものがメモリあるいはキャッシュに格納

[1] SPARC では Version 9[2]より、特別な ASI を使用することによってリトルエンディアンでアクセスすることが可能となった。

される。他の拡張命令との組み合わせが可能であり、例えば IRS-Add と組み合わせた場合、加算の結果についてエンディアンを反転させたものがデスティネーションレジスタに格納される。

（5）PILI: Processor Interrupt Level Instructions

プロセッサの割込みレベルを調節することにより、割込みハンドラ間でのクリティカルセクションを作成することが可能である。SPARC Version 8 ではプロセッサ状態レジスタ(PSR)に対する書き込み命令(WRPSR)によってプロセッサ割込みレベル(PIL)を変更することが可能であるが、書き込み前に書き込み値を生成するために数命令の実行が必要となり、効率が良くない。本コアでは PSR 内の PIL フィールドを 1 サイクルで変更する命令が提供される。これによりクリティカルセクションの高速な制御が可能となる。

a) SAVE&WRITE_PIL

SPARC における ancillary sate registers(ASR)への書き込み命令である WRASR 命令を使用する。この命令の実行により、現在の PIL の値が専用の待避レジスタへ書き込まれ、同時に命令内の即値で指定された値が PIL フィールドに書き込まれる。待避レジスタはレジスタセットモード毎に独立して存在する。

b) RESTORE_PIL

ASR からの読み出し命令である RDASR 命令を使用する。この命令の実行により、実行レジスタセットモードに対応する待避レジスタ内の値で PIL が更新される。

８．必要ハードウェア量の見積もり

本コアのソースコードは VHDL で記述されている。このソースコードを論理合成した結果を示す。合成には Synopsys 社の Design Compiler (2000.05)を使用した。表 2 に合成結果から得られるゲート数を示す。この合成では、コンフィグレーションとしてレジスタセット数が 4、8 エントリのプリフェッチ＆ストアバッファ、64 ビット幅のメモリバスインターフェース、8K バイトの命令キャッシュ、8K バイトのデータキャッシュとした。表中の“ Total ”は、ソースにおける階層構造をフラットにして合成した場合のもので、したがって個々のゲート数の総和とは値が異なっている。

表 2　ゲート数

Entity	Gate count
INSTRUCTION_BLOCK	1,508
ADDRESSB_LOCK	1,513
EXECUTE_BLOCK	31,495
PREFETCH/STORE_BUFFER	9,180
INSTRUCTION_CACHE_CONTROL	2,696
DATA_CACHE_CONTROL	4,997
OTHERS	1,414
REGISTER_FILES	17,037
INST_CACHE TAG MEMORY	17,663
DATA_CACHE TAG MEMORY	18,035
INST_CACHE DATA MEMORY	51,360
DATA_CACHE DATA MEMORY	51,360
Total	208,258

９．命令セット

本コアのサポート命令を表 3 に示す。SPARC Version 8 の命令セットのうちハードウェアで実現したものと、拡張命令群をサポート命令としている。非サポート命令はハードウェアでは実装されてないものであり、プログラム中にこれらが出現した場合、unimplement trap を発生させ、ソフトウェアでエミュレートする。

表 3　命令セット

サポート命令		
LOAD/STORE	LDSB(A) LDSH(A) LD(A) STB(A) ST(A) LDSTUB(A)	LDUB(A) LDUH(A) LDD(A) STH(A) STD(A) SWAP(A)
LOGICAL	AND(CC) OR(CC) XOR(CC)	ANDN(CC) ORN(CC) XNOR(CC)
ARITHMETIC/SHIFT	ADD(CC) SUB(CC) UMUL(CC) UDIV(CC) SLL	ADDX(CC) SUBX(CC) SMUL(CC) SDIV(CC) SRL/A
CONTROL TRANSFER	BICC(all) CALL TICC(all)	 JMPL RETT
READ/WRITE REG.	RD/WRY RD/WRTBR	RD/WRPSR
OTHERS	SETHI UNIMP	NOP
SPECIAL INST. EXTENSION	IRSI SAVE&WRITE_PIL BTI	CLFI RESTORE_PIL DCPI
非サポート命令 (Emulated by trap routine)		
LOAD/STORE	LD(D)F LDFSR ST(D)F ST(D)C	LD(D)C LDCSR STFSR/DFQ STCSR/DCQ
ARITHMETIC	TADD/SUBCC(TV)	MULSCC
CONTROL TRANSFER	BFCC SAVE	BCCC RESTORE
READ/WRITE REG.	RD/WRASR	RD/WRWIM
OTHERS	STBAR Floating-point Instructions	FLUSH Coprocessor instructions

１０．まとめ

本プロジェクトにおいて、汎用 RISC プロセッサアーキテクチャを拡張して組み込み用途に適した RISC コアを新規開発した。本コアは基本的には SPARC 命令セットアーキテクチャを採用している。また、組み込みシステムの用途、ハードウェアサイズ、あるいは処理能力や消費電力に対する要求にしたがって、プロセッサ内の各要素のサイズ

／有無を選択可能としているため、多目的組み込み用RISC コアライブラリとして位置付けられる。

本コアの特徴として、複数のプロセッサコンテクストを内蔵することによる多重割込み要求に対する高速割込み応答機構と、データキャッシュプリフェッチや高速 DMA を可能とするデータキャッシュの明示的制御機構がある。また、容易なプログラミングや高速実行を支援するために、レジスタセット間命令、バイトオーダー反転命令、割込みレベル変更命令などを拡張命令によりサポートしている。全ての拡張命令は SPARC の実装依存命令を使用しているため、既存のコンパイラ／アセンブラが利用可能である。

VHDL で記述されたソースコードは、多くの EDA ベンダの設計 CAD システムにおいて、特定のターゲットライブラリ／チップに限定されず、あらゆる FPGA や ASIC ライブラリへの適用が可能である。本 RISC コアは、オープンソースコードの位置付けをとることにより、組み込みシステムにおいて近年注目されている再構成可能な FPGA や ASIC 技法によるシステムオンチップ（SOC）の開発にソースコードレベルで容易に適用可能となることを目的としている。

１１．参加企業及び機関

・株式会社三菱総合研究所

・三精システム株式会社

１２．参考文献

［1］SPARC International, Inc. : The SPARC Architecture Manual Version 8, Prentice-Hall, Inc. (1992)

［2］SPARC International, Inc. : The SPARC Architecture Manual Version 9, Prentice-Hall, Inc. (1994)

［3］http://gcc.gnu.org/

［4］特願平 11-354203（商用に使用する場合は、科学技術振興事業団に要問い合わせ）

［5］松本尚：高性能組み込み用プロセッサアーキテクチャの検討、電子情報通信学会技術研究報告、Vol.100、No.248、p17～24(2000)

［6］Wolf-Dietrich Weber and Anoop Gupta : Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results. In Proc. of the 16th ISCA, p273～280(1989)