組込みシステム開発支援オープンソースライブラリの開発

Development of Open-Source Libraries for Supporting Development of Embedded Systems

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 have developed hardware and software libraries for supporting development of real-time embedded systems. The libraries consist of a real-time RISC core with flexible components, peripheral circuits, basic routines, software libraries and a binary generator. All hardware descriptions were written in a standard hardware description language, VHDL, which makes it possible to apply them to most of commercial EDA tools. Moreover, programs running on the RISC core can be written in C language by using a GNU C compiler and linker, which makes it easy to develop firmware. The libraries are intended to be open-source free and support research and development of various embedded systems.

１．背景

近年 LSI の適用分野が組込みシステムにおいて広範囲に広がりつつある。組込みシステムにおいて ASIC 手法でチップを設計・開発する際に、アプリケーションの機能を実現する回路および周辺回路を独自設計し、その制御回路はベンダが提供する IP（知的所有権）などの既存の CPU コアあるいは CPU チップ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。この場合、IP に付随する権利やソースコード改変の問題、あるいは周辺回路とのインタフェースの適合性といったことが問題となる。制御用 CPU は ASIC の設計で最も共通化した構成要素であり、開発期間およびコストをおさえるためには、インタフェースが柔軟で、再利用可能な CPU コアがフリーかつオープンソースコードで利用可能であることが求められる。

多くの組込みシステムにおいて、機械／機器を制御するという意味合いから、汎用システム以上にリアルタイム性が求められる傾向にある。リアルタイム処理は基本ソフトウェア側のタスクスケジューリングに大きく依存するが、制御 CPU が他からの個々の要求へ高速なレスポンスを達成することにより、リアルタイム処理における制約時間に余裕を持たせることが可能となる。組込みシステムにおける制御 CPU は、機器／周辺回路からの割込みにより要求に応答するのが通常であり、リアルタイム処理の実現のためにはこの割込みに対する高速な応答機構が求められる。

提案者は平成１２年度未踏ソフトウェア創造事業において、上記の要求に応えるべく「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」のテーマのもとに RISC コアを開発した。この RISC コアは、キャッシュサイズやレジスタ数、外部メモリへのアクセス幅などを選択可能とすることにより、処理速度、消費電力、あるいは実装コストなど、ターゲットシステムの要求に合った構成を提供することが可能である。また、割込みに対する高速な応答を実現するためにマルチコンテクストアーキテクチャを導入し、割込み専用のプロセッサコンテクストを複数持つことが可能となっている。

２．目的

前年度に開発したこの RISC コアは、最終的にはフリーなオープンソースコードとすることで研究／開発を支援することを目的としている。コア全般を標準的なハードウェア記述言語である VHDL を使用して設計しており、多くのベンダの CAD ツールに適用可能であるが、実現回路に関して可読性という意味では完成度が低いのが現状である。すなわち、プロセッサ全体の記述ということからもファイル数およびサイズが大きいため、ファイルの階層構造の見直しおよび各ファイル内の実現回路／機能のコメントなどを充実させて初めて利用可能となる。本開発の第一の計画として、コアのオープン化に向けてソースコードの整備を行った。また、コアの仕様およびユーザマニュアル的なドキュメントを整備した。

新規に開発したプロセッサや DSP などの特殊なプロセッサを組込みシステムの制御プロセッサとして使用した場合、コンパイラが存在しないか、存在したとしても生成されたコードの実行効率が悪いことが多く、このような組込みシステムの開発ではアセンブラ言語によるソフトウェア開発が今なお行われる傾向がある。本 RISC コアは実行プログラムの開発の負担を抑えるため、命令セットに既存のプロセッサアーキテクチャである SPARC 命令セットを採用している。これにより GNU C Compiler（gcc）[2] などの既存のコンパイラあるいはアセンブラが利用可能である。しかし個々の処理プログラムは C 言語によって記述可能であるが、プロセッサの基本ルーチンであるリセットハンドラ、例外処理ルーチン、割込みテーブルなどが用意されておらず、またそれらをリンクするリンカも手作業によっていた。本開発における第二の計画としてこれら基本ルーチンおよびリンカの開発を行った。

本 RISC コアはリアルタイム処理のための機構としてマルチコンテクストアーキテクチャの他、データキャッシュに対する拡張命令ベースの制御機構を各種持つ。これらの機構を起動するためには、SPARC の実装既存命令である alternate space load/store 命令を発行する必要があるが、既存のコンパイラでは自動的にこの命令を含んだコードを

生成することはない。そのためこれらの機構を C 言語から利用可能とするために、拡張命令が埋め込まれたランタイムライブラリを作成する必要があった。また、その他組込みシステムで頻繁に用いられるルーチンである時計／日付機能、シリアル通信機能、DMA 機能を活用するライブラリなどを合わせて第三の計画として開発を行った。

近年組込みシステムの用途は携帯電話や家電製品に組込まれるような比較的低コスト、低パフォーマンスなものから、高速通信機器で必要とされる高いパフォーマンスのものまで多岐にわたる。組込み用 CPU は高性能汎用プロセッサの高速化技術を削減した構成のものが一般的であり、システム内の制御対象の数が多く、かつ時間制約の厳しいアプリケーションではリアルタイムの要求に応えるのが困難な場合がある。このような対象に対応するために、本コアではマルチプロセッサ構成を可能としている。具体的には命令セット内に同期不可分命令を含み、かつ命令制御でキャッシュコンシステンシを保つことが可能となっている。このようにマルチプロセッサのための基本的機能は備えているが、割込みに対する高速応答のためには、外部からの要求を受ける割込みコントローラが直接割込み処理を複数のプロセッサ間で割り当て／スケジューリングすることが望ましい。第四の開発項目として、このスケジューリング機能内蔵の割込み要求コントローラを開発した。

第五の開発項目として、多目的組込みシステム用ライブラリとして必要不可欠な、タイマー回路、DMA 回路およびシリアル通信（UART）回路の作成を行った。また、これらをコアから制御するドライバを併せて開発した。

３．RISC コアアーキテクチャ

図 1 に本 RISC コアの構成図を示す。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。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 における外部メモリバスのビット幅などが設定可能となっている。

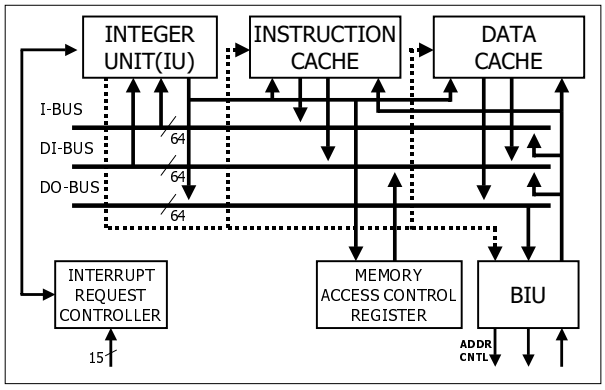

図 1　ブロック図（全体構成）

（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。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。この複数のレジスタセットにより、割込みに対する高速な応答を実現する。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。

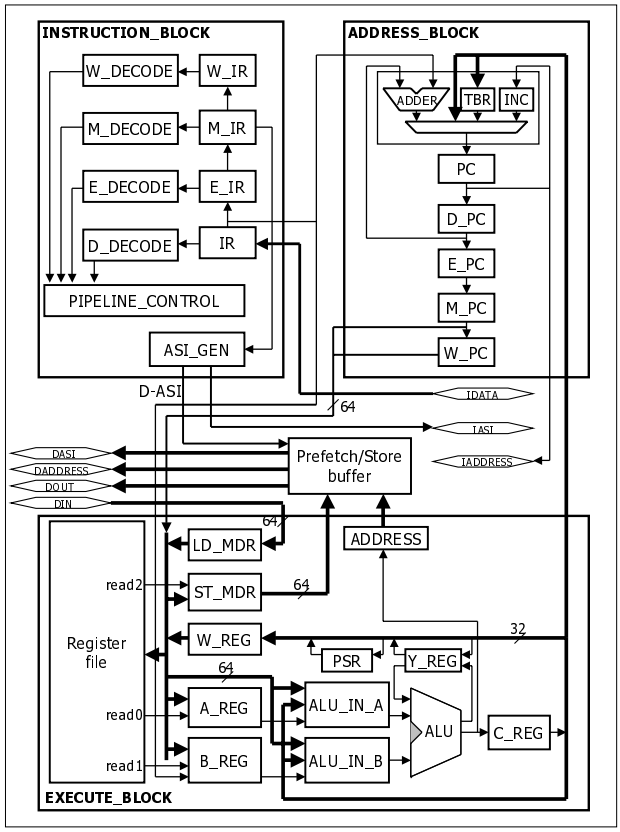

図 2　インテジャユニット

（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。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。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、データのプリフェッチおよび高速 DMA 転送を可能とする。

（3）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、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。

（4）メモリアクセス制御レジスタ

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、プリフェッチ＆ストアバッファの ON/OFF などを指定するフィールドを持つ。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。

（5）バスインタフェースユニット

本コアはバスインタフェースを介してインターロック式のメモリバスに接続される。バス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。

4．周辺回路

組込みシステムで頻繁に使用される周辺回路として、タイマ回路、シリアル通信回路、および DMA 回路をそれぞれ開発した。これらは全て VHDL で記述しており、必要に応じて取捨選択可能としている。

（1）タイマ回路

本ライブラリ回路は独立した 2 チャネルの 32 ビットタイマであり、2 チャネルは同一の機能を持つ。各チャネルともカウンタ入力クロックを選択可能であり、システムクロックの 2、4、8、16、32 倍の周期がある。設定によりフリーランカウンタあるいは周期カウンタとして機能し、各チャネル毎にコンペアマッチング割込み、オーバフロー割込みが独立に要求可能である。

（2）シリアル通信回路

本ライブラリ回路は、独立した 2 チャネルのシリアルコミュニケーションインタフェース(SCI)機能を備える。2 チャネルは同一の機能を持つ。SCI は調歩同期式通信でシリアル通信が可能である。独立した送信部と受信部を備えているため、全二重通信が可能である。また、送信部、受信部ともにダブルバッファ構造となっているため、シリアルデータの連続送信、連続受信が可能である。

（3）DMA 回路

本ライブラリ回路は、4 チャネルのダイレクトメモリアクセスコントローラ（DMAC）を内蔵する。DMAC は、DACK（転送要求受け付け信号）付きデバイス、外部メモリ、周辺モジュール（DMAC を除く）間のデータ転送を CPU に代わって高速に行うことができる。DMAC を使用することにより、CPU の負荷を軽減するとともにシステム全体の動作効率を向上させる。アドレス空間は 4G バイトであり、データ転送単位はバイト(8 ビット)、ハーフワード(16 ビット)、ワード(32 ビット)から選択可能である。

5．マルチプロセッサ支援機構

高速処理能力を必要とする組込みシステムに対応するために、本 RISC コアはマルチプロセッサ構成を可能とする。基本的には命令セット内に同期不可分命令を含み、かつ命令制御でキャッシュコンシステンシを保つ機構が存在することからマルチプロセッサが可能であるが、割込みに対する高速応答のためには、外部から要求を受ける割込みコントローラが直接割込み処理を複数のプロセッサへ分配／スケジューリングすることが望ましい。これを実現する割込み要求分配器（IRD：Interrupt Request Distributor）を開発した。

SPARC version 8 では外部の割込み要求信号線は 15 チャネルである。本プロセッサコアでは 2 つのレジスタ（RSMAR および IRLAR）により、割込み要求の物理線と割込みレベル、および使用するレジスタセットモードの対応を可変としている。基本的にはこれらによりチャネルと割込みレベルが対応付けられるが、マルチプロセッサ構成の場合は受理した割込み要求をどのプロセッサで実行するかが問題となる。この問題に対して各プロセッサの各割込みレベルに対応する、実行中およびペンディング中を表す状態レジスタを設け、この状態レジスタを用いて各プロセッサの負荷値を計算し、これにしたがって負荷分散を行う方式をとる。ただし、プロセッサコアの中心部分でこの状態レジスタを保持して IRD に対して出力することは、動作周波数に少なからず影響を及ぼす可能性が大きいため、状態レジスタは IRD 内部で保持し、プロセッサからこの状態をセット、リセットする方式を採用する。すなわち、プロセッサは割込み要求を受けると命令実行によって IRD 内の状態レジスタをセットし、割込み処理の終了時にリセットする。以上の方式により、ハードウェアとソフトウェア制御でプロセッサ割り当てを行う割込み要求分配器（IRD）を開発した。図 3 に IRD のブロック図を示す。

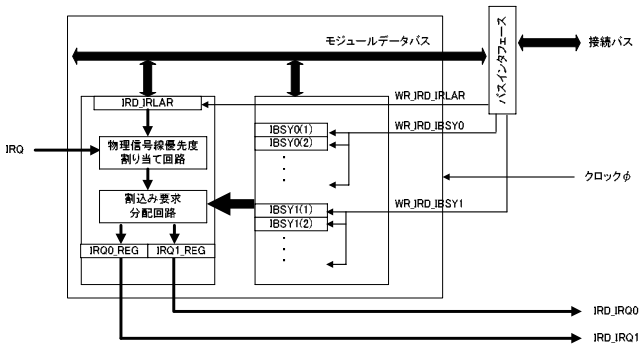

図 3　割込み要求分配器（IRD）のブロック図

6．プログラム開発支援

（1）基本ルーチン

平成 12 年度に開発した組込み用プロセッサコアは命令セットに SPARC アーキテクチャを採用しているため、GNU C Compiler [2]などの既存のコンパイラ、アセンブラが利用可能である利点があるが、プロセッサの内部構成は市販の SPARC プロセッサとは異なるものであり、プロセッサの利用者に対して基本的な操作方法を示しておくことが重要である。本開発ではプロセッサの初期化を行うリセットハンドラ、割込み発生時のジャンプ先を指定する割込みテーブルおよび基本例外処理ルーチンの例を作成した。リセットハンドラおよび例外処理ルーチンについては、コアの構成あるいはプログラムのメモリマップによって異なる処理／アドレスを選択可能とする必要があるため、

m4 マクロにより記述した。

（2）デバイスドライバ

開発した周辺回路（タイマ、シリアル通信および DMA）を操作するデバイスドライバを開発した。それぞれ初期化ルーチン、割込み処理ルーチン、およびシステム側へのインタフェースを提供する。後述のランタイムライブラリ関数から呼び出され実行される。

（3）ランタイムライブラリ

本 RISC コアは SPARC の実装依存命令を利用し、各種リアルタイム処理支援機構を実現している。C 言語からこれらの機構を利用可能とするために、実装依存命令が埋め込まれたランタイムライブラリを作成した。実行速度を重視する機能は、関数呼び出しのオーバヘッドを削減するためにマクロ関数として実装した。

・レジスタセット間データ移動マクロ

任意のレジスタセットモードの実行時に、任意のレジスタセットをターゲットとしたデータ移動を可能とする拡張命令を使用するマクロ関数である。

・キャッシュ制御マクロ

データキャッシュの明示的制御機構を使用するマクロ関数であり、キャッシュのヒット／ミスにかかわらず外部メモリから強制的にブロックをフィルする機構と、指定されたキャッシュ内のブロックが更新されていた場合に外部メモリをアップデートする機構を使用する。

・プロセッサ割込みレベル変更マクロ

1 命令の実行でプロセッサの状態レジスタ内の割込み受け付けレベルを変更するマクロ関数である。これによりハンドラ間のクリティカルセクションを高速に制御可能である。

（4）その他のライブラリ関数

タイマ回路を制御することにより、タイマを直接操作する関数、現在の日付／時刻を得る関数、プログラムの経過時間を得る関数を実装した。また、シリアル通信回路を制御することにより、ユーザプログラムからシリアル送信、受信を行うライブラリ関数を実装した。メモリ領域と送受信バイト数を指定することによりシリアル通信が行われる。さらに、DMA 機構を使用するライブラリ関数を実装した。

（5）リンカ

ユーザプログラムをシステムの基本ルーチンとリンクして実行バイナリコードを生成するリンカを作成した。ユーザはアプリケーションプログラムやシステム依存の割込みルーチンを C 言語で記述することにより、図 4 のように基本ルーチンとリンクされる。

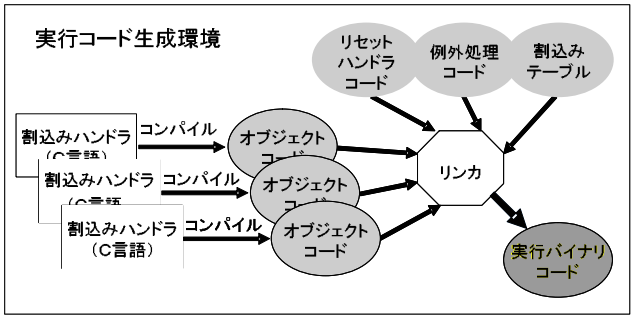

図 4　リンカによる実行コード生成

図 5 に実行バイナリが使用するメモリマップを示す。

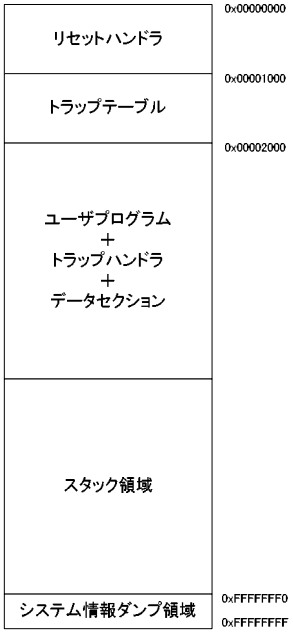

図 5　メモリマップ

7．まとめ

本プロジェクトで開発した RISC コア、周辺回路およびマルチプロセッサ支援機構は、最終的にはフリーなオープンソースコードとすることで研究／開発を支援することを目的としている。コア全般を標準的なハードウェア記述言語である VHDL を使用して設計しており、多くのベンダの CAD システムに適用可能である。利用容易性の観点からソースコードの整備を行った。また、コアおよび周辺回路の仕様およびユーザマニュアル的なドキュメントを整備した。

プログラム開発支援として、リセットハンドラ、例外処理ルーチン、各種デバイスドライバ、ランタイムライブラリ、実行バイナリ生成リンカを実装した。これにより、システム開発時に、実際のプログラムをシミュレーションすることによる検証が可能となる。また、実装したライブラリはコアアーキテクチャのリアルタイム処理機構を使用するものであり、システムの高速なハンドリングを可能とする。

現在実装した回路およびライブラリ類をプロトタイプボード上で検証中であり、検証が終了次第開発物を組込みシステム開発支援オープンソースライブラリとして公開することを予定している。

8．参加企業及び機関

・三菱マテリアル株式会社

・三精システム株式会社

9．参考文献

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

［2］http://gcc.gnu.org/