修士論文

プロセッサ設計支援ツールの実装と
ハード/ソフト協調学習システムの評価

氏　　名	:	難波　翔一朗
学籍番号	:	6124050168-5
指導教員	:	山崎　勝弘　教授
提 出 日	:	2007年2月16日

立命館大学大学院　理工学研究科　情報システム学専攻

内容梗概

本論文では、本研究室で開発しているハード/ソフト協調学習システム上で、ハードウェアとソフトウェアの関係を理解する重要性について検討し、命令セット設計とプロセッサ実装の学習フローを提案する。また、これらの学習フローを実現するためのツールとして、プロセッサ設計支援ツール（命令セット定義ツール、汎用アセンブラ、プロセッサモニタ、プロセッサデバッガ）の開発を行い、その設計と検証について述べる。さらに、本研究室の 4 回生を対象としたプロセッサ設計の実例を示し、本システムの学習効果の考察と評価を行う。

プロセッサデバッガ・モニタの設計では、RS232C ポートを利用し、ホスト PC と FPGA ボードを接続させ、ボード上のスイッチや LED を使用することなく実機上のプロセッサのデバッグを行える機能を盛り込んだ。また、プロセッサをデバッグするコマンドとして、メモリ・レジスタへのデータの読み書き、プロセッサの実行と停止、ブレイクポイントの設定機能を実現した。本ツールを用いることで、研究室の 4 回生によって複数のアーキテクチャのプロセッサが実機上に実装され、その動作検証を行えた。研究室での学習を通して、16 ビットの SOAR プロセッサ、9 種類の MONI プロセッサの実装を行い、複数のプログラムの実行を行った。また、ARM ライク命令セットについても試作され、ハード/ソフト協調学習システムを用いることで行えた。

プロセッサ設計支援ツールの実装により、従来のハード/ソフト協調学習システムでは FPGA 上への実装までに 150 時間所要であったが、MONI プロセッサの実装では、9 種類のアーキテクチャそれぞれにおいて 30 時間で実機検証でき、大幅な設計時間の短縮が可能になった。また、MONI 以外のプロセッサとして ARM ライク命令セットも試作され、命令セット設計を取り入れたハード/ソフト協調学習システムとしては、その利用価値を評価することができた。

目次

図目次

表目次

1. はじめに

1.1 研究背景

2006 年、地上デジタル放送が全国で開始されたのを筆頭に、WiFi、Bluetooth、ワイヤレス USB など無線通信に関する規格整備が整ってきたことから、デジタルテレビ、携帯電話、ポータブル音楽プレーヤ、デジタルカメラ、ゲーム機、ノートパソコン、カーナビゲーションシステムなど、ネットワーク接続可能な製品が数多く販売されるようになった。これらユビキタスネットワークに対応した電子機器の組み込みシステムでは、その特性上、小規模で低消費電力であることが求められる一方、多様化したニーズに応えるべく、様々な機能を実現しなければならない。一般家庭向けの家電製品においても、近年では、規格や技術の進展は目まぐるしく、製品の設計サイクルは短縮されているにも関わらず、高性能・高機能なものが求められている。コンピュータ制御が必要な機器内の組み込みシステムには、厳しい要求を素早く実現することが不可欠となっている。

そこで、デジタル機器向けの組み込みシステム開発の現場では、無線通信処理や動画・音声処理など、多くの製品で共通して使用できるモジュールは、以前の設計資産やベンダーが提供する IP を利用し、設計の効率化を図っている。また、半導体メーカや電機メーカ各社では、予めどの製品でも使用できる汎用的なプラットフォームを用意しておき、設計するターゲットシステムにあわせて専用ハードウェアやプロセッサのカスタマイズを行うプラットフォームベース設計の手法を提案している。LSI 設計においては、これらの手法を用いることで、ソフトウェア設計量の削減と開発期間の短縮を実現しながらも、要求を満たす組み込みシステムの開発を行っているが、しかし、この手法は同時に、基本的なコンピュータアーキテクチャに加え、ハードウェアとソフトウェアの性能のトレードオフを完全に理解していることが前提となっていることも事実である。

このような背景から、大学教育にはハードウェアとソフトウェアを深く理解し、その上でシステム設計を提案できる人材を育成することが求められている。特に、プロセッサを中心とするシステムでは、ハードウェアの内部構造からその上で動作するソフトウェアのプログラミングスキルまで、ハード/ソフト協調設計で必要な要素が集約されており、これらの知識を有する学生を育てることは、半導体業界において強く望まれていることである。一方、大学の計算機工学の教育現場でもプロセッサの設計技術に関する学習は重要なポイントとされ、様々な大学でプロセッサを用いた教育システム例が報告されている。中でも、ハードウェアとソフトウェアの学習を組み合わせた教育システムは、書き換え可能な LSI の FPGA が広まった事もあり、現在、多くの大学から成果が報告されている[1-5]。しかし、その多くは、ハードウェアとソフトウェアの学習を別々の実験で取り上げるものであり、両者の性能のトレードオフや相乗効果につて学べるシステムは少ない。ハードウェアとソフトウェアそれぞれの性能を両立して引き上げることは、システム設計者にとって非常に重要な技術であり、両者の関係を深く理解できる学習システムの開発が渇望されている。そこで、本研究室ではプロセッサにおけるハードウェアとソフトウェアの設計を通じて、

両者のトレードオフを学習するハード/ソフト協調学習システムの研究を進めている[8-15]。

1.2 研究目的

ハード/ソフト協調学習システム（以下 HSCS）は、プロセッサにおけるソフトウェアとハードウェアを設計することで両者の実装技術を身につけ、性能のトレードオフを学習する教育システムである。本システムの最終的な目的は、プロセッサの上で実現されるシステムの高性能化技術を理解することである。我々はこの目的を達成するためには、命令セットアーキテクチャの選定、プロセッサアーキテクチャの高速化手法、さらに、それらをうまく結びつける知識が必要であると考えた。本研究では、協調学習システムに独自命令セットの設計とその命令セットを実現するプロセッサの開発を繰り返し行う学習フローを追加する。そして、学生が主体的にシステムを構築できる学習環境を整える。学習内容の拡張に伴い、HSCS で扱う内容の技術レベルは引き上げられ、本システムが対象とする大学の学部生にとっては厳しい内容になると予想される。この問題を緩和するため、我々は、プロセッサの設計を支援するツールを開発し、学生が容易にシステムに最適な命令セット設計とそのハードウェアを構築する環境を提供する。すなわち、命令セットを定義・検証するためのツールとして命令セット定義ツール、汎用アセンブラ・シミュレータを、また、プロセッサを実装するためのツールとしてプロセッサデバッガ・モニタを設計する。

これまでのハード/ソフト協調学習システムの研究では、基本命令セット MONI の設計と実装、FPGA ボードコンピュータの実装、命令セット定義ツール・汎用アセンブラの設計、プロセッサデバッガの試作を行ってきた。本研究ではこれらのプロセッサ設計を支援するツールの開発と FPGA ボードへの実装を行うこと、及び本ツールを用いて複数のプロセッサをハード/ソフト協調学習システム上に実装し、本システムの有効性の検証を行うことを目的とする。対象とするプロセッサは MONI、SOAR、及び ARM ライクのプロセッサである。これらのプロセッサをプロセッサデバッガと接続して FPGA 上に実装し、ホスト PC 上のプロセッサモニタから検証用のコマンドを実行することで、プロセッサの動作を検証できることを示す。本論文では、これら一連の設計と実装結果について述べ、最後に本システムにおけるプロセッサ設計支援ツールの評価と考察を行う。また、本ツールを利用してプロセッサを作成した本研究室の 4 回生の学習内容について示し、その教育効果を考察する。

1.3 関連研究

1.3.1 大学における計算機工学の教育システム

(1) KITE プロジェクト[1,2]

熊本大学で設計された 16 ビットのプロセッサを利用した計算機工学の学習システム。現在では FPGA を搭載した Super-KITE ボードを用いることで、KITE マイクロプロセッサ以外のプロセッサも実装でき、システムレベル設計の教育で活用されている。

(2) City-1[5]

広島市立大学での学生実験で利用されている教育システム。FPGA を利用して、RISC や CISC のプロセッサ設計から、ボードの試作まで幅広く学習できるシステム。先に対象の問題を与え、これを計算できるプロセッサを学生が設計する学習フローを持つ。

1.3.2 プラットフォームベース設計

プラットフォームベース設計では、組み込みプロセッサ・バス・周辺モジュールなどを共通化することで、製品を跨いだアプリケーションの共有を実現し、ソフトウェア開発の効率を向上させている。ハードウェアの開発においても、ターゲットシステムに特化した専用ハードウェアの設計やコンフィギャブルプロセッサへの命令の追加など、必要最低限の作業によるシステム開発を実現している。

(1) UniPhier：松下電器産業株式会社[25]

松下電器産業株式会社が開発した、デジタル家電向けの総合プラットフォーム。デジタル各機器で共通に使用できる CPU、ビデオコーディック、OS、専用ハードウェアなどをまとめ、様々な機器に搭載することで製品の開発効率を目指した技術である。

(2) platformOViA：NEC エレクトロニクス株式会社[24]

NEC エレクトロニクス株式会社が提供している、モバイル機器、デジタル AV 機器、車載システムを主に対象とした、システムソリューションプラットフォーム。platformOViA を利用することで、設計からハード/ソフトの分割開発からシステムの検証までをまとめて管理し、早期のシステム実装を目指すシステムである。

2. ハード/ソフト協調学習システム

2.1 従来の研究成果

限られたハードウェア資源しか与えられない組み込みシステムの上で、最高のパフォーマンスを得るには、搭載されているプロセッサのアーキテクチャに適したソフトウェア開発を行わなければならない。プロセッサアーキテクチャによっては、命令順序の変更で性能が数段向上することや、命令セットを生かしたコーディング行うことによってメモリアクセスを軽減できることもしばしばある。そこで、我々はプロセッサのアーキテクチャの理解がソフトウェアの高速化につながると考え、その関係を理解するために、アセンブリプログラミングとハードウェア設計を融合させたハード/ソフト協調学習システム（以下HSCS）の研究を進めてきた。HSCSの学習フローを図 1に示す。

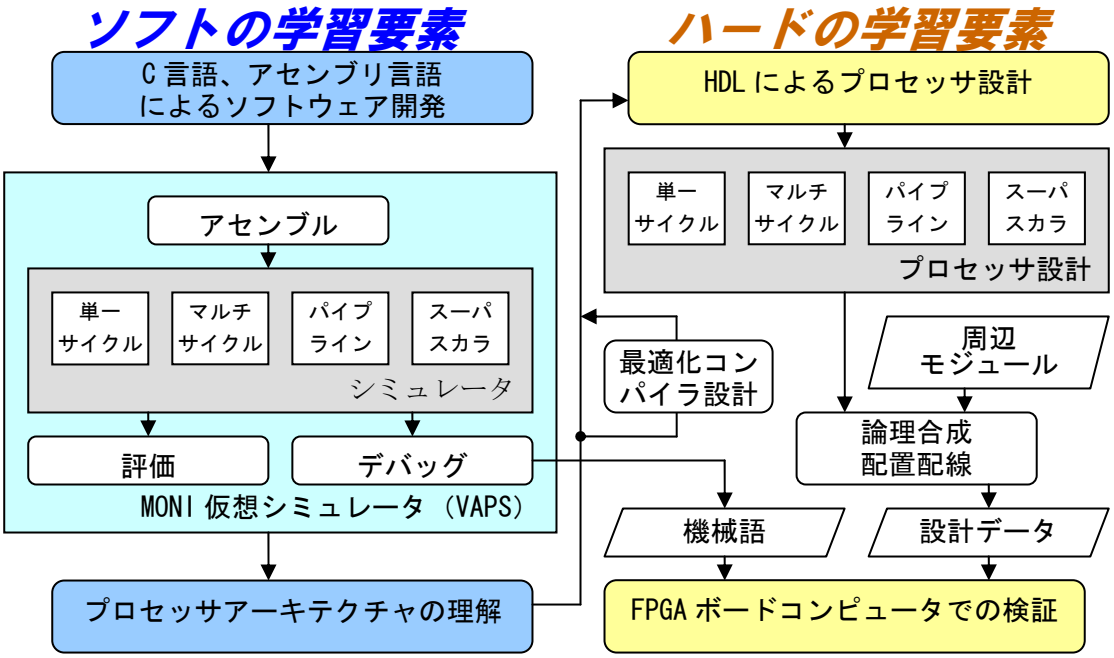

図 1　ハード/ソフト協調学習システムの学習フロー

HSCS は、学習用の命令セット MONI を対象に、アセンブリプログラミング（ソフトウェア学習）と、HDL によるプロセッサ設計（ハードウェア学習）を行う学習システムである。MONI は、学習用の命令セットとして定義した 3 オペランド形式、43 命令からなる RISC 命令セットであり、ハードウェアとしてはシングルサイクル、マルチサイクル、パイプライン、スーパスカラのアーキテクチャが実現可能である。HSCS におけるソフトウェア学習では、MONI の複数のプロセッサアーキテクチャに対応した GUI ベースの命令セットシミュレータ VAPS（Variable Architecture Processor Simulator）を利用し、アセンブリプログラミングを行う。VAPS では、シミュレーション時にプロセッサ内部のデータフロ

ーが確認でき、プロセッサアーキテクチャを学びながらプログラミングを行うことができる。ハードウェア学習では HDL によって設計したプロセッサを FPGA 上に搭載し、ソフトウェア学習で作成したプログラムを実行することで実機上での検証が可能である。これらの学習を通して、学習者はハードウェアを理解した上でのソフトウェア開発能力を身につけることができる

2.2 プロセッサ設計を取り入れた協調学習システム

従来のHSCSは、プロセッサを意識したプログラミング学習のための教育システムであった。我々は、HSCSの目標をコンピュータシステムの体系的理解に拡張し、プロセッサ設計を通じてハードウェアとソフトウェアの構成技術と、それぞれの性能のトレードオフを学習する教育システムに発展させる。本システムでは、学生に独自の命令セットと、プロセッサの設計を繰り返し行わせ、両者の関係を深く理解させることが最大の目的となる。命令セットを設計することで、システムで動作するソフトウェアの性能を調べ、プロセッサを設計することで命令の実現方法を検討する。両者を併せて設計できて初めて高い性能のシステムを実現できる。図 2にHSCSの学習体型のイメージを示す。

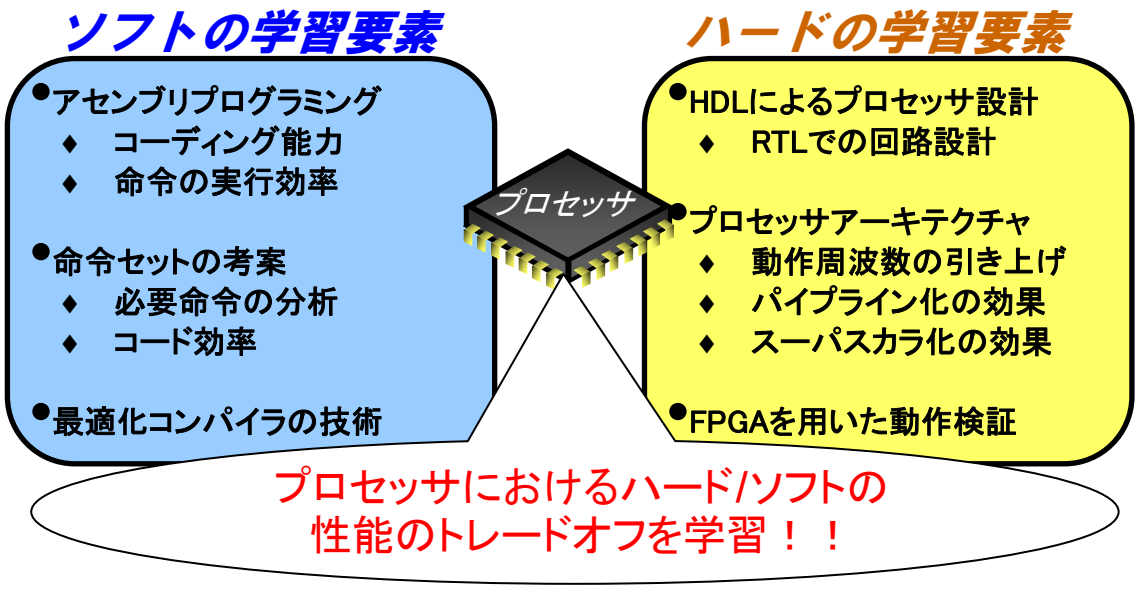

図 2　ハード/ソフト協調学習システムの学習体系イメージ

図 2で示すように、本システムで求める学習要素は、大きく分けてソフトウェア面とハードウェア面に分けられる。ここに挙げた要素は、いずれもその理解力が大きくシステムの性能を左右するものであり、本システムにおいて重要な学習のポイントである。

ソフトウェア面の学習要素では、最もハードウェアのインタフェースに近い部分の理解を重要な点として挙げる。1 点目は、低級言語でのプログラミング技術の習得である。アセンブリプログラミングを通して、ループや分岐の効率的な実現方法を身につけると同時に、

プログラムが及ぼすシステムへの影響を知る。2 点目は、命令セットの考案である。システムに合ったビット幅や命令を選択できるかどうかは対象問題の特性を分析することでもあり、ハードウェアのアーキテクチャを決定することにもつながる。これにより、システム全体の分析能力とコード効率を考えた命令セット設計能力を身につける。3 点目は、コンパイラの最適化技術の学習である。高級言語から生成したプログラムをパイプラインやスーパスカラのプロセッサで、より無駄なく実行するための技術を学ぶ。

ハードウェア面では､1 点目に HDL によるハードウェア設計の習得を挙げる｡ここでは、基本的な HDL の文法やシミュレーションのスキルを身につける。2 点目は、プロセッサアーキテクチャの理解である。プロセッサアーキテクチャには様々な高速化技術が存在するが、必ずしも高度な技術が最適とは限らない。どのアーキテクチャが望ましいのかは自分の考えた命令セットや対象とするシステムによって適切に判断し、システムや命令セットに応じてどのようなアーキテクチャで実現すべきかを決定できることが重要なポイントである。アーキテクチャの決定では、例えば、「対象システムでは分岐が発生することが多いのでパイプライン化の効果は上がらない」や、「2 オペランドの命令セットを採用したのでスーパスカラ化が容易である」といった内容が判断材料になる。3 点目として、実機上での動作検証能力の育成を挙げる。チップ上に HDL で設計したプロセッサを実装し、自作のプログラムを実機上で動作させることで、システム全体の性能評価できる点も重要な要素である。

2.3 システムの機能

図 3にプロセッサ設計支援ツールを取り入れた協調学習システムを示す。まず、学習者はサンプルプログラムの課題について､MONI命令セットを用いてアセンブリプログラミングを行い､MONIシミュレータで評価する。ここで基本的なプロセッサのハードウェアアーキテクチャや命令実行時の動作について理解し、アセンブリプログラミングに必要な最低限の命令やアドレッシングモードについて学ぶ。次に学習者はMONI命令セットを拡張し、サンプルプログラムをより効率よく実行できる命令セットを作成する。命令セットの定義には命令セット定義ツールを、評価には汎用アセンブラ･シミュレータを用いることで、実際の課題プログラムをシミュレーションしながら、拡張した命令セットが有効であるかを検証する。ここでは実行効率を意識した命令セット設計を行い、ソフトウェア面での高効率化技術を養う。最後に、作成した命令セット用のプロセッサ設計と実装を行う。プロセッサコアの実機検証には我々が開発したプロセッサ評価用のFPGAボードコンピュータシステムとプロセッサデバッガ・モニタを用いる。ここでは特に、論理的には実現可能な命令でも、実際のハードウェアでは最大動作周波数の低下や回路規模の増大を招くことを体験し、要求を満たすハードウェア設計技術を習得する。要求を満たさない場合は再度命令セット定義からやり直し、命令セットとプロセッサの設計を繰り返し行うことでハードウェアとソフトウェアの境界を理解する。

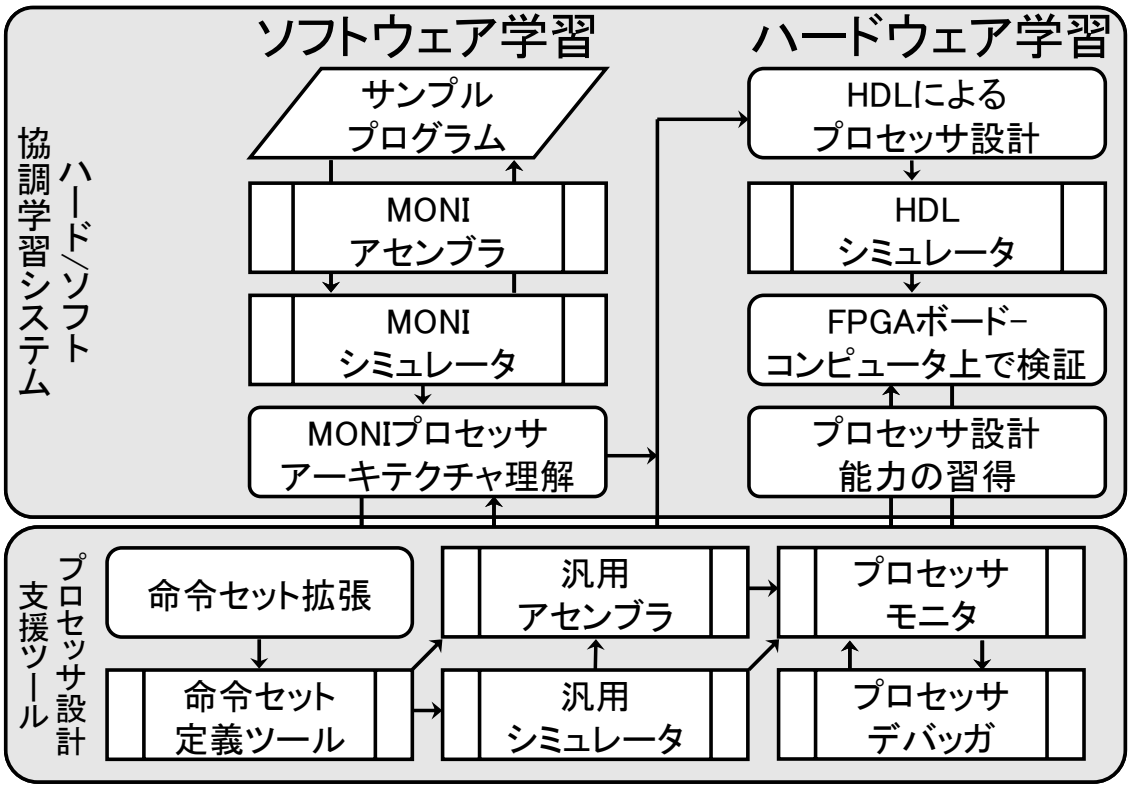

図 3　プロセッサ設計支援ツールを取り入れた協調学習システム

本学習システムにおいて、必要になるのがプロセッサ設計支援ツールである。本ツールは、プロセッサ設計で必要な情報を一括管理し、学生にプロセッサにおけるハードウェアとソフトウェアの関係の学習を促す。学生が一からプロセッサを設計することもできるが、本ツールでは基本命令セット MONI の拡張による学習スタイルを推奨する。現在は、命令セットを定義するツール、汎用的なアセンブラ、実機上での検証環境を個別に提供しているが、将来的にはこれらを連携させ、命令セットやアーキテクチャの変更がシステムに与える影響を評価できる環境構築を目指す。この構想が実現できれば、ある高級言語で記述されたシステムを異なるプロセッサ上で実行することも可能となり、よりハード/ソフトのトレードオフを理解できる学習システムに発展する。

3. プロセッサ設計支援ツールの設計

3.1 プロセッサ設計支援ツールの概要

プロセッサ設計支援ツールは、HSCS を利用する学生が必要最低限の知識で命令セットとプロセッサを設計できるよう開発したツールである。本ツールは、命令セットの作成と管理を行う命令セット定義ツール、設計した命令セットを評価するための汎用アセンブラ・シミュレータ、HDL で設計したプロセッサを FPGA 上でデバッグするプロセッサデバッガ・モニタによって構成される。本ツールは、大きく分けて 2 つの学習領域で利用する。1 つは命令セットの設計、もう 1 つはプロセッサの設計である。

命令セット定義ツール、及び汎用アセンブラ・シミュレータは命令セットを定義する学習で利用する。命令セット定義ツールは、ビット幅やフィールドを順に定義するだけで命令セットを定義でき、その性能を汎用アセンブラ・シミュレータで確認することができる。命令セットを新しく作成した場合、アセンブラの開発は欠かせない作業であるが、汎用アセンブラを導入することで、学生はアセンブラの設計から解放され、命令セットの作成や改良にのみ専念することができる。また、汎用シミュレータを利用すると、設計した命令セットで繰り返し動作させながら行える環境が整うことから、学生にとっては、より具体的に命令の追加や拡張の効果をしることができ、容易に独自命令セットを定義できるメリットがある。

プロセッサデバッガとプロセッサモニタは、プロセッサ開発の学習で用い、FPGA ボード上に自作プロセッサを実装検証できる。今までの HSCS では、LED やスイッチなど限られた入出力しかデバッグ用に利用できないことが実機実装の大きな課題であったが、FPGA 上にプロセッサデバッガを搭載することで、ホスト PC 上のプロセッサモニタからコマンド操作することで自作のプロセッサのデバッグが行うことを可能とした。

本章では、プロセッサ設計支援ツール構成する各ツールの設計について詳しく述べる。

3.2 命令セット定義ツールと汎用アセンブラ・シミュレータ

学習者に命令セット設計を課題として与えるには、命令セットの定義とその検証を容易に繰り返し行える環境を整える必要があった。そこで我々は、必要な情報を入力することで命令セットの定義を行う命令セット定義ツール、及び命令セットを評価する汎用アセンブラを開発した。命令セット定義ツールと汎用アセンブラは、システムを設計する上でどのような命令を選択するか、また、どのようなプロセッサアーキテクチャを構成するかという二点を重点的に学習するためのツールとして提供する。すなわち、学生が命令セットの設計と検証を容易に繰り返し行えること、及びプロセッサ設計のフィードバックを命令セット設計にかけられることを本ツールの目的とする。図 4に命令セット定義ツール、汎用アセンブラ、汎用シミュレータを用いた命令セットを定義する流れを示す。

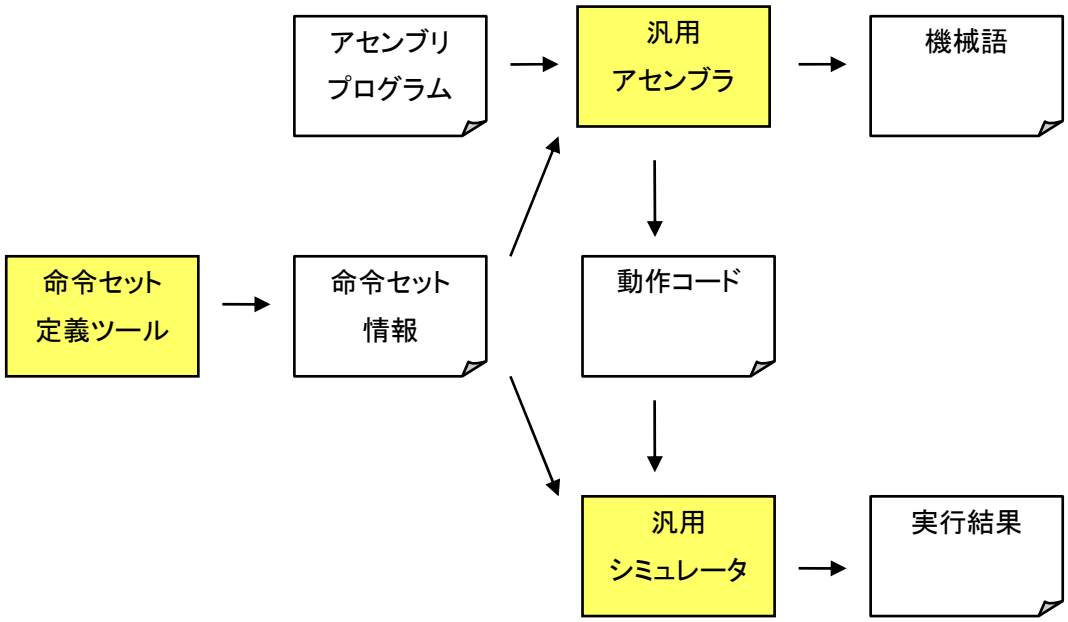

図 4　命令セット定義の流れ

命令セット定義ツールにより作成した命令セットは、命令セット情報としてファイルに落とされる。汎用アセンブラでは、命令セット情報と、オリジナル命令セットで組まれたアセンブリプログラムを入力することで、それぞれの命令セットに応じた機械語が生成される。また、同時に汎用シミュレータに入力する動作コードも生成される。汎用シミュレータは、動作コードと命令セット情報内のレジスタやメモリに関する情報を入力にとり、シミュレーションを行う。出力として、命令セットの解析結果が得られるツールとする。

3.2.1 命令セット定義ツールの概要

命令セット定義ツールは、命令セットの情報（ISI：Instruction-Set Information）を一括して管理するISIM（Instruction Set Information Manager：命令セット情報マネージャ）に命令の情報を設定・更新・取得することで命令セットの定義が行えるツールである。図5にISIMの概観を示す。ISIMは、大きく分けてデータ部と解析部に分けられる。データ部は、アーキテクチャやフィールド、命令といった命令セットに必要な定義情報をインスタンスとして保持している。また、新たに命令などを追加する場合に矛盾が生じないように管理する。解析部は、すでに定義された命令セットの情報をファイルから読み込む際に、データの整合性があるかチェックするモジュールである。本ツールで定義可能な項目を表1に示す。本ツールで対象とする命令セットは、定義のしやすさを考慮し、RISCに限定する。よって、定義可能な命令セットは固定命令長で 8～64 ビットの 4 種類から選択するものとした。また、アドレッシングモードはレジスタ間接と即値を用意した。その他のアドレッシングモードは、これらを組み合わせることによって定義可能としている。

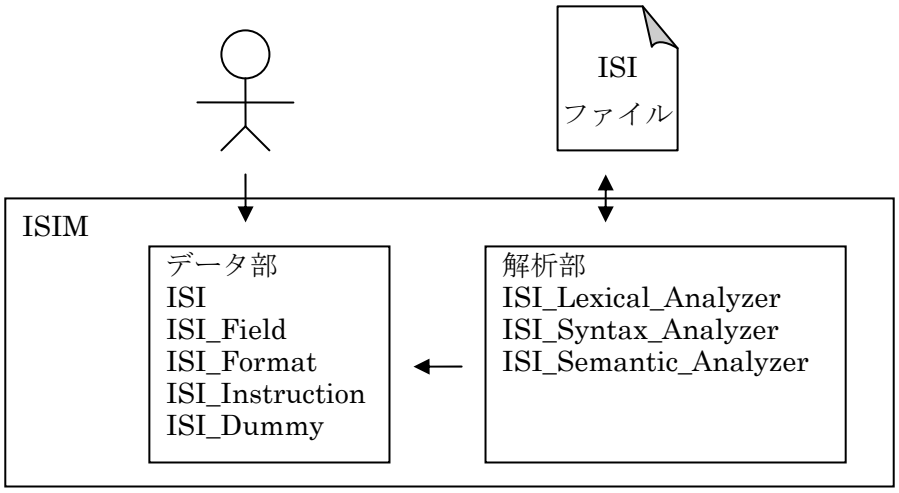

図 5　命令セットマネージャの全体イメージ

表 1　命令セット定義ツールで定義可能な項目

項目	対応
命令長	8, 16, 32, 64bit 固定長
命令セット・命令形式	自由に設定可能
プロセッサの構成様式	RISC を対象
レジスタ	汎用レジスタ, PC, SP
アドレッシングモード	レジスタ間接, 即値, 他
メモリアーキテクチャ	自由に設定可能
擬似命令（マクロ命令）	対応

3.2.2 汎用アセンブラの概要

汎用アセンブラは、ISIのオペコードとそれに対応する機械語の情報をもとに、入力されたアセンブリソースをアセンブルするモジュールである。ISIを利用することで汎用アセンブラは、様々な命令セットのアセンブリプログラムを汎用的にアセンブルできる。汎用化のために、アセンブリソースは表 2に示す構文に限定する。

表 2　汎用アセンブラの構文

項目	構文
コメント	“//”から改行まで ”/*”から”*/”まで
ラベル	最初の文字はアルファベット、以降は英数字で表記 命令の前に、“:”によって区切る 絶対アドレスと相対アドレスに変換
ニーモニック	ISI で定義されている命令を、ニーモニックとして扱う
数値定数	10 進数、8 進数、16 進数、2 進数から記述可
定数マクロ	利用可
オペランド	定数で記述（レジスタアクセスに”$”などをつけない）

3.3 プロセッサモニタの設計

3.3.1 プロセッサモニタの概要

プロセッサモニタは、プロセッサ設計の学習で利用するソフトウェアアプリケーションである。学生がHDLによって設計したプロセッサをFPGA上で動作検証する際に、ホストPC上から操作するツールとして利用する。プロセッサモニタの役割は 2 つある。1 つは、ユーザからのデバッグ要求を解釈しプロセッサデバッガに向け指示を与えること。もう 1 つは、ホストPC上に展開した仮想メモリバッファとプロセッサデバッガ内のメモリの同期をとることである。図 6にプロセッサモニタの構成を示す。プロセッサモニタのトップモジュールはMonitorである。Monitorはユーザとの入出力を担当するインタフェースを主に担う。Monitorの内部には、プロセッサデバッガ内のメモリを摸したMemoryとプロセッサデバッガとの通信データが格納されているFrameモジュールを持つ。さらに、Frameは自身のデータを送信するためのポートとしてrs232cモジュールを内包する。rs232cモジュールはシリアルポートに関する情報を保有し、プロセッサデバッガ上のrs232cモジュールとの通信を確立する。

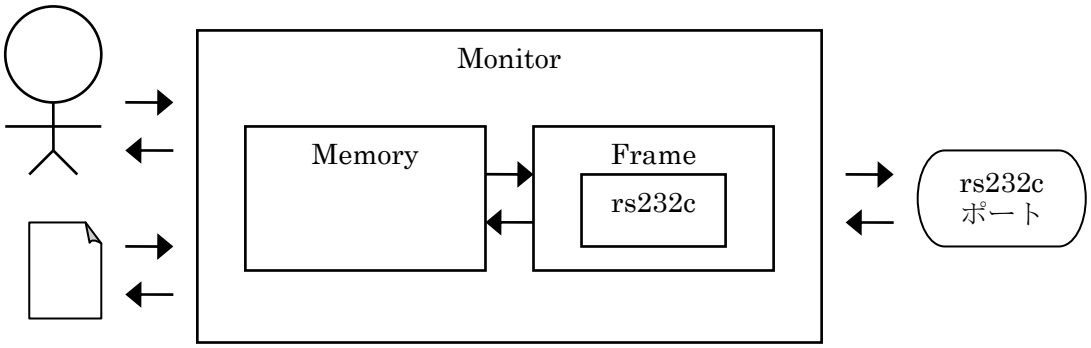

図 6　プロセッサモニタの構成

プロセッサモニタでは、FPGA 上のメモリ内容を read コマンドによって受信し、内部の Memory モジュールに保持する。ユーザはこのデータを各種デバッグ用のコマンドで自由に編集でき、編集後、送信コマンドによって再度 FPGA 上に展開する。このようにメモリの情報をホスト PC 上で再現することで、実機上のデータを自由に扱える仮想デバッグ環境を提供する。プロセッサモニタの導入により、FPGA ボード上の限られた入出力によらず、柔軟なデバッグ操作が可能となる。

3.3.2 デバッグコマンド

プロセッサモニタでは、ユーザ側とプロセッサデバッガ側で異なるコマンド体系を扱う。すなわち、ユーザが入力するユーザコマンドと、プロセッサモニタがプロセッサデバッガに指示を与えるアプリケーションコマンドである。これら 2 系統のコマンドを使用するこ

とで、ホスト PC 上からの単純なコマンド操作で複雑なデバッグ処理を指示することができる。それぞれのコマンド体系を説明する。

(a) ユーザコマンド

ユーザコマンドは、プロセッサモニタがユーザに提供するコマンド体系である。表 3にユーザコマンドの一覧を示す。ユーザコマンドには、ファイルアクセス、メモリデータの送受信、プロセッサの実行などユーザが直感的にわかりやすいコマンドが用意されている。また、同じ送信コマンドでも引数を変更することによって、DM・IM・レジスタ・プログラムカウンタなど様々なターゲットを指定することができる。

表 3　プロセッサデバッガのユーザコマンド一覧

コマンド	意味
send (all)	全バッファを書き込み
send dm (アドレス) (量)	dm バッファからデータメモリに書き込み
send im (アドレス) (量)	im バッファから命令メモリに書き込み
send rf (アドレス) (量)	rfバッファからレジスタファイルに書き込み
send pc	pc バッファからプログラムカウンタに書き込み
send bp	bp バッファからブレイクポイントメモリに書き込み
read (all)	全バッファにメモリ・レジスタ値を読み込み
read dm (アドレス) (量)	dm バッファにデータメモリの値を読み込み
read im (アドレス) (量)	im バッファに命令メモリの値を読み込み
read rf (アドレス) (量)	rfバッファにレジスタファイルに値を読み込み
read pc	pc バッファにプログラムカウンタの値を読み込み
read bp	bp バッファにブレイクポイントメモリの値を読み込み
load all [ファイル]	全バッファにファイルの内容を読み込み
load dm [ファイル]	dm バッファにファイルの内容を読み込み
load im [ファイル]	im バッファにファイルの内容を読み込み
load rf [ファイル]	rfバッファにファイルの内容を読み込み
load pc [ファイル]	dm バッファにファイルの内容を読み込み
load bp [ファイル]	im バッファにファイルの内容を読み込み
save all [ファイル]	全バッファをファイルに保存
save dm [ファイル]	dm バッファをファイルに保存
save im [ファイル]	im バッファをファイルに保存
save rf [ファイル]	rfバッファをファイルに保存
save pc [ファイル]	pc バッファをファイルに保存
save bp [ファイル]	bp バッファをファイルに保存
set bp [ポイント]	ブレイクポイントの設定
del bp [ポイント]	ブレイクポイントの削除
list (all)	全バッファの表示
list dm	dm バッファを表示
list im	im バッファを表示
list rf	rf バッファを表示
list pc	pc バッファを表示
list bp	bp バッファを表示
init (all)	全バッファの初期化
init dm	dm バッファを初期化
init im	im バッファを初期化
init rf	rf バッファを初期化
init pc	pc バッファを初期化
init bp	bp バッファを初期化
run (all)	最後まで実行
run next	次のブレイクポイントまで実行
run clk	次のクロックまで実行
run inst	次の命令まで実行
halt	プロセッサの停止
help	コマンド一覧の表示
mode	プロセッサモニタのインタフェースモードを変更
exit	プロセッサモニタの終了

(b) アプリケーションコマンド

アプリケーションコマンドはプロセッサデバッガで処理できるよう複雑な引数やオプションを指定しないコマンド体系である。本モジュールで実行できるアプリケーションコマンドを表 4に示す。このコマンドはプロセッサモニタとプロセッサデバッガ間で使用される通信コマンドであり、直接ユーザが意識することはない。ホストPC上でユーザが入力したユーザコマンドは、プロセッサモニタによってアプリケーションコマンドに変換され、フレームという単位にパッキングされてプロセッサデバッガに送信される。プロセッサデバッガからホストPCにデータを送信する場合も、アプリケーションコマンドの形式を利用する。フレームについては3.4節のプロセッサデバッガの設計で詳しく述べる。

表 4　プロセッサデバッガのアプリケーションコマンド一覧

コマンド	コマンド名	コマンド値	意味
NOP	PD_INST_NOP	0x0000	何もしない
PUT DM	PD_INST_PUT_DM	0x0001	DM に書き込み
PUT IM	PD_INST_PUT_IM	0x0002	IM に書き込み
PUT RF	PD_INST_PUT_RF	0x0003	RF に書き込み
PUT PC	PD_INST_PUT_PC	0x0004	PC に書き込み
PUT BP	PD_INST_PUT_BP	0x0005	BP に書き込み
GET DM	PD_INST_GET_DM	0x0006	DM を読み出し
GET IM	PD_INST_GET_IM	0x0007	IM を読み出し
GET RF	PD_INST_GET_RF	0x0008	RF を読み出し
GET PC	PD_INST_GET_PC	0x0009	PC を読み出し
GET BP	PD_INST_GET_BP	0x000a	BP を読み出し
SET BP	PD_INST_SET_BP	0x000b	BP をセット
DEL BP	PD_INST_DEL_BP	0x000c	BP を削除
RUN MPU	PD_INST_RUN_MPU	0x000d	プロセッサを実行
RUN INST	PD_INST_RUN_INST	0x000e	次の命令まで実行
RUN CLK	PD_INST_RUN_CLK	0x000f	次のクロックまで実行
HATL MPU	PD_INST_HALT_MPU	0x0010	プロセッサの強制終了

3.3.3 プロセッサモニタの設計

プロセッサモニタは、ユーザ側とプロセッサデバッガ側で異なるコマンド体系を扱うため、モジュールの設計において 2 つインタフェースを基軸に置く。ユーザとの対話的な入出力によってユーザコマンドを受け付け、要求されたデバッグ処理を複数のアプリケーションコマンドによって実現する。この構成により 2 つのコマンド体系を明確に切り分けることができ、プロセッサデバッガの仕様変更や、コマンドの追加に強いシステムが構築できる。本モジュールでは、上記の設計方針で実現するため、オブジェクト指向の考え方を取り入れ、コマンドの情報を保持・編集できるようなクラスにモジュールに分割した。プロセッサモニタのクラスを図 7に示す。プロセッサモニタでは、Monitorクラスがユーザコマンドの情報を管理し、アプリケーションコマンドの送受信通信制御を行う。ユーザコマンドのデータはMonitor内部のメンバ変数で保持し、パラメータによって内部の制御を容易

に行える構成とする。一方、プロセッサデバッガと複数のフレームを繰り返しやりとりしなければならないアプリケーションコマンドは、クラスによって機能分割し、フレーム生成、通信データの整合性のチェックを一元管理する。Memoryモジュールは、FPGA上のメモリの内容をホストPC上でモデリングするためのクラスであり、Monitorクラスと 1 対 1 の関係である。rs232cクラスはFrameクラスが内部に保持し、フレームの送受信の制御をカプセル化している。

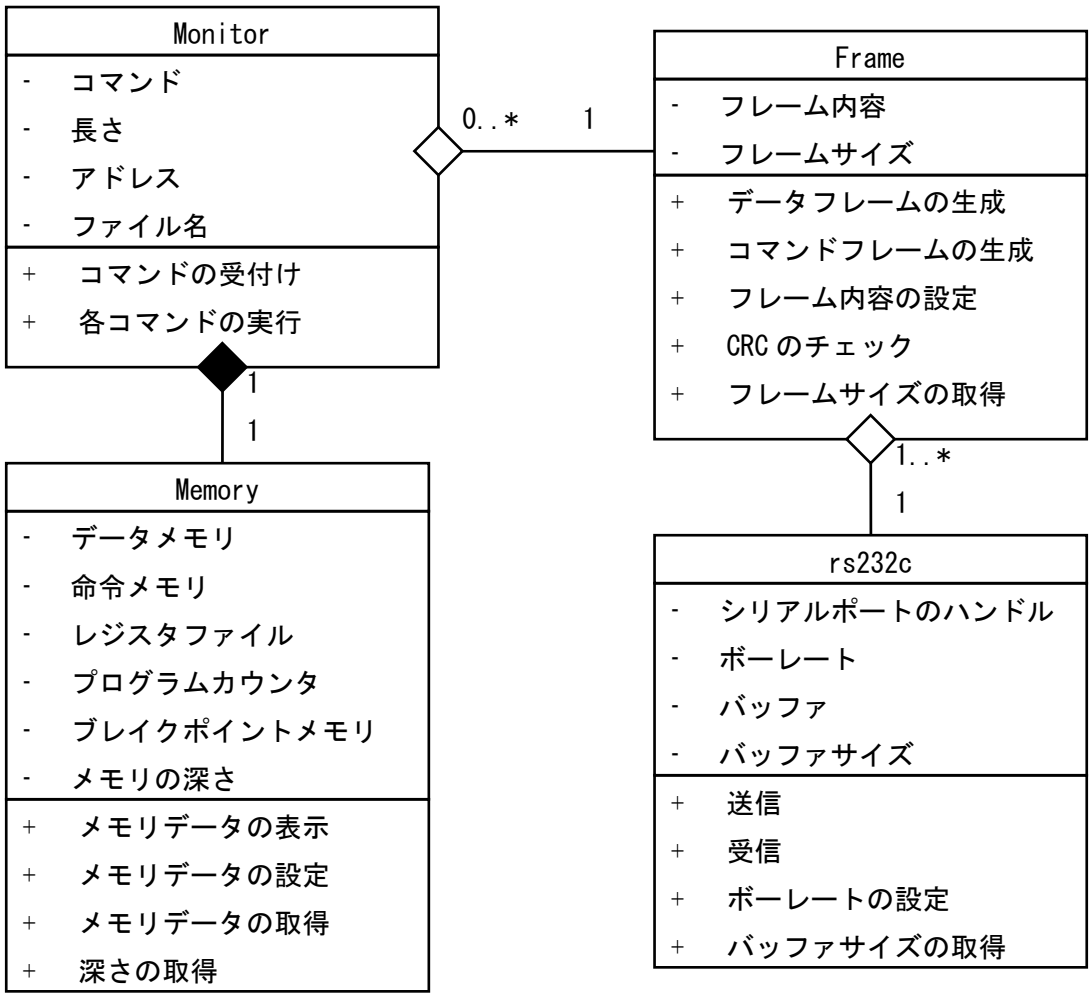

図 7　プロセッサモニタのクラス

プロセッサモニタの処理の流れは、1 つのユーザコマンドに対し 1 つ以上のアプリケーションコマンドの発行というスタイルで行う。データメモリの受信コマンドを例にとり、プロセッサデバッガのデータの通信手順を図 8に示す。まず、プロセッサモニタ上でユーザのコマンドを受け付け解釈する。受信コマンドの場合は、受信用のバッファを生成し、データの受信に備える。次に、プロセッサデバッガにデータの読み出し要求を送信する。要求はコマンドフレームにパッキングされたアプリケーションコマンドによって伝えられる。

プロセッサデバッガは受信したアプリケーションコマンドを実行し、連続して内部のデータを送信する。プロセッサモニタでは、これらのデータを一時的な受信バッファに保管し、全データ受信後にデータの抽出を行う。受信データにはヘッダやCRC等の情報も含まれているので、一度Frameオブジェクトに代入し、メンバ関数によって生データを読み出す。

データメモリの読み出し以外のコマンドも同様に、プロセッサモニタがアプリケーションコマンドを送信することで実行が可能である。

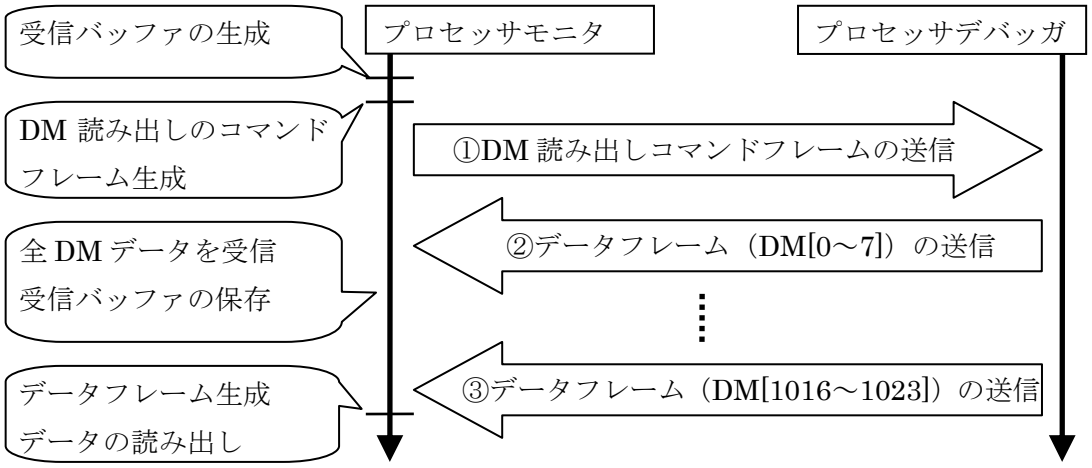

図 8　プロセッサモニタの通信手順

3.4　プロセッサデバッガの設計

3.4.1 プロセッサデバッガの概要

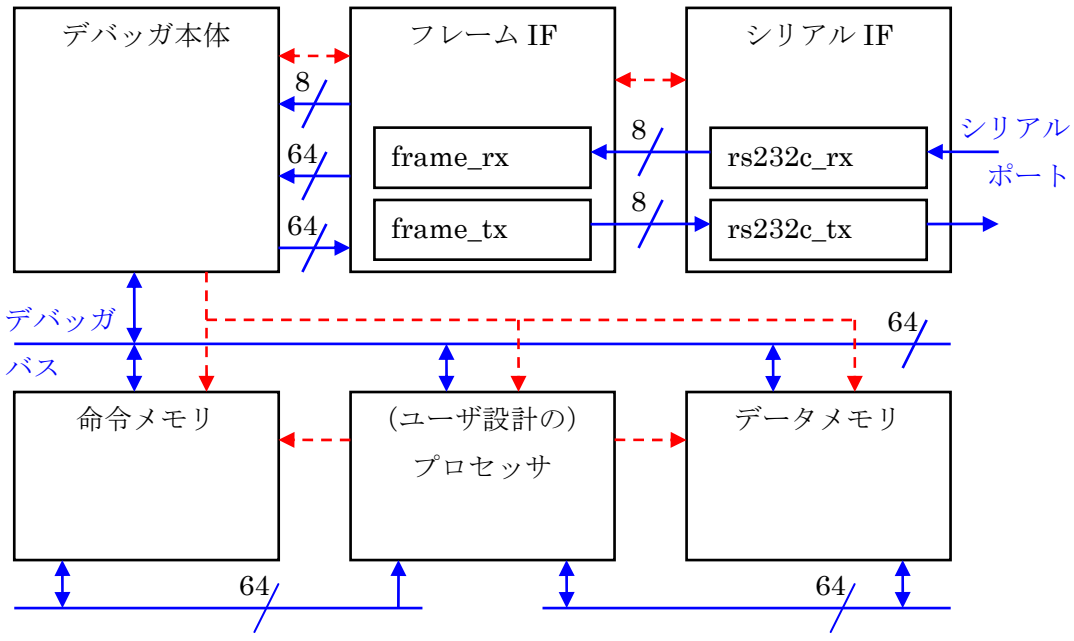

図 9　プロセッサデバッガの構成

プロセッサデバッガは、HSCSを利用して設計したプロセッサを、実機上で動作検証するためのハードウェアIPである。本モジュールと接続して論理合成をかけることで様々なプロセッサがFPGAに実装でき、ホストPCと通信することで実行が可能である。本モジュールが担当する処理は、ホストPC上のプロセッサモニタから要求される様々なデバッグコマンドをFPGA内部で処理し、必要に応じてデータの送受信を行うことである。図 9にプロセッサデバッガの構成を示す。本ツールは、シリアル通信を行うシリアルIF、フレームの生成・解釈を行うフレームIF、デバッグを行うデバッガ本体、命令・データメモリと、FPGA上で動作検証したいユーザ設計のプロセッサによって構成される。

3.4.2 設計思想

プロセッサデバッガの利点は、FPGA ボードの LED やスイッチを使用せずパソコン操作によって実機のデバッグを進められることである。この目的を達成するため、本モジュールの設計は次の点に重点を置いて行った。

(1) HSCS 上で学生が作成したプロセッサを FPGA ボードに搭載するために欠かせない周辺モジュールを提供すること
(2) FPGA ボード上の LED やスイッチを用いずに、ホスト PC からの指示によってプロセッサの検証を行えること
(3) FPGA やホスト PC の環境によって発生する問題ではなく、プロセッサに関わる問題に集中し、自作プロセッサの実装と動作検証に専念できること
(4) 命令セットシミュレータでは抽出できないハードウェアの特性によるバグを検出できること

以上の点をふまえて、設計するプロセッサデバッガの特徴を表 5に示す。

表 5　プロセッサデバッガの特徴

項目	特徴
利用目的	・周辺モジュールとして利用可 ・ハードウェアデバッガとして利用可
通信機能	・RS232C を用いたシリア通信 ・データ転送は 10 バイト(=1 フレーム)の単位で行う
メモリアーキテクチャ	・データメモリと命令メモリを分けたハーバードアーキテクチャを採用 ・データの入出力ポートの共有/個別の両方に対応 ・アクセスのタイミングはクロックの立ち上がり/立ち下がりで選択可 ・DCM でクロック速度を調節することで同期/非同期アクセスを選択可
システムバス	・デバッグ用バスとプロセッサ用バスを分離

プロセッサデバッガの役割の 1 つとして、FPGA 上でプロセッサを動作させるための環境整備がある。そもそも、プロセッサは単独では FPGA に搭載しても動作しないモジュールである。プロセッサを動作させるためには、チップの電源が投入されてから切断されるまでの間に、命令メモリとデータメモリに命令とデータを格納し、外部からの制御信号に

よってプロセッサを起動し（プログラムを実行し）、実行終了後に結果をメモリから読み出す必要がある。しかし、これら一連の流れを制御する周辺モジュールを、プロセッサを作成した学習者本人が用意することは難しく、学習の本質を突くものではない。また、それら周辺モジュールは、様々な仕様のプロセッサを実装する上でも共通する部分が多いことなどから、プロセッサデバッガでは、プロセッサを FPGA 上で動作させるために必要なモジュールとして、命令メモリ・データメモリ・システムシーケンサ・DMA コントローラ・バスコントローラを組み込み、プロセッサを検証するための周辺モジュールとして提供することにした。

プロセッサデバッガのもう一つの役割として、実機上で動作するデバッガがある。ハードウェア設計では命令セットシミュレータ等で正常な実行結果が得られていたとしても、チップ上では動作しない現象がしばしば発生する。その主な原因は、レジスタ間に複雑な演算が挿入されることで動作周波数が低下し、システムのクロック周波数を下回るためであるが、これを実機上で確認するためには専用のデバッガをハードウェア設計者自身が作成し、レジスタを観測しなければならなかった。そこでプロセッサデバッガでは、ほとんどのプロセッサが搭載している汎用レジスタを観測する機能を搭載し、実行途中のプロセッサの振る舞いを追跡しながら検証できるよう工夫を盛り込んだ。プロセッサデバッガは実行中のプロセッサの PC（Program Counter）の値を常に観測し、予め設定されていたブレイクポイントに達するとプロセッサを停止させることができる。このときに、内部のレジスタにデバッグ用のポートを接続しておけば、プロセッサデバッガを通してレジスタの観測・更新が可能である。

プロセッサデバッガは、できるだけ対象となる FPGA ボードやホスト PC を限定しないよう、通信には一般的に広く使われているシリアル通信の規格（RS232C / UART）を利用する。また、内部のハードウェアアーキテクチャを簡単化するため、通信で使用するコマンドやデータはすべて 10 バイト（＝1 フレーム）単位とする。

本ツールでは、データメモリと命令メモリを分離するハーバードアーキテクチャを採用する。また、各メモリのデータポートは入力と出力で別のものを用意した。同時に、読み書きで同じポートを使用するプロセッサには、書き込みイネーブル信号によって入出力を切り替える共有ポートも提供している。メモリアクセスのためのクロックエッジは、合成前に Verilog-HDL のソースの一部を変更することで簡単に切り替えられる。加えて、DCM（Digital Clock Manager：デジタルクロックマネージャ）によって、メモリに供給するクロックを制御することで、プロセッサと同期したデータの読み書きを行うかどうかが選択可能である。DCM によってプロセッサとは別の数十倍速いクロックをメモリに供給することで、仮想的にクロック同期しないメモリアクセスを可能にしている

本ツールの内部バスの構成では、デバッグ用に使用するデータバスと、プロセッサ実行時に使用するバスを分離する。この構成により、プロセッサデバッガのバスプロトコルをユーザが設計するプロセッサに適応しなくてもよく、より自由度の高いオリジナルプロセ

ッサを搭載できる。

3.4.3 プロセッサデバッガで使用する技術

(1) RS232C / UART

RS232Cは送信用・受信用の信号線を 1 本ずつ使ってデータの送信を行うシリアル通信の規格である。データをビット単位で送信するため、図 10に示すプロトコルに準拠しなければならない。

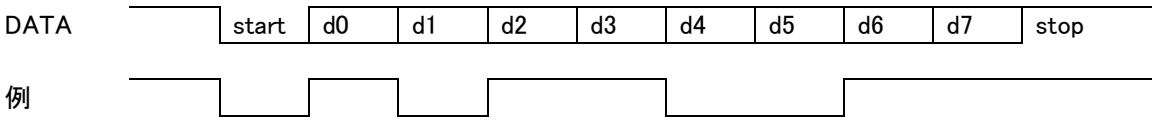

図 10　RS232C の通信プロトコル

RS232Cでは、スタートビットが 0、ストップビットが 1 である。通常、データが送信されない場合は常にストップビットがドライブされている。スタートビットが一度有効になると、その後から 8 ビット分が下位ビットから順にデータとして送信される。図 10の例では 0xCD（1100_1101）を送信している。RS232Cの規格では予め通信速度が定義されているが、本システムでは 119,200bpsを使用する。このパラメータはシリアルIFのインスタンス化時に指定できる。

(2) フレーム

プロセッサデバッガでは、何度も送受信するコマンドやデータを確実・単純・高速に行えるよう、10 バイトを 1 フレームとし通信を行う。フレームを通信の単位に採用することで、プロセッサデバッガの通信機構は非常に簡単な状態遷移機械で構成することができる。フレームの構成を図 11に示す。

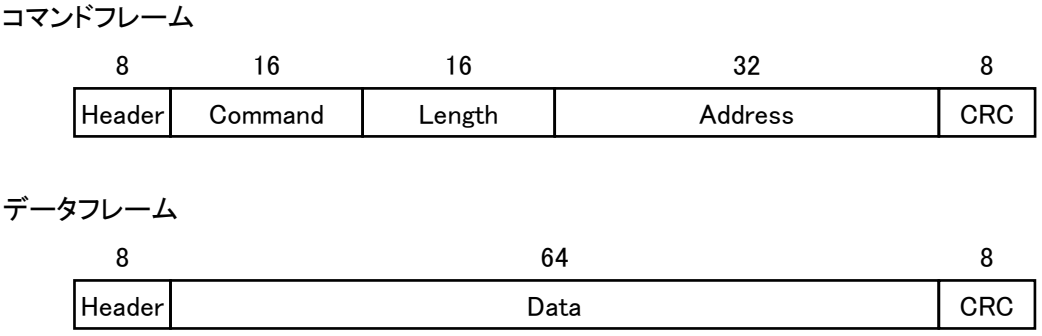

図 11　フレームの構成

フレームにはデータフレームとコマンドフレームの 2 種類がある。どちらのフレームも

先頭の 8 ビットはHeaderフィールドで、以下に続くビット列がコマンドなのかデータなのかを表している。ヘッダにはASCIIコードを用いる。現在使用しているコードはSTX（0x01）、SOH（0x02）であり、前者がデータを、後者がコマンドを表すヘッダとする。また、末尾 8 ビットにはCRCフィールドがあり、HeaderとCRCを除くフィールドのバイト単位の加算結果が格納されている。CRCフィールドにより転送されたデータが誤っていないかを受信側でチェックすることができる。CRCの算出方法を図 12に示す。

- **コマンドフレームの場合**

```
CRC = Command[15:8] + Command[7:0] + Length[15:8] + Length[7:0]
    + Address[31:24] + Address[23:16] + Address[15:8] + Address[7:0]
```

- **データフレームの場合**

```
CRC = Data[63:56] + Data[55:48] + Data[47:40] + Data[39:32]
    + Data[31:24] + Data[23:16] + Data[15:8] +  Data[7:0]
```

図 12　フレームの CRC の算出方法

コマンドフレームには 1 つのアプリケーションコマンド（引数やオプションを含む）が格納されている。コマンドの種類は Command フィールドに格納されている。さらに、データ転送コマンド用として Length フィールドに転送データ長、Address フィールドに転送開始アドレスが格納されている。

データフレームには 64 ビットのデータを含むことができる。この 64 ビットは、プロセッサデバッガ内のバスにそのまま流されるデータである。プロセッサによっては 64 ビットのバスすべてを利用しないものもあるため、有効なデータは下位から詰める。それぞれのフィールドのバイトオーダ（エンディアン）はビッグエンディアン（値の上位バイトがフィールドの先頭）とし、シリアル IF での転送も上位バイトから順に行う。つまり、フレームの上位 9 バイト目から転送し、最下位の 0 バイト目が最後に転送される。

3.4.4 プロセッサの接続インタフェース

プロセッサデバッガにデバッグ対象のプロセッサを搭載するためには、図 13のインタフェースに準拠しなければならない。プロセッサデバッガで用意しているプロセッサの接続先は、データメモリ（DM）、命令メモリ（IM）、レジスタファイル（RF）、プログラムカウンタ（PC）である。このうち、DMとIMはプロセッサの外部との接続であり、必ず接続しなければならない。RFとPCはデバッグのために用いるポートであり、プロセッサモニタのデバッグコマンドを実行したい場合に接続する。

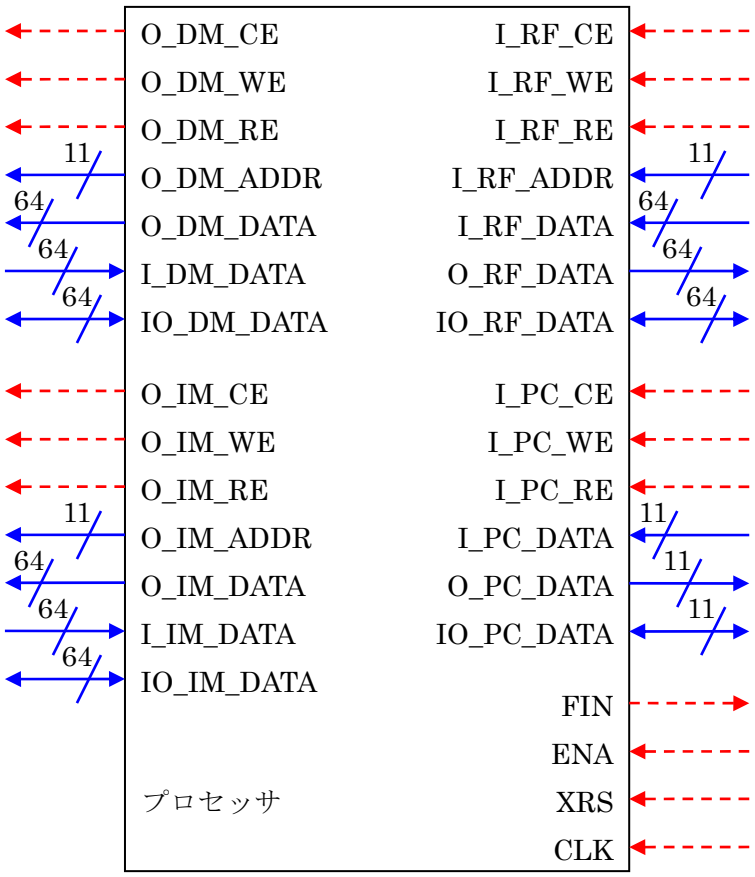

図 13　プロセッサデバッガに搭載するプロセッサのインタフェース

(1) IM / DM 関連

プロセッサデバッガでは、ハーバードアーキテクチャ（DM と IM のバスが別）のプロセッサを対象とするので、搭載するプロセッサには DM と IM それぞれの入出力ポートが必要である。DM と IM それぞれのバスにアクセスするにはチップイネーブル（I_xx_CE）、ライトイネーブル（I_xx_WE）、リードイネーブル（I_xx_RE）を制御しなければならない。これらは High アクティブの信号とし、データを読み出す際は、｛CE、WE、RE｝をそれぞれ｛1、0、1｝にする。データを書き込む場合は、｛CE、WE、RE｝をそれぞれ｛1、1、0｝にする。CE が 0 の場合はデータの読み書きは行えない。WE と RE を 1 にした場合は以前のデータが 1 クロックのみ読み出され､その後は書き込んだデータが読み出される。

基本的に、入力と出力で異なるポート（I_xxx_DATA、O_xxx_DATA）を使用するが、入出力で同じポートを共有するプロセッサにも対応している（IO_xxx_DATA）。その場合は、イネーブル信号の制御には注意する必要がある。IO_DM_DATA は読み出しの際、データの衝突を避けるためプロセッサからは z 値（はいインピーダンス）を出力する必要がある。データの幅は、様々なプロセッサに対応できるように最大で 64 ビットまで使用可能である。プロセッサが 16 ビットの場合は下位 16 ビットのみ使用する。

(2) RF / PC 関連

デバッグのためにプロセッサが停止中に内部のRFやPCにアクセス可能なポートを用意した。これらはENAが 0 のときにしか有効にならない。DMとIMへのバスアクセスと同様に、CE、WE、REが用意されているので、デバッガ本体からこれらがアサートされたときの動作内容を設計する。これにより、ホストPCからのコマンドによりデータの転送が可能になる。ここで注意しなければならないのが、バスのマスタがデバッガ本体である点である。このため、DM / IMとは入出力の方向が逆である。図 14に 8 列からなる 16 ビットのレジスタファイルをデバッグ用ポートに接続する例をVerilog-HDLの記述で示す。また、図15にこのときの配線図を示す。ENAが 0 かつ｛CE、WE、RE｝が｛1、0、1｝のときRF / PCの値をO_xxx_DATAに出力し、ENAが 0 かつ｛CE、WE、RE｝が｛1、1、0｝のときrf / pcにI_xxx_DATAのデータを代入する。

```
// RF に関するデバッグ用ポート
    input   [0:0] I_RF_CE;        // 動作更新イネーブル
    input   [0:0] I_RF_WE;        // 書き込みイネーブル
    input   [0:0] I_RF_RE;        // 読み出しイネーブル
    input  [10:0] I_RF_ADDR;      // アクセスアドレス
    input  [15:0] I_RF_DATA;      // 書き込みデータ
    output [15:0] O_RF_DATA;      // 読み込みデータ
    input   [0:0] ENA;            // MPU の動作許可信号

// MPU 内部の各モジュールから RF に接続されている線
    wire   [15:0] src_addr;
    wire   [15:0] src_data;
    wire   [15:0] dst_addr;
    wire   [15:0] dst_data;
    wire    [0:0] dst_we;

// 直接レジスタファイルに接続している線
    wire   [15:0] rs_addr;
    wire   [15:0] rs_data;
    wire   [15:0] rd_addr;
    wire   [15:0] rd_data;
    wire    [0:0] rd_we;

    assign src_data  = rs_data;
    assign O_RF_DATA = !ENA & I_RF_CE & ~I_RF_WE & I_RF_RE ? rs_data   : 15'dz;
    assign rs_addr   = !ENA & I_RF_CE & ~I_RF_WE & I_RF_RE ? I_RF_ADDR : src_addr;
    assign rd_data   = !ENA & I_RF_CE & I_RF_WE & ~I_RF_RE ? I_RF_DATA : dst_data;
    assign rd_addr   = !ENA & I_RF_CE & I_RF_WE & ~I_RF_RE ? I_RF_ADDR : dst_addr;
    assign rd_we     = !ENA & I_RF_CE & I_RF_WE & ~I_RF_RE ? 1         : dst_we;
```

図 14　レジスタファイルをデバッグポートに接続する記述

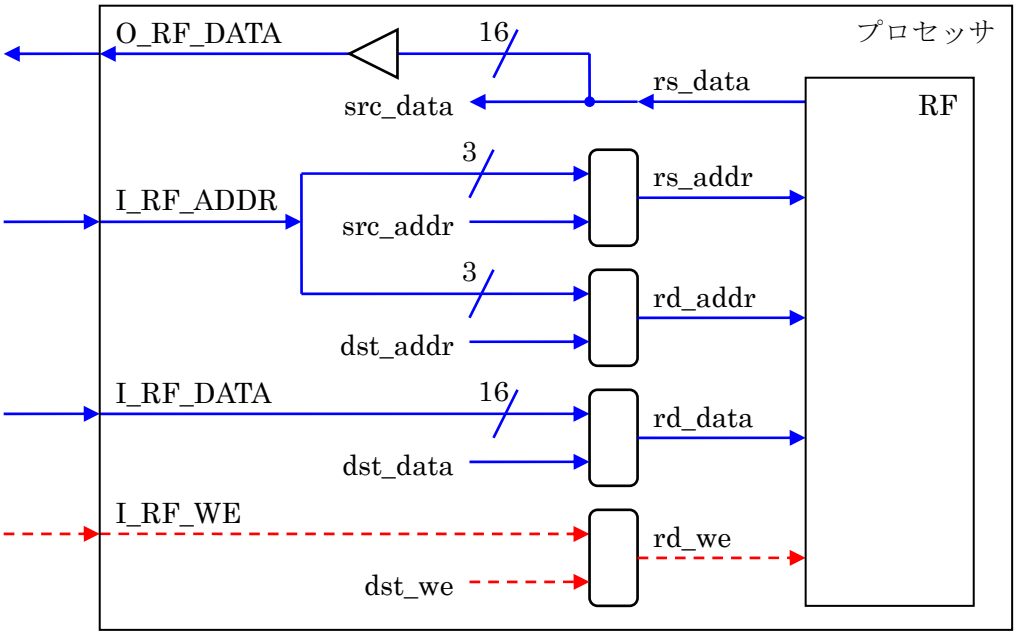

図 15　レジスタファイルをデバッグポートに接続する配線

(3)　その他

プロセッサを動作させるために、FIN、ENA、XRS、CLK の信号が用意されている。CLK はクロック、XRS は Low アクティブのリセットである。ENA は High アクティブの信号である、1 の場合のみプロセッサのレジスタが更新されるものとする。ENA が 0 の場合はプロセッサの動作は停止し、デバッガ本体からの指示で RF の読み書きを可能とする。FIN はプログラムの実行が終了したときにパルス（1 クロックのみ）で 1 となる信号である。プロセッサデバッガは FIN 信号が 1 になったのを確認し、状態遷移を行う。

4. プロセッサ設計支援ツールの実装

4.1 プロセッサモニタの実装

プロセッサモニタは、今後 GUI 環境への対応やプロセッサ設計支援ツールとの統合を見通し C++で実装を行った。また、開発環境として Micorsoft 社提供の Visual Studio 2005 を用いた。ただし、現在は MFC（Microsoft Foundation Class）は用いない。シリアルポートのアクセスには Windows の API を利用する。プロセッサモニタの実装では、段階的検証しながら行った。以下にプロセッサモニタの実装までの検証のステップを示す。

(1) ユーザインタフェースの実装

現在プロセッサモニタは、コマンドライン入力による CUI ベースのアプリケーションである。まず、ユーザが入力したコマンドを正しく解析する字句解析器を作成する。ユーザコマンドにはコマンド名といくつかの引数の組で入力され、コマンドによっては引数が省略されることがある。インタフェースの実装では、コマンドと引数の解釈を行い、これらが正しく認識されているかどうかを確認した。

(2) フレームデータの標準出力への出力

このステップでは、ユーザが入力したユーザコマンドがプロセッサデバッガで実行できるアプリケーションコマンドに変換されているかを確認する。アプリケーションコマンドの内容を確認するため、シリアルポートの代わりに標準出力に送信するフレームデータを書き出し、各フィールドの値や CRC の計算結果の検証を行った。

(3) RS232C ポートと FPGA 内のラッパモジュールによるデータ転送の実験

プロセッサモニタの実装で一番重要となるのが、シリアルポートへのデータの送信と受信である。ここでは、プロセッサモニタを実装する前にシリアルポートを利用して確実にデータの転送が行えるかをチェックする。チェックはホスト PC から送信する場合と、受信する場合の 2 通り行う。実験では、ホスト PC 上のファイルを開き、プロセッサモニタの RS232C モジュールを利用して送信するテストプログラムを組んだ。プロセッサモニタから FPGA へ送信する場合は、送信データをラッパモジュール内の SRAM にダンプし、送信完了後ボードのスイッチを操作して内容を確認した。プロセッサデバッガからデータを受信する場合は、受信データをホスト PC 上の標準出力に書き出すことで確認を行った。

(4) 仮想メモリのデータ転送

プロセッサモニタ全体を統合し、ユーザコマンドの検証を行う。主な検証項目は、ファイル内容のプロセッサモニタへのロード・セーブとプロセッサデバッガへの転送である。検証では、ホスト PC 上のファイルを用いて、データのロード→送信→受信→セーブを行い、実行前と実行後でファイルの内容が一致していることを確認する。

(5) プロセッサのデバッグ

最後に、デバッグコマンドを用いたプロセッサの検証を行う。検証を行ったユーザコマンドを表 6以下に示す。SOAR、MONIのプロセッサにおいてメモリへの読み書き、プロセッサの実行ができることを確認した。詳しいプロセッサの実装については5章で述べる。

表 6　検証を行ったユーザコマンド

コマンド	意味
send	全バッファを書き込み
read	全バッファにメモリ・レジスタ値を読み込み
load [ファイル]	全バッファにファイルの内容を読み込み
save [ファイル]	全バッファをファイルに保存
list	全バッファの表示
init	全バッファの初期化
run	最後まで実行
halt	プロセッサの停止
help	コマンド一覧の表示
exit	プロセッサモニタの終了

4.2 プロセッサデバッガの実装

4.2.1 対象FPGAボード

プロセッサデバッガの実装対象としてXillinx社のSpartan-3 Starter Kitボードを使用する[24]。図 16にSpartan-3 Starter Kitボードの構成を示す。Starter Kitボードには 20 万システムゲートのSpartan-3 FPGA①が搭載されている。その他にプロセッサデバッガの実装で使用するデバイスとして、2MBitのコンフィギレーション用PROM② 、2 つのSRAM④、外部との通信用のRS232 のシリアルポート⑥、4 つの 7 セグメントLED⑩、8 つのスライドスイッチ⑪、8 つのLED⑫、4 つのプッシュボタン⑬、50MHzのクロック発振器⑭などが搭載されている。

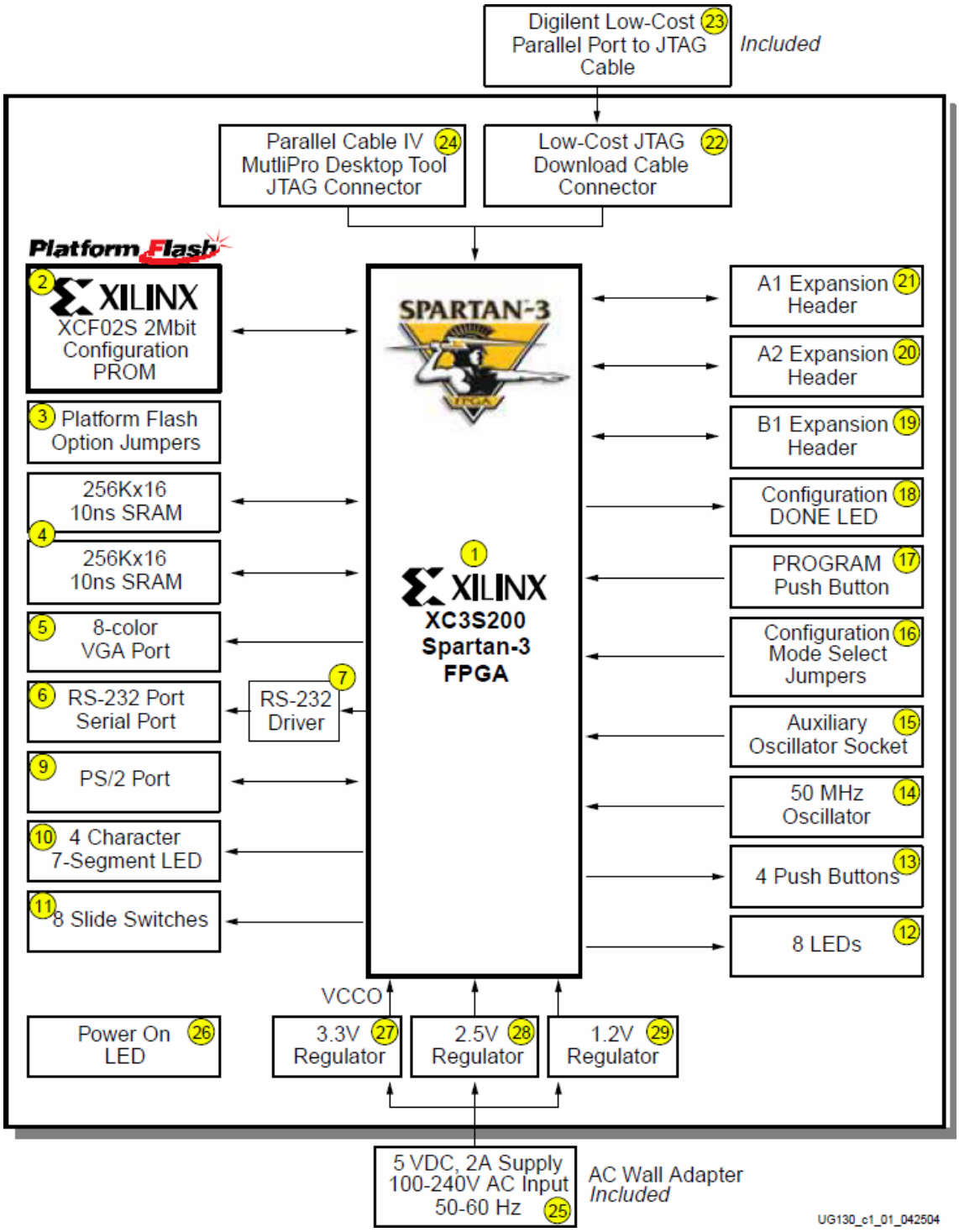

図 16　Spartan-3 Starter Kit ボードの構成

4.2.2 テスト手法

プロセッサデバッガは、FPGA上でプロセッサのデバッグを行えるモジュールであるが、自身の動作検証を行うには十分な構造ではない。そこで、プロセッサデバッガの実機上で

の検証を目的にラッパモジュールを作成した。図 17にラッパモジュールの構成を示す。本モジュールは、ステートコントローラ、外部SRAMアクセス用インタフェース、デジタルクロックマネージャ（DCM）、及びLEDコントローラを内包し、さらにFPGA外のSRAM、スライドスイッチ、7 セグメントLED、RS232 ポートを利用可能とした。本モジュールの実態はVerilog-HDLの記述とXILINX社が提供するIP、及びボード上の各デバイスである。ラッパモジュールの実装により、ホストPCとの通信データのダンプ、オンボードLEDへの内部状態の表示、ボード上のスライドスイッチを利用したデータの入力を可能とした。これらの機能を利用し、まず、プロセッサデバッガの動作検証を行う。その後、プロセッサデバッガと作成したプロセッサを接続してプロセッサの実装検証を行う。

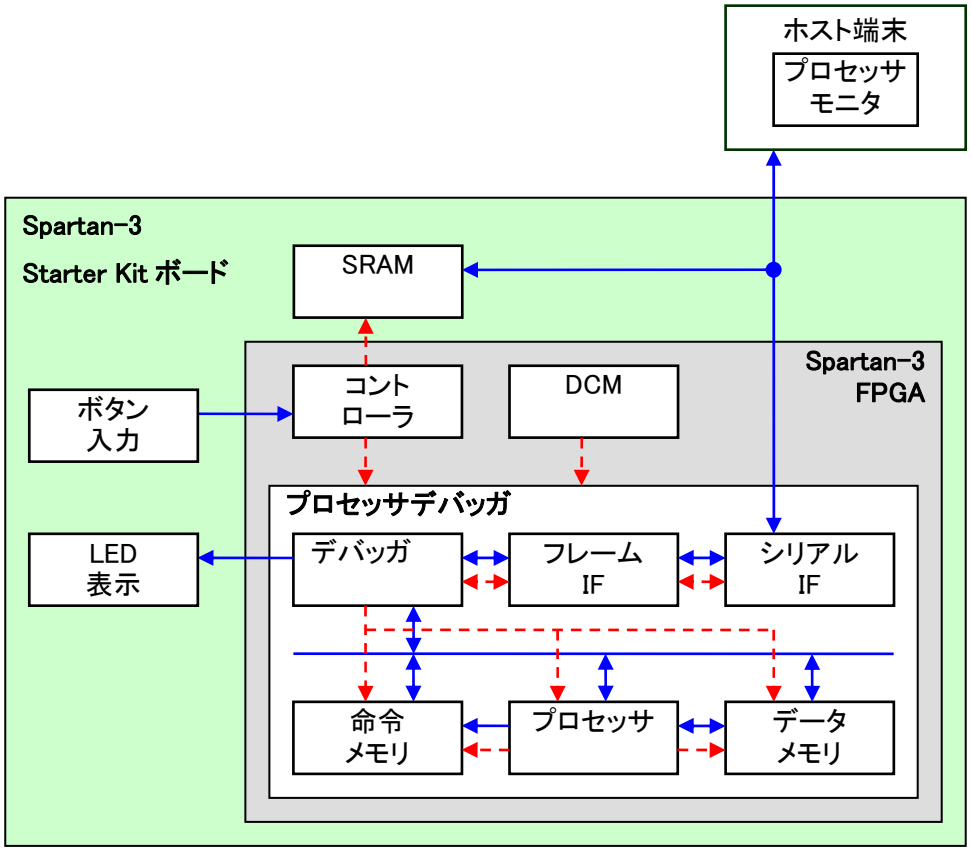

図 17　ラッパモジュールの構成

(1)　RS232C ポートによる通信の確立

まず、ホスト PC から送信されたシリアルデータが、FPGA 上で正しく受信できているかを確認する。シリアル IF・フレーム IF 間のバイトデータをオンボードの SRAM にダンプし、転送終了後 SRAM のデータを確認する手法で行った。次に、FPGA 上からデータが正しく送信されているかを確認する。常に固定のデータを送信するモジュールをシリアル IF に直結し、ホスト PC の端末で同じ値が受信できることを確認する。このとき、シリアル通

信で使用するボーレートは 119,200bps、FPGA ボード上で使用しているクロック周波数は 50MHz である。

当初、シリアルデータの送受信において大幅なデータ落ちが頻発していた。これは、ボードのクロック周波数がシリアル通信のボーレートで割り切れないことに起因する。受信データ 1 ビットをサンプリングする場合の使用クロック数が整数で割り切れなければ、サンプリングのタイミングに微妙なずれが生じ、これが累積することで 1 ビットを過不足して読み取ることになる。シリアル通信において 1 ビットを読み落とすことはデータの整合性を失うことに直結する。そこで、シリアルデータの受信回路にデータ開始の同期制御と雑音を除去する機構を追加した。スタートビットが入力されれば、すぐに受信状態に遷移し、サンプリングを行う。サンプリングは 1 ビットに対して 64 回行い、その相関値によって受信データのビットを決定するアーキテクチャに変更した。図 18に変更したシリアルIFの受信モジュールの構成を示す。受信ビットは 64 回ACCに加算され、1 ビットの受信が終わったとき、値が 32 よりも小さければ 0、32 以上なら 1 として扱う。これを 8 ビット分繰り返し、ストップビットを検出すれば、バイトデータとしてDOUTに出力する。受信機の動作イネーブルを通信レートの約 64 倍速く上げることで、FPGAボードボードのシステムクロックは変更せずに利用が可能である。この手法を用いることで、安定したデータの受信が可能になった。

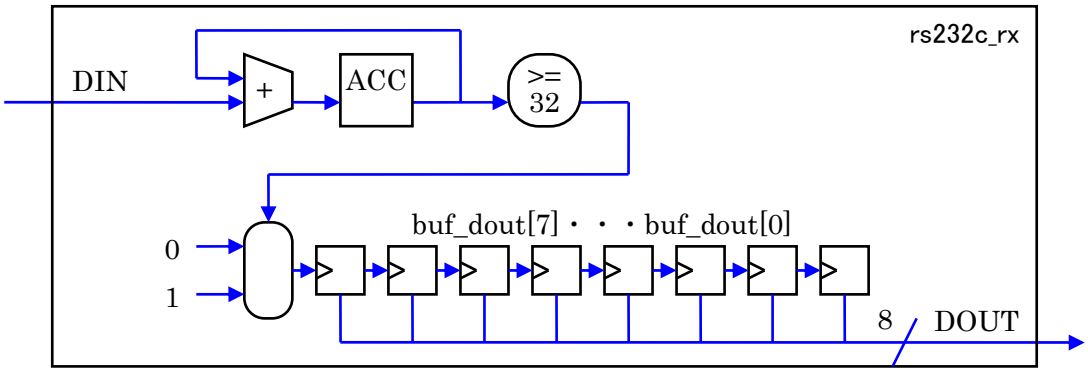

図 18　変更したシリアル IF の受信モジュールの構成

(2)　フレームの同期とデバッガの内部状態の確認

フレームの同期はフレーム IF で行う。フレーム同期がとれているかを確認するには、フレームとしてデータが正しいかを CRC によりチェックする機構を応用する。フレーム IF では CRC のチェックによって不正なフレームが検出された場合、そのデータはデバッガ本体には送信されず、フレーム IF 内で破棄される。一方デバッガ本体では、受信するフレームは状態遷移のトリガとして使用されている。以上から、デバッガ本体の内部状態を観測すればフレームの同期がとれていることが確認できる。フレーム同期の確認を行うため、ラッパモジュール上の 7 セグメント LED にデバッガの状態を表示し、転送データと状態の

遷移の仕方を照らし合わせることで検証を行った。

(3) プロセッサデバッガ内のデータメモリ・命令メモリへのデータの送受信

プロセッサデバッガ内のメモリへの転送の検証は、データ転送のコマンドを用いて行った。データの送信コマンドと転送データをホスト PC よりプロセッサデバッガに送信し、何も処理せずに、受信コマンドを与える。この結果送信時と同じデータが受信できることを確認した。

(4) プロセッサの実行と実行結果の検証

プロセッサを内部バスに接続し、実装を行ったのち、ホスト PC 上からのコマンドによって、データメモリへの書き込み、命令メモリへの書き込み、プロセッサの実行、データメモリの読み出しを順に行う。こととときブレイクポイントの設定などは行わない。プロセッサの実行後にデータメモリの値を確認し、所望の実行結果が得られているかを確認することでプロセッサの実装と動作検証が行えた。

(5) デバッグコマンドの検証

最後に、本研究の目的であるデバッグコマンドの実行を行う。テスト内容は主に 2 点ある。ブレイクポイントや命令ごとでのプロセッサの一時停止と、プロセッサ内部のレジスタの観測・更新である。これらは、プロセッサデバッガ上で N までの和やバブルソートなど簡単なプログラムの実行が確認されたプロセッサを対象に、他のプログラムをデバッグすることで行った。

4.3 実装規模

プロセッサデバッガをFPGA上に実装するために、SOARプロセッサを接続し論理合成をかけた。論理合成ツールとしてはXILINX社提供の総合開発環境ISE8.2iのXSTを用いる。プロセッサデバッガの合成結果を表 7に示す。LUT数は 3,840 中 3,840 使用し、FPGAの79%を使用した。また、最高動作周波数は、49.8MHzであった。

表 7　プロセッサデバッガの実装規模

項目	スライス	Flip Flop	LUT	LUT 使用率	最高動作周波数
使用数 / 総数	1,721 / 1920	1,221 / 3,840	3,069 / 3,840	79%	49.8MHz

5. プロセッサ設計によるハード/ソフト協調学習システムの評価

5.1 研究室におけるハード/ソフト協調学習システムを用いた学習

本研究室では、4 回生を対象に HSCS を利用した導入教育を行っている。始めに、MONI シミュレータによるアセンブリプログラミング学習を行う。ここで、計算機アーキテクチャに興味の沸いた学生には、HSCS を用いて卒業研究でプロセッサの実装までを行ってもらう。2006 年度の学習では、4 名の学生がアセンブリプログラミングに挑戦し、そのうち 2 名がプロセッサ設計を行った。そのうち 1 名は MONI 命令セットを用いて数種類のアーキテクチャを FPGA 上に実装した。また、もう 1 名も ARM ライクなオリジナル命令セットを設計し、HDL シミュレータ上で検証を行った。

5.2 MONIアセンブリプログラミング

2006 年度の研究室でのHSCSを利用した学習では、まず、学習者に自由に問題を設定してもらい、そのアセンブリプログラミングを行う課題を課した。アセンブリプログラミングはMONIシミュレータを用いて動作確認を行い、複数のアーキテクチャを選択して実行を行う。本学習の参考資料として、本学理工学部情報学科のボードコンピュータの実験資料を与えた。この課題によって作成されたMONIのプログラムと学生別の学習時間を表 8に示す。

表 8　MONI アセンブリプログラミングによる成果

学生	学習時間(時間)	プログラム
A	25	・整数の階乗 ・三角形の種類判定 ・二次方程式の根の判別. ・素数判定 ・符号付き整数の商と剰余 ・一次方程式直線 ・8×8 行列引き算/掛け算/足し算 ・BCD コード加算/減算
B	35	・素数判定
C	42	・素数判定(エラトステネスの篩い) ・線形合同法による乱数生成
D	12	・整数の階乗 ・三角形の種類判定 ・二次方程式の根の判別 ・素数判定 ・符号付き整数の商と剰余 ・平方根(ニュートン法) ・バブルソート ・挿入ソート ・選択ソート
平均	28	5.75 問

4 人の学生が学習した結果、平均して 28 時間で 5.75 問が作成された。作成されたプログ

ラムは、実験資料を参考にしながらも、学生自身が問題を考えコーディングした結果である。ほとんどのプログラムが資料の課題テーマと同じあったが、中には学生 C の様に同じ素数判定でも、資料にはないアルゴリズムで実現する学生もおり、シミュレータを用いたアセンブリプログラミングが効果的に利用された。また、プログラムによっては 100 行以上のプログラムを組む学生もいた。さらに、複数のソートプログラムを設計し、シミュレータの表示アーキテクチャを切り替えて、実行サイクル数などの性能比較を繰り返し行い、プログラムとアーキテクチャの考察を行う学生もいた。MONI アセンブリプログラミングを行った 4 人の学生のうち、2 人はプロセッサ技術に興味を持ち、プロセッサ設計まで行うことになった。後述する MONI プロセッサと ARM ライクプロセッサの設計を行った。

5.3 SOARプロセッサの実装

SOARプロセッサは、HSCSを利用して作成したプロセッサの 1 例であり、4 命令形式、25 命令からなる 16 ビットのシングルプロセッサである。SOARは、2004 年度に作成された正常動作を確認できているプロセッサであり、本研究では、プロセッサデバッガの検証、及び2006年度のプロセッサ設計例との比較に利用する。SOARの構成を図19に示す。SOARの特徴は、状態レジスタを用いた分岐命令を採用していることである。演算命令ごとに結果の状態をSR（Status Register：状態レジスタ）に保持し、ジャンプ命令で比較することで 8 条件での分岐が可能である。

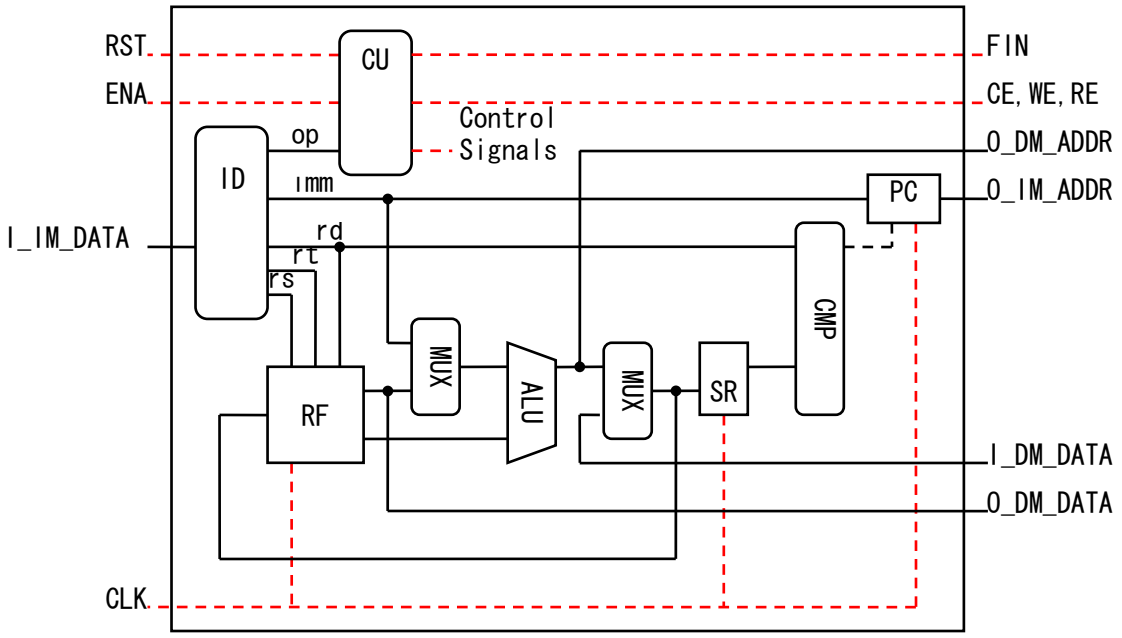

図 19　SOAR プロセッサの構成

SOAR プロセッサを用いることで、プロセッサデバッガの基本的なコマンドの検証を行

うことができた。データメモリ、命令メモリの読み書きに加え、シェルソート、挿入ソート、選択ソート、及び累乗計算のプログラムにおいて、ブレイクポイントを設定した実行とレジスタの読み書きを行った。また、実機上で無限ループに入ったプロセッサを強制終了することも確認した。これにより、FPGA ボードのスイッチや LED を駆使せず、ホスト PC 上で操作できるデバッグ環境を提供できたことを示せた。

次に、SOARの設計時間について考察する。SOARの実装はプロセッサ設計支援ツールを導入する以前のHSCSで行った。当時の設計時間を表 9に示す。SOARはHSCSにはないオリジナルプロセッサであったため、命令セット設計とHDLによるプロセッサ設計を合わせると 50 時間程度かかっている。また、設計したプロセッサの実装には、FPGA上への実装環境の構築と動作検証を合わせて 100 時間程度かかった。一方、HSCSの実装環境を利用してMONIの代わりに実装を行うと、インタフェースの調整などを含めて 5 時間程度で実機検証を行えた。

表 9　SOAR プロセッサの設計時間

工程	設計時間（時間）
命令セット設計	10
HDL によるプロセッサ設計	40
FPGA 実装環境の構築	40
FPGA 上での動作検証	60
HSCS を用いた FPGA 上への実装	5

5.4　MONIプロセッサの実装

2006 年度の学習では、MONIプロセッサを複数のアーキテクチャで実現する例があった[6]。表 8で示すAの学生が設計を行った。設計されたアーキテクチャは、シングルサイクル、3 段マルチサイクル、及び学生が自身で拡張した 4 段マルチサイクルである。また、それぞれのアーキテクチャにおいて、メモリアクセスで、クロック非同期、クロック同期、1 クロックの遅延有りのプロセッサを設計している。図 20にはこのうちMONIシングルサイクルプロセッサの構成を示す。また、表 10にMONIの設計時間を示す。

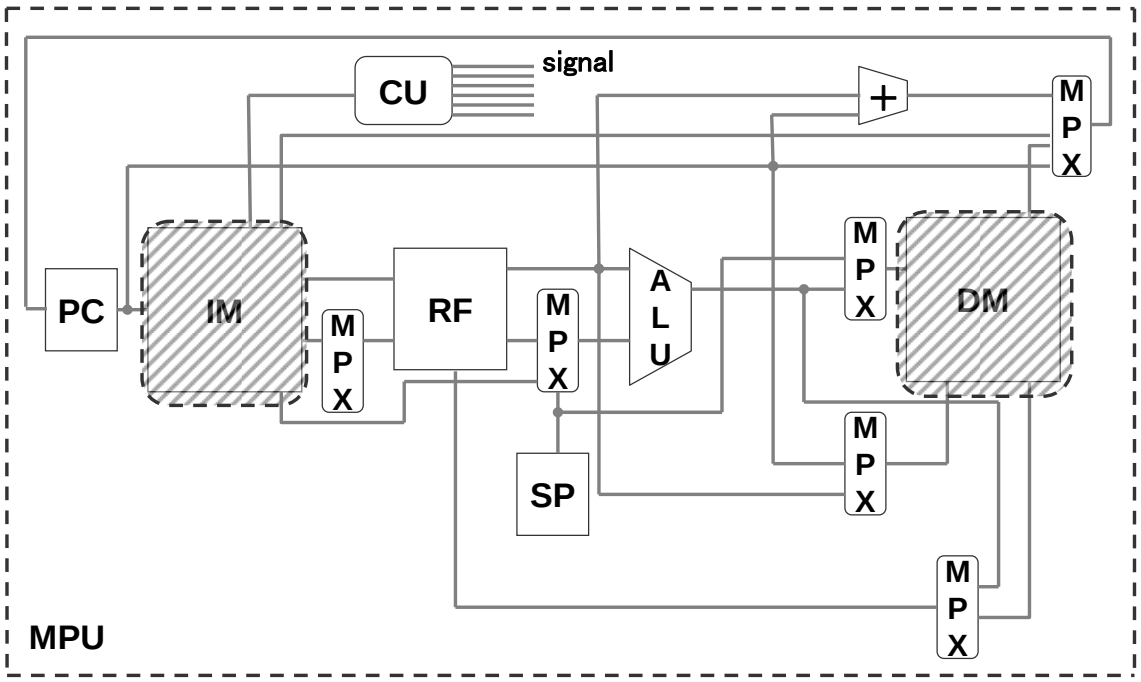

図 20　MONI シングルプロセッサの構成

表 10　MONI プロセッサの設計時間

メモリアクセス	クロック非同期			クロック同期			1 クロック遅延		
プロセッサアーキテクチャ	シングル	3 段	4 段	シングル	3 段	4 段	シングル	3 段	4 段
HDL によるプロセッサ設計時間（時間）	24	27	9	27	27	11	26	29	11

MONI は HSCS の基本命令セットで定義している命令セットであり、また、ソフトウェア学習において命令セットシミュレータ VAPS を利用することで内部のデータパスを確認できることから、HDL による設計では多くて 29 時間と比較的少ない時間で設計できている。アーキテクチャ別に見ると、3 段マルチサイクルの設計が最も時間がかかっており、シングルサイクルには無いフェーズの制御の設計が要因として考えられる。一方 4 段のマルチサイクルの設計では、3 段マルチサイクルの応用によって行ったことから、半分近く設計時間で達成できている。メモリアクセスの違い別で見てみると、アーキテクチャごとに設計時間の大きな差は見られなかった。これは、各メモリアクセスにおいて、シングルサイクル、3 段マルチサイクル、4 段マルチサイクルを設計したことから、同様の HDL コーディングスキルで行えたことが要因としてある。

MONIプロセッサは、プロセッサ設計支援ツールを用いてSpartan-3 Starter Kitボード上に実装まで行った。MONIプロセッサの実装規模を表 11に示す。メモリアクセスの違いに

よる設計規模の差異は小さく、ほぼ変わらない結果となった。最高動作周波数に着目すると、シングルサイクルでは約 30MHz、マルチサイクルでは約 43MHzと大きなスピードアップを実現した。マルチサイクルでは 3 段よりも 4 段の方が 1MHz程度速い結果が得られた。

表 11　MONI プロセッサの設計時間

メモリアクセス	クロック非同期			クロック同期			1 クロック遅延		
プロセッサ アーキテクチャ	シングル	3 段	4 段	シングル	3 段	4 段	シングル	3 段	4 段
スライス数	720	899	824	726	919	882	700	913	882
FPGA 使用率 （%）	37	46	42	37	47	45	36	47	45
最高動作周波数 （MHz）	31.1	43.7	44.4	31.1	43.5	45.0	32.7	43.5	45.2

5.5 ARMライクプロセッサの設計

HSCSの実用例として、ARMライクプロセッサの設計を示す[7]。ARMライクプロセッサは、ARM社が提供するARM命令セットを参考に、本研究室の学生が作成したオリジナルプロセッサである。図 21にARMライクプロセッサの構成を示す。ARMライクプロセッサでは 4 段のマルチサイクルプロセッサである。全命令において、条件実行が可能な構成である。ARMライクプロセッサの設計は、HDLによる設計までを行った。

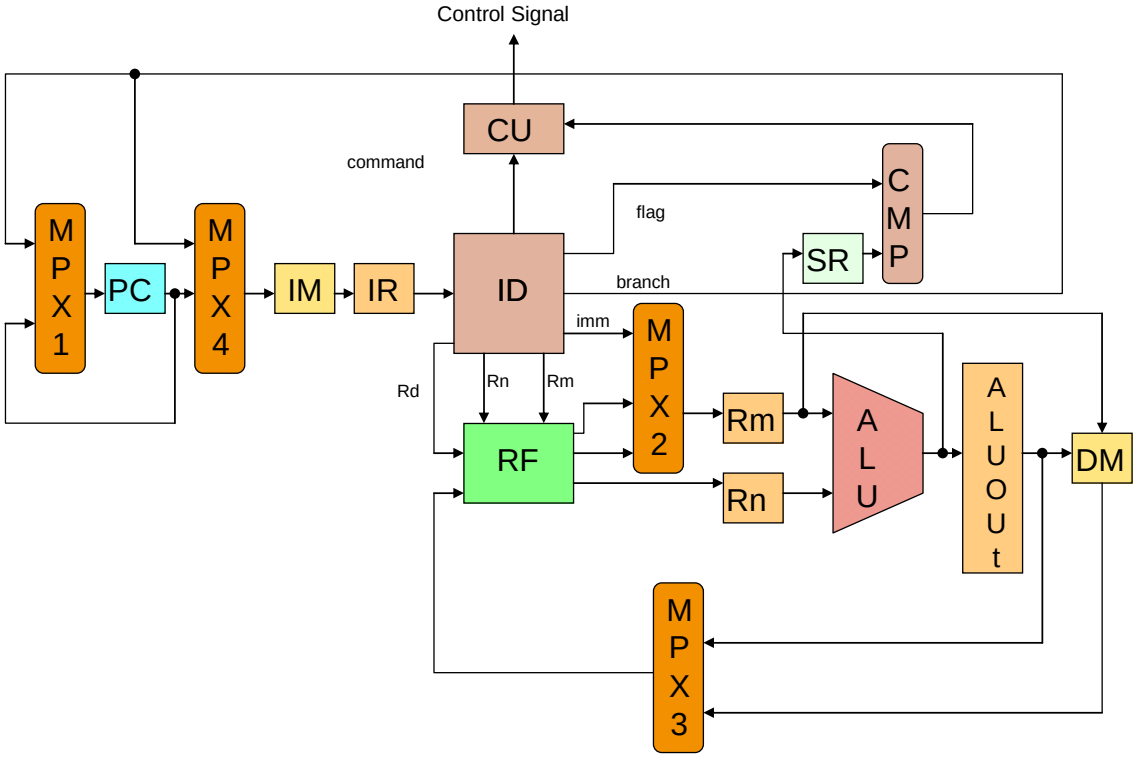

図 21　ARM ライクプロセッサの構成

5.6　システムの評価と考察

HSCS 上でプロセッサ設計支援ツールを利用したことで、複数のプロセッサを FPGA 上に実装することができた。本ツールを導入する以前に実装した SOAR プロセッサでは、FPGA 上で動作検証を行うまでに、約 150 時間かかっていたが、本ツールを用いて作成した MONI では、30 時間以下であり、FPGA 実装のための周辺モジュールを提供するという目的は達成できたといえる。また、FPGA 実装において、様々なデバッグコマンドを実行することで、ボードのスイッチや LED を使用することなく、実機検証することも SOAR プロセッサを実装することで検証することができた。今後は、学生が作成したプロセッサにおいて、同様にデバッグコマンドを実行し、実機上での検証を行えることを示すことが課題である。

HSCS の評価という観点では、従来の MONI のアセンブリプログラミングとプロセッサ設計の学習フローにおいて FPGA ボード上への実装までを行う設計例が出、評価できると考える。これは、アセンブリプログラミングにおいて、比較的容易な問題を多く設計し、MONI 命令セットを習熟していたことが大きな点である。また、MONI シミュレータ VAPS の上で示したアーキテクチャの図によって、回路イメージが持てたことも要因にあると思われる。一方、ARM ライクプロセッサの設計では、アセンブリプログラミングの評価を行える環境が不十分であったことから、HDL によるプロセッサ設計では難航する結果となった。しかし、MONI とは異なる命令セットを考案し試作できたことは、命令セット設計と

プロセッサ実装の学習フローを取り込んだ本システムの有用性を示す結果であり、今後、プロセッサ設計支援ツールを用いた FPGA 上への実装と検証に大いに活用できると期待できる。

6. おわりに

本研究では、ハードウェアとソフトウェアの関係を理解するツールとして、令セット設計とプロセッサ実装の学習フローを提案し、プロセッサ設計支援ツールの開発を行った。すなわち、命令セット定義ツール、汎用アセンブラ、プロセッサモニタ、及びプロセッサデバッガをハード/ソフト協調学習システムに導入した。

プロセッサデバッガ・モニタ設計では、フレーム単位の転送とシリアルポートを利用した通信プロトコルを実現し、FPGA 上のプロセッサをホスト PC 上からデバッグできる環境を整えた。主なデバッグコマンドとして、メモリ・レジスタへのデータの読み書き、プロセッサの実行と停止、ブレイクポイントの設定機能を実装し、ホスト PC 上のプロセッサモニタを利用することで、ボードのスイッチや LED の入出力を用いることなく実機上のプロセッサのデバッグを行うことを可能にした。本ツールを用いることで、研究室の 4 回生によって複数のアーキテクチャのプロセッサを実機上に実装し、その動作検証を行うことができた。研究室での学習を通して、16 ビットの SOAR プロセッサ、9 種類の MONI プロセッサの実装を行い、複数のプログラムの実行を行った。また、ARM ライクプロセッサの設計についても試作され、ハード/ソフト協調学習システムを用いたプロセッサ設計の実例を示した。

従来のハード/ソフト協調学習システムでは FPGA 上への実装までに 150 時間所要であったが、プロセッサ設計支援ツールを利用してプロセッサ設計を行った結果、本システムの基本命令セットである MONI プロセッサの実装では、9 種類のアーキテクチャにおいて、30 時間以下に短縮できた。これにより、ハード/ソフト協調学習システムにおける学習者の負担が減り、プロセッサ技術の学習に注力できる環境が整ったことを示した。また、今年度の研究室での学習例では、MONI 以外のプロセッサとして ARM ライク命令セットの設計と実装に挑戦した学生もおり、命令セット設計を取り入れたハード/ソフト協調学習システムとしては、その利用価値を評価することができた。

今後の課題として、MONI 以外の命令セット設計の評価環境として、汎用シミュレータの設計、最適化コンパイラの開発を行わなければならない。繰り返し、アセンブリプログラミングと命令セット設計を行える環境を構築し、学生がプロセッサにおけるハードウェアとソフトウェアの境目の技術について習得できる教育システムの実現が課題である。

謝辞

本研究の機会を与えてくださり、ご指導いただきました山崎勝弘教授、小柳滋教授に深く感謝します。また、本研究に関して貴重な助言、ご意見をいただきました、高性能計算研究室の皆様に心より感謝いたします。

最後に、本研究において多くのご協力をいただきました榎本氏、志水氏に深く感謝いたします。

参考文献

[1] 田中康一郎、小羽田哲宏、久我守弘、末吉敏則：教育用プロセッサ KITE とその開発支援環境,情報処理学会研究報告，Vol.93, No.49(ARC-100), pp.59-66，1993.

[2] 末吉敏則、小羽田哲宏、野崎貴弘、田中康一郎、久我守弘：FPGA を利用した教育用プロセッサ KITE-2 システムソフトウェア教育への対応情報処理学会研究報告, Vol.94, No.50(ARC-106), pp.25-32，1994.

[3] 重村哲至、守川和夫、力規晃、新田貴之、原田耕治、山田健仁：教育用マイコンボードを用いた HDL 演習環境の実現, 情報処理学会 研究報告, Vol.2005 No.15, pp.43-48, 2005.2.

[4] 加藤久晴，内田順平，宮岡祐一郎，戸川望，柳澤政生，大附辰夫，Packed SIMD 型命令を持つプロセッサ合成システムのためのリターゲッタブルコンパイラ， 電子情報通信学会技術研究報告, VLD2003-157, pp.41-46, 2004.3.

[5] 高橋隆一、大岩元：FPGA を用いたスーパースカラ設計教育に関する一考察，情処研報 コンピュータと教育, 2003-CE-69，2003.

[6] 志水建太：ハード/ソフト協調学習システム上でのプロセッサ設計とプロセッサデバッガによる検証，立命館大学理工学部卒業論文，2007.

[7] 榎本雄太 : ARM ライクプロセッサの設計と FPGA ボード上での検証，立命館大学理工学部卒業論文，2007.

[8] 中村浩一郎：命令定義可能なハード/ソフト・コラーニングシステム上でのプロセッサデバッガの設計と実装，立命館大学理工学研究科修士論文，2006.

[9] 池田修久：ハード/ソフト･コラーニングシステム上での FPGA ボードコンピュータの設計と実装，立命館大学理工学研究科修士論文，2004.

[10] 大八木睦：ハード/ソフト・コラーニングシステム上でのアーキテクチャ可変なプロセッサシミュレータの設計と試作，立命館大学理工学研究科修士論文，2004.

[11] 難波翔一朗、中村浩一郎、Hoang Anh Tuan、山崎勝弘、小柳滋：FPGA を用いたハード／ソフト・コラーニングシステムの開発，第 8 回 LSI IP デザイン・アワード，2006.5.18.

[12] 難波翔一朗、中村浩一郎、Hoang Anh Tuan、山崎勝弘、小柳滋：ハード/ソフト協調学習のための命令セット定義ツールとプロセッサデバッガの開発，FIT2006, N-009, 2006.9.5.

[13] 富樫望：命令セット定義可能な汎用アセンブラの設計と実装，立命館大学理工学部卒業論文，2006.

[14] 中川裕樹：ハード/ソフト協調学習のための汎用アセンブラのユーザインターフェースの設計と実装，立命館大学理工学部卒業論文，2006.

[15] 西田範行： ハード・ソフト協調学習のための汎用シミュレータの設計，立命館大学理工学部卒業論文，2006.

[16] John L. Hennessy, David A. Patterson 著，成田 光彰 訳： コンピュータの構成と設計（上）（下），日経 BP 社，1999.
[17] 株式会社 半導体理工学研究センター 開発第二部 IP 技術開発室 著，RTL スタイルガイド Verilog-HDL 編 第 4 刷発行，2005.8.15.
[18] 中森章 著，マイクロプロセッサ・アーキテクチャ入門，2004.4.1.
[19] 小林優 著，改訂・入門 Verilog HDL 記述―ハードウェア記述言語の速習&実践，CQ 出版，2004.
[20] 社会人向け VLSI 設計セミナー レクチャー用マニュアル，立命館大学 VLSI センター，2003.
[21] UART 非同期（調歩同期式）通信，http://www11.ocn.ne.jp/~akibow/AVR/GSN/UART.html，2006.
[22] なひたふ・電子回路の世界へようこそ，http://www.nahitech.com/nahitafu/，2006.
[23] 組み込みネット，http://www.kumikomi.net/index.html，2006.12.
[24] SPARTAN-3 STARTER KIT ボードマニュアル，XILINX，2006.
[25] UniPhier®， http://www.mec.panasonic.co.jp/micom/MicomFamily/uniphier.html，松下電器産業株式会社，2006.
[26] platformOViA ホーム，http://www.necel.com/platformovia/ja/index.html，NECエレクトロニクス株式会社ホームページ，2006.