卒業論文

ARM ライクプロセッサの設計と FPGA ボードへの実装の検討

氏名：榎本雄太
学籍番号：2210030055-2
指導教員：山崎勝弘教授
提出日：2007 年 2 月 19 日
立命館大学 理工学部 情報学科

内容梗概

本論文では、ハードウェア記述言語の Verilog HDL によるオリジナルのマイクロプロセッサの設計、及びハード/ソフト協調学習システムを用いての FPGA ボードコンピュータへの実装の検討を行った。設計対象のアーキテクチャには ARM プロセッサに用いられている命令セットからいくつかを用いている。また、作成したプロセッサは ARM の命令セットの使用を目標としていることから、ARM ライクプロセッサと名づけた。

本研究ではハード／ソフト協調学習システムのプロセスを利用して ARM ライクプロセッサの設計を通じてのプロセッサアーキテクチャの動作及びハードウェア記述言語の文法の習得、並びに、ハード/ソフト協調学習システムを用いての FPGA ボード上への実装の検討を行うことが目的である。

目次

図目次

.目次

1 はじめに

科学技術の発展により、パソコンなどの電子技術を駆使した商品の高性能化と低価格での入手を実現させてきた。また、それに伴い消費者の商品に対する要求も高くなっている。しかし、現時点での要求の実現に成功しても将来さらに条件の厳しい要求が発生することは明白であり、その結果、以前より短期間で高性能かつ安価な商品の開発を実現させる必要性が出てきた。

現時点で消費者の要求には、商品の高性能化、小型化があり、その要求を実現するためにシステム LSI を利用している。システム LSI を用いることで、マイクロプロセッサやメモリなどをワンチップにまとめることができる。この特徴を利用して必要な機能だけをワンチップにまとめることで、ユーザの要求をみたすことができる。現在、システム LSI はパソコン、自動車、携帯電話など様々な製品に搭載されており、製品の基幹部品として必要不可欠なものとなり、今後も需要は上昇すると考えられる。

また、近年の LSI 技術の発展はすばらしく、ワンチップに載せられるトランジスタ数は数十億を超えるようになった。さらに、半導体集積技術の向上によって、ワンチップ上に搭載できる機能も増加している。しかし、その技術を有効利用できる人材というのは、ハードウェアとソフトウェアの両方の知識を持った技術者に限られてしまう。なぜなら、ハードウェアでの設計の場合では高性能の計算が可能であるが設計コストが高くなり、ソフトウェア設計の場合では設計が容易だが性能面でハードウェアに劣ってしまうという特徴を理解していても、ハードとソフトの両方の知識がなければどこで切り分けてシステムを設計すれば効率的なのかということがわからないためである。だが、ハードとソフトの両方の知識を持ち、LSI を開発できる技術者は現時点では少数しか存在しないのが現状である。よって、LSI を用いた開発に必要不可欠なハードウェアとソフトウェア両方の知識を兼ね備えた人材の育成が現在急務となっている。

私の所属する研究室では、その人材の育成を目的としたハード/ソフト協調学習システムが考案された。このハード/ソフト協調学習システムとは、ハードウェアの分野とソフトウェアの分野を相互に関連づけながら両方のことを学ぶという学習システムである。

以上のような背景を踏まえ、本研究では大きく分けて次の二点を研究目的とする。

第一に、マイクロプロセッサを中心とする FPGA コンピュータシステムの設計と実装を行い、システム LSI として動作検証を行う。組み込みシステムには欠かせないマイクロプロセッサを実際に組み込みシステムで成功を修めている ARM プロセッサを参考に HDL を用いて設計することによって、プロセッサアーキテクチャの理解、HDL によるシステム設計の手法の習得を目的とする。

第二に、設計したプロセッサをハード/ソフト協調学習システムを用いて FPGA ボードへの実装の検討を行い、ハード/ソフト協調学習システムの有効性を示すことである。

本研究では最初にオリジナルプロセッサとして ARM ライクプロセッサを作成した。このプロセッサは ARM プロセッサを参考とした 3 オペランド命令形式の 32bit-RISC プロセッサである。ARM の命令セットから基本的な命令セットを選定し、実行できるようにした。次に、このプロセッサをハード/ソフト協調学習システムを用いての FPGA ボードへ実装の検討を行った。。また、設計用の HDL として、本研究では Verilog HDL を用いる。

2 ハード/ソフト協調学習システムとは

2.1 システム概要

ハード/ソフト協調学習システムとはハード・ソフトの両方を理解できる人材を育てるシステムであり、プロセッサアーキテクチャを意識したプログラミング能力を向上させることを目的としている[1,2]。

ソフトウェア面では、アーキテクチャが可変な命令セットシミュレータを用いて、アーキテクチャの仕組みの理解を促し、アセンブリ言語やC言語で書かれたプログラムの評価を行う。ハードウェア面では、ソフトウェア上での命令シミュレーションにより得たプロセッサアーキテクチャの知識を基にHDLによるプロセッサ設計を行い、そのプロセッサを FPGA ボードに実装し、評価を行うことによって、プロセッサアーキテクチャの理解を促すものである。

2.2 学習体系

実際に私が所属する研究室で作られた MONI プロセッサを例に挙げて学習体系を説明する。まず、学習体系の大きな流れとして、まず C 言語や MONI のアセンブリ言語を用いて実際にソフトウェアの開発を行うことでソフトウェア学習を行い、次に、そのプログラムを用いて MONI プロセッサを動かすことでプロセッサアーキテクチャの理解を促す。次に実際に HDL でプロセッサの設計を行うことで、シミュレータだけでは理解できない遅延などハードウェア特有の問題を解決できる能力を向上させる。図 1 に MONI を用いた協調学習システムの学習体系を示す。本研究の位置づけはハードウェア学習部の HDL によるプロセッサの設計である。

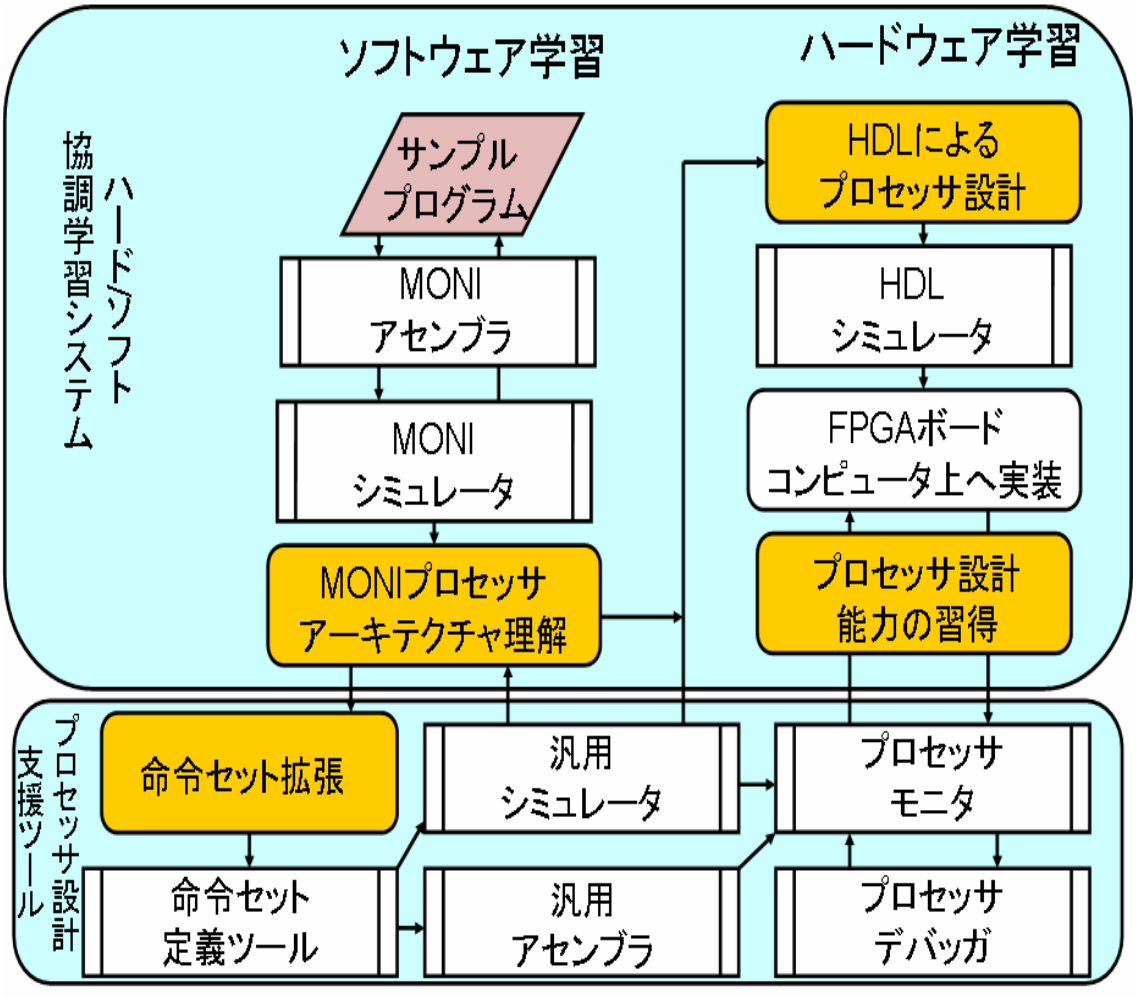

図 1:MONI を用いた協調学習システムの学習体系

3 ARMライクプロセッサの設計

3.1 設計思想

本研究において設計したマイクロプロセッサは、32bit マルチサイクルの RISC プロセッサである。命令セットは、ARM プロセッサで用いられている命令セットから基本の命令セットを選定した[3,5,6]。プロセッサ内部のマイクロアーキテクチャは、ARM の命令セットの特徴を生かせるようステータスレジスタと比較器を設置したほかは高速化のためできるだけ単純な構成とした。以下、このマルチサイクルマイクロプロセッサを、ARM のようなプロセッサということで ARM ライクプロセッサと呼ぶ。ARM ライクプロセッサには次のような特徴を持たせた。

・すべての命令長は 32bit 固定。
・3 つの命令形式を採用。
・3 オペランド方式の命令。
・15 個の汎用レジスタを搭載。
・ARM の命令セットの特徴を生かせるようステータスレジスタと比較器を設置。

3.2 命令セットアーキテクチャ

実装する条件コード及び命令セットの種類を以下に示していく。ARMはすべての命令について条件によって実行するかどうかを決めることができるのが特徴である。図2にARMの命令フォーマットを示す。図2のcondの部分に条件コードが入る。表1に実装する条件コードを示す。

図 2:ARM の命令フォーマット

表 1: 条件コード表

	[31: 28]	効果
EQ	0 0 0 0	Z セット
NE	0 0 0 1	Z クリア
AL	1 1 1 0	いずれも

また、今回採用した命令形式はデータ処理命令（AND、EOR、SUB、ADD、CMP、OR、MOV）とデータ転送命令（STR、LDR）、及び、分岐命令（B）の 3 つを採用した[4]。図 3 にレジスタと即値、図 4 にレジスタ同士の処理の場合のデータ処理命令の命令フォーマットを、図 5 にデータ転送命令の命令フォーマット、図 6 に分岐命令の命令フォーマットを示す。また、表 2 にデータ処理命令のオペコードと効果を示した表を、表 3 にデータ転送命令のオペコードと効果を示した表を、表 4 に分岐命令のオペコードと効果を示した表をそれぞれ示す。

	[31:28]	[27:26]	[25]	[24:21]	[20]	[19:16]	[15:12]	[11:8]	[7:0]
Ope Rd Rn imm(8bit)	cond	0 0	1	OP コード	0	Rn	Rd	0 0 0	imm(8bit)

図 3:レジスタと即値の命令フォーマット

	[31:28]	[27:26]	[25]	[24:21]	[20]	[19:16]	[15:12]	[11:4]	[3:0]
Ope Rd Rn Rm	cond	0 0	0	OP コード	0	Rn	Rd	0 0 0 0 0 0 0 0	Rm

図 4:レジスタ同士の命令フォーマット

表 2: データ処理命令表

命令	[24: 21]	効果
AND	0 0 0 0	Rd←Rn AND Rm
		Rd←Rn AND imm(8bit)
EOR	0 0 0 1	Rd←Rn EOR Rm
		Rd←Rn EOR imm(8bit)
SUB	0 0 1 0	Rd←Rn – Rm
		Rd←Rn – imm(8bit)
ADD	0 1 0 0	Rd←Rn + Rm
		Rd←Rn + imm(8bit)
CMP	1 0 1 0	Scc on Rn – Rm
		Scc on Rn – imm(8bit)
ORR	1 1 0 0	Rd←Rn OR Rm
		Rd←Rn OR Rm
MOV	1 1 0 1	Rd←Rm
		Rd←imm(8bit)

	[31:28]	[27:26]	[25:21]	[20]	[19:16]	[15:12]	[11:0]
Ope Rd [Rn]	cond	0 1	0 1 1 0 0	L	Rn	Rd	0 0 0 0 0 0 0 0 0 0 0 0

図 5:データ転送の命令フォーマット

表 3: データ転送命令表

命令	[20]	効果
STR	0	mem[Rn]←Rd
LDR	1	Rd←mem[Rn]

	[31:28]	[27:24]	[23:0]
B label	cond	1 0 1 0	24 ビット符号付きワード・オフセット

図 6:分岐命令の命令フォーマット

表 4: 分岐命令表

命令	書式	効果
B	B LABEL	分岐

3.3 プロセッサアーキテクチャの設計

図7にARMライクプロセッサのデータパスアーキテクチャを示す。

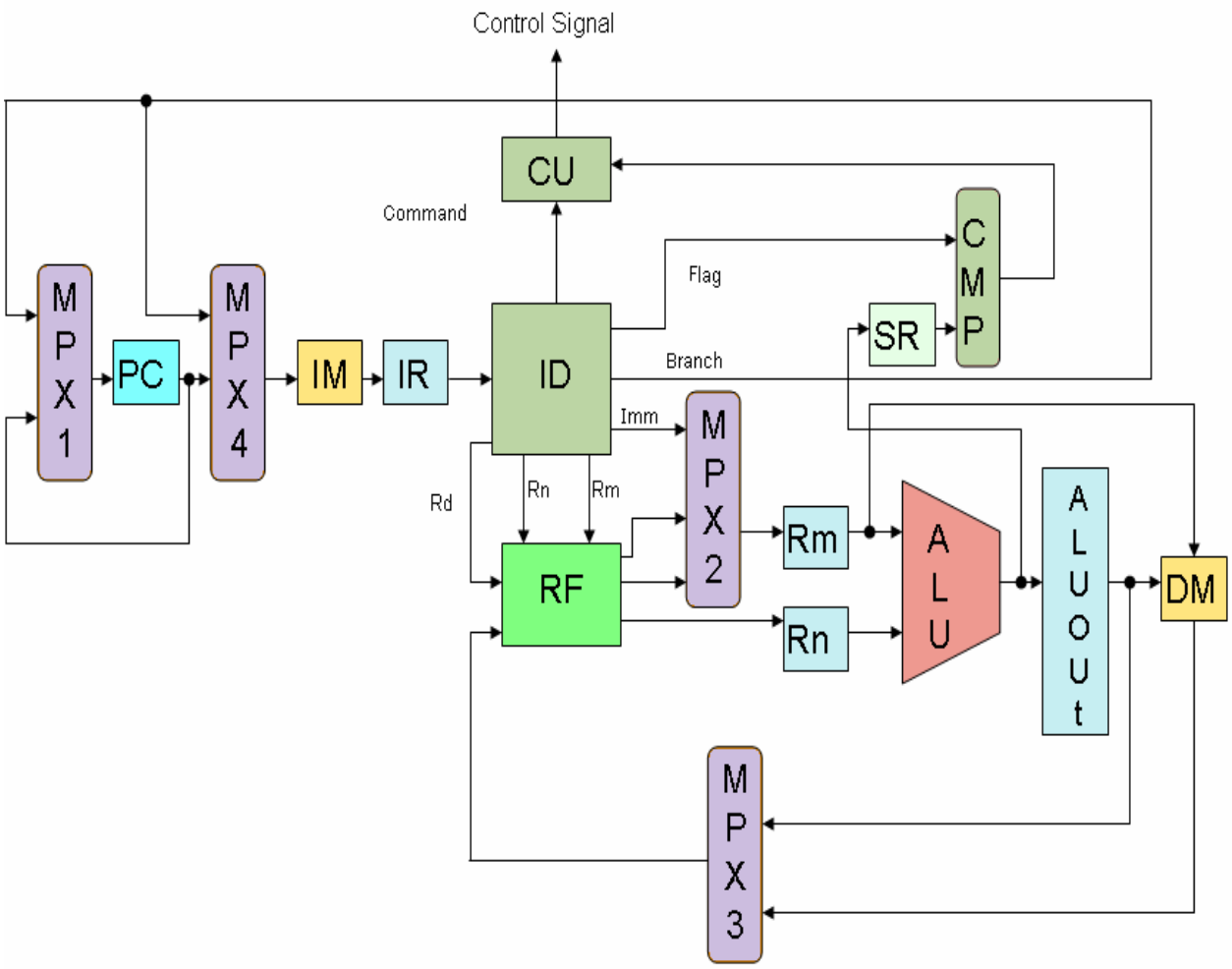

図 7:ARM ライクプロセッサの回路図

ARMライクプロセッサは次の要素から構成される。

・IR(Instruction Register)

命令メモリから読み出した命令を保存するレジスタ。

・ID(Instruction Decoder)

IR から読み出した命令を解読し、Command、Flag、Branch、Imm、Rd、Rn、Rm 信号を出力する。

・CU(Control Unit)

ID から命令を示す Command 信号を受け取り、各モジュールに制御信号を出力する。

・RF(Register File)

汎用レジスタ 15 個から構成される。デスティネーションレジスタ 1 つと、ソースレジスタ 2 つを同時に指定できる。

・Rn

Rn 信号で指定されたレジスタファイルの値を保存するレジスタ。

・Rm

Rm 信号で指定されたレジスタファイルの値を保存するレジスタ。

・ALU(Arithmetic Logic Unit)

2 入力 1 出力の算術論理演算装置。主に算術演算、論理演算を行う。

・ALUout

ALU から出力された値を保存するレジスタ。

・SR(Status Register)

条件コードが AL でかつ実行する命令がデータ処理命令のときの演算結果の状態を格納するレジスタ。状態はゼロフラグを立てることで 2 種類が表現可能。

・CMP(Comparator)

ID から出力された Flag 信号と SR からの出力を比較する。比較の結果、完全にマッチすれば CU に 1 を出力する。

・PC(Program Counter)

現在実行中の命令メモリのアドレスを示す。

・MPX(Multiplexer)

複数の入力信号から 1 つの出力信号を選択する。

以下に各命令の実行過程を図に示しながら説明していく。

(1)データ処理命令(AND、EOR、SUB、ADD、CMP、ORR、MOV)

データ処理命令の場合、0フェーズ目で命令メモリからIRへ命令を渡す。次に1フェーズ目でIRからIDへ命令を渡し、Command、Imm、Rd、Rn、Rm信号を出力し、CUで制御信号を各モジュールに出力したのち、Rn、Rmに値を書き込む。2フェーズ目でALUで処理を行い、ALUoutへ処理した結果を書き込む。また、実行された命令の条件コードがALだった場合SRにゼロフラグを立てるか立てないかを選択して状態を格納する。3フェーズ目でALUoutの値をRd信号が示すRFへと書き込む。図8にデータ処理命令の場合の流れを示す。

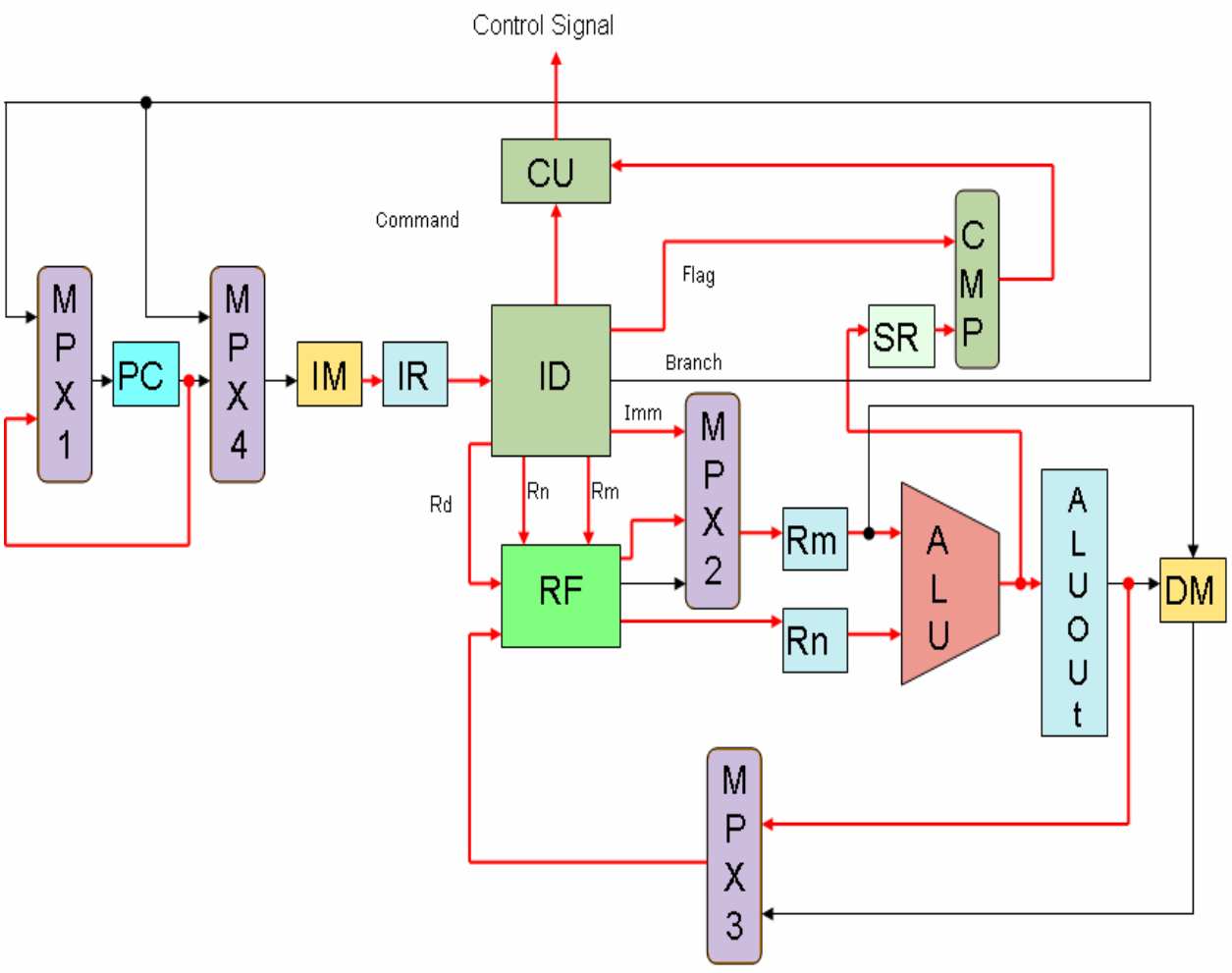

図 8:データ処理命令の実行過程

(2)データ転送命令(LDR、STR)

データ転送命令の場合、0フェーズ目で命令メモリからIRへ命令を渡す。次に1フェーズ目でIRからIDへ命令を渡し、Command、Rd、Rn信号を出し、CUで制御信号を各モジュールに出す。このとき命令がLDR命令ならばRnにデータアドレスを書き込む。STR命令ならばRmにRd信号の示すレジスタファイルの値を書き込み、Rnにデータアドレスを書き込む。2フェーズ目はLDR命令のときRnの値をそのままALUoutへ書き込む。STR命令のときはRmの値をデータとしてDMに書き込み、Rnの値をそのままALUoutへ書き込む。3フェーズ目でALUoutの値をアドレスとしてDMへ出力する。LDR命令のとき4フェーズ目で指定されたDMのアドレスの値をRd信号が示すRFへと書き込む図9にLDR命令の場合の流れを、図10にSTR命令の場合の命令の流れを示す。

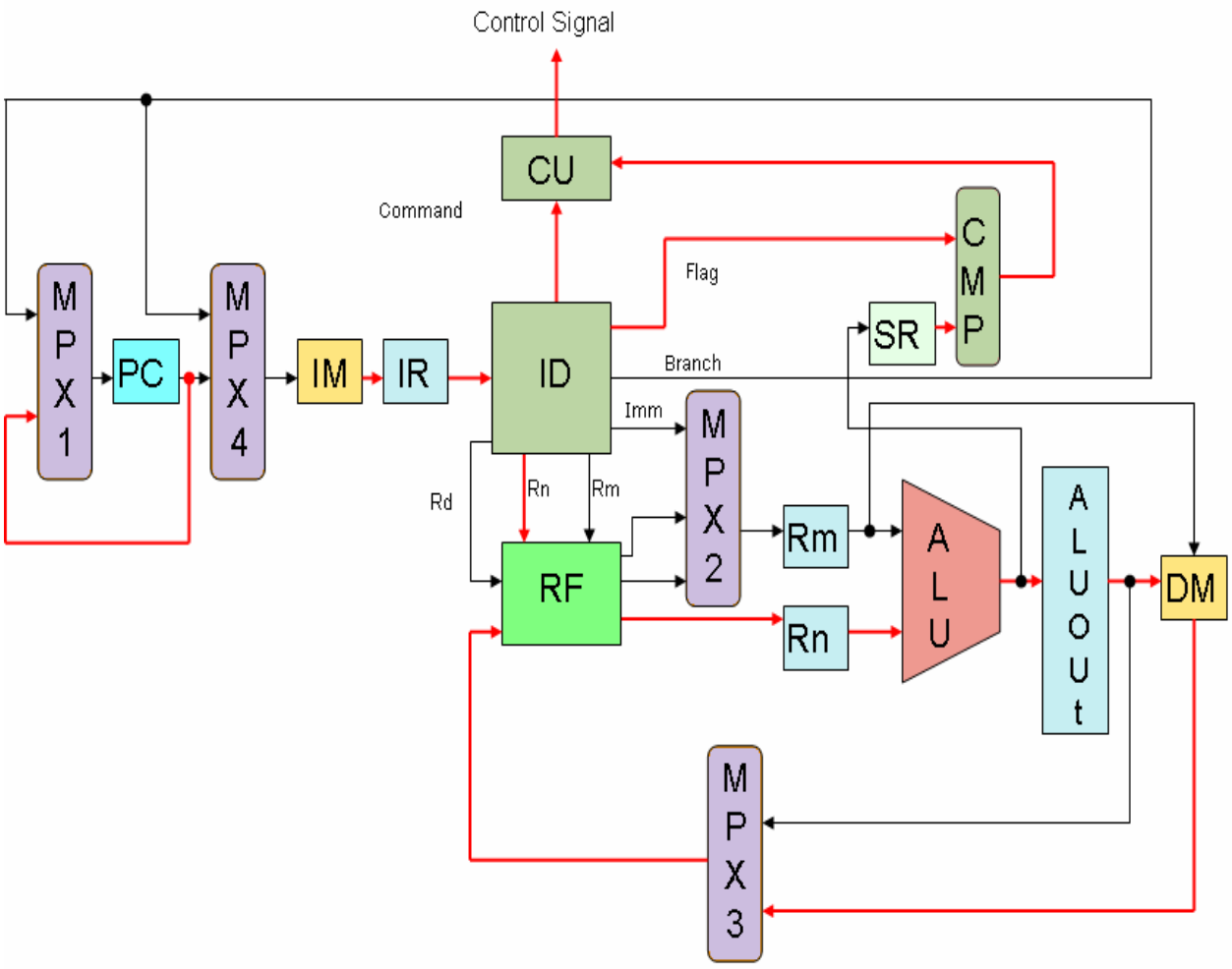

図 9:LDR 命令の実行過程

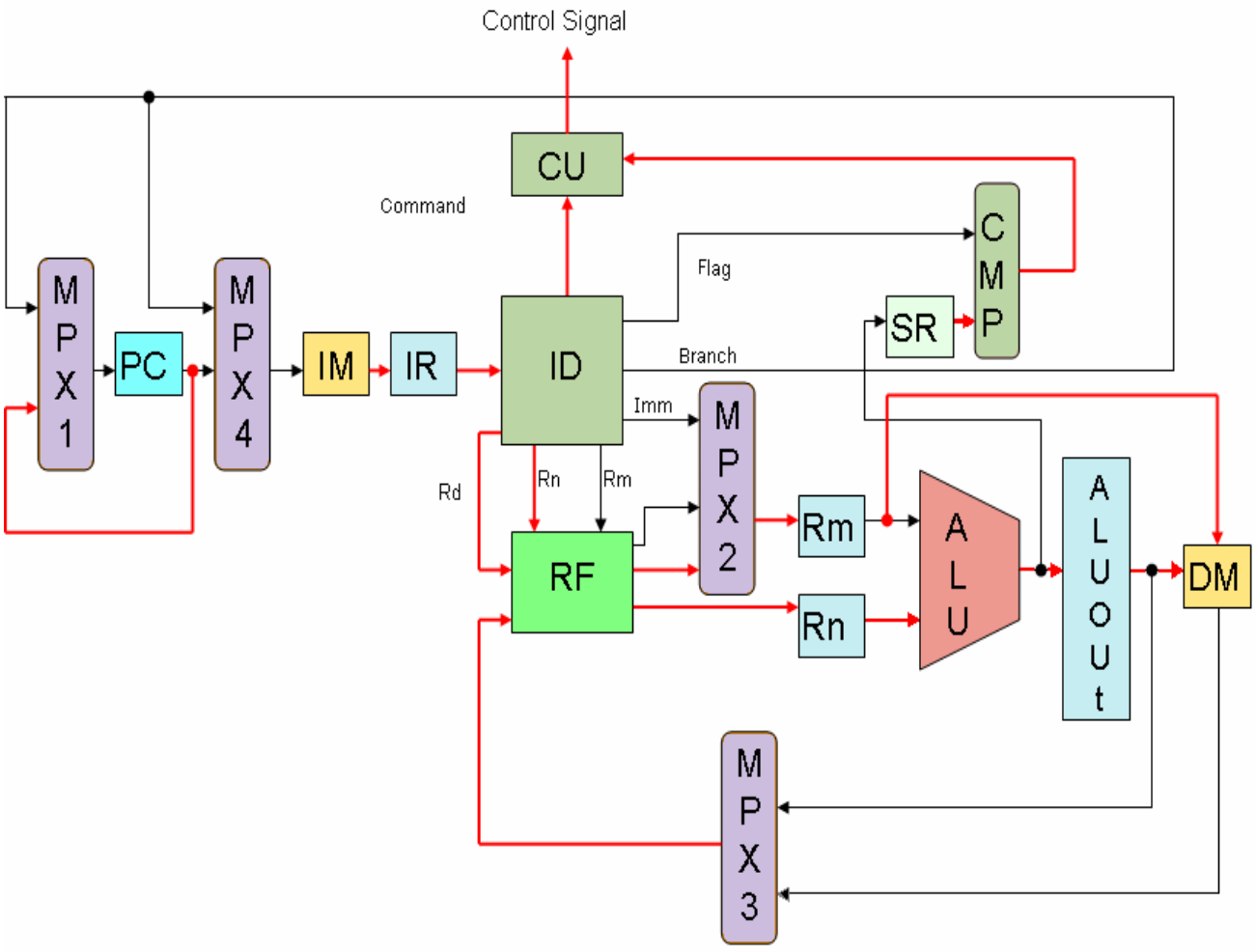

図 10:STR 命令の実行過程

(3)分岐命令

データ転送命令の場合 0 フェーズ目で命令メモリから IR へ命令を渡す。次に 1 フェーズ目で IR から ID へ命令を渡し、Command、Branch 信号を出し、CU で制御信号を各モジュールに出す。また、Branch 信号は即値で IM のアドレスを示している。図 11 に分岐命令の実行過程を示す。

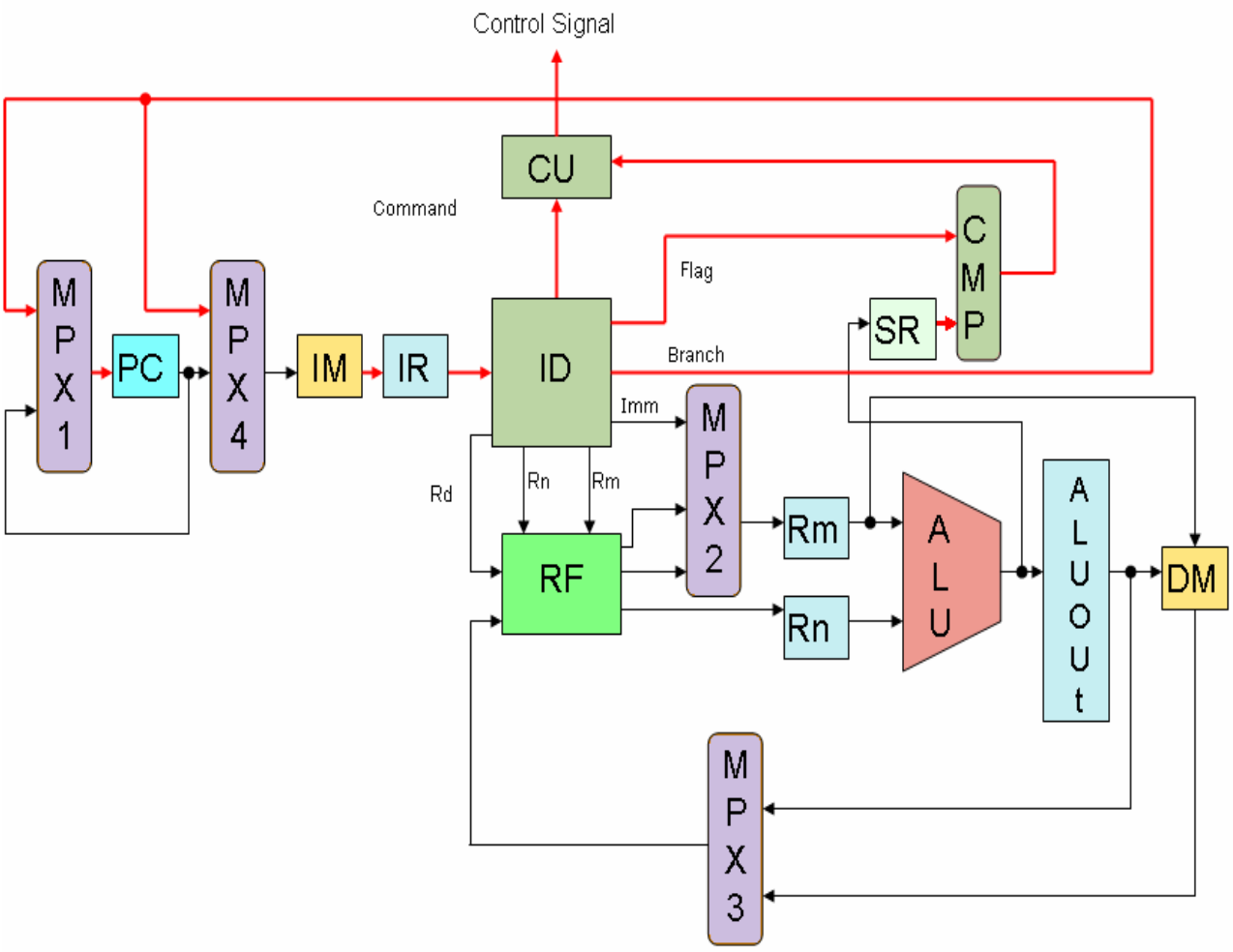

図 11:分岐命令の実行過程

4 Verilog HDLを用いたARMライクプロセッサの作成

設計したプロセッサを Mentor Graphics Corporation の ModelSim Xilinx Edition III v6.1e を用いて作成した。また、ハードウェア記述言語に Verilog HDL を用いた[7,8]。以下に一部のモジュールを外部仕様の図、及び、VerilogHDL で記述したソースの一部を用いて説明する。

(1)命令デコーダ ID

IR から読み出した命令を解読し、Command、Flag、Branch、Imm、Rd、Rn、Rm 信号を出力するユニットである。図 12 に ID の外部仕様を示す。

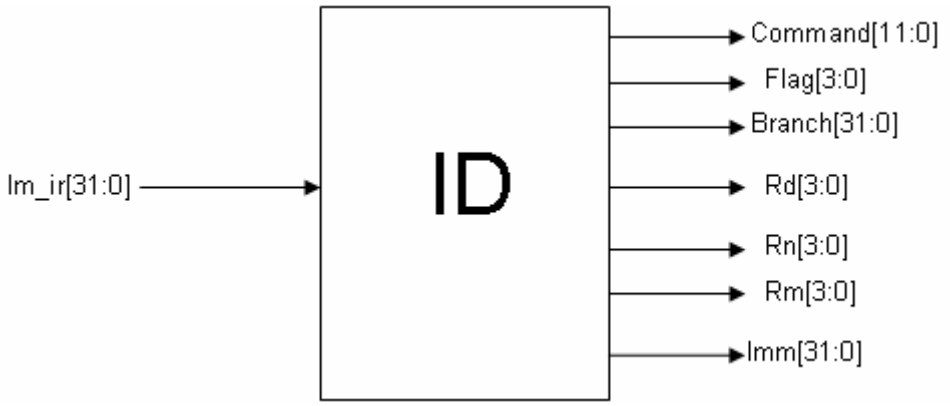

図 12:ID の入出力仕様

読み込んだ命令のうち 12 ビットを Command 信号、3 ビットを Rd、Rn、Rm 信号として出力する。また、条件コードが EQ のときには Flag 信号の 0001 を NE 信号のときには 0000 を出力し条件コードが AL のときには 1110 を出力する。また Branch 信号と Imm 信号はビットを拡張して出力する。図 13 に VerilogHDL で記述した ID のソースの一部を示す。今回は読み込んだ命令をビットごとに分けるだけなので、全ての信号を継続的代入文 assign を用いて書いている。

```
`timescale 1ns/1ps
module ID(
        ir_id_in,
          :
          :
          :

        imm
        );

          :
          :
          :


        assign  id_cu_out=(ir_id_in[27:24]==4'b1010)?{ir_id_in[31:24],4'b0000}:      //継続的代入文で全ての
                          ir_id_in[31:20];                                            信号の記述
                              :
                              :
                              :

endmodule
```

図 13:ID の HDL 記述

(2)ステータスレジスタ SR

条件コードが AL でかつ実行する命令がデータ処理命令のときの演算結果の状態を格納するレジスタである。図 14 に SR の外部仕様の図を示す。

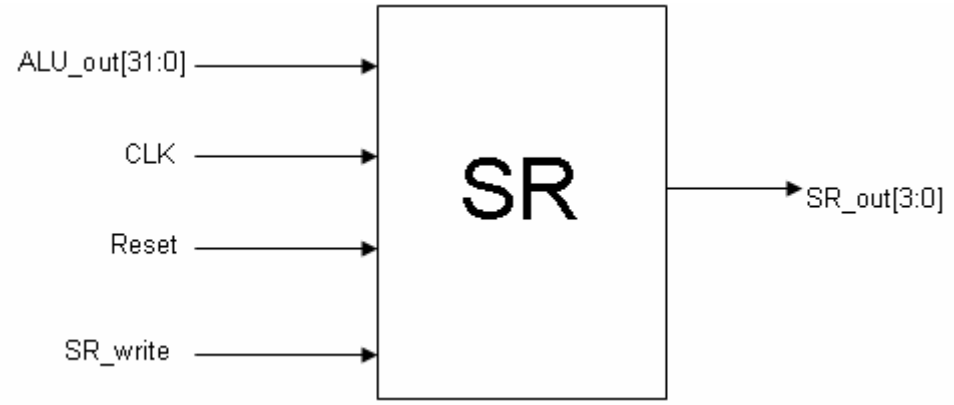

図 14:SR の入出力仕様

SR が書き込まれるのはデータ処理命令で、命令の条件コードが AL のときだけである。ALU の演算結果が 0 のとき 0001 を出力し、それ以外なら 0000 を出力する。図 15 に VerilogHDL で記述した SR のソースの一部を示す。always 文の中で条件によって出力する信号の違いを判別するようにしている。

```
`timescale 1 ns/1 ps
module SR(
        alu_sr_in,

        .
        .
        .

        );
        .
        .
        .
    assign zero=~|alu_sr_in[31:0];

    always@(posedge clk or negedge reset) begin          //出力信号の条件
        if(!reset) sr_cmp1_out <= 32'bz;
          .
          .
          .
        end
endmodule
```

図 15:SR の HDL 記述

(3)比較器 CMP

ID から出力された Flag 信号と SR からの出力を比較する。比較の結果、完全にマッチすれば CMP_out に 1 を出力する。図 16 に CMP の外部仕様の図を示す。

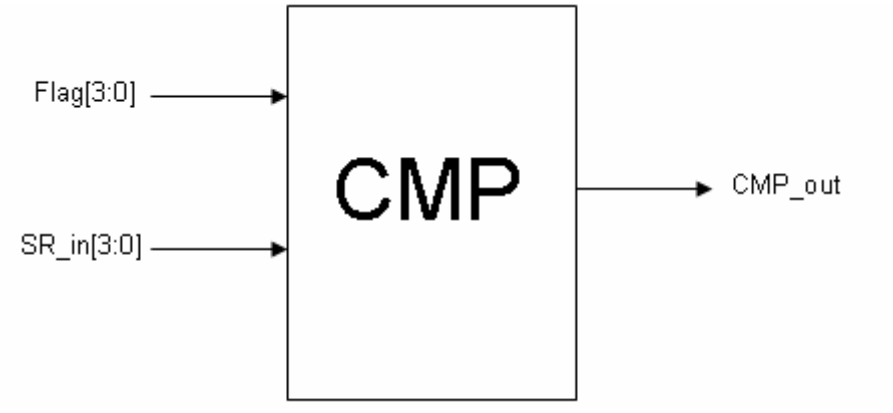

図 16:CMP の入出力仕様

CMP からの出力が 1 ならば条件コードが EQ、及び NE のとき命令が実行される。出力が 0 かつ条件コードが EQ、及び NE のときには次の命令が実行される。図 17 に VerilogHDL で記述した CMP のソースの一部を示す。

```
`timescale 1ns/1ps
module CMP(
        flag_cmp_in,
            :
            :
            :
                );
            :
            :
            :
    assign  cmp_cu_out=(flag_cmp_in==sr_cmp1_in)?1'b1 : 1'b0;
endmodule
```

図 17:CMP の HDL 記述

(4)制御回路 CU

ID から命令を示す command 信号を受け取り、各モジュールに制御信号を出力するユニットである。図 18 に CU の外部仕様の図を示す。

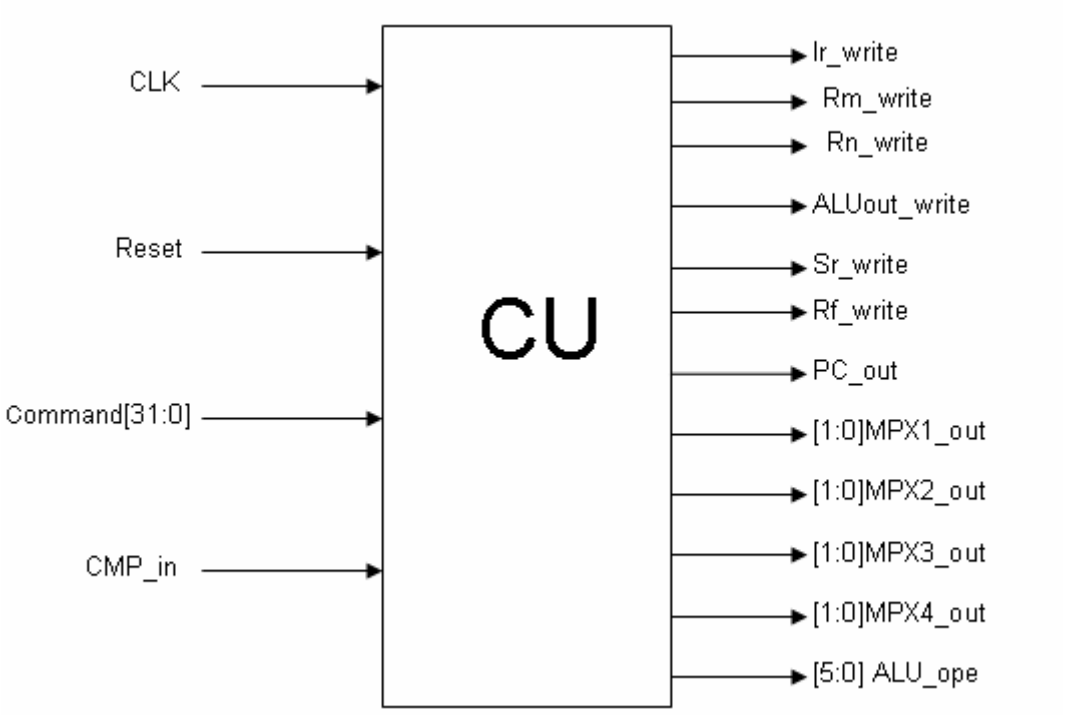

図 18:CU の外部仕様

CU で全ての命令のフェーズごとの信号を出力する。図 19 に VerilogHDL で記述した CMP のソースの一部を示す。最初に 10 クロック分の遅延を持たせた後にフェーズが 0 になる条件を always 文で書いた。また全ての命令及びフェーズごとに出力する信号を継続的代入文 assign を用いて書いたため、記述量が 6000 行を超えてしまった。

```
`timescale 1ns/1ps
module CU(
          .
          .
          .

      );

   parameter [11:0] AND=12'b111000000000, //IDからのCommand信号のパラメータ宣言
          .
          .
          .

   parameter [5:0] //ALUへの信号のパラメータ宣言

          .
          .
          .

   always @(posedge clk or negedge reset) begin //10クロック分遅延させる
      if(!reset) count<=0;
      else if(count==4'b1010) count<=4'b1010;
      else count<=count+4'b0001;
   end

   always @(posedge clk or negedge reset) begin //10クロック目にrestart信号を1にする
      if(!reset) restart<=0;
      else if(count==4'b1001) restart<=1;
      else restart<=restart;
   end

   always @(posedge clk or negedge reset) begin //フェーズが0になる条件
      if(!reset) pcount<=4'b0000;
      else if(!restart) pcount<=4'b0000;
      else if(pcount==4'b0000)pcount<=4'b0001;
      else begin //全ての命令でフェーズが0になる条件
         case(id_cu_in)

            AND: begin
            if(pcount==4'b0011) pcount<=4'b0000;
            else pcount<=pcount+4'b0001;
            end

          .
          .
          .
  end

  //全ての信号をassign文で書く
  assign cu_pc_out=(restart==1'b0)?0:
          .
          .
          .
endmodule
```

図 19:CU の HDL 記述

5 ハード/ソフト協調学習システムを用いたFPGAボードへの実装の検討

実際に作成したプロセッサをハード/ソフト協調システムを用いての FPGA ボードへの実装を検討する[1]。

FPGA ボードに実装する場合、FPGA 上でプロセッサを動作させるための環境整備を行わなければならなかった。プロセッサは単独では FPGA ボードに搭載しても動作しないので、プロセッサを動作させるためには、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、実行終了後に結果をメモリから読み出す必要がある。そして、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本人が用意しなければならなかった。また、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、実機上では動作しないことがある。その主な原因として、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ことがあげられるが、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を作成した本人が作成し、レジスタを観測しなければならなかった。

しかし、以上のことは学習を行う者に対しての負担が大きく、また、学習の本質を突くものではなかった。そこで、これらの負担を軽減させ、学習を行う者に FPGA ボードへの実装を行いやすくするべくプロセッサデバッガが開発された。このプロセッサデバッガには FPGA ボード上でプロセッサを動作させるための環境整備と実機上でのデバッガという 2 つの役目がある。1 つめの環境整備では、プロセッサの FPGA ボード上での実装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、検証するための周辺モジュールとして提供している。2 つめの実機上でのデバッガでは、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することで、実行途中のプロセッサの動作を追跡しながら検証を行うことができる。

図 20 にプロセッサデバッガに搭載するプロセッサのインターフェースの図を示す。今回はプロセッサデバッガの FPGA ボードへの実装に関連したポートを示している。プロセッサデバッガでは、ハーバードアーキテクチャのプロセッサを主な対象としており、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となる。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（CE）、ライトイネーブル（WE）、リードイネーブル（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 と RE を 1、WE を 0 にし、データを書き込む場合は、CE と WE を 1、RE を 0 にする。また、CE が 0 の場合はデータの読み書きを行うことはできない。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。

また、基本的に、入力と出力で異なるポートを使用するが、入出力で同じポートを共有するプロセッサにも対応できるようになっている。しかし、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り、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。

このプロセッサデバッガのインターフェースに対応したポートを今回作成した ARM ライクプロセッサの Verilog HDL のソースの一部を変えて繋げることで FPGA ボードへの実装が行える。

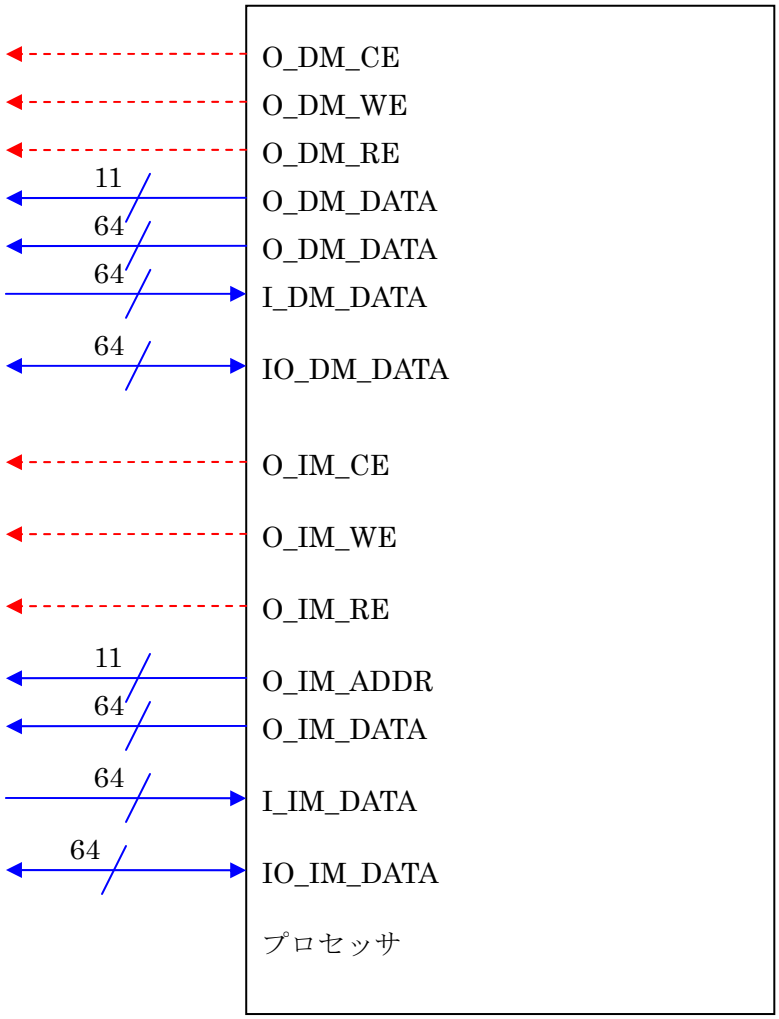

図 20:プロセッサデバッガに搭載するプロセッサのインターフェース

6 おわりに

本論文ではハードウェア記述言語 Verilog HDL を用いてオリジナルのプロセッサである ARM ライクプロセッサの設計を行った。命令セットは ARM プロセッサで実際に使われているものからいくつかを選定し設計・作成を行った。シミュレーションツールによる回路の論理検証の後、ハード/ソフト協調学習システムを用いての FPGA ボード上への実装の検討を行なった。これらの経緯からプロセッサアーキテクチャの動作、ハードウェア記述言語 Verilog HDL の文法の習得、及びハード/ソフト協調学習システムの有効性を示すことができたと考えられる。

今後の課題としては、ハード/ソフト協調学習システムを用いての FPGA ボード上への実装を行うことである。本研究で検討したプロセッサデバッガを用いて FPGA ボード上に実装を行うことで ARM ライクプロセッサの評価を行うとともに、ハード/ソフト協調学習システムの評価を行うことにより、システムの有用性や問題点を挙げていくことで、より良い学習システムを実現させていくことが課題である。

謝辞

本研究の機会を戴き、貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝致します。また、本研究に関して貴重なご意見をいただきました、難波翔一朗氏、志水建太氏をはじめ、同研究室内の方々の様々な励ましや助言に対し、深く感謝致します。

参考文献

[1] 難波翔一朗：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,立命館大学大学院 理工学研究科,修士論文,2007.

[2] 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.

[3] 古川達久：マルチサイクル・パイプライン方式による教育用マイクロプロセッサの設計と検証, 立命館大学理工学部卒業論文,2003.

[4] David A.Patterson, John L.Hennessy 著, 成田光彰 訳：コンピュータの構成と設計（上）,日経 BP 社,1999.

[5] Steve Furber 著, アーム株式会社 監訳：改訂 ARM プロセッサ,CQ 出版,2001.

[6] TECH I Vol.18 ARM プロセッサ入門,CQ 出版,2003

[7] Jayaram Bhasker 著, 佐々木尚 訳：Verilog HDL 論理合成入門,CQ 出版,2001.

[8] 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,2006.