卒業論文

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

氏名　　：志水　建太
学籍番号　：2210030185-0
担当教員　：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、ハードウェアとソフトウェアの両側の知識を持つ人材を育てるために、本研究室で開発を進めている、ハード/ソフト協調学習システムを使用し、Verilog-HDL を用いてシングルサイクルプロセッサ、3 段マルチサイクルプロセッサ、4 段マルチサイクルプロセッサのクロック非同期、クロック同期、1 クロック遅延を、それぞれプロセッサに対して設計を行った。設計したプロセッサは、本研究室で MIPS のサブセットとして定義した、教育用マイクロプロセッサ(MONI)である。

設計したプロセッサを RTL シミュレーション上で動作検証を行い、次に、FPGA 上に実装し、動作検証を行った。検証に用いたプログラムは、N までの和、素数判別、除算、三角形の判別、2 点を通る直線の方程式である。それぞれのプロセッサの性能比較を行い、また、プロセッサ設計を行うにあたり利用した、ハード/ソフト協調学習システムについても評価を行った。

目次

図目次

表目次

1. はじめに

近年の急速な半導体集積技術の進歩により、LSI の性能向上、小型化、省電力化等が可能となった。しかし、集積技術の進歩が非常に速いにもかかわらず、LSI の設計生産の向上が追いついていないため、設計生産性危機として問題化してきている。また、システム LSI を搭載した商品サイクルが短くなる中で、開発にかけられる期間もこれまで以上に短くなってきている。

システム LSI を短期間かつ低コストで、信頼性高く設計、製造することが要求されている現在、LSI 設計技術と組み込みソフトウェア設計技術の協調が必要となっている。しかし、ハードウェアとソフトウェアの切り分けが難しくなり、ハードとソフトの両側からシステム全体を見ることができる人材が、より必要になってきている。

LSI 開発においては、プロセッサとソフトウェアは切り離せない関係にあり、開発に携わる技術者にとって、ハードウェアとソフトウェア両方の知識は必要不可欠である。ハードウェア面では、プロセッサアーキテクチャの理解が大前提であり、開発プロセスや設計効率を意識した設計技術が求められる。ソフトウェア面ではハードウェアと密接に関わりを持ったアセンブリレベルでのプログラミングや、高級言語によるプログラミング技術が求められる。これらを解決するために早期の教育が必要となり、大学においてハードウェアとソフトウェアの関係を密にした教育が重要である。このため、本研究室ではハード/ソフト協調学習システムの開発を進めてきた。

ハード/ソフト協調学習システムとは、ハードウェアとソフトウェアの両方の知識を持つ人材を育てるための、協調学習システムである。まず、ソフトウェアの学習を行い、次にハードウェアの学習を行うことによって、アーキテクチャを意識したプログラミングを行えるようにする。[12][13]

以上のような背景を踏まえ、本研究ではハード/ソフト協調学習システムを利用し、実際にソフトウェア学習を行った上で、HDL を用いて教育用プロセッサ(MONI)の設計を行う。また、プロセッサデバッガを用いて設計したプロセッサを FPGA 上に搭載してデバッグを行い、プロセッサの評価を行う。さらに、ハード/ソフト協調学習システムの評価も行うことを目的とする。[8]

本研究はハード/ソフト協調学習システムで用いる MONI プロセッサを、Verilog-HDL を用いて、クロック非同期とクロック同期で動作をする、シングルサイクルプロセッサ、3 段マルチサイクルプロセッサ、4 段マルチサイクルプロセッサを設計する。また、命令メモリ、データメモリと、設計したプロセッサとの間に遅延を意識した、シングルサイクルプロセッサ、3 段マルチサイクルプロセッサ、4 段マルチサイクルプロセッサを設計する。まず、設計したプロセッサを ModelSim 上で検証を行い、次に、プロセッサデバッガと接続し、FPGA 上に搭載して検証を行う。

本論文では、第 2 章で、ハード/ソフト協調学習システムと、設計した教育用プロセッサ(MONI)のアーキテクチャについて詳しく説明する。第 3 章で、シングルサイクルプロセッ

サ、第 4 章で、3 段マルチサイクルプロセッサと 4 段マルチサイクルプロセッサについて述べる。第 5 章で、設計したプロセッサの評価とハード/ソフト協調学習システムの評価について述べる。

2. ハード/ソフト協調学習システム

2.1 ハード/ソフト協調学習システムの概要

ハード/ソフト協調学習システムとは、プロセッサアーキテクチャを意識しながらプログラミング学習を行うための、ハードウェアとソフトウェアの協調学習システムである。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、プロセッサアーキテクチャの理解、アセンブリ言語や C 言語で設計したプログラムや、命令セットの評価を通して学習する。ハードウェア面では、シミュレータで理解したプロセッサの知識を基に、HDL によるプロセッサ設計を行う。次に、設計したプロセッサを FPGA ボードコンピュータに搭載し、実機検証を行い学習する。このように、HDL によるプロセッサ設計とソフトウェア開発を融合させることで、アーキテクチャを意識したプログラミングを行えるようになることが、このシステムの目的である。

ハード/ソフト協調学習システムには次の特徴がある。ソフトウェアシミュレータによる可観測性とハードウェアを融合することで、ソフトウェアシミュレータだけでは分からないことを、ハードウェア設計を通して学ぶことができる。また、ボードコンピュータでは見えないプロセッサの内部動作を、シミュレータで観測できる。プロセッサアーキテクチャを段階的に学習することで、どのようにプロセッサが発展してきたか学習できる。アーキテクチャを意識した上でのプログラミング演習により、プロセッサを高速に動作させるプログラミングを学習できる。同じプログラムを異なるアーキテクチャで動作させ、プロセッサアーキテクチャの評価ができる。

図 1 にハード/ソフト協調学習システムの学習体系を示す。左側が、ソフトウェア学習である。学習者は、MONI 仮想シミュレータ上でアセンブリプログラミングを行い、MONI 仮想シミュレータで、単一サイクル、マルチサイクル、パイプライン、スーパースカラの 4 つから選択して、シミュレーションを行うことができる。これにより、MONI プロセッサの内部動作を理解しながらプログラミングができる。さらに、ハードウェアを意識したソフトウェアの設計手法を身につけることができる。右側が、ハードウェア学習である。実際に HDL を用いてプロセッサの設計を行い、FPGA 上に実装して、プロセッサの動作検証を行う。プロセッサ設計支援ツールでは、命令セット定義ツール、汎用アセンブラ、汎用シミュレータにより、命令セットを独自に定義できる。また、プロセッサモニタ、プロセッサデバッガにより FPGA 内部の観測や制御ができる。[9]

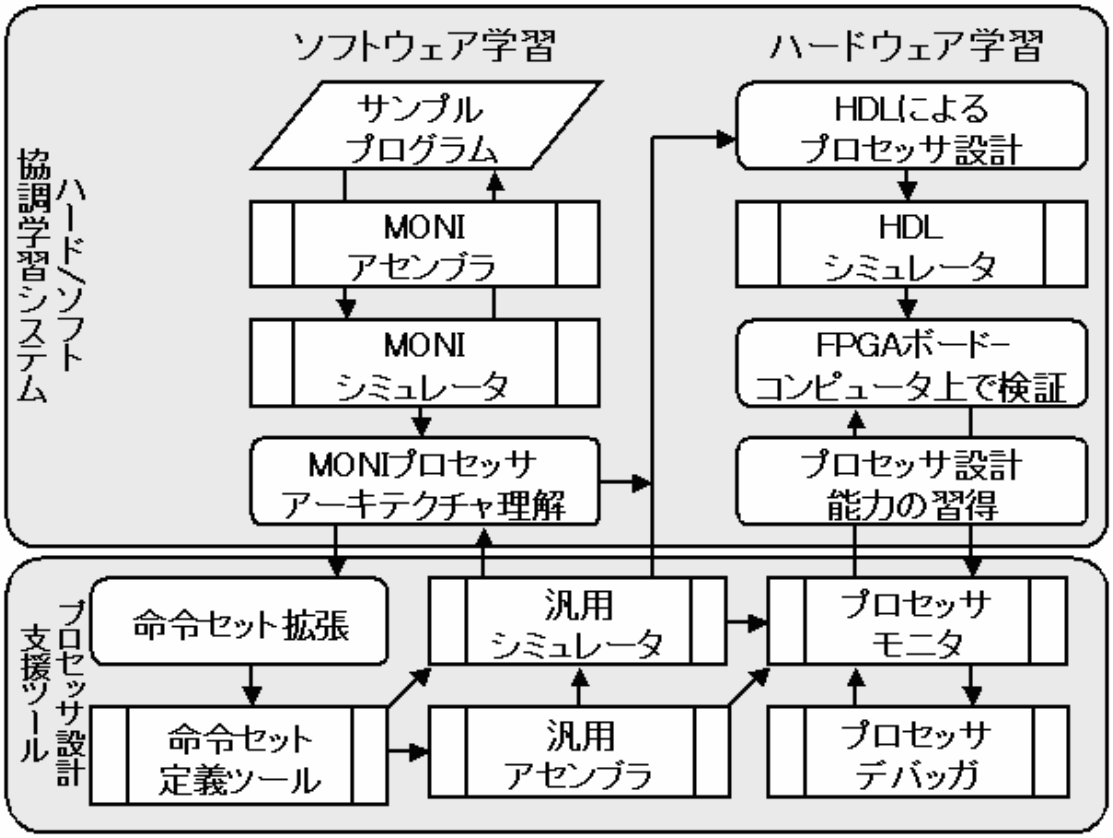

図 1：ハード/ソフト協調学習システムの学習体系

2.2 教育用プロセッサ(MONI)の概要

MONI とは、本研究室で MIPS のサブセットとして定義した、教育用マイクロプロセッサである。[7]シングルサイクル、マルチサイクル、パイプライン、スーパースカラの 4 種類のアーキテクチャがあり、マイクロプロセッサの初期段階であるシングルサイクル方式から、マルチサイクル方式、パイプライン方式、スーパースカラ方式を通じて段階的に学習することで、プロセッサアーキテクチャの理解を深めようという意図から用意されている。MONI には次のような特徴がある。

- 全ての命令語長は 16bit 固定
- 3 オペランドの命令方式
- 4 つの命令形式を用意
- 全 43 命令

次に、プロセッサデバッガと接続できるように設計した、MPU のインタフェースについて説明する。図 2 に MPU のインタフェースを示す。このインタフェースは、設計した全てのプロセッサ共通である。

IM, DM への制御は、CE(チップイネーブル), WE(ライトイネーブル), RE(リードイネーブル)で行う。{CE, WE, RE}が{1, 1, 0}で、データの書き込み、{1, 0, 1}で、データの読み出

しを行う。CE が 0 の場合は何も行わない。IO から始まるポートはデータの入力出力を一本で行う。DM へ書き込むときはデータを送り、読み取るときはデータを受け取る役割をする。ENA 信号が 1 に立ち上がり、立ち上がっている間、MPU が動作する。0 の場合 MPU の動作は停止する。0 の場合{CE, WE, RE}が{1, 1, 0}で、RF へデータを書き込み、{1, 0, 1}で、データを出力する。RST 信号は、0 にすることで、全てのモジュールにリセットがかかる。FIN 信号は、プログラムの実行が終了したときに 1 となる。終了命令である HALT 命令を受け取った時、1 クロックだけ立ち上がる。

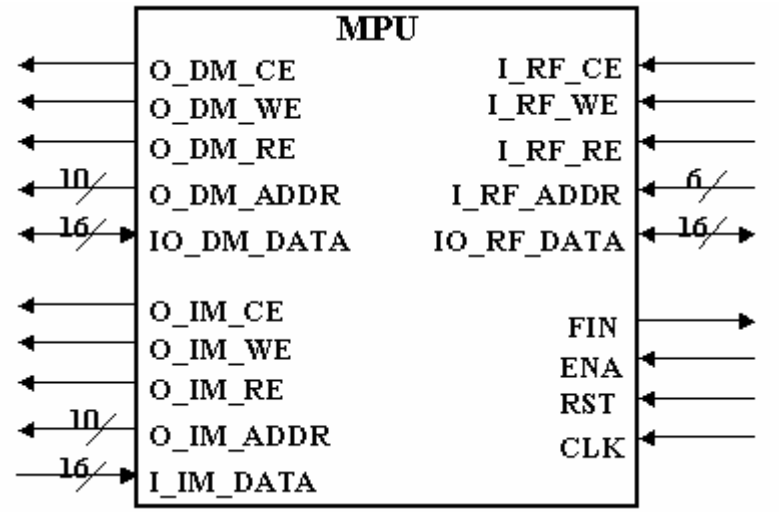

図 2 : MPU のインタフェース

MONI プロセッサの動作を開始するには、RST を 0 から 1 に切り替え、ENA を 1 にする。次のクロックの立ち上がりで命令メモリの 0 番地から命令の読み出しを行い、プログラムの実行が開始される。[9]

図 3 にクロック非同期とクロック同期を示す。クロック非同期とは、クロックのタイミングに合わせず、データを送る。クロックの切り替わりでデータが変わると、わずかな遅延はあるが、切り替わった値を元にデータを送る。クロック同期とは、クロックのタイミングに合わせデータを送る。前の値を元に、データを送る。また、CPU はキャッシュにアクセスして実行をしているが、キャッシュに必要なデータがなかった場合、メインメモリにアクセスを行わなければならず、アクセス時間が増えてしまう。そこで、命令メモリ、データメモリから MPU へデータが届くのに、1 クロックかかる遅延を考えて設計した。このプロセッサはクロック同期である。図 4 に遅延の関係を示す。命令メモリの場合は、立ち上がりでアドレスを受け取り、データを送る。1 クロック後のクロックの立ち上がりで MPU にデータが届く。データメモリの場合は、立下りでアドレスを受け取り、データを送る。1 クロック後の立下りで MPU にデータが届く。[15][22]

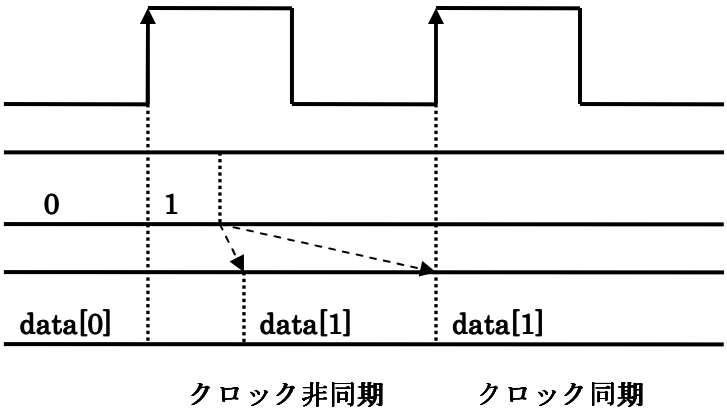

図 3：クロック非同期とクロック同期

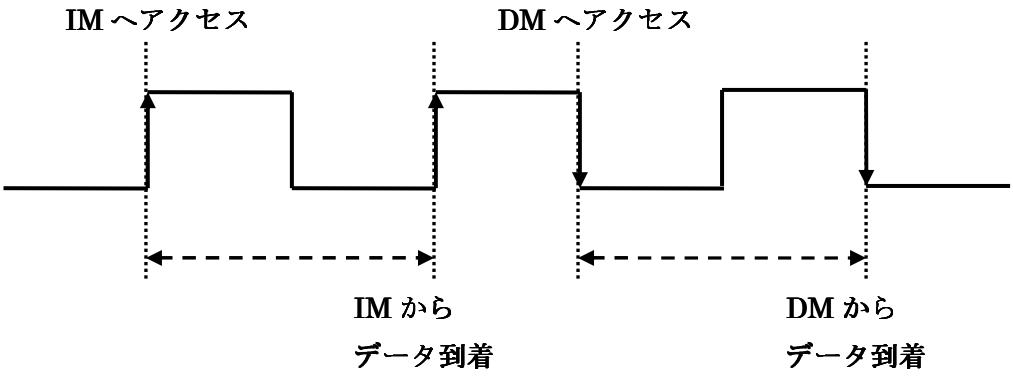

図 4：IM, DM と MPU との遅延がある場合のクロック

2.3 命令セットアーキテクチャ

MONI には、R 形式、I5 形式、I8 形式、J 形式の 4 種類の命令形式がある。図 5 に MONI の命令形式を示し、表 1 に各フィールドの意味を示す。

命令形式	フィールド				
	5	3	3	3	2
R	OP	Rs	Rt	Rd	Fn
I5	OP	Rs	Rt	Imm/Address	
I8	OP	Rs	Imm/Address		
J	OP	Target absolute address			

図 5：MONI の命令形式

表 1：命令フィールドの意味

フィールド	意味	bit 幅	用途
OP	Opecode	5	命令を識別
Rs	Source Register	3	演算元のレジスタ
Rt	Target Register	3	演算先のレジスタ
Rd	Destination Register	3	演算結果を格納
Fn	Function	2	命令を詳細に識別
Imm	Immediate	5~11	即値
Address	Address	5~8	データメモリのアドレス
Target absolute address	Target absolute address	11	ジャンプ先の絶対アドレス

R 形式の命令は、レジスタの値同士で演算を行う。Fn フィールドで、さらに詳細に演算の種類を指定できる。命令の種類は、ADD, SUB, AND, OR, XOR, SLT, SGT, SLE, SGE, SEQ, SNE, SLL, SRL, SRA, NOT である。I5 形式の命令は、レジスタの値と即値で演算を行い、また、レジスタとデータメモリ間のデータ転送を行う。命令の種類は、ADDI, SUBI, ANDI, ORI, XORI, SLTI, SGTI, SLEI, SGEI, SEQI, SNEI, SLLI, SRLI, SRAI, LD, ST である。I8 形式の命令はレジスタに即値の値を格納、レジスタの値によって分岐先にジャンプ、レジスタとデータメモリ間のデータ転送を行う。命令の種類は、LDHI, LDLI, BEQZ, BNEZ, PUSH, POP である。J 形式の命令は無条件にジャンプを行い、サブルーチンへ分岐、サブルーチンから復帰、また、何も実行されない命令や終了命令も含んでいる。命令の種類は、JUMP, CALL, RETURN, HALT, NOP である。

3. シングルサイクルプロセッサの設計と検証

3.1 シングルサイクル方式のアーキテクチャ

図 6 にシングルサイクルプロセッサのデータパスを示す。

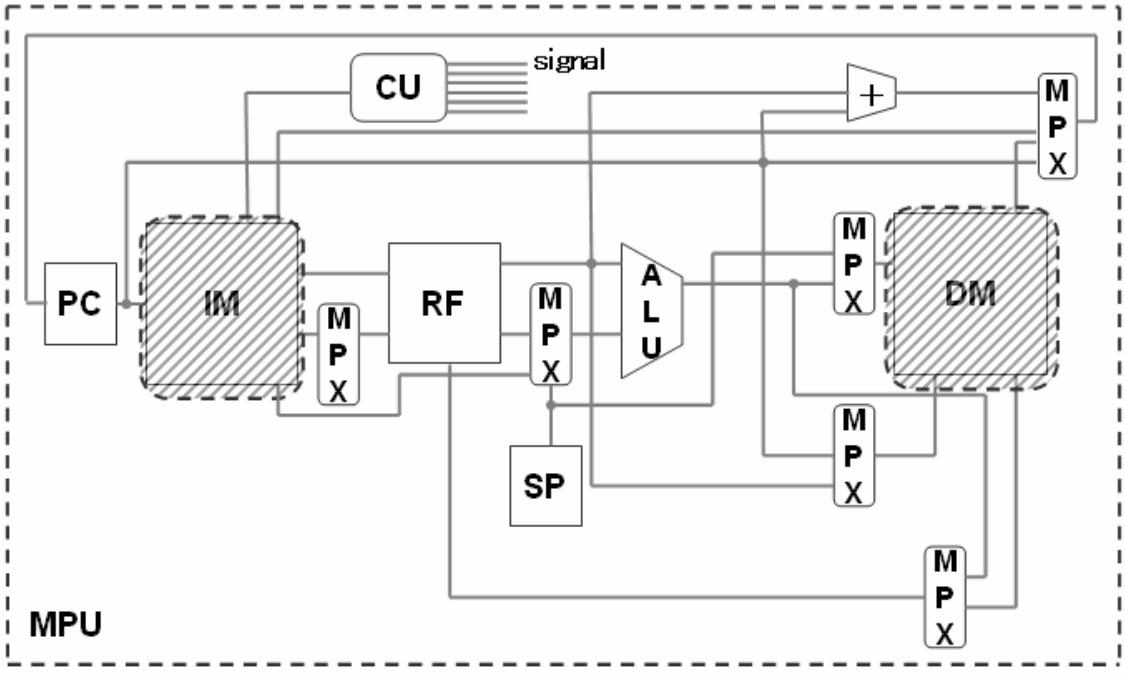

図 6：シングルサイクルプロセッサのデータパス

シングルサイクルプロセッサは、次のモジュールで構成される。網掛けになっている IM と DM のモジュールを除く部分を MPU とし、プロセッサデバッガと接続を行う。

- プログラムカウンタ(PC)

  命令メモリへアドレスを渡すレジスタ。

- レジスタファイル(RF)

  2 つのソースレジスタと、1 つのデスティネーションレジスタを指定できる。ソースレジスタは、アドレスを指定すると、クロックに非同期で出力する。デスティネーションレジスタは、クロックに同期して指定したアドレスにデータを書き込む。

- スタックポインタ(SP)

  PUSH, POP, CALL, RETURN 命令時に実行されるレジスタ。データメモリの一部分をスタックとして使用し、データメモリへアドレスを渡す。

- ALU

  算術演算、算術論理演算、論理演算、比較、ゼロ判定を行う。ゼロ判定信号は、プログラムカウンタへデータを送るマルチプレクサに送っている。

- プラス(+)

  分岐先のアドレスの計算を行い、また符号拡張も行う。制御信号は受け取らない。

● マルチプレクサ(MPX)

複数の入力から、制御信号によって出力を選択する。プログラムカウンタへ出力するマルチプレクサは、ALU からゼロ判定信号を受け取り、分岐命令時にゼロ判定制御信号も使用される。

● コントロールユニット(CU)

命令メモリから Opecode や Function を受け取り、各モジュールへ制御信号を送る。プラスモジュールへは信号を送らない。

3.2 命令の実行と制御

シングルサイクル方式では、1 クロックで 1 命令を実行する方式である。1 クロックの間で、最初のクロックの立ち上がりでプログラムカウンタを更新し、レジスタファイルは次のクロックの立ち上がりで書き込みを行う。スタックポインタもレジスタファイルと同じタイミングで更新する。データメモリの書き込みは立ち下がりに実行する。

クロック非同期の場合は、1 クロックで 1 命令を実行できる。しかし、クロック同期の場合は、2 クロックで実行する。これは、プログラムカウンタの更新と同時に命令の読み出しを行うので、プログラムカウンタの更新前の命令を読み出してしまうためである。更新後のデータが必要なため、1 クロック増やしている。

図 7 に命令メモリ、データメモリと MPU の間に 1 クロックの遅延があるシングルサイクルプロセッサの波形を示す。1 クロックの遅延がある場合は、1 命令を 4 クロックで実行することになる。プログラムカウンタは遅延がない場合と変わらず、最初の立ち上がりで更新を行う。クロック同期なので、プログラムカウンタを更新してから、次の立ち上がりから、1 クロック後に命令メモリからデータが届き、演算等が行われる。命令メモリからデータが届いてから、立下りでデータメモリに書き込みや読み込みを行う。読み込みの場合、次の立下りにデータが届く。最後の立ち上がりで、レジスタファイルとスタックポインタを更新する。

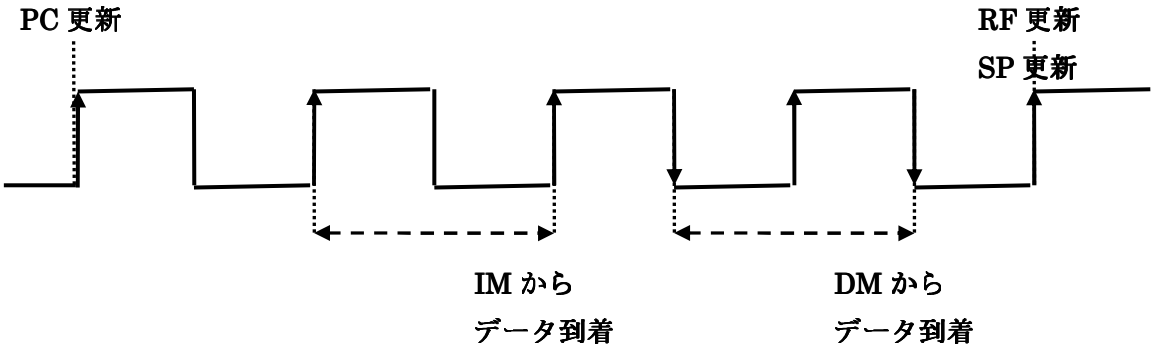

図 7：シングルサイクルプロセッサ(1 クロック遅延)のクロック

3.3 RTL シミュレーションによる検証と FPGA 上への実装

設計したシングルサイクルプロセッサ(非同期、同期、1 クロック遅延)を、Mentor Graphics 社の Xilinx Edition III v6.1e で RTL シミュレーションを行った。検証に用いたプログラムは、N までの和、素数判別、除算、三角形の判別、2 点を通る直線の方程式である。3 つのタイプのプロセッサで全てのプログラムに対し、実行結果は正しい結果を得られた。

次に FPGA 上へ実装するにあたり、論理合成を行った。論理合成ツールは、Xilinx 社提供の総合開発環境 ISE8.2i の XST を使用した。表 2 に設計規模と最高動作周波数を示す。非同期、同期、1 クロック遅延の設計規模では、ほぼ変わらない結果となった。最高動作周波数では、1 クロック遅延の場合、他と比べて 1MHz 程、速い結果が得られた。

表 2：シングルサイクルプロセッサの設計規模と最高動作周波数

プロセッサ	使用率(%)			最高動作
	スライス数	フリップフロップ数	4 入力 LUT 数	周波数(MHz)
シングル(非同期)	37	4	35	31.10
シングル(同期)	37	4	36	31.10
シングル(1 クロック遅延)	36	4	34	32.66

3.4 プロセッサデバッガを用いた検証

FPGA ボードは、Xillinx 社の Spartan-3 Starter Kit ボードを使用する。このボードには 20 万のシステムゲートがある。設計した非同期、同期、1 クロック遅延のプロセッサを、プロセッサデバッガと接続し、FPGA 上に搭載して検証を行った。検証に用いたプログラムは、N までの和、素数判別、除算、三角形の判別、2 点を通る直線の方程式である。

3 つのタイプのプロセッサの結果は、N までの和、素数判別、除算のプログラムでは正しい結果を得られた。しかし、3 つのタイプのプロセッサとも、三角形の判別、2 点を通る直線の方程式では正しい結果を得られなかった。この 2 つのプログラムには PUSH, POP, CALL, RETURN 命令を含んでいる。非同期の場合、シミュレーション上で確認した、原因と考えられる箇所を図 8 に示す。スタックポインタは立ち上がりで更新を行う。しかし、制御ユニットからの信号が、わずかに遅れて送られてくるため、更新後のデータも送ってしまう結果となる。シミュレーションと実機上での動作は、まったく同じとは言えない。同期と 1 クロック遅延の場合では遅延はないが、実機上ではうまく動作はしなかった。

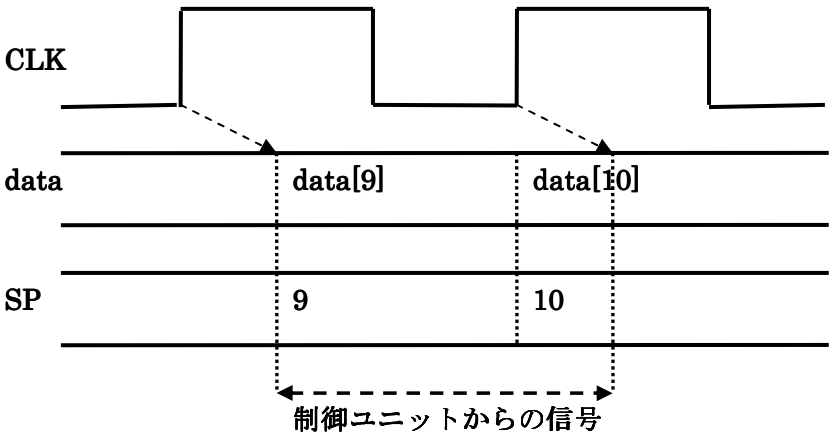

図 8：原因と考えられる箇所

4.　マルチサイクルプロセッサの設計と検証

4.1 マルチサイクル方式のアーキテクチャ

マルチサイクル方式では、1 命令を複数のフェーズに分け、それぞれのフェーズを 1 クロックで実行する。そのため、1 クロックサイクルを短くすることが可能で、処理をシングルサイクルプロセッサより高速化できる。マルチサイクルプロセッサ方式では、3 段と 4 段のマルチサイクルプロセッサを設計した。4 段マルチサイクルプロセッサはレジスタを増やすことによって、さらに 1 クロックサイクルを短くし、処理を早めることができるという考えから設計した。図 9 に 3 段マルチサイクルプロセッサのデータパス、図 10 に 4 段マルチサイクルプロセッサのデータパスを示す。

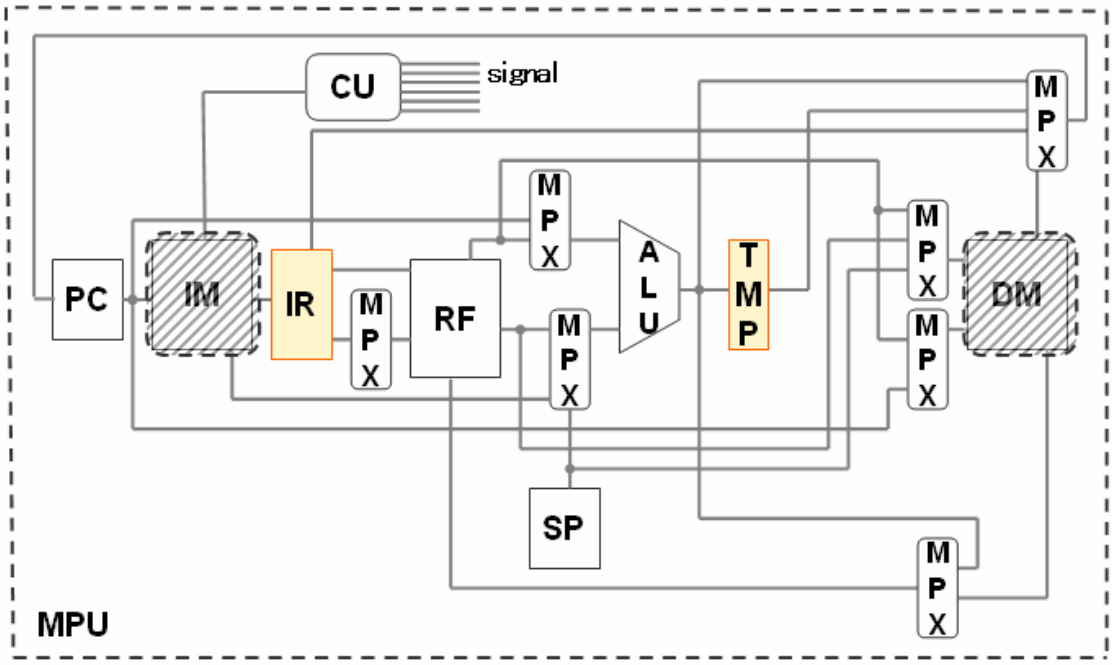

図 9：3 段マルチサイクルプロセッサのデータパス

3 段マルチサイクルプロセッサは、シングルサイクルプロセッサに以下のモジュールを追加して構成した。また、RF から ALU への出力の間でマルチプレクサを 1 個増やしている。PC, IR, RF, TMP, SP のモジュールはそれぞれ制御信号を受け取った後、クロックの立ち上がりで実行する。また、シングルサイクルプロセッサの場合と異なり、ゼロ判定信号は ALU からプログラムカウンタへ送る。

- 命令レジスタ(IR)

  命令メモリから送られてきたデータを記憶する。記憶することにより、数クロック後でも同じ命令を実行できる。

- TMP

  ALU で計算された分岐先アドレスを保持する一時的なレジスタ。条件分岐命令の時

に出力される。

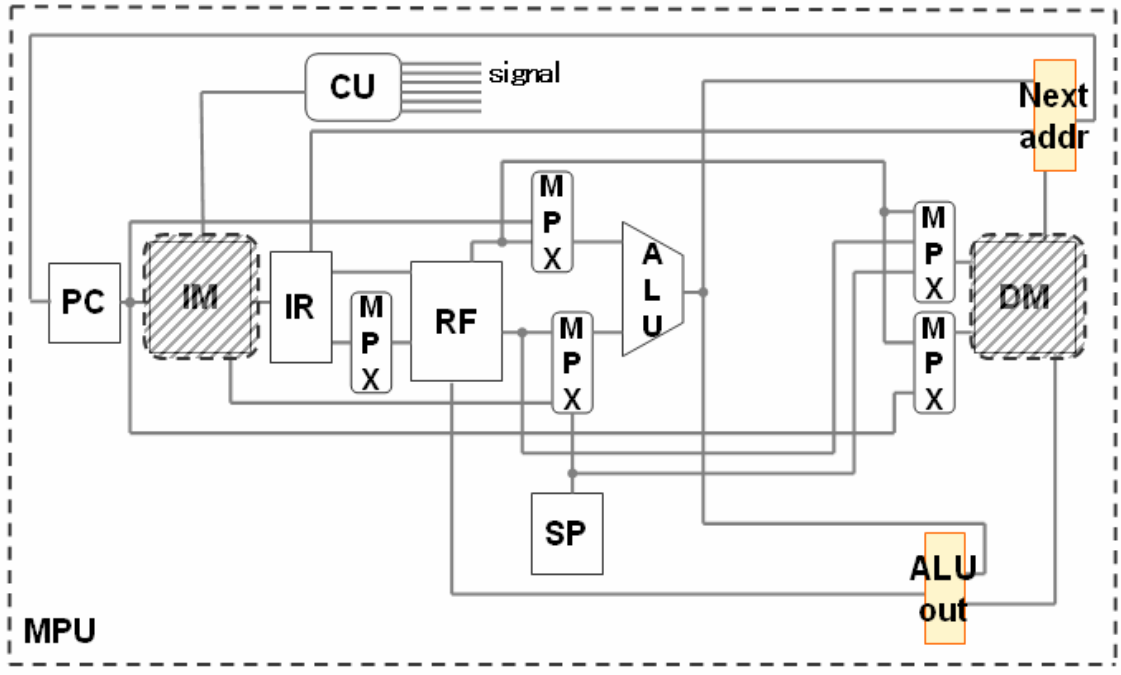

図 10：4 段マルチサイクルプロセッサのデータパス

4 段マルチサイクルプロセッサは、3 段マルチサイクルプロセッサから ALU で計算された分岐先アドレスを保持する TMP レジスタを削除し、プログラムカウンタへ出力するマルチプレクサ、レジスタファイルへ出力するマルチプレクサをそれぞれ、Nextaddr, ALUout のレジスタに変えて構成した。Nextaddr, ALUout のモジュールは、制御信号を受け取った後、クロックの立ち上がりで実行する。4 段マルチサイクルプロセッサは 3 段マルチサイクルプロセッサと異なり、ゼロ判定信号は ALU から Nextaddr へ送る。

- Nextaddr

  次の命令を指すアドレス、分岐先アドレス、ジャンプ先アドレス、CALL, RETURN 命令でのアドレスを保持するレジスタ。

- ALUout

  ALU で計算された結果や、DM から送られたデータを保持するレジスタ。

4.2 命令の実行と制御

マルチサイクル方式は 1 命令を複数のクロックで実行するため、実行中の命令を記憶しておく命令レジスタ(IR)や分岐先アドレスを記憶しておく一時的なレジスタ(TMP)の追加が必要である。

設計したマルチサイクルプロセッサはクロックの立ち上がりで制御ユニットから各モジュールに信号が送られ、フェーズ毎の実行を行う。表 3 に 3 段マルチサイクルプロセッサ

の実行サイクルを示し、表 4に4段マルチサイクルプロセッサの実行サイクルを示す。

表 3：3段マルチサイクルプロセッサの実行サイクル

命令 / フェーズ	P0	P1	P2
R	PC 更新 (PC++) 命令 フェッチ	命令デコード レジスタ 読み出し 分岐先 アドレス記憶 TMP ← PC+1 + Imm	レジスタ間の演算結果を RF に格納
I5			レジスタと即値の 演算結果を RF に格納
LDHI LDLI			RF に即値を格納
LD			DM からのデータを RF に格納
ST			RF からのデータを DM に格納
POP			SP 更新 (SP++) DM からのデータを RF に格納
PUSH			SP 更新 (SP--) RF からのデータを DM に格納
CALL			SP 更新 (SP--) PC+1 を DM へ格納 サブルーチンへ PC 更新
RETURN			SP 更新 (SP++) DM からの PC+1 に PC 更新
条件分岐命令			成り立つなら分岐先の アドレスへ PC 更新
JUMP			ジャンプ先アドレスへ PC 更新
NOP HALT			NOP：何も実行しない HALT：停止

3段マルチサイクルでは、全ての命令において、フェーズ 0 の PC 更新、命令フェッチ、フェーズ 1 の分岐先アドレスの記憶は共通である。また、全ての命令は 3 クロックで 1 命令を実行する。マルチサイクル方式では、実行中の命令が終わるまで、その命令を保持する必要があるため、命令フェッチでは命令メモリから命令を読み出し、命令レジスタに格納する。また、フェーズ 1 で制御ユニットへ命令の信号が送られる。よって、フェーズ 2 から各命令の制御信号を送り、・実行する。また分岐先アドレスを計算しておく。この段階ではどの命令を実行するかは分かっていない。分岐先アドレスを前もって計算しておいても問題はないので、条件分岐命令でなくてもアドレス計算を行う。フェーズ 2 から命令に

よって処理が異なる。

クロック非同期の場合は、R 形式、I5 形式の命令は、フェーズ 2 で演算を行い、レジスタファイルに書き込こむ。I8 形式である LDHI, LDLI 命令は、フェーズ 2 で即値をレジスタファイルに書き込む。LD や ST 命令は、フェーズ 2 でデータメモリにアドレスを送り、データメモリの書き込みや読み込みを行う。POP 命令は、フェーズ 2 でスタックポイントの更新、データメモリから送られてきたデータを、レジスタファイルに書き込む。PUSH 命令は、フェーズ 2 でスタックポインタの更新、レジスタファイルから送ったデータを、データメモリに書き込む。CALL や RETURN 命令は、フェーズ 2 でスタックポインタを更新、プログラムカウンタの更新を行い、CALL 命令はさらに、現在実行中の次の命令のアドレスをデータメモリに書き込む。条件分岐命令は、条件が成り立つならプログラムカウンタを上書きする。JUMP 命令は、無条件にプログラムカウンタを上書きする。NOP 命令は、何も実行しない。HALT 命令は、フェーズ 2 で終了し、それ以降は制御ユニットから信号を送らない。

クロック同期の場合は、CALL 命令のフェーズ 2 で、プログラムカウンタの更新と同時に、フェーズ 0 の命令フェッチを避けるため、1 クロック増やす。RETURN, 条件分岐, JUMP 命令も同じ理由で 1 クロック増やす。

1 クロックの遅延がある 3 段マルチサイクルプロセッサでは、フェーズ 0 とフェーズ 1 の間で 1 クロックの遅延がある。LD 命令のフェーズ 2 でアドレスがデータメモリに送られ、1 クロックの遅延があり、その後、レジスタファイルに書き込みを行う。POP 命令は、LD 命令と同じで、1 クロック後にレジスタファイルに書き込みを行う。RETURN 命令は、データメモリからデータが送られてくるまでに、1 クロックの遅延があり、その後、プログラムカウンタを更新する。また、クロック同期の場合と同じで、最後のフェーズでプログラムカウンタを更新した場合、1 クロック増やす。

表 4：4 段マルチサイクルプロセッサの実行サイクル

命令 / フェーズ	P0	P1	P2	P3
R	PC の記憶 Nextaddr ← PC++ 命令 フェッチ	命令デコード レジスタ 読み出し 分岐先 アドレス記憶 Nextaddr ← PC+1 + Imm	PC 更新 レジスタ間の演算結果を ALUout に格納	RF に格納
I5			PC 更新 レジスタと即値の演算結果を ALUout に格納	RF に格納
LDHI LDLI			PC 更新 ALUout に即値を格納	RF に格納
LD			PC 更新 DM からのデータを ALUout に格納	RF に格納
ST			PC 更新 RF からのデータを DM に格納	
POP			PC 更新, SP 更新（SP++） DM からのデータを ALUout に格納	RF に格納
PUSH			PC 更新, SP 更新（SP--） RF からのデータを DM に格納	
CALL			SP 更新（SP--） PC+1 を DM に格納 サブルーチンアドレスを Nextaddr に格納	PC 更新 PC ← Nextaddr
RETURN			SP 更新（SP++） DM からの PC+1 を Nextaddr に格納	PC 更新 PC ← Nextaddr
条件分岐命令			成り立つなら分岐先のアドレスへ PC 更新	
JUMP			ジャンプ先アドレスを Nextaddr に格納	PC 更新 PC ← Nextaddr
NOP HALT			NOP：何も実行しない HALT：停止	

4 段マルチサイクルでは、全ての命令においてフェーズ 0 の PC の記憶、命令フェッチ、フェーズ 1 の分岐先アドレスの記憶は共通である。フェーズ 2 から命令によって処理が異

なる。3 段マルチサイクルプロセッサでは、プログラムカウンタの更新をフェーズ 0 で行っているが、4 段マルチサイクルプロセッサでは、CALL, RETURN, JUMP 命令はフェーズ 3 で更新し、他の命令はフェーズ 2 で更新を行う。Nextaddr にプログラムカウンタへ与えるデータは格納しているので、フェーズ 2 以降で更新するようにした。

クロック非同期の場合は、R 形式、I5 形式の命令は、フェーズ 2 で演算を行い、ALUout に格納してから、フェーズ 3 でレジスタファイルに書き込こむ。LDHI, LDLI 命令は、フェーズ 2 で即値を ALUout に書き込み、フェーズ 3 でレジスタファイルに書き込む。LD 命令は、フェーズ 2 でデータメモリからのデータを ALUout に書き込み、フェーズ 3 でレジスタファイルに書き込む。ST 命令は、フェーズ 2 でレジスタファイルのデータをデータメモリに書き込む。POP 命令は、フェーズ 2 でスタックポイントの更新、データメモリから送られてきたデータを ALUout に書き込み、フェーズ 3 でレジスタファイルに書き込む。PUSH 命令は、フェーズ 2 でスタックポインタの更新、レジスタファイルから送ったデータを、データメモリに書き込む。CALL 命令は、フェーズ 2 でスタックポインタを更新、現在実行中の次の命令のアドレスを、データメモリに書き込み、サブルーチンアドレスを Nextaddr に書き込む。RETURN 命令は、フェーズ 2 でスタックポインタを更新、CALL 命令があるアドレスの次のアドレス PC+1 を、Nextaddr に書き込む。JUMP 命令は、フェーズ 2 で Nextaddr にジャンプ先のアドレスを書き込む。条件分岐命令, NOP, HALT 命令は、3 段マルチサイクルプロセッサと同じ実行サイクルである。

クロック同期の場合は、ST 命令のフェーズ 2 で、プログラムカウンタの更新と同時に、フェーズ 0 の命令フェッチを避けるため、1 クロック増やしている。PUSH, CALL, RETURN, 条件分岐, JUMP 命令も同じ理由で 1 クロック増やしている。

1 クロック遅延のある 4 段マルチサイクルプロセッサでは、フェーズ 0 とフェーズ 1 の間で 1 クロックの遅延がある。LD 命令のフェーズ 2 でアドレスがデータメモリに送られ、1 クロックの遅延があり、その後、ALUout に書き込み、次にレジスタファイルに書き込む。POP 命令は、LD 命令と同じで、1 クロック後に ALUout に書き込み、次にレジスタファイルに書き込む。RETURN 命令は、データメモリからデータが送られてくるのに 1 クロックの遅延があり、Nextaddr に書き込み、次にプログラムカウンタを更新する。また、クロック同期の場合と同じで、最後のフェーズでプログラムカウンタを更新した場合、1 クロック増やしている。

4.3 RTL シミュレーションによる検証と FPGA 上への実装

設計した 3 段マルチサイクルプロセッサと、4 段マルチサイクルプロセッサの非同期、同期、1 クロック遅延の RTL シミュレーションを行った。検証に用いたプログラムは、N までの和、素数判別、除算、三角形の判別、2 点を通る直線の方程式である。3 段と 4 段のそれぞれ 3 つのタイプのプロセッサで、全てのプログラムにおいて、正しい実行結果を得た。

表 5 と表 6 に論理合成を行った結果の設計規模と最高動作周波数を示す。設計規模では、

3段と4段も非同期、同期、1クロック遅延で、ほぼ変わらない結果となった。最高動作周波数では、3段と4段も3つのタイプで、ほぼ変わらない結果となった。

表 5：3段マルチサイクルプロセッサの設計規模と最高動作周波数

プロセッサ	使用率(%)			最高動作
	スライス数	フリップフロップ数	4入力LUT数	周波数(MHz)
3段マルチ(非同期)	46	7	44	43.71
3段マルチ(同期)	47	7	45	43.49
3段マルチ(1クロック遅延)	47	7	45	43.49

表 6：4段マルチサイクルプロセッサの設計規模と最高動作周波数

プロセッサ	使用率(%)			最高動作
	スライス数	フリップフロップ数	4入力LUT数	周波数(MHz)
4段マルチ(非同期)	42	7	41	44.35
4段マルチ(同期)	45	7	44	44.98
4段マルチ(1クロック遅延)	45	7	44	45.17

4.4 プロセッサデバッガを用いた検証

3段マルチサイクルプロセッサと4段マルチサイクルプロセッサの非同期、同期、1クロック遅延のプロセッサを、プロセッサデバッガと接続し、FPGA 上に搭載し検証を行った。検証に用いたプログラムは、Nまでの和、素数判別、除算、三角形の判別、2点を通る直線の方程式である。

実行結果は、3段と4段のマルチサイクルプロセッサで、それぞれ3つのタイプとも、全てのプログラムで正しい結果を得られた。

5. 設計したプロセッサとハード/ソフト協調学習システムの評価

5.1 プロセッサの性能比較と評価

表 2、表 5、表 6 から、シングルサイクルプロセッサ、3 段マルチサイクルプロセッサ、4 段マルチサイクルプロセッサの、それぞれ 3 つのタイプの設計規模を見ると、シングルサイクルからマルチサイクルでは設計規模は増えている。また、これはクロック毎の制御を行うためにマルチサイクルを拡張したためである。3 段から 4 段マルチサイクルでは減っている。これは TMP レジスタを削除し、2 つのマルチプレクサをレジスタに変えたためと考えられる。最高動作周波数では、シングルサイクルからマルチサイクルでは、上がっている。また、3 段から 4 段にすることによって、1Mz ほど上げることができた。シングルサイクルから 3 段マルチサイクルでは約 1.4 倍、シングルサイクルから 4 段マルチサイクルも約 1.4 倍向上する結果を得た。3 段マルチサイクルから 4 段マルチサイクルは約 1 倍という結果になった。

次に、それぞれのプロセッサの RTL シミュレーション上での実行時間を表 7 に示す。1 クロックは 20ns である。シングルサイクルから 3 段マルチサイクルではそれぞれのプログラムで約 3.0 倍のクロック数がかかり、シングルサイクルから 4 段マルチサイクルでは約 4.0 倍のクロック数がかかり、3 段マルチサイクルから 4 段マルチサイクルでは約 1.2 倍のクロック数がかかる。

最高動作周波数とクロック数の比率を比べると、シングルサイクルとマルチサイクルでは、マルチサイクルの方が遅いという結果になった。3 段マルチサイクルと 4 段マルチサイクルでもあまり、変化はなかった。しかし、3 段から 4 段へ拡張は、設計規模を減らすことができ、最高動作周波数も上げることができた。

表 7：RTL シミュレーション上でのプログラムの実行時間

プログラム / プロセッサ	実行時間(ns)				
	N までの和 N = 10	素数判別 13 の素数	除算 19 / 5	三角形判別 三辺 5,12,13	2 点を通る直線 2 点 (2,1),(8,5)
シングル(非同期)	900	4540	820	3640	4060
シングル(同期)	1800	9080	1640	7280	8160
シングル(1 クロック遅延)	3500	18060	3180	14460	16220
3 段マルチ(非同期)	2660	13580	2420	10880	12200
3 段マルチ(同期)	2860	15200	2660	11740	13460
3 段マルチ(1 クロック遅延)	3760	19740	3500	15520	17980
4 段マルチ(非同期)	3300	16440	2980	13740	15120
4 段マルチ(同期)	3520	18080	3260	14640	16660
4 段マルチ(1 クロック遅延)	4420	22620	4100	18420	21180

5.2 ハード/ソフト協調学習システムの評価

本研究で、ハード/ソフト協調学習システムを用いて、シングルサイクルプロセッサ、3段マルチサイクルプロセッサ、4段マルチサイクルプロセッサのクロック非同期、クロック同期、1クロック遅延を設計することができた。また、HDLでプロセッサの設計や、検証に用いるプログラムの他に、ハード/ソフト協調学習システムのソフトウェア学習で、階乗、二次方程式の根の判別、行列の乗算等、アセンブリ言語で作成してきたことにより、プロセッサアーキテクチャの理解はある程度できたと考えられ、プログラミング技術も身についたと考えられる。

プロセッサデバッガを用いて、設計したプロセッサをFPGA上に搭載して、作成したアセンブリプログラムを実行してプロセッサのデバッグを行った。本研究で使用したプロセッサデバッガで用いたコマンドは、命令メモリとデータメモリへのデータ送信と受信、レジスタファイルのデータの受信、通常実行である。今回、プロセッサのデバッグを行う際、使用したのはプロセッサデバッガのコマンドの通常実行、レジスタファイルの受信と、RTLシミュレーションである。シミュレーション上では正しい動作をしていたが、実機上では、正しい動作をしなかった場合があり、プロセッサのデバッグをするのに非常に時間がかった。また、データメモリのデータを受信した際、アクセスしていないところに不定値が入っている結果を得た。実行コマンドが、通常実行しかないため、実行過程を知ることができず、1命令毎に実行するコマンドがあるだけでも、デバッグの時間を減らせることができると考えられる。ブレイクポイントの設定や1命令毎に実行を行うコマンドがあれば、シミュレーションを用いず、プロセッサデバッガ上だけで、プロセッサのデバッグを行うことができると考えられる。

6. おわりに

本論文では、本研究室で開発を進めている、ハード/ソフト協調学習システムを用いて、MONI プロセッサのシングルサイクルプロセッサ、3 段マルチサイクルプロセッサ、4 段マルチサイクルプロセッサを設計した。それぞれのプロセッサに対し、さらにクロック非同期、クロック同期、1 クロック遅延の 3 タイプを設計した。RTL シミュレーション上で検証を行い、さらに、プロセッサデバッガを用いて FPGA 上に搭載して検証を行った。最後に、設計したプロセッサの評価やハード/ソフト協調学習システムの評価を行った。

今後の課題としては、プロセッサデバッガでのブレイクポイントの設定や、1 命令毎に実行するコマンドを追加して、プロセッサのデバッグを行うことがあげられる。また、命令セットを考案して、プロセッサの設計や、パイプライン等のアーキテクチャの設計があげられる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、本研究に関して様々な相談に乗って頂き、貴重な助言を頂いた難波翔一朗氏をはじめ、様々な面で、貴重な助言や励ましを下さった研究室の皆様に深く感謝いたします。

参考文献

[1] 池田修久：ハードウェア記述言語による単一サイクル/パイプラインマイクロプロセッサの設計, 立命館大学理工学部情報学科卒業論文, 2002.

[2] 大八木睦：ハードウェア記述言語による単一サイクル/パイプラインマイクロプロセッサの設計, 立命館大学理工学部情報学科卒業論文, 2002.

[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情報学科卒業論文, 2003.

[4] 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.

[5] 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.

[6] 難波翔一朗：FPGA 上での単一サイクルマイクロプロセッサの設計と実装, 立命館大学理工学部情報学科卒業論文, 2005.

[7] 藤原淳平：ハード/ソフト協調学習でのプロセッサアーキテクチャ可変な最適化コンパイラの設計, 立命館大学理工学研究科修士論文, 2005.

[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] 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, 館大学理工学研究科修士論文, 2007.

[10] 榎本雄太:ARM ライクプロセッサの設計と FPGA ボードへの実装と検討, 立命館大学理工
学部情報学科卒業論文, 2007.

[11] 池田修久, 中村浩一郎, 大八木睦, Hoang Anh Tuan, 山崎勝弘, 小柳茂：ハード/ソフト・コラーニングシステムにおける FPGA ボードコンピュータの設計, 情報処理学会第 66 回全国大会論文集, 2004.

[12] 大八木睦, 池田修久, 山崎勝弘, 小柳滋：ハード/ソフト・コラーニングシステムにおけるアーキテクチャ選択可能なプロセッサシミュレータの設計, 情報処理学会 第 66 回全国大会論文集, 2004.

[13] 大八木睦：ハード/ソフトコラーニング上でのアーキテクチャ可能なプロセッサシミュレータ(MONI シミュレータ)の使用法, 立命館大学理工学研究科, 2004.

[14] 池田修久, 中村浩一郎, Tran So Cong, 難波翔一郎：RC100 を用いた FPGA ボードコンピュータ 設計仕様書 Ver0.7.3.2, 立命館大学理工学研究科 高性能計算研究室・コンピュータシステム研究室, 2004.

[15] 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)(下), 日経 BP 社, 1999.

[16] 深山正章, 北川章夫, 秋田純一, 鈴木正國：HDL による VLSI 設計 VerilogHDL と

VHDL による CPU 設計, 共立出版, 2000.
[17] Jayaram Bhasker 著, 佐々木尚 訳 : VerilogHDL 論理合成入門 RTL 記述&ネットリストのリファレンス, CQ 出版, 2001.
[18] 飯塚肇, 甲斐宗徳 : 計算機ハードウェア, 丸善, 2001.
[19] 小林優 : 入門 Verilog-HDL 記述, CQ 出版, 2001.
[20] 並木秀明, 前田智美, 宮尾正大 : 実用入門ディジタル回路と Verilog-HDL, 技術評論社, 1996.
[21] 藤広哲也 : CPU は何をしているのか シリコンチップに秘められた脅威の世界, すばる舎, 2003.
[22] 安藤壽茂 : MYCOM ジャーナル アーキテクチャの話, http://journal.mycom.co.jp/column/architecture/, 2007/02/16.
[23] アスキーデジタル用語辞典 : http://yougo.ascii24.com/.
[24] IT 用語辞典 e-Words : http://e-words.jp/.