卒業論文

FPGA を用いたプロセッサ検証システムの設計と実装

氏　　名：　中谷 嵩之
学籍番号：　2210010149-5
指導教員：　山崎 勝弘　教授
提 出 日：　2005 年 2 月 21 日

立命館大学　理工学部　情報学科

内容概要

本論文では独自の命令形式を持つ RISC プロセッサとその機能を検証するための周辺モジュールをハードウェア記述言語 Verilog-HDL と FPGA 搭載ボード(celoxica 製 RC100 ボード)を用いて製作し、実装を行った。これからの LSI 設計の主な課題となるマルチプロセッサやプロセッサと専用ハードウェアを組み合わせたシステム LSI のため、プロセッサの動作の理解と、高位合成などの研究につながる HDL と CAD ツールの習得、プロセッサを含むシステム全体のアーキテクチャの理解が目的である。

プロセッサを含む全てのモジュールは HDL を用いて設計した。検証対象のプロセッサはシングルサイクル、パイプラインの両方式で実装し、システムを制御するシーケンサはマルチサイクルプロセッサとして実装することで、プロセッサのアーキテクチャの基本であるシングルサイクル、マルチサイクル、パイプラインの 3 方式を設計した。作成したモジュールをバスで連結して協調動作させた。HDL を用いた設計とシミュレーションだけでなく、FPGA 搭載ボードを用いた実装まで行い、実際に回路を動作させてプロセッサ及びシステムの機能を検証した。

実際に動作させて得られた情報を元に命令セットによるプロセッサの動作の違いと性能について考察した。また、シングルサイクル方式とパイプライン方式の MPU を比較することでパイプライン方式による性能の向上が確認できた。

目次

図目次

表目次

1. はじめに

1970 年代、デジタル回路の設計の初期において、LSI がシステムの各モジュールを実現していた頃においては回路設計は机上での回路図の記述によって行われ、その検証も机上での計算か実際に等価な回路を作成して行っていた。シミュレーション技術の発展と共に訪れたゲートレベル設計の時代は、回路の機能を示すゲートを並べることで設計を行った。記述されたゲートから回路を自動的に生成することで、設計の効率が上がり、その接続と回路構造をネットリストに変換することで計算機上での検証が可能になった。しかし、現在のように携帯電話や電子手帳、テレビなどのシステム自身を 1Chip 化する設計（SoC : System on Chip）がなされるようになり、回路規模が大きくなるにつれ、回路の記述量が膨大になって全体の把握が難しくなった。そのためゲート記述を用いてもミスが多くなり、ゲートによる記述だけでは論理的なミスなのか入力のミスなのかを判断することが困難で、ミスの特定と解決が困難になったため設計効率が低下した。また、シミュレーションに必要な時間が実用的な範囲を大きく超えるようになった。その結果として設計工程が要求された開発期間を満足できなくなり、より抽象度の高い回路記述方法が必要になった。

論理合成はより抽象度の高いレベルでの記述を実現するために開発された技術である。論理合成は RTL(Register Transfer Level)での記述からゲートを合成する技術で、1 行の RTL 記述から 10 以上のゲートを合成可能になり、生産性は大幅に上昇した。さらに RTL 記述を用いた論理シミュレーションが高速に実行可能になり、大規模 LSI の設計が可能になった。RTL 記述では主に HDL(Hardware Description Language)が用いられる。HDL は当初は論理合成ツールを販売する企業が独自に規定したものが使用されていたが、現在では Verilog-HDL と VHDL という 2 つの代表的な言語が使用されている。HDL の論理合成ツールと HDL シミュレータによって設計と検証が高速化し、現在のハードウェア設計を支えている。

現在及び近い将来、回路規模のさらなる増大によって回路設計が再び困難になると言われている。現在もさらに抽象度の高い高位言語による設計が研究されているが、高位言語による記述が主となっても、HDL の必要性は失われない。回路設計においては設計対象を逐次的に実行されるプログラムではなく、並列に実行されるハードウェアの集合として捕らえることが重要であり、それには HDL による設計が最適である。また、コストや制約が厳しいハードウェアでは、HDL を用いて様々な要求に適したアーキテクチャを記述することが最も重要となっている。

以上のような背景を踏まえ、本研究では HDL によるマイクロプロセッサの設計と FPGA への実装と検証を行った。検証対象の MPU として独自の命令セットを持つ 16bit RISC プロセッサを設計し、シングルサイクルとパイプラインの 2 つのアーキテクチャで実装した。また、MPU を動作させるためのメモリや入出力、MPU の動作を観測するモジュールも作成し、全てを組み合わせることでプロセッサ検証システムを実現した。システム全体の制御のために ROM 内のプログラムで動作する 8bit マルチサイクルプロセッサを設計し実装した。FPGA を用いて MPU をプロセッサ検証システムで動作させ、その結果を比較検討した。

この研究において実際に HDL を用いたトップダウン設計を行うことで、HDL と高位合成ツールやシミュレータ等 CAD ツールの習得、プロセッサとその周辺モジュールのアーキテクチャの理解を目的とする。プロセッサと専用ハードウェアから構成されるシステム LSI が一般的になり、プロセッサは現在ほぼ全てのディジタルシステムで用いられている。今後のマルチプロセッサコアによる並列処理やユビキタス時代におけるシステム設計の研究のため、プロセッサの構造と動作について理解を深め、ハードウェア設計の基礎を習得する。また、それによって様々なハードウェアの設計とプロセッサの動作について深く理解することを目的とする。さらに、実際にプロセッサ検証システムとして FPGA 上に実装することで、プロセッサを含むディジタルシステム全体のアーキテクチャの理解を深め、FPGA の利用方法の理解に努める。

第 2 章では作成したプロセッサ検証システムの動作とプロセッサのアーキテクチャについて述べ、第 3 章ではプロセッサの各周辺モジュールについて説明を行う。第 4 章では FPGA を用いたシステムの実装について述べ、検証結果に対する考察を行う。第 5 章では構成を行った KUE-CHIPII 検証システムとの比較を行う。

2. プロセッサ検証システムの構成

2.1 HDL を用いたトップダウン設計と FPGA による検証

HDL を用いたトップダウン設計フローは図 1 のようになる。

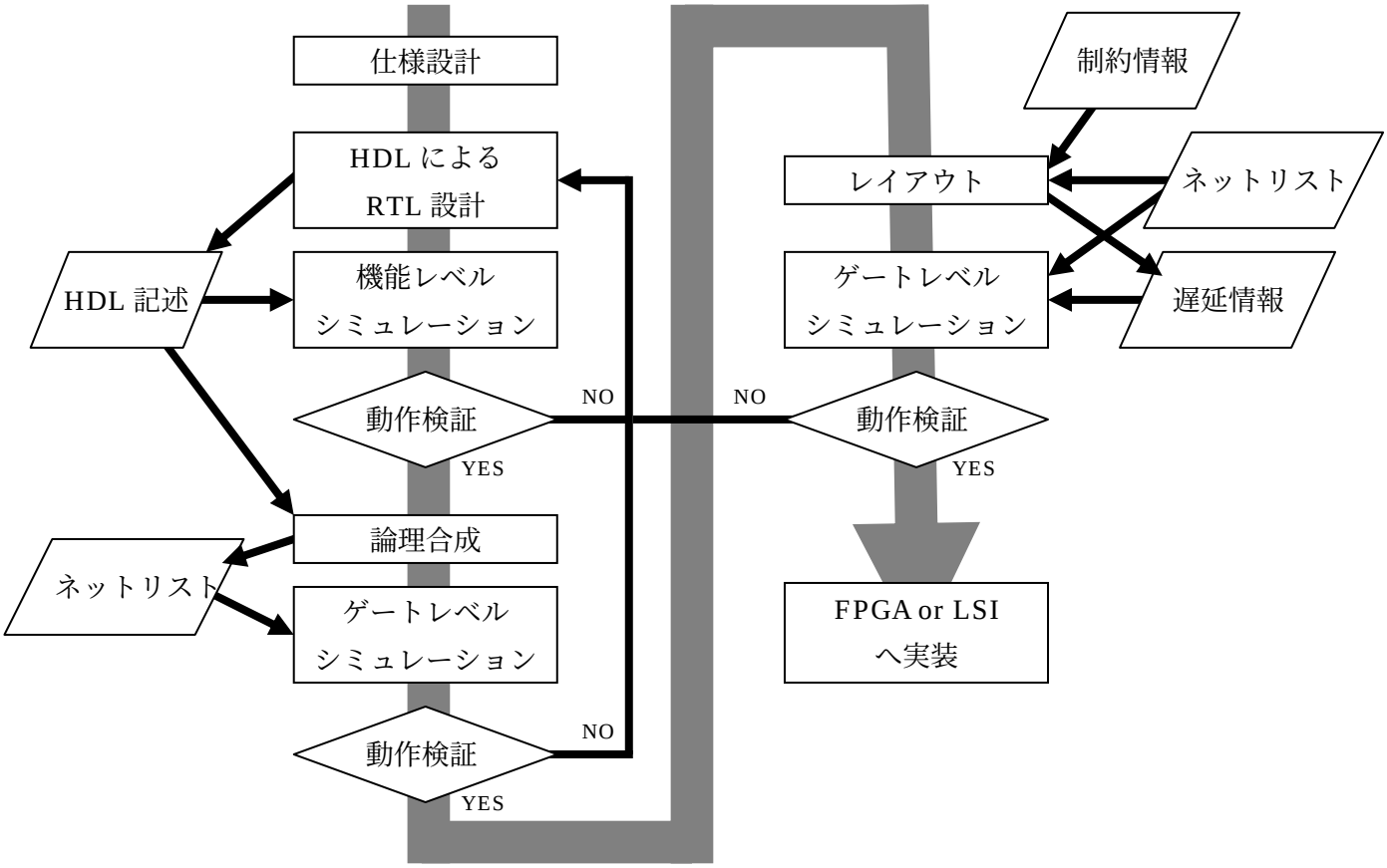


図 1：トップダウン方式による HDL 設計フロー

HDL を用いたトップダウン設計ではまず要求仕様から HDL による RTL 記述を行い、それを機能レベルでシミュレーションする[7]。シミュレーション結果が正しかった場合、RTL 記述を論理合成してネットリストを生成する。次にネットリストを用いてゲートレベルのシミュレーションを行い、仕様を満たす機能を有していれば、ネットリストと制約情報からレイアウトを行って遅延情報を得る。最後にネットリストと遅延情報から実際に LSI や FPGA に実装を模したシミュレーションを行う。各シミュレーションで間違いが発見されれば、HDL による記述を修正し、機能レベルのシミュレーションからやり直す。

トップダウン方式の設計において、シミュレーションは回路の機能が正常かどうか、回路が仕様を満たしているか確かめるために非常に重要である。しかし工程が進み、下位に移行するにつれてシミュレーションに要する時間が増大する。最下位レベルの実装を模したシミュレーションは大規

模な回路では数時間から数日かかることもある。そのため、最近では FPGA を用いて実現した等価回路による検証が盛んに行われている。

FPGA は Field Programmable Gate Array の略で、出荷後にに回路構成をプログラム可能な LSI のことであり、開発現場や製品への実装後に内部の機能を自由に変更することが可能である。同様の機能を有した回路に CPLD がある。CPLD は Complex Programmable Logic Device の略で、FPGA と同様にその機能を出荷後に機能を変更できるが、実現できる回路の構成と規模において FPGA よりも劣っている。FPGA は一般的には図 2 に示すように論理ブロックをアレイ状に並べて構成されており、論理ブロックの機能とそれらの接続情報を外部からダウンロードすることで機能を変更することが可能である。世界中に多数の FPGA ベンダーが存在し、回路構造が異なる様々な FPGA を発売している。今までの FPGA は動作速度が ASIC と比べて低速であり、価格も高価であることから、何度も書き換え可能であることを利用して主に回路の検証に用いられていた。しかし、プロセスの微細化技術と大量生産によってゲート数あたりのコストが減少し、少量生産のシステムの実装にも用いられている。

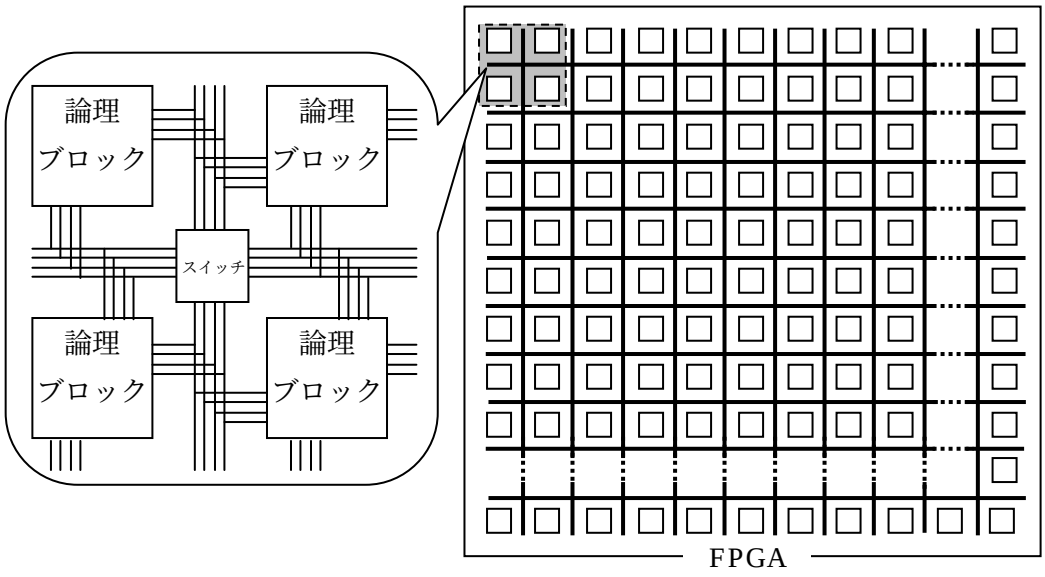


図 2：一般的な FPGA の内部構成

現在の回路の検証においては主に、同等の回路を作成する手法と、計算機上でのシミュレーションを行う手法が存在する。しかし、同等の回路を作成するにはコストと時間がかかり、機能や設計要求に問題があった場合に工程を遡ることが難しかった。計算機上でのシミュレーションは詳細なデータを取ることができるが、回路規模が増加するに従って必要時間が著しく増加し、大規模な回路では検証が現実的ではなかった。FPGA を用いると等価な回路が簡単に作成可能なので、設計フローの上流での機能検証はシミュレーションで行い、時間がかかるレイアウト後のゲートレベルシミュレーションは FPGA 上で回路を動作させることで、計算機上とは桁違いに高速なシミュレーションが可能になった。さらに構成を変更可能であるという FPGA の特徴を利用して、モジュ

ール内部の配線などを直接外部から観測することが可能で、同等の回路を作成したときよりもエラー箇所の特定が容易になるという利点もある。

最近では検証や研究だけではなく実際の製品にも FPGA が利用されている。今まで LSI は少品種大量生産が主であったが、アプリケーションの多様化とデザインの寿命が短くなったことにより、多品種少量生産になりつつある。LSI の設計に必要なコストはプロセスの微細化と回路規模の増加に伴って非常に高くなり、医療機器などの少量の生産では採算がとれなくなっている。このような場合、コストに見合うように価格を高くするしかなかったのだが、大量生産によって価格が下がった FPGA を用いることで、少量生産しかされないシステムでも低コストで実現が可能になった。

また、FPGA は IP、Intellectual Property と呼ばれる設計資産の利用が容易で、これを利用することで設計期間の短縮を図ることが可能である。実装後も機能の変更が可能なので、テクノロジやトレンドの変化に従って頻繁に仕様が変更されるシステムにも FPGA が用いられている。FPGA の回路規模が増えるに従い、FPGA にも IP やハードウェアとして CPU が内蔵されるようになった。これはほぼ全ての電化製品にプロセッサが搭載され、ソフトウェアによる制御がなされている現在の状況を反映している。

2.2 システム構成

プロセッサ検証システムにおいて検証の対象となるプロセッサとしてシンプルな MPU を作成した。この章では MPU を含むシステム全体の動作と MPU のアーキテクチャについて述べる。

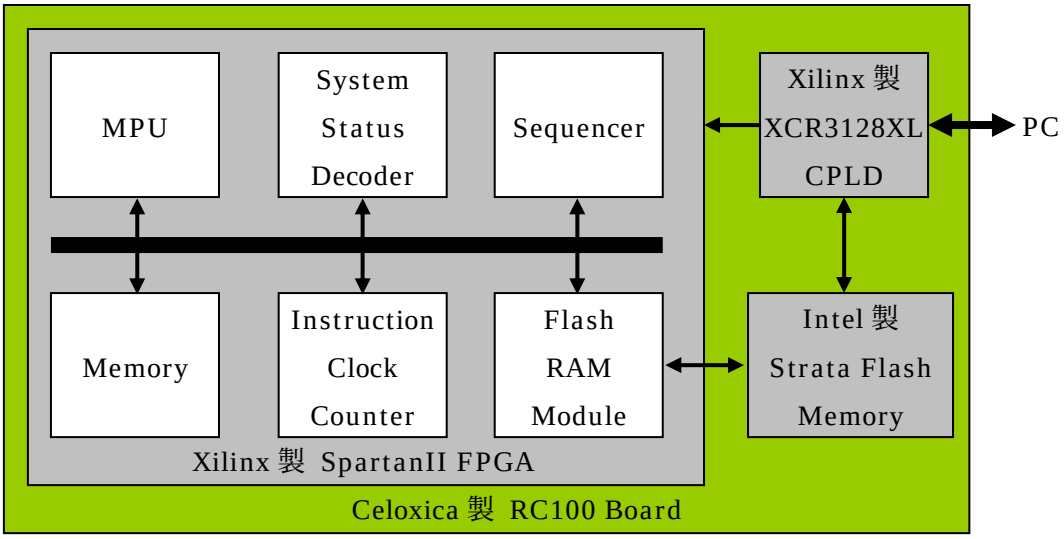


図 3：プロセッサ検証システムの構成

Xilinx 製 SpartanII FPGA、Xilinx 製 CPLD、Intel 製 Strata Flash Memory は Celoxica 製 RC100 ボードに搭載されている LSI である。FPGA 内の MPU、System Status Decoder、Sequencer、Memory、Instruction Clock Counter、Flash RAM Module の各モジュールを HDL によって作成、実装を行った。

MPU は検証対象のプロセッサ、Sequencer はシステム全体を制御するマルチサイクルプロセッサである。System Status Decoder は Sequencer の出力する値に従って各モジュールに制御信号を与え、Instruction Clock Counter は MPU の動作を監視してデータを集める役割を持つ。Memory は MPU の命令とデータを保持し、Flash RAM Module は Flash Memory のアクセスに用いる。

2.3 システムの動作

MPU を含むプロセッサ検証システムは以下のような手順でプロセッサの検証を行う。

2.3.1 FPGA プログラム

FPGA のコンフィグレーションを行い、各モジュールを FPGA の中に実装する。コンフィグレーションは FPGA へ回路のデータを送り、FPGA 内に回路を構成する段階である。コンフィグレーションに必要なデータは PC のパラレルケーブルからボード上の CPLD を通じて行われる。Sequencer 内の ROM はこの時に書き込まれ、変更されることはない。

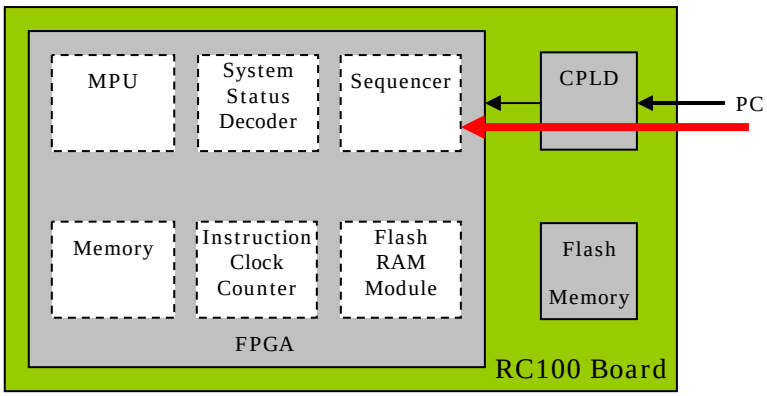


図 4：FPGA のプログラム

2.3.2 Flash Memory からのプログラムの読み出し

次に Flash Memory から FPGA 上の Memory へプログラムを転送する。アドレスの指定は FPGA 内のアドレスバスを通じて Sequencer が行い、Flash RAM Module の制御は System Status Decoder を通して Sequencer が行う。転送するプログラムの大きさは Sequencer のプログラムで指定する。プログラムの転送は Flash RAM Module からデータバスを通じて行われ、他のモジュールはデータバスへの出力しないように System Status Decoder によって制御される。Flash Memory から読み出しが終了する毎に Flash RAM Module から動作終了信号が出力され、それを確認してから Memory へ書き込む。

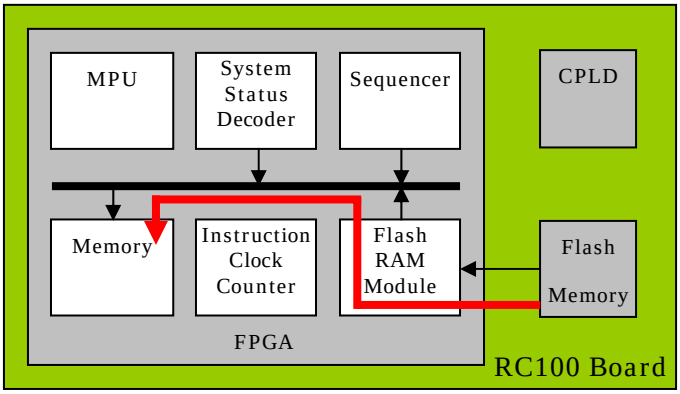


図 5：Flash Memory からのプログラムの読み出し

2.3.3 Flash Memory からのデータの読み出し

2.3.2 と同様にデータを Flash Memory から Memory へ転送する。

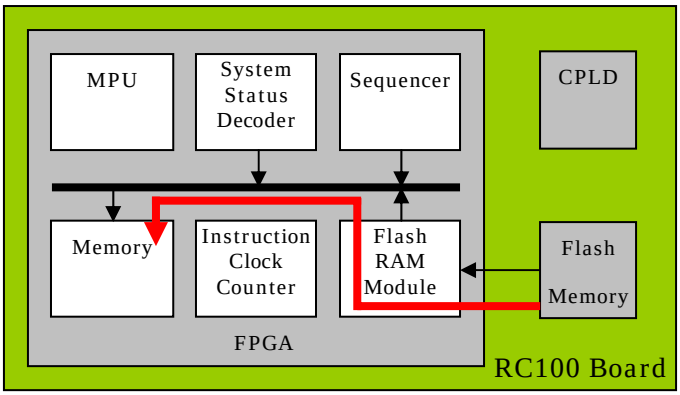


図 6：Flash Memory からのデータの読み出し

2.3.4 MPU によるプログラムの実行

MPU によってプログラムが実行される。MPU と Memory の間でのみデータのやりとりが行われ、他のモジュールは動作に一切関与しない。Sequencer は MPU からの終了信号の出力を監視する。

Instruction Clock Counter は MPU の命令フェッチを監視し、それぞれの命令が何度フェッチされたかを数える。

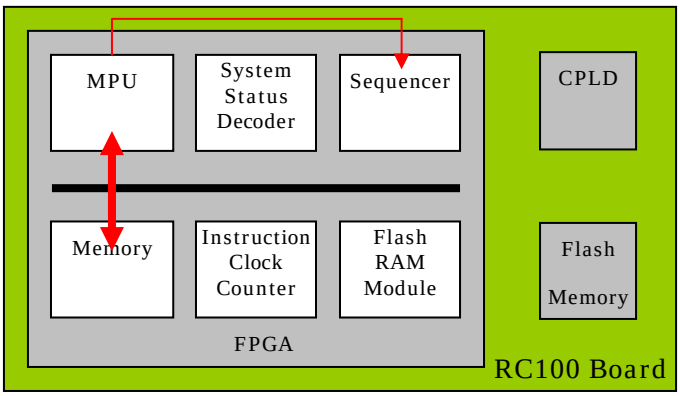


図 7：MPU によるプログラムの実行

2.3.5 Flash Memory のデータの消去

MPU の動作終了後、Flash Memory にデータを書き戻すためにまず Flash Memory 内のデータを消去する。その後、Flash Memory が書き込み可能な状態になるまで一定時間待機する。消去する Flash Memory のアドレス指定と Flash RAM Module の制御は System Status Decoder を用いて行う。

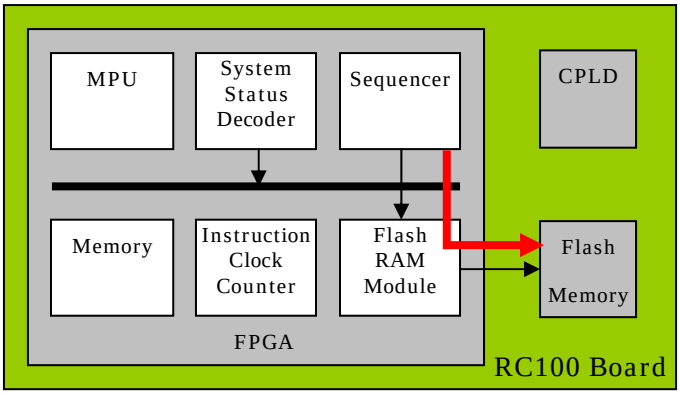


図 8：Flash Memory のデータの消去

2.3.6 Flash Memory へのデータの書き込み

最後に Memory 内のデータと Instruction Clock Counter の内容を Flash Memory へ書き戻す。2.3.2 や 2.3.3 と同様に Flash RAM Module や他のモジュールの制御は System Status Decoder が行い、アドレスの指定はアドレスバスを通じて Sequencer が行う。まず実行結果がデータバスと Flash RAM Module を通じて FPGA 内の Memory から Flash Memory へ書き込まれる。次に、Instruction Clock Counter の内容が Flash Memory へ書き込まれる。

Flash Memory 内のデータを PC から読み出すことで、プログラムの実行結果と検証に必要なデータを得ることができる。

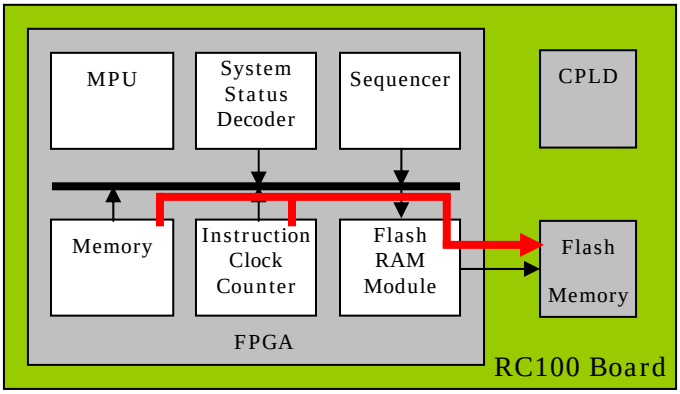


図 9：Flash Memory のデータの書き込み

2.4 MPU のアーキテクチャ

2.4.1 MPU のデータパス

MPU のデータパスを図 10 に示す。シングルサイクルの場合、MPU は図 10 のような構成を取る。パイプラインの場合は図 11 に示す。

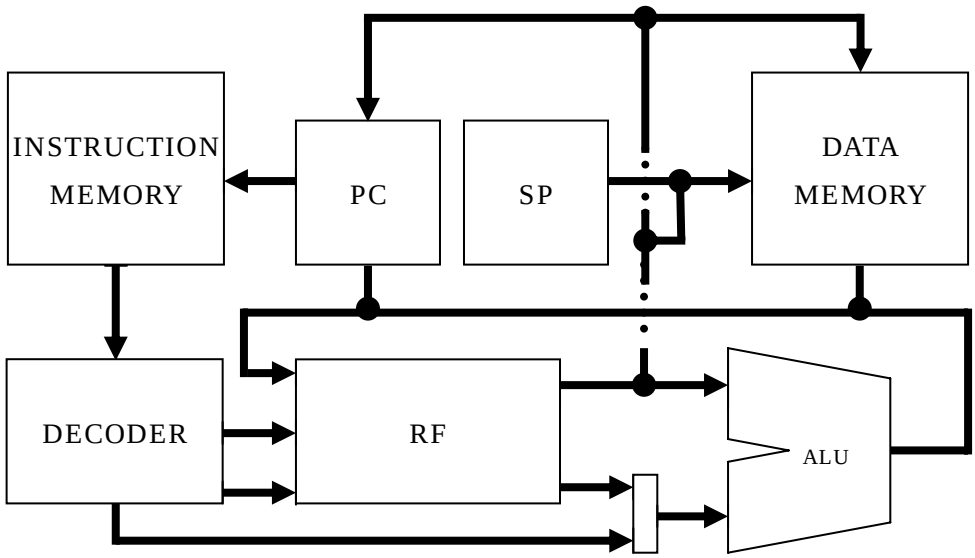


図 10：MPU のデータパス

MPU は以下のモジュールによって構成されている。INSTRUCTION MEMORY と DATA MEMORY は MPU の外部に実装されるメモリである。

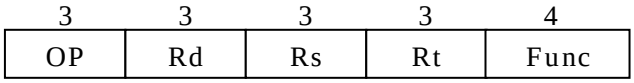
- PC：プログラムカウンタ。INSTRUCTION MEMORY のアドレスを示す。
- SP：スタックポインタ。DATA MEMORY にスタックのアドレスを与える。
- DECODER：メモリから与えられた命令を解釈し、各モジュールに制御信号を渡す。
- RF：レジスタファイル。変数や演算の途中結果を保持する。ALU のオペランド、DATA MEMORY に対するアドレスとデータ、PC を更新する値を出力する。
- ALU：算術演算と論理演算、比較演算、シフト演算を行う[8]。

MPU はハーバード・アーキテクチャの 16bit RISC プロセッサである。命令長が 16bit であるのは、回路規模を FPGA で実装可能な範囲に抑えるためであり、RISC プロセッサを選択したのはパイプライン方式のプロセッサ作成を考慮したからである。

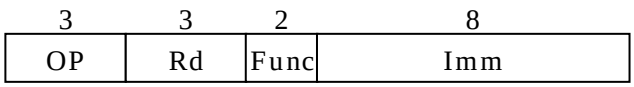
2.4.2 MPU の命令セット

MPU は以下のような単一命令長の命令フォーマットを持つ。単一命令長の命令は命令のデコードが容易で、高速動作するパイプラインに適している。[1, 2, 3]。

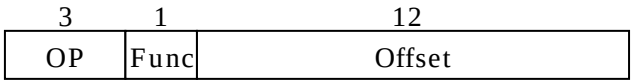
- R 命令



- I 命令



- J 形式



命令フィールド

- OP：オペレーションコード
- Rd：デスティネーションレジスタ
- Rs, Rt：ソースレジスタ
- Func：ファンクションコード
- Imm：命令の中で値として使われる即値データ
- Offset：アドレスのオフセット値

MPU は以下のような命令を持つ。

- 算術演算命令：ADD、SUB、ADC、SBC
- 論理演算命令：AND、OR、XOR、NOR
- シフト演算命令：SRL、SLL、SRA、SLA
- 比較命令：SLT
- 即値命令：ADI、SBI、LIL、LIH
- 条件分岐命令：BRZ、BNZ
- メモリアクセス命令：LD、ST、LR
- スタック命令：POP、PUSH
- ジャンプ命令：JR、JMP、JAL
- 内部割り込み命令：ITP
- 制御命令：NOP、HLT

MPU はシングルサイクル方式の場合、全ての命令を単一のサイクル(4 クロック)で実行する。MPU は多くの RISC プロセッサと同じくメモリを読み書きするのはメモリ命令のみの Load/Store

マシンであり、他の命令は全てレジスタ間の演算命令もしくはプログラムカウンタに対する操作である。

シングルサイクル方式の MPU の Verilog-HDL による記述量は約 360 行で、回路規模は 4,988 ゲート相当である。最高動作周波数は 39MHz であるが、CPI が 4 なので、実際の性能は 10MIPS ほどである。

2.4.3 パイプライン方式の MPU

パイプライン方式の MPU は図 11 のような構成を取る。

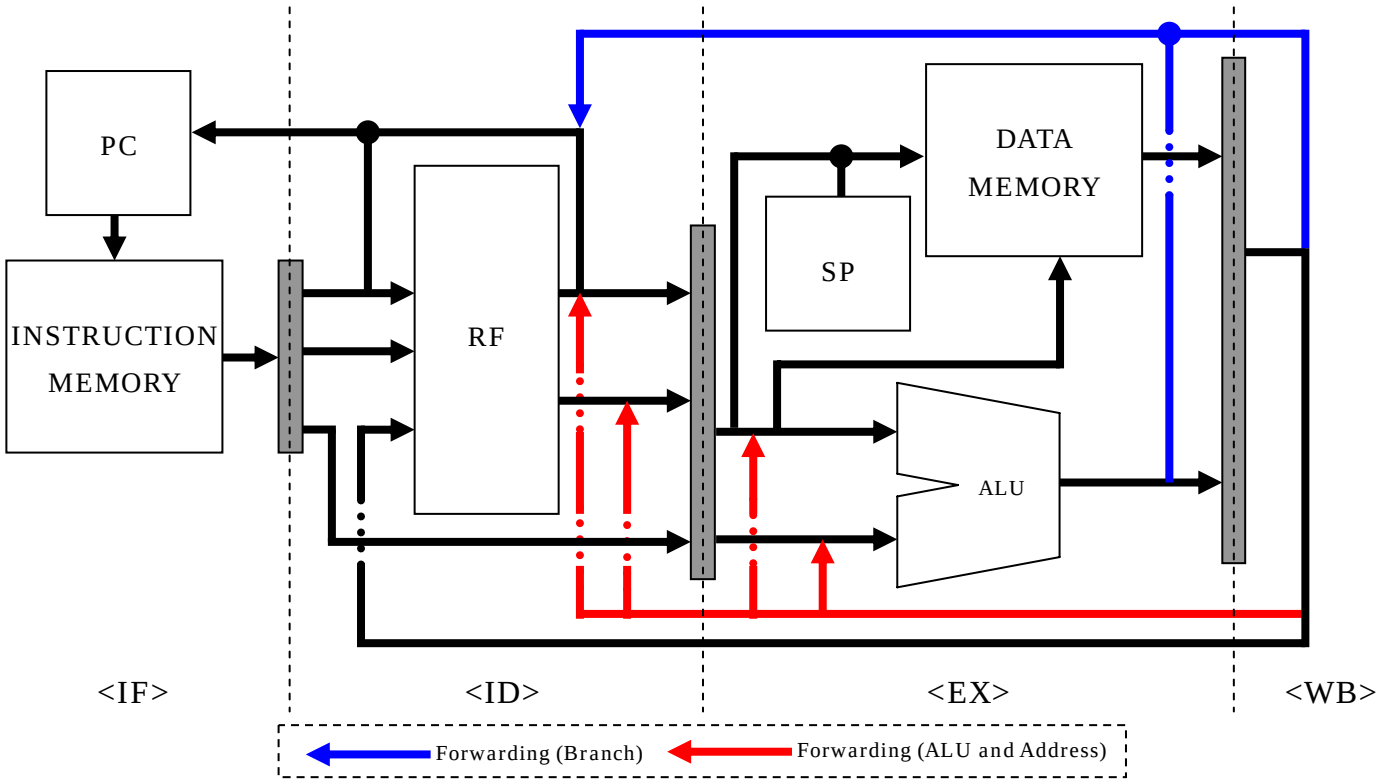


図 11：パイプライン方式 MPU のデータパス

パイプライン方式の MPU は、シングルサイクル方式のデータパスにパイプラインレジスタを挿入して構成する。データパスは IF、ID、EX、WB の 4 ステージに分かれている。
パイプライン方式の MPU の Verilog-HDL による記述量は約 410 行で、回路規模は 8,007 ゲート相当、最高動作周波数は 66MHz である。

(1) IF ステージ

IF ステージでは PC の示す値に従って INSTRUCTION MEMORY から命令を取り出し、パイプラインレジスタへ格納する。今回は同期式のメモリを用いたため、メモリの出力がそのまま

パイプラインレジスタの値となっている。PC の値は次の ID ステージによって変化する。ID ステージがパイプラインレジスタから分岐命令を読み出し、分岐が発生する場合は、ID ステージから送られるアドレスと PC の内部の値を加算した値を INSTRUCTION MEMORY へ出力する。ID ステージがジャンプレジスタ命令を読み出す場合は ID ステージから送られるアドレスを INSTRUCTION MEMORY へ出力する。ID ステージが相対アドレスジャンプ命令を読み出す場合、命令中の値と PC 内部の値を加算したものを出力する。そのいずれでもない場合、2 を加算したものを出力する。また、ジャンプ&リンク命令はジャンプした次の命令のアドレスをレジスタに保存するため、PC に 2 を加算した値もパイプラインレジスタへ格納する。

(2) ID ステージ

ID ステージでは命令レジスタから命令を読み込み、次の PC を決定すると共に、レジスタファイルから値を読み出して、パイプラインレジスタへ格納する。また、読み出したレジスタの番地や結果を書き込むレジスタのアドレス等もパイプラインレジスタへ格納する。読み出したレジスタの値を元に分岐の実行も判定する。そのために 2.4.3.5 のフォワーディングを EX ステージと WB ステージから行う。さらに、命令中の即値命令を EX ステージに高速に実行するため、命令に合わせたビット幅の拡張を行う。

(3) EX ステージ

EX ステージでは演算の実行と DATA MEMORY に対するメモリアクセスを行う。演算の実行は各演算命令の演算を ALU で行う。メモリアクセスはレジスタから読み出した値かスタックポインタの値をアドレスとして出力し、命令によってデータの書き込みか読み出しを行う。結果はパイプラインレジスタに格納される。ただし、メモリアクセスの場合は、INSTRUCTION MEMORY と同様に DATA MEMORY も同期式メモリを用いているため、DATA MEMORY の出力が直接パイプラインレジスタになっている。ALU の演算結果は分岐命令用に ID ステージへ送られる。

(4) WB ステージ

WB ステージでは結果をレジスタへと書き戻すだけのステージである。演算結果のデータやメモリアクセスによって読み出したデータは ID ステージに送られる。そして、EX ステージがパイプラインレジスタに書き込んだ値を元にレジスタファイルへのデータの書き戻しが実行される。このデータは分岐命令のために ID ステージへ、演算命令のために EX ステージへ送られる。

(5) フォワーディング

パイプライン方式は命令が IF、ID、EX、WB の各ステージを順に通過しながら実行される。この場合、前の命令による結果を後の命令が使用しているとき、後の命令の実行に前の命令のデータが間に合わないことがある。

例えば演算を実行するときの問題を考える。加算命令の次の命令がその加算結果に対するシフトであった場合、データが間に合わずハザードが起きる。前の加算命令がフェッチされたとき、次のシフト命令はまだメモリの中である。次のクロックでは加算命令が ID ステージへと進み、演算に必要なオペランドをレジスタファイルから読み出す。シフト命令は IF ステージで読み出される。次のクロックでは、加算命令が EX ステージで実行されて、その結果がパイプラインレジスタへ格納される。シフト命令は ID ステージでデコードされ、使用するレジスタの値を読み出すが、このとき読み出す値は前の加算命令の結果の筈であるが、前の命令は EX ステージにあり、WB ステージが実行されていないため、加算した結果がレジスタファイルへ格納されていない。よって、シフト命令が ID ステージで読み出したレジスタファイルの値はプログラムが本来指定した値ではない、即ち間違った値である。次のクロックで加算命令が WB ステージに入り、結果がレジスタに格納されるが、この時には次のシフト命令が EX ステージに入っており、間違った値のまま演算が実行される。また、さらに次の命令でも加算結果を用いる場合、パイプラインレジスタへ格納されるのとレジスタファイルへ格納されるのが同時のため、パイプラインレジスタへは加算結果が反映されず、さらに間違った演算が実行される。前の命令の演算結果を後の命令で用いる場合、演算の正当性を保証するためには演算結果を前のステージへ送る必要がある。これが演算に対するフォワーディングであり、図 11 の赤の矢印で示されている。この場合、WB ステージにある演算結果が EX ステージへ送られるため、もし EX ステージの命令が WB ステージの演算結果を用いる場合でも正常な演算結果が得られる。さらに ID ステージに WB ステージの演算結果を用いる命令があったとしても、ID ステージのパイプラインレジスタへ書き戻された最新の値が格納され、正常な演算が行われる。

分岐に対するフォワーディングは同様に演算結果を ID ステージへ送るものである。しかし、フォワーディングが不可能な場合が存在する。分岐命令が 2 個前の命令による結果を分岐の判定に用いる場合、演算に対するフォワーディングと同様に WB ステージからフォワーディングが可能である。しかし、1 つ前の命令の結果を判定に用いる場合、問題がある。一つ前の命令が ALU を用いる演算命令なら、ALU の演算結果をフォワーディング可能である。しかし、メモリを読み出す命令の場合、メモリの内容はステージの最後に得られるため、内容が得られると同時に分岐の判定が終了している。これではフォワーディングが不可能である。そこで、メモリを読み出す命令の後で、読み出した値を分岐の判定に使用している場合のみ、ストールさせて MPU を停止させる。ストールによって停止するのは IF ステージと ID ステージである。ストールを起こすと、ID ステージのパイプラインレジスタには何もしない命令が格納され、EX ステージのメモリを読み出す命令の結果が得られる。そうすることで、WB ステージからのフォワーディングが可能になる。またストールした場合でも WB ステージが正常に動作するように、レジスタファイルの書き込みは停止しない。今回の MPU はパイプラインが 4 段であるため、ストールが発生する条件が少ないが、一般的にパイプラインの段数が増加するに従い、ストールによるプロセッサの停止も増加する。

3. プロセッサ周辺モジュールの作成

FPGA ボード上にて実装したプロセッサ検証システムは図 3 に示した構成をとる。
この章では MPU を検証するために必要な各モジュールについて述べる。

3.1 Flash RAM Module

Flash RAM Module は Celoxica 製 RC100 ボードに搭載されている Intel 製 Strata Flash Memory を制御するモジュールである。Flash Memory はメモリ内容の読み出し、書き込み、消去にそれぞれ一定のステップが必要で、非同期に動作するため、モジュールは 2 段階の有限状態機械で構成されている。図 12 に各動作に必要なステップを示す。

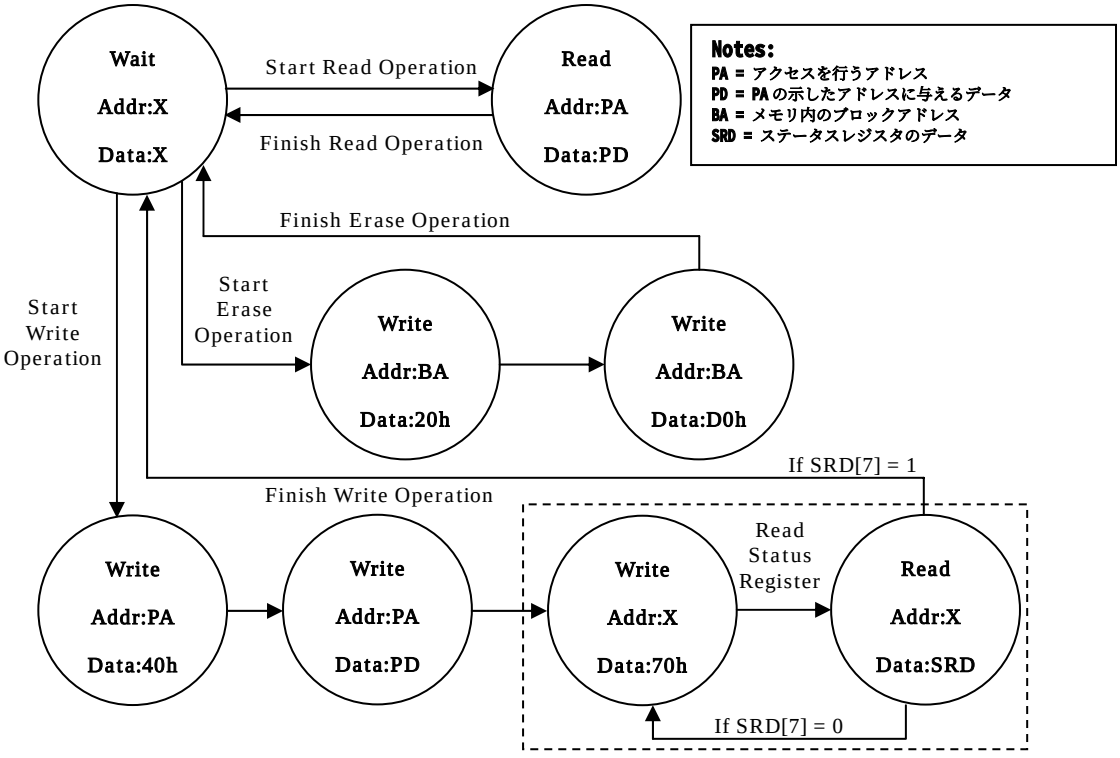


図 12：Flash Memory のオペレーション

各動作に必要なステップは Flash Memory への Read/Write である。図 12 の各ノードは必要なステップとその時のアドレスとデータを示している。

Read オペレーションは、単一の Read ステップで実行される。Read ステップで指定したアドレスのデータが Flash Memory から出力される。

Erase オペレーションは 2 つの Write ステップで実行される。2 つの Write ステップでデータバスから消去のためのコマンドを入力し、アドレスバスから消去するブロックのアドレスを入力する。

Write オペレーションは最も複雑で、4 つ以上のステップで実行される。まず Write ステップでデータバスから書き込みのためのコマンドを入力し、アドレスバスで書き込み先のアドレスを入力する。次の Write ステップではアドレスはそのままで、データバスから書き込むデータを入力する。3 番目と 4 番目のステップでは、書き込みが正常に行われたかを確認する。まず Write ステップでデータバスからステータスレジスタの読み込みのためのコマンドを入力する。アドレスバスは不定(X)となっていて、これは任意の値であることを示す。アドレスが不定なのは、コマンドがステータスレジスタに対する命令なので、通常のアドレス空間には存在せず、アドレス指定が必要ないからである。次の Read ステップでは、ステータスレジスタの内容を読み出す。1 つ前の Write ステップでステータスレジスタの読み出しコマンドを入力しているため、アドレスバスの入力は不定である。ステータスレジスタには Flash Memory の状態が格納されており、書き込みが成功した場合 8 ビット目が 1 になる。それを確認すると、Write オペレーションは終了する。確認できない場合、8 ビット目の 1 が確認できるまで 3 番目と 4 番目のステップを繰り返す。

各ノードの Read/Write ステップにおいても、Flash Memory は動作が非同期なので、要求されたタイミングに合わせて制御信号を変化させなければならない。信号を変化させるタイミングは Flash Memory の仕様書の記述に従い、ボードに搭載されている水晶発振子が 80MHz であることに基づいて図 13 のように決めた。仕様書の内容は記述量が多いため省略する。図 13 の左側の CLK 以外は Flash Memory との入出力である。Flash RAM Module は 40MHz(25ns)で動作するようにし、それに合わせて各制御信号の変化のタイミングが使用書に沿うように規定した。

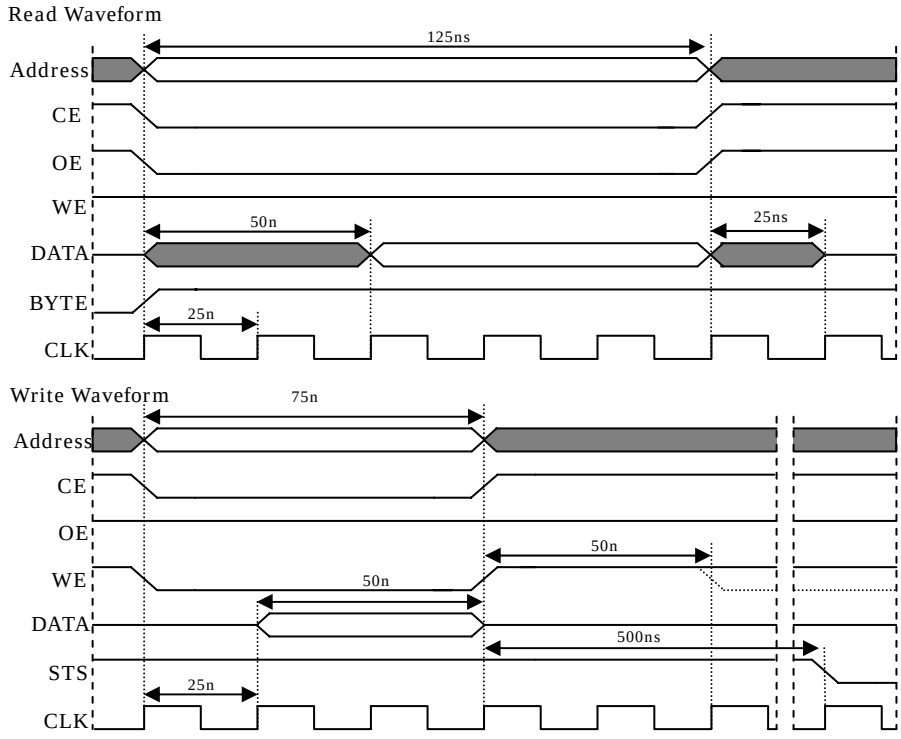


図 13：Flash Memory の Read/Write 波形図

Flash RAM Module の動作は、オペレーションとステップの 2 段階の状態遷移を行うので、モジュールも 2 段階の有限状態機械として作製した。構成を図 14 に示す。

最も上位の Interface 部はバスに接続され、読み出し、書き込み、消去の各オペレーションを開始する信号を受け取る。また、受け取った際にその時点でのバスの値を内部でラッチし、各動作中にバスを占有しない。Interface 部は各オペレーションの開始信号を Operation Control 部へ出力し、オペレーションが終了した場合は終了信号を受け取り、Read オペレーションの場合はデータと共にバスへ出力する。

Operation Control 部は各オペレーションを制御する。各オペレーションのステップを Read/Write Control 部に指示する。Operation Control 部の中は有限状態機械となっており、各状態へは Interface 部からの動作開始信号と、Read/Write Control 部の Read/Write 終了信号によって遷移する。

Read/Write Control 部は Operation Control 部からの開始信号で動作を開始する有限状態機械であり、一定のタイミングで Flash Memory への制御信号を変化させる。動作が終了すると終了信号を Read/Write Control 部へ出力する。

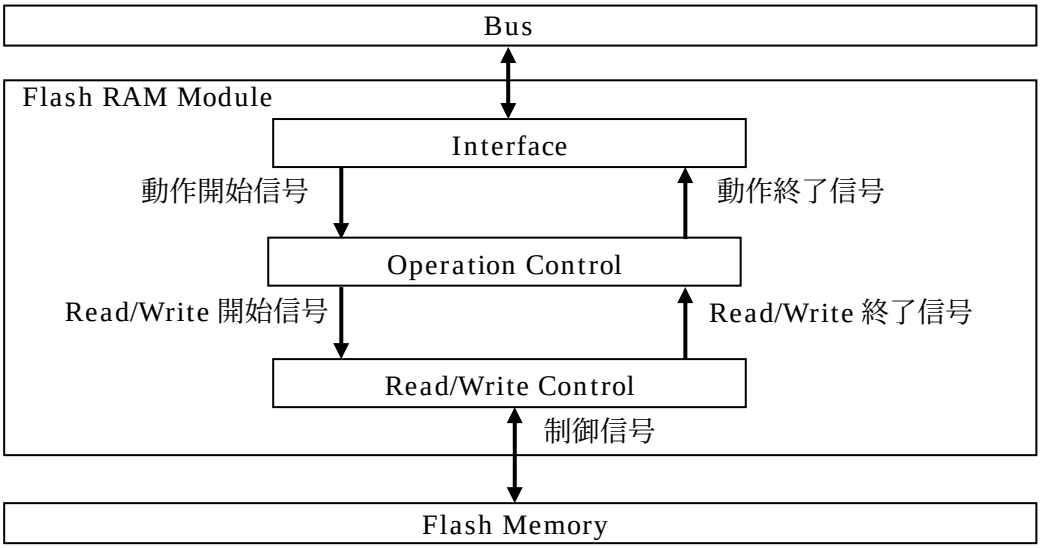


図 14：Flash RAM Module のブロック図

Flash RAM Module は Verilog-HDL で 210 行、回路規模は 1,118 ゲート相当であり、動作速度は 40MHz で動作する。

3.2 Sequencer

Sequencer は 8bit マルチサイクルプロセッサであり、内部の ROM に格納されたプログラムに従ってプロセッサ検証システムを制御する。

図 15 に Sequencer のデータパスを示す。

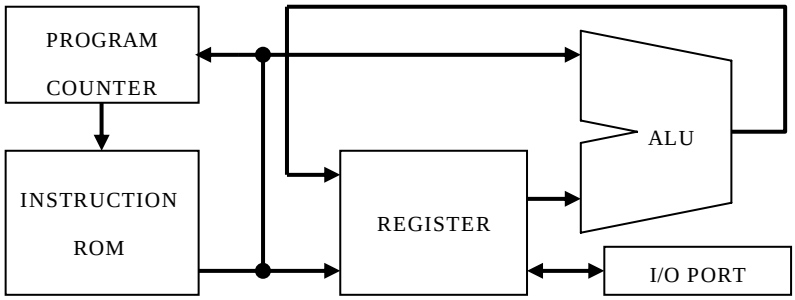


図 15：Sequencer のデータパス

Sequencer は以下のモジュールによって構成される。

- INSTRUCTION ROM：実行するプログラムを格納する。FPGA 内の RAM を用いて実装し、プログラムは FPGA のプログラム時に書き込む。
- PROGRAM COUNTER：ROM に現在のプログラムのアドレスを示す。分岐命令を用いて ROM 内のデータに書き換えることができる。
- REGISTER：8bit レジスタファイル。演算結果、変数などを一時的に保存する。
- ALU：算術演算、論理演算などを行う。
- I/O PORT：8bit 入出力ポート。Sequencer 外部とのデータのやりとりに用いる。

レジスタは内部に 4 つあり、変数の保持が可能である。同様に 4 つある I/O PORT は入出力を任意に設定でき、32bit の値の入出力やスイッチ・マトリクスなどの実現が容易である。

Sequencer は以下のような命令を持つ。

- 算術演算命令：ADD、SUB、ADC、SBC
- 論理演算命令：AND、OR、XOR、LDI
- 比較命令：CMP
- 入出力命令：RIO、WIO
- bit チェック命令：CHK
- 条件分岐命令：JPC
- 制御命令：NOP、HLT

Sequencer は通常のプロセッサとは異なり、メモリを読み書きする命令を持たない。データは全て定数として扱い、REGISTER に格納した変数との比較や演算に用いる。その代わり、割り込み信号の値を 1bit ずつ調べて分岐する bit チェック命令を持っている。これは外部からの制御信号に応答するためのもので、bit 数の少ない制御信号を扱い、素早い応答が求められる Sequencer において重要な命令である。Sequencer はこれらの命令を 1 から 4 サイクルかけて実行する。

モジュール全体の Verilog-HDL による記述量は約 500 行で、回路規模はメモリを含めて 18,552 ゲート相当、最高動作周波数は 53Mhz である。

3.3 Instruction Clock Counter

Instruction Clock Counter はプロセッサ検証システムにおいて、プロセッサの性能や特徴を調べる重要なモジュールである。Instruction Clock Counter は単純なカウンタを命令の種類の数だけ集めたものである。プロセッサの命令フェッチを監視し、どのような命令が何回実行されたか計測する。また、パイプライン方式の MPU では、ストールの回数を計測するレジスタを追加した。

このモジュールにより、プログラムが実行された場合にどのような命令が何度実行されたか観測が可能となり、プログラムの特性などが観測できる。また、プロセッサ内部の様々な状態の変化も調べることができる。

3.4 System Status Decoder

System Status Decoder は Sequencer から与えられた値に従って、プロセッサ検証システムの各モジュールに制御信号を与える。研究の当初は Sequencer が直接各モジュールに制御信号を与えていたが、Instruction Clock Counter などの機能を追加し、システムが遷移する状態が増えるにつれて制御信号が増加したため、このモジュールを用いて制御信号を管理するようにした。この System Status Decoder を用いることで、複数の制御信号を一度に変更でき、さらに仕様の変更においても Sequencer のプログラムの変更を最小限に抑えることができる。

4. FPGA ボードを用いた検証

2 章で述べた MPU と 3 章で述べた周辺モジュールを用いて、FPGA ボード上で検証を行った。この章では検証の手順と結果について述べる。

4.1 ハードウェアとソフトウェアの検証手法

一般的にハードウェアの検証はソフトウェアの検証よりも難しいとされている。ソフトウェアは逐次的に実行されるのに対し、ハードウェアはシステム全体が並列に実行される。そのためエラー箇所の特定が難しく、システム全体の信号を監視しなくてはならないので、検証速度も低下する。またハードウェアは設計段階での検証に加えて製造段階でのテストも存在し、テスト時間の増大も問題になっている。

ソフトウェアの検証ではデバッカの使用が一般的だが、ハードウェアの検証においてはソフトウェアによるシミュレーションが一般的である。しかし、シミュレーションはその精度と回路規模に指数的に時間がかかり、実際に LSI として実装した場合と同等の精度のシミュレーションを行うと、その時間は等価回路を用いた場合に比べて 1000 倍から 10 万倍以上の時間がかかる。検証の低速化は設計期間の増大と設計の危機を招き、複雑な回路は製品の工場におけるテストも困難になる。そのため、最近では様々なテスト容易化手法が提案されている。

DFT(Design For Test)は工場におけるテストが容易になるように設計段階でテスト用の回路を対象の回路に組み込む手法である。これによりテストが高速に実行でき、等価回路を作成した場合のテストも容易になる。しかし、回路の性能が低下したり、適用できる回路が限定されるという難点もある。

設計段階における検証の高速化では、FPGA による等価回路の実装が一般的であるが、出力できる信号の数が FPGA のパッケージによって規定されるため、内部信号の観測が難しい。そのため現在では回路内に観測やデバッグ用の回路を自動的に実装したり、回路内の特定の信号を一定時間毎に出力するモジュールなどが検証ツールとして一般的に広まりつつある。

4.2 検証に用いた MPU とシステム構成

検証に用いたプロセッサはシングルサイクル方式とパイプライン方式のプロセッサである。Memory には命令メモリとデータメモリを各 2k Byte ずつ実装した。RC100 ボード上には 7 セグメント LED が 2 つと、追加されたプッシュスイッチとトグルスイッチが実装されている。構成を図 16 に示す。7 セグメント LED は System Status Decoder からの情報を表示するために使用し、トグルスイッチは各モジュールのリセット信号に、プッシュスイッチは Sequencer の割り込み信号に用いた。7 セグメント LED はシステムが 2.2 章に示した各ステップのいずれにあるかを表示し、ステップによってはアドレスやデータを表示する。トグルスイッチは FPGA のプログラム直後に各モジュールをリセットするために用いる。プッシュスイッチは Sequencer の割り込みに用

い、システムの動作を中断させて状態を確認するために使用した。

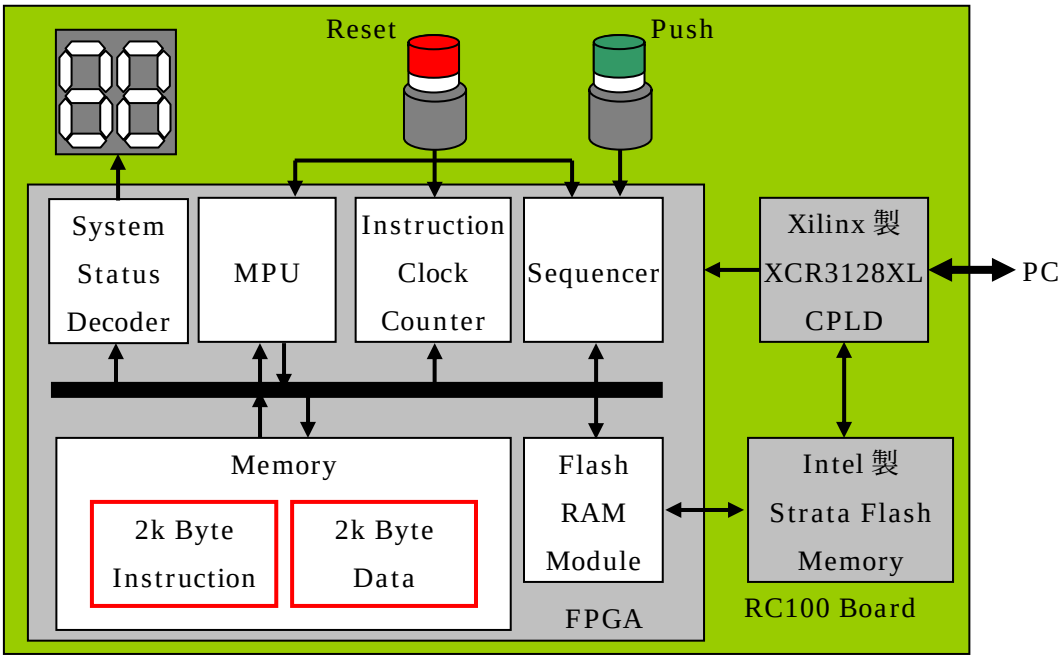


図 16：プロセッサ検証システムの実装構成

4.3 検証結果の評価

4.2 章で示したプロセッサ検証システムで「1 から 10 までの和」、「最大値探索(要素数 50)」、「掛け算」、「割り算」、「バブルソート」の各プログラムを実行し、実行時の命令の統計を取った。シングルサイクル方式で実行した結果を表 1 に、パイプライン方式で実行した結果を表 2 に示す。パイプライン方式ではストールの回数も計測した。また、各プログラム毎にシングルサイクル方式とパイプライン方式での実行クロック数の変化を表 3 に示す。

各表の制御命令は NOP、HLT など MPU を制御する命令である。レジスタ演算命令は算術演算命令と論理演算命令、シフト命令と比較命令を含む。いずれもレジスタ間で演算を行う命令である。即値命令は即値加減算と即値ロードを含む。ジャンプ命令には相対アドレッシング無条件分岐とレジスタアドレス無条件分岐、さらに内部割り込みを含む。

表 1：シングルサイクル方式 MPU による命令の実行頻度

命令	プログラム									
	1 から 10 までの和		最大値選択 (要素数 50)		掛け算		割り算		バブルソート (要素数 40)	
制御命令	1	2%	1	1%	1	1%	1	1%	2	0%
レジスタ演算命令	23	36%	104	26%	31	49%	34	52%	2882	33%
即値命令	15	24%	100	25%	5	10%	4	6%	1707	19%
条件分岐	12	19%	99	24%	15	24%	15	23%	1639	19%
メモリ命令	1	2%	49	12%	3	5%	4	6%	1772	20%
スタック命令	0	0%	0	0%	0	0%	0	0%	0	0%
ジャンプレジスタ命令	0	0%	0	0%	0	0%	0	0%	0	0%
相対アドレスジャンプ命令	11	17%	49	12%	7	11%	8	12%	819	9%
合計	63	100%	402	100%	63	100%	66	100%	8821	100%

4.3.1 シングルサイクル方式の評価

表 1 の結果から、各プログラムにごとに使用する命令の差異が確認できた。各プログラムがこの MPU によって実行されるとき、どのような命令の割合で実行されるかがわかる。「1 から 10 までの和」、「掛け算」、「割り算」はメモリアクセスを引数のロードと結果のストアのみに用いるため、メモリ命令が少なく、レジスタ演算命令が最も多い。即値命令の割合は少し異なっているが、条件分岐やジャンプ命令の割合は近い。「掛け算」と「割り算」のプログラムでは、その割合はほぼ同じである。メモリアクセスが多い「最大値選択」と「バブルソート」を比べると、メモリ命令が前者は 12%、後者は 20%と倍ほども違う。これは「最大値選択」がロードのみを用いるのに対し、「バブルソート」がロードとストアによって値の交換を行っているためだと考えられる。

特に目につくのは、条件分岐とジャンプ命令の割合である。「1 から 10 までの和」は条件分岐とジャンプ命令の割合がほぼ等しいのに対して、他のプログラムは条件分岐がジャンプ命令の倍ほどの割合である。「1 から 10 までの和」は C 言語の FOR 文に相当するプログラム構造の中で演算を行っているので、ループの抜けだしのチェックと、ループの先頭へのジャンプの割合が等しくなる。それに対して、他のプログラムが FOR 文のなかで IF 文を実行しているため、IF 文の実装のための条件分岐が FOR 文の回数と同じだけ増えたためだと考えられる。これらのことから、高位言語のプログラムと機械語命令の間には密接な関係があり、プロセッサの命令実行に特性がある場合、それを考慮することは機械語命令の高効率化に大きな効果があると考えられる。

さらに掛け算と割り算をプログラムで実行した場合、専用ハードウェアによる実行に対して非常に長い時間がかかることがわかった。

表 2：パイプライン方式 MPU による命令の実行頻度

命令	プログラム									
	1 から 10 までの和		最大値選択 (要素数 50)		掛け算		割り算		バブルソート (要素数 40)	
制御命令	3	5%	1	1%	2	3%	1	2%	2	0%
レジスタ演算命令	23	35%	106	26%	30	50%	32	50%	2882	33%
即値命令	15	23%	102	25%	6	10%	4	6%	1707	19%
条件分岐	12	18%	101	24%	13	22%	15	23%	1639	19%
メモリ命令	1	2%	51	12%	3	5%	4	6%	1772	20%
スタック命令	0	0%	0	0%	0	0%	0	0%	0	0%
ジャンプレジスタ命令	0	0%	0	0%	0	0%	0	0%	0	0%
相対アドレスジャンプ命令	11	17%	50	12%	6	10%	8	13%	819	9%
合計	65	100%	411	100%	60	100%	64	100%	8821	100%
ストール回数	0		0		0		1		0	

表 3：実装方式によるクロック数の違い

実装方式	プログラム毎のクロック数				
	1 から 10 までの和	最大値選択 (要素数 50)	掛け算	割り算	バブルソート(要素数 40)
シングルサイクル	251	1606	251	263	35283
パイプライン	65	411	60	64	8822
パイプライン/シングルサイクル	25.90%	25.59%	23.90%	24.33%	25.00%

4.3.2 パイプライン方式の評価

表 2 の結果を見ると、パイプライン方式での実行とシングルサイクル方式の実行では実行する命令の頻度があまり異ならないことがわかる。これはパイプライン方式でもその実行する命令の結果がシングルサイクルと同じであることが保証されているため、ほぼ同じ数になったと考えられる。また、4 段パイプラインの 2 段目で分岐を判定し、同期式のメモリを用いているために分岐時に損失が発生せず、シングルサイクルと命令の実行頻度が異ならない。もしパイプラインの段数が増え、分岐時に損失が発生するようになると、分岐スロットなどの対策によってプログラムの内容が変わってしまうため、命令の実行頻度が異なる。

ストールが発生したのは割り算のみであった。これは、割り算のプログラムが最初にロードした除数が 0 であるかどうかチェックするためである。このプログラムや他のプログラム中でもこれ以

外にロードした内容を直接分岐に用いることはない。これは分岐の条件が 0 かそれ以外かしかないからである。分岐の条件判定はロードした内容を比較命令で処理した後に実行されるからである。ストールが発生する状況が小さいのことは性能の向上に繋がるが、これは分岐命令の利便性が低いためだとも言える。様々な条件で分岐ができる命令セットの場合、ロードしたメモリの内容を直接分岐命令で処理するためハザードの確率が増えるが、その代わりプログラムの静的な命令数が減る。どちらが性能を向上させるかは様々なプログラムを実行し、その結果から判断する必要がある。

4.3.3 シングルサイクル方式とパイプライン方式の対比

表 3 にシングルサイクルとパイプラインの性能の違いが表れている。シングルサイクル方式とパイプライン方式は共に 1 つの命令を 4 つのステップで実行する。しかし、シングルサイクルでは 1 ステップに 1 クロックかかるが、パイプラインは各ステップを並列に実行するので、プロセッサ全体で 4 つのステップを並列に実行でき、1 クロックで 1 つの命令を完了できる。そのため、シングルサイクル方式と比べ、パイプライン方式では同一のプログラムを実行するために必要なクロック数は 4 分の 1 になっている。パイプライン方式の最高動作周波数はシングルサイクル方式の約 1.5 倍であり、パイプライン方式はシングルサイクル方式と比べ 6 倍程度の性能が得られた。

さらに、表 1、表 2 のような統計を利用すると、同一のプログラムを異なるプロセッサで実行したとき、命令セットによる性能の差異やプログラムとプロセッサの相性を知ることができる。同じプログラムでも、命令セットの違いによって、命令は大きく異なる。あるプログラムがあったとき、そのプログラムを実行するにはどのプロセッサが最適なのか判断する際に判断材料となる。また、マルチサイクルプロセッサの場合、命令の出現頻度を用いて CPI や平均実行時間を算出可能である[1]。

今回の実装では命令の種類による統計とストール回数のみの計測であったが、FPGA は柔軟にハードウェアの構成を変化させることができるのが大きな特徴であり、命令の統計の他にも様々な条件でデータを取ることが可能である。例えば、条件分岐の成功・不成功の割合や分岐予測の精度の計測、ある 1 命令やループの実行回数の計測などがある。また、パイプライン・プロセッサの場合はストールやハザードの回数、その発生部分を計測することにより、コンパイラの最適化性能を計ることが可能である。これら制御信号やプロセッサ内部の信号を実時間で監視し、計測することは難しく、このような検証を高速にできるのは FPGA による検証システムの大きな利点である。

また、今回使用した RC100 ボードには PC と通信が可能な機能が無かったため、検証システムが PC にデータを送ることが不可能だった。今後は通信機能を持ったボードを用いることで、ブレークポイントやレジスタ値のトレースを可能にし、プロセッサの内容を PC から直接観測できるようなシステムの構築を考えている。

5. 教育用プロセッサ KUE-CHIPII 検証システムとの比較

本年、同研究室の船附氏によって教育用プロセッサ KUE-CHIPII(Kyoto University Education Chip2)を搭載したボードコンピュータが FPGA によって実装された[11]。その研究に参加し、KUE-CHIPII を FPGA によって実装するための構成案を作成した。そこで、この KUE-CHIPII 検証システムと、プロセッサ検証システムを比較し、その差異を考察する。

5.1 教育用プロセッサ KUE-CHIPII

KUE-CHIPII は大学などでの計算機教育のための教材として開発された 8 ビット・マルチサイクルプロセッサである。KUE-CHIPII のアーキテクチャは非常に単純なものとなっており、計算機のアーキテクチャを理解するために必要な最低限の命令セットを持つ。しかも、内部のレジスタ、アキュムレータ、プログラムカウンタなどを外部から自由に観測・制御でき、プロセッサのアーキテクチャを理解するのに最適な構造になっている。

5.2 FPGA を用いた KUE-CHIPII 検証システム

KUE-CHIPII を FPGA を用いて実装するにあたり、FPGA ボードとして株式会社ヒューマンデータ社製「SpartanII 学習用ボード」を用いた。しかし、KUE-CHIPII は内部の信号の観測や制御のために多数のスイッチやディスプレイを必要としており、XSP-006 ボードだけでは不十分であった。そのため、スイッチやディスプレイは KUE-CHIPII ボードのものを利用し、KUE-CHIPII 自身とメモリを FPGA によって実装することとした。

図 17 にその構成を示す。

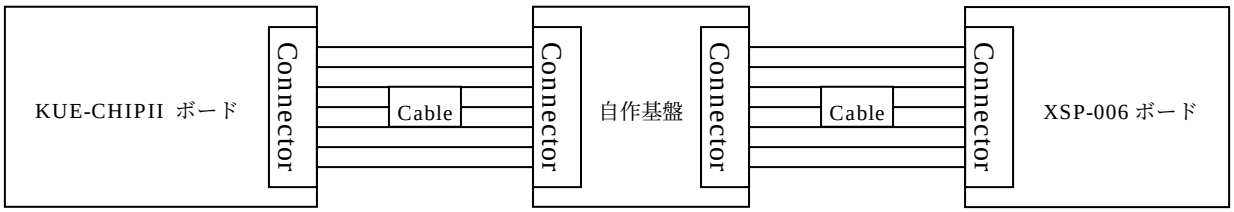


図 17：KUE-CHIPII 検証システムの構成

XSP-006 ボードと KUE-CHIPII ボードを自作基盤を用いて接続し、両基盤で異なるケーブルの電源とグラウンドの位置を自作基盤上で変更する。また自作基盤上では保護の必要がある信号線については抵抗を挿入する。

この構成によって実装された KUE-CHIPII 検証システムは、KUE-CHIPII 同士の通信以外の全

てのプログラムを実行することが可能である。KUE-CHIPII 検証システムは検証のために以下のような機能を有する。

- 内部レジスタの観測
- メモリ内容の観測、書き換え
- プログラム 1 クロック、1 命令実行
- 命令フェーズの表示
- 任意のクロック周波数によるプログラムの実行
- プログラム内容のプリンタ出力

これらの機能により、メモリにプログラムをスイッチで入力し、それらの 1 クロック・1 命令実行を行うことで、プログラム実行時のプロセッサ内部の変化を観測することが可能である。これにより、プロセッサの動作とアーキテクチャの理解を容易にするのがこの KUE-CHIPII 検証システムの目的である。

今回作成したプロセッサ検証システムと比較して、この KUE-CHIPII 検証システムはプロセッサ内部の観測において優れている。これは多数のディスプレイとスイッチを持っているため、多数の入出力装置を用いることで、KUE-CHIPII は観測を容易にしている。

検証システムとして用いる場合、入出力装置は多いほどよい。RC100 ボードは入出力装置が少ないため、検証システムの実装が困難であった。システムの設計において、その実装対象の選定が非常に重要であることがわかる。

6 おわりに

本論文ではプロセッサ検証システムを HDL によって設計し、それを実際に FPGA に実装を行い、その機能を確認した。MPU の命令セットは独自に設計したものを使用し、シングルサイクルとパイプラインという異なった方式で FPGA を用いて実装し、その性能の違いを検証した。その結果、パイプライン方式のプロセッサで非常に高い性能が得られ、その効果を実感することができた。

またシステム全体も HDL で記述した自作モジュールを構築することで、プロセッサだけではなくプロセッサを動作させるシステム全体の動作を理解すると共に、HDL や論地合成ツールの使用法、様々なレベルでのシミュレーション方法など大学院での研究へ繋がる基礎的な技術を習得できた。

システム上で 5 つの異なるプログラムを実行することで、高位言語プログラムの構造と機械語命令の関係とプロセッサ検証システムの機能についても観察した。その結果、高位言語プログラムの構造と機械語命令の構造に密接な関係があることがわかり、さらに詳しい検証を行うために必要な機能についての情報が得られた。

今後の課題としては、異なるボードを用いてのより実用的なプロセッサ検証システムの構築と、プロセッサを動作させる環境が異なる場合の性能比較が考えられる。プロセッサの性能の向上としては、スーパースカラ方式での実装や、浮動小数点演算機能の追加なども考えられる。

大学院ではマルチプロセッサシステムや、それに伴うプロセッサを結合するバスの最適化問題、リコンフィギャラブル・デバイスを用いた高性能計算機の設計などを学びたいと思う。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授、小柳滋教授に深く感謝いたします。

また、本研究に関して貴重なご意見をいただきました、古川達久氏、及び色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心より深く感謝いたします

参考文献

[1] John L.Hennessy, David A.Patterson:コンピュータの構成と設計(上)(下),日経 BP 社,1999.

[2] Andrew S. Tanenbaum:構造化コンピュータ構成, ピアソン・エデュケーション,2000.

[3] Jorgen Staunstrup,Wayne Wolf:Hardware/Software Co-Design,Principles and Practice, Kluwer Academic Pub.,1997.

[4] FrancoisXavier Standaert, Gael Rouvroy, JeanJacques Quisquater, JeanDidier Legat: A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL, FPGA'03, February 23–25, pp 216-224, 2003.

[5] Pablo Moisset, Pedro Diniz and Joonseok Park: Matching and Searching Analysis for Parallel Hardware Implementation on FPGAs, FPGA 2001, February 11-13, pp 125-133, 2001.

[6] Dirk Koch, Jurgen Teich: Platform Independent Methodology for Partial Reconfiguration, CF'04, April 14・6, pp 398-403, 2004.

[7] 小林優:入門 VerilogHDL 設計入門,CQ 出版社,2001.

[8] 鈴木昌治: ASIC 開発における数値演算回路設計の定石, Design Wave Magazine 2003 年 7 月号 pp 20-47, 2003.

[9] 浅井剛: ソフト・マクロの CPU を使おう! , Design Wave Magazine 2003 年 8 月号, 2003

[10] 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.

[11] 船附誠弘: 教育用マイクロ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情報学科卒業論文, 2005.