CPUの高位設計の自動検証システムの作成と学生実験への適用

北浜優子 北嶋暁 岡野浩三
東野輝夫 谷口健一

大阪大学大学院基礎工学研究科情報数理系専攻

概要

大阪大学情報工学科の3年次学生のCPU設計実験において一部，高位設計用の自動検証システムを導入した．自動検証システムは(1)命令セットアーキテクチャと，制御部を一状態機械で表したRTレベルの仕様に対して，後者が前者の正しい詳細化であることを証明するツールと，(2)複数状態機械で表した制御部の生成する入出力信号が(1)の制御部のものと等価であることを証明するツールの2つからなる．自動検証システムを用いたグループと用いなかったグループの設計作業時間の比較等について報告する．

Abstract

In this report, we present an experiment result in laboratory work of designing 16-bit CPUs for undergraduate students. A half of students use the following two verifiers to design 16-bit CPUs: (1) a verifier which checks the correctness of refinement between an instruction set architecture level's specification and its RT level's one, and (2) a verifier which checks the equivalence of I/O values between the controller of the RT level specification consisting of a single state machine and its refined controller consisting of multiple state machines. A comparison between groups using verifiers and groups not using verifiers is given.

1. はじめに

我々の研究グループでは，今までに，主にCPU設計例に対し，形式的手法を用いた詳細化の正しさを保証する段階的設計方法を考案したり[1]，自動証明を行う方法を考案し[2]，[3]，実際に自動検証システムを作成してきた．これらの研究では，実際に小規模の例題を通してこれらの手法やシステムの有用性等を確認してきたが，実際の設計過程での適用実験は行われていなかった．一方，最近，形式的手法による設計方法を学生に教育する必要性が認識されてきたが[4]，形式的手法による設計方法を大学学部レベルの演習あるいは実験という形で適用した例は著者らの知る限りでは存在しない．

段階的詳細化と形式的手法を組み合わせた設計方法をこのような学部実験に適用することは，形式的手法の有用性の評価や，学生教育の観点などからは非常に重要である．

そこで，大阪大学基礎工学部情報工学科の平成9年度の学部3年次の学生実験において，我々の自動検証システムを学生実験向きに作り直した上で，はじめて自動検証システムを実際に導入し，学生の論理設計にかかる作業時間等を調べることにした．

本学生実験では，従来，8bitのCPUを要求仕様から設計し，設計結果をROM, RAM, 74シリーズのTTL等で実装し動作確認するという工程を4, 5名のグループ実験として行ってきた．平成7年度より，設計にはCADシステム[1]を用い，動作確認には，FPGAボードを用いるようになった．CADシステムを用いることにより下位設計は大幅に省力化がなされたが，高位設計は相変わらず人力によるものであり，正しい詳細化に多くの時間がかかっていた．

導入した自動検証システムは次の2つのツールからなる．

RTレベル証明ツール——【入力】1. CPUの各命令前後におけるレジスタの値の変化を基本関数で記述した仕様，2. レジスタ，ALUなどの機能部品と，制御部，データパスとの接続関係，3. 一状態機械で表した制御部．【出力】2と3が，あらかじめ定められた基本関数，機能部品の意味定義のもとで1の正しい詳細化になっているか否かの答え．

制御部論理設計レベル証明ツール——【入力】1. 一状態機械で表した制御部，2. 複数状態機械で表した制御部．【出力】2が1の正しい詳細化になっているか否かの答え．

この2つのツールを使用することにより，CPU設計の高位の設計誤りを下位工程に持ち込まないことが期待できる．

実際の学生実験では，検証システムを用いたグループと用いなかったグループの2つの群に分け，あらかじめ定められた要求仕様を満たすCPUを作成するのに要した作業時間を調べた．得られた作業時間や学生へのアンケートなどから，自動検証システムをうまく導入すれば，設計作業の効率化および学生の形式的手法に対する理解の両面において，このような取組みが有用であろうという見通しを得た．

本稿の以降の構成は次のとおりである．2章では，実施した学生実験の内容について説明し，3章では，自動検証システムについて簡単に述べる．4章で実施結果についてまとめ，考察を行う．5章でまとめと今後の課題について述べる．

Applying Formal Method to High-Level Design of CPU in Laboratory Work for Undergraduate Students
Yuko Kitahama, Akira Kitajima, Kozo Okano, Teruo Higashino, Kenichi Taniguchi
Department of Informatics and Mathematical Science, Osaka University

[1] Altera社 Max+PLUS II, 一部 Altera社の高度教育プログラムに負う．

2. 学生実験の内容

2.1 学生実験の概要

大阪大学基礎工学部情報工学科では3年次の学生(約80名)を対象に,15週(180分/週)の授業でCPUの設計実験を行っている.本学生実験ではあらかじめ仕様(CPUの各命令の意味定義)を定めておいた16bit CPUを設計し,FPGAボード[2]で実際に動作させる[5].要求仕様で与えた命令はロード,ストア,算術,分岐等40個である.汎用レジスタ数は2つ,アドレッシングモードは,ロード,ストアにのみ即値・直接・レジスタ間接を使っている.算術演算はすべてレジスタ間に関するもののみである.実装に際しては,他にどのようなレジスタを使うか,どのようなデータパスアーキテクチャを用いるか,どのような制御部を用いるか等については自由度がある.また,希望により命令セットの追加(push, pop, call, return等)を認めた.

2.2 学生実験におけるCPU設計工程の概要

各グループが学生実験で行う各工程は以下のとおりである.

(1) 要求仕様決定とRTレベルアーキテクチャの設計

要求仕様(各命令実行前後のレジスタ値の関係や各命令のbinary表現を含む)(図1 a)を決定する.ついでこの要求仕様に基づき,使用レジスタ,ALU等各機能部品(図1 d)の動作定義を決定する.制御信号線や各機能部品とデータパス(図1 c)の接続を決定する.

(2) 一状態機械で表した制御部の設計と正しさの確認

一状態機械で表した制御部(図1 b)を設計し,この制御部や機能部品,データパスが要求仕様を正しく満たしていることを確認する.

(3) 機能部品の設計と正しさの確認

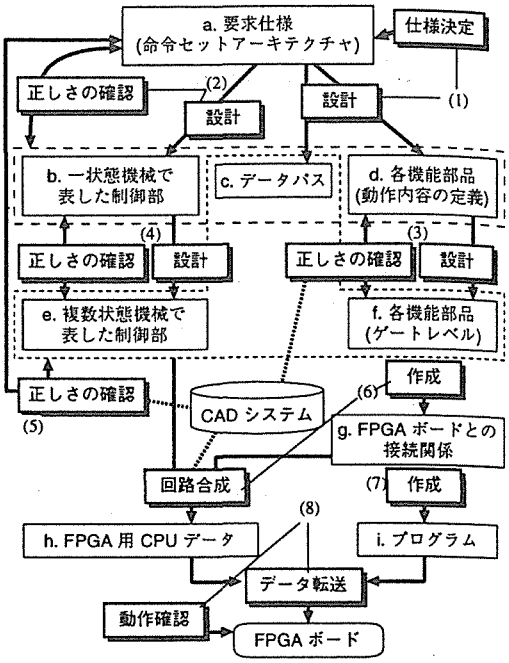
ハードウェア記述言語(今回Altra社のAHDLを用いている)で機能部品(図1 f)を実現し正しさを確認する.

(4) 複数状態機械で表した制御部の設計と正しさの確認

制御部から生成された制御信号にハザードが発生する問題を解決するために,状態遷移においてたかだか1つのFFしか値変化をしないような状態機械群から各制御信号を生成する設計がされる.

この工程では,そのような複数状態機械で表した制御部(図1 e)を設計し,これが一状態機械で表した制御部の正しい実現であることを確認する.ここでは,制御部のリセット信号も実現する.

[2]FPGAボードでは,DIPスイッチ,プッシュスイッチを入力として利用できる.また,3本(24 bit分)のバスを出力として利用できる.FPGAとしてAltera社FLEX8000シリーズの81188RC240を使用しており,約24000ゲート,約1000個のFFを持つ同期式回路を実現できる.



図中の(1)~(8)は本文中の2.2節における各作業工程を表す.

図1 CPU設計の概要

(5) 各部品の接続情報のCADへの入力と正しさの確認

各機能部品,制御部,データパスの接続情報をCADへ入力し,必要ならCAD上で合成した後,正しさの確認を行う.

(6) FPGAボード上で実装するための付加情報の入力

FPGAボード上の入出力端子,ROM; RAMへの接続情報(図1 g)をCADへ入力する.FPGA用CPUデータをCADで合成する.

(7) 課題プログラムの作成

設計したCPU上で動作させるプログラム(逆ポーランド式電卓)を作成する.

(8) FPGAボード上での正しさの確認

FPGAボードにCPU設計データ(図1 h)をダウンロードし,FPGAボードのROMに課題プログラムを載せ,動作確認を行う.

2.3 自動検証システムを用いた設計法

1章で述べたとおり,上記の工程(2)および(4)の正しさの確認においてはそれぞれRTレベル証明

```
...
define(SECONDWORD, read(mem, succ(pc)))

/* ロード命令 */
INSTRUCTION lda IS
        acc = "read(mem, SECONDWORD)";
        pc  = "succ(succ(pc))";
END INSTRUCTION;
...

/* 加算命令 */
INSTRUCTION addba IS
        acc = "add(acc, bcc)";
        c   = "carry(add(acc, bcc))";
        z   = "is_zero(add(acc, bcc))";
        m   = "is_minus(add(acc, bcc))";
        pc  = "succ(pc)";
END INSTRUCTION;
...

/* 条件分岐命令 */
INSTRUCTION jpc IS
        pc  = IF "c" THEN "SECONDWORD"
                     ELSE "succ(succ(pc))";
END INSTRUCTION;
...
```

図2　要求仕様の記述例 (一部)

ツール, 制御部論理設計レベル証明ツールを用いるという設計方法を導入する.

従来, 設計者が仕様を注意深く追っていくことでしか確認できなかった, 工程 (2) の正しさの確認や, CAD 上での波形シミュレーションで行っていた, 工程 (4) の正しさの確認は, この設計方法では, 自動で行え, また, 正しさを完全に保証できる. ただし, 厳密には工程 (4) では, 自動検証システムを用いて正しさの確認を行った後, 制御部のリセットを非同期式の外部入力信号線として後で追加し, その動作確認を CAD 上で行う作業も含まれる.

2.4 比較のためのグループ分け

比較のため今回の実験では, 全 28 グループ (1 グループあたり 3, 4 名. メンバーの構成はランダムに行われた) のうち, 14 グループに工程 (2) および (4) の正しさの確認において自動検証システムを使用させ, 証明に成功するまで原則として次工程に進ませないようにした.

3. 自動検証システム

自動検証システムは, 前章までで述べたように, RT レベル証明ツールと, 制御部論理設計レベル証明ツールとからなる. 各ツールとも, 入力となる記述を与えれば, 自動で証明を行う. 各記述に用いる言語は, ツール独自のものではあるが, 構文は本学生実験で用いる CAD システムの入力言語である AHDL に基づくものであり, 通常のハードウェア記述言語と本質的な違いがあるわけではない. 以下, それぞれについて概要を述べる.

3.1 RT レベル証明ツール

本ツールへの入力となる記述の例として, 要求仕様の記述例を図 2 に, データパス・機能部品の機能・制御部の記述の記述例を図 3 に示す.

本ツールでは, 証明を自動化するために, いくつかの制約を設けている. しかし, 実験で設計するような CPU のほとんどはこれらの制約を満たしている. 制約は, 具体的には次のとおりである. ハザードはないものと仮定している. また, リセット動作が正しく行われることの証明は対象としていない. また, CPU を停止する命令 (HALT) の正しさの証明も, 通常の命令とは異なるため, 対象としていない. また, 本ツールはパイプライン方式の命令実行制御は扱えない. ただし, 命令の完了前に次の命令コードの読み出しを始める (命令のプリフェッチ) 方式は扱える.

```
...
    WHEN s3 =>
        IF "insLDA or insLDB" THEN
        % nextstate %
            ctrl = s4B;
        % action %
            /RD = "L";
            Adrselmode = "AselPC";
            /LDMAR = "L";
            ALUmode = "alupassB";

        ELSIF "insSTA" THEN
            ...
```

(a) 制御部の記述の一部

```
...
/* ALU の機能記述 */
FUNCTION alu (in_a, in_b, mode) IS
  BEGIN
    out = IF "mode = alupassB" THEN
                "in_b"
        ELSIF "mode = aluadd" THEN
                "add(in_a, in_b)"
        ELSIF "mode = alusub" THEN
                "sub(in_a, in_b)"
            ...
        ELSE    "";
    aluc = IF "mode = aluadd" THEN
                "carry(add(in_a, in_b))"
         ELSIF "mode = alusub" THEN
                "carry(sub(in_a, in_b))"
         ELSE   "";
  END FUNCTION;
...

        /* 部品間の接続関係 */
        alu.in_a    = acc_unit.out;
        alu.in_b    = databus.out;
        alu.mode    = ctrl.alumode;
...
```

(b) 機能部品の機能およびデータパスの記述の一部

図3　制御部が一状態機械で表される RT レベルの CPU の記述例

本ツールで証明する正しさとは, 直観的には, 設計した回路でどの命令をどの順に実行しても, その実行結果が要求仕様のとおりであることである. 本ツールで実際に調べる内容は, その正しさの十分条件であり, 具体的には以下の各条件が要求仕様で定義された各命令すべてについて成り立つことである.

(1) 制御部の状態遷移図上で, その命令の実行を表す遷移系列 (以降パスと呼ぶ) が少なくとも 1 つ存在し, また, それらのパスを開始するための実行条件は同時にはたかだか 1 つだけが真となり, かつ, いずれのパスでも, そのパスを実行後次の命令を実行すること (次の命令をプリフェッチしてもよい).

(2) その命令を実行するパスすべてについて, そのパスを実行したとき, 各レジスタ値が要求どおりに変化すること.

```
  ...
/RD = !seqRD[0].q;
  ...

% seqRD initialvalue = H; %
seqRD[0].d =  (
    s6 & (insCPATB # insCPBTA
        # insLDAXB # insLDBXA)
  # s16
  # s19
  # s10
  # s13
  # s17 & (insEXCAB # insSETSPI # insCMPAI
          # insLDAI # insLDBI # insPOPA
          # insRETURN # njump # operator
          # SETorRESET # insCMPAB # insLOTRA
          # insLOTRB # insLOTLA # insLOTLB
          # insNOTA # insNOTB)
  # s23
)  ...
```

図 4　複数状態機械で表した制御部の記述例 (一部)

3.2　制御部論理設計レベル証明ツール

本ツールへの入力としては, 一状態機械で表した制御部の記述 (RT レベル証明ツールで使用したもの) と複数状態機械で表した制御部の記述を与える. 後者の記述例を図 4 に示す.

本ツールで証明する正しさとは, 一状態機械と複数状態機械とを, 初期状態から任意の同じ入力を与えて動作させた場合に, いずれの時点でも同じ出力であることである. 実際に本ツールで調べていることはその十分条件であり, 具体的には, 一状態機械で表した制御部において, 初期状態から実行可能なすべてのパスについて, 複数状態機械で表した制御部でも対応するパスがそれぞれ 1 つだけ存在することと, それらのパス上で対応する遷移それぞれについて, 出力が式として一致することである.

4.　実施結果と考察

4.1　測定項目と測定データ

今回は, 学生の作業時間を測定し, 検証システムを用いたグループと用いなかったグループとの違いを, 主にその作業時間に基づき評価した. 評価項目としては, 他にも, 完成した CPU にどれだけ誤りが含まれているかなどが考えられるが, 今回は対象としていない.

データの収集は, 学生からの報告の形で行った. 具体的には, あらかじめ作業内容を分類しておき, すべてのグループについて, 1 回の実験時間ごとに, 行った各作業項目とそれに要した時間を報告させた. また, 数値には表れない要因が分かるように, 感想も報告させた.

測定の結果得たデータは以下のとおりである.

- 学生からの報告に基づく各作業項目別作業時間 (表 1)

  各作業工程に着目し, その作業工程にどれだけの時間を要したかを各グループごとに表している. 各作業工程の内容は 2 章で述べたとおりである. ここでは, 作業をしている人数については考慮しない. その理由として以下のことがあげられる. (i) 正規の実験時間中は, PC が 1 グループ当り 1 台なので複数人で作業をする効果が少ないこと, (ii) 複数人での作業効率が上がるのは, AHDL 入力など一部の作業であり, これらは全体の作業量から考えると少ないこと. なお, 集計の仕方としては, 他にも, 各人の作業時間を別々に測定して加える方法や, 各作業工程ごとに, 協調作業による効果を反映させた重みを与えて, 本質的な作業量を計算するなどの方法も考えられるが, 報告に誤差があることや, 適切な重みが簡単には分からないことから, 今回は採用しなかった.

  各項目の数値では, グループの 1 人以上のメンバーが作業をしていれば, その作業工程の作業時間として加算する. また, 同時刻に 2 つ以上の作業工程を行っていれば, それぞれの作業工程の作業時間に (重複して) 加算する. したがって, この表の総計には時間的な重複が含まれている.

- 各グループごとの実実験時間 (表 2)

  実際に各グループが実験を終了するまでに要した時間を表している. つまり, グループの 1 人以上のメンバーが作業をしていた時間の総計である. したがって, この値には時間的な重複が含まれていない. なお, 時間外実験中に 1 人だけが作業をしていることもあるが, この時間も含めている.

各表は実験を行った全 28 グループのうち, 有効なデータが得られた 20 グループ (検証システムを用いた 11 グループ, 用いなかった 9 グループ) のデータについて示している. 残りの 8 グループについては, 各作業項目とそれに要した時間が工程別に区分して報告されていない, または, CPU 設計課題を最後までやり遂げることが時間的に不可能となったので途中で課題を中断し他の課題を与えた, という理由でデータから除外した.

4.2　考　　察

表 1 および表 2 から, 今回の実験においては, 検証システムを用いたグループの方が用いなかったグループよりも作業時間の総計はやや多かったといえる. ただし, 作業内容の様々な状況を考慮すると, 検証システムを用いたグループは, 作業効率自体は, 用いなかったグループと比べても悪くはなかったと推測できる. その根拠として, 検証システムを用いたグループは用いなかったグループに比べて作業進行上不利な点があったことがあげられる.

以下では, まず表中のいくつかの数値について, 数値には表れない状況を考慮した上での考察を行い, 次に, 検証システムを用いたグループにとって不利であったと思われる点を考察する.

今回の実験では, 検証システムを用いたグループは, 検証システムの使用法や入力の記述および出力

表 1　学生からの報告に基づく各作業項目別作業時間 (単位　分)

(a) 検証システムを用いなかったグループの作業時間

作業工程	グループ T-1	T-2	T-3	T-4	T-5	T-6	T-7	T-8	T-9	平均
(1) 要求仕様決定と RT レベルアーキテクチャの設計	180	240	180	180	180	210	270	360	300	233
(2) 一状態機械で表した制御部の設計と正しさの確認	290	330	480	620	180	330	330	180	30	308
(3) 機能部品の設計と正しさの確認	1530	960	1680	900	810	1800	1580	990	910	1240
(4) 複数状態機械で表した制御部の設計と正しさの確認	1170	1170	1650	1290	3250	2380	2330	1530	1450	1802
(5) 各機能部品の接続情報の CAD への入力と正しさの確認	870	1020	530	330	1110	330	1100	960	150	711
(6) FPGA ボード上で実装するための付加情報の入力	500	680	700	440	1200	260	1740	530	800	761
(7) 課題プログラムの作成	930	810	960	60	150	540	360	360	530	522
(8) FPGA ボード上での正しさの確認	750	820	1420	930	360	480	1200	300	850	790
総計	6220	6030	7600	4750	7240	6330	8910	5210	5020	6368

(b) 検証システムを用いたグループの作業時間

	V-1	V-2	V-3	V-4	V-5	V-6	V-7	V-8	V-9	V-10	V-11	平均
(1)	420	360	180	150	210	570	360	180	120	180	150	262
(2)	180	300	360	730	750	720	370	780	600	480	590	533
(3)	1260	270	1080	570	780	1020	900	1320	780	1050	630	878
(4)	720	1710	2940	1590	990	1980	1645	1270	2055	2040	2460	1764
(5)	150	1290	210	720	1050	150	1110	150	150	150	150	480
(6)	920	920	740	260	680	440	285	740	1450	530	1000	724
(7)	780	240	150	0	1080	870	180	1080	830	540	780	594
(8)	1110	960	3060	975	1230	960	1560	900	1000	570	2420	1340
総計	5540	6050	8720	4995	6770	6710	6410	6420	6985	5540	8180	6575

表 2　各グループごとの実実験時間 (単位　分)

グループ	T-1	T-2	T-3	T-4	T-5	T-6	T-7	T-8	T-9	平均
実実験時間	4790	3590	5180	3650	6480	4300	5640	3360	3670	4518
グループ	V-1	V-2	V-3	V-4	V-5	V-6	V-7	V-8	V-9	V-10	V-11	平均
実実験時間	3360	4890	7080	4570	4860	3990	5100	4090	5025	3810	5890	4788

メッセージの理解に時間がかかったものと考えられる. これは表 1 において, 工程 (2) の平均値が (a) では 308 であるのに対して, (b) では 533 であるというところに表れている. 工程 (4) については, (b) のグループには不利な点があるにもかかわらず (後述), 検証システムの効果が現れたため (b) の方がやや少ないという結果になっているものと考えられる.

本実験では, 工程 (2) の設計時にメモリアクセスのクロック数を考慮に入れなくてはいけない. しかし, 検証システムはこのことは判定の対象外であるので, 考慮に入れていなかった設計を行っていても, 正しいと判断する. 実際このような設計をして工程 (8) まで進んだあと, あと戻りしてしまったグループ (例えば V-1) が複数存在していた. このようなあと戻りに要した作業時間は工程 (8) の作業時間に含まれている. これは, 表 1 の工程 (8) の平均値が (a) では 790 であるのに対して, (b) では 1340 である理由の 1 つになっていると考えられる.

今回の実験では設計とは直接関係のない作業に時間をかけているグループがあった. グループ V-3 では工程 (6) の FPGA ボード上の入出力端子の接続情報の入力ミスが工程 (8) の時間数増加につながっている (アンケートによって確認した). 一方, グループ V-11 は工程 (8) において課題プログラムのデバッグにほとんどの時間 (他グループよりも多い) をかけている. これらのことが工程 (8) のデータに影響しているものと考えられる.

また, 工程 (3) や工程 (5) での正しさの確認は CAD 上のシミュレーションを用いて行われるが, それが不十分で, 下位工程である工程 (8) にバグが持ち込まれ, あと戻りをしなければならなかったグループもあった. これらのことも工程 (8) のデータに影響しているものと考えられる.

以下に検証システムを用いたグループにとって不利であったと思われる点を列挙する.

- 検証システムには使いやすさに関する問題があった.

  今回学生が使用した検証システムは, エラーメッセージが不親切なためエラーの個所を発見しにくい場合があったり, 設計した制御部によっては, 検証システムを実行してから検証結果が出力されるまでに 30 分程度という比較的長い時間を要することもあるという問題点があった. また, 2 つの検証ツールの使い方について学習

する時間が余分に必要であった (工程 (2) と工程 (4) で使う検証ツールはその入力の記述や, 出力メッセージの形式が異なる).

なお, 今回の実験では, 検証システムの使い方については, マニュアルを配布したのみで, 形式的手法や検証システムの使い方に関する講義は行わなかった.

- 検証システムを用いているにもかかわらず, 制御部のリセットの正しさの確認は CAD 上でのシミュレーションで行わなければならなかった.

  ただし, このような不利な状況にもかかわらず, 結果的に, 検証システムを用いるグループは全体としてこの点で費やした時間はそれほど多くなかった.

命令セット追加を自由に行わせたが, これに関する差異はこの実験ではみられなかった.

作業時間に関しては, 設計手法の違いによる差以上に, 各個人の能力の差や各グループの協調性の差が影響していることが考えられる. したがって, これらの表の数値から設計手法の違いによる作業時間の差異を単純に比較することはできない.

5. おわりに

本稿では, 学生実験において, 形式的手法を用いることにより, 主として設計の過程でどのような効果が現れるかを調べた結果について報告した. この取り組みはまだ試行段階であるため, 明らかな成果が得られたわけではないが, 実験の実施方法や検証システムの改善により, 設計作業の効率化および学生の形式的手法に対する理解の両面において, このような学生実験が有効になるであろうという見通しが得られた.

より有効に 2 つの手法を比較するために今後改善すべき課題を以下に列挙する.

- 設計した CPU の信頼性についても調べる.

  今回の実験では, 設計した CPU が正しく動作することの確認は, 与えられた課題プログラムが正しく動作するかどうかにより行った. この課題プログラムは CPU の持つすべての命令を用いなくても作成できるため, この課題プログラムが正しく動作したとしても, すべての命令について CPU が正しく動作することの確認にはならない.

  設計した回路が正しいかどうかについては, 検証システムを用いたグループが, 用いなかったグループに比べて, 劣ることはないであろうと思われる. しかし, 今回の学生実験で行ったような比較的小規模な設計において, 形式的手法を用いた場合とそうでない場合とで, 設計した回路が正しく動作するかどうかについて, 明確な差がでるかどうかが明らかではない.

  そこで, 学生実験で作成した CPU にどの程度の誤りが残っているのかを調べる必要がある. その方法としては, 例えば, 設計が終了した後に, 誤りがないことを調べるためのテストを詳しく行うという方法や, 検証システムを用いなかったグループについて, 検証システムを適用してみるなどの方法が考えられる.

- 設計とは直接関係のない作業を分離して測定する.

  今回の実験において, 学生が, 本来の設計作業とは直接関係のない作業に多くの時間を費やしていたことがわかった. その作業の具体的な例として, 検証システムや CAD システムなどの使い方の習得や FPGA ボードの不具合への対処などがあげられる. これらについては, 学生実験という性質上ある程度はさけられないものであるが, 測定の際分離すべきであった. 工程 (7) や工程 (8) は測定には関係ない項目であるが, 実際には工程 (3) や工程 (5) で発見すべきバグが工程 (8) まで持ち込まれてしまう例があったり, 工程 (7) のデバッグが工程 (8) で行われているなどの理由から, 工程 (7), 工程 (8) をデータから分離することが不可能であった.

- 自動検証システムを改良する.

  今回の実験において使用した自動検証システムは, 効率を重視していない実装であったため, 証明にかなりの時間を要することもあったが, 効率を考えた実装を行えば, 数分程度で証明ができると思われる.

参考文献

[1] 北道淳司, 東野輝夫, 谷口健一, 杉山裕二. 代数的手法を用いた同期式順序回路の段階的設計法. 電子情報通信学会論文誌 (A), Vol. J77-A, No. 3, pp. 420–429, March 1994.

[2] 北嶋暁, 森岡澄夫, 島谷肇, 東野輝夫, 谷口健一. 代数的手法を用いた CPU KUE-CHIP2 の段階的設計の正しさの自動証明. 電子情報通信学会論文誌 D-I, Vol. J79-D-I, No. 12, pp. 1017–1029, 1996.

[3] 森岡澄夫, 北嶋暁, 島谷肇, 東野輝夫, 谷口健一. 一つの EFSM の複数 EFSM による実現の正しさの一証明法. 第 52 回情処全大, 1K-1, March 1996.

[4] Edmund M. Clarke and Jeanette M. Wing. Formal methods: State of the art and future directions. Technical Report CMU-CS-96-178, Carnegie Mellon University, August 1996.

[5] 97 年度実験配布資料. 情報工学実験 C 指導書 97 年度版, 完全同期式 CPU 設計法 97 年度版, 形式検証ツールの使い方, 実験 C で用いる諸ツールの使い方に関するヒント 97 年度版, (株) オブジェクト, プログラムユニバーサルボード ユーザーズマニュアル (平成 8 年), ALTERA, Max+Plus II AHDL 1995.