設 計 自 動 化 79－1
（1996. 2. 9）

# 代数的手法を用いたCPU KUE-CHIP2の段階的設計およびその正しさの証明

北嶋 暁　森岡澄夫　島谷 肇　東野輝夫　谷口健一

大阪大学基礎工学部情報工学科

従来から我々が提案している代数的手法を用いた順序回路の設計法，検証法や支援システムの有用性を調べるため，教育用CPU KUE-CHIP2を要求仕様から段階的に設計し，その正しさの証明を行った．我々の証明支援系は作業の自動化をめざしており，本例題ではほとんどの部分が自動で行えた．設計では最適化（命令プリフェッチ）も行い，SFL記述へ自動変換し，パルテノンで回路（ネットリスト）を得た．設計と証明の作業時間も2週間程度であり，設計法や証明支援システムが実用上有用であると思われる．

# A Step-by-Step Design of a CPU KUE-CHIP2 and Its Correctness Proof based on Algebraic Methods

Akira KITAJIMA, Sumio MORIOKA, Hajime SHIMATANI,
Teruo HIGASHINO and Kenichi TANIGUCHI

Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University

In this paper, we present an example of step-by-step design of a CPU KUE-CHIP2, which was originally designed in Kyoto University for educational purposes, and its correctness proof. We refined the specifications of the CPU step by step from its requirement description, which is written in our algebraic language, to the RT level description. At the refinement process, we optimized the CPU so that the circuit can perform the instruction pre-fetch. Then we obtained the net-list of the circuit using the hardware synthesizer Parthenon. Our verifier proved almost automatically the correctness of each refinement, using algebraic methods. The work load for the design and the proof was about two weeks.

## 1 はじめに

本稿では，従来から我々が提案している代数的手法を用いた順序回路の設計法，検証法や支援システムの有用性を調べるため，情報工学教育用に京都大学で開発された CPU KUE-CHIP2[1] を要求仕様から段階的に設計し，その正しさの証明を行った．

設計した回路が要求仕様を満たしていることを完全に保証したいが，その証明は一般には困難である．そこで我々は，証明を実際上可能とするため，要求仕様から具体レベル（論理設計レベルなど）まで段階的に詳細化し，各段階ごと，詳細化が正しいことを証明するという方法をとっている[2]．詳細化では，無駄な遷移が入る場合があるので，状態図変形機能[3]を用いて，それを取り除くなどの最適化を行う．詳細化の正しさの証明は，我々の証明支援系[4, 5]を用いて行う．その証明支援系は，場合分けや項書き換え，および整数上の論理式（プレスブルガー文）の恒真性判定などを一定の手順で自動実行する機能を持ち，証明の自動化をめざしている．

従来，提案してきた手法の有用性を確かめるため，いくつかの例題に対し設計・検証実験を行ってきたが，これらには以下の問題点があった．(i) 例題が単純な機能の回路であったり，あるいは，部分的に（一部の重要な性質について）しか証明を行っていなかった．(ii) 最適化（CPU における命令プリフェッチなど）を行った回路に対しては，それが要求仕様を満たすことの形式的な定義がなく，証明を統一した枠組の中で行うことができなかった．

そこで今回，最適化を行った回路に対してもその正しさを定義したうえで，比較的複雑な機能を持つ回路を設計し，その正しさを完全に証明することにした．そのような例題として KUE-CHIP2 を選んだ．その理由は，通常の CPU が持つ命令セットがほぼサポートされていること（ロード，ストア，算術演算，論理演算，分岐等，十数種類の命令を持ち，それぞれ数種類のアドレッシングモードがある）と，多くの設計例が公開されているので，設計した回路の評価（性能比較）が行いやすいことである．

設計は，次のように段階的に行った：(1) CPU のレジスタごとに，CPU の各命令実行によって，その内容がどのように変わるべきかを，我々の代数的言語 ASL[2] を用い，要求仕様として記述する．(2) どのような演算やデータ転送を，どのような順番で行って各命令を実行するかを決め，その正しさを証明する．(3) 一命令実行の途中で次の命令の実行を始めるような最適化（命令プリフェッチの導入）を行い，その正しさを証明する．(4) バスアーキテクチャ（どのような機能部品を用い，それらをどのように接続するか）を決め，各機能部品（ALU など）の実現が正しいことや[6]，各状態遷移で行う演算やデータ転送が，定めた部品間の結線上で正しく行えることを証明する．(5) 得られた仕様（ASL 記述）を SFL 記述へ自動変換し[7]，NTT で開発された論理自動合成システム パルテノンにより回路のネットリストを合成して，その速度やゲート数が設計者の意図どおりであることを確かめる（必要なら適当な段階から再設計する）．

本例題では，設計および証明の作業を，十数個の設計誤りの修正やそれに伴う再証明も含め，2 週間程度で行うことができた．得られた回路は，要求仕様を満たすことが完全に保証されており，かつ，ゲート数は 1800 程度，最大動作周波数は 9MHz 程度（各命令の実行サイクル数は 1〜3 クロック）と，他の KUE-CHIP2 の設計例と比べても遜色ない性能のものである．証明は我々の証明支援系を用い，ほぼ自動で行えた．これらの結果より，我々の設計法や証明支援システムが実用上有用であると思われる．

以下，2 章では，従来より提案してきた回路の記述・設計法について簡単に述べ，KUE-CHIP2 の段階的設計の概略と，その性能評価の結果について述べる．3 章では，今回拡張した回路の実現の正しさの定義，KUE-CHIP2 の実現の正しさの証明の概略，および証明実験の結果について述べる．

## 2 KUE-CHIP2 の段階的設計

ここでは，まず，従来我々が提案してきた同期式順序回路の段階的設計法[2]について簡単に説明した後，KUE-CHIP2 の段階的設計の実際について，順を追って述べる．

### 2.1 同期式順序回路の段階的設計法

同期式順序回路は，任意の設計レベルにおいて，拡張有限状態機械（EFSM）でモデル化できる．回路の仕様は，EFSM の各遷移の動作内容（その遷移の実行により，各レジスタの値がどのように変わるべきか）の指定と，遷移の実行指定（どの状態でどのような条件が成り立つとき，どの遷移を実行して，どの状態へ移るか）からなる．各レベルの仕様は，我々の代数的言語 ASL/ASM[2] を用いて記述する．

我々の設計法では，要求仕様に回路の機能を記述した後，具体レベル（例えば論理設計レベル）まで段階的に詳細化する（必要なら最適化も行う[3]）．詳細化は，上位の一つの遷移を下位でより具体的な遷移系列に置き換える（展開する）ことにより行う．

### 2.2 要求仕様の記述

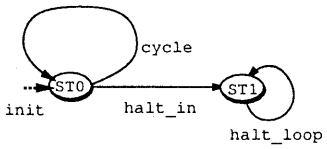
以下，KUE-CHIP2 の段階的設計の概要を順に述べる．

要求仕様（レベル 1 とする）には，各レジスタごと，CPU の各命令の（一回の）実行によって，内容がどのように変化すべきかを記述した．

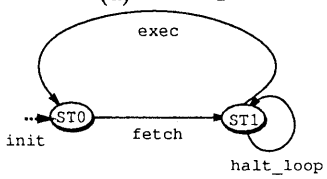
レベル 1 の状態遷移図を図 1(a) に示す．遷移 `cycle` が CPU 命令の 1 回の実行に対応する．ただし，Halt 命令が実行された場合は，遷移 `halt_in` を経て，ループ (`halt_loop`) を実行し続ける．

遷移 `cycle` の動作内容の記述を図 2 に示す（実行指定の記述については，省略する）．例えば，プログラムカウンタ PC については，RCF など一語命令を実行した場合には値が 1 増加し，分岐命令を実行した場合は分岐先のアドレスになり，ST など二語命令を実行した場合には 2 増加することが書かれている[1]．

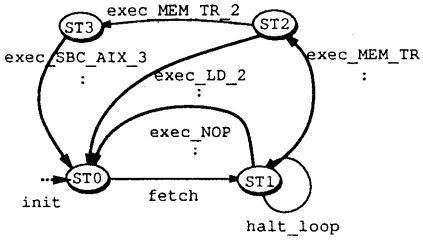
[1] 本来は，要求仕様レベルでは各命令の語数やフォーマットを決めておかず，詳細化の過程で決めていくのが望ましい．しかし本稿では，KUE-CHIP2 の各命令の語数などが文献 [1] で決められているため，要求仕様の記述の段階から命令の語数を決めておくことにした．



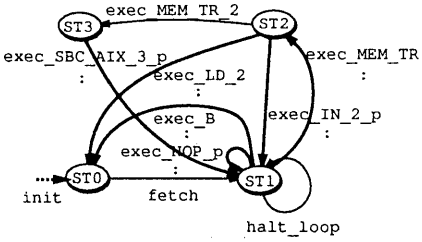

(a) レベル 1




(b) レベル 2




(c) レベル 3




(d) レベル 4

各図とも, 分岐時の遷移の実行条件は省略している. また, 太い矢印は, いくつかの遷移をまとめて表したものである.

図 1　詳細化の概要

### 2.3　各命令の実行手順の決定

レベル 1 の遷移 cycle の動作を, 命令フェッチ (遷移 fetch) と, それに引き続く各命令の実行 (遷移 exec) により展開 (詳細化) して, レベル 2 の仕様を得た (状態図を図 1 (b) に, 各遷移の動作内容の一部を図 3 に示す) [2]. レベル 1 とレベル 2 の遷移の対応関係を, 図 5 に示す.

次に, レベル 2 の遷移 exec をさらに展開して (フェッチした命令のアドレッシングモードにより分岐して, 別々の処理を行う), レベル 3 の仕様を得た (図 1 (c)). レジスタ間演算命令や NOP 命令などでは, ST1 から, その命令の演算・データ転送を実行して ST0 に戻る. 即値演算を行う命令では, ST1 から, 命令の 2 語目 (即値) をフェッチして ST2 に移り, 次に演算等を行って ST0 に戻る. 直接・間接アドレッシングモードの命令では, ST2 から, 指定されたアドレスの内容をフェッチした後, 演算等を行って ST0 に戻る.

[2] レベル 1 の遷移 halt_in は遷移 fetch で実行することにした.

```
(* 遷移 cycle の動作内容 *)
PC(cycle(s))
    == if (inst(mget(MEM(s),PC(s))) = RCF
             or ....) then
          inc_1(PC(s))
   else if ((inst(mget(MEM(s),PC(s))) = B
             or ....) then
          addr(mget(MEM(s),inc_1(PC(s))))
   else if (inst(mget(MEM(s),PC(s))) = ST
             or ....) then
          inc_1(inc_1(PC(s)))
   else
          PC(s);
          ...
CF(cycle(s))
    == if (inst(mget(MEM(s),PC(s))) = RCF) then
                FALSE
       else if .......
            .......
(* 遷移 halt_in の動作内容 *)
CF(halt_in(s)) == CF(s);
          .......
(* 遷移 halt_loop の動作内容 *)
PC(halt_loop(s)) == PC(s);
CF(halt_loop(s)) == CF(s);
          .......
```

図 2　要求仕様の記述の一部

```
(* フェッチサイクル fetch の動作内容 *)
IR(fetch(s)) == mget(MEM(s),PC(s));
PC(fetch(s)) == inc_1(PC(s));
                .......
(* 実行サイクル exec の動作内容 *)
PC(exec(s)) == if (inst(IR(s)) = RCF
                        or ....) then
                   PC(s)
               else if (inst(IR(s)) = B
                        or ....) then
                   addr(mget(MEM(s),PC(s)))
               else if (inst(IR(s)) = ST
                        or ....) then
                   inc_1(PC(s))
               else
                   PC(s);
               .......
CF(exec(s)) == if (inst(IR(s)) = RCF) then
                 FALSE
              .......
          .......
```

図 3　レベル 2 の記述の一部

### 2.4　命令プリフェッチの導入

命令によっては, その最後の実行サイクルで, メモリアクセスや, 命令レジスタ (IR) およびプログラムカウンタ (PC) の参照を行わないものがある. このような場合, 最後の実行サイクルで次の命令のフェッチを同時に行う (命令プリフェッチ) ことにすれば, 命令の実行を高速化できる.

そこで, レベル 3 の仕様を, 可能な場合には命令プリフェッチを行うように変更して, レベル 4 の仕様を得た (図 1 (d)). 例えば NOP 命令の最終サイクルの動作と次の命令のフェッチは, レベル 3 では exec_NOP と fetch という二回の遷移で行っていたが, レベル 4 では両方の動作を exec_NOP_p という一回の遷移で行うことにした.

この変更は, 我々の状態図変形機能[3]を用いて行った.

### 2.5　機能部品・データパスの決定

レベル 4 までは, 各遷移で行う演算やデータ転送を, どのようなバスアーキテクチャの上で行うか (ど

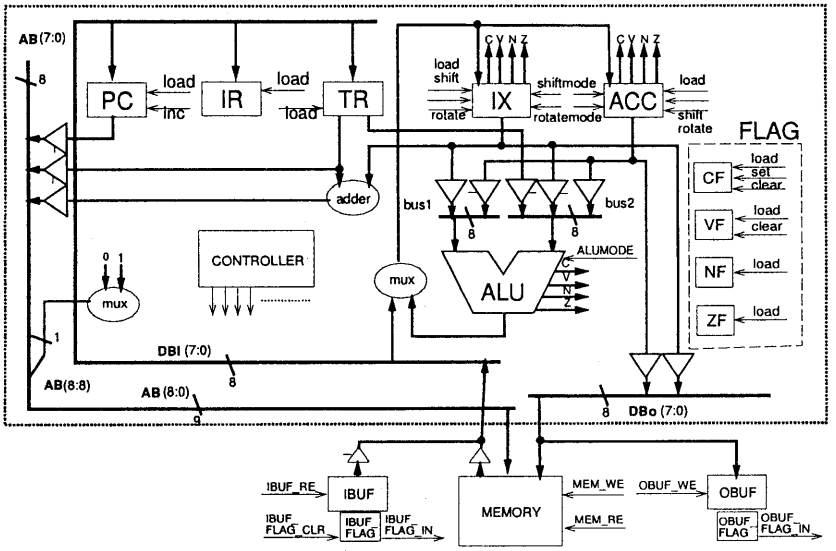



図 4　採用したバスアーキテクチャ

```
cycle(s) == exec(fetch(s));
halt_in(s) == fetch(s);
```

図 5　レベル 1 とレベル 2 との対応関係

表 1　設計した KUE-CHIP2 の性能

| | |
|---|---|
| 命令実行サイクル数 | 1〜3 |
| 最大動作周波数 (周期) | 9.09　MHz (110ns) |
| ゲート数 | 1767 |

のような機能部品を用い，それらをどのように接続するか）を決めずに設計を進めた．

バスアーキテクチャを決定してレベル 5 の仕様を得た．採用したアーキテクチャを図 4 に示す．ALU は一つで，8 ビットの算術演算（加減算）機能と，論理演算機能を持つ．各レジスタは，ロード，インクリメント，シフトなどの機能を持つ．各機能部品の接続は，(ゲート数の削減よりはむしろ）命令実行サイクル数を少なくすることを目標とし，レベル 4 で同時に行われる演算・データ転送が，そのまま並列に実行できるように定めた（レベル 4 の一動作が，そのままレベル 5 の一動作に対応する）．

図 4 のバスアーキテクチャのもとでは，各命令の実行サイクル数は，(アドレッシングモードにより) 1〜3 クロックとなる．

### 2.6　ハードウェア自動合成系による回路の性能評価

レベル 5 の記述を SFL 記述に自動変換し[7]，パルテノンにより回路のネットリストを得て，その性能評価を行った．ゲート数および回路の最大動作周波数を表 1 に示す．公開されている他の KUE-CHIP2 設計例と比べて，ゲート数は少し多くなったが，動作速度は速くなっており，意図した性能の回路を得ることができた．

## 3　KUE-CHIP2 の設計の正しさの証明

### 3.1　回路の実現の正しさの定義

従来，一本の遷移を展開する場合については，その（一回の）展開の正しさを定義して，そのもとで証明を行っていた[5]．しかし，今回の KUE-CHIP2 の設計のように，一本の遷移の展開だけではなく，最適化（命令プリフェッチの導入）も行った場合については，その正しさの定義がなかった．

そこで今回，上位の複数の（連続した）遷移を，下位で遷移系列で置き換える場合などについても，証明を同じ枠組の中で行えるようにするため，回路の実現の正しさ（レベル $k$ の回路が要求仕様を満たすこと）の定義を行った．以下の定義は，文献 [8] などに示されている，回路の実現の正しさの一般的な定義に準拠している．

以下，実現の正しさの定義について述べる．まず，要求仕様の記述および（任意の）レベル $k$ の記述に対し，以下を定義する．

要求仕様における（EFSM 上の）各遷移 $T_{(1,m)}$，各レジスタ $reg_{(1,r)}$ の組に対し，

- $T_{(1,m)}$に対応するレベル $k$ の状態遷移の時系列の始点となる遷移の（名前の）集合 $B_{(k,m)}$と，終点となる遷移の集合 $E_{(k,m)}$．
- $T_{(1,m)}$に対応するレベル $k$ の状態遷移の時系列の上で，$reg_{(1,r)}$の値が実現される（べき）状態の集合 $L_{(k,m,r)}$．その集合の各要素は，＜遷移名，前後を示すフラグ＞の組であり，その名前の遷移の実行直前（または直後）の状態で，$reg_{(1,r)}$の値が実現されるべきものとする．
- レベル $k$ のレジスタ $reg_{(k,1)}$,$reg_{(k,2)}$,$\cdots$の値から，$reg_{(1,r)}$ に対応する値を取り出す関数 $R_{(k,r)}$．

以上の定義のもとで，レベル $k$ が要求仕様を正しく実現していることを，次のように定義する．

要求仕様における，初期状態からの状態遷移の時

系列 $S_1$ と（任意のレジスタ初期値，任意の入力系列を仮定する），レベル $k$ における状態遷移の時系列 $S_k$ との間で，以下の条件 1,2 が，$S_1$ 上の任意の遷移 $T_n$ に対して成り立つこと：

条件 1：$S_k$ を，以下のような，隣接した[3]区間 $I_{(k,1)}$, $\cdots$, $I_{(k,n)}$, $\cdots$ に区切ることができること：区間 $I_n$ は，遷移 $t_B \in B_{(k,m)}$ で始まり，遷移 $t_E \in E_{(k,m)}$ を終点とする（区間 $I_{(k,n)}$ は，$S_1$ 上の遷移 $T_n$ に対応する）．

条件 2：要求仕様の各レジスタ $reg_{(1,r)}$ について，以下が成り立つこと：$T_n$ における $reg_{(1,r)}$ の変化が don't care でなければ，$T_n$ の終点の状態 $s$ と区間 $I_{(k,n)}$ 中の状態 $s' \in L_{(k,m,r)}$ において，

$$reg_{(1,r)}(s) = R_i(reg_{(k,1)}(s'), reg_{(k,2)}(s'), \ldots)$$

が成り立つこと．

### 3.2 各詳細化の正しさの証明

**3.1** では，レベル $k$ の回路が要求仕様を満たすことを定義したが，実際の証明作業では，各詳細化（および変形）ごとに，その正しさを証明する．レベル $k-1$ の回路が要求仕様を満たしているとき，それからレベル $k$ への詳細化（変形）の正しさが証明されれば，レベル $k$ も（推移的に）要求仕様を満たす．

以下，各詳細化（変形）の正しさをどのように示したか，および，我々の証明支援系を用いて証明実験を行った結果について述べる．

レベル 1,2 間，およびレベル 2,3 間の詳細化は，いずれも（従来通り）一本の遷移を展開することにより行った [2]．これらの場合，**3.1** における $B_{(k,m)}$, $E_{(k,m)}$, $L_{(k,m,r)}$, $R_{(k,r)}$ を，以下のように定めた（図 6 参照）：要求仕様における状態時系列上の各遷移（いずれも `cycle`）に対して

- $B_{(k,m)}$ は，{ 遷移 `fetch` }．
- $E_{(k,m)}$ は，各命令の最後の遷移の集合．例えばレベル 2 では { 遷移 `exec` }．
- $L_{(k,m,r)}$ は，いずれのレジスタ $reg_{(1,r)}$ に対しても，{ <各命令の最後の遷移，後> }（展開に用いた遷移系列の終点の状態）．例えばレベル 2 では，{ <遷移 `exec`, 後> }．
- $R_{(k,r)}$ は，（詳細化ではレジスタ名を変更しなかったので）$reg_{(k,r)}$．□

以上の定義のもとで，これらの詳細化の正しさの証明では，（従来どおり）「展開に用いたレベル $k+1$ の遷移系列の動作内容が，展開したレベル $k$ の遷移の動作内容を満たす」ことを示す．

なお，レベル 4,5 間の詳細化については，$B_{(k,m)}$, $E_{(k,m)}$, $L_{(k,m,r)}$, $R_{(k,r)}$ は上と異なったものになるが，証明のやり方は同じである．

証明は，両レベルの記述，およびその間の対応関係の記述を，我々の証明支援系[4, 5]に与えることにより，自動で行えた．我々の証明支援系は，場合分け，項書き換え，整数上の論理式（プレスブルガー文）の恒真性判定などを，一定の手順で組み合わせて，自動的に実行する機能を持つ．

[3]各区間が，互いに接しているか，後から始まった区間の終りが前の区間の終りより先にならないように重なっていること．

例えば，レベル 1 からレベル 2 への詳細化の正しさの証明において，「レベル 1 の遷移 `cycle` に書かれた PC の動作仕様が，両レベル間の遷移の対応関係のもとで，レベル 2 で正しく実現されている」ことを，証明支援系は次のようにして自動証明した．

(1) レベル 1 の記述中の PC の動作内容の公理

$$PC(cycle(s)) == if\ (inst(mget(MEM(s), PC(s))) = RCF\ then \cdots)$$

と，両レベル間の状態遷移の対応関係

$$cycle(s) == exec(fetch(s))$$

とから，式

$$PC(exec(fetch(S0))) == if\ (inst(mget(MEM(S0), PC(S0))) = RCF\ then \cdots)$$

を作る（$S0$ は，遷移系列の始点を表す，構造的帰納法の定数[5]）．この式が，レベル 2 の遷移 `fetch` および遷移 `exec` の動作内容の公理のもとで成り立てばよい．

(2) レベル 2 の遷移 `fetch`, `exec` の動作内容の公理を書き換え規則として，上の式を書き換える．この場合は

$$if\ (inst(mget(MEM(S0), PC(S0))) = RCF\ then \cdots) = (if\ (inst(mget(MEM(S0), PC(S0))) = RCF\ then \cdots))$$

となる．

(3) 上の式が成り立つことを，整数上の論理式の恒真性判定ルーチンを用いて示す．

### 3.3 プリフェッチ導入の正しさの証明

レベル 3,4 間の変形（命令プリフェッチの導入）では，**3.1** における $B_{(k,m)}$, $E_{(k,m)}$, $L_{(k,m,r)}$, $R_{(k,r)}$ を，以下のように定めた（図 6 参照）：要求仕様における状態時系列上の各遷移（いずれも `cycle`）に対して

- $B_{(k,m)}$ は，{ 遷移 `fetch`, プリフェッチを行う各遷移 }．
- $E_{(k,m)}$ は，各命令の最後の遷移の集合．
- $L_{(k,m,r)}$ は，プリフェッチによって値が変わるレジスタ（PC, IR）については，{ <プリフェッチを行う各遷移，前> }．それ以外のレジスタについては，{ <各命令の最後の遷移，後> }．
- $R_{(k,r)}$ は $reg_{(k,r)}$．□

この正しさの証明では，以下を，それぞれ証明支援系により示した：レベル 4 で（プリフェッチを行う一回の）遷移 `EF` で実現された，レベル 3 の各遷移系列 `fetch(E(s))` について，

- 遷移 `E` の実行前後で，PC,IR の値が変わらないこと．
- 遷移 `fetch` の実行前後で，PC,IR 以外のレジスタの値が変わらないこと．
- 遷移系列 `fetch(E(s))` の動作内容と遷移 `EF` の動作内容が等価であること．□

以上より，レベル 3 における遷移 `E` の実行直後（遷移 `fetch` の実行直前）の各レジスタ値が，レベル 4 において，`EF` の実行直前（あるいは直後）に実現されていることが保証される（図 6 参照）．これらの証明は，自動で行えた．

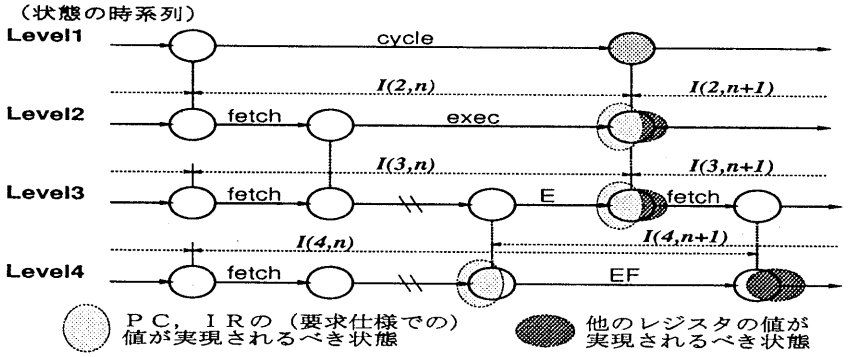



図 6　KUE-CHIP2 の各レベル間の状態の対応

表 2　証明に要した CPU 時間

SONY NWS-5000 (100MIPS, 64MB メモリ) 使用.

| | |
|---|---|
| レベル 1 ～ レベル 2 | 0.1 時間 |
| レベル 2 ～ レベル 3 | 4.6 |
| レベル 3 ～ レベル 4 | 0.06 |
| レベル 4 ～ レベル 5 | 8.0 |
| レベル 5 の機能部品（ALU） | 36.0 |

### 3.4　機能部品の実現の正しさの証明

今回は，得られた回路の正しさを完全に保証するため，各詳細化の正しさの証明に加えて，レベル 5 の設計で導入した 8 ビット ALU の実現の正しさ（ゲートレベルの回路が，正しく加減算等を行うこと）も証明した[6]．なお，他の機能部品については，パルテノンの部品ライブラリに含まれているものをそのまま使用したので，特に証明は行わなかった．

証明では，まず (i)ALU が行うべき算術・論理演算の内容（仕様）と，ALU 内の各論理ゲートの結線（実現）とを，それぞれプレスブルガー文（整数変数，論理変数，整数上の加減算・比較演算，および論理演算からなる一階述語論理式）で記述し，次に (ii) 任意の入力値に対して，それに対する実現の出力値が，仕様に書かれた演算の結果と等しいことを，プレスブルガー文真偽判定ルーチンにより調べた．この方法では，ALU が正しく演算を行うことを，整数上で証明することができる．証明は，約 $2^{20}$ 通りの回路入力について出力が正しいことを示す必要があるために，約 36 時間（SONY NWS-5000, 100MIPS 使用）の CPU 時間を要したが，自動で行うことができた．

### 3.5　証明実験の結果

設計および証明には，十数個の設計誤りの修正などの作業も含めて，2 週間程度要した．証明が成功したときの CPU 時間を表 2 に示す．

今回，例題として用いた KUE-CHIP2 は 8 ビット CPU であったが，各レベル間の詳細化の正しさの証明は，任意のレジスタビット長に対して行っているので，16 ビット CPU，あるいは 32 ビット CPU であっても，命令セットの数が KUE-CHIP2 程度であれば，今回と同程度の時間で証明できると思われる．

## 4　おわりに

本稿では KUE-CHIP2 を，最適化も含めて段階的に設計し，その実現の正しさを証明した．また，最適化を行った場合でも統一した枠組で証明できるようにするため，従来我々が用いていた実現の正しさの定義を，拡張した．

今回の設計・証明実験では，我々の段階的設計法により，KUE-CHIP2 のような比較的機能が複雑な回路であっても，他と比べても遜色のない性能の回路を得ることができた．証明もほぼ自動で，かつ実用的な時間内に行うことができた．これより，我々の設計法や証明支援システムが実用上有用であると思われる．

## 参考文献

[1] 神原弘之, 安浦寛人: “LSI 設計教育と計算機ハードウェア教育のためのマイクロプロセッサ: KUE-CHIP2,” 信学技報 VLD95-92, pp. 45–52 (1995–07).

[2] 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), J77-A, No. 3, pp. 420–429 (1994–03).

[3] 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一: “代数的手法を用いた順序回路の段階的設計支援システムにおける状態図変形機能,” 第 8 回回路とシステム軽井沢ワークショップ論文集, E2-2 (1995–04).

[4] Kitamichi, J., Morioka, S., Higashino, T. and Taniguchi, K.: “Automatic Correctness Proof of the Implementation of Synchronous Sequential Circuits Using an Algebraic Approach,” *Proc. 2nd Int. Conf. on Theorem Provers in Circuit Design (TPCD 94)* (Kropf, T. and Kumar, R. eds.) , Vol. 901 of LNCS, pp. 165–184, Springer Verlag (1995).

[5] 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一: “代数的言語で記述した抽象的順序機械型プログラムの設計検証の自動化,” 情報処理学会論文誌第 36 巻第 10 号, pp. 2409–2421 (1995–10).

[6] 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一: “整数上の論理式の恒真性判定アルゴリズムを用いた組み合わせ論理回路の実現の正しさの証明,” 第 49 回情処全大 (6), pp. 87–88 (1994–09).

[7] 北嶋 暁, 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一: “代数的言語 ASL による回路設計支援システムにおける SFL 記述への詳細化とその変更及びそれらの正しさの検証,” 第 49 回情処全大 (6), pp. 99–100 (1994–09).

[8] Windley, P. J., “Verifying Pipelined Microprocessors,” *Proc. CHDL '95*, pp. 503–511 (1995–08).