Vol. 35 No. 8　　情　報　処　理　　Aug. 1994

解　説

論理設計の形式的検証

5. 代数的手法による仕様記述と設計及び検証†

谷口健一††　北道淳司††

1. はじめに

本稿では代数的言語・手法を用いて，同期式順序回路の要求仕様記述，段階的設計，形式的検証をどのように行うかについて説明する．ここでは，一つの制御部をもつ同期式順序回路を対象とする．

一般に記述言語は，記述する立場からは記述能力が高いほうが便利である．代数的言語より高階論理[2]や時相論理のほうが記述能力は高いが，一般に道具立てが大きくなると，検証支援系などでの実行時間が多くかかるなど不都合もないわけではない．その観点からは，同期式順序回路の上流工程における設計に必要にして十分な記述言語，枠組みを用いるのがよい．代数的言語及び枠組みはそれらの点で有力な一つの候補であると思われる[3]．検証について言えば，代数的枠組みは公理による項書換えなど，検証に用いる手段は単純であるが，反面，検証の手間は繁雑であり，支援系による設計や検証の容易化，自動化が重要である．

設計検証を実際上可能にするための段階的設計について **2.** で，代数的言語を用いて順序回路をどのように記述するかについて **3.** で述べる．メモリ内のデータをソートする回路を例に要求仕様から論理設計レベルに至るまでの設計について **4.** で，それらの設計に対する検証について **5.** で述べる．その回路に対しては，筆者らが開発した代数的言語 ASL に基づく設計支援システムにより設計及び検証が行われた．

2. 段階的設計と検証

同期式順序回路の要求仕様として，処理内容からみた回路の機能（たとえばメモリ内のデータのソートを一動作で行うとみなし，その動作の前後で各部品の内容がどう変わるか，どのような述語をみたすか）を記述する．設計は，どんなアルゴリズムでそれを実現するか，どんな回路アーキテクチャにするか，そこの基本動作をどんな順で実行すればよいかなどを逐次決めていくことである．

設計が正しいことを証明する（本稿で検証というのはこの意味）のは一般に困難である．回路全体の自由な設計を許した場合の検証や，計算の最終結果のみが等しければよいといった意味での検証は，回路規模が大きくなればきわめて難しい．したがって，正しさの証明を実際上可能にするには，設計法をある程度制限して，局所的に詳細化せざるをえない．しかし一方では，設計者の意図する性能の回路が得られる設計法であることも重要である．

検証を実際上可能とするためには，たとえば，仕様の段階的詳細化は，要求仕様の一動作を次のレベルの動作の系列で置き換え，以降のレベルでは，各動作ごとに独立して，下位のより具体的な動作からなる系列で置き換える．そうすれば，検証は，その置換えにより，置き換えられたもとの各動作の内容がそれぞれ正しく実現されているかを調べるだけでよい．

しかし，回路アーキテクチャを決めるようなレベルにおいては，分岐条件判定のために必要なデータをメモリなどから読み出し，特定のレジスタに転送しておくための状態遷移を新たに導入することも必要となる．また，上述のような詳細化では，上位レベルの各動作はそれぞれ独立して下

† Design and Verification of Sequential Logic Circuits Based on Algebraic Method by Kenichi TANIGUCHI and Junji KITAMICHI (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University).

†† 大阪大学基礎工学部情報工学科

位レベルの動作系列で展開されるので，あるレジスタに値が保持されているのに，次の動作系列で同じ値をそのレジスタに転送する，というような無駄な動作を含むかもしれないし，引き続く二つの動作が一つの動作として実行できるかもしれない．回路性能の点からはそのような無駄な動作の削除や動作の併合などを行うことが必要である．これらの変形は，支援系により，等価性を保証しながら行うのがよい．

3. 代数的言語を用いた同期式順序回路の記述スタイル

一般に代数的言語における一つの記述（テキスト）は，関数の構文の指定といくつかの公理からなる．公理とはプリミティブなデータタイプ及びその関数を用いて，新しいデータタイプ及びその上の関数について，その性質やその値の定義式を変数付きの等式（$\alpha==\beta$ の形）で書いたものである．公理の変数にそのデータタイプの要素を代入すると一般に無限個の等式が得られるが，一つの記述はそれらの等式集合から得られる表現式集合上の合同関係（関数演算で閉じている同値関係）を表わす[1]．

同期式順序回路の仕様は，任意のレベルにおいて，回路の各動作に対して抽象的レジスタ群の値がどのように変わるかを指定する動作内容の記述 $\boldsymbol{D}$，どの条件が成り立つときにどの動作を実行すべきかを指定する実行制御の記述 $\boldsymbol{C}$ からなる（拡張有限状態機械モデル）．

代数的に記述するために，部品の値などのすべての情報を含んでいる抽象状態という抽象データタイプを導入する．回路の動作は抽象状態を引数とし抽象状態を値域とする状態遷移（“状態遷移関数”と呼ぶ）で表わす．抽象状態は，初期状態から現在に至るまでの状態遷移の系列に対応する．抽象状態 S における各レジスタの値は，そのレジスタに対応する関数 F（“状態成分関数”と呼ぶ）を用いて $F(S)$ で表わされる．

動作内容 $\boldsymbol{D}$ は，各状態遷移 T について，抽象状態 S での各成分の値 $F(S)$ と，S で状態遷移 T を行った後の抽象状態 $T(S)$ での各成分の値 $F(T(S))$ の関係を公理を用いて記述したものである．

たとえば状態遷移 $T1$ におけるデータ転送（レジスタ $F1$ には $F1$ と $F2$ の値の和を，レジスタ $F2$ には $F1$ と $F2$ の値の差を入れる）は，次のように表わせる．

$$F1(T1(s))==F1(s)+F2(s)$$
$$F2(T1(s))==F1(s)-F2(s)$$
$$F3(T1(s))==F3(s)$$

なお以下では，$F3(T1(s))==F3(s)$ のように値が変わらないことを表わす公理は省略することがある．

各部品の機能及び部品間の結線を決めた論理設計レベルでも同じ記述スタイルで動作内容を記述できる．

実行制御 $\boldsymbol{C}$ は，代数的には，抽象度の高いレベルでは有限の状態名集合を導入して，現在の状態名を表わす特別な状態成分関数 (CONTROL) と各状態遷移 T について T が実行されるべきかの遷移条件を記述する関数 (VALID) を用いて表わせる．これらの関数による公理の内容は，いわゆる状態遷移図を用いて表わすことができ，以下では，これらの公理の内容を状態遷移図で表わすことがある．

レジスタ転送レベル以降のレベルでは，CONTROL を複数のフリップフロップで実現したり，マイクロプログラムカウンタで実現するが，いずれのレベルでも同じスタイルで記述する．

各レベルで同じ記述スタイルを採用すれば，たとえば検証には，同一の検証支援機能を用いることができる．

4. 代数的言語を用いた同期式順序回路の設計

4.1 要求仕様記述

メモリ MEM に格納されているデータをレジスタ I1, I2 で指定される範囲内だけソートする回路を考える．テキストに対するパラメータ N, H が与えられており，メモリの最大アドレスは N, 各要素のデータタイプは大きさ H までの正数である．この回路の要求仕様は，たとえば**表-1**のように表わされる☆（以下，これをレベル1の記述と呼ぶ）．状態成分として，メモリ MEM, レジスタ I1, I2 を導入し，状態遷移としてソー

☆ 仕様をいわゆる補助関数を用いて書く枠組みなども提案されている[4]．同じ内容を異なる基本述語を用いていろいろな公理で書ける．たとえば，sort 1 は

sort 1′: $0\leqq I1(INIT)\leqq p\leqq q\leqq I2(INIT)\leqq N\rightarrow$
$MEM(sort(INIT))[p]\leqq MEM(sort(INIT))[q]==TRUE$

のように書いてもよい．検証の仕方が変わる．

表-1 ソート回路の要求仕様（レベル1の記述）

```
(a) 動作内容の公理
sort 1: 0≦I1(INIT)≦I2(INIT)≦N→
          ordered(MEM(sort(INIT)),
              I1(INIT), I2(INIT))==TRUE
sort 2: 0≦I1(INIT)≦I2(INIT)≦N→
          seteq(MEM(INIT), MEM(sort(INIT)),
              I1(INIT), I2(INIT))==TRUE
sort 3: 0≦I1(INIT)≦I2(INIT)≦N→
          (0<I1(INIT)→
              arrayeq(MEM(INIT), MEM(sort(INIT)),
                  0, I1(INIT)−1))
        ∧(I2(INIT)<N→
              arrayeq(MEM(INIT), MEM(sort(INIT)),
                  I2(INIT)+1, N))==TRUE
(b) 実行制御の公理
CONTROL(INIT)==start;
CONTROL(sort(INIT))==end;
VALID(INIT)==TRUE;
VALID(sort(INIT))==TRUE;
(* τ: α==β におけるτは公理 α==β のラベルである*)
```

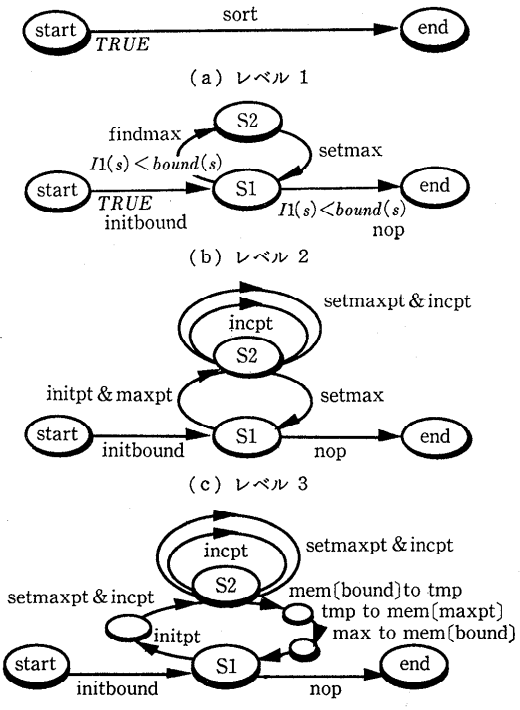
トを行う状態遷移 sort を導入する．

動作内容は**表-1** (a) のように記述される．初期抽象状態 INIT におけるメモリ MEM，レジスタ I1，I2 の値は初期設定されている，公理 sort 1 は，初期抽象状態 INIT におけるレジスタ I1，I2 で指定される範囲内の MEM の要素が sort 後には昇順に並んでいること，公理 sort 2 は，sort 前後でレジスタ I1，I2 で指定される範囲内の MEM の要素が多重集合として等しいこと，公理 sort 3 は，レジスタ I1，I2 で指定される範囲外の MEM の要素は不変であることを表わしている．ただし，ordered (a, i, j) は配列 a の i, j 間の要素が昇順であること，seteq (a, b, i, j) は配列 a, b の i, j 間の要素が多重集合として等しいこと，arrayeq (a, b, i, j) は配列 a, b の i, j 間の要素がそれぞれ等しいことを表わす基本述語である．なお簡単にするため，関数の構文やパラメータ指定の記述は省略されている．

実行制御（**表-1** (b)）は初期状態 start から状態遷移 sort を行って状態 end に遷移して停止するように書いている（**図-1** (a)）．

4.2 詳 細 化

設計検証を実際上可能とするために，次のような段階的設計が考えられる．レベル k の記述 $t=$ <動作内容 $\boldsymbol{D}$, 実行制御 $\boldsymbol{C}$> が与えられているとしよう．設計者は以下の手順で一段階の詳細化を行い，レベル $k+1$ の記述 $t'=<\boldsymbol{D}', \boldsymbol{C}'>$ を得る．これを必要な具体レベル（たとえば論理設計レベル）まで繰り返し行う．



(* (c), (d)では分岐条件の記述は省略されている *)

図-1 ソート回路の詳細化の概要

(1) 必要なら新たな状態成分を導入し，より具体的な状態遷移を考案し，記述する（レベル $k+1$ の動作内容 $\boldsymbol{D}'$）．

(2) 対応関係 $\boldsymbol{M}$ を考案する．対応関係としては以下の二つがある（それぞれ公理で記述される）．

- レベル k の各状態遷移 T とレベル $k+1$ の状態遷移列（代数的には T の関数型の記述であり，if 関数を用いた条件分岐，順次実行，相互再帰などを用いる），あるいは，始点終点をもつ状態遷移図．
- レベル k の各状態成分 F とレベル $k+1$ の状態成分の対応関係．

(3) レベル k の実行制御 $\boldsymbol{C}$ と $\boldsymbol{M}$ から $\boldsymbol{C}'$, を自動合成する．必要なら状態遷移図の変形（最適化）も行う．これらは支援系を用いて等価性を保証しつつ行う．状態遷移図の変形ではどんな変形をどこにどのような順で適用するかは一般に設計者が考える．

設計の正しさの検証は，レベル k の動作内容 $\boldsymbol{D}$ の各公理がレベル $k+1$ の動作内容 $\boldsymbol{D}'$ と対応関係 $\boldsymbol{M}$ 上の定理として成り立つことを調べれば

よい．ビット長などを具体的に決めたならば，上位レベルの記述における形式パラメータの値を決めた仕様の「実現」となる．

表-1 の要求仕様をマックスソート法で実現する段階的設計の例を以下に述べる．設計は，筆者らの代数的言語 ASL による設計支援システムを用いて行われた．各レベルの詳細化が正しいことは検証済みである．検証の方法や手間については **5.** で説明する．

ASL では，文脈自由文法を用いて関数の構文を自由に指定できる（mixfix など）[4]．設計支援システムの主な機能をあげる．

- 下位レベルの状態遷移図合成機能：上位レベルの動作の実行制御の記述 C と対応関係 M（再帰は末端再帰に限る）から下位レベルの動作の実行制御の記述 C' を合成する．
- 状態遷移図簡約機能：状態遷移図の変形（最適化）に必要と思われる 10 数種のルール（冗長な遷移の除去，遷移の順序の入れ換えなど）を用意しており，各ルールの適用可能条件の判定やルールの適用を行う．
- 検証支援機能：項書換え，場合分けの管理，補題の追加，整数上の論理式の恒真性判定[6]，構造的帰納法を用いたときの不変式や検証済みパスの管理，一定の手順による検証作業の自動実行[8] などの機能からなる．
- 入力論理合成機能及びマイクロプログラム合成機能：部品の機能やバス構成（及びマイクロ命令の構成）を決めているようなレベルにおいて，各部品の制御入力論理式やゲートの制御入力論理式あるいはマイクロプログラムを合成する．

レベル 1 の状態遷移 sort を，マックスソート法を用いて詳細化する．レベル 2 では新たに状態成分 bound, max, maxpt を導入し，状態遷移 initbound, findmax, setmax, nop を導入する．

動作内容は **表-2** (a) のものとする．状態遷移 initbound では bound が I2 に設定され，findmax では MEM の I1 から bound の間の最大要素の場所が maxpt に，その最大値が max に格納される．setmax では bound の位置の要素が maxpt の位置に，max の内容が bound の位置に格納され，bound の値が 1 減らされる．isge (a, $p1$, $p2$, $p3$) は配列 a の要素 $a[p3]$ が，$a[p1], \cdots, a[p2]$ のどれよりも小さくないことを表わす基本述語である．たとえば公理 smax 3 は遷移 setmax で MEM の bound と maxpt の位置以外の要素は不変であることを表わしている．

表-2 マックスソート回路のレベル 2 の記述

```
(a) 動作内容
(*** 遷移 initbound の動作内容 ***)
bound(initbound(s))==I2(s);
(*** 遷移 findmax の動作内容 ***)
define '前提条件 1':='(0≦I1(s)≦bound(s)≦N)';
fmax 1: 前提条件 1→
  (I1(s)≦maxpt(findmax(s))≦bound(s))==TRUE;
fmax 2: 前提条件 1→
  isge(MEM(s), I1(s), bound(s), maxpt(findmax(s)))
  ==TRUE;
fmax 3: 前提条件 1→
  (max(findmax(s))=MEM(s)[maxpt(findmax(s))])
  ==TRUE;
(*** 遷移 setmax の動作内容 ***)
  bound(setmax(s))==bound(s)−1;
define '前提条件 2':=
  '(0≦maxpt(s)≦N∧0≦bound(s)≦N)';
smax 1: 前提条件 2→
    MEM(setmax(s))[bound(s)]=max(s)==TRUE;
smax 2: 前提条件 2→
  (¬(maxpt(s)=bound(s))→
    MEM(setmax(s))[maxpt(s)]=MEM(s)[bound(s)])
  ==TRUE;
smax 3: 前提条件 2→
  ((maxpt(s)≦bound(s)→
  [ (0<maxpt(s)→
    arrayeq(MEM(s), MEM(setmax(s)), 0, maxpt(s)−1))
  ∧(maxpt(s)+1<bound(s)→
    arrayeq(MEM(s), MEM(setmax(s)),
            maxpt(s)+1, bound(s)−1))
  ∧(bound(s)<N→
    arrayeq(MEM(s), MEM(setmax(s)), bound(s)+1, N))])
  ∧(bound(s)<maxpt(s)→
  [ (0<bound(s)→
    arrayeq(MEM(s), MEM(setmax(s)), 0, bound(s)−1))
  ∧(bound(s)+1<maxpt(s)→
    arrayeq(MEM(s), MEM(setmax(s)),
            bound(s)+1, maxpt(s)−1))
  ∧(maxpt(s)<N→
    arrayeq(MEM(s), MEM(setmax(s)), maxpt(s)+1, N))]))
  ==TRUE;
(b) 実行制御
CONTROL(INIT)==start;
CONTROL(initbound(s))==S1;
    ⋮
VALID(INIT)==TRUE;
VALID(initbound(s))==VALID(s)∧CONTROL(s)=start;
    ⋮
```

状態遷移 sort とこれらの遷移との対応関係は **表-3** のものとする．状態遷移 findmax と setmax を bound が I1 に等しくなるまで繰り返すように指定されている．レベル 1 とレベル 2 の成分は同名のものを対応させるので成分の対応関係は省略する．sort をこの対応関係により置き換えてレベル 2 の状態遷移図を合成する（**表-2** (b), **図-1** (b)）．

レベル2より，I1 から bound までのメモリ MEM の要素を順に調べるための状態成分 pt を導入して findmax を展開し，レベル3の記述を得る（**図-1** (c)）．さらに，レベル3の状態遷移 initpt&maxpt, setmax を展開し，レベル4の記述を得る（**図-1** (d). 動作内容は**表-4**）．たとえば setmaxpt & incpt に関する公理は maxpt, pt, max へのパラレル代入を表わしている．

4.3 レジスタ転送レベルへの設計

レジスタ転送レベルへの設計は，

（1）設計者が，実現可能な各部品の機能を決定し，記述する．あるいは使用可能な部品を選択する．

（2）設計者が，接続関係を決定し，記述する．この(1)と(2)よりこのレベルの動作内容（1クロックでのデータ転送）が決定する．

（3）設計者が上位レベルの一状態遷移を，各部品，各ゲートの制御入力にどのような値を与え，何クロックで実現できるかを考案し（状態遷移の対応関係に相当），それが正しいかを証明する（部品の機能，クロック数などを限定すれば，自動的に制御入力値系列を生成することは可能である）．

図-2 のアーキテクチャを用いてレベル4の動作内容を実現することにしよう．

各部品の機能，それらの接続関係を公理で表わす．**表-5** (a) では，図-2 の各部品の機能（たとえば，レジスタ REG の出力は，一クロック後，制御入力ctl_{REG} の値が HD のときは不変，LD のときは入力 $data_{REG}$ の値にセットされる）を記述している．**表-5**(b)では，バスへの出力（たとえば，ADDRESS-BUS1 には，ゲートの制御入力 $g11$ が ON，それ以外が OFF のとき，bound の値が出力されるなど）を指定している．**表-5**(c)では，各部品やゲートの入力（たとえば，bound の制御入力には回路全体の状態遷移を表わす CK の第2引数 ctl_{bound} が与えられ，データ入力には ADDRESS-BUS1 が接続していること）を指定している．表-5 により，図-2 のアーキテクチャで実行可能なデータ転送が定義される．

レベル4の各状態成分は，導入した部品で実現

表-3 レベル1とレベル2の対応関係

```
sort(s)==loop1(initbound(s));
loop1(s)==if(I1(s)<bound(s))
              then loop1(setmax(findmax(s)))
              else nop(s);
```

表-4 レベル4の動作内容

```
pt(initpt(s))==I1(s);
maxpt(setmaxpt&incpt(s))==pt(s);
pt(setmaxpt&incpt(s))==pt(s)+1;
max(setmaxpt&incpt(s))==MEM(s)[pt(s)];
⋮
```

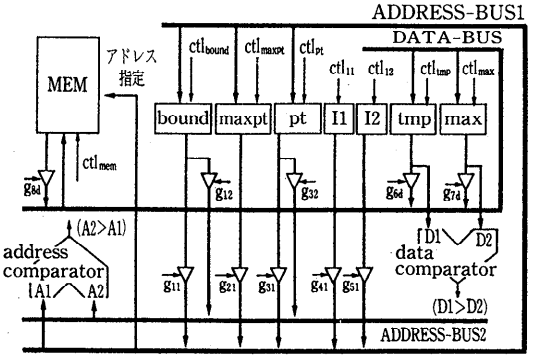


図-2 レベル5で採用するアーキテクチャ

表-5 レベル5の動作内容

```
(a) 部品の機能
REG(ck_R(s_REG, ctl_REG, data_REG))
    ==    if ctl_REG=HD then REG(s_REG)
        else if ctl_REG=LD then data_REG;
  ⋮
(b) バスへの出力の指定
ADDRESS-BUSI(s, g11, g21, g31, g41, g51)==
      if g11=ON∧g21=OFF∧...∧g51=OFF
        then    CNT(proj_1(s))
 else if g11=OFF∧g21=ON∧g31=OFF∧...∧g51=OFF
        then    REG(proj_2(s))
 else if g11=OFF∧g21=OFF∧g31=ON∧g41=OFF
      ∧g51=OFF    then    CNT(proj_3(s))
 else if g11=OFF∧...∧g31=OFF∧g41=ON∧g61=OFF
        then    REG(proj_4(s))
 else if g11=∧OFF∧g41=OFF∧g51=ON
        then    REG(proj_5(s));
  ⋮
(c) 各部品のデータ入力の指定
CK(s, ctl_bound, ctl_maxpt, ctl_pt, ctl_I1, ctl_I2, ctl_tmp, ctl_max, g11,
      g12, ..., gsd, ctl_μPC, adr, ctl_MEM)
  ==[ck_C(proj_1(s), ctl_bound, a1bus)
    ck_R(proj_2(s), ctl_maxpt, a1bus)
    ck_C(proj_3(s), ctl_pt, a1bus)
    ck_R(proj_4(s), ctl_I1, input)
    ck_R(proj_5(s), ctl_I2, input)
    ck_R(proj_6(s), ctl_tmp, dbus)
    ck_R(proj_7(s), ctl_max, dbus)
    ck_MEM(proj_8(s), ctl_MEM, a1bus, dbus)
    ck_μpc(proj_9(s), ctl_μPC, adr,
          a2bus>a1bus,
          REG(proj_7(s))>REG(proj_6(s)))];
  where a1bus:=ADDRESS-BUS1(s, g11, g21, g31, g41, g51)
  where a2bus:=ADDRESS-BUS2(s, ...)
  where dbus:=DATA-BUS(s, ...)
(* where は ASL のマクロ記法であり，α==β where
r:=δ は，α 及び β に出現する r を δ に置き換えた公理
を表す *)
```

され，その対応関係は**表-6** (a) で表わされる．たとえば，bound はカウンタ CNT で，maxpt はレジスタ REG で実現されることを表わしている．回路全体の状態は各部品の部分状態の並びで表わされ，bound はその1番目の要素である．proj_n は n 番目の要素の部分状態を取り出す基本関数である．

レベル4の各状態遷移に対して，その動作内容を実現する制御入力の並び (CK の引数) を考案する（**表-6**(b)）．たとえば，遷移 setmaxpt&incpt に対しては，g_{31} を ON に (ADDRESS-BUS 1 に pt が出力される)，maxpt の制御入力を LD にして，pt から maxpt へのデータ転送を行う．同時に，g_{8d} を ON とし，アドレス指定が pt となっているメモリから DATA-BUS に出力し，その値を max が読み取る．さらに $\mathrm{ctl_{pt}}$ を +1 にし，pt の値を1増加させる．レベル4の記述において遷移後の値の指定がない成分に対しては，その引数値は，どのような値でもよい（表-6 では -- で表わしている）．**5.** で，設計者が考案したアーキテクチャ（表-5）と対応関係（表-6）により，レベル4の動作内容（表-4）が実現されているかどうかの検証について述べる．

次に，制御部をマイクロプログラム方式で実現するとしよう*．回路全体の状態遷移 CK を1マイクロコードで実現することにする（いわゆる水平型マイクロプログラム）．すなわち，状態遷移の対応関係における CK の引数値の並びがその状態遷移を実現するマイクロコードとなる．マイクロプログラムカウンタの機能や実行条件判定に用いる組合せ回路の機能・個数を決定する．この例では，マイクロプログラムカウンタとして，$\mathrm{ctl_{\mu pc}}$ = BRN 1∧cond 1 = FALSE∨$\mathrm{ctl_{\mu pc}}$ = BRN 2∧cond 1 = TRUE∨$\mathrm{ctl_{\mu pc}}$ = BRN 3∧cond 2 = TRUE の分岐条件により入力 adr の値が μpc にセットされる条件分岐，カウントアップ，停止の機能をもつカウンタを用いる．その機能により，レベル4の実行制御（図-1 (d)）が実現できるように各分岐部をより簡単な条件の分岐群に展開し，分岐条件の判定に必要なデータをメモリから取り出す遷移（状態 S2 における MEM[pt]>max の判定に必要なデータを MEM から tmp に取り出す遷移）を追加し，さらに，状態の簡約を行う．次いで，

表-6 レベル4と5の対応関係

(a) 状態成分の対応

```
bound(s)==CNT(proj_1(s));
maxpt(s)==REG(proj_2(s));
pt(s)==CNT(proj_3(s));
I1(s)==REG(proj_4(s));
I2(s)==REG(proj_5(s));
max(s)==REG(proj_6(s));
tmp(s)==REG(proj_7(s));
MEM(s)==MEM(proj_8(s));
```

(b) 状態遷移の対応

```
initbound(s)==CK(s, LD, --, --, HD, HD, --, --,
        OFF, OFF, OFF, OFF, ON
        OFF, OFF, OFF, OFF, OFF, +1, --, RD);
initput(s)==CK(s, HD, --, LD, HD, HD, --, --,
        OFF, OFF, OFF, ON, OFF,
        OFF, OFF, OFF, OFF, OFF, +1, --, RD);
setmaxpt&incpt(s)==CK(s, HD, LD, +1, HD, HD, LD, --,
        OFF, OFF, ON, OFF, OFF,
        OFF, OFF, ON, OFF, OFF, +1, --, RD);
  ⋮
```

* 直接論理方式の場合は，状態レジスタの機能，実行条件判定に用いる組合せ回路の機能・個数を決定し，分岐部の展開，状態遷移の追加，状態の簡約を行い，各制御入力への入力論理を合成する．さらに状態割当てを行い，状態レジスタの制御入力への入力論理を合成する[7]．

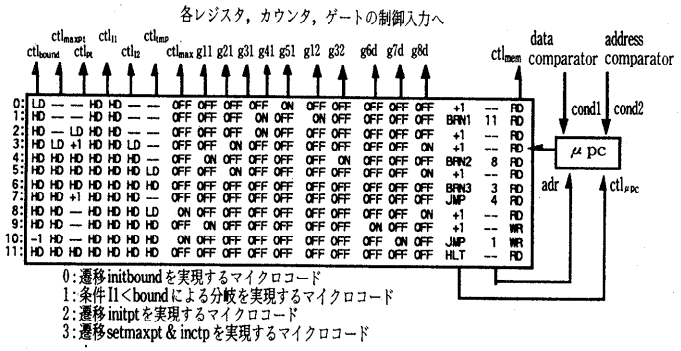


図-3 レベル5の制御部

各状態にマイクロプログラムのアドレスを割り付け，マイクロプログラム（**図-3**）を得る．これらは支援系を用いて正しさを保証しながら行った．

5. 詳細化の正しさの検証

4.2, **4.3** で述べた詳細化の正しさを保証するためには，次の(1)，(2)を示せばよい．

(1) $\boldsymbol{D}$ の各公理が，$\boldsymbol{D}'$ と $\boldsymbol{M}$ における定理[2]である（$\boldsymbol{D}$ の公理の変数に任意の要素を代入して得られる各等式が $\boldsymbol{D}'$ と $\boldsymbol{M}$ 上の合同関係で成り立つ）こと．

(2) $\boldsymbol{D}'$ と $\boldsymbol{M}$ の公理系が無矛盾である（異なる“値”が合同とならない）こと．条件(1)より，レベル k の各遷移がレベル $k+1$ で正しく実現される．条件(2)は通常，公理の形によって保証される．たとえば，関数型や順序機械型の記述なら無矛盾である[4]．

代数的枠組みでは，定理であることを示すために次のような手段を用いる[4]．

(a) 公理による書換え
(b) 場合分け
(c) 基本データタイプ上の論理式の恒真性判定
(d) 基本関数，基本述語に関する性質（補題）の追加
(e) 構造的帰納法☆

前述の設計のレベル2への詳細化の正しさの証明について説明する．公理 sort 1 に関する公理が定理として成り立つことは，

$$0 \leqq \mathrm{I1(INIT)} \leqq \mathrm{I2(INIT)} \leqq N \rightarrow \mathrm{ordered(MEM(sort(INIT)),\ I1(INIT),\ I2(INIT))}$$

が真であることを示すことである．以下，構造的帰納法を用いて証明する．

(1) $0 \leqq \mathrm{I1(INIT)} \leqq \mathrm{I2(INIT)} \leqq N$ を仮定する．

(2) 図-1 (b) の状態 S1 における不変式を検証者が考案する．不変式 $R(s)$ を**表-7** に示す．

状態 S1 から findmax, setmax を行って状態 S1 に至るパスに対する証明すべき式は，定数 S を用いて，$(R(S) \wedge \mathrm{I1}(S) < \mathrm{bound}(S)) \rightarrow R(\mathrm{setmax(findmax}(S)))$ で表わされる．$\mathrm{I1}(S) < \mathrm{bound}(S)$ は S1 における findmax の分岐条件である．

(3) 基本述語の性質を表わす補題を導入し☆，公理の形で記述する（**表-8**）．これらの公理の各変数に適当な代入を行って得られる式を証明すべき式の前提条件として追加する（たとえば，表-8の公理(1)の変数 mem 1, int 1, int 2, int 3 にはそれぞれ MEM (setmax (findmax(S))), I1(S), bound(S), I2(S) を代入した）．

(4) 追加して得られた式を，setmax, findmax の公理を用いて項書換えを行う．findmax の公理

表-7 レベル2の状態 $S1$ における不変式 $R(s)$

$$\begin{aligned}&\mathrm{I1}(s)=\mathrm{I1(INIT)}\\ \wedge\ &\mathrm{I2}(s)=\mathrm{I2(INIT)}\\ \wedge\ &\mathrm{I1}(s) \leqq \mathrm{bound}(s) \leqq \mathrm{I2}(s)\\ \wedge\ &(\mathrm{bound}(s) < \mathrm{I2}(s) \rightarrow\\ &\quad \mathrm{ordered(MEM}(s), \mathrm{bounp}(s)+1, \mathrm{I2(INIT)})\\ &\quad \wedge \mathrm{isge(MEM}(s), \mathrm{I1(INIT)}, \mathrm{bound}(s)+1, \mathrm{bound}(s)+1))\end{aligned}$$

表-8 検証に用いた基本述語の性質の例

```
(* ordered な領域を1つ左へ拡大 *)
(1) : (0≦int 1<int 2+1≦int 3≦N
∧ordered(mem 1, int 2+1, int 3))
∧isge(mem 1, int 1, int 2+1, int 2+1)
→ordered(mem 1, int 2, int 3)==TRUE;
(* arrayeq なら seteq *)
(2) : (0≦int 1≦int 2≦N
∧arrayeq(mem 1, mem 2, int 1, int 2))
→seteq(mem 1, mem 2, int 1, int 2)==TRUE;
(* arrayeq ならその一部も arrayeq *)
(3) : (0≦int 1≦int 2≦int 3≦int 4≦N
∧arrayeq(mem 1, mem 2, int 1, int 4))
→arrayeq(mem 1, mem 2, int 2, int 3)==TRUE;
(* ordered かつ arrayeq なら ordered *)
(4) : (0≦int 1≦int 2≦N
∧ordered(mem 1, int 1, int 2)
∧arrayeq(mem 1, mem 2, int 1, int 2))
→ordered(mem 2, int 1, int 2)==TRUE;
(* 左右入れ換えたデータを付加しても seteq は保存 *)
(5) : (0≦int 1<int 2≦N
∧(int 1+1<int 2→seteq(mem 1, mem 2, int 1+1, int 2-1))
∧mem 1[int 1]=mem 2[int 2]
∧mem 1[int 2]=mem 2[int 1]
→seteq(mem 1, mem 2, int 1, int 2)==TRUE;
(* seteq なら isge はより大きい値に対し保存 *)
(6) : (0≦int 1≦int 2≦N
∧0≦int 3≦N
∧0≦int 4≦N
∧seteq(mem 1, mem 2, int 1, int 2)
∧isge(mem 1, int 1, int 2, int 3)
∧mem 1[int 3]≦mem 2[int 4])
→isge(mem 2, int 1, int 2, int 4)==TRUE;
```

☆ 状態遷移の対応関係は関数型の記述であっても，順次実行，条件分岐，末端再帰のみに限定すれば，始点終点がそれぞれ一つの状態遷移図に変換できる．閉路がある場合，途中状態において各成分関数間で成り立つ性質（不変式，アサーション）を検証者が考案し，(i) 始点から状態遷移を行ったとき，不変式が成立すること，(ii) 不変式を仮定し，状態遷移実行後もそこでの不変式が成立すること，(iii) 不変式を仮定し，終点で証明すべき定理の内容が真となること，(iv) 状態遷移の実行がいつかは終了すること，を示せばよい．(i)～(iii)は上述の手段(a)～(d)を用いて示す．(iv)はソフトウェアの停止性と同じ手法で行うが，個々の式が成り立つことは上述の手段(a)～(d)で証明する．

☆ これらは正しいと仮定する．

の変数 s に S を，setmax の公理の変数 s に findmax(S) を代入したものを前提条件として追加する．得られた式の恒真性判定を行う．

このような手順をすべてのパスに対して行って，sort 1（の部分正当性）が満たされることを示す．

停止性の証明は，この例では関数 $f(s)$=bound(s)-$I1(s)$ が状態遷移の繰返し実行中非負で，繰返しにより単調減少することを示せばよい．今の例では次の式が真であることを示せばよい．

$$R(s)\rightarrow(f(s)\geqq 0\wedge f(s)<f(setmax(findmax(s))))$$

公理 sort 2, sort 3 も同じように示せる．設計例のレベル 4 まで上述のような方法で検証できる．

論理設計レベルの検証も代数的手法により同じ支援系を用いて行うことができる．たとえば，表-4 の・理の＝＝を等号＝においた各式が，レベル 5 の記述の上で成り立つことは，各部品の機能や接続関係を表わす公理（表-5）及び成分や遷移の対応関係を表わす公理（表-6）による項書換えと場合分けにより示せる*．

筆者らの ASL 検証支援系を用いて行った全レベルの検証に要した計算時間などを**表-9** にまとめる**．式長は恒真性判定を行った式中の →，∧，∨，¬，≧，＝，＋，－，数値，項（bound (setmax (findmax(S))) など）などの個数である．項数は式中の異なる項の個数である．性質記述数は基本述語の性質を表わす公理の変数に代入を行って前提に追加された式の総数である．同表 (b) は，基本述語の性質（変数へ代入済）は各パスで使うものの和集合を与え，各パス共通の前提を用いた場合のデータである．

各レベルで，検証すべき公理（上位レベルの動作内容の公理）ごと検証者が不変式と共通の前提を与えただけで***，あとは支援系がパスの列挙，証明すべき式の生成などを行い，全自動で検証を行った．ASL 検証支援系では，かなり大きな式でも高速に処理できる．

表-9 検証に要した計算時間など

(a) パスごと異なる前提を用いた場合

レベル 1-2 間	CPU 時間	式長	項数	性質記述数
sort 1				
パス start-S 1	0.12	55	5	0
パス S 1-S 2-S 1	2.48	949	39	22
パス S 1-end	0.15	196	11	4
sort 2				
パス start-S 1	0.12	82	5	2
パス S 1-S 2-S 1	1.78	609	31	14
パス S 1-end	0.12	43	7	0
sort 3				
パス start-S 1	0.12	100	5	2
パス S 1-S 2-S 1	0.77	471	27	6
パス S 1-end	0.12	71	8	0
停止性	0.4	63	8	0

(b) 全てのパスに共通の前提を用いた場合

レベル 1-2 間				
sort 1	8.53	最大 1093	最大 42	25
sort 2	3.70	最大 645	最大 31	16
sort 3	2.62	最大 511	最大 29	8
レベル 2-3 間				
合計	6.80			
停止性	0.3			
レベル 3-4 間				
合計	4.48			
レベル 4-5 間				
合計	2.18			

(秒，SONY NEWS 5000)

6. おわりに

本稿で述べたように段階的詳細化の各レベルで閉じた記述を書いておけば，設計変更及び検証はしかるべきレベル以降のみ行えばよい．

フォーマルアプローチによるハードウェアの上流工程における設計及び検証は，ソフトウェアのそれに比べ，計算モデルが単純及び簡単な処理が多いなどの点から，より実用の見込みがあると思われる．筆者らの経験から，設計及び検証のための支援系を用意すれば，(一つの制御部をもつ) 同期式順序回路の設計及び検証には代数的手法は一つの有望な方法であると言える．

参考文献

1) 稲垣康善，坂部俊樹：抽象データタイプの代数的仕様記述法の基礎，情報処理，Vol. 25, No. 1, pp. 47-53 (1984). No. 5, pp. 491-501 (1984), No. 7, pp. 708-716 (1984), No. 9, pp. 971-986 (1984).
2) Joyce J. J.: Formal Verification and Implementation of a Microprocessor, VLSI Specification, Verification and Synthesis, pp. 129-157, Kluwer Academic Publisher, (1988).

* 簡単な CPU のように遷移の対応関係の動作系列に繰返しがなく，上位レベルの動作内容を簡単な演算を施した値のデータ転送に制限すれば，その検証はいわゆる記号シミュレーションによっても証明できる[2]が，代数的手法の項書換えと場合分けを用いても行える．対応に用いる動作系列に繰返しがある場合や，述語を用いて仕様を記述した場合は，記号シミュレーションでは検証できないが，代数的な手法なら可能である．

** 検証は補題の考案なども含め，大学院生森岡澄夫君が行った．なお，彼は ASL 検証支援系の実行速度及び操作性を飛躍的に向上させた．そのため，本稿の例では記述変更による全検証のやり直しが必要となった場合でも数時間の作業で行えた．

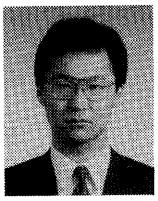
*** 不変式や基本述語の性質（補題）及び変数への代入は，一般に何回かの検証作業ののち見出される．

3) Stavridou, V., Goguen, J.A., Stevens, A., Eker, S.M., Aloneftis, S.N. and Hobley, K.M.: FUNNEL and 2OBJ: Towards as Integrated Hardware Design Environment, IFIP Conf. on Theorem Provers in Circuit Design, pp. 197-223, NorthHolland (1992).
4) 東野輝夫，関 浩之，谷口健一：代数的仕様から関数型プログラムの導出とその実行，情報処理，Vol. 29, No. 8, pp. 881-896 (1988).
5) 東野輝夫，北道淳司，谷口健一：整数上の線形制約の処理と応用，コンピュータソフトウェア，Vol. 9. No. 6, pp. 31-39 (1992).
6) 谷口健一，北道淳司：代数的手法を用いた仕様記述と設計及び検証，回路とシステム軽井沢ワークショップ論文集，pp. 375-380 (Apr. 1993).
7) 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，信学論，Vol. J77, A No. 3, pp. 420-429 (1994).
8) 森岡澄夫，北道淳司，東野輝夫，谷口健一：同期式順序回路の設計検証例，情報処理DA シンポジウム '93 論文集，pp. 73-76 (Aug. 1993).

(平成6年1月5日受付)

谷口 健一（正会員）

1942 年生．1965 年大阪大学工学部電子工学科卒業．1970 年同大博士課程修了．同年，同大基礎工学部助手．現在，同大情報工学科教授．工学博士．この間，オートマトンと言語理論，計算の複雑さ，プログラム設計開発の代数的手法及び支援システム，ハードウェアの仕様記述と詳細化およびその検証，並列分散システムの設計法などに関する研究に従事．



北道 淳司（正会員）

1965 年生．1988 年大阪大学基礎工学部情報工学科卒業．1991 年同大学院後期博士課程中退．同年，同大学情報工学科助手．ハードウェアの仕様記述と設計などに関する研究に従事．