

代数的手法を用いたマイクロプログラム制御方式順序回路の設計例

植原 孝一 北海道 淳司 東野 輝夫 谷口 健一

大阪大学 基礎工学部 情報工学科

あらし

従来、我々は代数的手法を用いた同期式順序回路の記述法及び設計法を提案し、その評価を行っている。本報告では、この設計法を一部拡張し、要求仕様からマイクロプログラム制御方式順序回路を設計し、またマイクロプログラムを生成するための設計法を提案する。この設計法によって正しさの保証された回路及びマイクロプログラムが得られる。この設計法を、クイックソートを行う同期式順序回路に適用し、要求仕様から段階的に詳細化して回路及びマイクロプログラムを得た。その回路は、人手で書き下した（LSI設計の例題として日本電子工業振興協会がまとめたもの）ものと同一で、マイクロプログラムもサイズ及びソートに要する総ステップ数において同程度のものである。

A Design of a Quick Sort Circuit with Microprogrammed-Controller
Using Algebraic Methods

Kouichi NARAHARA Junji KITAMICHI Teruo HIGASHINO Kenichi TANIGUCHI

Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University
Toyonaka-shi, 560 Japan

Abstract

We have proposed an algebraic method for top-down design of synchronous sequential logic circuits, and are investigating its usefulness. In this paper, we extend this method for designing of circuits with microprogrammed-controller. As a design example, we explain a refinement process for a quick sort circuit. The derived circuit and microprogram satisfy requirements and run efficiently.

1. まえがき

マイクロプログラム制御方式順序回路の設計において重要なことは、回路とマイクロプログラムの正しさ（回路がマイクロプログラムのもとで動作すれば、要求どおりに動作するか）を保証できること、マイクロプログラムの実行効率が悪くないものが得られること、部品やバス構成やマイクロコードの意味を設計者が自由に設計できること等が挙げられる。従来の研究では、一動作の内容がレジスタ転送レベルの動作系列の記述を入力としマイクロプログラムを生成するシステム等に関するものが多い^[6]が、その入力となる動作系列自体が回路の要求仕様を満たしているかどうかや、生成された回路やマイクロプログラムが正しいかどうかの議論は行われていない。

我々は代数的言語ASLを用いた同期式順序回路の仕様記述法及び（抽象レベルで記述された）要求仕様からの段階的設計法を提案し、その評価を行っている^{[1][2]}。この設計法は、設計者が設計すべき箇所（解法（アルゴリズム）や用いる部品、バス構成など）は自由に設計し、その設計の正しさをASL検証支援系を用いて証明し、また機械的に設計が行える箇所や最適化などは支援系を用いて機械的に行うというものである。従って、得られる回路の正しさを保証し、設計の自由度の高い設計法である。この設計法ではこれらの手順を繰り返すことにより、最終的に結線制御方式の回路が得られる。

この設計法は、(a)部品の機能やバス構造などを意識せずに設計を行う機能設計レベルのものと、(b)部品の機能や入力論理式を設計するレベルのものに分けられるが、本報告では、機能設計レベルにおける設計法は(a)のままで、(b)をマイクロプログラム制御方式のものが得られるよう次のように変更する。(a)を用いて各状態遷移を一マイクロ命令で実行できるレベルに詳細化された回路に対して、(b)では次の手順で設計を行う。用いる部品の機能や入出力の接続関係、マイクロ命令の各フィールドの意味などを設計者が設計し、それらを用いて上位レベルのデータ転送や条件分岐が行えるようなマイクロコードを決定する。そのコードが正しいかどうかを証明し、それらのマイクロコードにアドレスを割り付けてマイクロプログラムを導出する。

また、我々の提案する設計法は完全なトップダウンであるが、効率の良いマイクロプログラムを得るためには設計により生じる無駄を取り除くこと（いわゆる最適化^[7]、状態遷移図の簡単化と呼ぶ）が不可欠である。必要と思われる種類の簡単化に関してルール化を行った。ルールの適用の自動化や支援についても検討を行っており、一部実現を行った。

この設計法をクイックソートを行う回路に適用した。回路の要求仕様から段階的に詳細化し、ほぼ同じアーキテクチャ（構成部品やバス構成）を用い、人手で書き下したマイクロプログラム^[4]と、サイズ及びソートに要す

る総ステップ数において同程度のものを得た。

以下、本報告では、2. で機能設計レベルの設計手順(a)について、3. でこの設計手順を例題に適用し一動作が一マイクロ命令に相当する回路を得るまでの設計について、4. でそこからマイクロプログラムを得る設計手順(b)及び例題への適用について述べる。

2. 代数的手法を用いた順序回路の記述法及び機能設計の概略

同期式順序回路の仕様は、動作内容の定義Dと動作の実行制御の記述Cからなる。まず、回路の動作を状態遷移（“状態遷移関数”と呼ぶ）で表し、初期状態と状態遷移からなる項(表現式)を“抽象状態”と呼ぶ。抽象状態Sでの各部品の値Fはその値のデータ型を値域とする関数F(S)で記憶されと考え、その関数を“状態成分関数”と呼ぶ。回路の動作内容の定義Dは、各状態遷移を行うと状態成分の値がどのように変化するかを公理の形で記述する。動作の実行制御Cでは、有限制御部の値（以下、状態名と呼ぶ）がどのような値で、どのような分岐条件が成り立つときにどの状態遷移を行うか（状態遷移の実行条件）、またそのとき状態名はどのような値になるかを記述する。これらは、状態名に着目すると図1のような状態遷移図で表される。但し、状態遷移図を代数的言語ASLで記述するため、2つの状態成分関数CONTROL、VALIDを導入する。CONTROLは状態名（状態遷移図の節点のラベル）を保持し、状態遷移後のCONTROLの値がどのような値になるか（枝の行き先）を記述する。VALIDは状態遷移の実行条件を記述する^[1]。

上位レベルの仕様 $t = \langle D, C \rangle$ が与えられた時、設計者は、(1)下位レベルで用いる状態成分、状態遷移を決定し、それらを用いてD'を定義する。
(2)上位レベルの状態遷移と下位レベルの状態遷移系列との対応関係(必要なら上位レベルの状態成分と下位レベルの状態成分との対応関係も加えた)Mを考案し、
(3)Mが正しいか(Mで指定された順にD'を実行すればDの要求を満たすか)をASL検証支援系を用いて証明する。証明に失敗した時は、D'やMの修正を行い、成功した時は、
(4)状態遷移図CとMから、下位レベルの状態遷移図C'を合成し、下位レベルの仕様 $t' = \langle D', C' \rangle$ を得る（自動合成のためのプログラムが作成されている）。
(5)必要ならば、C'を簡単化する。得られた状態遷移図をC''とし、このレベルの回路を、 $t'' = \langle D', C'' \rangle$ とする（これらの手順の実例を、3. で示す）。

要求仕様から上述の手順を繰り返す、レジスタ転送レベルの記述を得る。

3. 例題の段階的詳細化の概略

用いる例題の要求仕様及び途中レベル(レベル4)までの詳細化は文献[2]のものと同じで、詳細は省略する。

要求仕様を記述するレベル（レベル1）では、一状態

遷移でソートを実行する状態遷移SORT, 状態成分としてソートされる要素を格納する配列Eを導入し, 回路の仕様記述を行う. レベル2では, クイックソートアルゴリズムを採用することを決定し, 文献[4]の分割操作に相当する状態遷移splitなど, 分割範囲を格納しておくスタックST, スタックポインタpを状態成分関数として導入する. レベル3では, 状態成分として基準値を格納しておくレジスタXや基準値より大きな(または小さな)要素を探すための配列Eへのポインタi, jを, 状態遷移として基準値より大きな(または小さな)要素を見つけるmove_jやmove_iを導入し, 分割操作をより具体的な状態遷移で対応させる. レベル4へは, そのmove_jやmove_iを, 配列の要素を一つずつ調べるようにより細かな状態遷移の繰り返しに対応させる. レベル1からレベル4までの各レベルの状態遷移図は図1のようになる(但し, 状態名, 分岐条件等は省略されている).

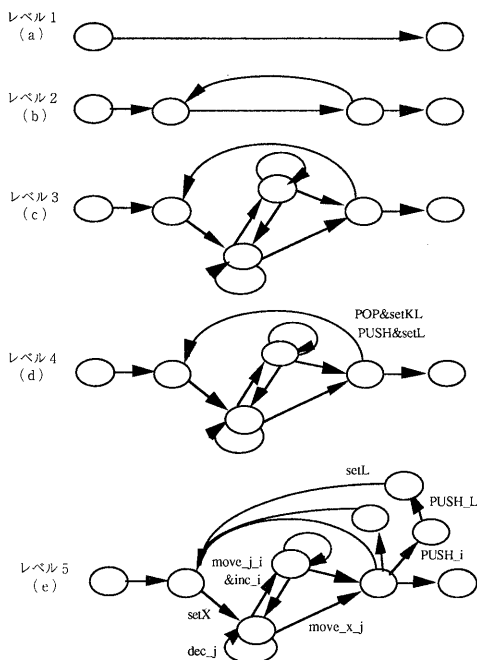


図1 レベル1からレベル5までの状態遷移図

マイクロプログラム制御部を用いるため, レベル4から6への設計は, 以下のように行った.

[例3.1] レベル4からレベル5への設計

レベル4までは, 分割範囲を記憶しておくスタックSTは, 一度のスタック操作で分割範囲を指定するための上限下限のポインタを一度にプッシュあるいはポップすることが出来るものを用いている(例えばポインタが16bitで, スタックSTは32bit幅のメモリを用いる). レベル5ではスタックに, ポインタと同じデータタイプのメモリを用いることにし, 一回のスタック操作で1ポインタのプ

ッシュ及びポップが出来るスタックst, スタックポインタp2を導入する. 従って, レベル4でのスタックSTに関する2つの動作を細分化する(図1(d)から(e)). このレベルまでは状態遷移図の簡単化を行う箇所はなかった.

[例3.2] レベル5からレベル6への設計

マイクロプログラム制御方式ではレジスタなどは一つのレジスタファイルにまとめることによってハードウェアの量を減らし, 且つ融通性を高めることが多い. この例題でもポインタなどはレジスタファイルRFにまとめ, ソートされるべき配列E及びスタックstを, メモリRAMにまとめる. レジスタA, レジスタBを導入し, レジスタファイルの各要素間でデータ転送や演算を行なう時は, レジスタAまたはレジスタBを媒介して転送する. またレジスタAはメモリRAMのアドレスレジスタとして, レジスタBはメモリバッファレジスタとして利用する(図2).

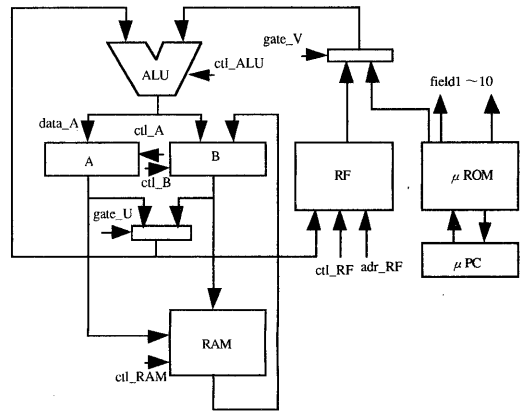


図2 採用するアーキテクチャ

各状態遷移は, 採用するアーキテクチャ上で, 一度に(1クロックで)実行できるものを用いる(実行できるかどうかは4.4(1)の過程で調べる). 例えば, レジスタAとレジスタファイルの要素間でデータ転送する状態遷移 $A \leftarrow L, L \leftarrow A$ は導入するが, レジスタファイルの要素間でデータ転送する状態遷移は導入しない.

次にレベル6において導入した状態遷移の動作内容の定義を決定する(表1). 例えば, 状態遷移 $A \leftarrow L$ に関する動作内容の定義は, 状態遷移 $A \leftarrow L$ を行うとレジスタAの内容はレジスタファイルRFのadr_L番目(レジスタL)の内容がAに転送されることを公理を用いて記述する(ここで用いられている状態遷移名 $A \leftarrow L$ などは便宜上用いる. 状態遷移名自身が動作内容を定義しないことに注意). また $A \leftarrow A+1 \& E \leftarrow B$ のように, Aの値は1増やされ, 同時にメモリRAMにデータレジスタBの値を書き込むといったパラレルな動作は, 各成分毎にそのデータ転送の内容を記述する. 表1では, データ転送が起こらない成分に関しては公理を省略している(例えば $A \leftarrow L$ では, A以外の成分は不変なので, $B(A \leftarrow L(s)) = B(s)$ 等は省略).

次に、レベル5の状態遷移とレベル6の状態遷移系列の対応Mを考案する(表2)。例えば、レベル5の状態成分jの値を1減らす状態遷移dec_jを、レベル6のA←j, A←A-I, j←Aの順に状態遷移を実行する系列に対応させている。これらの対応でレベル4の各状態遷移で定義されているデータ転送が正しく行われるかを証明し、レベル5の状態遷移図とこれらの対応から、レベル6の状態遷移図を合成し(図3)、レベル6の回路を得る。

$A(A \leftarrow \beta(s)) = \text{get}(\text{RF}(s), \text{adr}_\beta)$ β は K, L, i, j, X, p2 など
 $\text{RF}(\beta \leftarrow A(s)) = \text{put}(\text{RF}(s), \text{adr}_\beta, A(s))$
 $B(B \leftarrow E(s)) = \text{get}(\text{RAM}(s), A(s))$
 $\text{RAM}(E \leftarrow B(s)) = \text{put}(\text{RAM}(s), A(s), B(s))$
 $\text{RAM}(A \leftarrow A+1 \& E \leftarrow B(s)) = \text{put}(\text{RAM}(s), A(s), B(s))$
 $A(A \leftarrow A+1 \& E \leftarrow B(s)) = A(s)+1$
 $A(A \leftarrow A+1(s)) = A(s)+1$

表1 レベル6の動作内容の定義

$\text{setX}(s) = X \leftarrow B(B \leftarrow E(A \leftarrow i(s)))$
 $\text{dec}_j(s) = j \leftarrow A(A \leftarrow A-I(A \leftarrow j(s)))$
 $\text{move_j_i \& inc_i}(s) = i \leftarrow A(A \leftarrow A+1 \& E \leftarrow B(A \leftarrow j(s)))$
 $\text{move_x_i}(s) = E \leftarrow B(B \leftarrow X(A \leftarrow i(s)))$

表2 レベル5とレベル6の対応関係

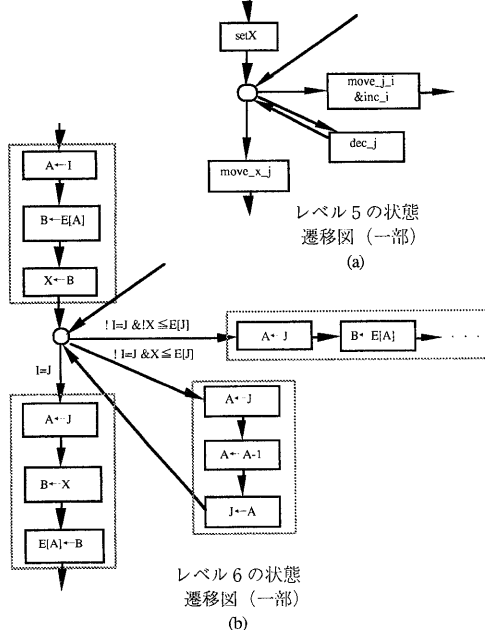


図3 レベル5とレベル6の状態遷移図

3.1 状態遷移図の簡単化

我々の提案する設計法は完全なトップダウン方式のために、得られた状態遷移図中には冗長な状態遷移が生じる場合がある。状態遷移図の簡単化を行うことによって、これらの無駄な状態遷移を取り除き、効率のよい回路を

導くことができる。ここでは、必要な簡単化操作をルール化し、その適用結果について述べる。2. で述べた設計法は、具体化された回路の正しさを保証しているが、状態遷移図の簡単化ルールの適用においても、回路の正しさは保証されなければならないが、ここで用いるルールは全て一般的に用いられているルールであり、ルールの正しさに関する議論は省略する。

[前提] このレベルの動作内容の定義を行う公理は全て、次の形の公理からなる (tjに関する公理は複数個ある。以降このような公理を $\text{exp} \rightarrow \text{Fi}$ と略記)。

$\text{Fi}(\text{tj}(s)) = \text{exp}(s)$;

expはFk(s)(kは自身でもよい)や基本関数や定数からなる式である。 $\text{exp} \rightarrow \text{Fi}$ に対して、tjでFiに代入が行われた、expに用いられているFkに対して、Fkが参照されたという。特に $\text{Fi} \rightarrow \text{Fi}$ の時やFiに関する公理がないときは、代入参照が行われたとはいわない。

・ 節点vに入射する各枝ei (eiの始点をvi, 分岐条件をcondiとする) において、(1)~(3)の何れかが成り立つとき、節点vでは成分FiとFjの値が等しい。このとき、節点vにおいて関係 $\text{Fi} = \text{Fj}$ が成り立つという。

(1)eiにおいて、 $\text{Fi} \rightarrow \text{Fj}$ が行われる。

(2)condiが真ならば論理式 $\text{Fk} = \text{Fl}$ (kはi,lはjでもよい) が真となり、eiにおいて $\text{Fk} \rightarrow \text{Fi}$ および $\text{Fl} \rightarrow \text{Fj}$ が行われる。

(3)viで関係 $\text{Fk} = \text{Fl}$ (kはi,lはjでもよい) が成り立ち、eiにおいて $\text{Fk} \rightarrow \text{Fi}$ および $\text{Fl} \rightarrow \text{Fj}$ が行われる。

・ 節点vから始まる任意のパスにおいて、状態成分Fiに対して代入を行う枝eに到達する前に、分岐条件やデータ転送でFiが参照されるようなパスがあるとき、状態成分Fiを節点vではlive、それ以外の成分はdeadと呼ぶ。

次に、用いるルールを示す。

[ルール1]

始点がv1, 終点がv2, 分岐条件がcondであるような枝eがあり、v2に入射する枝がeのみであり、eで行われる各 $\text{Fp} \rightarrow \text{Fq}$ に対し次の条件の何れかが成り立つとき、

(条件1) v1において関係 $\text{Fp} = \text{Fq}$ が成り立つ、

(条件2) condが成り立つなら論理式 $\text{Fp} = \text{Fq}$ が真となる。

(条件3) v2において Fq がdeadである。

(操作) eを取り除き、v2を始点とする各枝e1,...,ekをv1から出射するようにし、実行条件を、condと各枝e1,...,ekのもとの実行条件との論理積とする。

[ルール2]

次の2つの条件が共に成り立つとき、

(条件) 分岐点vから出射する枝の中に同じラベルTの枝e1,...,ekがある。

(条件) 枝e1,...,ekの分岐条件に用いられている状態成分Fiは枝e1,...,ekで代入されない、または、各枝の分岐条件cond1,...,condkに対し、T実行後の成分値を用いて等価な分岐条件cond1',...,condk'が書けること(例えば,condiが $\text{Fp} > 0$, Tで $\text{Fp}+1 \rightarrow \text{Fp}$ が実行される時, cond1'は $\text{Fp}-1 > 0$)。

(操作) 枝 $e_1, \dots, e_k$ を取り除き, ラベルTの枝 e (始点は v とし, 終点を ve)を追加する. e の分岐条件は $cond_1, \dots, cond_k$ の論理和とする. 枝 $e_i (1 \leq i \leq k)$ の終点から出射する各枝 e_{ij} (分岐条件を c_{ij} とする)を e の終点 ve から出るようにし, e_{ij} の分岐条件を $cond_i' + c_{ij}$ との論理積とする.

[ルール 3] (ルール 2 の逆の操作)

v を始点とする枝 e (分岐条件 $cond$, 状態遷移T, 終点 ve)と, 互いに排他的な m 個の条件 $cond_1, \dots, cond_m$ に対して,
(操作) e を取り除き, m 個の節点 $v_1, \dots, v_m$ を作り, v からそれらの節点にそれぞれ枝 $e_1, \dots, e_m$ (分岐条件はそれぞれ $cond$ と条件 $cond_1, \dots, cond_m$ との論理積, 状態遷移T)を作る. 各節点 $v_i (1 \leq i \leq m)$ から, ve から出射する枝 $e_1', \dots, e_k'$ と (分岐条件, 状態遷移, 終点)が同じ枝 k 本をそれぞれ出射させる. このとき, 節点 v や $v_1, \dots, v_m$ から出射する枝の分岐条件が真となる動作列が存在しえないならばそれらの枝は取り除く (図4では v_2 から u_1 , v_1 から u_2 への枝がとり除かれている).

[ルール 4]

次の条件を共に満たすとき

(条件 1) 節点 v に入射する枝の中に同じラベルTの枝 $e_1, \dots, e_k$ があり,

(条件 2) 枝 $e_1, \dots, e_k$ の始点では分岐が起こらない.

(操作) 枝 $e_1, \dots, e_k$ を取り除き, 終点が v でラベルTの枝 e を追加する. 枝 $e_1, \dots, e_k$ の始点に入射していた各枝 e_{ij} を e の始点 u に入射させる.

[ルール 5]

(操作) 複数の枝の始点となっている v に対して, ルール 4 と逆の操作を行う.

[ルール 6]

(条件) 節点 v において, 関係 $F_i = F_j$ が成り立つならば,

(操作) v から出射する枝 e の分岐条件や, v から出射する枝 e (のT) で実行される $exp \rightarrow F_k$ の exp で用いられている状態成分 F_i を F_j に (または F_j を F_i に) 置換する. この置換で, 状態遷移図中の他の枝のTと異なる状態遷移となるので, 状態遷移名Tを変更する.

[ルール 7]

単純ループ (ループから脱出, ループに合流する点は1つのみ, それを v とする) において, 次の3つの条件が成り立つとき,

(条件) ループ内の各枝 $e_1, \dots, e_n$ や v での分岐条件において参照されない成分 F_j があり,

(条件) 枝 $e_1, \dots, e_n$ に, F_j への代入 $F_k \rightarrow F_j$ のみを行う枝 e_j が存在し,

(条件) v において F_j がdeadまたは, v において関係 $F_j = F_k$ が成り立つならば,

(操作) v において F_j がdeadならば e_j を取り除く, そうでなければ, e_j を取り除き, ループから脱出する側の v の分岐先に e_j を挿入する.

[ルール 8] (条件 3 が成り立つ時のルール 1 の逆の操

作)

(条件) v において, F_i がdeadの時,

(操作) 枝 e (始点 vs , 終点 ve , 分岐条件は恒真, 状態遷移は $exp \rightarrow F_i$)を作り, v に入射する枝を vs に入射させ, v から出射する枝を ve から出射させる.

[ルール 9]

(条件) v から出射する枝が2本 (e_1, e_2 , その終点を v_1, v_2 , 状態遷移を T_1, T_2 とする) あり, その分岐条件が $a \& b$, $\text{not } (a \& b)$ の時,

(操作) 枝 e_1, e_2 を取り除き, 節点 vm と枝 e_1', e_2', e_3', e_4' を作る. e_1' の始点を v , 分岐条件を a , 終点を vm , 状態遷移をNOPとする. e_2' の始点を v , 分岐条件を $\text{not } a$, 終点を v_2 , 状態遷移を T_2 とする. e_3' の始点を vm , 分岐条件を b , 終点を v_1 , 状態遷移を T_1 とする. e_4' の始点を vm , 分岐条件を $\text{not } b$, 終点を v_2 , 状態遷移を T_2 とする.

[例 3. 3] ルールの適用例

上述のルールをレベル6の状態遷移図に適用した結果, ルールを25回適用し, レベル6の状態遷移図における節点の数は53から47に減らすことが出来た.

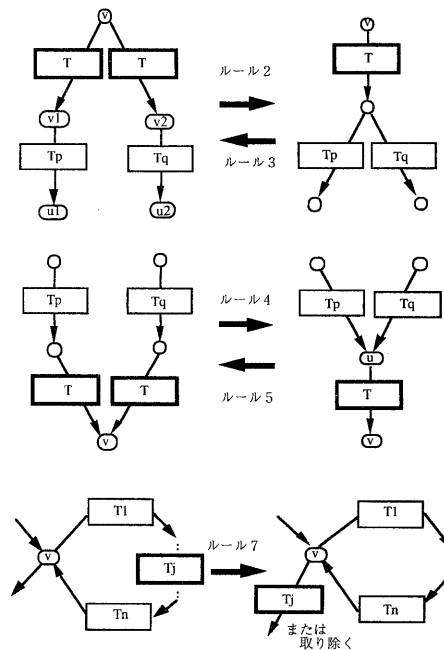


図4 ルールの例

4. マイクロプログラムの導出

3. で得られたレベル6の回路(以下, 回路 α と呼ぶ)から, 各状態成分に対応するモジュール, 各モジュールの入出力及びバスの接続関係を決定し (この手順は従来の結線制御方式のものと同じである. 以降, 従来の方法を旧方法, 今回提案する方法を新方法と呼ぶ), 最終的にマイクロプログラムを導出する設計手順(得られる回

路を回路 β と呼ぶ)について説明する。

4.1 回路を構成するモジュールの決定

回路 α の各状態成分に対応するモジュールの機能を定義する。旧方法ではCONTROLに対応するモジュールは自動合成されるが、新方法では、状態成分CONTROLに対応するモジュールをマイクロプログラムのアドレスを指定するモジュール(以下 μ PCと呼ぶ)に対応付ける。ここでは、値を1増やすマイクロコード中のアドレスフィールドの値をロードする μ PCを前提とする。

[例4.1]モジュールの例

回路 β での状態遷移を各記憶・演算モジュールごとに考える。例えば、回路 α の状態成分Aに対応する回路 β のモジュールAでのローカルな状態遷移(以下"部分状態遷移"と呼ぶ)を $ck_A(S_A, load_A, data_A)$ で表し($load_A$ はモジュールAの制御入力, $data_A$ はデータ入力信号を表す), $init_A$ をモジュールAの初期状態とすると, モジュールAの抽象状態(以下"部分抽象状態"と呼ぶ)は $init_A$ と $ck_A(S_A, load_A, data_A)$ からなる項(表現式)によって表される。また, モジュールAを関数 $A(S_A)$ のように部分抽象状態を引数とする関数で表す。回路 β のモジュールAや μ PCは次のような公理で定義される機能を持つモジュールを使用する。

```
A(ck_A(S_A, load_A, data_A))
  = if load_A = 00 then A(S_A)
    else if load_A = 01 then A(S_A) + 1
    else if load_A = 10 then A(S_A) - 1
    else if load_A = 11 then data_A
    else A(S_A);

 $\mu$ PC(init) == 0
 $\mu$ PC(CK(S $_{\mu$ PC}, ctl1 $_{\mu$ PC}, ctl2 $_{\mu$ PC}, adr $_{\mu$ PC}, data $_{\mu$ PC))
  = if ctl1 $_{\mu$ PC} = 0 and (
    ctl2 $_{\mu$ PC} = 001 or
    ctl2 $_{\mu$ PC} = 010 and data $_{\mu$ PC = 0 or
    ctl2 $_{\mu$ PC} = 011 and data $_{\mu$ PC != 0 or
    ctl2 $_{\mu$ PC} = 100 and data $_{\mu$ PC > 0 or
    ctl2 $_{\mu$ PC} = 101 and data $_{\mu$ PC <= 0 or
    ctl2 $_{\mu$ PC} = 110 and data $_{\mu$ PC < 0 or
    ctl2 $_{\mu$ PC} = 111 and data $_{\mu$ PC >= 0) then adr $_{\mu$ PC
    else  $\mu$ PC(S $_{\mu$ PC}) + 1
```

表3 用いるモジュールの例

4.2 モジュール間の接続関係、マイクロ命令の意味の決定

次に、4.1で決定したモジュールの入出力の接続関係を決定する。

[例4.2] バス構成、接続関係の定義

次に4.1で決定したモジュールの入出力関係及びバスの接続関係を記述する。本報告では、文献[4]とほぼ同じ図2のバス構成及びモジュールの接続関係を採用することにする。一部、制御部の構成が異なる。文献では、命令レジスタMIRを用い、マイクロ命令実行をパイプライン化するようにしているが、ここでは、簡単のため、マ

イクロROMの出力を各部品制御入力に接続するアーキテクチャを採用している。2本のバスU,Vの構成を決定する(図2)。例えば、バスUはバスのゲート信号 ctl_U が値0のとき、モジュールAの値、1のときモジュールBの値が出力させる(表4)。

また、回路全体での状態遷移 $CK(S, micro-code)$ を導入する。状態遷移 CK は、回路全体の抽象状態 S と、状態 S において実行されるべきマイクロコード(一命令) $micro-code$ を引数とする。各モジュールに1対1に対応する部分状態遷移を、表4のように回路全体での状態遷移 CK を各状態成分の部分状態遷移を並行して実行することに対応付ける。各モジュールの入力信号をどのバスやモジュールに接続するかを各部分状態遷移のデータ入力に相当する引数に接続されているモジュール、バスを与えることにより指定する。

```
bus_U(S, gate) = if gate = 0 then A(proj_1(S))
                  else if gate = 1 then B(proj_2(S));

define ALU = ALU(field_4(micro-code), bus_U, bus_V);
define busU = bus_U(S, field_3(micro-code));
define busV = bus_V(S, field_5(micro-code));
field1(micro-code) = [get(micro-code, 0), get(micro-code, 1)];
:
CK(S, micro-code)
  = | ck_A(proj_1(S), field_1(micro-code), ALU),
    ck_B(proj_2(S), field_2(micro-code),
      RAM_out(RAM(proj_4(S)), A(proj_1(S))), ALU),
    ck_RF(proj_3(S), field_4(micro-code),
      B(proj_2(S)), field_6(micro-code)),
    ck_RAM(proj_4(S), B(proj_2(S)), field_9(micro-code),
      A(proj_1(S))),
    ck_ $\mu$ PC(proj_5(S), field_5(micro-code),
      field_7(micro-code), field_8(micro-code), ALU) |

VALID(CK(S, micro-code)) ==
VALID(CK(S)) and micro-code = get( $\mu$ ROM,  $\mu$ PC(proj_5(S)));
```

表4 (図2に対応する) 各モジュールの入出力の接続関係

ここで $proj_k$ は並び $[...]$ の k 番目の要素を取り出す射影関数である(並びを作る関数と射影は基本関数と考えている)。 $proj_k(S)$ は抽象状態 S での k 番目のモジュールの部分抽象状態を表す関数である。例えば、 CK_A の第2引数 $load_A$ として $field_1(micro-code)$ 、第3引数 $data_A$ として $ALU(...)$ が用いられる。これは、モジュールAの制御入力 $load_A$ にはマイクロ命令の1番目のフィールドが接続され、データ入力 $data_A$ にはALUの出力が接続されることを表している。

マイクロ命令の(各フィールドの)意味は、例えば、マイクロ命令 $micro-code$ の1番目のフィールドを表す $field_1(micro-code)$ は状態成分Aの制御入力 $data_A$ に接続され、モジュールAの機能から、 $field_1(micro-code)$ が値10のときはAの値は1増加し、 $field_1(micro-code)$ が値01のときはAの値は1減少することなどの様に、表4の接続関係と表3の各モジュールの定義によって定義される。

CKの引数micro-codeの値は、VALID関数を用いて指定され、ここでは定数 μ ROMの μ PC番目が与えられると記述している。以下では、実行すべきマイクロコードは μ ROMの μ PC番目のものであることを前提とする。

各モジュールの定義とモジュール間及び各フィールドとの接続関係から、各モジュールの制御入力、データ入力、バス構成及びマイクロ命令の意味が決定されている。以下、 μ ROMを導出する方法について述べる。

4.3 状態遷移図の変更

マイクロプログラム制御方式で制御部を実現する場合は、4.1や4.2で決定された μ PCの機能や接続関係により、一マイクロコードで実現できる分岐数や分岐条件は制約される。そのような場合、各分岐が一マイクロコードで実現できるように分岐部を展開する（実現できるかどうかは4.4(1)の過程で調べる）。分岐部の展開は、3.1で述べた状態遷移図の簡単化ルールを用いる。

[例4.3]分岐部の展開

例題では、 μ PCが、分岐を行なうマイクロ命令の分岐先アドレスを一つしか指定できないので（表3）、状態遷移図の分岐数は最大二分岐になる。このために回路 α の状態遷移図で三分岐以上の分岐を行なっているノードは二分岐を組み合わせて実現する。ここでは、図3(b)の簡単化後のものを例に説明する。

状態 a における分岐部を、条件 $I=A$ による分岐部と条件 $X \leq E[A]$ による分岐部からなるグラフに展開する（ルール9、図5(a)から(b))。次に、条件 $X \leq E[A]$ の判定は、採用したアーキテクチャでは、メモリEの値はデータバッファBに一旦格納してからでないとALUを用いて判定できないので、Eの値を（deadである）Bに転送する状態遷移($B \leftarrow E[A]$)を追加し、分岐条件も変更する（ルール8、1、6。図5(b)から(c)）。

さらに、状態 a から出射する二枝が一マイクロ命令で実現できるように（この場合 $B \leftarrow E[A]$ と $J \leftarrow A$ は同時に実

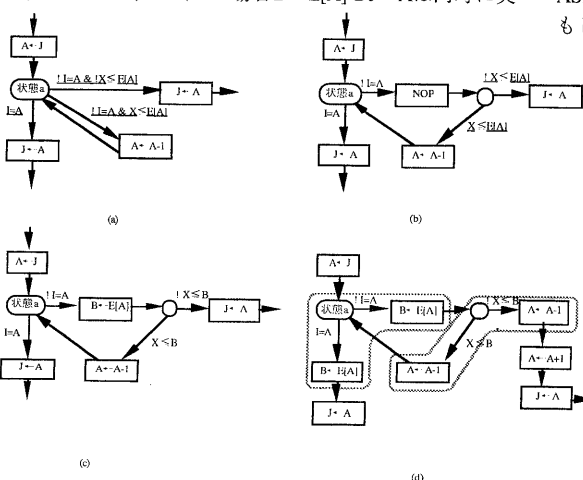


図5 分岐部の変更例

行できない）、分岐部を变形する（ルール8。図5(c)から(d)、状態 a の下方でBがdeadである）。

このようにして、状態 a におけるループ部分は、文献[4]のものと同様にループが2マイクロコードで実現できるものが得られた。また、全体の状態遷移図に関しても、条件部の変更と簡単化を行った結果、ルールを27回適用し、状態数が49のものが得られた。

4.4 マイクロコードの割り当て及びマイクロプログラムの作成

最後に、4.1、4.2で与えられた回路（表3と表4）と μ ROMの内容で、回路 α の状態遷移図が指定する順番どりに、指定されるデータ転送が行われるように、 μ ROMの内容を決定する。以下に述べる手順は部品の機能やマイクロ命令のフィールドの意味を制限した場合、コード割り付けを機械的に行うことができ、それらを実行するシステムに関する研究も行われており、それらの成果を利用することもできる^{[9][6]}。

一般には、一つの枝を複数のコードに対応付けたり、部分グラフを一つのコードに対応付けて、マイクロプログラムを得ることもできるが、ここでは、各節点に対し節点から出射する枝のデータ転送や分岐を、一マイクロコードの実行に対応付ける。

(1) 各節点とマイクロコードの対応付け

回路 α の各節点に対して、節点から出射する枝（状態遷移）で行われるデータ転送や分岐を、一マイクロコードの実行に対応付け（各フィールドの内容を決定し、但し、ジャンプ先アドレスフィールドは(2)で決定する）、その内容が正しいことを証明する。この証明は、決定したデータ転送用フィールドの内容と4.1.4.2で決定したモジュールの機能及び結線関係で、回路 α の各状態遷移で要求されているデータ転送（表1）や条件分岐（4.3で得られた状態遷移図）が行われるかどうか調べるもので、ASL検証支援系を用いて機械的に判定することができる。

もし、マイクロコードの割り付けや証明ができない場合は、4.3に戻って状態遷移図を展開し直すか、4.1.4.2に戻ってアーキテクチャの再設計を行う。

例えば、分岐の無い節点ならば、データ転送用フィールドの内容を、出射する一本の枝のデータ転送を実現するように考案し、それが正しいかを証明する。制御用フィールドは μ PCを+1させる内容か無条件ジャンプをさせる内容とする（何れを選択するかは(2)で決定する）。分岐のある節点ならば、出射する2本の枝のデータ転送を実現するようにデータ転送用フィールドの内容を考案し、それが正しいかを証明する。制御用フィールドは、分岐を正しく行わせる内容とする（ジャンプ先アドレスフィールドは手順(2)で決定される）。 μ PCの機能により制御用フィールドの内容として複数の候補がある場合（例えば、 $I=A$ の判定をするの

に、 $I=A$ による分岐と、 $\text{not } I=A$ による分岐ができるような μPC の場合)、何れを選択するかは(2)で決定する。

ここで、状態遷移図中の同名の状態遷移に対しては、データ転送用フィールドは同じ内容となる。

(2) アドレスの割付けとアドレスフィールドの決定

手順(1)で各節点に対応付けられたコードにアドレスを割り付け、未決定であったジャンプ先アドレスフィールドの内容を決定し、 μROM を完成する。

(a)開始点(例えば初期状態)のアドレスを0とし、この箇所から節点に数値(アドレス)を割り付ける。

(b)アドレス(n)を割り付けた節点において、

(b-1)分岐がない場合、

(b-1-1)次の節点のアドレスが未決定のとき、制御用フィールドの値が $\mu\text{PC}+1$ させる内容とし、次の節点のアドレスを $n+1$ とし、その節点から(b)を繰り返す。

(b-1-2)次の節点のアドレスが既に決っている(m)とき、制御用フィールドの値をmに無条件ジャンプにする。

(b-2)分岐している場合、

(b-2-1)2つの分岐先の節点のアドレスのうち片方が既に決っている(m)場合、決っている方にジャンプするように制御用フィールドの候補1つを選び、そのmをアドレスフィールドの内容とする。そのような候補が無い場合は(b-2-2)と同じ操作を行う。

(b-2-2)2つの分岐先の節点のアドレスが両方とも既に決っている(l,m)場合、制御用フィールドの候補から適当な1つを選び、分岐条件が満たされる方の節点のアドレス(l)をアドレスフィールドの内容とする。分岐条件が満たされない方の節点に対しては、新たに節点を作り、そのアドレスを $n+1$ とし、mに無条件ジャンプするようなコードを対応付ける。

(b-2-3)2つの分岐先の節点のアドレスが両方とも未決定の場合、制御用フィールドの候補から適当な1つを選び、分岐条件が満たされる方の節点のアドレスは(c)で決める。もう一方の節点のアドレスを $n+1$ とする。

(c)未決定アドレスフィールドを含むコードが対応付けられている節点の中から1つを選んで、その分岐先の節点のアドレス及びアドレスフィールドに、割付けられていない数値(nまで割り付けていれば $n+1$)を割付け、その分岐先の節点から手順(b)を繰り返す。

(d)もし、新たに割り付けるアドレスnがアドレスフィールド幅mに対して、 $n \geq 2^m$ のように、フィールド内におさまらない場合は、アドレスフィールド幅を増やすか、モジュールとその接続関係などの再設計を行う。

以上の手順により、節点に割り付けられたアドレスとマイクロコードから μROM が完成し、回路 α を実現する回路 β (表3と表4と μROM)が得られた。

5. 設計法の評価

例題では、マイクロプログラムは25bit幅で、サイズが49のものが得られた。文献[4]の人手により書き下したマ

イクロプログラムのうち、ソートを行う箇所のサイズは66である。実行ステップ数は、提案した設計法によって得られたものの方が若干多い。このように、要求仕様からの完全なトップダウンによる設計法であるにもかかわらず、効率の良いものが得られている。

このようにある程度の規模で効率のよい回路を得るには(マイクロプログラム制御方式の回路にかかわらず、結線制御方式の回路の場合でも)、状態遷移図の単純化を行うことが不可欠と思われる。現在、状態遷移図の単純化に関して、どのような支援が有効かについて検討を行っている。例えば、計52回ルールを適用したうち、27回ルール1を適用した。使用頻度の高いこのルールに着目し、ある箇所からルール1を適用する箇所を調べ、ルールを適用するルーチンを作成した。(1)適当な箇所(例えば初期状態)から調べ始めて、ルールを適用できる最初の1つを見つけ、適用するに7.3秒程度、(2)設計者が適用できそうな箇所を指定した場合、3.8秒程度要した。

6. あとがき

本報告で提案した設計法を用いることによって、例題に対して、人手で書き下したマイクロプログラムと同程度の効率のよい、正しさの保証された(但し、実際には検証は一部しか行っていない)回路が得られた。また、トップダウンによる設計でも、状態遷移図の単純化などの手法を用いれば、効率のよいマイクロプログラムを得られることがわかった。現在、状態遷移図の単純化は、設計者が、どのルールをどの箇所でもどのように(例えば、ルール6でどの分岐条件のどの箇所を書き換えるかなど)適用するかを指定しなければならないが、その指定や適用できるかどうかの判定を支援(あるいは自動化)することは、設計効率の向上につながると思われる。今後、これらの支援系を作成し、設計の具体例をとおり、本設計法や支援系の有効性についてさらに調べたい。

文献

- [1]杉山裕二, 北道淳司, 谷口健一: “代数的手法を用いた順序回路の記述とその詳細化について”, 電子情報通信学会技術研究報告COMP88-7, pp.61-70(1988-05).
- [2]北道淳司, 杉山裕二, 谷口健一: “クイックソートICの代数的記述と制御回路の自動生成について”, 情報処理学会研究報告DT46-13, pp.95-102(1989-02).
- [3]北道淳司, 谷口健一: “代数的手法を用いたマイクロプログラム制御方式順序回路の階層的設計”, 情報処理学会第42回(平成3年前期)全国大会講演論文集, pp.6-170-171(1991-03).
- [4](社)日本電子工業振興協会編: “LSI設計用記述言語の標準化に関する調査研究”, (昭63-03).
- [5]大田他: “複雑な実行順序制御方式のマイクロプログラムのためのマイクロアセンブラ”, 情報処理学会第38回(平元前)全国大会講演論文集, pp.3-1546-1547(1989-03).
- [6]池永他: “高位の仕様記述に基づく専用プロセッサ設計支援システム”, 情報処理学会第39回(平元後)全国大会講演論文集, pp.3-1862-1863(1989-10).
- [7]J.Bhasker: “An Optimizer For Hardware Synthesis”, IEEE Design&Test of Computers, Vol.7, No.5, pp.20-36(1990-10).

盛光印刷所