代数的手法を用いた
同期式順序回路の段階的設計および形式的検証

北道　淳司

平成１０年８月

代数的手法を用いた
同期式順序回路の段階的設計および形式的検証

北道　淳司

平成１０年８月

内容梗概

本論文は，筆者が大阪大学大学院基礎工学研究科に在学中から同大学大学院にて助手として引き続き携わった，ディジタル回路の設計の上流工程における設計及び形式的検証の方法に関する研究をまとめたものである．

近年，ハードウエアの設計規模の増大に伴い，論理設計レベルより設計抽象度の高い，いわゆる設計の上流工程におけるハードウエア仕様記述及びそれからの自動設計あるいは設計支援に関する研究が注目されている．また，設計により得られた回路の正しさを保証するための検証期間の増大に対して，効率的な形式的検証法が望まれている．

本研究の目的は，要求される処理全体を一状態遷移で表すといったような回路の要求仕様を記述する抽象度の高いレベルから具体的な論理設計レベルに至るまでの回路設計の上流工程において，代数的手法を用いた同期式順序回路の設計法及び設計の正しさの形式的検証法を提案し，その有用性を確認することである．

本研究では，具体的に設計された回路が要求仕様を満たすことを保証するために，設計は要求仕様から段階的にかつ各段階においては回路を局所的に具体化することにし，各段階における局所的な具体化が正しいことを形式的に証明するという方法を採用する．さらに各段階の設計法，設計の正しさの形式的な定義及び検証法について統一的な方法を採用することにする．これらの方針のもとに，回路モデルとしてどのようなものが適当か，回路の局所的な具体化としてどのような内容であれば自然な設計が行えるか，具体化の正しさの形式的な検証は自動的に行えるかまたは検証者による考案が必要であるならば容易に作業が行えるか，局所的な具体化のみからなる設計に対してどのような修正・改善手法を設計法に取り入れれば回路設計に長じた設計者が具体レベルで創意工夫を凝らして設計したものに比べてどのくらいの性能のものが得られるか等を調べるために以下を行った．

まず，要求仕様を記述するレベルから論理設計レベルまでの任意の設計レベルにおける同期式順序回路の形式的仕様記述スタイルとして拡張有限状態機械モデルを用い，これを代数的言語 ASL で記述する．拡張有限状態機械モデルは，回路の制御部に相当する有限状態機械及び状態成分と呼ばれるレジスタ群からなる．状態遷移には状態成分の値などを用いた実行

条件及び状態遷移時に行われる動作内容が付加されている．動作内容は，状態遷移前後の状態における各状態成分の値の間で成り立つべき関係式の形，あるいは各状態成分の値をもとに演算を実行した結果を並列に更新する代入式の形で与えられる．設計者は，各状態成分のデータタイプとして論理型，整数型以外にも抽象スタックなどを表す抽象データタイプを用い，これらのデータタイプ上の関数を用いて回路の仕様記述を行うことができる．また，意味定義が厳密でかつ簡単な代数的言語を用いることにより，２つの回路の等価性の形式的定義，段階的設計及び形式的検証における機械的処理が行える．このモデルのもとで，上位及び下位の回路及び２つの回路の状態及び成分に対する対応が与えられたとき，設計の正しさは，上位及び下位の回路が初期状態から動作するとき，対応する状態における対応する成分の値の系列が等しいこととして定義される．

次に，要求仕様から具体的な論理設計レベルまでの段階的な設計法を提案した．各段階の設計は以下の手順で行われる．与えられた上位の回路における各状態遷移に対し，それぞれ独立して動作内容や実行条件の判定を実現するために，必要ならば新たに状態成分を追加して動作内容や実行条件の判定を実現する拡張有限状態機械を設計する．この拡張有限状態機械に用いられている各状態遷移の動作内容や実行条件の判定はより具体化されたものが用いられる．次に上位の回路の制御部と具体化された拡張有限状態機械の制御部分から下位の回路全体の制御部を合成し，得られた制御部を簡約化する．この手順を繰り返し，具体レベルでは，部品の機能，バス構成，制御部の構成（ワイヤード論理制御方式あるいはマイクロプログラム制御方式など）などからなる回路を設計し，制御フリップフロップの入力論理式やマイクロプログラムによる制御部を合成する．このような各段階において回路を局所的に具体化するという制約に基づいて設計された回路は，具体レベルで直接設計されたものに比べて，データ転送の並列度が低いなどの点で各資源の有効な利用が行えないという可能性がある．そこで，提案する設計法には，回路に含まれる冗長な動作などを取り除いたり，２つの動作を並列な一動作に取りまとめるなどの簡約化のための手順を取り入れている．これらの簡約化や実行条件の設計は回路の制御構造の一部を変更する12種類の簡約ルールを用いて行う．これらの簡約ルールは，あらかじめ適用前後における回路の等価性を保証しており，各

ルールの適用条件の判定及びルールの実行などを支援系を用いて行う．この方法を用いて，クイックソート法及びマックスソート法によるソート回路及びいくつかの形式的検証のためのベンチマーク回路（簡単な CPU，GCD 回路など）を設計した．いずれも回路に要求される動作を一状態遷移で表すレベルから段階的に設計を行ったにもかかわらず，制御部の最適化作業を行うことによって人手で直接具体レベルにおいて設計して得られたものと同じ回路を得ることができた．

設計の正しさの証明は，上位の回路が初期状態から実行する遷移数の帰納法を用いることによって実際に可能となる．この帰納法の帰納段階は，上述の設計手順において設計者によって考案された具体的な状態遷移からなる拡張有限状態機械が対応する上位の回路の状態遷移の動作内容を正しく実現していることを示すことである．この証明は，具体的な状態遷移からなる拡張有限状態機械の制御部上の，例えば繰り返し制御の終了条件を判定する状態のような特定の状態において各状態成分の値の間に成り立つべき不変表明を検証者が考案し，それらを用いて構造的帰納法により証明する．この証明は筆者の属する研究グループが開発した拡張有限状態機械型プログラムの設計検証のための代数的検証支援系を用いて行うことができる．検証支援系は，不変表明を用いた帰納法の証明手順の管理やユーザが定義した関数や述語に関する補題の管理の支援を行い，証明すべき論理式を合成し，その式の真偽を判定する．不変表明や補題の考案など試行錯誤を必要とするが，本支援系によって検証者はそれらの作業のみを行えばそのほかの検証作業はすべて検証支援系が一定の手順で項書き換え等の代数的証明機能を適用し，自動的に行うことができる．上述の設計例題に対しては，CPU の検証は全て自動的に，GCD 回路及びソート回路の検証は段階的設計の最初の数レベルの検証は不変表明や関数や述語に関する補題が必要であり人手による試行錯誤が必要であったが，それ以降は自動で行えた．

さらに代数的証明機能の一つである真偽判定手続きの高速化を目指した新しいデータ構造を提案し，その上での処理方法を実現した．新たに作成した真偽判定系は整数上の加減算，不等号，等号，限定子及び論理演算を含む論理式を処理することができる．提案する判定系では，処理中の式全体を不等号などからなる整数上の論理式を表す構造及び論理部分を表す

BDDs（二分決定グラフ）の構造との組み合わせによって保持している．判定系は，保持している全ての部分グラフの各頂点をテーブル上に管理することによって共通部分グラフを共有し，その上での高速な処理を実現している．提案する手法との比較のために用いた森岡の判定系はいくつかの高速化手法が実現されているが，データの共有化の手法は取り入れていない．森岡の判定系との比較のために行った評価実験によって同程度の判定時間で論理式の判定が行えることがわかった．

これらの設計及び検証実験により，設計においては局所的に具体化し，その具体化を形式的に証明する，全レベルで統一された記述法，設計法及び検証法を採用するといったような制約があるのもかかわらず，実際に要求仕様を記述するレベルから論理設計レベルまで段階的に，支援系を用いることによって各段階の設計の正しさを形式的に検証しながら回路設計を行うことができることがわかり，提案する回路設計法，形式的検証法及び新たに提案した判定系の有用性が確認された．

関連発表論文

[1]谷口健一，北道淳司，東野輝夫：“整数上の線形制約の処理と応用”，コンピュータソフトウエア，Vol.9，No.6，pp. 31-39(1992-06).

[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，Vol.J77-A，No.3，pp.420-429(1994-03).

[3]谷口健一，北道淳司：“代数的手法による仕様記述と設計及び検証”，情報処理，Vol.35，No.8，pp.742-750(1994-08).

[4]J.Kitamichi，S.Morioka，T.Higasino and K.Taniguchi：“Automatic Correctness Proof of Implementation of Synchronous Sequential Circuits Using Algebraic Approach”，Proceedings of the 1994 Conference on Theorem Provers in Circuit Design (TPCD94)，Vol.901 of LNCS，pp.165-184，Springer Verlag (1995-09).

[5]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，pp.281-286(1995-04).

[6]J. Kitamichi，N. Funabiki and S. Nishikawa：“Proposal of Data Structure for Presburger Arithmetic and its Application to Circuits Verification”，Proceedings of 1997 Int.Symp. on Nonlinear Theory and its Applications(NOLTA97)，Vol.2，pp.1233-1236(1997-11).

[7]景山洋行，北道淳司，船曳信生：“整数上のある制約論理式の処理のためのBDDの拡張とそれを用いた回路検証”，DAシンポジウム'98論文集，pp.89-94(1998-07).

目次

1　緒論

1.1　本研究の背景と概要

近年，ハードウエアの設計規模の増大に伴い，回路設計の上流工程における回路仕様記述法及びそれからの自動設計法あるいは設計の支援に関する研究が注目されている．現在，提案されている回路仕様記述言語[1]として，VHDL (VHSIC Hardware Description Language) [2], Verilog HDL，SFL (Structured Function description Language) [3]などがあげられる．これらは従来の論理設計レベルの記述言語であるネットリストなどに比べて，具体的な論理を実現する部品やその間の配線などを考慮しないなどの意味において，記述レベルとしてより抽象度が高く，回路の機能や階層的な構成なども記述することができる．また，このような言語を用いて記述された回路仕様は，資源の割り付け，動作手順のスケジューリング，状態割り付け，論理の最適化などを施され，さらにレイアウト設計などより下位の設計自動化ツールを経て，実際の回路が得られる[4]．これらの設計の上流工程のことは方式設計 (System Design) レベル，機能設計 (Functional Design) レベルと呼ばれる．

また，設計すべき回路規模の増大につれ，設計により得られた回路に対する検証期間の増大は深刻な問題になりつつある．特に回路仕様記述言語からの設計の支援あるいは自動化が進むにつれ，設計の上流工程における検証期間は設計工程全体のかなりの時間を占めつつある．一般に検証手法としては，初期状態及び入力系列を回路に与えて出力結果を観察するというテストによる非形式的手法と，形式的回路モデル及び数学的な証明手法を用いた形式的手法によるものとに大別される．従来論理設計レベルにおいては，論理式をコンパクトに表現し，その表現に対する高速な処理が行えるBDDs（Binary Decision Diagrams，二分決定グラフ）[5]を用いた形式的手法が有望とされており，論理設計及び論理検証の分野で実用システムに組み込まれつつあるが，大規模な回路あるいはより抽象レベルにおける回路の検証には限界があると思われる．設計の上流工程における検証では，取り扱うデータタイプとして論理型よりも抽象レベルの高い整数型あるいは抽象データタイプなどを扱うことのできる記述言語及びそれに対する検証手法が望まれる．記述能力の高い高階論理[6]や時相論理[38]を用いた回路仕様記述及び形式的検証に関する研究などが行われているが，これらの言語の意味

定義や検証の枠組みは複雑であり，自動検証や高速な処理という立場から，設計の上流工程における回路の検証にはより適した手法が望まれる．

筆者の属する研究グループでは，従来よりプログラムの設計の正しさを形式的に検証するための方法を提案し，その方法に基づく検証支援系を試作し，提案手法の評価のために実用プログラムに対する検証実験を行ってきた[25, 43, 44]．その手法とは，代数的言語[44]を用いてシステムの要求仕様記述からプログラムまでを記述し，項書き換え，証明すべき式の恒真性判定の決定手続きへの帰着などの代数的定理証明の方法を用いてプログラムの正当性を検証する[25]というものである．代数的言語は，その意味定義が厳密であり，高階論理や時相論理よりも言語の枠組みが簡明であるにもかかわらず，抽象データタイプを記述者が自由に定義できるなど回路仕様記述言語としての能力は十分であると考えられる．代数的言語ASL[44]に対する検証手法も，項書き換えや決定手続きへの帰着など高速な検証技法の組み合わせによって検証を行うことができ，設計の上流工程における形式的検証手法としては有望であると考えられる．

本研究の目的は，要求される処理全体を一状態遷移で表すといったような回路の要求仕様を記述する抽象度の高いレベルから具体的な論理設計レベルに至るまでの上流工程における回路設計に対して，代数的手法を用いた同期式順序回路の設計法及び設計の正しさの形式的検証法を提案し，その有用性を確認することである．

本研究では，具体的に設計された回路が要求仕様を満たすことを保証するために，設計は要求仕様から段階的にかつ各段階においては回路を局所的に具体化することにし，各段階における局所的な具体化が正しいことを形式的に証明するという方法を採用する．さらに各段階の設計法，設計の正しさの形式的な定義及び検証法について統一的な方法を採用することにする．これらの方針のもとに，回路モデルとしてどのようなものが適当か，回路の局所的な具体化としてどのような内容であれば自然な設計が行えるか，具体化の正しさの形式的な検証は自動的に行えるかまたは検証者による考案が必要であるならば容易に作業が行えるか，局所的な具体化のみからなる設計に対してどのような修正・改善手法を設計法に取り入れれば回路設計に長じた設計者が具体レベルで創意工夫を凝らして設計したものに比べてどのく

らいの性能のものが得られるか等を調べるために以下を行った.

要求仕様を記述するレベルから論理設計レベルまでの任意の設計レベルにおける回路の仕様記述，各レベルの設計の正しさの定義及びその形式的検証のために，同期式順序回路の形式的仕様記述スタイルを定める．記述スタイルとして拡張有限状態機械モデル[43]を用い，これを代数的言語 ASL で記述する．拡張有限状態機械モデルは，回路の制御部に相当する有限状態機械及び状態成分と呼ばれるレジスタ群からなる．状態遷移には状態成分の値などを用いた実行条件及び状態遷移実行時に行われる動作内容が付加されている．動作内容とは，抽象度の高いレベルでは，状態遷移前後の各レジスタ値の間で成り立つべき関係であり，具体的なレベルでは，各レジスタの値をもとに演算を実行した結果を並列に更新する代入式である．設計者は，各レジスタのデータタイプとして論理型，整数型以外にも抽象データタイプを用い，実行条件及び動作内容としてこれらのデータタイプ上の関数を用いて回路の仕様記述を行うことができる．要求仕様を記述するレベルから，制御部をマイクロプログラムやワイヤードロジックで実現し，データパスを具体的に何 bit のレジスタやどのような算術論理演算が行える ALU を用いるかといった具体的な部品で構成されるレベルまで，このスタイルを用いて記述することができる．このことにより各レベルにおいて同一の枠組みの設計法，設計の正しさの定義及びその形式的検証法を適用することができる．また，意味定義が厳密でかつ簡単な代数的言語を用いることにより，2つの回路の等価性の形式的定義を行うことができ，設計及び形式的検証において機械的な処理が行える．

このような様々な設計レベルにおいては，それぞれのレベルに対応する回路モデルあるいは言語を用意するという立場もある．しかし，そのような場合，抽象的な要求仕様を記述するレベルから具体的な論理設計レベルまで，それぞれのレベルにおいて異なる設計法を用いざるをえず，かつ設計の正しさを議論する場合各レベルで用いる言語の意味も含めて議論せざるをえない．本論文では段階的な設計において常に同じ設計方法を繰り返し用い，その設計の正しさを任意のレベルで同じ枠組みで議論し，同じ検証支援システムで検証を行うことを可能にするため，各レベルにおいて統一の回路モデル及び記述スタイルを採用することにする．

次に，要求仕様から具体的な論理設計レベルまでの段階的な設計法を提案した．抽象度の高いレベルでは，与えられた上位の回路における各状態遷移で指定されている状態遷移前後の各レジスタ値の間で成り立つべき関係を実現するため，必要ならば新たに状態成分を追加し，各状態遷移それぞれに対して拡張有限状態機械（以下では，上位の回路の状態遷移に対応する状態遷移系列，あるいは遷移の対応と呼ぶ）を設計する．この拡張有限状態機械に用いられている各状態遷移の動作内容や実行条件の判定は，より具体化されたものが用いられる．遷移の対応による動作内容が上位の回路の動作内容を満たすことを証明し，次に上位の回路の制御部とすべての遷移の対応から下位の回路の全体の制御部を合成し，最後に得られた制御部を最適化する．この手順を繰り返し，具体的なレベルでは以下の手順で設計が行われる．与えられた上位の回路に対して，まず設計者が定めた部品の機能，バス構成，制御部の構成（ワイヤード論理制御方式あるいはマイクロプログラム制御方式など）などを設計し，上位の回路の各状態遷移を用いるアーキテクチャにおいてどの転送路を用いてどの部品にデータを転送するか，それらのステップをどの順に実行させるかという部分的な制御を設計する．次に用いるアーキテクチャ上で指定された通りにデータ転送を行えば上位の回路の動作内容を正しく実現することを証明する．さらに，各実行条件の判定を用いるアーキテクチャ上で実現できるように動作内容の場合と同様に設計する．最後に，上位の回路の制御部と動作内容の設計より得られた遷移の対応より回路全体の制御を求め，制御フリップフロップの入力論理式やマイクロプログラム[13]を合成する．

本論文において提案する回路の段階的設計の内容は，既発表の文献[7, 8, 9, 11, 12, 13, 14]に基づいている．

本研究で採用した設計方法では，各レベルにおける設計において，上位の回路の状態遷移の動作内容を実現する遷移の対応や実行条件の判定の実現をいかに工夫して設計したとしても，冗長な状態遷移や有限状態が含まれてしまう可能性がある．例えば，A，B，Cを成分として，状態遷移Tの実行条件がA=Bであり，動作内容がBとCの加算の結果のAへの転送（B+C→Aと表す）であるとする．下位での実現では，これらの成分を全てレジスタファイルRFに格納するもの（例えば成分Aに対応するRFの要素をRF[A]と表す）とし，実行条

件の判定は比較器 EQ を用いて汎用レジスタ REG と RF のある要素との比較のみ実行でき，加算は演算器 ALU を用いてレジスタ REG と RF のある要素との演算を実行し，その結果は汎用レジスタに格納されるというようなアーキテクチャを採用するものとする．このアーキテクチャのもとでは，上位の回路の転送 B+C→A は，転送 RF[B]→REG，転送 RF[C]+REG →REG 及び REG→RF[A] の順に実行するような遷移の対応として設計される．また，実行条件の判定 A=B は，転送 RF[B]→REG 及び判定 EQ(REG, RF[A]) の順に実行するように設計される．もとの状態遷移 T の実行を制御する部分は，実行条件の判定を行う制御の後，動作内容を実現する遷移系列を制御するように合成されたとすると，動作内容を実現する遷移系列を実行する状態（状態 ST1 と呼ぶ）では，REG にはすでに RF[B] の値が格納されているので，転送 RF[B]→REG の遷移は冗長である．そこで，この遷移を取り除き，回路の実行制御の高速化を行う．

しかし，与えられた回路に対して設計者がそれぞれの回路の各有限状態毎に対応をつけられないくらい自由に複雑な簡約作業を行った場合，簡約化前と簡約化後の回路との等価性（簡約化の正しさ）の証明は，一般に困難である．そこで，本手法では，そのような簡約化を行う場合どのような簡約化が行われるかを分類し，１２個の簡約化ルールを定め，設計者はそれらのルールのみを用いて回路の簡約化を行うというという方法を用いる．まず，それぞれのルール毎に回路上でどのような条件が成り立てばルールが適用できるかということをまとめ，これらの簡約化作業を支援するシステムを作成した[14]．上述の例を用いて簡約作業の様子を説明する．状態 ST1 において実行される転送 RF[B]→REG を除去することを考える．状態 ST1 で簡約ルールの適用条件が成り立つかどうかを判定する．この場合は，簡約ルールの適用条件である，状態ST1 において RF[B] と REG の値が常に等しいかどうかということを支援系が判定する．この判定は，支援系が指定された状態 ST1 から遷移を逆向きに指定された有限回探索し，探索範囲内の有限状態内から指定された状態に到達したときに必ずレジスタ間にそのような関係が成り立つかどうかを調べる．例の場合，２遷移逆向きに探索すると，転送 RF[B]→REG を実行させる状態に到達し，この状態を経由して状態 ST1 に到達したときに常に RF[B] と REG が必ず等しくなり，ST1に到達するには必ずこの状態を経由しな

ければならないことを調べる．そのことが成り立つことを判定した後，支援系は対応する簡約ルールを実行させる．設計者は，この簡約機能を用いるときは，GUI を用いてどの有限状態においてどのルールを適用するかを指定すれば，支援系は上述のように適用条件が成り立つかどうかを自動的に調べ，簡約ルールを適用する．

この方法を用いて，クイックソート法とマックスソート法によるソート回路及びいくつか公開されているベンチマーク回路（簡単な CPU，GCD 回路）を設計した[7, 8, 9, 12, 13, 14]．それらのうち CPU の設計は数日，クイックソート回路の設計はいくつかのレベルにおいて行った制御部の簡約化作業も含めて数週間で行うことができた．これらの設計では，段階的な手法にもかかわらず，いずれの場合も，人手で直接論理設計レベルにおいて創意工夫を行い実現したのと同じ回路[26]を得ることができた．

設計の正しさは，上位及び下位の回路が初期状態から動作するとき，対応する状態における各成分の値の系列が等しい，すなわち出力履歴が等価であることとして定義される．この証明は，上位の回路が初期状態から実行する遷移数の帰納法を用いることによって実際に可能となる．この帰納法の帰納段階は，上述の設計手順において設計者が考案した遷移の対応どうりに下位の回路が状態遷移を実行したときに上位の回路の動作内容を正しく実現していることを示すことである．この証明は，遷移の対応における例えば繰り返し制御の終了条件を判定する状態のような特定の状態において，各レジスタの値の間に成り立つべき不変表明やユーザが定義した関数や述語に関する補題を検証者が考案し，それらを用いて構造的帰納法により証明する．この証明は，筆者の属する研究グループが開発した拡張有限状態機械型プログラムの設計検証のための検証支援系[10]を用いて行うことができる．検証支援系は，不変表明を用いた帰納法の証明手順の管理や用いる関数や述語に関する補題の管理の支援を行い，証明すべき論理式を合成しその式の真偽を判定する．不変表明や補題の考案など試行錯誤を必要とするが，本支援系を用いることによって，検証者はそれらの作業のみを行えばそのほかの検証作業はすべて機械的に行うことができる．上述の設計例題に対する検証を検証支援系を用いて行った．CPU の検証は数秒で完全に自動的に，GCD 回路及びソート回路の検証は段階的設計の最初の数レベルの検証は不変表明や関数や述語に関する補題が必要で

あり人手による試行錯誤が必要であったが，それ以降は自動で行えた[15, 16].

本論文において提案する回路の形式的検証の内容は，既発表の文献[7, 8, 10, 15, 16]に基づいている

また，筆者はさらにこの真偽判定手続きの高速化を目指した新しいデータ構造を提案し，その上での処理方法を実現した[17,45]．新たに作成した真偽判定系は整数上の加減算，不等号，等号，限定子及び論理演算を含む論理式を処理することができる．提案する判定系では，処理中の式全体を，不等号などからなる整数上の論理式を表す構造及び論理部分を表すBDDsの構造との組み合わせによって保持している．判定系は，保持している全ての部分グラフの各頂点をテーブル上に管理することによって共通部分グラフを共有し，その上での高速な処理を実現している．提案する手法との比較のために用いた森岡の判定系[46]は，提案する手法で用いられている上述の共通部分式の共有という手法は取り入れられていないが，いくつかの高速化のための工夫が行われている．森岡の判定系との比較のために行った評価実験によって同程度の判定時間で論理式の判定が行えることがわかった[45].

これらの設計及び検証実験により，実際に要求仕様から論理設計レベルまで段階的に，支援系を用いることによってその正しさを保証しながら回路設計を効率よく行うことができることがわかり，提案する回路設計法，形式的検証法及び新たに提案した判定系の有用性が確認された.

以下，２章では，本論文で対象とする回路モデルを定め，そのモデルの代数的な記述法を定義し，その回路モデルに対する段階的な設計法について述べる．３章では，提案する方法に基づく設計支援システムの実現の仕方とそれを用いたクイックソート法によるソート回路の記述とその具体的な設計手順について述べる．４章では，設計の正しさを形式的に検証するために用いられる検証技法とそれらをどの順に適用するかについて述べ，それらの適用順を実現した検証支援システムを用いておこなったCPU，GCD回路などの検証実験の結果について述べる．５章では，提案する検証法において用いられるプレスブルガー文の真偽判定のための新たなデータ構造の提案，それを用いた判定系の実現及び，判定系の評価のための実験について述べ，６章で，本研究に対する評価を行う．

1.2 ディジタル回路の設計の上流工程における形式的検証について

ディジタル回路の形式的手法による回路検証及びそれらの中での本研究の位置づけについてまとめる．

回路検証は，どのような回路モデルを採用するのか，何を検証するのか，どのような検証手法を用いて調べるかという点で，いくつかの分類が行われる．

まず，対象となる回路の設計レベルに着目して回路モデルの分類を取り上げる．回路の設計レベルとしては，設計の抽象レベルの順に方式・機能・論理・レイアウトの順に設計が行われると考えられる．方式設計とは設計する回路のアーキテクチャを決めるレベルである．例えば，プロセッサの設計では，命令セットやパイプラインの方式（何段のパイプラインにするか，制御方式はどうするかなど）の設計が行われるレベルである．機能設計とは，用いる部品の機能（ALU ならば演算機能は四則演算のみか浮動小数点演算を採用するか，入出力を何ビットにするかなど）を決定し，各部品の制御機能も設計するレベルである．論理設計レベルでは，上位の回路で決定されたデータパス及び制御部の各部品（制御部の状態割り付けなども行われる）を実現する論理を決定し論理の最適化が行われるレベルである．レイアウト設計のレベルでは，設計された論理に対して，配置，配線が行われるレベルである．

近年では，方式設計あるいは方式設計レベル以上のレベル（例えばプロセッサの設計では，プロセッサに対するプログラムの設計も含むようなレベル）あるいは，ハードウエア及びソフトウエアの協調設計（Software Hardware co-design）を行うようなレベルも考えられる．本研究で対象とするレベルは，これらのレベルも含む要求仕様を記述するような抽象度の高いレベルから機能設計を終えるようなレベルまでを対象としている．これらのレベルでは，レジスタの bit 幅を抽象化することによってレジスタを整数として取り扱うというように，用いるデータタイプが論理設計レベルよりも抽象度の高いものを取り扱うことができる．また，現在のところ，このような抽象度の高いレベルでは，配線や各ゲートでの伝播遅延あるいは回路の面積及び消費電力等も省略されるのが一般的である．

本研究で採用する拡張有限状態機械モデル及び代数的記述法では，もちろん論理設計レベルの回路モデル及び記述を行うことができ，設計法及び検証法も採用することができるが，

論理設計レベル以降に関しては現在確立されている方法を用いればよいという立場をとる．例えば，記述及び設計については具体的な代数的記述からSFLのような回路記述言語に変換し合成系パルテノンを用いてより下位の設計を行えばよいし，それらの設計レベルにおいて検証が必要になるのであれば，例えばBDDsを用いた論理検証法を用いて検証を行えばよい．

また，対象となる回路の種類によってもいくつかの分類が行われる．組み合わせ回路と順序回路，特に順序回路に関しては，さらに同期式と非同期式の回路に分類される．組み合わせ回路は，内部に状態を持たない（内部のプリミティブな論理素子間のデータフローグラフにおいて閉路がない）回路である．組み合わせ回路は，設計の上位においては入出力間のみの関係あるいは入力を引数として出力を返す関数として取り扱われ，論理型以外に整数や抽象データタイプ上の関数として取り扱われる場合もある．下位においては，基本論理ゲートとして扱われ，さらに下位ではプリミティブな論理素子の遅延や配線遅延を含むようなモデルとして取り扱われる．

同期式順序回路は，状態を保持するレジスタと次状態及び出力を与える組み合わせ回路からなるモデルとして扱われる場合が多い．論理設計レベルでは有限状態機械モデル，より抽象度の高いレベルでは，抽象データタイプを保持できる抽象レジスタを成分に持つ拡張有限状態機械（Extended Finite State Machine）モデルやプロセス代数モデルなども用いられることがある．

例えば，VHDLでは，設計の上位での記述はbehaviorの記述と呼ばれ処理の順番を記述し，具体レベルではネットリストを記述することもできる．SFLでは，機能設計レベルから論理設計レベルの同期式順序回路を記述対象と考え，合成系パルテノンを用いて中間言語に変換し，論理圧縮などを経てEDIFネットリストを合成する．UDL/I [54]は，同期式回路から非同期式回路までを記述することができ，設計レベルとしては機能設計及び論理設計レベルを対象としている．UDL/Iにおいて着目すべきは，言語の設計時において言語の形式的意味定義が行われている点（厳密には，UDL/Iにおける動作を記述する部分を対応する核言語に置き換えた上での意味定義が行われている）であるが，代数的言語に比べて意味定義が間接的であり複雑である．

また，形式的仕様記述法としては，高階論理[29]，時相論理[38]，一階論理[28]などが用いられている．また，記述言語としては，回路記述のためにプログラミング言語Cに文法的な制約を加えた言語で記述するという方法や，プロトコル仕様記述言語 LOTOS を用いて記述するという方法も用いられている．

本研究では同期式順序回路を対象とし，設計対象とする任意のレベルにおいて拡張有限状態機械モデルを採用することにする．拡張有限状態機械モデルは，代数的言語 ASL を用いて記述される．代数的言語 ASL では，いわゆる代入式（ある状態における各部品の値をもとに演算を実行し，その結果を指定された部品の次状態における値とする）だけではなく，動作の前後における各部品の間に成り立つべき関係を定めることにより仕様を記述することもできる．また，記述者がデータタイプ及びその上での関数や述語を自由に定義できそれらを用いて仕様記述を行うことができ，対象とする回路の上流工程での仕様記述に適していると考えられる．また，代数的な定理証明手法を用いて組み合わせ回路の機能の検証を行うこともできる[16, 17].

検証の内容については，以下のように分類される場合がある．ここでは，上位の仕様を要求仕様，下位の仕様を実現仕様と呼ぶことにする．文献[53]から引用すると

1）回路の要求仕様が十分であるか.

2）回路の実現仕様は，要求仕様を満足するか.

3）回路の実現仕様は，いくつかの重要な性質を満たすか.

1)は，requirements captureと呼ばれる．2)は，implementation verification（実現検証）あるいは実現仕様と要求仕様の等価性の検証，3)は，design verification（設計検証）と呼ばれる．いくつかの文献では，2）のことを設計検証という文献もあり，本論文においても，2章以降においては，取り扱う検証問題である2）を”設計の正しさの検証”あるいは”設計検証”と呼ぶので注意されたい．3）において重要な性質とは，ある特定の状態に到達しない（例えば，交差点の信号機制御回路において，いかなる情況であろうと交差する両方の信号がともに青を出力する状態に到達しないという性質），デッドロックなどが起きない，などのこ

とである．

論理設計レベルでの２つの順序回路の等価性は，２つの回路が同じ初期状態から動作を開始し，同じ入力系列を読み込む場合，対応する各成分の値が等しいことと定義される[56]．本研究では，要求仕様での状態遷移と状態成分，及び実現仕様での状態遷移と状態成分のレベルが異なる場合も対象とする．従って，実現仕様の状態には，要求仕様の状態と対応する状態と，要求仕様の状態遷移の途中状態に相当するので要求仕様の状態には対応しない状態とが存在すると考える．また，要求仕様の 32bi tのレジスタが，実現仕様では 16bit の２つのレジスタで実現されるというような場合も考慮することにする．本研究での実現の定義では，実現仕様の状態に対して要求仕様の状態に対応する状態及び対応しない状態を区別する状態に対する述語 IS（Interested State）を導入し，実現仕様では述語 IS が真となるような状態での出力に着目することし，要求仕様の各状態と実現仕様の着目する状態において，成分の対応のもとで各成分の値が等しいとき，要求仕様と実現仕様は等価であると定義する．このように状態あるいは動作という概念において抽象度の異なるレベルでの等価性を議論するときは，本研究のように実現仕様の状態に関する情報を与えるという方法[49]や，実現仕様の状態から要求仕様の状態への対応を与える関数（abstraction function）を利用する方法[55]等が用いられる．

提案する手法では，上述の回路の等価性を代数的枠組みを用いて証明するために，上位の各状態遷移に対して下位の具体的な遷移の系列を対応させ，その対応によって上位の状態遷移が定めている動作内容を下位の回路の遷移の系列が正しく実現していることを証明し，その系列の実行を終了した状態を着目する状態とすれば，初期状態からの帰納法により等価性を証明したことになる．また，さらに下位の遷移系列において繰り返し構造がある場合は，不変式（Invariants）を用いた構造的帰納法を用い，証明すべき問題を，複数の”ある関係が成り立つ状態から動作を開始して，有限回状態遷移を実行した状態において証明すべき関係が成り立つかどうか”という十分条件の判定問題に帰着させ，その判定問題をその他の代数的な定理証明機能で解くという方法を用いている．

設計検証は，検証すべき性質として”要求仕様と等価であるか”という検証を考えた場合，実現検証を含んでいると考えることができる．論理設計レベルにおける設計検証に用いられる主要な形式的手法は，時相論理の一種である計算木論理（CTL）[56]があげられる．与えられた順序回路に対して成り立つべき性質をCTL式で記述し，その性質が回路上で成り立つかどうかを，回路の状態と入力の組を節点とするようなKripke構造，及びCTL式を構成する演算それぞれに対して定義されるKripke構造上のノード集合に対する操作を用いて検証する．モデル検査（Model Cheking）や状態数え上げ（State Enumerate）等の検証アルゴリズムには，これらの操作を用いる．Kripe構造上の状態集合の表現を簡潔に及びそれへの操作を高速に行うために，論理設計レベルのこれらの検証では一般にBDDsが用いられる．

その他に，上述の論理設計レベルでの形式的手法であるモデル検査や状態数え上げを，レジスタに整数値，演算に加減算等を用いることができるようなレベルでの検証に応用した研究[35,48]も行われている．

定理証明系を用いた検証は，検証すべき性質をその形式的記述法によって項（論理式）で表し，回路記述などを公理とした時にそれらの上でその項が真であること（定理として成り立つこと）を証明することである．代数的手法では，公理$\alpha == \beta$を左辺から右辺への書き換え規則と見なした項書き換え，場合分け，不変式を用いた構造的帰納法，Presburger文真偽判定手続きへの帰着などの方法を用いて証明する．

その他に，存在限定子を施された変数のSkolem関数への置き換え，全称限定子で束縛されている変数の定数への置き換え（instantiation）などの基本的な推論規則を検証系に実現している例もある[49].

強力な推論規則を用意したとしても計算時間を非常に要したり，高速で実行することのできる簡単な推論規則を数多く用意したとしてもそれらの適用が煩雑であったりするなど，定理証明の方法にもいくつかのトレードオフがあると思われる．本研究では，拡張有限状態機械上の指定された状態において指定されたレジスタ間の関係が成り立つかという検証対象を限定し，状態の対応や不変表明などを導入し，証明すべき問題を複数のより具体的な問題に帰着させ，具体化された問題を一定の手順で代数的定理証明技法を適用するという自動化を

行っている．

設計の高位における形式的検証手法として数多くの提案がなされているが，それらの比較のために用いるためのベンチマークは，多く提供されているとはいえない．文献[53]によるものには，記述レベルとしてアルゴリズムが指定されているレベル（algorithmic level ）からゲートレベルまで，組み合わせ回路（N bit adder）と順序回路（Traffic Light Controller，GCD，Tamarack Processor，Systolic Array），同期（前述のもの）と非同期（FIFO，Single Pulser ），データパスと制御部からなるモデル（GCD，Tamarack Processorなど）とそうでないモデル（Systolic Arrayなど）など，多くの種類のものが含まれているが，それぞれ分類での例題の種類は多いとはいえない．

しかも形式的検証手法は対象とする回路モデル及び検証の内容によって，問題の難しさが大きく異なり，今後，細分化された各分類においてより多くのベンチマーク回路が用意されることが望まれる．本研究で取り上げたソート回路等は，同期式順序回路の設計において，アルゴリズム自身の設計も含んだ抽象度の高いレベルから具体的な論理設計レベルまでの幅広いレベルの設計及び検証を行っており，上流工程における設計及び検証のベンチマークとして適したものであると思われる．

2　同期式順序回路の形式的記述および段階的設計

2.1　序言

本章では，単一の制御部を有する同期式順序回路（以下単に，順序回路と呼ぶ）に対して，要求仕様を記述する抽象的なレベルから，各部品の機能やバス構成などを記述する具体的なレベルまでの統一的な回路モデル及びその記述法について定義し，要求仕様から具体仕様までの段階的設計法について述べる．

以下，2.2では本論文で取り扱う順序回路の形式的モデルについて定義し，それを代数的言語で形式的に記述する方法について，2.3では，上位の回路および下位の回路の２つの回路仕様が与えられたとき下位の回路が上位の回路を正しく実現していること（設計の正しさ）の形式的定義について，2.4では，要求仕様から具体仕様まで段階的な設計方法について述べる．

2.2　代数的記述言語を用いた同期式順序回路の形式的記述法

順序回路は，拡張有限状態機械（Extended Finite State Machines，略してEFSM．あるいは，レジスタつき有限状態機械などと呼ばれる場合もある）でモデル化される．EFSMの記述法には，ソフトウエアの分野などにおいて用いられている代数的仕様記述言語ASLのサブセットであるASL/ASMと呼ばれるクラスが存在する[43]．しかし，ソフトウエア（プログラム）では，コンパイラが，動作の実行順をプログラムから抽出・解釈し，使用する計算機に応じた目的コードを生成するが，ハードウエア（回路記述言語）では，動作の実行順を規定するいわゆる制御部を，設計者が明記しなければならないということに注目しなければならない．仕様記述の立場からは，たとえば，SFL (Structured Function description Language)[3]のようにいくつかの制御用の構文を定め，それらを用いて記述するという立場もある．本論文では，2.3における実現の正しさの定義において，回路の実行制御をも含めて陽に議論するので，各仕様において制御機構に関する記述の意味定義を明確にすべきという立場をとる．もちろん，用いる構文の意味定義が以下で述べる代数的な構文に等価に変換できるのであれば，記述者にとって簡便に記述できるような（特に代数的言語にこだわることなく）構文を用いて回路を記述してもよい．

ここでは，ASL/ASM の記述法をもとに，回路の仕様記述法を定義する．まず，回路または回路を構成する部分回路（以下モジュールと呼ぶ）の状態を表す抽象データタイプを導入する．以下，このデータタイプを STATE 型（モジュールの場合は，第i番目のモジュールの状態を表すデータタイプを STATEi のように表す）と呼ぶ．回路の動作を状態遷移に対応させる．各動作は状態遷移関数で表される．状態遷移関数は，現状態を表す STATE 型の項と各入力を表す各入力の型の項の並びを引数とし，次状態を表す STATE 型の項を与える関数である．

STATE 型の特別な関数として，初期状態を表す初期化関数 INIT（あるいは init）を用いる．初期化関数の引数は，各成分の初期値などを表す回路に対するパラメータである．これらがないときは INIT は定数として扱われる．回路の取り得る状態は，たとえば INIT，T1(INIT) のように，初期化関数 INIT に状態遷移関数を施した項で表される．回路の各部品（状態成分と呼ぶ場合もある）が保持する値は，状態成分関数で表す．

代数的記述は，その記述で用いられる定数，関数および関数の引数のデータタイプを指定する文法と，文法により導かれる項の間での合同関係を定める公理からなる．以下での議論においては，回路を表現する公理についてのみ着目するので，文法に関する議論は省略する．また，公理はさらに，前提とする基本データタイプや基本関数の意味定義に関する部分と，それらを用いて記述すべき問題の意味などを表した部分にわけることができる．以下，本論文では，後者の部分のみについて議論し，前者の基本関数に関する部分については，特に断わらない限り省略する．

定義 1　順序回路

記述 C が次の形をしているとき，C を同期式順序回路（あるいは単に回路）と呼ぶ．

(1)基本関数を除いた他の関数は，各 STATEi (i = 1, ..., p)に対して次のように分類される．

(a)基本データタイプ以外の引数を持たず（引数がなくても良い），値域が STATEi の関数．各 i に付き一個のみ．以下では，INITi と書く．

(b)引数に STATEi のタイプのものを一つもち（他に基本データタイプの引数があっても良い），値域が STATEi の関数．複数個可．Ti に添え字をつけた Tij を用い，便宜上第一引数が STATEi のタイプを持つものとする．

(c)引数に STATEi のタイプのものを一つもち（他に基本データタイプの引数があっても良い），値域が基本データタイプの関数．複数個可．Fi に添え字をつけた Fij を用い，便宜上第一引数が STATEi のタイプを持つものとする．

(d)上記の(c)に含まれる関数のうち，値域として論理値をもつ関数 VALID，モジュール i の有限状態名の集合の要素を値域とする関数 CONTROLi（これらの集合を CONTROL 関数と呼ぶ）を，特に区別して用いる．

STATEi は i 番目のモジュールの抽象状態を表すデータタイプで，INITi はその初期状態を表す関数（初期化関数とも呼ぶ），Tij は状態遷移関数，Fij は状態成分関数である．

上記の分類に含まれない補助的な関数を用いてもよい．以下ではそれらも含めて基本関数と呼ぶ．

(2)上記(1)の関数の意味を定義する公理は次の形のもののみからなる．

(イ)Fik（INITi（xi1,...,xin） == expik(xi1,...,xin)

(ロ)Fik（Tij（si,xi1,...,xin)) == expikj(si, xi1,...,xin)

ただし，expik(...) は，基本関数と括弧内の変数 xi1，...，xin で構成される表現式（項）で，expikj(...) は，基本関数，モジュール i の状態成分関数，括弧内の変数 xi1，...，xin 及びで構成される表現式である．Fik（Tij（si，xi1,...,xin))の値の性質のみを記述したいとき，以下の形の公理を用いて，その性質を記述してもよい．

P（Fik（Tij（si,xi1,...,xin)), si,xi1,...,xin） == TRUE

ここで，P(...)は，状態siでの各状態成分の値Fj(si)，遷移Tij（si,xi1,...,xin)後の成分Fikの値Fik（Tij（si,xi1,...,xin))，変数及び基本関数からなる述語である．　□

上記の(イ)の形の公理は，各モジュールの初期状態における各状態成分関数の値を与えている．また(ロ)の形の公理は，各モジュールの状態遷移後の状態成分関数の値が取るべき値を指定している．

回路全体の抽象状態を全モジュールの状態の並び [STATE1,...,STATEn] で，回路全体の初期状態を全モジュールの初期状態の並び [INIT1,...,INITn] で，回路全体の状態遷移を全モジュールの状態遷移の並び [T1,...,Tn] で表す．以下では，回路全体の状態遷移を CK と表すこともある．また，回路全体の状態からモジュールiの部分状態を取り出す関数として Πi などを用いる．これらの関数に関して

Πi([STATE1,...,STATEn]) == STATEi

が成り立つ．これらの関数は基本関数として用い，これらの公理があることも前提として用いることとする．

同期式順序回路では，ソフトウエアなどと異なり，回路自身が状態遷移の実行順を決めなければならない．本論文では，状態遷移の実行順を記述するために，状態に関する述語（VALID 関数と呼ぶ）を導入し，状態 S において遷移 T を実行すべきとき，VALID(T(S)) が真と，実行すべきときでないとき偽となるように書くことにする．また VALID(S) が真であるとき状態 S を VALID 状態であるとよぶ．

代数的に VALID 関数を採用する理由は以下の通りである．回路の記述が与えられたとき，初期状態及び状態遷移に関する文法より，その記述の上で状態に関する項の集合は，初期状態に任意数の任意の状態遷移関数を施した項からなる．しかし，回路が初期状態から動作し得る状態集合はその一部分であり，回路の動作を考える上では実際に動作し得る状態のみに着目したい．そこで，実際に動作し得る状態集合を規定するための特別な VALID 関数を導入することにする．しかし，実際に動作し得る全状態についてそれぞれ VALID 関数を指定するのは実際上不可能であるので，初期状態から帰納的に実際に動作し得る状態を指定することにする．したがって，VALID 関数の(イ)の形の公理は，初期状態における引数変数が満たすべき条件を与える（以下の例では VALID(INIT)==TRUE とだけ書かれる場合が多い）．
VALID 関数の(ロ)の形の公理は，

VALID(Tij（s,xi1,...,xin)) == VALID(s) and expv(xi1,...,xin)

と書かれる．これは，状態 s から状態遷移 Tij を実行すべき条件を expv(...) で指定していると見なすこともできるし，このテキストにおいて，状態 s が VALID 状態であるとき，状態 s から状態遷移 Tij を実行したあとの状態は，条件 expv(...) が成り立つときに VALID 状態であるが，成り立たないときは VALID 状態でない（初期状態から到達し得ない状態である）ということを意味する．また，(イ)及び(ロ)の形の公理のみによって VALID 関数が指定されるので，回路が動作し得る状態遷移系列において，VALID 関数をその他の状態成分と同様に考えるのであれば成分VALIDの値は初期状態からその値は常に真であり，一旦偽となればそれ以降真となることは無い（このような状態を停止状態と呼ぶ）．

以下では，任意の状態 si において，任意の二つの状態遷移 Ti，Tj に対して，項 VALID(Ti(si)) と項 VALID(Tj(si)) が真となることはないと仮定する．すなわち，回路は各状態においていわゆる決定的な動作を行うものとする．

本論文では，順序回路を制御部に着目して拡張有限状態機械でモデル化することにする．この場合，有限状態名を保持するレジスタ（制御レジスタ，有限状態レジスタ，単に制御部と呼ぶ場合もある）を成分 CONTROLi で表し，その性質を上記の定義における(2)の形の公理で指定する．

以下では，VALID 及び CONTROL 関数に関する公理の集合を制御部の記述（または単に制御部），それ以外の(イ)，(ロ)の形の公理の集合を動作内容の記述（または単に動作内容）と呼ぶ．補助関数は，状態を持たない組み合わせ回路やバス構造を記述するために用いられる．

例１は，回路の要求仕様に相当するレベルである．動作内容は，抽象的な状態遷移関数を実行した前後の各成分が満たすべき値を，基本述語を用いて指定している．例２は，例１よりやや具体的なレベルである．動作内容は，レベル１と同様抽象的に記述された部分もあり，いわゆるレジスタ転送レベルの記述のように，各遷移で各成分間でどのようにデータ転送を行うかを具体的に指定した記述も混在するレベルである．

記述する対象が具体化されるにつれて，すべての動作内容の記述は，いわゆるレジスタ転送レベルに相当するものとなる．

例３は，各成分を実現する部品を具体的に記述し，部品の入力論理やバス構成などを具体的に記述したレベルである．代数的には，さらに具体的なレベルまで記述できるが，およそ例３のレベルは，CAD ツールが備わっている回路記述言語で記述可能なレベルであり，すでに，それ以降の設計の自動化が行われているレベルであるので，本論文が扱う具体的なレベルは例３のレベルまでとする．

例１及び例２でのこれらの記述から，回路はいわゆる状態遷移図で表すことができる．例３では，CONTROL 関数はいわゆるカウンタで実現されている．このレベルの VALID 関数は，停止条件及び各部品の入力への論理式も表している．

例１　代数的仕様記述の例（ソート回路：要求仕様レベル）

ここでは，メモリの指定された範囲の要素を昇順に並べ替えるソート回路を例に，抽象レベルでの記述から具体的レベルまでの代数的仕様記述を説明する．各レベルで行われる設計に関しては 2.4 で述べる．

もっとも抽象的なレベル（以下レベル１）では，一連のメモリ内の要素を並べ替える動作を一つの遷移 SORT の実行で表す．状態成分関数としては，ソートの対象となる整数配列Eを用いる．遷移 SORT によってソートされる配列の下限上限をそれぞれ状態成分関数 K，L で表す．

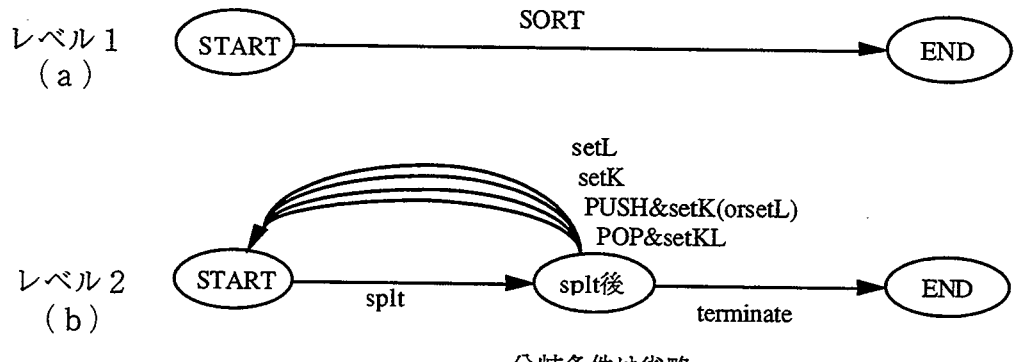
このレベルの記述を表１にまとめる．$\alpha:\beta$ の形の式は，α がラベルであり，β が公理を表す．公理を指定するときは，ラベルを用いておこなう．公理INIT1～INIT3は，状態成分関数 E，K，Lの初期状態 init(init_data, max_E,min_E) での値を指定している．init_data，max_E，min_Eは，それぞれ外部から与えられるパラメータであり，各成分の初期値を表す．

公理 SORT1 及び SORT2 は，抽象状態 s から状態遷移 SORT を実行した後の状態における配列E の値 E(SORT(s)) が満たすべき性質を記述している．公理 SORT1 は配列 E(SORT(s)) の要素が昇順に並んでいることを表し，公理 SORT2 は，遷移 SORT の実行前後における配列 E の各要素が集合として同じであること，すなわち状態遷移中に，要素が欠落・混入しないことを意味している．関数 get(a,b) は，配列 a の b 番目の要素を取り出す基本関数，number(a,i,j, k) は，配列 a の i 番目から j 番目までの要素の中に k と等しい要素の数を与える基本関数である．以上がレベル１の動作内容を表す公理の集合である．

CONTROL1，2 及び VALID1，2 は，レベル１の制御部を表す．ここでは，状態遷移 SORT を初期状態から一度実行し，その後停止するという制御を行わせたいとする．すなわち，初期状態 init(...) では，CONTROL の値は START であるので，VALID(SORT(init(...))) が真となり，状態遷移 SORT が実行される．その結果，CONTROL の値は END となる．その状態において真となるVALID 関数は無いので，回路は停止状態となる．レベル１の制御部の記述を状態遷移図で表したものが，図１(a)である．

表１　ソート回路の要求仕様

```
text level1(init_data, max_E, min_E);

INIT1:E(init(init_data, max_E, min_E)) == init_data;
INIT2:K(init(init_data, max_E, min_E)) == min_E;
INIT3:L(init(init_data, max_E, min_E)) == max_E;

SORT1:min_E <= K(s) and K(s) <= i and i < j
      and j <= L(s) and L(s) <= max_E imply
        get(E(SORT(s)),i) <= get(E(SORT(s)),j) == TRUE;
SORT2:min_E <= K(s) and K(s) <= L(s) and L(s) <= max_E imply
        number(E(s),K(s),L(s),d) <= number(E(SORT(s)),K(s),L(s),d) == TRUE;

CONTROL1:CONTROL(init(init_data, max_E, min_E)) == START;
CONTROL2:CONTROL(SORT(s)) == END;

VALID1:VALID(init(init_data, max_E, min_E)) == TRUE;
VALID2:VALID(SORT(s)) == VALID(s) and CONTROL(s) = START;

end;
```



分岐条件は省略

図１　クイックソート法によるソート回路のレベル１とレベル２の状態遷移図

例２　ソート回路をクイックソート法により実現したレベル

ここでは，レベル１に対してやや具体的なレベルの回路の仕様記述の例について述べる．ここでは，ソートをクイックソート法を用いて行うことを決めたレベルである．しかし，メモリの決められた範囲の要素を基準値より大きいあるいは小さい集合として分割する操作は，抽象的な一遷移で表している．

このレベルでは，状態成分としてレベル１の状態成分の他に分割操作後，分割すべき場所を保持するレジスタ b_position，分割すべき範囲（上限下限の組）を保持しておくスタック ST及びスタックポインタ p が加えられている．用いる状態遷移としては，遷移 splt，PUSH&setL，setL，setK，POP&setKL，terminate がある．それぞれ以下の動作を行う．

遷移 splt は，レジスタ K 及び L によって指定された範囲内の要素をある基準値で分割する．その基準値の番地を成分 b_position が保持している．K から b_position までの要素は基準値以下，b_position から L までの要素は基準値以上である．基準値としてどのような値が用いられるかはこのレベルでは決めない．

遷移splt後の，各成分K，b_position，L の値の関係から次の５つの遷移が実行される．

（場合１）K と b_position の間，b_position と L の間が一以上ある場合．

この場合，K，L をそれぞれKと b_position-1 に設定し，分割操作 splt を行い，その後 K，L をそれぞれ b_position+1 と L に設定し，さらに分割操作 splt を行う（順は逆でも良い）．ここで，b_position+1 と L の値をスタックに格納しておく．これらの動作を遷移 PUSH&setL が行う．その後分割操作 splt を繰り返す．

（場合２）K と b_position の間が一以上あるが，b_position=L あるいは b_position=L-1 の場合．

この場合，場合１に対し，b_position と L の間を splt する必要はない．L の値を b_position-1 の値に設定すればよい．この動作を遷移 setL が行う．その後分割操作 splt を繰り返す．

（場合３）b_position と L の間が一以上あるが，K=b_position あるいは K+1=b_position の場合．

この場合，場合２と同様に，b_position と K の間を splt する必要はない．K の値を

b_position+1 の値に設定すればよい．この動作を遷移 setK が行う．その後分割操作を繰り返す．

（場合４）b_position=L あるいは b_position=L-1 かつ，K=b_position あるいは K+1=b_position かつスタック ST に格納されているデータがある場合．

この場合，すでに格納されている範囲のペアを取り出し（スタックに蓄えられているどのデータでも良いが，ここでは，スタックポインタpがさす番地のデータを取り出す），その値を K，L に設定すればよい．これらの動作を遷移POP&setKLが行う．その後分割操作 splt を繰り返す．

（場合５）b_position=Lあるいはb_position=L-1かつ，K=b_positionあるいはK+1=b_positionかつスタック ST に格納されているデータがない場合．

この場合，初期状態において指定されたすべての範囲の分割操作 splt が終了（すなわちソートが終了）したので，遷移 terminate を実行して停止状態に至る．

これらの遷移の動作内容及び，各遷移の実行順を実現した制御部の記述を表２に示す．

以下では

$$F(T(s)) == F(s)$$

の形の公理は，特に断らない限り省略する．

レベル２の制御部を，状態遷移図にしたものが図１(b)である．

表2　クイックソート回路のレベル２の記述例

```
text level2(init_data, max_E, min_E);

INIT1:E(init(init_data, max_E, min_E)) == init_data;
INIT2:K(init(init_data, max_E, min_E)) == min_E;
INIT3:L(init(init_data, max_E, min_E)) == max_E;
INIT4:p(init(init_data, max_E, min_E)) == 0;

splt1:min_E <= K(s) and K(s) <= L(s) and L(s) <= max_E imply
        number(E(s),K(s),L(s),d) <= number(E(SORT(s)),K(s),L(s),d) == TRUE;
splt2:min_E <= i aand i < K(s) or L(s) < i and i <= max_E imply
        get(E(SORT(s)),i) = get(E(SORT(s)),i) == TRUE;
splt3:min_E <= K(s) and K(s) <= i and i <= b_position(splt(s))
      and b_position(splt(s))  <=  max_E imply
        get(E(SORT(s)),i) <= get(E(SORT(s)),b_position(splt(s))) == TRUE;
splt4:min_E <= b_position(splt(s)) and b_position(splt(s)) < j
      and j <= L(s) and L(s) <= max_E imply
        get(E(SORT(s)),b_position(splt(s))) <= get(E(SORT(s)),j) == TRUE;

L(PUSH&setL(s)) == b_position(s) - 1;
p(PUSH&setL(s)) == p(s) + 1;
L(PUSH&setL(s)) == put2(ST(s),p(s) + 1 , pair(b_position(s) + 1 , L(s));

(*PUSH&setLの代わりに以下でも良い
K(PUSH&setL(s)) == b_position(s) + 1;
p(PUSH&setL(s)) == p(s) + 1;
L(PUSH&setL(s)) == put2(ST(s),p(s) + 1 , pair(K(s),b_position(s) - 1);
*)

K(setK(s)) == b_position(s) + 1;

L(setL(s)) == b_position(s) - 1;

K(POP&setKL(s)) == high_side(get2(ST(s),p(s));
L(POP&setKL(s)) == low_side(get2(ST(s),p(s));
p(POP&setKL(s)) == p(s) - 1;

CONTROL(init(init_data, max_E,min_E)) == START;
CONTROL(splt(s)) == splt後;
CONTROL(PUSH&setL(s)) == START;
CONTROL(setK(s)) == START;
CONTROL(setL(s)) == START;
CONTROL(POP&setKL(s)) == START;
CONTROL(terminate(s)) == END;
```

```
VALID(init(init_data, max_E, min_E)) == TRUE;
VALID(splt(s)) == VALID(s) and CONTROL(s) = START;
VALID(PUSH&setL(s)) == VALID(s) and CONTROL(s) = splt後 and
                       K(s) + 1 < b_position(s) and     b_position(s) < L(s) - 1;
VALID(setK(s)) == VALID(s) and CONTROL(s) = splt後 and
                  not K(s) + 1 < b_position(s) and     b_position(s) < L(s) - 1;
VALID(setL(s)) == VALID(s) and CONTROL(s) = splt後 and
                       K(s) + 1 < b_position(s) and not b_position(s) < L(s) - 1;
VALID(POP&setKL(s)) == VALID(s) and CONTROL(s) = splt後 and
                  not K(s) + 1 < b_position(s) and not b_position(s) < L(s) - 1 and
                  p(s) > 0;
VALID(terminate(s)) == VALID(s) and CONTROL(s) = splt後 and
                  not K(s) + 1 < b_position(s) and not b_position(s) < L(s) - 1 and
                  p(s) = 0;
end;
```

例３　ソート回路の具体仕様レベル（制御部をワイヤード論理方式で実現した場合）

ここでは，採用する具体的なモジュールやその入出力の結線を記述するレベルであり，しかも，制御部をワイヤードロジックで実現した場合の代数的記述について述べる．このレベルをレベル６と呼ぶことにする．

レベル６では，メモリ，レジスタ，ALU などが具体的な記述が与えられ（ただし，各部品のデータ入力に関しては整数型の変数を用いて，出力はそれらの値あるいは整数型の関数などによって指定されている），部品間のレジスタ転送を表す各状態遷移関数は，一クロックで実行できる．用いる部品の中からレジスタとメモリの記述を表２に示す．

表３　クイックソート回路のレベル６で用いるモジュールの記述例

```
(*  レジスタの記述  *)
text register(name, init_data);

name(init_name(init_data)) == init_data;
name(CK(s_name, name_ctl, name_in)) ==
  if name_ctl = LOAD then name_in
              else name(s_name); (* name_in が HOLD のとき  *)

end;

(*  メモリの記述  *)
text memory(name, init_data);

name(init_name(init_data)) == init_data;
name(CK(s_name, name_in, name_ctl, name_adr)) ==
  if name_ctl = WRITE then put(name(s_name), name_adr, name_in)
              else name(s_name); (* name_in が READ のとき  *)

end;
```

テキストには，各部品の名前 name がパラメータとして与えられており，回路全体を表すテキストから名前 name を具体的に指定されて読み込まれることにより，それぞれの具体的な名前を持つ部品として回路全体を表すテキストに展開される．

このレベルでは，2本のバス bus1，bus2 を用いることにする．レジスタ i 及びレジスタ K のデータ入力が bus1 に，レジスタ j 及びレジスタ L のデータ入力が bus2 に，常に接続されている．回路全体の記述を表3に示す．

回路全体の状態は，各部品の状態の組[s_K, s_L, s_i, ..., s_p, s_E, s_ST]で表される．VALID な初期状態は，[initK(max_E), initL(min_E), init_i(*), ..., init_p, initE(init_data), init_ST(*)]である．状態遷移関数は，このレベルではクロックにより各部品が同期して実行することを表す遷移

[CK_K(s_K, k_in, k_ctl),

CK_L(s_L, l_in, l_ctl),

CK_i(s_i, i_in, i_ctl),

:

CK_p(s_p, p_in, p_ctl),

CK_E(s_E, E_in, E_ctl, E_adr),

CK_ST(s_ST, ST_in, ST_ctl, ST_adr)]

一つである．この状態遷移に関する VALID 関数は，停止状態を表す条件と各引数にどのような値を与えるかを表す式の論理積で表される．実際には，各引数（各部品の入力）に，式が表す組み合わせ回路を接続し，各部品に同一のクロックを供給した回路に相当する．

代数的言語 ASL において，γ where $\alpha = \beta$ からなる文は，γ に現れる文字列 α を式 β で置き換えた文と同じである．

表4　クイックソート回路のレベル6の回路全体の記述

```
text level6(init_data, max_E, min_E);

include register(K, max_E);
include register(L, min_E);
include register(i, *);(*  * はドントケア　任意の値を表す  *)
    :
include counter(p,0);
include memory(E,init_data);
include memory(ST,*)

VALID([initK(max_E),initL(min_E),init_i(*),...,init_p,initE(init_data),init_ST(*)]) == TRUE;
VALID([CK_K(s_K,k_in,k_ctl),
       CK_L(s_L,l_in,l_ctl),
       CK_i(s_i,i_in,i_ctl),

       CK_p(s_p,p_in,p_ctl),
       CK_E(s_E,  E_in, E_ctl, E_adr),
       CK_ST(s_ST,ST_in,ST_ctl,ST_adr)]) ==
   VALID([s_K,s_L,s_i,...,s_p,s_E,s_ST]) and
   not CONTROL(s_CONTROL) = END  (* 停止状態の条件  *)
   and
     k_in = bus1   (*  レジスタKのデータ入力が常にbus1に接続していることを指定 *)
   and
     k_ctl = if CONTROL(s_CONTROL) = splt後 and
                  not K(s_K) + 1 < i(s_i) and     i(s_i) < L(s_L) - 1 or
                CONTROL(s_CONTROL) = splt後 and
                  not K(s_K) + 1 < i(s_i) and not i(s_i) < L(s_L) - 1 and p(s_p) > 0
             then LOAD
             else HOLD (*  レジスタKの制御入力の論理の指定 *)
   and
     l_in = bus2
   and
     l_ctl = if CONTROL(s_CONTROL) = splt後 and
                     K(s_K) + 1 < i(s_i) and     i(s_i) < L(s_L) - 1 or
                CONTROL(s_CONTROL) = splt後 and
                K(s_K) + 1 < i(s_i) and not i(s_i) < L(s_L) - 1 or
                CONTROL(s_CONTROL) = splt後 and
                not K(s_K) + 1 < i(s_i) and not i(s_i) < L(s_L) - 1 and
                  p(s_p) > 0
             then LOAD
             else HOLD
        :
```

```
  }
where
  bus1 = if CONTROL(s_CONTROL) = split後 and
             not K(s_K) + 1 < i(s_i) and not i(s_i) < L(s_L) - 1 and
               p(s_p) > 0
         then high_side(get2(ST(s_ST),p(s_p))
  else if CONTROL(s_CONTROL) = START
         then K(s_K)
  else       i(s_i) + 1
where
  bus2 = if CONTROL(s_CONTROL) = split後 and
             not K(s_K) + 1 < i(s_i) and not i(s_i) < L(s_L) - 1 and
               p(s_p) > 0
         then low_side(get2(ST(s_ST),p(s_p))
  else if CONTROL(s_CONTROL) = START
         then L(s_L)
  else       j(s_j) - 1;
```

2.3 同期式順序回路の設計の正しさの定義

本節では，上位及び下位の２つの回路仕様が与えられたとき，上位の回路の状態と下位の回路の状態との対応関係を与え，その上で下位の回路が上位の回路を正しく実現していることの形式的定義を与える．

同じ基本関数を前提とする２つの記述レベルの異なる回路仕様 A，B を考える．仕様 A は，抽象レベルの高い上位の回路仕様であるとする．仕様 B は，仕様 A に対してより具体的な成分などを用いて記述される下位の回路仕様であるとする．仕様 A を仕様 B で実現するとき，仕様 A の一つの状態遷移は，仕様 B では一般に複数の状態遷移を連続して実行することによって実現されることが多い．この時仕様 B の状態には，仕様 A の状態に対応する状態と，仕様 A の一つの遷移を実現すべき状態遷移の系列の途中の状態とが存在すると考えられる．前者のような状態を以下では着目すべき状態と呼び，仕様 B の着目すべき状態か否かを述語 IS (Interested State)で与え，状態 β が着目する状態なら IS(β) が真，そうでなければ偽となるように定義する．

仕様 A の各状態成分関数 Fk に対し，EXPFk(S) を仕様 B の各状態成分関数 fi，基本関数及び回路全体の抽象状態を表すデータタイプの変数 S のみからなる表現式とする．このとき，

F1(S) == EXPF1(S)

F2(S) == EXPF2(S)

:

を仕様 B から仕様 A への状態成分の対応と呼び，状態成分の対応の集合を EXPF と表す．

仕様 A において，初期状態から状態 α に至るまでの全状態成分関数値の列を出力列 O（α）と表す．仕様 B において，状態成分の対応 EXPF と状態に関する述語 IS が与えられたとき，初期状態から状態 β までの状態のなかで着目すべき状態での列<EXPF1,...EXPFn>の列を出力列 OIS（β）と表す．２つの出力列 O（α）と OIS（β）が IS 及び EXPF のもとで等価とは，列を構成する要素の数が同じであり，かつ，各要素の値がそれぞれ等しいことである．

上位の回路の各成分と下位の回路の異なった状態における成分との対応という立場で，上述の出力列等価を拡張して議論することもできる．例えば，パイプライン CPU などのよう

に，ある命令が実行が終わったときには，次のいくつかの命令が実行途中であるというような場合である．これらの場合の議論は文献[20]及び[51]で行われている．

定義　2　順序回路の実現の正しさ

2つの仕様A，仕様B，仕様Bから仕様Aへの状態成分の対応EXPF，仕様Bの状態に関する述語ISが与えられたとする．次の条件を満たすとき，EXPF及びISのもとで仕様Bは仕様Aを実現するという．

（1）仕様Aの任意のVALID状態αに対して，仕様Bにおいて VALID状態かつIS(β）が真である状態βが存在し，2つの出力列O（α）とOIS（β）がIS及びEXPFのもとで等価である．

（2）特に条件(1)においてαが停止状態であるとき，βも停止状態である．　□

条件（1）は仕様Bが仕様Aといつも同じ値を出力することを表し（以下，出力等価の条件），条件(2)は，2つの回路が同時に停止するということを表す（以下，同時停止の条件と呼ぶ）．

しかし，一般には，無限系列である2つの出力系列の比較は困難である．そこで，本論文では状態遷移の対応という概念を導入し，状態遷移の対応が正しいことを示せば順序回路の実現の正しさを保証できるという方法を採用する．

仕様Aの初期状態と各状態遷移関数Tjに対して，仕様Aの各モジュールの初期状態を表す関数INITj(...)に対しEXPINITj(...)を仕様Bの各状態遷移関数ti，基本関数及び変数sのみからなる表現式とし，仕様A の各モジュールの状態遷移関数Tjに対してEXPTj(S)を仕様Bの各状態遷移関数ti，基本関数及び変数sのみからなる表現式とする．このとき，

INIT1(...) == EXPINIT1(...)

INIT2(...) == EXPINIT2(...)

　　:

T1(S) == EXPT1(S)

T2(S) == EXPT2(S)

:

を仕様 B から仕様 A への状態遷移の対応と呼び，これらの公理の集合を単に EXPT と表す．状態遷移の対応は，仕様 A において状態遷移 Tj を実行する時に，仕様 B では EXPTj で指定された順に複数の状態遷移を実行するということを表す．

定義　3　対応の正しさ

２つの仕様 A，仕様 B，仕様 B から仕様 A への状態成分の対応 EXPF，仕様 B から仕様 A への状態遷移の対応 EXPT が与えられているものとする．このとき次の条件が成り立つとき，仕様 A，B に対して対応 EXPT と EXPF は正しいという．

（条件１）仕様 A の各公理

Fi(INIT(...)) == initi(...)

Fi(Tj(s)) == expij(...)

に対して，仕様 B の公理，EXPF 及び EXPT の公理を用いて

Fi(INIT(...)) ≈ initi(...)

Fi(Tj(s)) ≈ expij(...)

である（$\alpha \approx \beta$ は，対応 EXPT と EXPF 及び仕様 B とをあわせた代数的記述上で定理として成り立つことを表す．あるいは，仕様 A において同値類である α と β が，対応と仕様 B のもとでも同じ同値類に属するということを意味する）

（条件２）上記の各表現式 initi(...) 及び expij(...) などが，値を持つこと．

（条件３）仕様 B，状態成分の対応 EXPF，状態遷移の対応 EXPT を合わせたテキストが，無矛盾であること．　□

対応 EXPT と EXPF が正しいとき，

EXPFi(EXPINIT(...)) ≈ initi(...)

EXPFi(EXPTj(s)) ≈ expij(...)

が成り立つ．このことは，仕様 B が EXPT で指定された順に状態遷移を実行すれば，仕様 B の各成分は，EXPF のもとで仕様 A で要求された値に等しくなることを意味する．

同期式順序回路の設計において取り扱う状態遷移の対応のクラスについては，2.4 で定める．

定理 1　順序回路の実現の正しさの十分条件

2 つの仕様 A，B，仕様 B から仕様 A への状態成分の対応 EXPF，仕様 B から仕様 A への状態遷移の対応 EXPT が与えられ，仕様 A，B に対して対応 EXPT と EXPF が正しいとき，仕様 B の状態に関する述語 IS が存在し，EXPF 及び IS のもとで仕様 B は仕様 A を実現する．

□

[証明]

仕様 A の状態 α の初期状態より状態遷移を行った回数Nを用いた帰納法により証明を行う．以下では，仕様 A の各関数，定数にAのサフィクスを，仕様Bの各関数，定数に仕様 B のサフィクスをそれぞれ付加する．

（初期段階）

N＝1 の時，β ＝EXPINIT(...) とする．IS(EXPINIT(...)) を真とし，EXPINIT(...) のすべての部分項に対応する状態 γ に対する IS(γ) を偽とする．VALID 関数の定義より，INITA は VALID 状態である．INITA に対応する仕様 B の状態 β（β が初めて着目する状態である）は

VALID(β) = VALID(EXPINIT(...))　(*　β の定義より　*)

= VALID(INITA)　(*　公理 INITA==EXPINIT(...) より　*)

= true　(*　INITA が VALID 状態より　*)

より VALID 状態である．同様に

OIS(EXPINIT(...)) = [EXPF1(EXPINIT(...)), EXPF2(EXPINIT(...)),...,EXPFn(EXPINIT(...))]

= [F1(INITA(...)), F2(INIAT(...)),...,Fn(INITA(...))]　(*対応の正しさより*)

= O(INITA(...))

より，出力等価の条件が成り立つ．

（帰納段階）

仕様Aにおいて，初期状態よりN回状態遷移を行ったVALID状態αに対し，仕様Bにおいて出力等価であり，なおかつISが真となるようなVALID状態βが存在するものとする．

仕様Aにおいて，αから実行されるべき状態遷移は高々一つであり，存在するならこれをTjとする．仕様Bにおける状態EXPTj(β)におけるISを真とし，状態βから状態EXPTj(β)に至るまでの各状態におけるISを偽とする．

$$\begin{aligned} VALID(Tj(\alpha)) &\approx VALID(\alpha)\ and\ bj(\alpha) \\ &\approx bj(\alpha) \end{aligned}$$

$$\begin{aligned} VALID(EXPTj(\beta)) &\approx VALID(Tj(\beta)) \\ &\approx VALID(\beta)\ and\ bj(\beta) \\ &\approx bj(\beta) \end{aligned}$$

仮定より，状態αと状態βでの各状態成分はEXPFのもとで等しいので，bj(α)とbj(β)の値は等価である．よって，EXPTj(β)は，Tj(α)がVALID状態であればVALID状態である．また，仕様Aにおいて，αが停止状態であれば，βも停止状態であることがいえる．

同様に，対応が正しいことを利用して

$$O(Tj(\alpha)) \approx OIS(EXPTJ(\beta))$$

がいえ，初期状態INITAからN+1回状態遷移を行ったVALID状態Tj(α)と状態EXPTj(β)は出力等価であり，同時に停止することがいえる．

以上の証明より，定理1が証明された． □

上述の証明において，仕様Bの状態に関する述語ISの与え方も示されている．

2.4 要求仕様レベルから論理設計レベルまでの段階的な回路設計法

本節では，要求仕様から具体仕様まで段階的に回路を設計し，かつその設計により得られた回路が要求仕様を正しく実現しているということを実際に形式的に証明できる設計手順について具体的に述べる．

以下では，仕様 A を正しく実現することを形式的に証明することが可能である仕様 B の設計手順を示す．

設計手順

仕様 A が与えられたとき，仕様 A を正しく実現することを形式的に証明することが可能である仕様 B の設計手順は以下の通りである．ここでは，仕様 B の動作内容と状態遷移の対応を設計者が考案し，それから仕様 B の制御部を合成するという方法である．

(1)仕様 B で用いる状態成分，状態遷移を決定し，それらから仕様 B の動作内容を決定する．

(2)仕様 B から仕様 A への状態成分の対応 EXPF を決定する．

(3)仕様 B から仕様 A への状態遷移の対応 EXPT を決定する．

(4)仕様 A，B に対して，対応 EXPF と EXPT が正しいことを証明する．

(5)仕様 B の状態に関する述語 IS，制御部を合成する．

(6)必要なら，得られた仕様 B を簡約化する．

(7)具体的なレベルであれば，マイクロプログラムや制御フリップフロップの入力論理を合成する．

これ以降のレベルでは，組み合わせ回路の設計，状態割り付け，各論理の圧縮を行い，これ以降の自動設計ツールを用いて実際の回路を得ることができる．

上記手順(4)の証明については 4 章で，設計全体の支援については 3 章で述べる．以下では，状態遷移の対応を記述するクラスを定め，手順(5)における述語 IS 及び制御部の生成法について述べる．

本設計法は階層的に繰り返し適用されるということを前提としているので，仕様 A 及び仕

様 B の制御部ともに次の形の公理のみを扱うことにする．

VALID(INIT(...)) == TRUE

VALID(Ti(s)) == VALID(s) and (

CONTROL(s) = state1 and b1(s) or ... or

CONTROL(s) = staten and bn(s)

CONTROL(INIT(...)) == START

CONTROL(Ti(s)) ==

if CONTROL(s) = state1 and b1(s) then state1'

else if ...

else staten'

ここで，bi(s) は CONTROL，VALID 以外の状態成分関数からなる論理式である．

また，動作内容の公理には，CONTROL，VALID 関数が現れてはならない．前述した例 1 から例 3 での各仕様は，この条件を満たしている．

定義 4　制御部の公理の形と状態遷移の対応のクラス

遷移の対応 EXPF の各公理の右辺をなす表現式 EXPTi の形の構成方法は，以下のように定義する．ti を仕様 B の状態遷移関数，COND を基本関数及び仕様 B の状態成分関数からなる論理式とする．表現式 EXPTi として許される表現式集合 SB は次の(a)～(d)によって再帰的に構成されるもののみからなる．EXP1，EXP2∈SBであるとき

(a) ti ∈SB

(b) EXP2(EXP1) ∈SB

あるいは

EXP1・EXP2 ∈SB

(c) if COND then EXP1 else EXP2 ∈SB

(d) (EXP1)∈SB

(e) if COND then EXPTi（EXP1） else EXP2 ∈SB

COND は，EXPTi を実行すべき状態における仕様 B の各成分関数の値をもとに計算される式である．

構成法(a)は，仕様 B の遷移はそれだけで対応として許されることを，(b)は EXP1 が表す遷移系列を実行したのち，EXP2 が表す遷移系列を実行することを，(c)は，開始状態で COND が成り立つとき，EXP1 が表す遷移系列を実行し，成り立たないとき EXP2 が表す遷移系列を実行することを，(e)は，COND が成り立つ間，EXP1 が表す遷移系列を繰り返し実行し，成り立たないとき，EXP2 が表す遷移系列を実行することを表す． □

また，遷移の対応 Tj(s)=EXPTj(s) における EXPTj(s) は，始点が一つ終点が一つ，また途中節点において複数の遷移が出射する枝がある場合は各枝に付加されている実行条件が互いに排他的であり，任意の節点から終点への経路が必ず存在する状態遷移図として表すことができる．以下では，簡単のため，遷移の対応を状態遷移図で表すこともある．

例えば，上述の定義におけるTiの記述に対しては，state1，...というラベルを付加された節点を始点とし，state1'，...というラベルを付加された節点を終点とする枝e1，...があり，各枝 ej には＜遷移名 Ti，実行条件 CONDj>というラベルが付加されていると見なすことができる．各遷移に対してこれらの部分グラフの集合を，状態遷移図と定義する．以下では，枝に対して，その遷移に与えられた動作内容も付加されたものと考える場合もある．

下位の回路仕様の制御部の合成アルゴリズム

入力　上位の回路仕様における制御部の記述

　　状態成分の対応 EXPF

　　状態遷移の対応 EXPT

出力　下位の回路仕様における制御部の記述．ただし，上位の回路仕様における制御部の各公理が，下位の回路仕様の公理，状態成分の対応 EXPF 及び状態遷移の対応のもので定理となること．

アルゴリズム　上位の回路の制御部の記述，状態遷移の対応 EXPT にそれぞれ対応する状態

遷移図 SD，SDT を求める．状態遷移図 SD の各枝を，対応する SDT で置き換え SD' を得る．ただし，各 SDT の始点から出射する枝の各実行条件には，その条件と元の枝の実行条件との論理積を与える．SD' の各実行条件における各状態成分を，状態成分の対応 EXPF で置き換え，得られた状態遷移図に対応する制御部の記述を下位の回路の制御部とする．

アルゴリズムの正しさの証明．

状態遷移の対応関係の構成方法による，上位の回路の状態遷移図の初期制御部状態からの構造的帰納法により証明することができる．アルゴリズムにおける記述から状態遷移図の生成及びそれらの状態遷移図による置き換え回数が有限であるので，本アルゴリズムは停止する．

本論文では，以下の理由により対応関係のクラスを定めた．上述の対応関係は末端再帰による状態遷移系列の繰り返しのみ許している．ただし，それらの組み合わせが可能であるので，それらの組み合わせからなる状態遷移系列は，始点終点がそれぞれ一個であるような有限状態遷移図に対応するクラスである．より広いクラスとして，一般の再帰を許すクラスがある．この場合，いわゆる再帰呼び出しが行われたとき，呼び出しが終わったときに復帰すべき有限状態の値や状態成分の値などを記憶すべきスタックやスタックポインタを新たに導入し，さらにそれらへのデータの転送なども含めた制御を実現しなければならない．一般に，回路記述においては，これらの部品の記述（特にスタック）は，回路規模を著しく大きくしてしまう原因となり得，また実行に要するクロック数を増大させてしまう．設計者が実行回数を削減しよう思って考案した状態遷移の対応であっても，その中に一般の再帰が含まれる場合，合成される制御部の規模が増大し，実行クロック数が当初の予想に反して増大してしまうことになりうる．また，これらの制御部を最適化して規模を小さくするという方法も考えられるが，一般の再帰を含んだ対応に対する効率的な制御部合成は一般には難しい．

本手法で定めたクラスでは，関数 CONTROL が保持する有限状態の値は増えるが，新たな成分は追加しない．また，いくつかの nop(有限状態の値のみが変わり，成分は何も変わらない遷移）が追加されてしまう可能性があるが，以下で述べる簡約化手順においてそれらは取り除かれる．

以下では，本手法を用いることにより，定めた状態遷移の対応のクラスが十分であり設計者の意図する設計が自由に行えること，実際に設計の正しさが検証できること，設計手順における簡約化により人手で直接設計するのと同性能の回路が段階的手法で得られることを，ソート回路の例を中心に説明する．

設計例　クイックソート回路の段階的設計

例１で示したソート回路の要求仕様から論理設計レベルへの設計例を示す．クイックソート法により実現される．一つの遷移 SORT の実行で表すレベル１の仕様から，制御部をワイヤード論理方式による実現（レベル１からレベル６まで）と，マイクロプログラム制御方式による実現（レベル１からレベル７’）まで段階的に設計を行った（図２）．人手によりレジスタ転送レベルにおける設計との比較のために，用いる部品の機能，バス構造などのデータパスアーキテクチャは文献[26]のものと同じものとした．いずれの実現においても，論理設計レベルにおいて人手により実行条件の判定やメモリのアクセスに要するデータパスに創意工夫を行ったのと同じ回路（ワイヤード論理方式の制御部においては，状態数，分岐部の分岐条件，各動作によるレジスタ転送の種類．マイクロプログラム制御方式の制御部においては，同じマイクロプログラム）が得られた．

レベル１及びレベル２の記述は，それぞれ表１及び表２に示した．レベル１からレベル２への設計は，以下の手順で行った．

(1)レベル２で用いる状態成分，状態遷移を決定し，それらから表２における動作内容（CONTROL，VALID 関数以外の公理）を決定する．レベル２で用いる状態遷移は，分割操作を行う split，例２で述べた５つの場合のそれぞれを実行する PUSH&setL，setL，setK，POP&setKL，terminateである．

(2)状態成分の対応は，レベル１の各成分に対し，レベル２の同名の成分を対応させる．公理としては

FA(s)==FB(s)

という形の公理となる以下では，各レベルで同じ名前を用いているのでこれらの公理は省略する．

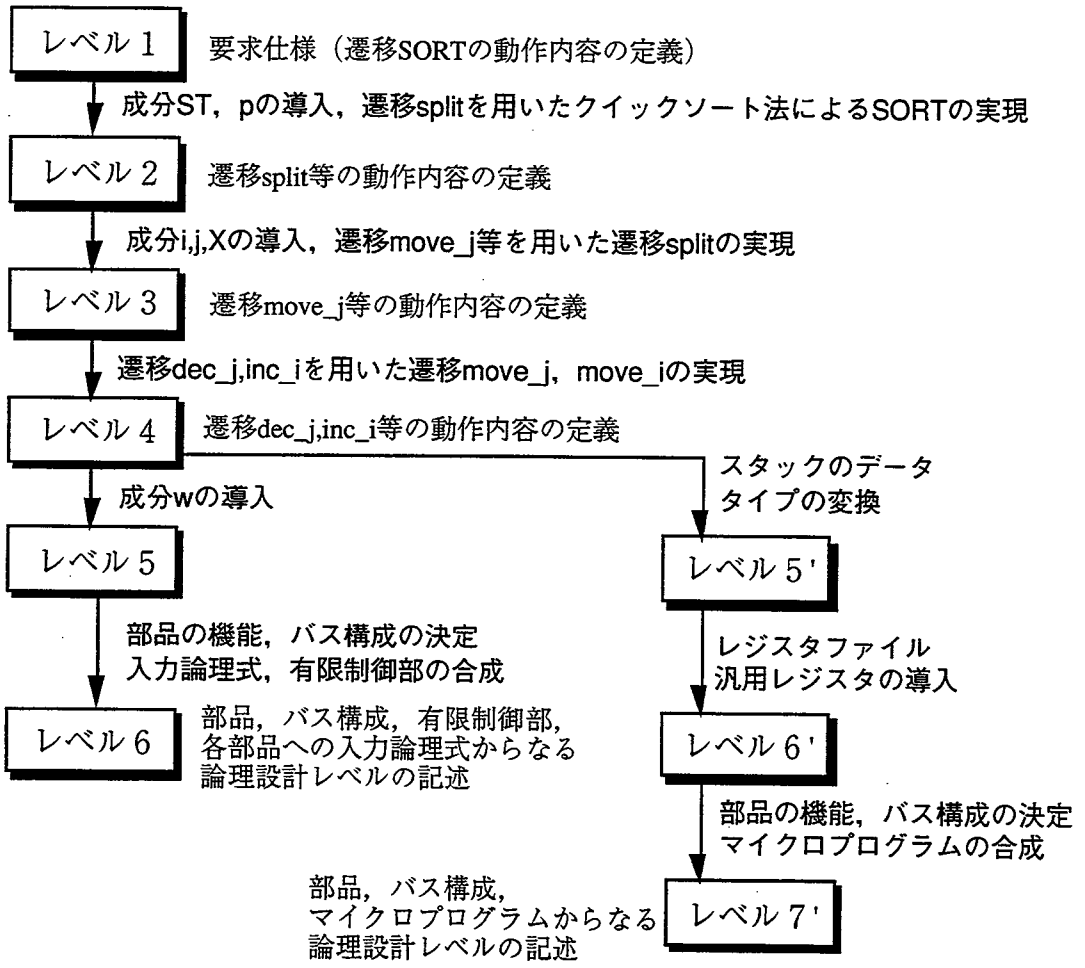


図2　クイックソート回路の階層的設計の概要

表５　状態遷移の対応の代数的記述の例

```
SORT(s) == loop_1(split(s));
loop_1(s) ==
      if        not(K(s) = b_position(s) or K = b_position(s) - 1)
        and not(b_position(s) = L(s) or b_position(s) + 1 = L(s))      then
                  SORT(PUSH&setL(s))
 else if     not(K(s) = b_position(s) or K(s) = b_position(s) - 1)
     and   (b_position(s) = L(s) or b_position(s) + 1 = L(s))
    then
                  SORT(setL(s))
 else if     (K(s) = b_position(s) or K(s) = b_position(s) - 1)
        and not(b_position(s) = L(s) or b_position(s) + 1 = L(s))      then
                  SORT(setK(s))
 else if     (K(s) = b_position(s) or K(s) = b_position(s) - 1)
        and   (b_position(s) = L(s) or b_position(s) + 1 = L(s))
         and not p(s) = 0  then
                  SORT(POP&setKL(s))
 else
                  terminate(s);
```

(3)レベル１の状態遷移 SORT に対応する状態遷移の対応を設計する．例２で述べた順に制御を行うための状態遷移の対応を表５に示す．状態遷移図で表すと図１(b)の形をしている．

(4)レベル１と２との間の検証は，いわゆるクイックソートアルゴリズムを用いた実現であるので省略する．

(5)レベル２の状態に関する述語ISは，初期状態と終了状態でのみ真である．レベル１の状態遷移図（図１(a)）と状態遷移の対応から，図１(b)の制御部が合成される．

(6)図１(b)では START から出射すべき nop は簡約化されている．

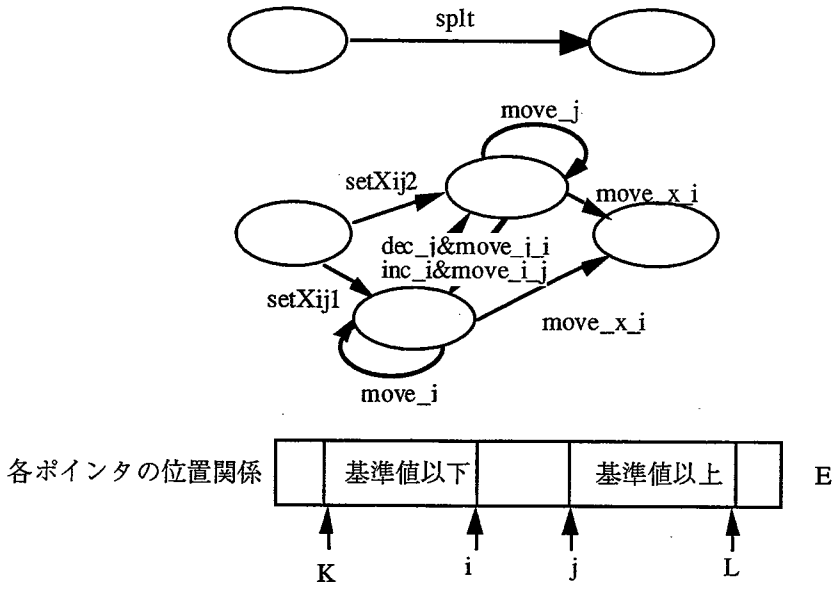


図３　クイックソート回路のレベル２からレベル３への設計における遷移の対応

レベル２からレベル３への設計では，状態遷移 splt を具体化する．用いた遷移の対応を図３に示す．レベル３では，新たに状態成分 X，i，j を導入する，成分 X は分割のための基準値を格納するレジスタである．成分 i，j は，メモリ E のどのアドレスまで分割操作が終わったかを示す E へのポインタを格納するレジスタである．

状態遷移 splt は，レジスタ K と L が指定するメモリ E の要素を，ある基準値より大きいデータと小さいデータとに分割する遷移である．この状態遷移を以下のように設計する．まず，開始状態からの遷移 setXij1(または2)で，i に K （または K+1)の値を，j に L -1 (または L)の値を転送し，分割の基準値（setXij1 ではメモリ E の j 番目の要素，setXij2 ではメモリ E の i 番目の要素）を X に転送する．２つの途中状態における遷移の繰り返し実行で，j 番目のメモリの要素の値（RAM[j]）（あるいはi番目の要素の値）を基準値 X と比較し，j を減らしながら (J←J-1)（あるいは i を増やしながら），基準値より小さい（あるいは大きい）要素を探す．見つかったときは，メモリの要素を入れ換える(dec_j&move_j_i または inc_i&move_i_j). i と j の値が等しくなったとき，基準値を i （または j ）のところに戻し (move_x_i)，分割操作を終了する．このとき，K と i の間の要素は基準値以下であり，j と L の間の要素は基準値以上である．

レベル３以降の設計に関しては，図２にその設計指針を示し，詳細は省略する．レベル４まで設計を行った後，制御部の実現方法及びデータパスの構成にしたがって，２つの実現を行った．以下では，ワイヤード論理方式の設計とマイクロプログラム制御方式の設計と呼ぶことにする．

ワイヤード論理方式の設計においては，段階的な設計を行う上で生じる不要な状態遷移の消去や分岐部分の展開を行う必要がなかった．この設計で得られた回路は，文献[26]のものと同じ有限状態数13，各状態における各動作の実行条件，各動作における動作内容が同じという点で，まったく同じ回路が得られた．

マイクロプログラム制御方式の設計においては，用いるアーキテクチャとして図４のものを用い，各情報をレジスタファイルに格納し，各情報の変更，演算，分岐条件の判定などは，汎用レジスタに一旦取り出してから行うというものであったので，冗長な状態遷移の消去並びに実際に用いるアーキテクチャで制御を実現できるように分岐部分の展開を簡約支援系を用いて行った．この設計で得られた回路は，文献[26]のものと同じデータパス（図４），同じマイクロプログラム（マイクロワードの各項目の意味，マイクロプログラムの内容，総マイクロ命令数66）が同じという点で，まったく同じ回路が得られた．ただし，文献[26]におけるマイクロプログラム中に存在するコードの順番のバグ，ターゲットレジスタの取り違えによるバグなどのない，正しいマイクロプログラムが得られている．

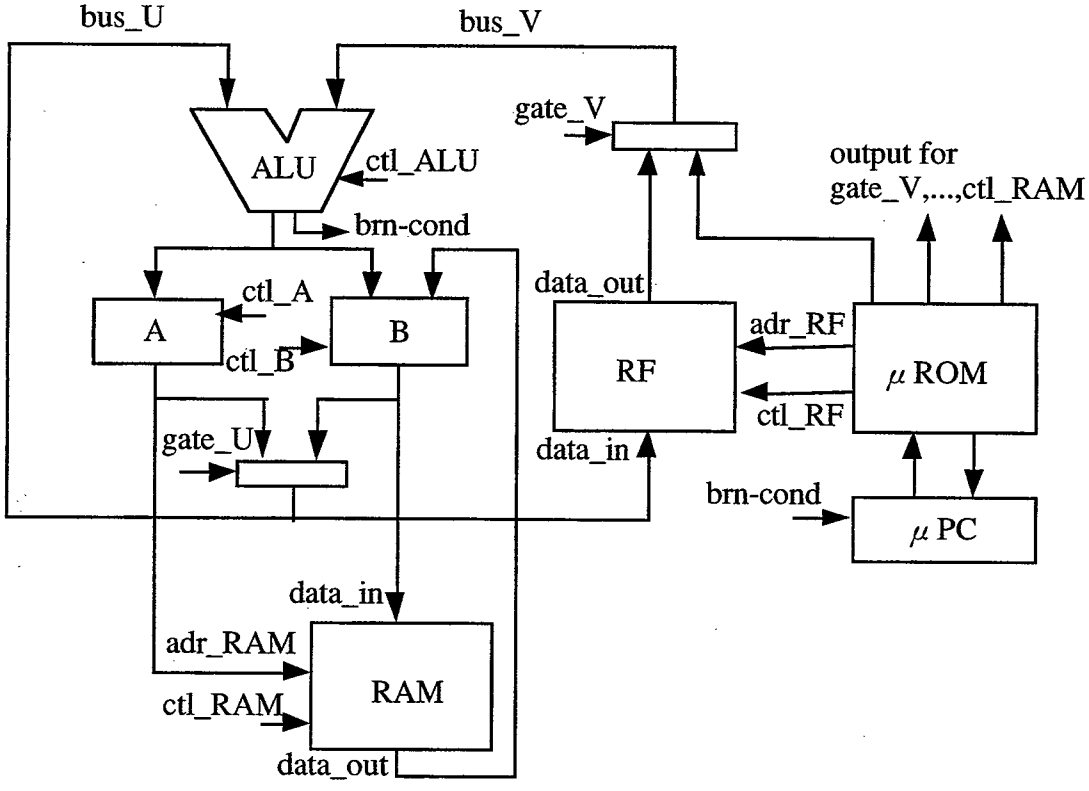


図4　クイックソート回路のマイクロプログラム制御方式アーキテクチャの概要

2.5 結言

本章では，同期式順序回路を記述するモデルとして拡張有限状態機械を用い，それを代数的言語ASLにより記述する方法，抽象レベルで与えられた仕様を段階的に設計する方法を提案し，それを用いた例題回路の設計例について述べた．

提案する手法を用いてマックスソート法によるソート回路[7]，回路検証のベンチマークセットから CPU[9]，GCD 回路[15]などの設計を行った．いずれの回路設計においても，要求仕様を記述する抽象的なレベルから段階的に回路設計を行い，人手で設計するのと同じクオリティの論理設計レベルの回路を設計することができた．これらの設計では，設計者が与えた状態遷移の対応どおりに遷移を展開するのみで，制御部の簡約化を行うことなく設計を行った．

採用した拡張有限状態機械は，制御部が一つのものであるが，一般に設計に用いられる回路では，制御部への配線等の集中をさけるため複数の制御部により実現する場合が多い．また，複数の制御部により実現する方が回路記述言語で記述する際にも自然に記述できる場合も多い．それらの場合，制御部が複数ある拡張有限状態機械を定義し，それを代数的言語で記述することもできる．それらに関する議論あるいは実現の正しさ，状態遷移の対応関係などの拡張は，文献[19-24]で述べられている．

3　提案する設計法に基づく設計支援システム

3.1　序言

本章では，2章で提案した段階的な設計手順のための支援システムの機能，それを用いた評価実験及び支援システムの評価について述べる．設計支援システムは，段階的な設計の履歴の管理，状態遷移の対応を表す状態遷移図の入力，合成により得られた状態遷移図の簡約化などに対し，GUI（グラフィカルユーザインターフェイス）を利用して作業の容易化を目的としている．

3.2では，段階的な回路設計支援システムの機能，特に回路の最適化を行うための状態図簡約化機能について，3.3では支援システムを用いた評価実験について述べる．

3.2　提案する設計法にもとづく段階的回路設計支援システムの概要

2章で提案した代数的手法を用いた段階的な設計手順に対する設計支援システムの機能について述べる．各レベルの仕様は，その制御部の記述に着目すると状態遷移図で表すことができ，テキストで編集するより状態遷移図そのもので編集するほうが，設計が直観的でわかりやすい場合が多い．それらの設計にかかわる部分を，GUIを用いて実現することにする．

また，段階的な設計で得られた実現に対しては，例えば処理に必要とするクロックをもっと速くしたいので途中レベルにおいて設計した動作アルゴリズムを変更したいという場合がしばしばある．そこで途中レベルから動作アルゴリズムの変更や部品の割り当ての変更などの再設計を行なうので，設計全体の管理が必要である．本システムでの階層的設計における設計の履歴を管理させる機能により，途中レベルからの再設計も容易に行なえる．

本システムは，次の4種類の機能を有する．

（1）設計履歴管理機能

段階的な設計の過程を表示する．設計レベルの指定，指定したレベルより下位のレベルの設計の破棄などが行える．

（2）状態遷移図機能

指定されたレベルの仕様に対応する状態遷移図を表示する．有限状態や状態遷移の属性

（名前や分岐条件など）の表示，設計する状態遷移の指定などが行える．

（3）制御部合成機能

指定された状態遷移に対する遷移の対応の設計作業の支援を行なう．有限状態や状態遷移の作成・削除，属性の指定・変更が行える．

（4）状態遷移図簡約化支援機能

状態遷移図の最適化を行うための作業を支援する機能である．

これらの機能は，現在大阪市立大学の松浦敏雄教授らの Key3 Project グループが開発した Xwindow システム上での汎用描画ライブラリを元に，状態遷移図への操作などの機能を追加して作成されている．

(1)から(3)の実現に関しては，文献[11]で述べられている．(4)についての詳細は文献[14]で述べられている．

その他，具体的なレベルからのワイーヤードロジック制御方式における各部品の入力に対する制御論理や制御部自身の制御論理の合成[12]，マイクロプログラム制御方式を採用した場合のマイクロプログラムの合成[13]，具体レベルの記述からハードウエア記述言語への記述の変換などを行う機能も実現されている．

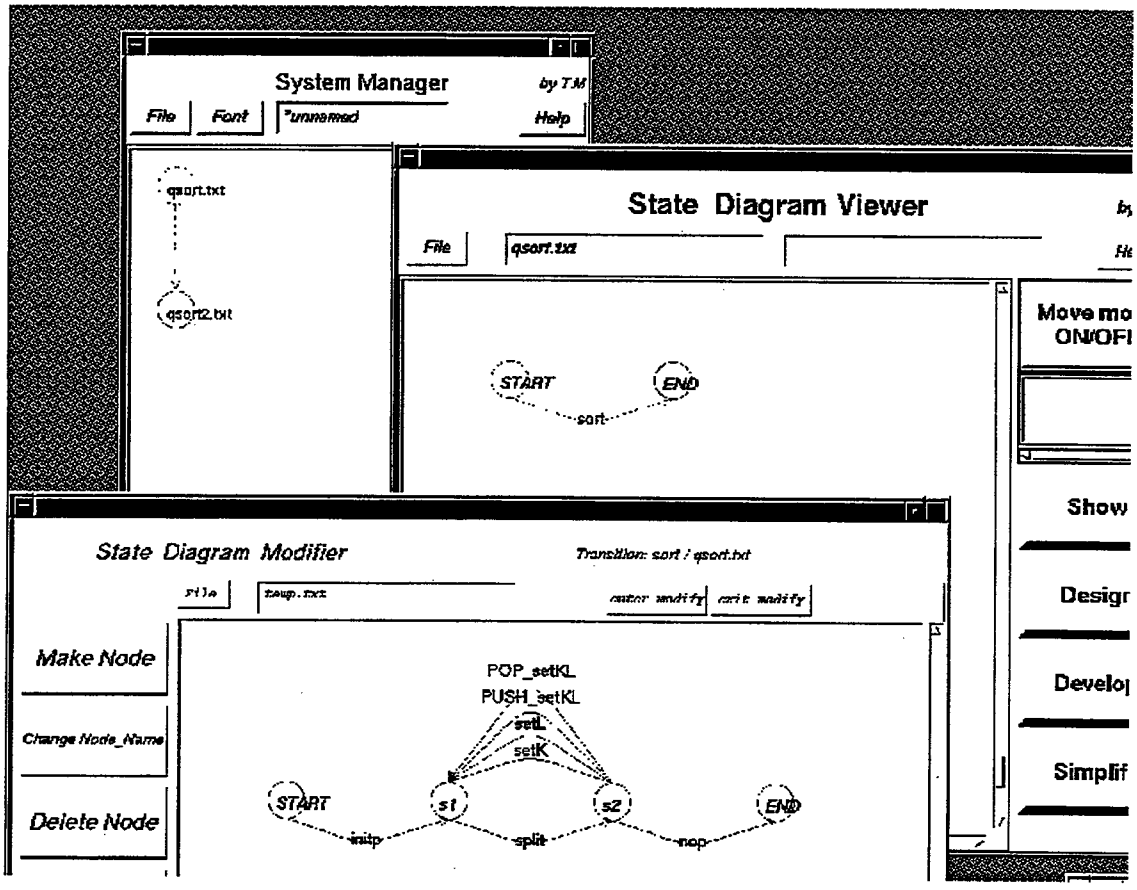


図５　段階的設計のための支援系のGUIの概要

状態遷移図簡約化支援機能

段階的設計によって得られた回路に対する簡約化においては，簡約化の前の制御部（状態遷移図）と簡約化後の制御部とでは，例えば繰り返し実行のためのループ制御構造を変更してしまうような場合，状態間の対応を機械的に付けられない場合が多い．従って，そのような簡約化において，最適化後の回路が元と等価かどうかを証明するには，回路全体としての等価性の証明をせざるを得なくなるが，この証明は一般に難しい．そこで，等価性を保存することが保証されている比較的簡単な状態遷移図簡約ルールを何回か適用し，目的の回路の状態遷移図を得るという方法を採用する．例えば，ある動作において，あるレジスタ値の代わりに別のレジスタ値を用いたいという場合，どのような状況でその動作を行う状態に到達していても，その二つのレジスタの値は等しい（すなわちその関係が不変関係として成り立つ）ということを証明してはじめて適用できるようなルールもある．不変関係の内容は比較的単純であるので，簡約に伴って制御構造が複雑になっても，不変関係の証明はそれほど難しくない．不変関係の証明は，４章で述べる検証手順と同じ方法によって証明できる．

提案する設計法では，例えば抽象レベルにおける記述での動作の分岐部分における実行条件は，メモリの内容や複数のレジスタの値や演算を含む条件式を用いて自由に指定できる．なぜならば，その方が設計の上流工程における主要な設計項目である動作アルゴリズムの考案のみを行うことができるし，設計の正しさの検証においても，分岐条件の判定のため必要なデータの転送や演算のための動作がないので，検証において必要となる不変式が容易に考案できるからである．しかし，アーキテクチャを実現する具体的なレベルでは，設計されて得られた回路仕様における実行条件の判定も，採用されるアーキテクチャのもとで実現されなければならない．このとき，実行条件に用いられている関数・述語を機能部品が実現するプリミティブな関数・述語からなる式で実現し，その式による実行条件の判定のためにレジスタに必要なデータを取り出し，そのデータに対して演算を実行して，条件を判定するというように，分岐部分に対し分岐条件の判定に必要なレジスタ転送を行う状態遷移を挿入し，その判定が行えるように，分岐部分を変形し，その前後の状態遷移系列との最適化も行う必要がある．これらの設計における状態図の変形にも本支援系を用いる．

簡約ルール

等価性を保存したまま状態図を簡約するための，6種類12個の簡約ルールをまとめ，そのルールだけを使って簡約作業を行うことにした．ルールの適用順については，作業者が考案することとした．ここでは，簡約ルールの概要について述べる．

採用するルールの決定においては，各ルールがなるべく簡単で汎用的なものになるように，かつ，ルールを組み合わせて適用することでなるべく作業者の意図通りの簡約操作が行えるようにした．また，それぞれのルールは，適用条件が成り立つもとで適用すれば，適用前と後では状態遷移図の等価性（回路の等価性）は保存されることは保証されている．今回採用した簡約ルールは，それによって行われる操作により，以下のように分類される：

(A) 異なる遷移による置換（遷移の動作内容の追加・変更・削除），

(B) 遷移の分岐条件の変更，

(C)幾つかの遷移の取りまとめ・状態の分割，

(D)レジスタ値を変更しない遷移（nop）や実行条件が恒偽の遷移の削除，

(E)レジスタ値を変更しない遷移（nop）の挿入．

各ルールの詳細は，文献[13]で述べられている．以下に，例として簡約ルール(A)を示す．

(ルールA)レジスタへ転送する値の変更

作業者が指定した遷移 t において，レジスタ reg に式 exp で表される値が転送されるとき，以下の条件のいずれかが成り立てば，式 exp を作業者が与えた式 exp' に変更できる．なお，動作内容が指定されていないレジスタ reg に対して式 exp で表される値が転送させるという動作内容を追加したい場合は無条件に，レジスタ reg に式 exp で表される値が転送されるとき，その動作内容を削除したいときは次の条件のうち(ii)が成り立てば行える．

(i) t の実行後の reg の値を指定する動作内容の公理がない．

(ii) t より後で実行される任意の遷移系列について，reg に別の値が代入されるまでの間，reg の値が参照されない．

(iii) t の実行直前に，不変関係 exp = exp' が成り立つ．

このルールなどを使い，例えば，同時に実行できるデータ転送のとりまとめなどを行える．引き続く二遷移で行われるデータ転送 A←B と B←C を，一遷移にまとめる例を，図 6 に示す．簡約前の状態遷移図の一部を図 6 (a)とする．状態 s1 から出射する枝を取り除くことを目標とする．状態 s1 から出射する状態遷移におけるデータ転送 A←B の内容を状態 s2 から出射する状態遷移で実行させるという方針を立てる．まず，後者の状態遷移でデータ転送 A←Bが実行可能かを判定させ後者の遷移に付け加える（(a)から(b)）．前者の状態遷移からデータ転送 A←B を取り除く（(b)から(c)）．最後に，データ転送を行わないように変更された前者の状態遷移を取り除く（(c)から(d)）．以上により，目的を達することができる．

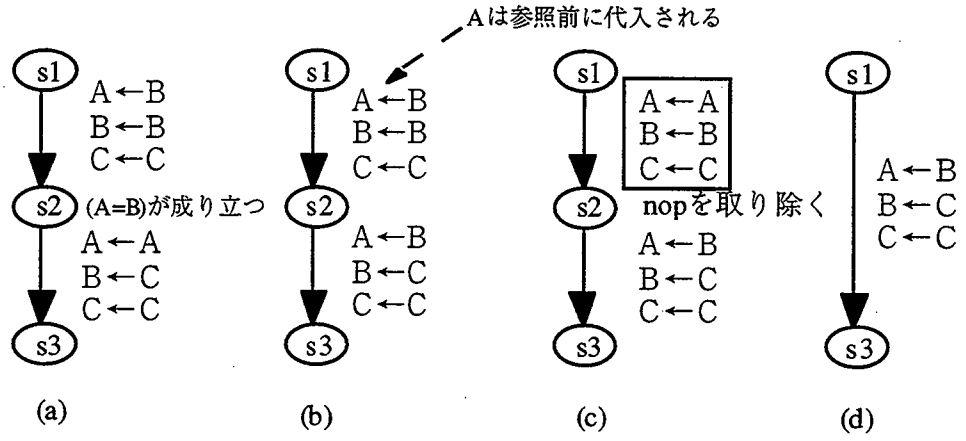


図 6　簡約ルールを用いたデータ転送の取りまとめ

状態図簡約支援のための機能

ここでは，本状態図簡約支援系の主な機能について簡単にまとめる．

・作業者が指定した簡約ルールの適用条件の自動判定と，ルールの実際の適用を行う．各簡約ルールの適用条件は，以下のように分類できる：

(a)状態図が，ルールを適用できる形になっているかどうかの判定.

例えば，上述の例では，図6(c)から(d)での簡約において状態s2に他の遷移が入射する場合実行できない.

(b)ルールを適用しようとする遷移tから後の状態において，指定されたレジスタの値が，次の代入より前に参照されるかどうかの判定.

例えば，ルールAの適用条件の(ii)が該当する.

(c)ルールを適用しようとする遷移tの実行前の状態で，レジスタ間にある条件式（ルールAでのexp = exp'など）が不変関係として成立するかどうかの判定

(a)，(b)については，本支援系が，状態図の各ノードの接続関係，各遷移の動作内容，分岐条件などから，自動で判定する.

ルールの適用条件を表す式が恒真であるか（不変関係として成り立つか）という証明は，4章にて述べる設計の正しさの検証における不変表明が不変式となることの証明と同じ手法を用いて判定する．具体的には，作業者が状態Siに不変表明P_Siを与え，それらが実際に不変式として成り立つことを証明する場合，「回路の初期状態または状態図上の閉路の始点から始まり，Siに到達する任意の遷移系列Tに対して，Tの実行後にP_Siが成り立つこと」を表す式が，各遷移の動作内容の指定，各遷移の実行順の指定，および作業者が与えた基本関数の性質に関する補題（基本関数の定義など）のもとで成り立つことを，4章で述べる検証法と同様の方法により判定する.

・状態図を図示する機能

図7に本支援系の実行画面を示す．簡約作業中の状態図が図示される（各遷移の動作内容や，分岐条件なども表示できる）．作業者は，表示された状態図などをユーザインタフェースとして，使用するルールなどの指定を行える.

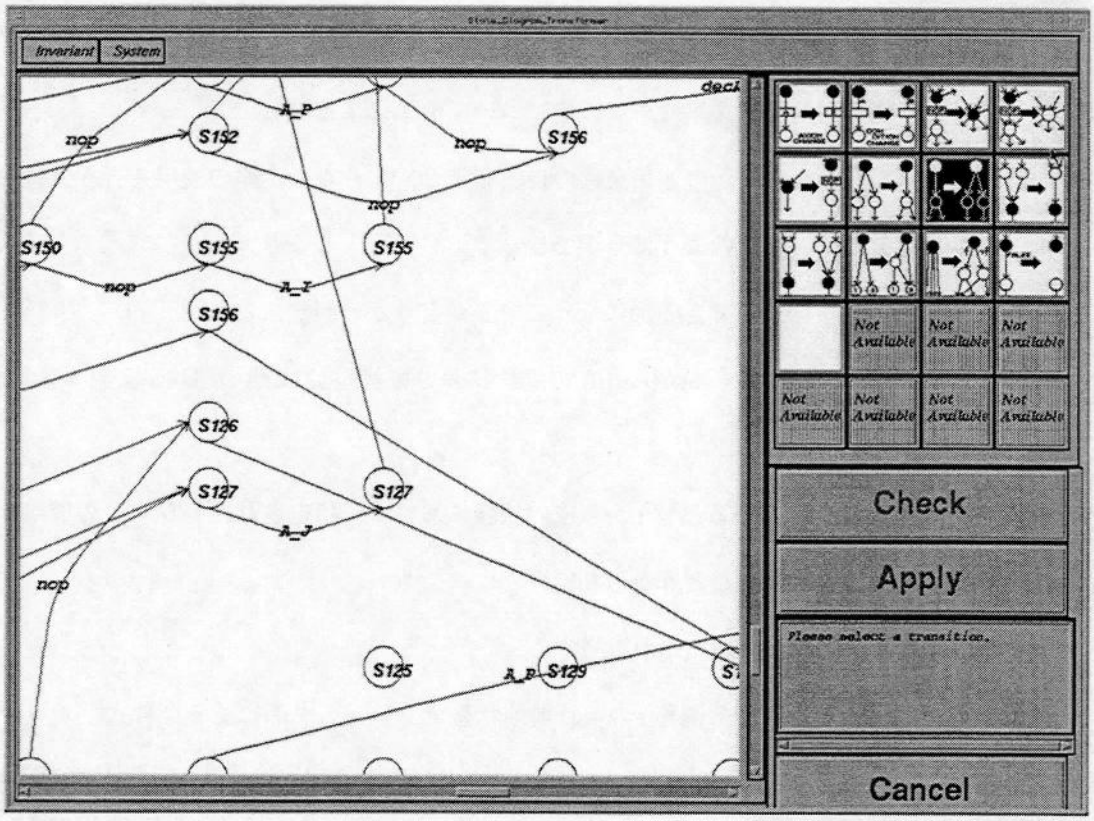


図７　簡約支援系のGUIの様子

3.3　段階的回路設計支援システムを用いた評価実験

３章において述べたソート回路の設計に対して，本支援系を用いて設計作業を行った．

図８にレベル３のソート回路の状態遷移図を示す．図１(a)に示したレベル１の状態遷移図から，２章で述べた設計により得られたものである．

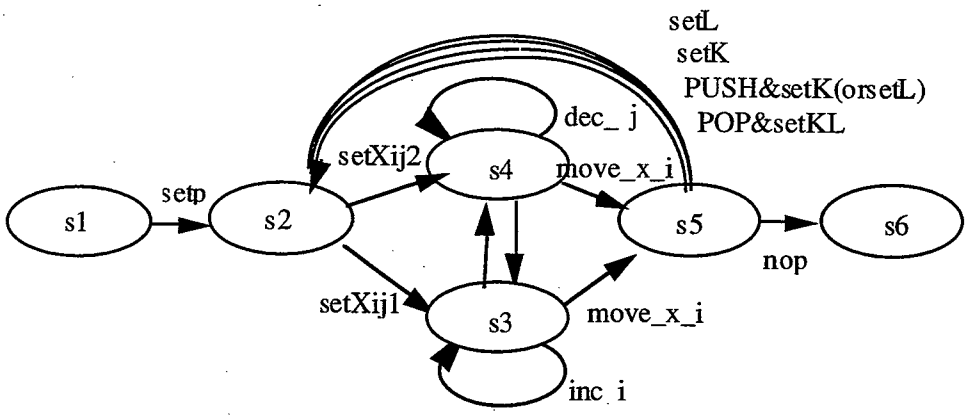


図８　クイックソート回路のレベル３における状態遷移図

ソート回路のレベル３において，状態 s3 及び s4 における処理に着目する．この状態にて実行される処理のうち，j を減らす状態遷移 dec_j（あるいはiを増やす状態遷移 inc_i）の繰り返し実行と，その実行条件であるメモリの j 番目の要素（あるいは i 番目の要素）と基準値 X を比較する等の条件判定を高速に（例えば必要なマイクロコード数が少ないように）処理できれば，この箇所の実行ステップ数が全体のそれに占める割合が大きいので，全体の処理の高速化が行える．また，回路規模を縮小するために，アーキテクチャとして各成分 K，L などのデータを全て１つのレジスタファイルに格納するようなものを採用することにする．そこで，このレベル以降の設計では，図４で示すアーキテクチャを採用する．A，B はインクリメント（+1）あるいはデクリメント（-1）が行えるカウンタである．A の出力は，bus__U に出力することもできるし，ソートすべきデータが格納される RAM のアドレス指定にも用いることができる．B の出力は，bus__U に出力することもできるし RAM の入出力データバッファとしても用いられる．条件分岐における条件の判定はすべて ALU で行われる．K，L などはすべてレジスタファイル RF に格納されている．このアーキテクチャを採用するの

で，図８の各遷移や分岐条件の判定（判定のために必要なデータをRFから取り出す動作も含む）を，個々に展開した場合，例えば状態s3における状態遷移図は図９(a)のようになり，状態s3における分岐部の分岐条件の判定のための遷移実行とjの値を１減らす遷移実行は，合計７つの遷移の実行が必要である．

しかし，このアーキテクチャでは，例えば，状態s3及びs4における自己ループにおいて，レジスタAにj(あるいはi)の値を常に保持しておくことにより，RAMのj番目（あるいはi番目）の要素の取りだしB←RAM[A]と分岐条件の判定A=i（あるいはA=j)，A←A-1（あるいはA←A+1)の実行と分岐条件の判定X<=B（あるいはX>=b)，の２マイクロコードで実現できる図９(b)．このように図９(a)から図９(b)に簡単化することにより，この繰り返しに要するステップ数をかなり減らすことができる．

この例では，図９(a)の状態遷移図と図９(b)では，二つの状態遷移図が等価であること（例えば，繰返しが終了したとき各レジスタが同じ値となっている）の証明は難しいと思われる．さらに，図８における状態s2からs5，そして再びs2（又はs3)，あるいはs3からs5，そして再びs3(又はs2)に至る繰り返しにおいても回路の処理速度を上げるため，その経路中の遷移数を減らしたい．ここでは，s5，s2をいくつかの等価な状態に分け，各系列毎の最適化も行うことにより，それぞれの処理に要するステップ数を減らすこともできる．この部分の簡約化作業の詳細については文献[14]に述べられている．

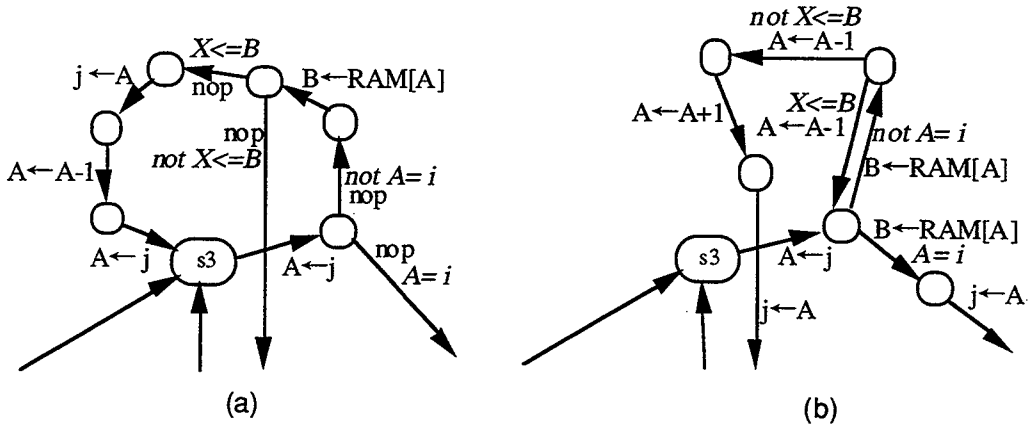


図９　提案する設計法で得られる遷移図と簡約化作業によって得られた遷移図（部分）

ここでは，上述の例における状態 s3 での自己ループにおける最適化について述べる．この最適化は以下の手順で行われる

・A←j を消去する（図１０(a)）．

A←j の動作内容を nop に変更する．得られた nop を消去する．

・s3 を分割する（図１０(b)）．

s3 を３つに分割する．そのうちの２つ（上方から入射する２節点）を併合する．

・A←j を消去する（図１０(c)）．

A←j の動作内容を nop に変更する．このとき，補題として，メモリの j 番地にデータを書き込んで，それ以降メモリにデータを書き込まず，j 番地のデータを読み出すという内容の補題を追加した（以下，このようにメモリの内容に関する補題の追加に関してはその内容を省略する）．次に，得られた nop を消去する．

・j←A をループの出口に追加する（図１０(d)）．

nop を挿入する．その動作内容が「Aの値をメモリのj番地に書き込む」という内容になるように変更する（メモリに関する補題を追加した）．

・ループ内部の j←A を消去する（図１０(e)）．

j←A の動作内容を nop に変更する．次に，得られた nop を消去する．

・条件分岐とレジスタ転送を同時に行うようにする（図１０(f)）．

右側の分岐部分の X<B の nop の動作内容をもつ遷移を A←A-1，A←A+1 と展開する．nop の動作内容をもつ遷移を消去（２箇所）し，nop の動作内容を B←RAM[A] に変更する．

このようにして，１６回の簡約ルールの適用により，状態s3におけるループ部分は自己ループ内の状態遷移数が２つのもの（２マイクロコードで実現できるもの）が得られた．

これらの作業では，まず変形の目標（ループで実行する遷移数を減少させる）を決め，次に上述の概略手順を決め，続いて支援系を用いながらルールの適用順の考案，補題の考案，及び簡約を行った．クイックソート回路のこの例で述べたアーキテクチャを採用した場合の設計では，簡約化作業には数日程度とやや時間がかかったが，適用順の考案自体は特に難しくはなかった．各ルールにおける CPU 時間は，適用条件の判定及び適用の実行にそれぞれ，数秒あるいは一秒以下（SONY　NEWS5000 使用)である．作業時間のほとんどは，支援系を用いた適用順の考案や試行錯誤である．

本手法により得られた回路は要求仕様から段階的に設計を行ったにも関わらず，制御部にワイヤード論理方式あるいはマイクロプログラム制御方式をそれぞれ採用した場合において，文献[5]と，それぞれ同じデータパスのもとで，同じ実行ステップ数の制御部が得られた．文献[26]の人手で直接設計したマイクロプログラムには，例えば，分割の範囲をスタックにプッシュする順番が間違っている，i と j などオペランドレジスタの書き間違いがあるなどの設計誤りが存在する．本手法で得られた回路は，最上位でのクイックソートアルゴリズムの正しさの検証は自明であるとして省略しているが，以降のレベルでの各レベルにおける設計の正しさは検証支援系を用いて証明しており，また簡約化においても簡約前後の回路の等価性を

保証しているルールのみを用いて簡約簡約作業を行っているので，最終的に得られた回路の正しさを保証することができる．このことにより，参考とした回路の設計誤りを発見することができた．

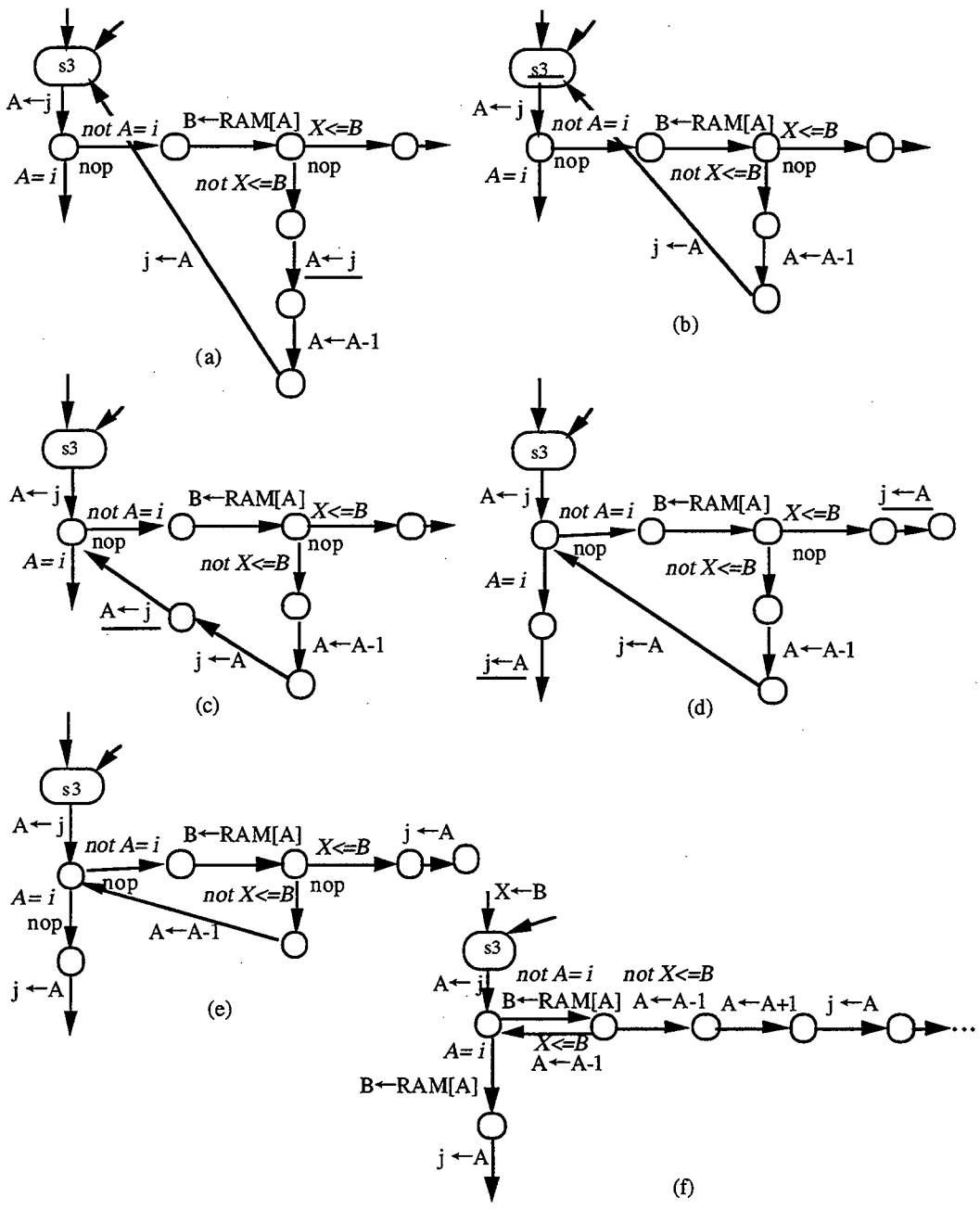


図１０　クイックソート回路の状態図変形の過程（一部）

3.4 結言

本章では，２章で提案した段階的に設計を行うための設計支援システムを用いて，回路記述法に基づく回路の要求仕様から実際に設計を行い，人手で具体レベルで直接設計したのと同じクオリティの回路を自然に設計することができ，また設計の正しさを保証しながら設計が行えることを示し，提案した段階的設計手法の有効性を示した．

例題としては，クイックソート法によるソート回路を用いたが，そのほかマックスソート法によるソート回路およびいくつかの形式的検証のためのベンチマーク回路（簡単なCPU，GCD回路など）においても，同様の結果が得られた．これらの回路については，簡約化作業は不要であった．

4　同期式順序回路設計の正しさの形式的検証

4.1　序言

本章では，2章において定義した状態遷移の対応の正しさを証明するための形式的検証法について述べる．2つの仕様A，B，仕様Bから仕様Aへの状態成分の対応EXPF，仕様Bから仕様Aへの状態遷移の対応EXPTが与えられたとき，以下の3条件を証明する．

（条件1）仕様Aの各公理$\alpha = \beta$に対して，Bの公理，EXPF及びEXPTの公理を用いて$\alpha \approx \beta$であること．

（条件2）上記の各表現式 initi(...) 及び expij(...) などが，値を持つこと．

（条件3）仕様B，仕様Bから仕様Aへの状態成分の対応EXPF，仕様Bから仕様Aへの状態遷移の対応EXPTを合わせたテキストが，無矛盾であること．

（条件2）については，基本定数及び基本関数が値を持つことを前提とすることで，自明な証明となる．（条件3）については，論理設計レベルまで設計が行われ，具体レベルにおける公理の形が定義1を満たしていれば，満たされることが保証されている．

以下4.2では，（条件1）を満たすことの検証において用いる証明技法とそれらを組み合わせた検証手順について述べ，4.3では，提案する検証法に基づいた簡単なCPU，GCD回路，マックスソート法を用いたソート回路の設計の正しさの検証の例について述べる．

4.2　設計の正しさの形式的検証に用いる検証技法とそれらを用いた検証手順

提案する検証法において用いられる証明技法は以下の通りである．

(A)仕様B及び対応を記述する公理を書き換え規則と見なしたときの項書き換え

公理 F(x1,...,xn) == exp(x1,...,xn)を左辺から右辺への書き換え規則と見なして，証明すべき式に現れる部分項 F(p1,...,pn) を exp(p1,...,pn) と書き換える．

(B) if 関数などが用いられているときの条件による場合分け

証明すべき式 F の部分論理式 f(p1,...,pn)に対して，f(p1,...,pn) imply F ∧ not f(p1,...,pn) imply Fを証明する．ここで，α imply β は not $\alpha \vee \beta$である．

(C)プレスブルガー文の真偽判定アルゴリズムの利用

プレスブルガー文とは，整数及び論理変数，∧，∨，¬(not)，＋，－，∀，∃，＝，＜からなり，かつ自由変数がない式のことである．プレスブルガー文は，その真偽が決定可能であり，真偽を決定するアルゴリズムが知られている．証明すべき式には，抽象データタイプを値域とする部分論理式が含まれており，それらを含み値域が整数あるいはBool型となるような部分論理式を整数変数あるいは論理変数と置き換え，さらにそれらの変数を冠頭で全称限定子で束縛した式（プレスブルガー文となる）の式判定を行う．

(D)基本関数の補題の利用

基本関数 p(x1,...,xn) に対して明らかに成り立つべき関係を補題として用いる．例えば，Q(p(x1,...,xn))==TRUE である関係が成り立つとする．証明すべき式を R としたとき，式 Q(p(x1,...,xn)) imply R を証明する．

(E)不変表明を用いた構造的帰納法

Floyd 流の不変表明を用いた証明法である．回路モデルとして採用している拡張有限状態機械に対しては例えば，制御部の有限状態に着目し，指定する有限状態 Si にて成り立つべき不変表明 Q を検証者が考案する．初期状態から Si に到達したとき，常に Q が成り立つこと（不変表明であること）及び，不変表明 Q を前提として証明すべき式が成り立つことを上述の技法(A)～(D)を組み合わせて証明する．

上述の(C)～(E)の方法は，いずれも証明すべき式の十分性を調べていることに相当する．

次に，上述した検証技法を組み合わせた設計の正しさの検証手順について述べる．

状態遷移Tの動作内容 A を証明すべき式とする．下位の回路記述（成分の対応も含むとする）を B，基本関数に関する補題を P，状態遷移の対応を EXPT とする．ただし，A は，Pre imply Post という形をしており，Pre は状態遷移 T の実行前の各状態成分間の関係を表す式であり，Post は状態遷移実行前後の各状態成分間の関係を表す式とする．一般に Preのことを T の事前条件，Post のことを T の事後条件と呼ぶこともある．

（手順１）状態遷移の対応 EXPT において閉路がある場合，対応 EXPT が表す状態遷移図における途中状態における不変表明 INV を検証者が考案する．このとき，初期状態，不変表明が設定された途中状態及び終了状態の間は，閉路を含まないように不変表明を設定する．

（手順２）証明すべき式 A，状態遷移の対応 EXPT 及び不変表明 INV から，Floyd 流の不変表明を用いた構造的帰納法による証明における各証明段階において証明すべき式 IND を求める．証明すべき式 IND は，手順１の不変表明の設定によって，閉路を含まない初期状態，不変表明が設定された途中状態及び終了状態の間の各経路毎に，生成される．証明すべき式 IND は以下のように構成される．

経路の開始状態における不変表明（あるいは証明すべき式Aの前提条件 Pre）

∧　経路上の各状態遷移の動作内容の論理積

∧　経路上の各状態遷移の実行条件の論理積

imply

経路の開始状態における不変表明（あるいは証明すべき式Aの結論 Post）

すべての式 INV が恒真であることが証明されれば，証明すべき式 A も真であることを結論できる．

（手順３）検証者が，証明すべき式 IND に対して必要と思われる基本関数に関する補題 P を考案し，それらを前提とする．

基本関数に関する補題 P

imply

証明すべき式 IND

以下では，この式を IND' と呼び，この式を証明する．

（手順４）式 IND' に対して，下位の回路記述 B の各公理を書き換え規則と見なして書き換え，式 IND" を得る．

（手順５）式 IND" には，抽象データタイプを値域とする部分論理式が含まれており，それらを含み値域が整数あるいは Bool 型となるような部分論理式を新たな整数変数あるいは論理変数と置き換え，さらにそれらすべての変数を冠頭で全称限定子で束縛した式（プレスブルガー文となる）を求め，その式判定を行う．

以上の手順においては，（手順１）における不変表明の考案及び（手順２）における基本

関数に関する補題の考案が検証者が行うべき作業である．これらは，不変表明における各成分間の関係の欠如及び基本関数に関する補題の欠如によって，（手順５）におけるプレスブルガー文の真偽判定で偽と判定される場合（これは設計が正しくないと判定されたわけではなく，正しいことの十分条件が成り立たないと判定されたので），（手順１）における不変表明の再考案や（手順２）における補題の再追加など試行錯誤が必要である．また，それらに伴い，一旦真であると判定された経路に関しても，開始状態あるいは終了状態における不変表明が変更された場合，再度上述の手順により再検証しなければならない．

これらの試行錯誤に伴う煩雑な検証手順に応じた各検証技法の適用，及び検証行程管理を行う支援系については，抽象的順序機械型プログラムの正当性の検証支援系[10]を用いて行うことができる．抽象的順序機械型プログラムとは，本論文で定義した拡張有限状態機械から制御部を取り除き，それに状態遷移の対応を加えたものとみなすことができる．

4.3 提案手法のための形式的検証支援システムとそれを用いた検証

ここでは，TPCD ベンチマークセット[53]のなかから，非パイプライン CPU（Tamarack CPU），最小公倍数を求めるGCD回路，マックスソート法によるソート 回路 の設計の正しさの検証を検証支援システムを用いて行った実験結果について述べる．

TamarackCPU の設計検証

非パイプライン CPU の設計の正しさの検証例について述べる．設計レベルとして，一命令実行を一状態遷移で表すレベル（レベル１）と，メモリアドレスレジスタやメモリデータレジスタなどを導入し，一クロックでそれらの間のデータ転送を行うレベル（レベル２．まだバス構成などは決めていないので，レジスタ間は自由に転送が行える）を考える．

回路記述としてはレベル１，レベル２ともに，動作内容は全てレジスタ転送の形（F(T(S))== exp(S)）で記述されている（表６及び表８）．動作内容及び実行条件に用いられている基本関数は各レベルとも同じものを用いている．図１１のレベル１の状態遷移図において，右端の自己ループが一命令実行を表し，停止命令によって IDLE 状態に遷移する．IDLE 状態では，外部からの入力によって PC，ACC（アキュムレータ），MEM（メモリ）の設定を行う状態遷移が行われる．

設計は，レベル１の各状態遷移を繰り返し構造のない制御によって展開し，レベル２を得ている．このときに用いられた状態遷移の対応における条件分岐の条件には，各レベルで用いられているのと同じ基本関数を用いて指定されている（表７）．

この場合の検証は，以下の手順で自動的に行うことができる．

(1)レベル１の各動作内容 AX に対して，状態遷移の対応の公理で書き換え AX2 をえる．

(2) AX2 を，レベル２の公理で書き換え AX3 を得る．

(3) AX3 に現れるif文の条件で場合分けし，各場合の式 AX4i を得る．

(4) AX4iそれぞれに対し，プレスブルガー文判定ルーチンで真となることを判定する．

一般に基本述語が用いられている場合，それらに関する補題が必要であるが，AX4i に現れる基本関数は，全て同じ引数に対して関数が施されており，基本関数間の関係を用いる必要

がなかったので，上記の手順で自動的に判定が行えた．

検証に必要な計算時間は，SUN classic を用いて93秒であった．

表6　CPUのレベル1の動作内容の記述

```
  (*  成分ACCに対する動作内容の記述  *)
if OP(iget(MEM(s),PC(s))) = ADD then
            ACC(CYCLE(s)) = ACC(s) + iget(MEM(s),ADR(iget(MEM(s),PC(s))))
else if OP(iget(MEM(s),PC(s))) = SUB then
            ACC(CYCLE(s)) = ACC(s) - iget(MEM(s),ADR(iget(MEM(s),PC(s))))
else if OP(iget(MEM(s),PC(s))) = LD then
            ACC(CYCLE(s)) = iget(MEM(s),ADR(iget(MEM(s),PC(s))))
else
            ACC(CYCLE(s)) = ACC(s) == TRUE;

  (*  成分MEMに対する動作内容の記述  *)
if OP(iget(MEM(s),PC(s))) = ST then
            MEM(CYCLE(s)) = iput(MEM(s),ADR(iget(MEM(s),PC(s))),ACC(s))
else
            MEM(CYCLE(s)) = MEM(s) == TRUE;

     :
```

表7　CPUのレベル1とレベル2の対応の記述

```
CYCLE(s)== EXEC(FETCH(s));

EXEC(s) == if OP(IR(s)) = JMP then EXECjmp(s)
else if OP(IR(s)) = JZR then EXECjzr(s)
else if OP(IR(s)) = ADD then EXECadd(s)
else if OP(IR(s)) = SUB then EXECsub(s)
else if OP(IR(s)) = LD then EXECld(s)
else if OP(IR(s)) = ST then EXECst(s)
else nop(s);

FETCH(s) == mem_ir(pc_mar(s));
EXECadd(s)== inc_pc(add(ir_mar(s)));
EXECsub(s)== inc_pc(sub(ir_mar(s)));
EXECst(s)== inc_pc(acc_mem(ir_mar(s)));
EXECld(s)== inc_pc(mem_acc(ir_mar(s)));
EXECjmp(s)==ir_pc(s);
EXECjzr(s)==if ACC(s)= 0 then ir_pc(s) else inc_pc(s);
```

表8　CPUのレベル2の動作内容の記述

```
PC(pc_mar(s))   == PC(s);
ACC(pc_mar(s))  == ACC(s);
MAR(pc_mar(s))  == PC(s);
MEM(pc_mar(s))  == MEM(s);

PC(mem_ir(s))   == PC(s);
ACC(mem_ir(s))  == ACC(s);
IR(mem_ir(s))   == iget(MEM(s),MAR(s));
MEM(mem_ir(s))  == MEM(s);

PC(nop(s))   == PC(s);
ACC(nop(s))  == ACC(s);
MEM(nop(s))  == MEM(s);

PC(ir_mar(s))   == PC(s);
```

```
ACC(ir_mar(s))  == ACC(s);
MAR(ir_mar(s))  == ADR(IR(s));
MEM(ir_mar(s))  == MEM(s);

PC(add(s))   == PC(s);
ACC(add(s))  == ACC(s) + iget(MEM(s),MAR(s));
MEM(add(s))  == MEM(s);

PC(sub(s))   == PC(s);
ACC(sub(s))  == ACC(s)- iget(MEM(s),MAR(s));
MEM(sub(s))  == MEM(s);

PC(acc_mem(s))   == PC(s);
ACC(acc_mem(s))  == ACC(s);
MEM(acc_mem(s))  == iput(MEM(s),MAR(s),ACC(s));

PC(mem_acc(s))   == PC(s);
ACC(mem_acc(s))  == iget(MEM(s),MAR(s));
MEM(mem_acc(s))  == MEM(s);

PC(ir_pc(s))   == ADR(IR(s));
ACC(ir_pc(s))  == ACC(s);
MEM(ir_pc(s))  == MEM(s);

PC(inc_pc(s))   == PC(s)+ 1;
ACC(inc_pc(s))  == ACC(s);
MEM(inc_pc(s))  == MEM(s);
```

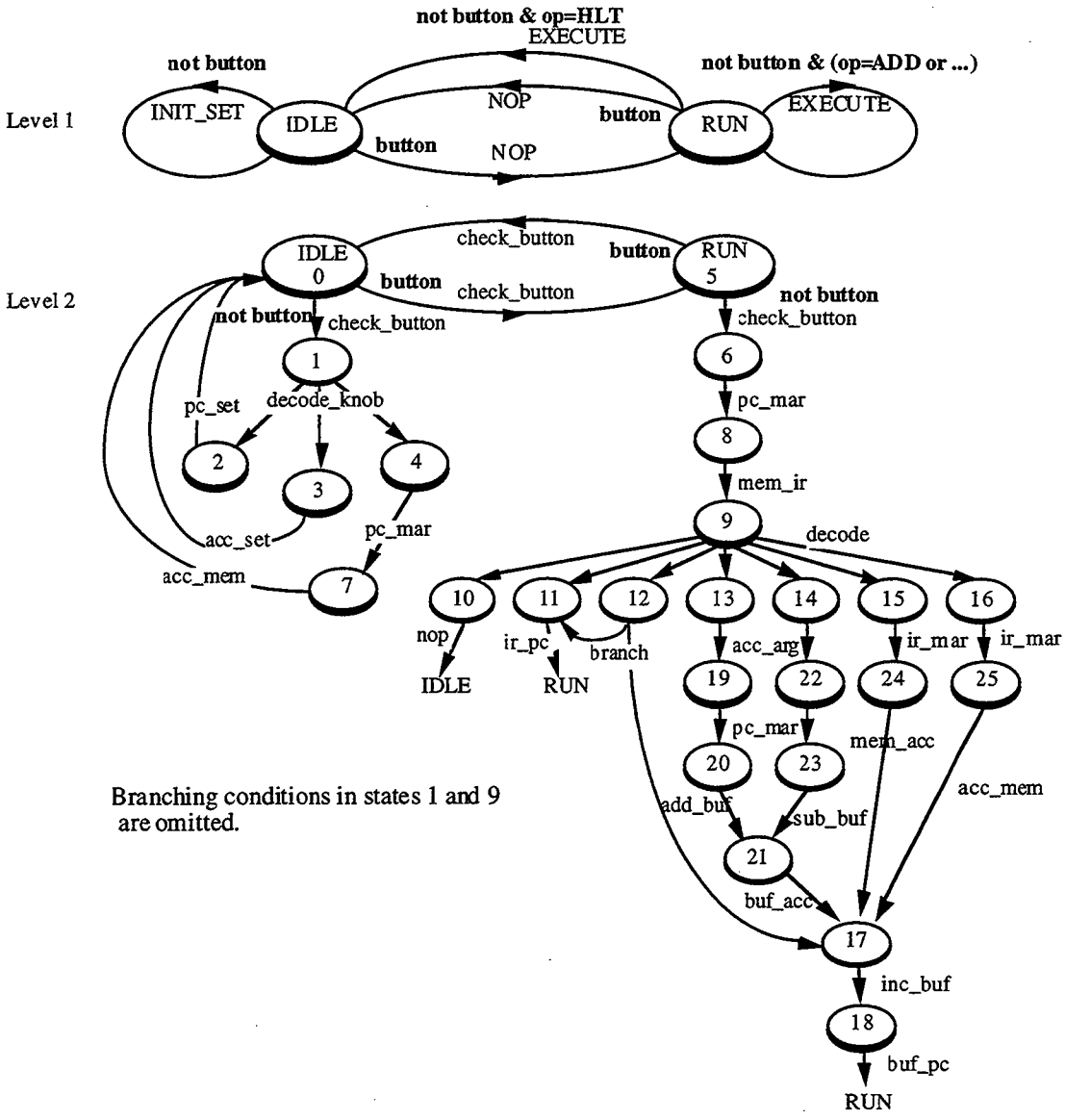


図１１　CPUの設計の概要

GCD回路の設計検証

TPCDベンチマークの検証問題の一つであるGCD回路の設計の正しさの検証を行う．ベンチマークでは，ユークリッドの互除法を用いたアルゴリズム及び，具体的なレベルのデータパス及び2個のD-フリップフロップからなる制御部が与えられている．

これに対し，GCD回路の要求仕様と言うべき入出力関係のみを表すレベルをレベル1とし，ユーグリッドの互除法を実現するのに必要なレジスタを導入し，アルゴリズムを実現する状態遷移図で実現したレベルをレベル2とする．これに対して，割り込み入力を付加したレベルをレベル3とし，アーキテクチャを全て具体的に定めたレベルをレベル4とする．

回路記述としては，レベル1において，要素が整数であるような集合の最大要素を求める関数MaxMember，集合の共通要素を求める関数InterSection，約数の集合を求める関数DivisorSetを組み合わせて動作内容を記述している（表9）．レベル2，レベル3の記述をそれぞれ，表11，表13に示す．レベル1からレベル3までの状態遷移図及びレベル4のアーキテクチャを図12に示す．

レベル1からレベル2の設計検証では，中間状態LOOPにおいて，計算の途中結果を保持しているレジスタA，Bの値の最大公約数が初期状態における値（入力値）の最大公約数と等しいという内容の不変表明（表12）を設定し，基本関数に関する補題（表11後半）を設定して検証を行った．レベル2からレベル3での検証は，リセット動作が付け加えられたのみであり，(1)リセットが入力された場合次の状態で初期状態に到達する，(2)リセットが入力された場合計算が終了したことを示すフラグが立たない，(3)リセットが入力されない場合レベル2と同じ動作をするということを各動作内容や実行条件の公理の形から機械的にチェックできる．レベル3とレベル4の検証は，CPUの設計の正しさの検証の場合と同様に自動的に行える．2章で述べた設計手順では入力論理を合成したが，この例では，具体的に制御部が与えられており，その制御部が正しいことも動作内容の検証と同じように検証した．

検証に必要であった計算時間は，レベル1からレベル2が2秒，レベル3からレベル4が制御部の検証を含め11秒（SUN classic）であった．

表9　GCD回路のレベル１の記述

```
(1 <= A(s) and A(s) <= B(s) and B(s) <= N)
imply GCD(calcgcd(s))
        = MaxMember(InterSection(DivisorSet(A(s)),DivisorSet(B(s)))) == TRUE;
```

表１０　GCD回路のレベル１とレベル２の対応の記述

```
calcgcd(s) ==   state_S1(SetMaxX1andSetMinX2(s));
state_S1(s) ==  if (EqualZero(MOD(X1(s),X2(s)))) then
                        X2toGCD(s)
                else
                        state_S1(X2toX1andSetModX2(s));
```

表１１　GCD回路のレベル２の記述と用いた基本関数に関する性質

```
X1(SetMaxX1andSetMinX2(s)) == B(s);      (* B(s) >= A(s) *)
X2(SetMaxX1andSetMinX2(s)) == A(s);      (* B(s) >= A(s) *)

X1(X2toX1andSetModX2(s)) == X2(s);
X2(X2toX1andSetModX2(s)) == MOD(X1(s),X2(s));

GCD(X2toGCD(s)) == X2(s);

(** SameSet に関する性質 **)
(* init: ns1=InterSection(DivisorSet(A(s0)),DivisorSet(B(s0))) *)

SameSet(ns1,ns1) == TRUE;

(* 推移律 *))
```

```
(* setMod: ns1=I(D(A(s0)),D(B(s0))), ns2=I(D(X2(s)),D(X1(s))), ns3=I(D(MOD(X1(s)
,X2(s))),D(X2(s))) *)
(* setMod: ns1=I(D(X1(s)),D(X2(s))), ns2=I(D(X2(s),D(MOD(X1(s),X2(s)))), ns3=I(D
(MOD(X1(s),X2(s))),D(X2(s))) *)

SameSet(ns1,ns2) and SameSet(ns2,ns3)
imply SameSet(ns1,ns3) == TRUE;

(* setMod: ns1=I(D(X1(s)),D(X2(s))), ns2=I(D(X2(s)),D(X1(s))), ns3=I(D(MOD(X1(s)
,X2(s))),D(X2(s))) *)

SameSet(ns1,ns2) and SameSet(ns1,ns3)
imply SameSet(ns2,ns3) == TRUE;

(** InterSection (と SameSet) に関する性質 **)
(* setMod: ns1=X1(s),ns2=X2(s) *)
(* setMod: ns1=D(X2(s)),ns2=D(MOD(X1(s),X2(s))) *)

SameSet(InterSection(ns1,ns2),InterSection(ns2,ns1)) == TRUE;

(** おなじ公約数のセットなら、その中の最大値も同じ **)
(* X2toGCD: ns1=InterSection(DivisorSet(A(s0)),DivisorSet(B(s0))),
   ns2=InterSection(DivisorSet(X2(s)),DivisorSet(X1(s))) *)

SameSet(ns1,ns2)
imply MaxMember(ns1) = MaxMember(ns2) == TRUE;

(** 数学的に証明されている性質 **)

(** 余りは 0 以上 除数 未満 **)
(* setMod: n1=X1(s),n2=X2(s) *)

(1 <= n2 and n2 <= n1 and n1 <= N)
imply (0 <= MOD(n1,n2) and MOD(n1,n2) <= n2 - 1) == TRUE;

(* setMod: n1=X1(s),n2=X2(s) *)
(1 <= n2 and n2 <= n1 and n1 <= N)
imply (0 <= MOD(n1,n2) and MOD(n1,n2) <= n2 - 1) == TRUE;

(1 <= n2 and n2 <= n1 and n1 <= N and (not EqualZero(MOD(n1,n2))))
imply 1 <= MOD(n1,n2) == TRUE;

(** n1≦n2 かつ n1 は n2 の約数 → GCD(n1,n2) = n1 **)
```

```
(* X2toGCD: n1=X2(s),n2=X1(s) *)

(1 <= n1 and n1 <= n2 and n2 <= N and EqualZero(MOD(n2,n1)))
imply MaxMember(InterSection(DivisorSet(n1),DivisorSet(n2))) = n1 == TRUE;

(** n2とn1の公約数の集合 = n1と(n2 mod n1)の公約数の集合 **)
(* setMod: n1=X2(s),n2=X1(s) *)

(1 <= n1 and n1 <= n2 and n2 <= N and not EqualZero(MOD(n2,n1)))
imply SameSet(InterSection(DivisorSet(n2), DivisorSet(n1)),
              InterSection(DivisorSet(n1), DivisorSet(MOD(n2,n1)))) == TRUE;
```

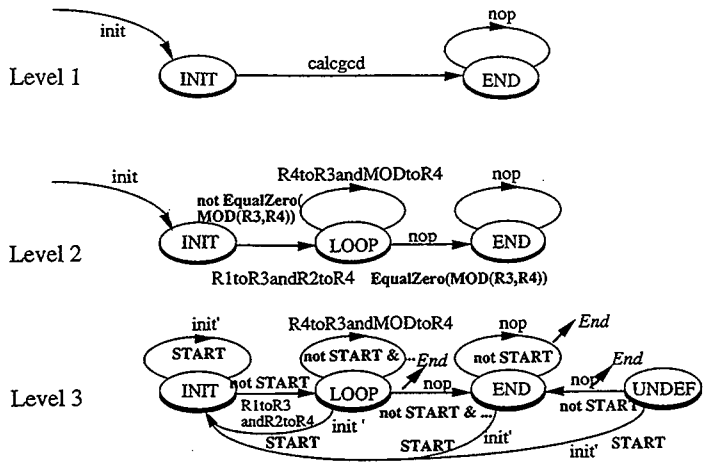
表１２　GCD回路のレベル１からレベル２への設計の正しさの検証で用いた不変式

```
1 <= R3(s) <= R1(S0) and
1 <= R4(s) <= R2(S0) and
R4(s) <= R3(s) and
SameSet(Intersection(DivisorSet(R1(S0)),DivisorSet(R2(S0))),
              (Intersection(DivisorSet(R3(s))  ,DivisorSet(R4(s))))
```

表１３　GCD回路のレベル４およびレベル３と４との対応の記述

```
(*D-FFの仕様*)
D_FF(CK_d(d_s,d_in)) == d_in;
QZ(d_s) == NOT(D_FF(d_s));
Q(d_s) == D_FF(d_s);

(*  データパスの定義　各D-FFの入力結線の指定*)
define 'minimaxAB' := 'MinMax(A,B)';

define 'EqualZero' := 'EQ0(outMod)';
define 'outMod' := 'Mod(REG(proj_3(s)),REG(proj_4(s)))';

CK(s ,START, A, B, StoreVergl,StoreLoop,SelectLoop)
```

```
==   [ CK_r(proj_1(s),StoreVergl,Mul(A,B,minimaxAB))
       CK_r(proj_2(s),StoreVergl,Mul(B,A,minimaxAB))
       CK_r(proj_3(s),StoreLoop ,Mul(REG(proj_4(s)),REG(proj_1(s)),SelectLoop ))
       CK_r(proj_4(s),StoreLoop ,Mul(outMod , REG(proj_2(s)),SelectLoop))];

(*データパスに関する状態遷移の対応*)
                                    (* StoreVergl,StoreLoop,SelectLoop *)
init (s,START,A,B) == CK(s,START,A,B, TRUE    , undef   , undef    );
set34(s,START,A,B) == CK(s,START,A,B, FALSE   , TRUE    , TRUE   );
div(s,START,A,B)   == CK(s,START,A,B, undef   , TRUE    , FALSE   );
nop(s,START,A,B)   == CK(s,START,A,B, undef   , FALSE   , undef   );


(*制御部の記述*)
 (*制御部の状態遷移の記述  制御部の結線の記述*)
CK_c(s,START,eq0) =
        [ CK_d(proj_5(s) , p29)
          CK_d(proj_6(s) , p32)];

define 'p37' := 'BUF(START)';
define 'p32' := 'OR( p49, AND3(p35,p36,QZ(proj_5(s))))';
define 'p49' := 'AND3(p35,QZ(proj_5(s)),QZ(proj_6(s)))';
define 'p29' := 'OR(AND(p35,Q(proj_5(s))), AND3(p35,eq0,Q(proj_6(s))))';
define 'p35' := 'NOT(START)';
define 'p36' := 'NOT(eq0)';

(* 制御部の出力信号の定義*)
StoreVergl(s,START,eq0) == BUF( START ) ;
StoreLoop(s,START,eq0) ==  p32 ;
SelectLoop(s,START,eq0) == p49 ;
End_imp(s,START,eq0) == p29;

(*  制御部の状態遷移の対応  *)
init (s,START,A,B) == CK_c(s,START,EqualZero);
set34(s,START,A,B) == CK_c(s,START,EqualZero);
div(s,START,A,B)   == CK_c(s,START,EqualZero);
nop(s,START,A,B)   == CK_c(s,START,EqualZero);

End(s,START,A,B) == End_imp(s,START,EqualZero);

define 'EqualZero' := 'EQ0(outMod)';
define 'outMod' := 'Mod(REG(proj_3(s)),REG(proj_4(s)))';
```



Level4 (データパス＋２個のD-FFによる制御部)

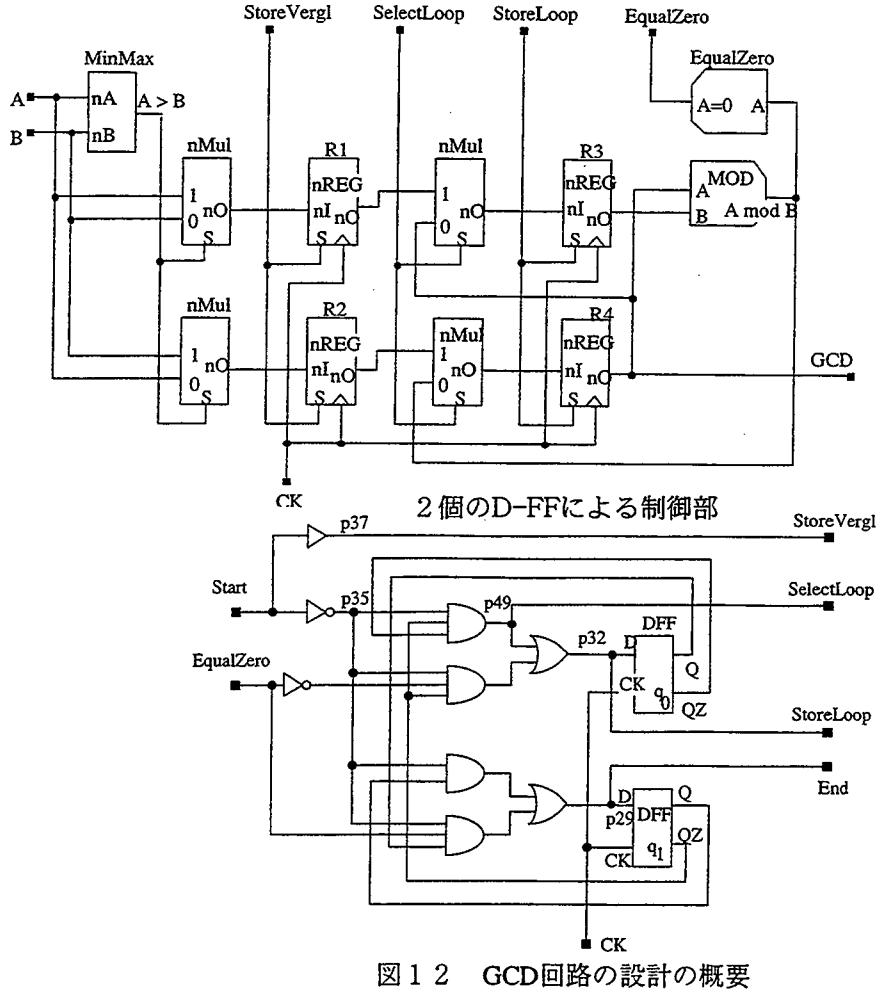


図１２　GCD回路の設計の概要

マックスソート法によるソート回路の設計検証

２章及び３章で示したソート回路の別の実現も行った．

ここでは，マックスソート法を採用した設計と設計の正しさの検証について述べる．レベル１の仕様はクイックソート法による設計のときと同じものである．レベル２では，メモリ内から最大値を保持する箇所を見つける遷移と，その場所の要素と未ソートの最右端の場所の要素を入れ換える遷移を導入して実現するレベル，レベル３はメモリ内の最大値を左端から一つづつ探索するアルゴリズムを実現したレベル，レベル４は，初期値の設定や要素の入れ換えなどが実際のアーキテクチャで実行できるように具体化したレベル，レベル５は具体的にアーキテクチャを定めマイクロプログラムを合成するレベルである．各レベルの状態遷移図及び用いたアーキテクチャを図１３に示す．

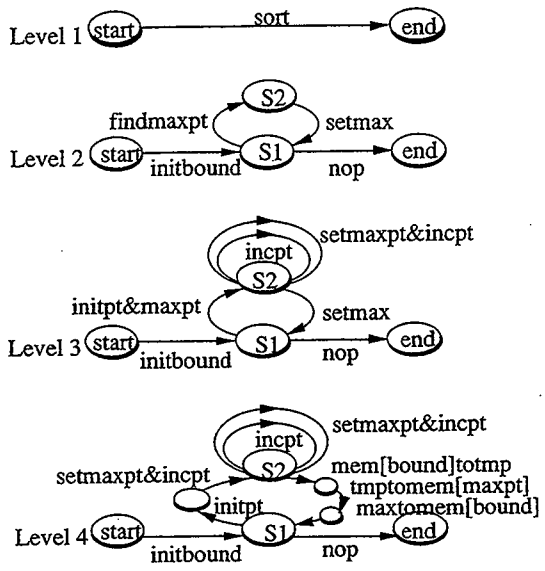
レベル１からレベル２への設計検証では，メモリのどこの部分までソートが終了しているかなどを表した述語を用いて不変表明を与え，それらの述語やメモリに関する補題（書き込んだところから読み取った場合，書き込まれたデータが取れ出されるなど）を用いて検証を行った．レベル２からレベル３も不変表明が必要であったが，それ以降の検証は自動的に行うことができた．各レベルの検証に要した計算時間は以下の通りである．

レベル１からレベル２　107秒

レベル２からレベル３　20秒

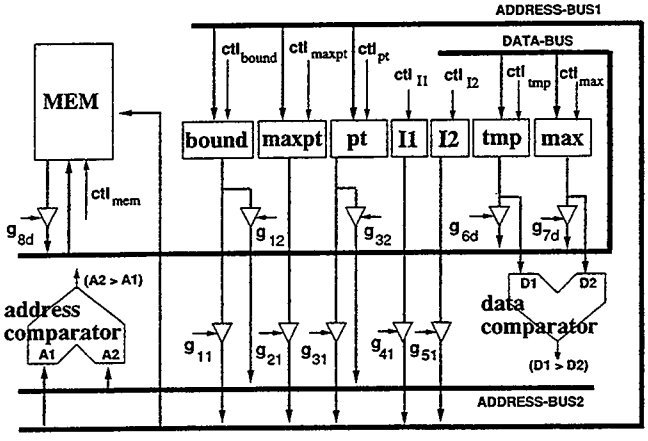
レベル３からレベル４　17秒

レベル４からレベル５　9秒



Level5 (データパス +マイクロプログラムによる制御部

データパス



マイクロプログラムによる制御部

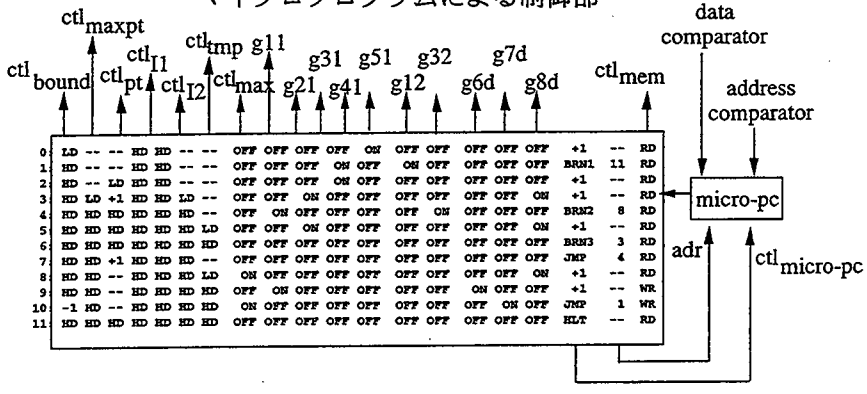


図１３　マックスソート回路の設計の概要

4.4 結言

本章では，設計の正しさを保証するための形式的な検証方法について述べ，その方法に基づきいくつかの設計例題に対する検証実験の結果について述べた．

一般には証明の難しい，記述レベルの異なる２つの同期式順序回路の等価性の判定問題を，状態成分の対応及び状態遷移の対応，そして必要ならば，不変表明を用いた構造的帰納法を利用することによって，元の等価性の問題を帰納法の各段階の証明であるより具体的な複数の部分問題に帰着しそれぞれの部分問題を解くことによって，もとの証明すべき問題を証明している．これらの部分問題はさらにプレスブルガー文真偽判定問題に帰着される．

このように定理証明系を設計の上流工程における回路検証に適用した事例はいくつか報告されている[28,29,49,55]．例えば，文献49における検証システムPVSにおける手法は，ここで述べた方法によく似ている．例えば，項書き換え，場合分け，プレスブルガー文のサブクラスに対する決定手続きの利用などの検証技法を用いている．この検証方法と比べると，今回提案した検証手法は，２章で述べた設計方法に基づき検証手順を定式化し，その手順の中で検証者が行うべき作業を不変表明と補題の考案のみに限定するなど検証作業を単純化しているので，証明手順の考案や多くのコマンドの操作などの煩わしさを排除している．

また，文献[10]及び[46]では，プレスブルガー文に対する真偽判定ルーチンの高速化を実現しており，項書き換えや場合分けの機能を使わなくとも，それらの機能を真偽判定ルーチンに組み込んで，より高速な式判定が行えることを示している．項書き換えに関する組み込みは，公理 $\alpha == \beta$ に対して，両辺に現れる変数に適当な式を代入し $\alpha' = \beta'$ という論理式を生成し，証明すべき式Fに対し，$\alpha' = \beta'$ imply F の真偽を判定する．場合分けについては，場合分けを生じさせる if 関数に関する処理を判定アルゴリズム内に組み込むことによって対処している．

提案する検証法において，必要とされる計算時間のほとんどは，プレスブルガー文の真偽判定に要する時間である．これについては従来，文献[10]及び[46]で実現された判定系（以下従来の判定系と呼ぶ）を用いて検証を行ってきた．しかし，証明すべき検証問題がいくつかの部分問題に帰着されるにしても，不変表明の長さや補題の数が大きくなるような問題につ

いては，判定系に入力される式長が大きくなるので，部分問題一つの判定に多大な計算時間が必要となる．そこで，５章においてプレスブルガー文に対する高速な真偽判定を実現するためのデータ構造とその上での処理系を提案する．

5　検証に用いるプレスブルガー文真偽判定のための一高速化手法

5.1　序言

本章では，4章において述べた回路の設計の正しさの形式的検証に用いられる検証機能の一つであるプレスブルガー文真偽判定手続きのための新たなデータ構造とその上でのデータ処理アルゴリズムとその実装について述べる．

プレスブルガー文[18]は，整数及び論理変数，∧，∨，¬(not)，＋，－，∀，∃，＝，＜からなり，なおかつすべての変数が全称限定子∀あるいは存在限定子∃によって束縛されている論理式である．プレスブルガー文は真偽判定が可能である．真偽判定アルゴリズムは，Cooper のアルゴリズム[34]や Fourier-Motzkin の消去法に基づくアルゴリズム[37]などが知られている．しかし，いずれの場合においてもプレスブルガー文の真偽判定は極めて計算量を必要とするので，プレスブルガー文のサブクラスに対する効率的なアルゴリズムやそのサブクラスの実用への応用に関する研究[36, 46, 47, 52, 50]が数多く行われている．

新たに提案するデータ構造は，論理設計レベルの設計及び検証に用いられているBDDs[5]を利用している．BDDs は，論理関数を比較的コンパクトに表現することができ，論理関数を表すデータ構造に対する処理も高速に行えることから，論理合成及び形式的検証システムに取り入れられている．近年の回路の設計対象の大規模化などの理由から，扱うことのできる論理式のクラスを拡張し，取り得る値は有限ではあるが整数変数を取り扱うデータ構造であるBMDs（Binary Monent Diagrams)に対する提案も行われている[39].

4章では，回路の検証のために利用する検証手法の一つとしてプレスブルガー文という整数上の制約論理に対する恒真性判定ルーチンを採用することを述べた．森岡らによって，プレスブルガー文のサブクラスに対する効率的なアルゴリズム[46]が提案されているが，ここでは，プレスブルガー文を真に含む，より大きなクラスの論理関数を表現するデータタイプを提案し，その上での効率的なアルゴリズム[17,45]を提案する．本手法の評価のために森岡の判定系との比較を行った．大規模な組み合わせ回路の検証の判定で20%高速に判定が行える場合もあることがわかった．

以降，5.2において，新たに提案するデータ構造とそれを用いた判定系の実現について，

5.3では，本手法と従来手法の判定系の比較実験について述べる．

5.2 プレスブルガー文の表現のためのデータ構造とその上での処理方法

本論文で取り扱う論理関数のクラスは，整数及び論理変数，∧，∨，¬(not)，＋，－，∀，∃，＝，＜からなる．このクラスに対して，すべての変数が全称限定子∀あるいは存在限定子∃で束縛されているものをプレスブルガー文という．このクラスにおいては，限定子に対する処理アルゴリズム[34]が知られており，提案するデータ構造はこの手続きを高速に処理し，その結果を比較的コンパクトに表現することができる．

まず，整数データを含む論理式の一つのクラス（拡張プレスブルガー関数，以下P関数と呼ぶ）について述べる．

定義5　P関数

P関数は，以下の構文で定義される．構文にて用いられる記号集合は{true，false，∧，∨，¬(not)，（，），＋，－，∀，∃，＝，＜，0，1，...，x1，...，y1，...｝である．ここで，x1，...は整数変数，y1，...は論理変数とする．

項τは以下の構文のみで定義される．

(1-1) τ ::＝ x1：...：0：1：...

(1-2) τ ::＝(τ＋τ)：(-τ)

P関数Fは以下の構文のみで定義される．

(2-1) F ::＝ y1：...：true：false

(2-2) F ::＝ τ ＝ τ ：τ ＜ τ ：n ｜ τ

(2-3) F ::＝(F∧F) ：(F∨F) ：¬F

(2-4) F ::＝∀x F ：∃x F　□

簡単のため，自明な（）を省略したり，x＋xを2xと表す．また，∃w（w＋w＋...＋w＝X）（ただしwの個数をn）をn｜Xと表現する．n｜Xは式Xが整数定数nで割り切れることを意味する．

P 関数において全称記号∀及び存在記号∃で束縛される変数を束縛変数，そうでない変数を自由変数という．自由変数を含まないP関数は，プレスブルガー文である．プレスブルガー文の真偽判定問題は決定可能であることが知られて，Cooper のアルゴリズムやFourier-Motzkin の消去法に基づくアルゴリズムが知られている．提案する手法では，Cooper のアルゴリズムを実装のために，適したデータ構造を採用する．

もともとのプレスブルガー文には，論理変数は無いが，自由変数を含まないP関数とプレスブルガー文とは，表現能力という点において等価なクラスである．

採用するデータ構造を，P 関数とそれに相当するグラフを用いて説明する．

（関数１）$F(X1,X2) = 2 < x1 + x2 \vee 2\,x2 < 3 \vee \neg\, x1 - 2\,x2 < 0$

関数 F は P 関数である．この関数は，提案するデータ構造を用いると図１４のように表現される．

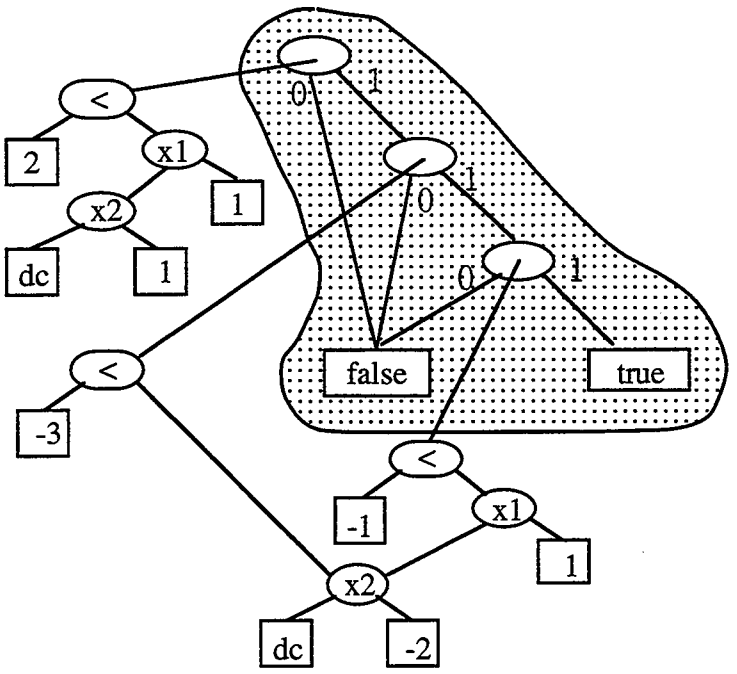


図１４　提案するデータ構造の例

関数Ｆは，３つの整数部分2 < x1 + x2，2 x2 < 3 ，¬ x1 - 2 x2 < 0 と，論理部分 a1∨a2∨a3 からなると見なせる．論理部分に関する構成方法は BDDs と同じであり，図１４の網線で囲まれた部分である．整数部分 a1，a2，a3 に対応する論理変数を表す内部ノードは，三つの子ノードへのポインタをもつ．整数部分は，整数変数の線型リストと整数定数とで実現され，根が＝，＜，｜のいずれかであることを表している．線型リストの各内部頂点は整数変数を表しており，右枝がその変数の係数との乗算を，左枝が加算を表している．根が＝あるいは＜の時は，線型リストにおいて左枝を最後までたどって到達する頂点はdc(ドントケア)となり，根が｜のときはこの部分にて整数定数を保持している．根が＝あるいは<のとき，根ノードの左側の子ノードが整数定数を保持し，根が｜のときは，保持すべきn｜mの整数定

数 n を保持している．後者の構造を採用した理由は，全称限定子あるいは存在限定子に対する処理の実行により，整数定数のみが異なる整数部分が大量に生成されるからであり，この構造を採用することにより，変数部分の線型リストを共有することができるからである．

提案するデータ構造において，同じ構文の部分式は同じ部分グラフとして共有される．図１４では，式 $2x_2$ が共有されている．図１４では，見やすさのため整数定数の終端ノードは共有化されていないように描いている．

データ構造は，入力となる P 関数の構文解析に従って，Bottom-up 的に構築される．ここでは，存在限定子に対するアルゴリズムの概略についてのみ述べる．その他の構文に対する処理は，上述のデータ構造から自明であるので省略する．存在限定子に対するアルゴリズムは，論理式∃x F(x) が与えられたとき，F(x) に対応するデータ構造と論理的に等価で変数xを含まない論理式に対応するデータ構造に変換するものである．

BDDs 及び BMDs の統合ライブラリである文献[41]の BXD ライブラリを拡張し，各構文に対するデータ構造を構築する処理ためのライブラリを作成した．構成されるグラフにおいて，論理変数の順位をすべての整数変数より上位と定めている．また論理変数間，整数変数間の順位は，入力の P 関数の構文解析で表れた順としている．グラフの様々な種類のノードは，BDDs における手法と同様に，共通のハッシュテーブルに登録・管理し，共通部分項の共有化を行っている．

限定子∃の処理アルゴリズムについて説明する．全称限定子は，¬，∃，¬の順にそれぞれに対応する処理アルゴリズムを実行することで実現される．存在限定子に対する処理アルゴリズムはプレスブルガー文の真偽判定アルゴリズムとして知られている Cooper のアルゴリズムに基づいたものである．

F を変数 x を含む P 関数とする．以下では ∃xF(x) を変数 x を含まない∃xF(x) と論理的に等価な関数に置換する手順を述べる．∃xF(x) に対して，F(x) が真となる可能性のある整数関数を F(x) から抜きだし，それぞれを x に代入し，代入されたものの論理和を求めるというアルゴリズムである．

F(x) に対応するデータ構造 D が与えられる．

(Step A)変数 x の係数を最小公倍数でそろえる

D における x の全ての係数を求め，それらの最小公倍数 H を求める．D における x の全ての係数が H となるように変数 x を含む式に適当な数をかける．このデータ構造を D' とする．D' 中の Hx を X と置き換え，それと H¦X との論理積を施したものを D" とする．D" が表す関数 F"(X) と F(x) は論理的に等価である．

(Step B)変数 X に代入する候補を求める

D" における，X を含む整数部分を表現する論理関数を求める．それらの論理関数は以下の形のものである．

$(\alpha < X)$，$(\beta < -X)$，$(X=\gamma)$，$(X \mid \delta)$，$\neg(\varepsilon < X)$，$\neg(\zeta < -X)$，$\neg(X=\eta)$，$\neg(X \mid \theta)$

簡単のため，α などギリシャ文字は X 以外の変数，整数定数からなる式である．

(StepC)代入されたすべての関数の論理和を求める

あらかじめ，δ，θ の最小公倍数を求め N とする．

以下の式(1)～(4)の論理和を求める．

(1)D"の$(\alpha < X)$，$\neg(\zeta < -X)$を真に，$(\beta < -X)$，$\neg(\varepsilon < X)$，$(X=\gamma)$，$\neg(X=\eta)$の項を偽に置き換えたデータ構造D'"を求め，そのデータ構造に現れるX（項$(X \mid \delta)$と項$\neg(X \mid \theta)$に現れる）にそれぞれ1～Nの値を代入したデータ構造D'"1，...，D'"N

(2)D"のXに，ε,...,ε - (N -1) をそれぞれ代入したデータ構造Dε 1，...，Dε N

(3)D"のXに，$-\beta$ - 1-1,...,$-\beta$ -1 -N をそれぞれ代入したデータ構造Dβ 1，...，Dβ N

(4)D"のXに，γ をそれぞれ代入したデータ構造Dγ

(5)D"のXに，η -1...，η --1-N をそれぞれ代入したデータ構造Dη

これら(1)，(2)，(3)，(5)において生成されるデータ構造には，代入により X を消去された各整数部分はそれぞれ整数定数部分のみが異なるものが大量に含まれる．提案するデータ構造では，整数定数部分と整数変数からなる線型リストを分離した構造にしているので，大量に整数定数部分のみが異なる項を生成したとしても整数変数からなるリストは全て共有される．

5.3 実現したプレスブルガー文真偽判定系の評価実験

提案手法の評価のための検証実験を，以下で述べる２つの組み合わせ回路の検証から用いて行う．４bitALU（TTL74382）とHLSynth95ベンチマークセットのfp_addの一部の検証である．ここで，比較実験に組み合わせ回路の検証を用いたのは，４章で述べたように，提案手法における検証では不変式を用いて検証問題をいくつかの部分問題に分割すること，簡単な回路の検証であっても用いる基本関数に対する補題が数多く用いないと検証できない場合があることなどから，回路規模とそれに対する検証の難しさが対応しない場合があることからである．

４bitALUの検証について述べる．入出力が整数変数で記述された要求仕様と，それをFPGA用のCADで論理合成した結果から得られた入出力がすべて論理変数で記述されるレベルである実現仕様との出力等価を検証する．

図１５が要求仕様の一部である．整数変数はプリフィクスとして$がつけられている．機能選択入力[S0,S1,S2]に従って個別の機能が選択され，例えば[F,T,F]の時は，データ入力$Aと$Bの減算が実行され，$Fspecに整数の結果が，$iCn4specにキャリーアウトが出力される．図１６は，実現仕様の一部である．imply，if then，if then elseは，P関数を構成する各関数の複合関数として定義される．

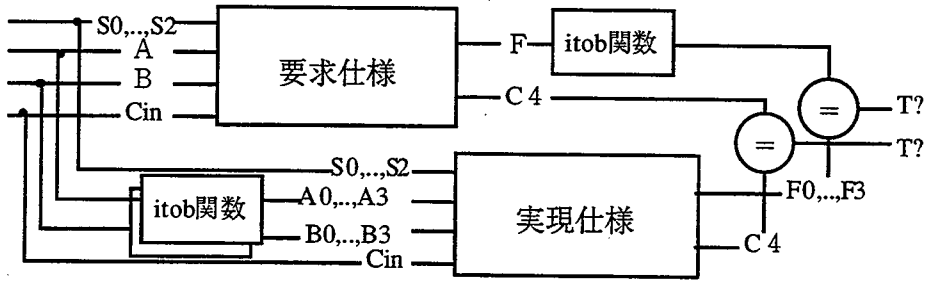
図１７は，FPGA用CADソフトを用いてTTL-IC 74382の仕様を論理合成して得られたネットリストを，内部頂点も含めて論理関数に変換した結果である．データ出力Fの最下位ビットF0が，その他の内部頂点(プリフィクスが_LC の変数）や機能選択入力[S0,S1,S2]の論理関数として表されている．これらの要求仕様と実現仕様が，各変数の組み合わせに対して，すべての出力が同じであることを検証する．図１７に示す２つの方法を用いた．

```
|

  :
∧
 ((¬ S2 ∧ S1 ∧ ¬ S0) imply (
       if $iCn = 1 then (
            (if $A - $B >= 0 then (
                 ($Fspec = $A - $B) ∧
                 ( $iCn4spec = 1))
             else (
                 ( $Fspec = $A - $B + 16) ∧
                 ($iCn4spec = 0)))
       else
    :
   )
 )
)
```

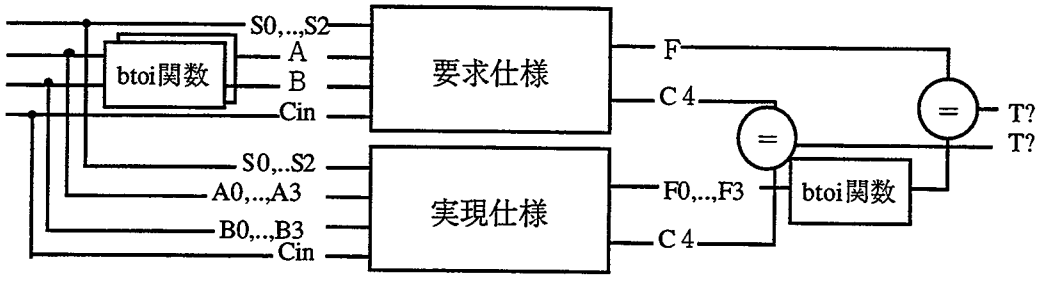
図１５　74382の要求仕様

```
  :
∧ (
F0 = CIN ∧ ¬ _LC009 ∧ ¬ _LC011 ∧ S0 ∧ ¬ S2
   ∨ CIN ∧ ¬ _LC009 ∧ ¬ _LC011 ∧ S1 ∧ ¬ S2
   ∨ ¬ _LC038 ∧  ¬ S0 ∧  ¬ S1
   ∨ ¬ _LC038 ∧ S2
   ∨ ¬ CIN ∧ ¬ _LC038 )
∧ (
  :
```

図１６　74382の実現仕様



(a)整数入力ABなどに対する全出力の等価判定



(b)論理入力A0,..,B0,..などに対する全出力の等価判定

図１７　74382の検証の概要

itob及びbtoi関数は，それぞれ，整数入力を論理出力に，論理入力を整数出力に変換する関数である(図１８).

図１７(a)(b)は，要求仕様及び実現仕様に対してそれぞれ同じ入力を与えたときに出力が同じであるかという出力等価を判定する回路を表している．(b)に対応する証明すべき式は以下のような形をしている．

∀全変数
((要求仕様
　∧
　実現仕様
　∧
　要求仕様の入力へのbtoi関数
　∧
　実現仕様の出力からのbtoi関数)
　imply
　　実現仕様の出力のbtoi=要求仕様の出力)　　(1)

```
( A0 = ( $A = 1 or $A = 3 or ... or $A = 15))
  and
( A1 = ($A = 2 or  $A = 3 or ... or $A = 15))
 and
( A2 = ( $A >= 4 and $A <= 7
        or $A >= 12 and $A = 15))
  and
( A3 = ( $A >= 8 and $A <= 15))
```

(a) 整数変数$Aから論理変数[A0..A3]へのitob関数

```
if not A3 and not A2 and not A1 and not A0
    then $A = 0
else if not A3 and not A2 and not A1 and     A0
    then $A = 1
       :
else if     A3 and     A2 and     A1 and not A0
    then $A = 14
else $A = 15
```

(b))論理変数[A0..A3]から整数変数$Aへのbtoi関数

図１８　データタイプの異なる変数の対応の記述

ただし，(a)では，整数変数の定義域（4bitデータであるので０～１５）を前提として証明を行う．例えば，以下のような式を判定する．

∀全変数

((0 <= $A ∧ $A <= 15) ∧ (0<= $B ∧ $B <= 15))

imply(証明すべき式) (2)

判定系は，式(1)あるいは式(2)を入力として，要求仕様と実現仕様の２つの回路が等価なときのみ真を出力する．

比較として用いた従来の判定系は，文献[46]で提案されている方法である．この判定系は，本手法と同じ Cooper のアルゴリズムを採用している．また，この判定系は，すべての変数が冠頭の全称限定子で束縛されているプレスブルガー文のみを扱うことができ，そのことを利用して，全称限定子を適用する順番をヒューリスティックに決定するなどの工夫により高速な式判定を実現している．しかしデータの内部表現は，本手法のように内部頂点の共有を行っていない．

表１４　検証例題に対する実行結果

	論理変数	整数変数	従来	提案する判定系
74382				
論理出力で比較	56	4	0.3	1.5
整数出力で比較	56	4	0.3	1.5
HLSynth95 FP_ADD の出力nan_res	45	39	256.1	210.1

(秒)

PentiumII 300MHz 128MBmem

判定結果を表１４に示す．検証(a)及び(b)では整数変数の全称限定子の消去によって０～１５までのすべての数が代入されてしまう(例えば変数 $A を消去する場合，図１８の関数の記述等から，上述の存在限定子に対するアルゴリズムにおいて０から１５の値がそれぞれ $A に代入される）結果，問題の複雑さという点において同等な問題であり，実行時間に差は見られなかった．本例題では，１整数変数は４論理変数の組を表現するだけであるので，入力の規模は，全て論理変数に換算すると70程度と考えて良い．従来手法に比較して５倍程度の時間を要している．

次に，さらに大規模な組み合わせ回路の検証実験について説明する．用いるのは，MCNC ベンチマークセットのなかの HLSynth95 の浮動小数点加減算器 fp_add である．

浮動小数点は，１ビットの符号，２３ビットの仮数及び１６ビットの指数の組（計４０ビット）で表現される．要求仕様及び実現仕様では，仮数及び指数を整数変数で表現する．

要求仕様は，各出力毎に，変数に入力変数のみからなる P 関数を用いて記述される．実現仕様は，fp_add の VHDL 記述を元に，中間ネットも整数変数あるいは論理変数として扱い，図１６と同様な論理表現に変換して得られたものを用いた．

本例題では，出力のうち，異常入力が与えられたときに立つ NaN フラグについてのみ，要求仕様における出力と実現仕様における出力が等価になるかを調べた．本例題では，入力出力共におなじデータタイプであるので，データ変換の必要がない．この結果を表１４の最下段に示す．本例題で用いた出力は依存する入力変数が2個と少ない例である．けれども，従来手法が出力変数自身あるいはそれらにデータ依存関係のある端点に対応する変数から消去するヒューリスティックを採用しており，本検証問題では出力に依存する変数から消去を行い高速に判定が行えるはずであるのに対して，本手法では証明すべき式において出現する順に変数を消去するという単純な方法を採用しているにも関わらず．20%程度計算時間が短く判定を行うことができた．

本例題は，一つの整数変数が最大２３ビットに対応し，検証例題としてはかなり大きな例であるが，本実験により，提案する判定系によって検証可能であることが示された．その他の出力においては，従来手法でも 10000 秒以上要したので実行を打ち切り，本手法でも全称

限定子を施す前までのデータ構造の作成は行えたが，全称限定子に対する処理を実行する途中においてメモリ不足による実行中止となりデータを収集できなかった．

5.4 結言

本章では，プレスブルガー文真偽判定のための新たなデータ構造の提案と評価実験の結果を示した．本手法は，BDDs と整数型のデータ構造を保持する線型リストの組み合わせという比較的単純なデータ構造であるにもかかわらず，共通の部分グラフの共有などの特徴により，従来手法より高速に大規模回路の検証を行うことができることを示した．

本論文では，回路検証の例として組み合わせ回路の例を用いたが，4 章で述べた順序回路に対する検証にも適用可能である．今後提案した手法を順序回路のより大規模な検証に適用し，本手法の評価並びに提案する順序回路の検証手法の評価を行いたい．

今後さらなる高速化及び省メモリ化のために，従来手法あるいは BDDs において採用されている高速化及び省メモリ化のための手法を適用したり，また，提案するデータ構造特有の改善手法を考案し，本ライブラリの改良を行いたい．今回のデータ構造では，表現すべき論理式に対するデータ構造の一意な表現を保証できない．例えば，論理式 A=1 ∧A+B=5 と論理式 A=1 ∧B=4 あるいは A>1 ∧A>5 と論理式 A>1 は，いずれも等価な論理式といえるが，提案するデータ構造では，異なるデータ構造によって表現される．与えられた論理式に対して一意に表現することができれば，論理式の等価性は容易に調べることができ，部分式の共有もより効率的に行うことができる．プレスブルガー文の一意な表現方法としては，文献[48]及び[50]にて提案されている方法があり，これらの手法との比較も検討したい．

また従来の判定系はすべての変数が冠頭で全称限定子で束縛されるプレスブルガー文というあるクラスのみに対する判定系であるのに対し，今回提案した判定系は，一般のプレスブルガー文に対する真偽判定が行える．今後，回路検証においてプレスブルガー文で表すことのできる興味ある検証問題について取り組みたいと考えている．

また，一方では，取り組むべき検証問題に対して十分なクラスで高速に処理できるような論理式のクラスで十分であるという立場もある[47, 48, 49]．状態数え上げなどの本論文では

取り上げられていないその他の検証問題に対する本データ構造を用いた処理系の能力やそれぞれの問題に特化された高速化手法なども興味ある問題である.

6　結論

本研究によって得られた成果，及び今後の本研究の発展の方向についてまとめる．

本研究では，設計の上流工程におけるハードウエア設計の支援及び設計の正しさの形式的検証の支援を目標として，設計あるいは検証手順の一部を自動化あるいは使用者が容易に作業が行えるための支援システムを作成し，評価実験を行った．

提案した設計法及び検証法は，記述スタイルあるいは設計においていくつかの制約に基づいている．例えば，回路仕様においては制御部が一つであること，実現の定義においては要求仕様と実現仕様において状態を単位として出力（各成分関数値）の比較を行っていること，設計では各状態遷移毎に下位の状態遷移系列を対応させ具体化すること，検証においては不変表明は制御部状態を単位として設定することなどである．しかし，評価実験を行った CPU，ソート回路，GCD 回路などでは，回路の動作全体を一つの遷移で表すような抽象レベルから設計者の意図する階層的な設計を自然に行うことができ，回路の簡約を行えば人手による設計と同じ品質の回路が設計できることを示した．また検証においては，CPU においては自動的に，ソート回路及び GCD 回路においては検証者がいくつかの補題の設定を行えば，後の作業は自動的に支援系が行うなど，実際に設計検証が実際に可能であることを示した．

現在，本研究の成果に基づき，以下の発展的な研究が行われている．文献[20-24]では，複数の有限制御部を有する回路に対して，本論文で提案した実現の定義および状態遷移の対応関係を拡張し，PCI バスインターフェイスを有するシステムの設計の正しさの検証を行っている．また，この手法に基づき，Out-of-Order 実行型のパイプライン CPU の設計とその設計の正しさの検証への取り組みが行われている[51]．

また，本研究で提案したプレスブルガー文に対するデータ構造を用いた検証系は，大規模な組み合わせ回路の検証例題において従来手法とほぼ同等に式判定が行えるが，高速化手法はまだ十分に検討されているとはいえない．今後，BDDs における変数の適当な順序付けなどさらにデータをコンパクトに表現し，高速な処理を実現するための手法などを今回提案した検証系に組み込み，上述の Out-of-Order 実行型のパイプライン CPU の設計の正しさの検証

などより大規模な回路の検証に取り組みたいと考えている.

謝辞

本研究を行うにあたり，常日頃より適切な御指導を賜り，また，忍耐強く励ましていただきました大阪大学大学院基礎工学研究科情報数理系専攻 谷口 健一 教授に心より深謝申し上げます．

本研究を行うにあたり，有益な御助言および御指導頂きました情報数理系専攻の故 西川 清史 教授に深謝申し上げます．

筆者が，大阪大学基礎工学部情報工学科および同大学院に在籍中から同大学に助手として勤務に携わっている現在にわたりまして，御指導，御教授くださいました情報数理系専攻の菊野　亨　教授，今井　正治　教授，柏原 敏伸　教授，藤原 融　教授，都倉 信樹　教授，萩原 兼一　教授，藤井　護　教授，井上 克郎　教授，宮原 秀夫　教授，橋本 昭洋　教授に深謝いたします．

本研究を進めるにあたり，適切なご助言をしてくださいました岡山大学の杉山裕二教授，大阪大学大学院基礎工学研究科情報数理系専攻の船曳 信生 助教授，東野 輝夫 助教授，岡野 浩三 助手に深謝いたします．

本研究を進める上で，CAD ツールの貸与ならびに有益なご助言をいただきました日本電信電話株式会社光ネットワークシステム研究所の小栗 清 様ならびにコミュニケーション科学研究所の並列処理グループの皆様に心より感謝いたします．

本研究の初期の段階において，共同して研究に携わっていただいたシャープ株式会社 横山 昌生 様，日本アイ・ビー・エム株式会社 森岡 澄夫 様に感謝いたします．

常日頃より御討論していただいた谷口研究室，西川研究室の方々，試作システムの作成など協力いただいた両研究室において筆者と同じ研究グループに所属されていた卒業生の方々に感謝いたします．

参考文献

[1]安浦寛人，“論理合成時代のハードウエア記述言語”，情報処理，Vol33，No.11（1992).

[2]D.L.Perry，メンターグラフィックスジャパン訳：“VHDL”，アスキー出版局(1996).

[3]中村行宏，小野定康：“ULSIの効果的な設計法”，オーム社(1994).

[4]高橋隆一，吉村猛：“ハイレベルシンセシスの動向”，電子情報通信学会論文誌A，Vol.J74-A，No.2，pp.143-151(1991).

[5]Shin-ichi Minato：“Binary Decision Diagrams and Applications for VLSI CAD”， Kluwer Academic Publishers(1996).

[6]M. Gordon：“Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware， in Formal Aspects of VLSI Design Proceedings of the 1985 Edinburgh Conference on VLSI， edited by G. J. Milne and P. A. Subrahmanyam，North-Holland， pp. 153-177(1986).

[7]谷口健一， 北道淳司：“代数的手法による仕様記述と設計及び検証”，情報処理，Vol.35，No.8，pp.742-750(1994).

[8]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”，第6回回路とシステム軽井沢ワークショップ，pp.375-380(1993).

[9]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(1994).

[10]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).

[11]村上 尚，北道 淳司，西川 清史，谷口 健一：“代数的手法を用いた同期式順序回路のグラフィック環境での状態図編集による設計支援システムの開発”，1995年電子情報通信学会総合大会，A-121 (1995).

[12]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合成について"，情報処理学会研究報告，DT46-23(1989).

[13]楢原考一，北道淳司，東野輝夫，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設計例”，信学技報，FTS91-44(1991-20).

[14]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，pp.281-286(1995).

[15]Junji Kitamichi， Sumio Morioka， Teruo Higashino and Kenich Taniguchi：“Automatic Correctness Proof of Implementation of Synchronous Sequential Circuits Using Algebraic Approach”，2nd Conference on Theorem Prover in Circuit Design (TPCD94) (1994).

[16]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合せ回路の実現の正しさの証明”，情処全大4L-01(1994).

[17]J. Kitamichi， N. Funabiki and S. Nishikawa: “Proposal of Data Structure for Presburger

Arithmetic and its Application to Circuits Verification”, 1997 Int. Symp. on Nonlinear Theory and its Applications, Vol.2, pp.1233-1236(1997).

[18]北道淳司, 東野輝夫, 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エア, Vol.9, No.6, pp.31-39(1992).

[19]北道淳司, 森岡澄夫, 東野輝夫, 谷口健一: “並列実行される動作におけるデータ代入の衝突の判定”, 情処全大4H-01(1993).

[20]竹中崇, 北道淳司, 西川清史: “複数の制御部を持つ同期式順序回路の一設計検証法”, 情報処理学会論文誌, Vol.39, No. 7, pp.2308-2322(1998).

[21]齋藤義勝, 竹中崇, 北道淳司, 船曳信生: “複数の制御部をもつ同期式順序回路に対する不変式の形式的検証法”, 情処研報Vol.97, No.119, 97-DA86, pp.41-48 (1997).

[22]齋藤 義勝, 竹中 崇, 北道 淳司, 西川 清史: “代数的手法を用いた複数の制御部を持つ同期式順序回路に対する設計および検証支援系の開発”, 1997年電子情報通信学会総合大会講演論文集 A-3-23, p.128 (1997).

[23]竹中 崇, 北道 淳司, 西川 清史, 谷口 健一: “シストリックアレーによる回路設計の正しさの一証明法”, 情報処理学会第52回全国大会 (1996).

[24]竹中 崇, 北道 淳司, 西川 清史: “複数モジュールにより構成される回路仕様に対する効率的な形式的検証法”, 電子情報通信学会技術研究報告, VLD97-87, FTS97-50 (1997).

[25]岡野浩三, 北道淳司, 東野輝夫, 谷口健一: “順序機械型プログラムの階層的設計法と

在庫管理プログラムの開発例”， 電子情報通信学会論文誌 D-I， Vol. J76-DI， No.7，
pp.354-363 (1993).

[26]社団法人 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”，
(1988).

[27]Wayne Wolf， Andres Takach and Tien-Chien Lee：“Architectural Optimization Methods for Control-Dominated Machines”， in Raul Camposano and Wayne Wolf(eds.)， High-Level VLSI Synthesis， Kluwer Academic Publishers， pp.231-254(1991).

[28]S.Garland， J.Guttag：“Verification of circuits using LP”， Proc. of The Fusion of Hardware Design and Verification， North Holland(1988).

[29]J.J.Joyce：“Formal Verification and Implementation of a Microprocessor”， VLSI Specification， Verification and Synthesis， Kluwer Academic Publishers， pp.129-157(1988).

[30]M.C.Mcfarland：“Formal Verification of Sequential Hardware: A Tutorial”， IEEE Trans. on CAD of IC &Sys.， Vol.12， No.5(1993).

[31]R.Hojati， A.Isles， D.Kirkpatrick and R.K.Brayton：“Verification Using Uninterpreted Functions and Finite Instantiations”， N.Srivas and A.Camilleri(Eds)， 1st Int. Conf.， FMCAD96(1996).

[32]K.Schneider and T.Kropf：“A Unfired Approach for Combining Defferent Formalisms for Hardware Verification”， N.Srivas and A.Camilleri(Eds)， 1st Int. Conf. FMCAD96(1996).

[33]F.J.Cantu, A.Bundy, A. Smaill and D. Basin : " Experimants in Automating Hardware Verification Using Inductive Proof Planning" , N.Srivas and A.Camilleri(Eds) , 1st Int. Conf. FMCAD96(1996).

[34]D.C.Cooper : "Theorem Proving in Arithmetic without Multiplication" , Machine Intelligence, No.7(1972).

[35]T. Bultan, R. Gerber and W. Pugh : "Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetic" , Int. Conf. on Computer Aided Verification 97(1997).

[36]K. Naoi and N. Takahashi : "An n^3u Upper Bound on the complexity for Deciding the Truth of a Presbueger Sentence Involving Two Variables Bounded Only by Existential Quantifiers" , IEICE Trans. Inf. Syst., Vol.E80-D, No.2(1997).

[37]W.Kelly, V.Maslov, W.Pugh, E. Rosser T.Shpeisman and D. Wonacott : The Omega Project: Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs, http://www.cs.umd.edu/projects/omega/.

[38]E.M.Clarke , E.A.Emerson and A.P. Sistla : "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications" , ACM Trans. on Programming Languages and Systems, Vol.8, No.2, pp.224-263(1986).

[39]R.E. Bryant and Y.-A Chen : "Verification of Arithmetic Circuits with Binary Moment Diagrams" , Int. Conf. 32nd DAC, pp.535-541 (1995).

[40]R.Drechsler, B. Becker and S. Ruppertz : " The K*BDD: A Verification Data Structure" ,

IEEE Design & Test of Comp. April (1997).

[41]BXD package page , http://www.cs.cmu.edu/afs/cs/usr/ yachen/www/bxd.html

[42]G.D.Hachtel and F. Somenzi：“Logic Synthesis and Verification Algorithms”， Kluwer Academic Publishers(1996).

[43]大蘆雅弘，杉山裕二，谷口健一：“代数的言語ASLにおける抽象的順序機械型プログラムとその処理系”，信学論，Vol.J73-D-I，No.12，pp.971-978(1990).

[44]嵩忠雄，谷口健一，杉山裕二，関浩之：“代数的言語ASL/* - 意味定義を中心に - ”，信学論，Vol.J-D，No.7，pp.1066-1074(1986).

[45]景山洋行，北道淳司，船曳信生：“整数上のある制約論理式の処理のためのBDDの拡張とそれを用いた回路検証”，DAシンポジウム'98論文集，pp.89-94(1998-07).

[46]森岡澄夫：“プレスブルガー文真偽判定手続きを用いたプログラム正当性証明”，大阪大学大学院基礎工学研究科博士論文(1997).

[47]T.Amon， G.Borriello， T. Hu and J.Liu：“Symbolic Timing Verification of Timing Diagrams using Presburger Formulas”， Int. Conf. 34nd DAC(1997).

[48]T.R.Shiple， J.H.Kukula and R.K.Ranjan：“A Comparison of Presburger Engines for EFSM Reachability”， 10th Int. Conf CAV'98， Vol. 1427 of LNCS(1998).

[49]M.Srivas， H.Rueβ， and D. Cyrluk：“Hardware Verification Using PVS”， Formal Hardware

Verification - Methods and Systems in Comparison, Vol. 1287 of LNCS(1997).

[50]P.Wolper and B. Boigelot：“An Automata-Theoretic Apporoach to Presburger Arithmetic Contraints (Extended Abstruct)”， 2nd Int. Symp.， SAS'95， Vol. 983 of LNCS(1995).

[51]小林英史，竹中崇，北道淳司，船曳信生，谷口健一：“out-of-order型パイプラインCPUの機能検証”，DAシンポジウム'98論文集，pp.83-88(1998-07).

[52]R.E.Shostak：“ On the SUP_INF Method for Proving Presburger Formulas”， Journal of ACM, Vol.24，No.4， pp.529-543(1977).

[53]Thomas Kropf：“Benchmark-Circuits for Hardware -Verification v1.2.0", ftp:// goethe . ira . uka . de / pub / hvg / benchmarks/ Whole_documentation.pdf.

[54]星野民夫，唐津修：“UDL/I”，解説“ハードウエア記述言語”情報処理，Vol.33，No.11(1992).

[55]Jorgen Staunstrup：“Design Verification using SYNCHRONIZED TRANSITIONS", Formal Hardware Verification - Methods and Systems in Comparison， Vol. 1287 of LNCS(1997).

[56]平石裕実，浜口清治：“論理関数処理に基づく形式的検証法”，解説“論理設計の形式的検証”，情報処理，Vol.35，No.8(1994).

表目次

図目次