設 計 自 動 化 83－2
（1997．2．14）

代数的手法によるPCIバスコントローラの設計検証

竹中 崇　景山 洋行　北道 淳司　西川 清史
大阪大学大学院基礎工学研究科情報数理系専攻
{t-takena,kageyama,kitamiti,nisikawa}@ics.es.osaka-u.ac.jp

本報告では，複数の制御部をもつ回路を対象にした一つの設計法および検証法について提案する．従来，我々の研究グループでは，代数的手法を用いた単一の制御部をもつ同期式順序回路の要求仕様からの段階的設計法及び形式的な設計検証法を提案してきたが，これを複数制御部を持つ回路に拡張する．設計及び検証の対象としたシステムには，PCIローカルバスに対して，その機能に注目した抽象度の高い回路（上位レベルの回路）と，PCIバス，PCIバスコントローラなどからなるモジュール群で設計した抽象度の低い回路（下位レベルの回路）がある．本手法では，上位レベルの回路から下位レベルの回路を設計し，下位レベルの回路が上位レベルの回路を正しく実現していることを検証する．この際，回路はそれぞれ公理で記述し，設計者は両レベルの回路の対応関係を公理の形で与える．上位レベルの回路の各公理が下位レベルの回路の公理，対応関係の公理および基本演算の公理などのもとで定理として成り立つことを項書換え，整数制約上の論理式の恒真性判定ルーチン等を用いて証明する．

Formal Verification of PCI Bus Interface Controllers based on Algebraic Methods

Takashi TAKENAKA, Hiroyuki KAGEYAMA,
Junji KITAMITI and Seishi NISHIKAWA
Division of Informatics and Mathematical Sciences,
Graduate School of Engineering Science, Osaka University

In this paper, we propose the design and formal verification methods for synchronous sequential circuits with more than one control units using the PCI local bus system. The proposed methods are based on the extension of our existing methods for circuits with a single control unit. Our design method proceeds the circuit design by transforming the circuit at the higher level into the more detailed circuit at the lower level and giving the correspondence relation between the levels. Each level circuit and correspondence relation is described in axiom. At the same time, our verification method verifies that all the functions of the higher level circuit are realized in the lower level circuit by proving that each axiom of the higher level holds as the theorem under the lower level axiom, the axiom of the correspondence relation and the axiom of the basic calculus and so on. The term rewriting and the decision procedure of the logic formula on the integer agreement are adopted in the verification method. In the PCI local bus system of this paper, the higher level circuit describes the abstracted functions whereas the lower level circuit includes the PCI bus module, the PCI bus controller modules and so on.

1　はじめに

近年のハードウェア設計の大規模化に伴い，より抽象度の高いレベルにおける仕様記述からの高位設計に関する研究や，高位設計により得られた回路が正しく動作するかどうかを検証（ここでは，厳密な意味での証明）するための形式的手法に関する研究が盛んに行われている．筆者の属する研究グループでは，単一制御部を持つ同期式順序回路に対して，高位レベルにおける設計および設計の正しさの形式的検証を行うために，代数的手法を用いた設計および検証支援システムを構築し，それらの有用性について評価してきた [1, 2, 3]．

一般に回路設計においては仕様記述の繁雑さや回路の局所密集を避けるために回路をいくつかのモジュールに分割し，複数の制御部により実現することが多い．そこで，本研究では，従来の手法に対して複数の制御部を有する回路に対する設計および検証方法を提案する．さらに，近年計算機において一般的に用いられつつある PCI(Peripheral Componet Interconnect) バス [4, 5] を有するシステムの PCI バスコントローラに対して本手法を適用した[1]．PCI バスを持つシステムは，現実に広く使用されており，また，一つの PCI バスコントローラで，受信，送信の 2 種類の独立した動作を行うので，単一制御部によりこれらを制御するよりもそれぞれの動作を制御する複数の制御部で実現することが適当であるなどの理由から，本手法の例題として取り上げる．

従来手法では，上位レベルの回路の状態と下位

[1]今回例題として使用した PCI バスを持つシステムはバルテノン研究会バルテノン研究会による「第 3 回 ASIC デザインコンテスト規定課題 PCI バスインターフェイス」[6] および「PCI バス シミュレーション環境」[7] に準拠している．

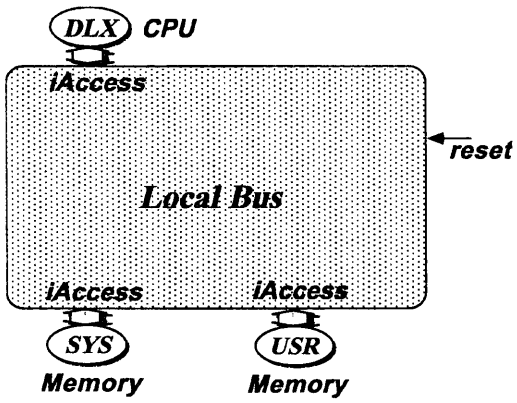


図 1-a: 上位レベルの回路の構成

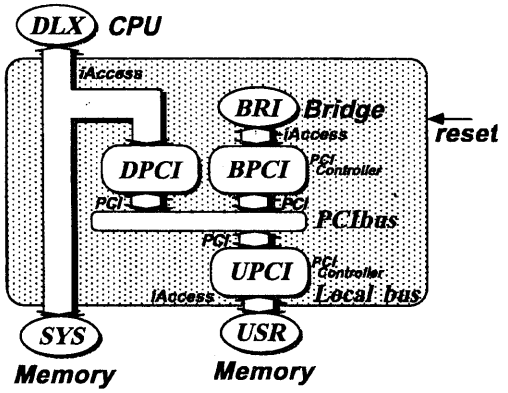


図 1-b: 下位レベルの回路の構成

図 1: 対象となるシステムの回路の構成

レベルの回路の状態との対応という概念のもとで議論を行ったが，本手法では，上位レベルの回路の状態における入出力端子やレジスタの値が，下位レベルの回路における複数の状態にわたってそれぞれ対応するような場合も含めて実現の定義を行い，設計法及び検証法を拡張している．

以下，2で設計および検証の対象となるシステムの説明を行い，3で代数的な仕様記述法について述べる．さらに，4で実現の正しさについて議論を行い，5で設計法について説明し，6で設計の正しさの証明法について述べる．

2 設計及び検証の対象となるシステム

2.1 システムの概要

設計及び検証の対象とするシステムは，ローカルバスとプロセッサ DLX，2 つのメモリ (システムメモリ SYS，ユーザーメモリ USR) からなる．ローカルバスと各デバイスとの間は iAccess というアクセスインターフェイスで接続されている．図 1-aにその概略を示す．本稿ではローカルバスに対して，上位レベルの回路及び下位レベルの回路の 2 つのレベルの回路を考える．

2.2 上位レベルの回路

ローカルバスは外部端子として DLX と 2 つのメモリに接続するための iAccess 端子群とリセット端子 reset を持つ．

iAccess インターフェイスは，文献 [6] により定義されており，物理的に接続される端子群 (以下 iAccess 端子群) とその端子群を経由してアクセスするためのプロトコル (以下 iAccess プロトコル) が規定されている．iAccess 端子群はアドレス端子 Adr，データ端子 Data などデータ が転送される端子 (iAccess データ端子)，メモリライト要求端子 WriteMem，ライト完了通知端子 WriteDone など，デバイスからローカルバスにデータ転送を要求する場合に使用される制御端子 (iAccess マスタ端子)，メモリライト要求端子 WriteMemReq，ライト完了通知端子 WriteReqDone などローカルバスからデータ転送を要求される場合に使用される制御端子 (iAccess スレーブ端子) および，リセット信号 Reset など共通に使用される端子からなる．

図 2に iAccess インターフェイスを介したデータ転送時に生じる各端子の挙動の例を示す．DLX があるメモリに対して書き込み要求を行う場合，iAccess データ端子のうち Adr 端子にそのデバイスのアドレスを，Data 端子に書き込むべきデータを，Be 端子にデータのバイトイネーブル情報をそれぞれドライブし，iAccess マスタ端子のうち，WriteMem 端子をアサートする (図 2: Time 0)．次に，iAccess データ端子のうち Adr 端子にそのデバイスのアドレスが，Data 端子に書き込みたいデータが，Be 端子にデータのバイトイネーブル情報がそれぞれドライブされ，iAccess スレーブ端子の WriteMemReq 端子がアサートされることにより，バスからメモリに対して書き込み要求が行われる (図 2: Time 1)．メモリは書き込みが終了すると WriteReqDone 端子をアサートすることにより書き込みが終了したことをバスに通知する (図 2: Time 2)．そして，iAccess マスタ端子の WriteDone がアサートされることにより DLX に対して書き込みの終了が通知される (図 2: Time 3)．

2.3 下位レベルの回路

下位レベルの回路は PCI バスコントローラ DPCI，UPCI，BPCI，バス PCIbus およびブリッジ BRI などのモジュールから構成されている．概略を図 1-b に示す．

PCI バス [4, 5] はデータ幅が 32 もしくは 64 ビットである同期式のバスアーキテクチャである．PCI バスを経由するデータ転送をトランザクションと呼ぶ．特にトランザクションにおいて，アドレス線を利用してしてデータ転送の相手モジュールを指定する (転送元となる) モジュールをマスタ (Master)

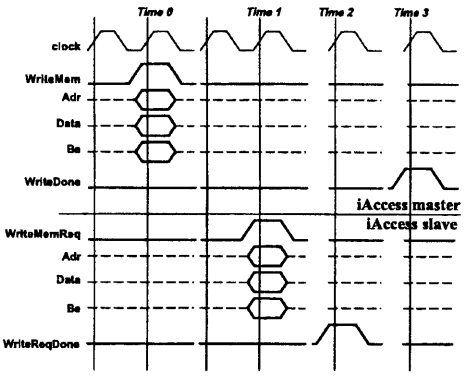


図 2: iAccess インターフェイス: ライト・アクセスにおけるタイムチャート

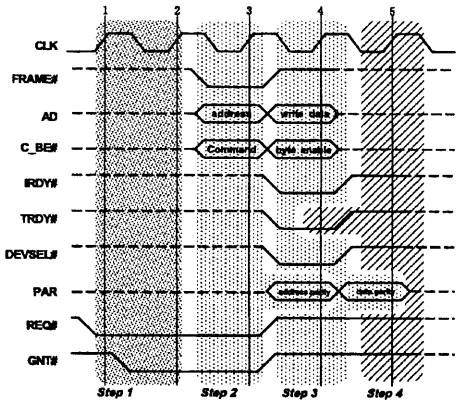


図 3: PCI インターフェイス: 典型的なトランザクションにおけるタイムチャート

またはイニシエータ (Initiator) と呼び，マスタが指定したモジュールをターゲット (Target) と呼ぶ．

典型的なトランザクションの例を図 3，図 4に示す．この例は，プロセッサからメモリに対するデータ転送の様子を表している．図 3において，CLK はクロックを表し，FRAME# はバスが使用されているときにアサートされる信号線であり AD，C_BE#はそれぞれ，アドレス/データ 線，バスコマンド/バイトイネーブル 線を表す．また，IRDY#，TRDY#はそれぞれイニシエータ，ターゲットがデータの授受をする準備ができている時にアサートされる信号線である．DEVSEL#はデータ転送を受けるターゲットデバイスが確定している時にアサートされる信号線である．PAR は AD および C_BE のパリティがドライブされる信号線である．そして，REQ#，GNT#は，それぞれバスの使用権要求/使用権許可を表す信号線である．

イニシエータは，トランザクションの実行に先だってバスの使用権を獲得する．そのために信号線

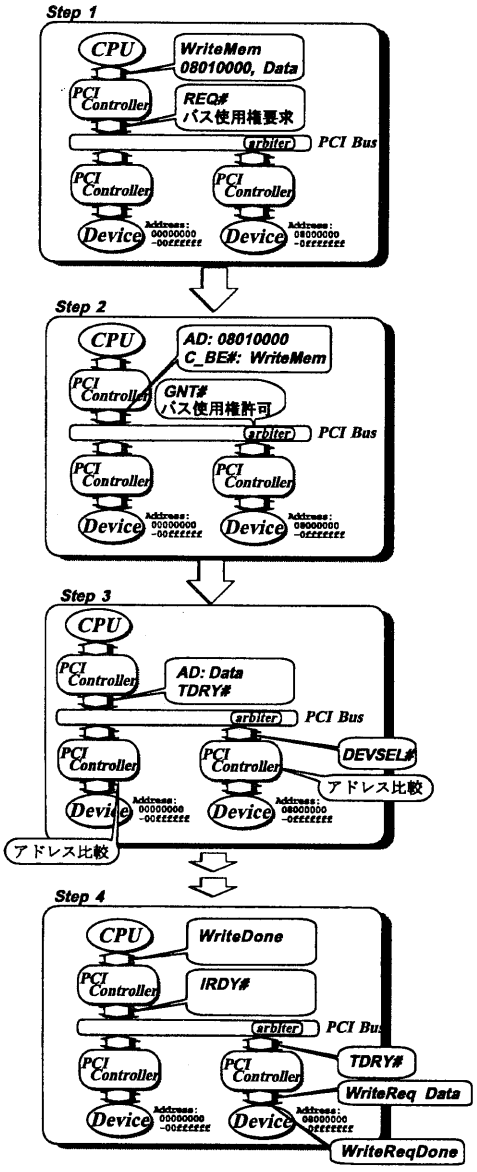


図 4: 典型的なトランザクションが行われる順序

REQ#をアサートする．イニシエータがバスからバスの使用権を獲得することをアービトレーションといい，バスの使用権の調停を行う回路をアービタという．アービタはイニシエータに対し信号線 GNT#をアサートしてバスの使用権を許可する (図 4:Step 1)．

バスの使用権を獲得して，かつ直前のトランザクションが終了していてバスがアイドル状態である時 (信号線 FRAME#と信号線 IRDY#がどちらもアサートされていないとき) に，イニシエータは信号

線 FRAME#をアサートしバスサイクルが始まることを通知し，信号線 AD にアドレス，信号線 C_BE#にバスコマンドを送る (図 4:Step 2).

全てのデバイスは信号線 FRAME#, AD, C_BE#を監視しおり，信号線 AD および信号線 C_BE#をデコードしてアドレス情報とバスコマンド情報をとり出し，自分がアクセスされたことを判定したデバイス (ターゲットとなる) は信号線 DEVSEL#をアサートして自分がこのサイクルに応答することを宣言する．イニシエータは信号線 AD にデータを，信号線 C_BE#にバイトイネーブル情報を送り，信号線 IRDY#をアサートして信号線 AD 上に有効なデータがドライブされていることを通知する．さらに，信号線 FRAME#をディアサートしてこのデータがこのトランザクションの最後のデータであることを通知する (図 4:Step 3).

ターゲットは信号線 TRDY#をアサートしてデータを受け取ることができることを通知し，信号線 AD と信号線 C_BE#上のデータを読む．イニシエータは信号線 TRDY#がアサートされているのでデータ転送が完了したことを知り，信号線 IRDY#をディアサートし，信号線 AD, C_BE#のドライブを止める．ターゲットは信号線 FRAME#がディアサートされているのでトランザクションが終了することを知り信号線 TRDY#と信号線 DEVSEL#をディアサートする (図 4:Step 4).

以上により，プロセッサからメモリに対する書き込み動作の手順が終了したことになる．

3 代数的仕様記述

3.1 概要

各レベルの同期式順序回路の記述には，我々の研究グループが提案してきた代数的記述言語 ASL/ASM を用いる[8].

一つの代数的記述は，文法と公理からなる．文法は関数の入出力のデータタイプを指定し，公理は，文法によって生成される関数がネストされたもの (項と呼ぶ) の合同関係を定義する．

レジスタ，メモリなど状態を持つ部品を状態成分関数 Fで表し，回路の動作を状態遷移関数 Tで表す．回路の全部品の情報を含む抽象状態をデータタイプ STATE で表し，初期状態を STATE 型の定数 INIT で表す．状態遷移関数は STATE 型から STATE 型を返す関数とし，状態遷移 Tにおいてある値 $exp(s)$ を成分 Fへレジスタ転送するということを下記のような公理で記述する．

$$F(T(s)) == exp(s);$$

ただし，$exp(s)$ は状態成分関数と基本演算・述語よりなる表現式である．

制御部 kの値を保持するレジスタを $\mathrm{CONTROL_k}$ で表す．$\mathrm{CONTROL_k}$は，STATE 型から制御部 kの状態値集合の要素 (状態名) を返す関数である．状態遷移の実行順を指定するための特別な関数 $\mathrm{VALID_k}$を導入する．状態 s において

```
公理 F1: USR.WriteMemReq(DLXWriteMemUSR(s))
          == True;
公理 F2: USR.Adr(DLXWriteMemUSR(s))
          == DecodeMemAdrFunc(DLX.Adr(s));
公理 F3: USR.Data(DLXWriteMemUSR(s))
          == DLX.Data(s);
公理 F4: USR.Be(DLXWriteMemUSR(s))
          == DLX.Be(s);
公理 F5: USR.ReadMemReq(DLXWriteMemUSR(s))
          == False
 ....
 ....
公理 V1: VALID(DLXWriteMemUSR(s))
  == VALID(s)
     and CONTROL(s) = cWait
     and DLX.WriteMem(s)
     and not DLX.ReadMem(s)
     and ....;
公理 C1: CONTROL(DLXWriteMemUSR(s))
          == WMUDoneWait;
 ....
 ....
```

表 1: 上位レベルの回路の仕様記述

$\mathrm{VALID_k}(T(s))$ が真となるとき，制御部 kは遷移 Tを実行させる．ただし，任意の状態 s において遷移 Tを実行する制御部は一意に決まり，各制御部において同時に $\mathrm{VALID_k}(T(s))$ を真にする遷移 Tはただ一つである．状態 s において実行される状態遷移は，各 $\mathrm{VALID_k}$で指定される複数の遷移を並行に実行することを表す (いわゆる直積) 遷移である．各制御部は同期して各遷移を実行させ，それぞれ次の状態に移る．

3.2 上位レベルの回路

上位レベルの回路では，ローカルバスの機能にのみ注目し，外部端子への入力により出力がどのように変化するかを有限状態機械で表す．

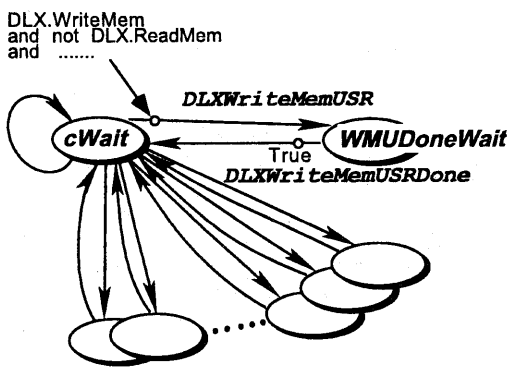


図 5: 上位レベルの回路の有限状態機械の概略図

図 5に上位レベルの回路の制御の概略図を示す．制御は，通常 cWait にあって，メモリライトなどの動作要求入力があった場合，それに対応する遷移を行い，cWait にもどるということを繰り返す．

例えば，端子 DLX.WriteMem が True(DLX からのライト要求があることを表す) かつ，データ端子 DLX.Adr 上のデータがメモリ USR を示すアドレスである，端子 DLX.ReadMem などの他の iAccess マスタ/スレーブ端子が False であるなどの条件で，遷移 DLXWriteMemUSR(メモリ USR に対して書き込みを行う遷移) を実行する．さらに，遷移 DLXWriteMemUSRDone(メモリが書き込みに対して応答する遷移) を実行することを制御している．

表 1に上位レベルの回路の仕様記述を示す．公理 F1 から 公理 F5 までが遷移 DLXWriteMemUSR の動作に関する公理の一部を表している．公理 F1 は 遷移 DLXWriteMemUSR を行った後の端子 USR.WriteMemReq の値が True であることを表し，公理 F2 は端子 USR.Adr の値が，端子 DLX.Adr の値に 基本演算 DecodeMemAdrFunc を施した値となることを表している．この 基本演算 DecodeMemAdrFunc は DLX 上のアドレスをメモリ USR 上のアドレスにマッピングする 演算である (各ビットの機能を決める演算の実現はより下位のレベルで行えば良い)．また，公理 F3,F4 が 遷移 DLXWriteMemUSR を行った後の端子 USR.Data,USR.Be の値が，それぞれ 遷移を実行する前の DLX.Data,DLX.Be の値となることを表している．公理 F5 は遷移後の端子 ReadMemReq の値が False となることを表している．さらに 公理 V1 及び 公理 C1 は 遷移 DLXWriteMemUSR が 制御部の値が cWait であり，DLX.WriteMem が True でありかつその他の条件が成り立つ時に実行され，遷移の実行後，制御部の値は WMUDoneWait となることを表している．

4 実現の正しさ

4.1 正しさの定義

上位レベルの回路の仕様記述に対して，設計された下位レベルの回路が与えられたとき，その設計の正しさを議論する．一般には上位レベルの回路，下位レベルの回路それぞれに対して同じ入力系列を与えたときに同じ出力系列が得られる場合に下位レベルの回路は上位レベルの回路を正しく実現しているとされる．ここで考える実現は，上位レベルの回路と下位レベルの回路の記述レベルが異なるので，上位レベルの回路において一状態において与えられる入出力が，下位レベルの回路においては連続した状態系列で与えられる場合がある．そこで，我々は，両レベル間において対応する状態という概念を導入し，入力および出力の対応を議論してきた．ここではさらに，上位レベルの回路の出力すなわち各成分が，下位レベルの回路のいくつかの状態におけるそれぞれの成分に対応させるということを考える．この対応を成分毎の状態の対応と呼ぶ．

図 6に成分ごとの状態の対応の例を示す．この図は上位レベルの回路，下位レベルの回路における対応する状態を表している．とくに下位レベルの

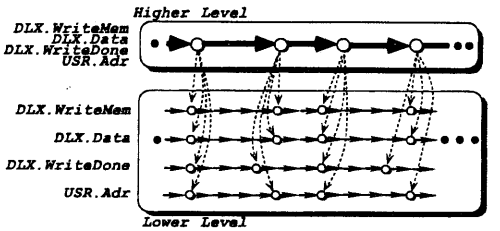


図 6: 成分ごとの状態の対応

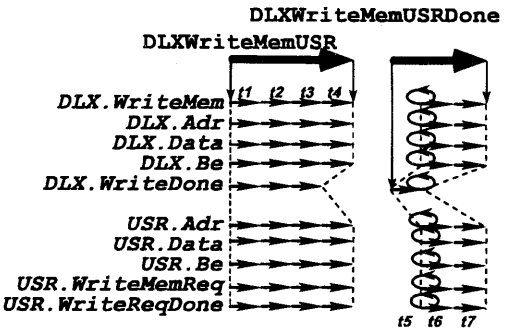


図 7: 成分ごとの遷移の対応

回路においては，実行系列を成分ごとに分けて表している．下位レベルの回路の実行系列においてそれぞれ成分ごとに白丸で示している状態が，それぞれ矢印つき点線で示す上位レベルの回路の状態と対応する状態である．

[定義 4.1] (実現の正しさ)
2 つの回路（初期状態も含む）と，成分ごとの状態の対応が与えられている．初期状態から 2 つの回路が動作するとき，各成分の値がこの対応関係のもとでそれぞれ等しければ，2 つの回路は与えられた対応のもとで等価であるという． □

4.2 正しさの十分条件

一般に初期状態からの入力系列，出力系列は無限に存在するので，与えられた成分ごとの状態の対応の元で常に等しいことを確かめるのは現実的に不可能である．そこで，従来は，上位レベルの回路の各遷移ごとに下位レベルの回路の遷移系列を対応させ，下位レベルの回路の遷移系列が行う動作と上位レベルの回路の一遷移が行う動作が等価であるかを示すことにより，下位レベルの回路が上位レベルの回路を正しく実現することを示した．

ここでは，前述のように実現の定義を変更したので，遷移の対応についても，成分ごとの遷移の対応という概念を導入する[2]．例えば図 7に示すような上位レベルの回路の遷移 DLXWriteMemUSR に

[2]成分ごとの遷移の対応で指定される遷移系列には，逐次実行，分岐，閉路がある．

```
((DLX.WriteMem(ss) = DLX.WriteMem(s)
  and DLX.Adr(ss) = DLX.Adr(s)
  and .....
  and DLX.WriteDone(ss)
        = DLX.WriteReqDone(s)
  and ....
  imply (
    DLX.Adr(DLXWriteMemUSR(ss))
        = DLX.Adr(t4(t3(t2(t1(s)))))
    and .....
    and DLX.WriteReqDone(DLXWriteMemUSR(ss))
        = DLX.WriteDone(t3(t2(t1(s))))
    and .....
  )))
    == True ;
```

表 2: 成分ごとの対応関係 M

対する成分ごとの遷移の対応は，代数的に表2のように公理で記述される[3]．

成分ごとの遷移の対応が与えられたときに，上位レベルの回路の一遷移が実行する動作と対応する下位レベルの回路の遷移系列が実行する動作が等価であることを示すことにより，実現の正しさを保証できる．

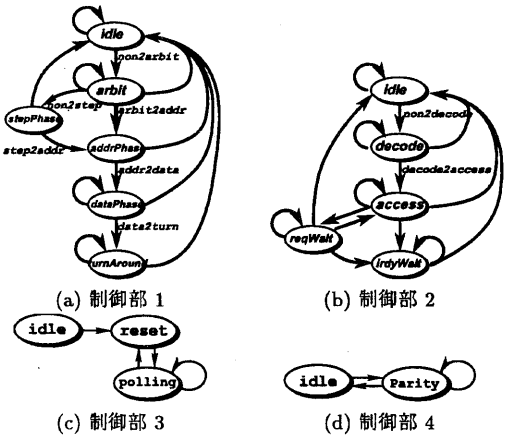
[定理 4.1] (実現の正しさの十分条件)
2つの回路 A，B(初期状態も含む)，成分ごとの遷移の対応 Mが与えられている．次の条件 1,2,3 を満たし，かつ，遷移の対応 Mから成分ごとの状態の対応 M'を構成した時，回路 Bが成分ごとの状態の対応 M'のもとで回路 A を正しく実現する．
(条件 1) (遷移に衝突が無い)
回路 A の各遷移 T_jについて，対応 Mに現れる直積遷移 ts_jにおいて，同一レジスタへの異なる値の同時書き込みなどの衝突が生じていないこと．(代数的には，ts_jを含む項が無矛盾であること) これは，ts_jを構成する各遷移に関する公理の右辺を調べれば容易に判定できる．
(条件 2) (対応通りに遷移させれば回路 A の遷移が実現できる) 回路 A の各遷移 T_jと，成分ごとの遷移の対応 Mで指定される遷移系列に現れる遷移が等価であること．
(条件 3) (対応通りに遷移させる制御部が実現できる) 回路 Bの制御部が，回路 A の実行制御と成分ごとの遷移の対応があらわす実行制御をただしく実現していること． □

5 設計

5.1 設計の概要

本手法では，以下の手順にしたがって設計を行うことによって実現の正しさを保証する．
(step 1) (遷移，制御部を設計)
下位レベルの回路の遷移を考案し，これを実行する下位レベルの回路の制御部を設計する．
(step 2) (対応を設計)
上位レベルの回路の各遷移と対応する下位レベルの回路の成分ごとの遷移系列との対応 Mを作る．
(step 3) (遷移系列が正しいことを検証)
考案した遷移系列が成分ごとの対応のもとで上位レベルの回路の遷移を正しく実現していることを証明する．
(step 4) (制御部が正しいことを検証)
設計した制御部が下位レベルの回路の遷移系列を正しく制御することを証明する．

[3]本稿では上位レベルの回路の成分に対してただ一つの下位レベルの回路の成分が対応しているものとする．ただし，我々の手法では上位レベルの回路の成分に対して複数の下位レベルの回路の成分を組み合わせて対応させることも可能である．



(a) 制御部 1　(b) 制御部 2
(c) 制御部 3　(d) 制御部 4

図 8: PCI バスコントローラの概略図

5.2 設計例

(Step 1) 上位レベルの回路 (図 1-a) に対して，DLX, ユーザメモリ等を制御する PCI バスコントローラ (DPCI, UPCI,BPCI)，PCI バス (PCIbus) および PCI ブリッジ (BRI) などのモジュールを導入して設計を行う．

図8に PCI バスコントローラの概略図を示す．一つの PCI バスコントローラは 4 つの制御部を有する．制御部 1 は PCI イニシエータ動作を制御し，制御部 2 は PCI ターゲット動作を制御する．制御部 3 は PCI イニシエータ動作および PCI ターゲット動作の起動を行い，制御部 4 はパリティ生成およびチェックを行うための制御部である．以下，3 つのバスコントローラの各成分，端子名の先頭に DPCI 等のプレフィックスをつける．

表3に下位レベルの回路の仕様記述例を示す．公理 i1 は 状態 s において，端子 UPCI.WriteMemReq が True であるときかつそのときのみ，端子 USR.WriteMemReq が True となることを，公理 d1 は 状態 s において，端子 BUS.C_BEenb1 が True であるときに，端子 BUS.nC_BE と端子 BUS.nC_BE1 の値が等しくなる，すなわちそれぞれ接続されていることを指定している．公理 r1 は遷移 UPCInon2decode を実行した後に UPCI.TCR の値が，遷移前の端子 UPCI.nC_BE の値と等しくなることを指定している．公理 v1 は 遷移

```
公理 i1:USR.WriteMemReq(s)
        == if (UPCI.WriteMemReq(s))
          then True
          else False;
....
公理 d1: (BUS.C_BEenb1(s)
     imply BUS.nC_BE(s) = BUS.nC_BE1(s))
         == True ;
....
公理 r1: UPCI.TCR(UPCInon2decode(s))
         == UPCI.nC_BE(s) ;
....
公理 v1: VALID_UPCI.Target(UPCInon2decode(s))
       == (VALID_UPCI.Target(s)
     and (CONTROL_UPCI_SuperVisor(s)
           = polling)
     and (CONTROL_UPCI_Target(s) = idle)
     and UPCI.nRST(s)
     and (not UPCI.iactive(s))
     and (not UPCI.tactive(s))
     and (not UPCI.nFRAMEin(s))
     and UPCI.nDEVSELin(s));
....
公理 c1:CONTROL_UPCI_Target(UPCInon2decode(s))
     == decode ;
....
```

表 3: 下位レベルの回路の仕様記述例

UPCInon2decode を実行するための条件を指定しており，公理 c1 は 遷移 UPCInon2decode を実行した後の制御部 UPCI_Target の値が decode であることを指定している．

(Step 2) 成分ごとの対応関係の例として上位レベルの回路の遷移 DLXWriteMemUSR に対応する下位レベルの回路の遷移系列の概略を図 7に示す．この図における遷移 t1,t2,...,t7 は下位レベルの回路の遷移の直積遷移を表している．

遷移 t1 は，転送元の PCI コントローラである DPCI がバスに対してバスの使用権を要求する遷移を表している．遷移 t2 は，DPCI が PCI バスに対して転送先のアドレスとバスコマンドをのせる遷移を表している．遷移 t3 は，残りの UPCI,BPCI がアドレスをデコードするために，バス上のアドレスをレジスタに格納する遷移を表している．遷移 t4 は，DPCI が書き込むデータとバイトイネーブルをバスにのせ，UPCI がバス上のこのデータとバイトイネーブルを端子 UPCI.Data, UPCI.Be を通して，USR.Data,USR.Be に出力し，同時に USR.Adr にアドレスを出力し，USR.WriteMemReq を True にする遷移を表している．また，遷移 t5 は，USR.WriteReqDone が入力されたときに DLX.WriteDone を出力する遷移を表している．遷移 t6, t7 は，バスの使用をおえるための動作を行う遷移を表している．表 2に示す公理は，上位レベルの回路の遷移 DLXWriteMemUSR に対して，端子 DLX.Adr については t1 から t4 までが対応し，端子 DLX.WriteDone については t1 から t3 までが対応していることなどを表している．

6 正しさの検証

6.1 対応が正しいことの検証 (Step 3)

状態成分ごとの遷移の対応の通りに遷移させれば，レジスタ転送が実現できること，すなわち，上位レベルの回路の各遷移 T_j と，状態成分ごとの遷移の対応 M で指定される下位レベルの回路における遷移系列 ts_j に現れる遷移が等価であることの証明法について述べる．

対応が正しいということは，代数的には上位レベルのレジスタ転送に関する公理が下位レベルの回路のテキスト，対応関係 M と基本演算・述語[4]のもとで定理となることである．

状態遷移 T_j に関する公理は，次のような形の公理で表すことができる[3]．

$$Pre_j(ss) \supset Post_j(ss, T_j(ss)) == \text{True}$$

ここで，Pre_j は状態遷移 T_j の遷移条件を表す述語で，上位レベルの回路のテキスト上の状態遷移 T_j に関する VALID に関する公理の左辺である．また，$Post_j$ は状態遷移 T_j の動作および遷移後の制御部状態を表す述語で，T_j の公理の左辺と左辺が等しいことを表す式である．

ただし，下位レベルの回路のテキストの公理のうち，証明には各成分において検証を行う上位レベルの回路の遷移に対応する下位レベルの回路の遷移に関する公理のみ使用する．例えば，遷移 DLXWriteMemUSR に関する証明においては，図 7における遷移 DLXWriteMemUSR の開始点に相当する点線から終点に相当する点線で囲まれる状態及びその状態における各成分の値のみを用いるということである．このように，前後に実行される可能性のある遷移での動作による影響を取り除いて証明を行うことにより，上位レベルの各遷移についてそれぞれ独立に下位レベルの対応する遷移と同等であることをしめす．

論理式が成り立つことの証明にはプレスブルガー文[9]と呼ばれる整数制約上の論理式の判定ルーチン[10]を用いる．このとき，公理などに設計者が導入した述語等を用いている場合は，検証者がこの述語に関する補題 P を証明すべき論理式 Q の前提とし，式 $P \supset Q$ を作成し，その式に対して出現する部分項を変数に置き換えて，論理式の判定を行う．

6.2 制御部が正しいことの検証 (Step 4)

設計した制御部が下位レベルの回路の遷移系列を正しく制御することを証明するには，下位レベルの回路上で初期状態からの状態探索を用いる．この証明手続きは，入力として，1) 上位レベルの回路の制御部，2) 下位レベルの回路全体，3) 成分ごとの遷移の対応 M が与えられており，次の手順で行う．

(Step 1) 上位レベルの回路の制御部と成分ごとの遷移の対応 M にしたがって，下位レベルの回路

[4] DecodeMemAdrFunc など，設計者が導入した演算に関する性質を表す公理．

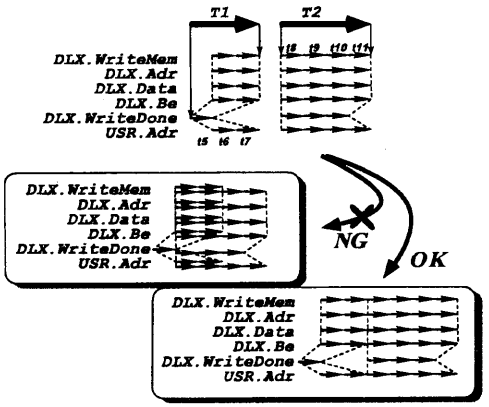


図 9: 制御部の検証

の遷移をならべる．ただし，Mは状態成分ごとに与えられているので，上位レベルの回路の存在する遷移の終了状態にあたる下位レベルの回路の状態は状態成分ごとに存在する．そこで，上位レベルの回路の連続する遷移に対して対応する下位レベルの回路の遷移系列を連続して並べる際には，任意の状態成分に対して，遷移が重ならないように並べる．遷移が連続しないときは，この間は値を保持する遷移を挿入する (図 9)．遷移をならべてできた状態遷移図を C_mとする．

(Step2) C_mにおける各遷移Tに対して，下位レベルの回路の各制御部が実行する遷移を確定する．この時，確定した各制御部の遷移の直積遷移がC_mにおける遷移Tと同じであること，各制御部がたかだか一つの遷移を制御していること（複数の遷移を制御しているときは，正しい実現でないと判定する）を確認する．制御すべき遷移がない制御部は，何も行わない遷移（NOP）を実行しているものとする．

(Step 3) 下位レベルの回路の各制御部の値の組（直積制御部の値）の集合Sを$\{Init_B\}$とする．ただし，$Init_B$は下位レベルの回路の制御部の初期状態とする．

(Step 4) C_mと下位レベルの回路の制御部を初期状態より同時に動かしながら，下位レベルの回路の制御部にしたがってC_mに同じ状態名を割り振る．割り振った状態名をSに加える．C_mのある状態から直積遷移Tを実行し状態S_nに遷移する際に次の条件を満たす時には停止し，正しい実現ではないと判定する．

- 下位レベルの回路の直積制御部がTを実行できないとき．
- 状態S_nの状態名が未割当で，下位レベルの回路でTを実行した次の状態名がすでにS_nに含まれているとき（下位レベルの回路には，上位レベルの回路には存在しない制御の閉路が存在する）．
- 状態S_nはすでに状態名が割当済みで，下位レベルの回路でTを実行した次の状態名がS_nに含まれていないとき（下位レベルの回路は，上位レベルの回路で指定されている制御の閉路を実現していない）．

(Step 5) 状態遷移図C'_mの全ての状態に唯一の直積状態名を割り振ることができれば停止し，正しい実現と判定する．

7 おわりに

本稿では，状態成分ごとの状態の対応と，これを用いた正しさの定義及び十分条件について提案し，PCI バスコントローラの設計を例にしてこの正しさの十分条件を満たすような設計法について提案した．現在，ここで提案した設計法に基づいて検証を進めており，部分的ではあるが本稿で述べた例について設計検証を行った．今後，検証にどの程度の手間がかかるのか，検証の自動化について従来手法を使用できるか，さらなる自動化を行えるか等について検討する．また，本論文では議論しなかったが，対応に閉路がある場合の停止性についての検証や，対応のクラス，遷移の重ね方が現実の回路を扱うのに十分か，制御部の検証は直積機械を作るが，これは実際的であるか等を調べたい．

参考文献

[1] 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).

[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, 信学論 A, Vol. J77-A, No. 3, pp. 420–429 (1994).

[3] 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, 情処論, Vol. 36, No. 10, pp. 2409–2421 (1995).

[4] PCI special Interest Group: *PCI Local Bus Specification Revision 2.1* (1995).

[5] 滝誠一: PCI バスの詳細と応用へのステップ, CQ 出版社 (1995).

[6] パルテノン研究会: 第 3 回 ASIC デザインコンテスト規定課題「PCI バスインターフェイス」(1996). http://www.kecl.ntt.co.jp/car/parthe/html/psfile/pci_spec.ps.

[7] 永見康一: PCI バス シミュレーション環境 (1996). http://www.kecl.ntt.co.jp/car/parthe/html/psfile/pci_sim.ps.

[8] 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 D-I, Vol. J-73-D-I, No. 12, pp. 971–978 (1990).

[9] 東野輝夫, 北道淳司, 谷口健一: 整数上の線形制約の処理と応用, コンピュータソフトウェア, 9, 6, pp.31-39 (1992).

[10] 森岡澄夫, 東野輝夫, 谷口健一: 全ての変数が存在記号で束縛された冠頭標準形プレスブルガー文の真偽判定プログラム, 信学技報, SS95−18, pp.63-70(1995).