計算機アーキテクチャ 127－6
設 計 自 動 化 86－6
（1997. 12. 11）

複数の制御部を持つ同期式順序回路に対する不変式の形式的検証法

齋藤 義勝　竹中 崇　北道 淳司　船曳 信生
大阪大学大学院基礎工学研究科情報数理系専攻
{y-saitoh,t-takena,kitamiti,funabiki}@ics.es.osaka-u.ac.jp

高位設計における複数の制御部を持つ同期式順序回路を対象とする，不変式の証明を用いた一形式的検証法を提案する．不変式とは設計者あるいは検証者が回路の動作中にレジスタや制御信号などの間に成り立つと考える関係である．回路記述は，複数の有限状態部 (制御部) とそれらが制御するデータパスからなる．不変式は各制御部の任意の有限状態に設定することができる．制御部における実行条件，データパスにおける演算および不変式は，論理型，整数型，ユーザが定義したデータタイプの変数および関数などを用いて記述することができる．検証は，いくつかの不変式，データパス，実行条件，ユーザ定義関数に関する補題を前提として，証明すべき不変式を証明するという，有限状態に関する構造的帰納法を用いて行う．この証明作業は，各制御部と等価な直積制御部を生成しその上で行う．直積制御部の生成時に到達不能な状態の削減をはかる．プレスブルガー文真偽判定ルーチンを利用して，証明すべき式の十分条件を判定する．本手法の有用性をPCIバスインターフェース回路を用いて評価する．

A Formal Method of Proving Invariants for Synchronous Sequential Circuits with Plural Controllers

YOSHIKATSU SAITOH, TAKASHI TAKENAKA,
JUNJI KITAMICHI, AND NOBUO FUNABIKI
Department of Informatics and Mathematical Sciences,
Graduate School of Engineering Science, Osaka University

In this paper, we propose a formal method of proving invariants for synchronous sequential circuits with plural controllers in high level design. The invariant is the relationship among registers and signals that the user asserts. The circuit design consists of several finite state machines (controllers) and datapath. The invariants can be set at arbitrary finite states by user. The execution conditions, operations in datapath, and invariants are described with variables and functions over Boolean, integer, and abstract data type defined by user. The invariants are proved on the assumption that several invariants, datapath, execution conditions, and lemmas for user-defined functions are correct. This proof uses the structural induction on finite states. On proving invariants, the system a product controller which is equivalent to each controllers and proves by structural induction with the product controller. We eliminate unreachable states on generating the product controller. The sufficient conditions of expressions to prove are proved with a decision algorithm for the truth of Presburger sentence. We evaluate the proposed method for verifying the correctness of the PCI bus interface circuit design.

1 はじめに

近年ハードウェアの設計において，回路の設計が非常に大規模かつ複雑化しており，より抽象度の高いレベルの記述からの高位合成に関する研究や，設計によって得られた回路の機能が正しいことを検証する手法に関する研究が盛んに行われている [1, 2]．回路の検証においては従来のシミュレーション等の手法では，回路が大規模化するにつれテストベクトルの数が増大し，回路の正しさを保証することが難しくなっている．一方，回路の正しさを数学的に証明するという，形式的検証法が注目されている．形式的検証法には記号モデル検査を用いた方法 [3, 4] や，定理証明を用いた方法 [5] 等が提案されている．

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法を提案し，その有用性について評価してきた [6, 7]．また回路の機能上の性質を，レジスタや制御信号線等の部品間に常に成り立つ関係式 (不変式) として検証者が与え，関係式が正しいことの証明を行うという手法が提案された [8]．

一般に，回路設計では回路記述の繁雑さや回路の局所集中を避けるために，複数の有限状態部 (制御部) とそれらが制御するデータパスからなる回路として，回路を実現する場合が多い．そこで本稿では，複数の制御部とそれらが制御するデータパスの集合からなる回路モデルを対象とした，形式的検証法を提案する．部品は論理型や整数型だけでなく，ユーザが定義するデータタイプを取ることができる．部品への値の転送には，複数の部品

の値を引数としたユーザが定義した関数の値を用いることができる．各制御部は，任意の部品を制御可能であり，実行条件の指定はユーザ定義関数の値や他の制御部の値を用いることができる．回路の一動作は，各制御部の指定する動作を並列に実行することと，各制御部の次状態への遷移として表される．不変式は，各制御部の任意の有限状態ごとに与えるものとする．

本稿での回路の検証では，このような回路仕様に対して，任意の初期状態 (各部品の初期値の組) から任意の入力系列が与えられて回路が動作しても，与えられた不変式が成り立つかという問題を考える．提案する検証法では，いくつかの不変式，データパス，実行条件，ユーザ定義関数に関する補題を前提として，証明すべき不変式が成り立つことを証明するという，有限状態に関する構造的帰納法を用いる．まず各制御部と等価な直積制御部を生成し，その上で帰納法を行う．直積制御部の生成時において到達不能な状態を生成しないように，直積遷移の実行条件が偽となる十分条件を判定したり，上述の不変式が成り立つことの十分条件を判定したりするために，プレスブルガー文真偽判定ルーチン [9] を利用している．判定すべき式は，ユーザ定義関数などを含むため，それらの関数を含む項を論理/整数変数に置換した式の真偽を判定する．これは元の式の十分条件を判定しているので，もし真あるいは偽と判定できない場合がある．その場合，新たな不変式や，証明すべき式の前提にユーザ定義関数に関する補題等を追加する必要がある．

提案手法を評価するため，PCI バスインターフェース回路の設計および検証を行った．証明すべき不変式の他にいくつか新たに不変式を設定するなど試行錯誤を行い，何度か直積制御部を生成したが，到達不能な状態をかなり削除することができ，実用的な時間で検証が行えた．

以下，2 において設計および検証の対象とする回路の回路記述について述べ，3 において提案する不変式の形式的検証法を述べる．4 において直積制御部の生成時に行う状態数削減方法を述べる．さらに 5 において例題とした PCI バスインターフェース回路の概要とその検証結果を述べる．

2 対象とする回路とその記述法

2.1 回路の記述法

対象とする回路モデルは，複数の有限状態部 (制御部) と，それらが制御するレジスタ，制御信号線等の部品間のデータ転送の集合である．複数の制御部を持つ回路を考えた場合，一つの制御部は一つの**拡張有限状態機械** (EFSM，レジスタ付 FSM) に対応する．各制御部は同期して並列に遷移を実行し，任意の部品を制御可能であるとする．部品は論理型や整数型だけでなく，ユーザが定義するデータタイプを取ることができる．これらを代数的言語を用いて，回路の各部品がどのように変化するかを指定する**動作内容** D と，どの条件が成り立つときにどの遷移を実行するかを指定する**実行制御** C で記述する [6, 7]．

回路の全部品の情報を含む抽象状態 s を導入する．回路の初期抽象状態は定数 init とする．回路の動作は抽象状態 s から遷移後の抽象状態への関数である**遷移関数** $t(s)$ で表す．任意の抽象状態 s における記憶を持つ部品 F_i の値は，抽象状態からその部品の値を取り出す**状態成分関数** $F_i(s)$ で表す．式には，論理演算 $\wedge, \vee, \neg, \rightarrow$ [1]，整数演算 $+, -, =, <$ と状態成分関数に加えて，ユーザが自由に定義したデータタイプやその上での関数も使用できる．公理 $A == B$ は，公理の変数にそのデータタイプの項を代入して得られる等式集合上の合同関係を表している．

動作内容 D は以下のように定義される (以下ではデータパスと呼ぶこともある)．

[定義 2.1] (動作内容 D)
動作内容 D は以下の 4 種類の形をしている公理のみを扱う

$$F_i(t_j(s)) == \mathrm{value}_{ij}(s) \quad (1)$$

$$F_i(\mathrm{init}) == \mathrm{const}_i \quad (2)$$

$$\begin{aligned} F_i(s) == {} & \text{if } (\mathrm{CONTROL}_k(s) = S_p) \\ & \quad \text{then } \mathrm{value}_{ip}(s) \\ & \text{else if } (\mathrm{CONTROL}_k(s) = S_q) \\ & \quad \text{then } \mathrm{value}_{iq}(s) \\ & \quad \vdots \end{aligned} \quad (3)$$

$$F_i(s) == \mathrm{value}_i(s) \quad (4)$$

ただし $\mathrm{value}_{ij}(s)$ および $\mathrm{value}_i(s)$ は関数 $\mathrm{CONTROL}_k$ を含まない，部品の値を表す関数である．□

式 (1)(2) はレジスタ，メモリ等の記憶を持つ部品の動作を記述する．式 (1) は抽象状態 s から状態遷移 t_j を実行した後の F_i の値が，抽象状態 s での各部品の値から求められる $\mathrm{value}_{ij}(s)$ になることを指定している．これを単に「遷移 t_j における F_i の動作内容は $\mathrm{value}_{ij}(s)$ である」ということもある．式 (2) は F_i の初期値が const_i であることを指定している．

式 (3)(4) は，制御信号線等の記憶を持たない部品の動作を記述する．式 (3) の第 1,2 行目は，制御部 k の値 (状態名) が S_p である場合に，F_i の値は $\mathrm{value}_{ip}(s)$ になることを指定している．これを単に「制御部 k の状態 S_p における F_i の動作内容は $\mathrm{value}_{ip}(s)$ である」ということもある．式 (4) は，制御部の値にかかわらず常に F_i の値は $\mathrm{value}_i(s)$ の値であることを指定している．これを「$F_i(s) == \mathrm{value}_i(s)$ は回路全体の動作内容」といい，回路の結線情報等を記述する．

[1] $A \rightarrow B$ は $\neg A \vee B$ を意味する

実行制御 C は各制御部ごとの遷移の実行順序を指定する．制御部 k の実行制御 C はいわゆる状態遷移図であらわすことができ，特別な関数 $\mathrm{CONTROL}_k$ と VALID_k を用いて以下のように定義する．

[定義 2.2] (実行制御 C)
$\mathrm{CONTROL}_k(t_i(s))$ は制御部 k における遷移 t_i が入射する状態名を指定する．$\mathrm{CONTROL}_k(\mathrm{init})$ は制御部 k における初期状態名を指定する．

$\mathrm{VALID}_k(t_i(s))$ は制御部 k における遷移 t_i の実行条件を指定する．実行条件にはユーザ定義関数の値や他の制御部の値を用いることができる．$\mathrm{VALID}_k(\mathrm{init})$ は真であり，これは初期状態から遷移可能であることを指定している．ただし，各制御部において $\mathrm{VALID}(t_i(s))$ が真になる遷移 t_i は高々一つであるとする． □

遷移 t_i に関する CONTROL, VALID の二つの関数の公理群によって，遷移 t_i どの有限状態にあってどの実行条件が成り立つ時，どの有限状態に遷移するかという情報を指定している．以下ではこれらの情報を状態遷移図で表すこととする．

2.2 複数制御部回路の動作

N 個の制御部を持つ複数制御部回路全体の動作を形式的に定義する．採用する回路モデルでは，各制御部は同期して並列に遷移を実行し，任意の部品を制御可能である．各制御部の遷移 $t_1,\ldots,t_N$ を並列に実行する回路全体の遷移を各制御部の遷移の組 $[t_1,\ldots,t_N]$ で表すものとする．制御部の遷移 t_i と回路全体の遷移 $[t_1,\ldots,t_N]$ に関して $t_i \in \{t_1,\ldots,t_N\}$ である場合，t_i と $[t_1,\ldots,t_N]$ は**対応関係**にあるという．

回路全体の遷移の実行条件および動作内容をそれぞれ以下のように定義する．

[定義 2.3] (回路全体の遷移の実行条件)
回路全体の遷移の実行条件 $\mathrm{VALID}([t_1,\ldots,t_N](s))$ は以下のように定義される．

$$\mathrm{VALID}([t_1,\ldots,t_N](s)) == \bigwedge_{i=1}^{N} \mathrm{VALID}_i(t_i(s))$$

□

これは回路全体で実行条件が成り立つ直積遷移は，各制御部で実行条件が成り立つ各遷移からなることを意味している．

[定義 2.4] (回路全体の遷移の動作内容)
各制御部の遷移 $t_1,\ldots,t_N$ の動作内容の集合のうち，左辺が $F_i(\cdots)$ となる公理がただ一つ存在する場合，回路全体の遷移 $[t_1,\ldots,t_N]$ の動作内容は，制御部の遷移の動作内容 $F_i(t_j(s)) == \mathrm{value}_{ij}(s)$ 中の遷移関数 $t_j(s)$ を $[t_1,\ldots,t_N](s)$ に置き換えた公理 $F_i([t_1,\ldots,t_N](s)) == \mathrm{value}_{ij}(s)$ の集合とする．

左辺に $F_i(\cdots)$ となる公理が存在しない場合，$F_i([t_1,\ldots,t_N](s)) == F_i(s)$ という公理が存在するものとする．

左辺に $F_i(\cdots)$ となる公理が二つ以上存在する場合や，$F_i(s)$ の値が定まらない場合[2]，回路にデータの衝突が生じているという． □

この定義では，扱う回路では各制御部が同期して並列にデータ転送を行うことを規定している．

各制御部の状態名が $S_1,\ldots,S_N$ である場合，回路全体の状態名はその組 $[\![S_1,\ldots,S_N]\!]$ で表す．

以降では回路の制御部の動作に注目するため，制御部の状態名を単に**状態**と呼ぶ．

3 不変式証明による回路の検証法

3.1 不変式の定義

不変式とは，プログラムの正当性の検証において導入された概念 [10] で，特定のプログラム状態において，初期値や変数間において成り立つべき関係を述べたものである．ここでは，回路が満たすべき機能上の性質を，各状態でのレジスタや制御信号等の部品間の関係，すなわち不変式ととらえることにする．以下では検証者が，回路に対して各部品間で成り立つであろうと考案した関係を**不変表明**とよぶ．

[定義 3.1] (不変式)
回路 A が，任意の初期抽象状態および任意の入力系列に対して，データの衝突が生じることなく動作するとき，不変表明 Q が常に真であると証明された場合，式 Q を回路 A における不変式であるという． □

不変表明および不変式は，以下の形のみ許される．

$$\begin{aligned} Q(s) == & \\ & \text{if } (\mathrm{CONTROL}_k(s) = S_i) \text{ then } Q_{ki}(s) \\ & \text{else if } (\mathrm{CONTROL}_k(s) = S_j) \text{ then } Q_{kj}(s) \\ & \vdots \end{aligned}$$

ここで $Q_{ki}(s)$ は 関数 $\mathrm{CONTROL}_k$ を含まない述語である．第 2 行目は制御部 k の状態 S_i において不変式 $Q_{ki}(s)$ が成り立つことを表している．

検証者は各制御部の任意の状態に不変表明を設定することができる．一般に，回路全体の状態に直接，不変表明を設定したいと考えるかもしれないが，ここでは，設計エントリが各制御部ごとになされるのと同様，検証エントリも各制御部ごとに行うという考えのもとに，検証法を提案した[3]．

3.2 不変式の証明法

複数の制御部を持つ回路において，与えられた関係式が不変式であることを証明する場合，具体的なレベルであれば，記号モデル検査 [3, 4] など

[2] 例えば $F_1(s) == F_2(s)+1, F_2(s) == F_1(s)+1$ と指定された場合．

[3] ただし不変式を $Q(s) ==$ if $(\mathrm{CONTROL}_1(s) = S_1 \wedge \ldots \wedge \mathrm{CONTROL}_N(s) = S_N)$ then $P(s)$ とすれば，回路全体の状態 $[\![S_1,\ldots,S_N]\!]$ に直接 $P(s)$ を設定することは可能である．

の方法を用いても行うことができる．本手法では，ユーザが定義した抽象的な関数を，動作内容における値の計算や制御部の実行条件に含むレベルを考えているので，以下に述べる状態に関する構造的帰納法を用いて，不変関係が成り立つことの十分条件を調べる方法を採用することにする．

状態に関する構造的帰納法は，単一の制御部を持つ回路に対する手法 [8] を基本としている．この手法による不変式の証明は以下の通りである．回路を一つの EFSM で表し，EFSM 上のいくつかの状態に不変表明が設定されている．初期状態も含めて不変表明が設定されている状態間で遷移系列を区切る．区切った遷移系列の始状態および終状態には，それぞれ不変表明 Q_a, Q_b が設定されているものとする．遷移系列に設定された動作内容，実行条件，ユーザが定義した関数に関する性質を表した公理 (等式関係) を P とした時，$\underline{Q_a \wedge P \rightarrow Q_b}$ の恒真性を判定する．初期状態からの全ての区切られた遷移系列に対して上記の証明が行われれば，状態に関する構造的帰納法により，任意の初期状態および任意の入力に対して，回路の各状態に与えられた全ての不変表明が不変式であるということができる．

複数の制御部を持つ回路において，不変式を証明する場合，回路の各制御部は任意の部品を制御可能であることから，その不変表明が設定されている制御部だけでなく，回路全体の動作に注目しなければならない．そこで本手法では，回路全体の動作を表す，元の複数制御部回路と等価な直積制御部を生成する．各制御部の動作内容や不変表明は，対応する直積制御部の各状態に存在するものとする．その直積制御部上で，文献 [8] と同様の方法を用いて検証作業を行う．直積制御部の生成法は **3.3** で述べる．

制御部 k の状態 S_i に設定されている不変表明 Q_{ki} が，状態 S_i における不変式であるかを証明する場合を考える．状態遷移の区分方法は以下の通りである．制御部 k上で状態 S_i から，初めて不変表明が設定されている状態まで，遷移を逆に探索する．直積制御部上における証明に用いる遷移系列は，この制御部 k で求められた遷移系列をもとにし，その途中に出現する式 (たとえば他の制御部上で設定された不変式) も証明に用いる．

検証手順は，以下の手順で行う．各制御部と等価な直積制御部は既に生成されているものとし，データの衝突はないものとする．

[Step 1] 制御部 k 上で状態 S_i から不変表明もしくは不変式が設定されている状態 (自分自身も含む) まで，遷移を逆向きに有限回たどる．結果として求まる制御部 k 上の遷移系列の集合を $TSset$ とし，その遷移系列の始状態に設定されている不変表明あるいは不変式を Q_{kj} とする．

また，有限回で不変表明もしくは不変式が設定されている状態までたどれない，すなわち閉路が存在する場合は，その閉路の途中状態に不変表明を設定する必要がある． □

[Step 2] [Step 1] で求めた制御部 k 上の各遷移系列 $ts_j \in TSset$ に対して，ts_j に対応する全ての直積遷移系列の集合 $PTSset_j$ を求める．各直積遷移系列 $pts \in PTSset_j$ に対して，式 (5) を生成する．

$$
\begin{aligned}
&Q_{kj} \\
&\wedge \text{遷移系列 } pts \text{ 中の実行条件および動作内容} \\
&\wedge \text{回路全体の動作内容} \\
&\wedge \text{遷移系列 } pts \text{ 上の状態に与えられた} \\
&\qquad \text{動作内容，補題，他の制御部の不変式} \\
&\rightarrow Q_{ki} \qquad (5)
\end{aligned}
$$

遷移系列 pts 中の実行条件および動作内容とは，遷移系列中の全ての遷移の実行条件と，動作内容の公理の "==" を "=" に置換した式の論理積である．ユーザが定義した関数に関する補題も判定式に加える．

各項であらわれる抽象状態 s には，遷移系列の始状態 S_{pts} からの実行順序を表す項 $T_n(\cdots T_1(S_{pts})\cdots)$ の適当な部分項を代入する． □

[Step 3] 式 (5) には，ユーザが定義した関数などが含まれているため，それらの関数を含む項を論理/整数変数に置換し，その変数が任意の値を取っても置換後の式 (全自由変数を全称記号で束縛した冠頭標準形) が真あるいは偽となるかを判定する．判定結果が真であったら，式 Q_{kj} が成り立つ場合に式 Q_{ki} は状態 S_i おける不変式であると判定する．この判定にはプレスブルガー文真偽判定ルーチン [9] を用いる．また初期状態における不変式は常に成り立つものとする． □

プレスブルガー文真偽判定ルーチンを利用したこの式判定は，式 (5) の十分条件を判定しているので，実際にはユーザが定義する関数に関する補題が不足して，不変式が証明できないという可能性がありうる．これらに対しては，いくつかの状態に不変式を追加したり，ユーザ定義関数に対する補題を設定したりして，試行錯誤によって証明する必要がある．

図 1 に例を示す．制御部 1 の状態 S_c における関係式 Q_{1c} を証明する場合を考える．制御部 1 の状態 S_a には不変式 Q_{1a} が存在するものとする．**[Step 1]** によって制御部 1 上の 3 個の遷移系列が求まる (制御部 1 上の点線)．その 3 個の遷移系列に対応する直積制御部上の遷移系列は計 6 個存在する (直積制御部上の点線)．その 6 個の直積遷移系列全てに対して式 (5) の十分条件の判定を行う．

3.3 直積制御部の生成

直積制御部の生成は，初期状態からの到達できる全状態を求める問題 (状態数え上げ)[11] に帰着できる．以下にそのアルゴリズムを示す．

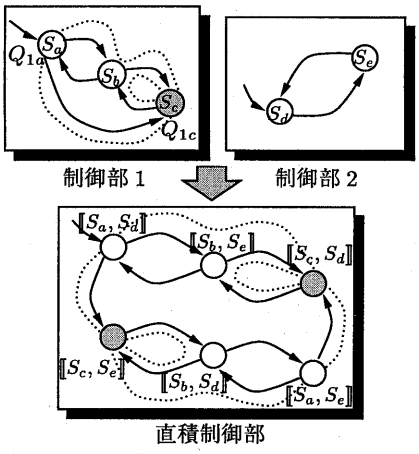

図 1: 不変式の証明

[手続き 3.1] (直積制御部生成のアルゴリズム)

```
procedure MAKE_PRODUCT(S_1^init, ..., S_N^init) {
  Reached = From = {[[S_1^init, ..., S_N^init]]};
  repeat {
    To = ∅;
    for each [[S_1, ..., S_N]] ∈ From
      for each [t_1, ..., t_N] ∈ TRANS([[S_1, ..., S_N]])
        To = To ∪ {NS([[S_1, ..., S_N]], [t_1, ..., t_N])};
    From = To − Reached;
    Reached = Reached ∪ From;
  } until (From = ∅)
}
```

ここで関数 TRANS($[\![S_1,\ldots,S_N]\!]$) は，直積状態 $[\![S_1,\ldots,S_N]\!]$ から出射する全ての直積遷移の集合を表す関数である．回路全体の実行条件および動作内容は，**2.2** で述べたように各制御部の遷移の組合せによって作成する．関数 NS($[\![S_1,\ldots,S_N]\!]$,$[t_1,\ldots,t_N]$) は，直積状態 $[\![S_1,\ldots,S_N]\!]$ から直積遷移 $[t_1,\ldots,t_N]$ を実行した後の直積状態を求める関数である．直積状態 $[\![S_1,\ldots,S_N]\!]$ の不変式は各状態 s_i に設定された不変式 Q_i の論理積とする．

関数 NS を求める際に，データの衝突が起こるかを検査する．$[t_1,\ldots,t_N]$ の動作を表す各公理と回路全体の公理から，データの依存関係を調べればよい．

直積到達可能状態の集合 Reached は有限であり，その要素数は **repeat** の繰り返しにおいて，単調増加することから，このアルゴリズムは必ず終了する．

4 直積制御部の状態数削減法

直積制御部の状態数は各制御部の状態数の積に比例するため，直積制御部の状態数爆発という問題が生じる．**3.2** で述べた不変式の証明法において，一つの不変式を証明するために真偽判定が必要な式数は，制御部の遷移系列に対応する直積遷移系列が何個存在するかにも依存する．したがって不変式の証明時間を減少させるためには，直積制御部の状態数をできるだけ少なくする必要がある．

本手法では直積制御部の生成時に，到達不能な遷移を削除してその遷移以降の状態数え上げを行わないことによって，直積制御部の状態数の削減をはかる．その削減方法として **[手法 1]** 遷移の実行条件による削除，**[手法 2]** 動作内容を前提とした遷移の実行条件による削除，という二つの手法を提案する．ここで行う式判定も，その式の十分条件をプレスブルガー文真偽判定ルーチンを用いて判定する．

4.1 [手法 1] 遷移の実行条件による削除

直積遷移 $[t_1,\ldots,t_N]$ の実行条件 VALID($[t_1,\ldots,t_N](s)$) が充足不能であれば，直積遷移 $[t_1,\ldots,t_N]$ が実行されることはない．したがって直積遷移 $[t_1,\ldots,t_N]$ を追加生成せず，それ以降の状態数え上げを行わない．この方法では実行条件の真偽判定が必要になるが，直積制御部の状態数を削減することができる．

手続き 3.1 の直積制御部生成のアルゴリズムを，直積制御部の生成時に状態数削減を行うように修正した差分を以下に示す．ここで関数 ELIMINATE($[t_1,\ldots,t_N]$) は，直積遷移 $[t_1,\ldots,t_N]$ を削減できる場合に真となる関数である．

[手続き 4.1] (直積制御部生成 (修正差分))

```
        ⋮
  for each [[S_1, ..., S_N]] ∈ From
    for each [t_1, ..., t_N] ∈ TRANS([[S_1, ..., S_N]])
      if (¬ELIMINATE([t_1, ..., t_N])) then
        To = To ∪ {NS([[S_1, ..., S_N]], [t_1, ..., t_N])};
        ⋮
```

図 2 に例を示す．状態数え上げが直積状態 $[\![S_a,S_d]\!]$ まで行われたとする．さらにこの直積制御部を生成すると $(R(s)\geq 1)\wedge(R(s)<0)$ という充足不能である実行条件を持つ遷移が生成される．よってこの遷移を削除し，それ以降の状態数え上げを行わない．

4.2 [手法 2] 動作内容を前提とした遷移の実行条件による削除

直積遷移 $[t_1,\ldots,t_N]$ が出射する直積状態における動作内容および回路全体の動作内容の全ての公理を "==" を "=" に置換した式の論理積を *Action* とする．動作内容 *Action* が成り立つ場合，直積遷移の実行条件 VALID($[t_1,\ldots,t_N](s)$) が充足不能であるなら，直積遷移 $[t_1,\ldots,t_N]$ が実行されることはない．したがって直積遷移 $[t_1,\ldots,t_N]$ を追加生成せず，それ以降の状態数え上げを行わない．この方法では，制御信号の値や回路の結線情報という回路動作を前提にするため判定する式長が大き

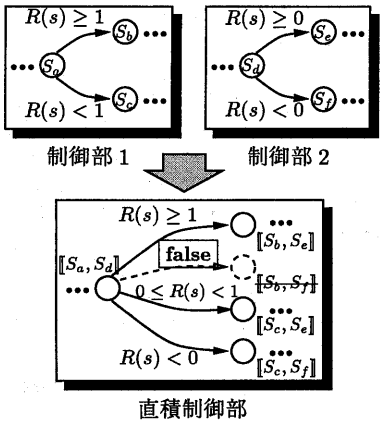

図 2: 遷移の実行条件による削除

くなり，式の真偽判定に時間がかかる．しかし 4.1 で述べた [手法 1] に比べて，より多くの遷移を削除することができる．この場合も直積制御部生成のアルゴリズムを手続き 4.1 と同様な修正を行う．

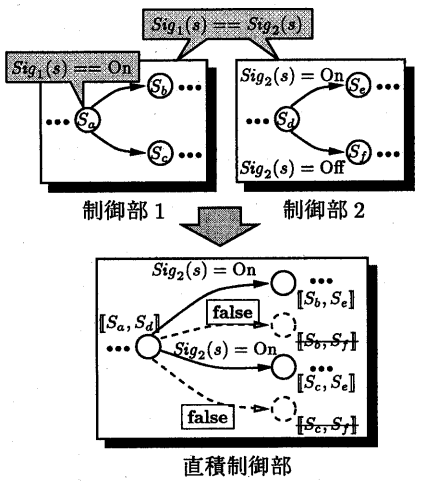

図 3: 動作内容を前提とした遷移の実行条件による削除

図 3 に例を示す．状態数え上げが直積状態 $[\![S_a, S_d]\!]$ まで行われたとする．制御部 1 の状態 S_a における制御信号の値 $Sig_1(s) == \mathrm{On}$ が動作内容として与えられている．制御部 2 の遷移は制御信号 Sig_2 の値によって条件分岐している．制御部 1 の遷移の実行条件は省略する．また回路の結線情報 $Sig_1(s) == Sig_2(s)$ が回路全体の動作内容として与えられている．$(Sig_1(s) = \mathrm{On}) \wedge (Sig_1(s) = Sig_2(s))$ を前提とした場合，$Sig_2(s) = \mathrm{Off}$ は充足不能である．よって 2 個の実行されることのない直積遷移を削除する．

5 例題: PCI バスインターフェース回路

5.1 PCI バスインターフェース回路の構成

パルテノン研究会が主催する ASIC デザインコンテストの規定課題として仕様を定められた「PCI バスインターフェース回路」[12] を例題として，その設計および検証を行った[4]．このインターフェース回路は，iAccess という比較的単純なデータ転送プロトコルと PCI バス信号プロトコルを相互変換する機能を有している (図 4)．この回路を用いることにより，周辺デバイス設計者は，PCI バス信号プロトコルの制御回路を設計することなく，デバイスの主機能の設計に専念することができる．

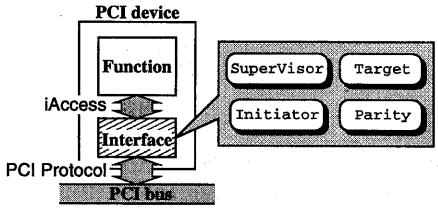

図 4: PCI バスインターフェース回路の概要

一つのインターフェース回路は 4 個の制御部 SuperVisor, Initiator, Target, Parity からなる．制御部 SuperVisor は回路のリセット動作および通常動作を制御する．制御部 Initiator は PCI のイニシエータ (アクセスの主体となる PCI デバイス) としてバストランザクションの生成動作を行い，制御部 Target は PCI のターゲット (アクセスの対象となる PCI デバイス) としてのバストランザクションに応答する．制御部 Parity はパリティ信号を生成する．

5.2 PCI バスインターフェース回路の設計および検証

インターフェース回路を設計し，同じ回路 A,B,C とバスからなるシステムにおいて，ある回路から別の回路へのデータ転送 (リード メモリサイクル) が正しく行われるかを検証した．ここでは，イニシエータとしてバストランザクションの生成を行う回路 A，そのトランザクションのターゲットとして応答する回路 B，そのトランザクションのターゲットとして応答しない回路 C と，役割を固定して考える．回路 A および回路 B のデバイスコア (インターフェース回路を介して PCI バスに接続されている回路部分) の役割をする送受信回路 (制御部 Tester, 状態数: 3) を新たに設計した．以降では，回路 A の制御部 Initiator を制御部 A.Initiator というように表記する．

送受信回路は，イニシエータ側の役割として，メモリリード要求を回路 A に送信し，データを受信するまで待ち状態で待機し，受信したデータがア

[4] ただしバスコントローラの初期化等を行うコンフィギュレーション機構は省略している．

ドレスに対応したものであるか(この関係を不変表明 Q とする)を検査する．

この送受信回路を含めた回路上で，動作途中でリセットが行われない状況でリードメモリ要求が発行された時，リードメモリサイクルが正常に動作するかを検証するには，不変表明 Q が不変式であることを証明すればいい．これを証明するのに，以下に示す(1)から(6)までの，リードメモリサイクルにおける実際の実行される動作手順(図5)[5]を参考にして，制御部 A.Initiator，制御部 B.Target，制御部 Tester の状態に，データ転送が適切に行われていることを表す関係式を設定する．それらが不変式であること示して，目的の不変表明 Q を証明する．

(1) 送受信回路が回路 A に対して iAccess プロトコルでリードメモリ(アドレス，バイトイネーブルマスク，転送ワード数が引き渡される)を要求する．
(2) 回路 A が PCI バスを介してリードメモリを要求し，回路 B がこの要求に応答する．
(3) 回路 B が送受信回路に対して iAccess プロトコルでリードメモリを要求する．
(4) 送受信回路が回路 B に対してアドレスに対応するデータを iAccess プロトコルで送信する．
(5) 回路 B が回路 A に対して PCI バスを介してデータを送信する．
(6) 送受信回路が回路 A に対して iAccess プロトコルでデータを送信する．

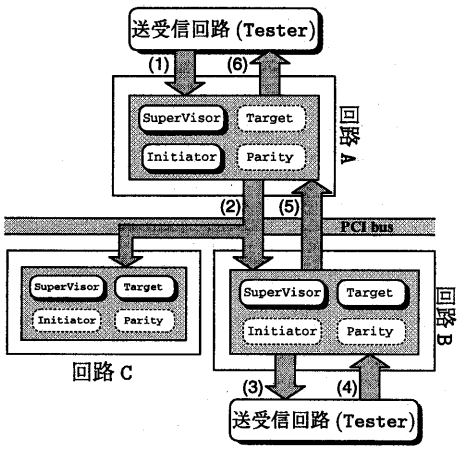

図 5: リードメモリサイクルの動作

今回は簡単のためイニシエータ側(回路 A)は制御部 SuperVisor(状態数: 2), Initiator(状態数: 6)を持ち，ターゲット側(回路 B,C)は制御部 SuperVisor(状態数: 2), Target(状態数: 5)を持つものとする．

以下では回路 A と回路 B の間で PCI バスを介してデータ転送が行われる，図 5 の (2) と (5) の部分の検証を説明する．図 6 にその動作を行う制御部 A.Initiator と制御部 B.Target の制御の様子を示す．ただしリセット信号などによる初期状態 i.Idle と t.Idle への遷移は省略し，今回の検証条件では実行されない遷移を点線で示す．

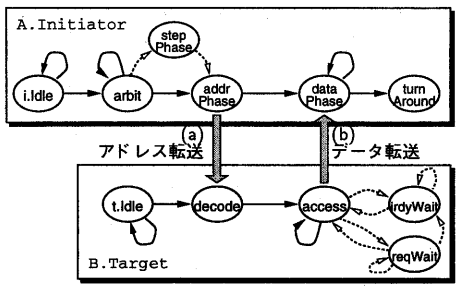

図 6: 制御部 A.Initiator と制御部 B.Target

図 5 の動作 (2) と (5) が正しく行われているかを検証するのに，以下の二つの関係式を不変表明として設定し，それが不変式であることの証明を行った．

(a) 制御部 B.Target の状態 decode における，回路 A に入力されるアドレス，バイトイネーブルマスク，転送ワードが回路 B に送信されていることを示す関係式
(b) 制御部 A.Initiator の状態 dataPhase における，回路 B がアドレスに対応するデータを回路 A に送信していることを示す関係式

これらの関係式は，成り立つべきいくつかの等式の論理積として表されている．またアドレスとデータの対応関係は，仮想的なメモリを考えて，そのメモリの機能を補題の形で与え，上記を証明している．

5.3 結果と評価

このインターフェース回路の設計および検証に要した時間は，証明すべき不変式の他に新たに不変式を設定したり，補題を設定したりする試行錯誤も含めて，約 55 時間であった．

提案した検証法を実現する支援系 [13] を用いて，検証作業を行った．検証は上記の関係式および補題を設定するだけで，後の証明作業は支援系により自動的に行うことができた．本支援系は図 7 のような GUI を用いており，検証者は視覚的に回路の設計および検証の状況を把握できる．また，不変式や補題等を追加設定して証明を行うなどの試行錯誤や，設計を修正した場合の検証状況の管理などを支援する機能を有する．

回路 A, B, C と送受信回路からなる制御部を 7 個持つ回路と，回路 A, B と送受信回路からなる制御部を 5 個持つ回路に対する，4.1 および 4.2 で述べた直積制御部の状態数削減の効果を表 1 に示す．計算時間の測定には PentiumII 266MHz, 128MB メモリを使用した．

[5] 図 5 では送受信回路は二つに分かれているが，実際は一つの制御部として設計している

	状態数削減手法	直積制御部†1 状態数	遷移数	生成時間	状態数削減に要した†2 式判定時間	式判定回数	不変式証明に要した†3 計算時間	式判定回数
5 制御部	なし	360	9450	129.6 秒	—		N/A	21825 回
	手法 1	86	979	6.1 秒	4.7 秒	2920 回	563.7 秒	1634 回
	手法 1 & 2	**14**	**25**	28.4 秒	28.1 秒	724 回	**77.4 秒**	**44 回**
7 制御部	なし	N/A	N/A	N/A	—		N/A	N/A
	手法 1	506	18439	925.3 秒	223.7 秒	92304 回	N/A	38014 回
	手法 1 & 2	**18**	**35**	**358.2 秒**	356.2 秒	4992 回	**581.8 秒**	**68 回**

表 1: 直積制御部の生成および不変式の証明の結果

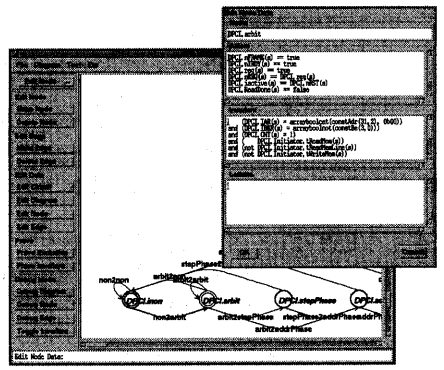

図 7: 支援系による回路の設計および検証の様子

表 1 の †1 の列は生成された直積制御部の状態数，遷移数，直積制御部生成に要した CPU 時間をそれぞれ表している．†2 の列は状態数削減を行うために要した式の真偽判定時間とその判定回数をそれぞれ表している．†3 の列は試行錯誤によって得られた不変表明を計 7 状態に設定した場合の，その証明に要した CPU 時間と式の真偽判定回数をそれぞれ表している．回路記述を決定し，これ以降の修正がないため，直積制御部の生成は一度だけである．不変式証明に要した計算時間は直積制御部生成に要した時間も含んでいる．

提案した状態数削減手法によって，生成された直積制御部の状態数および遷移数を小さく抑えることができている．手法 1 と手法 1 & 2 の場合で直積制御部の生成時間を比較してみると，5 制御部回路の場合は手法 1 & 2 は，より長い式を判定しているため，手法 1 に比べて時間を要している．しかし 7 制御部回路の場合は手法 1 は状態数削減が十分でないため，手法 1 & 2 の方がより短い時間で生成を行えている．また直積制御部の状態数および遷移数が小さくなることにより，不変式証明のために真偽判定が必要な式数が少なくなり，その証明を実用的な時間で行えた．

6 おわりに

複数の制御部を持つ回路を対象とした，不変式の形式的検証法を提案し，PCI バスインターフェース回路の設計および検証を行った．

本手法は判定すべき式の十分条件を判定しているので，補題や不変式を追加設定するなどの試行錯誤が必要になる．補題や不変式は，検証する回路に依存するため，今後はさまざまなタイプの複数制御部回路の設計および検証を行い，提案手法の有効性を評価したい．

参考文献

[1] Daniel Gajski, Nikil Dutt, Allen Wu, and Steve Lin: High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers (1992).

[2] Michael C. McFarland: Formal Verification of Sequential Hardware: A Tutorial, IEEE Trans. Comput.-Aided Design Integrated Circuits and Systems, Vol. 12, No. 5, pp. 633–654 (1993).

[3] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill: Symbolic Model Checking for Sequential Circuit Verification, IEEE Trans. Comput.-Aided Design Integrated Circuits and Systems, Vol. 13, No. 4, pp. 401–424 (1994).

[4] Kenneth L. McMillan: Symbolic Model Checking, Kluwer Academic Publishers (1993).

[5] N. A. Harman and J. V. Tucker: Algebraic Models and the Correctness of Microprocessors, Proc. of Correct Hardware Design and Verification Methods, Vol. 683 of LNCS, pp. 92–108, Arles, France (1993), Springer-Verlag.

[6] 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).

[7] 北道淳司, 東野輝夫, 谷口健一, 杉山裕二: 代数的手法を用いた同期式順序回路の段階的設計法, 信学論 A, Vol. J77-A, No. 3, pp. 420–429 (1994).

[8] 森岡澄夫, 北道淳司, 東野輝男, 谷口健一: 同期式順序回路の設計検証法, DA シンポジウム論文集, pp. 73–76 (1993).

[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, Vol. 9, No. 6, pp. 31–39 (1992).

[10] David Gries: The Science of Programming, Springer-Verlag (1981), (邦訳: 筧捷彦 訳, プログラミングの科学, 情報処理シリーズ 14, 培風館 (1991)).

[11] Gray D. Hachtel and Fabio Somenzi: Logic Synthesis and Verification Algorithms, Kluwer Academic Publishers (1996).

[12] パルテノン研究会: ASIC デザインコンテスト規定課題「PCI バスインターフェース」, http://www.kecl.ntt.co.jp/car/parthe/html/psfile/pci_spec.ps (1997).

[13] 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, 信学会総大 A-3-23, p. 128 (1997).