形式的手法によるキャッシュ・プロトコルの設計検証
— 超並列計算機 JUMP-1 への適用例 —

福島直人[†]　浜口清治[†]
富田眞治[†]　矢島脩三[†]

共有メモリ型超並列計算機 JUMP-1 のクラスタ内部のキャッシュ・プロトコルの設計検証に形式的手法を適用した事例について報告する。形式的手法はシミュレーションと異なり、要求される条件がどんな場合にでも成立することを保証することを目的とする。本事例では、時相論理式と呼ばれる、時間の概念を表現することのできる論理式により、JUMP-1 のメモリ・モデルにおける公理のいくつかを記述して、形式的検証ツール SMV を用いて検証を行なった。この結果、JUMP-1 のキャッシュ・プロトコルに見のがされていた設計誤りを発見することができた。

Formal Verification of a Cache Protocol Design
— A Case Study in the Massively Parallel Computer JUMP-1 —

Naoto Fukushima,[†] Kiyoharu Hamaguchi ,[†] Shinji Tomita[†]
and Shuzo Yajima[†]

We report on an attempt to apply a formal technique to the design verification of the cache coherence protocol of the Massively Parallel Computer JUMP-1. In this case study, we used a formal design verifier SMV, which takes finite state machines as designs and temporal logic formulas as specifications. SMV can guarantee exhasutively that the specifications are satisfied by the disign. We verified some axioms for the memory model of JUMP-1, and we found design errors in the protocol.

1. はじめに

現在我々は、超並列計算機 JUMP-1[1)]の開発・設計に取り組んでいる。JUMP-1 は共有メモリ型の並列計算機であり、ハードウェアでキャッシュのコヒーレンスを保っている。しかし、JUMP-1 のキャッシュ・コヒーレンスのプロトコルは複雑であり、シミュレーションによる手法でその正当性を確かめるのは困難であるので、適切な手法を用いた検証が必要となる。本稿は超並列計算機 JUMP-1 のキャッシュ・プロトコル設計に対して、形式的手法を用いて検証を行なった事例について報告する。

検証にあたっては形式的検証ツール SMV[3)]を用いる。SMV は、有限状態機械 (順序回路) 記述言語で書かれた設計記述と、時間に関する表現を許した論理である時相論理[5)]で書かれた仕様記述を入力として、与えられた仕様記述を設計記述が満足しているかを検査するツールである。また SMV は設計に誤りがある場合には、仕様記述を満たさない動作系列を出力することができる。しかし、SMV が扱うことができる回路の規模は状態変数の数にして 50 から 70 程度なので、このツールを用いて大規模な回路を検証するには何らかの意味で回路を抽象化する必要がある。本事例においては、検証の対象を JUMP-1 のクラスタ内部に限定する等の方法により検証が可能となるようにしている。

近年、計算機の設計の際にはメモリの動作を定めたメモリ・モデルを記述するようになってきており、JUMP-1 においてもメモリ・モデルが定められている。この場合、キャッシュ・プロトコルはメモリ・モデルにのっとって動作する必要があり、それを検証する必要がある。しかし、過去のキャッシュ・プロトコル検証は、メモリ・モデルのことには触れられておらず、満たすべき仕様は非常に簡素化されていた。そこで本事例では、ある程度メモリ・モデルに基づいた仕様を記述し、より厳密な設計検証を目指した。

以下、2節で形式的検証について概要を説明し、3節で超並列計算機 JUMP-1 の概略について述べる。その後、4節で、本検証における仕様記述と設計記述に

[†] 京都大学工学部
Faculty of Engineering, Kyoto University

ついて、最後に5節で検証結果について述べる。

2. 形式的検証

形式的検証手法は設計記述と仕様記述を与えて、設計が仕様を満足するかどうかを完全に保証しようとするものである。本事例では形式的検証のためにCarnegie Mellon 大学で開発された SMV と呼ばれるシステムを利用した。SMV の入力は大きく次の二つに分けられる。

- 設計記述 :SMV の記述言語により、有限状態機械 (順序回路) として与える。
- 仕様記述 :時相論理 CTL5) の論理式として与える。

CTL(Computation Tree Logic) は、通常の命題論理を拡張して、時間に関する性質を書くことができるようにした時相論理の一種である。CTL には通常の論理演算子 (論理和 ($+$)、論理積 ($\cdot$)、論理否定 ($\neg$)、含意 ($\rightarrow$) など) に加えて、時相演算子と呼ばれる特別の演算子を持つ。時相演算子は有限状態機械の動作系列 (入力に対する状態の遷移系列) についての性質を書くためのものである。時相演算子は有限状態機械の動作系列に関する演算子 (**A**,**E**) と時間に関する演算子 (**F**,**G**,**X**,**U**) を組み合わせたものが用いられる。それらの直観的な意味は次のとおりである。

$\mathbf{EX}\phi$ ($\mathbf{AX}\phi$)： 現在の状態に対して、ある (すべての) 次の状態でϕが成立する。

$\mathbf{EG}\phi$ ($\mathbf{AG}\phi$)： 現在の状態から始まる、ある (すべての) 動作系列上の全ての状態において、常にϕが成立する。

$\mathbf{EF}\phi$ ($\mathbf{AF}\phi$)： 現在の状態から始まる、ある (すべての) 動作系列上において、いつかϕが成立する状態がある。

$\mathbf{E}[\eta\mathbf{U}\phi]$ ($\mathbf{A}[\eta\mathbf{U}\phi]$)： 現在の状態からのある (すべての) 動作系列において、いつかϕが成立する状態があり、かつ最初にϕが成立するまではどの状態でもη が成立し続ける。

これらを用いると、例えば「もし信号 req が high になると、必ずいつかは信号 ack が high になる」は、

$$\mathbf{AG}(req \rightarrow \mathbf{AF}\,ack)$$

のように書ける。

設計記述をあらわす有限状態機械はSMV 専用の記述言語を用いて記述する。

SMV の検証アルゴリズムの原理は、有限状態機械を状態遷移図の形に展開して、仕様の成否を状態遷移図上で全探索によって判定するというものである。ただしこの手法では、設計が大きくなると状態数が非現実的に大きくなるため、二分決定グラフ6)を利用した状態空間の圧縮とその上での検証アルゴリズムを用いている。詳細は文献4),5) を参照されたい。

設計記述が与えられた仕様記述を満たしていない場合、SMV はその反例、すなわち仕様を満足しない動作系列を出力することができる。

3. 超並列計算機 JUMP-1 の概略

3.1 構 成

JUMP-11)は、クラスタ構造を採用した共有メモリ型並列計算機である。各クラスタは、RDT7)と呼ばれるネットワークで結合されている。

クラスタの構成を図1に示す。構成要素としては、基本的に4 つのプロセッサ ·ユニット、2 つのメモリ ·ユニット、及び他クラスタとのインタフェースとなるルータがあげられる。プロセッサ ·ユニットとメモリ ·ユニットはクラスタ ·バスと呼ばれるバスで接続されている。この図に示されるようにJUMP-1 では、主記憶が各クラスタに分散して配置されている分散共有メモリを採用している。

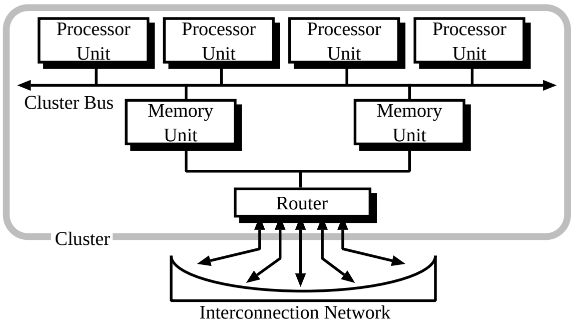

図1 クラスタの構造

後述するが、本事例はプロセッサ ·ユニット部の検証を中心に行う。プロセッサ ·ユニット部の構成を図2に示す。プロセッサ ·ユニットは、1 次キャッシュ内蔵のプロセッサ (Sun SuperSPARC+)、および 2 次キャッシュからなる。

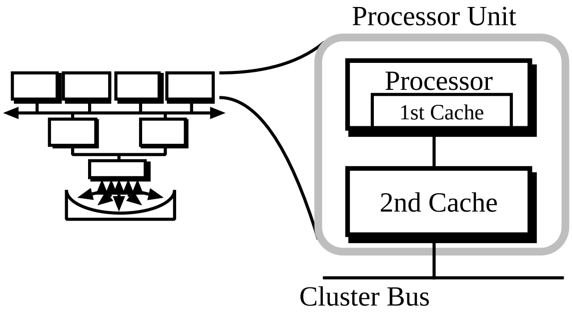

図2 プロセッサ ·ユニットの構造

3.2 メモリ ·モデル

メモリ ·モデルは、メモリ ·システムがプログラマにとってどのように動作しているように見えるかを記

述したものである。

JUMP-1ではメモリ・モデルとして、Weak Ordering9)に基づくメモリモデルを採用している。このモデルは、あるプロセッサにおけるメモリ・アクセスに関して、(1) 同期命令を挟んだ2つのメモリ・アクセスはその順序に関してプログラム・オーダを保証する、(2) 同期命令を挟まない2つのメモリ・アクセスはその順序に関してプログラム・オーダを保証しない、というモデルである。

本稿においては、このメモリ・モデルを幾つかの公理を用いて表し、それを仕様として設計検証を行う。

3.3 キャッシュ・プロトコル

先に述べたように、JUMP-1ではハードウェアによるキャッシュ・コヒーレンス制御を行う。コヒーレンス制御はクラスタ内ではスヌープ方式、クラスタ間はディレクトリ方式で行う。

キャッシュ・プロトコルを特徴づける項目として、(1) コヒーレンス制御の度にメモリへの書き戻しを行う / 行わない、(2) キャッシュへの書き込みの際に他のキャッシュを無効化する / 更新する、(3) キャッシュ・ミスヒットにより生ずる要求に対して誰が応答するか、がある。これらについて、JUMP-1では以下の方式を採用している。

- メモリへの書き戻しについて
  コヒーレンス制御がクラスタ内で閉じている場合はメモリへの書き戻しは行わず、クラスタ間にまたがっている場合はメモリへの書き戻しを行う。
- 書き込み時の無効化/更新について
  キャッシュ・ブロックごと、またアクセスごとにどちらで行うかを切り替え可能である。
- キャッシュ・ミス時の要求に対する応答
  JUMP-1キャッシュ・プロトコルはオーナシップ・プロトコルを採用している。すなわち、クラスタ内のバスにShared Read Request , Exclusive Read Requestが流れたとき、要求に対する応答はオーナシップを所有しているプロセッサ・ユニットが行う。
  JUMP-1において、オーナシップは最後に当該キャッシュ・ブロックに対する操作を行ったプロセッサ・ユニットが所有する。すなわち、オーナシップは、あるプロセッサ・ユニットが読み出し、書き込みを行うと、そのプロセッサ・ユニットにオーナシップが移動する。

次にプロトコルの詳細について述べる。後述するが、検証の結果、一部に設計誤りが発見された。しかし、検証で発見された設計誤りを明確にするため、ここでは設計誤りを含む仕様をそのまま記述する。

3.3.1 2次キャッシュの状態

2次キャッシュは5つの状態、Invalid, Exclusive Dirty, Local Shared Clean, Local Shared Dirty, Global Shared Cleanを持つ。これらの各状態の意味は、表1に示す通りである。クラスタ内でキャッシュ・ブロックが共有されている場合、最後に書き込みを行ったプロセッサ・ユニットがLocal Shared Dirtyで、残りはLocal Shared Cleanである。また、先に述べたように、コヒーレンス制御がクラスタ間にまたがっている場合は、メモリへの書き戻しを行うため、Global Shared Dirtyという状態は存在しない。

表1　2次キャッシュの状態

状態	説明
Invalid	当該ブロックは無効化されている。
Exclusive Dirty	当該ブロックの最新データは、自デバイスのキャッシュにしか存在しない。
Local Shared Clean	当該ブロックはクラスタ内で共有されている可能性がある。
Local Shared Dirty	当該ブロックはクラスタ内で共有されている可能性がある。内容は主記憶とは異なり、主記憶への書き戻しの責任を持つ。
Global Shared Clean	当該ブロックはクラスタ間で共有されている可能性がある。内容は主記憶と等しい。

3.3.2 トランザクション

あるメモリ・アクセスによって開始されるデータの移動、及びそれに伴うシステムの状態遷移をトランザクションと呼ぶ。トランザクションは、データの移動や状態遷移の要求を行う要求パケットと、それに対する0個以上の応答パケットからなる。

本事例の検証対象であるJUMP-1のクラスタ・バスには全部で31種類のパケットが存在する。そのうち、本稿に関係するパケットを以下に示す。下線の引いていないパケットは要求パケット、引いてあるパケットは応答パケットである。

Shared Read Request : 読み出し要求。

Exclusive Read Request : 読み出し要求兼無効化要求。

Write Back Request : キャッシュブロックのリプレースに伴う、メモリへの書き戻し要求。

Update Direct Request : 更新要求。

Invalidation Request : 無効化要求。

Read Reply : Shared Read Requestに対する応答。

Exclusive Read Reply : Exclusive Read Requestに対する応答。

また、JUMP-1ではスプリットトランザクションを採用している。すなわち、要求パケットと応答パケットの間に、別のトランザクションのパケットがバスを流れても良い。

4. キャッシュ・プロトコルの仕様記述と設計記述

4.1 検証対象の抽象化の方針

SMV は状態変数の数にして 50 から 70 程度の回路なら扱うことができる。しかし、JUMP-1 全体構成をそのまま記述した場合の変数の数はこれをはるかに越える。したがって、形式的検証を行うためには、何らかの形で設計の抽象化を行う必要がある。今回、抽象化は以下の方針で行うことにした。

設計記述の規模に関して、本検証では単一クラスタ構成を対象とすることにする。これは、現在の形式的検証ツールでは非常に大きな設計を扱うことは困難であるが、JUMP-1 では単一クラスタ内にプロセッサ・ユニットが 4 つあり、ある程度の並列度があることから、検証対象を単一クラスタ構成としても有用な結果が得られると考えられたからである。また、クラスタ内の構成は以下のようにする。クラスタ内のプロセッサ数はまず 2 つとした設計記述に対して検証を行い、順次増やしていく。メモリ・ユニットは本来は図 1 に示すように 2 つであるが、本検証においては簡単のため 1 つとしている。

また、設計記述の抽象度に関しては、本事例ではプロセッサ・ユニット部の設計を検証の対象とすることから、プロセッサ・ユニット部の設計記述は必要な限り詳しく、メモリ・ユニット部の設計記述は簡素化することにした。

4.2 仕様記述

本節では、JUMP-1 のキャッシュ・プロトコルが満たすべき仕様をどのように表すかについて述べる。

先にも述べたように JUMP-1 では、Weak Ordering に基づくメモリ・モデルを採用している。JUMP-1 メモリ・モデルのセマンティクスは、SPARC アーキテクチャV8[10]に示されているメモリ・モデルのセマンティクスと同様に以下の公理によって表される。

オーダ　ストア操作はメモリ上において全順序関係が存在する。

終了　すべてのストアは最終的に終了する。

値　ロード操作により得られる値は、同一アドレスに対する最近のストアの値である。ここで最近の値とは、

(1) 自分自身が発行し、かつまだメモリにはあらわれていないストアの値。このようなストアが存在しなければ、

(2) メモリに一番最後にあらわれたストアの値。

を意味する。

ストア・ストア　プログラム・オーダで同期命令により分離された 2 つのストアはメモリ・オーダにおいてプログラム・オーダと同一の順序であらわれる。

JUMP-1 メモリ・モデルにはこれ以外にもアトミック性と呼ばれる公理が存在するが、この公理は仕様記述が難しいため、この公理の検証は行なわない。

本事例においては上記の公理を以下のように解釈して検証を行う。

(1) オーダ

この公理の成立に関する検証は行なわない。なぜなら JUMP-1 ではストアはクラスタ内においてはクラスタ・バスで順序付けされるが、この場合全順序関係が存在するのはバスの性質上自明であるからである。

(2) 終了

もしあるプロセッサがストアを行ったときキャッシュの状態が Exclusive Dirty の場合は、キャッシュにデータを書き込むだけで終了する。つまりこの場合の動作を正しく設計するのは容易である。よって、設計記述の規模を削減するためにも、キャッシュの状態が Exclusive Dirty の時のストアは必ず終了するように設計されているものとし、検証の対象外とする。本事例で検証するのはキャッシュの状態がそれ以外の場合、すなわちストアに伴うトランザクションが発生した場合、いずれ必ず他のキャッシュは無効化/ 更新されるという性質である。

(3) 値

上記の終了の公理が成立しているならば、ロードを行ったときにキャッシュにヒットした場合、キャッシュの内容は一番最近の値であるということができる。なぜなら、自プロセッサ・ユニットのキャッシュの内容はすべての当該ブロックへのストアの度に更新されているからである。つまり、終了の公理が成立しているという前提の下ではロード時にキャッシュ・ヒットした場合はキャッシュの値を返すだけで値の公理が成立しているということができ、この場合の動作も正しく設計するのは容易であるため、検証の対象外とする。よって、本事例においてはキャッシュの状態が Invalid である時にロードを行うと当該プロセッサ・ユニットにその時点における最新の値が返ってくるということを検証することにする。ところで「その時点における最新の値」に関して、あるキャッシュがロードの要求を出したとき、それに対して応答を行うのは、3.3節で示したオーナシップを持つプロセッサであるが、オーナシップの定め方からそのプロセッサはその時点における最新の値を持っていると予想される。よって、設計記述の規模の削減のためにも、ロード要求に対して応答が返ってきたとき、その値はその時点における最新の値であるとし、検証するのは要求に対する応答が必ず得られるということにする。

(4) ストア・ストア

この公理は JUMP-1 で用いるプロセッサ SuperSPARC+は同期命令の発行をそれ以前に発行したストアがすべて終了するまで待つことから、成立するのは自明である。

以上で示したように、本事例は終了と値の公理の成立を検証する。これらの CTL 記述は次節で述べる。

4.3 設計記述

以上に示した抽象化方針と満たすべき仕様記述に基づき、どのような設計記述を作成したかを示す。

まずプロセッサ・ユニットの内部について、本事例では 4.2節で述べたように、プロセッサのロードやストアに伴う処理が当該プロセッサ・ユニット内に閉じている場合は検証の対象外とすることにした。そこで本検証では、キャッシュの状態が Exclusive Dirty の時はストアが発生しない、またキャッシュの状態が Invalid でないときはロードは発生しないものとし、プロセッサとプロセッサ–2 次キャッシュ間は、設計記述を省略している。このことにより 4.2節に示した仕様を CTL で表すと、終了の公理は次のようになる。

$$\begin{aligned}&\mathbf{AG}(\ PR^1_{ST}\ \&\ PR^1_{data}=x\ \&\ !(PR^2_{state}=Inv)\\&\qquad\rightarrow\mathbf{AF}(PR^2_{state}=Inv)\mid\mathbf{AF}(PR^2_{data}=x)\ )\end{aligned}$$

値の公理については、

$$\mathbf{AG}(PR^1_{LD}\rightarrow\mathbf{AF}(\ !(PR^1_{state}=Inv)\ )\ )$$

と表せる。ただし上記式中においては

$PR^i_{\{ST,LD\}}$: プロセッサ i がストア/ロードを行った

PR^i_{data} : プロセッサユニット i のキャッシュの値

PR^i_{state} : プロセッサユニット i のキャッシュの状態

を意味している。

さらに、設計記述の状態数を削減するため以下のような抽象化も行った。

(1) クラスタ・バスのアービトレーションはランダムに行われる。

(2) 各キャッシュにおけるブロック数は 1。

(3) キャッシュ・ブロックのサイズは 1 ビット。

これらは文献11)における Futurebus+のプロトコル検証においても用いられた抽象化である。

5. 実行結果

前節で述べた方法により検証を行った結果、JUMP-1 キャッシュ・プロトコルが 4.2節で述べた仕様を満たしていない設計誤りが発見された。本節では、発見された設計誤りとそれに対するプロトコルの変更点をのべる。また、プロセッサ・ユニット数の違いにより実行時間などにどのような差が生じたかについても述べる。

5.1 発見された設計誤り

値、終了の公理それぞれを満足しない動作系列が発見された。

(1) 値の公理を満たさない動作系列

次に示す動作系列は値の公理を満たさない。あるプロセッサ・ユニット (P_A) の状態が Invalid のときにロードを行ったとする。この時、他のプロセッサ・ユニット (P_B) のキャッシュが Exclusive Dirty であった場合、P_Bが応答を行なう。ここで、P_Aのキャッシュは Local Shared Clean になり、P_Bのキャッシュは Local Shared Dirty になる。一方で、オーナシップは P_B から P_Aに移る。ところで、キャッシュの状態が Local Shared Clean の場合は、そのブロックはリプレースされうる。よって、P_Aがリプレースされて、その後 P_Aがロードを行った場合、P_Bはオーナシップを持っていないので応答を行なわない。また、メモリに最新のデータは存在しないので、メモリも応答を行なわない。よって、応答は永久に返ってこないことになる。これは、Dirty なキャッシュブロックがオーナシップを持っていないことが原因である。よって、ロードによる応答が返ってきたとき、応答を行なったキャッシュブロックが Exclusive Dirty または Local Shared Dirty だった場合は、ロードを行なったプロセッサのキャッシュブロックの状態を Local Shared Dirty に、それ以外の場合は Local Shared Clean となるようにする。また、応答を行ったプロセッサのキャッシュブロックの状態は Local Shared Clean となるようにする。この変更を行い再度検証を行うとこの問題は生じないことが確認された。

(2) 終了の公理を満たさない動作系列

終了の公理は次のようなときに満たされない場合があることが明らかとなった。まず、プロセッサ・ユニット A(P_A) のキャッシュがオーナシップを持っていて、プロセッサ・ユニット B(P_B) の状態が Invalid であるとする。このとき P_B がストアを行ったとする。JUMP-1 では更新系のストアの場合ライト・ミスはリード・ミス + ストア・ヒットとして扱われるため、まず読み出し要求のパケットが生成され、P_Bから送出される。この読み出し要求には P_Aが応答を行う。P_Bはこれを受信した後、更新要求パケットを生成する。しかし、もし P_Aがこれとほぼ同時に、読み出し応答とは別に、プロセッサのストアにともなう無効化要求のパケットを生成しており、P_Aの無効化要求が P_B の更新要求よりも先にクラスタ・バスに送出されることがありうる。この場合、P_Bのキャッシュが無効化されるので、更新要求は読み出し要求に変更される。また P_Aはストアにともないオーナシップを得る。この時点での状態は最初の状態と等しい。つまり、以上の動作を無限に繰り返す場合には、P_B のストアが永久に終了しないことになる。

JUMP-1 のキャッシュ・プロトコルが終了の公理を満たしているというためにはこれは直される必要がある。しかし上記に示した系列は、P_Aが当該アドレスに対する無効化型ストアを無限に繰り返す場合にかぎって発生し、途中でストアが行われなくなった場合には P_B のストアは完了する。一方、あるプロセッサが、あるアドレスへの無効化型ストアを無限に繰り返して行う

という状況は通常ほとんど起こりえないと考えられること、また、実際には、この動作系列に陥らないようにするにはプロトコルの簡単な変更で直すことはできないため、修正は容易ではないことから、設計の修正は行なっていない。ただし、JUMP-1 においてはメモリ·アクセスが一定時間経っても終了しないことを検知するタイムアウト検出機構を実装しており、上記の動作系列に陥った場合はこれにより検出できる。

5.2 プロセッサ·ユニット数の違いによる差

クラスタ内のプロセッサ·ユニットの違いにより、SMV の実行時間や出力結果がどれだけ変化したかを示す。なお、本検証は Sun SPARCStation20、主記憶 512MB のマシンの上で行い、また SMV に対する実行オプションはすべて同じである。

表 2に各プロセッサ·ユニット数ごとの実行時間と使用メモリを示す。この表に示されるように、プロセッサ·ユニット数が 1 つ増えると、実行時間と使用メモリが大幅に増加する。今回はプロセッサ·ユニット数が 4 つの場合は実行結果を得ることができなかった。

表 2 SMV の実行時間と使用メモリ

プロセッサ·ユニット数	実行時間 (分)	使用メモリ (MB)
2	28	9.0
3	647	71

SMV の出力結果についてはプロセッサ·ユニット数が 2 つの時と 3 つの時とでは違いはなかった。このことから、プロセッサ·ユニット数が 4 つの時も出力結果に違いはないと予想される。

6. おわりに

本稿における検証はメモリ·モデルに完全に基づいたものではなく、設計記述も実設計データを簡単化したものを用いている。また、5.2節の結果からも、SMV を含めて、現在の形式的検証ツールの能力では複数クラスタ構成を取り扱うことはできないと考えられる。このように、本事例で紹介した時相論理による形式的手法によって、キャッシュ·プロトコル設計における全ての検証問題を解決することはやはり難しいと予測される。しかし、本事例を通じて示したように、形式的設計検証は、その網羅性によって、通常容易に発見することのできない設計誤りを発見しうる、という意味において非常に有効な手段であり、より抽象的なレベルでの設計記述、また、初期段階における設計に対しては、有効な検証手法の１つであると考えられる。

謝辞

本研究は一部、試験研究 (A)(1) 課題番号 06508001 による。

参考文献

1) 平木 敬 他: 超並列プロトタイプ計算機 JUMP-1 の構想, 情処研報 93-ARC-102 pp.73-84,1993 年 10 月.

2) 富田 眞治:コンピュータアーキテクチャI, 丸善株式会社,1994 年.

3) K. L. McMillan: *Symbolic Model Checking: An Approach to the State Explosion Problem,*PhD thesis, Carnegie Mellon University, 1992.

4) J. R. Burch et al.: *Sequential Circuit Verification Using Symbolic Model Checking,* Proc. 27th Design Automation Conference, pp.46-51, June 1990.

5) 平石 裕実、浜口 清治 :論理関数処理に基づく形式的検証手法, 情報処理, Vol.35,No.8,pp.710-718.1994 年 8 月.

6) R. E. Bryant: *Graph-based algorithms for boolean function manipulaton* IEEE Trans. on Computers, C-35(8), 1986.

7) 相互結合網とルータ. 文部省重点領域研究. 超並列原理に基づく情報処理基本体系. 第 3 回シンポジウム予稿集.pp.257-279. 1993 年 9 月.

8) L. Lamport: *How to make a multiprocessor computer that correctly executes multi process programs* IEEE Trans. on Computers, Vol.28, No.9 pp.241-248, Sep. 1979.

9) M. Dubois, C. Scheurich, and F. Briggs: *Memory access buffering in multiprocessors* Proc. 13th Int'l Symp. Comp. Arch., pp.434-442, June 1986.

10) 相越 克久、田中 長光 訳 多田 好克 監訳 SPARC アーキテクチャ·マニュアルバージョン 8. SPARC International,Inc. 1992 年 11 月.

11) E.M.Clarke, O.Grumberg, H.Hiraishi, S.Jha, D.E.Long ,K.L.McMillan, and L.A.Ness. “Verification of the Futurebus+ cashe coherence protocol. ” L.Claesen,editor,*Proceedings of the Eleventh International Symposium on Computer Hardware Description Languages and their Applications.* North-Holland, April 1993.