計算機アーキテクチャ 119－38
（1996．8．29）

超並列計算機 JUMP-1 のキャッシュシステムの論理設計検証

舟本一久† 福島直人† 五島正裕†
森眞一郎† 中島浩† 富田眞治†

メモリアクセスを並列実行し，そのレイテンシを隠蔽するメモリアクセスバッファリング技術は，メモリアクセスの高速化だけでなく，キャッシュシステムの複雑化ももたらした．

このような複雑なキャシュシステムの設計検証には，形式的検証手法や通常のシミュレーションによる設計検証手法は適さない．複雑過ぎて形式的に検証することが出来ず，出力が非決定的であるため，出力の期待値が一意に定められないためである．

我々は，このようなキャッシュシステムの設計検証として，シミュレーションを実行し，対象がメモリモデルを満たしているか調べることで設計検証を行なう．本手法では，メモリモデルを満たす非決定的な出力の期待値を参照モデルと呼ばれるモジュールに動的に求めさせる．

A Logic Design Verification Method for the Cache System of Massively Parallel Computer JUMP-1

KAZUHISA FUNAMOTO,† NAOTO FUKUSHIMA,† MASAHIRO GOSHIMA,†
SHIN-ICHIRO MORI,† HIROSHI NAKASHIMA† and SHINJI TOMITA†

The memory-access-buffering technique hides the memory access latency by executing memory accesses in parallel. But at the same time it complicates cashe systems.

To such systems, it is almost impossible to apply the formal design verification method or the design verification methods by excuting simulation. Bacause it is too complicated to apply formal ones, and its output sequences are non-deterministic.

In this paper, we describe the design verification method by excuting simulation with the reference model that dynamically calculates all acceptable output sequences. By calculating them dynamically, the reference model deals with the non-deterministic situations efficiently.

1. はじめに

本稿では，現在我々が開発している超並列計算機 JUMP-1 のキャッシュシステムに適用した論理設計検証手法について述べる．JUMP-1 のキャッシュシステムの設計検証には，状態数の多さ，出力の非決定性などの問題がある．そこで我々はシミュレーションを行ない，対象に対する入出力の関係がメモリモデルを満たすことを確認することで検証を行なう手法を用いた．

以下本稿では，2章で，問題の概略について述べた後，3章にてより具体的な手法について説明する．

2. 背景

本章では，背景として JUMP-1 キャッシュシステムの概要について述べ，その設計検証を行なう上での問題点を明らかにした後に，その解決の方針を述べる．

† 京都大学大学院工学研究科情報工学教室
Department of Information Science, Graduate School of Engineering, Kyoto University

2.1 JUMP-1 キャッシュシステムの概要

本節ではまず，本稿の対象である JUMP-1 キャッシュシステムについて，設計検証を行なう上で特徴的な点について述べる．

JUMP-1 は，クラスタ構造を持つ共有メモリ型の並列計算機である．図1に JUMP-1 のクラスタを示す．クラスタは主に，4つのプロセッサ・ユニット，メモリ・ユニット，および，ネットワーク・インタフェースで構成される．プロセッサ・ユニットは，要素プロセッサ（PE），外部2次キャッシュ，及び，2次キャッシュ・コントローラからなる．PE には，1次キャッシュを内蔵する市販の RISC プロセッサ，SuperSPARC を用いる．4つのプロセッサ・ユニットとメモリ・ユニットはバスによって結合されており，スヌープ方式によってキャッシュ・コヒーレンス制御が行なわれる．

現在我々は，クラスタのコヒーレント・キャッシュ・システムの設計を行なっており，本稿ではこの部分の論理設計検証について述べる．

メモリアクセス・バッファリング

JUMP-1 キャッシュシステムは，ハードウェアに対

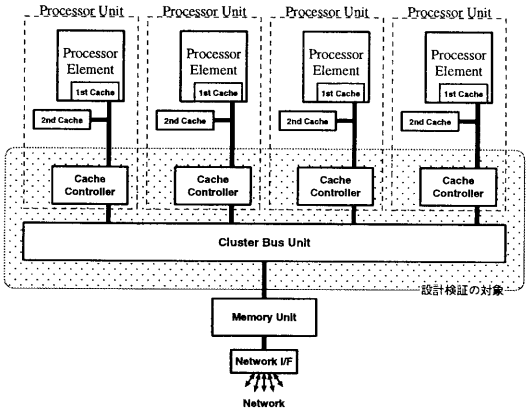

図1　JUMP-1 クラスタの構成

する制限の弱いメモリモデルに基づいて，**メモリアクセス・バッファリング**を実装する（以下では，単にバッファリングとする）．バッファリングとは，1つのメモリアクセスを複数のステージに分割し，各ステージごとに並列に処理する技術である．この技術を用れば，1つのアクセスをパイプライン処理することでそのレイテンシを隠蔽し，複数のアクセスを並列に処理することでスループットを向上させることができる．

2.2　設計検証における問題点

一方で，バッファリングを行なうと，メモリアクセスの開始から完了までが不可分に行なわれないため，設計検証において，**状態数の増加** と **出力の非決定性** という2つの問題を生じる．

状態数の増加

バッファリングを行なうと，アクセスが未完了である状態を考慮する必要があるため，キャッシュシステムの状態の数は爆発的に増加する．設計検証の手段の1つとして，実装が仕様を満たすことを形式的に証明する手法があるが☆，状態数の多さ故に，システムの論理設計検証に適用することは，少なくとも現状では不可能であると言ってよい．

出力の非決定性

またバッファリングを行なうと，メモリアクセスの開始から完了までが不可分に行なわれないため，**クリティカルなアクセス**がシステム内に同時に存在することがある．クリティカルなアクセスとは，同一アドレス，あるいは，同一キャッシュブロックなどへのアクセスをいう．

JUMP-1 キャッシュシステムは，クリティカルなアクセスを発見した場合には，あたかもそれらのアクセスが逐次的に開始されたかのように処理する[1)]．

☆ 実際，JUMP-1 のコヒーレンス・プロトコルには形式的手法を適用したが，プロトコルのような高い抽象度でさえ，バッファリング機構まで考慮した検証は行なえなかった[1)]．

この逐次化制御を行なうための機構は，JUMP-1 キャッシュシステムにおいて設計上重要な点であると同時に，その複雑さ故にバグが好発する部位でもある．従って設計検証では，この部分を重点的に検査すべきである．

しかし，シミュレーションによる設計検証では，クリティカルなアクセスの取り扱いが問題となる．JUMP-1 キャッシュシステムの仕様は，メモリの状態が非決定的となることを許す．非決定的な状況では，システムの振る舞いは，システムの直前の状態によって変化するため，クリティカルな入力に対する出力の期待値を静的に求めることは難しい．

2.3　設計検証の方針

前章での議論を踏まえて，本章では，本稿で述べる設計検証手法の基本方針を述べる．

メモリモデル

本手法は，JUMP-1 キャッシュシステムの実装に対してシミュレーションを行ない，入力と出力の関係が JUMP-1 メモリモデルを満たすことを確認するというものである．

メモリモデルは，プログラムから見えるメモリの振る舞いを規定した仕様で，メモリアクセス命令間の時相論理式として厳密に記述することができる．

キャッシュシステムへの入出力は，プログラムからのメモリアクセス命令によってのみ行なわれるので，メモリモデルはキャッシュシステムの抽象度の高い仕様を与える．従って，キャッシュシステムの実装がメモリモデルを満たせば，その実装は論理的に☆☆正しいといえる．

参照モデル

本手法では，**参照モデル**というモジュールを用いて設計検証を行なう．参照モデルは，ハードウェア記述言語（VHDL）によってメモリモデルを直接的に表現したものである．そして，キャッシュシステムの実装と，参照モデルの両方にテストベクトルを与え，それぞれの出力を比較することで設計検証を行なう．メモリが非決定的である状態では，参照モデルは，メモリモデルが許す全ての出力を求める．実装からの応答は，求められた出力の何れかであればよい．

参照モデルは，テストベクトルに加えて検証対象の出力を観測しながら動的に期待値を計算する．動的に計算することによって，メモリモデルの持つ非決定性を効率的に取り扱うことができる．非決定性の取り扱いについては4章で詳述する．

任意の入力系列に対して参照モデルは正しい出力を計算し，実装の正しさを判定することが出来るので，テストベクトルに対する制約は全くない．出来るだけ多くのクリティカルなアクセスを含むように生成してやればよい．

☆☆ この手法では性能的なバグを発見することはできない

3. 設計検証手法

では，具体的な設計検証手法について説明する．まず，本手法を理解する上で重要な JUMP-1 のメモリモデルについて概説した後に，設計検証システムの説明を行なう．

3.1 JUMP-1 メモリモデルの概要

JUMP-1 メモリモデルは，時相論理式を用いて厳密に定義できるものであるが，本節では，JUMP-1 のメモリモデルについて直感的に説明する．

メモリアクセス命令

プロセッサの命令パイプラインが JUMP-1 キャッシュシステムに対して発行する命令は，(1) ロード，(2) ストア，(3) ストアバリア の 3 種類である．

ストア命令の完了する順序

JUMP-1 のメモリモデルは，Weak Consistency と呼ばれるクラスに属しており，ストア命令の完了する順序は一般に非決定的である．ストア命令はストアバッファに対して実行され，その完了を待たずにひき続く命令の実行が開始される．プログラムからストア命令の完了を確認するためには，ストアバリア命令を用いる．ストアバリア命令を実行すると，未完了のストア命令が存在する間，引き続く命令の実行がブロックされる．従って，2 つのストア命令は，ストアバリア命令に隔てられた場合に限り，プログラムに記述された順序で完了することが保証される．

非 決 定 性

メモリモデルはメモリの状態が非決定的となることを許す．メモリモデルは，実行結果が，メモリモデルで定義された観念的なマシンモデルで生じ得る結果の何れかとなることを要求する．

従って，キャッシュシステムの実装がメモリモデルを満たしているかを判断するためには，参照モデルは，これら非決定的な正しい出力結果の全てを用意する必要がある．

ストアバリア命令と$\overline{\text{PEND}}$

JUMP-1 の実装では，ストアバリア命令は，$\overline{\text{PEND}}$ 信号によって実現される．$\overline{\text{PEND}}$ 信号とは，キャッシュシステムが生成する信号で，物理的な信号線によって命令パイプラインに伝えられる．ストアバリア命令が発行されると命令パイプラインは，$\overline{\text{PEND}}$ 信号がディアサートされるまで引き続く命令の実行をブロックする☆．従って，キャッシュシステムは，システム内に当該プロセッサの発行したストアのうち未完了のものが存在する間，$\overline{\text{PEND}}$ 信号をアサートすることを要求される．

本手法では参照モデルがストア命令の完了を知る手段としてこの信号を用いる．

☆ 実際には，それに先立って，ストアバッファのドレインを行なう．

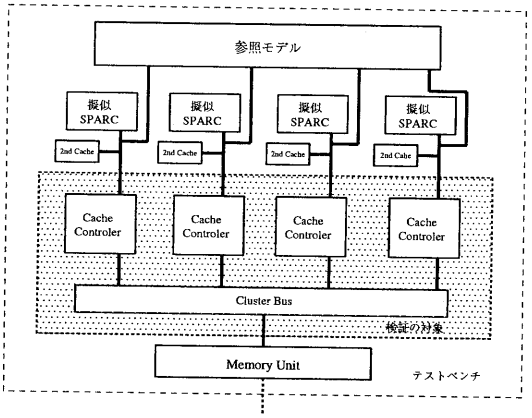

図 2　設計検証系

3.2 設計検証手法

本節では，設計検証手法について具体的に説明する．なお，本検証の要である参照モデルについては，4章にてより詳しく説明する．

3.2.1 設計検証システム

本手法では，設計検証の対象に，ハードウェア記述言語により記述されたテストベンチを接続し，それら全体に対してシミュレーションを行なう．

図 2に設計検証システムの全体を示す．図中，PE，および，2 次キャッシュは，それぞれを動作 (behavior) レベルでモデリングしたものである．PE は内部に 1 次キャッシュとストアバッファを備える．

さて，メモリモデルは，プログラムからの観測されるメモリの振る舞いを記述すると述べた．従って，メモリモデルにより規定されるのは，PE の命令パイプラインと 1 次キャッシュ/ストアバッファの境界である．従って参照モデルは，(検証対象からではなく，) PE 内部の，命令パイプラインと 1 次キャッシュ/ストアバッファの間の入出力，すなわち命令パイプラインからのメモリアクセス命令と 1 次キャッシュ/ストアバッファからのリードリプライを観測することになる．特に，アクセスが 1 次キャッシュにヒットした場合には，出力は，1 次キャッシュから返ってくることに注意されたい．検証対象が 1 次キャッシュに対する一貫性制御を正しく行なっていれば，1 次キャッシュから正しい値が応答される．

3.3 テストベクトル

本検証ではテストベクトルをランダムに生成する．

なぜこのようなことが可能なのか説明する．本設計検証においては，入力系列とそれに対して応答された出力系列の関係のみから検証を行なう．従って，検証対象への入力であるメモリアクセスの，(1) 種類 (ロード、または、ストア)，(2) アドレス、および，(3) ストア・データに関しては，ランダムに生成して構わない．但し，検証を行ないたいクリティカルな系列の生起確

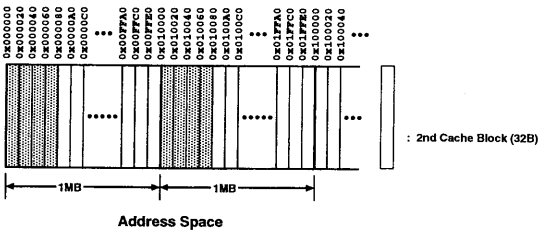

図3　アクセスするアドレス空間

率が十分に高くなるように，アクセスを行なうアドレス空間を狭くしてやる必要がある．また，2次キャッシュの置換え操作，書き戻し操作が発生する必要があるので，**図3** において網のかかった8ブロックに対してアクセスを行なうこととする．

なお，ストア・データに関しては，設計検証の効率化のため決まったフォーマットに従ったものを用いる．このことの詳細については，4.1 節にて述べる．

4. 参照モデル

参照モデルは，メモリモデルを満たす期待値を動的に計算するモジュールである．参照モデルは，各 PE が発行するストアを受け取ってメモリモデルを満たすすべての応答を動的に計算する．各 PE がロードを発行した場合には，検証対象からの応答を受け取り，その値がメモリモデルを満たすかどうかを判定する．3.1 節で述べたようにメモリモデルはメモリの状態が非決定的となることを許す．この非決定性の取り扱いが参照モデルの設計の鍵となる．

本章では，まず参照モデルの概略を述べた後，具体的な説明を行なう．

4.1 参照モデルの概略

本節では，参照モデル概略について述べる．まずはじめに世代という概念を定義し，キャッシュシステムに与えるストア・データについて説明した後，参照モデルの基本的な構成と動作について説明する．

世　　代

3.1節で述べたように，命令パイプラインからストアの完了を確認するには，ストアバリア命令によって間接的に$\overline{\text{PEND}}$ 信号を観測するが，本手法では，それに加えて$\overline{\text{PEND}}$ 信号を直接観測する．$\overline{\text{PEND}}$ 信号が一度ディアサートされてから再びディアサートされるまでの期間を，1つの世代と定義する．この定義によると，世代の終了時には，$\overline{\text{PEND}}$ はディアサートされるので，当該 PE の発行したすべてのストアは完了していなければならない．

ストアデータ

3.3節にて，テストベクトルの種類とアドレスに関してはランダムに生成すると述べた．ここでは，ストアの書き込むデータの値について説明する．

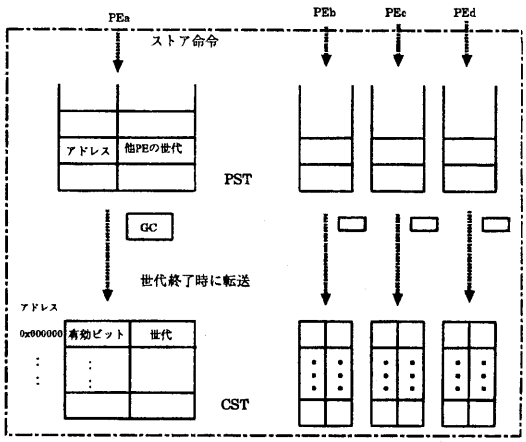

図4　参照モデルの構成

本手法では，同一世代に発行されたストアは同一の値を書き込むこととする．JUMP-1 メモリモデルでは，プログラムに現れる順序でストア命令が完了することが保証されるのは，ストアバリア命令に隔てられたときのみである．従って，同じ世代において同一アドレスに対して複数のストア命令が発行された場合には，どの結果が最終的にメモリに反映されても構わない．検証においては，それらのストア命令が区別できる必要はないので，それらの値は同一であって構わない．

理由は後述するが，本手法では，ストアデータの値として PE の ID とその命令が発行された世代の番号を書き込むこととする．

参照モデルの構成/動作

それでは，参照モデルの構成/動作の概要について説明しよう．

参照モデルは，その結果がロード命令によって読まれてもよいストア命令の履歴を全て記録することによって，出力の期待値を求める．参照モデルは，PE 毎に未完了ストアテーブル（以下，**PST**）と既完了ストアテーブル（以下，**CST**）の2種類のデータ構造を用意し，ストア命令の履歴を記録する（図4）：

PST　当該世代に発行された，完了していない可能性のあるストア命令をすべて記録する．

CST　それより前の世代で発行された，既に完了したストア命令を，各アドレスに対して，高々1つ，記録する．

JUMP-1 のメモリモデルでは，非決定的状況を許す：すなわち，未完了のストアが存在する場合には，既に完了しているストアか，未完了のストアのうちいずれかの結果がロードされてよい．したがって，各 PE の PST/CST のいずれかにあるストアの値がメモリモデルを満たす．

プロセッサからストアがあった場合には，まず，そのストアの情報を PST に記録し，PST の履歴は世代の終了時に CST に移す．ある世代が終了する時に

は，当該プロセッサの行なったストアのうちに未完了のものは存在しないので，ロードされてよいのはその世代で行なったストアのデータのみである．前項で述べたとおり，同一世代では同一の値をストアするので，CST には，各アドレスに対して，その値を記録すればよい．以前に CST にあった情報は上書きされる．PST の履歴を CST に移す操作は，不要となった古い履歴を破棄することに相当する．

ところで，世代を定義するのに用いた$\overline{\text{PEND}}$ 信号は，設計検証対象である JUMP-1 キャッシュシステムが生成するものである．このように対象の生成する信号を用いてよい理由を述べる：$\overline{\text{PEND}}$ 信号により記録を破棄されたストア命令の結果がキャッシュシステムより応答された場合には参照モデルは対象に誤りがあると判断するが，それは，その応答自体が誤りであるか，$\overline{\text{PEND}}$ の誤りによって記録を破棄されたか，あるいは，その両方かであり，どちらの場合も対象に誤りがあることを正しく発見することができる．

4.2 参照モデルの実装

参照モデルは，期待値の数を減らすために，いくつかの手法を用いてストア命令の記録を破棄する．本節では，これらの手法を中心に参照モデルの実装について説明する．

4.2.1 参照モデルの構成

図 4に参照モデルの構成を示す．参照モデルは，既に述べたように，ストア命令の記録を保持するために，PE 毎に PST と CST の 2 つのデータ構造を持つ．さらに，PE 毎に世代を数える世代カウンタ (GC) を持つ．

PST は，当該世代に発行されたストア命令の記録を保持するバッファで，バッファには，(1) ストアが行なわれるアドレスと (2) ストア命令が発行された時点での他の PE の世代が記録される．4.1節で述べたように，ストアするデータは，PE の ID と世代番号なので，改めてストアデータを記録する必要はない．

一方，CST は，その世代より前に発行されたストア命令の中で，結果がメモリに反映されている可能性のあるものの記録である．CST は，バイト毎に，(1) 有効ビットと (2) そのストア命令が発行された世代を記録する．PST と同様に，ストアデータが PE の ID と世代を表す番号であるため，ストアデータを記録する必要はない．

4.2.2 参照モデルの動作

参照モデルの動作は，(1) リクエスト受け付け時，(2) 世代終了時，(3) リプライ到着時の 3 つに分けられる．以下にそれぞれの場合について説明する．

(1) リクエスト受け付け時の参照モデルの動作

PE から受け取ったストア命令は，一旦 PST の対応するストアバッファに格納される．このときストア命令の対象となるアドレスと他 PE の世代を記録する．この記録は，後述するように，このストア命令が発行

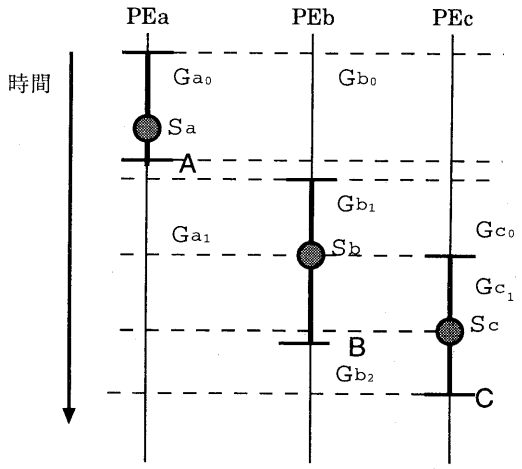

図 5　世代終了時のストア命令の記録の破棄
—：世代の変わる点
●：同一アドレスに対するストア命令
Gx_i：PEx の世代 i
世代 Gc_1 終了時に，Sa の記録を破棄する．

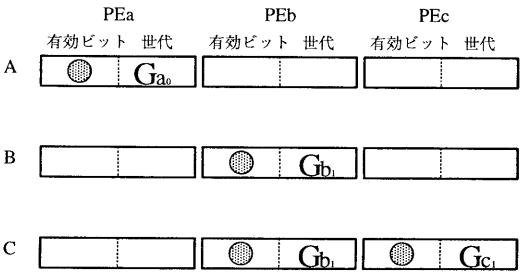

図 6　CST の更新
B 点では Sa の記録を破棄するが，C 点では Sb を破棄しない

された時点で他のストア命令が完了していたか評価するために用いる．

(2) 世代終了時の参照モデルの動作

世代終了時の動作の基本ポリシは以下に述べる通りである: あるストア命令が完了したあとで，同一アドレスに対し別のストア命令を発行すると，最終的に後者のストア命令がメモリに反映される．従って，後者のストア命令が完了した時点で，前者の記録を破棄してよい．

ここで，ストア命令の記録の破棄とは，対応する PST のエントリ，CST の有効ビットをそれぞれ消去，リセットすることをいう．

図 5を例に説明する．図中,—は世代の変わる点を，●は同一アドレスに対するストア命令を．そして Gx_i は PEx の世代 i をそれぞれ表す．

PEa が世代 Ga_0 において，ストア命令 Sa を発行する．このストア命令の記録は一旦 PST に格納され，世代 Ga_0 が終わる A 点で CST に転送される．CST の

有効ビットをセットし，このストア命令を発行した世代を記録する．(図6のA)．

次に，PEbが世代Gb_1においてストア命令Sbを発行する．このストア命令の記録は，一旦PSTに格納される．このとき，他PEの世代を記録しておく(この場合，PEaがGa_1，PEcがGc_1)．そしてB点で世代Gb_1が終わると，Sbの記録は，CSTに移される．CSTの有効ビットをセットし，このストア命令を発行した世代を記録する．

このとき，同時に他PEのCSTをチェックする．PSTで記録しておいた他PEの世代と，CSTに記録されている世代を比較すると，Sbが発行された時点で既にSaは完了していたことが分かる．よって，PEaのCSTの有効ビットをリセットしてSaの記録を破棄する(図6のB)．

最後に，PEcが世代Gc_1においてストア命令Scを発行する．同様に，このストア命令の記録は一旦，PSTに格納される．このとき他PEの世代を記録する(この場合，PEaが世代Ga_1，PEbが世代Gb_1)．そしてC点で世代Gc_1が終わると，Scの記録はCSTに移される．CSTの有効ビットをセットし，このストア命令を発行した世代を記録する．

このとき，Sbの場合と同様に他PEのCSTをチェックする．この場合，PSTで記録しておいた他PEの世代と，CSTに記録されている世代を比較すると，Scが発行された時点ではSaは完了していない可能性がある．従って，この場合は，Sbの記録に対しては何もせず，Sbの記録をCSTに書き込む(図6のC)．

(3) リプライ到着時の参照モデルの動作

参照モデルは，JUMP-1クラスタから出力を受け取ると，2つのことを実行する．すなわち，その出力がメモリモデルを満たすか判断することと，その出力を受けたことによりそれ以降メモリに反映していてはならないストア命令の記録を破棄することである．

メモリモデルを満たしている出力の期待値は，全てのPST，CSTに記録されているストア命令の何れかによる結果である．

リプライ到着時の参照モデルの動作を図7を例に説明する．PEa，PEb，PEcが図7のようなタイミングでそれぞれSa，Sb，Scのストア命令を発行する．但し，A点で世代Gc_1が終わり，Scの記録をPSTからCSTに移した際，CSTに記録されていた他のストア命令の記録を破棄できたとする．つまり，これ以降Sa，Sb，Scの何れかの結果しかロードできないとする．

PEbが，D点以降にこれらのストア命令と同一のアドレスに対してロード命令を発行した場合を考える．

この場合，出力の期待値は，PSTに記録されているSa，Sb，あるいはCSTに記録されているScの結果である，従って，それ以外の結果がキャッシュシステムから出力された場合には，キャッシュシステムのエラーとなる．

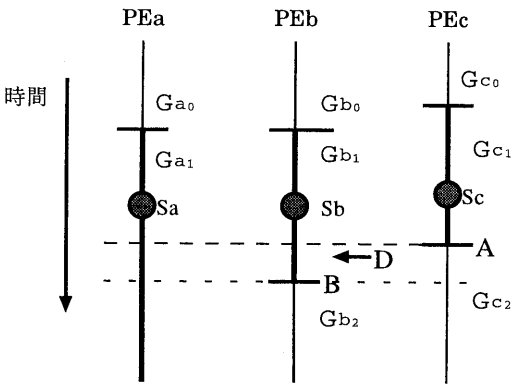

図7　リプライ到着時の参照モデルの動作
—：世代の変わる点
●：同一アドレスに対するストア命令
Gx_i：PExの世代i
Sa，Scの結果がロードされた場合，Sbの記録を破棄する

もし，Sbの結果がロードできた場合には，どのストア命令の記録も破棄しない．しかし，SaまたはScの結果が出力された場合には，PST内のSbのエントリを消去することでSbの記録を破棄する．

5. おわりに

本稿では，出力の期待値を動的に計算しながらシミュレーションを実行することで，キャッシュシステムの設計検証を行なう手法について述べた．

本稿で述べた手法を実際JUMP-1キャッシュシステムに適用した結果，いくつかのバグを見つけることが出来た．

本稿で述べた設計検証手法をキャッシュシステムに対して適用することが出来たのは，キャッシュシステムの外部仕様が，簡潔に定義されたメモリモデルによって厳密に規定されている，という事実によるところが大きい．このために，参照モデルのようなモジュールを高い抽象レベルで設計することができ，出力の非決定性に対処することが出来たといえる．

参考文献

1) 福島直人，浜口清治，富田眞治，矢島脩三: 形式的手法によるキャッシュ・プロトコルの設計検証，情報処理研究会報告 96-ARC-117. pp.1-6. (1996).