社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8（20）
2008 / 3 / 28

準形式的モデル検査のハードウェア実装による高速化の検討

森下 賢志† 吉田 浩章†† 藤田 昌宏††

† 東京大学工学部電子工学科 〒113-8656 東京都文京区本郷 7-3-1
†† 東京大学大規模集積システム設計教育研究センター 〒113-0032 東京都文京区弥生 2-11-16
E-mail: †morishita@cad.t.u-tokyo.ac.jp

あらまし 近年設計が大規模になり、複雑化が進むにつれて検証の重要性が増している。現在の重要な検証手法の一つであるモデル検査には適用する回路規模が大きくなると状態爆発を起こし、実用的な時間内で検査が終了しないという問題がある。そのため回路規模や検証時間の面でより効率的なモデル検査が必要とされている。現在の効率的なモデル検査手法の1つに、コンパイルドシミュレーションを基にした準形式的限定モデル検査[8]がある。本稿ではそのアルゴリズムの一部をハードウェアで実装し、各々の処理に対してハードウェア・ソフトウェア協調実行に向けた最適化を行うことで高速化したモデル検査を提案する。また例題を用いた実験によって実際に高速に検証を行い、提案手法の有効性を確認した。
キーワード モデル検査

A Hardware Acceleration for Semi-Formal Model Checking

Satoshi MORISHITA†, Hiroaki YOSHIDA††, and Masahiro FUJITA††

† Department of Electronics Engineering, Faculty of Engineering, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan
†† VLSI Design and Education Center, University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-0032 Japan
E-mail: †morishita@cad.t.u-tokyo.ac.jp

Abstract The verification becomes important now as the design becomes complex and large-scale. Model checking which is one of the most important verification has been held back by the state explosion problem, which is the problem that the number of states grows exponentially in the number of system components. So we propose an efficient model checking method in this paper. We have enhanced a semi-formal bounded model checking [8] by using a hardware accelerator and modified the codes to be easily implemented on a hardware. The experimental results with some examples show that the proposed method can execute model checking in short time.
Key words Model Checking

1. はじめに

1.1 研究の背景

近年、半導体の集積・製造技術の進歩は著しく、製造できるLSIのゲート数が3年で4倍という急速なスピードで増加している。しかし検証可能なゲート数の増加はそれに追いついておらず、その差は年々深刻化している。この差を埋めるためにより効率的な検証手法が必要となっている。現在LSI設計における設計検証は、主にシミュレーションによって行われているが、大規模な回路を仕様上可能な入力パタン全てに対してシミュレーションすることは現実的には難しい。そのためコーナーケースが存在し、バグなどが完全に取り除かれないまま検証を終えてしまう可能性がある。一方、形式的検証は回路の正しさを数学的に証明する検証手法であり、入力パタンによらず網羅的に検証を行うことができる。しかし多くの問題は最悪の場合、回路の大きさに対して指数的に計算量が増大するため、適用可能な回路の規模が限られる。そのため、形式的検証の検証時間や回路規模の面での効率化が重要である。

1.2 関連研究

1.2.1 モデル検査

形式的検証の一つに、モデル検査 (Model Checking) [1] と呼ばれるものがあり、これは一般に設計対象から導出された数学的モデルが、仕様を満たしているかどうかを調べることを指す。具体的な手法としては、まず回路の中の変数やレジスタ値の変化を状態遷移として扱い、設計を有限状態機械 (FSM:Finite State Machine) の形で表現し、満たすべき性質 (プロパティ) を時相論理式 (Temporal Logic) で表現する。次に設計の FSM がプロパティを満たすかどうかを判定する。全ての状態においてプロパティが満たされるならば、設計がそのプロパティに対して正しいことが証明される。しかしモデル検査は設計が大きくなると、設計が持つ状態変数の数に対して検証に必要な計算量が最悪の場合指数的に増大する。

現在のところプロパティの判定方法は、明示的 (Explicit) な手法と、暗黙的 (Implicit) な手法の二通りがある。前者は最も基本的な手法で、明示的に FSM の状態遷移を辿りながらプロパティの式を評価する手法である。一般的に利用されている代表的なツールとしては SPIN [2] がある。後者は FSM の状態や状態遷移を論理式で表現してプロパティの式との積を取り、その式を評価するシンボリックモデル検査 [3] と呼ばれる手法である。シンボリックモデル検査では辿るべき状態遷移一つの論理式で表すことができるため、明示的な手法と比べて FSM を小さい表現で表すことができる。式の評価は二分決定グラフ (BDD:Binary Decision Diagram) [4] に格納することや、充足可能性問題 (SAT 問題) に帰着させることで行われる。代表的なシンボリックモデル検査ツールとしては、BDD を使ったものには SMV [3] が、SAT を使ったものには NuSMV [5] などがある。

またモデル検査で扱うことができる設計規模を大きくする手法の一つとして、限定モデル検査 (BMC:Bounded Model Checking) [6] がある。これは順序回路を検証範囲をある状態から一定のサイクル数だけ展開し、1 つの組み合わせ回路として扱うことで検証を効率的にする手法である。先に述べた SPIN や NuSMV は、限定モデル検査を行うことができる。

1.2.2 準形式的限定モデル検査

現在の効率的なモデル検査手法の一つに、コンパイルドシミュレーションを基にした準形式的限定モデル検査 [8] がある。この手法は暗黙的な方法ではなく、明示的に全ての入力パタンに対してプロパティを満たすかどうかを、シミュレーションによって検査するという手法を取っている。単純に全入力パタンをシミュレーションすることは現実的に難しいが、この手法ではシミュレーションと同時に回路を解析することで、シミュレーションの必要が無い冗長な入力パタンを動的に割り出す。それを次回以降の実行から除外することでシミュレーション回数を大幅に減らす。

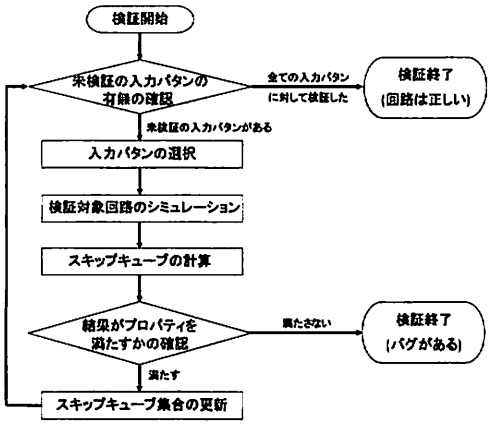

図 1 準形式的限定モデル検査のフロー

図 1 にこのモデル検査のフローを示す。最初にコンパイルドシミュレーション用のプログラムを作成する。これには検証対象回路、プロパティ、検証対象回路に対応する解析用回路が含まれる。この回路の出力によって、検証対象回路が入力に対してプロパティを満たすかどうかを判別することができる。次にシミュレーションに対する入力を選択する。既にシミュレーションを行ったもしくはシミュレーションをする必要が無いと判断された入力パタンの集合を論理関数として保持するため、この関数に含まれない入力パタンが選択される。次に選択された入力パタンに対してシミュレーションを行う。この手法では、シミュレーションを行うとその結果を計算すると同時に、それと全く同じ結果を導く入力パタンの部分集合を求めることができる。結果が同じになるとわかっている、すなわちシミュレーションを行う必要がない入力パタンの集合をスキップキューブと呼び、これらは次回以降の入力パタンの選択候補から除外される。

スキップキューブはキューブ表現によって表される。キューブ表現とは、変数を "0, 1, −" の 3 つの値を使って示す表現である。" − " は値が 0, 1 どちらでも良いということを表す。例えばキューブ $\{0,1\}$ と $\{0,0\}$ の和集合は $\{0,-\}$ となる。スキップキューブは論理ゲートごとに入力と出力の依存関係を元に計算される。例えば、5 入力 AND 回路に対して入力パタン $\{0,1,1,1,1\}$ でシミュレーションを行ったとする。この時出力は 0 となるが、これは 1 番目の入力が 0 であることに由来し、1 番目の入力が 0 である限りは他の入力が変化しようとも同じ結果になる。よってこの場合スキップキューブは $\{0,-,-,-,-\}$ となり、これに該当する入力パタンに対してはシミュレーションを行わない。判明したスキップキューブは、検証が終わるまで保存し、そのキューブを満たす入力パタンは次回以降シミュレーションを行わないようにしなければならない。このような除外された入力の集合をスキップキューブ集合と言う。この手法ではスキップキューブ集合の表現に BDD [4] を使用する。

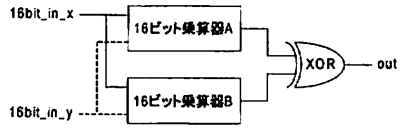

図 2　例題 MULT32-n の回路

シミュレーションを繰り返すことで、プロパティを満たすと証明された入力パタンがスキップキューブ集合に追加されていき、全ての入力パタンがスキップキューブ集合に含まれれば回路はプロパティに対して正しいと証明される。プロパティを満たさない入力パタンが見つかった場合は反例としてその入力パタンが返される。

1.3　本稿の目的

第 1. 小節で述べたように、現在の LSI 設計において形式的検証の効率化は重要である。そこで本研究では形式的検証の一手法である、モデル検査を高速化することを目的する。研究のベースとなるモデル検査には、第 1.2.2 小節で紹介した、コンパイルドシミュレーションを基にした準形式的な限定モデル検査の手法を使用する。このモデル検査の一部は本質的に並列性が高く、ハードウェアで実行することで高速化が期待される。

2.　準形式的モデル検査の高速化

2.1　プロファイリングに基づいた解析

設計を高速化するためには、まず設計中のどの処理にどれだけの時間がかかっているかを把握しなければならない。設計のボトルネックを高速化することができれば全体の実行時間を大きく減らすことができる。本研究ではプロファイルよってこの設計のボトルネックを特定し、その処理を最適化する。

プロファイルを行うために、まずコンパイルドシミュレーションを基にした準形式的なモデル検査を行うプログラムを作成した。また検証に使用する例題として、図 2 を作成した。これは 2 つの乗算器に 16 ビット × 2 のパタンを入力し、その出力の n ビット目が同じになっているかを検証する回路となっている。同じ出力ならば排他的論理和ゲートの出力は 0 となる。2 つの乗算器は同じものなので当然同じ出力を出し、排他的論理和ゲートの出力は常に 0 となるはずである。これを例題 MULT32-n とする。そしてモデル検査プログラムで MULT32-10、すなわち乗算器の出力の 10 ビット目の検証をし、そのプロファイルを行った。この時 CPU：Core 2 Duo T7200 (2.0GHz)、メモリ:2GB の計算機を使用し、プロファイラには IBM 社の Rational Quantify [7] を使用した。プロファイル結果を表 1 に示す。項目は図 1 にそって分類されている。

表 1　各処理が占める実行時間の割合 (MULT32-10 の場合)

	処理時間が占める割合
入力パタンの選択	14 %
検証対象回路のシミュレーション	24 %
スキップキューブの計算	17 %
スキップキューブ集合の更新	44 %

2.2　設計記述の最適化による高速化

準形式的モデル検査をプロファイルした結果、ボトルネックはスキップキューブ集合の更新だと判明した。準形式的モデル検査では検査を終えた入力パタンの集合を、

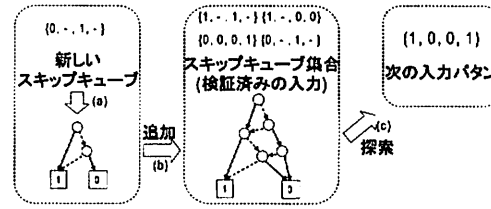

図 3　準形式的モデル検査において BDD を扱う処理の手順

BDD [4] で保持しており、ここでは BDD に関する演算がボトルネックとなっている。BDD を扱う部分は入力パタンの選択とスキップキューブ集合の更新であり、その手順は図 3 のようにスキップキューブ表現から BDD 表現への変換 (a)、BDD 同士の論理和演算 (b)、BDD に含まれない入力パタンの選択 (c) の 3 つで構成される。これらの演算で使用した一般的な BDD を扱う関数を、このモデル検査で必要な処理のみに特化した関数に置き換えることで、それぞれの手続きの回数を減らす。

本研究で提案する (a) の手順を図 4 を例に説明する。まず最初に、キューブ表現に含まれない変数は BDD では省略されるため、無視する。次に BDD において最も終端に近いノードとなる変数を選択する。この例では変数の順序は (a → c → d) であるため、d が選択される。この最も終端に近い変数については、1 終端と 0 終端を子ノードに持つノードを作成する。次に二番目に終端に近い変数を選択し、先ほど作成したノードと 0 終端を子ノードに持つノードを作成する。これを繰り返し、キューブに含まれる全ての変数に対してノードが作られるまで繰り返す。この手法によってキューブ表現から BDD 表現への変換を効率的に行うことができる。しかし、一度作った BDD に後から ($b = 1$) などの条件を追加することは困難になっている。そのため、今回のように最初から作る BDD の要素が全てわかっている場合のみ使うことができる。また準形式的モデル検査では、(a) や (b) に ITE 関数と呼ばれる、BDD の様々な演算を行うことができる汎用関数が使われている。しかし (a) で ITE 関数を使う必要がなくなり、(b) では ITE 関数は論理和にしか使われないため、(b) で使われていた ITE 関数を BDD の論理和演算関数に置き換えることで手続きを減らすことができる。

2.3　一部ハードウェア化による高速化

第 2.2 小節で最適化を行ったプログラムに対して再びプロファイルを行うと、結果は表 2 のようになり、ボトルネックが検証回路のシミュレーションとスキップキュー

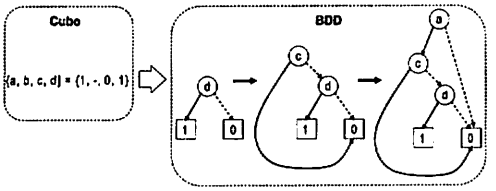

図 4　本研究でのキューブから BDD への変換

表 2　手続き最適化後の各処理が占める実行時間の割合 (MULT32-10 の場合)

	処理時間が占める割合
入力パタンの選択	2 %
検証対象回路のシミュレーション	49 %
スキップキューブの計算	34 %
スキップキューブ集合の更新	13 %

ブの計算に移行している。これらの部分は本質的に並列性が高く、ハードウェアで並列に行うことによって高速に計算できると考えられる。一方 BDD を使用する部分はハードウェアによって効率的に実行することは難しい。これは BDD による処理の大部分をメモリアクセスが占めるため、演算のアルゴリズム自体を変更しなければハードウェアで実行してもメモリアクセス回数は変わらず、高速化が難しいからである。よってここではシミュレーション部分とスキップキューブの計算部分をハードウェアで、BDD に関する部分をソフトウェアで実装し協調実行させることで高速化を図る。

まず検証対象回路については、元々組み合わせ回路もしくは限定モデル検査として順序回路を組み合わせ回路に展開したものであるため、そのままハードウェアにし実行することは容易である。次にスキップキューブの計算のハードウェア化について検討する。スキップキューブの変数は三値を取り得るため、変数 1 つにつき 1.5 ビットの情報量となる。しかし回路の入力端子におけるスキップキューブの値は入力パタンに依存し、スキップキューブは回路の入力端子から出力端子まで順に計算されていくため、一度のシミュレーション内では全てのスキップキューブが入力パタンに依存している。例えば 4 入力回路で入力パタン $\{1,1,0,0\}$ をシミュレーションする時に、$\{0,-,-,-\}$ や $\{1,-,-,1\}$ などの入力パタンと違う値を取るスキップキューブが生まれることは無い。よって本来 1 変数につき 1.5 ビットの情報であるスキップキューブは、変数が任意か決まっているかを区別する 1 ビットの情報と入力パタンに分割でき、計算中は前者の 1 ビットの情報のみで済む。この 1 ビットの情報のみに変換されたスキップキューブをスキップキューブのビット表現 (以下ビット表現) と呼ぶ。例えばスキップキューブ $\{1,-,0,-\}$ は入力パタン $(1,1,0,0)$ とビット表現 $\{1,0,1,0\}$ で表される。

スキップキューブの計算方法は 5 種類に分けられ、このうち入力端子、インバータ、レジスタの 3 つについては計算としては値の代入のみなので、ハードウェア化は容易である。次にコントロール値を持たない 2 入力ゲートの場合は 2 つのスキップキューブの積集合、すなわち各ビットごとの積を取ればよく、これもハードウェア化が容易である。しかしコントロール値を持つ 2 入力ゲートの場合は、元の回路でのゲートへの入力によって計算法が変わってしまい、特に入力の両方がコントロール値である場合、2 つのスキップキューブから、内包する入力パタンが多いものを選択する。このような、入力によって複雑に分岐する計算を組み合わせ回路で行うのは困難である。そこで本研究ではハードウェア化に向けて、入力の両方がコントロール値である場合は何も比較せずに一方のスキップキューブを選択するとする。この場合内包する入力パタンが少ない方を選択してしまい、1 度に除外できる入力パタンが少なくなる可能性がある。しかしこの点についてはほとんど影響が無いことを後の実験で示す。この変更によって、スキップキューブの計算は変数ごとに個別で行うことが可能になるため、入力 (a,b)、出力 out、コントロール値 u でスキップキューブの n ビット目を計算する場合、表 3 の真理値表を満たす 4 入力 1 出力の回路を、スキップキューブの変数の数だけ作ればよい。例えば a の値がコントロール値であり b の値がコントロール値でない場合、b のスキップキューブが out のスキップキューブに代入される。これを満たす 4 入力 1 出力の回路は図 5 である。

表 3　コントロール値を持つ 2 入力ゲートのスキップキューブの計算の真理値表

a の値	b の値	out のスキップキューブ
$\overline{u}$	u	a のスキップキューブ
u	$\overline{u}$	b のスキップキューブ
u	u	a と b のスキップキューブの論理和
$\overline{u}$	$\overline{u}$	a のスキップキューブ

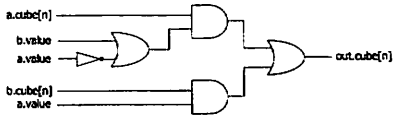

図 5　コントロール値を持つ 2 入力ゲートのスキップキューブの計算回路の 1 ビット分

スキップキューブの変数の数は入力数に比例し、変数 1 つごとに計算回路が必要となる。計算回路が最も大きくなるのは図 5 の回路が必要な場合であり、n 入力回路では n 個の変数を持つため、1 ゲートに付き最大 $4n$ ゲートの回路を必要とする。ここでは 1 ゲートとは二入力の論理演算ゲート 1 つを指す。よって検証対象回路が m ゲートだとすると、スキップキューブ計算回路は $O(nm)$ ゲートの規模になる。

2.4　入力パタンの投機的生成による HW/SW 通信の最適化

プログラムの一部をハードウェア化し、ハードウェア・

ソフトウェア協調実行をする際に問題になるのが、ハードウェア-ソフトウェア間の通信によるオーバーヘッドである。一部ハードウェア化したことでその部分の処理が高速化しても、通信時間が加算されることでプログラム全体は遅くなってしまう可能性もある。

第 2.3 小節のように準形式的モデル検査の一部をハードウェア化した場合、一度のシミュレーションごとに、入力パタンとスキップキューブをハードウェア-ソフトウェア間でやりとりする。よって検証終了までにシミュレーション回数の 2 倍の通信が行われる。また準形式的モデル検査ではある入力パタンに対してシミュレーションを行った場合、その結果が出るまで除外される入力パタンがわからないため、次の入力を選択することはできない。そのため準形式的モデル検査をハードウェア・ソフトウェア協調実行した場合ソフトウェアからハードウェアに入力パタンが送られると、ハードウェアからスキップキューブが送られてくるまでソフトウェアは動作しない。このような通信モデルでは通信のレイテンシがそのままオーバーヘッドとして実行時間に上乗せされ、ボトルネックになる可能性が高い。検証対象の入力数を n、検証終了までのシミュレーション回数を m とすると、準形式的モデル検査のハードウェア・ソフトウェア協調実行時の通信の様子は図 6 のようになる。

通信遅延がボトルネックとなっている場合、通信回数を減らすことでこれを解消できる。その方法の 1 つに複数回分のデータをまとめて通信するという方法がある。一度に送るデータが小さくても通信には一定のオーバーヘッドがかかってしまうため、同じ量のデータを送るならば小さなデータを多数送るより、大きなデータを少ない回数で送るほうが通信のオーバーヘッドは小さい。準形式的モデル検査では複数の入力パタンを一度に生成し、それらをまとめてハードウェアに送信し、全てに対してシミュレーションを行った後スキップキューブをまとめてソフトウェアに送るという手順を取ることで、通信のオーバーヘッドを減らすことができる。しかしこのような手順の場合スキップキューブが逐次的に更新されないため、除外すべき入力パタンが判明しても複数回のシミュレーションが終わるまでそれは反映されず、無駄なシミュレーションを行う可能性がある。このシミュレーション回数の増加によるデメリットよりも、複数個の入力パタンをまとめて送信することによる通信回数の削減のメリットの方が大きい時、投機的生成は有効となる。図 6 のような通信モデルを変更し、一度に S 個の入力パタンを送信するようにした場合、図 7 のようになる。投機的生成によって通信回数に $1/S$ がかかるが m が増加してしまう。増加したシミュレーション回数を m' と置く。m'/S が m より小さければ投機的生成は有効である。

投機的実行の有効性を検討するために、一度に複数の入力パタンを生成した場合 m' がどの程度になるかを S を変化させてシミュレーションすることで測定した。図

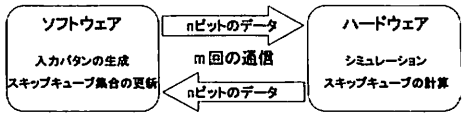

図 6 HW/SW 協調実行における通信のボトルネック

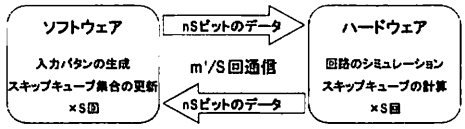

図 7 投機的実行による通信回数の削減

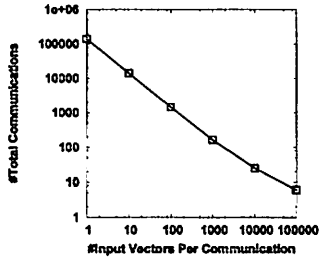

図 8 投機的実行時の通信回数 (MULT32-9 の場合)

8 に MULT32-9 を投機的実行した実験結果を示す。横軸に一度に生成する入力パタンの数 (S)、縦軸に検証が終了するまでの通信回数 (m'/S) を取ったグラフである。S が増えるにつれ検証終了までのシミュレーション回数は大幅に減少している。よって準形式的モデル検査はハードウェア-ソフトウェア間の通信のオーバーヘッドが大きくても、入力パタンを複数個シミュレーションを行う投機的実行によって通信のオーバーヘッドを減らし高速化できると考えられる。

3. FPGA と PC を用いたハードウェア・ソフトウェア協調実行

第 2. 節で述べた高速化を実現するために、計算機と FPGA による実験を行った。実験ではシミュレーション部分とスキップキューブの計算部分を FPGA に実装し、計算機上のソフトウェアと実際に協調実行した。計算機はプロファイル時と同じ計算機を、FPGA は Xilinx 社の Virtex-II Pro を使用した。計算機と FPGA 間の通信には UDP/IP を使い、イーサネット経由でこれを行った。これは UDP/IP がイーサネット上における比較的オーバーヘッドの少ない通信プロトコルだからである。また UDP/IP で通信を行う際のハードウェア側の制御に PowerPC を使用した。これは Virtex-II Pro に搭載されているプロセッサである。

まず準形式的モデル検査をソフトウェアのみで実装し、例題 MULT32-n を検証した。MULT32-n を通常のシミュレーションで検証するには 2^n 回行う必要があるが、準形式的モデル検査ではスキップキューブによってその 60% 程度のシミュレーション回数で検証を行うことができた。

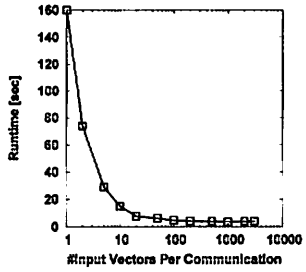

図 9　検証時間の面での投機的実行の有効性 (MULT32-9 の場合)

次に MULT32-9 の場合のシミュレーション及びスキップキューブの計算を FPGA に実装しソフトウェアと協調実行したところ、検証時間は 160 秒程度となった。しかし MULT32-9 をソフトウェアのみで検証した場合 26 秒程度で検証でき、ハードウェア・ソフトウェア協調実行によって検証速度は大幅に遅くなった。この時スキップキューブの計算を変更したことによるシミュレーションの増加は少なかった。よってハードウェア-ソフトウェア間の UDP/IP 通信のオーバーヘッドがボトルネックになり、遅くなってしまったと考えられる。

第 2.4 小節で述べたように、通信によるオーバーヘッドは投機的実行によって減らすことができる。今回の MULT32-9 の実験の通信は、1 回のシミュレーションにつき 32 ビットのデータを送受信していた。よってここでは、一度に S 個の入力を生成し、$32 \times S$ ビットのデータを送受信する形で、投機的実行を行った。MULT32-9 の検証を投機的実行し、図 9 に横軸に S、縦軸に検証全体の実行時間を取ったグラフを示す。S を増やすことで、投機的実行をしない $S = 1$ の時に比べて検証時間が短くなっており、投機的実行によって通信のオーバーヘッドを小さくすることができたことが確認できた。その結果検証時間は 3.7 秒になり、ソフトウェアのみの時より 7 倍高速に検証を行えた。

次に複数の MULT32 に対してハードウェア・ソフトウェア協調実行モデルの投機的実行を行い、複数の例題に対する有効性を確かめた。一部ハードウェア化する前のオリジナルモデルの実行時間と、投機的ハードウェア・ソフトウェア協調実行モデルの実行時間、そして 2 つを比較した時の高速化の倍率を表 4 に示す。どの例題に対してもハードウェア・ソフトウェア協調実行による高速化は有効であり、平均で 6.7 倍の高速化が見られた。これにより本研究の提案手法による単形式的検証の高速化の効果が実験によって実証された。

表 4　MULT32 の検証の実行時間 [秒] と高速化の倍率 (オリジナル/協調実行)

例題	オリジナル	協調実行	高速化の倍率
MULT32-8	5.7	1.0	5.7
MULT32-9	25.9	3.7	7.0
MULT32-10	122.5	16.5	7.4
MULT32-11	575.3	88.0	6.5
			平均： 6.7

4. 結　　論

本研究では形式的検証を効率的に実行することを目的とし、コンパイルドシミュレーションを基にした単形式的限定モデル検査の手続きを最適化し、一部をハードウェアで協調実行することによって高速化する手法を提案した。また実験によってその有効性を実証した。

単形式的限定モデル検査の高速化のために、まず記述の最適化を行った。一般的な BDD を扱う関数を単形式的モデル検査に特化した形にし、手続きを減らすことで高速化を図った。その後ハードウェア・ソフトウェア分割を行い、本質的に並列性が高い処理である回路のシミュレーションやスキップキューブの計算処理を、ハードウェアで行うことで高速化を図った。また上記の手法によってソフトウェアとハードウェアそれぞれの処理は高速化できたが、実際にハードウェア・ソフトウェア協調実行を行う際にはハードウェア-ソフトウェア間の通信のオーバーヘッドがボトルネックとなりやすい。そこで本研究では投機的実行による通信の最適化を行い、通信のオーバーヘッドを小さくする手法を提案した。

そして提案手法を取り入れた単形式的限定モデル検査を実装し、計算機と FPGA を用いてハードウェア・ソフトウェア協調実行を実際に行った。実験では高速化を行う前の単形式的限定モデル検査に対し 6.7 倍程度高速化することができ、提案手法の有効性を実証した。

文　　献

[1] E. M. Clarke, O. Grumberg, D. A. Peled, Model checking. MIT Press, 2000.

[2] G. J. Holzmann, "The Model Checker SPIN," IEEE Trans. on Software Engineering, vol.23, no.5, pp.279-295, May 1997.

[3] K. L. McMillan, Symbolic model checking: an approach to the state explosion problem. Kluwer Academic Publishers, 1993.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677-691, Aug. 1986.

[5] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," In Proc. of the International Conference on Computer Aided Verification, pp.359-364, July 2002.

[6] A. Biere, A. Cimatti, E. M. Clarke and Y. Zhu, "Symbolic Model Checking without BDDs," in Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes In Computer Science 1579, Springer-Verlag, pp.193-207, 1999.

[7] Rational Software Corporation, Quantify ユーザーズガイド, 1999.

[8] J. D. Bingham and A. J. Hu, "Semi-Formal Bounded Model Checking," In Proc. of the 14th International Conference on Computer Aided Verification 2002, Lecture Notes In Computer Science 2404, Springer-Verlag, pp.236-249, July 2002.