形式検証とランダムシミュレーションを併用した大規模ハードウェア設計検証
Large-scale Hardware Verification by Using Formal Method and Random Simulation Together

窪田 純士
Junshi Kubota
大阪大学大学院情報科学研究科
Email: j-kubota@ist.osaka-u.ac.jp

垣内 洋介
Yosuke Kakiuchi
広島工業大学情報学部情報工学科
Email: y.kakiuchi.du@it-hiroshima.ac.jp

浜口 清治
Kiyoharu Hamaguchi
島根大学総合理工学部
数理・情報システム学科
Email: hama@cis.shimane-u.ac.jp

***Abstract*—Formal verification is one of automated techniques to guarantee the correctness of hardware designs. The computational cost of formal verification, however, grows larger for a large-scale design, so that we cannot get the solution. In this paper, we propose a method to utilize intermediate data, which formal verification has generated, for a random simulation. Our method aims to avoid spoiling the computational cost for formal verification even if it does not terminate.**

I. はじめに

集積回路設計は大規模化が進むとともに短納期が求められるようになり，動作の機能的な正しさを保証する検証技術の重要性が高まっている．集積回路の一般的な設計検証はシミュレーションで行われ，検証者は計算機上で回路の入力信号に値を印加し，回路から得られる出力を確認する．しかし，この方法では人手が介することによる見落としや仕様の誤解といった問題が発生する上に，大規模な設計においては数百もの入力信号線の全ての値の組合せを１つ１つ人間が調べることはまず不可能である．

そこで，形式検証がシミュレーションを補完するために用いられることがある．形式検証とは，計算機によって回路の正しさを数学的に証明する技術で，なかでも，仕様を表現した式と設計記述を与えると，検証用ソフトウェアが「設計が仕様を満たすかどうか」を検査するモデル検査は広く使われている．

本研究で対象とするのは，同期式ディジタル回路，つまり順序回路である．順序回路は記憶領域であるレジスタとレジスタの次状態を計算するための関数にあたる組合せ回路からなり，レジスタの値の更新はクロックパルスに同期して行われる．このクロックタイミングを数えるための単位をサイクルと呼ぶことにする．順序回路を形式的に表現すると $M = (S, I, O, f, g, s^0)$ で表すことが可能で，ここで S は状態集合，I は順序回路への入力値の集合，O は回路の出力値の集合，f は次状態関数で $S \times I \to S$，g は出力関数で $S \times I \to O$，s^0 は初期状態を表す．

１ビットのレジスタが n 個存在する場合，その値の組合せが１つの状態となる．つまり，１つの状態 $s \in S$ はレジスタを表すブール変数 r_i のベクトルとして以下のように表すことができる．

$$s = (r_0, r_1, \ldots, r_i, \ldots, r_n) \quad r_i \in \{0, 1\}$$

実際には $\{0,1\}^n$ の要素のうち初期状態 s^0 に f を任意回適用して到達可能なものだけを状態集合に含めるため $S \subseteq \{0,1\}^n$ である．

基本的なシミュレーションによる検証では，t サイクル目における入力値である $i^t \in I$ $(t = 0, 1, \ldots)$ を人間が考えて計算機に与えると，次状態と出力 $s^{t+1} = f(s^t, i^t)$, $o^{t+1} = g(s^t, i^t)$ が計算され，その値を人間が確認することになる．つまり，検証を行う場合，$p = \langle i^0, i^1, \ldots, i^k \rangle$ を考え，順次試さねばならない．この p をシミュレーションにおける入力パターンと呼ぶことにする．回路の入力パターンの数は，入力信号線の数と検証で考慮するサイクル数 k に相乗して増加する．そのため，網羅的なシミュレーションを全て人手で行おうとすると，膨大な数のパターンを調べる必要があり，見落としも生じるため，一通り人手でシミュレーションを行った後に，入力パターンをランダムに生成するランダムシミュレーションを併せて用いることが多い．しかし，何も制約を与えないランダムシミュレーションは，回路が想定していない無意味な入力値を大量に生成してしまう．

一方で，形式検証では，計算機を用いて自動的に状態や出力値を検査する．具体的には，回路とそれが満たすべき仕様を論理式としてモデル化し，式が成立するかどうかを判定する．式が成立するかどうかという問題の解法としては，充足可能性判定 (SAT) 問題に帰着して各変数への値の割り当てを探索する方法が一般的になってきており，ヒューリスティックによって計算効率を高めた SAT ソルバが多数開発されている．た

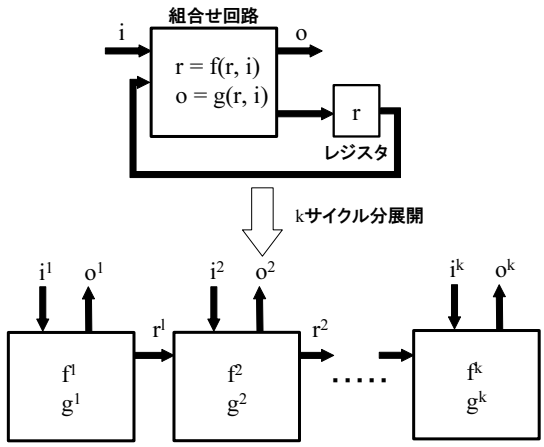

図 1. 有界モデル検査

だし，本質的な計算量は指数的であり，検証対象である設計の規模が増大すると，計算が終わらず実時間で結果が得られないことがある．

SAT を用いた形式検証で「$r_2 = 1$ になった 3 サイクル後に $r_1 = 0$ でなければならない」というような複数サイクルにまたがる仕様を検証したい場合は，順序回路 M の f と g を検査するだけでは単一サイクル内での動作しか検証できず不十分で，M を有限サイクル分展開した組合せ回路を作る必要がある．この方法は有界モデル検査 (bounded model checking) [1] と呼ばれる．有界モデル検査では，図 1 に示すように次状態関数と出力関数の k サイクル分のインスタンスが全て生成される．

本研究では，計算量が大きく，有界モデル検査が完了しなかった場合を想定し，モデル検査の計算過程で得られた中間情報を用いてランダムシミュレーションの制約式を自動的に作り出すことを考える．これによって，ランダムシミュレーションでの無意味な入力パターンの生成を抑え，ランダムシミュレーションの達成度を早く上げることができる可能性がある．

II. DPLL アルゴリズムと習得節

本研究の有界モデル検査における SAT 問題は，いわゆる CNF-SAT である．すなわち，変数 $r = (r_0, r_1, ..., r_n)$ を含む和積標準形 (CNF: Conjunctive Normal Form) の命題論理式 ϕ が与えられたとき，ϕ を真にするような割り当てが存在するかどうかを判定する問題である．r の各変数への真偽値割り当てを $\mathbf{r} = (\mathbf{r_0}, \mathbf{r_1}, ...\mathbf{r_n})$ で表し，割り当て $\mathbf{r}$ の下での式 ϕ の真偽を述語 $\phi|_{\mathbf{r}}$ で表すとすると，SAT 問題は $\exists \mathbf{r}.\phi|_{\mathbf{r}}$ を計算することになる．

形式検証でよく用いられる充足可能性判定アルゴリズムに DPLL(Davis-Putnam-Logemann-Loveland) アルゴリズムがある．ここでは，多くの DPLL SAT ソルバで標準的に採用されている矛盾解析を含んだ DPLL アルゴリズム [2] を図 2 に示す．アルゴリズムの詳細説明は割愛するが，未割り当ての変数を 1 つ選び，それに真偽値を割り当てて，矛盾が起こるかどうかを確認

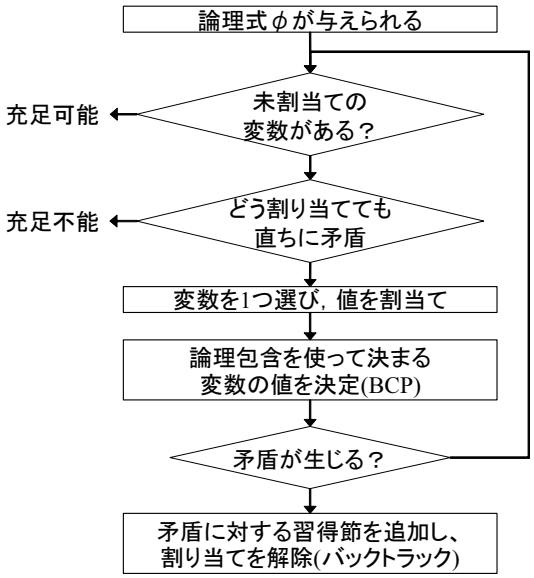

図 2. DPLL アルゴリズム

し，起こらなければ次の変数に割り当て，という具合に未割り当ての変数が無くなるまで繰り返す．もし，矛盾が起こった場合は，矛盾の直接の原因となる割り当てを解析して，その否定を節として元の式 ϕ に追加する．この節は習得節 (learnt clause) と呼ばれる．習得節を加えることで同じ矛盾を再度引き起こさないような探索の枝刈りが実現されている．

III. 習得節を用いたランダムシミュレーション

有界モデル検査が終了しなかった場合，あきらめてシミュレーションによる検証に切り替える必要があるが，内部の学習情報である習得節は全て無駄になってしまう．そこで，本研究では習得節による枝刈り情報を制約式としてランダムシミュレーションで流用することを考える．制約付きランダムシミュレーションの概念図を図 3 に示す．図 3 左は制約の無いランダムシミュレーションである．ここでの状態空間とは，レジスタ値と出力値の組 $(r_1, r_2, \ldots, r_n, o_1, o_2, \ldots, o_l)$ のうち取りうる全てとする．ランダムシミュレーションでは長さ k サイクルのランダムな入力パターン $\langle i^1, i^2, \ldots, i^k \rangle$ を何度も与えて状態空間を探索し，訪れた状態を「カバーした」として記録する．図中では 1 本の矢印が 1 回の入力パターンでのシミュレーションに対応する．そして，十分に未カバーな状態が減ると検証を完了する．そのため，未カバーな状態に到達するような入力パターンをできるだけ少ない試行で生成する必要がある．

一方，図 3 右は制約付きランダムシミュレーションである．有界モデル検査で得られた習得節を制約として用いることでランダムシミュレーションの生成を制限することができる．この制約で刈り取られた空間（図中黒塗り部）はすでに有界モデル検査で「探索の必要がない」と判断された領域で，あらかじめ入力パターンのランダム生成時に除いておくことで状態空間を狭め，未カバー領域に到達しやすくすることが可能だと

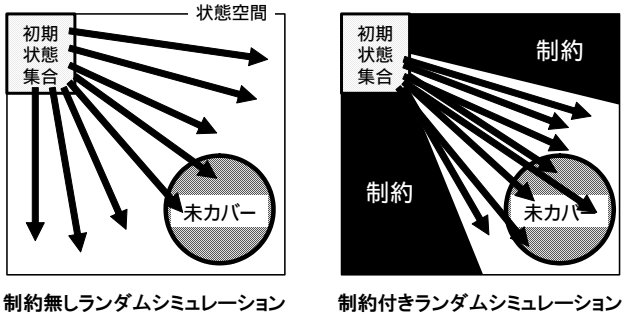

図 3. 制約付きランダムシミュレーション

考えられる．

有界モデル検査を中断すると，それまでの計算による習得節を CNF 式の節集合として得ることができる．ここではそれらの論理積を取った式を ψ とする．有界モデル検査では k サイクル目までのレジスタ，入力および出力の全変数に対する割り当てを探索しており，習得節はすでに探索が失敗することがわかっている割り当ての否定だとすでに述べた．よって，後の探索で同じ割り当てを行おうとするとこの節が偽になり，割り当てを抑止する．

例えば，矛盾の原因を解析した結果 $i_3^1 \wedge \neg i_2^3$ であったとしよう．これは「『1 サイクル目の入力 3 の値が 1, 3 サイクル目の入力 2 の値が 0』と割り当てた際に矛盾が起きたことを意味する．そこで，この否定を習得節 $\psi = \neg i_3^1 \vee i_2^3$ として，元の式 ϕ に追加する．この習得節は「1 サイクル目の入力 3 の値が 1」かつ「3 サイクル目の入力 2 の値が 0」のとき偽になるため，同じ矛盾が引き起こされることがない．

習得節には入力変数だけでなく出力変数やレジスタ変数も含まれることがあるが，有界モデル検査では，順序回路は展開された組合せ回路になっているので，レジスタ変数は中間変数に過ぎない．つまり，習得節内にレジスタ変数が入っていた場合，入力のみの式に巻き戻すことが可能である．例えば，r_1 の次状態関数が $r_1 = r_1 \vee i_1 \vee \neg i_2$ であったとすると，r_1^3 は以下のようにレジスタの初期状態と入力のみに巻き戻すことができる．出力についても同様である．

$$\begin{aligned} r_1^3 &= r_1^2 \vee i_1^3 \vee \neg i_2^3 \\ &= (r_1^1 \vee i_1^2 \vee \neg i_2^2) \vee i_1^3 \vee \neg i_2^3 \\ &= ((r_1^0 \vee i_1^1 \vee \neg i_2^1) \vee i_1^2 \vee \neg i_2^2) \vee i_1^3 \vee \neg i_2^3 \end{aligned}$$

得られた習得節の式 ψ に対し，ランダムシミュレーションの入力パターン生成においては，ψ を満たすような入力パターンのみを許す，という制約として利用する．

IV. 実装

以上の提案手法をプログラムとして実装することを考える．実装された手法の処理手順を図 4 に示し，各ステップでの処理について説明する．

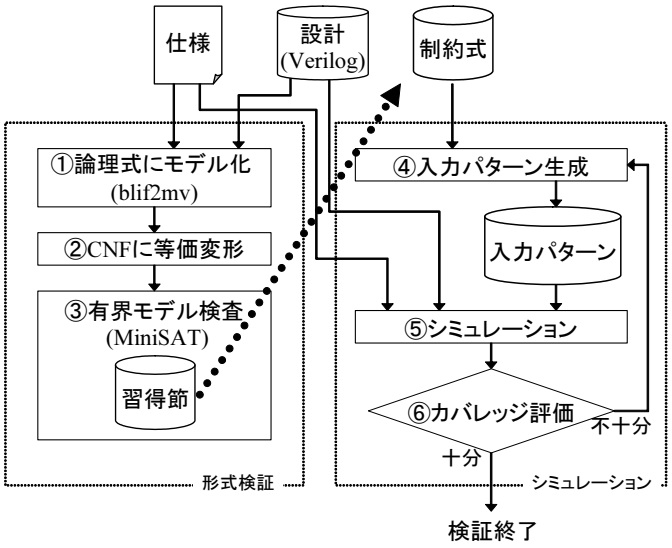

図 4. 提案手法の処理手順

1) 論理式にモデル化
ハードウェア記述言語である Verilog-HDL で書かれた検証対象の設計を論理式でモデル化し，仕様とともに 1 つの式 ϕ とする．Verilog から論理式を得るソフトウェアとしては VIS [9] を部分的に用いた．
2) CNF に等価変形
得られた式 ϕ は任意の形式の論理式であるので，SAT ソルバに入力できるよう CNF に等価変形を行う．方法は論理演算子 1 つに対して 1 つ中間変数を設ける Tseitin encoding [4] という手法を用いる．
3) 有界モデル検査
CNF 形式の ϕ を DPLL SAT ソルバである MiniSat [3] に入力しモデル検査を行う．MiniSat はオープンソースの SAT ソルバである．充足解を出さないまま一定時間が経過すると自動的に終了し，習得節を取り出すように内部に手を加える．得られた習得節は前章で述べたような巻き戻しなどの解析を行い，ランダムシミュレーションの入力パターンを生成する際の制約式 ψ とする．以上が形式検証を用いる部分である．
4) 入力パターンのランダム生成
制約式 ψ を満たすようなランダムに入力パターン $i = \langle i^1, i^2, \ldots, i^k \rangle$ を生成する．制約によって変数の取る値が一部制限されるため，制約無しランダムシミュレーションよりランダム化の値の範囲は狭くなる．文献 [5], [8] に示されているサンプリングアルゴリズムを参考にして，入力パターンのランダム生成を実装している．
5) シミュレーション
シミュレーションを行う CAD ツールに，設計と生成された入力パターン i を与えて，レジスタトランスファレベルの回路シミュレーションを行

う．シミュレーション結果は人手もしくはアサーションによって仕様と照合しバグが発見されれば検証を中断する．

6) カバレッジ評価
大規模な設計においては，シミュレーションで網羅的な検証を行おうとすると膨大な数のパターンが必要となり，状態空間全てを検証するのは実質不可能であるため，カバレッジと呼ばれる評価指標を用いてシミュレーションの完了を判断する．コードや条件，変数の値の変化など様々な評価基準を用いたカバレッジがあるが，ここでは状態カバレッジを用いる．順序回路の状態集合 S の要素と出力集合 O の要素の組のうち，シミュレーションで観測された組の集合を $V(S,O)$ とすると，状態カバレッジ Cov は以下のような式で定義される．

$$Cov[\%] = \frac{|V(S,O)|}{|\{(s,o) \mid s \in S,\ o \in O\}|} \times 100$$

すなわち Cov は状態と出力値の全ての組のうちの観測されたものの割合を表す．これが定められた閾値を超えたらシミュレーションを終了し，そうでなければ新たに入力パターン i をランダム生成してシミュレーションを繰り返す．このカバレッジの閾値は設計の種類や規模によって変わるため，開発の中で経験的に定められるのが一般的である．

V. 関連研究

有界モデル検査における習得節をランダムシミュレーションに利用する点が，本研究の新たな着目点である．制約によってランダムシミュレーションにおけるカバレッジを上昇させるという点では，いくつかの関連研究が存在する．タグカバレッジに着目し，カバレッジを改善するようなシミュレーションパターンの自動生成を行っている研究 [6] や，トグルカバレッジから情報量を計算し，パターンのランダム生成に制約をかける研究 [7] が報告されている．

VI. まとめ

本論文では，検証対象の設計規模が大きいために形式検証が完了しなかった場合，内部の学習データである習得節をランダムシミュレーションに用いる手法を提案した．これにより，ランダムシミュレーションを行う際の入力パターン生成が制限され，未カバー状態に到達しやすくなる．つまり，カバレッジを早く上昇させ，シミュレーションを短時間で終了させることに貢献する．現在，部分的に実装が完成しているため，今後は全体を実装し，ハードウェア設計に適用して実験を行っていく．

参考文献

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu: “Symbolic model checking without BDDs,” Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, pp.193-207, 1999.

[2] J. P. Marques Silva and K. A. Sakallah: “GRASP - a new search algorithm for satisfiability,” Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design, pp.220-227, 1996.

[3] Niklas Een, Niklas Sörensson: “An Extensible SAT-solver,” Theory and Applications of Satisfiability Testing, Vol.2919, pp.333-336, 2003.

[4] G. S. Tseitin: “On the complexity of derivation in propositional calculus,” Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp.466-483, Springer, 1983.

[5] A. Nadel: “Generating diverse solutions in SAT,” Proceedings of the 14th international conference on Theory and application of satisfiability testing, pp.287-301, 2011.

[6] S. Tasiran, F. Fallah, D.G. Chinnery, S.J. Weber and K. Keutzer: “A functional validation technique: biased-random simulation guided by observability-based coverage,” Proceedings of the International Conference on Computer Design: VLSI in Computers & Processors, pp.82-88, 2001.

[7] S. M. Plaza, I. L. Markov and V. Bertacco: “Toggle: A Coverage-guided Random Stimulus Generator,” Proceedings of International Workshop on Logic Synthesis, 2007.

[8] N. Kitchen and A. Kuehlmann: “Stimulus generation for constrained random simulation,” Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design, pp.258-265, 2007.

[9] “The VIS Homepage,” ⟨http://vlsi.colorado.edu/~vis/⟩

問い合わせ先

〒 731-5193
広島市佐伯区三宅 2-1-1
広島工業大学情報学部情報工学科
垣内 洋介