Vol.2014-SLDM-165 No.17
Vol.2014-EMB-32 No.17
2014/3/15

# アサーション自動生成とそのシミュレーションによる完全検証

藤田 昌宏[1,a)]　松本 剛史[1,b)]　城 怜史[2,c)]

**概要**：アサーションベース検証は、SoC に代表される大規模ハードウェア設計に対するもっとも効果的な検証手法の 1 つとして広く利用されている。検証品質は、チェックしたアサーションの質と量に依存するが、人手で十分なアサーションを迅速に用意することは現実には難しい。このため、仕様や設計記述からアサーションを自動生成する技術が研究開発されている。正しくない設計記述から生成されたアサーションは誤っている可能性があるが、自動生成されたアサーションを設計者が検査することにより、有益な情報として処理できる。本稿では、与えられた外部あるいは内部信号間の論理的な関係を一般的に抽出する新規手法の提案と、それを組合せ回路（あるいは時間軸方向に一定数展開された順序回路）に適用した実験結果について報告する。LUT（Look Up Table）で表現することで、数万ゲート規模の回路中の与えられた信号間に存在する論理的な関係を SAT ソルバーにより効率的に求めることができる。手法自体は、組み込みシステム一般にも適用できる。

**キーワード**：アサーションベース検証、形式的手法、アサーション生成、シミュレーションベース検証

# Automatic Generation of Assertions and their Formal Verification through Simulations

Masahiro Fujita[1,a)]　Takeshi Matsumoto[1,b)]　Satoshi Jo[2,c)]

***Abstract:***
Assertion Based Verification (ABV) is one of the most important verification methods for SoC and complicated hardware system designs. Quality of verification depends on the amount of assertions checked, but it may not be easy to prepare sufficient assertions by designers. There have been works which try to automatically generate assertions from specification and/or design description. Although possibly incorrect assertions may be generated from buggy designs, by the analysis on generated assertions with designers, even such information could be useful. In this paper, we propose a new method by which logical relationships among internal and external signals can be extracted and present its experimental results applied to combinational circuits. With a formulation of the problem using LUT (Look Up Table), assertions on circuits having tens of thousands gates can efficiently be generated using SAT solvers. The proposed method can be applied to embedded systems in general.

***Keywords:*** Assertion Based Verification, Formal Method, Assertion Generation, Simulation-based Verification

## 1. はじめに

アサーションベース検証 [1] は、シミュレーションでもまた形式的手法でも検査することができ、SoC に代表される

[1] 東京大学大規模集積システム設計教育研究センター
VLSI Design and Education Center, The University of Tokyo
[2] 東京大学大学院工学系研究科電気系工学専攻
Dept. of Electrical Engineering and Information Systems, The University of Tokyo
[a)] fujita@ee.t.u-tokyo.ac.jp
[b)] matsumoto@cad.t.u-tokyo.ac.jp
[c)] jo@cad.t.u-tokyo.ac.jp

大規模ハードウェア設計に対するもっとも効果的な検証手法の 1 つとして広く利用されている。検証品質は、チェックしたアサーションの質と量に依存し、よく考えられた十分な量のアサーションは、設計の初期段階で多くの設計バグを検出することができる。アサーションは、検証のみではなく、設計の要点を説明するドキュメントとしても有用である。アサーションベース検証が効果的にバグを検出するためには、ハードウェア設計記述言語（HDL）の 5 行から 10 行ごとに 1 つのアサーションが必要だと言われている。残念ながら、設計者がこの量のアサーションを迅速に用意することは、現実には難しいため、仕様や設計記述からアサーションを自動生成する技術が研究開発され、実際の設計でも利用されている。正しくない設計記述から生成されたアサーションは誤っている可能性があるが、自動生成されたアサーションを設計者が検査することにより、有益な情報として処理できる。

本稿では、組合せ回路（あるいは一定回数時間軸方向に展開された順序回路）に対する設計記述からの自動アサーション生成に関し、以下の結果を示す。

- LUT（Look Up Table）ベースの新しいアサーション生成手法を実験結果と共に示す。
- 提案手法では、十万ゲート規模の設計に対し、設計者の指定した信号間の論理的な関係を自動的に抽出できる。現状では、信号は人手で指定する。
- 提案手法は、アサーション候補の生成と反例に基づく候補の絞り込みの 2 つのプロセスから構成される。
- 提案手法がアサーションを生成する際には、少数のテストベクターも同時に生成され、そのテストベクターに対してアサーションが成り立てば、すべての入力ベクターに対してアサーションが成り立つことが保障される。数百入力の回路に対しても、通常数百個以下のテストベクターが生成される。つまり、これら数百個のテストベクターでアサーションが成り立てば、すべての入力ベクター（N 入力の組合せ回路では $2^N$ 個のテストベクター）で成立することが保障される。
- 提案するアサーション生成手法は、回路最適化にも適用できる。生成されたアサーションは、同じ論理を別の形で実現する手段を示しており、回路最適化のための変換に利用できる。例えば、後の例で示すように、リップルキャリー加算器からキャリールックアヘッド加算器を生成できる。

なお、LUT は問題の数学的定式化のためのみに利用しており、回路の実装とは無関係である。また、[2], [3] に示されている、distinguishing input という考えを利用しても実現できる。特に、 [3] で示されている guided random によるテストベクター生成手法は本提案手法内で、効率よく利用できると考えられる。

本稿は、以下の構成になっている。まず 2 章では例題を用いて、本提案手法と他手法との関連について説明する。続く 3 章で提案手法の一般化について説明し、4 章で実験結果を示す。最後の 5 章で結論と今後の課題について議論する。

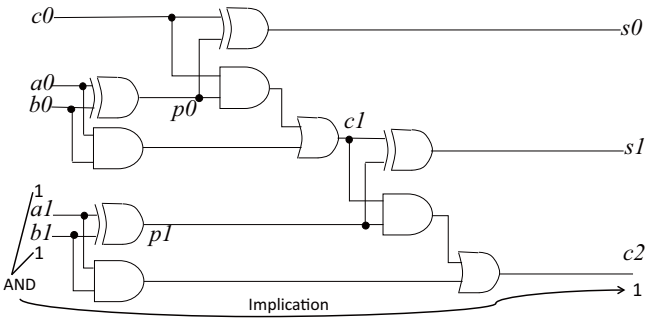


**図 1** Ripple carry adder and an implication

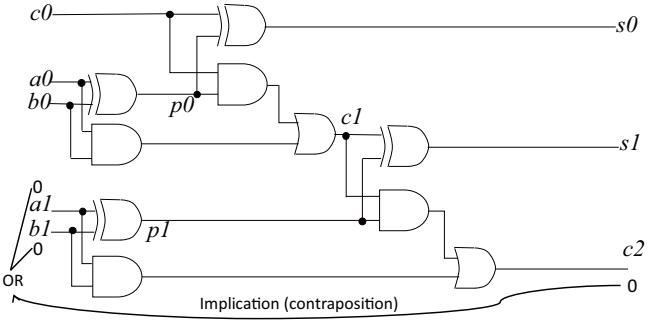


**図 2** Contraposition of the implication in Figure 1

## 2. アサーション生成例

本章では、図 1 に示す 2 ビット加算器を例として、提案手法や関連手法の考え方について説明する。$a0, a1$ と $b0, b1$ がそれぞれ 2 ビットの入力であり、 $c0$ はキャリー入力である。そして $s0, s1$ と $c2$ が 2 ビット出力とキャリー出力である。

アサーションを生成するもっとも簡単な方法は、テスト生成などで利用されている回路中の “implication” を計算することである。implication とは、回路中の信号に部分的に値を仮定し、それらの値の伝搬から、信号値に対する論理的な関係を抽出することである。例えば、図の加算器の場合、 $a1$ と $b1$ が共に 1 の場合、$c2$ は常に 1 となる。implication が得られると、その対偶も成立するので、図 2 に示す関係も成立する。図から、 $c2$ が 0 の場合, $a1$ か $b1$ の少なくとも片方は 0 でなければならないことがわかる。このように結論の部分が disjunctive になっている場合には、通常の implication ではこれ以上推論を進めることができない。implication は “learning” とも呼ばれており、SAT ソルバーの中でも効率化のために広く利用されている。implication を効率よく利用することで、故障に対する高速自動テストパタン生成（Automatic Test Pattern Generation, ATPG）[4], [5] が開発されている。

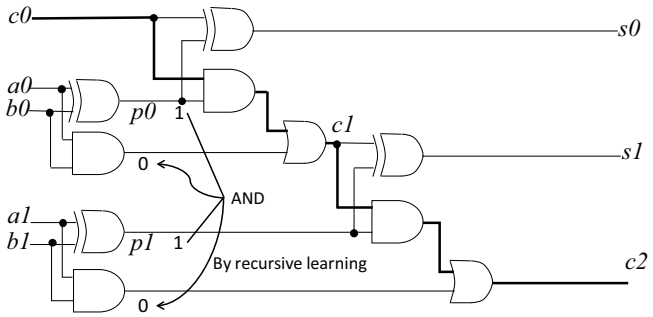


**図 3** More complicated implication: Recursive learning example

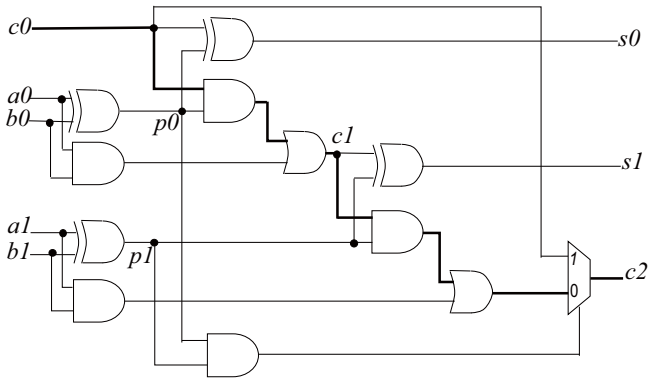


**図 4** Carry bypass adder generated through implications

implication の結論部分が disjunctive になっている場合にさらに解析を進めるには場合分けが必要で、 “recursive learning” と呼ばれる [6]。recursive learning では、結論のそれぞれの場合を別々に解析し、結果的にすべての場合で成立するものを探す。例題の加算器に recursive learning を適用した結果を図 3 に示す。まず $p0$ と $p1$ の両方が 1 であるとする。$p0$ が 1 で $p0$ は exclusive-OR ゲートの出力であるため、入力である $(a0, b0)$ の値は、(0,1) か (1,0) となる。いずれの場合も、 $a0$ と $b0$ に接続されている AND ゲートの出力は 0 となる。図から分かるように、同様の議論は $p1 = 1$ の時も成立する。これらを合わせると、回路で太線で示された信号パス上のゲートのサイド入力はいずれも non-controlling 値（OR ゲートに対しては 0、AND ゲートに対しては 1）となり、$c2$ の値と $c0$ の値は同じになることがわかる。これから、次のアサーションが導かれる：$(p0 \wedge p1) \Rightarrow (c2 = c0)$。

このアサーションを活用すると、リップルキャリー加算器を図 4 に示すようなキャリーバイパス加算器に変換することができる。図に示すように、$p0$ と $p1$ が共に 1 である場合には、キャリー出力である $c2$ を直接キャリー入力である $c0$ に接続することができる。$(p0 \wedge p1) \Rightarrow (c2 = c0)$ のようなアサーションを implication に基づく方法で生成するためには、$p0$, $p1$, $c0$, $c2$ に対し、適切な値を入れて回路の振舞を解析する必要があり、基本的に試行錯誤が必要で、必ずしも容易ではない。

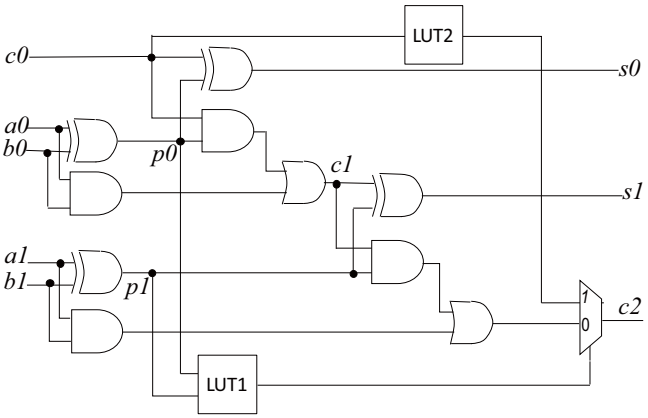


**図 5** LUT-based assertion generation for the carry bypass

提案手法では、まず、論理的な関係を抽出したい変数を決める。現状では、これらは設計者らにより人手で指定されると仮定している。implication のようにこれらの変数に試行錯誤的に入れていくのではなく、図 5 に示すように LUT とマルチプロセッサを回路中に挿入する。この回路を解析することで、以下に示す形のアサーションが生成される。

$$Logic\_LUT1(p0, p1) \Rightarrow (c2 = Logic\_LUT2(c0)) \quad ...(1)$$

LUT はその入力に関するすべての論理関数を表現することができるため、これは、変数 $p0$, $p1$, $c0$, $c2$ を含む一般的なアサーションを表現しているとも言える。そのためには、LUT1 と LUT2 は適切な論理関数を表現する必要があるが、この問題は、一般的には、文献 [7] に示されているように Quantified Boolean Formula （QBF）問題となる。本稿では、文献 [8], [9], [10] に示されている手法を基づき、SAT ソルバーのみを利用して QBF 問題を解く。後述の実験結果に示されているように、本手法は 10 万ゲート規模の回路を扱うことができる。

なお、図 5 に示す回路において LUT1 が定数 0 関数となると、いつも図 1 に示す回路と等価になる。これを避けるため、LUT1 は定数 0 関数以外のものを探すようにする。

## 3. 提案手法

我々の提案手法は、我々が従来から提案している、LUT による数学的定式化に基づく論理回路の部分合成 [9], [10] を利用している。この手法は、回路中の一部が不明な（部分回路の入力は決定しているが、その内部がどのようになっているかが分からない）場合に、不明部分を LUT として扱い、LUT としてどのような関数を実現しなければ、回路全体が正しくならないかという数学的問題に定式化し、それを QBF ソルバーではなく、SAT ソルバーを繰り返し利用して解いている。SAT ソルバーを利用していることから、10 万ゲート規模の回路にも適用できることが分かっている [10].

前章で例を用いて説明したアサーション生成手法は、以

下のように一般化できる。すなわち、図 5 に示す 2 ビット加算器に対する LUT を用いた定式化は、図 6 に示すように一般化できる。まず、図 6 に示す LUT の入力を決定する。現状では、これは設計者によって与えられると仮定している。自動的に適切な入力を決定することは、重要な問題であるが、ここでは触れない。すると残された問題は、回路全体の論理として、LUT1 と LUT2 に対する適切な論理を決定し、LUT1 と LUT2 が無い回路と等価になるようにすることである。これは、文献 [9], [10] で解いている問題と基本的に同じであり、それらに対する手法がそのまま適用できる。まず、1 つの回路全体に対する入力に対して正しく動作するように LUT の関数を決定する。これは、SAT 問題として定式化でき、定式化に現れる変数も LUT に関するものだけであり少なく、迅速に解くことができる。次に、得られた LUT の論理関数の解の候補が本当に正しいか否かを LUT をそのようにプログラムした回路と仕様との等価性を検証する。これも SAT 問題となるため、SAT ソルバーでそのまま解くこともできるし、文献 [5] に示されるような組合せ回路に対する等価性検証ツールを利用して解くこともできる。後者は、内部では SAT ソルバーだけでなく、様々な手法が適用されており、より大規模な回路に対して等価性を検証することができる。

文献 [9], [10] の実験結果によると、LUT に必要な論理を同定するのに必要なテストベクターの数は、数万ゲート規模の回路に対しても、LUT の入力数が数個程度の場合には、数十から数百で十分であることが示されている。つまり、アサーションを完全に同定するのに必要なテストベクターの数は、回路全体の入力数が数百あっても、非常に少なくて済むということである。これらのテストベクターに対して、アサーションが成立していることがシミュレーションなどで分かれば、その時点で回路全体の入力に対するすべての可能なテストベクターに対して、そのアサーションが成立することが保障される。これから、例えば、シミュレーションのためのテストベクターが適切に選ばれていれば、ごく少数のシミュレーション結果から自動生成されたアサーションがすべてのテストベクターで成立することになる。

このアサーション生成問題は、文献 [9], [10] で扱っている問題と基本的には同じであるが、1 つ制限が追加される。元の回路の信号と LUT1 によって生成された信号を選択しているマルチプレクサの制御入力、つまり LUT2 の出力が、いつも元の信号を選択する、つまり 0 定数関数だと、回路が自明に正しくなるが（元の回路と同じなので）、アサーション生成はできない。つまり、LUT2 は 0 定数関数であってはならない。これは単に LUT2 の真理値表がすべて 0 であってはならないということであり、これを条件として追加して処理する。LUT2 が定数 0 関数ではないとして、提案手法を図に示す 2 ビット加算器に適用すると、期待通りのアサーションである $(p0 \wedge p1) \Rightarrow (c2 = c0)$ が得られる。この加算器の入力は 5 個あるので、すべてのテストベクターの数は、$2^5$=32 となる。しかし、文献 [9], [10] を適用すると、3 個のテストベクターで、100%アサーションが成立することが確認できることがわかる。このように、100%正しいアサーションが、少数のテストベクターから生成できている。

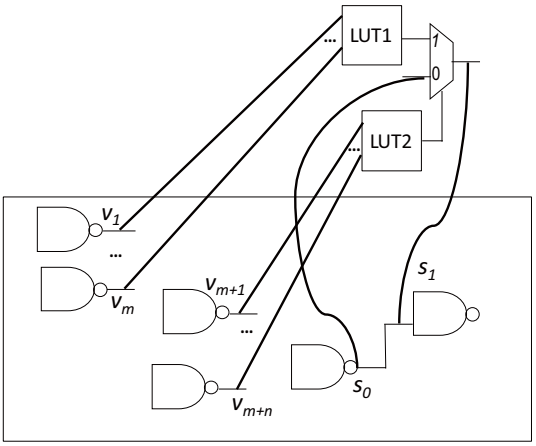


**図 6** General framework for LUT-based assertion generation

なお、アサーションが生成できない場合、つまり図 6 において、LUT1 や LUT2 の解が存在しないことが証明された場合には、LUT1 や LUT2 の入力を他のものに変えて、処理を続けることになる。後述の実験結果によると、LUT1 や LUT2 の入力をランダムに選んでも、多くの場合において、アサーションを見つけることができている。

## 4. 実験結果

提案するアサーション生成手法を文献 [9], [10] をもとに実装した。すべての実験は、4GB メモリと Intel Core2 Duo 3.33GHz をもつ PC 行った。また、実装では、回路変換などに AIGER [11] や ABC [12] を利用した。また、SAT ソルバーは MINISAT [13] を使っている。ベンチマーク回路としては、ISCAS85 の組合せ回路を利用している。

実験では、それぞれの回路に対し、LUT の大きさ（入力数）が 2、3、4、5 の 4 種類に対してアサーションを生成している。それぞれの大きさの LUT に対し、アサーションを調べるべき信号をランダムに選ぶ作業を 100 回行い、それらに対してアサーションを求めている。信号をランダムに選んでいるので、図 6 の回路にループができてしまう場合もあるが、その場合にはその信号の選択をやめ、次の選択に移っている。ループを生じない場合には、それらの信号間にアサーションが成立する場合もあるし、また成立しない場合もある。

実験結果を表 1 と表 2 に示す。実行時間がすべて秒であり、表 1 はアサーションが存在しない、つまり存在しないことを証明している場合であり、表 2 がアサーションに生

成に成功した場合である。ループがある場合は処理をやめているので、表 1 と表 2 の合計が 100 試行とはならない。表 1 では、それぞれのベンチマーク回路に対し、それぞれの LUT の入力数ごとに、100 試行中のアサーションが見つからなかった場合の回数、見つからない、つまり存在しないことを示すのに要した平均実行時間と、最大実行時間、またアサーションが存在しないこと示すための平均テストベクター数と最大テストベクターが示されている。実行時間は、最大でも数分であり、必要なテストベクター数も多くても数百で、アサーションが存在しないことを証明できている。

表 2 でも処理時間や必要なテストベクター数が同様である。表 2 には 2 つの実験結果が示されている。図 6 のように定式化すると、LUT1 と LUT2 は複数の解をもつ可能性がある。つまり、複数のアサーションがそれらの信号間に成立する可能性がある。表 2 の 5 列から 8 列までの結果は、解を 1 つ見つけるまでのものであり、9 列から 12 列までの結果は、解を見つけた後、他に解が無いことを証明するまでのものである。後者は前者の処理を含んでいる。処理時間やアサーション生成に必要な完全なテストベクター数は、先ほどと同様、それぞれ多くても、数分、数百となっている。

特に必要なテストベクターが少数で済むという事実は、少数のシミュレーション結果のみからアサーションを生成しても、100%正しいアサーションになっている可能性があるということであり、シミュレーション結果からのアサーション生成において、重要なことを示唆していると言える。

**表 1** アサーションが見つからない場合の実験結果

| Circuit | #Gate | #in of LUTs | Cases of no assertion | Ave. time | Max. time | Ave. #vector | Max. #vector |
|---|---|---|---|---|---|---|---|
| c499 | 202 | 2 | 59/100 | 1.5 | 4 | 7.4 | 20 |
| | | 3 | 36/100 | 2.9 | 8 | 14.1 | 36 |
| | | 4 | 31/100 | 7.1 | 25 | 25.9 | 77 |
| | | 5 | 34/100 | 15.3 | 63 | 40.4 | 132 |
| c880 | 383 | 2 | 72/100 | 2.4 | 5 | 11.3 | 20 |
| | | 3 | 56/100 | 6.3 | 12 | 25.1 | 40 |
| | | 4 | 49/100 | 15.3 | 31 | 45.9 | 73 |
| | | 5 | 39/100 | 49.0 | 91 | 95.2 | 138 |
| c1355 | 546 | 2 | 29/100 | 2.9 | 5 | 13.0 | 20 |
| | | 3 | 20/100 | 6.4 | 13 | 23.2 | 39 |
| | | 4 | 19/100 | 17.2 | 33 | 45.6 | 70 |
| | | 5 | 7/100 | 53.6 | 95 | 88.7 | 126 |
| c1908 | 880 | 2 | 47/100 | 3.1 | 5 | 11.8 | 19 |
| | | 3 | 35/100 | 7.8 | 16 | 23.3 | 37 |
| | | 4 | 24/100 | 22.7 | 44 | 45.1 | 68 |
| | | 5 | 21/100 | 70.4 | 144 | 86.4 | 130 |
| c2670 | 1193 | 2 | 62/100 | 4.3 | 10 | 11.5 | 20 |
| | | 3 | 61/100 | 12.3 | 29 | 23.2 | 40 |
| | | 4 | 50/100 | 39.4 | 87 | 46.2 | 74 |
| | | 5 | 45/100 | 130.3 | 315 | 88.0 | 144 |
| c3540 | 1669 | 2 | 41/100 | 3.9 | 9 | 10.4 | 19 |
| | | 3 | 34/100 | 11.5 | 26 | 21.6 | 37 |
| | | 4 | 10/100 | 33.1 | 87 | 40.6 | 73 |
| | | 5 | 15/100 | 138.9 | 304 | 89.7 | 139 |
| c5315 | 2406 | 2 | 79/100 | 7.5 | 14 | 13.3 | 20 |
| | | 3 | 72/100 | 20.1 | 45 | 24.5 | 40 |
| | | 4 | 72/100 | 64.2 | 160 | 47.4 | 80 |
| | | 5 | 60/100 | 235.9 | 578 | 93.9 | 154 |
| c6288 | 2406 | 2 | 5/100 | 4.4 | 6 | 8.4 | 11 |
| | | 3 | 2/100 | 10.5 | 11 | 15.5 | 16 |
| | | 4 | 0/100 | | | | |
| | | 5 | 0/100 | | | | |

## 5. 結論

指定された信号間にアサーションを自動生成する手法を提案し、組合せ回路対する実験結果を示した。LUT を利用して数学的定式化に基づいており、SAT ソルバーを利用して効率よくアサーションを生成したり、アサーションが存在しないことを証明することができる。実験結果から、2,000 ゲート規模の回路では、数分の処理時間で処理でき、また、アサーション生成に必要なテストベクター数も数百以下で済むことがわかる。

この少ないテストベクターでアサーションを同定できるということは、多くの示唆を含んでいる。シミュレーション結果からアサーションを生成する場合、少数の結果から生成しても、すべての入力に対して成立するアサーションとなっている可能性もあることになる。今後、種々の応用として検討していきたい。

## 参考文献

[1] Harry D. Foster, Adam C. Krolnik, David J. Lacey: Assertion-Based Design, Springer, 2004.
[2] Sumit Gulwani, Susmit Jha, and Ashish Tiwari: Automated modular synthesis applied to bit-vector circuits, *32nd ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, 2011.
[3] Susmit Jha, Sumit Gulwani, Sanjit Seshia, Ashish Tiwari: Oracleguided component-based pro- gram synthesis, *32nd International Conference on Software Engineering (ICSE)*, 2010.
[4] J.A. Waicukauski, P.A. Shupe, D.J. Giramma, A. Matin: ATPG for Ultra-Large Structured Designs, *International Test Conference (ITC)*, 1990.
[5] Jawahar Jain, Rajarshi Mukherjee, Masahiro Fujita: Advanced Verification Techniques Based on Learning, *32nd annual ACM/IEEE Design Automation Conference*, 1995.
[6] W. Kunz, D. Pradhan: Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits, *International Test Conference (ITC)*, 1992.
[7] Hratch Mangassarian, Hiroaki Yoshida, Andreas G. Veneris, Shigeru Yamashita, Masahiro Fujita: On error tolerance and Engineering Change with Partially Programmable Circuits, *17th Asia and South Pacific Design Automation Conference (ASP-DAC 2012)*, 2012.
[8] Mikolas Janota, William Klieber, Joao Marques-Silva, Edmund M. Clarke: Solving QBF with Counterexample Guided Refinement, *15th International Conference on Theory and Applications of Satisfiability Testing (SAT 12)*, 2012.
[9] Satoshi Jo, Takeshi Matsumoto, Masahiro Fujita: SAT-Based Automatic Rectification and Debugging of Combi-

**表 2** アサーションが見つかる場合の実験結果

| | #Gate | #in of LUTs | Found assertion | to find assertion | | | | to find no more assertion | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Ave. time | Max. time | Ave. #vector | Max. #vector | Ave. time | Max. time | Ave. #vector | Max. #vector |
| c499 | 202 | 2 | 21/100 | 0.29 | 2 | 1 | 8 | 1 | 3 | 7.3 | 17 |
| | | 3 | 34/100 | 0.44 | 2 | 1.7 | 13 | 3.2 | 6 | 19.4 | 35 |
| | | 4 | 42/100 | 1.1 | 8 | 4.9 | 34 | 7.0 | 17 | 33.1 | 69 |
| | | 5 | 44/100 | 2.0 | 18 | 7.6 | 59 | 20.2 | 69 | 61.5 | 144 |
| c880 | 383 | 2 | 12/100 | 0.42 | 2 | 1.8 | 8 | 0.67 | 3 | 2.9 | 11 |
| | | 3 | 25/100 | 0.36 | 2 | 1.1 | 7 | 2.8 | 8 | 13.7 | 36 |
| | | 4 | 32/100 | 2.0 | 16 | 7.0 | 48 | 12.1 | 29 | 39.8 | 72 |
| | | 5 | 27/100 | 2.1 | 20 | 7.0 | 58 | 18.9 | 70 | 44.4 | 126 |
| c1355 | 546 | 2 | 19/100 | 1.2 | 3 | 4.4 | 14 | 2.1 | 4 | 10.1 | 17 |
| | | 3 | 28/100 | 0.82 | 5 | 3.0 | 22 | 4.5 | 10 | 20.9 | 35 |
| | | 4 | 27/100 | 0.92 | 7 | 3.7 | 25 | 13.7 | 30 | 40.3 | 72 |
| | | 5 | 25/100 | 2.88 | 22 | 9.08 | 53 | 46.8 | 103 | 82.3 | 136 |
| c1908 | 880 | 2 | 17/100 | 1.5 | 4 | 5.4 | 14 | 2.7 | 5 | 11.4 | 17 |
| | | 3 | 23/100 | 0.78 | 3 | 2.9 | 14 | 5.4 | 11 | 19.1 | 32 |
| | | 4 | 23/100 | 0.87 | 6 | 2.7 | 19 | 13.6 | 36 | 33.1 | 64 |
| | | 5 | 30/100 | 0.7 | 4 | 2.6 | 13 | 52.8 | 167 | 69 | 144 |
| c2670 | 1193 | 2 | 19/100 | 1.5 | 5 | 3.9 | 12 | 2.7 | 7 | 7.1 | 16 |
| | | 3 | 22/100 | 1 | 6 | 2.5 | 14 | 8.5 | 20 | 17.3 | 33 |
| | | 4 | 36/100 | 1.9 | 14 | 4.1 | 26 | 26.1 | 67 | 33.2 | 65 |
| | | 5 | 39/100 | 3.5 | 26 | 6.2 | 37 | 86.3 | 306 | 62.8 | 143 |
| c3540 | 1669 | 2 | 39/100 | 0.92 | 5 | 2.3 | 13 | 2.5 | 7 | 7.2 | 16 |
| | | 3 | 39/100 | 0.79 | 7 | 1.7 | 16 | 5.69 | 21 | 13.4 | 35 |
| | | 4 | 50/100 | 1.3 | 8 | 2.9 | 17 | 22.4 | 75 | 30.4 | 69 |
| | | 5 | 59/100 | 5.5 | 200 | 5.5 | 112 | 79.7 | 308 | 61.2 | 141 |
| c5315 | 2406 | 2 | 8/100 | 0.38 | 1 | 0.13 | 1 | 3.9 | 9 | 8.4 | 16 |
| | | 3 | 13/100 | 1.4 | 7 | 3.1 | 13 | 17.7 | 33 | 22.6 | 35 |
| | | 4 | 13/100 | 3.7 | 15 | 6.15 | 21 | 61.6 | 121 | 46.3 | 70 |
| | | 5 | 29/100 | 14.4 | 122 | 13.1 | 69 | 241.9 | 539 | 93.6 | 149 |
| c6288 | 2406 | 2 | 80/100 | 0.45 | 2 | 0.41 | 4 | 4.4 | 178 | 1.3 | 10 |
| | | 3 | 80/100 | 0.44 | 3 | 0.49 | 6 | 7.1 | 379 | 1.8 | 19 |
| | | 4 | 74/100 | 0.76 | 19 | 0.89 | 15 | 3.1 | 58 | 2.8 | 27 |
| | | 5 | 76/100 | 0.70 | 24 | 0.64 | 26 | 3.4 | 46 | 3.3 | 38 |

national Circuits with LUT Insertions, *Asian Test Symposium (ATS)*, Nov. 2012

[10] Masahiro Fujita, Takeshi Matsumoto, Satoshi Jo: Partial synthesis through sampling with and without specification, *Intenational Conference on Computer Aided Design (ICCAD)*, 2013.

[11] AIGER Homepage *http://fmv.jku.at/aiger/*

[12] Robert K. Brayton, Alan Mishchenko: ABC: An Academic Industrial-Strength Verification Tool, *22nd International Conference on Computer Aided Verification (CAV 2010)*, pp.24–40, 2010.

[13] Niklas S Sorensson, Niklas Een: A SAT Solver with Conflict-Clause Minimization, *SAT*, 2005.