Vol.2017-SLDM-181 No.29
Vol.2017-EMB-46 No.29
2017/11/7

ゲートの種類とゲート入力信号探索による論理最適化・デバッグ手法

藤田 昌宏[1,a)]　ガラバギ アミル　マスー[2,b)]

概要：組合せ回路とそれと等価でない仕様、そして回路中の 1 つのゲートが与えられた時に、修正後の回路が仕様と等価となるような、そのゲートの種類とゲートの入力信号を探索する問題を考える。これは基本的に論理設計のデバッグ問題であり、またゲートの入力信号線の探索ができるため、レイアウトを考慮した論理最適化や LUT（Look up Table）からなる回路の簡単化にも適用できる。従来からゲートの種類の変更のみ、あるいはゲートの入力信号の探索範囲を限定した手法は提案されているが、本提案手法ではゲートの入力信号は回路中のすべての中から、種々のコスト関数を考慮して選択することができる。基本的に SAT 問題と Set-covering 問題に帰着しており、数千ゲート規模以上でも問題なく探索できることを実験結果により示す。

キーワード：論理最適化、論理デバッグ、回路再構成、ゲート入力探索

Logic Optimization and Debugging method Based on Input and Type Selection of a Gate

Masahiro Fujita[1,a)]　Amir Masoud Gharehbaghi[2,b)]

Abstract: Here we discuss the following problem: Given a combinational circuit, a specification which is not equivalent to the circuit, and an internal gate in the circuit, make the circuit equivalent to the specification by changing the inputs and type of the gate. This is a basic problme for logic debugging and also can be used for layout-oriented logic minimization and optimization of LUT (Look up Table) based circuits, as appropriate inputs to the gate can be automatically searched. While there have been proposed techniques which only change the types of gates and search inputs of gates with very limited ways, the proposed method can search a cost effective set of inputs to the gate out of all internal signals and primary inputs in the circuit. The proposed method reduces the problem into SAT and set-covering problems, and we show experimentally that circuits having thousands of gates can be processed.

Keywords: Logic optimization, Logic debugging, Circuit restructuring, Fan-in selection

1. はじめに

本稿では、仕様と不一致な組合せ回路が与えられた時、回路中の 1 つのゲートの入力信号とそのゲートが実現する関数（ゲートのタイプに相当するが、ここではそのゲートは AND や NOR などの特定の種類の論理関数だけではなく、任意の論理関数を実現できるように変更可能とする）を変更することにより、仕様と等価にする問題を考える。これは、論理設計デバッグ作業における基本操作であるとともに、回路構造を変更することもできるだめ、論理最適化における基本操作であるとも言える。

従来から、回路中のゲート（1 つまたは複数）のタイプを変更することのみにより、上記の問題を解く手法 [1][2]

[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] 東京大学工学系研究科
School of Engineering, University of Tokyo
[a)] fujita@ee.t.u-tokyo.ac.jp
[b)] amir@cad.t.u-tokyo.ac.jp

Vol.2017-SLDM-181 No.29
Vol.2017-EMB-46 No.29
2017/11/7

や、そのゲートの入力を予め決めておいた内部信号の集合の中からのみ探索する手法 [3][4][5] などが提案されている。一般に、ゲートの種類のみを変更する場合には、それらのゲートを LUT（Look up Table）に置き換えることにより、QBF（Quantified Boolean Formula）問題に帰着できる。しかし、ゲートの入力を変更する場合には、単純に QBF 問題として定式化することはできない（あるいはまだできていない）。

しかし、ゲートのタイプのみを変更するだけでは、デバッグできる範囲が大きく限定されてしまう。このため、ゲートの入力も変更する必要があるが、一般に回路中には多数の信号が存在し、それら全てがゲートの入力となる候補であるため、それらを順にチェックしていくような単純な探索は、回路規模が一定以上の場合は現実的ではない。

そこで、多数ある内部信号から、ゲートの入力とすることで、回路全体を正しくできるようなものを効率的に探索する手法が望まれる。本稿では、全ての内部信号や外部入力から、ゲートの入力を効率的に探索する手法を提案し、実験により評価する。最終的に SAT 問題と Set-covering 問題に帰着できるため、数千ゲート以上の規模の回路にも適用できる。

また、ゲートのタイプと入力の両方を変更すると、回路の構造や構成が変更できることになり、一般的な多段論理最適化問題にも適用できると考えられる。さらに、論理デバッグ問題と ECO（Engineering Change Order）問題は基本的に同じであり、強力な ECO 手法としても利用可能である。

本論文は以下の構成になっている。次章では、例題を通して本稿で扱う問題を明確化する。3 章で提案手法を示す。続く章では、ITC99 ベンチマーク回路を用いた実験結果を示し、提案手法に関するまとめを示す。

2. 1 つのゲートのタイプと種類を変更することによる論理デバッグ

ゲートのタイプのみを変更し、そのゲートの入力は変更しないことにすると、実設計のバグによる実験結果では、デバッグできる範囲が半分程度になってしまうことが分かっている [5]。そのため、一定範囲の内部信号から選択する手法が提案されているが、それでもデバッグできる範囲は限定されてしまうため、可能な範囲でより多くの内部信号から選択することが望ましい。特に仕様と実装の間のずれが論理的に「小さい」場合には特にそうである。

これについて明確化するために、ここでは、仕様とその実装のずれが論理的に最小である場合を考える。議論を簡単にするために、仕様と実装された回路の間のずれは、1 つの出力に対してのみで、その他の出力は完全に正しいとする。図 1 上に示すように、元の仕様 S を正しく実装している回路 I があるとする。これに対して、仕様の真理値表

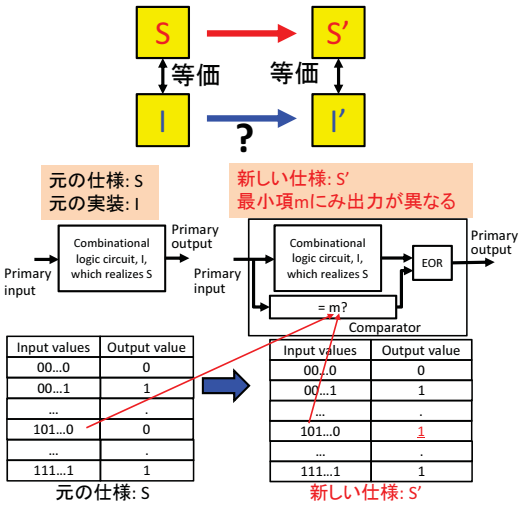


図 1　仕様の最小変更

の 1 つの行のみの出力を反転した仕様 S' を作成する。そして回路 I を修正して I' とし、S' と等価とすることを考える。仕様 S と S' の差は、真理値表では 1 つの行のみであり、論理的あるいは機能的な差は最小であると言える。なお、このような仕様変更は、Approximate computing 手法でも基本操作の 1 つである。

ここでは仕様 S' は人工的に作っているが、デバッグ問題としては、バグのある回路が実現している論理が仕様と真理値表で 1 つの行のみ異なる場合のデバッグをしていることになる。なお、仕様 S' は真理値表を直接記述してしまうと入力数の大きい回路を扱えなくなるので、図 1 下に示すように、元の仕様 S を正しく実装している回路に、真理値表の変更する行に対応する最小項を検出し、その場合だけ出力が反転するような構成にすることで、修正した仕様 S' を表現することにする。このようにすれば、論理的には、真理値表で 1 つ行のみ出力が異なる仕様 S' を多入力の回路に対しても表現できる。

積和形論理式として実装されている場合には、真理値表で 1 つの行のみ異なるように修正することは、例えばカルノーマップでの操作が一部異なるだけなので、比較的容易に実装を修正、つまりデバッグ可能な場合も多いと考えられる。しかし、実装が多段論理回路である場合には、以下に示すように、このような仕様変更に対するデバッグは一般的に非常に難しい。

以下、例を用いて議論していく。図 2(a) に ISCAS85 ベンチマーク回路の中で最も小さい回路である c17 を示す。最小項 01110、つまり x0=0, x1=1, x2=1, x3=1, x4=0 に対して、出力 y0 を反転させる回路（図 1 の S' に相当）を同図 (b) に示す。この際、同図 (a) の回路の各ゲートのタイプのみを変更して、同図 (b) と等価にする問題が、ゲートのタイプのみを変更してデバッグする問題となる。この例の場合、全てのゲートのタイプを変更可能とするには、同図 (c) に示すように各ゲートを LUT に置き換え、仕様

Vol.2017-SLDM-181 No.29
Vol.2017-EMB-46 No.29
2017/11/7

と等価となるような各 LUT の関数を探索する問題であり、文献 [6] に示す手法により、QBF 問題として解くことができる。なお、この処理はカリフォルニア大学バークレー校で研究開発が進められ、また配布されている論理合成検証ツール ABC[7] に実装されており、「&fftest -A 4」とコマンドを実行することで LUT の解を求めることができる（あるいは解が無い場合にはそれを証明できる）。実際この例の場合には、解は存在せず、全てのゲートのタイプを変更しても仕様に合わせることはできない。

従来の研究で扱っている、多段論理回路において、1 つのゲートのタイプを間違えるバグは、真理値表としては少数の行が変更されるのではなく、多数の行が変更されている。つまり、構造としてはバグは小さいが、論理や機能としては大きく変更されている。設計者が勘違いや記述間違いなどをした場合には、ゲートのタイプを間違えるバグが発生することも多いと考えられるが、ここではそのような大規模な機能変更ではなく、真理値表でごく少数の行のみ、間違った出力がだされるような、論理や機能の変更が非常に小さい場合について考える。

なお、図 1 の場合には、値を変更する最小項が何であれ、ゲートのタイプのみの変更では仕様に合わせることができないことは、簡単に証明できる。同図 (a) の元の回路では、出力 y0 は x4 に依存していない（回路中に入力 x4 から出力 y0 に到達できるパスが無い）。しかし、入力 x4 を含む 1 つの最小項に対してのみ出力 y0 を値を変更すると、その最小項の入力 x4 の値を反転した最小項に対しては正しい値のままなので、その出力値は必ず異なることになる、つまり新しい仕様では出力 y0 は入力 x4 に依存することになり、ゲートの種類の変更では対応できないことが分かる。

では、出力 y0 が元々依存する入力 x0, x1, x2, x3 にからなる最小項 1 つに対して出力値を変更する場合はどうであろうか？この例では、4 入力なので最小項は全部で 16 個あるが、全てのゲートのタイプの変更のみで新仕様に合わせられるは 1 つの場合だけであった。このように多段論理回路では、最小の仕様変更に対応するには、ゲートのタイプの変更のみでは一般的には非常に難しいと言える。そこで、文献 [8][9] に示す手法を基に、1 つのゲートのタイプだけでなく、その入力も変更することで、論理デバッグする手法について説明する。

3. SAT 問題と Set-covering 問題による定式化

1 つのゲートの入力とタイプの両方を変更場合、それによりデバッグ可能となる条件とは、その入力（の集合）に対する論理関数を適切に決めることにより、回路全体が正しくなるということになる。言い換えると、そのような論理関数が存在するような入力となる信号の集合を探索することになる。では関数が存在する条件とはどのようなことであろうか？関数とは 1 つ入力に対して 1 つの出力値が決まるということであり、出力値を異なるようにするには、必ず入力値も異ならなければならないということである。逆に全ての異なる出力値が要求される場合にゲートの入力値も異なれば、回路全体を正しくするそのゲートが実現すべき関数が存在すると言える。

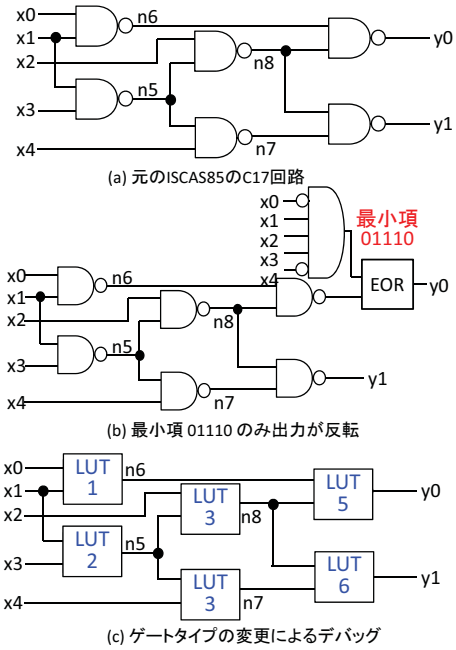


図 2　例題 c17 回路

つまり、異なる出力値が要求される外部入力値 in1 と in2 に対しては、ゲートの入力値 v1 と v2 も異なる必要がある。この条件を図示したものが、図 3 である。今、回路全体が正しくなるような、ゲートの出力信号 t に対する新しい入力とタイプを求めることを考える。入力値が in1 の場合には、t の値を 0 とすると外部出力は正しくなるが、t の値が 1 の場合には外部出力が正しくならないとする。同様に、入力値が in2 の場合には反対に、t の値を 1 とすると外部出力は正しくなるが、t の値が 0 の場合には外部出力が正しくならないとする。すると、入力値が in1 と in2 の時で、t は異なる値を実現しなければならない。つまり、t の関数の入力の値は異なる必要がある。今、v をゲートの入力候補とし、in1 と in2 の時の v の値をそれぞれ v1 と v2 とすると、v1 と v2 は値が異なる必要がある。ここで v は一般的には 1 変数ではなく、複数の変数のベクトルであるため、ベクトル値として異なる必要がある。

この条件を SAT 問題として定式化したものが図 3 に示されている。図では上の条件とその否定が示されており、否定が UNSAT つまり、否定を満たすような in1 と in2 が無いと証明されれば、そのような v はゲートの入力として正しく、適切な関数が存在することが保証される。一旦、ゲートに対する関数が存在することが保証されているゲートの入力が確定すると、あとは従来手法 [2] でゲートのタ

Vol.2017-SLDM-181 No.29
Vol.2017-EMB-46 No.29
2017/11/7

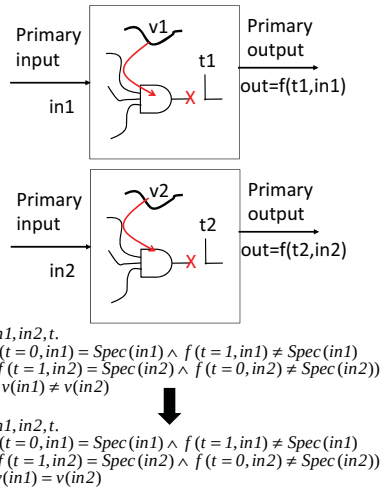


図 3 ある入力 v による論理関数を t で実現することにより回路全体が正しくなる条件

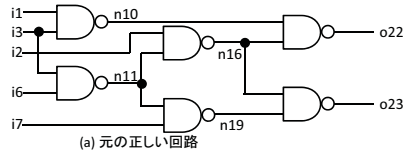
イプを決定できる。

ここまでは、ゲートの入力 v の候補があり、それが正しいか否かを判定するのみであり、入力 v の探索はできていない。ではどのようにすれば探索できるであろうか？入力 v が間違っている場合には、図 3 の下に示す式は SAT となる。その時の in1 と in2 の値に対しては、v の値は異なる必要があり、それは v に対する「必要条件」となる。このような必要条件としての in1,in2 のペアを蓄積していくことにより、v の候補の範囲が順次狭まっていき、最終的に候補を絞り込めると考えられる。

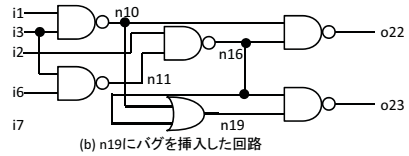
紙面の都合でアルゴリズムの詳細を示すのは難しいので、ここでは例で説明することにする。図 4(a) に元の回路を示し、(b) に誤った回路を示す。つまり、(b) の回路に対し、ゲート n19 の入力と論理関数を適切に選ぶことにより、(a) と等価にする問題を考える。n19 を外部入力の関数として考えると、比較的容易に同図 (c) に示すように i3, i6 i7 でデバッグできることが分かる。しかし、実際には同図 (a) とすればよく、内部信号も利用することで、2 入力ゲートとして実現できる。

同図 (b) の回路における n19 の入力である、n10 と n16 を v として、図 3 の下の式の SAT 問題を解くと、SAT になり、in1=10001, in2=01111 が得られる。この in1 と in2 の外部入力値に対しは、値が異なる内部信号や外部入力信号を選択する必要がある。そのような信号の 1 つとして、図 3(d) から、例えば n11 を選択できる。つまり、現在は v の候補として n11 の 1 つのみを選択している。今度は、前回の in1, in2 以外の値で v を n11 のみにした場合に図 3 の下の SAT 問題を解くと、SAT となり、図 5 に示すように、今度は in1=10000, in2=00001 が得られる。

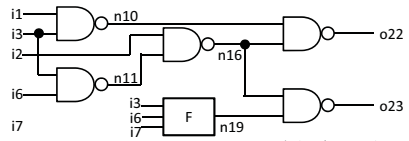
これで in1, in2 それぞれ 2 つずつ入力値が得られている。in1 に対しては n19 は 0 でなければならず、in2 に対しては n19 は 1 でなければらならず、また in1, in2 それぞれ 2 つ



(a) 元の正しい回路



(b) n19にバグを挿入した回路



(c) n19を外部入力の関数としてデバッグした回路

i1	i2	i3	i6	i7	n10	n11	n16	n19	o22	o23
1	0	0	0	1	1	1	1	0	0	1
0	1	1	1	1	1	0	1	1	0	0

(d) 正しい回路にin1, in2に対するシミュレーション結果

i1	i2	i3	i6	i7	n10	n11	n16	N19	o22	o23
1	0	0	0	1	1	1	1	0	0	1
1	0	0	0	1	1	1	1	1	0	0

(e) tが0の時のみ正しくなる入力in1

i1	i2	i3	i6	i7	n10	n11	n16	N19	o22	o23
0	1	1	1	1	1	0	1	0	0	1
0	1	1	1	1	1	0	1	1	0	0

(e) tが1の時のみ正しくなる入力in2

図 4 ゲート n19 の適切な入力を探索する例

i1	i2	i3	i6	i7	n10	n11	n16	n19	o22	o23
1	0	0	0	0	1	1	1	1	0	0
0	0	0	0	1	1	1	1	0	0	1

(a) 正しい回路にin1, in2に対するシミュレーション結果

i1	i2	i3	i6	i7	n10	n11	n16	N19	o22	o23
0	0	0	0	1	1	1	1	0	0	1
0	0	0	0	1	1	1	1	1	0	0

(b) tが0の時のみ正しくなる入力in1

i1	i2	i3	i6	i7	n10	n11	n16	N19	o22	o23
1	0	0	0	0	1	1	1	0	0	1
1	0	0	0	0	1	1	1	1	0	0

(c) tが1の時のみ正しくなる入力in2

	i1	i2	i3	i6	i7	n10	n11
a	1	1	1	1	0	0	1
b	0	0	0	0	1	0	0
c	0	1	1	1	0	0	1
d	1	0	0	0	1	0	0

(d) 解くべきSet-covering問題

図 5 ゲート n19 の適切な入力を探索する例のつづき

ずつ入力値が求まっているので、今度は合計 4 通りに場合（以下、それら場合を a, b, c, d と呼ぶ）について、v の値は異なる必要がある。それを表の形で表現したものが図 5(d) である。a, b, c, d はそれぞれ in1 と in2 の値の組合せであり、個々の組合せに対して値が異なる信号は 1 とし、同じ場合には 0 とした表である。これは covering table と呼ばれるもので、全ての行に少なくとも 1 つの 1 があるような最小の列の集合を求めることになる。この Set-covering 問題に対しては、種々のアルゴリズムが提案されており、それを用いて解くことになる。

この手法を用いて、c17 の回路に対して、図 2 に示すように真理値表で 1 つの行のみ変更を行った仕様に対するデバッグ結果を図 6 と図 7 に示す。これらから分かるように、どの最小項に対してもデバッグ可能となっている。

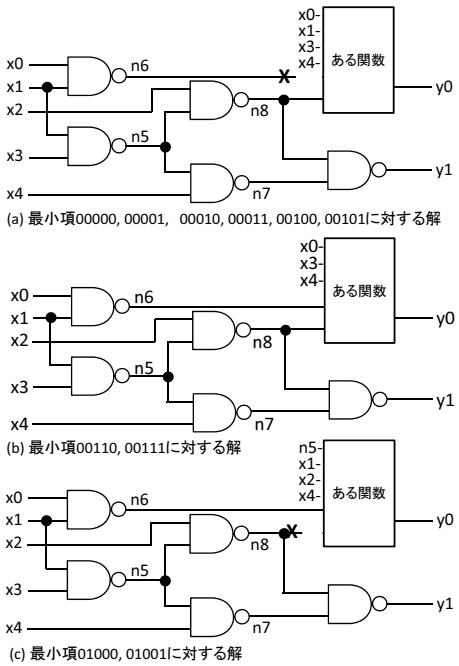


図 6 例題 c17 回路に対する 1 つの最小項の出力変更に対するデバッグ結果 (1)

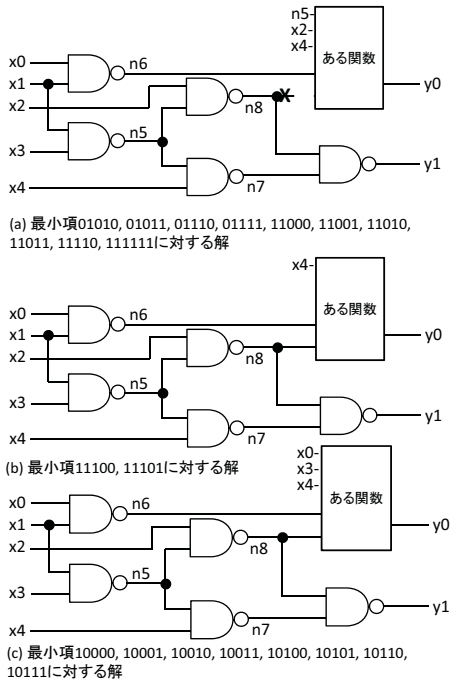


図 7 例題 c17 回路に対する 1 つの最小項の出力変更に対するデバッグ結果 (2)

4. 実験結果とまとめ

ITC99 ベンチマーク回路を用いて実験を行った。実験では、仕様変更を 3 種類定義している。それぞれ、1 つの最小項のみに対して出力値を反転した仕様、1 つの積項内の全ての最小項に対して出力値を反転した仕様、そして、2 つの積項内の全ての最小項に対して出力値を反転した仕様の 3 種類である。最小項と積項はランダムに生成し、回路の各出力 1 つずつに対して、それぞれ 10 回変更した仕様を生成して、実験した。例えば、b03 の回路の場合、外部出力は 28 あるので、合計 20*10=280 回実験している。

入力とタイプを変更するゲートは、実験を単純にするため、仕様変更のあった外部出力に直接接続されている、回路上最終段のゲート 1 つとしている。なお、利用した計算機は、128MB のメモリを持つ Dual Xeon E5-2690 2.9GHz であり、実行時間の単位は秒である。

実験結果を表 1、表 2、表 3 に示す。実行時間はいつも 30 秒以下であり、SAT 問題と Set-covering 問題に定式化して解くことで、ゲートの入力を回路中の全ての信号に対して探索することが可能となっている。表に示すように、デバッグに成功した割合は、最小項 1 つの変更の場合は 13%、積項 1 つの場合は 23%、積項 2 つの場合は 32%であった。つまり、仕様変更が真理値表の出力値の異なる数が多いほど、デバッグしやすいという結果となっている。多段論理回路で実装されている場合、論理的な仕様変更あるいは、仕様と実装との差が小さいほど、そのデバッグは難しいと言える。Approximate computing などでは、仕様変更を小さくしたいが、それに実装を合わせことは簡単ではない。

デバッグできた場合には、変換したゲートの入力数が、3 つの実験で順に平均で、1.1 (3.4-2.3)、 1.4 (3.7-2.3)、1.9 (4.2-2.3) 個多くの入力数がゲートに必要となっている。つまり、回路はそれだけ大きくなっている。Approximate computing では回路を小さくするような仕様変更を見つける必要がありが、それにはどうすればよいかについては、今後検討していく予定である。

参考文献

[1] I. Pomeranz, S. M. Reddy: On diagnosis and correction of design errors, *ICCAD*, Nov. 1993.
[2] Masahiro Fujita, Satoshi Jo, Shohei Ono, Takeshi Matsumoto: Partial synthesis through sampling with and without specification, *ICCAD*, Nov. 2013.
[3] P. Y. Chung, I. N. Hajj: Diagnosis and correction of multiple design errors in digital circuits, *IEEE Trans. on VLSI Systems*, vol. 5, no. 2, June 1997.
[4] Y. Kukimoto, M. Fujita, R. K. Brayton: A redesign technique for combinational circuits based on gate reconnections, *ICCAD*, Nov. 1994.
[5] K. Oshima, T. Matsumoto, M. Fujita: A debugging method for gate level circuit designs by introducing programmability, *VLSI-SoC*, Oct. 2013.
[6] M. Fujita, A. Mishchenko: Efficient SAT-based ATPG techniques for all multiple stuck-at faults, *ITC*. Oct. 2014.
[7] R. K. Brayton, A. Mishchenko: ABC: An Academic Industrial-Strength Verification Tool, *CAV*, 2010.
[8] A. M. Gharehbaghi, M. Fujita: A New Approach for Debugging Logic Circuits without Explicitly Debugging Their Functionality, *ATS*, Nov. 2016.
[9] A. M. Gharehbaghi, M. Fujita: A new approach for selecting inputs of logic functions during debug, *ISQED*, Mar. 2017.

Vol.2017-SLDM-181 No.29
Vol.2017-EMB-46 No.29
2017/11/7

表 1 ITC99 ベンチマーク回路に対する実験結果（最小項 1 つのみ変更）

	外部出力数	外部入力平均	ゲート入力平均	繰返し >100 回	解の数	実験回数	成功率 (%)	in1,in2 生成回数平均	実行時間平均	ゲート入力数平均	繰返し回数平均	総実行時間平均
b03	28	7.9	2.3	0	68	280	24	9.1	0.9	2.7	9.4	1.0
b04	65	12.4	2.0	1	135	650	21	10.2	2.0	3.2	12.0	2.4
b05	55	11.8	2.1	0	11	550	2	12.5	1.9	3.3	12.6	1.9
b06	6	4.0	2.8	0	0	60	0	4.9	0.2	2.6	4.9	0.2
b07	45	22.5	2.4	2	44	450	10	25.1	9.3	4.2	26.2	10.1
b08	21	7.0	2.0	0	57	210	27	7.3	0.2	3.1	9.0	0.2
b09	20	13.5	2.4	0	1	200	1	16.2	0.4	4.1	16.2	0.4
b10	17	9.4	2.6	0	69	170	41	7.8	0.2	4.0	9.1	0.2
b11	30	13.3	2.3	0	33	300	11	13.3	3.5	3.5	14.5	4.0
b12	116	14.0	2.3	46	71	1160	6	13.0	6.4	3.5	13.0	6.7
b13	46	7.0	2.1	0	97	460	21	6.6	0.3	3.1	7.3	0.3
平均	41	11.2	2.3	4.5	53.3	408.2	13	11.5	2.3	3.4	12.2	2.5

表 2 ITC99 ベンチマーク回路に対する実験結果（1 つの積項内の全ての最小項を変更）

	外部出力数	外部入力平均	ゲート入力平均	繰返し >100 回	解の数	実験回数	成功率 (%)	in1,in2 生成回数平均	実行時間平均	ゲート入力数平均	繰返し回数平均	総実行時間平均
b03	28	7.9	2.3	0	133	280	48	10.3	0.7	3.5	12.6	0.8
b04	65	12.4	2.0	1	126	650	19	11.6	1.8	3.5	13.0	2.1
b05	55	11.8	2.1	0	40	550	7	14.3	1.5	3.8	14.8	1.6
b06	6	4.0	2.8	0	0	60	0	5.0	0.1	2.8	5.0	0.1
b07	45	22.5	2.4	2	72	450	16	26.2	11.7	4.5	27.5	13.9
b08	21	7.0	2.0	0	36	210	16	7.7	0.1	3.1	8.6	0.2
b09	20	13.5	2.4	0	9	200	5	16.9	0.4	4.4	17.3	0.4
b10	17	9.4	2.6	0	57	170	34	7.9	0.2	3.9	8.2	0.2
b11	30	13.3	2.3	0	58	300	19	15.0	4.6	3.9	17.0	5.2
b12	116	14.0	2.3	46	368	1160	32	13.9	6.1	4.0	16.5	8.2
b13	46	7.0	2.1	0	115	460	25	7.2	0.1	3.1	7.9	0.2
平均	40.8	11.2	2.3	4.5	92.2	408.2	23	12.4	2.5	3.7	13.5	3.0

表 3 ITC99 ベンチマーク回路に対する実験結果（2 つの積項内の全ての最小項を変更）

	外部出力数	外部入力平均	ゲート入力平均	繰返し >100 回	解の数	実験回数	成功率 (%)	in1,in2 生成回数平均	実行時間平均	ゲート入力数平均	繰返し回数平均	総実行時間平均
b03	28	7.9	2.3	0	225	280	80	11.5	0.7	5.4	21.8	1.2
b04	65	12.4	2.0	1	165	650	25	13.3	2.4	4.5	18.1	3.2
b05	55	11.8	2.1	0	74	550	13	17.0	2.8	4.5	18.1	3.2
b06	6	4.0	2.8	0	0	60	0	5.2	0.1	2.7	5.2	0.1
b07	45	22.5	2.4	2	129	450	29	25.8	6.2	5.1	28.5	8.4
b08	21	7.0	2.0	0	56	210	27	9.4	0.3	3.5	10.9	0.3
b09	20	13.5	2.4	0	23	200	12	17.8	0.5	4.8	19.1	0.6
b10	17	9.4	2.6	0	85	170	50	8.4	0.2	4.2	9.6	0.3
b11	30	13.3	2.3	0	46	300	15	16.8	5.9	3.9	17.7	7.0
b12	116	14.0	2.3	46	445	1160	38	15.8	5.0	4.6	18.9	6.5
b13	46	7.0	2.1	0	174	460	38	8.0	0.4	3.6	9.9	0.4
平均	40.8	11.2	2.3	4.5	129.3	408.2	32	13.5	2.2	4.3	16.2	2.8