Vol.2013-SLDM-161 No.5
2013/5/16

プログラム可能な素子を利用したゲートレベル回路のデバッグ手法

大島 浩資[1,a)]　城 怜史[1,b)]　松本 剛史[2,c)]　藤田 昌宏[2,d)]

概要：VLSI が大規模化する中で，回路のデバッグを自動的に行うアルゴリズムの研究が重要となっている．本稿ではディジタル回路のデバッグ手法として，ルックアップテーブル (LUT) やマルチプレクサ（MUX）を用いた手法に注目する．LUT や MUX が挿入された回路を，製造後の故障や回路の修正に利用する手法は既に提案されている．しかし，この手法をデバッグに利用する場合には，LUT を挿入する位置を決定する必要があり，またデバッグのために，LUT の入力に信号を追加することが必要な場合もある．本稿では，デバッグを行うために LUT の入力に追加する信号を絞り込む手法を提案する．すなわちバグが生じた際，異なる複数の入力信号に対して LUT への入力が同じであるとき，値の異なる信号を追加することが必要になる．これを利用して実験を行い，産業用の例題に対して効率良く追加する信号を探索できることを示す．

キーワード：ゲートレベル回路，デバッグ，ルックアップテーブル，マルチプレクサ

A Debugging Method for Gate Level Circuit with Programmable Devices

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

Abstract:
As the size and complexity of VLSI circuits increase, automated debugging becomes more important. In this paper, we propose a debugging method using programmable circuits such as look up tables (LUTs) and multiplexors (MUXs). There exists a work where a circuit with bugs or faults can be corrected by deriving a configuration of LUTs and MUXs so that the circuit behaves correctly. However, for a given buggy circuit design, where LUTs and MUXs are inserted and which signals are connected to them for debugging is another challenging problem. In this work, we propose a method to identify a set of signals that need to be connected to one of inserted LUTs for debugging, so that a circuit under debugging can be corrected. By using a fact that a signal can be added to a set of input signals of an LUT when the values of the signal are different for two patterns where every other input signal has the same value, we can largely reduce the number of candidates signals. The experimental results on an industrial example show our method can efficiently find a set of signals that should be connected to LUTs for debugging.

Keywords: gate-level circuit, debugging, look-up table, multiplexor

1. はじめに

VLSI が様々な目的で利用されている現在，回路の規模はますます増大し，1 チップに 10 億個を超えるトランジスタを搭載するような製品が利用されている．回路規模の増大に伴い，回路の設計や検証，デバッグにかかるコストも大きくなっている．製造可能なトランジスタ数の増加に対

[1] 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
[2] 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
[a)] oshima@cad.t.u-tokyo.ac.jp
[b)] jo@cad.t.u-tokyo.ac.jp
[c)] matsumoto@cad.t.u-tokyo.ac.jp
[d)] fujita@ee.t.u-tokyo.ac.jp

し，設計効率の向上のスピードは遅れていて，設計生産性の問題と呼ばれている．また，回路を修正するために，再設計を必要とするケースも数多くある．設計全体の 6 割が 1 回以上のリスピン (製造後の再設計) を必要としていて，その多くがが回路機能の誤りを原因とするものだと言われている．

このような状況に対応するために，回路のデバッグを支援する技術が重要になっている。本稿では，ディジタル回路の設計において，ゲートレベルの回路を対象とするデバッグ手法について扱う．ゲートレベル回路のデバッグは，抽象度が低く，バグを人手で発見することが難しい．また回路にバグが発見され，論理合成を再度行う場合，時間的なコストも大きくなってしまうため，これを防ぐ必要がある．

本稿で扱うデバッグ手法は，Partially-Programmable Circuit（PPC）[1] の考え方を元に，バグのある位置にルックアップテーブル（LUT）やマルチプレクサ（MUX）を挿入し，LUT の真理値表や MUX の選択信号を適切に設定することで，回路のバグを修正するというものである．PPC は，回路のうち一部のゲートを LUT や MUX で置換し，製造後の故障に対応できるようにした回路である．この考え方を元に，ゲートレベル回路に挿入された LUT に対して，その真理値表を SAT ソルバを用いて効率的に探索する手法が提案されている [2].

これらの手法に基づき，本稿では，バグを含む回路が与えられた場合に，LUT や MUX を適切に挿入して，回路を修正する手法を提案する．すなわち，バグが生じた場合に，バグの原因となっているゲートを LUT に置換し，その真理値表を正しく定めることで回路のデバッグを行う．しかし，この方法では，回路の接続に問題がある場合に，ゲートを LUT に置換するだけでは，バグを修正することができない．

このようなバグの修正に対応するため，本稿では挿入した LUT に信号を追加する手法を提案する．信号を追加する手法においては，追加する信号の候補が増えるほど計算時間が長くなると考えられるため，バグを修正するために必要な信号を効率良く調べる手法が必要である．本稿では，MUX を利用して複数の信号をまとめて調べる手法と，値に注目して追加する候補となる信号を絞り込む手法を提案する．これらの手法により，ゲートを LUT に置換するだけでは対応出来なかったバグを修正できることを述べる．

本稿の流れは以下の通りである．まず第 2 節で，関連研究として LUT の真理値表を探索する手法について述べる．次に第 3 節で，本稿で提案する，バグを修正するために必要な信号を LUT に追加する手法について述べる．そして，第 4 節で実験結果を述べ，第 5 節でまとめと今後の課題を述べる．

2. 関連研究: LUT の真理値表の探索

PPC において故障やバグに対応するためには，LUT の真理値表を適切に定める必要がある．すなわち，LUT の真理値表を定めることで，回路が与えられた入力に対して所望の出力を返すようになれば，真理値表の変更で故障やバグに対応できたことになる．文献 [2] および本稿においては，仕様として正しい回路が与えられているものとし，真理値表を正しく設定できたか否かを，正しい回路と LUT を含む回路との等価性検証により調べる．

この問題の計算量を減らし，高速に計算する手法として，Counterexample-Guided Abstraction Refinement（CEGAR）と呼ばれる手法が提案されている [3][4]．CEGAR を用いた真理値表の探索は，まず回路への入力のうちいくつかの値に対して仕様を満たす真理値表の有無を調べる．調べる入力の集合を $X = (x_0, \cdots, x_n)$ とすると，式 (1) を満たす v を求める問題となる．

$$\exists v. \bigwedge_{i=1}^{n} f(v, x_i) = SPEC(x_i) \quad (1)$$

ただし，v は LUT の真理値表で，この問題において求めたいのは式 1 を満たす v だということになる．また x は回路全体の primary input，$SPEC$ は回路が満たすべき仕様である．

これは SAT 問題であるので，SAT ソルバを用いて解くことができる．ここで，条件を満たす真理値表 v が得られなかった場合には，回路を仕様通りに修正できる真理値表は存在しないため，探索を終了する．条件を満たす真理値表 v が得られた場合，次にその真理値表を用いた場合に，仕様を満たさなくなるような入力が存在するか否かを調べる．これは式 (2) で表される．

$$\exists x. f(v, x) \neq SPEC(x) \quad (2)$$

そして，再び SAT ソルバを用いてこの問題を解く．このような x が存在する場合には，x は真理値表 v に対する反例であり，その x を x_{n+1} として調べる入力 X に追加し，もう一度式 (1) を解く．x が存在しない場合には，真理値表 v によって，すべての入力に対して回路の出力が仕様通りになるということなので，v を解として探索を終了する．以上のフローをまとめると，図 1 となる．

文献 [2] では図 1 のフローを用いて実験を行い，高速に解を求めることに成功している．

3. デバッグへの利用

3.1 プログラム可能な素子を利用したデバッグ手法

本稿では，第 2 節の手法を応用し，設計段階におけるデバッグの修正を行う．本節ではこの手法の概要を説明する．

まず，手法が扱う対象として，バグを含むゲートレベル

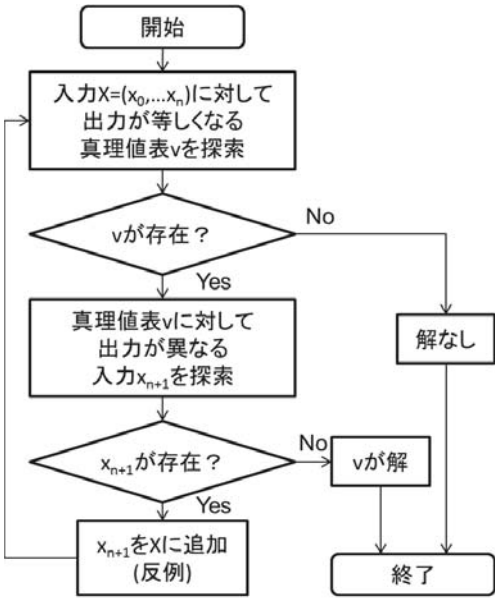

図 1 LUT の真理値表の探索フロー

の回路と，正しい仕様が与えられているものとする．本稿においては，仕様として，正しいゲート回路が与えられているものとする（ただし，回路構造は異なっていても構わない）．

手法の全体のフローとしては，まず，バグが含まれている回路において，バグを含んでいると考えられるゲートの候補を LUT に置換する．回路のバグ位置を特定する手法としては，複数の手法が提案されている [5][6] が，本稿の手法では，バグの位置を 1 ヶ所に絞らなくてもデバッグを行うことができる．例として，仕様を満たしていない出力からゲートを入力へ向かってさかのぼり，その上にあるゲートを LUT に置換するという方法を取ることができる．

そして，第 2 節の手法で，回路を仕様と等価にするための（デバッグするための）真理値表を探索する．真理値表が得られた場合，LUT を対応するゲートで再度置き換えれば，仕様と等価な回路が生成され，デバッグに成功したことになる．

真理値表が得られなかった場合には，LUT の真理値表の変更によってはデバッグを行えないことを意味する．このようになる原因としては，LUT の配置や接続が不足しているということがあり，それを修正して再び真理値表を探索する．本稿では第 3.2 節において，回路の接続についての問題点を述べ，それに対処するために本稿で提案する手法を，第 3.3，3.4 節で述べる．

以上のフローを図にまとめると，図 2 となる．

3.2 LUT に置換する手法における問題点

第 3.1 節で述べた手法において問題となるのが，ゲートを LUT に置換するだけでは修正できないバグが存在することである．

そのバグの例を図 3 に示す．図 3 のバグは，AND ゲートであるべき回路が OR ゲートとなっていることに加え，OR ゲートに入力として信号 D が接続されていない．ゆえに，ゲートを LUT に置換しても，LUT に信号 D が入力されていないので，どのような真理値表の変更によっても，バグを修正して正しい回路と等価にすることは不可能である．

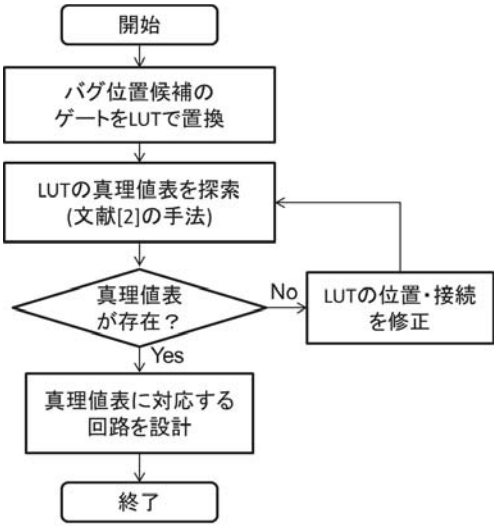

図 2 本稿で提案するデバッグのフロー

このようなバグに対応するためには，LUT に信号を追加する必要がある．このとき考える必要があるのが，追加する信号をどのように選択し，信号が追加された LUT の真理値表をどのように探索するかである．追加する信号の候補としては，回路全体の入力（primary input），回路の中間変数および出力，そしてそれらの論理関数で表される信号がある．しかし，これらの信号すべてを 1 つずつ LUT の入力に追加し，表 1 のフローを繰り返していては，信号数が多い場合に時間がかかってしまう．よって，バグを修正するために必要な信号の候補を絞り込み，それらに対して効率良く解の探索を行う手法が必要になる．

この問題に対して本稿は，あらかじめ絞り込まれた複数の候補に対してバグを修正可能な信号を探索する手法として，MUX を利用する手法を提案する．さらに，追加する信号の候補を，入力が与えられたときに実際に取る信号値に基づいて絞り込む手法を提案する．第 3.3，3.4 節では，バグの原因となっているゲートがあらかじめ LUT に置換されているものとし，真理値表が存在しなかった場合に，信号を追加する手法と，追加する信号の候補を絞り込む手法について説明する．

3.3 提案手法 1: MUX を利用する手法

第 3.2 節で述べたように，信号を 1 つ 1 つ追加し，図 1 のフローで解を探索することも可能であるが，かかる時間も長くなってしまう．

本節で提案する手法では，LUT に信号を追加する際に，追加する候補となる信号をまず MUX に入力し，その出

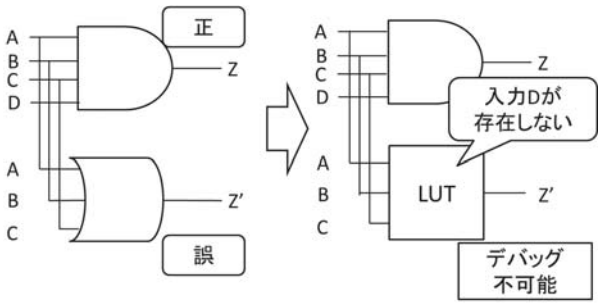

図 3 デバッグ不可能な回路の例

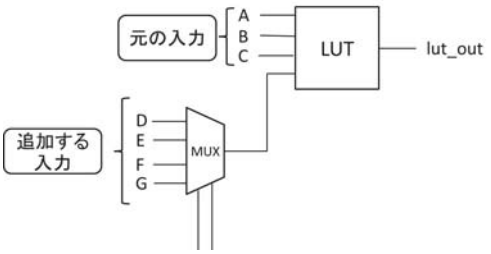

図 4 MUX の挿入

力を LUT に追加する．すなわち，バグの原因となっているゲートが LUT に置換されているとき，MUX を追加し，図 4 のような回路を作成する．図 4 の例において，信号 A,B,C は，LUT で置換されたゲートが元々持っていた入力で，信号 D,E,F,G は，追加すべき信号の候補である．そして，追加すべき信号の候補のうち 1 つが MUX の選択信号によって選ばれ，LUT の入力に追加される．

このような回路を作成し，回路を修正するための MUX の選択信号を，LUT の真理値表と合わせて図 1 のフローで探索する．解が得られた場合，その解で選択されている信号がゲートに追加すべき信号であったことになる．解が得られなかった場合には，MUX に挿入する信号の組を入れ替えて，もう一度探索を行う．この方法により，より多くの信号の候補を，信号を入れ替える回数を少なくして探索することが可能になる．

LUT ではなく MUX を使う理由は，MUX の選択信号の数が，LUT の真理値表の信号の数より少ないためである．例えば 4 入力の場合，MUX の選択信号は 2bit であるが，LUT の真理値表の信号は 16bit である．探索の対象となる信号の数はデバッグの実行時間に大きく影響するので，探索範囲を絞ることは重要である．

3.4 提案手法 2: 追加する信号の候補を絞り込む手法

LUT へ信号を追加する際，信号の候補を絞り込むことは重要である．第 3.3 節の手法を用い，複数の信号の候補をまとめて調べることも可能であるが，追加する信号の候補の数が膨大になれば，探索にかかる時間も非常に長くなると考えられる．

本節では，LUT へ追加する信号の候補を，信号が取る値に注目して絞り込む手法を提案する．

3.4.1 追加すべき信号についての考察

本節では，回路に挿入されている LUT が 1 つである場合について考える．このとき，図 1 のフローで解なしと判定され，バグを修正できない場合，以下の条件を満たす異なる入力 x_i，x_j が，図 1 の探索フローにおける入力 X の中に含まれている（図 5）．

(1) 回路全体の入力が x_i，x_j であるときの，LUT への入力 $l_{in}(x_i)$，$l_{in}(x_j)$ が等しい．よってこのとき，LUT の出力 $l_{out}(x_i)$，$l_{out}(x_j)$ も等しくなる．

(2) 出力 $y(x_i)$，$y(x_j)$ が仕様 $SPEC(x_i)$，$SPEC(x_j)$ を満たすために，LUT の出力 $l_{out}(x_i)$，$l_{out}(x_j)$ が異なっている必要がある．

図 5 において，回路全体の入力 x_i，x_j は異なっているので，回路が満たすべき仕様 $SPEC(x_i)$，$SPEC(x_j)$ も異なっている場合がある．本節では，バグの原因となっているゲートは LUT で置換されていると仮定していて，仕様を満たさない出力（$s_k(x_j)$ とする）があれば，LUT の出力 $l_{out}(x_j)$ が変われば $s_k(x_j)$ の値も変わる可能性がある．しかし，LUT への入力 $l_{in}(x_i)$，$l_{in}(x_j)$ は等しいため，いかなる真理値表 v によっても，LUT の出力 $l_{out}(x_i)$，$l_{out}(x_j)$ は等しくなる．よって，仕様 $SPEC_k(x_i)$，$SPEC_k(x_j)$ を同時に満たせない場合，LUT の真理値表を変更するだけでは，仕様を満たすように回路を修正することは不可能である．

この場合，仕様を満たすためには，入力 x_i，x_j が与えられたときの LUT の出力が異なっている必要がある．そのためには，LUT への入力 $l'_{in}(x_i)$，$l'_{in}(x_j)$ を異なるものとする必要がある．すると，入力 $l'_{in}(x_i)$，$l'_{in}(x_j)$ に対応する真理値表の値を異なるものとすることで，出力 $l_{out}(x_i)$，$l_{out}(x_j)$ も異なるものとなり，仕様を満たすことができる可能性がある．

よって，図 5 の状態からデバッグをするためには，LUT の入力を異なるものとするために，入力 x_i，x_j が与えられているときに値が異なっている信号を，LUT の入力に追加することが必要である．追加する信号を $A(x)$ とし，$A = 0$ かつその他の入力が l_{in} であるときの真理値表を $P_{\bar{A}}(l_{in})$，$A = 1$ かつその他の入力が l_{in} であるときの真理値表を $P_A(l_{in})$ とすると，出力 l_{out} は $l_{out} = P_{\bar{A}}(l_{in}) * \bar{A} + P_A(l_{in}) * A$ となる．よって，入力 x_i のときの A の値が $A(x_i) = 0$，入力 x_j のときの A の値が $A(x_j) = 1$ となる信号を追加すれば，そのときの出力は $l'_{out}(x_i) = P_{\bar{A}}(l_{in})$，$l'_{out}(x_j) = P_A(l_{in})$ となり，$P_{\bar{A}}$ と P_A を異なる値とすれば，入力 x_i と x_j に対して出力 l'_{out} を異なるものにすることが可能である．

3.4.2 追加する信号の候補を絞り込む手法

第 3.4.1 節の議論を踏まえ，信号を追加する手法を提案する．まず，信号を追加する必要があるのは，図 1 のフローにおいて真理値表 v が存在しない場合である．このとき，図 5 に示す状態になっているため，信号を追加することで

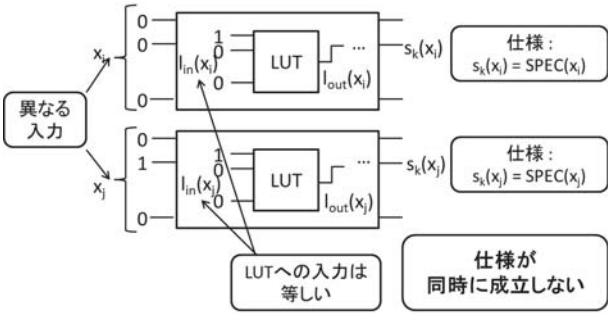

図 5 デバッグ不可能な例

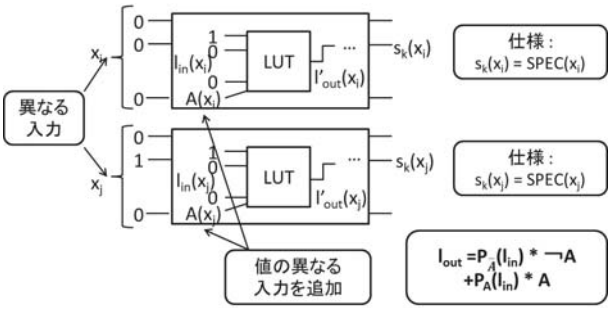

図 6 信号の追加

デバッグできる可能性がある．

次に，追加する信号の候補は，LUT への入力が等しくなるような 2 つの異なる回路全体への入力 x_i，x_j があった場合に，その入力を与えたときに異なっている信号となる．ここで，図 1 のフローで真理値表 v が存在しないとなった場合，図 5 に示す状態は最後に追加した入力（x_n とする）が原因で生じているので（それ以前の入力 $x_1, \cdots, x_{n-1}$ に対して回路と等価にできる真理値表が存在しない場合，その時点で解なしと判定される），$x_j = x_n$ である．

よって，追加する信号を探すために，まずは x_i を図 1 の X の中から探す．x_i の候補となるのは，入力が x_j であった場合と LUT への入力が等しくなる入力であり，このことはシミュレーションで調べることができる．

次に，LUT へ追加する信号の候補として，入力 x_i，x_j が与えられたときに，値が異なっている信号を探す．これを満たす信号についても，シミュレーションで調べることができる．ここで値が異なっていた信号を，追加する信号の候補 $A = (a_0, \cdots, a_n)$ とする．

以上により，LUT の入力に追加する信号の候補が絞られたら，その信号のうちの 1 つ a_0 を追加し，それによって入力 $X = (x_0, \cdots, x_n)$ に対して仕様と等価になる真理値表を作ることができるか否かを調べる．作ることができる場合，少なくともこれまでに調べた入力 X に対しては仕様を満たすことができているので，LUT の入力 a_0 を追加し，入力 X もそのままの状態で図 1 のフローへ戻る．そうでない場合，追加した入力 a によっては仕様を満たすことができないので，a_0 を候補から除外し，候補のうち次の信号 a_1 を追加して入力 X に対して真理値表を探索することを

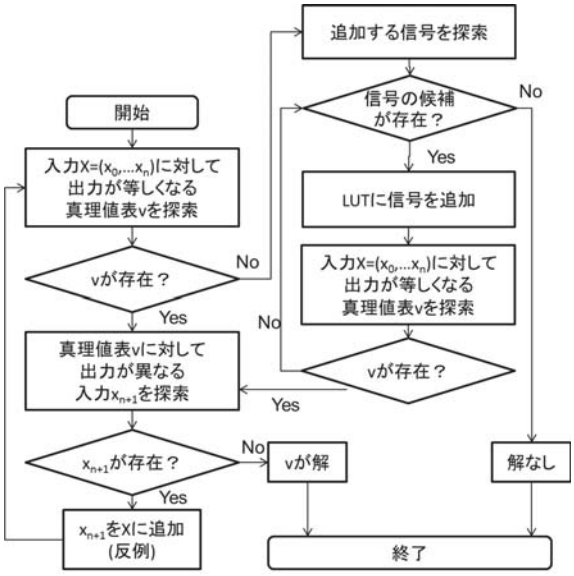

図 7 信号の追加を加えたフロー

再度行う．追加する信号の候補がなくなった場合には，現在の LUT の配置，信号の追加方法ではデバッグ不可能であるものとして，探索を終了する．

信号の追加を加えたフローは，図 7 となる．この手法を利用して絞り込んだ入力の複数の候補に対し，MUX を利用してデバッグに必要な信号を同時に探索することも可能である．

4. 実験

4.1 実験方法

例題を用いて，第 3.3 節および第 3.4 節の手法を用いてデバッグを行った．例題は産業用のもので，実際に行われた回路の修正をバグとしている．例題は Verilog で記述され，ゲート数が 2763 あり，そのうち 1 ヶ所にバグがある．図 1 および図 7 において，真理値表および反例の探索を行う SAT ソルバとしては，PicoSAT[7] を利用し，Verilog の SAT 式への変換には ABC[8] および AIGER[9] を用いた．

実験は，OS: Linux, CPU: Intel Core 2 Duo 3.33GHz, Memory: 4GB のマシン上で行った．

4.1.1 MUX を利用する手法

第 3.3 節の手法を利用したデバッグの実験を行う．実験にあたっては，まずバグの原因となっているゲートを LUT で置換する．本実験においては，あらかじめシミュレーションにより仕様と異なる出力を探し，その出力の logic cone のうち 5 ゲート遡ったゲートまでを LUT へ置換し，それぞれの入力に同じ MUX の出力を追加するという方法をとった．MUX の入力の候補としては primary input のすべてからランダムに選択した．

そして，MUX と LUT が挿入された回路に対し，図 1 のフローで真理値表と MUX の選択信号を探索する．解がない場合，MUX の入力を別の信号に入れ替え，もう一度フ

表 1　MUX を利用する手法の実験結果

MUX の入力数	回数	時間（変換/SAT ソルバ）[sec]
1（MUX なし）	(20)	timeout
16	15	5281(5175/106)
64	4	12794(12555/239)
256	1	211(209/2)

表 2　追加する信号の候補を絞り込む手法の実験結果

回数	時間 (変換/SAT ソルバ) [sec]
29	524(522/2)

ローを実行する．本実験においては，5 時間以内にデバッグが可能になるかを調べた．

比較として，MUX がなく，追加する入力を 1 つとした場合の実験も行った．

4.1.2　追加する信号の候補を絞り込む手法

第 3.4 節の手法を利用したデバッグの実験を行う．LUT へ置換するゲートは，本実験においてはバグの原因となっているゲートのみとした．追加する信号の候補は，回路の入力だけでなく，中間変数も含めた回路にあるすべての信号を対象とした．

図 7 のフローにおいて，解が見つからなかった場合に値に基づいた信号の探索を行うが，この実験では Icarus Verilog[10] を用いたシミュレーションにより値を調べている．

4.2　実験結果

第 3.3 節の手法を利用し，MUX を用いて解の探索を行った実験結果を表 1 に示す．また，第 3.4 節の手法を利用し，追加する信号の候補を絞り込み探索を行った実験の結果を表 2 に示す．

どちらの実験においても，解を得ることに成功している．表 1 において「回数」とは，解が見つからなかった場合に，MUX の入力を入れ替えた回数を指す．また表 2 の「回数」は，その信号で真理値表を見つけられなくなった場合に，信号を入れ替えた回数を指す．

表 1 の実験は，MUX を利用しない場合に 5 時間以内に終了しなかったのに対し，MUX を利用した場合に解を見つけることに成功している．また Verilog から SAT 式への変換に使われた時間が多く，SAT ソルバを利用する部分についてはそれほど時間がかかっていない．

MUX が 256 入力の場合に最も時間がかかっていないのは，1 回で修正に必要な信号を MUX の入力に選ぶことができたためであると考えられる．

表 2 においては，シミュレーションを含んだ時間を示している．表 1 のうち，MUX の入力数が 1，16，64 の場合に比べて時間はかかっていない．効率良く信号を選択することで，探索の時間が短くなっている．

5.　結論と今後の課題

本稿では，ゲートレベル回路のデバッグ手法として，ゲートを LUT に置換し，仕様を満たすための真理値表を探索することでデバッグを行う手法について説明した．さらに，それだけでは修正不可能なバグに対し，LUT への入力を追加することでデバッグを行う手法として，MUX を追加する手法と，信号値に注目して追加する信号を絞る手法を提案した．

今後の課題としては，シミュレーションではなく形式的な手法で条件を満たす信号を探し，追加する候補となる信号を探索することも考えられる．

また，1 つの信号を追加することで修正可能なバグというのは，取り扱えるバグの範囲としては，まだ限定的である．より広いバグを修正するためには，LUT などを用いて回路になかった論理関数を追加することも考えられるが，計算量も多くなるため，バグのモデルなどを用いて，探索の範囲を限定する必要があると考えている．

参考文献

[1] Yamashita, S., Yoshida, H. and Fujita, M.: Increasing Yield Using Partially-Programmable Circuits, *Proc. of Workshop on Synthesis And System Integration of Mixed Information technologies*, pp. 237–242 (2010).
[2] Jo, S., Matsumoto, T. and Fujita, M.: SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions, *Test Symposium (ATS), 2012 IEEE 21st Asian*, pp. 19–24 (2012).
[3] Janota, M. and Marques-Silva, J.: Abstraction-Based Algorithm for 2QBF, *Theory and Applications of Satisfiability Testing (SAT) 2011*, Lecture Notes in Computer Science, Vol. 6695, pp. 230–244 (2011).
[4] Janota, M., Klieber, W., Marques-Silva, J. and Clarke, E.: Solving QBF with Counterexample Guided Refinement, *Theory and Applications of Satisfiability Testing (SAT) 2012*, Lecture Notes in Computer Science, Vol. 7317, pp. 114–128 (2012).
[5] Fahim Ali, M., Veneris, A., Smith, A., Safarpour, S., Drechsler, R. and Abadir, M.: Debugging sequential circuits using Boolean satisfiability, *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pp. 204–209 (2004).
[6] 李在城，松本剛史，藤田昌宏: 論理関数の充足不可能性に注目した論理回路デバッグ手法の検討, 情報処理学会研究報告 SLDM, pp.1–6 (2012).
[7] Biere, A.: PicoSAT Essentials, *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, pp. 75–97 (2008).
[8] Brayton, R. and Mishchenko, A.: ABC: An Academic Industrial-Strength Verification Tool, *Computer Aided Verification*, Vol. 6174, pp. 24–40 (2010).
[9] AIGER, http://fmv.jku.at/aiger/.
[10] Icarus Verilog, http://iverilog.icarus.com/.