プログラム可能な素子を利用した大規模回路の自動修復手法

城　怜史[1,a)]　松本 剛史[2,b)]　藤田　昌宏[2,c)]

概要：現在、回路製造後のバグや仕様変更への対応として, プログラム可能な素子としてルックアップテーブル (LUT) を導入した回路が提案されている。回路の一部のゲートを LUT に置換し, 正しい回路と等価になるような LUT の真理値表を見つけることで, 回路製造後のバグや仕様変更に対応できることがある。LUT の真理値表を求めるのに非常に時間がかかってしまう為, 大規模回路への応用が困難であるという問題がある。近年 CEGAR に基づき SAT 問題を繰り返し解くことで QBF 問題を高速に解く手法が提案された。本稿では, その手法を応用することで LUT の真理値表を高速に求める手法を提案する。提案手法を用いた実験では, 通常の QBF ソルバを用いるよりも高速に LUT の真理値表を求めることができた。

キーワード：Partially-Programmable Circuit (PPC)、LUT, QBF 問題、SAT 問題

An Automatic Rectifying Method for Large-Scale Circuit with Programmable Devices

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

Abstract: Introducing partial programmability in circuits by replacing some gates with look up tables (LUTs) can be an effective way to improve post-silicon or in-field rectification and debugging. Although finding configurations of LUTs that can correct the circuits can be formulated as a QBF problem, solving it by state-of-the-art QBF solvers is still a hard problem for large circuits and many LUTs. In this paper, we present a rectification and debugging method for combinational circuits with LUTs by repeatedly applying Boolean SAT solvers. Through the experimental results, we show our proposed method can quickly find LUT configurations for large circuits with many LUTs, which cannot be solved by a QBF solver.

Keywords: Partially-Programmable Circuit (PPC), LUT, QBF, SAT

1. はじめに

近年、半導体技術の進歩により製造できる LSI の規模は飛躍的に向上している。そのため、設計誤りの無い完全に正しい回路を設計することは非常に困難になっている。また、LSI の複雑な製造プロセスにおいて回路中に配線の短絡や断線といった物理的な欠陥が生じることで、結果としてある割合で不良品が発生してしまう。加えて、回路の詳細が決定された設計終盤やさらには製造後に仕様が変更され、設計変更が必要となる場合がある。このようなバグや仕様変更に対して、プログラム不可能な回路において製造後に回路修正を行う場合、再設計・再製造 (re-spin) を行う必要が生じたり、ソフトウェアの大幅変更が必要となったりする為、莫大な費用と時間がかかってしまう。

この問題を解決するために製造後にプログラム可能な素子を導入した回路が提案されている。プログラム可能な素子を導入することでバグや仕様変更に対応することが出来るようになると、回路中の素子をプログラムしなおすことで回路修正をすることが出来るようになり、再設計・再製造するコストを抑えることが出来るようになると考えられ

[1] 東京大学大学院工学系研究科電気系工学専攻
Department of Electrical Engineering and Infomation Systems, School of Engineering, 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, University of Tokyo
[a)] jo@cad.t.u-tokyo.ac.jp
[b)] matsumoto@cad.t.u-tokyo.ac.jp
[c)] fujita@ee.t.u-tokyo.ac.jp

ⓒ 2013 Information Processing Society of Japan

る。そのような回路の 1 つとして、文献 [1] で提案されている Partially-Programmable Circuit(PPC) がある。PPC では、回路中の論理ゲートの一部を Look-Up Table(LUT) と呼ばれるメモリ素子で置き換えている。LUT の真理値表を変更することである結線が冗長となる時、その結線にいかなる故障が生じても LUT の真理値表を変更することで正常に動作させることが出来る。PPC では、これを利用してそのような故障に強い結線が増えるように、LUT を挿入して回路に冗長性をもたせている。

LUT の真理値表は以下の問題を解くことによって得られる。"ある LUT の真理値表が存在する時すべての入力パタンに対して正しい回路 (仕様) と出力が等価になるかどうか"
この問題を解くことで、回路が正しく動作するような LUT の真理値表を決定することができ、バグや仕様変更に対応することが出来る。文献 [2] では LUT の挿入により多くのバグや故障に対応できることが示されている。この問題は第 2 節にて後述する 2QBF 問題に定式化することができる。しかし、2QBF 問題を解くことが困難である為、大規模回路を扱うことができていない。

近年、この QBF 問題を高速に解く手法として CEGAR と呼ばれる手法を利用して 2 段階の SAT 問題として解く手法が提案された。そこで本稿では、この手法を応用して高速に LUT の真理値表を求める手法を提案し、その評価を行う。

本稿の構成は以下の通りである。まず第 2 節で QBF 問題や 2 段階の SAT 問題として QBF 問題を解く手法について述べ、第 3 節でその手法を用いて回路修復を行う提案手法について述べる。第 4 節では行った実験の結果を示し、第 5 節においてまとめを行う。

2. QBF 問題

2.1 QBF 問題

Quantified Boolean Formulas(QBF 問題) とは、以下のような全称限量子や存在限量子を含む論理関数が与えられた時、論理式 φ を満たす変数 z が存在するかどうか判定する問題である。

$$Q_1 z_1 ... Q_n z_n.\phi \quad (1)$$
$$Q_i \in \forall, \exists$$

論理式 ϕ は z_i 及び 1(true),0(false) からなる論理式である。ここで、$\exists x_1 ... \exists x_n$ や $\forall y_1 ... \forall y_n$ を $\exists X$ や $\forall Y$ と書く ($X = x_1 ... x_n$,$Y = y_1 ... y_n$)。このように変数をベクトルにまとめることで、以下のように、全称限量子 (∀) を 1 つにまとめることができる QBF 問題を 2QBF 問題という。これは全ての入力 Y に対して論理関数 ϕ の値を 1 にするような入力 X が存在するかどうかを判定する問題である。

$$\exists X \forall Y.\phi(X, Y) \quad (2)$$
$$X = (x_1, ... x_n), Y = (y_1, ... y_m)$$

QBF 問題は PSPACE 完全問題として知られており、NP 完全問題として知られる SAT 問題よりも解くのに時間がかかる。近年 QBF ソルバも大きく進歩しているが、SAT 問題と比べるとずっと小さい規模の問題しか扱うことができない [3]。

2.2 反例を用いた QBF 問題の解法

近年文献 [3] で提案された CEGAR に基づく 2QBF 問題を高速に解くアルゴリズムについて述べる。扱う状態数を減らす為に、問題やモデルを抽象化することがあるが、過度に抽象化してしまうことで、実際の問題やモデルには適さない反例が生じることがある。問題やモデルの抽象化のレベルを下げ、その反例が出ないように抽象化を行う。これを繰り返すことで、問題やモデルを適度に抽象化する手法を CounterExample Guided Abstraction Refinment(CEGAR) という。

CEGAR に基づくアルゴリズムでは、抽象化された問題と実際の問題の 2 つの問題を考える。この 2 つの問題の間には以下のような性質がある。

(1) 抽象化された問題が解を持たない時、実際の問題も同様に解を持たない
(2) 抽象化された問題の解が、実際の問題の解でない時、それを示す実際の問題における例を反例とよぶ
(3) 反例が存在しない時、その抽象化された問題の解は、実際の問題の解である

まず、抽象化された問題を解く。解が存在しない場合、性質 1 より、実際の問題も解を持たない。解を持つ場合、実際の問題の解であるかどうか調べる。反例が存在しない場合、性質 3 より、抽象化された問題の解が求める解である。一方、反例が存在する時、その反例が抽象化された問題の解とならないように抽象化を修正する。これを繰り返すことで実際の解を求める。このアルゴリズムを用いて 2QBF 問題を解くことができる。まず、2QBF 問題には以下のような性質がある。

- ν が $\exists X \forall Y.\phi$ の解となるのは、ν が式 (3) を満たす時だけである。この時、$\beta^{|Y|}$ は 0 または 1 からなる長さ $|Y|$ のベクトルの集合を指す。

$$\bigwedge_{\mu \in \beta^{|Y|}} \phi[Y/\mu] \quad (3)$$

- ν が $\exists X \forall Y.\phi$ の解となるのは、式 (4) を満たす Y が存在しない時だけである。

$$\neg\phi[X/\nu] \quad (4)$$

この性質を利用すると、CEGAR を用いた解法は以下のようになる。

ⓒ 2013 Information Processing Society of Japan

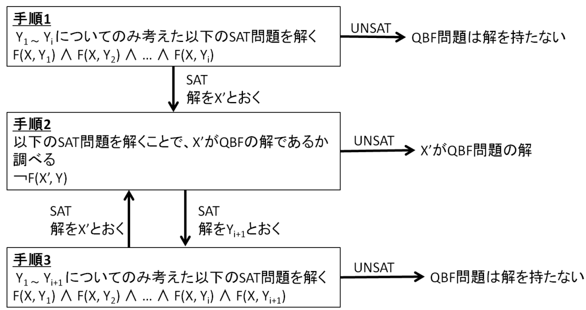

図 1 CEGAR を用いた 2QBF の解法

(1) 式 (5) は式 (6) のように抽象化することができる。抽象化された問題 (式 (6)) を SAT 問題として解くことで、解候補 ν を探す。

$$\exists X.\phi(X,Y_1) \wedge \phi(X,Y_2) \wedge ...\phi(X,Y_n) \tag{5}$$

$$\exists X.\phi(X,Y_1) \wedge ... \wedge \phi(X,Y_i) \tag{6}$$

$$Y_1, ...Y_i \in Y, i \leq n$$

(2) 抽象化された式 (6) の SAT 問題が解を持たない時、抽象化された問題が解を持たないので、実際の 2QBF 問題も解を持たない。式 (6) の SAT 問題が解を持つ時、得られた解 ν を X' と置く。次に式 (7) の SAT 問題を解くことで X' が QBF 問題の解であるかどうかを調べる。

$$\neg\phi[X/\nu] \Leftrightarrow \exists Y.\overline{\phi(X',Y)} \tag{7}$$

(3) 式 (7) の SAT 問題が解を持たない時、全ての Y において $\phi(X',Y) = 1$ となるので、X' が求める QBF 問題の解である。一方、式 (7) の SAT 問題が解を持つ時、得られた解を Y_{i+1} と置く。Y_{i+1} は X' が解であることの反例となる。式 (6) の SAT 問題に反例 Y_{i+1} を追加した式 (8) のの SAT 問題を解いて、式 (5) の解の候補を探す。

$$\exists X.\phi(X,Y_1) \wedge ... \wedge \phi(X,Y_i) \wedge \phi(X,Y_{i+1}) \tag{8}$$

(4) 解の候補が見つからなくなるか、反例が見つからなくなるまで、手順 1〜3 を繰り返す。解候補が見つからなかった場合、2QBF 問題は解を持たない。反例が見つからなかった場合、2QBF 問題の解はその時の解候補である。

以上をまとめると図 1 のようになる。

3. プログラム素子を利用した回路の自動修復

3.1 提案手法

図 2 のように入力パターンを Y、LUT の真理値表を X とするとプログラム可能素子を用いた回路修正は、"ある LUT の真理値表が存在する時すべての入力パタンに対して正しい回路 (仕様) と出力が等価になるかどうか"という問題と等価である。この問題は、$\exists X \forall Y \phi(X,Y)$ という 2QBF 問題に定式化することが出来る。前述の通りプログラム可能素子を用いた回路修正は重要なことだが、現状では QBF 問題を解くことの困難さゆえに大規模回路を扱うことはできない。そこで本節では、前述の 2QBF 問題の解法を応用して高速に LUT の真理値表を求める手法を提案する。手順は以下の通りである。

(1) 与えられた入力パタン $Y_1, ..., Y_i$ が全て正しい回路と等価な動作をするような LUT の真理値表 X=(x_1, … , x_m) を求める。ここで、x_j は j 番目の LUT の真理値表を表す。そのために、図 3 に示すように、入力パタンごとに正しい回路と LUT を挿入した回路の出力パタンが等価である場合に 1 を出力する回路を作り、これらの回路の全ての出力の積を出力する回路を作る。この回路の出力が 1 であれば、与えられた全ての入力パタンについて、LUT を挿入した回路は正しい回路と等価であると言える。そこで、図 3 の回路の出力が 1 となるような LUT の真理値表 X が存在するかどうかを SAT 問題として解く。解が存在する場合、得られた LUT の真理値表を X_1 とする。解が存在しない場合、どのように LUT の真理値表を変えても LUT が挿入された回路を修正することはできない。

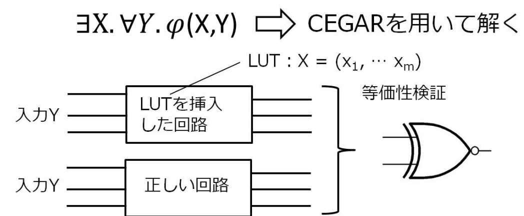

図 2 提案手法

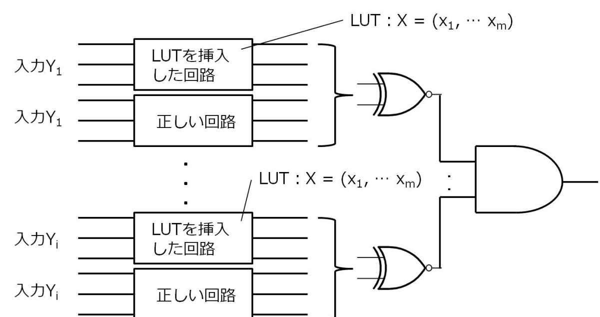

図 3 手順 1:LUT の真理値表 X_1 を求める

(2) 次に求めた LUT の真理値表が全ての入力パターンに対して正しい回路と等価な動作をするのか調べる。図 4 のように、入力パタンごとに正しい回路と LUT を挿入した回路の出力パタンが異なる時に 1 を出力する回路を作る。この回路の出力が 1 となれば、その時の入力パタンについては正しい回路と LUT を挿入した

ⓒ 2013 Information Processing Society of Japan

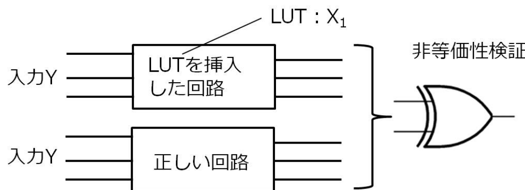

図 4 手順 2:出力が異なる入力 Y_{i+1} を求める

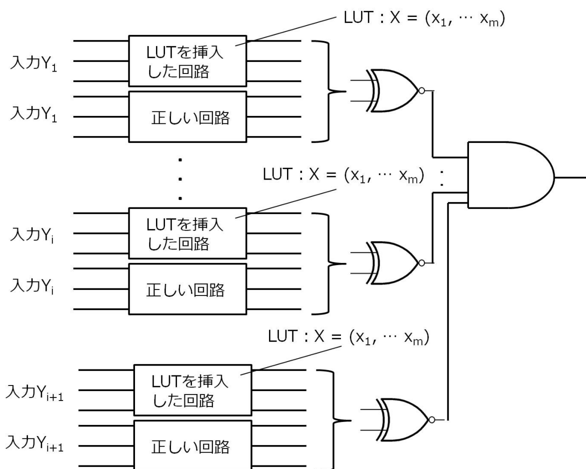

図 5 手順 3:抽象化修正後の LUT の真理値表を求める

回路の動作は異なるので、X_1 は求める LUT の真理値表とは異なる。図 4 の回路の出力が 1 となるような入力パタン Y が存在するかどうかを SAT 問題として解く。解が存在する時、得られた入力パタンを Y_{i+1} とする。Y_{i+1} は X_1 が解であることの反例となる。解が存在しない場合、全ての入力パタンに対して、正しい回路と LUT を挿入した回路の動作が等価なので、X_1 が求める LUT の真理値表となる。

(3) 手順 1 の入力パタンと手順 2 で求めた反例となる入力パタンを合わせた $Y_1, ..., Y_i, Y_{i+1}$ 全てに対して正しい回路と等価な動作をするような LUT の真理値表 X を求める。図 5 のように、図 2 の回路に入力パタン Y_{i+1} を加えた回路を考える。図 5 の回路の出力が 1 となるような LUT の真理値表 X が存在するかどうかを SAT 問題として解く。

(4) 与えられた入力パタンについて正しい回路と LUT を挿入した回路の動作が等価になる LUT の真理値表が見つからなくなるか、LUT の真理値表 X について正しい回路と LUT を挿入した回路の動作が異なる入力パタンが見つからなくなるまで手順 1～手順 3 を繰り返す。LUT の真理値表が見つからない場合は、どのような LUT の真理値表に対しても、LUT が挿入された回路を修正することはできない。入力パタンが見つからない場合はその時の LUT の真理値表が求める LUT の真理値表である。

以上の手法を用いることで、通常の QBF ソルバを用いて LUT の真理値表を求めるよりも高速に LUT の真理値表を求めることができると考えられる。

3.2 等価検証ツールの利用

前節の手順 2 において、SAT 問題を解くことで正しい回路と LUT が挿入された回路の動作が等価であるかどうか判定しているが、等価性検証ツールを用いて判定することも可能である。等価性検証とは、与えられた 2 つの回路が論理的に等価であることを検証することである。等価性検証は、設計を変更したときに誤りが生じていないか、設計を合成したときに合成ツールによってバグが生じていないか、などを検証する目的で変更前後や合成前後の設計記述に対して行われる。

内部等価点を利用して回路を分割して等価性を調べる等、等価性検証ツールは大規模回路への応用が進んでおり、組合せ回路の等価性検証、あるいはフリップフロップの対応の取れる 2 つの順序回路の等価性検証は、現在数百万～数千万ゲート規模でも対応が可能とされている。この等価性検証ツールを用いることで SAT ソルバを用いた前節の手法よりも大規模な回路が扱えるようになると考えられる。

4. 実験

4.1 実験方法

本節では、本研究で行った実験のフローを説明する。回路中のゲートを LUT に置き換えた回路が元の回路と同じ動作をするような LUT の真理値表を求めるという実験を行った。前節で述べた提案手法を用いて実験を行ったほか、通常の QBF ソルバを用いた実験も行ない、結果を比較した。実験環境は以下の通りである。

- 実験の中で用いたツールは、ABC(version abc70930[4])、AIGER(version aiger-1.9.4[5])。
- 提案手法の中で用いた SAT ソルバは、PicoSAT(version picosat-936[6])。比較に用いた QBF ソルバは sKizzo(version v0.8.2[7])。
- 実験環境: CPU Intel(R) Core(TM)2 Duo 3.33GHz、メモリ:4GB

提案手法で述べた回路は全て Verilog を用いて記述した。ABC[4] を用いて Verilig から blif 形式に変換し、AIGER[5] を用いて blif 形式から CNF 形式へと変換し、CNF 形式の論理式を SAT ソルバで解いた。

順序回路についてはフリップフロップで対応の取れる回路のみを扱った。図 6 のようにフリップフロップの入力と出力を入出力にもつ組み合わせ回路を作り、時間軸上で一定回数展開することで、組み合わせ回路として処理を行った。LUT は、時間軸展開を行う前にゲートと置き換えており、回路全体で見ると、同じ真理値表の LUT が時間軸展開数ずつ存在している。実験結果の表において、LUT の数を 10 個、20 個などと書いているが、これは、時間軸展開する前に置き換えた LUT の数である。また、フリップフ

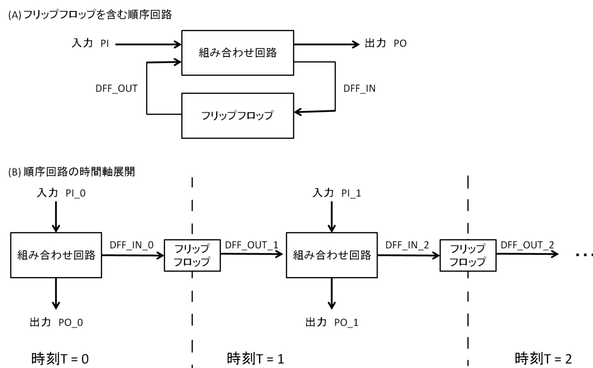

図 6 順序回路の時間軸展開

ロップ内の変数の初期値は全て 0 として実験を行った。

回路は、ISCAS '85 benchmark circuits から組み合わせ回路として c499, c880, c1355, c1908, c2670, c3540, c5315, c6288 の 8 個、順序回路として s9234, s15850, s38584 の 3 個を用いた。また、OpenCores[8] の OpenRISC 1200 を用いた実験も行った。各ベンチマークについて LUT を 10 個、20 個、50 個、100 個挿入し実験を行った。LUT に置き換えるゲートは全てランダムに選んだ。第 3.1 節で述べた提案手法の手順 1 における初期入力 Y は、$Y_1 = (0, 0, 0, 0...), Y_2 = (1, 1, 1, 1...)$ の 2 つとした。各ベンチマークにつき 20 回ずつ実験を行った。組み合わせ回路のベンチマークについては QBF ソルバを用いた実験も行った。その際 LUT に置き換えるゲートは提案手法による実験と同じものを用いた。

4.2 等価性検証ツールの利用

第 3.2 節で述べたように提案手法の手順 2 における SAT 問題は組み合わせ回路と正しい回路と LUT が挿入された回路の動作が等価であるかどうか判定しているが、SAT ソルバの代わりに等価性検証ツールを用いて判定することも可能である。等価性検証ツールは大規模回路への応用が進んでおり、SAT ソルバを用いて等価性を判断するよりも高速になると考えられる。そこで、回路規模の大きい Open RISC 1200 について等価性検証ツールを用いた実験を行った。本研究では、等価性検証ツールとして Synopsys 社の Formality[9] を利用した。Formality に元の回路と LUT を挿入した回路の Verilog を与えて等価であるかどうかを調べた。実験環境は前節の実験と同じ環境を利用した。

4.3 実験結果

第 4.1 節で述べた実験の結果は、組み合わせ回路については表 1 のように、順序回路の結果は表 2、表 3 のようになった。サイクル数とは第 3.1 節で述べた提案手法の手順 1〜3 を何回繰り返したかを指す。また、平均時間は、一定時間内に解けた問題を解くのにかかった平均時間を指す。組み合わせ回路については 1 時間以内、順序回路については 5 時間以内に解けた問題でのみ平均時間を出しており、これらの時間以内に解けなかった問題については平均時間に反映されていない。

次に表 2、表 3 を見ると、OpenRISC を 3 回以上時間軸展開して LUT を 50 個以上挿入した時、回路規模や LUT の数が大きくなると、提案手法を用いても 5 時間以内に LUT の真理値表を決めることが出来ていないことが分かる。

表 3 を見ると、SAT ソルバを用いた時には解くことの出来なかった問題も等価性検証ツールを用いることで解くことが出来るようになっていることが分かる。一方で回路規模が小さい時は、SAT ソルバを用いた方が早く解くことができている。これは、等価性検証ツールの起動や終了、回路の読み込み等に時間がかかり、オーバーヘッドとなっている為だと思われる。1 サイクルごとに等価性検証ツールにかかる時間は、時間軸展開が 2 回の時は 30 秒程度、3 回の時は 50 秒程度、4 回の時は 80 秒程度であり、SAT 問題を解く時間や ABC の処理にかかる時間と比べると、回路規模に対する実行時間の増え方が少ないことがわかる。このことから回路規模が更に大きくなると、等価性検証ツールを用いた方法による実行時間と SAT ソルバを用いた方法と実行時間の差は更に広がっていくと思われる。

5. 結論と今後の課題

本研究では、プログラム可能な素子として LUT を回路に挿入し、LUT を挿入する前の回路と等価になるような LUT の真理値表を求める効率的な手法を提案し、その評価を行った。

LSI 製造後に回路にバグや仕様変更が生じた場合、回路の再設計・再製造等が必要となり莫大なコストがかかってしまう。この問題への対応として回路に LUT を挿入した PPC と呼ばれる回路が提案されているが、LUT の真理値表を求めることが難しく、大規模回路を扱うことが難しかった。LUT の真理値表を決定するにあたり、通常の QBF ソルバではなく、CEGAR を用いたアルゴリズムを利用することで、LUT の真理値表を高速に求めることを可能にした。また、等価性検証ツールは回路規模が大きくなると SAT 問題を解くよりも高速に等価性を判定出来る為、大きい回路を扱う時は、等価性検証ツールを用いた手法の方が優位性があることがわかった。

本研究では、プログラム可能な素子として回路に LUT を挿入して実験を行った。文献 [10] では LUT とは異なるプログラム可能素子が提案されている。今後はこのような LUT 以外のプログラム可能素子の導入について検討したい。また、QBF 問題は、形式的検証やモデル検査等様々な応用が考えられる。CEGAR を用いて QBF 問題を高速に解くアルゴリズムは、本研究で提案した回路修正以外にも応用することが出来ると考えられる。今後は形式的検証やモデル検査等への応用についても検討していきたい。

ⓒ 2013 Information Processing Society of Japan

参考文献

[1] S. Yamashita, H. Yoshida, and M. Fujita, "Increasing Yield Using Partially-Programmable Circuits," *Proc. of The 17th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2010)*, pp.237–242, 2010.

[2] Hratch Mangassarian, Hiroaki Yoshida, Andreas G. Veneris, Shigeru Yamashita, and Masahiro Fujita, "On error tolerance and Engineering Change with Partially Programmable Circuits," *Proc. of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC 2012)*, pp.695–700, 2012.

[3] Mikolas Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke, "Solving QBF with Counterexample Guided Refinement," *Proc. of The 15th International Conference on Theory and Applications of Satisfiability Testing (SAT '12)*, pp.114–128, June 2012.

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

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

[6] A. Biere: "PicoSAT Essentials," *Journal on Satisfiability Boolean Modeling and Computation (JSAT)*, Vol.4, pp.75–97, 2008.

[7] sKizzo – a QBF solver: *http://skizzo.info/*.

[8] OpenCores Homepage, *http:http://opencores.org/*

[9] Synopsus Formality Homepage, *http://www.synopsys.com*

[10] Yasuhiro Okamoto, Yoshihiro Ichinomiya, Motoki Amagasaki, Masahiro Iida, Toshinori Sueyoshi, "A Configuration Memory Reduced Reconfigurable Logic Cell Architecture for Area Minimization," *Proc. of International Conference on Field Programmable Logic and Applications (FPL 2010)*, pp.304–309,2010.

表 1 組み合わせ回路の実験結果

ベンチマーク	ゲート数	LUT	提案手法			sKizzo	
			解けた数 (in 3,600 sec)	実行時間 (sec)	サイクル数	解けた数 (in 3,600 sec)	実行時間 (sec)
c499	202	10	20/20	0.7	3.15	20/20	7.7
		20	20/20	1.2	6.45	20/20	18.25
		50	20/20	3.35	16.75	17/20	279.2
		100	20/20	7.3	16.75	6/20	880.8
c880	383	10	20/20	3.2	15.3	18/20	594.2
		20	20/20	7.4	28.3	13/20	41.8
		50	20/20	27.9	65.5	17/20	154.8
		100	20/20	99.4	123.0	17/20	183.1
c1350	546	10	20/20	4.6	18	2/20	1735.5
		20	20/20	10.75	32.3	0/20	Time out
		50	20/20	26.7	53.9	0/20	Time out
		100	20/20	72.75	89.35	0/20	Time out
c1908	880	10	20/20	4.7	15.9	14/20	655
		20	20/20	10.35	27.4	0/20	Time out
		50	20/20	26.1	47.15	0/20	Time out
		100	20/20	72.85	79.1	0/20	Time out
c2670	1193	10	20/20	5.5	11.0	20/20	1.8
		20	20/20	14.7	21.0	14/20	25.9
		50	20/20	65.2	48.6	14/20	835.8
		100	20/20	207.6	87.4	10/20	1899.2
c3540	1669	10	20/20	4.85	10.85	8/20	1239
		20	20/20	12.15	21.2	0/20	Time out
		50	20/20	53.3	49.2	0/20	Time out
		100	20/20	205.45	92.3	0/20	Time out
c5315	2406	10	20/20	7.35	12.95	0/20	Time out
		20	20/20	19.3	23.7	0/20	Time out
		50	20/20	79.45	52.85	0/20	Time out
		100	20/20	236.75	52.85	0/20	Time out
c6288	2406	10	20/20	6.2	6.1	0/20	Time out
		20	20/20	15.5	10.6	0/20	Time out
		50	18/20	589.8	24.4	0/20	Time out
		100	15/20	586.1	41.4	0/20	Time out

表 2 提案手法による順序回路の実験結果

ベンチマーク	時間展開数	ゲート数	LUT	解けた数 (in 18,000 sec)	実行時間 time (sec)	サイクル数
s9234	3	6081	10	20/20	1.25	1.1
			20	20/20	2.3	2.15
			50	20/20	4.35	3.8
			100	20/20	8.45	6.3
	5	10135	10	20/20	3.4	1.8
			20	20/20	5.3	2.75
			50	20/20	14.85	6.3
			100	20/20	39.7	11.85
	10	20270	10	20/20	13.2	3
			20	20/20	23.55	5.1
			50	20/20	79.75	11.5
			100	20/20	221.35	20.55
s15850	3	10344	10	20/20	5.1	2.05
			20	20/20	7.15	2.95
			50	20/20	21.75	6.95
			100	20/20	64.7	14.05
	5	17240	10	20/20	11.95	2.5
			20	20/20	27.4	5.2
			50	20/20	105.25	12.8
			100	20/20	363.25	25.65
	10	34480	10	20/20	59.75	4.65
			20	20/20	186.3	10.4
			50	20/20	679.7	22.6
			100	20/20	4830.5	39.5
s38584	3	34344	10	20/20	5.6	0.65
			20	20/20	7.7	1.15
			50	20/20	14.3	2.45
			100	20/20	20.05	3.4
	5	57240	10	20/20	17.05	1.4
			20	20/20	34.05	2.9
			50	20/20	112.5	7.4
			100	20/20	259.55	13
	10	114480	10	20/20	289.95	7.15
			20	20/20	738.4	13.5
			50	20/20	7686	23
			100	8/20	4830.5	39.5

表 3 OpenRISC1200 の実験結果

時間展開数	ゲート数	LUT	提案手法			等価性検証ツールを利用		
			解けた数 (in 18,000 sec)	実行時間 (sec)	サイクル数	解けた数 (in 18,000 sec)	実行時間 (sec)	サイクル数
2	35402	10	20/20	31	5.45	20/20	1.69.1	5.4
		20	20/20	61.5	8.95	20/20	277.5	8.95
		50	20/20	188.15	18.2	20/20	639	19.7
		100	20/20	499.45	31.7	20/20	1153.25	35.3
3	53103	10	20/20	144.8	11.45	20/20	646.65	12.2
		20	20/20	387.1	21	20/20	1114.05	20
		50	20/20	1713.95	48.1	20/20	2820.35	47.3
		100	12/20	10253.75	84.5	20/20	5668.8	87.65
4	70804	10	20/20	1310	16.9	20/20	1685.4	18
		20	20/20	1945	35	20/20	1685.4	35
		50	1/20	17588	64	20/20	9094.35	81.95
		100	0/20	Timeout	Timeout	10/20	12905.3	108.3

ⓒ 2013 Information Processing Society of Japan