Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

Inductive Invariantでの効率的フリップフロップ選択による形式的検証とその応用

小池良吾[a)]　藤田昌宏

概要：近年では論理回路の設計が大規模化し、設計を数学的に検証する形式的検証技術はより重要な技術になっている。その形式的検証技術の 1 つに順序回路の到達可能性を解析する Inductive Invariant がある。Inductive Invariant では特定のフリップフロップの集合に対して、順序回路が到達可能状態を計算することができる。Inductive Invariant では回路が持つすべてのフリップフロップからその一部を対象として計算を行うことで得られる状態は到達可能状態の上位集合となる。このようにすることで真の到達可能状態を計算するよりも計算量を抑えて解析を行うことが可能である。ここで計算の対象とするフリップフロップによって計算される到達可能状態の上位集合の大きさが変化する。本論文ではこのフリップフロップの選択方法を効率的に探索するアルゴリズムを提案し、またその計算量を削減する方法を提案した。提案手法はまずフリップフロップを選択する問題を QBF 問題として定式化し、それをインクリメンタルに解くことによって効率的な探索を可能にした。また Inductive Invariant の応用として計算された到達不可能な状態を利用して回路を最適化する方法を提案した。いずれの手法に関しても実験によって有効性を確認した。

1. はじめに

近年の VLSI 技術の発展に伴い、大規模で複雑な回路の開発が可能になってきている。しかし設計するシステムが複雑になるに応じて、その設計段階でバグを含む危険性も高まってしまう。ハードウェアは製造にコストと時間を多く必要とするため、設計上バグが存在すると多大な悪影響を及ぼしかねない。そのため検証の重要性は設計手法そのものと比べても増加している。

順序回路の形式的検証の 1 つは到達可能性の解析である。順序回路で到達可能な状態 (Reachable 状態) は全状態に対して極めて少ないことが一般的であり、そのような状態を求めることは検証だけでなく回路最適化にも応用できる。ただし順序回路において Reachable 状態そのものを計算することは計算量の観点から難しく、その上位集合 (スーパーセット) を計算することが有用である。[1], [2], [3] などで Reachable 状態のスーパーセットが用いられている。

Inductive Invariant とは Reachable 状態のスーパーセットを計算する手法の 1 つである [4], [5], [6]。Inductive Invariant では回路が持つフリップフロップ (FlipFlop,FF) の一部を対象として、Reachable 状態のスーパーセットを QBF 問題として定式化する。この計算では対象とする FF の選び方によって計算されるスーパーセットの大きさが変化する。適切な FF を選ぶことができれば少ない計算量で小さなスーパーセット、つまり真の Reachable 状態に近い集合を計算することができる。そのため FF の選び方が重要である。[7], [8] などでは Inductive Invariant における FF の選び方が提案されている。これらの手法はある選ばれた FF の選び方 1 パターンを対象として Inductive Invariant を計算するものである。本論文では FF の選び方を網羅的に探索する問題をインクリメンタルな QBF 問題として定式化する。それを解くことによって複数の FF の選び方の中でも適切な選び方を求めることができる。またこの手法では複数のインクリメントされた QBF 問題を解くことになるが、1 つ前の問題の結果を利用して次の QBF 問題の計算量を削減することが可能である。

また Inductive Invariant によって計算された Unreachable 状態を利用して回路を最適化する方法についても提案する。Unreachable 状態は通常の動作では決して実現しないため、その状態での回路の出力はどのような値でもよくドントケアとすることができる。順序回路の最適化は一般的にその組み合わせ回路部分にのみ適用されるが、提案手法は回路を順序回路として最適化可能である。

2. Inductive Invariant

Inductive Invariant とは順序回路の Reachable 状態のスーパーセットを計算する手法の 1 つである [4],[5],[6]。Induc-

a) koike@cad.t.u-tokyo.ac.jp

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

tive Invariant の計算では状態変数である FF を入力とするような論理関数 F を考える。関数 F は現在の状態を入力とするものと、次の状態を入力とするものの 2 つを利用する。これら 2 つの論理関数 F は同じものである。

ここでこの論理関数 F が「現在の状態で真ならば次の状態でも真」という条件を満たすとする。このような論理関数 F は Inductive Invariant と呼ばれる。F は「現在の状態で真ならば次の状態でも真」であるため以下の条件を満たすことになる。

$$F(ps) \Rightarrow F(ns) \tag{1}$$

ここで ps, ns はそれぞれ現在の状態と次の状態である。すべての状態で上式が成り立つような F を考えると以下のようになる。

$$\exists F \quad \forall ps, ns \quad F(ps) \Rightarrow F(ns) \tag{2}$$

$$\Leftrightarrow \exists F \quad \forall ps, ns \quad \overline{F(ps)} \vee F(ns) \tag{3}$$

これを回路として表現すると図 1 のようになる。この計算では初期状態から始まって次の状態でも真であるということを繰り返して Reachable 状態をすべて論理関数 F に含むことになる。つまり F が 1 となるような状態は Reachable 状態のスーパーセットに含まれ、0 となるような状態はスーパーセットに含まれない。ここで F としてどのような入力に対しても常に出力が 1 となるような論理関数 (定数 1 関数) を考えると、$F(ns) = 1$ が常に成立するためこの関数は条件を満たすことになる。しかし任意の入力で出力が 1 であるということは、すべての状態が Reachable 状態であるということと同値であり Reachable 状態を計算する意味がない。よってそのような定数 1 関数を除外する以下のような制約を設ける。

$$F \neq \text{const } 1 \tag{4}$$

実際の順序回路でもすべての状態が Reachable なものも存在するが例外的でありここでは扱わない。順序回路では全状態数の中で Reachable 状態は極めて少ないのが一般的である。また出力が常に 0 となるような関数 F も常に条件を

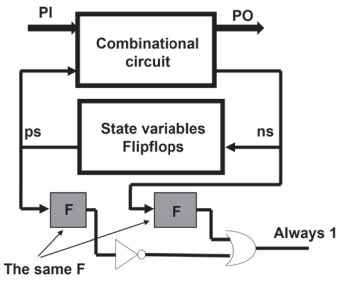

図 1 QBF 問題として定式化された Inductive Invarinat の計算

満たすことになるが、初期状態は常に Reachable であることを考えるとそのような関数 F は存在しない。初期状態が常に Reachable であることを考えると、以下の条件も満たされている。is は初期状態である。

$$F(is) = 1 \tag{5}$$

また本論文では初期状態 is はすべての FF の値が 0 である状態を考えることにする。それ以外の初期状態も解析手法を修正することで同様に扱うことができるが本稿では扱わない。

またここでは関数 F が LUT(Look Up Table) で表現されているとする。LUT とは任意の真理値表の値をそのまま関数として実装できるプログラム可能素子である。式 (3) で関数 F を LUT と考える以下のようになる。

$$\exists LUT \quad \forall ps, ns \quad \overline{LUT}(ps) \vee LUT(ns) \tag{6}$$

このようにして Inductive Invariant は式 (6) を解くことで見つけることができる。ここで m 入力の LUT は 2^m 個の変数を持つため、入力が多い場合は変数の数が指数爆発して問題を解くことができない。現状では LUT への入力が 15~20 程度を超えると計算が終わらないことがわかっている。しかし現在の一般的な順序回路は数千以上の FF を持つことも珍しくなく、そのような入力数を持つ LUT に対して QBF 問題を解くことは現実的でない。そこで回路の持つすべての FF を LUT の入力とせずに FF の一部 (FF 全体の集合のサブセット) を LUT の入力とすることを考える。回路が全体で m 個のフリップフロップ $S_m = \{s_0, s_1, \cdots, s_{m-1}\}$ を持つとする。そこから n 個の FF のサブセット $S'_n = \{s'_0, s'_1, \cdots, s'_{n-1}\} \subset S_m$ を選択し LUT の入力とする。以降でも大文字の S は FF 全体を表し、サブセットには $'$ をつける。またその要素はそれぞれ小文字で表す。つまり s'_i は i 番目の入力として選択されている FF である。次にサブセットの選び方を $SelectFF$ とすると以下のようになる。

$$S'_n = SelectFF(S_m) \tag{7}$$

FF のサブセットによって ps, ns の一部である $ps'(S'_n), ns'(S'_n)$ が決定される。

これによって以下のような FF のサブセットを用いた定式化を行うことができる。

$$\exists SelectFF \quad \exists LUT \quad \forall ps', ns' \quad \overline{LUT}(ps') \vee LUT(ns') \tag{8}$$

このようにして FF のサブセットの選び方 $SelectFF$ が決定すれば、それを用いて Inductive invariant を計算し Reachable 状態のスーパーセットを得ることができる。もしあるサブセットに対して計算された LUT が 1 定数関数であったり見つかった Unreachable 状態の数が少なかった場合は、サブセットの選び方 $SelectFF$ を変えるか、サブセットの大きさを変えて (n を変化させて) 再度計算を行うことにな

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

る。Inductive invariant の計算では見つかった Unreachable 状態の数が多ければ多いほど計算時間も短くなるため、適切な FF のサブセットを選んで計算を行うことが計算時間の面でも検証という観点でも重要である。

3. インクリメンタルなフリップフロップ選択を利用した効率的探索

Inductive Invariant で入力とする FF の選び方は [7], [8] などで提案されている。しかしそれらの手法ではある FF のサブセット S'_n を入力として計算を行い、問題が解けなければ入力数を増やすか別の方法でサブセットを選び直すトライアンドエラーが必要になる。ここでは FF の選択方法を QBF 問題で網羅的に探索する方法を提案する。選び方を全探索すると m 個の FF から n 個の FF を選ぶパターンは ${}_mC_n$ 通りであり、n が極めて小さい範囲以外では実用的ではない。そのためサブセット全体を探索するのではなく、ヒューリスティックを用いて状態数を削減しながら探索することを提案する。ここからは回路全体が m 個のフリップフロップ $S_m = \{s_0, s_1, \cdots, s_m\}$ を持ち、そこから LUT の入力となる n 個のフリップフロップ $S'_n = \{s'_0, s'_1, \cdots, s'_n\}$ $(s'_i \in S_m)$ を選ぶことを考える。提案アルゴリズムは大きく分けて以下のような 2 ステップで構成される。

(1) 一番最初のサブセット S'_{init} を求める問題を解く。この時 S'_{init} は要素数が十分小さい FF の集合である。
(2) そのサブセット S'_{init} に 1 つ FF を追加する QBF 問題を解く。これを繰り返して n 個のフリップフロップ S'_n を構成し、LUT の入力とする。

次章からこの 2 ステップに関して詳細な説明を行う。

3.1 最初のフリップフロップのサブセットの求め方

ここではインクリメンタルに FF を追加していく一番初めのサブセット S'_{init} を見つける方法を説明する。一番初めの解を見つける際にはサブセットの要素数が小さい場合を想定し、その場合 MUX を用いて FF のサブセットを網羅的に探索することが可能である。ある k で MUX を用いて網羅的に要素数 k のサブセットを探索する問題は以下のように定式化できる。

$$\exists ct_k\ \exists LUT_k\ \forall ps'(S'_k), ns'(S'_k)$$
$$\overline{LUT}(ps') \vee (LUT(ns')) \wedge MUX(S_m, ct_k) = S'_k \quad (9)$$

ここで ct_k は MUX の制御信号で、S'_k が選択された FF のサブセットである。この式は制御信号 ct_k を変化させることで選択する k 個の FF を網羅的に探索する問題である。ここでの MUX は k 個の FF を選ぶため k 出力の MUX である。また上述のように k が小さい範囲以外でこの式は計算量的に解くことができないことに注意する。$k = 1$ から始めてこの問題が SAT になるまで k を増加させて網羅的に探索を行う QBF 問題を解く。$k = i_{init}$ のときに問題が SAT になったとしてその QBF 問題を QBF_{init} と書く。その時の FF のサブセットを S'_{init} とする。またこの問題がある k で UNSAT になるのは「どのような k 個の FF のサブセットを用いても定数 1 以外の LUT を計算できない」時である。そのため k が小さい範囲でも十分に解を見つけることができる。この QBF_{init} により一番小さなサブセット S'_{init} を構成することができる。

3.2 インクリメンタルなフリップフロップ選択

次にある FF のサブセット S'_k が与えられた時、そこに FF を追加して S'_{k+1} を構成する方法について説明する。3.1 章で最初のサブセット S'_{init} を求められるため、そこから 1 つずつ FF 数 k を大きくしていくことで求めたいサイズ n まで計算するというものである。ここで選ばれるサブセットは 1 つ小さいサイズのときに選ばれていたサブセットに 1 つ追加で FF を追加して構成する。

$$S'_{k+1} = S'_k \vee \{s'_{k+1}\} \quad (10)$$

s'_{k+1} は新たに追加される FF であり、今まで選ばれていない FF の中から 1 つ選ばれる。つまり MUX を用いて以下のように選択される。

$$s'_{k+1} = MUX(\overline{S'_k},\ ct) \quad (11)$$

$\overline{S'_k}$ は S'_k の補集合であり、S'_k で選択されていない FF 全体である。このようにして S'_{k+1} を S'_k から構成することを繰り返して S'_n を構成する。ある k に対して MUX は単一の FF を選択すればいいため、1 出力 MUX を用いることができる。そして探索するのは選ばれていない FF の数 $m - k$ から 1 つであるため、そのパターン数は ${}_{m-k}C_1 = m - k$ であり、組合せ爆発を起こさない。ある k で S'_k が LUT の入力とされているときに、新しく $k+1$ 個目に追加する FF を選択する QBF 問題は以下のようになる。

$$\exists ct_{k+1}\ \exists LUT_{k+1}\ \forall ps'(S'_{k+1}), ns'(S'_{k+1}))$$
$$\overline{LUT}(ps') \vee (LUT(ns')) \wedge s'_{k+1} = MUX_1(\overline{S'_k},\ ct_{k+1}) \quad (12)$$

この QBF 問題を k について区別するため QBF_{k+1} と呼ぶ。また MUX が 1 出力であることを明確にするため 1 出力 MUX は MUX_1 と書く。また ct_k, LUT_k はそれぞれ QBF_k での MUX の選択信号、その選択信号によって選ばれた FF のサブセットを用いて計算される LUT である。QBF_k の解をまとめて Sol_k と書く。$Sol_k = \{ct_k, LUT_k\}$ である。この問題を解くことで制御信号 ct によって新たに追加される 1 つの FF が選択されることになる。またこのような選択手法を用いても FF のサブセット全体を探索できるわけ

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

ではないことに注意する。探索できるのは S'_k で選択されている FF に 1 つの FF を追加して構成できるサブセットのみである。最終的には S'_n を求めたいため、$k = n$ まで k を変化させながら複数回 QBF$_k$ を解くことになる。

次にこのようにサブセットを構成する際に 1 つ前の QBF の結果を利用して QBF 問題の計算量を削減する方法について説明する。QBF$_k$ の結果から QBF$_{k+1}$ の変数を削減する手法である。まずある FF のサブセット S'_k, S'_{k+1} を用いて計算された LUT の関係性について説明する。これは [7] で述べられている Inductive Invariant において変数の数を削減する手法と同様の考え方である。LUT_k の i 行目の出力を out^i_k と書く。また LUT の入力 in_i は選択された FF のサブセット S'_k であるため $in_{k-1} \cdots in_1 in_0 = s'_{k-1} \cdots s'_1 s'_0$ である。ここでもしある i に対して $out^i_k = 0$ であったとする。このとき以下のように対応する状態 $in_{k-1} \cdots in_1 in_0 = s'_{k-1} \cdots s'_1 s'_0 = i$ は到達不可能である。そこで S'_k に対して 1 つ FF を追加した S'_{k+1} について考えると、まず LUT の入力は 1 つ FF が追加されて $in_k in_{(k-1)} \cdots in_1 in_0 = s'_k s'_{k-1} \cdots s'_1 s'_0$ となる。そして下式が成り立つ。

$$s'_k s'_{k-1} \cdots s'_1 s'_0 = i \tag{13}$$

$$\Leftrightarrow\ s'_{k-1} \cdots s'_1 s'_0 = i \wedge s'_k = 0 \tag{14}$$

式 (14) の状態は左側の条件 $s'_{k-1} \cdots s'_1 s'_0 = i$ が到達不可能であるため、式 (13) の状態が到達不可能であることがわかる。つまり任意の s'_{k+1} に対して $out^i_{k+1} = 0$ である。任意の s'_{k+1} に対して $out^{i+2^k}_{k+1} = 0$ である。これは式 (11) のように S'_k に対して追加で 1 つ FF を追加している場合にのみ成り立つものである。つまり $S'_k \subset S'_{k+1}$ であることから成り立つものである。そのため残りの未確定の変数のみを新たに計算すれば良い。これを繰り返すことで、k をインクリメントするたびに Unreahcable であると確定している変数を削減していくことができる。このことより見つかる Unreachable 状態の数に関しても以下のようなことがわかる。LUT_k における出力の 1 の数、つまり out^i_k のうち 1 であるものの数を $N(LUT_k, 1)$ と書き、0 の数を $N(LUT_k, 0)$ と書く。

$$N(LUT_{k+1}, 0) \geq 2N(LUT_k, 0) \tag{15}$$

$$N(LUT_{k+1}, 1) \leq 2N(LUT_k, 1) \tag{16}$$

このことから式 (11) による FF の選択では任意の FF が問題の解になることがわかる。見つかっている LUT_k が定数 1 関数でないならば、任意の FF を追加しても定数 1 関数以外の LUT が見つかるからである。つまり式 (12) では任意の ct_k が QBF 問題の解になってしまう。しかし FF を追加したことによって新たに Reachable 状態のスーパーセットが小さくならなければ (Unreachable 状態が多く見つからなければ) 意味がない。つまり式 (12) に以下のような制約を加える必要がある。

$$N(LUT_{k+1}, 0) > 2N(LUT_k, 0) \tag{17}$$

$$\Leftrightarrow N(LUT_{k+1}, 0) \neq 2N(LUT_k, 0) \tag{18}$$

ここまでをまとめてある k のときに FF のサブセット S_k が選択されているときに、$k+1$ で新しく FF を選択する問題 QBF$_{k+1}$ は以下のようになる。

$$\begin{aligned}&\exists ct_{k+1}\ \exists LUT_{k+1}\ \forall ps'(S'_{k+1}), ns'(S'_{k+1}))\\ &\left[\ \overline{LUT}(ps') \vee (LUT(ns'))\right] \wedge \left[s'_{k+1} = MUX_1(\overline{S'_k},\ ct_{k+1})\right]\\ &\qquad \wedge \left[N(LUT_{k+1}, 0) \neq 2N(LUT_k, 0)\right]\end{aligned} \tag{19}$$

この QBF 問題が SAT であった時見つかった解は追加した制約から LUT_k よりも多くの Unreachable 状態を含むことになる。逆にこの問題が UNSAT になった場合はどのような FF を追加しても LUT_k よりも多くの Unreachable 状態を含む LUT_{k+1} を構成することはできない。この場合はランダムに 1 つの FF を追加することにする。つまり QBF$_k$ が SAT であればより良い解が見つかり、UNSAT であればランダムに FF が追加される。

3.3 インクリメンタル選択時の全解探索と解の順位付け

ここまでで FF をインクリメンタルに MUX によって選択する手法を式 (19) のように定式化した。しかし式 (19) の解は 1 つとは限らない。そのような解全体の中で適切なものを選んでいくことで、最終的に S'_n で見つかる Unreachable 状態が多くなるはずである。同様に式 (9) にも解が複数存在することが一般的であり同様に最良の解を選ぶ必要がある。QBF$_k$ の解の中で i 番目に見つかった解を Sol^i_k とする。QBF 問題の解をすべて、もしくは複数求める場合にはすでに見つけた解以外の解を探すという条件を加えて再度 QBF 問題を解けば良い。式 (19) の解をすべて見つける場合には、i 個目の解を探索しているとして以下の条件を加えれば良い。

$$Sol^i_k \neq Sol^j_k \quad (1 \leq j < i) \tag{20}$$

これにより見つかる新しい解は今までに見つかった解と異なるものになる。

次に全ての解 Sol^i_k の中で最良なものを選ぶために複数の解にスコアをつける方法について説明する。解ごとにスコアをつけ、そのスコアが最も高いものを解として採用することにする。見つかった解 (ct^i_k によって選択される S'^i_k と、そのサブセットによって計算される LUT^i_k) の中でより良い解であることの条件は以下の 2 つであると考えることができる。

(1) その FF を選んだ際に新たに見つかる Unreachable 状態の数が多い。

(2) その FF がこれまでの解全体に含まれる回数が多い。

これら 2 つの条件を考えて、スコアの付け方として以下のようなもの提案する。まず FF ごとのスコアを計算し、それにより解ごとのスコアを計算する。あるフリップフロップ s に対するスコアを $score(s)$ と書く。またある解 Sol に対するスコアを $score(Sol)$ と書く。

まず最初の問題、QBF_{init} を解く前に任意の $s \in S_m$ のスコアを $score(s) = 0$ と初期化する。次にある k に対して QBF_k の解であるサブセット S'_k と LUT_k が見つかったとする。ここでそのサブセットに含まれる $s' \in S'_k$ にスコア $N(LUT_k, 0)/2^k$ を加算する。

$$score(s') \mathrel{+}= \frac{N(LUT_k, 0)}{2^k} \quad (s' \in S'_k) \tag{21}$$

これを QBF_k の新しい解が見つかるたびに繰り返す。すべての解が見つかったあと、それぞれの解に対するスコアを以下のように計算する。最初にすべての解のスコアを $score(Sol^i_k) = 0$ と初期化する。次に以下のようにスコアを更新する。

$$score(Sol^i_k) \mathrel{+}= score(s') \quad (s' \in S'^i_k) \tag{22}$$

これによって解ごとのスコアを計算することができた。このスコアが最も高い解を QBF 問題の解として選択する。

この手法は毎回の解の選択で最もスコアが高いもの、つまり毎回の問題で最良の解を選んでいく手法である。ただしこの方法で最終的に選ばれた S'_n が全サブセットの中で最良であるとは限らない。今回は全サブセットを網羅的に探索することはできないため、以上のようなヒューリスティック的手法を採用する。

3.4 アルゴリズム全体の流れ

ここまでのインクリメンタルに FF を選択する手法についてまとめて全体のフローを説明する。今 S_m からサブセット S'_n を選び、そのサブセットによる Inductive Invariant である LUT_n を求める問題を想定する。

(1) $k = 1$ から QBF_{init}、式 (9) を解き、問題が SAT になるまで k をインクリメントする。SAT になった場合、その問題 QBF_{init} の解をすべて求める。

(2) QBF_{init} の解について前述したスコアリングによりスコアをつけ、そのスコアが最も高いものを解として選択し、$k = i_{init} + 1$ とする。

(3) FF を追加する問題 QBF_k を解く。SAT になった場合、QBF_k の全ての解を探索する。QBF_k が UNSAT で解が 1 つも存在しない場合はランダムに追加する FF を選ぶ。

(4) QBF_k について前述したスコアリングによりスコアをつけ、そのスコアが高いものを解として選択する。QBF_k が UNSAT であった場合にはランダムに FF を選んでいるため、スコアリングは必要ない。k をインクリメントして、$k \leq n$ であればステップ 3 に戻る。それ以外であればステップ 5 に進む。

(5) その時点のサブセット S'_n が LUT の入力となっている FF のサブセットで、それによって計算されている LUT_n が Inductive Invariant である。

このようなアルゴリズムでサイズが n のサブセット S'_n と、そのサブセットを入力とした LUT が計算できる。このアルゴリズムではインクリメンタルに計算を行っているため、$i_{init} \leq k \leq n$ の任意の k に対して、LUT_k も計算されている。またこのアルゴリズムでは 3.2 章で述べたような前の問題で計算された LUT から Unreachable だとわかっている状態には変数を割り当てない。このことから前の問題で多くの Unreachable 状態を発見できている場合に多くの変数を削減することができ、計算量を抑えることができる。また最終的な解は最終的に選択された FF のサブセット S'_n が与えられた場合には、それによる元々の Inductive Invariant の結果と同じになる。つまり $S'_n = SelectFF(S_m)$ と与えられれば、式 (8) によってこのアルゴリズムと同様の解 LUT_n を得ることができる。

4. Inductive Invariant の回路最適化への応用

次に Inductive invariant の計算結果の補集合として得られる Unreachable 状態によって回路を最適化する手法について検討する。Unreachable 状態に対応する FF の値は決して実現しないので、組み合わせ回路部分から見ると外部ドントケアとなり外部ドントケアを利用可能な最適化手法を適用することで回路サイズの縮小が期待できる。外部ドントケアとは 0 でも 1 でも論理最適化ツールが自由に決めることができる値のことである。また inductive invariant によって計算された到達不可能な状態では、LUT のサイズが大きければ大きいほど多くの Unreachable 状態を見つけることができる。そのため最適化の際にも LUT のサイズを大きくすればするほど回路を大きく削減することができると考えられる。

[5] では図 2 のように到達不可能な場合にはその FF の値をすべて 0 として扱う手法が提案されている。これは外部ドントケアを利用できる論理合成ツールは一般的に広まっておらず、外部ドントケアとせず Unreachable 状態ではいつも同じ入力値 (ここではすべて 0) とすることで回路の簡単化を図っている。実際に最も広く利用されているオープンソースの論理合成ツール ABC[9] では外部ドントケアを利用する機能がない。組み合わせ回路に入力される値を LO_{Cin}、元々の回路の FF の値を LO とすると以下のようになる。

$$LO_{Cin} = \begin{cases} LO & \text{in reachable states} \\ 0s & \text{in unreachable states} \end{cases} \tag{23}$$

ここでは以上の最適化手法を改良して図 3 のように回路

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

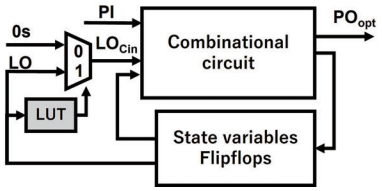

図 2 [5] で用いられている Unreachable 状態を利用した回路最適化の方法

を最適化することを考える。Unreachable 状態は通常の動作では決して実現しないため、その状態における回路の出力は外部ドントケア X に固定することが可能である。ドントケアには回路内部の制約から発生する内部ドントケアと、上記のような別の解析により与えられる外部ドントケアが存在する。もともとの回路の出力を PO、新しく最適

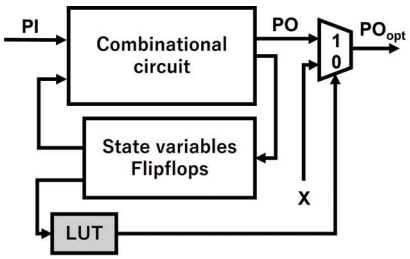

図 3 Unreachable 状態を利用して回路の出力を外部ドントケア X に固定する回路

化する回路の出力を PO_{opt} とすると以下のようになる。

$$PO_{opt} = \begin{cases} PO & \text{in reachable states} \\ X & \text{in unreachable states} \end{cases} \quad (24)$$

X は外部ドントケアを表す。この手法では Unreachable 状態において、組み合わせ回路に入力される FF の値ではなく、回路の出力を固定する。また固定する値は 0 ではなく、外部ドントケアとすることでより強力な最適化を行えることを期待する。

[5] と本手法の最も大きな違いは回路を最適化する手法である。[5] の手法は固定される値がすべて 0 であるため、一般的な回路と同様すべての信号は 0,1 の 2 値である。この場合、回路の最適化には ABC のような一般的な論理合成ツールに含まれる論理最適化を用いることができる。それに対して本手法では外部ドントケアを含めて最適化を行う必要がある。一般的な論理合成ツールは 0,1 の 2 値のみを扱えて、外部ドントケアを含んだ最適化を行うことができない。提案する外部ドントケアは順序回路の Unreachable 状態から計算されるものであり、そのようなドントケアを扱うことは一般的に複雑で難しい問題となる。ここでは外部ドントケアを含む回路の最適化方法として許容関数 (Permissible function) を計算する Transduction Method[10] を用いる。許容関数は 0,1 に加えてドントケアを扱うことができ、それを BDD を用いて実装することが可能である [11]。回路中のそれぞれのノードに対して許容関数を計算し、最適化後の回路がもとの許容関数を満たすように回路中のゲート、ワイヤーを削減していく。

この外部ドントケアを用いた最適化は順序回路としての最適化であることに注意する。多くの最適化ツールは順序回路に対してその組み合わせ回路部分のみを最適化する。つまり順序回路の FF の入力と出力をすべて切り離し、FF の出力を新しい入力として扱う。そしてそのような組み合わせ回路に対して最適化を行う。それに対して提案手法は Unreachable 状態を考慮することで最適化を行う。そのため最適化後の回路と元々の回路を組み合わせ回路として比較すると一致しないが、その分強力な最適化になると期待できる。

5. 実験

5.1 実験セットアップ

本章では提案したインクリメンタルな FF 選択アルゴリズムによる到達可能性の解析及び外部ドントケアを利用した回路最適化について行った実験について述べる。順序回路のベンチマークとして ISCAS89 の回路 [12] を用いた。初めに回路を And-Inverter-Graph(AIG)[13] に変換して計算を行った。AIG とはすべてのゲートを 2 入力の AND ゲートと NOT ゲートに変換して表現したものである。ISCAS89 の回路の入力数 (PI)、出力数 (PO)、FF 数 (FF)、AIG での AND ゲート数 (AND) はそれぞれ表 1 のようである。またすべての実験でタイムアウトは 3600 秒とした。QBF ソルバーとしては ABC 内に実装されたものを用いた。また回路最適化の実験では Transduction Method として ABC 内に実装された BDD パッケージ [14] を用いた。

表 1 ISCAS89 ベンチマーク回路の仕様

circuit	PI	PO	FF	AND
s27	4	1	3	8
s298	3	6	14	102
s344	9	11	15	105
s382	3	6	21	140
s386	7	7	6	166
s400	3	6	21	148
s420.1	18	1	16	160
s444	3	6	21	155
s641	35	24	19	146
s953	16	23	29	347
s1196	14	14	18	477
s1238	14	14	18	532
s5378	35	49	179	1389
s9234	19	22	228	1958
s13207	31	121	669	2719
s15850	14	87	597	3560
s38584	12	278	1452	12400

5.2 到達可能性の解析

ここでは ISCAS89 回路 3 章で提案したアルゴリズムで Inductive Invariant を計算した結果について示す。既存手法との比較のために FF の選択方法として、ファイルの記

載順に上から選ぶ方法と [7] で提案されている不定値の伝搬を解析する手法でも同様の計算を行った。結果は表 2 のようになった。一番上の行の Description order、X based analysis、Proposed はそれぞれファイルの記載順で選択する手法、不定値解析を行う方法、提案手法を表している。また FFs、LUTsize、number of states はそれぞれ回路の持つ FF 数、用いた LUT の入力数、その LUT が持つ真理値表の行数を表している。number of states は LUT size が k の時、2^k となる。それぞれの手法で number of 1s、ratio、Time はそれぞれ LUT における 1 を出力する行数 ($N(LUT, 1)$)、全体の行数に対して 1 を出力する行数のパーセンテージ ($\frac{N(LUT_k,1)}{2^k}$)、およびかかった時間である。TO はタイムアウト、3600 秒を表している。また 3 つの手法を比較して最も小さい Reahcable 状態のスーパーセットを計算できたものを赤字で示している。実験は LUT size を 5,10,15 と変化させて行った。

結果からわかるようにほとんどすべての回路で提案手法が最も小さなスーパーセットを発見することができた。回路が持つ FF の数が少ない回路では FF のサブセットを選択する必要がないためすべての手法で同様のスーパーセットを発見している。その場合、提案手法以外の 2 つの手法は網羅的な探索を行っていないため計算時間が短いという結果になっている。また提案手法では LUT のサイズを大きくしたときに見つかっている Unreachable 状態の比率 (表中の ratio) が下がっている。これは提案手法における FF を追加したときに解となるスーパーセットが小さくなる条件、式 (17) により適切な FF が選択されているということである。

5.3 外部ドントケアを用いた回路最適化

ここでは ISCAS89 回路に対して 4 章で提案した外部ドントケアを利用した最適化を行った結果を示す。実験では LUT のサイズを 5,10,15 と変化させて実験を行ったが、回路中の FF 数が 15 個よりも少ないものに関しては適宜小さいサイズの LUT を用いた。また外部ドントケアを含む最適化方法は ABC 内に実装された mspf コマンドに外部ドントケアを挿入することで行った。それ以外に同じ mspf コマンドを用いて外部ドントケアを用いない場合と、ABC 内の最適化 dc2 を用いて最適化を行う実験も行った。この 2 つの手法は LUT を挿入しない元々の回路に適用した。提案手法以外の 2 つの手法は Unreachable 状態を考慮しない最適化であるので組み合わせ回路部分のみの最適化ということになる。外部ドントケアを利用せずに mspf を適用した場合には内部ドントケアのみが考慮されて最適化が行われる。3 つの手法のいずれの最適化も回路サイズが変化しなくなるまで複数回適用した。結果は表 3 のようになった。Proposed 列の LUT size、number of 1s はそれぞれ LUT のサイズ、LUT における 1 を出力する行数 ($N(LUT, 1)$) である。また ratio は LUT で全体の行数に対して 1 を出力する行数のパーセンテージを示した。つまり $\frac{N(LUT_k,1)}{2^k}$ である。AND(with LUT)、AND(MSPF & remove LUT) はそれぞれ LUT を付与した図 3 の回路の AND ゲート数、その回路を MSPF によって最適化した後に LUT を取り除いた回路のゲート数である。また mspf、dc2、original はそれぞれ mspf コマンドで外部ドントケアを用いず最適化した後のゲート数、dc2 コマンドで最適化した後のゲート数、元々の回路のゲート数である。これらの最適化は組み合わせ回路部分にのみ影響しており、LUT を挿入しているわけではないため提案手法でこれらのゲート数と比較されるのは AND(MSPF & remove LUT) のゲート数である。そのためこれらの行は LUT size には依存せずに一定である。最終的な最適化後の回路が最も小さいものを赤字で示した。

多くの回路においても LUT のサイズが大きいときには提案手法での最適化が最も回路を小さくできていることがわかる。Unreachable 状態を考慮して外部ドントケアを挿入する最適化が有効であることが確認できる。一方で LUT が小さい場合には他の最適化方法の方が回路が小さい場合も存在する。LUT が小さい場合には Reachable 状態のスーパーセットが大きいため、外部ドントケアとして扱われる状態が少ないため提案手法の有効性が下がるためであると考えることができる。

またタイムアウトになっている 2 つの回路に関しては BDD のノード数が大きくなってしまう。これはドントケアを利用する Transduction Method の本質的な問題だと考えれれ、dc2 の高速な計算時間と回路サイズとのトレードオフになっている。

6. 結論

本論文では MUX を用いて網羅的にフリップフロップを選択する方法を定式化し、それをインクリメンタルな QBF 問題として解くことで計算量を抑えながらより良いフリップフロップのサブセットを見つける方法について提案した。そのアルゴリズムによって求められたフリップフロップのサブセットは既存手法よりもほとんどすべてのベンチマーク回路において非常に小さな Reachable 状態のスーパーセットを計算可能であった。またアルゴリズムの中で計算量を削減するためにインクリメンタルに問題を解いているが、それによって見つかっているスーパーセットが小さければ小さいほど高速に計算を行うことが可能であった。

また Inductive Invariant の応用として計算された Unreachable 状態を利用して外部ドントケアを導入する手法を提案した。また外部ドントケアを扱える最適化手法として Transduction Method を用いた。この最適化手法によって回路を組み合わせ回路部分のみだけではなく順序回路として最適化することが可能である。最適化結果として BDD を生成できる多くの回路に対して一般的な最適化ツール ABC

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

表 2　ISCAS89 ベンチマークに対する Inductive Invariant の計算：FF の選択方法はファイルの記載順、不定値解析、提案手法の 3 つ

				Description order			X based analysis			Proposed		
circuit	FFs	LUTsize	number of states	number of 1s	ratio [%] (1s/all)	Time[s]	number of 1s	ratio [%] (1s/all)	Time[s]	number of 1s	ratio [%] (1s/all)	Time[s]
s27	3	3	8	**6**	**75.00%**	0.01	**6**	**75.00%**	0.01	**6**	**75.00%**	0.43
s298	14	5	32	20	62.50%	0.01	10	31.25%	0	**7**	**21.88%**	3.35
		10	1024	56	5.47%	0.04	49	4.79%	0.03	**41**	**4.00%**	4.73
		14	16384	**218**	**1.33%**	1.63	**218**	**1.33%**	1.98	**218**	**1.33%**	9.23
s344	15	5	32	20	62.50%	0.01	32	100.00%	0.02	**20**	**62.50%**	1.29
		10	1024	513	50.10%	0.26	888	86.72%	0.42	**270**	**26.37%**	9.71
		15	32768	**2625**	**8.01%**	37.01	**2625**	**8.01%**	43.06	**2625**	**8.01%**	126.10
s382	21	5	32	16	50.00%	0.01	20	62.50%	0.01	**7**	**21.88%**	11.85
		10	1024	73	7.13%	0.04	190	18.55%	0.13	**20**	**1.95%**	20.75
		15	32768	337	1.03%	5.64	225	0.69%	4.66	**36**	**0.11%**	29.84
s386	6	5	32	9	28.13%	0.01	11	34.38%	0.01	**8**	**25.00%**	0.99
		6	64	**13**	**20.31%**	0.02	**13**	**20.31%**	0.01	**13**	**20.31%**	2.23
s400	21	5	32	16	50.00%	0	20	62.50%	0	**7**	**21.88%**	5.29
		10	1024	73	7.13%	0.03	190	18.55%	0.09	**20**	**1.95%**	11.38
		15	32768	337	1.03%	6.18	225	0.69%	3.96	**234**	**0.71%**	23.07
s444	21	5	32	20	62.50%	0	20	62.50%	0	**9**	**28.13%**	6.10
		10	1024	400	39.06%	0.2	190	18.55%	0.15	**46**	**4.49%**	12.20
		15	32768	4036	12.32%	54.84	449	1.37%	7.83	**225**	**0.69%**	25.33
s641	19	5	32	10	31.25%	0.01	32	100.00%	0.02	**2**	**6.25%**	0.90
		10	1024	320	31.25%	0.16	115	11.23%	0,06	**12**	**1.17%**	4.31
		15	32768	1325	4.04%	16.06	1438	4.39%	14.27	**147**	**0.45%**	7.77
s953	29	5	32	15	46.88%	0.01	9	28.13%	0.01	**7**	**21.88%**	11.85
		10	1024	47	4.59%	0.03	37	3.61%	0.02	**20**	**1.95%**	20.75
		15	32768	58	0.18%	0.67	79	0.24%	1.14	**36**	**0.11%**	29.84
s1196	18	5	32	32	100.00%	0	22	68.75%	0	**16**	**50.00%**	3.91
		10	1024	207	20.21%	0.08	199	19.43%	0.09	**78**	**7.62%**	10.71
		15	32768	1609	4.91%	19.02	1102	3.36%	14.31	**724**	**2.21%**	53.71
s1238	18	5	32	32	100.00%	0	22	68.75%	0	**17**	**53.13%**	3.45
		10	1024	207	20.21%	0.08	199	19.43%	0.09	**117**	**11.43%**	12.12
		15	32768	1609	4.91%	19.29	1102	3.36%	13.08	**769**	**2.35%**	71.22
s5378	179	5	32	32	100.00%	0	17	53.13%	0	**6**	**18.75%**	67.16
		10	1024	64	6.25%	0.03	513	50.10%	0.19	**23**	**2.25%**	98.07
		15	32768	2048	6.25%	20.39	16385	50.00%	157.36	**35**	**0.11%**	136.22
s9234	228	5	32	32	100.00%	0	32	100.00%	0	**1**	**3.13%**	0.97
		10	1024	1024	100.00%	0.43	1024	100.00%	0.43	**1**	**0.10%**	2.10
		15	32768	32768	100.00%	390.47	32768	100.00%	388.53	**1**	**0.00%**	3.83
s13207	669	5	32	**1**	**3.13%**	0	32	100.00%	0	**1**	**3.13%**	1.46
		10	1024	**1**	**0.10%**	0	515	50.29%	0.21	**1**	**0.10%**	3.12
		15	32768	8	0.02%	0.17	16449	50.20%	184.37	**1**	**0.00%**	4.80
s15850	597	5	32	8	25.00%	0	32	100.00%	0	**1**	**3.13%**	0.75
		10	1024	256	25.00%	0.11	1024	100.00%	0.47	**1**	**0.10%**	3.71
		15	32768	385	1.17%	5.54	32768	100.00%	356.01	**1**	**0.00%**	7.23
s38584	1452	5	32	**1**	**3.13%**	0	16	50.00%	0	**1**	**3.13%**	3.03
		10	1024	4	0.39%	0	512	50.00%	0.2	**1**	**0.10%**	5.99
		15	32768	4	0.01%	0.08	16384	50.00%	187.78	**1**	**0.00%**	9.20

の最適化コマンドよりも強力な最適化を行うことが可能であり、本手法の有効性を確認できた。

参考文献

[1] Cho, H., Hachtel, G., Macii, E., Poncino, M. and Somenzi, F.: Automatic state space decomposition for approximate FSM traversal based on circuit analysis, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 12, pp. 1451–1464 (online), DOI: 10.1109/43.552079 (1996).

[2] Govindaraju, S., Dill, D., Hu, A. and Horowitz, M.: Approximate reachability with BDDs using overlapping projections, *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pp. 451–456 (online), DOI: 10.1145/277044.277169 (1998).

[3] Yang, Z., Ashar, P. and Gupta, A.: SAT-Based Image Computation with Application in Reachability Analysis, Vol. 1954, pp. 354–371 (online), DOI: 10.1007/3-540-40922-X_22 (2000).

[4] Fujita, M.: Logic analysis and optimization with quick identification of invariants through one time frame analysis, *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015*, pp. 102–107 (online), DOI: 10.1109/MEMCOD.2015.7340476 (2015).

[5] Fujita, M.: Detection of test Patterns with Unreachable States through Efficient Inductive-Invariant Identification, *Proceedings of the Asian Test Symposium*, Vol. 2016-Febru, pp. 31–36 (online), DOI: 10.1109/ATS.2015.13 (2015).

[6] Fujita, M.: Automatic identification of assertions and invariants with small numbers of test vectors, *Proceedings of the 33rd IEEE International Conference on Computer Design, ICCD 2015*, pp. 463–466 (online), DOI: 10.1109/ICCD.2015.7357149 (2015).

[7] Koike, R. and Fujita, M.: Efficient Reachability Analysis

Vol.2022-ARC-248 No.29
Vol.2022-SLDM-198 No.29
Vol.2022-EMB-59 No.29
2022/3/10

表 3　ISCAS89 ベンチマークに対する外部ドントケアを用いた最適化及び ABC の dc2 コマンドによる最適化 (TO は 3600 秒)

	Proposed						mspf		dc2		original
	LUT size	number of 1s	ratio[%] (1s/all)	AND (with LUT)	AND (MSPF & remove LUT)	Time[s]	AND	Time[s]	AND	Time[s]	AND
s27	1	2	100.00%	8	**8**	0.01	**8**	0.01	**7**	0.01	8
	2	3	75.00%	13	**7**	0.01	**8**	0.01	**7**	0.01	
	3	6	75.00%	11	**7**	0.01	**8**	0.01	**7**	0.01	
s298	5	20	62.50%	140	**66**	0.02	**79**	0.02	**75**	0.02	102
	10	56	5.47%	185	**57**	0.02	**79**	0.02	**75**	0.02	
	14	218	1.33%	199	**51**	0.05	**79**	0.02	**75**	0.02	
s344	5	20	62.50%	114	**94**	0.04	**100**	0.01	**94**	0.01	105
	10	513	50.10%	160	**94**	0.05	**100**	0.01	**94**	0.01	
	15	2625	8.01%	2151	**92**	0.05	**100**	0.01	**94**	0.01	
s382	5	16	50.00%	146	**111**	0.1	**108**	0.04	**100**	0.01	140
	10	73	7.13%	184	**109**	0.31	**108**	0.04	**100**	0.01	
	15	337	1.03%	254	**102**	0.52	**108**	0.04	**100**	0.01	
s386	5	9	28.13%	176	**98**	0.07	**95**	17.68	**110**	0.08	166
	6	13	20.31%	180	**87**	0.32	**95**	17.68	**110**	0.08	
s400	5	16	50.00%	154	**96**	3.31	**101**	3.05	**105**	0.03	148
	10	73	7.13%	192	**95**	6.22	**101**	3.05	**105**	0.03	
	15	337	1.03%	262	**85**	42.79	**101**	3.05	**105**	0.03	
s444	5	20	62.50%	158	**99**	5.02	**108**	4.45	**103**	0.01	155
	10	400	39.06%	163	**95**	5.27	**108**	4.45	**103**	0.01	
	15	4036	12.32%	268	**95**	23.89	**108**	4.45	**103**	0.01	
s641	5	10	31.25%	150	**120**	3.25	**128**	8.05	**120**	0.01	146
	10	320	31.25%	150	**120**	4.12	**128**	8.05	**120**	0.01	
	15	1325	4.04%	194	**111**	19.41	**128**	8.05	**120**	0.01	
s953	5	15	46.88%	355	**250**	0.41	**262**	13.13	**311**	0.03	347
	10	47	4.59%	461	**248**	1.24	**262**	13.13	**311**	0.03	
	15	58	0.18%	504	**236**	92.92	**262**	13.13	**311**	0.03	
s1196	5	32	100.00%	477	**433**	2.12	**380**	6.43	**424**	0.01	477
	10	207	20.21%	599	**361**	5.91	**380**	6.43	**424**	0.01	
	15	1609	4.91%	1800	**361**	6.12	**380**	6.43	**424**	0.01	
s1238	5	32	100.00%	532	**389**	6.71	**382**	6.23	**460**	0.06	532
	10	207	20.21%	654	**377**	43.99	**382**	6.23	**460**	0.06	
	15	1609	4.91%	1855	**325**	3518.68	**382**	6.23	**460**	0.06	
s1494	5	32	100.00%	673	**381**	4.58	**380**	5.83	**511**	0.09	673
	6	48	75.00%	687	**412**	4.12	**380**	5.83	**511**	0.09	
s5378	5	32	100.00%	1389	**1163**	121.21	**1144**	174.07	**987**	0.05	1389
	10	64	6.25%	1404	**1018**	52.59	**1144**	174.07	**987**	0.05	
	15	2048	6.25%	1404	**1018**	51.24	**1144**	174.07	**987**	0.05	
s9234	5	1	3.13%	1962	**1123**	312.52	**1672**	21.94	**1344**	0.11	1958
	10	1	0.10%	1967	**949**	2251.11	**1672**	21.94	**1344**	0.11	
	15	32	0.10%	1967	**949**	2011.3	**1672**	21.94	**1344**	0.11	
s13207	5	1	3.13%	2723	**2213**	2218.43	**2419**	42.19	**2030**	0.09	2719
	10	1	0.10%	2728	**2176**	1301.7	**2419**	42.19	**2030**	0.09	
	15	8	0.02%	2730	**2011**	3011.55	**2419**	42.19	**2030**	0.09	
s15850	5	8	25.00%	3560	×	TO	×	TO	**2720**	0.04	3565
	10	256	25.00%	3560	×	TO	×	TO	**2720**	0.04	
	15	385	1.17%	3615	×	TO	×	TO	**2720**	0.04	
s38584	5	1	3.13%	12404	×	TO	×	TO	**9792**	0.62	12400
	10	4	0.39%	12406	×	TO	×	TO	**9792**	0.62	
	15	4	0.01%	12411	×	TO	×	TO	**9792**	0.62	
Average					74.77%		77.81%		81.73%		100.00%

Based on Inductive Invariant Using X-value Based Flipflop Selection, *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 34–40 (online), DOI: 10.1109/ISQED51717.2021.9424357 (2021).

[8] Aanand, A. and Singh, V.: Enhanced Reachability Analysis Using Inductive Invariant Computation, Master's thesis, Indian Institute of Technology, Bombay (2021).

[9] Synthesis, B. L. and Group, V.: ABC:A System for Sequential Synthesis and Verification (2005).

[10] Muroga, S., Kambayashi, Y., Lai, H. and Culliney, J.: The transduction method-design of logic networks based on permissible functions, *IEEE Transactions on Computers*, Vol. 38, No. 10, pp. 1404–1424 (online), DOI: 10.1109/12.35836 (1989).

[11] Matsunaga, Y. and Fujita, M.: Multi-level logic optimization using binary decision diagrams, *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pp. 556–559 (online), DOI: 10.1109/ICCAD.1989.77012 (1989).

[12] Brglez, F., Bryan, D. and Kozminski, K.: Combinational profiles of sequential benchmark circuits, *IEEE International Symposium on Circuits and Systems,*, pp. 1929–1934 vol.3 (online), DOI: 10.1109/ISCAS.1989.100747 (1989).

[13] Biere, A.: The AIGER And-Inverter Graph (AIG) Format Version 20070427 (2007).

[14] Miyasaka, Y., Mishchenko, A. and Fujita, M.: A Simple BDD Package without Variable Reordering and Its Application to Logic Optimization with Permissible Functions (2019).