IPSJ Transactions on System LSI Design Methodology Vol.7 46–55 (Feb. 2014)

[DOI: 10.2197/ipsjtsldm.7.46]

Regular Paper

SAT-based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions

Satoshi Jo[1,a)] Takeshi Matsumoto[2,b)] Masahiro Fujita[2,3,c)]

Received: May 24, 2013, Revised: August 30, 2013,
Accepted: October 30, 2013, Released: February 14, 2014

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. The proposed method first finds a candidate of LUT configurations that can correct a given circuit by SAT solvers. Then, it checks the correctness of the candidate by checking equivalence between the circuit with LUTs and its specification. Although this can be solved as SAT problem, we introduce to use commercial equivalence checker to improve the efficiency. If the result of the check is "not equivalent", an input pattern showing the non-equivalence will be added, and the method repeats with the pattern added. Through the experimental results on ISCAS '85 benchmark circuits and Open RISC 1200 microprocessor design, 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, Boolean satisfiability, Quantified Boolean Formula, look-up table

1. Introduction

Due to the continuous increase of chip size and complexity, it is extremely difficult to generate 100% correct fabricated chips. There are varieties of reasons why designers cannot use the manufactured chips as they are: logical and electrical bugs, last minutes changes of specifications, various and complicated manufactured faults, and others. If there are no in-field programmability in the chips, they cannot simply be used, or significant efforts may be required from the usage aspect of the chips, such as large revisions of their controlling software in order to hide the problems from the end-users of the chips. Moreover, when designing large chips, such as SoCs, many advanced features must be implemented as quickly as possible, and so, it is very helpful to incorporate IPs (Intellectual Properties: reusable circuit blocks) into a new design as much as possible. IPs may not be used as they are, as their functionality including their interfaces can be slightly incompatible to the rest in the chip. If there are in-field programmability in the IPs, such incompatibility can be rectified and the IPs can be used with re-programming.

Studies in the errors and bugs of microprocessors show small additional programmability could be sufficient in most cases [1]. By introducing small amount of in-field programmability, large portions of the problems of the chips may be avoided without disabling intended functionalities (which are typically newly-developed enhanced ones), while they have to be disabled by workarounds in software if programmability is not introduced.

Introducing programmability to gate-level circuits is recently studied in Ref. [2]. In the work, PPC (Partially Programmable Circuit) has been proposed. In PPC, some of the original (sets of) gates are replaced with look-up tables (LUTs) or multiplexers whose functionality can be modified in the fields just like FPGA by utilizing scan chains or others. PPC has been evaluated recently from the viewpoints of design rectification under errors, bugs, faults, and Engineering Change Order (ECO, small changes of specification), and it has been shown that significant percentages of the problem can be avoided by introducing small numbers of LUTs into the designs [3].

In this paper, following the idea of PPC, we present several target problems which can be solved by introducing programmable circuits into combinational circuits, that is, replacing some of the original gates with LUTs. Then, we propose a circuit rectification and debugging method to efficiently solve those problems using SAT solvers. Although the selection of LUT insertions/replacements in the target designs is just based on simple heuristics or random selections, our rectification and debugging method can deal with much larger circuits than the techniques shown in Ref. [3], which is confirmed by experimental results on several benchmark designs including 16-bit combinational mul-

[1] Department of Electrical Engineering and Information Systems, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] VLSI Design and Education Center, The University of Tokyo, Bunkyo, Tokyo 113–0032, Japan
[3] CREST, JST
[a)] jo@cad.t.u-tokyo.ac.jp
[b)] matsumoto@cad.t.u-tokyo.ac.jp
[c)] fujita@ee.t.u-tokyo.ac.jp
A preliminary version of this paper has been published in the proceedings of the 21st Asia Test Symposium, pp.19–24, November, 2012 (DOI:10.1109/ATS.2012.55).

tipliers which are considered to be hard to analyze with Boolean methods, such as SAT and BDD.

As can be seen from Ref. [3], the problems can be formulated as satisfiability problem of Quantified Boolean Formula (QBF), which we call QBF evaluation problem (or QBF problem, in short) in this paper. That is, the problem is formulated as satisfiability checking of the following statement:

"Under appropriate programs for LUTs (existentially quantified), the circuit behaves correctly for all possible input values (universally quantified)"

When satisfiable, a program for LUTs with which the statement is satisfied is a solution of the problem. In Ref. [3], state-of-the-art QBF solvers with some heuristic problem decomposition on parallel computing machines are used to solve the QBF evaluation problems. Although QBF solvers have been improved a lot recently, the QBF evaluation problems they can solve are much smaller than the normal SAT problems, as the complexity of QBF evaluation problems is much higher than that of normal SAT problems. So in Ref. [3], only relatively small benchmarks are tried.

Recently, a new approach to solve QBF evaluation problems has been proposed [4]. By utilizing CEGAR (Counter Example Guided Abstraction Refinement) paradigm, which has been used with lots of success in formal verification area [5], [6], QBF problems can be solved by repeatedly applying normal SAT solvers. The evaluation results in Ref. [4] show that QBF solvers with CEGAR paradigm outperform the other QBF solvers significantly. Inspired from this approach, in this paper, we show a method which tries to solve a satisfiability problem of QBF for rectification and debugging of combinational circuits with LUTs by repeated applications of normal SAT solvers. In this paper, first, we propose to simply apply the method in Ref. [4] to solving QBF problems in order to find the configurations of LUTs. Then, we improve its efficiency by introducing to use equivalence checkers. Equivalence checkers are used in our method, instead of SAT solvers in Ref. [4], when we check whether a candidate solution of a QBF problem is really a solution of the problem. The proposed method is complete in the sense that it finishes with a solution if there exists, or proves there is no solution, under sufficient given time. Experimental results show that much larger circuits with much more LUTs than Ref. [3] can be examined by the proposed method. For example, 16 bit combinational multipliers with 100 LUTs can be analyzed within several minutes.

Our contributions can be summarized as follows:

- Our proposed method reduces the original QBF problems shown in Ref. [3] into a series of two SAT problems, one for partial requirements coming from universally quantified variables, and the other for checking equivalence between rectification candidates and specifications.
- The former SAT problems are relatively easy as only variables which represent truth table values of the inserted LUTs are treated as primary inputs. So, as long as the numbers of inserted LUTs are not large (less than several hundred or so), they can be solved very quickly.
- The latter SAT problems are basically equivalence checking between two combinational circuits. Therefore, we can utilize the well developed Boolean comparison methods, such as Refs. [7], [8], [9], which will dramatically improve the performance. There are well developed commercialized tools that implement most of the techniques. As such Boolean equivalence checkers can deal with circuits having millions of gates, at least theoretically we can say such large circuits may potentially be rectified and debugged with proposed method. Through experimental comparisons, we show that solving the latter SAT problems can be faster when commercial equivalence checkers are used instead of normal SAT solvers.
- Although our implementation used in the experiments are relatively very simple and naive, the results are surprisingly good compared with the previous results [3], in terms of circuit sizes and processing speed.
- As discussed in Section 1.1, our proposed method has many potential applications in circuit design and debugging which requires efficient solving methods such as ours to solve problems with practical size. For example, using our method, we can do debug or ECO for circuits implemented on FPGAs even after routing and placement are done. Even if debug or ECO happens after its layout is finalized on an FPGA, we can apply the proposed method to change only LUT functionality. This could be very useful for FPGA based designs as well as FPGA based prototyping.

The paper is organized as follows. In the next section, we clarify the problems we are trying to solve and show possible practical applications where our proposed method can effectively work. In Section 3, related works on the use of programmable circuits for rectification and debugging are reviewed. Then in the following section, the proposed method is presented with details. In Section 5, experimental results with ISCAS '85 benchmark circuits are shown. The last section gives concluding remarks.

1.1 Target Applications

Figure 1 shows some examples of target applications that our proposed method can solve. In those applications, it is essential to find a configuration, i.e., truth table values for LUTs and/or selecting variable values for MUXes, with which the entire logic circuit behaves equivalently to the given specification. However, in such applications, methods to find such configurations have to deal with a large number of logic gates and LUTs/MUXes, in practice. Although only LUTs are inserted in circuits as programmable logics, in this paper, extension to handling MUXes and other types of programmable logics is straightforward.

Bug correction.

Debugging gate-level designs is still a large concern in VLSI designs. There are some existing works to identify the possible locations of bugs such as Refs. [13], [21], which enables us to know which locations should be corrected. However, those methods cannot provide a logic function of each bug location to correct the entire circuit. Assume a set of LUTs and a logic specification on primary inputs and outputs are given, our proposed method finds a configuration of each LUT for correction, which means that a corrected logic function at some location in the circuit can be automatically derived. Note that those inserted LUTs

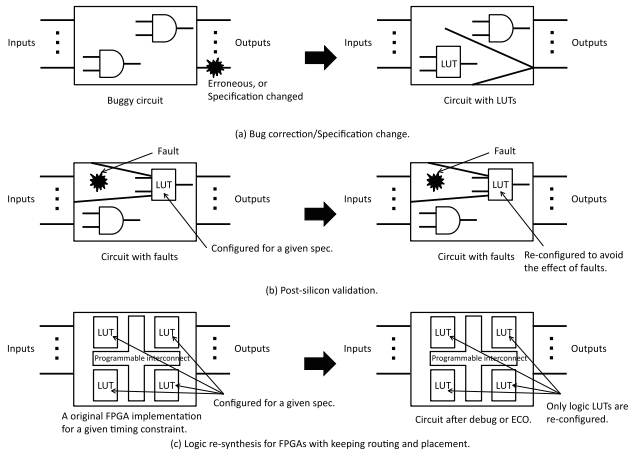

Fig. 1 Target applications.

are not necessarily required to be physically implemented in the circuit. In this way of using programmability, the purpose is finding a logic function for correction at some specified location. Figure 1 (a) shows this situation.

Specification change.

In a similar way to bug correction discussed above, our proposed method can solve a specification change problem if a set of gates that can be changed with constraints on timing and other constraints and a new logic function are given.

Post-Silicon Validation.

As shown in Ref. [2], adding programmability in circuits can improve the chance of validation after VLSI chips are fabricated. In this case, programmable devices such as LUTs and MUXes are physically implemented in circuits, and our proposed method tries to find a configuration of them so that the fabricated circuit can behave correctly even with faults and/or design bugs. This is shown in Fig. 1 (b).

Bug correction of FPGAs without changing routing and placement.

Nowadays, FPGA is widely used to implement or emulate designs. While it has a great advantage that it can be re-programmed when a design is changed (due to refinement, debug, specification change, and others), re-synthesis takes a long time since it includes time consuming physical synthesis. Therefore, if we can re-program only logic LUTs in FPGAs and keep the original routing and placement results, as shown in Fig. 1 (c), the runtime of re-synthesis can be reduced much. We consider this as one of the most important applications of the proposed method.

2. Illustrative Example

In this section, we briefly explain how our method works to solve problems described in Section 1.1.

Assume that a combinational circuit shown in **Fig. 2** is the original design. Also assume that the designers have decided to use 2-input LUTs. Note that using 2-input LUTs is just an example. Although we use 2-input LUTs throughout the paper, any numbers of inputs can be processed in the same way. If we introduce a LUT having N inputs, we need 2^N variables to represent its truth table.

Then, a set of candidate locations (there are many other sets as well) can be the ones shown in **Fig. 3**, and if all are replaced with 2-input LUTs, the circuit becomes the one shown in **Fig. 4**. There are design decisions to be made in terms of which types of LUTs should be introduced (i.e., the number of LUT inputs) and where they should be inserted (maybe all possible locations, or any subsets of them). Once these are fixed and decided by the designers, we can generate QBF to be solved for each problem we like to deal with, such as bug fixes, ECO, fault tolerance, and others as shown in Fig. 1.

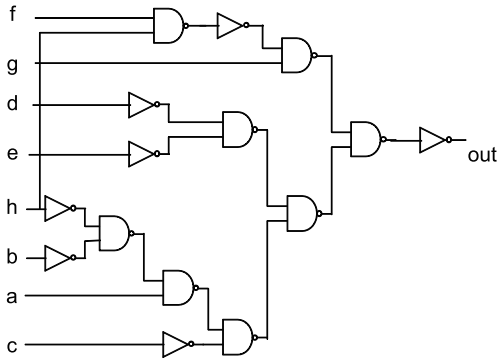

Fig. 2 An example combinational circuit.

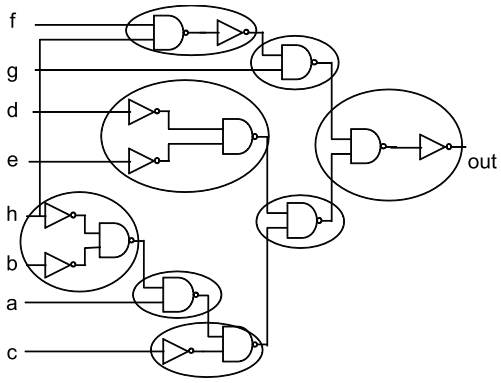

Fig. 3 A set of candidates to be replaced by 2-input LUTs.

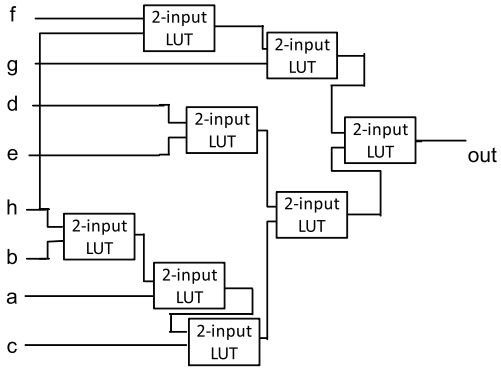

Fig. 4 After all candidates are replace by 2-input LUTs.

For a given circuit with LUTs and logic specification, our proposed method tries to find a configuration (i.e. truth table values of LUTs) to satisfy the specification. In this example, assume its logic specification of the output *out* is represented as $SPEC(a,b,c,d,e,f,g,h)$, the QBF that needs to be solved for satisfiability is:

$$\exists(LUT\_config.).\forall\{a,b,c,d,e,f,g,h\}.out = SPEC(a,b,c,d,e,f,g,h)$$

where LUT_config denotes a set of truth table values of all inserted LUTs. Since the formula has only one universal quantifier, it is a two-level QBF. Solving those generated QBF are the target of our proposed method which will be shown in the later sections.

3. Related Works

3.1 Debugging and ECO Methods for Gate-level Circuits

Efficiency of debugging and ECO is a key issue in modern VLSI designs. In Ref. [13], a debugging method based on Boolean satisfiability is proposed for gate-level circuits. This method identifies logic gates that need to be replaced with another logic function in order to make all given counterexamples corrected for the same input patterns. This is achieved by inserting a multiplexer at an output of each logic gate, which enables for a logic circuit under debugging to select either the original value computed in the circuit or an arbitrary value given by the other input of the multiplexer. Such logic gates can be found by formulating the problem as a satisfiability (SAT) problem and solving with SAT solvers. If all counterexamples are corrected by selecting the appropriate value through the free variable of the input of the multiplexer at an output of a gate, the circuit can be corrected by replacing the gate with another logic function, at least for those counterexamples.

To improve the efficiency of ECO in VLSI design, it is very important to make re-synthesized portions as small as possible [10], [11]. Recently Krishunaswamy et al. address this problem by optimizing a logic difference between a circuit under ECO and a modified functional specification [12]. The optimization is realized by identifying equivalent sub-circuits to the modified specification, based on SAT-based equivalence checking. To avoid re-doing time-consuming placement and routing processes, programmability in a circuit under ECO plays an important role. In Ref. [14], Yoshida et al. proposes a patchable hardware architecture where functional modifications are achieved by introducing memory elements in a circuit. The architecture consists of a programmable datapath, a fixed controller, and a memory for patches. According to given functional modifications, control signals are generated using the patch memory instead of the controller. Also, in Ref. [2], a partially-programmable circuit (PPC) is proposed to improve yield. In PPC, some logic gates are replaced with LUTs to introduce programmability, which enables ECO even after manufactured without additional placement and routing. In PPC, efficiently finding a configuration of LUTs to meet ECO is essential in practical use. In this paper, targeting a circuit where some original gates are replaced by LUTs, we propose a way to quickly find the LUT configuration.

3.2 QBF Solvers

Solving LUT configurations for a given functional specification can be formulated as a QBF evaluation problem [3]. There are many techniques proposed to efficiently solve QBF problems [18], [19], [20], and the performance of solvers has been improved over the last decades, which can be seen in a series of competitions called QBFEVAL [15]. However, even for the state-of-the-art solvers, solving industrial-sized practical QBF problems is still a hard problem.

Recently, Boolean SAT-based QBF solving are proposed in Ref. [4], [16]. Here, let us briefly explain how the method in Ref. [16] works for 2-level QBFs in the form of $\exists X \forall Y.\phi$, where ϕ is a propositional formula and X and Y denote a set of variables, respectively. When this QBF is satisfiable, we call a valuation of X a solution of the problem if it satisfies the formula. If no such valuation of X exists, the QBF is unsatisfiable. The QBF problem can be rewritten as a Boolean SAT problem for $\wedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu]$, where $\mathcal{B}^{|Y|}$ denotes a set of valuation of Y. The method first takes one or more valuations of Y, which is denoted as $W \in B^{|Y|}$. Then, $\wedge_{\mu \in W} \phi[Y/\mu]$ is solved as a Boolean SAT problem. If UNSAT, it immediately means the original QBF problems are elaborated to value 0. Otherwise, SAT solvers generate a valuation ν of X which satisfies ϕ for all valuations of Y in W. Then, the method checks whether ν is a solution of the QBF problem, which is solved as another Boolean SAT problem. If ν is not a solution (i.e., some valuations of Y do not satisfy ϕ with ν), a counterexample is generated by SAT solvers and added to W. This process is repeated until a solution ν is found or the QBF is proved to be unsatisfiable. If the number of iterations is small, we can quickly find a solution of a given QBF evaluation problem by the method.

We show how the method works using the following example.

$$\exists a, b \forall c, d.(a + b + c + \neg d)(a + \neg b + \neg c + \neg d)(\neg a + \neg b + c + d)$$

First, we take one or more valuations of (c, d). We take $(0, 0)$ in this example. Then, we solve the following SAT problem, where c and d in the formula are substituted by 0. As the formula is SAT, we get a valuation $(a, b) = (0, 0)$ from SAT solvers, for example.

$$\exists a, b.(a + b + 0 + \neg(0))(a + \neg b + \neg(0) + \neg(0))(\neg a + \neg b + 0 + 0) \Leftrightarrow \exists a, b.(\neg a + \neg b)$$

Next, we check whether $(a, b) = (0, 0)$ is a solution of the original QBF problem by checking whether there exist any valuations of (c, d) which make the formula evaluated to 0. If exist, $(a, b) = (0, 0)$ is not a solution of the original problem, since a solution must make the formula evaluated to 1 for any valuation of (c, d). This can be checked by solving the following SAT problem. If $(a, b) = (0, 0)$ is a solution of the original problem, the following formula is unsatisfiable.

$$\exists c, d.\neg((0 + 0 + c + \neg d)(0 + \neg(0) + \neg c + \neg d)(\neg(0) + \neg(0) + c + d)) \Leftrightarrow \exists c, d.\neg(c + \neg d)$$

Here, we get a valuation $(c, d) = (0, 1)$ from SAT solvers, which can be seen as a counterexample showing $(a, b) = (0, 0)$ is not a solution. Then, we check whether there exists any valuation of (a, b) which satisfies the formula for both $(c, d) = (0, 0)$ and $(c, d) = (0, 1)$ at the same time, by solving the following SAT problem.

$$\exists a, b.(a + b + 0 + \neg(0))(a + \neg b + \neg(0) + \neg(0))(\neg a + \neg b + 0 + 0)(a + b + 0 + \neg(1))(a + \neg b + \neg(0) + \neg(1))(\neg a + \neg b + 0 + 1) \Leftrightarrow \exists a, b.(a + b)(\neg a + \neg b)$$

This time, we get a valuation $(a, b) = (1, 0)$ and check whether it is a solution of the original QBF problem or not.

$$\exists c, d.\neg((1 + 0 + c + \neg d)(1 + \neg(0) + \neg c + \neg d)(\neg(1) + \neg(0) + c + d)) \Leftrightarrow \exists c, d.\neg(1)$$

The above formula is unsatisfiable, which means there is no valuation of (c, d) which makes the formula evaluated to 0. Therefore, $(a, b) = (1, 0)$ is proved to be a solution of the given QBF prob-

lem. Please note that we need two valuations of (c, d) even though there are four possible valuations.

In Ref. [4], the method is generalized to deal with an arbitrary number of quantifiers. In this work, we apply this QBF solving method based on counterexample guided abstraction refinement (CEGAR) in order to derive LUT configurations in circuits with LUTs and show the effectiveness through the experiments.

4. Proposed Method for Corrections with LUT

As discussed in the previous sections, our rectification and debugging problem with LUT insertion is formulated as two-level QBF problem. We basically replace sub-circuits with LUTs. If we use 2-input LUTs, this replacement becomes the one shown in Ref. **Fig. 5**. The 2-input LUT can be represented by introducing four variables, $v_{00}, v_{01}, v_{10}, v_{11}$, each of which corresponds to the value of one row of the truth table. Those four variables are multiplexed with the two inputs to the original gate as control variables. If we introduce M of 2-input LUTs, the circuit has $4 \times M$ more variables. We represent those variables as v_{ij} or v (vector of v_{ij}). As shown in **Fig. 6**, v variables are treated as primary inputs as they are programmed (assigned appropriate values) before using the circuit. t variables in the figure correspond to intermediate variables for the circuit. They appear in the CNF of the circuit for SAT/QBF solvers.

For simplicity, we assume single output combinational circuits in the following, but as can be easily seen, extension to multiple output circuits is straitforward. If the logic function at the output of the circuit is represented as $f(x)$ where x is an input variable vector, after replacements with LUTs, the QBF to be solved for satisfiability becomes:

$$\exists v. \forall x. f(v, x) = SPEC(x)$$

where $SPEC$ is the logic function that represents the specification

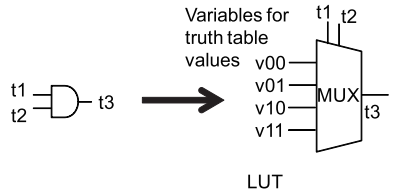

Fig. 5 LUT is represented with multiplexed four variables as truth table values.

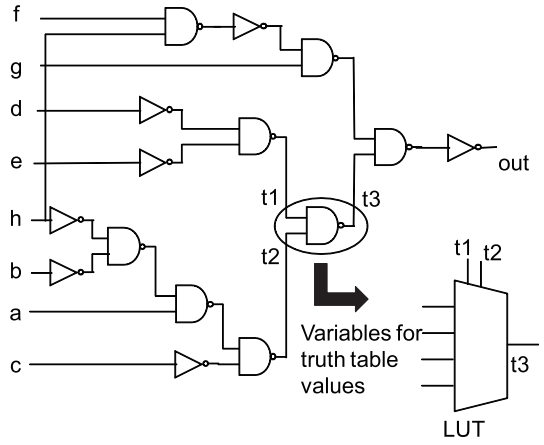

Fig. 6 Introduction of one 2-input LUT into the circuit.

to be implemented.

Although this can be simply solved by any QBF solvers theoretically, as can be seen from the experimental results, only small circuits or small numbers of LUTs can be successfully processed. Instead of doing that way, we here like to solve given QBF problems by repeatedly applying normal SAT solvers using the ideas shown in Ref. [4].

The basic idea is the following. Instead of checking all value combinations on the universally quantified variables, we just pick up some small numbers of value combinations and assign them to the universally quantified variables. This would generate SAT problems whose satisfiability is a necessary conditions for the satisfiability of the original QBF. Please note that here we are dealing with only two-level QBF, and so if universally quantified variables got assigned actual values (0 or 1), the resulting formulae become simply SAT problem. For example, if we assign two combinations of values for x variables, the resulting SAT problem to be solved becomes like:

$$\exists v.(f(v, assign1)) = SPEC(assign1)$$
$$\wedge(f(v, assign2) = SPEC(assign2)),$$

where $assign1$ and $assign2$ are variable assignments to x, respectively. Since $assign1$ and $assign2$ are assignments to the input variables x, they are considered as input patterns for the circuit. If the function $SPEC$ is given as the gate-level circuit, the values of $SPEC(assign1)$ and $SPEC(assign2)$ can be calculated simply by simulating the specification circuit. Even if not, we can calculate them with respect to $SPEC$, anyway.

Then we can simply apply any SAT solvers to them. If there is no solution (i.e., the result of the SAT problem is UNSAT), we can conclude that the original QBF is also unsatisfiable. If there is a solution found (i.e., the result of the SAT problem is SAT), we need to make sure that that is actually a solution for the original QBF problem. In this case, we can identify the value of v which makes the formula satisfiable from the variable assignment generated by SAT solvers. Here, we denote this variable assignment to v as $v\_candidate$. Please note that $v\_candidate$ is a constant vector where 1 or 0 is assigned to all variables in v. To check $v\_candidate$ is an actual solution of the original QBF problem, we simply make sure the following:

$$\forall x. f(v\_candidate, x) = SPEC(x)$$

This is again solved by SAT solvers by complementing the formula:

$$(\forall x. f(v\_candidate, x) = SPEC(x))$$
$$\equiv \exists x. f(v\_candidate, x) \neq SPEC(x).$$

and checking if this has a solution or not. If this does not have any solution, then the current solution $v\_candidate$ is actually a solution of the original QBF. But if this has a solution, say x_sol, that is a counter example for the current solution $v\_candidate$ and so is added to the conditions as shown below:

$$\exists v.(f(v, assign1)) = SPEC(assign1)$$
$$\wedge(f(v, assign2) = SPEC(assign2))$$

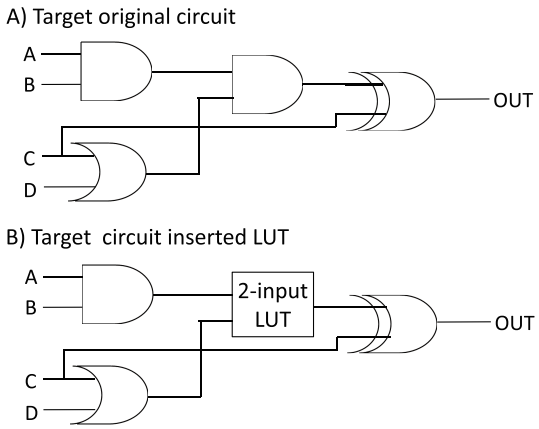

Fig. 7 Target circuit with and without LUT.

$$\wedge(f(v, x\_sol) = SPEC(x\_sol).$$

And then, the solving process is just repeated.

Checking *v_candidate* is a solution of the original QBF problem is just an equivalence checking problem between two combinational circuits. Therefore, instead of solving by SAT solvers, equivalence checker for combinational circuits can solve it faster. Such equivalence checkers can check equivalence faster especially when two circuits have larger similarity, since it utilizes internal equivalence points to decompose the problem to smaller ones. The condition that two circuits have large similarity is satisfied in many cases of our target applications. For example, in post-silicon validation, some gates in the original circuit are replaced with LUTs in the fabricated chip. Thus, equivalence checker can utilize the similarity in checking equivalence.

Here, we show a rectification example of a circuit. The target circuit is shown in **Fig. 7** (A). We replace AND gate in the circuit with the 2-input LUT as shown in Fig. 7 (B). Our method tries to find a truth table of the LUT which makes the functions of these two circuits equivalent. If the logic function at the output of the circuit with the LUT is represented as $f(v, A, B, C, D)$, where v represents the truth table of the LUT, the QBF formula to be solved becomes:

$$\exists v.\forall A, B, C, D.f(v, A, B, C, D) = SPEC(A, B, C, D)$$

where SPEC is the logic function at the output of the original circuit. First, we take one or more input patterns of the circuit. In this example, we take $(A, B, C, D) = (0, 0, 0, 0)$ and assign them to the two circuits. Then, the SAT problem to be solved becomes:

$$\exists v.(f(v, 0, 0, 0, 0)) = SPEC(0, 0, 0, 0)$$

The satisfiability of this formula is a necessary condition for that the original QBF problem has a solution. Solving this SAT problem, we get $v = (0, 0, 0, 0)$ from SAT solvers as the formula is SAT. n is a candidate of the truth table of the LUT. Next, we check whether it is a real solution or not by solving the following SAT problem.

$$\exists A, B, C.f(0000, A, B, C, D) \neq SPEC(A, B, C, D)$$

If the result of this SAT problem is UNSAT, it means that there is no input pattern which makes two circuits behave differently, and

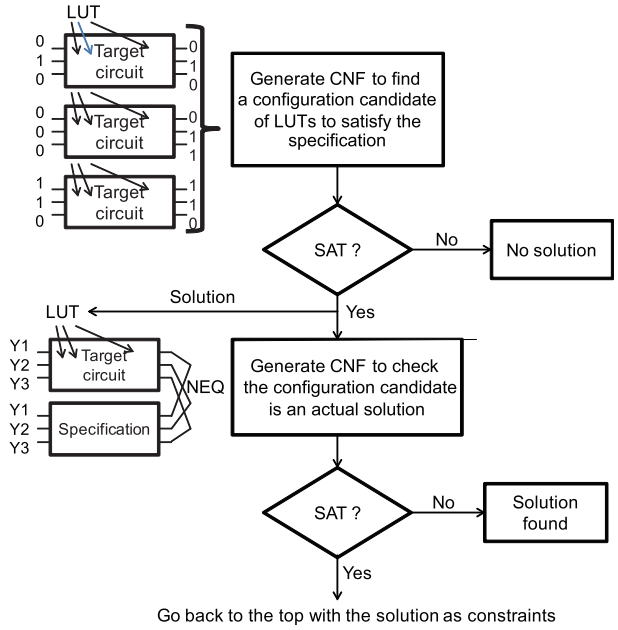

Fig. 8 Overall flow of our proposed method.

the candidate v is proved to be a real solution. In this example, the result is SAT and we get $(A, B, C, D) = (1, 1, 1, 1)$ as a counterexample. We add it to the conditions as an additional essential qualification. Then, to find the next candidate of a solution, the following SAT problem is solved.

$$\exists v.(f(v, 0, 0, 0, 0)) = SPEC(0, 0, 0, 0)$$
$$\wedge(f(v, 1, 1, 1, 1) = SPEC(1, 1, 1, 1))$$

We repeat this process until a solution is found or the problem is proved to have no solution. In this example, we get $v = (1, 0, 0, 0)$ as the next candidate of the truth table of the LUT by solving the SAT problem above. To check whether it is a real solution or not, the following SAT problem is solved.

$$\exists A, B, C, D.f(1000, A, B, C, D) \neq SPEC(A, B, C, D)$$

This time, the formula above is UNSAT, and and $v = (1, 0, 0, 0)$ is proved to be a solution of the QBF problem.

The overall flow of the proposed method is shown in **Fig. 8**. A specification and a target circuit with LUTs are assumed to be given.

First, we prepare a set of input patterns to start with, which can be generated in a random way, a user-guided, or others. For those patterns, the expected output values are calculated for the specification. Then, the first SAT problem is generated and solved by SAT solvers to find a configuration candidate of LUTs to satisfy the specification. If it is UNSAT, we immediately conclude that there is no configuration of LUTs to satisfy the specification. If SAT, a configuration candidate is obtained from a variable assignment that SAT solvers return.

Next, the method generates and solve the second SAT problem to check this configuration candidate is an actual solution. The SAT problem is made so that it is true when there exist one or more input patterns that makes the outputs of the target circuit different from the specification. If it is SAT, an input pattern to make them inequivalent from each other is obtained from a variable assignment by SAT solvers. This pattern is added to the first

Table 1 Experimental results for benchmarks having solutions.

Benchmark	Original gates	LUT replaced	Proposed			sKizzo [18]	
			Solved (in 3,600 sec)	Average time (sec)	Average iterations	Solved (in 3,600 sec)	Average time (sec)
c499	202	10	20/20	0.7	3.2	20/20	7.7
		20	20/20	1.2	6.5	20/20	18.3
		50	20/20	3.4	16.8	17/20	279.2
		100	20/20	7.3	30.4	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	18.0	2/20	1,735.5
		20	20/20	10.8	32.3	0/20	Time out
		50	20/20	26.7	53.9	0/20	Time out
		100	20/20	72.8	89.4	0/20	Time out
c1908	880	10	20/20	4.7	15.9	14/20	655.0
		20	20/20	10.4	27.4	0/20	Time out
		50	20/20	26.1	47.2	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	1,899.2
c3540	1669	10	20/20	4.9	10.9	8/20	1,239
		20	20/20	12.2	21.2	0/20	Time out
		50	20/20	53.3	49.2	0/20	Time out
		100	20/20	205.5	92.3	0/20	Time out
c5315	2406	10	20/20	7.35	13.0	0/20	Time out
		20	20/20	19.3	23.7	0/20	Time out
		50	20/20	79.5	52.9	0/20	Time out
		100	20/20	236.8	92.7	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

SAT problem and the process is repeated. Otherwise, the candidate is a solution. This second SAT problem can be solved by combinational equivalence checkers, instead of SAT solvers.

5. Experimental Results

5.1 Setup

The proposed method has been implemented and evaluated with ISCAS'85 benchmark circuits [23] and OpenRISC 1200 microprocessor design taken from Opencores [24]. For each circuit, we replaced 10, 20, 50, and 100 of 2-input gates with LUTs randomly and compared with the original logic functions as specification. When the results by our proposed method are compared with those by a QBF solver, we used the completely same designs including LUT replacement, though LUTs are inserted randomly.

The implementation details are shown below:

- Our implementation accepts gate-level netlist designs written in Verilog.
- The designs are processed with ABC (version abc70930 [22]) and AIGER (version aiger-1.9.4 [25]) to translate to CNF formulae.
- PicoSAT (version picosat-936 [26]) and sKizzo (v0.8.2 [18]) are used as our base SAT and QBF solvers, respectively.
- Synopsys Formality is used as an equivalence checker.
- All experiments are run on CPU Intel(R) Core(TM)2 Duo 3.33 GHz with 4GB of memory.

Table 2 Experimental results for benchmarks that do not have solutions.

Bench-mark	Original gates	LUT replaced	Solved (in 3,600 sec)	Average time (sec)	Average iterations
c880	383	10	5/5	1.4	8.0
		20	5/5	1.6	6.6
		50	5/5	0.8	3.8
		100	5/5	4.8	16.0
c2670	1193	10	5/5	3.8	7.8
		20	5/5	7.2	11.8
		50	5/5	8.8	11.2
		100	5/5	15.8	14.2
c6288	2406	10	5/5	0.6	1.6
		20	5/5	1.2	3.0
		50	5/5	4.8	8.4
		100	5/5	12.0	14.0

5.2 Comparison between Our Proposed Method and QBF Solver

First, we conducted a set of experiments on combinational circuits taken from ISCAS'85 benchmarks in order to compare the performance of our proposed method and QBF solver, sKizzo. In this experiments, for each number of LUT replacement, we generated 20 different circuits.

The results are shown in **Table 1**. In the table, "Original gates," "LUT replaced," and "Solved" denote the number of gates in the original circuit before LUT replacement, the number of LUTs inserted, and the number of circuits the method could solve within 3,600 sec, respectively. Also, "Average time" and "Average iterations" show the average processing time and the average number of iterations of the loop shown in Fig. 8, in all cases that the

method could solve. Note that cases we could not get a solution within 3,600 sec are excluded from the average calculation.

As can be seen from the table, simple application of QBF solvers can only solve the cases of small circuits with small numbers of LUTs, whereas the proposed method can solve almost all cases (except for some cases of c6288 with large numbers of LUTs) within several minutes.

An important thing we need to recognize in those results is the numbers of iterations are relatively small. For example, c6288 with 100 LUTs has 32 primary inputs and $2^2 \times 100 = 400$ truth table variables, however, only 41.4 iterations (i.e., 41.4 input patterns applied in the first SAT problem in Fig. 8) are required to identify truth table values of all 100 LUTs. In the sense, the results show that our proposed method works well to solve the problem of finding LUT configurations in gate-level circuits.

Table 3 Experimental results for sequential circuits.

Bench-mark	Time-frames	Original gates	LUT replaced	Solved (in 18,000 sec)	Average time (sec)	Average iterations
s9234	3	6081	10	20/20	1.3	1.1
			20	20/20	2.3	2.2
			50	20/20	4.4	3.8
			100	20/20	8.5	6.3
	5	10135	10	20/20	3.4	1.8
			20	20/20	5.3	2.8
			50	20/20	14.9	6.3
			100	20/20	39.7	11.9
	10	20270	10	20/20	13.2	3.0
			20	20/20	23.6	5.1
			50	20/20	79.8	11.5
			100	20/20	221.4	20.6
s15850	3	10344	10	20/20	5.1	2.1
			20	20/20	7.2	3.0
			50	20/20	21.8	7.0
			100	20/20	64.7	14.1
	5	17240	10	20/20	12.0	2.5
			20	20/20	27.4	5.2
			50	20/20	105.3	12.8
			100	20/20	363.3	25.7
	10	34480	10	20/20	59.8	4.7
			20	20/20	186.3	10.4
			50	20/20	679.7	22.6
			100	20/20	4,830.5	39.5
s38584	3	34344	10	20/20	5.6	0.7
			20	20/20	7.7	1.2
			50	20/20	14.3	2.5
			100	20/20	20.1	3.4
	5	57240	10	20/20	17.1	1.4
			20	20/20	34.1	2.9
			50	20/20	112.5	7.4
			100	20/20	259.6	13.0
	10	114480	10	20/20	290.0	7.2
			20	20/20	738.4	13.5
			50	20/20	7,686.0	23.0
			100	8/20	4,830.5	39.5

5.3 No Solution Cases

To evaluate the performance of our proposed method for cases that do not have solution, we applied our method to three ISCAS'85 benchmark circuits, c880, c2670, and c6288. In the experiments, we intentionally changed the specification circuits so that the target circuit with LUTs cannot behave equivalently to the specification for any LUT configuration.

Table 2 shows the results. All results in Table 2 show that the proposed method can actually prove the non-existence of solutions. As can be seen from the table, such cases are solved relatively faster with less iterations than cases having solutions.

5.4 Sequential Circuits

Three sequential circuits taken from ISCAS'85 benchmark were experimented. To handle sequential circuits, we unrolled them in 3, 5, and 10 timeframes (clock cycles) and replaced each flip-flop with a single wire. The resulting circuits are just combinational, and our proposed method can handle them without any extension. For LUT insertion, we inserted LUTs before the timeframe unrolling, that is, a unrolled circuit has $T \times N$ LUT instances after unrolled, where T is the number of unrolled timeframes and N is the number of LUTs inserted before unrolled. In addition, by unrolling timeframes, one LUT in a circuit before unrolled is replicated T times, but those T LUTs have the same truth table values.

The results are shown in **Table 3**. "Timeframes" denotes the number of unrolled timeframes. In the experiments, the upper limit of runtime was set to 18,000 sec. Compared to the results shown in Table 1, the average number of iterations tends to smaller when the number of replaced LUTs are same. A possible reason to explain this trend is the ratio of LUTs over total gates. Since the possible truth table values that a LUT can take are constrained by surrounding logic gates, we can say that less number of input patterns (i.e., constraints) are required to find a configuration of LUTs if the ratio of LUTs is smaller. Also, from the results in Table 3, we can see that the number of iterations increases almost linearly with the number of LUTs, which implies

Table 4 Experimental results for OpenRISC 1200 microprocessor design.

			Proposed (SAT solver only)			Proposed (EQ checker)		
Time-frames	Original gates	LUT replaced	Solved (in 18,000 sec)	Average time (sec)	Average iterations	Solved (in 18,000 sec)	Average time (sec)	Average iterations
2	35402	10	20/20	31.0	5.5	20/20	169.1	5.4
		20	20/20	61.5	9.0	20/20	277.5	9.0
		50	20/20	188.2	18.2	20/20	639.0	19.7
		100	20/20	499.5	31.7	20/20	1,153.3	35.3
3	53103	10	20/20	144.8	11.5	20/20	646.7	12.2
		20	20/20	387.1	21	20/20	1,114.1	20.0
		50	20/20	1,714.0	48.1	20/20	2,820.4	47.3
		100	12/20	10,253.8	84.5	20/20	5,668.8	87.7
4	70804	10	20/20	1,310.0	16.9	20/20	1,685.4	18.0
		20	20/20	1,945.0	35.0	20/20	1,685.4	35.0
		50	1/20	17,588.0	64.0	20/20	9,094.4	82.0
		100	0/20	Timeout	Timeout	10/20	12,905.3	108.3

the number of iterations does not explode when a large number of LUTs are inserted in circuits.

5.5 Microprocessor Designs

To evaluate how much speed-up can be achieved by introducing combinational equivalence checker in solving the second SAT problem, we applied our proposed method to OpenRISC 1200 microprocessor design with several different settings shown in **Table 4**. From the results in the table, we can see that using an equivalence checker improves the performance for problem instances with 3 or 4 timeframe unrolling and 50 or 100 LUTs. However, for circuits with less gates and less LUTs, using an equivalence checker reduces the performance due to the overhead in invoking the equivalence checker. We can conclude that equivalence checker is more useful in our proposed method when circuits has more gates and more LUTs, compared to using SAT solvers for checking equivalence.

6. Conclusions

In this paper, we have shown a rectification and debugging method for combinational circuits with LUT insertions based on QBF formulation. The QBF problems are not solved by QBF solvers for satisfiability, but solved by repeatedly applying normal SAT solvers. The experimental results show significant improvement over existing methods especially for large circuits and circuits having many LUTs. This is partly because the number of iterations to find and check a solution candidate is small in most cases. In addition, we propose to utilize combinational equivalence checkers to accelerate solving SAT problems to check that a solution candidate is actually a solution. This utilization of equivalence checkers makes the method faster for large circuits with many LUTs, which is confirmed through the experiments.

References

[1] Sarangi, S., Narayanasamy, S., Carneal, B., Tiwari, A., Calder, B. and Torrellas, J.: Patching Processor Design Errors with Programmable Hardware, *IEEE Micro*, Vol.27, No.1, pp.12–25 (2007).

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

[3] Mangassarian, H., Yoshida, H., Veneris, A.G., Yamashita, S. and Fujita, M.: On error tolerance and Engineering Change with Partially Programmable Circuits, *The 17th Asia and South Pacific Design Automation Conference* (*ASP-DAC 2012*), pp.695–700 (2012).

[4] Janota, M., Klieber, W., Marques-Silva, J. and Clarke, E.M.: Solving QBF with Counterexample Guided Refinement, *The 15th International Conference on Theory and Applications of Satisfiability Testing* (*SAT '12*) (2012) (to appear).

[5] Das, S. and Dill, D.L.: Counter-example based predicate discovery in predicate abstraction, *4th International Conference on Formal Methods in Computer-Aided Design* (*FMCAD 2002*), pp.19–32 (2002).

[6] Clarke, E.M., Gupta, A. and Strichman, O.: SAT-based counterexample-guided abstraction refinement, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.23, No.7, pp.1113–1123 (2004).

[7] Jain, J., Mukherjee, R. and Fujita, M.: Advanced Verification Techniques Based on Learning, *32nd Annual ACM/IEEE Design Automation Conference*, pp.420–426 (1995).

[8] Kuehlmann, A., Paruthi, V., Krohm, F. and Ganai, M.K.: Robust boolean reasoning for equivalence checking and functional property verification, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.21, No.12, pp.1377–1394 (2002).

[9] Mishchenko, A., Chatterjee, S., Brayton, R.K. and Een, N.: Improvements to combinational equivalence checking, *2006 IEEE/ACM International Conference on Computer-Aided Design*, pp.836–843 (2006).

[10] Brand, D., Drumm, A., Kundu, S. and Narain, P.: Incremental synthesis *IEEE/ACM International Conference on Computer-aided Design* (1994).

[11] Fujita, M., Kakuda, T. and Matsunaga, Y.: Redesign and Automatic Error Correction of Combinational Circuits, *Logic and Architecture Synthesis*, G. Saucier (Ed.), pp.253–262, North-Holland: Elsvier Science Publishers B.V. (1991).

[12] Krishunaswamy, S., Ren, H., Modi, N. and Puri, R.: DeltaSyn: An Efficient Logic Difference Optimizer for ECO Synthesis, *2009 International Conference on Computer-Aided Design*, pp.789–796 (2009).

[13] Smith, A., Veneris, A., Ali, M.F. and Vialas, A.: Fault Diagnosis and Logic Debugging Using Boolean Satisfiability, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.10, pp.1606–1621 (Oct. 2005).

[14] Yoshida, H. and Fujita, M.: An energy-efficient patchable accelerator for post-silicon engineering changes, *9th International Conference on Hardware/Software Codesign and System Synthesis* (*CODES+ISSS 2011*), pp.13–20 (2011).

[15] Peschiera, C., Pulina, L., Tacchella, A., Bubeck, U., Kullmann, O. and Lynce, I.: The Seventh QBF Solvers Evaluation (QBFEVAL'10), *13th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2010*), pp.237–250 (2010).

[16] Janota, M. and Marques-Silva, J.: Abstraction-Based Algorithm for 2QBF, *14th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2011*), pp.230–244 (2011).

[17] Benedetti, M.: Evaluating QBFs via Symbolic Skolemization, *11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference* (*LPAR 2004*), pp.285–300 (2004).

[18] Benedetti, M.: sKizzo: A Suite to Evaluate and Certify QBFs, *20th International Conference on Automated Deduction* (*CADE-20*), pp.369–376 (2005).

[19] Ginuchiglia, E., Marin, P. and Narizzano, M.: Reasoning with Quantified Boolean Formulas, *Handbook of Satisfiability*, pp.761–780, (2009).

[20] Ginuchiglia, E., Narizzano, M. and Tacchella, A.: QuBE++: An Efficient QBF Solver, *5th International Conference on Formal Methods in Computer-Aided Design* (*FMCAD 2004*), pp.201–213 (2004).

[21] Safarpour, S., Mangassarian, H., Veneris, A., Liffiton, M.H. and Sakallah, K.A.: Improved Design Debugging Using Maximum Satisfiability, *Formal Methods in Computer Aided Design* (*FMCAD 2007*), pp.13–19 (2007).

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

[23] Hansen, M.C., Yalcin, H. and Hayes, J.P.: Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering, *IEEE Design and Test*, Vol.16, No.3, pp.72–80 (1999).

[24] OpenCores Homepage, available from ⟨http://opencores.org/⟩.

[25] AIGER Homepage, available from ⟨http://fmv.jku.at/aiger/⟩.

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

[27] sKizzo – a QBF solver, available from ⟨http://skizzo.info/⟩.

Satoshi Jo received his B.S. degree in electronic engineering from The University of Tokyo, Tokyo, Japan, in 2013. He is currently a Master's course student in the Department of Electrical Engineering and Information Systems, The University of Tokyo. His research interests include formal verification and debugging of hardware designs.

Takeshi Matsumoto received his B.S., M.S., and Ph.D. degrees in electronic engineering from The University of Tokyo, Tokyo, Japan, in 2003, 2005, and 2008, respectively. He has been a member of VLSI Design and Education Center in The University of Tokyo since 2008. His research interests include computer-aided design and formal verification, especially for high-level designs of digital systems.

Masahiro Fujita received his Ph.D. degree in engineering from The University of Tokyo, Tokyo, Japan, in 1985. He then joined Fujitsu Laboratories Ltd., Atsugi, Japan. From 1993 to 2000, he was with Fujitsu's U.S. research office and directed the CAD Research Group. In March 2000, he joined the Department of Electronic Engineering, The University of Tokyo, as a Professor. He is currently a Professor with the VLSI Design and Education Center, University of Tokyo. He has been involved in many research projects on various aspects of formal verification.

(Recommended by Associate Editor: *Kiyoharu Hamaguchi*)