2013 IEEE 43rd International Symposium on Multiple-Valued Logic

Debugging of Reversible Circuits Using πDDs

Laura Tague[1] Mathias Soeken[1,2] Shin-ichi Minato[3] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[3]Hokkaido University, Sapporo, 060-0814, Japan
{laurat,msoeken,drechsle}@informatik.uni-bremen.de, minato@ist.hokudai.ac.jp

***Abstract*—Different kinds of decision diagrams have played key roles in advancements for the synthesis of reversible circuits in the recent past. In this work, decision diagrams are used to efficiently debug reversible circuits in cases when they do not match their intentional specification. It can automatically be checked whether the faulty circuit is almost equal to a given function, i.e. it can realize the function by slightly modifying the circuit, e.g. by adding or changing a gate. For this purpose, πDDs are used which allow for a compact representation of a set of permutations.**

I. Introduction

Reversible logic serves as underlying technology for many technologies such as quantum computation [1], optical computing [2], and low power design [3], [4]. It is based on the principle that every operation must be reversible such that it can be executed in both directions. Besides computing an output pattern for an input assignment, a reversible function can also determine the inputs for a given output. Based on different gate libraries many models have been proposed to describe reversible circuits that realize reversible functions e.g. for its use in quantum computers or low power CMOS designs [5].

Throughout the recent years, significant achievements were often achieved by utilizing decision diagrams of different kind. Decision diagrams offer a compact representation for Boolean functions and matrices and have been widely applied in the design of reversible circuits. As examples, *Binary Decision Diagrams* (BDDs) have been applied for exact, heuristic, and hierarchical synthesis of both reversible and irreversible functions [6], [7], [8]. As an alternative to BDDs, the application of Kronecker functional decision diagrams, an extension of BDDs, has lead to further improvements [9]. *Quantum Multiple-valued Decision Diagrams* (QMDDs) [10], enabling a compact representation for complex matrices, have been used for both equivalence checking [11] and synthesis of large reversible functions ensuring a minimal number of lines [12]. Similar data-structures have efficiently been applied for the simulation and verification of quantum circuits [13], [14]. In fact, decision diagrams have been the key methodology for breakthroughs in the design of reversible circuits. BDDs allowed synthesis of minimal circuits for a significant amount of functions [6] and they enabled the synthesis of large Boolean functions with more than 100 variables [7]. For the latter case, the main problem of the algorithm is the huge amount of additional lines which impedes the practical applicability of that approach. However, the problem of additional lines in the synthesis of large functions has been solved again with decision diagrams, in particular using QMDDs [12].

However, while BDDs and QMDDs offer a compact representation for functions and matrices, the recently introduced πDDs [15] allow for a compact representation of permutations. Hence, they are an interesting extension to the set of considered decision diagrams in the design of reversible circuits. Since reversible functions constitute permutations on the input assignments, they can naturally be expressed using this data structure. In fact, πDDs do not only allow a compact representation for single permutations, but for a set of permutations. Therefore, they can particularly be applied for many problems where the above mentioned data structures are not advantageous.

The application of πDDs in the design flow for reversible circuits has been briefly investigated in [16] by conducting small experiments in order to show for which tasks πDDs may be beneficial and for which not. It turns out that πDDs are not advantageous in comparison to other data structures when being applied for synthesis of reversible circuits. In fact, they often perform significantly worse as the πDD representation for many reversible functions is exponential, as an example already such an elementary circuit as the inverter. As a result, large functions do not profit from πDDs. However, when considering several functions at once, πDDs demonstrate their strength.

This fact is exploited in the approach that is proposed in the present paper. A method is illustrated for debugging of reversible circuits [17] in cases when the circuit does not match the intended specification, i.e. the reversible function to be synthesized. If the circuit matches the function by applying a slight modification, e.g. by adding a new gate at an arbitrary position, we call the circuit *almost-equal* with respect to the considered function. Two algorithms have been developed where one algorithm solves the problem using a

0195-623X/13 $26.00 © 2013 IEEE
DOI 10.1109/ISMVL.2013.22

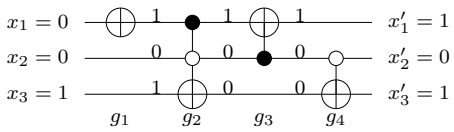


Fig. 1. Reversible circuit

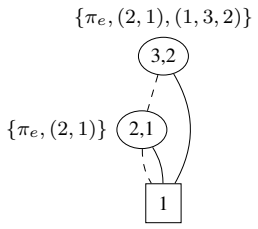


Fig. 2. πDD

naïve approach and the other one utilizes πDDs. The method based on πDDs clearly outperforms the naïve approach and illustrates the potential of this new data structure for specific tasks in the design flow of reversible functions. Although in this paper missing gate faults are considered, the algorithm can be extended to similar scenarios as well, e.g. a wrong gate, or gates that are interchanged. Furthermore, the proposed algorithm based on πDDs is generic such it can easily be extended to other possibly gate libraries, e.g. multiple-valued ones, since permutations are used as the underlying representation. Every reversible gate represents a permutation independently on the base of the respective logic.

Related work can be found in connection with the test of reversible circuits (see e.g. [18]), however, the corresponding algorithms start from a given circuits whereas in the case of debugging arbitrary circuits need to be analyzed. Other debugging work has been proposed, e.g. in [19]. Here, the authors present an algorithm that can fix faulty circuits by replacing a gate with a new sub-circuit that can consist of more than a single gate.

The paper is structured as follows. The next section provides the background, while the problem formulation is illustrated in Section III. Section IV explains the proposed algorithms in order to solve the problem and possible extensions to other logics and gate libraries are illustrated in Section V. Experimental results are given in Section VI and the paper concludes with Section VII.

II. Preliminaries

A. Reversible Functions and Circuits

A function $f\colon \mathbb{B}^n \rightarrow \mathbb{B}^n$ is called reversible if it is bijective, i.e. for each output pattern it is always possible to determine the corresponding input pattern. As a result, reversible functions represent permutations on the set $0, \ldots, 2^n - 1$. Reversible functions can be realized using reversible circuits.

Reversible circuits differ from conventional circuits, since e.g. fanout and feedback are not directly allowed. Usually, they are built as a cascade of reversible gates including e.g. the Toffoli gate, the Fredkin gate, or the Peres gate. In this paper, we focus on circuits composed of Toffoli gates. Given a set of variables $X = x_1, \ldots, x_n$ a Toffoli gate is a tuple (C, t) with $C \subset \bigcup_{x \in X} \{x, \overline{x}\}$ such that $\forall x \in X : \{x, \overline{x}\} \not\subset C$ being the set of *control lines* and $t \in X$ with $\{t, \overline{t}\} \cap C = \emptyset$ being the *target line* of the gate. A Toffoli gate inverts the target line if, and only if, all control lines x_i ($\overline{x_i}$) are set to 1 (0). Positive (negative) literals in C are called positive (negative) control lines.

Example 1: Fig. 1 shows a reversible circuit with three lines and composed of four gates. The target lines are denoted by $\bigoplus$, while a $\bullet$ represents a positive control line and a $\circ$ represents a negative control line. For example, assigning the input pattern 001 to the circuit results in the output pattern 101. Due to the reversibility, this computation can be performed in both directions.

B. πDDs

A πDD [15] is a decision diagram that allows for a compact and canonical representation for sets of permutations. πDDs are derived from ZDDs [20], a decision diagram that offers a compact representation of sets. πDDs exploit that any permutation can be decomposed uniquely into a sequence of transpositions that swap two items. For example, the permutation $(3, 5, 2, 1, 4)$ can be decomposed into a set of transpositions $\tau_{(2,1)}\tau_{(3,2)}\tau_{(4,1)}\tau_{(5,4)}$. This can be interpreted as follows: First the items 5 and 4 are swapped, then 4 with 1 and so on until the identity permutation $\pi_e = (1, 2, 3, 4, 5)$ is obtained. The canonical property of the sequence of transpositions is guaranteed by always swapping the item with the highest absolute value first.

According to this principle, the vertices in πDDs are labeled using the respective transposition (in comparison, in ZDDs the vertices are labeled using the set element). The terminal vertices 1 and 0 represent the set containing the identity permutation π_e and the empty set $\emptyset$, respectively. As an example, Fig. 2 illustrates a πDD that represents the permutations $\pi_e, (2, 1), (1, 3, 2)$. Traversing this πDD from the top to the bottom leads to the transpositions to be applied so that eventually the identity permutation results.

Several operations can be carried out efficiently on πDDs e.g. counting the number of permutations which is equivalent to counting the number of 1-paths in BDDs or ZDDs. Furthermore, calculating the Cartesian product $P * Q = \{\alpha\beta \mid \alpha \in P, \beta \in Q\}$ is efficient, which is the set of all possible composite permutations chosen from P and Q. In fact, calculating on πDDs is performed by adding the number

TABLE I
PERMUTATIONS FOR ALL POSITIVELY CONTROLLED TOFFOLI GATES ON 3 LINES

μ	$T_{0,\mu}$					$T_{1,\mu}$					$T_{2,\mu}$				
0		$\binom{000}{001}$	$\binom{010}{011}$	$\binom{100}{101}$	$\binom{110}{111}$		$\binom{000}{010}$	$\binom{001}{011}$	$\binom{100}{110}$	$\binom{101}{111}$		$\binom{000}{100}$	$\binom{001}{101}$	$\binom{010}{110}$	$\binom{011}{111}$
1			$\binom{010}{011}$		$\binom{110}{111}$			$\binom{001}{011}$		$\binom{101}{111}$			$\binom{001}{101}$		$\binom{011}{111}$
2				$\binom{100}{101}$	$\binom{110}{111}$				$\binom{100}{110}$	$\binom{101}{111}$				$\binom{010}{110}$	$\binom{011}{111}$
3					$\binom{110}{111}$					$\binom{101}{111}$					$\binom{011}{111}$

of vertices although the number of elements is multiplied [15].

C. Gate Libraries

Reversible gates and reversible circuits realize reversible functions and a reversible function in turn represents a permutation. As a consequence, we can compare circuits and functions using their respective permutations. To exploit the properties of πDDs, it is important to consider several functions at once, as it is for example the case in gate libraries. We refer to the gate library that consists of all positively controlled Toffoli gates as T_n with

$$T_n = \bigcup_{t=0}^{n-1} \bigcup_{\mu=0}^{2^{n-1}} T_{t,\mu}$$

where $T_{t,\mu}$ is the permutation that is realized by a Toffoli gate with target line t and control lines μ. Here, the control lines of a gate are represented as the binary expansion of μ excluding the target line.

Table I lists all permutations for positively controlled Toffoli gates acting on 3 lines. For the sake of an improved readability, transpositions $\tau_{(x,y)}$ are written $\binom{x}{y}$.

III. PROBLEM FORMULATION

In this work we are considering a debugging task that checks whether a faulty circuit is *almost-equal* to a given function based on a specific fault model. For illustration purposes, functions are considered that do not fulfill their functional specification due to a missing gate, however, the proposed techniques can be extended in order to work other fault models and therefore with other definitions of *almost-equality* as well. These definitions can e.g. include a wrong gate (due to a missing control) or gates that have been interchanged.

The problem formulation is exemplary illustrated by means of Fig. 3 using the circuit in Fig. 1. Given a circuit with n lines and d gates the possible search space is $(d+1) \cdot |L_n|$ where L_n is a gate library on n lines. That is, before each gate and also after the last gate, one gate from the gate library can possibly be inserted. As an example, $|T_n| = n \cdot 2^{n-1}$ for the case of positive controlled Toffoli gates.

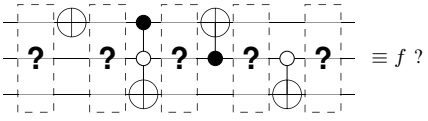


Fig. 3. Problem formulation

A naïve algorithm that solves the problem of determining whether a circuit is almost-equal to a given function is exhaustively iterating the search space by inserting a gate at each possible position and then checks the circuit and the function for equality. This procedure is very time consuming in case the circuit is not almost-equal to the function, otherwise the run-time depends on the position in which the missing gate must be inserted.

We are proposing an alternative method that is making use of πDDs. After the πDD has been built, it is sufficient to call one operation on it to decide whether the circuit is almost-equal to the given function or not. Of course, the complexity is shifted to building the πDD, however, the run-time is significantly shorter in comparison to the naïve approach as the experimental results will show. Furthermore, no difference in run-time is observed when comparing almost-equal circuits to non almost-equal circuits. Hence, the algorithm is also more robust with respect to the result.

IV. ALGORITHMS

This section describes both algorithms that have been developed in order to determine whether some faulty circuit G is almost-equal to a given function f. First, the naïve approach that exhaustively inserts gates at any possible position is presented while the πDD-based approach is explained and illustrated by means of examples afterwards.

A. Naïve Approach

The naïve approach takes all Toffoli gates for n lines and insert them one by one at all the possible positions in the circuit and then checks whether the circuit G realizes the considered function f, e.g. by simulation. It can be formalized as follows.

Algorithm N (*Naïve Approach*). Given a reversible function $f\colon \mathbb{B}^n \to \mathbb{B}^n$ and a circuit $G = g_1 \dots g_d$, such that G is not a representation of f, this algorithm determines whether

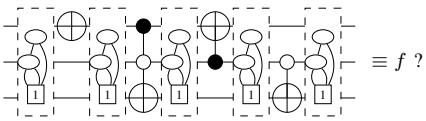


Fig. 4. Idea of the πDD-based approach

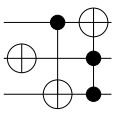


Fig. 5. Example circuit for the application of the πDD approach

there is a gate $\hat{g}$ which can be inserted at any position in G such that the modified G represents f. If such a gate $\hat{g}$ exists, then G is almost-equal to f.

N1. [Initialization.] Set $p \leftarrow 0$.

N2. [Loop over T_n.] For any gate $\hat{g} \in T_n$, perform step N3.

N3. [Insert $\hat{g}$.] Set $G' \leftarrow g_1 \dots g_p \hat{g} g_{p+1} \dots g_d$. If G' realizes f', terminate and return *true*.

N5. [Next position?] If $p < d$, set $p \leftarrow p + 1$ and return to step N1, otherwise terminate and return *false*. ∎

Clearly, Algorithm N is not efficient, which becomes evident in particular in step N2 since the size of the Toffoli gate library is exponential with respect to the number of lines. As a result, the above mentioned technique is not applicable to circuits of a larger scale.

B. πDD-based Approach

In this section, an alternative is presented which makes use of πDDs, an efficient data structure for sets of permutations. Using πDDs allows for constructing and considering several reversible functions or circuits at once using operations that can be carried out efficiently on the data structure.

The idea is illustrated by means of Fig. 4 which is based on Fig. 3. At each position where a gate could be missed, a πDD is inserted that represents all possible gates, in this case all Toffoli gates, that could be inserted to fix the circuit. Combining all combinations into a single πDD, it allows for the consideration of all possible represented functions at once.

Given a circuit $G = g_1 \dots g_d$ consisting of n lines where each gate g_i can be described by its permutation π_{g_i} and a function f represented by π_f, checking whether G is almost-equal to f can be solved using πDDs by checking if $\pi_f \in F$, where

$$F = \bigcup_{p=0}^{d} \left(\{\pi_{g_1} \dots \pi_{g_p}\} * T_n * \{\pi_{g_{p+1}} \dots \pi_{g_d}\} \right). \quad (1)$$

That is, for each position $p = 0, \dots, d$ a set of function is created by making use of the Cartesian product on πDDs, and all these resulting sets are joined via union.

Example 2: Fig. 5 shows a circuit with three Toffoli gates. By applying the πDD-based approach on this circuit, the function

$$\begin{aligned} F \;=\; & (T_3 * T_{1,0} * T_{2,1} * T_{0,3}) \cup \\ & (T_{1,0} * T_3 * T_{2,1} * T_{0,3}) \cup \\ & (T_{1,0} * T_{2,1} * T_3 * T_{0,3}) \cup \\ & (T_{1,0} * T_{2,1} * T_{0,3} * T_3) \end{aligned}$$

is obtained.

V. Extension of the πDD-based Approach

This section briefly illustrates how the above described algorithm based on πDDs can be extended in order to both support alternative gate libraries and alternative debugging problems.

A. Alternative Gate Libraries

Since reversible gates represent reversible functions which in turn represent permutations, the algorithm can readily be extended to support alternative gate libraries. Since the gate library is represented as a πDD in Eq. (1), it just needs to be replaced accordingly. In this way, also multiple-valued gate libraries can be considered. As an example, the Muthukrishnan-Stroud gate [21], which realizes a ternary reversible function, is represented by the permutation

$$\begin{aligned} & (0, 1, 2, 4, 5, 3, 8, 6, 7, \\ & \;10, 11, 9, 14, 12, 13, 15, 16, 17, \\ & \;20, 18, 19, 21, 22, 23, 25, 26, 24) \end{aligned}$$

which can be realized as a πDD.

B. Alternative Debugging Problems

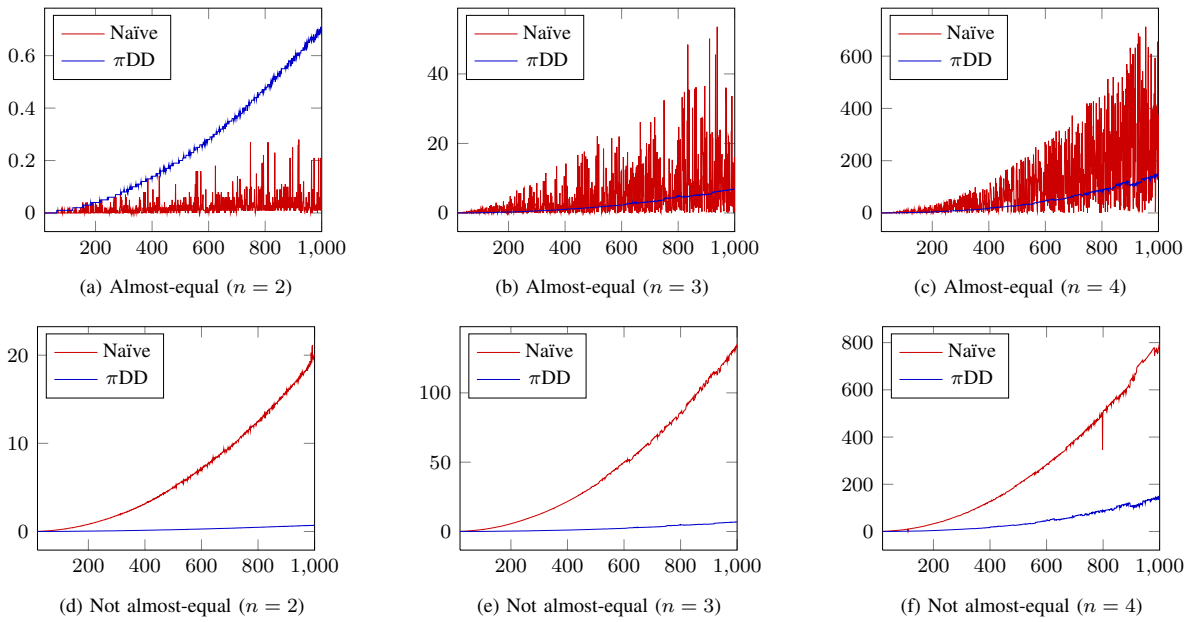
Similarly, other debugging problems can be considered when adjusting Eq. (1). If e.g. almost-equality is defined according to a missing control fault, the equation is written

$$F = \bigcup_{p=1}^{d} \left(\{\pi_{g_1} \dots \pi_{g_{p-1}}\} * \Pi_{g_p} * \{\pi_{g_{p+1}} \dots \pi_{g_d}\} \right).$$

where Π_{g_p} is a πDD that represents a set of permutations resulting from removing a control line from g_p.

VI. Experimental Evaluation

We have implemented both algorithms in C++ on top of RevKit [22] and the SAPPOROBDD package. The experiments were carried out on a 3.10 GHz Intel Core i5 machine running Linux 2.6.32. In order to evaluate the algorithm, random circuits consisting of $n = 2, 3, 4$ lines and $d =$



(a) Almost-equal ($n = 2$) (b) Almost-equal ($n = 3$) (c) Almost-equal ($n = 4$)

(d) Not almost-equal ($n = 2$) (e) Not almost-equal ($n = 3$) (f) Not almost-equal ($n = 4$)

Fig. 6. Direct comparison of approaches

$20, \ldots, 1000$ gates have been created. For each circuit two experiments have been made:

- It has been checked against a function which is almost-equal (AE). These functions have been created by randomly deleting a gate from the circuit and simulating it afterwards.
- It has been checked against functions which is not almost-equal (NAE).

The missing gate fault has been chosen as respective debugging problem and all positive controlled Toffoli gates are used as gate library.

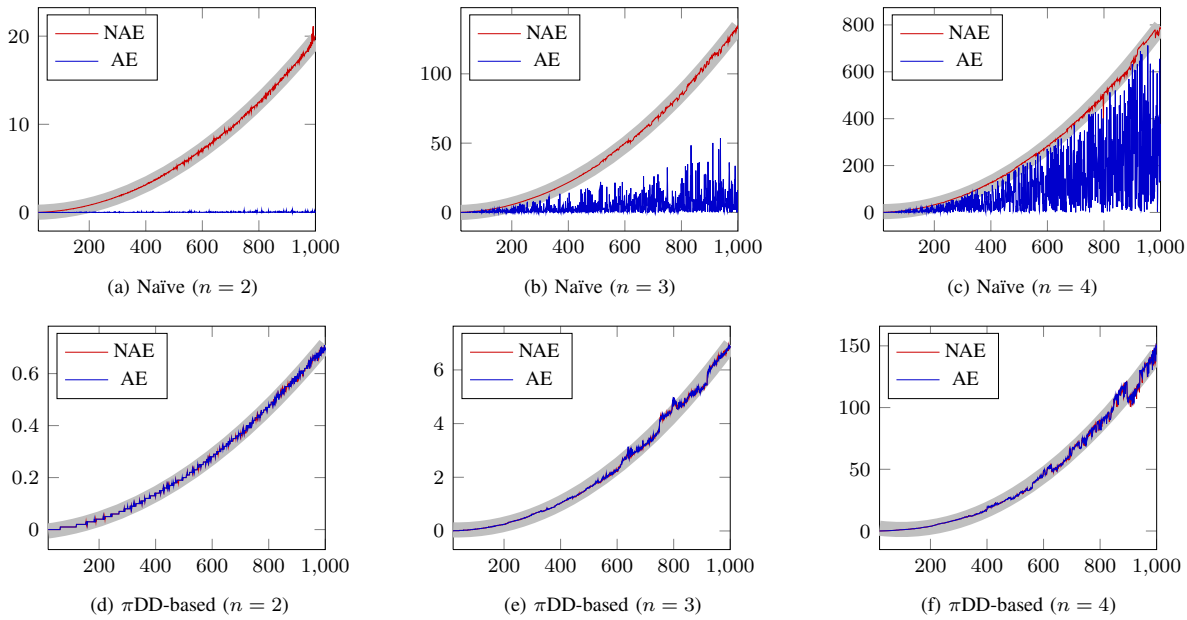
The results are represented graphically by means of Figs. 6 and 7. The x-axis and y-axis represent the number of gates d and the run-time in seconds, respectively. In Fig. 6 the approaches are directly compared and it can be seen that the πDD-based approach (blue line) clearly outperforms the naïve approach (red line). Only in the case of not almost-equal circuits on 2 lines, the πDD-based approach is slower than the naïve approach, but already for 3 lines, this relation has been turned and the πDD-based approach is significantly faster.

Fig. 7 displays the same results but represents the data in a different manner. Here, for each approach, i.e. naïve and πDD-based, and for each considered number of lines, both the results for the almost-equal and for the not almost-equal experiments are plotted into the same graph. The plots illustrate that the proposed πDD-based approach is robust with respect to the outcome of the algorithm, i.e. whether the circuit is almost-equal or is not almost-equal to the given function. In contrast, the naïve approach is not robust, since in case the function is almost-equal, the algorithm may stop earlier. The main differences between both algorithms is that the πDD-based approach requires the most computational effort in an initialization phase that is the same in both cases, whereas basically no initialization phase is required by the naïve approach and all computational effort is spent in the exhaustive search.

In order to estimate the complexity we have performed regression analysis on the results and determined that the best fit was given a polynomial regression fit of degree 2 which is also indicated by the thick gray plot in Fig. 7. Hence, also the complexity of both approaches is very likely to be quadratic with respect to the number of gates. In fact, from the naïve approach it can also be directly determined that the complexity must be quadratic by analyzing Algorithm N.

VII. Conclusions

In this paper, we have presented a debugging approach that makes use of πDDs. For this purpose, it is exploited that πDDs can represent a set of permutations and since reversible functions represent permutations, πDDs can be used to efficiently represent several reversible functions and circuits at once. As a result, faults in circuits, e.g. due to a missing gate or a missing control line, can efficiently be detected. Experiments have shown their applicability and demonstrate that they are more scalable than naïve approaches. Since the presented algorithm is based on permutations, it can be tailored to work with any reversible gate library. Further, also other debugging problems can be addressed.



(a) Naïve ($n = 2$) (b) Naïve ($n = 3$) (c) Naïve ($n = 4$)

(d) πDD-based ($n = 2$) (e) πDD-based ($n = 3$) (f) πDD-based ($n = 4$)

Fig. 7. Robustness analysis

In future work we want to consider such additional fault models and gate libraries. Also, we like to extract heuristics from the exact πDD problem formulation in order to accelerate the algorithm. This can e.g. be done by restricting the elements of the gate library.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. New York, NY, USA: Cambridge University Press, Oct. 2000.

[2] R. Cuykendall and D. R. Andersen, "Reversible optical computing circuits," *Optical Letters*, vol. 12, no. 7, pp. 542–544, Jul. 1987.

[3] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, Jul. 1961.

[4] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of Landauer's principle linking information and thermodynamics," *Nature*, vol. 483, pp. 187–189, Mar. 2012.

[5] A. D. Vos, *Reversible Computing - Fundamentals, Quantum Computing, and Applications*. Weinheim: Wiley, Oct. 2010.

[6] R. Wille, H. M. Le, G. W. Dueck, and D. Große, "Quantified Synthesis of Reversible Logic," in *Design, Automation and Test in Europe*. IEEE, Mar. 2008, pp. 1015–1020.

[7] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Design Automation Conference*. ACM, Jul. 2009, pp. 270–275.

[8] P. Kerntopf, "A New Heuristic Algorithm for Reversible Logic Synthesis," in *Design Automation Conference*, Jun. 2004, pp. 834–837.

[9] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition," in *Int'l Design and Test Workshop*, Dec. 2010, pp. 143–148.

[10] D. M. Miller and M. A. Thornton, "QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits," in *Int'l Symp. on Multiple-Valued Logic*. IEEE Computer Society, May 2006, p. 30.

[11] R. Wille, D. Große, D. M. Miller, and R. Drechsler, "Equivalence Checking of Reversible Circuits," in *Int'l Symp. on Multiple-Valued Logic*. IEEE Computer Society, May 2009, pp. 324–330.

[12] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler, "Synthesis of Reversible Circuits with Minimal Lines for Large Functions," in *Asia and South Pacific Design Automation Conference*, Jan. 2012.

[13] G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Quantum Circuit Simulation*. Dordrecht, Heidelberg, London, New York: Springer, Dec. 2009.

[14] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, and S.-Y. Kuo, "An XQDD-based verification method for quantum circuits," *IEICE Transactions*, vol. 91-A, no. 2, pp. 584–594, 2008.

[15] S. Minato, "πDD: A New Decision Diagram for Efficient Problem Solving in Permutation Space," in *Theory and Applications of Satisfiability Testing*, Jun. 2011, pp. 90–104.

[16] M. Soeken, R. Wille, S. Minato, and R. Drechsler, "Using πDDs in the Design for Reversible Circuits," 2013, selected Submissions from the Reversible Computation Workshop 2012, in press.

[17] J. P. Hayes, I. Polian, and B. Becker, "Testing for missing-gate faults in reversible circuits," in *Asian Test Symposium*, Nov. 2004, pp. 100–105.

[18] H. Zhang, R. Wille, and R. Drechsler, "Improved Fault Diagnosis for Reversible Circuits," in *Asian Test Symposium*, Nov. 2011, pp. 207–212.

[19] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler, "Debugging reversible circuits," *Integration*, vol. 44, no. 1, pp. 51–61, Jan. 2011.

[20] S. Minato, "Zero-Supressed BDDs for Set Manipulation in Combinational Problems," in *Design Automation Conference*, Jun. 1993, pp. 272–277.

[21] A. Muthukrishnan and C. R. Stroud, "Multivalued logic gates for quantum computation," *Phys. Rev. A*, vol. 62, p. 052309, Oct. 2000.

[22] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "RevKit: A Toolkit for Reversible Circuit Design," in *Workshop on Reversible Computation*, Jul. 2010, pp. 69–72, RevKit is available at www.revkit.org.