Debugging of Toffoli Networks

Robert Wille[1] Daniel Große[1] Stefan Frehse[1] Gerhard W. Dueck[2] Rolf Drechsler[1]

[1]*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany*
{rwille,grosse,sfrehse,drechsle}@informatik.uni-bremen.de
[2]*Faculty of Computer Science, University of New Brunswick, Fredericton, Canada*
gdueck@unb.ca

***Abstract*—Intensive research is performed to find post-CMOS technologies. A very promising direction based on reversible logic are quantum computers. While in the domain of reversible logic synthesis, testing, and verification have been investigated, debugging of reversible circuits has not yet been considered. The goal of debugging is to determine gates of an erroneous circuit that explain the observed incorrect behavior.**

In this paper we propose the first approach for automatic debugging of reversible Toffoli networks. Our method uses a formulation for the debugging problem based on Boolean satisfiability. We show the differences to classical (irreversible) debugging and present theoretical results. These are used to speed-up the debugging approach as well as to improve the resulting quality. Our method is able to find and to correct single errors automatically.

I. Introduction

As predicted by the well-known Moore's law the electronic industry celebrated great achievements in the last 45 years. In 1965 Moore postulated that the number of transistors in integrated circuits doubles every 18 months. However, due to this exponential growth and hence the continuous shrinking of the transistor sizes, fundamental physical limits will be reached, i.e. the transistor itself will approach the atomic scale. In addition the increasing power consumption of electronic devices becomes a serious problem. Therefore reversible logic attracted considerable research attention. Since reversible computation is information-lossless, i.e. data is transformed without erasing any of the original information, power dissipation is reduced or even eliminated [1]. Furthermore, reversible computation is the basis for quantum computing [2]. In this domain it has been shown that some important problems such as factorization can be solved exponentially faster than by currently known methods.

The idea of reversible computing can be traced back to Landauer [1] and Bennett [3], and has been further refined by Toffoli [4]. Since synthesis of reversible logic circuits differs significantly from classical logic circuits – fan-out and feedback are forbidden – a strong focus on synthesis approaches emerged (see e.g. [5], [6], [7], [8], [9], [10], [11], [12]). Furthermore, simulation [13], [14], optimization [15], [16], testing [17], [18], [19], and verification [20], [21] have also been investigated. However, to the best of our knowledge the problem of *debugging* has not been considered yet. While methods for simulation, testing, or verification can only be used to detect the existence of errors, there is no support to locate the source of an error. For instance, errors may result from bugs in synthesis as well as optimization tools, or manual modifications of the circuit.

In this paper we propose the first approach to automatically determine error candidates explaining the erroneous behavior of a reversible circuit. More precisely, given an erroneous circuit and a set of counterexamples describing the error(s), our approach returns sets of gates, whose replacements with other gates fix the counterexamples. For classical circuits, debugging has been studied intensely (see e.g. [22], [23], [24], [25], [26], [27]). Regarding the quality, methods based on *Boolean satisfiability* (SAT) have shown to be efficient and robust. The proposed debugging approach makes also use of SAT. However, we show that utilizing the SAT formulation from [25] does not lead to the desired results, i.e. already for single errors each gate of a reversible circuit becomes an error candidate – obviously a very poor solution for the debugging problem. Hence, we devise a SAT-based formulation for debugging of reversible circuits which takes the ideas of [25] and integrates the properties of reversible gates. As a result, the reported set of error candidates is significantly reduced.

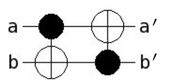
Besides the automatic debugging approach we also present theoretical results. For a restricted error model, i.e. assuming single missing control line errors, a significant number of gates can be excluded from being error candidates by virtue of the given number of counterexamples. Thus, the size of the SAT instance is reduced or in the best case no SAT call is needed leading to a speed-up of the overall debugging process.

Furthermore, we show that it can be decided for single errors whether an error candidate is an error position, i.e. replacing the respective gate not only fixes the counterexamples but also ensures that the correct behavior of the circuit remains the same. This becomes possible, since at each gate of a reversible circuit the erroneous behavior for a single error can be fixed while preserving the overall functionality. This theoretical result is used for a tight integration of debugging and fixing. Thus, the error can be located to correct for instance the optimization or synthesis tool and additionally a fix for the circuit is provided. In summary, due to the tight integration a higher quality of the results is achieved.

The remainder of this paper is structured as follows: To keep the paper self-contained, reversible logic, the debugging problem, and the state-of-the-art SAT-based approach for irreversible circuits are briefly reviewed in Section II. Section III introduces SAT-based debugging for reversible logic, describes improvements, and discusses observations for multiple errors. The integration of automatic fixing and debugging is described in Section IV. Finally, Section V gives experimental evaluations of the proposed methods while Section VI concludes the paper.

a	b	a'	b'
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

(a) Truth Table



(b) Toffoli Circuit

Fig. 1. Toffoli Circuit

II. Background

A. Reversible Logic

Reversible logic realizes functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$, that map each possible input vector to a unique output vector. In other words bijections are represented. Since fan-out and feed-back are not allowed, each reversible circuit G consists of a cascade of reversible gates g_i, i.e. $G = g_0 g_1 \cdots g_{d-1}$ where d is the number of gates. The inverse G^{-1} of a reversible network G (representing the function f^{-1}) can be obtained by reversing the order of the gates of G.

In the recent past, many reversible gates have been studied. *Multiple control Toffoli gates* [4] are widely used: Let $X := \{x_1, \dots, x_n\}$ be the set of domain variables. A multiple control Toffoli gate has the form $TOF(C, t)$, where $C = \{x_{i_1}, \dots, x_{i_k}\} \subset X$ is the set of *control lines* and $t = \{x_j\}$ with $C \cap t = \emptyset$ is the *target line*. The gate maps $(x_1, \dots, x_n)$ to $(x_1, \dots, x_{j-1}, x_j \oplus x_{i_1} \dots x_{i_k}, x_{j+1}, \dots, x_n)$. If no control lines are given (C is empty), then the target line is inverted, i.e. the input vector of the gate is mapped to $(x_1, \dots, x_{j-1}, x_j \oplus 1, x_{j+1}, \dots, x_n)$.[1] In the following, we refer to multiple control Toffoli gates for brevity as Toffoli gates.

As an example Fig. 1 shows a Toffoli circuit representing the function specified by the given truth table. The network is drawn in standard notation (see e.g. [2]) and includes $d = 2$ gates.

In the rest of this paper we only consider reversible circuits consisting of Toffoli gates.

B. Debugging Problem

As their classic counterparts, reversible circuits may contain errors because of bugs in synthesis as well as optimization tools, or manual modifications, respectively. For reversible logic these include all types of errors like missing or additional control lines, misplaced target lines, missing gates, etc. They can be detected with counterexamples, i.e. a set of input assignments leading to wrong values at the output of the circuit.

Similar to irreversible logic, the goal of debugging is to identify gates in the network that explain the erroneous behavior. More precisely, given an erroneous circuit G and a set of counterexamples, a set of *error candidates* is returned. An error candidate is a set of gates g_i of G that can be replaced by other gates such that for each counterexample the correct output values result. The size of an error candidate is given by the number of gates (later denoted by k).

Determining error candidates is crucial to understand the erroneous behaviour and to fix the respective source of the error. However, due to the increasing sizes of the circuits this is only manageable with automatic techniques.

[1]The multiple control Toffoli gate with no control line (with one control line) is also called NOT (CNOT) gate.

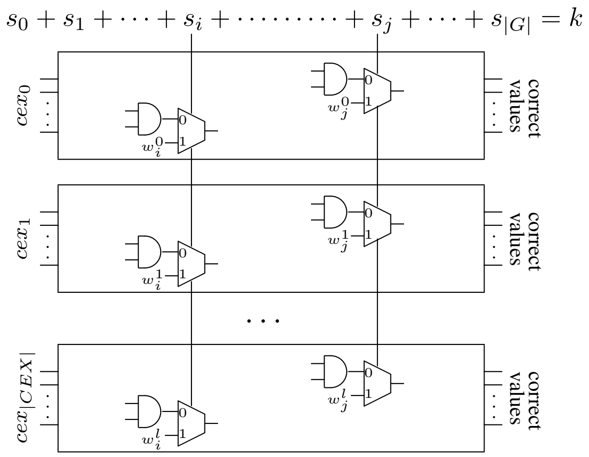


Fig. 2. SAT-based Debugging Approach

C. Debugging of Irreversible Circuits

Methods based on *Boolean satisfiability* (SAT) have been demonstrated to be very effective for debugging irreversible logic [25]. Here, the erroneous circuit and a set of counterexamples are used to create a SAT instance. Solving this instance using well engineered SAT solvers (see e.g. [28]) returns solutions from which the desired set of error candidates can be determined.

The general structure for the debugging problem that is converted to a SAT instance is shown in Fig. 2. For each counterexample, a copy of the circuit is created, whose inputs are assigned to values provided by the counterexamples (denoted by $cex_0, \dots, cex_{|CEX|}$). The outputs are assigned to the correct values. Furthermore, each gate g_i is extended by additional logic: A multiplexor with select line s_i is added. If s_i is assigned to 0, then the output value of gate g_i is passed through, i.e. the gate works as usual. Otherwise (if $s_i = 1$), an unrestricted value is used (available via a new free variable w). Therefore, if g_i is an erroneous gate, the SAT solver can assign $s_i = 1$ and choose the correct gate value to enable correct values at the output of the circuit.

As depicted in Fig. 2, the same select value s_i is used for a gate g_i with respect to all duplications. This ensures that free values of the respective output signals are only used, if circuit outputs are corrected for *all* counterexamples. Furthermore, the total number of selects s_i set to 1 is limited to k. Starting with $k = 1$, k is iteratively increased until the SAT instance becomes satisfiable. Then, each satisfying assignment yields an error candidate of size k. All gates with s_i set to 1 are contained in the error candidate. By performing *all solution SAT* (i.e. determining all solutions from the instance), all error candidates are calculated.

For a more detailed description of SAT-based debugging, we refer to [25].

III. Debugging of Reversible Circuits

In this section the SAT-based debugging approach for reversible circuits is presented. Also improvements for missing control errors are given.

A. Formulation for Reversible Circuits

The debugging approach described in Section II-C has been demonstrated to be very effective for determining error candidates of irreversible circuits. However, we show that the formulation from [25] gives very poor results here, since the

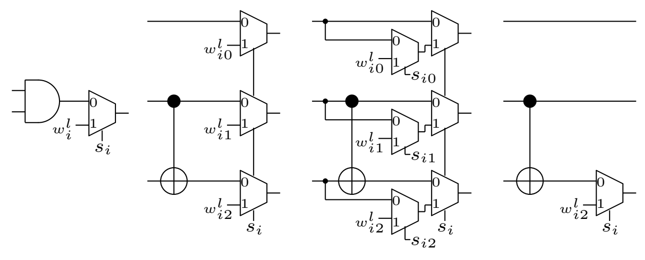


Fig. 3. Additional Logic for Debugging

properties of reversible circuits are not incorporated. In classic debugging only one-output gates for and, or, xor, etc. are considered [25]. Therefore, only a single multiplexor as shown in Fig. 3(a) is added to express whether a gate g_i can become part of an error candidate. In contrast, for reversible circuits each gate has n outputs in total. Thus, in a straight-forward application of the classical debugging approach, n multiplexors with the same select s_i are added to the debugging formulation (see Fig. 3(b)). However, as the following lemma shows the debugging results are meaningless.

Lemma 1. *Let G be an erroneous circuit. Using the classical debugging approach with the additional logic formulation depicted in Fig. 3(b) and an arbitrary set of counterexamples, for each gate g_i of G a satisfying solution with $s_i = 1$ exists. Thus, all gates are returned as error candidates.*

Proof: Let $G = G_1 g_i G_2$ be an erroneous circuit with a set of counterexamples. A gate g_i is determined as error candidate, if a satisfying assignment $s_i = 1$ exists such that the correct output value for each counterexample can be calculated. Using the additional logic formulation depicted in Fig. 3(b), assigning $s_i = 1$ enables unrestricted values for *all* n outputs of the gate g_i. To obtain the values leading to the correct circuit output, just G_2^{-1} has to be applied. This can be performed for each gate g_i of G. ■

Lemma 1 shows that the existing SAT-based debugging formulation for irreversible circuits is too general for reversible circuits. In fact, assigning s_i to 1 should imply that the output values of gate g_i cannot be chosen arbitrary, but with respect to the functionality of Toffoli gates. The two main properties of Toffoli gates are:

- At most one line (the target line) is inverted if the respective control lines are assigned to 1 and
- all remaining lines are passed through.

A new formulation respecting these properties is given in Fig. 3(c): For each output of a gate g_i, a second multiplexor with a new select s_{ib} is added ($0 \leq b < n$). By restricting $s_{i0} + \cdots + s_{in-1}$ to 1 we ensure that the value of at most one line is modified, if s_i is set to 1. All remaining values are passing through. Thereby, the multi-output behavior including the reversibility is reflected in the debugging formulation.[2]

Nevertheless, as in irreversible debugging the resulting error candidates are still an approximation of the error positions as shown by the following example.

Example 1. *Consider the circuit realization of function* 3_17 *with an injected missing control error at gate g_5 depicted in Fig. 4. The function as well as the circuit can be found in [29].*

[2]Note that using multiplexors obviously makes the considered circuits non-reversible. However, this formulation is only used as a logic encoding of the debugging problem.

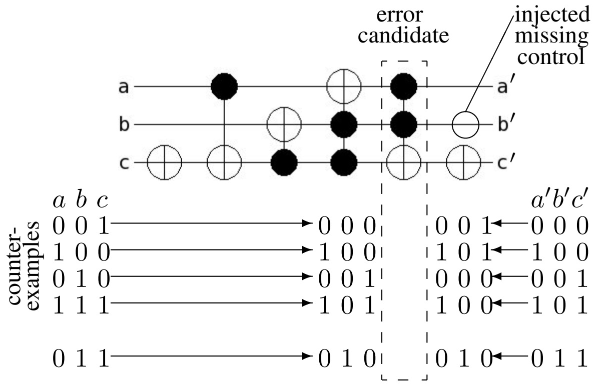


Fig. 4. Circuit with Single Error

The missing control error leads to four counterexamples given in the first four rows of Fig. 4. Besides g_5, the proposed debugging formulation also returns g_4 as an error candidate (marked by a dashed rectangle). This is, because replacing g_4 with a NOT gate at line c leads to correct output values for all counterexamples as shown in the first four rows. However, g_4 is not an error position, since for the NOT gate replacement a wrong output (wrt. the specification of 3_17*) using* 011 *as input is computed as can be seen in the fifth line of the figure. Thus, only g_5 is a error position where a fix for all counterexamples is possible and the correct behavior of the circuit remains the same.*

Recall that our approach determines all error candidates. Hence, for the example this means that gate g_5 is not missed. In fact, the user only needs to consider 2 out of 6 gates to locate the error. In general, the experiments in Section V clearly demonstrate that a significant reduction of the potential error candidates is achieved by the proposed debugging approach.

In the next section for a restriction to a certain kind of errors, a simplification and a theoretical result that builds the basis for an improvement of the debugging approach is presented.

B. Improvements for Control Line Errors

The proposed debugging formulation needs a substantial amount of additional logic. This can be reduced, if a restricted error-model is assumed. In this section we describe a simpler debugging formulation for control line errors. This simplification leads to a faster calculation of error candidates and thus should be applied, if the source of an error can be limited to this type of errors.

Control line errors include both, missing as well as additional control lines in a Toffoli gate. They may occur, when e.g. optimization approaches or the designer himself manipulates control lines of Toffoli gates. In particular, deleting control lines is used by optimization approaches (see e.g. [15]) since they reduce the quantum costs for the considered circuit. Errors caused by deleting control lines can be seen as missing control errors, too.

Since missing control errors (as well as additional control errors) only affect the target line of a Toffoli gate, the debugging formulation can be simplified to the one shown in Fig. 3(d). Here, multiplexors are only added for the target line of each gate. If a gate g_i includes a control line error, only the value of the target line can be erroneous. By assigning s_i to 1, the SAT solver can choose the correct value and thus enable correct outputs. In this case, g_i becomes an element of an error candidate.

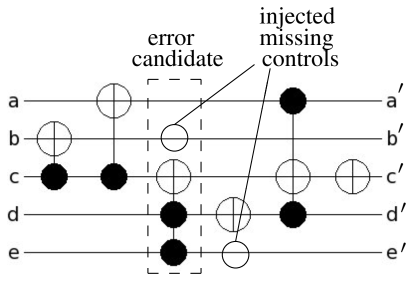


Fig. 5. Circuit with Multiple Error

Furthermore, if a single missing control error is assumed, the following holds:

Lemma 2. *Let G be a reversible circuit with a single missing control error and $|CEX|$ the total number of counterexamples for this error. Then, the erroneous gate includes $c = n - 1 - \log_2 |CEX|$ control lines.*

Proof: Let G be a reversible circuit with a missing control error in gate g_i containing c control lines. To detect the erroneous behaviour, (1) all control lines of g_i have to be assigned to 1 and (2) another line of g_i (the missing control line) has to be assigned to 0. Due to the reversibility, these values can be propagated to the inputs of the circuit leading to $|CEX| = 2^{n-c-1}$ different counterexamples in total. From this, one can conclude

$$\begin{aligned} |CEX| &= 2^{n-c-1} \\ \log_2 |CEX| &= \log_2 2^{n-c-1} \\ \log_2 |CEX| &= n-c-1 \\ c &= n-1-\log_2 |CEX|. \end{aligned}$$

■

Exploiting Lemma 2, the number of gates that have to be considered can be reduced significantly. In some cases, this reduction already leads to a single gate and therewith to the desired error position (see Section V). But even if the automatic debugging approach has to be invoked, improvements can be observed since additional logic as depicted in Fig. 3(d) only has to be added to gates containing exactly $c = n - 1 - \log_2 |CEX|$ control lines.

C. Debugging Multiple Errors

In general, the proposed formulation can also be used for multiple errors (i.e. for $k > 1$). However, the case of multiple errors is more complicated. All solutions for a concrete k (that have been found incrementally by starting from $k = 1$) only guarantee that the counterexamples are corrected, but the correct behavior of the circuit may be changed. This is illustrated in the following example:

Example 2. *Consider the circuit realization of the function* alu *depicted in Fig. 5. In this circuit a multiple error has been injected at gate g_2 and at gate g_3, respectively. If the proposed debugging approach is executed, for $k = 1$ exactly one solution (g_2) is returned. However, by exhaustive enumeration it has been checked, that no replacement for gate g_2 exists, such that the circuit realizes the function specification. In fact, an adequate replacement of gate g_2 only fixes the counterexamples.*

A similar observation has also been made for irreversible circuits. In first experiments (not described in this paper) this case has not been found often. Obviously, this can be handled by increasing k and calculating the error candidates for $k+1$. A detailed consideration of such kinds of multiple errors is left for future work.

IV. Automatic Fixing

This section introduces an important result for automatically fixing single and multiple errors. Furthermore, an approach for single errors is proposed that allows to compute the error positions for a circuit. Replacing a gate given by an error position not only fixes all counterexamples but also ensures that the correct behaviour of the circuit remains the same.

For a given specification F (usually provided as a reference circuit used to obtain the counterexamples) an erroneous circuit G can be fixed by applying the following lemma.

Lemma 3. *Let F be an error-free representation of a reversible function and $G = G_1 g_i G_2$ be an erroneous realization/optimization of F.*[3] *Then G can be fixed by replacing any gate g_i of G with a cascade of gates $G_i^{fix} = G_1^{-1} F G_2^{-1}$.*

Proof: Since $G^{-1}G$ realizes the identity function, it holds:

$$\begin{aligned} G_1 G_i^{fix} G_2 &= F & \Leftrightarrow \\ G_1^{-1} G_1 G_i^{fix} G_2 G_2^{-1} &= G_1^{-1} F G_2^{-1} & \Leftrightarrow \\ G_i^{fix} &= G_1^{-1} F G_2^{-1} \end{aligned}$$

Thus, replacing g_i with G_i^{fix} fixes the erroneous circuit G. ■

At a first glance, applying this lemma for fixing the erroneous circuit G, i.e. replacing g_i by G_i^{fix} (which includes the circuit F), leads to a larger circuit than F itself. However, almost always G_i^{fix} can be simplified to a cascade of just a few gates. Moreover, in case of a single error there exists at least one gate g_j whose replacement G_j^{fix} consists of a single gate. To determine whether G_i^{fix} is equivalent to a single gate, all possible input combinations are simulated and the resulting specification is matched against Toffoli gates. To find a fix where G_i^{fix} is equivalent to a single gate, for all gates g_i ($0 \leq i < d$) G_i^{fix} has to be computed and checked thereafter. For larger circuits such a straight-forward method is not feasible. Therefore, by integrating fixing and debugging the check has to be carried out only for the error candidates – a significantly smaller number of gates. Furthermore, in case of single errors the application of the lemma to each error candidate yields whether the respective error candidate is an error position or not – a very strong result.

Please note that more than one error position is possible. Consider the circuit given in Fig. 6. Here, a single missing control error has been injected at gate g_1. However, two valid fixes are possible.

In summary, by integrating debugging and fixing the runtime as well as the quality of the results are improved. In

[3]Note that G can also contain multiple errors.

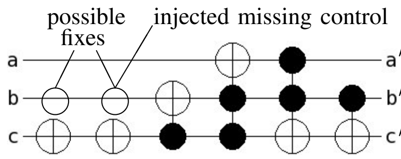


Fig. 6. Erroneous Circuit with Fixes

TABLE I
DEBUGGING (DETERMINING ERROR CANDIDATES)

(a) Arbitrary errors

				REVDBG		
BENCHMARK	d	n	$\|CEX\|$	CAND.	TIME	RED.
3_17	6	3	4(2)	2	<0.01s	66.7%
4_49	16	4	8(2)	2	<0.01s	87.5%
4gt4	6	5	4(2)	1	<0.01s	83.3%
4mod5	9	5	16(1)	2	<0.01s	77.8%
alu	6	5	32(3)	1	<0.01s	83.3%
ham3	5	3	4(2)	2	<0.01s	60.0%
ham7	23	7	16(1)	1	<0.01s	95.7%
hwb4	17	4	8(2)	1	<0.01s	94.1%
hwb5	55	5	8(2)	4	0.03s	92.7%
hwb6	126	6	16(1)	16	0.13s	87.3%
hwb7	289	7	16(1)	14	1.04s	95.2%
hwb8	637	8	4(2)	2	10.82s	99.7%
hwb9	1544	9	30(3)	1	95.32s	99.9%
plus63mod4096	429	12	32(3)	17	18.97s	96.0%
plus63mod8192	492	13	8(2)	4	19.3s	99.2%
plus127mod8192	910	13	16(1)	98	64.19s	89.2%
urf1	11554	9	128(12)	–	Ab.	–
urf2	5030	8	64(6)	1	1911.54s	99.9%
urf3	26468	10	256(25)	–	Ab.	–

(b) Missing control errors

				REVDBG		TARGET LINES ONLY		$\|CEX\|$-BASED Reduction		
BENCHMARK	d	n	$\|CEX\|$	CAND.	TIME	CAND.	TIME	CAND.	TIME	RED.
3_17	6	3	2(2)	2	<0.01s	2	<0.01s	4/2	<0.01s	66.7%
4_49	16	4	8(2)	10	<0.01s	5	<0.01s	2/1	<0.01s	93.8%
4gt4	6	5	2(2)	3	<0.01s	3	<0.01s	2/2	<0.01s	66.7%
4mod5	9	5	8(2)	6	<0.01s	4	<0.01s	4/1	<0.01s	88.9%
alu	6	5	16(1)	5	<0.01s	1	<0.01s	2/1	<0.01s	83.3%
ham3	5	3	4(2)	3	<0.01s	1	<0.01s	1/1	<0.01s	80.0%
ham7	23	7	32(3)	5	0.02s	1	<0.01s	17/1	<0.01s	95.7%
hwb4	17	4	8(2)	4	<0.01s	1	<0.01s	1/1	<0.01s	94.1%
hwb5	55	5	8(2)	6	0.03s	2	<0.01s	25/1	<0.01s	98.2%
hwb6	126	6	16(1)	7	0.06s	1	0.01s	20/1	0.01s	99.2%
hwb7	289	7	32(3)	250	7.70s	42	0.17s	28/5	0.05s	98.3%
hwb8	637	8	8(2)	34	13.50s	8	0.39s	167/1	0.11s	99.8%
hwb9	1544	9	16(1)	68	71.71s	9	0.58s	443/1	0.30s	99.9%
plus63mod4096	429	12	16(1)	296	18.18s	49	0.17s	84/12	0.04s	97.2%
plus63mod8192	492	13	512(51)	101	605.80s	15	5.08s	43/4	1.54s	99.2%
plus127mod8192	910	13	64(6)	237	285.09s	31	1.81s	128/8	0.54s	99.1%
urf1	11554	9	128(12)	–	Ab.	2	36.40s	1/1	<0.01s	99.9%
urf2	5030	8	64(6)	12	2202.43s	4	8.38s	1/1	<0.01s	99.9%
urf3	26468	10	256(25)	–	Ab.	–	Ab.	1/1	<0.01s	99.9%

particular, in case of single errors we can guarantee to locate the source of an error *and* provide a fix for each error position consisting of a single gate.

V. EXPERIMENTAL EVALUATION

The proposed methods have been implemented in C++ and evaluated on a set of reversible circuits taken from [29]. Due to page limitation only the major results of this evaluation are documented in this section. For solving the respective instances the SAT solver MiniSat [28] was used. The documented run-times include the times for instance generation and solving. All experiments have been carried out on an AMD Athlon 3500+ with 1 GB of main memory. The timeout (denoted by Ab.) was set to 5000 CPU seconds.

A. Automatic Debugging of Errors

In a first series of experiments the proposed debugging approaches for error candidate determination are considered. To this end, single arbitrary errors and single missing control errors have been randomly injected to circuits taken from [29]. More precisely, a gate has been replaced with another gate and a control line has been removed, respectively. Counterexamples describing the errors were generated using a SAT-based equivalence checker.

For debugging arbitrary errors, the approach proposed in Section III-A was used (denoted by REVDBG). For missing control errors additionally the improvements of Section III-B are possible, namely the consideration of target lines only (denoted by TARGET LINES ONLY) and the application of Lemma 2 (denoted by $|CEX|$-BASED REDUCTION).

The results are summarized in Table I. Column BENCHMARK gives the circuit name. Column d, column n, and column $|CEX|$ give the number of gates, the number of lines, and the number of counterexamples (the number within the brackets denotes the number of counterexamples for which the circuit has been duplicated). Furthermore, for each approach the number of obtained error candidates (denoted by CAND.) and the overall run-time in CPU seconds (denoted by TIME) are provided. Column CAND. for $|CEX|$-BASED REDUCTION includes two values. The first denotes the remaining gates after applying Lemma 2, the second gives the final number of error candidates after running the SAT-based debugging approach. Finally, column RED. lists the best reduction obtained by our approaches, i.e. the percentage of gates that are identified as non-relevant (meaning the error is *not* located at these gates).

As shown in the table, a significant amount of gates can be automatically identified as non-relevant for debugging the error. Reductions of at least 66.7% – for larger circuits of more than 90% – are achieved. As an example, for the arbitrary error in circuit *hwb9* with 1544 gates a single error candidate is obtained in less than 100 CPU seconds. The quality of the resulting set of error candidates often depends on the strategy used. For example, to identify the missing control error in circuit *hwb7* still 250 (out of 289) error candidates have to be considered after applying the REVDBG approach. Here, restricting the error model and using the improvements not only leads to a speed-up, but also to a smaller number of error candidates. In particular, this is effective for the circuits *hwb4* and *urf1*. Here, just by applying Lemma 2 the set of error candidates is reduced to the single erroneous gate.

B. Integrating Fixing of Errors

The error candidates obtained by the debugging approaches can be used to fix the counterexamples, but it is not ensured that they also represent error positions. However, as described in Section IV the integration of debugging and fixing for single errors gives the error positions. Thus, we also provide experiments for the integrated approach.

The results are shown in Table II.[4] Here, either the exhaustive approach introduced in Section IV has been applied (denoted by EXHAUST.) or previously the gates are reduced using the appropriate debugging method, i.e. REVDBG for arbitrary errors or $|CEX|$-BASED REDUCTION for missing control errors (both denoted by DBG+FIX in Table II). Again, BENCHMARK, d, n, and CAND. denote the name, the number of gates, the number of lines, and the number of obtained error candidates of each benchmark, respectively. The documented overall run-times (including the run-time for debugging when applied) is given in column TIME. The number of the resulting error positions is provided in column POS.

[4]Note that only circuits, where more than one error candidate is returned by our approach, have to be considered to determine the error positions. However, to demonstrate the performance of fixing, results for all circuits are given in Table II.

TABLE II
FIXING (DETERMINING ERROR POSITIONS)

BENCHMARK	d	n	EXHAUST.	DBG+FIX		
			TIME	TIME	CAND.	POS.
ARBITRARY ERRORS						
3_17	6	3	<0.01s	<0.01s	2	1
4_49	16	4	0.01s	<0.01s	2	2
4gt4	6	5	<0.01s	<0.01s	1	1
4mod5	9	5	<0.01s	<0.01s	2	1
alu	6	5	0.01s	<0.01s	1	1
ham3	5	3	<0.01s	<0.01s	2	1
ham7	23	7	0.22s	0.01s	1	1
hwb4	17	4	0.01s	0.01s	1	1
hwb5	55	5	0.27s	0.06s	4	1
hwb6	126	6	3.14s	0.51s	16	1
hwb7	289	7	33.35s	2.67s	14	1
hwb8	637	8	347.31s	11.90s	2	1
hwb9	1544	9	4341.01s	98.32s	1	1
plus63mod4096	429	12	Ab.	143.89s	17	1
plus63mod8192	492	13	Ab.	90.82s	4	1
plus127mod8192	910	13	Ab.	3346.03s	98	1
urf1	11554	9	Ab.	Ab.	–	–
urf2	5030	8	Ab.	1917.28s	1	1
urf3	26468	10	Ab.	Ab.	–	–
MISSING CONTROL ERRORS						
3_17	6	3	<0.01s	<0.01s	2	2
4_49	16	4	<0.01s	<0.01s	1	1
4gt4	6	5	<0.01s	<0.01s	2	1
4mod5	9	5	0.01s	<0.01s	1	1
alu	6	5	<0.01s	<0.01s	1	1
ham3	5	3	<0.01s	<0.01s	1	1
ham7	23	7	0.21s	0.01s	1	1
hwb4	17	4	0.04s	<0.01s	1	1
hwb5	55	5	0.29s	0.01s	1	1
hwb6	126	6	3.08s	0.03s	1	1
hwb7	289	7	33.31s	0.63s	5	1
hwb8	637	8	347.05s	0.67s	1	1
hwb9	1544	9	4621.41s	3.33s	1	1
plus63mod4096	429	12	3352.62s	87.97s	12	1
plus63mod8192	492	13	Ab.	73.16s	4	1
plus127mod8192	910	13	Ab.	270.79s	8	1
urf1	11554	9	Ab.	28.98s	1	1
urf2	5030	8	Ab.	5.97s	1	1
urf3	26468	10	Ab.	141.95s	1	1

As expected the exhaustive approach (iterating over all gates and applying Lemma 3) is not feasible for large designs. In contrast, integrating debugging and fixing enables efficient determination of both, error positions and corrections. Automatic debugging and fixing is possible for circuits with up to $5,000$ gates when arbitrary errors are considered. If additionally a restricted error model is assumed, circuits with more than $26,000$ gates can be handled due to the proposed improvements.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed first methods for automatic debugging errors in Toffoli circuits. We have shown, that a one-to-one adaption of classical approaches does not lead to the desired results. Hence we devised a formulation that integrates the properties of reversible gates. Furthermore, improvements for (missing) control errors as well as methods for automatic fixing of errors have been introduced. In particular, the latter can be utilized for single errors to increase the quality of the obtained results.

However, debugging of reversible circuits is still at the beginning. Thus, several aspects are left for future work. In particular, multiple errors have to be considered in detail. First observations regarding this have been given in Section III-C. Additionally, different error models (similar to the ones identified in [19] for reversible testing) have to be studied.

REFERENCES

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Dev.*, vol. 5, p. 183, 1961.

[2] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[3] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev*, vol. 17, no. 6, pp. 525–532, 1973.

[4] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, W. de Bakker and J. van Leeuwen, Eds. Springer, 1980, p. 632, technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[5] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 710–722, 2003.

[6] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conf.*, 2003, pp. 318–323.

[7] W. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis." *IEEE Trans. on CAD*, vol. 25, no. 9, pp. 1652–1663, 2006.

[8] P. Gupta, A. Agrawal, and N. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2317–2330, 2006.

[9] R. Wille and D. Große, "Fast exact Toffoli network synthesis of reversible logic," in *Int'l Conf. on CAD*, 2007, pp. 60–64.

[10] M. Saeedi, M. Sedighi, and M. S. Zamani, "A novel synthesis algorithm for reversible circuits," in *Int'l Conf. on CAD*, 2007, pp. 65–68.

[11] R. Wille, H. M. Le, G. W. Dueck, and D. Große, "Quantified synthesis of reversible logic," in *Design, Automation and Test in Europe*, 2008, pp. 1015–1020.

[12] R. Wille, D. Große, G. W. Dueck, and R. Drechsler, "Reversible logic synthesis with output permutation," in *VLSI Design*, 2009.

[13] G. F. Viamontes, M. Rajagopalan, I. L. Markov, and J. P. Hayes, "Gate-level simulation of quantum circuits," in *ASP Design Automation Conf.*, 2003, pp. 295–301.

[14] D. Goodman, M. A. Thornton, D. Y. Feinstein, and D. M. Miller, "Quantum logic circuit simulation based on the QMDD data structure," in *Int'l Reed-Muller Workshop*, 2007.

[15] J. Zhong and J. Muzio, "Using crosspoint faults in simplifying toffoli networks," in *IEEE North-East Workshop on Circuits and Systems*, 2006, pp. 129–132.

[16] D. Y. Feinstein, M. A. Thornton, and D. M. Miller, "Partially redundant logic detection using symbolic equivalence checking in reversible and irreversible logic circuits," in *Design, Automation and Test in Europe*, 2008, pp. 1378–1381.

[17] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault testing for reversible circuits," *IEEE Trans. on CAD*, vol. 23, no. 8, pp. 1220–1230, 2004.

[18] M. Perkowski, J. Biamonte, and M. Lukac, "Test generation and fault localization for quantum circuits," in *Int'l Symp. on Multi-Valued Logic*, 2005, pp. 62–68.

[19] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, "A family of logical fault models for reversible circuits," in *Asian Test Symp.*, 2005, pp. 422–427.

[20] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007, pp. 69–74.

[21] 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.

[22] C.-C. Lin, K.-C. Chen, S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng, "Logic synthesis for engineering change," in *Design Automation Conf.*, 1995, pp. 647–651.

[23] A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. on CAD*, vol. 18, no. 12, pp. 1803–1816, 1999.

[24] D. W. Hoffmann and T. Kropf, "Efficient design error correction of digital circuits," in *Int'l Conf. on Comp. Design*, 2000, pp. 465–472.

[25] A. Smith, A. G. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[26] M. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler, "Post-verification debugging of hierarchical designs," in *Int'l Conf. on CAD*, 2005, pp. 871–876.

[27] G. Fey, S. Safarpour, A. G. Veneris, and R. Drechsler, "On the relation between simulation-based and SAT-based diagnosis," in *Design, Automation and Test in Europe*, 2006, pp. 1139–1144.

[28] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[29] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, pp. 220–225, RevLib is available at http://www.revlib.org.