Enhancing Debugging of Multiple Missing Control Errors in Reversible Logic

Jean Christoph Jung Stefan Frehse Robert Wille Rolf Drechsler

Institute for Computer Science
28359 Bremen, Germany
{jeanjung,sfrehse,rwille,drechsle}@informatik.uni-bremen.de

ABSTRACT

Researchers are looking for alternatives to overcome the upcoming limits of conventional hardware technologies. Reversible logic thereby established itself as a promising direction so that several methods for synthesis, verification, and testing of reversible circuits have already been proposed. However, also methods for debugging, i.e., to determine error candidates in case of a failed verification, are required to complete the design flow. Even if first approaches have already been proposed, debugging of reversible circuits still is in the beginning. In this paper, we present an alternative method to automatically debug reversible circuits. We thereby focus on missing control errors – an established error model in the design of reversible circuits. A new notion of an error candidate is proposed that relies on the observation of a necessary condition for error locations in reversible circuits. Using this notion, a set of error candidates is obtained that differs from the error candidates returned by previous methods. Thus, combining the approaches enhances the overall debugging flow. Experimental results demonstrate that a higher accuracy is obtained in significantly shorter run-time.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: LOGIC DESIGN—*Design Aids*

General Terms

Design, Verification

Keywords

Debugging, Reversible Logic, Boolean Satisfiability (SAT)

1. INTRODUCTION

The exponential growth of transistor density in integrated circuits (according to Moore's Law) will reach its limits in the future. When the size of single transistors reaches the atomic scale, the continuously shrinking feature sizes of digital circuits are coming to an end. Moreover, the increasing power dissipation of electronic devices is a major barrier in the development of smaller and more powerful computer chips already today. Because of this, researchers are looking for post-CMOS technologies in hardware design.

Reversible logic [1–3] is one promising direction, which has been intensely studied in the last decade. This logic enables applications particularly in areas like quantum computation [4] or low power design [5], but also in optical computing [6], DNA computing [2], and nanotechnologies [7]. However, in comparison to conventional circuit design, reversible logic is subject to certain restrictions, e.g., fanout and feedback are not allowed [4]. Thus, the well-developed design flow for conventional circuits is not applicable for reversible logic. As a result, new approaches e.g., for synthesis [8,9], verification [10–12], and testing [13] of reversible circuits, have been developed.

In this context, also debugging of reversible logic is of interest. As in the design of other complex systems, ensuring the correctness of the circuit is crucial. But, while verification methods, e.g., based on simulation or formal techniques, only prove or disprove the existence of errors, debugging aims for directly locate them in the circuit. Since (manual) debugging of circuits including a large number of gates is a difficult task, a first *automatic* approach for reversible circuits has been introduced in [14]. However, particularly for circuits including multiple errors, this approach tends to generate large sets of error candidates that still have to be considered afterwards.

In this work, we propose an alternative debugging approach for multiple errors. We thereby focus on missing control errors, i.e., errors arising from the deletion of control lines in the considered reversible gates. This is motivated by the fact that removing control lines in reversible gates is an appropriate way to reduce the cost of a circuit (see e.g., [15]). Furthermore, missing control errors are an established error model in the area of testing (see e.g., [13]). Thus, such kinds of errors may often occur during the design of reversible circuits.

Our approach exploits the observation that erroneous gates in the circuit must be "activated" by the respective counterexamples. Conversely, if a gate is not activated by any counterexample, it can be removed from the set of gates to be considered. A similar, simulation-based idea has already been proposed in [16]. However, only single errors have been addressed there, whereas here multiple errors are targeted. Based on the observation, a new notion of error candidates

GLSVLSI'10, May 16–18, 2010, Providence, Rhode Island, USA.

v_1	v_2	v_1'	v_2'
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

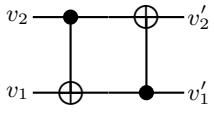

Figure 1: Reversible logic

is proposed. Afterwards, this notion is used to encode the determination of error candidates as a *hitting set problem*. Using this notion often error candidates different to the ones obtained by the approach from [14] are obtained. Thus, combining both approaches further improves the accuracy.

The rest of the paper is structured as follows: In Section 2, we review the basic notions of reversible circuits and briefly recapitulate debugging. Section 3 discusses the previous approach and provides the main motivation of our algorithm, which is introduced in Section 4. How to combine the considered approaches is described in Section 5. Finally, Section 6 gives experimental results obtained by the proposed methods and Section 7 concludes the paper.

2. BACKGROUND

2.1 Reversible Logic

Reversible logic realizes bijective functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ using *reversible circuits*. A reversible circuit G is a cascade of reversible gates g_i, i.e., $G = g_1 g_2 \dots g_d$. In this work, we consider *Multiple Control Toffoli gates*, in the following called *Toffoli gate*. A Toffoli gate over the set of *lines* $L = \{1, \dots, n\}$ has the form $g(C, t)$, where $C \subset L$ is the set of *control lines* and $t \in L \setminus C$ is the *target line*. A single Toffoli gate $g(C, t)$ realizes the bijective function

$$(v_1, \cdots, v_n) \mapsto (v_1, \cdots, v_{t-1}, v_t \oplus \bigwedge_{c \in C} v_c, v_{t+1}, \cdots, v_n).$$

That is, if all control line variables v_c are assigned to 1, the target line v_t is inverted. Under this assignment the gate is called *activated*. All other input values v_k with $k \in L \setminus \{t\}$ pass the gate unaltered. Note that the set of control lines may be empty. In this case the target line is always inverted. This gate is called *NOT gate* and represents a special case of a Toffoli gate.

Example 1. *Figure 1 shows a reversible circuit with two Toffoli gates realizing the function given in the truth table. The circuit is drawn in the standard notation (see e.g., [4]) and includes two Toffoli gates, i.e., $d = 2$.*

In the following, we denote the sub-circuit $g_1 g_2 \dots g_i$ by G_i. Furthermore, if $v \in \mathbb{B}^n$ is an input vector, then $G(v)$ denotes the output of G on input v.

2.2 The Debugging Problem

In the design of reversible circuits, errors may be introduced, e.g., during the synthesis or optimization.

The existence of errors can be detected by several verification methods [10–12]. Given a specification $\mathcal{S}$ and an implementation G, verification methods prove their functional equivalence or return a set $\mathbb{T}$ of counterexamples, i.e., a subset of the set $\{t \in \mathbb{B}^n \mid \mathcal{S}(t) \neq G(t)\}$ of all input vectors witnessing the difference between implementation and specification.

If the implementation differs from the specification, the source of the error is of interest. That is, the circuit has to be *debugged*. The debugging problem is to find a set of gates that are responsible for the erroneous behavior. Typically, this is a manual and time-consuming task that motivates the development of automatic approaches.

Since usually the number of errors in a circuit is not known, existing debugging approaches for both reversible and conventional circuits (e.g., [14,17]) apply an iterative approach to determine *error candidates*: Starting with $k = 1$, k is iteratively increased until a set of k gates (an error candidate) is identified that explains the erroneous behavior.

2.3 Missing Control Error Model

In this work we consider the *missing control error* model. Intuitively, missing control errors occur when control lines of a Toffoli gate are removed. This kind of errors may be introduced, e.g., by optimization approaches, since removing control lines reduces the costs of a circuit [15]. Furthermore, missing control errors are an established error model used in testing of reversible circuits [13]. Formally, missing control errors are defined as follows:

Definition 1. *Let $\mathcal{S}$ be a specification and G be an erroneous circuit realization. The circuit G contains k missing control errors, if there are k gates $g_{m_1}, \dots, g_{m_k}$ that can be replaced by gates $g'_{m_1}, \dots, g'_{m_k}$ with additional control lines such that the obtained circuit complies with the specification $\mathcal{S}$. Any set $\{g_{m_1}, \dots, g_{m_k}\}$ that fulfills this condition is called* error location.

3. CORRECTION-BASED DEBUGGING

This section briefly reviews the correction-based debugging approach [14]. Afterwards, we discuss its limitations, which motivate our alternative approach.

3.1 Approach

In [14], a first approach for debugging reversible circuits has been introduced (in the following denoted by *correction-based debugging*). The basic idea of this approach is inspired by debugging of conventional circuits as suggested in [17]: For each available counterexample, a copy of the circuit is created. The input of each copy is thereby constrained to the values given by the counterexample, while the output is constrained to the correct values. Furthermore, every gate is enriched with an *error correction logic*.

More precisely, a switch with select input s_i is added at the output of the target line (v_t) for each gate g_i as shown in Figure 2. If s_i is assigned to 0, the gate works as usual, i.e., $v'_t = v_t$. Otherwise, an arbitrary value is injected to correct the (possibly) wrong behavior at this gate. The same s_i-variables are used in all copies. Furthermore, the total number of s_i-variables allowed to be assigned to 1 is limited to k. The problem of finding a set of gates that can be corrected is translated to an instance of the *Boolean satisfiability (SAT)* problem: If there is a valid assignment to all signals for a given k, then each gate whose corresponding

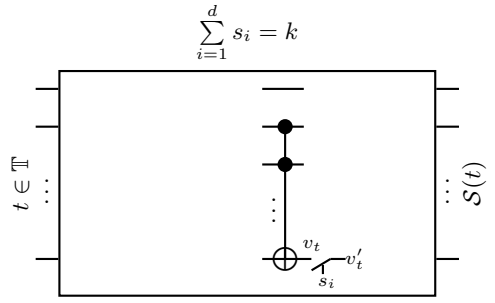

Figure 2: Correction-based Debugging [14]

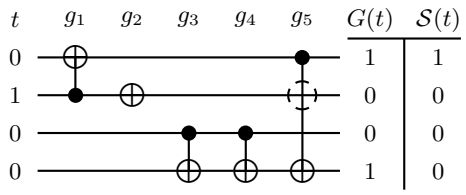

Figure 3: Circuit with one missing control error

s_i-variable is assigned to 1 belongs to an possible error candidate. Otherwise, it has been proven that no error location of size k exists. This instance can be solved using common SAT solvers (e.g., [18]).

3.2 Limits

Correction-based debugging suffers from high run-times for large circuits. This is mainly caused by the duplication of the whole circuit for each counterexample, which significantly blows up the respective SAT formulation. On the other hand, applying fewer counterexamples often leads to decreased accuracy, i.e., a high number of error candidates is obtained.

Besides that, in certain cases also the accuracy of the results is poor, in particular for multiple errors. For example, correction-based debugging cannot distinguish between neighboring gates that have the same target line. This is illustrated in the following example.

EXAMPLE 2. *Consider the circuit G shown in Figure 3 (the dashed circle indicates the missing control line) and a counterexample $t = (0, 1, 0, 0)$. For $k = 1$, the correction-based approach determines that the error can be corrected by inserting an appropriate value at the bottom line. However, it can not determine the concrete error positions, since a correction can be inserted at any of the gates g_3, g_4, and g_5, respectively.*

Furthermore, also error candidates are determined that can be excluded by a simple observation.

EXAMPLE 3. *Reconsider the circuit G from Figure 3 and $\mathbb{T} = \{t\}$ with $t = (0, 1, 0, 0)$ and $S(t) = (1, 0, 0, 0)$. Obviously, t is a counterexample, since the output of $G(t) = (1, 0, 0, 1)$ differs with the specification. The correction-based method yields (amongst others) the error candidate $\{g_4\}$, because a the correction logic can insert a 1 at g_4 that leads to the correct output. However, the gate g_4 can not be subject to a missing control error, since adding control lines would not lead to a different output on line 4.*

These examples motivate further investigation of the debugging problem for missing control errors. In particular, the last observation can be exploited for a new debugging approach. In the following section, this alternative is introduced.

4. HITTING SET-BASED DEBUGGING

In this section, we propose a new debugging approach for missing control errors. The general idea is based on the observation that – assuming the missing control error model – an erroneous gate behaves very similar to the correct gate. More precisely, a difference can only be observed when one of the omitted control lines is assigned to 0 and the remaining control lines are assigned to 1. Then, the erroneous gate toggles the target line, while the correct one does not change it. Thus, in order to show a missing control error, a counterexample has to activate at least one erroneous gate. Our debugging algorithm exploits this observation and tries to identify k gates, such that every counterexample activates at least one of them. This can be formulated as a hitting set problem [19].

In the following the approach is described. At first, we give a more detailed description of the underlying idea. Afterwards we propose an efficient way to solve the resulting hitting set formulation.

4.1 Observation and Problem Formulation

At first, the basic observation is formulated in the following lemma:

LEMMA 1. *Let G be a circuit with missing control errors at k gates $g_{m_1}, \ldots, g_{m_k}$ and let $t \in \mathbb{B}^n$ be an input vector. If t is a counterexample, then it activates at least one of the gates $g_{m_1}, \ldots, g_{m_k}$.*

PROOF. We prove this by contraposition: If t does not activate any erroneous gate, then it is not a counterexample. Let G' be the circuit that is obtained by replacing the erroneous gates $g_{m_1}, \ldots, g_{m_k}$ with the correct gates $g'_{m_1}, \ldots, g'_{m_k}$. Without loss of generality, we assume $m_1 < m_2 < \ldots < m_k$. For all i with $i < m_1$, it holds that $G_i(t) = G'_i(t)$, because G_i and G'_i are the same circuits. By assumption, t does not activate g_{m_1}, so it does not activate g'_{m_1}. Thus, $G_{m_1}(t)$ equals $G'_{m_1}(t)$. We can continue this argumentation until we reach the end of the circuit and obtain that $G(t) = G'(t)$. So t is not a counterexample. □

Based on this lemma, a set $\mathcal{S}$ of gates can be excluded from being an error location, if there is a counterexample that does not activate any of the gates in $\mathcal{S}$, since every counterexample has to activate at least one gate of an error location. We illustrate this insight using a small example.

EXAMPLE 4. *Assume an erroneous circuit with five gates $g_1, \ldots, g_5$ and four counterexamples $\mathbb{T} = \{t_1, \ldots, t_4\}$. Using simulation, the activated gates can easily be computed for each counterexample. Figure 4 shows a possible scenario. A cross at position (t_i, g_j) indicates that t_i activates g_j. For example, t_3 activates g_3, but does not activate g_2. According to Lemma 1, the set of gates $\{g_3, g_4\}$ cannot be an error location, since t_4 activates neither g_3 nor g_4. On the other hand, possible error locations are $\{g_3, g_4, g_5\}$ and $\{g_1, g_3\}$.*

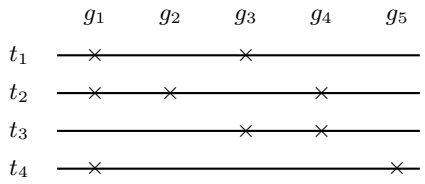

Figure 4: Scenario from Example 4

More formally, let $\mathbb{T}$ be a set of counterexamples. Then, the set G_t of gates that are activated by counterexample $t \in \mathbb{T}$ is defined by

$$G_t = \{g_i \in G \mid G_{i-1}(t) \text{ activates } g_i\}. \tag{1}$$

For example, in Figure 4 the set G_{t_2} is $\{g_1, g_2, g_4\}$, because t_2 activates g_1, g_2 and g_4. Lemma 1 can now be extended to take a set of counterexamples into account (instead of only one).

LEMMA 2. *Given an erroneous circuit G with error location $H := \{g_{m_1}, \ldots, g_{m_k}\}$, and a set of counterexamples $\mathbb{T}$. Then, for every counterexample $t \in \mathbb{T}$ it holds that*

$$H \cap G_t \neq \emptyset \tag{2}$$

PROOF. Let $t \in \mathbb{T}$ be a counterexample. According to Lemma 1 it activates at least one gate $g^* \in H$. By definition it is also $g^* \in G_t$, thus $G_t \cap H \neq \emptyset$. □

Overall, Lemma 2 states a necessary condition for error locations. Therefore, all sets of gates that fulfill Equation (2) can be regarded as error candidates. Based on this observation, the problem of finding error candidates can be formulated as a *hitting set problem* [19]: Given m counterexamples $t_1, \ldots, t_m$ and their corresponding activated gates $G_{t_1}, \ldots, G_{t_m}$, the task is to determine a set H that shares at least one element with each G_{t_i}. Every solution to the hitting set problem corresponds directly to an error candidate. In particular, if the hitting set problem has no solution with cardinality k, it has been proven that the considered circuit contains more than k missing control errors. This is illustrated in the following example.

EXAMPLE 5. *Again, consider the example from Figure 4. There is no singleton set H that "hits" an activated gate for every counterexample, because $\bigcap_{i=1}^{4} G_{t_i} = \emptyset$. Thus, no error candidate with $k = 1$ exists.*

4.2 Formulation as a SAT Instance

The hitting set problem is an $\mathcal{NP}$-complete problem [19]. To determine all error candidates of size k in a naive way, $\binom{d}{k}$ (i.e., $O(d^k)$) checks have to be performed. To avoid this, we propose a SAT-based formulation, i.e., we encode the hitting set problem as an instance of Boolean satisfiability such that the instance is satisfiable iff there is an error candidate of size k. Moreover, all solutions of the SAT instance correspond directly to error candidates. Passing the resulting instance to a SAT solver, the underlying hitting set problem can be solved very fast due to sophisticated techniques like search space pruning (see e.g., [18]).

The respective instance is created as follows: For every gate g_i of the erroneous circuit, a Boolean variable s_i is introduced. Assigning s_i to 1 means that the respective gate g_i is part of an error candidate. Accordingly, for every counterexample $t \in \mathbb{T}$, a clause

$$C_t = \bigvee_{g_i \in G_t} s_i \tag{3}$$

is added stating that for each set G_t at least one gate has to be in the error candidate. Finally, the constraint

$$\sum_{i=1}^{d} s_i = k \tag{4}$$

ensures that the number of variables $s_1, \ldots, s_d$ assigned to 1 is equal to k (i.e., exactly k gates g_i are part of an error candidate).

EXAMPLE 6. *The scenario shown in Figure 4 yields the constraints $C_{t_1} = s_1 \vee s_3$, $C_{t_2} = s_1 \vee s_2 \vee s_4$, $C_{t_3} = s_3 \vee s_4$, and $C_{t_4} = s_1 \vee s_5$ (according to Equation 3) as well as the constraint $s_1 + \cdots + s_5 = k$ (according to Equation 4). A satisfying assignment for $k = 2$ would be $s_1 = 1$, $s_2 = 0$, $s_3 = 1$, $s_4 = 0$, and $s_5 = 0$ from which the error candidate $\{g_1, g_3\}$ from above can be derived.*

Note that the SAT formulation is very compact, since every additional counterexample only leads to one additional clause. Passing instances created in this way to the SAT solver, the hitting set problem can be solved very fast even for circuits containing tens of thousands gates and counterexamples, respectively.

5. RELATION AND COMBINATION WITH CORRECTION-BASED DEBUGGING

The notion of error candidates introduced in Section 4 differs from previous approaches. In the correction-based approach error candidates are defined as sets of gates that can be replaced with other gates so that for each counterexample the correct output values result. In contrast, the method proposed in the last section relies on the observation that every counterexample activates at least one gate in the error candidate.

Both definitions of error candidates are necessary conditions of error locations, so both are reasonable notions for debugging algorithms. In Section 3.2 it was already shown that the correction-based approach yields error candidates that can be excluded using the newly proposed method. However, it is also the case that our approach yields error candidates that can be eliminated because no correction can be found there. Furthermore, both approaches agree on some error candidates that are not the error location, i.e., also their combination will not precisely locate the error. To illustrate this we resume Example 3.

EXAMPLE 7. *Consider again the circuit G in Figure 3 with the counterexample $t = (0, 1, 0, 0)$. A valid solution of the hitting set-based approach for $k = 1$ is $\{g_2\}$, since $G_t = \{g_1, g_2, g_5\}$ and $G_t \cap \{g_2\} \neq \emptyset$. On the other hand, applying the correction-based approach this error candidate can be disproven to be an error location. In fact, no correction can be inserted at g_2, because this would affect only line 2, which has no influence on the erroneous line 4. The only error candidate that is obtained by both approaches is $\{g_5\}$, the error location.*

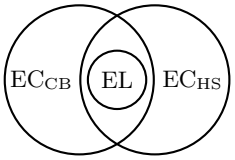

Figure 5: Relation of the resulting error candidates of both approaches

Overall, Example 3 and Example 7 show that both debugging approaches lead to different results. Figure 5 illustrates the relationship between the sets of error candidates obtained by the hitting set-based approach (EC_{HS}) and the correction-based approach (EC_{CB}). Intersecting both results reduces the number of error candidates and thus leads to a better approximation of the error location (EL).

To exploit the different notions of error candidates we propose to combine correction-based debugging with the presented method. Since both approaches are solved using a translation to SAT, they can easily be combined by taking the conjunction of the according CNF formulations. The overhead of augmenting the correction-based method with our approach is very small, since the cardinality constraint in Equation (4) is shared and thus for every counterexample only one clause needs to be added.

6. EXPERIMENTAL RESULTS

In this section, we empirically compare the hitting set-based debugging (in the following abbreviated with HS) and the correction-based debugging (CB). Then, we demonstrate the benefit obtained by combining the two approaches (CB+HS).

6.1 Setup

The evaluation is conducted as follows: By randomly removing control lines from some gates, we inject three errors into a set of circuits taken from RevLib [20]. Then, a set of at most 10 000 counterexamples is generated using methods from [12]. The overall counterexample generation time was less than five minutes for all circuits. Based on the set of counterexamples, SAT instances according to Section 4.2 (for the hitting set formulation) and according to [14] (for the correction-based formulation) are constructed. For CB, circuit copies for five (randomly chosen) counterexamples are created. The combination CB+HS is constructed by the conjunction of the SAT instances for HS and CB. All experiments were carried out on an Intel Xeon 3GHz with 4GB main memory running Linux. We used Minisat2 [18] as SAT solving engine.

Since the designer typically does not know the correct number of errors, we run the different debugging algorithms for $k = 1$, $k = 2$, and $k = 3$, respectively. For $k = 3$ the injected error is an error candidate and was returned by the respective approach (if no timeout occurs before). We compare the methods with respect to three criteria: First, the number of error candidates that are obtained before the error is correctly identified should be as small as possible, since the designer has to consider all of them in the worst case. Second, it is important to refute wrong assumptions on the number k of errors, i.e., the debugging approach should exclude all error candidates of a size less than the size of the error location. And third, of course all results should be available as fast as possible.

6.2 Results

We report on the obtained results in Table 1 and Table 2, respectively.

Consider first Table 1. The first columns list details of the circuits, i.e., the name, the number of lines, and the number of gates, respectively. Columns EC and Run Time denote the number of error candidates and needed time, respectively. A timeout (2000 seconds) is indicated by *To*. At every timeout, the error cardinality where it occurs is provided. Note that error candidates may be delivered before the timeout.

We observe that both CB and HS are not satisfying in many cases since they either yield tens of thousands of error candidates or timeout (and thus, may not pinpoint the designer to the error location). However, HS does not abort as often as CB. In particular, CB tends to timeout on large instances – sometimes without delivering any candidate. This confirms our discussion from Section 3.2. When considering the combination CB+HS, it can be seen that the best of the two approaches is conserved. Accuracy is increased significantly, since for most of the instances the designer obtains a feasible set of error candidates. Furthermore, while CB fails to deliver any error candidate for large instances (e.g., *urf2, urf4*, and *urf5*), both HS and CB+HS are able to correctly identify the error location. Only for the instances *mod1048576adder* and *testalu* the results are not yet satisfying because the set of error candidates is too large to be helpful for the designer.

Consider now Table 2. Here, the algorithms ability to refute wrong assumptions on the number of erroneous gates is documented. Since three errors are injected, the debugging method should disprove $k = 1$ and $k = 2$. In the table, we list the number of circuits for which the respective algorithm can disprove the error cardinalities 1 and 2. We used the same circuits as in Table 1. It can be seen that $k = 1$ can be disproven for the majority of the instances by all approaches – CB+HS even shows for 10 of the 11 instances the non-existence of a single error. Both CB and HS can refute $k = 2$ in less than half of the instances. In contrast, the combination CB+HS excludes error candidates of size 2 in 9 of 11 cases. Thus, the combination outperforms the single approaches also with respect to this criterion.

In summary, more accurate error candidates can be obtained in significantly shorter run-time using the CB+HS approach.

7. CONCLUSION

In this paper, we introduced an alternative method for debugging of reversible circuits. For a dedicated error model, namely missing control line errors, our method improves the results of the previously proposed debugging approach [14]. The observation that error locations have to be activated by every counterexample is thereby exploited. The resulting a hitting set problem is encoded as a SAT instance, which yields a very fast enumeration of the error candidates. Then, we illustrated the differences of the presented approach as well as the correction-based algorithm and proposed the combination of both.

Table 1: Results for debugging circuits with three errors.

Circuit Characteristics			Correction-based (CB)		Hitting-Set based (HS)			Combination (CB+HS)	
Circuit	n	d	EC	Run time	$\|\mathbb{T}\|$	EC	Run Time	EC	Run Time
ham15	15	132	14	0.3	10000	9 048	21.9	14	4.9
hwb6	6	126	86	8.4	28	1 008	0.5	1	1.0
hwb7	7	289	2 228	161.8	32	1 865	3.4	11	7.9
hwb8	8	637	4	**To.**$^{@k=3}$	82	203 480	**To.**$^{@k=3}$	32	311.2
hwb9	9	1 544	4	**To.**$^{@k=3}$	106	610	457.8	2	880.8
mod1048576adder	40	210	44 104	135.0	10000	22 157	25.9	22 157	61.8
sym9_148	10	210	220 440	**To.**$^{@k=3}$	80	4	0.1	4	0.1
testalu	103	359	5 694	24.5	10000	12 335	**To.**$^{@k=3}$	4 087	46.7
urf2_152	8	5 030	0	**To.**$^{@k=1}$	153	1	18.1	1	14.2
urf4_187	11	32 004	0	**To.**$^{@k=1}$	1201	1	166.9	1	165.9
urf5_158	9	10 276	0	**To.**$^{@k=1}$	291	1	21.8	1	32.6

Table 2: Excluded error cardinalities for 11 circuits

k	CB	HS	CB+HS
1	9	7	10
2	3	5	9

In the experiments, we empirically compared our algorithm, the correction-based algorithm, and their combination. We showed that combining both approaches, improves the efficiency and the accuracy. More precisely, the number of error candidates is significantly decreased. Hence, the designer is provided with better information about the error location. Furthermore, results are obtained in significantly shorter run-time.

8. ACKNOWLEDGMENT

This work was supported by the German Research Foundation (DFG) (DR 287/20-1).

9. REFERENCES

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

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

[3] 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.

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

[5] B. Desoete and A. D. Vos, "A reversible carry-look-ahead adder using control gates," *INTEGRATION, the VLSI Jour.*, vol. 33, no. 1-2, pp. 89–104, 2002.

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

[7] R. C. Merkle, "Reversible electronic logic using switches," *Nanotechnology*, vol. 4, pp. 21–40, 1993.

[8] 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.

[9] R. Wille and R. Drechsler, "BDD-based Synthesis of Reversible Logic for Large Functions," in *Design Automation Conf.*, 2009, pp. 270–275.

[10] 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.

[11] 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.

[12] R. Wille, D. Große, D. M. Miller, and R. Drechsler, "Equivalence checking of reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2009.

[13] 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.

[14] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler, "Debugging of Toffoli networks," in *Design, Automation and Test in Europe*, 2009, pp. 1284–1289.

[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] S. Frehse, R. Wille, and R. Drechsler, "Efficient simulation-based debugging of reversible logic," in *Int'l Symp. on Multi-Valued Logic*, 2010.

[17] 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.

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

[19] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, 1972, pp. 85–103.

[20] 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*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.