Automatic Test Pattern Generation for Multiple Missing Gate Faults in Reversible Circuits
Work in Progress Report

Anmol Prakash Surhonne[1,2], Anupam Chattopadhyay[1], and Robert Wille[3]

[1] Nanyang Technological University, Singapore
[2] Technical University of Munich, Germany
[3] Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
anmolpra001@e.ntu.edu.sg, anupam@ntu.edu.sg, robert.wille@jku.at

Abstract. Logical reversibility is the basis for emerging technologies like quantum computing, may be used for certain aspects of low-power design, and has been proven beneficial for the design of encoding/decoding devices. Testing of circuits has been a major concern to verify the integrity of the implementation of the circuit. In this paper, we propose the main ideas of an ATPG method for detecting two missing gate faults. To that effect, we propose a systematic flow using *Binary Decision Diagrams* (BDDs). Initial experimental results demonstrate the efficacy of the proposed algorithms in terms of scalability and coverage of all testable faults.

1 Introduction

Reversible circuits represent an emerging technology based on a computation paradigm which significantly differs from conventional circuits. In fact, they allow bijective operations only, i. e., n-input n-output functions that map each possible input vector to a unique output vector. Reversible computation enables several promising applications and, indeed, surpasses conventional computation paradigms in many domains including but not limited to quantum computation (see, e. g., [1]), certain aspects of low-power design (as experimentally observed, e. g., in [2]), encoding and decoding devices (see, e. g., [3, 4]), or verification (see, e.g., [5]).

Accordingly, also the consideration of the design of reversible circuits received significant interest. In comparison to conventional circuit design, new concepts and paradigms have to be considered here. For example, fanout and feedback are not directly allowed. This affects the design of reversible circuits and requires alternative solutions. To this end, several design approaches have been introduced. An overview of that is, e. g., provided in [6, 7].

In parallel, how to physically build reversible and quantum circuits is being investigated and led to first promising results (see, e. g., [8, 9]). With this, also the question of how to prevent and detect faults in the physical realization became relevant. In particular for quantum computation, this is a crucial issue: Quantum systems are much more fault-prone than conventional circuits, since the phenomenon of quantum de-coherence forces the qubit states to decay – resulting in a loss of quantum information which, eventually, causes faults. Faults also do originate from the fact that quantum computations are conducted by a stepwise application of gates on qubits.

As a result, researchers studied different fault models and the respective methods for *Automatic Test Pattern Generation* (ATPG). In that regard, one

of the earliest works on different fault models for quantum circuits is [10], which proposed these models based on the implementation principles of quantum circuits using trapped ion technology [1]. The types of fault model included, for example, the *single missing gate fault* (SMGF), the *partial missing gate fault* (PMGF), and the multiple missing gate fault (MMGF). However, mainly ATPG methods for single faults have been proposed thus far (see e.g. [11–13]).

2 Background

To keep the remainder of this work self-contained, this section briefly reviews the basics of reversible circuits as well as ATPG and the fault models considered for this kind of circuits.

2.1 Reversible Circuits

Reversible circuits are digital circuits with the same number of input signals and output signals. Furthermore, reversible circuits realize bijections, i.e. each input assignment maps to a unique output assignment. Accordingly, computations can not only be performed from the inputs to the outputs but also in the other direction. Reversible circuits are composed as cascades of reversible gates. The *Toffoli gate* [14] is widely used in the literature and also considered in this paper.

Definition 1. *Given a set of variables or signals* $X = \{x_1, x_2, \ldots, x_n\}$*, a* Toffoli gate $G(C, t)$ *is a tuple of a possibly empty set* $C \subset X$ *of* control lines *and a single* target line $t \in X \setminus C$*. The Toffoli gate inverts the value on the target line if all values on the control lines are set to* 1 *or if* $C = \emptyset$*. All remaining values are passed through unaltered. In the following, Toffoli gates are also denoted as* Multiple Controlled Toffoli *(MCT) gates.*

2.2 Test of Reversible Circuits

As in conventional circuits, *Automatic Test Pattern Generation* (ATPG) methods for reversible circuits aim at determining a set of stimulus patterns (denoted as *testset*) in order to detect faults in a circuit with respect to an underlying fault model. A single missing gate fault is defined as follows.

Definition 2. *Let* $G(C, t)$ *be a gate of a reversible circuit. Then,* a Single Missing Gate Fault *(SMGF) appears if instead of* G *no gate is executed (i.e.* G *completely disappears). The method to detect SMGF is widely studied and can be referred in previous works.*

Definition 3. *Let* $\mathcal{G}$ *be a set of* k *gates from a reversible circuit. Then,* a Multiple Missing Gate Fault *(MMGF) appears if instead of* $\mathcal{G}$ *no gates are executed (i.e. all gates* $\mathcal{G}$ *completely disappear in* G*).*

In the following, we consider MMGFs with two missing gates. However, the methods described below can easily be extended for an arbitrary number of faulty gates. From here on forward, MMGF is referred to as missing of two gates. In order to detect MMGFs, the respective gates have to be activated so that the faulty behaviour can be observed at the outputs of the circuit. However, in case of multiple faults, the absence of one gate within the circuit may cause the deactivation of another gate in the circuit – leading to masking effects. Besides that, the absence of two gates may lead to no change in the outputs – leading to an undetectable fault. Because of that, the dependencies of two gates considered as one MMGF have to be analyzed in order to generate a test pattern.

Definition 4. *Two gates G_x and G_y $(x < y)$ are said to be dependent if the target line of G_x is involved in the activation of the SMGF of G_y. More precisely, a Toffoli gate G_y is dependent from gate G_x, if the target line of G_y is the control line of G_x or if G_y is dependent on another gate G_z which is dependent from G_x.*

3 ATPG for MMGF Detection

This section describes the proposed approach for ATPG of MMGFs. The goal is to obtain a test set that covers all possible faults with a minimum number of test patterns. The proposed solution has four phases. First, test patterns for SMGFs, i.e. for the single faults, are obtained and compactly stored in a BDD. Based on that, the dependencies already discussed in the previous section are analyzed. The results from these two steps (i.e. the patterns for all SMGFs as well as the information about the dependencies of the gates in the currently considered circuit) are then utilized in order to obtain MMGF test sets. Finally, a covering algorithm is applied to minimize the obtained test set – yielding a minimal result covering all MMGFs.

3.1 Test Generation for SMGFs

In order to obtain all desired test patterns, it is assumed that only one gate is faulty at a time in this step. Also, the faults are detected at the primary outputs of the circuits and no distinction is made between different types of lines like output, garbage etc. Constant inputs of the circuit are assumed to be variable for the purpose of testing. For a circuit with n lines and N gates, there are N SMGFs, and the test patterns are obtained by activating the considered gate. Overall, this yields 2^{n-k} possible test patterns that can be obtained for testing a SMGF. In order to compactly store them, *Binary Decision Diagrams* (BDDs, [15]) are applied.

3.2 Dependency Analysis

The next step is to analyse the dependencies between all combinations of two faulty gates as discussed in Section 2.2. The following pseudo-code describes how the dependencies between the gates are obtained.

Algorithm 1 Dependency Analysis

```
N : Number of gates of the circuit.
G_x : Gate numbered x.
Table[N] : Table storing the dependencies of the N gates.
for i = 0 ; i < N ; i++ do
    for j = 0 ; j < i ; j++ do
        if targetLine(G_j) = controlLine(G_i) then
            Insert G_j to Table[i].
            Insert all the gates G_j is dependent from to Table[i].
        end if
    end for
end for
```

3.3 MMGF Test Generation

Using the test patterns for SMGFs as well as information about the dependencies of all combinations of two faulty gates, now the respective test patterns for MMGFs with two faulty gates can be obtained. More precisely, without loss of generality, consider two gates G_x and G_y as well as their test patterns for corresponding SMGFs (denoted as $S(G_x)$ and $S(G_y)$). If these two gates are independent to each other, we determine two test patterns (denoted as $M(G_y, G_x)_1$ and $M(G_y, G_x)_2$): one test pattern to activate the gate G_x and not G_y and another to activate G_y. More precisely:

$$M(G_y, G_x)_1 = S(G_y) \qquad M(G_y, G_x)_2 = (S(G_x) \cap \overline{S(G_y)})$$

If the two gates are dependent on each other, we determine two other tests patterns: one test pattern to activate the gate G_x and not G_y and vice versa. More precisely:

$$M(G_y, G_x)_1 = (S(G_y) \cap \overline{S(G_x)}) \qquad M(G_y, G_x)_2 = (S(G_x) \cap \overline{S(G_y)})$$

Besides that, masking may occur leading to untestable faults (as discussed in Section 2.2). A fault is untestable using this method

$$M(G_y, G_x)_i = \{\emptyset\} \text{ where } i = \{1, 2\}$$

Using this as basis, the respective determinations can efficiently be conducted on the BDD. More precisely, the BDD containing the SMGF test patterns are manipulated for all the MMGF yielding a BDD with n inputs and $2 * {}^{N}C_2$

3.4 Minimal Test Set Determination

Once all the test patterns for the individual MMGFs are obtained, it is tried to derive the minimal testset covering all the faults. To that effect, two different techniques are proposed.

First, row and column reduction of a covering table [16] is implemented following a greedy scheme. *Second*, a covering algorithm is implemented using the BDDs to determine a minimum cover of the stored patterns. For a circuit with n lines, the covering BDD has 2^n inputs and 1 output. Having that, the minimum test set is equivalently represented by the minimum-weighted path from the output of the BDD to the 1-terminal of the BDD, where the *then* arc has a weight of 1 and the *else* arc has a weight of 0.

4 Experimental Results

The ideas proposed above have prototypically been implemented on a Ubuntu Linux system running on a Intel(R) Core(TM) $i7-3630QM$ CPU $64bit@2.4Ghz$ and $6GB$ of RAM. For the first two steps, i.e. determining the SMGF testset and the dependency analysis, Revkit [17] has been applied. For the remaining steps, the BDD package *CUDD* [18] was employed. All experiments were conducted on benchmark circuits obtained from Revlib [19]. Table 1 presents the obtained experimental results. The results show that the algorithm performed well for larger circuits covering a large number of faults. We obtained 100% fault coverage for circuits like $rd73\_140$ and $rd84\_142$, whereas the worst performance was for the circuit $ex3\_229$ with a coverage of 28.6. This was due to a large number

Table 1: Experimental Results

				SMGF	MMGF								
Circuit	N	n	Type	TP	TP_G	TP_{BDD}	TF	D	U	%D	%U	T_G	T_{BDD}
cm82a_208	22	8	MCT	4	9	8	231	21	107	9.1	46.3	0.08s	6s
ex3_229	7	6	MCT	2	3	3	21	4	15	19	71.4	0.01s	0.01s
graycode6_47	5	6	MCT	1	4	3	10	0	0	0	0	0.01s	0.04s
ham3_102	5	3	MCT	2	4	4	10	6	1	60	10	0.01s	0.01s
hwb4_52	11	4	MCT	2	6	6	55	41	3	74.5	5.5	0.01s	0.04s
hwb5_55	24	5	MCT	3	12	11	276	192	78	69.6	28.3	0.03s	0.1s
majority_239	8	6	MCT	3	4	4	28	6	18	21.4	64.3	0.01s	0.01s
mini-alu_167	6	4	MCT	3	5	5	15	15	1	100	6.7	0.01s	0.02s
mod10_171	10	4	MCT	3	6	6	45	29	20	64.4	44.4	0.01s	0.03s
mod5adder_128	15	6	MCT	2	8	7	105	59	66	56.2	62.9	0.02s	0.12s
mod5d1_63	7	5	MCT	1	4	4	21	4	2	19	9.5	0.01s	0.02s
mod8-10_177	14	5	MCT	2	6	6	91	53	41	58.2	45.1	0.01s	0.03s
rd32-v0_66	4	4	MCT	2	2	2	6	2	2	33.3	33.3	0.01s	0.03s
rd53_137	16	7	MCT	2	9	8	120	44	36	36.7	30	0.05s	0.07s
sym6_145	36	7	MCT	1	6	6	630	0	180	0	28.6	0.1s	0.13s
xor5_254	7	6	MCT	1	3	3	21	2	10	9.5	47.6	0.01s	0.06s
ham7_105	21	7	MCT	3	9	5	210	35	14	16.7	6.7	0.05s	-
hwb5_53	55	5	MCT	4	21	21	1485	1396	98	94	6.6	0.02s	0.15s
hwb6_56	126	6	MCT	8	43	43	7875	7614	328	96.7	4.2	1.7s	1.9s
3_17_13	6	3	MCT	2	4	4	15	12	7	80	46.7	0.01s	0.01s
root_255	99	13	MCT	14	14	-	4851	660	660	13.6	13.6	16s	-
rd73_140	20	10	MCT	3	6	-	190	72	0	37.9	0	1s	-
rd84_142	28	15	MCT	3	8	-	378	125	0	33.1	0	90s	-
adr4_197	55	13	MCT	6	13	-	1485	65	525	4.4	35.4	38s	-
ham15_108	70	15	MCT	8	26	-	2415	1474	154	61	6.4	100s	-
hwb7_59	289	7	MCT	14	71	-	41616	40735	742	97.9	1.8	48s	-
hwb7_60	166	7	MCT	8	32	-	13695	13683	71	99.9	0.5	7s	-
0410184_169	46	14	MCT	1	9	-	1035	315	274	30.4	26.5	8s	-

N - Number of gates n - Number of lines.
TP - Number of test patters covering all SMGF.
TP_G - Number of test patterns covering all MMGF using greedy method.
TP_{BDD} - Number of test patterns covering all MMGF using BDD based covering algorithm.
TF - Total Number of MMGF faults. D - Number of dependencies.
U - Number of untestable faults.
T_G Time taken for obtaining test patterns using greedy method.
T_BDD Time taken for obtaining test patterns using BDD based covering algorithm.
%D - Percentage of dependencies = $(D/TF) * 100$
%U - Percentage of untestable faults = $(U/TF) * 100$

of NOT gates, and hence the performance could be improved by DFT techniques. For the circuits $rd32 - v0\_66$ and $root\_255$, the SMGF and MMGF test patterns are identical. This is because all the dependent faults of these circuits are untestable. Considering the two covering methods, i.e. the greedy heuristic vs. the BDD-based approach, clearly shows the difference in runtime – especially for large circuits, where the BDD-based covering could not be completed after a long time. This was expected as the BDD-based approach obtains an exact, i.e. minimal cover, while the heuristic solution only approximates that. Besides that, it can be observed that the test set size determined by the greedy heuristic is, for most cases, the same as the minimal size obtained by the BDD-based approach. That is, the quality of the heuristic is rather good and often yields test sets which are close to the optimum. It should also be noted that the untestable faults are due to the function of the algorithm, and other methods can be used

to detect these faults, which will be considered in the future work where we try to increase the coverage to 100%.

Acknowledgement.
This work has partially been supported by the EU COST Action IC1405.

References

1. M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.
2. A. Berut, 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, 2012.
3. R. Wille, R. Drechsler, C. Osewold, and A. Garcia-Ortiz, "Automatic design of low-power encoders using reversible circuit synthesis," in *Design, Automation and Test in Europe*, 2012, pp. 1036–1041.
4. A. Zulehner and R. Wille, "Taking one-to-one mappings for granted: Advanced logic design of encoder circuits," in *Design, Automation & Test in Europe*, 2017.
5. L. Amarú, P.-E. Gaillardon, R. Wille, and G. De Micheli, "Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 175–180.
6. R. Drechsler and R. Wille, "From truth tables to programming languages: Progress in the design of reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2011, pp. 78–85.
7. M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuits - a survey," *ACM Computing Surveys*, 2011.
8. L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, p. 883, 2001.
9. 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.
10. 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.
11. K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault testing for reversible circuits," *Asian Test Symp.*, pp. 410–416, 2003.
12. J. P. Hayes, I. Polian, and B. Becker, "Testing for missing-gate-faults in reversible circuits," *Asian Test Symp.*, pp. 100–105, 2004.
13. R. Wille, H. Zhang, and R. Drechsler, "ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization," in *IEEE Computer Society Annual Symposium on VLSI*, 2011, pp. 120–125.
14. 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.
15. R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.
16. M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Springer Science & Business Media, 2004, vol. 17.
17. M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "RevKit: An open source toolkit for the design of reversible circuits," in *Reversible Computation 2011*, ser. Lecture Notes in Computer Science, vol. 7165, 2012, pp. 64–76, RevKit is available at www.revkit.org.
18. F. Somenzi, "Cudd: Cu decision diagram package release 2.4. 2," 2009.
19. 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.