Fault Ordering for Automatic Test Pattern Generation of Reversible Circuits

Robert Wille* Hongyan Zhang* Rolf Drechsler*†

*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{rwille,zhang,drechsle}@informatik.uni-bremen.de

***Abstract*—Reversible circuits are an attractive computation alternative as they build the basis for many emerging technologies such as quantum computation or low power design. Since first physical realizations of reversible circuits have already been presented in the past, how to efficiently test such circuits became a current research topic. Consequently, several approaches for *Automatic Test Pattern Generation* (ATPG) have been presented in the past.**

However, the order in which the respective faults are targeted has a significant effect on the resulting test size. While determining good fault orderings has intensely been considered for the test of conventional circuits, according strategies for reversible circuits have not been evaluated yet. This is done in this paper. To this end, a fault ordering scheme is presented that explicitely exploits the reversibility of the underlying circuits. Experimental results show that the proposed scheme leads to improvements of up to 65% in the size of the testset.

I. 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, superiors conventional computation paradigms in many domains including but not limited to quantum computation or low power design (see e.g. [1], [2], [3]).

In comparison to conventional circuit design, new concepts and paradigms have to be considered. For example, fanout and feedback are not directly allowed. This makes the design of reversible circuits different and requires alternative solutions. To this end, different approaches ranging from synthesis (see e.g. [4], [5], [6], [7], [8]), optimization (see e.g. [9], [10]), verification (see e.g. [11], [12], [13]), and debugging (see e.g. [14]) have been introduced. An overview of that is e.g. provided in [15].

Even if all this still is basic research, first physical realizations have already been presented (see e.g. [16], [17]). Motivated by this, also issues concerning testing of these new circuits are getting addressed by researchers. In this work, we particularly consider *Automatic Test Pattern Generation* (ATPG).

In ATPG, a set of test pattern is generated which is capable of detecting all possible faults in a given circuit assuming a certain fault model. A major goal is thereby to keep the size of the testset as small as possible. First approaches follow thereby a greedy and branch-and-bound scheme [18] or applied ILP formulations [19]. ATPG approaches that can handle large circuits make use of formals methods like Boolean satisfiability [20] or Pseudo Boolean Optimization [21].

However, the order in which faults are targeted by the respective ATPG engines significantly affects the size of the resulting testset. This has already been investigated for conventional ATPG. As a consequence, several fault ordering strategies have been developed for this domain in which e.g. faults are classified by their "hardness" of detection (see e.g. [22], [23], [24], [25]). Test patterns obtained for faults considered to be "hard" to detect very likely also detect many of the faults considered to be "easy" to detect. Hence, "harder" faults are targeted first during ATPG.

In this paper, a corresponding scheme is proposed for ATPG of reversible circuits. Instead of simply adopting conventional fault ordering techniques, we are directly making use of the underlying reversibility of the considered circuits. In fact, reversible circuits allow for an easy calculation of the number of test patterns available to detect a fault. This number provides a perfect metric to denote the "hardness" of a fault by which the faults can accordingly be ordered. Experimental results show that applying the proposed metric reduces the size of the testsets by up to 65% in comparison to the currently applied ordering schemes.

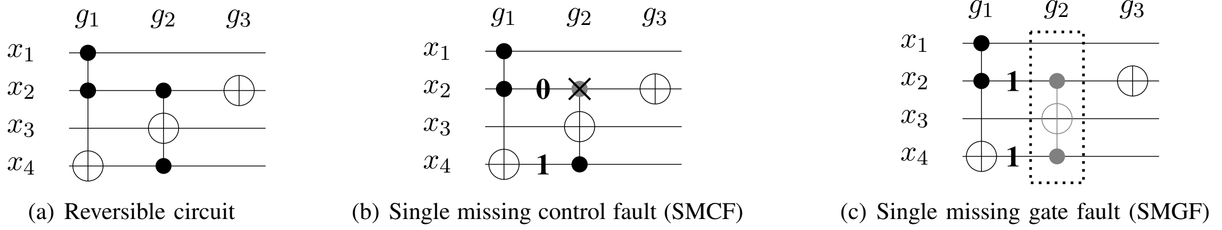


Fig. 1. Reversible circuit with different faults

In the remainder of this paper, the contribution is presented as follows. The next section introduces the basics on reversible circuits. A brief review on test of reversible circuits is provided in Section III. Afterwards, the effect of fault ordering is discussed in Section IV before the proposed fault ordering scheme is presented in Section V. Finally, experimental results are provided in Section VI, while Section VII concludes the paper.

II. Reversible Logic

A reversible function is a function $f{:}\mathbb{B}^n \to \mathbb{B}^m$ over inputs $X = \{x_1, ..., x_n\}$ with two properties: (1) its number of inputs is equal to its number of outputs (i.e. $n = m$) and (2) it maps each input pattern to a unique output pattern. A reversible circuit is a realization of a reversible function. Accordingly, reversible circuits also have n-inputs, n-outputs, and map each input pattern to a unique output pattern. Because of that, the output assignment can be obtained from the input assignment *and* vice versa. In comparison to conventional circuits, fanout and feedback are not directly allowed in reversible circuits [1]. As a result, every reversible circuit G is composed of a cascade of reversible gates g_i, i.e. $G = g_1 g_2 ... g_d$. In this work, we consider the most widely used reversible gate, the Toffoli gate [26]. A Toffoli gate is defined as follows:

Definition 1: A *Toffoli gate* over the set of inputs $X = \{x_1, ..., x_n\}$ has the form $g(C, x_t)$, where $C \subset X$ is the set of *control lines* and $x_t \in X \setminus C$ is the *target line*. A single Toffoli gate $g(C, x_t)$ realizes the bijective function

$$(x_1, ..., x_n) \mapsto (x_1, ..., x_{t-1}, x_t \oplus \bigwedge_{x_c \in C} x_c, x_{t+1}, ..., x_n).$$

That is, the target line x_t is inverted if (1) all control line variables $x_c \in C$ are set to 1 or (2) the set of control lines is empty, i.e. $C = \emptyset$. In these cases, the gate is called *activated*. All other signals x_k with $x_k \in X \setminus \{x_t\}$ always pass the gate with their value unaltered.

Example 1: Fig. 1(a) shows an example of a reversible circuit which is composed of Toffoli gates. This circuit has four circuit lines and three Toffoli gates, i.e. $n = 4$ and $d = 3$. Control lines are denoted by a $\bullet$, while the target line is denoted by an $\oplus$.

III. Test of Reversible Circuits

For a given circuit G, the goal of *Automatic Test Pattern Generation* (ATPG) is to create a testset $\mathbb{T}_{\mathcal{F}}$, i.e. a set of stimulus patterns, which detects all faults provided in a fault list $\mathcal{F}$. The fault list $\mathcal{F}$ is composed by all possible faults that may occur in the circuit according to a given fault model.

A. Fault Models

In this paper, we explicitly consider the fault models introduced in the following definition:

Definition 2: Let $g(C, x_t)$ be a Toffoli gate of a circuit G. Then,

1) a *Single Missing Control Fault* (SMCF) occurs if, instead of g, a gate $g'(C', x_t)$ with $C' = C \setminus \{x_i\}$ is executed (i.e. a gate with a missing control line x_i is executed instead of g)[1].
2) a *Single Missing Gate Fault* (SMGF) appears if, instead of g, no gate is executed (i.e. g completely disappears).

In order to detect a fault, the respective gates have to be activated so that the faulty behavior shows up at the outputs of the circuit. Depending on the considered fault, this requires certain input assignments [19]. More precisely:

Definition 3: Let $g(C, x_t)$ be a Toffoli gate of a circuit G.

1) To detect an SMCF in g, all control lines in C (except the missing one) have to be set to 1, while the missing control line has to be set to 0. The assignment of the remaining lines can arbitrarily be chosen.
2) To detect an SMGF in g (i.e. the disappearance of g), all control lines in C have to be set to 1, i.e. g simply has to be activated. The assignment of the remaining lines can arbitrarily be chosen.

[1]Note that in the literature (e.g. in [19]), the SMCF model is also called *Partial Missing Gate Fault Model*.

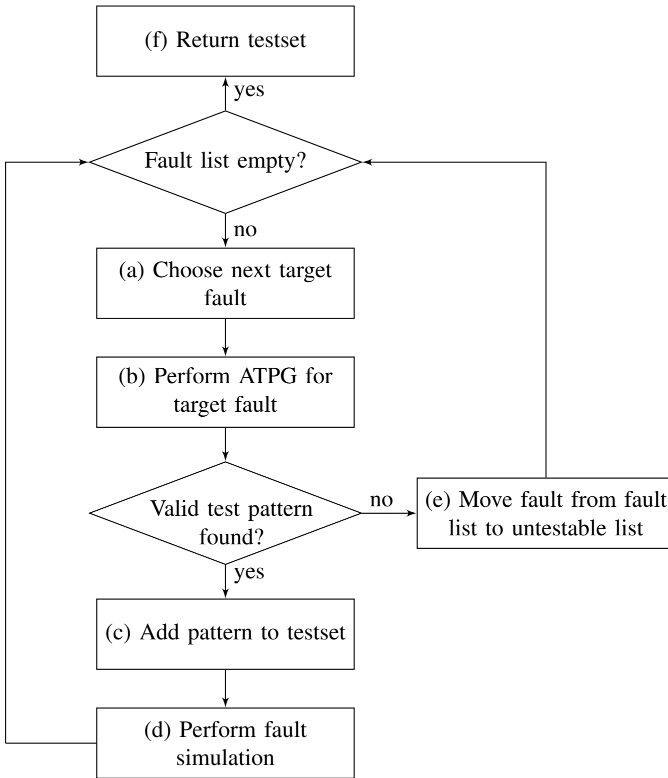


Fig. 2. General flow for ATPG

Example 2: Fig. 1 illustrates an SMCF (b) and an SMGF (c), respectively, which can occur in the reversible circuit previously introduced in Fig. 1(a). The respective assignments needed to detect these faults are also given.

B. General Flow

In general, testing of reversible circuits is easy. Already for conventional circuits, it has been shown that a maximized information output implies the highest probability of fault detection [27]. Since reversible circuits are information loss-less, the best possible information output is guaranteed. Only if additional constraints like constant inputs have additionally to be considered, ATPG becomes harder (as discussed in [21]). Then, faults might be classified to be *untestable*.

Consequently, in order to create a complete testset $\mathbb{T}_{\mathcal{F}}$ for a given fault list $\mathcal{F}$, the ATPG flow as depicted in Fig. 2 is applied: As long as the fault list is not empty (i.e. faults exist which do not have been classified yet), a new fault is chosen (Step (a)). Afterwards, it is tried to generate a test pattern for that fault. To this end, approaches e.g. based on simulation or Boolean satisfiability can be applied (see e.g. [21] for details). If this was successful (i.e. if a valid test pattern has been obtained), the generated pattern is added to the testset (Step (c)). Additionally, fault simulation is performed, i.e. the pattern is simulated and further faults which are detected by this are removed from the fault list (Step (d)). In contrast, if no valid test pattern has been found (i.e. if the fault is untestable), the respective fault is moved from the fault list to the list of untestable faults (Step (e)). This list can be used later e.g. to optimize the considered circuit. If all faults have been classified, the process terminates (Step (f)). Applying this flow, a complete testset for all testable faults under a certain fault model is generated.

IV. Effect of Fault Ordering

Following the ATPG flow as shown in Fig. 2, the order in which faults are targeted by the ATPG engine has a significant effect. This is illustrated by the following example.

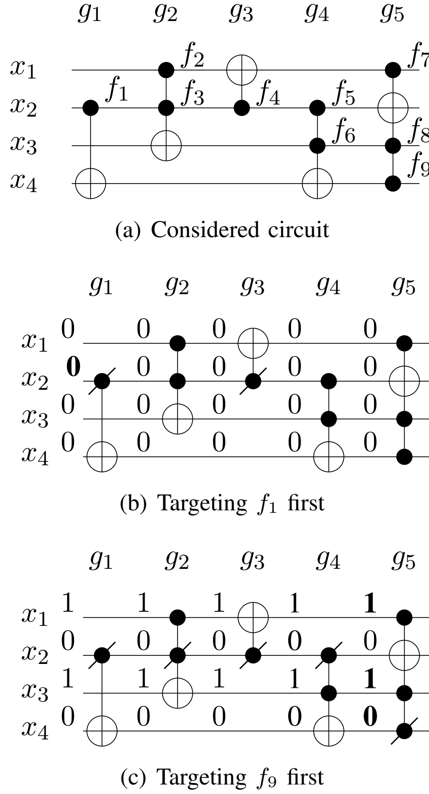
Example 3: Consider the circuit shown in Fig. 3(a) together with a fault list $\mathcal{F} = \{f_1, \ldots, f_9\}$ composed of single missing control faults. If the fault f_1 is targeted first, a test pattern has to be generated that sets all the control lines of g_1 (except the missing one) to 1, while the missing one has to be set to 0 (indicated bold in Fig. 3(b)). This could lead to the pattern 0000 as shown in Fig. 3(b). A subsequent fault simulation unveils that this pattern additionally detects the fault f_4.

However, if instead of f_1 the fault f_9 is targeted first, significantly more faults could be detected. For example, this could lead to a test pattern 1010 as shown in Fig. 3(c) which additionally detects faults f_1, f_3, f_4, and f_5, i.e. a much larger set.

These observations are not new. Similar behavior can be observed for ATPG of conventional circuits. Motivated by this, several fault ordering strategies have been developed for this domain (see e.g. [22], [23], [24], [25]). In general, they are based on the following premise: Faults are considered first for which a pattern is probably hard to generate. This is motivated by the fact that test patterns for "hard" faults very likely also detect many of the "easy" faults as well. Hence, the total number of test patterns to be generated can be kept smaller. In order to classify a fault to be whether "hard" or "easy", several schemes have been applied for conventional circuits. However, to the best of our knowledge, corresponding schemes for faults in reversible circuit have not yet been proposed.

V. Proposed Fault Ordering Scheme

In this section, we present the fault ordering scheme proposed in this work. The general idea is similar to the fault ordering schemes applied for conventional circuits, i.e. "harder" faults are targeted first. But, the classification of a fault f being "hard" or "easy" is different



(a) Considered circuit

(b) Targeting f_1 first

(c) Targeting f_9 first

Fig. 3. Effect of fault ordering

and explicitly exploits the reversibility of the underlying circuit structure. In fact, this classification is based on the number of possible test patterns that could detect f. This number can easily be obtained by the number of control lines in the corresponding gate. More precisely:

Lemma 1: Let f be an assumed SMCF in a gate $g(C, x_t)$ with n lines in total and $|C|$ control lines. Then, a total of at most $2^{n-|C|}$ test patterns exist that could detect f.

Proof: In order to detect an SMCF, all control lines except the missing one have to be set to 1, while the missing one itself has to be set to 0. In total, this makes $|C|$ fix assignments to gate g. The values of all remaining lines (including the target line) can arbitrarily be chosen. ∎

Lemma 2: Let f be an assumed SMGF in a gate $g(C, x_t)$ with n lines in total and $|C|$ control lines. Then, a total of at most $2^{n-|C|}$ test patterns exist that could detect f.

Proof: The same argumentation as for SMCF applies except that all control lines have to be set to 1. ∎

Obviously, a fault for which fewer test patterns exist is "harder" to detect than faults for which more test patterns are principally available. This is illustrated in the following example.

Example 4: Consider again the circuit from Fig. 3(a). The first fault f_1 can be detected by all test patterns setting the inputs of gate g_1 to `-0--`, whereby "–" denotes an arbitrary assignment. For the remaining faults, test patterns leading to the following assignments to the inputs of the respective gates are required:

- For f_2 the assignment `01--` to gate g_2
- For f_3 the assignment `10--` to gate g_2
- For f_4 the assignment `-0--` to gate g_3
- For f_5 the assignment `-01-` to gate g_4
- For f_6 the assignment `-10-` to gate g_4
- For f_7 the assignment `0-11` to gate g_5
- For f_8 the assignment `1-01` to gate g_5
- For f_9 the assignment `1-10` to gate g_5

The assignment of the faults f_7, f_8, f_9 of g_5 have the fewest don't care assignments, i.e. – occurrences. Hence, these faults should be classified to be in "hardest" fault class and targeted first. Then, the test patterns generated for these "hard" faults may detect also the "easy" faults. In fact, the three test patterns 0100, 1010, and 0011 which detect these "hardest" faults already detect all the other faults in the considered circuit.

Note thereby that Lemma 1 and Lemma 2 constitute upper bounds. In fact, due to additional constraints (like constant inputs as discussed in [21]), the actual number of possible test patterns could be less than $2^{n-|C|}$. In the worst case, even no test pattern could be available (if this fault is untestable). Nevertheless, since both bounds provide easy to calculate approximations of the "hardness" of a fault, they represent a plausible criteria for sorting the fault list.

Motivated by the discussions from above, we suggest a fault ordering scheme which targets all faults according to the number of control lines of the gate to which the fault is associated to. Faults belonging to gates with a larger number of control lines are thereby targeted first.

Example 5: Considering again the circuit from Fig. 3(a), the proposed fault ordering scheme would lead to the following order in which faults are targeted: First f_7 is addressed, followed by f_8, f_9, f_2, f_3, f_5, f_6, f_1, and eventually f_4.

As shown by an experimental evaluation, whose results are reported in the next section, already this simple scheme leads to a significant compaction of the complete testset for a given circuit.

VI. Experimental Evaluation

In order to evaluate the proposed fault ordering scheme, the ATPG flow shown in Fig. 2 has been implemented in C++ on top of *RevKit* [28]. For the generation

of a test pattern targeting the currently considered fault (Step (b) in Fig. 2), SAT-based ATPG as introduced in [21] has been applied. Then, for a set of benchmark circuits (taken from *RevLib* [29] including some of the largest circuits available so far), complete testsets have been generated. To this end, two fault ordering schemes have been applied: (1) the scheme that has been used so far [21] which simply targets faults according to their occurrence within the circuit (from left to right) and (2) the proposed scheme as introduced in Section V. All these experiments have been carried out on an Intel(R) Xeon(R) CPU $\times 4$ with 32 GB main memory.

The results are summarized in Table I. The first columns denote thereby the name of the circuit (denoted by CIRCUIT), the number of gates (denoted by d), the number of lines (denoted by n), and the number of constant inputs (denoted by c). The Column ΔC denotes the maximal difference in the number of control lines at the gates in the respective circuits. Afterwards, the results are distinguished between the numbers obtained when the SMCF model is applied and the numbers obtained when the SMGF is applied. For both cases, the total number of faults to be tested (denoted by $|\mathcal{F}|$) is provided together with the size of the resulting testset (denoted by #TS) and the run-time needed to obtain these results (in CPU seconds; denoted by TIME) when either the existing fault ordering scheme (denoted by PREVIOUS) or the proposed fault ordering scheme (denoted by PROPOSED) is applied. The columns denoted by IMPR. list the improvement (in percent) of the size of the testset obtained by the proposed scheme in comparison to the previously applied scheme. The results are sorted in two parts, based on whether the considered circuits do not include constant inputs (top part of Table I) or whether the considered circuits do include with constant inputs (bottom part of Table I).

First of all, the results show that the different fault ordering schemes hardly affect the run-time of the ATPG flow. Sometimes the previously applied scheme is faster, sometimes the proposed scheme. But in all cases, the difference is rather minor.

However, with respect to the quality, i.e. the size of the resulting testsets, significant improvements can be observed if the proposed scheme is applied. In the clear majority of cases, the proposed scheme leads to much more compact testsets than the previously applied scheme. In the best case, improvements of up to 65% for SMCF and up to 59% for SMGF can be achieved – on average improvements of 13.89% and 14.35% are documented, respectively.

Besides that, the results also clearly confirm the discussion from Section V: The best improvements can be achieved for circuits which gates have a large difference in the number of control lines. For example, the circuit *plus127mod8192_162* is composed of gates with only one control lines (including an "easy" fault) and gates with 13 control lines (including "hard" faults). That is, the "hardness" of the respective faults is significantly different for this circuit. Hence, the proposed fault ordering scheme has a greater impact leading to a considerably improved testset.

VII. CONCLUSION

In this paper, we presented a fault ordering scheme for reversible circuits that explicitly exploits the reversibility of the considered circuits. In fact, we showed that the "hardness" of detection of a fault can easily be derived from the number of control lines in the gate including the fault. Targeting the fault whose gate have a larger number of control lines first during ATPG significantly reduces the size of the testset. In the best case, improvements of up to 65% can be achieved.

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. Oswald, 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] 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.

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

[7] R. Wille, S. Offermann, and R. Drechsler, "SyReC: A programming language for synthesis of reversible circuits," in *Forum on Specification and Design Languages*, 2010, pp. 184–189.

[8] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler, "Synthesis of reversible circuits with minimal lines for large functions," in *ASP Design Automation Conf.*, 2012, pp. 85–92.

[9] 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.

[10] D. M. Miller, R. Wille, and R. Drechsler, "Reducing reversible circuit cost by adding lines," in *Int'l Symp. on Multi-Valued Logic*, 2010, pp. 217–222.

[11] 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.

TABLE I
EXPERIMENTAL RESULTS

CIRCUIT					SMCF						SMGF					
					$\|\mathcal{F}\|$	PREVIOUS		PROPOSED		IMPR.	$\|\mathcal{F}\|$	PREVIOUS		PROPOSED		IMPR.
	d	n	c	ΔC		#TS	TIME(S)	#TS	TIME(S)	(%)		#TS	TIME(S)	#TS	TIME(S)	(%)
CIRCUITS WITHOUT CONSTANT INPUTS																
4_49_16	16	4	0	3	24	8	<0.01	6	<0.01	25.00	16	5	<0.01	3	<0.01	40.00
ham7_104	23	7	0	2	34	6	<0.01	5	<0.01	16.67	23	5	<0.01	4	<0.01	20.00
ham15_108	70	15	0	3	125	10	0.05	9	0.05	10.00	70	10	0.05	9	0.05	10.00
ham15_109	109	15	0	1	126	10	0.08	8	0.06	20.00	109	5	0.03	5	0.04	0.00
ham15_107	132	15	0	7	352	45	0.53	25	0.3	44.44	132	20	0.22	11	0.13	45.00
hwb7_61	236	7	0	2	693	41	0.57	32	0.46	21.95	236	25	0.34	19	0.27	24.00
hwb7_62	331	7	0	5	582	47	0.84	34	0.65	27.66	331	28	0.51	19	0.37	32.14
hwb8_113	637	8	0	6	2214	93	4.47	59	3.47	36.56	637	51	2.33	35	1.91	31.37
plus127mod8192_162	910	13	0	12	5704	786	75.05	272	24.33	65.39	910	275	23.5	112	9.49	59.27
hwb9_119	1544	9	0	6	5812	136	18.26	102	15.09	25.00	1544	80	10.35	53	7.73	33.75
hwb9_123	1959	9	0	7	3596	148	18.55	93	12.82	37.16	1959	72	8.87	51	6.92	29.17
urf3_155	26468	10	0	0	52936	25	61.24	25	64.07	0.00	26468	31	74.03	31	79.47	0.00
CIRCUITS WITH CONSTANT INPUTS																
mini-alu_84	20	10	6	2	27	6	<0.01	6	<0.01	0.00	20	4	<0.01	3	<0.01	25.00
rd84_142	28	15	7	1	49	18	0.04	15	0.03	16.67	28	9	0.01	9	0.01	0.00
4_49_7	42	15	11	2	61	7	0.02	9	0.02	-28.57	42	6	0.01	5	0.01	16.67
hwb5_13	88	28	23	2	131	11	0.13	11	0.12	0.00	88	7	0.07	7	0.07	0.00
hwb6_14	159	46	40	2	241	12	0.39	12	0.40	0.00	159	10	0.33	10	0.33	0.00
ex5p	647	206	198	2	904	24	13.82	19	10.97	20.83	647	20	11.82	21	12.21	-5.00
spla	1709	489	473	1	2711	35	126.22	38	135.03	-8.57	1709	38	139.98	41	147.44	-7.89
table3	1988	554	540	2	2997	41	215.16	49	251.18	-19.51	1988	37	192.27	42	217.63	-13.51
pdc	2080	619	603	2	3135	49	308.83	50	315.81	-2.04	2080	45	295.78	54	361.47	-20.00
alu4	2186	541	527	2	3390	41	210.13	39	192.59	4.88	2186	51	255.35	52	261.65	-1.96
ex1010	2982	670	660	2	4543	31	274.56	29	249.65	6.45	2982	25	220.24	22	185.72	12.00
AVERAGE IMPROVEMENTS										13.89%						14.85%

CIRCUIT: Name of benchmark circuit d: Number of gates n: Number of gates c: Number of constant inputs
SMCF: Results obtained assuming the missing control fault model SMGF: Results obtained assuming the missing gate fault model
PREVIOUS: Results obtained using the existing fault order scheme PROPOSED: Results obtained using the proposed fault order scheme
$|\mathcal{F}|$: Total number of faults to be tested #TS: Size of the resulting testset TIME: Run-time in CPU seconds needed to obtain the results

[12] 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.

[13] J. Seiter, M. Soeken, R. Wille, and R. Drechsler, "Property checking of quantum circuits using quantum multiple-valued decision diagrams," in *Reversible Computation 2012*, ser. Lecture Notes in Computer Science, vol. 7581, 2012, pp. 183–196.

[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] 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.

[16] 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.

[17] 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.

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

[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] H. Zhang, R. Wille, and R. Drechsler, "SAT-based ATPG for reversible circuits," in *Int'l Design & Test Workshop*, 2010.

[21] R. Wille, H. Zhang, and R. Drechsler, "ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization," in *IEEE Annual Symposium on VLSI*, 2011, pp. 120–125.

[22] S. Lee and K. Saluja, "An algorithm to reduce test application time in full scan designs," in *Int'l Conf. on CAD*, 1992, pp. 17–20.

[23] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, "Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits," *IEEE Trans. on CAD*, vol. 14, no. 12, pp. 1496–1504, 1995.

[24] S. Lee, B. Cobb, J. Dworak, M. Grimaila, and M. Mercer, "A new ATPG algorithm to limit test set size and achieve multiple detections of all faults," in *Design, Automation and Test in Europe*, 2002, p. 94.

[25] M. Messing, A. Glowatz, F. Hapke, and R. Drechsler, "Using a two-dimensional fault list for compact automatic test pattern generation," in *Latin-American Test Workshop*, 2009.

[26] 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.

[27] V. D. Agrawal, "An information theoretic approach to digital fault testing," *IEEE Trans. on Comp.*, vol. 30, no. 8, pp. 582–587, 1981.

[28] 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.

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