Determining Minimal Testsets for Reversible Circuits Using Boolean Satisfiability

Hongyan Zhang Stefan Frehse Robert Wille Rolf Drechsler
Institute of Computer Science, University of Bremen
28359 Bremen, Germany
{zhang,sfrehse,rwille,drechsle}@informatik.uni-bremen.de

***Abstract*—Reversible circuits are an attractive computation model as they theoretically enable computations with close to zero power consumption. Furthermore, reversible circuits found significant attention in the domain of quantum computation. With the emergence of first physical realizations for this kind of circuits, also testing issues become of interest. Accordingly, first approaches for automatic test pattern generation have been introduced. However, they suffer either from their limited scalability or do not generate a minimal testset. In this paper, a SAT-based algorithm for the determination of minimal complete testsets is proposed. An experimental evaluation of the proposed method shows that the algorithm is applicable to reversible circuits with more than 2 000 gates.**

I. Introduction

Reversible circuits are n-inputs, n-outputs circuits in which every input pattern maps to a unique output pattern. As a result, computations in reversible circuits can be performed in both directions, i.e. from the inputs to the outputs and vice versa.

The reversibility of computation has intensely been studied as it provides an alternative to conventional circuits where power dissipation can theoretically be reduced or even eliminated [1], [2]. While in conventional logic energy amounting to $kT \cdot \ln 2$ is dissipated for each lost bit of information (where k is the Boltzmann's constant and T is the temperature), reversible circuits are not affected by this. This makes reversible circuits interesting for low-power applications in the future.

Besides that, reversible circuits received significant attention in the domain of quantum computation [3]. Here, qubits instead of bits are applied which cannot only store the conventional Boolean values 1 and 0, but any superposition of them. Because of this, quantum circuits can solve many practical relevant problems significantly faster than their conventional counterparts. Since every quantum computation inherently is reversible, reversible circuits are of interest in this domain.

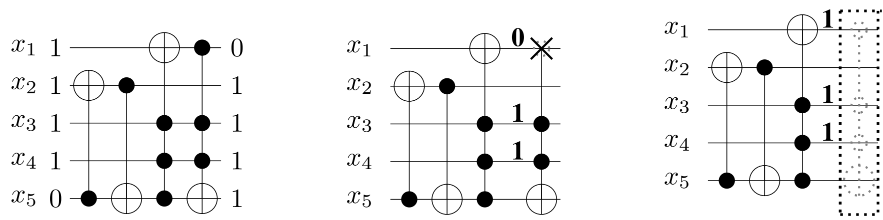
Motivated by these applications, researchers started to develop respective design methods for this new computation model (see e.g. [4], [5], [6], [7], [8]). With the emergence of first physical realizations for this kind of circuits, also testing issues become of interest. In this context, researchers studied different fault models and the respective methods for *Automatic Test Pattern Generation* (ATPG).

While conventional fault models (like the stuck-at fault model) have been considered at the beginning [9], new models addressing physical realizations of reversible circuits have been introduced later [10]. Among them are the missing gate fault model and the missing control line fault model (also known as the partial missing gate model). These fault models remain to be computationally tractable, while at the same time being applicable to different kinds of technologies.

Along with the fault models, researchers also started to develop test pattern generation methods. A major goal is thereby to keep the size of the testset (i.e. the number of test patterns needed to detect all considered faults in a circuit) as small as possible. Different approaches based on greedy and branch-and-bound methods [10], ILP formulations [11] as well as SAT-based approaches [12] and PBO-based methods [13] have been introduced for this purpose. However, they suffer either from their limited scalability (i.e. they are only applicable to circuits with a small number of gates [10], [11]) or do not generate a minimal testset [12], [13].

In this paper, we present an approach which determines a minimal testset for a given reversible circuit. The general idea is to iteratively check, whether for a given circuit and a given fault list a testset detecting all faults with only k patterns exists. By starting these checks with $k = 1$ and iteratively increasing k by one, minimality is ensured. The respective checks are thereby conducted by solvers for Boolean satisfiability. Experiments demonstrate that using the proposed approach, minimal testsets for reversible circuits can efficiently be generated.

The remainder of this paper is structured as follows. The next section introduces the basics on reversible circuits as well as on Boolean satisfiability. Section III introduces the fault models considered in this paper and defines the term of a minimal testset. Afterwards, the proposed approach is described in Section IV. Finally, experiments results are provided in Section V, while Section VI concludes the paper.



(a) Reversible circuit

(b) Single missing control fault (SMCF)

(c) Single missing gate fault (SMGF)

Fig. 1. Reversible circuit with different faults

II. Background

In this section, reversible circuits and the basics of Boolean satisfiability are briefly reviewed.

A. Reversible Circuits

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 a conventional circuits, fanout and feedback are not allowed in reversible circuits [3]. 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 [14]. 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 assigned 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 values x_k with $x_k \in X \setminus \{x_t\}$ always pass the gate unaltered.

Example 1: Fig. 1(a) shows an example of a reversible circuit which is composed of Toffoli gates. This circuit has five circuit lines and four Toffoli gates, i.e. $n = 5$ and $d = 4$. Control lines are denoted by a $\bullet$, while the target line is denoted by an $\oplus$. The annotated values demonstrate the computation of the respective gates for a certain input pattern. In this case, the gates g_2 and g_3 are activated.

B. Boolean Satisfiability

The *Boolean satisfiability* (SAT) problem is defined as follows:

Definition 2: Let h be a Boolean function. Then, the SAT problem is to determine an assignment to all variables of h such that h evaluates to 1 or to prove that no such assignment exists. In the case a satisfying assignment exists, the respective instance is called *satisfiable* (SAT); otherwise the instance is called *unsatisfiable* (UNSAT). Usually, the Boolean formula is thereby given in *Conjunctive Normal Form* (CNF), i.e. in a product-of-sum representation. A CNF consists of a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation.

Example 2: Let h be a Boolean function in CNF with $h = (x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$. Then, $x_1 = 1$, $x_2 = 1$, and $x_3 = 1$ is a satisfying assignment for h. The values of x_1 and x_2 ensure that the first clause becomes satisfied while x_3 ensures this for the remaining two clauses.

Because of its significance in both, theoretical research and practical applications, the SAT problem is one of the intensely studied $\mathcal{NP}$-complete problems. The $\mathcal{NP}$-completeness of the SAT problem has been proven by Cook in 1971 [15]. Despite this complexity, very efficient algorithms and techniques have been developed in order to solve this kind of problems. Besides learning [16] and efficient implication strategies [17] also very powerful term re-writing techniques [18] are applied today. Because of that, nowadays instances including hundreds of thousands of variables and clauses, respectively, can be solved in short time.

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 faults provided in a fault list $\mathcal{F}$. The fault list $\mathcal{F}$ is composed of 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 [11] and defined as follows:

Definition 3: 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. More precisely:

Definition 4: 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 assigned to 1, while the missing control line has to be assigned 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 assigned to 1, i.e. g simply has to be activated. The assignment of the remaining lines can arbitrarily be chosen.

Example 3: 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. Complete and Minimal Testsets

Having a fault list $\mathcal{F}$, a testset should be created that detects as many as possible of the faults provided in $\mathcal{F}$. Moreover, usually a *complete* testset is desired.

Definition 5: A testset $\mathbb{T}_{\mathcal{F}}$ is *complete* with respect to a fault list $\mathcal{F}$ iff each fault $f_i \in \mathcal{F}$ can be detected by applying at least one test pattern of $\mathbb{T}_{\mathcal{F}}$.

If there are no further restrictions (as e.g. constant primary inputs as shown in [13]), determining a complete testset is easy for reversible circuits, since a complete testset can be computed in polynomial time with respect to the the number of faults. In fact, it is sufficient to apply the respective input assignments given in Definition 4 to the gate with the considered fault and, afterwards, simulate this assignment towards the primary inputs. Since reversible circuits have full controllability [19], this can easily be performed. Furthermore, observability is also always ensured, i.e. in case of a fault, the faulty behavior is always shown on at least one primary output.

Applying this procedure to every fault to be considered, a complete testset can be determined. Moreover, since a single test pattern might cover more than one fault, fault simulation can be performed after each test pattern generation. That is, each newly obtained test pattern is simulated and further faults which are additionally detected are removed from the fault list. This obviously reduces the size of the testset.

However, even if determining a complete testset for a reversible circuits is easy, the size of the testset should be kept as small as possible. In fact, a *minimal* testset is desired.

[1]Note that in the literature (e.g. in [11]), the SMCF model is also called *Partial Missing Gate Fault Model*.

Algorithm 1: minATPG$(G, \mathcal{F})$: ALGORITHM DETERMINING A MINIMAL TESTSET.

Input: G a reversible circuit, $\mathcal{F}$ a fault list
Output: A minimal complete testset $\mathbb{T}_{\mathcal{F}}$

```
1  begin
2      k = 1;
3      while true do
4          enc = encode(ATPG(G, F, k));
5          if solve(enc) = UNSAT then
6              k = k + 1;
7          else
8              return T_F obtained from the satisf. assgmt.
9          end
10     end
11 end
```

Definition 6: Let $\mathbb{T}_{\mathcal{F}}$ be a complete testset detecting all faults $f_i \in \mathcal{F}$. The testset $\mathbb{T}_{\mathcal{F}}$ is a *minimal* testset iff there exists no testset $\mathbb{T}'_{\mathcal{F}}$ with $|\mathbb{T}'_{\mathcal{F}}| < |\mathbb{T}_{\mathcal{F}}|$ also detecting all these faults.

Determining a minimal testset is much harder than determining a complete testset. In fact, for certain fault models it has been proven that minimal testset generation is even $\mathcal{NP}$-hard (see e.g. [20]). In this paper, we propose an approach that uses Boolean satisfiability techniques to generate a minimal testset for the SMCF and the SMGF models.

IV. DETERMINING A MINIMAL TESTSET

In this section, we show how a minimal complete testset for a given circuit G and a fault list $\mathcal{F}$ can be determined using techniques for Boolean satisfiability. The general flow is presented first. Afterwards, details on the structure of the proposed SAT instance as well as on the actual encoding are provided.

A. General Flow

In order to determine a minimal testset, an iterative approach is proposed. The basic idea of the flow is as follows: Given a circuit G and a fault list $\mathcal{F}$, first it is checked whether a complete testset exists that consists of $k = 1$ test pattern only. If no such test pattern can be determined, k is increased by one. This procedure is repeated until a testset results which detects all faults provided in $\mathcal{F}$. By iteratively increasing k by one, minimality is ensured.

This procedure is formalized in Algorithm 1. The function *minATPG($G, \mathcal{F}$)* gets the circuit G as well as $\mathcal{F}$ and initializes $k = 1$. Then, in each iteration the question "Does there exist a testset consisting of k patterns and detecting all faults $f \in \mathcal{F}$ in the circuit G?" is encoded as a SAT instance (Line 4) and passed to a solver (Line 5). If the solver returns UNSAT, no

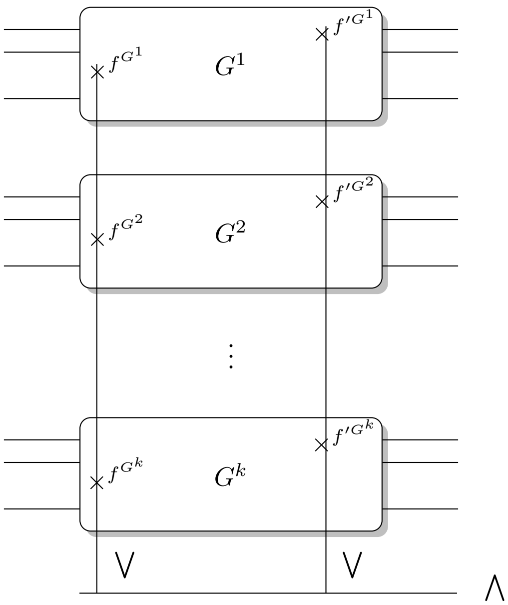


Fig. 2. SAT encoding for a testset of size k

such testset exists and k is increased by one (Line 6). Otherwise (i.e. if the solver returns SAT), the complete minimal testset can be obtained from the satisfying solution (Line 8).

While this represents the main flow of the proposed approach, details on the instance as well as on the concrete encoding are provided in the next sections.

B. Structure of the Instance

According to Algorithm 1, a SAT instance encoding the question "Does there exist a testset consisting of k patterns and detecting all faults $f \in \mathcal{F}$ in the circuit G?" needs to be created. To this end, the considered circuit G is copied k-times so that k different input patterns can be assigned to the circuit's primary inputs (allowing for a testset of size k). Furthermore, for each fault $f \in \mathcal{F}$ and for each circuit copy G^i with $1 \leq i < k$, a variable f^{G^i} is created. This variable is constrained so that f^{G^i} evaluates to 1 (0) if a test pattern is applied to G^i that detects the fault f (that does not detect the fault f). Furthermore, constraints are added ensuring that at least one variable f^{G^i} with $1 \leq i < k$ is assigned to one. By doing so, it is ensured that the considered fault f is detected in at least one circuit copy, i.e. by at least one test pattern.

More formally, the instance

$$\text{ATPG}(G, \mathcal{F}, k) = \bigwedge_{i=1}^{k} G^i \wedge \bigwedge_{f \in \mathcal{F}} (\bigvee_{i=1}^{k} f^{G^i}) \tag{1}$$

is encoded. The general structure of this instance is also depicted in Fig. 2.

Passing this instance to a SAT solver, the solver tries to determine an input assignment for each circuit copy so that each fault is detected in at least one circuit copy. In other words, the SAT solver takes over the task of determining the test patterns. If this is possible, i.e. if the instance is satisfiable, the respective test patterns can easily be obtained from the satisfying assignment of the respective variables. If in contrast the SAT solver returns unsatisfiable, it has been proven that no complete testset with k patterns exists.

C. Encoding of the Instance

In order to encode the instance presented in the last section, a formulation based on the SAT-based ATPG approach proposed in [12] is applied. The encoding of the respective circuit copies as well as the encoding of the respective fault constraints are introduced in the following.

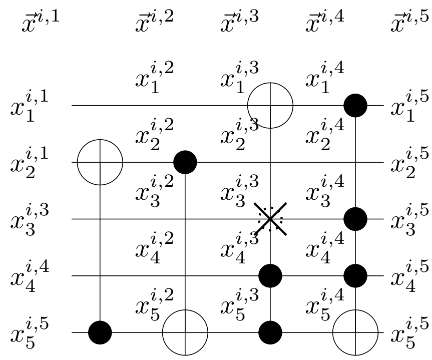
1) Encoding the Circuit Copies G^i: First, the encoding of the circuit copies is presented. Given a reversible circuit G with n lines and d gates, k circuit copies need to be encoded. For each copy G^i with $1 \leq i < k$, variables $\vec{x}^{i,\mu} = x_n^{i,\mu}, x_{n-1}^{i,\mu} \dots x_1^{i,\mu}$ for $\mu \in \{1, \dots, d+1\}$ are introduced representing the assignment to the primary inputs (for $\mu = 1$), the primary output (for $\mu = d+1$) as well as the inputs and outputs of the gates (for $2 \leq \mu \leq d$), respectively. Fig. 3 shows those variables for one copy G^i of the circuit from Fig. 1(a).

In order to model the functionality of the circuit copy, the following constraints are added to the SAT instance for each circuit copy, i.e. for each $i \in \{1, \dots, k\}$:

$$\bigwedge_{j=1}^{d} \bigwedge_{l=1}^{n} x_l^{i,j+1} = \begin{cases} x_l^{i,j}, & \text{if } x_l^{i,j} \text{ represents a control line of gate } g_j \\ x_l^{i,j} \oplus \bigwedge_{x_c \in C_j} x_c, & \text{if } x_l^{i,j} \text{ represents the target line of gate } g_j \\ x_l^{i,j}, & \text{else (i.e. if } x_l^{ij} \text{ represents neither a control line nor a target line of gate } g_k) \end{cases}$$

That is, for each gate g_j in the circuit copy, the respective input/output mapping is constrained (depending on the position of the control and target lines). In other words, the values of all lines (except the target line) are passed through (first and third case), while the output value for the target line is determined depending on the input values of the control and the target line, respectively (second case). The bottom-left part of Fig. 3 shows the respective constraints for one copy G^i of the circuit from Fig. 1(a).

2) Encoding the Faults Variables f^{G^i}: Finally, the encoding of the constraints for the f^{G^i}-variables is presented. As described above, f^{G^i} is supposed to evaluate to 1 (0) if a test pattern is applied to G^i that detects the fault f (that does not detect the fault f). Consequently, just the respective input



Encoding the circuit copy G^i:

$x_1^{i,2} = x_1^{i,1}$ $\quad x_2^{i,2} = x_2^{i,1} \oplus x_5^{i,1}$
$x_3^{i,2} = x_3^{i,1}$ $\quad x_4^{i,2} = x_4^{i,1}$
$x_5^{i,2} = x_5^{i,1}$

$x_1^{i,3} = x_1^{i,2}$ $\quad x_2^{i,3} = x_2^{i,2}$
$\dots$

Encoding f^{G^i} for an SMCF in g_3:
$f^{G^i} = x_4^{i,3} \wedge x_5^{i,3} \wedge \overline{x}_3^{i,3}$

Encoding f^{G^i} for an SMGF in g_3:
$f^{G^i} = x_3^{i,3} \wedge x_4^{i,3} \wedge x_5^{i,3}$

Fig. 3. SAT encoding for an SMCF and an SMGF

assignments for the given fault as provided in Definition 4 has to be applied to the corresponding gate.

More precisely, let $f \in \mathcal{F}$ be an SMCF in a gate $g_j(C_j, t_j)$ with a missing control $x_m \in C_j$. Then, the corresponding f^{G^i}-variable is constrained as follows:

$$f^{G^i} = \left(\bigwedge_{x_c \in C_j \setminus \{x_m\}} x_c^{ij} \right) \wedge \overline{x}_m^{ij}$$

Similarly, let $f \in \mathcal{F}$ be an SMGF in a gate $g_j(C_j, t_j)$. Then, the corresponding f^{G^i}-variable is constrained as follows:

$$f^{G^i} = \bigwedge_{x_c \in C_j} x_c^{ij}$$

The bottom-right part of Fig. 3 shows the respective constraints for an SMCF and an SMGF in gate g_3.

V. Experimental Evaluation

The proposed approach has been implemented in C++ on top of RevKit [21]. Boolector [18] was used as the satisfiability solver. Circuits taken from RevLib [22] were evaluated with respect to both considered fault models. The experiments have been carried out on an AMD Opteron $\times 4$ processor with 3GHz and 32GB main memory running Linux. The timeout was set to 3 600 CPU seconds.

The results have been compared to previously introduced ATPG approaches for reversible circuits, namely the SAT-based approach introduced in [12] and the PBO-based approach introduced in [13], respectively. Both approaches are not intended to compute minimal testsets. However, the difference between the size of the testsets obtained by these approaches and the size of the (minimal) testsets obtained by the proposed approach is investigated.

The results are presented in Table I. The first four columns describe the characteristics of the evaluated circuits, namely (1) the name of the circuit, (2) the number of gates, (3) the number of lines, and (4) the number of constant inputs. Column $|\mathcal{F}|$ denotes the number of faults to be considered for the respective fault model. Afterwards, the results obtained by the previously introduced methods are reported, i.e. the size of the testsets obtained by the SAT-based approach and the size of the testsets obtained by the PBO-based approach, respectively. Finally, the size of the testsets determined by the proposed approach is given in column MINATPG and the required run time is given in column TIMES. Timeouts are reported by *T.O.*, whereas the $>$ denotes the considered testset size before the timeout occurs. This value provides a lower bound.

As can be seen from the results, minimal testsets can efficiently be obtained for circuits including approx. 100 gates. In fact, less than one second is needed for this purpose. Besides that, also larger circuits can be handled. Hence, for the first time, it was possible to generate a minimal testset of a circuit composed of more than 2 000 gates. So far, minimal testsets have been presented only for significantly smaller circuits (e.g. in [11]).

Using these results, also conclusions on the quality of other ATPG methods for reversible circuits can be made. While the size of the test patterns obtained by the SAT-based approach still is way beyond the minimum (e.g. for *ham15_107*, 26 test patterns are generated for the SMGF; in fact, six would be sufficient), the PBO-based approach leads to testsets which are very compact (in fact, the size of these testsets only slightly differs from the minimum).

VI. Conclusion

In this paper, an approach to determine minimal complete testsets for the single missing control fault model and the single missing gate fault model are introduced, respectively. The approach iteratively checks for a testset with a certain size. If there is no testset with the considered number of test pattern, the approach continues with one more test pattern. Finally, by iteratively incrementing the number of test patterns, a minimal complete testset is eventually be determined.

The experimental evaluation shows that the approach is able to handle circuits with more than 2 000 gates for both considered fault models. If the approach aborts due to limited computational resources, a lower bound for the size of the minimal complete testset is provided.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported by the German Research Foundation (DFG) (DR 287/20-1).

TABLE I
RESULT FOR MINIMAL COMPLETE TESTSETS FOR SMCF AND SMGF

				SMCF – *Single Missing Control Fault*					SMGF – *Single Missing Gate Fault*				
Circuit	d	n	c	$\|\mathcal{F}\|$	SAT [12]	PBO [13]	MINATPG	TIME(S)	$\|\mathcal{F}\|$	SAT [12]	PBO [13]	MINATPG	TIME(S)
4gt4-v0_78	13	5	1	18	5	6	4	0.91	13	4	3	2	0.39
4gt12-v0_86	14	5	1	20	4	5	4	0.88	14	3	2	1	0.46
decod24-enable_32	14	9	6	17	3	3	2	0.13	14	2	1	1	0.47
mod5d1_16	15	8	3	19	5	3	2	0.77	15	4	2	1	0.28
4_49_16	16	4	0	24	7	5	4	0.77	16	6	3	2	0.28
miller_5	16	8	5	24	5	4	3	0.7	16	3	2	1	1.14
3_17_6	17	7	4	20	5	5	4	0.74	17	5	2	1	0.53
mini-alu_84	20	10	6	27	6	4	3	0.74	20	5	2	1	0.63
rd53_131	28	7	2	24	11	11	9	1.85	28	5	5	4	0.44
rd84_142	28	15	7	49	16	8	7	0.95	28	9	4	3	0.55
sym6_63	29	14	8	43	11	7	6	0.42	29	7	3	2	0.42
4_49_7	42	15	11	61	7	5	4	0.83	42	6	3	2	0.13
ham15_108	70	15	0	125	9	9	8	1.17	70	10	8	7	0.98
hwb5_13	88	28	23	131	11	7	4	0.46	88	8	4	3	1.10
ham15_109	109	15	0	126	8	9	6	0.35	109	6	4	3	0.52
ham15_107	132	15	0	352	25	16	12	760.63	132	26	7	6	0.59
hwb6_14	159	46	40	241	13	7	6	0.71	159	10	4	3	0.48
ex5p	647	206	198	904	19	18	>12	*T.O.*	647	24	11	9	22.85
spla	1709	489	473	2711	42	19	>13	*T.O.*	1709	34	18	12	2088.79
alu4	2186	541	527	3390	38	18	12	1978.12	2186	40	12	10	1802.56

CIRCUIT: name of the circuit d: number of gates n: number of lines c: number of constant inputs $|\mathcal{F}|$: number of faults to be tested
SAT: number of test patterns obtained by SAT-based ATPG PBO: number of test patterns obtained by PBO-based ATPG
MINATPG: number of test patterns obtained by proposed approach TIME: required run-time in CPU seconds for the proposed approach

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

[4] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, “Reversible logic circuit synthesis,” in *Int’l Conf. on CAD*, 2002, pp. 353–360.

[5] R. Wille, D. Große, G. Dueck, and R. Drechsler, “Reversible logic synthesis with output permutation,” in *VLSI Design*, 2009, pp. 189–194.

[6] D. Maslov, G. W. Dueck, and D. M. Miller, “Techniques for the synthesis of reversible Toffoli networks,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 4, 2007.

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

[8] 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.

[9] 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.

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

[11] 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.

[12] H. Zhang, R. Wille, and R. Drechsler, “SAT-based ATPG for reversible circuits,” in *International Design and Test Workshop*, 2010.

[13] 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.

[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] S. A. Cook, “The complexity of theorem proving procedures,” in *Symposium on Theory of Computing*, 1971, pp. 151–158.

[16] J. P. Marques-Silva and K. A. Sakallah, “GRASP: A search algorithm for propositional satisfiability,” *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.

[17] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an efficient SAT solver,” in *Design Automation Conf.*, 2001, pp. 530–535.

[18] R. Brummayer and A. Biere, “Boolector: An efficient SMT solver for bit-vectors and arrays,” in *Tools and Algorithms for the Construction and Analysis of Systems*, 2009, pp. 174–177.

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

[20] S. Tayu and S. I. S. Ueno, “On the fault testing for reversible circuits,” in *Algorithms and Computation*, ser. LNCS, vol. 4835, 2007, pp. 812–821.

[21] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, “RevKit: A toolkit for reversible circuit design,” in *Workshop on Reversible Computation*, 2010, pp. 69–72, RevKit is available at http://www.revkit.org.

[22] 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.