ATPG for Reversible Circuits Using Simulation, Boolean Satisfiability, and Pseudo Boolean Optimization

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

***Abstract*—Research in the domain of reversible circuits found significant interest in the last years – not least because of the promising applications e.g. in quantum computation and low-power design. First physical realizations are already available, motivating the development of efficient testing methods for this kind of circuits. In this paper, complementary approaches for automatic test pattern generation for reversible circuits are introduced and evaluated. Besides a simulation-based technique, methods based on Boolean satisfiability and pseudo-Boolean optimization are thereby applied. Experiments on large reversible circuits show the suitability of the proposed approaches with respect to different application scenarios and test goals, respectively.**

I. Introduction

Reversible circuits, i.e. circuits performing reversible operations only, build the basis for many emerging technologies that may enhance or even replace conventional computer chips in the future. As the most prominent example, quantum circuits [1] are inherently reversible. With the help of this kind of circuits, many important problems (e.g. factorization or database search) can be solved faster than with conventional methods (see e.g. [2]).

Additionally, even CMOS-based reversible circuits (like the ones introduced in [3]) have some important benefits. 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 information lossless [4], i.e. they are not affected by this [5]. Since with the ongoing miniaturization $kT \cdot \ln 2$ is going to become a crucial value, reversible circuits are considered as a good alternative in particular in domains like low-power design.

As a result, research in the domain of reversible circuits found significant interest in the last years. Different approaches ranging from synthesis (see e.g. [6], [7]), optimization (see e.g. [8], [9]), verification (see e.g. [10], [11]), and debugging (see e.g. [12]) have been introduced. A new circuit model (namely a cascade of reversible gates where no fanout and feedback is allowed [1]) is thereby applied.

However, although large reversible circuits (or quantum circuits, respectively) have not been physically built yet, first promising realizations are already available (see e.g. [13], [3]). Hence, efficient testing methods for this kind of circuits are required. As a result, researchers studied different fault models and the respective methods for *Automatic Test Pattern Generation* (ATPG).

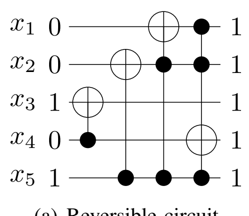
In one of the first studies [14], the stuck-at fault model was thereby applied. Later, it was shown that the validity of the stuck-at fault model is limited for reversible circuits [15]. As a consequence, new models have been introduced: originally, the missing gate fault model [15] followed by the partial missing gate model (also known as missing control line fault model), the repeated gate model, and further ones [16]. These fault models have been shown 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 investigate test pattern generation methods. The focus was thereby on the determination of minimal or, at least, very small testsets. Greedy and branch-and-bound methods [15] as well as ILP formulations [16] have been applied for this purpose. However, these approaches have been evaluated on small circuits only. Furthermore, these approaches are based on the fact that reversible circuits have ideal controllability and observability making ATPG very easy. But, this is not always the case. For example, reversible circuits realizing practical functions (e.g. arithmetic) often include additional constraints. In particular, *constant inputs* are frequently used in this context. Because of this, the controllability may decrease and, thus, complicates ATPG significantly.

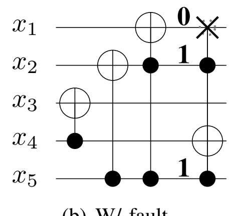
In this paper, we introduce ATPG flows for both, reversible circuits with and without additional constraints. Therefore, three complementary approaches for test pattern generation are applied: a simple one based on simulation and two more elaborated methods utilizing efficient solve engines for *Boolean satisfiability* (SAT) (for which preliminary results previously have been discussed in [17]) and *pseudo-Boolean optimization* (PBO). All of them have their respective benefits and drawbacks with respect to the different application scenarios (with additional constraints or without additional constraints) as well as with respect to the different test goals (compact testsets vs. efficient generation). This is discussed in more detail later in Section IV.

The different performances of the approaches become evident in the experimental evaluation: While the simulation-based approach is quite fast if circuits without additional constraints are considered, it clearly is outperformed otherwise. Then, the SAT-based method provides a better alternative. If not run-time but quality (i.e. the size of the resulting testset) is the crucial factor, the PBO-based approach performs best. To the best of our knowledge, this represents the first evaluation of ATPG with respect to different application scenarios and test goals, respectively.

The rest of this paper is structured as follows: The first section introduces the necessary background including a brief introduction to reversible circuits, test of reversible circuits, as well as SAT and PBO. The proposed ATPG flows are then described in Section III. The actual test pattern generation represents thereby the most crucial and most complex task. The respective simulation-based, SAT-based, and PBO-based methods addressing this task are introduced in Section IV. Finally, experimental results are presented in Section V and conclusions are drawn in Section VI, respectively.



(a) Reversible circuit



(b) W/ fault

Fig. 1. Reversible circuit and circuit with SMCF

II. Background

To keep the remainder of the paper self-contained, this section introduces reversible circuits, provides a brief overview on testing of reversible circuits, and reviews core techniques utilized in this work.

A. Reversible Circuits

A logic function $f{:}\mathbb{B}^n \to \mathbb{B}^m$ over inputs $X = \{x_1, ..., x_n\}$ is *reversible* iff (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. That is, reversible functions represent bijections. Reversible circuits are realizations of reversible functions. A reversible circuit G is a cascade of reversible gates g_i, i.e. $G = g_1 g_2 ... g_d$, where no fanout and feedback is allowed [1]. In this work, we consider the most widely used reversible gate, the Toffoli gate [18].

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

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

That is, if all control line variables x_c are assigned to 1, the target line t is inverted. Under this assignment the gate is called *activated*. All other input values x_k with $k \in X \setminus \{t\}$ pass the gate unaltered. Note that the set of control lines may be empty. In this case, the gate works as a *NOT gate*, i.e. the target line is always inverted.

Example 1: Fig. 1(a) shows a reversible circuit including 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 $\oplus$. The annotated values demonstrate the computation of the respective gates for a certain input pattern. In this case, gates g_2, g_3, and g_4 are activated.

B. 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 testset is called *complete*, if it includes patterns that detect all testable faults under the assumed model. In this paper, we explicitly consider the single missing control faults defined as follows:

Definition 2: Let $g(C, t)$ be a Toffoli gate of a circuit G. Then, a *Single Missing Control Fault* (SMCF) appears if instead of g, a gate $g'(C', t)$ is executed, whereas $C' = C \setminus \{x_i\}$ with $x_i \in C$ and $x_i \neq t$ (i.e. a control line is removed).

In order to detect a fault, the respective gates have to be activated so that the faulty behavior can be observed at the outputs of the circuit. This requires certain input assignments [16]. More precisely, to detect an SMCF at gate $g(C, t)$, 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 be arbitrarily chosen.

Example 2: Fig. 1(b) illustrates an SMCF which can occur in the reversible circuit previously introduced in Fig. 1(a). The respective assignment needed to detect this fault is also given.

Note that the approach presented in this paper can be extended for other fault models as well (in particular to the single additional control fault model or the single missing gate fault model). Then, just the assignments to the faulty gate have to be adjusted. However, due to page limitations, in the following we focus on SMCFs.

The aim of automatic test pattern generation is to determine a compact, but complete, testset including as less as possible test patterns.

C. SAT and PBO

Solvers for *Boolean satsifiability* (SAT) and *pseudo-Boolean optimization* (PBO) are core technologies utlized in this work for the purpose of ATPG. Both problems are defined as follows:

Definition 3: The *Boolean satisfiability problem* determines an assignment to the variables of a Boolean function $\Phi : \{0, 1\}^n \to \{0, 1\}$ such that Φ evaluates to 1 or proves that no such assignment exists. The function Φ is thereby given in *Conjunctive Normal Form* (CNF). Each CNF is a set of clauses where each clause is a set of literals and each literal is a propositional variable or its negation.

Definition 4: The *pseudo-Boolean optimization problem* determines a satisfying solution for a pseudo-Boolean function $\Psi : \{0, 1\}^n \to \{0, 1\}$ which – at the same time – minimizes an objective function $\mathcal{F}$. The *pseudo-Boolean function* Ψ is thereby a conjunction of constraints defined by $\sum_{i=1}^{n} c_i \dot{x}_i \geq c_n$, where $c_1 \ldots, c_n \in \mathbb{Z}$ and $\dot{x}_i$ either is a positive or a negative literal. The *objective function* $\mathcal{F}$ is defined by $\mathcal{F}(x_1, \ldots, x_n) = \sum_{i=1}^{n} m_i \dot{x}_i$ with $m_1, \ldots, m_n \in \mathbb{Z}$.

Example 3: Let $\Phi = (x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$. Then, $x_1 = 1, x_2 = 1$, and $x_3 = 1$ is a satisfying assignment solving the SAT problem.

Accordingly, let $\Psi = (2x_1 + 3x_2 + \overline{x}_3 \geq 3)(2x_1 + x_2 \geq 2)$ and $\mathcal{F} = x_1 + x_2 + x_3$. Then, $x_1 = 1, x_2 = 0$, and $x_3 = 0$ is a solution to the PBO problem, satisfying Ψ and at the same time minimizing $\mathcal{F}$.

Both, SAT and PBO, are well investigated problems. In the past efficient solving algorithms (so called *SAT solvers* or *PBO solvers*, respectively) have been proposed (see e.g. [19], [20]). Instead of simply traversing the complete space of assignments, intelligent decision heuristics, powerful learning schemes, and efficient implication methods are thereby applied. In case of PBO, it is also common to translate the respective instance into a sequence of SAT instances in order to efficiently determine a solution [21]. In the following, we apply these techniques as black boxes delivering the solution for the proposed ATPG problem formulations.

III. ATPG Flows

This section introduces two flows for ATPG of reversible circuits. The aim is to provide an algorithm that generates a complete testset covering all faults depending on a given fault model (stored in a *fault list*). Two different criteria are thereby considered: On the one hand, the testsets should be as small as possible. On the other hand, the results should be available as fast as possible.

As already shown for conventional circuits, *controllability* and *observability* are thereby crucial factors. The former criterion defines the ability to establish a certain signal value by setting values at the primary inputs. The latter one defines the ability to determine a certain signal value by observing

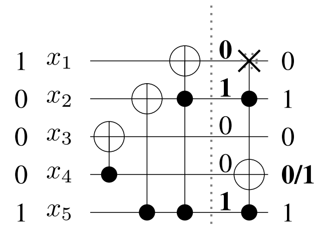


Fig. 2. Exploiting observability & controllability

the primary output values. Due to the reversibility, both tasks can easily be performed in general reversible circuits. In fact, given a fault for which a test pattern should be generated, the values of the primary inputs (primary outputs) triggering this fault (showing the fault) can easily be obtained be performing the computations in the respective direction.

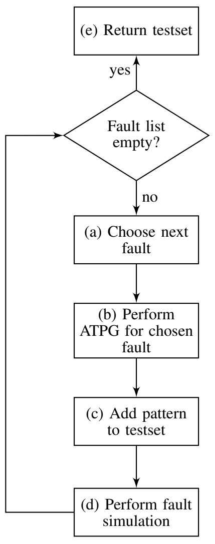
Example 4: Consider the circuit shown in Fig. 2 including an SMCF at gate g_4. In order to detect this fault, a test pattern is required which establishes the signal value 1 at all control lines of g_4 and the signal value 0 at the missing control line, respectively. Such a pattern can easily be derived by simulating an appropriate input assignment to g_4 (e.g. 01001 in Fig. 2) backwards to the inputs of the circuit (leading to 10001 in Fig. 2). Then, the faulty behavior can be detected at the outputs of the circuits (instead of the desired output 01001, the faulty output 01011 is generated in the example of Fig. 2).

Exploiting this observation, an ATPG flow for reversible circuits as shown in Fig. 3(a) can be applied to generate a complete testset. As long as there are faults, which are not covered by the already determined test patterns, a new fault is selected (Step (a)). Then, for this fault a test pattern as described above is created (Step (b)) and added to the testset (Step (c)). Since this test pattern might cover further faults, fault simulation is performed next, i.e. the pattern is simulated and further faults which are detected by this are removed from the fault list (Step (d)). This procedure is repeated until no faults are left, i.e. until the fault list is empty. Then, the generated patterns form the complete testset (Step (e)).

However, this flow can only be applied if observability and controllability in reversible circuits remain ideal. But, this may not always be the case. For example, reversible circuits realizing practical functions (e.g. arithmetic) often include additional constraints. In particular, *constant inputs* are frequently applied in this context. Because of this, the controllability may decrease and, thus, complicates Step (b) of the described flow. Moreover, in some cases even untestable faults may result.

Example 5: Consider the circuit in Fig. 4. The two constant inputs make it impossible to detect the highlighted missing control fault at gate g_4. In fact, all possible assignments 01001, 01011, 01101, and 01111 needed to activate g_4 (and therewith the faulty behavior) cannot be established since the corresponding primary input patterns 10001, 10111, 10101, and 10011 conflict with the constant input assignments.

Consequently, an extended ATPG flow (shown in Fig. 3(b)) has to be applied if reversible circuits with additional constraints are considered. This flow differs from the previous one by the fact that Step (b) is supposed either to create a test pattern which satisfies the additional constraints or to prove that no such test pattern exists. If a valid test pattern has been obtained, the flow continues as described above. If in contrast, the fault has been proven to be untestable, the fault is moved from the fault list to a list of untestable faults (Step (f)). This list can be used later e.g. to optimize the considered circuit.


(e) Return testset
yes
Fault list empty?
no
(a) Choose next fault
(b) Perform ATPG for chosen fault
(c) Add pattern to testset
(d) Perform fault simulation
(a) W/o add. constraints

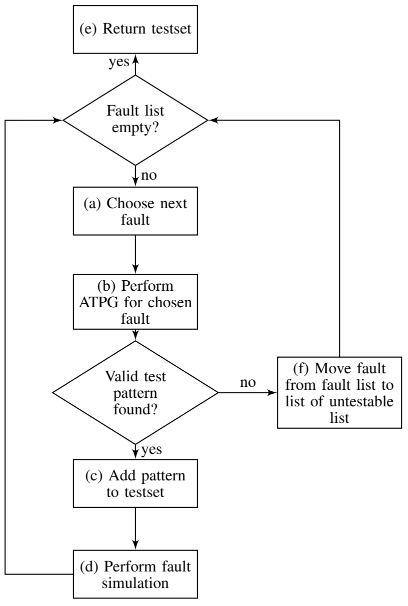

(e) Return testset
yes
Fault list empty?
no
(a) Choose next fault
(b) Perform ATPG for chosen fault
Valid test pattern found?
no
(f) Move fault from fault list to list of untestable list
yes
(c) Add pattern to testset
(d) Perform fault simulation
(b) W/ add. constraints

Fig. 3. ATPG flow for reversible circuits

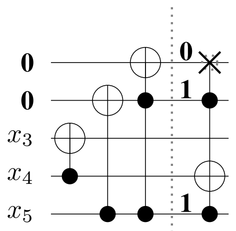


Fig. 4. Circuit with untestable fault

IV. Test Pattern Generation

In both flows covered in the previous section, Step (b), i.e. the actual test pattern generation, represents the most crucial and most complex task. In the following, three complementary methods for this task are described: a simulation-based method, a SAT-based method, and a PBO-based method. All of them have their respective benefits and drawbacks with respect to the test goal (compact testsets vs. efficient generation) as well as with respect to the additional constraints.

A. Simulation-based

As illustrated in Example 4, test pattern generation is easy if circuits without additional constraints are considered. Then, the faulty behavior simply is activated by an appropriate input assignment which is subsequently propagated towards the primary inputs and the primary outputs. Using simulation, this can be done in linear time with respect to the number of gates the circuit is composed of. In fact, this provides a very simple and efficient way to generate test patterns.

In contrast, if additional constraints have to be considered, simulation reaches its limits quite fast. Then, the obtained test patterns have to be validated against constant input values. If this validation fails, a different test pattern needs to be generated. This process continues until either a valid one is determined or no further appropriate test patterns can be generated (and the fault has been classified to be untestable).

Example 6: Consider the circuit shown in Fig. 5 including an SMCF at gate g_4. In order to activate the faulty behavior,

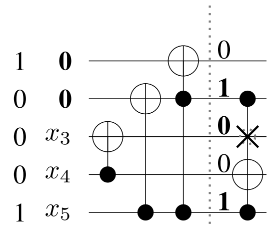


Fig. 5. Simulation-based ATPG

first the input assignments 01001 and 01011 are applied to g_4 and propagated towards the inputs. Both lead to invalid test patterns contradicting the constant input $x_1 = 0$ and $x_2 = 0$. Not before the input assignment 11001 is applied, a valid test pattern, namely 00001, can be obtained.

In the worst case, this procedure amounts to an exponential number of simulation runs. In case of the SMCF model, $2^{n-|C|}$ different patterns exist for a given fault (where $|C|$ is the number of control lines of the considered gate). This is, because the values of the control lines are fix, but the assignment of the remaining $n - |C|$ lines can be arbitrarily chosen in order to detect the fault. In particular for larger functions with $n \gg 1$, this is crucial. Thus, simulation provides an efficient solution for the ATPG problem only if reversible circuits without additional constraints are considered. Beyond that, no special treatment ensuring a compact testset is provided by this method.

B. SAT-based

The exponential behavior of ATPG for reversible circuits with additional constraints cannot be avoided. However, by using efficient solving techniques, this process can be accelerated. To this end, an alternative is proposed which makes use of SAT solvers. Therefore, the ATPG problem is reformulated as a SAT instance asking "Is there a valid assignment to all primary inputs of the given circuit so that the considered faulty behavior is triggered?".

To encode this, a Boolean function Φ over the following variables is created:

- $\vec{x}^1 = x_n^1, x_{n-1}^1 \dots x_1^1$ representing the assignment to the respective primary inputs of the circuit,
- $\vec{x}^{d+1} = x_n^{d+1}, x_{n-1}^{d+1} \dots x_1^{d+1}$ representing the assignment to the primary outputs of the circuit, and
- $\vec{x}^k = (x_n^k x_{n-1}^k \dots x_1^k)$ with $2 \leq k \leq d$ representing the input (output) assignment to the respective gate g_k (g_{k-1}).

Example 7: Consider the reversible circuit depicted in Fig. 6(a). The variables needed to encode the ATPG problem for this circuit as a SAT instance are annotated to the respective gate inputs and outputs.

Having these variables, the ATPG problem is formulated as the conjunction of the following three constraints. First, the functionality of the given circuit is encoded, i.e. the constraint

$$\bigwedge_{k=1}^{d} \bigwedge_{i=1}^{n} x_i^{k+1} = \begin{cases} x_i^k, & \text{if } x_i^k \text{ represents a control line of gate } g_k \\ x_i^k \oplus \bigwedge\limits_{x_c \in C_k} x_c, & \text{if } x_i^k \text{ represents the target line of gate } g_k \\ x_i^k, & \text{else (i.e. if } x_i^k \text{ represents neither a control line nor a target line of gate } g_k) \end{cases}$$

is added to the SAT instance. That is, for every gate $g_k(C_k, t_k)$ in the circuit, 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 ($x_i^{k+1} = x_i^k$), while the output value for the target line is determined depending on the input values of the control and the target line ($x_i^{k+1} = x_i^k \oplus \bigwedge_{x_c \in C_k \setminus \{x_i^k\}} x_c$).

Afterwards, constraints are added ensuring that the faulty behavior is activated. In the case that a test pattern for an SMCF at the ith line of gate $g_k(C_k, t_k)$ should be generated, the constraint

$$(x_i^k = 0) \wedge (\bigwedge_{x_c \in C_k \setminus \{x_i^k\}} x_c = 1)$$

is added. For the remaining cases, these constraints are applied accordingly. Other fault models can be supported by using corresponding assignments.

Finally, the additional constraints are added to the instance. Here, we consider constant inputs. That is, for each constant input, constraints are added, ensuring that the respective variable x_i^1 is set to the corresponding value.

Example 7 (continued): Using the variables introduced in Fig. 6(a), a SAT instance is created asking for a test pattern which detects the SMCF at gate g_4. Therefore, the three constraints as shown in Fig. 6(b) are added.

Afterwards, all these constraints are converted into CNF – the common input format for SAT solvers. Since only Boolean operations like equality, AND, or XOR are used, this can be done quite easily.

If the solver determines a satisfying assignment for the resulting instance, a valid test pattern can be obtained from the assignment to $x_1^1 \dots x_n^1$. If in contrast the SAT solver returns unsatisfiable, it has been proven that no test pattern considering the additional constraints exists – the respective fault is untestable under these constraints.

In comparison to the simulation-based method, this approach makes full use of modern solving techniques. Thus, in case of a reversible circuit with additional constraints, a valid test pattern can be generated much more efficiently. Nevertheless, also the SAT-based method does not provide a special treatment ensuring a compact testset.

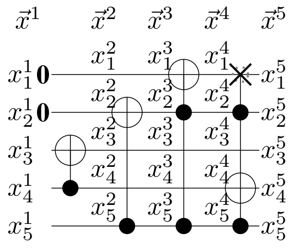
C. PBO-based

To address the demand for a compact testset, another ATPG method is introduced. Therefore, we adjust the problem for which a solution should be obtained: Instead of determining a test pattern which detects one particular fault, an approach is introduced which generates a test pattern detecting as many faults as possible. PBO solvers are utilized for this task.

All the variables already introduced in Fig. 6(a) for SAT-based ATPG are thereby applied again. The same holds for most of the constraints. In fact, the functional constraints and the additional constraints are reused. But, in order to encode the faults, new variables and constraints are introduced. In case of the SMCF model, for each undetected fault a new variable $flt_i^{g_k}$ and the constraint

$$flt_i^{g_k} = (x_i^k = 0) \wedge (\bigwedge_{x_c \in C_k \setminus \{x_i^k\}} x_c = 1)$$

is added. That is, $flt_i^{g_k}$ is set to 1 if an input assignment is applied which detects a missing control fault at line i at gate g_k. Otherwise, this variable is set to 0.



$\vec{x}^1$ $\vec{x}^2$ $\vec{x}^3$ $\vec{x}^4$ $\vec{x}^5$
x_1^1 0
x_2^1 0
x_3^1
x_4^1
x_5^1
x_1^2 x_2^2 x_3^2 x_4^2 x_5^2
x_1^3 x_2^3 x_3^3 x_4^3 x_5^3
x_1^4 x_2^4 x_3^4 x_4^4 x_5^4
x_1^5 x_2^5 x_3^5 x_4^5 x_5^5

(a) Variables

Functional constr.:

$x_1^2 = x_1^1$ $\quad x_2^2 = x_2^1$

$x_3^2 = x_3^1 \oplus x_4^1$ $\quad x_4^2 = x_4^1$

$x_5^2 = x_5^1$

$x_1^3 = x_1^2$ $\quad x_2^3 = x_2^2 \oplus x_5^2$

$\dots$

Fault constraints:

$x_1^4 = 0$

$x_2^4 = 1$

$x_5^4 = 1$

Additional constr.:

$x_1^1 = 0$

$x_2^1 = 0$

(b) SAT constraints

Functional constr. and add. constr. are reused from the SAT formulation.

Fault constraints:

$flt_4^{g_1} = (x_4^1 = 0)$

$\dots$

$flt_5^{g_4} = (x_1^4 = 1) \wedge (x_2^4 = 1) \wedge (x_5^4 = 0)$

Objective function:

$\overline{flt_4^{g_1}} + \overline{flt_5^{g_2}} + \overline{flt_2^{g_3}} + \overline{flt_5^{g_3}} + \dots + \overline{flt_5^{g_4}}$

(c) PBO constraints

Fig. 6. SAT and PBO formulation for a SMC fault

Example 8: Consider again the circuit depicted in Fig. 6(a). To formulate the ATPG problem as a PBO instance, the functional constraints and the additional constraints from Fig. 6(b) are reused. In addition, the fault constraints as shown in Fig. 6(c) are added.

Accordingly, variables and constraints can be added for other fault models as well.

Having this instance, the objective function need to be formulated. Recall that a test pattern should be generated which detects as many faults as possible. Each $flt_i^{g_k}$ variable set to 1 represents a fault to be detected under the current assignment. Thus, the PBO problem to be solved is to determine a satisfying solution for the proposed instance which maximizes the number of variables $flt_i^{g_k}$ set to 1. Using common PBO solvers, this is formulated in the objective function as follows:

$$\mathcal{F} = \sum_{k=1}^{d} \sum_{x_i \in C} \overline{flt_{c_i}^{g_k}}$$

Note that – according to the definition of the PBO problem (see Section II-C) – PBO solvers try to find a satisfying solution which *minimizes* $\mathcal{F}$. Due to the inversion of the the respective $flt_{c_i}^{g_k}$ variables, the solver in fact maximizes $\mathcal{F} = \sum_{k=1}^{d} \sum_{x_i \in C} flt_{c_i}^{g_k}$.

Example 8 (continued): For the considered example, the objective function as shown in Fig. 6(c) is applied. Using this, a test pattern is obtained which detects as many faults represented by $flt_{c_i}^{g_k}$ as possible.

Similar to the SAT-based approach, all these constraints finally are converted into the proper format and passed to the respective solve engine. Since PBO solvers do not work with Boolean formulas provided in CNF, all constraints need to be encoded in terms of pseudo-Boolean constraints. This is straight-forward: Each clause $\dot{x}_{i1} \vee \dot{x}_{i2} \vee \dots \vee \dot{x}_{ij}$ of the CNF simply is replaced with an equivalent constraint $\dot{x}_{i1} + \dot{x}_{i2} + \dots + \dot{x}_{ij} \geq 1$.

Then, the PBO solver determines a result from which a test pattern can be derived. Additionally, the concrete faults which are detected by this pattern can be obtained from the assignment to the corresponding $flt_{c_i}^{g_k}$-variables. In the next iterations, all $flt_{c_i}^{g_k}$-variables representing detected faults (as well as their constraints) are then removed from both, the formula and the objective function. That is, the proposed formulation is applied to the remaining faults only. In doing so, Step (a) and Step (d) of the flow presented in Fig. 3 can be omitted. The whole ATPG flow terminates either if test patterns for all faults have been generated or if the instance becomes unsatisfiable. In the latter case, all faults still included in the fault list are then classified to be untestable.

Overall, the PBO-based approach eventually provides an alternative to the previously introduced methods. As also confirmed by the experiments discussed in the next section, quite compact testsets are generated for both, circuits with additional constraints and circuits without additional constraints. In contrast, the complexity and therewith the run-time increases due to the fact that in addition to the determination of a satisfying assignment also an objective function is minimized.

V. Experimental Evaluation

This section presents experimental results obtained by the proposed approaches. Therefore, all methods have been implemented in C++. *MiniSAT* [19] and *clasp* [20] are applied as solve engine for the SAT-based approach and the PBO-based approach, respectively. As benchmarks, reversible circuits from RevLib [22] have been taken including some of the largest realizations available so far. Evaluations have been performed considering the single missing control fault model, the single additional control fault model, and the single missing gate fault model. But due to page limitations, only the results obtained with the SMCF model are reported. However, the drawn conclusions hold for the other models as well. The experiments have been carried out on an AMD Phenom II $\times 4$ with 8 GB main memory. The timeout (denoted by TO) was set to 3600 CPU seconds.

Table I provides the results. The first columns denote thereby the name of the circuit (Circuit), the number of gates (d), the number of lines (n), the number of constant inputs (c), and the number of possible faults to be tested (#F). Afterwards, the size of the resulting testset (#TS), the number of untestable faults (#UT), and the run-time in CPU seconds needed to obtain these results (Time) are reported for all proposed approaches. In case of a timeout, Column #TS reports the number of patterns generated so far along with the percentage of faults covered by them.

Two different goals are considered in ATPG: quality (i.e. determining an as compact as possible testset) and run-time. If the former one is the crucial factor, the PBO-based approach leads to the best results. In fact, for most of the benchmarks, testsets with the smallest number of patterns are obtained (exceptions are *ham7_104, ham15_109, 4gt4-v0_78, 4gt12-v0_86*). On average, 33.26% less test patterns in comparison to the SAT-based approach and 46.41% less test patterns in comparison to the simulation-based approach are observed, respectively. If in contrast run-time is the main criteria, a more divergent picture results. Then, the simulation-based approach clearly outperforms the other approaches if circuits without additional constraints are considered. In fact, all benchmarks can be handled in just a few seconds, while the SAT-based and PBO-based approach need some minutes or even timeout. But, as also discussed in Section IV-A, simulation is not efficient for circuits with additional constraints. Here, the SAT-based approach is the better alternative.

TABLE I
EXPERIMENTAL RESULTS

Circuit	d	n	c	#F	SIMULATION-BASED #TS	#UT	TIME(S)	SAT-BASED #TS	#UT	TIME(S)	PBO-BASED #TS	#UT	TIME(S)
W/o additional constraints													
4_49_16	16	4	0	24	7	0	<0.01	7	0	0.01	5	0	0.03
ham7_104	23	7	0	34	6	0	0.01	5	0	0.01	6	0	0.03
0410184_169	46	14	0	49	10	0	0.02	12	0	0.08	3	0	0.06
ham15_108	70	15	0	125	10	0	0.02	9	0	0.13	9	0	0.21
ham15_109	109	15	0	126	9	0	0.02	8	0	0.23	9	0	0.27
ham15_107	132	15	0	352	43	0	0.06	25	0	0.76	16	0	0.82
hwb7_61	236	7	0	693	46	0	0.14	34	0	1.78	27	0	2.61
hwb7_62	331	7	0	582	57	0	0.18	36	0	1.83	29	0	4.64
hwb8_113	637	8	0	2214	93	0	0.77	57	0	6.25	44	0	28.27
plus127mod8192_162	910	13	0	5704	232	0	4.39	269	0	29.20	104	0	1280.72
hwb9_119	1544	9	0	5812	136	0	2.80	101	0	16.62	82	0	245.42
hwb9_123	1959	9	0	3596	138	0	2.58	91	0	29.08	79	0	439.90
urf3_155	26468	10	0	52936	26	0	7.86	25	0	60.78	(89.63)5	0	TO
W/ additional constraints													
one-two-three-v0_97	11	5	2	23	4	9	0.01	3	9	0.02	3	9	0.02
4gt4-v0_78	13	5	1	18	8	0	<0.01	5	0	<0.01	6	0	0.02
4gt12-v0_86	14	5	1	20	7	0	<0.01	4	0	<0.01	5	0	0.02
decod24-enable_32	14	9	6	17	3	0	0.08	3	0	<0.01	3	0	0.02
mod5d1_16	15	8	3	19	3	0	<0.01	5	0	<0.01	3	0	0.02
graycode6_11	15	11	5	10	3	0	0.03	1	0	<0.01	1	0	<0.01
miller_5	16	8	5	24	6	0	0.02	5	0	<0.01	4	0	0.03
3_17_6	17	7	4	20	5	0	0.01	5	0	<0.01	5	0	0.03
mini-alu_84	20	10	6	27	6	0	0.78	6	0	0.01	4	0	0.03
rd53_131	28	7	2	24	10	0	0.01	11	0	<0.01	11	0	0.11
rd84_142	28	15	7	49	12	0	185.47	16	0	0.03	8	0	0.14
sym6_63	29	14	8	43	7	0	17.36	11	0	0.03	7	0	0.10
4_49_7	42	15	11	61	7	0	268.18	7	0	0.01	5	0	0.10
hwb5_13	88	28	23	131	(35.11)1	–	TO	11	0	0.07	7	0	0.24
hwb6_14	159	46	40	241	(35.68)1	–	TO	13	0	0.31	7	0	0.60
sym9_148	210	10	1	756	50	0	0.16	23	0	0.92	14	0	1.79
alu_8	453	91	64	730	(30.82)1	–	TO	27	48	9.70	9	48	54.42
ex5p	647	206	198	904	(30.97)1	–	TO	19	0	11.40	18	0	32.77
spla	1709	489	473	2711	(31.83)1	–	TO	42	0	272.87	19	0	719.77
apex2	1746	498	459	2787	(30.10)1	–	TO	70	0	148.96	(54.36)1	–	TO
table3	1988	554	540	2997	(33.83)1	–	TO	49	0	242.64	23	0	2089.34
pdc	2080	619	603	3135	(34.32)1	–	TO	49	0	297.46	(78.31)3	–	TO
alu4	2186	541	527	3390	(28.67)1	–	TO	38	0	189.11	18	0	905.58
ex1010	2982	670	660	4543	(32.16)1	–	TO	27	0	226.65	25	0	841.68

CIRCUIT: name of the circuit d: number of gates n: number of lines c: number of constant inputs #F: number of faults to be tested
#TS: number of patterns in the determined testset #UT: number of untestable faults TIME: required run-time in CPU seconds
In case of a timeout, Column #F reports the number of patterns generated so far along with the percentage of faults covered by them.

VI. CONCLUSION

In this paper, approaches for automatic test pattern generation for reversible circuits have been introduced and evaluated. Circuits without and with additional constraints are thereby considered. The proposed methods make use of simulation as well as solvers for Boolean satisfiability and pseudo-Boolean optimization. As shown by the experiments, each of these techniques is suitable for a particular application or test goal, respectively. While the simulation-based approach is quite fast if circuit without additional constraints are considered, it clearly becomes outperformed otherwise. Then, the SAT-based method provides a better alternative. If instead the size of the testset is the major criterion, the PBO-based approach performs best – but also requires larger run-time.

ACKNOWLEDGEMENT

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

REFERENCES

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

[2] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Foundations of Computer Science*, pp. 124–134, 1994.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[9] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] H. Zhang, R. Wille, and R. Drechsler, "SAT-based ATPG for reversible circuits," in *Int'l Design & Test Workshop*, 2010, pp. 149–154.

[18] 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.

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

[20] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *Int'l Joint Conference on Artificial Intelligence*, 2007, pp. 386–392.

[21] N. Eén and N. Sörensson, "Translating pseudo-Boolean constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[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.