Acknowledgements

This thesis was written during my work as a PhD student in the research group for computer architecture at the University of Bremen.

In particular, I want to thank Prof. Dr. Rolf Drechsler and Dr. Robert Wille for their guidance and support. I am sure it would have not been possible without their help.

My special thanks go to Prof. Dr. Hans-Jörg Kreowski for his willingness to review my dissertation.

Moreover, I want to thank all members of the research group for their contributions, academic and non-academic.

I also want to thank the company enago for proofreading.

Last but not least, I want to express my gratitude to my parents, my husband, and my 5-years-old son for their never ending support and love.

Bremen, January 2013

Contents

Chapter 1

Introduction

Computer technology is one of the most brilliant inventions of human beings. In less than 50 years, with the rapid development and wide application of computer technology, the world has changed so greatly that it is difficult to imagine a day without computers. As the heart of any computing device, microprocessors are becoming more and more complex and powerful. Modern microprocessors are almost exclusively made of transistors based on CMOS technology. Thanks to the progress in semiconductor and manufacturing processes, the number of transistors in a computer doubles every 18 months, and at the same time, the cost of this extra computing power decreases exponentially (Moore's law). Despite the rapidly increasing number of transistors, the size of transistors continues to decrease. While the first computer ENIAC included 17,468

vacuum tubes, weighting 27 t [Kop02], an Intel Core i5 processor has 995 million transistors confined to an area of 37.5 × 37.5 mm [Wik12]. However, this trend based on traditional CMOS technology, will end one day (maybe very soon) because of the limitations imposed by physics and manufacturing facilities. Furthermore, with increasing numbers of transistors, power consumption and high failure rates also increase and become crucial issues in designing high-performance digital circuits.

To meet the demands for more computational power, alternatives are needed that go beyond the scope of traditional (CMOS) technologies. Reversible logic marks a promising new direction in which all operations are performed in an invertible manner. That is, in contrast to traditional logic, all computations can be reverted. A simple standard operation such as the logical OR illustrates that reversibility is not guaranteed in traditional circuit systems. Indeed, it is possible to obtain the inputs of an OR gate if the output is assigned 0 (for then both inputs must be assigned 0). However, it is not possible to determine the input values if the OR output is assigned 1. In contrast, reversible logic allows only bijective operations, that is, n-input – n-output functions map each possible input vector into an unique output vector.

The original motivation for studying reversible circuits is the possibility of nearly energy-free computation. In [Lan61], it is shown that traditional irreversible circuits necessarily dissipate energy owing to the

erasure of information. However, it is possible to perform reversible computations with arbitrarily small energy dissipation (as shown in [Lan61; Ben73; FT82]). This makes reversible computation an attractive alternative that forms as the basis for emerging technologies and has wide application in quantum computation, low-power design, optical and DNA computing, and nanotechnologies.

Although reversible logic has been considered for decades (see e.g., [Lan61; Tof80; Per85]), it recently got new impetus from the introduction of the first physical realizations of computing machines based on this paradigm. For example, in [VSB+01], a first (small) quantum circuit (which is inherently reversible) was introduced that can solve the factorization problem in polynomial time; for conventional circuits, solutions require exponential time. In [DV02], a first reversible circuit based on conventional CMOS technology was presented.

In comparison to traditional logic, the new computation paradigm of reversible logic causes certain difficulties, but it also enables certain simplifications. For example, fanout and feedback are not directly allowed in reversible logic. This makes the design of reversible circuits harder and requires alternative design methods. Different approaches including synthesis (see e.g., [MMD03; SPMH03; WD09; WOD10]), optimization (see e.g., [FTM08; MWD10]), verification (see e.g., [VMH07; WLTK08]), and debugging (see e.g., [WGF+09]) have been introduced.

Although all this is still basic research, these advancements will in future pose key challenges for testing. As has already been proven for conventional circuits, test costs increase steadily with each new generation of chip. With the increasing complexity of integrated circuits, testing has become one of the most expensive and time-consuming tasks in circuit design. Test costs can now amount to 40% of the overall product cost [WWW06]. More effective test technologies have become key to success in today's competitive markets. Furthermore, emerging technologies have higher failure rates compared to conventional CMOS technologies. For example, in quantum technology, quantum states are fragile and error-prone owing to their nanoscale interactions with the environment (decoherence) [NC00]; therefore, efficient testing methods are essential for successful implementations of quantum circuits. Hence, to manufacture reversible circuits, efficient testing methods are required. For this purpose, researchers have studied different fault models [PHM04; PFBH05] as well as methods for *Automatic Test Pattern Generation* (ATPG) [HPB04; PFBH05] and *fault diagnosis* [RTPP04; PBP05]. However, in all the previous studies, only simple reversible circuits were considered, and the work is just beginning.

This thesis contributes to efficient testing methods (ATPG and fault diagnosis) for reversible circuits. Formal methods like Boolean satisfiability are exploited. In the following, the contributions are briefly intro-

duced in the order in which they appear in this thesis. More detailed descriptions of the problems and proposed solutions are given at the beginning of each chapter.

Chapter 2 - Preliminaries: To keep the thesis self-contained, preliminaries are provided in this chapter. Reversible circuits and the respective fault models are described. Then, the basic concepts of satisfiability solvers are briefly explained.

Chapter 3 - Automatic Test Pattern Generation: In this chapter, a current ATPG method is reviewed. The discussion about applicability leads to the introduction of a new ATPG flow. An ATPG method using efficient solver engine is proposed for improving the new ATPG flow.

Chapter 4 - Advanced Automatic Test Pattern Generation: More complex ATPG methods are introduced for generating compact or even minimal testsets. Furthermore, with respect to different application scenarios and test goals all proposed ATPG approaches are evaluated in this chapter.

Chapter 5 - Testing Multiple Faults: Testing multiple faults for reversible circuits is discussed and a testing flow is provided. ATPG meth-

ods with and without explicit consideration of multiple faults are proposed by using solver engines. Moreover, an efficient approach is introduced for fault masking checking using formal methods.

Chapter 6 - Fault Diagnosis: In comparison with conventional methods of fault diagnosis, improved approaches are presented for reversible circuits. The efficiencies of the approaches are demonstrated using experimental results.

Chapter 7 - Conclusion: The thesis is summarized in this chapter and conclusions are presented.

The main ideas in this thesis have already been published or are expected to be published in the following articles.

- Chapter 3:

  H. Zhang, R. Wille and R. Drechsler. SAT-based ATPG for Reversible Circuits. In *5th International Design & Test Workshop (IDT)*, pp. 149-154, Abu Dhabi, 2010.

- Chapter 4:

  R. Wille, H. Zhang and R. Drechsler. Fault Ordering for Automatic Test Pattern Generation of Reversible Circuits. In *International*

Symposium on Multiple-Valued Logic (ISMVL), Toyama, 2013.

R. Wille, H. Zhang and R. Drechsler. ATPG for Reversible Circuits using Simulation, Boolean Satisfiability, and Pseudo Boolean Optimization. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 120-125, Chennai, 2011.

H. Zhang, S. Frehse, R. Wille and R. Drechsler. Determining Minimal Testsets for Reversible Circuits using Boolean Satisfiability. In *10th IEEE Africon,* Livingstone, 2011.

- Chapter 5:

  H. Zhang, R. Wille and R. Drechsler. Testing of Multiple Faults for Reversible Circuits. 2012. In preparation.

- Chapter 6:

  H. Zhang, R. Wille and R. Drechsler. Improved Fault Diagnosis for Reversible Circuits. In *Asian Test Symposium (ATS)*, New Delhi, 2011.

Chapter 2

Preliminaries

To keep this thesis self-contained, this chapter provides basic definitions and notations. The chapter starts by describing reversible circuits and the respective fault models. This provides the basis for all approaches described in the thesis. Because many of the proposed methods exploit satisfiability solvers, the basic concepts of these techniques are reviewed. All descriptions are brief; for more in-depth treatments, references are given in the appropriate sections.

2.1 Background

2.1.1 Reversible Circuits

A logic function $f{:}\mathbb{B}^n \rightarrow \mathbb{B}^m$ 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., $G = g_1 g_2 ... g_d$, in which no fanout and feedback are allowed [NC00]. In this study, we consider the most widely used reversible gate, the Toffoli gate [Tof80].

Definition 2.1. 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 function

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

That is, if all control line variables x_c are assigned 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.

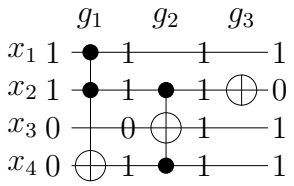


FIGURE 2.1: Reversible circuit

Example 2.1. *Fig. 2.1 shows a reversible circuit containing four circuit lines and three Toffoli gates, so* $n = 4$ *and* $d = 3$*. Control lines are denoted by a* $\bullet$*, while the target line is denoted by a* $\oplus$*. The annotated values demonstrate the computation of the respective gates for a certain input pattern. In this case, gates* g_1 *and* g_2 *are activated and* g_3 *is a NOT gate.*

2.1.2 Fault Models

Because reversible circuits are still an emerging concept, the actual manufacturing technologies used for their physical implementation are still open. Consequently, it must be understood that any fault model discussed today is necessarily speculative. However, the fault models described today have interesting properties that may turn out to be useful for defining and analyzing future generations of fault models that consider specific technology details. In one of the first studies [PHM04], the stuck-at fault model was applied. Later, it was shown that for reversible circuits, the validity of the stuck-at fault model is limited [HPB04]. As

a consequence, new models have been introduced: originally, the missing gate fault model was introduced [HPB04], followed by the partial missing gate model (also known as the missing control line fault model), the repeated gate model, and the multiple missing gates fault model [PFBH05]. These fault models have been shown to be computationally tractable, and at the same time, applicable to different types of technologies. In this study, we explicitly consider the fault models introduced in the following definitions.

Definition 2.2. 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).

Definition 2.3. Let $G = g_1 g_2 ... g_d$ be a reversible circuit.

1. A *Multiple Missing Gate Fault* (MMGF) occurs if s SMGFs appear simultaneously with $0 < s \leq d$ (d is the number of gates in the circuit). Note that the s SMGFs are not restricted to only consecutive gates like the definition of MMFG in [PFBH05].

[1]Note that in the literature (e.g., in [PFBH05]), the SMCF model is also called a *Partial Missing Gate Fault Model*.

2. A *Multiple Missing Control Fault* (MMCF) occurs if s SMCFs appear simultaneously with $0 < s \leq \sum |C|$, $\sum |C|$ is the sum of all control lines in the reversible circuit.

To detect a single fault (e.g., an SMCF or SMGF), the respective gates have to be activated so that the faulty behavior shows up at the outputs of the circuit. Depending on the fault model being considered, this requires certain input assignments [PFBH05].

Definition 2.4. 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 1, while the missing control line has to be assigned 0. The assignment of the remaining lines can be chosen arbitrarily.

2. To detect an SMGF in g (i.e., the disappearance of g), all control lines in C have be assigned to 1, i.e., g has to be activated. The assignment of the remaining lines can be chosen arbitrarily.

Testing multiple faults is more complex than testing single faults. In Chapter 5 the detection of multiple faults for reversible circuits will be discussed more precisely.

Example 2.2. *Fig. 2.2 illustrates an SMCF that occurs in the reversible circuit introduced in Fig. 2.1. The control line at the second line of* g_2 *is*

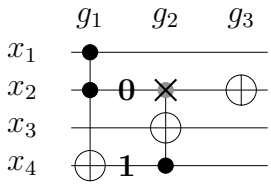


FIGURE 2.2: Single Missing Control Fault (SMCF)

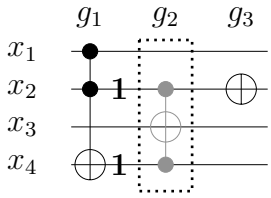


FIGURE 2.3: Single Missing Gate Fault (SMGF)

missing. To detect this SMCF, the assignment before the second line and the fourth line of g_2 have to be 0 *and* 1*, respectively.*

Example 2.3. *Fig. 2.3 illustrates an SMGF. In the reversible circuit introduced in Fig. 2.1, the second gate g_2 is missing. To detect this SMGF, the second line and the fourth line of g_2 have to be set to* 1*.*

Example 2.4. *Fig. 2.4 illustrates an MMCF, which occurs in the reversible circuit introduced in Fig. 2.1. The control line at the first line of g_1 and the control line at the second line of g_2 are missing. In other words, the two SMCFs appear simultaneously.*

Example 2.5. *Fig. 2.5 illustrates an MMGF. In the reversible circuit introduced in Fig. 2.1 the first gate g_1 and the third gate g_3 are missing simultaneously.*

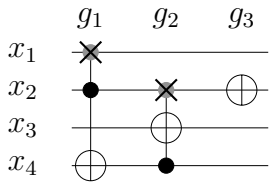


FIGURE 2.4: Multiple Missing Control Fault (MMCF)

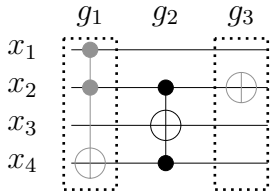


FIGURE 2.5: Multiple Missing Gate Fault (MMGF)

2.2 Satisfiability Solvers

Solvers for the *Boolean satisfiability problem* (SAT problem) [DGP97] and the *pseudo-Boolean optimization problem* (PBO problem) [Chr80] are core technologies utilized in this thesis. The SAT problem is one of the central *NP*-complete problems [Coo71]. Despite this proven complexity, efficient solving algorithms have been developed, which have found great success as proof engines for many practical problems. Examples occur in the domain of automatic test pattern generation [Lar92; TED10], debugging [SVAV05] and verification [BCCZ99; PBG05]. The PBO problem is a generalization of the SAT problem. The real world problem is transformed to a SAT or a PBO instance (in most cases in *Conjunctive Normal Form* (CNF)). Afterward, a solver is applied to calculate a

solution, that is, to find a satisfying assignment to the input variables.

Both problems are defined as follows:

Definition 2.5. The *Boolean satisfiability problem* (SAT problem) determines whether there exists an assignment to the variables of a Boolean function $\Phi : \{0,1\}^n \to \{0,1\}$, such that Φ evaluated to be 1 or to prove 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 2.6. The *pseudo-Boolean optimization problem* (PBO problem) determines a satisfying solution for a pseudo-Boolean function $\Psi : \{0,1\}^n \to \{0,1\}$, which simultaneously minimizes an objective function $\mathcal{O}$. The *pseudo-Boolean function* Ψ is thereby a conjunction of constraints defined by $\sum_{i=1}^{n} c_i x_i \geq c_n$, where $c_1 \ldots, c_n \in \mathbb{Z}$ and x_i either is a positive or a negative literal. The *objective function* $\mathcal{O}$ is defined by $\mathcal{O}(x_1, \ldots, x_n) = \sum_{i=1}^{n} m_i x_i$ with $m_1, \ldots, m_n \in \mathbb{Z}$.

Example 2.6. *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* $\mathcal{O} = x_1 + x_2 + x_3$. *Then,* $x_1 = 1, x_2 = 0$, *and* $x_3 = 0$ *is a solution*

Algorithm 1: Solving algorithm in modern SAT solvers

```
begin
    while decision( ) do
        propagation();
        if conflict( ) then
            if conflict_analysis() then
                backtrack();
            end
            else
                return UNSAT;
            end
        end
    end
    return SAT;
end
```

to the PBO problem, satisfying Ψ*, and at the same time, minimizing* $\mathcal{O}$*.*

Both SAT and PBO are well investigated problems. In recent years, efficient solving algorithms (called *SAT solvers* or *PBO solvers*) have been proposed (see e.g., [DP60; DLL62; ES04; SS05; GKNS07]). Most of them apply the steps described in Algorithm 1. To begin, a decision (*decision()*) is made by choosing a variable for which a new value is assigned. Then, implications due to this assignment are determined (*propagation()*). This may cause a conflict (*conflict()*), so *conflict_analysis( )* is then carried out. If the conflict can be resolved by undoing assignments from previous decisions, *backtrack()* is done. Otherwise, the instance is unsatisfiable. If no further decisions can be made, i.e., all variables are assigned and no conflicts occur, then all clauses are satisfied and the

problem is solved.

Instead of simply traversing the complete space of assignments, intelligent decision heuristics [GN02], powerful learning schemes and efficient implication methods [MS99; MCZ+01] are applied. In the 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 [ES06]. In this thesis, we apply these techniques as black boxes for obtaining solutions to the proposed testing problems.

Chapter 3

Automatic Test Pattern Generation

Automatic Test Pattern Generation (ATPG) is the task of generating effective test patterns that detect faults in a circuit according to a given fault model. This chapter deals with ATPG issues for reversible circuits. To exploit the properties of reversible circuits new ATPG methods are proposed.

Early studies addressed some of the ATPG issues for reversible circuits (see e.g., [PHM04; PFBH05]). In one of the first studies, Patel et al. considered the stuck-at fault model [PHM04]. They applied *Integer Linear Programming* (ILP) and decomposition techniques to generate minimal or at least small testsets that detect all stuck-at faults in a given

reversible circuit. Later, it was shown that for reversible circuits, the validity of the stuck-at fault model is limited [HPB04]. As a consequence, new models have been introduced, e.g., SMGF and SMCF [PFBH05] (Section 2.2). These fault models have been shown to be computationally tractable, and at the same time, applicable to different types of technologies. Along with them, new methods for ATPG have been introduced.

Because test pattern generation is generally easy for reversible circuits (as explained in Section 3.1), the focus is on the determination of a minimal or, at least, small testset. Greedy and branch-and-bound methods [HPB04] as well as ILP formulations [PFBH05] have been applied for this purpose. While the greedy approaches are very fast (but do not guarantee minimal results), the exact approaches suffer from high runtimes, i.e., they are only applicable to small circuits.

However, in all the previous studies, only simple reversible circuits were considered. Moreover, additional constraints often have to be considered. In particular, *constant inputs* frequently occur. They are needed to realize irreversible functions [MD04]. Furthermore, synthesis of more complex functionality is so far only possible if significant number of constant inputs are available (see e.g., [FTR07; WD09]).

In this chapter, it is illustrated why ATPG for reversible circuits with constant inputs is difficult and why in such cases, previous ap-

proaches cannot be efficiently applied. Then, an alternative method using solvers for *Boolean satisfiability* (SAT) [ES04] is proposed. The general idea is motivated by SAT-based ATPG for conventional circuits, which serves as a complementary alternative [Lar92; DEF+08]. Experiments demonstrate that by using SAT-based ATPG, complete testsets for reversible circuits with constant inputs can be efficiently generated, while the respective simulation-based approach (inspired by a previous study) may often timeout. The proposed *SAT-based* ATPG has been published in [ZWD10].

In the following, Section 3.1 briefly reviews the current ATPG method and it discusses the applicability of this approach when additional constraints (like constant inputs) have to be considered. This leads to the introduction of a new ATPG flow in Section 3.2. In Section 3.3, this ATPG flow is then improved by the SAT-based approach. Finally, experimental results are presented in Section 3.4 and conclusions are drawn in Section 3.5.

3.1 Simulation-based ATPG

In this section, test generation for reversible circuits is applied by a simulation-based ATPG. In comparison with conventional irreversible circuits, it can be shown that testsets can be efficiently determined by

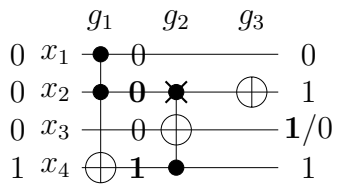


FIGURE 3.1: ATPG by simulation without constant inputs

simulations. Furthermore the limits of the simulation-based ATPG will be presented for situation in which constant inputs must be considered.

As already shown for testing of conventional circuits, *controllability* and *observability* are crucial factors [Agr81]. *Controllability* defines the ability to establish a certain signal value by setting values at the primary inputs. *Observability* defines the ability to determine a certain signal value by observing the primary output values. Both tasks can be easily performed in general reversible circuits owing to reversibility. 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 by performing the computations in the respective direction – test patterns can easily be determined just by simulations.

Example 3.1. *Consider the circuit shown in Fig. 3.1, which includes an SMCF at gate* g_2*. To detect this fault, a test pattern is required that establishes the signal value* 0 *at the missing control line of* g_2 *and the signal value* 1 *at all other control lines. Such a pattern can easily be*

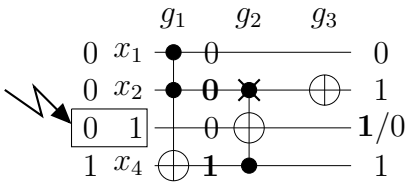


FIGURE 3.2: ATPG by simulation with constant inputs

derived by simulating an appropriate input assignment to g_2 (e.g., 0001 *in Fig. 3.1) backwards to the primary input of the circuit (leading to* 0001 *in Fig. 3.1). Then, the faulty behavior can be detected at the primary output of the circuit (instead of the desired output* 0101*, the faulty output* 0111 *is generated in the example of Fig. 3.1).*

This works well so long as no additional constraints are assumed. However, this is often not the case. In particular, *constant inputs* are frequently used in reversible circuits. They enable the realization of reversible circuits for irreversible functions [MD04]. Moreover, synthesis of reversible circuits for large functions is so far not possible without using constant inputs (see e.g., [FTR07; WD09]). As a consequence circuits realizing large functions have a significant number of additional constraints in terms of constant inputs. This makes ATPG much more difficult, as illustrated in the following example.

Example 3.2. *Consider the circuit shown in Fig. 3.2, which includes an SMCF at gate g_2. This circuit has a constant input. Applying the same simulation-based procedure as in Example 3.1 may result in a conflicting*

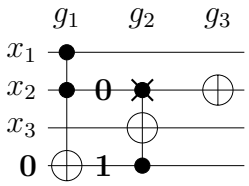


FIGURE 3.3: Circuit with an untestable fault

test pattern (e.g., with x_3 assigned 0, which contradicts the assumption that x_3 is a constant input assigned 1). Thus, an alternative test pattern has to be determined.

However, to obtain a test pattern for a certain fault that is consistent with additional constraints (like constant inputs), an exponential number of possibilities must be checked in the worst case. More precisely, if $|C|$ denotes the number of control lines at the considered gate, $2^{n-|C|}$ different patterns have to be considered for an SMCF in the worst case, because the values of the control lines are fixed, but the assignment of the remaining $n - |C|$ lines can be arbitrarily chosen to detect the fault. Accordingly, $2^{n-|C|}$ patterns for an SMGF have to be checked in the worst case. Thus, when considering additional constraints, the originally easy task of ATPG for reversible circuits becomes much more difficult.

Furthermore, it is even possible that for certain faults in a circuit with constant inputs, no appropriate test pattern can be determined at all. For example, in Fig. 3.3, the constant input at x_4 makes it impossible to detect the highlighted missing control fault at gate g_2. In fact, none of

the possible assignments 0001, 0011, 1001, and 1011 needed to activate g_2 and the faulty behavior can be established, because the corresponding primary input patterns 0001, 0011, 1001, and 1011 conflict with the constant input assignments. In this case, the fault is called *untestable*[1]. Analogous to conventional circuits, if a fault has been confirmed to be untestable, the circuit can be simplified. For example in the circuit of Fig. 3.3, the control connection at the second line at gate g_2 can be removed.

3.2 ATPG Flow

Considering the additional constraints, a new ATPG flow for reversible circuits is presented in Fig. 3.4.

To generate a complete testset, all faults to be considered are stored in a fault list (Step (a)). As long as the fault list is not empty (i.e., faults exist that are not yet detected by already determined test patterns), a new fault is chosen (Step (b)). Afterward, ATPG for this fault is performed (Step (c)). If this could be successful (i.e., if a valid test pattern incorporating additional constraints could be obtained), the generated pattern is added to the testset (Step (d)). Additionally, fault simulation

[1]Note that the decision whether a fault is untestable is often not as simple as in the example of Fig. 3.3. If a test pattern for a fault of a gate g_k with $k \gg 1$ should be determined, it is significantly harder to show the untestability.

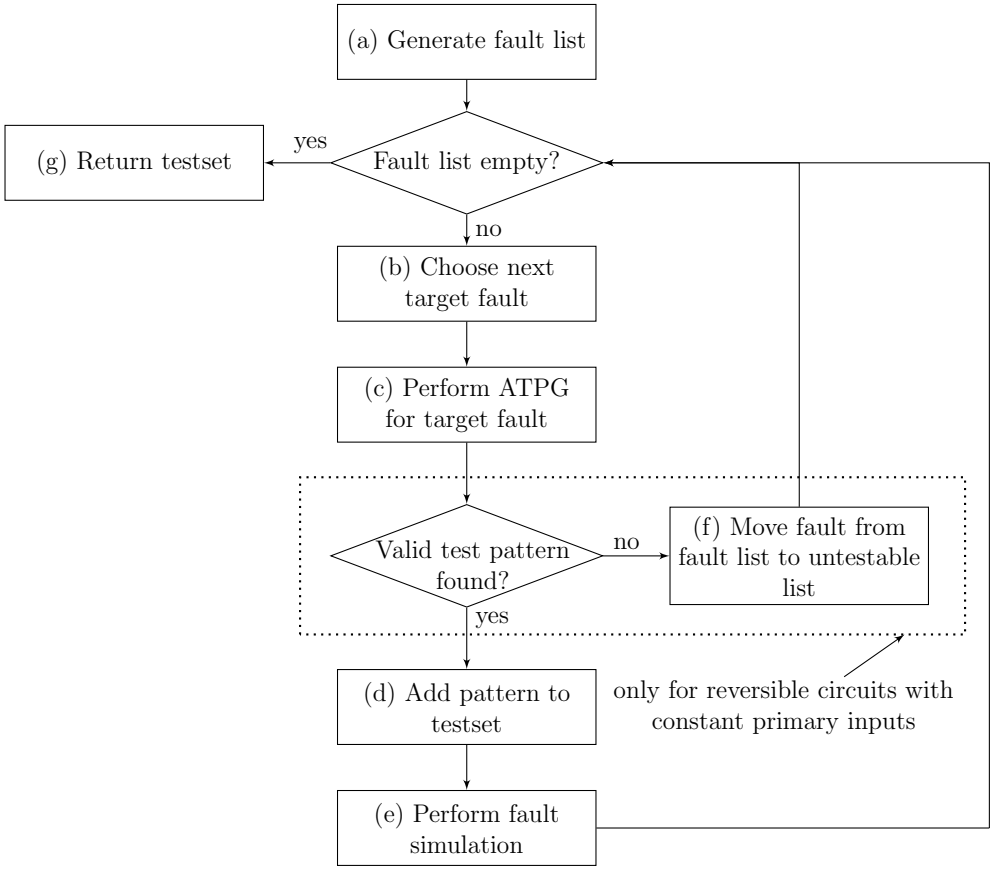


FIGURE 3.4: Test generation flow for reversible circuits

is performed, i.e., the pattern is simulated for all remaining faults and further faults detected by this simulation are removed from the fault list (Step (e)). 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 (f)). This list can be later used, e.g., to optimize the circuit. If all faults have been classified, the process terminates (Step (g)). By applying this flow, a complete testset is generated

for all testable faults under a certain fault model.

In contrast to circuits without constant inputs, untestable faults may occur in circuits with constant inputs, i.e., a valid test pattern does not exist. Hence, the decision of a valid test pattern in Step (c) is only necessary for circuits with constant inputs because an untestable fault may exist only in such circuits (Step (f)). Therefore, the area within the dotted lines in Fig. 3.4 is required only for reversible circuits with constant inputs.

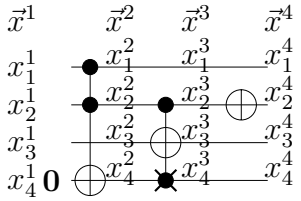
3.3 SAT-based ATPG

The exponential behavior of ATPG for reversible circuits with additional constraints cannot be avoided. However, by using efficient solving techniques, the process can be accelerated. An alternative is then proposed by using SAT solvers. Therefore, the ATPG problem is reformulated as a SAT instance by 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:

- let $\vec{x}^1 = x_n^1, x_{n-1}^1 \dots x_1^1$ represent the assignment to the respective primary inputs of the circuit,



(a) Variables

(1) Functional constraints:

$x_1^2 = x_1^1, \quad x_2^2 = x_2^1, \quad x_3^2 = x_3^1, \quad x_4^2 = x_4^1 \oplus (x_1^1 \wedge x_2^1)$

$x_1^3 = x_1^2, \quad x_2^3 = x_2^3, \quad x_3^3 = x_3^2 \oplus (x_2^2 \wedge x_4^2), \quad x_4^3 = x_4^2$

$x_1^4 = x_1^3, \quad x_2^4 = \overline{x_2^3}, \quad x_3^4 = x_3^3, \quad x_4^4 = x_4^3$

(2) Constant inputs constraints:

$x_4^1 = 0$

(3) Fault constraints:

$x_4^2 = 0, \;\; x_2^2 = 1$

(b) SAT constraints

FIGURE 3.5: SAT formulation for a SMC fault

- let $\vec{x}^{d+1} = x_n^{d+1}, x_{n-1}^{d+1} \ldots x_1^{d+1}$ represent the assignment to the primary outputs of the circuit, and

- let $\vec{x}^k = (x_n^k x_{n-1}^k \ldots x_1^k)$ with $2 \leq k \leq d$ represent the input (output) assignment to the respective gate g_k (g_{k-1}).

Example 3.3. *Consider the reversible circuit in Fig. 3.5(a). The variables needed to encode the ATPG problem of this circuit as a SAT instance are annotated to the respective gate inputs and outputs.*

With these variables, the ATPG problem is formulated as the conjunction of the following three constraints.

1. Functional constraints: 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. The respective input/output mapping is constrained (depending on the position of the control and target lines) for every gate $g_k(C_k, t_k)$ in the circuit. 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 lines.

$$x_i^{k+1} = x_i^k \oplus \bigwedge_{x_c \in C_k \setminus \{x_i^k\}} x_c.$$

2. Constant inputs constraints: the additional constraints of the constant primary inputs are added to the instance. For each constant primary input constraints are added to ensure that the respective variable x_i^1 is set to the corresponding value.

3. Fault constraints: constraints are added to ensure that the faulty behavior is triggered. When 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. In other remaining cases, these constraints are applied accordingly. Other fault models can be supported in corresponding assignments (Section 2.1.2).

Example 3.3. *(continued)*
Using the variables introduced in Fig. 3.5(a), a SAT instance is created by asking for a test pattern that detects the SMCF at gate g_2. Therefore, the three groups of constraints shown in Fig. 3.5(b) are added.

All these constraints are then converted into CNF, which is the common input format for SAT solvers. This can be done very easily (see e.g., [Tse68]), because only Boolean operations such as equality, AND, or XOR are used. The following example shows this in detail.

Example 3.4. *x_1, x_2, and x_3 are three variables. Boolean operations such as equality, AND, and XOR can be converted into CNF.*

- *The equality operation $x_1 = x_2$ can be converted into CNF:*

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) = 1$$

- *The AND operation $x_1 \wedge x_2 = x_3$ can be converted into CNF:*

$$(\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3}) = 1$$

- *The XOR operation $x_1 \oplus x_2 = x_3$ can be converted into CNF:*

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee \overline{x_3}) = 1$$

If the solver then determines a satisfying assignment for the resulting instance, a valid test pattern can be obtained from the assignment to $x_1^1 \ldots x_n^1$. At the same time, other faults detected by this pattern can also be obtained from this assignment (this substitutes Step (e) in the ATPG flow from Fig. 3.4). In contrast, if the SAT solver returns an unsatisfiable result, then it has been proven that no test pattern exists: the respective fault is untestable under these constraints.

Generally, using a SAT solver promises a more efficient solution. Instead of a naive enumeration of all possible test patterns, the SAT solver makes full use of modern solving techniques. For example, during the search process, conflict-based learning is exploited, i.e., reasons for conflicts are learned. This learned information prevents the solver from re-entering no-solution search space, i.e., large parts of the search space are pruned. In contrast, this information is not available when each pattern is separately checked. As a result, SAT-based ATPG for circuits with constant inputs is more efficient than the simulation-based approach. This is also confirmed by the experiments described in the next section.

3.4 Experimental Results

Both the simulation-based (Section 3.1 and Fig. 3.4) and the SAT-based ATPG (Section 3.3) for circuits with constant inputs have been implemented on top of RevKit [SFWD10] in C++. For the latter method, MiniSat [ES04] was used as the underlying SAT solver. The two approaches were evaluated using circuits from RevLib [WGT+08]. For both methods, the target fault was simply chosen according to its occurrence within the circuit (from left to right). Complete testsets for the SMCF and SMGF models were determined. The experiments were carried out on an Intel(R) Xeon(R) CPU ×4 with 32 GB of main memory. The

TABLE 3.1: Experimental results for simulation-based and SAT-based ATPG (Single Missing Control Fault Model)

CIRCUIT	d	n	c	$\|\mathcal{F}\|$	SIMULATION		SAT		#UT
					#TS	TIME(S)	#TS	TIME(S)	
WITHOUT CONSTANT PRIMARY INPUTS									
4_49_16	16	4	0	24	7	<0.01	8	<0.01	0
ham7_104	23	7	0	34	6	<0.01	6	<0.01	0
ham15_108	70	15	0	125	12	<0.01	10	0.05	0
ham15_109	109	15	0	126	8	<0.01	10	0.08	0
ham15_107	132	15	0	352	51	0.06	45	0.53	0
hwb7_61	236	7	0	693	61	0.18	41	0.57	0
hwb7_62	331	7	0	582	66	0.22	47	0.84	0
hwb8_113	637	8	0	2214	122	0.84	93	4.47	0
plus127*	910	13	0	5704	372	7.88	786	75.05	0
hwb9_119	1544	9	0	5812	121	2.45	136	18.26	0
hwb9_123	1959	9	0	3596	137	2.44	148	18.55	0
urf3_155	26468	10	0	52936	27	8.09	25	61.24	0
WITH CONSTANT PRIMARY INPUTS									
mini-alu_84	20	10	6	27	6	0.3	6	<0.01	0
rd84_142	28	15	7	49	14	138.94	18	0.04	0
4_49_7	42	15	11	61	7	120.43	7	0.02	0
hwb5_13	88	28	23	131	>1	*T.O.*	11	0.13	0
hwb6_14	159	46	40	241	>1	*T.O.*	12	0.39	0
alu_8	453	91	64	730	>1	*T.O.*	43	15.07	48
ex5p	647	206	198	904	>1	*T.O.*	24	13.82	0
spla	1709	489	473	2711	>1	*T.O.*	35	126.22	0
table3	1988	554	540	2997	>1	*T.O.*	41	215.16	0
pdc	2080	619	603	3135	>1	*T.O.*	49	308.83	0
alu4	2186	541	527	3390	>1	*T.O.*	41	210.13	0
ex1010	2982	670	660	4543	>1	*T.O.*	31	274.56	0

Circuit: name of the circuit; d: number of gates; n: number of lines; c: number of constant inputs; $|\mathcal{F}|$: total number of faults to be tested; #UT: the number of untestable faults; plus127*: plus127mod8192_162

TABLE 3.2: Experimental results for simulation-based and SAT-based ATPG (Single Missing Gate Fault Model)

CIRCUIT	d	n	c	$\|\mathcal{F}\|$	SIMULATION		SAT		#UT
					#TS	TIME(S)	#TS	TIME(S)	
WITHOUT CONSTANT PRIMARY INPUTS									
4_49_16	16	4	0	16	9	<0.01	5	<0.01	0
ham7_104	23	7	0	23	4	<0.01	5	<0.01	0
ham15_108	70	15	0	70	11	<0.01	10	0.05	0
ham15_109	109	15	0	109	8	<0.01	5	0.03	0
ham15_107	132	15	0	132	25	0.03	20	0.22	0
hwb7_61	236	7	0	236	26	0.07	25	0.34	0
hwb7_62	331	7	0	331	22	0.07	28	0.51	0
hwb8_113	637	8	0	637	48	0.31	51	2.33	0
plus127*	910	13	0	910	73	1.01	275	23.5	0
hwb9_119	1544	9	0	1544	66	1.14	80	10.35	0
hwb9_123	1959	9	0	1959	60	1.01	72	8.87	0
urf3_155	26468	10	0	26468	23	6.42	33	60.64	0
WITH CONSTANT PRIMARY INPUTS									
mini-alu_84	20	10	6	20	4	0.06	4	<0.01	0
rd84_142	28	15	7	28	9	185.4	9	0.01	0
4_49_7	42	15	11	42	5	221.83	6	0.01	0
hwb5_13	88	28	23	88	>1	*T.O.*	7	0.07	0
hwb6_14	159	46	40	159	>1	*T.O.*	10	0.33	0
alu_8	453	91	64	453	>1	*T.O.*	39	7.63	4
ex5p	647	206	198	647	>1	*T.O.*	20	11.82	0
spla	1709	489	473	1709	>1	*T.O.*	38	139.98	0
table3	1988	554	540	1988	>1	*T.O.*	37	192.27	0
pdc	2080	619	603	2080	>1	*T.O.*	45	295.78	0
alu4	2186	541	527	2186	>1	*T.O.*	51	255.35	0
ex1010	2982	670	660	2982	>1	*T.O.*	25	220.24	0

Circuit: name of the circuit; d: number of gates; n: number of lines; c: number of constant inputs; $|\mathcal{F}|$: total number of faults to be tested; #UT: number of untestable faults; plus127*: plus127mod8192_162

timeout (denoted by $T.O.$) was set to 3600 CPU seconds.

The results for the SMCF and SMGF models are summarized in Tables 3.1 and 3.2, respectively. The first columns give the name of the circuit (denoted by CIRCUIT including the ID as used in RevLib), the number of gates (denoted by d), the number of lines (denoted by n), and the number of constant inputs (denoted by c). The remaining columns give the number of possible faults (denoted by $|\mathcal{F}|$), the number of resulting patterns in the complete testset (denoted by #TS), the number of untestable faults (denoted by #UT), and the required runtime to obtain them (in CPU seconds and denoted by TIME).

The discussion in the previous section is confirmed by the results for runtime (Column TIME). For circuits with constant inputs, it is much more efficient to use the SAT-based method to perform ATPG than to use the simulation-based method. In many cases, complete testsets can be achieved by the SAT-based method, whereas the simulation-based approach may timeout for circuits having a significant number of constants.

3.5 Summary

Because of high observability and controllability, test generation is not complicated for reversible circuits. Testsets can be generated even with a simulation-base ATPG. However, when constant inputs have to be

considered, simulation-based ATPG then is outperformed by SAT-based ATPG. This is because an exponential number of simulations must be done in simulation-based ATPG. Furthermore, unlike the simulation-based method, the SAT-base approach makes full use of modern solving techniques. For a reversible circuit with constant inputs, a valid test pattern can be generated much more efficiently by SAT-based ATPG. Nevertheless, neither method ensures a compact testset. To address the need for compact testsets, approaches based on greedy and branch-and-bound methods [HPB04] as well as ILP formulations [PFBH05] have been introduced. However, these approaches apply only to circuits with small numbers of gates. In this context, approaches that guarantee a compact testset for large circuits have to be considered. For this purpose, new approaches are introduced in the next chapter.

Chapter 4

Advanced Automatic Test Pattern Generation

This chapter introduces compact testset generation, i.e., the generation of compact or minimal testsets. The following schemes and approaches are proposed: a fault-ordering scheme, a *pseudo-Boolean optimization* (PBO)-based method for generating compact testsets, and a *Boolean satisfiability* (SAT)-based algorithm for determining minimal testsets.

A fault-ordering scheme for ATPG of reversible circuits is proposed in Section 4.1. Studies of conventional ATPG (see e.g., [LS92; KPKR95; LCD+02; MGHD09]) have shown that the size of the resulting testset is significantly affected by the order in which faults are targeted by ATPG engines. However, here, instead of simply adopting conventional fault

ordering technique, the reversibility of the circuit is exploited.

A PBO-based ATPG is proposed in Section 4.2. The main idea is as follows: Instead of determining a test pattern that detects one particular fault, a test pattern that detects as many faults as possible in the remaining fault list is determined using PBO solvers, so that compact testsets can be generated.

An approach that determines a minimal testset for a given reversible circuit is proposed in Section 4.3. This is realized by iteratively checking whether a testset detecting all faults with only v patterns exists for a given circuit and a given fault list. By starting these checks with $v = 1$ and iteratively incrementing v by one, a minimal testset is ensured. The respective checks are then performed with solvers for Boolean satisfiability. Experiments demonstrate that by the proposed approach, minimal testsets for reversible circuits can be efficiently generated.

In Section 4.4, the efficiencies of the three approaches are demonstrated from the results obtained from an evaluation. Together with the experimental results introduced in the last chapter, all ATPG methods (i.e., the simulation-based, the SAT-based, the PBO-based ATPG, the ATPG by applying the fault ordering scheme and the approach of determining minimal testsets) are evaluated with respect to different application scenarios and test goals (compact testsets vs. efficient generation). The conclusions are given in Section 4.5.

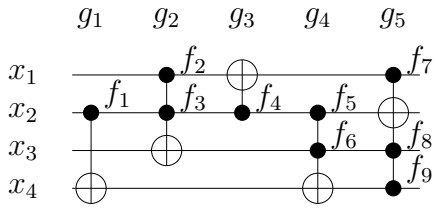


FIGURE 4.1: A reversible circuit with 9 SMCFs

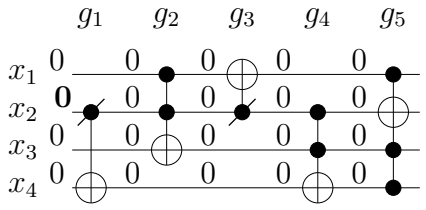


FIGURE 4.2: Effect of fault ordering by targeting f_1 first

Preliminary versions of the approaches and schemes presented in this chapter have been published in [WZD13; WZD11; ZFWD11].

4.1 Fault Ordering

This section describes fault ordering for ATPG of reversible circuits; based on this discussion a fault-ordering scheme is proposed.

Following the ATPG flow shown in Fig. 3.4, the order in which faults are targeted by the ATPG engine has a significant effect on the size of the testsets. This is illustrated by the following example.

Example 4.1. *Consider the circuit shown in Fig. 4.1 together with a fault list* $\mathcal{F} = \{f_1, \ldots, f_9\}$ *composed of single missing control faults. If*

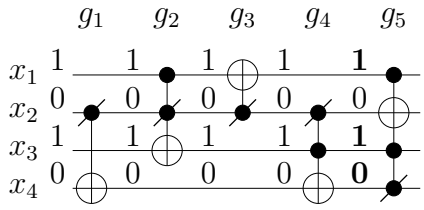


FIGURE 4.3: Effect of fault ordering by targeting f_9 first

the fault f_1 is targeted first, all the control lines of g_1 (except the missing one, which has to be set to 0) have to be set to 1 *in order to generate a test pattern (indicated in bold in Fig. 4.2). This could lead to the pattern* 0000 *as shown in Fig. 4.2. A subsequent fault simulation reveals that this pattern also detects the fault f_4.*

However, if instead of f_1, the fault f_9 is targeted first, significantly more faults could be detected. As an example, this could lead to a test pattern 1010*, as shown in Fig. 4.3, which also detects faults f_1, f_3, f_4, and f_5.*

These observations are not new. Similar behavior has been observed for ATPG of conventional circuits. Motivated by this, several fault ordering strategies have been developed (see e.g., [LS92; KPKR95; LCD+02; MGHD09]). They are generally based on the following premise: Faults whose patterns are probably hard to generate should be considered first. This is motivated by the fact that test patterns for "hard" faults may also detect many "easy" faults. Hence, the total number of test patterns to be generated can be reduced. To classify a fault as "hard" or "easy",

several schemes have been applied to conventional circuits. However, corresponding schemes for faults in reversible circuits have not yet been proposed.

The general idea of a fault-ordering scheme for ATPG of reversible circuits is similar to the schemes applied for conventional circuits, i.e., "harder" faults are targeted first. However, the classification of a fault f as "hard" or "easy" is different and explicitly exploits the reversibility of the underlying circuit. In fact, the classification is based on the number of possible test patterns that detect the fault f. This number can be easily obtained from the number of control lines in the corresponding gate.

Lemma 4.1. *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. 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 $|C|$ fixed assignments to gate g. The values of all remaining lines (including the target line) can be chosen arbitrarily. □

Lemma 4.2. *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 argument as for an 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 the faults for which more test patterns are available in principle. This is illustrated in the following example.

Example 4.2. *Consider again the circuit in Fig. 4.1. The first fault* f_1 *can be detected by all test patterns by setting the inputs of gate* g_1 *to* `-0--`*, where "-" 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*, assign* `01--` *to gate* g_2
- *for* f_3*, assign* `10--` *to gate* g_2
- *for* f_4*, assign* `-0--` *to gate* g_3
- *for* f_5*, assign* `-01-` *to gate* g_4
- *for* f_6*, assign* `-10-` *to gate* g_4
- *for* f_7*, assign* `0-11` *to gate* g_5
- *for* f_8*, assign* `1-01` *to gate* g_5
- *for* f_9*, assign* `1-10` *to gate* g_5

The assignments for faults f_7, f_8, f_9 of g_5 have the fewest arbitrary assignments; hence, these faults should be classified into "hardest" fault class and targeted first. Then, the test patterns generated for these "hard" faults may also detect the "easy" faults. In fact, the three test patterns 0100*,* 1010 *and* 0011*, which detect these "hardest" faults, also detect all other faults in the considered circuit.*

However, note that Lemma 4.1 and Lemma 4.2 constitute upper bounds. In fact, owing to additional constraints (like constant inputs, as discussed in Section 3.1), the actual number of possible test patterns could be less than $2^{n-|C|}$. In the worst case, even no test pattern might be available (if the fault is untestable). Nevertheless, because both bounds provide an easy way to determine approximations to the "hardness" of a fault, they represent a plausible objective for sorting the fault list.

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

Example 4.3. *Considering again the circuit in Fig. 4.1, the proposed fault-ordering scheme could 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.*

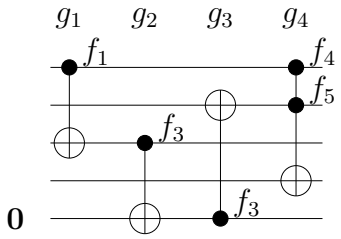


FIGURE 4.4: A reversible circuit with constant inputs

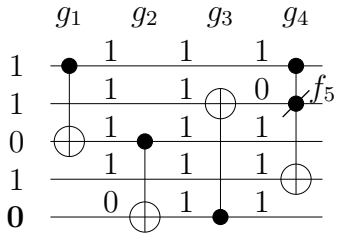


FIGURE 4.5: Targeting f_5 in reversible circuit with constant inputs

As shown by an experimental evaluation, whose results are reported in Section 4.4, this simple scheme leads to a significant compaction of the complete testset especially for a given circuit without constant inputs. For circuits with constant inputs, the situation is more complex and for most cases, the definition of the "hardness" of a fault only based on the number of the control lines is not enough. The following example explains that in detail.

Example 4.4. *Considering the circuit in Fig. 4.4 that includes a constant input at the fifth primary input-line and five SMCFs. This proposed fault-ordering scheme could lead the following order in which faults are targeted: First f_4 and f_5 are addressed, followed by f_3, f_2, and f_1. Then,*

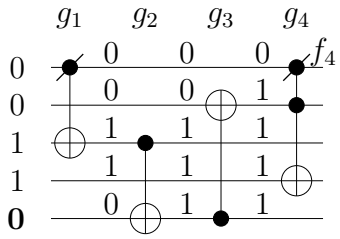


FIGURE 4.6: Targeting f_4 in reversible circuit with constant inputs

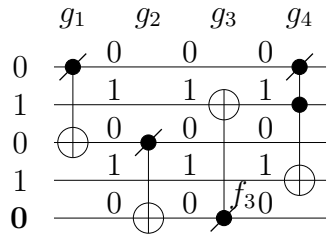


FIGURE 4.7: Targeting f_3 in reversible circuit with constant inputs

a testset may be determined that includes three test patterns 11010, 00110, *and* 01010.

- *By targeting* f_5, *a test pattern* 11010 *could be determined (Fig. 4.5). For* f_5, *the assignments to gate* g_4 *is* `01---`.

- *By targeting* f_4, *a test pattern* 00110 *could be determined, that detects also* f_1 *(Fig. 4.6). For* f_4, *the assignments to gate* g_4 *is* `10---`.

- *By targeting* f_3, *a test pattern* 01010 *could be determined, that detects also* f_1, f_2, *and* f_4 *(Fig. 4.7).*

  If we do not consider the constant input 0 *at the fifth line, then the assignments to* g_3 *for* f_3 *is* `----0`. *However, because there is a*

constant input 0 *at the fifth input line, the first and third lines at* g_3 *must be also assigned to* 0 *to ensure that the fifth line at* g_3 *is* 0. *That is, for* f_3, *the assignments to* g_3 *should be* `0-0-0`. *Then, compared to the other faults,* f_3 *has the fewest arbitrary assignments and should be targeted first.*

In fact, when f_3 *is targeted first, followed by* f_5, f_4, f_1, *and* f_2 *a testset including only two test patterns* 01010 *and* 11010 *could be determined.*

4.2 PBO-based ATPG

This section provides a description of the PBO-based ATPG for determining compact testsets. Using PBO solvers, a test pattern will be determined to detect as many faults as possible in the remaining fault list. Therefore, the ATPG problem is reformulated as a PBO instance by asking

> "Is there a valid assignment to all primary inputs of the given circuit so as to trigger as many fault behaviors of the considered faults as possible?"

To encode this, a pseudo-Boolean function Ψ over the following variables is created:

- $\vec{x}^1 = x_n^1, x_{n-1}^1 \ldots x_1^1$ represents the assignment to the respective primary inputs of the circuit,

- $\vec{x}^{d+1} = x_n^{d+1}, x_{n-1}^{d+1} \dots x_1^{d+1}$ represents 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 $2 \leq k \leq d$ represents the input (output) assignment to the respective gate g_k (g_{k-1}).

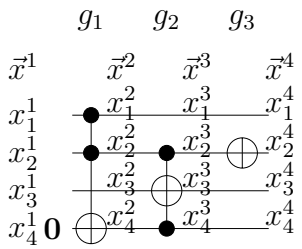
Having these variables, the functionality of the given circuit and the constant inputs are encoded as described for SAT-based ATPG in Section 3.3.

Example 4.5. *The reversible circuit shown in Fig. 4.8(a) is the same circuit as that in Fig. 3.5(a). The variables, the constraints for the functionality of the circuit, and the constants introduced in Section 3.3 for SAT-based ATPG all apply.*

To encode the faults, new variables and constraints are introduced. For the SMCF model, a new variable f_j and the constraint

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

are added for each undetected fault (e.g., an SMCF at the i^{th} line of gate g_k). The variable f_j is set to 1 if an input assignment is applied that detects a missing control fault at line i on gate g_k. Otherwise, this variable is set to 0. Variables and constraints can be accordingly added for other fault models as well.



(a) Variables

(1) Functional constraints:

$x_1^2 = x_1^1, \quad x_2^2 = x_2^1, \quad x_3^2 = x_3^1, \quad x_4^2 = x_4^1 \oplus (x_1^1 \wedge x_2^1)$

$x_1^3 = x_1^2, \quad x_2^3 = x_2^3, \quad x_3^3 = x_3^2 \oplus (x_2^2 \wedge x_4^2), \quad x_4^3 = x_4^2$

$x_1^4 = x_1^3, \quad x_2^4 = \overline{x_2^3}, \quad x_3^4 = x_3^3, \quad x_4^4 = x_4^3$

(2) Constant inputs constraints:

$x_4^1 = 0$

(3) Variable and constraint for each fault:

SMCF at 1^{th} line of g_1

$f_1 = (x_1^1 = 0) \wedge (x_2^1 = 1)$

SMCF at 2^{th} line of g_1

$f_2 = (x_1^1 = 1) \wedge (x_2^1 = 0)$

SMCF at 2^{th} line of g_2

$f_3 = (x_2^2 = 0) \wedge (x_4^2 = 1)$

SMCF at 4^{th} line of g_2

$f_4 = (x_2^2 = 1) \wedge (x_4^2 = 0)$

(4) Objective function: $\mathcal{O} = \overline{f_1} + \overline{f_2} + \overline{f_3} + \overline{f_4}$

(b) PBO constraints

FIGURE 4.8: PBO formulation for all SMC faults

Example 4.5. *(continued)*

Continuing the example in Fig. 4.8, variables and constraints for each fault are added in Fig. 4.8(b).

With this instance an objective function is formulated. As mentioned before, a test pattern should be generated that detects as many faults as possible. Each f_j 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 that maximizes the number of variables f_j set to 1. By applying common PBO solvers, this goal is formulated in the objective function as follows:

$$\mathcal{O} = \sum_{j=1}^{l} \overline{f_j}$$

Note that according to the definition of the PBO problem (Section 2.2), PBO solvers try to find a satisfying solution that *minimizes* $\mathcal{O}$. Owing to the inversion of the respective f_j variables, the solver in fact maximizes $\mathcal{O} = \sum_{j=1}^{l} f_j$, i.e., the number of detected faults is maximized.

Example 4.5. *(continued)*

Continuing the example in Fig. 4.8, the objective function for this example circuit is applied. A test pattern is obtained that detects as many faults, f_j, as possible.

Similar to the SAT-based approach, all these constraints are finally converted into the proper format and passed to the respective solver engine. Because 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 straightforward: Each clause $x_{i1} \vee x_{i2} \vee \cdots \vee x_{ij}$ of the CNF is simply replaced with an equivalent constraint

$$x_{i1} + x_{i2} + \cdots + x_{ij} \geq 1.$$

Then, the PBO solver determines a result from which a test pattern can be derived. Additionally, the concrete faults that are detected by this pattern can be obtained from the assignment to the corresponding f_j-variables. In the next iterations, all f_j-variables representing detected faults (as well as their constraints) are removed from both the formula and the objective function. The proposed formulation is applied only to the remaining faults. This results in a new ATPG flow based on the flow presented in Fig. 3.4. In fact, Step (b) and (e) of the flow in Fig. 3.4 can be omitted. The whole ATPG flow terminates either when test patterns for all faults have been generated or when the instance becomes unsatisfiable. In the latter case, all faults still included in the fault list are classified as untestable.

Compared with the simulation- and SAT-based ATPG introduced in the previous chapter, the PBO-based approach provides a better alternative for generating compact testsets. As confirmed by the experiments discussed in Section 4.4, compact testsets are generated for circuits with and without constant inputs. However, runtime increase because in addition to the determining a satisfying assignment an objective function is also minimized.

4.3 Determining Minimal Testsets

In this section, an approach is proposed for determining a minimal complete testset for a given circuit and a fault list by using techniques for Boolean satisfiability. After the introduction of the general concept, details on the structure of the proposed SAT instance and on the actual encoding are provided.

4.3.1 General Concept

To determine a minimal testset, an iterative approach is proposed. The basic concept is as follows: Given a circuit G and a fault list $\mathcal{F}$, first it is checked whether there exists a complete testset that consists of only the $v = 1$ test pattern. If such a test pattern can not be determined, v is increased by one. This procedure is repeated until a testset is found

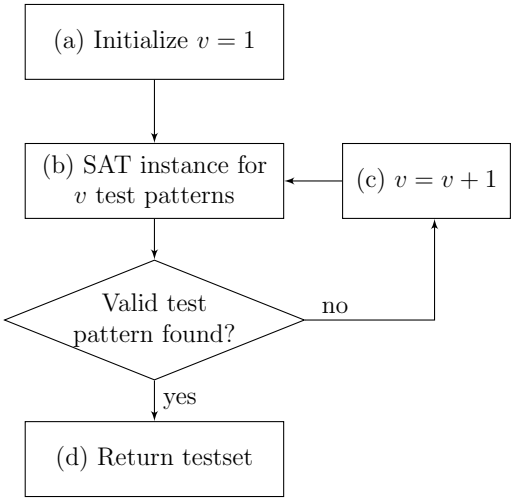


FIGURE 4.9: Test generation flow for determining a minimal testset

that detects all faults in $\mathcal{F}$. By iteratively increasing v by one, a minimal testset is ensured.

The flow of this procedure is shown in Fig. 4.9. At the beginning v is initialized to one, i.e., $v = 1$ in Step (a). The following question is encoded as a SAT instance and passed to a solver (Step (b)):

> "Does there exist a testset consisting of v patterns detecting all faults $f \in \mathcal{F}$ in the circuit G?"

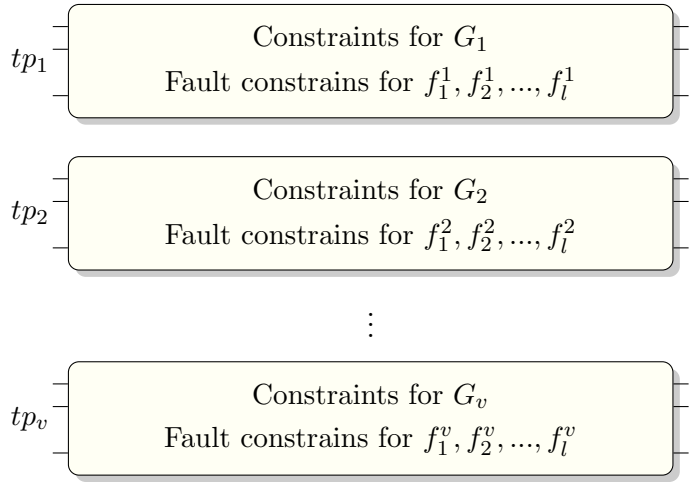
If the solver returns UNSAT, which means no such testset exists, then v is increased by one (Step (c)); otherwise the complete minimal testset can be obtained from the satisfying solution (Step (d)). This process is repeated until the valid testset is found.

Note that there is a limited for this approach. If there are untestable

faults in the fault list, the approach does not terminate, because the SAT solver always returns UNSAT for an untestable fault and then the value v is incremented. To avoid this infinite loop, the maximum size of the testset should be defined. The maximum size of the testset is the same as the number of the faults in the fault list because the size of the testset can be no larger than this number. If the minimal testset is not determined before the number of test patterns becomes the same as the maximum size of the testset, the process should be ended, and it is ensured that there are untestable faults in the fault list. To find the untestable faults in a fault list, the SAT-based or the PBO-based ATPG can be performed.

4.3.2 Structure of the Instance

As mentioned in Fig. 4.9, a SAT instance encoding the question described in Section 4.3.1 needs to be created. The considered circuit G is then copied v-times so that v different input patterns can be assigned to the circuit's primary inputs (allowing for a testset of size v). Furthermore, for each fault $f_j \in \mathcal{F}$ ($\mathcal{F} = \{f_1, ..., f_l\}$) and for each copy G^r of the circuit with $1 \leq r < v$, a variable f_j^r is created. If a test pattern is applied to G^r and if it sets f_j^r to 1 (0), then the fault f_j is (is not) detected by this pattern. In addition, constraints are added to ensure that at least one variable f_j^r with $1 \leq r < v$ is assigned 1. By this means, it is ensured



To ensure each fault is detected by at least one testpattern:

$$f_1^1 \vee f_1^2 \vee ... \vee f_1^v = 1,$$
$$f_2^1 \vee f_2^2 \vee ... \vee f_2^v = 1,$$
$$..., \qquad ...,$$
$$f_l^1 \vee f_l^2 \vee ... \vee f_l^v = 1$$

FIGURE 4.10: SAT encoding for a testset of size v

that the considered fault f_j is detected in at least one copy of the circuit, i.e., by at least one test pattern.

More formally, the instance

$$\text{ATPG}(G, \mathcal{F}, v) = \bigwedge_{r=1}^{v} G^r \wedge \bigwedge_{j=1}^{l} (\bigvee_{r=1}^{v} f_j^r) \tag{4.1}$$

is encoded. The general structure of this instance is shown in Fig. 4.10.

After passing this instance to a SAT solver, the solver tries to deter-

mine an input assignment for each copy of the circuit so that each fault is detected in at least one copy of the circuit. In other words, the SAT solver takes over the task of determining the test patterns. If this is possible which means the instance is satisfiable, the respective test patterns can be easily obtained from the satisfying assignment of the respective variables. In contrast, if the SAT solver returns unsatisfiable, it is proven that no complete testset with v patterns exists.

To encode the instance presented in Fig. 4.10, a formulation based on the SAT-based ATPG introduced in Section 3.3 is applied. The encoding of the respective circuit copies and fault constraints are identical to the description for the SAT-based ATPG (Section 3.3).

4.4 Experimental Results

This section presents experimental results obtained from the approaches proposed above. The proposed fault-ordering scheme was applied to the SAT-based ATPG introduced in Section 3.3. All proposed approaches were implemented in C++ on top of RevKit [SFWD10]. MiniSat [ES04] was used as solver engine for the SAT-based approaches. As the satisfiability solver, *clasp* [GKNS07] was used for the PBO-based approach and *Boolector* [BB09] was used for the approach of determining minimal testsets. As benchmarks, reversible circuits from RevLib [SFWD10] were

TABLE 4.1: Experimental results for all proposed ATPG approaches (Size of the reulting testset)

CIRCUIT	d	n	c	Δc	$\lvert\mathcal{F}\rvert$	(1) SIMU.	(2) SAT	(3) F.O.	(4) PBO	(5) MINI.TS
WITHOUT CONSTANT PRIMARY INPUTS										
4_49_16	16	4	0	3	24	7	8	6	5	4
ham7_104	23	7	0	2	34	6	6	5	6	4
ham15_108	70	15	0	3	125	12	10	9	9	8
ham15_109	109	15	0	1	126	8	10	8	9	6
ham15_107	132	15	0	7	352	51	45	25	16	12
hwb7_61	236	7	0	2	693	61	41	32	27	>10
hwb7_62	331	7	0	5	582	66	47	34	28	>12
hwb8_113	637	8	0	6	2214	122	93	59	44	>8
plus127*	910	13	0	12	5704	372	786	272	104	>12
hwb9_119	1544	9	0	6	5812	121	136	102	83	>8
hwb9_123	1959	9	0	7	3596	137	148	93	80	>8
urf3_155	26468	10	0	0	52936	27	25	25	>6	>4
WITH CONSTANT PRIMARY INPUTS										
mini-alu_84	20	10	6	2	27	6	6	6	4	3
rd84_142	28	15	7	1	49	14	18	15	8	7
4_49_7	42	15	11	2	61	7	7	9	5	4
hwb5_13	88	28	23	2	131	>1	11	11	7	4
hwb6_14	159	46	40	2	241	>1	12	12	7	6
ex5p	647	206	198	2	904	>1	24	19	18	>12
spla	1709	489	473	1	2711	>1	35	38	19	>13
table3	1988	554	540	2	2997	>1	41	49	23	>12
pdc	2080	619	603	2	3135	>1	49	50	>2	>12
alu4	2186	541	527	2	3390	>1	41	39	18	12
ex1010	2982	670	660	2	4543	>1	31	29	25	>12

Circuit: name of the circuit; d: number of gates; n: number of lines; c: number of constant inputs; Δc: maximal difference in the number of control lines at the gates in the respective circuit; $\lvert\mathcal{F}\rvert$: total number of faults to be tested;
plus127*: plus127mod8192_162

Table 4.2: Experimental results for all proposed ATPG approaches (Runtime)

Circuit	d	n	c	Δc	$\lvert\mathcal{F}\rvert$	(1) Simu.	(2) SAT	(3) F.O.	(4) PBO	(5) mini.TS
WITHOUT CONSTANT PRIMARY INPUTS										
4_49_16	16	4	0	3	24	<0.01	<0.01	<0.01	<0.01	0.77
ham7_104	23	7	0	2	34	<0.01	<0.01	<0.01	0.01	0.38
ham15_108	70	15	0	3	125	<0.01	0.05	0.05	0.14	1.17
ham15_109	109	15	0	1	126	<0.01	0.08	0.06	0.20	0.35
ham15_107	132	15	0	7	352	0.06	0.53	0.30	0.65	760.63
hwb7_61	236	7	0	2	693	0.18	0.57	0.46	2.41	*T.O.*
hwb7_62	331	7	0	5	582	0.22	0.84	0.65	4.22	*T.O.*
hwb8_113	637	8	0	6	2214	0.84	4.47	3.47	27.50	*T.O.*
plus127*	910	13	0	12	5704	7.88	75.05	24.33	1463.02	*T.O.*
hwb9_119	1544	9	0	6	5812	2.45	18.26	15.09	291.72	*T.O.*
hwb9_123	1959	9	0	7	3596	2.44	18.55	12.82	521.53	*T.O.*
urf3_155	26468	10	0	0	52936	8.09	61.24	64.07	*T.O.*	*T.O.*
WITH CONSTANT PRIMARY INPUTS										
mini-alu_84	20	10	6	2	27	0.30	<0.01	<0.01	<0.01	0.74
rd84_142	28	15	7	1	49	138.94	0.04	0.03	0.04	0.95
4_49_7	42	15	11	2	61	120.43	0.02	0.02	0.03	0.83
hwb5_13	88	28	23	2	131	*T.O.*	0.13	0.12	0.15	0.46
hwb6_14	159	46	40	2	241	*T.O.*	0.39	0.40	0.48	0.71
ex5p	647	206	198	2	904	*T.O.*	13.82	10.97	43.91	*T.O.*
spla	1709	489	473	1	2711	*T.O.*	126.22	135.03	940.60	*T.O.*
table3	1988	554	540	2	2997	*T.O.*	215.16	251.18	2558.53	*T.O.*
pdc	2080	619	603	2	3135	*T.O.*	308.83	315.81	*T.O.*	*T.O.*
alu4	2186	541	527	2	3390	*T.O.*	210.13	192.59	1131.01	1978.12
ex1010	2982	670	660	2	4543	*T.O.*	274.56	249.65	992.45	*T.O.*

Circuit: name of the circuit; d: number of gates; n: number of lines; c: number of constant inputs; Δc: maximal difference in the number of control lines at the gates in the respective circuit; $\lvert\mathcal{F}\rvert$: total number of faults to be tested;
plus127*: plus127mod8192_162

TABLE 4.3: Improvement with respect to quality
(for results in Table 4.1)

CIRCUIT	(3) VS. (2)	(4) VS. (2)	(4) VS. (3)	(5) VS. (4)
WITHOUT CONSTANT PRIMARY INPUTS				
4_49_16	25.00	37.50	16.67	20.00
ham7_104	16.67	0.00	-20.00	33.33
ham15_108	10.00	10.00	0.00	11.11
ham15_109	20.00	10.00	-12.50	33.33
ham15_107	44.44	64.44	36.00	25.00
hwb7_61	21.95	34.15	15.63	–
hwb7_62	27.66	40.43	17.65	–
hwb8_113	36.56	52.69	25.42	–
plus127*	65.39	86.77	61.76	–
hwb9_119	25.00	38.97	18.63	–
hwb9_123	37.16	45.95	13.98	–
urf3_155	00.00	–	–	–
WITH CONSTANT PRIMARY INPUTS				
mini-alu_84	0.00	33.33	33.33	25.00
rd84_142	16.67	55.56	46.67	12.50
4_49_7	-28.57	28.57	44.44	20.00
hwb5_13	0.00	36.36	36.36	42.86
hwb6_14	0.00	41.67	41.67	14.29
ex5p	20.83	25.00	5.26	–
spla	-8.57	45.71	50.00	–
table3	-19.51	43.90	53.06	–
pdc	-2.04	–	–	–
alu4	4.88	56.10	53.85	–
ex1010	6.45	19.35	13.79	–

plus127*: plus127mod8192_162

(2) SAT-based ATPG, (3) Fault ordering applied (SAT-based) ATPG,
(4) PBO-based ATPG, (5) Minimal testset

used. All experiments were carried out on an Intel(R) Xeon(R) CPU ×4 with 32 GB main memory. The timeout (denoted by *T.O.*) was set to 3600 CPU seconds.

Two different goals are considered in ATPG: runtime and quality (i.e., determining a testset that is as compact as possible). With respect to these two goals, all considered ATPG methods were compared with to one another. Evaluations were based on the SMCF model. However, all proposed approaches can easily be extended to other fault models, such as the SMGF model, by adjusting of certain assignment conditions. The results of the evaluation are similar to those shown in Section 3.4.

Table 4.1 and 4.2 summarize the experimental results for detecting the SMCF using all proposed ATPG approaches: (1) simulation-based ATPG, (2) SAT-based approach, (3) fault-ordering applied (SAT-based) ATPG, (4) PBO-based ATPG, and (5) the approach of determining minimal testsets.

The first four columns characterize the circuits: the name of the circuit (denoted by CIRCUIT), number of gates (denoted by d), number of lines (denoted by n), and number of constant inputs (denoted by c). Column ΔC contains the maximum difference in the number of control lines at the gates in the respective circuits. The total number of faults to be tested is denoted by $|\mathcal{F}|$. The sizes of the resulting testset and the runtimes in CPU seconds needed to obtain these results from the five

ATPG are presented in the last five columns of Table 4.1 and 4.2.

Based on the experimental results in Table 4.1, the quality of the ATPG approaches are compared with each other. The improvements are shown in Table 4.3.

4.4.1 Evaluation of Fault-ordering Scheme

The results show that for the SAT-based approach, the proposed fault-ordering scheme hardly affects the runtime of the ATPG flow. Five benchmarks need more runtime when using the SAT-based ATPG with the proposed fault-ordering scheme than when using the ATPG without the scheme. However, the other 13 benchmarks need less runtime when using ATPG with fault-ordering scheme. In all cases, the difference is rather small. In comparison with the PBO-based approach and the approach of determining minimal testsets, the SAT-based approaches lead to better results in terms of runtime.

With respect to quality, significant improvements are found for the SAT-based ATPG when the proposed fault-ordering scheme is applied. In most cases, the proposed scheme leads to much more compact testsets than the previously applied scheme. In the best case, an improvement ((3) vs. (2)) in Table 4.3 of up to 65% for SMCF can be achieved. For some circuits, more test patterns are generated because of the additional constraints (the constant inputs), e.g., *4_49_7* and *table3*. However, on

average, an improvement of 13.89% is documented.

The results also clearly confirm the discussion in Section 4.1: the best improvements can be achieved for circuits without constant inputs, whose gates have large differences in the number of control lines. For example, the circuit *plus127mod8192_162* is composed of gates with one control line (including an "easy" fault) and gates with 13 control lines (including "hard" faults). This means that 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. For circuits with constant inputs, the situation is more complex. Because the effect of the constant inputs, some improvements ((3) vs. (2)) in Table 4.3 are even negative (4_49_7, *spla*, *table*3, and *pdc*).

4.4.2 Evaluation of PBO-based ATPG

In comparison to the SAT-based ATPG with and without the fault-ordering scheme, the PBO-based approach leads to the best results regarding the size of the resulting testsets (quality). For most of the benchmarks, testsets with the smallest number of patterns are obtained (except *ham7_104* and *ham15_109*). Significant improvements can be obtained from the PBO-based approach for some large benchmarks, e.g., for *plus127mod8192_162* improvements ((4) vs.(2) and (4) vs.(3)) of up to 86.77% and 61.76% can be achieved. The average improvements are

38.40% and 26.27%, respectively.

As explained in Section 4.2, when using the PBO-based ATPG compared to the SAT-based approaches, runtime increases because an objective function has to be minimized in addition to determining a satisfying assignment.

However, the PBO-based ATPG is faster than the approach of determining minimal testsets. For most benchmarks, testsets can be achieved by the PBO-based ATPG, but the approach of determining minimal testsets either needs more time or may not even terminate in the defined maximum time. In addition, the PBO-based approach leads to testsets that are very compact (in most cases only one more test patterns than the minimum) for circuits with and without constant inputs.

4.4.3 Evaluation of the Approach for Determining Minimal Testsets

As can be seen from the results, minimal testsets can be efficiently obtained for circuits containing approximately 100 gates even with constant inputs. Less than one second is needed for this purpose. For example, the minimal testsets for *hwb5_13* and *hwb6_14* can be achieved with *0.46* and *0.71* s, respectively, while the simulation-based ATPG does not terminate in 3600 CPU seconds.

So far, generating minimal testsets has been presented only for small

Table 4.4: Comparison of ATPG methods

Method	without Constant Inputs		with Constant Inputs	
	Efficiency	Quality	Efficiency	Quality
Simulation-based	+++	O	- - -	O
SAT-based	+	-	+	-
Fault Ordering	+	+	+	-
PBO-based	-	++	-	++
Minimal Testset	- - -	+++	- -	+++

circuits (e.g., for circuits with a maximum of 291 gates, as in [PFBH05]). With this approach larger circuits can be handled. For the first time, it is possible to generate a minimal testset for a circuit composed of more then 2,000 gates, i.e., *alu4* with 2184 gates and 3390 SMCFs.

Other than the SAT-based approaches and the PBO-based ATPG, determining minimal testsets needs more runtime because more complex instances have to be solved. Moreover, the other ATPG methods do not guarantee a minimum.

4.5 Summary

Table 4.4 summarizes the conclusions on the quality and efficiency of all the proposed ATPG methods. The table compares results for circuits with and without constant inputs. In the table, "+", "-", and "O" indicate positive, negative, and optional evaluation, respectively.

If the quality is the crucial factor, the approach for determining minimal testsets leads to the best results. The second best results are pro-

vided by the PBO-based ATPG. While the size of the test patterns obtained from the SAT-based approaches with and without applying the fault-ordering scheme still is larger than the minimum, the PBO-based approach leads to testsets that are very compact for circuits with and without constant inputs. Most cases require only one more test pattern than the minimum.

For circuits without constant inputs, applying the fault-ordering scheme significantly improves the quality of the testsets for the SAT-based ATPG. However, for circuits with constant inputs, the improvement provided by fault ordering is not obvious. In this context, new schemes of fault ordering have to be considered. For example, the "hardness" of the faults whose assignments are effected by the constant inputs should be re-defined in the further.

In the simulation-based ATPG, a different test pattern can be generated every time for a fault because of the random setting of the input assignment. Therefore, the size of the testset is not fixed for a circuit with or without constant inputs. Hence, it is difficult to compare the quality of the simulation-based ATPG with the other ATPG methods.

If runtime is the main criteria, a more divergent picture emerges. When only circuits without constant inputs are considered, the simulation-based approach clearly outperforms the other approaches. In this approach, all benchmarks can be handled in just a few seconds, while the

other approaches need some minutes (the SAT-based approaches) or even timeout (the PBO-based ATPG and the approach of determining minimal testsets). However, as discussed in Section 3.1, simulation is not efficient for circuits with constant inputs. In such situations, SAT-based approaches are the better alternative.

Chapter 5

Testing Multiple Faults

This chapter introduces approaches for testing multiple faults in reversible circuits. Compared to testing single faults, testing multiple faults is more difficult; e.g., in the case of the *Multiple Missing Gate Fault* (MMGF) model, the maximum number of possible multiple faults in a reversible circuit with d gates is $2^d - 1$. Nevertheless, testing multiple faults is important because a circuit with multiple faults can malfunction even when it passes the complete test for single faults (Section 5.1).

So far, little research has been done on testing multiple faults for reversible circuits. In [PHM04], the multiple fault model used for reversible circuits was the stuck-at model, which is known to be unsuitable for reversible circuits [HPB04]. In [HPB04; PFBH05] methods for testing the MMGF model where introduced for the first time. However, the

MMGF model is restricted to multiple consecutive missing gates, and these methods apply only to small circuits.

In this chapter, methods are proposed for testing MMGF without restriction. Based on *Boolean satisfiability* (SAT) and *pseudo-Boolean optimization* (PBO), the complete testsets of MMGF can be efficiently generated and applied to large circuits. The general idea is as follows: test patterns targeting single faults are generated and simultaneously the ATPG process is modified so that patterns are ensured to detect possibly many multiple faults. In this way, many multiple faults can be eliminated before checking for fault masking, i.e., to check whether faults can be detected by the given test patterns. If undetected multiple faults remain, ATPG methods explicitly targeting multiple faults will be performed; these methods either generate further test patterns or prove that the remaining faults are untestable.

In Section 5.1, a PBO-based method is proposed for generating test patterns that target single faults but are also ensured to detect possibly many multiple faults. Then, in Section 5.2, a SAT-based approach is introduced for generating test patterns that explicitly target multiple faults. Using the two previous ATPG methods, a complete testset for testing multiple faults can be generated. The general testing flow for multiple faults is introduced in Section 5.3. To improve the testing flow, an additional check for fault masking using SAT is proposed in Section 5.4.

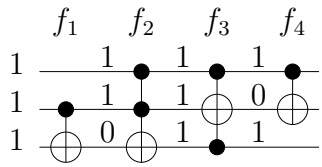


FIGURE 5.1: Test pattern 111 detects all SMGFs

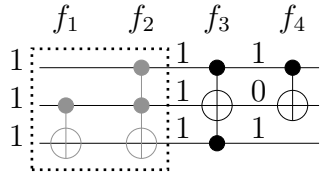


FIGURE 5.2: Undetected MMGF by test pattern 111

Finally, experimental results and conclusions are reported and drawn in Sections 5.5 and 5.6, respectively.

5.1 PBO-based ATPG for Multiple Faults

The following examples show that a complete testset for single faults could possibly not detect all multiple faults because the fault behaviors of the single faults may mask each other.

Example 5.1. *Consider the circuit shown in Fig. 5.1 with four SMGFs* f_1, f_2, f_3, *and* f_4 *at gates* g_1, g_2, g_3, *and* g_4, *respectively. There are then 15 MMGFs, i.e.,* $(f_1), (f_2), ..., (f_1, f_2, f_3, f_4)$.

If the four SMGFs are considered, a testset composed of only one test pattern 111 *can be determined. The pattern* 111 *detects all these four SMGFs because the values before the control lines of all four gates are*

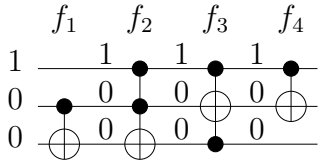


FIGURE 5.3: 100 detects f_4 but not f_3, f_2, f_1

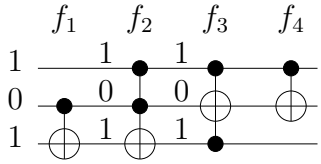


FIGURE 5.4: 101 detects f_3 but not f_2, f_1

set to 1 *(Fig. 5.1). However, the test pattern* 111 *cannot be guaranteed to detect all MMGFs, e.g., the MMGF* (f_1, f_2) *cannot be detected by* 111*. In other words, the faulty behaviors of SMGFs* f_1 *and* f_2 *mask each other by* 111 *(Fig. 5.2).*

In fact, pattern 111 *does not detect MMGF* (f_1, f_2), (f_1, f_2, f_3, f_4), *and* $(f_3, f_{.4})$. *In addition, to check which MMGF is not detected by the test pattern* 111*, fault simulations must be performed for all 15 MMGFs.*

However, there are better test patterns, that can be ensured to detect many multiple faults, even without checking by fault simulation.

Example 5.1. *(continued)*

If instead of test pattern 111*, four test patterns* 100, 101, 010*, and* 110 *are generated for the four single faults, respectively; these four test patterns are ensured to detect all MMGFs. Then, checking for fault masking*

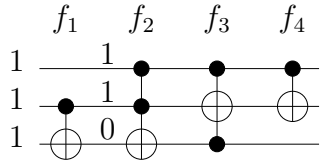


FIGURE 5.5: 111 detects f_2 and f_1

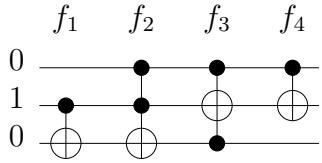


FIGURE 5.6: 010 detects MMGF (f_1, f_2)

can be done using only a very few fault simulations. Each of these test patterns detects a target SMGF and simultaneously detects SMGFs as few as possible before the target fault. More precisely, the test patterns behave as follows:

- 100 *detects* f_4 *but not* f_1, f_2, *and* f_3 *(Fig. 5.3). This means that* 100 *is ensured to detect MMGFs* (f_1, f_4), (f_2, f_4), (f_3, f_4), (f_1, f_2, f_4), (f_1, f_3, f_4), (f_2, f_3, f_4), *and* (f_1, f_2, f_3, f_4) *because in all the multiple faults only the behavior of* f_4 *can be observed.*

- 101 *detects* f_3 *but not* f_1 *and* f_2 *(Fig. 5.4). Pattern* 101 *is ensured to detect MMGFs* (f_1, f_3), (f_2, f_3), *and* (f_1, f_2, f_3).

- 111 *detects* f_1 *and* f_2 *(Fig. 5.5). Then,* 111 *is not ensured to detect MMGF* (f_1, f_2) *and fault masking must be checked. In fact,* 111

does not detect (f_1, f_2)*. Hence, a further test pattern has to be generated for this MMGF.*

- 010 *is generated for testing MMGF* (f_1, f_2) *(Fig. 5.6).*

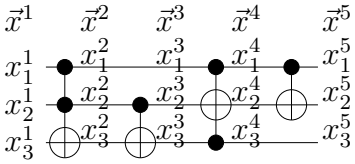
Motivated by these observations, the general idea for testing MMGF is as follows: a test pattern is generated by targeting an SMGF; this test pattern detects the target fault and simultaneously detects other SMGFs as few as possible before the target fault. In this way, the test pattern is ensured to detect possibly many MMGF (even without checking for fault masking).

For this purpose, the PBO-based ATPG described in Section 4.2 can be used. However, this approach determines a test pattern by detecting as many single faults as possible in the fault list. Therefore, the difference between the instances for testing a single fault and testing multiple fault lies only in the objective function $\mathcal{O}$. When testing multiple faults the objective function should be:

$$\mathcal{O} = \sum_{j=k-1}^{l} f_j$$

for all faults before the target fault f_k.

In addition, to ensure that the faulty behavior of the target SMGF is



(a) Variables

1. Constraint for functionality of circuit:
   $x_1^2 = x_1^1, \quad x_2^2 = x_2^1, \quad x_3^2 = (x_1^1 \wedge x_2^1) \oplus x_3^1$
   $x_1^3 = x_1^2, \quad x_2^3 = x_2^2, \quad x_3^3 = x_2^2 \oplus x_3^2$
   $x_1^4 = x_1^3, \quad x_2^4 = (x_1^3 \wedge x_3^3) \oplus x_2^3, \quad x_3^4 = x_3^3$
   $x_1^5 = x_1^4, \quad x_2^5 = x_1^4 \oplus x_2^4, \quad x_3^5 = x_3^4$

2. Constraint for SMGF f_4 at g_4:
   $x_1^4 = 1$

3. Constraint for the SMGF before f_4:
   $f_1 = x_1^1 \wedge x_1^2, \qquad f_2 = x_2^2, \qquad f_3 = x_3^1 \wedge x_3^3$

4. Objective function:
   $\mathcal{O} = f_1 + f_2 + f_3$

(b) PBO Constraints

FIGURE 5.7: PBO formulation for testing multiple faults

triggered, a constraint is added.

$$\bigwedge_{x_c} x_c = 1$$

This means that the assignment of each control line on the considered gate must be set to 1 (Section 2.2).

Example 5.2. *Consider the circuit in Fig. 5.7(a). The variables needed*

to encode the ATPG problem for this circuit as a PBO instance are annotated to the respective gate inputs and outputs.

Using those variables, a PBO instance is created asking for a test pattern that detects SMGF at g_4 and at the same time, as few single faults as possible before the target SMGF. The constraints and the objective function are shown in Fig. 5.7(b).

The resulting instance is passed to a PBO solver. If the solver determines a satisfying assignment, a valid test pattern can be obtained from the assignment to $x_1^1, ..., x_n^1$. At the same time, all single faults before the target fault, which are undetected by this pattern, can be obtained from the assignment to the corresponding f_j-variables. Afterwards, the multiple faults, which are ensured to be detected by this pattern, can be easily determined by using these undetected single faults. In contrast, if the PBO solver returns unsatisfiable, the target single fault is proven to be untestable.

5.2 SAT-based ATPG for the Remaining Multiple Faults

In most cases, the ATPG approach introduced in the previous section generates a testset that targets single faults, and at the same time, detects possibly many multiple faults. However, this approach does not guaran-

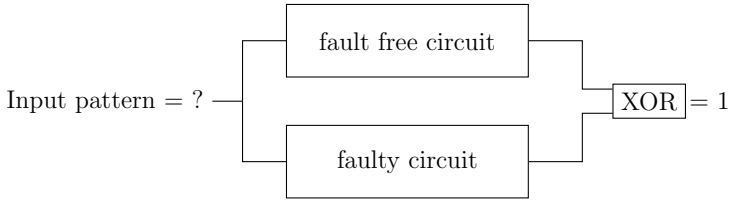
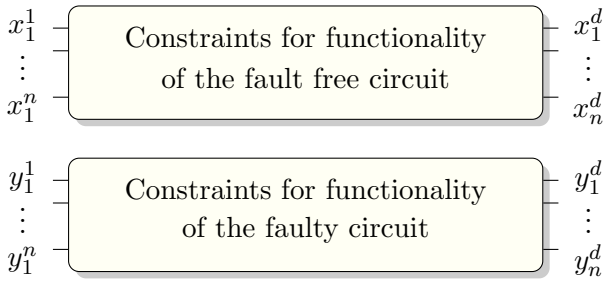


FIGURE 5.8: XOR-Scheme for ATPG of reversible circuit

tee completeness, i.e., some multiple faults might not be detected. Moreover, owing to fault masking, some faults may actually be untestable. For these faults, an alternative ATPG based on SAT is introduced here. As illustrated in Fig. 5.8, a fault-free circuit and a faulty circuit can be joined into a SAT instance. By setting the value of the Boolean difference (XOR) between the two outputs as 1, self-assignments to the inputs of the two circuits are made, i.e., if a satisfiable result returns, then the obtained input pattern detects the fault in the faulty circuit. Otherwise, the fault is proven to be untestable. In fact, the idea of this XOR-scheme is not new. In testing of conventional circuits using SAT engine, the XOR-scheme has been applied in several algorithms (see e.g., [DEFT09; ED12]).

The general structure of the SAT instance for the XOR-scheme is shown in Fig. 5.9. Compared to the SAT-based ATPG introduced in Section 3.3, two circuits (fault-free and faulty) have to be encoded. In addition, instead of constraints for triggering the faulty behavior for a single fault, constraints for assuring the same assignments to the inputs



To ensure same assignments of the inputs:

$$x_1^1 = y_1^1, \quad ..., \quad x_n^1 = y_n^1$$

To ensure different assignments of the outputs:

$$(x_1^d \oplus y_1^d) \vee ... \vee (x_n^d \oplus y_n^d) = 1$$

FIGURE 5.9: SAT instance structure for the XOR-scheme

as well as different assignments to the outputs between the two circuits are needed. Most constraints on the functionality for the faulty circuit are the same as those for the fault-free circuit. The difference lies only in the constraints for the faulty gates, e.g., for the SMGF that occurs at gate g_k, the constraint

$$x_t^{k+1} = x_t^k$$

is added for the target line of g_k.

Example 5.3. *Consider the circuit in Fig. 5.7(a) again. To determine a test pattern for the MMGF (f_1, f_2) the constrains shown in Fig. 5.10 are added in the SAT instance.*

Similar to the SAT-based approach introduced in Section 3.3, a test

(1) Constraints for functionality of fault free circuit are the same as the constraints in Fig. 5.7(b).

(2) Constraints for functionality of faulty circuit:

$y_1^2 = y_1^1, \quad y_2^2 = y_2^1, \quad y_3^2 = y_3^1, \quad y_1^3 = y_1^2, \quad y_2^3 = y_2^2, \quad y_3^2 = y_3^1$

$y_1^4 = y_1^3, \quad y_2^4 = y_2^3 \oplus (y_1^3 \wedge y_3^3), \quad y_3^4 = y_3^3, \quad y_1^5 = y_1^4, \quad y_2^5 = y_2^4 \oplus y_1^4, \quad y_3^5 = y_3^4$

(3) Constraints for ensuring same assignment of inputs:

$x_1^1 = y_1^1, \quad x_2^1 = y_2^1, \quad x_3^1 = y_3^1$

(4) Constraints for ensuring different assignment of outputs:

$(x_1^5 \oplus y_1^5) \vee (x_2^5 \oplus y_2^5) \vee (x_3^5 \oplus y_3^5) = 1$

FIGURE 5.10: SAT formulation for the XOR-scheme

pattern detecting the target multiple fault can be derived from a satisfying result from the SAT solver. Otherwise, the SAT solver returns unsatisfiable and the multiple fault is proven to be untestable.

In fact, the SAT-based ATPG using the XOR-scheme can also be utilized to generate test patterns for single faults. However, the SAT-based ATPG introduced in Section 3.3 is more efficient for single faults because only one circuit is encoded in the SAT instance, i.e., fewer constraints have to be solved by the SAT solver.

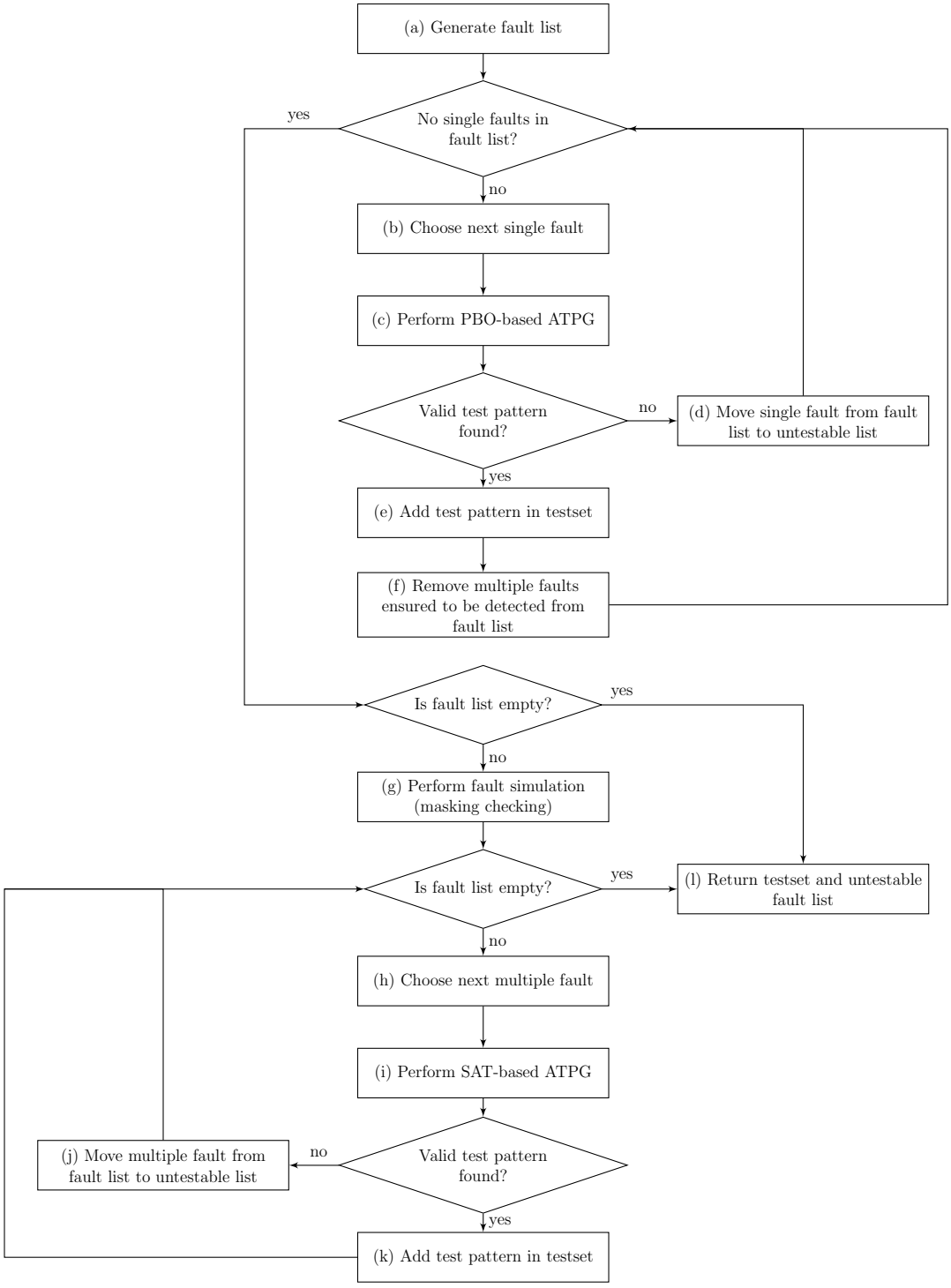


FIGURE 5.11: ATPG flow for multiple faults

5.3 Testing Flow

Using the test patterns generated by the PBO-based and SAT-based approaches introduced in the last two sections, all multiple faults can be detected (or proven untestable). The testing flow is shown in Fig. 5.11.

All multiple faults to be considered are stored in a fault list (Step (a)). Until there are no single faults (i.e., a multiple fault is constituted only by one single fault) in the fault list, a single fault is targeted (Step (b)). Then, the PBO-based approach introduced in Section 5.1 is performed (Step (c)); this determines a test pattern that detects the target fault and as few single faults as possible before the target fault. If a valid test pattern is derived, the pattern is added to the testset (Step (e)), and at the same time, all multiple faults (implicated by the undetected single faults before the target fault) that are detected by the test pattern can be easily obtained and removed from the fault list (Step (f)). Otherwise, if no valid test pattern is determined, the target fault is proven to be untestable, and it is moved from the fault list to the list of untestable faults (Step (d)).

When there are no more single faults in the fault list, but the fault list is still not empty, then there are still multiple faults, and a fault simulation is performed for all the faults remaining in the fault list. Faults detected by the patterns in the testset are removed from the fault

list (Step (g)). After the fault simulation if there are still undetected faults in the fault list, a target multiple fault is chosen (Step (h)) and the SAT-based approach introduced in Section 5.2 is performed (Step (i)). If the SAT-based approach is successful, a further test pattern is added to the testset and the target fault is removed from the fault list (Step (k)). Otherwise, the target multiple fault is proven to be untestable and it is moved from the fault list to the untestable list (Step (j)). This SAT-based approach is performed on each remaining multiple fault in the fault list. The process continues until all faults in the fault list are classified, providing a complete testset and a list of untestable faults (Step (l)).

5.4 Additional Checking for Fault Masking

Applying the testing flow shown in Fig. 5.11, a complete testset can be generated to detect all testable multiple faults. However, if we only use fault simulation (Step (g)) to check for fault masking (to find undetected faults by test patterns), the proposed testing flow only applies to small circuits (circuits with only a few gates). Therefore, in this section an additional method is introduced to check for fault masking. The method uses SAT and can exactly determine the undetected multiple faults for a given test pattern, so that more multiple faults can be removed from the testing flow (Step (f)).

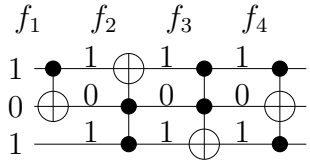


FIGURE 5.12: Fault masking checking for a given test pattern

In Step (c) of Fig. 5.11, a test pattern is generated by the PBO-based approach; this pattern detects the target single fault and as few other single faults as possible before the target fault. Afterward, based on undetected single faults, possibly many multiple faults, which are ensured to be detected by the test pattern, can be identified and removed from the fault list. However, in most cases, the only test patterns that can be generated detect the target single fault as well as some single faults before the target fault. Many multiple faults remain and have to be checked by fault simulation (Step (g)), even if they are in fact detected by the generated test pattern. This is illustrated by the following example.

Example 5.4. *Consider the circuit in Fig. 5.12. A test pattern* 101 *is determined to detect* f_4 *and as few as possible of* f_1, f_2*, and* f_3*. Nevertheless,* 101 *also detects* f_1*.*

The multiple faults ensured to be detected are $(f_2, f_4), (f_2, f_3, f_4)$*, and* (f_3, f_4) *because the faulty behaviors of* f_2 *and* f_3 *are not observable by* 101*.*

Because 101 *also detects* f_1*, the faulty behavior of* f_1 *can be observed.*

However, the following multiple faults

$$(f_1, f_4), \quad (f_1, f_2, f_4), \quad (f_1, f_3, f_4) \quad \textit{and} \quad (f_1, f_2, f_3, f_4)$$

cannot be guaranteed to be detected by 101. *In fact, among these multiple faults, only* (f_1, f_2, f_3, f_4) *is undetected.*

Because the determined test pattern from PBO-based ATPG has a high possibility to detect many multiple faults, even if these multiple faults cannot be ensured to be detected (because single faults before the target fault are also detected), an additional check for fault masking should be performed between Step (e) and (f) of the ATPG flow in Fig. 5.11. Therefore, instead of the common fault simulation (whose complexity is very high), a SAT-based method is proposed for checking for fault masking, i.e., for determining undetected faults for a given test pattern. The check for fault masking is formulated as a SAT instance by asking,

> "Is there a multiple fault among the considered faults, that occurs in the circuit but that lets the assignment to primary output be unchanged (same as the fault-free assignment) for the given assignment to the primary input?"

Similar to the encoding for the SAT-based ATPG introduced in Section 3.3, the variables $\vec{x}^k = (x_n^k x_{n-1}^k ... x_1^k)$ with $(1 < k < d+1)$ are intro-

duced to represent the assignment to the primary inputs (for $k = 1$), the primary outputs (for $k = d + 1$), and the inputs (outputs) of the gates (for $2 \leq k \leq d$).

With these variables, the functionality of the fault-free gates (the considered multiple faults do not occur at these gates) are encoded at first, i.e., the following constraints are added to the SAT instance:

$$x_i^{k+1} = \begin{cases} x_i^k \oplus \bigwedge\limits_{x_c \in C_k} x_c, & \text{if } x_i^k \text{ represents the target line of} \\ & \text{gate } g_k, \\ x_i^k, & \text{else.} \end{cases}$$

This encoding is repeated from the SAT-based ATPG in Section 3.3. For example, if the considered multiple faults may occur between gate $g_{i^{target}}$ (which contain the target SMGF for the PBO-based ATPG) and gate $g_{i^{detected}}$ (which contain the detected SMGF before the target fault), the gates being ensured fault free are all gates g_k with $1 \leq k < i^{detected}$ or $i^{target} < k < d$.

Afterward, the functionality of the gates in which the multiple fault may occur are encoded with two additional variables f_k and r_k (representing the faulty functionality and fault-free functionality of the target line, respectively) together with the following constraints:

- $x_i^{k+1} = x_i^k$, if x_i^k does not represent the target line g_k.

  The values of all lines (except the target line) of g_k are passed through.

- $f_k = (\bigwedge_{x_c \in C_k} x_c) \wedge \overline{(x_t^k \oplus x_t^{k+1})}$, where x_t^k represents the target line of g_k.

  f_k represents the functionality of an SMGF. If f_k is set to 1, the faulty behavior of an SMGF must be triggered ($\bigwedge_{x_c \in C_k} x_c = 1$), but the value of the target line is unchanged ($\overline{(x_t^k \oplus x_t^{k+1})} = 1$).

- $r_k = (\bigwedge_{x_c \in C_k} x_c) \oplus \overline{(x_t^k \oplus x_t^{k+1})}$, where x_t^k represents the target line of g_k.

  r_k presents the fault-free functionality of g_k. The value of the target line is changed only if all control lines are set to 1.

- $f_k \oplus r_k = 1$, if $i^{detected} < k < i^{target}$.

  This constraint ensures that the functionality of the gates at which the fault may occur is either fault free or faulty.

- $f_k = 1$, if $k = i^{detected}$, or $k = i^{target}$.

  The target single fault and the detected single fault are ensured to

occur in the circuit.

Having the variables for faulty functionality of the gates, the constrain

$$f_{i^{detected}} \vee f_{i^{detected}+1} \vee ... \vee f_{i^{target}-1} \vee f_{i^{target}} = 1$$

is added to ensure that a multiple fault must occur in the circuit.

In addition, there may be more than one undetected multiple fault for a given test pattern. Therefore, all determined undetected multiple faults have to be eliminated. For example, if an MMGF occurs at gates $g_{i^{detected}}$, g_{i^1}, g_{i^2}, and $g_{i^{target}}$, and if this fault is already determined to be undetected by the test pattern, then the constraint

$$f_{i^{detected}} \wedge f_{i^1} \wedge f_{i^2} \wedge f_{i^{target}} = 0$$

is added to ensure that this fault is eliminated.

Finally, constraints are added to ensure that the value of the primary output is fault free. If the primary input is x_1^{in}, ..., x_n^{in} and the fault free primary output is x_1^{out}, ..., x_n^{out}, then the following constraints are added:

$$x_1^1 = x_1^{in}, \ ..., \ x_n^1 = x_n^{in}$$

$$x_1^d = x_1^{out}, \ ..., \ x_n^d = x_n^{out}$$

(1) Constraints for g_1:

$x_1^2 = x_1^1$, $x_3^2 = x_3^1$, $f_1 = x_1^1 \wedge \overline{(x_2^1 \oplus x_2^2)}$, $r_1 = x_1^1 \oplus \overline{(x_2^1 \oplus x_2^2)}$, $f_1 \oplus r_1 = 1$

(2) Constraints for g_2:

$x_2^3 = x_2^2$, $x_3^3 = x_3^2$, $f_2 = (x_2^2 \wedge x_3^2) \wedge \overline{(x_1^3 \oplus x_1^4)}$, $r_2 = (x_2^2 \wedge x_3^2) \oplus \overline{(x_1^3 \oplus x_1^4)}$, $f_2 \oplus r_2 = 1$

(3) Constraints for g_3:

$x_1^4 = x_1^3$, $x_2^4 = x_2^3$, $f_3 = (x_1^3 \wedge x_2^3) \wedge \overline{(x_3^3 \oplus x_3^4)}$, $r_3 = (x_1^3 \wedge x_2^3) \oplus \overline{(x_3^3 \oplus x_3^4)}$, $f_3 \oplus r_3 = 1$

(4) Constraints for g_4:

$x_1^5 = x_1^4$, $x_3^5 = x_3^4$, $f_4 = (x_1^4 \wedge x_3^4) \wedge \overline{(x_2^4 \oplus x_2^5)}$, $r_4 = (x_1^4 \wedge x_3^4) \oplus \overline{(x_2^4 \oplus x_2^5)}$, $f_4 \oplus r_4 = 1$

(5) Constraints for ensuring SMGF at g_1 and g_4 must occur:

$f_1 = 1,\ f_4 = 1$

(6) Constraints for ensuring a multiple fault must occur:

$f_1 \vee f_2 \vee f_3 \vee f_4 = 1$

(7) Constraint for ensuring MMGF (f_1, f_2, f_3, f_4) is eliminated:

$f_1 \wedge f_2 \wedge f_3 \wedge f_4 = 0$

(8) Constraints for primary inputs and outputs:

$x_1^1 = 1$, $x_2^1 = 0$, $x_3^1 = 1$, $x_1^5 = 1$, $x_2^5 = 1$, $x_3^5 = 1$

FIGURE 5.13: SAT formulation to check for fault masking

Example 5.5. *Consider the circuit in Fig. 5.12 again. The constraints for determining an undetected MMGF among the faults* (f_1, f_4), (f_1, f_2, f_4), (f_1, f_3, f_4) *by test pattern* 101 *(the primary output is* 111*) are added in Fig. 5.13. The undetected fault* (f_1, f_2, f_3, f_4) *is assumed to have been already determined.*

The resulting instance is passed to a SAT solver. If the solver returns a satisfying assignment, an undetected multiple fault can be derived from the assignments of the corresponding f_k-variables. As long as a satisfiable result returns, a new SAT instance is generated to determine a new undetected multiple fault. Otherwise (if an unsatisfiable result is returned), all undetected faults among the considered multiple faults by the given test pattern are determined.

5.5 Experimental Results

The proposed approaches were implemented in C++ on top of RevKit [SFWD10]. As the underlying solving engines, *clasp* [GKNS07] (for the PBO-based ATPG introduced in Section 5.1) and *MiniSAT* [ES04] (for the SAT-based ATPG introduced in Section 5.2 and the SAT-based additional checking for fault masking of Section 5.4) were applied. As benchmarks, reversible circuits from RevLib [SFWD10] were used. The complete testset for the MMGF model was generated from the testing

TABLE 5.1: Experimental results for testing multiple faults
(Multiple Missing Gate Fault Model)

CIRCUIT	d	n	c	$\|\mathcal{F}\|$	SIMU. CHECKING		SAT. CHECKING		#UT	XOR
					#TS	TIME(S)	#TS	TIME(S)		#TP
4_49_16	16	4	0	16	9	0.15	9	0.04	0	0
mini-alu_84	20	10	6	20	10	2.32	10	0.17	0	1
ham7_104	23	7	0	23	13	2.68	13	0.09	0	0
rd84_142	28	15	7	28	28	0.14	28	0.33	0	0
4_49_7	42	15	11	42	12	451.11	12	1.04	0	0
ham15_108	70	15	0	70	–	*T.O.*	32	2.19	0	0
hwb5_13	88	28	23	88	–	*T.O.*	15	14.80	0	0
ham15_109	109	15	0	109	–	*T.O.*	30	9.55	0	0
ham15_107	132	15	0	132	–	*T.O.*	70	5.09	0	2
hwb6_14	159	46	40	159	–	*T.O.*	26	101.52	0	0
hwb7_61	236	7	0	236	–	*T.O.*	67	71.62	0	0
hwb7_62	331	7	0	331	–	*T.O.*	64	317.91	0	0
alu_8	453	91	64	453	–	*T.O.*	209	3019.98	4	4
hwb8_113	637	8	0	637	–	*T.O.*	121	682.55	0	0
ex5p	647	206	198	647	–	*T.O.*	>37	*T.O.*	0	0
plus127*	910	13	0	910	–	*T.O.*	550	2772.70	0	34

Circuit: name of the circuit; d: number of gates; n: number of circuit lines; c: number of constant inputs; $|\mathcal{F}|$: number of the SMGF, the number of the considered MMGF is $2^{|\mathcal{F}|}-1$; #TS: number of test patterns in the provided test set; TIME: runtime in CPU seconds; #UT: number of untestable faults; #TP: number of test patterns targeting multiple faults; plus127*: plus127mod8192_162

flow, introduced in Section 5.3, including the (SAT-based) additional checking for fault masking, introduced in Section 5.4. For comparison, the complete testset was also determined using fault simulation for the additional checking for fault masking. All experiments were carried out on an Intel(R) Xeon(R) CPU ×4 with 32 GB main memory. The timeout (denoted by *T.O.*) was set to 3600 CPU seconds.

The results are presented in Table 5.1. The first four columns characterize the evaluated circuits: (1) name of the circuit, (2) number d of

gates, (3) number n of circuit lines, and (4) number c of constant inputs. Column $|\mathcal{F}|$ denotes the number of SMGFs in the circuit, and the number of considered MMGFs is $2^{|\mathcal{F}|} - 1$.

The remaining columns give the number of the determined patterns of the complete testset (denoted by #TS), the required runtime to obtain them (in CPU seconds and denoted by TIME), the number of untestable faults (denoted by #UT), and the number of the test patterns (denoted by #TP) that explicitly target multiple faults using the SAT-based ATPG introduced in Section 5.2. The results are distinguished by the methods used for additional checking for fault masking, i.e., by whether fault simulation was used or whether the proposed SAT-based approach was used.

The results for runtime (Column TIME), confirm the efficiency of using the SAT-based method for additional checking for fault masking. The complete testset can be efficiently obtained for circuits composed of more than 900 gates (containing more than $2^{900} - 1$ MMGF) by using the SAT-based additional checking for fault masking. In contrast, when using fault simulation, the complete testset can only be obtained for small circuits (timeout is reached for circuits having more than 42 gates).

Furthermore, in many cases, the testset generated from the PBO-based ATPG introduced in Section 5.1 can detect all multiple faults, i.e., no fault remains (test patterns explicitly targeting multiple faults are

not needed). This means, that the test patterns targeting single faults provide high coverage for multiple faults.

5.6 Summary and Future Work

Test patterns targeting single faults were generated by the PBO-based approach, and these test patterns were found to provide high coverage for multiple faults. For the still undetected multiple faults, a SAT-based approach using the XOR-scheme was proposed to generate further test patterns. As an improvement over checking for fault masking by fault simulation, a SAT-based method was proposed that can efficiently determine undetected multiple faults for a given pattern. Using the proposed testing flow, complete testsets for MMGF were generated. For the first time, complete testsets for circuits with more than 900 gates (containing more than $2^{900} - 1$ MMGFs) were obtained.

In future work, the testing flow will be applied to other fault models, such as the *Multiple Missing Control Fault* model. The PBO-based ATPG (introduced in Section 5.1) and additional checking for fault masking using the SAT-based method (introduced in Section 5.4) will be modified to apply to other fault models.

Chapter 6

Fault Diagnosis

While the task of *testing* is to detect whether a circuit is faulty, the task of *diagnosis* is to determine exactly where the fault occurs in the circuit. In this chapter, fault diagnosis for reversible circuits is discussed and diagnostic approaches are introduced.

Despite the progress made in developing appropriate design and test methodologies, *fault diagnosis* of reversible circuits has hardly been considered. Only preliminary approaches and results are available [RTPP04; PBP05]; these include simulation-based methods and so called *fault tables*. Both approaches require that to detect the reason for faulty behavior, all possible assignments to a circuit must made. However, because the number of simulations and the size of fault tables increases exponentially (because all possible assignments are considered), these methods

can only be applied to very small circuits. More efficient approaches have been introduced for conventional circuits [ABF90; AFPB01; ABKS03; VCAA04], but these have neither been applied to nor optimized for reversible circuits.

In this chapter, conventional fault diagnostic methods are applied to reversible circuits, we find that this leads to a diagnostic flow that can be applied to large circuits. Furthermore, new approaches to fault diagnosis are proposed (Section 6.2), that explicitly exploit the advantages of reversible circuits. Therefore, satisfiability solvers and a new structural analysis are utilized. The experimental results show that by exploiting reversibility, diagnosis can be significantly performed faster than by the pure application of conventional methods. In the best case, runtime improvements of more than an order of magnitude can be achieved. A preliminary version of the work presented in this chapter has been published in [ZWD11].

In Section 6.1, conventional fault diagnosis is reviewed and illustrated by means of reversible circuits. Section 6.2 presents improvements realized by exploiting reversibility during fault diagnosis. Experimental results are documented in Section 6.3, and conclusions are drawn in Section 6.4.

6.1 Applying Conventional Fault Diagnosis

If a given circuit is faulty, test engineers are interested in the reason for the fault. Therefore, a fault model must be assumed first. Based on the fault model, efforts are made to narrow the reason for the faulty behavior to a certain location. From that fault location, the corresponding physical fault can be isolated. To narrow down the fault, *fault diagnosis* is applied beforehand, i.e., all available test patterns are applied to the circuit. Based on responses in the presence of the fault (*fault signatures*), fault locations not responsible for the faulty behavior can be excluded. This process is continued until a unique fault location is determined. Sometimes new patterns distinguishing two faults may be needed or faults need to be classified as equivalent. This is done using methods for *Diagnostic Test Pattern Generation* (DTPG) and *fault-equivalence checking*. In the following, the resulting flow (based on [ABF90]) is reviewed and applied to reversible circuits.

The general concept of applying conventional fault diagnosis is as follows: For each test pattern of a given testset, the corresponding fault signatures for each possible fault are determined and stored in a *fault dictionary*. To determine the signatures, fault simulation is applied for each fault as well as for each test pattern. Then, the same test patterns are applied to the circuit under test. The responses are compared with

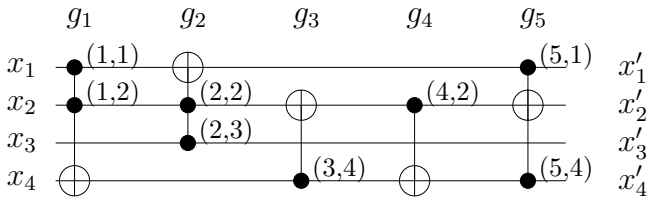


FIGURE 6.1: A example reversible circuit

each entry in the fault dictionary and all faults that lead to different signatures are excluded from further consideration.

Example 6.1. *Fig. 6.1 shows a reversible circuit with eight possible faults under the SMC fault model. To denote the faults, the notation* (j, i) *is used where* j *denotes the faulty gate and* i *denotes the position of the missing control line. The top of Fig. 6.2 shows a corresponding fault dictionary obtained using a testset composed of the two test patterns 1010 and 0100. The dictionary is constructed as a diagnostic tree. Faults detected by the same signatures are grouped together.*

Initially, all possible faults are considered . By applying the test pattern 1010 to the circuit and examining the responses, the reason for the faulty behavior is narrowed down to five subgroups. For example, $(4, 2)$ *can be distinguished from other faults by the output response 1111, because all other faults lead to a different response. In contrast, if the output response 1010 is observed, three faults (namely* $(1, 1)$*,* $(2, 3)$*, and* $(5, 1)$*) still need to be considered. To further refine these results, different test patterns have to be applied.*

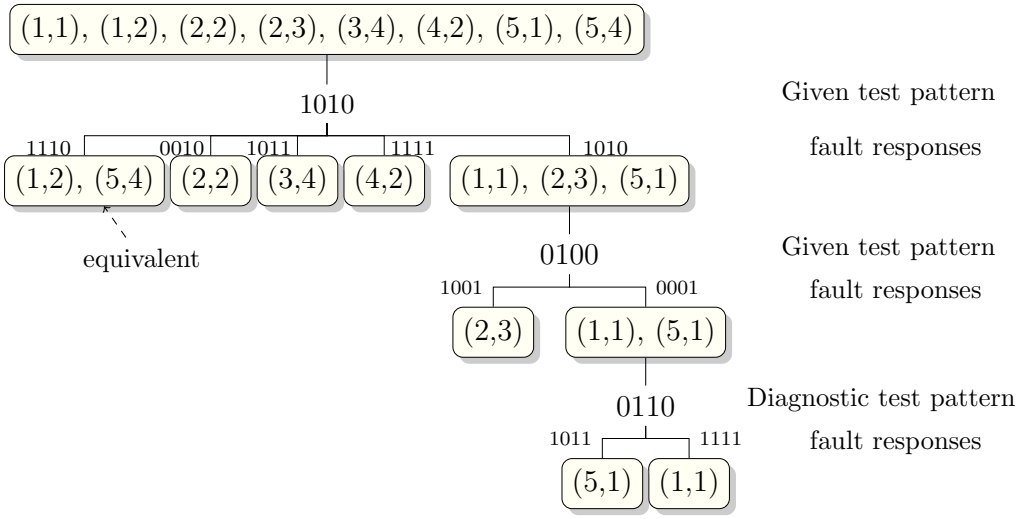


FIGURE 6.2: Diagnostic tree

During fault diagnosis, the given set of test patterns may not be sufficient to distinguish all faults; hence, further test patterns have to be generated. Therefore, dedicated methods for diagnostic test pattern generation are applied that determine not an arbitrary test pattern but a pattern that distinguishes one fault from another. Most of these methods are based on common ATPG approaches in which additional diagnostic capabilities are included. For example, the SAT-based ATPG using the XOR scheme (introduced in Section 5.2) can be used. Instead of considering only one fault for the common ATPG, two faulty circuits are joined in a SAT instance. Therefore, the DTPG problem is reformulated as a SAT instance by asking,

"Is there a valid assignment to the primary inputs of the

two faulty circuits so that the Boolean difference of the two corresponding primary outputs is set to 1 ?".

The encoding is the same as that described in Section 5.2. From a satisfying assignment for the resulting instance, a test pattern can be obtained that distinguishes the two faults; otherwise, it has been proven that no such test pattern exists.

Example 6.2. *Consider again the circuit and the fault dictionary in Figs. 6.1 and 6.2, respectively. The two given test patterns 1010 and 0100 are not sufficient to distinguish fault* $(1, 1)$ *from* $(5, 1)$*. Thus, another test pattern (namely 0110) is obtained by means of DTPG methods. Using this test pattern, the two faults can be distinguished.*

Sometimes not all faults are distinguishable from each other. In fact, two faults are said to be *equivalent*, iff there is no pattern that leads to different output responses for the two. All equivalent faults are grouped together into an *equivalent fault class*. It is crucial to be able to prove that there is no test pattern that distinguishes between two faults. Structural analysis [ABF90] can be applied for this purpose but it is often incomplete. Thus, functional analysis [AFPB01; ABKS03; VCAA04] also has to be applied, although in the worst case, this might lead to exponential runtimes.

Example 6.3. *Consider again the circuit and the fault dictionary from*

Figs. 6.1 and 6.2, respectively. It can be proven that the faults $(1, 2)$ *and* $(5, 4)$ *are equivalent, i.e., there exists no test pattern that distinguishes these two faults. This means that if the input assignment 1010 leads to the output response 1110, no further refinement is possible and both faults must be considered to detect the reason for the faulty behavior.*

6.2 Improved Fault Diagnosis for Reversible Circuits

While all the approaches introduced in the previous section are fully applicable to reversible circuits (as illustrated in the example), they do not exploit the advantages offered by reversible circuits. As a consequence, there is room for improvement when fault diagnosis is considered for reversible circuits. This section introduces new methods for DTPG and for fault equivalence checking that makes use of these possibilities. Use of these methods results in an improved fault diagnostic flow for reversible circuits.

6.2.1 Diagnosis Test Pattern Generation

Two advantageous properties of reversible circuits with respect to test purposes, are good *controllability* and *observability*. These can simplify how test patterns are generated for detecting a certain fault. Good con-

trollability and observability can also be exploited when a test pattern is created that distinguishes faults in a given group. Therefore, assignments triggering certain faults are considered again. More precisely, all possible assignments that trigger at least one of the faults in a certain group are considered (without explicitly enumerating them). Based on the results, a test pattern is determined that detects at least one other fault, but as few as possible, in the considered group. This can be formulated as a PBO instance. Recall the PBO-based ATPG described in Section 4.2, which determines a test pattern that detects as many faults as possible in the fault list. The difference between the PBO instance for ATPG-problems and the PBO instance for DTPG-problems lies in the objective function $\mathcal{O}$, i.e., for the DTPG-problem, the objective function should be

$$\mathcal{O} = \sum_{j=1}^{l} f_j$$

for a considered fault group $\mathcal{F} = \{f_1, ..., f_l\}$.

Furthermore, to ensure that at least one fault is detected by the generated test pattern, the following constraint is added:

$$f_1 \vee f_2 \vee ... \vee f_l = 1$$

The other constraints of an instance, such as the constraints for functionality of the circuit, are the same as those for the PBO-based ATPG

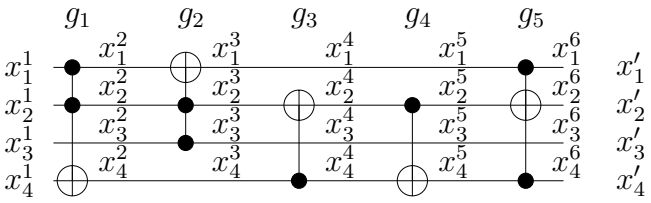
introduced in Section 4.2.

Example 6.4. *To encode the DTPG-problem for the circuit in Fig. 6.1, with the two faults* $(1, 1)$ *and* $(5, 1)$*, we introduce variables* $x_1^1, \ldots, x_4^1, \ldots, \ldots, x_1^6, \ldots, x_4^6$ *to represent the assignment to the respective circuit signals and variables* f_1 *and* f_2 *to represent the activation of faults. The respective constraints for this instance are partially shown in Fig. 6.3.*

If a test pattern can be generated that detects at least one other fault, but as few as possible, the resulting instance is passed to a PBO solver to determine a satisfying assignment. In this case, the respective test pattern can be obtained from the assignment to all variables $x_n^1, \ldots, x_1^1$. If no satisfying solution can be determined (i.e., if the PBO solver returns unsatisfiable), all the faults in $\mathcal{F}$ have been proven to be untestable (although this is usually determined prior to fault diagnosis).

Example 6.5. *Consider again the instance in Fig. 6.3. One satisfying solution to this instance leads to the test pattern 0110 (obtained from the assignment* $x_1^1 = 0$*,* $x_2^1 = 1$*,* $x_3^1 = 1$*, and* $x_4^1 = 0$*). This test pattern detects the fault* $(1, 1)$*, but not the fault* $(5, 1)$ *in the circuit shown in Fig. 6.1.*

While this encoding offers an improved alternative for DTPG of reversible circuits, it does not entirely solve the fault-equivalence checking problem. Therefore, it is necessary to introduce a further check, as discussed in the next section.



(a) Variables

Functionality of the circuit:

$x_1^2 = x_1^1$, $x_2^2 = x_2^1$, $x_3^2 = x_3^1$, $x_4^2 = x_4^1 \oplus (x_1^1 \wedge x_2^1)$

$x_1^3 = x_1^2 \oplus (x_2^2 \wedge x_3^2)$, $x_2^3 = x_2^2$, $x_3^3 = x_3^2$, $x_4^3 = x_4^2$

$x_1^4 = x_1^3$, $x_2^4 = x_2^3 \oplus x_4^3$, $x_3^4 = x_3^3$, $x_4^4 = x_4^3$

$x_1^5 = x_1^4$, $x_2^5 = x_2^4$, $x_3^5 = x_3^4$, $x_4^5 = x_4^4 \oplus x_2^4$

$x_1^6 = x_1^5$, $x_3^6 = x_3^5$, $x_4^6 = x_4^5$, $x_2^6 = x_2^5 \oplus (x_1^5 \wedge x_4^5)$

Activation of faults $(1,1)$ *and* $(5,1)$*:*

$f_1 = \overline{x_1^1} \wedge x_2^1$ *for* $(1,1)$

$f_2 = \overline{x_1^5} \wedge x_4^5$ *for* $(5,1)$

At least one fault should be detected:

$f_1 \vee f_2 = 1$

Objective function:

$\mathcal{O} = f_1 + f_2$

(b) PBO Constraints

FIGURE 6.3: PBO formulation for DTPG problem

6.2.2 Improved Fault-equivalence Checking

In most cases, the DTPG approach introduced in the last section generates a test pattern that detects at least one other fault, but as few as possible, in a considered group. However, sometimes only test patterns that can be determined detect *all* faults. Then, it remains unclear whether this test pattern distinguishes at least one fault. In fact, a single test pattern can still distinguish two faults if it leads to different responses at the primary output[1].

Whether the test pattern obtained by DTPG distinguishes at least one fault can be easily checked by fault simulation. If the simulation leads to the same responses for all faults, another test pattern that distinguishes the faults might exist. The proposed fault equivalence checking method either determines such a test pattern (showing that the faults are not equivalent and providing a new diagnostic test pattern) or proves that no such test pattern exists (showing that the faults are equivalent).

Therefore, we apply an adjusted structural analysis that also exploits the good observability of reversible circuits. The general idea is based on the following observations.

Without loss of generality, assume two SMCFs f_1 (occurring in gate g_a) and f_2 (occurring in gate g_b with $a < b$) have to be distinguished. Fur-

[1]This is also the reason why in the proposed DTPG approach a test pattern is determined that detects as few as possible other faults and not a test pattern that

ther assume that (1) no test pattern exists that only detects either f_1 or f_2 (such a test pattern would have been determined during DTPG) and (2) the determined test pattern does not distinguish the two faults. For fault f_1, the faulty behavior is introduced in the target line of gate g_a and it propagates toward the primary outputs. The faulty behavior can be detected by at least one value of the primary output-lines (compared to the expected fault-free value). To distinguish the fault f_1 from f_2, it must be possible to distinctively retrace at least one of these faulty output-lines to f_1. This is only possible, if fault f_1 can be propagated to lines that neither influence the target nor the control lines of g_b, or at least one assignment to the control lines of g_b can be changed because of f_1. Otherwise, the faulty behavior of the primary output lines could also be caused by f_2.

Example 6.6. *Consider again the circuit from Fig. 6.1 and the two faults $f_1 = (1, 2)$ in gate g_1 and $f_2 = (5, 4)$ in gate g_5. The DTPG approach has already confirmed that no test pattern exists that solely detects one of these faults. Furthermore, a structural analysis shows that in the case of fault f_1, the faulty behavior is always propagated through the fourth circuit line (here, the faulty behavior is introduced) and the second circuit line (propagated by gate g_3), i.e., the faulty behavior of f_1 can only be detected at these two lines.*

does not detect at least one of the other faults.

However, the second circuit line is the target line where the faulty behavior of f_2 would be introduced. Furthermore, the fourth circuit line is a control line of g_5. Because f_1 and f_2 are always detected by the same test patterns and additionally those test patterns always require a certain assignment to the control lines of g_5 to activate f_2, the fourth line leads to an indistinguishable response. Thus, it is impossible to distinguish whether the faulty behavior is caused by f_1 or f_2. The two faults are equivalent.

Based on these observations, two faults f_1 and f_2 can easily be classified to be fault equivalent. Therefore, only a simple structural analysis has to be performed. The structural analysis can be decomposed into two questions:

Question 1 Is there at least one line that propagates the fault f_1 and influences neither the target line nor any of the control lines of gate g_b? If there is no such line, then

Question 2 Is there a test pattern that does not trigger f_2 when f_1 occurs?

If at least one of these questions is true, then it is ensured that there is a test pattern to distinguish the two faults.

To address the first question, if the analysis shows that at least one line propagating the fault f_1 may be influencing neither the target line

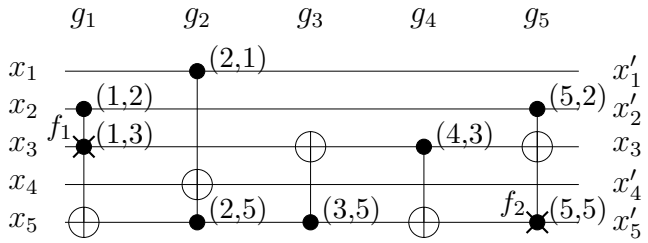


FIGURE 6.4: Example circuit for proof of fault equivalence

nor any of the control lines of gate g_j, then we must check whether a test pattern can be determined that enables the faulty behavior to propagate to this line. This is illustrated by the following example.

Example 6.7. *Consider the circuit shown in Fig. 6.4 and the two faults $f_1 = (1, 3)$ in gate g_1 and $f_2 = (5, 5)$ in gate g_5. The DTPG approach has already confirmed that no test pattern exists that only detects one of these faults. The structural analysis shows that there is one circuit line, namely the fourth line, that propagates the fault f_1; however, the fourth lines does not influence the target line nor any of the control lines of gate g_5. However, to propagate that fault through the fourth circuit line, gate g_2 needs to be activated. Therefore, a test pattern needs to be determined that detects f_1 and that also sets all control lines of g_2 to one.*

To generate such a test pattern, a SAT solver is utilized. Therefore, an instance as described in Section 3.3 is generated. However, instead of the fault-free circuit, the faulty circuit (injected f_1) is encoded in the

instance. Further, the following constraint is added:

$$\bigwedge_{x_c \in C_{act}} x_c = 1$$

where g_{act} is the gate to be activated, thereby ensuring that g_{act} propagates the faulty behavior.

If the SAT solver returns a satisfying assignment, then a test pattern can be obtained that distinguishes f_1 and f_2. If the instance is unsatisfiable, it must be checked whether other gates and circuit lines exist through which the faulty behavior of f_1 can propagate. If for all such cases, no test pattern can be obtained, then the second question in the structural analysis has to be considered.

To generate a test pattern that does not trigger the fault f_2, when the fault f_1 occurs, the SAT solver is used again. Therefore, an instance like that described for the first question is generated (but without the constraint for the activated gate g_{act}). Additionally, the constraint

$$\bigwedge_{x_c \in C \setminus i} x_c \wedge \overline{x_i} = 0$$

is added to ensure that fault f_2 in gate g_b is not triggered.

If a new test pattern can be obtained from the satisfying assignment of the SAT solver, then f_1 and f_2 are distinguished by this test pattern. Otherwise, f_1 and f_2 are proven to be fault equivalent. The structural

analysis is very efficient; in fact, it needs only linear time with respect to the number of gates in the circuit. In contrast, additional ATPG runs may require considerably more time. However, these checks are rarely necessary. In almost all cases, either DTPG or the structural analysis leads to a result. That is, by combining the improved DTPG and fault-equivalence checking methods, an improved fault diagnostic flow results as summarized in the following section and experimentally evaluated in Section 6.3.

6.2.3 Resulting Fault Diagnostic Flow

Fig. 6.5 shows the proposed fault diagnostic flow for reversible circuits; this flow incorporates the methods introduced in previous sections. Initially, the test engineers provide a fault list and an initial testset (a). If this testset already distinguishes all faults, a fault dictionary is returned and the fault diagnosis terminates (b). Otherwise, the proposed DTPG method from Section 6.2.1 is applied to a group of faults that are still not distinguished (c). If a (diagnostic) test pattern that distinguishes at least one fault returns, it is added to the testset (d). Otherwise, fault-equivalence checking, as proposed in Section 6.2.2, is invoked. That is, a structural analysis (e) checks whether the respective faults can be classified to be equivalent. If they are equivalent, they are added to the fault-equivalence class (f). Otherwise, we check whether a test pattern can be

FIGURE 6.5: The proposed fault diagnosis flow

determined that satisfies the restrictions discussed in Section 6.2.2 (g). If this is successful, the resulting (diagnostic) test pattern is added to the testset (d). Otherwise, the faults have been proven to be equivalent and they are added to the fault-equivalence class (f). This process continues until all faults are either distinguished or classified to be fault equivalent.

6.3 Experimental Results

The proposed approaches were implemented in C++ on top of RevKit [SFWD10]. As underlying solving engines, *clasp* [GKNS07] (for DTPG) and *MiniSAT* [ES04] (for the additional ATPG runs during fault-equivalence checking) were used. The performance of the approach proposed in Section 6.2.2 was evaluated on a set of benchmark circuits taken from RevLib [WGT+08]. For comparison, those circuits also diagnosed using the conventional methods reviewed in Section 6.1. Single missing control faults are considered in the following. For other fault models (e.g., SMGF), the proposed approaches can be easily extended by just adjusting certain assignment conditions. Therefore, the results from the evaluation should be very similar. All experiments were carried out on an AMD Opteron ×4 processor with 8GB main memory running Linux. The timeout was set to 3 600 CPU seconds (denoted by *T.O.*).

The results are presented in Tables 6.1 and 6.2. The first four columns

TABLE 6.1: Experimental results for DTPG w/o improved approaches (Size of testsets)

Circuit	d	n	c	$\|\mathcal{F}\|$	#TS	CONVENTIONAL DTP	CONVENTIONAL *EF*	PROPOSED DTP	PROPOSED *EF*
WITHOUT CONSTANT PRIMARY INPUTS									
4_49_16	16	4	0	24	5	6	0	6	0
ham7_104	23	7	0	34	6	6	0	6	0
ham15_108	70	15	0	125	9	13	0	13	0
ham15_109	109	15	0	126	9	11	0	11	0
ham15_107	132	15	0	352	16	53	28	70	28
hwb7_61	236	7	0	693	27	31	26	31	26
hwb7_62	331	7	0	582	28	31	0	31	0
hwb8_113	637	8	0	2214	44	68	99	65	99
hwb9_119	1544	9	0	5812	83	107	433	110	433
hwb9_123	1959	9	0	3596	80	100	12	98	12
WITH CONSTANT PRIMARY INPUTS									
mini-alu_84	20	10	6	27	4	8	0	8	0
rd84_142	28	15	7	49	8	14	0	14	0
4_49_7	42	15	11	61	5	8	0	8	0
hwb5_13	88	28	23	131	7	10	0	10	0
hwb6_14	159	46	40	241	7	10	0	10	0
ex5p	647	206	198	904	18	24	0	24	0
spla	1709	489	473	2711	19	>54	–	105	4
table3	1988	554	540	2997	23	>34	–	40	4
alu4	2186	541	527	3390	18	>40	–	47	2
ex1010	2982	670	660	4543	25	29	0	29	0

Circuit: name of the circuit; d: number of gates; n: number of circuit lines; c: number of constant inputs; $|\mathcal{F}|$: number of faults to be tested; #TS: number of test patterns in the provided testset; DTP: number of diagnostic test patterns obtained by the respective approaches; EF: number of equivalent faults;

TABLE 6.2: Experimental results for DTPG w/o improved approaches (Runtime)

Circuit	d	n	c	$\|\mathcal{F}\|$	CONVEN. DTPG	PROPOSED DTPG	TIME IMPR (%)
WITHOUT CONSTANT PRIMARY INPUTS							
4_49_16	16	4	0	24	0.03	0.02	33.33
ham7_104	23	7	0	34	0.02	0.02	0
ham15_108	70	15	0	125	0.69	0.32	53.62
ham15_109	109	15	0	126	1.55	0.32	79.35
ham15_107	132	15	0	352	43.51	8.4	80.69
hwb7_61	236	7	0	693	11.29	12.86	-13.91
hwb7_62	331	7	0	582	8.11	8.18	-0.86
hwb8_113	637	8	0	2214	204.45	203.05	0.68
hwb9_119	1544	9	0	5812	2667.42	2271.47	14.84
hwb9_123	1959	9	0	3596	752.88	657.92	12.61
WITH CONSTANT PRIMARY INPUTS							
mini-alu_84	20	10	6	27	0.07	0.06	14.29
rd84_142	28	15	7	49	0.17	0.15	11.76
4_49_7	42	15	11	61	0.16	0.11	31.25
hwb5_13	88	28	23	131	0.72	0.58	24.14
hwb6_14	159	46	40	241	0.93	0.84	9.68
ex5p	647	206	198	904	49.95	17.63	64.70
spla	1709	489	473	2711	*T.O.*	796.73	>87.69
table3	1988	554	540	2997	*T.O.*	281.92	>94.36
alu4	2186	541	527	3390	*T.O.*	463.84	>93.84
ex1010	2982	670	660	4543	2092.82	386.06	68.32

Circuit: name of the circuit; d: number of gates; n: number of circuit lines; c: number of constant inputs; $|\mathcal{F}|$: number of faults to be tested; TIME: runtime in CPU seconds; Impr.: improvement of the proposed diagnosis flow with respect to the runtime

characterize the evaluated circuits: (1) name of the circuit, (2) number d of gates, (3) number n of circuit lines, and (4) number c of constant inputs. Column $|\mathcal{F}|$ denotes the number of faults to be considered. In Table 6.1, the column labeled #TS contains the number of test patterns in the provided testset. This testset was obtained by the PBO-based ATPG introduced in Section 4.2. The remaining columns in these two tables present the results obtained by applying the conventional diagnostic flow (Section 6.1) and the proposed diagnostic flow (Section 6.2). In Table 6.1, the column contain the number of diagnostic test patterns (denoted by *DTP*) and number of equivalent faults (denoted by *EF*). In Table 6.2, the runtime in CPU seconds and the improvement in the proposed diagnostic flow with respect to runtime are presented.

The results confirm that the application of conventional approaches leads to a fault diagnostic flow that is applicable to large circuits. Furthermore, applying the proposed diagnostic flow does not lead to a significant decrease in the quality of the results. In fact, the numbers of generated diagnostic test patterns are almost the same as those generated by conventional methods; sometimes (e.g., for *hwb7_61* and *hwb9_123*) the number is slightly better.

However, there are significant differences in runtimes. For all benchmarks, the proposed diagnostic flow clearly outperformed the conventional methods. In the best case, improvements of more than an or-

der of magnitude were achieved. Additionally, the proposed approach scales better for larger circuits. Thus, fault diagnosis of reversible circuits clearly profits from approaches that explicitly exploit the inherent reversibility of the circuits.

6.4 Summary and Future Work

Based on conventional diagnostic methods, an improved diagnostic flow (including improved diagnostic test pattern generation and improved equivalence checking) was proposed for reversible circuits. Thereby, the advantageous properties of reversible circuits were explicitly exploited so that diagnosis can be performed significantly faster than by pure applications of conventional methods.

In future work, the diagnosis of multiple faults should be considered. In fact, the SAT-based method for checking for fault masking (introduced in Section 5.4), which determines all undetected faults for a given test pattern, can be reused for this task. Then, instead of determining all undetected faults by a test pattern (with the fault-free output), all possible faults for a faulty result (primary output) can be found by the SAT-based approach (also called *Effect-Cause Analysis* [WWW06; JG03]).

Chapter 7

Conclusion

In recent decades, computer technologies have continued to advance with great success. However, in the near future, the ongoing miniaturization of integrated circuits will reach its limits. To circumvent those limits, alternatives will be needed. Reversible logic provides an attractive alternative. Manufacturing conventional circuits has taught us that it is important to develop efficient methods for testing circuits and detecting faults in circuits. However, few algorithms are available for *Automatic Test Pattern Generation* (ATPG) and for *fault diagnosis* of reversible circuits. In general, study of testing for reversible circuits is just beginning, only circuits containing a few gates have been considered.

In this thesis, efficient ATPG methods and improved fault diagnostic approaches have been proposed for reversible circuits. The current

7. CONCLUSION

ATPG methods were reviewed and the limited applicability of the methods for circuits having additional constraints (such as constant inputs) were discussed. Then, a SAT-based ATPG was introduced to be an improvement when applied to circuits having constant inputs. Furthermore, schemes and approaches were proposed and tested for generating compact and even minimal testsets. These include a fault-ordering scheme based on the reversibility of the circuits, a PBO-based method for generating compact testsets, and a SAT-based algorithm for determining minimal testsets. These proposed ATPG approaches are complementary for different application scenarios and test goals. Moreover, testing multiple faults was considered. Using SAT and PBO solvers, a testing flow was introduced that can efficiently generate complete testsets for the *Multiple Missing Gate Fault* model in circuits containing more than 900 gates. Further, new approaches for efficient fault diagnosis of reversible circuits were proposed and tested. By exploiting the advantage of reversibility, diagnosis can be performed significantly faster than conventional methods. Overall, using the approaches proposed in this thesis, the problems posed by testing and diagnosis of reversible circuits can be solved more efficiently.

For future work, two tasks remain prominent. The first is fault diagnosis for multiple faults. As has already been investigated for conventional circuits, *Effect-Cause Analysis* [WWW06; JG03] can be utilized

for diagnosing multiple faults. Therefore, formal methods should be introduced for this task; e.g., the SAT-based method for checking for fault masking, introduced in Section 5.4, could be modified to determine all possible faults for a given faulty output. The second task is to generate complete testsets for multiple faults under other fault models, such as the *Multiple Missing Control Fault* model. Hence, future work will need to focus on improvements or modifications of the methods presented in Chapter 5.

References

[ABF90] M. Abramovici, M. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[ABKS03] V. D. Agrawal, D. H. Baik, Y. C. Kim, and K. K. Saluja. Exclusive test and its applications to fault diagnosis. In *VLSI Design*, pages 143–148, 2003.

[AFPB01] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana. Fault equivalence identication using redundancy information and static and dynamic extraction. In *IEEE VLSI Test Symposium*, 2001.

[Agr81] V. D. Agrawal. An information theoretic approach to digital fault testing. *IEEE Trans. on Comp.*, 30(8):582–587, 1981.

[BB09] 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, pages 174–177, 2009.

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.

[Ben73] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev*, 17(6):525–532, 1973.

[Chr80] J. Chrissis, editor. *The Solution of Nonlinear Pseudo-Boolean Optimization Problems Subject to Linear Constraints*. Virginia Polytechnic Institute and State University, 1980.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Symposium on Theory of Computing*, pages 151–158, 1971.

[DEF+08] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Trans. on CAD*, 27:1329–1333, 2008.

[DEFT09] R. Drechsler, S. Eggersglüss, G. Fey, and D. Tille. *Test Pattern Generation using Boolean Proof Engines*. Springer, 2009.

[DGP97] D. Du, J. G, and P. Pardalos, editors. *Satisfiability Problem: Theory and Applications: Dimacs Workshop, March 11-13, 1996*, volume 35. American Mathematical Soc., 1997.

[DLL62] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:506–521, 1960.

[DV02] B. Desoete and A. De Vos. A reversible carry-look-ahead adder using control gates. *INTEGRATION, the VLSI Jour.*, 33(1-2):89–104, 2002.

[ED12] S. Eggersglüss and R. Drechsler. *High Quality Test Pattern Generation and Boolean Satisfiability*. Springer, 2012.

[ES04] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[ES06] N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[FT82] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[FTM08] 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*, pages 1378–1381, 2008.

[FTR07] K. Fazel, M.A. Thornton, and J.E. Rice. ESOP-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206 –209, 2007.

[GKNS07] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Int'l Joint Conference on Artificial Intelligence*, pages 386–392, 2007.

[GN02] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe*, pages 142–149, 2002.

[HPB04] J. P. Hayes, I. Polian, and B. Becker. Testing for missing-gate faults in reversible circuits. In *Asian Test Symp.*, pages 100–105, 2004.

[JG03] N. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, 2003.

[Kop02] J. Kopplin. An illustrated history of computers,

2002. http://www.computersciencelab.com/ ComputerHistory/ HistoryPt4.htm.

[KPKR95] S. Kajihara, I. Pomeranz, K. Kinoshita, and S.M. Reddy. Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. *IEEE Trans. on CAD*, 14 , Issue: 12:1496 – 1504, Dec. 1995.

[Lan61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183, 1961.

[Lar92] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[LCD+02] 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*, page 94, 2002.

[LS92] S.Y. Lee and K.K. Saluja. An algorithm to reduce test application time in full scan designs. In *Int'l Conf. on CAD*, pages 17 – 20, 1992.

[MCZ+01] M.W.Moskewicz, C.F.Madigan, Y. Zhao, L. Zhang, and S.Malik. Chaff: Engineering an efficient sat solver. In *Design Automation Conf.*, pages 530–535, 2001.

REFERENCES

[MD04] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on CAD*, 23(11):1497–1509, 2004.

[MGHD09] 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.

[MMD03] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conf.*, pages 318–323, 2003.

[MS99] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[MWD10] D. M. Miller, R. Wille, and R. Drechsler. Reducing reversible circuit cost by adding lines. In *Int'l Symp. on Multi-Valued Logic*, 2010.

[NC00] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge Univ. Press, 2000.

[PBG05] M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[PBP05] D. Pierce, J. Biamonte, and M. Perkowski. Test set generation and fault localization software for reversible circuits. In *Int'l Symp. on Representations and Methodologies for Emergent Computing Technologies*, September 2005.

[Per85] A. Peres. Reversible logic and quantum computers. *Phys. Rev. A*, (32):3266–3276, 1985.

[PFBH05] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes. A family of logical fault models for reversible circuits. In *Asian Test Symp.*, pages 422–427, 2005.

[PHM04] K. N. Patel, J. P. Hayes, and I. L. Markov. Fault testing for reversible circuits. *IEEE Trans. on CAD*, 23(8):1220–1230, 2004.

[RTPP04] K. Ramasamy, R. Tagare, E. Perkins, and M. Perkowski. Fault localization in reversible circuits is easier than for classical circuits. In *Workshop on Logic and Synthesis*, 2004.

[SFWD10] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. RevKit: A toolkit for reversible circuit design. In *Workshop on Reversible Computation*, pages 69–72, 2010. RevKit is available at http://www.revkit.org.

[SPMH03] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes.

Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, 2003.

[SS05] H. M. Sheini and K. A. Sakallah. Pueblo: A modern pseudo-boolean SAT solver. In *Design, Automation and Test in Europe*, pages 684–685, 2005.

[SVAV05] A. Smith, A. G. Veneris, M. F. Ali, and A. Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Trans. on CAD*, 24(10):1606–1621, 2005.

[TED10] Daniel Tille, Stephan Eggersglüß, and Rolf Drechsler. Incremental solving techniques for sat-based atpg. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Volume 29, Number 7:pp. 1125–1130, July 2010.

[Tof80] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

[Tse68] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in:

J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, pp. 466-483, 1983.).

[VCAA04] A. Veneris, R. Chang, M. S. Abadir, and M. Amiri. Fault equivalence and diagnostic test generation using ATPG. In *IEEE International Symposium on Circuits and Systems*, 2004.

[VMH07] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Checking equivalence of quantum circuits and states. In *Int'l Conf. on CAD*, pages 69–74, 2007.

[VSB+01] 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*, 414:883, 2001.

[WD09] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conf.*, pages 270–275, 2009.

[WGF+09] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler. Debugging of Toffoli networks. In *Design, Automation and Test in Europe*, pages 1284–1289, 2009.

[WGT+08] 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*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[Wik12] Intel core, 2012. http://en.wikipedia.org/ wiki/ Intel_Core, last modified on 20 November 2012.

[WLTK08] S.-A. Wang, C.-Y. Lu, I-M. Tsai, and S.-Y. Kuo. An XQDD-based verification method for quantum circuits. *IEICE Transactions*, 91-A(2):584–594, 2008.

[WOD10] R. Wille, S. Offermann, and R. Drechsler. SyReC: A programming language for synthesis of reversible circuits. 2010. submitted.

[WWW06] L. Wang, C. Wu, and X. Wen. *VLSI Test Principles and Architectures - Design for Testability.* Morgan Kaufmann Publishers, 2006.

[WZD11] 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*, pages 120–125, 2011.

[WZD13] R. Wille, H. Zhang, and R. Drechsler. Fault ordering for automatic test pattern generation of reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, Toyama, 2013.

[ZFWD11] H. Zhang, S. Frehse, R. Wille, and R. Drechsler. Determining minimal testsets for reversible circuits using boolean satisfiability. In *10th IEEE Africon*, 2011.

[ZWD10] H. Zhang, R. Wille, and R. Drechsler. Sat-based atpg for reversible circuits. In *5th International Design & Test Workshop (IDT)*, pages 149–154, 2010.

[ZWD11] H. Zhang, R. Wille, and R. Drechsler. Improved fault diagnosis for reversible circuits. In *Asian Test Symposium (ATS)*, 2011.

List of Figures

LIST OF FIGURES

LIST OF FIGURES

Nomenclature

ATPG Automatic Test Pattern Generation

CNF Conjunctive Normal Form

DTPG Diagnostic Test Pattern Generation

MMCF Multiple Missing Control Fault

MMGF Multiple Missing Gate Fault

NP Nondeterministic polynomial

PBO Pseudo-Boolean optimization

SAT Boolean satisfiability

SMCF Single Missing Control Fault

SMGF Single Missing Gate Fault

LIST OF FIGURES

$\boldsymbol{C}$	Set of control lines
$\boldsymbol{c}$	Number of constant inputs
$\boldsymbol{d}$	Number of gates (depth) of a circuit
$\mathcal{F}$	Fault list
$\boldsymbol{n}$	Number of inputs of a circuit
$\boldsymbol{O}$	Objective function
$\boldsymbol{t}$	Target line

Index

INDEX