FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Automatic Test Pattern Generation of Zero-Aliasing Test for General Output Response Compactor

by

Robert Hülle

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Department of Digital Design

Prague, August 2022

Supervisor:
doc. Ing. Petr Fišer, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Co-Supervisor:
doc. Ing. Jan Schmidt, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

Testing of digital circuits is increasingly important and complex, as the size and complexity of contemporary circuits increase. Two topics related to digital circuit testing are the aim of this dissertation thesis. The main topic is a generation of test sets with zero aliasing in temporal compactors. The secondary topic is a generation of test sets for application-oriented testing of field-programable gate arrays.

An approach to test generation of application-oriented tests for circuits implemented in FPGAs is presented. An ATPG natively using a combined fault model of stuck-at and bit-flip faults is proposed. The properties and relations of the two fault models are analyzed. This work forms a basis and provides necessary tools for subsequent research, the main aim of this thesis, and also further research on application-oriented FPGA testing that is not part of this thesis.

An algorithm for generating tests with zero fault aliasing in output response compactors is proposed. Output response compaction is an important tool in digital circuit testing and design for testability techniques. The compaction decreases the amount of data needed to be transferred from the tested circuit, at the price of introducing fault aliasing. Typical methods to reduce fault aliasing include manipulating existing tests or changing the compactor design. The proposed method is able to directly generate tests with reduced aliasing without changing the compactor design. This leads to higher fault coverage and possibly smaller compactors.

In particular, the main contributions of the dissertation thesis are as follows:

1. An encoding scheme for bit-flip fault model for application-oriented FPGA testing.
2. Algorithm for generating a zero-aliasing test for general output response compactor.

Keywords:

ATPG, Multiple-Target Test Generation, Output Response Compaction, Output Response Aliasing, zero aliasing, SAT, Pseudo-Boolean Optimization, Multiple-Input Signature Register.

Abstrakt

Důležitost a obtížnost testování současných číslicových obvodů roste se zvyšující se komplexitou dnešních obvodů. Tato disertační práce má 2 témata. Hlavním tématem je generování testu s nulovým maskováním poruch v kompaktorech odezvy. Vedlejším cílem je generování testu pro aplikační testování obvodů implementovaných v FPGA čipech.

V první části této práce je prezentován postup pro generování testu pro aplikační testování obvodů FPGA. Je navržen algoritmus využívající kombinovaného poruchového modelu s poruchami trvalá nula a změna bitu. Tyto poruchové modely, jejich vlastnosti a vzájemný vztah jsou analyzovány. Tento výstup poskytuje nutné nástroje a postupy pro další výzkum: hlavní téma této práce i pokračující výzkum aplikačního testování FPGA, který není součástí této práce.

V druhé části této práce je prezentován algoritmus pro generování testu s nulovým maskováním poruch v kompaktorech odezvy. Kompakce odezvy je důležitý nástroj v testování obvodů a v návrhu pro testovatelnost. Kompakce zmenšuje množství dat, která je nutné přenést z testovaného obvodu, za cenu vzniku maskování poruch. Typické metody pro potlačení maskování jsou úprava již existujícího testu, nebo změna návrhu kompaktoru. Navržený algoritmus je schopný přímo vytvořit test se sníženým maskováním beze změny v návrhu kompaktoru. To vede k vyššímu poruchovému pokrytí, případně k možnosti použití menších kompaktorů.

Hlavní přínosy této práce jsou následující:

1. Postup pro zakódování poruchového modelu změněného bitu v aplikačním testování obvodů FPGA.

2. Algoritmus pro generování testu s nulovým maskováním pro obecný sekvenční kompaktor odezvy.

Klíčová slova:
ATPG, Multiple-Target Test Generation, Output Response Compaction, Output Response Aliasing, zero aliasing, SAT, Pseudo-Boolean Optimization, Multiple-Input Signature Register.

Acknowledgements

First of all, I would like to express my gratitude to my dissertation thesis supervisor, doc. Ing. Petr Fišer, Ph.D., and co-supervisor, doc. Ing. Jan Schmidt, Ph.D. They have been a constant source of encouragement and insight during my research and helped me with numerous problems and professional advancements.

Special thanks go to the staff of the Department of Digital Design, who maintained a pleasant and flexible environment for my research. I would like to express special thanks to the department management for providing most of the funding for my research.

My research has also been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/121/OHK3/1T/18, SGS17/213/OHK3/3T/18, SGS20/211/OHK3/3T/18, and by the Czech Grant Agency, grant No. GA16-05179S.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Digital Circuits

ASIC	Application-Specific Integrated Circuit
CA	Cellular Automaton
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
LFSR	Linear-Feedback Shift Register
LUT	Look-Up Table

Testing

ADI	Accidental Detection Index
ATE	Automated Test Equipment
ATPG	Automated Test Pattern Generator
BF	Bit-Flip fault
BIST	Built-In Self Test
CUT	Circuit Under Test
DfT	Design for Testability
MISR	Multiple-Input Signature Register
MTTG	Multiple-Target Test Generation
ORA	Output Response Analyzer
PI	Primary Input(s)
PO	Primary Output(s)
PPI	Pseudo-Primary Input(s)
PPO	Pseudo-Primary Output(s)
SA	Stuck-At fault

Logical Gates

AND	And Gate, $y = a \wedge b$
NAND	Nand Gate, $y = \neg(a \wedge b)$
OR	Or Gate, $y = a \vee b$
NOR	Nor Gate, $y = \neg(a \vee b)$
XOR	Exclusive-OR Gate, Nonequivalence, $y = \neg(a \iff b)$
XNOR	Exlusive-NOR Gate, Equivalence, $y = a \iff b$

Miscellaneous Abbreviations

CNF	Conjunctive Normal Form
GF(q)	Galois Field of order q
PBO	Pseudo-Boolean Optimization problem
PRNG	Pseodo-Random Number Generator
SAT	Boolean Satisfiability problem
SOP	Sum of Products

Chapter 1

Introduction

This chapter introduces the area of research, describes the motivation, and states the goals for this dissertation thesis.

1.1 Motivation

Today's world and human society run on digital systems. From entertainment to essential services to life-critical systems and beyond, our reliance on digital systems is ever increasing. It is therefore essential that these systems are reliable and their eventual failure is not detrimental to human life, health, property, or the environment. The testing of digital circuits is one of the tools that we use to prevent such hazards.

As the complexity and size of digital circuits continue to increase, so do the complexity and price of their testing. Therefore, even though the topic has been thoroughly researched, there is room, and indeed the need, for further improvement and innovation.

1.2 Problem Statement

Main factors that contribute to the testing cost are the *test application time*, the *number of test points*, the *volume of transferred data*, and the *memory of the test equipment* (ATE). To decrease these costs, the design of the would-be-tested circuit is changed to aid in the testing. These techniques are collectively called *Design for Testability* (DfT). Without DfT, the testing of complex circuits is not only expensive but practically impossible.

The two techniques that are relevant to this dissertation thesis are the *Built-In Self-Test* (BIST) and the *scan chain* design. A part of these techniques is the *Output Response Analysis* (ORA), a method used to *compact the output response*, i.e, decrease the amount of data transferred from the circuit during its testing. While the amount of transferred data can be decreased, the response compaction can introduce *aliasing*, a loss of information about the presence of a fault, leading to a decrease in the fault coverage.

The aim of this dissertation thesis is to improve the test generation to decrease the aliasing during the response compaction while also decreasing the size of the compactor.

1.3 Goals of the Dissertation Thesis

1. An encoding scheme for a bit-flip fault model for application-oriented FPGA testing.
2. Algorithm for generating a zero-aliasing test for general output response compactor.

1.4 Structure of the Dissertation Thesis

The thesis is organized into 5 chapters as follows:

1. *Introduction*: Describes the motivation behind our efforts together with our goals. There is also a list of contributions of this dissertation thesis.
2. *Background and State-of-the-Art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.
3. *Encoding Bit-Flip Fault Model in SAT Instance*: Presents the first contribution of this thesis, an ATPG for a combined fault model for FPGA.
4. *Augmented ATPG with Zero-Aliasing Constraints*: Presents the second contribution of this thesis, an ATPG preventing aliasing in the output response compaction.
5. *Conclusions*: Summarizes the results of our research, suggests possible topics for further research and concludes the thesis.

Chapter 2

Background and State-of-the-Art

In this chapter, we present a brief overview of the state-of-the-art and theoretical background from the area of digital circuits testing, automated test pattern generation and output response compaction.

2.1 Digital Circuits Testing

The testing of digital circuits is important in all phases of the circuit life cycle. Circuits are tested several times during the manufacturing process and their distribution, e.g., on a wafer, before and after encapsulation. A common requirement is that circuits can be tested during their deployment when they are integrated in larger systems.

2.1.1 Combinational Logic Testing

The basic idea behind the circuit testing is that we apply some input to the *circuit under test* (CUT) and check for expected output. If the actual output differs from the precomputed output, we know that there is something wrong with the circuit, there is a defect present. There can be many types of defects present in the circuit; for this thesis, we only consider defects that can be modeled as faults on the logical level, namely *stuck-at fault* (SA, S) and *bit-flip fault* (BF).

To test a circuit completely would mean to apply all possible vectors to its inputs. While this may be possible for small circuits, for those with hundreds of inputs it is not. As a compromise, we can test the circuit with a smaller number of test patterns that are carefully selected. A simple test would be to apply a certain number of pseudo-random test patterns. These can be implemented with *Pseudo-Random Number Generators* (PRNG), e.g., *Linear-Feedback Shift Register* (LFSR), ring generators, but also simple counters and adders. Other techniques can be used to improve this test, such as reseeding [69, 26, 65]. The effectiveness of this type of test is in its principle probabilistic.

The test patterns can be also computed with *deterministic* methods. These methods generate a test set suited to a specific circuit. They will ideally test for all faults or function-

ality of the CUT. These methods can be divided into two main groups, the *functional tests* that can test the circuit's functionality, and the *structural tests* that can test for known internal defects, modeled as faults. In this thesis, we use the structural test methods.

Combinational logic outputs, barring the signal propagation delay, depend only on the current value applied to its inputs. Thus, applied test patterns are fundamentally independent of each other and their ordering is, for testing, irrelevant. This property can be utilized for other objectives, e.g., compacting the test [49, 28, 70].

2.1.2 Sequential Logic Testing

The testing of sequential circuits is also done by applying some test patterns to the inputs of the CUT and observing for expected output response. In contrast to the combinational logic, however, we would need to test all possible sequences of inputs.

A shorter test can be computed by either functional or structural tests [47, 40]. The complexity of computing such tests is significantly higher and produced tests are also longer. This makes testing of such logic circuits infeasible.

2.1.3 Design for Test

A solution to testing sequential circuits is to design them to be easily testable. These *Design for Test* (DfT) techniques include strict separation of sequential and combinational logic, e.g., state memory and transition function for finite state machine (FSM). When properly separated, memory elements and combinational logic can be tested independently.

2.1.4 Scan Chain

Scan chain is a DfT method widely used to partition a sequential circuit into a combinational part and a memory. It works by connecting (chaining) all registers in the design to one or several shift registers. The operation of shifting the contents of such chain in or out is called scan.

In this design, the inputs and outputs of chained registers act as Pseudo-Primary Outputs (PPO) and Pseudo-Primary Inputs (PPI), respectively. The combinational part can be thus tested as a normal combinational circuit by applying a test pattern to its PIs as usual and its PPIs by the scan-in. Conversely, the response to the test is read as usual from POs and by the scan-out from PPOs.

It is possible to add scan registers to the circuit's PIs and POs. These registers are bypassed during normal operation but act as inputs and outputs during testing. This reduces the number of test points to only input and output of the scan chain and unifies the treatment of the circuit's Primary and Pseudo-Primary inputs/outputs. Additionally, the interconnections between different parts of the chip or between different chips can be also tested. Such arrangement is referred to as a boundary scan.

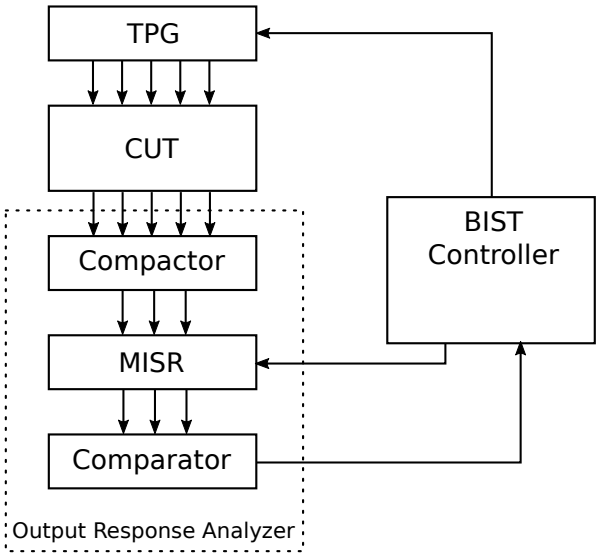

Figure 2.1: Example of a BIST architecture.

2.1.4.1 Built-In Self-Test

The *Built-In Self-Test* (BIST) is a DfT technique to add all circuitry needed for the application and evaluation of the test in the design. This makes the circuit capable of testing itself, without the need for external test equipment (ATE). This technique is used both in manufacturing testing and in in-application testing.

Typical BIST design, as shown in Figure 2.1, consists of:

- Circuit Under Test (CUT),
- Test Pattern Generator (TPG),
- Output Response Analyzer (ORA),
- BIST controller.

The *test pattern generator* is responsible for generating test stimuli for the CUT. Examples of how the TPG can be implemented include memory and PRNG.

The *output response analyzer* is responsible for checking for the expected response. It can be divided into the spatial and temporal compactor and comparator, described in greater detail later in this chapter.

2.2 Automated Test Pattern Generator

An *Automated Test Pattern Generation* (ATPG) is a process of automatic (algorithmic) search for a sequence of test patterns that can detect faults in the tested circuit. For testing contemporary digital systems it is necessary to create the test in a fast and automated way. There are two approaches to the test generation, *functional* testing and *structural* testing.

In *functional testing* we view the circuit as a black box and construct the test per the description of its behavior. The major disadvantage of such a test is that we are generally not able to compute the achieved coverage. Despite this, the method is still used for example to test Intellectual Property (IP) cores for which we may not know their internal structure.

In *structural testing* we view the circuit as a network of elementary parts, e.g., for Application Specific Integrated Circuits (ASIC) that would be interconnected standard cells. The elements of the circuit can and do have many kinds of *defects*, e.g., open circuit or open transistor. We model these *defects* on logical level as *faults*. It is important to choose such a fault model that can cover the most common defects.

The *stuck-at line* (S@) fault model is the most widely used one, for it is simple and can detect most of the common defects in an ASIC design. A S@ fault can be either S@0 or S@1, which means that the signal, where the fault is located, has a permanent logical value of 0 or 1, respectively. It is necessary to consider this fault on every output and input of a gate, or before and after each signal branching.

In this thesis, we work only with structural testing.

ATPGs for structural testing can be categorized into three groups: *structural* [58, 25, 24, 61], *algebraic* [62], and generators using *Boolean Satisfiability problem* (SAT) [34].

2.2.1 Fault Model

Physical defects in a circuit are modeled at the logic level as faults. Most commonly used fault models include:

Stuck-at line The stuck-at (SA, S) models a fault as a signal that has a permanent value of 0 (stuck-at 0) or 1 (stuck-at 1). In a simulation, the signal is connected to a constant driver. S model is the simplest fault model used in practice and is strong enough for most applications [39, 36].

Bridging The bridging fault models a defect where one signal is influenced by some other signal. Interaction between these signals can be complex; this fault can lead to a sequential behavior in combinational circuits.

Bit-flip The bit-flip (BF) models a *single event upset* (SEU). When SEU happens, single or multiple bits can be flipped in the configuration memory. This model is useful for an FPGA circuit, where the bit-flip in a configuration memory can cause a change of function in a *look-up table* (LUT) or change the signal routing in interconnection blocks.

2.2.2 Structural ATPG

One of the oldest algorithms for generating a test is the D-algorithm [58]. This algorithm works with the multi-valued logic of $\{0, 1, X, D, \bar{D}\}$, where D and $\bar{D}$ represent logical value of 1 changed to 0 due to the present fault in the circuit and vice versa.

The location of a fault is excited by setting the affected signal to an appropriate value (D or $\bar{D}$). Other signals are then set to propagate the changed value to the output and to excite it from inputs. When a conflict in value assignment is encountered, backtracking is used. The complexity of this algorithm is $O(2^s)$ where s is the number of signals in the circuit.

The PODEM algorithm is an improvement over D-algorithm [25], efficient, especially for a class of circuits that includes the error-correction type of circuits. Instead of trying to assign a value to all signals, only primary inputs (PIs) are considered, and other signal values are propagated from PIs. This reduces its complexity to only $O(2^n)$ where n is the number of PIs. Heuristics are used to efficiently search the space of PI combinations, but the search is still exhaustive.

The FAN algorithm [24] further improves upon PODEM by employing several techniques. These techniques include multiple parallel backtraces, early backtrace termination, assigning as many signal values as possible by unique implication, and unique path sensitization.

SOCRATES is yet another improvement of FAN algorithm [61]. It uses learned implications to speed-up assigning of signal values. The learning is static (implications are precomputed) and dynamic (implications are discovered during the run of the ATPG).

2.2.3 ATPG Based on Boolean Satisfiability

One method to generate a test pattern that is recently gaining popularity both in academia and industry is converting the ATPG problem to the solving Boolean Satisfiability Problem (SAT). SAT-based ATPGs are more flexible in general and are also proficient in identifying redundant faults [34, 18, 10].

2.2.3.1 Boolean Satisfiability Problem

The problem of satisfiability of Boolean formula is the problem of deciding whether there exists such an assignment of Boolean variables that satisfies given formula.

Definition 2.2.1. Boolean Satisfiability Problem (SAT)

Let there be a set of Boolean variables $V = \{v_1, v_2, \dots v_n\}$. A literal is Boolean variable v_i or its negation $\neg v_i$. A clause is a logical disjunction of one or more literals. The Boolean formula in Conjunctive Normal Form (CNF) is logical conjunction of zero or more clauses.

Given Boolean formula F, the Boolean Satisfiability Problem (SAT) is then a problem of determining whether there exists such assignment $V \to \{0, 1\}$ that satisfies the Boolean formula F, i.e., the Boolean formula evaluates to one for the given variables assignment.

Note 2.2.2. A Boolean formula that is not in CNF can be transformed to CNF in polynomial time and space using the Tseitin transformation [67]. While this transformation is not

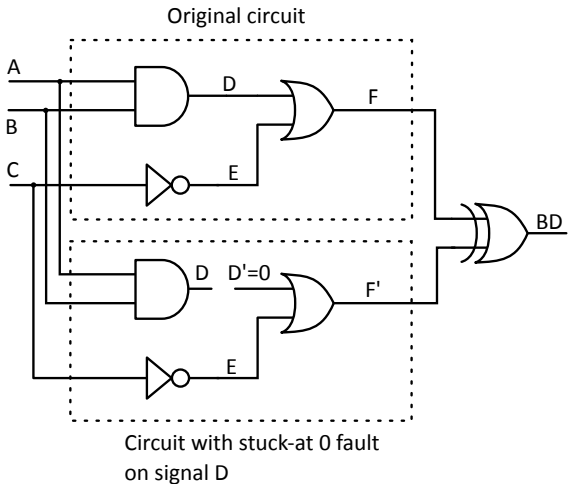

Figure 2.2: Example of a conceptual circuit for modeling fault in CUT.

mandatory for the SAT itself, it is practical because SAT solvers are working with this representation and as will be shown below, the ATPG process leads naturally to this form.

While SAT belongs to the NP-hard class of problems, in recent years there has been much progress in the field of SAT solvers. Modern solvers, such as MiniSAT [17], can quickly solve large SAT instances, especially instances that emerge in practical applications such as test pattern generation.

2.2.3.2 SAT-Based ATPG

SAT-based ATPG uses the SAT problem and an SAT solver to generate test patterns. The basic idea is similar to the Boolean difference [62]; the CUT is replicated in two instances, one as the original (unaltered) circuit, the other with a modeled fault (see Figure 2.2). This *conceptual circuit*, also called *miter*, is then transcribed as a (CNF-)SAT instance. [34, 18, 10].

The *miter* is constructed by connecting inputs of the fault-free and the faulty replica of the CUT. The outputs are compared using an XOR gate. The fault is detected if the outputs differ (signal $BD = 1$ in Figure 2.2) for the same inputs, whence the connected inputs and compared outputs. If such an input pattern exists that would satisfy the constrained output ($BD = 1$), then this pattern (or patterns) can test the modeled fault. This is essentially the Circuit Satisfiability Problem (CSAT) which is then transformed into SAT.

The transformation of a circuit to a CNF is straightforward. We only need to express all gates by their characteristic functions. If all the characteristic functions are expressed in CNF, the CNF of the whole circuit is simply a conjunction of these characteristic functions [34].

As was stated before, SAT is an NP-complete problem. Even so, modern solvers can solve instances from the ATPG process fast enough that the bottleneck is in fact in the generation of the SAT instance itself. To the point that techniques targeted at the SAT instance generation speedup are being researched, e.g., the *dynaminc clause activation* [14].

2.2.4 Test Compaction

Practical test generation includes a *test compaction* to reduce the number of test patterns, ideally without losing the fault coverage. These algorithms can be divided into two distinct groups: The *static test compaction*, performed after the test has been generated. And the *dynamic test compaction*, performed during the test generation.

2.2.4.1 Static Test Compaction

The *static test compaction* is performed after obtaining a test set from an ATPG. This compaction can be therefore run independently of the ATPG. The compaction is performed by manipulating the test pattern ordering or simulation ordering, simulating faults, and identifying the non-essential test patterns [49, 28, 70].

2.2.4.2 Dynamic Test Compaction

The *dynamic test compaction* is performed during the ATPG process, on an incomplete test set. It is also beneficial for the ATPG to utilize extra information from the compaction while generating the next test pattern [42, 50, 32, 66, 19, 20].

Usual dynamic compaction techniques use unspecified bits of a test pattern to target additional faults. The compaction achieved by this approach is sensitive to the additional fault selection (fault ordering) and the chosen subset of unspecified bits in the test pattern, both are guided by heuristics. [42, 50].

MTTG *Multiple-Target Test Generation* (MTTG) techniques [66, 32, 19] solve the problem of unspecified bits selection by targeting multiple faults in one step. Modern SAT solvers are robust enough to efficiently find a test pattern for all selected faults or to prove that no such test pattern exists. This approach suffers from the need to efficiently find a subset of faults that can be tested by a single test pattern. In practice, faults are added to the targeted subset incrementally.

OTG In the MTTG, a test pattern detecting all selected faults is searched for. If such a test pattern does not exist, the test pattern is not generated, and a new fault set must be selected. *Optimization-based MTTG* (OTG) techniques [20] do not prove the non-existence of such a test pattern. Instead, they compute a test pattern, which tests some subset of selected faults. The optimization criterion is the number of detected faults.

The algorithm presented in [20] works by constructing a miter for all selected faults. D-chains are also generated for each selected fault. The D-chains are not constrained to

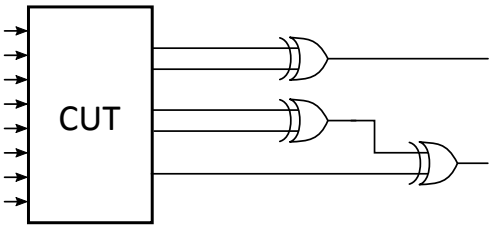

Figure 2.3: Example of a spatial compactor.

detect all faults, but their activation is done by the solver. The number of active D-chains, and thus detected faults, is maximized by the optimization function.

2.3 Output Response Analysis

The *output response analysis* is the process of checking for expected output response during the test. To reduce the amount of transferred data, a *output response compaction* is performed. This compaction can be separated to two steps: *spatial response compaction* and *temporal response compaction*. A brief overview of existing response compaction designs follows.

Note 2.3.1. The *spatial response compaction* and *temporal response compaction* are in some literature known also as *static response compaction* and *dynamic response compaction*. In this thesis, we use the former terms to avoid confusion with the *static test compaction* and the *dynamic test compaction*.

2.3.1 Spatial Response Compaction

Spatial response compaction is a process of reducing the number of observed signals in the circuit while under test, but it does not change the number of functional outputs (Figure 2.3). The main motivations for using spatial response compaction are reducing the response size, the number of testing outputs, and subsequent compaction logic (in temporal compaction) and reducing test application time (by decreasing the scan-out time).

A big issue in response compaction is the danger of aliasing which occurs when the response of a faulty circuit is mapped by the compactor to the same response that belongs to the fault-free circuit. Ideally, we want space compactors that introduce no aliasing at all. To achieve zero aliasing, designs using only XOR gates are often used, for the XOR gate will always propagate a fault syndrome that arrives at one of its inputs [8], [12]. The problem with such a design is that if the fault syndrome arrives even times at XOR tree inputs (even-sensitized faults), the fault is aliased, so only using XOR gates may lead to higher aliasing or lower compaction ratio.

In paper [7] two methods to combat even-sensitized faults are presented. First, it is shown that most faults in benchmark circuits can be odd-sensitized, but there exists a

relatively small amount of faults that are even-sensitized by every test pattern (for the fault).

The first method, the *multiplexed parity trees*, inserts a multiplexer between CUT's PO and the parity tree. The size of the multiplexer and the number of control signals depends on the number and propagation properties of even-sensitized faults in the CUT. It is shown that a fault is rarely sensitized to all POs and a procedure to reduce the number of multiplexers and complexity of its control logic is also presented.

The second method is exploiting the fact that even-sensitized faults are only possible in circuits with branching. By adding an observation point to the CUT, it is possible to change the even-sensitized fault to the odd-sensitized one. Each observation point is then ANDed to a control signal, making the number of control signals and the number of multiplexers dependent on the number of observation points.

By using these two techniques, it is possible to achieve zero-aliasing in a parity tree space compactor.

The paper [8] presents a method for designing linear zero-aliasing space compactors with bounded overhead. The method expresses relationships between PO of the CUT as a graph and the task of partitioning outputs to be combined using a parity tree is then reduced to the graph coloring problem.

In paper [44] the way to construct a zero-aliasing space compactor by using elementary gates is described. This algorithm is using the preexisting test as a guide to combining the circuit's outputs using AND, NAND, OR, and NOR gates. First, responses of all faults are computed using multi-valued logic $\{0, 1, X, D, D'\}$ where D and D' denoted a value changed from 1 to 0 and vice versa due to presence of a fault. Until continuation is not possible, two outputs are selected to be combined with a new gate. If this leads to aliasing, a new test pattern is generated by an ATPG for the aliased fault. When no such test pattern can be found, the pairing with the selected gate is rejected and another gate and/or output pairing is tried. Otherwise, the compactor is augmented by the gate, and output responses for all faults are recalculated. When no other such pairing and gate remain to be tried, the algorithm ends.

Note 2.3.2. The order of output pairings and gates tried is determined by heuristics to minimize the number of ATPG invocations and especially the number of unsuccessful ATPG runs. This hints that the algorithm was intended to be used with structural ATPGs, which may have problems with proving that a fault is redundant.

In paper [35] the algorithm described in the previous paragraph is augmented to not require an ATPG. When all AND, NAND, OR, and NOR gates are exhausted, the XOR (or XNOR) gate is then tried as a last resort. This method provides a similar compaction ratio to the previous one without the need for an ATPG. Both of these methods provide a better compaction ratio with less area overhead than designs using purely XOR gates.

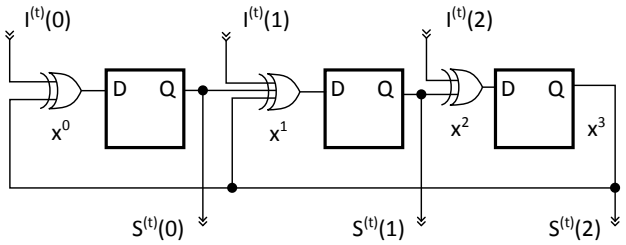

Figure 2.4: Example of LFSR-based MISR with a characteristic polynomial of $x^3 + x + 1$.

2.3.2 Temporal Response Compaction

Temporal response compaction is a process of reducing the number of responses to different test patterns that are applied in different clock cycles. This means using some kind of sequential circuit where we build the circuit *signature* from incoming responses. At the end of testing, we compare the CUT signature with the precomputed one. As in the spatial compaction, here we also want to minimize the aliasing.

There are many architectures of temporal compactors. One of the more prevalent ones is the *Multiple-Input Signature Register* (MISR). In its most common form, it is formed by an LFSR with parallel inputs. An example is shown in Figure 2.4, $input(x)$ denotes inputs from the CUT (or space compactor), $signature(x)$ denotes outputs to the comparator (for signature checking). This particular compactor gets the most attention in aliasing analysis and minimization of.

In paper [46] the *periodic quotient compression* technique is described. When using LFSR as an output response analyzer (ORA), it behaves as a polynomial divider over GF(2). What we call the circuit signature is the reminder after the division, while the quotient is ignored. This method constructs the LFSR in such a way that the *quotient* is periodical, which can then be checked easily with the circuitry of the size linearly bound with the length of the period.

The paper additionally presents proof that for any CUT there exists an LFSR which produces a periodic quotient, for any period. A method for finding such LFSR is presented.

This method achieves zero-aliasing for single-output circuits. It does not however achieves this for multi-output circuits, although it does reduce the amount of aliasing for such circuits.

Another method that utilizes periodic response compaction is described in paper [15]. While usually the test pattern generation and output response compaction are considered separately, this method influences test patterns to achieve zero-aliasing response compaction.

By reordering the test patterns, they achieve a periodic response of the CUT. Such a response can be easily checked in a similar way to the previous method.

Also as in the previous method, zero-aliasing is achieved for single output circuits while for multiple output circuits this is not guaranteed.

In paper [2] a method for response compaction is presented that converts the response to a stream of alternating values. The response of the CUT is applied to the inputs of a MISR, but instead of checking the signature at the end of testing, the contents of one memory element of the MISR are read every clock cycle. This binary steam is however not alternating, so additional combinational circuits are added, the *cover circuits*. The test patterns are applied to these cover circuits and their output is combined with the output of the MISR to produce alternating output. The fault is detected by the disruption of the output value alternation. This method does not achieve zero-aliasing.

In paper [16] the analysis of aliasing probability in *augmented signature testing* is presented. What authors mean by augmented signature testing is nothing else than an LFSR-based MISR with a quotient detector. The quotient detector is a circuit that uniquely identifies the quotient of an LFSR.

In paper [45] a generalized model based on LFSR is described (GLFSR). Under this model, LFSR, MISR, and multiple MISR designs are special cases. A new framework for the analysis of aliasing probability in the GLFSR model is presented.

By using these results, they have shown that some GLFSR compressor exists for all CUTs with arbitrarily small aliasing probability.

In paper [33] a method to estimate the compaction quality of compactors that is based on the entropy of the compactor state is presented. Using this method, LFSR, non-linear cyclic feedback shift registers (CFSR), and counting compactors are compared.

It is shown that a CFSR can be as good as an LFSR with a primitive polynomial. Counting compactors are worse than both CFSR and LFSR. It is also shown that while the number of possible LFSR is 2^n, for the CFSR it is 2^{2^n}, thus expanding the number of possible compacting structures significantly. It is hinted that the quality of non-linear compactors is dependent on the CUT and finding such a compactor is left an open question.

Chapter 3

Encoding Bit-Flip Fault Model in SAT Instance

In this chapter, we present a method to generate a test for application-oriented FPGA testing. This approach has been published in [A.1].

3.1 Introduction

3.1.1 Problem Statement

Field-Programmable Gate Arrays (FPGAs) are typically tested with dedicated test targeting their regular structure [64, 52, 54]. The FPGA chip is tested whole, independently of the target application (the circuit implemented in the FPGA). This approach requires *reconfiguration* of the tested FPGA. It is, therefore, suitable for *manufacture-oriented testing*, but unsuitable for *application-oriented testing*, i.e., testing configurations must be uploaded in place of the application [56].

In the *application-oriented testing*, the circuit implemented in FPGA is tested as if it is custom logic [56, 48, 59, 13]. The FPGA chip itself is not tested whole, only parts used in the uploaded design are tested. The FPGA configuration is not modified for testing [55].

Testing of a circuit implemented in an FPGA requires a specific fault model. It has been shown that the commonly used stuck-at (SA) fault model is insufficient [48, 3]. Particularly, LUT contents, interconnects, and device family-specific features must be tested specifically. Note that in many previously published application-oriented FPGA testing approaches, standard stuck-at or gate-level fault models are used [13, 55, 57].

3.1.2 Method Overview

Since FPGA fabric contains many different device-specific features, fault models used for ASIC testing, such as the stuck-at fault model, are not suitable [48], [3]. Most past and contemporary FPGAs consist of:

1. look-up Tables (LUTs) of different sizes, typically contained in Configurable Logic Blocks (CLBs), together with flip-flops,

2. device-specific primitives, such as fast carry chains (dedicated XOR gates) and multiplexers,

3. interconnects,

4. I/O and other communication blocks, and

5. special complex features, such as block-RAMs, DSP blocks, CPUs, etc.

In this chapter, we focus on the first three types only, as their description can be generalized readily. Moreover, the last two types typically require special approaches to their testing [38, 64, 52, 53, 54, 29]. Also, only combinational circuits will be considered for simplicity; testing of flip-flops or sequential circuits, in general, would require a sequential ATPG process or a special DfT approach [51].

The fault model used is a combination of the stuck-at (SA) and bit-flip (BF) fault models. The BF models a single changed bit in the configuration memory of LUTs. The configuration memory of interconnects is not considered for simplicity, as interconnects design and its configuration are device-specific. Instead, the SA model is used for testing interconnects. Single faults are assumed in this chapter, however, the model can be readily extended to support multiple faults. For this fault model, an SAT-based ATPG is presented and its properties are discussed.

3.2 Application-Oriented FPGA Testing

3.2.1 The Overall Algorithm (ATPG)

The SAT-based ATPG works by duplicating a part of a circuit and modeling the fault in the duplicated part [34]. For the typical stuck-at fault model, the faulty signal is duplicated and forced to have the stuck-at value. The rest of the circuit that is dependent on this signal (output cone) is also duplicated. The output of this duplicated circuit is then XORed with the original circuit and the Circuit Satisfiability Problem (CSAT) is then solved by reduction to an SAT instance, which is then solved by an SAT solver [17, 34]. Any satisfying variable assignment then represents a test vector. In the case where no such variable assignment exists, the fault is identified as redundant.

In addition to stuck-at faults, we are also considering single bit-flips in LUTs [41, 3]. We model these faults by duplicating a given LUT and injecting a fault in it, by flipping one bit in its memory. Then we duplicate the output cone of the affected LUT in the same way as with the stuck-at faults.

3.2.2 The Proposed Combined Fault Model

As denoted in the Introduction, we aim at application-oriented testing. Thus, only the implemented circuit is tested, disregarding the unused parts of the chip. This scenario allows testing the interconnect using a standard stuck-at model, in contrast to transistor-oriented FPGA interconnection testing, where switching matrices that are known only after the place&route phase, must be considered [38, 64]. Simple gates can also be tested using the stuck-at model. However, the stuck-at model is not sufficient for LUT testing [48, 3]. Therefore, a *single bit-flip* fault model is used for this purpose. This basically complies with the idea of [48], where stuck-at faults in all LUT cells were considered. However, one-half of these faults were found redundant and had to be explicitly removed from the fault list. The bit-flip fault model directly eliminates this problem.

As shown in [56, 3], the initial circuit description (non-mapped netlist) is not suitable for ATPG purposes. However, the logic of the mapped circuit can be easily obtained from commercial FPGA synthesis tools [57]. As a result, the *mapped* netlist can be described as a multi-level Boolean network, where nodes are described in a Sum of Products (SOP) form. This network can then be directly used for test generation.

In our algorithm, we describe the mapped logic by a network of *general nodes*. By general nodes we understand arbitrary single-output functions; however, the approach can be easily extended for multi-output functions too (for the case of contemporary 2-output LUTs, for example).

For implementation purposes, the BLIF format [68], where each node is described as a sum-of-products (SOP), is well-suitable. Essentially, two types of nodes can be present in the mapped netlist:

1. a k-input LUT node, whose function (LUT content) can be described by an SOP,
2. any other simple function (XOR, MUX, etc.) that can be described in an SOP form as well.

Summarised, the proposed fault model derived from the multi-level network of SOP nodes consist of:

1. all single bit-flips in the LUT, i.e., 2^k faults for a k-input node,
2. single stuck-at-0 and stuck-at-1 at all inputs and outputs of each node.

3.2.3 Generating Test for Different Fault Models

While the general ATPG algorithm remains the same, one step in particular differs across the fault models – the generation of a miter.

The difference is in how we model the fault itself. A stuck-at fault is modeled by disconnecting the faulty signal from the fault-free circuit and setting its value using a unit clause [34]. When modeling a bit-flip fault, there is no discontinuity in the faulty circuit. Instead, we model this fault by flipping the output for a particular LUT input vector.

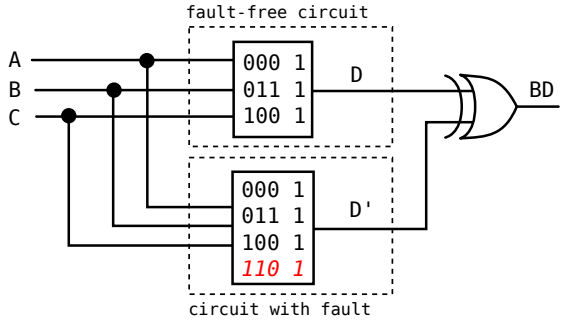

Figure 3.1: Example of miter with a bit-flip fault. The output of the circuit must be 1 to detect the fault. Bit-flip is distinguished by italics

$$F(a,b,c) = \{0,2,3,6,7\}$$

on-set

a	b	c	y	
0	-	0	1	$(a \lor c \lor y)$
-	1	-	1	$(\neg b \lor y)$

off-set

a	b	c	y	
1	0	-	0	$(\neg a \lor b \lor \neg y)$
-	0	1	0	$(b \lor \neg c \lor \neg y)$

CNF

$$(a \lor c \lor y) \land (\neg b \lor y) \land (\neg a \lor b \lor \neg y) \land (b \lor \neg c \lor \neg y)$$

Figure 3.2: An example function defined by a list of 1-minterms, its minimized on-set, off-set, and their respective CNF clauses

For example, let us have a LUT described by 1-minterms $\{000, 011, 100\}$ and a bit-flip at address `110` which can be described by adding a minterm $\{110\}$, see Figure 3.1.

3.2.4 Characteristic Function of LUT

The basic idea behind this method is an algebraic transformation of the characteristic function of a node to CNF. We start with the equivalence of the output and the function of the node. Then we proceed by applying Boolean algebra to transform this formula into CNF [34].

In practice, this is done by finding a representation of the node function (on-set) and its complement (off-set) in an SOP form. These SOPs are implicating the function output, one or zero, respectively. By using DeMorgan's rules, we transform these implications to products-of-sums (POS) ORed with the output. Then we simply distribute the output

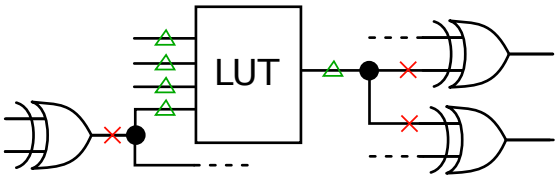

Figure 3.3: Dominance of different faults. Triangle denotes a SA fault dominated by a bit-flip fault. Cross denotes a SA fault that may not be dominated.

inside the POS forms. This leaves us with two CNFs that together form a characteristic function of the node as can be seen in Figure 3.2. Assuming that the nodes are simple functions with a small number of inputs, this approach is computationally feasible. However, this method can produce up to 2^k clauses of a length up to $k+1$ literals, where k is the number of inputs of the node.

3.2.5 Dominance between Stuck-at and Bit-Flip Faults

It can be shown that in a circuit with prevalent LUT nodes all testable stuck-at faults that are located at LUT inputs or at the LUT output, are dominated by some bit-flip faults.

A stuck-at fault located at the output of a LUT may be not covered by a bit-flip in the LUT, if and only if there is no such bit-flip that would change the value of this signal in the same way as the stuck-at fault (for any input vector). However, such LUT would have a constant output, independent of its input, thus it would be redundant and so would be the stuck-at fault.

For stuck-at faults that are located at input signals of a LUT to be not covered by some bit-flip would mean that there exists no logical assignment of remaining LUT inputs that would cause observable change at the LUT output for different values of the faulty signal. That would mean that the output of the circuit is independent of the signal value and thus the stuck-at fault would be redundant.

There is, however, no guarantee of dominance for stuck-at faults that are not directly adjacent to any LUT, be it at signals between two non-LUT elements or at signals after or before branching. Examples of such stuck-at faults are shown in Figure 3.3.

3.3 Experimental Results

For our experiments, we used 279 circuits from benchmarks MCNC, LGSynth'91 [71], LGSynth'93 [37], ISCAS'85 [5], ISCAS'89 [4] and IWLS 2005 [1]. For sequential circuits, combinational parts were extracted. The circuits were then synthesized by Xilinx Vivado 2015.2, for the Artix-7 architecture. After the synthesis step, we extracted the circuit structure from the EDIF format to BLIF [68].

We have measured the ratio of stuck-at faults that are covered by bit-flip faults, and the ratio of bit-flip faults that are covered by stuck-at faults. Results for several selected

Table 3.1: Redundant faults and coverage of non-redundant faults in selected circuits

	stuck-at				bit-flip				total	
	faults redundant		testable coverage		faults redundant		testable coverage		faults redundant	
circuit	#	[%]	#	[%]		[%]		[%]	#	[%]
alu4	2484	0.85	2463	100	9060	17.27	7495	68	11544	13.7
barrel16a	1246	0	1246	91.1	3208	1.5	3160	90.8	4454	1.08
cm42a	136	0	136	100	160	0	160	100	296	0
cmb	162	0.62	161	100	342	13.16	297	31	504	9.13
dalu	2492	6.86	2321	100	8056	35.74	5177	87.4	10548	28.9
des	6626	0.09	6620	100	15960	20.4	12704	95.6	22586	14.4
dsip	8960	0	8960	100	17944	0	17944	87.5	26904	0
mux	118	0	118	100	320	0	320	23.1	438	0
s9234	5098	0.53	5071	98.9	10584	17.01	8784	82.2	15682	0
average		0.33		99.6		10.15		69.5		7.42

circuits can be seen in Table 3.1; average values obtained from all the tested circuits are shown in the last row.

We have found that almost all stuck-at faults are dominated by bit-flip faults. An example of a circuit, where there are stuck-at faults that are not dominated by a bit-flip fault, is circuit `barrel16a`, which contains primitives, such as multiplexers.

For bit-flip faults, we have observed a dominance by stuck-at faults ranging from 17% to 100%, with an average of 69.4% and a median of 72.6%.

We have found a surprisingly large set of redundant faults; the observed average ratio of redundant faults was 0.33% for stuck-at faults, 10.15% for bit-flip faults and 7.42% for all faults, as can be seen in Table 3.1. Such a high amount of redundant bit-flip faults is due to a smaller controllability and observability of these faults, i.e., the probability that such test vector exists, so all inputs of a given LUT are set to excite the fault *and* it is propagated to a primary output. Note that propagation to output is equivalent to stuck-at faults propagation, but excitation needs more signals to be set to a specific value.

We have also examined how the final test lengths differ, for the two fault models and for different orderings of faults. Results of measurements for few selected circuits can be seen in Table 3.2. We have found that the ordering of faults has a small impact on the number of testing vectors. When we look at the ratio of these two orderings, we see that it ranges from 0.81 up to 1.11, with an average of 0.97 and a standard deviation of 0.04. This means that there is no significant difference between these two orderings; the difference is just due to the algorithmic noise [63].

3.4 Conclusions

In this chapter, we have presented our SAT-based ATPG with two fault models, the *stuck-at* and the *bit-flip* models. Compared to previous methods, our ATPG works natively

Table 3.2: Length of test for different fault ordering

circuit	stuck-at	bit-flip	SA,BF	BF,SA	$\frac{\text{SA,BF}}{\text{BF,SA}}$
alu4	413	2844	2866	2871	0.998
barrel16a	112	766	745	804	0.927
cm42a	16	16	16	16	1
cmb	24	227	218	228	0.956
dalu	158	1181	1165	1190	0.979
des	296	2232	1944	2214	0.878
dsip	313	4190	4052	4212	0.962
mux	33	306	297	306	0.971
s9234	375	2649	2618	2647	0.989
average					0.971

with both SA and BF faults. A combination of these models is suitable for application-oriented FPGA testing. The suitability of this fault model is further examined in [A.4], which is not part of this thesis. Also, in the scope of the thesis, the development of this ATPG algorithm served to prepare necessary tools for further research, namely the ZATPG algorithm presented in the next chapter.

We have examined the two fault models in the context of application-oriented FPGA testing and their interaction with respect to their mutual dominance. We have found that most stuck-at faults are dominated by bit-flip faults. The cases where stuck-at faults are not dominated include FPGA primitives, such as multiplexers. Bit-flip faults, on the other hand, are generally dominated by stuck-at faults to a much smaller degree, ranging from as low as 17%.

We conclude that for complete coverage of these two fault models, both must be considered. However, most stuck-at faults are dominated by bit-flip faults. These are easily identifiable from the circuit structure, as they are adjacent to LUTs and their dominance is independent of the circuit function. They constitute a majority of stuck-at faults, thus their omission may lead to a significant ATPG speed-up. To reach a complete fault coverage, structurally not dominated stuck-at faults must be considered too since some of them are not covered by bit-flip faults (they are not dominated functionally).

We have also examined the overall lengths of tests generated with the fault-dropping technique for two orderings of faults and have found no significant impact of fault ordering on the number of generated test vectors.

Chapter 4

Augmented ATPG with Zero-Aliasing Constraints

In this chapter, we present our method to prevent aliasing in the output response analyzer. The method works by augmenting the ATPG process itself, i.e., the aliasing is prevented during a test pattern generation. This approach has been published in [A.2] and [A.3].

4.1 Introduction

4.1.1 Problem Statement

When generating a test for a digital circuit, there are several important factors to consider. For the sake of clarity, we focus mainly on the fault coverage, the test length, and the size of the testing logic. Additional important factors are test application time and the volume of transferred data.

The test application time can be approximated by the test length. For the purpose of this thesis, they are effectively equivalent. The volume of data transferred during the test application is considered only for the direction from the CUT to a tester (ATE, BIST...), again for the sake of clarity. The data that is transferred is in the simplest case a single signature from a signature register. The opposite direction of data transfer is discussed briefly in the discussion but is otherwise left for future work.

4.1.2 Method Overview

Conventional SAT-based ATPG searches for a test pattern without considering further processing of its output response. The only requirement is that the tested fault is detected at combinational outputs (POs and PPOs) of the CUT. Each test pattern and fault that are detected are treated on their own. There is no notion of test ordering or response compaction in the ATPG process. The problem of possible aliasing in compactors is treated separately, with an already generated test.

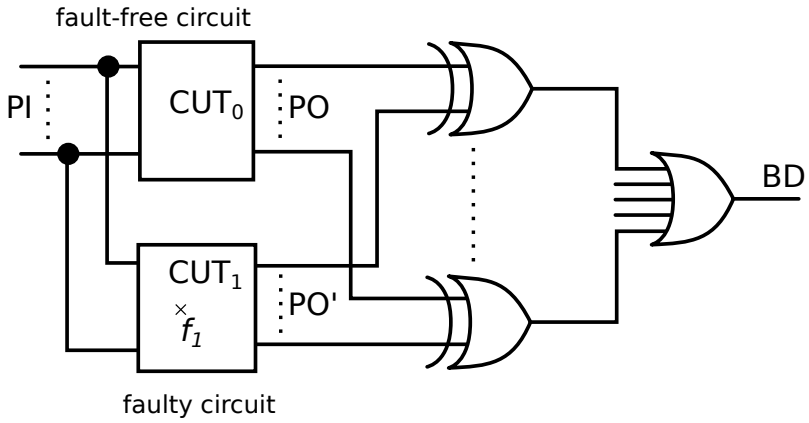

Figure 4.1: Conceptual circuit (miter) for an SAT-based ATPG.

Approaches based on altering (or re-running) the ATPG to reduce aliasing in the *spatial* compactor have been proposed, e.g., [9, 44]. However, up to our knowledge, no method of generating aliasing-free test patterns for *temporal* compactor has been proposed prior to our papers [A.2, A.3].

The focus of this chapter is, therefore, on the aliasing in *temporal compactors* and its prevention. Aliasing in *spatial compactors* is not considered, it is assumed that no new untestable faults are introduced; the spatial compactor is irredundant. Instead, the spatial compactor is considered a part of the CUT. This has the advantage that all irredundant faults in the spatial compactor are *explicitly* tested. For the remainder of this chapter, the spatial compactor won't be discussed again unless necessary.

In our approach, we constrain the ATPG process itself to generate test patterns for the CUT, including the temporal compactor. Thus, no test patterns recomputation is needed; test patterns exhibiting no aliasing are produced directly. The output of the ATPG, the test, is a *sequence* of test patterns for *given* temporal compactor. Should a different compactor be used or the test be manipulated (e.g. reordered), the zero-aliasing property of the test is no longer valid.

4.2 Augmenting the ATPG

We build on the idea of extending the conceptual circuit (miter) from Figure 4.1 [30, 60]. The extended miter (Figure 4.2) introduces additional circuitry that computes fault aliasing.

Let us consider the search for a suitable test pattern p_i, applied to PI for the fault f_1. Our miter initially corresponds to Figure 4.1, ensuring that the response at PO' to a test pattern differs from the response at PO of the fault-free circuit CUT_0.

In the context of temporal compaction, we know that in the presence of the fault f_1, the internal state of the compactor (*partial signature*) will differ from the partial signature of the fault-free circuit. This is because the fault f_1 was newly tested by the test pattern

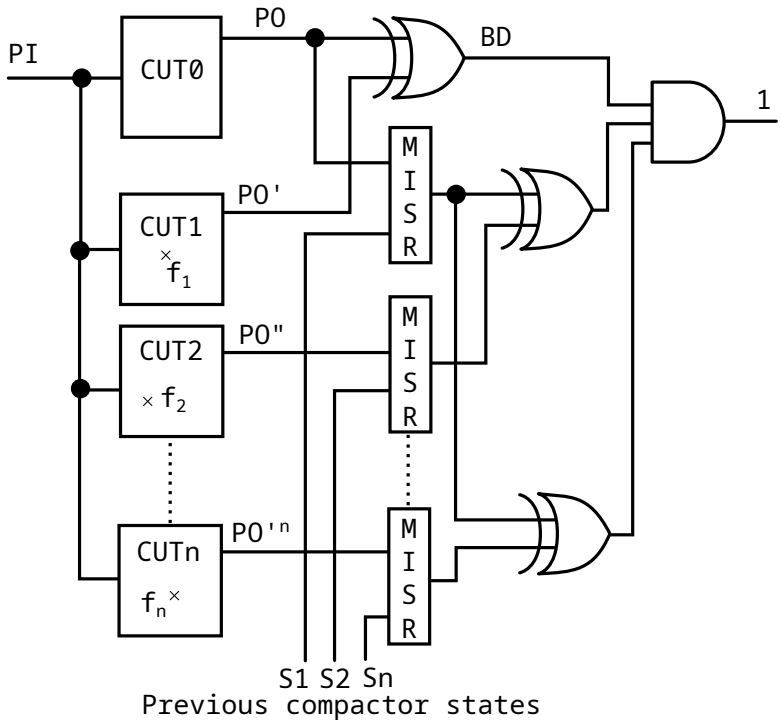

Figure 4.2: Extended miter for finding non-aliasing test vector p for the fault f_1.

p_i.

This assumption, however, needs not to hold for other faults that were tested by some previously generated test pattern(s). If we know of such faults, f_2–f_n, and they additionally have a potential to be aliased by p_i, i.e., after application of p_i, the partial signature in the MISR could be the same as the partial signature for the fault-free circuit (for details on their identification see Subsection 4.5.1). It means that although some fault (e.g., f_2) has been detected by a previously generated test pattern, the MISR returns to the sequence indicating no fault. To prevent this, we must ensure that the next state of the MISR (the next partial signature) will stay different from the fault-free case when any such fault is present. We can include anti-aliasing constraints in form of additional circuitry in the miter (Figure 4.2).

The next partial signature is computed by applying the transition function of the MISR (its combinational part, T_{MISR}) to the present partial signature and the circuit response. In the fault-free case, let us call the present partial signature S_1. In the case of a fault f_2 producing a response PO'', let us call the partial signature S_2. To avoid the above-mentioned aliasing, all next states caused by the faults f_1–f_n must differ from that of the fault-free case. The extended miter (Figure 4.2) compares the results of the MISR transition function applied to $T_{MISR}(S_1, PO)$ and $T_{MISR}(S_2, PO'')$, respectively. Hence, the extension of the miter consists of additional replicas of the CUT (CUT_2–CUT_n), each with one fault (f_2–f_n) modeled. Their output responses (PO''–PO'^n) are inputs for the transition function of the compactor (block "MISR" in Figure 4.2, see Subsection 4.3 for

details). The values of partial signatures S_j can be obtained by sequential simulation, or by extracting signal values from the SAT instance solution.

All constraints, as well as the Boolean difference of the original miter, must be applied simultaneously when generating a test pattern. That is expressed by the last AND gate of the miter, which can be implemented as a conjunction of clauses in the CNF instance (since the AND output is forced to 1).

By including multiple copies of the CUT, we increase the number of clauses and variables significantly (i.e., the size of the SAT instance), especially if a high number of anti-aliasing constraints is in effect. However, it has been shown on a problem of multi-targeted faults that SAT solvers are robust enough for this approach to be feasible [19]. We expect that the SAT solver will be efficient in finding the constrained test pattern or proving that no such pattern exists and aliasing is unavoidable (aliasing of some fault f_j in the current step, by test pattern p_i).

The necessary assumption of sequential generation of test vectors, i.e., test vectors p_1 – p_{i-1} must have been generated *prior to* p_i, leads to an unfortunate consequence that the generated test vectors cannot be further rearranged or compacted after the test was generated, as this would invalidate the guarantee of zero aliasing. However, it has been shown [23] that much more compact, i.e., shorter test can be obtained when fault dropping is not used in the main ATPG loop; in standard ATPG practice [23], highly redundant test is generated first, and then it is compacted to reduce its size. This is unfortunately not possible in our case. Consequently, longer tests are expected.

The preliminary experiments have shown that the test length also depends on the fault ordering used. Therefore, we use the fault ordering based on *accidental detection index* (ADI) [43]. The experimental results for different fault orderings are presented in Section 4.6.

4.3 Unrolling the Sequential Circuit

The aliasing happens *after* application of a test pattern that is being generated. That means that we need to know the future state of the temporal compactor *during* test pattern generation. Simulation of the compactor is, however, possible only after a test pattern was generated. To avoid this problem, the transition function T_{MIRS} can be encoded in the miter and the SAT instance by *unrolling* the compactor.

The method of unrolling can be used to express the future state of the compactor during the test pattern generation, provided we know the *previous* state of the compactor. Then, the future state can be computed by extracting the combinational part of the compactor. Simulating the compactor is then part of SAT-solving during the search for a test pattern – the compactor is included in the miter for which SAT is solved. An example of an unrolled compactor from Figure 2.4 is depicted in Figure 4.3.

In the miter in Figure 4.2, this unrolled circuit is shown as MISR blocks, one for each anti-aliased fault and one for the fault-free circuit (note that these are combinational

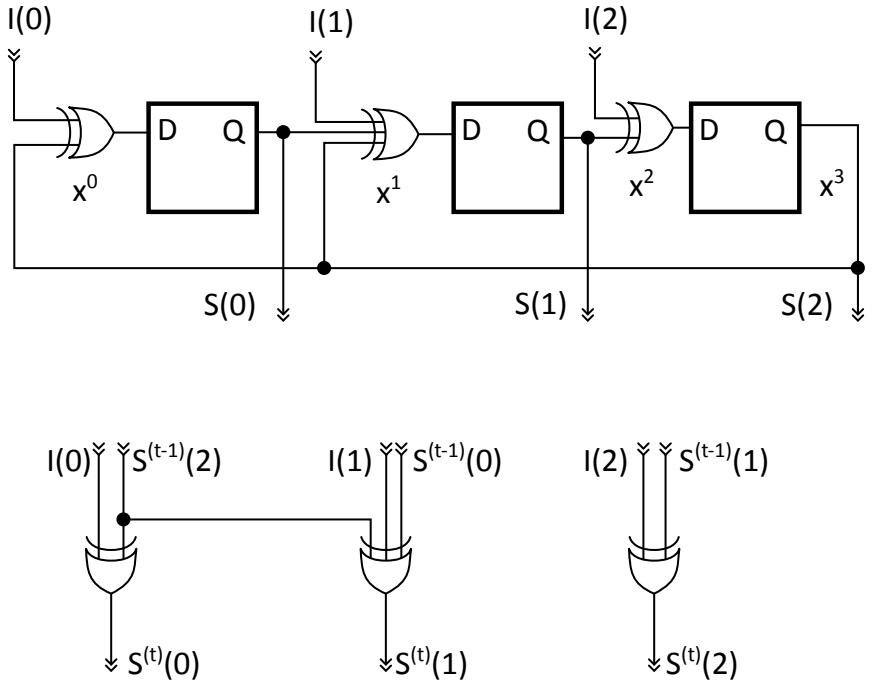

Figure 4.3: Example of a temporal compactor and corresponding unrolled circuit

circuits, not actual MISRs). The previous internal state of the compactor is represented by vectors S_1 and S_2–S_n for the fault-free and faulty circuits respectively.

This extended miter can be transformed to the CNF the same way as in the conventional SAT-ATPG. Due to the nature of CNF, it is easy to generate the CNF of the extended miter. For simplicity, we can use the CNF from the transformation of the classical miter and add new parts of the extended circuit by simply concatenating new clauses to the CNF. This includes clauses describing the circuits CUT_2–CUT_n, the unrolled compactor, and constraints forcing the outputs to be non-zero vectors. In Figure 4.2, this concatenation is represented by the AND gate.

4.4 Simplification for Linear Compactors

For linear temporal compactors, the *superposition principle* can be used to separate their state and input into an *error component* and a *correct component*. These components can be then used independently.

The output response coming from the CUT can be divided into two parts, $R = R_c \oplus R_e$, where R is the response of the CUT, R_c is the response of the fault-free circuit, and R_e is the *error difference* of the output.

For illustration of this, we have chosen the LFSR-based MISR compactor from Figure 2.4. If we use the superposition principle for the MISR, we can split the internal state into two states, $S = S_c \oplus S_e$, where S is the state of the MISR, S_c is the state for the fault-free circuit, and S_e is the *error difference* (error component) for the faulty circuit.

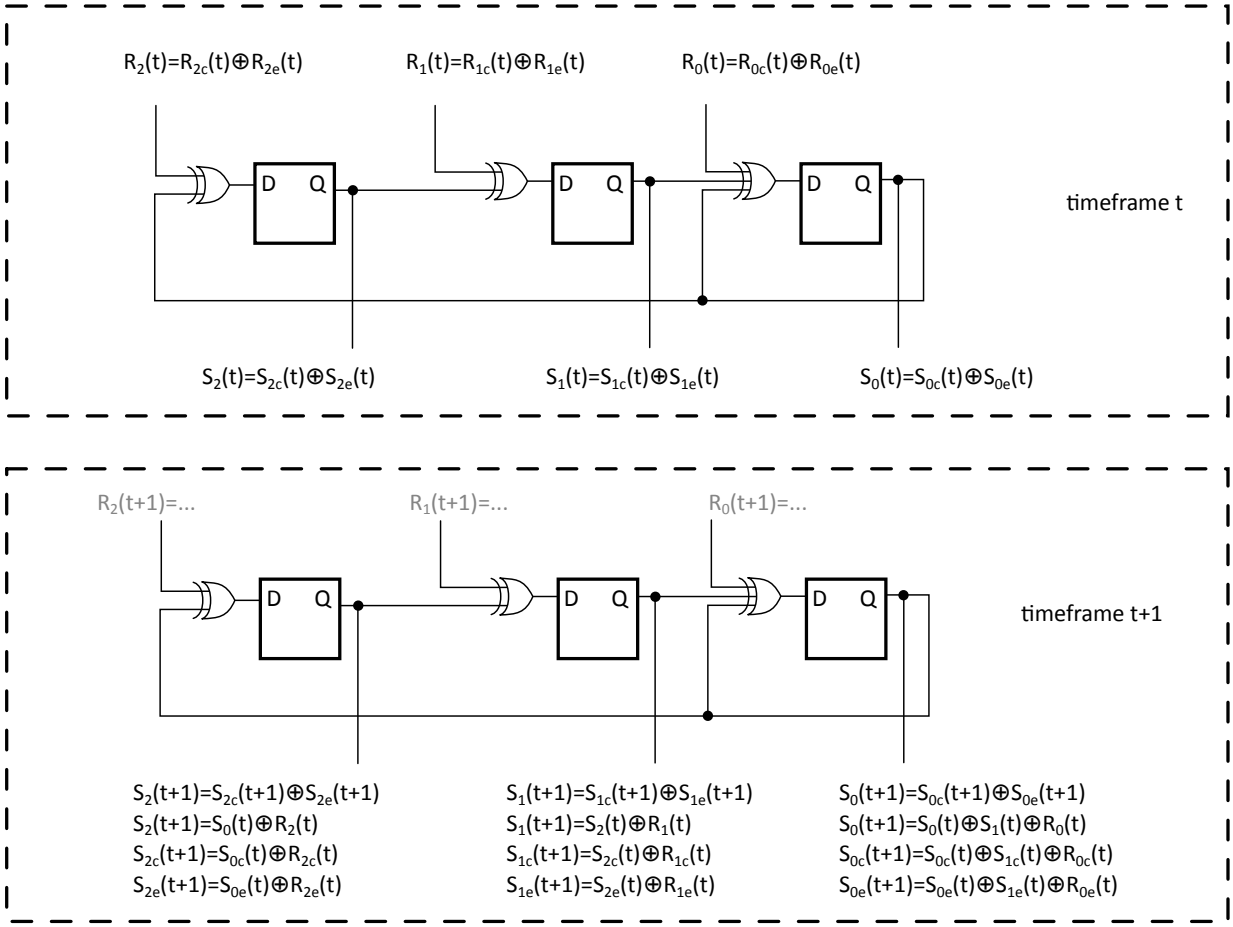

Figure 4.4: Example of the state computation of a linear compactor.

The next partial signature is computed as $S^{(t+1)} = T_{MISR}(S^{(t)}, R^{(t)})$. For example, the MISR from Figure 2.4 has a transition function $T_{MISR}((s_0, s_1, s_2), (r_0, r_1, r_2)) = (s_2 \oplus r_0, s_0 \oplus s_2 \oplus r_1, s_1 \oplus r_2)$.

Due to the linearity of T_{MISR}, where $T_{MISR}(a \oplus b) = T_{MISR}(a) \oplus T_{MISR}(b)$, we can compute the *correct* and the *error* states separately as $S_c^{(t+1)} = T_{MISR}(S_c^{(t)}), R_c^{(t)})$ and $S_e^{(t+1)} = T_{MISR}(S_e^{(t)}), R_e^{(t)})$, respectively. This is illustrated in Figure 4.4.

To compute the *signature* of the CUT, we only need to compute the $S_c^{(l)}$, where l is the length of the test. Similarly, to analyze the *aliasing*, we only need to consider $S_e^{(l)}$, where a non-zero vector means that a fault was detected and the zero vector means that it was not detected.

To prevent aliasing for *all* single faults after application of the whole test, we need to prevent aliasing after application of *every* test pattern for *every* single fault. Note that aliasing for a fault f_s can happen only if this fault was previously detected. Let's say that, after application of t-th test pattern (in test step t), the *error* component of the partial signature $S_e^{(t)}$ is non-zero, meaning that a fault (for example f_s) was detected. In step

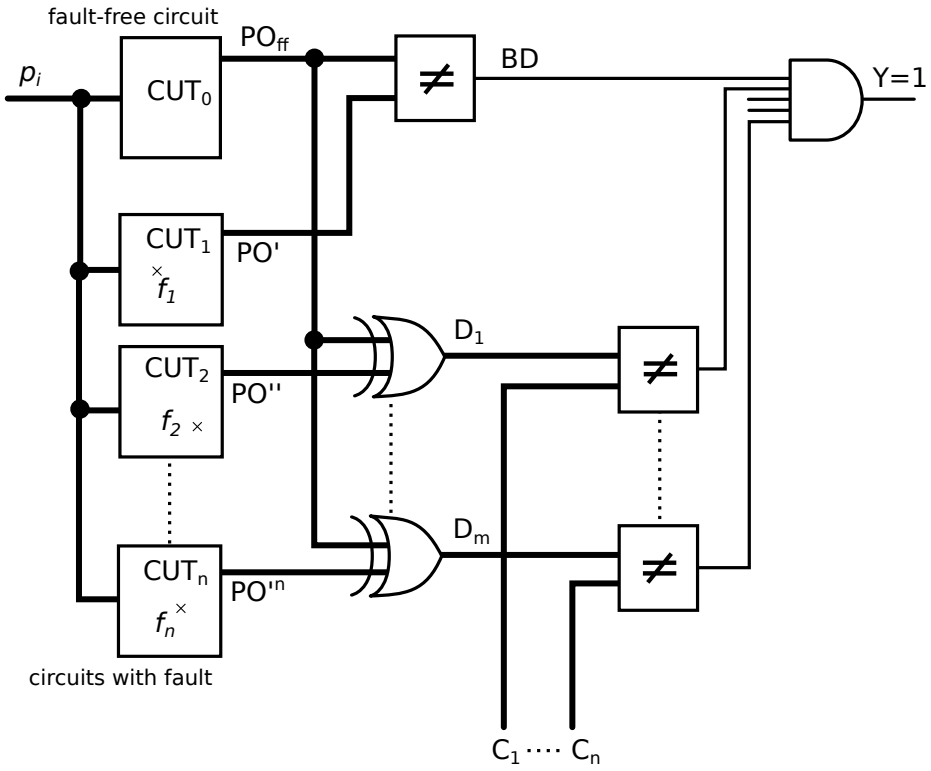

Figure 4.5: Miter for finding zero-aliasing test pattern, simplified for linear compactors.

$t+1$, we apply the next test pattern testing some other fault. If we then get $S_e^{(t+1)}$ equal to zero (partial signature equal to the signature of the fault-free circuit), an aliasing event occurred.

We can express the output response that would cause aliasing as $R_e^{(t)} = T_{MISR}(S_e^{(t)}, 0)$. This can be done in compactors, where the response $R^{(t)}$ is directly XORed with the internal state $S^{(t)}$. We can thus precompute the *error difference* of the response causing the aliasing after application of the next test pattern.

By using the compactor with the *error* state in the simulation step, we can omit the unrolled combinational part of the compactor. The simplified miter can be seen in Figure 4.5. Here, the vectors C_1 through C_n represent constraints of the *error component* of the response. These output vectors are constrained to not have the value that would cause aliasing. This is done by solving an equation $T_{MISR}(S_e^{(t)}, R_e^{(t)}) = 0$.

4.5 Algorithm

The main execution loop of the proposed ZATPG procedure expressed in pseudocode is shown in Algorithm 4.1. The algorithm is initialized by setting the list of uncovered faults F_u to all non-redundant faults (line 3). After the initialization, the algorithm works by repeatedly (lines 4–24) iterating over the list of uncovered faults F_u (lines 6–23) until either

Algorithm 4.1 Overview of the ZATPG algorithm.

```
1: procedure ZATPG(M)
2:     P ← ( )
3:     F_u ← F
4:     repeat
5: fault loop:
6:         for all f ∈ F do
7:             if f ∉ F_u then
8:                 continue
9:             end if
10:            C ← {}
11:            p ← ATPG(f, C)
12:            while a p was generated do
13:                F_a, F_d ← SIM(P, p)
14:                if (|F_a| ≤ M) ∧ (|F_a| < |F_d|) then
15:                    P ← append(P, p)
16:                    F_u ← F_u \ F_d
17:                    F_u ← F_u ∪ F_a
18:                    continue fault loop
19:                end if
20:                C ← C ∪ CONSTR(P, F_a)
21:                p ← ATPG(f, C)
22:            end while
23:        end for
24:    until P was not updated in last iteration
25:    return P
26: end procedure
```

a complete fault coverage is achieved or no test pattern with zero aliasing can be found anymore.

The requirement of preventing aliasing for *all* faults in *every* step is too strong. The algorithm stops prematurely when no test pattern without aliasing can be found. To alleviate this problem, we introduce relaxation: some small amount of aliasing is allowed to occur, as we are able to *re-detect* the aliased fault by the end of the test. For this purpose, we introduce an adjustable parameter M, and the generated test pattern is accepted even with aliasing if this aliasing is lower than M. An additional requirement is that the number of newly (re-)detected faults is higher than the number of aliased faults, i.e., the fault coverage is improved.

During the iteration, first, a test pattern p is generated as in usual ATPG with no additional constraints (lines 10, 11). The aliasing caused by this pattern is then analyzed by simulation for *all* faults (line 13). If the number of aliased faults $|F_a|$ is lower than the number of newly detected faults $|F_d|$ and $|F_a|$ is not higher than the parameter M value,

the pattern p is accepted.

In the case of the above-mentioned condition for accepting the pattern p not being met, the set of aliased faults F_a is fault-simulated and a set of constraints C is constructed (line 20) and a new pattern p is generated with these additional constraints. This is continued until the pattern acceptance condition is met or no pattern exists that would satisfy all the constraints.

When the test pattern p is accepted, it is appended at the end of the test patterns sequence P (line 15). Additionally, detected faults F_d are removed from the list of undetected faults F_u (line 16). Conversely, aliased faults F_a are added back to the list of undetected faults F_u (line 17).

The algorithm stops when no new pattern was generated for all remaining uncovered faults (line 24).

Our implementation also contains optional fault ordering randomization, either at the beginning of the algorithm or in every step of the *fault loop* (6). If not used, fault ordering is preserved as the fault list is generated or provided from outside.

The algorithm uses the following procedures:

4.5.1 SIM

The procedure *SIM* runs a sequential simulation of the partial test sequence for every fault. It simulates the compactor's *error state* after application of a partial test sequence P, including the newly generated test pattern p. Here, the *error state* is the difference between the states of the fault-free and faulty circuit, as described in Subsection 4.4.

As an implementation speed up, we need not simulate the entire (partial) test sequence. Instead, we save the resulting compactor state of the simulation for each fault and simulate the newly generated test pattern only. The saved state is updated when the test pattern is accepted and appended to the test sequence.

Faults that are newly detected by p and those that are aliased, are identified. New detection of a fault is indicated by changing the *error state* of the fault from the zero vector to a non-zero vector after application of p. Conversely, the aliasing of a fault is detected by changing its *error state* from a non-zero vector to a zero vector. The sets F_a and F_d of aliased and detected faults are then returned.

4.5.2 ATPG

The procedure *ATPG* generates a test pattern p_i for a fault f and constraints p_i to not cause aliasing, according to information given in C. C is a set of pairs of a potentially aliased fault f_j and the response (vector) $R_{e,j}$ to f_j that would cause the aliasing. The miter (Figure 4.5) is extended for the fault f_j by adding a copy of the CUT, CUT_j. The response vector from C is then used to constrain the value C_j in the miter. In practice, a blocking clause is added to signal D_j.

The generated test pattern p_i, if any, is then returned.

4.5.3 CONSTR

The procedure *CONSTR* computes constraints for the ATPG to prevent the aliasing of faults selected in fault simulation (procedure *SIM*). This is done as described in Subsection 4.4.

For example, to prevent aliasing of a fault f_j, the internal state of the MISR is simulated using only error differences of output responses to all test patterns generated prior to the current test pattern p_i (which is not computed yet). Then, the final transition of the MISR is simulated with zero input vector, corresponding to the output response where fault f_j was not detected. The output response (for f_j) that would cause aliasing is then inferred from the internal state of the MISR.

This response (error difference) is then forwarded together with the fault F_j to the *ATPG* procedure, where the constraints are incorporated into the miter.

4.6 ZATPG: Experimental Results

4.6.1 Experimental setup

In our experiments, we are using benchmark circuits from ISCAS'85 [6] and some circuits from ITC'99 [11].

For each circuit, we consider several sizes w of the temporal compactor. Since the CA-based MISRs are a promising alternative to LFSR-based MISRs, because of their proclaimed lower aliasing [27], we have studied their efficiency in connection with ZATPG. Our initial assumption is that CA-based MISRs could perform better, in sense of aliasing. Experiments were conducted to prove or disprove this theory.

We use and compare different linear compactors, an LFSR-based MISR and cellular automata (CA) based MISR. The selected LFSR-based MISR always has a primitive characteristic polynomial. CA-based MISRs with rules 90, 150, or a hybrid 90/150 rule were used.

As a spatial compactor, we are using a parity tree forest (disjoint XOR trees) design. The number of spatial compactor outputs matches the width of the MISR (w). Thus, the spatial compactor is constructed randomly as w disjoint XOR trees, while its irredundancy is guaranteed by simulation.

For each circuit, we append a spatial compactor to CUT primary outputs and process the resulting circuit as a whole, i.e., the spatial compactor does not affect ATPG workflow once it has been generated. Therefore, the additional faults in the compactor are also targeted by the ATPG. All faults in the compactor are testable under the assumption that faults at the CUT POs are testable, i.e., that each PO can be set to both logical values 1 and 0.

We only consider faults that are testable, meaning we do not include redundant faults in any of our aliasing or coverage percentages. The coverage of 100% can thus be achieved for every circuit. Redundant faults are detected prior to the ZATPG run, by executing a standard SAT-based ATPG process.

Table 4.1: Test length for LFSR-based MISR of 9 bits

circuit	orig	0decr	decr	incr	incr0
b04	151	**117**	135	150	168
b11	131	**117**	**117**	132	135
c1355	**91**	96	97	118	122
c1908	146	136	**133**	168	167
c2670	253	285	274	262	**244**
c499	65	66	**63**	81	83
c5315	274	**221**	232	256	259
c7552	**258**	300	265	307	302
c880	**81**	82	86	100	113
total	1450	1420	**1402**	1574	1593

For both ZATPG and conventional ATPG, we use the SAT-based ATPG ([A.1], Chapter 3) which uses MiniSAT [17] as an SAT-solver.

4.6.2 Fault Ordering

To determine the most promising fault ordering, we first need to have some metric by which to compare. As we are generating a test with zero aliasing, or complete fault coverage, measuring aliasing/coverage is a clear candidate for consideration.

We have found from our experiments that the fault ordering has no or insignificant effect on fault coverage. No trend, as to which ordering achieves better coverage, was observed. As the measurements are inconclusive, we do not include them here.

When we look at the length of the test generated, we find a different picture. For this measurement, we have chosen one size of the LFSR-MISR, $w = 9$ bits. This is the size of the LFSR-MISR, where we achieve complete coverage for all measured circuits.

The measured test lengths for different fault orderings are shown in Table 4.1. The orderings used are *orig* which is simply the unchanged fault ordering from the circuit description, *decr* which has faults sorted according to ADI in decreasing order, *0decr* which treats faults with the ADI of 0 as a special value and places them at the front of the fault-list, *incr* and *incr0* which place faults in increasing order.

The fault orderings *decr* and *0decr* lead to the smallest test size overall while the orderings *incr* and *incr0* lead to longest tests. This result is in agreement with results from [43]. Therefore, we use the *decr* ordering for all subsequent experiments.

4.6.3 Fault coverage

To decide the performance of the ZATPG, we measured fault coverage for different compactors. Results for LFSR-based MISR and CA-based MISR with the hybrid rule 90/150 (CA90150-MISR) are shown in Tables 4.2 and 4.3, respectively. Other investigated compactors, the CA with rule 90 and the rule 150 are not shown, as their performance was worse

Table 4.2: Fault coverage for conventional and augmented ATPG (LFSR-MISR)

MISR size		2		3		4		5	
circuit	faults	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]
b04	2846	80.36	79.02	87.62	90.15	95.71	96.06	97.96	99.12
b11	2382	80.94	80.48	89.58	90.13	93.52	94.95	97.39	99.49
c499	970	82.37	78.35	89.98	88.33	97.2	94.72	96.16	97.2
c880	1582	77.43	81.67	90.32	92.59	95.12	97.66	96.76	100.
c1355	2618	78.19	73.49	88.72	92.05	95.87	95.83	97.21	96.21
c1908	2581	85.08	76.09	93.33	86.51	96.7	92.94	98.8	94.56
c2670	3613	75.64	73.01	91.58	87.15	95.43	95.18	97.78	97.53
c5315	7964	–	–	90.34	91.02	94.57	96.41	97.32	98.02
c7552	10921	–	–	87.46	88.56	94.29	95.26	97.9	97.37

MISR size		6		7		8		9	
circuit	faults	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]
b04	2846	98.7	99.89	99.61	100.	99.79	99.86	99.79	100.
b11	2382	98.95	100.	99.58	100.	99.75	100.	99.83	100.
c499	970	98.34	97.51	98.44	99.48	99.37	100.	99.79	100.
c880	1582	98.98	100.	99.87	100.	100.	100.	99.87	100.
c1355	2618	98.97	99.69	99.85	99.73	99.58	100.	99.96	100.
c1908	2581	99.73	99.07	99.3	99.96	99.81	100.	99.96	100.
c2670	3613	98.45	97.75	99.64	100.	99.72	99.06	99.78	100.
c5315	7964	98.93	99.9	99.5	100.	99.69	100.	99.82	100.
c7552	10921	99.07	98.09	99.39	99.96	99.71	100.	99.7	100.

than both LFSR-MISR and CA90150-MISR. We then compare the results of ZATPG with the results of conventional SAT-ATPG. The columns "ZATPG" show the fault coverage achieved by our algorithm, whereas the columns "ATPG" show the coverage of a test generated by an ATPG that does not consider the compactor. The coverage is a percentage of faults that are detected after the compaction. These are the measurements for acceptable aliasing parameter M set to ∞. No zero-aliasing spatial compactors were constructed for circuits c5315 and c7552 with output size 2. Aliasing in temporal compactors and coverage for these sizes was not measured, as the aliasing in the spatial compactor would skew the results. Thus the respective entries are missing

The achieved results are comparable, but our ZATPG performs slightly worse for compactors of small size. As the size of the compactor increases, ZATPG is catching up and overtaking ATPG in terms of coverage.

Worse performance at small compactors is caused by the fact that the ZATPG stops generating new patterns, when no improvement in coverage can be made, even if there is no pattern for some remaining faults. ATPG, on the other hand, generates test patterns

Table 4.3: Fault coverage for conventional and augmented ATPG (CA-MISR 90/150)

MISR size		2		3		4		5	
circuit	faults	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]
b04	2846	80.36	79.02	78.55	80.8	95.18	95.78	95.92	94.82
b11	2382	80.94	80.48	77.52	73.74	94.83	95.25	95.37	97.39
c499	970	82.37	78.35	84.3	83.99	94.82	90.06	97.82	87.45
c880	1582	77.43	81.67	82.03	82.91	95.56	97.78	95.88	98.03
c1355	2618	78.19	73.49	78.94	82.42	94.07	93.99	94.72	94.07
c1908	2581	85.08	76.09	81.08	80.61	96.9	90.88	96.19	91.73
c2670	3613	75.64	73.01	80.23	72.83	95.26	95.54	96.53	94.65
c5315	7964	–	–	75.8	77.5	94.54	95.91	96.55	95.83
c7552	10921	–	–	77.15	68.46	94.68	93.53	95.53	93.58

MISR size		6		7		8		9	
circuit	faults	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]
b04	2846	99.15	99.96	92.67	96.19	99.68	100.	99.75	100.
b11	2382	98.74	99.92	93.04	95.36	99.28	99.96	99.79	100.
c499	970	99.17	99.58	95.73	95.62	99.79	94.57	99.06	99.9
c880	1582	99.05	100.	95.74	95.93	99.43	100.	100.	100.
c1355	2618	98.81	99.16	94.82	95.17	99.23	100.	99.19	100.
c1908	2581	99.38	96.7	95.45	95.18	99.84	100.	99.3	99.88
c2670	3613	98.72	100.	94.62	84.43	99.31	97.58	99.5	100.
c5315	7964	98.55	99.92	92.66	92.5	99.43	99.71	99.51	99.12
c7552	10921	99.04	99.07	94.6	92.1	99.45	99.65	99.52	99.84

for all faults without considering the compactor; the loss of coverage is then caused only by aliasing in the MISR.

Both ATPG and ZATPG perform slightly worse for the CA90150-MISR in comparison to the LFSR-MISR. Especially for small compactor sizes, the drop in coverage is noticeable. Even so, ZATPG achieves higher coverage than ATPG in larger CA90150 compactors. This observation denies our initial theory – CA-based MISRs are generally not more efficient than LFSR-based MISRs when used in connection with ZATPG. This could be attributed to the high efficiency of the ZATPG process, which is able to find zero-aliasing test vectors regardless of the compactor design.

4.6.4 Aliasing

The targeted aliasing for ZATPG is zero, but as can be seen from Table 4.4, this cannot be always achieved. The Table shows how the choice of the parameter M influences the results. The column “aliasing” shows the aliasing in the LFSR-based MISR, the column

Table 4.4: Aliasing in LFSR-MISR

	M	3		5		10	
circuit	MISR size	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]
c499	6	0.31	95.01	0.21	99.06	0.10	99.90
c880	5	0.13	98.98	0.51	97.72	0.00	100
c1355	6	0.04	92.87	0.15	93.72	0.19	97.16
c2670	6	0.08	94.09	0.08	96.50	0.00	100
c7552	7	0.00	99.67	0.00	99.83	0.00	99.86

	M	50		∞		ATPG
circuit	MISR size	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	coverage [%]
c499	6	0.00	96.57	0.00	96.57	99.38
c880	5	0.00	100	0.00	100	97.78
c1355	6	0.00	100	0.00	100	98.51
c2670	6	0.36	98.00	0.36	98.00	99.06
c7552	7	0.00	99.95	0.00	99.95	99.51

"coverage" shows the total test coverage, including aliasing.

This illustrates the need to relax the requirement of zero-aliasing in the ZATPG. Allowing only a small number of aliasing by setting M to values less than 10 leads to worse performance of the ZATPG compared to ATPG. On the other hand, increasing the amount of allowed aliasing to values over 50 does not further increase the coverage, hinting that the amount of aliasing during computation of single test pattern p_i is not higher. We choose to not limit aliasing during our testing ($M = \infty$).

4.6.5 Robustness

Both the fault coverage and test length achieved by ZATPG naturally depend on the fault ordering [22]. However, a *robust* algorithm should minimize this dependency.

We have examined the robustness of ZATPG on selected circuits by randomly changing the initial ordering of faults, as given in the description of the algorithm (Subsection 4.5, line 6 of Algorithm 4.1).

Then, we have observed the fault coverage and test length for different random fault orderings. As the ADI-based fault selection strategy has been found efficient in reducing the test length, it is included in the experiments as a reference.

4.6.5.1 Fault coverage

Distributions of fault coverage for circuits c2670 and c1355 [6] are shown in Figure 4.6, these were computed for the smallest LFSR-MISR size where the original fault ordering leads to the complete fault coverage. The LFSR-MISR sizes are 6 bits for c1355 and 7 bits

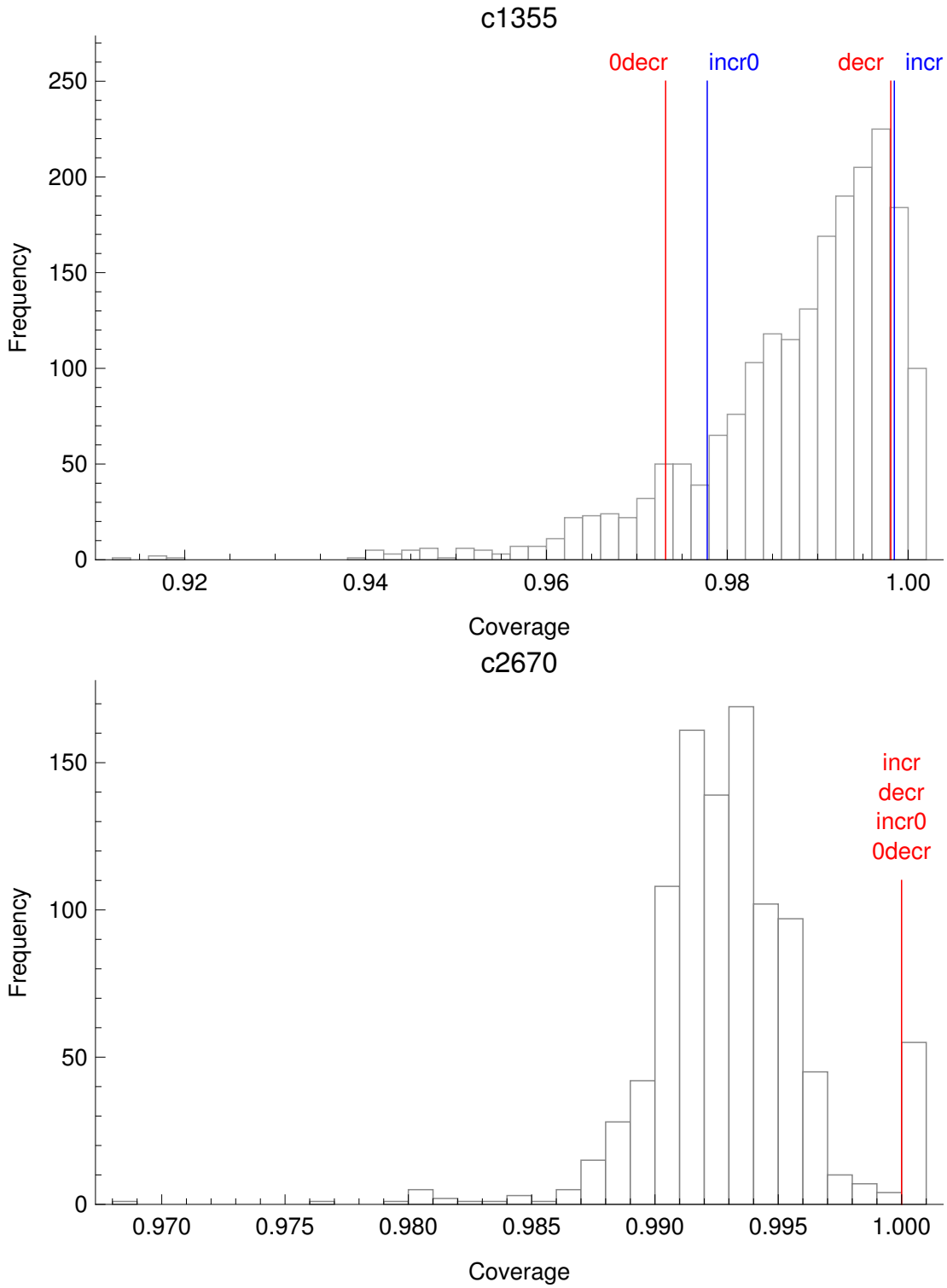

Figure 4.6: Robustness of ZATPG: test coverage for random fault ordering.

for c2670. For circuit c1355, 2007 measurements were made and for circuit c2670, 1003 measurements were made. The parameter $M = \infty$ was used for both circuits.

We can see that the algorithm is not robust but there is a clear trend towards a complete fault coverage in circuit c1355, in circuit c2670 this trend is not so clear. For these circuits, 6.8 %, and 5.5 % of orderings, respectively, lead to complete fault coverage. This result suggests that it might be possible to achieve complete fault coverage in smaller compactors if we were able to find a better fault selection strategy.

4.6.5.2 Test length

Distributions of the test lengths generated for circuits c1355 and c1908 are shown in Figure 4.7. The compactor used was the LFSR-MISR with width $w = 6$ and $w = 8$, respectively. We see that the test length varies wildly, up to a factor of 2 for the circuit c1355.

While the ADI orderings *decr* and *decr0* performed better than the original ordering, they still produce significantly bigger tests than is necessary. This suggests that the algorithm is not robust and there is a possibility of improvement, possibly by finding a better fault ordering strategy.

Circuits c2670 and b04 are examples of circuits where test lengths for the majority of fault orderings are 253 and 127. Despite this, the difference between the best and worst ordering is still over the factor of 2 for the circuit c2670. While ADI orderings behave as expected for circuit b04, circuit c2670 is an anomaly as the orderings *incr* and *incr0* give better results than the *decr* and *0decr* orderings. Also, note that the ADI orderings themselves are computed using the Monte Carlo simulation.

4.6.6 Compactor size

The size of the smallest MISR for which a complete fault coverage was found is shown in Table 4.5, for both LFSR-MISR and CA-90/150-MISR. The column "ATPG" shows the needed size of a MISR for which conventional ATPG found a complete (and zero-aliasing) test. The column "ZATPG" shows the needed MISR size for our algorithm to achieve complete fault coverage.

The search for the smallest MISR with zero aliasing and complete coverage was done by testing all sizes of the compactor from the smallest up to the size where zero aliasing was achieved.

Our ZATPG algorithm needs significantly smaller compactors because it is guided towards zero aliasing. The conventional ATPG, on the other hand, produces a test for all faults without heeding the compactor and the aliasing is then a result of the aliasing probability in the compactor.

In comparison to LFSR-MISR, the CA-MISR achieves zero aliasing in compactors of similar sizes, if slightly smaller in the case of ATPG. In the case of ZATPG, the size is also similar, but slightly higher. In both cases, however, ZATPG achieves a smaller compactor size with zero aliasing, except for circuit c7552. This is more distinct for the LFSR-MISR.

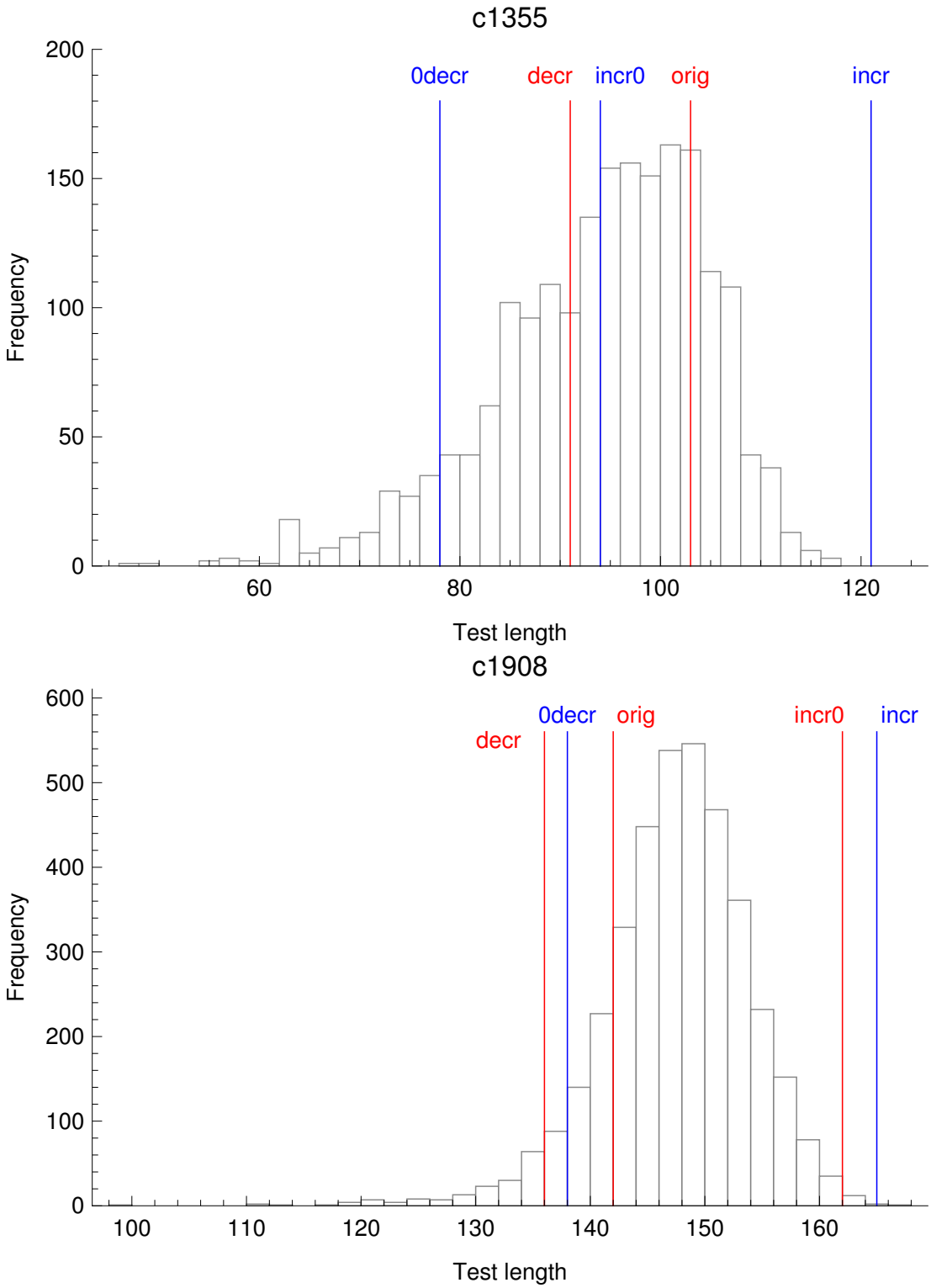

Figure 4.7: Robustnes of ZATPG: test set length from random fault ordering.

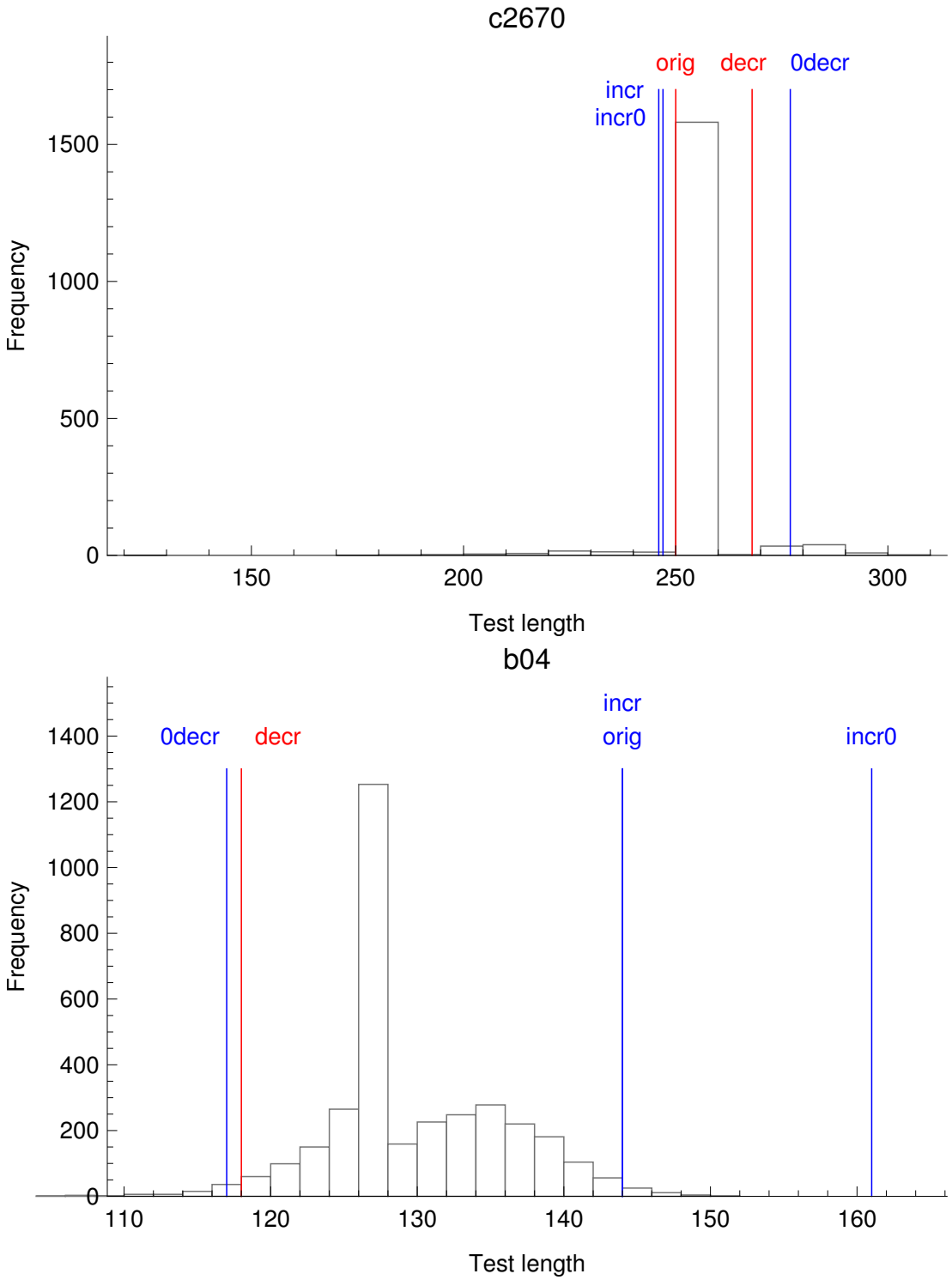

Figure 4.8: Robustnes of ZATPG: test set length from random fault ordering.

Table 4.5: Minimal size of MISR with zero-aliasing test

		LFSR-MISR		CA-MISR 90/150	
circuit	faults	ATPG	ZATPG	ATPG	ZATPG
b04	2846	10	7	10	8
b11	2382	11	6	10	9
c499	970	15	8	10	10
c880	1582	12	5	10	6
c1355	2618	11	8	12	8
c1908	2581	10	8	10	8
c2670	3613	11	7	12	6
c5315	7964	12	7	12	10
c7552	10921	8	8	9	10

Table 4.6: Minimal size of MISR with zero-aliasing test

		run time [s]	
circuit	faults	ATPG	ZATPG
b04	2846	17	73
b11	2788	12	41
c499	970	3	5
c880	1582	3	16
c1355	2618	6	53
c1908	2581	10	39
c2670	3613	77	297
c5315	7964	283	1299
c7522	10921	280	2452

4.6.7 Algorithm computation time

A comparison of the computation time of the ZATPG and a conventional SAT-ATPG can be seen in Table 4.6. The computation time was measured for the MISR sizes, where each algorithm achieved zero aliasing.

Figure 4.9 illustrates the dependence of the ZATPG computation time on the MISR size. It is apparent that the computation time sharply rises as the MISR size decreases. This is due to the high aliasing probability in the small compactors, which leads to a high amount of ATPG re-runs with additional constraints. Indeed, the computation time for the larger MISR sizes is closing the gap between ZATPG and a conventional SAT-ATPG. This gap is never really closed, however, as ZATPG is necessarily running more fault simulations of the CUT.

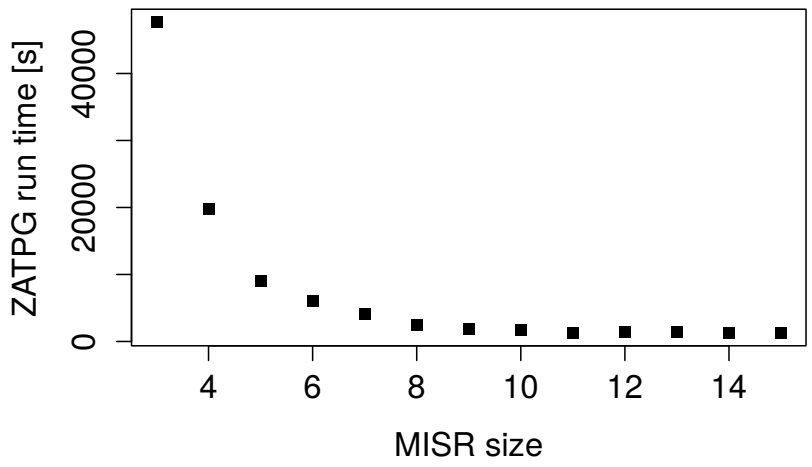

Figure 4.9: Runtime of the ZATPG algorithm depending on the MISR size for the c7552 benchmark circuit.

4.7 Optimization-Based ZATPG

A critical problem with the ZATPG algorithm is the resulting test length. This is given by the fact that the algorithm does not use any form of test compaction, static or dynamic. Additionally, a fault can be tested multiple times, as it is repeatedly detected and aliased. A related problem is the test generation time, caused by the repeated rejection of a test pattern, especially for small compactors.

In this section, we present our approach to reduce *both* the aliasing and the test length. For that, we use an optimizing Pseudo-Boolean Optimization (PBO) solver in the ATPG process. We build on our previous algorithm, ZATPG, described earlier in this chapter. We call this revised algorithm ZATPG-PBO.

These are preliminary results, the ZATPG-PBO was not published in peer-reviewed paper as of the time of writing this text.

4.7.1 Constraining Fault Aliasing

When using hard constraints on aliasing, we can get into a situation where no test pattern can be generated for our selection of anti-aliased faults. To avoid such a situation we use an optimization criterion to minimize the number of aliased faults. We let the solver select (some) minimal subset of anti-aliased faults to increase the probability that a test pattern for the currently selected fault exists.

Adding all faults to the miter for anti-aliasing would be too expensive – both in the miter construction and PBO solving. Thus we only add faults that are likely to be aliased.

We detect such faults by simulation after a test pattern is generated. If necessary, we construct a new miter with appended faults and try to generate the test pattern again.

Example 4.7.1. Let us have a set of faults $F_a = \{f_1, \dots f_n\}$ that were aliased by a newly generated test pattern p_1. The pattern is simulated and if the overall coverage has decreased because of aliasing, the pattern p_1 is not accepted. The aliased faults from F_a are added to the miter. A new test pattern p_2 is generated. The PBO solver minimizes the number of faults in F_a that are aliased by p_2. The pattern p_2 is simulated and if the overall fault coverage increases, it is accepted.

4.7.2 Test Compaction

Dynamic test compaction is used, because of the test's strict sequentiality – the aliasing depends on all test patterns and their ordering. For this reason, static test compaction or any other test post-processing cannot be used.

For this, we adapt recent OTG techniques [20]. With this method, we do not test a single fault, instead, we select a set of faults F_t to be tested. Again, we use a PBO solver to select the maximal subset $F'_t \subset F_t$ of faults that can be tested by a single test pattern.

We combine this optimization criterion with the previous one, the aliasing minimization. Our goal is to maximize the overall fault coverage gained by the generated test pattern. The optimization cost of one newly detected fault is the same as the cost of one fault that was not aliased, i.e. we do not care if the pattern detects fault f_{t1} while aliasing fault f_{a1} or if neither is detected and aliased. This also leads to decreased complexity of the constructed miter – the newly selected faults and anti-aliased faults are uniform in the miter and PBO instance.

4.7.3 Miter and PBO Instance

The miter is constructed similarly to the ZATPG algorithm (Figure 4.2). The difference is that the faulty responses are not hard-constrained to be different from the fault-free response. Instead, they are included in the optimization criterion of the PBO instance. The optimization criterion is to maximize the number of differing faulty responses. See signals BD_1 through BD_n in Figure 4.10.

The output of the solver is the test pattern with the maximum of selected faults detected (maximum of the BD_i signals set to the value 1). These are either newly targeted faults or faults with prevented aliasing. Because the difference outputs are not constrained, this test pattern will always exist, possibly decreasing the fault coverage.

4.7.4 The Algorithm Overview

A high-level view of the overall ZATPG-PBO algorithm is shown in Algorithm 4.2. The flow of the algorithm is as follows.

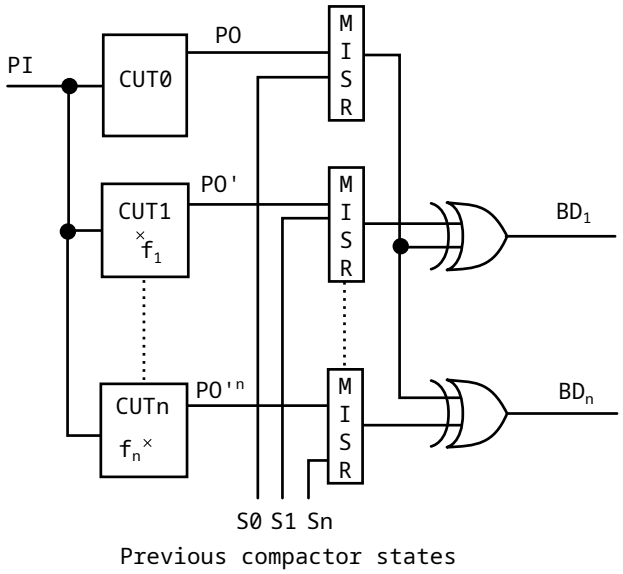

Figure 4.10: Miter for multiple targeted and anti-aliased faults.

Algorithm 4.2 General overview of the algorithm.

```
1: procedure ATPG(C)
2:     P ← ∅
3:     F ← faults(C)
4:     repeat
5:         F_t ← select(F)
6:         F_a ← ∅
7:         repeat
8:             p ← genPattern(C, F_t ∪ F_a)
9:             F_d, F_a ← simulate(C, p)
10:        until |F_d| − |F_a| ≥ M_i ∨ |F_a| ≤ M_a
11:        if |F_d| − |F_a| ≥ M_i then
12:            P ← append(P, p)
13:            F ← update(F, F_d, F_a)
14:        end if
15:    until F = ∅ ∨ select(F) = ∅
16:    return P
17: end procedure
```

4.7.4.1 Fault generation

On line 3, a fault list for the circuit C is prepared. This fault list is precomputed and ordered by the accidental detection index (ADI) [43]. We also use this step to remove redundant faults from the fault list.

4.7.4.2 Fault selection

On line 5, a set F_t of targeted faults is selected from F. On the first invocation or after a test pattern was accepted, up to M_f first faults are selected. If a satisfactory test pattern was not found, faults F_t are re-targeted and first $n\frac{M_f}{2}$ faults are skipped for every (n) unsuccessful search.

4.7.4.3 Pattern generation

On line 8, a test pattern p is generated, as described in previous subsections. The first run is done only for targeted faults F_t. Subsequent runs are done also for aliased faults F_a. This step is repeated until (line 10) a satisfactory pattern is generated or the number of aliased faults grows over the maximum number of anti-aliased faults M_a. Care must also be taken to detect that no satisfactory pattern exists. We do this by detecting that F_a was not changed since the last iteration.

4.7.4.4 Fault simulation

On line 9, a fault simulation is done for *all* faults and the fault coverage is analyzed; aliased F_a and newly detected F_d faults are recorded. In addition to the circuit under test, the compactor is also simulated. The internal state of the compactor is kept for each fault by the algorithm.

4.7.4.5 Accepting pattern

If the generated pattern is satisfactory – increases the fault coverage by at least M_i (line 11) – it is recorded (line 12). The internal state of the compactor is also updated for each fault and fault-free circuit. The fault list F is updated (line 13), detected faults are removed, and aliased faults are added back. The fault list remains ordered by ADI.

4.7.4.6 Algorithm termination

The algorithm terminates when all faults are covered or if the fault list is exhausted (line 15). In the case of fault list exhaustion, there are some faults left. As the last attempt, the entire algorithm can be optionally restarted with an increased number of targeted faults M_f. This restart keeps the already generated test set and the internal state of the algorithm. This is however effective only if the number of undetected faults is greater than M_f.

4.8 ZATPG-PBO: Experimental Results

In this Section, the experimental results for the ZATPG-PBO are presented. The used PBO solver is the Minisat+ [21]. Experiments were run on the circuits from the ISCAS'85 benchmark [6]. The used fault model is the stuck-at model.

The benchmark circuits were modified in the following way: first, an irredundant spatial output response compactor without aliasing was randomly generated. The compactor was appended to the primary outputs of the CUT. The spatial compactor was constructed as a collection of disjoined XOR trees. Second, an LFSR-MISR was used as the temporal output response compactor. The size of the LFSR was chosen to be equal to the number of spatial compactor outputs.

4.8.1 Pilot experiments

In our pilot experiments, we focused on finding suitable parameter settings for the algorithm. We have chosen the following setup:

- default *targeted faults set* size, $M_f = 20$,
- the maximal allowed *targeted faults set* size, $M_{fm} = 800$,
- the acceptable improvement, $M_i = 1$,
- the allowed *anti-aliasing window* size, $M_a = 100$.

We have determined the best size of the targeted faults set (M_f) to be around 20 – 50. This range provides the best compromise between the test length and computation time for tested circuits. We have chosen the lower value and allowed the momentary increase of the targeted set size; due to the size of tested circuits, $M_{fm} = 800$ means that the maximum size of the targeted faults set is effectively unrestricted. In Figures 4.11 and 4.12 is shown the influence of this parameter on the computation time and the test length.

The impact of the anti-aliasing window (M_a) is the same as the size of the targeted faults set, as the implementation and encoding in CNF are identical. However, the aliasing is rather small for most patterns, and for many is not considered at all. The average impact of the parameter is lower than the number of targeted faults. The parameter of acceptable improvement (M_i) is set to 1, as the ZATPG-PBO uses the fault coverage improvement as the optimization criterion.

4.8.2 Fault Coverage and Test Length

Up to our knowledge, there are no similar works to be able to compare with. Thus, we present only a comparison with our previous method [31]. The comparison is shown in Table 4.7.

The results are presented for MISR sizes ranging from 4 to 7. In the column *ATPG* is the fault coverage achieved with a classical ATPG as presented in [31]. First, a full-coverage

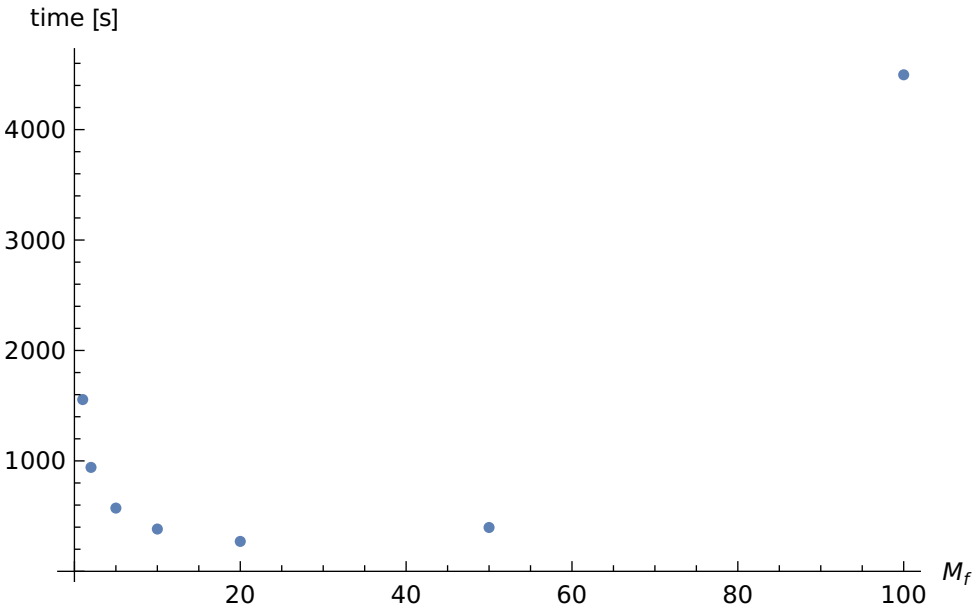

Figure 4.11: Computation time for circuit c2670.

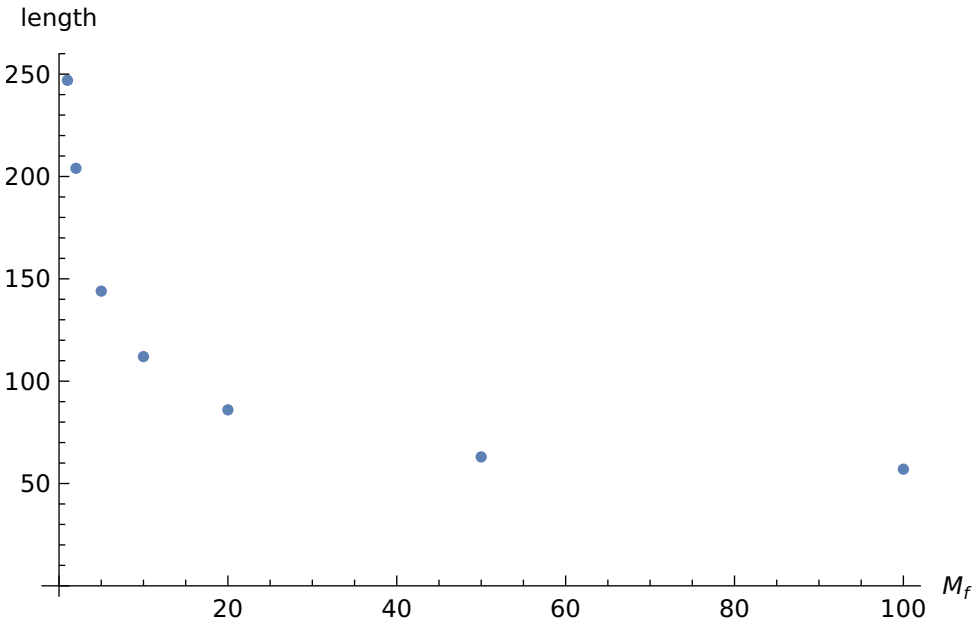

Figure 4.12: Test length for circuit c2670.

Table 4.7: Fault coverage and test length

MISR SIZE		4					5				
algorithm		ATPG	ZATPG		ZATPG-PBO		ATPG	ZATPG		ZATPG-PBO	
circuit	faults	[%]	[%]	length	[%]	length	[%]	[%]	length	[%]	length
c499	2550	97.20	94.72	52	96.47	72	96.16	97.2	52	99.88	90
c880	1820	95.12	97.66	75	99.83	32	96.76	100	80	100	28
c1355	2950	95.87	95.83	75	97.97	75	97.21	96.21	66	99.80	86
c1908	2829	96.70	92.94	78	97.00	102	98.80	94.56	86	99.72	116
c2670	4108	95.43	95.18	119	99.81	81	97.78	97.53	151	100	87
c5315	8930	94.57	96.41	150	-	-	97.32	98.02	151	99.62	93
c7552	11529	94.29	95.26	128	-	-	97.90	97.37	155	99.83	110
b04	3572	95.71	96.06	75	-	-	97.96	99.12	106	99.94	74
b11	3266	93.52	94.95	87	98.59	105	97.39	99.49	124	99.75	91

MISR SIZE		6					7				
algorithm		ATPG	ZATPG		ZATPG-PBO		ATPG	ZATPG		ZATPG-PBO	
circuit	faults	[%]	[%]	length	[%]	length	[%]	[%]	length	[%]	length
c499	2550	98.34	97.51	51	99.80	84	98.44	99.48	61	100	89
c880	1820	98.98	100	78	100	27	99.87	100	90	100	26
c1355	2950	98.97	99.69	91	100	87	99.85	99.73	89	100	87
c1908	2829	99.73	99.07	130	99.79	112	99.30	99.96	132	100	112
c2670	4108	98.45	97.75	186	100	82	99.64	100	268	100	86
c5315	8930	98.93	99.90	189	100	86	99.50	100	217	100	80
c7552	11529	99.07	98.09	160	99.99	100	99.39	99.96	269	100	93
b04	3572	98.70	99.89	114	99.94	66	99.61	100	118	100	68
b11	3266	98.95	100	126	99.91	88	99.58	100	121	99.97	87

test is generated for the combinational circuit with a spatial compactor but without any knowledge about the temporal compactor. Second, fault simulation is run with the temporal compactor and the fault coverage is analyzed.

In the column *ZATPG* is the coverage and test length achieved with the ZATPG. This is the fault coverage achieved by zero-aliasing ATPG with hard anti-aliasing constraints.

In the column *ZATPG-PBO* are the coverage and test length achieved by the algorithm described in this paper. We have chosen to present experiments with the parameter for the number of targeted faults set to $M_f = 20$. This setting returned the best overall results over the tested circuits. The missing values indicate that the algorithm did not terminate within a time limit for the computation.

The new algorithm, ZATPG-PBO, achieved higher coverage than ZATPG in most cases. It also achieved lower test length in cases where the fault coverage is similar with a notable exception in the circuit c499. Lower test lengths in ZATPG for other circuits can be attributed to lower fault coverage in ZATPG.

4.8.3 Anti-aliasing

In Table 4.8 is shown maximally used anti-aliasing, i.e., how many aliased faults we appended to the miter to prevent aliasing, up to the maximum $M_a = 100$. Only aliasing that

Table 4.8: Maximal anti-aliasing

MISR	3	4	5	6	7	8	9
c499	-	100	100	100	36	13	12
c880	100	100	52	24	6	4	5
c1355	100	100	100	78	32	11	5
c1908	-	100	100	45	18	14	13
c2670	100	100	100	62	81	21	18
c5315	-	-	100	100	39	13	0
c7552	-	-	100	100	65	41	29
b04	-	-	100	71	18	12	11
b11	100	100	96	21	24	8	9

Table 4.9: Computation time

MISR circuit	3 [s]	4 [s]	5 [s]	6 [s]	7 [s]	8 [s]	9 [s]
c499	-	58402	3029	1558	361	198	178
c880	4894	591	202	182	124	202	93
c1355	105807	6192	1324	590	414	346	156
c1908	-	19619	3106	529	334	285	312
c2670	291931	3066	411	364	271	233	317
c5315	-	-	3514	2582	1890	895	982
c7552	-	-	11419	5471	4387	1943	1541
b04	-	-	1360	752	333	202	221
b11	38549	1696	861	391	332	312	293

required appending anti-aliased faults to the miter is considered. The actual aliasing can be higher if the overall fault coverage increases.

The maximal anti-aliasing increases with smaller compactor sizes. As the probability of aliasing increases, so does the anti-aliasing. The maximum of 100 was reached in all circuits for sufficiently small compactors. In such case, the algorithm uses fault re-targeting rather than appending more anti-aliased faults. In most of these cases, full coverage was not achieved. This suggests that fault re-targeting is not as effective as anti-aliasing.

4.8.4 Computation time

In Table 4.9 are shown computation times. The computation time increases for smaller compactors, where the aliasing probability is higher. This leads to bigger miters and PBO instances and also to the higher number of PBO solver invocations.

There is a steep step in computation time for all circuits. This coincides with runs where full coverage was not achieved. This also indicates that fault re-targeting is not as efficient as anti-aliasing.

4.9 Conclusions

A modified SAT-based ATPG process, *ZATPG* for finding a test with zero aliasing in any given temporal compactor is presented. In contrast to other approaches to minimizing the aliasing, ZATPG does not require any compactor modification. Conversely, it can be used for *any* spatial and temporal compactor design, as long, as they can be *unrolled*, i.e., their combinational part can be extracted.

A simplified algorithm for linear space compactors is also presented. This algorithm is then evaluated by the criteria of fault coverage and resulting test length.

As a consequence of generating a test with zero (or reduced) aliasing in the test generation process, the compactor size can be reduced, while preserving the fault coverage. For tested benchmark circuits, we achieved complete fault coverage with MISRs of smaller sizes than the conventional ATPG. This is due to ZATPG being guided towards zero-aliasing, whereas in the test produced by other ATPG it is dependent on the aliasing probability in the compactor.

Two different linear compactor designs are used, an LFSR-based and a cellular automata-based MISR. Our initial assumption that the CA-based MISR could exhibit smaller aliasing, has not been confirmed. Probably this is because of the ZATPG flexibility, i.e., its ability to find zero-aliasing test sequences regardless of the compactor used.

The fault coverage and test length produced by ZATPG depend on the ordering the faults are processed in. In this respect, ZATPG is not robust. It does however overtake this disadvantage with an increasing compactor size. Moreover, an ADI fault selection heuristic has been used, to improve the test length. Even so, the test length is the main disadvantage of the ZATPG.

To improve the test length and to a lesser extent the computation time, an optimization variant of the ZATPG, the ZATPG-PBO, was presented. This variant of the algorithm uses a Pseudo-Boolean optimization to maximize the fault coverage of each test pattern. An MTTG technique is combined with the minimization of aliasing in a compactor.

This change to the algorithm improves both the fault coverage and the test length for most compared circuits.

Chapter 5

Conclusions

Testing of contemporary digital circuits is increasingly complex and important, thus, the testing has been and continues to be, extensively researched. The aim of this dissertation thesis are two topics from this research area: an application-oriented testing of circuits implemented in FPGAs and an ATPG generating zero-aliasing test sets.

In Chapter 1, an introduction and goals of this thesis are presented.

In Chapter 2, the theoretical background and state-of-the-art relevant to this thesis is presented. An overview of digital circuit testing is presented. The focus is given to testing techniques involving design for testability and built-in self-test. A brief introduction to automated test pattern generators is given, with an overview of test compaction techniques. Output response analysis is discussed with a focus on output response compaction and fault aliasing in response compactors.

In Chapter 3, the first contribution of this thesis is presented. An application-oriented FPGA testing is analyzed. A combined fault model with stuck-at and bit-flip faults is proposed, and an ATPG working natively with both fault models is presented. The properties of the two fault models are analyzed and experimentally examined.

In Chapter 4, the second contribution of this thesis is presented. Fault aliasing in temporal output compactors is considered and an ATPG algorithm (ZATPG) is presented to decrease or eliminate the aliasing. ZATPG is experimentally examined and its strong and weak points are identified. A modified version of the algorithm, based on pseudo-Boolean optimization, is presented to eliminate one of the weak points, the test length. It is shown that inclusion of dynamic test compaction techniques, together with optimization-based aliasing prevention, lead to decreased test length *and* decreased fault aliasing.

In Chapter 5, summary and contributions are presented. Possible future work is suggested and the thesis is concluded.

5.1 Summary

A combined fault model for application-oriented FPGA testing was proposed. Properties of two fault models, stuck-at and bit-flip, were examined in the context of application-oriented

FPGA testing and their interaction was analyzed.

An ATPG for generating a zero-aliasing test, ZATPG, was proposed. The algorithm was shown to improve fault coverage and reduce fault aliasing in output response compaction. As a preliminary result, an optimization-based version of the algorithm, ZATPG-PBO, was presented. ZATPG-PBO mitigates a weak point of the ZATPG; the test length is significantly improved.

These results were presented in the scientific community and published in proceedings of 2 international conferences and 1 journal. Both of these results we cited by other authors in journal papers.

5.2 Contributions of the Dissertation Thesis

1. Direct encoding of the bit-flip faults and experimental analysis of a combined fault model using bit-flip and stuck-at faults: Practical output of this work is an SAT-based ATPG that works natively with both bit-flip and stuck-at faults. This result directly ties to the second contribution, where it serves as a basis to construct necessary tools. The research has further continued in related work [A.4] that is not part of this thesis.

2. Zero-aliasing ATPG, ZATPG: An ATPG that generates a test with zero, or minimized, aliasing in temporal output response compactors (sequential circuits). ZATPG, and its optimization-based variant, can be used to improve fault coverage by lowering the amount of fault aliasing in compactors. The technique can be used for any fault model and output response compactor that can be described as a Boolean formula (CNF).

5.3 Future Work

The author of the dissertation thesis suggests exploring the following:

- Other fault models and compactor designs could be incorporated with our technique and analyzed.

- Test pattern generation, in a BIST circuit, is not considered. It would be interesting to investigate the possibility to include this aspect of circuit testing in our algorithm.

Bibliography

[1] C. Albrecht. IWLS 2005 benchmarks. Technical report, June 2005.

[2] T. Bogue, M. Gossel, H. Jurgensen, and Y. Zorian. Built-in self-test with an alternating output. In *Proceedings Design, Automation and Test in Europe*, pages 180–184, Feb. 1998. doi:10.1109/DATE.1998.655854.

[3] J. Borecký, M. Kohlík, P. Kubalík, and H. Kubátová. Fault models usability study for on-line tested FPGA. In *14th Euromicro Conference on Digital System Design (DSD)*, pages 287–290, Aug 2011. doi:10.1109/DSD.2011.42.

[4] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems, 1989*, pages 1929–1934 vol.3, May 1989. doi:10.1109/ISCAS.1989.100747.

[5] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proceedings of IEEE Int'l Symposium Circuits and Systems (ISCAS'85)*, pages 677–692. IEEE Press, Piscataway, N.J., 1985.

[6] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *IEEE International Symposium Circuits and Systems (ISCAS'85)*, pages 677–692. IEEE Press, Piscataway, N.J., 1985.

[7] K. Chakrabarty and J. P. Hayes. Test response compaction using multiplexed parity trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(11):1399–1408, Nov 1996. doi:10.1109/43.543772.

[8] Krishnendu Chakrabarty. Zero-aliasing space compaction using linear compactors with bounded overhead. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(5):452–457, Aug. 2002.

[9] Krishnendu Chakrabarty and P. Hayes, John. Efficient test response compression for multiple-output circuits. In *IEEE International Test Conference*, pages 501–510, Oct. 1994.

[10] Huan Chen and J. Marques-Silva. A two-variable model for SAT-based ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(12):1943–1956, Dec 2013. doi:10.1109/TCAD.2013.2275254.

[11] F. Corno, M. S. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. In *IEEE Design Test of Computers*, pages 44–53 vol.17, Jul 2000.

[12] S. R. Das, M. Sudarma, M. H. Assaf, E. M. Petriu, W. B. Jone, K. Chakrabarty, and M. Sahinoglu. Parity bit signature in response data compaction and built-in self-testing of VLSI circuits with nonexhaustive test sets. *IEEE Transactions on Instrumentation and Measurement*, 52(5):1363–1380, Oct 2003. doi:10.1109/TIM.2003.818547.

[13] Karl-Heinz Diener, Günter Elst, Eero Ivask, Jaan Raik, and Raimund Ubar. FPGA design flow with automated test generation. In *11th Workshop on Test Technology and Reliability of Circuits and Systems*, pages 120–123, 1999.

[14] Rolf Drechsler, Daniel Tille, and Stephan Eggersglüß. Speeding up SAT-based ATPG using dynamic clause activation. In *Asian Test Symposium*, pages 177–182, Nov. 2009.

[15] Geetani Edirisooriya and P. Robinson, John. Test generation to minimize error masking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(4):540–549, April 1993.

[16] Geetani Edirisooriya, P. Robinson, John, and Samantha Edirisooriya. On the performance of augmented signature testing. In *IEEE International Symposium on Circuits and Systems*, pages 1607–1610, May 1993.

[17] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24605-3_37.

[18] Stephan Eggersglüß and Rolf Drechsler. Robust algorithms for high quality test pattern generation using Boolean satisfiability. In *IEEE International Test Conference*, pages 1 –10, Nov. 2010.

[19] Stephan Eggersglüß, R. Krenz-Baath, Andreas Glowatz, Friedrich Hapke, and Rolf Drechsler. A new SAT-based ATPG for generating highly compacted test sets. In *15th IEEE Design and Diagnostics of Electronic Circuits and Systems*, pages 230–235, April 2012.

[20] S. Eggersglüß, K. Schmitz, R. Krenz-Bååth, and R. Drechsler. On optimization-based ATPG and its application for highly compacted test sets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):2104–2117, 2016. doi:10.1109/TCAD.2016.2552822.

[21] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 11 2006.

[22] Petr Fiser, Jan Schmidt, and Jiri Balcarek. Sources of bias in EDA tools and its influence. In *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 258–261, April 2014.

[23] F. Flores, Paulo, C. Neto, Horácio, and P. Marques-Silva, João. On applying set covering models to test set compaction. In *Symposium on VLSI Circuits*, pages 8–11, March 1999.

[24] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32(12):1137–1144, Dec 1983. doi:10.1109/TC.1983.1676174.

[25] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30(3):215–222, March 1981. doi:10.1109/TC.1981.1675757.

[26] Sybille Hellebrand, Janusz Rajski, Steffen Tarnick, Srikanth Venkataraman, and Bernard Courtois. Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers. *IEEE Transactions on Computers*, 44(2):223–233, February 1995.

[27] P. D. Hortensius, R. D. McLeod, and H. C. Card. Cellular automata-based signature analysis for built-in self-test. *IEEE Transactions on Computers*, 39(10):1273–1283, Oct 1990. doi:10.1109/12.59857.

[28] M. S. Hsiao, E. M. Rudnick, and J. H. Patel. Fast algorithms for static compaction of sequential circuit test vectors. In *Proceedings. 15th IEEE VLSI Test Symposium (Cat. No.97TB100125)*, pages 188–195, April 1997. doi:10.1109/VTEST.1997.600260.

[29] W.-K. Huang, F.J. Meyer, N. Park, and F. Lombardi. Testing memory modules in SRAM-based configurable FPGAs. In *International Workshop on Memory Technology, Design and Testing*, pages 79–86, Aug 1997. doi:10.1109/MTDT.1997.619399.

[30] M. Hunger, Sybille Hellebrand, Alejandro Czutro, Ilia Polian, and Bernd Becker. ATPG-based grading of strong fault-secureness. In *15th IEEE International On-Line Testing Symposium*, pages 269–274, June 2009.

[31] Robert Hülle, Petr Fišer, and Jan Schmidt. ZATPG: SAT-based test patterns generator with zero-aliasing in temporal compaction. *Microprocessors and Microsystems*, 61:43 – 57, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0141933118300966, doi:https://doi.org/10.1016/j.micpro.2018.05.001.

[32] Jau-Shien Chang and Chen-Shang Lin. Test set compaction for combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(11):1370–1378, Nov 1995. doi:10.1109/43.469663.

[33] M. Kopec. Can nonlinear compactors be better than linear ones? *IEEE Transactions on Computers*, 44(11):1275–1282, Nov. 1995.

[34] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, Jan. 1992.

[35] Yongxia Liu and Aijiao Cui. An efficient zero-aliasing space compactor based on elementary gates combined with XOR gates. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 95–100, Nov. 2013.

[36] Edward J. McCluskey and Chao-Wen Tseng. Stuck-fault tests vs. actual defects. In *Proceedings of the 2000 IEEE International Test Conference*, ITC '00, pages 336–, Washington, DC, USA, 2000. IEEE Computer Society.

[37] K. McElvain. IWLS'93 Benchmark Set: Version 4.0. Technical report, May 1993.

[38] H. Michinishi, T. Yokohira, T. Okamoto, T. Inoue, and H. Fujiwara. A test methodology for interconnect structures of LUT-based FPGAs. In *The Fifth Asian Test Symposium*, pages 68–74, Nov 1996. doi:10.1109/ATS.1996.555139.

[39] S. D. Millman and E. J. McCluskey. Detecting bridging faults with stuck-at test sets. In *Proceedings the 1988 International Test Conference, New Frontiers in Testing*, pages 773–783, Sep 1988. doi:10.1109/TEST.1988.207864.

[40] Peter Muth. A nine-valued circuit model for test generation. *IEEE Transactions on Computers*, C-25(6):630–636, Jun 1976.

[41] M. Nicolaidis. *Soft Errors in Modern Electronic Systems*. Frontiers in Electronic Testing. Springer US, 2010.

[42] I. Pomeranz, L. N. Reddy, and S. M. Reddy. Compactest: a method to generate compact test sets for combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(7):1040–1049, July 1993. doi:10.1109/43.238040.

[43] Irith Pomeranz and M. Reddy, Sudhakar. The accidental detection index as a fault ordering heuristic for full-scan circuits. In *Design, Automation and Test in Europe*, pages 1008–1013, 2002.

[44] Bahram Pouya and Alan Touba, Nur. Synthesis of zero-aliasing elementary-tree space compactors. In *IEEE VLSI Test Symposium*, pages 70–77, 1998.

[45] K. Pradhan, D. and K. Gupta, Sandeep. A new framework for designing and analyzing BIST techniques and zero aliasing compression. *IEEE Transactions on Computers*, 40(6):743–763, 1991.

[46] K. Pradhan, D., M. Reddy, Sudhakar, and K. Gupta, Sandeep. Zero aliasing compression. In *Fault-Tolerant Computing: 20th International Symposium*, pages 254–263, June 1990.

[47] Gianfranco R. Putzolu and J. Paul Roth. A heuristic algorithm for the testing of asynchronous circuits. *IEEE Transactions on Computers*, C-20(6):639–647, Jun 1971.

[48] M. Rebaudengo, Sonza Reorda, Matteo, and M. Violante. A new functional fault model for FPGA application-oriented testing. In *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 372–380, 2002.

[49] L. N. Reddy, I. Pomeranz, and S. M. Reddy. ROTCO: a reverse order test compaction technique. In *Proceedings Euro ASIC '92*, pages 189–194, June 1992. doi:10.1109/EUASIC.1992.228026.

[50] S. Remersaro, J. Rajski, S. M. Reddy, and I. Pomeranz. A scalable method for the generation of small test sets. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 1136–1141, April 2009. doi:10.1109/DATE.2009.5090834.

[51] M. Renovell, P. Faure, J.M. Portal, J. Figueras, and Y. Zorian. IS-FPGA: a new symmetric FPGA architecture with implicit scan. In *International Test Conference*, pages 924–931, 2001. doi:10.1109/TEST.2001.966716.

[52] M. Renovell, J. Figueras, and Y. Zorian. Test of RAM-based FPGA: methodology and application to the interconnect. In *15th IEEE VLSI Test Symposium*, pages 230–237, Apr 1997. doi:10.1109/VTEST.1997.600278.

[53] M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. SRAM-based FPGA's: testing the LUT/RAM modules. In *International Test Conference*, pages 1102–1111, Oct 1998. doi:10.1109/TEST.1998.743311.

[54] M. Renovell, J.M. Portal, J. Figuras, and Y. Zorian. Minimizing the number of test configurations for different FPGA families. In *8th Asian Test Symposium (ATS'99)*, pages 363–368, 1999. doi:10.1109/ATS.1999.810776.

[55] Michel Renovell. Some aspects of the test generation problem for an application-oriented test of SRAM-based FPGAs. *Journal of Circuits, Systems and Computers*, 12(02):143–158, Feb. 2003.

[56] Michel Renovell, M. Portal, J., P. Faure, J. Figueras, and Yervant Zorian. Analyzing the test generation problem for an application-oriented test of FPGAs. In *IEEE European Test Workshop*, pages 75–80, May 2000.

[57] Michel Renovell, M. Portal, J., P. Faure, J. Figueras, and Yervant Zorian. TOF: a tool for test pattern generation optimization of an FPGA application oriented test. In *9th Asian Test Symposium*, pages 323–328, Dec. 2000.

[58] J.P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10(4):278–291, July 1966. doi:10.1147/rd.104.0278.

[59] M. Rozkovec, J. Jeníček, and O. Novák. Application dependent FPGA testing method. In *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, pages 525–530, Sept 2010. doi:10.1109/DSD.2010.65.

[60] Jan Schmidt, Petr Fiser, and Jiri Balcarek. The influence of implementation type on dependability parameters. *Microprocessors and Microsystems*, 37(6):641–648, 2013. URL: http://www.sciencedirect.com/science/article/pii/S0141933113000896, doi:https://doi.org/10.1016/j.micpro.2013.06.005.

[61] M.H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: a highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, Jan 1988. doi:10.1109/43.3140.

[62] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson. Analyzing errors with the boolean difference. *IEEE Transactions on Computers*, C-17(7):676–683, July 1968. doi:10.1109/TC.1968.227417.

[63] Warren Shum and H. Anderson, Jason. Analyzing and predicting the impact of CAD algorithm noise on FPGA speed performance and power. In *International ACM Symposium on Field-Programmable Gate Arrays*, pages 107–110, Feb. 2012.

[64] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of FPGA interconnect. In *International Test Conference*, pages 404–411, Oct 1998. doi:10.1109/TEST.1998.743180.

[65] Nur Alan Touba and C. Krishna. Reducing test data volume using LFSR reseeding with seed compression. In *IEEE International Test Conference*, pages 321–330, October 2002.

[66] G. Tromp. Minimal test sets for combinational circuits. In *1991, Proceedings. International Test Conference*, pages 204–, Oct 1991. doi:10.1109/TEST.1991.519511.

[67] G.S. Tseitin. On the complexity of derivation in propositional calculus. In J orgH. Siekmann and Graham Wrightson, editors, *Automation of Reasoning*, Symbolic Computation, pages 466–483. Springer Berlin Heidelberg, 1983.

[68] University of California, Brekeley. Berkeley logic interchange format (BLIF), 2005.

[69] S. Venkataraman, Janusz Rajski, Sybille Hellebrand, and S. Tarnick. An efficient bist scheme based on reseeding of multiple polynomial linear feedback shift registers. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 572–577, 1993.

[70] Xijiang Lin, J. Rajski, I. Pomeranz, and S. M. Reddy. On static test compaction and test pattern ordering for scan designs. In *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*, pages 1088–1097, Nov 2001. doi:10.1109/TEST.2001.966735.

[71] S. Yang. Logic synthesis and optimization benchmarks user guide: Version 3.0, Jan. 1991.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] Hülle, R.; Fišer, P.; Schmidt, J.; Borecký, J. SAT-ATPG for Application-Oriented FPGA Testing. In: *Proceedings of the 15th Biennial Baltic Electronics Conference*. Tallin, Estonia, 2016.

The paper has been cited in:

- Ben Ahmed, A.; Mosbahi, O.; Khalgui, M.; Li, Z. Boundary Scan Extension for Testing Distributed Reconfigurable Hardware Systems *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I-REGULAR PAPERS*, vol. 66, no. 7, pp. 2699 - 2708, 2019. ISSN 1549-8328.
- Kanno, Y.; Toba, T.; Shimamura, K.; Kanekawa, N. Design Method for Online Totally Self-Checking Comparators Implementable on FPGAs *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, vol. 28, no. 3, pp. 726 - 735, 2020. ISSN 1063-8210.
- Zhang, F.; Guo, C.; Zhang, S.; Chen, L.; Li, X.; Sun, H.; Meng, Y.; Chen, Q. Research on Hex Programmable Interconnect Points Test in Island-Style FPGA *ELECTRONICS*, vol. 9, no. 12, 2020.

[A.2] Hülle, R.; Fišer, P.; Schmidt, J. SAT-based ATPG for Zero-Aliasing Compaction. In: *Proc. of the 20th Euromicro Conference on Digital System Design*. Vienna, Austria, 2017.

[A.3] Hülle, R.; Fišer, P.; Schmidt, J. ZATPG: SAT-based test patterns generator with zero-aliasing in temporal compaction. In: *Microprocessors and Microsystems*. Amsterdam, Netherlands, 2018.

The paper has been cited in:

- Novak, O.; Rozkovec, M.; Pliva, J. Decompressors using nonlinear codes. *Microprocessors and Microsystems*, vol. 76, 2020.

[A.4] Borecký, J.; Hülle, R.; Fišer, P. Evaluation of the SEU Faults Coverage of a Simple Fault Model for Application-Oriented FPGA Testing. In: *Proceedings of the 23rd Euromicro Conference on Digital Systems Design*. Kranj, Slovenia, 2020.

Remaining Publications of the Author Relevant to the Thesis

[A.5] Hülle, R. *The State-of-the-Art of Logical Circuits Automated Test Pattern Generation and Output Response Compaction.* Ph.D. Minimum Thesis, Faculty of Information Technology, Prague, Czech Republic, 2017.

[A.6] Hülle, R.; Fišer, P.; Schmidt, J. Generování testu pro prostředky vestavěné diagnostiky. In: *Počítačové Architektury & Diagnostika PAD 2016 - Sborník příspěvků.* Brno, Czech Republic, 2016.

[A.7] Hülle, R.; Fišer, P.; Schmidt, J. Generování testu s nulovým maskováním poruch. In: *Počítačové architektúry & diagnostika PAD 2017 - Zborník príspevkov.* Bratislava, Slovakia, 2017.

[A.8] Hülle, R.; Fišer, P.; Schmidt, J. ZATPG: SAT-based ATPG for Zero-Aliasing Compaction. In: *Proceedings of the 6th Prague Embedded Systems Workshop.* Prague, Czech Republic, 2018.

[A.9] Hülle, R.; Fišer, P.; Schmidt, J. PBO-Based Fault Selection for Compact Test Generation. In: *Proceedings of the 7th Prague Embedded Systems Workshop.* Prague, Czech Republic, 2019.