SAT-based ATPG for Zero-Aliasing Compaction

Robert Hülle, Petr Fišer, Jan Schmidt
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
Email: {hullerob, fiserp, schmidt}@fit.cvut.cz

***Abstract*—Aliasing in the test response compaction is an important source of fault coverage loss. Methods to avoid the aliasing generally require modification of the compactor to some extent. This can lead to a higher compactor complexity and consequently to higher area overhead, longer signal propagation delays, etc.**

We propose a novel method, the Zero-aliasing ATPG (ZATPG), which is able to reduce the aliasing without need of designing new compactors. ZATPG works by augmenting the SAT-based ATPG process to constrain test pattern generation to produce no aliasing in the compactor. The method is general enough to be applicable to any compactor design.

We demonstrate our method on a LFSR-based MISR compactors, using the Single Stuck-At fault model. Our method is able to find a test with zero aliasing and complete fault coverage for smaller compactors than conventional, unguided ATPG. Thus, the area overhead of the compactor can be reduced, while the complete fault coverage is preserved.

***Index Terms*—ATPG, response compaction, aliasing, stuck-at fault, SAT, zero-aliasing, LFSR, MISR.**

I. Introduction

In digital circuit testing, there are several contradicting considerations. One such consideration is the test length, and consequently the test application time and cost. On the other side there is a need to create such a test, that would detect all of the most likely defects in the circuit which could cause the circuit to fail. To save the test application time and cost, additional circuitry dedicated to testing can be added to the circuit design.

Typical testing environment, as is conceptually depicted in Figure 1, includes a Test Pattern Generator (TPG), a Circuit Under Test (CUT), a spatial compactor, a temporal compactor and a Comparator. The TPG has a role of applying test patterns to a CUT. It can be implemented in a circuitry entirely or it can be fed by testing data from an external test equipment (ATE). The spatial compactor is located at the outputs of the CUT and is responsible for reduction of the number of signal lines. Its outputs are fed to the temporal compactor, that is compacting individual test responses to a single value, the *signature*.

The reduction of response data volume can lead to aliasing, when the response of a faulty circuit is mapped to the same signature as the response of a fault-free circuit. Aliasing is thus one of sources of test coverage loss.

Aliasing in spatial compaction happens when a fault which is sensitized on the functional outputs of the CUT is not sensitized to the outputs of the spatial compactor. An example is a compactor that combines two signal lines with an AND gate. If one input is set to value 0, a faulty value on the other input is masked. It is possible to find a spatial compactor with zero aliasing for a given test and CUT [1], [2]. It is also possible to construct the compactor for a given CUT and a newly generated test [3].

Aliasing in temporal compaction means that after application of all test patterns and compaction of all test responses, the resulting signature is identical to the signature of a fault-free circuit. One important difference from aliasing in the spatial compaction is that the aliasing can be introduced at a later time by a test pattern that is used to test another fault.

Methods to reduce aliasing in temporal compactors include designing compactors with lower aliasing probability [4], [5] or manipulating the output response, so it can be checked by specially designed compactors [6]–[8].

Decreasing the aliasing probability in the compactor has the drawback of an increase of the compactors complexity. This can introduce higher area overhead, longer critical paths, higher energy dissipation, etc.

Manipulating the output response has the drawback of the need to design a new response compactor. This prevents from using this method when there is no control over the design of the compactor.

We propose a method to achieve lower aliasing and thus higher fault coverage by influencing the test generation in ATPG. Our method is able to work with existing compactors; no new compactor design needs to be created.

The paper is structured as follows: Section II gives a basic overview of the topic and existing methods. Section III describes our method to solve problems described above. Section IV documents our experiments and evaluation of our method. Section V draws a conclusion to the paper.

II. Preliminaries and Related Work

A. Spatial Compaction

The spatial compactor is a *combinational circuit* attached to the CUT outputs to reduce their count, while trying to propagate all fault syndromes to its outputs.

Parity trees (XOR trees) have a property of always sensitizing a fault through one input, irrespective of the logic value on the other input. This makes it a particularly popular design of space compactors [2], [9]. All faults that are sensitized on an odd number of outputs are propagated through the compactor and thus they are covered. It is shown in [10] that

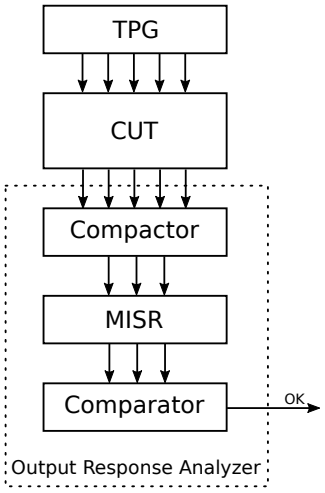

Fig. 1. Example of a typical test environment.

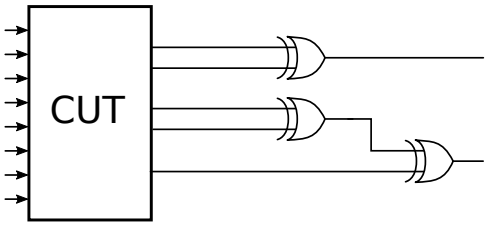

Fig. 2. Example of a simple space compactor with a XOR trees.

most of faults (at least in benchmark circuits) can be odd-sensitized and the compactor can be augmented to propagate the remaining even-sensitized faults by inserting additional observation points to the CUT and using multiplexers in the parity tree. Another approach to designing a zero-aliasing space compactor is partitioning of the CUT's primary outputs (POs) to multiple parity trees, so that a fault is always odd-sensitized in at least one parity tree [2].

It is also possible to design zero-aliasing space compactors using other elementary gates than XOR gates. In [3], a space compactor is iteratively constructed by checking for aliasing in a precomputed test and employing an ATPG to try to find a new test pattern for the aliased fault. This method is further extended by allowing XOR gates and eliminating the need for an ATPG in [1].

B. Temporal Compaction

The temporal compaction, similarly as the spatial compaction, is a method of reducing the volume of test response data that needs to be checked against correct response. While the spatial compaction reduces the size of responses to individual test patterns, the temporal compaction is reducing the volume of response data during the entire test. This is achieved by combining responses from several consecutive test patterns in a *sequential circuit*, to produce one or several combined responses, the *signature*.

The aliasing in this kind of compactor is harder to suppress and reason about, partly because of more degrees of freedom

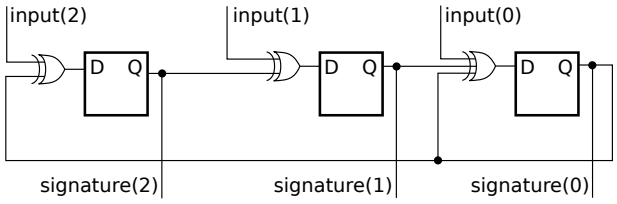

Fig. 3. Example of a LFSR-based MISR with characteristic polynomial $P(x) = x^3 + x + 1$

in both response compaction and test generation. A plethora of temporal compactor designs was published, one example of a popular design is the Multiple-Input Signature Register (MISR) and the Linear Feedback Shift Register (LFSR). The LFSR and LFSR-based MISRs are simple circuits and when correctly used, they provide a long period and low aliasing probability. An example of LFSR-based MISR is in Figure 3.

The probability of aliasing in the LFSR-based MISR with a primitive polynomial is 2^{-n}, that is under the assumption of random input values. Test patterns and especially test responses are however not random. The aliasing probability depends on the CUT and the test, and it can be significantly higher [4].

Lowering the aliasing probability in the compactor can be made by exerting a partial control over the test sequence to produce periodic [6] or alternating [7] output responses. These responses can be efficiently checked by a simple circuit. This method, however, requires a design and usage of an appropriate compactor.

A method to achieve zero-aliasing compaction for single-output circuits [8] is by augmenting a LFSR to produce a periodic quotient. The periodicity of the quotient is checked in addition to computing the signature as the reminder of polynomial division in the LFSR.

In all these methods, the test generation is independent of the response compaction. In some methods, the test sequence is manipulated to achieve reduced or zero aliasing.

In this paper we propose a novel method to decrease or eliminate aliasing by constraining the process of finding a test sequence for a given circuit and response compactor. That is, by eliminating the aliasing directly in the ATPG during the test patterns generation.

C. Automated Test Pattern Generator

1) Structural ATPGs: Traditionally, the problem of test generation is solved by structural ATPGs that are based on the D-algorithm [11], such as PODEM [12], FAN [13], and SOCRATES [14].

These algorithms are generally efficient for most faults, there is however a class of faults that are hard for structural ATPGs. Finding a test pattern for these faults is usually given up after reaching a backtracking limit. Other problematic class of faults are redundant (undetectable) faults. Proving that a fault is redundant is also computationally expensive.

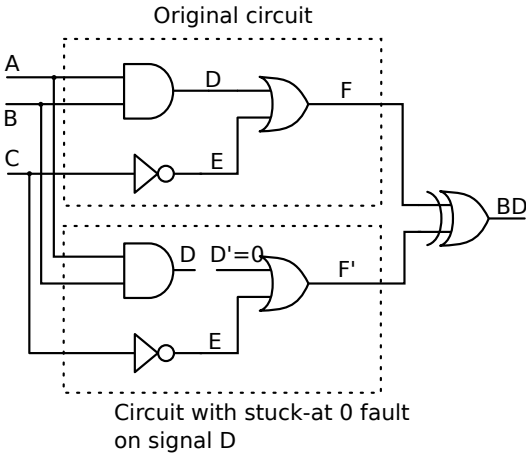

Fig. 4. Example of a conceptual circuit for modeling fault in CUT.

2) SAT ATPG: The Boolean satisfiability-based ATPG (SAT-ATPG) works by applying the Tseitin transformation [15] to transform structural description of a circuit to a representation in a Conjunctive Normal Form (CNF). This allows to create a conceptual circuit illustrated in Figure 4. This conceptual circuit is composed of a fault-free circuit and a duplicated circuit with a tested fault modeled. The fault is detected when the output of a faulty circuit differs from the output of the fault-free circuit. That is expressed in the conceptual circuit by combining the respective outputs with a XOR gate. We then search for such values at the primary inputs (PIs), for which the output signal has logical value 1. This is done by solving the transformed CNF description by a SAT solver.

In practice, only a part of the CUT is transcribed to CNF; we need to consider only the input cones of POs that are in the output cone of the tested fault. Additionally, only the output cone of the fault needs to be duplicated, the rest of the circuit can be shared with the fault-free circuit.

With the advent of modern SAT solvers, such as MiniSAT [16], that are efficient in solving instances seen in the ATPG domain, SAT-based ATPGs are not only feasible, but they provide several advantages over structural ATPGs [17]. The main advantage is the robustness of SAT solvers, that is, a SAT-ATPG is able to efficiently find test patterns for all faults, including faults that are hard for structural ATPGs. Redundancy of a fault is also proved efficiently.

There are of course disadvantages, the main one being the performance. SAT-ATPGs are noticeably slower than structural ATPGs for most of easily tested faults, thus in practice a combination of structural and SAT ATPGs is usually employed. There is an ongoing work on both SAT solvers and SAT ATPGs that has a potential to close the gap between structural and SAT ATPGs [18], [19].

III. Proposed Method

Conventional SAT-based ATPG searches for a test pattern without considering further processing of the response to the generated pattern. The only requirement is that the tested fault is detected on combinational outputs of the CUT. Each and every test pattern and faults that are detected are treated by their own. There is no notion of test ordering or response compaction.

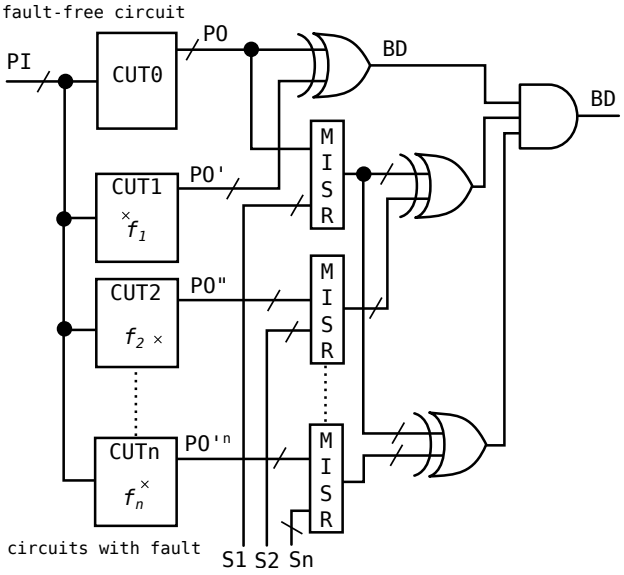

Fig. 5. Conceptual circuit for finding non-aliasing test vector.

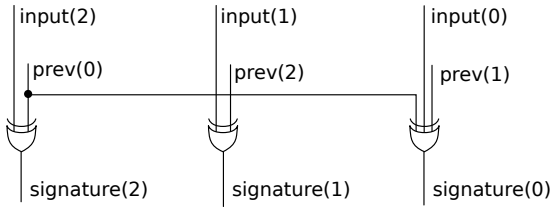

Fig. 6. Example of unrolled temporal compactor from Figure 3

A. Augmenting the SAT-ATPG

We augment the ATPG process to generate a test with zero aliasing in the compaction of test responses. We achieve that by adding additional constraints to the SAT instance representing the tested fault that would prevent aliasing. We do this by expanding the conceptual circuit as portrayed in Figure 5. We call this method *ZATPG*.

The expanded conceptual circuit consists of a fault-free circuit *CUT0* and a circuit *CUT1* with the tested fault f_1 as described in Figure 4. To this circuit, we then add constraints preventing aliasing in selected faults f_2 through f_n. These constraints take a shape of additional replicas of the CUT, each with its corresponding fault modeled. The output of these circuits is then constrained to differ from the value that would cause the aliasing.

The aliasing happens *after* application of test pattern that is being generated. That means that we need to know the future state of the temporal compactor *during* test pattern generation. This state is however dependent on the test pattern.

The method of unrolling can be used to express the future state of the compactor during the test pattern generation. We can do that by extracting the combinational part of the compactor. Simulating the compactor is then part of SAT-solving during the search for a test pattern. An example of an unrolled compactor from Figure 3 is depicted in Figure 6.

In the conceptual circuit from Figure 5, this unrolled circuit is shown as two MISR blocks (note that these are combinational circuits, not actual MISRs). The previous internal state of the compactor is represented by vectors *S1* and *S2* for the fault-free and faulty circuit respectively. Outputs of these compactors represent the next state (partial signature) for the fault-free circuit and the circuit with the fault f_2. We add a constraint that the next state of the compactor differs for the fault-free circuit and the circuit with the fault f_2 by connecting the outputs of the combinational part of the compactor by XOR and setting the output to logical 1.

This extended conceptual circuit can be transformed to the CNF is the same way as with the conventional SAT-ATPG. Due to the nature of CNF, there is another way to transform the conceptual circuit. We can use the CNF from transformation of classical conceptual circuit and add new parts of the extended circuit by simply concatenating new clauses to the CNF. This includes clauses describing circuits *CUT2–CUTn*, the unrolled compactor, and constraints forcing the outputs to be non-zero vectors. The last AND gate in the Figure 5 needs not to be modeled due to the clauses already being in logical conjunction.

In practice, only the output cone of each considered fault is duplicated in the CNF representation, the rest of the circuit, and thus clauses in the SAT instance, is shared with the fault-free circuit. It has been shown on a problem of multi-targeted faults, that SAT solvers are robust enough for this approach to be feasible [20]. We expect that the SAT solver will be efficient in finding the constrained test pattern or proving that no such pattern exists and the aliasing is unavoidable (for fault f_1 in the current step). Not every fault in every step needs to be constrained against aliasing. Only if we detect aliasing of the fault f_2 after simulation of test pattern for the fault f_1, we run ATPG again with the additional constraint. This speeds up the ATPG if aliasing is not likely to occur.

B. Simplification for Linear Compactors

For linear temporal compactors, we can use the *superposition principle* to separate their state and input to an *error component* and a *correct component*. We can then use these components 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 the compactor we have chosen a LFSR-based MISR from Figure 3. 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

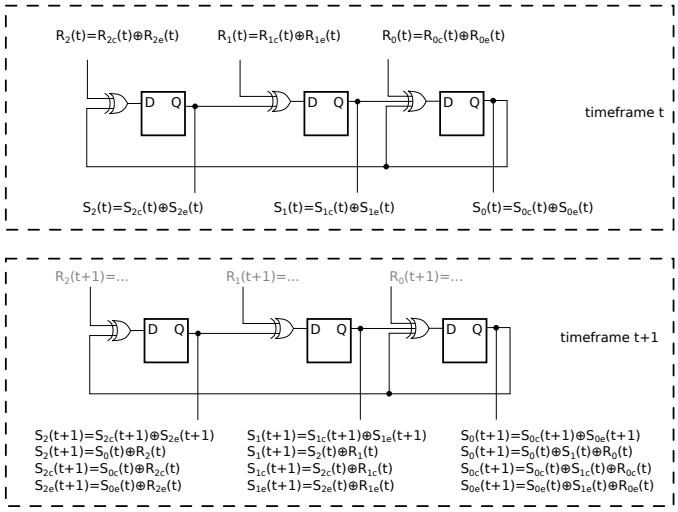

Fig. 7. Example of the state computation of a linear compactor.

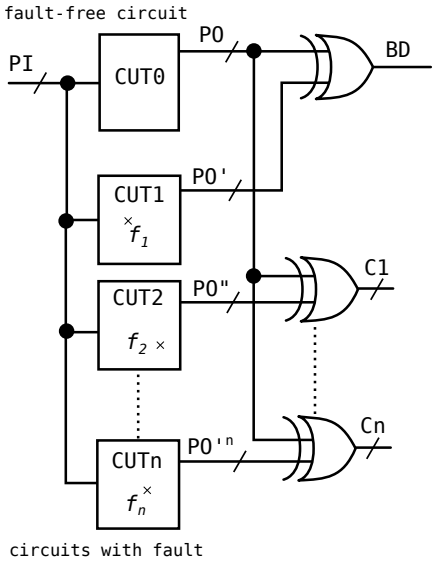

Fig. 8. Conceptual circuit for finding zero-aliasing test pattern, simplified for linear compactors.

for the fault-free circuit, and S_e is the *error difference* (error component) for the faulty circuit.

The next state of MISR is computed as $S^{(t+1)} = F(S^{(t)}) \oplus R^{(t+1)}$, where F represents the step function of the MISR. For example, the MISR from Figure 3 has a step function $F((s_0, s_1, s_2)) = (s_0 \oplus s_1, s_2, s_0)$.

Due to the linearity of F, where $F(a \oplus b) = F(a) \oplus F(b)$, we can compute the *correct* and the *error* states separately as $S_c^{(t+1)} = F(S_c^{(t)}) \oplus R_c^{(t+1)}$ and $S_e^{(t+1)} = F(S_e^{(t)}) \oplus R_e^{(t+1)}$, respectively. This is illustrated in Figure 7

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 the fault was detected and the zero vector means that it was not detected.

In our method, in order to prevent aliasing at the end of the test, we need to prevent aliasing in every test step since the fault was detected. In other words, we are constraining the ATPG to not allow $S^{(t+1)}$ to be a zero vector if it was a non-zero vector in the previous step.

We can express the output response that would cause aliasing as $R_e = F(S_e^{(t)})$. Note that $F(S_e^{(t)}) = S_e^{(t+1)}$ iff R_e is zero. We can thus precompute the *error* 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 conceptual circuit can be seen in Figure 8.

C. Algorithm

The main execution loop of our algorithm expressed in a pseudocode follows:

```
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
```

At the beginning, the list of uncovered faults F_u is initialized by all testable faults F (line 3). This is because no fault was detected, when the test patterns sequence P is empty (line 2).

After the initialization, the algorithm works by repeatedly (lines 4–24) iterating over the list of uncovered faults F_u (lines 6–23, 7).

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 (line 13). If the number of aliased faults $|F_a|$ is lower than the number of newly detected faults $|F_d|$ and the $|F_a|$ is not higher than parameter M, the pattern p is accepted.

In the case of the above-mentioned condition to accept the pattern p not being met, the set of aliased faults F_a is analyzed and a set of constraints C is constructed (line 20) and a new pattern p is generated with additional constraints. This is continued until the condition is met or no pattern can be generated due to additional constraints.

In the case of the test pattern p being 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 again added 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).

The algorithm makes use of following procedures:

1) SIM: The procedure *SIM* simulates the compactor *error state* for every fault after application of a partial test sequence P, that was generated up to this point, and the new test pattern p. We are caching the *error state* of all faults after the test sequence P. For each pattern p, only one simulation step needs to be performed.

This procedure identifies faults that are newly detected by p and those that are aliased. 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 the zero vector. The sets F_a and F_d of aliased and detected faults are then returned.

2) ATPG: The procedure *ATPG* generates a test pattern for a fault f and a set of constraints C. It creates a conceptual circuit for detecting the fault f while preventing aliasing of faults described in C. This conceptual circuit is then transformed to a CNF description and solved by a SAT solver.

To prevent aliasing, for each fault f_a from C, its output cone is duplicated, as it is for fault f. In contrast to fault f, however, the difference of its outputs from the CUT is not constrained to be non-zero, but to be different from the vector described in C that would cause aliasing.

Generated test pattern, if any, is then returned.

3) CONSTR: The procedure *CONSTR* computes constraints which prevent aliasing of given faults.

For example, to prevent aliasing of fault f_2, constraints are computed in the following way. First, the compactor with the *error component* of partial signature is simulated under assumption that no aliasing occurred (input vector is zero). From the resulting state (partial signature), the *error component* of the response that would cause aliasing is computed.

This response is then added to the CNF representation of the conceptual circuit from Figure 8 as a response to fault f_2 that is not allowed, for it would cause aliasing to occur in the compactor. This is done by appending a single clause C_2 with inverted variables, that forces the *error component* of the response to be different.

IV. Experiments

A. Experimental setup

In our experiments, we are using benchmark circuits from ISCAS'85 [21] and some circuits from ITC'99 [22].

For each circuit, we consider several sizes s of the temporal compactor, LFSR-based MISR. The MISR always has a primitive characteristic polynomial.

As a spatial compactor, we are using a parity tree (XOR tree) design, with multiple outputs. The number of spatial compactor outputs matches the width of the MISR (s). The spatial compactor is constructed as s disjoint XOR trees, while zero aliasing is guaranteed by simulation and possible restructuring in case of aliasing occurrence.

For each circuit, we append the spatial compactor to its outputs and then work with this combination as if it was one circuit. This means that the number of testable faults in one circuit varies due to the additional faults in the compactor.

In the experiment, we compare the aliasing and fault coverage achieved by a conventional ATPG with aliasing and fault coverage obtained by our method (ZATPG). By a conventional ATPG, we mean an ATPG that does not take the compactor into account during test generation.

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.

For both ZATPG and conventional ATPG [23] we use the SAT-based ATPG which uses MiniSAT [16] as a SAT-solver.

B. Experimental results

1) Fault coverage: A summary of the fault coverage achieved by conventional ATPG and by ZATPG is shown in Table I. The columns "ZATPG" show the 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 ∞.

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

2) Aliasing: The targeted aliasing for ZATPG is zero, but as can be seen from Table II, this cannot always be achieved. The Table shows how the choice of the parameter M influences the results. The column "aliasing" shows the aliasing in the compactor, the column "coverage" shows the total test coverage, including aliasing. The ZATPG was not able to find any test vector that would cause zero aliasing. We amended this by relaxing the requirement for zero-aliasing. This relaxation is controlled by the parameter M, which limits how many faults can be aliased by a new vector, but only if more new faults are covered.

3) Robustness: The coverage achieved by ZATPG is dependent on the order of faults selection. We have examined the robustness of ZATPG on selected circuits by changing the ordering of faults with random permutations. The ordering of faults changes the order in which faults are selected for test pattern generation (line 7 of algorithm). It does not change the manner of test pattern generation itself.

Distributions of coverage for circuits c2670 and c1355 [21] are in Figure 9, these were computed for the smallest MISR size where default fault ordering leads to the complete fault coverage. For the circuit c1355, 2007 measurements were made and for the 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 the circuit c1355, in the circuit c2670 this trend is not so clear. For these circuits, 6.8 % and 5.5 % of orderings, respectively, lead to a 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) Compactor size: The size of the smallest MISR for which a complete fault coverage was found is in Table III. 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 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 the aliasing is naught. This is needed due to the small robustness of both ATPGs.

Our algorithm needs a significantly smaller compactor, 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.

5) Algorithm computation time: The comparison of the computation time of the ZATPG and a conventional SAT-ATPG can be seen from Table III. The computation time was measured for the MISR sizes, where each algorithm achieved zero aliasing.

Figure 10 illustrates the dependence of the ZATPG computation time on the MISR size. Is 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 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

TABLE I
FAULT COVERAGE FOR CONVENTIONAL AND AUGMENTED ATPG

MISR size		2		3		4		5		6		7		8	
circuit	faults	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]	atpg [%]	zatpg [%]
b04	2846	78.71	79.20	91.10	87.80	94.65	95.11	97.99	97.89	98.80	99.68	99.54	99.96	99.86	99.96
b11	2382	78.30	75.99	89.37	92.02	94.58	96.09	98.02	98.86	98.78	99.71	99.41	99.96	99.87	100
c499	970	82.37	70.10	90.70	86.78	95.65	77.23	97.82	90.87	98.86	96.57	98.85	99.90	99.79	100
c880	1582	79.33	83.19	90.95	92.34	93.79	98.86	97.65	100	98.48	100	99.30	100	99.24	100
c1355	2618	79.76	75.63	90.56	86.05	95.45	92.85	97.01	97.09	99.27	100	99.65	99.73	99.77	100
c1908	2581	78.61	72.99	92.25	84.41	96.39	89.99	98.29	94.95	99.03	98.33	99.69	99.88	99.65	99.96
c2670	3613	79.91	73.04	92.69	90.78	94.54	93.85	98.11	98.45	98.61	98.00	99.83	100	99.53	100
c5315	7964	[a]		89.45	90.29	94.75	95.01	97.01	98.52	98.53	99.90	99.57	99.82	99.90	100
c7552	10921	[a]		90.62	88.42	94.85	92.97	97.55	96.35	98.28	98.94	99.32	99.95	99.65	100

[a] Empty cells indicate that no zero-aliasing spatial compactor was available.

TABLE II
ALIASING IN AUGMENTED ATPG

	M	3		5		10		50		∞		ATPG
circuit	MISR size	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	aliasing [%]	coverage [%]	coverage [%]
c499	6	0.31	95.01	0.21	99.06	0.10	99.90	0.00	96.57	0.00	96.57	99.38
c880	5	0.13	98.98	0.51	97.72	0.00	100	0.00	100	0.00	100	97.78
c1355	6	0.04	92.87	0.15	93.72	0.19	97.16	0.00	100	0.00	100	98.51
c2670	6	0.08	94.09	0.08	96.50	0.00	100	0.36	98.00	0.36	98.00	99.06
c7552	7	0.00	99.67	0.00	99.83	0.00	99.86	0.00	99.95	0.00	99.95	99.51

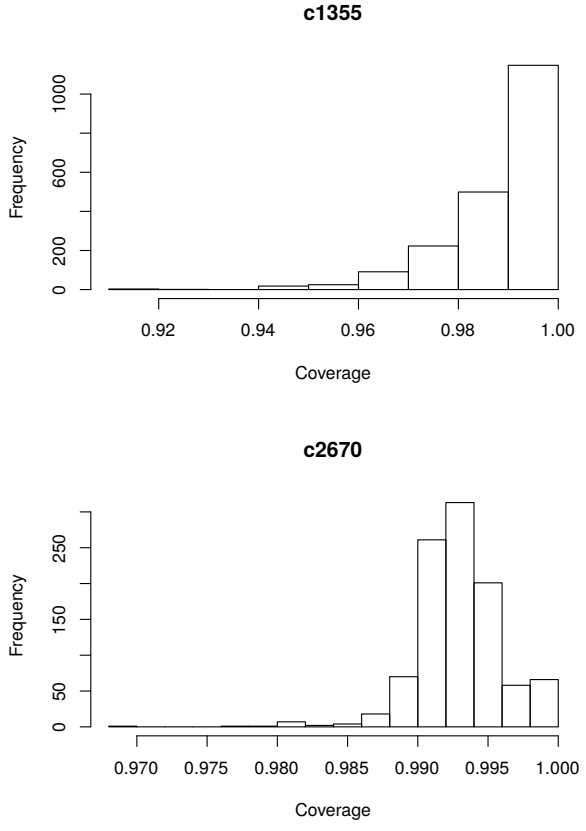

Fig. 9. Robustness of ZATPG: test coverage for random fault ordering.

TABLE III
MINIMAL SIZE OF MISR WITH ZERO-ALIASING TEST

		MISR size		run time [s]	
circuit	faults	ATPG	ZATPG	ATPG	ZATPG
b04	2846	11	9	17	73
b11	2788	10	8	12	41
c499	970	12	8	3	5
c880	1582	12	5	3	16
c1355	2618	10	6	6	53
c1908	2581	12	9	10	39
c2670	3613	12	7	77	297
c5315	7964	14	8	283	1299
c7522	10921	12	8	280	2452

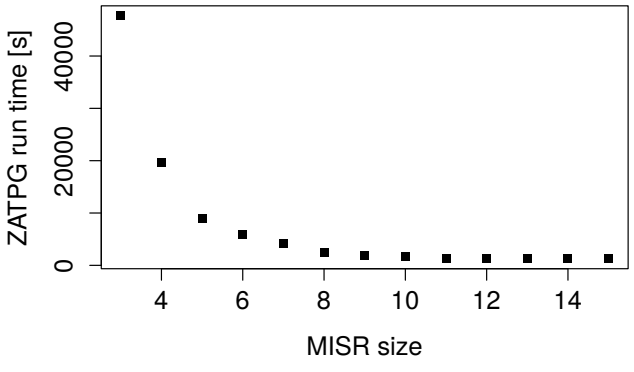

Fig. 10. Runtime of the ZATPG algorithm depending on the MISR size for the c7552 benchmark circuit.

ZATPG is necessarily running more fault simulations of the CUT.

V. Conclusion

A modified SAT-based ATPG process, *ZATPG* for finding a test with zero aliasing in any given temporal compactor is presented. A simplified process for linear space compactors is also presented and its properties are shown on the example of a LFSR-based MISR.

Our algorithm generated a test with comparable coverage to that of a conventional ATPG. The coverage of ZATPG is slightly lower for MISRs of small size. That being said, the coverage rises with the size of MISR faster than with conventional ATPG.

For benchmark circuits, we achieved complete fault coverage with a MISR of smaller size than the conventional ATPG. This is due to ZATPG being guided towards zero-aliasing, whereas in the the test produced by other ATPG it is a product of aliasing probability in the compactor.

The coverage of ZATPG for MISRs depends on the order in which faults are processed. In this respect, ZATPG is not robust. It does however overtake this disadvantage with an increasing compactor size.

Our further work would include creating a heuristics for faults selection, or fault ordering. Another heuristics would be deciding which faults can be aliased and which should not be. Together, these heuristics could improve ZATPG's robustness.

This algorithm could also greatly benefit from modern SAT-ATPG techniques, such as dynamic clause learning [19] and dynamic clause activation [18].

Acknowledgment

This work was partially supported by the grant GA16-05179S of the Czech Grant Agency, "Fault Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features" (2016-2018).

This research has been in part supported by CTU grant SGS17/213/OHK3/3T/18.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"

References

[1] Y. Liu and A. Cui, "An efficient zero-aliasing space compactor based on elementary gates combined with XOR gates," in *IEEE/ACM International Conference on Computer-Aided Design*, 11 2013, pp. 95–100.

[2] K. Chakrabarty, "Zero-aliasing space compaction using linear compactors with bounded overhead," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 5, pp. 452–457, 08 2002.

[3] B. Pouya and A. Touba, Nur, "Synthesis of zero-aliasing elementary-tree space compactors," in *IEEE VLSI Test Symposium*, 1998, pp. 70–77.

[4] P. D. Hortensius, R. D. McLeod, and H. C. Card, "Cellular automata-based signature analysis for built-in self-test," *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1273–1283, Oct 1990.

[5] K. Pradhan, D. and K. Gupta, Sandeep, "A new framework for designing and analyzing BIST techniques and zero aliasing compression," *IEEE Transactions on Computers*, vol. 40, no. 6, pp. 743–763, 1991.

[6] G. Edirisooriya and P. Robinson, John, "Test generation to minimize error masking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 4, pp. 540–549, 04 1993.

[7] 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*, Feb 1998, pp. 180–184.

[8] K. Pradhan, D., M. Reddy, Sudhakar, and K. Gupta, Sandeep, "Zero aliasing compression," in *Fault-Tolerant Computing: 20th International Symposium*, 06 1990, pp. 254–263.

[9] 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*, vol. 52, no. 5, pp. 1363–1380, Oct 2003.

[10] K. Chakrabarty and J. P. Hayes, "Test response compaction using multiplexed parity trees," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 11, pp. 1399–1408, Nov 1996.

[11] J. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278–291, July 1966.

[12] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215–222, March 1981.

[13] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, Dec 1983.

[14] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126–137, Jan 1988.

[15] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning*, ser. Symbolic Computation, J. Siekmann and G. Wrightson, Eds. Springer Berlin Heidelberg, 1983, pp. 466–483.

[16] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds. Springer Berlin Heidelberg, 2004, vol. 2919, pp. 502–518.

[17] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[18] R. Drechsler, D. Tille, and S. Eggersglüß, "Speeding up SAT-based ATPG using dynamic clause activation," in *Asian Test Symposium*, 11 2009, pp. 177–182.

[19] R. Drechsler and S. Eggersglüß, "Increasing robustness of SAT-based delay test generation using efficient dynamic learning techniques," in *14th IEEE European Test Symposium*, May 2009, p. 6.

[20] S. Eggersglüß, R. Krenz-Baath, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *15th IEEE Design and Diagnostics of Electronic Circuits and Systems*, April 2012, pp. 230–235.

[21] 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)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

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

[23] R. Hülle, P. Fišer, J. Schmidt, and J. Borecký, "SAT-ATPG for application-oriented FPGA testing," in *15th Biennial Baltic Electronics Conference*, 10 2016, pp. 83–86.