# Reducing Output Response Aliasing Using Boolean Optimization Techniques

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*—In digital circuit testing, output response compaction can have a significant impact on fault coverage. The loss of fault coverage is caused by aliasing in the output response compaction. Classical approaches to reducing (eliminating) fault aliasing are based on modifications of the compactor design or modifying precomputed test sequence. In this paper, we propose a completely different approach based on a dedicated test pattern generation algorithm. The algorithm generates a test sequence with minimal aliasing for targeted faults. As the generated test sequence is tailored to given static and dynamic compactor structures, any response compactor can be used without a change in the design. We expand on our previous work, zero-aliasing ATPG, and incorporate pseudo-Boolean optimization techniques in the process.**

**The algorithm is evaluated using an LFSR-based MISR on a selection of benchmark circuits. A comparison with a state-of-the-art ATPG process without anti-aliasing measures is drawn.**

***Index Terms*—ATPG, multiple-target test generation, SAT, pseudo-Boolean optimization, fault aliasing, response compaction, test compaction, LFSR**

## I. Introduction

The low test length and the low test application time are essential in both after-manufacture and in-deployment testing. The *Desing for testability* (DfT) techniques are used to simplify and speed-up test generation, test application, and to increase the fault coverage.

*Output response compaction*, a DfT technique, can be used to decrease the amount of read-out data during testing. This is useful both for *Built-In Self-Test* (BIST) and testing with external equipment. A *fault aliasing* can occur during the response compaction, decreasing the fault coverage. The output response compaction can be divided into *spatial* (static) and *temporal* (dynamic) parts.

*Spatial response compaction* is performed by a combinational circuit. It reduces the amount of data produced by a single test pattern, i.e., the output response bit width. Consecutive test patterns do not influence each other. The aliasing can be prevented in this compactor by its proper design [1]–[4].

*Temporal response compaction* reduces the number of response patterns and creates a signature of the test response. This is usually done by a *Multiple-Input Signature-Register* (MISR) – a sequential circuit that accumulates the output responses over multiple clock cycles in its internal state. One example of a MISR is a *Linear Feedback Shift Register* (LFSR) with parallel inputs.

Preventing aliasing in a temporal compactor is more complex because consecutive test patterns and output responses influence each other. Usual methods to prevent aliasing are manipulating the compactor design [5]–[9] or manipulating preexisting test pattern sequence [10].

A different approach to reducing the aliasing in the temporal response compaction is by constraining the ATPG algorithm to prevent aliasing in the compactor [11]. In this approach, the design of the compactor must be known during test generation. The ATPG then generates a test that is not aliased in the temporal (sequential) compactor. While this approach can achieve zero aliasing even for small and simple compactors, the trade-off is in the resulting test length and test generation time. Additionally, the generated test cannot be processed further, e.g., by a *static test compaction* [12]–[14].

*Dynamic test compaction* [15]–[20] is, in contrast to static compaction, performed during test pattern generation, on the incomplete test set. It is also beneficial for the ATPG to utilize extra information from the compaction while generating the next test pattern. Typical dynamic compaction tries to use the unspecified bits of generated test patterns to target additional faults. Other methods of dynamic compaction include *multiple-target test generation* (MTTG) [17]–[19] and *optimization-based MTTG* (OTG) [20] that can target multiple faults at once. Dynamic compaction can produce a more compact test set than static compaction, at the price of increased computational complexity. In this paper, we adopt modern dynamic compaction principles to the zero-aliasing test generation process. As a result, the test length is significantly reduced, compared to the original ZATPG algorithm.

In particular, the main contributions of this paper are as follows:

- Dynamic test compaction applied in zero-aliasing SAT-based ATPG
- Optimization-based fault selection for anti-aliasing
- Evaluation and comparison with a state-of-the-art ATPG without anti-aliasing measures

The paper is structured as follows:

In Section II, there is a brief overview of recent work in the field as well as an introduction to tools being used. In Section III, the newly proposed method is described. The experimental

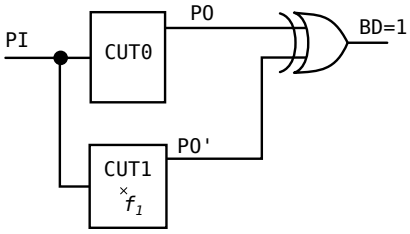


Figure 1: Miter: conceptual circuit for SAT-based ATPG

results are presented in section IV. Conclusions and discussed possibilities for future work are drawn in the last section V.

## II. Preliminaries

### A. SAT-ATPG

*SAT-based ATPG* is an ATPG that uses the Boolean satisfiability problem (SAT) and a SAT solver to generate test patterns [21].

Construction of a *miter*, a conceptual circuit, is one of the possible ways to construct an SAT instance and generate a test pattern. Illustrated for the fault $f_1$ in Figure 1, two replicas of the circuit under test are created. One for the original fault-free circuit (CUT0), and the second for a faulty circuit (CUT1). The fault $f_1$ is modeled in the replica CUT1. Inputs (PIs) of the two replicas are connected and the outputs (POs) are compared. The output signal $BD$ of the miter indicates whether a test pattern $p$ on the PIs detects the fault $f_1$ or not. Finding a test pattern is then a problem of finding an input vector that sets the output of the circuit to the value 1.

This problem is then translated to SAT by means of the Tseitin transformation [21], [22]. The generated SAT instance is usually in the *conjunctive normal form* (CNF) as that is the input form that SAT solvers use.

When the SAT instance is satisfiable, then the certificate (Boolean assignment of the variables) corresponds to the values of signals in the miter. The test pattern is then extracted as the values of the input signals.

### B. Multiple-Target Test Generation

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.

*Multiple-Target Test Generation* (MTTG) techniques [17]–[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.

### C. Optimization-Based MTTG

In the MTTG, a test pattern detecting all *selected* faults is found. 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 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 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.

### D. Reducing Aliasing in ATPG

The main idea behind the reduction of fault aliasing in an ATPG is constraining an SAT instance to not allow aliasing for selected faults (ZATPG) [11].

In this approach, the faults that were aliased by the previous (non-accepted) test pattern are selected for constraining (anti-aliasing). Constraining is done in the miter. In addition to generating a basic miter, as in Figure 1, the faults selected for anti-aliasing are appended as additional faulty replicas, similarly to the MTTG.

The aliasing happens in the temporal compactor, which is a sequential circuit. Aliasing can be detected by reading the internal compactor state only one clock cycle after the test pattern is applied to the inputs. As is illustrated in Figure 2, for the detection of aliasing during the SAT instance solving, the *unrolled temporal compactors*, i.e., their combinational parts, need to be appended to the outputs (POs) of each anti-aliasing replica. Outputs of these unrolled compactors are then constrained to differ from the fault-free compactor. The SAT solver is then forced to find a test pattern that would not alias these faults. The existence of such pattern is not guaranteed.

The requirement of zero aliasing for every test pattern is, however, too strong; instead, a relaxed constraint is used. The aliasing is allowed to occur, as long as the overall coverage is increased, i.e., the number of newly detected faults is higher than the number of aliased faults. If not, the aliased faults are added to the miter and the process repeats.

In short, the algorithm for generating a test pattern for a single fault is as follows. This procedure is performed for all faults in the fault list. First, a classical miter with the fault-free CUT (CUT0) and a selected fault $f_1$ (CUT1) is used to find a test pattern. Second, fault simulation is performed to identify the aliased faults. Third, if the test pattern does not increase the fault coverage, aliased faults ($f_2$, $\ldots f_n$) are added to the miter and a test pattern is generated again. The resulting miter is illustrated in Figure 2.

If no test pattern can be found for the currently tested fault, it is skipped and visited later – it is moved to the end of the fault list. For more details on this basic ZATPG procedure, see [11].

This approach leads to high test generation times and the length of generated test is also not optimized. Further, the *static test compaction* cannot be used, because the aliasing

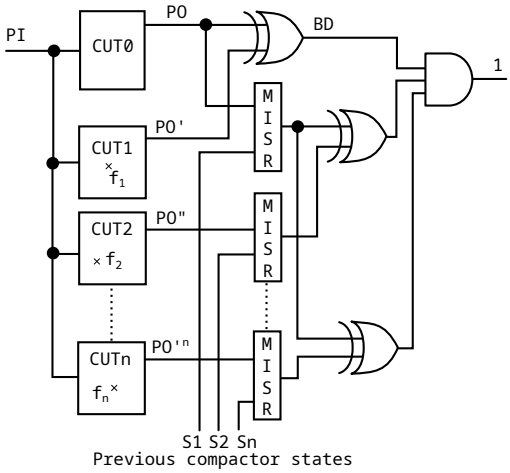


Figure 2: ZATPG: restricting fault aliasing

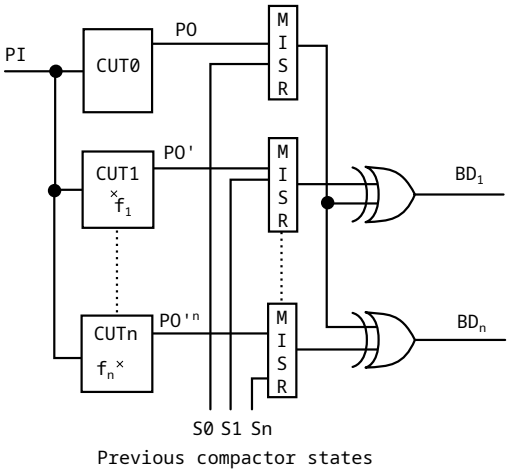


Figure 3: Miter for multiple targeted and anti-aliased faults.

(and zero-aliasing property of the test) is dependent on the exact test pattern sequence and its ordering. Therefore, *dynamic test compaction* must be used to reduce the number of test patterns. This was our main motivation for the proposed method described in the following section.

## III. The Proposed Method

In this section, we describe 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. The resulting proposed algorithm will be referenced as ZATPG-PBO.

### A. Minimizing Fault Aliasing

In our approach, we build on our previous algorithm, ZATPG [11]. 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 (see Section II-D).

To avoid such a situation, we use an optimization criterion to minimize the number of aliased faults. Therefore, 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 time. 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 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.

### B. 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. The faults are selected in the order of increasing *accidental detection index* ADI [23], i.e., faults that are less likely to be detected by an arbitrary pattern are selected first. 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.

### C. Miter and PBO Instance

The miter is constructed in a similar manner to the ZATPG algorithm (Figure 2), with the difference being 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 3 and Equation 1.

$$\sum_{i=1}^{n} BD_i = max. \tag{1}$$

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 selected for anti-aliasing. Because the difference outputs are not constrained, this test pattern will always exist, possibly decreasing the fault coverage. The final change in fault coverage is known only after fault simulation.

### D. The Algorithm Overview

**Algorithm 1** 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
```

A high-level view of the overall ZATPG-PBO algorithm is shown in Algorithm 1.

The algorithm is parametrized by the following variables:

| | |
|---|---|
| $M_f$ | default number of targeted faults |
| $M_{fm}$ | maximum of targeted faults |
| $M_i$ | acceptable improvement |
| $M_a$ | the maximum of anti-aliased faults |

The flow of the algorithm is as follows:

*1) Fault Generation:* On the line 3, a fault list for the circuit $C$ is prepared. This fault list is precomputed and ordered by the ADI [23]. We also use this step to remove redundant faults from the fault-list.

*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, new faults for $F_t$ are selected (re-targeted [24]) and the first $n\frac{M_f}{2}$ faults are skipped for every $(n)$ unsuccessful search.

*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) 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.

*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.

*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 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 $M_f$.

## IV. Experimental Results and Discussion

In this Section, the experimental results are presented. Achieved fault coverage for small-sized MISR is evaluated and compared to two ATPGs without anti-aliasing awareness, the Atalanta [25] ATPG and our implementation the OTG algorithm [24].

### A. Experimental Setup

The ZATPG-PBO algorithm was implemented in the Go programming language. The used PBO solver is the Minisat+ [26]. The OTG algorithm was likewise re-implemented in the Go language, according to the algorithm described in [24]. Experiments were run on the circuits from ISCAS'85 [27], ISCAS'89 [28], ITC99 [29], and EPFL [30] benchmarks. For sequential circuits, their combinational profiles were extracted. The used fault model is the single stuck-at fault.

An output response compactor was appended to the primary outputs of the circuits in the following way: first, an irredundant spatial output response compactor was generated. The compactor was appended to the primary outputs of the CUT. The spatial compactor was constructed as a collection of disjoint elementary-gates trees [2], [3]. Second, an LFSR-based MISR with a primitive polynomial was used as the temporal output response compactor. The number of outputs of the spatial compactor was chosen to be equal to the size of the used MISR.

The experiments were run on the national grid infrastructure, consisting of heterogenous computation nodes. As such, computation times of the algorithm are only approximative.

From our pilot experiments, we have chosen the following setup: The maximal allowed *targeted test* window is set to $M_{fm} = 100$. Default *targeted test* window is set to the value $M_f = 50$. The acceptable improvement is set to $M_i = 40$. The allowed *anti-aliasing window* is set to $M_a = 100$.

### B. Fault Coverage and Test Length

Fault coverage and test length for selected benchmark circuits are presented in Table I. Results for MISR sizes from 4 to 7 bits are selected, as the ZATPG-PBO generated a complete

Table I: Fault Coverage and Test Length

| MISR SIZE | | | 4 | | | | | | 5 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| algorithm | | | ZATPG-PBO | | OTG | | Atalanta | | ZATPG-PBO | | OTG | | Atalanta | |
| circuit | PO | faults | [%] | length | [%] | length | [%] | length | [%] | length | [%] | length | [%] | length |
| c499 | 32 | 990 | 83.33 | 14 | 95.35 | 52 | 96.16 | 53 | 92.42 | 30 | 97.07 | 52 | 98.18 | 53 |
| c880 | 26 | 1754 | 99.89 | 23 | 94.93 | 19 | 93.04 | 59 | 100.00 | 22 | 97.32 | 18 | 97.09 | 56 |
| c1355 | 32 | 2702 | 89.45 | 29 | 93.49 | 84 | 94.30 | 85 | 96.45 | 62 | 97.56 | 84 | 97.04 | 85 |
| c1908 | 25 | 3805 | 90.17 | 41 | 93.43 | 110 | 92.88 | 120 | 96.08 | 66 | 96.19 | 108 | 96.85 | 120 |
| c2670 | 140 | 4704 | | | - | | | | 99.57 | 47 | 97.66 | 47 | 95.73 | 70 |
| c7552 | 108 | 14253 | | | - | | | | | | - | | | |
| s832 | 24 | 1640 | 94.45 | 73 | 95.67 | 109 | 95.91 | 116 | 98.90 | 99 | 98.66 | 114 | 98.35 | 113 |
| s1238 | 32 | 2310 | 92.81 | 60 | 94.72 | 123 | 94.46 | 128 | 95.67 | 69 | 97.45 | 125 | 96.84 | 130 |
| b04 | 74 | 2553 | | | - | | | | 99.80 | 40 | 97.06 | 38 | 95.93 | 67 |
| b05 | 60 | 3671 | | | - | | | | 99.16 | 47 | 97.03 | 48 | 96.49 | 84 |
| b11 | 37 | 2227 | 97.80 | 51 | 94.57 | 54 | 94.79 | 73 | 99.55 | 50 | 96.32 | 50 | 97.75 | 68 |
| b12 | 127 | 4615 | 94.19 | 60 | 95.47 | 97 | 93.24 | 149 | 99.37 | 89 | 97.55 | 98 | 97.68 | 150 |
| cavlc | 11 | 3052 | 96.76 | 125 | 93.64 | 162 | 94.00 | 136 | 98.10 | 124 | 97.38 | 157 | 96.43 | 136 |
| dec | 256 | 1840 | 54.24 | 14 | 96.30 | 256 | 95.65 | 256 | 49.62 | 12 | 98.04 | 256 | 97.93 | 256 |
| i2c | 142 | 5014 | | | - | | | | 99.78 | 54 | 97.15 | 61 | 96.93 | 119 |
| priority | 8 | 2861 | 99.62 | 26 | 94.48 | 18 | 94.34 | 33 | 100.00 | 24 | 98.18 | 16 | 96.64 | 32 |

| MISR SIZE | | | 6 | | | | | | 7 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| algorithm | | | ZATPG-PBO | | OTG | | Atalanta | | ZATPG-PBO | | OTG | | Atalanta | |
| circuit | PO | faults | [%] | length | [%] | length | [%] | length | [%] | length | [%] | length | [%] | length |
| c499 | 32 | 990 | 99.90 | 56 | 98.38 | 52 | 98.48 | 53 | 100.00 | 54 | 99.70 | 52 | 99.39 | 53 |
| c880 | 26 | 1754 | 100.00 | 24 | 99.09 | 20 | 98.86 | 58 | 100.00 | 20 | 99.03 | 18 | 99.20 | 59 |
| c1355 | 32 | 2702 | 100.00 | 88 | 98.41 | 84 | 98.67 | 85 | 100.00 | 86 | 98.93 | 85 | 99.48 | 85 |
| c1908 | 25 | 3805 | 96.48 | 63 | 98.63 | 110 | 99.00 | 117 | 99.97 | 109 | 99.50 | 109 | 99.66 | 118 |
| c2670 | 140 | 4704 | 100.00 | 56 | 98.94 | 48 | 96.19 | 70 | 99.74 | 41 | 99.36 | 45 | 97.53 | 71 |
| c7552 | 108 | 14253 | 99.90 | 55 | 98.79 | 48 | 98.61 | 151 | 100.00 | 50 | 99.22 | 47 | 99.23 | 139 |
| s832 | 24 | 1640 | 99.27 | 105 | 99.39 | 109 | 99.45 | 114 | 100.00 | 108 | 99.39 | 111 | 99.51 | 114 |
| s1238 | 32 | 2310 | 96.75 | 70 | 99.22 | 123 | 98.92 | 127 | 99.78 | 113 | 99.26 | 128 | 98.92 | 130 |
| b04 | 74 | 2553 | 99.96 | 42 | 98.94 | 39 | 98.90 | 66 | 100.00 | 39 | 99.29 | 39 | 99.29 | 67 |
| b05 | 60 | 3671 | 100.00 | 53 | 99.18 | 51 | 98.72 | 84 | 100.00 | 49 | 98.86 | 48 | 99.54 | 83 |
| b11 | 37 | 2227 | 99.91 | 57 | 98.83 | 53 | 98.25 | 69 | 99.96 | 50 | 99.55 | 53 | 98.92 | 67 |
| b12 | 127 | 4615 | 99.85 | 88 | 98.44 | 91 | 97.83 | 145 | 100.00 | 92 | 99.68 | 96 | 99.46 | 145 |
| cavlc | 11 | 3052 | 99.15 | 135 | 98.92 | 156 | 97.97 | 136 | 99.64 | 142 | 99.57 | 156 | 98.69 | 136 |
| dec | 256 | 1840 | 68.21 | 62 | 99.24 | 256 | 99.24 | 256 | 78.75 | 126 | 99.46 | 256 | 99.08 | 256 |
| i2c | 142 | 5014 | 99.84 | 52 | 99.04 | 57 | 97.37 | 117 | 99.92 | 55 | 99.48 | 63 | 98.62 | 116 |
| priority | 8 | 2861 | 100.00 | 26 | 99.20 | 18 | 98.15 | 32 | 99.97 | 19 | 99.65 | 19 | 98.67 | 32 |

test in this range for most circuits. Missing values indicate that the computation did not terminate within the time limit or irredundant static compactor was not available.

The results are compared with Atalanta ATPG [25], which uses static test compaction, and with the OTG [24], which uses the same dynamic test compaction as the ZATPG-PBO.

Our algorithm produces tests of similar length as the OTG while achieving higher fault coverage in most cases, especially for bigger compactors. The OTG and Atalanta produce fault coverage consistent with the theoretical probability of fault masking in an LFSR ($2^{-n}$).

For some circuits, the ZATPG-PBO performs poorly for small compactor sizes. This can be explained by early termination in the case when no test pattern capable of improving fault coverage is found for any remaining fault. Of note is the circuit *dec*, where ZATPG-PBO performs very poorly; full coverage is achieved for MISR of size 8 bits with a test length of 258 patterns. Note that both OTG and Atalanta generated exhaustive tests – 256 patterns for 8-bit PI.

### C. Computation time

Table II shows 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 a higher number of PBO solver invocations.

Table II: Computation time

| MISR | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| circuit | [s] | [s] | [s] | [s] | [s] | [s] | [s] |
| c499 | 149694 | 4398 | 3713 | 1623 | 1497 | 747 | 564 |
| c880 | 239315 | 75849 | 52237 | 74553 | 40264 | 18681 | 13332 |
| c1355 | 74617 | 15462 | 19391 | 10894 | 4052 | 2427 | 1701 |
| c1908 | 237705 | 98428 | 47796 | 19248 | 9143 | 5209 | 3776 |
| c2670 | - | - | 71337 | 31625 | 10699 | 5732 | 3787 |
| c7552 | - | - | - | 89363 | 52530 | 57445 | 5712 |
| s832 | - | 1567 | 1139 | 985 | 953 | 901 | 813 |
| s1238 | 55594 | 31575 | 13013 | 5015 | 3089 | 2425 | 1736 |
| b04 | - | - | 304154 | 19479 | 12266 | 4369 | 2082 |
| b05 | - | 208634 | 212417 | 84267 | 14621 | 7401 | 5516 |
| b11 | 41364 | 34454 | 18744 | 12873 | 5656 | 2722 | 2778 |
| b12 | - | 115641 | 38641 | 14748 | 8751 | 6479 | 3654 |
| cavlc | 8200 | 9863 | 5513 | 3443 | 2985 | 3459 | 3167 |
| dec | 647 | 1622 | 831 | 5222 | 8406 | 10342 | 7600 |
| i2c | - | - | 254353 | 10211 | 7395 | 2391 | 1823 |
| max | - | - | - | - | 271157 | 101279 | 70130 |
| priority | 5919 | 6299 | 496 | 191 | 186 | 186 | - |

For some circuits, there is a visible step in computation time that coincides with runs where the number of anti-aliased faults reached limit $M_a = 100$. This also indicates that fault re-targeting is not as efficient as fault anti-aliasing.

## V. Conclusions and Future Work

The aim of this work was to propose a test generation process minimizing aliasing in the temporal response compactor, *without the need to modify its structure*. This is in contrast to other approaches that aim at the compactor redesign. As a result, satisfactory fault coverage can be achieved, even for small compactors, without sacrificing the test length. Therefore, the BIST area can be reduced "for free"; all the effort is moved to the algorithmic level.

The presented method is a modified ATPG process based on a Pseudo-Boolean Optimization problem, ZATPG-PBO. This algorithm is an improvement upon our previous algorithm, ZATPG [11].

The algorithm is evaluated in terms of fault coverage, test length, and computation time. Our algorithm achieves better fault coverage than ATPG without fault anti-aliasing. The test length is comparable to a state-of-the-art ATPG with modern dynamic test compaction.

The algorithm was tested with the LFSR temporal compactor but can work with any compactor for which the combinational logic can be extracted.

As a future work, we propose evaluating different temporal compactors, for example, cellular automata or non-linear compactors. Additionally, the influence of all parameters and their interaction is not completely understood.

## Acknowledgment

The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics". Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic. Computational resources were provided by the ELIXIR-CZ project (LM2018131), part of the international ELIXIR infrastructure.

## References

[1] 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, Aug. 2002.

[2] 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*, Nov. 2013, pp. 95–100.

[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] H. Assaf, Mansour, B. Jone, Wen, M. Sahinoglu, R. Das, Sunil, A. Hossain, S. Biswas, and M. Petriu, Emil, "On a new graph theory approach to designing zero-aliasing space compressors for built-in self-testing," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 10, pp. 2146–2168, 2008.

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

[6] 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.

[7] G. Edirisooriya, P. Robinson, John, and S. Edirisooriya, "On the performance of augmented signature testing," in *IEEE International Symposium on Circuits and Systems*, May 1993, pp. 1607–1610.

[8] M. Kopec, "Can nonlinear compactors be better than linear ones?" *IEEE Transactions on Computers*, vol. 44, no. 11, pp. 1275–1282, Nov. 1995.

[9] 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.

[10] 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, April 1993.

[11] R. Hülle, P. Fišer, and J. Schmidt, "ZATPG: SAT-based test patterns generator with zero-aliasing in temporal compaction," *Microprocessors and Microsystems*, vol. 61, pp. 43 – 57, 2018.

[12] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: a reverse order test compaction technique," in *Proceedings Euro ASIC '92*, June 1992, pp. 189–194.

[13] 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)*, April 1997, pp. 188–195.

[14] 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)*, Nov 2001, pp. 1088–1097.

[15] 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*, vol. 12, no. 7, pp. 1040–1049, July 1993.

[16] 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*, April 2009, pp. 1136–1141.

[17] Jau-Shien Chang and Chen-Shang Lin, "Test set compaction for combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1370–1378, Nov 1995.

[18] G. Tromp, "Minimal test sets for combinational circuits," in *IEEE International Test Conference*, Oct 1991, pp. 204–209.

[19] 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.

[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*, vol. 35, no. 12, pp. 2104–2117, 2016.

[21] 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.

[22] 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.

[23] I. 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*, 2002, pp. 1008–1013.

[24] S. Eggersglüß, K. Schmitz, R. Krenz-Baath, and R. Drechsler, "Optimization-based multiple target test generation for highly compacted test sets," in *2014 19th IEEE European Test Symposium (ETS)*, May 2014, pp. 1–6.

[25] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits," Dep't of Electrical Eng., Virginia Polytechnic Institute, Tech. Rep.

[26] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 11 2006.

[27] 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.

[28] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS'89)*, May 1989, pp. 1929–1934 vol.3.

[29] 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.

[30] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *International Workshop on Logic & Synthesis (IWLS)*, 2015.