Optimization-based Multiple Target Test Generation for Highly Compacted Test Sets

Stephan Eggersglüß*† Kenneth Schmitz*† René Krenz-Bååth‡ Rolf Drechsler*†

*Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{segg,kenneth}@informatik.uni-bremen.de

‡Hochschule Hamm-Lippstadt
59063 Hamm, Germany
Rene.Krenz-Baath@hshl.de

†Cyber-Physical Systems
DFKI GmbH
28359 Bremen, Germany
Rolf.Drechsler@dfki.de

***Abstract*—Test compaction is an important aspect in the post-production test since it is able to reduce the test data and the test costs, respectively. Current ATPG methods treat all faults independently from each other which limits the test compaction capability. This paper proposes a new optimization based SAT-ATPG for compact test set generation. Robust solving algorithms are leveraged to determine fault groups which can be detected by the same test. The proposed technique can be used during initial compact test generation as well as a post-process to increase the compactness of existing test sets, e.g. generated by commercial tools, in an iterative manner. Experimental results on industrial circuits and academic benchmarks show that this technique is able to significantly reduce the pattern count down to 40% for the initial test generation and down to 30% for the iterative reduction.**

I. Introduction

The manufacturing test is an important step in the production process of computer chips. A test set is applied to each fabricated chip in order to detect defective devices. One important factor for the test costs is the test data volume and the size of the test set. The growing complexity of today's designs leads to rapidly increasing test data and consequently to high test costs. Therefore, much effort is spent to reduce the test data. Two different techniques are generally used to reduce the test data. Test compression [1] applies additional hardware to compress test cubes and responses. Test compaction techniques reduce the number of test patterns (ideally without reducing the fault coverage) to save test data. This paper is concerned with new test compaction techniques.

A core technique in this context is *Automatic Test Pattern Generation* (ATPG). The task of ATPG is to generate a test set which has a high fault coverage and, at the same time, is as small as possible to avoid high test costs. Structural ATPG algorithms such as FAN-based approaches are known to be very fast in generating tests for easy-to-test faults, but the number of faults for which these approaches abort due to reasons of complexity is increasing. This can effect the fault coverage significantly. On the other hand, SAT-based ATPG approaches suffer from the higher run time for easy-to-test faults, but are very robust for hard faults. Generally, SAT-based ATPG algorithms are able to produce a higher fault coverage for circuits with many hard-to-test faults [2].

Test compaction techniques can be classified into static and dynamic techniques. Static techniques [3], [4] work on an existing test set and remove redundant tests as well as merge test cubes. Dynamic test compaction techniques [5]–[9] influence the test generation procedure itself. Typically, unspecified bits of generated test sets are assigned to detect further faults. This is computationally very intensive but yields a more compacted test set than static compaction. The generation of new tests to remove other tests, e.g. Two-by-One reduction, has been proposed in [6]–[8].

Additionally, formal solving engines have been applied to improve test compaction. Dynamic compaction for SAT-based ATPG has been proposed in [10]. This technique relies on the identification of necessary line assignments to group faults. A QBF solving engine was applied in [11] in order to maximize the number of unspecified bits in a test cube for a given sensitized path. This can be used to merge test patterns or increase the compression ability. The SAT-based ATPG approaches in [12] and [13] use optimization procedures to produce a compact test set. The technique in [12] explicitly targets a single fault and encodes additional local fault detection conditions for undetected faults into the SAT instance. The solver maximizes these conditions to generate a test that is likely to detect many other faults. However, this technique partly relies on an runtime-intensive classical dynamic compaction scheme using an additional target generator. The technique in [13] works as a post-process. The necessary assignments of sensitized paths are extracted from a pre-generated test set and an optimization procedure is applied to generate tests which detects as many of these paths as possible. However, since the detection paths are fixed in this method, the flexibility is missing to detect a previously detected fault through a different path.

All approaches described so far have in common that they treat faults rather independently from each other. The work proposed in [14] introduces a SAT-based ATPG formulation which targets multiple faults at the same time (*Multiple Target Test Generation*, MTTG). Given a set of faults F, the SAT-based ATPG procedure will provide a test which detects all faults in F if one exists. If only two faults $f_1 \in F$ and $f_2 \in F$ exist which are not testable together due to conflicting necessary assignments, no test can be generated. Consequently, most run time is spent to identify a set of faults which can be detected by the same test. This typically involves a large number of time-consuming SAT solver calls.

This shortcoming is addressed in this paper. The problem of MTTG is formulated as a Boolean optimization problem. The task of determining a compatible fault set is integrated into the problem formulation itself. By this, the powerful solving techniques of modern SAT solvers can be leveraged. Given a set of faults F, the problem formulation is such that a test will be generated which detects the highest possible number of faults out of F. This prevents the costly explicit search for

a set of faults which can be detected together. The selection of the faults is implicitly integrated into an optimization function and is therefore tightly integrated into the solving process. Due to the robust underlying solving engine, solvers are highly suited to solve complex problems such as multiple-target test generation.

The proposed technique can be applied in two different ways. It can be used to generate a compact initial test set as well as to improve the compactness of an existing test set. This is especially of practical relevance since test sets generated by commercial ATPG tools can be improved afterwards if the test engineer is not satisfied with the produced test set. The experimental results on industrial circuits show that the proposed approach is able to reduce the pattern count significantly if the technique is applied in an iterative manner.

This paper is structured as follows. Section II briefly introduces SAT-based ATPG and optimization-based algorithms. Section III describes the SAT formulation for optimization-based MTTG. The integration into the ATPG flow during initial test generation and as a post-process is shown in Section IV. Experimental results are given in Section V. Finally, conclusions are drawn in Section VI.

II. Preliminaries

On key aspect of the robustness of SAT-based algorithms is the problem formulation as a Boolean formula in *Conjunctive Normal Form* (CNF) [15]. A CNF Φ is a conjunction of m clauses. A clause ω is a disjunction of n literals. A literal λ is a Boolean variable in its positive (λ) or negative ($\overline{\lambda}$) form. The problem formulated in CNF is solved by a SAT solver which generates a solution to show that the CNF is satisfiable (SAT) or proves that no such solution exists, i.e. the formula is unsatisfiable (UNSAT). SAT solving algorithms have a high ability to solve hard problems in reasonable run time. The homogeneous structure of the CNF allows for the application of fast implication techniques and powerful conflict-based learning schemes. It is shown in [2] that SAT-based ATPG is able to produce higher fault coverage than classical structural ATPG techniques.

In the following, the problem formulation in CNF is briefly described. More detailed information can be found in [16]. First, the relevant circuit part C_f with gates G and signal lines S for detecting the fault f is identified. A Boolean variable x is assigned to each signal line $s_x \in S$ in this part representing its logical value. Then, each gate $g \in G$ is transformed into a set of clauses Φ_g using the respective input and output variables. The circuit CNF Φ_{C_f} is obtained by a conjunction of the gates' CNFs: $\Phi_{C_f} = \prod_{i=1}^{l} \Phi_{g_i}$. This formula is then augmented with the CNF for the faulty part of the circuit and fault detection constraints, e.g. D-chains, described by Φ_F. The final SAT instance processed by a SAT solver is given by $\Phi_f^{test} = \Phi_{C_f} \cdot \Phi_F$.

Recently, optimization SAT solvers have been applied in the field of test generation, e.g. [12], [13]. The application is related to classical SAT-based ATPG. The underlying problem is formulated in CNF. But instead of settling for an arbitrary solution for the problem as SAT solvers do, optimization SAT solvers are able to process an optimization function which determines the quality of the solution. Depending on the implementation, these kind of solvers generate new solutions in an iterative manner until the best solution is found. This process is heavily guided by learned information and, thus, is very powerful. A common form of the optimization function $\mathcal{F}$ is to connect constant values c_i to Boolean literals x_i:

$$\mathcal{F} = c_1 \cdot x_1 + \ldots + c_k \cdot x_k$$

By this, all constants, whose associated literals evaluate to true, are accumulated. The optimization solver returns the assignment which satisfies the CNF and, at the same time, optimizes $\mathcal{F}$, i.e. minimize or maximize. The run time of the solving process is strongly influenced by the size of $\mathcal{F}$.

III. Optimization-based Multiple Target Test Generation

This section describes the *Optimization-based MTTG* (OTG). This includes the SAT formulation of the OTG problem including the construction of the necessary optimization function. The integration of the OTG approach into the initial test set generation as well as the application as a post-process to improve the test compactness will be described in Section IV.

Classical compaction methods compute an assignment for the detection of one fault first. This assignment is then used as a constraint during the detection of other faults.[1] The detection of other faults depends on the chosen assignment, i.e. the generated test cube. Since each fault is treated independently, this assignment might prevent faults being detected by the same test, although it would be possible if a different test would have been generated.

The aim of the MTTG technique is to handle the test generation of multiple faults in one step. Given a set of faults[2] $F = \{f_1, \ldots, f_n\}$, an MTTG approach is able to generate a test for all faults $f \in F$ if one exists. This leads to the advantage that potentially more faults can be detected by one test and hence less test patterns are necessary. However, if no such test for all faults exists, i.e. at least two faults $f_i, f_j \in F$ are conflicting, no test will be generated. The problem is, therefore, the identification of a fault set in which all faults are non-conflicting. Typically, many inconclusive ATPG calls are necessary for test generation which limits the application possibilities and the effectiveness.

We propose to formulate the MTTG problem as an optimization problem that formal solving algorithms can be applied. Given a set of faults $F = \{f_1, \ldots, f_n\}$, the goal is to generate one test which detects the maximum possible number of faults out of F. By this, the identification of a non-conflicting fault set is inherently done by the solving algorithm itself. The advantage of the application lies in the integrated powerful learning techniques. Formal (SAT-based) optimization solvers such as clasp [17] are able to learn correlations between signal assignments very effectively. By this, conflicts between faults can be internally identified and used by the internal solving algorithm to guide the search towards a non-conflicting fault set.

The formal problem description for test generation for the fault set F given to an optimization solver consists of a SAT instance Φ_F and the optimization function $\mathcal{F}_F$. First, the construction of the SAT instance is described.

[1] Or in other words, the remaining X values are specified to detect further faults.

[2] This approach is applied to *single* stuck-at and transition faults.

1) A structural analysis is applied to identify the relevant circuit part C_F. The relevant circuit part contains all signals and gates which can structurally influence the fault activation or propagation. This part is transformed into the CNF Φ_F^C.
2) The faulty output cone of each fault (including the fault site) $f_1, \ldots, f_n \in F$ is identified and transformed into CNF: $\Phi_{f_1} \cdot \ldots \cdot \Phi_{f_n}$
3) Fault detection constraints, i.e. D-chains [18], are generated for each fault: $\Phi_{f_1}^D \cdot \ldots \cdot \Phi_{f_n}^D$. These constraints are used to establish a D-chain from the fault site to an observation point. This is done by assigning a D-variable to each line l in the faulty output cone. The D-variables have the following meaning: If $D_l = 1$ holds, then there is a difference in the correct and faulty circuit on line l and there is a path from l to an observation point where all D-variables are assigned with 1. On the other hand, if $D_l = 0$ holds, no implication is performed.

In contrast to previous methods, the D-chains are not explicitly triggered, i.e. they are inactive. This is necessary for the application of the optimization algorithm. By this, the solving process is able to dynamically activate and deactivate the detection of certain faults in order to find the maximum set of non-conflicting faults.

In summary, the underlying SAT instance Φ_F is given by:

$$\Phi_F = \Phi_F^C \cdot \Phi_{f_1} \cdot \ldots \cdot \Phi_{f_n} \cdot \Phi_{f_1}^D \cdot \ldots \cdot \Phi_{f_n}^D$$

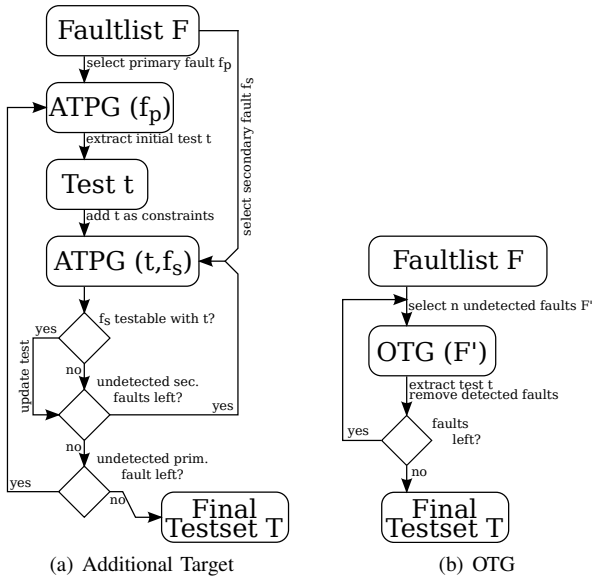
Under the assumption that there is no contradiction in the circuit logic itself, the SAT instance Φ_F is always satisfiable as it is necessary. Next, the construction of the optimization function $\mathcal{F}$ is described. As mentioned above, the fault detection for fault f on line l_f can be triggered by setting $D_{f_l} = 1$. By this, a D-chain from the fault site to an observation point will be established if possible. Since the goal is to establish as many D-chains as possible, the optimization function is formulated over the D-variables of the fault sites of F.

Pseudo-Boolean optimization solvers typically process the optimization function in the following form: $\mathcal{F} = c_1 \cdot \lambda_1 + \ldots + c_m \cdot \lambda_m$ with constant $c_i \in \mathbf{Z}$ and with λ_i as a Boolean literal. The formula is interpreted in an arithmetic way. The result is the accumulation of all constants for which the corresponding Boolean literal evaluates to true. Typically, the optimal solution is the minimum result.

Let $F = f_1, \ldots, f_n$ be the target fault set and let furthermore d_{f_i} be the corresponding D-variable for each fault $f_i \in F$ which triggers the fault detection. Then, the optimization (minimization) function is formulated over $d_{f_1}, \ldots, d_{f_n}$:

$$\mathcal{F}_F = 1 \cdot \overline{d}_{f_1} + \ldots + 1 \cdot \overline{d}_{f_n}$$

The SAT instance Φ_F and the corresponding function $\mathcal{F}_F$ is eventually given to an optimization solver to compute a test which is then directly stored in the final test set. In spite of the robustness of the underlying solving engine, it might sometimes happen that the approach is not able to prove that the found solution is optimal. However, the search process works in an iterative manner, i.e. whenever a better solution is found, it will be enumerated. Because of this, a good and in most cases near-optimal, solution is found nonetheless.



(a) Additional Target (b) OTG

Fig. 1. Dynamic Compaction Flow

IV. Compaction and Retarget Stage

The previous section introduced the OTG formulation for a given set of faults F. This section discusses how this method can be integrated into the test generation flow. In particular, it is shown how the fault set F is constructed. Additionally, two different applications are shown:

1) A dynamic compaction method is proposed in which OTG is used to generate an initial test set.
2) An initial test set T exists and the proposed OTG formulation is used to improve the compactness of T by constant or improved fault coverage.

A. Initial Test Set Generation

A common dynamic compaction procedure for generating a compacted test set is shown in Figure 1(a). First, a primary fault f_p is selected and a test cube is generated. This test cube is then extended by a loop over a list of secondary faults. Typically, fault lists are structurally ordered. e.g. based on fanout-free regions. Primary faults and secondary faults are then processed according to the ordering.

If a secondary fault f_s can be additionally detected by specifying X-bits, the test is updated and the loop is continued with the extended test until all secondary tests have been processed. This procedure is then continued by selecting other yet undetected primary faults until all faults are classified.

In contrast, the proposed procedure for the application of OTG is shown in Figure 1(b). Instead of choosing primary and secondary faults, a set of n yet undetected faults is selected and given to the OTG as targets. The selection of the faults is based on the fault list ordering. The effort of determining which faults are non-conflicting and consequently can be detected by one test is completely passed to the reasoning engine. A test will be generated detecting the maximum number of non-conflicting faults. This test is fault simulated and all faults detected by this test will be dropped from the fault list. Next, a set of n undetected faults is selected again and given to the OTG. Since this set is based on the fault list ordering as well, all

these faults not detected by the test from the previous OTG call, are also included in the fault set.

Untestable faults can also be identified by the OTG approach. Since all testable faults are dropped from the fault list during the procedure, there are only untestable faults left at the end. If no fault from a given set of faults F is testable, the procedure returns the maximum of 0 detected faults. Then, it can be concluded that all faults $f \in F$ are untestable. However, it turned out that less run time is needed if the detection of one selected fault f is encoded directly into the SAT instance. When the SAT instance is unsatisfiable, fault f is untestable. If not, a test will be generated detecting f and the maximum number of additional faults. A negative impact on the pattern count could not be observed.

This approach is able to produce a compact test set and, at the same time, uses the advantage of robust formal reasoning engines to produce a high test coverage.

B. Improving Existing Test Sets

An additional application of the OTG procedure is the improvement of existing test sets. Typically, ATPG tools generate an initial test set, but do not provide many possibilities to improve this test set afterwards if the test engineer is not satisfied with the compactness. A common method is to truncate the test set if the test set is to large for the tester and, by this, loose fault coverage. A method to improve the test set without loosing fault coverage is presented in this section.

The techniques proposed in [6], [7] try to remove one or more tests at a time using a greedy search. The algorithm tries to find a test set for all essential faults of N tests resulting in $N-1$ tests, i.e. Two-by-One reduction. An essential fault is a fault that is detected by exactly one test in T. If a test $t \in T$ does not have any essential faults, it becomes redundant and can be removed. However, the approach is computationally to intensive for $N > 2$ as reported in [8] and, thus, limited in its applicability.

The approach in [8] proposed a more global view and presented a technique to improve the pattern count by the distribution of all essential faults of one test t to other tests. In order to distribute essential faults, one essential fault f^e is selected first. Then, the approach iterates over each test in the test set and tries to generate a test detecting all faults which were previously detected plus f^e. Once all essential faults of a test t have been distributed to other tests, the test t can be removed. However, the combination of every essential fault with every test pattern is a huge overhead for today's circuits with a large number of faults as well as patterns.

The technique proposed in this section also uses the notion of essential faults. However, instead of distributing single essential faults explicitly to other tests, the approach targets all essential faults of a selected test subset $T' \in T$ at once. By this, an effective N-by-M with $M < N$ is produced which is more powerful than the previous proposed approaches. For this, we extend the definition of essential faults to *set-essential faults*. Set-essential faults are those faults which are only detected by patterns of a test subset $T' \in T$. This includes all essential faults as well as those faults which remain undetected when T' would be removed, i.e. those faults which are globally detected more than once, but only from tests out of T'.

The following procedure is used:

1) First, an essential fault identification for the complete fault set F is performed. This is implicitly done by counting the number of detections by the initial test set T.
2) Next, a test subset T' is heuristically selected. Then, all set-essential faults detected by T' are identified. This fault set is described by $F_e = f_1^e, \ldots, f_m^e$.
3) The fault set F_e is retargeted by the OTG procedure in an iterative manner until all faults are detected. The resulting test set is given by T^*.
4) If $|T^*| \leq |T'|$ holds, T' is replaced by T^*. The fault detection statistics are updated for the further identification of set-essential faults.
5) This procedure continues until all tests $t_i \in T$ have been processed.

This retargeting procedure can be repeatedly applied to improve the compactness of the test set further.[3] The powerful underlying reasoning engine allows for a consideration of several hundred faults at once. The improved compactness is achieved without fault coverage loss. A significant advantage of this technique is that it is able to process large pattern sets (independently from the source of the test set) and that it can be flexibly applied depending on the resources the test engineer is able to spent and, by this, provides a powerful alternative to test set truncation.

V. Experimental Results

This sections presents the experimental results obtained by the proposed approaches. Both algorithms were applied to academic benchmark circuits as well as to industrial circuits provided by NXP Semiconductors. The results are compared to a state-of-the-art commercial ATPG tool as well as to a previous SAT-based approach [12].[4] The methods were implemented in C++ and the experiments were conducted on an Intel Xeon E3-1240 (GNU/Linux, 64bit, 32GByte RAM, 3.4 GHz) in single-threaded mode. The optimization solver *clasp* [17] was used as reasoning engine.

Table I gives the results for stuck-at faults for the benchmark circuits.[5] The results for transition faults (launch-on-capture) are given in Table II. Column *Commercial* presents the results for the commercial ATPG. The results for the SAT-based approach [12] using an additional target generator are shown in column *SAT [12]*. Column *Initial OTG* gives the results for the initial test generation with 50 simultaneously injected faults as well as 200 faults. The reduction results of the *Retarget* stage are given in the columns *It.* with the respective number of iterations. Here, the test subset is chosen such that roughly 200 set-essential faults are targeted in one OTG call. The run times are given in columns *Rt* $[sec]$ and the produced pattern count is shown in columns named *Pat*.

The results show that the commercial ATPG is very fast, but produces a larger test set compared to the SAT-based

[3] It has to be ensured that the selected test subsets differ from the previous iteration, e.g. by shuffling the test ordering.

[4] A fair comparison to the pattern counts of the approach in [13] is not possible, since their method is only applied to combinational cores using enhanced full scan (transition faults).

[5] Some results fall below the lower bound computed in [8]. A possible reason is that the synthesized netlists may differ.

TABLE I. EXPERIMENTAL RESULTS – STUCK-AT

	Commercial		SAT [12]		Proposed Approach						
					Initial OTG				Retarget		
					#Faults=50		#Faults=200		It. 10	It. 50	It. 100
Circuit	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Pat	Pat	Pat
s5378	0.7	218	15.7	88	0.6	82	7.6	80	79	79	78
s9234	0.3	162	137.3	153	2.1	141	21.2	114	110	106	106
s13207	0.6	303	199.9	260	5.3	299	14.4	252	239	235	233
s15850	0.8	183	243.0	115	19.4	134	280.8	111	105	99	97
s35932	1.0	60	37.8	23	5.0	126	9.7	43	33	27	25
s38417	1.6	169	221.0	89	21.7	127	261.8	97	93	83	80
s38584	1.8	206	197.4	141	14.1	271	156.4	148	145	127	119
b04	0.3	91	6.8	66	0.4	60	3.4	59	57	57	57
b14	3.2	790	8656.74	728	63.6	732	98.5	663	621	602	596
b15	25.8	521	7810.0	489	55.8	459	158.4	404	363	342	338
b20	9.3	845	not avail.		264.2	725	766.5	648	632	600	596

TABLE II. EXPERIMENTAL RESULTS – TRANSITION

	Commercial		SAT [12]		Proposed Approach						
					Initial OTG				Retarget		
					#Faults=50		#Faults=200		It. 10	It. 50	It. 100
Circuit	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Pat	Pat	Pat
s5378	0.9	364	56.0	156	2.0	104	14.6	93	91	83	82
s9234	3.6	492	455.0	339	8.0	230	35.0	192	191	185	176
s13207	2.3	577	414.4	312	10.9	315	43.7	254	238	226	218
s15850	3.9	312	324.8	149	18.4	134	137.4	116	111	105	102
s35932	3.0	110	52.1	38	14.3	115	56.7	43	38	35	33
s38417	4.4	341	823.0	160	117.2	175	641.3	165	162	142	138
s38584	6.3	588	887.3	401	44.9	370	481.4	257	257	246	232
b04	0.6	163	14.1	91	0.8	80	2.9	76	71	67	67
b14	1.0	1438	55788.0	1481	187.9	1118	453.3	1050	861	802	789
b15	79.7	1300	46342.3	1676	433.0	1345	1435.0	945	842	713	669
b20	55.9	1897	379999.1	3047	934.5	1656	3696.3	1116	885	781	756

approaches. On the other hand, the SAT-based approach [12] requires a higher run time especially for the larger benchmarks and for transition faults. This is caused by the applied additional target loop. The proposed OTG approach is a fast alternative to the SAT-based method when injecting a small set of faults simultaneously, i.e. 50. Often, a pattern count reduction can be already observed with this configuration, especially for the transition faults. The full impact on the pattern count can be seen by injecting more faults, i.e. 200. While the reduction is mostly moderate or sometimes even negative for the small s-circuits, the pattern counts of the larger benchmarks with more patterns are tremendously reduced. For example, the pattern count of b20 (transition faults) drops from 1897 (commercial) and 3047 (SAT) to 1116 patterns. This is a reduction to 59% and 37%, respectively. Compared to the SAT-based approach, only 1% of the run time is needed.

The results of the retarget stage shows that the low pattern count can be further reduced significantly by the iterative application of the proposed OTG technique. For example, applying this technique 10x (50x,100x) to b20, the pattern count can be reduced to only 47% (41%, 40%). It is noticeable that especially the large test sets can be significantly reduced by the proposed approach.

Next, the application to industrial circuits will be discussed. The results are listed in Table III (initial test set) and Table IV (retargeting). Since these circuits are more complex, the ATPG was not able to generate tests for each fault. Therefore, the number of aborted faults is added to the table. Two different setups for the commercial ATPG tool are used. Since the default configuration produced a very high number of aborted faults, the tool is started with a very high backtrack limit to allow for a fair comparison with the proposed approach. The OTG approach is also started twice with 100 as well as 200 injected faults.

The newly proposed OTG approach generates smaller test sets than the commercial tool for the majority of the benchmarks. It is important to mention that due to its robustness, the SAT-based ATPG obtains a reduced amount of aborted ATPG calls, hence achieves higher fault coverage and therefore generates additional tests. This effect appears especially for circuit p77k where the commercial ATPG tool aborts more than 16 thousand times and generates 552 test patterns. In contrast to that, the proposed approach generates only 549 test patterns while aborting only once. In particular, this example demonstrates how the robustness of modern SAT engines is able to reduce the number of unclassified faults and concurrently improves the compactness of the test set.

The new approach achieves test set reductions for the majority of the industrial circuits. Please consider that the tremendous run times consumed by the commercial ATPG tool are caused by the very high backtrack limit, which was chosen to achieve a comparable amount of aborted faults and hence a meaningful comparison of the achieved number of test patterns. However, the OTG approach does not deliver a more compact initial test set than the commercial ATPG for a few benchmarks, i.e. p77k and p81k.

The results depicted in Table IV were achieved by the newly proposed retargeting stage. As parameter, 50 set-essential faults are picked for retargeting. The results achieved for every evaluated benchmark are listed within an individual set of columns, where every set contains the run time per iteration step in seconds and the size of the resulting test pattern set after each iteration. In almost all cases, the test set can be reduced after each iteration, where the number of additionally removed test patterns can also increase during later iterations. The bottom row depicts the accumulated time of all iterations and the resulting test pattern set after static compaction. Especially the already highly compact test set for p35k, which was roughly 40% smaller than the test set generated by the commercial ATPG tool, was additionally reduced down to almost 30% of the commercial ATPG test set.

TABLE III. EXPERIMENTAL RESULTS - INDUSTRIAL CIRCUITS - STUCK-AT

	Commercial, default			Commercial, high backtrack limit			OTG, #Faults=100			OTG, #Faults=200		
Circuit	Rt $[sec]$	**Pat**	Aborted	Rt $[sec]$	**Pat**	Aborted	Rt $[sec]$	**Pat**	Aborted	Rt $[sec]$	**Pat**	Aborted
p35k	30.2	**1543**	1	267.3	**1523**	0	5092.1	**634**	0	14801.5	**624**	0
p45k	8.90	**2103**	41	75.8	**2100**	0	74.3	**1853**	0	169.4	**1780**	0
p77k	683.5	**538**	16064	162573.3	**552**	16360	59216.1	**566**	29	72949.8	**549**	1
p78k	6.8	**82**	7	17.8	**82**	0	306.4	**78**	0	1051.5	**78**	0
p81k	41.9	**379**	328	2536.2	**379**	9	1387.4	**564**	0	3868.2	**556**	0
p100k	62.8	**2051**	717	2788.0	**2054**	202	233.8	**1753**	0	510.2	**1690**	0

TABLE IV. EXPERIMENTAL RESULTS - INDUSTRIAL CIRCUITS - STUCK-AT - RETARGET

Circuit →	p35k		p45k		p81k		p100k	
Iteration ↓	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat	Rt $[sec]$	Pat
initial	–	634	–	1853	–	564	–	1753
1	1871.6	589	483.3	1832	1153.3	563	1277.1	1728
2	1978.5	547	451.1	1828	1212.1	549	1234.0	1710
3	2057.2	519	447.6	1817	1309.9	525	1256.5	1701
4	1972.0	505	450.2	1813	1332.8	485	1251.9	1693
5	2116.3	494	445.7	1803	1354.7	456	1281.0	1689
6	2123.8	480	446.6	1801	1300.4	427	1262.4	1687
7	2072.2	468	443.3	1799	1797.2	399	1253.0	1683
8	2115.8	459	444.2	1794	1374.7	382	1259.0	1680
9	2418.5	449	440.7	1790	1737.3	366	1254.6	1680
10	2350.3	446	442.4	1789	1464.1	355	1264.5	1680
Total	26241.9	446	4672.9	1789	15551.5	347	13027.9	1680

Future work is the improvement of the run time. So far, the complete test set has been evaluated during retargeting. This procedure could be improved by heuristics to determine subsets of patterns for which the compaction might be worthwhile. Another issue is the compatibility with test compression logic for which a certain amount of don't cares is necessary. In future work, we will address OTG techniques which are able to constrain the amount or distribution of don't care bits.

VI. CONCLUSIONS

The size of the test set is an important cost factor in the post-production test of digital circuits. The increasing size and complexity of the circuits lead to increasing pattern counts and increasing test costs. Therefore, new techniques to reduce the pattern count and, at the same time, yield a high fault coverage are of high importance.

In this paper, we have proposed a new optimization-based test formulation which is able to target multiple faults in a single step. Given a subset of faults, the approach is able to generate a test which detects the maximum number of non-conflicting faults in this subset. The underlying SAT-based reasoning engine is powerful enough to target several hundred faults at once. This OTG technique is integrated into a dynamic compaction scheme to initially generate a compact test set providing a high fault coverage.

Additionally, a new retarget stage has been proposed in which the test set can be reduced in an iterative manner using the new OTG technique. This method is a powerful alternative to test set truncation with the ability to maintain a high fault coverage. Experimental results on benchmark and industrial circuits have shown that the test set can be significantly reduced by the proposed technique.

VII. ACKNOWLEDGMENT

This work has been supported by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative.

REFERENCES

[1] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design & Test of Computers*, vol. 23, no. 4, pp. 294–303, 2006.

[2] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On acceleration of SAT-based ATPG for industrial designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[3] 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, 1993.

[4] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy, "On static test compaction and test pattern ordering for scan designs," in *International Test Conference*, 2001, pp. 1088–1097.

[5] P. Goel and B. C. Rosales, "Test generation and dynamic compaction of tests," in *International Test Conference*, 1979, pp. 189–192.

[6] J.-S. Chang and C.-S. 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, 1995.

[7] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 12, pp. 1496–1504, 1995.

[8] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *International Conference on Computer-Aided Design*, 1998, pp. 283–289.

[9] S. Remersaro, J. Rajski, S. M. Reddy, and I. Pomeranz, "A scalable method for the generation of small test sets," in *Design, Automation and Test in Europe*, 2009, pp. 1136–1141.

[10] A. Czutro, I. Polian, P. Engelke, S. M. Reddy, and B. Becker, "Dynamic compaction in SAT-based ATPG," in *IEEE Asian Test Symposium*, 2009, pp. 187–190.

[11] M. Sauer, S. Reimer, I. Polian, T. Schubert, and B. Becker, "Provably optimal test cube generation using quantified Boolean formula solving," in *ASP Design Automation Conference*, 2013, pp. 533–539.

[12] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *International Conference on Computer-Aided Design*, 2013, pp. 85–90.

[13] M. Sauer, S. Reimer, T. Schubert, I. Polian, and B. Becker, "Efficient SAT-based dynamic compaction and relaxation for longest sensitizable paths," in *Design, Automation and Test in Europe*, 2013, pp. 448–453.

[14] S. Eggersglüß, R. Krenz-Bååth, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *IEEE Symposium on Design and Diagnosis of Electronic Circuits and Systems*, 2012, pp. 230–235.

[15] A. Biere, M. Heule, H. v. Maaren, and T. W. (Eds.), *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009.

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

[17] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *International Joint Conference on Artificial Intelligence*, 2007, pp. 386–392.

[18] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.