SAT-Based ATPG beyond Stuck-at Fault Testing

Hellebrand, Sybille; Wunderlich, Hans-Joachim

it - Information Technology Vol. 56(4) 21 July 2014

doi: http://dx.doi.org/10.1515/itit-2013-1043

Abstract: To cope with the problems of technology scaling, a robust design has become desirable. Self-checking circuits combined with rollback or repair strategies can provide a low cost solution for many applications. However, standard synthesis procedures may violate design constraints or lead to sub-optimal designs. The SAT-based strategies for the verification and synthesis of self-checking circuits presented in this paper can provide efficient solutions.

Preprint

General Copyright Notice

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

This is the author's "personal copy" of the final, accepted version of the paper published by *DE GRUYTER*.

©2015 **DE GRUYTER**

SAT-Based ATPG beyond Stuck-at Fault Testing

Applications to Fault Tolerance

Sybille Hellebrand[1]; Hans-Joachim Wunderlich

Kurzfassung: Die Fortschritte in der Mikroelektronik sind mit einer Reihe von Problemen verbunden, die durch einen robusten Entwurf abgefangen werden können. Selbstprüfende Schaltungen ermöglichen in vielen Fällen kostengünstige Lösungen. Standardsyntheseverfahren können jedoch einerseits Fehlertoleranzeigenschaften zerstören und andererseits das spezifische Optimierungspotential nicht vollständig ausnutzen. Die in diesem Artikel vorgestellten SAT-basierten Verifikations- und Syntheseverfahren für selbstprüfende Schaltungen lösen dieses Problem effizient.

Abstract: To cope with the problems of technology scaling, a robust design has become desirable. Self-checking circuits combined with rollback or repair strategies can provide a low cost solution for many applications. However, standard synthesis procedures may violate design constraints or lead to sub-optimal designs. The SAT-based strategies for the verification and synthesis of self-checking circuits presented in this paper can provide efficient solutions.

SAT-basierte Testmustererzeugung, Fehlertoleranz, Selbstprüfende Schaltungen, Synthese/SAT-based ATPG, Fault Tolerance, Self-Checking Circuits, Synthesis

1 Introduction

IC manufacturing in state-of-the-art technologies not only has to cope with high defect densities, but nano-scale circuits also suffer from parameter variations and an increased susceptibility to external noise [3, 5]. In this scenario, a traditional design based on worst case assumptions cannot fully benefit from technology scaling. As a consequence, “robust” design strategies have gained increasing attention in recent years [14, 20].

As classical fault tolerant architectures like *n*-modular redundancy are too costly for many applications [25], concurrent error detection combined with rollback or repair strategies has become more and more attractive as a low cost alternative [14, 35, 36]. In this context, self-checking circuits benefit from well-established design rules and fault detection properties [27]. However, standard synthesis procedures are not tailored to the specific requirements of self-checking logic. On the one hand, they may violate design constraints, such that fault tolerance properties must be verified after synthesis [15, 22, 23]. On the other hand, they are not able to take advantage of the specific optimization potential for code generators, prediction logic and checkers [9].

Since the fault tolerance properties of self-checking circuits are related to the detectability or undetectability of faults during system operation, synthesis and verification of self-checking circuits are based on automatic test pattern generation (ATPG) in [9, 22, 23].

In the following, this paper discusses both applications in more detail. In particular, it will be shown that SAT-based ATPG perfectly supports the specific requirements in this domain. The rest of this paper is organized as follows. In Sections 2 and 3 the necessary background on SAT-based ATPG and on self-checking

[1] sybille.hellebrand@uni-paderborn.de

circuits is briefly summarized. Then the SAT-based technique for analyzing fault tolerance properties is presented in Section 4, followed by the SAT-based technique for the synthesis of fault secure circuits.

2 SAT-Based ATPG

For a given fault, the task of ATPG is to derive an input pattern, such that the circuit outputs differ in the fault free and the faulty case, or to prove that such a pattern does not exist ("redundancy identification"). As illustrated in Figure 1, conceptually this can be solved by comparing copies of the fault free and the faulty circuit. The circuit outputs are combined with pairwise XOR operations, the outputs of which are connected to an OR gate. To find a test input, at least one output of an XOR gate must be driven to "1", i.e. the output of the OR gate must evaluate to "1".

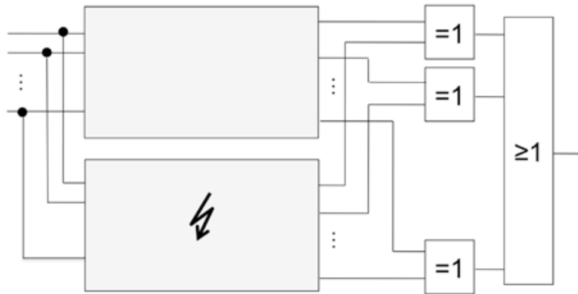


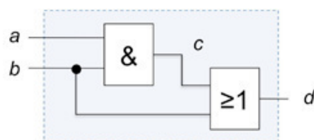
Figure 1: ATPG model: Comparing fault free and faulty copies.

If the circuit netlist (structural model) is replaced by the underlying Boolean expressions, then the ATPG problem corresponds to "satisfying" a Boolean formula. In complexity theory the Satisfiability Problem (SAT) is defined as the problem to decide whether a variable assignment for a Boolean formula in conjunctive form ("product of sums") exists, such that the formula evaluates to "1". This problem is known to be NP-complete, and based on this result it has been proven that the ATPG problem is NP-complete as well [24].

Despite this strong relationship between ATPG and SAT solving, SAT-based ATPG was considered as impractical for a long time. On the one hand, there was a lack of efficient SAT solvers, and on the other hand structural methods could better exploit the circuit characteristics to speed up the search [16, 17, 26, 39]. Thus, early proposals for SAT-based ATPG did not make their way into industrial practice [28].

However, with the development of advanced SAT solvers [29, 34, 38], SAT-based ATPG has gained increasing attention in recent years [6, 7, 13]. In particular, it has been shown that SAT-based ATPG is more effective for hard to test faults and for redundancy identification [6, 7, 13]. The strength in redundancy identification particularly qualifies SAT-based ATPG for analyzing fault tolerance properties, which often requires to show that faults cannot be detected [23, 24].

As it is beyond the scope of this paper to explain SAT-based ATPG in detail, in the following only the basic steps are briefly summarized. The first step is to construct a SAT instance from a circuit netlist. Here the logic functions of gates are represented in conjunctive normal form (CNF). For example, an AND gate with logic function $c = a \cdot b$ is described by the product $(a + \neg c)(b + \neg c)(\neg a + \neg b + c)$. The CNF for the complete circuit is obtained as the product of all gate CNFs as illustrated in Figure 2. Tseitin transformation achieves this task in linear time and also ensures that the number of terms in the CNF only grows linearly with the circuit size [44].



CNF: $(a + \neg c)\ (b + \neg c)\ (\neg a + \neg b + c)$
$(\neg c + d)\ (\neg b + d)\ (b + c + \neg d)$

Figure 2: Conjunctive Normal Form (CNF).

The next step is fault injection, i.e. constructing the CNF for the combined fault free and faulty circuit introduced above. Finally, the output variable is constrained to 1 by adding it as an additional factor to the CNF. The resulting CNF is then processed by a SAT solver.

Modern SAT solvers typically use the classical Davis–Putnam–Logemann–Loveland (DPLL) search algorithm with additional techniques for improving the efficiency [10, 11]. For example the SAT solver Chaff employs the Variable State Independent Decaying Sum (VSIDS) heuristic [34], which has been adopted also by many other solvers. Other strategies to speed up the search process are for example incremental SAT solving or exploiting thread parallelism in the implementation [8, 29, 38, 41, 46].

3 Self-Checking Circuits

This section briefly summarizes important robustness properties of self-checking circuits, which are in the focus of this paper [18, 27]. To implement a Boolean function as a self-checking circuit, the outputs are encoded as elements of an error detecting code $O \subset \{0,1\}^m$, and a checker is added for monitoring the outputs. If outputs outside the code space O are observed, an error is indicated. The inputs $I \subset \{0,1\}^n$ may span the complete n-dimensional space or may also be encoded. Figure 3 shows an example with parity prediction.

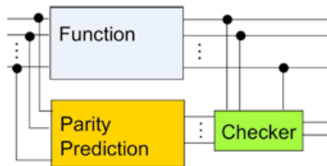


Figure 3: Self-checking circuit with parity prediction.

Encoding and checking the outputs of a circuit does not a priori guarantee the detection of all faults in a given fault model Φ. For a circuit $f: I \rightarrow O$ and an input $i \in I$ let $f(i, \varphi)$ denote the output of f in the presence of fault $\varphi \in \Phi$. If there is an incorrect output $f(i, \varphi) \neq f(i)$ with $f(i, \varphi) \in O$, then the fault is not detected by the checker and Silent Data Corruption (SDC) occurs. Fault secure circuits are a first step towards avoiding SDC.

Definition 1: A circuit is called *fault secure* w.r.t. Φ, if for all faults $\varphi \in \Phi$ and all inputs $i \in I$ the condition $f(i, \varphi) \notin O$ or $f(i, \varphi) = f(i)$ holds.

In a fault secure circuit, a fault $\varphi \in \Phi$ may be masked by all patterns, and thus remain undetected. Although fault secureness guarantees correct operation $f(i, \varphi) = f(i)$, this may not be sufficient in practice. Faults can accumulate over time and invalidate the single fault assumption of Definition 1. To avoid the problems of fault accumulation, it must be guaranteed that all faults can be detected.

Definition 2: A circuit is called *self-testing* w.r.t. Φ, if for all faults $\varphi \in \Phi$ there is an input $i \in I$, such that $f(i, \varphi) \notin O$.

Definition 3: A circuit is called *totally self-checking* w.r.t. Φ, if it is fault secure and self-testing w.r.t. Φ.

Assuming that there is enough time to detect a fault before the next one occurs, totally self-checking circuits completely avoid fault accumulation. The requirement of self-testability can be relaxed, if secure fault accumulation is guaranteed.

Definition 4: A circuit is called *strongly fault secure* w.r.t. Φ, if for all faults $\varphi \in \Phi$

(i) f is self-testing and fault secure w.r.t. φ or

(ii) f is fault secure w.r.t. φ, and if an additional fault $\psi \in \Phi$ occurs, then either (i) or (ii) holds for the multiple fault $\langle\varphi, \psi\rangle$.

Fault secure circuits can be constructed by combining an inverter free design with unidirectional codes [40]. More advanced design strategies are described in [27].

4 Analyzing Fault Tolerance

As pointed out in the introduction, during synthesis the design guidelines for self-checking circuits may be violated, for example by logic sharing. Figure 4 illustrates this problem for the parity prediction circuit of Figure 3.

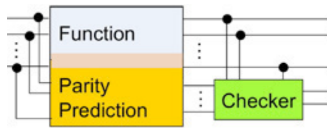


Figure 4: Self-checking circuit with parity prediction: logic sharing violates the design guidelines.

As a consequence, the fault tolerance properties of self-checking circuits must be verified, if synthesis cannot be fully controlled or if design guidelines are relaxed. Most classical approaches rely on fault simulation to exclude undesired behavior in the presence of faults [4, 30, 47]. However, a complete proof then requires exhaustive simulation of all faults and all input patterns. To avoid this, more recent approaches apply BDD-based or SAT-based analysis as well as SAT-based ATPG [15, 21, 22, 23]. Subsection 4.1 explains how fault secureness and self-testability can be checked with SAT-based ATPG while Subsection 4.2 focuses on strong fault secureness.

4.1 Checking Self-Testability and Fault Secureness

As explained below, verifying self-testability and fault-secureness corresponds to ATPG under constraints, which can be ensured by working with testbenches as shown in Figure 5 [22].

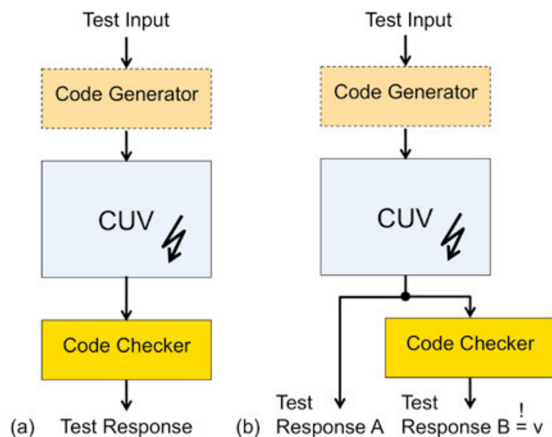


Figure 5: ATPG model for checking self-testability and fault secureness.

To prove the self-testability of the circuit under verification (CUV) according to Definition 2, for each fault $\varphi \in \Phi$ a test pattern $i \in I$ must found, such that $f(i, \varphi) \notin O$. Applying ATPG to the CUV directly may result in test responses $f(i, \varphi) \neq f(i)$ with $f(i, \varphi) \in O$. The code checker in the ATPG model of Figure 5a guarantees fault detection by the checker and thus $f(i, \varphi) \notin O$. If the input space is restricted to a proper subset of the Boolean n-space, e.g. due to encoded outputs of a preceding component, then $i \in I$ is an additional constraint for ATPG, which can be ensured by adding the code generator to the testbench.

Fault secureness of the CUV according to Definition 1 is proven, if there is no fault $\varphi \in \Phi$ which produces a wrong output inside the output code space. Let v denote the output value of the checker which characterizes output code words $o \in O$, e.g. v = “01” or v = “10” in case of a dual rail checker. Then the constraint “B = v” in Figure 5b restricts test responses to code words, and faults can only be detected at output A. If a fault $\varphi \in \Phi$ is detected at output A, then it is a counter-example for for fault secureness, because $f(i, \varphi) \neq f(i)$ and $f(i, \varphi) \in O$ for some input $i \in I$. Thus, the CUV is fault secure, if all faults in the CUV are undetectable in this testbench. As explained in Section 2, SAT-based ATPG is particularly suitable for this task because of its strengths in redundancy identification.

4.2 Grading Strong Fault Secureness

A strongly fault secure circuit is characterized by benign fault accumulation. Verification of strong fault secureness therefore requires the analysis of multiple faults. In [23] SAT-based checking of fault secureness and self-testability as explained in Subsection 4.1 are used as core procedures of an iterative algorithm for grading strong fault secureness.

In the i-th iteration, the algorithm classifies faults of multiplicity i into three groups. If a multiple fault $\varphi \in \Phi$ is self-testable and fault secure, it is classified as “secure”. If it is not fault secure, it is classified as “insecure”, and if it is fault secure, but not self-testing, then it is classified as “unknown”. Secure faults fulfill condition (i) in Definition 4 and can be excluded from further analysis. Insecure faults can lead to SDC even without fault accumulation, therefore they are also dropped from the list. For each unknown fault φ and each single fault $\psi \in \Phi$ the multiple fault $\langle\varphi, \psi\rangle$ of multiplicity $i + 1$ is analyzed in the next iteration. The algorithm stops when all faults are classified or the maximum multiplicity is reached.

As soon as an insecure fault is detected, the circuit is not strongly fault secure. A metric based on the critical multiplicity of faults quantifies the robustness of circuits in this case. The critical multiplicity of a single fault $\varphi \in \Phi$ is defined as $c(\varphi) = i$, if φ becomes part of an insecure fault in the i-th iteration. A high critical multiplicity reflects that the fault is benign with respect to fault accumulation. The maximum value for an insecure fault corresponds to the maximum number of iterations of the algorithm.

To limit the computational effort, the algorithm can be stopped after a user-defined number of iterations i_{max}. In this case, estimates are obtained by assuming all unknown faults as secure or insecure, respectively. If a fault is assumed to be insecure at this step, then its critical multiplicity is set to i_{max}. The experiments in [23] have shown that these optimistic and pessimistic bounds often converge within a few iterations.

The multiple fault analysis can be further improved by by incremental SAT-solving [8]. Whenever an unknown fault is combined with a new single fault ψ, then the SAT-instance for the multiple fault $<\varphi, \psi>$ can be built from the SAT-instance for φ. The achieved speedup in SAT solving allows to increase i_{max} and thus to improve the accuracy of the algorithm.

5 Code Synthesis for Fault-Secure Circuits

Fault-secure circuits may be synthesized either from scratch or by augmenting predesigned and already preverified coreware with additional logic. While the first approach may lead to optimized solutions with respect to area and performance, it may also require a prohibitively high design effort. The latter approach keeps system logic almost untouched, and additional functional verification tasks are not required. Figure 6 depicts the general scheme. Here, the functional logic is not altered at all, and only prediction logic, code generators and comparators are added.

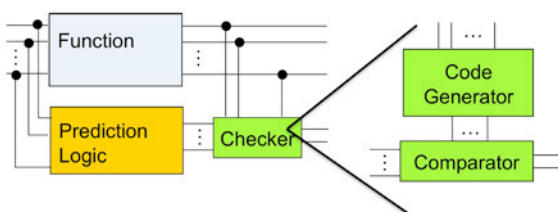


Figure 6: Adding fault secureness to IP cores.

5.1 Code Definition and Parity Prediction

This section focuses on parity codes, where the code generator consists of multiple parity trees, and each circuit output is assigned to one or more parity groups feeding the respective parity trees. Figure 7 shows the parity trees of a Hamming code as an example.

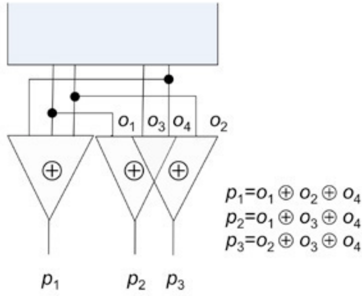


Figure 7: Parity trees of a Hamming code.

According to Definition 1, fault secureness w.r.t. a fault model Φ is achieved in this case, if for all faults $\varphi \in \Phi$ and all inputs $i \in I$ the fault is masked or the parity check provides $P(f(i, \varphi)) \neq P(f(i))$, where P denotes the parity function. The circuit is not fault secure, if a fault $\varphi \in \Phi$ affects the circuit outputs without being detected, i.e. $f(i, \varphi) \neq f(i)$ and $P(f(i, \varphi)) = P(f(i))$. This happens, if logic is shared in the transitive fanin $C(O_p)$ of the core outputs O_p in a parity group and a multibit error of even magnitude is produced at the outputs [18]. For instance in Figure 7, a multiple error at o_1, o_2 and o_3 will lead to even errors in all p_i and cannot be detected.

Definition 5: Circuit outputs o_x and o_y are called *structurally independent*, if their input cones $C(o_x)$ and $C(o_y)$ are disjoint, $C(o_x) \cap C(o_y) = \emptyset$, otherwise *structurally dependent*.

Fault-secureness for all single stuck-at faults can thus be achieved by ensuring structural independence between the outputs in a parity group [18].

One synthesis method is to use a single parity bit and replicate shared logic [12, 42]. As this solution may cause a high area overhead, many schemes work with multiple parity groups and encode each group individually. Touba et al. introduce a greedy algorithm to find a parity code with potentially low area cost [43]. The prediction logic is synthesized together with the original circuit under structural constraints.

During code definition and the synthesis of the prediction logic additional objectives can be followed: For example, concurrent error detection with parity codes has been extended to fault diagnosis and error correction [1], and the approaches in [2] and [19] target a low power implementation. Some other techniques try to detect just as many errors as possible at lower overhead. In [32] and [45] only critical soft errors are addressed. Similarly, the method in [33] focuses on faults with high sensitization probability only. Furthermore, split-parity codes can detect all odd multibit errors with certainty and all even errors with a probability of 50 % [37].

All these techniques may result in significant hardware overhead especially for the prediction logic often exceeding the costs for duplication with comparison. The next subsection presents an approach to minimize this overhead by applying SAT-based ATPG.

5.2 Output Partitioning

As explained in Subsection 5.1, self-checking circuits based on parity codes are fault secure w.r.t. all single stuck-at faults, if the circuit outputs in a parity group are structurally independent. To reduce the area overhead, the number of parity groups should be minimized, and the parity group size should be increased as much as possible. This can be modeled as a graph problem as described in the sequel.

A dependency graph $G = (V, E)$ is constructed with vertices V corresponding to the circuit outputs, and edges E corresponding to pairs of dependent outputs. Then an independent set in the graph defines a parity group with structurally independent circuit outputs.

Definition 6: Let $G = (V, E)$ be an undirected graph. A subset $W \subset V$ of vertices is called an *independent set*, if for all $v, w \in W$ the condition $(v, w) \notin E$ holds.

A partitioning $\mathcal{P}_k$ of V into k independent sets $V_1, ..., V_k$ provides a parity code, which guarantees fault secureness w.r.t. all single stuck-at faults. Yet even logic sharing of outputs within a single group can be allowed as long as the fault effects are propagated to an odd number of outputs covered by another parity group. This relaxation increases the search effort and helps to reduce the area overhead. Figure 8 illustrates these constructions.

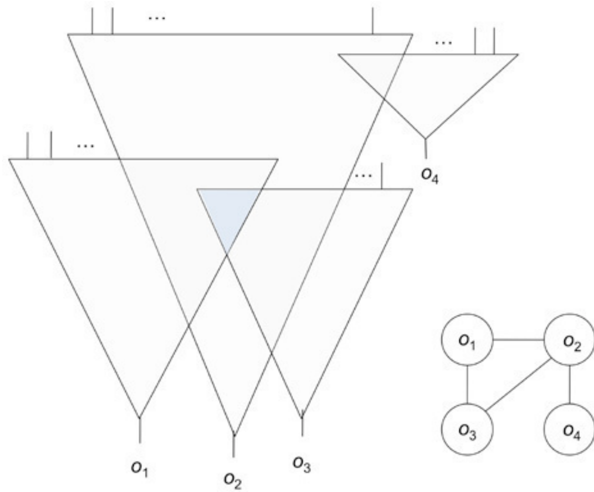


Figure 8: Output cones and dependency graph.

On the left side partially overlapping circuit cones with outputs o_1, ..., o_4 are shown. The corresponding dependency graph is depicted on the right side. The analysis of the dependency graph leads to three independent sets which define the output groups to be observed: $V_1 = \{o_2\}$, $V_2 = \{o_1, o_4\}$, and $V_3 = \{o_3\}$. However, faults in the intersection of the output cones $C(o_1)$ and $C(o_3)$ (shadowed area) may also be detected at output o_2. In this case, the observation of two groups $V_1 = \{o_2\}$, $V_2' = \{o_1, o_3, o_4\}$ is sufficient, and hardware can be saved. These savings come at the cost of some uncertainties, as a fault in the shadowed area may not be observed at o_2 but at both o_1 and o_3 where the error signal may be masked.

Hence, the entire code synthesis consists of three steps:

1) Prepartition the outputs into parity groups.

2) For each fault in overlapping cones of two outputs in one group, prove the fault secureness property.

3) For each counter example modify the partitioning by further partitioning and reordering.

For non-redundant circuits with full input space the resulting circuit is also self-testing and thus totally self-checking. In this case fault accumulation can be handled as described in Section 3.

5.3 Code Synthesis

For the first synthesis step, a greedy procedure can be applied which provides a solution $\mathcal{P}_k = \{V_1, ..., V_k\}$ with small k, such that:

1) $\biguplus_{1 \le \kappa \le k} V_\kappa = V$, and

2) $\bigwedge_{x \neq y \in V_\kappa} \left(\left(C(x) \cap C(y) = \varnothing\right) \vee \bigvee_{z \in V \setminus V_\kappa} \left(C(x) \cap C(y) \subset C(z)\right) \right)$.

This prepartitioning delivers a small number of parity groups as starting point for the fault secure synthesis. Although it cannot guarantee fault-secureness yet, it reduces the runtime of the formal analysis in the second step.

The analysis of the overlapping output cones in the second step corresponds to finding counter examples for fault secureness. With the techniques introduced in Subsection 4.1, this problem can be mapped to a SAT-based ATPG problem using the testbench of Figure 5b. Overall, the set of all insecure faults $\Phi_{insecure}$ is obtained in this step.

Parity splitting in the third step relies on simulating insecure faults. For each insecure fault $\varphi \in \Phi_{insecure}$ an input pattern $i \in I$ is selected with $f(i, \varphi) \neq f(i)$ and $P(f(i, \varphi)) = P(f(i))$. This input pattern is denoted by $i(\varphi)$. Then all insecure faults $\varphi \in \Phi_{insecure}$ are simulated with the respective pattern $i(\varphi)$ to find the outputs and parity groups to which the fault effect is propagated. Let $V_\kappa(\varphi)$ denote the subset of outputs in V_κ to which the fault effect of φ propagates under $i(\varphi)$. To avoid SDC, $V_\kappa(\varphi)$ and thus V_κ must be split into at least two groups, such that the fault effect is not masked anymore.

There are $2^{|V_\kappa(\varphi)|-1}-1$ possibilities of splitting V_κ appropriately. For each case the number of faults with re-established fault secureness is determined by SAT-based analysis. Then the group and splitting with highest reduction of insecure faults is selected. The result is one additional parity group. Since new groups are created, a topological analysis is performed after each splitting to investigate whether adding some output to available groups can further reduce the number of insecure faults. An output may be included in an additional group, if it does not share any logic with the outputs in that group.

For large $V_\kappa(\varphi)$ the explicit analysis of each candidate splitting is too computationally expensive. In this case, a topological analysis is used to split V_κ into its independent set V_κ^{ind} and the complement $V_\kappa \setminus V_\kappa^{ind}$ which has minimum logic sharing in its input cones.

The process of analyzing overlapping cones and parity group splitting proceeds until no more faults violate the fault secureness conditions. The parity prediction and checker logic are synthesized separately from the functional circuit. As the checker must be self-checking, cascadable two-rail checkers with six logic gates may be used to build the checker for *k* parity groups [27]. As the results in [9] show, the resulting circuit is fault secure with an area overhead which is significantly lower than the overhead for the approaches reported in the literature so far [2, 31].

6 Conclusions

Verification and synthesis of self-checking circuits is strongly related to test generation and redundancy identification. SAT-based ATPG is particularly suitable for this application domain because of its strengths in dealing with hard-to-detect faults and redundancy identification.

7 Acknowledgement

A large part of this work has been performed in close cooperation with the group of Bernd Becker within the framework of the DFG project RealTest under WU 245/5-2 and HE 1686/3-2.

References

[1] S. Almukhaizim and Y. Makris, "Fault tolerant design of combinational and sequential logic based on a parity check code," Proceedings IEEE International Symposium on Defect and Fault Tolerance (DFT'03), Cambridge, MA, USA, 2003, pp. 563-570

[2] S. Almukhaizim, P. Drineas, and Y. Makris, "Entropy-driven paritytree selection for low-overhead concurrent error detection in finite state machines," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 25, No. 8, August 2006, pp. 1547-1554

[3] R. Baumann, "Soft errors in advanced computer systems," IEEE Design & Test of Computers, Vol. 22, No. 3, 2005, pp. 258-266

[4] C. Bolchini, et al., "The design of reliable devices for mission-critical applications," IEEE Trans. on Instrumentation and Measurement, Vol. 52, Dec. 2003, pp. 1703-1712

[5] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," IEEE Micro, Nov. 2005, pp. 10-16

[6] A. Czutro, et al., "Tiguan: Thread-parallel integrated test pattern generator utilizing satisfiability analysis," Proceedings International Conference on VLSI Design (VLSID'09), New Delhi, India, 2009, pp. 227-232

[7] A. Czutro, et al., "Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability Analysis," International Journal of Parallel Programming, Vol. 38, June 2010, pp. 185–202

[8] A. Czutro, et al., "SAT-ATPG using preferences for improved detection of complex defect mechanisms," Proceedings 30th IEEE VLSI Test Symposium (VTS'12), Maui, Hawaii, USA, April 2012, pp.170-175

[9] A. Dalirsani, M. Kochte, H.-J. Wunderlich, "SAT-based Code Synthesis for Fault-Secure Circuits," Proceedings 16th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'13), New York City, NY, USA, October 2013, pp. 39-44

[10] M. Davis, G. Logemann, D. Loveland, "A Machine Program for Theorem-Proving," Communications of the ACM, Vol. 5, 1962, pp 394-397

[11] M. Davis and Hilary Putnam, "A Computing Procedure for Quantification Theory," Journal of the ACM, Vol. 7, No. 3, 1960, pp. 201-215

[12] K. De, et al., "RSYN: a system for automated synthesis of reliable multilevel circuits," IEEE Transactions on VLSI Systems, Vol. 2, No. 2, June 1994, pp. 186-195

[13] R. Drechsler, et al., "On Acceleration of SAT-Based ATPG for Industrial Designs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 27, No.7, July 2008, pp.1329-1333

[14] D. Ernst, et al., "Razor: Circuit-Level Correction of Timing Errors for Low Power Operation," IEEE Micro, Vol. 24, No. 6, Nov.-Dec. 2004, pp. 10-20

[15] G. Fey, et al., "Effective robustness analysis using bounded model checking techniques," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 30, No. 8, August 2011, pp. 1239-1252

[16] H. Fujiwara, T. Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Transactions on Computers, Vol.C-32, No.12, Dec. 1983, pp.1137-1144

[17] P. Goel, "An implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Transactions on Computers, Vol. 30, No. 3, March 1981, pp. 215-222

[18] M. Goessel, et al., "New Methods of Concurrent Checking," Springer, 2008

[19] S. Ghosh, N. Touba, and S. Basu, "Synthesis of low power CED circuits based on parity codes," Proceedings. IEEE VLSI Test Symposium (VTS'05), Palm Springs, CA, USA, 2005, pp. 315-320

[20] P. Gupta, et al., "Underdesigned and Opportunistic Computing in Presence of Hardware Variability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.32, No.1, Jan. 2013, pp. 8-23

[21] J. P. Hayes, I. Polian, and B. Becker, "An Analysis Framework for Transient-Error Tolerance," Proceedings IEEE VLSI Test Symposium (VTS'07), xx, CA, USA, 2007, pp. 249-255

[22] M. Hunger and S. Hellebrand, "Verification and analysis of self-checking properties through ATPG," Proceedings 14th IEEE International On-Line Testing Symposium (IOLTS'08), Rhodes, Greece, July 2008, pp. 25-30

[23] M. Hunger, et al., "ATPG-Based Grading of Strong Fault-Secureness," IEEE International On-Line Testing Symposium 2009 (IOLTS'09), Sesimbra-Lisbon, Portugal, June 2009, pp. 269-274

[24] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," IEEE Transactions on Computers, Vol. C-24, No. 3, March 1975, pp. 242-249

[25] I. Koren and C. M. Krishna, "Fault-Tolerant Systems," San Francisco, CA, USA: Morgan-Kaufman Publishers, 2007

[26] W. Kunz and D. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems-Test, Verification, and Optimization," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 13, No. 9, 1994, pp. 1143-1158

[27] P. K. Lala, "Self-Checking and Fault-Tolerant Digital Design," Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001

[28] T. Larrabee, "Test pattern generation using boolean satisfiability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 11, No. 1, Jan 1992, pp. 4-15

[29] M. Lewis, T. Schubert, B. Becker, "Multithreaded SAT Solving," Proceedings Asia and South Pacific Design Automation Conference (ASP-DAC'07), Yokohama, Japan, Jan. 2007, pp.926-931

[30] J.-C. Lo and E. Fujiwara, "Probability to achieve TSC goal," IEEE Transactions on Computers, Vol. 45, No. 4, April 1996, pp. 450-460

[31] S. Mitra and E. McCluskey, "Which concurrent error detection scheme to choose?" Proceedings IEEE International Test Conference (ITC'00), Atlantic City, NJ, USA, 2000, pp. 985-994

[32] K. Mohanram and N. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," Proceedigns IEEE International Test Conference (ITC'03), Charlotte, NC, USA, 2003, pp. 893-901

[33] K. Mohanram, et al., "Synthesis of low-cost parity-based partially self-checking circuits," Proceedings 9th IEEE On-Line Testing Symposium (IOLTS'03), Kos Island, Greece, July 2003, pp. 35-40

[34] M. W. Moskewicz, et al., "Chaff: engineering an efficient SAT solver," Proc. Design Automation Conf. (DAC'01), Las Vegas, NV, USA 2001, pp. 530-535

[35] M. Nicolaidis, "Time Redundancy Based Soft-Error Tolerant Circuits to Rescue Very Deep Submicron," Proc. 17th IEEE VLSI Test Symposium, San Diego, CA, USA, April 1999

[36] M. Nicolaidis, "GRAAL: A New Fault Tolerant Design Paradigm for Mitigating the Flaws of Deep Nanometric Designs," Proceedings IEEE International Test Conference (ITC'07), San Jose, USA, October 2007, pp. 1-10

[37] M. Richter and M. Goessel, "Concurrent checking with split-parity codes," Proceedings IEEE International On-Line Testing Symposium (IOLTS'09), Sesimbra-Lisbon, Portugal, June 2009, pp. 159-163

[38] T. Schubert, M. Lewis, and B. Becker, "antom – Solver Description," in SAT Race, 2010

[39] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 7, No. 1, January 1988, pp. 126-137

[40] J. E. Smith and G. Metze, "Strongly Fault Secure Logic Networks," IEEE Transactions on Computers, Vol. C-27, No. 6, June 1978, pp. 491-499

[41] *D. Tille, S. Eggersglüß and R. Drechsler, "Incremental Solving Techniques for SAT-Based ATPG," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 29, No. 7, July 2010, pp. 1125-1130*

[42] *N. A. Touba and E. J. McCluskey, "Logic synthesis for concurrent error detection," Tech. Rep. 93-6, Center for Reliable Computing, Stanford Univ., Stanford, CA, USA, Nov. 1993*

[43] *N. Touba and E. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 7, July 1997, pp. 783-789*

[44] *G.S. Tseitin, "On the Complexity of Derivations in Propositional Calculus," in Studies in Constructive Mathematics and Mathematical Logics (A. Slisenko, Ed.), 1968*

[45] *R. Vemu, et al., "A low-cost concurrent error detection technique for processor control logic," Proceedings Design, Automation and Test in Europe (DATE'08), 2008, Munich, Germany, pp. 897-902*

[46] *J. Whittemore, J. Kim and K. A. Sakallah, "SATIRE: A New Incremental Satisfiability Engine," Proceedings Design Automation Conference (DAC'01), Las Vegas, NV, USA pp. 542–545, June 2001*

[47] *S. Zhang and J. C. Muzio, "Evaluating the safety of self-checking circuits," Journal of Electronic Testing – Theory and Applications (JETTA), Vol. 6, No. 2, 1995, pp. 243-253*

Prof. Dr. Sybille Hellebrand
chairs the Computer Engineering Group (DATE) at the Institute of Electrical Engineering and Information Technology at the University of Paderborn. Her research interests are in the area of test, diagnosis, and fault tolerance of integrated circuits and systems.

Address: Computer Engineering Group, EIM-E, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
Mail: sybille.hellebrand@uni-paderborn.de

Prof. Dr. Hans-Joachim Wunderlich
is Director of the Institute of Computer Architecture and Computer Engineering (ITI) at the University of Stuttgart. He has authored and co-authored more than 200 publications in the area of test, reliability, and fault tolerance.

Address: Institut of Computer Engineering (ITI), University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
Mail: wu@informatik.uni-stuttgart.de