Int J Parallel Prog
DOI 10.1007/s10766-009-0124-7

Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability Analysis

Alexander Czutro · Ilia Polian · Matthew Lewis · Piet Engelke · Sudhakar M. Reddy · Bernd Becker

Received: 21 July 2009 / Accepted: 14 December 2009
© Springer Science+Business Media, LLC 2009

Abstract Efficient utilization of the inherent parallelism of multi-core architectures is a grand challenge in the field of electronic design automation (EDA). One EDA algorithm associated with a high computational cost is automatic test pattern generation (ATPG). We present the ATPG tool TIGUAN based on a thread-parallel SAT solver. Due to a tight integration of the SAT engine into the ATPG algorithm and a carefully chosen mix of various optimization techniques, multi-million-gate industrial circuits are handled without aborts. TIGUAN supports both conventional single-stuck-at faults and sophisticated conditional multiple stuck-at faults which allows to generate patterns for non-standard fault models. We demonstrate how TIGUAN can be combined with conventional structural ATPG to extract full benefit of the intrinsic strengths of both approaches.

Keywords Thread-parallel SAT · SAT-based automatic test pattern generation

1 Introduction

Semiconductor manufacturing processes are yield processes: a significant fraction of manufactured microchips are defective [1]. To prevent delivery of such defective products to the customer, every circuit is tested by special automated test equipment.

A. Czutro · I. Polian (✉) · M. Lewis · P. Engelke · B. Becker
Computer Architecture Group, Institute for Computer Science, Albert-Ludwigs-University, Georges-Köhler-Allee 51, 79110 Freiburg i. Br., Germany
e-mail: polian@informatik.uni-freiburg.de; ilia@polian.de

S. M. Reddy
ECE Department, University of Iowa, Iowa City, IA, 52242, USA

Published online: 01 January 2010

This equipment applies input vectors called test patterns to the chip and compares the responses calculated by the chip with pre-computed reference responses. The circuit passes the test if all responses match.

The testing process is associated with significant cost which may reach up to 40% of the total manufacturing cost according to the International Technology Roadmap for Semiconductors. To reduce this cost, test patterns are generated by tools called automatic test pattern generators (ATPG) [17,19–21,34]. Although test generation is an NP-complete problem, state-of-the-art test generators are able to handle large industrial multi-million-gate designs.

ATPG tools use the notion of a *fault model* which is an abstraction of actual manufacturing defects. For instance, the (single-)stuck-at fault model assumes that one circuit line is permanently stuck at 0 or 1 due to a defect. Given a *fault list* (e.g., the complete list of all stuck-at-0 and stuck-at-1 faults in the circuit), an ATPG tool would try to generate a compact set of test patterns (*test set*) which detects the faults in the fault list. If no test pattern is found for a fault, an ATPG tool attempts to prove that this fault is *redundant* and no test pattern could detect it. Undetected faults not proven redundant are considered *aborted* or *unclassified*.

The advent of multi-core architectures raises the question how ATPG, among other electronic design automation (EDA) algorithms, can benefit from these architectures in an optimal way. Significant research on distributed ATPG has been performed in the past [18]. However, it was based on the general assumption that communication between processor nodes is very expensive. This assumption is not necessarily true for multi-core architectures where individual cores are located in physical proximity to each other, sharing fast caches, or are connected by fast buses.

Traditional deterministic automatic test pattern generation (ATPG) algorithms work directly on the circuit structure [17,20,21,34], possibly in conjunction with additional data structures such as implication graphs [44] or advanced techniques to prune the solution space [19,45]. It has long been known that an ATPG problem can be reduced to a Boolean satisfiability (SAT) instance and solved using a SAT solver [28,43]. However, this approach has not become widely adopted as the structural approaches tended to exhibit better performance.

It has recently been shown that SAT-based ATPG outperforms structural approaches for several classes of faults [10]. One such class consists of redundant faults. SAT solvers are routinely used to prove unsatisfiability in applications such as equivalence checking [26], and a number of techniques have been developed to quickly prune large parts of the solution space. In contrast, structural ATPG methods may need to traverse almost the complete solution space to make sure that no test pattern for a fault exists. It has also been reported that there are testable faults for which structural ATPG performs a large number of backtracks to find a pattern while SAT-based ATPG swiftly finds a solution [10].

The ability to handle redundant faults is becoming more important for two reasons. First, defects in nanoscale manufacturing technologies may not be described adequately by stuck-at faults [2]. Non-standard fault models such as resistive bridging faults [15,33] or interconnect opens [23,37] may impose very specific conditions on the lines in the circuit, which are, in many cases, impossible to satisfy, so the fault is undetectable. Second, redundant structures are being increasingly used to enhance

circuit reliability and yield [40,46]. A significant fraction of faults in these structures are not detectable. To accurately estimate the defect coverage, the proof that the fault in question is undetectable (rather than aborted) is essential.

In this article we present the ATPG tool TIGUAN (Thread-parallel Integrated test pattern Generator Utilizing satisfiability Analysis) which is based on the in-house SAT solver MiraXT [29]. MiraXT is a state-of-the-art SAT solver which incorporates various optimization techniques developed in the last few years. Moreover, it supports thread parallelism, thus fully utilizing the performance of multi-processor systems or multi-core processors. In contrast to other existing tools [10,16], TIGUAN is tightly coupled with the SAT engine and can dynamically control its internal parameters such as which preprocessing steps are performed or number of threads to be used. Additionally, we present a two-phase method which utilizes MiraXT's inherent parallelism without wasting too much time for thread initialization on easy instances.

Another feature of TIGUAN is the support of the general *conditional multiple-stuck-at* (CMS@) fault model. The model allows faulty effects to be present on multiple circuit lines (victims) simultaneously if a number of conditions on other lines (aggressors) are satisfied. Many static non-standard fault models can be mapped to conditional multiple-stuck-at faults, making TIGUAN a flexible tool to handle various defect classes.

Experiments demonstrate that TIGUAN can generate complete stuck-at test sets for large industrial circuits with up to several million gates without aborts. For two classes of non-standard fault models (represented by CMS@ fault lists) TIGUAN completely classifies all ISCAS and ITC benchmarks and most industrial circuits. TIGUAN also outperforms earlier SAT-based ATPG tools.

The remainder of the article is organized as follows. The CMS@ fault model and the mapping of other fault models to the CMS@ fault model is introduced in the next section. Section 3 introduces SAT-based ATPG, gives the overall flow of TIGUAN, and explains how parallelism is integrated. Experimental results using up to 16 threads are reported in Sect. 4 for stuck-at faults as well as more complex faults mapped to CMS@ faults. Section 5 concludes the article.

2 CMS@ Fault Model

TIGUAN incorporates the *conditional multiple-stuck-at* (CMS@) fault model which includes the standard single-stuck-at fault model and is related to generic fault modeling approaches such as fault tuples [9] or the Generalized Fault Model [27]. A CMS@ fault with r *aggressor lines* and s *victim lines* consists of a list $\{a_1/a_1^{val}, \ldots, a_r/a_r^{val}\}$ and a list $\{v_1/v_1^{val}, \ldots, v_s/v_s^{val}\}$, where each a_i and each v_j denotes a signal line and all a_i^{val} and v_j^{val} stand for a logical value (0 or 1). A circuit under a CMS@ fault exhibits faulty behavior under any input vector which sets every aggressor line a_i to a_i^{val}. In this case, the value on each victim line v_j changes to v_j^{val}.

A single-stuck-at-fault is represented by a CMS@ fault with an empty aggressor list and a victim list consisting of one entry. In the following, we explain the mapping of other fault models to CMS@ faults.

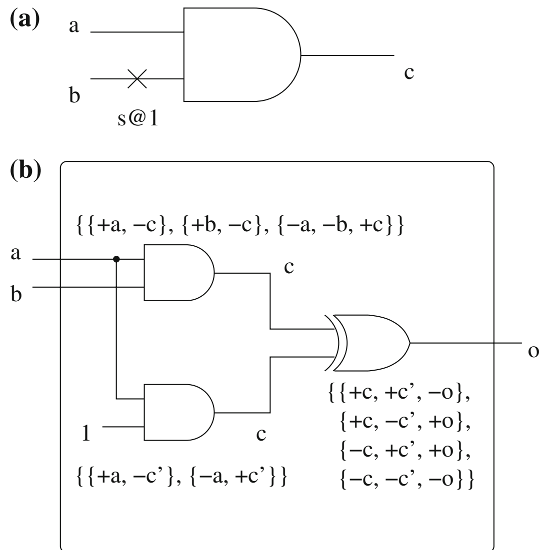


Fig. 1 Example circuit (**a**) and its miter circuit (**b**)

2.1 Gate-Exhaustive Testing

Gate-exhaustive testing requires that every single-stuck-at fault at the output of a gate is detected using all valid value combinations on the inputs of that gate [6]. A stuck-at-1 fault at the output of an AND2 gate would be tested independently by three patterns, one justifying 00 at the gate's inputs, one justifying 01 and one justifying 10. Gate-exhaustive testing was demonstrated to be effective in identifying hard-to-detect defects on actual manufactured silicon [6]. Generally, $2^n - 1$ test patterns must be generated for a stuck-at-1 fault at the output of an n-input AND or NOR gate and for a stuck-at-0 fault at the output of a NAND or OR gate. One pattern must be generated for the opposite stuck-at fault, respectively. 2^{n-1} patterns must be generated for a stuck-at-1 or a stuck-at-0 fault at the output of an XOR or XNOR gate.

Gate-exhaustive testing is easily mapped to the CMS@ fault model. For instance, testing the stuck-at-1 fault at line c of the circuit in Fig. 1a requires the detection of three CMS@ faults: f_1 with $A = \{a/0, b/1\}$, $V = \{c/1\}$; f_2 with $A = \{a/1, b/0\}$, $V = \{c/1\}$, and f_3 with $A = \{a/0, b/0\}$, $V = \{c/1\}$. Similar transformations are performed for other gate types.

2.2 Resistive Bridging Faults

Bridging faults with non-zero bridge resistance may impact the behavior of a digital circuit in a non-trivial way [15,33]. In general, a short defect with a non-zero resistance R_{sh} between interconnects a and b imposes intermediate voltages V_a and V_b between 0 and V_{DD} on the affected interconnects. These voltages are interpreted as logic values by the gates driven by a and b, depending on the logic thresholds of the gates. To detect a resistive short defect with a given resistance, specific values (detection conditions)

on the gates driving the shorted interconnects may be required, and the fault effect may be visible on one or multiple gates driven by the shorted interconnects.

These detection conditions may differ for short defects which involve the same pair of interconnects but have different resistances R_{sh}. It was shown in [13,39] that for every pair of interconnects a and b there is a finite number of representative resistances R_{sh} such that a test set which detects all short defects with these resistances covers all possible short defects between a and b. It is possible to formulate CMS@ faults which correspond to short defect with representative resistances (the mapping is discussed in [13] and is omitted here for brevity).

3 Tiguan

Given a circuit, a CMS@ fault list and a set of parameters which includes a timeout value, TIGUAN generates a test set which detects all faults for which a test pattern could be found within the time budget. All faults in the lists are classified as either detected, undetectable, or aborted (not classified within the time budget).

3.1 Test Generation Procedure

TIGUAN selects a fault from the fault list and attempts to generate a pattern for this fault by formulating a SAT instance in conjunctive normal form (CNF) and handing it to the MiraXT engine (described below). A miter circuit consisting of the fault-free circuit and the circuit with the fault injected, both connected to an XOR network, is constructed and represented in conjunctive normal form (CNF). Figure 1b shows the miter circuit and the corresponding CNF parts for the fault in Fig. 1a. Note that in CNF format, $\{\{+a, -c'\}, \{-a, +c'\}\}$ corresponds to $(a \vee \neg c') \wedge (\neg a \vee c')$, etc. We employ the D-chain technique [43] to speed up the computation. For every circuit line l through which the fault effect could be propagated, a new variable D_l is introduced which equals 1 if and only if line l is sensitized, i.e., the values in the fault-free and the faulty sub-circuits of the miter circuit differ. New clauses are added to ensure the existence of a propagation path, i.e., at least one output o must be sensitized ($D_o = 1$) and every gate with a sensitized output must have at least one sensitized input. The clauses belonging to the D-chain are not shown in Fig. 1b for the sake of simplicity.

The CNF formula is handed to the thread-parallel SAT solver engine MiraXT [29] along with a timeout. If MiraXT finds a model (i.e., a satisfying variable assignment) of the SAT instance, the test pattern is derived from the solution. If MiraXT reports that the instance is unsatisfiable, the fault is proven to be undetectable. TIGUAN can be started in the fault-dropping mode; all yet-undetected faults in the fault list are simulated with generated patterns and covered faults are marked detected and excluded from further processing. We employ an in-house 32-bit pattern-parallel CMS@ fault simulator, so fault dropping is invoked after 32 new patterns have been accumulated. The ATPG process is continued until all faults have been classified.

It is possible to speed up the computation by a technique known as *incremental solving*. Significant parts of CNFs generated for different faults are identical, and such common parts can be reused in multiple invocations of the SAT solver. We performed

extensive experiments with incremental solving and found that our current implementation does not benefit from it. Currently, we perform structural analysis to exclude from the CNF circuit parts which cannot be affected by the considered fault. For incremental solving, larger parts of the circuit must be included in the CNF. Overall, the overhead associated with generating the larger CNFs exceeded the time savings due to incremental solving in our experiments. Hence, we currently do not employ incremental solving. The performance of MiraXT is tuned by adjusting several solver-internal control variables to values appropriate for ATPG instances.

TIGUAN also provides a mode in which the percentage of don't-cares (Xes) in the generated patterns is maximized. An X in a test pattern indicates that the pattern will detect the fault irrespective of the logic value (0 or 1) at this bit position. Test patterns with a large number of Xes are desirable for two reasons. First, the Xes can be assigned values which are useful to detect other faults, thus reducing the number of patterns in the test set (test compaction [31]). Second, the freedom to assign the Xes is essential to compress the patterns and thus reduce the test data volume (test compression [32]). The injection of Xes is performed by the SAT engine; on top of that, an input-output-cone analysis similar to [11] is performed to identify further Xes. We are currently integrating more elaborate methods of test set relaxation [12,24] into TIGUAN to achieve very high don't-care densities comparable to percentages obtained by structural ATPG approaches.

3.2 Multi-Threaded Solving

Parallel test pattern generation requires an intelligent partitioning of the problem being solved into smaller sub-problems and distribution of these sub-problems to individual threads. To allow the threads to run and cooperate efficiently, an appropriate representation of the data shared among the threads and an efficient mechanism to access this data are required. In the following, we first describe the data organization and then provide an overview of how TIGUAN partitions the test generation problem being solved.

3.2.1 Distributed Data Organization

There are two basic paradigms to implement search-space parallelism: message passing and shared memory. Message passing is typically used on classical workstation clusters. The computation is divided into multiple processes which may run on different processors and communicate with each other by sending messages according to a protocol such as MPI [42]. Shared memory refers to physical storage which can be accessed by all threads running on multiple processors. It is generally assumed that message passing requires relatively long time and provides limited bandwidth, but is scalable to a large number of processors. In contrast, shared memory can provide data rates comparable with 'regular' memory in a uniprocessor environment. This is particularly the case for state-of-the-art multi-core systems which include sophisticated mechanisms to accelerate accesses to a core's local memory by other cores located on separate processors. A shared level-3 cache is used for most communication between

cores on the same processor die. The drawback of a shared-memory solution is congestion when the number of different threads simultaneously accessing the memory is high.

TIGUAN implements parallelism based on the shared memory paradigm. There have been two reasons for this design decision. First, the volume of the data communicated is relatively high. Second, the target environments for TIGUAN are multi-core processors rather than workstation clusters. The number of cores in today's processors is not very large, and mechanisms to support shared-memory accesses are provided. Note that a version of MiraXT designed for clusters of multi-core workstations which uses both shared-memory and message passing exists [38] but is not used in this work.

MiraXT and thus TIGUAN maintains one common *Shared Clause Database* which provides access to the full instance for all threads. Note that clauses representing the fault-free circuit, the circuit with the fault injected and auxiliary clauses from the D-chains are treated equally. During solving, every thread generates *conflict clauses* which indicate parts of the search space which contain no solution and can be excluded from the consideration. These conflict clauses are communicated to other threads by inserting them into the Shared Clause Database (knowledge sharing). Furthermore, conflict clauses may become obsolete, so clause deletion has to be performed. Both clause insertion and deletion are implemented by optimized algorithms which minimize the need for locks and reduce the performance overhead due to lock conflicts to fractions of a per cent. Details on clause insertion and deletion can be found in [38].

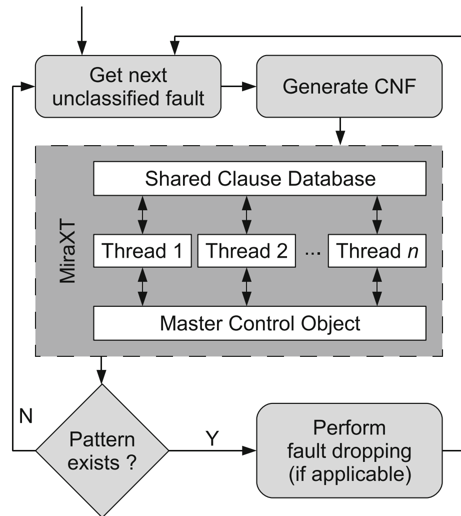
Every thread keeps in its local memory a *Watched Literals Reference List*, which contains selected literals of each clause. Empirical data has shown that this restricted information is sufficient for most solving operations of a thread, resulting in a good cache utilization. If a thread requires the complete clause information, it must access the Shared Clause Database, which may require inter-processor communication. Special data structures are provided for a thread to quickly recognize clauses which have been inserted into the Shared Clause Database since the thread's last access to it [38].

3.2.2 Problem Partitioning and Solving

When run in single-threaded mode (no parallelism), TIGUAN hands the generated CNF to the MiraXT engine, which performs Davis-Putnam-style SAT solving [8]. Necessary assignments to variables (e.g., unit clauses) are identified, a *decision variable* is selected and assigned, and implications of this decision are collected using *Boolean Constraint Propagation*. In general, the decision may lead to a conflict (inconsistency). Then, a conflict clause is generated to avoid entering the inconsistent part of the solution state later on, and one or several decisions are undone (backtracking). This process is continued until either all variables are assigned and consistent (satisfying assignment has been found), or a conflict is identified with no option to backtrack (the instance is proven unsatisfiable).

Parallel SAT solving dynamically divides the solution space and assigns sub-problems to individual threads. This is managed by the *Master Control Object* (MCO) of very limited complexity. MCO essentially makes messages available to threads and does not intervene with a thread's computation process. MCO also

Fig. 2 TIGUAN's flow



manages running and idle threads which are waiting for new sub-problems. The flow of TIGUAN in multi-threaded mode is shown in Fig. 2.

After CNF generation, the multi-threaded solver starts by giving the complete instance to one of the threads, and it begins the solving process. All other threads communicate to the MCO that they are idle. Idle threads are put into sleep mode in which they do not poll and consequently do not cause communication overhead. Running threads poll the MCO periodically whether any global events have occurred.

Possible global events are 'instance has been solved by another thread', 'timeout has been exceeded', and 'idle threads exist'. In the latter case, the running thread divides its sub-problem into two parts, wakes up one of the sleeping threads and transfers control of one part to this thread. If a thread's sub-problem is unsatisfiable, it re-enters the idle state. The problem is proven unsatisfiable if all threads become idle.

3.3 Two-Stage Method

It has been noted, e.g., in [21], that sophisticated performance enhancements are effective for relatively few hard-to-detect faults while slowing down the processing of easy-to-detect faults. We observed that, with average SAT solving time per fault below 0.1 s for most circuits, various optimizations do not result in a net run-time gain. This is also true for thread parallelism: the overhead to initialize the threads and set up the communication infrastructure does not appear to be justified for most faults.

Consequently, we implemented the two-stage ATPG strategy shown in Fig. 3. In the first stage, easy-to-detect faults are processed using low-overhead procedures, and in the second stage the full computational power of TIGUAN is applied to the residual hard-to-detect faults. If a structural ATPG is available, it can be used as the first-stage tool. Otherwise, TIGUAN is run in the single-thread mode with an aggressive time limit, e.g., 1 s. Structural ATPGs tend to be faster for easy-to-detect faults, and they

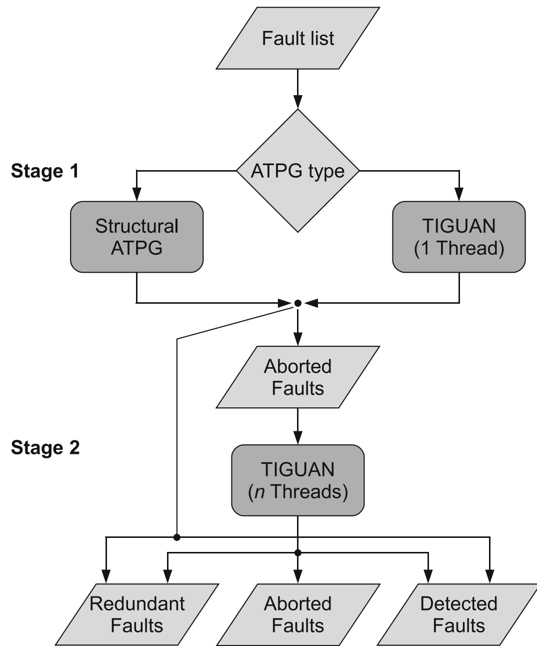


Fig. 3 Two-stage fault classification flow

often produce very compact test sets because they incorporate sophisticated test compaction algorithms, yet they typically classify less faults then SAT-based ATPGs. In the second stage, TIGUAN employing thread parallelism is applied to the hard-to-detect faults aborted by the first-stage tool. The solving times for these faults tend to be much higher than the thread initialization overhead, so that multi-threaded solving provides net benefits.

4 Experimental Results

TIGUAN was applied to ISCAS 85 circuits [5] and combinational cores of ISCAS 89 circuits [4], ITC 99 circuits [7] and industrial circuits provided by NXP. The measurements for stuck-at faults (Tables 1, 2, 3 and 4) were performed on a 2.8 GHz AMD Opteron computer with 16 GB RAM, the measurements for up to 16 threads (Fig. 4) were performed on an AMD Opteron system with four quad-core processors with 64 GB RAM, and the measurements for non-standard fault models (Tables 5 and 6) were performed on a 2.3 GHz machine with 4 GB RAM (TIGUAN is a 32-bit application which uses only up to 4 GB RAM).

4.1 Single-Threaded Single-Stuck-at ATPG

Table 1 reports ATPG results for industrial circuits using fault dropping and 20 s time-out per fault (a fault was classified as aborted if no pattern was found within 20 s). The name of the circuit, the number of gates and collapsed faults and the distribution

Table 1 Results of single-threaded TIGUAN for stuck-at faults with fault dropping for NXP circuits, timeout 20 s per fault

Circuit	Gates	Faults	Det.	Red.	Ab.	Pat.	Time per fault (s)			T (s)
							CNF	SAT	FSIM	
p35k	48,927	67,733	66,721	1,012	0	11,536	0.033	0.0278	0.0007	1,364
p45k	46,075	68,760	68,564	196	0	3,604	0.005	0.0017	0.0008	47
p77k	75,033	1,20,348	1,13,049	7,299	0	5,318	0.029	0.3455	0.0510	5,454
p78k	80,875	1,63,310	1,63,310	0	0	468	0.005	0.0006	0.0061	7
p81k	96,722	2,04,174	2,02,981	1,193	0	7,529	0.010	0.0017	0.0015	162
p89k (*)	92,706	1,50,538	1,48,604	1,934	0	9,868	0.007	0.0015	0.0018	154
p100k	1,02,443	1,62,129	1,61,404	725	0	5,142	0.006	0.0032	0.0028	91
p141k (*)	1,85,360	2,82,428	2,79,189	3,239	0	8,893	0.050	0.0337	0.0024	1,706
p267k	2,96,404	3,66,871	3,65,423	1,448	0	11,579	0.020	0.0031	0.0037	447
p269k (*)	2,97,497	3,69,055	3,67,607	1,448	0	11,633	0.018	0.0031	0.0046	436
p286k (*)	3,73,221	6,50,368	6,40,103	10,264	1	20,243	0.041	0.0490	0.0062	3,456
p295k (*)	3,11,901	4,72,022	4,68,174	3,847	1	22,786	0.024	0.0053	0.0042	1,159
p330k	3,65,492	5,40,758	5,35,070	5,656	32	23,392	0.038	0.0388	0.0048	3,208
p378k	4,04,367	8,16,534	8,16,534	0	0	1,107	0.022	0.0007	0.0145	44
p388k (*)	5,06,034	8,81,417	8,76,750	4,665	2	11,975	0.029	0.0078	0.0065	830
p469k	49,771	1,42,751	1,40,869	1,762	120	578	0.094	4.4455	1.7238	13,139
p951k (*)	11,47,491	15,57,914	15,42,633	15,281	0	20,899	0.060	0.0011	0.0119	2,668
p1522k (*)	11,93,824	16,97,662	16,81,874	15,788	0	63,549	0.073	0.0099	0.0173	9,324
p2927k	25,39,052	35,27,607	34,12,613	1,14,907	87	39,842	0.156	0.0308	0.0602	33,758

of the faults into classes detected (Det.), provably redundant (Red.) and aborted (Ab.) is shown in columns 1 through 6. Column 7 contains the number of generated patterns. The average time (in seconds) per fault for CNF generation, SAT solving and fault simulation (fault dropping) can be found in columns 8 through 10, the total time (T [s]) in column 11. No thread parallelism of the MiraXT engine was employed.

Circuits marked by asterisk (*) contain tristate elements. TIGUAN replaces bufif1 gates by AND gates and notif1 gates by NAND gates which retains the circuit's functionality. To prevent bus contention, an additional clause which ensures that at most one driver is active at the same time can be generated. We did not generate such a clause in our experiments.

TIGUAN can handle multi-million-gate designs with very few aborts and in limited time. The number of patterns is rather large, however we point out that no compaction techniques such as reverse-order simulation were employed. The option to maximize don't-cares was not used.

Tables 2 and 3 compare the performance of TIGUAN (without thread parallelism) with the best published results by the state-of-the-art SAT-based tool PASSAT available to us [10,16] (only results for circuits quoted in [10,16] are reported in Tables 2

Table 2 Results of single-threaded TIGUAN without fault dropping for ISCAS, ITC and NXP circuits for stuck at faults and comparison with [16]

Circuit	Gates	Faults	TIGUAN				PASSAT	
			Det.	Red.	Ab.	T (s)	Ab.	T (s)
c0432	203	524	520	4	0	0.5	0	2.6
c0499	275	758	750	8	0	1.0	0	21.0
c1355	619	1,574	1,566	8	0	4.5	0	32.5
c1908	938	1,879	1,870	9	0	4.6	0	14.4
c3540	1,741	3,428	3,291	137	0	14.0	0	47.9
c7552	3,827	7,550	7,419	131	0	19.4	0	106.5
s01494	686	1,506	1,494	12	0	0.6	0	2.7
s05378	3,221	4,603	4,563	40	0	4.1	0	14.3
s15850	11,067	11,725	11,336	389	0	47.8	0	121.3
s38417	25,585	31,180	31,015	165	0	89.7	0	191.3
b10	197	486	486	0	0	0.1	0	0.3
b11	579	1,436	1,434	2	0	1.0	0	4.8
b12	1,127	2,827	2,826	1	0	1.5	0	5.6
b14	5,923	16,167	16,137	30	0	122.1	0	1426.8
b15	8,026	21,282	20,545	737	0	378.8	0	2673.6
p81k	96,722	2,04,174	2,02,981	1,193	0	4,429	0	12,116
p89k	92,706	1,50,538	1,48,604	1,934	0	2,544	0	5,755
p100k	1,02,443	1,62,129	1,61,404	725	0	2,102	19	15,397
p141k	1,85,360	2,82,428	2,79,189	3,239	0	29,938	236	95,452
p951k	11,47,491	15,57,914	15,42,633	15,281	0	1,58,875	132	1,66,791

Table 3 Comparison of number of aborts (Ab.) and run time for TIGUAN and PASSAT [10] with fault dropping

ITC 99 circuits					NXP circuits				
Circuit	PASSAT		TIGUAN		Circuit	PASSAT		TIGUAN	
	Ab.	T (s)	Ab.	T (s)		Ab.	T (s)	Ab.	T (s)
b14	0	19.0	0	13.2	p35k	0	1,561.0	0	1,364.0
b15	0	24.0	0	44.0	p81k	0	583.0	0	162.0
b17	0	142.0	0	123.6	p89k	0	573.0	0	154.0
b18	0	1,350.0	0	341.8	p100k	0	410.0	0	91.0
b20	0	56.0	0	29.4	p141k	0	4,740.0	0	1,706.0
b21	0	59.0	0	33.3	p469k	77	6,180.0	120	13,139.0
b22	0	95.0	0	36.0	p951k	1	18,300.0	0	2,668.0

Table 4 Performance of the two-stage approach with TIGUAN as the first-stage tool

Circuit	Two-stage approach								One-stage approach (from Table 1)		No timeout	
	First stage (timeout 1 s)		Second stage (timeout 20 s)									
			2 Threads			4 Threads						
	T (s)	Faults left	Ab.	T (s)	Total time (s)	Ab.	T (s)	Total time (s)	Ab.	Time (s)	Ab.	Time (s)
p77k	4,545	1,322	0	1,354	5,899	0	1,003	5,548	0	**5,454**	0	5,454
p286k	2,115	126	1	1,232	**3,347**	1	1,609	3,724	1	3,456	0	3,497
p295k	1,062	3	1	62	**1,124**	1	66	1,128	1	1,159	0	1,228
p330k	2,376	70	17	616	2,992	16	491	**2,867**	32	3,208	0	23,475
p388k	800	2	2	41	841	2	40	840	2	**830**	0	1,263
p469k	17,929	2,680	28	3,343	21,272	3	2,152	20,081	120	**13,139**	0	30,815
p1522k	9,295	22	0	15	**9,310**	0	19	9,314	0	9,324	0	9,324
p2927k	25,856	666	80	3,298	29,154	73	3,120	**28,976**	87	33,758	0	50,812

and 3).[1] Results in Table 2 have been generated with fault dropping switched off and timeout of 20 s (as in [16]). We quote the best numbers achieved by PASSAT among different learning techniques presented in [16]. Table 3 compares results obtained using fault dropping and timeout of 20 s with columns 4 and 5 in Table VI in [10] (run times were converted into seconds). Although the same industrial circuits were used in [10,16], some of them were named differently: circuits p44k, p49k, p80k, p88k, p99k, p177k and p1330k in [10,16] correspond to circuits p35k, p469k, p81k, p89k, p100k, p141k and p951 in Tables 2 and 3, respectively.

TIGUAN outperforms PASSAT both with respect to aborts and run time. For circuits p89k, p141k and p951k, part of the run time advantage is due to the simplified encoding of tristate elements for the three circuits mentioned above (PASSAT switches to multi-valued logic which includes the high-impedance value if a circuit contains tristate elements). All other circuits are purely Boolean and do not require multi-valued logic.

4.2 Multi-Threaded Performance

We performed two experiments on TIGUAN in two-stage mode described in Sect. 3.3. In the first experiment, single-threaded TIGUAN with a timeout of 1 s per fault was used as the first-stage tool. In the second experiment, a state-of-the-art commercial structural ATPG tool was used as the first-stage tool.

Table 4 summarizes the results of the first experiment for circuits with at least one abort during the first stage. Column 2 gives the run time of the first stage. The number of faults aborted during the first stage and targeted by the second stage can be found

[1] An AMD Athlon with 2.2 GHz and 1 GB RAM was used in [16]. A dual-core Xeon with 3 GHz and 32 GB RAM was used in [10].

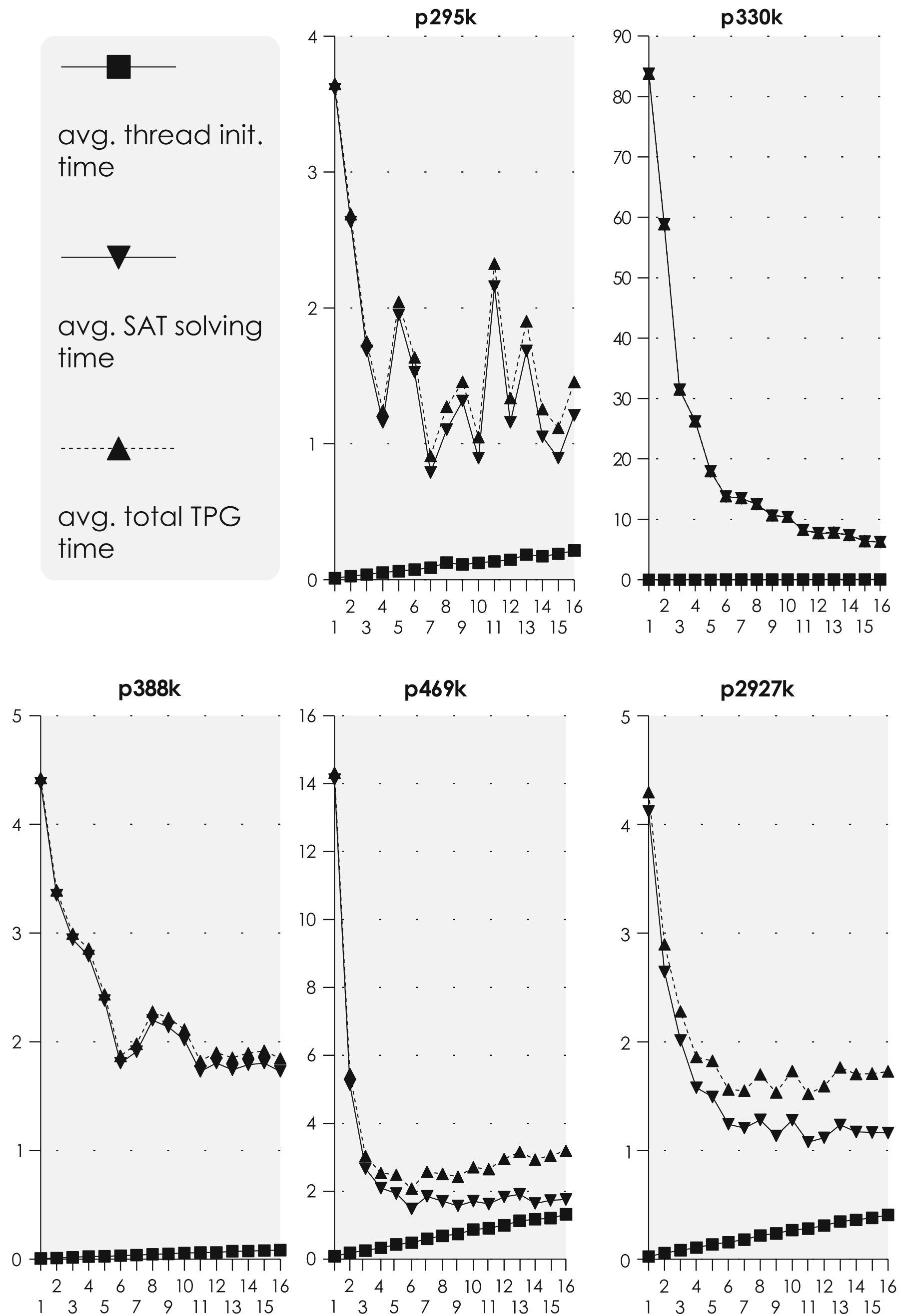


Fig. 4 Average thread initialization times, SAT solving times and total test generation time for stuck-at faults aborted by structural ATPG

Table 5 Results for gate-exhaustive testing with fault dropping, timeout 20 s per fault

Circuit	Gates	Faults	Distribution			Pats.	Run time (s)	
			Det.	Red.	Ab.		Per fault	Total
c5315	2,608	12,084	10,194	1,890	0	1,069	0.0004	4
c6288	2,480	9,664	7,934	1,730	0	439	0.0019	18
c7552	3,827	15,050	12,345	2,705	0	1,227	0.0006	9
cs13207	9,441	26,004	22,950	3,054	0	1,381	0.0006	15
cs15850	11,067	29,922	26,703	3,219	0	1,213	0.0009	26
cs35932	19,876	60,064	46,484	13,580	0	128	0.0004	23
cs38417	25,585	70,236	66,228	4,008	0	2,425	0.0003	20
cs38584	22,447	75,278	64,629	10,649	0	1,549	0.0003	24
b17	25,719	1,38,230	97,826	40,404	0	6,041	0.0040	554
b18	76,513	3,96,886	2,92,165	1,04,721	0	16,084	0.0058	2,313
b20	12,991	66,444	52,049	14,395	0	5,048	0.0028	187
b21	13,168	66,420	52,444	13,976	0	5,597	0.0029	192
b22	18,789	94,022	73,540	20,482	0	5,522	0.0026	244
p330k	3,65,492	11,66,046	10,37,130	1,28,843	73	36,401	0.0102	11,934
p378k	4,04,367	13,70,984	11,91,909	1,79,075	0	1,980	0.0037	5,117
p388k	5,06,034	16,63,442	14,63,686	1,99,754	2	17,317	0.0049	8,220
p469k	49,771	3,12,784	2,41,562	70,844	378	652	0.1618	50,603
p951k	11,47,491	32,50,198	28,84,773	3,65,425	0	28,050	0.0089	28,863
p1522k	11,93,824	37,08,692	33,50,769	3,57,923	0	80,404	0.0140	52,036
p2927k	25,39,052	70,48,378	62,53,392	7,94,723	263	51,340	0.0241	1,69,859

in column 3. The second stage was run for 2 and 4 parallel threads with a timeout of 20 s. For each scenario, the number of aborts during the second stage, its run time and the cumulative run time of the first and the second stage are given in columns 4 through 9.

Columns 10 and 11 give the number of aborts and the run time of the one-stage method from columns 6 and 11 of Table 1, respectively. Note that the timeout for the one-stage method was 20 s. The minimal run time of columns 6, 9 and 11 is marked bold. This indicates the minimal time which is required for the complete ATPG process by either two-stage or one-stage approach. The two-stage method with multi-threading always yields less aborts than the one-stage approach and reduces the ATPG time for more than half of the circuits. 2-thread parallelism often yields lower run times while using 4 threads helps to reduce aborts.

For reference, the final two columns of Table 4 report the number of aborts and the time which TIGUAN consumes when started without a time limit. Clearly, all faults are classified without aborts. Note that all circuits not included in Table 4 have already been classified without aborts using a timeout of 1 s. Hence, TIGUAN completely classifies all faults in the industrial circuits (as it does for ISCAS and ITC circuits not included in Tables 1 and 4).

Table 6 Results for resistive bridging faults with fault dropping, timeout 20 s per fault

Circuit	Faults	Distribution			Patterns	Run time (s)	
		Det.	Red.	Ab.		Per fault	Total
c5315	28,214	19,594	8,620	0	1,661	0.0008	23.50
c6288	33,603	20,086	13,517	0	1,320	0.0037	125.28
c7552	32,028	19,024	13,004	0	1,224	0.0013	41.94
cs13207	20,366	15,107	5,259	0	1,115	0.0007	14.42
cs15850	20,061	14,803	5,258	0	1,090	0.0014	28.18
cs35932	27,160	9,332	17,828	0	133	0.0015	41.97
cs38417	25,976	20,174	5,802	0	1,619	0.0011	27.34
cs38584	26,602	17,207	9,395	0	1,486	0.0012	32.43
b17	41,651	7,966	33,685	0	2,925	0.0142	591.01
b18	42,881	8,753	34,128	0	3,926	0.0250	1,070.18
b20	44,378	8,073	36,305	0	2,285	0.0104	461.74
b21	44,915	8,027	36,888	0	2,293	0.0104	467.61
b22	44,824	8,551	36,273	0	2,170	0.0108	482.63
p330k	23,716	20,991	2,725	0	4,428	0.0216	511.33
p378k	27,898	23,659	4,239	0	529	0.0060	166.41
p388k	24,637	21,495	3,142	0	2,139	0.0112	274.79
p469k	45,528	13,444	31,837	247	774	0.4523	20,594.07
p951k	21,967	20,106	1,861	0	1,958	0.0149	326.58
p1522k	22,731	19,167	3,564	0	5,731	0.0522	1,186.51
p2927k	22,638	19,351	3,286	1	3,761	0.0634	1,434.36

In the second experiment, a structural ATPG was used as the first-stage tool, and multi-threaded TIGUAN with a different number of threads and with no timeout was used as the second-stage tool. The number of aborts by the first-stage tool was 19, 186, 66, 210 and 5,601 for circuits p295k, p330k, p388k, p469k, and p2927k, respectively. TIGUAN was able to classify all of these aborted faults. The average run times per fault for up to 16 threads are shown in Fig. 4 in graph form. The total time per fault consists of the thread initialization time, the SAT solving time and time for auxiliary processes such as CNF generation and fault dropping (their contribution is negligible for hard-to-detect faults considered).

It can be seen that the thread initialization time increases linearly with the number of threads (and thus cores) used. The SAT solving time per fault decreases rapidly for the first 4–6 available cores and then stagnates, fluctuates randomly or even increases. One reason for run time increase could be memory congestion due to constant rate of conflict clause generation by individual threads: currently each thread produces roughly 10,000 conflict clauses per second, so the total conflict clause production increases linearly with the number of threads used. We are investigating an adaptive strategy in this regard.

4.3 Non-Standard Fault Models

Table 5 reports the application of TIGUAN to generate gate-exhaustive test sets for larger ISCAS, ITC and NXP circuits. The number of faults (column 3) significantly exceeds the number of gates (column 2). A significant fraction of the generated faults are redundant (column 5). There are little aborts (column 6). The run times per fault exceed those for stuck-at faults but are generally reasonable (column 8).

To demonstrate test generation for resistive bridging faults, we first generated resistive bridging fault lists by selecting, for each circuit, 10,000 pairs of interconnects randomly. For every pair of interconnects, we calculated the representative resistances using the tool flow from [13] and assuming the same technology parameters as in [13]. For every section, we generated one conditional multiple-stuck-at fault. The aggressor list consisted of the conditions on the inputs of the gates driving the shorted interconnects. The victim list included all inputs of the gates driven by the shorted interconnects on which an erroneous value was interpreted.

Table 6 summarizes the performance of TIGUAN for the resistive bridging fault list generated as explained in Sect. 2.2. The format of the table is similar to Table 5. The number of CMS@ faults equals 10,000 multiplied by the average number m of representative resistances per resistive bridging fault. This number ranges between 14,489 for b13 and 45,528 for p469k. There are again no aborts for almost all circuits while the run times are reasonable. We also applied the two-stage method, observing results similar to the case of stuck-at faults: the number of aborts was reduced, and the run time went down for circuits with the largest SAT solving time. We are not aware of comparable results by PASSAT or any other SAT-based tool. Existing resistive bridging fault ATPGs [14,36] cannot handle multi-million gate designs.

5 Conclusions

TIGUAN currently can completely classify all single-stuck-at faults in both large industrial circuits and structurally complex ISCAS circuits without aborts. It is also an effective and flexible tool to generate tests for non-standard fault models for which no adequate dedicated ATPG tool is available. This is achieved by providing a mapping between the non-standard model and conditional multiple-stuck-at fault model which TIGUAN supports. The two-stage approach allows to identify hard-to-detect faults for which sophisticated optimization strategies of the SAT engine and thread parallelism are effective.

One research direction for the future is the incorporation of state-of-the-art static and dynamic compaction [3,22,25,31,35] and test set relaxation techniques [12,24] to reduce the pattern count. We also plan to extend the CMS@ concept to dynamic fault models such as delay faults [41], and power droop [30]. Moreover, we investigate the theoretical findings on fault vs. search parallelism [18] to better utilize novel multiprocessor and multi-core architectures with ultra-fast interprocessor communication.

Acknowledgments Parts of this work have been supported by the German Research Council under project BE 1176/14-1 and by the Alexander-von-Humboldt Foundation. We are thankful to Juergen Schloeffel

of NXP Hamburg for providing industrial circuits and Tobias Schubert of University of Freiburg for fruitful discussions on SAT solving.

References

1. Abramovici, M., Breuer, M.A., Friedman, A.D.: Digital Systems Testing and Testable Design. Computer Science Press, New York (1990)
2. Aitken, R.C.: New defect behavior at 130 nm and beyond. In: European Test Symposium, pp. 279–284 (2004)
3. Ayari, B., Kaminska, B.: A new dynamic test vector compaction for automatic test pattern generation. IEEE Trans. CAD **13**(3), 353–358 (1994)
4. Brglez, F., Bryan, D., Kozminski, K.: Combinational profiles of sequential benchmark circuits. In: International Symposium on Circuits and Systems, pp. 1929–1934 (1989)
5. Brglez, F., Fujiwara, H.: A neutral netlist of 10 combinational circuits and a target translator in fortran. In: International Symposium on Circuits and Systems, Special Sess. on ATPG and Fault Simulation, pp. 663–698 (1985)
6. Cho, K.Y., Mitra, S., McCluskey, E.J.: Gate exhaustive testing. In: International Test Conference (2005)
7. Corno, F., Sonza Reorda, M., Squillero, G.: RT-level ITC 99 benchmarks and first ATPG results. IEEE Des. Test Comput. **17**(3), 44–53 (2000)
8. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
9. Desineni, R., Dwarkanath, K.N., Blanton, R.D.: Universal test generation using fault tuples. In: International Test Conference, pp. 812–819 (2000)
10. Drechsler, R., Eggersglüß, S., Fey, G., Glowatz, A., Hapke, F., Schlöffel, J., Tille, D.: On acceleration of SAT-based ATPG for industrial designs. IEEE Trans. CAD **27**(7), 1329–1333 (2008)
11. Eggersglüß, S., Drechsler, R.: Improving test pattern compactness in SAT-based ATPG. In: Asian Test Symposium, pp. 445–452 (2007)
12. El-Maleh, A.H., Al-Utaibi, K.: An efficient test relaxation technique for synchronous sequential circuits. IEEE Trans. CAD **23**(6), 933–940 (2004)
13. Engelke, P., Braitling, B., Polian, I., Renovell, M., Becker, B.: SUPERB: simulator utilizing parallel evaluation of resistive bridges. In: Asian Test Symposium, pp. 433–438 (2007)
14. Engelke, P., Polian, I., Renovell, M., Becker, B.L: Automatic test pattern generation for resistive bridging faults. J. Electron. Test. Theory Appl. **22**(1), 61–69 (2006)
15. Engelke, P., Polian, I., Renovell, M., Becker, B.: Simulating resistive bridging and stuck-at faults. IEEE Trans. CAD **25**(10), 2181–2192 (2006)
16. Fey, G., Warode, T., Drechsler, R.: Reusing learned information in SAT-based ATPG. In: VLSI Design, IEEE Computer Society, pp. 69–76 (2007)
17. Fujiwara, H.: FAN: A fanout-oriented test pattern generation algorithm. In: IEEE International Symposium on Circuits and Systems, pp. 671–674 (1985)
18. Fujiwara, H., Inoue, T.: Optimal granularity of test generation in a distributed system. IEEE Trans. CAD **9**(8), 885–892 (1990)
19. Gizdarski, E., Fujiwara, H.: SPIRIT: a highly robust combinational test generation algorithm. IEEE Trans. CAD **21**(12), 1446–1458 (2002)
20. Goel, P.: An implicit enumeration algorithm to generate tests for combinational logic circuits. IEEE Trans. CAD **30**, 215–222 (1981)
21. Hamzaoglu, I., Patel, J.H.: New techniques for deterministic test pattern generation. J. Electron. Test. Theory Appl. **15**, 63–73 (1999)
22. Hamzaoglu, I., Patel, J.H.: Test set compaction algorithms for combinational circuits. IEEE Trans. CAD **19**(8), 957–963 (2000)
23. Hillebrecht, S., Polian, I., Engelke, P., Becker, B., Keim, M., Cheng, W.-T.: Extraction, simulation and test generation for interconnect open defects based on enhanced aggressor-victim model. In: Internatinal Test Conference, pp 1–10 (2008)
24. Kajihara, S., Miyase, K.: On identifying don't care inputs of test patterns for combinational circuits. In: Internatinal Conference on CAD, pp. 364–369 (2001)
25. Kajihara, S., Pomeranz, I., Kinoshita, K., Reddy, S.M.: Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. IEEE Trans. CAD **14**(12), 1496–1504 (1995)
26. Kropf, T.: Introduction to Formal Hardware Verification. Springer, Berlin (2000)

27. Kundu, S., Zachariah, S.T., Chang, Y.-S., Tirumurti, C.: On modeling crosstalk faults. IEEE Trans. CAD **24**(12), 1909–1915 (2005)
28. Larrabee, T.:. Efficient generation of test patterns using Boolean difference. In: Internatinal Test Conference, pp. 795–801 (1989)
29. Lewis, M., Schubert, T., Becker, B.: Multithreaded SAT solving. In: ASPDAC 2007, Yokohama, Japan, January 2007. 12th Asia and South Pacific Design Automation Conference (2007)
30. Polian, I., Czutro, A., Kundu, S., Becker, B.: Power droop testing. IEEE Des. Test Comput. **24**(3), 276–284 (2007)
31. Pomeranz, I., Reddy, L.N., Reddy, S.M.: COMPACTEST: a method to generate compact test sets for combinational circuits. In: Internatinal Test Conference, pp. 194–203 (1991)
32. Rajski, J., Tyszer, J., Kassab, M., Mukherjee, N.: Embedded deterministic test. IEEE Trans. CAD **23**(5), 776–792 (2004)
33. Renovell, M., Azaïs, F., Bertrand, Y.: Detection of defects using fault model oriented test sequences. J. Electron. Test. Theory Appl. **14**, 13–22 (1999)
34. Roth, J.P.: Diagnosis of automata failures: a calculus and a method. IBM J. Res. Dev. **10**, 278–281 (1966)
35. Rudnick, E.M., Patel, J.H.: Efficient techniques for dynamic test sequence compaction. IEEE Trans. Comput. **48**(3), 323–330 (1999)
36. Sar-Dessai, V., Walker, D.M.H.: Resistive bridge fault modeling, simulation and test generation. In: Internatinal Test Conference, pp. 596–605 (1999)
37. Sato, Y., Yamazaki, I., Yamanaka, H., Ikeda, T., Takakura, M.: A persistent diagnostic technique for unstable defects. In: Internatinal Test Conference, pp. 242–249 (2002)
38. Schubert, T., Lewis, M., Becker, B.: PaMiraXT: Parallel SAT solving with threads and message passing. J. Satisfiability, Boolean Model. Comput. **6**, 203–222 (2009)
39. Shinogi, T., Kanbayashi, T., Yoshikawa, T., Tsuruoka, S., Hayashi, T.: Faulty resistance sectioning technique for resistive bridging fault ATPG systems. In: Asian Test Symposium, pp. 76–81 (2001)
40. Siewiorek, D.P., Swarz, R.S.: Reliable Computer Systems—Design and Evaluation. Digital Press, Belford (1992)
41. Smith, G.L.: Model for delay faults based upon paths. In: International Test Conference, pp. 342–349 (1985)
42. Snir, M., Otto, S.W., Walker, D.W., Dongarra, J., Huss-Lederman, S.: MPI: The Complete Reference. MIT Press, Cambridge (1996)
43. Stephan, P., Brayton, R., Sangiovanni-Vincentelli, A.: Combinational test generation using satisfiability. IEEE Trans. CAD **15**(9), 1167–1176 (1996)
44. Tafertshofer, P., Ganz, A.: SAT based ATPG using fast justification and propagation in the implication graph. In: Internatinal Conference on CAD, pp.139–146 (1999)
45. Wang, C., Reddy, S.M., Pomeranz, I., Lin, X., Rajski, J.: Conflict driven techniques for improving deterministic test pattern generation. In: Internatinal Conference on CAD (2002)
46. Zhang, M., Mitra, S., Mak, T.M., Seifert, N., Wang, N.J., Shi, Q., Kim, K.S., Shanbhag, N.R., Patel, S.J.: Sequential element design with built-in soft error resilience. IEEE Trans. VLSI Syst. **14**(12), 1368–1378 (2006)