TIGUAN: Thread-parallel Integrated test pattern Generator Utilizing satisfiability ANalysis*

Alejandro Czutro*, Ilia Polian*, Matthew Lewis*, Piet Engelke*, Sudhakar M. Reddy**, Bernd Becker*

*Institute for Computer Science
Albert-Ludwigs-University
D-79110 Freiburg i. Br., Germany

**ECE Department
University of Iowa
Iowa City, IA 52242, USA

Abstract

We present the automatic test pattern generator 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.

1 Introduction

Traditional deterministic automatic test pattern generation (ATPG) algorithms work directly on the circuit structure [1–4], possibly in conjunction with additional data structures such as implication graphs [5] or advanced techniques to prune the solution space [6, 7]. It has long been known that an ATPG problem can be reduced to a Boolean satisfiability (SAT) instance and solved using a SAT solver [8, 9]. 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, 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 [11]. Non-standard fault models such as resistive bridging faults [12, 13] or interconnect opens [14, 15] may impose very specific conditions on the lines in the circuit, which are, in many cases, impossible to satisfy, so the fault is undetectable.

*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.

Second, redundant structures are increasingly used to enhance circuit reliability and yield [16, 17]. 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.

State of the art in SAT-based ATPG is currently given by the tool PASSAT [10] which is integrated into NXP's structural ATPG framework AMSAL. Performance enhancement of SAT-based ATPG by utilizing learning techiques has been discussed in [18].

In this paper 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 [19]. 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 PASSAT, TIGUAN is tightly coupled with the SAT engine and can dynamically control its internal parameters such as preprocessing steps to be performed or number of threads to be used. Moreover, we present a two-phase method which allows to utilize MiraXT's inherent parallelism in a meaningful way.

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 comparable SAT-based ATPG tools.

The remainder of the paper 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 gives the overall flow of TIGUAN. Experimental results for stuck-at faults as well as more complex faults mapped to CMS@ faults are reported in Section 4. Section 5 concludes the paper.

2 CMS@ Fault Model

TIGUAN considers 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 [20] or the Generalized Fault Model [21]. 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.

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 [22]. 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 [22].

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 the example of the stuck-at-1 fault at the output c of an AND2 gate with inputs a and b, three CMS@ faults are injected. All three faults have the same victim list $\{c/1\}$; the aggressor lists of the three faults are $\{a/0, b/0\}$, $\{a/0, b/1\}$ and $\{a/1, b/0\}$, respectively. 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 [12, 13]. In general, a short defect with resistance R_{sh} between interconnect a driven by gate A and interconnect b driven by gate 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. The voltages V_a and V_b are determined by the electrical parameters of transistors within gates A and B and the number of transistors which are activated. The latter parameter depends on the logic values on the inputs of gates A and B. Hence, to detect a resistive short defect with a given resistance, specific values on the gates driving the shorted interconnects may be required, and the fault effect may be visible on one or more gates driven by the shorted interconnects.

The detection conditions may differ for short defects which involve the same pair of interconnects but have different resistances R_{sh}. It has been shown in [23, 24] that there exists a partition of the continuous space of R_{sh} values into $m + 1$ *sections* $[R_0, R_1]$, $[R_1, R_2]$, ..., $[R_{m-1}, R_m]$, $[R_m, \infty]$, where $R_0 := 0 < R_1 < \cdots < R_m < \infty$, such that the logical behavior of the circuit is identical for R_{sh} values within one section. This means that a test pattern generated for a short defect with a fixed resistance detects all defects between the same interconnects with a resistance from the same section. In other words, to fully cover all possible resistive short defects between interconnects a and b, it is sufficient to generate a test set which detects $m + 1$ representative defects (one from each section).

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 section information using the tool flow from [24] and assuming the same technology parameters as in [24]. 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.

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. The generation of the CNF is described in detail in [8, 9]. We apply the usual speed-up techniques such as D-chains [9]. The MiraXT engine incorporates several methods to accelerate SAT solving, which are described in [19] along with details on the multi-threading solving mechanism. We tuned the performance of MiraXT by adjusting several solver-internal control variables to values appropriate for ATPG instances. Based on extensive empirical data, we decided not to reuse parts of a CNF generated for one fault when considering other faults.

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 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.

Table 1: ATPG for stuck-at faults with fault dropping for NXP circuits, timeout 20 seconds per fault

Circuit	# gates	# faults	# detected	# redundant	# aborts	# patterns	Time per fault [s]			Total time
							CNF gen.	SAT	FSIM	T [s]
p35k	48927	67733	66721	1012	0	11536	0.033	0.0278	0.0007	1364
p45k	46075	68760	68564	196	0	3604	0.005	0.0017	0.0008	47
p77k	75033	120348	113049	7299	0	5318	0.029	0.3455	0.0510	5454
p78k	80875	163310	163310	0	0	468	0.005	0.0006	0.0061	7
p81k	96722	204174	202981	1193	0	7529	0.010	0.0017	0.0015	162
p89k (*)	92706	150538	148604	1934	0	9868	0.007	0.0015	0.0018	154
p100k	102443	162129	161404	725	0	5142	0.006	0.0032	0.0028	91
p141k (*)	185360	282428	279189	3239	0	8893	0.050	0.0337	0.0024	1706
p267k	296404	366871	365423	1448	0	11579	0.020	0.0031	0.0037	447
p269k (*)	297497	369055	367607	1448	0	11633	0.018	0.0031	0.0046	436
p286k (*)	373221	650368	640103	10264	1	20243	0.041	0.0490	0.0062	3456
p295k (*)	311901	472022	468174	3847	1	22786	0.024	0.0053	0.0042	1159
p330k	365492	540758	535070	5656	32	23392	0.038	0.0388	0.0048	3208
p378k	404367	816534	816534	0	0	1107	0.022	0.0007	0.0145	44
p388k (*)	506034	881417	876750	4665	2	11975	0.029	0.0078	0.0065	830
p469k	49771	142751	140869	1762	120	578	0.094	4.4455	1.7238	13139
p951k (*)	1147491	1557914	1542633	15281	0	20899	0.060	0.0011	0.0119	2668
p1522k (*)	1193824	1697662	1681874	15788	0	63549	0.073	0.0099	0.0173	9324
p2927k	2539052	3527607	3412613	114907	87	39842	0.156	0.0308	0.0602	33758

TIGUAN also provides a mode in which the percentage of don't cares (Xes) in the generated patterns is maximized. This property is essential for static as well as dynamic compaction [25] and test compression [26]. The injection of Xes is performed by the SAT engine; on top of that, an input-output-cone analysis similar to [27] is performed to identify further Xes. We are currently integrating more elaborate methods of test set relaxation [28, 29] 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. This must be complemented by an appropriate representation of the data shared among the threads and an efficient mechanism to access this data. 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

MiraXT and thus TIGUAN implement parallelism based on the shared memory paradigm (rather than message passing). A common clause data base contains pointers to clauses of the original SAT instance as well as conflict clauses produced during solving. Note that clauses representing the fault-free circuit, clauses representing the circuit with the fault injected and auxilliary clauses from D-chain are treated equally. Every thread keeps a local list which contains two selected literals of each clause, called watch literals. If a thread requires the complete clause information, it must access the global data base, which may require inter-processor communication. State-of-the-art multi-processor and multi-core systems include mechanisms such as AMD's Hyper-Transport Bus which accelerate this kind of communication.

Utilizing a common clause database allows threads to share clauses. This concept is called knowledge sharing and basically allows the solver threads to learn from each others' mistakes (conflicts). Before inserting a clause into the common database, the thread analyzes whether clauses recently inserted by other threads are more effective, i.e., prune larger parts of the search space. An optimized fine-grained lock management is implemented. It has been shown to reduce the performance overhead due to lock conflicts to fractions of a per cent. Clause deletion is implemented by a two-stage garbage collection strategy which almost eliminates the need for locks.

3.2.2 Problem partitioning and solving

The solving and the partitioning of the instance is managed by the *master control object* (MCO) of very limited complexity. MCO essentially forwards messages between threads and does not intervene with a thread's computation process. MCO also manages running and idle threads which are waiting for new sub-problems.

After CNF generation, several preprocessing steps are performed to simplify the instance. The multithreaded solver starts by giving the complete decision tree 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 one of the sleeping threads and transfers control of one part to this thread. If a thread's sub-problem is unsatisfiable, it inserts the required conflict clauses into the data base and enters the idle state. The problem is unsatisfiable if all threads become idle.

Table 2: ATPG without fault dropping for ISCAS, ITC and NXP circuits for stuck at faults and comparison with [18]

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	1574	1566	8	0	4.5	0	32.5
c1908	938	1879	1870	9	0	4.6	0	14.4
c3540	1741	3428	3291	137	0	14.0	0	47.9
c7552	3827	7550	7419	131	0	19.4	0	106.5
s01494	686	1506	1494	12	0	0.6	0	2.7
s05378	3221	4603	4563	40	0	4.1	0	14.3
s15850	11067	11725	11336	389	0	47.8	0	121.3
s38417	25585	31180	31015	165	0	89.7	0	191.3
b10	197	486	486	0	0	0.1	0	0.3
b11	579	1436	1434	2	0	1.0	0	4.8
b12	1127	2827	2826	1	0	1.5	0	5.6
b14	5923	16167	16137	30	0	122.1	0	1426.8
b15	8026	21282	20545	737	0	378.8	0	2673.6
p81k	96722	204174	202981	1193	0	4429	0	12116
p89k	92706	150538	148604	1934	0	2544	0	5755
p100k	102443	162129	161404	725	0	2102	19	15397
p141k	185360	282428	279189	3239	0	29938	236	95452
p951k	1147491	1557914	1542633	15281	0	158875	132	166791

3.3 Two-stage method

It has been noted, e.g. in [4], 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 second 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 a two-stage ATPG strategy. In the first stage, TIGUAN is run in the single-thread mode with an aggressive time limit. In the second stage, TIGUAN is applied to the remaining hard-to-detect faults employing thread parallelism.

4 Experimental Results

TIGUAN was applied to ISCAS 85 circuits and combinational cores of ISCAS 89 circuits, ITC 99 circuits and industrial circuits provided by NXP. The measurements for stuck-at faults (Tables 1 – 4) were performed on a 2.8 GHz AMD Opteron computer with 16 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.

4.1 Single-threaded single-stuck-at ATPG

Table 1 reports ATPG results for industrial circuits using fault dropping and 20 seconds timeout per fault (a fault was classified as aborted if no pattern was found within 20 seconds). The name of the circuit, the number of gates and collapsed faults and the distribution of the faults into classes detected, provably redundant and aborted is shown in columns 1 through 6. Column 7 contains the number of generated patterns.

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

ITC-99 circuits					NXP circuits				
Circ.	PASSAT		TIGUAN		Circ.	PASSAT		TIGUAN	
	Ab.	T [s]	Ab.	T [s]		Ab.	T [s]	Ab.	T [s]
b14	0	19.0	0	13.2	p35k	0	1561.0	0	1364.0
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	1350.0	0	341.8	p100k	0	410.0	0	91.0
b20	0	56.0	0	29.4	p141k	0	4740.0	0	1706.0
b21	0	59.0	0	33.3	p469k	77	6180.0	120	13139.0
b22	0	95.0	0	36.0	p951k	1	18300.0	0	2668.0

The 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 PASSAT available to us [10, 18] (only results for circuits quoted in [10, 18] are reported in Tables 2 and 3).[1] Results in Table 2 have been generated with fault dropping switched off and timeout of 20 seconds (as in [18]). We quote the best numbers achieved by PASSAT among different learning techniques presented in [18]. Table 3 compares results obtained using fault dropping and timeout of 20 seconds 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, 18], some of them were named differently: circuits p44k, p49k, p80k, p88k, p99k, p177k and p1330k in [10, 18] 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 Multithreaded performance

We ran TIGUAN in the two-stage mode described in Section 3.3. The limits for the first and the second stage were 1 and 20 seconds, respectively. Table 4 summarizes the results for circuits with at least one abort during the first stage. Column

[1] An AMD Athlon with 2.2 GHz and 1 GB RAM was used in [18]. A dual-dual-core Xeon with 3 GHz and 32 GB RAM was used in [10].

Table 4: Performance of thread-parallel two-stage approach for single-stuck-at faults

Circuit	Two-stage approach											One-stage approach (Timeout 20 s) From table 1		No timeout (no aborts) T [s]
	First stage (Timeout 1 s)		Second stage (Timeout 20 s)											
	T [s]	Faults left	1 thread			2 threads			4 threads			Aborts	T [s]	
			aborts	T [s]	tot.time	aborts	T [s]	tot.time	aborts	T [s]	tot.time			
p77k	4545	1322	0	2940	7485	0	1354	5899	0	1003	5548	0	**5454**	5454
p286k	2115	126	1	1459	3574	1	1232	**3347**	1	1609	3724	1	3456	3497
p295k	1062	3	1	45	**1107**	1	62	1124	1	66	1128	1	1159	1228
p330k	2376	70	31	806	3182	17	616	2992	16	491	**2867**	32	3208	23475
p388k	800	2	2	40	840	2	41	841	2	40	840	2	**830**	1263
p469k	17929	2680	141	10434	28363	28	3343	21272	3	2152	20081	120	**13139**	30815
p1522k	9295	22	0	63	9358	0	15	**9310**	0	19	9314	0	9324	9324
p2927k	25856	666	92	3929	29785	80	3298	29154	73	3120	**28976**	87	33758	50812

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 in column 3. The second stage was run for 1, 2 and 4 parallel threads with a timeout of 20 seconds. 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 12.

Columns 13 and 14 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 seconds. The minimal run time of columns 6, 9, 12 and 13 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 multithreading 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 column of Table 4 reports the time which TIGUAN consumes when started without a time limit (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 one second. Hence, TIGUAN completely classified all faults in the industrial circuits (as it did for ISCAS and ITC circuits not included in Tables 1 and 4).

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 exceed those for stuck-at faults but are generally reasonable (column 8).

Table 6 summarizes the performance of TIGUAN for the resistive bridging fault list generated as explained in Section 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 sections per resistive bridging fault. This num-

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

Circuit	Gates	Faults	Distribution			Pats.	Run time [s]	
			Det.	Red.	Ab.		per flt.	total
c5315	2608	12084	10194	1890	0	1069	0.0004	4
c6288	2480	9664	7934	1730	0	439	0.0019	18
c7552	3827	15050	12345	2705	0	1227	0.0006	9
cs13207	9441	26004	22950	3054	0	1381	0.0006	15
cs15850	11067	29922	26703	3219	0	1213	0.0009	26
cs35932	19876	60064	46484	13580	0	128	0.0004	23
cs38417	25585	70236	66228	4008	0	2425	0.0003	20
cs38584	22447	75278	64629	10649	0	1549	0.0003	24
b17	25719	138230	97826	40404	0	6041	0.0040	554
b18	76513	396886	292165	104721	0	16084	0.0058	2313
b20	12991	66444	52049	14395	0	5048	0.0028	187
b21	13168	66420	52444	13976	0	5597	0.0029	192
b22	18789	94022	73540	20482	0	5522	0.0026	244
p330k	365492	1166046	1037130	128843	73	36401	0.0102	11934
p378k	404367	1370984	1191909	179075	0	1980	0.0037	5117
p388k	506034	1663442	1463686	199754	2	17317	0.0049	8220
p469k	49771	312784	241562	70844	378	652	0.1618	50603
p951k	1147491	3250198	2884773	365425	0	28050	0.0089	28863
p1522k	1193824	3708692	3350769	357923	0	80404	0.0140	52036
p2927k	2539052	7048378	6253392	794723	263	51340	0.0241	169859

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

Circuit	Faults	Distribution			Patterns	Run time [s]	
		Det.	Red.	Aborts		per fault	total
c5315	28214	19594	8620	0	1661	0.0008	23.50
c6288	33603	20086	13517	0	1320	0.0037	125.28
c7552	32028	19024	13004	0	1224	0.0013	41.94
cs13207	20366	15107	5259	0	1115	0.0007	14.42
cs15850	20061	14803	5258	0	1090	0.0014	28.18
cs35932	27160	9332	17828	0	133	0.0015	41.97
cs38417	25976	20174	5802	0	1619	0.0011	27.34
cs38584	26602	17207	9395	0	1486	0.0012	32.43
b17	41651	7966	33685	0	2925	0.0142	591.01
b18	42881	8753	34128	0	3926	0.0250	1070.18
b20	44378	8073	36305	0	2285	0.0104	461.74
b21	44915	8027	36888	0	2293	0.0104	467.61
b22	44824	8551	36273	0	2170	0.0108	482.63
p330k	23716	20991	2725	0	4428	0.0216	511.33
p378k	27898	23659	4239	0	529	0.0060	166.41
p388k	24637	21495	3142	0	2139	0.0112	274.79
p469k	45528	13444	31837	247	774	0.4523	20594.07
p951k	21967	20106	1861	0	1958	0.0149	326.58
p1522k	22731	19167	3564	0	5731	0.0522	1186.51
p2927k	22638	19351	3286	1	3761	0.0634	1434.36

ber 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.

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 [25,30–33] and test set relaxation techniques [28, 29] to reduce the pattern count. We also plan to extend the CMS@ concept to dynamic fault models such as delay faults [34], and power droop [35]. Moreover, we investigate the theoretical findings on fault vs. search parallelism [36] to better utilize novel multi-processor and multi-core architectures with ultra-fast interprocessor communication.

6 References

[1] J.P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev.*, 10:278–281, 1966.

[2] P. Goel. An implicit enumeration algorithm to generate test for combinational logic. *IEEE Trans. on Comp.*, 30:215–222, 1981.

[3] H. Fujiwara. FAN: A Fanout-Oriented Test Pattern Generation Algorithm. In *IEEE International Symposium on Circuits and Systems*, pages 671–674, 1985.

[4] I. Hamzaoglu and J.H. Patel. New techniques for deterministic test pattern generation. *Jour. of Electronic Testing: Theory and Applications*, 15:63–73, 1999.

[5] P. Tafertshofer and A. Ganz. SAT based ATPG using fast justification and propagation in the implication graph. In *Int'l Conf. on CAD*, pages 139–146, 1999.

[6] E. Gizdarski and H. Fujiwara. SPIRIT: A highly robust combinational test generation algorithm. *IEEE Trans. on CAD*, 21(12):1446–1458, 12 2002.

[7] C. Wang, S.M. Reddy, I. Pomeranz, X. Lin, and J. Rajski. Conflict driven techniques for improving deterministic test pattern generation. In *Int'l Conf. on CAD*, 2002.

[8] T. Larrabee. Efficient Generation of Test Patterns Using Boolean Difference. In *Int'l. Test Conference*, pages 795–801, 1989.

[9] P. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on CAD*, 15(9):1167–1176, September 1996.

[10] 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 Trans. on CAD*, 27(7):1329–1333, 2008.

[11] R. Aitken. New defect behavior at 130nm and beyond. In *European Test Symposium (Emerging Ideas Contribution)*, pages 279–284, 2004.

[12] M. Renovell, F. Azaïs, and Y. Bertrand. Detection of defects using fault model oriented test sequences. *Jour. of Electronic Testing: Theory and Applications*, 14:13–22, 1999.

[13] P. Engelke, I. Polian, M. Renovell, and B. Becker. Simulating resistive bridging and stuck-at faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2181–2192, Oct. 2006.

[14] Y. Sato, I. Yamazaki, H. Yamanaka, T. Ikeda, and M. Takakura. A persistent diagnostic technique for unstable defects. In *Int'l Test Conf.*, pages 242–249, 2002.

[15] S. Hillebrecht, I. Polian, P. Engelke, B. Becker, M. Keim, and W.-T. Cheng. Extraction, simulation and test generation for interconnect open defects based on enhanced aggressor-victim model. In *Int'l Test Conf.*, 2008. In press.

[16] D.P. Siewiorek and R.S. Swarz. *Reliable Computer Systems – Design and Evaluation*. Digital Press, 1992.

[17] M. Zhang, S. Mitra, T.M. Mak, N. Seifert, N.J. Wang, Q. Shi, K.S. Kim, N.R. Shanbhag, and S.J. Patel. Sequential element design with built-in soft error resilience. *IEEE Trans. on VLSI*, 14(12):1368–1378, 2006.

[18] G. Fey, T. Warode, and R. Drechsler. Reusing learned information in SAT-based ATPG. In *VLSI Design*, pages 69–76, 2007.

[19] M. Lewis, T. Schubert, and B. Becker. Multithreaded SAT solving. In *ASPDAC 2007*, Yokohama, Japan, January 2007. 12th Asia and South Pacific Design Automation Conference.

[20] R. Desineni, K.N. Dwarkanath, and R.D. Blanton. Universal test generation using fault tuples. In *Int'l Test Conf.*, pages 812–819, 2000.

[21] S. Kundu, S.T. Zachariah, S.-Y. Chang, and C. Tirumurti. On modeling crosstalk faults. *IEEE Trans. on CAD*, 24(12):1909–1915, 2005.

[22] K.Y. Cho, S. Mitra, and E.J. McCluskey. Gate exhaustive testing. In *Int'l Test Conf.*, 2005.

[23] T. Shinogi, T. Kanbayashi, T. Yoshikawa, S. Tsuruoka, and T. Hayashi. Faulty resistance sectioning technique for resistive bridging fault ATPG systems. In *Asian Test Symp.*, pages 76–81, 2001.

[24] P. Engelke, B. Braitling, I. Polian, M. Renovell, and B. Becker. SUPERB: Simulator utilizing parallel evaluation of resistive bridges. In *Asian Test Symp.*, pages 433–438, 2007.

[25] I. Pomeranz, L.N. Reddy, and S.M. Reddy. COMPACTEST: A method to generate compact test sets for combinational circuits. In *Int'l Test Conf.*, pages 194–203, 1991.

[26] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee. Embedded deterministic test. *IEEE Trans. on CAD*, 23(5):776–792, 5 2004.

[27] S. Eggersglüß and R. Drechsler. Improving test pattern compactness in SAT-based ATPG. In *Asian Test Symp.*, pages 445–452, 2007.

[28] S. Kajihara and K. Miyase. On identifying don't care inputs of test patterns for combinational circuits. In *Int'l Conf. on CAD*, pages 364–369, 2001.

[29] A.H. El-Maleh and K. Al-Utaibi. An efficient test relaxation technique for synchronous sequential circuits. *IEEE Trans. on CAD*, 23(6):933–940, 2004.

[30] 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 Trans. on CAD*, 14(12):1496–1504, 1995.

[31] B. Ayari and B. Kaminska. A new dynamic test vector compaction for automatic test pattern generation. *IEEE Trans. on CAD*, 13(3):353–358, 1994.

[32] I. Hamzaoglu and J. Patel. Test set compaction algorithms for combinational circuits. *IEEE Trans. on CAD*, 19(8):957–963, 2000.

[33] E.M. Rudnick and J. Patel. Efficient techniques for dynamic test sequence compaction. *IEEE Trans. on Computers*, 48(3):323–330, 1999.

[34] G.L. Smith. Model for Delay Faults Based upon Paths. In *Int'l Test Conf.*, pages 342–349, 1985.

[35] I. Polian, A. Czutro, S. Kundu, and B. Becker. Power droop testing. *IEEE Design & Test Magazine*, 2007.

[36] T. Fujiwara, H.; Inoue. Optimal granularity of test generation in a distributed system. *IEEE Trans. on CAD*, 9(8):885–892, 1990.