Speeding up SAT-based ATPG using Dynamic Clause Activation

Stephan Eggersglüß Daniel Tille Rolf Drechsler
Institute of Computer Science, University of Bremen
Bibliothekstr. 1, 28359 Bremen, Germany
{segg, tille, drechsle}@informatik.uni-bremen.de

***Abstract*—SAT-based ATPG turned out to be a robust alternative to classical structural ATPG algorithms such as FAN. The number of unclassified faults can be significantly reduced using a SAT-based ATPG approach. In contrast to structural ATPG, SAT solvers work on a Boolean formula in *Conjunctive Normal Form* (CNF). This results in some disadvantages for SAT solvers when applied to ATPG, e.g. CNF transformation time and loss of structural knowledge. As a result, SAT-based ATPG algorithms are very robust for hard-to-test faults, but suffer from the overhead for easy-to-test faults.**

We propose the SAT technique *Dynamic Clause Activation* (DCA) in order to reduce the run time gap between structural and SAT-based ATPG algorithms and, at the same time, retain the high level of robustness. Using DCA, the SAT solver works on a partial formula of a logic circuit which is dynamically extended during the search process using structural knowledge. Furthermore, efficient dynamic learning techniques can be easily integrated within the proposed technique. The approach is evaluated on large industrial circuits.

***Keywords*-SAT; ATPG; Formal methods; CNF;**

I. Introduction

The post-production test is an important task in the design flow and ensures that no erroneous chip is delivered to customers. The test set is generated by dedicated *Automatic Test Pattern Generation* (ATPG) algorithms such as FAN [1]. Classical ATPG algorithms are very fast and can classify many faults in only short time. However, due to the grown complexity of today's circuits, the number of faults which cannot be classified increases and the high fault coverage demands of the industry is compromised.

In contrast to classical structural ATPG algorithms, ATPG algorithms based on *Boolean Satisfiability* (SAT) do not work on a circuit structure, but on a Boolean formula in *Conjunctive Normal Form* (CNF). Therefore, the ATPG problem has to be transformed into CNF and a SAT solver, e.g. [2]–[4], is applied to solve the formula. SAT-based ATPG [5]–[8] turned out to be an efficient complement to existing classical ATPG algorithms classifying many faults for which structural algorithms find no solution in reasonable time. Recently, it was shown in [9] that the reuse of learned information can further reduce the number of unclassified delay faults significantly. Nonetheless, although being very robust in classifying many hard-to-test faults, SAT-based ATPG algorithms suffer from the overhead for solving easy-to-test faults which represent the majority of all faults. In particular, this is caused by the following drawbacks of CNF-based SAT solvers:

- Loss of structural knowledge – Classical ATPG algorithms benefit strongly from the structural knowledge about the ATPG problem. Due to the transformation into CNF, this knowledge is typically lost.
- Transformation into CNF – Although the complexity of the CNF transformation is linear in the number of gates, the time is not negligible. Especially in the ATPG domain where many instances based on the same circuit have to be solved, the transformation time is a significant overhead as reported in [6], [8].
- Completely specified solution – Due to reasons of efficiency, CNF-based SAT solvers compute a solution where all Boolean variables are specified, although many variables could be assigned with don't cares. This is disadvantageous in particular for test compaction techniques.

A. Related Work

Circuit-based SAT solvers [10], [11] have been developed orthogonally to CNF-based SAT solver. These solvers do not work on the CNF representation anymore but directly on the circuit structure. By this, no transformation is needed and structural knowledge about the problem such as signal correlation is retained and can be exploited. However, due to the loss of the homogeneous structure of the CNF, core techniques of CNF-based SAT solvers, e.g. conflict analysis [2], cannot be applied in such an efficient way

Hybrid SAT solvers [12]–[14] work – at least partly – on the problem in CNF representation but exploit structural knowledge to speed up the search. Thereby, most of the approaches benefit primarily from the explicit modeling of *Observability Don't Cares* (ODC). By this, the search space can be pruned, but the handling of ODCs in the SAT techniques such as conflict analysis causes additional computational overhead.

The only circuit-oriented SAT approach that was applied to ATPG problems was KF-ATPG [15], which is based on CSAT [10]. KF-ATPG is applied to path sensitization problems, where only justification of values is needed. A comparison with KF-ATPG is given in this paper.

B. Contributions

In this paper, we propose the new SAT technique *Dynamic Clause Activation* (DCA). Using DCA, the SAT solver works on a subset of the original problem instance which is extended dynamically using structural information of the connectivity of the netlist. This procedure has the following advantages:

- Exploitation of structural knowledge – Due to retaining the gate connectivity information, structural knowledge can be used during the search process.
- Transformation into CNF – The CNF for the circuit is created only once. The required clauses are dynamically activated and by this the overhead of creating a complete SAT instance for each fault is significantly reduced.
- Implicit modeling of ODCs – Due to the DCA technique, ODCs are modeled implicitly. By this, SAT core techniques, e.g. *Boolean Constraint Propagation* (BCP)

```
1  begin dll()
2      while(TRUE)
3          if (!decide()) then
4              return SATISFIABLE;
5          while(deduce() == CONFLICT)
6              if (!resolveConflict()) then
7                  return UNSATISFIABLE;
8  end
```

Figure 1. Pseudocode of the DLL Algorithm

and conflict analysis, do not have to be modified and retain their efficiency. Additionally, the generated tests contain an increased number of unspecified bits.

The modeling of ODCs or structural information such as the j-frontier [6] in SAT-based algorithms is not new. However, DCA is the first technique that permits the efficient combination of these techniques. We further show that efficient dynamic learning techniques [9], i.e. the reuse of learned information, can easily be integrated into the new technique. The application of DCA in SAT-based ATPG results in a significant speed-up and the run time gap between structural and SAT-based ATPG approaches is minimized or even closed. At the same time, the high level of robustness of SAT-based ATPG can be retained.

The paper is organized as follows: in Section II, the classical DLL algorithm is explained as well as the basic concepts of modern SAT solvers. Furthermore, this section briefly reviews the general circuit-to-CNF conversion. Section III introduces the new DCA technique. In Section IV, the integration of dynamic learning is explained. Experimental results are provided in Section V and conclusions are drawn in Section VI.

II. Preliminaries

A. SAT Algorithm and Techniques

In this section, the basic concepts of SAT solving including the modern interpretation of the classical DLL algorithm [16] and the state-of-the-art SAT techniques are briefly reviewed. Given a Boolean formula f, the SAT problem is defined as the question whether there exists an assignment a such that $f(a) = 1$. The task of a SAT solver is to determine such an assignment or to prove that no such assignment exists. Common SAT solvers take a formula represented in CNF as input. A CNF or SAT instance Φ is a conjunction of clauses. A clause is a disjunction of literals. Literals are Boolean variables in positive or negative polarity. The CNF Φ is satisfied if all clauses are satisfied. A clause is satisfied if at least one of its literals is satisfied.

The pseudocode of the DLL algorithm is shown in Figure 1. The algorithm searchs for an assignment which satisfies Φ by iteratively choosing values for variables (*decide()*). The resulting implications of this decision are detected by the *deduce()* function (also called BCP). If *deduce()* detects a conflict, i.e. all literals in at least one clause evaluate to false, the function *resolveConflict()* tries to resolve this conflict by undoing former decisions, i.e. backtracking. If the conflict cannot be resolved, Φ is unsatisfiable. A solution is found when all variables are assigned and no conflict exists, i.e. Φ is satisfiable.

If a conflict occurs in modern SAT solvers, it is analyzed. As a result, non-chronological backtracking is performed and a learned clause (or conflict clause) is added to the SAT instance. Learned clauses prevent the SAT solver at an early stage from running into the same conflict again and by this prune the search space. For a detailed description of non-chronological backtracking and conflict analysis, we refer to [2], [17]. If different SAT instances share a large number of clauses, e.g. coming from the same part of a circuit, it is possible to share conflict clauses between subsequent instances to reduce the overall run time. This concept is also known as incremental SAT [18].

Table I
CNF for Gate Types

gate type	CNF
$c = a$ AND b	$(c + \overline{a} + \overline{b}) \cdot (\overline{c} + a) \cdot (\overline{c} + b)$
$c = a$ NAND b	$(\overline{c} + \overline{a} + \overline{b}) \cdot (c + a) \cdot (c + b)$
$c = a$ OR b	$(\overline{c} + a + b) \cdot (c + \overline{a}) \cdot (c + \overline{b})$
$c = a$ NOR b	$(c + a + b) \cdot (\overline{c} + \overline{a}) \cdot (\overline{c} + \overline{b})$
$c =$ NOT a	$(\overline{c} + \overline{a}) \cdot (c + a)$

B. Circuit-to-CNF Conversion

ATPG problems have their origin in the circuit structure. However, modern SAT solvers work on a problem in CNF representation. Therefore, the problem instance has to be converted from circuit structure to CNF. This conversion is briefly reviewed. More information can be found in [19].

Given a circuit $C = (G, S)$ where G is the set of gates and S the set of signal lines in C, the CNF Φ_C of the circuit is derived as follows. A Boolean variable is assigned to each signal line $s \in S$ of the circuit. The value corresponds to the logic value on this signal line. Each gate $g \in G$ is then converted to a set of clauses Φ_g by building the characteristic function $\chi(g)$ via a truthtable or algebraic conversion.

The clauses of the basic gate types with inputs a, b and output c can be found in Table I. The CNF Φ_C of circuit C is defined as the conjunction of the CNF Φ_g of each gate $g \in G$. The final SAT instance Φ_{SAT} is then obtained by a conjunction of Φ_C with the fault specific constraints, e.g. the fault, in CNF Φ_{con} :

$$\Phi_{SAT} = \Phi_C \cdot \Phi_{con}$$

Note that a fault usually affects only a small part of the circuit. In order to make the SAT instance as small as possible, Φ_C only contains the CNF of this part.

III. Dynamic Clause Activation

In this section, the *Dynamic Clause Activation* is described in detail. At first, the overall algorithm is given in Section III-A. A detailed description of the clause activation methodology is then presented in Section III-B.

A. Overall Algorithm

The approach exploits the property of circuit-oriented problems that Φ_C contains no contradiction itself and unsatisfiability has its reason in the given fault-specific constraints Φ_{con}. If Φ_{con} is justified considering Φ_C, then the SAT instance is satisfiable. In contrast to common SAT solvers, the approach does not work on a static SAT instance Φ_{SAT}, but on a dynamic SAT instance named Φ_{dyn} which is extended during the search process. To allow for a dynamic extension of the SAT instance, the proposed SAT engine using DCA is divided into two parts: a *circuit part* and an *algorithmic part*. This is illustrated in Figure 2.

The circuit part contains the complete CNF of the circuit (Φ_C) and serves as a database, whereas the algorithmic part contains the search algorithm working on a local CNF Φ_{dyn}. The search algorithm consists of state-of-the-art SAT

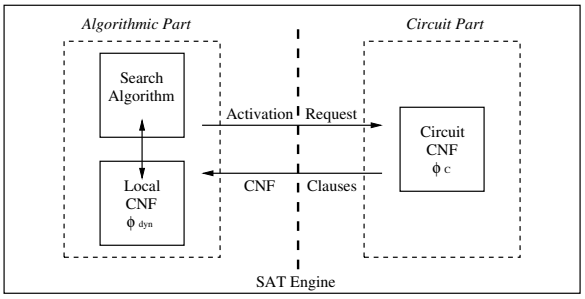


Figure 2. Division of the Proposed SAT Engine

techniques such as conflict analysis and fast BCP. The search algorithm is executed only on Φ_{dyn}. At the beginning of the search, only the clauses of Φ_{con}, i.e. the fault-specific constraints, are stored in Φ_{dyn}.

The DCA technique can be integrated into the DLL procedure (shown in Figure 1). An activation request is sent to the circuit part whenever a variable is assigned, i.e. in the functions *decide()* and *deduce()*. By this, Φ_{dyn} is dynamically extended during the search process. If a conflict in Φ_{dyn} cannot be resolved, then the SAT instance is UNSAT, i.e. the fault is redundant, because any extension of Φ_{dyn} would also result in UNSAT. By this, redundant faults can be classified much faster when the reason of the untestability is locally bounded. In this case, the complete CNF does not have to be built. In contrast, SAT (testable), is determined if no more activation requests are sent and all activated clauses are satisfied. It is important to point out that each request has to be sent before doing BCP for this assignment.

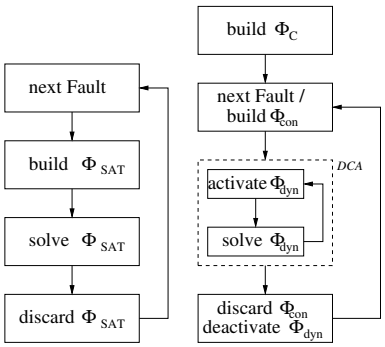
After solving one formula, Φ_{dyn} is cleared. All clauses are deactivated. The classical test generation flow as well as the alternated flow incorporating DCA is shown in Figure 3(a) and Figure 3(b), respectively. The time consuming step of circuit-to-CNF conversion is done only once instead of for each fault. This step is replaced by the activation methodology which can be efficiently implemented using pointers to clauses. Only Φ_{con} has to be extracted for each run. Its size is typically very small.

The correct and efficient handling of the activation methodology is crucial in this procedure. Therefore, a detailed description of this technique and of the integration of structural knowledge are given below.

B. Efficient Activation Methodology Using Implicit ODCs

For the activation methodology, the *Structural Watch List* (SWL) – a new data structure which is part of the circuit part – is proposed. Two entries in the SWL are assigned to each variable x in the circuit: one for the positive form and one for the negative form (similar to the watch list used for fast BCP [17]). In other words, one entry is assigned to each literal of x. Each entry in the list contains again a list storing those clauses which should be activated, i.e. added to Φ_{dyn}, if the corresponding value is assigned to x, e.g. if 0 is assigned to x, the clauses registered at $\overline{x}$ are activated.

The activation methodology is motivated by the following property of a circuit-oriented problem: if the controlling value of a gate g is assigned to an input of g, the remaining inputs of g do not influence the output value of g anymore. The value of the other inputs can be considered as don't care.[1] Furthermore, if the problem is to justify certain values



(a) Classic (b) Dynamic

Figure 3. SAT-based ATPG flow

in the circuit, these don't care inputs do not influence the search at all. A variable which does not influence the outcome of the search process can be denoted as ODC as defined in [13]. Based on this definition, the following activation methodology is proposed.

Let s be a signal line, v_s the corresponding Boolean variable and g_s the gate with s as the outgoing line. Further, let Φ_{g_s} be the set of clauses describing the logic function of g_s. If either 0 or 1 is assigned to v_s, only those clauses of Φ_{g_s} are activated which are *not* satisfied under the current assignment of v_s. As a result, some of the input variables $i_1, \ldots, i_n$ of g_s may remain unconstrained and, consequently, unassigned during the search process, because they do not influence the outcome of the search process. Hence, such a variable can be considered as ODC. Because the variable is not explicitly declared as ODC but only remains unconstrained, it is called *Implicit ODC*. Due to the property that ODCs do not influence the outcome of the search, the procedure is still complete.

The SAT instance is satisfiable if, and only if, all activated clauses are satisfied. Consequently, the break condition differs from those of modern SAT solvers, where the instance is usually satisfiable when all variables are assigned and no conflict exists. A further improvement of the break condition is the use of a list similar to the j-frontier known from ATPG algorithms (*j-list*) [6]. Clauses that are activated and therefore have to be satisfied are stored in the j-list and can also be removed during backtracking. Generally, a clause c can be removed from the j-list if the assignment that caused the activation of c is undone. (Due to page limitations, the concrete modeling of the j-list is not described here.) The following example demonstrates the activation procedure.

Example 1: Consider the circuit shown in Figure 4. The corresponding CNF is given in Table II. Each clause is registered in the SWL at one literal. This literal is denoted in column *entry*. Suppose the target is to check whether the output of the circuit h can be 0: $\Phi_{con} = (\overline{h})$.

The procedure of the SAT algorithm with DCA runs as follows. At first, 0 is assigned to h. Clause (1) is activated and inserted into the j-list due to the SWL entry. Suppose the decision heuristic then chooses f to be assigned to 0 to satisfy clause (1). As a consequence, clause (4) is activated and inserted into the j-list. A decision $a = 1$ then satisfies clause (4). Both clauses contained in the j-list are satisfied

[1]Contrary to structural ATPG algorithms, common SAT solver cannot explicitly assign a variable with a don't care value. Their efficiency is based in part on the fact that their search is defined over Boolean variables.

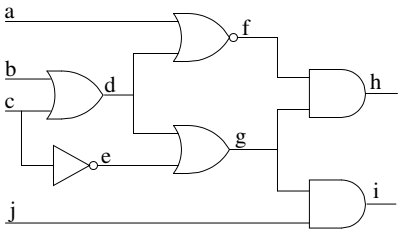


Figure 4. Example Circuit

Table II
CNF AND SWL ENTRIES FOR THE EXAMPLE CIRCUIT IN FIGURE 4

Cls.No.	entry	Clause	Cls.No.	entry	Clause
(1)	$\overline{h}$	$(h+\overline{f}+\overline{g})$	(10)	d	$(\overline{d}+b+c)$
(2)	h	$(\overline{h}+f)$	(11)	$\overline{d}$	$(d+\overline{b})$
(3)	h	$(\overline{h}+g)$	(12)	$\overline{d}$	$(d+\overline{c})$
(4)	$\overline{f}$	$(f+a+d)$	(13)	e	$(\overline{e}+\overline{c})$
(5)	f	$(\overline{f}+\overline{a})$	(14)	$\overline{e}$	$(e+c)$
(6)	f	$(\overline{f}+\overline{d})$	(15)	$\overline{i}$	$(i+\overline{g}+\overline{j})$
(7)	g	$(\overline{g}+d+e)$	(16)	i	$(\overline{i}+g)$
(8)	$\overline{g}$	$(g+\overline{d})$	(17)	i	$(\overline{i}+j)$
(9)	$\overline{g}$	$(g+\overline{e})$			

and consequently the SAT instance is satisfiable with the partial assignment $(a = 1, f = 0, h = 0)$. The variables d and g or their transitive fanin cones become implicit ODCs, because their values remain unassigned.

IV. INTEGRATION OF DYNAMIC LEARNING

Common SAT algorithms benefit significantly from the learned information in terms of conflict clauses. When a conflict occurs during the search process, a conflict clause is learned which avoids running in the same conflict again. As a result, large parts of the search space are pruned. Recently, it was shown in [9] that the reuse of learned information, i.e. conflict clauses, can reduce the number of unclassified faults significantly. Here, it is shown how dynamic learning techniques can be integrated into the DCA technique.

After each run, those conflict clauses which can be reused are extracted and stored into a watch-list. In [9], a variable-based activation scheme is used. A learned clause c is added to the SAT instance Φ_{SAT} when the corresponding "watched" variable is included in Φ_{SAT}. Here, a *literal-based activation scheme* is proposed. A *Learned Watch List* (LWL) which is very similar to the SWL is used to keep track of the learned clauses. The LWL is also contained in the circuit part. In this scheme, a learned clause $c = (l_1, \ldots, l_n)$ is registered in the list of one or more literals contained in c. Then, the existing activation methodology is used to activate the learned clauses in the same way the original circuit clauses are activated.

None of the learned clauses is activated before the search process starts. The learned clauses are dynamically added to Φ_{dyn} during the search process. When an activation request for a literal l_i is sent, not only the clauses from the SWL are activated, but additionally those registered at $\overline{l}_i$ in the LWL. Learned clauses registered at literals from other parts of the circuit are explicitly *not* touched. The same holds for clauses registered at l_i (unless l_i becomes negated). The activated learned clauses are treated differently only in one way. Because learned clauses are redundant, they are ignored in the break condition.

Example 2: Again, consider the circuit shown in Figure 4. Assume that in a previous run, the conflict clause $c = (\overline{h}+\overline{b})$ was learned. The conflict clause is registered in the LWL in the list of literal b. When $b = 1$ occurs during the search process, c is activated and $h = 0$ is directly implied. Otherwise, when $b = 0$, c remains deactivated since it is not registered at $\overline{b}$.

V. EXPERIMENTAL RESULTS

In this section, experimental results are presented. The proposed techniques are implemented in C++ on top of the data structures of the SAT solver MiniSat [4] resulting in the new SAT engine *DynamicSAT*. DynamicSAT in combination with the dynamic learning techniques proposed in [9] is named *DynamicSAT+*.

Experiments were conducted for three different fault models: path delay, stuck-at and transition. The concrete SAT modeling of the fault models is slightly different than for classical SAT algorithms. Details cannot be given due to page limitations. The experiments for path delay test generation (non-robust) were run on a 64-bit Intel Xeon (GNU/Linux, 3GHz, 32,768MB), whereas the experiments for stuck-at and transition test generation were run on a 64-bit AMD Opteron (GNU/Linux, 2.8GHz, 32,768MB).

The experiments were performed on large industrial circuits containing unknown states and tri-state elements provided by NXP Semiconductors, Germany. Note that the name of the circuit roughly denotes the size of the circuit, e.g. p2787k contains over 2.7 million elements. For path delay test generation, DynamicSAT is compared to the state-of-the-art SAT solver MiniSat [4] and KF-ATPG [15] which uses a path-status graph to keep track of unsensitizable path segments and by this works in an incremental manner. Since KF-ATPG does not support unknown states and tri-state elements, the comparison is made with ISCAS'89 and ITC'99 benchmarks. For stuck-at and transition test generation, the proposed approaches are compared to a highly optimized industrial FAN-based algorithm as well as to the SAT-based ATPG approach PASSAT [7] which uses MiniSat as core engine.

The total memory usage of DynamicSAT is higher than of PASSAT. This is due to the complete circuit CNF which is kept in the memory during the search process. Overall, twice as much memory is needed for the complete ATPG process. Reducing the memory consumption of DynamicSAT, e.g. by circuit partitioning, is therefore future work.

A. Path Delay Faults

Table III shows the experimental results for non-robust path delay test generation. In the upper part, DynamicSAT is compared to KF-ATPG. For each circuit, 1,000,000 path delay faults are chosen randomly (except s35932, which contains only 394,282 path delay faults). Since KF-ATPG does not support a timeout, no timeout was used for the other approaches to allow for a fair comparison. The run time is measured in CPU minutes (m) or CPU hours (h) and presented in columns entitled *Time*. The fastest approach is highlighted. DynamicSAT outperforms KF-ATPG clearly by up to a factor of 67 (for s35932). The average speed-up factor is 14. The run time can often be further reduced by DynamicSAT+.

In the lower part of the table, DynamicSAT is compared to MiniSat on industrial circuits. The search process of a SAT solver can be measured in restarts. The timeout was

Table III
EXPERIMENTAL RESULTS FOR PATH SENSITIZATION

Circ	KF-ATPG Ab.	KF-ATPG Time	DynamicSAT Ab.	DynamicSAT Time	DynamicSAT+ Ab.	DynamicSAT+ Time
s15850	-	11:16m	-	2:06m	-	**1:41m**
s35932	-	6:42m	-	**0:06m**	-	0:09m
s38417	-	48:10m	-	6:08m	-	**3:49m**
s38584	-	16:14m	-	**2:49m**	-	3:27m
b14	-	24:39m	-	1:33m	-	**1:14m**
b15	-	12:26m	-	**0:44m**	-	0:45m
b17	-	18:43m	-	2:55m	-	**0:38m**
b18	-	1:13h	-	22:12m	-	**5:24m**
b19	-	2:08h	-	1:22h	-	**19:59m**
b20	-	53:31m	-	3:11m	-	**2:01m**
b21	-	55:48m	-	3:20m	-	**1:59m**
b22	-	3:16h	-	27:27m	-	**11:49m**
Circ	MiniSat Ab.	MiniSat Time	DynamicSAT Ab.	DynamicSAT Time	DynamicSAT+ Ab.	DynamicSAT+ Time
p177k	2,453	3:06h	**7**	27:34m	**0**	28:51m
p462k	**0**	14:14m	**1**	6:13m	**0**	6:14m
p565k	212	8:24m	529	11:20m	57	15:06m
p1330k	**0**	24:40m	**3**	9:09m	**0**	9:46m
p2787k	133	50:00m	26	28:30m	**0**	28:46m
p3327k	554	1:52h	**15**	47:50m	**2**	46:35m
p3852k	1,887	3:23h	**1**	57:48m	**0**	57:07m

set to 7 restarts for each fault. For each circuit, 20,000 path delay faults describing a (long) path with more than 50 gates are chosen for test generation. Faults which could not be classified within the interval are called *unclassified*. In order to indicate the robustness of each single approach, those numbers are highlighted (in column *Ab.*) which denotes that the number of unclassified faults is less than 0.1% of the total number of targeted faults.

For nearly all circuits (except p565k), DynamicSAT is faster than MiniSat. The highest speed-up factor is 6.7 (p177k). The average speed-up factor is 2.9. Besides the reduced run time, the number of unclassified faults can also be reduced significantly. In total, the number of unclassified faults is reduced by 89%. The integration of dynamic learning techniques (DynamicSAT+) leads to comparable run time (except for p565k), but the number of unclassified faults is further reduced by 90% compared to DynamicSAT.

B. Stuck-at Faults

The experimental results for stuck-at test generation are presented in Table IV. The results of the highly optimized industrial FAN-based algorithm are shown in column *FAN* and *FAN inc*. FAN inc uses a highly increased backtrack limit to reduce the number of unclassified faults. The results for PASSAT, DynamicSAT and DynamicSAT+ are presented in the corresponding columns. The timeout was set to 7 restarts for each fault. The number of unclassified faults is highlighted if the number is less than 0.1% of all faults. All experiments were conducted with fault dropping enabled.

The average number of specified bits are reported in column *Bits*. PASSAT uses a post-processor to reduce the specified bits as described in [20]. Note that the number of specified bits of all SAT-based approaches are higher than for the industrial FAN algorithm which cannot be given here. The presented numbers show that DynamicSAT produces slightly less specified bits on average than PASSAT with a post-processor.

The fastest ATPG approach is clearly the FAN algorithm having the smallest run time for nearly all circuits. However, FAN produces a large number of unclassified faults. The FAN inc approach can reduce the number of unclassified faults only slightly, but needs much more run time. PASSAT can reduce the number of unclassified faults drastically compared to both FAN configurations. However, the run time increases significantly.

Using the proposed DynamicSAT approach, the run time of SAT-based ATPG can be significantly reduced up to a factor of 10 compared to the run time needed by PASSAT. The run time is reduced by a factor of 3.7 on average. At the same time, the number of unclassified faults is in most cases equal or only slightly increased. As a result, DynamicSAT is much faster than PASSAT and retains the low number of unclassified faults. The run time of DynamicSAT+ is comparable to the run time of DynamicSAT, but the number of unclassified faults could be considerably reduced.

C. Transition Faults

The experimental results for transition faults are presented in Table V. Here, a general timeout of 72 hours is used. Concerning the average number of specified bits, DynamicSAT has in most cases less specified bits than PASSAT with post-processor (up to a factor of 4). DynamicSAT+ even increases the average number of don't cares. As it can be seen at the increased run time and grown number of unclassified faults, ATPG for transition faults is more complex than for stuck-at faults. Here, the number of unclassified faults is highlighted if the number is less than 1% of all faults.

The FAN approach can be again considered as the fastest algorithm. However, the advance is not as clear as for the stuck-at fault model. Furthermore, the number of unclassified faults is very large. FAN inc does not perform very well. The reduction of unclassified faults is for most circuits small, but the increase in run time is very high. For most circuits, PASSAT produces constantly fewer unclassified faults compared to the FAN approaches. Compared to PASSAT, DynamicSAT can decrease the run time for most circuits. For p2787k, except FAN, all approaches could not process all faults during 72 hours. Instead of the run time, the relative number of processed faults (of 4,063,500 faults) is given in the respective column.

Most notable is the run time reduction for circuit p177k. Here, a speed-up factor of over 60 can be achieved. At the same time, the number of unclassified faults is significantly decreased. The average speed-up factor is 9.6. For some circuits, DynamicSAT even outperforms the FAN algorithm. Generally, the run time difference between FAN and SAT-based ATPG can be considerably reduced. Again, the run time of DynamicSAT and DynamicSAT+ is comparable, but the number of unclassified faults is significantly reduced. DynamicSAT+ is the only approach for which the number of unclassified faults is less than 1% for all circuits.

VI. CONCLUSIONS

SAT-based ATPG algorithms are very robust for hard-to-test faults but suffer from the overhead for easy-to-test faults. Classical structural ATPG algorithms are usually very fast, but have problems to cope with hard-to-test faults which occur more and more frequently in today's complex designs. The new SAT technique of *Dynamic Clause Activation* (DCA) was presented for speeding up SAT-based ATPG and, at the same time, retaining the high level of robustness.

Experimental results on large industrial circuits for different fault models have shown that the proposed techniques

Table IV
EXPERIMENTAL RESULTS FOR ATPG FOR STUCK-AT FAULTS

	FAN		FAN inc		PASSAT			DynamicSAT			DynamicSAT+		
Circ	Ab.	Time	Ab.	Time	Bits	Ab.	Time	Bits	Ab.	Time	Bits	Ab.	Time
p44k	**0**	1:13m	**0**	1:13m	236	**0**	58:49m	102	**0**	5:40m	99	**0**	5:46m
p57k	197	1:21m	138	6:55m	90	**2**	4:29m	65	**7**	1:06m	59	**2**	1:08m
p77k	**0**	0:07m	**0**	0:07m	16	**0**	0:14m	10	**0**	0:08m	10	**0**	0:08m
p80k	**18**	1:41m	**12**	2:40m	115	**0**	5:33m	54	**0**	1:34m	54	**0**	1:34m
p88k	**56**	1:36m	**35**	2:34m	56	**0**	3:46m	51	**0**	1:39m	50	**0**	1:38m
p99k	988	1:29m	455	6:05m	52	**3**	1:54m	29	**17**	1:29m	29	**8**	1:44m
p177k	**99**	2:13m	**59**	3:32m	88	**3**	2:59h	307	**1**	7:00m	307	**1**	7:13m
p462k	1,009	9:17m	785	11:06m	45	**66**	1:25h	42	**409**	20:25m	42	**97**	24:55m
p565k	**263**	8:41m	**175**	11:21m	31	**0**	27:42m	30	**0**	19:57m	30	**0**	19:37m
p1330k	**412**	12:38m	**282**	13:33m	54	**0**	49:55m	55	**57**	25:49m	55	**47**	36:00m
p2787k	218,292	2:21h	165,699	11:53h	130	10,719	25:07h	92	43,806	13:11h	107	**212**	20:40h

Table V
EXPERIMENTAL RESULTS FOR ATPG FOR TRANSITION FAULTS

	FAN		FAN inc		PASSAT			DynamicSAT			DynamicSAT+		
Circ	Ab.	Time	Ab.	Time	Bits	Ab.	Time	Bits	Ab.	Time	Bits	Ab.	Time
p44k	**648**	18:45m	**539**	30:29m	333	**0**	5:38h	83	**30**	19:41m	74	**22**	19:34m
p57k	2,467	14:45m	**1,465**	1:06h	225	**44**	1:00h	115	**70**	17:07m	94	**48**	15:08m
p77k	49,452	9:05m	36,803	1:49h	37	5421	1:17h	34	10,772	52:57m	32	**0**	5:42m
p80k	4,527	10:56m	**3,022**	53:35m	326	**9**	12:15m	152	**9**	9:59m	150	**5**	8:44m
p88k	6,964	1:13h	3,833	3:54h	91	**0**	33:44m	93	**0**	36:15m	84	**0**	35:45m
p99k	13,448	56:14m	11,462	8:28h	59	**73**	21:55m	73	**85**	28:50m	69	**53**	30:12m
p177k	13,206	46:53m	7,992	3:25h	267	19,002	TO	137	**442**	1:11h	128	**181**	2:12h
p462k	17,452	1:15h	13,353	3:41h	64	**551**	7:09h	62	**122**	1:15h	58	**5**	1:14h
p565k	**9,708**	3:47h	**5,548**	7:37h	122	**412**	4:12h	110	**480**	2:43h	112	**178**	2:44h
p1330k	**11,328**	2:40h	**5,679**	4:12h	219	**3**	9:18h	52	**41**	2:47h	48	**11**	2:47h
p2787k	698,387	19:22h	321,025	*83%*	168	**34,689**	*70%*	95	116,492	*89%*	91	**5,357**	*80%*

are able to significantly reduce the run time of SAT-based ATPG by several factors on average and reduce or even close the run time gap between structural and SAT-based ATPG approaches. Furthermore, DynamicSAT produces only a small number of unclassified faults which can be further reduced by the integration of dynamic learning techniques. The application of the proposed techniques in the field of dynamic compaction and the reduction of the memory consumption is focus of future work.

ACKNOWLEDGMENT

Parts of this research work were supported by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA (01M3172B) and by the German Research Foundation (DFG) (DR 287/15-1).

REFERENCES

[1] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Trans. on Computers*, vol. 32, no. 12, pp. 1137–1144, 1983.

[2] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[3] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Design Automation Conf.*, 2001, pp. 530–535.

[4] N. Eén and N. Sörensson, "An extensible SAT solver," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[5] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 15, pp. 1167–1176, 1996.

[6] J. P. Marques-Silva and K. A. Sakallah, "Robust search algorithms for test pattern generation," in *Int'l Symp. on Fault-Tolerant Computing*, 1997, pp. 152–157.

[7] 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 Computer-Aided Design for Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[8] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille, *Test Pattern Generation using Boolean Proof Engines*. Springer, 2009.

[9] S. Eggersglüß and R. Drechsler, "Increasing robustness of SAT-based delay test generation using efficient dynamic learning techniques," in *IEEE European Test Symp.*, 2009, pp. 81–86.

[10] F. Lu, L.-C. Wang, K.-T. Cheng, and R. Huang, "A circuit SAT solver with signal correlation guided learning," in *Design, Automation and Test in Europe*, 2003, pp. 892–897.

[11] C.-A. Wu, T.-H. Lin, C.-C. Lee, and C.-Y. Huang, "QuteSAT: A robust circuit-based SAT solver for complex circuit structure," in *Design, Automation and Test in Europe*, 2007, pp. 1313–1318.

[12] M. K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver," in *Design Automation Conf.*, 2002, pp. 747–750.

[13] S. Safarpour, A. Veneris, and R. Drechsler, "Improved SAT-based reachability analysis with observability don't cares," *Jour. of Satisfiability, Boolean Modeling and Computation*, vol. 5, pp. 1–25, 2008.

[14] Z. Fu, Y. Yu, and S. Malik, "Considering circuit observability don't cares in CNF satisfiability," in *Design, Automation and Test in Europe*, 2005, pp. 1108–1113.

[15] K. Yang, K.-T. Cheng, and L.-C. Wang, "Trangen: a SAT-based ATPG for path-oriented transition faults," in *ASP Design Automation Conf.*, 2004, pp. 92–97.

[16] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Comm of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[17] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *Int'l Conf. on Computer-Aided Design*, 2001, pp. 279–285.

[18] O. Shtrichman, "Pruning techniques for the SAT-based bounded model checking problem," in *Correct Hardware Design and Verification Methods*, ser. LNCS, vol. 2144, 2001, pp. 58–70.

[19] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[20] S. Eggersglüß and R. Drechsler, "Improving test pattern compactness in SAT-based ATPG," in *IEEE Asian Test Symp.*, 2007, pp. 445–450.