# A Fast Untestability Proof for SAT-based ATPG

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

***Abstract*—Automatic Test Pattern Generation (ATPG) based on Boolean satisfiability (SAT) has been shown to be a beneficial complement to traditional ATPG techniques. Boolean solvers work on instances given in *Conjunctive Normal Form* (CNF). The required transformation of the ATPG problem into CNF is one main part of SAT-based ATPG and needs a significant portion of the overall run time. Solving the SAT instance is the other main part. Here, the time needed is often negligible – especially for untestable faults**

**This paper presents a preprocessing technique that accelerates the classification of untestable faults. Those occur more frequently with increasing design sizes in industrial practice. In order to avoid overhead on testable faults, an untestability prediction is motivated. This increases the robustness of the entire ATPG process. The efficiency of the proposed method is shown during the experiments.**

## I. Introduction

The continuous growth of today's circuit designs requires a constant improvement of state-of-the-art *Computer Aided Design* (CAD) tools. The post-production test is a vital step in the design flow. It ensures the functional correctness of a circuit. To guarantee high quality production, this step is very important.

In practice, a fault model is usually used to abstract from the physical defects. To test the circuit for correctness with respect to the fault model used, test patterns have to be computed. If there exists a test pattern for a particular fault $F$, then $F$ is called *testable*; otherwise $F$ is called *untestable*.

In this work, the stuck-at fault model is used. To generate a test pattern for a stuck-at fault, there exist many sophisticated algorithms. The D-algorithm [11] was the first algorithm that traversed the search space by backtracking. Improvements concerning decision strategies and propagation/justification were given in PODEM [5] and FAN [4]. Further algorithms are Socrates [12] and Hannibal [7]. All these algorithms have in common that they directly work on the circuit structure.

In contrast, there also exist approaches based on Boolean satisfiability (SAT) [8], [13], [14], [2]. In particular for hard-to-solve problem instances, SAT-based methods proved to be highly advantageous. Nowadays, SAT-based ATPG is a promising complement to the classical algorithms.

Since most modern SAT solvers (e.g. [9], [10], [6], [3]) work on an instance representation in *Conjunctive Normal Form* (CNF), a new SAT instance has to be generated for each fault[1]. In [15], it was shown that the run time needed for instance generation is a significant part of the overall run time and often even dominates it. Especially CNFs of untestable faults can mostly be solved very easily, because in industrial circuits the reason for the conflict is often bounded locally. In those cases, building the entire SAT instance is a large overhead.

With the growth of the industrial designs, the number of untestable faults increases considerably. Today's circuits contain hundreds of thousands of untestable faults. As a result, avoiding the above mentioned overhead can improve the robustness of the overall SAT-based ATPG process.

This paper presents a preprocessing method with the objective to accelerate the SAT instance generation by only building partial CNFs. During a detailed motivation it is shown that the technique is only useful for untestable faults. Therefore, the method is aimed for large industrial circuits. Here, significant speed-ups can be observed. To avoid useless preprocessing steps, a straightforward untestability prediction is given. Hence, using the technique does not slow down the ATPG process for small circuits. As a result, the overall robustness can be increased.

This work is structured as follows: in the next section a brief overview on SAT-based ATPG is given. In Section III the motivation is presented in more detail. A preprocessing technique is discussed in Section IV. Experimental results and conclusions are given in Section V and Section VI, respectively.

## II. Previous Work

To make the paper self-contained, this section presents a short overview on SAT-based ATPG. First, a general explanation is given. Afterwards, the generation of a CNF

[1]In preliminary studies, we also experimented with a circuit SAT solver, but did not consistently observe improvements in run time or memory use for ATPG.

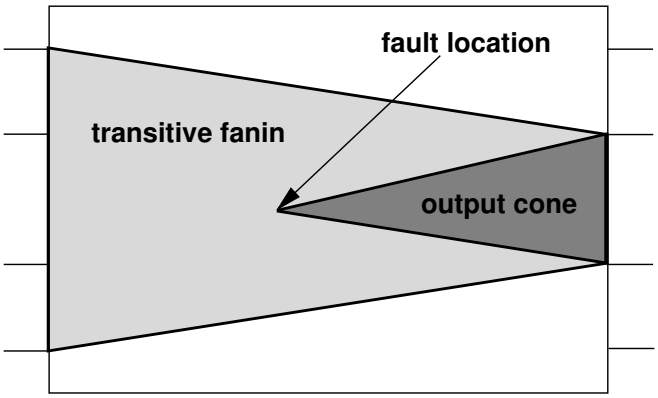



Fig. 1. Extraction of the influenced circuit parts

for a specific fault is illustrated. Finally, a run time analysis provides further insight.

### A. SAT-based ATPG

To create a test pattern for a stuck-at fault, an assignment to the inputs has to be found that guarantees at least one different output value between the faulty circuit and the faultless circuit. While classical algorithms work directly on the circuit structure to find such an assignment, in SAT-based ATPG the question whether there exists a test pattern for a particular fault $F$ is encoded into a Boolean formula, which is satisfiable if and only if $F$ is testable. Then, a SAT solver proves either satisfiability or unsatisfiability of the formula. A test pattern – if it exists – can be derived directly from the satisfying assignment.

Modern SAT solvers work on instances represented as CNFs. A CNF is a conjunction of clauses, a clause is a disjunction of literals and a literal is the positive or negative occurrence of a Boolean variable. A SAT instance is satisfied if all clauses are satisfied; a clause is satisfied if at least one of its literals is satisfied; a positive or a negative literal is satisfied if the respective variable is assigned positively or negatively, respectively.

How to transform an ATPG problem into a SAT instance is explained in the following.

### B. Circuit-to-CNF Conversion

Consider the schematically depicted circuit in Figure 1. Here, a brief overview on the circuit-to-CNF conversion is given.

After the fault location has been marked, the fault site's output cone is traversed by a depth first search. This determines all *Primary Outputs* (POs) that may be influenced by the fault, i.e. all POs where a difference between the faulty circuit and the faultless circuit could be observed. The transitive fanin of these POs influences the detection of the fault and must be marked, too. To generate the SAT instance for the given fault, this part of the circuit has to be considered.

As introduced in [13], two Boolean variables $g_c$ and $g_f$ are assigned to each gate $g$ in order to represent the gate's value in the correct circuit and in the faulty circuit, respectively. A gate's CNF is generated by building its characteristic function. The conjunction of all CNFs results in the CNF for the circuit.

To find a difference between the correct circuit and the faulty circuit, an additional Boolean variable $g_d$ is assigned to each gate. If the variable $g_d$ is true, the gate's values in both circuits differ. Therefore, the constraint

$$g_d = 1 \rightarrow g_c \neq g_f$$

is added to the CNF in form of the two clauses

$$(\overline{g_d} + g_c + g_f) \cdot (\overline{g_d} + \overline{g_c} + \overline{g_f}).$$

To compute a test pattern for a fault, there must be a path from the fault site to an output, where the assignment of each variable $g_d$ is true. Following the notation in [13], this path is called a *D-chain*. Therefore, if a gate is on a D-chain, one successor must be on a D-chain as well. This property – encoded by the constraint

$$g_d \rightarrow \bigvee_{i=1}^{n} h_d^i,$$

where the gates $h^1, \ldots, h^n$ denote the successors of gate $g$ – is also added to the CNF. Moreover, the variable $g_d^f$, where the gate $g^f$ represents the faulty gate, is set to true in order to inject a difference at the fault site.

As a result, the SAT instance, generated this way, is satisfiable if and only if a D-chain exists, i.e. the SAT instance is satisfiable if and only if the fault is testable.

Finally, as described earlier, this SAT instance is given to a SAT solver. After the classification, the CNF is completely discarded. Therefore, the circuit-to-CNF conversion has to be done for each single target faults.

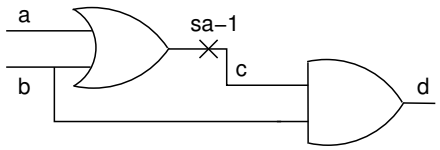
### C. Incremental Instance Generation

In [15] a detailed run time analysis of state-of-the-art SAT-based ATPG algorithms applied to industrial circuits is given. The two basic steps – SAT instance generation and solving – are compared with respect to their run time. Moreover, the classification result was included into this comparison.

It was shown that two main observations hold:

- Surprisingly, the generation time exceeds often the solving time.
- The solving time of testable instances exceeds the solving time of untestable instances significantly.

Based on these observations, an incremental solving scheme was proposed. By generating partial SAT instances




(a) Circuit $C$

$$
\begin{aligned}
\omega_1 &: \quad (\overline{c_c} + a_c + b_c) \\
\omega_2 &: \quad (c_c + \overline{a_c}) \\
\omega_3 &: \quad (c_c + \overline{b_c}) \\
\omega_4 &: \quad (d_c + \overline{b_c} + \overline{c_c}) \\
\omega_5 &: \quad (\overline{d_c} + b_c) \\
\omega_6 &: \quad (\overline{d_c} + c_c) \\
\omega_7 &: \quad (d_f + \overline{b_c} + \overline{c_f}) \\
\omega_8 &: \quad (\overline{d_f} + b_c) \\
\omega_9 &: \quad (\overline{d_f} + c_f) \\
\omega_{10} &: \quad (\overline{d_d} + d_c + d_f) \\
\omega_{11} &: \quad (\overline{d_d} + \overline{d_c} + \overline{d_f}) \\
\omega_{12} &: \quad (\overline{c_d} + c_c + c_f) \\
\omega_{13} &: \quad (\overline{c_d} + \overline{c_c} + \overline{c_f}) \\
\omega_{14} &: \quad (\overline{c_d} + d_d) \\
\omega_{15} &: \quad (c_f) \\
\omega_{16} &: \quad (c_d)
\end{aligned}
$$

(b) CNF $\phi_C$

Fig. 2. Example for an untestable fault

it is possible to speed up the entire ATPG process significantly.

However, the proposed approach achieves better results only for testable faults. If the considered fault is untestable, the incremental method creates little overhead. Due to the significant larger portion of testable faults in an industrial circuit, however, the speed-ups outweigh this drawback.

## III. Motivation

This section gives the motivation for a preprocess, that is able to accelerate the SAT-based ATPG for untestable faults.

Following the observations in Section II-C, untestability is often proven in almost no time, i.e. unsatisfiability of the respective SAT instance is shown in a few propagation steps only. In this case, the conflict occurs due to a contradiction in the circuit. Figure 2 gives an example.

In circuit $C$, depicted in Figure 2(a), a stuck-at 1 fault is modeled on signal line $c$. Obviously, the fault is untestable, since it is impossible to inject a difference (signal $b$ has to be set to 0) and to propagate it (signal $b$ has to be set to 1) at the same time.

Figure 2(b) shows the SAT instance $\phi_C$, describing this particular fault. The notation follows Section II-B. Due to clauses $\omega_{15}$ (that models the injection of the fault) and $\omega_{16}$ (that models the propagation of the difference), the propagation will lead to a conflict. Therefore, analog to the circuit-based algorithm, the fault is proven to be untestable by a directly implied contradiction.

In both classical ATPG and SAT-based ATPG, this contradiction is bounded locally. Even if the circuit $C$ is a subcircuit of large design, denoted by $C'$, the conflict occurs immediately. Let $\phi_{C'}$ be the CNF that describes $C'$. Since $\phi_{C'}$ contains $\phi_C$ as *unsatisfiable core*[2], the SAT instance is proven to be unsatisfiable just by a few propagation steps.

Those unsatisfiable cores can be found frequently in SAT-based ATPG for industrial designs. With the growing design sizes, this number even increases. However, the reason for untestability may not occur as directly as shown in the example, but rather due to restrictions to the primary inputs. The fast classification can be made thus easily anyway.

As a result, a SAT instance describing an untestable fault is generated completely, although only an instance describing an unsatisfiable core, i.e. describing the subcircuit, would be sufficient to classify the fault. As mentioned in Section II-C, the run time for generating an instance is a significant part of the overall run time. Building a complete instance where a partial one is sufficient is, therefore, an avoidable overhead.

A technique to overcome this drawback, is proposed in the next section.

## IV. Preprocessing Method

As mentioned in the previous section, untestable faults in industrial designs are often easy to classify, since the reason for the untestability is bounded locally. During preliminary studies it turned out, that most untestable faults (about 90%) can be classified only by considering the fault site's fanin cone in conjunction with restrictions to the primary inputs during the SAT instance generation. Figure 3 gives an illustration. It can be seen that the circuit part, considered during circuit-to-CNF conversion, is significantly smaller than using the traditional method (illustrated in Figure 1). Although only the marked part of the circuit is transformed into CNF, the fault is proven to be untestable. In this context, the fanin cone marked in Figure 3 is an unsatisfiable core of the complete SAT instance, generated following the traditional method.

This property can be used to perform a preprocessing step. Transforming only the fault site's fanin cone into CNF needs significantly less time than generating the entire SAT instance. If this partial CNF is unsatisfiable,

[2]A CNF $\chi$ is called *unsatisfiable core* of an unsatisfiable CNF $\chi'$ if $\chi$ is a sub-CNF of $\chi'$ that is already unsatisfiable.

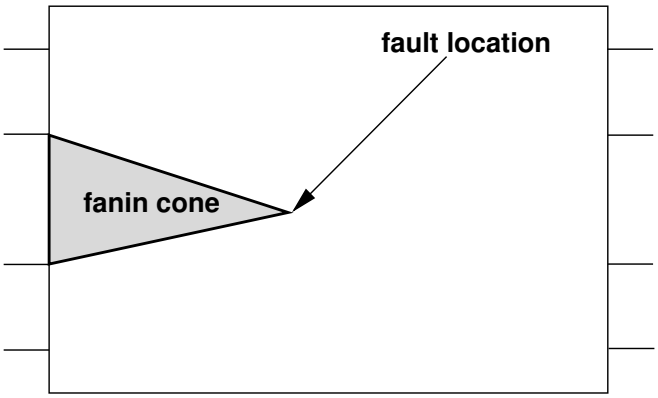



Fig. 3. Extraction of circuit parts in a preprocessing step

the fault is classified correctly. Otherwise, the entire SAT instance has to be generated.

However, the described technique is only useful if the fault is untestable. Otherwise, a classification can never be given after solving this partial CNF, i.e. it is overhead. As a result, the preprocess should only be applied to prove the untestability of faults, that are likely to be untestable. Although, certainly, the classification result is unknown before starting the ATPG process, in the following, a straightforward way to estimate the classification result of a particular fault is given.

Fault lists are usually not generated randomly, i.e. consecutive faults are mostly in the same region of the circuit. Moreover, faults in the same region often share the same properties, e.g. due to restrictions it is impossible to inject a fault. Therefore, the classification result of entire circuit parts may be equivalent.

Based on this observation, it is likely that a fault is testable if the previous fault is testable and, analog, a fault is untestable if the previous fault is untestable. To confirm this hypothesis, Table I presents information achieved by an ATPG run. The first column shows the name of the circuit. Both the publicly available ITC'99 benchmarks [1] as well as industrial circuits, provided by NXP Semiconductors, have been considered. Column *P(0|0)* gives the conditional probability (in percent) for a fault to be untestable if the previous fault is untestable. Analog, column *P(1|1)* gives the conditional probability for a fault to be testable if the previous fault is testable.

As can be seen, the value *P(0|0)* is for almost all industrial circuits, i.e. except for p44k, p88k and p99k, greater than 90 percent. For the ITC'99 benchmarks, however, this value is always less than 75 percent. The value *P(1|1)* is for all circuits, except for p77k, greater than 80 percent. These observations strengthen the hypothesis made above.

In the following, the proposed preprocessing technique is summarized. In order to speed up the run time for

TABLE I
PROBABILITY OF A FAULT TO HAVE THE SAME CLASSIFICATION RESULT AS THE PREDECESSOR IN THE FAULT LIST

| Circuit | P(0\|0) | P(1\|1) |
|---|---|---|
| b14 | 51.92 | 98.31 |
| b15 | 69.73 | 94.11 |
| b17 | 72.76 | 94.70 |
| b18 | 71.94 | 97.14 |
| b20 | 46.22 | 97.54 |
| b21 | 52.25 | 97.52 |
| b22 | 30.02 | 97.18 |
| p44k | 69.90 | 95.79 |
| p77k | 95.81 | 65.33 |
| p80k | 95.16 | 99.94 |
| p88k | 72.82 | 97.19 |
| p99k | 83.02 | 96.38 |
| p141k | 96.49 | 97.18 |
| p177k | 96.42 | 97.05 |
| p456k | 90.23 | 86.86 |
| p462k | 96.80 | 83.79 |
| p565k | 93.20 | 92.09 |
| p1330k | 93.10 | 89.26 |
| p2787k | 98.33 | 83.62 |
| p3327k | 90.23 | 95.65 |
| p3852k | 92.28 | 94.52 |

generating a SAT instance, only a partial CNF, consisting of the fault site's fanin cone, is built up prior to the actual TPG process for a particular fault. If this SAT instance is unsatisfiable, the fault is classified to be untestable and the next fault can be considered. Otherwise, the entire CNF has to be generated. Since this method is promising only for untestable faults, it is only applied if the previous fault is untestable.

## V. Experimental Results

In this section, experimental results are given. The preprocessing approach, described in the last section, was implemented as a prototype into the ATPG tool of NXP Semiconductors. MiniSat [3] was used to solve the SAT instances. All experiments were carried out on an Intel Xeon System (3.4 GHz, 32 GByte, Linux).

Two benchmark sets have been considered: the publicly available ITC'99 benchmarks [1] and industrial circuits, provided by NXP Semiconductors Germany GmbH, Hamburg, Germany. The names of the NXP benchmarks indicate the number of elements contained in a circuit, e.g. the circuit p3852k consists of approximately 3.85 million elements.

Table II gives an overview on the overall run times of the ATPG process. The circuit's name is shown in the first column. The second column presents the number of targets, i.e. the number of faults after fault collapsing. The number of untestable targets is given in the third column. This information is important since the preprocessing step aims to accelerate the test pattern generation process of

TABLE II
RUN TIMES FOR THE ATPG PROCESS

| Circuit | Targets | Untestable | Traditional | | Fast all | | | | Fast estimated | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Ab. | Time | Ab. | Class. | Prep. | Time | Ab. | Class. | Prep. | Time |
| b14 | 22,700 | 156 | 0 | 0:56m | 0 | 0:55m | 0:10m | 1:06m | 0 | 0:55m | 0:00m | 0:55m |
| b15 | 21,850 | 727 | 0 | 1:07m | 0 | 1:04m | 0:10m | 1:14m | 0 | 1:05m | 0:02m | 1:07m |
| b17 | 76,493 | 1,958 | 0 | 2:54m | 0 | 2:48m | 0:32m | 3:20m | 0 | 2:48m | 0:04m | 2:52m |
| b18 | 264,043 | 2,844 | 0 | 9:06m | 0 | 8:46m | 2:02m | 10:48m | 0 | 8:57m | 0:03m | 9:00m |
| b20 | 45,461 | 319 | 0 | 2:14m | 0 | 2:12m | 0:27m | 2:39m | 0 | 2:12m | 0:01m | 2:13m |
| b21 | 46,156 | 378 | 0 | 2:22m | 0 | 2:21m | 0:27m | 2:48m | 0 | 2:21m | 0:01m | 2:22m |
| b22 | 67,540 | 344 | 0 | 2:48m | 0 | 2:47m | 0:32m | 3:19m | 0 | 2:48m | 0:01m | 2:49m |
| p44k | 64,105 | 2,385 | 0 | 49:21m | 0 | 46:54m | 8:35m | 55:29m | 0 | 48:51m | 0:34m | 49:25m |
| p77k | 163,310 | 9,181 | 0 | 0:27m | 0 | 0:25m | 0:01m | 0:26m | 0 | 0:25m | 0:00m | 0:25m |
| p80k | 197,834 | 124 | 0 | 6:33m | 0 | 6:32m | 0:12m | 6:44m | 0 | 6:32m | 0:00m | 6:32m |
| p88k | 147,742 | 2,640 | 0 | 2:14m | 0 | 2:06m | 0:21m | 2:37m | 0 | 2:12m | 0:01m | 2:13m |
| p99k | 162,019 | 2,141 | 0 | 1:35m | 0 | 1:34m | 0:28m | 2:02m | 0 | 1:35m | 0:00m | 1:35m |
| p141k | 267,948 | 13,815 | 0 | 3:02h | 0 | 2:48h | 3:54m | 2:52h | 0 | 2:49h | 0:04m | 2:49h |
| p177k | 268,176 | 13,840 | 0 | 2:37h | 0 | 2:20h | 3:44m | 2:24h | 0 | 2:20h | 0:04m | 2:20h |
| p456k | 740,660 | 35,396 | 14 | 47:23m | 14 | 41:24m | 6:45m | 48:09m | 14 | 41:53m | 0:50m | 42:43m |
| p462k | 673,465 | 132,249 | 0 | 1:10h | 0 | 1:04h | 2:06m | 1:06h | 0 | 1:04h | 1:37m | 1:06h |
| p565k | 1,025,273 | 28,287 | 0 | 6:25m | 0 | 5:16m | 0:24m | 5:40m | 0 | 5:30m | 0:02m | 5:32m |
| p1330k | 1,510,574 | 44,299 | 0 | 1:00h | 0 | 41:39m | 5:31m | 47:10m | 0 | 41:55m | 1:39m | 43:34m |
| p2787k | 2,395,388 | 651,868 | 15 | 15:15h | 8 | 6:46h | 34:04m | 7:20h | 11 | 6:54h | 20:25m | 7:14h |
| p3327k | 4,557,842 | 109,622 | 914 | 73:46h | 917 | 70:44h | 14:15h | 84:59h | 916 | 72:39h | 12:53m | 72:52h |
| p3852k | 5,507,779 | 164,988 | 849 | 39:01h | 851 | 36:45h | 2:52h | 39:37h | 846 | 37:50h | 7:17m | 37:57h |

untestable faults only. It can be seen that the ITC'99 benchmarks as well as the small industrial circuits contain only few untestable faults. Therefore, it is unlikely that the proposed technique achieves significant speed-ups on these circuits. On the other hand, the industrial benchmark p2787k contains more than 650,000 untestable faults.

Three configuration have been considered. Results of the traditional SAT-based ATPG – as shown in [2] – are given in column *Traditional*. Secondly, this approach was enhanced by performing the preprocessing step, described in Section IV, to all faults considered during ATPG. Results are given in column *Fast all*. Finally, the preprocessing step was applied only to those faults that are estimated to be untestable, i.e. those faults whose predecessor in the fault list is untestable. Result of this approach can be found in column *Fast estimated*.

For each method, the total run time for the ATPG process and the number of aborts are presented in column *Time* and column *Ab.*, respectively. An abort occurs after 10 MiniSat restarts. Moreover, for the methods "Fast all" and "Fast estimated", the total run time is split into the pure classification time (column *Class.*) and the overhead due to preprocessing steps that are unable to classify the fault (column *Prep.*).

Comparing the traditional method with the "Fast all" approach, it can be seen that for the small circuits the overall run time increases only slightly. On most of the large circuits, on the other hand, a speed-up could be achieved. The ATPG process for circuit p2787k was even accelerated by a factor of two. The increased run time for circuits p3327k and p3852k can be explained by the significant overhead for useless preprocessing steps with an amount of almost three hours and more than 14 hours, respectively. Considering the pure classification time, i.e. without regarding this overhead, results in speed-ups for every circuit.

The "Fast estimated" method employs an untestability prediction in order to reduce the overhead mentioned above. Indeed, this overhead could be decreased significantly for all benchmarks, e.g. for circuit p3327k from more than 14 hours to less than 13 minutes. As a result, the overall run times, compared to the "Fast all" approach, can be reduced. In comparison to the traditional method, the impact on circuits with few untestable faults is small. The ATPG process for large circuits, however, could be accelerated considerably. Thus, the robustness regarding overall run time can be increased significantly using the proposed technique.

In Table III the average CNF sizes, i.e. the number of variables (column *Vars*) and the number of clauses (column *Cls*), of all three methods described above are given. In case of the two approaches incorporating the preprocessing technique, the given numbers refer to those SAT instances that are able to classify a particular fault. If a fault is proven to be untestable by the partial SAT instance, generated during the preprocessing step, then the size of this partial CNF is considered; otherwise, the size of the entire CNF is considered. In all approaches, only the clauses added during the circuit to CNF conversion (see Section II-B) are given, i.e. no conflict clauses are

TABLE III
AVERAGE CNF SIZES

| Circuit | Traditional | | Fast all | | Fast estimated | |
|---|---|---|---|---|---|---|
| | Vars | Cls | Vars | Cls | Vars | Cls |
| b14 | 5,421 | 14,163 | 5,379 | 14,046 | 5,398 | 14,100 |
| b15 | 7,207 | 19,238 | 6,877 | 18,345 | 7,047 | 18,810 |
| b17 | 6,385 | 16,576 | 6,108 | 15,828 | 6,212 | 16,109 |
| b18 | 6,153 | 15,713 | 6,011 | 15,336 | 6,050 | 15,439 |
| b20 | 7,532 | 19,836 | 7,487 | 19,713 | 7,507 | 19,769 |
| b21 | 7,675 | 20,256 | 7,616 | 20,094 | 7,641 | 20,163 |
| b22 | 7,383 | 19,426 | 7,344 | 19,319 | 7,368 | 19,383 |
| p44k | 29,819 | 72,767 | 29,401 | 71,632 | 29,497 | 71,891 |
| p77k | 544 | 1,374 | 531 | 1,340 | 531 | 1,340 |
| p80k | 4,311 | 9,929 | 4,310 | 9,926 | 4,311 | 9,929 |
| p88k | 2,352 | 5,536 | 2,318 | 5,442 | 2,331 | 5,475 |
| p99k | 2,580 | 5,933 | 2,563 | 5,883 | 2,570 | 5,900 |
| p141k | 33,566 | 95,887 | 20,554 | 57,159 | 20,660 | 57,449 |
| p177k | 37,402 | 108,504 | 22,063 | 62,006 | 22,186 | 62,345 |
| p456k | 6,724 | 18,617 | 4,861 | 12,749 | 4,991 | 13,114 |
| p462k | 4,408 | 12,677 | 3,762 | 10,569 | 3,770 | 10,595 |
| p565k | 1,688 | 4,331 | 1,026 | 2,538 | 1,113 | 2,769 |
| p1330k | 16,757 | 52,511 | 11,234 | 33,407 | 11,266 | 33,502 |
| p2787k | 16,859 | 56,299 | 5,665 | 18,124 | 5,718 | 18,300 |
| p3327k | 34,428 | 75,032 | 29,196 | 61,426 | 29,347 | 61,812 |
| p3852k | 20,619 | 47,178 | 16,684 | 36,444 | 16,824 | 36,788 |

considered.

It can be seen that using the proposed technique results in smaller CNF sizes. Especially for large circuits with many untestable faults, the savings are significant. Obviously, since the preprocess is applied for each target, the "Fast all" method creates the smallest SAT instances. Nevertheless, the CNFs, built by the "Fast estimated" approach, are only slightly larger.

To summarize, the configuration "Fast all" – where the proposed preprocessing technique is applied to each fault – achieves the best results with respect to CNF size and classification time. However, the overhead for useless preprocessing steps is a drawback. The method "Fast estimate", on the other hand, reduces this overhead thanks to an untestability prediction. Therefore, the entire ATPG process can be accelerated significantly.

## VI. Conclusion and Future Work

The contribution of this paper is a method to accelerate SAT-based test pattern generation for untestable faults in large industrial circuits. This is done by generating only a partial CNF during a preprocess. In order to reduce possible overhead, an untestability prediction is given.

The experimental results confirm that the robustness of SAT-based ATPG can be increased using the new technique. While the impact on small circuits is slight, the overall run time of the ATPG process for large industrial circuits, containing many untestable faults, can be significantly reduced.

It is focus of future work to combine the proposed method with the incremental instance generation scheme. Additionally, more sophisticated prediction heuristics could be developed.

## Acknowledgements

This work was funded in part by DFG grant DR 287/15-1.

Furthermore, the authors would like to thank Stephan Eggersglüß from the University of Bremen, Germany, and René Krenz-Bååth from Mentor Graphics, Hamburg, Germany, for helpful discussions.

## References

[1] F. Corno, M. Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results. In *IEEE Design & Test of Comp.*, pages 44–53, 2000.

[2] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Trans. on CAD*, 27:1329–1333, 2008.

[3] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[4] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. on Comp.*, 32:1137–1144, 1983.

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

[6] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe*, pages 142–149, 2002.

[7] W. Kunz. HANNIBAL: An efficient tool for logic verification based on recursive learning. In *Int'l Conf. on CAD*, pages 538–543, 1993.

[8] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[9] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[10] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

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

[12] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Trans. on CAD*, 7(1):126–137, 1988.

[13] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.

[14] P. Tafertshofer, A. Ganz, and K. Antreich. Igraine - an implication graph based engine for fast implication, justification, and propagation. *IEEE Trans. on CAD*, 19(8):907–927, 2000.

[15] D. Tille and R. Drechsler. Incremental SAT-instance generation for SAT-based ATPG. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 68–73, 2008.