Characterization of Possibly Detected Faults by Accurately Computing their Detection Probability

Jan Burchard‡, Dominik Erb§ and Bernd Becker*

‡ Mentor Graphics, Hamburg, Germany
§ Infineon Technologies AG, Neubiberg, Germany
* University of Freiburg, Freiburg, Germany

***Abstract*—With ever more complex and larger VLSI devices and higher and higher reliability requirements, high quality test with a large fault and defect coverage is becoming even more relevant. At the same time, when unspecified or unknown input values (X values) have to be considered in a pattern, commercial ATPG tools are sometimes not capable of determining whether a fault can be tested – but there is at least a chance to detect the fault, as 0/X or 1/X could be propagated to at least one output. Consequently, these faults are considered to be *possibly detected* and often counted towards the overall fault coverage with a weighting factor. However, as the actual probability to detect these faults with the considered test pattern is not taken into account, this could lead to an over- or underestimation of their real fault coverage, falsifying the test results.**

We introduce a #SAT-based characterization algorithm for this class of faults. This new algorithm is, for the first time, able to accurately compute the detection probability for faults marked as possibly detected by state-of-the-art commercial tools.

Our experimental results for the largest ITC'99 benchmarks as well as larger industrial-circuits show that our algorithm can accurately determine the detection probability for most of the possibly detected faults and also identify faults that are completely untestable or found with a probability of 100 % irrespective of the assignment of the inputs with an X value.

Furthermore, they show that the detection probability is circuit-dependent and consequently should not just be estimated by a simple weighting factor but requires a more in-depth evaluation. Otherwise, there is a high risk that the achieved results could clearly be to optimistic or pessimistic with regard to the real fault coverage.

***Keywords*-Circuit testing, Automatic test pattern generation, Possibly detected, #SAT, unknown values**

I. Introduction

With the ever increasing size and complexity of modern VLSI circuit designs the potential for defects resulting in yield loss or even customer returns is growing. At the same time, there are higher and higher reliability requirements in many fields – e.g., in the automotive sector and in medical technology. The trend in these industries can be described as aiming for 0 DPPM (defective parts per million) shipped, down from the single or low double digit DPPM range that is already achievable in today's world. Since it is basically impossible to create nanometer-scale integrated circuits (IC) without any defects – which could be caused by a single missing or misaligned atom – the quality of the device test has to be improved instead.

For digital circuits, structural testing based on a fault model is essential. Exhaustively testing the complete functionality of a modern circuit is not possible because there are simply too many input combinations and internal states. Instead, structural test is used to prove the absence of the modeled faults in the device by applying test patterns. These patterns are created by ever more advanced automatic test pattern generation (ATPG) tools.

When the test patterns contain unknown or unspecified values (X) – for example because some circuit inputs cannot be controlled, or should not be specified because of compaction or relaxation reasons– it might not be possible to show the testability of some of the modeled faults by the generated patterns. However, if at least one assignment to these circuit inputs (X-sources) allows the detection of the fault, there is at least a chance to detect this fault. Consequently the faults are also not definitely untestable. In state-of-the-arte commercial and academic tools, these faults are marked as *possibly detected* and often partially counted towards the overall percentage of detected faults by a weighting factor [1], [2].

However, without further investigation, it is impossible to tell the probability to detect these faults and consequently to judge the accuracy of the weighting factor. At this point our proposed #SAT-based characterization algorithm comes into play. With it, the possibly detected fault can be put into one of three groups:

1) Definitely detected: no matter how the X values are assigned, the fault is always detected.
2) Definitely not detected: no matter how the X values are assigned, the fault is never detected.
3) Potentially detected: the fault is detected for some assignments of the X values.

While there are other approaches to determine whether a test pattern with X values definitely detects a possibly detected fault (e.g., accurate X-aware fault simulation [3], [4]), the presented approach is the first that accurately calculates the detection probability for faults that are neither always testable nor completely untestable.

Hence, for all faults of group 3, the algorithm **accurately computes the probability** that a potentially detected fault is actually detected. With this knowledge, the weighting factor does not have to be purely estimated anymore but could either by specified more accurately or it is also possible to specify a probability limit – and hence only all possibly detected faults with a higher probability (e.g. over 90 %) are counted to the overall fault coverage. Based on such an more accurate calculation of the achieved fault coverage, the design for testability (DFT) requirements can be more accurately determined (e.g., the insertion of test points), resulting in a lower overhead or better results in direction of 0 DPPM. In detail this paper presents:

- A #SAT-based characterization algorithm for possibly detected stuck-at faults[1].
- An accurate computation of the detection probability for potentially detected faults.
- Three advanced SAT-based optimizations to reduce the formula size and increase the solving speed.
- A thorough evaluation for different scenarios with test patterns from a commercial tool on large benchmarks including industrial circuits.
- An investigation into the scalability of the approach by applying grid-scale computing to the solve process.

Our experimental results for large benchmark and industrial circuits show that the real testability of faults that are considered to be possibly detected by a state-of-the-art commercial tool is highly circuit dependent. Furthermore, if X values originate from unknown values, for example because of uninitialized flip-flops, the detection probability is generally decreased and an accurate characterization of these faults becomes all the more relevant.

The remainder of this paper is structured as follows: The subsequent Section II introduces the basic concepts, e.g., automatic test pattern generation and SAT- and #SAT-solving. These are used for the proposed fault characterization approach described in Section III. In Section IV the characterization is evaluated on a wide range of different circuits. Section V concludes the contribution with a short summary and outlook on future work.

[1]Note: While the presented approach focuses on possibly detected stuck-at faults, it could easily be applied to other faults models as well.

978-3-9819263-0-9/DATE18/©2018 EDAA

II. Preliminaries

A. Automatic Test Pattern Generation

An IC can suffer from a myriad of different manufacturing defects which can influence its behavior and result in a wide range of errors, from all out complete failures to rare or intermittent malfunctions. Since the number of possible defects is almost endless, an abstraction, a so called *fault model*, is used to create a finite set of faults which can be tested. This model is based on faults with a clearly defined behavior which have some correspondence to real world defects. One widely used model is the stuck-at model [5], which assumes that a single gate input or output within the circuit is either always '0' or always '1'. This fault model gives a finite and clearly defined set of potential faults: For every line in the circuit there are only two possible faults (the line is stuck-at-1 or stuck-at-0).

In addition, many other fault models exist and are often used to augment the stuck-at model to cover additional defects. These include the transition-delay [6], transistor stuck-off [7], interconnect open [8] and cell-aware [9] model, to name just a few.

To systematically test an IC after production, certain input patterns (called *test patterns*) are applied to the circuit. These patterns are designed to create a difference on at least one of the outputs if one of the modeled faults is present. For example, an output might have the value '0' in the fault-free circuit (also known as the *good* value) but the value '1' if a certain fault is present (the *bad* value); This is abbreviated as 0/1.

An automatic test pattern generation (ATPG) algorithm creates these test patterns based on the circuit in question and a fault list. A large number of different ATPG approaches have been developed, which range from the basic D-algorithm [10] and its improvements [11], [12] to SAT-based solutions which work on a more abstract mathematical model [13], [14].

For many faults, it is sufficient to specify some of the circuit inputs to generate a valid test. The remaining inputs can be set to both '1' or '0' without changing the test outcome. In the test pattern such inputs are given the value X as they are unspecified. The value X is only used for the modeling of the unspecified inputs. When the test pattern is applied to the actual physical device, every X value is either '0' or '1'.

B. Unknown Values

Basic ATPG algorithms assume that all of the inputs of the circuit under test can be controlled and set to the required values. However, in real circuits this is often not the case: for some inputs it might not be possible to guarantee any value. These inputs therefore have to be considered as *unknown*. Just like an unspecified input, this is again modeled with the value X. Although *unspecified input value* and *unknown input value* are semantically different, in this specific context they mean the same thing: No matter how the X-input is assigned, the test pattern must still detect the fault. There are different reasons for the occurrence of unknown values at the inputs:

- The input is connected to a flip-flop that is not part of a scan-chain and the flip-flop's value cannot be initialized (in a reasonable time) through the combinational logic.
- The input is connected to another (on-chip) module or device that cannot be controlled or is not yet completely specified during the test generation.

Accurately handling X values (especially if there is more than one X-input) within an ATPG algorithm is difficult because reconverging X values might disappear and become '0' or '1' again [15].

Because it is so difficult to work with X-inputs accurately, most ATPGs (including commercial tools) are pessimistic and assume the worst case for X values or do not perform a full analysis of the X-interdependencies.

C. Possibly Detected Faults

Often a single test pattern can detect multiple different faults. In addition, some patterns might *possibly* detect a fault. A fault is possibly detected if there is no output with a definitive good-bad difference but there is at least one output where the good value is well defined (0 or 1) and the bad value is unknown (X). This case might occur because the fault changes the signal propagation within the circuit and connects an X-input to an output.

As an example, consider the circuit in Figure 1. The pattern "01X" definitely detects a stuck-at-1 fault at the first input of G_1 (shown in yellow). In addition, it might also detect a stuck-at-0 fault in input 1 of G_2 (shown in red).

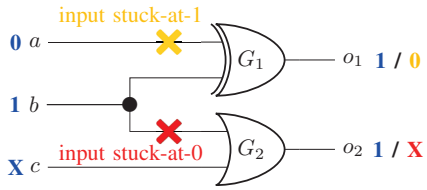

Figure 1. Faults detected by the pattern $01X$ in a circuit. The stuck-at-0 fault is only possibly detected.

When the pattern is applied to a real circuit, input c would be assigned to either '0' or '1'. Hence, with the stuck-at-0 fault, the output o_2 would either have the value 1/0 or 1/1. In the first case, the fault is detected (because the output does not have the required good value), in the latter case it is not detected. Thus, the fault is considered to be possibly detected and has a detection probability of 50 %.

Clearly, the detectability of a possibly detected fault depends on the final test pattern (without X values). However, this pattern might not yet be available during the test generation and might be out of the control of the test generator. When reporting the final fault coverage of the set of generated test patterns, not considering any possibly detected faults would result in a pessimistic estimate of the final coverage since many of these faults might actually be detected. Conversely, assuming that all possibly detected faults are definitely detected would potentially overestimate the real fault coverage. Thus, in commercial tools a user-definable percentage of possibly detected fault (e.g., 50 %) is counted towards the overall fault coverage. However, without further insight into the nature of the possibly detected fault, such estimates are without any guarantees and should be handled with caution. As our experiments have shown, the real percentage is circuit dependent and a more accurate analysis (like the one provided by the presented approach) is therefore, strongly, advised.

D. #SAT-Solving

SAT solvers are used to find a single satisfying assignment to a Boolean formula or to prove that no such assignment exists. If there is more than one satisfying assignment, the solver stops as soon as the first one is found.

Whereas a SAT problem is limited to determining whether a satisfying assignment for a formula exists at all, the #SAT problem ask a more general question:

"How many different assignments satisfy the formula?"

From the number of satisfying assignments it might be possible to derive a probably for the modeled effect to occur. Solving a formula with a SAT solver returns a single example of how a problem can be solved. However, from the number of satisfying solutions – computed by a #SAT solver – the likelihood that the problem is solved can often be derived.

For more details on #SAT-solving, the reader is referred to recent publications in this field [16]–[19].

Generally, solving the #SAT problem is difficult (the problem is #P-complete), even more so than the SAT problem which is *only* NP-complete. Hence, timeouts do occur and some formulas are not solvable even with a large amount of time.

We utilize the #SAT solver *countAntom* [18] which offers a unique soft timeout mechanism. Once the solver reaches the soft timeout, it only continues with the computation when the predicted total solve time is below a second, hard timeout. This way, formulas that are very difficult to solve can be aborted early, after the soft timeout. At the same time, the solver is not too restricted and can still spend a sufficiently large amount of time on some of the formulas until the hard timeout is reached.

III. Characterization of Possibly Detected Faults

In this paper we introduce a #SAT-based approach for the characterization of possibly detected faults. As input, our algorithm takes a possibly detected fault, the set of test patterns that possibly detect the fault as well as the gate level circuit information. This information is readily available from commercial ATPG tools. In this work we focus on possibly detected stuck-at faults, but the same principle can be applied to almost any fault model without any large modifications.

When a fault is possibly detected by different test patterns, these patterns are characterized one after the other. The remainder of this section describes the procedure to analyze a single fault-test pattern tuple.

Recall, that a fault is possibly detected by a test pattern because one of the circuit outputs becomes X when the fault is present. This X at the output exists because the test pattern contains X values. The goal of the presented algorithm is to compute the probability that the fault is detected when all X values are chosen freely. This probability can be anywhere in the range of 0 % to 100 %. Notably, faults with a detection probably of 0 % are definitely not tested by the test pattern – no matter how the X values are chosen – and should therefore, clearly, not be considered for the overall fault coverage. Similarly, faults with a detection probability of 100 % are always tested by the pattern and can be added to the list of covered faults without hesitation.

The basic principle of the approach can be described as follows: First, a Boolean formula is constructed in a way that every satisfying assignment of the formula corresponds to one unique assignment of the X-inputs of the circuit. Then, the number of satisfying assignments of the formula is counted with a #SAT solver. Finally, based on this number the detection probability of the fault is computed.

A. Construction of the Boolean Formula

The Boolean formula is at the heart of the proposed approach. It is constructed by analyzing the combinational core of the circuit and is optimized to be as small as possible. To this end, the first step is the computation of the so called cones of influence of the fault (see Figure 2).

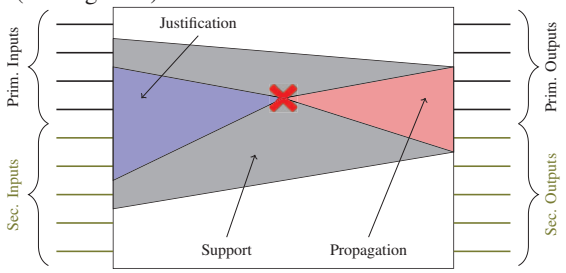

Figure 2. The cones of influence of the fault affected gate (red cross).

Only the gates in the justification- (blue), support- (gray) or propagation-cone (red) of the fault have any influence on the testability of the fault in question. Hence, only these gates are considered.

Next, a copy of the propagation cone is created. This copy is used to model the behavior of the circuit when the fault is present, whereas the original corresponds to the fault-free circuit.

Finally, the fault detection condition is added to the circuit. A fault is detected at an output if, and only if, the value of the output differs between the fault-free and fault affected version of the circuit. This is modeled by adding an XOR-gate between corresponding outputs of the two circuit versions. If the output of the XOR-gate is '1', the output shows a difference.

The final considered circuit consists of the gates in the justification- and support-cone of the fault, two copies of the propagation cone, and the difference-detecting XOR-gates. This circuit is then transformed into a Boolean formula by applying the Tseitin transformation [20] to each gate. This process introduces a new variable for each signal within the circuit, but results in a formula with a size that is only linear in the number of transformed gates. The last step in the formula generation is to ensure that any satisfying assignment to the formula makes the fault effect visible at an output. To this end, the variables $(x_1, \ldots, x_n)$ corresponding to the outputs of the previously introduces XOR-gates are combined into a new clause $(x_1 \vee x_2 \vee \cdots \vee x_n)$ that is added to the formula. Hence, the formula is only satisfiable if at least one of the x_i variables is assigned to $true$. This – in turn – implies that at least one output shows the difference.

To speed up the solve process, it is helpful to augment the Boolean formula with additional information regarding the propagation of the difference between the faulty and fault-free version of the circuit. This difference must propagate from the output of the fault-affected gate to at least one of the circuit outputs. By analyzing the relationships between a difference occurring at the inputs and output of a gate, a *D-chain* can be encoded [13], [21]. Different variants of D-chains encode the chain using implications from the fault site to the circuit outputs ("forward") or from the circuit outputs towards the fault size ("backward") as well as indirectly [22]. For the experiments an indirect hybrid D-chain is used as this D-chain proved to be the most beneficial in a SAT-based stuck-at ATPG [22].

B. Constraining the Inputs

The Boolean formula created in the previous step encodes the propagation of input values through the fault-free and fault affected circuit. Furthermore, it is ensured that the formula is only satisfiable if at least one output differs between these two versions.

Next, the variables of the formula that correspond to the inputs of the circuit have to be restricted to match the considered test pattern. If an input has to be '1' according to the test pattern, the variable is forced to be $true$. And similarly, it is forced to be $false$ for a '0'-input. For inputs with an X value, the variable is not restricted to any value. Hence, the #SAT solver can freely choose an assignment for these inputs.

C. Computing the Probability

The previously generated Boolean formula has very few truly free variables: only the variables corresponding to X-inputs in the test pattern. All other variables – that where introduced during the Tseitin transformation – actually depend on the assignment of these input variables. Nonetheless, the number of possible assignments to the free variables is exponential to their number. Therefore the number of satisfying assignments can not be determined by simply trying all possibilities even for only a few dozen X-inputs.

Hence, the number of satisfying assignments to the formula is instead computed with the help of a #SAT solver. Since all but the input variables are restricted (their value is implied by another variable), the overall number of satisfying assignments is equal to the number of different possible assignments of the X values that make the fault visible at at least one output.

The final step in the characterization of the fault-test pattern pair is the computation of the probability that a freely chosen

assignment of the X values makes the fault effect visible. To this end, the previously computed number of satisfying assignments of the formula is divided by the total number of different possible assignments to the X-inputs:

$$successful\ test\ probability = \frac{\#satisfying\ assignments}{2^{\#X\ inputs}}$$

For the calculation, only the number of X-inputs that are actually part of the input cone of influence of the fault (see Section III-A) must be considered. Only these inputs are actually modeled in the circuit, and therefore only they are represented by free variables in the formula.

D. Optimizations

The previously introduced algorithm contains all the parts that are required to compute the detection probability for a possibly detected fault. In addition, three different improvements were developed. These improvements further increase the solving speed by reducing the formula size and by supplying the solver with additional information about the input assignments. Furthermore, they sometimes allow for a quick and early determination that a fault is always detected or never detected.

1) Propagation Cone Refinement: The propagation cone in Section III-A is computed using structural information: When a gate g is connected to the output of a gate in the cone, then g is in the cone as well. However, it might not be possible to propagate the fault effect to all of the outputs in the propagation cone – some of them might simply never show a difference. Thus, these outputs need not be included into the Boolean formula.

To calculate the set of outputs that are not required, an incremental, SAT-based approach is utilized. First, a Boolean formula just like before is created. This formula is then iteratively extended with the assumption that the fault is definitely visible at output i and solved by the SAT solver. If the formula is unsatisfiable, the fault effect cannot be propagated to output i and output i can be ignored. If the fault effect cannot be propagated to any output at all, it is already determined that the fault is definitely not detected by the test pattern at.

2) Input X Refinement: The considered test pattern usually contains many X values. For some faults, it might be that some of these X-inputs must have a fixed value all the time to successfully test the fault. These inputs can than be forced to that fixed value in the Boolean formula to simplify the computation.

The calculation to determine the fixed X-inputs works similar to the previously discussed propagation cone refinement. Again, the Boolean formula is created as before. Then, the formula is iteratively extended with the assumption that input i is $true$ and then with the assumption that it is $false$. If the formula becomes unsatisfiable for either option, the input must have the inverse value. Should the formula become unsatisfiable for both cases, the fault is not detected at all, again.

3) Always Satisfied Formulas: For some faults, it is irrelevant how the X values in the test pattern is assigned: No matter the assignment, the fault is simply always detected. To avoid having to count all different assignments of the X-inputs, these faults are filtered by the third improvement. To determine if the formula is always satisfied, no matter how the X-inputs are assigned, it is modified slightly: Instead of adding a clause that ensures that *at least one* of the outputs shows a difference between the fault-free and faulty circuit, the inverse condition – that none of the outputs shows a difference – is encoded.

Thus, the generated formula is only satisfiable if there is at least one assignment to the X-inputs that does *not* show a difference. Conversely, if the formula is unsatisfiable, every possible X-assignment will reveal the fault, and its detection probability is 100 %. The detection of faults that are always detected requires only a single SAT-solver call.

IV. Evaluation

The presented accurate characterization algorithm is evaluated in two different scenarios: First we analyze the faults that are only possibly detected by normal stuck-at ATPG patterns (Section IV-A). Secondly, we force some of the circuit's inputs to always be X which simulates a lack of controllability during the test application (Section IV-B). For each experiment, standard stuck-at test patterns are generated with a commercial ATPG tool. This gives a list of test patterns as well as a set of only possibly detected faults. For these faults the ATPG tool did not create a test pattern that definitely detects the fault.

The test pattern list and the set of possibly detected faults is used as input for the proposed characterization algorithm. Internally, we utilize the #SAT solver *countAntom* [18], [19] with a soft timeout of 5 minutes and a hard timeout of 30 minutes. For the input and output refinement optimizations the SAT solver *antom* [23], [24] is used. All experiments are performed on a single core of an Intel Xeon E5-2643 CPU clocked at 3.3 GHz. In addition, at the end of the section we also provide information on how our algorithm performs on a cluster with 256 cores to indicate the scalability of the approach – as it also supports to be run on a cluster.

The experiments were performed on the largest ITC'99 benchmark circuits [25] as well as large industrial circuits. The circuits are assumed to be full-scan. Hence, all of the flip-flops are both controllable and observable by the tester. The circuits are synthesized using the 45 nm version of the NanGate library [26] which contains many complex cells.

A. Default ATPG Patterns

For the first experiment we use a commercial ATPG tool to generate test patterns with its default settings. In Table I the result of the pattern generation is summarized. Here, basic circuit information including the number of inputs (primary plus secondary), the total number of test patterns, number of only possibly detected (PD) faults as well as the average number of test patterns that possibly detected each of these faults is shown.

Table I
THE RESULTS OF THE TEST PATTERN GENERATION.

	Circuit	# Cells	# Inputs	# Test Patterns	# PD Faults	Avg. # Pat. per PD
ITC'99	b15	3 395	487	468	111	2.11
	b17	11 345	1 452	745	566	2.97
	b20	5 844	523	618	134	2.14
	b21	5 899	523	644	139	2.05
	b22	8 144	736	525	83	2.48
Industrial	p78k	2 977	3 151	139	7	6.43
	p100k	25 633	5 565	2 057	157	9.84
	p267k	47 986	15 435	926	210	13.54
	p378k	125 824	15 705	258	268	9.35
	p388k	118 920	20 586	941	1 848	12.66

The possibly detected faults are then characterized by our proposed approach and the results summarized in Figure 3.

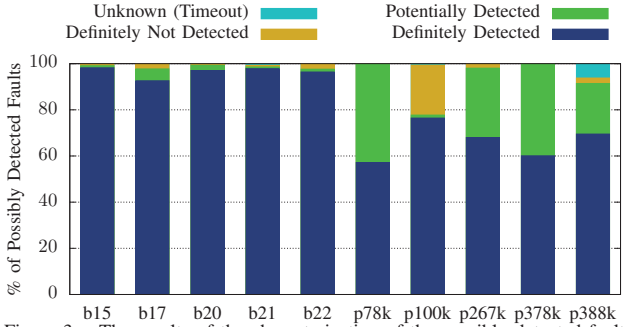

Figure 3. The results of the characterization of the possibly detected faults.

The vertical axis of the graph shows the percentage of possibly detected faults. These are distributed into four groups (from bottom to top): The dark blue bar represents faults that are

definitely detected – which means that there is at least one pattern which has a 100 % detection probability for each of these faults. Faults that are counted towards the green bar are potentially detected. Here, there is no pattern with a 100 % detection probability but at least one pattern has a detection probability of $> 0\,\%$. The yellow bar stands for the faults which are definitely not detected because all of the patterns which could detect these faults have a detection probability of 0 %. Finally, the light blue bar represents fault for which could not be characterized due to timeouts.

Figure 4 shows the average overall detection probability of all the possibly detected faults. If a fault is possibly detected by different test patterns, the pattern with the largest detection probability is used to compute the average across all faults.

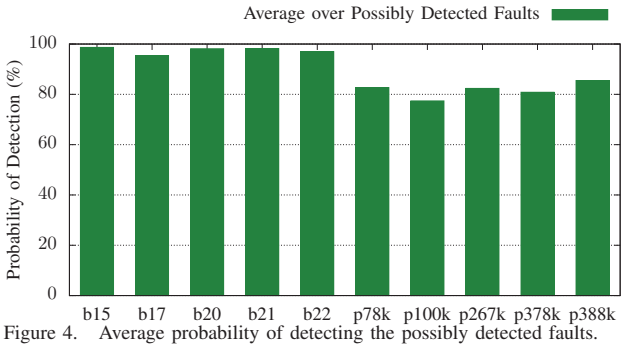

Figure 4. Average probability of detecting the possibly detected faults.

The results show the variability of the probability of detecting a possibly detected fault, depending on the considered circuit. A simple factor-based crediting of possibly detected faults towards the overall fault coverage of the test set seems not advisable given these observations. Without further information, like those provided by our accurate characterization approach, the test engineer cannot decide whether 50 % (standard value of our considered ATPG tool), 77 % ($p100k$) or 99 % ($b21$) of the possibly detected faults should be counted to the fault coverage. Furthermore, when only considering the faults that are potentially detected (without the definitely detected or not detected faults), the average detection probability is only in the range of about 37 % to 50 %.

Here, our method provides the necessary information to compute accurate fault coverage figures. With this information, it can also be decided which faults require additional attention to bring the overall fault coverage to the targeted coverage.

The average time to characterize a possibly detected fault for a test pattern ranges from 25 ms ($b20$) to 33 s ($p388k$) with an average runtime of 6.7 s per fault across all circuits. Our fast characterization approach is supported by the advanced optimizations presented in Section III-D which, amongst others, reduces the number of outputs that are modeled in the Boolean formula by 32 % on average. Furthermore, the number of X-inputs where the X can be assigned freely is reduced by 6 % on average. Of the considered formulas, 79 % are determined to correspond to faults that are always detected.

B. ATPG Patterns with X-Inputs

For the second set of experiments, it is assumed that some of the circuit inputs are always X. These X-inputs could, for example, be flip-flops that are not part of any scan-chain and therefore cannot be controlled by the tester. This is simulated in two experiments, where a subset of inputs (1 % and 2 % of the inputs, chosen at random) is forced to be X. For both experiments, we use a commercial ATPG tool to generated a new set of test patterns. The results are summarized in Table II.

Unlike the previous example, there is a much higher number of possibly detected faults. Furthermore, many of these faults are detected by a large number of different patterns. This creates a very large number of different fault-pattern combinations that

Table II
THE RESULTS OF THE TEST PATTERN GENERATION WHEN SOME OF THE INPUTS ARE FORCED TO X.

	Circuit	1 % X Probability #PD Faults	1 % X Probability Avg. #Pat. per PD	2 % X Probability #PD Faults	2 % X Probability Avg. #Pat. per PD
ITC'99	b15	58	8.24	390	5.94
	b17	116	4.97	427	10.22
	b20	77	25.52	128	20.27
	b21	2 555	11.88	2 403	10.28
	b22	232	13.56	1 940	14.92
Industrial	p78k	460	13.45	986	13.54
	p100k	542	14.23	1 158	15.06
	p267k	1 270	16.81	3 517	22.02
	p378k	3 095	9.07	5 737	9.27
	p388k	5 823	16.09	8 090	21.59

have to be tested one after another. Therefore, the soft timeout for the fault characterization is lowered to 30 seconds with a hard timeout of 180 seconds. Figure 5 visualizes the characterization of the possibly detected faults and Figure 6 the detection probability.

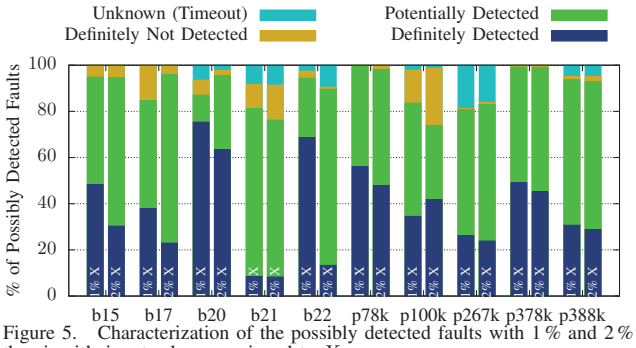

Figure 5. Characterization of the possibly detected faults with 1 % and 2 % of the circuit's inputs always assigned to X.

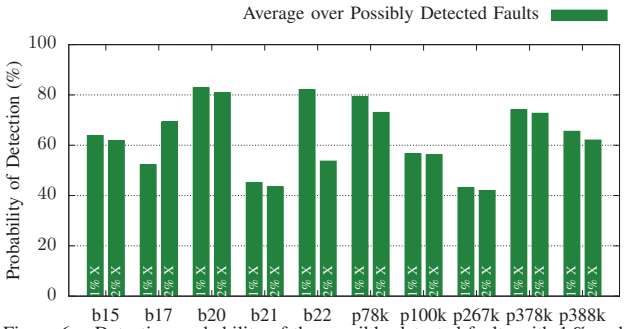

Figure 6. Detection probability of the possibly detected faults with 1 % and 2 % of the circuit's inputs assigned to X.

The lowered timeout length as well as a larger selection of faults with a more varied difficulty slightly increases the number of timeouts. Nonetheless, a large percentage of faults is successfully characterized.

Unlike the first experiments, the number of faults that are definitely detected or definitely not detected is much smaller. This means that even more faults are possibly detected and an accurate computation of the detection probability for these faults becomes even more important. The detection probability of the different faults is much more varied among the circuits and even depending on the percentage of X-inputs. This highlights the necessity of an accurate characterization.

In Figure 7 the distribution of the detection probabilities of the possibly detected faults in circuit $p378k$ with 2 % of inputs assigned to X is shown in greater detail as a histogram. Generally, the detection probabilities of the faults are well distributed. However, detection probability peaks at 50 % and 100 % can be observed. The faults with a 50 % detection probability might be always testable if one of the X-inputs is set to a fixed value. The faults with a detection probability of 100 % are always detected and can be considered as tested without restrictions.

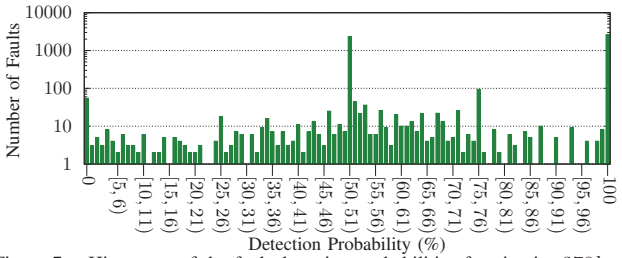

Figure 7. Histogram of the fault detection probabilities for circuit $p378k$ with 2 % of inputs assigned to X.

C. Scaling the Approach

Although the proposed approach is able to characterize a vast majority of faults, even on larger industrial circuits, for some faults timeouts do occur. These faults might have a particularly large input cone of influence and, hence, a huge amount of different possible input combinations that could reveal the fault. Often, even in these cases, the #SAT solver is able to quickly compute the number of valid input assignments – during the experiments the largest number of satisfying assignment was about 10^{2216}. This shows the power of #SAT solvers that can provide accurate results for such formulas. Nonetheless, some timeouts remain. These could either be considered as possibly detected faults, not detected faults, or, if a complete characterization is required, the timeout of the solver could be increased. Furthermore, *countAntom* offers a distributed parallel mode which utilizes more resources but gives faster solving times.

Since it is not the focus of this work (more information on the scalability of #SAT can be found in [19]), we performed only a small experiment with 4 faults which require a long solving time with a single thread. The corresponding Boolean formulas are solved with 256 CPU cores on a cluster. The solve-times are compared in Table III. Clearly, given sufficient resources, even very difficult formulas can be solved in an acceptable time and the detection probability of the corresponding faults can be accurately characterized.

The parallelization of the solver can, of course, also be used to increase the overall characterization time for complex and large circuits.

Table III
COMPARISON OF THE #SAT SOLVER SOLVE TIME WITH 1 AND 256 CORES.

Formula	Solve time (s) *countAntom* 1 core	*dCountAntom* 256 cores
1	1 135.4	10.7
2	1 970.6	16.3
3	843.8	11.0
4	1 701.6	18.3

V. CONCLUSION

In this paper a novel characterization approach for possibly detected faults was introduced. It is both accurate and scalable to larger circuits. Our algorithm utilizes an abstract Boolean model of the circuit and a #SAT solver to compute the number of different assignments to the X values in a test pattern that reveal the considered fault. Based on this number the probability of a successful test is computed. To the best of our knowledge, our algorithm is the first that allows for an accurate computation of the probability to detect possibly detected faults

The evaluation shows that the testability of the possibly detected faults is highly dependent on the circuit and distribution of the X-inputs. For accurate results, the detection probability should not be estimated through a simple weighting factor. With the presented approach it was possibly to accurately compute the detection probability for almost every fault and to generate the information required for an optimal test outcome. Based on such an exact calculation of the achieved fault coverage, the design for testability requirements can be more accurately determined, resulting in a lower overhead or better results in the direction of 0 DPPM. Furthermore, we showed that the presented approach can be scaled to more difficult problems by utilizing the solver in a distributed parallel manner.

In the future we plan on extending our characterizer into a two-step approach which first applies a circuit simulator that is more accurate with regard to X values than that within the commercial tools. This might allow for a faster detection of faults that are always or never detected. Furthermore, extending the analysis to other, more advanced, fault models could yield further insights into the importance of considering possibly detected faults.

Overall, this paper shows that the probability of detecting a possibly detected fault depends on different factors and an accurate characterization should be performed to avoid being overly optimistic or pessimistic.

ACKNOWLEDGMENTS

The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

REFERENCES

[1] *Tessent Shell Reference Manual*, Mentor Graphics Corporation.
[2] *TetraMAX ATPG Quick Reference*, Synopsys.
[3] M. Elm, M. A. Kochte, and H.-J. Wunderlich, "On determining the real output Xs by SAT-based reasoning," in *Proc. IEEE Asian Test Symposium*, 2010, pp. 39–44.
[4] H.-Z. Chou, K.-H. Chang, and S.-Y. Kuo, "Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers," *IEEE Trans. CAD*, vol. 29, no. 4, pp. 646–651, 2010.
[5] J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the diagnosis of switching circuit failures," in *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, Oct 1961, pp. 152–160.
[6] E. R. Hsieh, R. A. Rasmussen, L. J. Vidunas, and W. T. Davis, "Delay test generation," in *Proceedings of the 14th Design Automation Conference*, ser. DAC '77. IEEE Press, 1977, pp. 486–491.
[7] R. L. Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *Bell System Technical Journal*, vol. 57, no. 5, pp. 1449–1474, 1978.
[8] S. M. Reddy, I. Pomeranz, H. Tang, S. Kajihara, and K. Kinoshita, "On testing of interconnect open defects in combinational logic circuits with stems of large fanout," in *Proceedings. International Test Conference*, 2002.
[9] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, "Cell-aware test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1396–1409, Sept 2014.
[10] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM Journal of Research and Development*, vol. 10, no. 4, July 1966.
[11] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215–222, March 1981.
[12] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, Dec 1983.
[13] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design*, pp. 4–15, 1992.
[14] P. Tafertshofer and A. Ganz, "SAT based ATPG using fast justification and propagation in the implication graph," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov 1999, pp. 139–146.
[15] D. Erb, M. A. Kochte, S. Reimer, M. Sauer, H. J. Wunderlich, and B. Becker, "Accurate QBF-based test pattern generation in presence of unknown values," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 2025–2038, Dec 2015.
[16] M. Thurley, "sharpSAT: Counting models with advanced component caching and implicit BCP," in *SAT 2006*, 2006.
[17] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, "Combining component caching and clause learning for effective model counting," in *SAT 2004*, 2004. [Online]. Available: SangetalSAT2004.pdf
[18] J. Burchard, T. Schubert, and B. Becker, "Laissez-faire caching for parallel #SAT solving," in *SAT 2015*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2015.
[19] ——, "Distributed parallel #SAT solving," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2016.
[20] G. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Studies in Constructive Mathematics and Mathematical Logic*, 1968.
[21] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, Sep 1996.
[22] J. Burchard, F. Neubauer, P. Raiola, D. Erb, and B. Becker, "Evaluating the effectiveness of D-chains in SAT-based ATPG," in *2017 18th IEEE Latin American Test Symposium (LATS)*, March 2017.
[23] T. Schubert, M. Lewis, and B. Becker, "Antom – solver description." SAT Race, 2010.
[24] T. Schubert and S. Reimer, "antom," in *https://projects.informatik.uni-freiburg.de/projects/antom*, 2016.
[25] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Des. Test*, vol. 17, no. 3, Jul. 2000.
[26] Si2, "NanGate FreePDK45 generic open cell library, v1.3," http://www.si2.org/openeda.si2.org/projects/nangatelib.