SAT-based Analysis of Sensitisable Paths

Matthias Sauer∗ Alexander Czutro∗ Tobias Schubert∗ Stefan Hillebrecht∗ Ilia Polian† Bernd Becker∗

∗ Albert-Ludwigs-University Freiburg Georges-Köhler-Allee 051 79110 Freiburg, Germany
{sauerm|aczutro|schubert}@atspinne.becker}
@informatik.uni-freiburg.de

† University of Passau Innstrafé 43 94032 Passau, Germany
ilia.polian@uni-passau.de

Abstract—Manufacturing defects in nanoscale technologies have highly complex timing behaviour that is also affected by process variations. While conventional wisdom suggests that it is optimal to detect a delay defect through the longest sensitisable path, non-trivial defect behaviour along with modelling inaccuracies necessitate consideration of paths of well-controlled length during test generation. We present a generic methodology that yields tests through all sensitisable paths of user-specified length. The resulting tests can be employed within the framework of adaptive testing. The methodology is based on encoding the problem as a Boolean-satisfiability (SAT) instance and thereby leverages recent advances in SAT-solving technology.

I. INTRODUCTION

Delay test methods traditionally focus on detecting defects by propagation through paths with as large accumulated delay as possible [1], [2], such as to expose defects that produce very small delays. Furthermore, it is expected that defects with larger delays will be detected through the same path. However, this assumption may reveal itself as overly simplistic in nanoscale technologies for a variety of reasons. Firstly, modelling inaccuracies and process variations [3], [4] lead to significant deviations between longest paths identified by pre-manufacturing analysis and longest paths measured on actual manufactured silicon. Secondly, failure mechanisms such as hold-time violations [5], [6] may require testing through short rather than through long paths. Thirdly, some gate-delay errors have been shown to only be detectable through paths of a well-defined intermediate length, i.e. paths that are neither too long nor too short [7], [8], [9]. One practical response to this challenge is adaptive testing [10]. A comprehensive suite of tests is generated and used in the circuit characterisation or yield-ramp-up phase. Which of the tests are to be employed in actual volume manufacturing test is decided based on their observed effectiveness in detecting defects; this decision may also be made during test application. For example, a test pair $t_1$ that sensitises a shorter path may be more effective than a test pair $t_2$ which sensitises a longer path. One reason for this mismatch could be modelling inaccuracy: while the sum of gate delays along the path sensitised by $t_1$ may be less than the respective number for $t_2$, pair $t_1$ may induce crosstalk or IR-drop, which increases the signal propagation delay along the path. These effects are generally difficult to consider during timing analysis, and they are also affected by process variations. To be applicable for adaptive test, high-quality automatic test pattern generation (ATPG) methods should be able to control the path length and generate a large number of alternative test pairs that sensitise different paths of predefined length.

In this paper, we present a generic approach to this problem: a method based on Boolean-satisfiability (SAT) that yields the complete set of tests that detect a gate-delay fault on a given set of gates by sensitising paths of user-defined lengths. The problem formulation of this paper is more general than in $K$-longest-path ATPG [1]. We consider arbitrary upper and lower limits for path lengths, and we obtain tests for all sensitisable paths of that length, rather than for up to $K$ paths. Note that the number of generated test pairs is intended to be large (typically one test pair for each considered path, even though some pairs may be valid for multiple paths) to provide a variety of tests from which the most effective ones can be selected by adaptive test. In some sense this is similar to $n$-detection [11], [12], [13], but the resulting test pairs are much more targeted towards defect detection and also more diversified, as they are guaranteed to sensitise different paths.

The algorithmic foundation for our method is Boolean satisfiability. A circuit represented by a gate-level netlist, the fault site (a gate output), and the path-length constraints are used to obtain a SAT-instance, i.e. a Boolean formula in Conjunctive Normal Form (CNF) which is then passed to a SAT-solver. A solution produced by the SAT-solver is a test vector for the delay fault at the fault site that sensitises a path of appropriate length. Using a feature of modern SAT-solvers called incremental solving, the solving continues to obtain the next solution, i.e. an alternative test vector which still detects the fault but sensitises a different path of appropriate length. This is repeated until no such paths are left.

The delays of all gates in the circuit are integer numbers between 1 and a user-defined integer constant $\alpha$; hence,
\( \alpha \) controls the efficiency-accuracy trade-off. E.g. \( \alpha = 1 \) corresponds to the unit-delay model (all gates have the same delay of 1) and yields the simplest SAT-instances. Larger \( \alpha \)-values lead to finer granularity but more complex SAT-instances.

Even though the methodology can be applied to arbitrary path-length constraints, the settings we use for evaluation are geared towards detecting small-delay defects. We consider gates on sensitisable paths whose length equals at least 90\% of the length of the longest sensitisable path in the circuit. We consider only gates that are roots of fan-out-free regions, as tests for them are usually valid for residual gates within the fan-out-free regions. For every target gate \( G \), we generate all tests for the delay fault at \( G \) that sensitise paths of length between \( L_G - k \) and \( L_G \), where \( L_G \) is the length of the longest sensitisable path through \( G \) and \( k \) is a user-provided parameter. For example, setting \( k = \alpha \) results in all paths which are up to one maximal gate-delay slower than the longest sensitisable path through \( G \). By doing so, we obtain a high-quality test set which is usable as a starting point for adaptive test. We present detailed results on the numbers and characteristics of paths found and the run times of the method for different parameter values.

The remainder of the paper is structured as follows. An overview of the method is given in Section II. The details on encoding the problem as a SAT-formula are explained in Section III. Various optimisations to make the formula more easily solvable for a state-of-the-art SAT-solver are introduced in Section IV. Experimental results are reported in Section V. Section VI concludes the paper.

II. OVERVIEW OF THE METHOD

The inputs of the presented method are a combinational circuit (gate-level net-list with a fixed integer delay for each gate)\(^2\) and a list of gates to be analysed. For every gate in that list, the method searches for the longest sensitisable paths that pass through that gate. A path is defined as sensitisable if a test pattern pair exists that produces a rising or falling transition at all gate outputs along that path, and if the path starts at a primary input and ends at a primary output. The length of a path is defined as the sum of the delays assigned to the gates along that path. The longest sensitisable path is therefore the path with the maximum delay.

The method consists of two stages. In the first stage, which will be referred to as Longest-Path Search, the length of the longest sensitisable path through a given gate is measured. The second stage called Path Enumeration generates test patterns that sensitise all long paths through the gate. The length of the target paths is derived from the first stage’s result.

\(^2\)Although we concentrated on fixed delays for the current paper, the framework can be easily extended to support more sophisticated delay models with different pin-to-pin delays as well as with different delays for rising and falling transitions.

A. Longest-Path Search

In the first stage called Longest-Path Search, for every target gate \( G \) in the list, the length \( L_G \) of the longest sensitisable path through \( G \) is computed by means of iterative SAT-solving. While identifying structurally longest paths can be efficiently done using graph traversal, many of the structurally longest paths are not sensitisable in practice and cannot be used for test generation [1].

Since it is not possible to express length maximality in traditional SAT, determining the maximum length is done by constructing a series of SAT-instances and solving them. Let \( \lambda_G \in \mathbb{N} \) be the length of the longest structural path through \( G \). Note that \( \lambda_G \) is an input-pattern-independent constant that can be hard-coded into the SAT-instances. For all \( i \in \{0, \ldots, \lambda_G\} \), let \( S(G,i) \) be the SAT-instance that is satisfiable if and only if there is a test pattern pair that sensitises a path through \( G \), of length greater than or equal to \( i \). Then, the first SAT-instance to be solved is \( S(G, \frac{\lambda_G}{2}) \). After solving the first SAT-instance, a second SAT-instance has to be solved. It is either \( S(G, \frac{\lambda_G}{4}) \) or \( S(G, \frac{3\lambda_G}{4}) \) depending on the first SAT-instance’s satisfiability. In the manner of a binary search, the process is repeated until a number \( i' \in \{0, \ldots, \lambda_G\} \) is found such that \( S(G,i') \) is satisfiable while \( S(G,j) \) is unsatisfiable for all \( j > i' \). Then, \( L_G := i' \) represents the length of the longest sensitisable path through \( G \).

B. Path Enumeration

After the application of Longest-Path Search, the tool can apply Path Enumeration either to all gates or to a selection of gates depending on the application\(^3\). Let \( G \) be a target gate and \( L_G \) the length of its longest sensitisable path. Path Enumeration consists in counting all sensitisable paths through \( G \) that have a length between \( L_G - k \) and \( L_G \) for a fixed \( k \in \{0, \ldots, L_G\} \). We call the parameter \( k \) the algorithm’s length range. \( k \) strongly influences run-time as well as accuracy of the approach. However, there is no optimal value for that parameter; the parameter has to be chosen according to the criteria that the found paths are to meet. Note that the method is able to work with any \( k \)-value including \( k = L_G \), in which case all paths through the gate are taken into consideration.

The enumeration of all paths of a fixed length \( l, l \in \{ L_G - k, \ldots, L_G \} \), is done by formulating a SAT-instance that is satisfiable if a sensitisable path of length \( l \) exists. If such a path is identified (i.e. a Boolean solution is found), clauses are added to the SAT-instance such that it remains satisfiable only if a found sensitisable path of length \( l \) differs from the first one. This is repeated until the SAT-instance becomes unsatisfiable, which means that there are no more sensitisable paths of that length. Since the SAT-instance’s core does not change during this process, this stage strongly benefits from incremental SAT-solving.

\(^3\)For example, in a setting aimed at the detection of small-delay faults it is advisable to apply Path Enumeration only to gates with high \( L_G \)-values, i.e. gates that belong to long paths.
The final output of Path Enumeration can be represented by a set of histograms (one per gate) that represents the number of sensitisable paths of each length that pass through a gate. An example is shown in Figure 1. A set of test pattern pairs that sensitise the counted paths is a by-product of Path Enumeration. Since every countered path corresponds to the solution of a SAT-instance, the corresponding sensitising test pattern pair can be extracted from the same Boolean solution at no additional cost.

III. SAT-instances generation

In this section we describe how to construct the SAT-instances needed during Longest-Path Search and Path Enumeration. The SAT-instance-generation problem’s input is the gate-level net-list with a fixed integer delay for every gate, a target gate $g$ and a target length $l$. The aim is to construct a SAT-instance that is satisfiable if there is a sensitisable path through $g$ with length $l$. It is also possible to construct an instance that is satisfiable if the path length is greater than or equal to $l$ or less than or equal to $l$, depending on the current requirement.

The first step consists in assigning several Boolean variables to the lines in the circuit. In order to prevent an explosion of the SAT-instance’s complexity, we tried to keep the number of variables assigned to every line as low as possible. Given a gate $G$, the circuit is partitioned to form three relevant areas: the gate’s input cone (marked $A_1$ in Figure 2), the gate’s output cone (marked $A_2$) and the input cone of the primary outputs in area $A_3$ (marked $A_3$). The rest of the circuit is not modelled at all, as that area is not relevant to the problem since no signal value in this area influences the gate’s output signal or vice versa.

All wires in the areas $A_1$, $A_2$ and $A_3$ are assigned two independent variables that we call $G_w$ and $B_w$ for a wire $w$. These two variables can be interpreted in two different ways depending on what fault model is being used. $G_w$ and $B_w$ may stand for the wire’s value in the fault-free (“good”) and in the faulty (“bad”) circuit respectively. But they can also represent the wire’s value upon application of the first and of the second input vector. We use the latter interpretation. Hence the set of $G$-variables assigned to the primary inputs represents the first input vector while the $B$-variables represent the second input vector; thus, a difference between $G_w$ and $B_w$ represents a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transition at $w$. These variables are used to encode the functionality of each gate depending on its type. See [14], [15], [16] for details on how to produce the required type of clauses.

Every gate output in all three areas is assigned a third variable called $D_w$ for a wire $w$. Additional clauses are added to the SAT-instance such that every Boolean assignment that satisfies the SAT-instance assigns $D_w$ a logic 1 if and only if $G_w$ and $B_w$ differ from each other, i.e. if a transition takes place at $w$. We also implement the D-chain technique [17] with these variables. This technique, which has become standard in SAT-based ATPG [16], [18], adds structural information to the SAT-problem, hence speeding up the SAT-solving noticeably.

Wires in the areas $A_1$ and $A_2$ are assigned two more Boolean variables that allow to specify path validity. A path is defined as valid upon application of an input-vector pair (an input-vector pair is equivalent to a solution of the SAT-problem) if every gate output along the path, starting at a primary input and ending at a primary output, undergoes a transition, and if the target gate $G$ belongs to the path. Analogously, a gate is valid if it belongs to a valid path. Since the specification of validity in SAT is rather simple, and due to space limitations, we refrain from explaining it in detail. We also account for the case that a valid and an invalid path segment re-converge. This is the case if a path splits at a fan-out point in area $A_1$, travels through $A_3$ without passing the target gate $G$ and re-converges at a gate that belongs to $A_2$.

Encoding path lengths and the conditions imposed on them is the most challenging part of the presented method. Since the distance between any primary input and any inner gate changes dynamically depending on which path is being sensitised, an efficient encoding of that distance had to be devised in order to prevent variable and clause explosion that would render the SAT-instance too hard to be solved in acceptable time.

If a gate $g$ is a valid gate, i.e. it is connected to a primary output by at least one valid path segment, we define $g$’s depth to be the maximum distance between $g$ and a valid primary output. The depth of a gate is defined recursively as follows: if the gate’s output pin is a primary output, the distance is 0. If the gate’s output pin is not a primary output, let $g_1, \ldots, g_m$ be $g$’s valid successor gates,
let \( d_1, \ldots, d_m \) be their depths and let \( \delta_1, \ldots, \delta_m \) be their delays. Then \( g \)'s depth is \( \max\{d_1 + \delta_1, \ldots, d_m + \delta_m\} \). Figure 3 shows an example in which all gates have a delay of 1. In contrast, if a gate is not valid, its depth is not relevant and hence ignored. Note that the depth of a gate, as we define it here, is not a fixed number; it changes depending on what paths are sensitised by the currently applied test pattern pair.

Finally, the length of a full valid path is given by the depth of the primary input at which it starts. Note that, since a path can branch into two or more valid sub-paths, the term “path” stands for a set of paths through \( G \) that can be sensitised simultaneously, rather than just for one single path. So as not to exclude path sets composed of paths that can only be sensitised in combination with each other, we opted not to discern single paths from path systems.

Ever potentially valid gate is assigned a number of Boolean variables (called \( T \)-variables) that allows to store its depth using a unary encoding. A natural number \( n \) is represented by any string over \( \{0, 1\} \) that has the form \( b_n \ldots b_0 \), where \( m \geq 0 \) is an arbitrary natural number and \( b_{n+i} = 0 \) for all \( i \geq 1 \), i.e. the index of the leftmost 1 is \( n \). It is also possible to use binary encoding of a gate’s depth, resulting in a logarithmic (instead of linear) number of required variables. We performed experiments with both types of encodings and discovered that SAT-instances using unary encoding are far easier to solve despite the higher number of Boolean variables being assigned to each gate. The unary encoding’s advantage is that it allows for very efficient specification in SAT of the operations that are necessary to compute gate depths and to compare path lengths to constants. For instance, the maximum of two encoded numbers is easily computed by bit-wise disjunction. Analogously, the comparison of a constant number that is hard-coded into the SAT-instance (e.g. the target length \( l \)) and a dynamic number that changes depending on what Boolean solution the SAT-solver finds (e.g. the depth of a gate) is easily performed by comparing the most significant bits.

The only rather complex operation is the comparison of two dynamic numbers (e.g. the depth of two paths). In this case, a full bit-wise comparator is constructed and its functionality is encoded by applying Tseitin-transformation [14], [15]. The comparator’s number of literals is \( O(N^2) \) where \( N \) is the largest encoded number.

### IV. Optimisations

In order to improve the method’s efficiency, we applied a number of optimisation techniques. The first technique, called path look-ahead (PLA), is used during Longest-Path Search. In order to avoid unneeded iterations, whenever a path of known length is found, the minimal depth of all gates belonging to that path is updated. This narrows down the search domain for the binary search and reduces the number of SAT-instances to be generated and solved. Table I compares the running times of Longest-Path Search with and without the application of PLA.

Two other optimisations affect both Longest-Path Search and Path Enumeration. The first one is called dynamic \( T \)-variable resizing (DTVR) and has an effect on how many Boolean variables are used to encode the gate depth. Prior to the generation of a SAT-instance, we first determine by means of static path analysis the maximum and minimum depth the gate may have. Then, instead of encoding the gate’s real depth, we encode the difference between the real depth and the minimum depth, thus reducing the number of \( T \)-variables corresponding to that gate.

The second technique that affects both stages of the algorithm is called dynamic irrelevant-clause deactivation (DIRD) and is aimed at speeding up the SAT-solver’s performance by preventing it from making decisions on \( T \)-variables which become irrelevant when their corresponding gate belongs to no valid path. Clauses that describe operations on \( T \)-variables are added a negated version of the variable that expresses their corresponding gate’s validity. Then, whenever a Boolean assignment makes a gate non-valid, its \( T \)-clauses are satisfied by the new variable and the SAT-solver doesn’t need to make decisions on the \( T \)-variables in those clauses.

Table II compares the total running time of the complete method with and without the application of DTVR and DIRD.

---

**Table I**

Evaluation of optimisation technique PLA: running time (seconds) of Longest-Path Search

<table>
<thead>
<tr>
<th>Circuit &amp; PLA</th>
<th>No PLA</th>
<th>PLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs35932</td>
<td>941.93</td>
<td>560.62</td>
</tr>
<tr>
<td>cs37550</td>
<td>4656.23</td>
<td>1172.24</td>
</tr>
<tr>
<td>cs38417</td>
<td>18171.89</td>
<td>510.71</td>
</tr>
<tr>
<td>cs38584</td>
<td>1063.26</td>
<td>836.72</td>
</tr>
<tr>
<td>cs13207</td>
<td>5983.49</td>
<td>1509.84</td>
</tr>
<tr>
<td>cs15850</td>
<td>1249.31</td>
<td>358.81</td>
</tr>
<tr>
<td>cs35932</td>
<td>5983.49</td>
<td>1509.84</td>
</tr>
<tr>
<td>cs13207</td>
<td>1249.31</td>
<td>358.81</td>
</tr>
</tbody>
</table>

**Table II**

Evaluation of optimisation techniques DTVR and DIRD: total running time (seconds) of Longest-Path Search and Path Enumeration

<table>
<thead>
<tr>
<th>Circuit &amp; DTVR &amp; DIRD</th>
<th>No DTVR &amp; DIRD</th>
<th>Only DTVR</th>
<th>Only DIRD</th>
<th>DTVR &amp; DIRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs35932</td>
<td>941.93</td>
<td>560.62</td>
<td>485.91</td>
<td>510.71</td>
</tr>
<tr>
<td>cs37550</td>
<td>4656.23</td>
<td>1172.24</td>
<td>1172.24</td>
<td>836.72</td>
</tr>
<tr>
<td>cs38417</td>
<td>18171.89</td>
<td>510.71</td>
<td>510.71</td>
<td>1172.24</td>
</tr>
<tr>
<td>cs38584</td>
<td>1063.26</td>
<td>836.72</td>
<td>836.72</td>
<td>510.71</td>
</tr>
<tr>
<td>cs13207</td>
<td>5983.49</td>
<td>1509.84</td>
<td>1509.84</td>
<td>358.81</td>
</tr>
<tr>
<td>cs15850</td>
<td>1249.31</td>
<td>358.81</td>
<td>358.81</td>
<td>358.81</td>
</tr>
</tbody>
</table>

**Figure 3. Gate-depth computation**
Step 1: determine $L_G$ for every gate $G$

Step 2: for every gate $G$:

$L_G \geq \beta L$

yes

numerate paths of length $l_G$, $\ldots$, $L_G$

no
discard $G$

Figure 4

V. Experimental results

The tool was applied to ISCAS 85 benchmark circuits and to the combinational cores (indicated by the prefix cs in the tables) of ISCAS 89 and ITC 99 circuits (indicated by the suffix c in the tables). We omit ISCAS 85 circuit c6288 which is a multiplier known to have a very large number of paths, both sensitisable and not sensitisable, and required special handling in similar approaches such as [1].

In order to get realistic results, timing data provided by the Fraunhofer Institute for Integrated Circuits in Dresden [4] was used to derive integer delays for all gates depending on their types. In order to keep the number of target gates low while still being able to produce significant results, our target lists were composed of the root gates of fan-out-free regions. All measurements were performed on an AMD Opteron computer with 16 cores at 2.3 MHz and 64 GB RAM. Note that every process was allowed to use only one processor core and up to 4 GB RAM.

As SAT-solving back-end we chose a single-threaded version of the in-house SAT-solver antom [19] which supports efficient incremental SAT-solving with and without assumptions. SAT-solvers usually work with a selection of parameters that makes them suitable for SAT-instances of varying complexity degrees. However, after first experimental runs we discovered that the complexity of the SAT-instances we produced is rather homogeneous, and that most problems can be solved in incremental mode in very short times ranging from fractions of a second to a few seconds. Hence, we tuned antom to fit our problem's special requirements. In particular, we disabled some of the more sophisticated methods including Lazy Hyper Binary Resolution which seems to pay off for hard instances while we adjusted the parameters that control restarts such as very short times ranging from fractions of a second to a few seconds. Furthermore, we limited the number of gates in Path Enumeration to 5% of the total number of gates.

The achieved results are listed in Table III. The second to fourth columns list the average length of longest sensitisable paths found in Longest-Path Search, the length of the longest sensitisable path found in Longest-Path Search, and the total needed time in seconds, respectively. Columns 5 and 6 list the total number of found paths (all lengths between $L_G - 5$ and $L$) and the time in seconds required by Path Enumeration. The last two columns list the number of calls to the SAT-solver (i.e. the number of generated SAT-instances) by Longest-Path Search and Path Enumeration, and the total time in seconds needed by both stages.

It can be observed that Longest-Path Search's running time is primarily influenced by the complexity and size of $L$, where $L$ is the maximum of all $L_G$-values measured in Longest-Path Search; we set $\beta = 0.9$ but additionally limited the number of gates in Path Enumeration to 5% of all gates in the target list. The default values of the other parameters are length range $k = 5$, non-robust delay propagation and timing granularity $\alpha = 5$. Moreover, the default tool settings use all three optimisation techniques described in Section IV.
The results are displayed in Table IV. As can be seen, which is not necessarily related to the circuit size. In order to corroborate this empirical conclusion, a second experiment was performed to determine the correlation between the total running time and the length range. The results are displayed in Table IV. As can be seen, the amount by which the number of existing paths grows with rising values of \( k \), varies strongly between circuits. However, the time needed to count those paths increases only linearly with the number of existing paths, regardless of the circuit size.

To the best of our knowledge, this is the first approach that attempts to encode dynamically modifiable path lengths in SAT. Hence, we were not able to compare our tool to other SAT-based approaches. Furthermore, we opted not to compare our tool to existing structural approaches like [1] believing that an objective comparison is not possible as our tool is more versatile and not only counts significantly more paths, but also does on-the-fly generation of test pattern pairs to sensitise those paths.

VI. CONCLUSIONS

In this paper we presented a versatile approach for the efficient enumeration of sensitisable paths based on the use of a flexible SAT-based algorithm. The method’s applicability was demonstrated on various benchmark circuits and varying settings. Furthermore, several optimisation techniques that were developed in order to speed up the base algorithm, were introduced and their benefits were shown by means of experimental results. In contrast to other path-counting approaches, the presented method is able to handle complete path systems and arbitrary conditions imposed on the length parameter, which is very useful e.g. in the area of adaptive testing.

Future work includes the exploration of practical applications. A promising one is the estimation of a gate’s criticality under process variations based on what kind of sensitisable paths go through that gate. In this context, validation of the developed criticality metrics can be done by means of statistical simulation. A further possibility is the extension of the current tool such as to make it able to determine not only the number of existing paths, but also their sensitisation probability. This can be achieved by means of a \#SAT-algorithm, i.e. an algorithm that counts a given SAT-instance’s number of models.

ACKNOWLEDGEMENT

Parts of this work were supported by the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) under grants BE 1176–15/2, PO 1220-2/2, GRK 1103 and SFB/TR 14 AVACS.

REFERENCES


