2015 IEEE 24th Asian Test Symposium

Diagnostic Tests and Diagnosis for Delay Faults using Path Segmentation

Tino Flenker*, André Sülflow* and Görschwin Fey*†
*University of Bremen, Institute of Computer Science, 28359 Bremen, Germany
†Institute of Space Systems, German Aerospace Center, 28359 Bremen, Germany
Email: {flenker, andre.suelflow}@uni-bremen.de, goerschwin.fey@dlr.de

***Abstract*—Diagnosis of integrated circuits is an arduous process. Tools are needed which aid developers locating circuit's faulty parts faster. In this work path delay faults are considered. A simulation based diagnosis algorithm using diagnostic test patterns is introduced for locating the cause of the delay fault. Initial paths are segmented to improve the diagnosis accuracy. For each segment, additional diagnostic test patterns are generated using a solver for Boolean Satisfiability. The experimental results show that a significant improvement of the diagnostic accuracy is achievable with our approach.**

Keywords* — *diagnostic test pattern generation, delay fault diagnosis, segmentation

I. Introduction

Due to continuously shrinking feature size and increasing number of logic gates at the same time, diagnosis is a major concern when developing modern integrated circuits. Testing helps developers to detect a fault, while diagnosis helps locating the causes of failures more quickly and accurately. Delay faults may violate timing constraints and falsify the results produced by a circuit. Several delay fault models have been proposed. Among them, a prevalent fault model is the *Path Delay Fault* (PDF) model [1]. The PDF model captures small as well as large delay defects distributed along a path from the primary inputs to the primary outputs. Another fault model is the *Segment Delay Fault* (SDF) model [2]. The SDF assumes a large delay defect on a subpath of the circuit.

Different faults may mask each other, therefore the definition of robustness is introduced [1][3]. The robustness of tests is classified in two categories: *robust* and *non-robust*. A robust test detects a path delay fault regardless of whether another path delay fault is present. In contrast, a fault effect may be masked when non-robust tests are used. Thus, robust tests are more desirable.

If a circuit test discovers a delay fault, diagnosis starts to identify the cause. The search for the cause of delay requires a large effort which needs a significant proportion in the cycle of IC development. To reduce this effort, automated approaches for diagnosis have been introduced like in [4][5].

For our approach techniques for *Automatic Test Pattern Generation* (ATPG) are used to generate diagnostic *Test Patterns* (TP) for activating paths. Then, faulty behavior can be checked on the activated paths during diagnosis. In the literature several ATPG-techniques are known [6][7][8][9][10]. The authors of [7] generate diagnostic TPs using a genetic algorithm. They use a simulation based ATPG with a directed search mechanism. In addition they introduce a fitness function based on the distinguishable test pairs. In contrast to [7] we generate diagnostic TPs with direct search using *Boolean Satisfiability* (SAT). In [9], SAT-based ATPG for path delay faults is proposed. The authors generate non-robust and robust tests. They use multiple-valued logic in order to handle tri-state elements and environmental constraints occurring in industrial practice. Like in [9] we use multiple valued logic for the generation of diagnostic TPs and for post processing.

Diagnosis for path delay faults is also known in the literature [4][11][12][13]. The authors of [11] use a framework for PDF diagnosis based on effect-cause analysis and enhanced *Zero-suppressed Binary Decision Diagrams* (ZBDDs). In effect-cause analysis the results of the applied tests are examined to obtain all possible failing tests. For each failing test a structural representation is used to describe all possible faults, that could be the cause of the observed error. In [4] a method is proposed to locate segments that cause delays on circuit paths. They use delay bounds of tested paths to build linear constraints. Then a solver for linear programming solves the constraints to identify segments that cause extra delays. In contrast to [4], we use a simulation based approach and identify suspects by taking the intersection of suspects from all failing tests into account.

We propose a simulation based diagnosis algorithm for locating faulty gates using tests for path delay faults. We refine the initial diagnosis by segmenting the given paths. For that, we introduce three methods for segmentation. For all segments additional TPs are generated by a SAT-solver. The experimental results show that at the costs of additional TPs an improvement of the diagnosis accuracy in most cases by a factor between 2 and 4 but the best result yields a factor up to 36.

Our core contributions are a method for generating diagnostic TPs by path segmentation and an algorithm for diagnosis using the TPs.

This paper is structured as follows. In Section II preliminaries are introduced. An overview of our approach is given in Section III. The process for the segmentation of the paths is presented in Section IV, whereas the diagnosis algorithm is treated in Section V. Experimental results are shown in Section VI and the conclusions are drawn in Section VII.

II. Preliminaries

A combinatorial circuit c is represented by a directed acyclic graph $G = (V, E)$, referred to as the circuit graph, where V is the set of circuit nodes and E the set of edges, which corresponds to the connections between the gates in the circuit. The *successors* of a node $g \in V$ are given by a set of nodes $succ(g) = \{h | (g,h) \in E \wedge h \in V\}$. The function $outdegree(g) = |succ(g)|$ gets the number of successors of g. The *predecessors* of a node $g \in V$ are given by a set of nodes $pred(g) = \{h | (h,g) \in E \wedge h \in V\}$. The function $indegree(g) = |pred(g)|$ gets the number of predecessors of

This work was supported by the University of Bremens Graduate School SyDe, funded by the German Excellence Initiative, and the German Research Foundation (DFG, grant no. FE 797/6-2).

978-1-4673-9739-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ATS.2015.32

TABLE I. OFF–PATH CONSTRAINTS

	↑ robust	↓ robust	non-robust
AND/NAND	X1	S1	X1
OR/NOR	S0	X0	X0

g. A *path* P from node g_1 to node g_n is a sequence of nodes $(g_1, g_2, \ldots, g_n)$ with $(g_i, g_{i+1}) \in E$. Each sequential circuit is transformed to a combinatorial circuit for enhanced scan. Flip-flop inputs are treated like a primary output and the Flip-flop outputs are treated like a primary input by introducing *Pseudo Primary Input*s (PPI) and *Pseudo Primary Output*s (PPO). The function *IsPrimIn*(g) returns 1, if g is a (pseudo) primary input. Otherwise the function returns 0. If the gate g is a (pseudo) primary output, the function *IsPrimOut*(g) returns 1 and otherwise 0.

Each gate is considered over two time frames to detect transitions of a signal's value. If the value changes from 0 to 1, then we have a rising edge (↑) and a falling edge (↓) is present if the value changes from 1 to 0.

The difference of robust and non-robust tests for PDFs are the sensitization criteria of the off-path inputs. Table I shows how the logic values for both time frames are set for the robust and non-robust case [1][3]. The logic values *S0/S1* mean, that over both time frames the same logic value is set statically. The *X* which occurs in the first time frame means *don't care*. The polarity at this time frame can be either 0 or 1. At the second time frame the polarity is set to the non-controlling value of the given gate. For an *AND/NAND* gate the polarity is set to 1 and for an *OR/NOR* gate to 0.

It is assumed, that only a single gate in the circuit causes the delay fault.

III. OVERVIEW

The idea of our approach is to segment a set of initial paths and generate TPs which are used for the diagnosis based on simulation. We assume the TPs are generated once before the chip is physically available. After test generation, the tests are stored in a dictionary (through step (4)), which is used in diagnosis (from step (5)).

An overview of our methodology is presented in Fig. 1. First (1), initial paths are analyzed for a given circuit. Robust and non-robust TPs are generated. In the next step (2), paths are segmented to narrow the diagnosis along a failing initial path. For a better refinement the segments need to be sensitized by a test pattern. TP generation for the segments is done in step (3). For the segments, robust and non-robust TPs are generated. For each segment a distinguishing path is searched in step (4). The distinguishing path leads from the primary inputs over a segment to the primary outputs and helps to distinguish from the initial path. The distinguishing path shares only the gates of the segment with the initial path. Step (5) is the actual diagnosis. Here suspects are determined, using the results of the circuit test, executed with previously generated TPs (6).

IV. SEGMENTATION

This section describes the segmentation. Section IV-A presents three segmentation methods in detail. Section IV-B introduces how the segment sensitization and the search for the distinguishing paths is ensured.

A. Segment Generation

For segmentation a set of initial paths is given on which the segmentation is executed, where each initial path leads from

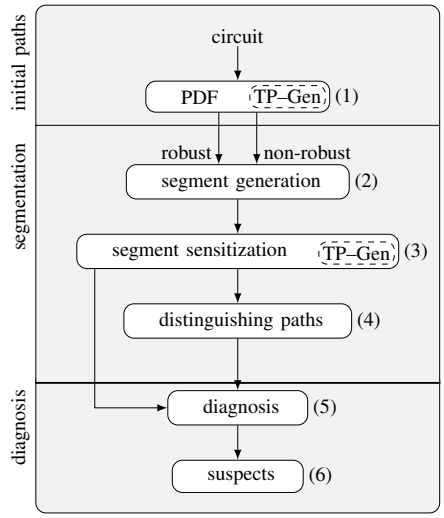

Fig. 1. Overview of methodology

PI to PO:

$$P = \{(g_0, \ldots, g_n) \mid \text{path}(g_0, g_n) \wedge \text{isPrimIn}(g_0) \wedge \text{isPrimOut}(g_n)\} \quad (1)$$

1) Non-overlapping Segments: Fig. 2 demonstrates the non-overlapping method of segmentation. The method does not accept overlapping segments. Thus, a TP can exactly be allocated to the gates of the segment. Each path is split at each fan-in/out.

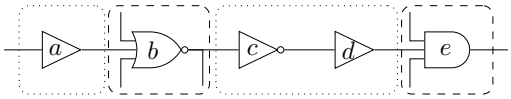

Fig. 2. Non-overlapping segments

The properties of a segment of the non-overlapping method of segmentation are defined in Equation 2. Inside a segment $(g_i, \ldots, g_n)$ the gates have only one predecessor or successor. That is ensured because *indegree* and *outdegree* inside the segment is one. In addition, there must be a path between g_i and g_n. A segment starts at a gate which has an indegree greater than one, or the predecessor has an outdegree greater than one, or the first gate of the segment is a primary input. The last gate has more than one successor, or the successor has more than one predecessor, or the last gate of the segment is a primary output of the circuit.

$$\begin{aligned} S_1 = \{(g_i, \ldots, g_n) \mid \forall_{j=i+1}^{n}.(&\text{indegree}(g_j) = 1) \\ &\wedge \forall_{j=i}^{n-1}.(\text{outdegree}(g_j) = 1) \wedge \text{path}(g_i, g_n) \\ &\wedge (\text{indegree}(g_i) > 1 \vee \text{outdegree}(\text{pred}(g_i)) > 1 \\ &\quad \vee \text{isPrimIn}(g_i)) \\ &\wedge (\text{outdegree}(g_n) > 1 \vee \text{indegree}(\text{succ}(g_n)) > 1 \\ &\quad \vee \text{isPrimOut}(g_n))\} \end{aligned} \quad (2)$$

2) Overlapping Segments: More predecessors or successors for a segment increase the possibilities to find a distinguishing path. The more distinguishable paths are found, the

more precise is the diagnosis. For this reason the overlapping method in Fig. 3, is introduced. Each segment has at least two inputs or outputs at the beginning or ending gate. Exceptions are segments that start or end at the primary inputs or primary outputs.

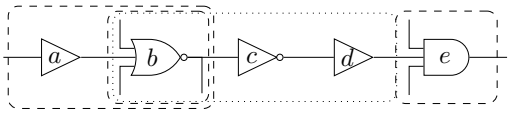

Fig. 3. Overlapping segments

Equation 3 shows the definition of the overlapping method of segmentation. This method generates a segment for each gate. Starting from the given gate g_k in each direction, a path is determined. The gates from g_{i+1} to g_k have an indegree of one. The gates from g_k to g_{n-1} have an outdegree of one. In addition, the first gate of the segment g_i has more predecessors than one or g_i is a primary input. The last gate of the segment g_n has more than one successor gate or is a primary output.

$$\begin{aligned} S_2 = \{(g_i, \ldots, g_k, \ldots, g_n) \mid \\ \text{path}(g_i, g_k) \wedge \forall_{j=i+1}^{k}.(\text{indegree}(g_j) = 1) \\ \wedge \text{path}(g_k, g_n) \wedge \forall_{j=i}^{k-1}.(\text{outdegree}(g_j) = 1) \\ \wedge (\text{indegree}(g_i) > 1 \vee \text{isPrimIn}(g_i)) \\ \wedge (\text{outdegree}(g_n) > 1 \vee \text{isPrimOut}(g_n))\} \end{aligned} \quad (3)$$

3) Overlapping Segments with start/end on PI/PO: The next overlapping method of segmentation is shown in Fig. 4. Starting from the segment, a search for the distinguishing path in one direction is needed only.

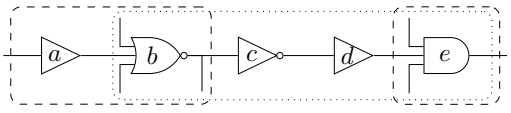

Fig. 4. Overlapping segments with PIs/POs as start/end

The definition of the overlapping method of segmentation with start/end on the PI/PO is shown in Equation 4. The segments $(g_i, \ldots, g_n)$ either start at the primary inputs and end at a succeeding gate with more than one successor or the segments end on a primary output and starting at a preceding gate with more than one predecessor.

$$\begin{aligned} S_3 = \{(g_i, \ldots, g_n) \mid \text{path}(g_i, g_n) \\ \wedge ((\text{isPrimIn}(g_i) \wedge \text{outdegree}(g_n) > 1) \\ \vee (\text{isPrimOut}(g_n) \wedge \text{indegree}(g_i) > 1))\} \end{aligned} \quad (4)$$

In summary, the three segmentations are briefly characterized as follows:

1) A TP is exactly assignable to the gates of a segment
2) Increased possibility to find distinguishing paths
3) Searching a distinguishing path in one direction is needed only

B. Segment Sensitization

This section describes the procedure to compute the distinguishing path for segments generated in Section IV-A. Our idea is to generate tests for sensitizing the segment, such that a delay is propagated through the segment but not through any other part of the initial path. TPs for the segments are generated by a SAT-solver.

For further explanation the example circuit shown in Fig. 5 is used. Gate g is assumed as the segment and the initial path propagates over the path f_{pdf}.

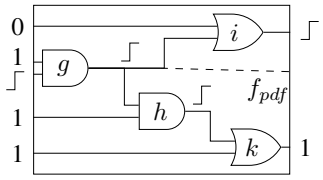

Fig. 5. Search for distinguishing paths

First, the transitive fan-in cone of the initial path will be transformed to *Conjunctive Normal Form* (CNF) [14]. Next, the side inputs of the segment are set to non-controlling values accordingly if the given gate is an AND/NAND or an OR/NOR gate. Next, new constraints ensure, that a distinguishing path for the segment will be returned by the SAT-solver, if exists. These constraints do not explicitly fix a path, but leave the search to the SAT-solver.

In [15] the author wants to propagate the from the D-Algorithm [16] known D-values to the primary outputs with SAT. In [17] for each gate in the circuit another variable and constraint is added. If the variable is set to 1, the constraint ensures that one of the successors propagates the D-value. Our idea is similar, however edges are propagated. We introduce such additional variables only for the gates of the segment's output cone (Equation 5) and input cone (Equation 6).

First, for each wire not on the initial path f_{pdf}, a new variable c_{gh} is added. The indices correspond to the source gate g and the target gate h of the wire. Next, variables for the first and the last gate of the segment c_g are added and set to 1. That activates the added implications represented by Equation 5 and 6. In Fig. 5 only gate g is the segment, hence, only c_g is added and set to 1. Equation 5 for gate g is constructed as follows: $c_g \rightarrow c_{gh} \vee c_{gi}$

$$c_g \rightarrow \bigvee_{x \in succ\,(g)} c_{gx} \quad (5)$$

$$c_g \rightarrow \bigvee_{x \in pred\,(g)} c_{xg} \quad (6)$$

If c_{gh} is set to 1, the edge is propagated along the wire from g to h. The following constraints are added to ensure this behavior.

$$c_{gh} \rightarrow \bigwedge_{x \in pred(h) \setminus g} (x = X1/X0) \quad (7)$$

In Equation 7 the wire c_{gh} between the gates g and h out of Fig. 5 is treated. All side inputs x of gate h are set to the non-controlling value $X1$.

If robust tests are desired, Equation 7 is extended. The side inputs need specific values for hazard-free forwarding of the edge through the path.

For each gate between the segment, in our example gate g, and the primary outputs, the extended constraints are shown

in Equation 8 and 9. Equation 8 is only added, if the given gate has 1 as the non-controlling value. Otherwise Equation 9 is added instead. For all succeeding gates of g holds if the implication variable c_{gx} is set to 1, g is assigned to a rising edge and all side inputs except g are set to $X1$ or g is assigned to a falling edge and the side inputs unequal to g are set to $S1$.

$$x \in succ(g)$$
$$c_{gx} \rightarrow ((g =\uparrow \wedge \bigwedge_{y \in pred(x) \setminus g} y = X1) \vee (g =\downarrow \wedge \bigwedge_{y \in pred(x) \setminus g} y = S1)) \quad (8)$$
$$c_{gx} \rightarrow ((g =\uparrow \wedge \bigwedge_{y \in pred(x) \setminus g} y = S0) \vee (g =\downarrow \wedge \bigwedge_{y \in pred(x) \setminus g} y = X0)) \quad (9)$$

Equation 10 and 11 show the extended functions for the gates between the segment and the primary inputs. The difference to Equation 8 and 9 is that the inputs of gate g are assigned to the rising or falling edge and the side inputs except gate x are set to $X1/S0$ or $S1/X0$.

$$x \in pred(g)$$
$$c_{xg} \rightarrow ((g =\uparrow \wedge \bigwedge_{y \in pred(g) \setminus x} y = X1) \vee (g =\downarrow \wedge \bigwedge_{y \in pred(g) \setminus x} y = S1)) \quad (10)$$
$$c_{xg} \rightarrow ((g =\uparrow \wedge \bigwedge_{y \in pred(g) \setminus x} y = S0) \vee (g =\downarrow \wedge \bigwedge_{y \in pred(g) \setminus x} y = X0)) \quad (11)$$

Finally, the SAT-solver tries to solve the given instance. If the result is satisfiable, a distinguishing path for the given segment with a TP exists.

The distinguishing path is not explicitly given by the SAT-solution. Therefore the distinguishing path needs to be determined. In the SAT-solution for each TP a depth first search for edge propagating gates is executed from the segment to the primary inputs and primary outputs to find the distinguishing path.

In Fig. 5 the search starts from gate g following the edge forwarding gates until reaching the primary outputs excluding the gates of the initial path f_{pdf}. If the search proceeds along gate h, no solution can be found because the delayed values are masked at gate k. Afterwards, the search continues and finds a distinguishing path along gate i to a primary output. Therefore, the distinguishing path for the initial path f_{pdf} is (g, i). To obtain the TP for the segment, the logic values of the primary inputs are taken from the SAT-solution.

V. Diagnosis

The algorithm for diagnosis is described in the following section. For that, a circuit test with all generated TPs for the initial paths and the distinguishing paths is assumed to be executed before. The output values resulting from the circuit test are stored in a map *tr* for diagnosis.

The general idea is to take the intersection of suspects of all failed TPs. To take the intersection is valid, because a single delay fault is assumed. From the intersection all guaranteed fault-free gates of passing TPs are removed. All remaining gates are the final set of suspects.

Now, at first two auxiliary functions are described in Algorithm 1 and 2. Subsequently, the main function for calculating the suspects is presented in Algorithm 3.

Even for robust tests Algorithm 1 computes the *activation cone* for a given *path* and TP *tp*. An activation cone only includes gates, which could be the cause of error by a given TP. The included gates are able to propagate an edge to the given path's primary output. The *cone*s are required later in the diagnosis in order to distinguish the possibly faulty gates from the fault-free ones. Line 2 simulates the TP to determine expected values of gates. To get the input cone of the path's output a breadth first search starting at the primary output of the path is carried out. For that, a queue is initialized with the output gate of the *pdf* at Line 4. Then the activation cone of the path's output is calculated. For each gate in the queue the predecessor gates ($predecessor$) are examined (Line 7). First the gate is checked, whether it is already visited. If the examined gate is already visited, the next gate is treated. Otherwise the gate is set to visited. In addition the polarity of the current gate is checked (Line 11). Gates with an edge (non-static value) forward the delay and are pushed to the queue, so that the predecessors of this gate will be examined later. Furthermore the examined gate is included in the cone. Finally, the activation cone is returned.

Algorithm 1 Calculation of activation cone for $path$ and tp

```
1:  function active_cone(circ, path, tp)
2:      sim ← simulate(circ, tp)
3:      cone ← {path.po()}
4:      queue.push(path.po())
5:      while ¬queue.empty() do
6:          current ← queue.pop()
7:          for all predecessor ∈ pred(current) do
8:              if predecessor.is_visited() then
9:                  continue
10:             predecessor.visit()
11:             if sim.polarity_of(predecessor) ∉ {S0, S1} then
12:                 queue.push(predecessor)
13:                 cone ← cone ∪ predecessor
14:     return cone
```

Algorithm 2 analyzes the suspects for a given path with TP tp. On Line 2 the activation cone for the path is calculated first. Afterwards, it is checked, whether the results of the circuit test (tr) have discovered a delay fault on the path's primary output. In case a delay is present the intersection of the activation cone and the suspects is computed. Otherwise, no error is detected and the gates of the activation cone are removed from the set of suspects (Line 7). Finally, the function returns, whether a delay is present on the path with the given TP or not.

Algorithm 3 computes the suspects sps for a given circuit. First, to take an intersection of two sets of suspects, an initial set of suspects is needed. For that purpose a failing initial path's test instance at Line 2 is randomly selected. Next, the active cone is taken as initial set of suspects. Then, for all given initial path's tests instances $pdfts$ the suspects analysis is carried out. On Line 7 the suspects for an initial path are

Algorithm 2 Analyze suspects for path

```
1: function analyse_path(circ, path, tp, suspects, tr)
2:     cone ← active_cone(circ, path, tp)
3:     delayed ← tr.value_of(tp, path.po()) ≠ path.po_exp()
4:     if delayed then
5:         suspects ← suspects ∩ cone
6:     else
7:         suspects ← suspects\cone                ▷ fault–free
8:     return delayed
```

computed and returned. If a delay fault is non-existent, the analysis for the next initial path is continued. If a fault is detected (Line 9), the analysis continues for each segment represented by the tested initial path. After checking all initial paths and segments, the remaining suspects are returned, where the faulty gate must be within.

Algorithm 3 Calculation of suspects

```
1: function compute_suspects(circ, pdfts, tr)
2:     pdft_i ← sel_delayed_rnd(pdfts)      ▷ initialize suspects
3:     sps ← active_cone(circ, pdft_i.path(), pdft_i.tp())
4:     for all pdft ∈ pdfts do
5:         ▷ step 1: pdf analysis
6:         delayed
7:             ← analyse_path(circ, pdft.path(), pdft.tp(), sps, tr)
8:         ▷ step 2: segment analysis
9:         if delayed then
10:            for all s ∈ segments(pdft.path()) do
11:                analyse_path(circ, s.dist_path(), s.tp(), sps, tr)
12:    return sps
```

VI. Results

Circuits from the ISCAS'85, ISCAS'89 and ITC'99 benchmark packages are used for the evaluation. The computations are executed on an *AMD Opteron™Processor 2222 SE* with 64 GB RAM and the runtime is measured in CPU seconds.

A. Evaluation of methods for segmentation

Table II shows the results for robust TPs and non-robust TPs. Circuits with more than 2,400 gates are considered only and the initial paths are generated randomly.

The first two columns show the name of the examined circuit and the number of gates. The third column (#detect) presents the number of detectable gates by the generated TPs for the initial paths. A gate is called detectable, if a test exists for an initial path, that propagates an edge along the given gate to an initial path's primary output.

In the first series of experiments, we focus on diagnosis without segmentation given in "*w/o segmentation*". Column four ($\%ac$) shows the achieved gate coverage of the initial paths. A gate is called covered, if a path through the gate with a TP exists, so that an edge is propagated through the gate to any primary output. The initial set of paths achieves a coverage between 17.3% to 54.9%. The number of TPs ($|tp|$) is always less-or-equal to the number of paths ($|p|$), because some TPs are re-usable for different paths.

The average number of suspects returned by the diagnosis is shown in column eight (sps). The average diagnosis accuracy is computed by iteratively assuming each single gate as defect and carrying out the diagnosis algorithm from Section V per gate. Finally, the average is taken over all the results. In some cases the basic diagnosis achieves a quite good diagnosis accuracy. For example, for *b22*, a circuit with $15,690$ detectable gates, an average diagnosis accuracy of 4.3 is achieved. That means, in the average case the algorithm returns for *b22* a set of suspects with 4.3 gates, including the erroneous gate in all cases.

To improve the total runtime, first all activation cones are computed for each pair of TP and path. Afterwards, the diagnosis is applied for each gate by assuming the gate as faulty. The runtime ($t(s)$) includes the computation for all active cones and the sum for the diagnosis for each gate. The runtime for the computation of the cones increases linear with the number of paths whereby the time of simulations increases linear with the number of test instances. A test instance consists of a path or segment and a corresponding TP. In case of circuit *b22* the consumed runtimes are $1,167$s for the calculation of all cones and $2,329$s for the diagnosis for all detectable gates. In sum, we obtain a runtime of $3,496$s.

For some circuits the achieved diagnosis accuracy is still unsatisfactory, so that manual diagnosis would be an arduous work. For example, the circuit *c6288* achieves a diagnosis accuracy of 37.9 with robust tests and 55.2 with non-robust tests only. To improve these results three methods for segmentation were introduced.

The remaining columns are related to diagnosis with non-overlapping segmentation ($s1$), overlapping segmentation ($s2$) and segmentation with overlapping segments starting/ending at the primary inputs/outputs ($s3$). The relative coverage ($\%rc$), is the percentage of gates covered by a segment relating to the covered gates of the initial paths ($\%ac$). The number of generated segments ($|seg|$), the additionally generated test patterns ($|+tp|$), the consumed time for the diagnosis ($t(s)$) for the segments and the achieved average number of suspects (sps) returned by the diagnosis with segments are also shown.

Table II shows a significant improvement of the diagnosis accuracy from diagnosis without segmentation to diagnosis with segmentation. The average improvement achieved is the factor of 5.2 in terms of the number of suspects. The runtime for the diagnosis with segments increases by the factors 3.7 for the non-overlapping segments, 8.8 for the method with overlapping segments and 16.7 for the last segmentation, respectively.

The overlapping segmentation with start/end at the primary inputs/outputs always achieves the best relative gate coverage. Segmentation $s3$ also generates most testable segments in comparison to the non-overlapping $s1$ and the overlapping $s2$ segmentation ($|seg|$). Segmentation $s3$ also produces most TPs and needs the longest runtime but also leads to better diagnosis accuracy. Thus, a trade-off between diagnosis accuracy and runtime has to be made.

These results show the efficiency of our diagnosis approach based on distinguishing paths. Reducing the large number of TPs is out of the scope of this work and will be addressed in future work.

B. Increase the number of initial paths

For circuits with a limited diagnosis accuracy despite of segmentation like $c5315$ the number of initial paths can be increased. More initial paths improve the diagnosis accuracy, because more tests and segments will be generated. Moreover, additional tests increase the number of detectable gates and the runtime for diagnosis.

TABLE II. EXPERIMENTAL RESULTS

robust																							
circ	#gates	#detect	w/o segmentation					w/ segmentation															
								s1					s2					s3					
			$\%ac$	$\|p\|$	$\|tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$
b14	10,042	4,642	33.6	704	704	375	**4.6**	21.9	3,654	1,984	1,586	**3.2**	71.4	11,174	5,283	4,539	**1.5**	99.9	20,379	9,609	10,095	***1.2***
b15	8,852	4,521	37.1	714	714	340	**5.2**	27.8	4,881	3,002	1,699	**4.2**	69.4	10,022	6,371	4,046	**3.5**	99.8	12,414	9,235	6,919	***2.5***
b17	32,229	14,451	35.4	2,438	2,434	3,304	**3.5**	27.5	19,168	11,370	16,726	**2.1**	78.6	39,688	25,313	41,650	**1.4**	99.5	47,135	36,029	71,372	***1.2***
b20	20,204	10,459	36.9	1,405	1,405	1,785	**5.4**	22.1	9,030	5,578	7,059	**4.2**	79.4	23,464	14,611	18,018	**2.9**	100.0	38,161	24,845	35,478	***1.2***
b21	20,549	10,236	36.4	1,442	1,442	1,644	**5.1**	21.1	8,772	5,059	6,619	**3.8**	75.5	22,650	13,452	17,060	**2.4**	100.0	39,709	24,280	34,856	***1.2***
b22	29,929	15,690	39.8	2,087	2,087	3,497	**4.3**	21.4	14,508	9,016	14,618	**3.0**	83.5	37,793	24,746	38,065	**1.6**	99.9	57,628	40,478	74,187	***1.2***
c5315	2,485	724	27.3	141	141	5	**2.9**	14.2	717	290	34	**2.2**	36.7	1,532	558	89	**1.8**	100.0	2,723	1,796	195	***1.4***
c6288	2,448	2,383	54.9	78	78	110	**37.9**	29.7	1,163	657	491	**1.5**	34.0	1,226	697	883	**1.5**	86.3	1,710	1,243	1,368	***1.3***
c7552	3,718	1,301	23.2	180	180	31	**10.5**	15.8	613	308	98	**6.8**	37.8	1,216	561	202	**4.6**	98.4	2,720	1,629	395	***2.4***
s13207	8,601	1,518	17.2	378	369	50	**5.5**	32.0	1,213	417	228	**4.4**	67.3	2,791	1,080	589	**4.1**	100.0	4,022	2,306	1,084	***4.0***
s15850	10,372	3,366	28.4	402	400	110	**7.9**	32.1	1,383	724	411	**5.2**	76.9	4,303	2,222	1,126	**4.2**	99.9	5,009	3,449	1,979	***4.1***
s35932	17,828	7,820	43.2	1,007	1,007	462	**8.2**	9.5	2,014	1,149	1,486	**7.1**	36.1	5,224	3,264	3,364	**5.4**	92.6	8,881	6,012	6,166	***3.3***
s38417	23,703	7,854	27.8	1,065	1,060	655	**5.0**	41.1	5,743	3,053	3,070	**3.5**	81.8	12,849	7,072	7,613	**2.5**	99.9	17,952	12,578	13,912	***2.4***
s38584	20,715	6,793	28.1	1,393	1,379	740	**6.2**	25.4	3,813	1,932	2,654	**5.1**	88.0	10,732	5,618	6,787	**3.4**	93.5	11,864	7,628	11,274	***3.3***
s5378	2,993	973	25.2	148	148	6	**5.8**	33.7	604	312	30	**4.4**	84.9	1,507	796	83	**3.1**	100.0	1,777	1,167	149	***2.9***
s9234	5,844	2,046	25.2	220	220	38	**9.9**	23.7	791	246	135	**8.2**	66.4	1,848	635	315	**6.2**	93.0	2,905	1,563	575	***5.8***

non-robust																							
circ	#gates	#detect	w/o segmentation					w/ segmentation															
								s1					s2					s3					
			$\%ac$	$\|p\|$	$\|tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$	$\%rc$	$\|seg\|$	$\|+tp\|$	$t(s)$	$\emptyset sps$
b14	10,042	4,794	32.9	709	709	624	**9.4**	22.2	3,735	1,967	2,151	**4.0**	72.3	11,638	5,223	5,787	**1.6**	100.0	22,194	10,768	12,782	***1.5***
b15	8,852	5,756	37.3	741	740	927	**9.9**	31.3	6,147	3,703	3,556	**4.2**	80.7	12,864	7,990	8,095	**2.9**	100.0	17,227	13,138	15,479	***2.4***
b17	32,229	18,149	35.3	2,522	2,516	10,513	**15.6**	29.8	22,720	13,039	33,033	**8.3**	86.6	48,475	29,548	72,024	**4.3**	100.0	61,446	48,041	125,358	***4.0***
b20	20,204	10,937	37.0	1,417	1,417	2,689	**8.3**	22.7	9,148	5,588	8,977	**3.1**	80.7	23,999	14,699	21,962	**1.8**	100.0	41,252	27,451	44,034	***1.6***
b21	20,549	10,585	36.3	1,454	1,454	2,470	**10.0**	21.4	8,892	5,085	8,596	**4.4**	76.4	23,317	13,511	21,080	**2.5**	100.0	43,113	27,130	43,889	***1.5***
b22	29,929	16,265	39.9	2,106	2,106	5,574	**12.9**	22.0	14,717	9,085	18,975	**4.2**	84.6	38,817	24,966	46,724	**2.1**	100.0	62,198	44,365	92,138	***1.9***
c5315	2,485	1,068	23.3	161	160	105	**221.8**	14.8	755	252	274	**109.0**	34.1	1,454	458	509	**86.2**	100.0	3,788	2,228	1,062	***58.4***
c6288	2,448	2,207	54.2	119	116	347	**55.3**	44.3	3,236	1,867	1,645	**1.6**	69.2	3,781	2,281	3,212	***1.5***	100.0	7,736	6,369	10,592	**3.5**
c7552	3,718	1,428	23.7	230	230	147	**139.4**	11.5	804	266	346	**69.1**	30.2	1,611	526	621	**61.6**	100.0	5,016	2,834	1,387	***32.0***
s13207	8,601	1,771	17.4	411	401	126	**8.9**	39.9	1,793	627	465	**5.7**	95.2	5,075	1,977	1,300	**4.3**	100.0	6,553	3,599	2,481	***4.2***
s15850	10,372	3,458	25.7	449	444	297	**9.9**	31.8	1,840	906	915	**6.5**	88.1	6,130	3,248	2,336	**4.7**	99.9	7,268	5,140	4,278	***4.4***
s35932	17,828	7,820	43.2	1,007	1,007	469	**8.2**	19.4	2,848	1,618	1,776	**6.5**	51.5	7,049	4,359	4,324	**5.2**	100.0	12,423	8,798	8,461	***3.3***
s38417	23,703	8,176	26.3	1,119	1,115	1,131	**7.9**	40.9	6,211	3,108	4,141	**4.0**	85.2	14,561	7,644	9,874	**2.7**	100.0	21,054	14,297	18,135	***2.5***
s38584	20,715	7,855	29.8	1,482	1,341	1,274	**19.0**	25.8	4,489	2,030	3,974	**17.3**	90.5	12,449	6,067	9,352	**15.2**	93.3	13,745	8,363	15,309	***15.1***
s5378	2,993	1,063	25.9	151	151	9	**9.0**	33.7	633	329	39	**6.7**	85.2	1,581	844	103	**3.9**	100.0	1,861	1,233	181	***2.9***
s9234	5,844	2,759	24.9	247	243	180	**39.6**	26.3	918	288	414	**25.8**	64.7	2,143	717	733	**11.4**	93.3	3,527	1,902	1,200	***10.4***

TABLE III. DIAGNOSIS ACCURACY FOR *c5315* WITH INCREASED NUMBER OF INITIAL PATHS

$\|p\|$	w/o segmentation	$s1$	$s2$	$s3$
161	221.8	109.0	86.2	58.4
319	51.5	27.7	25.0	23.5
482	25.9	16.3	15.1	14.9

As shown in Table III the diagnosis accuracy can be improved by increasing the number of initial paths. An improvement by the factor of 14.9 is achieved from the first diagnosis without segmentation (221.8) to diagnosis with segmentation (14.9).

VII. CONCLUSION

A simulation based algorithm for delay fault diagnosis has been proposed. To improve the initial diagnosis accuracy three methods for segmentation were introduced and diagnostic TPs for the paths and segments were computed by a SAT-solver.

As result, we significantly improved the diagnosis accuracy and figured out, that more improvements of the diagnosis accuracy can be achieved, when the number of initial paths increases.

For future work we focus on a reduction of the TPs for the segments e.g. with a heuristic approach. Further, a diagnosis algorithm to handle multiple delay faults will be examined.

REFERENCES

[1] C. J. Lin and S. Reddy, "On Delay Fault Testing in Logic Circuits," *Computer–Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 694–703, 1987.

[2] K. Heragu, J. Patel, and V. Agrawal, "Segment delay faults: a new fault model," in *VLSI Test Symposium*, 1996, pp. 32–39.

[3] A. Pramanick and S. Reddy, "On the design of path delay fault testable combinational circuits," in *Fault-Tolerant Computing*, 1990, pp. 374–381.

[4] Y.-Y. Chen and J.-J. Liou, "Diagnosis Framework for Locating Failed Segments of Path Delay Faults," *Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 755–765, 2008.

[5] Z. Wang, M. Marek-Sadowska, K.-H. Tsai, and J. Rajski, "Delay fault diagnosis using timing information," in *Quality Electronic Design*, 2004, pp. 485–490.

[6] P. Camurati, A. Lioy, P. Prinetto, and M. Reorda, "Diagnosis oriented test pattern generation," in *European Design Automation Conference*, 1990, pp. 470–474.

[7] P. Girard, C. Landrault, S. Pravossoudovitch, and B. Rodriguez, "A diagnostic ATPG for delay faults based on genetic algorithms," in *International Test Conference*, 1996, pp. 286–293.

[8] R. Tekumalla, "On test set generation for efficient path delay fault diagnosis," in *VLSI Test Symposium*, 2000, pp. 343–348.

[9] S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and R. Drechsler, "MONSOON: SAT-Based ATPG for Path Delay Faults Using Multiple-Valued Logics," *Journal of Electronic Testing*, vol. 26, no. 3, pp. 307–322, 2010.

[10] S. Prabhu, M. Hsiao, L. Lingappan, and V. Gangaram, "A SMT-based diagnostic test generation method for combinational circuits," in *VLSI Test Symposium*, 2012, pp. 215–220.

[11] S. Padmanaban and S. Tragoudas, "An implicit path-delay fault diagnosis methodology," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1399–1408, 2003.

[12] A. Krstic, L.-C. Wang, K.-T. Cheng, J.-J. Liou, and T. Mak, "Enhancing diagnosis resolution for delay defects based upon statistical timing and statistical fault models," in *Design Automation Conference*, 2003, pp. 668–673.

[13] Y. Lim, J. Lee, and S. Kang, "Path delay fault diagnosis using path scoring," in *International SoC Design Conference*, vol. 02, 2008, pp. II–199–II–202.

[14] G. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Automation of Reasoning*, ser. Symbolic Computation, J. H. Siekmann and G. Wrightson, Eds. Springer Berlin Heidelberg, 1983, pp. 466–483.

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

[16] J. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278–291, July 1966.

[17] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.