J Electron Test (2012) 28:843–856
DOI 10.1007/s10836-012-5333-0

A Non-Enumerative Technique for Measuring Path Correlation in Digital Circuits

Stelios N. Neophytou · Kyriakos Christou · Maria K. Michael

Received: 7 November 2011 / Accepted: 25 September 2012 / Published online: 19 October 2012
© Springer Science+Business Media New York 2012

Abstract The correlation between the physical paths of a digital circuit has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. When considering the complexity and tight timing constraints of modern circuits, this correlation affects both the design process and the testing approaches followed in manufacturing. In this work we quantify the diversity of a set of paths (or path segments), let these be critical I/O paths, error propagation paths for various fault models, or paths traced for diagnostic purposes. Circuit paths are encoded using Zero-Suppressed Binary Decision Diagrams (ZBDDs); the proposed method consists of a sequence of standard ZBDD operations to provide a measure of the overlap of the paths under consideration. The main contribution of the presented method is that, path or path segment enumeration is entirely avoided and, hence, a large number of paths can be considered in practical time. Experimentation using standard benchmark circuits demonstrates the effectiveness of the approach in showing the difference in path correlation between various critical I/O path sets as well as propagation paths during test application.

Keywords Path similarity · Zero-Suppressed Binary Decision Diagrams · Delay testing · Timing analysis · Path diversity

1 Introduction

A number of digital design automation procedures involve the examination and the characterization of the physical paths of a circuit's netlist. Applications including, automatic routing, design verification, fault diagnosis and test generation use information regarding the physical paths in order to accommodate various issues and provide efficient algorithms and techniques. The number of physical I/O paths in the circuit, as well as the number of path segments between two circuit locations, play an important role in almost all of the above processes. Also, it is often important to have an indication on the similarities (e.g. the number of common lines) between two or more paths. As an example of the above, the work in [1] considers a circuit partitioning problem where minimal cutsize and minimal delay per partition are desired. Special consideration should be given so that paths with large line overlap

Responsible Editor: C. Metra

This work was co-funded by the European Regional Development Fund and the Republic of Cyprus through the Research Promotion Foundation (Project NEA ΥΠΟΔΟΜΗ/ΣΤΡΑΤΗ/0308/26).

S. N. Neophytou (✉)
ECE Dept., University of Nicosia, Nicosia, Cyprus
e-mail: neophytou.s@unic.ac.cy

S. N. Neophytou · K. Christou · M. K. Michael
KIOS Research Center of Intelligent Systems and Networks, University of Cyprus, Nicosia, Cyprus

K. Christou · M. K. Michael
ECE Dept., University of Cyprus, Nicosia, Cyprus

K. Christou
e-mail: christou@ucy.ac.cy

M. K. Michael
e-mail: mmichael@ucy.ac.cy

are not part of the same cut since this gives partitions with unbalanced delays, although the cutsize can be minimal.

Apart from numerous design procedures that can use to their benefit information on the similarities of physical paths, techniques for design verification, manufacturing testing and diagnosis can also employ this information to significantly increase their efficiency. Specifically, simulation-based applications for design error and fault diagnosis, such as those in [3, 21], can dramatically increase their diagnostic resolution if the fault/error propagation path(s) of one fault is not a subpath of those of other faults/errors. In this manner, the fault is distinguishable. In such cases, diagnostic test generation approaches should consider path overlaps during the test generation process.

Path overlap is also important in testing. Due to the increased complexity and radical technology scaling in modern digital circuits, it is now apparent that older fault models such as the stuck-at and transition delay fault models, no longer suffice in detecting emerging chip defects. Test generation procedures try to enhance the quality of the generated test set in terms of various measures beyond stuck-at/transition fault coverage and test set size, in an attempt to increase the defect coverage. Some examples of such measures are testing for low power dissipation, multiple fault detections (*n*-detect), path delay fault (PDF) coverage, small delay defect (SDD) coverage, etc. A common concept that must be handled by the test generation procedures for many of the above models and measures is that of path activation and path propagation. The quality of the generated test sets in terms of defect coverage can be enhanced if the various tests activate (propagate) faults via activation (propagation) paths with small overlap [6, 12].

An important usage of the information regarding the similarity among a set of physical paths of a circuit is in test generation for delay faults. Delay faults caused by emerging technology defects and process variations can dramatically change the timing behavioral of a circuit and can lead to unacceptable yield loss, especially for high performance products [4]. The most difficult to identify delay faults are those that are due to small delays and are distributed across various circuit lines, widely known as small delay defects [6, 7, 20, 22]. Since the accumulation of small delay defects is highly related to the circuit's physical paths, the Path Delay Fault model (PDF) proposed in [15] is considered among the most accurate models for delay related defects. According to the PDF model, a fault is designated as the late transition across a physical path, either rising of falling. Since the number of physical paths in a circuit can be, in the worst case, exponential to the circuit size, deploying the PDF model in test generation is impractical, especially for dense circuits. A number of previously proposed test generation methodologies try to address the complexity issue of the PDF model by targeting a subset of the fault universe which only contains the *critical faults* [5, 18, 19, 23]. Different criteria and delay models define the "criticality" attribute of the faults according to the specific problem examined, with main objective to reduce the fault list and, thus, make the test generation procedure simpler and the obtained test set smaller. Most of the previous proposed techniques relate criticality with the length of the path or the slack of the path, i.e., the difference between the expected delay of the path and the maximum allowable delay in the circuit. The slack is either estimated using statistical methods [22, 23] or calculated using process variation data obtained from the manufacturing process [18]. The work in [22] takes an *n*-detect test set for transition faults as input and reduces its size by selecting those tests that sensitize the most critical paths according to a statistical approach. In an attempt to further improve the selection procedure the techniques of [5, 19] model additional technology parameters like coupling noise and power supply noise effects. The latter two approaches have proven the need for adding more criticality characteristics to physical paths than just their length.

The work in this paper proposes a non-enumerative methodology to identify path (segment) overlap in a given set of physical paths, in order to quantify the correlation between the paths of the set considered. This set can be a subset of the circuit's I/O paths (i.e., set of critical paths), or path segments representing, for example, error/fault activation or propagation paths. A major contribution of the proposed technique is that it avoids explicit path or path segment enumeration by using a canonical representation of the paths based on Zero-Suppressed Binary Decision Diagrams (ZBDDs) [9] which have been shown to cope well with huge numbers of paths [13]. The method applies a constant number of standard ZBDD operators (of polynomial complexity) to find the path overlaps and represent them implicitly in a ZBDD providing a metric for the correlation of the paths in the given set. The different statistics obtained by this procedure are then summarized into a single number that can be used to characterize the correlation between the paths of each set. As mentioned in the previous paragraph, different applications can use this measure (or the provided statistics) according to their nature and indented goal. For

example, path selection approaches for PDF or SDD test generation can use this measure to avoid selecting paths similar to paths that have already been selected, since this will, probably, not identify additional delays in the circuit. We elaborate more on the motivation for this particular application in Section 2.

Section 3 gives the detailed description of the methodology and all the appropriate definitions of the ZBDD operations used. Specifically, we define a measure for the average path overlap and explain its usage and meaning. We then provide some basic preliminaries on ZBDDs and explain how we use them in the proposed technique. Finally, we present and describe the basic steps of the proposed approach as well as a newly proposed operator that is used for calculating the overlap sizes in ZBDDs.

Section 4 shows the experimental results and presents the relevant discussion. We have used two different approaches in order to demonstrate the applicability of the proposed method and measure. The first one considers different groups of critical I/O paths and quantifies the overlap between the paths for each group. The second approach identifies the overlaps between the propagation paths obtained from different test sets and investigates their relation with defect coverage. Section 5 concludes the paper.

2 Motivation

The discussion in Section 1 mentions a number of different design automation applications that can benefit from using path overlap information. Here, we briefly present one such possible application which we use in our experiments to evaluate the proposed method.

Consider test generation methods for small delay defects (SDDs). A number of methodologies consider the PDF model (either explicitly or implicitly) to model the effect of such defects and try to reduce the number of faults by selecting the most important PDFs known as critical PDFs (see [5, 18, 19, 22, 23], among others). Criticality can be defined by various issues (often combined), such as path length and process variation parameters. These approaches do not explicitly consider path similarities and may result into selecting highly correlated paths. The latter is not necessarily undesired, but when distributed delays due to SDDs are targeted, selecting two (or more) correlated paths may not be the best choice. We demonstrate this case with an example.

Consider the circuit in Fig. 1 and assume, without any loss of generality, the fixed delay model with each

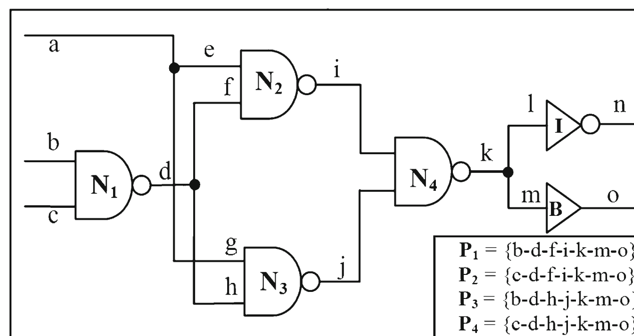

Fig. 1 A path selection method example

NAND gate having nominal delay of 3 time units while the buffer and the inverter have 2 and 1 units, respectively. Let us assume that the maximum allowable I/O delay is set to 12 units in order to allow a small slag from the maximum path delay (11 units). Additionally, assume that the PDF test budget for this circuit is only 2, i.e., we can afford only to consider two physical paths in the test generation process. Hence, the path selection method must select only 2 paths out of the 12 paths of the circuit to use as input to the test generation process. According to the gate delays, we identify that the critical paths in the circuit are four, $P_1 = \{$b-d-f-i-k-m-o$\}$, $P_2 = \{$c-d-f-i-k-m-o$\}$, $P_3 = \{$b-d-h-j-k-m-o$\}$ and $P_4 = \{$c-d-h-j-k-m-o$\}$. There exist six different groups of paths with cardinality 2; considering any one of these groups can be viewed as a valid input candidate to the PDF test generation. If, however, we examine the overlap of each group the conclusion may result in a different decision. The overlap information for the path pairs is given at the first 6 rows of Table 1. For instance, the paths of pair G_1 have an overlap of 85.71 % as there are 6 common lines out of the average path length which is 7 in this case (more details on this calculation are given in the next section). If a group with large overlap (e.g. G_1) is selected to be considered in the test generation procedure, its contribution in identifying small delays will be limited because the cumulative effect of the distributed

Table 1 Path overlap for the example of Fig. 1

Group	Paths	Overlap (%)
G_1	P_1, P_2	85.71
G_2	P_1, P_3	71.43
G_3	P_1, P_4	57.14
G_4	P_2, P_3	57.14
G_5	P_2, P_4	71.43
G_6	P_3, P_4	85.71
G_A	P_1, P_2, P_3	71.43

delays will fall in the paths' common segment i.e., in {d-f-i-k-m-o}. To make it more specific, consider that due to process variations gates N_1, N_3 and B exhibit excessive delay of 0.5 units while all the remaining gates exhibit their nominal delay. This situation can give unacceptable total delay across an I/O path of the circuit, as in the case of P_3 (delay is $3.5 + 3.5 + 3 + 2.5 = 12.5$ units). Recall that the test generation process has a 2 path budget so a decision should be taken on which of the 6 pairs should be considered. By testing the two paths in G_1 the excessive cumulative delay is not detected as both P_1 and P_2 will give total delay of 12 (i.e., $3.5 + 3 + 3 + 2.5$) which is acceptable. On the other hand, if a group with smaller correlation (overlap) between its paths (any other that G_1 and G_6) is selected, the delay will be detected.

Generally, when the group with the smaller correlation is selected, the possibility to detect an excessive cumulative delay across anyone of the considered paths is increased. For instance, consider that for the circuit in Fig. 1 we are given 3 groups of paths, G_1, G_2, and G_3, and we are asked to decide which of this group is better to be given as input to a test generation procedure. In order to get excessive delay in the circuit, more than 1 time units of delay should be distributed among the gates of a critical path. By selecting the group with the smaller overlap (i.e., G_3) the obtained tests detect any excessive delay across any of the paths considered here as critical. This happens because paths P_1 and P_4 cover all the possible combinations of sub-paths between the 4 paths that are considered as critical.

Path correlation is calculated across consecutive and non-consecutive path segments like the case of group G_5, i.e., paths P_2 and P_4. It is important to ensure that this calculation does not affect the efficiency of the path selection approach, so it must avoid explicit path or path segment enumeration, since the number of paths in a circuit can be exponential to the circuit size. This is a very important attribute, since the modern design optimization tools tend to give circuits with paths of similar length, increasing the number of candidate critical paths. We elaborate further on the path correlation calculation on Section 3.

3 Proposed Methodology

This section discusses the proposed methodology. Section 3.1 gives the necessary definitions for pairwise path overlap in a set of paths. All the steps described here use a canonical data structure (ZBDD) for implicit path representation and manipulation. We provide a brief overview of ZBDD based path encoding in Section 3.2. Sections 3.3 and 3.4 present the proposed ZBDD-based operations to calculate path overlap and an illustrative example, respectively.

3.1 Pairwise Path Overlap Calculation

Definition 1 Given two paths P_1 and P_2 the overlap operator $C(P_1, P_2)$ gives the set of lines that are common in P_1 and P_2.

The calculation of the path overlap between two paths of the same size is done by dividing the number of common lines over the paths' size i.e., $|C(P_1, P_2)| / |P_1|$. For paths of different size, an averaging of the two paths' length is required. Given two paths P_1 and P_2 the overlap percentage is given as the size of the overlap between the two paths over the average size of the two paths, i.e., $|C(P_1, P_2)| / (|P_1| + |P_1|)/2$. Depending on the targeted application this definition of the average overlap may change by selecting different path sizes as a reference. For example, it may be meaningful to divide over the minimum size of the two paths (i.e., $|C(P_1, P_2)| / \min\{|P_1|, |P_2|\}$) in order to consider paths segments fully contained in other paths. Similarly, the maximum size of the two paths may be used or any other combination that fits the targeted problem. While any of these approaches may be followed, without affecting the efficiency or the complexity of the proposed method, in this work we consider the path overlap with respect to the average size of the two paths. This approach makes the methodology proposed here more generic as it can be successfully used in different problems that require path overlap information.

Definition 2 Given a set of paths $G = \{P_1, P_2, ..P_n\}$, $\mathcal{D}$ is the pairwise path overlap operator computing the path overlap between all unique pairs of paths in G, given by:

$$\mathcal{D}(G) = \{C(P_1, P_2), C(P_1, P_3), \ldots C(P_1, P_n), C(P_2, P_3), \ldots C(P_{n-1}, P_n) : \left|C(P_i, P_j)\right| \neq 0\} \quad (1)$$

The above operator checks all the pairs of paths (P_i, P_j), where $P_i, P_j \in G$ and $i \neq j$ and gives the sets of lines that form the overlap between the paths of each pair and have a non-zero size. Eventually, the obtained set $\mathcal{D}(G)$ contains all the path segments that are common in any of these pairs. Using the resulting set $\mathcal{D}(G)$

we give the following definition for the average overlap of a set of paths:

Definition 3 Given a set of paths $G = \{P_1, P_2, \ldots P_n\}$, the average pair-wise overlap measure $O(G)$ is given by:

$$O(G) = \frac{\sum_{(P_i,P_j)\in\mathcal{D}(G)} \left|C(P_i, P_j)\right| / |\mathcal{D}(G)|}{\sum_{P_i\in G} |P_i| / |G|} \tag{2}$$

This measure gives the average pairwise overlap between the paths of a set of paths G. For example, the calculation of the average pairwise overlap for a path set G_A which contains P_1, P_2 and P_3 of the circuit of Fig. 1 is 71.43 % calculated as the average between the overlaps in G_1, G_2 and G_4, the pairwise combinations of the paths of G_A.

Note that the measure was selected to consider pairs of paths and not 3-tuples, 4-tuples and so on, in order to be generic and applicable in a broader range of applications. By selecting pairs we actually include all the combinations of paths (pairs, 3-tuples, 4-tuples, etc.) and, thus, incorporate the total correlation between the considered paths in the measure. On the other hand, the averaging in Definition 3 ensures that no double counting of common sub-paths occurs and each common sub-path contributes equally to the measure. Figure 2 illustrates three different examples of path overlap between three different paths, using Venn Diagrams, in order to demonstrate the application of the operators presented in the definitions above. Let each set correspond to a circuit path. The elements of a set correspond to the circuit lines constituting the path. Elements in the intersection of two sets are the lines that are common in the corresponding paths, i.e., form the overlap between the two paths. Consider first Fig. 2a with P_1 = {a-b-c-d-i-m}, P_2 = {d-e-f-g-h-i-m} and P_3 = {h-i-j-k-l-m}. The overlap operator of Definition 1 gives $C(P_1, P_2)$ = {d, e, i, m} and $|C(P_1, P_2)| = 4$. Similarly, $|C(P_1, P_3)| = 2$ and $|C(P_2, P_3)| = 3$. Applying Definition 2 on this set of paths gives $\mathcal{D}(G) = \{C(P_1, P_2), C(P_1, P_3), C(P_2, P_3)\}$ = {{d, e, i, m}, {i,m}, {i, m, h}}, i.e., all the non-empty path overlaps. Using the average pairwise overlap measure of Definition 3 we calculate the average overlap between the considered paths to be:

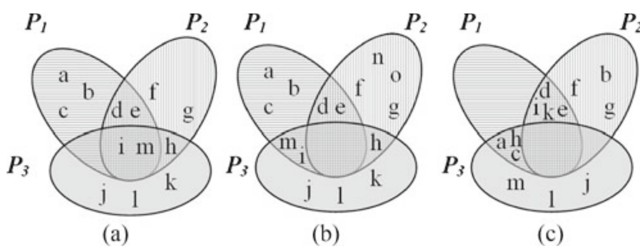

Fig. 2 Examples for the calculation of the average path overlap measure of Definition 3

$$O(G) = \frac{\sum_{\substack{i,j=1,\\ C(P_i,P_j)\in\mathcal{D}(G)}}^{3} \left|C(P_i, P_j)\right| / |\mathcal{D}(G)|}{\sum_{\substack{i=1,\\ P_i\in G}}^{3} |P_i| / |G|}$$
$$= \frac{(|C(P_1, P_2)| + |C(P_1, P_3)| + |C(P_2, P_3)|)/3}{(|P_1| + |P_2| + |P_3|)/3}$$
$$= \frac{(4+2+3)/3}{(7+7+6)/3} = 45\ \% \tag{3}$$

Observe that the two lines that form the overlap between all the three paths (i.e., i and m) contribute more in the above measure than the lines that are common only in two sets (paths). This becomes more clear if we consider the paths in Fig. 2b. Here, P_1 = {a-b-c-d-i-m}, P_2 = {d-e-f-g-h-n-o} and P_3 = {h-i-j-k-l-m}. For this case we get $O(G) = \frac{(2+2+1)/3}{(7+7+6)/3} = 25\ \%$ indicating that the average overlap is smaller, although the number of lines in the overlap of any two paths as well as the size of all paths considered are the same as in case (a). Obviously, the fact that two lines are in the overlap of all three paths in the diagram of Fig. 2a increases the average overlap compared with the diagram of Fig. 2b. This is reflected to the value of the measure introduced in Definition 3 even though the paths are considered in pairs and not in 3-tuples (25 % for (b) against 45 % for (a)). Furthermore, the averaging in both the numerator and the denominator ensures that the values of the measure for different path sets are comparable, when the sets are obtained for the same circuit. For instance, the paths in Fig. 2c have clearly higher average overlap than the previous two cases. Here, P_1 = {a-c-d-e-h-i-k}, P_2 = {b-d-e-f-g-i-k} and P_3 = {a-c-h-j-l-m}. However, one pair gives zero overlap (i.e., $|C(P_2, P_3)| = 0$) and it should not be considered by the measure. If we had only considered the sizes of the overlaps and the paths, the measure would indicate that the paths in Fig. 2a have more overlap than those in Fig. 2c (45 % for (a) against 35 % for (c)), which is not confirmed by the diagrams. Applying the measure to the paths represented by the diagrams in Fig. 2c gives a higher average overlap value i.e., $O(G) = \frac{(4+3)/2}{(7+7+6)/3} = 52.5\ \%$ confirming the larger overlap shown in the diagrams.

3.2 Path Representation and Manipulation

The proposed methodology uses Zero-Suppressed Binary Decision Diagrams (ZBDDs) to represent a large number of paths and obtains information on their overlap using standard ZBDD operations. ZBDDs [9] have been successfully used in representing a huge number of paths compactly in a non-enumerative fashion [12]. The representation of paths in a circuit requires only a polynomial number of standard ZBDD operations and can be generated and represented in a ZBDD format (as in [12]), by a single topological traversal. Extensive experimentation has shown that, even circuits with huge number of paths can be effectively represented using ZBDDs following the approach of [12], which avoids path enumeration.

Figure 5a shows the representation of set G_A of Table 1 that consists of paths P_1, P_2 and P_3 of Fig. 1, using ZBDDs. The vertices in the diagram correspond to circuit lines, and always follow a predefined order (here b < c < d < f < i < h < j < k < m < o). A path can be obtained by following the paths of the diagram that end in the terminal vertex 1 and considering only the solid (true) edges. The true edges correspond to the existence of a circuit line in a path, whereas the false (dashed) edges correspond to the absence of a line in a path. The shaded part of the diagram represents path $P_3 = \{$b-d-h-j-k-m-o$\}$. Observe that, while vertex f falls in the shaded area, it is not considered in the path since it is connected to h via a dashed edge. Likewise, the other two paths are represented in the diagram. In the same way ZBDDs can represent path segments that may not necessarily consist of consecutive circuit lines.

3.3 Overlap Identification Algorithm

Given a circuit C and a set of paths G, the proposed algorithm identifies a set of overlapping path segments D(G) in a non enumerative way using a constant number of standard ZBDD operators. Set G does not necessarily contain I/O full paths, but it can contain path segments as well. This allows the algorithm to handle path segments as well, which is desirable for a number of applications like fault diagnosis, where we want to distinguish between different sub-paths. The algorithm first implements the pairwise path overlap operator and then performs a final step to obtain the required overlap information.

The procedure of Fig. 3 outlines the basic steps of the algorithm. In the first step (line 01) the procedure builds the ZBDD that contains the paths in the given set G. This can be done following a procedure similar to those used in [2, 13]. These works present techniques to

```
Procedure: Path_Overlap_Identification
Inputs: C: Circuit Under Examination,
G: Set of Paths (or path segments) in C
Output: Overlap Statistics H
01: Z_0^T ← ZBDD_represent(C, G)
02: Z_1^T ← Z_0^T ⊗ Z_0^T
03: Z_2^T ← maximal (Z_0^T)
04: Z_3^T ← Z_2^T ∩ Z_1^T
05: Z^T ← Z_1^T / Z_3^T     # Z^T corresponds to D(G)
06: H ← retrieve_overlap_histogram(Z^T)
07: return overlap_statistics(H)
```

Fig. 3 Basic steps of the proposed method

obtain either all or some of the paths of a circuit based on specific criteria, using at most two linear traversals of the circuit. In our experimentation, we have used the approach of [2] to obtain the ZBDD for the critical/sensitized paths in the circuits considered. The step in line 02 applies a standard ZBDD operation called pairwise intersection ($\otimes$) which takes all pairs of the paths in the given set and identifies the overlap between the paths of each pair. The pairwise overlap operation examines all the pairs of paths in Z_0^T and gives a set containing the sets of lines (sub-paths) in each pair that form the overlap of the two paths. Hence, Z_1^T contains the overlaps that were identified between any two paths of set G. At this point we note that as a side effect of the pairwise intersection operator all the paths (or sub-paths) initially in G will be also contained in Z_1^T, since the operator does not exclude pairs of the same path. In this case a path initially in G is identified as overlap since the overlap between two identical paths is the path itself. Using the notation of Definition 1, Z_1^T contains all the $C(P_i, P_i)$ for all $i = 1, 2 \ldots |G|$ which are equal to P_i for all i. This situation is not desired when only the overlap between distinct paths is required. The next three steps of the algorithm concentrate on canceling this side effect considering all the possible complications. Lines 03 and 04 give the diagram that contains all the unwanted paths that should be removed from the pairwise intersection result. Intuitively, if we remove from Z_1^T all the paths in G this problem will be solved. However, this is not true when G contains not only I/O full paths (from a primary input to a primary output) but sub-paths as well. In this case it is not clear whether an overlap in Z_1^T is a true overlap or a sub-path initially contained in the given set G. The maximal operator

(line 03) removes from G all the paths that are proper sub-paths of other paths, and, hence, removes this disambiguation. This step is not necessary if G contains only full paths. The maximal operation on a set of full paths will return the set itself and so this step does not affect the case where G consists only of full paths. Line 04 identifies the paths that must be removed from Z_1^T. The set difference operation of line 05 removes all the unwanted paths from the obtained overlap set, keeping only the sub-paths that corresponds to the overlap of any two distinct paths in set G. At this point we have all the overlapping sub-paths in Z^T which is the ZBDD representation of the set D(G) used in Definition 2.

In line 06, the overlap information is retrieved from the ZBDD in a histogram-like structure, using a recursive operator, while line 07 returns different statistics for the obtained overlap set the most important of which is the average pairwise overlap (Definition 3) and the standard statistical measure of skewness of the histogram H. We elaborate further on the usage of the skewness measure in Section 4.

Since this is an algorithm that relies on the usage of ZBDDs we will discuss the algorithm's complexity with respect to basic ZBDD operations. The interested reader is referred to [10] for further details on the complexity of each ZBDD operation. The first step of the algorithm (line 01) is a preprocessing step carried out by a topological order, input-to-output, linear traversal of the circuit's netlist in order to represent the given paths with a ZBDD structure. This representation requires a linear, to the number of circuit's lines, number of basic ZBDD operations and it is known to scale very well when representing sparse combinational sets as is the case with the paths of a digital circuit [9, 10]. Moreover, previous works have shown experimentally that the representation of digital circuits' paths can be done in small time and memory, even for circuits with a huge number of paths like c6288 from the ISCAS85 benchmark suite (see [2, 13], among others). Beyond this preprocessing step, we discuss the complexity of the proposed approach to derive the ZBDD containing the path overlaps (lines 02–05) with respect to basic ZBDD operations. According to [10], the various basic ZBDD operations can be performed recursively and can be efficiently implemented (linear time with respect to the ZBDD size) when a table of precomputed results is maintained, referred to as *cache* (also see the implementation in [16]). These basic operations are Union, Intersection, Set Difference, Subset, Count, Change, Base and Empty [10].

In step 02 pairwise intersection is performed by using the Cross-Product operator of ZBDDs. This operator is also recursive in nature (same as the basic operations mentioned above) with an additional cost of at most two Union operations per ZBDD node. The two extra Union operations are performed only when the Cross-Product is applied between ZBDDs that are rooted with the same node in order to merge the two recursive results from each node's children [16]. In step 03 the maximal operator is also executed in a recursive manner with an additional cost of a Subset operation per ZBDD node in order to identify, at each node, whether there is a sub-path contained in a larger path [16]. The two operators of steps 04 and 05 perform Intersection and Set Difference, respectively. Hence, the algorithm's complexity sums up to a very small number of basic ZBDD operations per ZBDD node (among ZBDDs $Z_0^T \sim Z_3^T$ shown in Fig. 3), in the worst case. This implies a non prohibitive execution time for the proposed method which is verified by the resource requirements reported at the experimental results section.

The operator of step 06 is a newly proposed ZBDD operator, which is executed in linear time to the size of the ZBDD and, thus, does not increase the method's complexity. It avoids explicit path or path segment enumeration, and hence, does not contribute further to the algorithm's complexity. The pseudocode of Fig. 4 describes the operator of line 06 in Fig. 3. The operator takes a ZBDD (Z^T) as an argument and calculates

Procedure: Retrieve_Overlap_Histogram

Inputs: Z^T: ZBDD representing a set of Sub-Paths

Output: Histogram H

01: $H_{zero} \leftarrow [0, 0, 0 ..., 0]$, $H_{one} \leftarrow [1, 0, 0 ..., 0]$,

02: K = root(Z^T) *# root() gives the root vertex of ZBDD* Z^T

03: **if** k = terminal_vertex(0)

04: **return**(H_{zero})

05: **else if** k = terminal_vertex(1)

06: **return**(H_{one})

07: **else**

08: $H_T \leftarrow$ Retrieve_Overlap_Histogram(true_edge(K))

 # true_edge() gives the ZBDD rooted at solid line child of K

09: $H_E \leftarrow$ Retrieve_Overlap_Histogram(false_edge(K))

 # false_edge() gives the ZBDD rooted at dashed line child of K

10: $H_K \leftarrow$ right_shift(H_T) + H_E

11: **return** H_K

Fig. 4 Linear complexity operator for obtaining the overlap histogram

an array per ZBDD vertex that is used to represent the overlap histogram, in a recursive manner. Each position of the array holds the number of paths (or sub-paths) that have size (length) equal to the position. For example, array [0, 3, 5, 4, 1] represents a set of paths with 3 overlaps of size 1, 5 with size 2, 4 with size 3 and 1 overlap with size 4. The size of each array is statically defined to be equal to the largest path is the circuit (circuit's depth). The first position of the array indicates a zero size overlap and is only used for terminal vertex 1 to provide termination of the recursion (line 01). Lines 02–06 check if the root vertex of Z^T is a terminal vertex (terminal zero or terminal one) to indicate the termination of the recursive process. If the root of Z^T is the terminal 0 vertex, a zero histogram is returned to the calling process (line 04); for terminal 1 a histogram with a single one in the 0th position is returned (line 06). In all other cases (i.e., where the root of Z^T is not a terminal vertex) the procedure calls itself to calculate the histograms for the two children of Z^T root (lines 08 and 09). Next, the two arrays are merged together with the one coming from the true edge shifted by one position on the right (line 10) indicating the contribution of this child to a sub-path overlap. The procedure returns the histogram array to the calling process in line 11, until the root of Z^T is reached and the histogram is returned to the procedure of Fig. 3. Each vertex is only visited once. Even when a vertex is a child of two or more parents, the calculated array is cached after the first visit and used for all parents of that vertex. Thus, the complexity of the operator is linear to the size of the diagram. In the next subsection, we present an illustrative example of the overall process.

3.4 A Path Overlap Calculation Example

We next give an example to illustrate the execution of the algorithm. Consider again set G_A of the circuit of Fig. 1 (i.e., P_1, P_2 and P_3). Figure 5a shows the ZBDD representation that corresponds to the paths in G_A (Z_0^T in Fig. 3). Figure 5b shows the ZBDD after applying the pairwise intersection operator (Z_1^T in Fig. 3). Observe that this diagram contains a total of 6 paths (or sub-paths): (i) 3 sub-paths corresponding to all possible overlaps between the 3 paths and (ii) the 3 paths of the initial diagram. For instance, the shaded part of the diagram corresponds to the sub-path {d-k-m-o} which is the overlap between paths P_2, P_3. Note that, this information does not correspond to path segments only, but it also contains overlap information than spans in more than one path segment like the case of the shaded path. The intersection and set difference operations (lines 04 and 05 in Fig. 3) remove all the unwanted paths and keep only the overlaps as shown in Fig. 5c (Z^T in Fig. 3).

Finally, in the diagram of Fig. 5d we describe the execution of the recursive operator of Fig. 4 which calculates the overlap histogram. The operator maintains a size array per vertex (shown in black) holding the numbers and sizes of the overlaps so far, from bottom to top of the ZBDD. For example, the array of vertex f denotes that up to this vertex two sub-paths exist, one with size 3 (3rd position in the array) and one with size 5 (5th position in the array). The size of the array is statically determined by the largest true path in the ZBDD (6 in this case) plus one for the 0th position needed for recursion termination. Starting from the

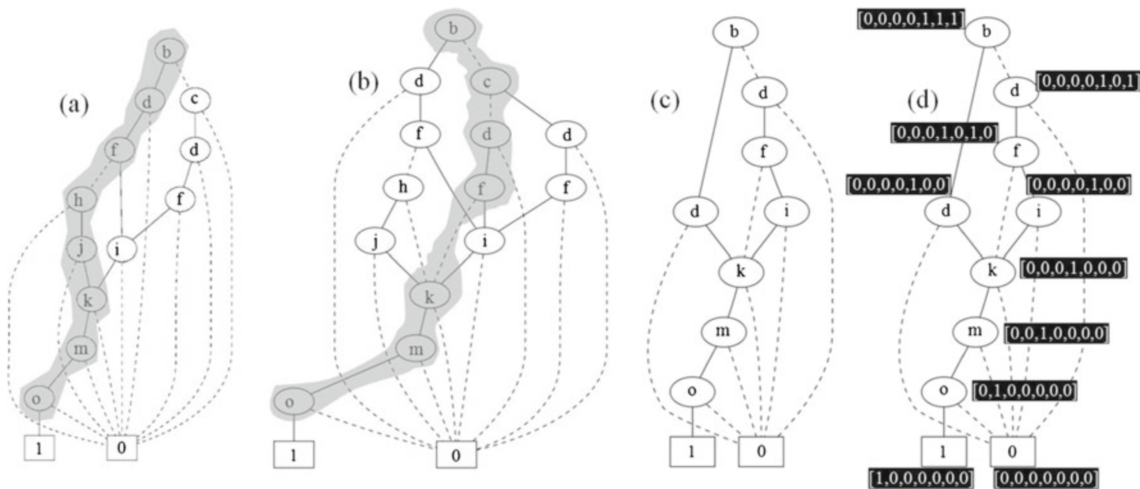

Fig. 5 Using ZBDDs for path set representation and overlap identification. Diagram representing (**a**) the paths in G_A, (**b**) after applying the pairwise intersection operator (Z_1^T in Fig. 3), (**c**) after applying intersection and set difference which results in keeping only the overlaps (Z^T in Fig. 3), and (**d**) execution of the algorithm that retrieves the overlap histogram (operator of Fig. 4)

root vertex of the diagram (a) the operator calculates the histogram array of each vertex by merging the histogram arrays of the two children of the vertex. For example, for vertex f it merges the arrays of vertices i and k by shifting the array coming from the solid edge (vertex i) to the right and adding it to the (non-shifted) array coming from the dashed line (vertex k). This way all the solid edges are considered in the overlap calculation and all the dashed lines are omitted. The process stops the recursion when it reaches one of the two terminal vertices where predetermined arrays are returned (H_{one} and H_{zero} of Fig. 4). The final array associated with the root of the diagram denotes that the obtained overlap set has three overlaps, one with size 4, one with size 5 and one with 6. This array corresponds to the histogram structure H of the algorithm. The average overlap percentage from Definition 3 is then calculated as the average of the three overlaps over the average size of path length in the given set, i.e., $O(G_A) = [(4+5+6)/3]/[(7+7+7)/3] = 71.43\ \%$.

3.5 Extension to Different Levels of Abstraction

The methodology presented in this work can be generalized to be used for different circuit's levels of abstraction, such as generic layout, technology-mapped layout, physical, etc., in order to be able to cope with different problems and/or meet different resource requirements. In this subsection, we discuss one such alternative that requires minimum modifications of the proposed approach.

Consider the case where layout data is available when the overlap information is to be calculated and that the problem under examination investigates delays that are directly proportional to the circuit lines' actual size. In other words, the solution should consider the actual physical size of each circuit line instead of the uniform size considered at the gate level (logical line length), in order to provide a more realistic representation of the circuit's delays. A pre-processing step can be invoked in order to assign weights to circuit lines at the gate level reflecting the connections' actual length at the layout level. This information can be used to augment the definition of the overlap in order to comprise the information obtained at the lower level.

Specifically, the approach can be trivially modified by just redefining how the path size and path overlap sizes are calculated such that it includes the delay information represented by the weights. We referred to this overloaded definition of size as the *magnitude*.

Definition 4 Given a path $P_1 = \{l_1 - l_2 - l_3 - \ldots - l_n\}$ and a set of weights $W(P_1) = \{w(l_1), w(l_2), w(l_3), \ldots, w(l_n)\}$ on each of the path's lines, the *magnitude* of P_1 is given by:

$$\|P_1\| = \sum_{i=1}^{n} w(l_i) \tag{4}$$

Definition 5 Given two paths $P_1 = \{l_1 - l_2 - l_3 - \ldots - l_n\}$ and $P_2 = \{k_1 - k_2 - k_3 - \ldots - k_m\}$ and two respective sets of weights $W(P_1) = \{w(l_1), w(l_2), w(l_3), \ldots, w(l_n)\}$ and $W(P_2) = \{w(k_1), w(k_2), w(k_3), \ldots, w(k_m)\}$ the *magnitude* of the two paths' overlap is given by:

$$\|C(P_1, P_2)\| = \sum_{l_j \in C(P_1, P_2)} w(l_j) \tag{5}$$

By substituting the size used in Definitions 2 and 3 by the magnitude given in Definitions 4 and 5, the algorithm incorporates the layout level information in the calculation of the path overlap and the corresponding results reflect the physical length of each line and not its logical length.

In Fig. 6 we briefly present one possible pre-processing step that can be used to obtain the weights for each line at the gate level using the layout representation of the circuit. This procedure takes as input the circuit description at the layout level and records the physical length of each line (lines 01–03) in array $W(L)$. Next, the greatest common divisor of all the lengths is calculated in order to make weights as small (and simple to use) as possible. Here, without loss of generality, we assume integer lengths. In the case where the lengths are not integers, the minimum length is taken, instead of the greatest common divisor. In lines 05 and 06 the obtained lengths are divided by the number obtained in line 04 and the procedure terminates, giving the desired lengths. After the methodology is applied as described in Section 3.3 and the overlap information is obtained, the metrics are calculated by substituting

```
Procedure: Calculate_Layout_Weights
Inputs: L: Circuit Under Examination (layout)
Output: W(L): Array of Line Weights
01: for each line l_i in L
02:     len = physical_length(l_i)
03:     W(L)[i] = len
04: GCD = greatest_common_divisor(W(L))
05: for i to |W(L)|
06: W(L)[i] = W(L)[i]/ GCD
07: return W(L)
```

Fig. 6 A pre-processing step to calculate line weights

the size (logical length) with the magnitude (physical length) in Definitions 2 and 3.

4 Experimental Results

The proposed method was implemented in C language using the decision diagram package of [16] (for all ZBDD related operations), and run on a 1GHZ Sun-Blade 2500 workstation with 4 GB of memory. We report results for two different test-related procedures involving paths. In Section 4.1 we give as input to the proposed methodology path sets that consist of critical I/O paths with less than 10 % slag. We use the library of the TSMC 0.18 um process to calculate the delays and obtain the slag of the paths and we report statistics for a number of indicative circuits from the ISCAS'85, ISCAS'89 and ITC'99 benchmark suites. In Section 4.2 we compare the average propagation path overlap (path segments), as well as other statistics, of two different test sets generated considering stuck-at faults.

4.1 Set of Critical I/O Paths with Different Average Path Overlap

We first show resources requirements for the proposed method in Table 2. The second column reports the number of paths in the considered path set, while columns 3 and 4 report the proposed methodology's requirements in CPU run time and memory, respectively. These numbers demonstrate that the method is very efficient and with memory requirements that are not prohibitive even for circuits with a large number of paths. In order to point out the efficiency of the proposed method, we compare it with a brute-force approach that does not avoid path enumeration (in the absence of any existing non-enumerative comparable method). The brute-force approach actually selects all path pairs enumeratively and identifies their common lines in order to give the overlap measure according to Definition 2. Column 5 reports the run time in seconds for the brute-force technique. For the circuits with a star (*) the brute-force technique was allowed to run for 48 h before aborted. It is obvious that the proposed methodology is significantly faster than the brute-force technique providing the desired results in seconds, even for circuits with a large number of paths such as those in the last rows of Table 2.

Table 2 Comparison with brute-force approach

Circuit	# Critical I/O paths	Proposed method CPU (s)	Proposed method Mem (MBs)	Brute force CPU
s641	104	0.02	5.65	1.26 s
s5378	360	0.02	13.20	15.21 s
b09_opt	372	0.04	8.57	16.25 s
b12_opt	472	0.02	5.83	26.17 s
s3271	650	0.02	12.95	49.66 s
c880	682	0.02	5.35	54.67 s
s1423	726	0.02	6.73	61.96 s
b11_opt	1,033	0.02	7.64	125.49 s
s713	3,830	0.02	8.65	1,726.22 s
b07_opt	6,659	0.02	6.65	1.45 h
b04_opt	8,960	0.20	5.27	2.62 h
s38584.1	24,300	1.02	75.53	19.30 h
b14_1_opt	33,528	0.60	26.54	36.75 h
c7552	80,640	0.02	23.32	*
c2670	116,100	0.82	15.01	*
c5315	138,720	0.02	31.10	*
c1908	161,280	0.17	29.70	*
b14_opt	175,456	7.06	41.75	*
b05_opt	439,103	0.14	37.64	*
b21_1_opt	1,949,696	0.16	74.71	*
b20_1_opt	1,950,650	0.16	77.81	*
c1355	1,959,600	5.78	84.65	*
b15_1_opt	2,209,040	0.78	95.41	*
c3540	4,641,360	2.14	124.82	*
c6288	2.13×10^{19}	7.95	76.10	*

Next we give the results for the overlap between different sets of paths, as well as some other measures. For each circuit we use as input, four different path sets of equal size. Sets G_1, G_2, G_3 and G_4 consist of critical paths (with less than 10 % slag) and have been selected from a pool of critical paths using four different approaches. G_1 and G_2 were obtained in a random fashion, G_3 was obtained by selecting paths with more overlaps in the longest paths and, finally, G_4 was obtained by selecting paths from different I/O cones. We first discuss in detail the obtained results for the medium size circuit s713 from the ISCAS'89 suite. For each one of the 4 path sets we show the overlap distribution in the histograms of Fig. 7. In the X axis of each histogram we show the overlap size in lines and in the Y axis the number of the sub-paths having this overlap size. The histograms provide a complete picture of the overlap distribution; however this information is not practical, especially if we want to use it to guide a path selection algorithm. The average pairwise overlap measure (O(G) from Definition 3) gives an indication of this distribution using a single number. For example, O(G_2) is 8.43×10^{-3} while O(G_4) is 5.14×10^{-3} indicating that the overlap is larger in G_2 than in G_4. Indeed the distribution of the overlaps, as well as the numbers for each overlap size indicate that this observation in true. Similarly, the overlap in G_3 is larger than in G_1 even thought their distributions seem to be similar. In the case of G_3 however, the number of sub-paths per overlap is much larger than in G_1.

While the measure from Definition 3 gives a very good indication about the amount of overlap in the

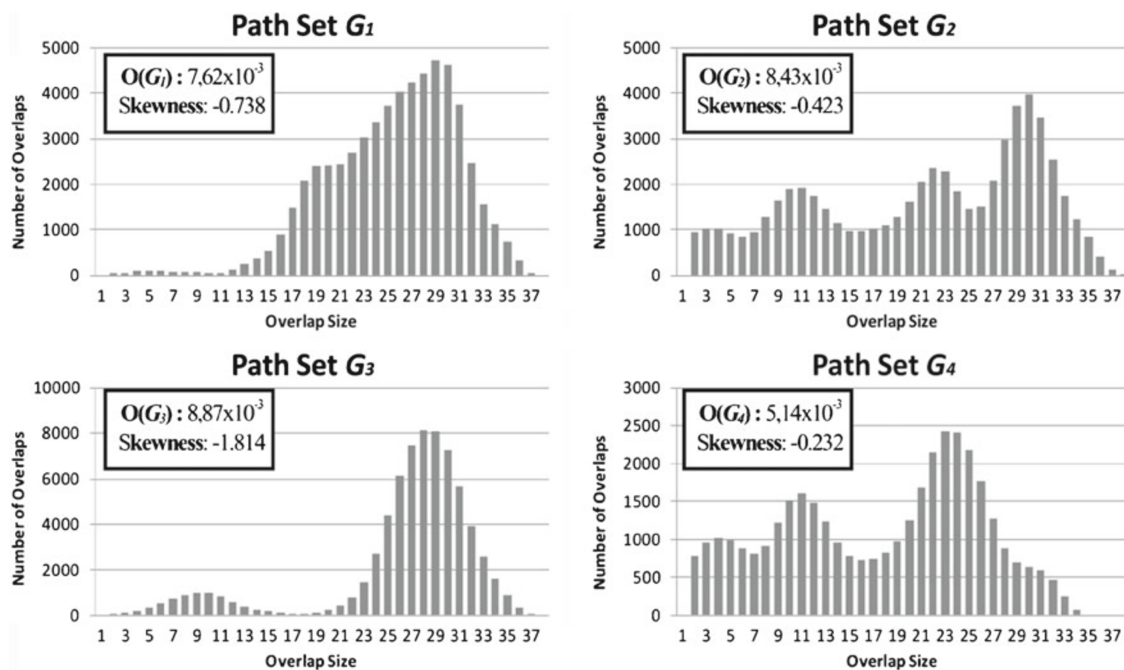

Fig. 7 Path overlap distribution for four different sets of paths of circuit s713

circuit, it gives no information on where the overlaps lay in the graph. For this reason we use the standard statistical measure of skewness which denotes if the majority of the values are situated on the left of the average (positive skewness) or on the right of the average (negative skewness). It is clear that if the desired property of the path set is to have small correlation, the skewness measure should be positive or very close to 0 (zero). For instance, observe that while sets G_2 and G_3 have similar average overlap values, their distributions are very different, and hence, their skewness values are different. The larger negative value of G_3 indicates that it contains larger overlaps than those in G_2, which is verified from the histogram. The combination of the skewness with the average overlap measure gives a very good picture of the actual histogram obtained in step 06 of the proposed algorithm (Fig. 3).

Table 3 reports the obtained statistics for all the circuits considered, for each of the four path sets G_1, G_2, G_3 and G_4 columns 1 and 2 report the circuit name and the number of paths considered in each set. Next, we report the statistics for each path set. In columns 3, 6, 9 and 12, we report the average pairwise path overlap and in columns 5, 8, 11 and 14 the values for the skewness measure. Columns 4, 7, 10 and 13 show the relative difference between the average overlap values, in order to provide a more indicative relationship between these values. The differences are calculated with respect to the smaller average overlap between the four sets. For instance, the average overlap of G_3 of circuit s38584.1 differs by 52.894 % from the corresponding value of G_2 which has the smaller overall value i.e., $(45.090 - 29.491)/29.491 = 52.8944$ %. On the other hand, the difference between G_2 and G_4 is only 8.779 %. Note that this difference is in some cases significant implying large differences in the average overlap of the considered set, even though the paths sets have exactly the same cardinality.

4.2 Evaluation of Stuck-at Test Sets for Propagation Path Overlap

In this subsection, we use as input to the proposed methodology sets that consist of fault propagation paths during test application. For this, we have used three different multiple-detect test sets for stuck-at faults, i.e., test sets that explicitly target each modeled stuck-at fault multiple (more than one) times. Multiple-detect test sets were selected for this case since they have been proven to increase the fault coverage of non-modeled defects including timing defects [8] that are tightly related to paths. We have used the multiple-detect test set of [12] for both ISCAS'85 and ISCAS'89 benchmark circuits, and the *n*-detect test sets of [11] and [14] for the ISCAS'85 and ISCAS'89, respectively. We use three different test sets for the comparison. The test sets obtained from the work in [14] are directly comparable with the method proposed in [12], yet there are not test sets available for ISCAS'85. For this purpose we have used the method of [11] to generate

Table 3 Statistics for the four different path sets obtained by the proposed method

Circuit	# of critical I/O paths	G_1			G_2			G_3			G_4		
		$O(G_1)$ ($\times 10^{-4}$)	Diff (%)	Skns	$O(G_2)$ ($\times 10^{-4}$)	Diff (%)	Skns	$O(G_3)$ ($\times 10^{-4}$)	Diff (%)	Skns	$O(G_4)$ ($\times 10^{-4}$)	Diff (%)	Skns
s641	104	316.044	7.358	0.048	469.986	59.650	0.287	298.688	1.462	0.011	294.384	0.000	0.688
s5378	360	265.609	9.570	0.014	267.453	10.331	0.000	277.949	14.661	0.001	242.410	0.000	0.127
b09_opt	372	76.692	2.392	0.281	74.901	0.000	0.365	95.817	27.925	0.454	95.290	27.222	−0.445
b12_opt	472	209.224	28.262	0.570	166.461	2.047	0.689	221.836	35.994	0.531	163.122	0.000	0.700
s3271	650	304.365	8.659	0.122	280.111	0.000	0.044	320.580	14.448	0.218	281.012	0.322	0.549
c880	682	151.840	18.093	0.433	147.502	14.719	0.242	137.515	6.951	−0.612	128.577	0.000	−0.008
s1423	726	212.248	13.080	−0.273	187.697	0.000	−0.179	275.993	47.042	−0.251	210.165	11.970	−0.356
b11_opt	1,033	524.203	13.740	0.268	460.877	0.000	0.310	726.997	57.742	0.295	564.773	22.543	0.518
s713	3,830	76.187	48.315	−0.738	84.278	64.067	−0.424	88.718	72.709	−181.843	51.368	0.000	−0.232
b07_opt	6,659	99.841	11.540	0.444	92.621	3.474	0.465	104.296	16.517	0.466	89.511	0.000	0.660
b04_opt	8,960	110.845	16.029	0.460	112.985	18.270	0.614	141.346	47.957	0.580	95.532	0.000	0.553
s38584.1	24,300	36.944	25.275	0.149	29.491	0.000	0.228	45.090	52.894	0.310	32.080	8.779	0.479
b14_1_opt	33,528	7.603	24.179	0.273	6.791	10.915	0.154	7.409	21.003	−0.008	6.123	0.000	0.330
c7552	80,640	28.852	13.137	0.107	30.023	17.730	0.012	28.252	10.788	−0.170	25.501	0.000	−0.021
c2670	116,100	5.922	5.923	0.149	5.825	4.187	0.129	7.437	33.021	0.146	5.591	0.000	0.219
c5315	138,720	25.835	0.000	0.035	27.883	7.927	0.020	33.055	27.950	0.074	27.574	6.731	0.166
c1908	161,280	6.382	4.483	0.022	6.341	3.820	−0.017	6.108	0.000	−0.049	6.264	2.551	0.098
b14_opt	175,456	1.241	1.245	0.992	1.285	4.843	0.819	1.226	0.000	0.166	1.348	9.983	0.840
b05_opt	439,103	5.719	31.182	0.348	5.118	17.391	0.266	4.360	0.000	0.200	4.967	13.932	0.376
b21_1_opt	1,949,696	0.195	0.000	0.309	0.487	149.857	0.309	0.798	309.992	0.107	0.487	149.908	0.309
b20_1_opt	1,950,650	0.815	93.894	0.335	0.462	9.946	−0.252	0.420	0.000	−0.335	0.480	14.167	−0.335
c1355	1,959,600	0.342	2.858	−0.135	0.332	0.000	−0.089	0.531	59.584	−0.161	0.342	2.812	0.160
b15_1_opt	2,209,040	0.477	31.444	0.602	0.379	4.419	0.515	0.507	39.657	0.422	0.363	0.000	0.702
c3540	4,641,360	0.420	5.723	0.107	0.397	0.000	0.154	0.416	4.637	0.112	0.401	0.906	0.141
c6288	2.13×10^{19}	0.415	8.639	0.211	0.382	0.000	0.263	0.429	12.304	0.220	0.395	3.403	0.245

Table 4 Comparing path overlaps between multiple-detect and n-detect test sets

Circuit	Multiple-detect test set [12]				n-detect test set [11]$^{\pm}$ and [14]$^{\pm\pm}$			
	# of tests	$O(G_1)$ $\times 10^{-4}$	Skns	BCE+	# of tests	$O(G_1)$ $\times 10^{-4}$	Skns	BCE+
c880	199	4.226	−0.858	0.97330	200$^{\pm}$	4.118	−0.811	0.97044
c1355	841	0.725	−0.583	0.92898	840$^{\pm}$	0.723	−0.607	0.92412
c1908	1,052	1.445	−0.397	0.96506	1070$^{\pm}$	1.521	−0.424	0.95872
c2670	538	1.085	−1.202	0.95853	550$^{\pm}$	1.089	−1.252	0.95863
c3540	1,163	1.935	−0.721	0.96127	1000$^{\pm}$	1.366	−0.994	0.96100
c7552	1,165	3.234	2.124	0.98045	780$^{\pm}$	4.125	1.956	0.97986
s208	279	42.334	0.423	0.98248	271$^{\pm\pm}$	46.221	0.370	0.95213
s298	232	28.651	0.541	0.95529	234$^{\pm\pm}$	26.772	0.571	0.99595
s344	136	28.782	−0.035	0.99627	138$^{\pm\pm}$	28.815	−0.035	0.98643
s382	252	17.096	0.140	0.98391	253$^{\pm\pm}$	17.766	0.196	0.99722
s386	209	13.400	−0.279	0.99707	201$^{\pm\pm}$	14.348	−0.231	0.94471
s420	410	14.628	−0.124	0.95167	433$^{\pm\pm}$	19.784	0.035	0.85641
s510	541	14.841	0.635	0.89850	543$^{\pm\pm}$	12.484	0.635	0.98383
s526	469	11.146	0.431	0.98386	492$^{\pm\pm}$	11.351	1.799	0.99382
s641	226	9.596	−0.475	0.99322	227$^{\pm\pm}$	13.604	−0.417	0.99125
s820	950	5.946	0.207	0.98227	949$^{\pm\pm}$	6.046	0.200	0.99959
s953	767	5.060	0.233	0.99941	766$^{\pm\pm}$	5.186	0.255	0.90938
s1196	1,233	4.200	−0.113	0.90742	1233$^{\pm\pm}$	4.391	−0.011	0.96564
s1423	269	5.001	−0.730	0.96881	269$^{\pm\pm}$	4.415	−0.684	0.97374
s1488	212	3.877	−0.278	0.96352	209$^{\pm\pm}$	4.157	−0.260	0.89162
s9234	1,190	7.129	−0.868	0.87967	1132$^{\pm\pm}$	6.577	−0.843	0.90058
s13207	2,334	9.316	−0.593	0.89989	2341$^{\pm\pm}$	9.124	−0.523	0.90784
s38417	789	18.471	−1.327	0.98811	784$^{\pm\pm}$	18.311	−1.250	0.98789

n-detect test sets that are comparable with those obtained from [12]. The experimentation consist of two steps. We first apply the test sets coloring the propagation paths for each pattern and collecting all the propagation paths in a single group of paths. Then we apply the proposed methodology and report the statistics from the obtained histograms. The overlap statistics are compared in relation to an estimation of non-modeled defect coverage obtained by the corresponding test.

Table 4 reports the statistics obtained for the different test sets generated for stuck-at faults. After the circuit name in column 1, we report results for the multiple-detect test set of [12]. Column 2 gives the size of the test set. Columns 3 and 4 report the average path overlap and the skewness measures, respectively. Column 5, reports the Bridging Fault coverage using a popular estimator called BCE+ proposed in [17]. The following four columns give the same results for the two n-detect test sets. For the ISCAS'85 the test sets are obtained using the method of [11] while for the ISCAS'89 the test sets are obtained from the work in [14]. Observe that in most cases, the test set with the largest value of the average path overlap measure ($O(G)$) gives larger defect coverage (bridging faults are used as surrogates). The test set sizes are very similar and so cannot attribute the difference in the BCE+. For example, for circuit s9234, the n-detect test set has a smaller value for the average path overlap than the multiple-detect and although its test set size is smaller, the defect coverage is higher. In cases where the value of $O(G)$ is similar for the compared test sets, like the case of s526, the skewness value resolves the issue. For s526, the n-detect test set has a larger value positive skewness indicating that the overlap sizes are smaller than the n-detect test set (closer to zero) and this could explain the larger defect coverage.

5 Conclusion

This paper presents an efficient way of identifying the pairwise path correlation between the paths in a set. A new methodology based on ZBDDs is proposed that gives a comprehensive statistical characterization for a given path set. The method is based on standard ZBDD operations of polynomial, to the size of the diagram, complexity. Experimentation using the proposed measure demonstrates its effectiveness via two different approaches. The first one shows how the proposed technique identifies similarities among various I/O critical path sets and can distinguish their characteristics based on just two measure values per test. The second one uses the proposed methodology to compare the propagation paths between multiple-detect and n-detect test sets in relation to their corresponding defect coverage.

References

1. Ababei C, Navaratnasothie S, Bazargan K, Karypis G (2002) Multi-objective circuit partitioning for cutsize and path-based delay minimization. In: IEEE/ACM international conference on computer aided design, pp 181–185. doi:10.1109/ICCAD.2002.1167532
2. Christou K, Michael M, Neophytou S (2010) Identification of critical primitive path delay faults without any path enumeration. In: IEEE VLSI test symposium, pp 9–14. doi:10.1109/VTS.2010.5469629
3. Huang SY, Cheng KT (1999) Errortracer: design error diagnosis based on fault simulation techniques. IEEE Trans Comput-Aided Des Integr Circuits Syst 18(9):1341–1352. doi:10.1109/43.784125
4. Kim KS, Mitra S, Ryan P (2003) Delay defect characteristics and testing strategies. IEEE Des Test Comput 20(5):8–16. doi:10.1109/MDT.2003.1232251
5. Krstic A, Jiang YM, Cheng KT (2001) Pattern generation for delay testing and dynamic timing analysis considering power-supply noise effects. IEEE Trans Comput-Aided Des Integr Circuits Syst 20(3):416–425. doi:10.1109/43.913759
6. Kumar M, Tragoudas S (2007) High-quality transition fault ATPG for small delay defects. IEEE Trans Comput-Aided Des Integr Circuits Syst 26(5):983–989. doi:10.1109/TCAD.2006.884863
7. Lin X, Tsai KH, Wang C, Kassab M, Rajski J, Kobayashi T, Klingenberg R, Sato Y, Hamada S, Aikyo T (2006) Timing-aware ATPG for high quality at-speed testing of small delay defects. In: Proceedings of the 15th Asian test symposium, ATS '06. IEEE Computer Society, Washington, DC, pp 139–146. doi:10.1109/ATS.2006.81
8. McCluskey E, Tseng CW (2000) Stuck-fault tests vs. actual defects. In: Proceedings. International test conference, pp 336–342. doi:10.1109/TEST.2000.894222
9. Minato S (1993) Zero-suppressed bdds for set manipulation in combinatorial problems. In: Design automation conference, pp 272–277. doi:10.1109/DAC.1993.203958
10. Minato S (1996) Binary decision diagrams and applications for VLSI CAD, vol 342. Springer
11. Neophytou S, Michael M (2007) Hierarchical fault compatibility identification for test generation with a small number of specified bits. In: IEEE international symposium on defect and fault-tolerance in VLSI systems, pp 439–447. doi:10.1109/DFT.2007.46
12. Neophytou S, Michael M, Christou K (2009) Generating diverse test sets for multiple fault detections based on fault cone partitioning. In: IEEE international symposium on defect and fault tolerance in VLSI systems, pp 401–409. doi:10.1109/DFT.2009.24
13. Padmanaban S, Michael M, Tragoudas S (2003) Exact path delay fault coverage with fundamental ZBDD operations. IEEE Trans Comput-Aided Des Integr Circuits Syst 22(3):305–316. doi:10.1109/TCAD.2002.807891
14. Pomeranz I, Reddy S (2007) Forming N-detection test sets without test generation. ACM Trans Des Autom Electron Syst (TODAES) 12(2):18
15. Smith G (1985) Model for delay faults based upon paths. In: Proc. of ITC, pp 342–349
16. Somenzi F (2005) CUDD: CU decision diagram package release 2.4.1. University of Colorado at Boulder
17. Tang H, Chen G, Reddy S, Wang C, Rajski J, Pomeranz I (2005) Defect aware test patterns. In: Proceedings of design, automation and test in Europe, vol 1, pp 450–455. doi:10.1109/DATE.2005.110
18. Tani S, Teramoto M, Fukazawa T, Matsuhiro K (1998) Efficient path selection for delay testing based on partial path evaluation. In: Proceedings. 16th IEEE VLSI test symposium, pp 188–193. doi:10.1109/VTEST.1998.670867
19. Tayade R, Abraham JA (2009) Critical path selection for delay testing considering coupling noise. J Electron Test Theory Appl 25(4–5):213–223. doi:10.1007/s10836-009-5105-7
20. Tayade R, Sundereswaran S, Abraham J (2007) Small-delay defect detection in the presence of process variations. In: 8th international symposium on quality electronic design. ISQED '07, pp 711–716. doi:10.1109/ISQED.2007.145
21. Waicukauski J, Lindbloom E (1989) Failure diagnosis of structured vlsi. IEEE Des Test Comput 6:49–60. doi:10.1109/54.32421
22. Yilmaz M, Chakrabarty K, Tehranipoor M (2010) Test-pattern selection for screening small-delay defects in very-deep submicrometer integrated circuits. IEEE Trans Comput-Aided Des Integr Circuits Syst 29(5):760–773. doi:10.1109/TCAD.2010.2043591
23. Zolotov V, Xiong J, Fatemi H, Visweswariah C (2010) Statistical path selection for at-speed test. IEEE Trans Comput-Aided Des Integr Circuits Syst 29(5):749–759. doi:10.1109/TCAD.2010.2043570

Stelios N. Neophytou received the Engineering diploma from the Computer Engineering and Informatics Department of University of Patras, Patras, Greece, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus, in 2003 and 2009, respectively.

Currently, he is a lecturer with the Electrical and Computer Engineering Department, University of Nicosia, Nicosia, Cyprus. His research concentrates on the area of electronic design and testing. His research interests include design for testability, algorithms for high quality testing, high quality fault models and online testing.

Kyriakos Christou holds a B.Sc. in computer science from University of Cyprus (2001), an MSc in Computer and Information Science from the University of Pennsylvania (2003), and a Ph.D. from Electrical and Computer Engineering Department at the University of Cyprus, (2012). His research interests include high-quality fault models, algorithms for high-quality testing, formal/semi-formal methods for both software/hardware verification and test.

Maria K. Michael received the B.S. and M.S. degrees in computer science and the Ph.D. degree in electrical and computer engineering from Southern Illinois University, Carbondale, in 1996, 1998, and 2002, respectively.

She taught as a lecturer with the Electrical and Computer Engineering Department, Southern Illinois University from 2001 to 2002, and as an assistant professor of computer science and engineering with the University of Notre Dame from 2002 to 2003. She is currently an assistant professor with the Electrical and Computer Engineering Department, University of Cyprus, Nicosia, Cyprus. Her research interests include test and fault diagnosis automation, semi-formal methods for logic and timing verification, symbolic techniques for test and verification (BDDs and SAT), design-for-testability, and fault tolerance and reliability of digital circuits.