A NOVEL FUNCTIONAL TESTING AND VERIFICATION TECHNIQUE FOR LOGIC CIRCUITS

Hussain Al-Asaad, Ganesh Valliappan, and Lourdes Ramirez
Department of Electrical & Computer Engineering
University of California
Davis, CA, U.S.A.

ABSTRACT

Functional verification plays a key role in the design verification cycle and the physical fault testing process. There are several functional verification methods that generate tests for modules independent of their implementation; however, these methods do not scale well for medium to large circuits. In this paper we introduce a new implementation-independent functional test generation technique that extracts a good set of functional vectors that are characterized by a small number of neighbors. Two input vectors of a function are considered neighbors if they produce the same output value of the function and the Hamming distance between them is one. Our method can be easily implemented and it generates tests by selecting input vectors that have fewer neighbors among all input vectors. Our experimental results demonstrate that our generated tests are significantly better than random tests. Moreover, our method can handle multiple-output circuits, and can be easily scaled to target large designs.

Keywords: simulation, functional testing/verification, logic circuits, implementation-independent testing.

1 INTRODUCTION

In the past decade, the research on testing and verification focused on designs with known implementations. Unfortunately, the implementation details are sometimes left unknown to the designer; the designer is given only a functional description of the component, making it difficult to verify or test. Consequently, the need for methods that can handle circuits in terms of a functional model is increasing especially for intellectual property circuits (IP), cores, and virtual components [6].

The complexity of digital systems continues to rise and the need for verification advances with it. It is known that design verification can account for more than half of the design cycle [10]. The process of design verification attempts to guarantee that the implementation meets the designer's original specifications with no design errors. Simulation-based verification uses input test vectors to verify that the design possesses the correct implementation. This verification technique is the most commonly used in industry today [9]. Functional verification is a simulation-based method that probes into the behavior of a design rather than its implementation.

Manufacturing Testing is also becoming more difficult with increased design complexity. The cost of fault-oriented test generation is rising and functional testing is becoming more attractive. It has been found in [4] that functional testing can achieve about as close to the coverage achieved by gate-level testing within a margin of about 4 percent.

This paper focuses on functional test generation methods for components independent of their implementation. This is both useful for design verification and manufacturing testing. The design is considered as a black box, but the functional behavior of the design is given. The goal is to develop a method that will obtain a small test set size, good physical fault and design error coverage, and low overhead. The paper is organized as follows: Section 2 gives an overview of previous work for functional verification and testing. In Section 3, our new functional verification method is introduced and described in greater detail. Section 4 discusses the simulation results obtained for various circuits. We then summarize our contributions and propose future improvements in Section 5.

2 BACKGROUND

Deterministic test generation is an effective test generation method that targets specific faults by incorporating the knowledge of the design implementation. Academic deterministic test generators are available, such as ATALANTA [7], that are able to efficiently create compact test sets that provide good fault coverage. Because of the added complexity in the deterministic algorithm, the amount of time that it takes to generate tests may take longer than other methods, but the fault coverage in many cases is greater. Since specific faults are targeted the test set size generally is smaller than random methods.

Random test generation can be an efficient method to produce tests. The implementation of a design is

not a factor when using random test generation. Techniques used for test generation do not require that every aspect of the design be known. For the DEC Alpha 21264 processor, which is a highly out-of-order, superpipelined processor, the chip verification team used pseudo-random test generation as part of the verification process. Because the design was so complex, using pseudo-random test generation allowed the verification engineers to focus on more crucial areas of the design. The percentage of bugs detected by pseudo-random tests resulted in 79 percent of total bugs [11]. This shows that random test generation does achieve a medium level of coverage.

Several functional test generation methods have been proposed. In [4], Akers presented a method that creates a Universal Test Set (UTS) for a logic network. The generated tests are independent of the implementation and target stuck-at-faults in the logic implementation of AND/OR trees. In his study, Akers shows that for AND/OR networks a UTS can be generated that detects not only all single stuck-at faults, but all multiple stuck-at faults as well. In generating a UTS, Akers suggests that each vector be compared bit-by-bit to another vector. Depending on the comparison between the bits, the vector is either added to the test set or dropped. This method can be easily implemented, however, it has two main disadvantages: (i) enumerating all input combinations to generate a UTS is not feasible for large circuits and (ii) the UTS may contain excessive number of tests, sometimes all input combinations.

Abadir and Reghbati [1] and later Chang et al. [5] developed functional test generation methods based on Binary Decision Diagrams (BDDs) [3]. The behavior of the design is represented using BDDs and a fault model is introduced that changes the structure of BDDs. BDDs were used for test generation because of their relative implementation independence, computational ease, and simple data structures. Their research demonstrated that comprehensive tests can be produced and the test set can be further simplified if some structural details are known [5]. The generated test sets provided perfect fault coverage for stuck-at faults in several implementations. The major disadvantages of functional verification with BDDs is its inability to obtain good coverage for lower level faults and that it cannot be applied to large circuits.

In the next section, we introduce our method, Boundary Testing (BT). The method lies somewhere in between deterministic test generation and random test generation in terms of its test set size and coverage. Deterministic test generation may obtain smaller test sets than BT, but may take more CPU time. Deterministic test generation needs the design details while BT does not. BT achieves smaller test sets and better coverage than random test generation as we demonstrate later in the paper.

3 NEW METHOD

The new BT method is advantageous when details of the design are not given to the designer. The ease of generating tests makes it an ideal method for testing and verification of combinational circuits. It focuses on the functional description of a component rather than its structural implementation.

The input vectors of a single-output function *z* are partitioned into the on-set $\{\alpha_0, \alpha_1,....\}$, where $z = 1$, and the off-set $\{\beta_0, \beta_1,...\}$, where $z = 0$. An input combination is denoted as α_i when it is in the on-set and β_i when it is in the off-set. An *n*-bit input vector α_i (or β_i) is a neighbor to another *n*-bit vector α_j (or β_j) if the Hamming distance between the vectors is one. Therefore to obtain a neighbor of an *n*-bit input vector α_i (or β_i) one bit is complemented. Each *n*-bit vector α_i (or β_i) can have from zero up to *n*-neighbors. Finally, when comparing vectors to one another, *N* denotes the number of neighbors that a vector has.

The focus of BT is the number of neighbors *N* for a particular vector. We hypothesize that a vector with fewer neighbors should detect more physical faults and design errors than a vector with more neighbors. Consequently, selecting input vectors with fewer neighbors should achieve better fault/error coverage.

In general the simplest way to obtain a BT is to create a Karnaugh-map representation of the component. Once a K-map is created it is easy to determine *N* for each vector. The test set is comprised of vectors with the least number of neighbors for both α and β. We illustrate the test generation process for BT using the majority circuit shown in Figure 1. The number of neighbors *N* are shown as the superscripts of the on-set (α) and off-set (β) values. In this case input vectors {000,111} both have three neighbors. The remaining input vectors {001, 010, 011, 100, 101, 110} have only one neighbor. After comparing vectors {011, 101, 110, 111} that are in the on-set, vector {111} is eliminated. Likewise {000} is eliminated from the vectors that are in the off-set. The vectors that remain in α and β are combined to form the test set {001, 010, 011, 100, 101, 110}. These vectors

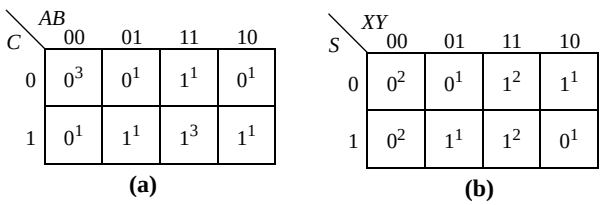

Figure 1 The K-maps for (a) a majority circuit and (b) a multiplexer circuit.

detect all the stuck-at-faults for the majority circuit, which results in complete coverage for this circuit. In this example, it can be easily seen which vectors should be in the test set. However, in more complex circuits, the process can be more complicated. Defining a cut-off can become more involved when the distribution of *N* is spread out. The general conditions for determining which vectors are in the test set are summarized below.

- If $N_{\alpha i} > N_\alpha$ then α_i can be eliminated, where α_i represents an input combination in the on-set and N_α is the cut-off parameter for the on-set.
- If $N_{\beta i} > N_\beta$ then β_i can be eliminated, where β_i represents an input combination in the off-set and N_β is the cut-off parameter for the off-set.

In the majority circuit example the cut-off parameters are $N_\alpha = 1$ and $N_\beta = 1$. Therefore after checking the vectors in the on-set and the off-set, all vectors with one neighbor or fewer are included in the test set. Setting the cut-off parameters to three will include all input vectors and clearly it is not the best test set possible.

Another example of applying BT to a multiplexer circuit is shown in Figure 1. In this example the α vectors are {011, 110, 111, 100} and the β vectors are {000, 001, 010, 101}. For the α vectors, the vectors with the least number of neighbors are {011, 100}. In β the vectors that remain in the test set are {010, 101}. Combining the α vectors and β vectors results in vectors {011, 100, 010, 101}. Again BT has obtained complete (100%) stuck-at fault coverage of the circuit using a minimum test set.

Achieving small test sets as well as good coverage is important in test generation. To achieve small test sets the BT algorithm must be modified. In the small examples of Figure 1 it was apparent which test vectors should be included in the test set, but it is not clear how to extract the test vectors when complex components are targeted. The distribution of neighbors was very conveniently clustered into groups in the previous examples. For larger circuits, the distribution may not be as well defined, therefore the cut-off parameters need to be determined. We next describe a concrete method that defines the cut-off parameters for simple single-output circuits. Larger single-output circuits are also explored. Finally multiple-output circuits are discussed.

Single-output circuits: Once it was discovered that BT generated good test sets for simple circuits, selecting good cut-off parameters becomes an important goal. Good cut-off parameters will ensure that an excessive number of redundant vectors will be avoided. To determine the best cut-off parameters, several simple circuits were explored. Different combinations of tests were used to explore where the best values for the cut-off parameters. For the circuit with the K-map shown in Figure 2(a), we simulated the test sets resulted from BT using every combination of N_α and N_β. For this circuit, the vectors in the on-set had only one or two neighbors, but the off-set had vectors with one, two, three or four neighbors. In the table shown in Figure 2(c), the columns and rows correspond to the various values of the cut-off parameters. For example, the stuck-at coverage of the test set in (row 1, column 1) corresponds to cut-off $N_\alpha = 1$ and $N_\beta = 1$ and hence the test set includes all vectors that have one neighbor. The coverage that those vectors achieve is 66.7%. The table shows the fault coverage of the circuit for each combination of the cut-off parameters N_α and N_β. In this example, there is a limit where *N* for α and β achieves a maximum of 100% fault coverage; adding more vectors does not change its coverage. In previous examples, the cut-off parameters were the same. This example shows that they need not be always the same. The optimal cut-off for α is one and for β is three. Therefore all vectors in α with one neighbor and all vectors in β with three or fewer neighbor are included in the test set.

The cut-off parameters must be determined for α and β separately because the distribution of α's and β's in a function will alter their cut-off independently. In the table shown in Figure 2(b), the first column is the number of neighbors *N* and the second and third columns are the corresponding number of vectors that have *N* neighbors. For example, the first row shows that four vectors in α and one vector in β have one neighbor. One way to define the cut-off parameters as the weighted average among the vectors of the on-set and those of the off-set and then rounded up to the nearest integer. So, we get $N_{\alpha ave} = 2$ and $N_{\beta ave} = 3$ for the circuit in Figure 2.

We have applied BT with an average for the cut-off parameters to several circuits and we have obtained complete stuck-at fault coverage. Our BT method is described in the flowchart shown in Figure 3. The method works well for small components. When components become too complex, an alternative way is to combine random test generation with

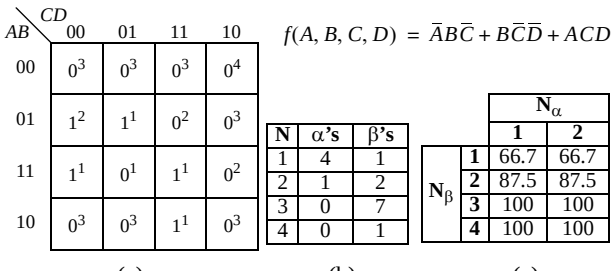

Figure 2 A four-input circuit: (a) K-map, (b) vector distribution, and (c) fault coverage.

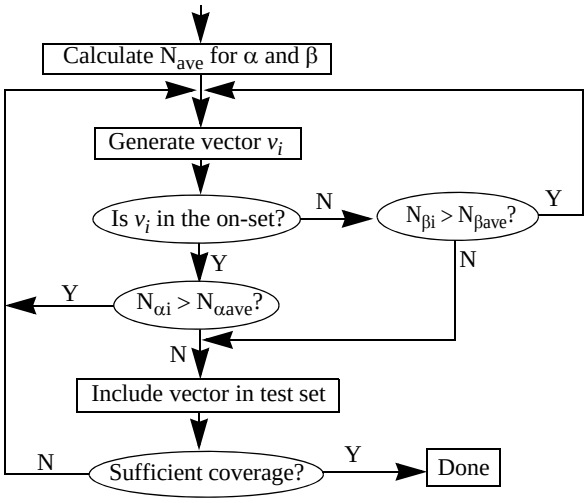

Figure 3 Flowchart of BT for single-output circuits using an average.

the algorithm. The BT described previously focused on small circuits where the number of neighbors can be easily identified using K-maps. The next step was to generate test vectors for larger circuits where it is often tedious to identify the number of neighbors for all input combinations. Therefore instead of mapping out a circuit and calculating N_{ave}, the test generation process is simplified. To generate test sets for larger input circuits the same principle was followed, the least number of neighbors for a given test vector provides better coverage than one with more neighbors. The difference in generating a test set was, instead of computing N_{ave} for the α and β, to randomly generate a vector and replace it with one of its neighbors that has fewer neighbors. Random vectors are generated until the test set reaches the required fault coverage. This can be described as follows:

- For a test set V and a random vector $v \in V$, v can be replaced by a neighbor v_i if v_i has less neighbors than v.

A tree diagram can be used to determine if a vector need to be replaced. Figure 4(a) shows a typical tree diagram for a single-output circuit with *n*-inputs. The first vector v is randomly generated and is the root of the tree. All neighbors of v are traversed and checked for fewer neighbors. The value in parenthesis indicates the output value in response to the test vector.

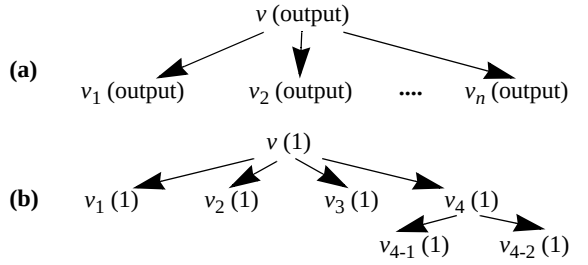

Figure 4 Tree diagram for single-output circuits: (a) general case and (b) an example.

We applied the randomized BT method described above for a sixteen-input two-level circuit with the function $= \bar{A}BP + FG\bar{E} + \bar{I}HJP + DCKM + \bar{N}LA$. To illustrate the test generation process, the circuit was first simulated with {1010011100111000} as the first randomly generated vector, v. This test vector has fourteen neighbors and is in α. Of these fourteen neighbors only four vectors {v_1 = 1010011100111100, v_2 = 1010011100101000, v_3 = 0010011100111000, v_4 = 1010111100111000} have fewer neighbors (thirteen neighbors) than v. These are better choices for the test set than v since they have fewer neighbors. Shown in Figure 4(b) is the tree diagram for v. Each vector with thirteen neighbors, {v_1, v_2, v_3, v_4}, obtained a coverage of at least 10.345%, whereas v obtained a coverage of only 6.897%. This was verified by simulating the circuit using FSIM [8] to determine the coverage of each vector. We further investigated whether vectors that had more neighbors compared to v would result in less fault coverage. An input combination with sixteen neighbors resulted in a coverage of 3.448%, which verifies our premise.

The above search for a better vector in the tree looked at nodes up to a *depth* of 1. The next step that was taken was to continue investigating the neighbors of the newly found test set by looking in the tree up to a depth of 2. In the above example, test vectors {v_1, v_2, v_3} each had children with thirteen or more neighbors, which indicates that none of them could be replaced. On the other hand v_4 had two children with twelve neighbors therefore v_4 could be removed and replaced by either $v_{4\text{-}1}$ = 1010101100111000 or $v_{4\text{-}2}$ = 1010110100111000. These two vectors had coverage of at least 13.793%, whereas v_4 resulted in 10.345%. This process can be repeated in the tree for a depth of 3 or more until no more vectors are found that has less number of neighbors than any vector in the test set. The question that is encountered while generating the test set is, what should be done when there are several test vectors with fewer neighbors? The example before traversed through each neighboring vector, but when there are too many to choose from it is difficult to determine what route to take since different routes in the tree can give you better or worse results. To make it easy to generate test sets it would be best to specify the maximum depth allowed during the search of the tree. Once all the paths have been traversed the vector with the least number of neighbors is chosen. If there are many vectors with the same number of neighbors then any one of them should be added to the test set. Adding all of them may result in too many redundant vectors.

Finally when all the paths are explored, one vector (the one with the least number of neighbors) can replace the randomly chosen vector, *v*. If no vector has fewer neighbors than *v*, then *v* is kept in the test set and is not replaced. In the example above, if the maximum depth allowed is 2, then $v_{4\text{-}1}$ or $v_{4\text{-}2}$ will replace *v* since it has less *neighbors* than any of the other vectors $\{v_1 \ldots v_4\}$. After *v* is exhausted another random vector is generated and the same process is repeated. Random vectors are generated until the required coverage is met.

An interesting question is what is maximum depth allowed to be used during the search process of the tree diagram. There is a trade-off here between the quality of the vector and the time needed to compute the good vector. Although we may obtain better vectors with larger depth, the computational needs become more expensive.

Multiple-output circuits: In dealing with multiple outputs, the exact flow that was implemented for single-output components cannot be followed since each input vector will not necessarily produce the same value for all outputs. The single output case could be used if tests were generated for each output independently, but may result in an excessive number of uninteresting test vectors.

In order to effectively handle multiple-output circuits, we need to further refine our neighbor definition. We classify the following two types of neighborhood:

- *Complete*: An *n*-bit input vector is a neighbor to another *n*-bit vector if the Hamming distance between the vectors is one and *all* the outputs of the circuit are identical.
- *Partial*: An *n*-bit input vector is a neighbor to another *n*-bit vector if the Hamming distance between the vectors is one and *some* of the outputs of the circuit are identical. This definition of a neighbor is less strict and hence it leads to searching a larger input vector space.

A *weight* must be defined for each input vector of a multiple-output circuit. The weight is a measure of the vector quality and it is used in the selection process of good vectors. It is obvious that the weight of a vector *v* for the case of complete neighborhood is the number of neighbors of *v*. For partial neighborhood, we consider the following methods:

- *M1*: The weight is defined as the average of the number of neighbors among the matching outputs.
- *M2*: The weight of a vector is the number of matching output bits amongst all the neighbors divided by the number of locations in which there is a matched bit.

In general the flowchart shown in Figure 5 is executed for multiple-output circuits for the search depth of 1. Note that the weight computation of vectors in the replace function in the flowchart is different for complete and partial neighborhoods. Similar flowcharts can be used for larger search depths.

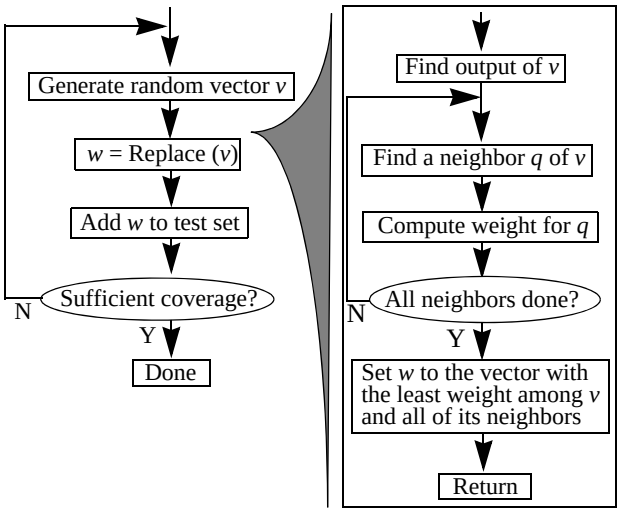

Figure 5 Flowchart of BT for multiple-output circuits.

4 SIMULATION RESULTS

In this section, BT is compared to random test generation since both of these methods can be used for implementation-independent components.

Single-output circuits: In finding a test set for single-output circuits, it was shown that a cut-off could be determined from calculating an average. This worked well, achieving good coverage of the circuit. The test set was minimal when dealing with small circuits and the BT method is relatively simple to implement. The method became more complex when dealing with larger circuits, often producing large number of vectors.

The BT algorithm was modified to generate tests for larger circuits as follows. First, random vectors are generated and then any random vector that has a neighbor with fewer neighbors is replaced. This method is quite easy to implement and works well. Although redundant vectors are introduced a high coverage can be achieved. Shown in Table 1 are the simulation results for several small four-input circuits. As demonstrated by the table, BT achieves bet-

Table 1 Results for 4-input single-output circuits.

Circuit	Random		BT	
	Coverage	Test set size	Coverage	Test set size
Ex1	100	15	100	10
Ex2	87.5	10	95.83	10
Ex3	82.76	10	96.55	10
Ex4	86.96	9	100	9
Ex5	100	14	100	10
Ex6	97.30	15	97.30	9
Ex7	100	11	100	8

ter or, in some cases, equal coverage compared to random test generation. Moreover, the test set size is less than or equal to that of random test generation. Hence, BT obtains better results when compared to random test generation. It is interesting to note that in Ex2 and Ex3 only two vectors are replaced and in Ex4, Ex5, and Ex7 only one vector is replaced by using BT. So few vector substitutions are made and as a result BT is able to achieve nearly 10% better coverage than random and the test set is reduced to about 35% in some cases.

Table 2 shows the simulation results for larger circuits. The same response that was seen in the small input cases carried on to the larger circuits. The test set size is greatly reduced by BT. In the first two components, there is a reduction of about 50% in the number of test vectors and for the carry out (c4 of a 4-bit ripple carry adder) component the savings is 12%. The coverage for random test generation and BT is the same for the MUX and carry out components, but the 16-input circuit achieves 17.2% increased coverage with BT. For these components, BT achieves equal or better coverage and less test vectors than random test generation.

Multiple-output circuits: For the multiple-output circuits, the procedure shown in Figure 5 is followed. We first applied the method M1 described in the previous section to small components. The method worked well for the components that were tested. Shown in Table 3 are two different components, the adder and the adder-subtractor. Each component was simulated using different implementations. The first version of the adder-subtractor uses ripple-carry logic and the second version uses carry-lookahead logic. Here again we see similar results as in the single-output case. Both methods, random and BT, achieve 100% coverage for the circuits considered. The difference here is that the test set for BT is always smaller. For the first three components in Table 3, BT is able to reduce the test set size by about 35%. In the last component, BT reduces the test set size considerably by 75%. Again the results indicate that BT is able to significantly reduce the test set size.

Table 2 **Results for single-output components with large number of inputs.**

Circuit	Random		BT	
	Coverage	Test set size	Coverage	Test set size
8-to-1 Multiplexer	100	165	100	72
16-input circuit	79.31	>600	96.55	267
Adder-c4	100	93	100	82

Table 3 **Results for multiple-output components using method M1.**

Circuit	Random		BT	
	Coverage	Test set size	Coverage	Test set size
ripple carry adder	100	28	100	18
carry-lookahead adder	100	64	100	42
adder-subtractor RC	100	28	100	17
adder-subtractor CL	100	93	100	24

In order to illustrate the effectiveness of BT for multiple-outputs on large circuits, a C++ CAD tool was developed. The input to the tool is the netlist file of the benchmark, the number of random vectors and the search depth. The tool generates the test set for the circuit and gives the outputs in terms of 2 sets of files: the original vector set and the improved vector set using BT. FSIM [8] was used to find the fault coverage of the generated tests.

The ISCAS-85 netlist files were used as input to the tool. Data was collected for 5 to 70 vectors being applied on the circuit. Moreover, the data was collected for varying search depths on some circuits. It should be noted that the obtained data for various depths for the same circuit were a result of different runs and hence the initial vectors considered were different in each case. So, the random vectors obtained for depth 1 and those of depth 3 should not be compared based on their fault coverage values.

We first look at the experimental results obtained when a complete neighborhood is enforced. We look closely at the ISCAS-85 benchmark c1908. The c1908 circuit consists of 33 inputs and 25 outputs. The tool was run on the circuit and better vectors captured at every interval of 5 from 5 to 70 random vectors for depths of 1 and 3. The graphs showing the fault coverage as a function of the number of vectors for both depths are shown in Figure 6. After applying BT on the circuit for a depth of 1, the fault coverage gain was as high as 4.5%. However, for a depth of 3, the gain in fault coverage was noted to be a maximum of 11.76% and had an average gain of 9.88% as compared to 2.74% for depth 1. In other words to achieve a fault coverage of say 70%, using BT for depth = 1 we need only around 23 vectors but normally would have needed 28 vectors. For depth = 3 and for a fault coverage of 70%, we would have needed 48 vectors with the random test set but 14 vectors using BT.

It is evident from the graphs that BT has increased the fault coverage. Also, as the depth is increased, we see an increase in the fault coverage gain. This may not be valid for all depth as after some depth, all the possible input vectors would have been covered.

The experimental results for all the ISCAS-85 benchmark circuits show that BT performed up to 5% better than random test generation for all circuits. The improvement is not significant sometimes due to the fact that vectors with complete neighborhood are often rare in these ISCAS-85 circuits. So, we experimented using the method M2 where a partial neighborhood is assumed. The results obtained by BT show an improvement of up to 12% in fault coverage.

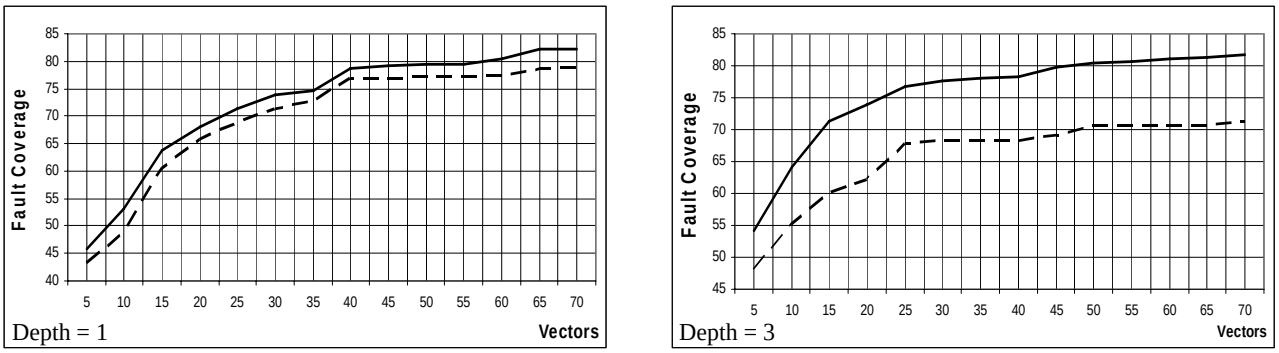

Figure 6 Fault coverage as a function of the number of vectors for the circuit c1908.

5 DISCUSSION

BT has proven to have several good qualities. It generates test vectors that achieve good fault and design error coverage. The cut-off parameter is introduced and implemented using an average for the on-set and off-set. Random test generation was incorporated to BT and showed to give good results for large single-output circuits. To handle multiple-output circuits, we introduced a modified BT process that achieves good fault and design error coverage. In summary, the characteristics of BT are:

- It generates up to 35% fewer vectors in a test set when compared to random test generation.
- It achieves up to 55% better coverage compared to random test generation.
- In BT, a vector that is replaced by another with fewer neighbors often achieves better coverage.
- It can be applied to testing or design verification.
- It is implementation independent.
- It is easy to be implemented.

BT has shown to generate good test sets, but can be greatly improved with slight modifications. Targeting specific faults can improve the performance of BT and hence achieve a higher degree of coverage and smaller test sets.

Moreover, combining BT with deterministic test generation is an attractive option. BT can be used until a desired coverage was achieved then the remaining faults could be targeted using deterministic test generation.

One of the weaknesses of BT is its dependence on random vectors. By locating "good" vectors (vectors with fewer neighbors) the test set would benefit significantly. More research is needed to determine the good seeds.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0092867.

REFERENCES

[1] M. S. Abadir and H. K. Reghbati, "Functional test generation for digital circuits described using binary decision diagrams", *IEEE Transactions on Computers*, Vol. C-35, pp. 375-379, April 1986.

[2] S. B. Akers, "Universal test sets for logic networks", *IEEE Transactions on Computers*, Vol. C22, pp. 835-39, September 1973.

[3] S. B. Akers, "Functional testing with binary decision diagrams", *Proc. International Conference on Fault Tolerant Computing*, 1978, pp. 75-82.

[4] S. A. Al-Arian, "Functional level ATPG and fault coverage", *Proc. SOUTHEASTCON*, 1991, pp.104-108.

[5] H. P. Chang, W. A. Rogers, and J. A. Abraham, "Structured functional level test generation using binary decision diagrams", *Proc. International Test Conference*, 1986, pp. 97-104.

[6] H. Kim and J. P. Hayes, "Realization-independent ATPG for designs with unimplemented blocks", *IEEE Transactions on Computer-Aided Design*, Vol. 20, pp. 290-306, February 2001.

[7] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits", Dept. of Electrical Engineering, Virginia Polytechnic Institute and State University, Rep. 12-93, 1993.

[8] H. K. Lee and D. S. Ha, "An efficient forward fault simulation algorithm based on parallel pattern single fault propagation", *Proc. International Test Conference*, pp. 946-55, 1991.

[9] D. Moundanos and J. A. Abraham, "Abstraction techniques for validation coverage analysis and test generation", *IEEE Transactions on Computers*, Vol. 47, pp. 2-14, January 1998.

[10] C. Pixley et al., "Commercial design verification: methodology and tools", *Proc. International Test Conference*, pp. 839-48, 1996.

[11] S. Taylor et al., "Functional verification of multiple-issue, out-of-order, superscalar alpha processor-The DEC Alpha 21264 microprocessor", *Proc. Design Automation Conference*, 1998, pp. 638-43.