IMPROVING COVERAGE OF CIRCUITS BY USING DIFFERENT FAULT MODELS COMPLEMENTING EACH OTHER

by

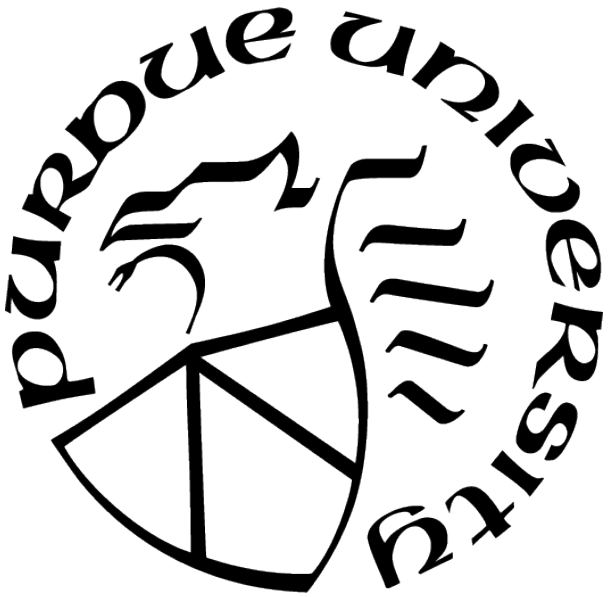
Oindree basu

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science



School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Irith Pomeranz, Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Cheng-Kok Koh

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

To my parents and friends who are always there for me

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to Prof. Irith Pomeranz, my advisor, for her constant guidance and support throughout my Masters, in spite of a global pandemic giving rise to adverse circumstances. Without her prompt feedback and constant encouragement, this work would not have been possible.

I would also like to thank my lab mate Hari Narayana Addepalli for providing help whenever I asked for it.

I am extremely grateful to Prof. Anand Raghunathan and Prof. Cheng-Kok Koh for being on my advisory committee.

Finally, I would like to express my heartfelt thanks to my family and friends both in India and in the United States, for always believing in me and motivating me throughout this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

DFM	Design-for-Manufacturability
ATPG	Automatic Test Pattern Generation
SSF	Single Stuck-at Fault

ABSTRACT

Various fault models such as stuck-at, transition, bridging have been developed to better model possible defects in manufactured chips. However, over the years as device sizes have shrunk, the probability of systematic defects occurring in chips has increased. To predict the sites of occurrence of such defects, Design-for-Manufacturability (DFM) guidelines have been established, the violations of which are modelled into DFM faults. Nonetheless, some faults corresponding to DFM as well as other fault models are undetectable, i.e., tests cannot be generated to detect their presence. It has been seen that undetectable faults usually tend to cluster together, leaving large areas in a circuit uncovered. As a result, defects occurring there, even if detectable, go undetected because there are no tests covering those areas. Hence, this becomes an important issue to address, and to resolve it, we utilize gate-exhaustive faults to cover these areas. Gate-exhaustive faults provide exhaustive coverage to gates. They can detect any defect which is not modelled by any other fault model. However, the total number of gate-exhaustive faults in a circuit can be quite large and may require many test patterns for detection. Therefore, we use procedures to select only those faults which can provide additional coverage to the sites of undetectable faults. We define parameters that determine whether a gate associated with one or more undetectable faults is covered or not, depending on the number of detectable and useful gate-exhaustive faults present around the gate. Bridging faults are also added for extra coverage. These procedures applied to benchmark circuits are used for obtaining the experimental results. The results show that the sizes of clusters of undetectable faults are reduced, upon the addition of gate-exhaustive faults to the fault set, both in the case of single-cycle and two-cycle faults.

1. INTRODUCTION

The detection of defects occurring in a chip constitutes a significant and vital step in the chip manufacturing process. Defects can be in the form of bridges and opens among others. To effectively detect different kinds of defects, various fault models have been developed to represent the physical defects as logical faults. Efforts have been made to ensure most defects are accounted for, so that they can be detected if they occur. Apart from modelling possible defects in a chip, fault models also make the test generation problem easier to handle, by creating specific targets for which test patterns can be generated. Once a particular fault related to any fault model is defined, a test for detecting the fault can be generated by a commercial test generation tool.

With changing technologies, new defects have emerged in fabricated chips. The scaling of devices over the last few years has increased the complexity of circuits to large extents. On one hand, the decreasing transistor sizes has complicated functionality and significantly increased frequency of operation. On the other hand, this has resulted in more variations, and consequently, the possibility of systematic defects occurring due to the manufacturing process has increased [1], [2], [3] . The defects-parts-per-million (DPPM) has taken a hit. It has therefore, become important to generate test patterns specifically targeting defects occurring as a result of imperfections in the silicon during chip manufacture. To that extent, Design-for-Manufacturability (DFM) guidelines have been laid down, the violations of which are considered to be sites of possible defects. These defects are then modelled into faults referred to as DFM faults [4], and tests are generated for their detection. According to the defect they are modelling, DFM faults may pertain to various fault models.

The difficulty in covering a circuit arises due to the fact that tests cannot be generated for every fault in the circuit. There are some faults which are undetectable, i.e. a tool cannot generate tests for detecting the presence of those faults. A fault in a circuit may be undetectable due to logic redundancy, resulting in the erroneous value at the fault site not being propagated to a point under observation. For activating a fault, a test pattern applies specific input patterns at a gate, and undetectable faults sometimes occur due to it being impossible to apply the required input pattern at the gate. Thus, defects at the site of an

undetectable fault may not have specific tests for their detection. They might accidentally get detected by another pattern, however, there is a chance that they might go undetected altogether. Therefore, it is always beneficial to have test patterns covering as much of the circuit as possible, so that defects in the actual chip do not go undetected.

In our work, we attempt to solve the problem of inadequate coverage of circuits by using different fault models complementing each other. Faults corresponding to DFM, stuck-at [5] and transition [6] fault models are used to directly generate tests for the circuit. The undetectable faults from these fault models leave uncovered areas. These areas, also referred to as holes in the circuit, are sites which do not have specific test patterns covering them. Hence, gate-exhaustive faults [7] around the holes are used to generate test patterns which provide extra coverage for these sites. Utilizing all these fault models and tests together, we develop a procedure which aims at giving as much coverage to the circuits as possible, so that defects in chips do not go undetected.

The chapter *Literature Review* explains and reviews previous work on different fault models, generation and detection of DFM faults, clustering of undetectable faults in a circuit, and gate-exhaustive faults. The next chapter, *Methodology*, explains the procedure for finding holes in circuits and subsequently covering them with gate-exhaustive faults, in detail. It also presents few examples to better demonstrate the methods used for covering the sites of undetectable faults. The fourth chapter *Results and Discussion* shows the data obtained from carrying out our procedure on benchmark circuits, and provides justification for the experimental results. Finally, *Conclusion* concludes the thesis with comments about the effectiveness of the procedure in decreasing the size of uncovered areas in a circuit.

2. LITERATURE REVIEW

2.1 Fault Models

Physical faults that could occur in a chip are modelled as logical faults representing their effects. This makes fault analysis feasible and makes testing technology-independent. A single fault model is not enough to encompass all kinds of defects that might be observed on a chip. Over the years several fault models have been developed to represent the actual defects more and more accurately. Some of them are listed below.

2.1.1 Single Stuck Fault Model

The single stuck fault (SSF) model is the first fault model that was studied and used to generate tests. It models a defect as a single line stuck at value *0* or *1*. The rest of the circuit functions as it is. The concept of a single line stuck at a value is fairly simple and can represent many different physical defects [5]. The number of single stuck faults in a circuit is relatively small as well. SSFs are modelled on inputs and outputs of a cell and each fan-out branch is considered a separate line. The total number of faults in the circuit is therefore, twice the number of lines (each line can be stuck at either 0 or 1).

A fault is detected by generating a test pattern for it. If a line l is stuck at a, the fault is denoted by l/a. A test for the fault l/a activates it by assigning value $\bar{a}$ to l. Since l is stuck at a, it does not take up the value $\bar{a}$, and an error is generated. The test pattern ensures that the error is propagated to an observable point p by making all lines in a path between l and p have faulty values. This path is said to be sensitized to fault l/a.

2.1.2 Transition Fault Model

Transition faults model defects that cause delayed transitions in the circuit. A line is said to have a transition fault when there is a more than acceptable delay in the time taken for it to change its value from high to low or from low to high. Similar to stuck-at faults, they too are modelled on the inputs and outputs of a cell. Two types of transition faults are

mentioned in [6], slow-to rise and slow-to fall. The slow-to-rise faults behave like stuck-at 0 faults while the slow-to-fall ones behave like stuck-at 1 faults.

Since transition faults model delays in transitions on a line, they require two-pattern tests for their detection [6]. The first pattern initializes the fault line with a logic value while the second pattern creates a transition (assigns 1 if the initialization pattern assigned 0 and vice-versa) on the line. If this transition is delayed for more than the desired time interval, an error is generated. The second pattern then propagates this error to an observable point and the transition fault is detected.

Two types of tests widely used to detect transition faults are skewed-load and broadside tests. In skewed-load tests ([8],[9]), the second test pattern is obtained by shifting the first pattern by one bit through the scan chain. Broadside tests [10], on the other hand, scan in only the first pattern and apply the response of the combinational circuit as the second pattern. Fig 2.1 and Fig 2.2 illustrate the timing diagrams of the two types of tests. Skewed-load tests are fast and effective, have a low complexity, and have been experimentally seen to provide significant coverage in benchmark circuits [9]. The broadside method, on the other hand generates limited tests and cannot guarantee a minimum coverage. However, it eases the restriction on scan enable signal timing. The scan enable signal, when high, activates the scan chain to function as a shift register. The signal does not need to be a critical signal in broadside tests as it is not activated when the second pattern is generated, unlike in skewed-load tests. Both types of tests are included in an effective transition test set.

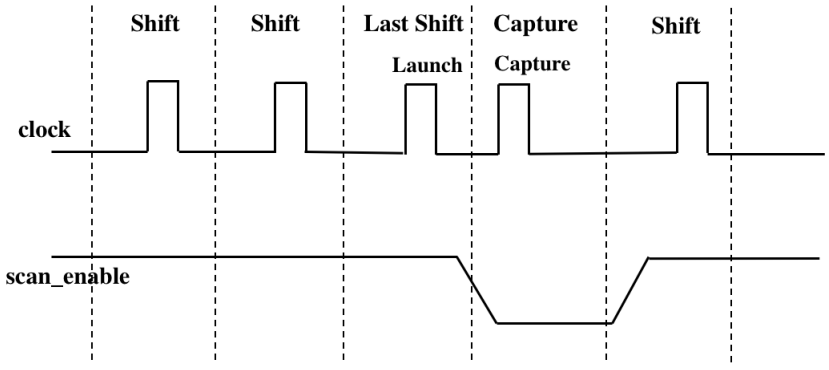


Figure 2.1. Timing Waveform for Skewed-Load Tests

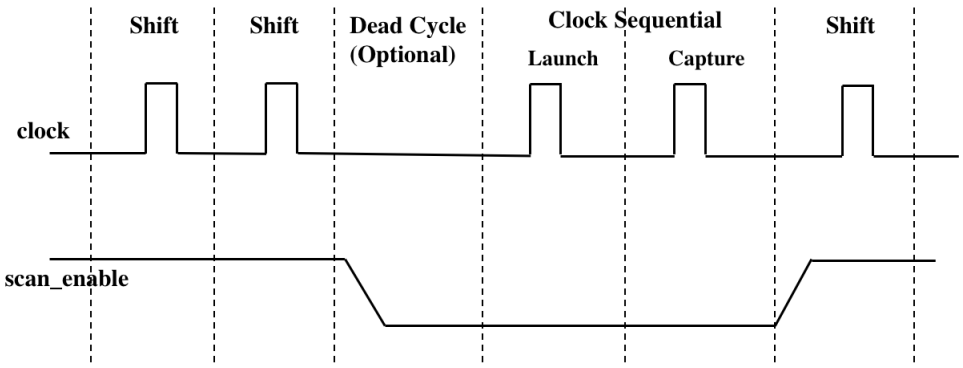


Figure 2.2. Timing Waveform for Broadside Tests

2.1.3 Bridging Fault Model

Bridging faults occur when two lines, unconnected in the fault free circuit, are accidentally shorted together. One way of modelling this connection is by considering that wired logic is performed at the connection such as wired AND or wired OR [11]. Another way is by considering that one of the two connected nets is the dominant net and the other is the follower net. As a result of the short, the follower net is driven to the logic value of the dominant net. This is the 4-way bridging fault model described in [12]. In a fault between two nets, the following four cases arise:

- Net 1 is dominant and has value 0, Net 2 is driven to value 0
- Net 1 is dominant and has value 1, Net 2 is driven to value 1
- Net 2 is dominant and has value 0, Net 1 is driven to value 0
- Net 2 is dominant and has value 1, Net 1 is driven to value 1

In each case, the follower net is considered to be stuck-at the value of the dominant net. Hence, the bridging faults can be detected in the same manner as stuck-at faults. For example, if there is a bridging fault corresponding to case 1, the fault is reduced to Net 2 being stuck-at 0, with the additional condition that Net 1 is also set to 0.

2.1.4 Cell-Aware Fault Model

The stuck-at, transition and bridging fault models described above consider faults only on the inputs and outputs of cells. They are referred to as external faults. Test patterns based on these fault models often tend to miss some defects internal to a cell. This approach, hence, becomes inadequate to keep up with an increasing demand to lower the defect rates. The cell-aware fault model was thus developed, to address defects internal to the cell explicitly. Experimental results with industrial designs in [13] show that the defect coverage of cell-aware test patterns is significantly better than stuck-at and transition fault patterns.

The methodology of converting an internal defect into a cell-aware fault is described in [14]. The paper proposes a layout extraction of the netlist and possible defects, followed by an analog simulation to obtain cell-input combinations for activating a particular defect. It also includes a synthesis step to create an optimized library containing multiple input patterns for each internal defect. The final step involves generating test patterns based on the input conditions stored in the library for each defect.

The cell-aware fault model defines input patterns on a cell that would excite a certain internal defect and produce a logical error on the output of the cell. The test for the defect assigns required values on the input lines of the cell and propagates the faulty output value to an observable point for fault detection. Tests for cell-aware faults can be single-cycle for static internal faults, or two-cycle for delay internal faults [15]. The two-cycle tests consist of two pattern tests similar to the tests used to detect transition faults.

2.2 Design-For-Manufacturability

With designs becoming more complex and process technology decreasing to sizes lesser than available wavelengths, the number of systematic defects is increasing rapidly. It is recommended, therefore, to follow some manufacturability guidelines that reduce the probability of defects occurring due to the manufacturing process. Design-for-Manufacturability (DFM) rules consist of such guidelines. They differ from design rules in that they need not be strictly followed and are applied when possible, in conformation with area and power con-

straints. The results reported in [16] show that correcting DFM guideline violations result in significant increase in yield.

Fixing all DFM violations is impossible in an actual layout. Moreover, it is usually not possible to know the fabrication issues in advance due to design constraints and the decreasing time to market. Therefore, there is an attempt to have tests for detecting defects that could result from DFM guideline violations. Some potential causes for systematic defects such as non-redundant *p-diffusion contact* and *single via* are defined in [4], and a procedure to identify affected transistors and translate the defects into gate level logic faults using switch-level simulation is described. The paper shows that targeting potential systematic defect sites resulting from DFM guidelines could improve the quality of tests significantly. A more extensive list of DFM guidelines categorized into four sub-groups, namely Via, Metal, Poly and Density is specified in [17]. This paper also briefly elucidates how the guideline violations are converted into faults conforming to stuck-at, transition and bridging fault models. Targeting cell internal DFM violations with cell-aware faults is proposed in [18]. The paper describes steps for obtaining cell input patterns for activating potential internal defects and generating tests for them using an ATPG (Automatic Test Pattern Generation) tool.

2.3 Clustering

There exist some faults, internal and external, for which no tests can be generated. These faults are said to be undetectable. Undetectable faults might occur for various reasons, one of which is logic redundancy. In such a case, it is not possible to propagate the faulty value on a line to a point of observation. Previous work has shown that undetectable faults tend to cluster together in the circuit. It was shown in [19] that for benchmark circuits, undetectable single stuck faults formed clusters in certain areas. Later, it was reasoned in [20] that an undetectable transition fault would definitely exist at the site of an undetectable single stuck fault. Furthermore, there would be some two-cycle skewed-load and broadside tests which could not be applied to the circuit resulting in more undetectable transition faults. An increased number of faults not being detected would mean bigger clusters forming in case of

transition faults as compared to single stuck faults. Similar clustering was observed in [21] for undetectable DFM faults. The faults were found to be concentrated in a few areas of the circuits.

Large clusters of undetected faults leave uncovered areas in the circuit, which lead to other detrimental effects. A defect in such an uncovered area might be detectable even though it is surrounded by undetectable faults. There is a possibility that tests don't exist for the particular defect in the test set, so it has a chance of going undetected. Moreover, it has been seen from [22] how defects at the site of undetectable faults invalidate tests for otherwise detectable faults, thus amplifying the negative effects of leaving uncovered areas. Also, faults resulting from DFM violations may not always accurately model an actual defect. In such cases, an undetectable fault leaves a potential detectable systematic defect uncovered. The above reasons provide the motivation behind attempts at covering test holes and reducing the sizes of undetectable fault clusters in a circuit.

2.4 Gate-Exhaustive Fault Model

First mentioned in [7], a gate-exhaustive test set was defined as one which applied all possible input patterns to cells, and observed the faulty response at a primary output. This approach ensured that any unexpected internal defect or a defect not modelled by any other fault set was accounted for.

Gate-exhaustive faults are similar to cell-aware faults in that they are defined as input patterns of a cell. The only difference is that these faults are not tied to any particular internal defect, rather, all input patterns are considered, and any of them are expected to produce a faulty value on the output line. Such an exhaustive test set is generated in an attempt to increase the defect coverage. It was demonstrated in [23], [24] and [25] that gate-exhaustive tests are more effective in detecting defective chips as compared to test sets from other fault models such as stuck-at and transition.

Since gate-exhaustive faults related to a particular gate include all possible input patterns that could be applied to the gate, the number of gate-exhaustive faults grows exponentially with the number of inputs and so do the number of tests for detecting them. Considering

all the faults at once is thus not feasible. Various papers have proposed the idea of utilizing a subset of the gate-exhaustive faults to generate tests for improving the fault coverage of a circuit. In [26], an iterative procedure for test generation of gate-exhaustive faults in benchmark circuits is described, to provide additional coverage to sites of undetectable stuck-at faults. For covering each undetectable fault, gate-exhaustive faults are added according to specified conditions, and the number is increased step-wise. The process is continued until a coverage metric for the fault is met. A more recent work on gate-exhaustive faults [27] proposes a Multiple Target Test Generation procedure for gate-exhaustive faults using Partial MaxSAT, which is a variation of the MaxSAT problem. It generates test patterns that can detect the maximum number of target gate-exhaustive faults by utilizing the Partial MaxSAT algorithm.

3. METHODOLOGY

In this chapter, we describe the method by which we attempt to utilize different fault models to reduce coverage holes in a circuit. The chapter is divided into three major sections. Section 3.1 describes clustering, the parameter we use to determine the coverage of a circuit. Section 3.2 explains gate-exhaustive faults, and how we select them to cover undetectable faults. In Section 3.3 the complete algorithm applied to benchmark circuits is elucidated step-wise.

3.1 Clustering

It was mentioned in the previous chapter that undetectable faults tend to cluster together in a circuit, leaving large areas uncovered. Defects occurring at these sites might be detectable even if the faults are not, hence it becomes detrimental to leave a significant area in a circuit uncovered. We intend to break these clusters of undetected faults by adding faults to the fault set and checking the coverage around the clusters, and subsequently generating additional tests where required. In the rest of this section, we provide our definition of adjacent gates and describe how we use it to form a cluster of gates.

Suppose we consider a fault set FS that includes faults corresponding to any of the previously discussed fault models, and apply ATPG to it to generate tests for fault detection. As a result, a test set T is generated and the set of undetectable faults U is obtained. Each fault in U corresponds to a gate. In case of an undetectable stuck-at, transition or bridging fault, it corresponds to the gate on whose input or output nets the fault is present. An undetectable cell-aware fault is related to the gate whose internal defect it addresses.

We define any two gates G1 and G2 as adjacent if there is a net connecting the two gates. This occurs in any of the following three cases:

- G1 and G2 each have one or more inputs that are fan-out branches of the same stem
- The output of G1 is connected to an input of G2
- An input of G1 is driven by the output of G2

If any one of the conditions is satisfied, the gates G1 and G2 are considered adjacent to each other. A cluster is formed when multiple gates are adjacent to each other. For example, if G1 and G2 are adjacent to each other, and G3 is adjacent to either G1 or G2 or both, G1, G2 and G3 form a cluster.

We obtain the set of gates G associated with the set of undetected faults U. Clusters are formed out of G to determine how large an area is left uncovered in the circuit. The procedure for obtaining clusters from a set of gates was described in [19], and has been explained here to provide a detailed representation. Initially each gate is a separate cluster C_i. If the gates in C_i and C_j are adjacent to each other, the two are merged together to form a cluster C_i, and C_j is removed. The process is repeated until there is no change in the remaining clusters, and therefore, the largest clusters have been obtained. The pseudo-code for forming clusters out a set of gates G is shown in Algorithm 1.

Algorithm 1: Procedure for forming Clusters out of Gates

```
Obtain set of gates G corresponding to undetected faults;
for every gate g_i in G do
    C_i = g_i ;
    add C_i to C [C is the set of clusters] ;
end
do
    for every C_i in C do
        if gate g_i in C_i is adjacent to g_j in C_j [j != i] then
            C_i = C_i + C_j ;
            remove C_j ;
        end
    end
while there is any change in C;
```

It is important to break the large clusters of gates associated with undetected faults. Therefore, faults are added to the fault set, to check for coverage, as well as to improve it when not sufficient, by adding tests. Additional test patterns generated for these faults provide direct coverage to the uncovered sites and increase the chances of detecting defects around those areas. As more and more tests are added and coverage is checked, gates associated with undetected faults which qualify as covered are removed from the large clusters, and consequently the clusters are broken into smaller groups.

Table 3.1. : Single Cycle Input Patterns for NAND Gate

A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.2. : Two-Cycle Input Patterns for NAND Gate

A	**B**	**Y**
01	01	10
01	11	10
11	01	10
10	10	01
10	11	01
11	10	01

3.2 Selection of Gate Exhaustive Faults for Coverage

There might be detectable defects in and around the clusters of undetectable faults in a circuit. Therefore, it is preferred to have a test set that does not leave the cluster sites

uncovered. To check for coverage at the undetectable fault sites and close test holes, we use gate exhaustive faults.

As described in the previous chapter, gate-exhaustive faults apply all possible input patterns on a gate, and propagate the erroneous value at the output of the gate to an observable point. The methods used to select gate-exhaustive faults for covering the sites of undetectable faults corresponding to different fault models is explained in this section, citing the example of a two-input NAND gate. Single-cycle gate-exhaustive faults of a NAND gate include the input patterns listed in Table 3.1 and two-cycle gate-exhaustive faults include those listed in Table 3.2. Columns A and B represent the two inputs of the gate. Column Y represents the output of the gate in fault free condition. In a faulty circuit with single-cycle gate-exhaustive faults, for a fault in Table 3.1, a test pattern assigns the values in columns A and B to the input nets of the gate, and propagates the faulty output value to an observable point. In case of a circuit with two-cycle gate-exhaustive faults, the test pattern for a fault assigns two values in two cycles to each input net of the gate, as shown in Table 3.2. The transition, which would have been brought about as a result of the change in input pattern, is delayed due to the fault, and the error value is transmitted to an observable point by the test.

The selection methods described in the following paragraphs are with respect to a two-input NAND gate.

3.2.1 Stuck-at

Let us consider the undetectable fault A stuck at 0. We need to assign the value A = 1 to excite the fault. The gate-exhaustive faults with input patterns that assign A = 1, namely the last two in Table 3.1, are added to the fault set and used to generate tests for covering the undetectable fault.

In general, if the input of a gate is stuck at a, the faults with input patterns assigning $\bar{a}$ to the input can be utilized to generate tests to provide additional coverage to the gate. If the undetectable stuck-at fault is on the output of a gate, the faulty value cannot be propagated

to an observable point due to logic redundancy. Hence, it is not possible to provide coverage even with gate-exhaustive faults.

3.2.2 Transition

The coverage of undetectable transition faults with gate-exhaustive faults is similar to stuck-at faults, except we use two-cycle faults for this. If input A has a slow-to-rise transition fault, a gate-exhaustive fault that causes A to make a low to high transition (0→1) is added to the fault set.

In our example of NAND gate, if A has a slow-to-rise transition fault, the first two input patterns listed in Table 3.2 are added to the fault set to generate tests.

3.2.3 Bridging

We use the 4-way bridging fault model in our work which labels one of the connected nets as dominant and the other as follower. The dominant net has a specific value, and this value is taken up by the follower net. Therefore, the follower net acts as if it is stuck at the value of the dominant net. Hence, for covering bridging faults with gate-exhaustive faults, we use the same procedure as for stuck-at faults. We add those gate-exhaustive faults to the fault set which have input patterns that activate the error at the follower net, and generate tests for them.

3.2.4 Cell-Aware

Cell-Aware faults consist of single cycle and two-cycle faults. They are modelled just like gate-exhaustive faults with a particular input pattern being assigned to a gate to activate a fault. The only difference is that in cell-aware faults, an input pattern is expected to excite a particular defect within the gate, whereas gate-exhaustive faults consider all possible input patterns. If a cell-aware fault is undetectable, it is either because it is impossible to assign the particular input pattern to the gate or because the faulty output value cannot be propagated to an observable point. Naturally, the gate-exhaustive fault with the same input pattern as

the undetectable cell-aware fault is also undetectable. Hence, we search for faults with input patterns in the logical neighborhood of the undetectable fault pattern.

Let us consider an undetectable single cycle cell aware fault which assigns A=0, B=0 as inputs of the NAND gate. Clearly, the gate-exhaustive fault with the same input pattern will also be undetectable. Hence, we consider those faults which are logically near the input pattern '00', i.e., '01' and '10'. In other words, we use those gate-exhaustive faults to generate tests whose input pattern differs by one bit from that of the undetectable fault. We try to cover the fault site by incorporating tests which excite all but one of the inputs of the gate in the same way as the original fault attempted to.

For two-cycle cell-aware faults, we use a similar procedure to obtain additional gate-exhaustive faults to generate tests. Suppose the undetectable fault has A=10, B=10 as the input pattern, indicating that inputs transitioning to 0 is the fault detection condition. Since, the gate-exhaustive fault with the same input pattern is undetectable, we add those faults which have inputs in the logical neighborhood of the undetectable fault pattern. Thus, the gate-exhaustive faults having input patterns A=10, B=11 and A=11, B=10 are added to the fault set for test generation.

If the faults with inputs in the near neighborhood of the undetectable fault pattern do not provide the required coverage, we use gate-exhaustive faults whose input patterns differ by more than one bit from those in the undetectable fault. We progressively move farther away from the logical proximity, if necessary, and if the faults exist, to provide adequate coverage to the particular gate. An example of the same is shown below.

Example: 1

Circuit: b15

Gate - U6566

Faults associated with the gate – 13 internal static faults

Undetectable faults associated with the gate – 4

No. of detectable gate-exhaustive faults required to cover the gate ($for\ k = 2$) – $4 * 2 = 8$

No. of detectable gate-exhaustive faults with one bit input pattern change – 6

No. of detectable gate-exhaustive faults with two bit input pattern change – 4

The above example presents the data for gate U6566 in benchmark circuit b15. The gate has thirteen internal static DFM faults associated with it, out of which four are undetectable. As described in Section 3.3, the number of gate-exhaustive faults required to cover a gate is k * *No. of undetectable faults*. For $k = 2$, the number of gate-exhaustive faults needed is eight. However, the number of detectable gate-exhaustive faults whose input patterns differ from the fault pattern by one bit is only six. Hence, we use two of the four detectable gate-exhaustive faults whose input patterns differ by two bits to provide the required coverage.

If the input pattern of a particular gate-exhaustive fault differs by more than one bit from the pattern of undetectable fault U_1 and by one bit from that of U_2, (U_1 and U_2 being internal faults of the same gate), the fault will be counted only once while calculating the number of gate-exhaustive faults added for that gate.

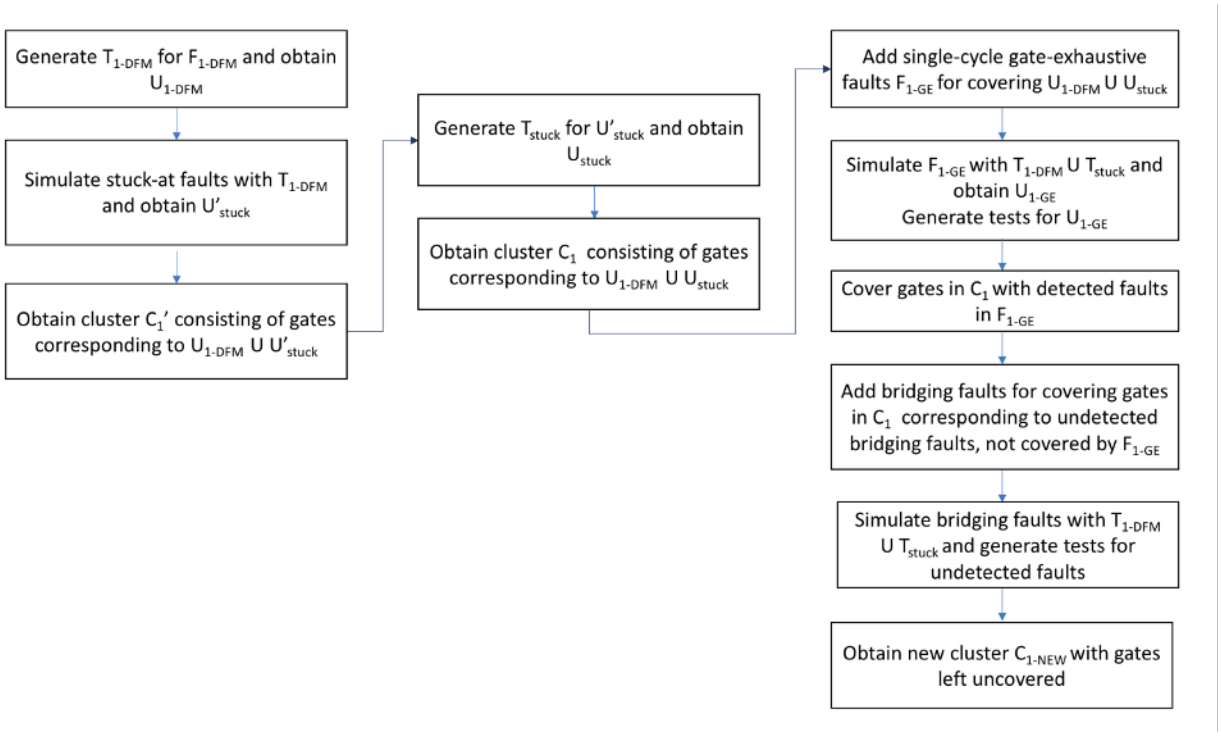


Figure 3.1. Algorithm for improving Coverage of a circuit for Single-cycle Faults

3.3 Procedure for Improving Coverage

For improving the coverage of a circuit, we use different fault models to complement each other. The procedure is carried out separately for single-cycle faults (stuck-at, bridging and single-cycle cell-aware and gate-exhaustive) and two-cycle faults (transition and two-cycle cell-aware and gate-exhaustive).

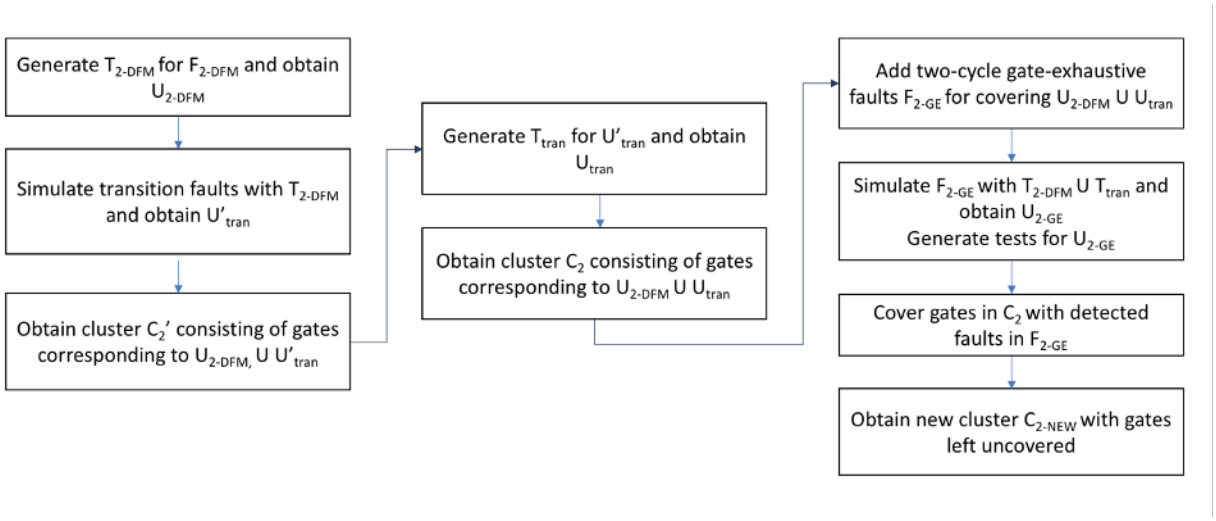


Figure 3.2. Algorithm for improving Coverage of a circuit for Two-cycle Faults

In the process of improving coverage, we consider the set of DFM, stuck-at and transition faults as the target set. We generate tests for these faults and obtain the undetectable faults from the ATPG process. The gate-exhaustive faults are then utilized to cover those areas left uncovered by the already generated test set. The overview of the procedure is outlined in Figure 3.1 and Figure 3.2 for single-cycle and two-cycle faults respectively. The detailed steps are elucidated below.

Step-I

The initial set of target faults are the DFM faults denoted by F_{DFM}. F_{DFM} is obtained by converting DFM violations in the layout of the circuit into faults, and it includes stuck-at, transition, bridging and cell-aware faults. The set of single-cycle faults consisting of stuck-at, bridging and single-cycle cell-aware faults is denoted by F_{1-DFM}, and the set of two-cycle

faults consisting of transition and two-cycle cell-aware faults is denoted by F_{2-DFM}. A single-cycle test set T_{1-DFM} is generated using ATPG. From this test generation procedure, a set of faults U_{1-DFM} is obtained, consisting of undetectable faults and faults for which the test generation procedure was aborted during ATPG. The set of all stuck-at faults in the circuit is then simulated with T_{1-DFM} and a new set of undetected faults is obtained, consisting of $U_{1-DFM} \cup U'_{stuck}$. A set of clusters C'_1, of gates associated with these undetected faults is formed, using the definition of adjacency stated in Section 3.1. The size of the largest cluster in C'_1 gives us an estimate of the area left uncovered by the test set T_{1-DFM}.

Step-II

To expand the test set further, patterns are generated for the undetected stuck-at faults in U'_{stuck}. The newly generated test set is denoted by T_{stuck}. T_{stuck} detects those stuck-at faults which are detectable but were left undetected by the DFM test set T_{1-DFM}. As a result of adding these tests, a new set of undetectable faults U_{stuck} is obtained. U_{stuck} is a subset of U'_{stuck}. A new set of clusters C_1 is created out of gates associated with faults in the set $U_{1-DFM} \cup U_{stuck}$.

Step-III

The large size of clusters at this point indicate that significant portions of the circuits are not yet covered. Hence, we use gate-exhaustive faults for providing the added coverage. For each stuck-at, bridging and single-cycle cell-aware fault in $U_{1-DFM} \cup U_{stuck}$, additional faults are added as described in Section 3.2. The set of single-cycle gate-exhaustive faults added for coverage is denoted by F_{1-GE}. F_{1-GE} is first simulated with the existing test set $T_{1-DFM} \cup T_{stuck}$. ATPG is carried out for the faults remaining undetected after simulation for test generation.

The procedure elucidated in *Steps I, II and III* are for single-cycle faults. Two-cycle faults are handled in a similar way. In *Step I*, transition faults are simulated with T_{2-DFM}, and $U_{2-DFM} \cup U'_{tran}$ is obtained. New tests are generated for the undetected transition faults in the circuit in *Step II*. The set of undetectable faults after this step is denoted by $U_{2-DFM} \cup U_{tran}$, and the set of clusters formed from these faults is denoted by C_2. The

undetectable transition and internal delay faults in $U_{2-DFM} \cup U_{tran}$ are then covered with additional gate-exhaustive faults F_{2-GE}. These added faults are first simulated with the existing test set $T_{2-DFM} \cup T_{tran}$, and then ATPG is carried out for test generation.

The number of detectable and undetectable gate-exhaustive faults obtained after the test generation procedure is noted, and is used to determine which gates should be removed from the cluster of gates corresponding to undetectable faults. The parameters used to classify a gate as covered or uncovered are described below.

Let us suppose that gate G has N_{Undet} number of undetectable faults associated with it after *Step II*. We add N_{GE} number of gate-exhaustive faults in *Step III* for the N_{Undet} undetectable faults, out of which N_{GE-Det} are detectable. If $N_{Undet} * k <= N_{GE-Det}$, then the gate is considered covered and removed from the appropriate cluster (single-cycle or two-cycle). k is a multiplicative factor that relates the number of undetectable faults associated with a gate and the number of additional detectable faults required to cover the gate. The value of k determines the target number of detectable gate-exhaustive faults required for coverage.

Some gates associated with undetectable bridging faults are not covered according to the conditions stated above. Their coverage is complemented by adding extra bridging faults to the fault set. Suppose the undetectable bridging fault is between dominant net N1 with value a and follower net N2. Additional bridging faults are considered between the dominant net N1 with value a and nets adjacent to N2. These faults are first simulated with $T_{1-DFM} \cup T_{stuck}$, and then tests are generated for the undetected faults after simulation. If $N_{Undet} * k <= N_{Det}$, N_{Det} being the total number of detectable gate-exhaustive and bridging faults added for the gate corresponding to the fault, it is removed from the cluster. An example where additional bridging faults along with gate-exhaustive faults give coverage to a gate is demonstrated below.

Example: 2

Circuit: b15

Gate – U5173

Faults associated with the gate – 3 4-way bridging faults

Undetectable bridging faults with gate as follower – 1

No. of detectable faults required to cover the gate ($for\ k = 2$) – $1 * 2 = 2$

No. of detectable gate-exhaustive faults – 1

No. of detectable bridging faults added – 1

The above example presents data for gate U5173 of circuit b15. The gate has three 4-way bridging faults associated with it out of which one fault with net U5173/Y as the follower is undetectable. To cover the undetectable fault with $k = 2$, we need two detectable gate-exhaustive faults. However, we have only one such fault, hence we try to cover it with a bridging fault. We add one additional bridging fault to the fault set to obtain the required coverage.

While adding gate-exhaustive faults to cover undetectable internal faults for a gate, it may happen that the number of faults required for coverage, i.e., $N_{Undet} * k$ do not exist. In such a case, we mark the gate as covered if all the gate-exhaustive faults that are selected to be added are detectable. An example of such a case is shown below.

Example: 3

Circuit: b15

Gate – FE_OFC55_CLOCK

Undetectable faults associated with the gate – 1

No. of detectable gate-exhaustive faults required to cover the gate ($for\ k = 2$) – $1 * 2 = 2$

Total no. of gate-exhaustive faults possible to be added –1

No. of detectable gate-exhaustive faults – 1

The above example presents data for gate FE_OFC55_CLOCK of circuit b15. The gate has one undetectable fault associated with it, and we require two additional faults to cover

the gate when the value of k is 2. The number of faults possible to be added is one, and it is detectable, hence we designate the gate as covered, as the two required faults do not exist.

New clusters C_{1-NEW} and C_{2-NEW} are formed from the remaining gates which are not covered even after the addition of gate-exhaustive faults. The sizes of these clusters are noted, and found to be significantly smaller than C_1 and C_2.

Among the gate-exhaustive faults added for coverage, some are detected by patterns from existing test sets ($T_{1-DFM} \cup T_{stuck}$ for single-cycle faults, and $T_{2-DFM} \cup T_{tran}$ for two-cycle faults), and some need additional patterns to be generated through ATPG. Hence, there are two ways in which we can choose gate-exhaustive faults to provide coverage:

- The number of additional tests generated can be maximized by selecting those gate-exhaustive faults which are not detected by existing test sets, where possible. This provides extra coverage to the circuit.
- The gate-exhaustive faults for additional coverage can be selected from among those that are detected by existing test sets, where possible. This ensures that the necessary coverage of the circuit is achieved without increasing the number of tests more than required.

We have implemented the second method while presenting results in the following chapter. The examples below demonstrate two cases where the added gate-exhaustive faults are detected by the existing test sets and hence, do not require additional patterns to be generated.

Example: 4

Circuit: s38584

Gate – U10374

Number of undetectable single-cycle faults associated with the gate – 3

Number of detectable gate-exhaustive faults required for coverage – 3 ($for\ k = 1$), 6 ($for\ k = 2$)

There exist 6 gate-exhaustive faults detected by $T_{1-DFM} \cup T_{stuck}$

Example: 5

Circuit: s38584

Gate – U11944

Number of undetectable two-cycle faults associated with the gate – 1 (transition fault at U11944/B)

Number of detectable gate-exhaustive faults required for coverage = 1 ($for\ k = 1$), 2 ($for\ k = 2$)

There exist 2 gate-exhaustive faults detected by $T_{2-DFM} \cup T_{tran}$

The above examples demonstrate cases where the required coverage is provided by gate-exhaustive faults which are detected by the existing test sets. Example 4 exhibits an instance for single-cycle faults and Example 5 exhibits that for two-cycle faults. In both the examples, all faults required for the cases $k = 1$ and $k = 2$ are detected by existing patterns and do not require additional tests to be generated.

4. RESULTS AND DISCUSSION

The procedure for improving the coverage of a circuit by adding gate-exhaustive and bridging faults at the sites of undetectable faults is described in the previous chapter. The algorithm as demonstrated in Figure 3.1 and Figure 3.2 is applied to benchmark circuits. The analysis for single-cycle faults and two-cycle faults is carried out separately.

Commercial tools are used for identifying DFM guideline violations and for test generation. We have used Calibre for detecting DFM violations in a layout. Encounter has been used to generate layout from a netlist. Fastscan has been used to carry out ATPG for all fault sets, and consequently to identify undetectable faults.

Two values of parameter k have been used for our analysis, $k = 1$ (results demonstrated in Tables 4.1 and 4.2), and $k = 2$ (results demonstrated in Tables 4.3 and 4.4). The required coverage for each gate associated with undetectable faults changes with the value of k. With $k = 2$, the number of detectable faults required for coverage is twice than that with $k = 1$.

For Tables 4.1 and 4.2, and Tables 4.3 and 4.4, the first column *Benchmark Circuits* lists the benchmark circuits to which our algorithm was applied. The data obtained from the evaluation of each circuit is presented in three rows corresponding to each circuit. The first row shows results related to DFM tests, according to the procedure described in *Step-I* in Section 3.3. The second row adds stuck-at fault tests (in Tables 4.1 and 4.2) and transition fault tests (in Tables 4.3 and 4.4) to single-cycle and two-cycle DFM fault tests respectively (*Step-II*). The third row presents the data after gate-exhaustive faults (for single-cycle and two-cycle faults) and bridging faults (only for single-cycle faults) have been added, for covering the sites of undetectable faults, and tests have been generated for them, in each circuit (*Step-III*).

The third column *Single/Two cycle Test Patterns* lists the total number of test patterns utilized for detecting the faults considered in each row. The number of new test patterns added in each row is the difference between the value in that row and the previous row. Column four, *Gates corresponding to Undetected Faults* shows the number of gates associated with undetected faults in each particular row. The fifth column *Largest Cluster Size* lists the

sizes of the biggest clusters formed from the gates in column four. Finally, the last column, *Run Time* lists the running time for the procedure in each row.

From the column *Largest Cluster Size* in each table, we see that for a circuit, the size of the largest cluster decreases from row one to row two, and either decreases or stays the same from row two to row three. The data in row three of each benchmark circuit indicates the size of the largest cluster after we have used additional gate-exhaustive and bridging faults to cover the sites of undetectable faults. The clusters for $k = 1$ are smaller than those for $k = 2$, because of a less strict target for gate coverage in case of $k = 1$. For similar reasons, the tests required for $k = 2$ are more than those necessary for $k = 1$.

Table 4.1. Single-cycle Faults for $k = 1$

Benchmark Circuits		**Single-cycle Test Patterns**	**Gates corresponding to Undetected Faults** (DFM+Stuck-at)	**Largest Cluster Size**	**Run Time**
b04	DFM (1-cycle)	167	81	65	2
	DFM (1-cycle) + Stuck-at	171	79	63	0.7
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	171	22	9	0.7
b14	DFM (1-cycle)	539	425	335	7.8
	DFM (1-cycle) + Stuck-at	608	340	260	0.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	609	9	2	4.3
b15	DFM (1-cycle)	925	1117	1015	66.1
	DFM (1-cycle) + Stuck-at	1000	1035	924	19.5
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	1004	19	8	4.9
b20	DFM (1-cycle)	647	878	315, 364	11.7
	DFM (1-cycle) + Stuck-at	713	754	256, 313	1.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	715	28	2	5.9
sparc_ffu	DFM (1-cycle)	490	1722	1627	12
	DFM (1-cycle) + Stuck-at	514	1692	1460	1.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	517	817	782	5.1
sparc_exu	DFM (1-cycle)	890	4138	3929	36.1
	DFM (1-cycle) + Stuck-at	950	4039	3843	3.4
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	956	1214	1170	11.4

Table 4.1. Continued..

s9234	DFM (1-cycle)	210	108	68	3.9
	DFM (1-cycle) + Stuck-at	227	86	55	0.7
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	228	8	1	1.6
s13207	DFM (1-cycle)	102	118	19	2.6
	DFM (1-cycle) + Stuck-at	109	92	14	0.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	109	73	13	1.6
s38417	DFM (1-cycle)	383	619	196,116	12.8
	DFM (1-cycle) + Stuck-at	407	539	167, 107	1.1
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	408	59	4,2	5.6
s35932	DFM (1-cycle)	164	630	518	11.4
	DFM (1-cycle) + Stuck-at	164	630	518	3.2
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	164	74	72	5.3
s38584	DFM (1-cycle)	296	578	323	10.4
	DFM (1-cycle) + Stuck-at	308	487	269	1.4
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	309	81	4	6

Table 4.2. Two-cycle Faults for $k = 1$

Benchmark Circuits		**2-cycle Test Patterns**	**Gates corresponding to Undetected Faults** (DFM+tran)	**Largest Cluster Size**	**Run Time**
b04	DFM (2-cycle)	109	100	96	1.6
	DFM (2-cycle) + Transition	156	24	8	0.8
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	158	15	6	1.6
b14	DFM (2-cycle)	260	578	556	4.1
	DFM (2-cycle) + Transition	484	39	26	3.1
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	487	33	25	2.1
b15	DFM (2-cycle)	609	1709	1649	105.9
	DFM (2-cycle) + Transition	1057	486	228	43.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	1170	171	132	3.5
b20	DFM (2-cycle)	340	1044	544, 436	7.9
	DFM (2-cycle) + Transition	594	123	52, 50	10.7

Table 4.2. Continued..

	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	600	64	25,25	2.9
sparc_ffu	DFM (2-cycle)	223	1544	1449	6.6
	DFM (2-cycle) + Transition	378	561	527	4.1
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	380	557	525	3.3
sparc_exu	DFM (2-cycle)	346	2878	2537	17.5
	DFM (2-cycle) + Transition	705	950	937	9.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	706	942	932	6.5
s9234	DFM (2-cycle)	105	164	125	2.5
	DFM (2-cycle) + Transition	191	22	7	1.0
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	191	20	7	1.6
s13207	DFM (2-cycle)	62	278	61, 33	2.2
	DFM (2-cycle) + Transition	84	206	41, 28	0.8
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	96	157	23,11	1.6
s38417	DFM (2-cycle)	145	1026	528	7.1
	DFM (2-cycle) + Transition	265	191	31, 20	2.9
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	270	165	31, 17	3.6
s35932	DFM (2-cycle)	98	214	133	7.0
	DFM (2-cycle) + Transition	110	49	31	1.7
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	111	31	31	3.5
s38584	DFM (2-cycle)	120	1246	943	6.2
	DFM (2-cycle) + Transition	232	262	28,27	2.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	242	181	27,15	3.0

Table 4.3. Single-cycle Faults for $k = 2$

Benchmark Circuits		**Single-cycle Test Patterns**	**Gates corresponding to Undetected Faults** (DFM+stuck-at)	**Largest Cluster Size**	**Run Time**
b04	DFM (1-cycle)	167	81	65	2.0
	DFM (1-cycle) + Stuck-at	171	79	63	0.7
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	171	37	15	0.7

Table 4.3. Continued..

b14	DFM (1-cycle)	539	425	335	7.8
	DFM (1-cycle) + Stuck-at	608	340	260	0.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	612	91	16	3.8
b15	DFM (1-cycle)	925	1117	1015	66.1
	DFM (1-cycle) + Stuck-at	1000	1035	924	19.5
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	1011	203	92	5.6
b20	DFM (1-cycle)	647	878	315, 364	11.7
	DFM (1-cycle) + Stuck-at	713	754	256, 313	1.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	720	208	34,16	6.0
sparc_ffu	DFM (1-cycle)	490	1722	1627	12.0
	DFM (1-cycle) + Stuck-at	514	1692	1460	1.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	531	899	860	5.0
sparc_exu	DFM (1-cycle)	890	4138	3929	36.1
	DFM (1-cycle) + Stuck-at	950	4039	3843	3.4
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	978	1644	1477	15.5
s9234	DFM (1-cycle)	210	108	68	3.9
	DFM (1-cycle) + Stuck-at	227	86	55	0.7
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	231	44	18	2.4
s13207	DFM (1-cycle)	102	118	19	2.6
	DFM (1-cycle) + Stuck-at	109	92	14	0.6
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	109	78	13	2.4
s38417	DFM (1-cycle)	383	619	196,116	12.8
	DFM (1-cycle) + Stuck-at	407	539	167, 107	1.1
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	411	305	52, 37	4.7
s35932	DFM (1-cycle)	164	630	518	11.4
	DFM (1-cycle) + Stuck-at	164	630	518	3.2
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	164	81	73	5.2
s38584	DFM (1-cycle)	296	578	323	10.4
	DFM (1-cycle) + Stuck-at	308	487	269	1.4
	DFM (1-cycle) + Bridging + Gate-Exhaustive(1-cycle)	323	213	68	6.3

Table 4.4. Two-cycle Faults for $k = 2$

Benchmark Circuits		**2-cycle Test Patterns**	**Gates corresponding to Undetected Faults** (DFM+tran)	**Largest Cluster Size**	**Run Time**
b04	DFM (2-cycle)	109	100	96	1.6
	DFM (2-cycle) + Transition	156	24	8	0.8
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	159	19	7	1.6
b14	DFM (2-cycle)	260	578	556	4.1
	DFM (2-cycle) + Transition	484	39	26	3.1
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	488	34	26	2.1
b15	DFM (2-cycle)	609	1709	1649	105.9
	DFM (2-cycle) + Transition	1057	486	228	43.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	1273	246	155	5.1
b20	DFM (2-cycle)	340	1044	544, 436	7.9
	DFM (2-cycle) + Transition	594	123	52, 50	10.7
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	604	115	50,50	2.9
sparc_ffu	DFM (2-cycle)	223	1544	1449	6.6
	DFM (2-cycle) + Transition	378	561	527	4.1
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	382	559	526	3.3
sparc_exu	DFM (2-cycle)	346	2878	2537	17.5
	DFM (2-cycle) + Transition	705	950	937	9.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	709	947	935	4.9
s9234	DFM (2-cycle)	105	164	125	2.5
	DFM (2-cycle) + Transition	191	22	7	1.0
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	191	20	7	1.6
s13207	DFM (2-cycle)	61	278	61, 33	2.2
	DFM (2-cycle) + Transition	84	206	41, 28	0.8
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	103	178	23, 16	1.6
s38417	DFM (2-cycle)	145	1026	528	7.1
	DFM (2-cycle) + Transition	265	191	31, 20	2.9
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	276	169	31,17	3.1
s35932	DFM (2-cycle)	98	214	133	7.0
	DFM (2-cycle) + Transition	110	49	31	1.7

Table 4.4. Continued..

	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	112	31	31	3.6
s38584	DFM (2-cycle)	120	1246	943	6.2
	DFM (2-cycle) + Transition	232	262	28,27	2.4
	DFM (2-cycle) + Gate-Exhaustive (2-cycle)	245	206	27,15	3.2

It is observed that for both single-cycle and two-cycle faults, very few additional test patterns are required to detect all the gate-exhaustive faults used for coverage of gates associated with undetectable faults. A major percentage of the additional gate-exhaustive and bridging faults are already detected by the test patterns from the previous row.

This observation is explained through Table 4.5 and 4.6. In Table 4.5, the second column shows the number of undetected single-cycle gate-exhaustive faults after simulating them with $T_{1-DFM} \cup T_{stuck}$, while the third column lists the actual number of undetectable single-cycle gate-exhaustive faults in the circuit. The data presented in Table 4.5 shows that only a small percentage of detectable single-cycle gate-exhaustive faults is left undetected by the test set $T_{1-DFM} \cup T_{stuck}$. Few or none of the faults from this small percentage are actually used for providing coverage to a gate. Most of the faults used are already detected by the existing test set, and hence we see a very small increase in the number of test patterns from row two to row three in Tables 4.1 and 4.3. An example of such a case is shown in the previous chapter (Example 4).

Table 4.6 presents similar data for two-cycle faults. Here we see that a significant number of detectable gate-exhaustive faults are left undetected by the existing test patterns. Nevertheless, we do not see a significant increase in the number of test patterns in row three except in circuit b15. This can be explained by the data in columns four and five in Table 4.6. Column four indicates that except b15, all the circuits have few or no undetectable DFM internal delay faults. Most of the gates belonging to these circuits only have undetectable transition faults associated with them, and as was the case in Example 5 in the previous chapter, the number of undetectable transition faults associated with a gate is less than or equal to two. Moreover, transition faults are easier to cover than internal delay faults

because they require less strict input pattern conditions (explained in Section 3.2). Hence, most of the gate-exhaustive faults required to cover these gates can be obtained from those detected by existing test patterns. Example 5 shows such a case in circuit s38584. However, in case of circuit b15, we have a significant number of undetectable internal delay faults as well. The number of internal delay faults per gate could exceed two, and they require stricter input patterns to cover them. Therefore, in case of circuit b15, we require a larger number of gate-exhaustive faults to be selected for coverage, and hence we see a significant increase in the number of additional tests required to be generated (Tables 4.2 and 4.4).

Table 4.5. Single-cycle Gate-exhaustive Faults

Benchmark Circuits	**Undetected 1-cycle Gate-Exhaustive faults simulated with** $T_{1-DFM} \cup T_{stuck}$	**Actual undetectable 1-cycle Gate-Exhaustive faults**
b04	524	511
b14	3431	3277
b15	6160	6068
b20	7935	7670
sparc_ffu	12103	12001
sparc_exu	27629	27058
s9234	863	816
s13207	720	663
s38417	6479	6009
s35932	5690	5683
s38584	5651	5271

Table 4.6. Two-cycle Gate-exhaustive Faults

Benchmark Circuits	**Undetected 2-cycle Gate-Exhaustive faults simulated with** $T_{2-DFM} \cup T_{tran}$	**Actual undetectable 2-cycle Gate-Exhaustive faults**	**Undetectable Internal Delay faults**	**Undetectable Transition faults**
b04	6065	4134	20	86
b14	38986	28898	0	50
b15	85421	66138	354	435
b20	88375	66616	0	147
sparc_ffu	149378	114591	0	620

Table 4.6. Continued..

sparc_exu	332754	247345	0	970
s9234	11664	8379	0	35
s13207	6867	5788	0	433
s38417	95536	62207	1	254
s35932	58263	45188	18	62
s38584	68957	47973	1	501

The circuits listed above consist of gates which have a maximum of four inputs. The number of inputs of a gate determines the number of gate-exhaustive faults associated with it. If we have circuits with larger gates, or treat small sub-circuits as gates, the number of internal faults associated with these gates will be higher. Consequently, the undetectable internal faults (single-cycle and two-cycle) per gate would increase, and so would the required number of gate-exhaustive faults necessary to cover them. In such a case, we might have a larger increase in the number of test patterns added in the third row, i.e., after adding gate-exhaustive faults for covering the sites of undetectable faults.

5. CONCLUSION

Undetectable faults from different fault models including Design-for-Manufacturability (DFM) tend to cluster together in a circuit. This leaves large areas of the circuit uncovered, and defects occurring in those areas have a chance of going undetected. Therefore, we attempted to cover these areas using gate-exhaustive faults. The total number of gate-exhaustive faults in a circuit can be very large, so we utilized only those faults which helped us specifically cover those areas with undetectable faults.

We developed procedures to select gate-exhaustive faults which could provide coverage to such areas. We also defined conditions for a gate associated with undetectable faults, which designated it as covered or uncovered, in relation to the number of detectable gate-exhaustive faults around it. We applied the developed procedures on benchmark circuits, separately for single-cycle and two-cycle faults. The size of the largest cluster of gates associated with undetectable faults was used as a measure of the area of a circuit left uncovered.

We also varied the parameters in our definition of gate coverage to observe the changes in the final coverage of circuits. From the data obtained, we observed that the size of clusters decreased as we included gate-exhaustive faults in our fault set. The reduction in cluster size was seen for single-cycle and two-cycle faults, even when the parameters were varied. The increase in the number of test patterns added due to gate-exhaustive faults was small, due to always having selected as many faults as possible from those already detected by existing test sets. Overall, the selection of gate-exhaustive faults from around the clusters and generation of tests for detecting them facilitated checking and increasing the coverage of circuits, and subsequently the probability of detecting defects occurring at the sites of undetectable faults increased.

REFERENCES

[1] C. Schuermyer, K. Cota, R. Madge, and B. Benware, "Identification of systematic yield limiters in complex asics through volume structural test fail data visualization and analysis," in *IEEE International Conference on Test, 2005.*, IEEE, 2005, 9–pp.

[2] R. Desineni, L. Pastel, M. Kassab, M. F. Fayaz, and J. Lee, "Identifying design systematics using learning based diagnostic analysis," in *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, IEEE, 2010, pp. 317–321.

[3] S. Kundu and A. Sreedhar, "Modeling manufacturing process variation for design and test," in *2011 Design, Automation & Test in Europe*, IEEE, 2011, pp. 1–6.

[4] D. Kim, M. E. Amyeen, S. Venkataraman, I. Pomeranz, S. Basumallick, and B. Landau, "Testing for systematic defects based on dfm guidelines," in *2007 IEEE International Test Conference*, IEEE, 2007, pp. 1–10.

[5] M. Abramovici, M. A. Breuer, A. D. Friedman, *et al.*, *Digital systems testing and testable design*. Computer science press New York, 1990, vol. 2.

[6] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition fault simulation," *IEEE Design & Test of Computers*, vol. 4, no. 2, pp. 32–38, 1987.

[7] E. J. McCluskey, "Quality and single-stuck faults," in *Proceedings of IEEE International Test Conference-(ITC)*, IEEE, 1993, p. 597.

[8] J. Savir, "Skewed-load transition test: Part i, calculus," in *Proceedings International Test Conference 1992*, IEEE Computer Society, 1992, pp. 705–705.

[9] S. Patil and J. Savir, "Skewed-load transition test: Part ii, coverage," in *Proceedings International Test Conference 1992*, IEEE Computer Society, 1992, pp. 714–714.

[10] J. Savir and S. Patil, "Broad-side delay test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1057–1064, 1994.

[11] K. C. Mei, "Bridging and stuck-at faults," *IEEE Transactions on Computers*, vol. 100, no. 7, pp. 720–727, 1974.

[12] C.-H. Wu and K.-J. Lee, "An efficient diagnosis pattern generation procedure to distinguish stuck-at faults and bridging faults," in *2014 IEEE 23rd Asian Test Symposium*, IEEE, 2014, pp. 306–311.

[13] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, "Cell-aware test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1396–1409, 2014.

[14] F. Hapke, W. Redemund, J. Schloeffel, R. Krenz-Baath, A. Glowatz, M. Wittke, H. Hashempour, and S. Eichenberger, "Defect-oriented cell-internal testing," in *2010 IEEE International Test Conference*, IEEE, 2010, pp. 1–10.

[15] A. D. Singh, "Cell aware and stuck-open tests," in *2016 21th IEEE European Test Symposium (ETS)*, IEEE, 2016, pp. 1–6.

[16] T. Herrmann, S. Malik, and S. Madhavan, "Quantified contribution of design for manufacturing to yield at 28nm," in *2014 19th IEEE European Test Symposium (ETS)*, IEEE, 2014, pp. 1–6.

[17] D. Kim, I. Pomeranz, M. E. Amyeen, and S. Venkataraman, "Defect diagnosis based on dfm guidelines," in *2010 28th VLSI Test Symposium (VTS)*, IEEE, 2010, pp. 206–211.

[18] A. Sinha, S. Pandey, A. Singhal, A. Sanyal, and A. Schmaltz, "Dfm-aware fault model and atpg for intra-cell and inter-cell defects," in *2017 IEEE International Test Conference (ITC)*, IEEE, 2017, pp. 1–10.

[19] I. Pomeranz and S. M. Reddy, "On clustering of undetectable single stuck-at faults and test quality in full-scan circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1135–1140, 2010.

[20] I. Pomeranz, "On clustering of undetectable transition faults in standard-scan circuits," in *29th VLSI Test Symposium*, IEEE, 2011, pp. 128–133.

[21] N. Wang, I. Pomeranz, S. M. Reddy, A. Sinha, and S. Venkataraman, "Resynthesis for avoiding undetectable faults based on design-for-manufacturability guidelines," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1022–1027.

[22] X. Lin and J. Rajski, "The impacts of untestable defects on transition fault testing," in *24th IEEE VLSI Test Symposium*, IEEE, 2006, 6–pp.

[23] R. D. Blanton and J. P. Hayes, "Properties of the input pattern fault model," in *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, IEEE, 1997, pp. 372–380.

[24] K. Y. Cho, S. Mitra, and E. J. McCluskey, "Gate exhaustive testing," in *IEEE International Conference on Test, 2005.*, IEEE, 2005, 7–pp.

[25] R. Guo, S. Mitra, E. Amyeen, J. Lee, S. Sivaraj, and S. Venkataraman, "Evaluation of test metrics: Stuck-at, bridge coverage estimate and gate exhaustive," in *24th IEEE VLSI Test Symposium*, IEEE, 2006, 6–pp.

[26] I. Pomeranz, "Iterative test generation for gate-exhaustive faults to cover the sites of undetectable target faults," in *2019 IEEE International Test Conference (ITC)*, IEEE, 2019, pp. 1–7.

[27] R. Asami, T. Hosokawa, M. Yoshimura, and M. Arai, "A multiple target test generation method for gate-exhaustive faults to reduce the number of test patterns using partial maxsat," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, IEEE, 2020, pp. 1–6.