TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Engineering

IAF70LT

Oyeniran Adeboye Stephen

IASMM 132085

DOUBLE PHASE FAULT COLLAPSING WITH LINEAR COMPLEXITY IN DIGITAL CIRCUIT

Master thesis

Prof. Raimund-Johannes Ubar

D.Sc. Institute of Computer Engineering, Tallinn University of Technology

Professor, Chair of Computer Systems Test and Verification

Tallinn 2015

Author’s Declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referenced. This thesis has not been presented for examination anywhere else.

Author: Adeboye Stephen Oyeniran.

06.04.05

Abstract

The issue of digital systems testing with respect to test pattern generation and fault diagnosis was investigated in this thesis work. Hence, we propose a new structural fault collapsing method with linear complexity with the aim of reducing the search space for test generation and fault diagnosis in digital systems. And this is the main objective and concentration of this thesis. Secondly, we proposed a method for estimating the achievable size of the collapsed fault set for a given SSBDD. This method is referred to as lower and upper bounds estimation for fault collapsing.

The fault collapsing method proposed in this thesis work is divided into two phases. The first phase of the fault collapsing method is regarded as a superposition of binary decision diagram (BDD) of logic gates into a model that extracts both function and data about the structural path of the circuit. Considerable number of faults can be collapsed at this stage, this is because the SSBDD has fewer nodes than the gate level models as a result of the compactness of the model. The second phase of this method involves topological analysis of the SSBDD model derived from the first phase. Both phases have linear complexity.

We proved with experimental result that the method proposed in this thesis offers better collapsing capabilities when compared with previous structural fault collapsing methods

Annotatsioon

Kahefaasiline lineaarse keerukusega algoritm rikete kollapseerimiseks digitaalskeemides

Käesolevas lõputöös uuriti digitaalsüsteemide testimist testvektorite genereerimise ja rikete diagnoosimise vaatepunktist. Töö peamiseks eesmärgiks oli leida meetod, mis lihtsustab rikketuvastust testvektorite otsinguhulga vähendamise kaudu. Magistritöö põhitulemusena pakutakse välja lineaarselt kasvava keerukusega meetod rikete kollapseerimiseks, mis võimaldab kiirendada testide genereerimist ja rikete simuleerimist ning muudab lihtsamaks rikketuvastuse digitaalsüsteemides. Töö kõrvaltulemusena leiti meetod, mis võimaldab hinnata rikete kollapsi piire. Rikete kollapseerimise meetod töötab kahes etapis. Esimene etapp seisneb otsustusdiagrammide SSBDD sünteesis, kus erinevate signaaliteedede rikked kollapseeruvad vastavat signaaliteed esindava graafitipu riketeks. Suurt hulka rikkeid saab selles etapis elimineerida, kuna SSBDD on kompaktsem ja sisaldab vähem sõlmi kui värati-tasemel mudel. Meetodi teine etapp seisneb SSBDD mudeli topoloogilises analüüsis ning on samuti lineaarse keerukusega, nagu eelmine etappki. Eksperimentide abil tõestati, et töös välja pakutud meetod pakub paremaid rikete vähendamise võimalusi võrreldes teiste autorite poolt varem pakututega

Acknowledgements

I will like to thank God for His grace to embark on this degree. I owe it all to Him for His grace of life and ability to complete this thesis.

Secondly, I will like to thank my supervisor, Prof. Raimond Ubar for his supervision and valuable advice to be able to complete this research successfully. Your guidance and advice have been very valuable, helpful. Thank you, you will never be forgotten.

I also appreciate Teet Evartson who agreed to review this master's thesis. Thank you for your time.

I will like to thank my colleagues, Galina Josifovska and Palle Kotta for your friendship through these 2 years of new discovery. You have made it easy to adapt to Tallinn Estonia.

My appreciation also goes to my African friends here in Tallinn, Pastor Chidi, Mr. Ogunyemi, Godswill, Ayo, John, Dayo, Funmi, Dami, my lovely baby Praise Nsehe and every member of RCCG Tallinn to mention but a few. I love you all; you have made it an interesting time so far. I will not forget you quickly.

My appreciation also goes to my future wife, soon we will be together and you will be proud of who I have been. I love you in advance.

Finally, I will like to thank my family for their support and understanding during these two years of absence from them. Your support have been a driving force to completing this study. I love you all.

This work was supported by IT academy under the monitoring of Estonian IT Foundation for Education (HITSA).

Table of abbreviations and terms

ATG	Automated Test Generation
BDD	Binary Decision Diagram
CMOS	Complementary metal–oxide–semiconductor
CPU	Central Processing Unit
CUT	Circuit Under Test
FFR	Fanout-Free Region
ICs	Integrated Circuits
MSF	Multiple Stuck-At Fault
ROBDD	Reduced Ordered Binary Decision Diagram
S-A-0	Stuck-At-0
S-A-1	Stuck-At-1
SAF	Stuck-At Fault
SSF	Single Stuck-At Fault
TG	Test Generation
SSBDD	Structurally Synthesized Binary Decision Diagram
S3BDD	Shared Structurally Synthesized Binary Decision Diagram
VLSI	Very Large Scale Integrated circuits

Table of Contents

List of figures

List of tables

1. Introduction

This thesis work focuses on improving the efficiency of fault simulation that enhances digital circuit testing by the introduction a method of fault collapsing.

This first chapter begins with the problem statement, followed by a description of task solved and methodology. Finally in this chapter is the overview or summary of work.

1.1. Background and Problem

According to Moore's law, the scale of ICs has doubled every 18 months [1], [2], [3], [4]. This consequently explains the exponential growth of digital systems and increases its application in different spheres of today's living. Virtually everyone today depends on digital devices.

Although according to [2], integrated electronics has demonstrated high reliability, but this could be challenged as the complexity of digital circuit increases due to the fact that in today's digital systems, lots of transistors are used whereas the reduced size is also considered. However, reliability should not also be compromised. This therefore means that a proper test has to be done efficiently on these systems before final use.

Testing of a system is an experiment in which the system is exercised, and its resulting response is analyzed to ascertain whether it behaved correctly [1].

During testing, we apply a set of stimuli (test patterns) to the input of a circuit or system usually known as circuit under test (CUT) and at the same time observing and analyzing the output. The analysis of the output involves comparing with expected circuit response. A circuit will be regarded as fault free if upon analysis of the output after subjecting the CUT to stimuli correlates to the expected result otherwise the circuit will be assumed faulty.

Since test pattern generation algorithms are time-consuming, it is important that a fault list used in a test process is reduced as much as possible. A small fault list reduces redundancy in generated test vectors as well as the overall test time [5].

This thesis work tries to address the issues of test efficiency by reducing the number of test patterns generation during testing and consequently reducing the time needed for testing. This is believed to be achievable if the number of faults to be tested is reduced by identifying redundant faults in a system or circuit to be tested and collapsing them.

Hence, a new structural fault collapsing method with linear complexity to reduce the search space for test generation and fault diagnosis in digital circuit is presented.

1.2. Description of the task solved

The main contribution of this thesis work is highlighted as below:

We proposed a new structural fault collapsing method with linear complexity to reduce the search space for test generation and fault diagnosis in digital circuits [6].

The thesis tries to compare the new algorithm to other fault collapsing algorithms, for example, structural fault collapsing by superposition of binary decision diagram BDDs proposed by [7]. The proposed method of fault collapsing shows with experimental data more efficiency and better time cost compared other structural fault collapsing methods [6].

The method proposed in this thesis is divided into two phases each with a goal of collapsing faults in a circuit. The first phase of this method is regarded as superposition of BDD of logic gate into a model that extracts both function and data about the structural path of the circuit [6], [8]. This model is known as SSBDD. Fault collapsing in this first phase results from the effect of the SSBDD model compaction [6].

The second phase of the method involves the topological analysis of the SSBDDs [6].

The experimental result of the method described in this thesis work is compared with other methods that have aimed at fault collapsing in terms of number of faults collapsed and the time.

1.3. Thesis Structure

The thesis is organized as follows. In Chapter 2, an overview of testing is given and a discussion on the various fault models used in digital circuit was discussed. We also looked into the challenges of test generation and fault diagnosis in digital systems. In Chapter 3, Double phase fault collapsing with linear complexity in digital circuit is discussed and a new method for increasing the number of collapsed faults was developed. Chapter 4 Explains the result of proposed method and comparison with other methods of fault collapsing. Finally, Chapter 5 The conclusion or summary of work is presented.

1.4. Overview of work

We propose a new structural fault collapsing method that is based on the two-phase topology analysis of the circuit description. The first phase is known as SSBDD synthesis which is carried out at the gate level by superposition of elementary BDD of logic gates while the second phase involves topological analysis of the SSBDD.

The proposed method uses SSBDD model which has fewer nodes than the gate-level models. Hence, we can already have a reduced number of fault set which could be regarded as initial fault collapsing. The fault collapsing capability not only decreases the space needed, it also leads to increased speed of fault simulation and test generation.

The contribution of this thesis is in providing an excellent and scalable collapsing method that is very promising for large circuits. This method has been proven to be more efficient than previous structural fault collapsing methods

2. Background

In this chapter, background information or topics relating to the focus of this thesis research is presented. This begins with a description of the concept of digital testing, followed by fault modeling. A discussion on fault collapsing is done in view of equivalent faults and dominant faults. The chapter will be concluded with discussion on test generation and fault diagnosis. We will draw the motivation for this work and propose for the next chapter a method of optimizing these operations.

2.1. Digital Testing

Testing, in general, is a process of checking the genuineness, quality, and reliability or correct functioning of something. In digital systems, testing is not far from this. It is a way to check the correctness of a system.

According to [1], testing of a system is an experiment in which the system is exercised and its resulting response is analyzed to ascertain whether it behaved correctly. If an incorrect behavior is detected, the second goal of a testing experiment may be to diagnose, or locate, the cause of the misbehavior. Diagnosis assumes knowledge of the internal structure of the system under test. These concepts of testing and diagnosis have a broad applicability. Consider, for example, medical tests and diagnosis, test-driving a car, or debugging a computer program.

Another interesting definition of testing, as given by [9], is that testing can be characterized as a black-box experiment. At each level of complexity, a digital system or a part of it can be regarded as a black box having a set of input and output terminals. We can determine correct functioning of the box by applying stimuli to the input and observing the responses at the output.

During testing, we apply a set of stimuli (test patterns) to the input of a circuit or system usually known as circuit under test (CUT) and at the same time observing or analyzing the output. The analysis of the output involves comparing it with expected circuit response. A circuit will be regarded as fault-free if upon analysis of the output after subjecting the CUT to stimuli correlates to the expected result, otherwise the circuit will be assumed faulty.

The Figure 1 below explains the testing process on a digital circuit under test. Test patterns are applied as input and a comparison is done between the output of the circuit and the expected response. Based on this a decision is made on whether the circuit passes or fails.

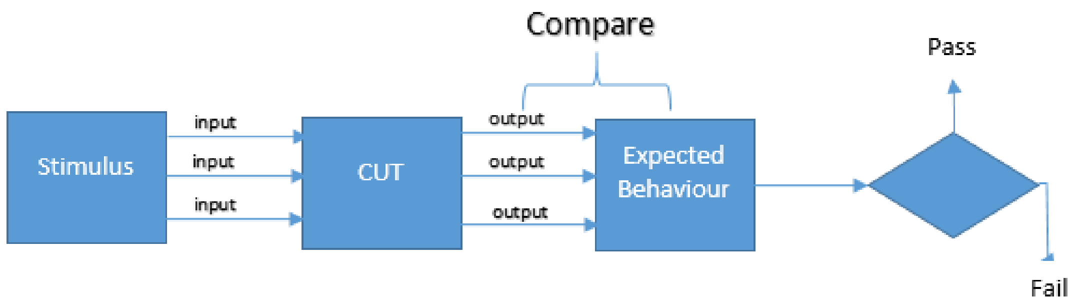


Figure 1: Digital Circuit Testing Process

Testing can be done at different levels of attraction, but of interest to this work is the logic level of abstraction. At this level of abstraction, we represent the information processed by logic values, which are represented as values of 0’s and 1’s [1].

Benefits of Testing

Reliability of a system is of utmost importance in today’s world owing to the fact that the effect of correct functioning of a system has extended across wider range of applications. A key requirement for obtaining reliability of a system is to determine that the system is error free. We can also say that the way to achieve system reliability is to check or ensure that the system is error free [10].

Quality systems are a product of correct and quality testing process. If the test procedure is good, but the system fails testing, then it is either that the fabrication process, design or specification is faulty [11].

According to [11], quality and economy are benefits of testing, where quality means satisfying the user’s need at minimum cost. Economy involves overall cost of manufacturing and testing.

2.2. Fault Modeling

Fault models are necessary for generating and evaluating a set of test vectors. According to the text on VLSI test principles, a good fault model should generally satisfy two criteria which are: (1) It should accurately reflect the behavior of defects, and (2) it should be computationally efficient in terms of fault simulation and test pattern generation [2].

Many fault models have been proposed, [1] which are: stuck-at faults, bridging fault, transistor fault, delay fault and open faults. At the same time, the behavior of all possible defects that can occur in a circuit unfortunately cannot be reflected accurately by a single fault model [2]. This results in the use of a combination of different fault models in the process of generating test vectors and evaluating them for testing purposes.

A fault model is a description at the digital logic level of the effects of some fault or combination of faults in the underlying circuitry. One of the advantages of fault model is that it is technology independent and works well in practice. However, it may fail to identify certain process specific faults (e.g. CMOS floating gates) and detects static faults only [13].

In general, structural fault models assume that components are fault-free, and only their interconnections are affected [1].

On a general view, a fault model could fall under one of the following assumptions: single fault assumption or multiple fault assumption. The single fault model assumes that only one fault occurs in a circuit at any given time. In this case, we can estimate the total number of possible single faults as **m×n**. Where m would be different types of faults that could occur at each potential fault site. In most cases or models, m is equal to 2 and n would be possible fault locations or sites.

On the other hand, multiple fault model allows more than one fault to occur simultaneously in a circuit. The number of possible fault combination in this case, could be given as 3^n-1, where n is the number of nodes and each node could either be of state SAF-0, SAF-1 or correct state with the assumption that each node may have m = 2 faults. This is an exponential increase compared to the single fault model.

[1] pointed that while multiple-fault model is more accurate than the single-fault assumption, the number of possible faults becomes impractically large when dealing with multiple fault model except in the case of small circuits with small number of fault types and fault sites. The good news according to [11], [12] is that 100% test for single stuck-at fault is known to cover a very high percentage usually greater than 99.6% of multiple stuck-at-faults.

In most cases, the number of single faults that is to be used for test vector generation in a circuit is usually less than m×n. This is because when considering the single fault assumption, two or more faults could have identical behavior for all possible input patterns [1]. These faults are regarded as equivalent faults. When this occurs, equivalent faults in such circuit can be represented by a single fault in the set of equivalent faults. The redundant equivalent fault could be removed yet preserving the quality of the test, and this in general leads to a reduction of the entire set of equivalent faults. This reduction is known as fault collapsing.

The remaining part of this thesis will address this concept of fault collapsing, and different methods of achieving this will also be discussed.

2.3. Collapsing

2.3.1. Stuck at fault

We understood that structural fault models assume that components are fault-free, and only their interconnections are affected [1]. This implies that circuits are modeled as an interconnection of Boolean gates. These interconnections can also be referred to as netlists [11]. The Boolean gates that make up the circuits are fault free, but the interconnections are affected by stuck-at faults.

Interconnections between gates can have any of the two types of faults. It could either be stuck-at 1(S-A-1) or stuck-at 0(S-A-0). If a line is stuck-at 0, for instance, it means that the line will have a fixed logic value of 0 regardless of the output of the gate driving the line. The same applies to stuck-at 1 faults.

We can also say that a stuck-at fault transforms the correct value on the faulty signal line to appear to be stuck at a constant logic value, either a logic 0 or a logic 1, and this again is referred to as stuck-at-0 (SA0) or stuck-at-1 (SA1), respectively [2].

Let's take for example an AND gate with two inputs (a and b) and an output z, it has 3 fault sites (2 on the primary input and 1 on the output) and 6 single stuck-at faults. Each fault sites has 2 stuck-at fault which are S-A-0 and S-A-1 respectively. This can be shown in the Figure 2 below.

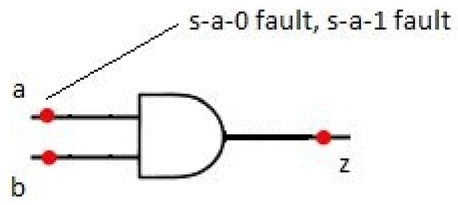


Figure 2: Stuck-at-Fault on a Circuit

Assuming we pass input vectors 1 for both a and b. The value of the output is expected to be 1. If the input a is stuck at 0, the value at output z will be 0. This will be regarded a faulty circuit value. Figure 3 below explains this.

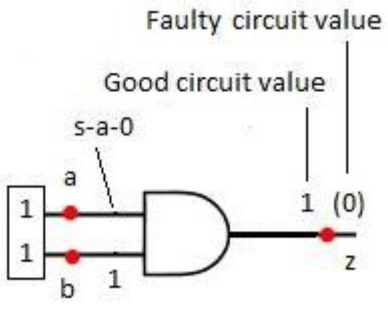


Figure 3: Testing for Faults

Stuck at fault model is of two types as mentioned above. They are: single stuck-at fault and multiple stuck at faults.

Single stuck-fault model (SSF) is regarded as classical or standard fault model. It is the most common used fault model. According to [1], the usefulness of SSF results from the following attributes:

- it presents many different physical faults.
- it is independent of technology.
- experience has shown that test that detect SSFs detect many non-classical faults as well.
- when compared to other fault models, the number of SSFs in a circuit is small.
- SSFs can be used to model other types of faults
- High SSFs coverage provides a high multiple stuck-at faults coverage.

The multiple stuck-fault (MSF) model is a straightforward extension of the SSF model in which several lines can be simultaneously stuck [1].

This thesis work will not extend its study into multiple stuck-at faults. The single stuck-at fault will be sufficient to implement and demonstrate the fault collapsing algorithm. And as mentioned previously, 100% test for single stuck-at fault are known to cover a very high percentage usually greater than 99.6% of multiple stuck-at-faults. Hence, we will focus only on single stuck-at fault model.

2.3.2. Test Generation

To simplify the objective of test generation, We would define it as a way to generate a test vector or pattern for a given fault in a given circuit or prove that it is untestable.

According to [15], a set of test must be generated within a given amount of computational effort. For some faults known as abortive faults, test generation could be terminated before a test is generated or proven untestable.

The task of test generation is finding a set of patterns or vectors that will fully test the circuit. This means that either all faults are detected or a maximal fraction of the testable faults are detected. Hence, the goal is to minimize the number of aborted faults within a given amount of computational effort.

When the test patterns are applied to a circuit, they cause all faulty circuit to exhibit behaviors different from that of the good circuits at the primary output. This implies that if there is a failure, it will be revealed when at least one of the primary output is different.

Test generation is a complex problem with many interacting aspects. The most important are:

• The cost of TG.

• The quality of the generated test.

• The cost of applying the test [1].

In view of the above points about test generation, it is safe to say that test set must be reasonable in the sense that we must be able to apply it parsimoniously to all circuits produced. It is true that using all possible input patterns will reveal faulty circuits, but this will be at the expense of cost and other resources like time of test and memory consumption.

In summary, testing cost can be estimated as test generation time, test application time, fault coverage, test storage cost (test length) and availability of automatic test equipment. Therefore, the motivation for this project is to reduce as much as possible the cost of testing. It is postulated that reducing the cost of test generation will impact the testing cost in general.

The TG cost also depends on the complexity or size of the circuit to be tested. Complex and large circuits with many inputs produce a huge set of possible faults. The main challenge here will be improving the efficiency, and cost of test generation by reducing test sets and search space without compromising the quality. This will considerably increase the speed of fault simulation. This is because reduced test sets and search space will reduce the number of faults to be simulated and improve the ability to cope with the problem of diagnostic resolution during fault diagnosis of digital systems [6].

Test generation can be broadly divided into three (3) types. Exhaustive, deterministic and random. In a combinational logic block that contains no redundant logic, we can test a device by applying all possible 2K possible input patterns, Where K is the number of inputs. This type of testing is regarded as exhaustive testing [13]. For small circuits, this is not an issue, it is very good as it covers a very high fault, but it becomes impracticable as the size of the circuit and number of input increases (unless circuit is partitioned into cones of logic less than or equal to 15).

[13] estimated the time required for carrying out an exhaustive test on a digital circuit with respect to the size and number of primary inputs of the circuit. This can be seen in Table 1 below on the assumption that a tester is capable of applying a test pattern every 100ns.

Input	Number of Test	Test Time
20	2^{20}	0.1 Sec
40	2^{40}	30.5 Hours
60	2^{60}	58500 Years

Table 1: Exhaustive Test Time Estimation

Random TG is less complex compared to deterministic TG. Random TG may be able to initially generate tests quickly, but it would be very inefficient to achieve higher fault coverage[16]. Although deterministic TG has possibly high fault coverage and cheaper to implement, it is however expensive to generate. It uses the structural and functional information from the circuit in the test generation process.

According to [17] the problem of ATG, which is known to belong to the class of the NP-complete problems can be viewed as a finite space search problem. For a circuit with N primary inputs, there exist 2^N combination of input assignments. This 2^N combination represents all points the finite search consist of [18].

The requirement for deterministic test generation is focused on generating test patterns for targeted faults. In general, for any 2^N combination of input assignments, only small portion of these combinations fulfill the requirement described. The problem of deterministic TG algorithm suffers from is that they generally are not able to identify the entire non-solution areas, but part of them [18].

In this case, the problem can be minimized by reducing the number of 2^N combination and we can be achieve this by collapsing redundant faults.

2.3.3. Fault Diagnosis

According to [2], during the IC design and manufacturing cycle, a manufacturing test screens out the bad chips.

As we have discussed, a circuit under test (CUT) fails when its observed behavior is different from its expected behavior. Hence the task of diagnosis is finding out why the CUT fails to behave as expected. This consists of locating the physical fault(s) in a structural model of the CUT.

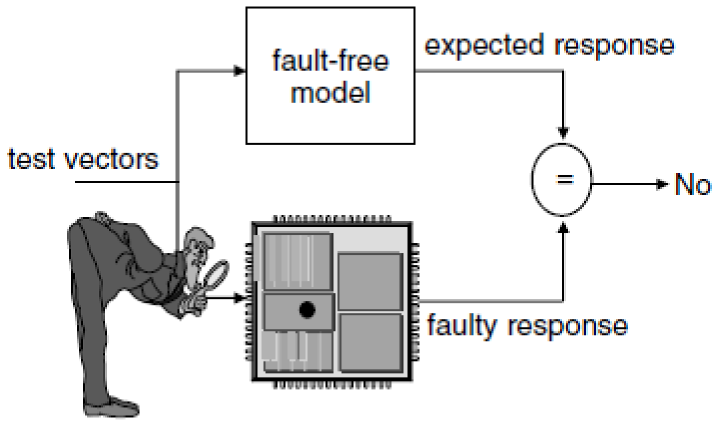


Figure 4: Diagnosis Process [2]

As illustrated in the Figure 4 above, during diagnosis, a comparison is made in relation to the behavior of a fault-free model which could also be referred to as the circuit under-diagnosis (CUD) to that of the faulty unit. During this process, test patterns are generated and introduced to both the CUD and the failing circuits. The responses from both (which are expected to differ) are observed in order to locate the faults.

We can view diagnosis based on two scenarios. One of which focuses on diagnosis in order to identify and make decisions about repair. This could be the case for some digital systems. The other scenario is in cases of digital VLSI chips where un-repairable and faulty chips must be discarded, even in these cases, diagnosis are done in the event where chip yield are low and performance are unacceptable [15]. The focus of diagnosis at this stage is to improve yield and performance, which will consequently affect the approach to the design. For instance, this will help to change chip design, design rules, design methodology, step(s) or fabrication.

Diagnosis can be either physical or logical. However, from research, it is known that physical diagnosis is expensive and destructive. Therefore, logical diagnosis is the first to be performed in order to first identify the likely faults before physical diagnosis is done. In this discussion, we will only focus on logical diagnosis.

The quality of a diagnosis can be measured either by identifying if it could pinpoint the fault site or by achieving complete diagnosis in considerably good time (which can be relative to size).

According to [2], the quality of pinpointing the fault site can be measured by the following indexes.

- Diagnostic resolution
- First-hit index
- Top-10 hit
- Success Rate

Diagnostic resolution can be defined as the total number of fault that can be identified by diagnosis. For some diagnostic tools, diagnostic accuracy cannot be measured by diagnostic resolution. For these tools, accuracy could be measured by first-hit index, top-10 hits or success rate as the case may be.

As mentioned previously, I will focus on combinational logic diagnosis. [1] has identified that fault diagnosis can be approached in two different ways. These are cause–effect analysis and effect–cause analysis. I will therefore shortly describe these approaches and point out possible problems that have been faced in these. In concluding this part, I will motivate my discussion by discussing the contribution of this thesis in trying to address these issues.

Cause-Effect Analysis

The cause-effect analysis uses fault simulation to determine the possible responses to a given test in the presence of faults. In this case, most of the work is done before the testing experiment. The approach here entails a database construction called a fault dictionary through an escalated or intensive fault simulation process. This is called cause-effect analysis because it starts with analyzing the possible faults (causes) and then determines their corresponding effects known as the response.

The fault dictionary stores the responses of a circuit to a test set in the presence of faults under a fault mode [20].

In order to locate faults, the task here would be to compare the actual response obtained from the CUT with the pre-computed responses stored in the fault dictionary. If the response obtained from the CUT matches that of the fault dictionary for one or more faults, then the dictionary will indicate the corresponding fault(s) and hence the CUT is diagnosed to have one of the faults.

Let's take an example of a CUD with four inputs (a, b, c, d) and one output (k) as shown in Figure 5 We can make assumption of generating six (6) test vectors {t1, t2, t3, t4, t5, t6}. Based on single stuck-at model, we can deduce after equivalent fault collapsing a fault universe given as {f1, f2, f3, f4, f5, f6}.

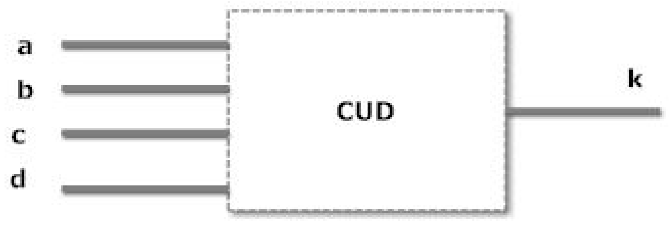


Figure 5: Example Circuit for Diagnosis

In Table 2, we show a full response table of the output signal k, which was gotten from complete fault simulation. The table includes values for both fault-free and faulty circuits.

Row data as in the table represents either faulty or fault-free circuits while column data represents the response of one test vector. However, it is safe to say from the simulation that the test set has 100% fault coverage.

CIRCUIT	**INPUT VECTORS(a, b, c, d)**					
	VI	**V2**	**V3**	**V4**	**V5**	**V6**
Fault Free	0	0	0	0	0	1
F1	0	1	1	1	1	1
F2	1	0	1	1	1	1
F3	1	1	1	0	1	1
F4	0	0	1	0	1	1
F5	0	0	0	1	1	1
F6	0	1	0	1	0	1

Table 2: Diagnostic Fault Response Table

Figure 6 shows a fault dictionary known as the fault diagnostic tree, which is built from the table. With the diagnostic tree, we can quickly know faulty circuits by simply traversing from the root of the tree to any of the leaf nodes. An example is for instance, we have a faulty chip with response of 101111. Traversing the tree helps us quickly identify that the faulty circuit is f2.

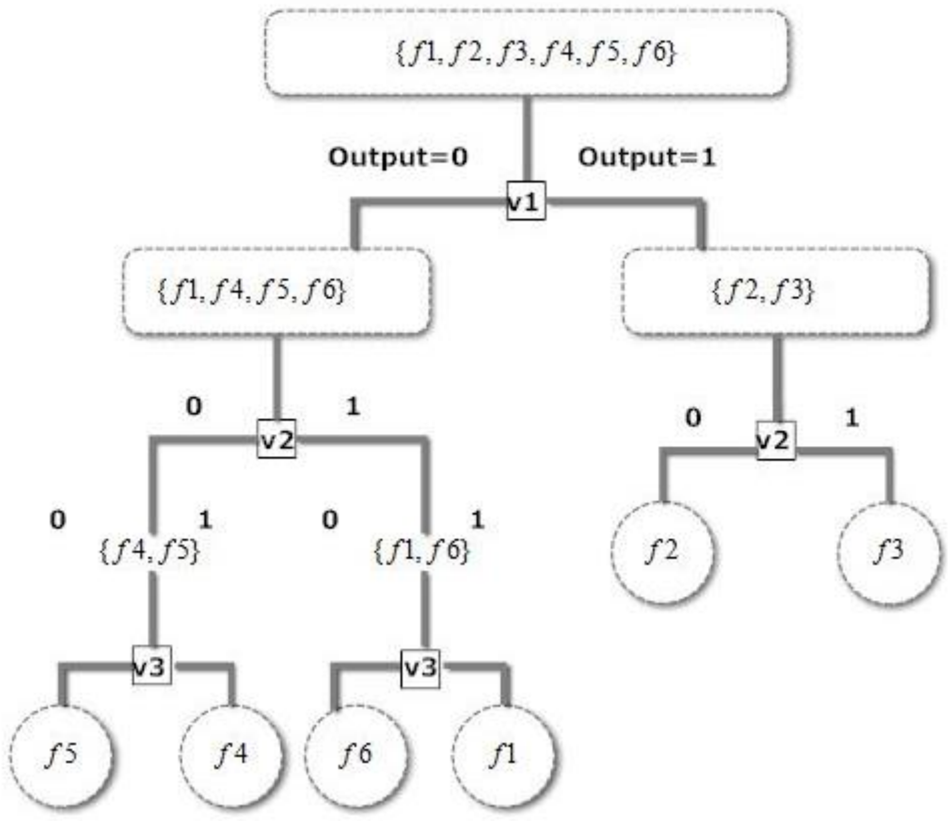


Figure 6: Fault Dictionary

The major issue with diagnosis involving the use of a fault dictionary is that for large circuits for instance, the dictionary is usually very large and would require lots of space and computational efforts especially if high diagnostic resolution is a focus. This however makes it impractical to be used in the diagnosis process. Reducing the computational time without reducing the diagnostic resolution is not an easy task. However, [22] evaluated this issue with computation time. They proposed that a small amount or number of diagnostic runs are sufficient to compensate for the nonrecurring effort of creating a small dictionary [21]. The only major issue here is still regarding the size of the dictionary. As mentioned previously, digital circuits are increasing in size making memory required for fault dictionary to also grow exponentially.

Several research have been made over the years in order to decrease the size of the fault dictionary, one of these was explained in [23], but none of these has been able to bring it down to an acceptable level.

Based on research, it is known that fault dictionary stores the response of the circuit to the test set in the presence of each fault, with this in view if we decrease the number of test vectors in the test set it will result in reduction in the size of the fault dictionary. This means that the smaller the test vectors the smaller the fault dictionary. The focus of this research is implementing an efficient way to reduce the size of the test vector without compromising the quality of diagnosis of a circuit. Therefore, reducing the number of faults reduces the size of fault dictionary and the focus of this research is towards reducing the number of faults in a circuit hence reducing the size of the fault dictionary during diagnosis.

Effect-Cause Analysis

This is also known as dynamic diagnosis. In this case, the effect or response of the circuit is analyzed or processed in order to determine the faults through Boolean reasoning of the CUD. Although this analysis could be superior to the cause-effect method because of its suitability for faults that are not stuck at faults and can be used in cases of multiple faults [2]. It however fails in the sense of very high computational time. It takes quite a long time to complete.

2.3.4. Equivalence Faults

Definition 2.3.4.1: Two faults of a Boolean circuit are called equivalent iff they transform the circuit such that two faulty circuits have identical output functions. Equivalent faults are also called indistinguishable and have exactly the same set of tests [11].

Single stuck-at faults at the inputs and output of a Boolean gate have structural equivalence relations [24]. Hence two faults f1 and f2 are equivalent if all tests that detect f1 also detect f2.

For example stuck-at 0 for all input and stack-at 1 at the output of a NAND gate are equivalent, likewise all stuck-at-0 faults of the input and output lines of an AND gate are equivalent.

The figures below show equivalent fault collapsing for Boolean gates, wires and fanouts.

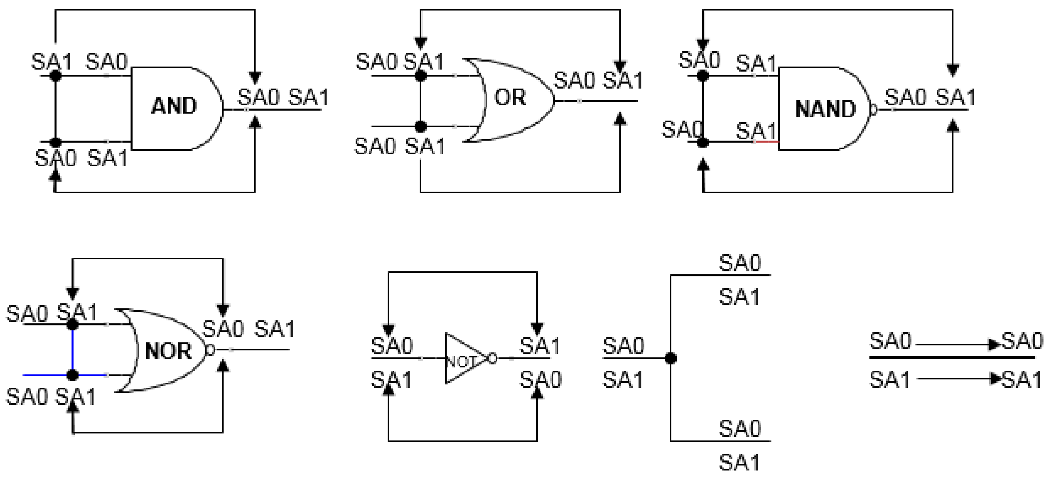


Figure 7: Equivalent Fault Collapsing

Equivalent fault collapsing principles and theorems could be applied to reduce fault sets in a combinational circuit. These theorems as discussed by [2] are as below and form the basic motivation behind fault collapsing by equivalence.

Theorem 2.3.4.1

A set of test vectors that detects all single stuck-at faults on all primary inputs of a fanout-free combinational logic circuit C will detect all single stuck-at faults in that circuit [1].

Proof: Assume, by contradiction, that a set of test Vector *T* detects all SSFs on the primary inputs of a circuit C but does not detect all internal SSFs. Because every line in C has only one fan-out, it is sufficient to consider internal faults only on gate outputs. Since *T* detects all SSFs on the primary inputs, there must be some gate G in C such that

T detects all faults in the circuit feeding G but does not detect some output fault *f*. First assume G is an AND gate. Then *f* cannot be the *s-a-0* fault, since this is equivalent to any input *s-a-0.* In addition, *f* cannot be the *s-a-1* fault, since this dominates any input *s-a-1.* Hence, such a fault *f* does not exist. A similar argument holds if G is an OR, NAND, or NOR gate. Therefore, all the internal faults in C are detected [1].

Theorem 2.3.4.2

A set of test vectors that detect all single stuck-at faults on all primary inputs and all fanout branches of a combinational logic circuit will detect all single stuck-at faults in that circuit [1].

Proof: We can also apply similar contradiction proof as done in theorem 2.3.4.1 [1].

When we apply these theorems, faults of a circuit are grouped into sets of equivalent faults. From each of these equivalence fault sets, one fault is then selected to form an equivalence collapsed set.

2.3.5. Dominant Faults

A fault is said to dominate another fault if all tests for the second fault detect the first fault. i.e. if all tests of some fault F1 detect another fault F2, then F2 is said to dominate F1. If we take for example an AND gate as shown in Figure 8, a s-a-1 fault on the output line of the gate dominates a s-a-1 fault on any input its line.

Hence dominant fault collapsing imply that if fault F2 dominates F1, F2 can be removed from the fault list.

In relation to equivalence fault collapsing, dominance fault collapsing also help to reduce stuck-at faults in any combinational circuit.

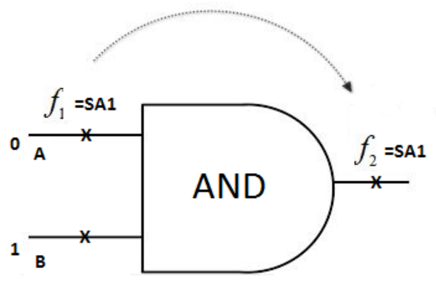


Figure 8: Dominant fault in AND gate

2.4. Conclusions

The main objective of this chapter was to give background information or topics relating to the focus of this thesis. The chapter dealt with a description of the concept of digital testing, In this chapter, I also explained the common fault models and a discussion on fault collapsing at gate level with respect to equivalent and dominant fault was done. Finally, we discussed about test generation and fault diagnosis, here we showed the difficulties encountered, one of which is the complexity and size of circuits, which produces a huge set of possible faults.

The performance of the test generation, fault simulation and fault diagnosis depend on the size of the model. However, the contribution of this thesis work is to present a way to improve the performance of these tasks and reduce the cost. A method, which uses SSBDD synthesis and topological analysis, was proposed.

Reducing the number of faults reduces the size of fault dictionary, improves test generation and fault simulation performance. The focus of this research is towards reducing the number of faults in a circuit thereby reducing the size of the fault dictionary during diagnsis, improving test generation and fault simulation performance.

The next chapter will discuss the method in detail and its opportunities in terms of reducing cost and resource at no trade off to quality.

3. Double Phase Fault Collapsing

In this chapter, the new structural fault collapsing method with linear algorithmic complexity is proposed. The method is based on the two-phase topology analysis of the circuit description.

The first section of this chapter discusses the first phase of the fault collapsing which is carried out on the gate level during superposition of Binary Decision Diagram(BDD) of logic gate [27-29].

In the second section of this chapter, I will give an overview of SSBDD and discuss the second phase of the procedure which is carried out by topological analysis of SSBDD. An overview of SSBDD will be appropriate in this section since the algorithm and method proposed involves a topological analysis of SSBDD.

The third section of this chapter will introduce the concept and develop higher and lower bounds of the size of collapsed faults.

3.1. Structurally Synthesized BDDs (SSBDD)

SSBDD is a planar, acyclic BDD that is obtained by superposition of elementary BDDs for logic gates [26]. SSBDDs were firstly proposed by [25].

BDDs were first introduced for logic simulation in [27] and then for test generation in [25], [28-29]. Several years after the proposal of BDDs, Bryant proposed a new data structure called Reduced Ordered BDDs (ROBDDs). The ROBDD proves the uniqueness of Boolean functions. Bryant's new data structure showed the simplicity of graph manipulation. Equivalence of any two Boolean function can be easily checked by comparing their ROBDDs. The image below shows a BDD and ROBDDs of a Boolean expression given as:

$$\mathbf{F = a \neg c + a \neg b \neg c + b \neg c}$$

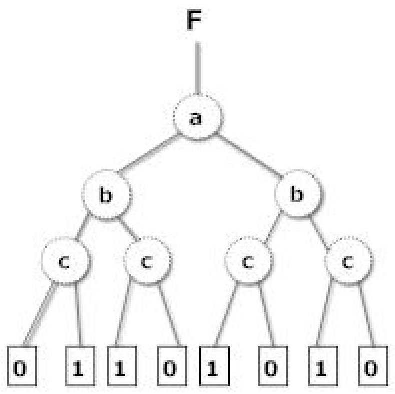


Figure 9: Binary Decision Diagram

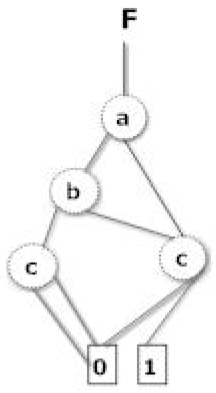


Figure 10: Reduced Order Binary Decision Diagram

In the ROBDD model, all the redundant nodes for cases where both edges point to the same node and all equivalent sub-graphs are shared.

This model however suffers from memory explosion problem which however hindered it from being widely adopted in large designs. It cannot also be used as a model for methods that require structural information about the design like representation of faults directly in the model.

The limitation of ROBDD has been looked into with the proposal of the SSBDD. In terms of size, SSBDD model is linear in respect to the circuit size whereas ROBDD can be of exponential size.

In [21], [30-32], SSBDDs were proposed for direct modeling of the structural aspects of circuits. This is in contrast to the traditional BDD which only represents logical functions. However, the most important difference between them is in their method of creation. SSBDDs are generated by superposition of BDDs while traditional BDDs are generated by Shannon's expansions that extract the function of the logic.

One other benefit of the SSBDD over the traditional BDD known as the linear complexity of the former is that in the SSBDD model, a circuit is represented as a system of SSBDDs, where for each fan-out-free region (FFR) of the circuit, a separate SSBDD is generated. While in the case of traditional BDD, the complexity is exponential.

Of interest to this research is one of SSBDD feature which is its stuck-at fault collapsing capability [33], eliminating the task of checking whether a fault belongs to the collapsed list or not. The application of SSBDD are: fault modeling (fault set collapsing), signal path modeling, structure modeling (back-annotation from SSBDDs), delay modeling, hazard modeling, test generating for the structural faults, fault simulation (test analysis) and fault diagnosis (fault location). All these application possibilities cannot be done by traditional BDDs.

Definition 3.1.1: A BDD is called SSBDD, if there is a one-to-one correspondence between non-terminal nodes of the BDD and signal paths in the combinational circuit. Non-terminal nodes of an SSBDD are labeled by subscripted input variables, which can be inverted or not. In fact, SSBDD model is further defined by construction [33].

Figure 11 shows an example of a combinational circuit with a single output y and no internal fan-outs. We represent the fan-out-free region (FFR) of the circuit by a single SSBDD in Figure 12.

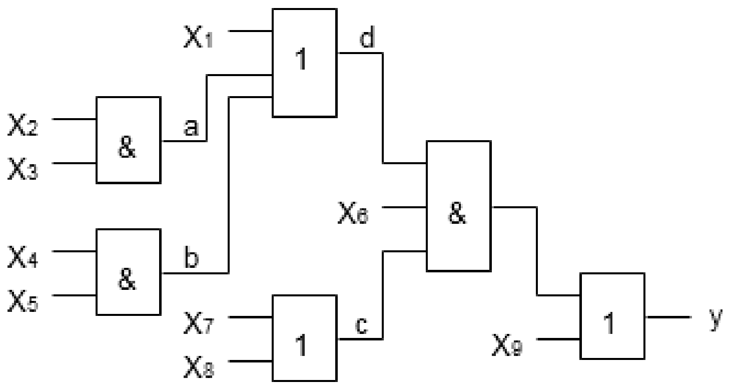


Figure 11: Combinational Circuit with Single Output

Each labeled section of the circuit maps to its representation on the SSBDD. For instance, the part labeled **(a)** in the SSBDD representation shows the AND of inputs x_2 and x_3 ***(x_1x_2)***, while the region labeled **(d)** ***(x_1v (x_2x_3) v (x_4x_5)).***

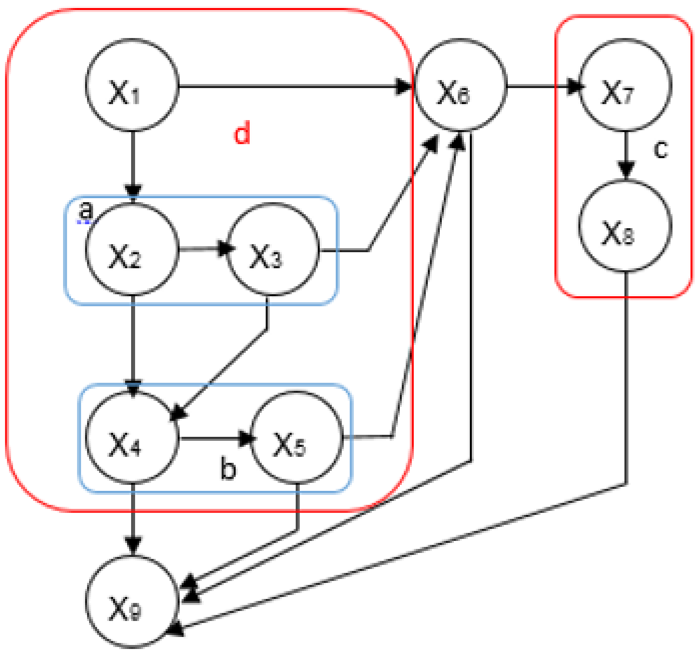


Figure 12: SSBDD for the circuit in Figure 11

SSBDDs are constructed from a FFR of a combinational circuit beginning usually from the output. The output could be either the primary output or a fan-out point of the circuit. Each logic gates are replaced by their elementary BDDs in a repeatedly in similar manner.

Elementary BDDs for logic gates are shown below.

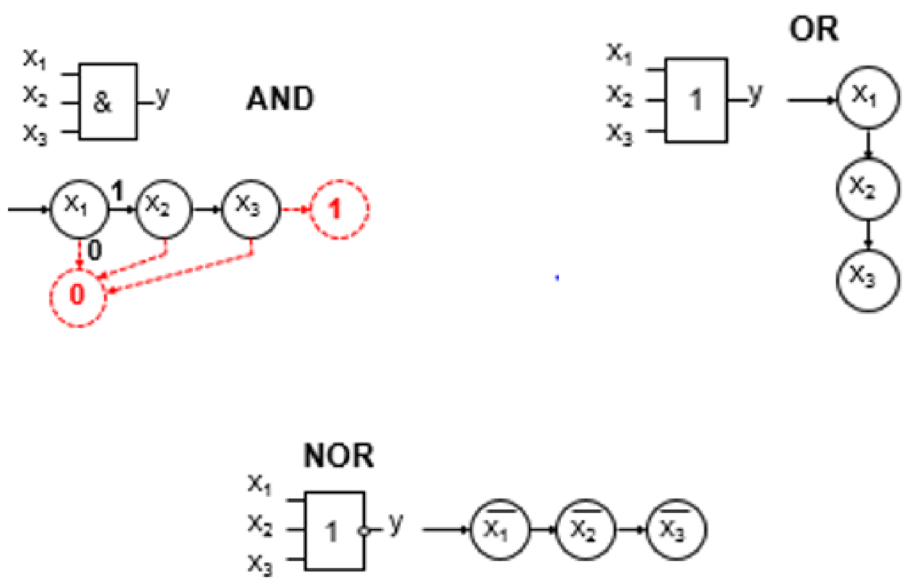


Figure 13: Elementary BDDs for logic gates

After each logic gates in the circuit are recursively substituted by their elementary BDDs, the next step involves using superposition procedure to combine them into a single graph. The procedure of superposition terminates in those nodes, which represent a primary input or a fan-out branch of the circuit [34].

The figures below demonstrate in steps the superposition procedure for construction of the circuit in Figures (I – V).

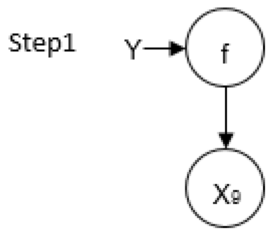


Figure 14: Superposition procedure (I)

In step 1, applying the rule of the elementary BDD for AND of f and x_9. The resulting SSBDD is shown above.

In step 2, the AND logic gate with 3 inputs **d**, **x6** and **c** are substituted by their elementary BDDs and combined to form a single graph by superposition procedure.

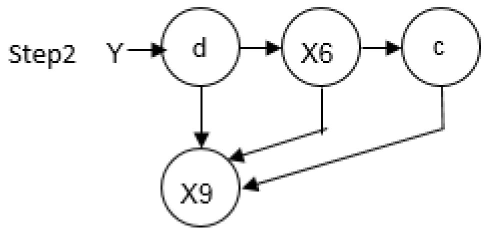


Figure 15: Superposition procedure (II)

In step 3, the OR logic gate with 3 inputs $\mathbf{x_1}$, **a** and **b** are substituted by their elementary BDDs and combined to form a single graph by superposition procedure.

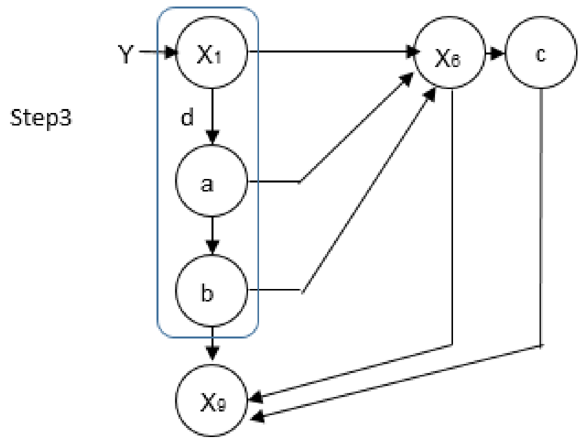


Figure 16: Superposition procedure (III)

Step 4 follows a similar procedure. Here, two AND logic gate with 2 inputs each **x_1**, **x_2** and **x_3**, **x_4** respectively are substituted by their elementary BDDs and combined to form a single graph by the same superposition procedure that is applied in the previous steps.

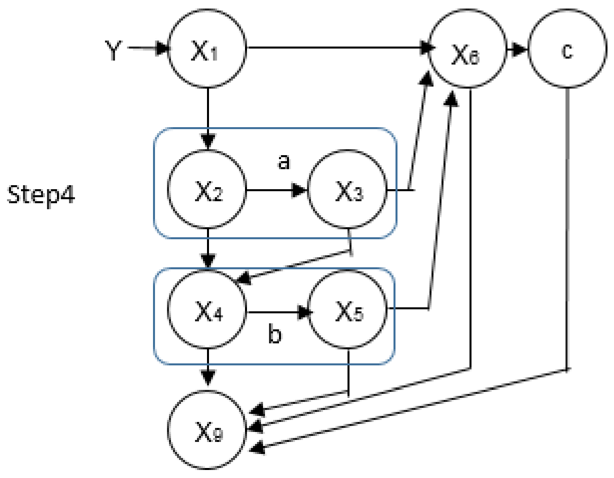


Figure 17: Superposition procedure (IV)

Step 5 is the final step where an OR logic gate with 2 inputs each **x_7** and **x_8** are substituted by their elementary BDDs and combined with the graph from the step 4 to form a single graph by the same superposition procedure that was applied in the previous steps.

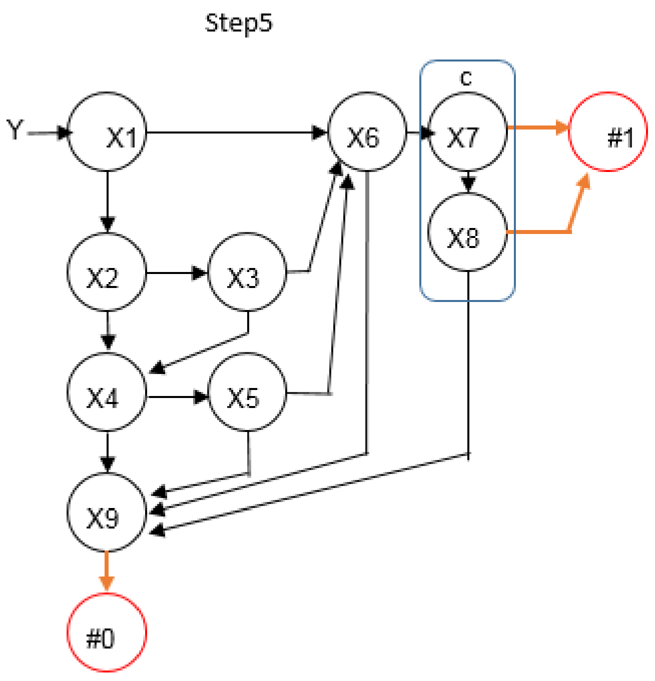


Figure 18: Superposition procedure (V)

Let us take example of a circuit with internal fan-out. The Figure 19 below shows an example of a circuit with internal fan-out. In this case, there are two FFR in the circuit and if we note the property of SSBDDs where for each fan-out-free region (FFR) of the circuit, a separate SSBDD is generated. Figures 20 and 21 shows the SSBDDs for each FFR on the circuit.

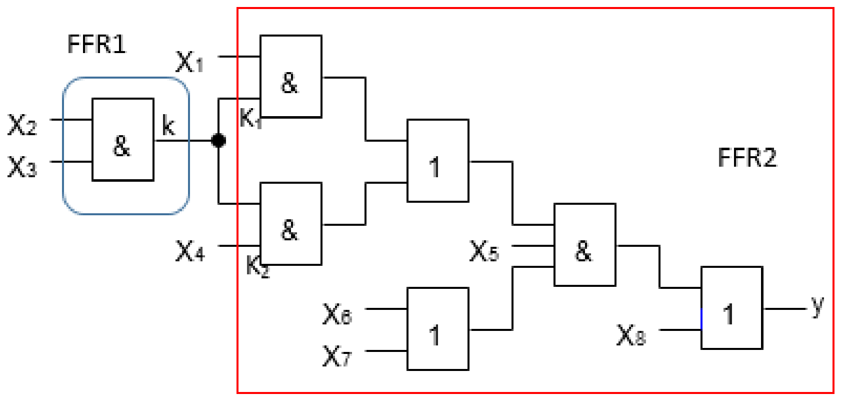


Figure 19: Combinational circuit with internal fan-out

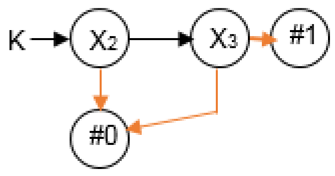


Figure 20: SSBDD for FFR1 in Figure 19

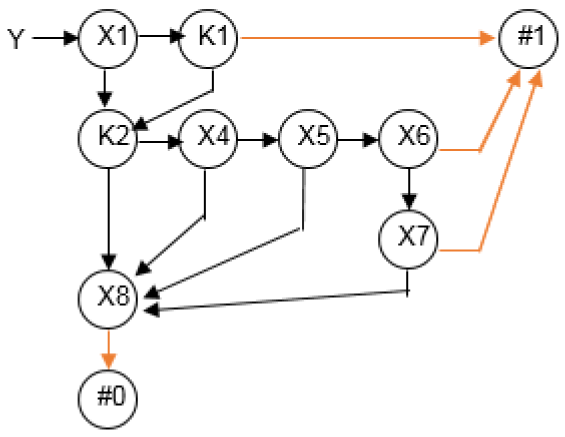


Figure 21: SSBDD for FFR2 in Figure 19

Figure 22 shows another example of a combinational circuit also with a single output y and no internal fan-outs. We represent the Fan-out-free Region (FFR) of the circuit by a single SSBDD as in Figure 23.

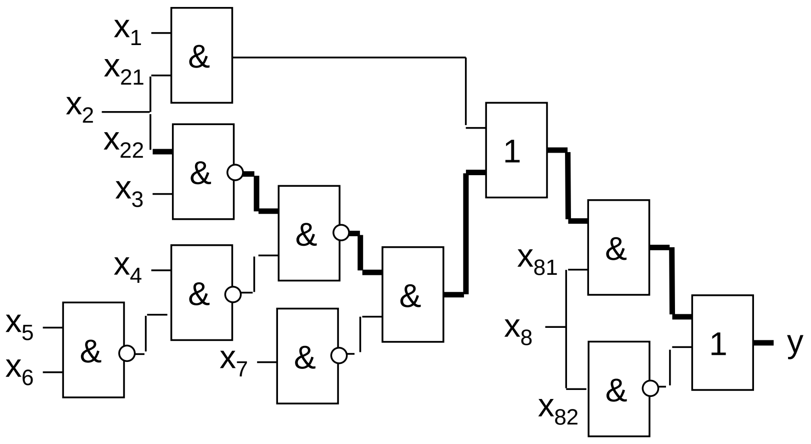


Figure 22: Combinational Circuit with Single Output y

In the SSBDD, internal nodes are labeled by the input variable of the corresponding FFR. Each of the fan-out stem variables is differentiated from the fan-out branch variable by another numeric subscript value. For instance, if we have a fan-out branch $\mathbf{x_2}$ with two fan-out stems, we can represent each of the stem as $\mathbf{x_{21}}$ and $\mathbf{x_{22}}$ respectively.

We can calculate the output signal of the circuit at any given input pattern by simply traversing the SSBDD graph from the root node up to the terminal node guided by the values of the node variables.

In the event that we concur with the idea that each node during simulation are exited to the right if the node variable has value 1, or downwards if the node variable has value 0. This then means that we would not need to label the edges of the graph by values. Entering the terminal node #1 as the result of graph traversing will mean the result of simulation as $y = 1$, and entering the terminal node #0 will mean $y = 0$.

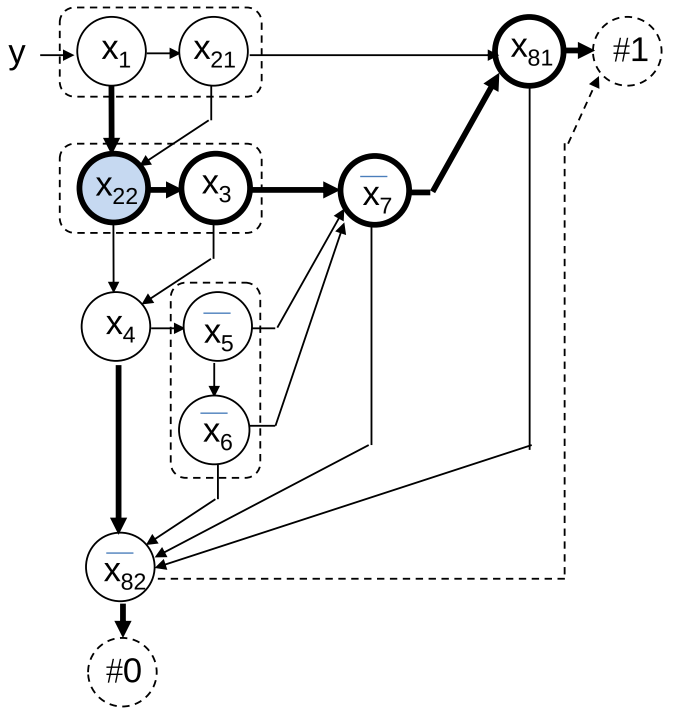


Figure 23: SSBDD for the circuit in Figure 22

If we have input pattern X^t as given in Figure 24, the output of y for circuit given in figure 22 with function $y = f(X)$ will be 1. During simulation of the pattern with SSBDD, the following nodes are traversed: x_1, x_{22}, x_3, $\neg x_7$, x_{81}, #1.

X	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
X^t	0	1	1	0	-	-	0	1	1
Tested: $x_{22} \equiv 0$, $x_3 \equiv 0$, $x_7 \equiv 1$, $x_{81} \equiv 0$									

Figure 24: Test pattern for detecting the faults SAF/0 or SAF/1

Definition 3.1.2. Let us call a path $L(a,b)$ in the SSBDD between two nodes a and b, activated by a given input pattern X^t, if by traversing the graph under guidance of X^t, the node b will be reached from a.

Each node in the SSBDD represents a signal path in the related circuit. For example, the SSBDD in Figure 23 consists of 10 nodes where each of them represents a corresponding signal path of the total ten (10) paths in the circuit in Figure 22. The one-to-one mapping between the nodes in the SSBDD and the signal paths in the related circuit is the result of the SSBDD synthesis from the circuit by superposition of BDDs [31]. A node variable in the SSBDD is inverted if the corresponding signal path of the circuit has odd numbers of inverters.

Stuck-at faults in SSBDDs are modeled at nodes and each node corresponds to a distinct path in an FFR [35]. As explained above in relation to the example of the SSBDD in Figure 23, we can generally say that the number of signal paths in an FFR of a combinational circuit is proportional (either less than or equal to) to the number of lines between the gates in the circuit. This consequently forms the basis for initial collapse of fault during SSBDD formation as the one-to-one mapping between nodes in the SSBDD and the signal path in the circuit produces fault collapsing as a by-product of the SSBDD synthesis [7].

For example, the initial number of gate level SAF faults in Figure 22 is 46 (two faults for each of 23 lines). During the first step of SSBDD synthesis, we reduced the number of representative faults from 46 up to 20, (two faults for each of 10 nodes in the SSBDD model). This implies that the procedure of SSBDD synthesis through superposition of BDDs compacts the fault location in a combinational circuit by about 50 percent. In the case of the circuit in Figure 22, there were 23 fault locations. By superposition, this was compacted to 10, which is about 56% in this case. This implies that the process of generating SSBDD through superposition of BDD implicitly performs collapsing of fault.

Stuck-at Faults on SSBDD

We can also apply the concept of stuck-at fault discussed previously in this work as it applies to logic gates in a combinational circuit to SSBDD model. In the gate-level descriptions of a circuit, stuck-at faults are modeled at the interconnections between the gates but are modeled in SSBDD at nodes.

Stuck-at fault modeling on SSBDD is synonymous to explicit fault collapsing using dominance relations along the signal path in a circuit. However, experiments have shown that SSBDD achieves in average even 2 % better compaction of the fault list than the traditional approach, reducing the fault lists in average about 1.5 times[33].

For example, the node x_{22} in the SSBDD represents the path from x_{22} to y in the circuit shown by bold lines in Figure 22. On the other hand, the faults SAF y/0 and SAF y/1 dominate the faults SAF x_{22}/0 and SAF x_{22}/1, respectively. The same dominance relation stands for all the faults along the bold path from x_{22} to y, regarding the related faults at x_{22}.

From this dominance relation, it results that all the faults along the signal path from x_{22} to y, except the faults SAF x_{22}/0 and SAF x_{22}/1, will be collapsed. The latter faults will form the final representative (collapsed) fault set for the signal path from x_{22} to y. However, these faults are represented in the SSBDD as the two faults of the node x22. This fault collapsing effect is similar to that of the fault folding presented in [36] for structural collapsing faults.

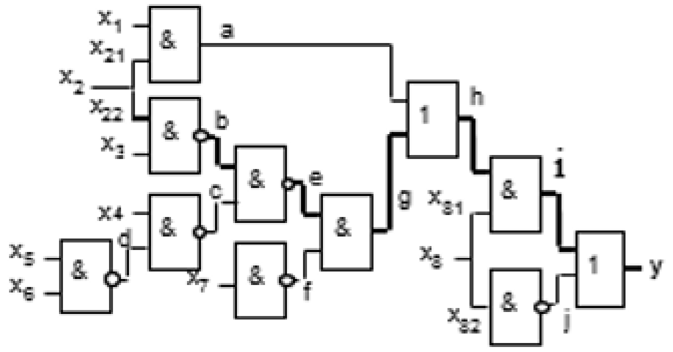


Figure 25: Combinational circuit with labeled interconnecting lines

In Figure 25 above, we labeled each interconnecting line as a-i. We would try to explain how dominance relation in SSBDD operates. In this case, the SSBDD in Figure 23 still suffices for the fig above with interconnecting lines labeled.

As mentioned above, during SSBDD synthesis we reduced the number of representative faults from 46 up to 20 (two faults for each of 10 nodes in the SSBDD model). Applying the rule of dominance for instance on the circuit, it would mean that all the fault along the signal path from x_{22} to y (b, e, g, i, and y) will be collapsed. This is because x_{22} s-a-0 will be dominated by b s-a-0, e s-a-0, g s-a-0, i s-a-0 and y s-a-0. The same applies to x_{22} s-a-1, it will be dominated by b s-a-1, e s-a-1, g s-a-1, i s-a-1. By rule of dominance, all these dominating faults will be collapsed.

The table below shows Faults dominating the SSBDD node faults of the circuit.

Node	**Fault**	**Dominating Faults**
X1	S-A-0	SAF a/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF a/1, SAF h/1, SAF i/1, SAF y/1
X21	S-A-0	SAF a/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF a/1, SAF h/1, SAF i/1, SAF y/1
X22	S-A-0	SAF b/0, SAF e/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF b/1, SAF e/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X3	S-A-0	SAF b/0, SAF e/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF b/1, SAF e/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X4	S-A-0	SAF c/0, SAF e/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF c/1, SAF e/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X5	S-A-0	SAF d/0, SAF c/0, SAF e/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF d/1,SAF c/1, SAF e/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X6	S-A-0	SAF d/0, SAF c/0, SAF e/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF d/1,SAF c/1, SAF e/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X7	S-A-0	SAF f/0, SAF g/0, SAF h/0, SAF i/0, SAF y/0
	S-A-1	SAF f/1, SAF g/1, SAF h/1, SAF i/1, SAF y/1
X81	S-A-0	SAF i/0, SAF y/0
	S-A-1	SAF i/1, SAF y/1
X82	S-A-0	SAF j/0, SAF y/0
	S-A-1	SAF j/1, SAF y/1

Table 3: Faults dominating the SSBDD node faults

The procedure of SSBDD synthesis can be regarded as the first phase of fault collapsing for the given circuit. In this first phase the table below shows fault collapsed through the process of SSBDD synthesis for ISCAS’85 circuits.

Circuit	Number of Faults	# Repr. Faults	Fault collapse
		SSBDD Synthesis	
c1355	2710	1618	1092
c1908	3816	1732	2084
c2670	5340	2626	2714
c3540	7080	3296	3784
c5315	10630	5424	5206
c6288	12576	7744	4832
c7552	15104	7104	8000

Table 4: Fault collapsed by SSBDD synthesis for ISCAS'85 circuits

In the next section we will present a method for additional fault collapsing directly on the SSBDD model. This is regarded as the second phase of the full fault collapsing procedure based on SSBDDs.

Summary

In this section we have seen the major differences between SSBDDs and traditional BDDs. It was mentioned that SSBDDs have linear complexity while BDDs have exponential complexity and apart from this, traditional BDDs do not represent the structure, they only represents logical functions, hence they cannot help in fault collapsing.

Each node in the SSBDD represents a signal path in the related circuit and the one-to-one mapping between the nodes in the SSBDD and the signal paths in the related circuit is the result of the SSBDD synthesis from the circuit by superposition of BDDs. This already results in the fault collapsing in the first stage of the method proposed.

3.2. Fault Collapsing on the SSBDD Model

This section describes the second phase of fault collapsing method proposed in this work. In this case, haven discussed the fault collapsing feature of SSBDD synthesis, it is however important to stress that topological analysis of SSBDDs at a higher macro level could still collapse more faults. This is possible because SSBDD synthesis helps construct a higher macro compacted model of the circuit.

Let us consider a test pattern generation for the fault SAF $x_{22}/0$ with SSBDD in Figure 23. As explained in [31], to test a node in the SSBDD we would have to activate three paths in the SSBDD modeled in Figure 23. The first path is $L(x_1, x_{22})$ which is from the root node x_1 to x_{22}, the second path is $L(x_{22}, \#1)$ from x_{22} to the terminal node #1, and the last path in this case is $L(x_4, \#1)$ which is from x_4 to #0. The node x_4 is the neighbor of x_{22} but does not belong to the path $L(x_{22}, \#1)$. As mentioned above, x_4 belongs to the path $L(x_4, \#1)$.

To further our discussion, we need to remember that Figure 24 shows the pattern X^t which activates all these paths and make testing of a node in the SSBDD possible. Bold lines in Figure 23 show the activated paths in the SSBDD.

We would also call a path that is activated by a given pattern and terminates in terminal node #1 as 1-path and a path that is activated by a given pattern and terminates in terminal node #0 as 0-path. Going further, we would call all the nodes that are traversed along the activated 1-path in direction 1 1-node and same goes for all the nodes that are activated along an activated 0-path in direction of 0 we would call it 0-node. Note that the direction 1 represents right direction while direction 0 represents the downward direction.

From the Figure 26 below, we highlighted the 1-path in red and the 0- path in blue. This implies that in this example, 1-nodes are X_{22}, X_3, $\neg X_7$ and X_{81} and the 0-nodes are X_4 and $\neg X_{82.}$

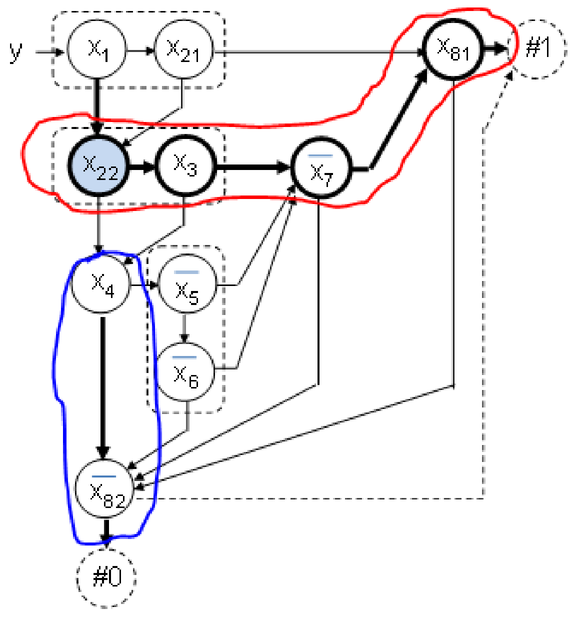


Figure 26: Showing paths and nodes

According to [26], if a test vector X^t activates in SSBDD a 0-path or 1-path, then only 0-nodes or 1-nodes have to be considered as candidate fault sites. This means that in the case of the SSBDD modeled above, the analysis of the path shows that all nodes except x_1 may be qualified as candidate fault sites. This property is essentially considered in order to speed up fault simulation. Upon further analysis, we would be able to confirm that all these candidate faults are detectable by the given test pattern X^t (in Figure 24).

The introduction of node related fault identification instead of node variable related fault identification will further simplify the discussion. In this case, we can represent cases of inversion in node variables uniformly to cases where nodes are not inverted. i.e. we can handle the fault cases at SSBDD nodes uniformly, independently of whether the node variable is inverted or not inverted. For example, we can refer to SAF x_7/1 in Figure 23 as SAF $\neg x_7$/0.

This means that all the 1-nodes or 0-nodes denoted by the variable ***a*** either inverted or not are the candidates for testing the faults SAF a/0 (or SAF a/1).

Theorem 3.2.1. The faults at two connected by an edge nodes ***a*** and ***b*** are equivalent iff the following two conditions are satisfied: (1) the nodes have the same neighbour *c*, and (2) the node ***b*** has a single incoming edge from ***a***.

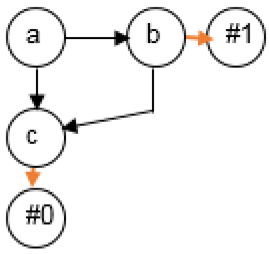


Figure 27: Equivalent Faults

The condition that the node a and b must have the same neighbour refers to the fact that both nodes can be tested by the same test pattern which activates the paths L(Root, a), $L(a,\#e)$ where $e \in \{0,1\}$, and the path $L(c,\#(\neg e))$. The second condition refers to the fact that this test pattern is the only one which can test both of the node faults SAF a/$\neg$ *e* and SAF b/$\neg$ *e*.

For example, the faults SAF a/0 and SAF b/0 are equivalent and one of them can be collapsed. The same principle can also be applied to the example in Figure 23. Here, the faults SAF x_{22}/0 and SAF x_3/0 are equivalent, and one of them can be collapsed.

Another property of SSBDD, which will help us achieve more fault collapse, is that SSBDDs have a Hamiltonian path through all the nodes except the terminal nodes #0 and #1. The Hamiltonian path determines the unique ranking order of the nodes. According to Figure 27 above, the nodes ***a*** and ***b*** are in the relationship $\boldsymbol{a} < \boldsymbol{b}$ if the node ***a*** will be traversed before ***b*** along the Hamiltonian path.

Theorem 3.2.2. The fault SAF *b*/0 dominates SAF *a*/0 (or SAF *b*/1 dominates SAF *a*/1) iff the following conditions are satisfied: (1) there exists a single 1-path (or a single 0-

path) through the nodes for detecting both of these faults, (2) ***a*** < ***b***, and (3) the node *b* has more than 1 incoming edges.

It follows from theorem 3.2.2 that The fault SAF *a*/0 dominates SAF *b*/0 (or SAF *a*/1 dominates SAF *b*/1) iff the following conditions are satisfied: (1) there exists a single 1-path through the nodes for detecting both of these faults, (2) ***a*** < ***b***, and (3) the node ***a*** can be tested by activating another path where ***b*** is not tested.

The first conditions demand that these faults can be detected by a single test pattern. This condition is same as the condition of the equivalency as explained in theorem 3.2.1. The second condition demands that there will be no other path for testing ***a*** and not testing ***b***. The third condition is needed to give the possibility to test ***b*** and not to test ***a***. From satisfying these conditions, it follows that any test for ***a*** must detect the related fault as well at ***b***. Hence, the fault at ***a*** is dominated by ***b***. If the third condition is not fulfilled, the related node faults at the nodes ***a*** and ***b*** are equivalent.

After some faults have been collapsed through synthesis of BDD, Theorems 1 and 2 are checked on the remaining nodes in pairs and upon this check we can determine more faults that could be collapsed or not. Based on the whole procedure explained in the previous section and this section. The combinational circuit in Figure 22 has initial gate level SAF of 56. i.e. two faults for each of the 23 lines. When we SSBDD synthesis, we reduced the faults to 20 as shown in Table 3. This implies that 36 faults were collapsed at this stage. The second procedure of fault collapsing implemented in this work further collapsed the faults more by 7. In the entire process, about 43 faults were collapsed (36 from stage 1 and 7 from stage 2) leaving the fault remaining reduced to 13.

The Table 5 below shows the faults collapsed during the second procedure of fault collapsing proposed with comments.

Node	**Collapsed**	**Comments**
x_1	SAF $x_1/0$	Equivalent with $x_{21}/0$
x_{21}	-	Both faults retained
x_{22}	SAF $x_{22}/0$	Equivalent with $x_3/0$
x_3	-	Both faults retained
x_4	SAF $x_4/0$	Dominates $\neg x_5/0$, $\neg x_6/0$
$\neg x_5$	SAF $\neg x_5/1$	Equivalent with $\neg x_6/1$
$\neg x_6$	-	Both faults retained
$\neg x_7$	SAF $\neg x_7/0$	Dominates $x_{22}/0$, $\neg x_5/0$, $\neg x_6/0$
x_{81}	SAF $x_{81}/0$	Dominates $x_{22}/0$, $\neg x_5/0$, $\neg x_6/0$
$\neg x_{82}$	SAF$\neg x_{82}/1$	Dominates $x_{81}/1$, and others

Table 5: Result of Fault collapsing for SSBDD in Figure 23

Summary

In this section, we described the second phase of fault collapsing method proposed in this work. The process of SSBDD synthesis in the first phase helps in the construction of a higher macro compacted model. Further analysis on the SSBDD model through the application of the rule of dominance and equivalence could help us achieve more fault collapsing. This we described in this section as topological analysis of the SSBDD model. In addition, we proved with examples that more faults were collapsed on the circuit beyond the ones achieved through SSBDD synthesis. The impact of collapsing more fault in a circuit enhances the performance and cost of test generation and fault simulation. It also reduces the size of the fault dictionary needed during fault diagnosis. This is what this research is aimed at achieving.

3.3. Lower and Higher Bounds For Fault Collapsing

Beside the method of fault collapsing proposed in this work, another contribution that is laudable is the proposal of a method of estimating the achievable size of the collapsed fault set for the given SSBDD. This concept is referred to as lower and higher bounds for fault collapsing.

To carry out this estimation, we would denote the number of all nodes in the SSBDD model of a combinational circuit as N and then denote the number of collapsed fault as C. It should be noted that N represents the result of the first phase of the fault collapsing done during SSBDD synthesis.

Given that the number of all SAF in the SSBDD model with N nodes is 2N, after fault collapsing in the model, the number of representative faults in the SSBDD model will be expressed as $R = 2N - C$. Fault collapsing effect achieved in the SSBDD model can be expressed as $R/2N$.

Theorem 3.4.1

The effect of fault collapsing in a given SSBDD model will be always in the boundary $1/2 < R/2N \leq 5/6$.

As a way to proof Theorem 3.4.1, we can conclude that any tree-like circuit with N input which could be represented by SSBDD with N nodes. Let us take for example the simplest form is either a single gate AND or OR gate with N number of inputs as shown in Figure 28 below. For such gates, we can collapse N-1 SAF/0 or SAF/1 faults. Hence $C = N-1$.

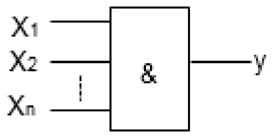


Figure 28: Single AND gate with N inputs

Substituting the value of C in the representation of representative faults after collapsing given as: R = 2N – C. We will however have R = 2N – (N -1). By solving the equation, R becomes N + 1. If N increases, we can determine the lower limit for R/2N as follows:

$$\lim_{n\to\infty}\frac{R+n}{2N+2n}=\lim_{n\to\infty}\frac{(N+1)+n}{2N+2n}=\lim_{n\to\infty}\frac{n}{2n}=\frac{1}{2}$$

We have therefore proved the above theorem 3.4.1 that the ratio of R/2N for any fault collapsed on a circuit is always greater than 1/2.

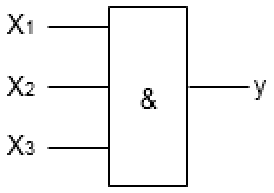


Figure 29: Single AND gate with 3 inputs

For example, a simple single AND gate with 3 primary input in Figure 29 above. The SSBDD model of the gate will also have 3 nodes. In this case, 2*N* = 6 and knowing that we can collapse N-1 SAF/0 or SAF/1 faults. i.e. 3-1 = 2. The number of representative faults will be R = 6-2 = 4, and *R*/2*N* = 4/6. Hence, 1/2 < 4/6 ≤ 5/6.

On the contrary, if we partition the set of N input for more than one gate in the tree-like circuit. The effect of this will be a reduction in the total value of C by one fault per added gate. This consequently increases the value of R/2N ratio.

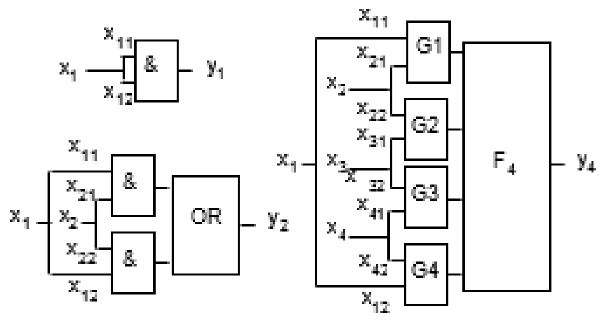


Figure 30: Tree-like circuits with increasing complexity

Let us take for example a single-input logic gate y_1 with N = 3 nodes in Figure 30 which may represent either an inverter or a buffer. The SSBDD model of this gate will have as well 3 nodes, which represent the fan-out stem and its 2 branches. Hence, 2N = 6. There are two equivalent SAF/0 faults on the branches (or SAF/1 in case of OR-gate) where one of them can be collapsed. Hence, the number of representative faults will be R = 6-1 = 5, and R/2N = 5/6.

We can apply the same logic to the tree-like circuit y_2 with two input gates in Figure 30, each of them having 2 input nodes, and with 2 fan-out nodes. The number of representative faults for this circuit is R = 2N = 2(6) = 12. There are again two equivalent SAF/0 faults for each AND-gates (or SAF/1 in case of OR-gates) where one of them can be collapsed each. Hence, the number of representative faults after fault collapsing will be R = 2N – 2 = (12 -2) =10, and again we get R/2N = 5/6.

Circuit y4 in Figure 30 shows a circuit y_4 which can be used to illustrate how we can expand the series of two circuits y_1 and y_2 into a series of expanded circuits y_1, y_2, y_3, y_4, … y_n which will consist of an input circuit IN_n as a chain of n 2-input gates, and an arbitrary tree-like circuit F_n. In each such a circuit, the ratio R/2N = 5/6 remains constant. In IN_n for each gate, only a single fault can be collapsed.

We can easily observe that any structural change inside the sub-circuit F_n will not change the ratio R/2N = 5/6. This is because all the faults in Fn will dominate the faults in INn. However, by adding n = 1, 2,… non-fan-out inputs to the sub-circuit IN_n we get R/2N*=

$(R+n)/(2N+2n)$, and by adding $n = 1,2,\ldots$ fan-out inputs with 2 branches to IN_n we will get $R/2N^{**} = (R+2n)/(2N+6n)$. Each addition of a fan-out branch is equivalent to the case of adding a single input node in the sense of fault collapsing. From above it follows:

$$R/2N^{**} < R/2N^{*} < R/2N \leq 5/6$$

Corollary 1: From Theorem 3.4.1, it directly follows that for any SSBDD with N nodes, the number of collapsed faults $C = 2N - R$ will belong to the interval $N/3 \leq C < N$. Hence, $N/3$ will serve as the lower pessimistic bound for the number of collapsed faults.

If we take example of the circuit in Figure 22 with SSBDD in Figure 23. The SSBDD has 10 nodes which imply that N = 10. From Table 5 where the result of Fault collapsing for SSBDD in Figure 23 was presented, we could see that there were 7 faults collapsed which implies that C = 7. Hence corollary 1 which implies that $N/3 \leq C < N$ is satisfied as $10/3 \leq 7 < 10$. The same follows if we apply Theorem 3.4.1. here for instance, N=10, meaning that 2N = 20. Given also that C = 7. R which is 2N – C is therefore equal to 20 – 7. The ratio of R/2N is hence given as 13/20 which still satisfies the range $1/2 < R/2N \leq 5/6$ ($1/2 < 13/20 \leq 5/6$).

Summary

We proposed in this section a method of estimating the achievable size of the collapsed fault set for the given SSBDD. This concept is referred to as lower and higher bounds for fault collapsing. Knowing the lower bounds for the given circuit can help in estimating the cost for further fault simulation and test generation. This can also help us in determining the expected gain of fault collapsing.

3.4. Algorithm Description

We propose a fault collapsing algorithm that is based on topological analysis of SSBDD model below. The input to the algorithm is SSBDD model for a given circuit and the output is a set of collapsed faults.

We define notations used in the algorithm as follows:

- *m - number of the current node*
- *n - number of the next node to m, n = m + 1*
- *n* - next node to n, n* = n + 1*
- *M - number of all nodes*
- *FI(n) - Fanin flag for the node n*
- *FI(n) = 1, If the node n has entries from more than one nodes which has not build a group*
- *d(m) - direction from node m to its neighbor n*
- *!d(m) - inverted direction d(m)*
- *m(d) - neighor of the node m in direction d*
- *m(d) = Ø, if the neighbor of m is terminal node (#0, #1)*
- *IN(m) - direction of input edges to node m*
- *C(m) – attribute of fault collapsing at node m*
- *C(m) ϵ {0,1,Ø} – possible coding {01, 10, 00} 0 - SAF0, 1 - SAF1, Ø - no fault collapsing*

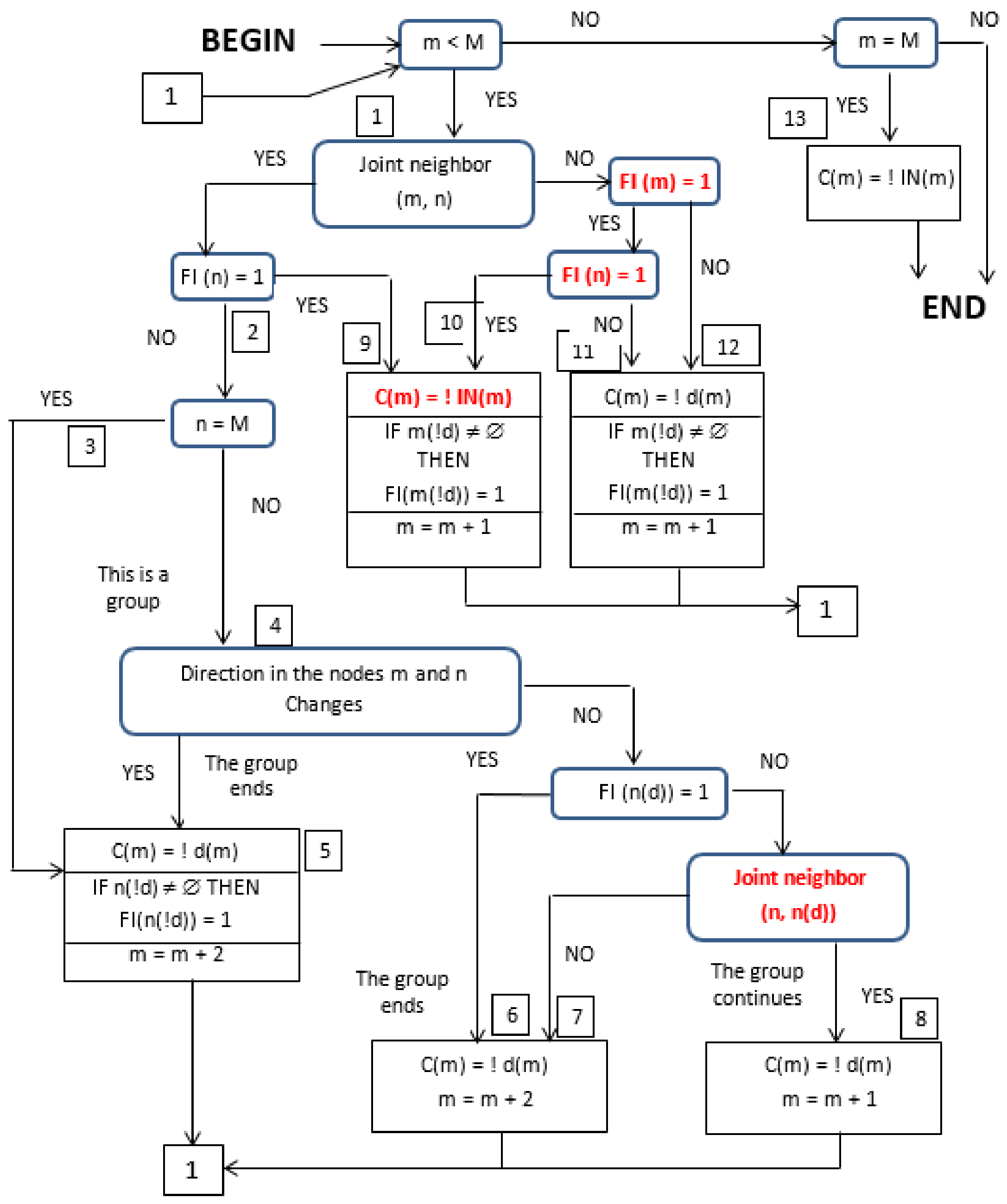


Figure 31: Proposed Method Algorithm

Let us explain the working principle of the algorithm by tracing some paths through it dictated by selected typical node structures of SSBDD models.

The main idea of the algorithm is in walking through the SSBDD model and analyzing during this walk the consecutive pairs of nodes to decide if one of them can be collapsed or not. In the following, we present the typical cases how different pairs of nodes can be embedded in SSBDDs, and how they will be handled by the algorithm.

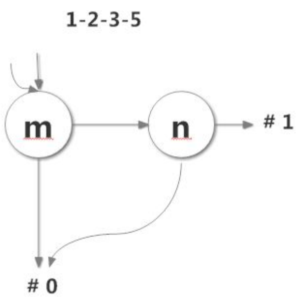


Figure 32: Walkthrough Example circuit (I)

Consider as an example a part of an SSBDD model in Figure 32. The numbers on the top of the model refer to the nodes (steps) in the algorithm, traced for this pair of nodes m and n (an extracted part of SSBDD model). Applying the algorithm, at step 1, equivalence is checked (m and n have neighbours). At step 2, the condition that **n** has a single incoming edge from a is checked(condition for dominance). The condition for equivalence is true while the condition for dominance is false. The condition 3 checks whether n is a terminal node and the group ends or not. In the case of the circuit above, the condition is true which means that the group ends. Equivalent check rule says that the node **m** can be collapsed The node is collapsed at SAF 0 or 1 depending on the inverse direction of **m.** Note that a group will end when the direction of m and n changes.

Similar walkthroughs for the node pairs (m,n) are illustrated in Fig.33-35.

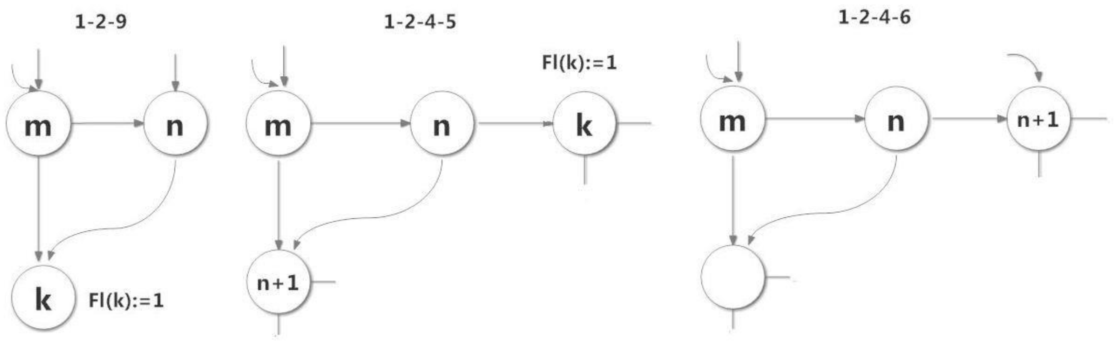


Figure 33: Walkthrough Example circuit (II)

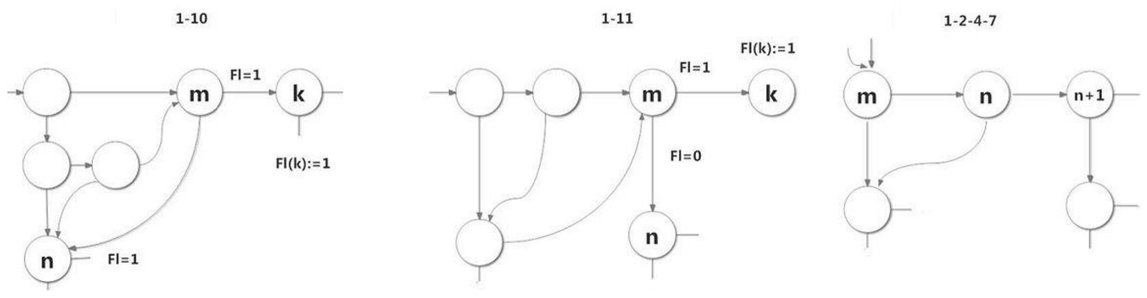


Figure 34: Walkthrough Example circuit (III)

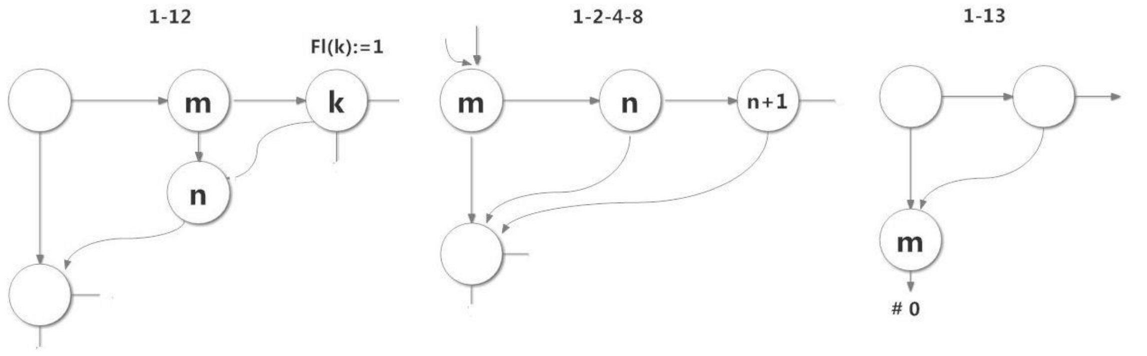


Figure 35: Walkthrough Example circuit (IV)

3.5. Conclusions

In this chapter we presented the solutions of the following tasks as the main content of the thesis: (1) the method of reasoning fault equivalence and dominance properties on SSBDDs, (2) the method of fault collapsing on SSBDDs, (3) developing the higher and lower bounds for the number of collapsed faults for the given circuit, and (4) the algorithm of fault collapsing, implemented as a software tool.

For the first task, we developed two theorems, which proved that analysis on the SSBDD model through the application of rule of dominance and equivalence could help us achieve more fault collapsing. This helped us in developing a fault-collapsing algorithm with linear complexity.

We described the method in the second task as topological analysis of the SSBDD model. This helps up achieve more fault collapsing on the circuit beyond the ones achieved through SSBDD synthesis. The impact of collapsing more fault in a circuit enhances the performance and cost of test generation and fault simulation. It also reduces the size of the fault dictionary needed during fault diagnosis.

The method described in the third task helps in estimating the achievable size of the collapsed fault set for the given SSBDD. Knowing the lower bounds for the given circuit can help in estimating the cost for further fault simulation and test generation. This can also help us in determining the expected gain of fault collapsing.

In task four, we described the fault collapsing algorithm and its working principle with walthrough examples. This algorithm is of linear complexity and was a result of the theorems we described in section 3.2.

Furthermore, a comprehensive discussion on the new structural fault collapsing method with linear algorithmic complexity was done. The method proposed is based on the two-phase topology analysis of the circuit description. First phase as discuss is SSBDD synthesis while the second phase involves topological analysis of the SSBDD.

We have shown that SSBDD has fewer nodes than the gate-level models, hence we can already have a reduced number of fault set. Hence improving fault collapsing. Secondly, the fault collapsing capability of the SSBDD proposed helps in eliminating the task of checking whether a fault belongs to the collapsed list or not.

Fault collapsing reduces in turn the model, the complexity of the model is reduced via the proposed model hence decreasing the requirement for the memory space for storing the model. The fault collapsing capability not only decrease the space needed, it also leads to increased speed of fault simulation and test generation.

4. Experimental Results

The fault collapsing experiments were carried out using ISCAS’85 [38], ISCAS’89 [39] and ITC’99 [40] benchmark circuits. The experimental platform was Intel(R) Core(TM) i7 VproTM L640 at 2.13 GHz, 4 GB RAM.

Figures 36, 37 and 38 show the percentage of collapsed faults of all fault in the benchmark circuits based on their sizes and complexity. Graphs 39, 40 and 41 shows the time for fault collapse in the circuits respectively.

The x-axis of the graphs represents the circuits with increasing increasing number of gates or faults. This then shows that the higher the number of faults in a circuit, the more faults we can collapse and the higher the time required for the collasing operation. The y-axis on the Figures 36-38 shows the number of faults collapsed in a circuit while it shows the time (in seconds) needed for the collapse operation on Figures 49-41.

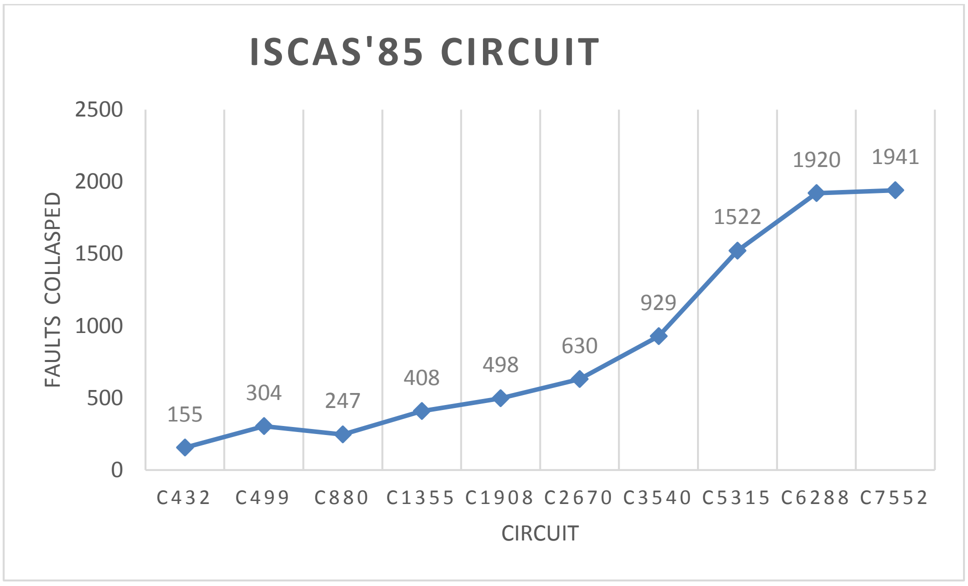


Figure 36: Collapsed faults for ISCAS'85 Circuit

In Figure 38, we could see a massive rise in the number of fault collapsed for circuit b18. This is because of the size of the circuit. The number of nodes in the circuit after SSBDD synthesis is 138989, this is about 82% more than that of circuit b20 (23688 nodes). This further proofs that the number of fault collapsed is proportional to the number of gates or faults in the circuit. The same explanation can be given for Figure 41 where more time was needed to collapse the fault in the circuit b18.

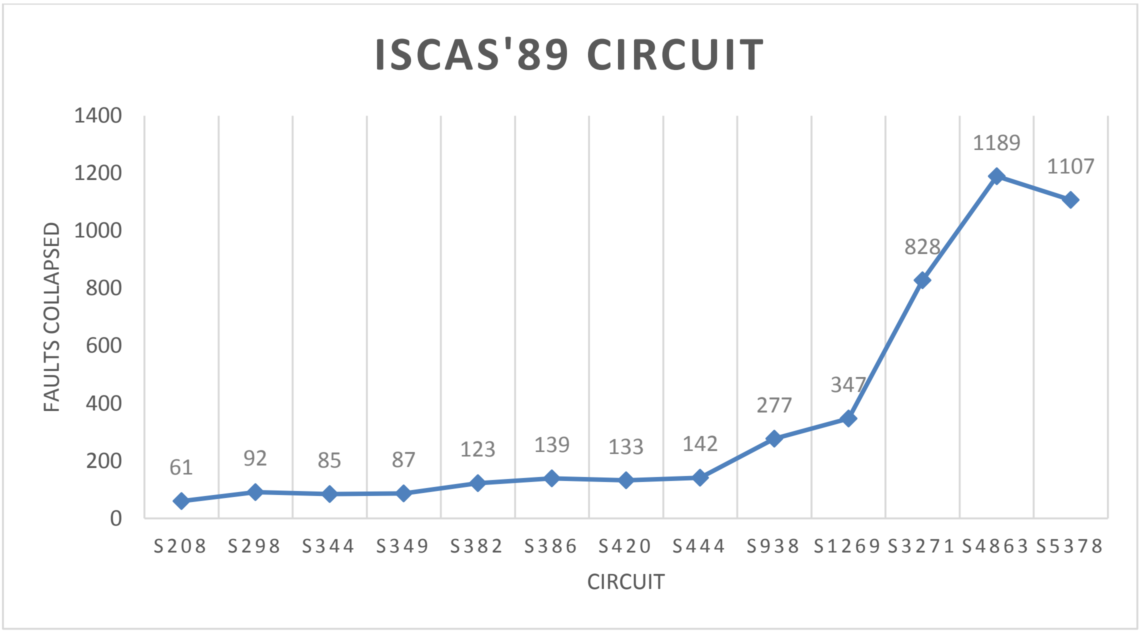


Figure 37: Collapsed faults for ISCAS'89 Circuit

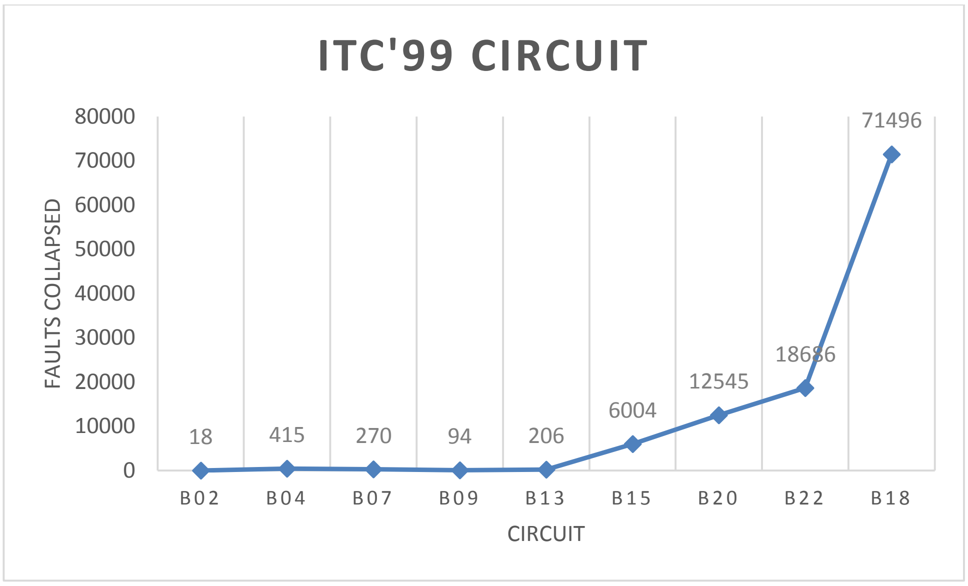


Figure 38: Collapsed faults for ITC'99 Circuit

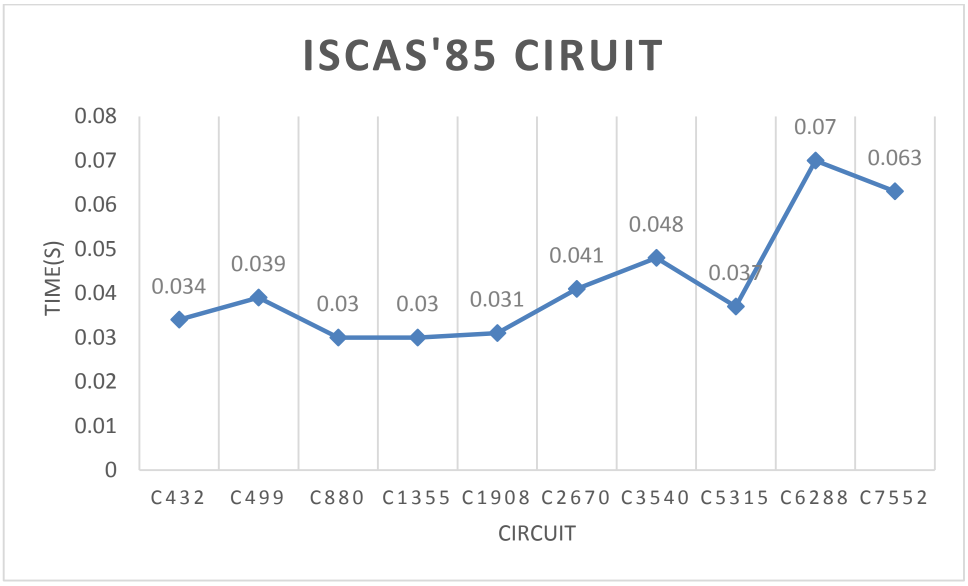


Figure 39: Fault collapse time for ISCAS'85 Circuit

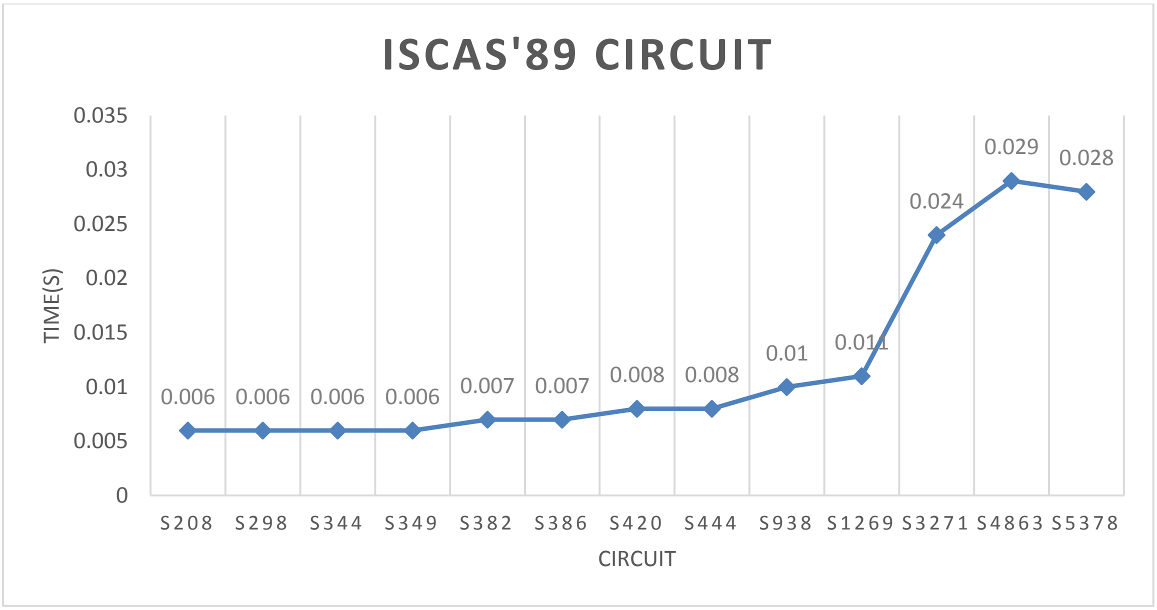


Figure 40: Fault collapse time for ISCAS'89 Circuit

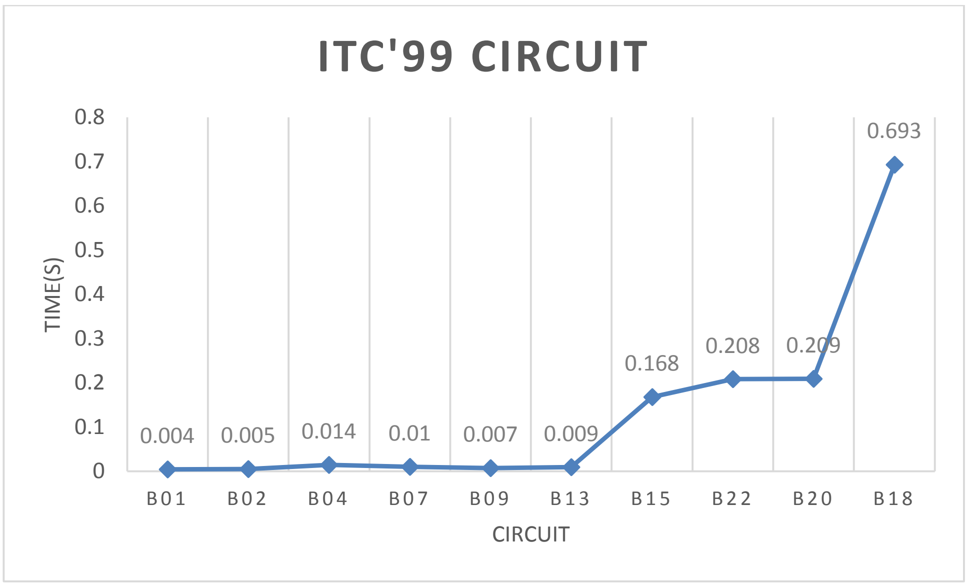


Figure 41: Fault collapse time for ITC'99 Circuit

In Table 6, the data are depicted for both, the 1st and 2nd phases of fault collapsing. In the table, the first column is the circuit under experiment, the second column with heading "# Faults" represents the total number of SAF faults in the gate-level circuit, the columns 3 and 4 show the representative fault set sizes after fault collapsing during SSBDD synthesis (I Phase), and after fault collapsing directly on SSBDDs (II Phase), respectively.

The 5th column shows the total number of the set of collapsed faults. The 6th, 7th and 8th columns show the time cost needed for fault collapsing during the 1st, and 2nd phases, and the total time cost, respectively.

Circuit	# Faults	# Repr. faults		Fault collapse	Time, ms		
		I Ph	II Ph		I Ph	II Ph	Σ
c1355	2710	1618	1210	1500	40	6	46
c1908	3816	1732	1243	2573	50	8	58
c2670	5340	2626	1996	3344	100	13	113
c3540	7080	3296	2367	4713	180	13	193
c5315	10630	5424	3902	6728	180	16	196
c6288	12576	7744	5824	6752	260	25	285
c7552	15104	7104	5163	9941	280	21	301

Table 6: Fault collapsing data of proposed method

The experimented result from the proposed method was compared with previous methods of fault collapsing. The Table 7 shows the result of this comparison. However, I could only carry out this comparison with former results on fault collapsing on ISCAS’85 circuits because results in literatures are only available for the ISCAS’85 circuits.

Circuit	Fault set size					CPU time, s	
	[36]	[41]	[42]	[7]	New	[42]	New
c1355	1234	1210	808	1100	1210	46	0.046
c1908	1568	1566	753	1286	1243	14	0.058
c2670	2324	2317	1853	2046	1996	110	0.113
c3540	2882	2786	2092	2584	2367	831	0.193
c5315	4530	4492	3443	4404	3902	72	0.196
c6288	5840	5824	5824	4832	5824	4	0.285
c7552	6163	6132	4707	5480	5163	232	0.301

Table 7: Comparison with other methods

In the table, column 2 represents the fault set size after applying structural fault collapsing i.e fault folding [36]. column 3 contains data that represents the fault set size of improved structural fault collapsing [41]. column 4 contains data that represents the result of functional fault collapsing [42]. Column 5 shows data of result of fault collapsing based on Shared SSBDDs(S3BDDs) which is an advanced and more compact version of SSBDDs. Column 6 shows the result of the proposed method of fault collapsing.

Although column 4 gives better fault collapsing it however uses a very time-consuming dominance graph analysis and transitive closure calculation and unfortunately, not a scalable method, and hence, not usable for complex circuits. Absolute speeds compasion of the methods is difficult because of missing data about computing platforms for [42]. However, the results clearly show the high dependence of the time cost on circuit size and structure for [42] whereas the proposed new method is well scalable and has a linear complexity. If we take for example, the difference in time cost needed for c3540 and c6288 for the method in [42] is 200 times whereas for the proposed method the time cost for both circuits is nearly the same.

5. Conclusions

This thesis aimed at discussing and proposing a new structural fault collapsing method with linear algorithmic complexity. Our focus was to reduce the search space for test generation and fault diagnosis in digital systems.

In chapter 2, we gave a background overview of topics relating to the research under review. In chapter 3, we made a comprehensive discussion on the proposed method. Results of experiment were done in chapter 4 and these we compared with previous methods of fault collapsing.

The method proposed is based on the two-phase topology analysis of the circuit description. First phase as discuss is SSBDD synthesis while the second phase involves topological analysis of the SSBDD.

Our focus was not only to capitalize on fault collapsing capability but to also decrease the space needed for test generation and fault diagnosis, improve or increase the speed of fault simulation and test generation.

The contribution of this thesis is in providing an excellent and scalable collapsing method which is very promising for large circuits. This method has been proven more efficient than previous structural fault collapsing methods.

The result of this research was presented in a paper titled double phase fault collapsing with linear complexity in digital circuit and was submitted to the 18th Euromicro Conference on Digital Systems Design in Funchal, Portugal in August 26-28, 2015.

For the future, I would look at optimizing the algorithm in order to achieve more faults collapse on digital circuits. Furthermore, I will also research on the applicability of the proposed method as it relates to hardware security.

References

[1] Abramovici, Miron; Breuer, Melvin A.; Friedman, Arthur D., “Digital Systems Testing and Testable Design”, John Wiley & Sons, Inc., Hoboken, New Jersey, 1990

[2] Laung-Terng Wang, Cheng-Wen Wu, Xiaoqing Wen., “VLSI Test Principles and Architectures Design for Testability,” Morgan Kaufmann Publishers, 2006

[3] G. Moore. Cramming More Components onto Integrated Circuits. – Reprint from IEEE proceedings on Electronics, Vol. 38, No. 8, 1965.

[4] R. W. Keyes, “The Impact of Moore's Law. – IEEE Solid-State Circuits, Issue”, Sept 2006.

[5] Mehran Nadjarbashi, Zainalabedin Navabi and Mohammad R. Movahedin, “Line Oriented Structural Equivalence Fault Collapsing”, Electrical and Computer Engineering Department Faculty of Engineering – Campus #2 – University of Tehran 14399, Tehran – IRAN

[6] Raimond Ubar, Lembit Jurimae, Elmet Orasson, Galina Josifovska, Stephen Adeboye Oyeniran, “Double Phase Fault Collapsing with Linear Complexity in Digital Circuits”, 2015

[7] R. Ubar, D. Mironov, J. Raiks, A. Jutman, “Structural Fault collapsing by Superposition of BDDs for Test Generation in Digital Circuit”, 2010

[8] A. Jutman, A. Peder, J. Raik, M. Tombak, “Structurally synthesized binary decision diagram”

[9] Daniel P. Siewiorek, Fellow, IEEE, and Larry Kwok-woon Lai, member, JEEE, “Testing of Digital Systems, proceedings of the IEEE”, vol. 69, no. 10, October 1981.

[10] Breuer, M. A.; Friedman A. D., “Diagnosis and Reliable Design of Digital Systems”, Computer Science Press, New York, 1976

[11] Micheal L. Bushnell, Vishwani D. Agrwal, “Essentials of Electronic Testing for Digital, Memory and Mixed-signal VLSI Circuits”

[12] V.K. Agarwal, A.F.S.Fung, “Multiple Fault Testing of Large Circuit by Single fault Test Set, IEEE Trans. On Computers”, vol C-30, no 11, pp855-856, Nov 1981

[13] Bob Strunz, Colin Flanagan, “Design for Testability in Digital Integrated circuits”, Tim Hall University of Limerick, Ireland

[14] M. RAY MERCER University of Texas, “Algorithms for Pattern Generation”, June 1998

[15] Niraj Jha, Sandeep Gupta, “Testing of Digital Systems”, Cambridge University Press, 2003

[16] Circuits S. Hemalatha, Mrs. K. Srividhya, “Automatic Test Pattern Generation for Digital” S. Hemalatha et al Int. Journal of Engineering Research and Applications ,ISSN : 2248-9622, Vol. 4, Issue 4 (Version 1), April 2014, pp.345-351

[17] P. Goel, “An implicit enumeration algorithm to generate test for combinational logic circuit” IEEE Trans, Comput.Vol C-30. Pp 215-222 March 1981

[18] Micheal H. Schulz, Erwin Trischer, Thomas M. Sarfert, “Socretes: A highly efficient automatic test pattern generation system”

[20] Baris Arslan, Alex Orailoglu, “Fault Dictionary Size Reduction through Test Response Superposition”, Computer Science and Engineering Department University of California, San Diego

[21] YU-RU HONG, Reducing Fault Dictionary Size for Million-Gate Large Circuits, “Purdue University and JUINN-DAR HUANG National Chiao Tung University”

[22] Irith Pomeranz, Sudhakar M. Reddy, On the Generation of Small Dictionaries for Fault Location Electrical and Computer Engineering Department University of Iowa

[23] Paul G. Ryan, W. Kent Fuchs, Irith Pomeranz “Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits”

[24] Vishwani D, Agrawal A, V. S. S. Prasad, Madhusudan V. Atre Rutgers University, Dept. of ECE Agere Systems Piscataway, NJ 08854, USA Bangalore 560066, India

[25] R. Ubar. Test Generation for Digital Circuits Using Alternative Graphs (in Russian), in Proc. Tallinn Technical University, 1976, No.409, Tallinn TU, Tallinn, Estonia, pp.75-81.

[26] Jaan Raik, Raimund Ubar, Sergei Devadze, Artur Jutman, Efficient Single-Pattern Fault Simulation on Structurally Synthesized BDDs, Tallinn University of Technology, Department of Computer Engineering, Raja 15, 12618 Tallinn, Estonia

[27] C.Y. Lee. Representation of Switching Circuits by BDDs”, in Bell System Techn. J., 1959, v.38, No7, pp.985-999.

[28] S.Akers. Binary Decision Diagrams,” IEEE Trans. on Comp., Vol.27, 1978, pp.509-516

[29] R.Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Trans on Comp, 1986, vol. C-35, 677-691

[30] M.Gorev, R.Ubar, S.Devadze. “Fault Simulation with Parallel Exact Critical Path Tracing in Multiple Core Environment”. Proc. of DATE, Grenoble, France, 2015, pp. 1-6.

[31] R.Ubar. “Test Synthesis with Alternative Graphs. IEEE Design & Test of Computers”. Spring 1996, pp. 48-59.

[32] R.Ubar, S.Devadze, J.Raik, A.Jutman. “Parallel X-Fault Simulation with Critical Path Tracing Technique”. Proc. of DATE, Dresden, Germany, 2010, pp. 1-6.

[33] A. Jutman, J. Raik, R. Ubar. “SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test”. – Proc. of 5th Int. Workshop on Boolean Problems, Germany, 2002, pp. 157-166.

[34] Jaan Raik, “Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams”. 2001

[35] Margit Aarna, Jaan Raik, Raimund Ubar, “Parallel Fault Simulation of Digital Circuits”

[36] K.To. “Fault Folding for Irredundant and Redundant Combinational Circuits”. IEEE Trans. on Computers, Vol. C-22, No. 11, pp.1008-1015, Nov. 1973.

[37] R. Ubar. “Overview about low and High-Level Decision Diagrams for Diagnostic Modelling of Digital Systems”. Facta Universitatis (Nis) Ser.: Elec. Energ. Vol24, no.3, Dec. 2011, 303-324

[38] F.Brglez, H.Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", Proc. of the International Test Conference, 1985, pp. 785-794.

[39] F.Brglez, D.Bryan, K.Kominski, "Combinational Profiles of Sequential Benchmark Circuits", Int. Symp. on Circuits and Systems, 1989, pp.1929-1934.

[40] F.Corno, M.S.Reorda, G.Squillero, "RT-level ITC'99 Benchmarks and First ATPG Results", In Proc. Of the IEEE Design & Test of Computers, Vol. 17, No. 3, 2000, pp.44-53

[41] A.V.S.S.Prasad, V.D.Agrawal, M.V.Atre. “A New Algorithm for Global Fault Collapsing into Equivalence and Dominance Sets”. Proc. of Int. Test Conference, pp.391-397, Oct.2002.

[42] R.Sethuram, M.L.Bushnell, V.D.Agrawal. “Fault Nodes in Implication Graph for Equivalence Dominance Collapsing, and Identifying Untestable and Independent Faults. Proc”. of VTS, pp.329-335, 2008.

Appendix 1 – Program Description and Manual

This application is a command application that runs on windows operating system. The application is saved as FaultCollapsing.exe. The following instruction helps successful execution of the application

1. Copy the application, FaultCollapsing.exe into a folder
2. Copy the agm file you want to collapse to the same folder
3. Open a command prompt on a window machine

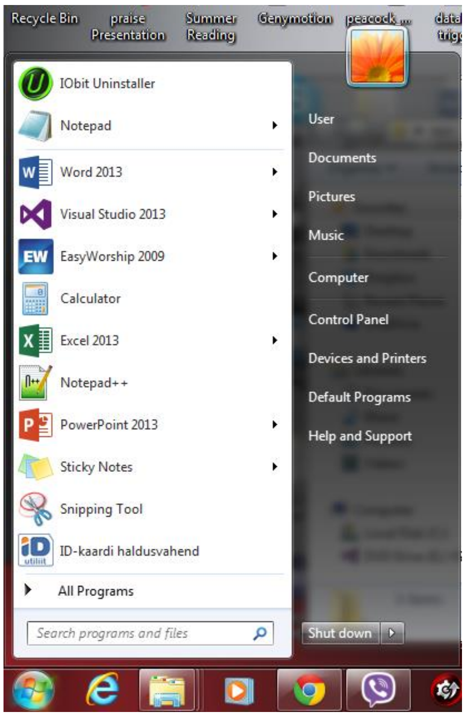
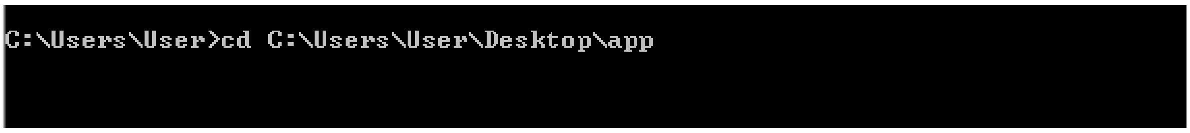


Figure 42: Launching command prompt

From the search program, type cmd and push the enter button on the keyboard. This will open up the command prompt

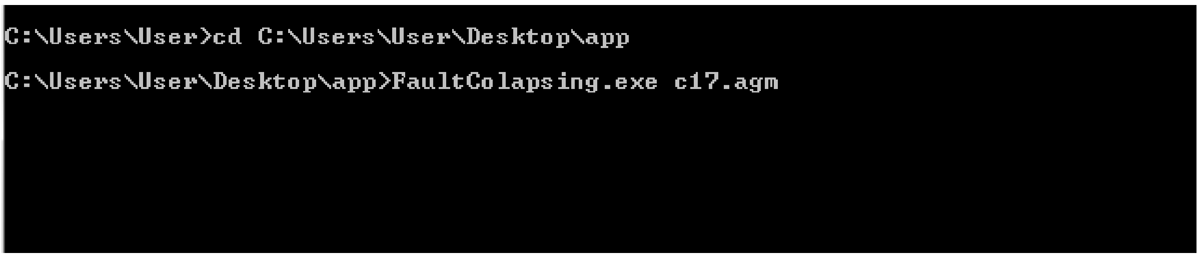
4. Change directory to the folder where the executable file is stored. E.g. cd C:\Users\User\Desktop\app



```
C:\Users\User>cd C:\Users\User\Desktop\app
```

Figure 43: Navigating to application

5. From the directory changed to on the command prompt, run the executable file and pass argument which is the agm file name which is in the same folder as the executable



```
C:\Users\User>cd C:\Users\User\Desktop\app
C:\Users\User\Desktop\app>FaultColapsing.exe c17.agm
```

Figure 44: Collapsing Fault with Application

6. The result of the collapse is displayed on the command prompt, but a comprehensive result of the algorithm is saved in the project directory. You can open with a standard text file reader. The file contains the nodes collapse and the stuck-at faults on these nodes(collapse attributes)

Note: The result of the algorithm is a text file with the same name as the agm file but with a c after it. Eg. For an agm file named **c17.agm**, the result of collapse will be **c17.agmc**

Appendix 2 – Source Code

```
// FaultColapsing.cpp : Defines the entry point for the console
application.

#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

//extern "C" {
#include "messages.h"
//}
#include "mudel.h"

clock_t startm, stopm;
#define START if ( (startm = clock()) == -1) {printf("Error calling
clock");exit(1);}
#define STOP if ( (stopm = clock()) == -1) {printf("Error calling
clock");exit(1);}
#define PRINTTIME printf( "%6.3f seconds used by the processor.",
((double)stopm-startm)/CLOCKS_PER_SEC);

#pragma warning(disable : C4018)
#pragma warning(disable : C4101)
#pragma warning(disable : 4996)

#define INV( d ) ( (d) == 0 ? 1 : 0 )
int getDirection(mtgt, mtgt1);
int getNDirection(ntgt, ntgt1);
FILE *ft;

int main(int argc, char* argv[])
{
	// collapse count and target (end) node index
	unsigned int i, j, tgt, mtgt, ntgt, mtgt1, ntgt1, nbtgt, nbtgt1;
	clock_t start, end;
	double cpu_time_used;
	//FILE *ft;

	unsigned int m, n; // number of node m and n where n =m+1
	unsigned int nb; // number of the neighbour node of n. nb = n+1
	unsigned int globalm, globaln; // global index of node m and n.
	where globaln = globalm+1
	unsigned int globalnb; // number of the next node globalnb =
	globaln+1
	unsigned int TM; // number of all nodes in all graphs
	unsigned int M; // number of all nodes
	unsigned int md; // neighbor of the node m in direction d
	unsigned int INm; // direction of input edges to node m
	unsigned int Cm;  //attribute of fault collapsing at node m
	unsigned int colapsecount, globalLenght;
	unsigned int mdirection, ndirection; // direction of nodes m and n
respectively

	start = clock();
```

```
START;
//Read AGM file and display attributes, otherwise exit
if (argc >1  && strlen(argv[1]) > 0) {

       read_ag(argv[1]);
       TM = NodCount; // number of all nodes
}
else{
       //printf("Please input the ssbdd file");
       Error("No model file", -1);
       end = clock();
       STOP;
       PRINTTIME;
       cpu_time_used = ((double)(end - start));
       printf(" %s: %g\n", "time spent", cpu_time_used);

       }
//writing result to file
if (argc > 1 && strlen(argv[1]) > 0) {
       char buffer[20];
       char *name = "c.txt";
       strcpy(buffer, argv[1]);
       strcat(buffer, name);
       ft = fopen(buffer, "wb");
}
fprintf(ft," MODE#   %s\n", (ModelType == STRUCTURAL_AGM) ?
"STRUCTURAL" : "FUNCTIONAL");
fprintf(ft, " %s: %s\n", "Circuit Name", argv[1]); // Name of
circuit
fprintf(ft, " %s: %d\n", "Number of all nodes", TM); // checking the
total number of nodes
fprintf(ft," %s: %d\n", "Number of graphs", GrpCount); // checking
the total number of graphs
fprintf(ft," %s: %d\n", "Number of variables", VarCount); //
checking the total number of graphs
fprintf(ft," %s: %d\n", "Number of inputs", InpCount); // checking
the total number of inputs
fprintf(ft," %s: %d\n", "Number of output", OutCount); // checking
the total number of output
fprintf(ft," %s: %d\n", "Number of constant", ConCount); // checking
the total number of constant

/* Analyze AGM for collapsible faults for each graph*/
//start = clock();

globalLenght = 0;
colapsecount = 0;

for (i = 0; i < GrpCount; i++) {
       m = n = 0;
       md = -1;
       M = GLEN(i)-1;

       // Array for storing flag values
       int *f = malloc(sizeof(int)*(M+1));
       memset(f, 0, sizeof(int)*(M+1));

       //fprintf(stderr, "\n=== GRP: %u\n", i);
       fprintf(ft, "\n === GRP: %u\n", i);
       globalLenght = globalLenght + GLEN(i) ;
```

```
                // check that the number of nodes in a graph is not less than
2
                if (GLEN(i) >= 2)
                {
                      // AND search, for each node in the graph
                      for (int k = 0; k < M ; k++)
                      {
                            if (m < M)
                              {
                                  n = m + 1;
                                  nb = n + 1;
                                  globalm = GBEG(i) + m;
                                  globaln = globalm + 1;
                                  globalnb = globaln + 1;

                                  mtgt = NDST(globalm, 0);//m target
                                  downwards
                                  ntgt = NDST(globaln, 0);// n target
                                  downwards

                                  mtgt1 = NDST(globalm, 1);//m target to the
                                  right
                                  ntgt1 = NDST(globaln, 1);//n target to the
                                  right

                                  if (nb <= M) // check that n is not
                                  terminal node otherwise n should be 0
                                  {
                                        nbtgt = NDST(globalnb, 0); //n
                                        neighbour target downwards
                                        nbtgt1 = NDST(globalnb, 1); // n
                                        neighbour target to the right
                                  }
                                  else{ nbtgt = 0; nbtgt1 = 0;
                                  //fprintf(ft," neighbour of n is terminal
                                  node\n");
                                  }// if n is terminal node then nb should be
                                  0

                                        // Determining joint neighbour
                                        if (mtgt == ntgt || mtgt1 == ntgt1)
                                        {
                                              //printf(" m and n have joint
                                              neighbours\n");
                                              // Checking the fan-in flag.
                                              if (f[n]== 1)
                                              {
                                                    //cm = !INm;
                                                if (NDST(globalm,
                                                INV(mdirection) != 0)) //if
                                                the neighbour of n in
                                                opposite direction is not
                                                terminal
                                                    {
                                                          f[NDST(globalm,
                                                    INV(mdirection))] = 1;
                                                    }
                                                    fprintf(ft," %s: %d\n",
                                                    "collapse found, SAF",
                                                    globalm);
                                                    colapsecount++;
```

```
                                     m = m + 1;
                                     continue;
                               }
                               else
                               {
                                      // n is not equal to m

                                     //direction of m to n
                                            mdirection =
                                           getDirection(mtgt,
                                           mtgt1);
                                            // direction of n
                                            to n+1
                                            ndirection =
                                           getNDirection(ntgt
                                           , ntgt1);

                                            //if direction
                                           doesn't change

if (mdirection == ndirection)
      {
      //check terminal nodes
      if (ntgt == 0 && ntgt1 == 0) // determine if terminal
      node is reached, no direction is checked. group ends
      {
         Cm = INV(mdirection);
        fprintf(ft," %s: %d\t %s: %d\n", "collapse found on",
        globalm, "SA",Cm);

 colapsecount++;
      m = m + 2;

 continue;
      }
      else{
      if (f[nb] == 1) // check that FI(n(d)) = 1

 Cm = INV(mdirection);
      fprintf(ft," %s: %d\t %s: %d\n", "collapse found on",
      globalm, "SA", Cm);

colapsecount++;

m = m + 2;
      }
      else
      {

// check joint neighbour between n and n(d)
      if (nb <= M) // to check that index of nb doesnt exceed
      limit

{

      if (ntgt == nbtgt || ntgt1 == nbtgt1)

      {

      // printf("  n and nb are neighbours");
```

```
                    Cm = INV(mdirection);  //c(m) = !d(m)
                          fprintf(ft," %s: %d\t %s: %d\n", "collapse found
                          on", globalm, "SA", Cm);

                    colapsecount++;

                    m = m + 1;

                    continue;

                    }

                     else

                     {

                     Cm = INV(mdirection);  //c(m) = !d(m)
                          fprintf(ft," %s: %d\t %s: %d\n", "collapse found
                          on", globalm, "SA", Cm);

                    colapsecount++;

                    m = m + 2;

                continue;

               }

             }
               }

         }
        }
        else  //check if direction change
        {
        Cm = INV(mdirection);

        fprintf(ft," %s: %d\t %s: %d\n", "collapse found on", globalm, "SA",
        Cm);
        if (NDST(globaln, INV(ndirection) != 0)) //if the neighbour of n in
        opposite direction is not terminal
        {

        f[NDST(globaln, INV(ndirection))] = 1;
        }

        colapsecount++;
        m = m + 2; // change to 2
        continue;
        }

    }

    }
    //if no joint neighbour but FI(m)=1
    else if ((mtgt != ntgt || mtgt1 != ntgt1) && (f[m]== 1))
    {
    // check if FI(n) =1
        if (f[n]==1)
        {
```

```
//C(m)=!IN(m)
//Cm = INV(mdirection);
//printf(" %s: %d\n", "collapse found, SAF", Cm);
if (NDST(globalm, INV(mdirection) != 0)) //if the neighbour of n in
opposite direction is not terminal
{
        f[NDST(globalm, INV(mdirection))] = 1;
}
        fprintf(ft," %s: %d\n", "collapse found, SAF", globalm);
        colapsecount++;
        m = m + 1;
        continue;
}
else
{
//m and n have no joint neighbours and no flag on m;
        Cm = INV(mdirection);
        fprintf(ft," %s: %d\t %s: %d\n", "collapse found on", globalm,
        "SA", Cm);
if (NDST(globalm, INV(mdirection) != 0)) //if the neighbour of n in
opposite direction is not terminal
{
        f[NDST(globalm, INV(mdirection))] = 1;
}
        colapsecount++;
        m = m + 1;
        continue;
}

}
//if no joint neighbour and FI(m)!=1
else if ((mtgt != ntgt || mtgt1 != ntgt1) && (f[m] == 0))
{
        Cm = INV(mdirection);
        fprintf(ft," %s: %d\t %s: %d\n", "collapse found on", globalm,
        "SA", Cm);

        if (NDST(globalm, INV(mdirection) != 0)) //if the neighbour of
        n in opposite direction is not terminal
        {
                f[NDST(globalm, INV(mdirection))] = 1;
        }
                //printf("  collapse found\n");
                colapsecount++;
                m = m + 1;
                continue;
                }


        }
        else if (m == M){
                globalm = GBEG(i) + m;

                //CM = !direction of input edge of node m(!IN);
                //Cm = INV(IN);
                //Cm = INV(mdirection);
                fprintf(ft," %s: %d\n", "collapse found, SAF",
                globalm);
                colapsecount++;
                break;
        }
        else{
               //printf("check this should never happens \n\n");
               break;
          }
         }

        }

        // skip graphs that are too short, cnt < 2 nodes
        else if (GLEN(i) < 2)  {
               //printf(" %s: %d\n", "graph lenght", GLEN(i));
               //printf(" %s: %d\n", "global lenght", globalLenght);
               //printf("this happens \n\n");
               continue;
        }

 }
        end = clock();
        STOP;
        cpu_time_used = ((double)(end - start) / CLOCKS_PER_SEC);
        printf(" %s: %g\n", "time spent", cpu_time_used);

        printf("\n");
        printf(" %s: %d\n", "Number of all nodes", TM); // checking
        the total number of nodes
        printf(" %s: %d\n", "Number of graphs", GrpCount); // checking
        the total number of graphs
        printf(" %s: %d\n", "Number of variables", VarCount); //
        checking the total number of graphs
        printf(" %s: %d\n", "Number of inputs", InpCount); // checking
        the total number of inputs
        printf(" %s: %d\n", "Number of output", OutCount); // checking
        the total number of output
        printf(" %s: %d\n", "Number of constant", ConCount); //
        checking the total number of constant
        printf("\n\nTotal collapse:  %u \n", colapsecount);

        fprintf(ft, "\n Total collapse:  %u \n", colapsecount);
        fprintf(ft, " %s: %g\n", "Time spent", cpu_time_used);
        PRINTTIME;

        fclose(ft);
        free_ag();
        return 0;
    }
        int getDirection(mtgt, mtgt1){
        unsigned int md; // neighbor of the node m in direction d
        md = 0;
        if (mtgt != 0 && mtgt1 != 0)
        {
               if (mtgt > mtgt1)
               {
                      md = 1;
               }
               else if (mtgt < mtgt1)
               {
                      md = 0;
               }
               //printf(" %s: %d\n", "direction of m to n", md);
        }
        else
        {
            if (mtgt == 0 && mtgt1 >= 1)
            {
                md = 1;
            }
            else if (mtgt >= 1 && mtgt1 == 0)
            {
                md = 0;
            }
            //printf(" %s: %d\n", "direction of m to n", md);
        }
            return md;
    }
        int getNDirection(ntgt, ntgt1){
            unsigned int nd; //neighbor of the node m in direction d
            nd = 0;
            if (ntgt != 0 &&  ntgt1 != 0)
            {
                if (ntgt > ntgt1)
                {
                    nd = 1;
                }
            else if (ntgt < ntgt1)
            {
                nd = 0;
            }
            //printf(" %s: %d\n", "direction of n to nd", nd);
        }

        else
        {
            if (ntgt == 0 && ntgt1 >= 1)
            {
                nd = 1;
            }
        else if (ntgt >= 1 && ntgt1 == 0)
        {
            nd = 0;
        }
        //printf(" %s: %d\n", "direction of m to n", nd);
        //return md;
    }
    return nd;
}
```