Test Pattern Generation and Fault Localization for Some Fault Models in Reversible Circuits

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy
in
Computer Science and Engineering

by
Mousum Handique

Under the guidance of
Prof. Jantindra Kumar Deka and Prof. Santosh Biswas

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
August, 2019

Declaration

I certify that:

a. The work contained in this thesis is original and has been done by me under the guidance of my supervisors.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in preparing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

f. Whenever I have quoted from the work of others, the source is always given.

Mousum Handique

Copyright

Attention is drawn to the fact that the copyright of this thesis rests with its author. This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

This thesis may be made available for consultation within the Indian Institute of Technology Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signature of Author...

Mousum Handique

Certificate

This is to certify that this thesis entitled, **"Test Pattern Generation and Fault Localization for Some Fault Models in Reversible Circuits"**, being submitted by **Mousum Handique**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under our supervision and guidance. The thesis, in our opinion, is worthy of consideration for the award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of our knowledge, it has not been submitted elsewhere for the award of the degree.

....

Jantindra Kumar Deka
Department of Computer Science and Engineering
IIT Guwahati

....

Santosh Biswas
Department of Computer Science and Engineering
IIT Guwahati

Dedicated to

My inspirational 'Maa'Aruna Handique

Although, I cannot hear her voice or see her smile no more. She walks beside me still just as did before

Acknowledgments

The work presented in this thesis would not have been possible without my close association with many people. It gives me immense pleasure to write down the acknowledgment for my Ph.D. thesis.

First and foremost, I would like to extend my sincere gratitude and respect to my thesis supervisor Prof. Jantindra Kumar Deka for his dedicated help, advice, inspiration, encouragement, and continuous support throughout my Ph.D. His regular critical reviews, constructive criticism, and unfailing co-operation greatly benefited me during the entire period of my research and writing of this thesis. Working under him has been really a great experience which I believed will help me throughout my entire life for pursuing not only better work but being a better human being as well. I am really glad to be associated with a person like Prof. Jantindra Kumar Deka in my life.

My special words of thanks should also go to my other thesis supervisor Prof. Santosh Biswas from the core of my heart. His constant guidance, cooperation, and support have always kept me going ahead. The kind of freedom he gave me during the period of my research is unbelievable. I owe a lot of gratitude to him for his timely and invaluable suggestions which have aided in making this work better and I feel privileged to be associated with a person like him.

I am grateful to Dr. Kamalika Datta from National Institute of Technology (NIT), Meghalaya, India and Prof. Indranil Sengupta from Indian Institute of Technology (IIT), Kharagpur for their valuable suggestion and cooperation during my research work.

My sincerest thank goes to the rest of my thesis committee members: Prof. Diganta Goswami, Prof. Arnab Sarkar, and Prof. Kanduru V. Krishna for their insightful comments and encouragement.

I am also thankful to all faculty members of the Department of Computer Science and Engineering, IIT Guwahati for their valuable suggestions. I must convey my heartfelt thanks to the staff members of the Department for their kind cooperation.

I would like to express my gratitude to the Director, the Deans and other managements of IIT Guwahati.

I would like to thanks the competent authority of Assam University, Silchar for giving me the leave for doing my Ph.D. at IIT Guwahati.

My heartfelt regard goes to my Deuta Purna Kanta Handique for his moral support and blessings.

I owe my deepest gratitude towards my wife Smriti for her external support and understanding of my work. Her patience and scarifies will remain my inspiration throughout my life. Her constant encouragement and faith have enabled me to overpower tough situations. I am thankful to my sons, Neelabh and Moumon for giving me happiness during my studies.

A big thank goes to Narayan Kalita, Sr. Superintendent, Hostel Affairs Bord (HAB), IITG for providing the Hostel accommodation whenever I needed during my research period. I am also grateful to all my friends and juniors (especially Mr. Awnish Kumar) for their kind help and support.

Date:

(Mousum Handique)
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Guwahati, India

Abstract

Due to the advancements of sub-micron/nanotechnology in semiconductors, computing platforms involve complex micro-architectural designs with multi-millions gates per chip with smaller sizes to meet the computation and performance demands. The complex design and smaller size are always associated with side-effects such as high energy dissipation, an increase in the probability of faults, etc. In conventional digital logic gates, the number of inputs and outputs lines can be different, which typically results in loss of information and heat dissipation. In 1973, Charles H. Bennett postulated that the information is lossless if the computation is reversible. Reversible functions are used in reversible computing. The applicability of the reversible computation can be used in the various types of reversible logic gates such as quantum computing, Quantum Cellular Automata (QCA), trapped-ion technology, optical computing, etc. Therefore, researchers have explored the reversible logic as a circuit design alternative, which is gaining popularity in recent times.

In the circuit design domain, the presence of manufacturing faults is a major concern that may lead to physical failure of the system. Even a single fault can cause a large deviation in the expected performance of a system. Fault detection and fault localization are two important phases in the field of circuit testing. With the increasing importance of the use of reversible logic circuits, testing of the reversible circuit is necessary to ensure their reliability. In this thesis, we consider some of the fault models to generate the test patterns to detect these faults in reversible circuits.

A reversible circuit realizes a reversible function and it is implemented by a linear cascade of reversible gates, where fan-out and feedback connections are not allowed. Due to the nature of the cascade connection, the circuit

consists of several levels from input to output and fault may occur at any level. In the first phase of our work, we consider the bridging faults at the input level. We identify a test set to detect all possible input bridging faults in a reversible circuit and establish that the test set is minimal and complete. It is also indicated that by adding one particular test vector to this test set, input stuck-at fault can also be detected.

In the next work, we consider intra-level bridging faults, i.e., single and multiple bridging faults that may occur at any level of the reversible circuits. The concept of path-level expression is introduced in this work to generate the test set to detect all possible intra-level bridging faults. It is also established that the generated test set is complete and minimal.

Stuck-at and bridging fault models are also used in conventional digital circuits to detect faults. There are some special fault models which are applicable for reversible circuits only. One such fault model is the missing gate fault model and in the third work we consider the missing gate fault model. We propose a method to generate the test set to detect single missing gate faults in a reversible circuit. Next, we enhance the test set to detect multiple missing gate faults. The generated test set is not a minimal one, so we use Integer Linear Programming (ILP) techniques to find the minimal test set. It is also established that the generated test set is complete. In this work, a correlation of other fault models is shown with missing gate fault model.

Since the missing gate fault model is specific to reversible circuits, so in our last work, we propose a method to locate the missing gate faults in reversible circuits. The test set generated to detect the missing gate faults is used to construct a fault localization tree and with the help of fault location tree, a method is proposed to locate the fault positions of missing gate faults.

We also perform the complexity analysis for all the proposed methods in this thesis and the computational complexities of the proposed methods are reasonable. The experiments are performed for all the proposed methods with benchmark circuits and comparisons are shown with some of the published works. It is observed that our proposed methods perform better than the existing schemes and fault coverage of our proposed methods is nearly 100%.

Contents

1 Introduction **1**
1.1 Preliminaries 5
1.1.1 Reversible Logic Function 5
1.1.2 Reversible Logic Gates and Libraries 5
1.1.3 Reversible Circuits 9
1.1.4 Fault Models in Reversible Circuit 10
1.1.4.1 Stuck-at Fault Model (SAF) 11
1.1.4.2 Bridging Fault Model (BF) 12
1.1.4.3 Missing-gate Fault Model 13
1.1.4.4 Crosspoint Fault Model 16
1.2 Motivation and Objectives of the Work 17
1.3 Related Work 18
1.4 Summary of Contributions 21
1.4.1 Test Generation for Input Stuck-at and Bridging Faults in Reversible Circuits 21
1.4.2 Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions 21
1.4.3 Test Generation for Multiple Missing-Gate Faults in Reversible Circuits 22
1.4.4 Fault Localization for Missing Gate Faults in Reversible Circuits . 23
1.5 Organization of the Thesis 24
1.6 Conclusion 25

2 Test Generation for Input Stuck-at and Bridging Faults in Reversible Circuits 27

2.1 Introduction . . . 27
2.2 Related Work . . . 28
2.3 Proposed Method . . . 30
2.3.1 Test Generation for Single Input Bridging Fault . . . 31
2.3.2 Test Set Generation for Single Input Stuck-at Fault . . . 35
2.4 Experimental results and Discussions . . . 38
2.5 Conclusion . . . 42

3 Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions 45

3.1 Introduction . . . 45
3.2 Related Work . . . 46
3.3 Proposed Method . . . 47
3.3.1 Local Test Pattern Generation Method . . . 51
3.3.2 Path Generation Method . . . 52
3.3.3 Complete Test Set Generation Method . . . 55
3.3.4 Complexity of the Proposed Method . . . 61
3.4 Experimental Results and Discussions . . . 62
3.5 Conclusion . . . 69

4 Test Generation for Multiple Missing-Gate Faults in Reversible Circuits 71

4.1 Introduction . . . 71
4.2 Related Work . . . 72
4.3 Proposed Method . . . 74
4.3.1 Detection Technique for Single Missing Gate Fault . . . 75
4.3.2 Complete Test Set Generation for Single Missing Gate Fault . . . 76
4.3.3 Complexity Analysis of complete test set generation . . . 78
4.3.4 Complete Test Set Generation for Multiple Missing Gate Fault . . . 78
4.3.5 Complexity Analysis of complete test set TS generation . . . 81
4.3.6 Determination of Minimal Complete Test Set . . . 81

4.3.6.1 Complexity Analysis of ILP formulation 83
4.3.7 Fault Coverage Evaluation with other Faults Models 84
4.4 Experimental Results and Discussions 86
4.5 Conclusions 92

5 Fault Localization for Missing Gate Faults in Reversible Circuits 95
5.1 Introduction 95
5.2 Related Work 96
5.3 Proposed Technique of Fault localization 100
5.3.1 Module 1: Complete Test Set Generation 100
5.3.2 Module 2: Test Response Generation for Fault Localization 101
5.3.2.1 Enumeration of SMGFs and MMGFs 103
5.3.2.2 Construction of Fault Analysis Table 104
5.3.2.3 Evaluating Equivalent Faults 105
5.3.3 Module 3: Construction of Fault Localization Tree 105
5.3.3.1 Reduction Process for Fault Localization Tree 110
5.3.4 Module 4: Traversal Process for Fault Localization 111
5.4 Experimental Results and Discussions 115
5.5 Conclusions 120

6 Conclusion and Future Work 121
6.1 Summarization 121
6.2 Future Work 124

References 127

Appendix A: Summary of Publications 135

List of Figures

1.1 Reversible Gates and their Gate Operation. 7
1.2 Reversible Circuit Structure. 9
1.3 Illustration of NCT and GT based Reversible Circuit 10
1.4 k-CNOT Reversible Circuit comprising of **two** 1-CNOT, **one** 2-CNOT, and **one** 3-CNOT gate 10
1.5 Illustration of Reversible $ham3_tc$ benchmark circuit: (a) Fault-free circuit, (b) SSF, (c) MSF, (d) SBF, and (e) MBF 12
1.6 Illustration of Reversible Circuit nth_Prime3_inc for various fault conditions under missing-gate fault model 14
1.7 Illustration of Crosspoint faults in Reversible Circuit 16

2.1 Test set for detecting single input bridging faults 32
2.2 Demonstration of single input bridging faults for N=8 and N=9 32
2.3 3_17tc_tfc benchmark circuit 34
2.4 $4b15g_2$ benchmark circuit 34
2.5 Comparison of the proposed work with the work of [1] 41
2.6 Comparison of the proposed work with the work of [2] 42

3.1 Demonstration of Reversible Circuit $n^{th}_Prime3_inc$ for various bridging fault conditions 49
3.2 Test set generation for detecting input bridging faults 52
3.3 Illustration of path generation for the circuit $n^{th}_Prime3_inc$ using backtracking 53

3.4 Demonstration of Complete Test Set generation for the circuit $n^{th}_Prime3_inc$ 59
3.5 Plot of CPU time vs. Number of input lines 64
3.6 Plot of CPU time vs. Number of gates 65
3.7 Plot of CPU time vs. Number of faults 66

4.1 Demonstration of Algorithm 2 for the circuit $rd32_\ v0_66$ 77
4.2 $Toffoli_double_4$ benchmark circuit 80
4.3 $ham3tc$ benchmark circuit 83

5.1 Block diagram of Fault Localization Method 99
5.2 $Miller_11$ benchmark circuit 102
5.3 Enumeration Process for the circuit $Miller_11$: (a) one gate (b) consecutive two gates (c) consecutive three gates (d) consecutive four gates, and (e) consecutive five gates 104
5.4 Fault Localization Tree for the circuit $Miller_11$ 108
5.5 Reduction Process of a given tree 109
5.6 Unit vertices of a fault localization tree for the circuit $Miller_11$ during the reduction process 111
5.7 Fault localization tree after the reduction Process for the circuit $Miller_11$112
5.8 Reduced fault location tree for the circuit $Miller_11$ with test vectors in binary form 113
5.9 Traversal process of fault $f_{g_2}^{(l_1,l_2)}$ for the circuit $Miller_11$ 115
5.10 Traversal process of fault $f_{g_2}^{(l_1,l_2)}$ for the circuit $Miller_11$ 116
5.11 Performance analysis of Number of faults vs. CPU time 119
5.12 Performance analysis of Number of test patterns vs. CPU time 119

List of Tables

1.1 Reversible function f 6
1.2 Permutation matrix of reversible function f. 6
1.3 Faulty and fault-free outputs for $ham3_tc$ benchmark circuit of Figure 1.5 13

2.1 Bridging fault coverage for the circuit 3_17tc_tfc 34
2.2 Bridging fault coverage for the circuit $4b15g_2$ 35
2.3 Stuck-at fault coverage for the circuit 3_17tc_tfc 37
2.4 Stuck-at fault coverage for the circuit $4b15g_2$ 37
2.5 Detection of input bridging and stuck-at faults (S = Stuck-at faults and B = Bridging faults) with CPU time (sec) for benchmark circuits 39
2.6 Comparison of the test vectors with [1] 40
2.7 Comparison of the test vectors with [2] 40

3.1 Truth Table for the $n^{th}_Prime3_inc$ circuit of Figure 3.1 with fault-free and faulty outputs 50
3.2 Meta Characters and Interpretation 56
3.3 Fault coverage table for the circuit $n^{th}_Prime3_inc$ with 2 test vectors . . 60
3.4 Complete test set for detection of bridging faults with simulation CPU time (sec) for the benchmark circuits 63
3.5 Comparison of the complete test set with [3] 67
3.6 Comparison of the complete test set with [4] 67
3.7 Comparison of the complete test set with [2] 68

4.1 Row and Column fault coverage table of the $ham3tc$ benchmark circuit . 84

4.2 Complete and Minimal test set for detection of SMGFs and MMGFS with CPU time (sec) for the benchmark Circuits 87
4.3 Comparison of the complete test set with [5], [6], [7], [8], [9] 89
4.4 Fault coverage range with SAF, PMGF and Crosspoint Fault Models by the proposed complete test set TS 91

5.1 Truth Table of the circuit $Miller_11$ 102
5.2 Input and corresponding output responses for the fault-free circuit $Miller_11$ 103
5.3 Fault analysis table for the circuit $Miller_11$ 106
5.4 Evaluation of equivalent faults for the circuit $Miller_11$ 107
5.5 Experimental results for the fault localization of SMGFs and MMGFs with CPU time (sec) for the benchmark circuits 117

List of Algorithms

1 Path generation algorithm 55

2 Complete test set TS_{SMGF} generation for detecting the SMGFs. 76

3 Complete test set generation for MMGFs 80

4 Fault localization algorithm. 114

Abbreviations

ATPG	**Automatic** Test Pattern Generation
BDD	**Binary** Decision Diagram
CNOT	**Controlled** NOT Gate
CFLTS	**Complete** Fault Location Test Set
DFT	**Design** for Testability
DTPG	**Diagnostic** Test Pattern Generation
EDA	**Electronic** Design Automation
ETG	**Extended** Toffoli gates
FEF	**Functionally** Equivalent Fault
GT	**Generalized** (n-bit) Toffoli gate Library
ILP	**Integer** Linear Programming
k-**CNOT**	k-**Controlled** NOT Gate; k is the integer
MCT	**Multiple**-control Toffoli Gate
MSF	**Multiple** Stuck-at fault
MBF	**Multiple** Bridging fault
MIBF	**Multiple** Input Bridging faults
MIRBF	**Multiple** Intra-level Bridging faults
MGF	**Missing** Gate fault
MMGF	**Multiple** Missing Gate fault
NCT	**NOT**, **CNOT**, and **Toffoli** gate Library
NCTF	**NOT**, **CNOT**, **Toffoli**, and **FREDKIN** gate Library
OTS	**Optimal** Test Set
PMGF	**Partial** Missing Gate fault
RGF	**Reduced** Gate fault
SAF	**Stuck**-at fault
SSF	**Single** Stuck-at fault
SBF	**Single** Bridging fault
SIBF	**Single** Input Bridging faults
SIRBF	**Single** Intra-level Bridging faults
SMGF	**Single** Missing Gate fault
UTS	**Universal** Test Set

Chapter 1

Introduction

Advancements of sub-micron/nanotechnology in semiconductors have led to massive growth in the electronics circuit design domain. New breakthroughs in semiconductor design and manufacturing enable to fabricate a complex circuit with more components in a small silicon area. Moore's law [10], which states that the density of the integrated circuits roughly doubles every two years, has withstood the test of time for the last three decades. Shrinking transistor dimensions and lower power voltages result in higher sensitivity and leading to significantly higher error rate. Smaller interconnect features and higher operating frequencies increase the number of errors in a circuit. The higher levels of integration and newly developed fabrication processes are capable of reducing the size of the chips and increasing the speed and package density. Also, the problem of power dissipation is increasing rapidly. Though several methods have been evolved for low power design, still power dissipation of semiconductor electronic devices is a major issue. Due to these difficulties that appear in the semiconductor technologies, researchers are exploring alternative ways to implement the computational functions in computing devices.

In conventional digital logic gates, number of inputs and outputs lines can be different, which typically results in loss of information or change of bit information. The amount of energy required to change of one bit of information, known as Landauer limit [11], is $KTln2$ Joules, where K is the Boltzmann constant (approximately $1.38\times(10^{-23})$J/K) and T is the temperature of the system in Kelvins. In 1973, Charles H.

Bennett [12] postulated that the information is lossless if the computation is reversible. Reversible functions are used in reversible computing. A function is called a reversible function if the number of outputs is equal to the number of inputs and a particular input always produces a unique output, i.e., the reversible functions are bijective in nature. This property allows us to deduce the inputs from the outputs [13]. In other words, reversible function involves operations that can be easily and exactly reversed or undone [14, 15]. Furthermore, the applicability of the reversible computation can be used in the various types of reversible logic gates such as quantum computing [16], Quantum Cellular Automata [17], trapped-ion technology [8], optical computing [18], etc. Therefore, researchers explore the reversible logic as a circuit design alternative, which is gaining popularity in recent times.

The reversible logic functions are implemented by the reversible logic operations, which are called reversible gates. Therefore, a set of reversible gates is required to implement the reversible circuits. Some of the basic reversible gates are NOT gate, CNOT or FEYNMAN gate [19], TOFFOLI gate [20], multiple-control Toffoli gate (MCT) [20], also known as k-CNOT gates, FREDKIN gate [21], PERES gate [22] and SWAP gate [23]. The collection of different reversible gates forms the gate libraries [24, 25], such as the NCT library (NOT, CNOT and TOFFOLI gate), GT library (n-bit TOFFOLI gate), NCTF library (NOT,CNOT, TOFFOLI, and FREDKIN gate), etc. To maintain the reversibility, the reversible circuit should satisfy certain conditions: n-input and n-output bijective function, no fan-out and no feedback connections [16]. These conditions leave the reversible circuit as a linear cascade network structure of reversible gates [26], i.e., if the ith gate in a reversible circuit is gate G_i, then the gate G_i is active if and only if the gate $G_{(i-1)}$ has produced an output and the output of $G_{(i-1)}$ is considered as an input of gate G_i. The reversible circuits can be implemented with quantum gates [27–30].

In the circuit design domain, the presence of manufacturing faults is a major concern that may lead to physical failure of the system. Even a single fault can cause a large deviation in the expected performance of a system. Fault detection and fault localization are two important phases in the field of circuit testing. The first phase involves detecting

the presence of faults in the circuit and later phase involves finding the exact location of these faults. With the increasing importance of the use of reversible logic circuits, testing of the reversible circuit is necessary for the proper functioning of reversible logic circuits. The different performance parameters that have been considered for testing of reversible circuits are Test size, Quantum cost, Execution Time, Gate count, Number of inputs and outputs, Garbage outputs, Fault coverage, etc. For evaluating the physical defects in reversible circuits, we also need some abstract representations of these physical defects. These abstract representations have been formulated based on the mathematical model called a fault model. More precisely, a fault model is generally used to abstract the effects of physical failures and also helps to simplify the complexity of the testing mechanism during circuit testing [31]. Hence, various fault models for reversible logic circuits have been proposed, where some of them are common to the conventional logic circuits [32]. In the stuck-at fault model, a stuck-at fault causes some of the inputs or output lines (signals) of a gate to be fixed at either 0 (stuck-at-0) or 1 (stuck-at-1) [31,33]. In the bridging fault model, a bridging fault occurs when two or more signal lines in the circuit are unintentionally or accidentally shorted together to create a wired logic that gives a faulty output of the circuit [34]. Bridging faults can occur at input, output and intermediate levels of the circuits, which are commonly referred to as intra-level bridging faults. Apart from these classical fault models, several other fault models in reversible circuits are introduced in the literature [7,8,35]. In the missing-gate fault model, a single missing-gate fault (SMGF) [7] occurs when any one of the reversible gates has completely disappeared from the circuit. Whereas the multiple missing-gate faults (MMGF) causes the complete removal of two or more consecutive reversible gates. A repeated-gate fault (RGF) under the missing-gate fault model is due to an unwanted replacement of a reversible gate by multiple instantiations of the same gate. The partial missing-gate fault (PMGF) is also considered under the missing-gate fault model in reversible circuits. A PMGF occurs in the circuit when one or more control connections are missing. In the crosspoint fault model, two types of faults exist in reversible circuits [35]. If the one or more control point is missing, then it is called disappearance crosspoint fault, and

where one or more additional control point(s) is erroneously added, then it is referred to as appearance crosspoint faults.

The test generation process for detecting the faults in reversible circuits is relatively simpler than conventional logic circuits because of the property of reversibility ensures high controllability and observability [33]. An input test vector is capable of detecting the fault if the input test vector produces different outputs for the faulty and fault-free circuits. In general, a test set is complete if all the possible faults are covered in a given reversible circuit. Such a test set is called a minimal complete test set if it contains the minimum number of test vectors [33]. Various testing approaches for reversible logic circuits have been proposed in the literature that address the generation of the complete test set based on some classical fault models [1, 1, 3, 4, 6, 33, 36–39] which are used in the conventional circuits also and some fault models [5–9, 40–43] which are specifically applicable for the reversible circuits.

In the present thesis, we focus on efficient test set generation for single bridging faults at first. We formally prove that the generated test set is minimal. Next, a minimal test set is derived to compute the fault detection for single stuck-at faults in the reversible circuit. This work is extended for obtaining the minimal complete test set for single and multiple intra-level bridging faults. Moreover, we propose a method for generating the test set for the SMGF based on the concept of reversibility (controllability and observability) such that the searching mechanism and computational cost are low. Later on, using the complete test set for SMGF and dependency of the gates information, we extract the complete test set for MMGF in reversible circuit with low computation cost and 100% fault coverage. Finally, we propose a fault localization technique for evaluating the exact faulty location for SMGF and MMGF in a given reversible circuit.

This chapter is organized as: Section 1.1 provides the basic introduction on reversible logic function, reversible gates and libraries, reversible circuit structure and fault models. Section 1.2 discusses the main motivation and objectives of our thesis work. Some of the previous works which are relevant to our thesis have been discussed in Section 1.3. Section 1.4 explains the detail contribution of the present thesis work. The organization

track of the thesis work is explained in Section 1.5. Finally, concluding remarks are presented in Section 1.6.

1.1 Preliminaries

1.1.1 Reversible Logic Function

A logic function $f(x_1, x_2, \ldots, x_n)$ with n Boolean variables is called a reversible logic function if it realizes a bijective function. The bijective function $f : \mathbb{B}^n \Rightarrow \mathbb{B}^m$ allows a permutation on the set of input vectors to produce an output vector such that each possible input vector can uniquely determine an output vector and vice versa, where the number of input vectors should be equal to the number of output vectors (i.e., $n = m$). To briefly illustrate a reversible function, we consider the reversible function with three variables as depicted in Table 1.1. Here, the function $f(2, 3, 0, 1, 5, 6, 4, 7)$ is permuted over $(0, 1, 2, 3, 4, 5, 6, 7)$ such that there exists a one-to-one mapping from input vectors to output vectors i.e., $f(0) = 2$, $f(1) = 3$, $f(2) = 0$, $f(3) = 1$, $f(4) = 5$, $f(5) = 6$, $f(6) = 4$, and $f(7) = 7$. The function f satisfied the surjective (onto) and injective (one-to-one) conditions. As a result, the function f is considered as a bijective function. Moreover, the bijective function can also be represented by a permutation matrix. The permutation matrix of this function f is shown in Table 1.2. The entries values of the permutation matrix are denoted as 1-entry and 0-entry, where 1-entry is represented as an input pattern (row) is mapped to an output pattern (column), and if there is no mapping between the input and output pattern, then it is denoted as a 0-entry. Table 1.2 has shown that each row and column have only one 1-entry in the permutation matrix of function f. Therefore, the output pattern is uniquely determined by the input pattern and also from the output pattern; the input pattern can be retrievable. Hence, the Boolean function f is called a reversible function.

1.1.2 Reversible Logic Gates and Libraries

The reversible logic functions are implemented by the reversible logic operations, which are called reversible gates. The formulation of a reversible gate consists of k-input and k-

Table 1.1: *Reversible function f*

Function f	Input Vectors			Output Vectors			Permutation of f
	x	y	z	x′	y′	z′	
0	0	0	0	0	1	0	2
1	0	0	1	0	1	1	3
2	0	1	0	0	0	0	0
3	0	1	1	0	0	1	1
4	1	0	0	1	0	1	5
5	1	0	1	1	1	0	6
6	1	1	0	1	0	0	4
7	1	1	1	1	1	1	7

Table 1.2: *Permutation matrix of reversible function f.*

Inputs	Outputs							
	000	001	010	011	100	101	110	111
000	0	0	1	0	0	0	0	0
001	0	0	0	1	0	0	0	0
010	1	0	0	0	0	0	0	0
011	0	1	0	0	0	0	0	0
100	0	0	0	0	0	1	0	0
101	0	0	0	0	0	0	1	0
110	0	0	0	0	1	0	0	0
111	0	0	0	0	0	0	0	1

output wires; it is called as a reversible $k \times k$ gate [33]. The reversible gates maintain the bijective property and have an equal number of inputs and outputs. For the construction of the reversible circuit, several gates have been proposed over the past decades, some of the classical reversible logic gates and libraries are discussed as below.

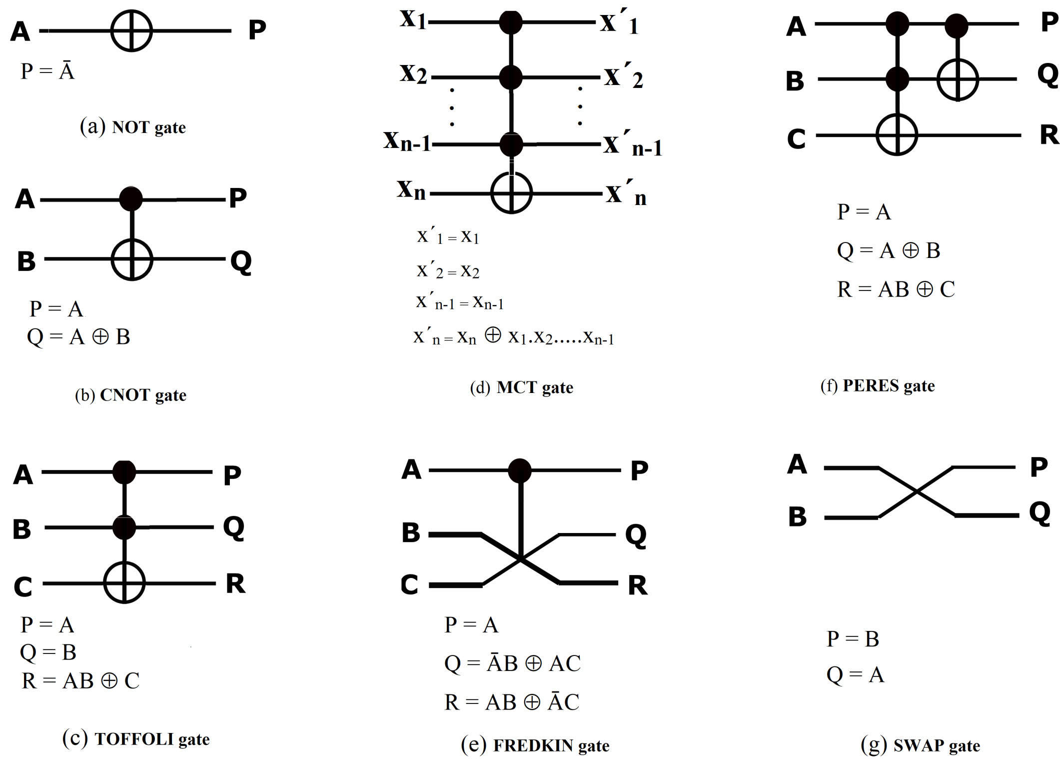


Figure 1.1: *Reversible Gates and their Gate Operation.*

i) **NOT** gate consists of 1-input and 1-output, i.e., 1×1 gate. The input A determines the output $\overline{A}$. In other words, the NOT gate maps $A \rightarrow A \oplus 1$. The symbolic representation of a reversible NOT gate is depicted in Figure 1.1 (a).

ii) **Controlled-NOT** (CNOT or FEYNMAN) [19] is a 2×2 gate. The formulation of the CNOT gate is the combination of one positive control line (•) and target line (⊕). Here, the output of the control line remains the same as the input control line and the value of the target line output is inverted if and only if the control

line input is activated as logic value 1. The operation of the CNOT gate realizes the bijective function which is represented as $(A, B) \rightarrow (A, A \oplus B)$. The symbolic representation and gate operation of a CNOT gate are shown in Figure 1.1 (b).

iii) **TOFFOLI** [20] is a 3×3 gates. The output of the control lines remain the same as the input of the control lines and the target line output is inverted if both the control lines are logic value 1; otherwise, output remains unchanged. It has the bijection relation with the input and output combination represented as $(A, B, C) \rightarrow (A, B, AB \oplus C)$. The symbolic representation and gate operation of a TOFFOLI gate are shown in Figure 1.1 (c).

iv) **Multiple-control TOFFOLI** (MCT) gate [20] is the logical extension of TOFFOLI gate is also known as k-CNOT gate. Here, k is number of control lines $x_1, x_2, \ldots, x_{n-1}$ and there is one target line x_n. Therefore, k-CNOT gate has k+1-inputs and k+1-outputs. The reversible logic function of k-CNOT gate is expressed as $(x_1, x_2, \ldots, x_{n-1}, x_n) \rightarrow (x_1, x_2, \ldots, x_{n-1}, x_1.x_2 \ldots x_{n-1} \oplus x_n)$. In k-CNOT gate operation, all the outputs of the positive control (denoted as •) lines remain the same as the input and the logic value of the target line t (denoted as $\oplus$) is only inverted if all the positive control input lines are at logic value 1. The symbolic representation and gate operation of a TOFFOLI gate are shown in Figure 1.1 (d).

v) **FREDKIN** [21] is a 3×3 gate that has one control line and two target lines. The control input remains unchanged and the output of the other two target inputs are swapped if the control input is logic value 1. The symbolic representation and gate operation of a TOFFOLI gate are shown in Figure 1.1 (e).

vi) **PERES** [22] is also a 3×3 gate. The output of the control input remains same and the output of the target input for the first gate is inverted if the control input is set as logic value 1. The output of the target input for the second gate is inverted if both control inputs are set as logic value 1. The symbolic representation and gate operation of a PERES gate are shown in Figure 1.1 (f).

vii) **SWAP** [23] has 2×2 gates which exchange the inputs. Figure 1.1 (g) shows the symbolic representation and gate operation of a SWAP gate.

1.1.3 Reversible Circuits

A reversible logic circuit realizes the reversible logic function and it can be implemented by a linear cascade of reversible gates [26], where fan-out and feedback connections are not directly allowed [16]. Consider a reversible circuit C=$g_1, g_2, \ldots, g_d$, where d is the total number of reversible gates, and also it is considered the circuit length. Suppose, gate g_i is the ith gate, for $1 \leq i \leq d$, then gate g_i is only active when the gate g_{i-1} has completed the gate operation. In a reversible circuit, each gate is categorized by the level, and the number of levels depending on the circuit length. The general structure of the reversible circuit with the cascade of d reversible gates is shown in Figure. 1.2.

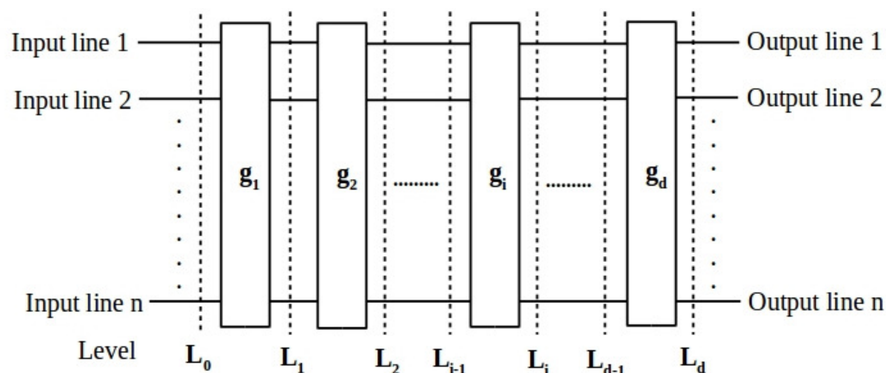


Figure 1.2: *Reversible Circuit Structure.*

Example 1.1.1. Figure 1.3 (a) and (b) show the reversible circuit based on NCT library and GT library, respectively. In Figure 1.3 (a), the first, second, and third gates are NOT, CNOT and TOFFOLI gate, respectively. and these gates form the NCT library. Figure 1.3 (b) contains only multiple-control Toffoli (MCT) gates and these gates form the generalized Toffoli gate (GT) library.

Example 1.1.2. Figure 1.4 illustrates the propagation of input vector “1010” to the primary output vector “ 1100”. Since all the k-CNOT gates are reversible, so each gate produces a unique output at each level during the gate operation. Here, the output vector “1110” is generated by gate g_3, which lies between levels 3 and 4, when the input vector “1010” is applied. Furthermore, if any gate produces different vector at their corresponding level, then a different output vector is produced in place of “1100” at the primary output level in the circuit.

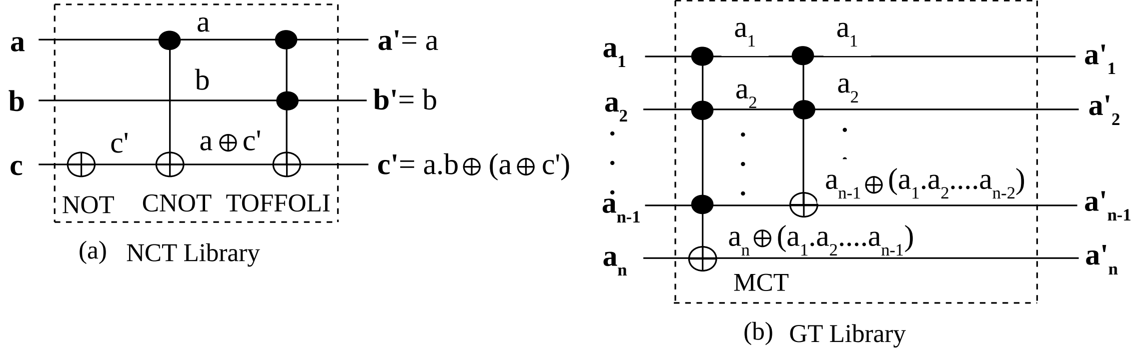


Figure 1.3: *Illustration of NCT and GT based Reversible Circuit*

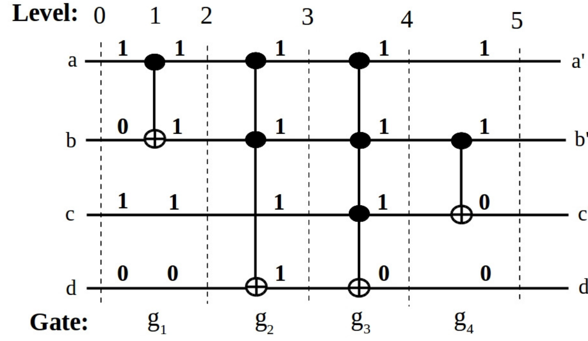


Figure 1.4: *k-CNOT Reversible Circuit comprising of* ***two*** *1-CNOT,* ***one*** *2-CNOT, and* ***one*** *3-CNOT gate*

1.1.4 Fault Models in Reversible Circuit

In a circuit design domain, there is a massive number of physical defects in a chip, and also it is impossible to count and analyze all possible faults. For evaluating the physical defects, we can not directly apply the mathematical treatment of testing. Thus, we need some abstract representations of these physical defects. These abstract representations have been formulated based on the mathematical model called a fault model. More precisely, a fault model is a mathematical model that describes the different levels of abstraction of physical fault in a system. The level of abstraction can be defined as behavioral, functional, structural, and geometric [31]. Based on the process of developing the mathematical mechanisms, the fault model can be applied to evaluate the faults and also helps to reduce the complexity of detecting the faults. Therefore, the fault model is an abstract representation of the physical defects in the system.

Let us assume that FM is the fault model that abstracts the faults in a way such that gates are assumed to be fault-free; however, the interconnection of gate netlist is faulty. Let $Z(x)$ be the logic function of a circuit C, where x represents an input vector assigned by the input lines of the circuit. Here, fault f denotes the faulty interconnection of the gate netlist. Due to the presence of fault f, the function $Z(x)$ of the circuit C transforms to the new function $Z_f(x)$, which is realized by the new circuit C_f. More precisely: FM=$\{Z(x) \stackrel{f}{\Longrightarrow} Z_f(x) \mid Z(x) \in C$ and $Z_f(x) \in C_f$, $\exists\, x$ where, $Z(x) \neq Z_f(x)\}$ represents the fault f in the circuit.

Depending on the reversible gate operations and structure of the circuit, few more fault models are required in addition to traditional fault models which are needed for detecting all faults in reversible circuits. Several fault models are applied to the NCT and GT libraries. Some of the fault models for reversible circuits are Stuck-at Fault (SAF) [33], Bridging Fault (BF) [3], Missing Gate Fault [7,8], etc. In this section, we discuss these fault models in details.

1.1.4.1 Stuck-at Fault Model (SAF)

The structural behavior of stuck-at fault model as described for the conventional logic circuit is also used in the reversible circuit testing. The stuck-at faults occur in a circuit when one of its inputs or outputs are fixed at either logic value 0 (stuck-at-0) or logic value 1 (stuck-at-1) [31,44]. If occurrence of a stuck-at fault is limited only to one line in the circuit, then it is called as the single stuck-at fault (SSF) and if the stuck-at fault is involved with more than one line, then it is called as multiple stuck-at fault (MSF). Patel et al. [33] stated that any test set is complete for detecting the stuck-at faults, if and only if each line at every level can be set to both 0 and 1 by the test sets.

Example 1.1.3. Figure 1.5 (b) and (c) show the effect of the single stuck-at faults and multiple stuck-at faults in *ham3_tc* benchmark reversible circuit, respectively. In Figure 1.5 (b), stuck-at 0 fault occurs at input line '*a*' at level 1 and the effect of this fault is propagated to the primary output of the circuit. The test vector required to detect this fault is $\langle 0\ 1\ 1\rangle$ or $\langle 1\ 0\ 0\rangle$ or $\langle 1\ 0\ 1\rangle$ or $\langle 1\ 1\ 0\rangle$, as depicted in Column 1 of Table 1.3. In Figure 1.5 (c), let multiple stuck-at fault (MSF), i.e., stuck-at 0 and stuck-at 1 occur at line '*a*' and '*c*', respectively at level 1. Here, if we apply the test vector $\langle 0\ 1\ 0\rangle$ to the circuit, the fault-free output would be $\langle 0\ 0\ 1\rangle$, as shown in column 2 (Fault-Free output) of Table 1.3. However, in the presence of the MSF fault, the output

will be ⟨0 1 1⟩ (marked as bold), as depicted in Column 4 of Table 1.3. Hence, the test vector ⟨0 1 0⟩ is capable of detecting all the MSF faults as shown in Figure 1.5. All other error outputs for the MSF are demonstrated in Column 4 (marked as bold for faulty outputs) of Table 1.3.

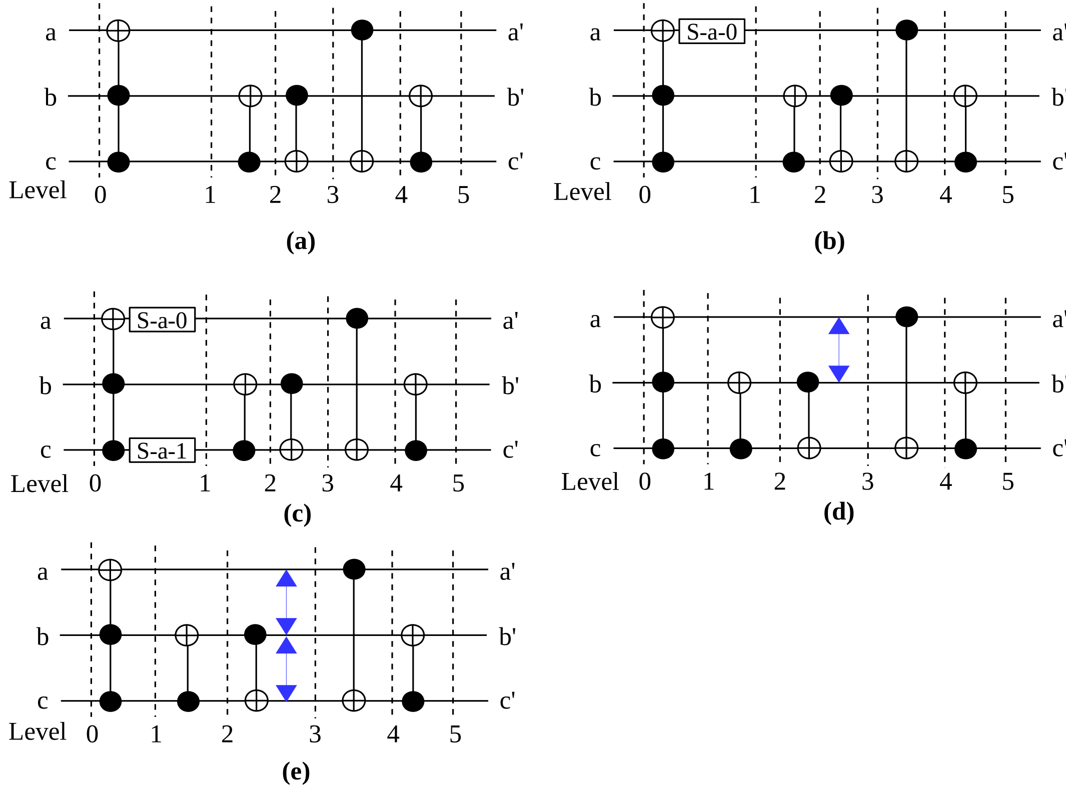


Figure 1.5: *Illustration of Reversible ham3_tc benchmark circuit: (a) Fault-free circuit, (b) SSF, (c) MSF, (d) SBF, and (e) MBF*

1.1.4.2 Bridging Fault Model (BF)

The bridging fault is considered as a structural fault model in the reversible circuit similar to the conventional circuit. A bridging fault occurs when two or more signal lines (wires) in the circuit are unintentionally or accidentally shorted together to create a wired logic that gives a faulty output of the circuit [34]. More precisely, the logic value on a signal line can be effected by the logic value of the other signal lines which are coupled or shorted together. The connection between the two lines, the logical effects of this fault are categorized by wired-OR and wired-AND bridging faults [31]. The bridging

Table 1.3: *Faulty and fault-free outputs for ham3_tc benchmark circuit of Figure 1.5*

Inputs			Fault-Free output			Faulty outputs											
						SSF			MSF			SBF			MBF		
a	b	c	a'	b'	c'	a'	b'	c'	a'	b'	c'	a'	b'	c'	a'	b'	c'
0	0	0	0	0	0	0	0	0	**0**	**1**	**0**	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0	0	1	0	**0**	**0**	**0**	**0**	**0**	**0**
0	1	0	0	0	1	0	0	1	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**
0	1	1	1	0	0	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**
1	0	0	1	1	1	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	0	1	1	0	1	**0**	**1**	**0**	**0**	**1**	**0**	1	0	1	**1**	**1**	**1**
1	1	0	1	1	0	**0**	**0**	**1**	0	1	1	1	1	0	1	1	0
1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	**0**	**0**	**0**

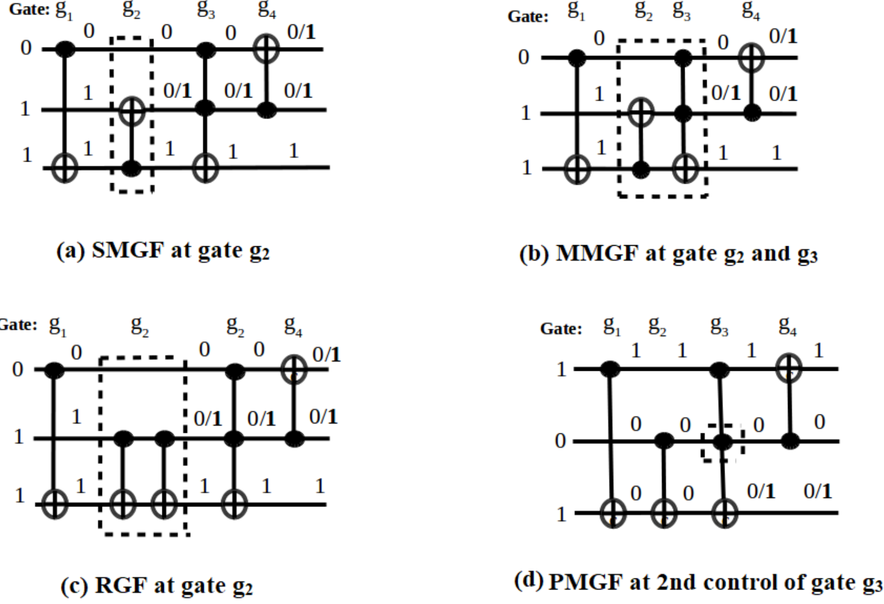
fault develops by shorted of two or more lines. The single bridging faults (SBF) refers to when only one pair of input lines exist. If more than two input lines are involved, then it is referred to as multiple bridging faults (MBF). In 2007, Rahaman et al. [3] stated that if a test set is capable to generate the opposite logic value to every pair of lines lying at the same level, then the test set is capable of detecting all the faults in that level.

Example 1.1.4. Figure 1.5 (d) shows that the two input lines 'a' and 'b' are connected together at level 3. The effect of this fault is represented by the truth table as shown in column 5 (mark as bold for faulty outputs) of Table 1.3. By comparing the fault-free with faulty output, the test vector $\langle 0\ 0\ 1\rangle$ or $\langle 0\ 1\ 0\rangle$ or $\langle 0\ 1\ 1\rangle$ or $\langle 1\ 0\ 0\rangle$ is able to detect the fault as mentioned in Figure 1.5 (d). Similarly, for the case of multiple bridging fault, the input lines 'a', 'b' and 'c' are connected with each other at level 3, as shown in Figure 1.5 (e). Column 6 (Faulty outputs) in Table 1.3 shows the possible test vectors as $\langle 0\ 0\ 1\rangle$, $\langle 0\ 1\ 0\rangle$, $\langle 0\ 1\ 1\rangle$, $\langle 1\ 0\ 0\rangle$, $\langle 1\ 0\ 1\rangle$, or $\langle 1\ 1\ 1\rangle$.

1.1.4.3 Missing-gate Fault Model

Apart from the traditional fault models, few more types of fault models need to be considered for describing the different kind of faults that occur in the k-CNOT based reversible circuit. These specific kind of faults are Single Missing-Gate Fault (SMGF) [7, 8], Multiple Missing-Gate Faults (MMGF) [8], Repeated-Gate Fault (RGF) [8], and

Partial Missing-Gate Fault (PMGF) [8].



(a) SMGF at gate g_2

(b) MMGF at gate g_2 and g_3

(c) RGF at gate g_2

(d) PMGF at 2nd control of gate g_3

Figure 1.6: *Illustration of Reversible Circuit nth_Prime3_inc for various fault conditions under missing-gate fault model*

i) **Single Missing-Gate Fault (SMGF):** A single missing gate fault occurs when any one of the k-CNOT gates is completely disappeared from the circuit. The physical justification of a SMGF is that the generated signal(s) for operating the gate is (are) short, missing, misaligned, or mistuned [7, 8]. The removal of a k-CNOT gate gives direct influences on the circuit functionality. Here, the total number of SMGFs is equal to the total number of gates N present in the circuit. In Figure 1.6.(a), g_2 gate is missing which is marked by a dotted box. If we apply input test vector $\langle 0\ 1\ 1 \rangle$ to the faulty circuit, the output would be $\langle 1\ 1\ 1 \rangle$ instead of normal primary output $\langle 0\ 0\ 1 \rangle$. Based on the k-CNOT gate operation, the SMGF can be detected by applying logic value 1 to the all positive control line ($\bullet$). Unconnected and target lines will be set as logic value 0 or 1 in the missing k-CNOT gate [8].

ii) **Multiple Missing-Gate Faults (MMGF):** Multiple missing gate faults (MMGF)

cause the complete removal of two or more consecutive k-CNOT gates. According to the authors in [8], the assumption of physical justification for the MMGF is that the gate operations implemented using laser is more likely to be disturbed for a period of time exceeding one gate operation. Hence, always consecutive gates are missing. Figure 1.6.(b) shows that g_2 and g_3 gates are missing which is marked by the dotted box. After applying the input test vector $\langle 0\ 1\ 1 \rangle$ the output vector would be $\langle 1\ 1\ 1 \rangle$ instead of $\langle 0\ 0\ 1 \rangle$. Furthermore, every SMGF is a subset of MMGFs [8].

iii) **Repeated-gate Fault (RGF):** A repeated-gate fault occurs when an unwanted k-CNOT gate replaced by multiple instantiations of the same gate. The physical justification for an RGF is the occurrence of long or duplicate pulses [8]. The effect of RGF is shown in Figure 1.6 (c), where g_2 gate is repeated once more in the circuit. If we apply the input test vector $\langle 0\ 1\ 1 \rangle$ to the circuit, then the output of the fault-free circuit is $\langle 0\ 0\ 1 \rangle$, but due to the presence of RGF, the output would be $\langle 1\ 1\ 1 \rangle$. Based on the RGF model, if the repeated gate occurs in even numbers of instances, then its effect is similar to the impact of SMGF; otherwise, the fault is redundant [8].

iv) **Partial Missing-Gate Fault (PMGF):** A partial missing-gate fault occurs in the circuit when one or more control connections are missing. The physical justification of a PMGF is defined as partially misaligned or mistuned gate pulse [8]. If in any k-CNOT gate, the control connection is missing only once (k=1), then it is called as the first-order PMGF, and control connections are missing more than once ($k > 1$), then its referred to as higher-order PMGF. Figure 1.6.(d) shows a first-order PMGF where the second control of gate g_3 is missing. If we apply input test vector $\langle 1\ 0\ 1 \rangle$ then the faulty output would be $\langle 1\ 0\ 1 \rangle$ instead of normal output $\langle 1\ 0\ 0 \rangle$ due to the presence of PMGF at gate g_3. It has been shown [8] that for the detection of a PMGF (first or higher order), the logic value 0 is set at least one of the affected control connections and the logic value 1 is applied to all other control connections. Remaining unconnected lines and target connections are set

as logic value 0 or, 1.

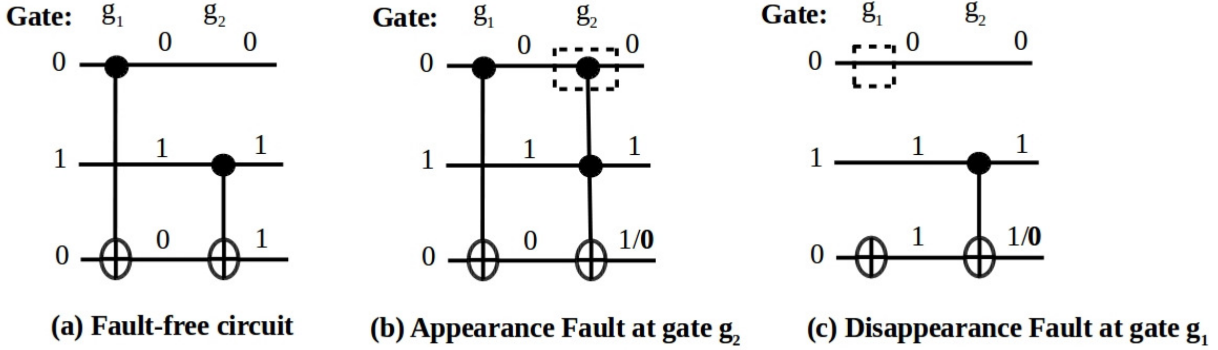


Figure 1.7: *Illustration of Crosspoint faults in Reversible Circuit*

1.1.4.4 Crosspoint Fault Model

The Crosspoint fault model is developed based on Programmable logic array (PLAs) [35]. Though a reversible circuit is a very different structure from a PLA, they are both based on a regular array design. The structure of the gate netlist in the reversible circuit is formed based on the regular array. Thus, this fault model may be effective to analyze the faults in reversible circuits. The Crosspoint fault consists of two types of faults. In k-CNOT gate netlist, when one or more additional control points are added erroneously, it leads to appearance faults. On the other hand, when one or more control points are removed erroneously, then it is called disappearance faults. The PMGF under the missing-gate fault model is the subset of the Crosspoint fault model [35]. Figure 1.7 (b) illustrates the functionality effects of appearance crosspoint fault where the first control point is erroneously added (marked as dotted box) at gate g_2. Thus, if we apply the test vector $\langle 0\ 1\ 0 \rangle$ to the circuit as shown in Figure 1.7 (b), then faulty output $\langle 0\ 1\ 0 \rangle$ generates instead of fault-free output $\langle 0\ 1\ 1 \rangle$. Similarly, Figure 1.7 (c) demonstrates the functionality effects of disappearance fault that is first control point of gate g_1 is erroneously removed (marks dotted box), due to that faulty output $\langle 0\ 1\ 0 \rangle$ generates instead of fault-free output $\langle 0\ 1\ 1 \rangle$.

1.2 Motivation and Objectives of the Work

The use of reversible circuits is gaining importance in the research community due to its reversible nature and lossless computation [12, 14, 15]. The information loss is related to the power dissipation [11] and reversibility ensures the identification of an input for a given output [13]. To ensure the reliability of a system, one needs to give the guarantee of a fault free system. To identify the fault in a circuit, it is essential to generate the required test vectors to identify the faults.

Exhaustive search technique is used to find a minimal complete test set [45], but the computational cost is high when the circuit size is large. Reversible circuits are built with cascading of reversible gates, so fault may occur at input, output and intermediate levels. Depending on the number of signal lines and number of gates of a reversible circuit, there is a possibility to get a large number of faults in a reversible circuit. Therefore several approximate and heuristic search techniques are being proposed to generate the test set for reversible circuits and these methods have their own merits and demerits. Most of the proposed methods in the literature mainly considered a particular fault model to get the set of test vectors. Generally, it is observed that while trying to reduce the size of test set, the percentage of fault coverage is less. On the otherhand, while trying to cover more faults the size of test set increases. Also, if more numbers of faults are being tried to consider by a method, then the complexity of computation increases.

By considering all these issues of test set generation for reversible circuits, we try to generate the minimal complete test sets for some fault models in reversible circuits. First, we consider the bridging fault models at the input level of reversible circuits and identify a minimal complete test to detect the bridging faults at the input level for any reversible circuit. It is also established that by adding a particular test vector to the test set, stuck-at faults can also be detected. Next, we extend the work to find the minimal complete test set to find intra-level single and multiple bridging faults, which may be extended to find intra-level stuck-at faults also. Moreover, the complexity for generating the complete test set for an intra-gate level fault is high, because the reversible gate at each level is dependent on its previous level of the circuit. Our next work is related

to find the test set to detect multiple missing gate faults. Fault localization is also an important problem in circuit testing domain and finally, a method has been proposed for localization of faults for missing gate fault model.

The main objectives of this thesis are summarized as follows:

- Test set generation to identify the bridging faults and stuck-at faults at input level of reversible circuits. The generated test set is a minimal complete test set.
- Test set generation for identifying input and intra-level bridging faults in reversible circuits. Both single and multiple bridging faults are being covered by the proposed method.
- Test set generation for identifying multiple missing gate faults.
- Determination of the location of a missing gate fault in reversible circuits.
- Exploring to get efficient methods for generating minimum test sets with low computational complexity so that methods are scalable also.

1.3 Related Work

The generation of complete test for a traditional circuit and as well as the reversible circuit with a minimal test set is an NP-hard problem [6]. For extracting a minimal complete test set, the exhaustive search technique is required [45]. Various testing approaches have been proposed previously in reversible circuits which are governed by the two main different methodologies for offline testing. The exact automatic test pattern generation (ATPG) provides the minimal complete test set, but the computational cost is high when the circuit size is large, and the randomized (heuristic) approaches for ATPG provide the complete test set, but that may not give a minimal solution [6]. On the other hand, the design for testability (DFT) methodology also generates the minimal complete test set with an additional burden of extra circuitry (i.e., DFT circuitry) [39]. The computation cost for generating the complete test set for intra-level faults is high,

because the reversible gate at each level is dependent on its previous level of the circuit. Suppose, in the stuck-at and bridging fault model, the fault may appear in the initial level or primary output level or maybe intra-gate level. Due to this, the size of generated complete test size may have exponential growth [43]. Therefore, generating the efficient minimal complete test set is the challenging task to detect all the possible faults in each level of the circuit. In 2004, Patel et al. [33] proposed an Integer Linear Programming (ILP) based deterministic approach with binary decision variables to construct the minimal test set for detecting stuck-at faults as well as cell-faults in reversible circuits consisting of gates from the NCT library. Because of its computational complexity, the method is applicable only for small circuits. To handle larger circuits, the authors suggested a circuit decomposition approach. The authors in [36,37,46] proposed an ATPG with DFT for detecting stuck-at faults in a k-CNOT reversible circuit. However, all these methods required an additional extra circuitry overhead, which leads to an increase in the quantum cost of a testable circuit. In 2007, Bubna et al. [4] proposed a Design-For-Test (DFT) methodology for detecting bridging faults in reversible circuits. This method required ($\lceil \log_2 n \rceil + 3$) test vectors for n input wires. Additional input wires are added to Toffoli gates to realize the DFT circuit. Rahaman et al. [3] showed that a test set of cardinality ($d \log_2 n$) is sufficient to detect all intra-level bridging faults, where d is the levels of the circuit and n is the number of input lines. In 2008, Sarkar et al. [1] presented a polynomial time algorithm for generating a set of universal test vectors for detecting all single and multiple input bridging faults of reversible circuits.

In 2004, Hayes et al. [7] showed that all Single Missing Gate Faults (SMGFs) (considered one gate missing at a time) in an N-gate k-CNOT based circuit can be detected with a maximum of $\lceil N/2 \rceil$ test vectors. Also, it is shown that Missing Gate Faults (MGFs) are detectable by a Design For Testability (DFT) method with one wire, and several 1-CNOT gates using a single test vector. In 2008, Rahaman et al. [40] proposed an augmented k-CNOT circuit by adding only one extra line along with duplication of each k-CNOT gate. It is also shown that a universal test set of size $(n+1)$ is sufficient to detect all SMGFs along with fault models, viz., RGFs and PMGFs. In 2016, Nagamani

et al. [6] proposed exact ATPG algorithms for generating the complete test set which can detect the single and multiple stuck-at faults, single and multiple missing gate faults, repeated gate faults, and partial missing gate faults. The proposed algorithms provide the optimal solution, but the complexities of these algorithms are exponential with respect to the gate count and the number of lines of the circuit. In 2017, Prakash Surhonne et al. [5] has generated Automatic test patterns for MMGFs detection; however, they considered that only two gates are missing. Based on the generated SMGFs which are stored in a Binary Decision Diagram (BDD) and the scheme generates the test patterns to cover all MMGFs by dependency analysis between two gates.

Another important area of testing of the reversible circuit is fault localization. Very few works are found in literature for fault localization based on the different fault models for reversible circuits. The authors in [47] proposed an adaptive tree-based approach for fault localization to detect and locate stuck-at faults in a reversible circuit. To speed up the computation of symmetric adaptive tree method, the authors also proposed a new greedy direct tree generation algorithm that eliminates the fault table generation and dynamically creates the adaptive fault tree without generating all test patterns. However, both approaches are applicable to small circuits because the simulation size of the fault table increases exponentially for large circuits. In 2005, Pierce et al. [48] also proposed a method to generate the fault table using all the possible vectors in the circuit. Based on the fault table, the fault localization tree is constructed with additional constraints for identifying the faulty behavior of the circuit. In 2011, Zhang et al. [49] proposed a new fault diagnosis approach for single missing control fault model in reversible circuits. This approach provided new methods for Diagnostic Test Pattern Generation (DTPG). Rahaman et al. [50] proposed a fault diagnosis technique in k-CNOT based reversible circuit using the missing gate fault model. The proposed technique required a universal test set (UTS) of size (n+1) that detects all single missing-gate faults (SMGFs), repeated-gate faults (RGFs), and partial missing-gate faults (PMGFs). This method requires one extra control line along with duplication of each k-CNOT gate of the original (n $\times$ n) k-CNOT based reversible circuit resulting in high area overhead.

1.4 Summary of Contributions

The main emphasis of this thesis is to generate the test patterns under various fault models and develop fault localization schemes for reversible circuits. This thesis has four major contributions which are discussed in the subsequent subsections.

1.4.1 Test Generation for Input Stuck-at and Bridging Faults in Reversible Circuits

In the first work of the thesis, the test pattern generation for single input bridging and stuck-at fault is considered. This work discusses a scheme for minimal test generation to detect all input bridging and stuck-at faults in reversible circuits. A method has been proposed to find the minimal test set to detect all single input bridging faults and formally prove that $\lceil \log_2 N \rceil$ number of test vectors is necessary and sufficient for detection of all the input bridging faults of a reversible circuit with N inputs. A simple scheme to identify the input stuck-at faults with the generated test set of bridging faults is also explored. It is established that addition of a particular test vector is sufficient to cover all the stuck-at faults at the input. Experimental results showed that the generated test set of the proposed scheme covers 100% faults and the test set size is much smaller as compared to the existing methods.

1.4.2 Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions

The second work of this thesis is related to the generation of the test patterns to detect input and intra-level bridging faults. In this work the method of the first work of the thesis is extended to find the test patterns to detect intra-level bridging faults in reversible circuits. Reversible circuits are constructed by cascading connection of reversible gates. Fault may occur in signal lines at any level. Most of the possible intra-level bridging faults, namely, Single Input Bridging Faults (SIBF), Multiple Input bridging faults (MIBF), Single Intra-level Bridging Faults (SIRBF) and Multiple Intra-level Bridging Faults (MIRBF) are addressed in this work. The test patterns generated for single input bridging faults are applied as local test patterns at each level. With the help of the

local test patterns, a path is generated from each level to input level using backtracking method. This method is based on the reversible property– "for every vector at the output of a reversible gate; there exists a unique vector at the input." After generating the paths, if some paths are not complete then complete path is generated with forward simulation method. A path is said to be complete if it traces from output to input level. Moreover, the path-level expression is introduced in the work to generate the complete test set. The path-level expression has the ability to capture the level-wise information on a given reversible circuit. After collecting all the path-level expressions, a matching process is applied to collect the required paths, which are used to generate the test set. It is also established that the generated test is the minimal and complete one. The computational complexity of the proposed method is found to be in logarithm order of the number of inputs and linear to the number of gates of a reversible circuit. The algorithm of the proposed complete test set generation method has been implemented and applied to various benchmark circuits based on NCT (NOT, C-NOT and TOFFOLI) and GT (Generalized TOFFOLI) gate libraries [24] and [25]. Based on the experimental results, it is evident that the number of test vectors generated by our proposed method is less or equal to the existing methods. Moreover, the proposed method covers more types of fault as compared to existing methods and achieves 100% fault coverage. Extension of this work to detect the intra-level stuck-at faults is straight forward, similar to the first work of this thesis.

1.4.3 Test Generation for Multiple Missing-Gate Faults in Reversible Circuits

The problem of identifying Multiple Missing Gates Faults (MMGF) is considered as the third work and Missing Gate Fault model is specific to reversible circuits only. This work presents a scheme for generating the complete test set to detect single and any number of consecutive multiple missing-gate faults in the reversible k-CNOT based circuits. The complete test set generation method is twofold. First, the test vectors for SMGFs (Single Missing Gate Faults) are generated by applying the local test pattern to all the gates of reversible circuits using the reverse simulation method. Secondly, based on the

complete test set for SMGFs and the structure of the k-CNOT based circuit, the test set for detecting all the MMGFs is constructed. This phase of the proposed method generates the complete test set which can detect the multiple missing gate faults, but the test set is not minimal. For achieving the minimality, a table is constructed covering the row and column faults, which is formulated as an ILP problem. The experimental results demonstrate that the minimized test set size is smaller or similar as compared to existing methods and attains 100% fault coverage. Empirically it is also established that the generated test vectors of this method also contributes the fault detection for the SAFs (Stuck-At Faults), PMGFs (Partial Missing Gate Faults), and appearance of crosspoint faults, but the fault coverage is not 100%. The fault coverage analysis has been done without modification of the generated complete test set. The correlations of these fault models indicates that there is a possibility to reconstruct the generated complete test set to cover all the possible faults in reversible circuits.

1.4.4 Fault Localization for Missing Gate Faults in Reversible Circuits

In this work, the fault localization method is presented to obtain the exact location of single and multiple missing gate faults in reversible logic circuits. The primary focus of this proposed method is to extract the unique test pattern for each missing gate faults in the k-CNOT based reversible circuit. The test set generated in the third work of the thesis is used to generate the test sequence or response for each single and multiple missing gate faults, and the unique sequence of test patterns are used for identifying the exact location of the faults. Based on these unique test patterns, a fault localization tree is constructed. Next, the traversal process is applied to the fault localization tree to determine the presence and location of the faults. Moreover, the proposed method is also capable of evaluating the equivalent faults in a given reversible circuit. Finally, we provide our experimental results that are validated on several reversible benchmark circuits [25]. From this proposed method, it is evident that the complete test set for detecting faults can be used further for fault localization of reversible circuits.

1.5 Organization of the Thesis

This thesis consists of six chapters including this introduction chapter. The remaining part of the thesis is organized as follows:

- **Chapter 2:** Test Generation for Input Stuck-at and Bridging Faults in Reversible Circuits
  - In this chapter, a minimal complete test set generation method is proposed which is capable of detecting all the possible single input bridging and single input stuck-at faults in reversible circuits. The existing research work related to this problem is also included in this chapter.
- **Chapter 3:** Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions
  - This chapter presents an ATPG method to determine the minimal complete test set for detecting the Single Input Bridging Faults (SIBF), Multiple Input Bridging Faults (MIBF), Single Intra-level Bridging Faults (SIRBF), and Multiple Intra-level Bridging Faults (MIRBF). Related work for this problem is also briefly introduced in this chapter.
- **Chapter 4:** Test Generation for Multiple Missing-Gate Faults in Reversible Circuits
  - This chapter proposes a scheme for generating the complete test set to detect single and any number of consecutive multiple missing-gate faults in the reversible k-CNOT based circuits. The correlation of Missing Gate Faults with other fault models such as SAFs, PMGFs and crosspoint faults is also included in this chapter. Related research work is also presented.
- **Chapter 5:** Fault Localization for Missing Gate Faults in Reversible Circuits
  - This chapter presents a novel fault localization method to obtain the exact location of single and multiple missing gate faults in reversible circuits.

- **Chapter 6:** Conclusion and Future Work
  - This chapter concludes the thesis by summarizing the main contributions and highlighting some important areas for future research.

1.6 Conclusion

In this introductory chapter, we explained the overview of the thesis with some necessary basic background of the reversible logic circuits. The main motivation and objectives are also reported therein. A brief summary of the main contributions is reported in this chapter. In the next chapter, we present our first contribution which is related to single input stuck-at and bridging faults in reversible circuits.

Chapter 2

Test Generation for Input Stuck-at and Bridging Faults in Reversible Circuits

2.1 Introduction

In this chapter, we consider the problem of testing of reversible circuit for NCT or GT library-based reversible circuits, specifically targeting test pattern generation for input stuck-at and bridging fault models in reversible circuits. The concept of stuck-at and bridging fault model as described for conventional logic is also used in the reversible circuit testing. In this present work, we consider particularly the single input stuck-at faults and bridging faults, where the fault occurs at the input level of the reversible circuits. The bridging faults between two lines at the same level can be detected by a test vector that sets the two lines to opposite logic values, i.e., 01 or 10, which was reported in [31]. Stuck-at faults at any particular level also require lines to be set to 0 or 1 to detect the faults at that level. We first generate the test patterns to detect single input bridging faults, and then subsequently reconstruct the test vectors for detecting single input stuck-at faults. The method produces a complete test set, which is also the minimal test set for detecting both single input bridging faults and single input stuck-at faults, of reversible circuits.

At first, a method is proposed to generate the test vectors to detect all input bridging faults of reversible circuits and formally we prove that the minimal test set contains $\lceil \log_2 N \rceil$ test vectors to detect all single input bridging faults of a reversible circuit with

N input-lines. Next, we show that by adding only one test vector to the generated test set of single input bridging faults can also detect all single input stuck-at faults. Based on the evidence of the proposed method, it is established that there is a correlation that allows the possibility for targeting at bridging fault model can also be applied for detecting faults under the stuck-at fault model. The experimental results show that the test set size is smaller as compared to some of the existing methods and covers 100% faults. Moreover, it has been observed that the proposed method is scalable to large circuits based on NCT and GT library.

This chapter is organized as follows: Section 2.2 provides an overview of related works which are relevant to our present work. Section 2.3 describes our proposed method for generating a complete minimal test set to detect input bridging and stuck-at faults in the reversible circuits. The experimental results and discussions of the proposed method are presented in Section 2.4. The concluding remarks are presented in Section 2.5.

2.2 Related Work

Some of the previous works that are relevant to the present work are briefly reviewed in this section. In 2004, Patel et al. [33] proposed an Integer Linear Programming (ILP) based approach with binary decision variables to construct the minimal test set for detecting stuck-at faults as well as cell-faults in reversible circuits with gates from the NCT library. Because of its computational complexity, the method is applicable only for small circuits. To handle larger circuits, the authors suggested a circuit decomposition approach. Patel et al. [33] stated that any test set is complete for detecting the stuck-at faults, if and only if each line at every level can be set to both 0 and 1 by the test sets. Also, the authors showed that if any test set is complete for the single stuck-at faults, then the same test set is also complete for multiple stuck-at faults.

In 2005, a method was proposed [46] for detecting all stuck-at faults that reduce the number of test vectors required to 3; however, an additional line and one extra control in every gate are required as overhead. In 2008, Ibrahim et al. [36] proposed a method for offline testing of multiple stuck-at faults in reversible circuits with only 2 test vectors

by adding a maximum of two input lines. In this method some K-CNOT gates are replaced by $(K + 1)$-CNOT gates and some are replaced by $(K + 2)$-CNOT gates. In 2011, Nayeem et al. [51] proposed a technique to convert an ESOP-generated reversible circuit into an online testable circuit. Here one needs to add some CNOT gates and replace the Toffoli gates with the extended Toffoli gates (ETG). Overall, this method required double the number of CNOT gates.

In 2007, Bubna et al. [4] proposed a Design-For-Test (DFT) methodology for detecting bridging faults in reversible circuits. The proposed method required $(\lceil log_2 N \rceil)+3$ test vectors for N input wires. Additional input wires are added to Toffoli gates to realize the DFT circuit. In 2018, Gaur et al., [38] presented a test methodology for the detection of stuck-at faults in reversible circuits based on multiple controlled toffoli gates. The circuit is modified by adding extra k-CNOT gates for those gates where the input test vector is likely to be inverted in the circuit. This DFT method showed that the $(n+1)$ dimensional general test set containing only two test vectors which are capable of detecting all the single and multiple stuck-at faults in k-CNOT based reversible circuits. Further, this DFT method is extended to identify the location of stuck-at faults in the circuit.

In 2007, Rahaman et al. [3] showed that a test set of cardinality $(d \log_2 n)$ is sufficient to detect all intra-level bridging faults in an n-input and n-output reversible circuit with d levels. In 2008, Sarkar et al. [1] presented a polynomial time algorithm for generating a set of universal test vectors for detecting all single and multiple input bridging faults of the reversible circuit. In 2011, Sarkar et al. [2] proposed a polynomial time algorithm to generate the universal test vectors for detecting the single and multiple input bridging faults in reversible circuits. The universal test generation method is based on the shift operation of the unitary matrix, where unitary matrix is mapped to the identify matrix by the shift operation. This method shows that $\lceil n/2 \rceil$ number of test vectors is sufficient to detect all the input bridging faults in a given reversible circuit.

It is observed from the literature that while trying to reduce the number of test vectors for complete fault coverage, there is an additional burden of extra circuitry in

the form of additional input lines or control connections for reversible gates.

In this work, we propose a method to find the minimal set of test vectors to detect all possible input bridging faults without any additional hardware. Also, we show that by adding one more test vector, we can also detect all possible input stuck-at faults.

2.3 Proposed Method

In this section, we first present a method to find the minimum set of test vector to detect all input bridging faults in a reversible circuit. Next, we show that this test set can be enhanced by adding one more test vector to detect all the input stuck-at faults. Some definitions are presented, which are used in subsequent discussions.

Definition 2.3.1. *A test vector (TV) is a set of binary inputs that is applied to a reversible circuit for the purpose of testing. Binary inputs* $\langle b_1\ b_2\ \ldots\ b_N \rangle$ *are provided to the input lines* $L = \{L_1, L_2,\ \ldots,\ L_N\}$*, where N is the number of input lines. Let the test set be* $TS = \{TV_1,\ TV_2 \ldots,\ TV_l\}$*, for* $1 \leq j \leq l$*,* $TV_j = \langle b_{1j}\ b_{2j}\ \ldots\ b_{Nj} \rangle$*, where* b_{ij} *is the* i^{th} *bit of* j^{th} *test vector and* $b_{ij} \in \{0, 1\}$*. For the test vector* TV_j*, at least one bit* b_{ij} *must be different from some other bit* b_{kj}*, where* $k \neq i$*.*

Definition 2.3.2. *The set* $\mathbf{F_B}$ *is the set consisting of all single input bridging faults in a given N-input reversible circuit. The single input bridging fault* $F_{(i,j)}$ *involves the pair of input lines* $\{L_i, L_j\}$*, where* $\{L_i, L_j\} \subseteq L$*, and is denoted by* $\{L_i \leftrightarrow L_j\}$*. In other words,* $F_B = \{F_{(i,j)} \in \{L_i, L_j\} | \{L_i, L_j\} \subseteq L, F_{(i,j)} \Rightarrow \{L_i \leftrightarrow L_j\}\}$*.*

Example 2.3.1. For a reversible circuit with N input lines, the number of possible single bridging faults in any level will be ${}^{N}C_2$ or $C(N, 2)$. For example, if $N = 3$, the number of single bridging faults will be C(3,2) = 3, corresponding to the pairs $\{L_1, L_2\}$, $\{L_1, L_3\}$ and $\{L_2, L_3\}$. The corresponding fault set will be $F_B = \{F_{(1,2)}, F_{(1,3)}, F_{(2,3)}\}$.

Definition 2.3.3. *The test vector* TV_k *is capable of detecting the bridging fault* $F_{(i,j)}$ *if and only if* $b_{ik} \neq b_{jk}$*.*

Example 2.3.2. For the set of input lines L = $\{L_1, L_2, L_3\}$, the test vector $TV_1 = \langle 0\ 0\ 1 \rangle$ is capable of applying opposite logic values to the pair of input lines $\{L_1, L_3\}$ and $\{L_2, L_3\}$. Thus, TV_1 detects the faults $F_{(1,3)}$ and $F_{(2,3)}$.

Definition 2.3.4. *The test set* $\mathbf{TS_S}$ *is the minimal complete test set to detect all the single input stuck-at faults. Let* TV_t*=(*$\langle b_{1t}\ b_{2t}\ \ldots b_{Nt} \rangle$*) be a test vector in* TS_S*, where* b_{it} *is the* i^{th} *bit of the* t^{th} *test vector and* $b_{it} \in \{0,1\}$*. If* b_{it}*=0 then* TV_t *is capable of detecting stuck-at-1, otherwise, it detects stuck-at-0.*

Definition 2.3.5. *Let* $\mathbf{F_S}$ *denote the set of all single input stuck-at faults in a given N-input reversible circuit. Here, we are using the notation* $SA\text{-}b_i\text{-}L_i$ *for single input*

stuck-at fault, which denotes a stuck-at fault of ith bit b_i at the ith input line L_i. The stuck-at fault, SA-b_i-L_i / SA-$\overline{b_i}$-L_i can only be detected when input line L_i is set to $\overline{b_i}$ / b_i. In other words, F_S=$\{SA$-b_i-L_i, SA-$\overline{b_i}$-L_i | SA-b_i-$L_i \Rightarrow \overline{b_i}$ and SA-$\overline{b_i}$-$L_i \Rightarrow b_i$ where, $b_i \in L_i\}$.

To compute F_S for each input line L_i, b_i can be set to either 0 or 1. For N lines, the number of single input stuck-at faults will be $2N$.

2.3.1 Test Generation for Single Input Bridging Fault

In this section, we discuss the proposed method to generate a minimal test set for detecting all single input bridging faults in a given reversible circuit.

As discussed earlier, a test vector is able to detect a single bridging fault between two lines if it sets the two lines to opposite logic values. The proposed method generates a test set where this property is satisfied for every pair of lines and is hence complete. It is also established that the generated test set is minimal.

The basic idea behind the test generation method is explained below. Let us consider the test set $TS_B = \{TV_1, TV_2, \ldots, TV_{\lceil \log_2 N \rceil}\}$. Let the first test vector TV_1 consist of alternating 0's and 1's. The second test vector TV_2 consists of an alternate sequence of two consecutive 0's and two consecutive 1's. Similarly, the nth test vector consists of 2^{n-1} consecutive 0's followed by 2^{n-1} consecutive 1's at the N-input lines L_N, where $1 \leq n \leq \lceil \log_2 N \rceil$. This process continues, till we get the last test vector $TV_{\lceil \log_2 N \rceil}$. This is shown diagrammatically in Figure 2.1.

Example 2.3.3. The process of test set generation with the help of examples are shown next. For N=8, the test set TS_B will consist of three test vectors:TV_1=($\langle$0 1 0 1 0 1 0 1$\rangle$), TV_2=($\langle$0 0 1 1 0 0 1 1$\rangle$), and TV_3=($\langle$0 0 0 0 1 1 1 1$\rangle$). The total number of bridging faults at the input is $|F_B| = C(8, 2) = 28$. We observe that TV_1 detects the faults $F_{(i,j)}$, where $|j - i| = 1, 3, 5, 7$. Similarly, TV_2 detects the faults $F_{(i,j)}$, where $|j - i| = 2, 3, 6, 7$. Finally, TV_3 detects the faults $F_{(i,j)}$, where $|j - i| = 4, 5, 6, 7$.

For $N = 9$, TS_B will have four test vectors as $\lceil \log_2 9 \rceil = 4$. The four test vectors are, TV_1=($\langle$0 1 0 1 0 1 0 1 0$\rangle$), TV_2=($\langle$0 0 1 1 0 0 1 1 0$\rangle$), TV_3=($\langle$0 0 0 0 1 1 1 1 0$\rangle$) and TV_4=($\langle$0 0 0 0 0 0 0 0 1$\rangle$). The complete list of single input bridging faults for $N = 8$ and $N = 9$ of this example is shown in Figure 2.2.

We now present two lemmas with proof that establishes the completeness and minimality of the test set.

TV_1 = 0 1 0 1 0 1......... N-bits
2^0 consecutive 0's followed by 2^0 consecutive 1's

TV_2 = 0 0 1 1 0 0........... N-bits
2^1 consecutive 0's followed by 2^1 consecutive 1's

⋮

TV_n = 0 0 0 0 1 1 1 1 .... N-bits
2^{n-1} consecutive 0's followed by 2^{n-1} consecutive 1's

⋮

$TV_{\lceil \log_2 N \rceil}$ = 0 0 0....0 1 1 1....1.......... N-bits
$2^{\lceil \log_2 N \rceil - 1}$ consecutive 0's followed by $2^{\lceil \log_2 N \rceil - 1}$ consecutive 1's

Figure 2.1: *Test set for detecting single input bridging faults*

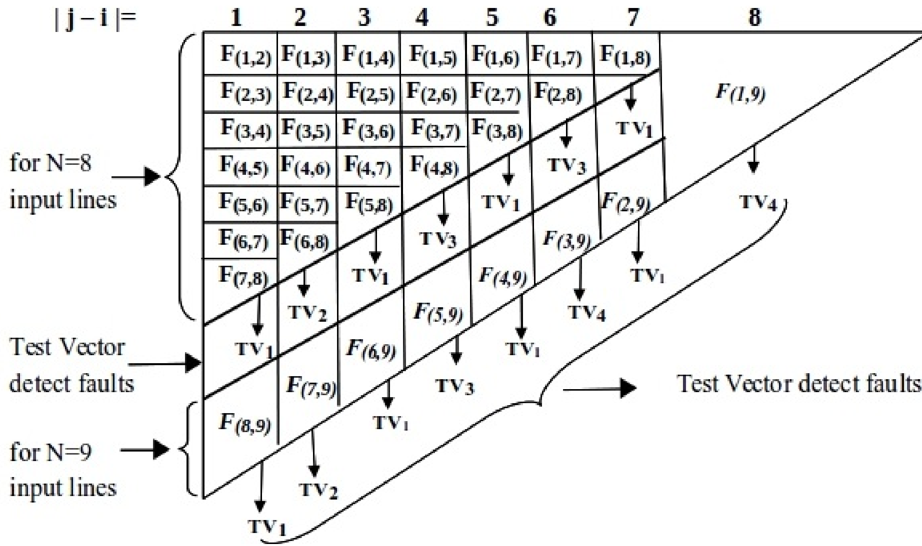


Figure 2.2: *Demonstration of single input bridging faults for N=8 and N=9*

Lemma 2.3.1. *The test set generated using the proposed method detects all single input bridging faults in a given N-input reversible circuit.*

Proof. Let us consider the test set for single input bridging faults TS_B=$\{TV_1, TV_2, \ldots, TV_{\lceil \log_2 N \rceil}\}$, where the size of any test vector (TV) is N; the number of input line is L_N for the given reversible circuit. The number of test vectors is $\lceil \log_2 N \rceil$. We know that all the single input bridging faults $F_{(i,j)}$ occur for all possible distance $k = |i - j|$,

where $1 \leq k \leq N-1$. The total number of test vectors generated by the proposed method is $n = \lceil \log_2 N \rceil$. The nth test vector TV_n=($\langle 0\ 0\ 0\ 0\ \ldots 1\ 1\ 1\ 1\ \ldots \rangle$), contains 2^{n-1} consecutive 0's followed by 2^{n-1} consecutive 1's. By this test vector, input 0 is provided to the first $N/2$ lines and input 1 is provided to the last $N/2$ lines. The distance between first input line L_1 having input 0 and first input line $L_{(N/2)+1}$ having input 1 is $N/2$ and these two input test bits can detect the single input bridging fault $F_{(1,(N/2)+1)}$. By this analogy, it is observed that the test vector TV_n can detect all the single input bridging faults $F_{(i,j)}$ where $(N/2) \leq |i-j| \leq N-1$. It is also observed that the fault $F_{((N/2),(N/2)+1)}$ between the input lines $L_{(N/2)}$ and $L_{(N/2)+1}$, which is one distance apart can also be detected by the test vector TV_n. But the fault $F_{(i,i+1)}$ where $1 \leq i \leq (N/2)-1$ cannot be detected by the test vector TV_n. So, the test vector TV_n can detect all possible single input bridging faults $F_{(i,j)}$ where the distance between input lines L_i and L_j lies in the range $[(N/2), \ldots, (N-1)]$. Now consider the $(n-1)^{th}$ test vector $TV_{n-1} = (\langle 0\ 0\ 0\ 0\ \ldots 1\ 1\ 1\ 1\ \ldots 0\ 0\ 0\ 0\ \ldots 1\ 1\ 1\ 1\ \ldots \rangle)$, which contains alternate sequence of 2^{n-2} consecutive 0's and 2^{n-2} consecutive 1's. In this test vector the entire test bits are divided into two parts and the bit pattern of first part is similar to the second part. First part contains the input bits for the lines from L_1 to $L_{(N/2)}$ and the second part contains the input bits for the lines from $L_{(N/2)+1}$ to L_N. With the similar reasoning, it can be stated that the test vectors TV_{n-1} can detect all possible single input bridging faults $F_{(i,j)}$ where the distance between input lines L_i and L_j lies in the range $[(N/4), \ldots, ((N/2)-1)]$. Similarly, the test vectors TV_2 can detect all possible single input bridging faults $F_{(i,j)}$ where the distance between input lines L_i and L_j lies in the range $[2, 3]$ and test vector TV_1 can detect all the single input bridging faults between adjacent lines. It is also observed that the test vector TV_1 can detect all faults $F_{(i,j)}$ where $|i-j|$ is an odd number. So, all faults are covered by some test vectors. Therefore, the test set TS_B is the complete test set for detecting all the single input bridging faults in a given reversible circuit. □

Lemma 2.3.2. *The test set TS_B is a necessary minimal test set for detecting all single input bridging faults in a given N-input reversible circuit.*

Proof. In Lemma 2.3.1, it is established that test set TS_B can detect all single input bridging faults of a given reversible circuit. The number of test vectors in TS_B is $\lceil \log_2 N \rceil$. Consider the kth test vector TV_k that contains an alternate sequence of 2^{k-1} consecutive 0's and 2^{k-1} consecutive 1's. This test vector can detect all the faults $F_{(i,j)}$, where the distance between input lines L_i and L_j lies in the range $[(N/2^{n-k+1}), \ldots, (N/2^{n-k}-1)]$. It is observed that every test vector TV_i of TS_B can detect all the faults $F_{(i,j)}$ where the distant between lines L_i and L_j belongs to a binary division of the entire range. So, if we remove any one test vector, then that binary division cannot be covered by the rest of the test vectors of TS_B and bound to miss some single input bridging faults. Also, if we replace the test vector TV_i by any other random test vector then some of the faults between some pairs of the input lines whose distance lies in the range $[(N/2^{n-k+1}) \ldots (N/2^{n-k}-1)]$ are bound to be missed. So, the test vector TV_k must be present in the test set TS_B, where $1 \leq k \leq \lceil \log_2 N \rceil$. □

Example 2.3.4. Consider the reversible benchmark circuit 3_17tc_tfc consisting of six gates as shown in Figure 2.3. The number of single input bridging faults is $|F_B| =$

$C(3,2) = 3$, and the required number of test vectors is 2. The two test vectors are: TV_1=(⟨0 1 0⟩) and TV_2=(⟨0 0 1⟩). Table 2.1 shows that TS_B is indeed capable of detecting all input bridging faults in the circuit.

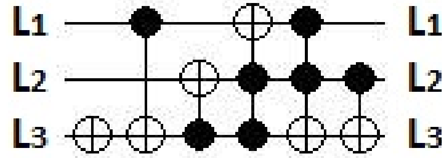


Figure 2.3: *3_17tc_tfc benchmark circuit*

Table 2.1: *Bridging fault coverage for the circuit 3_17tc_tfc*

Single input bridging fault	Test Set (TS_B)			Fault-free Output			Output			Output		
$F_{(i,j)}$							AND-bridging			OR-bridging		
	L_1	L_2	L_3	L_1	L_2	L_3	L_1	L_2	L_3	L_1	L_2	L_3
$F_{(1,2)}$	0	1	0	0	0	1	1	1	1	1	1	0
$F_{(1,3)}$	0	0	1	0	0	0	1	1	1	0	1	0
$F_{(2,3)}$	0	1	0	0	0	1	1	1	1	0	1	1

Example 2.3.5. Similarly, the reversible benchmark $4b15g_2$ consists of 15 gates as shown in Figure 2.4. The number of single input bridging faults for this circuit is $|F_B| = C(4,2) = 6$. Two test vectors are generated as: TV_1=(⟨0 1 0 1⟩) and TV_2=(⟨0 0 1 1⟩). Table 2.2 shows that the test set detects all single input bridging faults in the circuit.

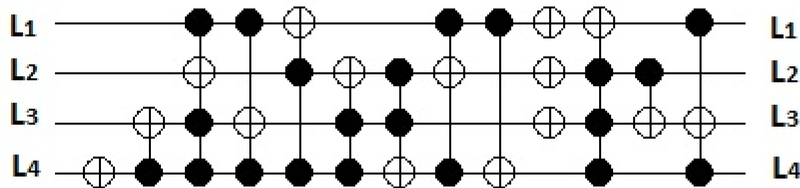


Figure 2.4: *4b15g_2 benchmark circuit*

Table 2.2: *Bridging fault coverage for the circuit* $4b15g_2$

Single input bridging fault	Test Set (TS_B)				Fault-free Output				Output				Output			
$F_{(i,j)}$									AND-bridging				OR-bridging			
	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4
$F_{(1,2)}$	0	1	0	1	1	0	1	0	1	1	0	0	0	0	1	1
$F_{(1,3)}$	0	0	1	1	1	1	1	0	1	1	0	0	0	1	1	1
$F_{(1,4)}$	0	0	1	1	1	1	1	0	0	1	0	1	0	1	1	1
$F_{(2,3)}$	0	1	0	1	1	0	1	0	1	1	0	0	1	0	1	1
$F_{(2,4)}$	0	0	1	1	1	1	1	0	0	1	0	1	1	0	1	1
$F_{(3,4)}$	0	1	0	1	1	0	1	0	0	0	0	0	1	0	1	1

2.3.2 Test Set Generation for Single Input Stuck-at Fault

We shall now show how a test set for detecting single stuck-at faults can be derived from the bridging fault test set as generated using the method explained in the previous subsection. We note that if a test set applies opposite logic values 0 and 1 to each of the input lines, then it can detect all single input stuck-at faults. The test set TS_B as discussed in the previous subsection may not satisfy this property for all cases, and hence it is unable to guarantee the detection of all single stuck-at faults.

Here we discuss how we can augment TS_B to generate a test set TS_S that can detect all single input stuck-at faults. Actually, we add one additional test vector to get the required test set as TS_S=$\{TV_1, TV_2, \ldots, TV_{\lceil \log_2 N \rceil}, TV_{\lceil \log_2 N \rceil+1}\}$. The extra test vector is generated as $TV_{\lceil \log_2 N \rceil+1}$=$(\langle b_{1q}\ b_{2q}\ \ldots\ b_{Nq} \rangle)$, where the first bit b_{1q}=1 and the last bit $b_N q$=0, while the remaining bits position can be arbitrarily filled up with 0 or 1. This is required because of the first bit b_{1i} of the ith $(1 \leq i \leq \lceil \log_2 N \rceil)$ test vector in TS_S is always 0, while the last bit b_{ni} of the ith $(1 \leq i \leq \lceil \log_2 N \rceil)$ test vector is always 1.

Example 2.3.6. For $N = 8$, the test set TS_S will contain four test vectors as: TV_1= $(\langle 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \rangle)$, TV_2=$(\langle 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \rangle)$, TV_3=$(\langle 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \rangle)$, and last test vector TV_4=$(\langle 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0 \rangle)$. The total number of single input stuck-at faults is $|F_S|$=16. It is observed that the fault SA-0-L_1 cannot be detected by any of the test vectors of TS_B. Similarly, the fault SA-1-L_8 cannot be detected by the test vectors of TS_B. So, it is essential to include the test vector TV_4 in the test set TS_S, which can detect both the faults SA-0-L_1 and SA-1-L_8.

The following two results are established for the proposed algorithms for minimality and completeness.

Lemma 2.3.3. *The test set TS_S detects all single input bridging faults and single input stuck-at faults in a given reversible circuit.*

Proof. In Lemma 2.3.1, it is established that all the single input bridging faults are detected by the test set generated by the proposed method. Let us consider the test set TS_S=$\{TV_1, TV_2, \ldots, TV_p, TV_q\}$, where $p = \lceil \log_2 N \rceil$, $q = \lceil \log_2 N \rceil + 1$. It is observed that the test vectors TV_1 to TV_p apply 0 for input line L_1. Similarly, 0 is applied to input line L_2 with respect to test vector TV_1 to TV_{p-1}, and 1 is applied to L_2 by the test vector TV_p. Similarly, a 1 is applied to L_N with respect to test vectors TV_1 to TV_p. Due to this binary encoding pattern, the first line L_1 always gets a 0 as input and the last line L_N gets a 1 as input. Due to this reason, these test vectors are capable of detecting all the single input faults SA-0-L_i and SA-1-L_i except the faults SA-0-L_1 and SA-1-L_N. However, the fault SA-1-L_N cannot be detected by the test set TS_B if $N = 2^x$ for some integer $x \geq 2$. Due to the inclusion of the test vector TV_q in the test set TS_S, it is possible to detect the faults SA-0-L_1 and SA-1-L_N. Hence, we can conclude that TS_S detects all single input stuck-at as well as all single input bridging faults. □

Lemma 2.3.4. *TS_S is a minimal test set for detecting all single input bridging faults and single input stuck-at faults in a given reversible circuit.*

Proof. In Lemma 2.3.2, it is established that all the test vectors of the test set TS_B are essential to detect all single input bridging faults in the circuit. Also it is shown that the test set TS_B is minimal. In Lemma 3, it is proved that the test set TS_S= $\{TV_1, TV_2, \ldots, TV_p, TV_q\}$ is the complete set of test vectors for detecting all single input stuck-at faults. Since TV_1 to TV_p are essential for detecting single input bridging faults, none of these can be removed from TS_S. Since all the single input stuck-at faults cannot be detected by the test set TS_B, so, the requirement of additional test vector is unavoidable. Since only one additional test vector has been added to the test set, so it establishes the minimality of the test set TS_S. Hence, the test set TS_S is a necessary and minimal test set for detecting all the single input bridging faults and single input stuck-at faults for a given reversible circuit. □

Example 2.3.7. Consider the reversible benchmark circuit 3_17tc_tfc as shown in Figure 2.3. The number of single input stuck-at faults in this circuit is $|F_S|$=6. According to the proposed method for generating the test set, the generated test set is TS_S=$\{TV_1, TV_2, TV_3\}$, where TV_1=($\langle 0\ 1\ 0 \rangle$), TV_2=($\langle 0\ 0\ 1 \rangle$) and TV_3=($\langle 1\ 0\ 0 \rangle$). Table 2.3 shows that the test set TS_S can detect all the single input stuck-at faults in the circuit.

Example 2.3.8. Consider the reversible benchmark circuit $4b15g_2$ which is shown in Figure 2.4. The number of single input stuck-at faults of this circuit is $|F_S|$=8. According to the proposed test set generation method, the test set is TS_S=$\{TV_1, TV_2, TV_3\}$, where TV_1=($\langle 0\ 1\ 0\ 1 \rangle$), TV_2=($\langle 0\ 0\ 1\ 1 \rangle$) and TV_3=($\langle 1\ 0\ 1\ 0 \rangle$). Table 2.4 shows that the test set TS_S is capable of detecting all the single input stuck-at faults in the circuit.

Table 2.3: *Stuck-at fault coverage for the circuit* 3_17tc_tfc

Single input stuck-at fault	Test Set (TS_S)			Fault-free Output			Output			Output		
SA-b_i-L_i							SA-0-L_i			SA-1-L_i		
	L_1	L_2	L_3	L_1	L_2	L_3	L_1	L_2	L_3	L_1	L_2	L_3
SA-0-L_1	1	0	0	1	0	0	1	1	1	not required		
SA-1-L_1	0	1	0	0	0	1	not required			1	1	0
SA-0-L_2	0	1	0	0	0	1	1	1	1	not required		
SA-1-L_2	0	0	1	0	0	0	not required			0	1	1
SA-0-L_3	0	0	1	0	0	0	1	1	1	not required		
SA-1-L_3	1	0	0	1	0	0	not required			0	1	0

Table 2.4: *Stuck-at fault coverage for the circuit* $4b15g_2$

Single input stuck-at fault	Test Set (TS_S)				Fault-free Output				Output				Output			
SA-b_i-L_i									SA-0-L_i				SA-1-L_i			
	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4	L_1	L_2	L_3	L_4
SA-0-L_1	1	0	1	0	0	0	0	1	0	1	0	1	not required			
SA-1-L_1	0	1	0	1	1	0	1	0	not required				0	0	1	1
SA-0-L_2	0	1	0	1	1	0	1	0	1	1	0	0	not required			
SA-1-L_2	0	0	1	1	1	1	1	0	not required				1	0	1	1
SA-0-L_3	0	0	1	1	1	1	1	0	1	1	0	0	not required			
SA-1-L_3	0	1	0	1	1	0	1	0	not required				1	0	1	1
SA-0-L_4	0	0	1	1	1	1	1	0	0	1	0	1	not required			
SA-1-L_4	1	0	1	0	0	0	0	1	not required				0	1	1	1

2.4 Experimental results and Discussions

The algorithms for the proposed test generation approach has been implemented and evaluated on several benchmark circuits. The reversible benchmark circuits, based on the NCT and GT gate libraries, have been considered [24]. The experimental results are reported in Table 2.5. The first five columns in Table 2.5 show the name of the benchmark circuit, gate model is used to implement the reversible circuit, the number of input lines/output lines (without garbage lines), the total number of input stuck-at faults, and the number of input bridging faults, respectively. Column 6, 7 and 8 in Table 2.5 represent the number of test vectors required for detecting both the input stuck-at and bridging faults, fault coverage range in percentage, and CPU time for generating the test set, respectively. We have experimented with circuits having up to 40 lines. The number of single input bridging faults in a circuit with 40 lines is more than 20,000, and our proposed approach is able to detect all the faults.

The experimental results are also compared with some of the previous works carried out for input stuck-at and bridging faults [1,2]. The authors in [1] proposed a polynomial-time algorithm to generate the universal test set for detecting the single and multiple input bridging faults and also input stuck-at faults. The comparison of our result with the result of [1] is reported in Table 2.6. Column 6, 7 and 8 of Table 2.6 indicate the number of test vectors generated by [1], the number of test vectors generated by the proposed method and the differences in the number of test vectors generated by the method of [1] in comparison to our proposed method, respectively. The performance analysis of the proposed work and the work of [1] is demonstrated by plotting of the graph as shown in Figure 2.5. Here, the horizontal axis shows the name of the benchmark circuit, and the vertical axis shows the number of test vectors to detect the faults. The solid black line and the solid red line indicate the size of the test set produced by the existing work [1] and the proposed work, respectively. In Figure 2.5, it is observed that the red line is far below the black line, specifically when the circuit size is large in terms of the number of inputs present in the circuit. Consider the circuit mod1048576adder where the total number of input stuck-at faults and bridging faults 80 and 21222, respectively.

Table 2.5: *Detection of input bridging and stuck-at faults (S = Stuck-at faults and B = Bridging faults) with CPU time (sec) for benchmark circuits*

Benchmark Circuit	Gate Model	No. of Input/ Output	No. of input Stuck-at faults	No. of input Bridging faults	No. of Test Vectors	% Fault Coverage	CPU Time (sec)
					B+S	B+S	
ham3	NCT	3/3	6	8	3	100	0.1101
3_17	NCT	3/3	6	8	3	100	0.1134
nth_prime3_inc	NCT	3/3	6	8	3	100	0.1045
4b15g_2	NCT	4/4	8	22	3	100	1.0351
hwb4\design#1	GT	4/4	8	22	3	100	1.0157
hwb4\design#3	NCT	4/4	8	22	3	100	1.0269
2of5	NCT	5/1	10	48	4	100	2.4590
Xor5	NCT	5/1	10	48	4	100	2.2533
mod5\design#1	NCT	5/1	10	48	4	100	2.2782
6sym	NCT	6/1	12	90	4	100	4.1121
hwb6	GT	6/6	12	90	4	100	3.9943
hwb7	GT	7/7	14	152	4	100	6.0333
rd73	NCT	7/3	14	152	4	100	6.1893
ham7	GT	7/7	14	152	4	100	6.1508
rd84	NCT	8/4	16	238	4	100	10.1319
9sym	NCT	9/1	18	352	5	100	11.2843
ham15	GT	15/15	30	1848	5	100	21.0351
mod1024adder	GT	20/20	40	4598	6	100	31.3244
mod1048576 adder	GT	40/40	80	21222	7	100	67.0073

Table 2.6: *Comparison of the test vectors with [1]*

Benchmark Circuit	Gate Model	No. of Input/ Output	No. of input Stuck-at faults	No. of input Bridging faults	No. of Test Vectors [1]	No. of Test Vectors [Proposed]	Difference [1] & [proposed]
					B+S	B+S	B+S
6sym	NCT	6/1	12	90	6	4	2
9sym	NCT	9/1	18	352	9	5	4
hwb7	GT	7/7	14	152	7	4	3
hwb8	GT	8/8	16	238	8	4	4
hwb6	GT	6/6	12	90	6	4	2
rd73	NCT	7/3	14	152	7	4	3
rd84	NCT	8/4	16	238	8	4	4
ham7	GT	7/7	14	152	7	4	3
ham15	GT	15/15	30	1848	15	5	10
mod1024adder	GT	20/20	40	4598	20	6	14
mod1048576 adder	GT	40/40	80	21222	40	7	33

Table 2.7: *Comparison of the test vectors with [2]*

Benchmark Circuit	Gate Model	No. of Input/ Output	No. of input Stuck-at faults	No. of input Bridging faults	No. of Test Vectors [2]	No. of Test Vectors [Proposed]	Difference [2] & [proposed]
					B+S	B+S	B+S
6sym	NCT	6/1	12	90	3	4	1
9sym	NCT	9/1	18	352	5	5	0
hwb7	GT	7/7	14	152	4	4	0
hwb8	GT	8/8	16	238	4	4	0
hwb6	GT	6/6	12	90	3	4	1
rd73	NCT	7/3	14	152	4	4	0
rd84	NCT	8/4	16	238	4	4	0
ham7	GT	7/7	14	152	4	4	0
ham15	GT	15/15	30	1848	8	5	3
mod1024adder	GT	20/20	40	4598	10	6	4
mod1048576 adder	GT	40/40	80	21222	20	7	13

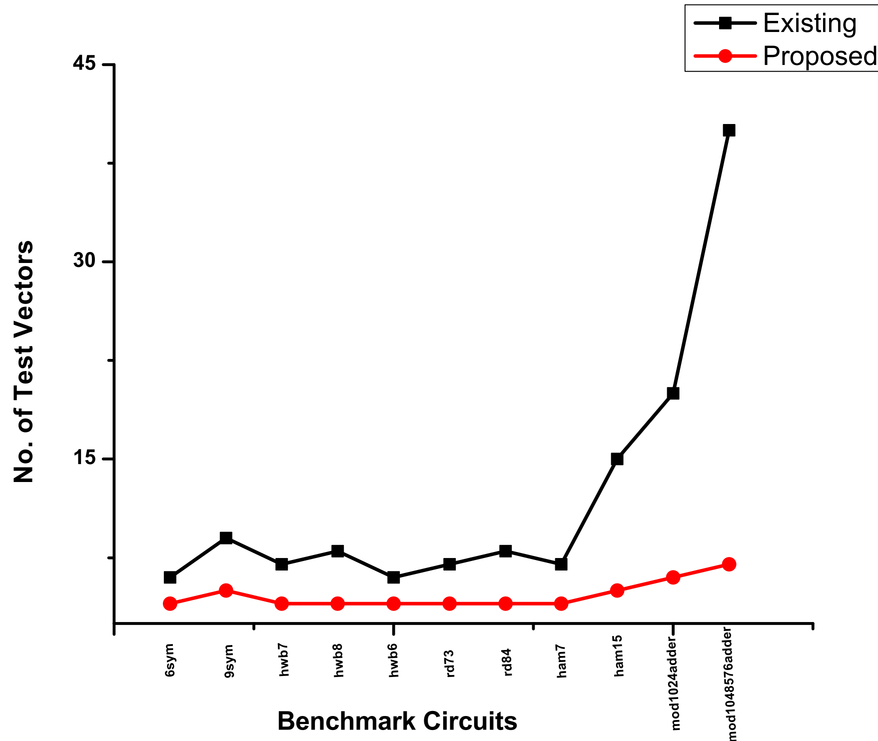


Figure 2.5: *Comparison of the proposed work with the work of [1]*

The proposed method efficiently produces less number of test vectors (7 test vectors) as compared to the number of test vectors (40 test vectors) by the method presented in [1]. Therefore, it is clear that the number of test vectors generated by our proposed method is less in comparison to the number of test vectors generated by the method of [1], and still our proposed method provides 100% fault coverage.

The result of the proposed method is also compared with the work of [2], and the comparison result is shown in Table 2.7 and the column entries are similar to Table 2.6. The authors in [2] proposed the universal test set generation method to detect all single and multiple input bridging faults based on the shift operation on the unitary matrix. The performance analysis of the proposed method in comparison with the method of [2] is illustrated in Figure 2.6. In Figure 2.6, it is observed that the line corresponding to the proposed work (solid red line) and the work of [2] (solid black line) are overlapping for some of the benchmark circuits. It means that the number of test vectors for both

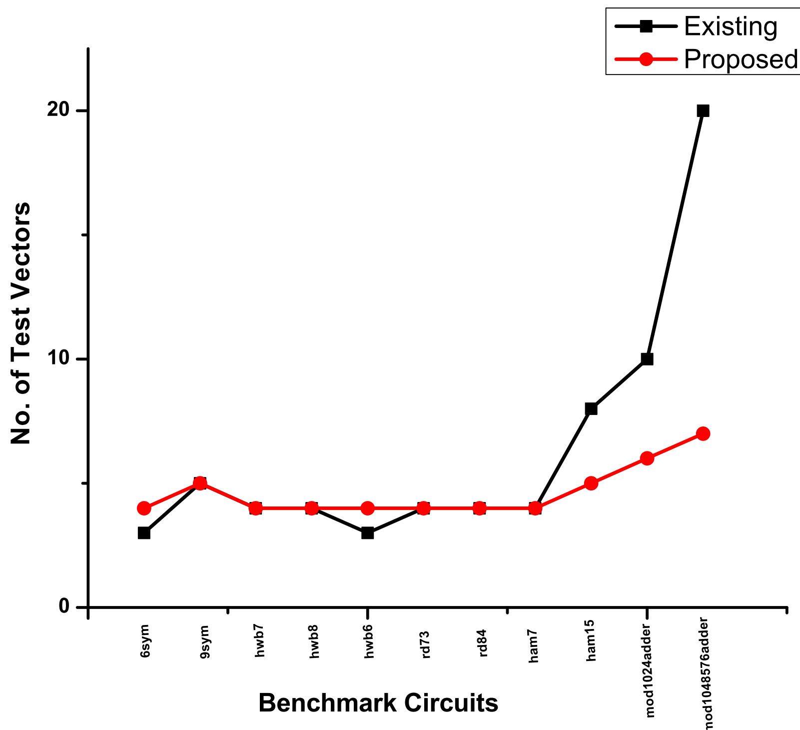


Figure 2.6: *Comparison of the proposed work with the work of [2]*

methods are equal. But, when circuit size is gradually increasing, then the proposed method generates less number of test vectors for detecting the faults as compared to the method of [2], which is shown in Figure 2.6. Finally, based on the reported experimental results, it is evident that the proposed method is better compared to the existing works in terms of number of test vectors for the detection of input stuck-at and bridging faults.

2.5 Conclusion

This chapter discussed a scheme for minimal test set generation to detect single input stuck-at and bridging faults in reversible circuits. It is proved formally that the required number of test vectors is $\lceil \log_2 N \rceil$ for the detection of input bridging faults of a reversible circuit with N inputs. Then it is illustrated that just another additional test vector is sufficient to cover all the input stuck at faults. Experimental results show that the test size of the proposed scheme is much smaller as compared to existing methods, yet

maintaining 100% fault coverage. In the next chapter, we discuss a test set generation method using the path-level expression that can detect the intra-level bridging faults in reversible circuits.

Chapter 3

Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions

3.1 Introduction

In the previous chapter, we proposed a method to generate the test set to detect single input bridging faults and stuck-at faults for reversible circuits. In this chapter, we extend the work to generate the test set to detect multiple input and intra-level bridging faults of reversible circuits. We consider the problem of testing for bridging faults in a reversible circuit designed with NOT, CNOT, Toffoli gates (NCT library) and generalized (n-bit) Toffoli gates (GT library). We propose an Automatic Test Pattern Generation (ATPG) method for generating the minimal complete test set for detecting the Single Input Bridging Faults (SIBF), Multiple Input bridging faults (MIBF), Single Intra-level Bridging Faults (SIRBF) and Multiple Intra-level Bridging Faults (MIRBF). Our proposed ATPG scheme is based on the reversible property– **"**for every vector at the output of a reversible gate, there exists a unique vector at the input.**"** To generate the test set, we introduce the notion of path-level expression and the generated test set is minimal. The proposed method is implemented and experiments are performed on various benchmark circuits with NCT and GT library. The analysis of the experimental results shows that the proposed method has 100% fault coverage, and the test set size is smaller than the existing methods.

In this work, the test patterns generated for single input bridging faults, as mentioned in the previous chapter, is considered as a local test pattern and applied in each level of the reversible circuit. With the help of the local test patterns, a path is generated from each level to the input level of the circuit using the backtracking method. For generating the complete test set for different types of bridging faults, the path-level expression is introduced, which indicates the level-wise information of each path on a given reversible circuit. Finally, generated path-level expressions are applied for the matching process to extract the minimal complete test set for detecting different types of bridging faults in reversible circuits.

The rest of this chapter is organized as follows. The related work on detection of bridging faults in the reversible circuits is given in Section 3.2. In Section 3.3, we describe our proposed method for generating the minimal complete test set to detect the bridging faults with the detailed illustrations. The experimental results and analysis of comparison with existing test pattern generation methods are presented in Section 3.4. Finally, concluding remarks are presented in Section 3.5.

3.2 Related Work

Some of the existing works on testing of Bridging Faults that is briefly reviewed in this section. In 2007, the authors in [4] have proposed a Design-For-Test (DFT) methodology for detecting the single intra-level bridging faults in a reversible circuit with n-bit TOFFOLI gates. The proposed DFT method generates the test set of size $(\lceil \log_2 N \rceil)+3$ and it is sufficient to detect single intra-level bridging and single stuck-at fault. However, this scheme requires an additional input wire to the n-bit TOFFOLI gates (as the DFT circuit.) In 2008, Rahaman et al. [3] showed that a test set of cardinality $(d \log_2 n)$ is sufficient for detecting all intra-level bridging faults in an n-input and n-output reversible circuit with d levels. In 2008, Sarkar et al. [1] presented a polynomial time algorithm, which generates a set of test vectors of size n for detecting all single and multiple input bridging faults and all input stuck-at faults in any n-input and n-output reversible circuit. In 2011, the authors in [2] have proposed a universal test set gener-

ation method of a reversible logic circuit based on the shift operation on the unitary matrix. The test set size by this scheme involves ($\lceil n/2 \rceil$) test vectors, which is sufficient to detect all single and multiple input bridging faults. In 2015, Nagamani et al. [39] proposed a deterministic ATPG algorithm to generate the complete test set for single and multiple intra-level bridging faults in a reversible circuit designed with the family of Toffoli, Peres and Fredkin gates. The complexity of the ATPG algorithm is $O(2^n)$, where n is the number of inputs in the reversible circuit.

In 2010, Chakraborty [52] proposed a design for testability approach for detecting the bridging faults in reversible circuits. The conventional AND-EXOR gates are used to represent the k-CNOT gates, i.e., the k-CNOT based reversible circuits are decomposed in the corresponding irreversible AND-EXOR network. The generated output function of the decomposed circuit resembles Positive Polarity Reed Mullar (PRRM) expressions consists of positive product term $k \geq 1$. This method required $3n + log2p + 2$ number of test vectors for detecting all the intra-level bridging faults, where n is the number of gates, p is the number of input lines for each gate.

From the literature review, we observe that while trying to reduce the number of test vectors for complete bridging fault coverage, there is an additional burden of extra circuitry (i.e, DFT circuitry). The heuristic approaches for ATPG provide the complete test set but, that may not give a minimal solution. Though by applying exact methods, ATPG technique can produce a minimal test set, but computational cost is high.

In this present work, an ATPG algorithm is proposed to generate the minimal complete test set for detecting the bridging faults of types SIBF, SIRBF, MIBF and MIRBF. This scheme does not require any additional hardware for DFT. The computational complexity of the scheme is analyzed and found to be in the logarithmic order in the number of inputs.

3.3 Proposed Method

A reversible circuit structure is a linear cascade structure [26] of reversible gates and bridging faults may occur in the lines (wires) at the levels between gates of the circuit.

Depending on the structure of the reversible circuit, there are several possibilities of intra-level bridging faults in reversible circuits.

1. **Single input bridging Faults:** If the bridging fault occurs between the two input lines at the initial level (input level), then it is called a single Input Bridging Fault (SIBF). The number of single input bridging faults is $C(N, 2)$, where N is the number of input lines in a given reversible circuit. If we consider both AND-Bridging Fault and OR-Bridging Fault then the total number of single input bridging faults is $\{2 \times C(N, 2)\}$. Figure 3.1 (b) shows the single input bridging faults.

2. **Multiple input bridging Faults:** If the bridging fault occurs between more than two input lines at the input level then, it is considered as a Multiple Input Bridging Fault ($MIBF$), which is shown in Figure 3.1 (c). The number of multiple inputs bridging faults is $\{^N C_3 +^N C_4 + \ \ldots +^N C_i + \ \ldots +^N C_N\}$, where $3 < i < N$. The total number of multiple input bridging faults (considering both AND-Bridging Fault and OR-Bridging Fault) is $\{2 \times (^N C_3 +^N C_4 + \ \ldots +^N C_i + \ \ldots +^N C_N)\}$.

3. **Single intra-level bridging Faults:** The Single Intra-Level Bridging Fault ($SIRBF$) occurs when the two shorted lines are at the same level in a given reversible circuit. However, the single input bridging faults are the subset of single intra-level bridging faults. The number of intra-level bridging faults is $|L_S| \times C(N, 2)$, where L_S is the set of levels, excluding the input level. The single intra-level bridging fault is shown in Figure 3.1 (d).

4. **Multiple intra-level bridging Faults:** The Multiple Intra-Level Bridging Fault ($MIRBF$) is represented by more than two lines getting shorted at the same level in the given reversible circuit, which is shown in Figure 3.1 (e). Here, the multiple input bridging faults are the subset of multiple intra-level bridging faults. The number of multiple intra-level bridging faults is $\{|L_S| \times (^N C_3 +^N C_4 + \ \ldots +^N C_i + \ldots +^N C_N)\}$. If we consider both AND-Bridging and OR-Bridging faults, then multiple intra-level bridging faults are just the double.

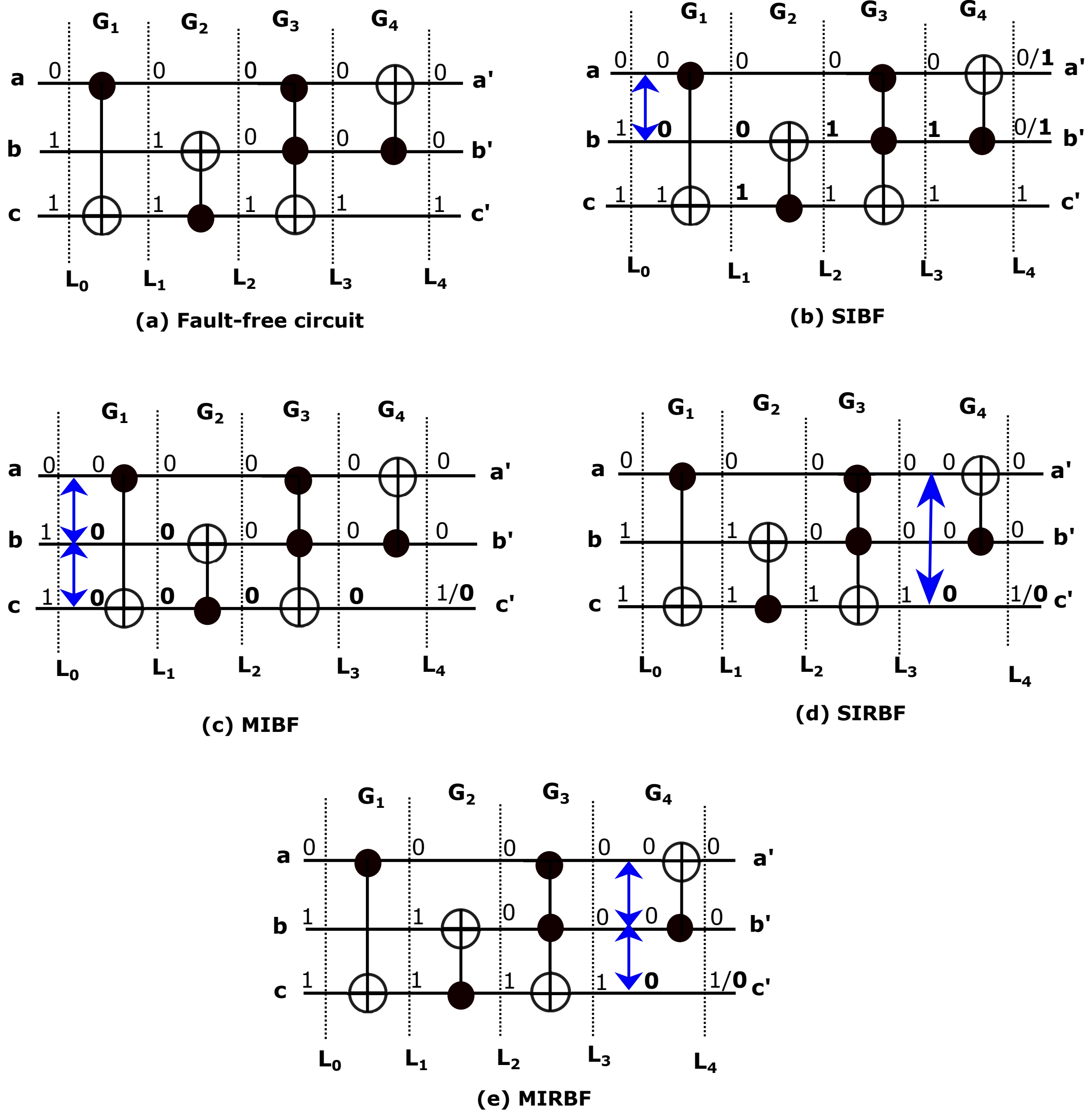


Figure 3.1: *Demonstration of Reversible Circuit* n^{th}*_Prime3_inc for various bridging fault conditions*

Table 3.1: *Truth Table for the n^{th}_Prime3_inc circuit of Figure 3.1 with fault-free and faulty outputs*

Inputs			Fault-Free output			Faulty outputs											
						SIBF			MIBF			SIRBF			MIRBF		
a	b	c	a′	b′	c′	a′	b′	c′	a′	b′	c′	a′	b′	c′	a′	b′	c′
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	1	0	1	1	0	**0**	**0**	**0**	**0**	**0**	**0**	1	1	0	**0**	**0**	**0**
0	1	1	0	0	1	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	0	0	0	1	0	**0**	**0**	**0**	**0**	**0**	**0**	0	1	0	0	1	0
1	0	1	1	0	0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	1	0	1	0	1	1	0	1	**0**	**1**	**0**	1	0	1	**0**	**0**	**0**
1	1	1	0	1	1	0	1	1	0	1	1	**1**	**1**	**0**	**1**	**0**	**0**

We consider the n^{th}_*Prime*3_*inc* reversible benchmark circuit as depicted in Figure 3.1 (a) and show the various bridging faults with the help of AND-Bridging fault model in the subsequent figures of Figure 3.1. Figure 3.1 (b) shows that an input line '*a*' is shorted with the input line '*b*' at the input level L_0. The effect of this fault is represented by the truth table as shown in Table 3.1. We have extracted the test vectors by comparing the fault-free output with the faulty output. Therefore, the test vector 010 or 011 or 100 or 101 is required to test this fault. Similarly, for multiple input bridging fault, the input lines '*a*', '*b*' and '*c*' are shorted with each other at the input level L_0. As shown in Table 3.1, the possible test vectors are 001, 010, 011, 100, 101, and 110 and any one of these test vectors is capable of detecting the multiple input bridging fault as shown in Figure 3.1 (c).

For single intra-level bridging faults, the input line '*a*' is shorted with the input line '*c*' at the 3rd level L_3 as depicted in Figure 3.1 (d). As mentioned in Table 3.1, the test vectors 001 or 011 or 101 or 111 is capable of detecting the single intra-level bridging faults, which is shown in Figure 3.1 (d). Figure 3.1 (e) shows the multiple intra-level bridging faults. For detecting these faults, the test vector 001 or 010 or 011 or 101 or

110 or 111 is needed.

In this section, we first present a method to find the minimum set of test vectors to detect all the single input bridging faults at the initial level in a given reversible circuit. After that, this test set is applied as local test patterns to all the levels (excluding the initial level) of a given reversible circuit. Next, we generate the paths between the levels of the circuit using a local test pattern with the help of backtracking. After generating all the possible paths, some paths are selected by the path-level expression. Here, the path-level expression is capable of producing a unique path for each level in the reversible circuit. Finally, generated path-level expressions are matched to obtain the minimal complete test set for detecting all the possible bridging faults in the reversible circuit. The detailed discussion of the proposed method is given in the following subsections. Before discussing the proposed method, we present some definitions, which are used in subsequent discussions.

Definition 3.3.1. *A test vector TV is a set of binary inputs $\langle b_1\ b_2\ \dots\ b_N \rangle$ that provides test inputs to the input lines $\{l_1, l_2,\ \dots l_N\}$ for$1 \leq i \leq N$, where $b_i \in \{0, 1\}$ and N is the number of input lines. The test vector (TV) is applied to a reversible circuit for testing.*

Definition 3.3.2. *A test set TS is the collection of all possible test vectors that detect the faults in F, where F is the fault set in a given reversible circuit. Let the test set TS $= \{TV_1,\ TV_2...,\ TV_l\}$, for $1 \leq j \leq l$, $TV_j = \langle b_{1j}\ b_{2j}\ \dots\ b_{Nj} \rangle$, where b_{ij} is the ith bit of jth test vector. In the binary inputs for the test vector TV_j, at least one-bit b_{ij} must be different from some other bit b_{kj} to detect the bridging fault between line i and k, where $i \neq k$. Let the test set TS_{L_i} be the set of all local test patterns that are applied at level L_i during test pattern generation.*

Definition 3.3.3. *A test set TS is called a complete test set in n-input reversible circuit that detects all the faults in F. A complete test set that contains the minimum possible test vectors, which are capable of detecting all the faults in F, is called a minimal complete test set.*

3.3.1 Local Test Pattern Generation Method

If a test is capable of applying opposite logic values to every pair of lines lying at the same level, it is capable of detecting all the bridging faults in that level. Based on this concept, we generate the minimal complete test set at the initial level and later on, these test vectors of the minimal test set are considered as local test patterns, which are applied to the other levels of the reversible circuit. The basic idea for generating the

local test patterns for detecting the single input bridging faults in the reversible circuits is explained below.

TV$_1$ = 0 1 0 1 0 1.......... N-bits
2^0 consecutive 0's followed by 2^0 consecutive 1's

TV$_2$ = 0 0 1 1 0 0............ N-bits
2^1 consecutive 0's followed by 2^1 consecutive 1's

⋮

TV$_n$ = 0 0 0 0 1 1 1 1 .... N-bits
2^{n-1} consecutive 0's followed by 2^{n-1} consecutive 1's

⋮

TV$_{\lceil log_2 N \rceil}$ = 0 0 0....0 1 1 1....1.......... N-bits
$2^{\lceil log_2 N \rceil - 1}$ consecutive 0's followed by $2^{\lceil log_2 N \rceil - 1}$ consecutive 1's

Figure 3.2: *Test set generation for detecting input bridging faults*

Here, we assume that the test set is denoted by TS and the test vector by TV_i, where $i \in \mathbb{N}$. Let us consider the test set $TS=\{TV_1, TV_2, \ldots, TV_{\lceil log_2 N \rceil}\}$, where N is the number of input lines in a given reversible circuit. Let the first test vector TV_1 consist of alternating 0's and 1's. The second test vector TV_2 consists of an alternating sequence of two consecutive 0's and two consecutive 1's. Similarly, the nth test vector consists of 2^{n-1} consecutive 0's followed by 2^{n-1} consecutive 1's at the N-input lines, where $1 \leq n \leq \lceil log_2 N \rceil$. This process continues until we get the last test vector $TV_{\lceil log_2 N \rceil}$. Figure 3.2 illustrates the above process.

Example 3.3.1. : If we consider N=5, the test set TS consists of three test vectors, viz., $TV_1=\langle 0\ 1\ 0\ 1\ 0 \rangle$, $TV_2=\langle 0\ 0\ 1\ 1\ 0 \rangle$, and $TV_3=\langle 0\ 0\ 0\ 0\ 1 \rangle$. We observe that due to the presence of opposite logic values of each test vector, the test set TS is capable of detecting all the bridging faults at the initial level (L_0) in a reversible circuit.

3.3.2 Path Generation Method

After generating the set TS, it is applied as a local test pattern to all the levels in a reversible circuit. A reversible circuit with N gates has (N+1) levels. Input to the circuit is termed as level L_0, and the final output of the circuit is termed as level L_N. Level L_i lies between the gates G_i and G_{i+1}. Faults basically occur in the signal wire

at any level and gates are assumed to be fault free. So the distinction of level from the gate is necessary. Some definitions are presented, which are used in the path generation method.

Definition 3.3.4. *A level set L_S is a set of all the levels in a reversible circuit to maintain the linear cascade structure. Let the level set be L_S=$\{L_0, L_1, L_2, \ldots, L_{GC}\}$, where each gate G_i lies between $(i-1)^{th}$ and i^{th} level, for $1 \leq i \leq GC$. $G_i \in \{NCT$ or GT library$\}$ and GC is the total number of gates present in the reversible circuit.*

Definition 3.3.5. *We define a path P_k, which is generated by the interaction of levels (L_j, L_{j-1}), where $k \in \mathbb{N}$. The path P_k is derived using the concept of backtracking. The backtracking is applied on local test pattern from level L_j to L_{j-1}. In other words, P_k=$\{\langle TV_i(L_j), TV_{i-1}(L_{j-1}\ \rangle\} \mid TV_i \in TS_{L_j}$ and $TV_{i-1} \in TS_{L_{j-1}}\}$. $TV_i(L_j)$ means test vector $TV_i \in TS_{L_j}$ is applied at level L_j. Here, the total number of P_k is equal to the total number of test vectors present at the input level L_0.*

The path P_k is generated between the two contiguous levels of the circuit. Initially, we apply all the local test patterns $TV_i \in TS$ in each level (excluding the initial level L_0) of the circuit. Let gate G_j lies between $L_{(j-1)}$ and L_j level and the local test pattern TV_i is applied at level L_j that contains the gate G_j. We back propagate (say also backtracking) TV_i to obtain the corresponding input test vector $TV_j \in TS_{L_j}$ at the input of the gate G_j, i.e., TV_j is generated at level L_{j-1}. Since the reversible gate is bijective, for a given TV_i, the corresponding TV_j is unique. Here, the path P_k is created between the two levels L_j and L_{j-1} through the test vector TV_i and TV_j, respectively.

Definition 3.3.6. *The path is said to be a complete path in a reversible circuit if $TV_i(L_{GC})$ is capable of tracing the $TV_j(L_0)$. The path may start at any level L_p, where $0 < p < GC$ and formulate this path to be complete when $TV_i(L_p)$ traces the $TV_j(L_{GC})$ using the forward simulation.*

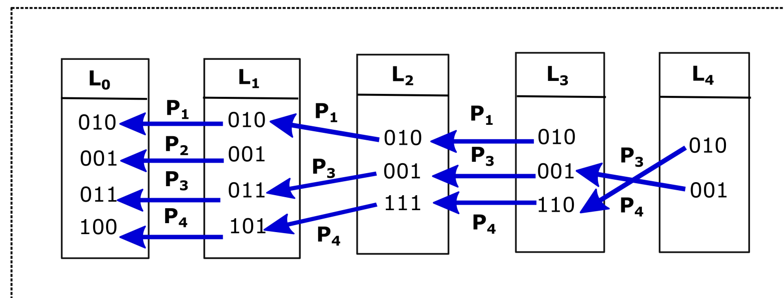


Figure 3.3: *Illustration of path generation for the circuit n^{th}_Prime3_inc using backtracking*

The detail illustration of the above definitions is as represented in Figure 3.3. Here, the set of levels $L_S = \{L_1, L_2, L_3, L_4\}$ (excluding the initial level L_0) and the total number of gates $GC = 4$ as per Definition 3.3.4. Let us consider the gate G_3, which lies between level L_2 and L_3. According to Definition 3.3.5, if we consider the path P_1 at level L_3, then this path is derived from level L_3 to L_2 by applying the local test pattern '010' and the derived path P_1 generates the test vector '010' at level L_2. Using the backtracking, the path P_1 at level L_3 generates the test vector '010' at the input level L_0. The derived paths are P_1, P_2, P_3 and P_4. Therefore, four test vectors $\{010, 001, 011, 100\}$ are generated at the initial level L_0 as shown in Figure 3.3. In this case, the path P_1 is not derived from the primary output level L_4, so it is not the complete path. For the path P_1 to be complete, we use the forward simulation from level L_3 to L_4. As mentioned in Definition 3.3.6, the paths P_3 and P_4 are complete.

Now, we present the algorithm for the path generation method in Algorithm 1.

Example 3.3.2. : We illustrate the path generation algorithm with an example in Figure 3.3. At first, we extract the required parameters L_S=$\{L_1, L_2, L_3, L_4\}$, TS=$\{010, 001\}$ and consider a fault-free truth table as $FaultFree_Table$ for the $n^{th}_Prime3_inc$ reversible circuit as shown in Table 3.1. As mentioned in the algorithm, we initialize the variable L_0 and P_k to empty values. Here, P_k provides the information of tracing a path between level L_4 and L_0, where L_0 is the input level of a given circuit and stores the test vectors which are generated by the derived path P_k. Initially, L_j=L_1 and L_{j-1}=L_0 and the variable j is incremented to 4 (i.e., GC=4). We consider the test vector $\langle$ 0 1 0 $\rangle$ in TS and apply it to the first level L_1. Using the backtracking method (called as $BackTracing$), the test vector $\langle$ 0 1 0 $\rangle$ at level L_1 extracts the test vector $\langle$ 0 1 0 $\rangle$ at level L_0 with the help of $FaultFree_Table$. Therefore, level L_0 is updated with the newly assigned test vector (010), which is generated by the path P_1 and it is derived from L_1 to L_0. Again, we consider the next test vector $\langle$ 0 0 1 $\rangle$ in TS and apply it to the first level L_1. Using the $BackTracing$ function, the test vector $\langle$ 0 0 1 $\rangle$ extracts the same test vector $\langle$ 0 0 1 $\rangle$ at level L_0 and produces the new path P_2, which is also derived from level L_1 to L_0. The next iteration is level L_2 and the test vector $\langle$ 0 1 0 $\rangle$ extracts the test vector $\langle$ 0 1 0 $\rangle$ at level L_1. In this case, the same path P_1 is considered from level L_2 to L_1 due to the same test vector $\langle$ 0 1 0 $\rangle$ being available in level L_1. There is no need to update the input level L_0 if the same path exists in the previous iteration. But, when we apply the test vector $\langle$ 0 0 1 $\rangle$ at level L_2, then new test vector $\langle$ 0 1 1 $\rangle$ is generated at level L_1 with the help of $FaultFree_Table$. Thus, the new path P_3 is created, which is derived from L_2 to L_0 and we update the input level L_0 with the test vector $\langle$ 0 1 1 $\rangle$. It is observed that the level L_0 is only updated when a new path is formed, otherwise we create the path between the current pair of levels. The same process is continued up to level L_4. In Figure 3.3, we have observed that four test vectors viz. $\langle$ 0 1 0 $\rangle$, $\langle$ 0 0 1 $\rangle$, $\langle$ 0 1 1 $\rangle$ and $\langle$ 1 0 0 $\rangle$ are available at the input level

```
Algorithm 1: Path generation algorithm
   Input: A level set L_S={L_1, L_2, ..., L_GC} of levels, set "TS" and
          "FaultFree_Table" in a reversible circuit
   Output: P_k is the generated path for each level present in L_S
 1 L_0 ← ∅
 2 P_k ← ∅
 3 for j ← 1 to GC do
 4     Applied TV_i ∈ TS to L_j
 5     BackTracing (TV_i, L_j, L_{j-1})
 6     Update the level L_0
 7     P_k ← Path from L_j to L_0
 8 return P_k
 9 Function BackTracing (TS, L_j, L_{j-1})
10     FaultFree_Table ← level-wise Truth table for fault-free circuit
11     for each level L_j ∈ L_S do
12         for each test vector TV_i(L_j) do
13             TV_k(L_{j-1}) extracts from TV_i(L_j) using FaultFree_Table
14             if TV_k(L_{j-1}) ≠ TV_i(L_j) then
15                 return BackTracing (TV_i, L_j, L_0)
16             else
17                 return BackTracing (TV_i, L_j, L_{j-1})
18             ConnectLevel (TV_i(L_{j-1}), TV_i(L_j))
19 Function ConnectLevel (TV_1, TV_2)
20     TV_1 ← test vector of L_{j-1}
21     TV_2 ← test vector of L_j
22     TV_1 ← TV_2 and TV_2 ← TV_1
23     Update the level L_{j-1} with newly assigned TV_1
```

L_0 and these test vectors are generated by the paths P_1, P_2, P_3, and P_4, respectively. Moreover, the test vector $\langle\ 0\ 1\ 0\ \rangle$ and $\langle\ 0\ 0\ 1\ \rangle$ at level L_4 produce the complete paths P_3 and P_4, which are shown in Figure 3.3.

3.3.3 Complete Test Set Generation Method

In this section, the proposed method is described to generate a complete test set for detecting all bridging faults in a given reversible circuit. The path generation method provides the level-wise interaction with the help of backtracking, and all the paths that are associated with the test vectors at their respective levels. The path-level expression is introduced for selecting a path P_k such that it covers the maximum number of bridging faults.

The validity of a path P_k in between the levels implies that there must exist derivation paths in a reversible circuit from the initial level L_0 to the last level L_{GC} and vice versa. Derivation paths can be expressed as an expression, termed as a path-level expression. A path-level expression consists of test vectors and metacharacters are used to describe some of the characteristics. Table 3.2 shows the metacharacters considered in this work.

Table 3.2: *Meta Characters and Interpretation*

Meta Character	Meaning
$P_k[\ ]$	Set of test vectors, characters and metacharacters used in path Pk
$\{n\}$	Repetition of n times
$\{L_j\}$	Specify the particular level L_j
$(L_k - L_j)$	Specify the level L_k to L_j
.	Specify the connection
+	Proceed to previous level

Definition 3.3.7. *The path level expression P_{KE} related to path P_k is defined as $P_k \sum_{i=GC}^{0} [TV_i(L_i).\{1\} \vee TV_j(L_i – L_j).\{|i - j + 1|\}]$, where $j < i$, $TV_j(L_i – L_j)$ indicates the presence of same test vector from level L_i to level L_j.*

The path-level expressions are used for selecting the proper paths such that selected paths provide the complete test set for detecting all the bridging faults. Let us consider the complete paths generated by the path-level expression P_{lE} and P_{mE}, which are described as follows:

P_{lE}: $P_l\ [TV_i(L_{GC} – L_k).\{|GC - k + 1|\} + TV_j(L_{k-1} – L_0).\{|k|\}]$

P_{mE}: $P_m\ [TV_j(L_{GC} – L_{k+1}).\{|GC - k|\} + \overline{TV_i}(L_k).\{1\} + TV_i(L_{k-1} – L_0).\{|k|\}]$

Consider two complete paths from the level L_{GC}, which are defined by the path-level

expression P_{lE} and P_{mE}. These two path-level expressions are matched to validate the derived path.

The path-level expression stores the unique test vector along with the corresponding level and repetition of the test vector to the previous levels. The path-level expressions P_{lE} and P_{mE} have $GC+1$ levels, where L_{GC} is the primary output level, and L_0 is the input level. The path-level expression P_{lE} indicates that the test vector TV_i is present from L_{GC} to L_k and the number of occurrences of TV_i is $|GC-k+1|$. Also, the test vector TV_j is associated with level L_{k-1} to L_0 and number of occurrence is $|k|$. Similarly, in path-level expression P_{mE}, the test vector TV_j is associated with levels L_{GC} to L_{k+1}, the test vector $\overline{TV_i}$ is associated with level L_k and the test vector TV_i is associated with level L_{k-1} to L_0. The occurrences of the test vectors TV_j, $\overline{TV_i}$ and TV_i are $|GC-k|$, 1 and $|k|$, respectively. If the path-level expression P_{lE} is matched with P_{mE}, it is observed that the test vector TV_i is the complement form at level L_k. To cover all the faults at level L_k, we need another path-level expression which is generated from the same level L_k. Suppose, the new form of path-level expression P_{PE} is derived from level L_k, which is expressed as:

$$P_{PE}\text{: } P_P\ [TV_j(L_k\text{–}L_1).\{|k|\} + \overline{TV_i}(L_0).\{1\}]$$

A complete path is required to cover all the levels of a given reversible circuit according to Property 1 and Property 2. The path-level expression P_{PE} does not generate a complete path because the process starts from level L_k. For extracting the complete path from the path-level expression P_{PE}, the test vector $TV_j(L_k)$ traces the test vector of primary output level L_{GC} with forward simulation. The following path-level expression is generated from the path-level expression P_{PE} which eventually gives a complete path.

$$P'_{PE}\text{: } P'_P\ [\overline{TV_i}(L_{GC}).\{1\} + TV_k(L_{GC-1}\text{–}L_{k+1}).\{|GC-k-1|\} + TV_j(L_k\text{–}L_1).\{|k|\} + \overline{TV_i}(L_0).\{1\}]$$

This path-level expression P'_{PE} is matched with the path-level expression P_{lE} and

P_{mE}. The path-level expression P'_{PE} assures that the test vector TV_j is present at level L_k. If we consider the path-level expression P_{mE} and P'_{PE}, then generated test vectors TV_i and $\overline{TV_i}$ fail to detect the faults at the initial level L_0. Similarly, the path-level expressions P_{lE} and P'_{PE} generate the test vector TV_i and $\overline{TV_i}$, respectively, which are unable to detect the faults at level L_{GC}. Therefore, for detecting all the faults at each level along with satisfying Property 3.3.1 and Property 3.3.2, all the three path-level expressions are needed. The test vectors TV_j, TV_i, and $\overline{TV_i}$ are produced by the path-level expressions P_{lE}, P_{mE} and P'_{PE}, respectively at the input level L_0.

Selection and matching of the path-level expressions are based on the following properties:

Property 3.3.1. Each level present in the level set L_S produces the unique test vectors which are generated by the derivation path. As a result, the same test vector cannot be present at the same level.

Proof. According to the controllability property of reversibility, any test vector of a particular level generates the unique test vector at the previous level using backtracking. The derivation path for each level is the interaction between the two test vectors of their corresponding levels. Therefore, each derivation path generates a unique test vector at a given level. Hence, the same test vector cannot occur at the same level, which is generated by the derivation path. □

Property 3.3.2. The generated test vectors for each level must be capable of producing opposite logic values to capture all possible bridging faults. So, some test vectors must exist to represent all the bridging faults in each level.

Proof. The local test patterns are applied for each level in L_S for extracting the test vector at the previous level using backtracking. These local test patterns are capable for producing the opposite logic values for each level as depicted in Figure 3.2. Furthermore, the derived path is created based on these local test patterns which are applied for each individual level in a given reversible circuit. Therefore, each level contains the local test patterns along with the test vectors which are generated by the derived paths. Hence, there exist some test vectors in each level which are capable of detecting all the bridging faults in a given reversible circuit. □

Example 3.3.3. The complete test generation process is illustrated with an example. Consider the reversible benchmark circuit $n^{th}_Prime3_inc$ as shown in Figure 3.4.

Consider the complete paths which occur at level L_4. The following path-level expressions express the complete path.

P_{4E}: $P_4[010.(L_4).\{1\} + 110.(L_3).\{1\} + 111.(L_2).\{1\} + 101.(L_1).\{1\} + 100.(L_0).\{1\}]$

P_{3E}: $P_3[001.(L_4–L_2).\{3\} + 011.(L_1–L_0).\{2\}]$

After matching the two expressions, it is observed that test vectors 001 and 110 are complement to each other at level L_3 and as a result it is not possible to detect the bridging faults at level L_3. According to the proposed method, the other path at level

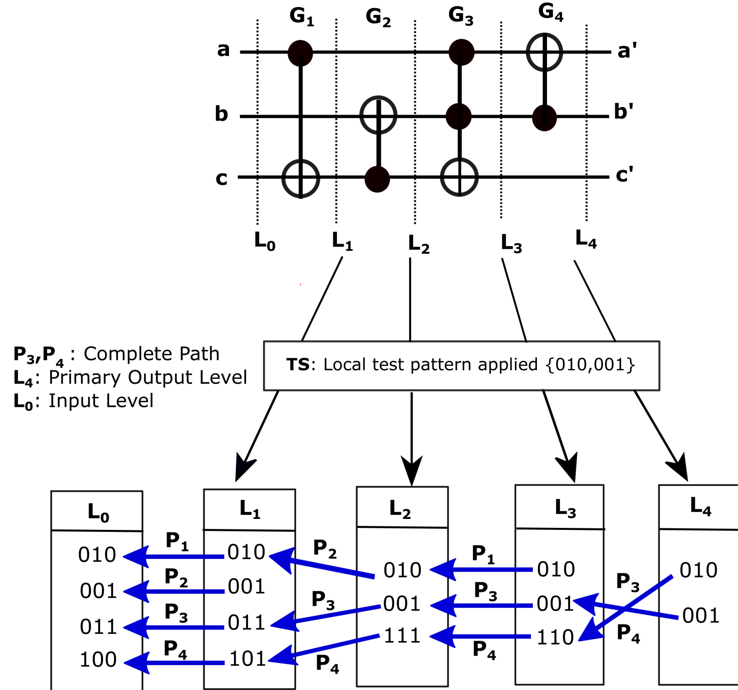


Figure 3.4: *Demonstration of Complete Test Set generation for the circuit n^{th}_Prime3_inc*

L_3 is considered as shown in Figure 3.4. The new path-level expression is

P_{1E}: $P_1[010.(L_3–L_0).\{4\}]$

The backtracking process of the path-level expression P_{1E} starts at level L_3. Therefore, the generated path by the path-level expression P_{1E} is not complete. For constructing the complete path from the path-level expression P_{1E}, the method of the forward simulation at level L_3 is used. The complete path generated by the path-level expression P'_{1E} is

P'_{1E}: $P'_1[110.(L_4).\{1\} + 010.(L_3–L_0).\{4\}]$

After matching the path-level expressions P_{3E} and P'_{1E}, it is observed that the generated test vector 001 and 110 are not capable of detecting the bridging faults at level L_4 due to complement form of each other. For detecting the bridging faults at level L_4, we need the path-level expressions P_{4E} and P_{3E}. If we consider the path-level expressions P_{4E} and P'_{1E} then the generated test vectors 010 and 111 are not able to detect the faults at level L_2, because the test vector 111 does not consist of opposite logic values. So, it is essential to consider all the path-level expressions P_{4E}, P_{3E} and P'_{1E} to generate the test vectors $100, 011$ and 010 at L_0, respectively. Moreover, by matching these path expressions Property 3.3.1 and Property 3.3.2 are satisfied. Hence, the test set $\{100, 011, 010\}$ is the complete test set to detect all the bridging faults at each level in the n^{th}_Prime3_inc reversible circuit.

As explained in Example 3.3.3, three test vectors are sufficient for detecting all the

Table 3.3: *Fault coverage table for the circuit n^{th}_Prime3_inc with 2 test vectors*

Test set size=2	Faults covered	% faults coverage
(000, 001)	08	53.33
(000, 010)	10	66.66
(000, 011)	10	66.66
(000, 100)	08	53.33
(000, 101)	10	66.66
(000, 110)	08	53.33
(000, 111)	06	40.00
(001, 010)	14	93.33
(001, 011)	14	93.33
(001, 100)	13	86.66
(001, 101)	13	86.66
(001, 110)	12	80.00
(001, 111)	12	80.00
(010, 011)	14	93.33
(010, 100)	13	86.66
(010, 101)	14	93.33
(010, 110)	12	80.00
(010, 111)	13	86.66
(011, 100)	13	86.66
(011, 101)	14	93.33
(011, 110)	14	93.33
(011, 111)	12	80.00
(100, 101)	14	93.33
(100, 110)	12	80.00
(100, 111)	12	80.00
(101, 110)	14	93.33
(101, 111)	12	80.00
(110, 111)	12	80.00

bridging faults in $n^{th}_Prime3_inc$ benchmark circuit according to the proposed path-level expression method. As an empirical study, we consider all the possible test sets with size 2 and apply to the circuit. The total number of single bridging faults for $n^{th}_Prime3_inc$ benchmark circuit is 15. The fault coverage for all possible test sets with two test vectors is reported in Table 3.3. It is observed that two test vectors are not sufficient to detect all the possible bridging faults. As per the path-level expression method, 3 test vectors are generated which are sufficient to detect all the single bridging faults in $n^{th}_Prime3_inc$ reversible circuit.

Lemma 3.3.1. *Proposed path-level expression method generates the minimal complete test set for detecting the bridging faults with 100% fault coverage in a given NCT or GT based reversible circuit.*

Proof. Let us assume that the path-level expression P_{lE} and P_{mE} are capable of generating the complete minimal test set at the initial level L_0 in a given reversible circuit. Hence, as per our proposed method, these expressions generate the complete path from level L_{GC} and are also able to satisfy Property 3.3.1 and Property 3.3.2. For proving this lemma, a counter example is considered. Let us consider that the path-level expressions P_{lE} and P_{mE} are not capable of generating the complete minimal test set. Then, we have to choose another path, which is expressed by the path-level expression P_{PE} at level L_{GC-1} to the initial level L_0. Therefore, path-level expression P_{PE} is unable to generate the complete path. To construct the complete path, we have derived the path-level expression P_{PE} from level L_{GC-1} to level L_{GC} using the forward simulation. The newly derived path-level expression is named as P'_{PE}. Now, we have three path-level expressions as P_{lE}, P_{mE}, and P'_{PE}. Each pair of path-level expressions are matched to validate the derived path. The matching of the path-level expressions appears based on the occurrence of the generated path. Suppose we consider that all the pairs of path-level expressions satisfy both the properties and, each path-level expression is capable of generating the complete path. It means that any pair of path-level expressions is capable of generating the complete test set at the initial level L_0. However, according to the proposed method, path-level expressions P_{lE} and P_{mE} are matched before the matching of the path-level expressions P_{lE} and P'_{PE} or P_{mE} and P'_{PE}. Hence, the path-level expressions P_{lE} and P_{mE} are capable of generating the complete minimal test set. So, there is a contradiction. □

3.3.4 Complexity of the Proposed Method

Consider an N-input reversible circuit with a total number of levels $L + 1$. According to the local test pattern generation method explained, we need $O(\lceil \log_2 N \rceil)$ number of test vectors at the initial level. Therefore, the total number of test vectors required for $L + 1$ levels is $O((L + 1)\lceil \log_2 N \rceil)$. According to the proposed method, the maximum number of paths that is generated using backtracking is $\lceil \log_2 N \rceil$ and the complexity

for generating these paths is $O(\lceil\log_2 N\rceil(L+1)\lceil\log_2 N\rceil)$. To evaluate the test vectors at each level, we generates the path expression for $(L+1)$ levels. For constructing the path expression, the time complexity is $O(L+1)$ because each path expression directly depends on the levels in the path. Hence, the total time complexity of our proposed method is $O((L+1)^2(\lceil\log_2 N\rceil)^2)$.

3.4 Experimental Results and Discussions

The algorithm of the proposed complete test set generation method has been implemented and applied to various benchmark circuits based on NCT and GT gate libraries [24] and [25]. A tool is implemented to generate the complete test set for fault models such as Stuck-at fault model, Bridging fault model, Missing gate fault model and Crosspoint fault model. The experimental result in Table 3.4 shows the total number of test vectors that is required for the detection of bridging faults ($SIBF$, $MIBF$, $SIRBF$, and $MIRBF$) with the CPU simulation time in seconds. The first five columns in Table 3.4 provide the benchmark circuit name, number of inputs: N + C (inputs + constant inputs), number of outputs: m + g (outputs + garbage outputs), number of gates, and the total number of bridging faults, respectively. Column 6 and 7 in Table 3.4 present the total number of test vectors that are required for detecting all possible bridging faults and the CPU simulation time, respectively.

Based on the experimental results as shown in Table 3.4, the analysis is provided for three parameter such as (i) CPU time vs. Input lines, (ii) CPU time vs. Gate count and (iii) CPU time vs. Number of faults. It is observed that the proposed method can handle reasonably large circuits.

(i) **CPU time vs. Input lines:** In Figure 3.5, it is observed that the variation in CPU time against the input lines is similar for some of the benchmark circuits, which is noticed by the persistent straight line in the initial part of the graph. For the circuits considered in this part of the graph, the number of gates and the number of total faults in those circuits are less and so the CPU time taken is also significantly low. Consider the circuits $hwb8_614$, $hwb9_1544$, and $rd73d2$

Table 3.4: *Complete test set for detection of bridging faults with simulation CPU time (sec) for the benchmark circuits*

Benchmark Circuit	No. of Inputs	No. of Outputs	No. of Gates	No. of Total Detectable Faults	No. of Test Vectors	CPU Time (sec)
	n + C	m + g		SIBF+MIBF+SIRBF+MIRBF		
Peres_9	3 + 0	3 + 0	2	24	2	0.034
Fredkin_6	3 + 0	3 + 0	3	32	2	0.166
nth_prime3	3 + 0	3 + 0	4	40	3	0.203
Miller_11	3 + 0	3 + 0	5	48	2	0.213
ham3d1	3 + 0	3 + 0	5	48	2	0.234
3_17_13	3 + 0	3 + 0	6	56	2	0.257
3_17_14	3 + 0	3 + 0	6	56	3	0.283
Toffoli_double_4	4 + 0	4 + 0	2	66	2	0.302
rd32d1	3 + 1	2 + 2	4	110	3	1.223
mini-alu_167	4 + 0	2 + 2	6	154	2	2.167
decode24_v0_38	2 + 2	4 + 0	6	154	4	3.834
mod10_171	4 + 0	4 + 0	10	242	4	4.157
hwb4-11-23	4 + 0	4 + 0	11	264	4	4.333
mspk_hwb4_12	4 + 0	4 + 0	12	286	4	4.712
4_49d3	4 + 0	4 + 0	12	286	4	4.017
mspk__4_49_13	4 + 0	4 + 0	13	308	4	3.477
mspk_4b15g_1	4 + 0	4 + 0	15	352	4	5.197
4_49d1	4 + 0	4 + 0	16	374	4	5.364
hwb4d1	4 + 0	4 + 0	17	396	3	5.002
4gt11_84	4 + 1	1 + 4	3	208	3	2.839
4gt11-v1_85	4 + 1	1 + 4	4	260	4	4.156
xor5d1	5 + 0	1 + 4	4	260	3	3.821
mod5d4	4 + 1	1 + 4	5	312	3	4.528
alu-v0_26	5 + 0	4 + 1	6	364	4	6.001
mod5d1_63	5 + 0	5 + 0	7	416	4	6.226
4mod7-v1_96	4 + 1	3 + 2	7	416	4	6.283
mod5d1	4 + 1	1 + 4	8	468	3	6.828
mod5d2	4 + 1	1 + 4	9	520	4	7.139
hwb5d1	5 + 0	5 + 0	55	2912	5	22.410
graycode6_47	6 + 0	6 + 0	5	684	4	6.936
ex3_229	5 + 1	6 + 0	7	912	4	8.036
mod5adder_128	6 + 0	6 + 0	15	1824	4	16.822
hwb6d3	6 + 0	6 + 0	42	4902	4	28.023
hw6d1	6 + 0	6 + 0	126	14478	7	61.044
ham7d1	7 + 0	7 + 0	23	5760	4	66.331
hwb7d1	7 + 0	7 + 0	289	69600	3	43.871
hwb8_614	8 + 0	8 + 0	614	303810	7	403.412
hwb9_1544	9 + 0	9 + 0	1544	780610	7	867.318
rd73d2	7 + 3	7 + 3	20	42546	4	215.971
6symd2	6 + 4	1 + 9	20	42546	3	286.909
hwb10-3631	10 + 0	10 + 0	3631	7358432	7	1197.964
hwb11-9314	11 + 0	11 + 0	9314	37930680	7	2960.651
cycle10_2d1	12 + 0	12 + 0	19	163320	5	128.948
9symd2	9 + 3	1 + 11	28	236814	5	717.283
rd84d1	8 + 7	4 + 11	28	1899616	4	2351.187
ham15d1	15 + 0	15 + 0	132	8712032	8	4547.57
cycle17_3d1	20 + 0	20 + 0	48	102758390	8	7002.426
mod1024adder1	20 + 0	20 + 0	55	117438160	7	9161.228

with number of input lines 8, 9 and 10, respectively. It is observed that the circuit $hwb9_1544$ with 9-input lines takes significantly higher time ($867.318 secs$) to generate the test vectors with reference to the circuits $hwb8_614$ ($403.412 secs$) and $rd73d2$ ($215.971 secs$). For the circuit $hwb9_1544$, the number of gates and a number of faults are relatively large, and so the time is taken to generate the test vectors is also more. Similar relationships between the number of gates and the number of faults of the circuits are observed where there is a transient response in the graph of Figure 3.5 (e.g., hwb10-3631, hwb11-9314 and $cycle10_2d1$).

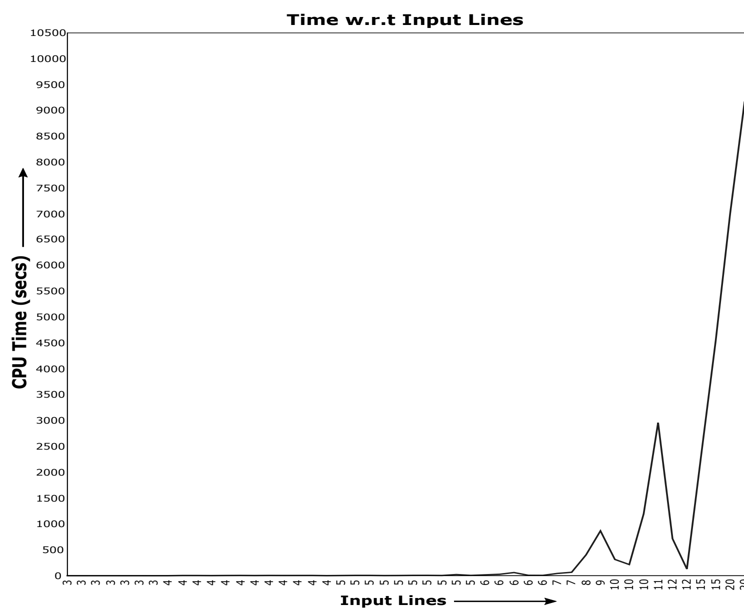


Figure 3.5: *Plot of CPU time vs. Number of input lines*

(ii) **CPU time vs. Gate count:** In Figure 3.6, we can perceive that there is a random increase and decrease in CPU time in the graph. High escalations can be observed for benchmark circuits $cycle17_3d1$ and $mod1024adder1$ with gate counts of 48 and 55, and take 7002.426 secs and 9161.228 secs CPU time to generate the test vector, respectively. The circuit hwb11-9314 has 9314 gates, but the CPU time taken to generate the test vectors is $2960.651 secs$. The gate count of the circuit $mod1024adder1$ is 55, which takes more CPU time to generate the test vectors compared to the circuit hwb11-9314 with 9314 gates. This happens due to more

number of inputs in *mod*1024*adder*1 (20 input lines) than *hwb*11-9314 (11 input lines).Therefore, the impact of the number of input lines is more significant than the number of gates present in the circuit.

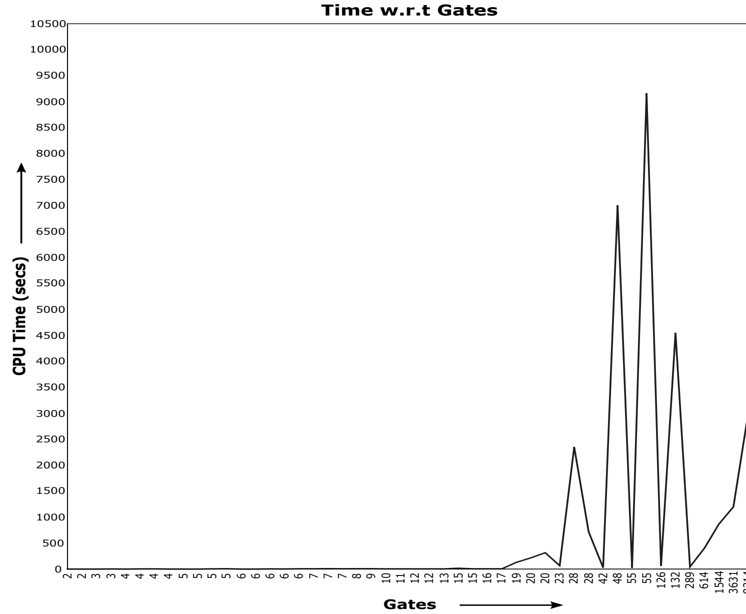


Figure 3.6: *Plot of CPU time vs. Number of gates*

(iii) **CPU time vs. Number of faults:** In Figure 3.7, we observe that there is an escalation in CPU time with the increase in the total number of faults for most of the benchmark circuits, but there is a reduction of CPU time for some circuits with more number of faults. Consider the circuit *hwb*11-9314 with total number of faults 37930680 and the circuit *ham*15*d*1 with total number of faults 8712032. The CPU time required to generate the test vectors are 2960.651*secs* and 4547.57*secs*, respectively. Though the number of faults in *ham*15*d*1 is less than the number of faults in *hwb*11-9314, but to generate the test vectors, the circuit *ham*15*d*1 takes more CPU time compared to the circuit *hwb*11-9314. This effect is due to the presence of more number of input lines in *ham*15*d*1 (15 input lines) compared to *hwb*11-9314 (11 input lines). It is also observed that *ham*15*d*1 has less number of gates (132 gates) than *hwb*11-9314 (9314 gates). Similar pattern is also observed in the graph for the circuit *hwb*10-3631 (10 inputs, 3631 gates,

7358432 faults, 1197.964 secs CPU time) and $rd84d1$ (15 inputs, 28 gates, 1899616 faults, 2351.187 CPU time).

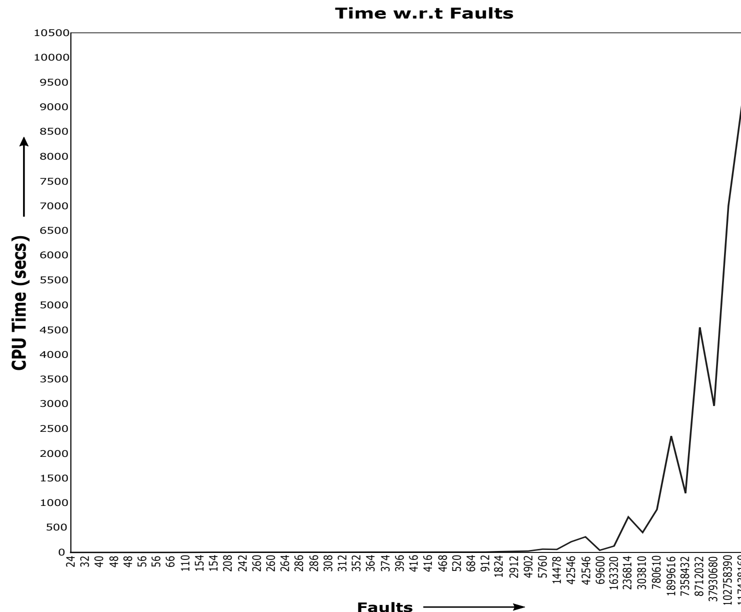


Figure 3.7: *Plot of CPU time vs. Number of faults*

Our experimental results are compared with [3], [4] , [2] and these are reported in Table 3.5, Table 3.6 and Table 3.7, respectively. The authors in [3] proposed an optimal test set generation method for all possible SIBF and SIRBF. In the proposed work, the SIBF, MIBF, SIRBF and MIRBF are considered. The total number of faults detected by the proposed scheme for AND and OR bridging faults are given in Column 4 in the tables. It is observed that the required number of test vectors to detect all the bridging faults by the proposed scheme is less compared to other techniques ([3], [4], [2]) for all the benchmark circuits. The maximum reduction of test vector size is 81.82% for $Ham7$ benchmark reversible circuit when compared to technique proposed in [3] which is tabulated in Table 3.5. The authors in [4] used the DFT method for generating the test vectors to detect the SIRBF and single stuck-at faults. In contrast, our proposed method generates the test vectors for detecting SIRBF with additional bridging faults such as SIBF, MIBF, and MIRBF. It is found that the number of test vectors required in case of our proposed method is almost similar to [4], however, with no extra circuit

Table 3.5: *Comparison of the complete test set with [3]*

Benchmark Circuit	No. of Inputs/Outputs	No. of Total Detectable Faults [3]	No. of Total Detectable Faults [Proposed]	No. of test vectors [3]	No. of test vectors [Proposed]
		SIBF+SIRBF	SIBF+MIBF+SIRBF+MIRBF		
Ham3\Design#1	3/3	18	48	3	2
Graycode6	6/6	90	684	9	3
4_49\design#3	4/4	78	286	5	4
Ham7\Design#1	7/7	504	5760	22	4
rd32\design#1	4/4	30	110	6	3
Xor\Design#1	5/5	50	260	8	3

Table 3.6: *Comparison of the complete test set with [4]*

Benchmark Circuit	No. of Inputs/Gates	No. of Total Detectable Faults [4]	No. of Total Detectable Faults [Proposed]	No. of Test Vectors [4]	No. of Test Vectors [Proposed]
		SIBF+SIRBF	SIBF+MIBF+SIRBF+MIRBF		
ham3tc	3/5	24	48	4	2
graycode6	6/5	126	684	5	4
hwb4-11-23	4/11	120	264	4	4
4_49-12-32	4/12	130	286	4	4
mod5adder-15	6/15	336	1824	5	4
ham7tc	7/23	672	5760	5	4
cycle17-3	20/48	10290	102758390	7	8
mod1024adder	20/55	11760	117438160	7	7
ham15tc1	15/132	15960	8712032	6	8
hwb9-1544	9/1544	69525	780610	6	7
hwb10-3631	10/3631	199760	7358432	6	7
hwb11-9314	11/9314	614790	37930680	6	6

Table 3.7: *Comparison of the complete test set with [2]*

Benchmark Circuit	No. of Inputs/Outputs	No. of Total Detectable Faults [2]	No. of Total Detectable Faults [Proposed]	No. of Test Vectors [2]	No. of Test Vectors [Proposed]
		SIBF+MIBF	SIBF+MIBF+SIRBF+MIRBF		
6symd2	6/1	90	42546	3	3
9symd2	9/1	352	236814	5	5
hwb7	7/7	152	69600	4	3
hwb8	8/8	238	303810	4	7
hwb6	6/6	90	21736	3	3
rd73	7/3	152	42546	4	4
rd84	8/4	238	1899616	4	4
ham7	7/7	152	5760	4	4
ham15	15/15	1848	8712032	8	8
mod1024adder	20/20	4598	117438160	10	7

overhead due to the non-adaptation of DFT method. The comparison of the proposed work with [4] is reported in Table 3.6. The work in [2] specifically targets generating the universal test set for both SIBF and MIBF and also for input stuck-at faults based on the shift operation on unitary matrix. It is found that the number of test vectors generated by our proposed method is equal when compared to the method of Sarkar et al. [2], but the proposed method covers more types of fault, which is clearly visible in Table 3.7.

Finally, according to the experimental results reported, it is evident that the number of test vectors generated by our proposed method is less or equal to the existing methods. Moreover, the proposed method covers more types of fault as compared to existing methods and achieves 100% fault coverage.

3.5 Conclusion

In this chapter, we discussed a scheme for minimal complete test set generation to detect all the bridging faults, which includes SIBF, SIRBF, MIBF, and MIRBF, in reversible circuits. The concept of path-level expression is introduced in this work to generate the complete test set. The path-level expression has the ability to capture the level-wise information on a given reversible circuit. After collecting all the path-level expression, a matching process is applied to collect the required paths which are used to generate the test set. It is also established that the generated test is the minimal one. The reversible circuits that consist of NCT and GT gate libraries are used in this work to carry out the experiments. The generated test set is capable for 100% fault coverage, which is shown with experimental results. The next chapter presents the test set generation methods for single and multiple missing gate faults. We also discuss the correlation of missing gate fault model with other fault models in reversible circuits.

Chapter 4

Test Generation for Multiple Missing-Gate Faults in Reversible Circuits

4.1 Introduction

In this chapter, we consider the problem of testing in reversible circuits with respect to Missing Gate Fault (MGF) Model, which is used explicitly for reversible circuits. The detail discussion of different types of faults under MGF is provided in Chapter 1. The present chapter provides an ATPG method to generate the complete test set for detecting the single and consecutive multiple missing gate faults (MMGFs) in the reversible circuits.

The proposed complete test set generation method is twofold. Firstly, the local test pattern is applied to each level of the k-CNOT gate and the reverse simulation method is used for identifying all the possible Single Missing Gate Faults (SMGFs). Secondly, using the complete test set for SMGFs and based on the structure of the k-CNOT based circuit, a test set is formulated. The generated test set is capable of detecting all the MMGFs and as well as the SMGFs in reversible circuits. However, the generated complete test set is not minimal. For achieving the minimality, a table is constructed covering row and column faults and an Integer Linear Programming (ILP) problem is formulated to achieve the minimality of the test set. The proposed method is designed for the k-CNOT based reversible circuit structure and it gives the minimal complete test set with nearly 100% fault coverage. Moreover, the correlation with other fault models like stuck-at faults, appearance crosspoint faults and partial missing gate faults to the

MMGF model is also established. The experimental results demonstrate that the size of the generated minimized test set is smaller or similar as compared to the existing methods and attains 100% fault coverage.

The rest of this chapter is organized as follows: Section 4.2 provides related works of fault detection under the missing gate fault model of reversible circuits. Section 4.3 describes the proposed ATPG algorithms for generating the complete and minimal test set to detect the SMGFs and MMGFs in reversible circuits. The comparison with other fault models is also included in this section. The experimental results of the proposed test set generation method and comparison with other fault models are reported in Section 4.4. Finally, concluding remarks are presented in Section 4.5.

4.2 Related Work

Some of the existing research works that are compatible with our work are briefly reviewed in this section. In 2004, Hayes et al. [7] showed that $\lceil N/2 \rceil$ test vectors detect all Missing Gate Faults (MGFs) in an N-gate k-CNOT circuit. Here, the MGF considers one gate missing at a time. Also, this method proposed that a single vector is capable of detecting all the MGFs of a given k-CNOT circuit by adding one wire and several 1-CNOT gates. The proposed DFT method inverts the values that correspond to the undetectable faults such that all detection conditions can be satisfied simultaneously. Based on the concept of the previous work, the authors in [8] proposed a different type of fault that occurs under the Missing-Gate Fault model. This work presented a method to generate the optimal test sets computed by integer linear programming (ILP) to detect the various types of MGFs such as Single Missing Gate Fault (SMGF), Repeated Gate Fault (RGF), Multiple Missing Gate Fault (MMGF), and Partial Missing Gate Fault (PMGF). Also, this work showed that the complete test set of SMGFs is not capable of detecting all the MMGFs. The total number of MMGFs in an N-gate circuit is $N(N + 1)/2$. In 2008, authors in [53] proposed a scheme that divides the circuit into sub-circuits to get the complete test. The division of the circuit is based on the dominant and independent relationship of the gates. If the consecutive k-CNOT gates are

dominant, then they are divided into the same sub-circuit; otherwise, these gates belong to different sub-circuits. This work generated the test vectors for each sub-circuits to obtain the complete test set. However, the generated complete test set by dividing the sub-circuit method is not minimal. The authors proposed the set covering method to get the minimal test set for detecting the SMGFs and MMGFs in k-CNOT circuits. The methodology proposed by them does not cover all the MMGFs and several additional test vectors are required to detect all the MMGFs. In 2008, Rahaman et al. [40] proposed a DFT method by adding only one extra line along with duplication of each k-CNOT gate to get a universal test set of size $(n + 1)$, which is sufficient to detect all SMGFs along with other faults like RGFs and PMGFs. However, this approach may not be suited for MMGFs, and for that additional test vectors are required. Further augmentation in the circuit is needed. In 2010, Kole et al. [54] proposed an algorithm for detecting SMGFs, MMGFs and RGFs. The proposed algorithm derived an optimal test set (OTS), where each gate is represented by a gate Id and each Id is used as a key to represent the permutation produced by the k-CNOT corresponding gate. Here, all permutations of size n for each gate Id are generated in a given an n-input reversible circuit and $N(N + 1)/2$ sets are constructed where circuit depth is N. The minimal set cover is used to derive an OTS from these sets. It is observed that if the circuit size is large in terms of a number of gates, then the construction of permutations for each gate Id is complex. In 2011, Zamani et al. [41] proposed a technique named as Ping-Pong testing that provided a test vector to the circuit and generated output is considered as the next vector to detect the SMGFs and RGFs. This technique showed 100% fault coverage for SMGF as well as single RGF. But, for the multiple MGF (MMGFs) fault coverage is 86% on an average. In 2014, Mondal et al. [9] proposed a Boolean generator which is developed by the Boolean difference method to derive the test set. The derived test is in the form of Boolean expression only, and it is capable of detecting all the SMGF in k-CNOT based reversible circuits. In 2016, Nagamani et al. [6] proposed an ATPG algorithm using the exact approach for generating the complete test set which can detect the single and multiple stuck-at faults, single and multiple missing gate faults, repeated

gate faults, and partial missing gate faults. These algorithms provided an optimal solution, but the complexity of these algorithms is exponential to the gate count and the number of lines in the circuit, which is not suitable for large circuits. In 2017, Prakash Surhonne et al. [5] provided a method to generate Automatic test patterns for MMGFs detection (considered only two gates are missing). It is based on the generated SMGFs which are stored in a Binary Decision Diagram (BDD) and test patterns to detect all MMGFs are generated by dependency analysis between the two gates.

It is observed that many of the approaches considered the DFT methodology for generating the test vectors to cover all the faults. In DFT methodology, there is an additional circuit overhead which is incorporated by additional input lines or control lines and additional gates. Moreover, by using an exact approach, the minimal complete test set can be generated, but the computational complexity grows exponentially for large reversible circuits. The proposed ATPG method to determine the complete test set for detecting all SMGFs and MMGFs using the reversible circuit properties (controllability and observability) without changing the structure of the circuit. Moreover, the proposed complete test set generation method has reasonable complexity and it can be applied to any large k-CNOT based reversible circuits.

4.3 Proposed Method

In this section, a method is proposed for k-CNOT reversible circuit to generate the complete test set which can detect the SMGFs and MMGFs. The proposed method starts with the generation of the complete test set for detecting all the SMGFs. A local test pattern is applied to each gate of the circuit and by using the reverse simulation technique, the complete test set is generated to detect the SMGFs of the given k-CNOT based reversible circuits. After analysis of the complete test set for SMGF, it is observed that the generated complete test set for SMGF is unable to detect all the possible MMGFs in a given reversible circuit. Therefore, a solution is formulated to generate the complete test set for detecting all the MMGFs considering the structure of the k-CNOT circuit and the complete test set for detecting SMGFs. The generated complete

test set is able to detect all the SMGFs and MMGFs. The generated test set to detect SMGF and MMGF is not minimal. To obtain the minimality, a table is constructed by using fault simulation with the generated test set. Integer Linear Programming (ILP) is formulated by using the fault simulation table and the minimal test set is obtained for a given circuit by the Branch and Bound technique of ILP.

Some terms are defined formally which are used to describe the proposed solution.

Definition 4.3.1. *A test vector* $\mathbf{TV_i}$ *is a combination of binary inputs that are applied to a reversible circuit for testing. The binary inputs* $\langle b_1\ b_2\ \ldots\ b_n\rangle$ *are assigned to the input lines, where* b_i *is the* i^{th} *bit that refers to the* i^{th} *line of the reversible circuit.*

Definition 4.3.2. *The test set* **TS** *is the set of test vectors that are required to test all the possible faults (SMGFs and MMGFs) in the reversible circuit. Let the test set be* TS=$\{TV_1,\ TV_2...,\ TV_k\}$*, for* $1 \leq i \leq k$*, then the test vector is* TV_i=$\langle b_{1i}\ b_{2i}\ \ldots\ b_{ni}\rangle$*, where* b_{ji} *is the* j^{th} *bit of* i^{th} *test vector and* $b_{ji} \in \{0,1\}$.

Definition 4.3.3. *A local test pattern* $\mathbf{TV_{lp}}$ *is a combination of binary inputs that are applied to each k-CNOT gate of the reversible circuit to activate the gate for detecting any faults therein. Let the local test pattern be* TV_{lp}=$\langle b_1\ b_2\ \ldots\ b_n\rangle$*, where* b_i *is* i^{th} *bit that refers to the* i^{th} *line for* $1 \leq i \leq n$ *and* $b_i = 1$.

Definition 4.3.4. *The test set* $\mathbf{TS_{SMGF}}$ *is the complete test set to detect all the single missing gate faults (SMGFs) in a given reversible circuit. In other words, the test set* TS_{SMGF} *is capable of detecting all the SMGFs, which occur at any level in the reversible circuit and* $TS_{SMGF} \subset TS$.

Definition 4.3.5. *The test set* $\mathbf{TS_{MIN}}$ *is the minimal complete test set to detect all the single and multiple missing gate faults in a given reversible circuit.*

Definition 4.3.6. $\mathbf{F_{SMGF}}$ *and* $\mathbf{F_{MMGF}}$ *are the sets consisting of all single and multiple missing gate faults, respectively in a given n-input reversible circuit.*

The notation f_{g_i} for SMGF is used to denote a single missing gate fault of the ith gate for $1 \leq i \leq N$. The multiple missing gate fault for missing of q number of gates is denoted as $f_{g_1,g_2\ \ldots g_q}$.

4.3.1 Detection Technique for Single Missing Gate Fault

Consider a reversible circuit consisting of N gates $\{g_1, g_2,\ \ldots, g_N\}$. For every gate g_i, i=1 to N, there are some control line(s) (denoted as $\bullet$), some unconnected line(s) and a target line (denoted as $\oplus$). To generate the test set for SMGFs, the gates are scanned from left to right. To activate the gate g_i for detecting any faults therein, a local test

vector is applied to the gate g_i by assigning the logic value 1 on the lines where control connections are present, and randomly we set the logic value to either 0 or 1 on all other lines (unconnected and target lines). In this work, the logic value 1's is considered for all lines in a given k-CNOT based circuit, and a local test pattern TV_{lp} represents it. With the applied local test pattern TV_{lp} to the gate g_i, the back propagation toward the input side is used to obtain the required test vector (TV_i) to detect the missing gate fault for gate g_i. We carry out fault simulation with test vector TV_i to detect SMGFs, and then we remove the faults that are detected by TV_i. These detected faults need not be considered for further iteration and we repeat this process until all the faults are covered for the fault set F_{SMGF}.

4.3.2 Complete Test Set Generation for Single Missing Gate Fault

In this section, the proposed method to generate a complete test set for detecting all single missing gate faults in a given reversible circuit is described. The complete test set generation method for single missing gate faults is presented in Algorithm 2.

Algorithm 2: Complete test set TS_{SMGF} generation for detecting the SMGFs.

Input: Reversible Circuit C with n input-lines and N-gates.
Output: The complete test set TS_{SMGF} for detecting the SMGFs.

1. Extract the required parameters n and N from circuit C comprising of N gates $g_1, g_2, \ldots, g_N$ with n input lines. Construct the local test vector $TV_{lp} = \langle b_1\ b_2\ \ldots b_n \rangle$, $b_i = 1$ and i= 1 to n
2. Generate the fault set F_{SMGF} consisting of all SMGFs in a given n-input reversible circuit.
3. Back propagate TV_{lp} for each gate g_i to obtain the corresponding input test vector TV_i (occur at input level) and $TS_{SMGF} \leftarrow TV_i$. Since every gate g_i is reversible, for a given TV_{lp}, the test vector TV_i is unique.
4. Carry out fault simulation with test vector TV_i to determine the faults in fault set F_{SMGF} that get detected.
5. Construct the complete test set TS_{SMGF} after eliminating the all duplicate test vector TV_i.

Example 4.3.1. The complete flow of Algorithm 2 for generating the test set TS_{SMGF} of the reversible benchmark circuit $rd32$-$v0_66$ is represented in Figure 4.1.

Let the benchmark circuit $rd32$-$v0_66$ be provided as input to Algorithm 2 having n-lines and N-gates. In Step 1, the required parameters $n = 4$ and $N = 4$ are extracted from the given circuit. Now, we construct the local test pattern TV_{lp}=$\langle b_1\ b_2\ b_3\ b_4 \rangle$, where

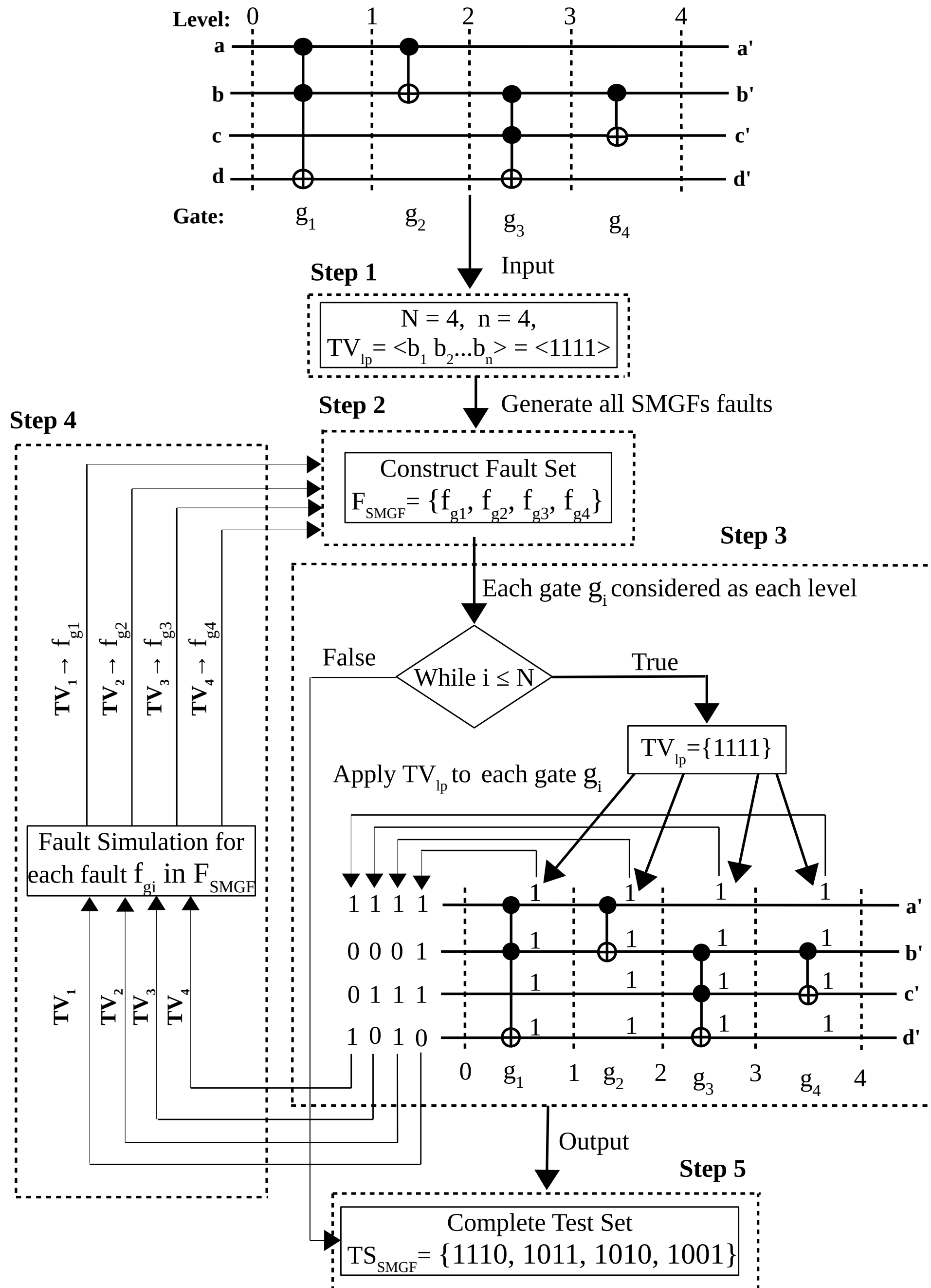


Figure 4.1: *Demonstration of Algorithm 2 for the circuit rd32_ v0_66*

bit b_i is assigned to logic value 1, where $i = 1$ to 4. In Step 2, the fault set $F_{SMGF} = \{f_{g1}, f_{g2}, f_{g3}, f_{g4}\}$ is generated based on the connections of N gates. In Step 3, we apply and back propagate $TV_{lp} = \langle 1\ 1\ 1\ 1\rangle$ to the gate g_4 and obtain the corresponding test vector $TV_4 = \langle 1\ 0\ 0\ 1\rangle$ at the input level. Similarly, the same process is repeated for the remaining gates g_3, g_2, and g_1 and the corresponding test vectors are $TV_3 = \langle 1\ 0\ 1\ 0\rangle$, $TV_2 = \langle 1\ 0\ 1\ 1\rangle$, and $TV_1 = \langle 1\ 1\ 1\ 0\rangle$, respectively. In Step 4, the fault simulation process for each fault f_{gi} in the fault set F_{SMGF} is carried out with the help of each test vector TV_i at the input level. For this example, the faults f_{g1}, f_{g2}, f_{g3}, and f_{g4} are identified by test vectors $TV_1 = \langle 1\ 1\ 1\ 0\rangle$, $TV_2 = \langle 1\ 0\ 1\ 1\rangle$, $TV_3 = \langle 1\ 0\ 1\ 0\rangle$, and $TV_4 = \langle 1\ 0\ 0\ 1\rangle$, respectively. Finally, complete test set $TS_{SMGF} = \{1110, 1011, 1010, 1001\}$ is constructed for the benchmark circuit $rd32\text{-}v0_66$ as mentioned in Step 5.

Lemma 4.3.1. *The test set, TS_{SMGF}, generated using the proposed method detects all single missing gate faults in a given reversible circuit with N-gates.*

Proof. Let us consider the k-CNOT circuit C consisting of gates $\{g_1, g_2, \ldots, g_N\}$. To activate the SMGF at gate g_i (i=1 to N), the local test pattern TV_{lp}=$\langle b_1\ b_2\ \ldots\ b_n\rangle$ = $\langle 1\ 1\ \ldots\ 1\rangle$ is applied at the output level of gate g_i. The controllability property ensures that on backtracking from any gate g_i with local test pattern TV_{lp} it is always possible to generate a unique vector (say TV_i) at the input gate-level. The observability property ensures that the generated test vector TV_i for each gate g_i produces fault-free output and the same test vector TV_i produces faulty output at the primary gate-level, if a fault occurs. Therefore, each test vector $TV_i \in TS_{SMGF}$ detects all the individual faults in the fault set F_{SMGF}. Hence, the test set TS_{SMGF} is the complete test set for detecting all the single missing gate faults in a given reversible circuit. □

4.3.3 Complexity Analysis of complete test set generation

For an n-input reversible circuit with N number of gates, the local test pattern TV_{lp} is applied for each gate g_i for detecting the SMGFs in the circuit. The generated test vector TV_i at the input level for each gate g_i is simulated N number of times (i.e., $|F_{SMGF}|$=N). Therefore, back propagation and fault simulation for N number of gates require linear time. Hence, the time complexity is $O(N)$ for generating the complete test set TS_{SMGF}.

4.3.4 Complete Test Set Generation for Multiple Missing Gate Fault

The generated test set TS_{SMGF} obtained by Algorithm 2 is sufficient for detecting all the single missing gate faults (SMGFs) in a reversible circuit, but the test vector $TV_i \in TS_{SMGF}$ is not capable of detecting all the multiple missing gate faults (MMGFs).

Moreover, a complete test set for SMGFs does not cover all the MMGFs [8]. For every gate g_i of a reversible circuit, there are some control line(s) (denoted as $\bullet$), some unconnected line(s) and a target line (denoted as $\oplus$).

In k-CNOT circuit structure, some of the lines contain only target connections, and in some lines both target and control connections are available. If only the target connections are present in a line, then some of the multiple missing gate faults can not be detected by the test set for detecting SMGFs. Consider the case that two consecutive gates are missing whose targets are in the same line, then the single missing gate fault of the first gate is nullified by the missing of the second gate and so missing of two gates cannot be detected by the test set of SMGFs. Therefore, for this category of circuits, the test vector $TV_i \in TS_{SMGF}$ cannot detect all the MMGFs in a given k-CNOT based circuit. For detecting all the MMGFs, some additional test vectors are included in the test set TS_{SMGF}. For this purpose, the test set TS is constructed as: $TS = \bigcup_{i=1}^{N} S(g_i) \cup TS_{SMGF}$, $S(g_i)$ represents all the possible test vectors TV_i for the corresponding SMGFs in gate g_i. In the other case, target and control connections are present on the same line. Consider a case where a control point is followed by a target point is a line. That is, gate g_i is connected as target point and gate g_j is connected as a control point in the line l. So, the missing of gate g_i effects the control connections of gate g_j which eventually effects the output of gate g_j. Missing of both the gates g_i and g_j effect the control connects of the next gate. The absence of the control and target in these gates directly effect the control and target connection of the next consecutive gates, and as a result, the primary output of fault-free circuit gets effected. Therefore, the generated test set TS_{SMGF} for detecting SMGFs is capable of detecting all the MMGFs in a given k-CNOT circuit for this category of circuits. The construction of the complete test set TS for detecting all MMGFs is given in Algorithm 3.

Example 4.3.2. Consider the benchmark circuit Toffoli_*double*_4 as illustrated in Figure 4.2. The circuit consists of two k-CNOT gates (i.e., N=2) and four input lines (i.e., n=4). Based on the circuit structure, as shown in Figure 4.2, there is an occurrence of two consecutive target connections on the same line 'd'. Hence, the generation of the complete test set TS is according to the Step:4 of Algorithm 3. The complete test set for SMGFs is TS_{SMGF}=$\{1110, 1111\}$ for the reversible circuit Toffoli_*double*_4 according to Algorithm 2. In the circuit of Figure 4.2, if the gate g_1 is missing then all

Algorithm 3: Complete test set generation for MMGFs

Input: k-CNOT based reversible Circuit.
N: Number of gates of the circuit.
n: Number of input lines of the circuit.
TS_{SMGF}: The complete test set for SMGFs.
$S(g_i)$: All the possible test vectors TV_i for SMGFs in gate g_i.
Output: The complete test set TS for detecting MMGFs

```
1 TS=TS_SMGF
2 flag=false
3 for i ← 1 to n do
4     if line l_i contains only target connection then
5         flag=true
6 if (flag) then
7     for j ← 1 to N do
8         TS=TS ∪ S(g_j)
9 return TS
```

possible test vectors to detect the missing gate g_1 is $S(g_1)$ = {1010, 1011, 1110, 1111} and similarly all possible test vectors to detect the missing gate g_2 is $S(g_2)$ = {1100, 1101, 1110, 1111}. Now, the required test set to detect all MMGFs is TS={$S(g_1) \cup S(g_2)$}$\cup TS_{SMGF}$={1010, 1011, 1110, 1100, 1101, 1111}. The obtained test set TS is capable of detecting all the faults in fault sets F_{SMGF} and F_{MMGF}. Thus, the test set TS is complete but not minimal to detect all MMGFs in the reversible benchmark circuit Toffoli_*double*_4.

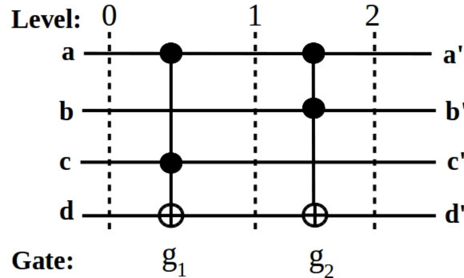


Figure 4.2: *Toffoli_double_4 benchmark circuit*

Lemma 4.3.2. *The test set TS generated using the proposed method detects all single and multiple missing gate faults in a given n-input k-CNOT based reversible circuit.*

Proof. Let us consider that the target connections of the gates g_i and g_{i+1} which are in the same line. It means that the operation of the gate g_i does not effect the gate g_{i+1} if they are missing together, i.e., fault-free and faulty output are indistinguishable, since the missing of two consecutive target connections does not effect the functional behavior of the circuit. If the generated test vector $TV_i \in TS_{SMGF}$ is applied, then missing of even number of consecutive gates shows the similar functional effect as compared

to the fault-free circuit. Due to this, the test set TS_{SMGF} is unable to detect all the MMGFs. In this case, all the possible test vectors $S(g_i)$ for all possible occurrence of SMGFs are considered. However, each single missing gate fault is also a multiple missing gate fault [8]. Thus, any test vector in $S(g_i)$ involves detecting two or more multiple missing gate faults where the target connection is in the same line for each gate. Hence, the generated test set $TS=\bigcup_{i=1}^{N} S(g_i) \cup TS_{SMGF}$ is capable of detecting all multiple missing gate faults in a given k-CNOT based circuit. In another case, the control connection of gate g_i is the target connection of the gate g_{i+1} and vice versa; then control connection of gate g_i directly effects the gate g_{i+1}. The complete removal of two gates g_i and g_{i+1} generate the faulty responses at primary output of the circuit, which is distinguishable with fault-free primary output responses. The multiple missing gate faults in this scenario are detected by the test vector $TV_i \in TS_{SMGF}$. Since the back propagation of each gate produces different vectors in subsequent levels, thus, at least one generated test vector $TV_i \in TS_{SMGF}$ can detect the multiple missing gates g_i and g_{i+1}. In the similar analogy, it can be stated that the test set TS ($TS = TS_{SMGF}$) is capable of detecting more than two multiple missing gate faults. □

4.3.5 Complexity Analysis of complete test set TS generation

The generation of the complete test set TS for all possible SMGFs and MMGFs is dependent on the number of input lines n and the number of gates N in the circuit. All the lines in the circuit are scanned to identify the type of connections (target and control) present in a line for each gate, and for checking the type of connection in a line requires constant time (C_1). The time complexity for checking the type of connections is $N \times n \times C_1$. Evaluation of the test vectors for each gate also requires constant time (C_2) and time complexity for evaluation of the test set for all the gates is $N \times C_2$. Hence, the time complexity for generating the test set TS is $N \times n \times C_1 + N \times C_2 = O(N \times n)$. The time complexity in the worst case is $O(N^2)$, where $n = N$.

4.3.6 Determination of Minimal Complete Test Set

The complete test set TS generated by Algorithm 3 to detect all MMGFs is not a minimal one. A method is proposed to derive the minimal complete test set TS_{Min}. For this purpose, firstly, a row and column fault covering table is constructed with the help of the complete test set TS and all possible faults present in F_{SMGF} and F_{MMGF} in a given reversible circuit. Secondly, Integer Linear Programming (ILP) Problem is formulated from the constructed row and column fault covering table. Finally, using

Branch and Bound technique of ILP, the test set TS_{MIN} is obtained for detecting all SMGFs and MMGFs. The following steps are carried out to generate TS_{MIN}:

(i) Using fault simulation with each test vector $TV_i \in TS$, the corresponding faults in F_{SMGF} and F_{MMGF} are determined. The row and column fault covering table is constructed in the form of a matrix $\mathbf{M_{r \times c}}$, where r is the number of test vectors present in TS and c is the number of all possible faults in a given reversible circuit. The value $\mathbf{m_{(i,j)}}$=1 denotes that the test vector TV_i in the ith row detects the corresponding j^{th} fault; otherwise, the fault is undetectable by the test vector TV_i of the i_{th} row.

(ii) Formulate the ILP model with binary decision variables t_i associated with each test vector TV_i. Let us consider $|TS| = d$, then there are d variables t_i, where $i =$ 0 to $d-1$ and $t_i \in \{0, 1\}$. The variable t_i is represented as i^{th} row of $\mathbf{M_{r \times c}}$ and, $\overline{T} = [t_0, t_1, \ \ldots, t_{d-1}]^T$. Here, we define:

$$\mathbf{M_{r \times c}.\overline{T}} = \Big[\sum_{\mathbf{i=0}}^{\mathbf{d-1}} \mathbf{m_{i1}t_i}, \sum_{\mathbf{i=0}}^{\mathbf{d-1}} \mathbf{m_{i2}t_i}, \ \ldots \sum_{\mathbf{i=0}}^{\mathbf{d-1}} \mathbf{m_{iN(N+1)/2}t_i}\Big]^{\mathbf{T}}$$

where, $\sum_{i=0}^{d-1} m_{ij}$ for all $1 \leq j \leq N(N+1)/2$ and if t_i=1, then corresponding test vector TV_i is able to detect the faults of jth row of $\mathbf{M_{r \times c}}$.

The Objective function for ILP is formulated as follows:

$\mathbf{min\ f}(t_0, t_1, \ \ldots, t_{d-1})$= t_0+t_1+ ...+t_{d-1}

subject to the constraints $\mathbf{M_{r \times c}.\overline{T}} \geq [\mathbf{1}, \mathbf{1}, \ \ldots, \mathbf{1}]^{\mathbf{T}}$

(iii) All the constraints $\sum_{i=0}^{d-1} m_{ij}t_i \geq 1$ are applied in LINGO 17.0 [55] and shows that at least one test vector $TV_i \in TS$ is able to detect any fault in a given circuit.

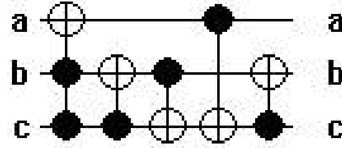


Figure 4.3: *ham3tc benchmark circuit*

Example 4.3.3. Consider the benchmark circuit *ham3tc* as shown in Figure 4.3. The total number of faults in the circuit *ham3tc* is 15. According to Algorithm 3, the extracted complete test set TS={011, 101, 110, 100} for the reversible benchmark circuit *ham3tc*. Using the fault simulation with each test vector $TV_i \in TS$ with the fault-free and faulty circuit, the corresponding faults are extracted for the circuit *ham3tc*. Based on this information, the row and column fault covering table is constructed according to the Step (i) to generate TS_{MIN}. Table 4.1 shows the fault coverage by each test vector TV_i. Next, we formulate the ILP model as mentioned in Step (ii): **min f**(t_0, t_1, t_2, t_3)=t_0+t_1+t_2+t_3, subject to the generated constraints as indicated in Table 4.1. In Step (iii), all the constraints generated by the ILP formulation are applied to the LINGO 17.0, and it gives t_0=1 and t_1=1. The respective test vectors for the binary decision variables t_0 and t_1 are 011 and 101, respectively. Therefore, the minimal test set is TS_{MIN}={011, 101} for the reversible benchmark circuit *ham3tc*.

4.3.6.1 Complexity Analysis of ILP formulation

For the determination of a complete minimal test set TS_{Min}, we consider 0-1 ILP, where each binary decision variable $t_0, t_1, \ldots, t_{d-1}$ can assume binary value either 0 or 1. Suppose, consider the number of constraints is k, and all the constraints are lower bound constraints. For the state space, let us consider all possible 2^d assignments of binary decision variables $t_0, t_1, \ldots, t_{d-1}$ in a non-deterministic manner. Based on the ILP formulation, the time taken for checking the feasible solution for each assignment is $O(d \times k)$. The minimized objective function is solved using the Branch and Bound technique of ILP and computing the value of the objective function for each feasible assignment needs $O(d)$ time.

Lemma 4.3.3. *The test set TS_{MIN} is a minimal complete test set for detecting all the single and multiple missing gate faults in a given n-input k-CNOT based reversible circuit.*

Proof. In Lemma 4.3.2, it is established that the test set TS can detect all single and multiple missing gate faults of a given reversible circuit. Moreover, each possible fault can be detected by one of the test vector $TV_i \in TS$ because of each TV_i satisfies the condition

Table 4.1: *Row and Column fault coverage table of the ham3tc benchmark circuit*

TV_i	F_{SMGF}					F_{MMGF}									
	f_{g1}	f_{g2}	f_{g3}	f_{g4}	f_{g5}	$f_{g1,g2}$	$f_{g2,g3}$	$f_{g3,g4}$	$f_{g4,g5}$	$f_{g1,g2,g3}$	$f_{g2,g3,g4}$	$f_{g3,g4,g5}$	$f_{g1,g2,g3,g4}$	$f_{g2,g3,g4,g5}$	$f_{g1,g2,g3,g4,g5}$
011	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
101	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0
110	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0
100	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1

following constraints are generated from the fault coverage table

$$\sum_{i=0}^{d-1} m_{ij} t_i \geq 1 \Longrightarrow M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \overset{constraints}{\Longrightarrow} \begin{array}{c} t_0 \geq 1 \\ t_0 + t_1 \geq 1 \\ t_1 + t_2 \geq 1 \\ t_0 + t_1 + t_2 + t_3 \geq 1 \\ t_1 + t_3 \geq 1 \\ t_0 + t_1 + t_2 \geq 1 \\ t_0 + t_3 \geq 1 \\ t_0 + t_1 + t_3 \geq 1 \end{array}$$

$\mathbf{M_{r \times c}}.\overline{\mathbf{T}} \geq [\mathbf{1}, \mathbf{1}, \ldots, \mathbf{1}]^{\mathbf{T}}$. Based on the ILP formulation, the objective function **min** f$(t_0, t_1, \ldots, t_{d-1})$ provides the smallest number of binary variable(s) which is associated with their corresponding test vector TV_i in the test set TS under the condition $\mathbf{M_{r \times c}}.\overline{\mathbf{T}} \geq [\mathbf{1}, \mathbf{1}, \ldots, \mathbf{1}]^{\mathbf{T}}$. Thus, all the constraints of a given condition are applied to optimal software that gives least number of variables, which are assigned to TS_{MIN}. Hence, TS_{MIN} is a minimal complete test set for detecting all single and multiple missing gate faults. □

4.3.7 Fault Coverage Evaluation with other Faults Models

The fault coverage is defined as the ratio of the actual number of detected faults to the total number of faults that occur in a circuit. Several fault models are used to detect different kind of faults. In this work, methods are proposed to generate the complete test set to detect SMGFs and MMGFs in a reversible circuit. There exists a correlation

between different fault models and it is also a good exercise to check the fault coverage of other fault models by the generated test set of another fault model. Therefore, the minimal test set generated by our proposed method to detect SMGFs and MMGFs is applied for detecting the faults in other fault models such as stuck-at fault (SAF), partial missing gate fault (PMGF), and appearance crosspoint fault. The correlation between different fault models are explained briefly.

(a) **Missing gate fault and Stuck-at fault model:** According to Patel et al. [33], the stuck-at faults (SAFs) can be detected by test vector such that each wire at every level of the circuit can be set to both logic value 0 and 1, while SMGFs and MMGFs are detected by setting the control connections to logic value 1 and other connections (the target and unconnected) are set arbitrarily to logic values 0 and 1. In the proposed method, a local test pattern to each k-CNOT gate g_i is applied and we traverse back towards input to obtain the test vector TV_i at the input level. During back propagation through the gates present in the circuit, the operations performed in each gate changes the local test patterns at each level and are set to logic value 0 or 1, which are capable of detecting stuck-at 1 (SA1) and stuck-at 0 (SA0), respectively. Thus, the complete test set TS for SMGFs and MMGFs satisfies the requirement for detecting most of the SAFs in a given reversible circuit.

(b) **Missing gate fault and PMGF model:** The detection criteria for a PMGFs is that the missing control input is set to logic value 0 and we assign the logic value 1 to all other control inputs. The remaining input lines are assigned arbitrarily to logic value 0 and 1. As per our proposed method, when we apply a local test pattern to a particular k-CNOT gate lying at a particular level in the circuit, then the test vectors get changed due to propagation through various gates at different levels. Therefore, there is a possibility to get some test vectors which satisfy the criteria for detecting the PMGFs. It may happen that all PMGFs can not be detected by the generated test set TS by our proposed method, then the test set can be reconstructed to cover all PMGFs.

(c) **Missing gate fault and Crosspoint fault model:** As described above, to detect the SMGFs and MMGFs, we apply a logic value 1 where control connections are present in the gate and we randomly fill by logic value either 0 or 1 for all other lines. To detect for appearance crosspoint faults, we apply a logic value 1 to all control connections of the gate and set logic value 0 to all other lines. Therefore, it is observed that the test set for detecting both SMGFs and MMGFs is capable of covering appearance crosspoint faults. In our proposed method, the complete test set TS is generated by a local test pattern for each gate with the help of back-propagating through various levels in the circuit. Therefore, some test vectors TV_i in the test set TS satisfies the testing criteria for detecting the appearance crosspoint faults. Moreover, the PMGFs are same as the disappearance crosspoint faults.

4.4 Experimental Results and Discussions

The algorithms for the test set generation for single and multiple missing gate faults have been implemented in Python 3.4 and run on a Core-i5 machine with an Intel Pentium (R) CPU-8250U@ 1.60GHz $\times$ 8 system, running Ubuntu v16.04 (64-bit) with 8 GB RAM. The minimal test set generation method has been implemented in LINGO Extended 17.00 software and running on the same core-i5 machine. The reversible benchmark circuits based on the k-CNOT gates have been considered [25] to perform the experiments. The number of test vectors required for the detection of SMGFs and MMGFs before and after minimization along with the CPU time taken as per our proposed method are reported for the experimental results. The experimental results are compared with some of the previous works performed on SMGF and MMGF [5–9]. The fault coverage range with other fault models such as stuck-at fault (SAF), partial Missing gate fault (PMGF) and crosspoint fault (appearance) in reversible circuits are also reported.

The experimental results are reported in Table 4.2. The first four columns in Table 4.2 provide the name of the benchmark circuit, number of input lines (n), number of gates

Table 4.2: *Complete and Minimal test set for detection of SMGFs and MMGFS with CPU time (sec) for the benchmark Circuits*

Benchmarks Circuit	n	N	Total No. Faults	TS	Time (sec)	TS_{MIN}	Time (sec)	% Difference
			(SMGF+MMGF)	(SMGF+MMGF)	TS	(SMGF+MMGF)	TS_{MIN}	(TS & TS_{MIN})
Peres_9	3	2	3	2	0.00	1	0.02	50
Fredkin_6	3	3	6	3	0.00	2	0.02	33.33
nth_Prime	3	4	10	4	0.00	1	0.04	75
ex_1_166	3	4	10	3	0.00	2	0.04	33.33
ham_3	3	5	15	4	0.00	2	0.02	50
3_17_13	3	6	21	4	0.00	2	0.07	50
rd32_v0_66	4	4	10	4	0.00	2	0.06	50
mini_alu_167	4	6	21	5	0.00	3	0.04	40
mod10_171	4	10	55	6	0.00	3	0.11	50
hwb4_52	4	11	66	8	0.00	2	0.09	75
4-49d3	4	12	78	8	0.00	3	0.11	62.50
4_49_16	4	16	136	7	0.00	4	0.17	42.85
hwb4tc	4	17	153	7	0.00	3	0.13	57.14
4gt11_84	5	3	6	3	0.00	2	0.06	33.33
Xor5d1	5	4	10	4	0.00	1	0.06	75
mod5d4	5	5	15	4	0.00	2	0.03	50
alu-v0_26	5	6	21	5	0.00	2	0.07	60
mod5d1_63	5	7	28	5	0.00	2	0.08	60
mod5d1	5	8	36	28	0.00	4	0.05	85.71
mod5d2	5	9	45	8	0.00	2	0.10	75
mod8-1-_177	5	14	105	8	0.00	3	0.12	62.50
rd32_273	5	20	210	15	0.00	7	0.06	53.33
hwb5_55	5	24	300	14	0.00	4	0.15	71.42
hwb5_53	5	55	1540	19	0.00	5	0.19	73.68
hwb5tc	5	56	1596	19	0.00	5	0.26	73.68
graycode6_47	6	5	15	5	0.00	1	0.06	80
ex3_229	6	7	28	7	0.00	3	0.05	57.14
Xor5-254	6	7	28	6	0.00	2	0.04	66.67
majority-239	6	8	36	6	0.00	3	0.06	50
mod5adder-128	6	15	120	7	0.00	3	0.16	57.14
2of5d1	6	18	171	15	0.00	5	0.05	66.67
mod5adder	6	21	231	10	0.00	4	0.06	60
hwb6tc	6	126	8001	40	0.03	8	0.67	80
rd53d1	7	12	78	11	0.00	2	0.11	81.81
2of5d2	7	12	78	12	0.00	2	0.13	83.33
ham7_105	7	21	231	15	0.00	3	0.13	80
ham7tc	7	24	300	15	0.01	4	0.17	73.33
rd53rcmg	7	30	465	23	0.01	4	0.11	82.60
rd53d2	8	12	78	12	0.00	2	0.10	83.33
cm82a-208	8	22	253	16	0.01	4	0.09	75
rd73-140	10	20	210	20	0.04	3	0.14	85
6symd2	10	20	210	19	0.04	3	0.15	84.21
9symd2	12	28	406	28	0.23	3	0.15	89.28
adr4-197	13	55	1540	47	0.82	6	0.11	87.23
0410184-169	14	46	1081	40	1.11	2	0.19	95
rd84-142	15	28	406	28	1.57	3	0.17	89.28
ham15-108	15	70	2485	53	3.60	8	0.20	84.90

(N), and the total number of faults for both SMGFs and MMGFs, respectively. Columns 5 and 7 in Table 4.2 present the number of test vectors required for detecting both SMGFs and MMGFs before and after minimization, respectively. The CPU time (sec) to generate the complete test set TS and minimal test set TS_{MIN} for the benchmark circuits are presented in Columns 6 and 8, respectively. Column 9 indicates the percentage of difference between the test set generated by the proposed method before and after minimization. From the reported results in Table 4.2, it is observed that the test set is minimized by more than or equal to 50% by the proposed minimization method for about 88% of the circuits, but still 100% fault coverage is retained. The maximum reduction of 95% of the test set for the reversible benchmark circuit 0410184-169 is achieved by our reduction technique. The minimum reduction of 33% of test set is observed for circuits $Fredkin$-6, ex-1-166, and $4gt$11-84.

The results of our proposed method are compared with the work of [5] and the comparison is shown in Columns 4, 5 and 11 of Table 4.3. The value '-' indicates that the results in the corresponding work [5] are not available. The authors in [5] proposed a greedy and BDD based covering method for generating the minimal test set for detecting all possible cases of two consecutive missing gate faults. The number of test patterns obtained in our proposed method is found to be less than the test patterns reported in [5]. Moreover, the method proposed in [5] can detect only two consecutive missing gate faults, whereas our proposed method can detect any number of consecutive missing gate faults. So, the fault coverage of our proposed method for multiple missing gate faults is more than that of [5]. The reduction of test set by our proposed test pattern generation method to detect MMGFs is more than or equal to 50% for more than 77% of the benchmark circuits that are used in the experiment. For the circuits $hwb6tc$ and 0410184-169, the test sets are reduced by 81.39% and 77.77%, respectively. For the circuits, $ex3_$ 229 and $rd32$-$v0_66$, the size of test sets are small [5] and so the scope of reducing the test set is less.

The authors in [6] proposed an ATPG algorithm to generate the complete test set using an exact approach for detecting missing gate faults along with single and multiple

Table 4.3: *Comparison of the complete test set with [5], [6], [7], [8], [9]*

Circuit	n	N	Number of Test Patterns							
			Gr. [5]	BDD [5]	[6]	Gr. [7]	B&B [7]	[8]	[9]	Proposed
			MMGF		MMGF	SMGF		MMGF	SMGF	SMGF+MMGF
ham3_102	3	5	4	4	2	2	2	2	3 or 4	2
3_17_13	3	6	4	4	2	2	2	2	2	2
rd32_v0_66	4	4	2	2	2	2	2	2	6	2
mini_alu_167	4	6	5	5	-	-	-	-	-	3
mod10_171	4	10	6	6	-	-	-	-	-	3
hwb4_52	4	11	6	6	-	-	-	-	-	2
4-49d3	4	12	-	-	-	-	-	-	4	3
4-49tc1	4	16	-	-	3	3	3	3	-	4
hwb4tc	4	17	-	-	4	2	2	4	7	3
Xor5d1	5	4	-	-	1	1	1	1	2	1
mod5d4	5	5	-	-	-	-	-	-	4	2
mod5d1-63	5	7	4	4	-	-	-	-	-	2
mod5d1	5	8	-	-	4	1	1	4	4	4
mod5d2	5	9	-	-	2	1	1	2	2	2
mod18-10-177	5	14	6	6	-	-	-	-	-	3
hwb5-55	5	24	12	11	-	-	-	-	-	4
hwb5-53	5	55	21	21	-	-	-	-	-	5
hwb5tc	5	56	-	-	5	5	5	5	-	5
graycode6_47	6	5	4	3	-	-	-	-	-	1
ex3_229	6	7	3	3	-	-	-	-	-	3
Xor5-254	6	7	3	3	-	-	-	-	-	2
majority_239	6	8	4	4	-	-	-	-	-	3
mod5adder_128	6	15	8	7	-	-	-	-	-	3
5mod5tc	6	17	-	-	6	1	1	6	-	6
2of5d1	6	18	-	-	5	4	4	5	-	5
mod5adders	6	21	-	-	4	3	3	4	-	4
hwb6tc	6	126	43	43	9	9	8	8	-	8
2of5d2	7	12	-	-	2	2	2	2	-	2
rd53d1	7	12	-	-	3	2	2	3	-	2
rd53_137	7	16	9	8	-	-	-	-	-	3
ham7_105	7	21	9	5	-	-	-	-	-	3
ham7tc	7	24	-	-	4	4	4	4	-	4
rd53rcmg	7	30	-	-	4	4	3	4	-	4
hwb7tc	7	291	-	-	14	15	4	14	-	12
rd53d2	8	12	-	-	2	2	2	2	-	2
cma82a_208	8	22	9	8	-	-	-	-	-	4
6symd2	10	20	-	-	3	2	2	2	-	3
rd73d2	10	20	6	-	3	3	3	3	-	3
9symd2	12	28	-	-	3	3	3	3	-	3
addr_197	13	55	13	-	-	-	-	-	-	6
0410184_169	14	46	9	-	-	-	-	-	-	2
rd84-142	15	28	8	-	3	3	3	3	-	3
ham15_108	15	70	26	-	8	-	-	-	-	8

stuck-at faults, repeated gate faults, and partial missing gate faults. The exact approaches aim to provide the optimal solution, but these approaches are computationally expensive (exponential complexity) for large circuits. Our proposed method generates a minimal test for covering all the single and multiple missing gate faults. From a computational point of view, our method requires linear time for obtaining the minimal test set. The comparison of our results with the results of [6] is shown in Columns 6 and 11 of Table 4.3. It is observed that the size of the test set generated by our proposed method is same as the size of test set generated by the method of [6] for most of the circuits, but the computational complexity of our proposed method is less.

The authors of the work [7] proposed two approaches, the first one is the greedy heuristic and the second one is based on exact branch and bound algorithm for generating the test vectors to detect the SMGFs. The method used for generating the test set in [7] is a DFT based approach which requires incorporation of additional testing circuits, so there is extra overhead in the hardware. The experiment results are presented and compared in Columns 7, 8 and 11, respectively of Table 4.3. It is observed that due to the use of DFT method, several gates are added to each circuit to detect the faults. Also, the work reported in [7] can detect only SMGFs, but our proposed method can detect both SMGFs and MMGFs. For most of the circuits, the size of test set produced by our method is comparable to the size of test given by the method of [7].

The authors in [8] used an exact automatic test patterns generation method for detecting SMGFs and MMGFs based on the integer linear programming. The comparison of our result with the result of [8] is reported in Columns 9 and 11 of Table 4.3. It is observed that size of test set is same in both the methods for most of the circuits, but the computational complexity of an exact automatic test pattern generation method is always more.

The experimental result in Columns 10 and 11 of Table 4.3 shows the comparison of the proposed work with the work done in [9]. The authors in [9] targeted for generating the test set for detecting only SMGFs using the method of Boolean difference generator. It is observed that the size of the generated test set is less in our proposed method

Table 4.4: *Fault coverage range with SAF, PMGF and Crosspoint Fault Models by the proposed complete test set TS*

Circuit	n	N	No. of Faults	No. of Faults	No. of Faults	Faults Coverage by TS [Proposed]	Faults Coverage by TS [Proposed]	Faults Coverage by TS [Proposed]
			SAF	PMGF	Crosspoint	SAF	PMGF	Crosspoint
4gt11_84	5	3	24	4	8	79.16%	50%	12.5%
xor5d1	5	4	26	4	12	76.92%	50%	58.33%
graycode6_47	6	5	32	5	20	87.50%	80%	50%
2of5d2	7	12	76	19	53	96.05%	89.47%	58.49%
3_17_13	3	6	32	7	5	100%	100%	100%
3_17_14	3	6	32	7	5	100%	71.42%	100%
fredkin_6	3	3	24	6	0	83.33%	50%	0
rd32_v0_66	4	4	28	6	6	89.28%	66.67%	50%
decode24_vo_38	4	6	36	8	10	94.44%	87.50%	40%
miller_11	3	5	32	8	2	100%	100%	100%
ham3_102	3	5	28	6	4	92.85%	83.30%	25%
ex_1_166	3	4	22	4	4	90.90%	75%	25%
nth_Prime3	3	4	24	5	3	100%	100%	100%
peres_9	3	2	16	3	1	50%	33.33%	0
decode24_v2_43	4	6	34	7	11	100%	100%	90.90%
mini_alu_167	4	6	52	16	2	100%	93.75%	100%
mod10_171	4	10	64	18	12	100%	94.40%	91.67%
4gt11-v1_85	5	4	26	4	12	80.76%	50%	16.67%
4mod5v1_24	5	5	32	6	14	87.50%	50%	42.85%
alu-v0_26	5	6	38	8	16	86.84%	50%	50%

for most of the benchmark circuits as compared to the method of [9]. The maximum reduction of 66.67% for the size of test set is found for the circuit $rd32d1$.

Finally, experiments are performed to check the fault coverage range of other fault models such as stuck-at faults (SAFs), partial missing gate faults (PMGFs) and crosspoint faults using the complete test set TS generated by our proposed method.

The total number of SAFs (SA0 and SA1) in a reversible circuit is given by $2(n + \sum_{i=1}^{N} g_i)$ [33], where g_i is the size of N gate of the circuit, and n is the total number of input wires. For determining the fault coverage range of PMGF fault model, the first-order PMGF is considered, i.e., only one control connection is missing at a time. The appearance crosspoint faults are considered in our experiments for crosspoint fault model. Table 4.4 presents the experiment results. The total number of SAFs, PMGFs and crosspoint faults are shown in Columns 4, 5 and 6, respectively. In Table 4.4, it is observed that SAFs, PMGFs and crosspoint faults coverage is 100% for the reversible benchmark circuits 3_17_13, *miller*_11, and *nth_prime*3 by the test set TS generated by our proposed method.

A total of 20 benchmark circuits are considered, out of these, only for the circuit *peres*_9 fault coverage of SAFs is 50%, and the fault coverage of SAFs is more than 75% for rest of the circuits. The fault coverage for PMGFs is 50% for 30% of the circuits and the fault coverage is more than 65% for rest of the circuits. In the crosspoint fault model, the highest deterioration of fault coverage is 12.50% for the circuit $4gt11$_84, and for the circuit *peres*_9, the test set TS is unable to detect any faults.

4.5 Conclusions

This work presented a scheme to generate the complete test set for detecting single and any number of consecutive multiple missing gate faults in the k-CNOT based reversible circuits. Experimental results show that the size of test set TS_{MIN} generated by our proposed method is smaller or similar as compared to the methods available in the literature and covers more faults by maintaining 100% fault coverage. The fault coverage range for the SAFs, PMGFs and appearance crosspoint faults with our generated complete

test set are also analyzed and checked using experimental results. In the next chapter, a fault localization method is presented to find the locations of missing gate faults by using the generated complete test set in this work to detect the missing gate faults.

Chapter 5

Fault Localization for Missing Gate Faults in Reversible Circuits

5.1 Introduction

In this chapter, the fault localization method is presented to obtain the exact location of single and multiple missing gate faults in reversible logic circuits. The primary focus of the proposed method is to extract the unique test pattern for each missing gate faults in the k-CNOT based reversible circuit. The complete test set generation methods to detect single and multiple missing gate faults are presented in the previous chapter. With the help of this generated complete test set, the unique test patterns are constructed for fault localization. The unique sequences of test patterns are used for identifying the exact location of the faults. Based on these unique set of test patterns a fault localization tree is constructed. Next, the traversal process is applied to the fault localization tree to determine the presence and location of the faults. Moreover, the proposed method is also capable of evaluating the equivalent faults in a given reversible circuit. Finally, we provide our experimental results that are verified on several reversible benchmark circuits.

At first, we generate the complete test set for detecting the missing gate faults and the input test set is derived from the complete test set which is applied to the circuit for obtaining the output responses of the fault-free and the faulty circuit. Now, the fault analysis table is constructed with the help of the output responses of each possible faults that may occur in the circuit. Based on the fault analysis table, the fault localization

tree is built, and the reduction process is applied to obtain the final fault localization tree which contains the exact unique output responses for each of the single and multiple missing gate faults along with the output responses of the fault-free circuit. Finally, the traversal process is applied to the fault localization tree to identify the location of faults in the reversible circuit. The traversal process of the fault localization tree is simple yet effective, and takes linear time to identify the exact location of the fault.

The rest of the chapter is organized as follows: Section 5.2 provides a discussion on related work on fault localization in reversible circuits. In Section 5.3, we describe our proposed fault localization method with the help of detailed illustrations. The experimental results and concluding remarks are presented in Section 5.4 and 5.5, respectively.

5.2 Related Work

Some of the prior works that are relevant to the work presented in this chapter are briefly reviewed in this section. In 2004, Ramasamy et al. [47] proposed an adaptive tree-based approach on functional test based fault localization to detect and locate stuck-at faults in a reversible circuit. In the proposed method, the symmetric adaptive tree is constructed by generating a standard complete and static fault table using all possible input vectors, which help to detect the reason of faulty behavior in the circuit. To speed up the computation of the symmetric adaptive tree method, the authors also proposed a new greedy direct tree generation algorithm that eliminates the fault table generation and dynamically creates the adaptive fault tree without generating all tests. However, both approaches are applied to small circuits because the simulation size of the fault table increases exponentially for large circuits. In 2005, Pierce et al. [48] also proposed a method to generate the fault table using all the possible vectors in the circuit and based on the fault table. The fault localization tree is constructed with adding some constraints for identifying the faulty behavior of the circuit. To handle the larger index fault table, the authors suggested a method of storing a fault table with decision diagrams which required less amount of memory as compared to storing a fault table as an array. In 2009, Wille et al. [56] proposed the first approach for

automatic debugging of TOFFOLI network based reversible circuit and the proposed debugging approach was formulated based on Boolean satisfiability. In this approach, given an erroneous circuit and a set of counter-examples describing the error(s) of the circuit, it returns a set of gates in the form of the error candidates, which are replaced with other gates to fixed the counter-examples. In 2010, Jung et al. [57] introduced another debugging method for multiple missing control errors in reversible circuits. The scheme used the concepts of the previous method [56] i.e., correction-based debugging and hitting set-based debugging. In the hitting set-based approach, a new notion of error candidates is introduced, and this notion is applied to encode the determination of error candidates. The authors also proposed that the combination of correction-based and hitting set-based approaches reduces the number of error candidates that lead to a improvement in the accuracy of an error location. In 2011, Zamani et al. [58] introduced the fault masking technique using the two majority voters with different error masking capabilities. These majority voters provide the information about the location of the fault for online and as well as offline diagnosis. In 2011, Zhang et al. [49] proposed a new fault diagnosis approach for single missing control fault model in reversible circuits. This approach provided new methods for Diagnostic Test Pattern Generation (DTPG) and fault equivalent checking. Again, in 2011, Rahaman et al. [50] proposed a fault diagnosis technique for k-CNOT based reversible circuit for the missing gate fault model. The proposed technique requires a universal test set (UTS) of size (n+1) that detects all single missing-gate faults (SMGFs), repeated-gate faults (RGFs), and partial missing-gate faults (PMGFs) with the addition of one extra control line along with duplication of each k-CNOT gate of the original $(n \times n)$ k-CNOT reversible circuit. Furthermore, the single missing gate fault (SMGF) is identified for each gate using a single test vector that is applied to an augmented circuit. In 2014, Mondal et al. [59] proposed a fault diagnosis technique in reversible circuits to extract the unique fault signature for each missing control fault. Based on the unique fault signature, a unique fault diagnosis tree is constructed, and the proposed fault diagnosis algorithm is applied to traverse the fault diagnosis tree for identifying the location of the fault.

In 2014, the authors in [60] proposed a fault diagnosis technique for single missing gate fault (SMGF), multiple missing gate fault (MMGF), and partial missing gate fault (PMGF) using the Multi-Input Signature Analyzer (MISA) in the form of successive tree structure. Pseudorandom test vectors are applied to the fault-free and faulty circuit for generating the output responses to generate the one-dimensional fault signature. The verification of the signature with the help of the circuit under test is performed for fault diagnosis. In 2016, Mondal et al. [61] developed a scheme to detect and localize the faulty zone for single missing gate fault (SMGF) and partial missing control fault (PMGF). The proposed scheme scanned the entire augmented circuit, block-wise from left to right and compared the input and output vector in a block-wise manner. If the input and output vectors do not match in a given block, then the faulty gate is present in that block. In 2017, the authors in [62] have proposed a similar approach for finding the location of fault as mentioned in [61], but the proposed approach is applied for new types of faults such as gate appearance fault and control appearance fault in the k-CNOT based reversible circuit.

Based on the discussion of the above literature review, we observe that most of the existing fault diagnosis methodologies have considered the diagnostic tree using the fault table generation with or without generating all test patterns. The scope of the efficient input test vectors from the complete test set (for detecting the faults) to build the fault diagnostic tree is limited in reversible circuits. Moreover, many of the approaches use a single test vector with additional circuit overhead to detect the location of the fault. In this work, we target both single and multiple missing gate fault localization in a reversible circuit designed with k-CNOT reversible gates without any additional circuit overhead, and fault localization tree is built by the unique output response of the complete test set for SMGF and MMGF instead of using all possible test vectors or pseudorandom test patterns.

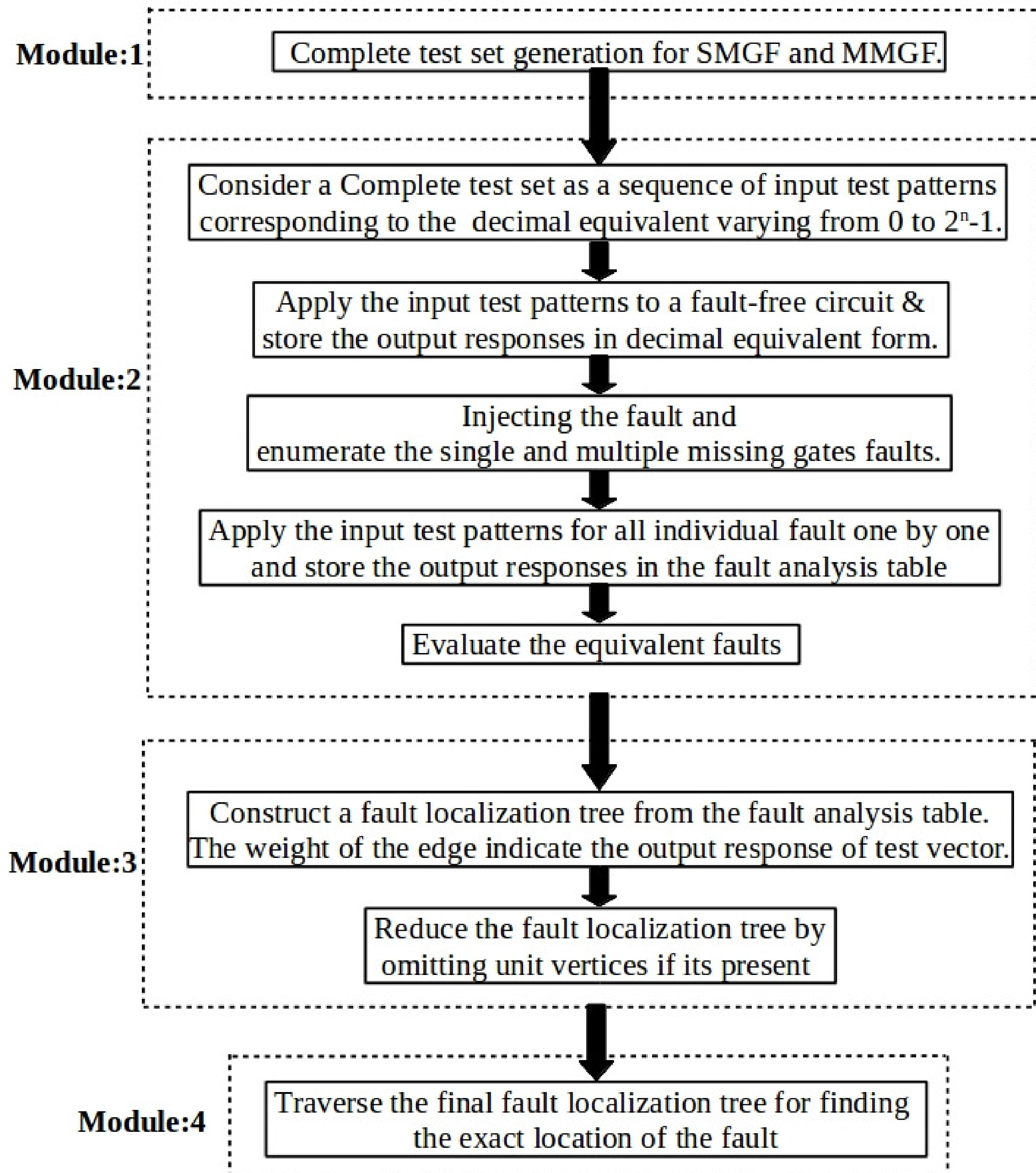


Figure 5.1: *Block diagram of Fault Localization Method*

5.3 Proposed Technique of Fault localization

In this section, we provide a fault localization technique to determine the exact location of faults under the missing gate fault model in reversible circuits. The proposed fault localization method that has four modules. The first module involves generating the complete test set for SMGF and MMGF in the reversible circuit. The second module describes the construction of a fault analysis table using the complete test set for SMGF and MMGF as a sequence of input test patterns in a given reversible circuit. The sequence of input test patterns are fed to the fault-free circuit, and the corresponding output test patterns are collected and stored. Then, the fault is injected to the fault-free circuit, and the same sequence of input test patterns are applied to the faulty circuit of each enumerated fault. The generated sequence of output test patterns for each enumerated fault in the faulty circuit is compared to the output test responses of the fault-free circuit. The comparison process provides the information of unique output responses for each enumerated fault in a given reversible circuit. The entire information is stored in the fault analysis table. The fault analysis table helps to detect if any equivalent faults are present in the circuit. In the third module, based on the fault analysis table, we construct the fault localization tree, and the reduction process is applied to obtain the minimized final fault localization tree. The minimized fault localization tree provides the exact unique identification of the sequence of output test patterns for each possible enumerated fault along with the sequence of output test patterns in the fault-free circuit. In the fourth module, the traversal process is applied to the final fault localization tree to identify the location of single and multiple missing gate faults. Figure 5.1 shows the modules of the proposed scheme and the detail discussion of each module is presented in the following sections.

5.3.1 Module 1: Complete Test Set Generation

In this section, we first consider the complete test set TS for detecting the single and multiple missing gate faults, as mentioned in Chapter 4. In our fault localization scheme, the unique output response is required for identifying the location of faults in a given

reversible circuit. For this purpose, the generated complete test set TS referred to as an input test set which is applied to the circuit for obtaining the output responses of the fault-free and the faulty circuit. Later on, these generated unique output responses are used to extract the location of single and multiple missing gate faults in reversible circuits.

5.3.2 Module 2: Test Response Generation for Fault Localization

Test pattern generation is an essential phase to distinguish among all the possible faults in a given reversible circuit. In our fault localization method, we consider the input test patterns that are fed to the reversible circuit and the generated corresponding output test responses are used to determine the fault location.

Definition 5.3.1. *The set of input patterns* $T_{in}=\{TV_1, TV_2 \ \ldots, TV_k\}$ *is obtained from the complete test set* TS *that is applied to the faulty (fault-free) circuit for observing the faulty (fault-free) output responses. For simplicity purpose, we order each test vector* $TV_i \in T_{in}$ *corresponding to the decimal values varying from 0 to* 2^n*-1.*

It may be the case that T_{in} is not sufficient to distinguish all faults uniquely. In that situation, T_{in} is appended by remaining p number of test vectors, where $p=|T_{in}|$ for n-input lines, one by one in sequential order until the generation of the unique sequence of output test patterns for all possible faults is completed. Considering this, the upper bound on input test set T_{in} size becomes $|TS| + p$.

Definition 5.3.2. *The set of output test patterns* $T_{out}=\{TV_{o1}, TV_{o2} \ \ldots, TV_{ok}\}$ *is generated by applying in test set* T_{in} *in a given reversible circuit. More precisely,* $TV_i \Rightarrow TV_{oi}$*, where* $1 \leq i \leq k$*,* $TV_i \in T_{in}$*, and* $TV_{oi} \in T_{out}$*. The set* T_{out} *determines the unique fault location of a particular fault present in the circuit. Similarly, we consider each test vector* $TV_{oi} \in T_{out}$ *corresponding to the decimal values varying from 0 to* 2^n*-1.*

Consider a reversible benchmark circuit $Miller_11$ as shown in Figure 5.2 and the truth table of this circuit is shown in Table 5.1. As per Algorithm 3 in Chapter 4, the complete test set of the $Miller_11$ circuit is TS={101, 110, 001, 111}. Thus, the TS is considered as the set of input test patterns T_{in}={001, 101, 110, 111}. Now, we apply the T_{in} to the fault-free $Miller_11$ circuit and the corresponding output responses T_{out} are obtained, as illustrated in Table 5.2.

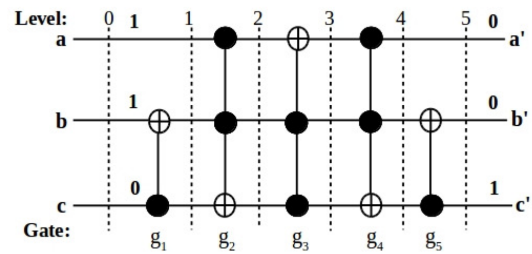


Figure 5.2: *Miller_11 benchmark circuit*

Table 5.1: *Truth Table of the circuit Miller_11*

Input			Output		
a	b	c	a′	b′	c′
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	1	1	1

Table 5.2: *Input and corresponding output responses for the fault-free circuit $Miller_11$*

T_{in}	Decimal equivalent form of T_{in}	T_{out} for fault free condition	Decimal equivalent form of T_{out}
001, 101, 110, 111	1, 5, 6, 7	011, 101, 001, 111	6, 5, 1, 7

5.3.2.1 Enumeration of SMGFs and MMGFs

Here, we consider the enumerated value as $fault_{free}$ in a given fault-free circuit. Based on a given circuit structure, we enumerate all single and multiple missing gate faults. Next, we assign the enumerated value to all the possible single and multiple missing gates at their corresponding levels in the circuit. Suppose, a reversible circuit consist of a linear cascade of k gates $\{g_1, g_2, \ldots, g_k\}$ and the levels are $\{l_0, l_1, \ldots, l_k\}$, for every gate g_i, where $i = 1$ to k. The assigned value $f_{g_i}^{(l_{i-1}, l_i)}$ for single missing gate fault is denoted as the ith faulty gate (missing) lies between the $(i-1)^{th}$ and i^{th} level in the circuit. For two consecutive multiple missing gates fault, the assigned value $f_{(g_i, g_j)}^{(l_{i-1}, l_j)}$ is denoted for faulty gate g_i and g_j $(i < j)$ are in between the $(i-1)^{th}$ and j^{th} level in the circuit. The enumeration process for various consecutive gate(s) is missing of the $Miller_11$ benchmark circuit is depicted in Figure 5.3.

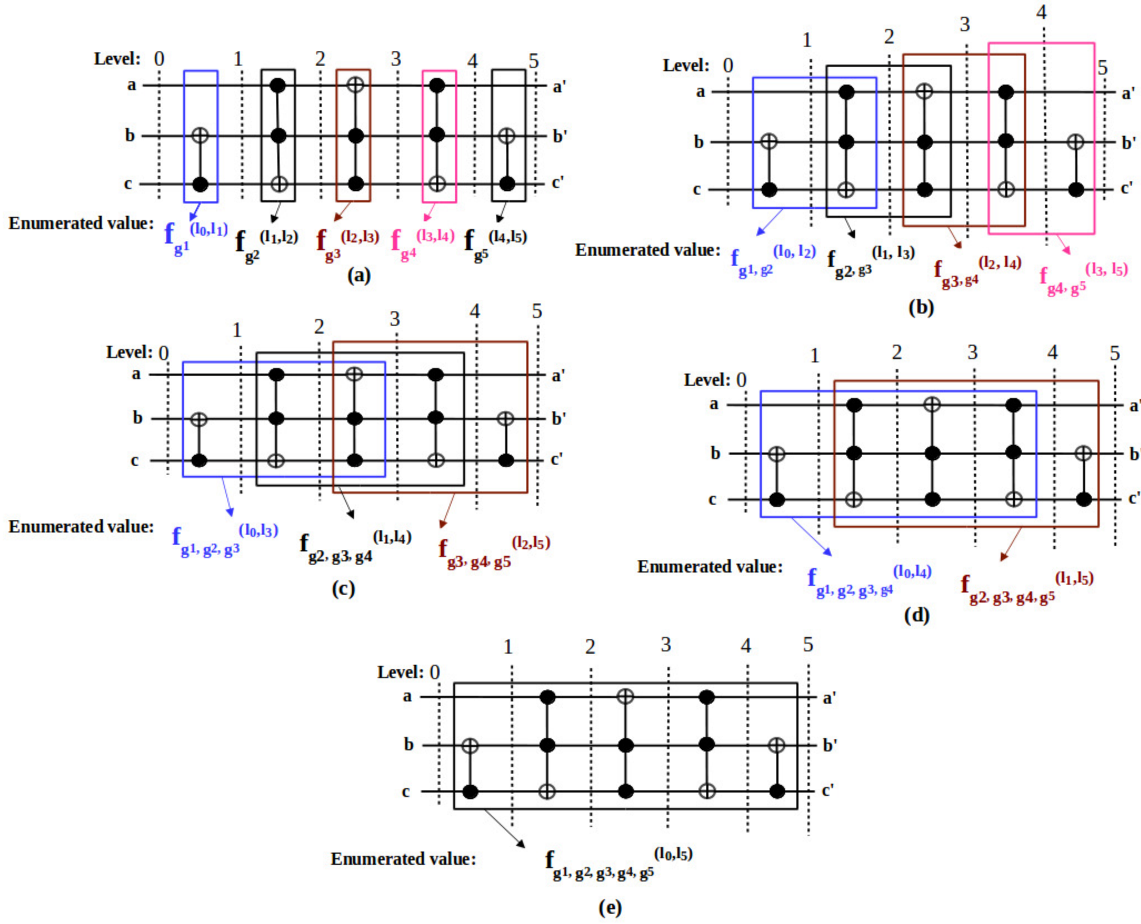


Figure 5.3: *Enumeration Process for the circuit Miller_11: (a) one gate (b) consecutive two gates (c) consecutive three gates (d) consecutive four gates, and (e) consecutive five gates*

5.3.2.2 Construction of Fault Analysis Table

For the construction of fault analysis table, we apply the set of input test patterns T_{in} for all individual enumerated faults one by one present in the circuit and store the set of output test patterns T_{out}. The fault analysis table determines the unique output responses for each fault from the output responses of a given fault-free circuit. If T_{in} of a given circuit generates the same output test patterns contained in set T_{out} for the two or more faults that may occur in the faulty circuit, then the remaining test vectors are appended one by one by maintaining a sequential order until the generation of the unique sequence for all the possible faults is complete. This process also leads

to evaluation of functionally equivalent faults. The fault analysis table for $Miller_11$ benchmark circuit is shown in Table 5.3. The left and right column in Table 5.3 present all possible enumerated faults and set of output test patterns T_{out}, respectively.

5.3.2.3 Evaluating Equivalent Faults

Two faults f_i and f_j are functionally equivalent if the applied test sequence of input test patterns produces the same test sequence of output test patterns by the two faulty circuits. In another word, there is no sequence of input test patterns that can distinguish between two faults f_i and f_j with their respective output responses. In this work, we consider the functional equivalent faults to imply equivalence under the input test set T_{in}.

Definition 5.3.3. *The faults f_i and f_j are said to be functionally equivalent faults (FEFs) under an input test set T_{in} iff $f_i(TV_i)$=$f_j(TV_i)$ for every test vector $TV_i \in T_{in}$.*

In Table 5.3, it is observed that the fault set $\{f_{g_3}^{(l_2,l_3)}, f_{(g_2,g_3,g_4)}^{(l_1,l_4)}, f_{(g_1,g_2,g_3,g_4,g_5)}^{(l_0,l_5)}\}$ produces the same output test patterns T_{out}={1,5,6,7}. Similarly, the fault set $\{f_{(g_2,g_3)}^{(l_1,l_3)}, f_{(g_3,g_4)}^{(l_2,l_4)}\}$ and $\{f_{(g_1,g_2,g_3,g_4)}^{(l_0,l_4)}, f_{(g_2,g_3,g_4,g_5)}^{(l_1,l_5)}\}$ produce the same output test patterns T_{out}={1,6,5,7} and T_{out}={3,7,6,5}, respectively. As per the fault analysis table, by appending the remaining test vectors {0,2,3,4} one by one to the set of input test patterns T_{in}={1,5,6,7} we obtain the unique output test patterns. The evaluation of equivalent fault process is shown in Table 5.4. Since the output responses match the respective fault set while considering the input test patterns {1,5,6,7,0,2,3,4}, therefore the above-mentioned fault sets are taken as the functionally equivalent faults in the $Miller_11$ benchmark circuit.

5.3.3 Module 3: Construction of Fault Localization Tree

The fault localization tree is a simple directed graph. The fault localization tree generates the unique path for the occurrence of each fault in a circuit, and also provides the unique path of a fault-free circuit. The fault localization tree is constructing with the help of a fault localization table. The construction process of a tree starts from the root vertex denoted as V_0. To create the new vertices in the next level, we are considering the output test pattern TV_{oi} as the weight of the directed edge for each row of the first

Table 5.3: *Fault analysis table for the circuit* $Miller_11$

Enumerated fault-free/faulty value	Decimal equivalent form of T_{out}
$fault_{free}$	6, 5, 1,7
$f_{g_1}^{(l_0,l_1)}$	3, 7, 1, 5
$f_{g_2}^{(l_1,l_2)}$	6, 1, 5, 7
$f_{g_3}^{(l_2,l_3)}$	1, 5, 6, 7
$f_{g_4}^{(l_3,l_4)}$	5, 6, 1,7
$f_{g_5}^{(l_4,l_5)}$	6, 7, 3, 5
$f_{(g_1,g_2)}^{(l_0,l_2)}$	3, 7, 5, 1
$f_{(g_2,g_3)}^{(l_1,l_3)}$	1, 6, 5, 7
$f_{(g_3,g_4)}^{(l_2,l_4)}$	1, 6, 5, 7
$f_{(g_4,g_5)}^{(l_3,l_5)}$	7, 6, 3, 5
$f_{(g_1,g_2,g_3)}^{(l_0,l_3)}$	3, 7, 5, 6
$f_{(g_2,g_3,g_4)}^{(l_1,l_4)}$	1, 5, 6, 7
$f_{(g_3,g_4,g_5)}^{(l_2,l_5)}$	3, 6, 7, 5
$f_{(g_1,g_2,g_3,g_4)}^{(l_0,l_4)}$	3, 7, 6, 5
$f_{(g_2,g_3,g_4,g_5)}^{(l_1,l_5)}$	3, 7, 6, 5
$f_{(g_1,g_2,g_3,g_4,g_5)}^{(l_0,l_5)}$	1, 5, 6, 7

Table 5.4: *Evaluation of equivalent faults for the circuit Miller_11*

T_{in}	Equivalent faults	T_{out}
1, 5, 6, 7, 0, 2, 3, 4	$f_{g_3}^{(l_2,l_3)}$	1, 5, 6, 7, 0, 2, 3, 4
	$f_{(g_2,g_3,g_4)}^{(l_1,l_4)}$	1, 5, 6, 7, 0, 2, 3, 4
	$f_{(g_1,g_2,g_3,g_4,g_5)}^{(l_0,l_5)}$	1, 5, 6, 7, 0, 2, 3, 4
	$f_{(g_2,g_3)}^{(l_1,l_3)}$	1, 6, 5, 7, 0, 2, 3, 4
	$f_{(g_3,g_4)}^{(l_2,l_4)}$	1, 6, 5, 7, 0, 2, 3, 4
	$f_{(g_1,g_2,g_3,g_4)}^{(l_0,l_4)}$	3, 7, 6, 5, 0, 2, 3, 4
	$f_{(g_2,g_3,g_4,g_5)}^{(l_1,l_5)}$	3, 7, 6, 5, 0, 2, 3, 4

entries of the second column in the fault localization table. For each internal vertices, including with the root vertex, stores the input test pattern $TV_i \in T_{in}$ based on their corresponding output test pattern. Similarly, this process continues until all the other subsequent entries are covered from the second column in the fault localization table to form the new vertices of their weighted edges. In fault localization tree, the root vertex and internal vertices are represented by circles, where squares represent the leaf nodes attached with enumerated fault-free and faulty values for identifying the exact location of the faulty or fault-free circuit.

The fault localization tree for *Miller_11* benchmark circuit is shown in Figure 5.4. Here, Table 5.3 shows the total number of fifteen output test patterns for each corresponding fault. Due to the presence of equivalent faults, we obtain only twelve unique output test patterns including the fault-free circuit for constructing the fault localization tree. The first entries of the output test patterns of the second column in Table 5.3 are 1, 3, 5, 6, and 7 which are considered as weighted directed edges to form the new vertices namely V_1, V_2, V_3, V_4, and V_5, respectively in the next level of the fault localization tree. Here, the root vertex V_0 stores the input test pattern 1 (denoted as V_0:1) and it helps

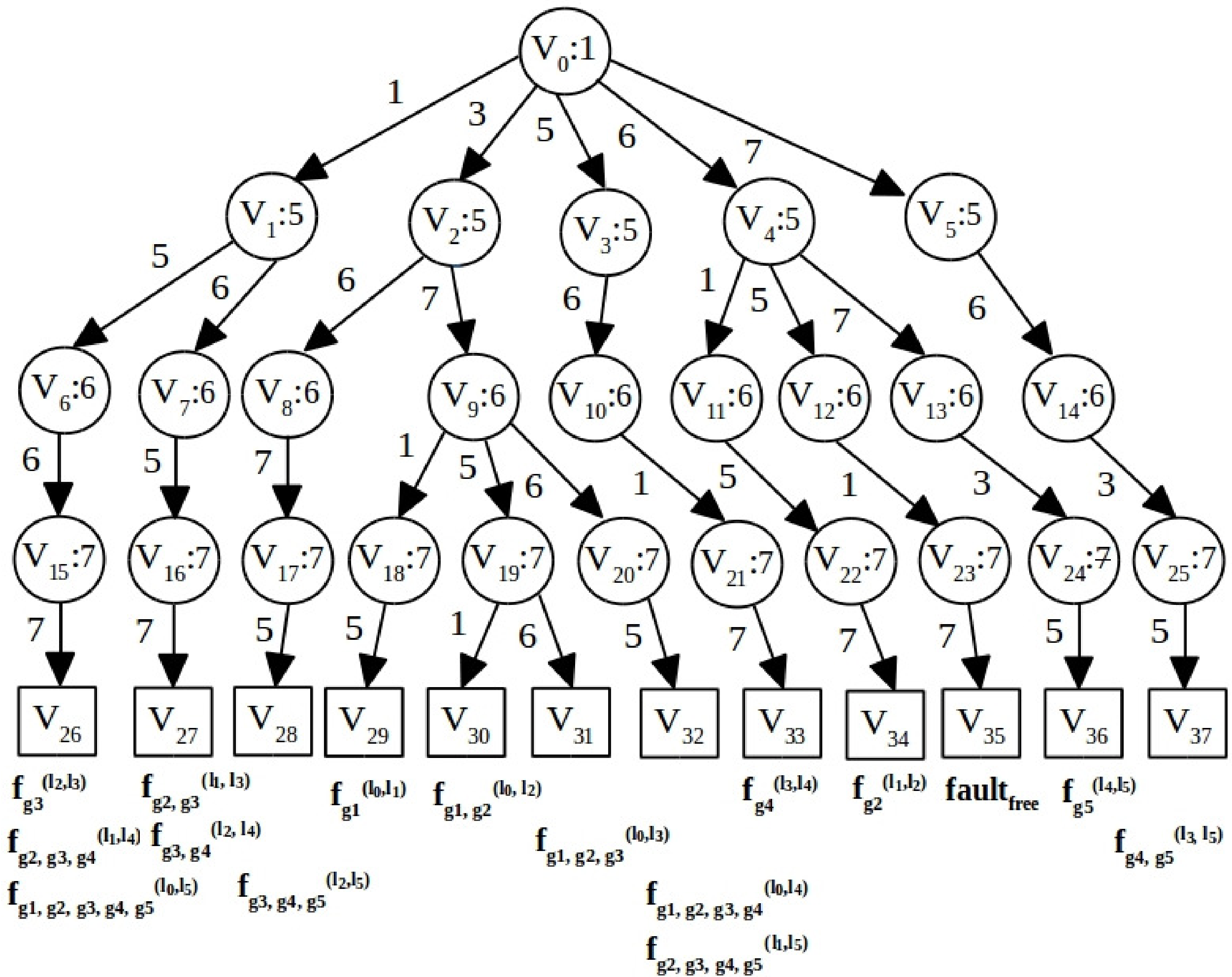


Figure 5.4: *Fault Localization Tree for the circuit Miller_11*

to keep track of all possible output test patterns as a weighted edge. Now, consider the subsequent output test patterns of each row in the fault localization table to create new vertices and weighted directed edge in the next level of the fault localization tree. As a result, the new vertices are $V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}$, and V_{14} and the corresponding output test patterns as directed with weighted values are 5, 6, 6, 7, 6, 1, 5, 7, and 6, respectively. Similarly, based on the unique output test patterns of the fault localization table, the entire tree is constructed with 38 vertices.

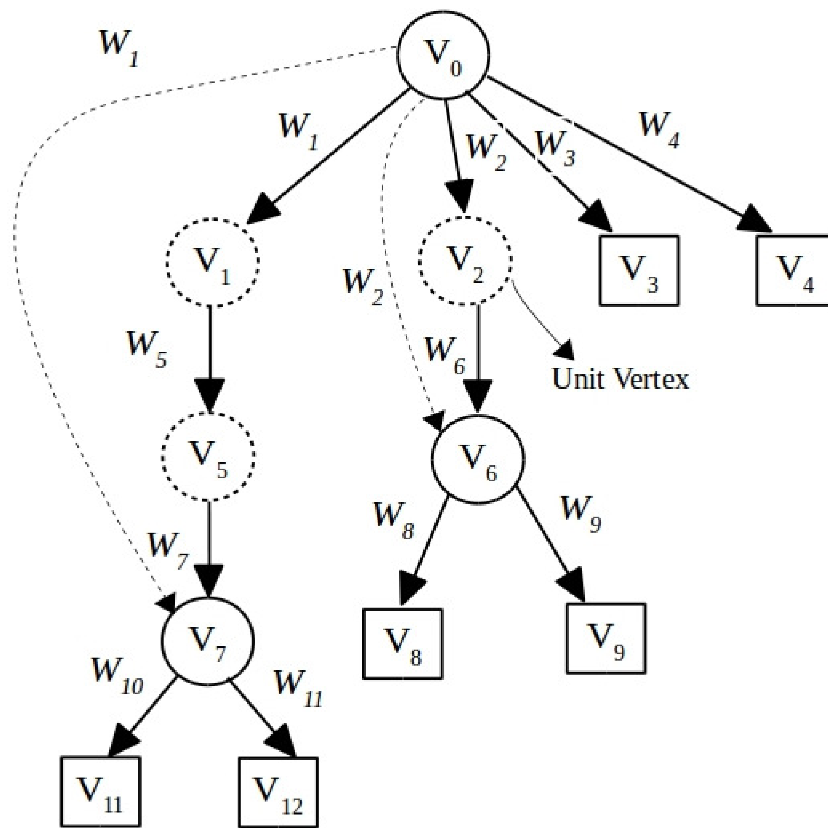


Figure 5.5: *Reduction Process of a given tree*

5.3.3.1 Reduction Process for Fault Localization Tree

The reduction process of fault localization tree involves removing all the unit vertex presence in the tree. If the number of edges incident out of a vertex V_i is only one, then it is called a unit vertex. More precisely, the vertex V_i is called unit vertex if it contains $d^+(V_i)$=1. Let us consider the directed edge e_i that connects the vertex V_i to V_j, and vertex V_j to V_d is connected through the directed edge e_d. Thus, the unit vertex V_j and the edge e_d incident out of a vertex V_j are deleted and the process establishes the connection of V_i to V_d through the edge e_i.

Example 5.3.1. To explain the reduction process, we consider the fault localization tree as depicted in Figure 5.5. The tree contains 12 vertices including the root vertex. Here, each incident edges W_i represent the output test patterns. In Figure 5.5, we observed that the in-degree and out-degree of the vertices V_1, V_2 and V_5 are only 1. Thus, we considered these vertices are unit vertices and are represented by dotted circles. After detecting the unit vertices, we remove all the unit vertices, and now, the adjacent vertex for V_0 is V_7 through the incident edge W_1. Similarly, the adjacent vertex for V_0 is V_6 through the incident edge W_2. Hence, the total number of vertices is 9, and the number of levels of the tree is reduced by 2 after the reduction process.

As per our fault localization technique, the reduction process is not required when no unit vertices are present in the fault localization tree. In fault localization tree for $Miller_11$ benchmark circuit, there are unit vertices namely V_3, V_5, V_6, V_7, V_8, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{20}, V_{21}, V_{22}, V_{23}, V_{24} and V_{25}, which are represented as dotted circles in Figure 5.6. Using the reduction process the vertex V_3 is deleted in level-1 and subsequently the vertices V_{10}, and V_{21} are also deleted. Then, the incoming weighted edge of V_3 is directly connected to vertex V_{33}. Thus, two vertices V_0 and V_{33} are adjacent. Similarly, the same reduction process applies to the remaining unit vertices. After the reduction process, the fault localization tree of the $Miller_11$ circuit contains only 18 vertices as depicted in Figure 5.7. The reduced fault localization tree of the $Miller_11$ circuit with test vectors in binary form is shown in Figure 5.8.

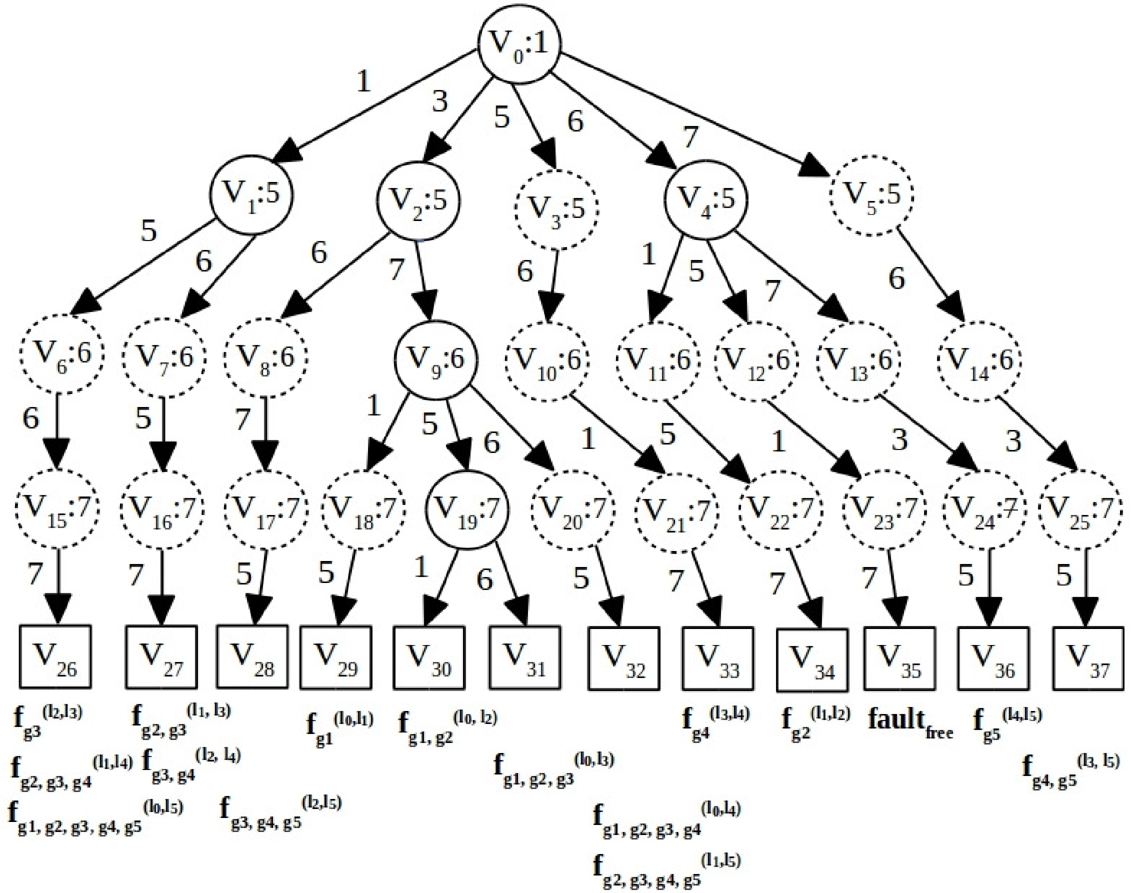


Figure 5.6: *Unit vertices of a fault localization tree for the circuit Miller_11 during the reduction process*

Property 5.3.1. The reduction process of fault localization tree is only possible if the sequence of vertices is monotonically increasing.

Proof. Let us assume that the sequence of vertices $\{V_0, V_1, \ldots, V_k\}$ is monotonically increasing and terminating in leaf node V_k. To maintain the monotonically increasing sequence of vertices, at least one vertex V_i for $0 < i < k$ must have in-degree and out-degree of 1 (i.e., $d^+(V_i) = d^-(V_i)$=1). Therefore, the vertex V_i is considered as unity vertex. If unity vertex is present, then the reduction process can be applied to the fault localization tree. □

5.3.4 Module 4: Traversal Process for Fault Localization

The reduced fault localization tree is sufficient enough to identify the exact location of the presence of fault(s) based on the applied input sequence of test patterns in the

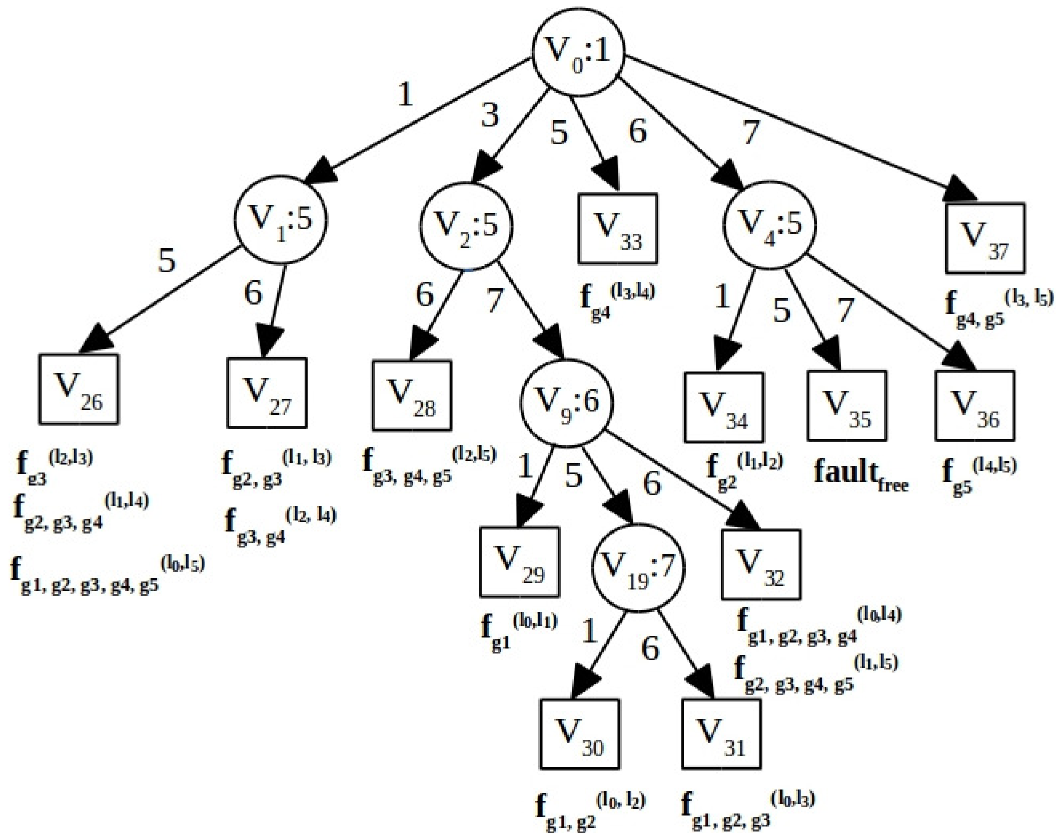


Figure 5.7: *Fault localization tree after the reduction Process for the circuit $Miller_11$*

reversible circuit. At first, we consider the root vertex V_0 in level-0 at the reduced fault localization tree. The next vertex (V_i) in level-1 is visited through the least weighted valued directed edge (W_k) as an output test pattern TV_{oi}. This output test pattern TV_{oi} is generated by the input test pattern TV_i of root vertex V_0 such that TV_i determines the W_k that is matched with the output test pattern TV_{oi} of a given circuit. If we find that the vertex V_i in level-1 is a leaf vertex then the attached value indicates the location of the fault/fault-free gate in a given circuit. If V_i is not leaf vertex then we move to the adjacent vertex V_j in level-2 through the least weight valued of directed edge (W_k) such that TV_i of vertex V_i determines the W_k that matches with the output test pattern TV_{oi} of a given circuit. The same process continues until we get the leaf vertex of a reduced fault localization tree.

Example 5.3.2. Let us consider that the gate g_2 is missing in the $Miller_11$ circuit

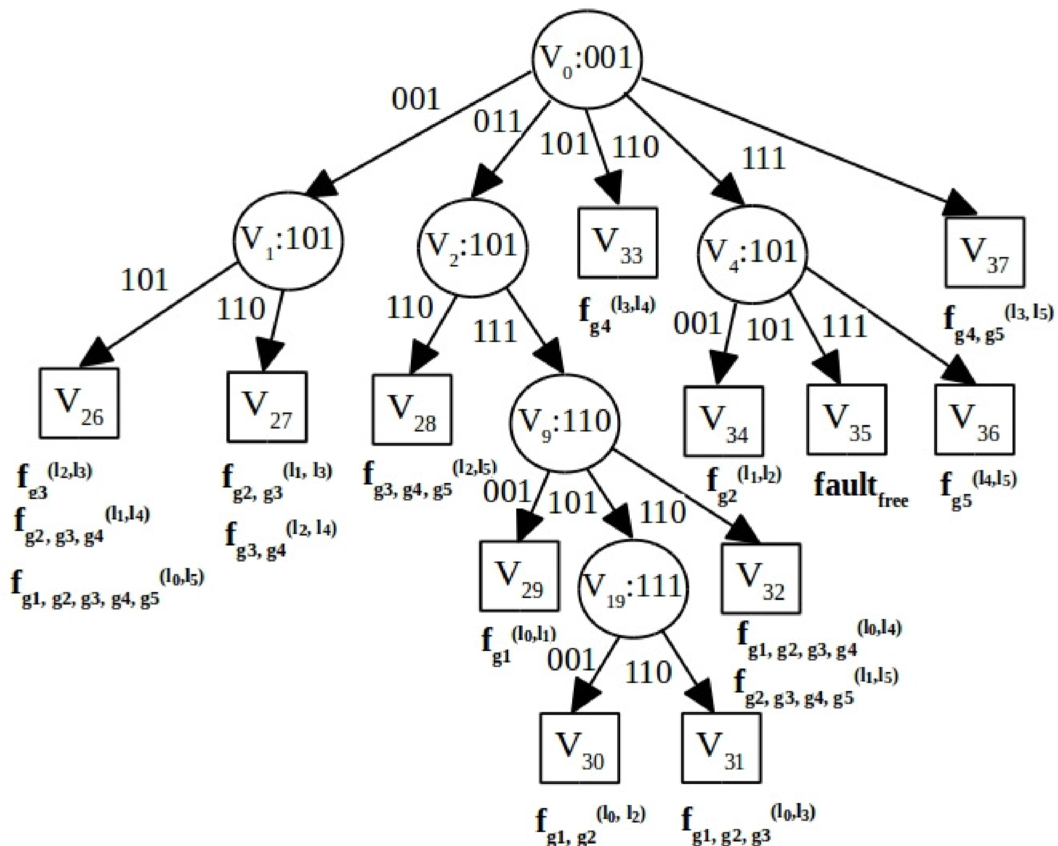


Figure 5.8: *Reduced fault location tree for the circuit $Miller_11$ with test vectors in binary form*

and the enumerated value of this fault is $f_{g_2}^{(l_1,l_2)}$ i.e., the gate g_2 is missing between level l_1 and l_2 as shown in Figure 5.3 (a). Figure 5.9 illustrates the traversal process in bold lines. At first, we start the root vertex V_0 which has 5 incident edges. Now, we choose the least weighted valued incident edge, i.e., 001, and it determines by the input test pattern 001 that is associated with the root vertex V_0. Here, the input test pattern 001 does not match with the output test pattern 001 ($001 \nRightarrow 001$) as the $2nd$ gate is missing. Therefore, we choose the next least weighted valued incident edge 011, and it also does not match with the output test pattern ($001 \nRightarrow 011$) of the faulty $Miller_11$ circuit. Similarly, the $3rd$ least weight valued incident edge 101 does not match with the output test pattern ($001 \nRightarrow 101$) in the faulty circuit. Then, we choose the $4th$ least weighted valued incident edge 110 and the input test pattern 001 matches with the output test pattern 110 ($001 \Rightarrow 110$) of the faulty $Miller_11$ circuit. So, the adjacent vertex of V_0 is V_4 through the incident edge 110. The vertex V_4 has 3 incident edges. Again, we choose the least weight valued incident edge 001 of V_4, and it is determined by the input test pattern 101 associated with the vertex V_4. Here, the input test pattern 101 matches with the output test pattern 001 ($101 \Rightarrow 001$) of a given faulty circuit. Next, the adjacent vertex of V_4 is V_{34}, and it is the leaf vertex. Thus, we extract the enumerated value

Algorithm 4: Fault localization algorithm.

1. Visit the root vertex V_0.
2. Initially, choose the least weighted valued incident edge (W_k) of V_0, which determines by the input test pattern TV_i of V_0.
3. Repeat step 4 until W_k=0.
4. If $TV_i \Rightarrow W_k$ matched with the output test pattern of a given circuit then visit adjacent vertex V_i and set W_k=0.
   Else choose the next least weighted valued incident edge (W_m) of V_0 such that $W_m > W_k$ and set W_k=W_m.
5. If V_i is the leaf vertex then write attached enumerated value of V_i & exit.
6. Choose the least weighted value incident edge (W_k) of V_i, which determines by the input test pattern TV_i of V_i.
7. Repeat step 8 until W_k=0.
8. If $TV_i \Rightarrow W_k$ matched with the output test pattern of a given circuit then visit adjacent vertex V_j and set W_k=0.
   Else choose the next least weighted valued directed edge (W_m) of V_i such that $W_m > W_k$ and set W_k=W_m.
9. Set V_i=V_j and goto step 5.

$f_{g_2}^{(l_1,l_2)}$ attached with vertex V_{34} which determines that gate g_2 is missing in between level l_1 and l_2.

Example 5.3.3. Consider the gate g_3 and g_4 are missing in $Miller_11$ circuit as shown in Figure 5.3 (b). The bold lines in Figure 5.10 show the traversal process. We start with the root vertex V_0, and the least weighted valued incident edge is 001. The corresponding input test pattern 001 matches with the output test pattern 001 (001 $\Rightarrow$ 001) of a given faulty circuit. Therefore, we visit the adjacent vertex V_1 and the vertex V_1 has 2 incident edges. Now, we choose the least weight valued incident edge 101, but the input test pattern 101 of the vertex V_1 does not match with the output test pattern 101 (101 $\not\Rightarrow$ 101) in the faulty circuit. Next, we choose the next least weight valued incident edge 110, where the input test pattern 101 matches with the output test pattern 110 (101 $\Rightarrow$ 110) in the faulty circuit. Therefore, we visit the adjacent vertex V_{27} which is the leaf vertex, and this stops the traversing process. The enumerated fault values are $f_{(g_2,g_3)}^{(l_1,l_3)}$ or $f_{(g_3,g_4)}^{(l_2,l_4)}$, and they are functionally equivalent faults by the input test set T_{in}.

Definition 5.3.4. *The input test set T_{in} is to be considered as a complete fault location test set (CFLTS) if it can distinguish among all the distinguishable faults and identify the functionally equivalent faults (FEFs) under the test set T_{in} in the circuit.*

Example 5.3.4. A CFLTS of the $Miller_11$ circuit in Figure 5.2 is {001, 101, 110, 111}.

Property 5.3.2. The generated path of the weighted incident edge from root vertex (V_0) to the leaf vertex uniquely identifies the exact location of the fault.

Proof. In fault localization tree, the outgoing edges from the root vertex (V_0) represent the output responses for each fault. The sequence of output responses is unique for each fault (except FEFs) as per the fault localization table. Each pair of vertices includes

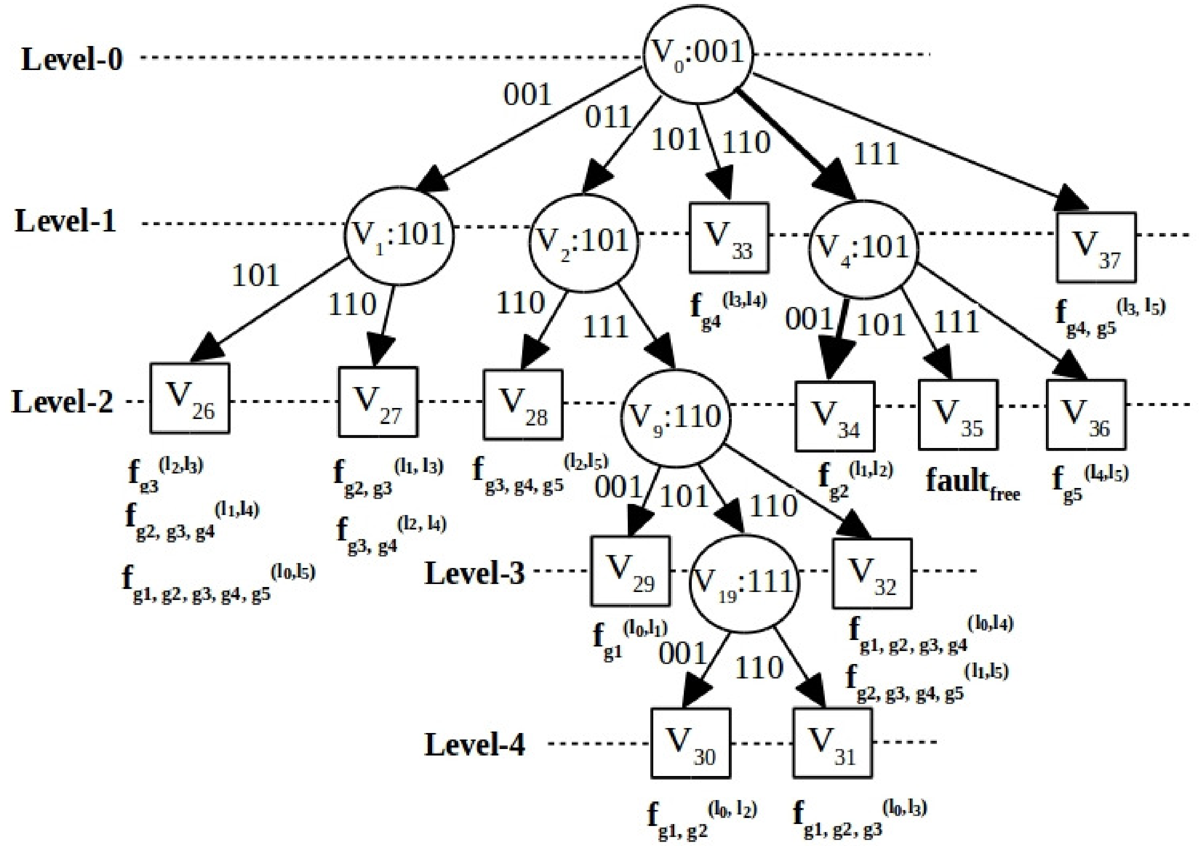


Figure 5.9: *Traversal process of fault $f_{g_2}^{(l_1,l_2)}$ for the circuit Miller_11*

only one directed edge and the weight valued incident edge is generated the path from the root to leaf vertex, which maintains the uniqueness of the output test patterns for each fault. If any fault occurs, then any one of the incident edge of a root vertex V_0 generates the path that identifies the location of their corresponding fault at the leaf node. □

5.4 Experimental Results and Discussions

The proposed fault localization method is implemented and tested with several reversible benchmark circuits based on the k-CNOT gates [25]. Python 3.4 is used to implement the algorithm and executed on a Core-i5 machine with an Intel Pentium (R) CPU-8250U@ 1.60GHz $\times$ 8 system, running Ubuntu v16.04 (64-bit) with 8 GB RAM. The experimental results are presented in Table 5.5. The first four columns describe the benchmark circuit

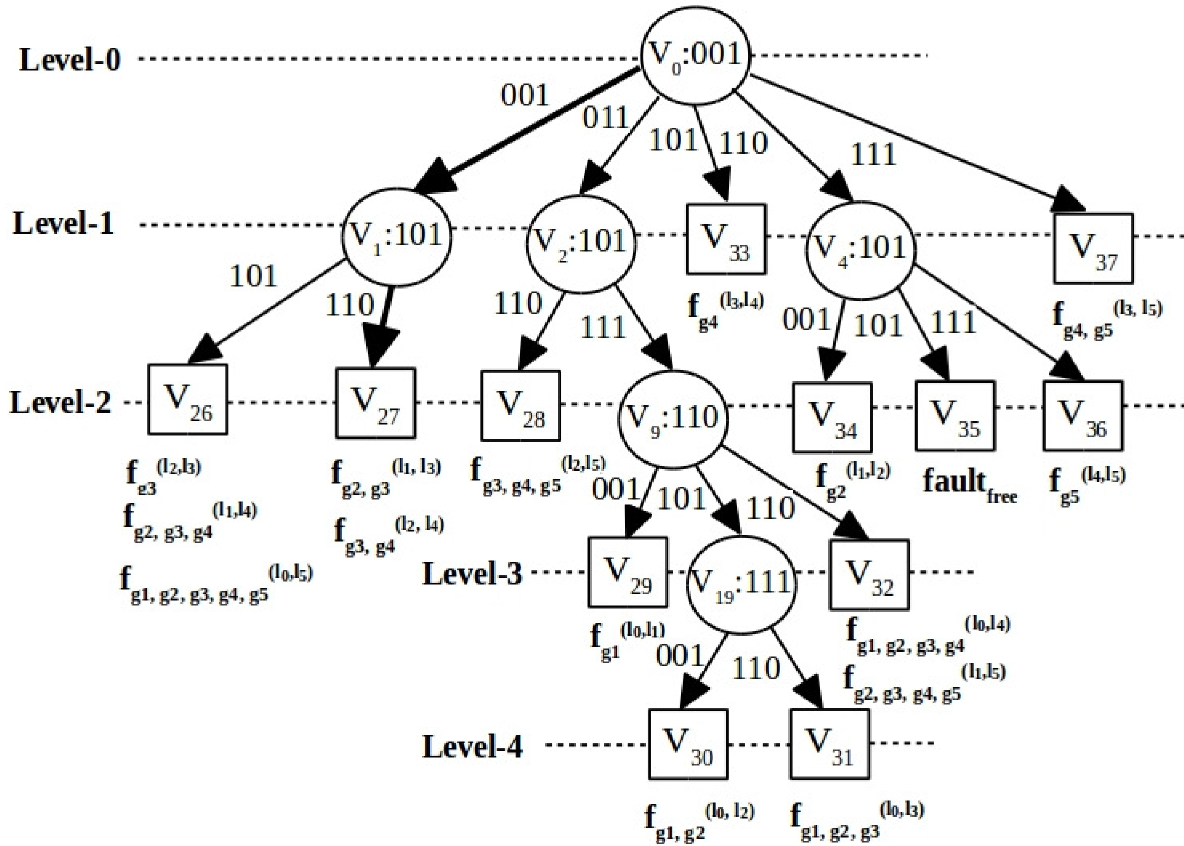


Figure 5.10: *Traversal process of fault* $f_{g2}^{(l_1,l_2)}$ *for the circuit* $Miller_11$

name, number of input lines (n), number of gates (N) and the total number of missing gate faults, respectively. The last four columns provide the number of test vectors in the complete tests for detecting the faults, number of functionally equivalent faults (FEFs), required number of test patterns (T_{in}) for finding the location for all the possible faults, and run-time in seconds, respectively. The experimental results show that the proposed fault localization method efficiently identifies the location of both SMGFs and MMGFs with 100% fault location coverage, and also identifies the equivalent faults present in the circuit. The complete test set TS obtained by the complete test set generation method of Chapter 4 is used in the proposed fault localization method.

Based on the experimental results as shown in Table 5.5, the analysis is provided for two parameters: (i) Number of faults Vs. CPU time and (ii) Number of test patterns

Table 5.5: *Experimental results for the fault localization of SMGFs and MMGFs with CPU time (sec) for the benchmark circuits*

Benchmark Circuit	n	N	No. of Faults (SMGF+MMGF)	TS	FEF	T_{in}	CPU Time (sec)
Peres_9	3	2	3	2	0	3	0.0156
Toffoli_double_4	4	2	3	4	0	4	0.0378
fredkin_6	3	3	6	3	2	3	0.0943
nth_Prime3	3	4	10	4	0	8	1.1627
ex_1_166	3	4	10	3	0	5	1.0097
rd32_V0_66	4	4	10	4	0	7	1.1268
4gt11-v1_85	5	4	10	4	0	4	1.0189
ham3_102	3	5	15	4	2	7	2.1608
miller_11	3	5	15	4	3	4	1.6045
4gt4-v0_80	5	5	15	4	1	6	1.8293
graycode6_47	6	5	15	5	0	10	2.3391
alu-v0_26	5	6	21	5	0	9	3.6311
3_17_13	3	6	21	4	0	8	3.0512
3_17_14	3	6	21	5	2	8	3.2788
mini-alu_167	4	6	21	5	2	7	2.9715
4_49_16	4	16	136	7	0	7	4.8218
rd73-140	10	20	210	20	0	12	6.3548
ham7_105	7	21	231	15	4	11	7.6219
cm82a-208	8	22	253	16	0	15	8.1131
hwb6tc	6	126	8001	40	0	18	11.9943

(T_{in}) Vs. CPU time. The detailed observations are as follows:

1. **Number of faults vs. CPU time:** In Figure 5.11, it is observed that there is an escalation in CPU time with the increase in the total number of faults for the benchmark circuits. The time required for fault localization is directly proportionate to the number of missing gate faults in the circuit. Consider the circuit *rd*73-140 with a total number of faults 210 and the circuit *ham*7-105 with a total number of faults 231. The CPU time required to generate the test patterns and for identifying the location of the faults for the circuit *rd*73-140 is 6.3548 *secs* and for the circuit *ham*7-105, it takes 7.6219 *secs*. Though the number of test patterns in *ham*7-105 (11 test patterns) is less than the number of test patterns in *rd*73-140 (12 test patterns), but to identify the location of the faults, the time required for the circuit *ham*7-105 is more as compared to the circuit *rd*73-140. This effect is due to the presence of more number faults in the circuit *ham*7-105. Based on the experimental results, it is also observed that the circuit 3_17_14 takes more time (3.2788 *secs*) to identify the location of the faults as compared to the circuit *mini-alu*_167 (2.9715 *secs*). Both the circuits have the same number of faults (21 faults), but the circuit 3_17_14 requires more number of test patterns (8 test patterns) for identifying the faults as compared to the number of test patterns for the circuit *mini-alu*_167 (7 test patterns). It is also observed that the circuit 3_17_13 and the circuit 3_17_14 are having the same number of faults (21 faults) and also having the same number of test patterns (8 test patterns). But to identify the location of faults, the circuit 3_17_14 requires more CPU time (3.2788 *secs*) as compared to the circuit 3_17_13 (3.0512 *secs*) for fault localization due to the presence of equivalent faults in 3_17_14 (2 equivalent faults).

2. **Number of test patterns vs. CPU time:** In Figure 5.12, it is observed that if the number of test pattern increases, the CPU time also increases for most of the benchmark circuits to identify the location of the fault. Consider the circuits *nth_Prime*3, *ex*_1_166, and *rd*32_*V*0_66 with the same number of faults (10 faults). But, the *nth_Prime*3 circuit requires more CPU time (1.1627 *secs*) for identifying

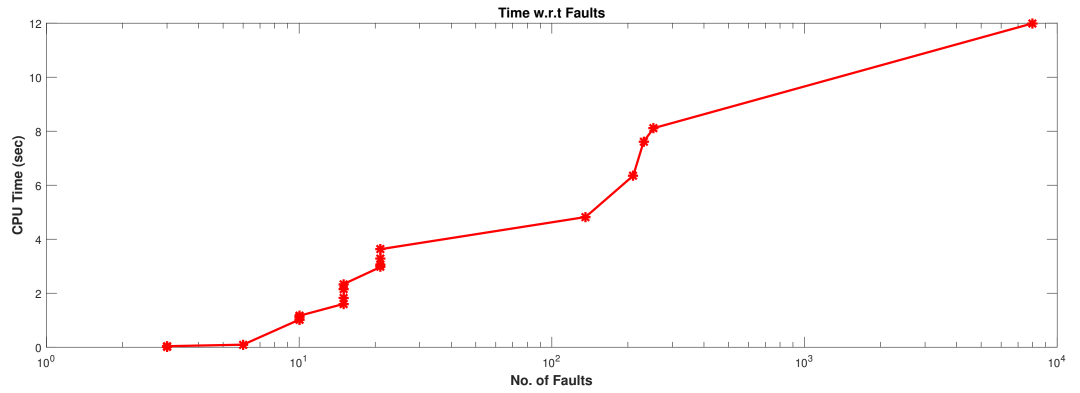


Figure 5.11: *Performance analysis of Number of faults vs. CPU time*

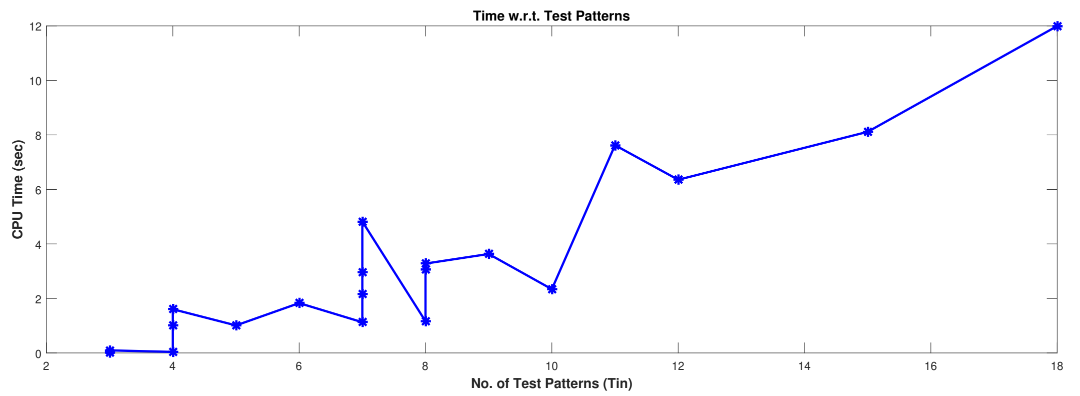


Figure 5.12: *Performance analysis of Number of test patterns vs. CPU time*

the location of the faults and this is due to the presence of more number of test patterns (8 test patterns) in the circuit as compared to the other two circuits. Also, it is noticed that the circuit $rd32_V0_66$ and the circuit 4_49_16 require equal number of test patterns (7 test patterns), but the circuit 4_49_16 takes more CPU time (4.8218 *secs*) as compared to the circuit $rd32_V0_66$ (1.1268 *secs*) to locate the faults. This happens due to the more number of faults in the circuit 4_49_16 (136 faults). As illustrated in Figure 5.12, the circuit 4_49_16 needs more time (4.8218 *secs*) as compared to the circuit nth_Prime3 (1.1627 *secs*) to identify the location of faults. This is due to the presence of more number of faults in the circuit 4_49_16 (136 faults) as compared to the circuit nth_Prime3 (10 faults), though the number of generated test patterns for the circuit 4_49_16 (7 test patterns) is less than that of the circuit nth_Prime3 (8 test patterns). Similar pattern is also observed in the graph for the circuit $ham7_105$ (11 inputs, 231 faults, 7.6219 *secs*) and the circuit $rd73_140$ (12 inputs, 210 faults, 6.3548 *secs*). Therefore, the impact of the number of faults is more significant than the number of test patterns in the circuit.

5.5 Conclusions

This chapter discussed a fault localization method for finding the location of single and multiple missing gate faults in the reversible k-CNOT based circuit. This method not only detects the faults but also distinguishes among them to obtain the exact location of the faults. Moreover, the generated input test pattern also determines the equivalent faults present in the circuit. In the next chapter, we summarize the main contributions of this thesis with some future directions of the current work.

Chapter 6

Conclusion and Future Work

In this chapter, we summarize the contributions of this thesis and outline some possible directions for future work.

6.1 Summarization

This thesis deals with the generation of test patterns for testing of reversible circuits. Circuit testing plays a major role in EDA (Electronics Design Automation) industries to ship-out the fault-free ICs from the industries. The major drawback of conventional digital circuits is the heat dissipation, and loss of information is one of the contributors for heat dissipation. Since there is no information loss in reversible circuits, so the research community of the circuit design is focusing towards the use of reversible circuits for system design. In reversible circuit domain also, circuit testing remains as an important phase for producing fault-free ICs.

The possible fault models for reversible circuits are described briefly in this thesis and we propose methods to generate the test set to detect faults using some of the fault models. Also, we explore the possibilities to locate the faults for Missing Gate Fault (MGF) model, and the MGF model is applicable to reversible circuits only. Now, we present the brief summaries of our work.

- Our first contribution, described in Chapter 2, is the test set generation method for detecting all single input bridging and stuck-at faults in reversible circuits. In

this work, it is established that the number of test vectors required to detect single input bridging and stuck-at faults is $\lceil \log_2 N \rceil + 1$ for a circuit having N-input lines. Based on the experimental results, it is evident that this approach overcomes the limitations of the previous works [1,2] in terms of the size of the complete test set. The experimental results show that the test set size of the proposed method is similar for the small circuit as compared to the existing method [2]. But, when the circuit size is large with respect to the number of input lines, then the proposed approach provides better results. Moreover, this approach can be efficiently used for large reversible circuits, which is tested by performing experiments with circuits having more than 40 input lines.

- Our next contribution, described in Chapter 3, addresses an efficient complete test set generation method for detecting all the single and multiple intra-level bridging faults of reversible circuits. In this approach, the local test pattern is applied for each level of the circuit that exploits the reversible properties, i.e., "*controllability*" and "*observability*". To obtain the complete test set for the intra-level bridging faults in a given k-CNOT circuit, the path-level expression is introduced, which is used to generate the minimal complete test set to detect all intra-level bridging faults in a reversible circuit. The scalability of the proposed method is measured with four parameters "CPU time", "Input lines", "Gate count", and "Number of faults" through experiments. Based on the performance analysis, it is observed that the impact of the number of input lines, and the number of faults are more significant than the number of gates present in the circuit. It is observed that the proposed method can handle reasonably large circuits. From the experimental results, it is noticed that the proposed work covers more faults as compared to the previous works [2–4]. Moreover, the number of test vectors generated by this approach is less or equal to the existing methods without using any extra circuit overhead. The time complexity of this approach is in logarithm order for generating the test set and also achieves 100% fault coverage.

- The next contribution, described in Chapter 4, provides an approach to generate

the complete test set for detecting the single and multiple missing gate faults in the k-CNOT based reversible circuits. This approach tries to improve the test generation process for detecting the multiple missing gate faults using the generated test patterns of SMGFs as well as the structure of k-CNOT circuits. For achieving the minimality, a table is constructed covering the row and column faults, which is used to formulate an ILP problem to get the minimal test set. The experimental results show that the minimized test set size is similar or smaller as compared to the existing methods [5–9]. This method is quite reasonable in terms of computational complexity, and efficiency is demonstrated through the experiments involving various k-CNOT based circuits. This method also attains 100% fault coverage. Moreover, the correlation of other fault models with missing gate model is shown without changing of the generated complete test set. Through the experimental results, it is observed that the generated complete test set of the proposed method covers most of the SAFs and PMGFs.

- Finally, our last contribution, described in Chapter 5, presents a fault localization method to obtain the exact location of SMGFs and MMGFs in reversible logic circuits. This method establishes that the generated complete test set to detect SMGFs and MMGFs can identify the location of missing gate faults. This approach uses the complete test set for SMGFs and MMGFs, as mentioned in Chapter 4, to extract the unique output responses for each missing gate faults in the k-CNOT based reversible circuit, which is used to locate missing gate faults. Based on the experimental results, the analysis is provided using three parameters "CPU time", "Number of faults", and "Number of test patterns". It is observed that the number of faults is more significant in terms of computation cost than the number of test patterns that is applied to the circuit for fault localization. This method is also capable of evaluating and identifying the functional equivalent faults (FEFs) with 100% fault location coverage.

6.2 Future Work

The work reported in the thesis keeps several directions open and there are scopes for future research in this area. In this section, we present some problems for future work.

- In this thesis, we have considered reversible circuits with positive controlled k-CNOT gates for generating the test patterns for detecting faults. In mixed controlled circuit topology, the polarity of control connections of a gate may be either positive or negative. Therefore, there is a possibility to extend our work for generating the efficient test pattern for different fault models of reversible circuits with mixed controlled circuit topology.

- In the mixed controlled circuit topology, there may be a possibility of flipping the controls (positive and negative). To detect the errors due to the flipping of polarity of the control connections (positive to negative or vice-versa), there may be a requirement to introduce new fault models to detect the faults due to the change of polarity in control connections. This may be an interesting direction to explore the possibilities to introduce a new fault model for reversible circuits and establish the correlation with the physical implementation of reversible gates with some technologies like quantum gates.

- In our work, we have observed that there is a correlation among the various types of fault models and testing approaches. There is a possibility to explore the generation of a universal test set, which can be used to detect all kind of faults in reversible circuits.

- Throughout the thesis, we have considered NCT and GT libraries, which are commonly used in the synthesis of reversible circuits. The proposed test pattern generation approaches can be extended to the different gate libraries such as NCTF (NOT, CNOT, TOFFOLT, FREDKIN) and newly proposed gate library [63] NCTSFPG3 (NOT CNOT TOFFOLI SWAP FREDKIN PERES G3), etc. for test pattern generation.

- In this thesis, we have considered the fault localization technique only for SMGFs and MMGFs in reversible circuits. There may be possibilities that the proposed fault localization approach may be extended to the other fault models in reversible circuits.

References

[1] P. Sarkar and S. Chakrabarti, "Universal test set for bridging fault detection in reversible circuit," in *2008 3rd International Design and Test Workshop.* IEEE, 2008, pp. 51–56. [Pg.xvii], [Pg.xix], [Pg.4], [Pg.19], [Pg.29], [Pg.38], [Pg.40], [Pg.41], [Pg.46], [Pg.122]

[2] P. Sarkar, B. Mondal, and S. Chakraborty, "Optimal universal test set for bridging faults detection in reversible circuit using unitary matrix," in *In Proc. of 2nd IEEE International Workshop on Reliability Aware System Design and TEST (RASDAT).* Nil, 2011, pp. 37–42. [Pg.xvii], [Pg.xix], [Pg.29], [Pg.38], [Pg.40], [Pg.41], [Pg.42], [Pg.46], [Pg.66], [Pg.68], [Pg.122]

[3] H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, "Optimum test set for bridging fault detection in reversible circuits," in *16th Asian Test Symposium (ATS 2007).* IEEE, 2007, pp. 125–128. [Pg.xix], [Pg.4], [Pg.11], [Pg.13], [Pg.19], [Pg.29], [Pg.46], [Pg.66], [Pg.67], [Pg.122]

[4] M. Bubna, N. Goyal, and I. Sengupta, "A dft methodology for detecting bridging faults in reversible logic circuits," in *TENCON 2007-2007 IEEE Region 10 Conference.* IEEE, 2007, pp. 1–4. [Pg.xix], [Pg.4], [Pg.19], [Pg.29], [Pg.46], [Pg.66], [Pg.67], [Pg.68], [Pg.122]

[5] A. P. Surhonne, A. Chattopadhyay, and R. Wille, "Automatic test pattern generation for multiple missing gate faults in reversible circuits," in *International Conference on Reversible Computation.* Springer, 2017, pp. 176–182. [Pg.xx], [Pg.4], [Pg.20], [Pg.74], [Pg.86], [Pg.88], [Pg.89], [Pg.123]

[6] A. Nagamani, S. Ashwin, B. Abhishek, and V. K. Agrawal, “An exact approach for complete test set generation of toffoli-fredkin-peres based reversible circuits,” *Journal of Electronic Testing*, vol. 32, no. 2, pp. 175–196, 2016. [Pg.xx], [Pg.4], [Pg.18], [Pg.20], [Pg.73], [Pg.86], [Pg.88], [Pg.89], [Pg.90], [Pg.123]

[7] J. P. Hayes, I. Polian, and B. Becker, “Testing for missing-gate faults in reversible circuits,” in *13th Asian Test Symposium.* IEEE, 2004, pp. 100–105. [Pg.xx], [Pg.3], [Pg.4], [Pg.11], [Pg.13], [Pg.14], [Pg.19], [Pg.72], [Pg.86], [Pg.89], [Pg.90], [Pg.123]

[8] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, “A family of logical fault models for reversible circuits,” in *14th Asian Test Symposium (ATS’05).* IEEE, 2005, pp. 422–427. [Pg.xx], [Pg.2], [Pg.3], [Pg.4], [Pg.11], [Pg.13], [Pg.14], [Pg.15], [Pg.72], [Pg.79], [Pg.81], [Pg.86], [Pg.89], [Pg.90], [Pg.123]

[9] B. Mondal, D. K. Kole, D. K. Das, and H. Rahaman, “Generator for test set construction of smgf in reversible circuit by boolean difference method,” in *2014 IEEE 23rd Asian Test Symposium.* IEEE, 2014, pp. 68–73. [Pg.xx], [Pg.4], [Pg.73], [Pg.86], [Pg.89], [Pg.90], [Pg.92], [Pg.123]

[10] G. E. Moore, “Cramming more components onto integrated circuits,” *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998. [Pg.1]

[11] R. Landauer, “Irreversibility and heat generation in the computing process,” *IBM journal of research and development*, vol. 5, no. 3, pp. 183–191, 1961. [Pg.1], [Pg.17]

[12] C. H. Bennett, “Logical reversibility of computation,” *IBM J. Res. Dev.*, vol. 17, no. 6, pp. 525–532, Nov. 1973. [Pg.2], [Pg.17]

[13] W. D. Pan and M. Nalasani, “Reversible logic,” *IEEE Potentials*, vol. 24, no. 1, pp. 38–41, 2005. [Pg.2], [Pg.17]

[14] M. P. Frank, “Throwing computing into reverse,” *IEEE Spectrum*, vol. 54, no. 9, pp. 32–37, 2017. [Pg.2], [Pg.17]

[15] ——, “Back to the future: The case for reversible computing,” *arXiv preprint arXiv:1803.02789*, 2018. [Pg.2], [Pg.17]

[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th ed. New York, NY, USA: Cambridge University Press, 2011. [Pg.2], [Pg.9]

[17] X. Ma, J. Huang, C. Metra, and F. Lombardi, "Reversible gates and testability of one dimensional arrays of molecular qca," *Journal of Electronic Testing*, vol. 24, no. 1-3, pp. 297–311, 2008. [Pg.2]

[18] C. Taraphdar, T. Chattopadhyay, and J. N. Roy, "Mach–zehnder interferometer-based all-optical reversible logic gate," *Optics & Laser Technology*, vol. 42, no. 2, pp. 249–259, 2010. [Pg.2]

[19] R. P. Feynman, "Quantum mechanical computers," *Foundations of physics*, vol. 16, no. 6, pp. 507–531, 1986. [Pg.2], [Pg.7]

[20] T. Toffoli, "Reversible computing," in *International Colloquium on Automata, Languages, and Programming.* Springer, 1980, pp. 632–644. [Pg.2], [Pg.8]

[21] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of theoretical physics*, vol. 21, no. 3-4, pp. 219–253, 1982. [Pg.2], [Pg.8]

[22] A. Peres, "Reversible logic and quantum computers," *Physical review A*, vol. 32, no. 6, p. 3266, 1985. [Pg.2], [Pg.8]

[23] D. Loss and D. P. DiVincenzo, "Quantum computation with quantum dots," *Physical Review A*, vol. 57, no. 1, p. 120, 1998. [Pg.2], [Pg.9]

[24] D. Maslov, "Reversible logic synthesis benchmarks page (2015), online: http://webhome.cs.uvic.ca/dmaslov/." [Pg.2], [Pg.22], [Pg.38], [Pg.62]

[25] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008).* IEEE, 2008, pp. 220–225. [Pg.2], [Pg.22], [Pg.23], [Pg.62], [Pg.86], [Pg.115]

[26] D. A. Maslov, "Reversible logic synthesis," Ph.D. dissertation, Fredericton, N.B., Canada, Canada, 2003, aAINQ98874. [Pg.2], [Pg.9], [Pg.47]

[27] A. N. E. Al-Rabadi, "Novel methods for reversible logic synthesis and their application to quantum computing," 2002. [Pg.2]

[28] H. Everitt, "Special issue on experimental aspects of quantum computing: Introduction," *Quantum Information Processing*, vol. 3, no. 1, pp. 1–4, 2004. [Pg.2]

[29] D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik: Progress of Physics*, vol. 48, no. 9-11, pp. 771–783, 2000. [Pg.2]

[30] C. Negrevergne, T. Mahesh, C. Ryan, M. Ditty, F. Cyr-Racine, W. Power, N. Boulant, T. Havel, D. Cory, and R. Laflamme, "Benchmarking quantum control methods on a 12-qubit system," *Physical Review Letters*, vol. 96, no. 17, p. 170501, 2006. [Pg.2]

[31] N. K. Jha and S. Gupta, *Testing of digital systems.* Cambridge University Press, 2003. [Pg.3], [Pg.10], [Pg.11], [Pg.12], [Pg.27]

[32] J. Rice, "An overview of fault models and testing approaches for reversible logic," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM).* IEEE, 2013, pp. 125–130. [Pg.3]

[33] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault testing for reversible circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 8, pp. 1220–1230, 2004. [Pg.3], [Pg.4], [Pg.7], [Pg.11], [Pg.19], [Pg.28], [Pg.85], [Pg.92]

[34] K. C. Mei, "Bridging and stuck-at faults," *IEEE Transactions on Computers*, vol. 100, no. 7, pp. 720–727, 1974. [Pg.3], [Pg.12]

[35] J. Zhong and J. C. Muzio, "Analyzing fault models for reversible logic circuits," in *2006 IEEE International Conference on Evolutionary Computation.* IEEE, 2006, pp. 2422–2427. [Pg.3], [Pg.16]

[36] M. Ibrahim, A. R. Chowdhury, and H. M. H. Babu, "Minimization of cts of k-cnot circuits for ssf and msf model," in *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems.* IEEE, 2008, pp. 290–298. [Pg.4], [Pg.19], [Pg.28]

[37] I. Polian and J. P. Hayes, "Advanced modeling of faults in reversible circuits," in *2010 East-West Design & Test Symposium (EWDTS).* IEEE, 2010, pp. 376–381. [Pg.4], [Pg.19]

[38] H. M. Gaur, A. K. Singh, and U. Ghaneka, "Design for stuck-at fault testability in mct based reversible circuits," *Defence Science Journal*, vol. 68, no. 4, pp. 381–387, 2018. [Pg.4], [Pg.29]

[39] A. Nagamani, B. Abhishek, and V. K. Agrawal, "Deterministic approach for bridging fault detection in peres-fredkin and toffoli based reversible circuits," in *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC).* IEEE, 2015, pp. 1–6. [Pg.4], [Pg.18], [Pg.47]

[40] H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, "On the detection of missing-gate faults in reversible circuits by a universal test set," in *21st International Conference on VLSI Design (VLSID 2008).* IEEE, 2008, pp. 163–168. [Pg.4], [Pg.19], [Pg.73]

[41] M. Zamani, M. B. Tahoori, and K. Chakrabarty, "Ping-pong test: Compact test vector generation for reversible circuits," in *2012 IEEE 30th VLSI Test Symposium (VTS).* IEEE, 2012, pp. 164–169. [Pg.4], [Pg.73]

[42] R. Wille, H. Zhang, and R. Drechsler, "Atpg for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization," in *2011 IEEE Computer Society Annual Symposium on VLSI.* IEEE, 2011, pp. 120–125. [Pg.4]

[43] A. Nagamani, S. Anuktha, N. Nanditha, and V. K. Agrawal, "A genetic algorithm-based heuristic method for test set generation in reversible circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 324–336, 2017. [Pg.4], [Pg.19]

[44] P. K. Lala, "An introduction to logic circuit testing," *Synthesis Lectures on Digital Circuits and Systems*, vol. 3, no. 1, pp. 1–100, 2008. [Pg.11]

[45] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial optimizationeureka, you shrink!* Springer, 2003, pp. 185–207. [Pg.17], [Pg.18]

[46] A. Chakraborty, "Synthesis of reversible circuits for testing with universal test set and c-testability of reversible iterative logic arrays," in *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*. IEEE, 2005, pp. 249–254. [Pg.19], [Pg.28]

[47] K. Ramasamy, R. Tagare, E. Perkins, and M. Perkowski, "Fault localization in reversible circuits is easier than for classical circuits," 2004. [Pg.20], [Pg.96]

[48] D. Pierce, J. Biamonte, and M. Perkowski, "Test set generation and fault localization software for reversible circuits," in *Proc. 7th International Symposium on Representations and Methodologies for Emergent Computing Technologies Tokyo Japan*, 2005. [Pg.20], [Pg.96]

[49] H. Zhang, R. Wille, and R. Drechsler, "Improved fault diagnosis for reversible circuits," in *2011 Asian Test Symposium*. IEEE, 2011, pp. 207–212. [Pg.20], [Pg.97]

[50] H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, "Fault diagnosis in reversible circuits under missing-gate fault model," *Computers & Electrical Engineering*, vol. 37, no. 4, pp. 475–485, 2011. [Pg.20], [Pg.97]

[51] N. Nayeem and J. Rice, "A simple approach for designing online testable reversible circuits," in *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*. IEEE, 2011, pp. 85–90. [Pg.29]

[52] A. Chakraborty, "Testing of bridging faults in and-exor based reversible logic circuits," *arXiv preprint arXiv:1009.5098*, 2010. [Pg.47]

[53] X. Fang-ying, C. Han-wu, L. Wen-jie, and L. Zhi-giang, "Fault detection for single and multiple missing-gate faults in reversible circuits," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 131–135. [Pg.72]

[54] D. K. Kole, H. Rahaman, D. K. Das, and B. B. Bhattacharya, "Derivation of optimal test set for detection of multiple missing-gate faults in reversible circuits," in *2010 19th IEEE Asian Test Symposium*. IEEE, 2010, pp. 33–38. [Pg.73]

[55] B. Zhang and V. D. Agrawal, "Three-stage optimization of pre-bond diagnosis of tsv defects," *Journal of Electronic Testing*, vol. 33, no. 5, pp. 573–589, 2017. [Pg.82]

[56] R. Wille, D. Große, S. Frehse, G. W. Dueck, and R. Drechsler, "Debugging of toffoli networks," in *2009 Design, Automation & Test in Europe Conference & Exhibition.* IEEE, 2009, pp. 1284–1289. [Pg.96], [Pg.97]

[57] J. C. Jung, S. Frehse, R. Wille, and R. Drechsler, "Enhancing debugging of multiple missing control errors in reversible logic," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI.* ACM, 2010, pp. 465–470. [Pg.97]

[58] M. Zamani, N. Farazmand, and M. B. Tahoori, "Fault masking and diagnosis in reversible circuits," in *2011 Sixteenth IEEE European Test Symposium.* IEEE, 2011, pp. 69–74. [Pg.97]

[59] B. Mondal, P. Das, P. Sarkar, and S. Chakraborty, "A comprehensive fault diagnosis technique for reversible logic circuits," *Computers & Electrical Engineering*, vol. 40, no. 7, pp. 2259–2272, 2014. [Pg.97]

[60] B. Mondal and S. Chakraborty, "A novel fault diagnosis in reversible logic circuit," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS).* IEEE, 2014, pp. 709–712. [Pg.98]

[61] B. Mondal, C. Bandyopadhyay, and H. Rahaman, "A testing scheme for mixed-control based reversible circuits," in *2016 Sixth International Symposium on Embedded Computing and System Design (ISED).* IEEE, 2016, pp. 96–100. [Pg.98]

[62] ——, "Detection and localization of appearance faults in reversible circuits," in *2017 7th International Symposium on Embedded Computing and System Design (ISED).* IEEE, 2017, pp. 1–5. [Pg.98]

[63] M. Y. Abubakar, L. T. Jung, M. N. Zakaria, A. Younesy, and A.-H. Abdel-Atyz, "New universal gate library for synthesizing reversible logic circuit using genetic programming," in *2016 3rd International Conference on Computer and Information Sciences (ICCOINS).* IEEE, 2016, pp. 316–321. [Pg.124]

Appendix: A

Journal Publications:

- Mousum Handique, Santosh Biswas, and Jantindra Kumar Deka, ***"Test Generation for Bridging Faults in Reversible Circuits Using Path-Level Expressions".*** Journal of Electronics Testing, vol. 35, no. 4, pp. 441–457, 2019
- Mousum Handique, Jantindra Kumar Deka, and Santosh Biswas, ***"An Efficient Test Set Construction Scheme for Multiple Missing-Gate Faults in Reversible Circuits".*** Journal of Electronics Testing, Springer, February 2020, DOI: https://doi.org/10.1007/s10836-020-05855-8

Conference Publications:

- Mousum Handique, Jantindra Kumar Deka, and Santosh Biswas, ***"Minimal test set generation for input stuck-at and bridging faults in reversible circuits.".*** TENCON 2017-2017 IEEE Region 10 Conference, IEEE, 2017. [**Best Paper Award**].