SYNTHESIS, TESTING AND TOLERANCE IN REVERSIBLE LOGIC

MD ASIF NASHIRY
Bachelor of Science, Islamic University, 2006
Master of Science, Islamic University, 2008

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

SYNTHESIS, TESTING AND TOLERANCE IN REVERSIBLE LOGIC

MD ASIF NASHIRY

Date of Defence: December 19, 2017

Dedication

To my beloved parents and

my daughter, Sarah Ibtida.

Abstract

In recent years, reversible computing has established itself as a promising research area and emerging technology. This is motivated by a widely supported prediction that conventional computer hardware technologies will reach their limits in the near future. This thesis focuses on three important areas of reversible logic, which is an area of reversible computing. Firstly, this thesis proposes a transformation based synthesis approach for realizing conservative reversible functions using SWAP and Fredkin gates. The proposed SWAP and Fredkin gates approach is compared with NOT, CNOT and Toffoli gates approach. Experimental results show that synthesizing conservative reversible functions using SWAP and Fredkin gates is more efficient than comparable approaches using NOT, CNOT and Toffoli gates.

Most existing synthesis approaches in reversible logic result in circuits that may not be optimal in terms of cost metrics such as the gate count, the number of garbage lines or the quantum cost. Hence, post synthesis optimization approaches are used to generate simplified circuits. This thesis proposes ten templates for optimizing SWAP and Fredkin gates-based reversible circuits. We have applied these templates in SWAP and Fredkin gates-based circuits, and achieved (on average) a 16% reduction in quantum cost.

Secondly, this thesis proposes an approach for the design of online testable reversible circuits. A reversible circuit composed of NOT, CNOT and Toffoli gates can be made online testable by adding two sets of CNOT gates and a single parity line. The performance of the proposed approach for detecting a single bit fault, a crosspoint fault and a family of missing gate faults has been observed. Discussion around the correctness of our approach and its overhead is also provided.

Thirdly, we have proposed an approach to achieve fault tolerance in reversible circuits. A design of a 3-bit reversible majority voter circuit is presented. The proposed majority voter circuit is simpler and of lower cost in terms of the gate count and the quantum cost than existing designs in the literature. This voter circuit can be used to design fault tolerant reversible circuits. We also provide designs for extending the voter circuit.

Acknowledgments

First and foremost, I would like to thank the Almighty for giving me the opportunity, strength, and patience to undertake this research. This work would not have been possible without His blessing.

I am using this opportunity to express my gratitude to everyone who supported me throughout this thesis. I would like to express my deepest gratitude and sincere appreciation to my supervisor, Dr. Jacqueline E. Rice, for her patience, support, effort, and engagement throughout this thesis. She provided necessary guidance and strategic direction to pursue my thesis, and helped me to navigate every aspect of my journey.

I would like to express my gratitude to my thesis committee members, Dr. Shelly Wismath, and Dr. Wendy Osborn, for their timely suggestions and useful remarks during my research. I would like to thank Dr. Mozammel Khan for his suggestion and guideline. I would like to extend my thanks to Dr. Hadi Kharaghani, Dr. Gerhard Dueck, and Dr. Hua Li for being part of my thesis committee.

I am thankful to the Natural Sciences and Engineering Research Council of Canada (NSERC), the Pacific Institute for the Mathematical Sciences (PIMS)-Alberta, and the School of Graduate Studies (SGS) of the University of Lethbridge for their financial support.

It is hard to express how grateful I am to my parents for their support and sacrifice. I would like to thank my wife, daughter, brothers, relatives, friends, and lab mates for their love, support and encouragement. Last but not least, I am also grateful to the researchers for their ideas and contributions in reversible logic.

Contents

Contents **vii**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**
1.1 Objectives . . . 2
1.2 Thesis Organization . . . 3

2 Background **5**
2.1 Logic Computation . . . 5
2.1.1 Traditional Logic Computation . . . 5
2.1.2 Reversible Logic Computation . . . 7
2.2 Logic Gates and Circuits . . . 9
2.2.1 Traditional Logic Gates . . . 9
2.2.2 Reversible Logic Gates . . . 10
2.3 Synthesis Approaches in Reversible Logic . . . 16
2.4 Cost Metrics in Reversible Logic . . . 18
2.5 Faults and Fault Testing in Reversible Logic . . . 21
2.6 Fault Tolerance in Reversible Logic . . . 22

3 Synthesis and Post-Synthesis **23**
3.1 Logic Synthesis in Reversible Logic . . . 23
3.1.1 Transformation Based Synthesis Approach . . . 24
3.1.2 A Proposed Transformation Based Synthesis Approach . . . 28
3.1.3 Comparison of Transformation Based Approaches . . . 30
3.2 Post Synthesis Optimization . . . 36
3.2.1 Proposed Templates . . . 41
3.2.2 Reversible Circuits Optimization . . . 51
3.3 Chapter Summary . . . 55
3.3.1 Contribution . . . 55
3.3.2 Conclusion . . . 56
3.3.3 Future Directions . . . 57

4 Fault Testing **60**
4.1 Fault Models and Fault Testing: An Overview 60
4.2 Fault Models for Reversible Logic 61
4.2.1 Crosspoint Fault Model 62
4.2.2 Missing Gate and Repeated Gate Fault Model 63
4.3 Fault Testing in Reversible Circuits 69
4.3.1 Offline Approaches 69
4.3.2 Online Approaches 74
4.4 New Online Testing Approach 83
4.4.1 Design 83
4.4.2 Analysis 85
4.4.3 Testing: Fault Detection 88
4.4.4 Comparison and Limitation 95
4.5 Chapter Summary 98
4.5.1 Contribution 98
4.5.2 Conclusion 98
4.5.3 Future Directions 99

5 Fault Tolerance in Reversible Logic **100**
5.1 Principle of Fault Tolerance 100
5.2 Techniques to Achieve Fault Tolerance 101
5.3 Testing vs. Tolerance 102
5.4 Approaches to Achieve Fault Tolerance 105
5.5 Fault Tolerance in Reversible Circuits 107
5.5.1 Existing Reversible Majority Voter Circuits 108
5.5.2 A Proposed Reversible Majority Voter 109
5.5.3 Fault Tolerant Full Adder Design 111
5.6 Extension of Reversible Majority Voter 112
5.7 Chapter Summary 117
5.7.1 Contribution 117
5.7.2 Conclusion 117
5.7.3 Future Directions 118

6 Conclusion **119**
6.1 Contribution and Conclusion 119
6.2 Future Works 121

Bibliography **124**

A Experiment Result **130**

List of Tables

2.1 Truth tables of traditional logic functions. 6
2.2 Truth tables of reversible logic functions. 7
2.3 Parity preserving and conservative reversible functions. 9
2.4 Truth table of a 3×3 Toffoli Gate. 13
2.5 Truth table of a 3×3 positive controlled Fredkin gate. 14
2.6 Transformation of an irreversible function to a reversible one. 17
2.7 Quantum cost of $n \times n$ Toffoli gates. 19
2.8 A 3-bit reversible function. 20

3.1 Truth table of a (3×3) reversible function. 25
3.2 Transformation stages of the function from Table 3.1. 27
3.3 Stages of SF based transformation of the function in Table 3.1. 30
3.4 Comparison between two circuits shown in Figures 3.1 and 3.2. 31
3.5 Performance comparison of NCT gate transformation based synthesis and SF gate transformation based synthesis. 32
3.6 Performance comparison of the minimal circuits generated using exact synthesis [67] with the circuits generated using SF gate based synthesis. 35
3.7 Percentage of reduction in GC and QC when using SF based transformation as compared to NCT based transformation. 37
3.8 Truth table for Template 5. 46
3.9 Results after applying the proposed templates on benchmark circuits. 55
3.10 Results after applying templates on randomly generated circuits. 59

4.1 Truth table of the circuit shown in Figure 4.1. 62
4.2 Output values at the different levels of the circuit shown in Figure 4.2. . . . 65
4.3 Output values at the different levels of a circuit shown in Figure 4.3. 67
4.4 Output values at the different levels of the circuit in Figure 4.4. 68
4.5 Test vector to detect appearance fault in the circuit shown in Figure 4.7. . . 71
4.6 Test vector to identify a SMGF for the circuit, as shown in Figure 4.8. . . . 72
4.7 Logic values at different levels to detect MGFs for the circuit, shown in Figure 4.9. 74
4.8 Truth tables of the circuits shown in Figure 4.18. 83
4.9 Truth table for the circuit shown in Figure 4.20(a) 86
4.10 Overhead for selected benchmark circuits. 96

5.1 Truth table for the reversible voter circuit. 110
5.2 Truth table of the 4-bit majority voter shown in Figure 5.12. 115

A.1 Results after applying templates on randomly generated circuits. 130

List of Figures

2.1 Some traditional circuits. 10
2.2 Some reversible gates of NCT gate family. 12
2.3 Some reversible logic gates of the SF gate family. 15
2.4 A reversible circuit consisting of three gates. 16
2.5 A 3×3-Toffoli gate operating as an AND gate. 18
2.6 Two different realizations of the same function using the SF gate family. . . 20

3.1 The circuit obtained from the function shown in Table 3.1. 27
3.2 Circuit resulting from SF based synthesis of the function from Table 3.1. . . 30
3.3 Circuit realization using NCT and SF gate families. 31
3.4 Further simplified circuits for the function shown in Table 3.1. 34
3.5 Two templates presented in [41]. 37
3.6 An example to demonstrate the deletion rule in a reversible circuit. 38
3.7 An example to demonstrate the moving rule for circuit simplification. . . . 39
3.8 A SF circuit where the moving rule does not work. 40
3.9 Template 1. 42
3.10 Template 2. 43
3.11 Template 3. 44
3.12 Template 4. 45
3.13 Template 5. 46
3.14 Template 6. 47
3.15 Template 7. 48
3.16 Template 8. 49
3.17 Template 9. 50
3.18 Template 10. 51
3.19 Moving rules for the SF gate families. 52
3.20 An example to show the role of the moving rule in circuit optimization. . . 53

4.1 Appearance and disappearance faults in a reversible circuit. 62
4.2 SMGF in a reversible circuit. 64
4.3 A RGF where a gate is replaced by an even number of instances of the same gate. 66
4.4 A RGF where a gate is replaced by an odd number of instances of the same gate. 66
4.5 An example of the occurrence of a MMGF. 69
4.6 A reversible circuit with the occurrence of a PMGF. 69
4.7 An example of a single appearance crosspoint fault. 70
4.8 An example of SMGF in a reversible circuit. 72

4.9 DFT based approach for missing gate testing [20]. 73
4.10 R1 and R2 gates proposed in [69]. 75
4.11 Block diagrams of pairing of $R1$ and $R2$ reversible gates. 76
4.12 An online testable circuit based on the approach proposed in [69]. 77
4.13 A CNOT gate and its deduced version for online testing. 77
4.14 TRC and TC for online testability. 78
4.15 A $(3+1)$-bit ETG gate. 79
4.16 An example of online testable reversible circuit based on [49]. 79
4.17 An example of online testable reversible circuit based on [28]. 80
4.18 Two existing reversible online testing approaches for PMGF. 82
4.19 Conversion of a Toffoli gate into a Duplicate Gate Block. 84
4.20 Transformation of a reversible circuit into its online testable equivalent. . . 85
4.21 Block diagram of an online testable reversible circuit. 86
4.22 Testing for the occurrences of SMGF and PMGF. 89
4.23 Testing for the occurrences of appearance crosspoint fault. 93
4.24 Testing for an occurrence of a single bit fault. 94

5.1 IG gate presented in [23]. 103
5.2 Parity Preserving Full Adder Circuit Composed of IG gates. 103
5.3 Parity Preserving Toffoli gate [18]. 104
5.4 Achieving fault tolerance using the concept of redundancy. 105
5.5 Three identical copies of a module using TMR in traditional computing. . . 108
5.6 A minimum triplicated voter circuit proposed in [72]. 108
5.7 A reversible majority voter as proposed in [6]. 109
5.8 A three input reversible majority voter. 110
5.9 Achieving fault tolerance using TMR. 111
5.10 A fault tolerant reversible full-adder circuit design. 112
5.11 A 4-bit majority voter. 113
5.12 A 4-bit majority voter designed from a 3-bit voter. 114
5.13 A 5-bit reversible majority voter. 116
5.14 A 6-bit reversible majority voter. 116

Chapter 1

Introduction

We are living in an age where there is an ever growing dependency on digital devices. The amount of information processed by digital devices continues to increases over time. In order to process this increasing volume of information, the number of components fabricated on integrated circuits of a digital device is also increasing over time. Over the past few decades, the size of the components has been reduced in order to increase the density of these components on integrated circuits. However, this pattern cannot continue forever because the current technology is approaching the physical limits of computing [14].

Limitations of traditional computing, such as heat dissipation, can become an obstacle for the further development of current technology [14, 15]. Logic in traditional computing is irreversible. In most cases, bits of information are destroyed during the logical operations that are performed in traditional computing. One of the fundamental limitations of traditional computing is that each time information is lost, energy is dissipated regardless of the underlying technology. In 1961 R. Landauer showed that $KTln2$ joules are dissipated each time an information bit is lost during a logical operation, where K is Boltzmann's constant and T is the operating temperature in Kelvin [30]. At room temperature, the amount of dissipated heat becomes 2.9×10^{-21} joules. For instance, when a two-input AND gate produces a single bit of output, this amount of energy is dissipated as heat. The amount of generated heat may not seem significant at present. However, as Moore's law [62], predicting a doubling of components every few years, has held true over the last several decades, this heat dissipation is becoming a major concern in traditional irreversible systems. Reversible computing [15] offers a solution to this potential deadlock of further development in traditional computing.

It was also shown by Charles Bennett that theoretical zero power dissipation can only be achieved if the circuit is logically reversible [5]. Reversible computing is bijective, and by definition reversible circuits are information-lossless [10]. Thus, by using reversible computation, the power dissipation which results according to Landauer's principle can be decreased or even eliminated. For this reason, in recent years reversible computation has established itself as a promising research area and emerging technology. Other reasons for research into reversible systems include connections to quantum computing [21], and applications in cryptography [64], nano-computing technologies [40] and digital processing [51]. It is widely believed that the next generation computers will be quantum computers. Quantum computations are reversible, and the circuits in quantum computing work on reversible functions [51]. Therefore, research on reversible logic is inevitable from various points of view.

1.1 Objectives

Most logic gates used in traditional computation are not reversible. In most cases, the relationship between the input and the output of a traditional gate is many-to-one. For example, an AND gate has two or more inputs and one output. However, the relationship between the input and the output of a reversible gate is one-to-one. Over the past few years, several reversible logic gates and synthesis methods have been proposed [11, 16, 37, 68]. However, analysing the logic gates and the synthesis approaches in order to optimize reversible circuits is still an active area of research. We hypothesize that one particular set of reversible gates can be more useful than another for realizing specific reversible functions. One of the objectives of this thesis is to develop a synthesis approach for realizing conservative reversible functions.

The number of gates of a circuit is considered an important complexity measure in traditional computing. In addition to this measure, the quantum cost and the number of garbage lines are two additional factors to consider in measuring the complexity of a reversible circuit. Template matching and rule based simplifications are two commonly used post-synthesis methods for optimizing reversible circuits. This thesis introduces new templates for simplifying reversible circuits.

The testing of reversible circuits is another important area. The physical implementation of quantum circuits will be different than the implementations of traditional circuits. Therefore, fault models designed for traditional circuits will not work for reversible circuits. Proposing a cost-effective testing approach for reversible fault models is another objective of this thesis.

Since a reversible circuit maintains a one-to-one relationship between inputs and outputs, achieving fault tolerance in such a system is not an easy task. A fault tolerant system can correctly perform its specified operations even in the presence of faults. This thesis also investigates the existing work on the design of fault tolerant reversible circuits and presents an efficient approach for achieving fault tolerance in reversible circuits.

1.2 Thesis Organization

The remainder of this dissertation is organized as follows.

Chapter 2 provides a detailed introduction to reversible logic and presents the required background knowledge necessary for this dissertation.

Chapter 3 discusses transformation based synthesis in reversible logic. A proposed approach based on this technique is presented in this chapter. The performance of this proposed approach is evaluated and compared with an existing approach. This chapter also introduces templates and presents a post synthesis optimization approach for simplifying reversible circuits.

Chapter 4 describes reversible fault models and fault testing approaches. The existing testing approaches in reversible circuits are analysed and the limitations of these approaches are identified. A testing approach for three reversible fault models is proposed later in this chapter.

Chapter 5 focuses on fault tolerance. The requirements for achieving fault tolerance in reversible circuits are discussed in this chapter. A majority voter circuit is presented, which can be used to design fault tolerant reversible circuits. An extension of this voter circuit and other areas of application of the proposed voter are also discussed in this chapter.

Chapter 6 highlights the contributions, concludes with discussions and provides directions for possible future work.

Chapter 2

Background

This chapter describes the fundamental concepts of reversible logic as well as the principles of synthesis, fault testing and fault tolerance in reversible logic.

2.1 Logic Computation

Logic plays a major role in modern day computation. Digital logic is a logic for representing and manipulating digital information. Logic in computer science deals with defining a problem in terms of Boolean functions, and designing a circuit in order to implement the functions. Logic in today's digital devices follows the principle of Boolean logic, which is also known as classical or traditional logic. As discussed in Chapter 1, there exists limits to the existing physical components of a digital system that implement traditional logic. Reversible logic offers one possible model to design a digital system to overcome such limitations.

2.1.1 Traditional Logic Computation

Like any other function, a traditional logic function maps one or more inputs to one or more outputs. More specifically, a traditional logic function, f, takes the form $f : B^n \longrightarrow B^m$, where n and m are non-negative integers, and $B = \{0, 1\}$ is a Boolean domain. The values of n and m may or may not be equal and in most cases $n > m$. For example, a simple circuit consisting of only one AND gate has two (or more) inputs, but has only one output. That is, logic computations in traditional approaches often map multiple inputs to fewer outputs. A convenient way to represent the relationship between inputs and outputs is by using a truth table. A truth table shows all possible input and output combinations for a specific logic function. A truth table for a Boolean logic function uses $(n+m)$ columns and 2^n rows to show the behaviour of a function for all possible input instances of the function of n input and m output.

Table 2.1: Truth tables of traditional logic functions.

(a) NOT operation.

I	O
0	1
1	0

(b) AND operation.

I_1	I_2	O
0	0	0
0	1	0
1	0	0
1	1	1

(c) Full adder operation.

I_1	I_2	c_{in}	sum	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

One of the simplest examples of a traditional logic function is a logical NOT operation, which takes one input and generates one output, as shown in Table 2.1(a). Tables 2.1(b) and 2.1(c) show functions for which the relationship between inputs and outputs is not one-to-one. Hence, for these functions it is not possible to determine the input states by observing only the output of a function. For example in Table 2.1(b), the output values are 0 for the input states $(0,0),(0,1)$, and $(1,0)$. By observing this output value, 0, it is not possible to determine whether the input values are $(0,0),(0,1)$, or $(1,0)$. A similar observation can be made from the truth table shown in Table 2.1(c) and most other traditional or classical logic functions. The relationship between inputs and outputs of a traditional function is not bijective (with the only exception being a NOT operation). This means that the relationship between inputs and outputs of a traditional function is not one-to-one and onto. Since bijective relationships between inputs and outputs do not exist for most traditional functions, it is not possible to determine an input instance from an output instance of a traditional function. For this reason traditional logic is sometimes referred to as irreversible logic.

2.1.2 Reversible Logic Computation

A reversible logic function has the form $f : B^n \longrightarrow B^n$, where n is a non-negative integer and the domain $B = \{0, 1\}$, with the key feature being that the function is bijective. More specifically, the number of inputs and the number of outputs of a reversible function are exactly the same. In particular, there is always a distinct output state for each of the possible input states.

Table 2.2: Truth tables of reversible logic functions.

(a) NOT operation.

I	O
0	1
1	0

(b) Contolled NOT operation.

I_1	I_2	O_1	O_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

(c) Full adder operation.

g_{in}	c_{in}	i_1	i_2	c_{out}	sum	g_{out1}	g_{out2}
0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	1
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	1	0	1
1	0	0	0	1	0	0	0
1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	0
1	0	1	1	0	0	0	1
1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	1	0	1

The work in this thesis is restricted to Boolean reversible logic functions. For a Boolean reversible function of n variables, a truth table requires $2n$ columns and 2^n rows in order to represent the function. Table 2.2(a) shows that a NOT operation in reversible logic has the same functionality as that of a traditional NOT operation (Table 2.1(a)). In fact, a NOT operation in traditional logic is reversible in nature. As long as there is a unique output for each input instance of a function, the function is reversible (Table 2.2). This relationship between the input and the output allows the determination of input values of a function from the output values. In order to transform an irreversible function into a reversible one, it is often necessary to include one or more extra variables on the output and/or input sides of a truth table of the irreversible function. For example, the truth table of a reversible full adder shown in Table 2.2(c) consists of extra variables on both the input and output as compared to the truth table of its irreversible counterpart shown in Table 2.1(c). The additional inputs are called constant inputs, or ancilla inputs. The additional outputs are non-functional outputs because these are not the output of interest of the reversible function. In a full adder, sum and carry are the two output bits of interest. In Table 2.2(c), two outputs, sum and c_{out}, represent the sum and the carry bits of a full adder. The other two outputs, g_{out1} and g_{out2}, are used to maintain the bijective relationship between the inputs and the outputs of this reversible full adder function. These non-functional outputs are known as garbage outputs, as they do not contribute to the property of the original output of a function.

Two important properties of reversible functions are the parity preserving property and the conservation property. A parity preserving reversible function, as the name suggests, preserves the parity of the input vectors to the corresponding output vector. For example, in a parity preserving reversible function, if the parity of an input vector is even, the parity of the output vector will also be even. Every output vector of the function shown in Table 2.3(a) preserves the parity (either odd or even) of the corresponding input vector. Hence, the function shown in Table 2.3(a) is an example of a parity preserving function. However, this function is not a conservative function. A reversible function is called a conservative function if and only if the number of 1s in the output vector is the same as that of the corresponding input vector, for all inputs. The function shown in Table 2.3(b) is an example of a conservative reversible function. Chapter 4 shows the role of the parity preserving property in the fault detection and testing of reversible circuits. A synthesis approach based on the conservation property is presented in Chapter 3.

Table 2.3: Parity preserving and conservative reversible functions.

(a) A parity preserving function.

input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	0	0

(b) A conservative function.

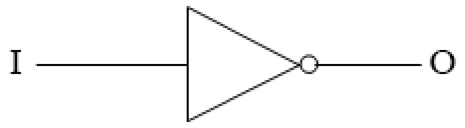
input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

2.2 Logic Gates and Circuits

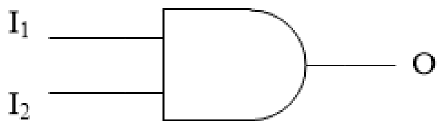
A logic gate performs one or more logical operations in order to implement a logical function. A circuit consists of one or more logic gates, which are connected by some form of interconnected media (e.g. wire). This section describes some logic gates as well as logic circuits in traditional and reversible logic computation.

2.2.1 Traditional Logic Gates

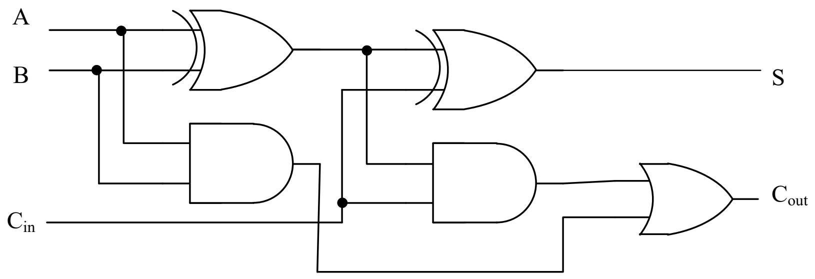
Several logic gates exist for designing circuits in traditional logic. The gates AND, OR and NOT are the primary logic gates. Figure 2.1 shows both a NOT and an AND logic gate. These two gates are used to implement the traditional NOT (Table 2.1(a)) and AND (Table 2.1(b)) functions, respectively. A NOT gate works as an inverter, which always takes one input and produces one output after inverting the input value. An AND gate, on the other hand, works on more than one input and produces a single output. A full adder circuit, which consists of traditional logic gates, is shown in Figure 2.1(c), and implements the function represented by Table 2.1(c). A traditional full adder circuit has three input lines and two outputs lines to generate sum and carry bits.



(a) A traditional NOT gate.



(b) A traditional AND gate.



(c) A traditional full adder circuit.

Figure 2.1: Some traditional circuits.

2.2.2 Reversible Logic Gates

A reversible circuit consists of one or more reversible gates that are connected in cascade. That is, the output(s) of a prior gate is/are connected as input(s) to the following gate. For example, consider a reversible circuit $g_2(g_1(x_1,x_2))$ where the outputs of a logic gate g_1 are connected as the inputs of the gate g_2 in order to implement a 2-bit reversible function, $f(x_1,x_2)$. Like a reversible logic function, one of the fundamental characteristics of a reversible gate is that the relationship between inputs and outputs is one-to-one. This implies that the output values of a reversible gate are distinct for any given input. So unlike a traditional irreversible gate, it is always possible to restore the input values of a reversible gate from the outputs. A traditional NOT gate is also a reversible gate, since it is possible to deduce the input of a NOT gate from its output. For example, 1 at the output of a NOT gate for a particular instance indicates that the input of that instance is 0.

A number of reversible gates exist in order to implement reversible functions. The two most widely used categories of reversible logic gates are [71] : the NCT (NOT-CNOT-Toffoli) gate family and the SF (SWAP-Fredkin) gate family.

NCT gate family

The NCT gate family (Figure 2.2) consists of NOT, CNOT and Toffoli gates. This is one of the most widely used gate families in reversible computing. The simplest possible reversible logic gate is a NOT gate. A NOT gate (Figure 2.2(a)), like a NOT gate in traditional logic, consists of one input and one output, which inverts the value of the input to generate the output. A CNOT gate and a Toffoli gate are variations of a NOT gate.

A CNOT gate [12] is a controlled-NOT gate with a control input and a target input. The value presented at the control point determines when to invert the value of the target input, or simply transfers the target input value to the output. Figure 2.2(b) shows a CNOT gate. The ($\bullet$) symbol in the CNOT gate represents a control point and the ($\oplus$) symbol indicates the target of a reversible gate. When the value of the control of a CNOT gate is 1, the gate inverts the value of the input that is connected to the target line. If the value of the control point is 0, the value of the target input remains unchanged to the target output. For example, when the inputs of a CNOT(I_1, I_2) gate in Figure 2.2(b) are $(1,0)$, the output (O_1,O_2) will be $(1,1 \oplus 0) \equiv (1,1)$. Unlike a NOT gate, a CNOT gate can control the inversion of an input value, hence its name is controlled-NOT or CNOT. A CNOT gate is also known as Feynman gate. A variation of a Feynman gate is a double Feynman gate. A double Feynman gate consists of one control point and two target lines. When the value of the control point is 1, the double Feynman gate inverts both input values that are connected to the target lines [53].

A Toffoli gate [68] is a form of the CNOT gate where the number of control points is

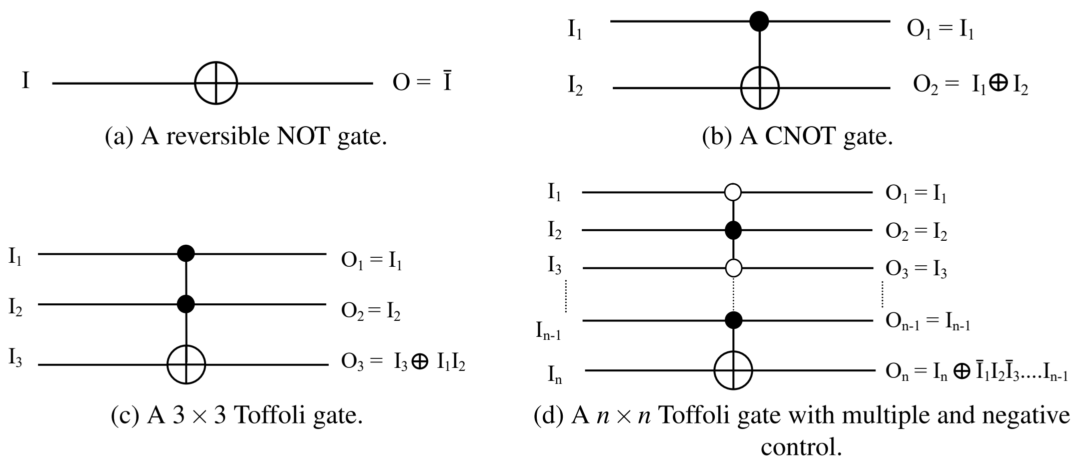


Figure 2.2: Some reversible gates of NCT gate family.

more than 1. For example, Figure 2.2(c) shows a 3×3 Toffoli gate with two control points and one target line. The two input lines, I_1 and I_2, are connected to the two control points and the third input I_3 is connected to the target line. The values of the inputs, which are connected to the control lines, remain unchanged at the output. The values presented at the input of these control lines determine when the gate will invert the value presented at the target input line. The target line in Figure 2.2(c) has the function: $O_3 = I_3 \oplus I_1 I_2$. This means the value of the target line will be inverted when both inputs that are connected to the control points have the value 1. For example, when $(I_1, I_2, I_3) \equiv (0,1,1)$, the output of the Toffoli gate will be $(O_1, O_2, O_3) \equiv (0,1,1)$. In this case there is no change on the value of the target input, since one of the control points (I_1) is equal to 0. However, when $(I_1, I_2, I_3) \equiv (1,1,1)$ the gate output will be $(O_1, O_2, O_3) \equiv (1,1,1 \oplus 1 \cdot 1) \equiv (1,1,1 \oplus 1) \equiv (1,1,0)$. In this case both control points have the value 1, which is the required condition for inverting the input values at the output. The truth table of a 3×3 Toffoli gate is shown in Table 2.4.

Based on the number k of control points of a gate, a logic gate in the NCT family is also referred to as a k-CNOT gate. For example, based on the number of control lines the NOT, CNOT and 3×3 Toffoli gates are also known as 0-CNOT, 1-CNOT and 2-CNOT gates respectively. A Toffoli gate with multiple controls is referred to as a multiple controlled Toffoli (MCT) gate. In addition, a Toffoli gate or a Feynman gate can have one or more

Table 2.4: Truth table of a 3×3 Toffoli Gate.

input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	0	0	1
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	1	1	1
1	1	1	1	1	0

negative control points. For example, a negative control Feynman (1-CNOT) gate inverts the input value at the target line if and only if the value of the control point is 0. A negative control is represented by a $(\circ)$ symbol in the NCT gate library. For example, Figure 2.2(d) shows a multiple controlled Toffoli gate where two input lines I_1 and I_3 are connected to the two negative control lines. A negative control, like a positive control of a logic gate, does not affect the value on the line to which it is connected. The output function of the target of a $n \times n$-Toffoli gate, as shown in Figure 2.2(d), is $O_n = I_n \oplus \overline{I_1} I_2\, \overline{I_3} I_{n-1}$. This means that when the values of all negative control points are 0, and the values of all positive control points are 1, the gate will invert the value that is connected to its target input.

Another common way to represent the gates of the NCT family is in the form of Toffoli gates. A TOF(C;T) notation is used to indicate a Toffoli gate with C control points and T target lines. For instance, TOF(0;T), TOF(1;T) and TOF(2;T) represent a NOT, a CNOT and a 3×3 Toffoli gates, respectively. An important property of a Toffoli gate is that it can be used to implement any reversible function. For this reason, a Toffoli gate is known as a universal gate in reversible logic [52].

SF Gate Family

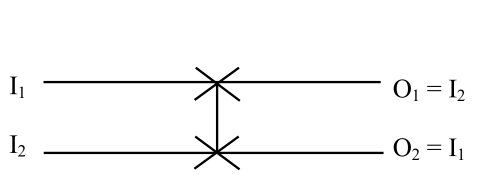
Two other important reversible logic gates are SWAP and Fredkin gates. A Fredkin gate [16] is a variation of SWAP gate with one or more control points. A Fredkin gate is also a universal gate in reversible logic, since it is possible to implement the three basic operations (NOT, AND and OR) using only Fredkin gates [52]. As discussed above, the logic gates in the NCT gate family invert the value of the target input based on certain conditions (in the case of CNOT, Toffoli gates) or without any condition (in the case of NOT gates). Instead of inverting the input value on a target line, the target lines of an SF gate interchange their values. Generally, a gate of the SF gate family has two target lines. When a gate interchanges the values of the target lines at the output without any condition, the gate is called a SWAP gate. However, if the swap or interchange of the target values occurs based on the values presented on the control lines, the gate is called a Fredkin gate. The truth table of a 3×3 positive controlled Fredkin gate is shown in Table 2.5. Figures 2.3(a) and 2.3(b)

Table 2.5: Truth table of a 3×3 positive controlled Fredkin gate.

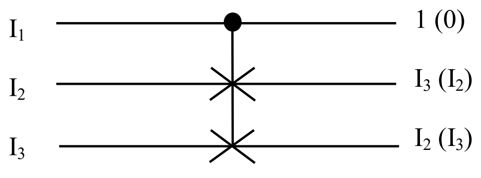
input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

show a SWAP gate and a 3×3 positive controlled Fredkin gate respectively. A SWAP gate always swap the values which are connected to its targets lines. For an input vector (I_1, I_2), the output of the SWAP gate will be $(O_1 = I_2, O_2 = I_1)$, as shown in Figure 2.3(a). However, a positive controlled Fredkin gate interchanges two target input values (I_2, I_3) if the value of the other input, I_1, which is connected to a control line (●), is 1. If the value of I_1 equals to 0, the Fredkin gate simply transfers all the input values to the corresponding output values without interchanging the target values. A 3×3 positive controlled Fredkin gate with a control point connected to I_1, and two target lines, I_2 and I_3, has the following output functions at

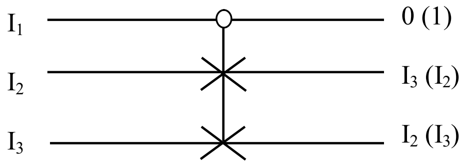
1. control point: $O_1 = I_1$
2. target line: $O_2 = \overline{I_1}I_2 \oplus I_1I_3$, and
3. target line: $O_3 = I_1I_2 \oplus \overline{I_1}I_3$.



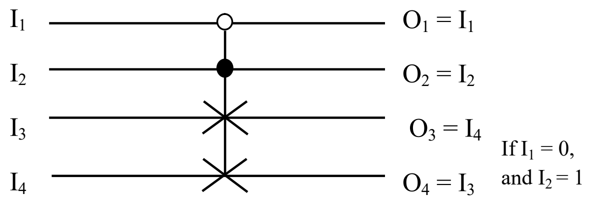
(a) A SWAP gate.



(b) A (3×3) positive controlled Fredkin gate.



(c) A (3×3) negative controlled Fredkin gate.



(d) A $(n \times n)$ multiple controlled Fredkin gate.

Figure 2.3: Some reversible logic gates of the SF gate family.

When a Fredkin gate consists of more than one control point, the Fredkin gate is called a multiple controlled Fredkin (MCF) gate. Like a Toffoli gate, a Fredkin gate can also have one or more negative controls (Figure 2.3(c)). In this case the gate swaps the values connected to the target lines if and only if a 0 value is presented at the negative control point. Figure 2.3(d) shows an example of a multiple controlled Fredkin gate consisting of both positive and negative control points. In this case a 1 at the positive control and a 0 at the negative control point activate the gate to interchange the values on the target lines.

One or more reversible gates can be used to generate a circuit in order to implement a reversible function. For example, consider a reversible function $f(I_1, I_2, I_3) \equiv (O_1, O_2, O_3)$ where (O_1, O_2, O_3) are defined as $(\overline{I_1}, \overline{I_1} \oplus I_2, (\overline{I_1} \oplus \overline{I_1}I_2) \oplus I_3)$ respectively. These three output functions can be implemented by the circuit $TOF(TOF(TOF(I_1); I_2); I_3)$ consisting of three reversible gates as shown in Figure 2.4. As seen in this example the outputs of one gate g_i are connected as the inputs for the gate g_{i+1} which appears immediately next to g_i in the circuit. The output of the NOT gate, $TOF(I_1)$, is $\overline{I_1}$ which is connected as an input of the following CNOT gate. This then computes $TOF(TOF(I_1);I_2) = TOF(\overline{I_1};I_2) = (\overline{I_1}, \overline{I_1} \oplus I_2)$. Finally, the outputs of this CNOT gate are connected as the inputs of the 3-bit Toffoli gate, which computes $TOF(TOF(TOF(I_1);I_2);I_3) = TOF(TOF(\ \overline{I_1};I_2);I_3) = TOF(\overline{I_1}, \overline{I_1} \oplus I_2; I_3) = (\overline{I_1}, \overline{I_1} \oplus I_2, \overline{I_1} \cdot (\overline{I_1} \oplus I_2) \oplus I_3) = (\overline{I_1}, \overline{I_1} \oplus I_2, \overline{I_1} \oplus \overline{I_1} \cdot I_2 \oplus I_3)$.

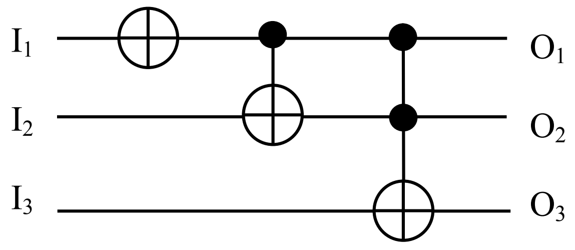


Figure 2.4: A reversible circuit consisting of three gates.

2.3 Synthesis Approaches in Reversible Logic

The concept of synthesis is very important in designing reversible logic circuits. Synthesis refers to the transformation of a logic function into a corresponding logic circuit. According to some synthesis approaches, if a logic function is irreversible, the first step of a logic synthesis is to transform the function into its reversible equivalent. One or more garbage lines and/or constant inputs are included in the original irreversible function in order to make the function reversible. The final step is to transform the reversible function into a logic circuit consisting of one or more reversible gates which are connected in cascade. The resulting circuit is a reversible circuit. There can be more than one reversible circuit for implementing a single function. Two important factors play a significant role in transforming irreversible functions into reversible circuits: (1) the number of garbage lines and/or constant inputs, which are included in order to transform the irreversible function to a reversible one; and (2) the use of different reversible logic gates for realizing the reversible function by a reversible circuit.

During the process of transforming an irreversible function into the corresponding reversible function, it is necessary to observe the output of the irreversible function. If the output of an irreversible function has a maximum number of k identical patterns, the minimum number of garbage lines required to make the function reversible is $\lceil log_2 k \rceil$ [35]. For instance, consider the irreversible AND function in Table 2.6(a). The output of this irreversible function has 3 occurrences of logic 0. Thus the minimum number of garbage lines required in order to transform the function into its reversible counterpart is $\lceil log_2 3 \rceil = 2$ as shown in Table 2.6(b). For the input line, $I_3 = 0$ in Table 2.6(b), the function performs the AND operation of input lines, I_1 and I_2, and generates the output at O_3. The other two output lines, O_1 and O_2, do not contribute to the final output, and hence these two lines are considered garbage lines. An important fact to observe is that the behaviour of the function in Table 2.6(b) is the same as the behaviour of a 3×3-Toffoli gate (Table 2.2(c)). Thus, a 2-input traditional AND operation can be implemented by a 3×3-Toffoli gate, as shown in Figure 2.5. Chapter 3 covers more regarding synthesis and post synthesis approaches in reversible logic.

Table 2.6: Transformation of an irreversible function to a reversible one.

(a) Irreversible AND function.

input		output
I_1	I_2	O
0	0	0
0	1	0
1	0	0
1	1	1

(b) One possible corresponding reversible AND function.

input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	0	0	1
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	1	1	1
1	1	1	1	1	0

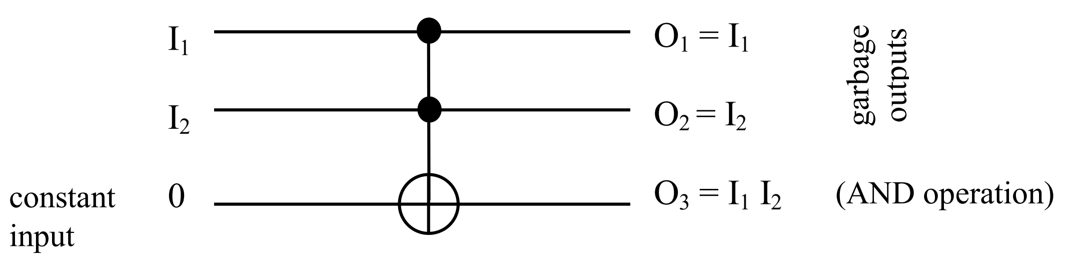


Figure 2.5: A 3×3-Toffoli gate operating as an AND gate.

2.4 Cost Metrics in Reversible Logic

Since a reversible function can be synthesized in multiple ways, it is necessary to evaluate the cost of the synthesis approaches. A number of metrics can be used to evaluate the efficiency of circuits for realizing the same reversible function [45]. Gate count is one of the most common cost metrics, particularly in traditional logic design. As the name suggests, gate count refers to the number of gates required in a circuit to realize a function. For example, one Toffoli gate is used to implement the AND operation, as shown in Figure 2.5. In this case the gate count is 1. However, gate count is not a useful parameter when the circuits consist of different types of gates. Gate count does not consider gate complexity; it instead simply counts the number of gates in a circuit. Gate count can be a good evaluator when two circuits consist of gates of the same size and complexity. Mohammadi et al. [45] proposed a new form of gate count by giving some weight to the gates based on the complexity of the gates of a circuit.

The number of garbage outputs in a circuit is another important cost metric. During the synthesis process in reversible logic, one or more garbage outputs may be needed for maintaining reversibility. The values of the garbage lines are not significant to the final output of a circuit. Therefore, it is desirable to minimize the number of garbage outputs. A circuit design with fewer garbage lines is considered a desirable design.

For many, quantum cost is the most important parameter for evaluating a reversible circuit from a design standpoint. Quantum cost is defined as the number of basic or primitive quantum gates which are required to design a reversible gate. The 1×1 and 2×2 quan-

Table 2.7: Quantum cost of $n \times n$ Toffoli gates.

Size (n)	Garbage	Name	Quantum Cost
1	0	NOT,t1	1
2	0	CNOT,t2	1
3	0	Toffoli,t3	5
4	0	Toffoli,t4	13
5	0	t5	29
5	2	t5	26
6	0	t6	61
6	1	t6	52
6	3	t6	38
7	0	t7	125
7	1	t7	80
7	4	t7	50
8	0	t8	253
8	1	t8	100
8	5	t8	62
9	0	t9	509
9	1	t9	128
9	6	t9	74
10	0	t10	1021
10	1	t10	152
10	7	t10	86
$n > 10$	0	tn	$2^n - 3$
$n > 10$	1	tn	$24n - 88$
$n > 10$	$n - 3$	tn	$12n - 34$

tum gates are considered primitive quantum gates. For example, quantum NOT, CNOT, V and V^+ gates are considered primitive quantum gates, and the quantum cost of these gates is considered to be 1. The quantum cost of a Toffoli gate and a Fredkin gate is 5, since five primitive gates are required for designing these gates. Barenco et al. [4] showed a realization of a 3×3 Toffoli gate consisting of 2 positive control points using five primitive quantum gates. Smolin and DiVincenzo [66] presented an implementation of a 3×3 positive-controlled Fredkin gate using five 2-qubit (quantum bit) elementary quantum gates.

Quantum compuation and quantum gates are beyond the scope of this thesis. A detailed discussion on quantum gates can be found in [4, 43, 51]. In addition, there have been a number of works such as [2, 4, 34, 35, 39, 33, 58] on minimizing the quantum cost for designing a reversible circuit. Table 2.7 shows the quantum cost of positive control Toffoli gates [71].

Table 2.8: A 3-bit reversible function.

input			output		
I_1	I_2	I_3	O_1	O_2	O_3
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	1	1	1

Another important factor to consider when designing reversible circuits is to minimize the number of garbage outputs. However, there are some cases where the quantum cost of a circuit can be reduced by adding garbage lines [33]. For example, as seen from Table 2.7, the quantum cost of a 6×6 Toffoli gate without including any extra line as a garbage line is 61. However, after adding 1 and 3 lines as garbage lines the quantum cost of a 6×6 Toffoli gate can be reduced to 52 and 38 respectively. Therefore, a large number of garbage lines does not always yield a higher quantum cost.

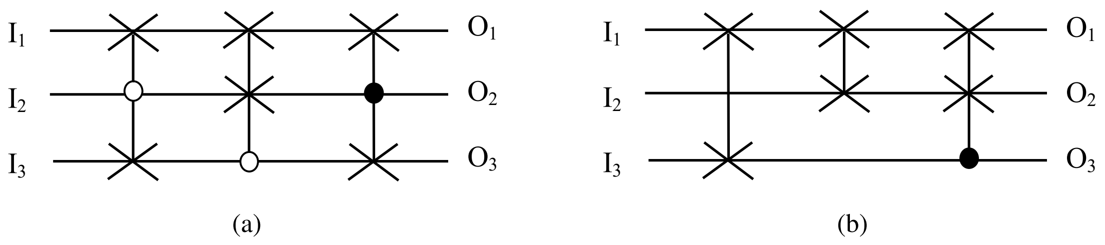


Figure 2.6: Two different realizations of the same function using the SF gate family.

Similarly a higher gate count for a circuit does not necessarily mean a higher quantum cost. Consider the 3×3 reversible function shown in Table 2.8. Figure 2.6 shows two different implementations of this function using only SWAP and Fredkin gates. Both implementations have a gate count of 3. The realization in Figure 2.6(a) uses two negative-controlled Fredkin gates and one positive-controlled Fredkin gate. The quantum cost of the first implementation is $(5+5+5) = 15$. The circuit in the second implementation, shown in Figure 2.6(b), uses two SWAP gates and one 3×3 positive-controlled Fredkin gate. Here the quantum cost is $(3+3+5) = 11$. Both circuits consist of 3 logic gates; however, the second circuit offers a more efficient circuit design, from the perspective of the quantum cost, than the first circuit.

2.5 Faults and Fault Testing in Reversible Logic

A fault leads to a failure of a system. A fault can be defined as an imperfection within some hardware and/or software components. Therefore, a fault leads the system to produce an incorrect output. This incorrect output is known as an error, which is a deviation from the accurate outcome [26]. Faults are the unwanted events for a system and, therefore, must be detected and removed.

A fault model is a model that considers the potential faults which may occur in a system [59]. An ideal fault detection method can detect all the faults of a particular fault model. A common fault model in traditional logic is the stuck-at fault model. Sometimes a wire can pass only high or low voltage signals due to the malfunction of some part of a circuit. That is, a path of a circuit is "*stuck at*" a particular voltage level. When a line of a circuit passes only high voltage due to the occurrence of a fault, the fault is considered to be a stuck-at 1 fault. When a line of a circuit passes only low voltage, the fault is considered to be a stuck-at 0 fault. As a consequence the circuit generates an incorrect output. For example, if a line is broken, it is considered open and the output of that line will always be 0. The fault in this situation is refered to be stuck-at 0. However, research suggests that the stuck-at fault model is not an appropriate fault model in reversible computing [20, 73]. A number of fault models for reversible logic have been proposed [20, 56, 73].

A method of testing is required in order to identify the occurrence of a fault in a circuit. Testing indicates whether a system is faulty or fault-free. Fault coverage is an important concept in testing which relates fault models to testing strategies. Fault coverage is a ratio of the number of faults detected by a testing method to the total number of detectable faults for a given fault model. Methods of testing in reversible computation are divided into two categories: offline testing and online testing [59]. Each has its own benefits and drawbacks. Online testing approaches apply the testing operations in real time. That is, online testing methods determine whether the output of a system is correct or incorrect while the system is performing its normal operations. However, this is not the case for an offline testing method. In offline testing, the system under consideration is taken out of its normal mode of operation, and then the method of testing is applied. Fault models and testing approaches in reversible logic are described in detail in Chapter 4 of this thesis.

2.6 Fault Tolerance in Reversible Logic

The difference between fault testing and fault tolerance is that fault testing is a process of error identification whereas fault tolerance is a process of error correction. Fault tolerance is an attribute that enables a system to generate correct output even in the presence of faults in the system. One or more techniques may be required in order to achieve fault tolerance. These may include fault detection, fault diagnosis, fault containment, fault recovery and fault masking. Each of these techniques requires additional logical or physical components of the system such as hardware, software and/or information [26]. For example, one way to achieve fault tolerance in a system is to replicate one or more physical components of the system. The techniques and approaches that can be used to achieve fault tolerance in reversible circuits are described in Chapter 5.

Chapter 3

Synthesis and Post-Synthesis

This chapter begins with providing the basic concepts of synthesis in reversible logic. A transformation-based synthesis is proposed in this chapter. This chapter also introduces templates for optimizing reversible circuits. Experimental results that evaluate the proposed templates are provided at the end of the chapter.

3.1 Logic Synthesis in Reversible Logic

Logic synthesis is the process of generating a circuit design, described as a cascade of gates, that can implement the desired logic function. The relationship between the inputs and the outputs of a logic function determines the number of the logic gates, type of logic gates used, and the order in which the logic gates appear in the circuit. If a logic function is already reversible, the synthesis process can take place immediately. However, if a logic function is not reversible, the first step in most synthesis algorithms is to transform the irreversible function into a reversible one. One or more garbage outputs and/or constant inputs are added to an irreversible function in order to transform the irreversible logic function into a reversible logic function. Section 2.3 in Chapter 2 shows an example of transforming an irreversible AND function into a reversible function. In most cases, a reversible circuit design with fewer garbage outputs and/or constant inputs is considered a desirable design. The minimum number of garbage outputs which are required in order to transform an irreversible function into a reversible function is $\lceil log_2 K \rceil$, where K is the maximum number of a repeated pattern in the output of an irreversible function [35].

A number of logic synthesis techniques in reversible logic have been proposed as described in [60]. In this work we focus on transformation based logic synthesis.

3.1.1 The Transformation Based Synthesis Approach [41]

A transformation based approach takes as its input a truth table of a reversible function, and applies reversible logic operations to transform the function into an identity function. The gates which perform these logic operations during the tranformation constitute the circuit that implements the input reversible function. The gates appear in the circuit in the same order in which the logical operations are performed during transformation. Before the synthesis takes place if a function is not reversible, the first step is to transform the irreversible function into a reversible function. Works such as proposed by Maslov et al. [35] and Miller et al. [42] describe techniques for this. One of the major advantages of a transformation-based synthesis approach is that the process of generating circuits based on this approach does not create any garbage output or constant input lines. Thus, in terms of the number of inputs and outputs lines, the size of the circuit generated by a transformation based synthesis is minimal.

The transformation based synthesis algorithm was proposed by Miller et al. [41]. The authors demonstrated two variations: a basic algorithm and a bidirectional algorithm, both based on the NCT gate library. In the basic algorithm, the reversible logic operations are applied to the output of the function's truth table. We assume that we are applying the algorithm to a reversible function of n variables. The objective is to make $f(i) = i$, for $i = 0$ to $2^n - 1$, where i is the i-th row of a reversible function, f. The following is the basis of the transformation based logic synthesis approach (from [41]):

Step 0: If $f(0) = 0$, no transformation is required and go to step 1. Otherwise, if $f(0) \neq 0$, apply one or more (1×1) Toffoli gates (i.e. NOT gates) in order to achieve $f(0) = 0$. After applying the NOT gate(s), the value 000 will be at the top row of the output truth table, as shown in Table 3.1.

Step 1: Repeat for $i = 1$ to $2^n - 1$: If $f(i) = i$, no transformation is required. If $f(i) \neq i$, apply the smallest $(k \times k)$ Toffoli gate, $k = 2$ to n in order to make $f(i) = i$. One or more gates may be required in order to achieve $f(i) = i$.

The choice of a gate during each step of the transformation is crucial in order to maintain convergence. The gate in one step of transformation must not change the bits of the previous steps. Consider the (3×3) reversible function shown in Table 3.1. The transformation based synthesis transforms this function to an identity function.

Table 3.1: Truth table of a (3×3) reversible function.

	input			output			
	a_i	b_i	c_i	a_o	b_o	c_o	
(0)	0	0	0	0	0	0	(0)
(1)	0	0	1	1	0	0	(4)
(2)	0	1	0	0	0	1	(1)
(3)	0	1	1	0	1	1	(3)
(4)	1	0	0	0	1	0	(2)
(5)	1	0	1	1	0	1	(5)
(6)	1	1	0	1	1	0	(6)
(7)	1	1	1	1	1	1	(7)

Table 3.2 shows the entire transformation process. Recall that the basic algorithm works on the outputs of the function. The goal is to ensure that for each i, we map $f(i) \longrightarrow i$, for $i = 0$ to $(2^n - 1)$ where n is the number of bits. At each stage, if a transformation is required, it is necessary to apply one or more logic gates in order to map $f(i) \longrightarrow i$. The bottom row of each column in Table 3.2 indicates the logic gate which is applied at the corresponding stage of transformation. Applying a gate to the function of one stage generates the value used for the next stage. Note that the notation $T(a;b)$ indicates a 1-CNOT gate with a control on line 'a' and a target on line 'b'. Similarly $T(a,b;c)$ indicates a (3×3) Toffoli gate, where the target is on the line 'c' and the controls of the gate are on input lines 'a' and 'b'. Applying the transformation based approach to the function shown in Table 3.1 takes place as follows:

Step 0: As shown in Table 3.2, the value at the first row of the output is $a^0b^0c^0 = 000$, which is already in its proper position (0-th position). Hence, no transformation is required for the first row.

Step 1: For the next row of the truth table, we need to map $f(100) \longrightarrow 001$. Applying two NOT gates in the a-th and c-th positions in order to transform $(a,b,c) \equiv 100$ to 001 does not work in this case, because the use of NOT gates would also invert the value of the first row of the truth table, which is not permitted. In this case two stages are required in order to achieve the desired transformation. A CNOT gate $T(a;c)$ can be applied, which transforms 100 to 101. Another CNOT gate $T(c;a)$ is required to apply in order to transform 101 to 001. Stages (i) and (ii) in Table 3.2 illustrate this step.

Step 2: The next required mapping is $f(101) \longrightarrow 010$. We need three CNOT gates $T(a;c)$, $T(a;b)$ and $T(b;a)$ in order to map 101 to 010. As we see from stages (iii), (iv) and (v), the three CNOT gates convert 101 to 010. Again, these three CNOT gates do not alter any of the previous rows of the truth table. However, the bits in the rows which are below the third row are changed.

Step 3: In this step we have 100 in the fourth row of the truth table in place of 011. Thus, the required mapping in this step is $f(100) \longrightarrow 011$. As similar to the previous step, this step also requires three stages of transformation in order to map $f(100) \longrightarrow 011$. The sequence of $T(a;c)$, $T(a;b)$ and $T(b,c;a)$ is used to transfer 100 into 011.

Step 4: At this step the fifth row of the truth table has a value of 101. The required mapping at his stage is $f(101) \longrightarrow 100$. A single gate $T(a;c)$ can be applied to perform this mapping, as we see from stage (ix) of Table 3.2.

Step 5: The required mapping at this step is $f(110) \longrightarrow 101$. We need to apply two (3×3) Toffoli gates $T(a,b;c)$ and $T(a,c;b)$ in order to map 110 into 101.

Step 6: At this step the values of the last two rows must be exchanged. We need to map $f(111) \longrightarrow 110$. A single (3×3) Toffoli gate $T(a,b;c)$ transforms the bits in the required places. At the end of this step, as shown in Table 3.2, for every possible i, the relationship $f(i) \longrightarrow i$ holds and the transformation process terminates its execution.

Table 3.2: Transformation stages of the function from Table 3.1.

(a)

			(Step 0)			(Step 1)						(Step 2)								
output			(i)			(ii)			(iii)			(iv)			(v)			(vi)		
a^0	b^0	c^0	a^1	b^1	c^1	a^2	b^2	c^2	a^3	b^3	c^3	a^4	b^4	c^4	a^5	b^5	c^5	a^6	b^6	c^6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	1	1	1	1	1	0	1	0	0	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0
1	0	1	1	0	1	1	0	0	1	0	0	1	0	1	1	1	1	0	1	1
1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1	0	1
			No Transformation			T(a;c)			T(c;a)			T(a;c)			T(a;b)			T(b;a)		

(b)

(Step 3)									(Step 4)			(Step 5)						(Step 6)		
(vii)			(viii)			(ix)			(x)			(xi)			(xii)			(xiii)		
a^7	b^7	c^7	a^8	b^8	c^8	a^9	b^9	c^9	a^{10}	b^{10}	c^{10}	a^{11}	b^{11}	c^{11}	a^{12}	b^{12}	c^{12}	a^{13}	b^{13}	c^{13}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
1	0	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0
0	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	0	1
1	1	0	1	0	0	1	0	0	1	0	1	1	0	1	1	1	1	1	1	0
1	0	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1	1
T(a;c)			T(a;b)			T(b,c;a)			T(a,c)			T(a,b;c)			T(a,c;b)			T(a,b;c)		

The circuit generated by the process, as shown in Table 3.2, is presented in Figure 3.1. The entire transformation process requires 8 CNOT gates and 4 Toffoli gates. Thus, the gate count, GC, is 12. Using the table in section 2.4, we can compute the quantum cost, QC, is $((8\times 1)+(4\times 5)) = 28$. The next subsection describes our proposed algorithm. We also show the circuit realization for the same function using our proposed approach.

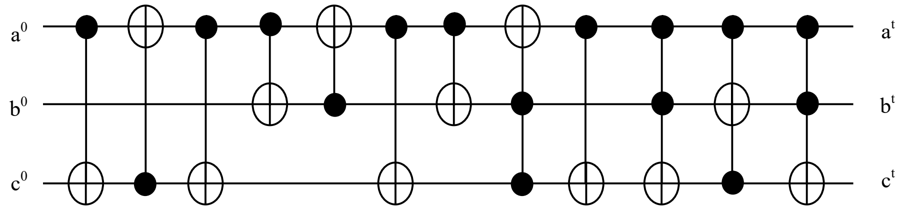


Figure 3.1: The circuit obtained from the function shown in Table 3.1.

3.1.2 A Proposed Transformation Based Synthesis Approach

Before describing our proposed approach, it is important to observe a significant property of the function shown in Table 3.1. The truth table of the function shows that for each row, the number of 1s in the input is equal to the number of 1s in the output. Thus, the function is a conservative function.

Our hypothesis is that a circuit realization for a conservative reversible function will be more efficient if we use SF gates instead of NCT gates. Thus we propose a SF gate based transformation approach [47]. The underlying idea of SF-based transformation synthesis is the same as the approach described previously in this chapter. The difference is that instead of using the logic gates from the NCT gate family, we use only SWAP and Fredkin gates to realize the transformations. While NCT gates manipulate bits by inverting the bits, SF gates interchange the bits when the control points of these gates satisfy the necessary condition. Since a conservative function has an equal number of 1s in the input and the output, our hypothesis is that the interchange of bits rather than the inversion of bits during the process of transformation will generate efficient circuits. The transformation based approach involves the mapping between two values consisting of the same number of 1s. The output of a SF gate consists of the same number of 1s as the input. Thus, the SF gates will be more suitable and require fewer logical operations than NCT gates for mapping one value into another of a conservative function.

As in the NCT version, the proposed approach examines one row of the truth table at each step. The objective is to make $f(i) = i$, for $i = 0$ to $2^n - 1$, where i is the i-th row of a reversible function f, and n is the number of inputs/outputs (bits) of the function. The following is the basis of SF gate base transformation approach.

Step 0: Since the function is a conservative function, the first row of the truth table of the function will be $f(0) = 0$. Thus, no transformation is required and go to step 1.

Step 1: For $i = 1$: If $f(1) = 1$, no transformation is required. If $f(1) \neq 1$, apply a SWAP gate in order to make $f(1) = 1$.

Step 2: Repeat for $i = 2$ to $2^n - 1$: If $f(i) = i$, no transformation is required. If $f(i) \neq i$, apply a SWAP gate or the smallest $(k \times k)$ Fredkin gate in order to make $f(i) = i$, where $k = 3$ to n. One or more gates may be required in order to achieve $f(i) = i$.

We use the same function from Table 3.1 to demonstrate the SF-based transformation synthesis. As before, the proposed approach begins with the output of the function. Table 3.3 shows the transformation stages. In this table $S(a,b)$ represents a SWAP gate with two targets, 'a' and 'b'. $F(a;b,c)$ represents a Fredkin gate with a control point on line, 'a', and two targets on lines 'b' and 'c'.

Step 0: The first row of the function shown in Table 3.1 is $f(000) = 000$, thus no transformation is required.

Step 1: The necessary transformation in the second row of the table is $f(100) \longrightarrow 001$. We need to map 100 to 001. A single SWAP gate, $S(a,c)$, can be used for this mapping. This SWAP gate interchanges the values of 'a' and 'c'. If we compare with the synthesis approach illustrated in Table 3.2, the NCT based transformation requires two stages (two gate levels) for the same mapping.

Step 2: After applying a SWAP gate at the second step, the value at the third row is 100. The required mapping in this step is $f(100) \longrightarrow 010$. Thus, we must transform 100 into 010. A SWAP gate, $S(a,b)$, can be used for the required mapping.

Step 3: At this step of transformation, the required mapping is $f(110) \longrightarrow 011$. A SWAP gate, S(a,c), could transform 110 to 011; however this mapping would also change the bits in the previous rows. One of the basic concepts of the transformation based synthesis is that a mapping in one step must not alter any of the previous rows. Hence, a (3×3) Fredkin gate can be used for the required mapping. We use a Fredkin gate, $F(b;a,c)$, which swaps the two target bits 'a' and 'c', when the control bit 'b' $= 1$.

Step 4: At this step the function in the fifth row is $f(100) = 100$, so no transformation is required. However, the required transformation in the sixth row is $f(110) \longrightarrow 101$. A (3×3) Fredkin gate $F(a;b,c)$ swaps the values of 'b' and 'c' when the value of 'a' is

1. By applying the Fredkin gate $F(a;b,c)$ at this step, the entire function is transformed to an identify function. The resulting circuit realization of the function from Table 3.1 is displayed in Figure 3.2.

Table 3.3: Stages of SF based transformation of the function in Table 3.1.

			step 0			step 1			step 2			step 3			step 4		
output			i			(ii)			(iii)			(iv)			(v)		
a	b	c	a^0	b^0	c^0	a^1	b^1	c^1	a^2	b^2	c^2	a^3	b^3	c^3	a^4	b^4	c^4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	0	1	0
0	1	1	0	1	1	1	1	0	1	1	0	0	1	1	0	1	1
0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0	1	0	1	1	1	1	0	1	0	1
1	1	0	1	1	0	0	1	1	1	0	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
			No Transformation			S(a,c)			S(a,b)			F(b;a,c)			F(a;b,c)		

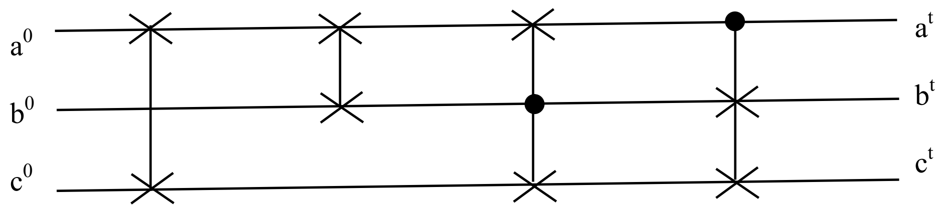


Figure 3.2: Circuit resulting from SF based synthesis of the function from Table 3.1.

3.1.3 Comparison of Transformation Based Approaches

Figures 3.1 and 3.2 show two circuits for the same function from Table 3.1. The former circuit is realized following the NCT gates transformation based approach from [41], while the latter circuit realization is generated by our SF based transformation approach. In Figure 3.2, we have a GC of 4 as compared to a GC of 12 for the circuit in Figure 3.1. The SF gate based transformation also performs better than the NCT based synthesis from the perspective of QC. The QC of the new implementation is $(2 \times 3) + (2 \times 5) = 16$, while the QC for the circuit realization in Figure 3.1 is 28. Table 3.4 shows the comparison between the two circuits shown in Figures 3.1 and 3.2.

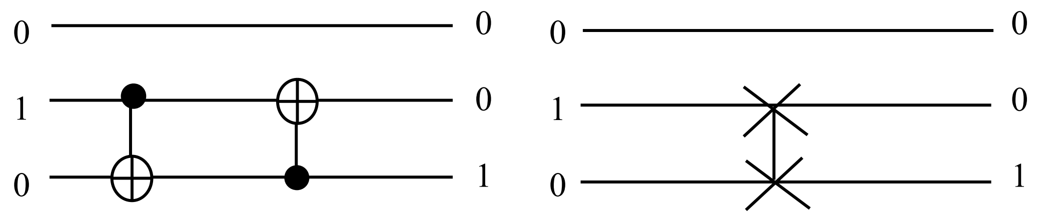
In order to compare the SF gate based transformation approach with the NCT gate based transformation approach from a wider perspective, we have generated all possible (3×3)

Table 3.4: Comparison between two circuits shown in Figures 3.1 and 3.2.

Metric	NCT	SF
GC	12	4
QC	28	16

conservative reversible functions. There are 36 conservative (3×3) reversible functions in total. We have realized all the 36 conservative (3×3) reversible functions using both the NCT based transformation approach and the proposed SF based transformation approach. The performance was evaluated in terms of GC and QC. Table 3.5 shows the results. After observing the entire table, the highest percentage of reduction in GC is 67. We achieve this reduction in GC for more than half of the (3×3) conservative reversible functions. The percentage of reduction in GC on average is 62%.

As we see from Table 3.5, the SF gate family based transformation synthesis approach performs extremely well compared to the other approach as far as GC is concerned. One of the reasons behind the performance improvement is that SF gates need fewer operations to map one value to another of a conservative function. Since the function is conservative, the mapping involves two bit combinations that have the same number of 1's and 0's. In general this type of transformation can be carried out with fewer operations if we exchange the bits of a row instead of inverting the bits. For example, in order to transform 010 into 001, we have two possible circuit realizations, as shown in Figure 3.3. The ability to change two bits at a time makes the SF gates more efficient than the NCT gates for realizing conservative reversible circuits.



(a) using NCT gate realization.

(b) using SF gate realization.

Figure 3.3: Circuit realization using NCT and SF gate families.

Table 3.5: Performance comparison of NCT gate transformation based synthesis and SF gate transformation based synthesis.

	NCT gate transformation		SF gate transformation		Reduction		Percentage of decrease	
No.	GC	QC	GC	QC	GC	QC	GC	QC
1	9	25	3	13	6	12	66.67	48.00
2	6	10	2	8	4	2	66.67	20.00
3	9	25	3	13	6	12	66.67	48.00
4	6	10	2	8	4	2	66.67	20.00
5	3	3	1	3	2	0	66.67	0.00
6	6	18	2	8	4	10	66.67	55.56
7	7	11	3	11	4	0	57.14	0.00
8	10	26	4	16	6	10	60.00	38.46
9	9	21	3	11	6	10	66.67	47.62
10	6	6	2	6	4	0	66.67	0.00
11	7	11	3	11	4	0	57.14	0.00
12	10	26	4	16	6	10	60.00	38.46
13	9	25	3	13	6	12	66.67	48.00
14	6	10	2	8	4	2	66.67	20.00
15	3	3	1	3	2	0	66.67	0.00
16	6	18	2	8	4	10	66.67	55.56
17	7	23	3	13	4	10	57.14	43.48
18	4	8	2	8	2	0	50.00	0.00
19	9	21	3	11	6	10	66.67	47.62
20	6	6	2	6	4	0	66.67	0.00
21	7	11	3	11	4	0	57.14	0.00
22	10	26	4	16	6	10	60.00	38.46
23	9	13	3	11	6	2	66.67	15.38
24	12	28	4	16	8	12	66.67	42.86
25	3	3	1	3	2	0	66.67	0.00
26	6	18	2	8	4	10	66.67	55.56
27	9	25	3	13	6	12	66.67	48.00
28	6	10	2	8	4	2	66.67	20.00
29	7	23	3	13	4	10	57.14	43.48
30	4	8	2	8	2	0	50.00	0.00
31	3	7	1	5	2	2	66.67	28.57
32	6	22	2	10	4	12	66.67	54.55
33	3	7	1	5	2	2	66.67	28.57
34	6	22	2	10	4	12	66.67	54.55
35	3	15	1	5	2	10	66.67	66.67
36	0	0	0	0	0	0	0	0

Table 3.5 shows that the SF based synthesis also performs better than the NCT based synthesis from the perspective of QC. SF based synthesis results in achieved lower QC for almost 70% of the 36 functions. For the remaining 30% functions, the QC is the same for both approaches. There is not a single instance where the NCT based synthesis performs better than our proposed approach. Indeed in one case there is a decrease in QC of 67%.

However, the average percentage of reduction of quantum cost using the SF based synthesis is 29%.

As mentioned above, the proposed transformation based synthesis using the SF gate family follows a greedy approach. We have designed our algorithm in this way because the basic transformation based synthesis algorithm from [41] also follows a greedy approach, and this allow us to make a fair comparison. At every step of transformation, the algorithm selects the lowest cost gate in terms of quantum cost. When there is a choice between a SWAP gate and a Fredkin gate in order to make a transformation happen, the algorithm selects a SWAP gate, since a SWAP gate has lower quantum cost than a Fredkin gate. For example, if we observe the second column (i) of Table 3.3, we need to transform 100 into 010. There are two choices for this mapping. We could use either a SWAP gate $S(a,b)$ or a negative controlled Fredkin gate, $F^{'}(c;a,b)$. A SWAP gate $S(a,b)$ exchanges the bits of 'a' and 'b'. A negative controlled Fredkin gate, $F^{'}(c;a,b)$, swaps the values of 'a' and 'b' when $c = 0$. Either of the two gates can serve the purpose at this stage. However, the proposed SF gate based transformation selects the SWAP gate $S(a,b)$ in this case, because a SWAP gate has lower quantum cost than a Fredkin gate. However, if we use a $F^{'}(c;a,b)$ at this stage, we get the circuit presented in Figure 3.4(a). The use of $F^{'}(c;a,b)$ gate reduces the QC from 16 to 13 compared with the circuit in Figure 3.2. In addition, one less gate is needed in this circuit realization. The interesting fact is that the circuit in Figure 3.4(a) can be simplified further. The choice of gate is one of the crucial factors in order to synthesize an efficient circuit. The circuit represented in Figure 3.4(b) is a further simplified version of the circuit shown in Figure 3.4(a). Figure 3.4(b) shows that the GC is 2 for this circuit. The QC for this circuit is $5+5 = 10$. Now if we compare the GC and QC of the circuit presented in Figure 3.4(b) with that of the NCT gate based basic transformation synthesis (Figure 3.1), the GC is reduced from 12 to 2, which is a six-fold reduction. The QC is reduced from 28 to 10, which results in almost 3 times the improvement in QC of a circuit which is realized using the SF gate based transformation synthesis as compared with that of NCT gate based transformation synthesis approach.

Thus we see that the SF gate based transformation approach performs much better than the NCT gate based transformation for all (3×3) conservative reversible functions. In addition, performance of the SF gate based transformation can be improved further if the selection of gate at each stage can be done intelligently rather than following a greedy approach.

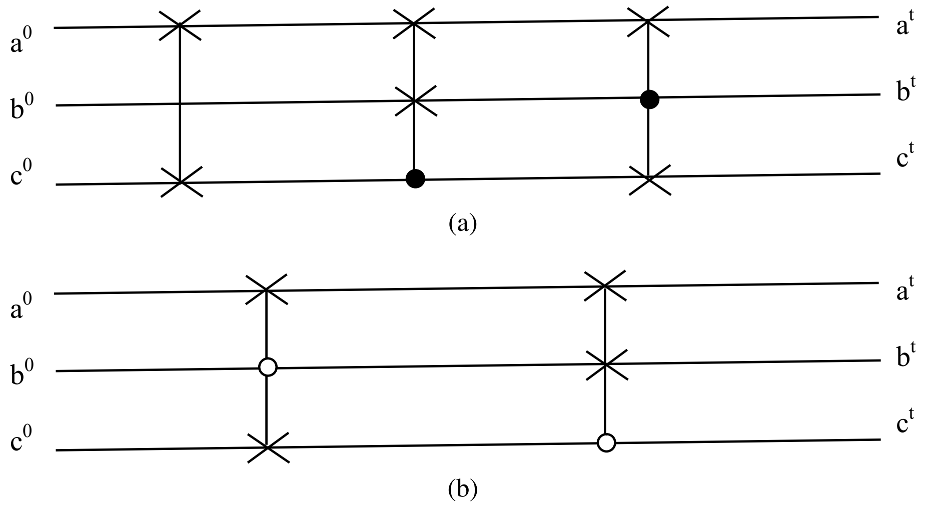


Figure 3.4: Further simplified circuits for the function shown in Table 3.1.

We have also compared the results from our proposed SF-based approach with the results from applying an exact synthesis approach [17] available in RevKit [67]. The comparison is shown in Table 3.6. The exact approach results in a circuit implementation with minimal gate count using the NCT gate library. There is no known exact approach that uses the SF gate library. There is not a single instance where the exact synthesis generates circuits with lower GC than the SF based transformation approach. The highest percentage of reduction in GC is 67%. However, the average percentage of reduction of GC using the SF based synthesis is 54%. The negative values in the table indicate the increment in QC using SF based synthesis over exact synthesis. The QC increases in the case of 10 functions out of all 36 conservative functions. The highest percentage of reduction in QC using our proposed synthesis approach is 67% and the percentage of reduction in QC on

Table 3.6: Performance comparison of the minimal circuits generated using exact synthesis [67] with the circuits generated using SF gate based synthesis.

	Exact Synthesis		SF gate synthesis		Reduction		Percentage of decrease	
No.	GC	QC	GC	QC	GC	QC	GC	QC
1	6	14	3	13	3	1	50.00	7.14
2	6	10	2	8	4	2	66.67	20.00
3	6	14	3	13	3	1	50.00	7.14
4	6	10	2	8	4	2	66.67	20.00
5	3	3	1	3	2	0	66.67	0.00
6	4	8	2	8	2	0	50.00	0.00
7	6	10	3	11	3	-1	50.00	-10.00
8	7	15	4	16	3	-1	42.86	-6.67
9	6	10	3	11	3	-1	50.00	-10.00
10	6	6	2	6	4	0	66.67	0.00
11	6	10	3	11	3	-1	50.00	-10.00
12	7	15	4	16	3	-1	42.86	-6.67
13	6	14	3	13	3	1	50.00	7.14
14	6	10	2	8	4	2	66.67	20.00
15	3	3	1	3	2	0	66.67	0.00
16	6	10	2	8	4	2	66.67	20.00
17	6	14	3	13	3	1	50.00	7.14
18	4	8	2	8	2	0	50.00	0.00
19	6	10	3	11	3	-1	50.00	-10.00
20	6	6	2	6	4	0	66.67	0.00
21	6	10	3	11	3	-1	50.00	-10.00
22	7	15	4	16	3	-1	42.86	-6.67
23	6	10	3	11	3	-1	50.00	-10.00
24	7	15	4	16	3	-1	42.86	-6.67
25	3	3	1	3	2	0	66.67	0.00
26	6	10	2	8	4	2	66.67	20.00
27	6	14	3	13	3	1	50.00	7.14
28	6	10	2	8	4	2	66.67	20.00
29	6	14	3	13	3	1	50.00	7.14
30	4	8	2	8	2	0	50.00	0.00
31	3	15	1	5	2	10	66.67	66.67
32	4	20	2	10	2	10	50.00	50.00
33	3	7	1	5	2	2	66.67	28.57
34	4	20	2	10	2	10	50.00	50.00
35	3	7	1	5	2	2	66.67	28.57
36	0	0	0	0	0	0	0	0

average is 8%. The reduction in GC can be explained by the fact that SF gates require fewer operations to implement swaps, which are the main operations carried out in conservative functions. However in general, SF gates have higher QC than their NCT equivalents, so there is less saving in QC. For example, a SWAP gate $S(b,c)$ can be used in order to transform $(a,b,c) = 010$ into 001. The GC and the QC for a single SWAP gate are 1 and 3 respectively. However, two NCT gates, $T(b;c)$ and $T(c;b)$, will be required in order to transform $(a,b,c) = 010$ into 001. In this case GC and QC are 2. Thus the GC is reduced by using the SF gate family, while the QC (for this example) is not.

We have also generated all possible (4×4) conservative functions. There are 414720 (4×4) conservative reversible functions. We have investigated the circuit realization for each of these functions using both the SF gate based synthesis and the NCT gate based synthesis. However, unlike the case of (3×3) functions, there are some circuit realizations where the GC and QC increase with the SF gate based transformation synthesis as compared to that of the NCT gate based synthesis. Table 3.7 shows the amount of reduction in GC and QC using the SF-based transformation instead of the NCT-based transformation. Table 3.7(b) shows that among all the 414720 (4×4) conservative reversible functions, the QC increases for 27213 functions and the GC increases for 2 functions (in one of these the GC increases only by 1). The QC reduces for almost 93% of the 414720 functions. The highest reduction in GC by using the SF gate based transformation as compared with that of the NCT gate based transformation synthesis is 27. The highest percentage of reduction in GC using our proposed synthesis approach is 87% and the percentage of reduction in GC on average is 61%. By using the SF gate based transformation synthesis we get a highest reduction in QC of 104, whereas the highest percentage of reduction of QC is 87%. For all (4×4) functions, the average percentage of decrease of QC is 35%.

3.2 Post Synthesis Optimization

A reversible circuit generated by a synthesis method may not be optimal from the perspective of the number of garbage lines, quantum cost and/or gate count. For example, Figure 3.2 shows a circuit generated by transformation based synthesis. This circuit can be simplified further, as shown in Figure 3.4. A circuit design that offers fewer garbage lines and/or lower GC and QC is desirable. After synthesis takes place, several strategies

Table 3.7: Percentage of reduction in GC and QC when using SF based transformation as compared to NCT based transformation.

(a) Best results.

	NCT	SF	Reduction	% Decrease
GC	15	2	13	86.67
QC	102	13	89	87.25

(b) Average results.

Reduction on average		No. of functions: 414720		
	% decrease	Decrease	Increase	% Decrease
GC	61	414706	2	99.99
QC	35	385650	27213	92.99

can be used in order to simplify reversible circuits, including template matching optimization [1, 9, 37, 58] and rule based optimization [2, 24].

A template consists of two patterns of gates which are equivalent to each other. Template matching is a process to find a pattern of gates that can be replaced by another equivalent pattern of gates in order to simplify a circuit design. Figure 3.5 shows two templates

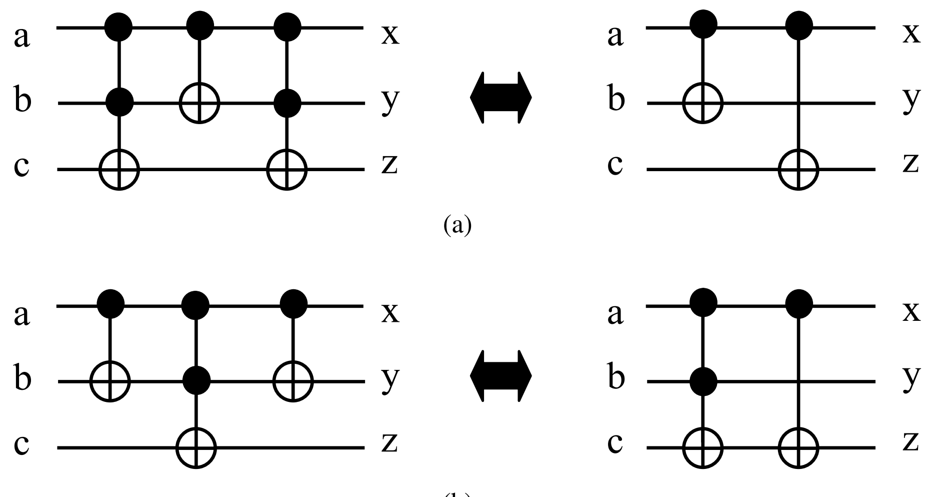


Figure 3.5: Two templates presented in [41].

presented in [41]. The output functions of both circuits in Figure 3.5(a) are evaluated as $x = a$, $y = a \oplus b$ and $z = a \oplus c$. Thus, these two circuits perform the same reversible function.

The left hand circuit has a GC of 3 and QC of 11. However, both the GC and QC of the right hand circuit are 2. Therefore, the right hand circuit design is more efficient in terms of GC and QC. Since both circuits perform the same function, it is preferable to use the right hand circuit in a circuit design. Miller et al. also introduce other templates for 2 and 3 input reversible circuits as well as a template matching algorithm [41]. This algorithm searches for a pattern of gates in a reversible circuit and replaces the pattern by another simpler pattern of gates. An extension of this algorithm is presented in [38]. Maslov et al. introduced some templates based on Toffoli and Fredkin gates in [36]. Templates based on both positive and negative controls are presented by Datta et al. [8] and Rahman et al. [58]. Iwama et al. also present rules which can be used to simplify reversible circuits [24]. Other rule based post synthesis optimization works include [2, 7].

Most template matching and rule based optimization techniques focus on reversible circuits based on either NCT gates or a combination of Toffoli and Fredkin gates. In this dissertation, we present optimization techniques for SF circuits based on template matching and rule-based simplifications. Some optimization rules designed for NCT gate families can also be used for SF circuit simplification. For example, the deletion rule can be used as an optimization technique for SF-based circuits. The deletion rule [7, 24, 38] states that two adjacent gates with the same target and control lines do not contribute to the functionality of the circuit. For example, consider the circuit shown in Figure 3.6. The output function of this circuit can be evaluated as $x = a$, $y = b$ and $z = ab \oplus c \oplus ab = c$. It is seen that the circuit is simply passing the input values to the corresponding output lines. So the two 3-bit Toffoli gates can be removed from the circuit.

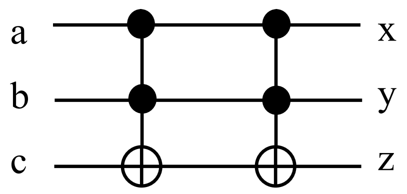


Figure 3.6: An example to demonstrate the deletion rule in a reversible circuit.

The deletion rule for NCT-based circuits can also be used to optimize SF circuits. However, not all NCT-based rules are useful for optimizing SF based circuits. For example, the moving rule proposed for NCT gates is a useful approach for simplifying reversible circuits. The moving rule states that two adjacent gates $g_1(c_1,t_1)$ and $g_2(c_2,t_2)$ can be interchanged if the target of one gate is not a control of another gate, i.e. $c_1 \cap t_2 = \emptyset$ and $c_2 \cap t_1 = \emptyset$.

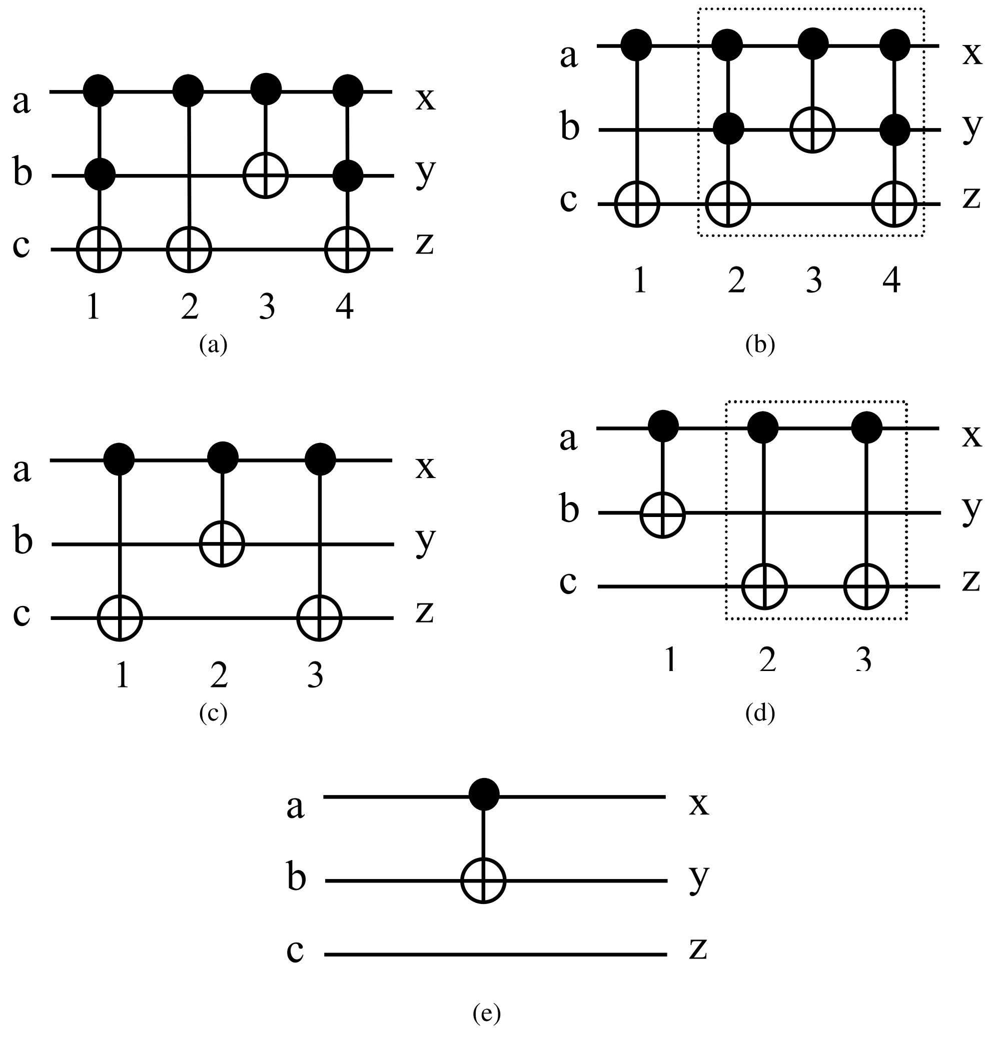


Figure 3.7: An example to demonstrate the moving rule for circuit simplification.

The moving rule is particularly useful in order to find a template in a circuit. For example, Figure 3.7 shows an example of the moving rule for NCT based circuit simplification. Figure 3.7(a) shows how the positions of gates labeled 1 and 2 can be interchanged, as the controls of gate 1 are not the target of gate 2 and vice versa. Figure 3.7(b) shows the

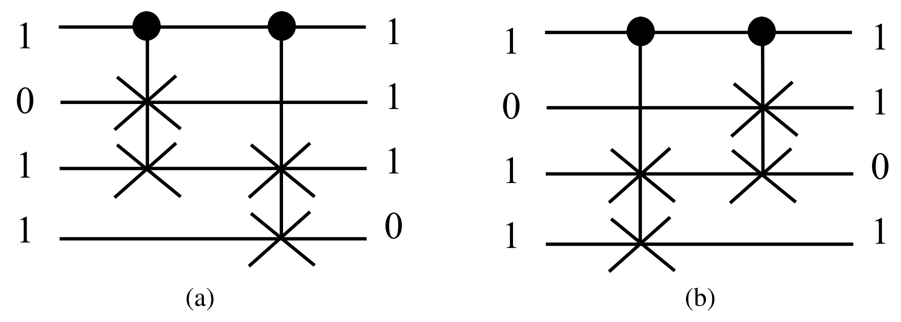


Figure 3.8: A SF circuit where the moving rule does not work.

circuit after interchanging gates 1 and 2. The three gates enclosed by a box match with template presented in Figure 3.5(a). After substituting the gates according to the template, the circuit becomes as shown in Figure 3.7(c). Gate 1 and gate 2 in Figure 3.7(c) satisfies the moving rule and after exchanging these two gates the circuit becomes as presented in Figure 3.7(d). Gates 2 and 3 in Figure 3.7(d) can be deleted according to the deletion rule. Figure 3.7(e) shows the circuit after applying templates. The moving rule plays a significant role in applying simplification rules and templates for NCT based circuit optimization.

However, this moving rule works only on NCT based reversible circuits. Figure 3.8 shows an example of a SF based circuit where the moving rule cannot be applied. According to the moving rule, two gates can be interchanged if controls of one gate are not the target of other gate. Figure 3.8(a) shows a circuit that consists of two 3-bit Fredkin gates. Since the control of one gate is not a target of another gate, the two gates are interchanged as shown in Figure 3.8(b). However, these two circuits are not equivalent. As shown in the figure, the two circuits generate different outputs for the same input vector. In this dissertation, we have modified the moving rule for applications in SF based reversible circuits. The modified moving rule is presented in section 3.2.2.

3.2.1 Proposed Templates

Most existing reversible circuit optimization techniques focus on NCT gates. This section presents templates for SF gate based reversible circuits. We consider both template matching and rule based simplification for circuit optimization. The basic difference between rule based simplification and template matching is that templates must match specific patterns of gates, while rules can be applied to a broad group of gates. For example, the deletion rule shown in Figure 3.6 is also true for n-bit gates. Template matching, on the other hand, includes two patterns of gates that may not be true for other n-bit gates. For example, our proposed Template 5, discussed later in this section, is an example of template matching optimization. For better understanding we refer to both templates and rules as templates in this dissertation. Note that $G(C;T)$ represents a gate G from the SF gates family. C and T represent the sets of the control points and the targets of G, respectively.

Template 1

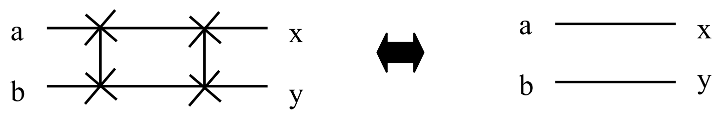
Two adjacent gates $G_1(C_1;T_1)$ and $G_2(C_2;T_2)$ can be removed from a circuit if $C_1 = C_2$ and $T_1 = T_2$. That is, if targets of a SWAP gate are on the same line as that of an adjacent SWAP gate, the two SWAP gates can be removed from the circuit. In case of a Fredkin gate, when the controls and targets of two adjacent gates are the same in polarity and operate on the same line, the two Fredkin gates have no effect on circuit operation.

$$SWAP(t_1,t_2)SWAP(t_1,t_2) \equiv I \tag{3.1a}$$

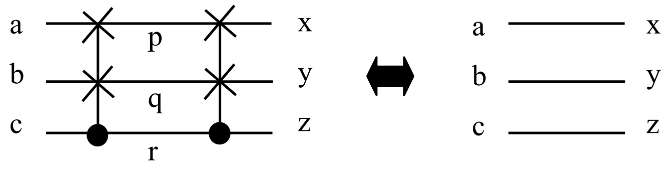
$$FRED(c;t_1,t_2)FRED(c;t_1,t_2) \equiv I \tag{3.1b}$$

$$FRED(\overline{c};t_1,t_2)FRED(\overline{c};t_1,t_2) \equiv I \tag{3.1c}$$

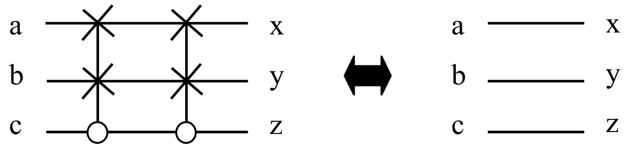
Figure 3.9 shows Template 1 and its variations based on different SF gates. Equation 3.1 shows the expressions for the templates shown in this figure. As it is seen, the circuits work



(a) SWAP gates.



(b) Positive control Fredkin gates.



(c) Negative control Fredkin gates.

Figure 3.9: Template 1.

as identity circuits, which simply pass the inputs to the outputs. For example, suppose the outputs of the first Fredkin gate in Figure 3.9(b) are p, q and r which operate on lines a, b and c respectively. The output functions of the first Fredkin gate are $p = a$, $q = \overline{a}b \oplus ac$, and $r = ab \oplus \overline{a}c$. Similarly, the output functions of the second Fredkin gate are $x = p = a$, $y = \overline{p}q \oplus pr = \overline{a}(\overline{a}b \oplus ac) + a(ab \oplus \overline{a}c) = \overline{a}b \oplus ab = b(a \oplus \overline{a}) = b$, and $z = pq \oplus \overline{p}r = a(\overline{a}b \oplus ac) \oplus \overline{a}(ab \oplus \overline{a}c) = ac \oplus \overline{a}c = c(a \oplus \overline{a}) = c$. That is, $x = a$, $y = b$, and $z = c$. Thus, the circuit in Figure 3.9(b) transfers the inputs to the outputs unchanged. The other two variants of Template 1 can be proven in similar ways.

Template 2

The next template can be applied when a cascade of a SWAP gate and a 3-bit positive control Fredkin gate appear in such a way that the targets of the SWAP and the Fredkin gates are on the same lines of a circuit. This sequence of two gates can be replaced by a 3-bit negative control Fredkin gate. The control and the targets of the negative control Fredkin gate appear on the corresponding lines where the control and the targets of the positive control Fredkin gate appear. That is, for two adjacent gates $G_1(T_1)$ and $G_2(C;T_2)$ if $T_1 = T_2$, the two gates can be replaced by $G(\overline{C};T_1)$. In addition, a sequence consisting of a SWAP and a negative control Fredkin gate can be substituted by a positive control Fredkin gate. Equation 3.2 shows the expressions and Figure 3.10 illustrates both versions of this template.

$$SWAP(t_1,t_2)FRED(c;t_1,t_2) \equiv FRED(\overline{c};t_1,t_2) \tag{3.2a}$$

$$SWAP(t_1,t_2)FRED(\overline{c};t_1,t_2) \equiv FRED(c;t_1,t_2) \tag{3.2b}$$

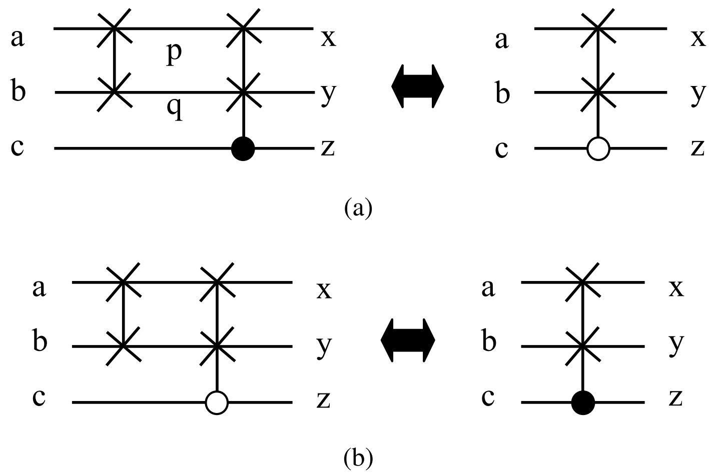


Figure 3.10: Template 2.

Suppose p and q are the two outputs of the SWAP gate in Figure 3.10(a). Here $p = b$ and $q = a$. The output of the Fredkin gate will be $x = \overline{c}p \oplus cq = \overline{c}b \oplus ca$, $y = cp \oplus \overline{c}q = cb \oplus \overline{c}a$, and $z = c$. The outputs of the negative control Fredkin gate in this figure are $x = \overline{c}b \oplus ca$, $y = cb \oplus \overline{c}a$, and $z = c$. Thus, the two circuits in this figure are equivalent to each other. Template 2 reduces both GC and QC by 1.

Template 3

Two adjacent gates $G_1(C_1;T_1)$ and $G_2(\overline{C_2};T_2)$ can be replaced by $G(T_1)$ if $C_1 = \overline{C_2}$ and $T_1 = T_2$. Template 3 can be applied when two 3-bit Fredkin gates with different polarity appear in such a way that the controls and targets of the gates are on the same lines. In such a case, the Fredkin gate pair can be replaced by a single SWAP gate. The targets of the SWAP gate appear on the same line as that of the Fredkin gates, as shown in Figure 3.11.

$$FRED(c;t_1,t_2)FRED(\bar{c};t_1,t_2) \equiv SWAP(t_1,t_2) \tag{3.3}$$

The figure shows that the negative control Fredkin gate interchanges the bits at the top two

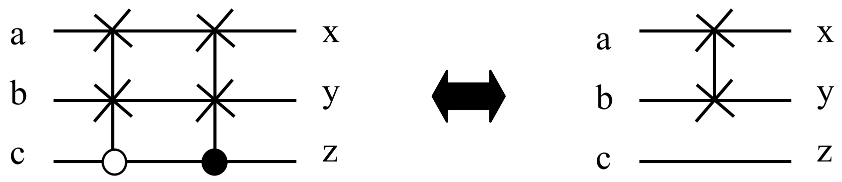


Figure 3.11: Template 3.

lines of a circuit when $c = 0$. Next, the positive control Fredkin gate swaps the bits at the same top lines when $c = 1$. Thus, regardless of the value presented at the bottom line, the values at the top two lines interchange, and this is what a SWAP gate does. Template 3 reduces QC by 70% and GC by 1.

Template 4

Template 4 can be applied when two Fredkin gates of different sizes appear in cascade in a circuit. This template is applicable when a 3-bit Fredkin gate, $FRED(C_1,T_1)$ and a 4-bit Fredkin gate, $FRED(C_2,T_2)$ appear in such a way that $C_1 \cap C_2 = C_1$, and $T_1 = T_2$. In other words, the targets of both gates are on the same line, and the control of the 3-bit Fredkin gate shares the line with any of the controls of the 4-bit Fredkin gate. If these two adjacent Fredkin gates appear in this manner, they can be replaced by a 4-bit mixed polarity Fredkin gate.

$$FRED(c_1;t_1,t_2)FRED(c_1,c_2;t_1,t_2) \equiv FRED(c_1,\overline{c_2};t_1,t_2) \tag{3.4a}$$

$$FRED(\overline{c_1};t_1,t_2)FRED(\overline{c_1},\overline{c_2};t_1,t_2) \equiv FRED(\overline{c_1},c_2;t_1,t_2) \tag{3.4b}$$

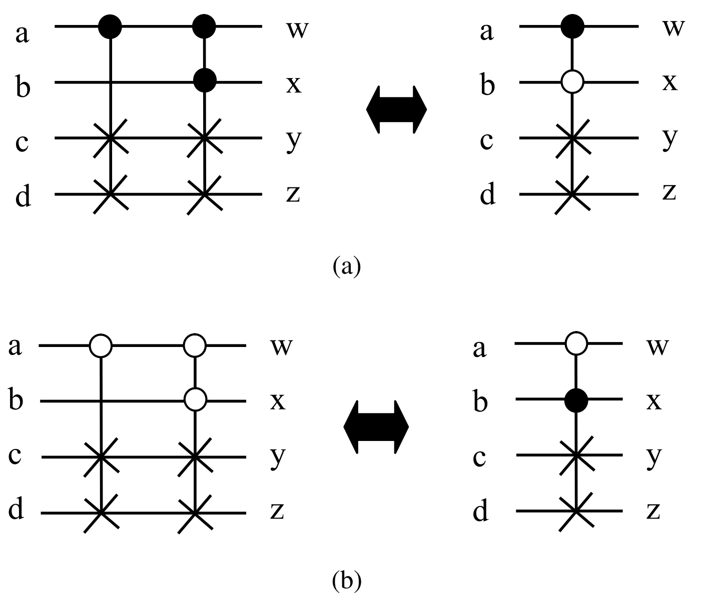


Figure 3.12: Template 4.

Figure 3.12 shows two versions of this template for both positive and negative polarity. The expressions for this two versions are given in Equation 3.4. As seen from Figure 3.12(a), the positive control of the 4-bit mixed polarity Fredkin gate is on the same line as the control of the 3-bit Fredkin gate. Template 4 reduces QC from 18 to 13. The GC is also reduced to 1.

Template 5

The next template is an example of a circuit optimization using template matching when a pattern of gates is replaced by another pattern of gates. Template 5 replaces a sequence of 3 gates to a sequence of 2 gates as shown in Figure 3.13. The QC is reduced by 3 after applying this template to a circuit. The truth table for Template 5 is presented in Table 3.8.

Both circuits in Figure 3.13 implement the function shown in Table 3.8.

$$SWAP(t_1,t_2)FRED(\overline{t_2};t_1,c)FRED(c;t_1,t_2) \equiv FRED(\overline{c};t_1,t_2)FRED(\overline{t_2};t_1,c) \tag{3.5}$$

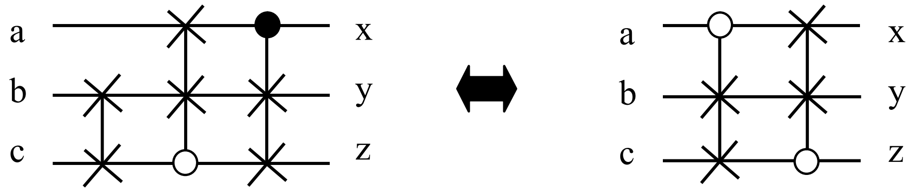


Figure 3.13: Template 5.

Table 3.8: Truth table for Template 5.

inputs			outputs		
a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

Template 6

Template 6 can be applied to simplify two n-bit Fredkin gates when $n \geq 4$. Two adjacent n-bit Fredkin gates $G_1(C_1 \cup c_i; T_1)$ and $G_2(C_2 \cup \overline{c_i}; T_2)$ can be replaced by a $n-1$-bit Fredkin gate $G_3(C_3; T_3)$, where $C_1 = C_2 = C_3$, and $T_1 = T_2 = T_3$.

$$FRED(c_1,c_2;t_1,t_2)FRED(\overline{c_1},c_2;t_1,t_2) \equiv FRED(c_2;t_1,t_2) \tag{3.6a}$$

$$FRED(\overline{c_1},\overline{c_2};t_1,t_2)FRED(c_1,\overline{c_2};t_1,t_2) \equiv FRED(\overline{c_2};t_1,t_2) \tag{3.6b}$$

$$FRED(c_1,c_2,c_3;t_1,t_2)FRED(c_1,c_2,\overline{c_3};t_1,t_2) \equiv FRED(c_1,c_2;t_1,t_2) \tag{3.6c}$$

$$FRED(\overline{c_1},\overline{c_2},\overline{c_3};t_1,t_2)FRED(\overline{c_1},c_2,\overline{c_3};t_1,t_2) \equiv FRED(\overline{c_1},\overline{c_3};t_1,t_2) \tag{3.6d}$$

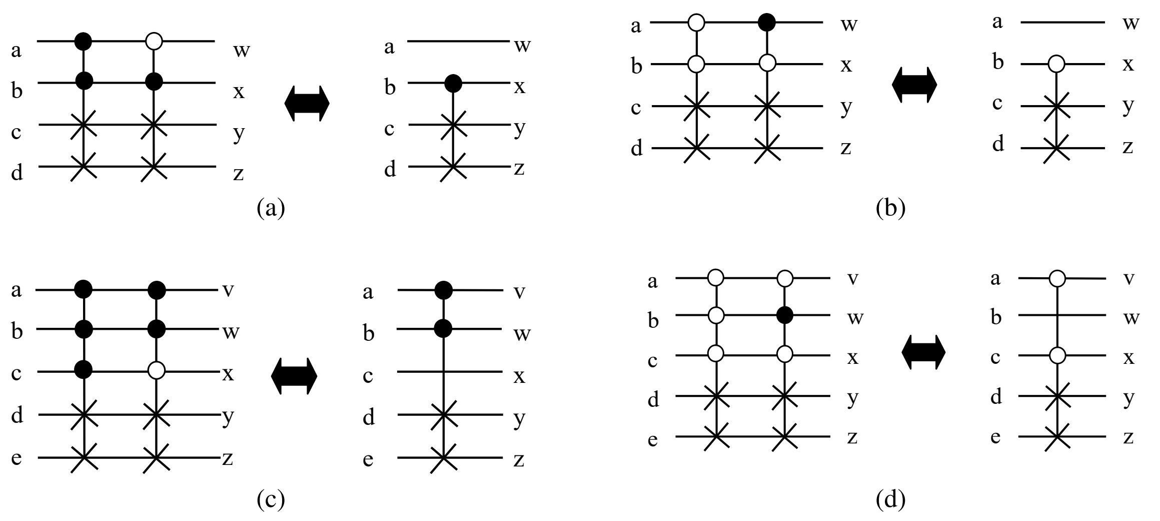


Figure 3.14: Template 6.

In other words this template can be used when the controls and targets of two n-bit Fredkin gates appear on the same lines and only one control point of a gate has a different polarity. In such a case the two n-bit gates can be replaced by a $(n-1)$-bit Fredkin gate. The target of the $(n-1)$-bit Fredkin gates appear on the same line as that of n-bit Fredkin gates and the control with opposite polarity will be removed. Figure 3.14 shows this template and its variations for 4-bit and 5-bit Fredkin gates, and Equation 3.6 shows the expressions for these four variations of Template 6. For 4-bit Fredkin gates, this template reduces GC by 1, and QC from 26 to 5.

Template 7

Template 7 can be applied when two adjacent 4-bit Fredkin gates appear in such a way that the controls and the targets of both gates are on the same line. However, the polarities of control points on same line are different. These Fredkin gates can be replaced by two positive control 3-bit Fredkin gates. The targets of both 3-bit Fredkin gates are on the same lines as that of the 4-bit Fredkin gates and controls will be on different lines as shown in Figure 3.15. The expression of this template is shown in Equation 3.7. The GC remains the same after applying this template, however, the QC reduces from 26 to 10.

$$FRED(c_1,\overline{c_2};t_1,t_2)FRED(\overline{c_1},c_2;t_1,t_2) \equiv FRED(c_1;t_1,t_2)FRED(c_2;t_1,t_2) \tag{3.7}$$

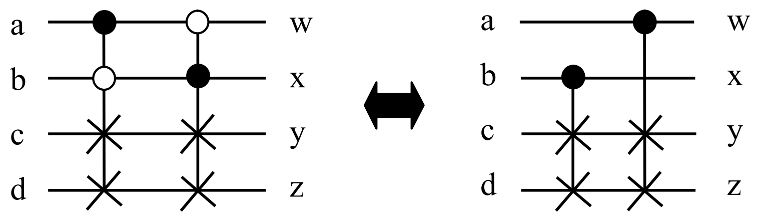


Figure 3.15: Template 7.

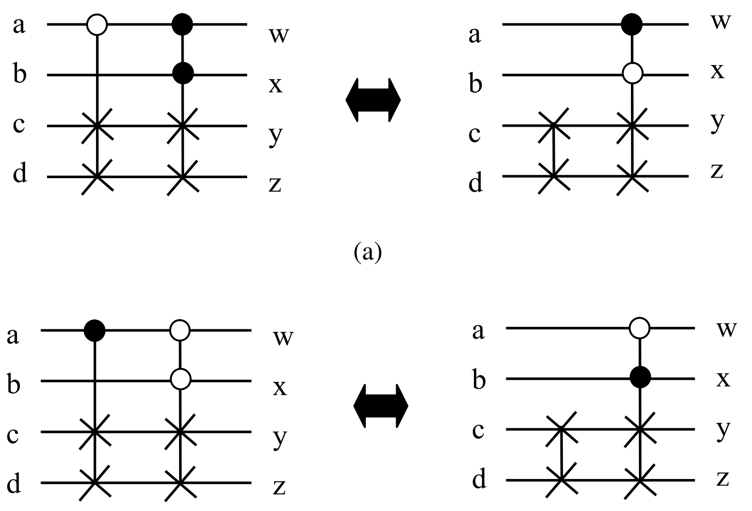
Template 8

Template 8 can be applied when a 3-bit Fredkin gate and a 4-bit Fredkin gate appear in such a way that the targets and the controls of both gates appear on the same line. However, the 4-bit Fredkin gate has the same polarity on its control points, and the control point of the 3-bit Fredkin gate is opposite in polarity than that of the 4-bit Fredkin gate. Figure 3.16(a) shows that both control points of the 4-bit Fredkin gate are positive, while the control point of the 3-bit Fredkin gate is negative. In this case these two Fredkin gates can be replaced by a cascade of a SWAP gate and a 4-bit mixed polarity Fredkin gate, as shown in Figure 3.16. The GC remains the same after applying this template, however, the QC is reduced by 2.

One variant of Template 8 is shown in Figure 3.16(b). Equation 3.8 shows the expressions of these two variations of Template 8.

$$FRED(\overline{c_1};t_1,t_2)FRED(c_1,c_2;t_1,t_2) \equiv SWAP(t_1,t_2)FRED(c_1,\overline{c_2};t_1,t_2) \tag{3.8a}$$

$$FRED(c_1;t_1,t_2)FRED(\overline{c_1},\overline{c_2};t_1,t_2) \equiv SWAP(t_1,t_2)FRED(\overline{c_1},c_2;t_1,t_2) \tag{3.8b}$$



(a)

(b)

Figure 3.16: Template 8.

Template 9

Template 9 is applicable to 5-bit Fredkin gates, as shown in Figure 3.17. When the controls and targets of a 5-bit positive control Fredkin gate and a 5-bit mixed polarity Fredkin gate appear on the same line, and two controls of the mixed polarity Fredkin gate are negative controls, then these two 5-bit Fredkin gates can be replaced by 3 positive control Fredkin gates. That is, two adjacent Fredkin gates $G_1(C \cup c_1 \cup c_2; T)$ and $G_2(C \cup \overline{c_1} \cup \overline{c_2}; T)$

can be replaced three gates $G_3(C,T)$, $G_4(C \cup c_1, T)$ and $G_5(C \cup c_2, T)$.

$$\begin{aligned} & FRED(c_1,c_2,c_3;t_1,t_2)FRED(\overline{c_1},\overline{c_2},c_3;t_1,t_2) \\ & \qquad \equiv FRED(c_3;t_1,t_2)FRED(c_1,c_3;t_1,t_2)FRED(c_2,c_3;t_1,t_2) \quad (3.9a) \\ & FRED(\overline{c_1},\overline{c_2},\overline{c_3};t_1,t_2)FRED(c_1,c_2,\overline{c_3};t_1,t_2) \\ & \qquad \equiv FRED(\overline{c_3};t_1,t_2)FRED(\overline{c_1},\overline{c_3};t_1,t_2)FRED(\overline{c_2},\overline{c_3};t_1,t_2) \quad (3.9b) \end{aligned}$$

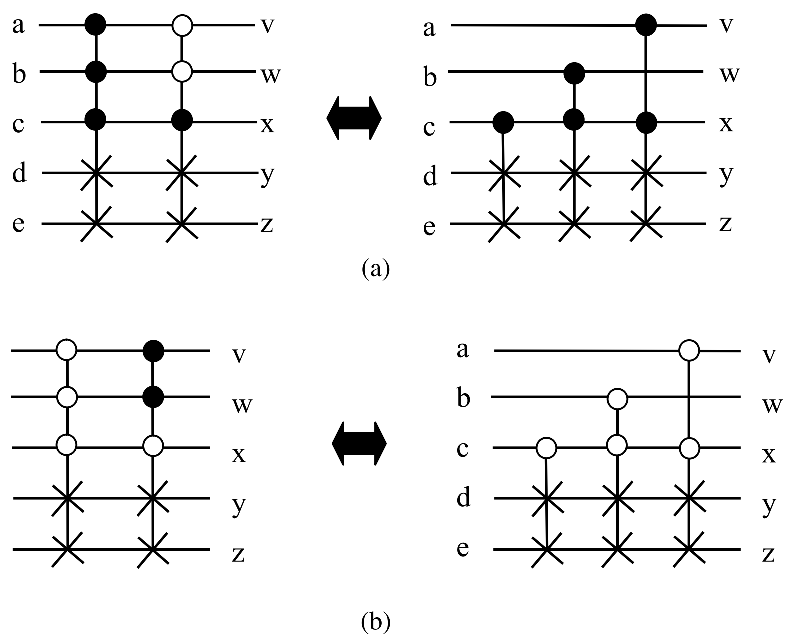


Figure 3.17: Template 9.

One of the advantages of this template is that this template replaces the mixed polarity Fredkin gates by the uniform polarity Fredkin gates. A variant of this template is shown in Figure 3.17(b). Unlike other templates, Template 9 increases GC by 1. However, QC is reduced from 58 to 31. Equation 3.9 shows the expressions of these two variations.

Template 10

Template 10 works for two 5-bit Fredkin gates. This template can be applied when the controls and the targets of two 5-bit Fredkin gates appear on the same line of a circuit. However, one control of each gate has negative polarity, and the control points with negative polarity are not on the same line. These two 5-bit Fredkin gates can be replaced by two 4-bit Fredkin gates, as shown in Figure 3.18. Equation 3.10 shows the expression of Template 10. The GC for this template remains the same, however QC is reduced from 58 to 26.

$$FRED(c_1,\overline{c_2},c_3;t_1,t_2)FRED(\overline{c_1},c_2,c_3;t_1,t_2) \equiv FRED(c_2,c_3;t_1,t_2)FRED(c_1,c_3;t_1,t_2) \tag{3.10}$$

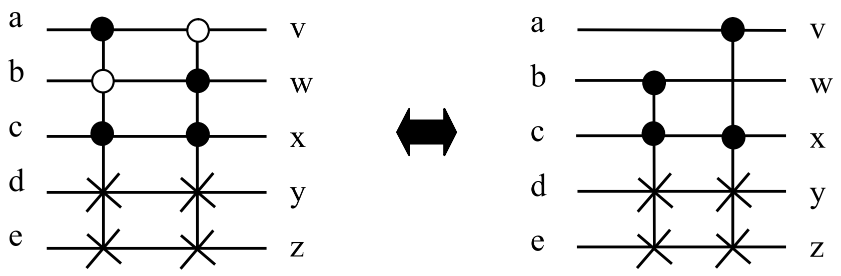
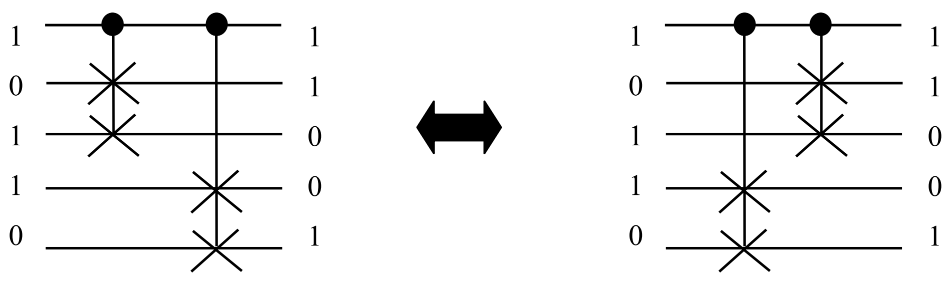


Figure 3.18: Template 10.

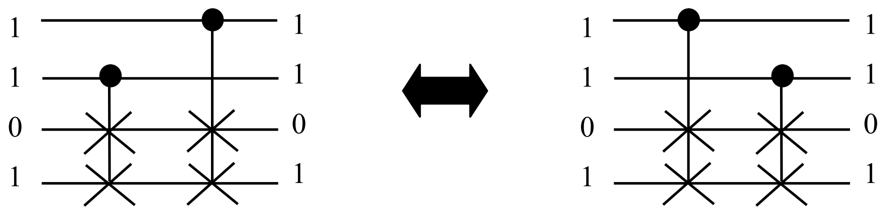
3.2.2 Reversible Circuits Optimization

The templates introduced in section 3.2.1 can be used to simplify reversible circuits. In section 3.2 we illustrated the significance of the moving rule in circuit simplification. We also demonstrated that the moving rule described in [7, 24, 37] works only on the NCT gate families. This section introduces a proposed modified moving rule which can be used for optimizing SF gate based circuits.

Two adjacent SF gates $G_1(C_1,T_1)$ and $G_2(C_2,T_2)$ can be interchanged if $C_1 \cap T_2 = \theta$ and $C_2 \cap T_1 = \theta$, and either $T_1 \cap T_2 = \theta$ or $T_1 = T_2$. In other words, two adjacent gates from the SF gate family can be interchanged if two conditions hold: (i) no control of one gate is a target of another gate, and (ii) targets of both gates are on the same line, or targets of both gates are on different lines. For example, Figure 3.19(a) shows that targets of the two gates are on two different lines. When these two gates are interchanged, the functionality of the circuit does not change. It is seen from the figure that both circuits generate the same output



(a) Targets on different lines.

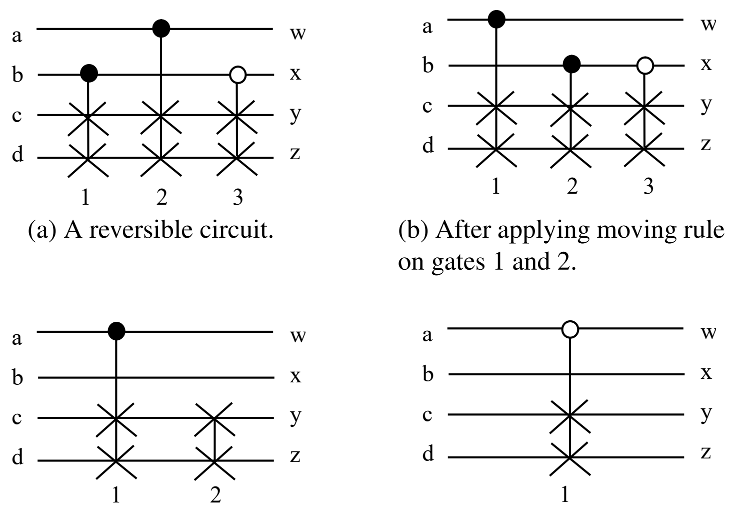


(b) Targets on the same lines.

Figure 3.19: Moving rules for the SF gate families.

for the same input. The moving rule can also be applied if targets of both gates are on the same lines, and the control points and the targets do not share any common line, as shown in Figure 3.19(b).

We have used an algorithm that incorporates the moving rule in order to apply the proposed templates. This algorithm is based on the algorithm presented in [58]. The moving rule increases the chances to match more templates, which can optimize a circuit even further. For example, consider the 4×4 reversible circuit shown in Figure 3.20(a). The GC and QC of this circuit are 3 and 15 respectively. The gates of this circuit do not match any of the proposed templates. However, it can be observed that for the gates labeled 1 and 2, no control point of any gate is on the target lines of the other gate. In addition, the targets of both gates are on the same line. So according to the moving rule, it is possible to interchange the position of these two gates. After applying the moving rule, the circuit becomes as shown in Figure 3.20(b). Now gates 2 and 3 match Template 3 from Figure 3.11. Gates 2 and 3 can be replaced by a SWAP gate, as shown in Figure 3.20(c). The two gates in Figure 3.20(c) match Template 2, which is presented in Figure 3.10(a). The resulting circuit after applying Template 2 is presented in Figure 3.20(d) with a GC of 1 and QC of 5.



(a) A reversible circuit.

(b) After applying moving rule on gates 1 and 2.

(c) After applying Template 3.

(d) After applying Template 2.

Figure 3.20: An example to show the role of the moving rule in circuit optimization.

The template matching algorithm maintains two lists of gates: an input list and an output list. The input list includes all the gates which appear in the original circuit. The output list stores the gates after the process of simplification is finished. The algorithm reads the input list and the list of all templates, and applies templates when a match is found. When a sequence of gates is replaced by a template, the new sequence of gates is stored in the output list. The algorithm processes the next sequence of gates from the input list. At each step, the algorithm decides whether a sequence of gates is to be replaced by a template or not. If no match is found for a sequence of gates, the algorithm applies the moving rule to increase the possibility of finding a match. A sequence of gates that does not match any template is also stored in the output list. Algorithm 1 shows the major steps involved in the template matching algorithm. The algorithm executes the *CHECK TEMPLATE* (*input gate list*) procedure in order to find a match. The algorithm terminates its execution when no more templates can be applied to the input gate list.

We have tested the algorithm on the benchmark circuits available on RevLib [71]. In the benchmarks there are only six circuits based on the SF gate family. The result of this experiment is presented in Table 3.9. The column labled 'Lines' in this table indicates the

Algorithm 1 Template matching algorithm.

```
1: Input   : input gate list
2: Output : output gate list
3: procedure CHECK TEMPLATE(input gate list)
4:     Count the number of gates in the input gate list
5:     Repeat while the number of gates in the input gate list $\geqslant 2$
6:     if a match is found then
7:         Apply Template (input gate list)
8:     else
9:         Apply Moving Rule (input gate list)
10:    end if
11: end procedure
12:
13: procedure APPLY TEMPLATE(input gate list)
14:    if two adjacent gates $g_i$ and $g_{i+1}$ match a template then
15:        Append the template gates to the output gate list
16:        Remove $g_i$ and $g_{i+1}$ from the input gate list
17:    end if
18:    if three adjacent gates $g_i$, $g_{i+1}$ and $g_{i+2}$ match Template 5 then
19:        Append the template gates to the output gate list
20:        Remove $g_i$, $g_{i+1}$ and $g_{i+2}$ from the input gate list
21:    end if
22:    Return
23: end procedure
24:
25: procedure APPLY MOVING RULE(input gate list)
26:    if two gates $g_i$ and $g_{i+1}$ can be interchanged then
27:        Append $g_{i+1}$ to the output gate list
28:        Remove $g_{i+1}$ from the input gate list
29:    else
30:        Append $g_i$ to the output gate list
31:        Remove $g_i$ from the input gate list
32:    end if
33:    Return
34: end procedure
```

number of input bits. The GC and QC under the ‘original circuit’ column indicates the GC and QC of the circuits before applying the templates. The GC and QC under the ‘optimized circuit’ column represent GC and QC of the circuits after applying the templates. The template matching reduces GC and QC for two of the six benchmark circuits. The best results are achieved for *hwb*4 circuit, which sees 18% GC and 9% QC reduction. The number of circuits considered for this experiment is not large enough, since the number of benchmark circuits based on the SF gate family are very few. In addition, one circuit consists of only one gate which cannot be further optimized.

Table 3.9: Results after applying the proposed templates on benchmark circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	GC	QC	GC	QC	GC	QC
fredkin	3	1	5	1	5	0	0
*hwb*4	4	11	65	9	59	18.18	9.23
*hwb*5	5	24	214	24	214	0	0
*decode*24	6	3	15	3	15	0	0
*hwb*6	6	65	1115	64	1112	1.54	0.27
*hwb*7	7	116	3998	166	3998	0	0

In order to evaluate the efficiencies of the proposed templates from a broader perspective, we randomly generated 500 SF based circuits. The number of lines of these circuits varied from 3 to 7, similar to the benchmark circuits in RevLib. Based on the number of gates, these circuits are of three different sizes: 10, 50 and 100. Our proposed approach for circuit optimization has been applied to these randomly generated circuits, and a portion of the result is presented in Table 3.10. The entire result in presented in Appendix A. The highest percentage of reduction of both GC and QC is 91%. The percentage of reduction of GC on average is 17%. The average reduction of QC is 16%.

3.3 Chapter Summary

3.3.1 Contribution

A transformation based synthesis approach [47] for realizing conservative reversible functions is presented in this chapter. This chapter also introduces 10 templates for optimizing SF gate based reversible circuits. A modified moving rule for circuit optimization is also proposed in order to increase the possibility of matching templates.

3.3.2 Conclusion

Transformation based synthesis offers function realization without including any additional garbage lines to circuits. In this chapter we have presented a transformation based synthesis approach based on SF gates to realize conservative reversible functions. We have generated all possible 3-bit and 4-bit reversible functions and realized these functions with both our proposed approach and the approach proposed in [41]. The approach presented in [41] is based on the NCT gate families. Our experimental results suggest that realization of conservative functions with SF gates is more efficient than NCT gates in terms of GC and QC. We have also compared the circuits generated using exact synthesis with SF based synthesis for implementing 3-bit conservative functions. Experimental results show that SF based synthesis generates significantly more efficient circuits than exact synthesis when comparing gate count, although slightly less so when comparing quantum cost. This is likely due to the high quantum costs of the SF gate family.

Our proposed SF based synthesis follows the principle of the NCT transformation based synthesis presented in [41]. A NCT transformation based synthesis approach works by mapping a reversible function into an identity function. During the process of transformation the operations performed at each stage must not affect the previous stages. One or more logic gates are applied to perform the logical operations at each stage. We have shown in section 3.1.3 that the choice of gates at each stage is very important in order to achieve a simplified circuit.

Reversible circuits generated with the transformation based synthesis may not be optimal. Template matching and rule based optimization techniques are two common approaches to simplify reversible circuits generated by transformation based synthesis. In this chapter we have presented 10 templates based on template matching and rule based optimization. We have tested the proposed templates to simplify reversible circuits consisting of only SF gates. Since few SF gate based circuits are available as benchmark circuits in RevLib, we have randomly generated 500 SF based reversible circuits. The results of experiments suggest that our proposed templates can contribute to optimizing SF gate based reversible circuits. The highest reduction in QC is 9% after applying the proposed templates on benchmark circuits. In case of randomly generated circuits we have achieved 16% reduction in QC on average.

3.3.3 Future Directions

Our experiments on synthesizing conservative functions reveal that there is a need for the classification of reversible functions. If the class of a reversible function is known in advance before synthesis takes place, it may be possible to generate more efficient circuits. In addition, improving the gate selection process during each stage of the transformation based synthesis is an important area of further research. The performance of the transformation based synthesis can be improved further if this approach consider the next stages before selecting a gate for the current stage. Instead of selecting a gate with a greedy approach, an intelligent gate selection technique may consider both previous and next stages prior to selecting a gate for the current stage. So a gate selected for the current stage will make it possible to transform the next stages with lower QC, and thus improve the overall performance.

We have compared the cost of the circuits generated using our proposed SF based synthesis with the minimal circuits generated using exact synthesis. We gates used in exact synthesis are NCT gates. Generating minimal circuits using SF based exact synthesis is an open area of further research.

Lastly, identifying more templates based on SF gates or on a combination of NCT and SF gates is also an area for future research. Our experimental results show that the QC is reduced up to 9% after applying the proposed templates on benchmark circuits. The QC can be reduced further using an efficient template matching algorithm. The algorithm which we have used to apply the templates uses an exhaustive search approach to match templates. In [58], Rahman et al. proposed a template matching algorithm that assigns ranks to the templates based on the amount of QC reduction offered by the templates. Thus their proposed algorithm applies templates that offer the best possible reduction in QC at a particular instant. This indicates that developing an efficient SF gate based template matching algorithm can also be an area of future study.

Table 3.10: Results after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	GC	QC	GC	QC	GC	QC
random173	5	100	695	94	673	6	3.17
random297	4	10	42	8	36	20	14.29
random172	6	55	878	55	878	0	0
random298	6	55	715	54	710	1.82	0.7
random171	7	10	203	10	203	0	0
random299	6	100	1385	97	1376	3	0.65
random178	7	55	1093	55	1093	0	0
random292	4	55	209	44	168	20	19.62
random177	4	10	42	9	39	10	7.14
random293	5	100	773	95	754	5	2.46
random176	3	100	300	28	84	72	72
random2	6	100	1444	98	1438	2	0.42
random294	5	10	111	10	111	0	0
random1	7	55	1268	55	1268	0	0
random175	3	55	165	25	75	54.55	54.55
random295	3	55	165	27	81	50.91	50.91
random290	7	100	2162	100	2162	0	0
random291	7	10	134	8	128	20	4.48
random170	5	100	757	97	746	3	1.45
random4	7	55	1097	51	1085	7.27	1.09
random3	6	10	96	10	96	0	0
random6	3	10	30	4	12	60	60
random179	7	100	1878	100	1878	0	0
random5	5	100	781	96	763	4	2.3
random8	5	100	624	95	609	5	2.4
random7	7	55	1164	55	1164	0	0
random9	7	10	246	10	246	0	0
random163	3	55	165	31	93	43.64	43.64
random285	7	10	192	10	192	0	0
random162	7	10	238	10	238	0	0
random286	5	55	375	50	360	9.09	4
random161	6	100	1379	98	1371	2	0.58
random287	7	100	2079	100	2079	0	0
random160	7	55	1428	55	1428	0	0
random288	5	10	49	9	44	10	10.2
random167	7	100	1964	100	1964	0	0
random281	6	100	1467	100	1467	0	0
random166	3	55	165	15	45	72.73	72.73
random282	5	10	43	10	43	0	0
random165	5	10	58	10	58	0	0
random283	6	55	887	54	882	1.82	0.56
random164	4	100	410	77	329	23	19.76
random284	4	100	412	85	355	15	13.83
random280	4	55	211	43	163	21.82	22.75
random169	6	55	690	54	687	1.82	0.43
random168	6	10	185	10	185	0	0
random289	3	55	165	15	45	72.73	72.73
random196	7	55	1170	55	1170	0	0
random195	3	10	30	6	18	40	40
random194	3	100	300	36	108	64	64
random193	5	55	392	53	384	3.64	2.04
random199	4	55	215	48	190	12.73	11.63
random198	7	10	126	10	126	0	0
random428	7	100	1947	99	1944	1	0.15
random300	7	10	122	10	122	0	0
random421	7	55	1110	55	1110	0	0
random301	6	55	738	55	738	0	0
random422	3	100	300	16	48	84	84
random302	3	100	300	30	90	70	70
random423	4	10	40	10	40	0	0
random303	3	10	30	2	6	80	80
random424	7	55	1253	52	1238	5.45	1.2
random500	6	100	1234	100	1234	0	0

Chapter 4

Fault Testing

Testing is a very important phase in the process of developing a system. Testing refers to a process to determine whether a system is faulty or fault-free. Testing is necessary to ensure the reliability and quality of a digital system [25]. Many testing approaches used for irreversible circuits cannot be applied to their reversible counterparts due to the nature of reversible gates and the underlying differences in computation strategy. Research in this area has focused on fault models and developing fault detection approaches. This chapter introduces reversible fault models and presents testing approaches for reversible circuits.

4.1 Fault Models and Fault Testing: An Overview

A fault is a failure of a system. In other words, a fault can be defined as a physical defect that leads a system to produce an incorrect output. Faults must be detected and removed. A fault model is a model that describes the types of faults which may occur in a system [59]. An ideal fault testing method detects all the faults of a particular fault model. The most common fault model for traditional logic circuit is the stuck-at fault model [56, 72, 73]. A stuck-at fault may occur due to the malfunction of some part of a circuit, such as a path or line of that circuit being able to pass only high or low voltage signals. That is, a path of a circuit may become stuck at a particular voltage level. When a line of a circuit passes only high voltage due to some undesirable event, the fault is considered to be a stuck-at 1 fault. This could occur in the case of a short in a circuit, for instance. On the other hand, if a line passes only low voltage, the fault is considered to be a stuck-at 0 fault. For instance, this could occur if a line or path of a circuit is broken such that voltage cannot pass through it. As a result, the logic circuit may give an incorrect result. However, research suggests that the stuck-at fault model is not suitable for reversible computing. For this reason, other fault models for reversible logic have been developed [20, 56, 73].

Testing determines whether a system is faulty or fault-free. Testing can be divided into two categories: offline testing and online testing [59]. Each of these strategies has its own benefits and drawbacks. Online testing methods determine whether the output of a system is correct or not, while the system performs its normal operations. In offline testing, a system under consideration is removed from its normal mode of operation before the method of testing is applied.

4.2 Fault Models for Reversible Logic

The underlying concepts of reversible gates are different from their irreversible counterparts. Therefore, the pattern of occurrence of faults in reversible circuits is likely to be different. Several technologies have been suggested for the physical implementation of quantum circuits, which are different than the traditional circuits. For example, the spin of electrically charged atoms can be used to represent qubits (quantum bits), and the states of these charged atoms can be changed by directing laser pulses on them [20, 56, 51]. That is, the reversible gate operations are likely to be implemented by means of pulses. Thus, faults such as stuck-at faults and bridge faults [59] which are wire-oriented are likely to be irrelevant in reversible circuits [20, 56]. Thus, the process of detecting faults will be different in reversible computing. A fault model considers all fault possibilities which may occur in a circuit. A number of fault models have been developed to cope with the nature of reversible circuits [59]. This section describes some fault models such as the crosspoint fault model [73] and the missing and repeated gate fault model [20].

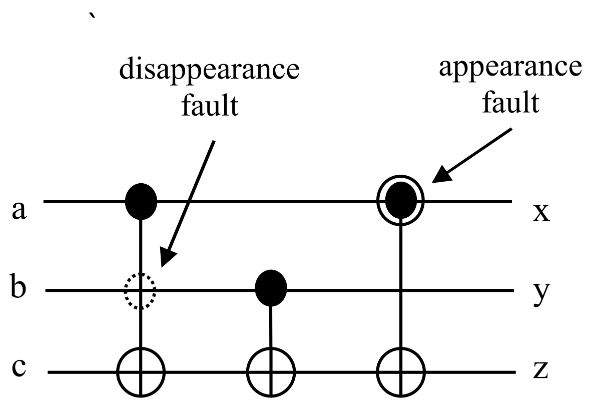


Figure 4.1: Appearance and disappearance faults in a reversible circuit.

Table 4.1: Truth table of the circuit shown in Figure 4.1.

Input	Output	
	Faulty	Fault free
000	000	000
001	001	001
010	011	010
011	010	011
100	101	100
101	100	101
110	111	110
111	110	111

4.2.1 Crosspoint Fault Model [73]

Most reversible gates have control points. The values of these control points determine whether a gate performs an operation on the target inputs. For example, a 1-CNOT gate inverts the value of target input when the control is 1. Thus, a fault in a control point can result in an incorrect output. Faults that are relevant to control points are known as crosspoint faults. Crosspoint faults are divided into two categories: appearance crosspoint faults and disappearance crosspoint faults [73]. When one or more control points are added erroneously to a gate, this is considered to be an appearance crosspoint fault. On the other hand, if a gate is missing one or more control points, this type of fault belongs to the category of disappearance crosspoint faults. Based on the number of affected control points, a crosspoint fault can be either a single crosspoint fault or a multiple crosspoint fault. When only one control point appears or disappears, the fault is considered a single crosspoint fault. When more than one control point is affected, the fault is considered a multiple crosspoint fault.

Figure 4.1 shows an example of appearance and disappearance crosspoint faults in a reversible circuit. When the faults are present, the circuit generates incorrect outputs. Table 4.1 shows the behaviour of the circuit with the presence of two crosspoint faults. Investigating the effect of faults on the circuit output is very important in order to develop fault detection and correction mechanisms. There are certain input vectors for which a fault does not affect the final output of a circuit. Therefore, in order to test a circuit for a particular fault, we need to consider those input vectors for which the fault affects the output of the circuit. For example, Table 4.1 shows that when the input of the circuit in Figure 4.1 is $abc = 000$, the circuit generates the corrected output even in the presence of two crosspoint faults. The faults do not affect the output of the circuit for this particular input. Thus, $abc = 000$ is not a desirable input for testing the circuit presented in Figure 4.1 for crosspoint faults. However, when input $abc = 100$, the circuit generates 101, which is incorrect. Therefore this input vector can be used to test the circuit for crosspoint faults.

4.2.2 Missing Gate and Repeated Gate Fault Model [56]

The missing and repeated gate fault model is a package of several fault models designed for reversible circuits. The package consists of four different categories of faults: the single missing gate fault model (SMGF), the multiple missing gate fault model (MMGF), the partial missing gate fault model (PMGF) and the repeated gate fault model (RGF). As mentioned earlier, gate operations in reversible circuits are likely to be by means of pulses [20]. SMGFs and MMGFs may occur in a reversible circuit for short, missing or mistuned gate pulses. RGFs may occur due to long or duplicated gate pulses and PMGFs may occur due to partially mistuned gate pulses [20, 56].

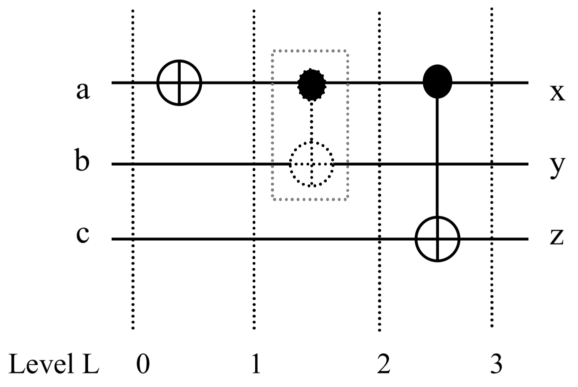


Figure 4.2: SMGF in a reversible circuit.

Single Missing Gate Fault Model [56]

The single missing gate fault (SMGF) model was developed to model faults which occur when an entire gate becomes inactive. That is, a SMGF occurs when an entire gate in a reversible circuit does not carry out its intended operation. As a consequence, a circuit may generate an incorrect output due to the complete disappearance of a gate.

Figure 4.2 shows an example of an occurrence of a SMGF in a reversible circuit. In this circuit, the highlighted CNOT gate is missing. The values at the different levels of this circuit are presented in Table 4.2. The correct value and incorrect values are represented by correct value/incorrect value notation in the table. An observation from Table 4.2 is that the effect of a fault which occurred on a single line is propagated to multiple lines in a circuit. As a result, more than one output bit may be incorrect as the result of an occurrence of the fault.

Repeated Gate Fault Model [56]

The repeated gate fault (RGF) model addresses faults that may occur due to an unwanted replacement of a gate by multiple instances of the same gate. One or more unwanted instances of a gate in a circuit can generate incorrect output. These unwanted instances of a gate are considered to be RGFs.

The RGF has two different effects based on whether a gate is replaced by an even or odd number of the same gate. According to [56], the number of erroneous instances of a

Table 4.2: Output values at the different levels of the circuit shown in Figure 4.2.

Gate Levels			
0	**1**	**2**	**3**
0	1	1	1
0	0	0/1	0/1
0	0	0	0/1
0	1	1	1
0	0	0/1	0/1
1	1	1	1/0
0	1	1	1
1	1	1/0	1/0
0	0	0	1/0
0	1	1	1
1	1	1/0	1/0
1	1	1	0/1
1	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0
0	0	0	0
1	1	1	1
1	0	0	0
1	1	1	1
0	0	0	0
1	0	0	0
1	1	1	1
1	1	1	1

correct/incorrect represents the correct and the incorrect values.

same gate may have the following two effects:

Case 1: If the number of instances of a gate is even, then the faulty effect of a RGF is identical to the effect of a SMGF.

Case 2: If the number of instances is odd, the fault does not affect the circuit output.

Figure 4.3 shows an example of an occurrence of a RGF in a reversible circuit, where the number of gate instances is even. A 1-CNOT gate is replaced by two instances of the

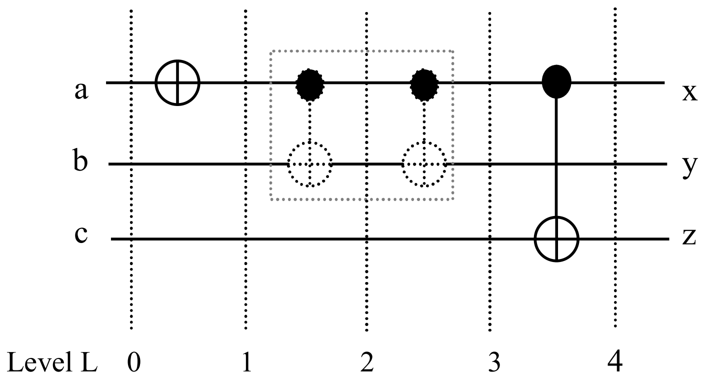


Figure 4.3: A RGF where a gate is replaced by an even number of instances of the same gate.

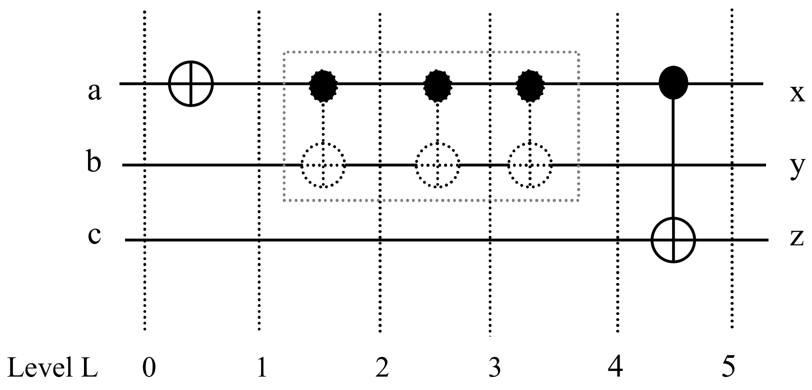


Figure 4.4: A RGF where a gate is replaced by an odd number of instances of the same gate.

same gate. The values at each gate level of the circuit are presented in Table 4.3.

It can be observed that the faulty effect of the SMGF presented in Table 4.2 (level 2) is the same as that of a RGF in Table 4.3 (level 3). This demonstrates the property presented in case 1. That is, with respect to the same gate, the effects of SMGF and a RGF are identical when a gate is replaced by an even number of instances of the same gate.

Figure 4.4 shows an example of a RGF for an odd number of instances of a gate (case 2). Here, a 1-CNOT gate is replaced by three of the same gate. The values of each gate level are presented in Table 4.4. The immediate effect of the RGF can be observed at level 4 of this circuit. The values presented in Table 4.4 are identical to the values when there is no fault in the circuit. That is, as case 2 states, when a reversible gate is replaced by an

Table 4.3: Output values at the different levels of a circuit shown in Figure 4.3.

Gate Levels				
0	**1**	**2**	**3**	**4**
0	1	1	1	1
0	0	1	0	0
0	0	0	0	0
0	1	1	1	1
0	0	1	0	0
1	1	1	1	1
0	1	1	1	1
1	1	0	1	1
0	0	0	0	1
0	1	1	1	1
1	1	0	1	1
1	1	1	1	0
1	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	0	0	0
0	0	0	0	0
1	1	1	1	1
1	0	0	0	0
1	1	1	1	1
0	0	0	0	0
1	0	0	0	0
1	1	1	1	1
1	1	1	1	1

odd number of the same gate, this unwanted repetition does not affect the final output of the circuit.

Multiple Missing Gate Fault Model [56]

The concept of the multiple missing gate fault (MMGF) model is similar to the SMGF model. However, the difference is that in a MMGF, more than one gate disappears from a reversible circuit instead of the disappearance of a single gate. The rectangle shown in Figure 4.5 indicates that the enclosed CNOT and Toffoli gates are missing from the circuit.

Table 4.4: Output values at the different levels of the circuit in Figure 4.4.

Gate Levels					
0	**1**	**2**	**3**	**4**	**5**
0	1	1	1	1	1
0	0	1	0	1	1
0	0	0	0	0	1
0	1	1	1	1	1
0	0	1	0	1	1
1	1	1	1	1	0
0	1	1	1	1	1
1	1	0	1	0	0
0	0	0	0	0	0
0	1	1	1	1	1
1	1	0	1	0	0
1	1	1	1	1	1
1	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
1	1	1	1	1	1
1	0	0	0	0	0
1	1	1	1	1	1
0	0	0	0	0	0
1	0	0	0	0	0
1	1	1	1	1	1
1	1	1	1	1	1

Partial Missing Gate Fault Model [56]

The partial missing gate fault (PMGF) model addresses a fault that occurs when instead of losing an entire gate, some parts of the gate become inactive. More specifically, when one or more control points of a gate become inactive, the fault can be considered to be a PMGF. The concept of a PMGF is similar to the concept of the crosspoint faults. A PMGF turns a n-CNOT gate into a k-CNOT gate, where $k < n$. The quantity $(n-k)$ is known as the order of a PMGF.

For example, a 2-CNOT gate has become a 1-CNOT gate due to the lose of one control

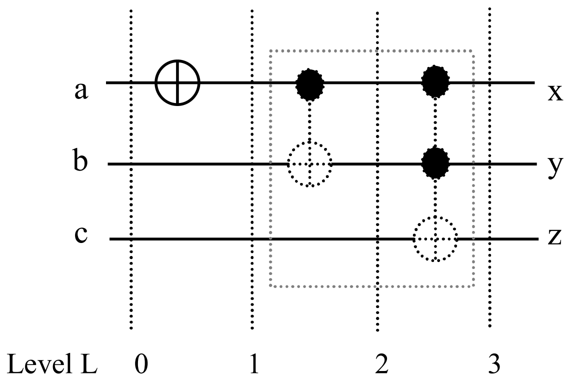


Figure 4.5: An example of the occurrence of a MMGF.

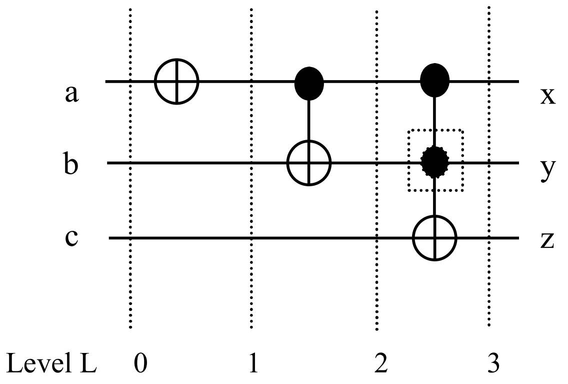


Figure 4.6: A reversible circuit with the occurrence of a PMGF.

point, as indicated by the rectangle in Figure 4.6.

4.3 Fault Testing in Reversible Circuits

Fault testing is a mechanism to determine whether a system is faulty or fault free. This section describes some existing offline and online testing approaches for reversible circuits.

4.3.1 Offline Approaches

With offline testing approaches, a method of testing is applied while the system is not performing its normal operations. Generally offline testing takes place when the load of the system is at a minimum. The disadvantage of this category of approaches is that an offline approach reduces system throughput, since a system can not perform its normal operations

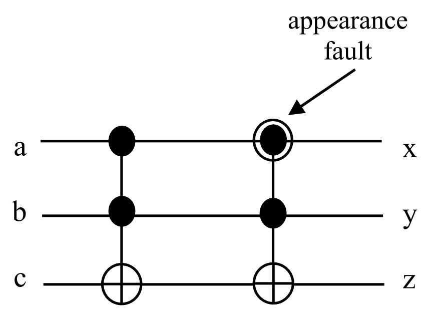


Figure 4.7: An example of a single appearance crosspoint fault.

while the system is in testing mode. The main advantage of offline testing approaches is that offline approaches reduce the design complexity of a system under consideration. This means that in most cases, no extra circuitry is required for offline testing [59]. The main concern of offline testing is to observe the known outputs of a system for specific inputs. An input vector, which is applied for testing purposes is called a test vector, and a set of test vectors is known as a test set. A complete test set can detect all faults for a particular fault model. One of the main challenges of offline testing approaches is to develop a minimum complete test set. Some offline approaches use additional circuitry for testing. Offline approaches based on this technique are known as design-for-test (DFT). The following subsections discuss two categories of offline testing approaches for two reversible fault models: crosspoint fault testing, and missing and repeated gate fault testing.

Crosspoint Fault Testing

Zhong et al. prosose a crosspoint fault testing approach [73]. They have found that a complete test set based on the crosspoint fault model can also detect all single appearance and disappearance crosspoint faults. The basic concept of testing for appearance crosspoint fault is to select an input test vector, which assigns 1 to an input line that contains control points and sets other input lines to 0. Suppose an appearance crosspoint fault occurs in the 1-CNOT gate, as indicated in Figure 4.7. Thus, the 1-CNOT becomes a 3-bit Toffoli gate. In this case an input vector $(a,b,c) = (0,1,0)$ can detect a difference between the correct output and an incorrect output. The output of this circuit for fault free operation is $(x,y,z) = (0,1,1)$. However, in the presence of this appearance fault, the output is $(x,y,z) = (0,1,0)$. The values of the test vector at different stages of the circuit is shown in Table 4.5. In this table the column $x_i\ y_i\ z_i$ indicates the intermediate value which is obtained after the 3-bit Toffoli gate is applied to the test vector.

Table 4.5: Test vector to detect appearance fault in the circuit shown in Figure 4.7.

(a) Incorrect output reflecting the presence of the fault.

Test Vector		$x_i\ y_i\ z_i$	xyz
a	0	0	0
b	1	1	1
c	0	0	0

(b) Correct output.

Test Vector		$x_i\ y_i\ z_i$	xyz
a	0	0	0
b	1	1	1
c	0	0	1

For a n-bit gate, a test vector for an appearance fault must set lines with control inputs to 1, while other lines to 0. Thus, for each gate of a circuit, one input vector is required in order to test for any single appearance crosspoint fault. Therefore, the size of a complete test set to detect all single appearance crosspoint faults in a circuit is, at most, the number of gates. A compete test set that consists of a minimal number of test vectors is considered to be the most efficient test set.

Missing and Repeated Gate Fault Testing

This section describes one approach to test reversible circuits for SMGF. Offline testing approaches for other fault models can be found in [56]. Since a SMGF removes an entire gate from a circuit, the number of possible SMGFs is equal to the number of gates in a circuit. The fault detection condition for a SMGF is to set all control inputs of the gate under consideration to 1 [56]. This fault detection condition assumes that all of the control points are positive.

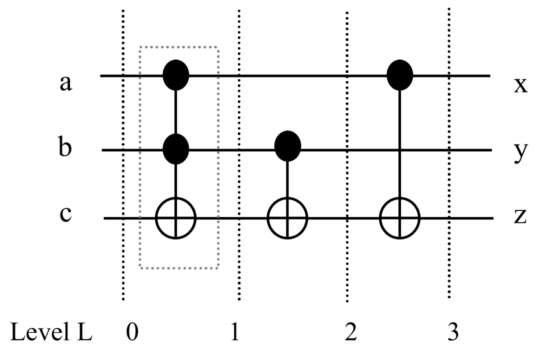


Figure 4.8: An example of SMGF in a reversible circuit.

For example, consider the reversible circuit shown in Figure 4.8. A test set to detect all SMGFs for this circuit can be $(1,1,x),(x,1,x),(1,x,x)$. The value x represents a don't care value; that is, the value of x in a test vector can be either 0 or 1. This allows some test vector to overlap, and thus a minimal complete test set can be $(1,1,x)$; for example $(1,1,0)$. Suppose the 3-bit Toffoli gate shown in Figure 4.8 is affected by a SMGF. Using the test vector $(1,1,0)$, Table 4.6 shows how the fault affects the test vector at different levels of the circuit. The correct and incorrect values are presented by correct/incorrect notation in Table 4.6. This example demonstrates how the test vector identifies a difference between a correct output and an incorrect output.

Table 4.6: Test vector to identify a SMGF for the circuit, as shown in Figure 4.8.

Gate Levels			
0	**1**	**2**	**3**
1	1	1	1
1	1	1	1
0	1/0	0/1	1/0

The correct and incorrect values are presented by correct/incorrect notation.

The above discussion suggests that generating a complete test vector can be a major task in designing offline testing approaches. In addition, the size of a complete test set should be minimal [54, 73]. In order to avoid difficulties in generating multiple test vectors, Hayes et al. [20] proposed a DFT offline testing approach. Their proposed approach works for the NCT gate family, and can detect a missing gate fault (MGF) using only a single test vector. In DFT based approaches, additional circuitry is added to a circuit. According to their approach, a reversible circuit can be tested for a MGF by adding one line, known as a DFT line, which contains control points for one or more additional 1-CNOT gates. The basic idea behind the approach is as follows. The condition to test for MGFs for any k-CNOT gate family is that a test vector must be in the form of $(1,1,......,1,t)$ [20], where t is the target input of a CNOT gate. Only this input vector, in form of $(1,1,......,1,t)$, can convert the target bit t to $\bar{t}$. All other input vectors will not affect the outputs. The purpose of each added 1-CNOT gate is to generate the test vector for every gate in the form of $(1,1,......,1,t)$. That is, a 1-CNOT gate is added in order to ensure that a control point of an existing gate is always 1.

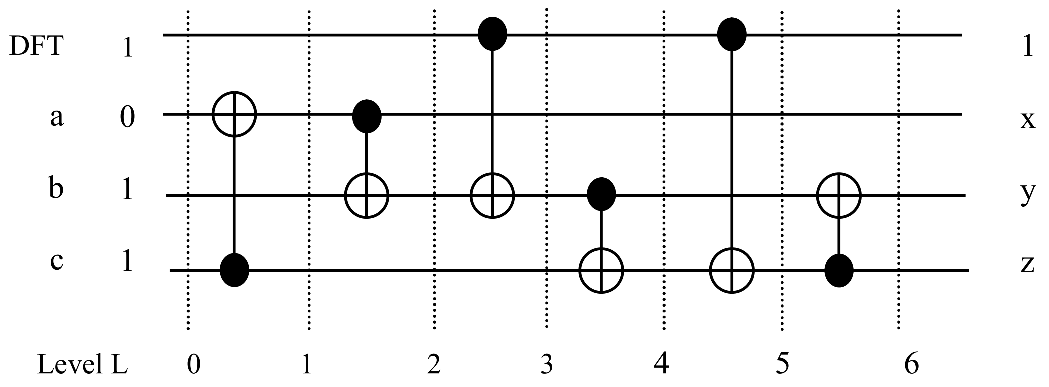


Figure 4.9: DFT based approach for missing gate testing [20].

This DFT based testing method begins by applying a test vector to the input of a circuit. The additional circuitry converts a bit of the vector to 1, at each gate level (when necessary), in order to test for a MGF. When the value of the DFT line is 0, the circuit performs its normal mode of operation. For testing a circuit, the DFT line must be set to 1. Figure 4.9 shows an example of a DFT based approach for testing a MGF. In this figure, the lines labeled 'a', 'b' and 'c' are part of the 3-bit original circuit. According to this approach, an additional line labeled DFT is added to the original circuit. The figure shows that the original circuit consists of four gates. The original gates are not connected to the DFT line. Two extra 1-CNOT gates are used to set the control points of next gates to 1. Figure 4.9 shows that the test vector is $(a,b,c) = (0,1,1)$, and the DFT line is 1. The test vector is chosen in such a way that the vector must set the control point of the gate which appears first to 1. The values at different levels of this circuit are presented in Table 4.7. It is seen from Table 4.7 that for the input test vector $(1,0,1,1)$, the DFT based testing approach sets 1 to control points of each gate in the circuit. The immediate effects of the two additional 1-CNOT gates can be observed at gate levels 3 and 5 from Table 4.7. For the test vector $(1,0,1,1)$, the circuit generates the output vector $(1,1,0,1)$. Any value other than this output vector indicates that a MGF is present in the circuit.

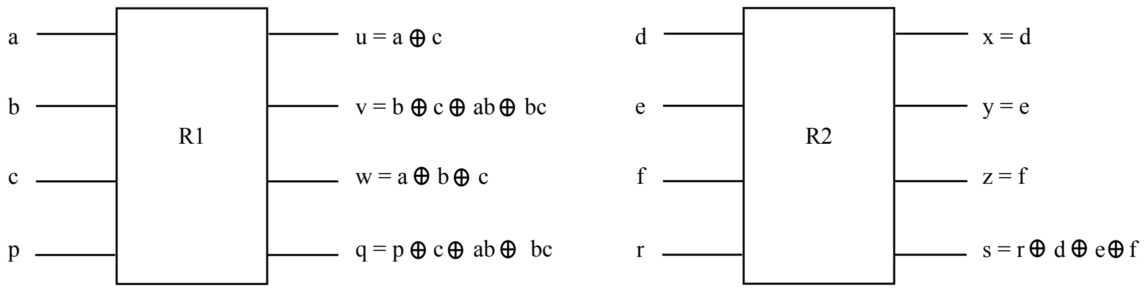
Table 4.7: Logic values at different levels to detect MGFs for the circuit, shown in Figure 4.9.

Gate Levels						
0	**1**	**2**	**3**	**4**	**5**	**6**
1	1	1	1	1	1	1
0	1	1	1	1	1	1
1	1	0	1	1	1	0
1	1	1	1	0	1	1

4.3.2 Online Approaches

In online approaches, a system is tested while the system performs its normal operations. Unlike in offline testing, it is not necessary to remove a system from its normal mode of operations. Thus online testing approaches increase system throughput, because a system never sits idle for testing purposes. With this approach, additional circuitry is attached to a system for testing. So rather than generating test vectors as in offline testing approaches, the primary design issue of online approaches is to design a minimal circuit to detect faults while the circuit is operating normally. This section describes some online testing approaches for reversible circuits.

Vasudevan et al. [69] proposed three new reversible logic gates for online testability.



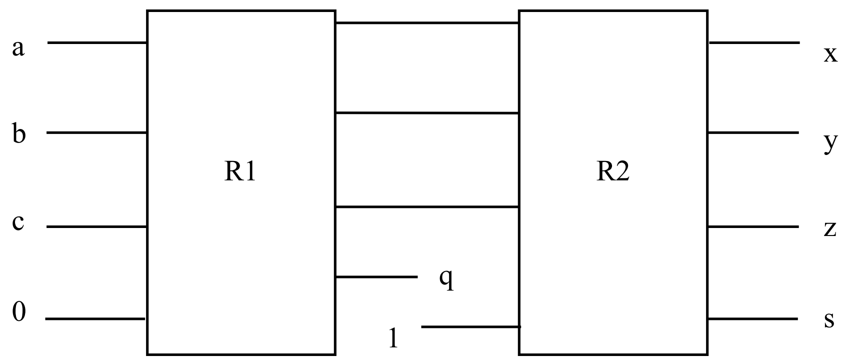
(a) A R1 gate. (b) A R2 gate.

Figure 4.10: R1 and R2 gates proposed in [69].

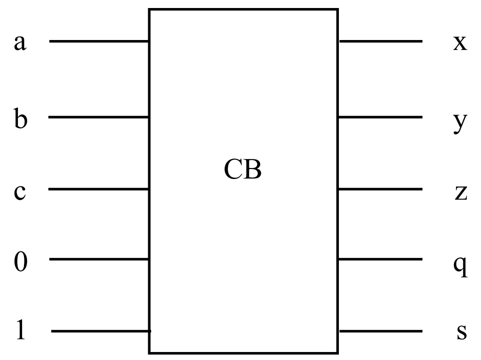
Two of the gates, $R1$ and $R2$, are used to create an online testable block, CB. Figure 4.10 shows the block diagrams and output functionalities of the $R1$ and $R2$ gates. $R1$ gates are universal gates; that is, $R1$ gates can be used to implement any reversible function. The idea behind the development of the $R2$ gate is to add online testable features to the $R1$ gate. When these two gates are connected in cascade, they work as a block with testable functionalities.

Figure 4.11(a) shows how the outputs of $R1$ are connected to the inputs of $R2$ in order to form a CB, as shown in Figure 4.11(b). As shown in this figure, the values of input lines p and r are set to 0 and 1, respectively, during normal operations. In a CB, the $R1$ gate is used to synthesize logic functions, and the $R2$ gate is used to add testing feature to the circuit. The input lines p and r and the output lines q and s are used as parity bits. If the $R1$ gate contains a fault resulting in incorrect output, the values of q and s will be the same. Opposite values of q and s indicates fault-free operations.

Another reversible gate, R, is also proposed by Vasudevan et al. [69]. The purpose of the R gate is to behave as a checker circuit and check the parity outputs of the CB. The checker circuit, which is called a two-pair two-rail checker, is constructed by connecting eight R gates. The checker circuit has four input lines and two output lines. The four input lines of the two-pair two-rail checker connects the parity output bits from two identical CBs. Finally, $e1$ and $e2$ produce opposite values if the input bits are the complement of each other, which indicates that the circuit generates a correct output. $e1$ and $e2$ are evaluated as



(a) $R1$ and $R2$ gates are connected in cascade.



(b) A $R1$-$R2$ pairing testable block.

Figure 4.11: Block diagrams of pairing of $R1$ and $R2$ reversible gates.

$e1 = x0y1 + x1y0$, and $e2 = x0x1 + y0y1$, respectively.

A similar online testing approach is proposed by Mahammad et al. [31]. There are three major phases to design an online testable circuit using this approach. Firstly, each gate of a circuit is transformed to a deduced reversible gate (DRG). Secondly, an identity gate is attached to the deduced reversible gate in order to form a testable reversible cell (TRC). Finally, parity outputs of each TRC are connected to build a testable circuit (TC). According to this approach, a reversible gate is transformed to its corresponding DRG by adding one parity input line, p_i, and one parity output line, p_o. The output parity line, p_o, performs exclusive OR operations between input data bits and the input parity bit, p_i. Thus, each n-bit gate is transformed to a $(n+1)$-bit DRG. For example, Figure 4.13 shows how a CNOT gate is transformed to its corresponding deduced version.

The DRG is also reversible and retains the original functionality of a gate. With this

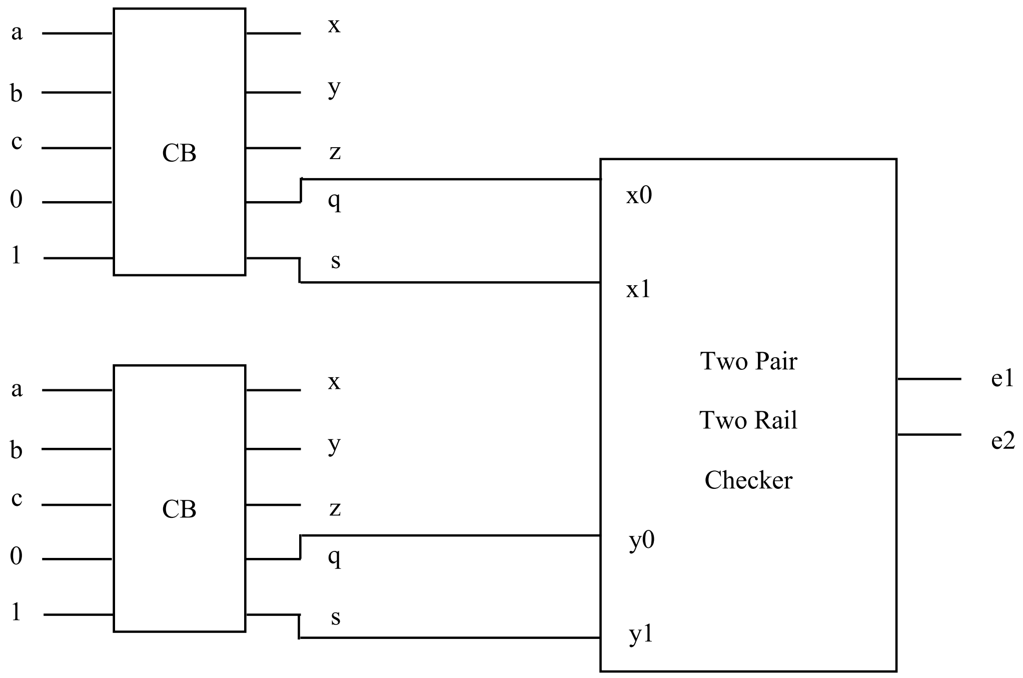
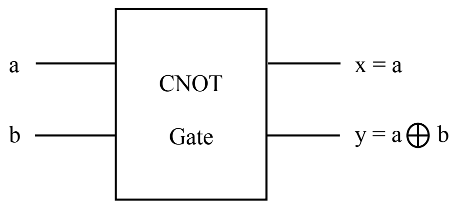
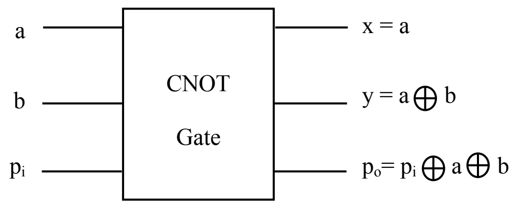


Figure 4.12: An online testable circuit based on the approach proposed in [69].



(a) Block diagram of a CNOT gate.

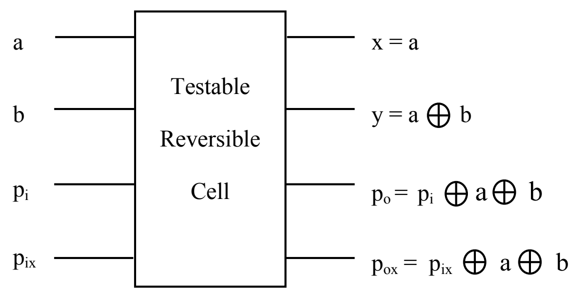


(b) Corresponding deduced version of a CNOT gate.

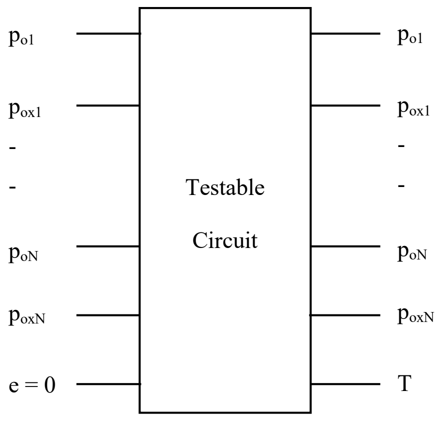
Figure 4.13: A CNOT gate and its deduced version for online testing.

approach, for an $(n \times n)$ reversible gate, an $(n \times n)$ identity gate is also required to achieve testability. The identity gate simply passes the input to its output lines. This identity gate is also transformed to its deduced form. Next, the input of the deduced identity gate is connected to the output of the deduced version of the original gate, which forms a $(n+2)$-bit TRC for a gate. For example, the TRC of the gate which is shown in Figure 4.13 (b) is presented in Figure 4.14 (a).

In Figure 4.14 (a), p_{ix} and p_{ox} are the parity input and output bits of the identity gate. The four input and output parity bits, as in this figure, can be used to determine whether the output of a gate is correct or not. As an indication of correct operation, when both


a
b
p_i
p_{ix}
Testable Reversible Cell
$x = a$
$y = a \oplus b$
$p_o = p_i \oplus a \oplus b$
$p_{ox} = p_{ix} \oplus a \oplus b$

(a) TRC for the gate shown in Figure 4.13(b).



(b) A general block diagram of a TC.

Figure 4.14: TRC and TC for online testability.

input parity bits are the same, the output parity bits should also be the same. Any unwanted change of a bit as a result of the occurrence of a fault results in the output parity bits being the complement of each other when the input parity bits are the same, which indicates that the output is incorrect. Each n-bit gate of a circuit is transformed to a $(n+2)$-bit TRC. Each TRC has n output bits and 2 output parity bits. These output parity bits of each TRC are connected to testable circuit, TC, as shown in Figure 4.14(b). A TC also contains another input line, e, which is generally initialized with 0. The value of the output line, T, of a TC is calculated as $T = e \oplus ((p_o1 \oplus p_ox1) + (p_o2 \oplus p_ox2) + ... + (p_oN \oplus p_oxN))$. When $T = 1$, it indicates that the output of the circuit is incorrect.

The online testing approaches presented in [31, 69] are not capable of detecting certain faults, as shown in [49]. For example, if two CBs, as shown in Figure 4.11(b), are required to implement a reversible function, and a fault occurs between the two CBs, the approach proposed in [69] cannot detect this fault. To address this Nayeem et al. [49] proposed an online testing approach. According to this approach, four modifications are required to make a reversible circuit online testable. Firstly, for each input line of an $(n \times n)$ reversible circuit, it is necessary to add a 1-CNOT gate before and after the original circuit. Secondly, a parity line, L, is added to the circuit. The target lines of the added CNOT gates connect the parity line, L. The third modification is that all n-bit Toffoli gates of the original circuit are replaced by $(n+1)$-ETG gates. ETG stands for extended Toffoli gate, which is a generalized version of a Toffoli gate. The difference between an ETG and a Toffoli gates is that an ETG has more than one target line, as shown in Figure 4.15. These three modifications are necessary to design an online testable reversible circuit based on the approach in [49]. However, if an odd number of NOT gates are present in an original circuit, another modification is required. In this case an extra NOT gate is added at the end of line parity line, *L*. If the number of NOT gates are even, the fourth modification is not required.

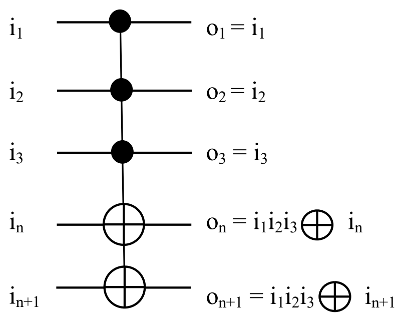
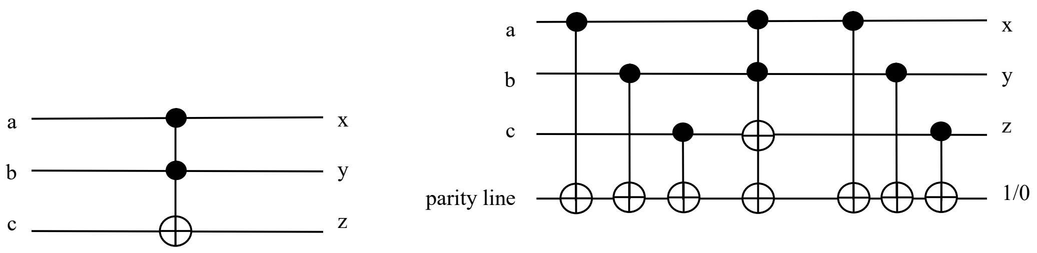

i_1
$o_1 = i_1$
i_2
$o_2 = i_2$
i_3
$o_3 = i_3$
i_n
$o_n = i_1i_2i_3 \oplus i_n$
i_{n+1}
$o_{n+1} = i_1i_2i_3 \oplus i_{n+1}$

Figure 4.15: A $(3+1)$-bit ETG gate.

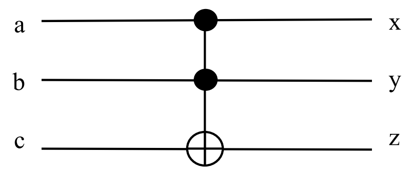


(a) A reversible circuit.

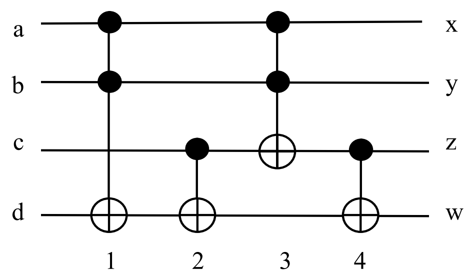
(b) An online testable circuit.

Figure 4.16: An example of online testable reversible circuit based on [49].

Figure 4.16(b) shows an online testable version of the reversible circuit shown in Figure 4.16(a). The initial value of the parity line *L* of a testable circuit is set to 0. At the circuit output, if the value of *L* becomes 1, it indicates the circuit is faulty. Targets of an ETG operate on the same set of control points as that of the original gates. So if a fault affects the output of the original target of a gate, the fault also affects the value on *L*, which is connected to the extended target of the gate. The purpose of the CNOT gates which are added before ETG gates is to read the parity of the line, *L*. Another set of CNOT gates is added after ETG gates for checking the parity on *L*. Targets of all these CNOT gates are connected to *L*, as shown in Figure 4.16. Thus, if a fault occurs and affects the output of a gate, the fault changes the value of *L* from 0 to 1, which indicates the system output is incorrect.



(a) A reversible circuit.



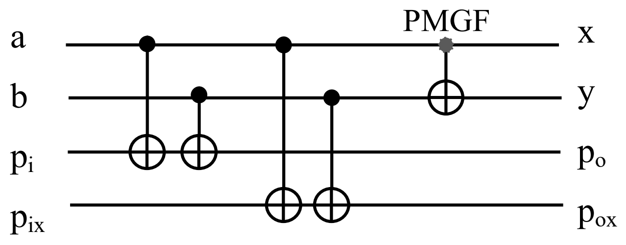
(b) An online testable circuit.

Figure 4.17: An example of online testable reversible circuit based on [28].

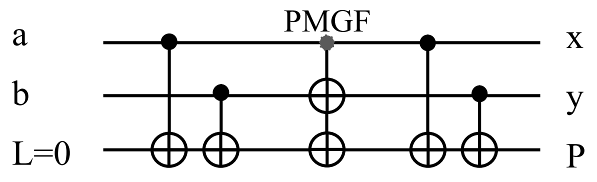
The approaches discussed above consider only single bit faults. Kole et al. [28] proposed an online testing approach that can detect SMGFs in NCT based circuits. According to this approach, each *k*-CNOT gate of the original circuit is transformed to its corresponding augmented reversible gate (ARG). Their approach also requires one additional parity line. An ARG contains four gates: three additional gates and the original gate. The targets of the additional gates connect to the parity line. When the output parity bit is the complement of the input parity bit, it indicates the output in incorrect. For example, Figure 4.17(b) shows a testable version of the reversible circuit shown in Figure 4.17(a). Suppose the gate labeled 3 in Figure 4.17(b) is missing. When the input vector $(a,b,c,d) = (1,1,0,1)$, the output vector will be $(x,y,z,w) = (1,1,0,0)$. This output is incorrect, which is identified as the value of 'w' is the complement of the value of 'd'. When the gate labeled 4 is missing, this fault will not affect the original circuit output, (x,y,z). However, the testable circuit does not always identify the occurrence of this fault. For example, when the input vector $(a,b,c,d) = (1,0,1,0)$, the output vector will be $(x,y,z,w) = (1,0,1,1)$. In this case $d \neq w$, which identifies the SMGF. However, when the input vector $(a,b,c,d) = (1,1,1,0)$, the output vector will be $(x,y,z,w) = (1,1,0,0)$. In this case $d = w$, which does not identify the SMGF.

As seen from the above discussion, most existing testing approaches rely on input and output parities in order to detect faults. Other research that also falls in this category includes [19, 23, 55, 61, 65]. Many authors address the problem of testing by proposing parity preserving gates in terms of block diagrams, and checks the parity of inputs and outputs of the parity preserving blocks. Moreover, as we have seen earlier, many works do not test their proposed approaches for reversible fault models. In recent years, Przigoda et al. has tested existing reversible online fault detection approaches against reversible fault models. They showed that parity preserving blocks are inadequate to detect missing gate faults in reversible logic [57]. The limitation of detecting a single bit fault by the approach proposed in [69] is described in [49]. Simply maintaining parities between the inputs and the outputs is not always enough to detect faults in reversible logic. For example, Figure 4.18 shows two reversible circuits based on two different online testing approaches. Figure 4.18(a) shows a TRC of a CNOT gate based on the approach proposed by Mahammad et al. [31]. As we discussed earlier, the input lines p_i and p_{ix} are the parity lines from two DRGs. So for a single gate of a circuit, four additional gates are required to form a TRC for the gate. Figure 4.18(b) shows an online testable version of a CNOT gate based on the approach proposed in [49]. With this approach, four additional CNOT gates are added to the online testable circuit, and the CNOT gate is transformed to its extended version, as we discussed earlier in this section. The value of the output parity, P, determines whether the circuit is faulty or fault free. If $P = 0$, it indicates that the output is correct.

Suppose the PMGF occurs in the circuits, as indicated in Figures 4.18(a) and (b). Since the control of the CNOT gate in Figure 4.18(a) is missing, the CNOT gate will work as a NOT gate with a target on line, 'b'. Thus, when $a = 0$ and $b = 1$, the correct output should be $x = 0$ and $y = 1$. However, from Table 4.8(a), it is seen that when $ab = 01$, the value of $xy = 00$, which in incorrect. With the approach presented in [31], as an indication of



(a) A TRC of a CNOT gate, according to the approach in [31].



(b) An online testable circuit for a CNOT gate, according to the approach in [49].

Figure 4.18: Two existing reversible online testing approaches for PMGF.

fault free circuits, when $p_i = p_{ix}$, output parities should be $p_o = p_{ox}$; and when $p_i \neq p_{ix}$, output parities should be $p_o \neq p_{ox}$. However, Table 4.8(a) shows that the output parities are the complement of each other even when the circuit generates an incorrect output, that is, $xy = 00$ when the inputs are $ab = 01$. A similar observation can be seen in Table 4.8(b). Due to the presence of a PMGF, the extended CNOT gate transformed to two NOT gates on line '*b*' and '*L*'. When $ab = 01$, the circuit shown in Figure 4.18(b) generates $xy = 00$, which is incorrect. However, the output parity line is still 0. A 0 on output parity line indicates that the output is correct, though the output is not correct in this case. Thus, the approach presented in [49] also fails to detect a PMGF. Similarly, the approaches presented in [31] and [49] can not detect incorrect output for other reversible fault models, such as SMGFs, RGFs, and crosspoint faults.

Our proposed approach is based on the testing technique presented in [49]. Nayeem et al. [49] examined their proposed approach against only single bit fault model. In addition, the authors only consider the case when a fault occurs in an original circuit. They did not consider the occurrence of faults in the extra circuitry which is added to the circuit. In this

Table 4.8: Truth tables of the circuits shown in Figure 4.18.

(a) Output of the circuit shown in Figure 4.18(a).

inputs				outputs			
a	b	p_i	p_{ix}	x	y	p_o	p_{ox}
0	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0
1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1

(b) Output of the circuit shown in Figure 4.18(b).

inputs			outputs		
a	b	L	x	y	P
0	0	0	0	1	0
0	1	0	0	0	0
1	0	0	1	1	0
1	1	0	1	0	0

work we consider all possible locations where a fault may occur in a circuit. In addition, we have tested our proposed online testing approach for three categories of reversible fault models.

4.4 New Online Testing Approach [46]

In this section, we are presenting our proposed online fault testing approach for detecting three types of faults in reversible circuits.

4.4.1 Design

Given a reversible circuit with L lines and N gates, the first step to transform a reversible circuit to its online testable equivalent is to add an extra line to the circuit. This line is a parity line, p, which is initialized to 0. The next step is to transform each k-CNOT gate of the circuit into a duplicate gate block. For each line of the original circuit, a 1-CNOT gate is added at the beginning and at the end of the original circuit. The targets of the additional CNOT gates are connected to the parity line, p.

A duplicate gate block (DGB) consists of two gates. A DGB includes a k-bit Toffoli gate with an additional k-bit Toffoli gate, as shown in Figure 4.19. The controls of the newly added gate (or duplicate gate) are on the same lines as that of the original gate. However, the target of a duplicate gate is connected to the parity line, p. In the case of a 0-CNOT gate, there is no control line, so a DGB consists of two 0-CNOT gates: one on the same line as

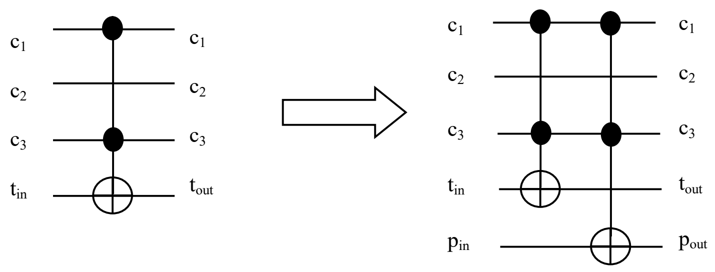
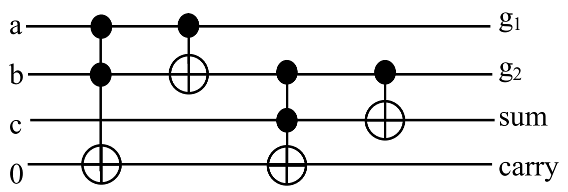


Figure 4.19: Conversion of a Toffoli gate into a Duplicate Gate Block.

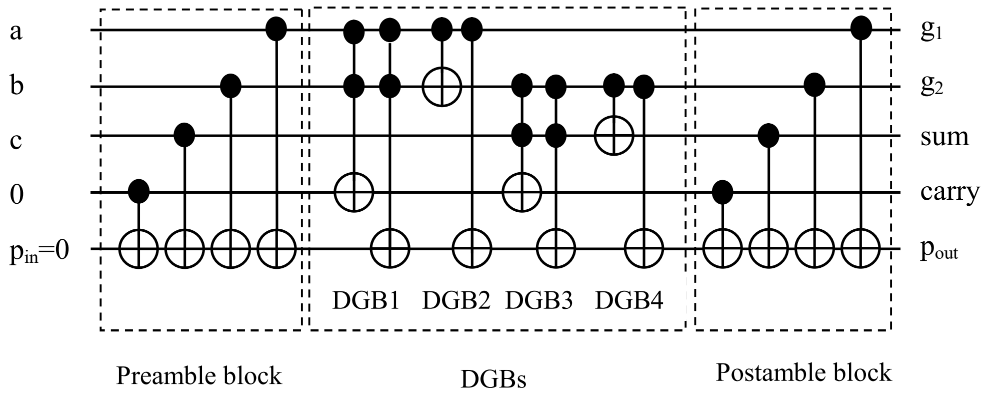
the original reversible gate and another on the parity line, p. DGBs appear in the same order as the gates in the original circuit. A total of L 1-CNOT gates are added at the beginning and the end of DBGs. We refer to the set of 1-CNOT gates which appears before DBGs as the preamble block. The set of 1-CNOT gates which appear after the DGBs, are referred to as the postamble block. Figure 4.20 illustrates the conversion. Figure 4.20(a) shows a reversible full adder circuit consisting of four gates. These four gates are transformed into their corresponding DBGs in an online testable version of a full adder circuit, as shown in Figure 4.20(b). A full adder circuit has four lines, as shown in Figure 4.20(a). Thus, four 1-CNOT gates are added in the preamble and postamble blocks. Targets of these additional 1-CNOT gates are connected to a parity line, p_{in}.

With our proposed approach, an entire testable circuit consists of three blocks in sequence: preamble block, DBGs and postamble block. If the QC of an original circuit is Q, and the circuit has L lines, the QC of the circuit's online testable equivalent will be $2L+2Q$, since the QC of a 1-CNOT gate is 1 [32, 71].

One major advantage of this approach is that circuit overhead reduces significantly for circuits with higher GC. The number of CNOT gates of a testable circuit remains the same if the number of gates in the original circuit increases. In addition, online testable circuits based on this approach are easy to design. That is, this approach reduces design complexity by simplifying the testable portion of the circuit. In addition, the design of this circuit is dynamic in nature. If a gate is added or removed from the original circuit, a CNOT gate on the corresponding line is added or removed.



(a) A full adder reversible circuit.



(b) Online testable equivalent of the full adder.

Figure 4.20: Transformation of a reversible circuit into its online testable equivalent.

4.4.2 Analysis

The first step before analysing the capability or behaviour of the proposed online testable circuit in detecting faults is to ensure whether the testable circuit performs its intended operations after adding extra circuitry. Figure 4.20(b) shows an online testable full adder circuit based on our proposed approach. As we see from the figure, the full adder circuit has three data input bits and two other inputs. Table 4.9 shows the truth table for the online testable full adder circuit. The sum and carry output bits are generated after adding three inputs a, b, and c. The output parity line is labeled as p_{out}, and g_1 and g_2 are two garbage outputs. It is important to observe that the output parity bits, p_{out} equals to 0 for all the rows, which indicates that the output is correct.

Figure 4.21 shows a general diagram of an online testable reversible circuit. P_x and Q_{xy} represent the parity line and the common lines, respectively. Targets and control lines of gates are treated as common lines. The outputs at the preamble block can be determined as follows: $Q_{11} = Q_{10}$, $Q_{21} = Q_{20}$,...,$Q_{L1} = Q_{L0}$ and $P_1 = P_0 \oplus Q_{10} \oplus Q_{20} \oplus Q_{30} \cdots \oplus Q_{L0}$.

Table 4.9: Truth table for the circuit shown in Figure 4.20(a)

inputs					outputs				
a	b	c	0	p_{in}	g_1	g_2	sum	carry	p_{out}
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	1	0	1	0
1	0	0	0	0	1	1	1	0	0
1	0	1	0	0	1	1	0	1	0
1	1	0	0	0	1	0	0	1	0
1	1	1	0	0	1	0	1	1	0

Since the parity line is initialized to 0, then $P_0 = 0$ and $P_1 = Q_{10} \oplus Q_{20} \oplus Q_{30} \cdots \oplus Q_{L0}$.

From the above equation, it can be observed that the preamble block works as a parity checker. That is, if the parity of the common lines at the input (level 0) is odd, after passing through the preamble block, the value on the parity line (P_1) at level 1 changes to 1. If the parity of the common lines at the input (level 0) is even, the value of the parity line, P_1, at level 1 remains 0. The output values of the preamble block on the common lines will equal to the input values. Thus, the value on parity line will be 1 when the parity of the common lines is odd. On the other hand, the value on parity line will be 0 when the parity of the common lines is even. This property is termed as the parity property.

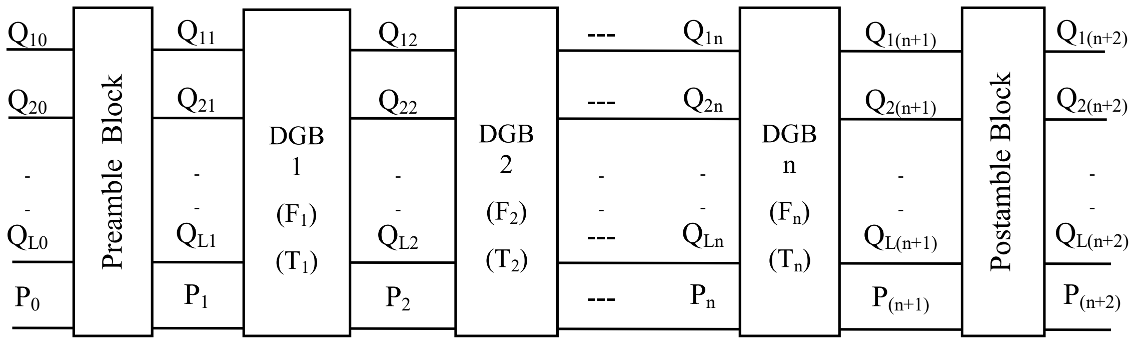


Figure 4.21: Block diagram of an online testable reversible circuit.

The output of the preamble block becomes the input to the cascade of DGBs. If there is no fault in the preamble block, the DGBs also preserve the parity property. Let F_x be the output function of a DGB. Let T_x and P_{x+1} be two target lines of the original gate and the duplicate gate of a DGB, respectively. The target line of a duplicate gate is always the parity line whereas the target line of the original gate is one of the common lines. For example, Figure 4.20(b) shows that the targets of the duplicate gates are on the parity line, whereas the targets of original gates can be on any line other than the parity line. Therefore, T_x is one of the lines amongst $\{Q_{1(x+1)}, Q_{2(x+1)}, \ldots, Q_{L(x+1)}\}$. Let $T_x = Q_{i(x+1)}$ where $i \in (1,2,3,\ldots,L)$; then $T_x = F_x \oplus Q_{ix}$ and $P_{(x+1)} = F_x \oplus P_x$.

From the above two equations, it is observed that if $F_x = 1$, $P_{(x+1)}$ and T_x invert the input values, P_x and Q_{ix}, respectively. If $F_x = 0$, the output of the DGB will equal to its input, and no change will take place. Thus, changes in T_x and $P_{(x+1)}$ take place simultaneously. In other words, changes in the parity of the common lines and $P_{(x+1)}$ take place simultaneously. We refer to this property of the DGB as simultaneous change property. The simultaneous change property ensures that the parity property remains consistent throughout the output of the DGBs. Thus, if the parity property is violated at the input of a DGB, the effect of this violation is passed to the output of the DGB.

The output of the cascade of the DBGs supplies the input to the postamble block. If there is no fault in any of the previous blocks, the input of a postamble block also satisfies the parity property. That is, if the parity of the common lines is even (odd) at level $(n+1)$ in Figure 4.21, the input parity, P_{n+1}, will be $0(1)$. The output equations of the postamble block are: $Q_{1(n+2)} = Q_{1(n+1)}$, $Q_{2(n+2)} = Q_{2(n+1)}$, $\ldots$, $Q_{L(n+2)} = Q_{L(n+1)}$; and $P_{(n+2)} = P_{(n+1)} \oplus Q_{1(n+1)} \oplus Q_{2(n+1)} \oplus Q_{3(n+1)} \cdots \oplus Q_{L(n+1)}$

From the above equation it is seen that if the parity of the common lines is odd at level $(n+1)$, then $P_{(n+1)}$ equals to 1 and hence, the output parity, $P_{(n+2)}$ becomes 0. On the other hand, if the parity of the common lines at level $(n+1)$ is even, then $P_{(n+1)}$ equals to 0, and $P_{(n+2)}$ will still be 0. Therefore, in a fault-free circuit operation, the input of the postamble block preserves the parity property, and the final output parity $P_{(n+2)}$ of the circuit will be 0. When the value of $P_{(n+2)}$ becomes 1, it indicates an incorrect output.

4.4.3 Testing: Fault Detection

This section shows our proposed online testing method to identify incorrect output in the presence of faults in a reversible circuit. We have considered different types of reversible fault models to illustrate our approach. We assume that at any given instant, at most, one fault is present in a reversible circuit. Faults that affect the output of a circuit change the value of the output parity bit from 0 to 1. That is, 1 at the output parity line indicates that the output is incorrect. We have considered three scenarios from the perspective of the location of occurrence of a fault in a circuit: **case 1:** when a fault occurs in an original gate of a DGB; **case 2:** when a fault occurs in a duplicate gate of a DGB; and **case 3:** when a fault occurs in either preamble or postamble blocks.

Missing Gate Fault Family

This section illustrates the mechanism of identifying MGFs in our proposed online testable reversible circuits. Figure 4.22 shows occurrences of a SMGF and PMGF in a testable version of a full adder circuit. However, we assume only one type of fault is present at a time in the circuit. In this figure the CNOT gates labeled from 1 to 4 constitute the preamble block. Gates marked from 5 to 12 are part of a series of DGBs. Gates 5 and 6 form a single DGB, where the gate labeled 5 is the gate from the original circuit and the gate labeled 6 is the duplicate gate. Note that targets of all duplicate gates in DBGs are connected to p_{in}. Gates from 13 to 16 are part of the postamble block, as indicated in Figure 4.22.

Single Missing Gate Fault and Repeated Gate Fault

Case 1: Suppose the original gate in a DGB is missing. A missing gate does not affect the circuit output if any of this gate's control points are 0. So in order to demonstrate how our approach identifies the occurrence of faults, we are considering an input vector that sets all control points of the original gate to 1. The output of the original gate is connected to one of the common lines. Recall from section 4.4.2 that targets and control lines of the gates of the testable circuit are treated as common lines. Because the original gate is missing, there will no change in common lines. However, since the duplicate gate of this DGB is connected to the parity line, and all control points of this gate are 1, the value on the parity line will be inverted. Therefore, this DGB output will violate the parity property. According to the simultaneous change property, this violation will be forwarded to the input of the postamble block. When the inputs of the postamble block do not follow the parity property, the parity line will generate 1 at its output, which indicates that the circuit is faulty.

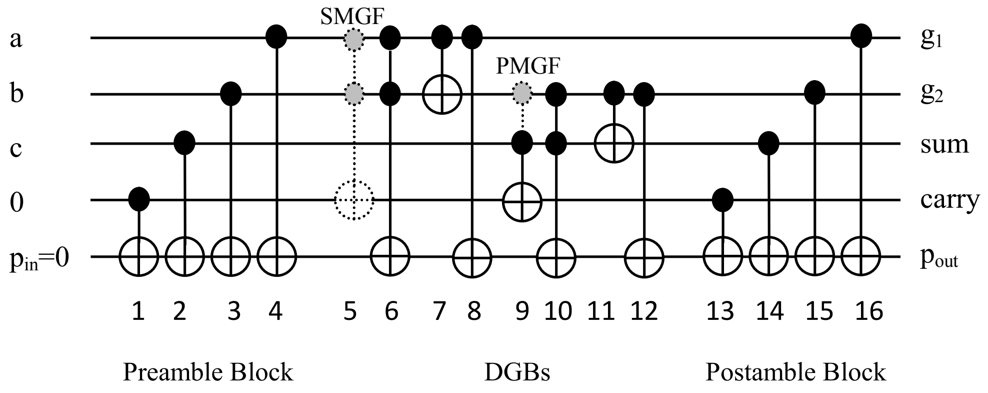


Figure 4.22: Testing for the occurrences of SMGF and PMGF.

For example, we assume that the original gate labeled as 5 and indicated by a dotted line in Figure 4.22 is missing. Note that gate 5 and gate 6 form a single DGB, where gate 5 is the original gate and gate 6 is the duplicate gate. The target of the duplicate gate is on the parity line, and control points of both gates are on the same line. If, during normal operations, the input vector is $(a,b,c,0,p_{in}) = (1,1,0,0,0)$, the duplicate gate (gate 6) inverts its target input, as both control points are 1. However, since the original gate is missing, the original gate (gate 5) will not invert its target input. So for this input vector, $(1,1,0,0,0)$, the circuit output will be $(g_1,g_2,sum,carry,p_{out}) = (1,0,0,0,1)$ instead of the correct output $(1,0,0,1,0)$. Similarly, when $(a,b,c,0,p_{in}) = (1,1,1,0,0)$, the circuit output will be $(g_1,g_2,sum,carry,p_{out}) = (1,0,1,0,1)$. It is seen from both examples that output parity, $p_{out} = 1$, which indicates that the circuit is faulty.

Case 2: Now suppose a SMGF occurs in the duplicate gate (gate 6) of a DGB in Figure 4.22. Similar to case 1, the fault will be identified for input vectors that set all control points of the duplicate gate to 1. For such an input vector, the original gate (gate 5) inverts its target input. So the output of the common lines will change because the output of the common lines depends on the original gate. However, since the target line of the faulty gate is connected to the parity line, the output parity line of this DGB will not be changed. So there is a violation of the simultaneous change property. This violation will also affect the parity property at the input of the postamble block. As a result, the final output parity of the circuit will be 1, which identifies the fault.

For example, when $(a, b, c, 0, p_{in}) = (1, 1, 0, 0, 0)$, the gate 5 inverts its target input, as both control points are set to 1. However, the gate 6 will not invert its target bit. So the simultaneous change property is violated, and the circuit output becomes $(g_1, g_2, sum, carry, p_{out}) = (1, 0, 0, 1, 1)$, and the output parity, p_{out} becomes 1, which indicates that the output is incorrect.

Case 3: Now consider the case when one of the gates in the preamble block is missing. Target lines of all 1-CNOT gates appear in the parity line, so a fault in the preamble block does not affect the common lines output. However, the fault must affect the parity output, P_1, which is given by $P_1 = Q_{10} \oplus Q_{20} \oplus Q_{30}, \ldots, Q_{L0}$. According to the parity property, the output parity bit of the preamble block should be 0 if the parity of the common lines is even, and the parity output should be 1 if the parity of the common lines is odd. Since one of the 1-CNOT gates is missing, the fault violates the parity property. This violation affects the output parity line of the circuit, and the output parity line, p_{out}, will be 1, which indicates an incorrect output.

As far as RGFs are concerned, if a gate is replaced by an odd number of instances of the same gate, the fault does not affect the circuit output. However, if the number of instances is even, the effect of this fault is identical to that of a SMGF [56]. Thus, the parity output of the proposed online testable circuit will be 1 for the occurrence of RGFs.

Partial Missing Gate Fault

Case 1: If a PMGF occurs in an original gate of a DGB, some control points of the gate become inactive. For such a fault to be detected, at least one of the missing control points should be 0, and the rest of the control points of the faulty gate should be 1 [56]. Thus, when the missing control point is 0, and all non-missing control points are 1, the faulty gate will invert its target input. Since the target of the original gate is one of the common lines, the parity of the common lines will be changed. However, all the control points of the duplicate gate are not 1. So the duplicate gate will not invert its target input. Thus, the value on the parity line of the DGB will not be changed. As a result, the parity property will be violated at the output of this DGB. According to the simultaneous change property, this violation will propagate throughout the cascade of DGBs to the input of the postamble block. In this case the output of the circuit will be incorrect, as indicated by 1 on the output parity line.

For example, suppose the control point of the gate labeled 9, as indicated in Figure 4.22, is missing. Since the control point of the 2-CNOT gate 9 is missing, this gate will work as a 1-CNOT gate. So when the input vector $(a,b,c,0,p_{in}) = (0,0,1,0,0)$, this gate will be active and invert its target input. However, both control points of the duplicate gate labeled 10 are not 1. So this duplicate gate remains inactive and will not invert its target input. As a result, the simultaneous change property will be violated and the final output of this circuit will be $(g_1,g_2,sum,carry,p_{out}) = (0,0,1,1,1)$, which is incorrect and can be identified as $p_{out} = 1$. Similarly when $(a,b,c,0,p_{in}) = (1,1,1,0,0)$, the output is $(g_1,g_2,sum,carry,p_{out}) = (1,0,1,0,1)$ instead of the correct output $(1,0,1,1,0)$. This fault can also be identified as $p_{out} = 1$. Therefore, a PMGF that may occur in a DGB's original gate can be identified.

Case 2: Now consider a case when a control point of a duplicate gate is missing. When the non-missing control points are 1 and the missing control point is 0, the duplicate gate will invert its target input. That is, the value of the parity line immediately after this duplicate gate will be inverted, since the target of the duplicate gate lies on the parity line. However, the common lines simply pass the inputs of the original gate to the output, as the original gate will be inactive. So the target of the original gate will not invert its input. Thus, there is a change in the value of the parity line, but parity of the common lines remains the same, which violates the simultaneous change property. As a consequence, the output parity of the circuit will be 1, which identifies the fault.

For example, a PMGF occurs in the duplicate gate labeled as 10 in Figure 4.22. Consider a case when the control point on line '*b*' of the duplicate gate (gate 10) is missing. So the gate labeled 10 becomes a 1-CNOT gate with the control on line '*c*'. Now, when $(a,b,c,0,p_{in}) = (0,0,1,0,0)$, the duplicate gate will be active, while the original gate will be inactive. The output of the online testable circuit will be $(g_1,g_2,sum,carry,p_{out}) = (0,0,1,0,1)$, which is incorrect, as indicated by the value of the output parity.

Case 3: If a PMGF occurs in the preamble block, a 1-CNOT gate will become a 0-CNOT gate. A 0-CNOT gate will change every bit on its input. Thus, the output parity bit of the preamble block will always be the opposite of the correct value. As in previous cases, this incorrect parity will propagate to the successive blocks of the circuit, and the circuit output will be affected. A similar situation will happen if a PMGF occurs in a postamble block. In both cases, the parity output will be 1.

Crosspoint Fault

This section discusses the mechanisms to detect disappearance and appearance crosspoint faults. A disappearance fault is identical to a PMGF. Thus, the effect of a disappearance fault and its detection mechanism is the same as that of PMGF. Here we discuss the detection of appearance faults.

Case 1: If an appearance fault occurs in an original gate of a DGB, one or more extra control points are added to the gate. This fault is detectable if at least one of the additional control points is 0, while other control points are 1. In this case the target of the faulty gate does not invert its input bit. However, the target of the duplicate gate in the DGB inverts its target input, which connects to the parity line. As a result, DGB output does not satisfy the parity property, and the circuit generates an incorrect output.

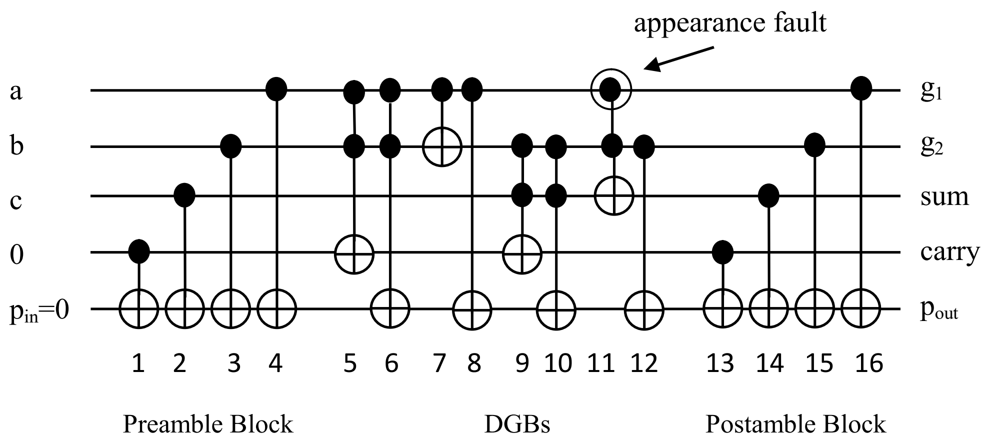


Figure 4.23: Testing for the occurrences of appearance crosspoint fault.

For example, suppose an appearance fault occurs in gate lebeled 11 in Figure 4.23. The original gate's control point which is on line '*a*' appears as a fault, as indicated in this figure. Gate labeled 11 becomes a 2-CNOT gate due to the presence of this fault. So for the input vector $(a, b, c, 0, p_{in}) = (0, 0, 1, 0, 0)$, gate 11 becomes inactive, as the control point on line '*a*' equals to 0. However, the duplicate gate 12 becomes active, as it is a 1-CNOT gate and the control of this gate is '*b*' is 1. So gate 12 inverts its target input. In this case the circuit will generate an incorrect output vector $(g_1, g_2, sum, carry, p_{out}) = (0, 1, 1, 1, 1)$. The parity output, p_{out}, becomes 1, which indicates that the circuit is faulty.

Case 2: When an appearance fault occurs in a duplicate gate of a DGB, the fault is detectable when any of the extra control points is 0, and all other control points are 1. In this case the original gate inverts the bit at the gate's target line. However, the control points of the duplicate gate are not 1, so the parity output remains the same as its input. As a result, similar to as in case 1, the parity property is violated. The violation will affect the circuit output, and the incorrect output can be identified by observing the parity output of the circuit.

For example, suppose a control point on line '*a*' is added to the duplicate gate 12 in Figure 4.23. In this case when the input vector is $(a, b, c, 0, p_{in}) = (0, 0, 1, 0, 0)$, the output vector is $(g_1, g_2, sum, carry, p_{out}) = (0, 1, 0, 1, 1)$. It is seen that the parity output is 1, which indicates an incorrect output.

Case 3: If an extra control point appears on a 1-CNOT gate of the preamble block, the fault will affect the circuit output only when the newly added control point is 0 and the old control point is 1. In such a case the gate will not invert its target bit. Thus, the preamble block generates a an incorrect output, which will not satisfy the parity property. Similarly if an appearance fault occurs in the postamble block, the parity output will be 1.

For example, when $(a, b, c, 0, p_{in}) = (0, 0, 1, 0, 0)$, the output vector, $(g_1, g_2, sum, carry, p_{out})$, becomes $(0, 0, 1, 0, 1)$. The output parity bit is 1, which identifies that the output is incorrect.

Single Bit Fault

Single bit fault model deals with a situation when exactly one output of a circuit is incorrect because of a change in a bit on a line. If a single bit fault occurs in our proposed online testable circuit, the output of common lines will not follow the parity property. This violation of the parity property will be propagated to the input of the postamble block, because of the nature of the simultaneous change property of DGBs. The input of the postamble block, in turn, will not follow the parity property. Hence, the parity output will be 1, which indicates the fault in the circuit.

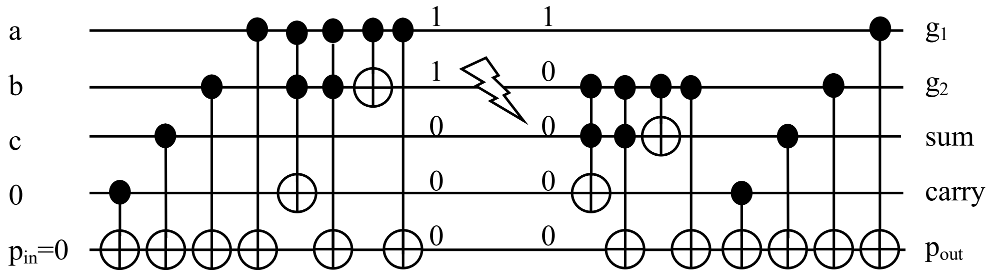


Figure 4.24: Testing for an occurrence of a single bit fault.

For example, suppose a single bit fault occurs between the second and third DGBs of a circuit as shown in Figure 4.24. When an input vector $(1,0,0,0,0)$ is applied to the circuit, the value on line '*b*' will be changed from 1 to 0. This change of bit violates the parity property. The corrected output of the circuit should be $(1,1,0,0,0)$; however the actual output, reflecting the fault, is $(1,0,0,0,1)$. The violation of the parity property is carried through the circuit and the value of the output parity line becomes high, which identifies the presence of the fault in the circuit in Figure 4.24.

If there is no fault in the preamble block and DGBs, the input of the postamble block will satisfy the parity property. However, if a fault occurs in the postamble block, the effect of this fault will be the same as that of the preamble block, as both preamble and postamble blocks are same in architecture. The parity output will also be 1 for the presence of a fault in postamble block.

To summarize, if a fault occurs in the original gate of a DGB, the output of the corresponding DGB does not satisfy the parity property. Moreover, because of the property of the simultaneous change property of DGBs, the violation of the parity property will be propagated to the input of the postamble block. When the input of the postamble block, in turn, does not follow the parity property, the output of the postamble block generates 1 on the parity line, which indicates that a fault occurs in the circuit.

4.4.4 Comparison and Limitation

Comparison

This section offers a comparison between our proposed approach and two other online testing approaches. Nayeem et al. [49] proposed an online testing approach for the detection of single bit faults. They used two sets of CNOT gates and a single parity line to make a reversible circuit online testable. In their approach, all *k*-Toffoli gates of an original circuit are transformed to $(k+1)$-Extended Toffoli Gates (ETG). We have designed an online testable reversible full adder circuit based on their approach proposed in [49]. The resulting

Table 4.10: Overhead for selected benchmark circuits.

Circuits		Original Circuit		Testable Circuit		Overhead (%)	
Functions	Lines	GC	QC	GC	QC	GC	QC
rd32	4	4	8	16	24	300%	200%
4b15g1	4	15	47	38	102	153%	117%
ham7	7	25	49	64	112	156%	129%
rd53	7	30	232	74	478	147%	106%
hwb8	112	449	1461	1122	3146	150%	115%
hwb9	170	699	2275	1738	4890	149%	115%
hwb10	10	3631	139470	7282	278960	101%	101%
frg2	1219	3724	12468	9886	27374	165%	120%

GC = Gate Count

QC = Quantum Cost

testable circuit has a GC of 12 and QC of 28. An online testable full adder circuit designed using our proposed approach is presented in Figure 4.20. This circuit has a GC of 16 and QC of 32. The QC and GC of our approach are slightly higher as compared to the approach in [49]. However, their approach only considers single bit faults, whereas our proposed approach can detect three types of faults.

Next we compare our approach with another online testing approach, proposed by Kole et al. [28]. Their approach requires each *k*-CNOT gate of the original circuit is transformed to its corresponding ARG consisting of four gates. Thus with their strategy, four gates are required to represent one gate. Therefore, in order to implement a full adder circuit, as shown in Figure 4.20(a), their approach requires a testable circuit with a GC of 16, which is the same as that of our approach. The QC of their testable circuit is 32, which is also the same as that of our approach. However, their approach considers only SMGFs.

Our proposed online testing approach is well suited for a circuit with a large number of gates. Table 4.10 presents the GC and QC of testable circuits after applying our approach to selected benchmark circuits [71]. From this table, it is seen that for circuits with a larger number of gates, our proposed approach actually results in a lower overhead (in terms of percentage of the original size). If we observe the first two benchmark circuits (with same number of inputs) from Table 4.10, the circuit overhead is significantly lower for the circuit $(4b15g1)$ with a higher GC. The reason behind this reduction is that the number of additional gates in preamble and postamble blocks does not depend on the number of gates in an original circuit, but rather depends on the number of bits (inputs) of the circuit. Circuits $rd32$ and $4b15g1$ have the same number of bits, however $4b15g1$ has almost four times more gates.

Compared with the approach presented by Kole et al. [28], our approach also offers efficient designs for circuits with a large number of gates, when we consider the QC. For instance, if we add one 1-CNOT gate to the original circuit presented in Figure 4.20(a), according to our approach the QC increases by 2, since the QC of a 1-CNOT gate is 1, and we duplicate the gate. However, according to the other approach [28], the QC increases by 4.

Limitations

There are some limitations of our proposed online testable circuit. Firstly, the proposed approach can not detect a single bit fault that occurs in a preamble block of a circuit. A single bit fault in a preamble block causes a reversible circuit to generate an incorrect output. However, in this case the output parity line will be 0. Thus, the output parity bit will not indicate that the circuit output is incorrect.

In addition, our approach fails to detect a particular case when dealing with MMGFs. MMGFs occur when several consecutive gates are missing in a circuit [56]. Suppose N consecutive original gates of a circuit become inactive, *i.e.* missing. N might be even or odd. For different input vectors, different gates amongst the missing gates may affect the input bits. Consider a case where an even number of gates are missing. Thus, the parity property will be violated an even number of times. When the parity property is violated an even number of times, the effect of the fault goes unnoticed on the parity line of a testable circuit. Thus, when the number of MMGFs is even, the output will be incorrect, however, the parity property will still be preserved. So the parity output will be 0, which will not indicate the presence of a fault, even though the output is incorrect.

Now consider an input vector for which the number of missing gates, which affects the parity line, are odd. The parity property is violated an odd number of times. Hence, the value of the parity output converts to 1, which clearly indicates an error in the output. Therefore, some but not all the possible MMGF faults are detectable by our proposed online approach.

4.5 Chapter Summary

4.5.1 Contribution

The chapter focuses on analysing and designing fault testing approaches for reversible circuits. The contribution of this chapter are as follows:

- Identify the limitations of the existing online testing approaches.
- Present an online testing approach for detecting three types of faults in reversible circuits.

4.5.2 Conclusion

In this chapter we have analysed fault models for reversible logic. We have observed the behaviour of different faults in reversible circuits. Studying and analysing the behaviour of faults is particularly important in order to develop fault testing strategies. We have also analysed existing testing strategies in reversible logic. Our study and analysis suggests that most existing reversible fault testing approaches have limited scope in fault detection. For example, some works have not considered the occurrences of faults in additional circuitry, some testing approaches have considered only one type of fault model, or some approaches have not considered reversible fault models.

This chapter presents an online testing approach [46] for reversible circuits based on the NCT gate library. With this approach, a reversible circuit can be converted to its online testable version by adding a set of CNOT gates and a single parity line. With these small modifications, we create a circuit that computes its original functionality, and in addition, the circuit will detect single bit faults, MGFs or crosspoint faults. The proposed approach requires $2(L+N)$ additional gates in order to make a reversible circuit consisting of L lines and N gates online testable. This increases the circuit overhead in terms of gate count and quantum cost, however in return, the circuit becomes online testable for three fault models. Our approach can also detect a fault even if the fault occurs in the additional circuitry, unlike other approaches in the literature. In addition, since the number of additional CNOT gates depends on the number of bits of a reversible circuit, the percentage of overhead in terms of gate count and quantum cost will be reduced for a circuit with larger gate count.

We have considered different fault scenarios in a reversible circuit and observed the output. If a fault occurs in the original gate of a circuit, and affects the input vector, the output will be incorrect and the output parity will become 1. We also observe that if a fault occurs in any of the extra circuitry, the original output of the circuit will not be affected. However the parity line will go high which clearly indicates the presence of a fault in the circuit.

4.5.3 Future Directions

Extension of the approach to detect all the possibilities of single bit faults and multiple missing gate faults is an area of further research. Our proposed approach is based on NCT gates. With this approach, it is possible to design an online testable circuit for any reversible function. However, the proposed online testable approach will not work for a circuit that contains SF gates. Analysing the interaction between the NCT gate and the SF gate families in a circuit in order to propose an online testing approach regardless of the types of gate is another open area for future work.

Chapter 5

Fault Tolerance in Reversible Logic

This chapter focuses on the principle of fault tolerance in digital system and highlights different design issues, and techniques for building fault tolerant reversible logic circuits. Our proposed approach for designing truly fault tolerant reversible circuits is also presented in this chapter.

5.1 Principle of Fault Tolerance

Fault tolerance is a unique attribute of a digital system, which enables a system to continue with the intended operations even under the influence of faults. A fault in a digital system can occur for different reasons, such as flaws in system specification and implementation, defects of system components, or external environmental disturbance. The occurrence of a fault in a system often causes an error. An error is a deviation of the output from the system's desired behaviour. With the presence of an error, a system may fail to perform its designated tasks. This is referred to as a system failure [3, 26, 50]. The cause-and-effect relationship among fault, error and failure states that a fault is the cause for an error and a failure, in turn, is the effect of an error [26]. The concept of fault tolerance comes into play for recovering a system from a potential state of failure. The property of fault tolerance enables a system to continue to function correctly in the presence of errors. Fault tolerance is the survival attribute or mechanism which returns a system from a potential erroneous state to an error free state [3]. The area of fault tolerance is not a new field in digital computing. Fault tolerance has been used in areas where digital computers perform crucial tasks such as space programs, military applications, and medical diagnosis treatment and activities [26].

5.2 Techniques to Achieve Fault Tolerance

A fault tolerant system must be capable of generating the expected outcome even in the presence of errors, either by correcting errors or bypassing errors. In order to design a fault tolerant system, it is necessary to build in redundancy of some type, generally hardware redundancy, software redundancy and/or information redundancy [26]. However, in the reversible domain, simply adding redundancy in order to combat system failure is not the only factor to consider. The addition of redundancy to a reversible system must ensure that one-to-one and onto relationships between the inputs and the outputs persist. Therefore, achieving fault tolerance in reversible logic is a challenging task.

With the concept of redundancy one or more techniques must be required in order to design a fault tolerant system [26, 50].

- Fault detection: a process of identifying whether or not a fault has occurred by observing the output of a system component.
- Fault location or fault diagnosis: takes place after fault detection. Fault location is used to locate the place where a fault has occurred or locate the component which is responsible for the fault.
- Fault containment: a process of isolating a fault in order to prevent the effect of a fault to propagate to other parts of the system.
- Fault recovery: takes place in order to restore the integrity of the system. A recovery process generally involves a replacement or reconfiguration of a faulty component, or some technique to bypass the effect of a fault.
- Fault masking: a dynamic process to provide corrected outputs in the system. The process of masking contributes to a fault tolerant system by hiding the effects of faults from the final output of the system.

5.3 Testing vs. Tolerance

The principle and techniques of testing are discussed in Chapter 4. Although the concepts of fault tolerance and testing are often confused, they are not the same. For instance, works on testing such as [46, 49, 69] present approaches that allow the circuit to be tested for faults, often using parity preservation. However, this does not provide fault tolerance, as defined later in this chapter.

Many works in the literature that address fault tolerant circuit designs in reversible logic use the term fault tolerant when really they are referring to testing. Works that fall into this category include [18, 22, 23, 29, 44, 53, 63]. In these works, the authors propose approaches which can fall into a category of testing. For example, Parhami [53] highlights the fact that most arithmetic and other processing functions do not preserve the parity of the input data at the output end. If the parity of input data is maintained throughout the computation, no intermediate error checking mechanism is required to detect faults in a circuit. The paper identifies some parity preserving reversible logic gates and presents a fault detection method based on parity preserving reversible gates. For example, Fredkin gates and double Feynman gates are parity preserving gates. A reversible circuit which is composed of only these two gates can detect an occurrence of fault as described by Parhami [53]. The paper also presents a design for a parity preserving full adder reversible circuit. The author concludes that fault tolerance can be achieved without adding any extra design effort if a reversible circuit is built using only parity preserving logic gates. Based on this concept, Islam et al. [23] proposed an approach to design what they claimed to be fault tolerant reversible circuits. The authors proposed a 4×4 parity preserving reversible gate, or IG gate, as shown in Figure 5.1. They offer an implementation of a reversible full adder circuit with two IG gates and claim that their proposed design is fault tolerant, suggesting again that fault tolerance can be achieved without any extra design effort if a reversible circuit is built using parity preserving gates. Their proposed full adder circuit is presented in Figure 5.2.

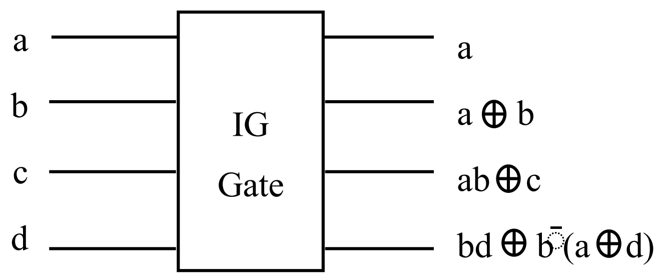


Figure 5.1: IG gate presented in [23].

Since an IG gate is a parity preserving gate, the proposed full adder circuit also preserves the parity. The sum and the output carry of the full adder in [23] are defined as $sum = a \oplus b \oplus c_{in}$ and $c_{out} = (a \oplus b)c_{in} \oplus ab$. For fault free operation, when $a = 1$, $b = 0$ and $c_{in} = 1$, the sum and the output carry are calculated as $sum = 1 \oplus 0 \oplus 1 = 0$ and $c_{out} = (1 \oplus 0) \cdot 1 \oplus 1 \cdot 0 = 1 \oplus 0 = 1$. Suppose a single bit fault occurs in the third line from the top of the circuit, as shown in Figure 5.2. The third output line of the first IG gate, which generates ab, is connected to the third input line of the second IG gate. As an effect of a single bit fault, as shown in Figure 5.2, the value of ab is inverted before contributing to the second IG gate. For example, when the full adder performs the addition of the three input bits, $a = 1$, $b = 0$ and $c_{in} = 1$; the third line from the top generates $ab = 0$. Due to the presence of the fault, the value of ab will be 1. This incorrect value of ab will be an input of the second IG gate, as shown in Figure 5.2. In this case the output carry is calculated as $c_{out} = (a \oplus b)c_{in} \oplus ab = (1 \oplus 0) \cdot 1 \oplus 1 = 1 \oplus 1 = 0$, which is incorrect. As we see, the proposed full adder circuit in [23] has no ability to generate corrected output in the presence of a fault. Thus, this full adder cannot be called a fault tolerant full adder.

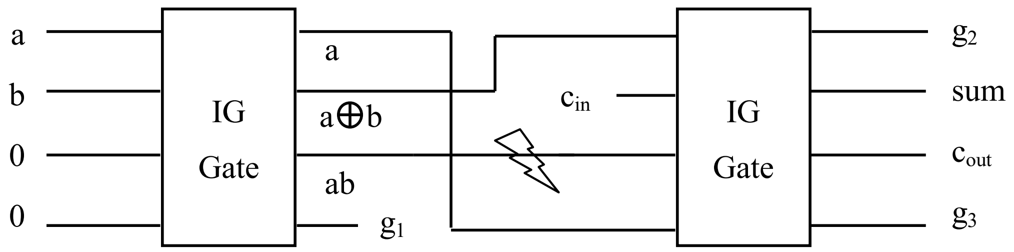


Figure 5.2: Parity Preserving Full Adder Circuit Composed of IG gates.

Similarly Haghparast el al. [18], the authors present a parity preserving Toffoli gate which is shown in Figure 5.3. According to this approach, a 3-bit Fredkin gate and a double Feynman gate are used to design a parity preserving Toffoli gate. Since a Feynman gate is already a parity preserving gate, the authors use two of their proposed parity preserving Toffoli gates and two double-Feynman gates to design what they claim is a fault tolerant full adder circuit. However, similar to the previous example, the full adder circuit presented in [18] has no ability to generated corrected output in the presence of faults in the circuit, and so is not fault tolerant.

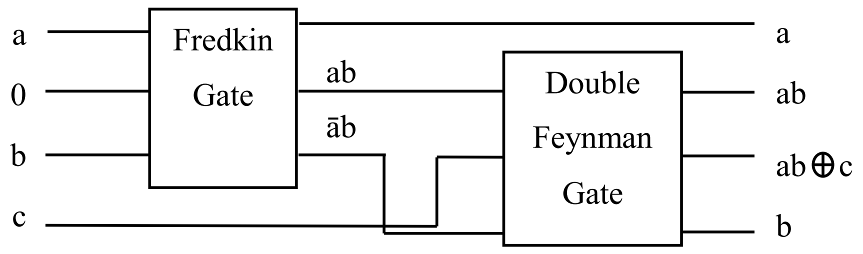


Figure 5.3: Parity Preserving Toffoli gate [18].

As discussed above, these works focus on preserving parity in reversible circuits. After ensuring parity preservation in their designs, the authors of these works claim that their proposed approaches are fault tolerant. Parity preservation can be used as a testing approach, which may indicate whether the output of a circuit is correct or incorrect. However, a fault tolerant circuit must have the capability to supply corrected (intended) values at the output even in the presence of faults in the circuit. A parity preserving circuit does not guarantee that the circuit is fault tolerant, since the use of parity preservation offers only error detection. Parity preservation does not ensure that the circuit generates corrected output in the event of error. Thus the designs based on parity preservation cannot be categorized as offering fault tolerance. Other works that indeed fall into the category of fault tolerance are described later in this chapter.

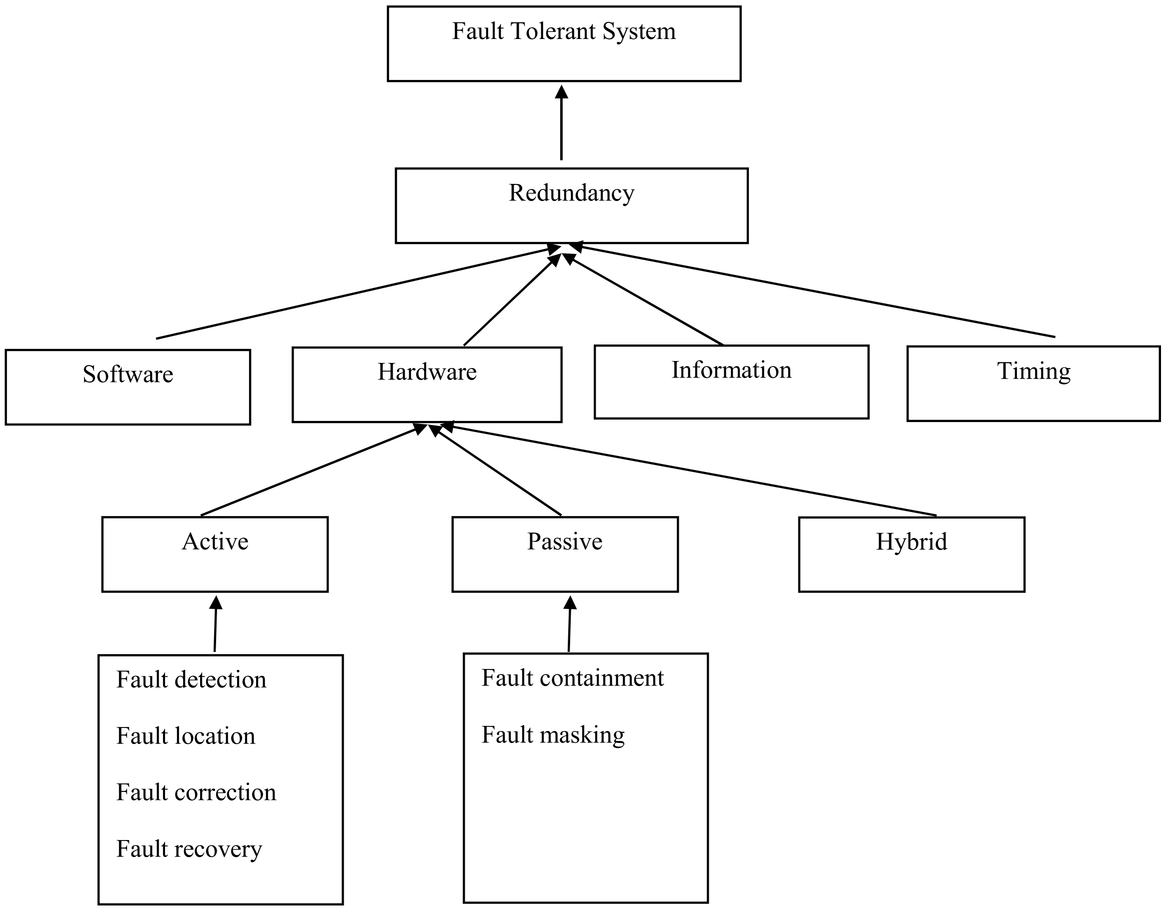


Figure 5.4: Achieving fault tolerance using the concept of redundancy.

5.4 Approaches to Achieve Fault Tolerance

Fault tolerance is generally achieved through the addition of redundancy. Redundancy to achieve fault tolerance can take several forms, generally hardware redundancy, software redundancy, information redundancy and/or timing redundancy [26]. This added redundancy comes at a cost. Redundancy is an additional resource in form of logical or physical components of a system, which helps to a system to achieve fault tolerance. For example, error detection and correction codes are some of the most common forms of information redundancy, and have been used extensively in information theory and data transmission [13]. Similarly additional hardware or software can be included in a system in order to achieve fault tolerance. The behavior of a system is an important factor to consider when choosing a particular type of redundancy to make the system fault tolerant. For example, the choice of redundancy that can be added to an information transfer system may not be the same as that of an information generation system. The input and the output of an information transfer system are the same (e.g telephone system), whereas the output of an information generation system may not always be the same as the input (e.g calculator). As a form of information redundancy, a code is generated by performing some calculation on the data on the sender side, and the generated code is included with the data to be sent (e.g. checksums or cyclic-redundancy codes) [13]. Upon receiving the data at the destination, the additional piece of information (i.e. the added code) allows the detection and correction of errors which may occur in transmission. Since data on both the sender side and the receiver side are the same, this technique of information redundancy is useful. However, this is not the case for a system that generates data instead of transferring data. Since the data on input and output are different, the types of error detection and correction techniques discussed above cannot be used in information generating systems.

Figure 5.4 shows a bottom-up diagram to achieve fault tolerance using the concept of redundancy. A common approach to achieve fault tolerance in an information generating system is to incorporate hardware redundancy by replicating one or more physical components of a system. The cost of this is initially high, but is ammortized over the lifetime of the system. Hardware redundancy can offer an active approach, a passive approach or a combination of both [26]. An active approach to fault tolerance works by detecting a fault, locating the fault and recovering the system through some form of reconfiguration. From the perspective of recovering a system from faults, an active approach can be considered an offline approach because the recovery process can only take place after the faulty system is taken out of operation. Fault tolerance can also be achieved using a passive approach that does not require detecting or reconfiguring, but rather masking the occurrence of faults. A passive approach is an online approach which can recover a system from the effect of faults in real time. That is, with a passive approach, it is not necessary to stop the system from performing its normal mode of operations. However, unlike an offline approach that actively removes faults, an online approach does not remove the faults from a faulty system, but instead prevents the effects of faults from affecting the original output of a system. In this dissertation we consider an online approach to achieve fault tolerance. Figure 5.4 shows the concept of hardware redundancy and the factors to consider in order to achieve fault tolerance in a digital system.

5.5 Fault Tolerance in Reversible Circuits

In our work we focus on fault masking to achieve fault tolerance in reversible logic. The reasons for choosing fault masking as a tool for designing a fault tolerant reversible logic circuit is the simplicity and the dynamic behaviour of fault masking techniques. Fault masking allows a system to generate the correct output in real time. As a passive approach, a fault masking technique does not need to reconfigure or replace any physical component of a system. A system which is designed using a fault masking technique hides faults from the output, i.e. a fault cannot affect the final outcome of a system. Therefore, a fault may affect the system locally, but cannot affect the global performance of the system.

Triple modular redundancy (TMR) has been used as one of the most common forms of passive hardware redundancy to design a fault tolerant system [6, 26, 50, 70]. The basic idea of TMR is to use three exact copies of a module (system), and a majority voter is used to combine the outputs from the triple modules. The majority voter is designed in such a way that it always generates the bit which appears on majority of the input lines. Therefore, if one of the modules generates an incorrect output, the majority voter will still generate the correct output by masking the output of the faulty module. The basic concept of TMR in traditional computing is depicted in Figure 5.5. Sometimes more than three modules are used to form *n*-modular redundancy, where *n* replicas of a module are connected to a majority voter.

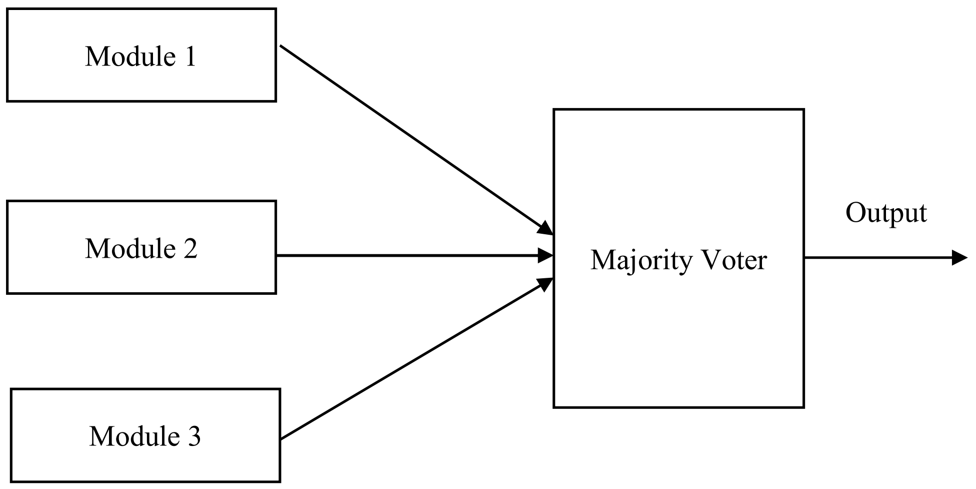


Figure 5.5: Three identical copies of a module using TMR in traditional computing.

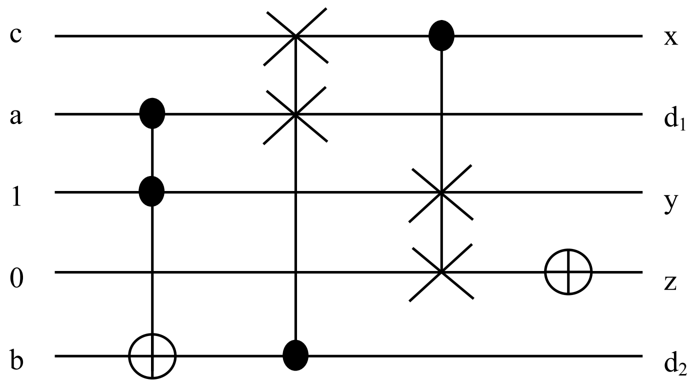


Figure 5.6: A minimum triplicated voter circuit proposed in [72].

5.5.1 Existing Reversible Majority Voter Circuits

There are only a few attempts in the literature to design majority voter circuits in reversible logic. In [72], the authors proposed two designs based on triple modular redundancy. In their proposed design, a voter circuit takes three inputs from the output of three copied circuits and generates three data outputs. In order to maintain reversibility, the voter used in their design is a 5-bit circuit with two garbage lines and two constant input lines. Their proposed voter circuit is presented in Figure 5.6. In this figure, 'x', 'y' and 'z' are the three data outputs. The design goal is to produce all 0 or all 1 on the three data outputs, thus masking any faulty output. However, our analysis suggests that this circuit does not generate the intended output. For example, when the input of this circuit is $abc = 010$, the output is $xyz = 011$, $d_1d_2 = 01$. However, according to their approach the correct data output for this particular input should be $xyz = 000$, as 0 is the majority bit in data inputs. This approach also needs additional logic to check whether all 3 outputs are 0s (or 1s).

A simplified majority voter circuit, as shown in Figure 5.7, is presented in [6]. The authors describe a reversible fault tolerant multiplexing scheme using a 3-bit repetition code. The voter consists of two CNOT gates and one Toffoli gate, and the majority output value is taken from the line labeled 'a_o'. For example, consider the case when the input bits are $abc = 011$. Since the values of both control points on line 'a' are 0, the two CNOT gates become inactive. However, the Toffoli gate is active, since $bc = 11$. Thus, the Toffoli gate inverts the value at line 'a' and sends 1 to output 'a_o', since the value of 'a' is 0. In this way the circuit generates the majority bit on line 'a_o'. The other two outputs, 'b_o' and 'c_o', are garbage outputs and can be ignored. The majority voter has a GC of 3 and a QC of $1+1+5=7$. The majority voter proposed in [6] can be used in designing fault tolerant reversible circuits by following the principle of triple modular redundancy.

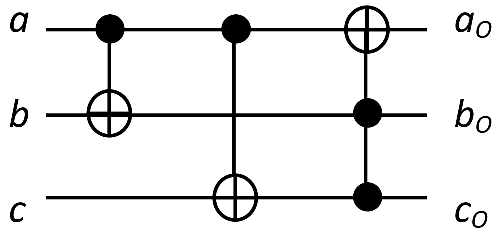


Figure 5.7: A reversible majority voter as proposed in [6].

5.5.2 A Proposed Reversible Majority Voter

In this section we present a simplified and cost effective approach for designing a majority voter in reversible logic [48]. To introduce our approach, a reversible three input majority voter is shown in Figure 5.8. The behaviour of this circuit is characterized by the truth table in Table 5.1.

The output of interest is 'a_o', which always gives the value appearing on the majority

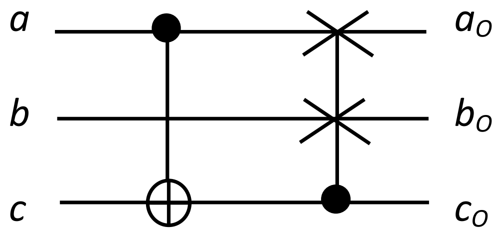


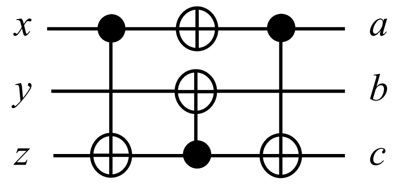
Figure 5.8: A three input reversible majority voter.

Table 5.1: Truth table for the reversible voter circuit.

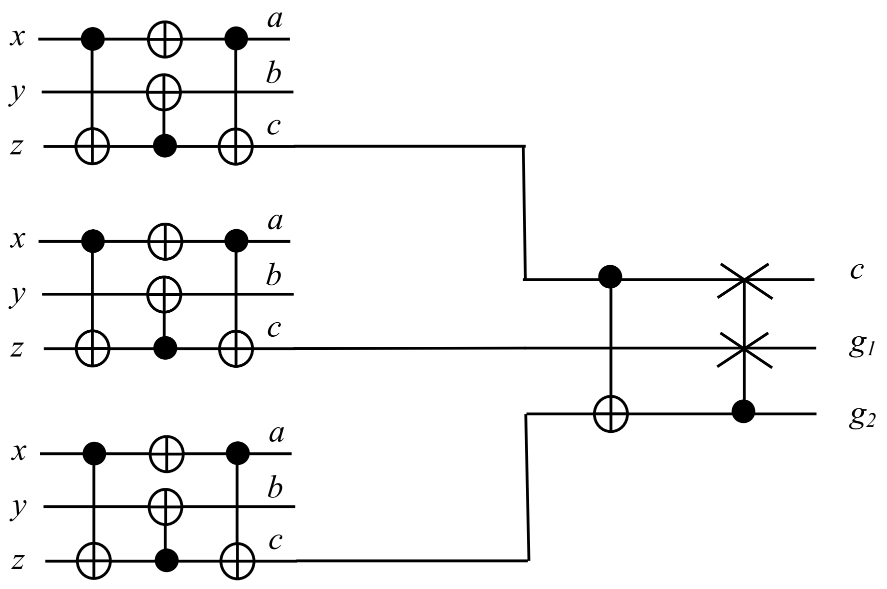
	after 1st gate	after 2nd gate
abc	$a'b'c'$	$a_ob_oc_o$
000	000	**0**00
001	001	**0**01
010	010	**0**10
011	011	**1**01
100	101	**0**11
101	100	**1**00
110	111	**1**11
111	110	**1**10

of the input lines. The other two lines are not used, and thus are considered garbage. The QC of this circuit is very low: 1 for the CNOT gate and 5 for the Fredkin gate, for a total of 6, which offers a small improvement as compared to the approach presented in [6]. As we see from Figure 5.8, the control of the Fredkin gate is $(a \oplus c)$. When $a = c$ then $(a \oplus c) = 0$, and the Fredkin gate becomes inactive and does not interchange the values of 'a' and 'b' at the output. In this case 'a' is the value used as the majority output. However, when $a \neq c$ then $(a \oplus c) = 1$, and the Fredkin gate becomes active and swaps the value of 'a' and 'b'. In this case the value of 'b' will be the majority output.

Our proposed majority voter circuit can be used to achieve fault tolerance in any reversible circuit. For example, consider a 3×3 reversible circuit shown in Figure 5.9(a). Suppose the output line 'c' is the output of interest for this circuit and the other two outputs are garbage. Figure 5.9(b) shows a fault tolerant design for the circuit from Figure 5.9(a). As we see from Figure 5.9(b), three copies of the circuit are used, since we are using the



(a) A reversible circuit.



(b) Fault tolerant version of the circuit presented in Figure 5.9(a).

Figure 5.9: Achieving fault tolerance using TMR.

principle of TMR. The three output lines from the three copied modules are connected to our proposed majority voter. If any one of the three copied modules becomes faulty, the majority voter masks the fault and sends the correct output to 'c'. The other two output lines 'g_1' and 'g_2' are the garbage output lines of the majority voter.

5.5.3 Fault Tolerant Full Adder Design

A fault tolerant full adder design based on our proposed majority voter is shown in Figure 5.10. Since there are two outputs (sum and carry) of interest in a full adder, two majority voters are required to ensure that faults in either output can be masked. As we see from Figure 5.10, there are three carry lines that are connected to the voter on top, while the bottom voter is connected to the three sum lines. This design can generate correct output

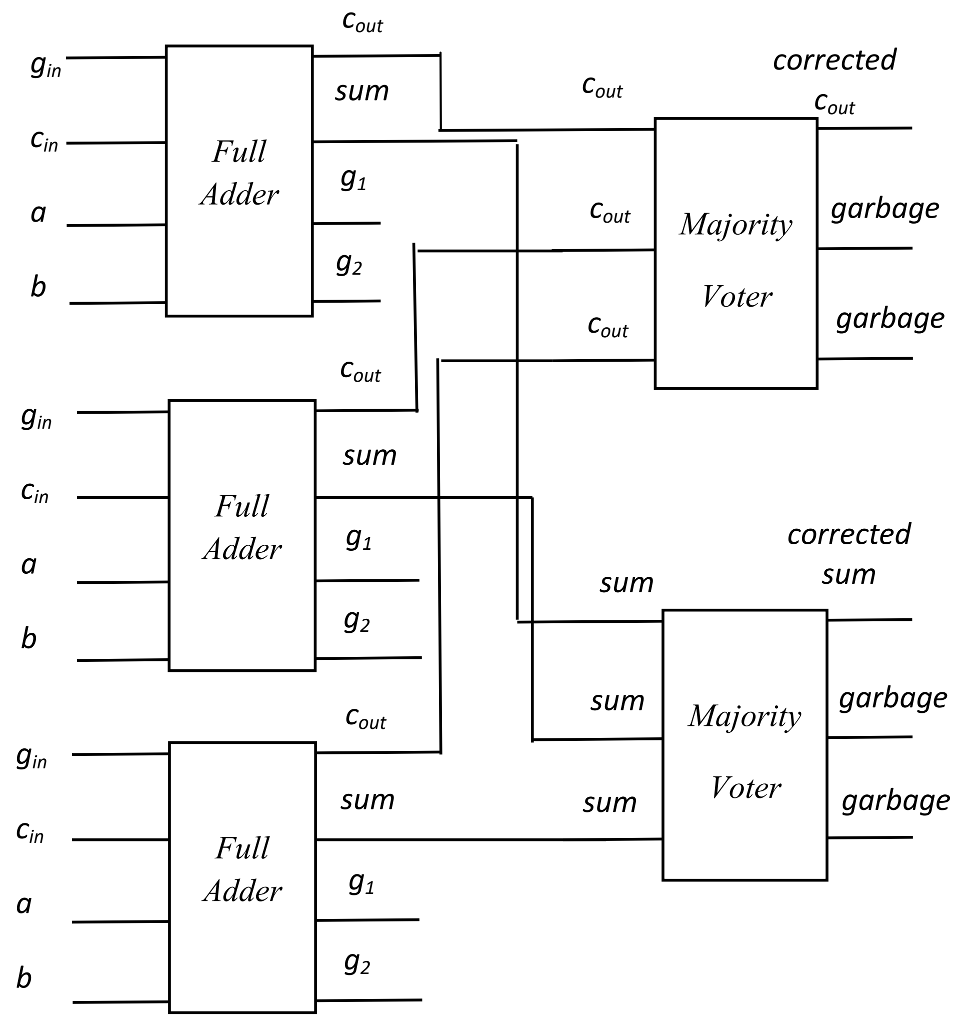


Figure 5.10: A fault tolerant reversible full-adder circuit design.

in the presence of faults from any of the proposed fault models [56, 59, 73] as long as the fault affects, at most, one of the triplicated full adders. Similarly this design can be applied to any type of circuit, albeit with a significant amount of hardware overhead.

5.6 Extension of Reversible Majority Voter

Although an odd value of n is desirable for a n-bit voter, a voter with an even number of inputs can sometimes be required in different applications. In this section, we present designs for both odd and even bit majority voters. We show that an even bit majority voter circuit can be used in designing an odd bit majority voter. First we consider the design of a n-bit majority voter, where n is even. When n is even, there are inputs for which there is no majority value; that is, the number of lines/input bits carrying 0 and the number of lines carrying 1 may be equal. In such cases we can consider both values to be the majority, or neither to be the majority, depending on the requirements of a particular application. One might refer to this as a "tie". For an instance, when we have a 4-bit input of $(0,1,1,0)$, the majority voter circuit could send either a 1 or 0 to the final output. For our design, we consider this to be a don't care condition.

Figure 5.11 shows a design of a 4-bit reversible majority voter consisting of three CNOT gates and one 4-bit Toffoli gate. The output that reflects the majority is labeled 'a_o'. The other three outputs are non-functional outputs. For example, consider an input vector $(a,b,c,d) = (1,0,0,0)$. Since $a = 1$, three CNOT gates are active and invert their corresponding target input bits. So the three bits (b,c,d) are transformed from $(0,0,0)$ to $(1,1,1)$. At this point the Toffoli gate becomes active and inverts the target bit $a = 1$ to 0. In this way the majority bit 0 is sent to the output line, 'a_o'. The GC and QC of this 4-bit majority voter is 4 and 16, respectively. However, it is possible to reduce these costs by using the functionality of a 3-bit majority voter.

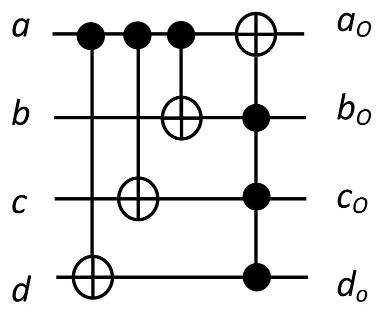


Figure 5.11: A 4-bit majority voter.

Lemma: For an even n, a $(n-1)$-bit majority voter is sufficient to determine the majority of n bits, assuming that "ties" are treated as don't cares.

Proof: Let x be a Boolean variable and $\{y_0,\ldots,y_{n-1}\}$ be the n Boolean inputs to a n-bit majority voter where n is even. Also, let l be the number of bits in $\{y_0,\ldots,y_{n-1}\}$ that take on the value of x. A sufficient condition for x to have the value of the majority of bits

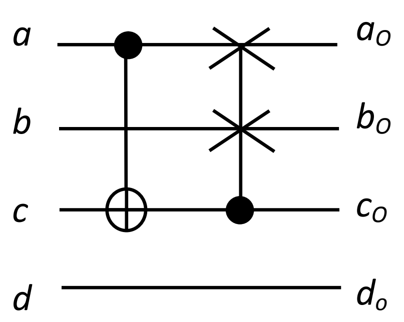


Figure 5.12: A 4-bit majority voter designed from a 3-bit voter.

in $\{y_0,\ldots,y_{n-1}\}$ is: $\lceil n/2 \rceil \leq l$ which is also a condition to determine the majority bit in a combination of bits $\{y_0,\ldots,y_{n-2}\}$ *i.e.* for the next lower (odd) value of n.

Figure 5.12 shows a design for a 4-bit majority voter. A 3-bit majority voter works on the input bits 'a', 'b' and 'c', and supplies the majority bit to the final output 'a_o'. The input bit 'd' is passed to 'd_o' unchanged. Table 5.2 shows the behaviour of the 4-bit majority voter from Figure 5.12. The design cost is the same as that of a 3-bit voter (GC of 2 and QC of 6). We also see that the majority among the first three bits serves as the final majority bit of all four bits, and so in fact, the fourth bit has no effect in determining the final outcome.

It is similarly possible to design a 6-bit voter by simply including a sixth line to a 5-bit majority voter. Figure 5.14 shows such a design. A 4-bit majority voter can be used in designing a 5-bit voter. The dashed box on right side of a 5-bit voter in Figure 5.13 shows a 4-bit majority voter. A constant input, p_{in}, is initialized to 0. A CNOT gate is connected from each of the inputs of the 4-bit voter to the parity line, p_{in}. The 4-bit voter manipulates the four bits $(b, c, d$ and $e)$ and sends the majority of these four bits to the point labeled, 'x'. The value of 'x' indicates three cases based on the number of appearance of 'x' in the four input bits: two, three or four times of appearance.

Case 1: When the value of 'x' appears exactly twice in $(b, c, d$ and $e)$, it indicates one of the don't care conditions and hence, the fifth bit at the line labeled, 'a', is the final majority bit. A group of CNOT gates, as shown in the dashed box in Figure 5.13, determines the parity of the four bits $(b, c, d$ and $e)$. In a don't care condition, the parity bit is $(x \oplus \bar{x} \oplus$

Table 5.2: Truth table of the 4-bit majority voter shown in Figure 5.12.

$abcd$	$a_ob_oc_od_o$	
0000	**0**000	
0001	**0**001	
0010	**0**010	
0011	**0**011	don't care
0100	**0**100	
0101	**0**101	don't care
0110	**1**010	don't care
0111	**1**011	
1000	**0**110	
1001	**0**111	don't care
1010	**1**000	don't care
1011	**1**001	
1100	**1**110	don't care
1101	**1**111	
1110	**1**100	
1111	**1**101	

$x \oplus \bar{x}) = 0$. In this case the 3-bit Fredkin gate, as shown in Figure 5.13, becomes inactive, so the bit at 'a' provides the majority bit at output 'a_0'. For example, when $(a,b,c,d,e) = (0,1,0,0,1)$, the 4-bit voter sends 1 to the point labeled 'x' in Figure 5.13. The CNOT gates block calculates the parity $(1 \oplus 0 \oplus 0 \oplus 1) = 0$. In this case both Fredkin gates become inactive, hence the Fredkin gates do not interchange the value of 'a' and 'x' at the output. The bit at 'a' goes to the output line, 'a_o', as the majority bit.

Case 2: When the value of 'x' appears exactly three times in the 4-bit inputs, the value of 'x' serves as the final majority bit of a 5-bit voter. In this case the parity line, $p_{in} = (x \oplus x \oplus x \oplus \bar{x}) = 1$. Thus, the 3-bit Fredkin gate becomes active and interchanges the value of 'a' and 'x'. The value of 'x' goes to the final output 'a_o'. Note that, the 5-bit Fredkin gate remains inactive, since 1 in the parity line deactivates a negative control of the Fredkin gate. For example, when $(a,b,c,d,e) = (1,1,0,0,0)$, the 4-bit voter supplies 0 to 'x'. In this case the output of the CNOT gates block is $(1 \oplus 0 \oplus 0 \oplus 0) = 1$, which activates the

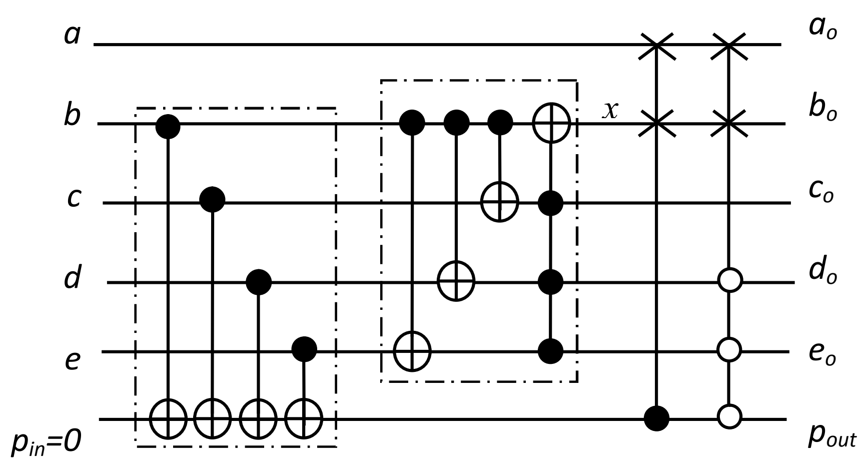


Figure 5.13: A 5-bit reversible majority voter.

3-bit positive control Fredkin gate. Thus the 3-bit Fredkin gate swaps the values of 'a' and 'x', and 0 goes to the final output line 'a_o' as the majority bit.

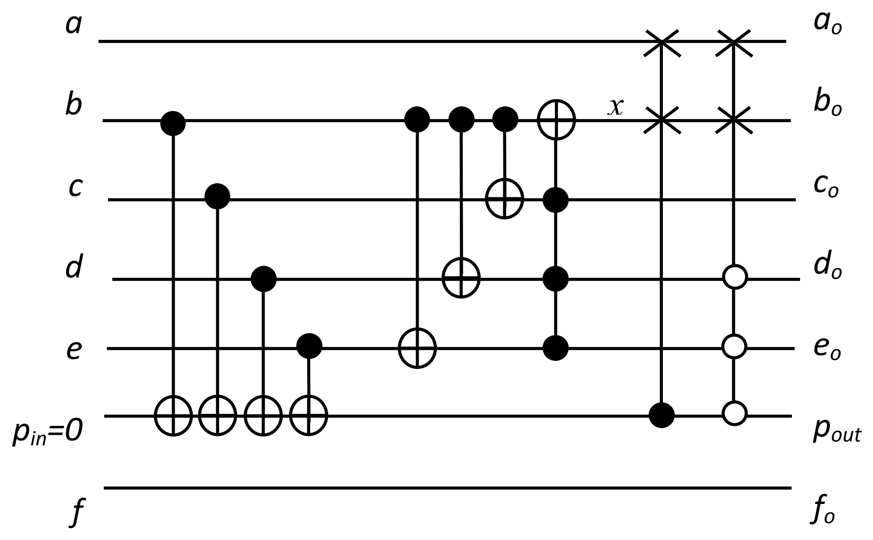


Figure 5.14: A 6-bit reversible majority voter.

Case 3: Here, we consider the case when the value of 'x' appears exactly four times in the 4-bit inputs (b,c,d,e). The last 5-bit negative control Fredkin gate is included in Figure 5.13 to supply the majority bit at the output when all four inputs of the 4-bit voter are the same and the fifth input line of a 5-bit voter has an opposite bit. For example, when $(a,b,c,d,e) = (0,1,1,1,1)$, the output of the 4-bit voter is $(1,0,0,0)$ and the parity line is

$(1 \oplus 1 \oplus 1 \oplus 1) = 0$. In this case the 3-bit Fredkin gate remains inactive. However the three 0s on the three lines labeled 'p_{in}', 'd' and 'e' activate the 5-bit negative control Fredkin gate. The 5-bit Fredkin gate swaps 'a' and 'x', thus the circuit sends 1 to the output line 'a_o'.

The GC and QC of a 5-bit voter is 10 and 54 respectively. We can use the 5-bit voter to design a 6-bit voter by simply including the sixth line with the same design cost, as shown in Figure 5.14. In this way it is possible to extend a majority voter circuit by building on designs for smaller voters.

5.7 Chapter Summary

5.7.1 Contribution

This chapter primarily focuses on designing fault tolerant reversible circuits. The key contributions of this chapter are as follows [48]:

- Present a 3-bit reversible majority voter circuit.
- Provide designs for extending the voter from 3-bit to 6-bit.
- Describe methodology to design fault tolerant reversible circuits using the voter circuit proposed in this chapter.

5.7.2 Conclusion

This chapter presents a 3-bit reversible majority voter circuit. The purpose of this circuit is to identify the bit value which appears more than any other bit value on the three input bits, assuming the use of Boolean (binary) values. Our proposed design is simpler and of lower cost in terms of gate count and quantum cost than existing designs in the literature. We also provide the designs for extending the voter from 3-bit up to 6-bits. Moreover, we demonstrate the application of our voter in fault tolerant reversible circuit design. We provide an overview and analysis of existing works that term themselves to be fault tolerant, but which do not meet the required characteristics to be categorized as such. Lastly, we present a design of a fault tolerant reversible full adder circuit. The proposed design can be used to make any reversible circuit fault tolerant. The proposed majority voter can be used to generate a corrected output in the presence of any type of fault as long as the fault affects a minority of the n input lines to the voter.

5.7.3 Future Directions

One of the crucial threats in designing a fault tolerant digital system using n-modular redundancy is the single point failure. In a n-modular redundant system, all the replicated modules are connected to a majority voter circuit. The majority voter bypasses the faults that may occur in the modules and generates the correct output. However, if the majority voter becomes faulty, the generated output may be incorrect. Therefore, failure of a majority voter leads to the failure of an entire system. For example, if one of the majority voters in our proposed fault tolerant full adder (Figure 5.10) fails, the full adder circuit fails to generate the correct sum or carry. Thus, making the majority voter robust is one of the main areas of further study. In addition, future work includes developing techniques for fault location as well as fault correction in reversible circuits.

Chapter 6

Conclusion

6.1 Contribution and Conclusion

This thesis contributes to three major areas of reversible computing: synthesis, testing and tolerance. Most existing reversible synthesis approaches focus on using NCT gates or combining NCT and SF gates to realize functions. In this thesis we have proposed approaches to realize reversible functions using only SF gates. A part of this work has been published in [47]. We have proposed a transformation based synthesis which can be used to realize conservative functions using SF gates. We have generated all possible 3-bit and 4-bit conservative reversible functions and realized these functions using both SF gates and NCT gates. Experimental results suggest that realization of reversible conservative functions using SF gates is more efficient than NCT gates in terms of GC and QC. For realizing 3-bit conservative functions, the percentage of reduction in GC and QC on average are 62% and 29% respectively. In some cases the percentage of reduction in GC and QC can be achieved up to 67%. We have also compared the minimal circuits generated using exact synthesis with our proposed SF based synthesis for implementing 3-bit conservative functions. Experimental results show that the percentage of reduction in GC and QC on average are 54% and 8% respectively using SF based synthesis over exact synthesis. For 4-bit conservative functions, the average percentages of reduction of GC and QC are 61% and 35% respectively using the SF based approach as compared to the NCT based transformation approach.

Since reversible circuits generated from most synthesis approaches may not be optimal, post synthesis optimization approaches can be used in order to improve the GC and QC of these circuits. This thesis introduces 10 templates based on template matching and rule based simplification approaches for optimizing SF based reversible circuits. We have considered templates for both positive and negative control Fredkin gates. We have identified the fact that some rules proposed for NCT gates can not be applied for SF gates, e.g. the moving rule. We have modified the moving rule in order to apply this rule for optimizing SF based reversible circuits. The proposed templates have been tested to optimize SF gate based benchmark circuits found in RevLib [71] and up to 9% reduction in QC has been achieved. We have also applied the proposed templates on 500 randomly generated SF gate based circuits, and we have achieved an average improvement of 16% in QC.

In the area of testing, we have described several reversible fault models and analysed the conditions to detect different types of faults in reversible circuits. We have investigated the existing reversible fault testing approaches and discussed the limitations of these approaches. The limitations of existing online fault testing approaches include (i) testing approaches that rely only on checking the input and the output parities of reversible circuits can not detect certain faults, (ii) testing approaches that have not considered reversible fault models, and (iii) testing approaches that have not considered the occurrences of faults in additional circuitry. In this thesis we have presented an online testing approach for reversible circuits, and this approach has been published in [46]. With this approach, it is possible to make any NCT based reversible circuit online testable. According to our approach, for a reversible circuit with L lines and N gates it is necessary to include a parity line, $2L$ CNOT gates and N additional duplicate gates to make the circuit online testable. The overhead for some selected benchmark circuits [71] based on our proposed approach has been calculated. Our approach results in a lower percentage of overhead for a circuit consisting of a large number of gates. We have discussed the mechanism of fault detection with our proposed online testable circuits by considering three reversible fault models. The reversible fault models considered include single bit fault model, missing and repeated gate fault model, and crosspoint fault model. The proposed online testing approach has been compared with other existing reversible online testing approaches from the perspective of gate count and quantum cost. Experimental results show that our proposed approach performs better than other existing reversible testing approaches. In addition, our approach can also detect a fault even if the fault occurs in the additional circuitry, unlike other approaches in the literature.

Lastly our research also makes contributions to the field of fault tolerant reversible circuits design. Our investigation in the area of fault tolerance in reversible logic reveals that most existing works that use the term fault tolerant are actually referring to fault testing. We have described the requirements and the strategies to achieve fault tolerance in reversible circuits. We have proposed a 3-bit majority voter circuit that can be used to design fault tolerant reversible circuits. The majority voter circuit generates the bit value which appears more than any other bit value on the three input bits. We have presented a design of a fault tolerant full adder circuit using our proposed voter circuit. Our proposed voter is simple in design, and offers lower cost in terms of GC and QC than the existing majority voter in reversible logic. We have demonstrated how to make any reversible circuit fault tolerant using our proposed majority voter. In addition, we have proposed some designs to extend the majority voter. We have shown that an $n-1$-bit voter circuit can be reused to design a n-bit voter circuit. In addition, we have shown that when the number of inputs of a voter circuit is even, i.e. when n is even, a $n-1$-bit voter circuit can be used as a n-bit voter. A part of this work has been published in [48].

6.2 Future Works

The findings of this thesis open several doors in different areas of reversible logic for further research, such as:

1. A comparison between NCT and SF gate based transformation approaches to realize conservative functions has been presented in Chapter 3. The results show that realization of conservative functions using SF gates is more efficient than NCT gates. This

indicates the fact that there is a need to classify reversible functions. Thus an important area of further study includes the classification of reversible functions based on their properties. A synthesis approach can generate more efficient circuits if it is possible to know the class of the reversible functions in advance.

2. The current version of our transformation based synthesis considers the previous stages of transformation while selecting gates for the current stage. This synthesis approach selects one or more gates to perform desired transformation in a stage using greedy method. The selection of gates is very important in order to design an efficient circuit. The gate selection process can be improved further if the gate selection process considers both previous and next stages of transformation.

3. Since most synthesis algorithms generate circuits which may not be optimal, more focus is needed in the field of post-synthesis optimization approaches. Most existing templates and rules are specified for a particular gate family. For example, the moving rule that works for NCT gates do not work for SF gates and vice-versa. Identifying more templates and rules which combine both NCT and SF gates is another area for further research.

4. An efficient template matching algorithm can increase the possibility to match templates in a reversible circuit. Rahman et al. proposed an algorithm that considers the rank of a template before applying the template to a circuit [58]. Their approach assigns ranks to the templates based on the amount reduction of quantum cost offered by each template. The proposed algorithm searches for the templates that offer the best possible reduction in GC. The algorithm which we have used to apply our proposed SF based templates can be improved based on the approach proposed in [58].

5. Our proposed testing approach can not detect a particular case of MMGF which we discussed in Chapter 4. Moreover, our testing approach can not detect a single bit fault that occurs in the preamble block of an online testable circuit. Extension of

our proposed testing approach in order to overcome these shortcomings is an area of further research.

6. The current version of our proposed testing approach is designed for NCT based circuits. An online testing approach for SF based circuits which is similar to our approach is presented in [27]. Combining these two approaches in order to design a general online testing approach that can be applied to any reversible circuit regardless of the types of gates is an important area of further study.

7. The majority voter circuit presented in Chapter 5 can be used to design a fault tolerant reversible circuit. The proposed majority voter circuit is based on TMR. One of the problems of a system based on TMR is a single point failure. In order to make the majority voter robust by providing a guard against this potential single point failure is another area for further research. One potential approach is to add an online testing feature to the voter, so the output of the voter can indicate whether the voter circuit itself is faulty or fault free.

8. This thesis presents a passive approach to design fault tolerant reversible circuits. With this approach, a circuit will be able to generate correct output by bypassing the effect of faults. A passive approach does not locate or correct faults. However, an active approach includes the techniques to locate and correct faults. Including a fault location technique to our proposed voter circuit design is another area of future work. Thus the voter circuit will not only generate the correct output, but also the circuit output can provide information to locate the faults. In this case it is possible that the values of the garbage output lines of a voter circuit can be used to locate the faults.

Bibliography

[1] Nabila Abdessaied, Mathias Soeken, Robert Wille, and Rolf Drechsler. Exact template matching using Boolean satisfiability. In *Multiple-Valued Logic (ISMVL), 2013 IEEE 43rd International Symposium on*, pages 328–333. IEEE, 2013.

[2] Mona Arabzadeh, Mehdi Saeedi, and Morteza Saheb Zamani. Rule-based optimization of reversible circuits. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 849–854. IEEE Press, 2010.

[3] Algirdas Avizienis. Fault-tolerance: The survival attribute of digital systems. *Proceedings of the IEEE*, 66(10):1109–1125, 1978.

[4] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[5] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[6] P Oscar Boykin and Vwani P Roychowdhury. Reversible fault-tolerant logic. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 444–453. IEEE, 2005.

[7] Xueyun Cheng, Zhijin Guan, Wei Wang, and Lingling Zhu. A simplification algorithm for reversible logic network of positive/negative control gates. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2442–2446. IEEE, 2012.

[8] Kamalika Datta, Gaurav Rathi, Robert Wille, Indranil Sengupta, Hafizur Rahaman, and Rolf Drechsler. Exploiting negative control lines in the optimization of reversible circuits. In *International Conference on Reversible Computation*, pages 209–220. Springer, 2013.

[9] Kamalika Datta, Indranil Sengupta, and Hafizur Rahaman. A post-synthesis optimization technique for reversible circuits exploiting negative control lines. *IEEE Transactions on Computers*, 64(4):1208–1214, 2015.

[10] Alexis De Vos. *Reversible computing: fundamentals, quantum computing, and applications*. John Wiley & Sons, 2011.

[11] K Fazel, MA Thornton, and Jacqueline E Rice. ESOP-based Toffoli gate cascade generation. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 206–209. IEEE, 2007.

[12] Richard P Feynman. Quantum mechanical computers. *Foundations of physics*, 16(6):507–531, 1986.

[13] A Behrouz Forouzan. *Data communications & networking (sie)*. Tata McGraw-Hill Education, 2006.

[14] Michael P Frank. Approaching the physical limits of computing. In *Multiple-Valued Logic, 2005. Proceedings. 35th International Symposium on*, pages 168–185. IEEE, 2005.

[15] Michael P Frank. Introduction to reversible computing: motivation, progress, and challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

[16] Edward Fredkin and Tommaso Toffoli. *Conservative logic*. Springer, 2002.

[17] Daniel Große, Robert Wille, Gerhard W Dueck, and Rolf Drechsler. Exact multiple-control toffoli network synthesis with sat techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5):703–715, 2009.

[18] M. Haghparast and K. Navi. Design of a novel fault tolerant reversible full adder for nanotechnology based systems. *World Applied Sciences Journal*, 3(1):114–118, 2008.

[19] Majid Haghparast and Soghra Shoaei. Design of a new parity preserving reversible full adder. *Journal of Circuits, Systems and Computers*, 24(01):1550006, 2015.

[20] John P Hayes, Ilia Polian, and Bernd Becker. Testing for missing-gate faults in reversible circuits. In *Test Symposium, 2004. 13th Asian*, pages 100–105. IEEE, 2004.

[21] Tony Hey. Quantum computing: an introduction. *Computing & Control Engineering Journal*, 10(3):105–112, 1999.

[22] Md Saiful Islam, Md Mahbubur Rahman, Zerina Begum, and Mohd. Zulfiquar Hafiz. Fault tolerant reversible logic synthesis: Carry look-ahead and carry-skip adders. In *Proceedings of the International Conference on Advances in Computational Tools for Engineering Applications*, pages 296–401, 2009.

[23] Md Saiful Islam, Muhammad Mahbubur Rahman, Zerina Begum, Mohd Zulfiquar Hafiz, and Abdullah Al Mahmud. Synthesis of fault tolerant reversible logic circuits. In *Testing and Diagnosis, 2009. ICTD 2009. IEEE Circuits and Systems International Conference on*, pages 1–4. IEEE, 2009.

[24] Kazuo Iwama, Yahiko Kambayashi, and Shigeru Yamashita. Transformation rules for designing cnot-based quantum circuits. In *Proceedings of the 39th annual Design Automation Conference*, pages 419–424. ACM, 2002.

[25] Niraj K Jha and Sandeep Gupta. *Testing of digital systems*. Cambridge University Press, 2003.

[26] Barry W Johnson. *Design and analysis of fault-tolerant digital systems*, volume 6. Addison-Wesley Reading, MA, 1989.

[27] Mozammel HA Khan and Jacqueline E Rice. Improved synthesis of reversible sequential circuits. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pages 2302–2305. IEEE, 2016.

[28] Dipak K Kole, Hafizur Rahaman, Debesh K Das, and Bhargab B Bhattacharya. Synthesis of online testable reversible circuit. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 277–280. IEEE, 2010.

[29] P Kiran Kumar, P Prasad Rao, and Kakarla Hari Kishore. Optimal design of reversible parity preserving new full adder/full subtractor. In *Intelligent Systems and Control (ISCO), 2017 11th International Conference on*, pages 368–373. IEEE, 2017.

[30] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM journal of research and development*, 5(3):183–191, 1961.

[31] Sk Noor Mahammad and Kamakoti Veezhinathan. Constructing online testable circuits using reversible logic. *IEEE transactions on instrumentation and measurement*, 59(1):101–109, 2010.

[32] Dmitri Maslov. Reversible logic synthesis benchmarks page. 2011. http://webhome.cs.uvic.ca/ dmaslov/.

[33] Dmitri Maslov, G Dueck, and Nathan Scott. Reversible logic synthesis benchmarks page. *Online: http://www. cs. uvic. ca/˜ dmaslov*, 2005.

[34] Dmitri Maslov and Gerhard W Dueck. Improved quantum cost for n-bit Toffoli gates. *arXiv preprint quant-ph/0403053*, 2004.

[35] Dmitri Maslov and Gerhard W Dueck. Reversible cascades with minimal garbage. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1497–1509, 2004.

[36] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. Fredkin/Toffoli templates for reversible logic synthesis. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 256. IEEE Computer Society, 2003.

[37] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. Simplification of Toffoli networks via templates. In *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pages 53–58. IEEE, 2003.

[38] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. Toffoli network synthesis with templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817, 2005.

[39] Dmitri Maslov, Christina Young, D Michael Miller, and Gerhard W Dueck. Quantum circuit simplification using templates. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1208–1213. IEEE, 2005.

[40] Ralph C Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4(1):21, 1993.

[41] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th annual Design Automation Conference*, pages 318–323. ACM, 2003.

[42] D Michael Miller, Robert Wille, and Gerhard W Dueck. Synthesizing reversible circuits for irreversible functions. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD'09. 12th Euromicro Conference on*, pages 749–756. IEEE, 2009.

[43] D Michael Miller, Robert Wille, and Zahra Sasanian. Elementary quantum gate realizations for multiple-control Toffoli gates. In *Multiple-Valued Logic (ISMVL), 2011 41st IEEE International Symposium on*, pages 288–293. IEEE, 2011.

[44] Sajib Kumar Mitra and Ahsan Raja Chowdhury. Minimum cost fault tolerant adder circuits in reversible logic synthesis. In *VLSI Design (VLSID), 2012 25th International Conference on*, pages 334–339. IEEE, 2012.

[45] Majid Mohammadi and Mohammad Eshghi. On figures of merit in reversible and quantum logic designs. *Quantum Information Processing*, 8(4):297–318, 2009.

[46] Md Asif Nashiry, Gite Gaurav Bhaskar, and Jacqueline E Rice. Online testing for three fault models in reversible circuits. In *Multiple-Valued Logic (ISMVL), 2015 IEEE International Symposium on*, pages 8–13. IEEE, 2015.

[47] Md Asif Nashiry, Mozammel HA Khan, and Jacqueline E Rice. Controlled and uncontrolled swap gates in reversible logic synthesis. In *International Conference on Reversible Computation*, pages 141–147. Springer, 2017.

[48] Md Asif Nashiry and Jacqueline E Rice. A reversible majority voter circuit and applications. In *Communications, Computers and Signal Processing (PACRIM), 2017 IEEE Pacific Rim Conference on*. IEEE, 2017.

[49] Noor M Nayeem and Jacqueline E Rice. Online fault detection in reversible logic. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on*, pages 426–434. IEEE, 2011.

[50] Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, 1990.

[51] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[52] W David Pan and Mahesh Nalasani. Reversible logic. *IEEE Potentials*, 24(1):38–41, 2005.

[53] Behrooz Parhami. Fault-tolerant reversible circuits. In *Proceedings of the Fortieth Asilomar Conference on Signals, Systems and Computers (ACSSC)*, pages 1726–1729, Oct. 29–Nov. 1 2006.

[54] Ketan N Patel, John P Hayes, and Igor L Markov. Fault testing for reversible circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(8):1220–1230, 2004.

[55] Goutam Paul, Anupam Chattopadhyay, and Chander Chandak. Designing parity preserving reversible circuits. In *International Conference on Reversible Computation*, pages 77–89. Springer, 2017.

[56] Ilia Polian, Thomas Fiehn, Bernd Becker, and John P Hayes. A family of logical fault models for reversible circuits. In *Test Symposium, 2005. Proceedings. 14th Asian*, pages 422–427. IEEE, 2005.

[57] Nils Przigoda, Gerhard Dueck, Robert Wille, and Rolf Drechsler. Fault detection in parity preserving reversible circuits. In *Multiple-Valued Logic (ISMVL), 2016 IEEE 46th International Symposium on*, pages 44–49. IEEE, 2016.

[58] Md Zamilur Rahman and Jacqueline E Rice. Templates for positive and negative control Toffoli networks. In *International Conference on Reversible Computation*, pages 125–136. Springer, 2014.

[59] Jacqueline E Rice. An overview of fault models and testing approaches for reversible logic. In *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pages 125–130. IEEE, 2013.

[60] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits a survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.

[61] Rakshith Saligram, Shrihari Shridhar Hegde, Shashidhar A Kulkarni, HR Bhagyalakshmi, and MK Venkatesha. Design of parity preserving logic based fault tolerant reversible arithmetic logic unit. *arXiv preprint arXiv:1307.3690*, 2013.

[62] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[63] Bibhash Sen, Siddhant Ganeriwal, and Biplab K Sikdar. Reversible logic-based fault-tolerant nanocircuits in QCA. *ISRN Electronics*, 2013, 2013.

[64] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Reversible logic circuit synthesis. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 353–360. ACM, 2002.

[65] Soghra Shoaei and Majid Haghparast. Novel designs of nanometric parity preserving reversible compressor. *Quantum information processing*, 13(8):1701–1714, 2014.

[66] John A Smolin and David P DiVincenzo. Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate. *Physical Review A*, 53(4):2855, 1996.

[67] Mathias Soeken, Stefan Frehse, Robert Wille, and Rolf Drechsler. Revkit: A toolkit for reversible circuit design. *Multiple-Valued Logic and Soft Computing*, 18(1):55–65, 2012.

[68] Tommaso Toffoli. *Reversible computing*. Springer, 1980.

[69] Dilip P Vasudevan, Parag K Lala, Jia Di, and James Patrick Parkerson. Reversible-logic design with online testability. *IEEE transactions on instrumentation and measurement*, 55(2):406–414, 2006.

[70] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[71] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[72] Masoud Zamani, Navid Farazmand, and Mehdi B. Tahoori. Fault masking and diagnosis in reversible circuits. In *Proceedings of the 16th IEEE European Test Symposium (ETS)*, pages 69–74, May 2011.

[73] Jing Zhong and Jon C Muzio. Analyzing fault models for reversible logic circuits. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2422–2427. IEEE, 2006.

Appendix A

Experiment Result

Table A.1: Results after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random130	3	55	165	15	45	72.73	72.73
random252	4	10	36	6	24	40	33.33
random373	5	55	369	51	357	7.27	3.25
random494	5	100	691	90	661	10	4.34
random253	5	55	439	50	418	9.09	4.78
random374	6	100	1278	96	1266	4	0.94
random495	6	10	183	10	183	0	0
random254	6	100	1475	92	1447	8	1.9
random375	3	10	30	6	18	40	40
random496	7	55	1098	55	1098	0	0
random255	4	10	48	7	35	30	27.08
random376	7	55	958	55	958	0	0
random497	4	100	394	88	354	12	10.15
random134	3	100	300	48	144	52	52
random490	5	55	401	53	391	3.64	2.49
random133	6	55	705	55	705	0	0
random370	6	55	815	55	815	0	0
random491	7	100	1910	100	1910	0	0
random132	3	10	30	4	12	60	60
random250	6	55	623	53	617	3.64	0.96
random371	5	100	726	98	720	2	0.83
random492	6	10	89	10	89	0	0
random131	5	100	741	95	724	5	2.29
random251	3	100	300	48	144	52	52
random372	4	10	42	10	42	0	0
random493	3	55	165	21	63	61.82	61.82
random78	5	10	56	10	56	0	0
random79	4	55	221	37	159	32.73	28.05
random74	7	100	2326	99	2319	1	0.3
random75	5	10	71	10	71	0	0
random76	3	55	165	25	75	54.55	54.55
random77	6	100	1350	97	1337	3	0.96
random81	7	10	115	8	107	20	6.96
random82	4	55	225	49	207	10.91	8
random83	6	100	1336	100	1336	0	0
random84	7	10	257	10	257	0	0
random80	4	100	406	87	359	13	11.58
random138	7	10	217	10	217	0	0
random137	5	100	756	95	735	5	2.78
random136	4	55	225	41	171	25.45	24
random135	6	10	91	10	91	0	0
random256	3	55	165	21	63	61.82	61.82

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random377	7	100	1979	96	1963	4	0.81
random498	5	10	66	9	61	10	7.58
random257	6	100	1164	93	1141	7	1.98
random378	7	10	356	10	356	0	0
random499	7	55	931	53	925	3.64	0.64
random258	3	10	30	2	6	80	80
random379	5	55	367	53	359	3.64	2.18
random139	7	55	1192	51	1180	7.27	1.01
random259	5	55	386	51	352	7.27	8.81
random241	4	55	205	46	170	16.36	17.07
random362	5	100	685	92	661	8	3.5
random483	6	10	111	10	111	0	0
random242	5	100	787	96	753	4	4.32
random363	3	10	30	4	12	60	60
random484	3	55	165	9	27	83.64	83.64
random243	4	10	40	8	34	20	15
random364	6	55	723	50	706	9.09	2.35
random485	5	100	695	95	680	5	2.16
random244	5	55	433	51	401	7.27	7.39
random365	5	100	673	97	664	3	1.34
random486	6	10	192	10	192	0	0
random123	5	10	84	10	84	0	0
random122	7	100	2447	99	2444	1	0.12
random480	7	10	195	10	195	0	0
random121	3	55	165	17	51	69.09	69.09
random360	3	10	30	2	6	80	80
random481	6	55	754	55	754	0	0
random120	7	10	161	10	161	0	0
random240	3	10	30	8	24	20	20
random361	7	55	1203	53	1197	3.64	0.5
random482	3	100	300	56	168	44	44
random67	7	55	1283	55	1283	0	0
random68	4	100	390	83	335	17	14.1
random69	5	10	82	9	79	10	3.66
random63	6	10	156	10	156	0	0
random64	5	55	373	54	366	1.82	1.88
random65	3	100	300	26	78	74	74
random66	5	10	47	8	41	20	12.77
random70	5	55	394	52	363	5.45	7.87
random71	6	100	1572	100	1572	0	0
random72	6	10	105	9	102	10	2.86
random73	7	55	1106	52	1097	5.45	0.81
random127	3	55	165	21	63	61.82	61.82
random249	6	10	100	10	100	0	0
random126	3	10	30	4	12	60	60
random125	7	100	2211	100	2211	0	0
random124	4	55	225	46	190	16.36	15.56
random245	4	100	386	82	324	18	16.06
random366	4	10	38	7	29	30	23.68
random487	7	55	1346	55	1346	0	0
random246	5	10	51	9	48	10	5.88
random367	4	55	223	50	204	9.09	8.52
random488	3	100	300	34	102	66	66
random129	5	10	91	9	86	10	5.49
random247	3	55	165	27	81	50.91	50.91
random368	5	100	696	93	673	7	3.3
random489	6	10	127	10	127	0	0
random128	5	100	776	95	759	5	2.19

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random248	6	100	1412	98	1406	2	0.42
random369	7	10	208	10	208	0	0
random152	3	100	300	26	78	74	74
random274	5	55	346	54	339	1.82	2.02
random395	4	100	406	83	339	17	16.5
random151	3	55	165	17	51	69.09	69.09
random275	3	100	300	36	108	64	64
random396	5	10	78	10	78	0	0
random150	5	10	84	10	84	0	0
random276	7	10	302	10	302	0	0
random397	7	55	1110	55	1110	0	0
random277	5	55	437	52	426	5.45	2.52
random398	7	100	1933	99	1930	1	0.16
random156	7	10	227	10	227	0	0
random270	7	10	120	10	120	0	0
random391	7	55	1259	55	1259	0	0
random155	5	100	695	97	686	3	1.29
random271	5	55	461	52	430	5.45	6.72
random392	4	100	400	90	370	10	7.5
random154	6	55	665	55	665	0	0
random272	5	100	788	93	761	7	3.43
random393	4	10	42	8	36	20	14.29
random153	6	10	102	8	92	20	9.8
random273	4	10	36	8	30	20	16.67
random394	4	55	237	49	219	10.91	7.59
random56	7	100	2336	99	2333	1	0.13
random57	4	10	42	10	42	0	0
random58	4	55	219	47	195	14.55	10.96
random390	6	10	181	10	181	0	0
random59	3	100	300	42	126	58	58
random52	7	55	1162	53	1156	3.64	0.52
random53	6	100	1202	98	1160	2	3.49
random54	6	10	78	10	78	0	0
random55	5	55	393	49	373	10.91	5.09
random60	6	10	127	10	127	0	0
random61	3	55	165	17	51	69.09	69.09
random62	5	100	846	99	843	1	0.35
random159	3	10	30	2	6	80	80
random158	3	100	300	36	108	64	64
random157	6	55	757	55	757	0	0
random278	6	100	1375	95	1360	5	1.09
random399	5	10	58	10	58	0	0
random279	4	10	40	8	34	20	15
random141	3	10	30	6	18	40	40
random263	7	100	2085	98	2079	2	0.29
random384	4	10	38	8	28	20	26.32
random140	6	100	1457	98	1451	2	0.41
random264	4	10	44	8	34	20	22.73
random385	5	55	436	54	433	1.82	0.69
random265	5	55	303	50	284	9.09	6.27
random386	5	100	770	95	751	5	2.47
random266	5	100	652	93	631	7	3.22
random387	7	10	291	10	291	0	0
random145	5	55	406	54	401	1.82	1.23
random380	3	100	300	32	96	68	68
random49	4	55	215	39	167	29.09	22.33
random144	5	10	78	9	75	10	3.85
random260	7	100	2067	100	2067	0	0

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random381	6	10	172	10	172	0	0
random143	7	100	2102	98	2096	2	0.29
random261	4	10	36	5	21	50	41.67
random382	4	55	223	38	152	30.91	31.84
random142	5	55	392	50	377	9.09	3.83
random262	3	55	165	21	63	61.82	61.82
random383	5	100	711	95	694	5	2.39
random45	3	10	30	2	6	80	80
random46	6	55	587	52	578	5.45	1.53
random47	3	100	300	22	66	78	78
random48	5	10	69	10	69	0	0
random41	5	100	678	92	652	8	3.83
random42	4	10	44	9	41	10	6.82
random43	7	55	970	55	970	0	0
random44	7	100	2357	100	2357	0	0
random50	4	100	376	72	280	28	25.53
random51	6	10	143	10	143	0	0
random149	5	100	688	95	651	5	5.38
random148	3	55	165	17	51	69.09	69.09
random147	6	10	138	10	138	0	0
random146	4	100	384	85	339	15	11.72
random267	4	10	42	9	39	10	7.14
random388	5	55	334	51	318	7.27	4.79
random268	6	55	748	54	745	1.82	0.4
random389	5	100	708	96	696	4	1.69
random269	3	100	300	32	96	68	68
random450	3	10	30	4	12	60	60
random330	7	10	238	10	238	0	0
random451	6	55	827	54	824	1.82	0.36
random210	7	10	277	10	277	0	0
random331	6	55	722	55	722	0	0
random452	4	100	396	81	335	19	15.4
random211	5	55	498	53	468	3.64	6.02
random332	3	100	300	38	114	62	62
random453	3	10	30	4	12	60	60
random38	3	100	300	24	72	76	76
random39	4	10	38	10	38	0	0
random34	3	55	165	13	39	76.36	76.36
random35	3	100	300	28	84	72	72
random36	4	10	38	9	35	10	7.89
random37	6	55	772	55	772	0	0
random30	4	10	36	7	27	30	25
random31	3	55	165	29	87	47.27	47.27
random32	7	100	2070	99	2067	1	0.14
random33	7	10	250	7	195	30	22
random40	5	55	450	54	427	1.82	5.11
random216	3	10	30	8	24	20	20
random337	4	55	231	49	209	10.91	9.52
random458	4	100	394	77	313	23	20.56
random217	7	55	1004	53	998	3.64	0.6
random338	6	100	1295	96	1281	4	1.08
random459	7	10	168	10	168	0	0
random218	5	100	699	89	660	11	5.58
random339	7	10	141	10	141	0	0
random219	6	10	98	10	98	0	0
random212	7	100	1762	100	1762	0	0
random333	4	10	42	6	26	40	38.1
random454	5	55	348	49	326	10.91	6.32

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random213	6	10	165	10	165	0	0
random334	4	55	229	46	194	16.36	15.28
random455	5	100	741	94	719	6	2.97
random214	7	55	969	55	969	0	0
random335	4	100	414	81	337	19	18.6
random456	7	10	219	10	219	0	0
random215	7	100	1912	100	1912	0	0
random336	7	10	135	10	135	0	0
random457	7	55	981	55	981	0	0
random440	6	100	1494	97	1483	3	0.74
random320	3	100	300	40	120	60	60
random441	5	10	71	10	71	0	0
random200	5	100	797	91	730	9	8.41
random321	5	10	67	10	67	0	0
random442	7	55	922	55	922	0	0
random27	6	10	154	10	154	0	0
random28	3	55	165	5	15	90.91	90.91
random29	4	100	400	82	342	18	14.5
random23	3	100	300	32	96	68	68
random24	6	10	114	8	108	20	5.26
random25	3	55	165	23	69	58.18	58.18
random26	3	100	300	34	102	66	66
random20	7	100	2196	98	2190	2	0.27
random21	3	10	30	4	12	60	60
random22	7	55	1076	55	1076	0	0
random209	4	100	400	82	342	18	14.5
random205	4	55	211	42	164	23.64	22.27
random326	6	100	1329	99	1326	1	0.23
random447	4	10	42	8	36	20	14.29
random206	6	100	1252	99	1249	1	0.24
random327	4	10	34	9	31	10	8.82
random448	7	55	1251	55	1251	0	0
random207	3	10	30	4	12	60	60
random328	5	55	459	54	452	1.82	1.53
random449	5	100	735	89	678	11	7.76
random208	6	55	718	52	709	5.45	1.25
random329	3	100	300	44	132	56	56
random201	7	10	155	10	155	0	0
random322	7	55	931	55	931	0	0
random443	3	100	300	28	84	72	72
random202	3	55	165	13	39	76.36	76.36
random323	3	100	300	40	120	60	60
random444	5	10	58	10	58	0	0
random203	6	100	1354	99	1351	1	0.22
random324	5	10	98	8	92	20	6.12
random445	4	55	207	42	160	23.64	22.71
random204	7	10	271	10	271	0	0
random325	7	55	1384	55	1384	0	0
random446	3	100	300	32	96	68	68
random230	4	100	410	79	335	21	18.29
random351	5	10	78	10	78	0	0
random472	3	55	165	19	57	65.45	65.45
random231	5	10	69	8	59	20	14.49
random352	6	55	800	53	774	3.64	3.25
random473	6	100	1382	100	1382	0	0
random232	7	55	1301	55	1301	0	0
random353	6	100	1301	97	1290	3	0.85
random474	3	10	30	8	24	20	20

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random233	6	100	1286	100	1286	0	0
random354	6	10	100	9	97	10	3
random475	4	55	221	46	190	16.36	14.03
random112	4	55	229	42	178	23.64	22.27
random16	4	55	219	38	164	30.91	25.11
random111	6	10	168	10	168	0	0
random17	3	100	300	30	90	70	70
random110	7	100	2256	100	2256	0	0
random18	3	10	30	8	24	20	20
random470	7	100	2139	100	2139	0	0
random19	4	55	227	49	205	10.91	9.69
random350	6	100	1311	97	1300	3	0.84
random471	4	10	40	10	40	0	0
random12	4	10	42	10	42	0	0
random13	5	55	345	52	336	5.45	2.61
random14	4	100	408	79	337	21	17.4
random15	7	10	395	10	395	0	0
random10	6	55	738	53	732	3.64	0.81
random11	7	100	1677	96	1619	4	3.46
random116	3	100	300	32	96	68	68
random238	6	55	663	54	660	1.82	0.45
random359	7	100	1998	98	1988	2	0.5
random115	6	55	686	54	681	1.82	0.73
random239	6	100	1539	100	1539	0	0
random114	7	10	260	10	260	0	0
random113	4	100	414	80	338	20	18.36
random234	4	10	46	5	23	50	50
random355	4	55	215	41	169	25.45	21.4
random476	7	100	2196	100	2196	0	0
random119	5	100	704	93	675	7	4.12
random235	4	55	225	45	187	18.18	16.89
random356	6	100	1457	96	1439	4	1.24
random477	6	10	109	9	106	10	2.75
random118	4	55	219	42	168	23.64	23.29
random236	6	100	1357	99	1354	1	0.22
random357	7	10	272	10	272	0	0
random478	6	55	847	53	841	3.64	0.71
random117	4	10	34	10	34	0	0
random237	7	10	162	9	159	10	1.85
random358	6	55	673	55	673	0	0
random479	4	100	406	86	352	14	13.3
random340	7	55	1220	55	1220	0	0
random461	3	100	300	44	132	56	56
random220	5	55	379	53	373	3.64	1.58
random341	6	100	1337	97	1328	3	0.67
random462	7	10	180	10	180	0	0
random221	7	100	2315	97	2304	3	0.48
random342	7	10	219	8	209	20	4.57
random463	3	55	165	27	81	50.91	50.91
random222	6	10	138	10	138	0	0
random343	6	55	702	55	702	0	0
random464	6	100	1472	93	1447	7	1.7
random101	5	100	787	95	772	5	1.91
random100	7	55	1381	55	1381	0	0
random460	3	55	165	19	57	65.45	65.45
random105	3	10	30	2	6	80	80
random227	7	100	2191	100	2191	0	0
random348	4	10	38	9	35	10	7.89

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random469	5	55	388	55	388	0	0
random104	4	100	390	76	306	24	21.54
random228	6	10	118	9	115	10	2.54
random349	3	55	165	15	45	72.73	72.73
random103	7	55	1020	54	1017	1.82	0.29
random229	7	55	1099	55	1099	0	0
random102	4	10	44	6	28	40	36.36
random109	5	55	378	54	373	1.82	1.32
random223	7	55	1284	55	1284	0	0
random344	5	100	659	92	627	8	4.86
random465	7	10	221	10	221	0	0
random108	4	10	42	10	42	0	0
random224	4	100	400	85	351	15	12.25
random345	7	10	183	10	183	0	0
random466	6	55	780	55	780	0	0
random107	7	100	2152	100	2148	0	0.19
random225	3	10	30	4	12	60	60
random346	6	55	712	55	712	0	0
random467	3	100	300	24	72	76	76
random106	3	55	165	19	57	65.45	65.45
random226	3	55	165	29	87	47.27	47.27
random347	7	100	1652	97	1641	3	0.67
random468	4	10	44	10	44	0	0
random174	4	10	42	10	42	0	0
random296	3	100	300	38	114	62	62
random173	5	100	695	94	673	6	3.17
random297	4	10	42	8	36	20	14.29
random172	6	55	878	55	878	0	0
random298	6	55	715	54	710	1.82	0.7
random171	7	10	203	10	203	0	0
random299	6	100	1385	97	1376	3	0.65
random178	7	55	1093	55	1093	0	0
random292	4	55	209	44	168	20	19.62
random177	4	10	42	9	39	10	7.14
random293	5	100	773	95	754	5	2.46
random176	3	100	300	28	84	72	72
random2	6	100	1444	98	1438	2	0.42
random294	5	10	111	10	111	0	0
random1	7	55	1268	55	1268	0	0
random175	3	55	165	25	75	54.55	54.55
random295	3	55	165	27	81	50.91	50.91
random290	7	100	2162	100	2162	0	0
random291	7	10	134	8	128	20	4.48
random170	5	100	757	97	746	3	1.45
random4	7	55	1097	51	1085	7.27	1.09
random3	6	10	96	10	96	0	0
random6	3	10	30	4	12	60	60
random179	7	100	1878	100	1878	0	0
random5	5	100	781	96	763	4	2.3
random8	5	100	624	95	609	5	2.4
random7	7	55	1164	55	1164	0	0
random9	7	10	246	10	246	0	0
random163	3	55	165	31	93	43.64	43.64
random285	7	10	192	10	192	0	0
random162	7	10	238	10	238	0	0
random286	5	55	375	50	360	9.09	4
random161	6	100	1379	98	1371	2	0.58
random287	7	100	2079	100	2079	0	0

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random160	7	55	1428	55	1428	0	0
random288	5	10	49	9	44	10	10.2
random167	7	100	1964	100	1964	0	0
random281	6	100	1467	100	1467	0	0
random166	3	55	165	15	45	72.73	72.73
random282	5	10	43	10	43	0	0
random165	5	10	58	10	58	0	0
random283	6	55	887	54	882	1.82	0.56
random164	4	100	410	77	329	23	19.76
random284	4	100	412	85	355	15	13.83
random280	4	55	211	43	163	21.82	22.75
random169	6	55	690	54	687	1.82	0.43
random168	6	10	185	10	185	0	0
random289	3	55	165	15	45	72.73	72.73
random196	7	55	1170	55	1170	0	0
random195	3	10	30	6	18	40	40
random194	3	100	300	36	108	64	64
random193	5	55	392	53	384	3.64	2.04
random199	4	55	215	48	190	12.73	11.63
random198	7	10	126	10	126	0	0
random197	6	100	1305	97	1294	3	0.84
random192	6	10	150	10	150	0	0
random96	4	10	44	8	34	20	22.73
random191	4	100	392	82	330	18	15.82
random97	4	55	223	47	195	14.55	12.56
random190	3	55	165	13	39	76.36	76.36
random98	3	100	300	30	90	70	70
random99	3	10	30	6	18	40	40
random185	4	100	432	80	352	20	18.52
random184	5	55	385	51	347	7.27	9.87
random183	7	10	115	10	115	0	0
random182	4	100	394	74	296	26	24.87
random189	6	10	140	10	140	0	0
random188	6	100	1307	98	1301	2	0.46
random187	6	55	604	55	604	0	0
random186	4	10	36	8	30	20	16.67
random89	5	100	694	91	659	9	5.04
random181	5	55	395	54	392	1.82	0.76
random85	3	55	165	23	69	58.18	58.18
random180	3	10	30	6	18	40	40
random86	3	100	300	44	132	56	56
random87	5	10	67	10	67	0	0
random88	3	55	165	15	45	72.73	72.73
random92	6	100	1310	97	1297	3	0.99
random93	5	10	51	8	45	20	11.76
random94	7	55	997	55	997	0	0
random95	5	100	764	99	757	1	0.92
random90	4	10	42	8	36	20	14.29
random91	5	55	346	48	305	12.73	11.85
random418	4	55	223	42	180	23.64	19.28
random419	6	100	1162	99	1157	1	0.43
random414	6	10	121	10	121	0	0
random415	7	55	1282	53	1276	3.64	0.47
random416	6	100	1284	100	1284	0	0
random417	7	10	217	10	217	0	0
random410	5	100	662	96	650	4	1.81
random411	7	10	208	10	208	0	0
random412	3	55	165	21	63	61.82	61.82

Table A.1: Experiment result after applying templates on randomly generated circuits.

Circuits		Original Circuit		Optimized Circuit		% of Reduction	
Function	Lines	OGC	OQC	NGC	NQC	PGC	PQC
random413	6	100	1420	98	1414	2	0.42
random407	6	100	1418	99	1415	1	0.21
random408	3	10	30	8	24	20	20
random409	5	55	414	52	399	5.45	3.62
random403	6	55	925	53	861	3.64	6.92
random404	7	100	2004	99	1981	1	1.15
random405	5	10	58	9	53	10	8.62
random406	7	55	1262	54	1257	1.82	0.4
random400	5	55	360	50	345	9.09	4.17
random401	3	100	300	50	150	50	50
random402	4	10	38	8	32	20	15.79
random430	6	55	549	55	549	0	0
random310	5	55	414	47	380	14.55	8.21
random431	7	100	2043	100	2043	0	0
random319	7	55	1083	55	1083	0	0
random315	4	10	44	9	37	10	15.91
random436	3	55	165	9	27	83.64	83.64
random316	7	55	1074	55	1074	0	0
random437	5	100	700	97	691	3	1.29
random317	5	100	736	91	681	9	7.47
random438	4	10	40	8	34	20	15
random318	3	10	30	6	18	40	40
random439	5	55	368	51	356	7.27	3.26
random311	4	100	408	84	356	16	12.75
random432	3	10	30	2	6	80	80
random312	7	10	158	10	158	0	0
random433	6	55	663	55	663	0	0
random313	3	55	165	23	69	58.18	58.18
random434	5	100	739	92	715	8	3.25
random314	4	100	396	81	331	19	16.41
random435	5	10	48	9	45	10	6.25
random420	7	10	206	10	206	0	0
random308	5	100	815	97	786	3	3.56
random429	7	10	173	10	173	0	0
random309	6	10	111	10	111	0	0
random304	6	55	914	55	914	0	0
random425	6	100	1251	97	1242	3	0.72
random305	4	100	412	88	368	12	10.68
random426	7	10	172	8	166	20	3.49
random306	6	10	147	8	137	20	6.8
random427	4	55	225	46	198	16.36	12
random307	4	55	225	49	207	10.91	8
random428	7	100	1947	99	1944	1	0.15
random300	7	10	122	10	122	0	0
random421	7	55	1110	55	1110	0	0
random301	6	55	738	55	738	0	0
random422	3	100	300	16	48	84	84
random302	3	100	300	30	90	70	70
random423	4	10	40	10	40	0	0
random303	3	10	30	2	6	80	80
random424	7	55	1253	52	1238	5.45	1.2
random500	6	100	1234	100	1234	0	0