SYNTHESIS AND TESTING OF REVERSIBLE TOFFOLI CIRCUITS

NOOR MUHAMMED NAYEEM
Bachelor of Science, University of Dhaka, 2008

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

MASTER OF SCIENCE

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

SYNTHESIS AND TESTING OF REVERSIBLE TOFFOLI CIRCUITS

NOOR MUHAMMED NAYEEM

Approved:

Signature	*Date*
Supervisor:	
Committee Member:	
Committee Member:	
External Examiner:	
Chair, Thesis Examination Committee:	

To my beloved parents

Abstract

Recently, researchers have been interested in reversible computing because of its characteristics to dissipate nearly zero heat and because of its applications in quantum computing and low power VLSI design. Synthesis and testing are two important areas of reversible logic. The thesis first presents an approach for the synthesis of reversible circuits from the exclusive-OR sum-of-products (ESOP) representation of functions, which makes better use of shared functionality among multiple outputs, resulting in up to 75% minimization of quantum cost, compared to the previous approach. This thesis also investigates the previous works on constructing the online testable circuits and points out some design issues. A simple approach for online fault detection is proposed for a particular type of ESOP-based reversible circuits, which is also extended for any type of Toffoli circuits. The proposed online testable designs not only address the problems of the previous designs but also achieve significant improvements of up to 78% and 99% in terms of quantum cost and garbage outputs, respectively.

Acknowledgments

I owe a great debt of gratitude to my supervisor Dr. Jacqueline E. Rice for her continuous support, inspiration and guidance throughout my M.Sc. program. I owe sincere thanks to my committee members, Professor Shelly Wismath and Dr. Kevin Grant. I am grateful to Professor Wismath for taking the time to give me detailed comments on an earlier draft of this thesis, which helped to improve the thesis.

I would like to thank Professor Guan Zhijin, Dr. Ding Weiping, and Dr. Hang Yueqin of Nantong University, China for their suggestions. I would also like to thank Professor Gerhard W. Dueck at the University of New Brunswick, Canada for providing me the shared cube synthesis tool. Special thanks to Navid Farazmand of Northeastern University, USA for his help in understanding the simulation results of the dual rail coding approach. Also thanks to Md. Raqibur Rahman for his discussions on ternary logic.

Finally, I would like to thank my parents, sister and brother for their encouragement and support.

Contents

List of Acronyms

BDD — Binary decision diagram

DFT — Design-for-test

DRG — Deduced reversible gate

ESOP — Exclusive-OR sum-of-products

ETG — Extended Toffoli gate

EXOR — Exclusive-OR

EXNOR — Exclusive-NOR

ILP — Integer linear program

M-S — Muthukrishnan-Stroud

MVL — Multiple-valued logic

OTG — Online testable gate

PPRM — Positive polarity Reed-Muller

TB — Testable block

TC — Test cell

TRC — Testable reversible cell

List of Tables

List of Figures

Chapter 1

Introduction

Traditional logic computation is irreversible since the outputs do not have enough information to reconstruct the inputs. In logic calculation, if there are p inputs and q outputs such that $p > q$, then at least $p - q$ bits of information are lost [55]. As an example, a logic gate with two inputs and one output destroys at least one bit of information during computation. In [55], for 16 different functions with two inputs and one output, the exact numbers of bits that are lost were calculated. Landauer's principle [20] states that each bit of information that is disregarded results in dissipation of heat, regardless of underlying technology. The amount of dissipated heat is at least $kT\ln 2$ joules for every bit of lost information, where k is the Boltzmann's constant and T is the absolute temperature. At room temperature, this amount becomes 2.9×10^{-21} Joules.

According to [24], the problem of heat dissipation arises from (1) technological deviation from ideality of switches and materials and (2) Landauer's principle. The current technologies have addressed the first part of the problem by reducing the heat loss. However, information loss in irreversible computation, which is the second part of the problem, will cause a considerable amount of heat generation in the near future due to increasing density of circuits. As an example [32], the packing densities in excess of 10^{17} logic devices in a cubic centimeter cause the devices to dissipate at least $kT\ln 2$, which result in more than 3,000,000 watts while operating at room temperature at a frequency of 10 gigahertz.

Reversible logic is being considered as an alternative of traditional irreversible logic since reversible computing does not erase or lose any information. As a result, reversible logic has a theoretical potential to dissipate no energy. According to Frank [10], reversible logic can recover a fraction of energy that can reach up to 100%. As there is no limit in reducing the heat dissipation in reversible logic, the amount of dissipated heat will become

very close to zero with the development of hardware.

Reversible logic is a vital part of quantum computing since quantum computation is reversible, and the physical reality of quantum logic can be illustrated by reversible logic [1]. Interested readers can refer to [41] for a detailed discussion of quantum computing. Reversible computing is also useful for other technologies including low power CMOS design [3], optical computing [43], nanotechnology [31], and bioinformatics.

1.1 Goals of the Thesis

Reversible logic uses a different set of reversible gates rather than traditional AND or OR gates to realize circuits. Moreover, fan-outs and loops are prohibited in reversible circuits. These cause the synthesis of reversible logic to be different from that of traditional irreversible logic, making synthesis a challenging area of research. A goal of this thesis is to present an optimized reversible logic synthesis approach for functions, especially large ones, given in the form of exclusive-OR sum-of-products (ESOP).

Testing is important for circuit design. This thesis aims to thoroughly examine the prior works on online testable approaches and presents new online testable reversible designs which are efficient in different cost metrics and deal with the problems encountered in the prior works.

1.2 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 introduces the reader to the world of reversible logic. The basics of reversible gates and cost metrics used for evaluating reversible circuits are covered. This chapter also provides an overview of various synthesis approaches and fault models for reversible

circuits.

Chapter 3 begins with the representation of ESOP functions and then focuses on a particular type of synthesis which uses this representation. Algorithms and examples are given so that readers will gain a clear knowledge of ESOP-based synthesis.

In Chapter 4, we first provide the motivational examples for improving the ESOP-based synthesis. We then propose an optimized approach, followed by an example to illustrate the approach step-by-step. Experimental results are also tabulated to show the superiority of this approach. This chapter has been published in [37] and [39].

Starting in Chapter 5, the focus of the thesis turns to testing of reversible logic. For offline testing, the covered topics include test set generation and design-for-test (DFT) methods. A major part of this chapter is devoted to online testing. We look at different approaches in detail for designing testable circuits and also analyze the issues related to these designs with suitable examples. Part of this chapter has been published in [38].

In Chapter 6, we present in detail two approaches for constructing online testable reversible circuits. Analyses of the presented approaches along with proofs are provided. Experimental results for a number of benchmarks are compared to the previously reported approaches. This chapter has been published in [40] and [38].

Chapter 7 concludes the thesis by highlighting the major contributions of this thesis and describing the further research directions in the areas of both synthesis and testing.

Chapter 2

Background

2.1 Logic Function

Let $A = \{0, 1\}$ be a set of binary or Boolean logic values. A logic function is simply a function of the form $f : A^p \rightarrow A^q$ for some natural numbers p and q. Such function can be expressed in terms of logical operations which are AND, OR, NOT, NAND, NOR, exclusive-OR (EXOR), and exclusive-NOR (EXNOR) operations. A logic gate implements a logical operation. In traditional irreversible logic, AND, OR, NOT, NAND, NOR, EXOR, and EXNOR gates are used as standard gates. Among all the logic gates, only NAND and NOR gates are universal since NAND gates alone (or alternatively NOR gates alone) can be used to implement any logic function. A circuit which is an electronic representation of a function is made from gates by interconnecting the outputs of some gates to the inputs of others. Logic synthesis is the process of transforming a logic function into a circuit design in terms of gates. The resultant circuit is called a realization of the given function.

2.2 Reversible Logic

A function is reversible if it is bijective (*i.e.*, one-to-one and onto) [49]. In other words, a reversible function has the same number of inputs as outputs (*i.e.*, $p = q$), and there is a one-to-one mapping between its input and output vectors. A reversible gate realizes a reversible function.

An example of a truth table for a particular reversible function with three inputs (k_1, k_2, and k_3) and three outputs (o_1, o_2, and o_3) is given in Table 2.1. The number of input variables and the number of output variables are the same. From the truth table, it can be seen

that for any two input vectors, the corresponding output vectors are different. Similarly, for any two output vectors, the corresponding input vectors are different.

Table 2.1: Truth table of an arbitrary reversible function.

k_1	k_2	k_3	o_1	o_2	o_3
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	1	0

The AND gate (see truth table in Table 2.2) is not reversible for the following two reasons:

- it has two input variables but only one output variable, and
- for three different input vectors, the output vectors are the same.

If the output is 1, its input vector is [1, 1] which can be uniquely identified. However, if the output is 0, unique identification of its input vector is not possible since there are three possible input vectors - [0, 0], [0, 1], and [1, 0]. Other traditional logic gates such as OR, NAND, NOR, and EXOR are also not reversible. The only exception is the NOT gate which is reversible.

Table 2.2: Truth table of an AND gate.

Inputs		Output
0	0	0
0	1	0
1	0	0
1	1	1

2.3 Reversible Gates

A different set of reversible logic gates are used to build reversible circuits since traditional logic gates other than the NOT gate are not applicable in reversible logic. The two most popular reversible gates are the Toffoli gates and the Fredkin gates.

2.3.1 Toffoli Gates

An n-bit Toffoli gate is a reversible logic gate that has n inputs and n outputs and that maps the input vector [k_1, k_2, ..., k_n] to the output vector [o_1, o_2, ..., o_n], where $o_j = k_j$ for j = 1, 2, ..., $n-1$ and $o_n = k_1 k_2 \cdots k_{n-1} \oplus k_n$. Here, the symbol $\oplus$ denotes the EXOR operation. The first $n-1$ bits are known as controls, and the last (n^{th}) bit is the target. This gate passes the input values at controls directly to the corresponding outputs without any change and toggles the target bit if and only if all input values at controls are 1.

The NOT gate is a special case of a Toffoli gate with n = 1 and no controls. The 2-bit (that is, n = 2) Toffoli gate is also known as the CNOT gate or Feynman gate. A NOT gate, a CNOT gate and an n-bit Toffoli gate are shown in Figure 2.1(a), Figure 2.1(b) and Figure 2.1(c).

A negative-control Toffoli gate maps the input vector [k_1, k_2, ..., k_n] to the output vector [o_1, o_2, ..., o_n], where $o_j = k_j$ for j = 1, 2, ..., $n-1$, $o_n = \bar{k}_1 k_2 \cdots k_{n-1} \oplus k_n$, and k_1 is a negative control. This gate may have one or more negative controls; in that case, the target bit is toggled if all positive controls have the value 1 and the negative controls have the value 0. A 3-bit Toffoli gate with a single negative control in its first input is shown in Figure 2.1(d).

The extended Toffoli gate (ETG) is a multi-target Toffoli gate proposed in [6]. In this thesis, we make use of an (n+1)-bit ETG with two target outputs o_n and o_{n+1} as shown in Figure 2.1(e). This gate has the input vector [k_1, k_2, ..., k_n, k_{n+1}] and the output vector [o_1,

o_2, ..., o_n, o_{n+1}], where $o_j = k_j$ for j = 1, 2, ..., $n-1$, $o_n = k_1k_2\cdots k_{n-1} \oplus k_n$, and $o_{n+1} = k_1k_2\cdots k_{n-1} \oplus k_{n+1}$. The first $n-1$ bits are controls and the last two bits are targets. Like a negative-control Toffoli gate, an ETG may have negative controls.

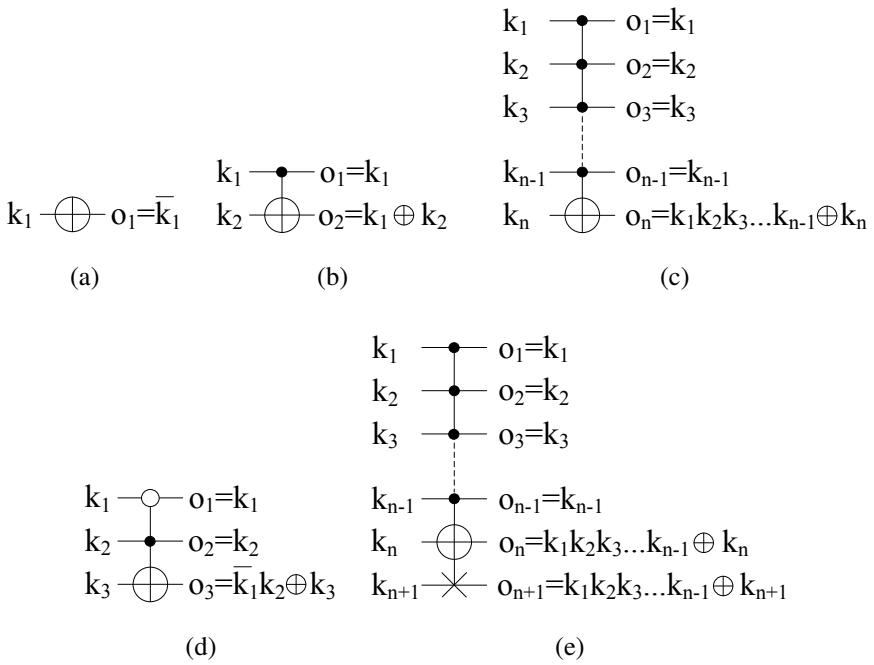


Figure 2.1: (a) A NOT gate, (b) A CNOT gate, (c) an n-bit Toffoli gate, (d) a 3-bit negative-control Toffoli gate, and (e) an (n+1)-bit ETG.

The n-bit ($n > 2$) Toffoli gates including the negative control Toffoli gates and ETGs are universal. For example, a 3-bit Toffoli gate has the mapping from [k_1, k_2, k_3] to [$o_1 = k_1$, $o_2 = k_2$, $o_3 = k_1k_2 \oplus k_3$]. Table 2.3 shows the truth table of this gate. When $k_3 = 1$, we get ($\overline{k_1k_2}$) which is the NAND operation of two inputs k_1 and k_2. Since the NAND gate is universal and the 3-bit Toffoli gate can work like a NAND gate, it is also universal.

2.3.2 *Fredkin Gates*

An n-bit Fredkin gate is a reversible logic gate that has n inputs and n outputs and that maps the input vector [k_1, k_2, ... k_{n-1}, k_n] to the output vector [o_1, o_2, ..., o_n], where $o_j = k_j$ for

Table 2.3: Truth table of the 3-bit Toffoli gate.

k_1	k_2	k_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

$j = 1, 2, ..., n-2$, $o_{n-1} = (\overline{k_1 k_2 \cdots k_{n-2}})k_{n-1} + k_1 k_2 \cdots k_{n-2} k_n$, and $o_n = (\overline{k_1 k_2 \cdots k_{n-2}})k_n + k_1 k_2 \cdots k_{n-2} k_{n-1}$. In other words, the first $n-2$ input values are passed directly to the corresponding outputs without any change, and the last two inputs are swapped if and only if all the first $n-2$ inputs are 1. The 2-bit (that is, $n = 2$) Fredkin gate is also known as the swap gate.

Like the Toffoli gates, the Fredkin gates with $n > 2$ are also universal. A 3-bit Fredkin gate has the input vector $[k_1, k_2, k_3]$ and the output vector $[o_1 = k_1, o_2 = \overline{k}_1 k_2 + k_1 k_3, o_3 = \overline{k}_1 k_3 + k_1 k_2]$. Its truth table is given in Table 2.4. By setting $k_3 = 1$, the OR operation of two inputs k_1 and k_2 can be realized from o_2.

Table 2.4: Truth table of the 3-bit Fredkin gate.

k_1	k_2	k_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

2.4 Reversible Circuit

A reversible circuit consists of only reversible gates which are interconnected without fan-out and feedback [49]. The fan-out refers to the maximum number of inputs that can be driven from one output of a gate. In a reversible circuit, the fan-out of a gate is at most one. If a reversible circuit is built using only Toffoli gates including NOT gates, CNOT gates and negative-control Toffoli gates, it is known as a Toffoli circuit. Figure 2.2 shows a Toffoli circuit where the three outputs of the 3-bit Toffoli gate are connected to the first three inputs of the 4-bit Toffoli gate.

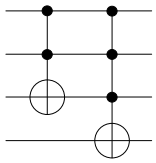


Figure 2.2: A Toffoli circuit.

2.5 Cost Metrics

A given function can be synthesized by several ways (see Section 2.6), resulting in different reversible circuits. This section outlines three cost metrics which are used to evaluate and compare different circuits realizing the same function.

2.5.1 Gate Count

Gate count is the simplest way to compare and evaluate different reversible circuits. It refers to the number of gates required to implement the circuit. It simply counts gates but does not take into account the complexity of the gates. As a result, it can compare different circuits only if the functionality (type) of the gates and the number of bits in the gates

used in circuits are similar [36]. For example, consider two circuits where the first circuit consists of three 2-bit Toffoli gates and the second circuit consists of two 10-bit Toffoli gates. According to this measure, the second circuit is better. However, a 10-bit Toffoli gate is more complex than a 2-bit Toffoli gate. Since gates have different numbers of bits, this simple measure fails to provide accurate information. Similarly, it cannot accurately evaluate circuits if one circuit contains only Toffoli gates and the other circuit contains both Toffoli and Fredkin gates.

2.5.2 *Garbage Output*

In reversible circuits, some outputs are required to maintain the reversibility property but do not behave as final results, nor are they used for further calculations. Such outputs are called garbage outputs. For example, a CNOT gate can be used to realize an EXOR operation as shown in Figure 2.3. The second output is $o_2 = i_1 \oplus i_2$, whereas the first output $o_1 = i_1$ is a garbage output. Thus a circuit for generating an EXOR function with a CNOT gate produces a garbage output.

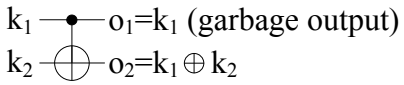


Figure 2.3: Garbage output in a reversible circuit.

Garbage outputs increase the number of lines[1] in a circuit and hence increase the width of the circuit. For some circuits, however, it is impossible to remove all the garbage outputs. It is more important to reduce the garbage outputs than the number of gates [26, 53]. Thus a challenging task in designing a reversible circuit is to minimize the number of required garbage outputs.

[1] We refer to bits in reversible circuits as lines in this thesis. The correct term in quantum computing is qubits.

2.5.3 *Quantum Cost*

The quantum cost is a very popular measure to compare reversible circuits. According to Maslov and Dueck [26], the quantum cost of a gate is defined as the number of basic quantum operations needed to realize the gate. Any reversible gate can be decomposed into basic quantum (1×1 and 2×2) gates. The number of basic quantum gates is the quantum cost.

The quantum cost calculation of an n-bit Toffoli gate is presented in [4], and improved in [25] by adding CNOT gates, and further improved in [23] by applying templates [30]. The quantum costs of the NOT gate, CNOT (2-bit Toffoli) gate, and 3-bit Toffoli gate are 1, 1, and 5, respectively. The number of bits in a Toffoli gate increases the quantum cost. For example, a 4-bit Toffoli gate has a cost of 13, whereas a 6-bit Toffoli gate has a cost of up to 61 [23]. If the number of lines in a circuit is greater than the number of bits in the Toffoli gate, then the extra (garbage) lines can help reduce the cost of the gate. For example, the cost of a 6-bit Toffoli gate is 52 if one garbage line is used, but is reduced to 38 if three garbage lines are used [23].

The costs of an n-bit Toffoli gate and an n-bit negative-control Toffoli gate with at least one positive control are exactly the same [2]. However, for an n-bit negative-control Toffoli gate with all negative controls, an extra cost of 2 is required if either zero garbage lines are used [2] or $(n-3)$ garbage lines are used [28, 2]. An additional cost of 4 is required if only one garbage line is used [28, 2].

The quantum costs for the Toffoli gates are given in Table 2.5. The first column indicates the number of bits in the Toffoli gates. The second column shows the number of garbage lines used to realize the gates. The third and fourth columns give the quantum costs for Toffoli gates with all positive and all negative controls, respectively. The first three columns are extracted from [23], while the last column is obtained based on the analysis given in [28, 2].

Table 2.5: Costs of n-bit Toffoli gates.

Number of bits n	**Garbage**	**Quantum Cost**	
		with all positive controls	**with all negative controls**
1	0	1*	1*
2	0	1	3
3	0	5	6
4	0	13	15
5	0	29	31
5	2	26	28
6	0	61	63
6	1	52	56
6	3	38	40
7	0	125	127
7	1	80	84
7	4	50	52
8	0	253	255
8	1	100	104
8	5	62	64
9	0	509	511
9	1	128	132
9	6	74	76
10	0	1021	1023
10	1	152	156
10	7	86	88
$n > 10$	0	$2^n - 3$	$2^n - 1$
$n > 10$	1	$24n - 88$	$24n - 84$
$n > 10$	$n - 3$	$12n - 34$	$12n - 32$

* 1-bit Toffoli gate (NOT gate) has no control.

The cost of an (n+1)-bit ETG is two plus the cost of an n-bit Toffoli gate, since it can be simulated by an n-bit Toffoli gate and two CNOTs. Similarly, an (n+1)-bit negative-control ETG has the cost of two plus the cost of a negative-control n-bit Toffoli gate. As given in [23], the cost of an n-bit Fredkin gate is computed as the cost of an n-bit Toffoli gate plus two.

We have discussed the quantum costs of different Toffoli and Fredkin gates. To calculate the quantum cost of a reversible circuit, we sum the quantum costs of the gates used in the circuit. As described in [36], comparisons in terms of gate count are not accurate if the reversible gates used in circuits have different functionalities and/or numbers of bits. The quantum cost is a good cost metric in this regard since it counts the number of elementary (quantum) gates required for the circuit.

2.6 Synthesis Approaches of Reversible Logic

There are a number of different approaches for synthesis of reversible logic circuits, including the transformation-based approach [33], the use of positive polarity Reed-Muller expressions (PPRM) [11], the use of ESOP expressions [9, 45], and the decision diagram-based approach [57].

2.6.1 Transformation-based Synthesis

The transformation-based approach proposed in [33] involves examination of a truth table of a given reversible function, and identifies transformations that can be applied to the output side of the truth table to match the input and output patterns. The application of these transformations then can be translated into a cascade of gates; if the transformations are restricted to only those corresponding to Toffoli gates, then the resulting cascade will

similarly consist only of Toffoli gates.

An improvement of the above approach is the bi-directional approach [33] which applies transformations both at the input side and output side of the truth table, resulting in a reduced number of gates in the circuit.

The circuits generated by the above two approaches are not optimal. The template matching technique [33] is thus applied to further optimize the circuits. A template consists of two sequences of gates where both sequences produce identical outputs, but the second sequence has smaller gate count and / or quantum cost than the first one. This technique looks for a match in the circuit for the first sequence of the template. If a match is found, the first sequence is replaced with the second one in the circuit. A library of templates is available in [33].

2.6.2 PPRM-based Synthesis

Gupta *et al.* [11] proposed a reversible logic synthesis approach which uses a PPRM representation of a function. This approach generates a search tree with the root node containing the PPRM expansions of all output variables. Common factors among the PPRM expansions of multiple outputs are identified. Each factor is then substituted into the PPRM expansions to create a new node with modified expansions. The new node is discarded if the substitution fails to reduce the number of terms in the expansions. Heuristics and a priority queue are used to process the tree efficiently. The substitutions that lead to the best solution are transformed into Toffoli gates.

2.6.3 *ESOP-based Synthesis*

An ESOP is a sum-of-products format in which the traditional OR operator combining the product terms is replaced with an EXOR operator. For example, a function $f = xy + yz$ in the SOP form can be rewritten as $f = xy \oplus \bar{x}yz$ which is in an ESOP form. Tools such as EXORCISM [35] can be used to generate an ESOP expression representing the given function.

In ESOP-based synthesis, ESOP terms are transformed into Toffoli gates. A basic synthesis approach was introduced in [9] and has been since improved upon in a number of works including [45, 47, 46]. Since our work is related to this particular type of synthesis, we review a number of ESOP-based approaches in detail in Chapter 3

2.6.4 *Decision Diagram-based Synthesis*

Wille and Drechsler [57] proposed a synthesis of Toffoli circuits from functions specified in binary decision diagrams (BDDs). A BDD of the given function is built using a tool such as CUDD [50]. Each node in the BDD is transformed into a cascade of Toffoli gates. Note that shared nodes in the BDD cause fan-outs in the resulting circuit, which are not allowed in reversible logic. To avoid fan-outs, additional lines are introduced to the circuit.

The approaches that we have described in this section generate Toffoli circuits. There are other approaches in the literature such as [27] and [7] that incorporate Fredkin gates in the circuits.

2.7 Fault Models

A fault is a physical defect in a system [19]. In other words, a system has a fault if there is a physical difference between the correct system and the current system [15]. Part of the

work in this thesis focuses on identifying faults in reversible circuits, and thus we briefly address fault models in this section. A fault model simplifies the analysis and complexity of testing by reducing the number of defects that have to be considered. Depending on the number of faults in a circuit, fault models can be categorized as single fault models and multiple faults models. The single fault model considers only one fault in a circuit whereas multiple faults model deals with several faults.

2.7.1 Stuck-at Fault Model

The stuck-at fault model is a very common fault model in traditional irreversible circuits. According to Lala [19], a stuck-at fault in a logic gate causes one of its inputs or outputs to be stuck either at a logic value 0 (stuck-at-0) or at a logic value 1 (stuck-at-1), irrespective of the inputs of the circuit. For example, consider a reversible circuit consisting of two Toffoli gates as shown in Figure 2.4. A stuck-at-1 fault in the first output of the first gate causes o_1 to have the value one, independent of its input values.

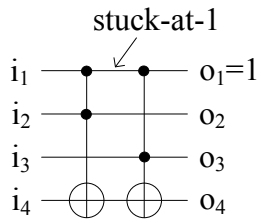


Figure 2.4: A stuck-at fault.

2.7.2 Bit Fault Model

The bit fault model has been considered in various articles such as [54, 53, 21, 22]. In this model, a fault in a gate changes the behavior of its outputs. A single bit fault is reflected to exactly one output of a gate, changing the correct value of the output to a faulty value. A

stuck-at fault sets the output of a gate to either 0 or 1, whereas a bit fault flips the output of a gate (from 0 to 1 or vice versa). Unlike the stuck-at fault model, this model depends on the input values.

2.7.3 Missing, Repeated and Reduced Gate Fault Models

For reversible logic, Hayes *et al.* [12] proposed three new fault models: missing gate, repeated gate, and reduced gate fault models. A brief discussion is given below.

Missing gate fault model In the missing gate fault model, a gate has completely disappeared from a circuit. Thus this gate is replaced by simple wire connections. For example, the first gate in the circuit shown in Figure 2.5(a) is removed due to the missing gate fault, resulting in the circuit shown in Figure 2.5(b).

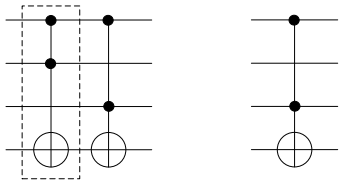


Figure 2.5: (a) Before and (b) after the occurrence of a missing gate fault.

Repeated gate fault model A repeated gate fault occurs if a gate is duplicated in a circuit. For example, the first gate in the circuit given in Figure 2.6(a) is duplicated in the circuit shown in Figure 2.6(b). Thus the latter circuit has two copies of that gate.

Reduced gate fault model In the reduced gate fault model, the output of a gate computes a partial function. For example, assume the correct output of a gate is $o = k_1k_2k_3 \oplus k_4$, where k_1, k_2, k_3, and k_4 are inputs. A reduced gate fault may cause the gate to compute $o = k_2k_3 \oplus k_4$ instead, which is incorrect.

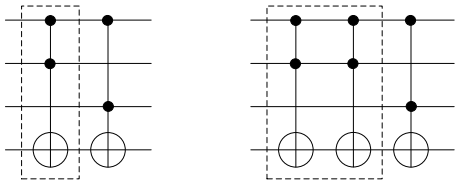


Figure 2.6: (a) Before and (b) after the occurrence of a repeated gate fault.

2.7.4 Crosspoint Fault Model

In [59], Zhong and Muzio proposed the crosspoint fault model for Toffoli gates. A crosspoint fault occurs if the existing controls of a Toffoli gate do not work properly or if extra control points change the behavior of a Toffoli gate. Such faults can therefore be classified as disappearance faults or appearance faults.

Disappearance fault A disappearance fault removes one or more control points from a Toffoli gate and consequently, reduces the number of bits in the gate. For example, consider the 4-bit Toffoli gate in Figure 2.7(a). A disappearance fault in the control point, connected to the second line, changes the size of the Toffoli gate from 4-bit to 3-bit as shown in Figure 2.7(b). It is noted that the disappearance fault in a Toffoli gate is the same as the reduced gate fault [59].

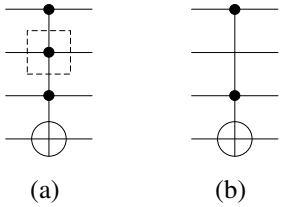
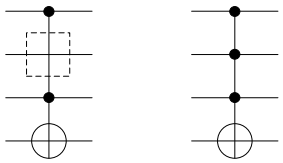


Figure 2.7: (a) Before and (b) after the occurrence of a disappearance fault.

Appearance fault An appearance fault introduces one or more additional control points to a Toffoli gate, increasing the number of bits in the gate. For example, consider the 3-bit Toffoli gate in Figure 2.8(a). Addition of one extra control in the second line

due to an appearance fault results in a 4-bit Toffoli gate as shown in Figure 2.8(b).



(a) (b)

Figure 2.8: (a) Before and (b) after the occurrence of an appearance fault.

Chapter 3

ESOP-based Synthesis

In an ESOP representation of a function, two or more product terms are EXORed together. A function in the ESOP form is commonly written as a list of cubes, known as a cube-list. A cube represents a term and has the form $x_1x_2...x_p\ f_1f_2...f_q$, where x_k for $k \in \{1, 2, ..., p\}$ is the input variable and f_j for $j \in \{1, 2, ..., q\}$ is the output variable. p and q are the number of input and output variables of the function.

We have $x_k \in \{1, 0, -\}$ and $f_j \in \{1, 0\}$. Let us assume that in any cube, 1 denotes the positive polarity, 0 denotes the negative polarity, and $-$ denotes the don't care value of the input variable x_k. Again, if $f_j = 1$, then the output variable f_j contains this cube (term); otherwise f_j does not contain the cube. A cube is called a shared cube if at least two output variables contain the cube. For example, given an ESOP function $f_1 = x_1x_2x_3 \oplus x_3x_4$ and $f_2 = x_1x_2x_3 \oplus x_4$, a cube-list is written as shown in Figure 3.1. Since the term $x_1x_2x_3$ is shared by two output variables f_1 and f_2, the corresponding cube `111- 11` is a shared cube, as shown in the first cube of cube-list in Figure 3.1. Note that although we have four terms in the function, we have a total of three cubes since one cube is a shared cube.

```
x1x2x3x4 f1f2
1 1 1 -  1 1
- - 1 1  1 0
- - - 1  0 1
```

Figure 3.1: An ESOP cube-list.

The ESOP-based reversible logic synthesis works with the ESOP representation of a function and synthesizes a reversible circuit by transforming the ESOP terms into a cascade of Toffoli gates. This synthesis is of interest because of the easy transformation of ESOP terms into a Toffoli network, as well as the ability to handle functions with large numbers of inputs. This chapter describes several different approaches for ESOP-based synthesis.

3.1 Basic Approach

Fazel *et al.* [9] proposed a basic ESOP-based reversible logic synthesis approach which works with the ESOP representation of a function and uses only Toffoli gates to implement the circuit. This approach requires two input lines corresponding to positive and negative polarities of each input variable x_k, and one output line (initialized by 0) for each output variable f_j. Thus the circuit initially has an empty cascade with $2p+q$ lines. A Toffoli gate is cascaded for each cube of each output. In other words, for each output variable where $f_j = 1$, each cube is mapped into a Toffoli gate where the controls of the gate are the input lines and the target is the output line f_j. If x_k has the positive polarity in the cube, then the control is connected to the input line x_k. If x_k has the negative polarity, then the control is connected to the input line $\bar{x}_k$. Thus this approach transforms the cube-list into a cascade of Toffoli gates. The algorithm is described below.

Algorithm 1: Basic ESOP-based synthesis approach [9]

Input: cube_list of a logic function

Output: gate_list, A cascade of Toffoli gates

Method

```
1. gate_list = NIL
2. for each x ∈ {input_variables}
3.     add_line (x)
4.     add_line (x̄)
5. for each f ∈ {output_variables}
6.     add_line (f)
7. for each cube ∈ cube_list
8.     control_list = NIL
9.     for each x ∈ {input_variables}
```

```
10.       if ( x in cube is 1)
11.          control_list.add(get_line(x))
12.       else if ( x in cube is 0)
13.          control_list.add(get_line(x̄))
14.    for each f ∈ {output_variables}
15.       if f in cube is 1
16.          target = get_line(f)
17.          gate = new Toffoli_gate(control_list, target)
18.          gate_list.add(gate)
19. return gate_list
```

Example 3.1.1. Consider an ESOP cube-list of four cubes with four inputs (x_1, x_2, x_3, x_4) and two outputs (f_1, f_2) as shown in Figure 3.2(a). The circuit generated by the basic ESOP-based approach is shown in Figure 3.2(b), which requires eight lines for input variables and two lines for output variables. For the first cube, two Toffoli gates are generated for two outputs. Then one Toffoli gate is generated for each of the last two cubes.

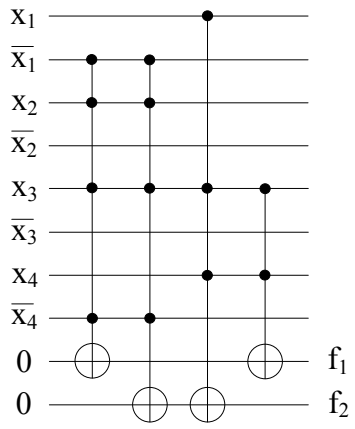
3.2 Optimization by Adding Not Gates

The basic approach always requires $2p+q$ lines, where p is the number of inputs and q is number of outputs. However, all the negated lines (lines corresponding to the negative polarity of input variables $x_1, x_2, \ldots, x_p$) are not utilized in all cases, and it is easy to get a negated line by inserting a NOT gate when necessary [9]. Thus the basic approach can be optimized by removing the negated lines, which considerably reduces the number of lines to $p+q$ but adds some NOT gates.

The basic approach with line optimization [9] is outlined in Algorithm 2 which is very similar to Algorithm 1. It also generates a Toffoli gate for each cube of each output where

x_1	x_2	x_3	x_4	f_1	f_2
0	1	1	0	1	1
1	-	1	1	0	1
-	-	1	1	1	0

(a)



(b)

Figure 3.2: (a) An ESOP cube-list and (b) a circuit generated by basic approach.

$f_j = 1$. The only difference is that one input line is used as a positive line and a negative line. Thus if the polarity of the input variable and the polarity of the corresponding line do not match, a NOT gate is added on the input line before inserting the Toffoli gate.

Algorithm 2: Basic ESOP-based synthesis approach [9] with line optimization

Input: cube_list of a logic function

Output: gate_list, A cascade of Toffoli gates

Method

```
1. gate_list = NIL
2. for each x ∈ {input_variables}
3.     add_line(x)
4. for each f ∈ {output_variables}
5.     add_line(f)
```

```
 6. for each cube ∈ cube_list
 7.    control_list = NIL
 8.    for each x ∈ {input_variables}
 9.       if x in cube is 1 or 0
10.          control_list.add(get_line(x))
11.          if polarity of x in cube ≠ polarity of get_line(x)
12.             gate = new NOT_gate(get_line(x))
13.             gate_list.add(gate)
14.    for each f ∈ {output_variables}
15.       if f in cube is 1
16.          target = get_line(f)
17.          gate = new Toffoli_gate(control_list, target)
18.          gate_list.add(gate)
19. return gate_list
```

Example 3.2.1. Consider the cube-list used in Example 3.1.1, which is also shown in Figure 3.3(a). The circuit generated by the basic approach with line optimization requires four NOT gates and four Toffoli gates as shown in Figure 3.3(b). The number of lines in this circuit is 6, compared to 10 lines in the circuit shown in Figure 3.2(b).

3.3 Cube Ordering Heuristics

The line optimization approach increases the number of gates because of inserting NOT gates. However, reordering of cubes in the cube-list can help reduce the number of NOT gates. To find the optimal reordering of cubes, the computation complexity is $O(n!)$, where n is the number of cubes in the cube-list. A large circuit can contain hundreds of cubes; thus it is infeasible to apply a brute force approach.

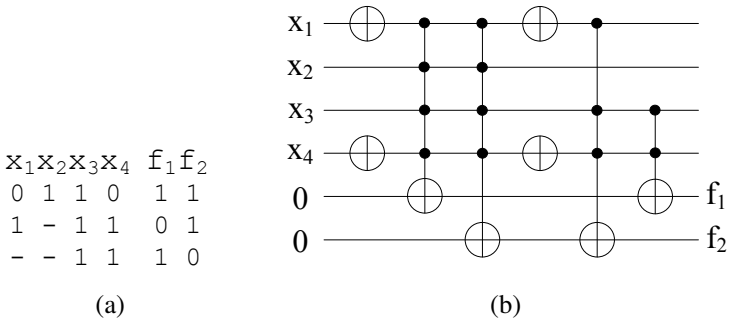


Figure 3.3: a) An ESOP cube-list and (b) a circuit generated by basic approach with line optimization.

In order to reorder the cubes, two heuristics using the alpha/beta cost metric [9] and autocorrelation coefficient based cost metric [45] were proposed. Both approaches follow the divide-and-conquer paradigm. The three steps are as follows.

- **Divide:** The input variable with the lowest cost metric is determined. Let x be such an input variable. The cube-list is divided into two lists based on the variable x. The first list consists of cubes which contain the positive polarity and the don't care value of x. The remaining cubes (*i.e.* cubes containing the negative polarity of x) are in the second list.
- **Conquer:** The two lists are ordered recursively.
- **Combine:** The two ordered lists are recombined to generate the final ordered list.

The two cost metrics are described in the following two subsections. Once the cube-list is ordered, the circuit can be realized using the basic ESOP-based approach with line optimization as described in the last section.

3.3.1 Alpha-beta Cost Metric

For the alpha/beta cost metric [9], the cost of each input variable x is calculated using Equation 3.1. An input variable has a lower cost if it is balanced in terms of its positive and negative forms and appears frequently in the cube-list. The coefficient α controls the variable frequency term, and β controls the balanced variable term.

$$cost_x = \frac{\alpha}{\sum_{i=1}^{n} |w_i|} + \beta \left| \sum_{i=1}^{n} w_i \right| \tag{3.1}$$

where x is the input variable,

$$w_i = \begin{cases} 1 \text{ if } x \text{ in } cube_i \text{ is } 1 \\ -1 \text{ if } x \text{ in } cube_i \text{ is } 0 \\ 0 \text{ if } x \text{ in } cube_i \text{ is } - \end{cases}$$

$cube_i$ is the i^{th} cube in the cube-list,

n is the number of cubes in the cube-list, and

α and β are two constants such that $\alpha + \beta = 1$.

Example 3.3.1. Consider an arbitrary cube-list of five cubes (C_1, C_2, C_3, C_4, C_5) with four input variables (x_1, x_2, x_3, x_4) and one output variable (f) as shown in Figure 3.4(a). We first calculate the cost of input variables x_1, x_2, x_3 and x_4. The values of the variable x_1 are 0, $-$, 0, $-$ and 1 in cubes C_1, C_2, C_3, C_4, and C_5, respectively. Thus $w_1 = -1$, $w_2 = 0$, $w_3 = -1$, $w_4 = 0$, and $w_5 = 1$ for variable x_1. If we assume α is 0.25 and β is 0.75 then the cost of x_1 is $cost_{x_1} = 0.25/3 + 0.75 \times 1 = 0.83$. The cost calculation of input variables is given in Table 3.1. Since x_3 has the lowest cost, the cube-list is split into two lists based on this variable as shown in Figure 3.4(b)-(c). The first list consists of C_3, C_4, and C_5 as these

cubes contain the positive value and don't care value of x_3, and the second list comprises the remainder of the cubes, *i.e.* C_1 and C_2. Costs are recalculated in each new list which is further split according to the new costs. The final reordered list is given in Figure 3.4(d).

	x_1	x_2	x_3	x_4	f
C_1:	0	-	0	-	1
C_2:	-	1	0	1	1
C_3:	0	0	1	-	1
C_4:	-	1	1	0	1
C_5:	1	-	1	1	1

(a)

	x_1	x_2	x_3	x_4	f
C_3:	0	0	1	-	1
C_4:	-	1	1	0	1
C_5:	1	-	1	1	1

(b)

	x_1	x_2	x_3	x_4	f
C_1:	0	-	0	-	1
C_2:	-	1	0	1	1

(c)

	x_1	x_2	x_3	x_4	f
C_5:	1	-	1	1	1
C_4:	-	1	1	0	1
C_3:	0	0	1	-	1
C_2:	-	1	0	1	1
C_1:	0	-	0	-	1

(d)

Figure 3.4: (a) Initial ESOP cube-list, (b) list 1 containing positive polarity and don't care value of x_3, (c) list 2 containing negative polarity of x_3, and (d) final reordered cube-list.

Table 3.1: Cost calculation of input variables for the cube-list in Figure 3.4(a).

Input variable x	$\sum_{i=1}^{5} \lvert w_i \rvert$	$\sum_{i=1}^{5} w_i$	α	β	$cost_x$
x_1	3	1	0.25	0.75	0.83
x_2	3	1			0.83
x_3	5	1			0.80
x_4	3	1			0.83

3.3.2 Autocorrelation Cost Metric

For the autocorrelation coefficient method [45], the cost metric is the autocorrelation coefficient of the input variable. The autocorrelation coefficient of a function f is defined as

follows [16]:

$$B(\tau) = \sum_{v=0}^{2^p-1} f(v) \times f(v \oplus \tau) \tag{3.2}$$

where p is the number of input variables and

τ is the value for which the coefficient value is calculated

The value of τ can range from 0 to $2^p - 1$. If $\tau = 0$, then there is no shift in the function as it is compared to itself with no changes. Since only the first order coefficients were used in [45], the number of ones in the binary value of τ is one. The coefficient value of a variable indicates the dependency of a function on that variable. For example, assume a variable ordering of $x_1x_2x_3x_4$. The calculation of $B(1000)$ estimates the dependency of a given function f on x_1. The lower the coefficient, the more dependent the function is on that particular variable. The following example describes how to calculate the autocorrelation coefficients of input variables and use them to reorder the cubes.

Example 3.3.2. Consider the cube-list used in Example 3.3.1, which is also shown in Figure 3.5(a). The autocorrelation coefficient of input variable x_1 is calculated as follows.

$$\begin{aligned}
B(1000) &= [f(0000) \times f(0000 \oplus 1000)] + [f(0001) \times f(0001 \oplus 1000)] + \ldots \\
&\quad + [f(1111) \times f(1111 \oplus 1000)] \\
&= [1 \times 0] + [1 \times 0] + [1 \times 0] + [1 \times 1] + [1 \times 0] + [1 \times 1] + [1 \times 1] \\
&\quad + [0 \times 1] + [0 \times 1] + [0 \times 1] + [0 \times 1] + [1 \times 1] + [0 \times 1] + [1 \times 1] \\
&\quad + [1 \times 1] + [1 \times 0] \\
&= 6
\end{aligned}$$

Similarly, we calculate the autocorrelation coefficients of x_2, x_3, and x_4, which are 8, 8, and 8 respectively. Since the coefficient value of x_1 is the lowest, the cube-list is split into two lists based on this variable as shown in Figure 3.5(b)-(c). The first list consists of C_2, C_4, and C_5 as these cubes contain the positive value and the don't care value of x_1, and the second list contains C_1 and C_3. Autocorrelation coefficients are recalculated in each new list which is further split according to the new values. The final reordered list is given in Figure 3.5(d).

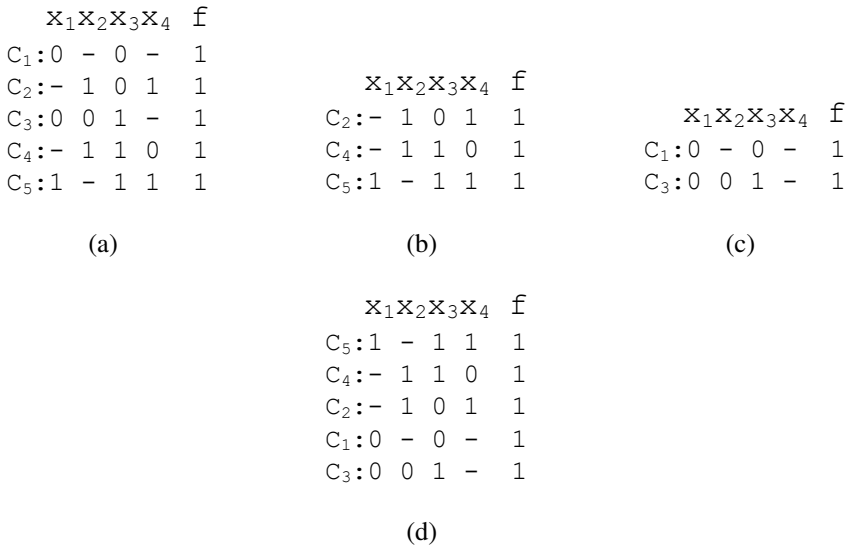


Figure 3.5: (a) Initial ESOP cube-list, (b) list 1 containing positive polarity and don't care value of x_1, (c) list 2 containing negative polarity of x_1, and (d) final reordered cube-list.

3.4 Optimization by Using Negative-control Toffoli gates

The negative-control Toffoli gate was proposed by Maslov and Miller [29] and was later used in [28, 2] including ESOP-based circuits [48, 46]. As described in Section 2.3.1, this gate has positive and negative controls. The use of this gate removes the NOT gates from the ESOP-based circuits; thus gate count and quantum cost are also reduced. The synthesis

approach is very similar to the basic approach [9] with line optimization (see Section 3.2). A Toffoli gate is generated for each cube of each output where $f_j = 1$. If x_k has the positive polarity in the cube, the positive control of the Toffoli gate is connected to the input line x_k. If x_k has the negative polarity in the cube, the negative control of the Toffoli gate is connected to line x_k. Thus the NOT gate is not required for cubes except for the one cube with $x_k = -$, $\forall\, k \in \{1, 2, \ldots, p\}$. For that particular cube, a NOT gate is added on the output line f_j if $f_j = 1$, for $j \in \{1, 2, \ldots, q\}$ in the cube.

Example 3.4.1. Consider the cube-list used in Example 3.1.1, which is also shown in Figure 3.6(a). A circuit which is built using positive and negative control Toffoli gates is shown in Figure 3.6(b). This circuit does not require any NOT gate, compared to the circuit in Figure 3.3(b) which requires four NOT gates.

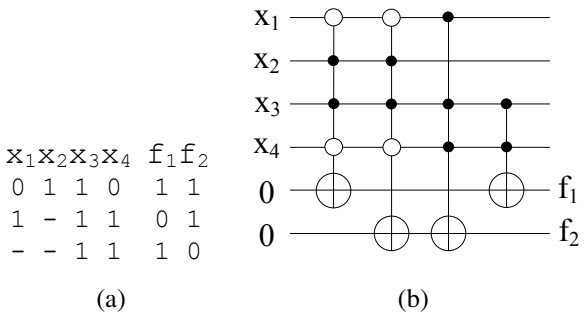


Figure 3.6: (a) An ESOP cube-list and (b) a circuit using negative-control Toffoli gates.

3.5 Shared Cube Synthesis

Approaches described in the previous sections produce similar Toffoli gates with identical controls but different targets when cubes are shared by two or more outputs. Shared cube synthesis [47, 46] improves the quality of the circuit by generating one Toffoli gate for each

of such shared cubes and transferring the cubes to other outputs through CNOT gates. The algorithm works as follows.

All pairs of outputs are examined to find a pair which has the largest number of shared cubes. One Toffoli gate is generated for each of these cubes, and the target line is the empty output line, *i.e.* the line which has not been used by the control or target part of any Toffoli gate. If both output lines are not empty, then one of the outputs is chosen arbitrarily as the target line. In the latter case, one CNOT gate needs to be added first to remove the impact of gates that exist on the line. In both cases, one CNOT gate is also added at the end to transform the shared functionality to other output. After that these two outputs are removed from the cube-list. The same process is repeated for other pairs of shared cubes. Finally, one Toffoli gate is added for each cube which is not shared by any other output. The algorithm is given below.

Algorithm 3: Shared-cube synthesis approach [46]

Input: cube_list of a logic function

Output: circuit, A cascade of Toffoli gates

Method

1. *circuit* = An empty cascade with *p*+*q* lines [*p* and *q* denote the number of input and output variables, respectively]
2. while there are still *shared cubes* in *cube-list*
3. (*i*, *j*) = a pair of outputs with the maximum number of *shared cubes*
4. if there are gates on both lines of *i* and *j*
5. add a CNOT gate to the *circuit* to remove the impact of other gates
6. add gates of *shared cubes* of (*i*, *j*) to *circuit*
7. add a CNOT gate to the *circuit* to transform the

```
        shared functionality to other output
 8.     remove shared cubes of (i, j) from the cube-list
 9. for each remaining cube in cube-list do
10.     add a gate to circuit
11.     remove the cube from cube-list
12. return circuit
```

Example 3.5.1. Consider a cube-list of four cubes with three input variables (x_1, x_2, x_3) and two output variables (f_1, f_2) as shown in Figure 3.7(a). A circuit generated by the basic approach with line optimization is given in Figure 3.7(b). As can be seen from the cube-list, both outputs share the first three cubes. Thus the shared cube synthesis produces three Toffoli gates with target line on f_1 and one CNOT gate to share these gates with f_2. At the end, one Toffoli gate is added for the last cube. The cascade is shown in Figure 3.7(c), which reduces the gate count by 2 and quantum cost by 14, compared to the circuit in Figure 3.7(b).

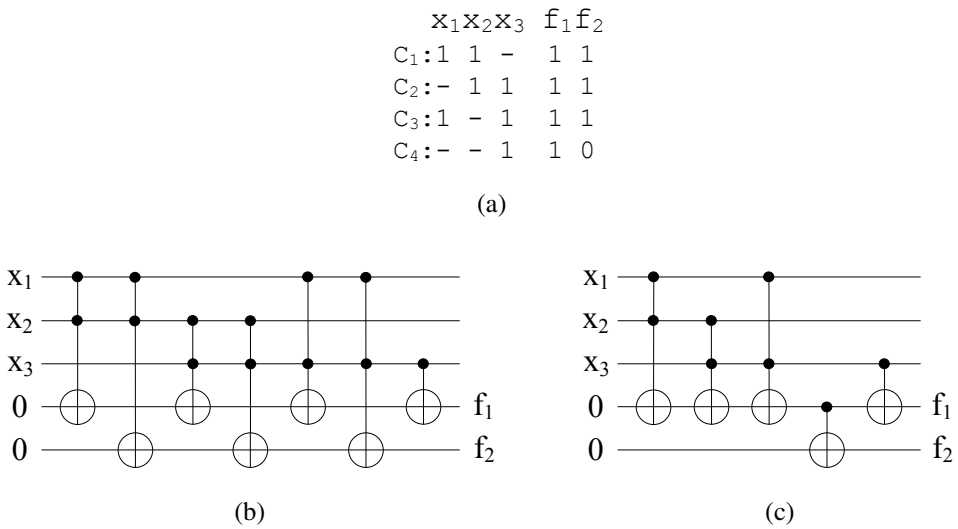


Figure 3.7: (a) An ESOP cube-list, and circuits generated (b) by basic approach with line optimization and (c) by shared cube synthesis.

3.6 Summary

In this chapter, we have discussed a number of approaches to realize reversible circuits from the ESOP representation of a function. The comparison results [45] show that the alpha/beta method, on average, performs better than the autocorrelation method in terms of number of gates and runtime. Shared cube synthesis reduces the quantum cost significantly since many large Toffoli gates are replaced by CNOT gates; thus it outperforms both the alpha/beta method and autocorrelation method.

Chapter 4

Improved ESOP-based Synthesis

In the last chapter, we discussed the shared cube synthesis [46, 47], a special type of ESOP-based approach. This chapter presents an improved shared cube synthesis that forms the contribution of this thesis to the area of reversible logic synthesis. Section 4.1 addresses the limitation of the previous work [46, 47] on utilizing the shared functionality when cubes are shared by more than two outputs. We present an optimized approach in Section 4.2, which overcomes the limitation by ensuring that the implementation of each cube in the cube-list requires exactly one Toffoli gate. Note that some CNOTs are also required to pass the shared functionality to other outputs. Experimental results are presented in Section 4.3.

4.1 Utilization of Shared Functionality

Shared cube synthesis works with multi-output functions if the ESOP terms (cubes) are shared by more than one output. For instance, given a multi-output function, $f_1 = x_1x_2 \oplus x_3x_4$ and $f_2 = x_1x_2x_3$, shared cube synthesis cannot improve the circuit as there is no shared term between f_1 and f_2. The existing shared cube synthesis [46, 47] discussed in Section 3.5 takes the best advantage of shared functionality if the ESOP terms are shared by only two outputs. However, if the shared terms exist in more than two outputs, transformation of each term may require more than one Toffoli gate, which is inefficient. The following two examples show that the existing approach can be further optimized. The optimized shared cube synthesis presented in this chapter produces one Toffoli gate for a cube and hence has the the potential of minimizing the gate count as well as quantum cost.

Example 4.1.1. Given a cube-list of the 3-input, 3-output function shown in Figure 4.1(a), a Toffoli cascade generated by the existing shared cube synthesis [46, 47] is shown in

Figure 4.1(b). This cascade requires two Toffoli gates for each of the three cubes. An equivalent Toffoli cascade depicted in Figure 4.1(c) generates one Toffoli gate for each cube, and hence minimizes the gate count by 4. Moreover, the quantum cost is reduced from 66 to 34.

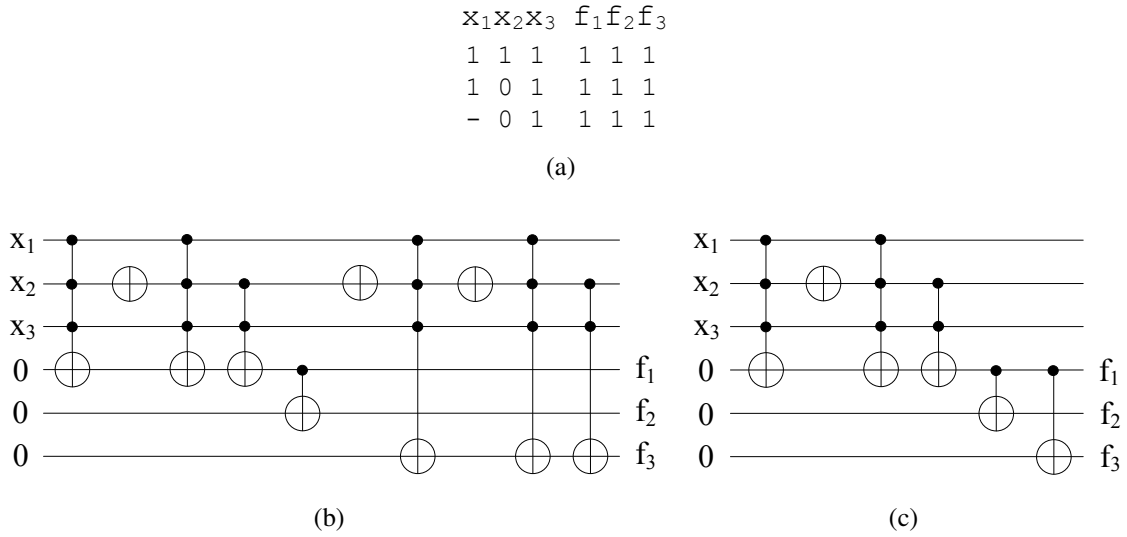
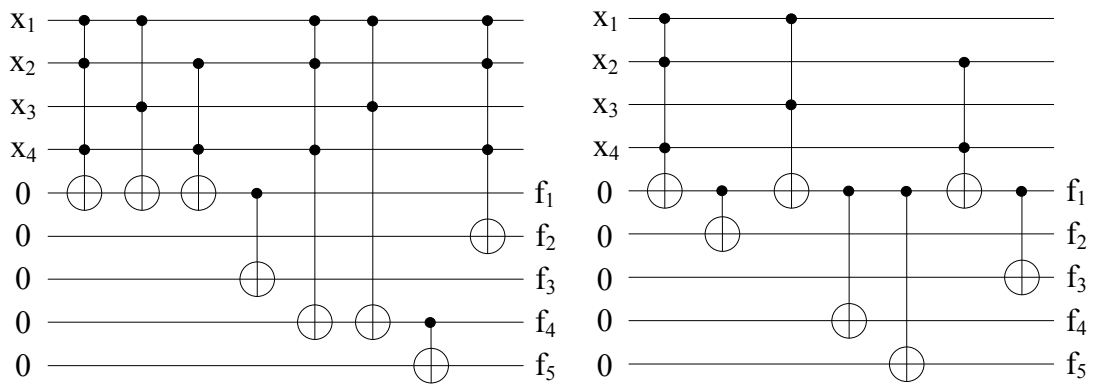


Figure 4.1: (a) An example cube-list, (b) a Toffoli cascade generated by the approach in [46, 47], and (c) an improved Toffoli cascade.

Example 4.1.2. Consider the cube-list given in Figure 4.2(a). The existing shared cube synthesis [46, 47] generates a Toffoli cascade containing three Toffoli gates for the first cube and two Toffoli gates for the second cube, a total of 8 gates as shown in Figure 4.2(b). However, an efficient synthesis optimizes the Toffoli cascade as shown in Figure 4.2(c). The quantum cost of this cascade is 27, in contrast to the former approach which costs 56. This example also shows an efficient way to make use of the shared functionality even if the cubes are not shared by all the outputs.

$x_1x_2x_3x_4$	$f_1f_2f_3f_4f_5$
1 1 - 1	1 1 1 1 1
1 - 1 -	1 0 1 1 1
- 1 - 1	1 0 1 0 0

(a)



(b) (c)

Figure 4.2: (a) An initial cube-list, (b) a Toffoli cascade generated by the approach in [46, 47], and (c) an improved Toffoli cascade.

4.2 Our Approach

Like other ESOP-based approaches, our proposed approach also works with the ESOP cube-list of a function. This approach optimizes the synthesis by generating exactly one Toffoli gate for one cube and by minimizing the CNOT gates required to transform the Toffoli gates to other output lines in the circuit. Minimization of CNOTs is performed by grouping together the Toffoli gates before passing these to other outputs via CNOTs. If the output line is not empty and a CNOT is required to remove the effect of other gates on that line before doing the transformation, we apply a technique to find a redundant CNOT gate before inserting a new one. A more detailed explanation of this approach with an example is given below.

In a cube-list, there can be some cubes which are not shared by multiple outputs. If the number of 1s in the output part of a cube is one, then only one output contains this cube and no other output shares it. This cube, called an ungrouped cube, will be dealt with

individually. Our proposed approach consists of the following two phases:

Phase 1 Generation of sub-lists

Phase 2 Transformation of sub-lists into gate-lists

Generation of sub-lists: This phase takes the original *cube-list* as its input and generates sub-lists as follows:

Step 1: Move ungrouped cubes from the *cube-list* into the *ungrouped-list*.

Step 2: Repeat Step 3 and Step 4 until *cube-list* is empty. Initialize the value of k with a 1 and increase its value by 1 after each iteration.

Step 3: Select a cube from the modified *cube-list* which is shared by the largest number of functions, *i.e.* has the maximum number of 1s in its output part. This cube and every other cube with an identical output part are moved to the $sub\text{-}list_k$.

Step 4: Select a cube from the *cube-list* which has the maximum number of 1s and which must have 0 at the output position at which the cubes in $sub\text{-}list_k$ have 0. In other words, the outputs that share this cube must also share all the cubes in $sub\text{-}list_k$. Afterwards this cube along with the cubes having identical output parts is moved from the *cube-list* to $sub\text{-}list_k$. This step continues until no such cube exists in the *cube-list*.

Transformation of sub-lists into gate-lists: In this phase, the sub-lists generated by phase 1 are transformed into a cascade of Toffoli gates. The *total-gate-list*, which is initially empty, will contain the final circuit at the end of this phase.

Step 1: For each $sub\text{-}list_k$, do Step 2 - Step 6.

Step 2: An output line p is selected as the Toffoli target line if the corresponding output contains all cubes in $sub\text{-}list_k$. If multiple such lines are found, choose one line arbitrarily that has not yet been used as a control or target of any Toffoli gate. If all such lines are occupied by other gates, choose the same line targeted in the last iteration, or any line arbitrarily.

Step 3: The *gate-list*$_k$ is initially empty. For each of the cubes in *sub-list*$_k$ perform steps 4-5 which add gates to the gate-list$_k$.

Step 4: Add a Toffoli gate that has a target on line p. The controls of this Toffoli are the input lines for which the input part of the corresponding cube contains zeros and ones. If the input part contains at least one zero, use a negative-control Toffoli gate. After that add a CNOT gate to transfer the gates to other output line(s) only if this cube is the last cube in the list that the output contains.

Step 5: If the line p has already hosted gates before the beginning of Step 2, adding CNOTs in Step 4 transfers those gates to other outputs as well. To remove this unwanted effect, also add CNOTs at the beginning of the *gate-list*$_k$. Note that insertion of this gate may cancel out another CNOT in the *total-gate-list*. If so, remove both of these gates.

Step 6: Append the *gate-list*$_k$ at the end of *total-gate-list*.

Step 7: Generate one Toffoli gate for each cube in *ungrouped-list* and append the gates to *total-gate-list*.

Example 4.2.1. An ESOP *cube-list* of six cubes with four input variables (x_1, x_2, x_3, and x_4) and five output variables (f_1, f_2, f_3, f_4, and f_5) is shown in Figure 4.3(a). The cubes are labeled C_1 to C_6. Among all the cubes only C_1 is ungrouped since the number of 1s in its output portion is 1 and so it is therefore separated from the *cube-list*. The resultant lists are shown in Figure 4.3(b). Now in the modified *cube-list*, C_3 has the highest number of 1s in its output part. Thus it is moved to the *sub-list*$_1$. Note that C_3 is not shared by f_5. Now from the remaining cubes (C_2, C_4, C_5, and C_6), a cube is selected whose output portion contains the highest number of 1s and which is not shared by output f_5 since f_5 does not contain C_3. Although C_2, C_4, C_5, and C_6 have the same number of 1s in their output parts, C_4 and C_6 are not allowed to move in this iteration since they are shared by f_5. Between the cubes C_2 and C_5, suppose that C_2 has been selected. C_2 along with C_5 is moved to the end of *sub-list*$_1$ since the output patterns of these two cubes are identical. There are no other cubes which

can be moved to *sub-list*$_1$. Figure 4.3(c) shows the cubes in *sub-list*$_1$ and *cube-list*.

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_1:	1	0	-	0	0	0	0	1	0
C_2:	1	1	-	-	1	1	1	0	0
C_3:	1	1	1	-	1	1	1	1	0
C_4:	1	0	0	1	0	1	0	1	1
C_5:	1	-	1	1	1	1	1	0	0
C_6:	1	-	0	-	1	0	1	0	1

Cube-list

(a) An initial cube-list.

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_1:	1	0	-	0	0	0	0	1	0

Ungrouped-list

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_2:	1	1	-	-	1	1	1	0	0
C_3:	1	1	1	-	1	1	1	1	0
C_4:	1	0	0	1	0	1	0	1	1
C_5:	1	-	1	1	1	1	1	0	0
C_6:	1	-	0	-	1	0	1	0	1

Current Cube-list

(b) Separation of ungrouped cubes from the cube-list.

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_3:	1	1	1	-	1	1	1	1	0
C_2:	1	1	-	-	1	1	1	0	0
C_5:	1	-	1	1	1	1	1	0	0

Sub-list$_1$

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_4:	1	0	0	1	0	1	0	1	1
C_6:	1	-	0	-	1	0	1	0	1

Current Cube-list

(c) Generation of sub-list$_1$.

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_4:	1	0	0	1	0	1	0	1	1

Sub-list$_2$

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_6:	1	-	0	-	1	0	1	0	1

Current Cube-list

(d) Generation of sub-list$_2$.

	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	f_5
C_6:	1	-	0	-	1	0	1	0	1

Sub-list$_3$

(e) Generation of sub-list$_3$.

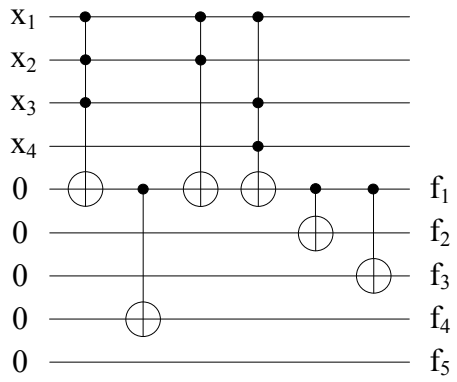
Figure 4.3: Cube-list and its sub-lists.

Now the current *cube-list* consists of C_4 and C_6. Both cubes have the same number of output ones. Consider that C_4 is chosen and moved to *sub-list*$_2$. We see that f_1 contains C_6 but not C_4. As a result, C_6 is not allowed to make a group with C_4. Figure 4.3(d) shows the *sub-list*$_2$. Now only one cube C_6 is remaining in the *cube-list*; thus in the next iteration moving this cube to *sub-list*$_3$ shown in Figure 4.3(e) completes the phase 1.

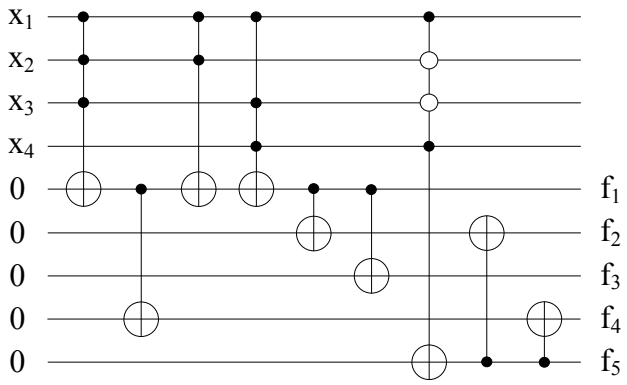
In phase 2, we transform the three sub-lists and ungrouped-list into a cascade of Toffoli gates. For *sub-list*$_1$ shown in Figure 4.3(c), outputs f_1, f_2, and f_3 have all the cubes in this list. Moreover, there are no gates on any of these output lines. Consequently, any of these lines can be used as a target line. Let f_1 be chosen as the target line. A Toffoli gate for C_3 targeting at f_1 is generated. Since f_4 does not share any cube other than C_3 in *sub-list*$_1$, one CNOT is required to transfer C_3 from f_1 to f_4. Next two Toffoli gates are generated for C_2 and C_5. Again, to transfer all the gates from f_1 to f_2 and f_3, two CNOTs are added. The Toffoli cascade for *sub-list*$_1$ is shown in Figure 4.4(a).

The *sub-list*$_2$, shown in Figure 4.3(d), has just one cube C_4 which is shared by f_2, f_4, and f_5. From Figure 4.4(a) we see that f_2 and f_4 are already occupied by other gates. Since the line f_5 is empty, this is chosen as the target line for *sub-list*$_2$. One negative-control Toffoli gate is added for C_4, which is transferred to f_2 and f_4 via CNOTs. Gates generated for this list are appended at the end of the cascade in Figure 4.4(a), which results in the circuit shown in Figure 4.4(b).

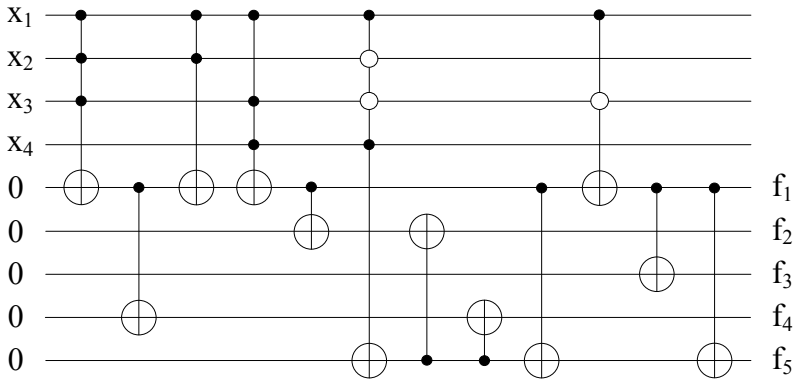
Next we consider the *sub-list*$_3$ in Figure 4.3(e), which contains one cube C_6. Outputs f_1, f_3, and f_5 share C_6, and the corresponding output lines are not empty (see Figure 4.4(b)). Let the target line be f_1. Since f_1 has gates on it, in order to eliminate the effect of these unexpected gates while transferring C_6 to f_3 and f_5, two more CNOTs are needed before generating the Toffoli gate for C_6. However, adding a new CNOT from f_1 to f_3 will cancel out the previously inserted CNOT (from f_1 to f_3) in the circuit shown in Figure 4.4(b). Therefore, this CNOT is removed rather than adding a new one (from f_1 to f_3). However a CNOT from f_1 to f_5 is required. Afterwards one negative-control Toffoli gate for C_6 and two CNOTs for sharing with f_3 and f_5 are added as shown in Figure 4.4(c). Finally, the ungrouped cube C_1 shown in Figure 4.3(b) is transformed directly into a negative-control Toffoli gate. The final cascade is shown in Figure 4.4(d).



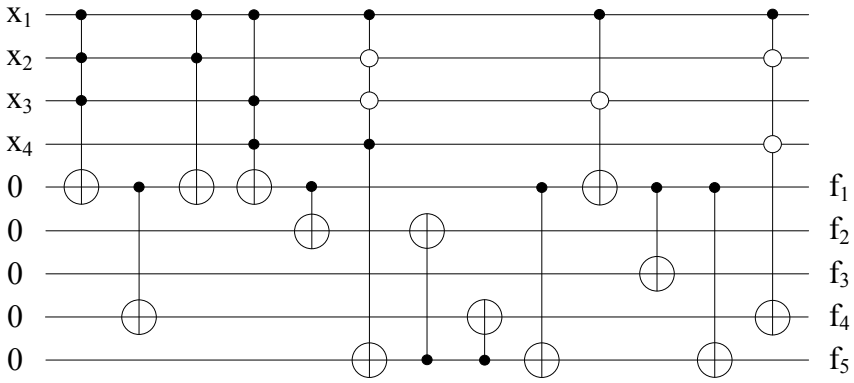
(a) A circuit equivalent to sub-list$_1$.



(b) Appending the circuit for sub-list$_2$ to the circuit of Figure 4.4(a).



(c) Appending the circuit for sub-list$_3$ to the circuit of Figure 4.4(b).



(d) A circuit equivalent to the cube-list in Figure 4.3(a).

Figure 4.4: Improved shared cube synthesis process.

4.3 Experimental Results and Discussions

Our proposed approach and the existing approach to shared cube synthesis discussed in [47, 46] have been developed in Java. The existing approach first reported in [47] does not use negative-control Toffoli gates; however, the usage of this type of gates was later suggested in [46] and [48]. Since our approach uses Toffoli gates including negative-control Toffoli gates, for fair comparison, negative-control Toffoli gates have also been incorporated to the circuits generated by the existing approach. The tool EXORCISM-4 [35] is used to generate the ESOP cube-lists for the benchmark circuits. The implemented programs have been run on a 2.4GHz Intel core 2 duo based system with 4GB RAM for 38 benchmark circuits collected from [58]. The execution time is negligible for both programs, and the results of this experiment are compared in Table 4.1.

In Table 4.1 GC and QC stand for gate count and quantum cost, respectively. In the first column, the name of the function is given. Columns two-three and four-five show the gate count and quantum cost of the circuits generated by the existing approach and the proposed approach, respectively. The last two columns indicate the improvements in percentage of the proposed approach over the previous one. Negative values indicate that the previous approach is better than the proposed one for that function.

It can be seen from the Table 4.1 that the proposed approach reduces the quantum cost for all functions except two functions 9symml and cordic. Circuits apex4, bw, ex5p, and seq are improved by more than 70% in terms of quantum cost. Moreover, a significant improvement is noticed for the functions cm42a, dc1, ham7, hwb8, in0, inc, misex1, pdc, and urf3. It is noted that improvements for 9symml and cordic are 0%. This is due to the fact that the function 9symml contains only one output; thus there is no shared functionality. For the function cordic, both approaches synthesize similar circuits due to only two outputs in the function, resulting in 0% improvement.

While our approach is far better in terms of quantum cost, the opposite trend is found

in the improvement column of gate count in Table 4.1. In our approach, 23 out of 38 circuits require more gates, and an increase of nearly 88% is noticed for apex4, which is the worst case. Since sizes of Toffoli gates used in the two approaches are different, we cannot expect an accurate measurement from the gate count comparisons as described in 2.5.1. Nevertheless, we have investigated to find the reason. Since our approach utilizes the shared functionality much better than the previous approach, many large Toffoli gates are replaced by CNOT gates. However, this replacement requires some extra CNOT gates to transfer the gates to other lines and to remove the impact of other gates when the output lines are not empty. This is clarified in Example 4.3.1.

The number of garbage outputs is another cost metric often used for evaluation of reversible circuits; however we note that both approaches perform exactly the same in this regard.

Example 4.3.1. Given a cube-list in Figure 4.5(a), circuits generated by the existing shared cube synthesis approach and our proposed approach are shown in Figure 4.5(b) and Figure 4.5(c), respectively. The former approach requires five Toffoli gates and two CNOT gates, whereas the latter approach reduces one Toffoli gate but adds three extra CNOT gates which are labeled as c_1, c_2, and c_3. CNOT gates c_1 and c_2 are added to transfer the first Toffoli gate in Figure 4.5(c) to lines f_2 and f_4. Again, c_3 is added since line f_2 is not empty when last Toffoli gate is required to transfer from f_2 to f_4. As expected, the latter approach reduces the quantum cost from 51 to 41.

4.4 Summary

In this chapter, we have proposed an improved shared cube synthesis approach which incorporates an efficient way to group the cubes even though some cubes are not shared by all outputs, resulting in better transformation of cubes into gates. A technique to elimi-

Table 4.1: Experimental results.

Circuit	Previous Approach		Our Approach		Improvement %	
	GC	QC	GC	QC	GC	QC
5xp1	57	901	58	786	-1.75	12.76
9symml	52	10943	52	10943	0	0
alu4	474	43265	454	41127	4.22	4.94
apex4	2988	134330	5622	35840	-88.15	73.32
apex5	593	39245	601	33830	-1.35	13.8
apla	60	2345	72	1683	-20	28.23
bw	194	2379	287	637	-47.94	73.22
cordic	777	187620	777	187620	0	0
C7552	64	1023	89	399	-39.06	61
clip	87	4908	78	3824	10.34	22.09
cm42a	32	266	42	161	-31.25	39.47
cu	25	864	28	781	-12	9.61
dc1	30	213	31	127	-3.33	40.38
dc2	51	1341	51	1084	0	19.16
decod	64	1023	89	399	-39.06	61
dist	110	5079	94	3700	14.55	27.15
dk17	34	1075	34	1014	0	5.67
ex1010	1487	105579	1675	52788	-12.64	50
ex5p	428	13875	646	3547	-50.93	74.44
f2	15	175	14	112	6.67	36
f51m	332	28835	327	28382	1.51	1.57
frg2	1435	127447	1389	112008	3.21	12.11
ham7	28	108	37	67	-32.14	37.96
hwb8	462	14431	480	8195	-3.9	43.21
in0	207	13156	245	7949	-18.36	39.58
inc	62	1425	75	892	-20.97	37.4
misex1	31	586	42	332	-35.48	43.34
misex3c	849	72735	822	49720	3.18	31.64
misex3	848	73867	854	49076	-0.71	33.56
mlp4	82	2744	80	2496	2.44	9.04
pdc	542	55887	649	30962	-19.74	44.6
root	52	2436	48	1811	7.69	25.66
sao2	42	5116	41	3767	2.38	26.37
seq	1189	139826	1287	33991	-8.24	75.69
sqr6	56	708	54	583	3.57	17.66
urf3	1464	89053	1501	53157	-2.53	40.31
wim	21	177	23	139	-9.52	21.47
z4ml	33	492	34	489	-3.03	0.61

$x_1x_2x_3x_4$	$f_1f_2f_3f_4$
1 1 - 1	1 1 1 1
1 - 1 -	1 0 1 0
- 1 1 -	1 0 1 0
1 - 1 1	0 1 0 1

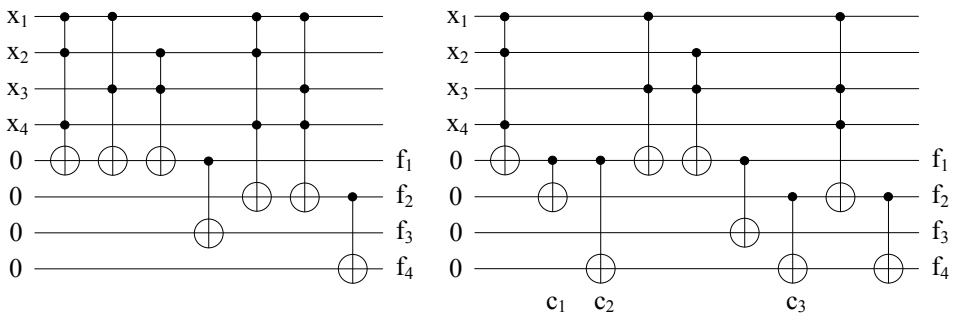


Figure 4.5: (a) An example cube-list, (b) a Toffoli cascade generated by the approach in [46, 47], and (c) a Toffoli cascade generated by our proposed approach.

nate redundant CNOT gates has also been added to the synthesis approach to reduce the number of CNOT gates. Like other ESOP-based approaches, this approach is also fast and able to synthesize very large functions. Experimental results show that our approach can reduce the quantum cost more than 70% for some circuits, as compared to the approach in [46, 47]. We note, however, that we do not have similar improvements when considering the gate count; further examination of this phenomenon suggests that this is due to the reduced complexity of gates that our method trends toward, as compared to the more complex gates used by the previous method.

Chapter 5

Testing of Reversible Circuits

Testing is required to ensure quality, availability, and reliability of a circuit or device. According to [56], testing can be performed online or offline, or a combination of both. Online testing is carried out when the system does its normal operation. In other words, testing and normal operation are performed simultaneously; therefore, faults are detected in real time. Offline testing is performed while the system is not in its normal operation. Since the whole system or a part of the system is taken to run the test, offline testing is usually performed when the load is low. In most cases, once a fault is detected by an online testing, the offline testing is applied to localize the fault in the system, reducing the repair time. Offline testing is also used for verifying the repaired part of the system before the whole system is made available for normal operation. In this chapter, we describe the approaches currently available for offline and online testing of reversible circuits.

5.1 Offline Testing

In offline testing, an input vector that is applied to the circuit for testing is referred to as a test vector. A set of all such vectors is called a test set. A test set is complete if it can detect all faults for a particular fault model. Research work found in the literature on offline testing of reversible logic includes test set generation and DFT methods. We briefly summarize these works for various faults models in the following two subsections.

5.1.1 Testing of Stuck-at Faults

According to Patel *et al.* [42], two important features of reversibility - controllability and observability - simplify the testing of reversible logic compared to traditional irreversible logic. It has been proved that any complete test set for the single stuck-at fault model is also complete for the multiple stuck-at faults model. However, finding a minimum complete test set is a NP-hard problem [51]. In [42], the problem of generating a minimal test set was mapped into an integer linear program (ILP) with binary decision variables. The ILP can generate the minimal test sets for small circuits; however, for large circuits, it becomes infeasible due to complexity. Thus a heuristic approach was proposed which partitions the large circuit into smaller circuits and applies the ILP formulation on these smaller circuits.

Chakraborty [5] showed that a circuit consisting of only n-bit Toffoli gates ($n > 2$) has a complete test containing p test vectors for detecting multiple stuck-at faults, where p is the number of lines in the circuit. This result can be extended for all Toffoli circuits with $n > 0$ by realizing NOT gates and CNOT gates from 3-bit Toffoli gates. Besides, a method was also proposed which modifies the circuit by introducing an extra line to reduce the number of test vectors in a complete test set to 3.

Ibrahim *et al.* [13] proposed a DFT method which adds up to two extra lines to the Toffoli circuit and replaces some Toffoli gates by larger ones with additional controls connected to extra lines. The complete test for the modified circuit under the multiple stuck-at faults model consists of only 2 test vectors, which is minimal.

5.1.2 Testing of Missing Gate, Repeated Gate, and Reduced Gate Faults

Hayes *et al.* [12] showed that unlike the stuck-at fault model, a complete test set that detects all single missing gate faults in a Toffoli circuit cannot detect all multiple missing gate faults. A maximum of $N/2$ test vectors are required to test all single missing gate faults, where N is the number of gates in the Toffoli circuit. A DFT method described in [12] adds one extra line and several CNOT gates to make the circuit testable for all missing gate faults using a single test vector. Rahaman *et al.* [44] proposed another DFT technique which uses one extra line and duplication of Toffoli gates such that the resulting circuit can detect all single missing gate faults, repeated gate faults, and reduced gate faults using a test set of $n+1$ vectors, where n is the number of lines in the given circuit.

5.2 Online Testing

This section discusses a number of different approaches to generate online testable reversible circuits and explores the issues regarding their designs on fault detection.

5.2.1 Testable Circuit Design Using R1, R2, and R Gates

Vasudevan *et al.* [54, 53] proposed a design methodology for constructing online testable reversible circuits. Three new reversible gates R1, R2 and R were introduced as shown in Figure 5.1. The R1 gate is used to realize NAND, OR, EXOR, and XNOR operations by setting different values on inputs. This gate has the parity output at q. The gate R2 passes the inputs through to the outputs, with a parity output again being computed at s, and in order to construct a testable block (TB), the gates R1 and R2 are cascaded by connecting

the first three outputs of R1 to the first three inputs of R2 creating a testable block with two parity outputs that can be compared to determine whether a single bit fault has occurred or not. Figure 5.2 shows the construction of a TB and its block diagram. The TBs are then used to realize the reversible circuit.

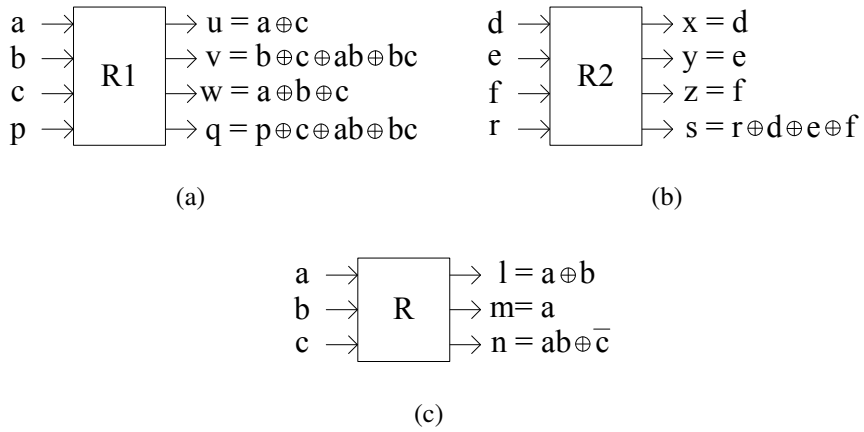


Figure 5.1: (a) R1 gate, (b) R2 gate, and (c) R gate.

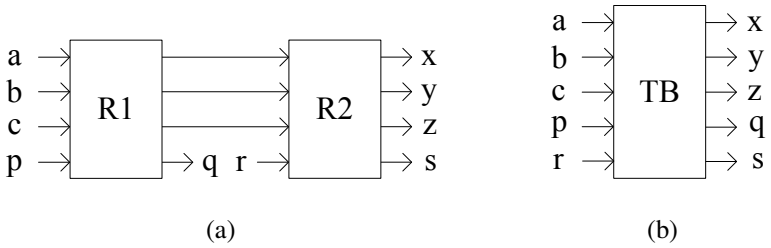


Figure 5.2: (a) Construction of a testable block (TB), and (b) its block diagram.

Since each TB generates two parity outputs, a two-pair two-rail checker circuit is also required to test the parities of two TBs. Let (q_1, s_1) and (q_2, s_2) be the parities of two TBs. The checker circuit takes these two pairs of parities as inputs and produces two outputs (e_1, e_2) as follows.

$$e_1 = q_1 s_2 + s_1 q_2$$

$$e_2 = q_1 q_2 + s_1 s_2$$

This checker circuit is built using eight R gates, and a block diagram is shown in Figure 5.3. If a circuit contains more than two TBs, a cascade of checker circuits is required.

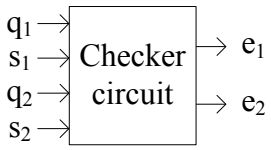


Figure 5.3: A two-pair two-rail checker circuit.

An online testable circuit for the function $f = ab + \bar{c}$ is implemented in Figure 5.4. The first TB realizes $\overline{ab}$ which is then fed along with input c to the second TB, producing the output $f = \overline{\overline{ab}c} = ab + \bar{c}$. The two parity outputs of each TB are connected to the checker circuit. By examining the outputs of checker circuit, the circuit detects a fault.

Analysis

Our investigation reveals that this approach cannot detect all single bit faults. If a fault occurs between two TBs, the circuit cannot detect it. Note that TBs generate parities which are tested by checker circuits; thus a fault in TBs is detected. Occurrence of any fault outside the TBs is left undetected. For example, in the circuit given in Figure 5.4, if a fault occurs at the first input of the second TB, the circuit is unable to detect it. As a result, this approach can detect some single bit faults, more specifically faults that occur in TBs, but cannot detect all faults.

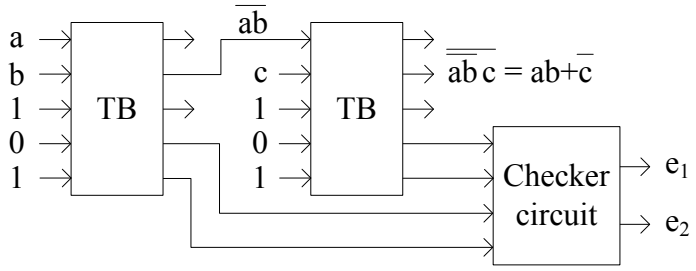


Figure 5.4: Online testable circuit for $f = ab + \bar{c}$, according to the design in [54, 53].

5.2.2 *Testable Circuit Design Using Testable Reversible Cells (TRCs)*

Mahammad *et al.* [21, 22] proposed an extension of the previous approach (Section 5.2.1), which can easily construct an online testable circuit from a given circuit. The conversion involves two steps as follows. The first step transforms each $n \times n$ reversible gate G used in the circuit into an $(n+1)\times(n+1)$ deduced reversible gate DRG(G). Given the input vector $[k_1, k_2, \ldots, k_n]$ and output vector $[o_1, o_2, \ldots, o_n]$ of an $n \times n$ gate as shown in Figure 5.5(a), an extra input p_{iG} and the corresponding output p_{oG} are added to construct an $(n+1)\times(n+1)$ DRG(G) as shown in Figure 5.5(b), which maps the input vector $[k_1, k_2, \ldots, k_n, p_{iG}]$ to the output vector $[o_1, o_2, \ldots, o_n, p_{oG}]$ where $p_{oG} = o_1 \oplus o_2 \oplus \ldots \oplus o_n \oplus p_{iG}$.

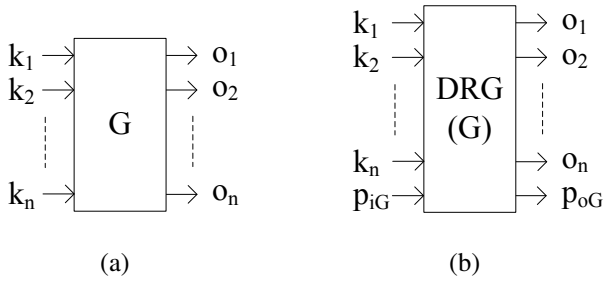


Figure 5.5: (a) G gate and (b) DRG(G).

The second step constructs a testable reversible cell (TRC) of G, denoted by TRC(G). Consider another $n \times n$ gate X which has the same input and output vectors. In other words,

inputs of X pass through to the outputs without any change. Like the previous step, DRG(X) is constructed by adding an input p_{iX} and the corresponding output p_{oX}. DRG(G) and DRG(X) are cascaded by connecting the first n outputs of DRG(G) to the first n inputs of DRG(X) in order to form an $(n+1)\times(n+1)$ TRC(G) with two parity outputs p_{oG} and p_{oX} as shown in Figure 5.6. Given $p_{iG} = p_{iX}$, p_{oG} and p_{oX} are complementary only if TRC(G) is faulty.

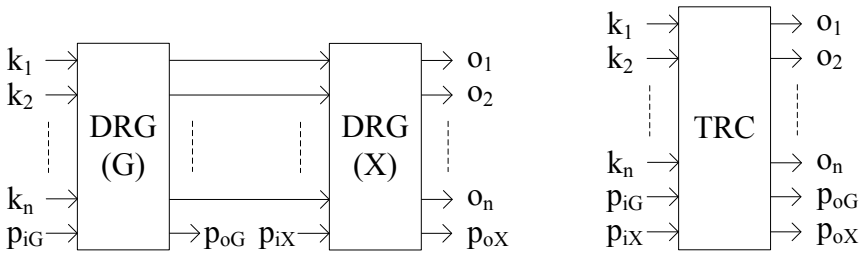


Figure 5.6: (a) Construction of a testable reversible cell TRC(G), and (b) its block diagram.

Afterwards, each gate in the circuit is replaced by its TRC. Assume there are m TRCs in the circuit. Let p_{oGj} and p_{oXj} be the parity outputs of the j^{th} TRC. To test all the parity outputs, a $(2m+1)\times(2m+1)$ test cell (TC) is formed. The first $2m$ inputs of TC are the parity outputs which pass through to the outputs. The last input is e which is set to 0 or 1, and the corresponding output is $T = ((p_{oG1}\oplus p_{oX1}) + (p_{oG2}\oplus p_{oX2}) + \ldots + (p_{oGm}\oplus p_{oXm})) \oplus e$. The block diagram of TC is given in Figure 5.7. According to the design, if a single bit fault occurs in the circuit then T becomes 1, provided $e = 0$. The conversion procedure is illustrated by the following example.

Consider a reversible circuit which computes $f = ab\oplus a\oplus c$ and is realized by a 3-bit Toffoli gate and a 2-bit Toffoli gate as shown in Figure 5.8(a). The 3-bit Toffoli gate and 2-bit Toffoli gate are replaced by a 5×5 TRC and 4×4 TRC. The parity outputs of these two TRCs are connected to a 5×5 TC to form a testable circuit as shown in Figure 5.8(b).

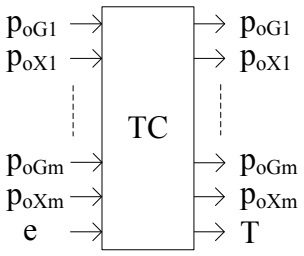


Figure 5.7: Test cell (TC).

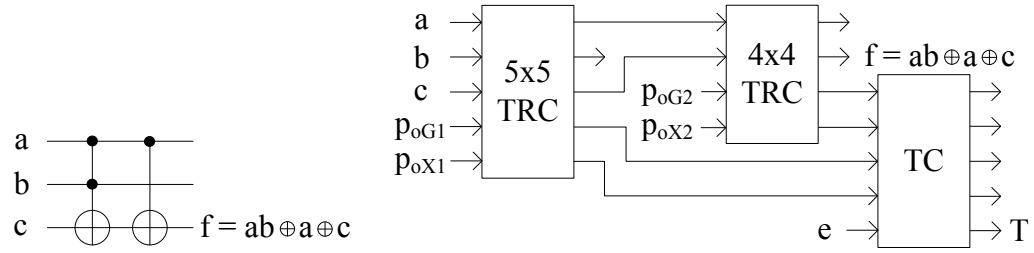


Figure 5.8: (a) A Toffoli circuit for $f = ab \oplus a \oplus c$, and (b) the corresponding testable circuit according to the design in [21, 22].

Analysis

We point out that the design flaw that we described in the previous approach also exists in this design. For example, any fault between the connections of two TRCs in Figure 5.8(b) is undetectable. Thus this approach fails to detect a fault between two TRCs. Mahammad *et al.* [21] claimed that

> *The resultant testable circuit can detect online any single-bit errors ...*

We have already showed that this approach cannot detect all single-bit errors. In [22], Mahammad and Veezhinathan asserted that

> *This paper has also proposed a methodology that automatically converts any circuit into an online testable reversible circuit with theoretically proved minimum garbage.*

In Chapter 6, we propose optimized approaches that can minimize the garbage, to an extent which is significantly smaller than their theoretically proved minimum garbage.

5.2.3 *Testable Circuit Design Using Online Testable Gates (OTGs)*

Thapliyal and Vinod [52] proposed an approach similar to the one described in Section 5.2.1. A new 4×4 reversible gate, online testable gate (OTG) introduced in their work has a parity output at q as shown in Figure 5.9. The R2 gate (see Figure 5.1(b)) is combined with the OTG as shown in Figure 5.10 to design a block with online testability feature. Two parity outputs s and q of this testable block are compared to check whether the block is faulty or not. In [54, 53] (see Section 5.2.1), the two-pair two-rail checker circuit was designed using eight R gates, whereas Thapliyal and Vinod [52] suggested a design using four 3-bit Toffoli gates and two 3-bit Fredkin gates.

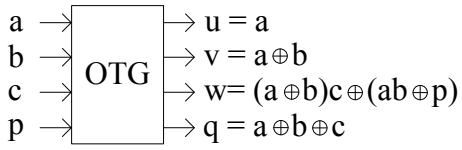


Figure 5.9: Online testable gate (OTG).

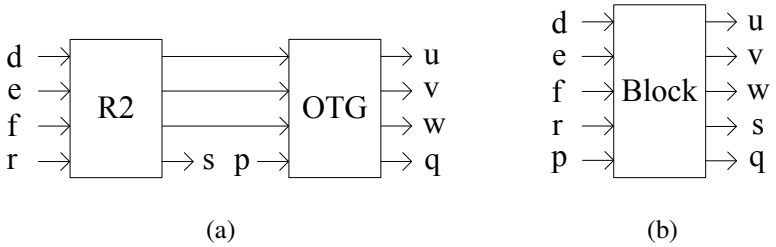


Figure 5.10: (a) Construction of a testable block, and (b) its block diagram.

As an example, this approach is applied to implement a testable circuit for the function

$g = ab \oplus c$, as shown in Figure 5.11. The first block realizes ab which is then fed along with input c to the second block, producing the output $g = ab \oplus c$. The checker circuit compares the parity outputs of two blocks and detects a fault if any.

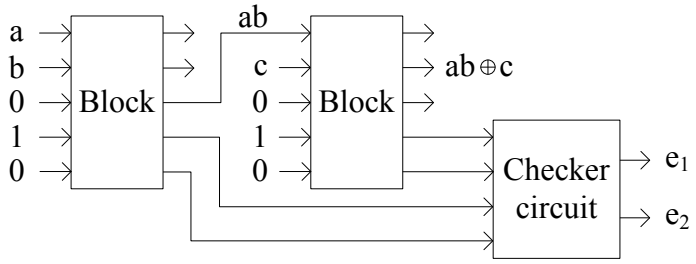


Figure 5.11: Online testable circuit for $g = ab \oplus c$, according to the design in [52].

Analysis

Like the previous two approaches, this approach also fails to detect a fault that occurs between two blocks. In our example circuit (see Figure 5.11), if a fault occurs between the connections of two blocks, it is not detectable.

5.2.4 Dual Rail Coding Approach

The dual rail coding approach proposed by Farazmand *et al.* [8] uses a set of 4×4 dual rail reversible gates for online fault detection. Each dual rail gate has two pairs of inputs, and two inputs of each pair are given in dual rail form, *i.e.*, the two inputs are the complement of each other (either 01 or 10). If the outputs appear in dual rail form, then there is no error. However, a non-dual rail form (either 11 or 00) represents a single fault. These dual rail gates are cascaded to generate the testable circuit. A fault in the circuit propagates to the end of the circuit. Thus the fault is detected by checking the outputs of the circuit. As a

result, no checker circuit is required to test the intermediate gates.

5.2.5 *Testable Circuit Design with Duplication of Gates*

Kole *et al.* [18] proposed a technique that can detect online any single missing gate faults in a circuit consisting of only Toffoli gates. This technique requires one extra line T which is tested to detect the fault. For each n-bit Toffoli gate in the circuit, three extra Toffoli gates, two CNOT gates and another n-bit Toffoli gate, are embedded to construct a testable circuit. Figure 5.12 shows how a Toffoli gate t_1 is surrounded by three Toffoli gates c_1, c_2, and t_2 to make it testable. According to the design, if the initial value and the final value of the line T are the same, then no gate is missing; otherwise the absence of a gate is assumed.

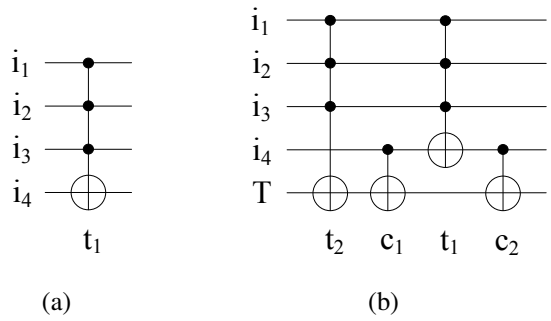


Figure 5.12: (a) A Toffoli gate t_1 and (b) its corresponding testable circuit.

Analysis

The testable circuit implemented in this way requires 4 times as many gates as the non-testable design requires. The complexity of this technique in terms of quantum cost is $2g + 2q$, where g and q are the gate count and quantum cost of the non-testable design, respectively. Thus the quantum cost of this design is more than twice the cost of the non-testable design.

5.3 Summary

This chapter has outlined the existing offline testing methods for a number of fault models. We have reviewed several approaches of online testing and discussed their design issues and limitations on detecting faults.

Chapter 6

Optimized Approaches for Online Fault Detection

This chapter describes new work that forms the contributions of this thesis to the area of reversible logic testing. In Section 6.1, we present a simple way to convert an ESOP-based reversible circuit into an online testable circuit which is able to detect online any single-bit faults. Section 6.2 extends this approach for any type of Toffoli circuits. Appropriate lemmas are also given to prove that both approaches can detect online any single-bit faults in the circuit. Experimental results are also reported in each section to compare the designs of our proposed approaches to that of previously reported approaches.

6.1 Testing of ESOP-based Circuits

The basic ESOP-based approach [9] with line optimization (Section 3.2), cube ordering heuristics (Section 3.3) using the alpha/beta cost metric [9] and the autocorrelation coefficient based cost metric [45], and optimization by using negative-control Toffoli gates (Section 3.4) all generate circuits which have separated input and output lines. A common structure of this type of circuits is that controls of the Toffoli gates are connected only to input lines and targets are connected only to output lines. A similar structure is kept when the given ESOP-circuit is converted into a testable circuit, allowing an easy detection of single-bit faults in the resulting circuit.

Though the previous work on shared cube synthesis [47, 46] described in Section 3.5 and the improved shared cube synthesis which we proposed in Chapter 4 fall into the category of ESOP-based approach, the generated circuits do not have the aforementioned structure. In this section, we restrict our approach for online testability to circuits with that structure.

6.1.1 Construction of a Testable Circuit from the ESOP-based Circuit

Given an ESOP function with p inputs and q outputs, consider a reversible circuit which realizes this function and has the structure mentioned earlier. Thus the circuit has p input lines and q output lines. To convert the given ESOP-based circuit into an online testable circuit, we need to add some NOT gates, CNOT gates and a parity line L which is initialized by a 0. The procedure is as follows.

Every n-bit Toffoli gate in the given circuit is replaced by a (n+1)-bit ETG. The connections of the first n bits of the ETG remain the same as that of n-bit Toffoli gate. The last, *i.e.* $(n+1^{st})$ bit of the ETG is connected to L. The NOT gates in the given circuit are also kept. For each NOT gate in the input and output line, one extra NOT gate is added on the line L. In total m extra NOT gates are required in this step, where m is number of NOT gates in the given circuit. Afterwards, the CNOT gates are inserted from all the output lines to the L line, requiring q more gates. In order to test the input lines, we add CNOT gates from each of the input lines to L before and after the whole circuit. This step requires $2p$ CNOT gates. Now in the resulting circuit, if a single fault occurs in any of the input lines, output lines or even in L, the value of L will be changed to 1. If no fault occurs, L will remain 0. It is important to note that this technique can also be applied for the ESOP-based circuit consisting of inverted-control Toffoli gates. This process of converting a circuit can be best described by the following example.

Example 6.1.1. For a given 4-input (I_1, I_2, I_3, I_4), 2-output (I_5, I_6) ESOP-based circuit shown in Figure 6.1(a), the corresponding online testable circuit generated by the proposed technique is shown in Figure 6.1(b). We can see that Toffoli gates (t_1, t_2, t_3, and t_4) are replaced by ETGs (e_1, e_2, e_3, and e_4). CNOT gates c_1 through c_{10} are added to test the input and output lines.

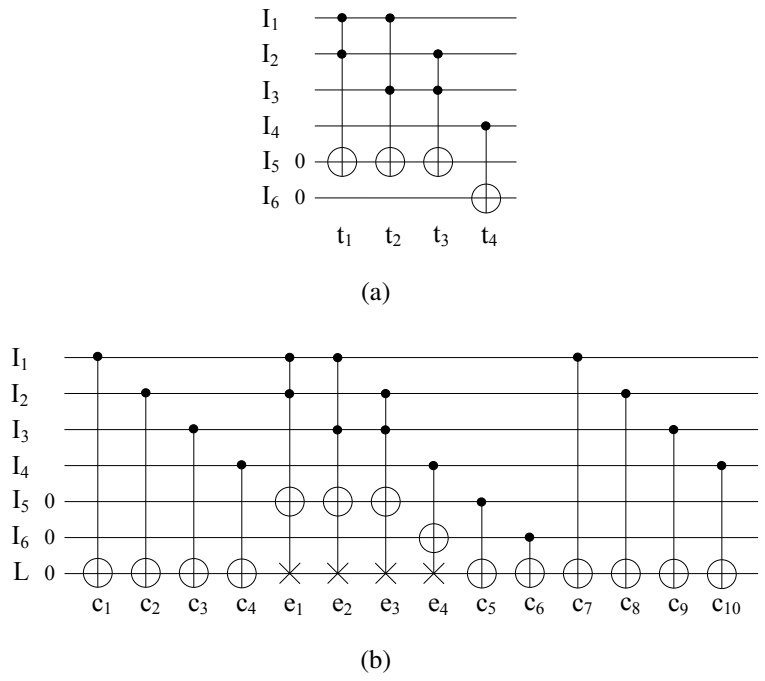


Figure 6.1: (a) An ESOP-based circuit, (b) Online testable reversible circuit.

6.1.2 Analysis

A fault in the target of an ETG certainly affects the gate as it changes the output of the target. However, a fault in control affects an ETG, only if it causes the target to have the faulty value. For instance, given that all controls of an ETG are positive, a fault in a control bit affects the gate if all fault-free control bits have the value 1. A single fault in the control of an ETG can propagate to target lines if the fault affects the gate. Thus a fault can cause multiple faults. In the proposed design, since controls of the ETGs and CNOTs are connected to input lines, faults on input lines can propagate to output lines and L by targets. However, faults on output lines and L cannot cause other lines to be faulty since no controls are connected to output lines or L. Our proposed approach can detect any single-bit fault even though it causes several faults. The following example shows how a single fault on a line can propagate to several lines.

Example 6.1.2. Consider a circuit consisting of two ETGs as shown in Figure 6.2. The lines I_1, I_2, I_3, and L are initialized by 0, 0, 1, and 0, respectively. Assume a fault occurs on I_1 just before the first ETG; thus the value of I_1 changes to 1. For the faulty lines, values are given in the form [fault-free value/ faulty value]. The fault on I_1 affects the first ETG and propagates to I_2 and L by the targets of this gate. As a result, two extra lines I_2 and L become faulty. The second ETG causes I_3 to have the faulty value and fixes the value of L. Therefore, after the second ETG, lines I_1, I_2 and I_3 are faulty.

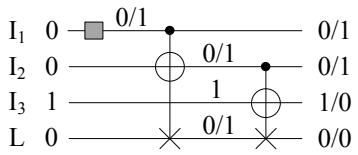


Figure 6.2: Fault propagation in multiple lines.

Lemma 6.1.1 proves that both targets of an ETG calculate the same function regardless of the occurrence of the fault. In this lemma, we consider all controls of the ETG to be positive. A similar lemma can be proved if some or all controls are negative. Lemma 6.1.2 proves the correctness of our technique. Two examples are also given to illustrate the propagation of a fault and the detection of such fault at the end of the circuit.

Lemma 6.1.1. *Consider an $(n+1)$-bit ETG which maps the input vector $[k_1, k_2, ..., k_{n-1}, k_n, k_{n+1}]$ to the output vector $[o_1, o_2, ..., o_{n-1}, o_n, o_{n+1}]$, where $o_j = k_j$ (for $j = 1, 2, ..., n-1$), $o_n = f \oplus k_n$, $o_{n+1} = f \oplus k_{n+1}$, and $f = k_1 k_2 \cdots k_{n-1}$.*

a) If faults occur in controls of the ETG and affect the gate, then both o_n and o_{n+1} compute $\overline{f}$; otherwise both outputs compute f.

b) If faults occur in targets of the ETG, then both o_n and o_{n+1} compute f.

c) If faults occur in targets and controls which affect the ETG, then both o_n and o_{n+1} compute $\overline{f}$; otherwise both outputs compute f. ◁

Proof. a) Consider faults in $k_i, k_j, \ldots, k_l$ such that $\{i, j, \ldots, l\} \subseteq \{1, 2, \ldots, n-1\}$. These faults affect the calculation of function f if the following two conditions hold.

i) $k_m = 1, \forall\, m \in \{1, 2, \ldots, n-1\} - \{i, j, \ldots, l\}$, and

ii) $k_i = k_j = \ldots = k_l = 0$ or $k_i = k_j = \ldots = k_l = 1$.

If both conditions are true, then the faults have impact on o_n and o_{n+1} since both outputs compute f. Thus $o_n = \overline{f} \oplus k_n$ and $o_{n+1} = \overline{f} \oplus k_{n+1}$. If at least one of the conditions is false, then the faults do not affect the calculation of f. Thus according to the definition, $o_n = f \oplus k_n$ and $o_{n+1} = f \oplus k_{n+1}$.

Therefore, if the faults have an effect, then o_n and o_{n+1} compute $\overline{f}$; otherwise these outputs compute f.

b) We consider three cases, depending on whether one of the targets is faulty or both targets are faulty.

Case 1. Assume a fault occurs in k_n. This fault does not propagate to o_{n+1} since o_{n+1} is independent of k_n; thus $o_{n+1} = f \oplus k_{n+1}$. However, due to the fault, $o_n = f \oplus \overline{k}_n$.

Case 2. Assume a fault occurs in k_{n+1}. The proof is similar to Case 1. We get $o_n = f \oplus k_n$ and $o_{n+1} = f \oplus \overline{k}_{n+1}$.

Case 3. Consider that faults occur in both k_n and k_{n+1}. Due to these faults, $o_n = f \oplus \overline{k}_n$ and $o_{n+1} = f \oplus \overline{k}_{n+1}$.

Hence, for any of the cases, both o_n and o_{n+1} compute f.

c) We consider two cases, depending on whether faults in controls affect the gate or not.

Case 1. First consider that faults in controls affect the gate. From (a), we can write $o_n = \overline{f} \oplus k_n$ and $o_{n+1} = \overline{f} \oplus k_{n+1}$.

Assume that faults also occur in both targets. According to (b), we can rewrite the outputs as follows: $o_n = \overline{f} \oplus \overline{k}_n$ and $o_{n+1} = \overline{f} \oplus \overline{k}_{n+1}$. Thus both outputs compute $\overline{f}$. Similarly, we can reach the same conclusion if a fault occurs in any of the targets.

Case 2. For the case that faults in controls have no effect on the gate, the proof is similar to (b).

□

Lemma 6.1.2. *If any single fault occurs on any line, the value of L changes to* 1 *and the fault is detected.* ◁

Proof. Consider an online testable circuit generated by the proposed technique which has N ETGs, p input lines, q output lines, and a parity line L. Let $G = \{g_1, g_2, ..., g_N\}$ be the set of ETGs used in the circuit, and let $I_1, I_2, ..., I_p$ be the input lines and $I_{p+1}, I_{p+2}, ..., I_{p+q}$ be the output lines. All output lines and L are initialized by 0. Given an (n+1)-bit ETG $g_i \in G$ and the input vector $[k_1, k_2, k_3, ..., k_{n-1}, k_n, k_{n+1}]$, the output vector is $[k_1, k_2, k_3, ..., k_{n-1}, fg_i \oplus k_n, fg_i \oplus k_{n+1}]$, where $fg_i = k_1k_2 \cdots k_{n-1}$, and n can be at most p+1. In order to prove this lemma, consider the following three cases:

Case 1. Assume that a single fault occurs on any input line, say I_z (for $z = 1, 2, ..., p$), which affects a set of gates, $X = \{x_1, x_2, ..., x_u\} \subseteq G$. Consider another set of gates, $Y = \{y_1, y_2, ..., y_v\}$ which is not affected by the fault. We have $G = X \cup Y$, and X or Y can be empty.

As described before, the last two outputs of a gate $g_i \in G$ computes the same function fg_i. A gate y_r in Y computes fy_r, for $r = 1, 2, ..., v$. However, from Lemma 6.1.1, due to the fault, each gate x in X computes $\overline{fx}_s$, for $s = 1, 2, ..., u$.

After the last gate in G, the line I_z has the faulty value $\bar{I}_z$, and L is $I_1 \oplus I_2 \oplus ... \oplus I_z \oplus ... \oplus I_p \oplus fy_1 \oplus fy_2 \oplus ... \oplus fy_v \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus ... \oplus \overline{fx}_u$. When all output lines are EXORed to L at the end, L becomes

$I_1 \oplus I_2 \oplus ... \oplus I_z \oplus ... \oplus I_p \oplus fy_1 \oplus fy_2 \oplus ... \oplus fy_v \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus ... \oplus \overline{fx}_u \oplus fy_1 \oplus fy_2 \oplus ... \oplus fy_v \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus ... \oplus \overline{fx}_u$

$= I_1 \oplus I_2 \oplus ... \oplus I_z \oplus ... \oplus I_p$

Finally, when all input lines are EXORed to L, the fault on I_z is propagated to L, and L becomes

$I_1 \oplus I_2 \oplus ... \oplus I_z \oplus ... \oplus I_p \oplus I_1 \oplus I_2 \oplus ... \oplus \bar{I}_z \oplus ... \oplus I_p$

$= I_z \oplus \bar{I}_z = 1.$

Since at the end, the line L contains 1, the circuit can detect the fault.

Case 2. Now consider that a single fault occurs on an output line I_{p+z} at any point, where $z = 1, 2, ..., q$. From Lemma 6.1.1, gates, which have connections with line I_{p+z} after the occurrence of the fault, have the faulty values in its next to last output, but these gates produce the fault free values on L. For example, a (n+1)-bit ETG will produce the faulty value in o_n due to the fault on I_{p+z} but fault free value in o_{n+1}.

Consider a set of gates, $X = \{x_1, x_2, ..., x_u\} \subseteq G$ which have connections with line I_{p+z} after the fault occurs. Consider another set of gates, $Y = \{y_1, y_2, ..., y_v\}$ such that $Y = G - X$. Each gate x_s in X computes fx_s, for $s = 1, 2, ..., u$. Similarly, each gate y_r in Y computes fy_r, for $r = 1, 2, ..., v$.

The fault on I_{p+z} propagates to the end of the line. Using Lemma 6.1.1, the value of the line I_{p+z} at the end is $fx_1 \oplus fx_2 \oplus ... \oplus fx_u \oplus 1$. As a result, when output and input lines are EXORed to L at the end, the faulty value of I_{p+z} appears at L. Thus L becomes

$I_1 \oplus I_2 \oplus ... \oplus I_p \oplus fx_1 \oplus fx_2 \oplus ... \oplus fx_u \oplus fy_1 \oplus fy_2 \oplus ... \oplus fy_v \oplus I_1 \oplus I_2 \oplus ... \oplus I_p \oplus fx_1 \oplus fx_2 \oplus ... \oplus fx_u \oplus fy_1 \oplus fy_2 \oplus ... \oplus fy_v \oplus 1 = 1$

Case 3. A fault can also occur on L. This causes L to have the value 1.

Hence, for any of these cases, the circuit detects the fault. □

The following two examples describe how this technique can detect a single fault.

Example 6.1.3. For a given online testable circuit with four input lines (I_1, I_2, I_3, I_4) and

two output lines (I_5, I_6) as shown in Figure 6.3, consider a single fault on input line I_2 just before the first ETG. For the faulty lines, output values are given in the form [fault-free value/ faulty value] after each gate. For other lines, only the fault free values are shown. As can be seen from the figure, the fault on I_2 causes another fault on output line I_5. However, the fault is detectable since the value of L changes to 1. This example illustrates that the circuit is able to detect a fault even though it propagates to multiple lines.

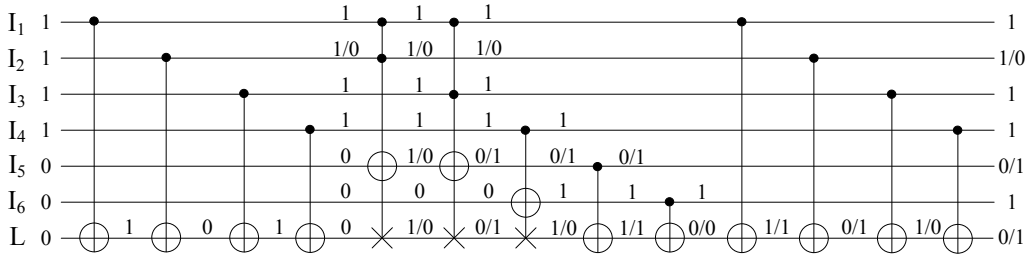


Figure 6.3: A single fault on input line I_2.

Example 6.1.4. For a given online testable circuit with four input lines (I_1, I_2, I_3, I_4) and two output lines (I_5, I_6) as shown in Figure 6.4, consider a single fault on output line I_5 between the first and second ETGs. Notice that the fault propagates to the end of I_5. Due to the fault, L becomes 1 and hence the fault is detected.

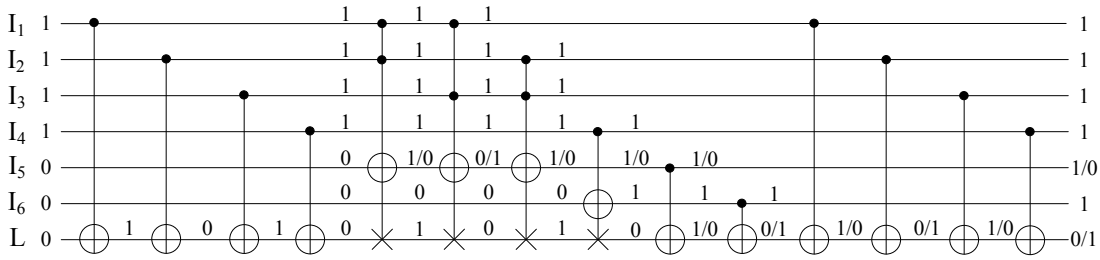


Figure 6.4: A single fault on output line I_5.

6.1.3 Experimental Results

A number of benchmark circuits collected from [58] have been implemented using our proposed approach and the earlier reported approaches [54, 53, 8, 21, 22]. Table 6.1 compares these approaches in terms of quantum cost and garbage outputs. In the table, columns QC and GO represent the quantum cost and the number of garbage outputs, respectively.

It can be seen from the table that our approach achieves a huge reduction in quantum cost and garbage outputs for every circuit. Our design produces 21.1 garbage outputs on average, whereas the best reported approach [21, 22] requires 799.9 (see Table 6.1). The average minimization of garbage amount is 99%, 98%, and 97% with respect to [54, 53], [8], and [21, 22], respectively.

We compute the improvement in quantum cost to be on average 50% (over [54, 53]), 58% (over [8]), and 22% (over [21, 22]).

6.1.4 Advantages of the Proposed Design

The proposed online testable reversible design has several advantages over the existing designs described in Section 5.2.

- The designs in [54, 53, 8] require a new synthesis approach to redesign the non-testable circuit in order to make it testable, whereas our technique is rather simple since it works on top of the ESOP-based circuit, adds some NOT and CNOT gates and replaces the Toffoli gates with ETGs.

- Unlike [54, 53, 21, 22], our design does not need any checker circuits. Though the approach in [8] does not need intermediate checker circuits, it does require a checker circuit for testing the final outputs.

- One important feature of our design is that gates added for testing a circuit do not

Table 6.1: Experimental results.

Circuit	Approach in [54, 53]		Approach in [8]		Approach in [21, 22]		Our approach	
	QC	GO	QC	GO	QC	GO	QC	GO
9symml	25598	5952	32220	532	12262	139	11065	9
apex5	195061	45362	213843	3518	69596	1557	53947	117
apla	17815	4142	20574	334	5457	185	3978	10
bw	19019	4422	25140	386	8757	620	4994	5
cm82a	2163	502	2823	50	370	45	176	5
con1	1647	382	1929	38	307	30	184	7
cu	5560	1292	6234	110	1834	82	1327	14
dk17	10376	2412	11478	186	2463	95	1791	10
ex2	1733	402	2241	42	267	23	171	5
ex5p	84852	19732	98898	1540	38740	1518	26970	8
f51m	128927	29982	164889	2670	42590	937	33165	14
frg2	218109	50722	247332	4048	264020	5073	205969	143
max46	15321	3562	19314	326	5585	114	4627	9
misex3	141354	32872	174732	2816	151363	2987	117215	14
rd73	16482	3832	20583	342	1862	140	1064	7
sqn	11236	2612	14256	240	3025	122	2209	7
sqrt8	5689	1322	7086	122	1056	76	657	8
sym9	24652	5732	31890	528	12262	139	11065	9
table3	223054	51872	265566	4294	112361	2019	87880	14
z4ml	3883	902	5070	88	1114	97	607	7
Average	57626.55	13400.5	68304.9	1110.5	36764.55	799.9	28453.05	21.1

produce any new garbage. Garbage outputs of our design come from the ESOP-based circuit, which are negligible when compared to the previous works.

6.2 Testing of Toffoli Circuits

A Toffoli circuit consists of only Toffoli gates including NOTs, CNOTs and negative-control Toffoli gates. The Toffoli circuits have no distinguished input or output lines; thus we use the term *line* in this section. The target and controls of a Toffoli gate can be connected to any line in the circuit. In Section 6.1, we have proposed an approach for constructing online testable reversible circuits from a certain type of ESOP-based circuits. In this section, we extend this approach so that it works for any type of Toffoli circuits.

6.2.1 Construction of a Testable Circuit from the Toffoli Circuit

Consider a Toffoli circuit consisting of p lines; in order to make such a circuit online testable, a parity line L is added, which is initialized with a 0. The detailed procedure is given below.

The procedure inserts CNOT gates from all lines to L at the beginning of the circuit. It then replaces every n-bit Toffoli gate by an (n+1)-bit ETG. The connections of the first n bits of the ETG are kept the same as that of n-bit Toffoli gate. The last bit of the ETG is connected to L. All NOT gates found on the lines are retained. If the number of NOT gates in the circuit is an odd number, an extra NOT gate is added at the end of line L; otherwise, no extra NOT gate is added. Finally, CNOT gates are added at the end from all lines to L.

This approach requires a total of $2p$ extra CNOT gates and at most one extra NOT gate to construct an online testable circuit. In the testable circuit, if a single fault occurs in any

line (including L), the value of L changes from 0 to 1. If no fault occurs, L remains 0. Thus by checking a single bit L at the end of the circuit, the fault is detected. Note that this procedure also works for any circuit consisting of inverted-control Toffoli gates.

Example 6.2.1. Consider a Toffoli circuit given in Figure 6.5(a) which has five lines (I_1, I_2, ..., I_5) and four Toffoli gates (t_1, t_2, t_3, and t_4). For constructing an online testable circuit, we add a parity line L as well as replace t_1, t_2, t_3, and t_4 by ETGs e_1, e_2, e_3, and e_4. We also insert five CNOT gates c_1 through c_5 at the beginning and five CNOT gates c_6 through c_{10} at the end of circuit. The resultant online testable circuit is shown in Figure 6.5(b).

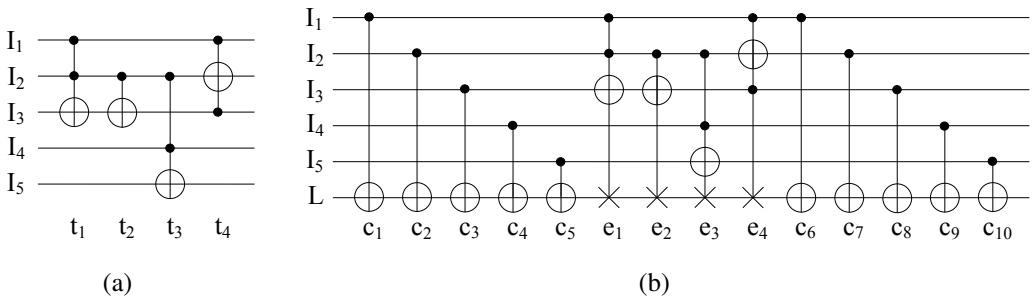


Figure 6.5: (a) A Toffoli circuit and (b) an online testable circuit.

6.2.2 Analysis

As described in Section 6.1.2, a single fault can generate multiple faults. In a testable circuit, a fault on any line (except parity line L) can cause any other lines including L to become faulty. It is important to note that a fault on the parity line L cannot propagate to other lines since controls of the CNOTs and ETGs are not connected to L. Lemma 6.2.1 proves that the proposed approach detects any single fault even if it propagates to other lines, causing multiple faults.

Lemma 6.2.1. *If any single fault occurs on any line, the circuit is able to detect it.* ◁

Proof. Consider an online testable circuit generated by the proposed technique which has N ETGs, p lines (I_1, I_2, ..., I_p), and a parity line L. Let $G = \{g_1, g_2, ..., g_N\}$ be the set of ETGs used in the circuit. Let the initial values of lines I_1, I_2, ..., I_p be i_1, i_2, ..., i_p. The line L is initialized with a 0. Given an (n+1)-bit ETG $g \in G$ and the input vector [k_1, k_2, k_3, ..., k_{n-1}, k_n, k_{n+1}], the output vector is [k_1, k_2, k_3, ..., k_{n-1}, $fg \oplus k_n$, $fg \oplus k_{n+1}$], where $fg = k_1k_2 \cdots k_{n-1}$, and n can be at most p. In order to prove this lemma, consider the following two cases.

Case 1. Assume that a single fault occurs on a line I_d (for d = 1, 2, ..., p) and propagates to multiple lines.

Let $W = \{w_1, w_2, ..., w_q\} \subseteq$ G be the set of gates that are not affected by the faults.

Let $X = \{x_1, x_2, ..., x_r\} \subseteq$ G be the set of gates such that faults occur only on controls and affect the gates.

Let $Y = \{y_1, y_2, ..., y_s\} \subseteq$ G be the set of gates with faults only on targets.

Let $Z = \{z_1, z_2, ..., z_t\} \subseteq$ G be the set of gates such that faults occur on targets and controls which affect the gates.

We have $G = W \cup X \cup Y \cup Z$, and W, X, Y or Z can be empty. According to the definition of the ETG, two targets of a gate $g \in G$ compute the same function fg. Similarly, a gate $w_j \in W$ computes fw_j for j = 1, 2, ..., q. A gate $x_l \in X$ computes $\overline{fx_l}$ for l = 1, 2, ..., r according to Lemma 6.1.1(a). A gate $y_u \in Y$ computes fy_u for u = 1, 2, ..., s according to Lemma 6.1.1(b). A gate $z_v \in Z$ computes $\overline{fz_v}$ for v = 1, 2, ..., t according to Lemma 6.1.1(c).

At the beginning of the circuit, all I_m lines, for m = 1, 2, ..., p are EXORed to parity line L. Thus the value of L, just before the first gate in G, is $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p$. After the last gate in G, the value of L becomes $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p \oplus fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus \ldots \oplus \overline{fx}_r \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz}_1 \oplus \overline{fz}_2 \oplus \ldots \oplus \overline{fz}_t$.

At the end, when all I_m lines, for m = 1, 2, ..., p are EXORed to L again, the faulty value of I_d (which is $\bar{i}_d$) propagates to L and hence L becomes $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p \oplus$

$fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus \ldots \oplus \overline{fx}_r \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz}_1 \oplus \overline{fz}_2 \oplus \ldots \oplus \overline{fz}_t \oplus i_1 \oplus i_2 \oplus \ldots \oplus \bar{i}_d \oplus \ldots \oplus i_p \oplus fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx}_1 \oplus \overline{fx}_2 \oplus \ldots \oplus \overline{fx}_r \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz}_1 \oplus \overline{fz}_2 \oplus \ldots \oplus \overline{fz}_t$

$= i_d \oplus \bar{i}_d = 1.$

Since at the end, the line L contains 1, the fault is detected.

Case 2. A fault can also occur on L. This causes L to have the value 1 since it was initialized with a 0. Hence, the circuit detects the fault. Note that a fault on I_d can cause other lines faulty (Case 1). However, a fault on L does not propagate to any I_d since controls of ETGs and CNOTs are not connected to L. □

Example 6.2.2. Consider an online testable circuit given in Figure 6.6. In the circuit, ETGs are labeled as e_1, e_2, e_3, and e_4. The initial values of lines I_1, I_2, I_3, I_4, and I_5 are i_1, i_2, i_3, i_4, and i_5. The parity line L is initialized with a 0. Assume that ETGs e_1, e_2, e_3, and e_4 compute f_1, f_2, f_3, and f_4 when the circuit is fault free. Before the first ETG e_1, the value of L is $i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5$. Let, $l = i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5$.

The ETG e_1 computes $f_1 = i_1 i_2$. Thus after this gate, the values of I_3 and L are $i_3 \oplus f_1$ and $l \oplus f_1$. Assume a fault occurs on I_2 between e_1 and e_2, which changes the value of I_2 from i_2 to $\bar{i}_2$. This fault affects e_2; thus e_2 computes $\overline{f}_2 = \bar{i}_2$. Note that the fault-free values of I_3 and L, after the gate e_2, are $i_3 \oplus f_1 \oplus f_2$ and $l \oplus f_1 \oplus f_2$, respectively. However, due to the fault, the values are $i_3 \oplus f_1 \oplus \overline{f}_2$ and $l \oplus f_1 \oplus \overline{f}_2$. Thus two targets of e_2 cause the lines I_3 and L to have the faulty values.

Now consider the ETG e_3. One of the controls connected to I_2 and target connected to L are faulty. Let, i_4 be 0. Thus the fault on I_2 does not affect e_3 since other fault-free control bit, connected to I_4, has the value 0. The fault on L does not have any impact on calculating the function $f_3 = \bar{i}_2 i_4 = \bar{i}_2 . 0 = 0$. After this gate, the value of I_5 is $i_5 \oplus f_3$, and the value of L is $l \oplus f_1 \oplus \overline{f}_2 \oplus f_3$.

The ETG e_4 has the control on I_3 and targets on I_2 and L, which are faulty. Assume that

other fault-free control bit, connected to I_1, has the value 1. Thus the fault on I_3 affects e_4 in calculating f_4. As a result, e_4 computes $\overline{f}_4 = i_1(i_3 \oplus f_1 \oplus \overline{f}_2)$. After the operation of this gate, the value on I_2 is $\bar{i}_2 \oplus \overline{f}_4$, and the value on L is $l \oplus f_1 \oplus \overline{f}_2 \oplus f_3 \oplus \overline{f}_4$.

At the end, when I_1, I_2, I_3, I_4, and I_5 are EXORed to L again, the value of L becomes

$l \oplus f_1 \oplus \overline{f}_2 \oplus f_3 \oplus \overline{f}_4 \oplus i_1 \oplus \bar{i}_2 \oplus \overline{f}_4 \oplus i_3 \oplus f_1 \oplus \overline{f}_2 \oplus i_4 \oplus i_5 \oplus f_3 = l \oplus i_1 \oplus \bar{i}_2 \oplus i_3 \oplus i_4 \oplus i_5$

$= i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5 \oplus i_1 \oplus \bar{i}_2 \oplus i_3 \oplus i_4 \oplus i_5 = 1.$

Since L is 1, the circuit detects the fault.

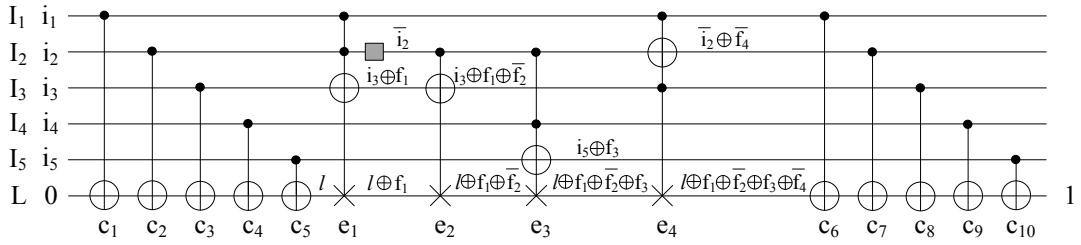


Figure 6.6: Fault detection in testable circuit.

6.2.3 *Experimental Results*

We have conducted our experiments for 20 benchmark circuits collected from [58]. We have first generated the Toffoli circuits using our improved ESOP-based synthesis as described in Section 4.2 and then applied our approach to make the circuits online testable. In Table 6.2 , we compare the testable circuits generated by our approach to that of previously reported approaches [54, 53, 8, 21, 22] in terms of quantum cost and garbage outputs. Note that QC and GO columns in the table represent the quantum cost and the number of garbage outputs, respectively. Our approach shows significant results in reducing the quantum cost and garbage outputs for all circuits.

Table 6.3 summarizes the improvements achieved by our approach. Quantum costs are

decreased by 74.72%, 78.68%, and 26.30% on average, as compared to [54, 53], [8], and [21, 22], respectively. Reductions of garbage outputs range from 96.89% to 99.84% in average, compared to the previous approaches.

The percentages of overheads in quantum cost and garbage outputs for integrating the online testability feature are given in Table 6.4. The quantum cost overhead for our approach is only 4.21% on average, compared to 312.28% for the approach in [54, 53], 388.67% for the approach in [8], and 41.40% for the approach in [21, 22]. It is interesting to note that our approach has absolutely no overhead in terms of garbage outputs since the testability feature does not produce any extra garbage. However, existing approaches produce extremely large number of garbage outputs; thus the average overheads are 63409.48%, 5163.03%, and 3113.74% for the approaches in [54, 53], [8], and [21, 22], respectively.

6.2.4 Comparisons with Our First Proposed Approach

This approach is an extended version of our first approach that we have presented in Section 6.1. Thus it inherits the advantages of the first approach, such as easy construction of a testable circuit, no checker circuit, and no extra garbage outputs. Our first proposed approach is limited only to reversible circuits generated by a particular synthesis approach. On the other hand, in this approach, there is no restriction on how the initial circuit is generated, just that the circuit be designed only from Toffoli gates. As indicated in Section 2.6, there are a number of approaches that can be used to generate such a circuit. Thus the results shown in Table 6.2 can be further improved by applying our approach on top of such optimized Toffoli circuits.

Table 6.2: Comparison of different online testable approaches.

Circuit	Approach in [54, 53]		Approach in [8]		Approach in [21, 22]		Our approach	
	QC	GO	QC	GO	QC	GO	QC	GO
9symml	25598	5952	32220	532	12262	139	11066	9
apex5	195061	45362	213843	3518	47847	1620	35354	117
apla	17815	4142	20574	334	2934	190	1859	10
bw	19019	4422	25140	386	4464	723	1249	5
cm82a	2163	502	2823	50	385	48	173	5
con1	1647	382	1929	38	307	30	184	7
cu	5560	1292	6234	110	1288	84	876	14
dk17	10376	2412	11478	186	1649	95	1113	10
ex2	1733	402	2241	42	267	23	171	5
ex5p	84852	19732	98898	1540	12484	1623	4918	8
f51m	128927	29982	164889	2670	37323	832	29072	14
frg2	218109	50722	247332	4048	150509	3616	115211	143
max46	15321	3562	19314	326	5585	114	4627	9
misex3	141354	32872	174732	2816	69153	2149	50826	14
rd73	16482	3832	20583	342	1617	115	959	7
sqn	11236	2612	14256	240	2053	100	1437	7
sqrt8	5689	1322	7086	122	835	63	530	8
sym9	24652	5732	31890	528	12262	139	11066	9
table3	223054	51872	265566	4294	31011	1767	20050	14
z4ml	3883	902	5070	88	1048	92	575	7
Average	57626.55	13400.5	68304.9	1110.5	19764.15	678.1	14565.8	21.1

Table 6.3: Improvements achieved by our approach.

	Parameter	Existing approaches		
		Approach in [54, 53]	Approach in [8]	Approach in [21, 22]
Achieved improvements	QC	74.72%	78.68%	26.30%
	GO	99.84%	98.10%	96.89%

Table 6.4: Overhead calculation of the testable design over the non-testable design.

Approach	Average overhead for testability feature	
	QC	GO
Approach in [54, 53]	312.28%	63409.48%
Approach in [8]	388.67%	5163.03%
Approach in [21, 22]	41.40%	3113.74%
Our approach	4.21%	0%

6.3 Coverage of Fault Models

Like the approaches in [54, 53] and [21, 22], our proposed work considers the single bit fault model. It is noted that the single bit fault model and single stuck-at fault model are very similar with the exception that the stuck-at fault model is independent on the initial values of the variables. Moreover, the behavior of a stuck-at fault can be translated into that of a bit fault, and vice versa. Consequently, a testable design which can detect bit faults can also detect stuck-at faults. It is worth noting that lemmas which are provided to prove the correctness of our approaches hold for both fault models.

As described in Sections 5.2.1, 5.2.2 and 5.2.3, previous approaches in [54, 53, 21, 22, 52] fail to detect all single faults; thus these approaches partially cover the bit fault and stuck-at fault models. In contrast, our both approaches as well as the approach in [8] fully cover two fault models. Table 6.5 summarizes this discussion.

Table 6.5: Coverage of fault models.

Approach	Bit fault model	Stuck-at fault model
Approach in [54, 53]	Partial	Partial
Approach in [21, 22]	Partial	Partial
Approach in [52]	Partial	Partial
Approach in [8]	Yes	Yes
Our Approaches	Yes	Yes

6.4 Summary

In this chapter, we have presented an approach for adding a testability feature to a particular type of ESOP-based circuits and then extended it for any type of Toffoli circuits. In our testable designs, the quantum cost overheads are very small, and in terms of garbage our overheads are 0. These are startling results, as previous approaches in this area require a significant increase in both of these measures. Unlike [54, 53, 21, 22, 52], our designs are fully testable under the bit fault and stuck-at fault models.

Chapter 7

Conclusion

Section 7.1 summarizes the contribution of this thesis, and Section 7.2 discusses the opportunities for further investigation.

7.1 Contributions

The contributions of this thesis span synthesis and testing of reversible circuits. Our results are summarized below.

Our first contribution, described in Chapter 4, is an improved ESOP-based reversible logic synthesis approach which leverages situations where cubes are shared by multiple outputs and ensures that the implementation of each cube requires just one Toffoli gate. Thus this approach overcomes the limitation of the previous work [46, 47] on taking advantage of the shared functionality. Moreover, this approach removes redundant CNOT gates, making it efficient in terms of quantum cost. This approach quickly synthesizes circuits, even from large functions with more than 250 variables.

Our next contribution, described in Chapter 5, addresses the shortcomings of the previously reported approaches [54, 53, 21, 22, 52] on detecting faults in testable circuits. Proposing two approaches in Chapter 6 for converting ESOP-based circuits and Toffoli circuits into online testable circuits are the key contributions of this thesis. It is noted that lemmas are provided to prove the correctness of our fault detection approaches, solving the problems of the previous works. The overheads of our testable designs over the non-testable designs are really small in terms of quantum cost and absolutely zero in terms of garbage output.

We expect that this thesis will have significant impacts on synthesizing large functions

as well as on testing reversible circuits online.

7.2 Future Work

This thesis opens up a number of possible research directions which are described below:

- The template matching technique, which only works for Toffoli circuits with all positive controls, optimizes a circuit by reducing the gate count and / or quantum cost. Our work on shared cube synthesis described in Chapter 4 and other works in [2, 48, 46] make use of negative control Toffoli gates. We plan to explore new templates tailored to the Toffoli circuits with positive and negative controls. The prior work on template matching [33] can be a good starting point for creating new templates by replacing the positive controls with negative ones in all possible ways. As an example, consider a template [33] with 6 positive controls as shown in Figure 7.1. The total number of ways to replace the positive controls with negative controls is $\binom{6}{1} + \binom{6}{2} + \ldots + \binom{6}{6} = 63$. Two of them are shown in Figure 7.2.

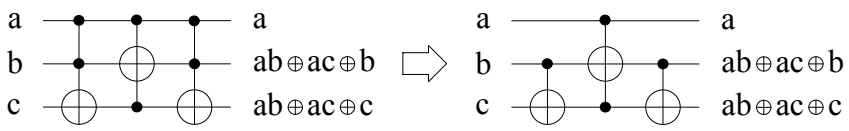
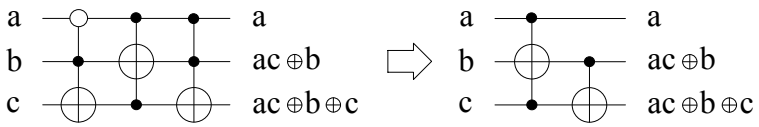
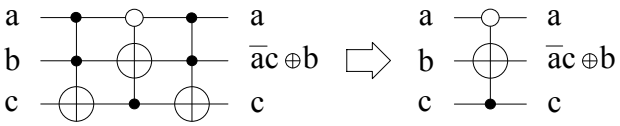


Figure 7.1: A template for Toffoli gates with only positive controls [33].

The first template (see Figure 7.2(a)) reduces the gate count by 1 and quantum cost by 9. Similarly, the second template (see Figure 7.2(b)) reduces the gate count from 3 to 1 and the quantum cost from 15 to 5. Thus templates created in this way have the potential to simplify the Toffoli circuits with positive and negative controls. This approach appears to create many templates, and applying all such templates may slow down the optimization process for large circuits. Thus finding an effective set of templates through rigorous analysis is an important avenue for future research.



(a) First template



(b) Second template

Figure 7.2: Two templates for Toffoli gates with positive and negative controls.

- Our proposed work on online testable designs deals with only two fault models. It would be interesting to explore the relationship among different fault models for detecting other faults.

- In this thesis, we have considered only the fault detection but not fault correction. A challenging task is to add the fault correction feature to our testable designs. The Hamming code technique is widely used to correct single-bit faults, which is also applicable in reversible logic [14]. Future work will consider this idea to recover reversible circuits from faults.

- Throughout the thesis we have considered Boolean reversible logic which is binary or two-valued logic. In contrast, multiple-valued logic (MVL) is a p-valued logic system with $p > 2$. For a detailed description about MVL, please refer to [34]. If $p = 3$, it is known as ternary logic. Our proposed work for constructing testable circuits described in Chapter 6 can be extended for ternary logic. This future work will involve transforming the ternary reversible gates [17] such as ternary Toffoli gates and Muthukrishnan-Stroud (M-S) gates into testable gates which will then be used to design online testable ternary circuits. Further investigation is required to make this approach feasible for MVL and to optimize the testable circuits.

Bibliography

[1] A. N. Al-Rabadi. *Reversible Logic Synthesis: From Fundamentals to Quantum Computing*. Springer-Verlag, 2004.

[2] M. Arabzadeh, M. Saeedi, and M. Zamani. Rule-based optimization of reversible circuits. In *Proceedings of Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 849–854, 2010.

[3] W. C. Athas and L. J. Svensson. Reversible logic issues in adiabatic CMOS. In *Proceedings of Workshop on Physics and Computation(PhysComp)*, pages 111–118, Dallas, TX, 1994.

[4] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52(5):3457–3467, Nov 1995.

[5] A. Chakraborty. Synthesis of reversible circuits for testing with universal test set and C-testability of reversible iterative logic arrays. In *Proceedings of the 18th International Conference on VLSI Design*, pages 249–254, 2005.

[6] J. Chen, X. Zhang, L. Wang, X. Wei, and W. Zhao. Extended Toffoli gate implementation with photons. In *Proceedings of 9th International Conference on Solid-State and Integrated-Circuit Technology (ICSICT)*, pages 575–578, China, 20-23 Oct 2008.

[7] J. Donald and N. K. Jha. Reversible logic synthesis with Fredkin and Peres gates. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 4(1):2:1–2:19, 2008.

[8] N. Farazmand, M. Zamani, and M. B. Tahoori. Online fault testing of reversible logic using dual rail coding. In *Proceedings of 16th IEEE International On-Line Testing Symposium (IOLTS)*, pages 204–205, 5-7 July 2010.

[9] K. Fazel, M. Thornton, and J. E. Rice. ESOP-based Toffoli gate cascade generation. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 206–209, Victoria, BC, Canada, 22-24 Aug. 2007.

[10] M. P. Frank. Introduction to reversible computing: motivation, progress, and challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390, Ischia, Italy, 4-6 May 2005.

[11] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2317–2330, 2006.

[12] J. P. Hayes, I. Polian, and B. Becker. Testing for missing-gate faults in reversible circuits. In *Proceedings of the 13th Asian Test Symposium*, pages 100–105, 2004.

[13] M. Ibrahim, A. R. Chowdhury, and H. M. H. Babu. Minimization of CTS of k-CNOT circuits for SSF and MSF model. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pages 290–298, Boston, MA, 2008.

[14] R. K. James, T. K. Shahana, K. P. Jacob, and S. Sasi. Fault tolerant error coding and detection using reversible gates. In *Proceedings of IEEE Region 10 Conference - TENCON*, pages 1–4, Taipei, 2007. doi: 10.1109/TENCON.2007.4428776.

[15] N. K. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, 2003.

[16] M. Karpovsky. Finite orthogonal series in the design of digital devices. *John Wiley & Sons*, 1976.

[17] M. H. A. Khan and M. A. Perkowski. Quantum ternary parallel adder/subtractor with partially-look-ahead carry. *Journal of Systems Architecture*, 53(7):453–464, 2007.

[18] D. K. Kole, H. Rahaman, and D. K. Das. Synthesis of online testable reversible circuit. In *Proceedings of 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 277–280, Vienna, 14-16 April 2010.

[19] P. K. Lala. *An Introduction to Logic Circuit Testing*. Morgan & Claypool, 2008.

[20] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[21] S. N. Mahammad, S. Hari, S. Shroff, and V. Kamakoti. Constructing online testable circuits using reversible logic. In *Proceedings of 10th IEEE International VLSI Design and Test Symposium (VDAT)*, pages 373–383, Goa, India, August 2006.

[22] S. N. Mahammad and K. Veezhinathan. Constructing online testable circuits using reversible logic. *IEEE Transactions on Instrumentation and Measurement*, 59(1):101–109, 2010.

[23] D. Maslov. Reversible logic synthesis benchmarks page. `http://www.cs.uvic.ca/~dmaslov/`.

[24] D. Maslov. *Reversible logic synthesis*. PhD thesis, University of New Brunswick, 2003.

[25] D. Maslov and G. W. Dueck. Improved quantum cost for n-bit Toffoli gates. *IEE Electronics Letters*, 39(25):1790–1791, 2003. Corrected and expanded version downloaded from `http://arxiv.org/PS_cache/quant-ph/pdf/0403/0403053v1.pdf`.

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

[27] D. Maslov, G. W. Dueck, and D. M. Miller. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(6):765–769, 2005.

[28] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, 2008.

[29] D. Maslov and D. M. Miller. Comparison of the cost metrics through investigation of the relation between optimal NCV and optimal NCT three-qubit reversible circuits. *IET Computers & Digital Techniques*, 1(2):98–104, 2007.

[30] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, pages 1208–1213, Washington, DC, USA, 2005. IEEE Computer Society.

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

[32] R. C. Merkle and K. E. Drexler. Helical logic. *Nanotechnology*, 7:325–339, 1996.

[33] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th annual Design Automation Conference (DAC)*, pages 318–323, 2003.

[34] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations*. Morgan & Claypool Publishers, 2007.

[35] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive sum-of-products. In *Proceedings of the 5th International Reed-Muller Workshop*, pages 242–250, Starkville, Mississippi, August 2001.

[36] M. Mohammadi and M. Eshghi. On figures of merit in reversible and quantum logic designs. *Quantum Information Processing*, 8:297–318, August 2009.

[37] N. M. Nayeem and J. E. Rice. Improved ESOP-based synthesis of reversible logic. In *Proceedings of the Reed Muller Workshop*, Tuusula, Finland, 25-26 May 2011.

[38] N. M. Nayeem and J. E. Rice. Online fault detection in reversible logic. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, Vancouver, Canada, October 2011.

[39] N. M. Nayeem and J. E. Rice. A shared-cube approach to ESOP-based synthesis of reversible logic. *Facta Universitatis, University of Niš*, 2011.

[40] N. M. Nayeem and J. E. Rice. A simple approach for designing online testable reversible circuits. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, Canada, August 2011.

[41] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

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

[43] P. Picton. Optoelectronic, multivalued, conservative logic. *International Journal of Optical Computing*, 2:19–29, 1991.

[44] H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya. On the detection of missing-gate faults in reversible circuits by a universal test set. In *Proceedings of the 21st International Conference on VLSI Design*, pages 163–168, 2008.

[45] J. E. Rice and V. Suen. Using autocorrelation coefficient-based cost functions in ESOP-based Toffoli gate cascade generation. In *Proceedings of 23rd Canadian Conference on Electrical and Computer Engineering (CCECE)*, Calgary, Canada, May 2010. downloaded Jun. 2010 from `http://www.cs.uleth.ca/~rice/publications/CCECE2010.pdf`.

[46] Y. Sanaee. Generating Toffoli networks from ESOP expressions. Master's thesis, University of New Brunswick, 2010.

[47] Y. Sanaee and G. W. Dueck. Generating toffoli networks from ESOP expressions. In *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 715–719, Victoria, BC, Canada, 13-18 June 2009.

[48] Y. Sanaee and G. W. Dueck. ESOP-based Toffoli network generation with transformations. In *Proceedings of 40th IEEE International Symposium on Multiple-Valued Logic*, pages 276–281, 2010.

[49] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[50] F. Somenzi. CUDD: CU Decision Diagram Package Release 2.3.1. University of Colorado at Boulder, 2001.

[51] S. Tayu, S. Ito, and S. Ueno. On the fault testing for reversible circuits. In *Proceedings of the 18th international conference on Algorithms and computation*, pages 812–821, Berlin, Heidelberg, 2007. Springer-Verlag.

[52] H. Thapliyal and A. P. Vinod. Designing efficient online testable reversible adders with new reversible gate. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1085–1088, New Orleans, LA, 27-30 May 2007.

[53] D. P. Vasudevan, P. K. Lala, D. Jia, and J. P. Parkerson. Reversible logic design with online testability. *IEEE Transactions on Instrumentation and Measurement*, 55(2):406–414, 2006.

[54] D. P. Vasudevan, P. K. Lala, and J. P. Parkerson. Online testable reversible logic circuit design using NAND blocks. In *Proceedings of IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 324–331, Los Alamitos, CA, USA, 10-13 October 2004.

[55] A. De Vos. *Reversible Computing: Fundamentals, Quantum Computing, and Applications*. Wiley-VCH, 2010.

[56] L. Wang, C. Wu, and X. Wen, editors. *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann, 2006.

[57] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference (DAC)*, pages 270–275, 2009.

[58] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Proceedings of 38th International Symposium on Multiple Valued Logic*, pages 220–225, 2008. RevLib is available at `http://www.revlib.org`.

[59] J. Zhong and J. C. Muzio. Analyzing fault models for reversible logic circuits. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 2422–2427, Vancouver, BC, 2006.