Towards Realizing Low Quantum Cost Reversible Logic Circuits

THESIS

Submitted in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Sai Phaneendra P

ID No. 2009PH230009H

Under the Supervision of

Dr. M. B. Srinivas

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

2017

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the thesis entitled "Towards Realizing Low Quantum Cost Reversible Logic Circuits" and submitted by Sai Phaneendra P ID No 2009PH230009H for award of Ph.D. of the Institute embodies original work done by him under my supervision.

Signature of the Supervisor

Dr. M. B. SRINIVAS

Professor

BITS-Pilani, Hyderabad Campus

Date:

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. M. B. Srinivas for his constant support and valuable guidance through out my work. I would also like to thank my Doctoral Advisory Committee members, Prof. BVVSN Prabhakar Rao, Dr. Sumit Kumar Chatterjee and Prof. Y Yoganandam for their time and their constructive comments and suggestions.

I take this opportunity to express my gratitude for Council of Scientific and Industrial Research (CSIR) - India and BITS-Pilani, Hyderabad Campus, India for providing financial assistance and infrastructure support during my thesis work.

I appreciate the help I received from Chetan Kumar V, Goutham M and Avinash S Vaidya during different stages of my work.

Furthermost, I would like to thank all members of Department of Electrical Engineering at BITS-Pilani, Hyderabad Campus, that have supported me by their suggestions and discussions. I would also like to thank members of Academic Research Division, Academic Registration & Counseling Division and Student Welfare Division at BITS-Pilani, Hyderabad Campus for their support during my thesis work.

Finally, I am grateful to my family members, especially my parents, Sridhar and Padmaja, and my wife, Keerthi, for their patience, encouragement and inspiration, without which this thesis would not have been possible.

Abstract

Research on reversible logic gained momentum in the past decade owing to its applications in quantum computing and low power circuit implementation. Reversible circuits realized by the existing synthesis techniques are often sub-optimal and optimization techniques are applied on them to reduce the 'cost', a metric used to compare reversible circuits. While several optimization techniques are present in the literature, finding optimal or near-optimal reversible circuit is still an open-problem.

In this thesis, a set of optimization techniques is proposed that can be applied on a pair of gates to reduce the cost of a reversible circuit. Initially, a decomposition technique is presented that decomposes a pair of gates into a set of smaller gates without changing the functionality. This technique is then used in conjunction with an Exclusive-OR Sum-of-Product (ESOP) based reversible circuit synthesis algorithm to check its efficiency. It is known that the decomposition technique does not always result in cost reduction for a given gate pair. This leads us to examine the condition that results in reduction and it has been found that the effectiveness of this technique is proportionate to the number of common control lines between the gate pair. In order to increase the number of common control lines, a transformation approach is presented.

Lastly, a representation is proposed to classify a pair of gates according to their control line characteristics. This classification helps in identifying the gate pairs that can be optimized as opposed to those that can not be. Based on this classification, a methodology is presented to reduce the cost of a given gate pair. In order to apply the proposed techniques on reversible circuits, these techniques are integrated into ‘greedy’ optimization algorithms. A set of benchmark circuits is applied on these algorithms and are compared with existing benchmark circuits. Results indicate that the proposed techniques lead to significant improvement in the cost of reversible circuits.

Contents

List of Figures

List of Tables

List of Abbreviations

BDD	Binary Decision Diagram
CCL	Complementary Control Line
CNOT	Controlled NOT
ECL	Equal Control Line
ESOP	Exclusive-or Sum-of-Product
MCT	Multi-Control Toffoli
MPMCT	Multi-Polarity Multi-Control Toffoli
QC	Quantum Cost
QR	Quadruple Representation
SOP	Sum-of-Product
UCL	Unequal Control Line

Chapter 1

Introduction

Power dissipation is a significant problem in designing small and high performance complex integrated circuits. Recent advancements in integration of circuits and fabrication techniques have helped reduce the power consumption but some part of power, which is independent of underlying technology, is dissipated due to information loss. In 1961 Landauer [1] proved that, a loss of single bit of information in irreversible (or conventional) logic computation leads to dissipation of heat energy of $kTln2$ Joules, where K and T are Boltzmann's constant and absolute temperature at which computation is performed, respectively. This principle has been experimentally validated recently [2] by measuring the amount of heat dissipated when a bit of information is erased. Further, in 1973, Bennett [3] showed that the dissipated energy is directly correlated to the number of information bits lost and if the computation can be made information-lossless then the energy dissipation can be reduced or theoretically eliminated. One way this can be achieved is by making computation reversible, i.e. a bijective mapping between inputs and outputs, resulting in no information loss.

Reversible computation thus, for one, is motivated by its implication to reduce the power consumption. Further, it also has relevance to quantum computing [4] because unitary transformation in quantum computing was shown to be reversible [5]. Quantum algorithms are known to solve some hard problems in polynomial time like Shor's algorithm [6] for factoring and Grover's algorithm [7] for searching for which classical computing is known to take exponential time. In applications of these algorithms, currently a large part of the computation is performed using classical computing which can be described in terms of classical functions [5]. If these were to be implemented on a quantum computer, the classical logic functions need to be translated into reversible logic functions (since quantum transformations are reversible in nature) and implemented using reversible gates and circuits. Apart from the applications in low power design and quantum computing, reversible computing also has applications in optical [8–10], DNA computing [11,12], cryptography [13], program inversion [14] and coding devices [15,16].

The bijective mapping of a reversible function does not allow for fan-out and feedback in the implementation of reversible logic circuits. These constraints limit the usage of existing conventional logic synthesis techniques to synthesize reversible logic functions. In order to implement these functions, new algorithms, techniques and design flows have been developed in the past decade [17,18]. A general design flow for reversible logic function implementation is shown in figure 1.1.

Synthesis techniques can be broadly categorized in to exact and heuristic. Exact synthesis techniques [19–21] are based on exhaustive search or Boolean satisfiability which search for optimal solutions to generate minimum cost circuits but are applicable only on functions with small number of variables (not more than 6) [18]. For functions with a large number of variables, heuristic approaches like transformation based [22–25], cycle based [26–28], binary decision diagram (BDD) based [29,30], Exclusive-OR Sum-

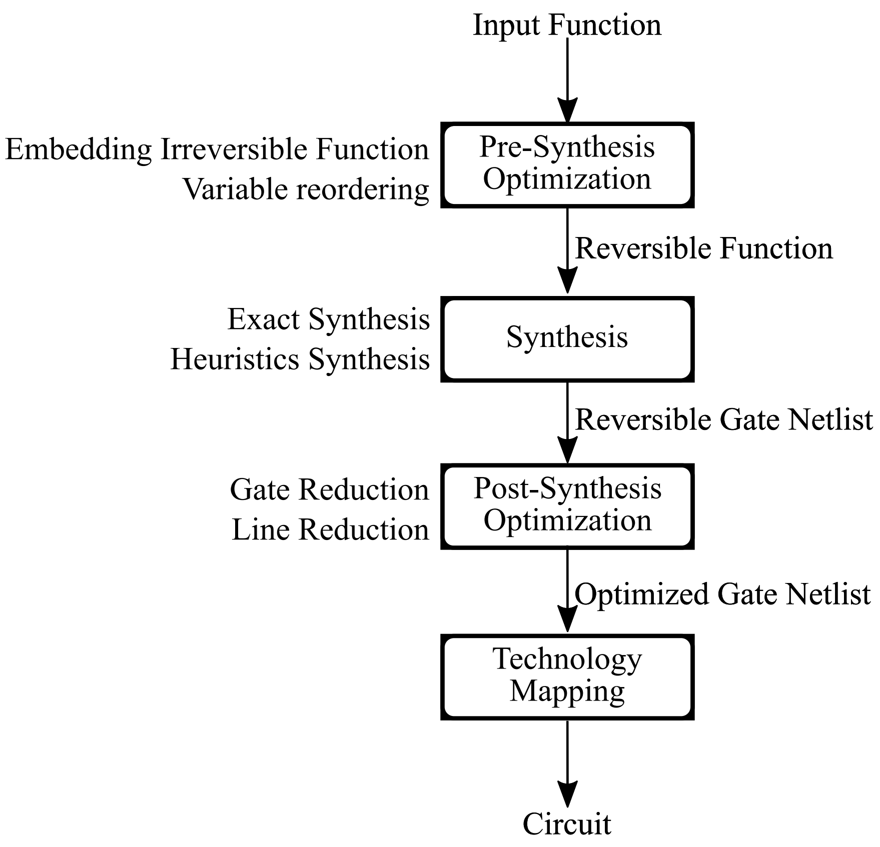


Figure 1.1: Design Flow for Reversible Logic Synthesis [17]

of-Product (ESOP) based approaches [31–36] are used. In the transformation based approach [23] input specification is given in the form of a truth-table. The lines of the truth-table are traversed and gates are added until it is transformed to a identity function truth-table. In the cycle based approach [26], input specification is given in the form of a permutation. This permutation is decomposed into a set of cycles and each cycle is synthesized separately and cascaded to generate final gate netlist. In the BDD based approach [29], a BDD is constructed from input specification and each node is replaced with corresponding gate netlist and added to the circuit. In ESOP based approach [31] the product terms of ESOP are directly mapped using equivalent reversible gates.

Heuristic synthesis approaches usually generate circuits which are sub-optimal and post-synthesis optimization techniques [25, 37–40] are typically applied to reduce the cost of the circuit. Optimization technique presented in [40] reduces the cost of a reversible circuit with the help of additional lines. However, increase in the number of lines directly affects the complexity of its implementation [41]. Technique described in [25] uses template-based local optimization to reduce the cost of the circuit. However, this method involves a search for templates which becomes complex as the circuit size increases. Rule-based optimization algorithms have been presented in [37–39] where a group of gates is replaced with a lesser cost gate netlist using a set of rules. However, when a pair of gates is considered, existing optimization techniques address only a few combinations of gate pairs. In this thesis, we explores different optimization techniques to reduce the cost of a reversible circuit using gate pairs and quantify their performance with respect to the existing ones.

1.1 Contributions of the Thesis

Contributions of this thesis may be described as follows:

- Initially a technique is developed to decompose a pair of gates to a network of smaller gates without any change in the functionality. This technique generates and eliminates any redundant gates that appear at the time of decomposition resulting in a gate netlist with reduced cost compared with initial pair. A cube decomposition algorithm is presented which integrates the proposed technique with ESOP based approach to synthesize reversible circuits with reduced cost.

- The decomposition technique however does not always result in cost reduction for a given gate pair. This lead us to examine the nature of a given gate pair that results in reduction and is found that the effectiveness of the technique is proportional to the number of common control lines between the gate pair. That is, larger the number of lines more effective the technique is. Thus, in order to increase the lines, a transformation approach named "Complementary Control Line Transformation" is proposed which transforms some of the unequal control lines to equal control lines with the help of extra gates. Further, we also found that this transformation enables application of some of the existing rules which cannot be applied on gates with more than one unequal control line.

- Finally, a framework based on quadruple representation, which classifies a pair of gates based on its control line characteristics, is proposed. This classification helps in identifying gate pairs that can be optimized as opposed to those that can not be. Further, this classification also helps in selecting appropriate techniques that can be applied on a gate pair to reduce the cost of its implementation. A 'greedy' optimization algorithm is presented that uses the classification and the proposed techniques to reduce the overall cost of the circuit.

1.2 Organization of the Thesis

Rest of the thesis is organized as follows:

Chapter 2 provides the necessary background required for the work described in this thesis. It describes reversible logic, reversible gates and their circuit implementation. A brief overview of quantum logic and quantum gates is presented. Finally, implementation of reversible gates and circuits using quantum gates and the cost metrics to evaluate them are discussed.

Chapter 3 presents a technique to decompose a pair of gates. Motivation for this approach and its application in ESOP based synthesis are discussed. This technique is then used in conjunction with an ESOP based reversible circuit synthesis algorithm to verify its efficiency.

Chapter 4 demonstrates the relation between the number of equal control lines in a given gate pair and the cost reduction achieved after applying the decomposition approach. Later, a transformation approach is described that increases the number of equal control lines in a given gate pair. Application of this transformation approach to different existing optimization techniques is then presented.

Chapter 5 proposes a framework based on a representation to classify a pair of gates. Existing and proposed optimization techniques are then categorized according to this representation. Finally, an optimization algorithm is described which is applied to these techniques to reduce the cost of a given reversible circuit.

Chapter 6 concludes the thesis and presents directions for future research.

Chapter 2

Preliminaries

2.1 Introduction

This chapter provides a brief summary of reversible logic, reversible gates and circuits and necessary background required for the work reported in this thesis. The first part of this chapter discusses reversible functions and conversion of irreversible functions to reversible functions. The second part gives details of different reversible gates and their functionalities, gate libraries and circuits. The third part discusses quantum gates that can be used to implement reversible gates and circuits. Finally, different cost metrics to evaluate these gates and circuits are presented.

2.2 Reversible Functions

In classical computing, a Boolean function is defined as

Definition 2.1 (**Boolean Function**)**.** A Boolean function is a mapping $f : \mathbb{B}^n \to \mathbb{B}$ where $\mathbb{B} = \{0, 1\}$ denotes the *Boolean values* and $n \in \mathbb{N}$.

The behavior of a function is represented as a tabular form known as *truth table*.

The left side of this table lists the permutations of *inputs* and the right side provides the results of the function as *outputs* for each input combination. Following examples show different Boolean functions and their truth tables:

Example 2.1. Figure 2.1 shows the truth table of a negation function ($\neg$) which has one input (a) and one output (z).

a	z
0	1
1	0

Figure 2.1: Negation Function ($\neg$)

Example 2.2. Figure 2.2 shows the truth table of an OR function ($\vee$) which has two inputs (a , b) and one output (z).

a	b	z
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.2: OR function ($\vee$)

A multi-output Boolean function with m outputs is a system of m Boolean functions $f_i(x_1, x_2, ..., x_n)$ where $1 \leq i \leq m$. For example, figure 2.3 shows a multi-output function, *Half adder*, with two inputs (a, b) and two outputs (*sum*, *carry*).

a	b	*sum*	*carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 2.3: Half Adder Function

In contrast to classical computing, reversible computing is limited to functions which can be operated in inverted fashion, i.e. inputs can be constructed from outputs. A reversible function is defined as:

Definition 2.2 (**Reversible Function**)**.** An m-input m-output function $f : \mathbb{B}^m \rightarrow \mathbb{B}^m$ where $m \in \mathbb{N}$ is said to be ***reversible*** if it has one-to-one and onto mapping i.e., bijective mapping between inputs and outputs.

Since the mapping is bijective, reversible functions can be operated in forward and backward directions resulting in no information loss. If a function f is reversible then its number of inputs is same as the number of outputs. For example, negation function ($\neg$) shown in figure 2.1 is a reversible function whereas OR function ($\vee$) shown in figure 2.2 is not. Even though the half adder function shown in figure 2.3 has same number of inputs and outputs, the mapping between inputs and outputs is not bijective and thus not reversible. These irreversible functions can be transformed in to reversible function by adding additional inputs/outputs and assigning don't-care values such that the mapping between inputs and outputs is bijective. This process of transforming an irreversible function to reversible function is called *embedding of irreversible function* [42].

Definition 2.3 (**Ancillary Inputs**)**.** The additional inputs that are added to make a function reversible are termed as *ancillary inputs* or *ancillae inputs*. In general, these inputs are assigned with constant values and thus they are also known as *constant inputs.*

Definition 2.4 (**Garbage Outputs**)**.** The additional outputs that are added to make a function reversible are termed as *garbage outputs.*

According to Theorem 1 given in [43], the minimum number of garbage outputs required to make a function reversible is $\lceil log_2(\mu) \rceil$, where μ is the maximum of number of times an output pattern is repeated in its truth table.

Example 2.3. Consider the OR function shown in figure 2.2. From the truth table, it can be seen that the output has three occurrences of '1'. Thus, the number of garbage outputs that is required to make this function reversible is $\lceil log_2(3) \rceil = 2$. Since, the number of outputs and inputs should be equal, one ancillary input is required to make the function reversible. After adding these additional inputs and outputs and assigning values to the don't cares, the final truth table is shown in figure 2.4. The outputs go_1 and go_2 are the garbage outputs and the input ai_1 is the ancillary input. Don't care values are assigned in such a way as to achieve bijective mapping between the inputs and outputs.

ai_1	a	b	z	go_1	go_2
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	1	1

Figure 2.4: Reversible OR function after Embedding Process

Example 2.4. Consider the half adder function shown in figure 2.3. The maximum number of occurrences of an output pattern is '2' for '10'. Thus at least one ($\lceil log_2(2) \rceil = 1$) garbage output and one ancillary input are required to make this function reversible. The reversible half adder after embedding the extra inputs and outputs is shown in figure 2.5 where go_1 and ai_1 are garbage output and ancillary input respectively.

The embedding of irreversible functions is not unique and there are several possibilities depending on the don't care assignments. For example figure 2.6 shows an alternative embedding of half adder function to make it reversible which is different from the embedding shown in figure 2.5. This process of embedding irreversible functions to reversible has been studied in the past and different approaches have been proposed in literature [42, 44–46].

ai_1	a	b	sum	$carry$	go_1
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	1

Figure 2.5: Reversible Half Adder after embedding process

ai_1	a	b	sum	$carry$	go_1
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	0	1

Figure 2.6: Reversible Half Adder Function with a different embedding process

2.3 Reversible Gates, Libraries and Circuits

A reversible gate is defined as,

Definition 2.5 (Reversible Gate). A gate is said to be *reversible* if it realizes reversible function i.e. bijection.

The variables of a reversible function are represented with circuit lines in the reversible gates. The commonly used reversible gates are NOT, CNOT, Toffoli [47] and Fredkin gates [48]. The characteristics and representations of these gates are discussed below:

NOT Gate A *NOT gate* has one input and one output and inverts the input.

CNOT Gate A *Controlled-NOT (CNOT) gate* or *Feynman gate* has a control line and a target line where the target line value gets inverted if the control line is equal to ‘1’.

Toffoli Gate A Toffoli gate has two control lines and one target line where the target line value gets inverted when both the control lines are set to ‘1’.

Swap Gate A Swap gate has two inputs, two outputs and swaps the values on the two lines.

Fredkin Gate A Fredkin gate is a Controlled-Swap gate with one control and two target lines. When the control line is set to ‘1’, the target lines are swapped.

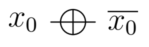
MCT Gate A n-bit multi-controlled Toffoli (MCT) gate is a generalized Toffoli gate with $n-1$ control lines and one target line.

MPMCT Gate A multi-polarity multi-control Toffoli (MPMCT) gate is similar to MCT gate but control lines can be either positive or negative. The *negative* control line in a gate is active when it is set to ‘0’.

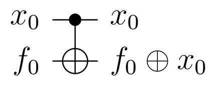
The representation of these gates is shown in figure 2.7. The positive control line is indicated with $\bullet$, negative control line with $\circ$ and target line with $\oplus$. The MCT and MPMCT gates are represented as $MCT(C;t)$ and $MPMCT(C;t)$ respectively, where C is the set of control lines and t is the target line.

2.3.1 Reversible Gate Libraries

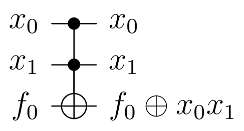
A gate library is a group of basic gates by which any function can be implemented. Similarly, a reversible gate library consists of gates that can realize any reversible function. There exist different reversible gate libraries as shown in Table 2.1:



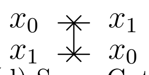
(a) NOT Gate



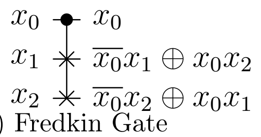
(b) CNOT Gate

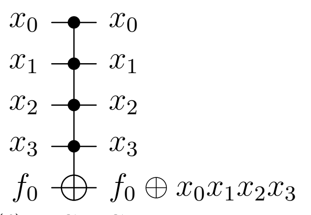


(c) Toffoli Gate

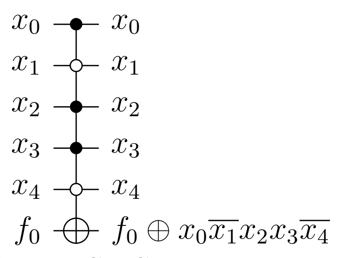


(d) Swap Gate


x_0
x_1
x_2
x_0
$\overline{x_0}x_1 \oplus x_0x_2$
$\overline{x_0}x_2 \oplus x_0x_1$
(e) Fredkin Gate



(f) MCT Gate



(g) MPMCT Gate

Figure 2.7: Reversible Gates

Table 2.1: Reversible Gate Libraries

Library	**Included Gate(s)**
NCT	NOT, CNOT and Toffoli Gates
NCTS	NOT, CNOT, Toffoli and Swap Gates
MCT	Multi-Control Toffoli Gates
MPMCT	Multi-Polarity Multi-Control Toffoli gates

2.3.2 Reversible Circuit

The bijective mapping in reversible functions imposes the constraints like no fan-out and no feedback in the implementation. Thus, reversible functions are realized by cascading reversible gates to form a *reversible circuit.* For example, a reversible circuit with cascade of four reversible gates is shown in figure 2.8. The representation of this circuit with MPMCT gates is given as follows:

$$MPMCT(\{x_1\};x_0)\circ MPMCT(\{x_0,x_1\};x_3)\circ MPMCT(\{\overline{x_0},x_1,\overline{x_2}\};x_3)\circ MPMCT(\{x_0\};x_1)$$

where $\circ$ denotes cascade of gates/circuits.

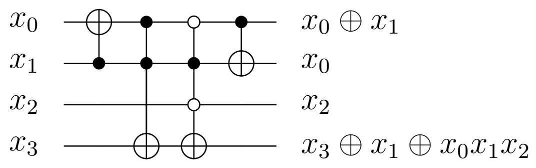


Figure 2.8: Example of Reversible Circuit with cascade of gates

Similar to reversible gates, reversible circuits also have same number of inputs and outputs and perform bijection. Also, reversible circuits can be operated in backward direction to realize inverse functions.

Example 2.5. Consider the reversible half adder function given in Example 2.4. The reversible circuit to implement this function is shown in figure 2.9. The *carry* is generated using a Toffoli gate with the help of an ancilla line (ai_1) and the *sum* is generated using a CNOT gate. This circuit can be operated in backward direction to get input a from Sum and b.

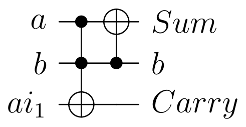


Figure 2.9: Reversible Half Adder Circuit

Now, definitions used in this thesis for different lines of a reversible circuit are given below:

Control Connection The *control connection* defines the state of connection i.e., positive or negative, of a given control line in a gate.

Equal Control Line (ECL) A line having same control connection and same value for a set of gates is termed as *equal control line (ECL)* .

Complementary Control Line (CCL) A line having positive control connection on one gate and negative control connection on another gate is termed as *complementary control line (CCL)* for the gate pair.

Unequal Control Line (UCL) A line having different control connections for a pair of gates is termed as *unequal control line (UCL).*

Unused Line A line which is neither a control line nor a target line of a gate is termed as *unused line* for that gate.

In order to the implement the classical Boolean functions in quantum algorithms, the functions are first realized using reversible circuits and the gates in the circuit are mapped to equivalent quantum circuits. In the next section different quantum gates and circuits are explained along with the mapping of reversible circuits to quantum circuit.

2.4 Quantum Gates and Circuits

In quantum computing, information is encoded into quantum states and is represented using *quantum bit* represented as *qubit.* In contrast to Boolean logic where classical bits take values of Boolean's 0 or 1, qubits not only take values of Boolean's 0 or 1 but also take superposition of these values. A qubit $|\psi\rangle$ can be described as [5]:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{2.1}$$

where $|0\rangle$ and $|1\rangle$ are column vectors corresponding to $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ respectively. The coefficients α_0 and α_1 are complex values, with $|\alpha_0|^2 + |\alpha_1|^2 = 1$, which represents the amplitudes of $|0\rangle$ and $|1\rangle$ respectively. When a qubit is measured, depending on the current state, 0 and 1 are returned with a probability of $|\alpha_0|^2$ and $|\alpha_1|^2$ respectively.

Definition 2.6 (Unitary Matrix). If the conjugate transpose ($U^\dagger$) of a matrix (U) is also its inverse then the matrix is termed as Unitary Matrix, i.e. $U^\dagger U = UU^\dagger = I$ where I is identity matrix.

For example, consider a matrix $U = \begin{bmatrix} i & 0 \\ 0 & 1 \end{bmatrix}$. The conjugate transpose of this matrix is $U^\dagger = \begin{bmatrix} -i & 0 \\ 0 & 1 \end{bmatrix}$ and $U^\dagger U = UU^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Thus, the matrix U is a unitary matrix.

A quantum gate is represented as $2^n \times 2^n$ unitary matrix operated on n qubits. For example, a single-qubit quantum gate where $n = 1$, the quantum gates represent a 2×2 unitary matrix. The commonly used single-qubit gates are discussed below:

Pauli-X Gate The Pauli-X gate transforms a qubit $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ to $|\psi\rangle = \alpha_1 |0\rangle + \alpha_0 |1\rangle$ i.e. interchanging the amplitudes of $|0\rangle$ and $|1\rangle$ and the transformation matrix is given in Eq. (2.2). Thus, this gate is also referred to as *NOT gate.*

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.2}$$

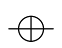
V **Gate** The square-root of NOT (Pauli-X) gate is referred to as V gate i.e. $V^2 = X$. The transformation matrix of this gate is given in Eq. (2.3). The cascade of two V gates results in a NOT gate.

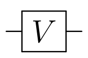
$$V = \frac{1+i}{2}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \tag{2.3}$$

$V^{\dagger}$ **Gate** The V^{+} gate is a complex conjugate of V gate i.e. $V^{+}V = I$ where I is a identity gate and the transformation matrix is given in Eq. (2.4). Similar to V gate, two consecutive V^{+}gates also result in a NOT gate.

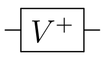
$$V^{+} = \frac{1-i}{2}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix} \tag{2.4}$$

The representation of NOT (Pauli-X), V and V^{+} gates are shown in figure 2.10.


(a) NOT Gate



(b) V Gate



(c) V^{+}Gate

Figure 2.10: Single-qubit Quantum Gates

Similar to a single-qubit quantum gate, a two-qubit quantum gate represents a 4×4 unitary matrix. A general two-qubit gate is a *controlled-U* gate which consists of control and target qubits. The unitary matrix U is applied on the target qubit only if the control qubit is $|1\rangle$ else it is unchanged. The transformation matrix is given in Eq. (2.5), where $U_{00}, U_{01}, U_{10}, U_{11}$ are elements corresponding to matrix U. The commonly used two-qubit quantum gates are explained as follows:

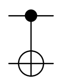
$$Controlled-U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{bmatrix} \tag{2.5}$$

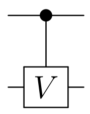
The Controlled-NOT (CNOT), Controlled-V and controlled-V^{+}gates are controlled-U gates where $U = X$, $U = V$ and $U = V^{+}$respectively. The transformation matrices of these gates are given in Eqs. (2.6), (2.7) and (2.8) and their representation is shown in figure 2.11.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.6}$$

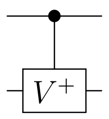
$$Controlled - V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{bmatrix} \tag{2.7}$$

$$Controlled - V^{+} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \end{bmatrix} \tag{2.8}$$


(a) CNOT Gate



(b) Controlled-V Gate



(c) Controlled-V^{+} Gate

Figure 2.11: Two-qubit quantum gates

Quantum circuits are implemented by cascading quantum gates on a set of qubits. The widely used quantum library in the literature to map reversible circuits to quantum circuits is NCV library [49]. This library consists of NOT, CNOT, Controlled-V and Controlled-V^{+} gates. The quantum NOT and CNOT gates are same as reversible

NOT and CNOT gates when applied on Boolean logic. The mapping of Toffoli gate to quantum circuit, for different combinations of control lines, using NCV gate library is shown in figure 2.12.

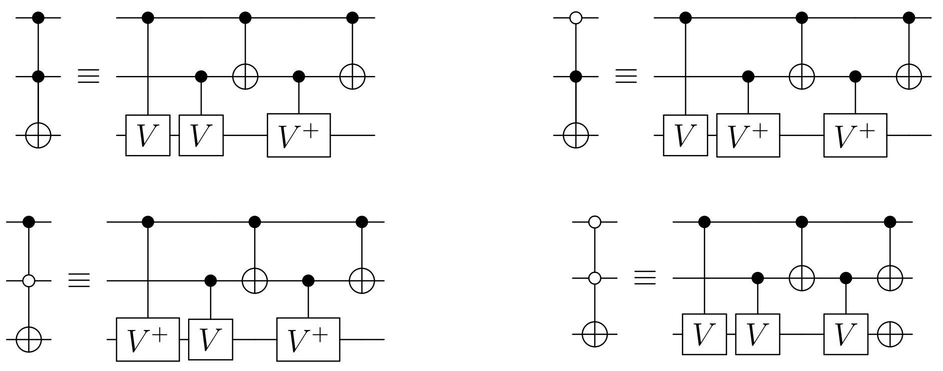


Figure 2.12: Mapping of Toffoli gate for different combinations of control lines using NCV library

Example 2.6. Consider the reversible circuit shown in figure 2.9. The gates in the circuit are mapped to their quantum equivalent circuits resulting in circuit shown in figure 2.13.

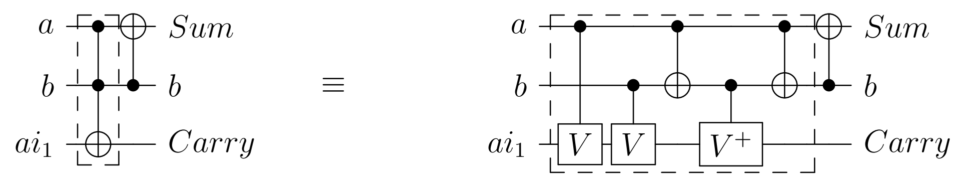


Figure 2.13: Mapping of Reversible Half Adder Circuit to Quantum Circuit

Since the quantum gates can be applied on at most two-qubits, the maximum possible mapping for MCT/MPMCT gates is with the number of control lines as 2 i.e. Toffoli gate. Thus these gates are decomposed into NOT, CNOT or Toffoli gates before mapping to NCV gate library.

2.5 Cost Metrics for Reversible and Quantum Circuits

In this section, different cost metrics to evaluate reversible circuits that are available in the literature are discussed.

2.5.1 Number of Gates

The ‘number of gates’ refers to the total number of reversible gates required to realize a given function. This metric depends on the gate library that is selected to implement the function. For example, the number of gates in the circuit shown in figure 2.8 is 4 when MPMCT gate library is selected but 8 when MCT gate library is selected. In this thesis, the widely used reversible gate library i.e. NCT gate library is chosen to compute the gate count in the circuit. The MCT/MPMCT gates in the circuit with more than two control lines are decomposed into NCT gates using the approach presented in Lemmas 7.2 and 7.3 of [49]. The cost of an MCT gate in terms of Toffoli gates is given in Eq. (2.9) and the total number of lines in circuit respectively. The total number of NCT gates that is present in a circuit gives the NCT cost of that circuit.

$$Cost_{NCT}(MCT) = \begin{cases} 8(c-3) & if\, c > \lceil \frac{n+1}{2} \rceil \\ 4(c-2) & if\, c \leq \lceil \frac{n+1}{2} \rceil \end{cases} \tag{2.9}$$

where c and n denote the number of control lines of a gate.

2.5.2 Quantum Cost

The Quantum Cost (QC) of a reversible gate is the number of primitive quantum gates required to implement the gate functionality. The primitive gates in NCV quantum library are NOT, CNOT, controlled V and controlled V+ gates [49]. The QC of a reversible gate is calculated according to RevLib [50] and is given in Table 2.2. Here c denotes the number of control lines and n denotes the total number of lines in the circuit. When all the control lines of an MCT gate are negative, the QC is increased by a value of 2. The QCs of each reversible gate in a circuit are summed up to calculate the QC of that circuit. As an example, the QC of the circuit shown in figure 2.8 is $1 + 5 + 13 + 1 = 20$.

Table 2.2: Quantum Cost of an MCT gate

c	*QC*		
	$(n-(c+1)) \geq$		
	0	1	$c-2$
0	1		
1	1		
2	5		
3	13		
4	29		26
5	61	52	38
6	125	80	50
7	253	100	62
8	509	128	74
9	1021	152	86
> 9	$2^{c+1}-3$	$24(c+1)-88$	$12(c+1)-34$

where c is number of control lines in an MCT gate;
n is total number of lines in the circuit

2.5.3 Number of Lines

The total number of qubits in a circuit indicates the number of lines required to implement the circuit. For example, the number of lines in the circuit shown in figure

2.8 is 4. For a reversible function, the number of lines is equal to the number of inputs. However in the case of classical irreversible functions, ancillary inputs and garbage output are added to realize a reversible circuit which increases this metric. Due to technology limitation, the number of qubits in a quantum circuit is restricted and thus different techniques have been presented and discussed in the literature to reduce the number of lines [41].

2.6 Summary

In this chapter, the preliminaries required to understand the work reported in the following chapters have been presented. The chapter introduced reversible logic functions and their characteristics along with procedure involved in the conversion of irreversible functions in to reversible ones. Different reversible gates and their properties, reversible gate libraries and reversible circuits have also been explained. Finally, mapping of reversible circuit to quantum circuits and cost functions used to evaluate these circuits have been discussed.

Chapter 3

Cube Decomposition Technique for Optimizing Reversible Logic Circuits

3.1 Introduction

As explained in Chapter 1, the existing synthesis algorithms/methods of reversible circuits typically generate sub-optimal circuits and thus optimization methods are applied to reduce circuit's cost. In this work, a decomposition technique that decompose pairs of gates to smaller gates while eliminating redundant gates is proposed. This technique is incorporated in Exclusive-OR Sum-of-Product (ESOP) based synthesis method which is then used to synthesize reversible functions. In the next section the background required to understand the decomposition technique and existing ESOP based synthesis methods are discussed.

3.2 Background

3.2.1 Decomposition of an MCT/MPMCT gate

Any MCT/MPMCT gate can be decomposed into a network of smaller gates [49]. A fundamental decomposition method was presented in Lemma 7.3 of [49] by which an MCT/MPMCT gate of size m (where $m \geqslant 5$) can be decomposed into a network of two gates, each of size p and two gates of size $m - p + 1$ each, given at least one unused line (line which is not a control or target line for the gate) in the circuit.

Example 3.1. Consider an MCT gate having 5 control lines, $G = MCT(x_0, x_1, x_2, x_3, x_4; f_0)$, shown in figure 3.1(a). According to Lemma 7.3 of [49], this gate can be decomposed into a network of four gates of size 3 (i.e., $p = 3$) as shown in figure 3.1(b).

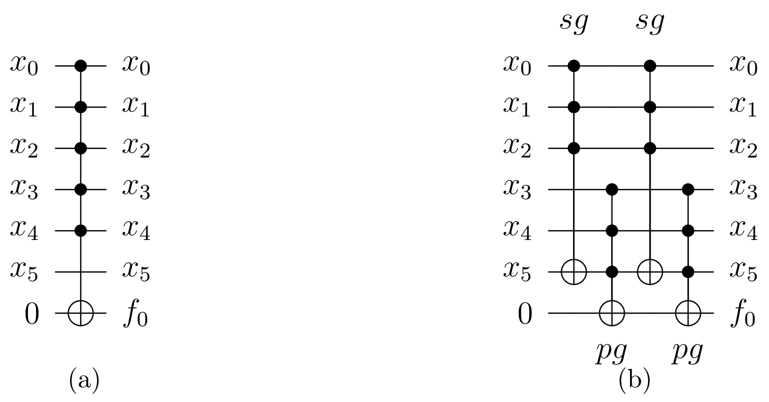


Figure 3.1: Decomposition of MCT gate into network of four smaller gates

In this work, gates in the network that are realized on an unused line are termed as *secondary gates* (represented as sg in figure 3.1(b)) and the gates that are realized on the actual target line, i.e., f_0 are termed as *primary gates* (represented as pg in figure 3.1(b)). The decomposition network of a gate G can be written in terms of 'sg' and 'pg' as:

$$G = sg \circ pg \circ sg \circ pg$$

where '$\circ$' denotes composition i.e. cascading of the gates/circuits.

3.2.2 Exclusive-Or Sum-of-Products (ESOP) based Reversible Circuit Synthesis

3.2.2.1 Exclusive-Or Sum-of-Products (ESOP)

A traditional Sum-of-Product (SOP) is a method of representing a function using AND-OR expression, i.e., OR of several product terms. Likewise, an ESOP is a type of representation in AND-XOR expression, i.e., Exclusive-OR of several product terms [51]. The individual product terms in an ESOP expression are called *cubes* and the variables in its *positive* or *negative* polarity form are called *literal*s. A k-variable function has different ESOP representations and thus many techniques, classified as exact and heuristic, are available in the literature to generate an ESOP expression. Exact technqiues [52–55] have been used to find ESOP expression with the smallest number of cubes but these techniques require large computation time and thus are applicable only for small functions. However, heuristic methods [56, 57] can find an ESOP expression even for large functions.

Example 3.2. Consider a function in SOP form as $f = x_0\overline{x_1} + x_0x_2$. The ESOP equivalent form of this function is $f = x_0\overline{x_1} \oplus x_0x_1x_2$. The term $x_0\overline{x_1}$ and $x_0x_1x_2$ in the function are termed as *cubes* and the variables x_0, x_1, $\overline{x_1}$, x_2 are termed as *literals*.

In general the cubes are represented as a vector of inputs, i.e. for n variables, where $x_0, x_1, \ldots, x_{n-1}$ are inputs and $x_i \in \{0, 1, -\}$, the cube C_i is represented as cube $C_i =< x_0x_1 \ldots x_{n-1} >$. The '$-$' in a cube indicates that the variable at that position has not appeared in the cube and is termed as a '*don't care*' literal. The ESOP functions are represented as a list of cubes called *cube list*s.

Example 3.3. Consider two functions $f_1 = x_0\overline{x_1} \oplus x_0x_1x_2$ and $f_2 = \overline{x_0} \oplus x_0x_1x_2$, where x_0, x_1 and x_2 are input variables and f_1 and f_2 are outputs. The cube list along with the cube outputs and representation of cubes for the functions f_1 and f_2 are shown in figure 3.2.

	x_0	x_1	x_2	f_1	f_2
$C_1(x_0\overline{x_1})$	1	0	$-$	1	0
$C_2(x_0x_1x_2)$	1	1	1	1	1
$C_3(\overline{x_0})$	0	$-$	$-$	0	1

(a)

C_i	$< x_0x_1x_2 >$
C_1	$< 10- >$
C_2	$< 111 >$
C_3	$< 0 - - >$

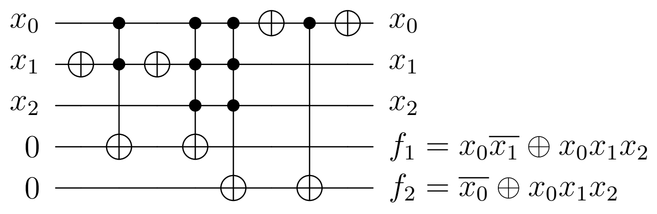
(b)

Figure 3.2: (a) *Cube list* and (b) representation of cubes for functions $f_1 = x_0\overline{x_1} \oplus x_0x_1x_2$ and $f_2 = \overline{x_0} \oplus x_0x_1x_2$

3.2.2.2 ESOP based Reversible Synthesis techniques

The ESOP based synthesis technique for reversible circuits has been introduced in [31] in which reversible as well classical functions can be synthesized in to reversible gate netlist. In this technique, the cubes corresponding to every output variable are mapped to an MCT gate. For example, considering the cube list shown in figure 3.2, the synthesized reversible circuit after applying this technique is shown in figure 3.3.

In order to reduce the quantum cost of synthesized circuits, different techniques have been presented [32–36]. A template matching approach has been presented in [32] by which different templates are applied on a cascade of MCT gates to reduce the quantum cost. A synthesis technique was presented in [33] which uses MPMCT gates



x_0
x_1
x_2
0
0
x_0
x_1
x_2
$f_1 = x_0\overline{x_1} \oplus x_0x_1x_2$
$f_2 = \overline{x_0} \oplus x_0x_1x_2$

Figure 3.3: ESOP based Synthesis Technique presented in [31]

to eliminate the usage of NOT gates for negative literals in the cubes. Further, a set of transformation rules is defined to transfer the targets on outputs to some input lines which helps in reducing the number of lines in the circuit. An ordering based technique has been presented in [34] to reorder the ESOP cubes such that the number of NOT gates is reduced. In addition to this, a set of transformation rules has also been presented to improve the quantum cost of the circuit. Based on these transformation rules, a simulated annealing based approach has been developed in [35] to reduce the quantum cost of circuit. In order to reduce the number of CNOT gates used for sharing MCT gates among their outputs, a shared cube synthesis technique has been presented in [36] which finds multiple outputs that have the largest number of common cubes. These common cubes are first mapped on to respective MCT gates and then CNOT gates are added to share the common cubes between the outputs. All these techniques use EXORCISM-4 tool [57] to generate the cube list from truth tables or SOP expressions. There are few other techniques [58, 59] targeting only reversible circuits to generate cube lists.

3.3 Cube Decomposition Technique

3.3.1 General Idea

There are many optimization methods available in the literature for reversible circuits [17]. One of these methods decomposes an MPMCT gate in a reversible circuit into a network of smaller gates and removes redundant gates, if any. In general, this method is applied independently on individual gates and thus generation of maximum number of redundant gates may not happen. Rather than applying decomposition on individual gates, if the decomposition is applied on a pair of gates with a systematic approach, the chances of generation of redundant gates are higher. For example, considering a pair of gates shown in figure 3.4(a), two different types of decomposition for the same pair are shown in figures 3.4(b) and 3.4(c). The first decomposition is applied independently on individual gates without any constraints whereas the second decomposition is applied with a constraint to generate identical primary gates. The dotted box in figure 3.4(c) contains primary gates that are equal and thus redundant. Hence they can be removed from the circuit thereby reducing the quantum cost of the final decomposed circuit.

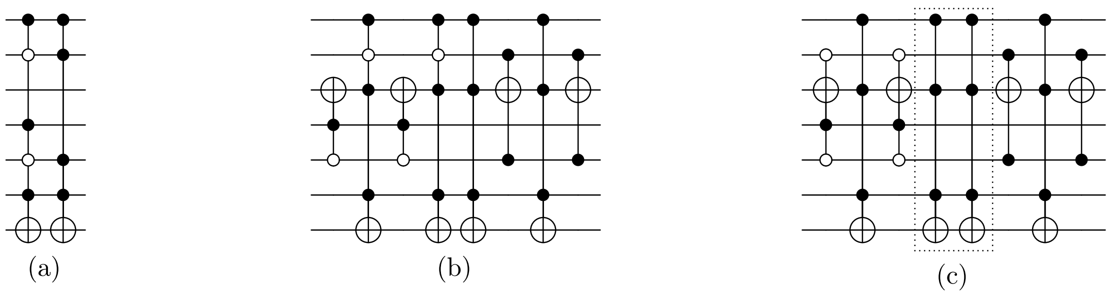


Figure 3.4: (a) Initial Circuit *(QC=78)* (b) Individual Gate decomposition *(QC=98)* (c) Pairwise decomposition *(QC=62)*

The following lemma proves different conditions by which redundant gates can be generated while decomposing a pair of gates.

Lemma 3.1. *Consider two MPMCT gates G_1 and G_2. If the primary (or secondary) gates of two decomposed MPMCT gates, G_1 and G_2, are identical or if one of the primary (or secondary) gates of $G_1(G_2)$ is identical to other gate $G_2(G_1)$, then redundant gates can be generated which can be removed from the circuit.*

Proof. Consider two MPMCT gates G_1 and G_2. The decomposed network of these gates in terms of their primary and secondary gates can be represented as,

$$G_1 = sg_1 \circ pg_1 \circ sg_1 \circ pg_1$$

$$G_2 = sg_2 \circ pg_2 \circ sg_2 \circ pg_2$$

where pg_1, pg_2 represent primary gates and sg_1, sg_2 represent secondary gates for gates G_1 and G_2 respectively. The general circuit realization for the cascade of gates G_1 and G_2 is given as,

$$\begin{aligned} G_1 \circ G_2 &= G_1 \circ G_2^{-1} \quad [\because \text{ MCT gates are self-inverse }] \\ &= sg_1 \circ pg_1 \circ sg_1 \circ pg_1 \circ pg_2 \circ sg_2 \circ pg_2 \circ sg_2 \end{aligned} \tag{3.1}$$

If a pair of gates are equal and are adjacent, they can be removed from the circuit [24]. In Eq. (3.1) the primary gates of two MPMCT gates are equal and adjacent and thus can be removed from the netlist resulting in

$$G_1 \circ G_2 = sg_1 \circ pg_1 \circ sg_1 \circ sg_2 \circ pg_2 \circ sg_2 \tag{3.2}$$

Similarly, if the primary gates of one of the gates is equal to the other undecomposed gate, then $pg_1 \circ G_2 = \varnothing$. Thus, the cascade of gate G_1 and G_2 can be reduced to

$$\begin{aligned} G_1 \circ G_2 &= sg_1 \circ pg_1 \circ sg_1 \circ pg_1 \circ G_2 \\ &= sg_1 \circ pg_1 \circ sg_1 \end{aligned} \tag{3.3}$$

□

Thus from the above lemma, a systematic decomposition of MPMCT gates into a network of smaller gates can result in the reduction of cost of circuit implementation. Even though the decomposition technique discussed earlier is applied on MPMCT gates, it can also be applied on cubes because they can be directly realized using MPMCT gates. This means that the gate decomposition technique can be incorporated in to ESOP based synthesis techniques which are applied on cubes.

The MPMCT gate realization of a cube C, represented as $G(C)$, can be decomposed into primary and secondary gates. The next subsection presents techniques to generate these primary and secondary gates for a given pair of cubes so as to satisfy Lemma 3.1.

3.3.2 Generation of Primary and Secondary gates for a Cube Pair

In order to generate the primary and secondary gates for a given pair of cubes, input variables of these cubes are assigned to three groups namely *equal*, *unequal* and *target groups*. The variables of a cube that are equal to the other cube (with same literal value) are assigned to *equal group* (E) and the variables of a cube that are not equal with the other cube are assigned to *unequal group* (U). The variables that are not present in the cube, i.e., variables with don't care term as values are assigned to *target* *group* (T). The representation for these three groups is given as follows

$$E_a = E_b = \{x_i : a_i = b_i \neq \text{`}-\text{'}\}$$

$$U_a = \{x_i : a_i \neq b_i, a_i \neq \text{`}-\text{'}\} \;\&\; U_b = \{x_i : b_i \neq a_i, b_i \neq \text{`}-\text{'}\}$$

$$T_a = \{x_i : a_i = \text{`}-\text{'}\} \;\&\; T_b = \{x_i : b_i = \text{`}-\text{'}\}$$

where, the groups E_a, U_a and $T_a$correspond to cube C_a and the groups E_b, $U_b$and $T_b$correspond to cube C_b. The elements a_i and b_i indicate the value of input variable x_i in cubes C_a and C_b respectively.

Example 3.4. Consider two cubes, $C_a =< 10-101 >$ and $C_b =< 11--11 >$. The *equal*, *unequal* and *target* groups of this cube pair are given as,

$$E_a = E_b = \{x_0, x_5\}$$
$$U_a = \{x_1, x_3, x_4\} \;\&\; U_b = \{x_1, x_4\}$$
$$T_a = \{x_2\} \;\&\; T_b = \{x_2, x_3\}$$

The variables in target group of a cube are taken as unused lines and they subsequently act as target lines for the generation of secondary gate. Considering that an unused line is represented as x_t and actual target line of an MPMCT gate is represented as t, the primary and secondary gates in terms of *equal* and *unequal* groups are given as,

$$pg = MPMCT(E, x_t; t) \;\&\; sg = MPMCT(U; x_t)$$

From the above expressions, E, U and t are known variables and the only unknown variable that needs to be derived is x_t. Depending on the availability of x_t in target groups of a cube pair, there arise different cases that generate primary and secondary gates. These cases are discussed in the following subsections.

3.3.2.1 Case 1:$T_a \cap T_b \neq \phi$

If $T_a \cap T_b \neq \phi$, then one of the variables (denoted as x_t) from the set $T_a \cap T_b$ can be chosen as target line for generation of secondary gates. Since a secondary gate's target line is also a primary gate's control line, x_t is added to the list of control lines of primary gates. Thus, the final primary and secondary gates representations are

$$sg_1 = MPMCT(U_a; x_t)\ ,\ sg_2 = MPMCT(U_b; x_t)$$

$$pg_1 = pg_2 = MPMCT(E_a, x_t; t)$$

The following example illustrates generation of primary and secondary gates for a cube pair that satisfies this case.

Example 3.5. Consider two cubes $C_a =< 10-101 >$ and $C_b =< 11--11 >$. The *equal*, *unequal* and *target* groups of this cube pair are

$$E_a = E_b = \{x_0, x_5\},$$

$$U_a = \{x_1, x_3, x_4\} \ \&\ U_b = \{x_1, x_4\},$$

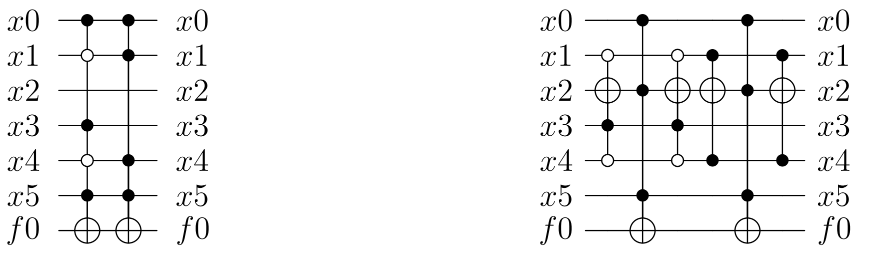
$$T_a = \{x_2\} \ \&\ T_b = \{x_2, x_3\}$$

From the target groups T_a and T_b, it is evident that the variable x_2 is common and can be used as a target line in generating secondary gates. Now, the primary and secondary gates are

$$sg_1 = MPMCT(x_1, x_3, x_4; x_2)\ ,\ sg_2 = MPMCT(x_1, x_4; x_2)$$

$$pg_1 = pg_2 = MPMCT(x_0, x_5, x_2; f_0)$$

Since the primary gates of the cube pair are equal, the equivalent cube pair realization is $G(C_a) \circ G(C_b) = sg_1 \circ pg_1 \circ sg_1 \circ sg_2 \circ pg_2 \circ sg_2$ from Lemma 3.1. The final circuit implementation of the cube pair C_a and C_b is shown in figure 3.5.



(a) Circuit before Decomposition (b) Circuit after Decomposition

Figure 3.5: Gate decomposition for a pair of cubes with the same empty line as target line

3.3.2.2 Case 2:$T_a \cap T_b = \phi$

The previous case can be used only if there exists at least one common variable in both the target groups of cube pair. In this subsection, another technique is presented when there are no common variables in target groups, i.e. $T_a \cap T_b = \phi$. Here, two variables x_{ta} and x_{tb} are chosen from groups T_a and T_b respectively. These variables act as empty lines for generation of secondary gates and also as control lines for primary gates for their respective cubes. Thus the primary gates can be represented as

$$pg_1 = MPMCT(E_a, x_{ta}; t)\ \&\ pg_2 = MPMCT(E_b, x_{tb}; t)$$

From the above equations, it is evident that the primary gates of the cube pair are not equal. In order to make them equal, the variable x_{tb} is added to primary gate of the cube C_a and the variable x_{ta} is added to primary gate of the cube C_b. Since x_{tb} is a control line in primary gate of cube C_a, it can be removed from the unequal group. Similarly, the variable x_{ta} can be removed from the unequal group of the cube C_b. Thus the final gate realization of primary and secondary gates is

$$pg_1 = pg_2 = MPMCT(E_a, x_{ta}, x_{tb}; t)$$

$$sg_1 = MPMCT(U_a - \{x_{tb}\}; x_{ta}) \text{ \& } sg_2 = MPMCT(U_b - \{x_{ta}\}; x_{tb})$$

The following example illustrates the generation of primary and secondary gates for a cube pair with no common variables in their target groups.

Example 3.6. Consider two cubes $C_a =< 10-101 >$ and $C_b =< 1-1011 >$. The *equal*, *unequal* and *target* groups of these cubes are:

$$E_a = E_b = \{x_0, x_5\}$$

$$U_a = \{x_1, x_3, x_4\} \text{ \& } U_b = \{x_2, x_3, x_4\}$$

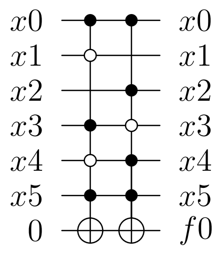
$$T_a = \{x_2\} \text{ \& } T_b = \{x_1\}$$

There is no common variable that exists in the target groups, T_a and T_b. Thus, x_2 is selected for the generation of secondary gate for the cube C_a and x_1 for the cube C_b, resulting in the primary and secondary gates as,

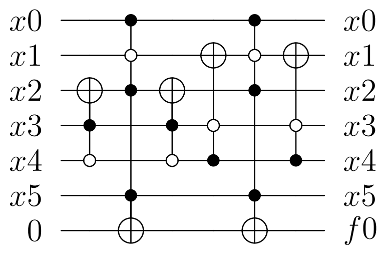
$$sg_1 = MPMCT(x_3, x_4; x_2), sg_2 = MPMCT(x_3, x_4; x_1)$$

$$pg1 = pg2 = MPMCT(x_0, x_5, x_1, x_2; f_0)$$

Since the primary gates of the cube pair are now equal, the cubes can be implemented as $G(C_a) \circ G(C_b) = sg_1 \circ pg_1 \circ sg_1 \circ sg_2 \circ pg_2 \circ sg_2$ following Lemma 3.1. The final circuit implementation for the cube pair C_a and C_b is shown in figure 3.6.



(a) Circuit before Decomposition



(b) Circuit after Decomposition

Figure 3.6: Gate decomposition for a cube pair with different empty lines as target lines

3.3.2.3 Case 3:$pg_a = G(C_b)$ or $sg_a = G(C_b)$

As proved in Lemma 3.1, in a cascade of two MPMCT gates, if one of the gates' primary gate is equal to the other gate, the cost of circuit realization is reduced. For a pair of cubes, this condition is satisfied when one of the cubes' unequal group has only one variable (denoted as x_t) and the same is present in the target group of the other cube. The cube with x_t in target group is decomposed in to primary and secondary gates as,

$$sg_1 = MPMCT(U_a; x_t)$$

$$pg_1 = MPMCT(E_a, x_t; t)$$

The following example illustrates the generation of primary and secondary gates for a cube pair with the condition discussed above.

Example 3.7. Consider two cubes $C_a =< 110-1 >$ and $C_b =< 1--11 >$. The *equal*, *unequal* and *target* groups for this cube pair are

$$E_a = E_b = \{x_0, x_4\}$$

$$U_a = \{x_1, x_2\} \ \& \ U_b = \{x_3\}$$

$$T_a = \{x_3\} \ \& \ T_b = \{x_1, x_2\}$$

The unequal group of cube C_b has only one variable, x_3, which is also present in the target group of cube C_a. This variable can now act as a target line to generate secondary gate for cube C_a and is expressed as,

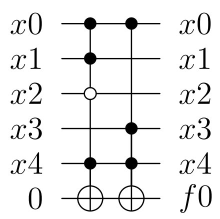
$$sg_1 = MPMCT(x_1, x_2; x_3)$$

$$pg_1 = MPMCT(x_0, x_4, x_3; f_0)$$

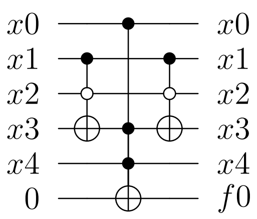
The gate implementation of cube C_b is $G(C_b) = MPMCT(x_0, x_3, x_4; t)$. Since the primary gate of one cube is equal to other cube's gate implementation, the cube pair can be realized as $G_1 \circ G_2 = sg_1 \circ pg_1 \circ sg_1$ following Lemma 3.1. The final circuit implementation for the cube pair C_a and C_b is shown in figure 3.7.

3.3.3 Algorithm for Cube Decomposition

In this section, the algorithm for cube decomposition technique is presented. The input to the algorithm is *cube_list*, which has a list of cubes and output is reversible gate



(a) Before Cube Decomposition



(b) After Cube Decomposition

Figure 3.7: Gate decomposition for a cube pair with $pg_a = G(C_b)$

net-list *GateNetlist*. In the algorithm, initially a cube C_i from the given *cube_list* is extracted. This cube is checked with the other cubes in the *cube_list* for any possible decomposition using the function *CheckingDecomposition*(). This function returns true if the pair of cubes can be decomposed with any one of the proposed decomposition techniques else it returns false.

If the return value is false, then the gate realization of cube C_i is added to the *GateNetlist*. If the return value is true, the decomposed gate netlist, $List_{deco}$, and its cost, $Cost_{deco}$, are obtained using the function *Decomposition*(). After all the cubes in the *cube_list* are checked for decomposition condition, the cube which has the least circuit cost after decomposition is termed as *BestCube* and its cost as *BestCost*. If the *BestCost* is less than the sum of the costs of gate realization of cube C_i and *BestCube*, the decomposed netlist of *BestCube* is assigned to *BestDeco* and is appended to the *GateNetlist*. If the *BestCost* is greater than the cost of gate realization of cube C_i plus the cost of *BestCube*, the gate realization of cube C_i is added to the *GateNetlist*. The pseudo code of this algorithm is given below:

This algorithm is illustrated with the following example.

Example 3.8. Consider the cube list, *cube_list*, with four cubes, C_1, C_2, C_3 and C_4, shown in figure 3.8 which is given as input to Algorithm 3.1.

In the first iteration, C_1 is removed from the *cube_list* and is checked for decom-

Algorithm 3.1 Cube Decomposition Algorithm

```
1: Input: $cube\_list$
2: Output: Reversible $GateNetlist$
3: begin
4: while $cube\_list$ is not empty do
5:     $flag = false$
6:     $C_i = pop(cube\_list)$
7:     $BestCost = \infty$
8:     for each cube $C_j \in cube\_list$ do
9:         if $CheckDecomposition(C_i, C_j)$ is $true$ then
10:            $Cost_{deco}, List_{deco} = Decomposition(C_i, C_j)$
11:            $InitCost = MCTCost(C_i) + MCTCost(C_j)$
12:            if $min\{BestCost, InitCost\} > Cost_{deco}$ then
13:                $flag = true$
14:                $BestCost = Cost_{deco}$
15:                $BestCube = C_j$
16:                $BestDeco = List_{deco}$
17:            end if
18:        end if
19:    end for
20:    if $flag$ is $true$ then
21:        $AddToArray(GateNetlist, BestDeco)$
22:        $DeleteCube(cube\_list, BestCube)$
23:    else
24:        $AddToArray(GateNetlist, GenerateMCT(C_i))$
25:    end if
26: end while
27: return $GateNetlist$
28: end
```

position with every other cube in the *cube_list* . First, the cube C_2 is checked with C_1 for decomposition using the function *CheckDecomposition*. Since none of the cases discussed above is satisfied, this function return false and moves to next cube in the list i.e., C_3. Similar to C_2, the cube C_3 does not satisfy any of the cases, thus resulting in *CheckDecomposition* function returning false. However, this function returns true when checked with the cube C_4 because it satisfies the condition described in *case 3*. Hence, the function *Decomposition* returns the decomposed gate netlist and its cost (27) after applying the decomposition technique as shown in figure 3.9. This cost is less than the sum of cost of implementation of cubes C_1 and C_4 individually i.e. 42 setting

	x_0	x_1	x_2	x_3	x_4	f
C_1	0	–	0	0	1	1
C_2	1	1	1	1	–	1
C_3	1	1	–	–	–	1
C_4	–	1	0	–	1	1

(a)

Figure 3.8: *Cube list*

the flag to *True*. Now, the cube C_4 is assigned as *BestCube*, its decomposed netlist as *BestDeco* and its cost as *BestCost*. After all the cubes in the list are checked with C_1, the flag is checked for true condition. Since the flag is true in this example, the *BestDeco* is added to the output netlist, *GateNetlist*, and the corresponding cube i.e. C_4 is removed from the *cube_list*. After first iteration, the status of *cube_list* and *GateNetlist* is shown in figure 3.10.

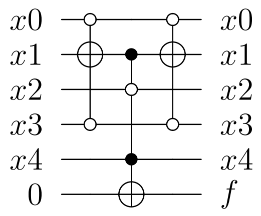
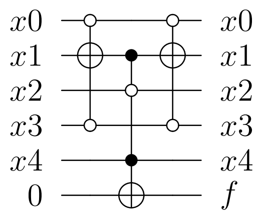


Figure 3.9: Decomposed gate netlist for the cube pair C_1 and C_4

	x_0	x_1	x_2	x_3	x_4	f
C_2	1	1	1	1	–	1
C_3	1	1	–	–	–	1

(a) Updated Cube List *cube_list*



(b) Output Gate Netlist *GateNetlist*

Figure 3.10: Status of cube list and output gate netlist after iteration 1

In the second iteration, C_2 is removed from the *cube_list* and is checked with cube C_3. Since this pair satisfies the condition described in *case 1*, the *Decomposition* function returns the decomposed gate netlist and its cost (38) as shown in figure 3.11. Since this cost is more than the cost of implementation of cubes C_2 and C_3 individually i.e. 34, setting the flag to *False*. As there are no more cubes left in *cube_list* and the flag is false, the gate implementation of cube C_2 is added to *GateNetlist*. After second iteration, the status of *cube_list* and *GateNetlist* is shown in figure 3.12.

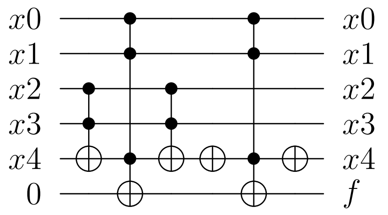
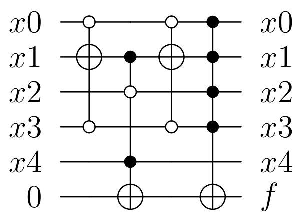


Figure 3.11: Decomposed gate netlist for the cube pair C_2 and C_3

	x_0	x_1	x_2	x_3	x_4	f
C_3	1	1	$-$	$-$	$-$	1

(a) Updated Cube List *cube_list*



(b) Output Gate Netlist *GateNetlist*

Figure 3.12: Status of cube list and output gate netlist after iteration 2

In the final iteration, there is only one cube left in the list and thus the gate realization of this cube is added to the *GateNetlist* as shown in figure 3.13.

3.4 Simulation Results and Comparisons

The proposed algorithm has been used to implement various reversible benchmark functions available in RevLib [50] and compared with existing ESOP based reversible

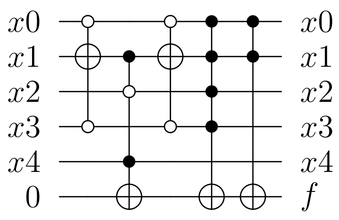


Figure 3.13: Output gate netlist after Final Iteration

circuit synthesis techniques such as [32,35,36,58,59]. The cubes list can be generated either by EXORCISM-4 tool or other cube list generation techniques [58, 59]. To evaluate the proposed technique, quantum cost reduction obtained by existing cube transformation techniques that use EXORCISM-4 tool are considered.

Table 3.1 compares the quantum cost of related and existing ESOP based synthesis techniques with the proposed one. The first column indicates the benchmark name in alphabetical order. Columns 2-6 indicate the quantum cost of existing ESOP based synthesis techniques for the corresponding benchmarks. The techniques presented in columns 2-4 use EXORCISM-4 tool to generate cube list whereas techniques presented in columns 5 and 6 use their own methods to generate the cube lists. Column 7 indicates the quantum cost of the corresponding benchmark using the proposed technique. The final column shows the percentage improvement of quantum cost achieved compared to the existing techniques that give the best reduction for that benchmark.

For the benchmarks considered, there is a reduction in quantum cost for most of the benchmarks when the proposed technique is applied as compared to existing ones. Benchmarks like *misex3*, *rd84*, *cordic*, *alu4*, *spla*, *table3* have improved by more than 14% when compared to existing synthesis techniques. Further, the proposed algorithm consumes a very less synthesis time with a maximum of 5 seconds for *frg2* benchmark when implemented in Python programming language on a workstation with Intel E3-1220 processor with 8GB of primary memory running Windows 7. For few benchmarks, the proposed technique results in more quantum cost when compared to [59] because their cube list generation method is different from the one followed in this work.

Table 3.1: Comparison with Existing ESOP Based Synthesis Methods

Benchmark	**[32]**	**[36]**	**[35]**	**[58]**	**[59]**	**Proposed Technique**	**% Impr**
5xp1	1349	786	807	865	-	759	**3.44**
9sym	5781	10943	3406	16487	1895	2222	-17.26
add6	6362	-	-	5084	2683	3629	-35.26
alu1	243	-	-	-	156	156	**0.00**
alu2	5215	-	3679	4476	-	3458	**6.01**
alu3	2653	-	1919	-	-	1828	**4.74**
alu4	48778	41127	38635	43850	-	31220	**19.19**
apex4	256857	35840	-	50680	51284	37018	-3.29
apex5	-	33830	33803	-	-	29842	**11.72**
apla	4051	1683	1709	-	-	1601	**4.87**
bw	-	637	790	-	2233	649	-1.88
C17	97	-	-	-	-	78	**19.59**
clip	6616	3824	3218	4484	-	2889	**10.22**
cm150a	844	-	-	-	-	785	**6.99**
con1	207	162	-	-	-	150	**7.41**
cordic	349522	187620	111955	-	-	91935	**17.88**

Table 3.1 Comparison with Existing ESOP Based Synthesis Methods (Continued)

Benchmark	**[32]**	**[36]**	**[35]**	**[58]**	**[59]**	**Proposed Technique**	**% Impr**
cu	1332	781	780	-	-	747	**4.23**
dc2	1956	1084	1099	-	-	1019	**6.00**
decod	1924	399	-	-	976	436	-9.27
dist	7414	3700	-	-	-	3367	**9.00**
e64	-	-	24345	-	-	23751	**2.44**
ex1010	183726	52788	-	-	77293	52467	**0.61**
ex2	153	-	-	-	118	140	-18.64
ex3	97	-	-	-	73	59	**19.18**
f2	274	112	-	-	116	107	**4.46**
f51m	34244	28382	25119	-	-	22042	**12.25**
frg2	-	112008	114239	-	-	97183	**13.24**
in0	22196	7949	-	-	-	7501	**5.64**
majority	147	-	-	-	106	106	**0.00**
max46	4432	-	-	-	3239	2875	**11.24**
misex1	1017	332	352	466	-	338	-1.81
misex3	122557	49076	54132	67206	-	42098	**14.22**
misex3c	118578	49720	-	85330	52600	42868	**13.78**
mlp4	3827	2496	-	-	-	2303	**7.73**

Table 3.1 Comparison with Existing ESOP Based Synthesis Methods (Continued)

Benchmark	**[32]**	**[36]**	**[35]**	**[58]**	**[59]**	**Proposed Technique**	**% Impr**
mux	826	-	-	-	784	768	**2.04**
pm1	582	-	-	-	290	188	**35.17**
radd	798	-	-	-	349	316	**9.46**
rd84	2598	-	1965	2062	-	1687	**14.15**
root	3486	1811	1583	-	-	1533	**3.16**
sao2	7893	3767	-	5147	-	3244	**13.88**
spla	-	-	45478	49419	-	28220	**37.95**
sqn	2170	-	-	-	1183	1222	-3.30
sqr6	1090	583	-	-	-	597	-2.40
sqrt8	584	-	-	461	-	314	**31.89**
squar5	476	-	-	251	-	231	**7.97**
t481	237	-	-	237	-	205	**13.50**
table3	86173	-	32286	35807	-	17454	**45.94**
urf3	-	53157	-	-	56766	51622	**2.89**
z4	674	489	-	-	260	388	-49.23

3.5 Summary

In this chapter, a decomposition technique has been presented to decompose a pair of gates into a set of smaller gates while eliminating redundant gates. This technique has been incorporated into ESOP synthesis techniques, since the cubes on which ESOP synthesis techniques are applied can be directly realized using the gates. A cube decomposition algorithm has been presented in which the decomposition technique is applied along with an ESOP based synthesis method to improve the synthesized gate netlist. This algorithm has been applied on a set of benchmarks to validate its efficiency. Results show a reduction in the quantum cost of several benchmark circuits when compared to the known ESOP based synthesis methods.

Chapter 4

Complementary Control Line Transformation for Optimizing Reversible Logic Circuits

4.1 Introduction

The decomposition technique presented in the previous chapter depends on the number of equal control lines in a reversible gate pair. In this chapter, an algorithm to transform a set of control lines of a gate pair to equal control lines is proposed. A set of rules is then applied on the transformed gate netlist to reduce the circuit's cost. Finally, an optimization method which integrates the algorithm and the rules to optimize a given gate netlist is discussed.

4.2 Motivation

A decomposition rule is derived using the decomposition technique presented in Section 3.3 and is given below:

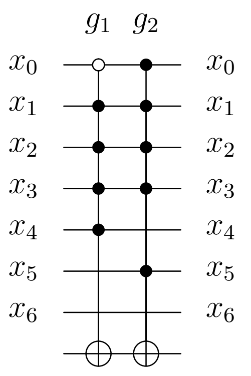
Decomposition Rule: Consider a pair of gates $g_a = MCT(E \cup U_a; x_t)$ and $g_b = MCT(E \cup U_b; x_t)$ where set E contains equal control lines and sets U_a and U_b contain control lines that are unequal with respect to other gate. If an unused line x_u is available, then the gate pair can be decomposed as:

$$\begin{aligned} g_a \circ g_b = MCT(U_a; x_u) \circ MCT(E \cup \{x_u\}; x_t) \circ MCT(U_a; x_u) \circ \\ MCT(U_b; x_u) \circ MCT(E \cup \{x_u\}; x_t) \circ MCT(U_b; x_u) \end{aligned} \quad (4.1)$$

This decomposition rule is illustrated with the help of the following example:

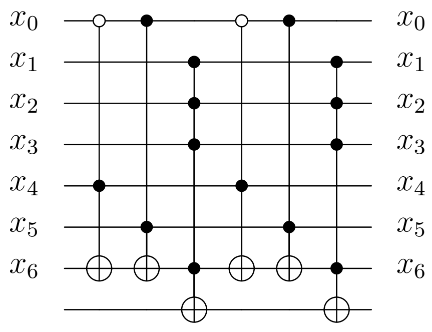
Example 4.1. Consider the gate pair g_1 and g_2 shown in figure 4.1(a). These gates have the same target line, three equal control lines, one unequal control line. Line x_4 has control connection only in g_1 while line x_5 has control connection only in g_2. As can be observed, line x_6 is unused line for this gate pair. Using the decomposition rule given by Eq. 4.1, these gates can be decomposed into a network of smaller gates. The resulting decomposed gate netlist is shown in figure 4.1(b).

It should be mentioned that the decomposition rule results in reduced quantum cost only if the number of unequal control lines in the gate pair,i.e. U_a or U_b, is less than half the total number of lines in the circuit. If this is not the case, then the rule results in a higher-cost gate netlist when compared with the quantum cost of individual gates. This result can be proved with the help of following lemma:


g_1 g_2
x_0 x_1 x_2 x_3 x_4 x_5 x_6
x_0 x_1 x_2 x_3 x_4 x_5 x_6

$Cost = 104$

(a) Initial Gate Pair


x_0 x_1 x_2 x_3 x_4 x_5 x_6
x_0 x_1 x_2 x_3 x_4 x_5 x_6

$Cost = 72$

(b) Decomposed Gate Netlist

Figure 4.1: Illustration of Decomposition Rule

Lemma 4.1. *In a circuit that has n lines, consider two gates $g_a = MCT(E \cup U_a; x_t)$ and $g_b = MCT(E \cup U_b; x_t)$ where set E contains equal control lines and sets U_a and U_b contains unequal control lines and an unused line x_u. The decomposition rule results in higher-cost gate netlist when compared to the cost of individual gates in the following conditions:*

1. $|U_a| \leq \lceil \frac{n+1}{2} \rceil$, $|U_b| \leq \lceil \frac{n+1}{2} \rceil$ *and* $|E| \leq 1$
2. $|U_a| > \lceil \frac{n+1}{2} \rceil$ *and/or* $|U_b| > \lceil \frac{n+1}{2} \rceil$

where $|X|$ indicates the number of elements in the set X, i.e. Cardinality of the set

Proof. The cascade of gates g_a and g_b can be given as:

$$g_a \circ g_b = MCT(E \cup U_a; x_t) \circ MCT(E \cup U_b; x_t) \tag{4.2}$$

Consider $|E| + |U_a| > \lceil \frac{n+1}{2} \rceil$ and $|E| + |U_b| > \lceil \frac{n+1}{2} \rceil$, the NCT cost (Eq. (2.9)) to implement the gate pair g_a and g_b is

$$\begin{aligned} &8(|E| + |U_a| - 3) + 8(|E| + |U_b| - 3) \\ =&16|E| + 8|U_a| + 8|U_b| - 48 \end{aligned} \tag{4.3}$$

The decomposed gate netlist after applying decomposition rule on gate pair g_a and g_b can be given as:

$$\begin{aligned} g_a \circ g_b \;=\;& MCT(U_a; x_u) \circ MCT(E \cup \{x_u\}; x_t) \circ MCT(U_a; x_u) \circ \\ & MCT(U_b; x_u) \circ MCT(E \cup \{x_u\}; x_t) \circ MCT(U_b; x_u) \end{aligned} \tag{4.4}$$

In order to replace the pair of gates, the decomposed gate netlist should have lesser cost when compared to the cost of the gate pair. The NCT cost (Eq. (2.9)) of the decomposed gate netlist for different cases is given below:

Case 1: If $|U_a| \leq \lceil \frac{n+1}{2} \rceil$, $|U_b| \leq \lceil \frac{n+1}{2} \rceil$ and $|E| > \lceil \frac{n+1}{2} \rceil$ then the NCT cost of the decomposed gate netlist is

$$\begin{aligned} & 2(4(|U_a| - 2) + 4(|U_b| - 2) + 8(|E| + 1 - 3)) \\ =& 16|E| + 8|U_a| + 8|U_b| - 64 \end{aligned} \tag{4.5}$$

In order to replace the pair of gates with the decomposed gate netlist, the cost computed by Eq. (4.5) should be less than the cost computed by Eq. (4.3)

$$\begin{aligned} \implies 16|E| + 8|U_a| + 8|U_b| - 64 \;&<\; 16|E| + 8|U_a| + 8|U_b| - 48 \\ -64 \;&<\; -48 \end{aligned}$$

Thus, it follows from the above inequalities that the gate pair can be replaced in this case by the decomposed gate netlist.

Case 2: If $|U_a| \leq \lceil \frac{n+1}{2} \rceil$, $|U_b| \leq \lceil \frac{n+1}{2} \rceil$ and $|E| \leq \lceil \frac{n+1}{2} \rceil$ then the NCT cost of the decomposed gate netlist is

$$\begin{aligned} & 2(4(|U_a| - 2) + 4(|U_b| - 2) + 4(|E| + 1 - 2)) \\ =& 8|E| + 8|U_a| + 8|U_b| - 40 \end{aligned} \tag{4.6}$$

In order to replace the pair of gates with the decomposed gate netlist, the cost computed by Eq. (4.6) should be less than the cost computed by Eq. (4.3)

$$\begin{aligned} \implies 8|E| + 8|U_a| + 8|U_b| - 40 &< 16|E| + 8|U_a| + 8|U_b| - 48 \\ |E| &> 1 \end{aligned}$$

Thus, from the above inequalities the gate pair can be replaced by the decomposed gate netlist only if $|E| > 1$ proving the first condition of the lemma.

Case 3: If $|U_a| > \lceil \frac{n+1}{2} \rceil$, $|U_b| > \lceil \frac{n+1}{2} \rceil$ and $|E| \leq \lceil \frac{n+1}{2} \rceil$ then the NCT cost of the decomposed gate netlist is

$$\begin{aligned} & 2(8(|U_a| - 3) + 8(|U_b| - 3) + 4(|E| + 1 - 2)) \\ =& 8|E| + 16|U_a| + 16|U_b| - 104 \end{aligned} \tag{4.7}$$

In order to replace the pair of gates with the decomposed gate netlist, the cost computed by Eq. (4.7) should be less than the cost computed by Eq. (4.3)

$$\begin{aligned} \implies 8|E| + 16|U_a| + 16|U_b| - 104 &< 16|E| + 8|U_a| + 8|U_b| - 48 \\ |E| + 7 &< |U_a| + |U_b| \qquad (4.8) \end{aligned}$$

In this case, the minimum value of $|U_a|$ (and $|U_b|$) and the maximum value of $|E|$ in terms of n are $\lceil \frac{n+1}{2} \rceil + 1$ and $\lceil \frac{n+1}{2} \rceil$ respectively. Substituting these value in Eq. (4.8) results in $n < 9$. Thus, for the maximum value of n i.e. 8, excluding target line and unused line utmost 6 control lines are possible for a gate. However for $n = 8$, the minimum value of $|U_a|$ is 6 which is the maximum number of control lines allowed in this case, resulting in $|E| = 0$. The condition for applying decomposition rule is $|E| > 0$ and thus proving the second condition of the lemma. □

The above lemma shows that if $|U_a| > \lceil \frac{n+1}{2} \rceil$ and $|U_b| > \lceil \frac{n+1}{2} \rceil$ then the decomposition rule cannot obtain the cost reduction. However, if the values of $|U_a|$ and $|U_b|$ are reduced such that $|U_a| \leq \lceil \frac{n+1}{2} \rceil$ and $|U_b| \leq \lceil \frac{n+1}{2} \rceil$ then the proposed decomposition rule can be applied to achieve the cost reduction. In order to reduce the values of $|U_a|$ and $|U_b|$, we propose a transformation technique which is presented in the following section.

4.3 Complementary Control Line Transformation

The transformation technique converts complementary control lines (CCLs) to equal control lines (ECLs) by adding extra gates to the circuit. This technique is explained in Lemma 4.2 given below and the transformation rules (illustrated in Fig. 4.2) used in this lemma are described as follows:

T1 $MCT(X \cup \{x_i, x_j\}; x_t) = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j)$

T2 $MCT(X \cup \{x_i, \overline{x_j}\}; x_t) = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, x_j\}; x_t) \circ MCT(\{x_i\}; x_j)$

T3 $MCT(X \cup \{\overline{x_i}, x_j\}; x_t) = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{\overline{x_i}, x_j\}; x_t) \circ MCT(\{x_i\}; x_j)$

T4 $MCT(X \cup \{\overline{x_i}, \overline{x_j}\}; x_t) = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{\overline{x_i}, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j)$

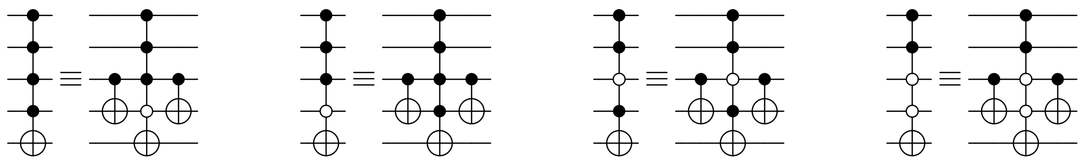


Figure 4.2: Illustration of Transformation Rules

Lemma 4.2 (CCL Transformation Rule). *Consider a pair of gates* $g_i = MCT(X \cup \{x_i, x_j\}; x_t)$ *and* $g_j = MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t)$*. The CCL* x_j *can be transformed to ECL using two extra CNOT gates and the resulting cascade of gates,* g_i *and* g_j*, is given as:*

$g_i \circ g_j = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j)$

Proof. The gates g_i and g_j can be represented as:

$$g_i = MCT(X \cup \{x_i, x_j\}; x_t)$$

$$g_j = MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t)$$

The transformation rules, T1, T2, T3 and T4, are applied on the gates depending on the control connections of lines x_i and x_j. Since gate g_i has $\{x_i, x_j\}$ and gate g_j has $\{\overline{x_i}, \overline{x_j}\}$, the transformation rules **T1** and **T4** are applied on the gates g_i and g_j respectively resulting in:

$$g_i = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j) \tag{4.9}$$

$$g_j = MCT(\{x_i\}; x_j) \circ MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j) \tag{4.10}$$

The cascade of gates, g_i and g_j, using Eqs. (4.9) and (4.10) is shown below:

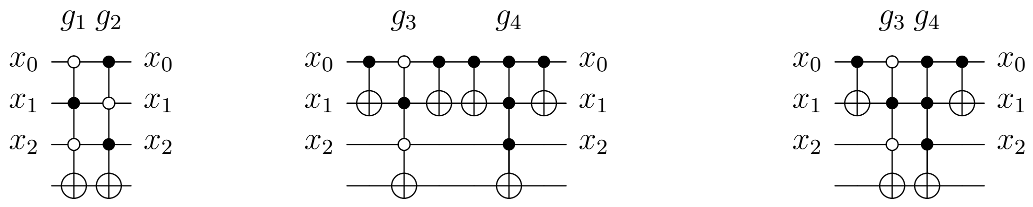
$$\begin{aligned} g_i \circ g_j \quad = \quad & MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j) \circ MCT(\{x_i\}; x_j) \circ \\ & MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j) \end{aligned}$$

The third and fourth gates in the above gate netlist are same and can be removed. The resulting gate netlist can be given as:

$$g_i \circ g_j = MCT(\{x_i\}; x_j) \circ MCT(X \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(Y \cup \{\overline{x_i}, \overline{x_j}\}; x_t) \circ MCT(\{x_i\}; x_j) \tag{4.11}$$

From Eq. (4.11), the control line x_j is transformed to ECL using two CNOT gates. It may be noted however that the line x_i cannot be transformed to ECL because it has to be used as a control line for the CNOT gates that are added. □

Example 4.2. Consider the gate pair shown in figure 4.3(a). This gate pair has three CCLs (x_0, x_1, x_2). Using the transformation rules T2 and T3, the gates g_1 and g_2 can be transformed to a gate netlist shown in figure 4.3(b). The CNOT gates that are adjacent to each other can be removed resulting in the gate netlist shown in figure 4.3(c). From the figures it can be seen that the CCL x_1 has transformed to an ECL.



(a) Original Gate Pair (b) After applying Transformation Rules T2 and T3 on gates g_1 and g_2 (c) Adding CNOT gates for line x_1

Figure 4.3: Illustration of Example 4.2

The above lemma and example explain the transformation of one CCL to ECL. In the next section, an algorithm is presented to convert more than one CCL. It has to be noted however that, in a gate pair with k CCLs, at most $k-1$ CCLs can be transformed to equal control lines. The remaining CCL cannot be transformed because it has to be used as a control line for the CNOT gates that are added. After the transformation, different rules can be applied on the transformed gate pairs to reduce their cost.

4.4 CCL Transformation Algorithm

The method for converting CCLs to ECLs for a given pair of gates is presented in Algorithm 4.1. This algorithm takes two gates G_i and G_j as inputs and returns a gate netlist G_t as output. The set of CCLs (represented as *ccl*), if any, in the given gate pair are extracted using the function *ExtractComplementLines*. From *ccl*, a line is randomly selected as the *baseline* using the function *random* and is subsequently removed from *ccl*. Now, for each line l_i in *ccl*, a CNOT gate with the *baseline* as control line and l_i as the target line is added to an intermediate gate netlist called *CG*. The function *value* determines whether the control connection of a line in a gate is positive or negative control. The line l_i in gates G_i and G_j is set to negative control if it has the same control connection as the *baseline*, else is set to positive control. This updates the initial gate pair G_i and G_j. Finally, the output gate netlist G_t is generated by cascading the CG before and after the updated gate pair G_i and G_j. The proposed gate transformation algorithm is illustrated in example 4.3.

Algorithm 4.1 Gate Transformation Algorithm for a Pair of Gates

```
1: **Input:** Gates $G_i, G_j$
2: **Output:** Gate Netlist $G_t$
3: **begin**
4: $ccl = ExtractComplementLines(G_i, G_j)$
5: **if** $ccl = \phi$ **then exit**
6: $baseline = random(ccl)$
7: $ccl = ccl - \{baseline\}$
8: $CG = \phi$
9: **for each** $l_i \in ccl$ **do**
10:     $append(CNOT(baseline, l_i), CG)$
11:     **if** $value(G_i, baseline) == value(G_i, l_i)$ **then**
12:         $G_i(l_i) = G_j(l_i) = NegativeControl$
13:     **else**
14:         $G_i(l_i) = G_j(l_i) = PositiveControl$
15:     **end if**
16: **end for**
17: $G_t = append(CG, G_i, G_j, CG)$
18: **return** $G_t$
19: **end**
```

Example 4.3. Consider two gates g_i and g_j shown in Fig. 4.4(a), given as inputs to Algorithm 4.1. First, the function *ExtractComplementLines* extracts the lines x_0, x_1, x_2 as CCLs for the gate pair and assigns them to the set *ccl* ($ccl = \{x_0, x_1, x_2\}$). Assuming that the *random* function selects the line x_0 as the *baseline* from *ccl*, it is removed from *ccl* and is updated to $\{x_1, x_2\}$.

In the first iteration, line x_1 from the set *ccl* is selected as line l_i. A CNOT gate with x_0 (the *baseline*) as control line and x_1 (line l_i) as target line is added to the gate netlist CG. Since the function *value* returns the control connections of x_0 and x_1 in gate g_i as negative and positive control respectively, the line x_1 in gates g_i and g_j is updated to positive control. At the end of the first iteration, status of the gate netlist CG and that of gates g_i and g_j is shown in Fig. 4.4(b).

The algorithm then proceeds to update the gate netlist CG and the gate pair g_i, g_j for the remaining lines in the *ccl*. Figure 4.4(c) shows the status after the second iteration. Finally, the gate netlist CG is added before and after the updated gates g_i and g_j to form the output gate netlist G_t as shown in Fig. 4.4(d).

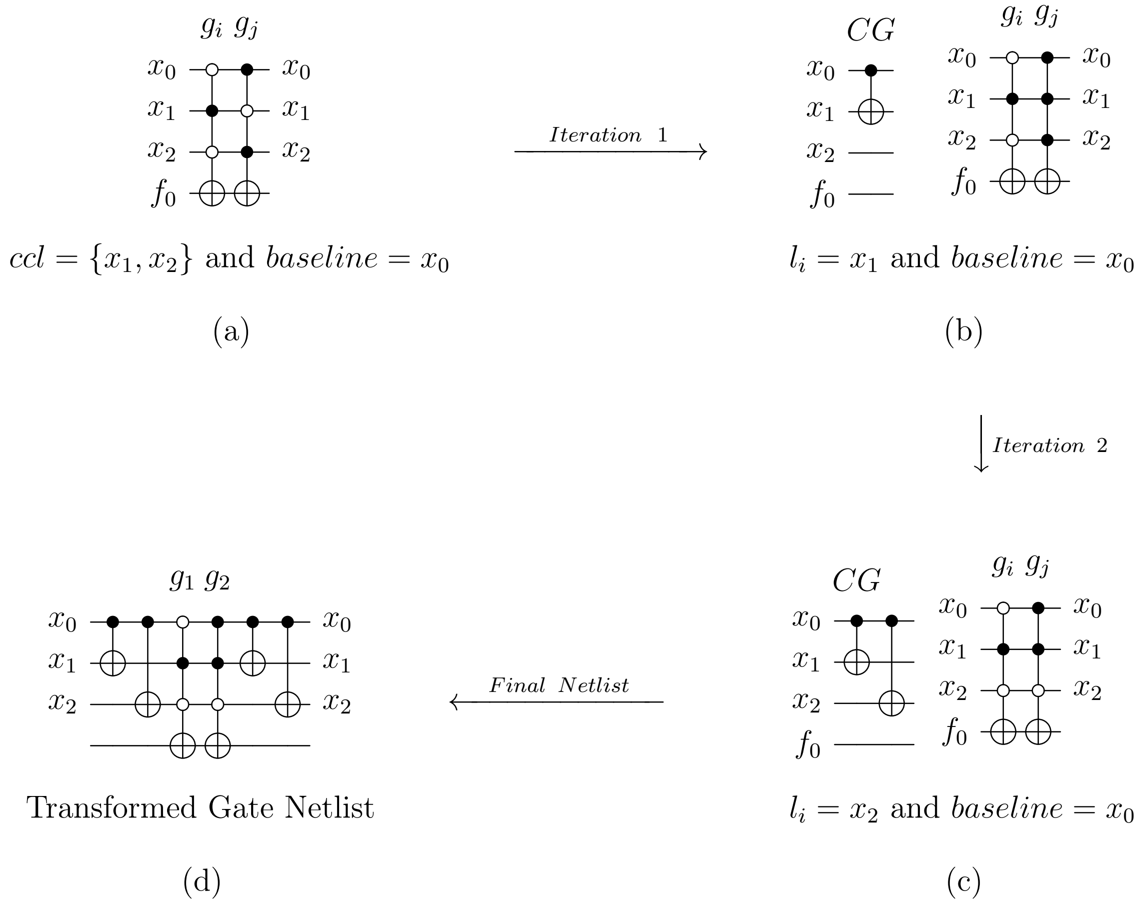


Figure 4.4: Illustration of Gate Transformation Algorithm (a) Initial Gate Pair (b) After Iteration 1 (c) After Iteration 2 (d) Final Transformed Gate Netlist

4.5 Application of CCL Transformation Algorithm

In this sub-section, the proposed transformation algorithm is applied such that to enable the decomposition rule discussed earlier and two existing rules to optimize the transformed gate netlist.

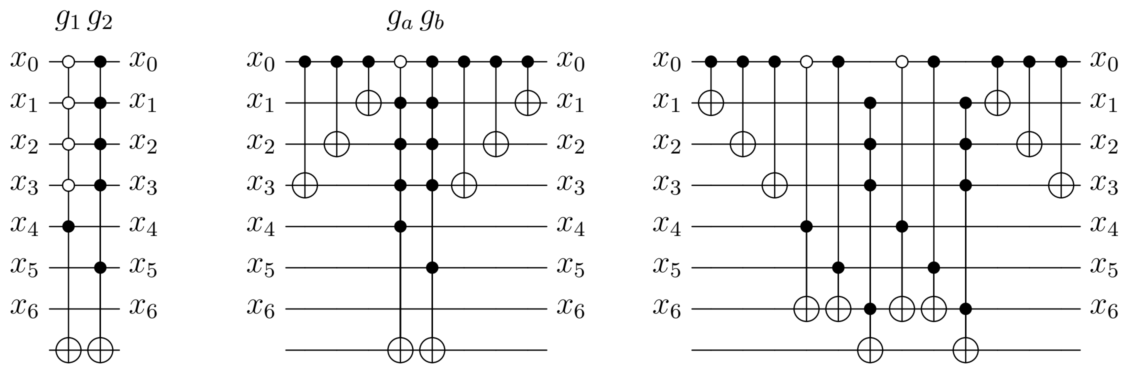
4.5.1 CCL Transformation Algorithm and Decomposition Rule

The proposed CCL transformation technique changes the number of equal and complementary control lines thereby changing the $|E|$, $|U_a|$ and $|U_b|$ of Eq. (4.1). If these values satisfy the conditions in Lemma (4.1) then the decomposition rule is applied on the transformed gate pair. Example 4.4 illustrates the usage of the decomposition rule after applying the transformation method for a pair of gates.

Example 4.4. Consider two gates g_1 and g_2 shown in figure. 5.10(a). The decomposition rule cannot be applied on this gate pair because there are no equal control lines i.e. $|E| = 0$. Applying the CCL transformation algorithm on this gate pair to transform CCLs to equal control lines results in the gate netlist as shown in figure 5.10(b). Since the gates g_a and g_b have three equal control lines, the decomposition rule can be applied. The final decomposed gate netlist is shown in figure 5.10(c) where it can be seen that the cost, represented as QC, has reduced from 104 to 78.

4.5.2 CCL Transformation Algorithm and Rule Based Optimization

Existing optimization rules for a pair of gates like merging rule, replacement rule e.t.c., presented in [38] can be applied on the gates which has utmost two CCLs. If the gate



$QC = 104$ $QC = 110$ $QC = 78$

(a) Original Gate Netlist (b) After applying Gate Transformation Algorithm (c) Final Decomposed Gate Netlist

Figure 4.5: Illustration of Example 4.4

pair is having more than two CCLs, these methods cannot be applied to improve the cost. However, the proposed transformation algorithm enables the usage of above rules on a gate pair that has one or more than one CCL. This is because the algorithm can reduce k CCLs in a gate pair to one CCL as described in Section 4.4.

The existing rules presented in [38] are briefly explained as follows:

1. *Merging Rule:* A pair of gates can be merged into a single gate if they have the same target lines, one CCL and any remaining control lines that are equal.

   This rule is represented as Equation 4.12 given below:

$$MCT(C \cup \{x_i\}; x_t) \circ MCT(C \cup \{\overline{x_i}\}; x_t) = MCT(C; x_t) \tag{4.12}$$

   where C is a set of equal control lines.

2. *Replacement Rule:* A pair of gates can be replaced with two gates if they have the same target lines, one CCL, a control line with control connection only on one gate and any remaining control lines that are equal.

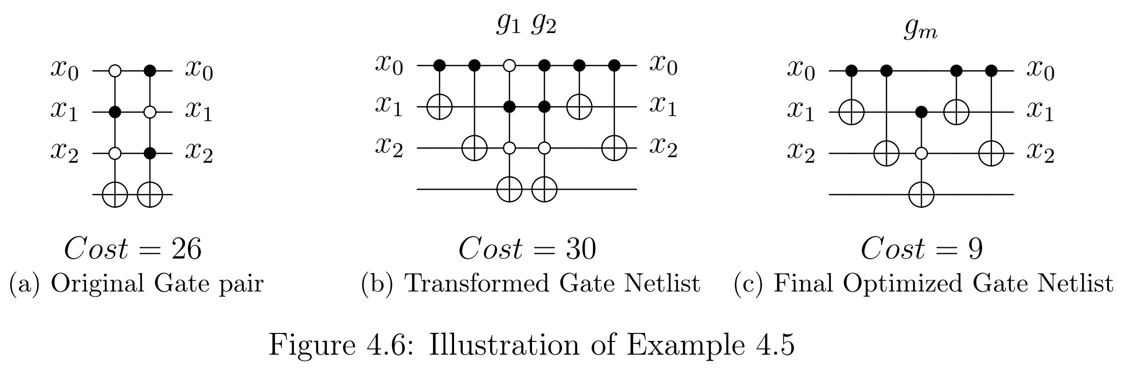
This rule is represented as Equation 4.13 given below:

$$MCT(C \cup \{x_i, x_j\}; x_t) \circ MCT(C \cup \{\overline{x_i}\}; x_t) = MCT(C \cup \{x_i, \overline{x_j}\}; x_t) \circ MCT(C; x_t) \quad (4.13)$$

where C is a set of equal control lines.

The process of transforming the gate pair and applying merging and replacement rules is illustrated with examples 4.5 and 4.6.

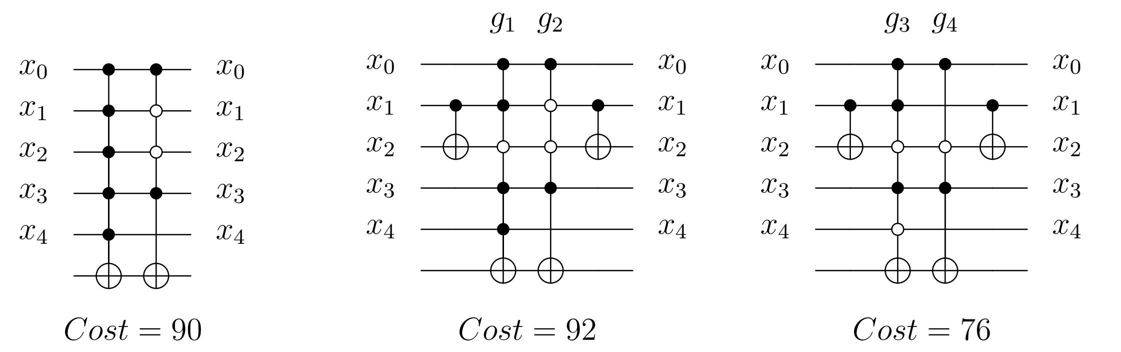
Example 4.5. Consider a pair of gates shown in figure 5.12(a). These gates cannot be optimized using merging rule as they have more than one CCL. After the application of the proposed algorithm however, the resulting gate netlist has only one CCL as shown in figure 5.12(c). The gates g_1 and g_2 in the resulting netlist have the same target line, one CCL and two equal control lines. Thus, g_1 and g_2 can be merged into a single gate g_m using the merging rule. The final gate netlist is shown in figure 5.12(d) where it can be seen that the cost of the gate netlist has reduced from 26 to 9.



(a) Original Gate pair (b) Transformed Gate Netlist (c) Final Optimized Gate Netlist

Figure 4.6: Illustration of Example 4.5

Example 4.6. Consider a pair of gates shown in figure 4.7(a). These gates cannot be optimized using the replacement rule because they have more than one CCL. After the algorithm is applied however, the resulting gate netlist has only one CCL as shown in figure 4.7(b). The gates g_1 and g_2 in the resulting netlist have the same target line, one CCL, the line x_4 with control connection only on gate g_1 and three equal control lines. Thus, g_1 and g_2 can be replaced with two other gates g_3 and g_4. The final gate netlist is shown in figure 4.7(c) where it can be seen that the cost of the gate netlist has reduced from 90 to 72.



(a) Original Gate pair (b) Transformed Gate Netlist (c) Final Optimized Gate Netlist

Figure 4.7: Illustration of Example 4.6

In the next sub-section a greedy optimization algorithm, that utilizes the reduction rules presented above to reduce the cost of the given gate netlist, is presented.

4.5.3 Greedy Optimization

In the greedy optimization algorithm given in 4.2, a reversible gate netlist G is given as input and a gate netlist G' is returned. Initially, the gate netlist G is traversed and the gates with equal control lines but different target lines are merged using the function *target_merging* [60]. This avoids regeneration of the same gate for different target lines. Since the reduction rules presented in Section 4.5.2 can be applied only on the gates with equal target lines, a set of segments consisting of gates with equal target lines is generated and assigned to *Segment*.

Algorithm 4.2 Greedy Optimization Method for a Gate Netlist

```
1: Input: Reversible Gate Netlist $G$
2: Output: Modified Gate Netlist $G'$
3: begin
4: $GN = target\_merging(G)$
5: $Segment = \phi$
6: $G' = \phi$
7: for each $g \in GN$ do
8:     $insert(Segment[target(g)], g)$
9: end for
10: for each $sg \in Segment$ do
11:     while until $sg$ is empty do
12:         $flag = false$
13:         $G_i = select\_gate(sg)$
14:         $RemoveGate(sg, G_i)$
15:         $CostTable(G_i) = \phi$
16:         for each $G_j \in sg$ do
17:             $Status = False$
18:             $G_t = \phi$
19:             $NewCost = \infty$
20:             $Status$, $G_t$, $NewCost = translate(G_i, G_j)$
21:             if $Status$ is $True$ then
22:                 $CostTable(G_i) = \{G_j, G_t, NewCost\}$
23:                 $flag = True$
24:             end if
25:         end for
26:         if $flag$ is $True$ then
27:             $G_p, G_{new} = LeastCost(CostTable(G_i))$
28:             $Append(G', G_{new})$
29:             $RemoveGate(sg, G_p)$
30:         else
31:             $Append(G', G_i)$
32:         end if
33:     end while
34: end for
35: return $G'$
36: end
```

For a *sg* in *Segment*, each gate G_i is paired with every other gate G_j and is given to the *translate* function. This function takes a gate pair and checks for the possibility of reduction using the transformation algorithm and the rules presented in Section 4.5.2. If a possibility exists for a reduction of that pair, the function returns *Status* as *True*, the reduced gate netlist as G_t and its cost as *NewCost*. Also, the gate netlist G_t and its cost $NewCost$ are added to the $CostTable(G_i)$ and the $flag$ is set to $True$.

After G_i is paired with every other gate G_j in sg, the status of $flag$ is checked. If the $flag$ is $False$, it indicates that there is no reduction possible for the gate G_i when paired with any other gate in that segment and the gate G_i is added to the output gate netlist G'. If it is $True$ then the function $LeastCost$ scans the $CostTable(G_i)$ and returns a gate G_p that results in maximum possible reduction when paired with G_i. This function also returns G_{new} which is the reduced gate netlist for the gate pair G_i and G_p. Finally, the gate netlist G_{new} is added to the output gate netlist G' and the gate G_p is subsequently removed from the segment sg. This process is repeated for each sg in the $Segment$.

4.6 Simulation Results

The greedy optimization method has been applied on different benchmark reversible circuits to evaluate its efficiency in terms of cost. The reversible gate netlist obtained from Exclusive-OR Sum of Product (ESOP) based synthesis method presented in [31] is given as the input for the optimization method.

The quantum cost for different benchmark circuits after applying the optimization method is shown in Table 4.1. The first column gives the benchmark name while the second and third columns provide the cost of input gate netlist and the gate netlist obtained after target merging, respectively. The fourth column gives the cost of the final gate netlist obtained after applying the optimization algorithm. The fifth and sixth columns give the percentage improvement over the initial gate netlist and the gate netlist obtained after target merging, respectively. It is seen from the Table 4.1 that in the best case, there is a considerable reduction of quantum cost of up to 84%. An average cost improvement of about 45% is observed over all the benchmarks considered in the table.

Table 4.1: Comparison with Input Gate Netlist

Benchmark	**Cost of Input Gate Netlist**	**Cost After Merge**	**Cost of Proposed Method**	**% Impr w.r.t Proposed Method**	
				Original	**After Merge**
9sym	10937	10937	3055	**72.07**	**72.07**
add6	6679	5157	3848	**42.39**	**25.38**
alu1	205	205	205	0.00	0.00
alu2	4623	4306	3090	**33.16**	**28.24**
alu3	2432	1976	1828	**24.84**	**7.49**
alu4	43635	36913	25007	**42.69**	**32.25**
apex4	252939	39818	39806	**84.26**	**0.03**
apex5	49161	31891	27960	**43.13**	**12.33**
apla	3806	1713	1601	**57.93**	**6.54**
bw	4464	820	820	**81.63**	0.00
C17	77	77	77	0.00	0.00
clip	7445	3842	1837	**75.33**	**52.19**
cm150a	803	803	785	**2.24**	**2.24**
con1	150	150	150	0.00	0.00

Table 4.1: Comparison with Input Gate Netlist (Continued)

Benchmark	**Cost of Input Gate Netlist**	**Cost After Merge**	**Cost of Proposed Method**	**% Impr w.r.t Proposed Method**	
				Original	**After Merge**
cordic	343959	172199	64504	**81.25**	**62.54**
cu	1191	747	747	**37.28**	0.00
dc2	1957	1097	1017	**48.03**	**7.29**
decod	2001	460	460	**77.01**	**0.00**
dist	7489	3723	2739	**63.43**	**26.43**
e64	26129	23751	23751	**9.10**	0.00
ex1010	178709	54154	52822	**70.44**	**2.46**
ex2	146	146	140	**4.11**	**4.11**
ex3	76	76	59	**22.37**	**22.37**
f2	262	118	83	**68.32**	**29.66**
f51m	30300	26533	19374	**36.06**	**26.98**
frg2	186500	103876	97001	**47.99**	**6.62**
in0	20639	7623	7501	**63.66**	**1.60**
majority	133	133	110	**17.29**	**17.29**
max46	4524	4524	2876	**36.43**	**36.43**
misex1	935	358	358	**61.71**	0.00

Table 4.1: Comparison with Input Gate Netlist (Continued)

Benchmark	**Cost of Input Gate Netlist**	**Cost After Merge**	**Cost of Proposed Method**	**% Impr w.r.t Proposed Method**	
				Original	**After Merge**
misex3	106810	46381	40771	**61.83**	**12.10**
misex3c	104924	46977	41886	**60.08**	**10.84**
mlp4	3878	2511	2129	**45.10**	**15.21**
mux	800	800	768	**4.00**	**4.00**
pm1	494	188	188	**61.94**	0.00
radd	721	659	483	**33.01**	**26.71**
rd84	2520	2334	1751	**30.52**	**24.98**
root	3618	1829	1548	**57.21**	**15.36**
sao2	7702	3688	3324	**56.84**	**9.87**
spla	94371	30518	26862	**71.54**	**11.98**
sqn	2096	1348	791	**62.26**	**41.32**
sqr6	989	609	549	**44.49**	**9.85**
sqrt8	583	477	312	**46.48**	**34.59**
squar5	365	234	231	**36.71**	**1.28**
t481	229	229	205	**10.48**	**10.48**
table3	79348	17662	16863	**78.75**	**4.52**

Table 4.1: Comparison with Input Gate Netlist (Continued)

Benchmark	**Cost of Input Gate Netlist**	**Cost After Merge**	**Cost of Proposed Method**	**% Impr w.r.t Proposed Method**	
				Original	**After Merge**
urf3	146687	53963	50761	**65.40**	**5.93**
z4	517	494	352	**31.91**	**28.74**

Table 4.2: Comparison with Existing ESOP based Synthesis Methods

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Method**	**% Impr**
5xp1	1349	786	865	-	807	773	**1.65**
add6	6362	-	5084	2683	-	3848	-43.42
alu2	5215	-	4476	-	3679	3090	**16.01**
alu3	2653	-	-	-	1919	1828	**4.74**
alu4	48778	41127	43850	-	38635	25007	**35.27**
apex4	256857	35840	50680	51284	-	39806	-11.07
apex5	-	33830	-	-	33803	27960	**17.29**
apla	4051	1683	-	-	1709	1601	**4.87**
bw	-	637	-	2233	790	820	-28.73
C17	97	-	-	-	-	77	**20.61**

Table 4.2: Comparison with Existing ESOP based Synthesis Methods (Continued)

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Method**	**% Impr**
clip	6616	3824	4484	-	3218	1837	**42.91**
cm150a	844	-	-	-	-	785	**6.99**
con1	207	162	-	-	-	150	**7.41**
cordic	349522	187620	-	-	111955	64504	**42.38**
cu	1332	781	-	-	780	747	**4.23**
dc2	1956	1084	-	-	1099	1017	**6.18**
decod	1924	399	-	976	-	460	-15.29
dist	7414	3700	-	-	-	2739	**25.97**
e64	-	-	-	-	24345	23751	**2.44**
ex1010	183726	52788	-	77293	-	52822	-0.06
ex2	153	-	-	118	-	140	-18.64
ex3	97	-	-	73	-	59	**19.18**
f2	274	112	-	116	-	83	**25.89**
f51m	34244	28382	-	-	25119	19374	**22.87**
frg2	-	112008	-	-	114239	97001	**13.40**
in0	22196	7949	-	-	-	7501	**5.64**
majority	147	-	-	106	-	110	-3.77
max46	4432	-	-	3239	-	2876	**11.21**

Table 4.2: Comparison with Existing ESOP based Synthesis Methods (Continued)

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Method**	**% Impr**
misex1	1017	332	466	-	352	358	-7.83
misex3	122557	49076	67206	-	54132	40771	**16.92**
misex3c	118578	49720	85330	52600	-	41886	**15.76**
mlp4	3827	2496	-	-	-	2129	**14.70**
mux	826	-	-	784	-	768	**2.04**
pm1	582	-	-	290	-	188	**35.17**
radd	798	-	-	349	-	483	-38.40
rd84	2598	-	2062	-	1965	1751	**10.89**
root	3486	1811	-	-	1583	1548	**2.21**
sao2	7893	3767	5147	-	-	3324	**11.76**
spla	-	-	49419	-	45478	26862	**40.93**
sqn	2170	-	-	1183	-	791	**33.14**
sqr6	1090	583	-	-	-	549	**5.83**
sqrt8	584	-	461	-	-	312	**32.32**
squar5	476	-	251	-	-	231	**7.97**
t481	237	-	237	-	-	205	**13.50**
table3	86173	-	35807	-	32286	16863	**47.77**
urf3	-	53157	-	56766	-	50761	**4.51**

Table 4.2: Comparison with Existing ESOP based Synthesis Methods (Continued)

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Method**	**% Impr**
z4	674	489	-	260	-	352	-35.38

A comparison of costs obtained from the optimization method with different ESOP based methods [32,35,36,58,59] is presented in Table 4.2. The first column gives the name of the benchmark circuit while the columns 2-6 indicate the quantum cost of respective benchmark circuits realized with existing ESOP based methods [32,35,36,58,59]. Column 7 provides the quantum cost to realize that benchmark using the proposed optimization method. Column 8 shows the percentage improvement of quantum cost achieved compared to the existing methods that give the best reduction for that benchmark.

It can be seen from the table that there is an improvement of up to 48% in quantum cost. For arithmetic benchmark circuits like *frg2*, *in0*,*max46*,etc., and large benchmark circuits like *misex3*, *table3*, etc., there is a reduction in the quantum cost. However, for some benchmarks like *add6, bw, z4* etc., the optimization method results in higher quantum cost when compared to the existing ones [35, 36, 59]. This is because of limited availability of a required gate pair that can be transformed using the algorithm and the rules presented earlier.

4.7 Summary

In this chapter, an algorithm to transform complementary control lines of a gate pair to equal control lines has been presented. It is shown that this algorithm enables the usage of a set of rules by converting CCLs to equal control lines. A greedy optimization technique which uses the transformation algorithm and the rules to optimize the given gate netlist has also been presented and discussed.

Chapter 5

Classification of a Pair of Reversible Gates

5.1 Introduction

Several techniques exist in the literature to reduce the quantum cost of a given gate pair. These techniques are chosen based on the control connections of gate pair control lines. However, there exists no methodology by which a pair of gates can be classified into different classes based on the control connections of each gate in that pair. In this work, a methodology to classify a pair of gates in a reversible circuit is proposed. This classification is used to identify gate pairs on which appropriate optimization techniques can be applied.

5.2 Quadruple Representation to Classify a Gate Pair

Quadruple representation (QR) for a gate pair, g_i and g_j, is denoted as $QR(g_i, g_j) = (\alpha, \beta, \gamma, \delta)$ where,

- α: Number of equal control lines in the gate pair
- β: Number of complementary control lines in the gate pair
- γ: Number of control lines that have control connection only in the gate g_i
- δ: Number of control lines that have control connection only in the gate g_j

For a gate pair g_i and g_j, the sets which contain control connections corresponding to α, β, γ and δ are defined as K, C, P and Q respectively. QR can be understood with the help of following example:

Example 5.1. Consider two gates g_1 and g_2 shown in the figure 5.1. They have two equal control lines, $K = \{x_0, x_3\}$, one complementary control line, $C = \{x_1\}$, the line with control connection only on gate g_1, that is, $P = \{x_2\}$ and the line with control connection only on gate g_2, that is, $Q = \{x_4\}$. Hence, according to the QR defined above, $QR(g_1, g_2) = (2, 1, 1, 1)$.

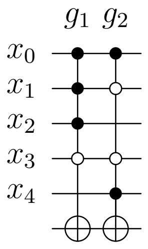


Figure 5.1: Example for quadruple representation (QR)

Lemma 5.1 stated below provides the possible number of classifications for a gate pair when at least one of the elements of the QR is not equal to zero. If all the elements in the QR of a gate pair are zero, then the gate pair is a cascade of two NOT gates that can be removed from the circuit.

Lemma 5.1. *In a reversible circuit, there exist* 15 *classifications for a gate pair g_i and g_j, where at least one of the elements in $QR(g_i, g_j)$ is a non-zero element.*

Proof. The $QR(g_i, g_j)$ is a quadruple with four elements α, β, γ and δ. Depending on the elements that are either zero or non-zero, there exist $2^4 = 16$ unique classifications. Of these, one classification has all its elements as zero. Thus, the number of classifications in which at least one of the elements is non-zero is $16 - 1 = 15$. □

An example for each classification along with their QR is shown in figure 5.2.

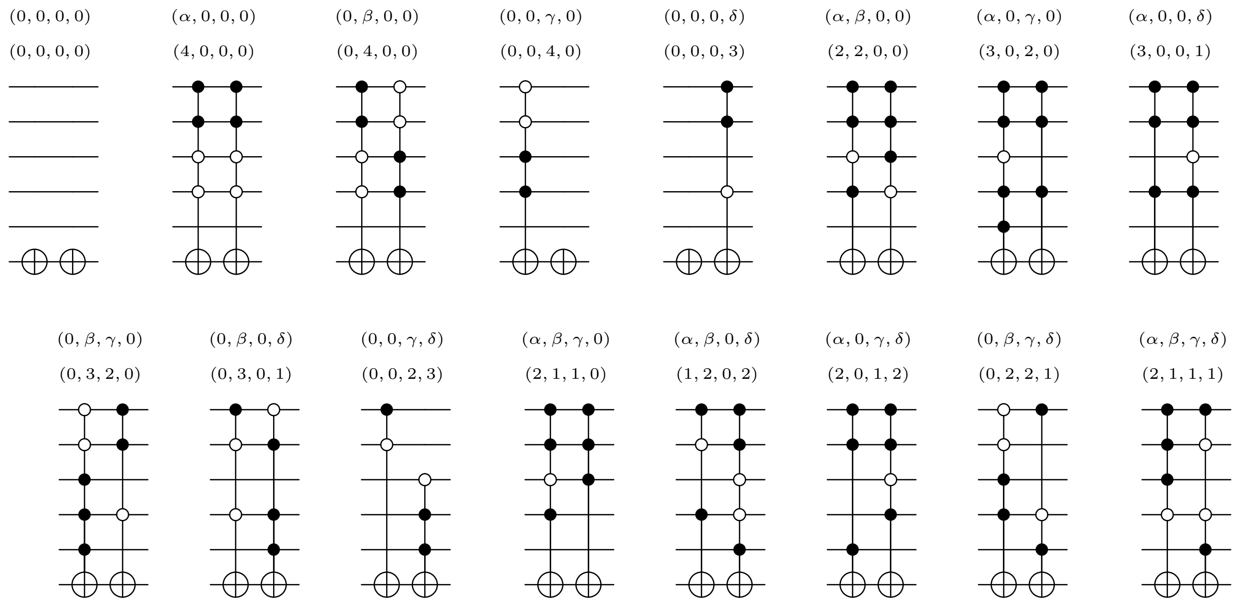

$(0,0,0,0)$ $(0,0,0,0)$
$(\alpha,0,0,0)$ $(4,0,0,0)$
$(0,\beta,0,0)$ $(0,4,0,0)$
$(0,0,\gamma,0)$ $(0,0,4,0)$
$(0,0,0,\delta)$ $(0,0,0,3)$
$(\alpha,\beta,0,0)$ $(2,2,0,0)$
$(\alpha,0,\gamma,0)$ $(3,0,2,0)$
$(\alpha,0,0,\delta)$ $(3,0,0,1)$
$(0,\beta,\gamma,0)$ $(0,3,2,0)$
$(0,\beta,0,\delta)$ $(0,3,0,1)$
$(0,0,\gamma,\delta)$ $(0,0,2,3)$
$(\alpha,\beta,\gamma,0)$ $(2,1,1,0)$
$(\alpha,\beta,0,\delta)$ $(1,2,0,2)$
$(\alpha,0,\gamma,\delta)$ $(2,0,1,2)$
$(0,\beta,\gamma,\delta)$ $(0,2,2,1)$
$(\alpha,\beta,\gamma,\delta)$ $(2,1,1,1)$

Figure 5.2: Examples for different QR classifications

There exist several rules in the literature to optimize (some) QR classifications as given below:

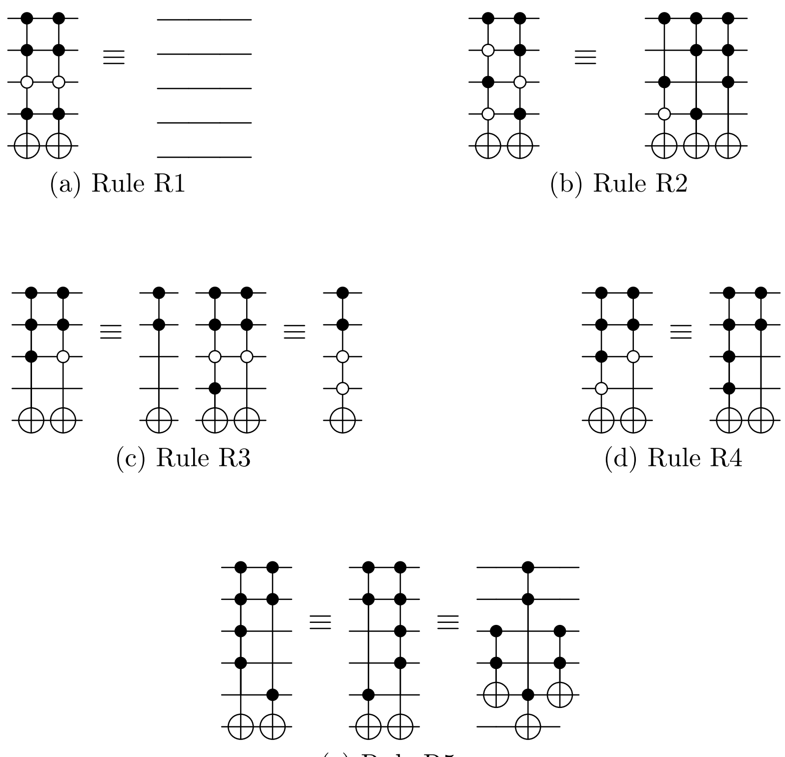
Rule R1 If $QR(g_i, g_j) = (\alpha, 0, 0, 0)$, then the gates are identical and can be removed from the circuit using the *deletion rule* presented in [24].

Rule R2 If $QR(g_i, g_j) = (\alpha, \beta, 0, 0)$ or $(0, \beta, 0, 0)$, then the gates can be decomposed into a network of smaller gates by applying the *rule based optimization* presented in [37].

Rule R3 If $QR(g_i, g_j) = (\alpha, 1, 0, 0)$ or $(\alpha, 0, 1, 0)$ or $(\alpha, 0, 0, 1)$, then the gates can be merged into single gate using the *merging rule* presented in [38].

Rule R4 If $QR(g_i, g_j) = (\alpha, 1, 1, 0)$ or $(\alpha, 1, 0, 1)$, then the gates can be replaced with a lower cost gate netlist using the *replacement rule* presented in [38].

Rule R5 If $QR(g_i, g_j) = (\alpha, 0, \gamma, 1)$ or $(\alpha, 0, 1, \delta)$, then the cost of the gate pair can be reduced using the *cube pairing* technique presented in [59].


(a) Rule R1

(b) Rule R2

(c) Rule R3

(d) Rule R4

(e) Rule R5

Figure 5.3: Examples for Existing Optimization Rules

An example for each of the rules discussed above is given in Fig 5.3. As mentioned earlier, these rules can be applied only on a few QR classifications. For example, classifications like $(\alpha, \beta, \gamma, \delta)$, $(\alpha, \beta, \gamma, 0)$, $(\alpha, \beta, 0, \delta)$, $(\alpha, 0, \gamma, \delta)$, $(\alpha, 0, \gamma, 0)$ and $(\alpha, 0, 0, \delta)$ do not have any optimization defined on them. In the following section, the mapping of the techniques presented in Chapters 3 and 4 to most of the classifications is discussed

5.3 Optimization Techniques for different QR classifications.

5.3.1 Optimization Technique for $QR(g_i, g_j) = (\alpha, \beta, \gamma, \delta)$ when $\alpha > 0$

The decomposition techniques presented in section 3.3 to decompose ESOP cubes can be directly applied to optimize MPMCT gates. This technique decomposes a pair of gates using the number of equal lines (α) between them and thus can be applied for QR classifications in which $\alpha > 0$, i.e. $(\alpha, 0, \gamma, 0)$, $(\alpha, 0, 0, \delta)$, $(\alpha, 0, \gamma, \delta)$, $(\alpha, \beta, \gamma, 0)$, $(\alpha, \beta, 0, \delta)$ and $(\alpha, \beta, \gamma, \delta)$.

Figures 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 illustrate the decomposition technique when applied to the gate pairs belonging to different classifications. The gate pairs are decomposed such that they generate redundant gates which are then removed from the circuit resulting in final gate netlists.

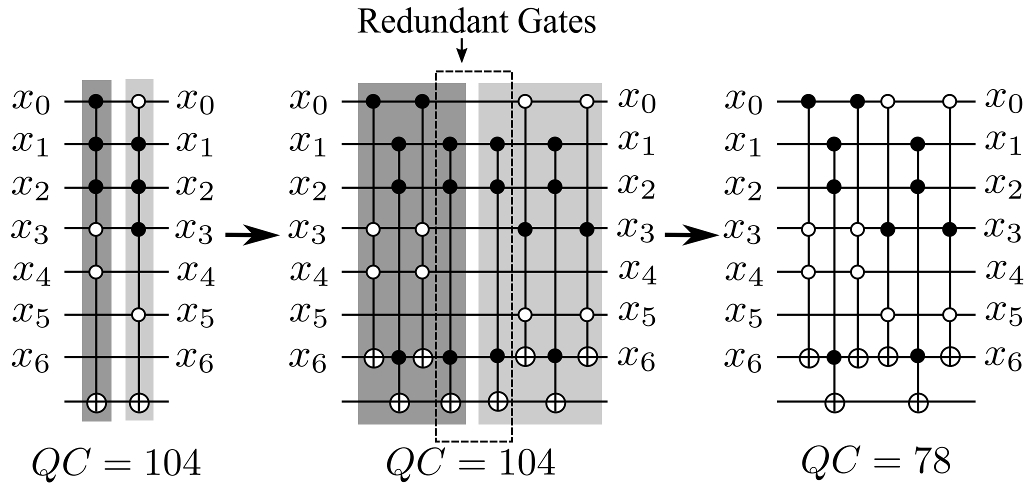


Figure 5.4: Example for $QR(g_i, g_j) = (\alpha, \beta, \gamma, \delta) = (2, 2, 1, 1)$

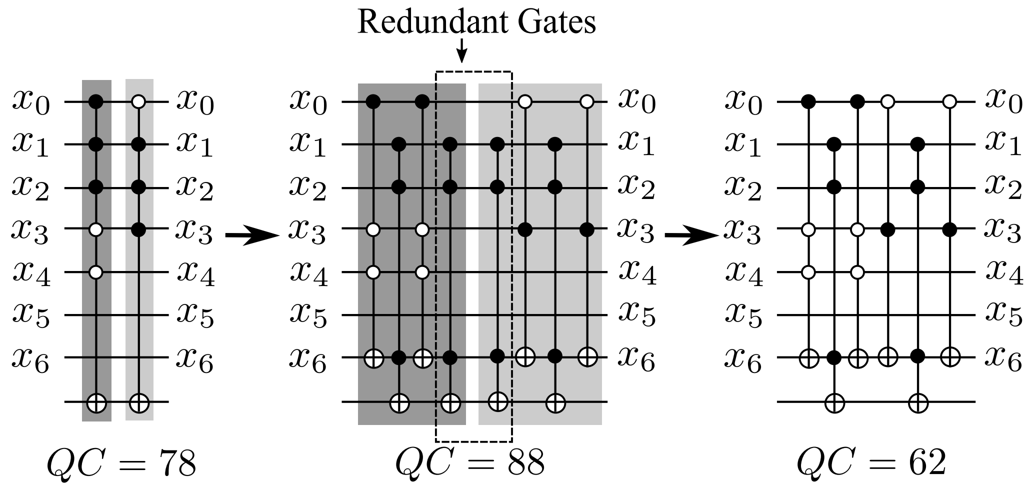


Figure 5.5: Example for $QR(g_i, g_j) = (\alpha, \beta, \gamma, 0) = (2, 2, 1, 0)$

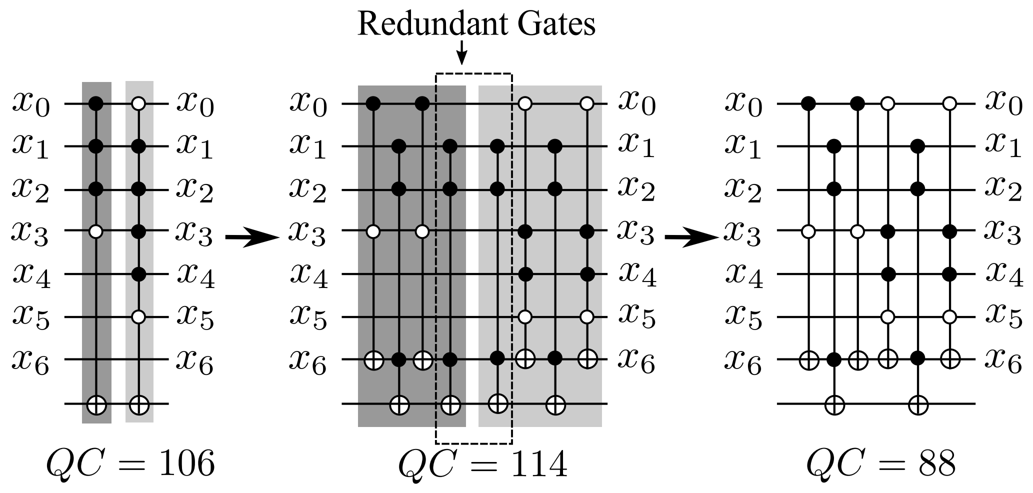


Figure 5.6: Example for $QR(g_i, g_j) = (\alpha, \beta, 0, \delta) = (2, 2, 0, 2)$

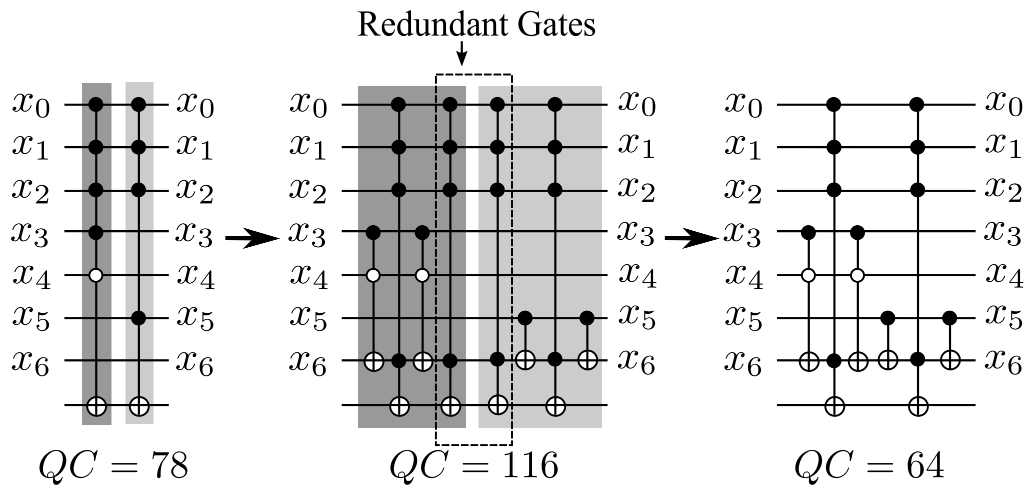


Figure 5.7: Example for $QR(g_i, g_j) = (\alpha, 0, \gamma, \delta) = (3, 0, 2, 1)$

It can be seen from Table 5.1 that there is a quantum cost reduction in each case

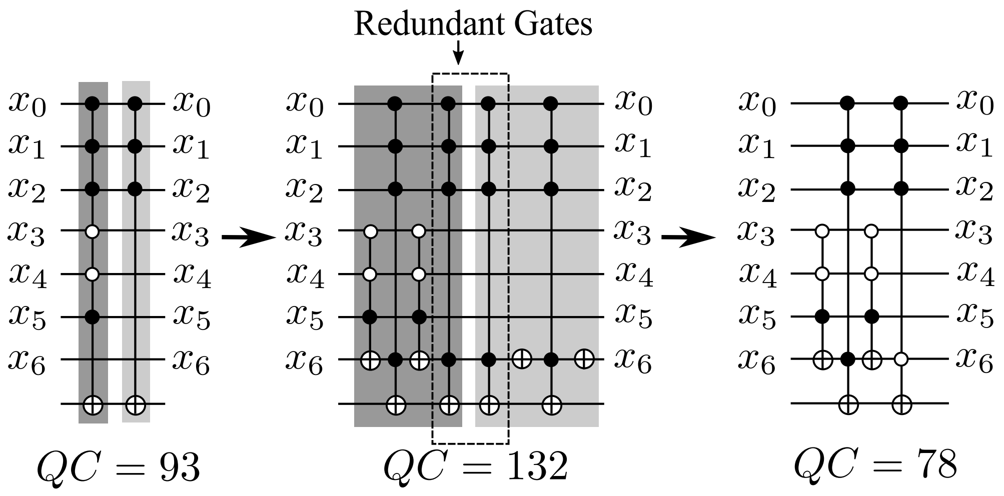


Figure 5.8: Example for $QR(g_i, g_j) = (\alpha, 0, \gamma, 0) = (3, 0, 3, 0)$

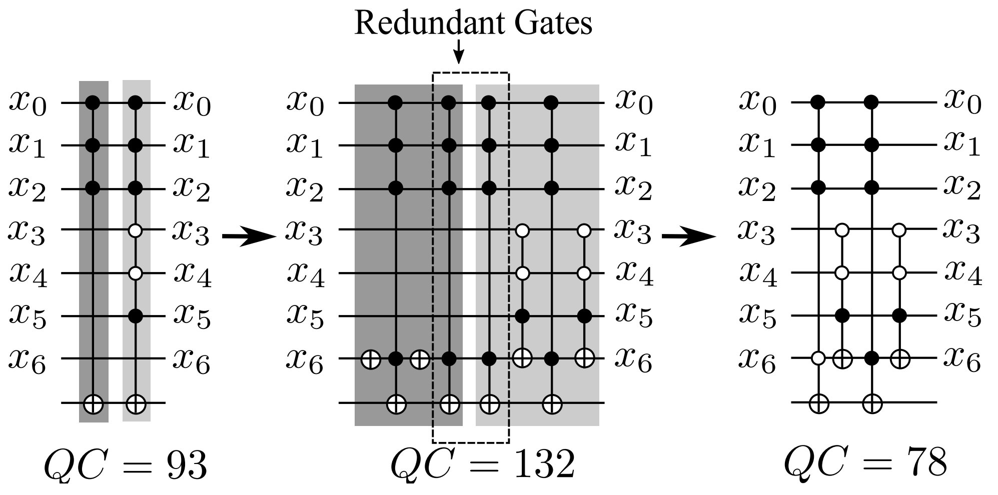


Figure 5.9: Example for $QR(g_i, g_j) = (\alpha, 0, 0, \delta) = (3, 0, 0, 3)$

after applying this decomposition rule.

As explained in Lemma 4.1, the decomposition technique results in reduced cost only if the number of unequal control lines in the gate pair, i.e. $U_a(= \beta + \gamma)$ or $U_b(= \beta + \delta)$, is less than half of the total number of lines in the circuit. However, if the values of $\beta + \gamma$ and $\beta + \delta$ are reduced such that $\beta + \gamma \leq \lceil \frac{n+1}{2} \rceil$ and $\beta + \delta \leq \lceil \frac{n+1}{2} \rceil$ then the decomposition technique can be applied to achieve the cost reduction. In order to reduce the values of $\beta + \gamma$ and $\beta + \delta$ and to increase the value of α, the CCL transformation approach presented in Section 4.4 can be applied.

The CCL transformation approach changes the number of equal and complementary

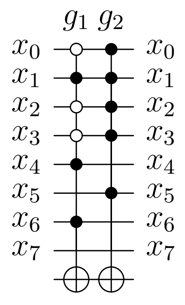
Table 5.1: Quantum Cost reduction for different classifications after applying decomposition Rule

$\boldsymbol{QR(g_i, g_j)}$	**Illustration**	**Quantum Cost (QC)**	
		Initial Gate Pair	**Effect of Applying Rule R6**
$(\alpha, \beta, \gamma, \delta) = (2, 2, 1, 1)$	Figure 5.4	104	78
$(\alpha, \beta, \gamma, 0) = (2, 2, 1, 0)$	Figure 5.5	78	62
$(\alpha, \beta, 0, \delta) = (2, 2, 0, 2)$	Figure 5.6	106	88
$(\alpha, 0, \gamma, \delta) = (3, 0, 2, 1)$	Figure 5.7	78	64
$(\alpha, 0, \gamma, 0) = (3, 0, 3, 0)$	Figure 5.8	93	78
$(\alpha, 0, 0, \delta) = (3, 0, 0, 3)$	Figure 5.9	93	78

control lines that modifies the QR of a gate pair. For example, consider a gate pair g_i and g_j with $QR(g_i, g_j) = (\alpha, \beta, \gamma, \delta)$. The transformation approach converts β CCLs to $\beta - 1$ ECLs resulting in $\alpha + \beta - 1$ equal and one complementary control line. Thus, the $QR(g_i, g_j)$ is transformed from $(\alpha, \beta, \gamma, \delta)$ to $(\alpha + \beta - 1, 1, \gamma, \delta)$. If the transformed QR satisfies the conditions presented in Lemma 4.1, then decomposition technique can be applied to reduce the cost of the circuit. This process of applying the transformation rule to modify the QR such that the decomposition technique can be applied is explained with the help of an example given below:

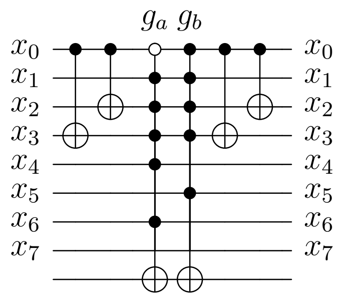
Example 5.2. Consider two gates g_1 and g_2 as shown in figure 5.10(a). The $QR(g_1, g_2) = (1, 3, 2, 1)$. According to Lemma 4.1, the decomposition rule does not result in cost reduction when applied on this gate pair. The CCL transformation technique is applied on this gate pair resulting in modified gate pair g_a and g_b with $QR(g_a, g_b) = (3, 1, 2, 1)$ as shown in figure 5.10(b). From this modified QR, it is evident that $\beta + \gamma \leq \lceil \frac{n+1}{2} \rceil$, $\beta + \delta \leq \lceil \frac{n+1}{2} \rceil$, $\alpha \leq \lceil \frac{n+1}{2} \rceil$ and the final decomposed gate netlist after applying the decomposition rule is shown in figure 5.10(c). It is evident from the figure that there is a reduction in the cost from 132 to 92.

So far, the QR classifications with $\alpha > 0$ have been covered with the proposed



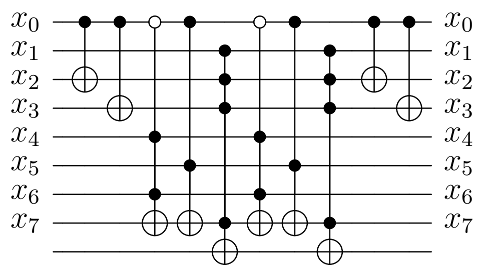
$QC = 132$

(a) Original Gate Netlist



$QC = 136$

(b) After applying CCL Transformation technique



$QC = 92$

(c) Final Gate Netlist

Figure 5.10: Illustration of Example 5.2

decomposition and transformation techniques. In the next section, the optimization technique for QR classifications with $\alpha = 0$ is discussed.

5.3.2 Optimization Technique for $QR(g_i, g_j) = (0, \beta, \gamma, \delta)$

The technique presented in the previous section are applicable when $\alpha > 0$ and can not be applied directly on the QR classifications of the type $(0, \beta, \gamma, \delta)$, $(0, \beta, 0, \delta)$ and $(0, \beta, \gamma, 0)$. However all these classifications have β CCLs, which can be converted to ECLs using the proposed CCL transformation approach. This approach transforms the QR types as given in the table below:

From the Table 5.2, it can be seen that the QR classifications after CCL transformation have $\alpha > 0$ and hence are a sub-set of QR classifications discussed in the Section

Table 5.2: QR Transformation

Before applying CCL Transformation	**After applying CCL Transformation**
$(0, \beta, \gamma, \delta)$	$(\beta - 1, 1, \gamma, \delta)$
$(0, \beta, \gamma, 0)$	$(\beta - 1, 1, \gamma, 0)$
$(0, \beta, 0, \delta)$	$(\beta - 1, 1, 0, \delta)$

5.3.1. The following example illustrates the process of applying the transformation approach to modify the QR such that the decomposition technqiue can be applied.

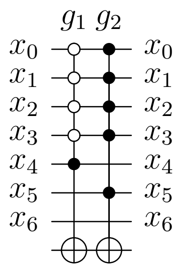
Example 5.3. Consider two gates g_1 and g_2 shown in figure 5.11(a). The $QR(g_1, g_2) = (0, 4, 1, 1)$. The CCL transformation approach is applied on this gate pair resulting in a modified gate pair g_a and g_b with $QR(g_a, g_b) = (3, 1, 1, 1)$ as shown in figure 5.11(b). The decomposition rule can now be applied to obtain the final reduced gate netlist as shown in figure 5.11(c). It is evident that there is a reduction in the cost from 104 to 78.

5.3.3 Optimization Techniques for Specific QR Classifications

In this section, different optimization techniques are presented for specific QR classifications.

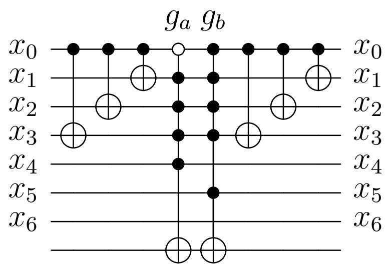
5.3.3.1 *For* $QR(g_i, g_j) = (\alpha, \beta, 0, 0)$ *and* $(0, \beta, 0, 0)$

The gate pairs that come under QR classifications of type $(\alpha, \beta, 0, 0)$ and $(0, \beta, 0, 0)$ can be reduced by applying rule **R2**. However, the CCL transformation technique further reduces the cost of the gate pair. This is achieved by converting $\beta - 1$ CCLs to ECLs which modifies the QR from $(\alpha, \beta, 0, 0)$ and $(0, \beta, 0, 0)$ to $(\alpha + \beta - 1, 1, 0, 0)$ and $(\beta - 1, 1, 0, 0)$ respectively. From the modified QRs it is evident that the rule **R3** can be applied to reduce the resultant gate netlist. This process is explained with the



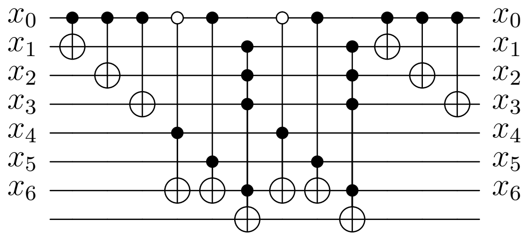
$QC = 104$

(a) Original Gate Netlist



$QC = 110$

(b) After applying CCL Transformation Technique



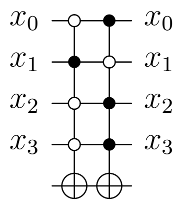
$QC = 78$

(c) Final Gate Netlist

Figure 5.11: Illustration of Example 5.3

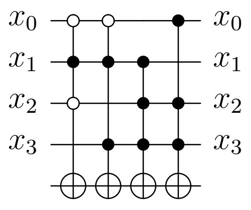
following example:

Example 5.4. Consider a pair of gates with $QR(g_1, g_2) = (0, 4, 0, 0)$ as shown in figure 5.12(a). The reduction of this pair using rule **R2** results in the gate netlist as shown in figure 5.12b. Applying CCL transformation on the gate pair modifies the QR to $(3, 1, 0, 0)$ as shown in figure 5.12(c). The modified gate pair can be merged to single gate using the rule **R3** as given in figure 5.12(d). From the figure it can be seen that the cost of the circuit, as a result of applying rule **R2** and CCL transformation with rule **R3,** reduces to 52 and 19 respectively.



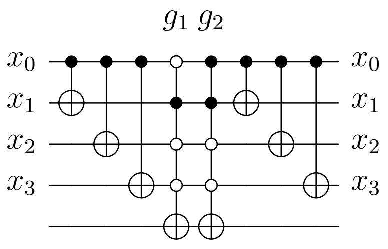
$QC = 58$

(a) Original Gate pair



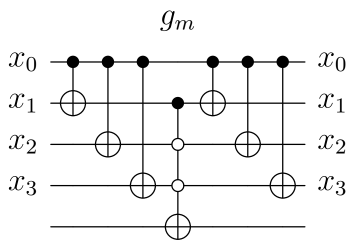
$QC = 52$

(b) Reduced netlist using rule R2



$QC = 64$

(c) Transformed Gate Netlist



$QC = 19$

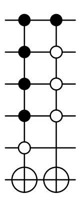
(d) Final Optimized Gate Netlist

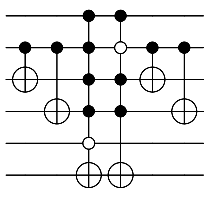
Figure 5.12: Illustration of Example 5.4

5.3.3.2 *For* $QR(g_i, g_j) = (\alpha, \beta, 1, 0)$ *and* $(\alpha, \beta, 0, 1)$

The rule **R4** can be applied on the classifications of type $(\alpha, \beta, 1, 0)$ and $(\alpha, \beta, 0, 1)$ only if $\beta = 1$. However, if $\beta > 1$ then the CCL transformation can be used to modify the QR classifications from $(\alpha, \beta, 1, 0)$ and $(\alpha, \beta, 0, 1)$ to $(\alpha+\beta-1, 1, 1, 0)$ and $(\alpha+\beta-1, 1, 0, 1)$ respectively. This enables the use of rule **R4** on the modified QRs which is explained in the following example:

Example 5.5. Consider a pair of gates with $QR(g_1, g_2) = (1, 3, 1, 0)$ as shown in figure 5.13(a). Applying CCL transformation on the gate pair modifies QR to $(3, 1, 1, 0)$ as shown in figure 5.13(b). Rule **R4** can be applied on the modified gate pair and the resulting gate netlist is shown in figure 5.13(c) in which it can be seen that there is a reduction of the cost from 78 to 69.


$QC = 78$
(a) Initial Gate Netlist


$QC = 82$
(b) After CCL Transformation

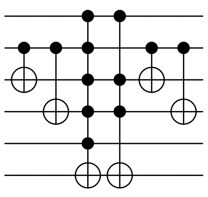

$QC = 69$
(c) Final Gate Netlist

Figure 5.13: Illustration of Example 5.5

5.3.3.3 *For* $QR(g_i, g_j) = (\alpha, \beta, 1, 1)$

An optimization technique is presented in this sub-section to reduce the cost of a gate pairs,g_i and g_j, with $QR(g_i, g_j) = (\alpha, 1, 1, 1)$. This technique decomposes the gate pair into a network of smaller gates and is explained with the help of following lemma:

Lemma 5.2 (Swap Rule). *Consider two gates g_i and g_j with same target line x_t. The $QR(g_i, g_j) = (\alpha, \beta, \gamma, \delta)$ and the sets of control lines corresponding to α, β, γ and δ are defined as K, C, P and Q respectively. If β, γ and δ are equal to 1, $C = \{x_C\}, P = \{x_P\}$ and $Q = x_Q$, then the gate pair g_i and g_j can be decomposed using the following rule:*

$$g_i \circ g_j = MCT(\{x_Q\}; x_P\}) \circ MCT(\{x_C\} \cup \{x_P\}; x_Q) \circ MCT(K \cup \{x_C\} \cup \{x_P\}; x_t) \circ$$
$$MCT(\{x_C\} \cup \{x_P\}; x_Q) \circ MCT(\{x_Q\}; x_P\})$$

Proof. The gates g_i and g_j are represented as $g_i = MCT(K \cup C \cup P; x_t)$ and $g_j = MCT(K \cup C \cup Q; x_t)$ respectively. If β, γ and δ are equal to 1, then there exists only one control line in the sets C, P and Q respectively, i.e. $C = \{x_C\}, P = \{x_P\}$ and $Q = \{x_Q\}$. Assuming the control line x_C has a positive control connection in the gate g_i and a negative control connection in the gate g_j, they can be written as:

$$g_i = MCT(K \cup \{x_C\} \cup \{x_P\}; x_t)$$

$$g_j = MCT(K \cup \{\overline{x_C}\} \cup \{x_Q\}; x_t)$$

The gate g_j can be decomposed into two gates using the rules presented in [37] which is given as follows:

$$g_j = MCT(K \cup \{x_C\} \cup \{x_Q\}; x_t) \circ MCT(K \cup \{x_Q\}; x_t)$$

The cascade of gate pair g_i and g_j is given as:

$$g_i \circ g_j = MCT(K \cup \{x_C\} \cup \{x_P\}; x_t) \circ MCT(K \cup \{x_C\} \cup \{x_Q\}; x_t) \circ MCT(K \cup \{x_Q\}; x_t)$$

Using rule R5, the first two gates in the above netlist can be decomposed into network of three gates as:

$$g_i \circ g_j = MCT(\{x_Q\}; x_P\}) \circ MCT(K \cup \{x_C\} \cup \{x_P\}; x_t) \circ MCT(\{x_Q\}; x_P\}) \circ MCT(K \cup \{x_Q\}; x_t)$$

The position of last two gates can be interchanged using the moving rule presented in [24] and the resulting gate netlist is given as:

$$g_i \circ g_j = MCT(\{x_Q\}; x_P\}) \circ MCT(K \cup \{x_C\} \cup \{x_P\}; x_t) \circ MCT(K \cup \{x_Q\}; x_t) \circ MCT(\{x_Q\}; x_P\})$$

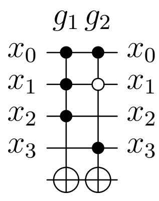
The second and third gates in the above netlist can be decomposed into a network of three gates using rule R5 and the resulting netlist is given as follows, thus proving the lemma:

$$g_i \circ g_j = MCT(\{x_Q\}; x_P\}) \circ MCT(\{x_C\} \cup \{x_P\}; x_Q) \circ MCT(K \cup \{x_C\} \cup \{x_P\}; x_t) \circ$$
$$MCT(\{x_C\} \cup \{x_P\}; x_Q) \circ MCT(\{x_Q\}; x_P\})$$

$\square$

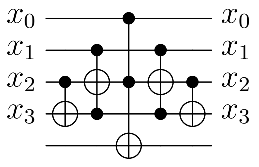
The following example illustrates the above lemma:

Example 5.6. Consider a pair of gates with $QR(g_1, g_2) = (1, 1, 1, 1)$ as shown in figure 5.14(a). The set of control lines corresponding to this QR are $K = \{x_0\}$, $C = \{x_1\}$, $P = \{x_2\}$ and $Q = \{x_3\}$. According to Lemma 5.2, the gate pair is decomposed into a gate netlist as shown in figure 5.14(b) where it can be seen that there is a reduction in the QC from 26 to 17.



$QC = 26$

(a) Initial Gate Netlist



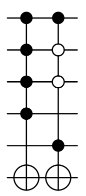
$QC = 17$

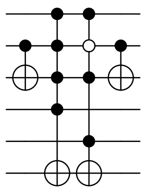
(b) Final Gate Netlist

Figure 5.14: Illustration of Example 5.6

Even-though *Swap Rule* adds two Toffoli and two CNOT gates, the gate pair is merged to a single gate with reduced control lines, thereby reducing the cost of the gate pair. This technique can be applied only if $\beta = 1$ but not if $\beta > 1$. Thus, if the gate pair has $QR(g_i, g_j) = (\alpha, \beta, 1, 1)$ then the CCL transformation technique can be applied to modify the QR classifications to $QR(g_i, g_j) = (\alpha + \beta - 1, 1, 1, 1)$ thus enabling the use of *Swap Rule* to reduce the cost of the gate pair. This process is illustrated in the following example:

Example 5.7. Consider a pair of gates with $QR(g_1, g_2) = (1, 2, 1, 1)$ as shown in figure 5.15(a). Since Lemma 5.2 can be applied only on gate pair with $\beta = 1$, CCL transformation is applied on this gate pair and the modified gate netlist is shown in figure 5.15(b) such that the QR is modified to $(2, 1, 1, 1)$. From the modified QR, *Swap Rule* can be applied on this gate netlist and the resulting circuit is shown in figure 5.15(c). From the figure, it is evident that the cost of the gate pair is reduced from 52 to 27.


$QC = 52$
(a) Initial Gate Netlist


$QC = 54$
(b) After applying CCL transformation approach

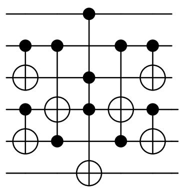

$QC = 27$
(c) Final Gate Netlist

Figure 5.15: Illustration of Example 5.7

There can be no optimization technique that covers QR classifications of type $(0, 0, \gamma, 0)$, $(0, 0, 0, \delta)$ and $(0, 0, \gamma, \delta)$ because the gate pairs do not have any dependent control lines like equal or complementary control lines. Table 5.3 summarizes the QR classifications and the optimization techniques that can be followed for respective classifications.

Table 5.3: Summary of Optimization Techniques for different QR classifications

$\boldsymbol{QR(g_i, g_j)}$	**Conditions**	**Optimization Techniques**
$(0, 0, 0, 0)$	-	Deletion Rule R1
$(\alpha, 0, 0, 0)$	-	Deletion Rule R1
$(\alpha, \beta, 0, 0)$, $(0, \beta, 0, 0)$	$\beta = 1$	Rule R3
	$\beta > 1$	CCL Transformation Approach + Rule R3
$(0, 0, \gamma, 0)$, $(0, 0, 0, \delta)$, $(0, 0, \gamma, \delta)$	-	No reduction technique
$(\alpha, \beta, \gamma, \delta)$, $(\alpha, \beta, \gamma, 0)$, $(\alpha, \beta, 0, \delta)$, $(\alpha, 0, \gamma, \delta)$, $(\alpha, 0, \gamma, 0)$, $(\alpha, 0, 0, \delta)$	($\beta = 0$, $\gamma = 0$ and $\delta = 1$) or ($\beta = 0$, $\gamma = 1$ and $\delta = 0$)	Rule R3
	($\beta = 1$, $\gamma = 1$ and $\delta = 0$) or ($\beta = 1$, $\gamma = 0$ and $\delta = 1$)	Rule R4
	($\beta > 1$, $\gamma = 0$ and $\delta = 1$) or ($\beta > 1$, $\gamma = 1$ and $\delta = 0$)	CCL Transformation Approach + Rule R4
	($\beta = 0$ and $\gamma = 1$) or ($\beta = 0$ and $\delta = 1$)	Cube Pairing Rule R5
	($\gamma = 1$ and $\delta = 1$)	Swap Rule
	$\beta + \gamma \leq \lceil \frac{n+1}{2} \rceil$, $\beta + \delta \leq \lceil \frac{n+1}{2} \rceil$ and $\alpha > 1$	Decomposition Technique
	$\beta + \gamma > \lceil \frac{n+1}{2} \rceil$, $\beta + \delta > \lceil \frac{n+1}{2} \rceil$, $\alpha \leq \lceil \frac{n+1}{2} \rceil$,$1 + \gamma \leq \lceil \frac{n+1}{2} \rceil$ and $1 + \delta \leq \lceil \frac{n+1}{2} \rceil$	CCL Transformation Approach + Decomposition Technique
	$\beta + \gamma > \lceil \frac{n+1}{2} \rceil$, $\beta + \delta > \lceil \frac{n+1}{2} \rceil$, $\alpha \leq \lceil \frac{n+1}{2} \rceil$,$1 + \gamma > \lceil \frac{n+1}{2} \rceil$ and $1 + \delta > \lceil \frac{n+1}{2} \rceil$	No reduction technique
$(0, \beta, \gamma, \delta)$, $(0, \beta, \gamma, 0)$, $(0, \beta, 0, \delta)$	$\beta + \gamma \leq \lceil \frac{n+1}{2} \rceil$, $\beta + \delta \leq \lceil \frac{n+1}{2} \rceil$ and $\beta > 2$	CCL Transformation Approach + Decomposition Technique
	$\beta + \gamma > \lceil \frac{n+1}{2} \rceil$, $\beta + \delta > \lceil \frac{n+1}{2} \rceil$, $\beta > 2$,$1 + \gamma \leq \lceil \frac{n+1}{2} \rceil$ and $1 + \delta \leq \lceil \frac{n+1}{2} \rceil$	
	$\beta + \gamma > \lceil \frac{n+1}{2} \rceil$, $\beta + \delta > \lceil \frac{n+1}{2} \rceil$, $1 + \gamma > \lceil \frac{n+1}{2} \rceil$ and $1 + \delta > \lceil \frac{n+1}{2} \rceil$	No reduction technique

5.4 Post-Synthesis Optimization Algorithm using QR Classification

A greedy algorithm for post-synthesis optimization is presented in Algorithm 5.1. This algorithm takes a reversible gate netlist G as input and gets an optimized gate netlist G' as output. Initially, the gate netlist G is traversed and the gates with equal control lines but different target lines are merged using the function $target\_merging$ [60]. This avoids regeneration of the same gate for different target lines.

Each gate G_i is paired with every other gate G_j in GN and is given to the $translate$ function. This function takes a gate pair, classifies it using the proposed representation and checks for the possibility of reduction using the optimization techniques presented in the Table 5.3. If a possibility exists for optimizing that pair, the function returns $Status$ as $True$, the optimized gate netlist as G_t and its cost as $NewCost$. Also, the gate netlist G_t and its cost $NewCost$ are added to the $CostTable(G_i)$ and the $flag$ is set to $True$.

After G_i is paired with every other gate G_j in GN, the status of $flag$ is checked. If the $flag$ is $False$, it indicates that there is no optimization possible for the gate G_i when paired with any other gate in the circuit and the gate G_i is added to the output gate netlist G'. If it is $True$ then the function $LeastCost$ scans the $CostTable(G_i)$ and returns a gate G_p that results in maximum possible reduction when paired with G_i. This function also returns G_{new} which is the reduced gate netlist for the gate pair G_i and G_p. If the gate netlist G_{new} consists of single gate then the gate is added to initial netlist GN else it is added to the output gate netlist G' and the gate G_p is subsequently removed from the netlist GN. This process is repeated for each gate in the netlist.

Algorithm 5.1 Post-Synthesis Greedy Optimization Method

```
1: Input: Reversible Gate Netlist $G$
2: Output: Modified Gate Netlist $G'$
3: begin
4: $GN = target\_merging(G)$
5: $G' = \phi$
6: while $GN$ is not empty do
7:     $flag = false$
8:     $G_i = select\_gate(GN)$
9:     $RemoveGate(GN, G_i)$
10:    $CostTable(G_i) = \phi$
11:    for each $G_j \in GN$ do
12:        $Status = False$
13:        $G_t = \phi$
14:        $NewCost = \infty$
15:        $Status$, $G_t$, $NewCost = translate(G_i, G_j)$
16:        if $Status$ is $True$ then
17:            $CostTable(G_i) = \{G_j, G_t, NewCost\}$
18:            $flag = True$
19:        end if
20:    end for
21:    if $flag$ is $True$ then
22:        $G_p, G_{new} = LeastCost(CostTable(G_i))$
23:        if $SizeOf(G_{new}) == 1$ then
24:            $Append(GN, G_{new})$
25:        else
26:            $Append(G', G_{new})$
27:        end if
28:        $RemoveGate(GN, G_p)$
29:    else
30:        $Append(G', G_i)$
31:    end if
32: end while
33: return $G'$
34: end
```

5.5 Simulation Results and Comparison

The post-synthesis optimization algorithm presented is applied on different benchmark reversible circuits to evaluate its efficiency in terms of cost. These circuits are obtained from RevLib library [50] and are given as inputs to the optimization algorithm. Table 5.4 presents a comparison of the proposed optimization algorithm with existing post-synthesis optimization techniques presented in [39] and [38]. The first column gives the name of the benchmark while second, third and fourth columns indicate the quantum cost of original gate netlist from RevLib and two existing techniques respectively. The fifth column gives the quantum cost of final gate netlist obtained after applying proposed optimization algorithm. The sixth, seventh and eighth columns give the percentage improvement over original gate netlist and the existing techniques. It can be seen from the Table 5.4 that in the best case, there is a considerable reduction of quantum cost of up to 78.5%. An average cost improvement of about 47% is observed over all the benchmarks mentioned in the table. This is because, the proposed optimization techniques optimize more number of QR classifications of gate pairs (as seen from Table 5.3) when compared to the techniques presented in [39] and [38].

Table 5.4: Comparison with Existing Post-Synthesis Optimization Algorithms

Benchmark	**Original Netlist [50]**	**[39]**	**[38]**	**Proposed Optimization**	**% Impr with**		
					[50]	**[39]**	**[38]**
cordic_218	349522	348566	348532	74985	78.55	78.49	78.49
apex4_202	238146	237748	158095	38826	83.70	83.67	75.44
sym9_193	14193	12747	13090	3485	75.45	72.66	73.38

Table 5.4: Comparison with Existing Post-Synthesis Optimization Algorithms (Continued)

Benchmark	**Original Netlist [50]**	**[39]**	**[38]**	**Proposed Optimization**	**% Impr with**		
					[50]	**[39]**	**[38]**
table3_264	80039	79326	61412	17963	77.56	77.36	70.75
in2_236	23814	23146	20600	7434	68.78	67.88	63.91
misex3_242	119177	115637	99119	38125	68.01	67.03	61.54
clip_206	6731	6535	6119	2354	65.03	63.98	61.53
misex3c_243	115190	111258	96064	39907	65.36	64.13	58.46
life_238	6767	5740	5744	2420	64.24	57.84	57.87
dist_223	7604	7288	6631	2875	62.19	60.55	56.64
sqn_258	2128	2041	1887	849	60.10	58.40	55.01
in0_235	20042	18999	16985	7761	61.28	59.15	54.31
inc_237	2145	2104	1745	929	56.69	55.85	46.76
apla_203	3444	3438	3029	1669	51.54	51.45	44.90
f51m_233	37417	33333	32882	18356	50.94	44.93	44.18
tial_265	56224	47145	47556	26644	52.61	43.48	43.97
max46_240	5444	4498	4538	2560	52.98	43.09	43.59
dc2_222	1898	1789	1688	1024	46.05	42.76	39.34
sqr6_259	1053	1034	876	567	46.15	45.16	35.27
decod_217	1746	1745	613	427	75.54	75.53	30.34

Table 5.4: Comparison with Existing Post-Synthesis Optimization Algorithms (Continued)

Benchmark	**Original Netlist [50]**	**[39]**	**[38]**	**Proposed Optimization**	**% Impr with**		
					[50]	**[39]**	**[38]**
5xp1_194	1418	1327	1155	819	42.24	38.28	29.09
pm1_249	384	354	275	197	48.70	44.35	28.36
cu_219	1148	1054	954	702	38.85	33.40	26.42
mux_246	1078	804	804	598	44.53	25.62	25.62
dc1_220	425	419	249	187	56.00	55.37	24.90
cm150a_210	1096	822	822	618	43.61	24.82	24.82
frg1_234	15266	14737	14702	11560	24.28	21.56	21.37

The reversible gate netlist obtained from Exclusive-OR Sum of Product (ESOP) based synthesis method presented in [31] is given as the input for the optimization algorithm. A comparison of costs obtained from the optimization algorithm with different optimization methods that are applied in ESOP based synthesis methods [32,35,36,58,59] is presented in Table 5.5. The first column gives the name of the benchmark circuit while the columns 2-6 indicate the quantum cost of respective benchmark circuits realized with existing ESOP based methods [32, 35, 36, 58, 59]. Column 7 provides the quantum cost to realize that benchmark using the proposed optimization method. Column 8 shows the percentage improvement of quantum cost achieved compared to the existing methods that give the best reduction for that benchmark.

It can be seen from the table that there is an improvement of up to 48% in quantum cost. For arithmetic benchmark circuits like *frg2*, *in0*,*max46*,etc., and large benchmark circuits like *misex3*, *table3*, etc., there is a reduction in the quantum cost. However, for some benchmarks like *add6, bw, z4* etc., the optimization method results in higher quantum cost when compared to the existing ones [35, 36, 59]. This is because only a small number of required gate pairs is available that can be transformed using the algorithm and the techniques presented earlier.

Table 5.5: Comparision with Exisiting ESOP Based Methods

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Optimization**	**% Impr**
5xp1	1349	786	865	-	807	741	**5.73**
9symml	5781	10943	16487	1895	3406	2709	-42.96
add6	6362	-	5084	2683	-	3132	-16.73
alu2	5215	-	4476	-	3679	2685	**27.02**
alu3	2653	-	-	-	1919	1810	**5.68**
alu4	48778	41127	43850	-	38635	21104	**45.38**
apex4	256857	35840	50680	51284	-	39832	-11.14
apex5	-	33830	-	-	33803	26358	**22.02**
apla	4051	1683	-	-	1709	1607	**4.52**
bw	-	637	-	2233	790	807	-26.69
C17	97	-	-	-	-	74	**23.71**
clip	6616	3824	4484	-	3218	2184	**32.13**
cm150a	844	-	-	-	-	438	**48.1**
con1	207	162	-	-	-	136	**16.05**
cordic	349522	187620	-	-	111955	64624	**42.28**

Table 5.5: Comparision with Exisiting ESOP Based Methods (Continued)

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Optimization**	**% Impr**
cu	1332	781	-	-	780	603	**22.69**
dc2	1956	1084	-	-	1099	980	**9.59**
decod	1924	399	-	976	-	461	-15.54
dist	7414	3700	-	-	-	2745	**25.81**
e64	-	-	-	-	24345	23751	**2.44**
ex1010	183726	52788	-	77293	-	49490	**6.25**
ex2	153	-	-	118	-	117	**0.85**
ex3	97	-	-	73	-	53	**27.4**
f2	274	112	-	116	-	87	**22.32**
f51m	34244	28382	-	-	25119	16850	**32.92**
frg2	-	112008	-	-	114239	88554	**20.94**
in0	22196	7949	-	-	-	7474	**5.98**
majority	147	-	-	106	-	134	-26.42
max46	4432	-	-	3239	-	2254	**30.41**
misex1	1017	332	466	-	352	360	-8.43
misex3	122557	49076	67206	-	54132	36624	**25.37**
misex3c	118578	49720	85330	52600	-	38037	**23.5**
mlp4	3827	2496	-	-	-	2079	**16.71**
mux	826	-	-	784	-	416	**46.94**
pm1	582	-	-	290	-	188	**35.17**

Table 5.5: Comparision with Exisiting ESOP Based Methods (Continued)

Benchmark	**[32]**	**[36]**	**[58]**	**[59]**	**[35]**	**Proposed Optimization**	**% Impr**
radd	798	-	-	349	-	470	-34.67
rd84	2598	-	2062	-	1965	1371	**30.23**
root	3486	1811	-	-	1583	1439	**9.1**
sao2	7893	3767	5147	-	-	2561	**32.01**
spla	-	-	49419	-	45478	26251	**42.28**
sqn	2170	-	-	1183	-	831	**29.75**
sqr6	1090	583	-	-	-	557	**4.46**
sqrt8	584	-	461	-	-	298	**35.36**
squar5	476	-	251	-	-	227	**9.56**
t481	237	-	237	-	-	205	**13.5**
table3	86173	-	35807	-	32286	16997	**47.35**
urf3	-	53157	-	56766	-	48836	**8.13**
z4	674	489	-	260	-	310	-19.23

5.6 Summary

In this chapter, a representation has been proposed to classify a pair of gates based on its control lines characteristics. This classification helps in identifying the gate pairs that can be optimized as opposed to those that cannot be. In addition, a set of optimization techniques has been proposed which can be applied on classifications that do not have existing techniques to reduce the cost. A 'greedy' optimization algorithm has been presented which uses these classifications and optimization techniques to improve the given reversible circuit.

Chapter 6

Conclusions and Future Work

In this thesis, a set of techniques has been proposed to reduce the quantum cost of reversible circuits. These techniques consist of cube decomposition and CCL transformation followed by an algorithm that combines these two techniques.

While the proposed decomposition technique helped in eliminating the redundant gates during the decomposition process, the cube decomposition method integrates the same with ESOP based synthesis to realize a given reversible function with reduced quantum cost.

Next, the conditions for which the above technique results in cost reduction have been derived. From these conditions, it has been shown that the effectiveness of the technique is proportional to the number of common control lines between a gate pair. A transformation approach, which transforms a given gate pair by increasing the number of common control lines, has been proposed and the decomposition technique has been applied resulting in further cost reduction.

Further, a classification technique for gate pairs has been proposed based on the pairs' control line characteristics. This helps in identifying gate pairs on which appropriate optimization techniques can be applied. Also, another set of techniques has been introduced to reduce the cost of gate pairs that do not have any existing optimization techniques defined on them. Finally, an algorithm that uses the classification and the proposed techniques to reduce the overall cost of a reversible circuit has been proposed.

The methodology described above has been applied on a set of reversible benchmark circuits and compared with those available in the literature. Results indicate that the proposed methodology leads to significant improvement in the quantum cost of the reversible circuits.

Future Work

The optimization techniques presented in this thesis were mainly focused on gate pairs. These techniques can further be expanded to more than two gates. Additionally, while the quantum cost of reversible circuits has been calculated using NCV gate library, more recent efforts have focused on Clifford+T gates because of their fault tolerant nature. It would be interesting to explore the cost effectiveness of the proposed methodology in this scenario.

Bibliography

[1] R. Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM J. Res. Dev.*, 5(3):183–191, jul 1961.

[2] Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483(7388):187–189, 2012.

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

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

[5] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information.* Cambridge university press, 2000.

[6] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. pages 124–134, 1994.

[7] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[8] Robert Cuykendall and David R Andersen. Reversible optical computing circuits. *Optics Letters*, 12(7):542–544, 1987.

[9] I Ciapurin, LB Glebov, and VM Smirnov. A scheme for efficient quantum computation with linear optics. *Phys. Rev. Lett*, 86(22):51885191, 2001.

[10] Wei-Bo Gao, Ping Xu, Xing-Can Yao, Otfried Gühne, Adán Cabello, Chao-Yang Lu, Cheng-Zhi Peng, Zeng-Bing Chen, and Jian-Wei Pan. Experimental realization of a controlled-not gate with four-photon six-qubit cluster states. *Physical review letters*, 104(2):020501, 2010.

[11] Himanshu Thapliyal and MB Srinivas. The need of dna computing: reversible designs of adders and multipliers using fredkin gate. In *Proc. SPIE*, volume 6050, page 605010, 2005.

[12] Tao Song, Shudong Wang, and Xun Wang. The design of reversible gate and reversible sequential circuit based on dna computing. In *Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on*, volume 1, pages 114–118. IEEE, 2008.

[13] John Patrick McGregor and Ruby B Lee. Architectural enhancements for fast subword permutations with repetitions in cryptographic applications. In *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*, pages 453–461. IEEE, 2001.

[14] Robert Glück and Masahiko Kawabe. A method for automatic program inversion based on lr (0) parsing. *Fundamenta Informaticae*, 66(4):367–395, 2005.

[15] Robert Wille, Rolf Drechsler, Christof Osewold, and Alberto Garcia-Ortiz. Automatic design of low-power encoders using reversible circuit synthesis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 1036–1041, San Jose, CA, USA, 2012. EDA Consortium.

[16] R. Wille, O. Keszocze, S. Hillmich, M. Walter, and A. Garcia-Ortiz. Synthesis of approximate coders for on-chip interconnects using reversible logic. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1140–1143, March 2016.

[17] Mehdi Saeedi and Igor L Markov. Synthesis and Optimization of Reversible Circuits - a Survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.

[18] Robert Wille and Rolf Drechsler. *Towards a Design Flow for Reversible Logic*. Springer Science & Business Media, 2010.

[19] Daniel GroBe, Xiaobo Chen, and Rolf Drechsler. Exact toffoli network synthesis of reversible logic using boolean satisfiability. In *Design, Applications, Integration and Software, 2006 IEEE Dallas/CAS Workshop on*, pages 51–54. IEEE, 2006.

[20] Daniel Große, Xiaobo Chen, Gerhard W Dueck, and Rolf Drechsler. Exact sat-based toffoli network synthesis. In *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pages 96–101. ACM, 2007.

[21] Daniel Große, Robert Wille, Gerhard W. Dueck, and Rolf Drechsler. Exact Multiple-Control Toffoli Network Synthesis With SAT Techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(5):703–715, 2009.

[22] Gerhard W Dueck, Dmitri Maslov, and D Michael Miller. Transformation-based synthesis of networks of toffoli/fredkin gates. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 1, pages 211–214. IEEE, 2003.

[23] D.M. Miller, D. Maslov, and G.W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323, June 2003.

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

[25] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4), September 2007.

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

[27] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. Reversible circuit synthesis using a cycle-based approach. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 6(4):13, 2010.

[28] Mehdi Saeedi, Mona Arabzadeh, Morteza Saheb Zamani, and Mehdi Sedighi. Block-based quantum-logic synthesis. *arXiv preprint arXiv:1011.2159*, 2010.

[29] Robert Wille and Rolf Drechsler. BDD-based Synthesis of Reversible Logic for Large Functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275. ACM, 2009.

[30] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 85–92. IEEE, 2012.

[31] K Fazel, M Thornton, and JE Rice. ESOP-based Toffoli Gate Cascade Generation. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209. Citeseer, 2007.

[32] JE Rice, KB Fazel, MA Thornton, and KB Kent. Toffoli Gate Cascade Generation using ESOP Minimization and QMDD-based Swapping. *Proceedings of the Reed-Muller Workshop (RM2009)*, pages 63–72, 2009.

[33] Yasaman Sanaee and Gerhard W Dueck. ESOP-based Toffoli Network Generation with Transformations. In *40th IEEE International Symposium on Multiple-Valued Logic (ISMVL), 2010*, pages 276–281. IEEE, 2010.

[34] JE Rice and NM Nayeem. Ordering techniques for ESOP-based Toffoli cascade generation. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim), 2011*, pages 274–279. IEEE, 2011.

[35] Kamalika Datta, Alhaad Gokhale, Indranil Sengupta, and Hafizur Rahaman. An ESOP-Based Reversible Circuit Synthesis Flow Using Simulated Annealing. In *Applied Computation and Security Systems*, volume 305 of *Advances in Intelligent Systems and Computing*, pages 131–144. Springer India, 2015.

[36] Noor M Nayeem and Jacqueline E Rice. A Shared-cube Approach to ESOP-based Synthesis of Reversible Logic. *Facta universitatis-series: Electronics and Energetics*, 24(3):385–402, 2011.

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

[38] Kamalika Datta, Indranil Sengupta, and Hafizur Rahaman. A Post-Synthesis Optimization Technique for Reversible Circuits Exploiting Negative Control Lines. *IEEE Transactions on Computers*, 64(4):1208–1214, 2015.

[39] Kamalika Datta, Gaurav Rathi, Robert Wille, Indranil Sengupta, Hafizur Rahaman, and Rolf Drechsler. Exploiting negative control lines in the optimization of reversible circuits. In *Proceedings of the 5th International Conference on Reversible Computation*, RC'13, pages 209–220, Berlin, Heidelberg, 2013. Springer-Verlag.

[40] D Michael Miller, Robert Wille, and Rolf Drechsler. Reducing reversible circuit cost by adding lines. In *40th IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, pages 217–222. IEEE, 2010.

[41] Robert Wille, Mathias Soeken, and Rolf Drechsler. Reducing the number of lines in reversible circuits. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 647–652. IEEE, 2010.

[42] DM Miller, R Wille, and GW Dueck. Synthesizing reversible circuits from irreversible specifications using reed-muller spectral techniques. In *Proc. Reed-Muller Workshop*, pages 87–96, 2009.

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

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

[45] Marek Perkowski, Robert Fiszer, Pawel Kerntopf, and Martin Lukac. An approach to synthesis of reversible circuits for partially specified functions. In *Nanotechnology (IEEE-NANO), 2012 12th IEEE Conference on*, pages 1–6. IEEE, 2012.

[46] Mathias Soeken, Robert Wille, Oliver Keszocze, D Michael Miller, and Rolf Drechsler. Embedding of large boolean functions for reversible logic. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(4):41, 2016.

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

[48] Edward Fredkin and Tommaso Toffoli. Conservative Logic. *International Journal of Theoretical Physics*, 21(3/4), 1982.

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

[50] R. Wille, D. Grosse, L. Teuber, G.W. Dueck, and R. Drechsler. RevLib: An Online Resource for Reversible Functions and Reversible Circuits. In *Multiple Valued*

Logic, 2008. ISMVL 2008. 38th International Symposium on, pages 220–225, May 2008.

[51] Tsutomu Sasao. And-exor expressions and their optimization. *KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE*, pages 287–287, 1993.

[52] Tsutomu Sasao. An exact minimization of and-exor expressions using bdds. In *IFIP WG10. 5 Workshop on Applications of Reed-Muller Expansion in Circuit Design*, 1993.

[53] Tsutomu Sasao. An exact minimization of and-exor expressions using reduced covering functions. In *Proc. of the Synthesis and Simulation Meeting and International Interchange*, pages 374–383, 1993.

[54] Takashi Hirayama and Yasuaki Nishitani. Exact minimization of and–exor expressions of practical benchmark functions. *Journal of Circuits, Systems, and Computers*, 18(03):465–486, 2009.

[55] Stergios Stergiou and George Papakonstantinou. Exact minimization of esop expressions with less than eight product terms. *Journal of Circuits, Systems, and Computers*, 13(01):1–15, 2004.

[56] Tsutomu Sasao. Exmin2: a simplification algorithm for exclusive-or-sum-of-products expressions for multiple-valued-input two-valued-output functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):621–632, 1993.

[57] Alan Mishchenko and Marek Perkowski. Fast Heuristic Minimization of Exclusive-sums-of-products. *Proceedings of the 5th Reed-Muller Workshop*, pages 242–250, 2001.

[58] Rolf Drechsler, Alexander Finder, and Robert Wille. Improving ESOP-Based Synthesis of Reversible Logic Using Evolutionary Algorithms. In *Applications of Evolutionary Computation*, volume 6625 of *Lecture Notes in Computer Science*, pages 151–161. Springer Berlin Heidelberg, 2011.

[59] Chandan Bandyopadhyay, Hafizur Rahaman, and Rolf Drechsler. Improved Cube List Based Cube Pairing Approach for Synthesis of ESOP Based Reversible Logic. In *Transactions on Computational Science XXIV*, volume 8911 of *Lecture Notes in Computer Science*, pages 129–146. Springer Berlin Heidelberg, 2014.

[60] Robert Wille, Mathias Soeken, Christian Otterstedt, and Rolf Drechsler. Improving the mapping of reversible circuits to quantum circuits using multiple target lines. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 145–150. IEEE, 2013.

List of Publications

Publications related to this thesis

- Sai Phaneendra P, Chetan V, and M. B. Srinivas, “Optimizing Reversible Circuits using Gate Pair Classification”, *ACM Journal on Emerging Technologies in Computing Systems.* (Communicated)
- Sai Phaneendra P, Chetan V, and M. B. Srinivas, “An ESOP Based Cube Decomposition Technique for Reversible Circuits”, *Lecture Notes in Computer Science, vol. 10301*, Springer, Cham, 2017, pp. 127--140.
- Sai Phaneendra P, Chetan V, and M. B. Srinivas, “Optimizing the Reversible Circuits Using Complementary Control Line Transformation", *Lecture Notes in Computer Science, vol. 10301*, Springer, Cham, 2017, pp. 111--126.

Other Publications

- C. Vudadha, P. S. Phaneendra, and M. B. Srinivas, “An Efficient Design Methodology for CNFET based Ternary Logic Circuits,” in *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS), .* IEEE, 2016, pp. 278–283.

- Subhankar Pal, Chetan Vudadha, P Sai Phaneendra, Sreehari Veeramachaneni, Srinivas Mandalika, "A New Design of an n-Bit Reversible Arithmetic Logic Unit", in 2014 Fifth International Symposium on Electronic System Design (ISED), 2014, pp. 224-225.

- P. S. Phaneendra, C. Vudadha, V. Sreehari, and M. B. Srinivas, "An optimized design of reversible quantum comparator," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems,* . IEEE, 2014, pp. 557–562.

- C. Vudadha, S. Katragadda, and P. S. Phaneendra, "2:1 multiplexer based design for ternary logic circuits," in *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia),* . IEEE, 2013, pp. 46–51.

- P. V. Saidutt, V. Srinivas, P. S. Phaneendra, and N. M. Muthukrishnan, "Design of Encoder for Ternary Logic Circuits," in *Microelectronics and Electronics (PrimeAsia), 2012 Asia Pacific Conference on Postgraduate Research in.* IEEE, 2012, pp. 85–88.

- C. Vudadha, P. P. Sai, V. Sreehari, and M. B. Srinivas, "CNFET based Ternary Magnitude Comparator," in *2012 International Symposium on Communications and Information Technologies (ISCIT),* . IEEE, 2012, pp. 942–946.

- Chetan Vudadha, P. Sai Phaneendra, V. Sreehari, Syed Ershad Ahmed, N. Moorthy Muthukrishnan, M. B. Srinivas, "Design of Prefix-Based Optimal Reversible Comparator," in *2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2010, pp. 201-206.

- Chetan Vudadha, P. Sai Phaneendra, V. Sreehari, Syed Ershad Ahmed, N. Moorthy Muthukrishnan, M. B. Srinivas, “Design and Analysis of Reversible Ripple, Prefix and Prefix-Ripple Hybrid Adders” in *2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2010, pp. 225-230.

- C. Vudadha, P. Sai Phaneendra, G. Makkena, V. Sreehari, N. M. Muthukrishnan, and M. B. Srinivas, “Design of CNFET based Ternary Comparator using Grouping Logic,” in *Faible Tension Faible Consommation (FTFC)*, 2012.

- C. Vudadha, G. Makkena, M. V. S. Nayudu, P. S. Phaneendra, S. E. Ahmed, S. Veeramachaneni, N. M. Muthukrishnan, and M. B. Srinivas, “Low-power Self Reconfigurable Multiplexer based Decoder for Adaptive Resolution Flash ADCs,” in *2012 25th International Conference on VLSI Design (VLSID), .* IEEE, 2012, pp. 280–285.

Biographies

Candidate Biography

Sai Phaneendra P received the M. E. degree in Microelectronics in 2011 from Birla Institute of Technology and Science (BITS)-Pilani, Hyderabad Campus, India, where he is currently working towards the Ph.D. degree. His current research interests include reversible logic and quantum computation, arithmetic circuits design and CNFET based Multi-Valued Logic Design.

Supervisor Biography

Prof. M. B. Srinivas is presently Dean, School of Engineering and Technology at B. M. L. Munjal University, Gurgaon, India. He was earlier a Professor of Electrical Engineering at Birla Institute of Technology and Science (BITS)-Pilani, Hyderabad Campus, India, from where he is currently on leave. He obtained his Ph.D. degree from Indian Institute of Science (IISc), Bangalore in 1991. His research interests include high performance logic design, VLSI arithmetic, data converters and reversible computing.