TERNARY MAX-MIN ALGEBRA WITH APPLICATION TO REVERSIBLE LOGIC SYNTHESIS

MUSHARRAT KHAN
Bachelor of Science, East West University, Bangladesh, 2014

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

MASTER OF SCIENCE

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

TERNARY MAX-MIN ALGEBRA WITH APPLICATION TO REVERSIBLE LOGIC SYNTHESIS

MUSHARRAT KHAN

Date of Defence: March 15, 2017

Dr. Jacqueline E. Rice Supervisor	Professor	Ph.D.
Dr. Howard Cheng Committee Member	Associate Professor	Ph.D.
Dr. Locke Spencer Committee Member	Assistant Professor	Ph.D.
Dr. Amir Akbary-Majdabadno Chair, Thesis Examination Committee	Professor	Ph.D.

Dedication

To them who advance reversible computing.

Abstract

Ternary reversible circuits are 0.63 times more compact than equivalent binary reversible circuits and are suitable for low-power implementations. Two notable previous works on ternary reversible circuit synthesis are the ternary Galois field sum of products (TGFSOP) expression-based method and the ternary Max-Min algebra-based method. These methods require high quantum cost and large number of ancilla inputs. To address these problems we develop an alternative ternary Max-Min algebra-based method, where ternary logic functions are represented as Max-Min expressions and realized using our proposed multiple-controlled unary gates. We also show realizations of multiple-controlled unary gates using elementary quantum gates. We develop a method for minimization of ternary Max-Min expressions of up to four variables using ternary K-maps. Finally, we develop a hybrid Genetic Algorithm (HGA)-based method for the synthesis of ternary reversible circuits. The HGA has been tested with 24 ternary benchmark functions with up to five variables. On average our method reduces quantum cost by 41.36% and requires 35.72% fewer ancilla inputs than the TGFSOP-based method. Our method also requires 74.39% fewer ancilla inputs than the previous ternary Max-Min algebra-based method.

Acknowledgments

I would like to express my heartfelt thanks and gratitude to my thesis supervisor Dr. Jacqueline E. Rice for her tireless effort for managing funds for my study, for continuous advice, encouragement, and guidance for both course work and research work, and most importantly for standing beside me during my academic and personal odds, without which this thesis would have not seen the day light. Simply saying thanks to her will be inadequate for what she has done for me.

I would also like to thank my committee members Dr. Howard Cheng and Dr. Locke Spencer for their encouragement and suggestions for improving the quality of my thesis. I specially thank Dr. Howard Cheng for his academic and administrative helps.

I must thank Administrative Support Ms. Barb Hodgson for her cordial support whenever I needed help from her.

I would like to thank anonymous reviewers of my two conference papers published in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, Canada, May 2016 for their invaluable suggestions to improve my papers that constitute a large part of my thesis.

I also thank other faculty members of the department and my fellow students for their unconditional helps and supports.

Finally, I must thank my parents for extending every possible supports and absorbing emotional pains to keep their only daughter alone staying several thousand miles away from home for this study.

This list is not an exhaustive list to thank. Anybody who directly or indirectly helped me in any aspect during my study also deserves a great thank.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation of the Thesis

It has been shown in [16] that theoretically ternary encoded realization is more compact than binary and requires 0.63 times the number of input lines. A more detailed discussion is given in Section 2.3. Ternary logic circuits are also practically realizable as described in [7]. Therefore, ternary is an alternative to traditional binary realization and may reduce the physical resources needed.

Any ternary gate (or circuit) that computes a function which is bijective can be considered to be a ternary reversible gate (or circuit). The outputs of a ternary reversible circuit are permutations of the inputs. This also means that a ternary reversible circuit has same number of inputs and outputs. Another way to describe this is to say that in ternary reversible circuits an input combination uniquely maps to an output combination, and similarly an output combination also uniquely maps to an input combination. Conventional ternary logic circuits are irreversible, since they map inputs to outputs in a many-to-one manner; that is, more than one input combination may map to a single output. A ternary irreversible circuit may have a different number of inputs and outputs. Advantages of ternary reversible circuits over irreversible circuits are discussed below.

Landauer [23] theoretically showed that binary irreversible logic operations dissipate $kTln2$ joules of heat energy when a bit of information is lost, where k is Boltzmann's constant and T is the operating temperature in kelvin. Bennett [2] showed theoretically that from a thermodynamic point of view, if a circuit is both logically and physically reversible,

then there will be no heat dissipation. Thus reversible circuits have been theorized to be energy lossless circuits. However, DeVos [6] experimentally found that in complementary metal oxide semiconductor (CMOS) reversible circuits, zero heat dissipation as suggested by Bennett is not possible. Reversible CMOS circuits still dissipate heat during switching of CMOS transistors; however the amount of heat dissipation for an equivalent operation is less than the thermodynamic limit of $kTln2$ joules, and DeVos stated that on average reversible circuits dissipate less heat than irreversible circuits. Thus binary reversible logic is a promising choice for low-power CMOS circuit design. It is generally believed that the same phenomena will also hold true for ternary reversible logic circuits [19, 32].

Quantum circuits are inherently reversible and are designed using the concepts of classical reversible circuit design [34]. Therefore, the study of reversible logic is very important in quantum computing and quantum information theory. Ternary quantum circuits are physically realizable using linear ion trap quantum technology [33]. Therefore, it is theoretically possible to develop ternary quantum circuits using ternary reversible circuit design concepts.

Further details are offered in Section 2.7.

The literature includes a number of works on synthesis of ternary reversible logic circuits. Generalized design of ternary reversible logic circuits is discussed in Chapter 3. Among the reported works, ternary Galois field sum of products (TGFSOP) expression-based design of ternary reversible logic circuits is the most practical method for functions with many input variables [16]. In this approach, ternary logic functions are first expressed as minimized TGFSOP expressions and then the TGFSOP expressions are realized as cascades of unary, M-S, Feynman, and Toffoli gates. However, this method requires exponential time to find an optimal subset of ternary Galois field expansions (TGFE) and, therefore, an evolutionary algorithm is used to obtain a near optimal solution in [16]. In addition, realization of the macro-level Feynman and Toffoli gates requires a very high quantum cost [15], which is the number of elementary quantum gates needed to realize a macro-

level gate. Realizations of reversible circuits sometimes need use of constant-initialized working inputs in addition to the primary inputs of the circuit. These constant-initialized working inputs are known as ancilla inputs. The number of ancilla inputs required in TGF-SOP expression-based circuit realization is also very high. More detailed discussions on this approach are given in Section 3.3. Another work close to our proposed approach is the ternary Max-Min algebra-based synthesis of ternary reversible circuits using generalized ternary gates (GTGs) [5]. This method requires a large number of GTGs and a very high number of ancilla inputs. More detailed discussion on this method is given in Section 3.3. Therefore, new synthesis methods must be developed to reduce the quantum cost and the number of ancilla inputs of ternary logic circuits.

To address these problems we suggest an alternative ternary Max-Min algebra-based method for reversible realization of ternary logic circuits. The outcomes of this thesis are summarized in section 1.2.

1.2 Outcomes of the Thesis

The outcomes of this thesis are summarized below:

- We propose a method of representing ternary logic functions using ternary Max-Min algebra. For this purpose we use ternary reversible literals to represent input combinations as minterms. The minterms corresponding to an specified output (0, 1, or 2) are combined by Max operation to form a Max of minterms (Max-Min) expression. We also propose composite literals to represent minimized Max-Min expression for ternary logic functions. Three minimized Max-Min expressions are generated for three sub-functions producing outputs 0, 1, and 2.

- We propose multiple-controlled unary gates to realize Max-Min expressions. We show realizations of multiple-controlled unary gates using unary and M-S gates.

- We propose three architectures for realizing ternary reversible circuits. Two sub-functions producing outputs 0 and 1; or 0 and 2; or 1 and 2 are realized as cascades of multiple-controlled unary gates and the other output constant is used as an ancilla input for realizing the two selected sub-functions.

- We propose a ternary Karnaugh map-based minimization of Max-Min expressions for ternary functions with up to four variables.

- We propose a hybrid Genetic Algorithm-based method for synthesis of ternary reversible circuits and experiment with up to five variable benchmark functions. The quantum cost and the number of ancilla inputs are much less than those of the TGF-SOP expression-based method [16]. The proposed method also requires fewer ancilla inputs than that reported in [5].

From the outcome of the thesis two conference papers are published as listed in references [14] and [13].

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows:

- In Chapter 2 we discuss the background concepts on ternary reversible logic. We introduce conventional binary logic first and then introduce ternary logic by drawing analogies with binary logic. We also discuss the advantages of reversible logic over conventional irreversible logic. Finally, we introduce the metrics used for measuring the realization complexities of a reversible circuit.

- In Chapter 3 we present an extensive review on synthesis methods of ternary reversible circuits. We also discuss realizations of macro-level ternary gates using elementary gates.

- In Chapter 4 we discuss our proposed method of representing ternary logic functions using ternary Max-Min algebra.

- In Chapter 5 we introduce the concept of our proposed macro-level ternary multiple-controlled unary gates. We also discuss our proposed method for realization of multiple-controlled unary gates using elementary quantum gates such as unary gates and M-S gates.

- In Chapter 6 we discuss our proposed method of realizing ternary Max-Min expressions using ternary multiple-controlled unary gates. We also discuss our proposed architecture for realizing a ternary logic function using two of the three sub-functions producing outputs 0, 1, and 2.

- In Chapter 7 we discuss our proposed ternary Karnaugh map-based minimization of ternary logic functions of up to four variables.

- In Chapter 8 we discuss our proposed hybrid Genetic Algorithm-based method for synthesis of ternary reversible circuits.

- In Chapter 9 we present our hybrid Genetic Algorithm-based synthesis results of up to five variable ternary benchmark logic functions. We also compare our results with those of [16] and [5].

- In Chapter 10 we conclude the thesis by summarizing the contributions of this thesis and discussing possible future works.

Chapter 2

Background

2.1 Introduction

In this chapter we introduce the background concepts for ternary reversible logic, which are useful for understanding the literature review as well as our proposed method of synthesizing ternary reversible circuits using ternary Max-Min algebra in the following chapters.

This chapter is organized as follows:

- In Section 2.2 we introduce conventional binary logic with emphasis on Reed-Muller representations of binary logic functions. Discussion of Reed-Muller representations of binary logic functions is helpful for understanding ternary Galois field sum of products (TGFSOP)-based synthesis of ternary reversible logic. This concept will also be very helpful for understanding our proposed ternary Karnaugh map (K-map)-based minimization in Chapter 7.

- In Section 2.3 we introduce ternary logic and discuss the advantages and disadvantages of ternary logic over conventional binary logic.

- In Section 2.4 we introduce ternary Max-Min algebra and ternary logic function representation using ternary Max-Min expressions. The knowledge of ternary Max-Min expressions will be useful for understanding the literature review of work [5], which makes use of Max-Min expression-based synthesis of ternary reversible circuits.

- In Section 2.5 we introduce ternary Galois field and ternary logic function representation using ternary Galois field sum of products (TGFSOP) expressions. The

knowledge of TGFSOP expressions will be useful for understanding the literature review of works [20, 21, 22, 16, 18], which make use of TGFSOP-based synthesis of ternary reversible circuits.

- In Section 2.6 we introduce the structures of ternary Karnaugh maps (K-maps) for minimization of ternary logic functions. The knowledge of ternary K-maps will be useful for understanding the literature review of work [5] where ternary K-maps are used. The knowledge of ternary K-maps will be particularly useful for understanding our proposed ternary K-map-based minimization in Chapter 7.

- In Section 2.7 we introduce reversible logic. We discuss the concept of an irreversible function and a reversible function. We then discuss the advantages and disadvantages of reversible logic over irreversible logic.

- In Section 2.8 we introduce ternary reversible logic. We first introduce commonly used ternary reversible gates. We then discuss TGFSOP-based ternary reversible logic synthesis using ternary reversible gates.

- In Section 2.9 we introduce two metrics for measuring the implementation complexities of a reversible circuit.

2.2 Reed-Muller Representations of Binary Logic Functions

Conventional digital circuits use binary logic. A binary logic function is a mapping $F : \{0,1\}^n \mapsto \{0,1\}$. Binary logic functions are defined, represented, and realized using Boolean algebra. Readers are referred to any standard textbook such as [26] for detailed discussions on binary logic gates, Boolean algebra, representation of binary logic functions using Boolean algebra, minimization of Boolean functions using K-maps, and realization of Boolean expressions using logic gates. In this section we discuss Reed-Muller representations of binary logic functions. Understanding of Reed-Muller representations of binary

logic functions will be helpful for understanding ternary Galois field sum of products (TGF-SOP) expression-based ternary logic synthesis of works [20, 21, 22, 16, 18] as well as our proposed ternary K-map-based method for minimization of ternary Max-Min expressions in Chapter 7.

In conventional Boolean logic, binary logic functions can be represented as sum of products (SOP) expressions and realized as AND-OR circuits (see [26]). In Reed-Muller expressions, binary logic functions are expressed as exclusive-OR sum of products (ESOP) expressions and realized as AND-EXOR circuits. There are different forms of Reed-Muller expressions [36]. The most commonly used form of Reed-Muller expression is called the exclusive-OR sum of products (ESOP) [36, 37].

Small binary logic functions can be minimized as ESOP expressions using K-maps [36]. We know that $A \oplus A = 0$ but $A \oplus A \oplus A = A$. This property of the EXOR operation means that on a K-map we must cover 1s an odd number of times. This property also allows including a 0 within a group of 1s on a K-map if that 0 is covered an even number of times. An even number of coverings of a 0 does not have any effect on the value of the function. Readers can see [36] for more details of K-map-based minimization of ESOP expressions.

A Boolean function can be simplified as an ESOP expression using the following procedure:

1. Represent the binary logic function on a K-map.

2. Identify the largest possible groups of cells such that a 1 is covered an odd number of times and a 0 is covered an even number of times. It is desirable to have as small a number of groups as possible. It may be necessary to form a group within another larger group to maintain the constraint that a 1 is covered an odd number of times and a 0 is covered an even number of times.

3. Determine the product terms for each group of cells. Combine the product terms using EXOR operators to form the simplified ESOP expression.

The above procedure is demonstrated using an example 4-variable binary logic function in Table 2.1 as follows:

1. The binary logic function in Table 2.1 is represented on the K-map in Figure 2.1.

2. The groups of cells are identified as groups 1, 2, 3, and 4, respectively. The cell 0010 containing a 0 is covered by groups 1 and 2; the cell 0100 containing a 0 is covered by groups 1 and 3; the cell 0101 containing a 0 is covered by groups 1 and 4; and the cell 0000 containing a 1 is covered by groups 1, 2, and 3. These groupings satisfy the requirement of ESOP minimization.

3. The product term for group 1 is $\overline{A}$; for group 2 is $\overline{B}\,\overline{D}$; for group 3 is $\overline{A}\,\overline{C}\,\overline{D}$; and for group 4 is $B\overline{C}D$. The simplified ESOP expression for the binary logic function in Table 2.1 is shown in (2.1).

$$F(A,B,C,D) = \overline{A} \oplus \overline{B}\,\overline{D} \oplus \overline{A}\,\overline{C}\,\overline{D} \oplus B\overline{C}D \tag{2.1}$$

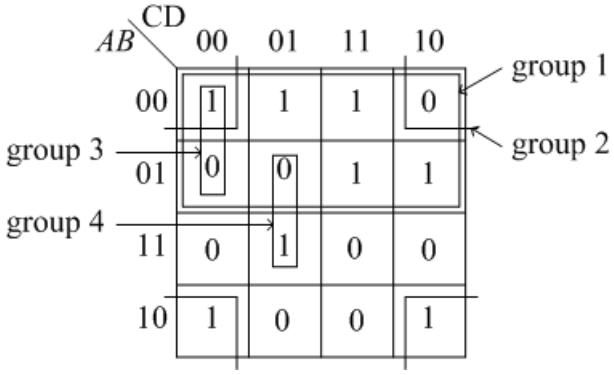

Figure 2.1: ESOP minimization for the Boolean function in Table 2.1.

It is shown in [37] that the ESOP expression is either smaller or at most equal to the corresponding SOP expression for a given binary logic function.

The realization of the ESOP expression of (2.1) is shown in Figure 2.2.

Table 2.1: An example 4-variable binary logic function.

$ABCD$	$F(A,B,C,D)$
0000	1
0001	1
0010	0
0011	1
0100	0
0101	0
0110	1
0111	1
1000	1
1001	0
1010	1
1011	0
1100	0
1101	1
1110	0
1111	0

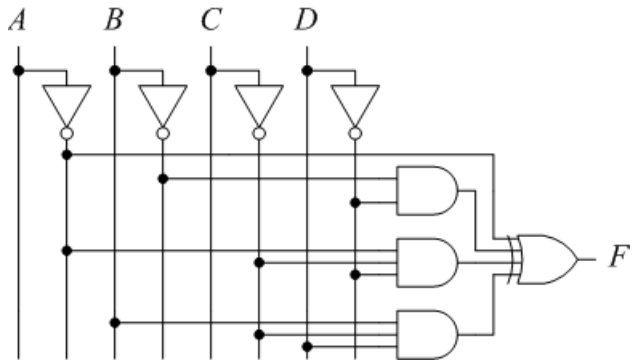

Figure 2.2: Realization of the ESOP expression (2.1).

2.3 Ternary Logic

A ternary logic function is a mapping $F : \{0,1,2\}^n \mapsto \{0,1,2\}$. In this section, we discuss the advantages and disadvantages of ternary logic over binary logic.

2.3.1 Advantages of Ternary Logic

A ternary logic representation enables a more compact and efficient information encoding than the equivalent binary logic representation [16]. The argument stated in [16] is as follows: if we assume that a digital circuit has N possible input combinations; then a binary circuit requires $\log_2 N$ input lines and a ternary circuit requires $\log_3 N$ input lines. Thus we have

$$\begin{aligned} \frac{\text{Number of input lines for ternary circuit}}{\text{Number of input lines for binary circuit}} &= \frac{\log_3 N}{\log_2 N} \\ &= \frac{\frac{\log_2 N}{\log_2 3}}{\log_2 N} \\ &= \frac{1}{\log_2 3} \\ &= 0.63 \end{aligned} \tag{2.2}$$

Therefore, a ternary encoded realization of a given binary logic function should require 0.63 times the input lines than the corresponding binary realization.

2.3.2 Disadvantages of Ternary Logic

Although ternary logic circuits should require fewer input lines than the equivalent binary logic circuits, ternary logic circuits are currently not a practical choice. The reasons are (i) ternary hardware implementation technology is still in the theoretical, simulation, and laboratory test levels [7], (ii) representing three ternary logic levels (0, 1, and 2) using voltage levels of existing technology is not yet effectively defined, and (iii) no computational model and programming language is developed. However, [7] gives simulation results of ternary circuit implementation using complementary metal oxide semiconductor (CMOS), resonant tunneling diode (RTD), and carbon nano tube technologies, demonstrating that ternary logic may be a choice for future computing.

2.4 Max-Min Algebra-Based Representation of Ternary Logic Functions

In this section we introduce ternary Max-Min algebra and ternary Max-Min algebra-based representation of ternary logic functions.

2.4.1 Ternary Max-Min Algebra

Ternary Max-Min algebra has the set of values $T = \{0,1,2\}$ and the following three operations:

Max (+): For $x, y \in T, x + y = \max(x, y)$

Min ($\cdot$): For $x, y \in T, x \cdot y$ (or xy) $= \min(x, y)$

Complement (or Negation) ($^-$): For $x \in T, \bar{x} = 2 - x$

Ternary Max, Min, and Complement operations are shown in the truth tables of Tables 2.2, 2.3, and 2.4, respectively.

Table 2.2: Ternary Max operation.

xy	$x + y = \max(x, y)$
00	0
01	1
02	2
10	1
11	1
12	2
20	2
21	2
22	2

Ternary Max-Min algebra satisfies the following properties for $x, y, z \in T$:

Commutativity: (i) $x + y = y + x$ and (ii) $xy = yx$

Associativity: (i) $x + (y + z) = (x + y) + z$ and (ii) $x(yz) = (xy)z$

Distributivity: (i) $x + (yz) = (x + y)(x + z)$ and (ii) $x(y + z) = xy + xz$

Table 2.3: Ternary Min operation.

xy	$x \cdot y = \min(x,y)$
00	0
01	0
02	0
10	0
11	1
12	1
20	0
21	1
22	2

Table 2.4: Ternary Complement (or Negation) operation.

x	$\overline{x} = 2 - x$
0	2
1	1
2	0

Idempotency: (i) $x + x = x$ and (ii) $xx = x$

Identity: (i) $x + 0 = x$, (ii) $x + 2 = 2$, (iii) $x \cdot 0 = 0$, and (iv) $x \cdot 2 = x$

Involution: $\overline{\overline{x}} = x$

De Morgan's Law: (i) $\overline{(x+y)} = \overline{x} \cdot \overline{y}$ and (ii) $\overline{(xy)} = \overline{x} + \overline{y}$

2.4.2 Ternary Max-Min Expression

A ternary logic function can be represented as a Max-Min expression. Ternary Max-Min expressions are defined as follows:

Variable: Any symbol that takes value from the set $T \in \{0,1,2\}$ is a ternary variable.

Literal: Literals are transformed forms of a variable. They are used to form Max-Min expressions. In the literature two types of literals are commonly used: 1-reduced Post literals [5] and 2-reduced post literals [39, 38, 10].

A 1-reduced Post literal of a variable x is represented as x^i, where $i \in \{0,1,2\}$. When

$x = i$, then $x^i = 1$, otherwise $x^i = 0$. The 1-reduced Post literals of a variable are shown in Table 2.5. A 2-reduced Post literal of a variable x is represented as x^i, where $i \in \{0,1,2\}$. When $x = i$, then $x^i = 2$, otherwise $x^i = 0$. The 2-reduced Post literals of a variable are shown in Table 2.6. This thesis makes use of different sets of literals to form Max-Min expressions as discussed in Chapter 4.

Table 2.5: 1-Reduced Post literals of a variable.

Variable	1-Reduced Post Literals		
x	x^0	x^1	x^2
0	1	0	0
1	0	1	0
2	0	0	1

Table 2.6: 2-Reduced Post literals of a variable.

Variable	2-Reduced Post Literals		
x	x^0	x^1	x^2
0	2	0	0
1	0	2	0
2	0	0	2

Minterm: When literals of variables of a function are combined using the Min operation, then the term is called a minterm. For example, for a 3-variable ternary logic function $F(x,y,z)$, xyz and xz are two examples of minterms.

Max-Min Expression: When two or more minterms are combined using Max operations, then the expression is called a Max of minterms (Max-Min) expression. For example, for a 3-variable ternary logic function $F(x,y,z)$, $F(x,y,z) = xy + yz + xyz$ is an example of a Max-Min expression.

2.4.3 Ternary Logic Function Representation Using Max-Min Expressions

There have been many attempts to represent ternary logic functions using ternary Max-Min algebra. Among them is the Max-Min algebra using 2-reduced Post literals [39, 38, 10].

In [10], for a given ternary logic function F, two sub-functions F_1 (corresponding to the output 1) and F_2 (corresponding to the output 2) are represented as Max-Min expressions. For determining the minterm for an input combination, an input value is assigned to a 2-reduced Post literal as shown in Table 2.6. For example, for a 3-variable ternary logic function $F(x,y,z)$, the minterm corresponding to the input combination 012 is $x^0y^1z^2$. When the input combination is $x=0$, $y=1$, and $z=2$, then $x^0=2$, $y^1=2$, and $z^2=2$. In that case, the minterm $x^0y^1z^2=2\cdot 2\cdot 2=2$. For other input combinations, the minterm $x^0y^1z^2$ will be 0. For example, for the input combination 011, $x=0$, $y=1$, and $z=1$, then $x^0=2$, $y^1=2$, and $z^2=0$. In this case, the minterm $x^0y^1z^2=2\cdot 2\cdot 0=0$. The total function $F(x,y,z)$ is represented as shown in (2.3).

$$F(x,y,z)=F_2(x,y,z)+1\cdot F_1(x,y,z). \tag{2.3}$$

From (2.3) we see that if an input combination produces output 1, then $F_1(x,y,z)=2$, $F_2(x,y,z)=0$, and $F(x,y,z)=F_2(x,y,z)+1\cdot F_1(x,y,z)=0+1\cdot 2=1$. If an input combination produces output 2, then $F_1(x,y,z)=0$, $F_2(x,y,z)=2$, and $F(x,y,z)=F_2(x,y,z)+1\cdot F_1(x,y,z)=2+1\cdot 0=2$. If an input combination produces output 0, then $F_1(x,y,z)=0$, $F_2(x,y,z)=0$, and $F(x,y,z)=F_2(x,y,z)+1\cdot F_1(x,y,z)=0+1\cdot 0=0$.

The truth table of a ternary half-adder is shown in Table 2.7. The expressions for the outputs are shown in (2.4) and (2.5).

$$C_{out}=0+1\cdot(A^1B^2+A^2B^1+A^2B^2) \tag{2.4}$$

$$S=A^0B^2+A^1B^1+A^2B^0+1\cdot(A^0B^1+A^1B^0+A^2B^2) \tag{2.5}$$

Table 2.7: Truth table of a ternary half-adder.

AB	$C_{out}S$
00	00
01	01
02	02
10	01
11	02
12	10
20	02
21	10
22	11

In [10], a ternary K-map-based minimization technique for ternary logic functions of up to 3 variables is presented. The ternary K-map is introduced in Section 2.6. Readers are referred to [10] for detailed discussion.

In [7], implementations of Max-Min expressions using decoders for generating all three 2-reduced Post literals, Min gates, and Max gates are presented. These gates are implementable using complementary metal oxide semiconductor (CMOS), resonant tunneling diode (RTD), and carbon nano tube technologies (see [7] for details).

We will discuss another application of ternary Max-Min algebra in the literature review of work [5] in Section 3.3, where 1-reduced Post literals are used.

2.5 Galois Field-Based Representation of Ternary Logic Functions

In this section we introduce ternary Galois field (TGF) and TGF-based representation of ternary logic functions.

2.5.1 Ternary Galois Field

A ternary Galois field (TGF or GF3) is a finite field, which consists of a set of elements $T = \{0,1,2\}$ and two binary operations: addition (denoted by $\oplus$) and multiplication (denoted by $\cdot$ or juxtaposition) as defined in Tables 2.8 and 2.9, respectively. Readers should

note that earlier in this thesis the $\oplus$ symbol is also used to represent the binary EXOR operation. Both uses are consistent with the standard usages in the literature. For the remainder of this thesis the $\oplus$ symbol will denote GF3 addition.

Table 2.8: GF3 addition operation.

xy	$x \oplus y$
00	0
01	1
02	2
10	1
11	2
12	0
20	2
21	0
22	1

Table 2.9: GF3 multiplication operation.

xy	$x \cdot y$
00	0
01	0
02	0
10	0
11	1
12	2
20	0
21	2
22	1

The TGF or GF3 satisfies the following properties for $x, y, z \in T$:

- Associativity: (a) $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ and (b) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- Commutativity: (a) $x \oplus y = y \oplus x$ and (b) $x \cdot y = y \cdot x$
- Identity: (a) $x \oplus 0 = x$ and (b) $x \cdot 1 = x$

- Distributivity: $x \cdot (y \oplus z) = (x \cdot y) \oplus (x \cdot z)$

2.5.2 Ternary Galois Field Sum of Products (TGFSOP) Expressions

A ternary logic function can be represented as a ternary Galois field sum of products (TGFSOP) expression. TGFSOP expressions are inspired by binary ESOP expressions. TGFSOP expressions are defined as follows:

Variable: Any symbol that takes value from the set $T \in \{0,1,2\}$ is a ternary variable.

Literal: Literals are transformed forms of a variable used to determine TGFSOP expressions. In practice different applications of ternary Galois field use different sets of literals of a variable. We will discuss literals used in each application during discussions of those applications.

Product Term: When literals of variables of a function are combined using the product (or multiplication) operation, then the term is called a product term. For example, for a 3-variable ternary logic function $F(x,y,z)$, xyz and xz are two examples of product terms.

TGFSOP Expression: When two or more product terms are combined using the sum (or addition) operations, then the expression is called a TGFSOP expression. For example, for a 3-variable ternary logic function $F(x,y,z)$, $F(x,y,z) = xy \oplus yz \oplus xyz$ is an example of a TGFSOP expression.

We will further discuss ternary logic function representation using TGFSOP expressions in the literature review of works [20, 21, 22, 16, 18] in Section 3.3.

2.6 Ternary Karnaugh Map

In this section we discuss ternary Karnaugh maps (K-maps), which are used in ternary logic synthesis applications for minimizing ternary logic functions. In particular in Chapter 7 we will propose a ternary K-map-based method for minimization of Max-Min expressions for reversible realizations of ternary logic functions.

2.6.1 Structures of Ternary K-Maps

The structures of two-variable, three-variable, and four-variable ternary K-maps are shown in Figures 2.3(a), 2.3(b), and 2.3(c), respectively.

Consider the top-left cell of Figure 2.3(a). Here the value of the variable A is 0 and the value of the variable B is 0. So the variable combination for this cell is $AB = 00$. In a similar manner the input combination corresponding to any cell can be determined.

If only one variable varies over two values, then the two cells are adjacent. In Figure 2.3(a) a cell at the left end of a row and a cell at the right end of the same row are considered as adjacent cells. Similarly, a cell at the top end of a column and a cell at the bottom end of the same column are considered as adjacent cells. In general, if two cells differ over 0 and 1; or 0 and 2; or 1 and 2 in only one variable, then they are adjacent and form a group of two cells. If four cells form a square, where two variables vary over two values, then they form a group of four cells. Similarly, if three cells differ over 0, 1, and 2 in only one variable, then they are adjacent and form a group of three cells.

2.6.2 Putting Ternary Logic Functions on a Ternary K-Map

An example two-variable ternary logic function $F(A,B)$ is shown in Table 2.10. A ternary logic function produces output 0, 1, or 2. But in many practical functions outputs corresponding to some input combinations are not definitely specified. This situation may arise for two reasons - (i) some input combinations of a function will never appear at the input of the associated circuit and thus the outputs corresponding to those input combinations are not definitely specified or (ii) the operation of the associated circuit is such that outputs corresponding to some input combinations may be either 0, 1, or 2 without affecting the functional operation of the circuit, thus the outputs corresponding to those input combinations are not definitely specified. These types of outputs are called don't care outputs and represented by an x. In Table 2.10 the input combinations 02, 12, and 22 produce don't care (x) outputs. This type of function is called an incompletely specified function.

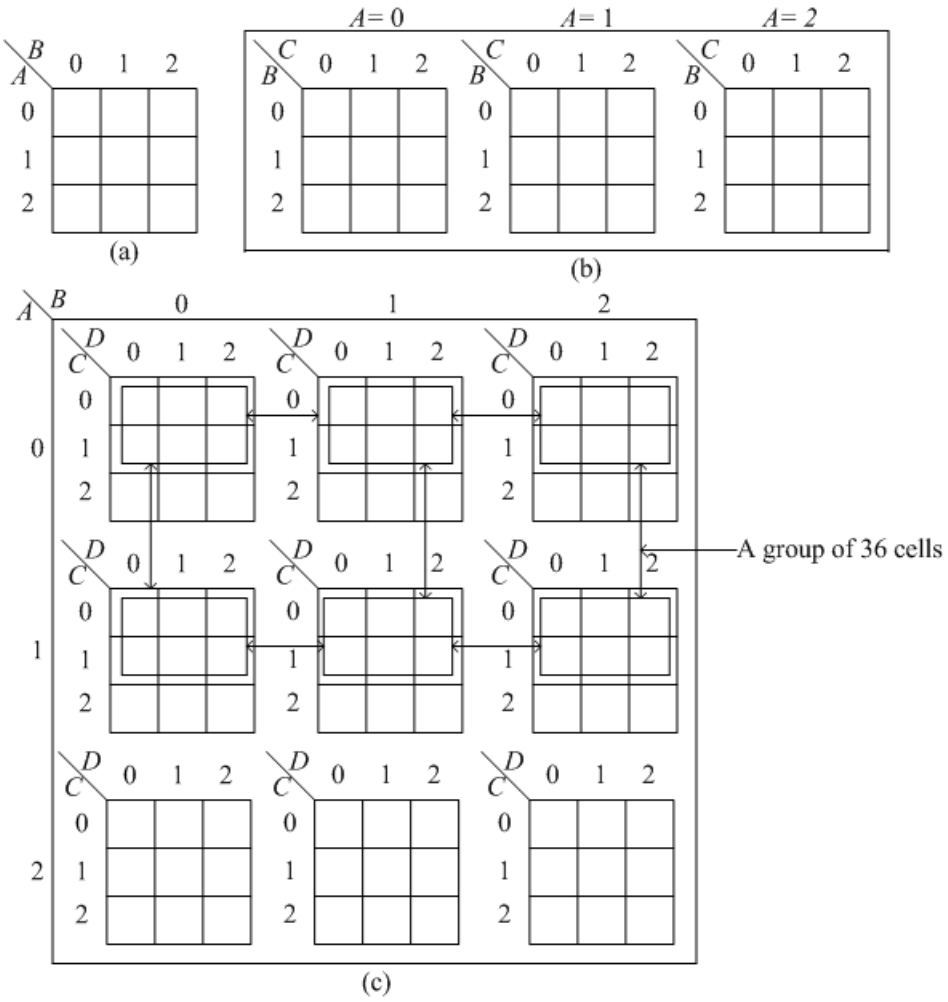

Figure 2.3: Structures of (a) a two-variable, (b) a three-variable, and (c) a four-variable ternary K-map.

The function $F(A,B)$ is put on the two-variable ternary K-map in Figure 2.4. The input combination $AB = 00$ produces an output 0. A 0 is put in the cell corresponding to the input combination $AB = 00$. The input combination $AB = 01$ produces an output 1. A 1 is put in the cell corresponding to the input combination $AB = 01$. The input combination $AB = 02$ produces a don't care (x) output. An x is put in the cell corresponding to the input combination $AB = 02$. Similarly other values of the function $F(A,B)$ are put on the map.

Grouping of cells on a ternary K-map depends on the specific application of ternary logic function minimization. We discuss grouping of cells on ternary K-map and corresponding minimization techniques in the literature review (Section 3.3), where an existing

Table 2.10: An example two-variable ternary logic function.

AB	$F(A,B)$
00	0
01	1
02	x
10	1
11	0
12	x
20	2
21	1
22	x

A \ B	0	1	2
0	0	1	x
1	1	0	x
2	2	1	x

Figure 2.4: Putting the ternary function $F(A,B)$ of Table 2.10 on a two-variable ternary K-map.

approach [5] is described, and in Chapter 7, where we propose our own approach to ternary Max-Min minimization using ternary K-maps.

2.7 Reversible Logic

In this section we introduce the concept of conventional irreversible logic versus reversible logic. We then discuss the advantages and disadvantages of reversible logic over irreversible logic.

2.7.1 Irreversible Logic Versus Reversible Logic

A binary irreversible function is a many-to-one mapping $F : \{0,1\}^n \mapsto \{0,1\}$, where n is the number of variables in the function. For example the EXOR function in Table 2.11 is an irreversible function, since input combinations 00 and 11 map to output 0; and input

combinations 01 and 10 map to output 1. A binary irreversible circuit may have a different number of inputs and outputs. In an irreversible circuit, the input combination corresponding to a given output cannot always be determined uniquely.

Table 2.11: Truth table of the binary EXOR function.

xy	$x \oplus y$
00	0
01	1
10	1
11	0

A ternary irreversible function is a many-to-one mapping $F : \{0,1,2\}^n \mapsto \{0,1,2\}$, where n is the number of variables in the function. For example, the ternary Max and Min functions shown in Tables 2.2 and 2.3 are irreversible functions.

A binary reversible function is a bijective mapping $F : \{0,1\}^n \mapsto \{0,1\}^n$, where n is the number of variables in the function. A binary reversible circuit has same number of inputs and outputs. Any irreversible function can be converted into reversible function by adding constant-initialized ancilla inputs (a constant-initialized input that is not part of the function input but required to realize a reversible circuit is known as ancilla input) and garbage outputs (the outputs that are not the primary outputs but required to realize a reversible circuit is known as garbage output) [28]. The irreversible EXOR function in Table 2.11 can be converted into reversible EXOR function by adding a garbage output $p = x$ and getting the desired EXOR output as $q = x \oplus y$ as shown in Table 2.12 [34]. From Table 2.12, it can be seen that the number of inputs and outputs are same. It can also be seen that all inputs uniquely map to outputs and the input corresponding to each output can be uniquely determined.

The output combinations of a reversible function is a permutation of the input combinations. A function with n inputs has 2^n input combinations. Therefore, there are $(2^n)!$ possible binary reversible functions with n inputs and outputs.

Table 2.12: Truth table of the binary reversible EXOR function.

xy	$p = x$ $q = x \oplus y$
00	00
01	01
10	11
11	10

A ternary reversible function is a bijective mapping $F : \{0,1,2\}^n \mapsto \{0,1,2\}^n$, where n is the number of variables in the function. We discuss ternary reversible logic further in Section 2.8.

2.7.2 Advantages of Reversible Logic Over Irreversible Logic

In a theoretical study, Landauer [23] showed that irreversible logic operations dissipate $kTln2$ joules of heat energy when a bit of information is lost, where k is Boltzmann's constant and T is the operating temperature in kelvin. In another theoretical study, Bennett [2] showed that from a thermodynamic point of view, if a circuit is both logically and physically reversible, then there will be no heat dissipation. Thus reversible circuits have been theorized to be energy lossless circuits. However, in a laboratory experiment, DeVos [6] found that in complementary metal oxide semiconductor (CMOS) reversible circuits, zero heat dissipation as suggested by Bennett is not possible. Reversible CMOS circuits still dissipate heat during switching of CMOS transistors; however the amount of heat dissipation for an equivalent operation is less than the thermodynamic limit of $kTln2$ joules, and DeVos stated that on average reversible circuits dissipate less heat than irreversible circuits. Thus binary reversible logic is a promising choice for low-power CMOS circuit design. It is generally believed that the same phenomena will also hold true for ternary reversible logic circuits [19, 32].

In a recent article, DeBenedictis [4] stated that current CMOS circuits use Boolean logic networks, where the main contributor to heat generation is the energy in the 0 and 1 signals

stored on wires. Using communications theory, DeBenedictis showed that when 0 and 1 are transmitted through a wire from the output of a gate and received at an input of another gate, then CV^2 joules of energy is turned into heat, where C is the capacitance in farads between the wire and the ground and V is the power supply voltage in volts. DeBenedictis also mentioned that in the literature different values of CV^2 energy are reported and the most common cited value is $CV^2 \geq 100kT$, where k is Boltzmann's constant and T is the ambient temperature in kelvin. DeBenedictis further stated that adiabatic [1] and reversible logic essentially recycle CV^2 joules of signal energy many times before dissipating it as heat, and on average the energy drawn from the power supply could be reduced by 100 times. DeBenedictis specifically mentioned that signal energy recycling cannot be done in irreversible Boolean networks. Therefore, instead of irreversible Boolean logic, reversible logic is a potential alternative for CMOS technology. Analogous to the case in binary logic circuits, in ternary logic circuits a signal (0, 1, or 2) propagates through a wire from the output of a gate to an input of another gate. In irreversible ternary CMOS circuits similar heat dissipation will happen due to signal propagation through the wires. This heat dissipation can be reduced by a significant amount by making the ternary logic circuit reversible.

Another very important attribute of reversible logic is that quantum circuits are inherently reversible. In a quantum algorithm, the corresponding circuits are designed using the concepts of classical reversible circuit design [34]. Therefore, study of reversible logic is very important in quantum computing and quantum information.

2.7.3 Disadvantages of Reversible Logic

Although reversible logic is a promising choice for low-power CMOS circuit design, it has the following disadvantages: (i) CMOS reversible hardware implementation technology is still in theoretical, simulation, and laboratory test levels [6], (ii) reversible computational model and programming language is not yet well developed, and (iii) the depth (length of

the longest signal propagation path) of a CMOS reversible circuit is larger than the equivalent irreversible circuit, and consequently makes the reversible CMOS circuit operation slower than irreversible circuit. However, as the heat removal from the current CMOS IC technologies is very difficult, CMOS reversible circuits are promise of future computing technology.

2.8 Ternary Reversible Logic

In this section we introduce elementary and macro-level ternary reversible gates. We then briefly discuss realizations of ternary reversible circuits using TGFSOP expressions.

2.8.1 Ternary Reversible Gates

In a one input ternary reversible function, there are three possible input combinations (0, 1, and 2). Therefore, there are $3! = 6$ possible single-input (or unary) reversible functions as shown in Table 2.13. A unary or single-input gate applies a transform from Table 2.13 on the input value. The symbol of the ternary unary gate is shown in Figure 2.5(a), where x is the input, U is any transform from Table 2.13, that is, $U \in \{+0, +1, +2, 01, 02, 12\}$, and Ux is the transformed output. Muthukrishnan and Stroud [33] showed that the unary transforms $U \in \{+0, +1, +2, 01, 02, 12\}$ can be realized using the linear ion trap model of quantum computing [3]. Thus, the unary gate is an elementary gate and physically realizable. Ternary unary gates are also called ternary shift gates, ternary single-qutrit gates, or ternary single-trit gates in the literature [22, 16].

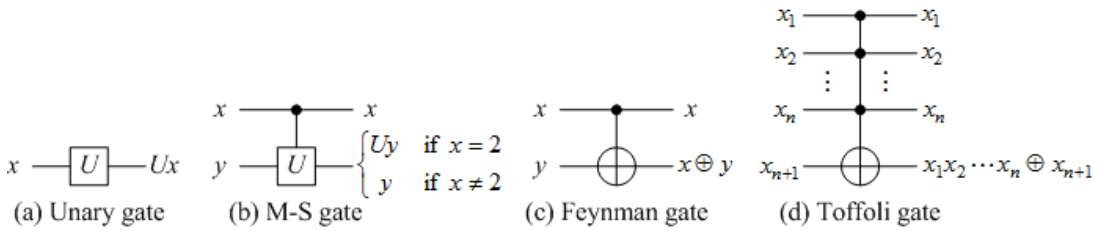

Figure 2.5: Ternary reversible (a) unary, (b) M-S, (c) Feynman, and (d) Toffoli gates.

Table 2.13: Ternary single-input (unary) reversible operations and their corresponding reversible literals.

Variable	Ternary reversible unary operations and corresponding literals					
x	x^{+0}	x^{+1}	x^{+2}	x^{01}	x^{02}	x^{12}
0	0	1	2	1	2	0
1	1	2	0	0	1	2
2	2	0	1	2	0	1

Muthukrishnan and Stroud [33] also showed that an input value can be transformed by any transform $U \in \{+0, +1, +2, 01, 02, 12\}$ if the value of another input is 2 using the linear ion trap model of quantum computing. This led to the implementation of a family of 2-input controlled gates, commonly called Muthukrishnan-Stroud (M-S) gates. The symbol of the ternary M-S gate is shown in Figure 2.5(b). In Figure 2.5(b), the input x is called the control input and is passed unchanged to the output. The input y is called the target input. The transform U is applied on the target input y if and only if the control input is $x = 2$, otherwise the target input y is passed unchanged to the output. The M-S family of ternary gates is the most well presented and widely used set of elementary ternary gates, which are physically realizable using linear ion trap quantum technology.

Readers should note that any operation that can be implemented as a single operation (such as unary transform or controlled unary transform) using quantum technology is considered to be an elementary or primitive operation and their corresponding implementation is referred to as an elementary or primitive quantum gate [34]. Theoretically, any unitary transform [34] can be implemented as an elementary quantum gate [33]. However, in the majority of work on reversible logic synthesis only unary and M-S gates are used as elementary quantum gates [12, 15, 16].

A widely used macro-level ternary gate is the 2-input Feynman gate. The symbol of the ternary Feynman gate is shown in Figure 2.5(c). In Figure 2.5(c), the input x is the control input and is passed unchanged to the output. The input y is the target input and the target

output is the GF3 addition of the inputs x and y, that is, $x \oplus y$. The ternary Feynman gate is a macro-level gate and must be realized using elementary unary and M-S gates [15].

Another widely used macro-level ternary gate is the multiple-controlled Toffoli gate. The symbol of the ternary Toffoli gate with n control inputs $(n > 1)$ is shown in Figure 2.5(d). In Figure 2.5(d), the inputs x_1 to x_n are control inputs and are passed unchanged to the outputs. The input x_{n+1} is the target input and at the output, has the value $x_1x_2 \cdots x_n \oplus x_{n+1}$, where the product $x_1x_2 \cdots x_n$ is the GF3 product. The ternary Toffoli gates are macro-level gates and must be realized using elementary unary and M-S gates [15].

The Feynman gate can be thought of as a Toffoli gate with one control input.

Perkowski *et al.* proposed a family of generalized ternary gates (GTGs) in [35]. The symbol of the GTG gate family is shown in Figure 2.6. In Figure 2.6, the input A is the control input and is passed unchanged to the output. The input B is the target input. The control input A controls a conceptual ternary multiplexer and decides the transform applied on the target input B. If the value of the control input A is 0, 1, or 2, then an x, y, or z transform is applied on the target input B, respectively, where $x, y, z \in \{+0, +1, +2, 01, 02, 12\}$. Depending on the values of x, y, and z, $6^3 = 216$ different GTGs are possible. The GTGs are macro-level gates and must be realized using elementary unary and M-S gates [12].

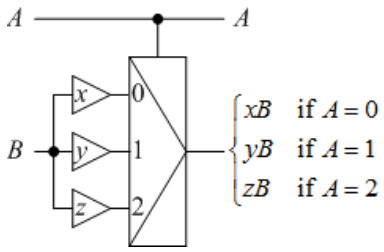

Figure 2.6: The family of generalized ternary gates (GTGs).

2.8.2 Realizations of TGFSOP Expressions Using Feynman and Toffoli Gates

An example TGFSOP expression for a ternary logic function $F(x, y, z)$ is given in (2.6). The implementation of the TGFSOP expression in (2.6) is shown in Figure 2.7. The func-

tion $F(x,y,z)$ is realized along a 0-initialized ancilla input (a constant-initialized input that is not part of the function input but required to realize a reversible circuit is known as ancilla input). The Feynman gate realizes $0 \oplus x = x$ at the target output. The first Toffoli gate realizes $x \oplus yz$ at the target output. Finally the second Toffoli gate realizes $(x \oplus yz) \oplus xyz = x \oplus yz \oplus xyz$ along the target output.

$$F(x,y,x) = x \oplus yz \oplus xyz \tag{2.6}$$

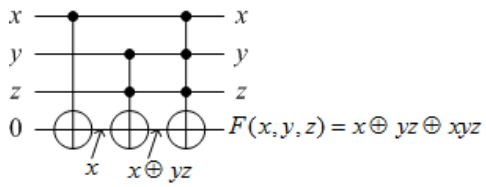

Figure 2.7: Realization of the TGFSOP expression (2.6).

Different applications of TGFSOP expressions use different sets of literals and also different sets of reversible gates. Thus their realization techniques are also different. We will discuss TGFSOP-based synthesis of ternary reversible circits during literature review in Section 3.3.

2.9 Reversible Circuit Realization Complexities

The complexities of reversible circuit realizations are measured using two metrics - quantum cost and number of ancilla inputs. In most of the ternary reversible logic synthesis work, it is assumed that the ternary reversible macro-level gates will be realized using elementary quantum gates such as unary and M-S gates. The total number of elementary quantum gates needed for realizing a ternary reversible circuit is called the quantum cost of that circuit. In most reversible circuit realization techniques the primary inputs are used as control inputs and the functional output is realized along a constant-initialized working input (see Figure 2.7). These constant-initialized working inputs are called ancilla inputs.

In a quantum circuit an increased quantum cost is a result of the implementation requiring more elementary quantum gates. This is likely to produce a larger circuit which may cause the circuit operation slower. On the other hand, an increased number of ancilla inputs increases the circuit width (number of quantum wires). In many quantum technologies, the circuit width is a limitation [19]. Therefore, the goal of synthesis of reversible circuits is to reduce both the quantum cost and the number of ancilla inputs.

Chapter 3

Literature Review

3.1 Introduction

Ternary reversible logic synthesis is a comparatively new research area. Very little work in this area has been reported in the literature. In this chapter we discuss works on realizations of macro-level ternary gates using elementary quantum gates. We then present works on ternary reversible circuit synthesis by sub-categorizing based on the synthesis techniques used.

Specifically, this chapter is organized as follows:

- In Section 3.2 we discuss previous works on realizations of macro-level ternary reversible gates such as generalized ternary gates (GTGs), Feynman gates, and Toffoli gates using ternary unary and M-S gates.

- In Section 3.3 we discuss previous works on synthesis techniques of ternary reversible circuits by sub-categorizing based on the synthesis techniques used.

3.2 Realizations of Macro-Level Ternary Reversible Gates

In ternary reversible logic synthesis techniques, primitive quantum gates such as unary and M-S gates and macro-level gates such as generalized ternary gates (GTGs), Feynman gates, and Toffoli gates are commonly used. Macro-level gates are larger building blocks built using primitive quantum gates and used as gates in the logic level synthesis techniques. As GTGs, Feynman gates, and Toffoli gates are macro-level gates, they must be realized

using unary and M-S gates. In this section, we discuss previous works on realizations of these macro-level gates using unary and M-S gates.

In [12] realization of the family of GTGs using unary and M-S gates is reported, which is reproduced in Figure 3.1. Operation of the circuit in Figure 3.1 can be explained as follows: The control values of the first, middle, and last M-S gates are $A \oplus 2$, $A \oplus 2 \oplus 2 = A \oplus 1$, and $A \oplus 2 \oplus 2 \oplus 2 = A$, respectively. If $A = 0$, then the control values of the three M-S gates are 2, 1, and 0, respectively. So, the first M-S gate applies the x transform on the target input B. If $A = 1$, then the control values of the three M-S gates are 0, 2, and 1, respectively. So, the middle M-S gate applies the y transform on the target input B. If $A = 2$, then the control values of the three M-S gates are 1, 0, and 2, respectively. So, the last M-S gate applies the z transform on the target input B. The resultant unary transform along the control input A is $2 \oplus 2 \oplus 2 = 0$. So, the control input A is restored at the output. Depending on the choice of the transforms x, y, and z, $6^3 = 216$ different generalized ternary gates can be realized. This realization requires a quantum cost of six and no ancilla input.

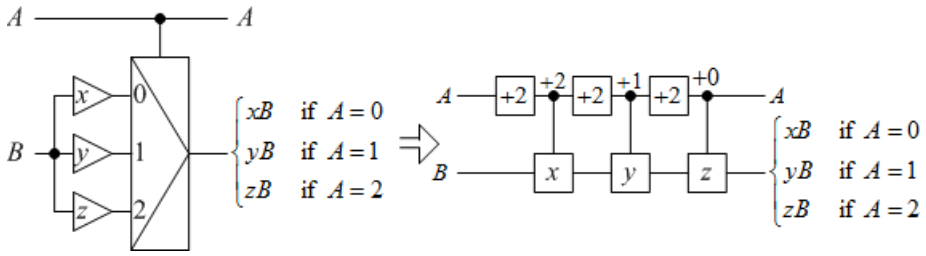

Figure 3.1: Realization of generalized ternary gate (GTG) from [12].

In [12] realization of the ternary Feynman gate using unary and M-S gates is presented. This realization is reproduced in Figure 3.2 and can be explained as follows: The control values of the first and the second M-S gates are x and $x \oplus 1$, respectively. If the value of the control input is $x = 0$, then the control values of the two M-S gates are 0 and 1, respectively. So, both the M-S gates do not apply any transform on the target input y. Thus, the target input y is passed unchanged to the target output and the target output value is $y = 0 \oplus y = x \oplus y$. If $x = 1$, then the control values of the two M-S gates are 1 and 2,

respectively. Thus, a +1 transform is applied on the target input y by the second M-S gate and the target output is $y \oplus 1 = 1 \oplus y = x \oplus y$. If $x = 2$, then the control values of the two M-S gates are 2 and 0, respectively. Thus, a +2 transform is applied on the target input y by the first M-S gate and the target output is $y \oplus 2 = 2 \oplus y = x \oplus y$. The resultant unary transform along the control input x is $1 \oplus 2 = 0$ and the control input x is restored at the output. The quantum cost of this realization is four and it requires no ancilla input.

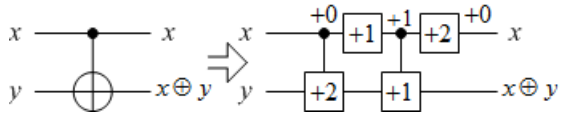

Figure 3.2: Realization of ternary Feynman gate from [12].

In [12] a three-input Toffoli gate is first realized using GTGs and Feynman gates as shown in Figure 3.3(a). The GTGs and the Feynman gates are then replaced by their realizations using unary and M-S gates. Cascaded unary gates are simplified into a single unary gate. The resulting realization using unary and M-S gates is reproduced in Figure 3.3(b). More details of this realization can be found in [12]. The realization requires a quantum cost of 16 and no ancilla input.

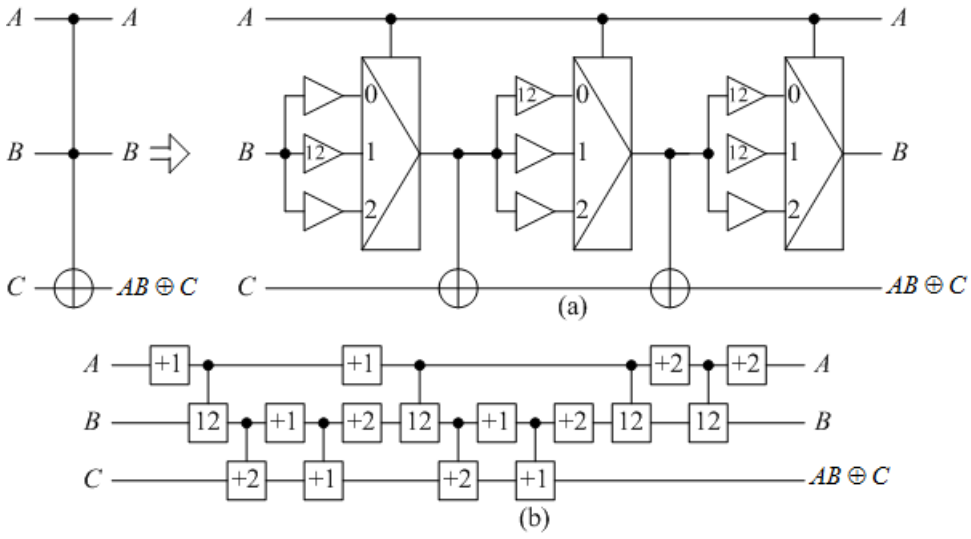

Figure 3.3: Realization of a three-input Tofoli gate from [12]. (a) Realization using GTGs and Feynman gates. (b) Realization after replacing the GTGs and the Feynman gates by their corresponding realizations using unary and M-S gates.

In [15] an architecture for realizing a multiple-valued (or d-valued with $d \geq 3$) Feynman gate using unary and M-S gates is presented. Realization of a ternary ($d = 3$) Feynman gate using that architecture is shown in Figure 3.4. This realization requires a quantum cost of four and no ancilla input. A detailed discussion of this architecture can be found in [15].

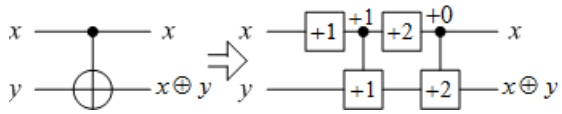

Figure 3.4: Realization of a ternary Feynman gate from [15].

In [15] an architecture for implementing a multiple-valued three-input Toffoli gate using unary and M-S gates is presented. Realization of a ternary three-input Toffoli gate using that architecture is shown in Figure 3.5. A more detailed discussion of this architecture can be found in [15]. The realization of a three-input Toffoli gate requires a quantum cost of 16 and no ancilla input.

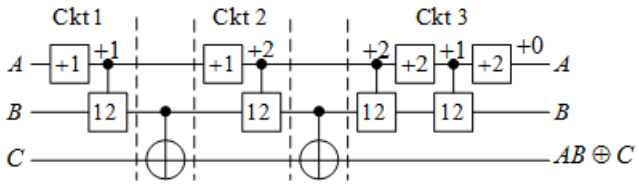

Figure 3.5: Realization of a three-input ternary Toffoli gate from [15].

The design for implementing a three-input ternary Toffoli gate is extended for realization of an n-input $(n > 3)$ ternary Toffoli gate. Interested readers can see [15] for more details of realization of n-input $(n > 3)$ ternary Toffoli gate. The quantum cost of an n-input $(n > 3)$ ternary Toffoli gate realization is $8 + 2\times$ [quantum cost of $(n-1)$-input ternary Toffoli gate] and it does not require any ancilla input.

3.3 Previous Works on Reversible Synthesis of Ternary Logic Circuits

3.3.1 Genetic Algorithm-Based Synthesis of Ternary Reversible Circuits

In [17] a steady-state genetic algorithm (GA) [29] based method is proposed for synthesizing ternary reversible circuits using GTGs. In the proposed method, multi-output ternary

functions are realized as cascades of GTGs. Constant-initialized ancilla inputs are used and the functional outputs are realized along any of the primary inputs or ancilla inputs. The primary input lines and the ancilla input lines are numbered. A GTG is represented by an ordered tuple of control wire number, target wire number, x transform number, y transform number, and z transform number. In the chromosome each gene is a string of digits and represents a GTG of the circuit. Initially for each chromosome multiple ancilla inputs and multiple GTGs are used. When the GA converges, then unused ancilla inputs are eliminated from the resulting circuit. If a GTG has the same line as both control and target, then that GTG is eliminated from the resulting circuit. If transforms of a GTG is $x = y = z = +0$, then that GTG is also eliminated from the resulting circuit. A complex fitness function is used consisting of number of outputs matched with the given functional outputs, number of total lines in the resulting circuit, number of GTGs in the resulting circuit, and total number of non +0 transforms used with the GTGs. Binary tournament selection is used to select two parents. A one-point crossover is used. The mutation operator randomly changes a gene. If the generated offspring is better than the lowest fit chromosome, then the lowest fit chromosome is replaced by the offspring and the population size is kept unchanged. Interested readers can see [17] for more details of the proposed GA.

3.3.2 Max-Min Algebra-Based Synthesis of Ternary Reversible Circuits

In [5] a method for mapping GTG cascades for ternary logic functions using ternary Max-Min algebra is proposed. The proposed ternary Max-Min algebra uses 1-reduced Post literals (see Table 2.5) and their compound forms for representing ternary logic functions using Max-Min expressions. A compound form of 1-reduced Post literal of a variable x is represented as $x^{i,j}$, where $i, j \in \{0,1,2\}$ and $i \neq j$. When $x = i$ or $x = j$, then $x^{i,j} = 1$, otherwise $x^{i,j} = 0$. The compound forms of 1-reduced Post literals are shown in Table 3.1.

The preliminary idea for deriving a Max-Min expression for a ternary logic function is illustrated using $Min(A,B,C)$ function as shown in the ternary K-map of Figure 3.6. The

Table 3.1: Compound Forms of 1-Reduced Post literals.

Variable	Compound Forms of 1-Reduced Post Literals		
x	$x^{0,1}$	$x^{0,2}$	$x^{1,2}$
0	1	1	0
1	1	0	1
2	0	1	1

input combination 222 producing output 2 is represented using the minterm $A^2B^2C^2$. The input combinations 111, 112, 121, 122, 211, 212, and 221 produce output 1. If the input combination 222 is considered a don't care for output 1, then the input combinations 111, 112, 121, 122, 211, 212, 221, and 222 form a group, where $A = 1$ or 2, $B = 1$ or 2, and $C = 1$ or 2. This group is represented using the minterm $A^{1,2}B^{1,2}C^{1,2}$. The total function is represented as shown in (3.1). For the input combination 222 producing output 2, the minterm $A^2B^2C^2 = 1$ and the minterm $A^{1,2}B^{1,2}C^{1,2} = 1$. In that case, $Min(A,B,C,) = 2 \cdot 1 + 1 \cdot 1 = 2$. For the input combination 111 producing output 1, the minterm $A^2B^2C^2 = 0$ and the minterm $A^{1,2}B^{1,2}C^{1,2} = 1$. In that case, $Min(A,B,C,) = 2 \cdot 0 + 1 \cdot 1 = 1$. Similarly, for other input combinations producing output 1, $Min(A,B,C) = 1$. For the input combination 000 producing output 0, the minterm $A^2B^2C^2 = 0$ and the minterm $A^{1,2}B^{1,2}C^{1,2} = 0$. In that case, $Min(A,B,C,) = 2 \cdot 0 + 1 \cdot 0 = 0$. Similarly, for other input combinations producing output 0, $Min(A,B,C) = 0$.

$$Min(A,B,C) = 2A^2B^2C^2 + 1A^{1,2}B^{1,2}C^{1,2} \tag{3.1}$$

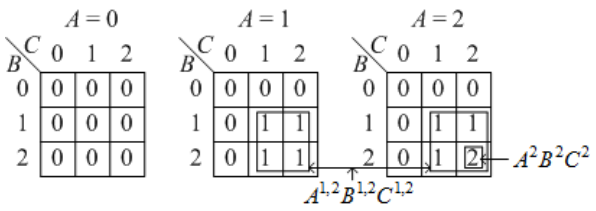

Figure 3.6: K-map representing ternary $Min(A,B,C)$ function.

In [5] the expression of (3.1) is realized using the GTG cascade as shown in Figure 3.7. Interested readers can see [5] for explanations of the operation of the circuit.

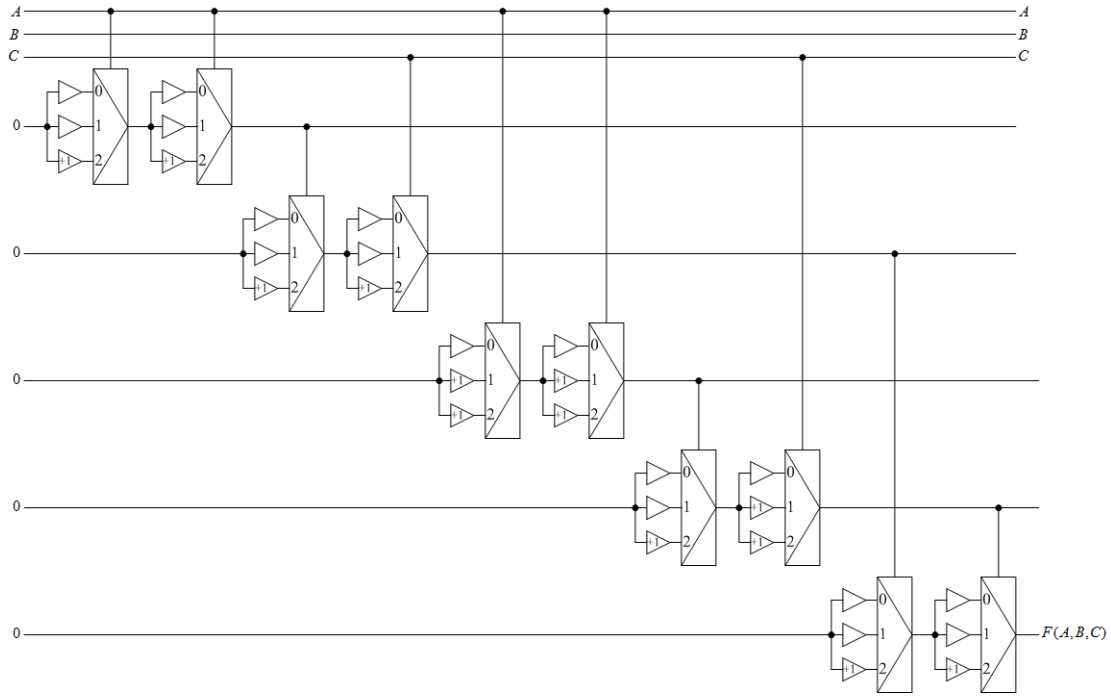

Figure 3.7: GTG cascade for the expression (3.1).

In [5] experimental results of 33 instances of 11 ternary benchmark functions from Appendix A are reported. In these realizations both the number of GTGs and the number of ancilla inputs are high. The quantum costs of the circuit realizations are not reported. As this method requires high numbers of both GTGs and ancilla inputs, it is not considered to be a practical approach for functions with large number of inputs.

3.3.3 Transformation-Based Synthesis of Ternary Reversible Circuits

In [31, 32] a transformation-based synthesis method for ternary reversible functions is presented. This method uses five unary operations $C1$, $C2$, N, D, and E which correspond to +1, +2, 02, 12, and 01 unary operations, respectively. Controlled versions of these gates are also used, where the control value is 2 and there may be one or more control lines. The transformation method starts with the truth table of the reversible function and then transforms are applied so that the function becomes an identity function. In every step of the

transformation one or more gates are added to the circuit. Interested readers can see [31, 32] for details of the transformation-based synthesis technique. The major disadvantage of this method is that it works using the truth table of the reversible function and thus will not be feasible for use with functions with large number of inputs, since the size of the truth table is exponential in the number of input variables.

3.3.4 TGFSOP-Based Synthesis of Ternary Reversible Circuits

In [20] a preliminary approach to TGFSOP-based ternary reversible logic synthesis is presented. In this work TGFSOP expressions suitable for reversible mapping of ternary logic functions were introduced for the first time. This work uses six reversible unary operations (called shift operations) and their corresponding literals as shown in Table 2.13 using different notations. These reversible unary operations are called shift operations and their corresponding unary gates are called shift gates. In this work $xx = x^2$ is also used as an irreversible composite literal. TGFSOP expressions are realized as cascades of unary (shift), swap, Feynman, and Toffoli gates. Experimental results of seven arbitrary multi-output functions expressed as TGFSOP expressions are reported. This work showed a practical method of mapping TGFSOP expressions into ternary reversible circuits. However, no method for expressing ternary logic functions as TGFSOP expressions is discussed.

A followup work is presented in [21], where a method for determining minimized TGFSOP expressions for ternary logic functions is proposed. In this work three sets of literals are used for determining TGFSOP expressions: six reversible literals, 12 irreversible composite literals, and three 1-reduced Post literals. In this work 16 ternary Galois field expansions (TGFEs) are developed. To determine and minimize TGFSOP expression for a given ternary logic function, a Kronecker ternary Galois field decision diagram (KTGFDD) is created using a heuristic algorithm and then the TGFSOP expression is determined from the KTGFDD. Interested readers can see [21] for details of KTGFDD and KTGFDD-based minimization of TGFSOP expressions for ternary logic functions. Experimental results of

50 instances of 12 benchmark functions from Appendix A are reported in terms of number of nodes in the generated KTGFDD and number of TGF products in the resulting TGFSOP expressions. However, mapping of TGFSOP expressions into reversible circuits is not discussed in this work.

The concepts in [20] and [21] are combined in [22]. Ternary logic functions are first represented using heuristically minimized KTGFDDs and then TGFSOP expressions are determined from the KTGFDDs. TGFSOP expressions are then mapped into reversible circuits as cascades of unary (shift), swap, Feynman, and Toffoli gates. Experimental results of 54 instances of 12 benchmark functions from Appendix A are reported in terms of number of nodes in the generated KTGFDD and number of TGF products in the resulting TGFSOP expressions. Experimental results of mapping TGFSOP expressions of these 54 benchmark function instances into reversible circuits are also reported in terms of numbers of copying (Feynman) gates, unary (shift) gates, swap gates, and Toffoli gates. Interested readers can see [22] for more details.

The concepts in [20, 21, 22] are extended in [16]. This method uses three 1-reduced Post literals as shown in Table 2.5 and six reversible literals as shown in Table 2.13 for determining TGFSOP expressions for ternary logic functions. It is shown that if a reversible literal is GF3 multiplied by 2, then the result is also a reversible literal. 55 ternary Galois field expansions (TGFEs) including 16 from [21, 22] are used for minimization of TGFSOP expressions. Applications of 55 TGFEs on a variable of a ternary logic function produce 55 different TGFSOP expressions. The minimum TGFSOP expression among these 55 possible TGFSOP expressions leads towards the minimization of TGFSOP expression. For each variable of the given ternary logic function one TGFE is selected so that applications of those TGFEs on all variables produce the minimized TGFSOP expression. A quantum-inspired evolutionary algorithm (QEA) [11] is used for selecting a suitable TGFE for each variable. The function is represented using its output vector. The don't care outputs are assumed to be 0. The TGFEs are applied on the output vector. The minimized TGF-

SOP expression is generated from the final transformed vector . The minimized TGFSOP expressions for the sum output $S(A,B,C)$ and the carry output $C_o(A,B,C)$ of a ternary full-adder are given in (3.2) and (3.3), respectively. Readers can see [16] for details of the proposed QEA, application of transforms on the output vector, and generating the TGFSOP expression from the transformed output vector.

$$S(A,B,C) = A^{02}C^{+1}C^{02} \oplus B^{02}C^{+1}C^{01} \oplus C^{+1} \tag{3.2}$$

$$C_o(A,B,C) = A^{0}B^{12}B^{+2}C^{1} \oplus ABC^{1} \oplus A^{12}BB^{+2}C^{+1} \oplus A^{12}A^{+2}B^{+2}B^{+2}C^{1} \oplus A^{12}A^{+2}BC^{+1} \tag{3.3}$$

In [16] a method for mapping TGFSOP expressions into reversible circuits as cascades of unary, M-S, Feynman, and Toffoli gates is presented. Using that mapping technique the reversible realizations of TGFSOP expressions (3.2) and (3.3) are shown in Figure 3.8. Interested readers can see [16] for more details of mapping TGFSOP expressions into reversible circuits.

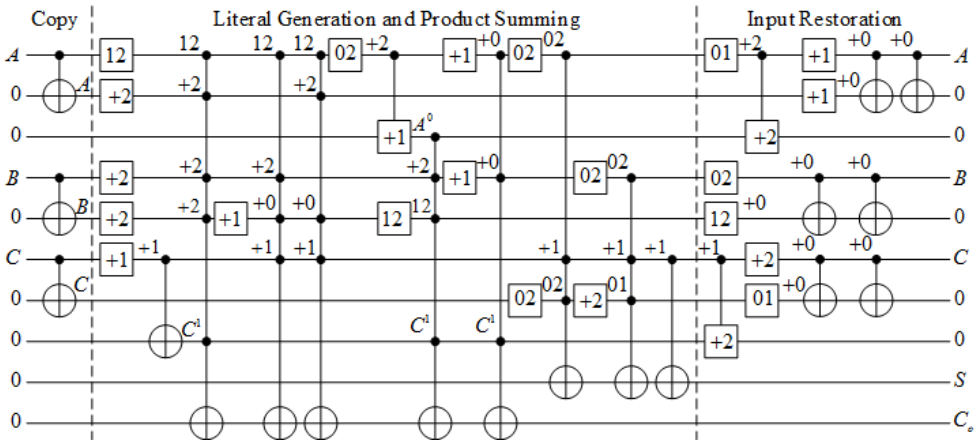

Figure 3.8: Reversible realization of a ternary full-adder using the TGFSOP expressions (3.2) and (3.3) using a cascade of unary, M-S, Feynman, and Toffoli gates from [16].

In [16] experimental results of a ternary half-adder (ha), half-subtractor (hs), full-adder (fa), and full-subtractor (fs) are reported as number of products in the canonical TGFSOP expression, number of products in the minimized TGFSOP expression, population size,

and generation required in the QEA. Quantum cost and the number of ancilla inputs are not reported. However, from [15] it can be noted that the quantum costs of Toffoli gates are very high. Therefore, the quantum costs of the cascades of Toffoli gates generated from TGFSOP expressions will also be very high. Another drawback of this approach is that suitable TGFEs for the variables are determined among 55 TGFEs using the QEA, which is usually a very time consuming process. However, the most important advantage of this approach is that a ternary logic function with a large number of inputs can be expressed as a TGFSOP expression and the TGFSOP expression can be mapped into a reversible circuit. Thus this method is an efficient technique for ternary reversible logic synthesis in comparison to other methods proposed in the literature.

3.3.5 Projection Operation-Based Synthesis of Ternary Reversible Circuits

In [25] a projection operation-based method is presented for synthesis of ternary reversible circuits. Six projection operations are introduced. These projection operations are basically three 2-reduced Post operations and three 1-reduced Post operations as shown in Tables 2.5 and 2.6, respectively. Implementations of the projection gates using GTGs are discussed. The projection gates may have more than one input. Implementations of the multiple-input projection gates using GTGs are also discussed. A ternary logic function is represented as a disjunction of minterms. The minterms are formed using projection literals. The product operation of the minterms is the Min operation. The disjunction operation is the Max operation. Ten simplification rules are proposed for minimization of the disjunction of minterms expressions. This work presented synthesis of a ternary half-adder and full-adder. However in these implementations the projection operations are manipulated in a way to express the sum outputs of the half-adder and the full-adder as $a \oplus b$ and $a \oplus b \oplus c_{in}$, respectively. These expressions make it possible to implement the half-adder and the full-adder using Feynman gates with few ancilla inputs and gate counts. These manipulations are function specific and, therefore, cannot be considered to be a general design

technique; rather they are customized implementations.

3.3.6 Group Theory-Based Synthesis of Ternary Reversible Circuits

In [24] a method for synthesizing a ternary logic circuit using a truth table is presented. This method transforms the irreversible truth table of the given function into a reversible truth table by adding constant-initialized ancilla inputs and garbage outputs (outputs that are not used as the output of the given function but are needed to make the circuit reversible are called garbage outputs). Using an algorithm based on group theory [8], the reversible truth table is mapped to a reversible circuit using ternary swap gates, NOT gates (three NOT operations are defined in this work), and Toffoli gates (a Toffoli gate is proposed in this work, where a 1 is GF3 added to the target input if and only if all the control inputs are 2). Experimental results for a ternary half-adder and full-adder are reported. The main claim of the work is that it reduces the ancilla inputs in comparison to the work in [25]. From Table V in [24], it can be seen that for the ternary half-adder, this method reduces ancilla inputs from 2 to 1, but increases the total gate count from 4 to 122 in comparison to the results in [25]. Similarly, for the ternary full-adder, this method reduces the ancilla inputs from 4 to 1, but increases the total gate count from 8 to more than 150. In addition, because this method works with the truth table of the given function, it cannot be used for functions with a large number of inputs.

Chapter 4

Ternary Logic Function Representation Using Max-Min Algebra

4.1 Introduction

In this thesis we propose a ternary Max-Min algebra for representing ternary logic functions for reversible realizations. Generic definition of ternary Max-Min algebra and representations of ternary logic functions as Max-Min expressions are discussed in Section 2.4. Different applications from the literature have used different set of literals to form Max-Min expressions. For example, in [10] 2-reduced Post literals are used and in [5] 1-reduced Post literals are used to form Max-MIn expressions. In this chapter we discuss our proposed ternary Max-Min algebra with different sets of literals and then discuss our proposed method of representing ternary logic functions as Max-Min expressions. In Chapter 6 we will propose reversible realizations of Max-Min expressions using multiple-controlled unary gates proposed in Chapter 5.

This chapter is organized as follows:

- In Section 4.2 we discuss two sets of literals that we use to form ternary Max-Min expressions for representing ternary logic functions for reversible realizations.
- In Section 4.3 we discuss our proposed method of representing ternary logic functions as Max-Min expressions .

4.2 Literals of the Proposed Ternary Max-Min Algebra

For reversible realizations of ternary logic functions, we use two types of unary operators (also called literal operators), which change the logic value of a variable and represent different literals of the variable.

The first type of operators that we primarily use for representing a ternary logic function as a Max-Min expression is the reversible operators [16]. There are $3! = 6$ ternary reversible operations as shown in Table 4.1. These operations result in six reversible literals as shown in the table. Like a variable, these reversible literals satisfy all the properties of ternary Max-Min algebra stated in Section 2.4. In this thesis we use only the reversible literals x^{+0}, x^{+1}, and x^{+2}, which satisfy the following property:

$$x^{+0} + x^{+1} + x^{+2} = 2 \tag{4.1}$$

Table 4.1: Ternary unary reversible operations and their corresponding literals.

Variable	Unary reversible operations and corresponding literals					
x	x^{+0}	x^{+1}	x^{+2}	x^{01}	x^{02}	x^{12}
0	0	1	2	1	2	0
1	1	2	0	0	1	2
2	2	0	1	2	0	1

The second type of unary operators we use are three non-reversible operators and their corresponding literals as shown in Table 4.2. These operations give the Max of two unary reversible operations and satisfy all the properties of ternary Max-Min algebra stated in Section 2.4. These non-reversible literals are used to minimize Max-Min expressions so that the quantum cost of the resulting reversible circuit is reduced (see Chapter 7).

Table 4.2: Ternary unary non-reversible operations and their corresponding literals.

Variable	Unary non-reversible operations and corresponding literals		
x	$x^{(0+1)} = x^{+0} + x^{+1}$	$x^{(0+2)} = x^{+0} + x^{+2}$	$x^{(1+2)} = x^{+1} + x^{+2}$
0	1	2	2
1	2	1	2
2	2	2	1

4.3 Ternary Logic Function Representation using Max-Min Algebra

4.3.1 Sub-Functions of a Ternary Logic Function

A ternary logic function F produces either 0, 1, or 2 as output. Let $F0$, $F1$, and $F2$ be three ternary-input binary-output sub-functions producing output 0, 1, and 2, respectively. In the proposed method of representing ternary logic function using Max-Min algebra, we split a ternary logic function F into three sub-functions $F0$, $F1$, and $F2$. In each sub-function the corresponding outputs are represented by 1, the don't care outputs remain unchanged, and the other two output values are represented by 0. An example two-variable ternary logic function and its three sub-functions are shown in the truth table in Table 4.3.

Table 4.3: An example two-variable ternary logic function and its three sub-functions.

AB	$F(A,B)$	$F0(A,B)$	$F1(A,B)$	$F2(A,B)$
00	0	1	0	0
01	1	0	1	0
02	x	x	x	x
10	1	0	1	0
11	0	1	0	0
12	x	x	x	x
20	2	0	0	1
21	1	0	1	0
22	x	x	x	x

4.3.2 Canonical Max-Min Expression for a Sub-Function

In our proposed method of representing a ternary logic function using Max-Min algebra the sub-functions $F0$, $F1$, and $F2$ are separately represented by three canonical Max-Min expressions comprising of canonical minterms of input combinations.

An input combination of a sub-function is represented by a canonical minterm (literals combined using the Min operation), where the input value is assigned to a reversible literal as shown in Table 4.4. Consider the sub-function $F1(A,B)$ in Table 4.3. The input combination 01 producing output 1 is represented by the canonical minterm $A^{+2}B^{+1}$; the input combination 10 producing output 1 is represented by the canonical minterm $A^{+1}B^{+2}$; and the input combination 21 producing output 1 is represented by the canonical minterm $A^{+0}B^{+1}$. The reversible literals are chosen such that the corresponding input value is changed to 2 and the value of the canonical minterm becomes 2. The choice of the logic value of a minterm to be 2 for the corresponding input combination will be evident during reversible circuit mapping of the Max-Min expression in Chapter 6, and is related to the gate implementation discussed in Chapter 5. Thus the sub-function $F1(A,B)$ is represented as in (4.2).

$$F1(A,B) = A^{+2}B^{+1} + A^{+1}B^{+2} + A^{+0}B^{+1} \tag{4.2}$$

Table 4.4: Reversible literal assignment to input values for determining the canonical minterm for an input combination.

Input	Literal
0	x^{+2}
1	x^{+1}
2	x^{+0}

For the input combination 21 the first minterm $A^{+2}B^{+1} = 1 \cdot 2 = 1$; the second minterm $A^{+1}B^{+2} = 0 \cdot 0 = 0$; and the third minterm $A^{+0}B^{+1} = 2 \cdot 2 = 2$. Thus $F1(A,B) = 1+0+2 = 2$. Similarly it can be shown that for the input combinations 01 and 10 the functional output of the sub-function $F1(A,B)$ is 2.

Three properties of this representation method are important to note:

- The minterm for the input combination 01 is $A^{+2}B^{+1}$. For the input combination 01 the minterm $A^{+2}B^{+1}$ becomes $2 \cdot 2 = 2$; for the input combination 00 the minterm $A^{+2}B^{+1}$ becomes $2 \cdot 1 = 1$; and for the input combination 02 the minterm $A^{+2}B^{+1}$ becomes $2 \cdot 0 = 0$. Therefore, *a minterm becomes 2 for its corresponding input combination and becomes 0 or 1 for other input combinations*.

- Consider the Max-Min expression of (4.2). For the input combination 01 the first minterm $A^{+2}B^{+1}$ becomes 2; the second minterm $A^{+1}B^{+2}$ becomes 0; and the third minterm $A^{+0}B^{+1}$ becomes 0. Thus $F1(A,B) = 2+0+0 = 2$. For the input combination 10 the first minterm $A^{+2}B^{+1}$ becomes 0; the second minterm $A^{+1}B^{+2}$ becomes 2; and the third minterm $A^{+0}B^{+1}$ becomes 1. Thus $F1(A,B) = 0+2+1 = 2$. For the input combination 21 the first minterm $A^{+2}B^{+1}$ becomes 1; the second minterm $A^{+1}B^{+2}$ becomes 0; and the third minterm $A^{+0}B^{+1}$ becomes 2. Thus $F1(A,B) = 1+0+2 = 2$. In general, the canonical Max-Min expressions for the sub-functions Fi ($i \in \{0,1,2\}$) is formed by determining the canonical minterms corresponding to the input combinations producing output i and then combining the canonical minterms by Max operations resulting in a canonical Max-Min expression for Fi. For an input combination producing an output i, the minterm for Fi corresponding to the given input combination becomes 2 and the other minterms of Fi become either 0 or 1. Thus, their Max operation produces a 2. Therefore, *a Max-Min expression of a sub-function becomes 2 for its corresponding input combinations and becomes 0 or 1 for other input combinations*.

- For the input combinations 01, 10, and 21 the sub-function $F1(A,B)$ of (4.2) becomes 2 as discussed earlier. For the input combination 00 the functional output is $F1(A,B) = 2 \cdot 1 + 1 \cdot 2 + 0 \cdot 1 = 1+1+0 = 1$ and for the input combination 22 the functional output is $F1(A,B) = 1 \cdot 0 + 0 \cdot 1 + 2 \cdot 0 = 0+0+0 = 0$. Thus for the in-

put combinations 01, 10, and 21 the sub-function $F1(A,B)$ becomes 2 and for other input combinations the sub-function $F1(A,B)$ becomes either 0 or 1. In general, *for an input combination producing output i the sub-function Fi ($i \in \{0,1,2\}$) becomes 2 and for an input combination producing an output other than i the sub-function Fi becomes either 0 or 1.*

The canonical Max-Min expressions for the sub-functions $F0(A,B)$ and $F2(A,B)$ from Table 4.3 are given in (4.3) and (4.4), respectively.

$$F0(A,B) = A^{+2}B^{+2} + A^{+1}B^{+1} \tag{4.3}$$

$$F2(A,B) = A^{+0}B^{+2} \tag{4.4}$$

In Chapter 7 we will see that the canonical Max-Min expression can be minimized to smaller Max-Min expressions using composite literals.

In Chapter 6 we will see that a reversible ternary circuit corresponding to a ternary logic function F can be synthesized using any two of the three sub-functions $F0$, $F1$, and $F2$.

Chapter 5

Ternary Multiple-Controlled Unary Gates

5.1 Introduction

In Chapter 4 we proposed a ternary Max-Min algebra with two sets of literals and demonstrated how to represent ternary logic functions as Max-Min expressions. In Chapter 6 we will discuss reversible realizations of Max-Min expressions using our proposed multiple-controlled unary gates. In this chapter we introduce our proposed macro-level ternary multiple-controlled unary gates. We also show their realizations using elementary quantum gates. This will allow us to compare quantum costs of our realizations with that of the previous works.

This chapter is organized as follows:

- In Section 5.2 we discuss ternary elementary quantum gates such as ternary unary gates and M-S gates.
- In Section 5.3 we propose ternary single-controlled unary gates and discuss their realizations using unary and M-S gates.
- In Section 5.4 we propose ternary multiple-controlled unary gates and discuss their realizations using unary and M-S gates.

5.2 Ternary Elementary Quantum Gates

Widely used ternary reversible gates are discussed in Section 2.8. Two elementary ternary quantum gates are briefly reintroduced here.

The symbol of the non-controlled ternary unary gate is shown in Figure 5.1(a), where x is the input, $U \in \{+1, +2, 01, 02, 12\}$ is a transform from Table 2.13, and Ux is the transformed output.

The symbol of the ternary M-S gate is shown in Figure 5.1(b). The input x is called the control input and is passed unchanged to the output. The input y is called the target input. The transform $U \in \{+1, +2, 01, 02, 12\}$ is applied on the target input y if and only if the control input is $x = 2$, otherwise the target input y is passed unchanged to the output. The M-S gate is a single-controlled unary gate.

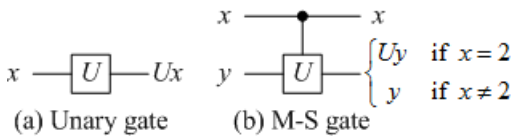

Figure 5.1: Ternary (a) unary and (b) M-S gates.

In this thesis we use only +1 and +2 transforms for U. It should be noted that the addition operation associated with +1 and +2 transforms are GF3 addition [22, 16].

5.3 Ternary Single-Controlled Unary Gates

5.3.1 Ternary Single-Controlled Unary Gates with Simple Control

In a single-controlled unary gate with simple control there is one control input and the control value is either 0, 1, or 2. Symbols and quantum-level realizations of three single-controlled unary gates are shown in Figure 5.2. In Figure 5.2(a), when $x = 0$, then the control value of the M-S gate becomes 0 + 2 (GF3) = 2 and the U transform is applied on the target input y. For other values of x, the control value of the M-S gate is not 2 and the target input y is passed unchanged. The input value of x is restored at the output, since $x + 2 + 1$ (GF3) $= x$. In Figure 5.2(b), when $x = 1$, then the control value of the M-S

gate becomes 1 + 1 (GF3) = 2 and the U transform is applied on the target input y. For other values of x, the control value of the M-S gate is not 2 and the target input y is passed unchanged. The input value of x is restored at the output, since $x+1+2$ (GF3) $= x$. In Figure 5.2(c), when $x = 2$, then the control value of the M-S gate is 2 and the U transform is applied on the target input y. For other values of x, the control value of the M-S gate is not 2 and the target input y is passed unchanged. Readers should note that the single-controlled unary gate with control value 2 of Figure 5.2(c) is actually the M-S gate. The quantum costs of single-controlled unary gates with control values 0 and 1 is three and that of control value 2 is one. These realizations do not require any ancilla input.

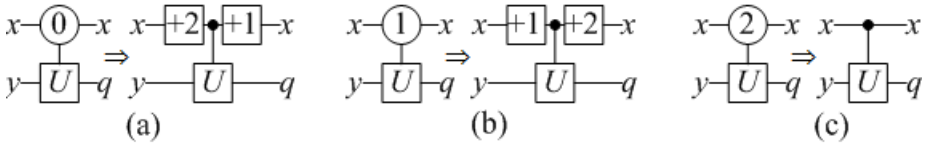

Figure 5.2: Ternary single-controlled unary gates with simple control.

5.3.2 Ternary Single-Controlled Unary Gates with Composite Control

In a single-controlled unary gate with composite control there is one control input and the control value is either 01, 02, or 12. For a composite control value, the single-controlled unary gate becomes active when either of the control values appears at the control point. Symbols and quantum-level realizations of three single-controlled unary gates with composite control values 01, 02, and 12 are shown in Figure 5.3(a), 5.3(b), and 5.3(c), respectively. In Figure 5.3(a), when $x = 0$, then the control values of the first and the second M-S gates are 0 + 2 (GF3) = 2 and 0 + 2 + 2 (GF3) = 1, respectively and the first M-S gate applies the U transform on the target input y. When $x = 1$, then the control values of the first and the second M-S gates are 1 + 2 (GF3) = 0 and 1 + 2 + 2 (GF3) = 2, respectively and the second M-S gate applies the U transform on the target input y. When $x = 2$, then the control values of the first and the second M-S gates are 2 + 2 (GF3) = 1 and 2 + 2 + 2 (GF3) = 0, respectively and both the M-S gates are inactive passing the target input y unchanged to

the target output. The control input is restored at the output, since $x+2+2+2$ (GF3) $= x$. In Figure 5.3(b), when $x = 0$, then the control values of the first and the second M-S gates are 0 and 0 + 2 (GF3) = 2, respectively and the second M-S gate applies the U transform on the target input y. When $x = 2$, then the control values of the first and the second M-S gates are 2 and 2 + 2 (GF3) = 1, respectively and the first M-S gate applies the U transform on the target input y. When $x = 1$, then the control values of the first and the second M-S gates are 1 and 1 + 2 (GF3) = 0, respectively and both the M-S gates are inactive passing the target input y unchanged to the target output. The control input is restored at the output, since $x+2+1$ (GF3) $= x$. In Figure 5.3(c), when $x = 1$, then the control values of the first and the second M-S gates are 1 and 1 + 1 (GF3) = 2, respectively and the second M-S gate applies the U transform on the target input y. When $x = 2$, then the control values of the first and the second M-S gates are 2 and 2 + 1 (GF3) = 0, respectively and the first M-S gate applies the U transform on the target input y. When $x = 0$, then the control values of the first and the second M-S gates are 0 and 0 + 1 (GF3) = 1, respectively and both the M-S gates are inactive passing the target input y unchanged to the target output. The control input is restored at the output, since $x+1+2$ (GF3) $= x$. The quantum costs of single-controlled unary gates with composite controls 01, 02, and 12 are five, four, and four, respectively. These realizations do not require any ancilla input.

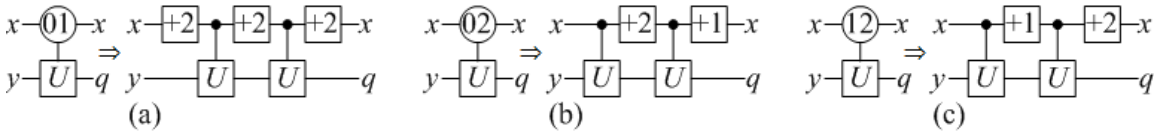

Figure 5.3: Ternary single-controlled unary gates with composite control.

5.3.3 Realization Complexities of Ternary Single-Controlled Unary Gates

The quantum costs and the number of ancilla inputs for realizations of ternary single-controlled unary gates are summarized in Table 5.1.

Table 5.1: Quantum costs and number of ancilla inputs for ternary single-controlled unary gate realizations.

Control Value	Quantum Cost	Ancilla Input
0	3	0
1	3	0
2	1	0
01	5	0
02	4	0
12	4	0

5.4 Ternary Multiple-Controlled Unary Gates

5.4.1 Ternary Multiple-Controlled Unary Gates with Simple Controls

In a multiple-controlled unary gate with simple controls there are two or more control inputs and the control values are either 0, 1, or 2. Symbol and quantum-level realization of an example triple-controlled unary gate with control values 0, 1, and 2, respectively, are shown in Figure 5.4. The principle of this realization is stated below:

1. The value of the top two control input lines are made 2 using unary gates. If the control value is 0, then a unary gate with +2 transform is used to make the value to be 0 + 2 (GF3) = 2. If the control value is 1, then a unary gate with +1 transform is used to make the value to be 1 + 1 (GF3) = 2. If the control value is 2, then no unary gate is used.

2. The transformed values of the top two control input lines are used as control values of two M-S gates with +1 transform. These two M-S gates make the value of the top 0-initialized ancilla input line to be 0 + 1 + 1 (GF3) = 2.

3. The value of the third control input line is made 2 using unary gate with appropriate transform. The transformed values of the top ancilla input line and the third control input line are used as control values of two M-S gates with +1 transform. These two M-S gates make the value of the second 0-initialized ancilla input line to be 0 + 1 +

1 (GF3) = 2.

4. For each successive input control line one 0-initialized ancilla input is added and the value of that ancilla input line is made 2 using two M-S gates with +1 transform, whose control lines are the control input line and the previous 0-initialized ancilla input line.

5. An M-S gate with $U \in \{+1, +2\}$ transform and the last ancilla input line as the control is used to realize the U transform.

6. The control input lines and the 0-initialized ancilla input lines are restored in the Input Restore part using inverse gates of the gates in the Control Implementation part. It should be noted that +1 and +2 transforms are the inverse of each other, since +1 + 2 (GF3) = 0.

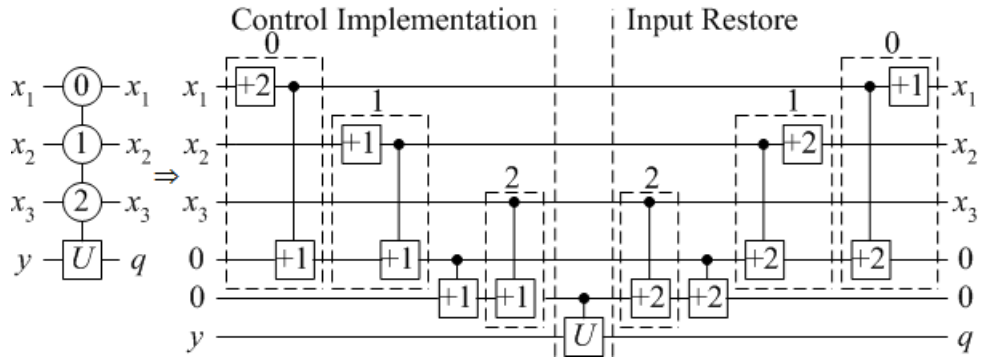

Figure 5.4: An example ternary triple-controlled unary gate with simple controls and its quantum-level realization.

The operation of the circuit in Figure 5.4 can be explained as follows. When $x_1 = 0$, then the control value of the first M-S gate is 0 + 2 (GF3) = 2 along the x_1 input line. When $x_2 = 1$, then the control value of the second M-S gate is 1 + 1 (GF3) = 2 along the x_2 input line. The first and the second M-S gates make the control value of the third M-S gate 0 + 1 + 1 (GF3) = 2 along the top 0-initialized ancilla input line. When $x_3 = 2$, then the control value of the fourth M-S gate is 2 along the x_3 input line. The third and the fourth M-S gates then make the control value of the fifth M-S gate 0 + 1 + 1 (GF3) =

2 along the bottom 0-initialized ancilla input line. As the fifth M-S gate becomes active, the U transform is applied on the target input y. For all other values of x_1, x_2, and x_3 the control value of the fifth M-S gate is not 2 and the U transform is not applied on the target input y. The Input Restore part restores the control inputs and the ancilla input constants at the output. Thus the circuit realizes a triple-controlled unary gate with control values 0, 1, and 2, respectively. This realization requires a quantum cost of 13 and two ancilla inputs. Using this technique any multiple-controlled unary gate with simple control values can be realized.

5.4.2 Ternary Multiple-Controlled Unary Gates with Composite Controls

In a multiple-controlled unary gate with composite controls there are two or more control inputs and the control values are either 01, 02, or 12. Symbol and quantum-level realization of an example triple-controlled unary gate with control values 01, 02, and 12, respectively, are shown in Figure 5.5. The principle of this realization is stated below:

1. The value of the top two control input lines are made 2 using unary gates. If the control value is 01, then two unary gates with +2 transform are used. When the control input value is 0, then the output of the first unary gate becomes 0 + 2 (GF3) = 2. When the control input value is 1, then the output of the second unary gate becomes 1 + 2 + 2 (GF3) = 2. If the control value is 02, then a unary gate with +2 transform is used. When the control input value is 2, then it is directly used. When the control input value is 0, then the output of the unary gate is 0 + 2 (GF3) = 2. If the control value is 12, then a unary gate with +1 transform is used. When the control input value is 2, then it is directly used. When the control input value is 1, then the output of the unary gate is 1 + 1 (GF3) = 2.

2. The two transformed values of the top control line are used as control values of two M-S gates with +1 transform. Either of them is active depending on the transformed values and apply a +1 transform on the top 0-initialized ancilla input. Similarly the

two transformed values of the second control line are used as control values of two M-S gates with +1 transform. Either of them is active depending on the transformed values and apply a +1 transform on the top 0-initialized ancilla input. Thus when both the two top control input lines match the control values, then the value of the top 0-initialized ancilla input line becomes 0 + 1 + 1 (GF3) = 2.

3. The top 0-initialized ancilla input line is used as the control line of a M-S gate with +1 transform, whose target line is the second 0-initialized ancilla input. The value of the third control input line is made 2 using necessary unary gate. The transformed values of the third control input line are used as controls of two M-S gates with +1 transform, whose target line is the second ancilla input line. Thus when the value of the top three control input lines match the control values, then the value of the second ancilla input line becomes 0 + 1 + 1 (GF3) = 2.

4. For each successive control input line one 0-initialized ancilla input is added and the value of that ancilla input line is made 2 using three M-S gates with +1 transform. The control line of the first M-S gate is connected to the previous ancilla input. The control lines of the other two M-S gates are connected to the transformed values of the control input.

5. An M-S gate with $U \in \{+1, +2\}$ transform and the last ancilla input line as the control is used to realize the U transform.

6. The control input lines and the 0-initialized ancilla input lines are restored in the Input Restore part using inverse gates of the gates in the Control Implementation part.

The operation of the circuit in Figure 5.5 can be explained as follows. When $x_1 = 0$ or $x_1 = 1$, then the control value of either the first or the second M-S gate is 2 along the x_1 input line. When $x_2 = 0$ or $x_2 = 2$, then the control value of either the third or the fourth M-S gate is 2 along the x_2 input line. When x_1 and x_2 input lines match their corresponding control

values, then the control value of the fifth M-S gate becomes 2 along the top 0-initialized ancilla input line. When $x_3 = 1$ or $x_3 = 2$, then the control value of either the sixth or the seventh M-S gate is 2 along the x_3 input line. When x_1, x_2, and x_3 input lines match their corresponding control values, then the control value of the eighth M-S gate becomes 2 along the bottom 0-initialized ancilla input line. As the eighth M-S gate becomes active, the U transform is applied on the target input y. For all other values of x_1, x_2, and x_3 the control value of the eighth M-S gate is not 2 and the U transform is not applied on the target input y. The Input Restore part restores the control inputs and the ancilla input constants at the output. Thus the circuit realizes a triple-controlled unary gate with control values 01, 02, and 12, respectively. This realization requires a quantum cost of 23 and two ancilla inputs. Using this technique any multiple-controlled unary gate with composite control values can be realized.

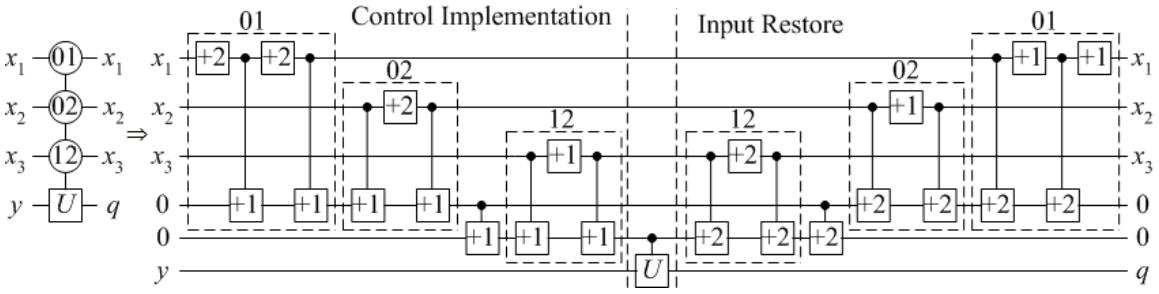

Figure 5.5: An example ternary triple-controlled unary gate with composite controls and its quantum-level realization.

5.4.3 Ternary Multiple-Controlled Unary Gates with Mixed Controls

A ternary multiple-controlled unary gate with mixed controls, that is, controls from the set $\{0, 1, 2, 01, 02, 12\}$ can be realized using the concepts discussed above. The symbol and the quantum-level realization of an example ternary triple-controlled unary gate with controls 01, 1, and 12, respectively, is shown in Figure 5.6. The operation of the circuit can be explained in a similar manner as done for the circuits of Figures 5.4 and 5.5. The realization requires a quantum cost of 21 and two ancilla inputs.

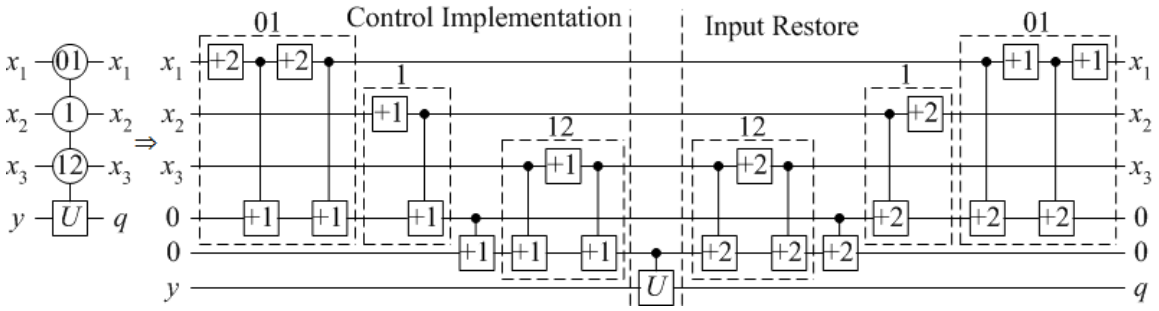

Figure 5.6: An example ternary triple-controlled unary gate with mixed controls and its quantum-level realization.

5.4.4 Realization Complexities of Multiple-Controlled Unary Gates

From the discussions of Figures 5.4 and 5.5, we see that the first and the second control input lines together require one 0-initialized ancilla input and each of the remaining control input lines requires one 0-initialized ancilla input. Thus the total number of ancilla inputs required to realize a multiple-controlled unary gate with n control input lines is

$$\text{Ancilla Input} = n - 1. \tag{5.1}$$

From Figure 5.4 we see that a 0 control requires two elementary gates in the Control Implementation part and two elementary gates in the Input Restore part totaling four elementary gates; a 1 control requires two elementary gates in the Control Implementation part and two elementary gates in the Input Restore part totaling four elementary gates; and a 2 control requires one elementary gate in the Control Implementation part and one elementary gate in the Input Restore part totaling two elementary gates. From Figure 5.5 we see that a 01 control requires four elementary gates in the Control Implementation part and four elementary gates in the Input Restore part totaling eight elementary gates; a 02 control requires three elementary gates in the Control Implementation part and three elementary gates in the Input Restore part totaling six elementary gates; and a 12 control requires three elementary gates in the Control Implementation part and three elementary gates in the Input Restore part totaling six elementary gates. From Figures 5.4 and 5.5 we

see that each 0-initialized ancilla input but the last one requires one elementary gate in the Control Implementation part and one elementary gate in the Input Restore part totaling two elementary gates. That is, each of the $((n-1)-1) = (n-2)$ ancilla inputs requires two elementary gates. Another elementary gate is required to implement the U transform. Thus the quantum cost for realizing a multiple-controlled unary gate with n control input lines is

$$\text{Quantum Cost} = n_0 \times 4 + n_1 \times 4 + n_2 \times 2 + n_{01} \times 8 + n_{02} \times 6 + n_{12} \times 6 + (n-2) \times 2 + 1, \tag{5.2}$$

where, n_0, n_1, n_2, n_{01}, n_{02}, and n_{12} are the number of control positions of 0, 1, 2, 01, 02, and 12, respectively and $n > 1$ is the number of control inputs.

Chapter 6

Ternary Reversible Circuit Synthesis Using Max-Min Algebra

6.1 Introduction

In Chapter 4 we proposed a ternary Max-Min algebra with two new sets of literals and showed the method of representing ternary logic functions as Max-Min expressions. In Chapter 5 we proposed ternary multiple-controlled unary gates for reversible realizations of ternary Max-Min expressions. In this chapter we propose a method for reversible realizations of ternary Max-Min expressions using ternary multiple-controlled unary gates.

This chapter is organized as follows:

- In Section 6.2 we discuss reversible circuit realizations from ternary Max-Min expressions using ternary multiple-controlled unary gates.
- In Section 6.3 we discuss architectures for reversible realizations of ternary logic functions.
- In Section 6.4 we show one example of reversible realization of a ternary logic function expressed as ternary Max-Min expressions using ternary multiple-controlled unary gates.
- In Section 6.5 we discuss post synthesis quatum cost reduction after quantum-level expansions of reversible circuits of ternary multiple-controlled unary gates.

6.2 Mapping of Ternary Max-Min Expressions into Reversible Circuits

Ternary logic function representation using ternary Max-Min algebra is discussed in Section 4.3. Each of the sub-functions $F0$, $F1$, and $F2$ is represented as canonical Max-Min expression. An example two-variable ternary logic function and its three sub-functions are shown in Table 4.3, which are reproduced here in Table 6.1. The canonical Max-Min expressions of $F0(A,B)$, $F1(A,B)$, and $F2(A,B)$ are given in equations (4.3), (4.2), and (4.4), respectively, which are reproduced here in equations (6.1), (6.2), and (6.3).

Table 6.1: An example two-variable ternary logic function and its three sub-functions.

AB	$F(A,B)$	$F0(A,B)$	$F1(A,B)$	$F2(A,B)$
00	0	1	0	0
01	1	0	1	0
02	x	x	x	x
10	1	0	1	0
11	0	1	0	0
12	x	x	x	x
20	2	0	0	1
21	1	0	1	0
22	x	x	x	x

$$F0(A,B) = A^{+2}B^{+2} + A^{+1}B^{+1} \tag{6.1}$$

$$F1(A,B) = A^{+2}B^{+1} + A^{+1}B^{+2} + A^{+0}B^{+1} \tag{6.2}$$

$$F2(A,B) = A^{+0}B^{+2} \tag{6.3}$$

Each sub-function is realized as reversible circuit using multiple-controlled unary gates separately. Reversible realization of a ternary Max-Min expression using ternary multiple-controlled unary gates is done as follows:

1. Realize each minterm using a ternary multiple-controlled unary gate. Map a literal

into a control value using the mapping shown in Table 6.2.

2. Place all the multiple-controlled unary gates in cascade putting the control positions along the corresponding input variable lines and the target position along a constant-initialized ancilla input. Choose the ancilla constant and the target transform of all multiple-controlled unary gates as follows:

   (a) For the sub-function $F0$ choose ancilla constant 1 and transform +2 so that 1 + 2 (GF3) = 0 or ancilla constant 2 and transform +1 so that 2 + 1 (GF3) = 0.

   (b) For the sub-function $F1$ choose ancilla constant 0 and transform +1 so that 0 + 1 (GF3) = 1 or ancilla constant 2 and transform +2 so that 2 + 2 (GF3) = 1.

   (c) For the sub-function $F2$ choose ancilla constant 0 and transform +2 so that 0 + 2 (GF3) = 2 or ancilla constant 1 and transform +1 so that 1 + 1 (GF3) = 2.

Table 6.2: Literal to control value mapping for realization of a minterm using a ternary multiple-controlled unary gate.

Literal	Control Value
x^{+0}	2
x^{+1}	1
x^{+2}	0
$x^{(0+1)}$	12
$x^{(0+2)}$	02
$x^{(1+2)}$	01

Reversible realizations of canonical Max-Min expressions of (6.1), (6.2), and (6.3) are shown in Figures 6.1(a), 6.1(b), and 6.1(c), respectively. Each canonical mintrem is realized using a ternary multiple-controlled unary gate, where each reversible literal is mapped to a control value as shown in Table 6.2. All multiple-controlled unary gates are then placed in cascade with constant-initialized ancilla input as target input line of all gates.

The circuit of Figure 6.1(a) can be explained as follows: The first minterm of (6.1) is $A^{+2}B^{+2}$. So the control value of the variable A is 0 and the control value of the variable B

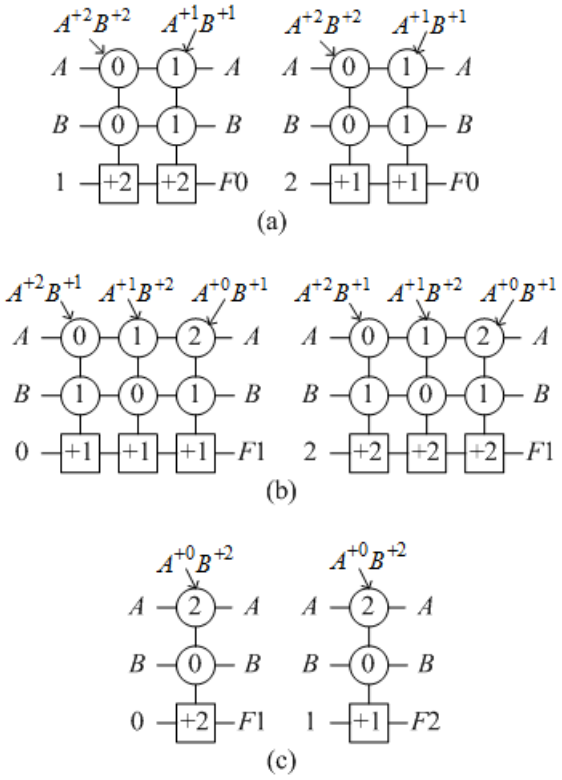

Figure 6.1: Reversible realizations of canonical Max-Min expressions from (a) equation (6.1), (b) equation (6.2), and (c) equation (6.3).

is 0. Therefore, the two control values of the first multiple-controlled unary gate are 0 and 0, respectively. The second minterm is $A^{+1}B^{+1}$. So the control value of the variable A is 1 and the control value of the variable B is 1. Therefore, the two control values of the second multiple-controlled unary gate are 1 and 1, respectively. These two multiple-controlled unary gates are cascaded. In the first circuit of Figure 6.1(a) the ancilla input constant is 1 and the target transforms of the multiple-controlled unary gates are +2. When the input value is $AB = 00$, then the first multiple-controlled unary gate is active and the second multiple-controlled unary gate is inactive. Therefore, the output is $F0 = 1 + 2$ (GF3) $= 0$. When the input value is $AB = 11$, then the first multiple-controlled unary gate is inactive and the second multiple-controlled unary gate is active. Therefore, the output is $F0 = 1 + 2$ (GF3) $= 0$. When the input value is other than $AB = 00$ or $AB = 11$, then both the multiple-controlled unary gates are inactive and the output is $F0 = 1$ (other than 0). Thus the sub-function $F0$ is realized (see Table 6.1). In the second circuit of Figure 6.1(a) the

ancilla input constant is 2 and the transforms of the multiple-controlled unary gates are +1. Operation of this circuit can be explained similarly. Reversible realizations of canonical Max-Min expressions (6.2) and (6.3) by the circuits of Figure 6.1(b) and Figure 6.1(c), respectively, can be explained in a similar manner.

The quantum costs of the circuits of Figures 6.1(a), 6.1(b), and 6.1(c) are calculated using formula (5.2) and are 18, 25, and 7, respectively.

In Chapter 7 we will discuss ternary K-map-based minimization of Max-Min expressions. The canonical Max-Min expressions for the sub-functions $F0(A,B)$ and $F2(A,B)$ in Table 6.1 cannot be minimized as smaller Max-Min expressions. However, the canonical Max-Min expression for the sub-function $F1(A,B)$ in Table 6.1 can be minimized as smaller Max-Min expressions, which is shown in (6.4) (see Chapter 7). Reversible realization of the minimized Max-Min expression of (6.4) is shown in Figure 6.2. The quantum cost of the circuit of Figure 6.2 is calculated using the quantum cost data in Table 5.1 and formula (5.2) and is 15. In comparison, the realization of the canonical Max-Min expression (6.2) representing the same sub-function requires a quantum cost of 25.

$$F1(A,B) = A^{+1} + B^{+1} + A^{+1}B^{+1} \tag{6.4}$$

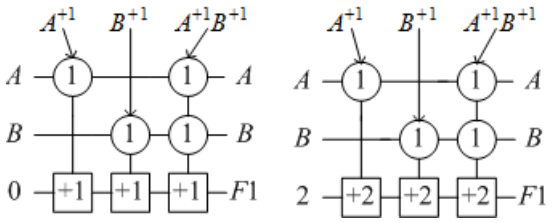

Figure 6.2: Reversible realization of minimized Max-Min expression (6.4).

6.3 Architectures of Ternary Reversible Circuit Synthesis

We propose the architectures of Figure 6.3 for reversible realizations of ternary logic functions. Let $QC0$, $QC1$, and $QC2$ are quantum costs of reversible realizations of sub-

functions $F0$, $F1$, and $F2$, respectively, of a ternary logic function F. The rules of selecting the architecture are as follows:

1. If $QC0 = max(QC0, QC1, QC2)$, then the architecture of Figure 6.3(a) is used. In this architecture the ancilla input is 0. If any input combination produces output 0, then the multiple-controlled unary gates of circuits of $F1$ and $F2$ do not apply any transform on the ancilla input constant 0. Thus the output is $F = 0$. If any input combination produces output 1, then one of the multiple-controlled unary gates of circuit of $F1$ applies a +1 transform on the ancilla input constant 0 and the multiple-controlled unary gates of circuit of $F2$ do not apply any transform on the ancilla input constant 0. Thus the output is $F = 0 + 1$ (GF3) $= 1$. If any input combination produces output 2, then the multiple-controlled unary gates of circuit of $F1$ do not apply any transform on the ancilla input constant 0 and one of the multiple-controlled unary gates of circuit of $F2$ applies a +2 transform on the ancilla input constant 0. Thus the output is $F = 0 + 2$ (GF3) $= 2$. Thus the function F is realized.

2. If $QC1 = max(QC0, QC1, QC2)$, then the architecture of Figure 6.3(b) is used. In this architecture the ancilla input is 1. If any input combination produces output 0, then one of the multiple-controlled unary gates of circuit of $F0$ applies a +2 transform on the ancilla input constant 1 and the multiple-controlled unary gates of circuit of $F2$ do not apply any transform on the ancilla input constant 1. Thus the output is $F = 1 + 2$ (GF3) $= 0$. If any input combination produces output 1, then the multiple-controlled unary gates of circuits of $F0$ and $F2$ do not apply any transform on the ancilla input constant 1. Thus the output is $F = 1$. If any input combination produces output 2, then the multiple-controlled unary gates of circuit of $F0$ do not apply any transform on the ancilla input constant 1 and one of the multiple-controlled unary gates of circuit of $F2$ applies a +1 transform on the ancilla input constant 1. Thus the output is $F = 1 + 1$ (GF3) $= 2$. Thus the function F is realized.

3. If $QC2 = max(QC0, QC1, QC2)$, then the architecture of Figure 6.3(c) is used. In this architecture the ancilla input is 2. If any input combination produces output 0, then one of the multiple-controlled unary gates of circuit of $F0$ applies a +1 transform on the ancilla input constant 2 and the multiple-controlled unary gates of circuit of $F1$ do not apply any transform on the ancilla input constant 2. Thus the output is $F = 2+1$ (GF3) $= 0$. If any input combination produces output 1, then the multiple-controlled unary gates of circuits of $F0$ do not apply any transform on the ancilla input constant 2 and one of the multiple-controlled unary gates of circuit of $F1$ applies a +2 transform on the ancilla input constant 2. Thus the output is $F = 2+2$ (GF3) $= 1$. If any input combination produces output 2, then the multiple-controlled unary gates of circuits of $F0$ and $F1$ do not apply any transform on the ancilla input constant 2. Thus the output is $F = 2$. Thus the function F is realized.

4. If there is a tie between any two of $QC0$, $QC1$, and $QC2$, then the tie is broken by selecting any one of the two corresponding architectures. If there is a tie among all three of $QC0$, $QC1$, and $QC2$, then the tie is broken by selecting any one of the three architectures.

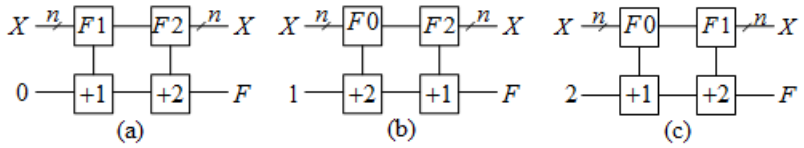

Figure 6.3: Architectures of reversible realizations of ternary logic functions.

The architecture of Figure 6.3 is selected in such a way that two sub-functions with lower quantum costs are implemented and the sub-function with the maximum quantum cost is not realized, instead the functional output of the sub-function is used as the ancilla constant in the realization process. This technique reduces the quantum costs of the realized circuits.

6.4 Ternary Reversible Circuit Synthesis Example

The quantum cost of realization of Max-Min expression (6.1) for the sub-function $F0(A,B)$ is $QC0 = 18$. The quantum cost of realization of Max-Min expression (6.4) for the sub-function $F1(A,B)$ is $QC1 = 15$. The quantum cost of realization of Max-Min expression (6.3) for the sub-function $F2(A,B)$ is $QC2 = 7$. From the values of $QC0$, $QC1$, and $QC2$, we see that $QC0 = max(QC0, QC1, QC2)$. Thus the architecture of Figure 6.3(a) is used for reversible realization of the function $F(A,B)$ in Table 6.1. The reversible realization of $F(A,B)$ is shown in Figure 6.4. The circuit of Figure 6.4 requires a quantum cost of 22 and two ancilla inputs (one ancilla input required in multiple-controlled unary gate realization and another ancilla input required for function realization).

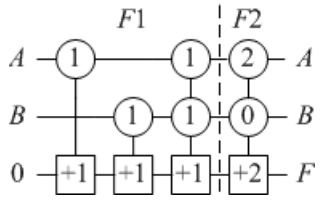

Figure 6.4: Reversible realization of the function $F(A,B)$ in Table 6.1 represented by Max-Min expressions of sub-functions $F0(A,B)$, $F1(A,B)$, and $F2(A,B)$ of (6.1), (6.4), and (6.3), respectively, using the architecture of Figure 6.3(a).

6.5 Post Synthesis Quantum Cost Reduction

Quantum-level expansion of the circuit in Figure 6.4 is shown in Figure 6.5. In the first and the third multiple-controlled unary gates, the control values along the input variable A are 1 and 1, respectively, and they are adjacent. The transform of the unary gate in the Input Restore part of the first multiple-controlled unary gate is +2. The transform of the unary gate in the Control Implementation part of the third multiple-controlled unary gate is +1. These two unary gates are adjacent along the input variable line A and they can be omitted, since 2 + 1 (GF3) = 0. In the second and the third multiple-controlled unary gates, the control values along the input variable B are 1 and 1, respectively, and they are adjacent. The transform of the unary gate in the Input Restore part of the second multiple-controlled

unary gate is +2. The transform of the unary gate in the Control Implementation part of the third multiple-controlled unary gate is +1. These two unary gates are adjacent along the input variable line B and they can be omitted, since $2+1$ (GF3) $= 0$. In the third and the fourth multiple-controlled unary gates, the control values along the input variable B are 1 and 0, respectively, and they are adjacent. The transform of the unary gate in the Input Restore part of the third multiple-controlled unary gate is +2. The transform of the unary gate in the Control Implementation part of the fourth multiple-controlled unary gate is +2. These two unary gates can be replaced by a single unary gate with transform +1, since 2 + 2 (GF3) = 1. Thus five unary gates in Figure 6.5 can be eliminated resulting in quantum cost of $22-5=17$.

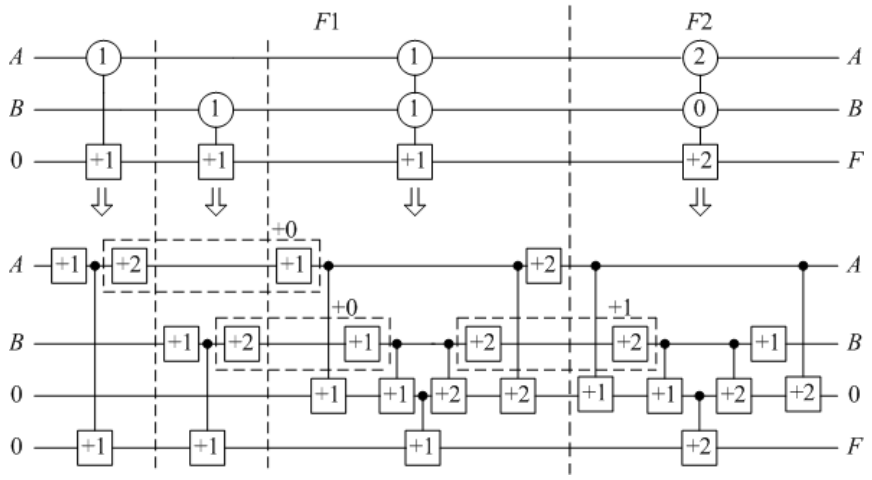

Figure 6.5: Quantum-level expansion of circuit of Figure 6.4 for post synthesis quantum cost reduction.

We have exhaustively determined all possible adjacent pair of control values and their quantum cost reduction as listed in Table 6.3.

Table 6.3: Post synthesis quantum cost reduction.

Adjacent Controls	Quantum Cost Reduction
0, 0	2
0, 1	1
0, 01	2
1, 0	1
1, 1	2
1, 01	1
01, 0	1
01, 1	2
01, 01	1
02, 0	2
02, 1	1
02, 01	2
12, 0	1
12, 1	2
12, 01	1

Chapter 7

Ternary K-map-Based Minimization of Ternary Max-Min Expressions

7.1 Introduction

In Chapter 4 we proposed ternary canonical Max-Min expressions for reversible circuit realizations of ternary logic functions. We devoted Chapters 5 and 6 to discuss our proposed method for quantum-level realizations of Max-Min expressions. In this chapter we discuss our proposed ternary K-map-based minimization of ternary Max-Min expressions. Since the goal of our ternary K-map-based minimization technique is to reduce quantum costs of the resulting reversible circuits, the concept of our proposed method for reversible realizations of Max-Min expressions is essential for understanding our proposed minimization method. We introduced the structures of ternary K-maps in Section 2.6. In the proposed ternary K-map-based minimization method, each of the three sub-functions $F0$, $F1$, and $F2$ of a ternary logic function F is minimized separately.

This chapter is organized as follows:

- In Section 7.2 we discuss the motivation of minimization of ternary Max-Min expressions to reduce the quantum costs of the resulting reversible circuits.
- In Section 7.3 we discuss grouping of adjacent cells on a ternary K-map for minimization of Max-Min expressions.
- In Section 7.4 we discuss minimization of ternary sub-functions and expressing them as minimized Max-Min expressions using ternary K-map-based method.

- In Section 7.5 we show several examples of ternary K-map-based minimization.
- In Section 7.6 we discuss similarities between the K-map-based ESOP (Chapter 2) and the proposed ternary K-map-based Max-Min expression minimizations.

7.2 Motivation of Minimization of Ternary Max-Min Expressions

Consider the Max-Min expression for an example sub-function $F1(A,B)$ shown in (7.1). The reversible realization of the Max-Min expression (7.1) is shown in Figure 7.1(a), which requires a quantum cost of 16 and two ancilla inputs (one for gate realization and another for target function realization). The Max-Min expression (7.1) can be rewritten as in (7.2) (see Table 4.2). The reversible realization of the Max-Min expression (7.2) is shown in Figure 7.1(b), which requires a quantum cost of 11 and two ancilla inputs (one for gate realization and another for target function realization). From Table 4.2, Max-Min expressions (7.1) and (7.2), and Figure 7.1, we see that if a variable varies over two values (0 and 1; or 0 and 2; or 1 and 2), then two minterms of a canonical Max-Min expression can be replaced by a single minterm and in the resulting reversible circuit realization two multiple-controlled unary gates with simple-control values can be replaced by a single multiple-controlled unary gate with composite control value resulting into a significant quantum cost reduction.

$$F1(A,B) = A^{+1}B^{+0} + A^{+1}B^{+2} \tag{7.1}$$

$$F1(A,B) = A^{+1}B^{+0} + A^{+1}B^{+2} = A^{+1}(B^{+0} + B^{+2}) = A^{+1}B^{(0+2)} \tag{7.2}$$

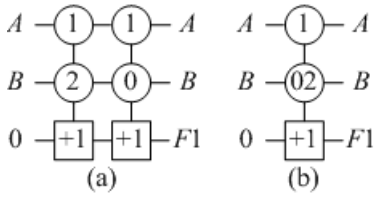

Figure 7.1: Reversible realizations of Max-Min expressions for (a) equation (7.1) and (b) equation (7.2).

Consider the Max-Min expression for an example sub-function $F1(A,B)$ shown in (7.3). The reversible realization of the Max-Min expression (7.3) is shown in Figure 7.2(a), which requires a quantum cost of 25 and two ancilla inputs (one for gate realization and another for target function implementation). The Max-Min expression (7.3) can be rewritten as in (7.4). The reversible realization of the Max-Min expression (7.4) is shown in Figure 7.2(b), which requires a quantum cost of only 3 and one ancilla input (for target function realization; a single controlled unary gate does not require any ancilla input for gate realization). From the property of reversible literals that $x^{+0}+x^{+1}+x^{+2}=2$ (see equation (4.1)), Max-Min expressions (7.3) and (7.4), and Figure 7.2, we see that if a variable varies over 0, 1, and 2, then three minterms of a canonical Max-Min expression can be replaced by a single minterm where the varying variable is missing and in the resulting reversible circuit realization three multiple-controlled unary gates with simple-control values can be replaced by a single multiple-controlled unary gate with one less control points resulting into a significant reduction of both quantum cost and ancilla inputs.

$$F1(A,B)=A^{+2}B^{+0}+A^{+2}B^{+1}+A^{+2}B^{+2} \tag{7.3}$$

$$F1(A,B)=A^{+2}B^{+0}+A^{+2}B^{+1}+A^{+2}B^{+2}=A^{+2}(B^{+0}+B^{+1}+B^{+2})=A^{+2}\cdot 2=A^{+2} \tag{7.4}$$

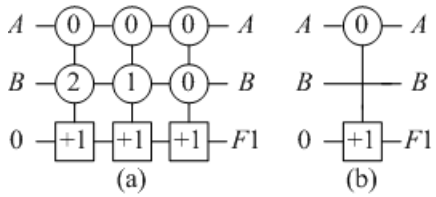

Figure 7.2: Reversible realizations of Max-Min expressions for (a) equation (7.3) and (b) equation (7.4).

We use the ternary K-map method introduced in sections 7.3 and 7.4 to identify the above two cases of variation of values of a variable and simplify the Max-Min expression from the ternary K-map. The ternary K-maps are discussed in section 2.6.

7.3 Grouping of Cells on a Ternary K-Map

7.3.1 Sizes of Groups

The sizes of groups of cells on a ternary K-map are as follows:

1. If a cell with a 1 is isolated (cannot be grouped with other adjacent cells), then that cell forms a group of one-cell, which represents a canonical minterm. This group size can be represented as 2^m or 3^m, where $m = 0$.

2. If two adjacent cells with 1s vary in 1-variable, then the two cells can be grouped. A non-reversible literal from Table 4.2 is used for the varying variable. Using the product rule of counting, the possible sizes of groups are as follows:

   1-variable variation:

   Group of 2 cells

   2-variable variation:

   Group of $2 \times 2 = 4$ cells

   3-variable variation:

   Group of $2 \times 2 \times 2 = 8$ cells

   4-variable variation:

   Group of $2 \times 2 \times 2 \times 2 = 16$ cells

   These group sizes can be represented as 2^m, where $0 < m \leq 4$.

3. If three adjacent cells with 1s vary in one variable, then the three cells can be grouped and the varying variable will be missing. Using the product rule of counting, the possible sizes of groups are as follows:

   1-variable variation:

   Group of 3 cells

   2-variable variation:

   Group of $3 \times 3 = 9$ cells

   3-variable variation:

Group of $3 \times 3 \times 3 = 27$ cells

4-variable variation:

Group of $3 \times 3 \times 3 \times 3 = 81$ cells

These group sizes can be represented as 3^m, where $0 < m \leq 4$.

4. Two adjacent cells with 1s may vary in one variable and three adjacent cells with 1s may vary in another variable forming a group of $2 \times 3 = 6$ adjacent cells. The variable varying in two adjacent cells appear as non-reversible literal in the minterm and the variable varying in three adjacent cells does not appear in the minterm. Using the product rule of counting, the possible sizes of groups are as follows:

   2-variable variation:

   Group of $2 \times 3 = 3 \times 2 = 6$ cells

   3-variable variation:

   Group of $2 \times 2 \times 3 = 2 \times 3 \times 2 = 3 \times 2 \times 2 = 12$ cells

   Group of $2 \times 3 \times 3 = 3 \times 2 \times 3 = 3 \times 3 \times 2 = 18$ cells

   4-variable variation:

   Group of $2 \times 2 \times 2 \times 3 = 2 \times 2 \times 3 \times 2 = 2 \times 3 \times 2 \times 2 = 3 \times 2 \times 2 \times 2 = 24$ cells

   Group of $2 \times 2 \times 3 \times 3 = 2 \times 3 \times 2 \times 3 = 2 \times 3 \times 3 \times 2 = 3 \times 2 \times 2 \times 3 = 3 \times 2 \times 3 \times 2 = 3 \times 3 \times 2 \times 2 = 36$ cells

   Group of $2 \times 3 \times 3 \times 3 = 3 \times 2 \times 3 \times 3 = 3 \times 3 \times 2 \times 3 = 3 \times 3 \times 3 \times 2 = 54$ cells

   These group sizes can be represented as $2^i \times 3^j$, where $i, j \geq 1$ and $0 < (i + j) \leq 4$.

   Let us assume that in a four-variable ternary logic sub-function $F1(A,B,C,D)$ the variable A varies in two values (0 and 1), the variable B varies in three values (0, 1, and 2), the variable C varies in two values (0 and 1), and the variable D varies in three values (0, 1, and 2), then the associated cells of the function form a group of $2 \times 3 \times 2 \times 3 = 36$ cells as shown in Figure 2.3(c).

7.3.2 Grouping Rules

Two grouping rules are stated below with explanations:

1. Consider an example two-variable sub-function $F1(x,y)$ represented on the ternary K-map of Figure 7.3(a). Please note that a blank cell contains a 0. In Figure 7.3(a) the cell 11 containing a 1 is covered twice, once by Group 1 and once by Group 2. For the input combination 11 both the groups will be active and the functional output will be $1+1$ (GF3) $=2$, which is incorrect. Also consider another example four-variable sub-function $F1(w,x,y,z)$ represented on the ternary K-map of Figure 7.3(b), where the cell 0011 containing a 1 is covered four times by Groups 1, 2, 3, and 4. For the input combination 0011 all the four groups will be active and the functional output will be $1+1+1+1$ (GF3) $=(1+1+1)+1$ (GF3) $=0+1$ (GF3) $=1$, which is correct. As $1+1+1$ GF(3) $=0$, to produce a correct output *a cell with a 1 must be covered by multiple of three plus one groups*.

2. Consider an example two-variable sub-function $F1(x,y)$ represented on the ternary K-map of Figure 7.4(a). In this ternary K-map 0s are explicitly shown. When a cell containing a 0 is included within a group of 1s, then that cell behaves like a cell containing a 1. In Figure 7.4(a) the cell 11 containing a 0 is covered two times by Groups 1 and 2. For the input combination 11 both the groups will be active and the functional output will be $1+1$ (GF3) $=2$, which is incorrect. Also consider another example two-variable sub-function $F1(x,y)$ represented on the ternary K-map of Figure 7.4(b), where the cell 11 containing a 0 is covered three times by Groups 1, 2, and 3. For the input combination 11 all the three groups will be active and the functional output will be $1+1+1$ (GF3) $=0$, which is correct. As $1+1+1$ (GF3) $=0$, *a cell containing a 0 can be included in a group of 1s with the restriction that it should be covered by a multiple of three groups*. This type of inclusions of 0s within groups of 1s will help reduce the Max-Min expression but the functional output will not be changed.

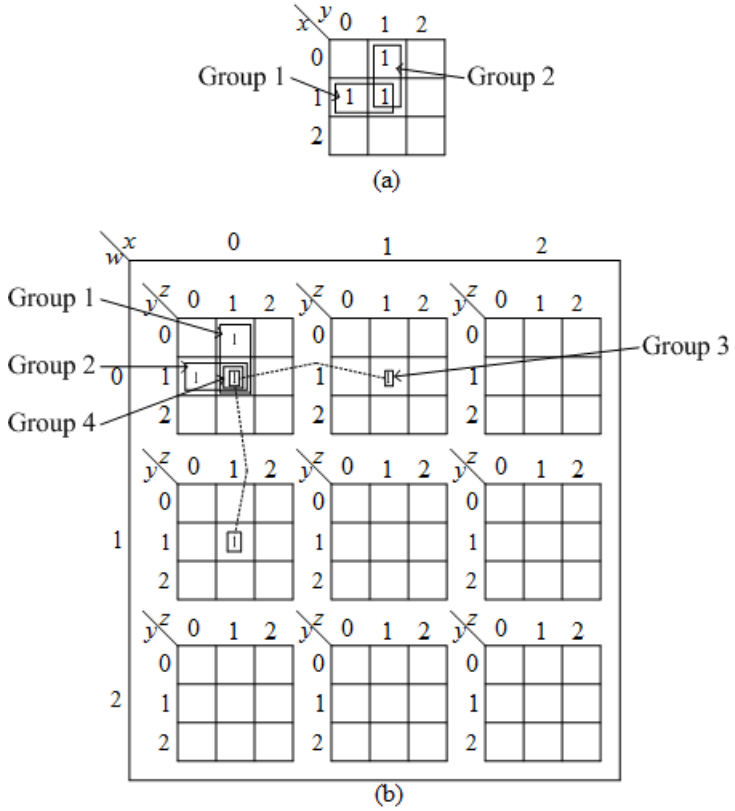

Figure 7.3: (a) A cell with a 1 is overlapped by two groups and (b) a cell with a 1 is overlapped by four groups on a ternary K-map.

7.4 Minimization of Ternary Max-Min Expression of a Ternary Sub-Function Using Ternary K-Map

7.4.1 Grouping and Determining Max-Min Expression

An example two-variable ternary logic function $F(A,B)$ and its three sub-functions $F0(A,B)$, $F1(A,B)$, and $F2(A,B)$ are shown in Table 7.1. The sub-function $F1(A,B)$ is represented on the ternary K-maps in Figure 7.5, where four possible groupings of cells are shown.

In Figure 7.5(a), cells corresponding to input combinations 10 and 11 are grouped as a group of two-cells. In this group $A = 1$ and B varies over 0 and 1. The minterm corresponding to this group is $A^{+1}B^{(1+2)}$. Cells corresponding to input combinations 01 and 11 are grouped as a group of two-cells. In this group A varies over 0 and 1; and $B = 1$. The minterm corresponding to this group is $A^{(1+2)}B^{+1}$. Cells corresponding to input combina-

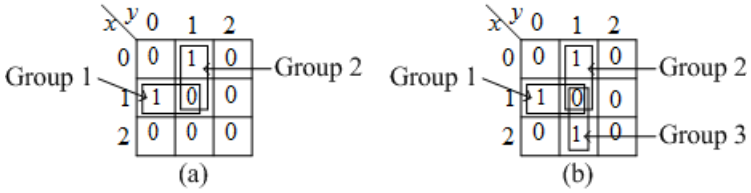

Figure 7.4: (a) A cell with a 0 is overlapped by two groups and (b) a cell with a 0 is overlapped by three groups on a ternary K-map.

Table 7.1: An example two-variable ternary logic function and its three sub-functions.

AB	$F(A,B)$	$F0(A,B)$	$F1(A,B)$	$F2(A,B)$
00	0	1	0	0
01	1	0	1	0
02	x	x	x	x
10	1	0	1	0
11	0	1	0	0
12	x	x	x	x
20	2	0	0	1
21	1	0	1	0
22	x	x	x	x

tions 11 and 21 are grouped as a group of two-cells. In this group A varies over 1 and 2; and $B = 1$. The minterm corresponding to this group is $A^{(0+1)}B^{+1}$. The cell corresponding to the input combination 11 is a cell with a 0 and it is grouped thrice in all three groups satisfying the grouping rule. The Max-Min expression from this K-map is given in (7.5).

$$F1(A,B) = A^{+1}B^{(1+2)} + A^{(1+2)}B^{+1} + A^{(0+1)}B^{+1} \tag{7.5}$$

In Figure 7.5(b), cells corresponding to input combinations 01 and 02 are grouped as a group of two-cells. In this group $A = 0$ and B varies over 1 and 2. The minterm corresponding to this group is $A^{+2}B^{(0+1)}$. Cells corresponding to input combinations 10 and 12 are grouped as a group of two-cells. In this group $A = 1$ and B varies over 0 and 2. The minterm corresponding to this group is $A^{+1}B^{(0+2)}$. Cells corresponding to input combinations 21 and 22 are grouped as a group of two-cells. In this group $A = 2$ and B varies over 1

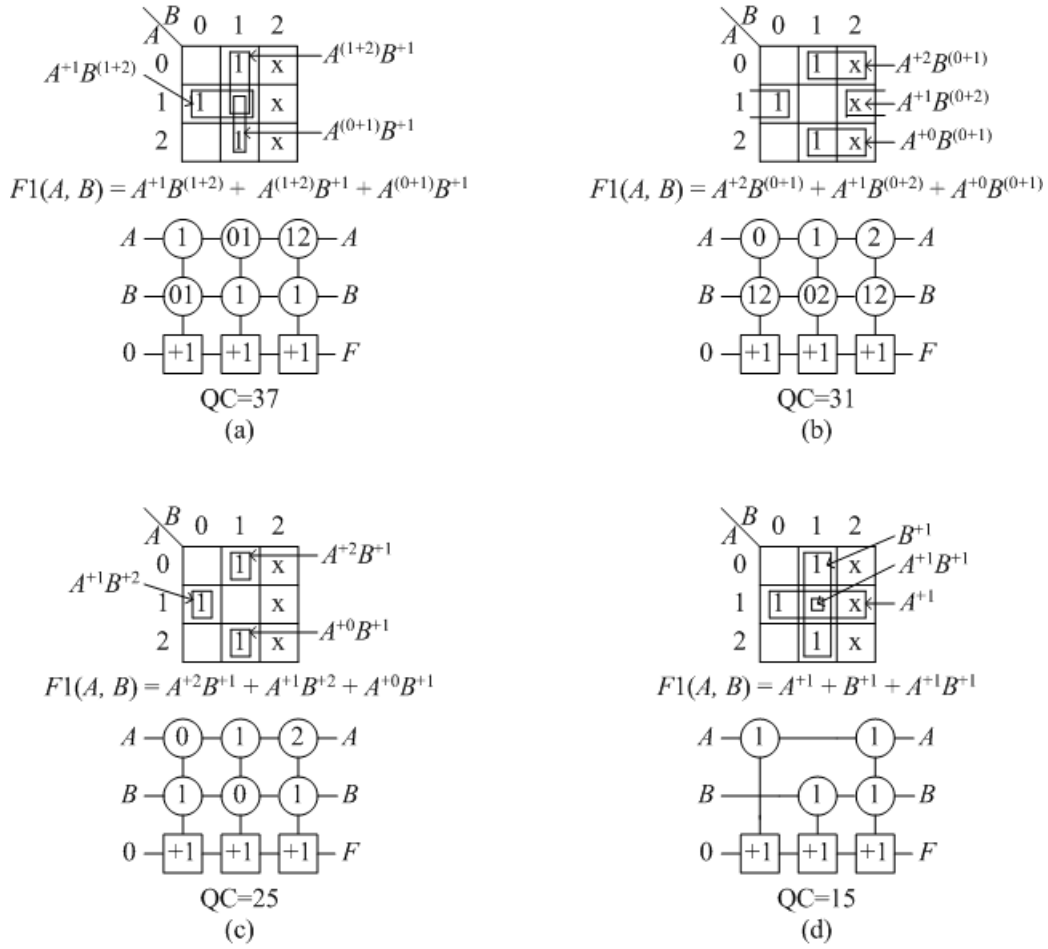

Figure 7.5: Four different solutions of the sub-function $F1(A,B)$ in Table 7.1.

and 2. The minterm corresponding to this group is $A^{+0}B^{(0+1)}$. In all three groups one cell with a 1 and one cell with an x are grouped together. The Max-Min expression from this K-map is given in (7.6).

$$F1(A,B) = A^{+2}B^{(0+1)} + A^{+1}B^{(0+2)} + A^{+0}B^{(0+1)} \tag{7.6}$$

In Figure 7.5(c), the cell corresponding to input combination 01 is grouped as a group of one-cell. In this group $A = 0$ and $B = 1$. The canonical minterm corresponding to this group is $A^{+2}B^{+1}$. The cell corresponding to input combination 10 is grouped as a group of one-cell. In this group $A = 1$ and $B = 0$. The canonical minterm corresponding to this group is $A^{+1}B^{+2}$. The cell corresponding to input combination 21 is grouped as a group

of one-cell. In this group $A = 2$ and $B = 1$. The canonical minterm corresponding to this group is $A^{+0}B^{+1}$. The Max-Min expression from this K-map is given in (7.7).

$$F1(A,B) = A^{+2}B^{+1} + A^{+1}B^{+2} + A^{+0}B^{+1} \tag{7.7}$$

In Figure 7.5(d), cells corresponding to input combinations 10, 11, and 12 are grouped as a group of three-cells. In this group $A = 1$ and B varies over 0, 1, and 2. The minterm corresponding to this group is A^{+1}. In this group one cell with a 1, one cell with a 0, and one cell with an x are grouped together. Cells corresponding to input combinations 01, 11, and 21 are grouped as a group of three-cells. In this group A varies over 0, 1, and 2; and $B = 1$. The minterm corresponding to this group is B^{+1}. In this group two cells with 1s and one cell with a 0 are grouped together. Cell corresponding to input combination 11 is grouped as a group of one-cell. In this group $A = 1$ and $B = 1$. The canonical minterm corresponding to this group is $A^{+1}B^{+1}$. The cell with a 0 corresponding to the input combination 11 is grouped thrice satisfying the grouping rule. The Max-Min expression from this K-map is given in (7.8).

$$F1(A,B) = A^{+1} + B^{+1} + A^{+1}B^{+1} \tag{7.8}$$

7.4.2 Some Observations

Four possible minimizations of the sub-function $F1(A,B)$ in Table 7.1 are shown in Figure 7.5. In Figure 7.5, Max-Min expressions, corresponding reversible circuits, and quantum costs (QCs) of the reversible circuits for all four minimization are shown. From Figure 7.5 we see that the quantum costs for four possible minimization are different and the quantum cost for the minimization of Figure 7.5(d) is the minimum. Therefore we need to develop intuitive methods to directly determine the minimum solution from the ternary K-map. After experimenting with several example functions we arrive at the following observations:

1. In the ternary K-map of Figure 7.5(a) three groups of two cells are formed having one cell with a 1 and another cell with a 0. The corresponding circuit requires a quantum cost of 37. In Figure 7.5(c) the same function is minimized without including the cell with a 0. The corresponding circuit requires a quantum cost of 25. In Figure 7.5(a) each group has exactly 50% cells with a 0, which increases the quantum cost. Consider another three-variable example sub-function $F1(A,B,C)$ represented in Figure 7.6. In Figure 7.6(a) three groups are formed with 50% or more cells with 0s in all three groups. The resulting circuit requires a quantum cost of 51. In Figure 7.6(b) three groups are formed without including any cell with a 0. The resulting circuit requires a quantum cost of 49. Again we see that inclusion of 50% or more cells with 0s in groups of 1s increases the quantum cost. Consider another four-variable example sub-function $F1(A,B,C,D)$ represented in Figure 7.7. In Figure 7.7(a) three groups are formed with less than 50% cells with 0s in all three groups. The resulting circuit requires a quantum cost of 71. In Figure 7.7(b) four groups are formed without including any cell with a 0. The resulting circuit requires a quantum cost of 96. Here we see that inclusion of less than 50% cells with 0s in groups of 1s reduces the quantum cost. We tested five more artificially generated example functions and found experimentally that if the number of cells with 0s in a group is less than 50%, then the quantum cost of the resulting circuit is decreased. We are theorizing that when 50% or more cells with 0s are included in a group, then the resulting Max-Min expression requires more composite literals; however this has not been proven. Circuits with more composite literals will require multiple-controlled unary gates with more composite controls. As the quantum cost of realization of composite controls are higher than those of simple controls (see Section 5.4), inclusion of 50% or more cells with 0s in a group increases the quantum cost. Thus we have developed our technique assuming that *in a group on the ternary K-map the number of cells with 0s must be less than 50%*.

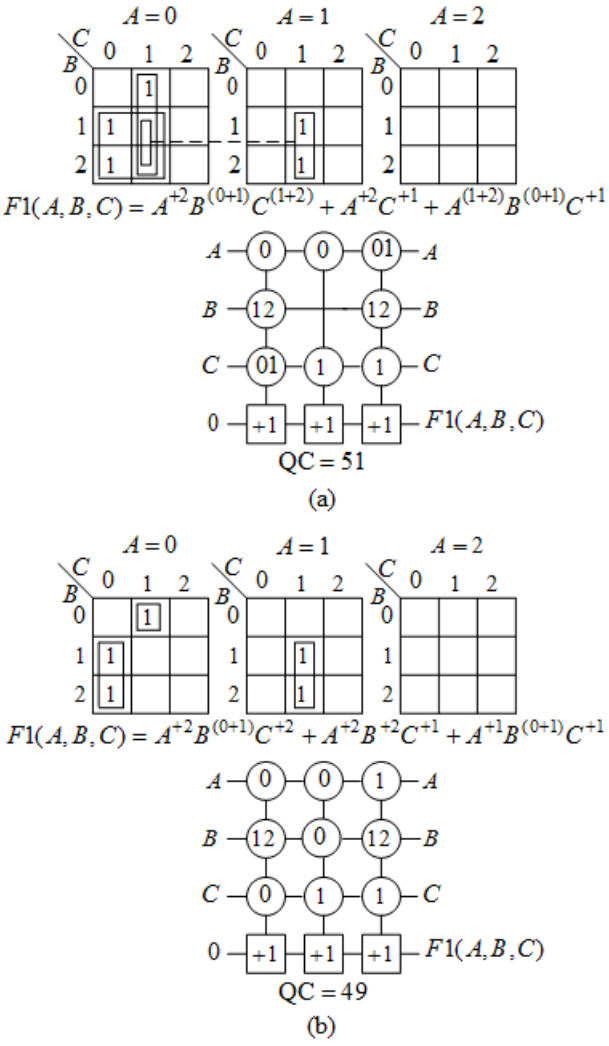

Figure 7.6: Two possible minimization of a three-variable sub-function $F1(A,B,C)$.

2. In the ternary K-map of Figure 7.5(b) three groups of two cells are formed having one cell with a 1 and another cell with an x. The corresponding circuit requires a quantum cost of 31. In comparison the circuit for the minimization of Figure 7.5(c) without including cells with xs requires a quantum cost of 25. In Figure 7.5(b) all three groups have 50% cells with an x, which increases the quantum cost. Consider another two-variable example incompletely specified sub-function $F1(A,B)$ represented in Figure 7.8. In Figure 7.8(a) the group has 50% cells with xs. The resulting circuit requires a quantum cost of 15. In Figure 7.8(b) the group is formed without including any cell with an x. The resulting circuit requires a quantum cost of 11. Again we

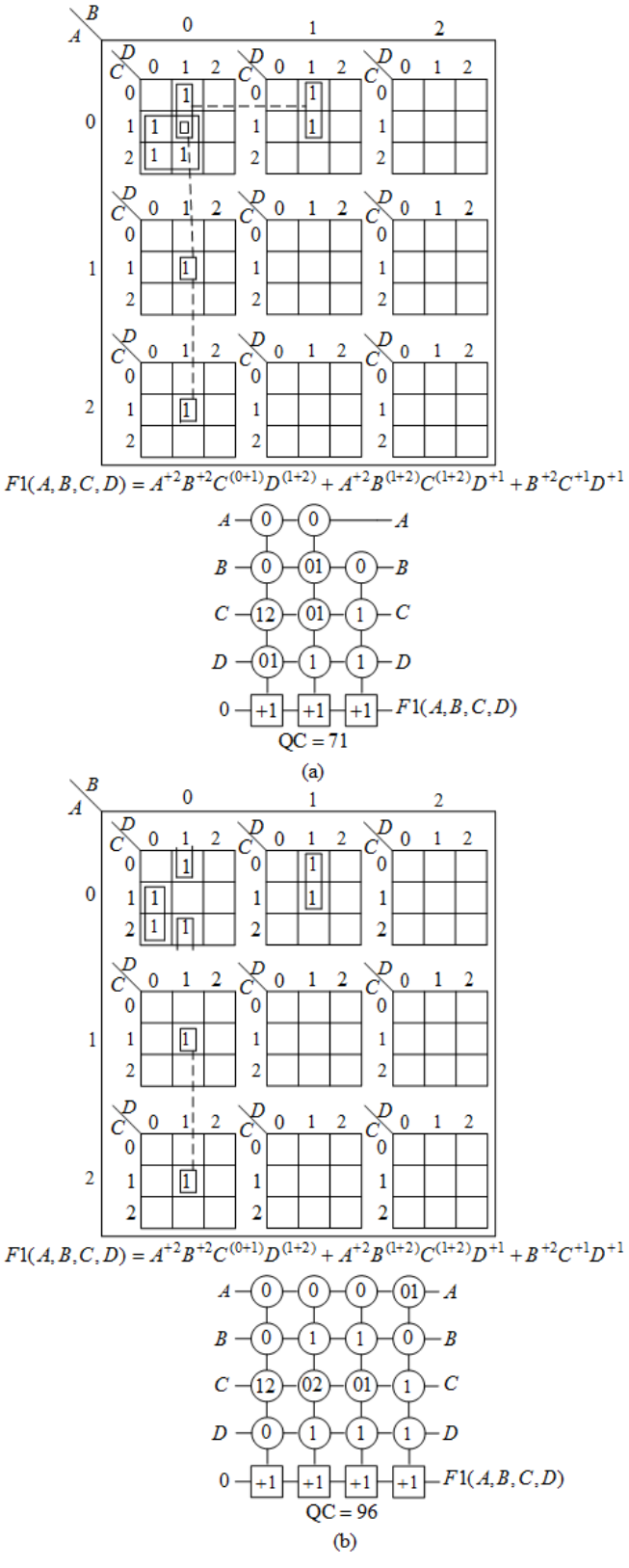

Figure 7.7: Two possible minimizations of a four-variable sub-function $F1(A,B,C,D)$.

see that inclusion of 50% cells with xs in groups of 1s increases the quantum cost. Consider another two-variable example incompletely specified sub-function $F1(A,B)$ represented in Figure 7.9. In Figure 7.9(a) the group contains less than 50% cells with an x. The resulting circuit requires a quantum cost of 15. In Figure 7.9(b) two groups are formed without including any cell with an x. The resulting circuit requires a quantum cost of 16. Here we see that inclusion of less than 50% cells with xs in groups of 1s reduces the quantum cost. We tested five more artificially generated example functions and found experimentally that if the number of cells with xs in a group is less than 50%, then the quantum cost of the resulting circuit is decreased. We are theorizing that when 50% or more cells with xs are included in a group, then the resulting Max-Min expression requires more composite literals; however this has not been proven. Circuits with more composite literals will require multiple-controlled unary gates with more composite controls. As the quantum cost of realization of composite controls are higher than those of simple controls (see Section 5.4), inclusion of 50% or more cells with xs in a group increases the quantum cost. Thus we have developed our technique assuming that *in a group on the ternary K-map the number of cells with xs must be less than 50%.*

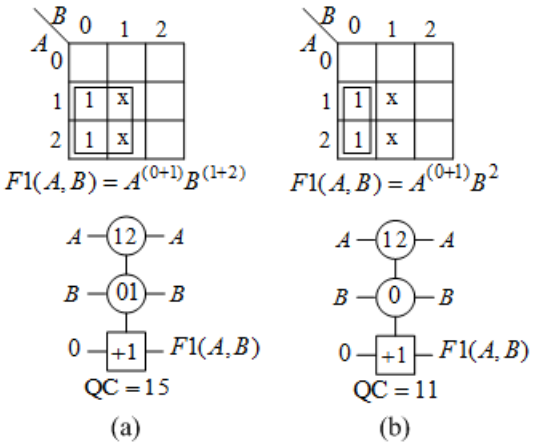

Figure 7.8: Two possible minimizations of a two-variable sub-function $F1(A,B)$.

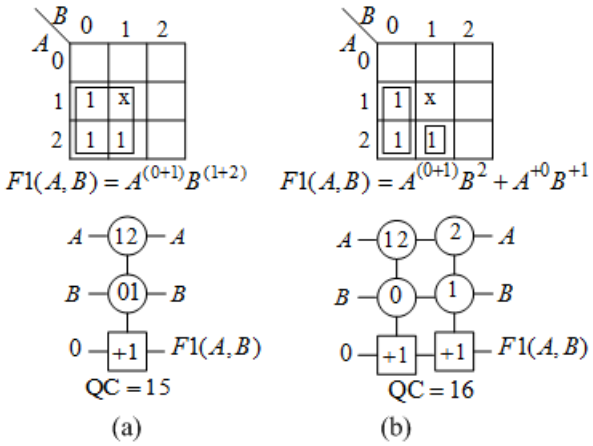

Figure 7.9: Two possible minimizations of a two-variable sub-function $F1(A,B)$.

3. In Figure 7.5(d) a group of one-cell with a 0 is used. The minimization of Figure 7.5(d) produces the optimum solution. Consider another two-variable example sub-function $F1(A,B)$ represented in Figure 7.10. In Figure 7.10(a) a group of two-cells with 0s is used. The resulting circuit requires a quantum cost of 18. In Figure 7.10(b) only the cells with 1s are grouped. The resulting circuit requires a quantum cost of 22. Again we see that inclusion of group of cells with only 0s reduces the quantum cost. We tested five more artificially generated example functions and found experimentally that inclusion of cells with only 0s in groups reduces the quantum cost. We are theorizing that inclusion of cells with 0s in groups provided that those cells are covered by a multiple of three groups reduces the quantum cost. Thus we have developed our technique assuming that *a group of cells with only 0s is useful in minimization of Max-Min expressions*. A group of cells with only 0s is formed within another larger group to make sure that cells with 0s are covered a multiple of three times.

7.4.3 K-Map-Based Minimization Method

Based on the two grouping rules of Section 7.3 and the three observations discussed above, the K-map-based minimization of Max-Min expression for a sub-function is as follows:

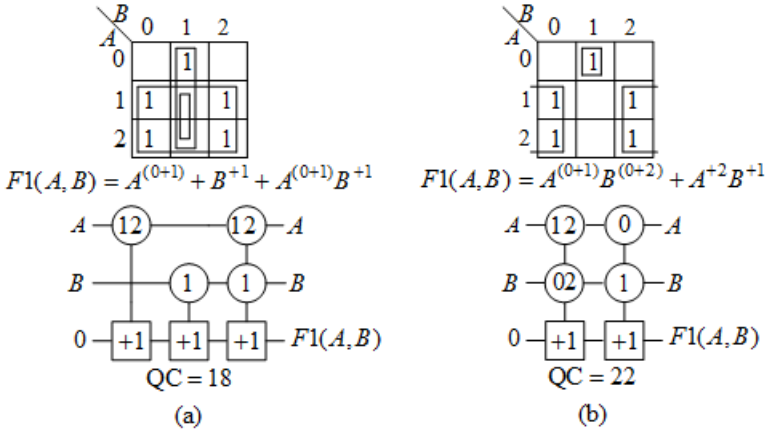

Figure 7.10: Two possible minimizations of a two-variable sub-function $F1(A,B)$.

1. Put the ternary-input binary-output sub-function on a ternary K-map.

2. Form the largest possible groups of cells maintaining the following restrictions:

   - A cell with a 1 must be covered by a multiple of three plus one groups.
   - A cell with a 0 must be covered by a multiple of three groups.
   - The number of cells with 0s in a group should be less than 50%.
   - The number of cells with xs in a group should be less than 50%.
   - A group of cells with only 0s is included within another larger group, if needed.

   At the same time select smallest possible number of groups.

3. Determine minterms for all groups and then combine them using Max operations to form the minimized Max-Min expression.

We show some examples of K-map-based minimization in the next section.

7.5 Ternary K-Map-Based Minimization Examples

In Figure 7.11(a) minimization of a two-variable sub-function $F1(x,y)$ is shown, where a group of two-cells with input combinations $xy = 01$ and $xy = 21$ and a group of three-cells with input combinations $xy = 10$, $xy = 11$, and $xy = 12$ are used. For the group of two-cells

the variable x varies over 0 and 2 and the variable $y = 1$. So, the minterm for this group is $x^{(0+2)}y^{+1}$. For the group of three cells the variable $x = 1$ and the variable y varies over 0, 1, and 2. So, the minterm of this group is x^{+1}. The resulting Max-Min expression from the K-map of Figure 7.11(a) is given in (7.9).

$$F1(x,y) = x^{+1} + x^{(0+2)}y^{+1}. \tag{7.9}$$

In Figure 7.11(b) minimization of a three-variable sub-function $F1(x,y,z)$ is shown. Here three groups of three-cells (the first group with input combinations 010, 011, 012; the second group with input combinations 001, 011, 021; and the third group with input combinations 011, 111, 211) are used, where the cell with a 0 corresponding to the input combination 011 is overlapped thrice. The resulting Max-Min expression from the K-map of Figure 7.11(b) is given in (7.10).

$$F1(x,y,z) = x^{+2}y^{+1} + x^{+2}z^{+1} + y^{+1}z^{+1}. \tag{7.10}$$

In Figure 7.11(c) minimization of a four-variable sub-function $F1(w,x,y,z)$ is shown, where a group of 27-cells with 1s is used and the resulting Max-Min expression from the K-map of Figure 7.11(c) is given in (7.11).

$$F1(w,x,y,z) = y^{+2}. \tag{7.11}$$

The truth table for a ternary full-adder and corresponding sub-functions are shown in Table 7.2.

The minimization of the Max-Min expression for the sub-function $Cout1$ of a ternary full-adder from the truth table in Table 7.2 is shown in Figure 7.12. In Figure 7.12 one group of three-cells and one group of six-cells are used, where the group of three-cells and the group of six-cells contain less than 50% cells with xs. However, the cells corresponding

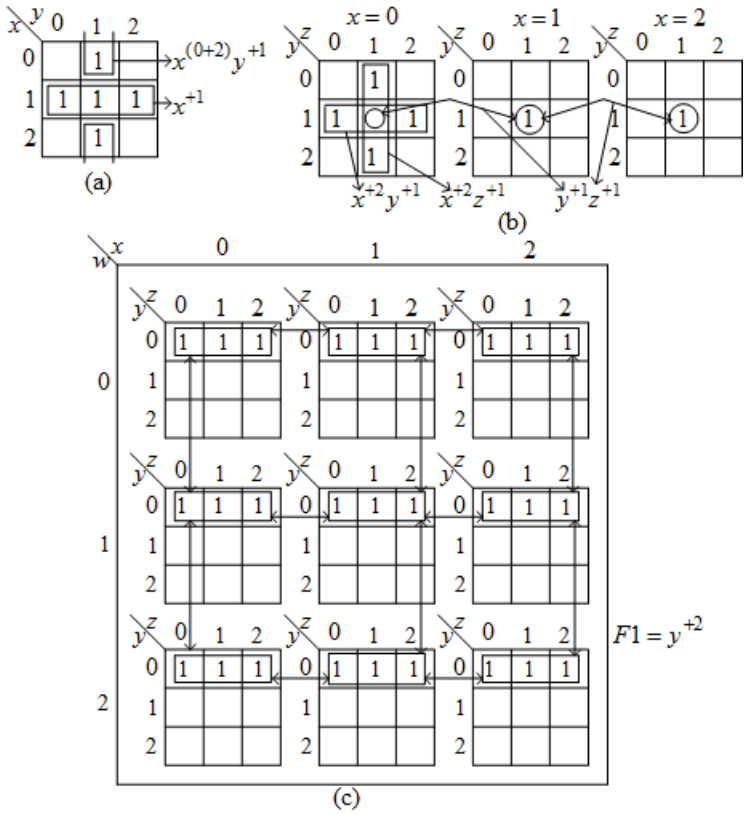

Figure 7.11: Examples of (a) a two-variable, (b) a three-variable, and (c) a four-variable ternary sub-function minimization.

to the input combinations 021, 111, and 201 cannot be grouped with other cells, since in that case the other cell with a 1 will be overlapped twice and the function will be changed. The cells with 1s also cannot be grouped with cells with xs satisfying the grouping rule of don't care (x) cells. Thus three groups of one-cell are used. The resulting Max-Min expression from the K-map of Figure 7.12 is shown in (7.12).

$$C_{out1} = A^{+0}B^{(0+1)} + A^{+1}B^{+0} + A^{+2}B^{+0}C_{in}^{+1} + A^{+1}B^{+1}C_{in}^{+1} + A^{+0}B^{+2}C_{in}^{+1} \qquad (7.12)$$

The minimization of sub-function $S0$ of a ternary full-adder from the truth table in Table 7.2 is shown in Figure 7.13. In Figure 7.13 cells with 1s cannot be grouped with other cells with 0s or xs, since in that case the condition of including cells with 0s and xs will not be satisfied. The resulting Max-Min expression from the K-map of Figure 7.13 is

Table 7.2: Truth table for a ternary full-adder with all sub-functions.

ABC_{in}	$CoutS$	$Cout0$	$Cout1$	$S0$	$S1$	$S2$
000	00	1	0	1	0	0
001	01	1	0	0	1	0
002	xx	x	x	x	x	x
010	01	1	0	0	1	0
011	02	1	0	0	0	1
012	xx	x	x	x	x	x
020	02	1	0	0	0	1
021	10	0	1	1	0	0
022	xx	x	x	x	x	x
100	01	1	0	0	1	0
101	02	1	0	0	0	1
102	xx	x	x	x	x	x
110	02	1	0	0	0	1
111	10	0	1	1	0	0
112	xx	x	x	x	x	x
120	10	0	1	1	0	0
121	11	0	1	0	1	0
122	xx	x	x	x	x	x
201	10	0	1	1	0	0
200	02	1	0	0	0	1
202	xx	x	x	x	x	x
210	10	0	1	1	0	0
211	11	0	1	0	1	0
212	xx	x	1	x	x	x
220	11	0	1	0	1	0
221	12	0	1	0	0	1
222	xx	x	x	x	x	x

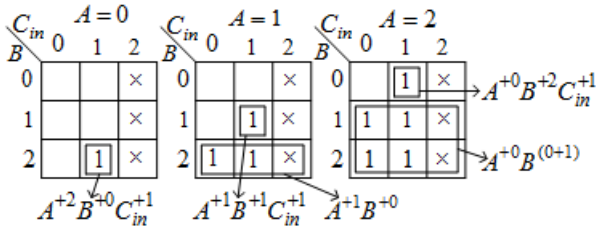

Figure 7.12: Minimization of Max-Min expression for sub-function $Cout1$ of a ternary full-adder from the truth table in Table 7.2.

shown in (7.13).

$$S0 = A^{+2}B^{+2}C_{in}^{+2} + A^{+2}B^{+0}C_{in}^{+1} + A^{+1}B^{+1}C_{in}^{+1} + A^{+1}B^{+0}C_{in}^{+2} + A^{+0}B^{+2}C_{in}^{+1} + A^{+0}B^{+1}C_{in}^{+2} \tag{7.13}$$

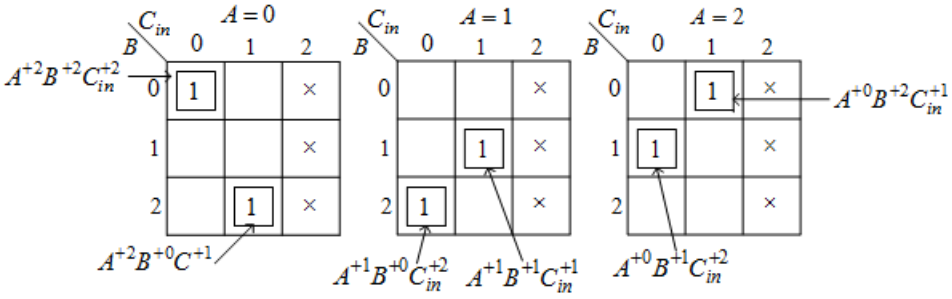

Figure 7.13: Minimization of Max-Min expression for sub-function $S0$ of a ternary full-adder from the truth table in Table 7.2.

In the K-map-based minimization of Max-Min expressions, groupings of cells are done while attempting to maintain the grouping constraints of cells with 1s and 0s and also satisfy the other observations. Thus manual minimization is very difficult.

7.6 Comparison Between K-Map-Based ESOP and Ternary Max-Min Minimizations

The ESOP expression is based on GF2 arithmetic and the ternary Max-Min expression is based on GF3 arithmetic (see Chapter 2). Therefore, both the minimization methods are similar. These similarities are summarized below:

1. The K-map-based ESOP function minimization is based on GF2 and a 1 can only be covered by an odd number of groups. The proposed ternary K-map-based Max-Min function minimization is based on GF3 and a 1 can only be covered by a multiple of 3 plus 1 groups.

2. In the K-map-based ESOP function minimization a 0 can be included in a group provided that 0 is covered by an even number of groups. In the proposed ternary K-map-based Max-Min function minimization a 0 can be included in a group provided that 0 is covered by a multiple of 3 groups.

3. In the K-map-based ESOP function minimization a smaller group can be formed within another larger group to ensure that a 1 is covered by an odd number of groups and a 0 is covered by an even number of groups. Similarly, in the proposed ternary K-map-based Max-Min function minimization a smaller group can be formed within another larger group to ensure that a 1 is covered by a multiple of 3 plus 1 groups and a 0 is covered by a multiple of 3 groups.

4. In the K-map-based ESOP function minimization the group size is 2^m, where $0 \leq m \leq n$ for an n-variable K-map. Thus the possible group sizes are 1, 2, 4, 8, 16, 32, and 64 for up to 6 variables. In the proposed ternary K-map-based Max-Min function minimization the possible group sizes are 2^m or 3^m, where $m = 0$; or 2^m, where $0 < m \leq 4$; or 3^m, where $0 < m \leq 4$; or $2^i \times 3^j$, where $i, j \geq 1$ and $0 < (i + j) \leq 4$ as discussed in Section 7.3. Thus the possible group sizes are 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 36, 54, and 81 for up to 4 variables.

Chapter 8

Hybrid Genetic Algorithm-Based Synthesis of Ternary Reversible Circuits Using Max-Min Algebra

8.1 Introduction

In Chapter 7 we proposed a ternary K-map-based minimization of ternary Max-Min expressions of up to four variables. From the discussions in Chapter 7 it is clear that the K-map-based method is very difficult to use for three and four variable functions. Moreover, using a ternary K-map for more than four variables is more difficult. For example a five-variable K-map will consist of three four-variable K-maps (see Figure 2.3(c) for structure of a four-variable K-map). Identifying adjacent cells in a five-variable ternary K-map will be very difficult. Therefore, a computer-aided method must be developed for minimization of ternary Max-Min expressions.

In a computer-aided method of minimizing Max-Min expressions, the following types of groups are potential groups for determining the minimum solution:

- All possible groups of 1s.
- All possible groups of 0s.
- All possible groups of 1s and 0s such that the number of 0s is less than 50% of the group size.
- All possible groups of 1s and xs such that the number of xs is less than 50% of the

group size.

- All possible groups of 1s, 0s, and xs such that both the numbers of 0s and xs are less than 50% of the group size.

The number of such potential candidate groups is very high. Thus, an exhaustive search for the minimum solution is an exponential time problem and will not be practically feasible. Therefore, we need to use some heuristic method for finding the minimum or near minimum solution.

It is well known that Genetic Algorithms (GAs) can be used for optimization of complex problems [29, 30]. In this chapter we propose a hybrid Genetic Algorithm [9] for minimization of ternary Max-Min expressions and then synthesis of the corresponding reversible circuits.

This chapter is organized as follows:

- In Section 8.2 we briefly introduce GAs.
- In Section 8.3 we discuss the process of generating potential minterms, which are the used as candidates for GA-based minimization of ternary Max-Min expressions.
- In Section 8.4 we discuss encoding of the Max-Min minimization problem into the GA domain; that is, we discuss the structure of the chromosome of the proposed GA for minimization of ternary Max-Min expressions.
- In Section 8.5 we propose a hybrid GA (HGA) for minimization of ternary Max-Min expressions.
- In Section 8.6 we discuss ternary reversible circuit synthesis from output of the HGA.

8.2 Brief Introduction to Genetic Algorithms (GAs)

A GA is a population-based meta-heuristic method for search and optimization problems. The basic unit of a GA is a chromosome. A chromosome is a collection of genes.

In binary encoded GAs each gene is either a 1 or a 0. When a gene is 1, then that gene is present in the solution. When a gene is 0, then that gene is absent in the solution. The solution of a search or optimization problem (called a phenotype) is encoded in such a way that it can be represented by a chromosome (called a genotype). The number of genes in a chromosome is called the chromosome length. Figure 8.1 shows an example chromosome of length eight.

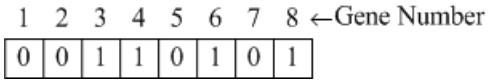

Figure 8.1: An example chromosome of length eight.

A collection of chromosomes is called the population. The number of chromosomes in a population is called the population size. Figure 8.2 shows an example population of size four with chromosome length eight.

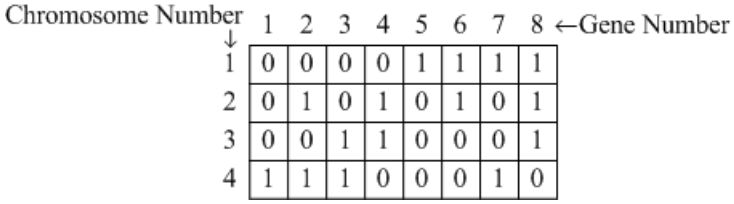

Figure 8.2: An example population of size four with chromosome length eight.

The structure of a steady-state GA [29] is shown in Algorithm 1. Each operation of a steady-state GA is discussed below.

In line 2 the procedure initialPopulation(populationSize, chromosomeLength) generates the initial population randomly. For determining the structure of the chromosome the problem specific solution (phenotype) is encoded into suitable chromosome structure (genotype). The encoding from phenotype to genotype is problem specific.

In lines 3 to 5 the procedure determineFitness(chromosome) determines the fitness of all the chromosomes. Fitness is the measure of "goodness" of the chromosome. The fitness function is problem specific. For determining the fitness the chromosome (genotype) is

Algorithm 1 Steady-State Genetic Algorithm

```
1:  procedure STEADY-STATE GENETIC ALGORITHM
2:      population=initialPopulation(populationSize, chromosomeLength)
3:      for (each chromosome in the population) do
4:          determineFitness(chromosome)
5:      end for
6:      maximumFitness=determineMaximumFitness(population)
7:      while (Termination Condition Not Reached) do
8:          parent1=selection(population)
9:          parent2=selection(population)
10:         {offspring1, offspring2}=crossover(parent1, parent2, P_C)
11:         offspring1=mutation(offspring1, P_M)
12:         offspring2=mutation(offspring2, P_M)
13:         determineFitness(offspring1)
14:         determineFitness(offspring2)
15:         duplicate1=duplicateChecking(population, offspring1)
16:         duplicate2=duplicateChecking(population, offspring2)
17:         if (duplicate1 = False) then
18:             replace(population, offspring1)
19:         end if
20:         if (duplicate2 = False) then
21:             replace(population, offspring2)
22:         end if
23:         newMaximumFitness=determineMaximumFitness(population)
24:         if (newMaximumFitness > maximumFitness) then
25:             maximumFitness=newMaximumFitness
26:         end if
27:     end while
28: end procedure
```

decoded to problem specific solution (phenotype) and then from the solution the fitness is calculated.

In line 6 the procedure determineMaximumFitness(population) determines the maximum fitness of the population and stores the maximum fitness along with the corresponding chromosome.

The while loop of lines 7 to 27 constitute a generation in the evolution process. The loop is terminated when the termination condition is reached. Three termination techniques are normally used in GAs [30] as discussed below:

1. The GA is terminated after a fixed number of generations. This technique does not always produce optimal results.

2. The GA is terminated when the result satisfies an optimal criteria. This technique requires prior knowledge of the solution.

3. The GA is terminated when the highest fit solution reaches a saturation condition; that is, the highest fitness value does not change for a predefined number of consecutive generations.

In lines 8 and 9 the procedure selection(population) selects two parents from the population. Three parent selection methods are normally used [29]:

1. Roulette Wheel Selection

2. Stochastic Universal Selection

3. Binary Tournament Selection

Roulette wheel selection and stochastic universal selection methods select a parent having higher fitness value with higher probability. These methods provide a way to exploit the properties of higher fit chromosomes, but do not provide means to explore the search space by creating diversification in the selection process. The highest diversification in parent selection can be provided by randomly selecting the parents. However, random selection of parents limits the opportunity of exploiting the properties of the higher fit chromosomes. Binary tournament selection combines the advantages of both the approaches and creates a balance between exploitation of the properties of higher fit chromosomes and exploration of the search space by providing diversity in the selection process. In binary tournament selection two chromosomes are randomly selected and the better fit chromosome is selected as parent1. Similarly parent2 is selected.

In line 10 the procedure crossover(parent1, parent2, P_C) generates two offspring from two parents using the crossover operation. The crossover operation is applied on two par-

ents with high crossover probability P_c. In one-point crossover, a crossover point is randomly determined and then either the left part or the right part of the crossover point of the two parents are swapped. One example of crossover operation is shown in Figure 8.3.

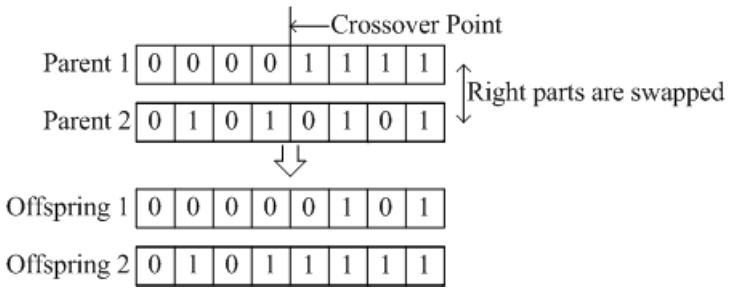

Figure 8.3: Crossover operation.

In line 11 the procedure mutation(offspring1, P_M) changes bits of offspring1 using the mutation operation. The mutation operation is applied with low mutation probability P_M. The mutation operation is illustrated in Figure 8.4. In line 12 offspring2 is similarly mutated.

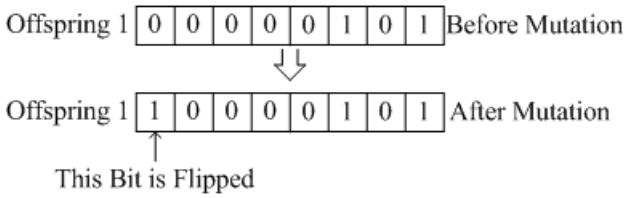

Figure 8.4: Mutation operation.

In line 13 the procedure determineFitness(offspring1) determines the fitness of the offspring1. Similarly in line 14 the procedure determineFitness(offspring2) determines the fitness of the offspring2.

In line 15 the procedure duplicateChecking(population, offspring1) determines whether offspring1 already exists in the population or not. If offspring1 already exists in the population, then duplicate1 is TRUE otherwise duplicate1 is FALSE. Similarly, in line 16 duplication status of offspring2 is determined.

In lines 17 to 19 if duplicate1 is FALSE, then the procedure replace(population, offspring1) replaces one chromosome of the population by offspring1. In the replace proce-

dure the minimum fitness of the population and the corresponding chromosome are determined. If the fitness of offspring1 is greater than the minimum fitness of the population, then the corresponding chromosome is replaced by offspring1. Similarly in lines 20 to 22 another chromosome in the population with minimum fitness is replaced by offspring2.

In lines 23 to 26 the chromosome with the maximum fitness in the new population is determined and saved.

8.3 Generation of Potential Candidate Minterms for Solution

8.3.1 Encoding of Variable Value Variation

The changes of values of a variable in a group of cells on a K-map are encoded using three bits as shown in Table 8.1. The leftmost bit represents the value 2, the middle bit represents the value 1, and the rightmost bit represents the value 0. A 0 in a bit position for a variable represents that in the given group the variable does not correspond to that value. A 1 in a bit position for a variable represents that in the given group the variable corresponds to that value. In a group, if a variable varies over more than one value, then the encoded representation of the variable has more than one 1.

Table 8.1: Variable value encoding in a group of cells.

Encoded form	Variable value in a group of cells
000	
001	0
010	1
011	01
100	2
101	02
110	12
111	012

An example two-variable ternary logic function and its three sub-functions are shown in Table 8.2. The sub-function $F1(A,B)$ is represented on a ternary K-map in Figure 8.5.

In Figure 8.5 the encoded input combinations of the cells containing 1s and xs are shown. Consider the cell corresponding to the input combination $AB = 01$ containing a 1. Here $A = 0$ and is encoded as 001 and $B = 1$ and is encoded as 010. The two codes are concatenated as 001 010 as the encoded input combination for the cell corresponding to the input combination $AB = 01$.

Table 8.2: An example two-variable ternary logic function and its three sub-functions.

AB	$F(A,B)$	$F0(A,B)$	$F1(A,B)$	$F2(A,B)$
00	0	1	0	0
01	1	0	1	0
02	x	x	x	x
10	1	0	1	0
11	0	1	0	0
12	x	x	x	x
20	2	0	0	1
21	1	0	1	0
22	x	x	x	x

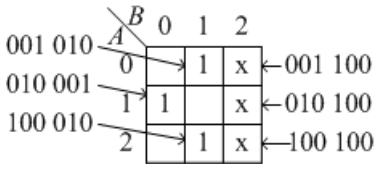

Figure 8.5: Sub-function $F1(A,B)$ of Table 8.2 on a two-variable ternary K-map.

The motivation of this encoding is that if a variable varies over two or three values, then the encoded representation can be determined by doing bit-wise OR of the individual codes. Two examples are shown in Figure 8.6.

Value	Code
0	001
1	010
01	011

Bit-wise ORed

Value	Code
01	011
12	110
012	111

Bit-wise ORed

Figure 8.6: Combination of encoded inputs by doing bit-wise OR.

Readers should note that this variable value encoding is not chromosome encoding for

GA. This variable value encoding is used in the minterm generation process discussed below.

8.3.2 Minterm Generation

From the discussions of Chapter 7 we see that any group except those containing a mix of 0s and xs are useful for minimization of Max-Min expressions. In the GA-based minimization process we need to generate all possible minterm candidates for a solution.

The generation process of candidate minterms for a solution is as follows:

1. In Stage-0 list all 3^n canonical minterms irrespective of functional output, where n is the number of variables in the function.

2. In Stage-1 generate all minterms corresponding to all groups of two-cells. Two cells can be grouped together if and only if they differ in only one variable.

3. In Stage-2 generate all minterms corresponding to all groups of three or four cells. Two groups of two cells can be grouped together if and only if they differ in only one variable.

4. In a similar manner generate all minterms of groups of Stage-n from groups of Stage-$(n-1)$.

5. Finally, to reduce the search space, eliminate groups less likely to contribute towards finding the solution (less potential groups) and determine the list of groups most likely to contribute towards finding the solution (potential candidate groups for solution).

A partial example of generating groups considering only three cells is shown in Figure 8.7. In Figure 8.7 the input combinations are shown in decimal notation. For example, consider the cell with input combination $AB = 00$. The corresponding input combination is denoted as (0)(0). Also consider the group of two-cells corresponding to $A = 0$ and $B = 0$ and 1. The corresponding input combination is written as (0)(01). Consider the group of

three cells corresponding to $A = 0$ and $B = 0$, 1, and 2. The corresponding input combination is written as (0)(012) = (0)(-). Here the missing variable is represented as (-). However in our minterm generation program the input combinations are represented in encoded form so that variation in only one variable can be easily detected and two encoded terms can be combined by bit-wise OR to get the resulting encoded term for the larger group.

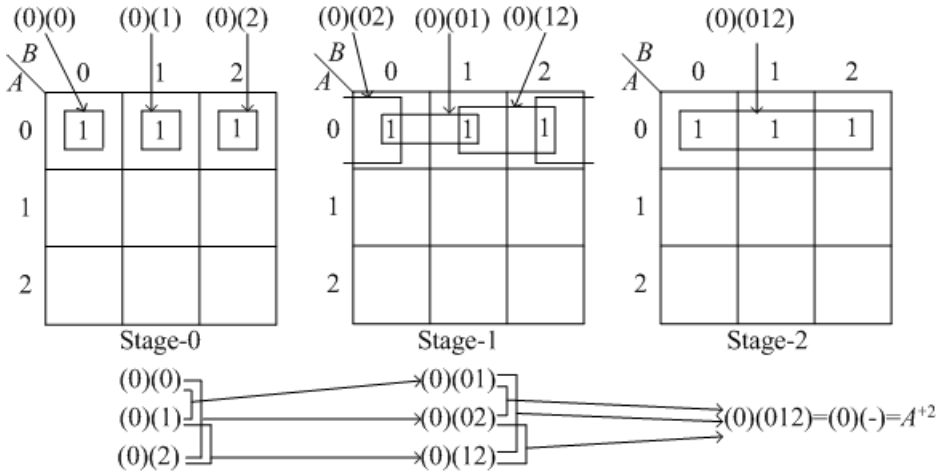

Figure 8.7: Partial example of generation of minterms.

In Figure 8.7 Stage-0 represents three canonical minterms. Stage-1 minterms (group size two) are generated from Stage-0 minterms. Stage-2 minterm (group size three) is generated from Stage-1 minterms.

8.3.3 Reducing the Number of Potential Minterms

From Chapter 7 we see that the minimization of a Max-Min expression is a difficult process. Therefore any minterm containing 1 to n literals, where n is the number of variables in the function, may be a potential candidate for final solution. For this reason we generate all possible minterms. However, from Section 7.4 we see that two types of minterms are less likely to contribute towards finding the optimum solution for obtaining an optimal minimization of Max-Min expressions. There are three more observations which identify three more types of minterms with lower potential for obtaining optimal solution using GAs. All five types of lower potential minterms are listed below:

1. A minterm corresponding to a group with equal to or more than 50% cells with 0s is a minterm with lower potential for obtaining an optimal (or near optimal) solution (observation from Section 7.4).

2. A minterm corresponding to a group with equal to or more than 50% cells with xs is a minterm with lower potential for obtaining an optimal (or near optimal) solution (observation from Section 7.4).

3. A minterm corresponding to a group with all cells with xs is redundant, since a minterm generated from a group with only cells with xs is not needed for the function. Thus a minterm corresponding to a group with all cells with xs is a minterm with lower potential.

4. A minterm corresponding to a group with cells containing only 0s and xs is a minterm with lower potential, since such a group does not properly match with cells with 0s of a valid group with cells containing a mix of 1, 0, and x.

5. A group of 3^n cells, where n is the number of variables in the function, is not needed. Thus this minterm is a minterm with lower potential.

In our proposed hybrid Genetic Algorithm (HGA) we excluded the above five types of minterns to reduce the search space so the HGA finds a good (optimal or near optimal) solution within a reasonable time.

8.3.4 Example of Minterm Generation

The concepts discussed above are illustrated in Tables 8.3 and 8.4 by generating all minterms for the sub-function $F1(A,B)$ in Table 8.2. In Tables 8.3 and 8.4, we also list the quantum cost (QC) of multiple-controlled unary gates corresponding to all minterms. The minterms are numbered from 0 for referencing purposes.

After elimination of minterms with lower potential we get the minterms listed in Table 8.5 as potential candidates for the optimal (or near optimal) solution.

Table 8.3: Minterm generation for sub-function $F1(A,B)$ in Table 8.2.

Initial Input: Stage-0				
Term No	Minterm	Output	Less Potential	QC
0	001 001	0		9
1	001 010	1		9
2	001 100	x	y	7
3	010 001	1		9
4	010 010	0		9
5	010 100	x	y	7
6	100 001	0		7
7	100 010	1		7
8	100 100	x	y	5
Stage-1				
Term No	Minterm	Combining Minterms	Less Potential	QC
9	001 011	0,1	y	13
10	001 101	0,2	y	11
11	011 001	0,3	y	13
12	101 001	0,6		11
13	001 110	1,2	y	11
14	011 010	1,4	y	13
15	101 010	1,7		11
16	011 100	2,5	y	11
17	101 100	2,8	y	9
18	010 011	3,4	y	13
19	010 101	3,5	y	11
20	110 001	3,6	y	11
21	010 110	4,5	y	11
22	110 010	4,7	y	11
23	110 100	5,8	y	9
24	100 011	6,7	y	11
25	100 101	6,8	y	9
26	100 110	7,8	y	9

8.3.5 Algorithm for Minterm Generation

A generalized algorithm for minterm generation is shown in Algorithm 2. In the algorithm for generating minterms the input is the list of 3^n input combinations represented in

Table 8.4: Minterm generation for sub-function $F1(A,B)$ in Table 8.2 (continued).

Stage-2				
Term No	Minterm	Combining Minterms	Less Potential	QC
27	001 111	9,10/9,13/10,13		3
28	011 011	9,18/11,14	y	17
29	101 011	9,24/12,15	y	15
30	011 101	10,19/11,16	y	15
31	101 101	10,25/12,17	y	13
32	111 001	11,12/11,20/12,20	y	3
33	011 110	13,21/14,16	y	15
34	101 110	13,26/15,17	y	13
35	111 010	14,15/14,22/15,22		3
36	111 100	16,17/16,23/17,23	y	1
37	010 111	18,19/18,21/19,21		3
38	110 011	18,24/20,22	y	15
39	110 101	19,25/20,23	y	13
40	110 110	21,26/22,23	y	13
41	100 111	24,25/24,26/25,26		1
Stage-3				
42	011 111	27,37/28,30/ 28,33/30,33		5
43	101 111	27,41/29,31/ 29,34/31,34		4
44	111 011	28,29/28,38/ 29,38/32,35	y	5
45	111 101	30,31/30,39/ 31,39/32,36	y	4
46	111 110	33,34/33,40/ 34,40/35,36	y	4
47	110 111	37,41/38,39/ 38,40/39,40		4
Stage-4				
48	111 111	42,43/42,47/43,47/ 44,45/44,46/45,46	y	0

Table 8.5: List of minterms which are potential candidates for optimal (or near optimal) solution.

Term No	Minterm	QC
0	001 001	9
1	001 010	9
2	010 001	9
3	010 010	9
4	100 001	7
5	100 010	7
6	101 001	11
7	101 010	11
8	001 111	3
9	111 010	3
10	010 111	3
11	100 111	1
12	011 111	5
13	101 111	4
14	110 111	4

encoded form and their corresponding outputs, where n is the number of input variables of the function. The output of the algorithm is the list of potential minterms candidates for optimal solution.

8.4 Problem Encoding into Genetic Algorithm Domain

Table 8.5 shows 15 minterms which are potential candidates for minimization of the sub-function $F1(A,B)$ in Table 8.2. In our proposed GA the length of the chromosome is equal to the number of potential minterms of the sub-function to be minimized. For the case of sub-function $F1(A,B)$ in Table 8.2 the length of the chromosome is 15. The genes of the chromosome are numbered from 0. The potential minterms are also numbered from 0 (see Table 8.5). The genes corresponding to the solution of the sub-function are made 1 and other genes are made 0. Two possible solutions of the sub-function $F1(A,B)$ in Table 8.2 and their corresponding chromosome are shown in Figure 8.8. In Figure 8.8(a)

Algorithm 2 Algorithm for Min-term Generation

```
1:  procedure MIN-TERM GENERATION
2:      prevStart=0 // index of the first term of the list
3:      prevEnd=3^n − 1 // index of the last term of the list
4:      while (TRUE) do
5:          start=prevStart // index of the first term of the current Stage
6:          end=prevEnd // index of the last term of the current State
7:          currentLoc=end // index of the last generated term
8:          for (i=start; i<end; i=i + 1) do // first term at start to (end - 1)
9:              for (j=i + 1; j<= end; j=j + 1) do // second term at (i + 1) to end
10:                 if (term[i] and term[j] varies in one variable) then
11:                     newTerm=term[i] bit-wise-OR term[j]
12:                     duplicate=0
13:                     for (m=end + 1; m<=currentLoc; m=m + 1) do
14:                         if (newTerm==term[m] then // duplicate term
15:                             duplicate=1
16:                             break
17:                         end if
18:                     end for
19:                     if (duplicate==0) then // non-duplicate term. add to list
20:                         currentLoc=currentLoc + 1 // location of new term
21:                         term[currentLoc]=newTerm
22:                     end if
23:                 end if
24:             end for
25:         end for
26:         if (currentLoc==end) then // no new term generated
27:             break
28:         else   // new term generated. continue for next Stage
29:             prevStart=end + 1 // index of first term of last Stage
30:             prevEnd=currentLoc // index of last term of last Stage
31:         end if
32:     end while
33:     for (i=0; i<=end; i=i +1) do
34:         if (term[i] is less potential) then
35:             delete term[i] from list
36:         end if
37:     end for
38: end procedure
```

the solution consists of the minterms 001 010, 010 001, and 100 010 with Term No 1, 2, and 5, respectively. Therefore the genes numbered 1, 2, and 5 are made 1 and the other genes are made 0. Similarly the chromosome of Figure 8.8(b) is created.

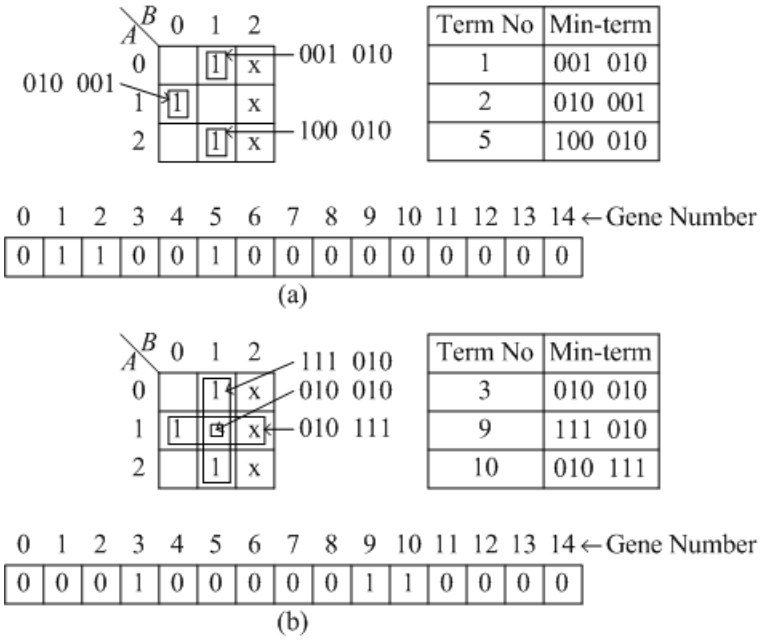

Figure 8.8: Chromosome structure of proposed Genetic Algorithm for minimization of Max-Min expression.

8.5 Proposed Hybrid Genetic Algorithm (HGA)

We have experimented with the generic GA as shown in Algorithm 1 for minimization of Max-Min expressions. The GA produces optimal solutions for some benchmark sub-functions from Appendix A; produces non-optimal solutions for some benchmark sub-function; and terminates with incorrect solutions for some benchmark sub-functions. The GA is a probability-based method and in many cases it fails to determine the correct solution. The experimental results are summarized in Table 8.6. In this table the interpretations of the column headings are as follows:

- The column Function represents the names of the ternary benchmark sub-functions from Appendix A.
- The column Variable represents the number of input variables of the sub-function.
- The column Term represents the number of minterms generated by the minterm generation program.

- The column Minterms and the column Quantum Cost under the heading Generic GA represent the number of minterms and the corresponding quantum cost generated by the generic GA.

- The column Minterms and the column Quantum Cost under the heading Optimum Solution from Table 9.1 represent the number of minterms and the corresponding quantum cost of the optimum solutions generated by the HGA from Table 9.1.

In Table 8.6 the entry $\times$ represents that the generic GA terminates with incorrect results for the corresponding benchmark sub-functions. From Table 8.6 we see the following:

- The generic GA produces optimum solutions for the benchmark sub-functions haCout1, haS0, haS1, haS2, hsCout1, hsS0, hsS1, hsS2, faS0, faS0, fsS1, and fsS2.

- The generic GA produces non-optimal solutions for the benchmark sub-functions haCout0 and hsCout0.

- The generic GA terminates with incorrect solutions for the benchmark sub-functions faCout0, faCout1, faS1, faS2, fsCout0, and fsCout1.

The generic GA does not produce optimal solutions for some benchmark sub-functions because the generic GA traps at a local optima for these sub-functions. The generic GA also terminates with incorrect solutions for some sub-functions because the generic GA does not explore a large enough search space to find correct solutions for these sub-functions.

To overcome the problem we propose a hybrid Genetic Algorithm (HGA) [9] for minimization of Max-Min expression. In a HGA a deterministic local optimization technique is used with the GA [9]. Our proposed HGA for minimization of Max-Min expression is shown in Algorithm 3.

The proposed HGA is similar to the generic GA as shown in Algorithm 1. However here the mutation operator is not used. Instead a local optimization is applied on the two offspring obtained from the crossover operation.

Table 8.6: Experimental results of generic GA-based minimization of some ternary benchmark sub-functions from Appendix A.

			Generic GA		Optimum Solution from Table 9.1	
Function	Variable	Terms	Minterms	Quantum Cost	Minterms	Quantum Cost
haCout0	2	25	13	119	3	23
haCout1	2	7	2	16	2	16
haS0	2	3	3	23	3	23
haS1	2	3	3	23	3	23
haS2	2	3	3	23	3	23
hsCout0	2	25	13	113	3	19
hsCout1	2	8	2	18	2	18
hsS0	2	3	3	23	3	23
hsS1	2	3	3	23	3	23
hsS2	2	3	3	23	3	23
faCout0	3	64	×	×	5	63
faCout1	3	64	×	×	5	57
faS0	3	11	6	82	6	82
faS1	3	11	×	×	6	82
faS2	3	11	×	×	6	82
fsCout0	3	64	×	×	5	71
fsCout1	3	64	×	×	5	59
fsS0	3	11	6	82	6	82
fsS1	3	11	6	82	6	82
fsS2	3	11	6	82	6	82

In the fitness determination procedure, we first generate the truth table of the function represented by the minterms of the chromosome. Then we compare the generated truth table with the truth table of the given function and determine the number of outputs matched (denoted by $outputMatched$). Our goal is to first realize the function by making $outputMatched = 3^n$, where n is the number of input variables in the given function, and then reduce the quantum cost of the generated circuit that realizes the function. Keeping these two goals in mind our targets are: (i) If the chromosome does not realize the given

Algorithm 3 Hybrid Genetic Algorithm for Reversible Synthesis of Ternary Logic Function using Max-Min Algebra

```
1:  procedure HYBRID GA-BASED SYNTHESIS
2:      population=initialPopulation(populationSize, chromosomeLength)
3:      for (each chromosome in the population) do
4:          determineFitness(chromosome)
5:      end for
6:      maximumFitness=determineMaximumFitness(population)
7:      noChangeCount=1
8:      while (noChangeCount ≤ maximumNoChangeCount) do
9:          parent1=binaryTournamentSelection(population)
10:         parent2=binaryTournamentSelection(population)
11:         {offspring1, offspring2}=crossover(parent1, parent2, P_C)
12:         determineFitness(offspring1)
13:         determineFitness(offspring2)
14:         offspring1=localOptimization(offspring1)
15:         offspring2=localOptimization(offspring2)
16:         duplicate1=duplicateChecking(population, offspring1)
17:         duplicate2=duplicateChecking(population, offspring2)
18:         if (duplicate1 = False) then
19:             replace(population, offspring1)
20:         end if
21:         if (duplicate2 = False) then
22:             replace(population, offspring2)
23:         end if
24:         newMaximumFitness=determineMaximumFitness(population)
25:         if (newMaximumFitness > maximumFitness) then
26:             maximumFitness=newMaximumFitness
27:             noChangeCount=1
28:         else
29:             noChangeCount=noChangeCount+1
30:         end if
31:     end while
32: end procedure
```

function, then increase $outputMatched$ and (ii) if the chromosome realizes the given function, then reduce quantum cost of the circuit. Therefore, the fitness function is defined as in (8.1), where $quantumCost$ is the quantum cost of the circuit generated by the chromo-

some. Our goal is to maximize the fitness.

$$fitness = \begin{cases} \frac{outputMatched}{3^n} & \text{if } outputMatched < 3^n \\ 1 + \frac{1}{quantumCost} & \text{if } outputMatched = 3^n \end{cases} \tag{8.1}$$

In our HGA the generations are terminated when the highest fit solution reaches a saturation condition; that is, the highest fitness value does not change for a predefined number of consecutive generations. Before beginning to generate solutions we set *noChangeCount* to 1 in line 7. If the *maximumFitnessValue* does not change in any generation, then we increment *noChangeCount* by 1 in line 29. If the *maximumFitnessValue* increases in any generation, then we reset *noChangeCount* to 1 in line 27. If *noChangeCount* reaches the predefined *maximumNoChangeCount* value, then we terminate the generations.

In our HGA we use binary tournament selection method as discussed in Section 8.2 for selecting two parents. For generating two offspring from the two parents we use one-point crossover as discussed in Section 8.2.

In the local optimization process, we deterministically identify three 1s such that if they are flipped to 0s, then the fitness of the offspring is increased. Then those three 1s are flipped, and all possible groups of three 1s are flipped. The motivation behind this technique is that a cell with a 1 must be covered by a multiple of three plus one groups. It may happen that a cell with a 1 is covered by four groups causing a higher quantum cost. It may happen that three groups can be omitted and the solution remains valid. Omission of these three groups will reduce the quantum cost and increase the fitness value. On the other hand, a cell with a 0 must be covered by a multiple of three groups. It may happen that a cell with a 0 is covered by three groups. Omission of these three groups does not change the solution. Thus, omission of these three groups will likely reduce the quantum cost and increase the fitness value. The algorithm for local optimization is shown in Algorithm 4.

The proposed HGA produces the minimum solution for the sub-function $F1(A,B)$ in Table 8.2 as shown in Figure 8.9, which requires 15 quantum cost.

Algorithm 4 Local Optimization of Offspring

```
1: procedure LOCAL OPTIMIZATION OF OFFSPRING
2:     fitnessOffspring=determineFitness(offspring) // initial fitness of the offspring
3:     for (i=0; i<(chromosomeLength - 2); i=i + 1) do // first index
4:         for (j=i + 1; j<(chromosomeLength - 1); j=j + 1) do // second index
5:             for (k=j + 1; k<chromosomeLength; k=k + 1) do // third index
6:                 if (offspring[i]==1 AND offspring[j]==1 AND offspring[k]==1) then
7:                     //three genes are 1, flip them to 0
8:                     offspring[i]=0
9:                     offspring[j]=0
10:                    offspring[k]=0
11:                    fitness=determineFitness(offspring) // fitness of new offspring
12:                    if (fitness>fitnessOffspring) then // fitness of offspring improved
13:                        fitnessOffspring=fitness // update fitness of offspring
14:                    else  // fitness of offspring not improved, flip the genes to 1 back
15:                        offspring[i]=1
16:                        offspring[j]=1
17:                        offspring[k]=1
18:                    end if
19:                end if
20:            end for
21:        end for
22:    end for
23: end procedure
```

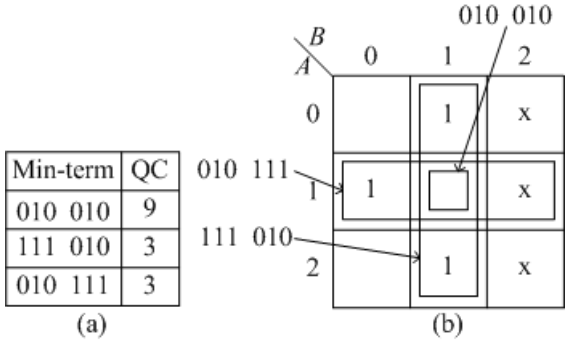

Figure 8.9: (a) Minimum solution for the sub-function $F1(A,B)$ in Table 8.2 produced by the HGA and (b) representation of the minterms of the solution on a ternary K-map.

We discuss the performance of the proposed HGA in Chapter 9.

8.6 Ternary Reversible Circuit Synthesis From Outputs of Hybrid Genetic Algorithm

8.6.1 Logic-Level Ternary Circuit Synthesis Using Multiple-Controlled Unary Gates from Output of the Hybrid Genetic Algorithm

If we combine Tables 4.4, 6.2, and the concept of value changes of a variable in a ternary K-map, then we get Table 8.7. Table 8.7 shows that the value changes of a variable are exactly the same as the control values of the multiple-controlled unary gate.

Table 8.7: Mapping of value change of a variable to literal and mapping of literal to control value of the multiple-controlled unary gate.

Value Change of a Variable x	Literal	Control Value of Multiple-Controlled Unary Gate
0	x^{+2}	0
1	x^{+1}	1
2	x^{+0}	2
01	$x^{(1+2)}$	01
02	$x^{(0+2)}$	02
12	$x^{(0+1)}$	12

As the value changes of a variable is directly shown in the output of the HGA in encoded form, the control value of the multiple-controlled unary gate can directly be determined from the output of the HGA by one-to-one mapping. The output of the HGA for the sub-function $F1(A,B)$ in Table 8.2 and its reversible realization using multiple-controlled unary gates is shown in Figure 8.10. The first minterm is 010 010. So the control value for the variable A is 1 and the control value for the variable B is 1. Therefore, the control values of the first multiple-controlled unary gate are 1 and 1, respectively. The second minterm is 111 010. As the variable A varies over 0, 1, and 2, the variable A will be missing in the minterm. The control value of the variable B is 1. Therefore, the second multiple-controlled unary gate has only one control value 1 along the input line B. The readers should note that a single-controlled unary gate is generalized as multiple-controlled unary gate with a single control. The third minterm is 010 111. The variable B varies over 0, 1, and 2; and the

value of the variable A is 1. Therefore, the third multiple-controlled unary gate has a single control value 1 along the input line A.

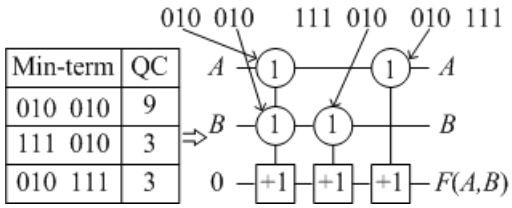

Figure 8.10: The output of the HGA for the sub-function $F1(A,B)$ in Table 8.2 and its reversible realization using multiple-controlled unary gates.

8.6.2 Post Synthesis Quantum Cost Reduction

Post synthesis quantum cost reduction aims to find adjacent elementary quantum gate pairs that cancel each other out, or can be combined to a single elementary quantum gate. This depends on the relative positions of the multiple-controlled unary gates (see Section 6.5). One example of post synthesis quantum cost reduction is illustrated in Figure 8.11 for the output generated by the HGA for the ternary benchmark (see Appendix A) sub-function 3cy20. In Figure 8.11(a) the output of the HGA for the sub-function 3cy20 and its corresponding reversible realization using multiple-controlled unary gates is shown. Quantum cost of this realization is calculated using formula (5.2) which gives us a value of 66. In the circuit of Figure 8.11(a) the control values of the first and the second multiple-controlled unary gates along the input line A are 0 and 1, respectively, and they are adjacent. These two adjacent 0 and 1 controls produce a quantum cost reduction of 1 (see Table 6.3). The control values of the fourth and the fifth multiple-controlled unary gates along the input line A are 0 and 0, respectively, and produce a quantum cost reduction of 2. The control values of the first and the second multiple-controlled unary gates along the input line B are 0 and 1, respectively, and produce a quantum cost reduction of 1. The control values of the fourth and the sixth multiple-controlled unary gates along the input line B are 0 and 0, respectively, and produce a quantum cost reduction of 2. The control values of the first and the second multiple-controlled unary gates along the input line C are 0 and 1, respectively,

and produce a quantum cost reduction of 1. The control values of the fifth and the sixth multiple-controlled unary gates along the input line C are 0 and 0, respectively, and produce a quantum cost reduction of 2. Thus, the total post synthesis quantum cost reduction is 9; and the final quantum cost of the realization is $66 - 9 = 57$.

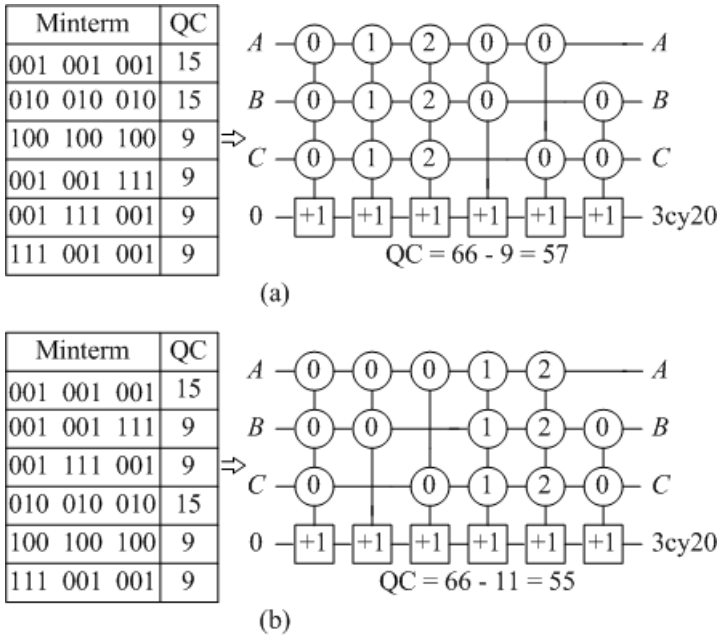

Figure 8.11: Post synthesis quantum cost reduction for ternary benchmark sub-function 3cy20.

Post synthesis quantum cost reduction depends on relative positions of the multiple-controlled unary gates. The minimum solution of 3cy20 benchmark sub-function has six minterms. An exhaustive test of maximum post synthesis quantum cost reduction requires testing of $6! = 720$ permutations of the minterms. For a large sub-function such exhaustive testing is not practical. Therefore, heuristic methods for reordering of the minterms should be developed. From Table 6.3 we see that adjacent control values 0, 0 (001, 001); 0, 01 (001, 011); 1,1 (010, 010) produce maximum quantum cost reduction of 2. If we sort the encoded minterms in ascending order, then these control values will be adjacent and contribute to more quantum cost reduction. Thus as a heuristic approach we sorted the minterms in ascending order. In Figure 8.11(b) the sorted minterms for the sub-function 3cy20 and the corresponding reversible circuit using multiple-controlled unary gates is shown. In this

case the post synthesis quantum cost reduction is 11 and the circuit realization cost becomes $66 - 11 = 55$. Thus we see that the post synthesis quantum cost reduction is improved after sorting the minterms because sorting of the encoded minterms in ascending order brings the control pairs 0, 0; 0, 01; and 1,1 to adjacent positions.

Chapter 9

Experimental Results

9.1 Introduction

In Section 8.3 we proposed an algorithm for generating minterms which are potential candidates for minimization of Max-Min expression. In Section 8.5 we proposed a hybrid genetic algorithm (HGA) for synthesis of ternary reversible circuits. In this chapter we show experimental results of ternary sub-function synthesis using the proposed HGA. In Section 6.3 we discussed architectures for reversible synthesis of a ternary logic function using any two of its three sub-functions. In Section 6.5 we discussed post synthesis quantum cost reduction of ternary reversible circuits synthesized using multiple-controlled unary gates. In Section 8.6 we proposed a heuristic to reorder the minterms in order to improve post synthesis quantum cost reduction. In this chapter we show experimental results for synthesis of ternary benchmark logic functions used in [17, 21, 27, 5] as defined in Appendix A using the proposed architectures of Section 6.3. We also show the experimental results after the post synthesis quantum cost reduction using the heuristic proposed in Section 8.6. Finally we compare our ternary reversible circuit synthesis results with those of [16] and [5].

This chapter is organized as follows:

- In Section 9.2 we show the experimental results of ternary benchmark sub-function synthesis using Algorithm 2 for minterm generation and Algorithm 3 (HGA) for synthesizing ternary benchmark sub-functions.
- In Section 9.3 we show the experimental results of synthesizing ternary benchmark

logic functions from Appendix A using the architectures proposed in Section 6.3 and after post synthesis quantum cost reduction using the heuristics proposed in Section 8.6.

- In Section 9.4 we compare our results with those of [16] and [5].

9.2 Results of Ternary Benchmark Sub-Function Synthesis Using Hybrid Genetic Algorithm (HGA)

Algorithm 2 (for generating minterms) and Algorithm 3 (the proposed HGA) are implemented using C++. The programs were compiled and run using the Microsoft Visual C++ 6.0 compiler. In Algorithm 2 the minterms are stored as an array of strings and manipulated using C++ string functions. In Algorithm 3 the population is stored as a two-dimensional array of short integers, where rows represent chromosomes and columns represent genes. The programs were run on a Personal Computer with Intel core-i7 processor, 2.4 GHz speed, 6 GB RAM, and Windows 7 Ultimate operating system.

The experimental results are summarized in Tables 9.1, 9.2, and 9.3. In these tables the interpretations of the column headings are as follows:

- The column Function represents the names of the ternary benchmark sub-functions from Appendix A.
- The column Var represents the number of input variables of the sub-function.
- The column Terms represents the number of minterms generated by the minterm generation program.
- The column Term Generation Time (sec) represents the time in sec required by the minterm generation program.
- The column Minterms or Gates represents the number of minterms selected by the proposed HGA to represent the sub-function using the minimized Max-Min expres-

sion, which is exactly equal to the number of multiple-controlled unary gates to realize the Max-Min expression (see Section 6.2).

- The column QC represents the quantum cost for realizing the resultant Max-Min expression using multiple-controlled unary gates.
- The column Number of Generations represents the number of generation required by the proposed HGA to converge to a minimum solution.
- The column HGA Time (sec) represents the time in sec required by the HGA to converge to the minimum solution.

We have experimented with 24 ternary benchmark logic functions from Appendix A. We refer to the number of minterms generated by the minterm generation program as *noTerms*. In the HGA the length of the chromosome is $chromosomeLength = noTerms$. The genetic algorithm (GA) parameters of population size, crossover probability, and maximum number of consecutive generations to test for the saturation condition each will influence the solution quality. Thus development of the GA is somewhat experimental and in practice these parameters are fine tuned through a number of experiments with different values of the parameters. Through these experiments a set of values are chosen to produce good (optimum or near optimum) solutions. After experiments we selected a population size of $10 \times noTerms$, $noChangeCount = 10 \times populationSize$ (see Algorithm 3 for termination condition of the proposed HGA), and crossover probability $P_C = 1.0$.

The HGA did not converge within 9 hours for the sub-functions mul3c0, mul3m0, prod40, 4cy20, prodMin40, sumMax40, prod50, sqsum51, and prodMin50. The reason is that the number of minterms are very high (from 158 to 9,862) and the search space is exponentially large. For example for the sub-function mul3m0 (see Table 9.2) the number of minterms is 158 and the size of the search space is $2^{158} = 3.653754093 \times 10^{47}$. In this case the HGA does not explore a reasonable portion of the search space to find a solution. The HGA converged for sub-functions prod30, prodMin30, sumMax32, sqsum50,

Table 9.1: Experimental results of minterm generation and HGA-based synthesis of ternary reversible logic circuits.

		Minterm Generation		HGA-Based Synthesis			
Function	Var	Terms	Term Generation Time (sec)	Minterms or Gates	QC	Number of Generations	HGA Time (sec)
haCout0	2	25	0.00	3	23	5,294	0.62
haCout1	2	7	0.00	2	16	1,399	0.00
haS0	2	3	0.00	3	23	599	0.00
haS1	2	3	0.00	3	23	599	0.00
haS2	2	3	0.00	3	23	599	0.00
hsCout0	2	25	0.00	3	19	5,708	0.61
hsCout1	2	8	0.00	2	18	1,606	0.02
hsS0	2	3	0.00	3	23	599	0.00
hsS1	2	3	0.00	3	23	599	0.00
hsS2	2	3	0.00	3	23	599	0.00
faCout0	3	64	0.08	5	63	38,512	42.82
faCout1	3	64	0.08	5	57	32,138	35.23
faS0	3	11	0.08	6	82	2,351	0.06
faS1	3	11	0.09	6	82	2,351	0.05
faS2	3	11	0.08	6	82	2,351	0.06
fsCout0	3	64	0.13	5	71	31,098	42.33
fsCout1	3	64	0.12	5	59	33,548	44.97
fsS0	3	11	0.13	6	82	2,351	0.09
fsS1	3	11	0.12	6	82	2,351	0.09
fsS2	3	11	0.12	6	82	2,351	0.09
prod30	3	158	0.08	3	33	33,771	299.57
prod31	3	4	0.08	4	48	799	0.00
prod32	3	4	0.08	4	48	799	0.00
sum30	3	9	0.08	9	117	1,857	0.11
sum31	3	9	0.08	9	117	1,857	0.11
sum32	3	9	0.08	9	117	1,857	0.11
3cy20	3	33	0.08	6	66	9,336	1.12
3cy21	3	6	0.08	6	78	1,199	0.02
3cy22	3	60	0.08	6	72	16,127	6.68

Table 9.2: Experimental results of minterm generation and HGA-based synthesis of ternary reversible logic circuits (continued).

		Minterm Generation		HGA-Based Synthesis			
Function	Var	Terms	Term Generation Time (sec)	Minterms or Gates	QC	Number of Generations	HGA Time (sec)
sqsum30	3	39	0.08	3	51	8,051	1.25
sqsum31	3	12	0.08	3	27	2,403	0.05
sqsum32	3	39	0.08	3	57	7,823	1.16
avg30	3	62	0.08	6	88	58,683	56.25
avg31	3	130	0.08	17	227	180,656	2,194.03
avg32	3	1	0.08	1	9	199	0.00
a2bcc0	3	25	0.08	4	56	5,981	0.49
a2bcc1	3	53	0.08	4	56	12,344	4.21
a2bcc2	3	10	0.08	4	60	2,192	0.05
mul3c0	3	264	0.08	-	-	-	-
mul3c1	3	3	0.08	3	33	599	0.00
mul3c2	3	1	0.08	1	9	199	0.00
mul3m0	3	158	0.08	-	-	-	-
mul3m1	3	4	0.08	4	48	799	0.00
mul3m2	3	4	0.08	4	48	799	0.00
prodMin30	3	158	0.10	3	33	33,771	303.85
prodMin31	3	32	0.09	3	45	7,098	1.05
prodMin32	3	1	0.09	1	9	199	0.00
sumMax30	3	1	0.08	1	15	199	0.00
sumMax31	3	32	0.08	3	57	7,788	1.05
sumMax32	3	158	0.08	3	33	35,331	285.80
3cyM20	3	31	0.08	4	42	6,530	0.63
3cym21	3	90	0.08	7	109	34,837	55.12
3cyM22	3	31	0.08	4	24	8,152	0.82
a2bccM0	3	7	0.09	1	9	1,402	0.01
a2bccM1	3	44	0.09	2	22	8,912	1.98
a2bccM2	3	102	0.08	2	12	20,569	33.80

Table 9.3: Experimental results of minterm generation and HGA-based synthesis of ternary reversible logic circuits (continued).

		Minterm Generation		HGA-Based Synthesis			
Function	Var	Terms	Term Generation Time (sec)	Minterms or Gates	QC	Number of Generations	HGA Time (sec)
prod40	4	1,280	4.40	-	-	-	-
prod41	4	8	4.48	8	136	1,599	0.09
prod42	4	8	4.51	8	136	1,599	0.09
sum40	4	27	4.65	27	495	6,071	34.73
sum41	4	27	4.65	27	495	6,071	34.65
sum42	4	27	4.65	27	495	6,071	34.78
4cy20	4	428	4.78	-	-	-	-
4cy21	4	18	4.54	18	330	3,747	5.13
4cy22	4	18	4.60	18	330	3,747	5.20
sqsum40	4	157	4.56	5	129	35,907	148.73
sqsum41	4	128	4.51	6	120	28,725	73.48
sqsum42	4	78	4.54	6	150	17,304	15.43
prodMin40	4	1,280	4.30	-	-	-	-
prodMin41	4	104	4.47	4	84	27,981	42.21
prodMin42	4	1	4.47	1	13	199	0.00
sumMax40	4	1	4.72	1	21	100	0.00
sumMax41	4	104	4.49	4	108	26,027	40.01
sumMax42	4	1,280	4.32	-	-	-	-
prod50	5	9,862	261.97	-	-	-	-
prod51	5	16	265.51	16	352	6,705	12.12
prod52	5	16	252.10	16	352	6,705	11.81
sqsum50	5	391	290.91	11	357	209,211	15,399.00
sqsum51	5	585	641.76	-	-	-	-
sqsum52	5	452	248.81	12	378	222,157	31,160.10
prodMin50	5	9,862	230.85	-	-	-	
prodMin51	5	312	242.46	5	135	123,722	3,510.83
prodMin52	5	1	253.50	1	17	199	0.00

sqsum52, and prodMin51 (see Tables 9.1, 9.2, and 9.3) which also had a high number of minterms (from 158 to 452). The GA is a probability-based method, and so the nature of these sub-functions helps the HGA to reach an optimum solution although the search space is exponentially large.

All solutions for the three and four variable sub-functions generated by the proposed HGA were manually tested using ternary K-map-based method and found to be the minimum solutions.

9.3 Results of Ternary Benchmark Circuit Synthesis

In Section 6.3 we proposed three architectures for synthesis of ternary logic functions. A ternary logic function F is realized using two of three sub-functions $F0$, $F1$, and $F2$ chosen depending on the quantum cost of the sub-functions. Consider the three-variable benchmark function sqsum3 in Table 9.2. Here $QC0 = 51$, $QC1 = 27$, and $QC2 = 57$. As $QC2 = max(QC0, QC1, QC2)$, the ancilla input constant is selected to be 2 and the sub-functions sqsum30 and sqsum31 are realized using multiple-controlled unary gates (see architecture of Figure 6.3(c)). The quantum cost of this realization is $51 + 27 = 78$. In Section 8.6 we proposed a heuristic of rearranging the minterms for better post synthesis quantum cost reduction. After post synthesis quantum cost reduction the quantum cost for reversible realization of sqsum3 function becomes 69. This realization requires multiple-controlled unary gates with three control lines. So the multiple-controlled unary gates require two ancilla inputs (see Section 5.4). Output realization requires one ancilla input. Thus three ancilla inputs are required.

Consider the benchmark function prod3 in Table 9.1. Here $QC0 = 33$, $QC1 = 48$, and $QC2 = 48$. As $QC1 = QC2 = max(QC0, QC1, QC2)$, either of the architectures of Figure 6.3(b) or (c) can be used. Consider the benchmark function sum3 in Table 9.1. Here $QC0 = QC1 = QC2 = 117$. So any of the three architectures of Figure 6.3 can be used.

Consider the benchmark function prod4 in Table 9.3. Here sub-function prod40 could

not be solved using the proposed HGA. However as sub-functions prod41 and prod42 could be solved, prod4 can be synthesized using the architecture of Figure 6.3(a), where the ancilla input constant is 0 and sub-functions prod41 and prod42 are synthesized. Other benchmark functions, where one of the three sub-functions could not be solved, can be synthesized similarly.

Reversible circuit synthesis results of 24 benchmark functions of Tables 9.1, 9.2, and 9.3 are summarized in Table 9.4. The table headings are interpreted as follows:

- The column Function represents the name of the benchmark function.
- The column Input represents the number of input variables of the benchmark function.
- The column Output represents the number of outputs of the benchmark function.
- The column Gates represents the number of multiple-controlled unary gates required to synthesize the benchmark function.
- The column Initial QC represents the quantum cost of initial synthesis.
- The column Reduced QC represents the quantum cost after post synthesis quantum cost reduction.
- The column Ancilla Inputs represents the number of ancilla inputs required to synthesize the benchmark functions.

9.4 Comparison of Circuit Synthesis Results With Previous Work

The TGFSOP-based synthesis method of reversible ternary logic circuits is claimed to be the most pragmatic method [16]. In [16] only four ternary benchmark function synthesis results were given. We have compared our synthesis results with those of [16] in Table 9.5. The table headings are interpreted as follows:

Table 9.4: Ternary reversible circuit realization results.

Function	Input	Output	Gates	Initial QC	Reduced QC	Ancilla Inputs
ha	2	2	8	62	54	3
hs	2	2	8	64	59	3
fa	3	2	17	221	184	4
fs	3	2	17	223	181	4
prod3	3	1	7	81	75	3
sum3	3	1	18	234	205	3
3cy2	3	1	12	138	121	3
sqsum3	3	1	6	78	69	3
avg3	3	1	7	97	81	3
a2bcc	3	1	8	112	100	3
mul3	3	2	12	138	136	4
prodMin3	3	1	4	42	40	3
sumMax3	3	1	4	48	43	3
3cyM2	3	1	8	66	66	3
a2bccM	3	1	3	21	19	2
prod4	4	1	16	272	248	4
sum4	4	1	54	990	830	4
4cy2	4	1	36	660	557	4
sqsum4	4	1	11	249	230	4
prodMin4	4	1	5	97	93	4
sumMax4	4	1	5	129	107	4
prod5	5	1	32	704	626	5
sqsum5	5	1	23	735	662	5
prodMin5	5	1	6	152	148	5

- The column Function represents the benchmark function name.
- The column In represents the number of inputs of the benchmark function.
- The column Out represents the number of outputs of the benchmark function.
- The columns QC and Ancilla under TGFSOP-based Method [16] represent the quantum cost (QC) and the number of ancilla inputs (Ancilla) required in TGFSOP-based method.

- The columns QC and Ancilla under Max-Min Expression-based Method represent the quantum cost (QC) and the number of ancilla inputs (Ancilla) required in our Max-Min expression-based method.

- The columns QC and Ancilla under % Reduction over TGFSOP-based Method represent the % reduction of the quantum cost (QC) and the number of ancilla inputs (Ancilla) in our method over the TGFSOP-based method. The % Reduction is calculated using the formula of (9.1).

$$\%\ \text{Reduction} = \frac{PreviousValue - OurValue}{PreviousValue} \times 100 \tag{9.1}$$

Table 9.5: Comparison of our Max-Min expression-based synthesis results with those of TGFSOP-based method [16].

			TGFSOP-based Method [16]		Max-Min Expression-based Method		% Reduction over TGFSOP-based Method	
Function	In	Out	QC	Ancilla	QC	Ancilla	QC	Ancilla
ha	2	2	44	4	54	3	-22.73	25.00
fa	3	2	636	7	184	4	71.07	42.86
hs	2	2	119	4	59	3	50.42	25.00
fs	3	2	543	8	181	4	66.67	50.00

From Table 9.5 we see that a significant reduction of quantum cost and number of ancilla inputs are achieved in our proposed method over the TGFSOP-based method, except the quantum cost for the ha benchmark function, which requires 22.73% more quantum cost than the TGFSOP-based method. In the TGFSOP-based method the sum output of the ha benchmark function is represented as $s = a \oplus b$, which allows to implement the sum output using only one ternary Feynman gate resulting in a lower quantum cost of the circuit.

A ternary Max-Min expression-based method of synthesizing reversible ternary circuits was reported in [5]. In [5] Max-Min expressions are formed using 1-reduced Post-literals and the Max-Min expressions are realized using generalized ternary gates (GTGs) (see

Section 2.8 for description of GTGs). In contrast we form our Max-Min expressions using reversible literals and non-reversible literals and the Max-Min expressions are then realized using multiple-controlled unary gates. In [5] experimental results of 33 ternary benchmark functions were reported in terms of number of GTGs and constant inputs (ancilla inputs). Quantum costs of the realizations are not reported, so we cannot compare our quantum costs with those of [5]. We compare our number of ancilla inputs with those of [5] for 14 common benchmark functions in Table 9.6. From Table 9.6 we see that our method saves a large number of ancilla inputs over [5].

Table 9.6: Comparison of number of ancilla inputs required in our method with those of [5].

Function	Input	Output	Method in [5]	Our Method	% Reduction over [5]
prod3	3	1	11	3	72.73
sum3	3	1	37	3	91.89
3cy2	3	1	21	3	85.71
sqsum3	3	1	15	3	80.00
avg3	3	1	15	3	80.00
a2bcc	3	1	14	3	78.57
prodMin3	3	1	5	3	40.00
sumMax3	3	1	9	3	66.67
3cyM2	3	1	11	3	72.73
a2bccM	3	1	13	2	84.62
prod4	4	1	28	4	85.71
sqsum4	4	1	30	4	86.67
prodMin4	4	1	7	4	42.86
sumMax4	4	1	15	4	73.33

Chapter 10

Conclusion and Future Work

10.1 Conclusion

A number of papers have been published on generalized synthesis of ternary reversible logic circuits. Among the reported works, ternary Galois field sum of products (TGFSOP) expression-based design of ternary reversible logic circuits is claimed to be the most practical method for functions with many input variables [16]. However, this method requires exponential time to generate TGFSOP expressions and the realization of TGFSOP expressions requires a high quantum cost and large number of ancilla inputs [16]. Ternary Max-Min algebra with 1-reduced Post literals is used to synthesize ternary reversible circuits using generalized ternary gates (GTGs) in [5]. This approach requires a large number of GTGs and the number of ancilla inputs required is very high. Unfortunately the quantum costs of circuit realizations are not reported in [5], and so we cannot compare our results to their work.

To address these problems we propose an alternative ternary Max-Min algebra-based method of synthesizing ternary reversible circuits. We propose a method of representing ternary logic functions using ternary Max-Min algebra, which uses ternary reversible literals and composite literals to represent minimized Max-Min expressions for ternary logic functions. For a ternary logic function F we generate three minimized Max-Min expressions for three sub-functions $F0$, $F1$, and $F2$ producing outputs 0, 1, and 2, respectively. The difference between the ternary Max-Min algebra-based method of [5] and our proposed ternary Max-Min algebra-based method is that in [5] 1-reduced post literals are used

to form ternary Max-Min expressions while in our method we use reversible and composite literals to form ternary Max-Min expressions.

For reversible realizations of Max-Min expressions we propose macro-level multiple-controlled unary gates, which are realized using elementary quantum gates such as unary and M-S gates.

We propose three architectures for realizing ternary reversible circuits using any two sub-functions; either sub-functions $F0$ and $F1$, or $F0$ and $F2$, or $F1$ and $F2$ are realized as cascades of multiple-controlled unary gates and the other output constant is used as ancilla input for realizing the two selected sub-functions.

We propose a ternary K-map-based method for minimization of Max-Min expressions for ternary logic functions with up to four variables.

Finally, we propose a hybrid genetic algorithm-based method for synthesis of ternary reversible circuits. Our process was tested with 24 ternary benchmark function with up to five variables. On average our method requires 41.36% less quantum cost and 35.72% less ancilla inputs than those of TGFSOP expression-based method of [16] for four benchmark functions common in experimental results of both the methods. Our method also requires on average 74.39% less ancilla inputs than that of [5] for 14 benchmark functions common in experimental results of both the methods.

10.2 Future Work

Although our proposed method of reversible ternary logic circuit synthesis using ternary Max-Min algebra outperformed the two previous approaches [16, 5], still there is room for improvement.

In Section 6.3, to reduce the quantum cost of the synthesized circuit, we proposed three architectures. In the proposed architectures, two of the three sub-functions $F0$, $F1$, and $F2$ of a function F are realized and the output corresponding to the other sub-function is used as the ancilla input. Discussions on correctness of the architectures are presented. However,

an end-to-end verification process would be a useful future addition.

In Section 7.4 we identified three observations for grouping cells on a ternary K-map to directly determine the minimum solution. These observations are implemented as heuristics to simplify the K-map-based minimization of Max-Min expressions. More experimentation or mathematical proofs to establish the effectiveness of these observations would be a future addition.

The search space of minterms that could form a good (optimal or near optimal) solution is very large, thus in Chapter 8, we used a hybrid Genetic Algorithm (HGA), which is a meta-heuristic algorithm, to minimize the Max-Min expressions. However, other heuristic algorithms such as 0-1 programming can also be investigated.

The algorithm for minterm generation in Section 8.3 is not optimized. There is scope to improve this algorithm.

In the proposed HGA-based minimization of Max-Min expressions, to reduce the search space, we identified some minterms to have lower potential for generating a good (optimal or near optimal) solution and excluded them from the solution generation process. More rigorous study is needed to further reduce the search space of the HGA and then ternary logic functions with more inputs can be solved.

In Section 8.5, we propose an algorithm for local optimization. The algorithm is not optimized and there is a room to redefine this algorithm in a more efficient way. In addition, further attempt can be made to develop more effective local optimization processes.

Post synthesis quantum cost reduction is very important for reducing the circuit realization complexity in terms of quantum costs. The ordering of the minterms selected by the HGA is very important for post synthesis quantum cost reduction. If the number of minterms in the minimum solution is N, then an exhaustive search requires testing $N!$ permutations of the minterms, which is not practical for large functions. In Section 8.6 we proposed a preliminary heuristic by sorting the encoded minterms in ascending order, which produced a significant post synthesis quantum cost reduction over unsorted minterms (see

Table 9.4). Still there is a huge scope for developing other heuristics for minterm ordering for additional post synthesis quantum cost reduction.

Comparison between complexities of binary reversible designs and equivalent ternary-encoded reversible designs does not yet exist in the literature, and will be an important aspect of future research.

Finally, the proposed concepts can be extended for reversible circuit synthesis of logic functions with higher radix such as quaternary logic functions (radix four) and quinary logic functions (radix five).

Bibliography

[1] W. C. Athas, L. Svensson, J. G. Koller, N. Tzartzanis, and E. Y. Chou. Low-power digital systems based on adiabatic-switching principles. *IEEE Trans. VLSIS*, 2(4):398–407, 1994.

[2] C. Bennett. Logical reversibility of computations. *IBM J. Res. Develop.*, 17(6):525–532, 1973.

[3] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Physical Review Letters*, 74:4091, 1995.

[4] E. P. DeBenedictis. The Boolean logic tax. *Computer*, 49(4):79–82, 2016.

[5] N. Denler, B. Yen, M. Perkowski, and P. Kerntopf. Synthesis of reversible circuits from a subset of Muthukrishnan-Stroud quantum multivalued gates. In *Proc. International Workshop on Logic Synthesis (IWLS 2004)*, Tamecula, California, June 2004.

[6] A. DeVos and Y. V. Rentergem. Power consumption in reversible logic addressed by a ramp voltage. In *Proc. 15th Int. Workshop Power Timing Model., Optim. Simul.*, LNCS 3728, pages 207–216, 2005.

[7] A. P. Dhande, V. T. Ingole, and V. R. Ghiye. *Ternary Digital System: Concepts and Applications*. SM Medical Technologies Private Limited, 2014.

[8] J. Dixon and B. Mortimer. *Permutation Groups*, volume 163. Springer Verlag, 1996.

[9] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2), 2006.

[10] V. R. Ghiye, A. P. Dhande, and S. H. Bonde. Ternary function minimization by map method. *International Journal of Computer Science and Electronics Engineering (IJCSEE)*, 1(5):614–620, 2013.

[11] K. H. Han and J. H. Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. Evolutionary Computation*, 6(6):580–593, 2002.

[12] A. I. Khan, N. Nusrat, S. M. Khan, M. Hasan, and M. H. A. Khan. Quantum realization of some ternary circuits using Muthukrishnan-Stroud gates. In *Proc. 37th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2007)*, page 20, Oslo, Norway, 14-16 May 2007.

[13] M. Khan and J. E. Rice. Synthesis of reversible logic functions using ternary Max-Min algebra. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1674–1677, Montreal, Canada, May 2016.

[14] M. Khan and J. E. Rice. Ternary Max-Min algebra for representation of reversible logic functions. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1670–1673, Montreal, Canada, May 2016.

[15] M. H. A. Khan. Quantum realization of multiple-valued Feynman and Toffoli gates without ancilla input. In *Proc. 39th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2009)*, pages 103–108, Naha, Okinawa, Japan, 21-23 May 2009.

[16] M. H. A. Khan. GFSOP-based ternary quantum logic synthesis. In *Proc. Optics and Photonics for Information Processing IV (SPIE Conference 7797)*, San Diego, CA, USA, August 2010.

[17] M. H. A. Khan and M. A. Perkowski. Genetic algorithm based synthesis of multi-output ternary functions using quantum cascade of generalized ternary gates. In *Proc. 2004 IEEE Congress of Evolutionary Computing (CEC)*, pages 2194–2201, Portland, Oregon, USA, June 2004.

[18] M. H. A. Khan and M. A. Perkowski. GF(4) based synthesis of quaternary reversible/quantum logic circuits. *Journal of Multiple-Valued Logic and Soft Computing*, 13:583–603, 2007.

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

[20] M. H. A. Khan, M. A. Perkowski, and P. Kerntopf. Multi-output Galois field sum of products synthesis with new quantum cascades. In *Proc. 33rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2003)*, pages 146–153, Tokyo, Japan, 16-19 May 2003.

[21] M. H. A. Khan, M. A. Perkowski, and M. R. Khan. Ternary Galois field expansions for reversible logic and Kronecker decision diagrams for ternary GFSOP minimization. In *Proc. 34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004)*, pages 58–67, Toronto, Canada, 19-22 May 2004.

[22] M. H. A. Khan, M. A. Perkowski, M. R. Khan, and P. Kerntopf. Ternary GFSOP minimization using Kronecker decision diagrams and their synthesis with quantum cascades. *Journal of Multiple-Valued Logic and Soft Computing*, 11(5-6):567–602, 2005.

[23] R. Landauer. Irreversibility and heat generation in the computation process. *IBM J. Res. Develop.*, 44:183–191, 2000.

[24] X. Li, G. Yang, and D. Zheng. Logic synthesis of ternary quantum circuits with minimal qutrits. *Journal of Computers*, 8(8):1941–1946, 2013.

[25] S. B. Mandal, A. Chakrabarti, and S. Sur-Kolay. Synthesis techniques for ternary quantum logic. In *41st IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, pages 218–223, 2011.

[26] M. Morris Mano and Michael D. Ciletti. *Digital Design With an Introduction to the Verilog HDL.* Pearson, 5th edition, 2013.

[27] `http://web.cecs.pdx.edu/~mperkows/`.

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

[29] P. Mazumder and E. M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Pearson Education Asia, 2002.

[30] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1996.

[31] D. M. Miller, G. W. Dueck, and D. Maslov. A synthesis method for MVL reversible logic. In *Proc. 34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004)*, pages 74–80, Toronto, Canada, 19-22 May 2004.

[32] D. M. Miller, D. Maslov, and G. W. Dueck. Synthesis of quantum multiple-valued circuits. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6):431–450, 2006.

[33] A. Muthukrishnan and C. R. Stroud. Multivalued logic gates for quantum computation. *Physical Review A*, 62(5):052309/1–8, 2000.

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

[35] M. Perkowski, A. Al-Rabadi, and P. Kerntopf. Multiple valued quantum logic synthesis. In *Proc. 2002 International Symposium on New Paradigm VLSI Computing*, pages 41–47, Sendai, Japan, December 2002.

[36] T. Sasao. *Logic Synthesis and Optimization*, chapter AND-EXOR Expressions and Their Optimization, pages 287–312. Kluwer Academic Publishers, 1993.

[37] T. Sasao. *Logic Synthesis and Optimization*, chapter Logic Synthesis with EXOR Gates, pages 259–286. Kluwer Academic Publishers, 1993.

[38] R. J. Spillman and S. Y. H. Su. Detection of single, stuck-type failures in multivalued combinational networks. *IEEE Transactions on Computers*, C-16(12):1242–1251, 1977.

[39] M. Yoeli and G. Rosenfeld. Logical design of ternary switching circuits. *IEEE Transactions on Electronic Computers*, EC-14(1):19–29, 1965.

Appendix A

Ternary Benchmark Functions

The following ternary benchmark functions are defined in [17, 21, 27]:
prod*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0 x_1 \cdots x_{n-1})$ mod 3. [Output is the GF3 product of n input variables.]
sum*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0 + x_1 + \cdots + x_{n-1})$ mod 3. [Output is the GF3 sum of n input variables.]
***n*cy*r*:** Input $x_0, x_1, \cdots, x_{n-1}$; output $y = [\sum_{i=0}^{n-1} + (\prod_{j=0}^{r-1} x_{(i+j) \bmod n})]$ mod 3. [A TGFSOP function of n input variables, where the product terms consist of r input variables in cyclic order. Example: 3cy2 is $y(a,b,c) = (ab + bc + ca)$ mod 3.]
sqsum*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0^2 + x_1^2 + \cdots + x_{n-1}^2)$ mod 3. [Output is the GF3 sum of squares of n input variables.]
avg*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = \text{int}[(x_0 + x_1 + \cdots + x_{n-1})/n]$ mod 3. [Output is the integer part of the average of n input variables expressed as mod 3 value.]
a2bcc: Input a, b, c; output $y = (a^2 + bc + c)$ mod 3. [An arbitrary function $a^2 + bc + c$ expressed as mod 3 value.]
thadd: Input a, b; output $c_o = \text{int}[(a+b)/3]$, $s = (a+b)$ mod 3. [Ternary half-adder.]
tfadd: Input a, b, c; output $c_o = \text{int}[(a+b+c)/3]$, $s = (a+b+c)$ mod 3. [Ternary full-adder.]
mul2: Input a, b; output $c_o = \text{int}(ab/3)$, $m = ab$ mod 3. [2-input ternary multiplier.]
mul3: Input a, b, c; output $c_o = \text{int}(abc/3)$, $m = abc$ mod 3. [3-input ternary multiplier.]
mami4: Input a, b, c, d; output $y = \max(a, b)$, $z = \min(c, d)$. [4-input arbitrary function, where the output y is maximum of the first two inputs and the output z is the minimum of the last two inputs.]
tsg: Input a, b; output b, a. [Ternary swap gate.]

In [5] the benchmark functions prod*n*, sum*n*, *n*cy*r*, sqsum*n*, avg*n*, and a2bcc are renamed as prodG*n*, sumG*n*, *n*cyG*r*, sqsumG*n*, avgG*n*, and a2bccG, respectively, to indicate that the associated operations are GF3 operations. Max-Min equivalent of these benchmark functions are defined in [5] as follows:
prodMin*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0 x_1 \cdots x_{n-1})$. [Output is the minimum of n input variables.]
sumMax*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0 + x_1 + \cdots + x_{n-1})$. [Output is the maximum of n input variables, where + is the Max operation.]
***n*cyM*r*:** Input $x_0, x_1, \cdots, x_{n-1}$; output $y = [\sum_{i=0}^{n-1} + (\prod_{j=0}^{r-1} x_{(i+j) \bmod n})]$. [A Max-Min function of n input variables, where the minterms consist of r input variables in cyclic order.

Example: 3cyM2 is $y(a,b,c) = (ab+bc+ca)$, where the product is the Min operation and the + is the Max operation.]

sqsumM*n*: Input $x_0, x_1, \cdots, x_{n-1}$; output $y = (x_0^2 + x_1^2 + \cdots + x_{n-1}^2)$. [Output is the Max of squares of n input variables. sqsumMn is equivalent to sumMax$n$, since $x^2 = x \cdot x = \min(x,x) = x$.]

a2bccM: Input a,b,c; output $y = (a^2 + bc + c)$. [An arbitrary function $a^2 + bc + c$ expressed as Max-Min expression.]