

ONLINE TESTING IN TERNARY REVERSIBLE LOGIC

MD. RAQIBUR RAHMAN
Bachelor of Science, University of Dhaka, 2004
Master of Science, University of Dhaka, 2005

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

© Md. Raqibur Rahman, 2011

I dedicate this thesis to my family, without whose enormous support, encouragement and sacrifice, I could not reach my destination.

Abstract

In recent years ternary reversible logic has caught the attention of researchers because of its enormous potential in different fields, in particular quantum computing. It is desirable that any future reversible technology should be fault tolerant and have low power consumption; hence developing testing techniques in this area is of great importance.

In this work we propose a design for an online testable ternary reversible circuit. The proposed design can implement almost all of the ternary logic operations and is also capable of testing the reversible ternary network in real time (online). The error detection unit is also constructed in a reversible manner, which results in an overall circuit which meets the requirements of reversible computing. We have also proposed an upgrade of the initial design to make the design more optimized. Several ternary benchmark circuits have been implemented using the proposed approaches. The number of gates required to implement the benchmarks for each approach have also been compared. To our knowledge this is the first such circuit in ternary with integrated online testability feature.

Acknowledgments

I would like to take the opportunity to express my sincere gratitude to my respected supervisor Dr. Jacqueline E. Rice. Throughout this research work she has provided not only enormous time and effort but also proper guidance in the time of difficulties whenever it became intricate to reach the solution.

I was fortunate to have Dr. Howard Cheng and Dr. Sajjad Zahir in my supervisory committee. I would like to thank my committee members for their excellent and insightful advises, reading of the thesis and providing valuable feedbacks.

I must mention the tremendous support and inspiration of my well-wisher colleague Noor Nayeem during my research work.

I am also thankful to Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this research.

Last but not least, I must mention my family, who have been on my path to give their shoulder whenever I need, or even whenever I don't.

Contents

Approval/Signature Page	ii
Dedication	iii
Abstract	iv
Acknowledgments	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
1 Introduction	1
1.1 General Introduction	1
1.2 Road map	3
2 Background	4
2.1 Reversible logic	4
2.1.1 Importance of reversible logic	5
2.2 Binary reversible logic gates	6
2.2.1 Garbage Outputs	7
2.3 Fault Models	8
2.4 Multi-Valued Logic (MVL)	9
2.5 MVL operators	10
2.6 Applications of MVL	12
2.6.1 MVL in Logic Synthesis	12
2.6.2 Logic simulation	13
2.6.3 Digital hardware testing	13
2.6.4 MVL circuit design	13
2.6.5 MVL in quantum logic	14
2.7 Ternary reversible logic	14
2.7.1 Ternary Galois field logic (GF3)	14
2.8 Ternary reversible gates	15
2.8.1 1-qutrit permutative gate	16
2.8.2 Muthukrishnan-Stroud (M-S) Gate	17
2.8.3 Ternary Shift Gates	18
2.8.4 Ternary Feynman gate	19
2.8.5 Ternary Toffoli gate	21
2.8.6 Ternary controlled-controlled gate	23

2.8.7	3-qutrit generalized Toffoli gate (GTG)	24
2.8.8	Construction of ternary reversible circuits	25
2.9	Online Testing in Reversible Logic Circuits	26
2.10	Online Testing	26
2.11	Works related to online testable reversible logic gates	27
2.11.1	Binary online testing using R, R1 and R2 gates (approach 1)	27
2.11.2	Testing using a universal reversible logic gate (approach 2)	29
2.12	Summary	31
3	Ternary Online Testable Circuits	33
3.1	Introduction	33
3.2	Design of the online testable ternary reversible logic block	33
3.2.1	The <i>TR1</i> block	35
3.2.2	The <i>TR2</i> block	38
3.2.3	The online testable block	39
3.3	Summary	43
4	Ternary Two-Pair Two-Rail Checker	44
4.1	Introduction	44
4.2	Designing the ternary rail checker circuit	44
4.2.1	Implementation of the ternary rail checker	45
4.2.2	Rejection of this design	46
4.3	New design of the ternary rail checker	46
4.3.1	Principle of the ternary rail checker	46
4.3.2	Elementary E gate	47
4.3.3	Architecture of the ternary rail checker circuit	49
4.4	Sample Design	51
4.5	Summary	52
5	Implementation and Improvement	53
5.1	Introduction	53
5.2	Internal designs	53
5.2.1	The <i>TR1</i> block	53
5.2.2	The <i>TR2</i> block	56
5.3	Upgraded design of the <i>TR1</i> block	57
5.4	The <i>TR</i> block	59
5.5	Limitation of the <i>4TR</i> block	60
5.5.1	Implementing copy function using the <i>4TR</i> block	60
5.6	Reducing the cost of the copy function	61
5.6.1	Method 1: 5×5 TR block (<i>5TR</i>)	61
5.6.2	Method 2: Combination of <i>4TR</i> block and non-testable ternary Feynman gate	66

5.6.3	Method 3: Combination of 5TR block and online testable Copy gate (TR_c)	66
5.6.4	Method 4: Combination of 5TR block, TR_c and online testable Multi Copy gate (TR_{mc})	68
5.7	Summary	70
6	Comparison and Analysis of Different Approaches	71
6.1	Introduction	71
6.2	Comparison among the proposed methods and non testable circuits	71
6.3	Overhead analysis	74
6.4	Summary	75
7	Conclusion and Future Works	76
7.1	Conclusion	76
7.2	Future Works	77
A	Appendix	79
A.1	Notation	79
A.2	3CyG2	80
A.2.1	Method 0	80
A.2.2	Method 1	81
A.2.3	Method 2	81
A.2.4	Method 3	82
A.2.5	Method 4	82
A.2.6	Non-testable ternary gates	83
A.3	ProdG3	83
A.3.1	Method 0	83
A.3.2	Method 1	84
A.3.3	Method 2	84
A.3.4	Method 3	85
A.3.5	Method 4	85
A.3.6	Non-testable ternary gates	86
A.4	SumG3	86
A.4.1	Method 0	86
A.4.2	Method 1	87
A.4.3	Method 2	87
A.4.4	Method 3	88
A.4.5	Method 4	88
A.4.6	Non-testable ternary gates	89
	Bibliography	90

List of Tables

2.1	Two-place MVL operators.	10
2.2	Truth table for two-place MVL operator for $p = 2, 3$	11
2.3	One-place MVL operators [23].	12
2.4	Truth table for one-place operator for $p = 2, 3$ [23].	12
2.5	Ternary Galois Field (GF3) operations [14].	15
2.6	Operations of 1-qutrit permutative gates, also referred to Z-transformations [11]. 16	
2.7	Operations of the M-S gate for $A = 2$ [11].	18
3.1	A subset of the truth tables for (a) the <i>TR1</i> block and (b) the <i>TR2</i> block.	41
4.1	Combination of fault-free inputs and outputs for the ternary rail checker.	45
4.2	Truth table and transformation table of the E gates.	48
4.3	Truth table for the possible input states and the corresponding outputs of E_a and E_b	51
5.1	Truth table of AB	54
5.2	Cost of the ternary gates used to realize the proposed blocks.	57
5.3	A subset of the truth tables for (a) <i>5TR1</i> Block and (b) <i>5TR2</i> Block.	63
5.4	Truth table for the MF gate.	64
5.5	Number of M-S gates required to implement the proposed blocks	70
6.1	Comparison of number of M-S gates for implementing the ternary bench- mark circuits.	72
6.2	Overhead comparison.	74

List of Figures

2.1	Binary Feynman gate.	6
2.2	(a) Binary Toffoli gate, and (b) the truth table of the binary Toffoli gate. . .	7
2.3	The ternary 1-qutrit permutative gate. (a) The commonly-used symbol as shown in [14], (b) and the cascading operations.	17
2.4	Symbol for the ternary Muthukrishnan-Stroud (M-S) gate [14].	18
2.5	1-qutrit ternary shift gates [12].	19
2.6	(a) The commonly-used symbol of the ternary 2-qutrit Feynman gate, and (b) the truth table [14].	20
2.7	(a) The 2-qutrit copy Feynman gate, and (b) the 2-qutrit basic Feynman gate [15].	20
2.8	The 3-qutrit ternary Toffoli gate, (a) with I/O mapping as shown in [14], (b) its symbol, and (c) the truth table.	22
2.9	The realization of a 3-qutrit ternary Toffoli gate using M-S gates [14]. . . .	23
2.10	Ternary controlled-controlled gate as shown in [11].	24
2.11	(a) Symbol of the generalized Toffoli gate, (b) its block diagram [15], and (c) its realization using the M-S gate [14].	24
2.12	Cascading gates to construct a ternary reversible circuit implementing $a \oplus b \oplus c$	25
2.13	(a) The R gate, (b) R1 gate, and (c) the R2 gate [36].	27
2.14	Testable logic block (TLB) [36].	28
2.15	Rail checker circuit using the R gate [36].	29
2.16	G gate and Deduced reversible logic gate DRG (G) as in [19].	30
2.17	Cascaded DRGs to form a TRC [19].	30
2.18	Testable Reversible Circuit (TRC) [19].	31
2.19	Test Cell (TC) [19].	31
3.1	Configuration of the <i>TR1</i> block.	35
3.2	AND operation in the <i>TR1</i> block and the corresponding truth table.	36
3.3	Mod-sum operation in the <i>TR1</i> block and the corresponding truth table. . .	36
3.4	The successor operation in the <i>TR1</i> block and the corresponding truth table.	37
3.5	The negation/complement operation in the <i>TR1</i> block and the corresponding truth table.	37
3.6	The mod-difference operation in the <i>TR1</i> block and the corresponding truth table.	38
3.7	The <i>TR2</i> block.	39
3.8	Configuration of the online testable ternary reversible block (TR).	39
3.9	Online testable ternary reversible block (TR).	39
3.10	(a) Fault-free operation, and (b) faulty operation of a circuit constructed from the <i>TR</i> block.	42
4.1	Ternary two-pair two-rail checker.	45
4.2	Block diagram for the new two-pair two-rail checker.	47

4.3	E gate.	48
4.4	Internal structure of the E gates (a) E_a , and (b) E_b	49
4.5	Internal architecture of the two-pair two-rail checker.	50
4.6	Online testable ternary reversible implementation of function $ab \oplus cd$	52
5.1	Internal diagram of the $TR1$ block.	53
5.2	Realization of the cascade of GTGs having output AB	55
5.3	Internal diagram of the $TR2$ Block.	56
5.4	Realization of the $TR2$ Block.	56
5.5	Upgraded internal design of the $TR1$ block.	58
5.6	Further upgraded design of the $TR1$ block.	58
5.7	Realization of the $TR1$ block.	59
5.8	Internal diagram of the TR block.	59
5.9	Realization of the TR block.	60
5.10	Copy operation using the $4TR$ block.	61
5.11	5×5 $TR1$ block.	62
5.12	Internal design of 5×5 $TR1$ block.	62
5.13	(a) Internal design of the modified Feynman (MF) gate and (b) symbol of the modified Feynman (MF) gate.	64
5.14	Internal design of the 5×5 $TR1$ block using the MF gate.	65
5.15	$5TR2$ block.	65
5.16	Internal design of the $5TR2$ block.	65
5.17	5×5 TR ($5TR$) block.	66
5.18	5×5 TR_c block.	67
5.19	Internal design of the 5×5 TR_c block.	67
5.20	5×5 TR_{mc} block.	69
5.21	Internal design of 5×5 TR_{mc} block.	69
6.1	Area overhead chart [25]	75
A.1	(a) Copy using $4TR$ and (b) notation used in the diagrams.	79
A.2	Realization of the benchmark $3CyG2$ using $4TR$ blocks.	80
A.3	Realization of the benchmark $3CyG2$ using $5TR$ blocks.	81
A.4	Realization of the benchmark $3CyG2$ using $4TR$ blocks and Feynman gates.	81
A.5	Realization of the benchmark $3CyG2$ using $5TR$ and TR_c blocks.	82
A.6	Realization of the benchmark $3CyG2$ using $5TR$, TR_c and TR_{mc} blocks.	82
A.7	Realization of the benchmark $3CyG2$ using non-testable ternary gates.	83
A.8	Realization of the benchmark $ProdG3$ using $4TR$ blocks.	83
A.9	Realization of the benchmark $ProdG3$ using $5TR$ blocks.	84
A.10	Realization of the benchmark $ProdG3$ using $4TR$ blocks and Feynman gates.	84
A.11	Realization of the benchmark $ProdG3$ using $5TR$ and TR_c blocks.	85
A.12	Realization of the benchmark $ProdG3$ using $5TR$, TR_c and TR_{mc} blocks.	85
A.13	Realization of the benchmark $ProdG3$ using non-testable ternary gates.	86

A.14	Realization of the benchmark <i>SumG3</i> using <i>4TR</i> blocks.	86
A.15	Realization of the benchmark <i>SumG3</i> using <i>5TR</i> blocks.	87
A.16	Realization of the benchmark <i>SumG3</i> using <i>4TR</i> blocks and Feynman gates. 87	
A.17	Realization of the benchmark <i>SumG3</i> using <i>5TR</i> and TR_c blocks.	88
A.18	Realization of the benchmark <i>SumG3</i> using <i>5TR</i> , TR_c and TR_{mc} blocks. . .	88
A.19	Realization of the benchmark <i>SumG3</i> using non-testable ternary gates. . .	89

Chapter 1

Introduction

1.1 General Introduction

The irreversible nature of today's circuits leads to the loss of information in the form of heat dissipation. The technical advancement of circuits in size reduction and integration of more and more complexity may soon be saturated because of this problem. Zhirnov *et al.* in [37] showed that existing Complementary Metal Oxide Semiconductor (CMOS) technology will soon reach a point when speeding up CMOS devices will be almost impossible because of the problem of heat removal. Although modern technologies in circuit design and implementation reduce the dissipation of heat dramatically [21], the best way to reduce heat dissipation is through reversible computing which recovers energy by conserving information when performing logic, storage and communication operations using reversible transformations [6]. This has been known since the 1960s, when Landauer [18] and later Bennett [1] both proved that reversible logic would be necessary for lower power dissipation in circuits. A gate (or circuit) is reversible whenever there exists a one-to-one mapping between the input values and the output values; hence there is no loss of information. Bennett's theorem [1] also suggests that in order for any future technologies to be reliable they must be reversible and this phenomenon holds true whether the logic model used is two-valued or multi-valued.

While many of us are familiar with Boolean (binary, two-valued) logic, both in traditional (irreversible) circuit design and reversible models, there are many advantages to moving to a multi-valued (MV) paradigm, or at least considering problems using a less restricted model. Circuits can sometimes show better characteristics if multi-valued logic (MVL) concepts are used instead of traditional binary (two-valued) concepts [6]. In this

thesis we focus on a three-valued logic system known as ternary logic [23]. Using a p -valued model ($p > 2$) sometimes provides a simpler solution for many existing problems [23]. Much research is taking place in MVL synthesis and optimization techniques; however the areas of modeling faults in ternary reversible circuits and real-time online fault detection have yet to be addressed. There is a great deal of motivation to do so, in particular since quantum computing is rapidly emerging as an area of importance. Since quantum logic operations are reversible, research on testing of ternary reversible logic has immense significance.

The unit of information in quantum logic system is known as a quantum digit or qudit. This unit of information is known as quantum ternary digit or qutrit in quantum ternary logic [11]. A set of measurable quantum states of an object represents the ternary logic values 0, 1 and 2 which are encoded into computational basis states to represent a qutrit. In simple words we can say that the ternary logic has three different states of signal as its input signal whereas binary has only two states [11].

In this thesis we focus on incorporating online testability in ternary logic circuits. Online testability is the ability of a circuit to test a portion of the circuit while the circuit is operating [36]. Being unable to find any previous work on ternary online testability, our research is conducted in two directions, universal testable design and variations. Our research includes designing ternary reversible logic blocks that can be combined to implement most basic ternary logic functions and are also capable of testing the functions while the circuit is under operation (online testing). To propagate the test result in a larger circuit we also design a ternary rail checker circuit. We apply the novel blocks to implement some ternary benchmark circuits. While applying our proposed novel blocks to implement the benchmark circuits we also introduce designs for the logic blocks that have been enhanced to improve their performance.

1.2 Road map

The presented work is organized as follows:

In the *Background* chapter, the classic objects of reversible logic theory, MVL, fault models and ternary reversible gates used in this work are discussed. With this introductory chapter the reader should become familiar with the objectives and notations of MVL, fault modeling, online testing and different ternary reversible gates.

Chapter 3, *Ternary Online Testable Circuits* describes in detail the proposed ternary online testable blocks and their internal design. This work has been published in [33].

Chapter 4, *Two Pair Two Rail Checker Circuit* discusses the design of a ternary rail checker circuit and its implementation. A realization of a sample function using the online testable blocks and the rail checker is also shown. This chapter has been published in [33].

The fifth chapter, *Implementation and Improvement of Underlying Design*, discusses the gate level implementation of the online testable blocks. An improvement of the proposed design is also suggested. A part of this chapter has been published in [32].

Chapter 6, *Comparison of Different Approaches* introduces some new designs with improved efficiency. Eight benchmark circuits have been implemented using all of the proposed designs and also using existing non-testable gates. A comparison among the designs in terms of the number of gates is also discussed.

The thesis concludes with the overall discussion in chapter 7 *Conclusion and Future Works* which highlights the major contributions of the work described and suggests new directions of research in the ternary reversible testing area.

Chapter 2

Background

2.1 Reversible logic

In traditional logic the number of inputs to a circuit may be different from the number of outputs. In many cases the variation is quite large. Usually the number of inputs is greater than the number of outputs. For example, in a two-bit binary full adder the number of inputs is three but the number of outputs is always two. As stated in [34] conservation of encoded input information at the output is often necessary in many fields of computation such as digital signal processing, communication and computer graphics. There are also other issues such as energy conservation and shifting to quantum technologies, both of which require the input signals to be restored at the outputs [34]. This concept of preserving input signals leads us to the concept of reversible logic.

Definition 2.1.1 “A gate is reversible if the (Boolean) function it computes is bijective” [34].

As stated in [34], a reversible gate must have an equal number of inputs and outputs. If the gate has k inputs and k outputs and there exists a one-to-one mapping between the inputs and outputs, the circuit can be defined to be a $k \times k$ reversible gate. However, when a reversible gate has a one-to-one mapping between the inputs and outputs, this may result in generating some outputs that are not required for the computation. These outputs are known as garbage outputs. Constant inputs are input lines that are set to a constant value. To keep the number of inputs equal to the number of outputs, introducing constant inputs and garbage outputs is sometimes unavoidable.

Logic synthesis can be considered to be the process of converting a high-level design description into an optimized gate level representation of a circuit [28]. A logic synthesis process using reversible gates should have the following features [7, 36]:

- The number of gates required in the resulting circuit should be as small as possible,
- the number of input constants should be as low as possible,
- the number of gates in the cascade should be the least possible, and
- the generation of garbage outputs should be as few as possible,

Bennett suggests that for a computation to be reversible two conditions are needed to be satisfied [1]. These conditions are logical reversibility and physical reversibility. Logical reversibility refers to the ability to reconstruct the input information from the logical outputs. An irreversible computer can be made reversible by saving the information the irreversible computer usually throws away. Therefore, all computation can be made logically reversible if all the information in that computation is preserved [1]. Physical reversibility refers to the ability of the device to run backward. This reversibility implies that the circuit is thermodynamically reversible. In other words, each operation will not convert any energy to heat. The objective of the thermodynamic design is reduce the energy dissipation and achieve thermodynamic equilibrium [1].

The ability to uniquely reconstruct the input sequence from the output sequence makes the circuit reversible, as this is not possible in irreversible circuits. In [36], the authors refer to a reversible circuit as being balanced as exactly half of the inputs produce ones at the outputs.

2.1.1 Importance of reversible logic

The primary motivation towards reversible computing is based on the fact that all irreversible logic operations produce a fundamental amount of waste energy. This problem will soon cause the current technological advancement to hit a roadblock in reducing the size of components in order to continue to meet the predictions of Moore's Law. To proceed

beyond this situation energy preserving and recovering techniques will be required [32]. Details are discussed in [1, 6, 18].

The other reason for carrying out research on reversible computing is its association with quantum gates since quantum gates are reversible [34]. All reversible gates follow the same rules independent of the input types, *i.e.* whether they are classical bits or quantum states. The major attraction of quantum computing is the speed of operation but quantum computing can also be thought of as a distinct case of classical reversible computing. However in order to use reversible circuits some form of reversible logic synthesis is necessary [34].

2.2 Binary reversible logic gates

There are several binary reversible logic gates discussed in [5, 7, 35]. In this section some of the popular binary reversible gates are described.

Definition 2.2.1 A 2×2 binary Feynman gate can be defined as $I = (A, B)$ and $O = (P = A, Q = A \oplus B)$ where I and O are the sets of inputs and outputs respectively [5].

Figure 2.1 illustrates the binary Feynman gate as shown in [29].

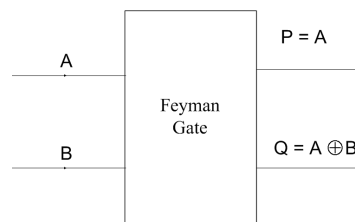


Figure 2.1: Binary Feynman gate.

Definition 2.2.2 A 3×3 binary Toffoli gate can be defined as $I = (A, B, C)$ and $O = (P = A, Q = B, R = AB \oplus C)$ where I and O are the sets of inputs and outputs respectively [35].

Figure 2.2 (a) illustrates the binary Toffoli gate as shown in [29] and Figure 2.2 (b) shows the truth table of the binary Toffoli gate as shown in [36]. Other binary reversible gates such as the Fredkin gate and the Peres gate are discussed in [29].

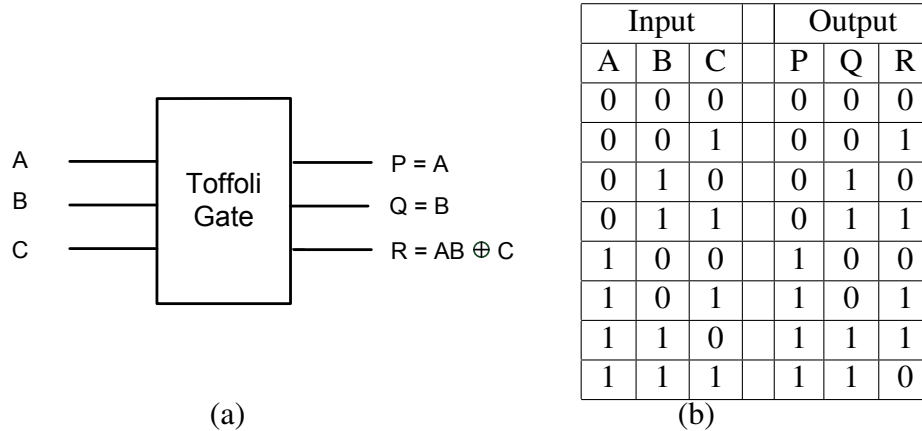


Figure 2.2: (a) Binary Toffoli gate, and (b) the truth table of the binary Toffoli gate.

2.2.1 Garbage Outputs

The binary Toffoli gate generates three outputs among which only one output (R) produces the desired result. The other two outputs may be regarded as garbage outputs if they cannot be used in computing the desired function. The presence of garbage outputs, however, is sometimes necessary in a reversible circuit to maintain the reversibility. Garbage is defined in [21] as the number of unutilized outputs required to convert a nonreversible function to a reversible one.

2.3 Fault Models

Recognizing and modeling the behavior of possible defects in a circuit is known as fault modeling. Jha and Gupta discuss how different levels of abstraction in circuit design can result in different fault models [8]. We briefly describe these varying models.

- Viewing a system from a behavioral point of view results in a variety of behavioral fault models. These types of fault models stem from the constructs used in a behavioral description, such as language like VHDL (VHSIC hardware description language; VHSIC: very-high-speed integrated circuit).
- Functional fault models focus on the faults in functional blocks of the system, with the goal of ensuring fault-free behavior of the blocks. In RAM, for instance, a multiple write fault is an example of a functional fault.
- Structural fault models deal with faults that occur at the interconnections of a design. The single stuck-at fault is a commonly used fault model in this category.
- Switch level fault models focus on the transistor level of a circuit. Two popular fault models in this category are stuck-open and stuck-on fault models. A permanently non-conducting faulty transistor generates a stuck-open fault while a permanently conducting faulty transistor results in a stuck-on fault.
- Geometric fault models identify the flaws in the layout of a chip, or manufacturer defects. Shorted lines in a chip layout are modeled by the bridging fault model which falls into this category.

Vasudevan *et al.* in [36] characterize the type of fault that our technique identifies as a *single bit error*. A single bit error occurs whenever the value of an output has been changed because of an internal circuit error, and is reflected on a single output. This is referring to a block of output values, within which the change on any one output line can be identified.

2.4 Multi-Valued Logic (MVL)

Multi-valued logic (MVL) refers to the logic system which utilizes variables that can take on a discrete and finite set of values [23]. For example, the binary logic system deals with values 0 and 1, whereas a ternary system deals with values 0,1 and 2. Besides encoding logic values, MVL methods may also be used to design the initial logic circuits with binary or MVL signal levels. A circuit may show better characteristics such as better performance or reduced wiring congestion if MVL concepts are used in the design phase instead of binary valued logic [23].

MVL requires different algebraic structures than ordinary Boolean algebra required for binary operations. Many theorems of Boolean algebra do not support a set of values that does not have cardinality of 2^n . This situation creates a lack of completeness in Boolean algebra for MVL. The behavior of a circuit is often described by mathematical equations or formulas. Another requirement for the circuit implementation from the mathematical definitions or formulas should be an easy but efficient circuit implementation of the algebraic operators [23]. A finite set of values including two identity elements and a set of operators are the basic elements of the algebra for MVL. These identity elements and operators are well defined over the finite set in one-place and two-place functions. We use the term one-place to refer to these functions that have one operand and the term two-place to refer functions that have two operands. Usually the finite sets of logic values have cardinality $p \geq 3$ in an MVL system [23].

MVL, particularly three and four valued systems, has been of interest to researchers because of connections to ternary digital circuits and computers. This is because it has become possible to implement the ternary logic system in computer hardware [3].

2.5 MVL operators

As stated earlier, a circuit's behavior can be defined using algebraic equations. The main focus of an MVL algebra is for the operators to have simple circuit implementations. A circuit developed from such operators will have fewer gates, and can implement general p -valued functions. Table 2.1 shows five such operators and their definitions as stated in [23]. Table 2.2 defines the operators for $p = 2, 3$ as stated in [23].

Table 2.1: Two-place MVL operators.

NAME	NOTATION	DEFINITION
Min	$x.y$	x if $x < y$, y otherwise
Max	$x + y$	x if $x > y$, y otherwise
Mod-sum	$x \oplus y$	$(x + y) \bmod p$
Mod-difference	$x \ominus y$	$(x - y) \bmod p$
Truncated sum	$x +_t y$	$\min(p - 1, \text{sum}(x, y))$

Table 2.2: Truth table for two-place MVL operator for $p = 2, 3$

	$p = 2$	$p = 3$
Min	\cdot 0 1	\cdot 0 1 2
	0 0 0	0 0 0 0
	1 0 1	1 0 1 1
Max	$+$ 0 1	$+$ 0 1 2
	0 0 1	0 0 1 2
	1 1 1	1 1 1 2
Mod-sum	\oplus 0 1	\oplus 0 1 2
	0 0 1	0 0 1 2
	1 1 0	1 1 2 0
Mod-difference	\ominus 0 1	\ominus 0 1 2
	0 0 1	0 0 2 1
	1 1 0	1 1 0 2
Truncated sum	$+_t$ 0 1	$+_t$ 0 1 2
	0 0 1	0 0 1 2
	1 1 1	1 1 2 2
		2 2 2 2

The number of one-place functions for a p -valued logic system is p^p . For binary ($p = 2$) there are four one-place functions: the identity function, negation, and two constant functions. For ternary systems ($p = 3$), there are 27 one-place functions including the identity, 3 constant functions and 23 other options for one-place operators. Table 2.3 shows some of the one-place operators as shown in [23]. Table 2.4 also shows the truth table of the operators for $p = 2, 3$ as shown in [23].

Table 2.3: One-place MVL operators [23].

NAME	NOTATION	DEFINITION
Cycle	x^k	$(x+k) \bmod p$
Successor	\vec{x}	$(x+1) \bmod p$
Predecessor	\overleftarrow{x}	$(x-1) \bmod p$
Negation	\bar{x}	$(p-1)-x$

Table 2.4: Truth table for one-place operator for $p = 2, 3$ [23].

P=2	x			\bar{x}		
	0			1		
	1			0		
P=3	x	x^1	x^2	\vec{x}	\overleftarrow{x}	\bar{x}
	0	1	2	1	2	2
	1	2	0	2	0	1
	2	0	1	0	1	0

There also exist MVL algebras based on logic operations for non-modular arithmetic operations over finite set of logic values; for example Lukasiewicz logic, Post logic algebra, Bochvar Logic, Kleene logic, Allen and Givone algebra [23]. These are out of the scope of this work, and will not be discussed.

2.6 Applications of MVL

The use of MVL concepts in the design stage of various fields of logic design can be clearly demonstrated from the examples in this section.

2.6.1 MVL in Logic Synthesis

A common tabular method to minimize binary sum of product (SOP) expressions is the Quine-McClusky method. When this method is used in a computer algorithm, a don't-care

value, X, in addition to logic values 0 and 1 is introduced. This new logic value converts the system into a three valued or ternary logic system consisting of {0, 1, X} [23].

2.6.2 Logic simulation

In some applications, a MVL set is used to represent different states of the system. For example, the Test Generation and Simulation System (TEGAS) uses a six-valued MVL set consisting of values {0, 1, X, U, D, E} [23]. Here {0,1} represent steady state logic values, X represents a simulator initialization value, U represents a transition signal from low to high, D represents a transition signal from high to low, and E represents transient behavior of the system [23].

2.6.3 Digital hardware testing

There exist different MVL algebras for detecting the presence of faults in a circuit. The process for detecting faults is to develop a subset of input assignment values and apply them on the circuit to determine the fault inside. One example is D-calculus algebra which is discussed in [23].

2.6.4 MVL circuit design

Decreasing wiring congestion in the circuit is a major concern in circuit design. For this purpose, MVL has great importance in circuit design. In MVL, it is possible to transmit more than two voltage or current values using a single conductor. This results in reduced wiring in the circuit [23].

2.6.5 MVL in quantum logic

Quantum computing uses the state of matter at the molecular, atomic or particle level to represent information. Usually a quantum algorithm involves more than two logic levels and hence it is easier to solve quantum problems with a MVL system than with a classical 2-valued (Boolean) system [23].

2.7 Ternary reversible logic

Ternary reversible/quantum research is introduced in [11] as a very new research area, but with motivation based on both quantum and multi-valued computing. In [12], the author discusses the fact that ternary functions can be easily expressed using Ternary Galois Field Sum of Product (TGFSOP) expressions, regardless of the number of input variables. Quantum 1-qutrit and 2-qutrit gates are also realizable using existing quantum technology [12].

Ternary logic, which deals with three different logic values, is the simplest introduction of MVL. The memory unit in ternary logic is referred to as a *qutrit* (quantum ternary digit). Qutrits may have logic values 0,1 or 2. In an implementation of a ternary reversible system these logic values may be represented by different distinguishable states of a photon's polarization, or atomic spin [13].

2.7.1 Ternary Galois field logic (GF3)

Different works on Galois field logic show that Ternary Galois Field Sum of Products (TGFSOP) expressions can be helpful in ternary quantum logic synthesis [15, 10, 14]. Addition (\oplus) and multiplication (\cdot) are the two principle mathematical operations in Ternary Galois Field (GF3) logic. GF3 operates on three basic input literals, $I = \{0, 1, 2\}$. GF3 operations are exactly the same as the modulo 3 operations [14, 15]. The operations are

defined in [14] as given in Table 2.5.

Table 2.5: Ternary Galois Field (GF3) operations [14].

\oplus	0	1	2	\cdot	0	1	2
0	0	1	2	0	0	0	0
1	1	2	0	1	0	1	2
2	2	0	1	2	0	2	1

A TGFSOP representation is mentioned as a natural choice for multi-valued quantum logic synthesis in [15]. The author in [15] states that realization using cascades of quantum gates can directly implement TGFSOP expressions, or at least this type of expression is suitable for a factorization process which leads to factorized cascades of quantum gates. Therefore, the TGFSOP representation is considered to be the most suitable representation for ternary functions [15]. In this thesis we use “ \oplus ” to refer to GF3 addition, and the absence of any symbol to refer to GF3 multiplication.

2.8 Ternary reversible gates

Some common ternary reversible gates are the ternary implementation of the binary reversible gates such as the Feynman and Toffoli gates [5, 35]. In addition, Muthukrishnan and Stroud demonstrated the realization of MVL for quantum computing using quantum technologies such as ion-trap [24], which led to their proposal of the Muthukrishnan and Stroud (M-S) gate. The M-S gate is considered to be one of the elementary gates in ternary logic synthesis [13]. A brief description of these reversible gates is given in the following sections, as well as an introduction to the most basic ternary gates such as the 1-qutrit permutative gate.

2.8.1 1-qutrit permutative gate

1-qutrit permutative gates are discussed in [11] and [15]. These works state that any transformation of the qutrit state can be represented by a 3×3 unitary matrix [11]. This transformation is known as a Z-transformation. A Z-transformation shifts or permutes the input states to generate the desired output states. For example, the (+1) Z-transformation can be represented by the following 3×3 matrix.

$$Z(+1) = \begin{vmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}$$

This Z-transformation shifts the input qutrit by 1. There are numerous Z-transformations that can be defined by varying 3×3 matrices, but the most useful transformations are shown in Table 2.6. There are five ternary one-place operations corresponding to the permutation of ternary values 0,1 and 2 and each of these can be constructed as ternary reversible gates. These gates are known as ternary 1-qutrit permutative gates. Table 2.6 shows the operations of these 1-qutrit permutative gates, and the symbol for the ternary 1-qutrit permutative gate is shown in Figure 2.3 (a) [11].

Table 2.6: Operations of 1-qutrit permutative gates, also referred to Z-transformations [11].

Input	Output				
	+1	+2	12	01	02
0	1	2	0	1	2
1	2	0	2	0	1
2	0	1	1	2	0

Two 1-qutrit permutative gates act as another 1-qutrit permutative gate if they are cascaded together [11]. The cascading gates have a serial effect on the input which produces

a resulting output that is similar to the single output of another 1-qutrit gate. Figure 2.3 (b) shows the resultant 1-qutrit gate for two cascaded 1-qutrit gates as shown in [11].

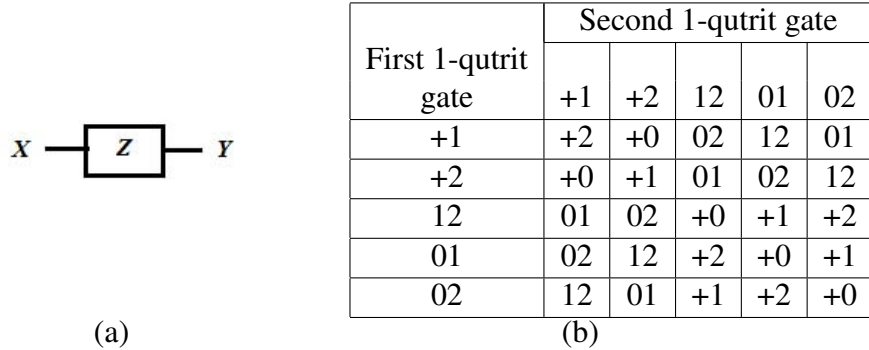


Figure 2.3: The ternary 1-qutrit permutative gate. (a) The commonly-used symbol as shown in [14], (b) and the cascading operations.

2.8.2 Muthukrishnan-Stroud (M-S) Gate

The M-S gate is a 2-qutrit multi-valued gate which can be realized using ion-trap technology. The M-S gate can be represented by the symbol shown in Figure 2.4 [14]. The value of input A controls the value of Q , which is the Z -transformation of input B whenever $A = 2$, where $Z \in \{+1, +2, 12, 01, 10\}$. If $A \neq 2$ then input Q is passed unchanged as $Q = B$ [14]. The cost of these elementary quantum gates is usually measured by the number of 1- and 2-qutrit transformations required to implement their behavior using some quantum realizations [22]. The M-S gate is considered to be an elementary quantum building block which in [14] is defined to have a cost of 1. Therefore, the cost metric used in this thesis is the number of M-S gates required to construct a ternary gate or circuit. Further details on the Z -transformation and Galois field logic are given in [14, 15] and [30]. Table 2.7 shows how the M-S gate generates different outputs depending on the Z -transformation applied to a controlled input.

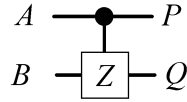


Figure 2.4: Symbol for the ternary Muthukrishnan-Stroud (M-S) gate [14].

Table 2.7: Operations of the M-S gate for $A = 2$ [11].

Input B	Output Q				
	Z(+1)	Z(+2)	Z(12)	Z(01)	Z(02)
0	1	2	0	1	2
1	2	0	2	0	1
2	0	1	1	2	0

2.8.3 Ternary Shift Gates

Six 1-qutrit ternary shift gates have been proposed by [12]. These gates are based on GF3 addition and multiplication operations. Figure 2.5 shows the symbols and operation of these six 1-qutrit ternary shift gates [12]. An example may help to clearly understand the table. If the value of x is 1, the output of a self-dual-shift gate would be $(2x + 2) \bmod 3 = 1$.

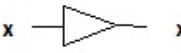
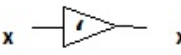
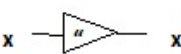



Gate Name	Gate Symbol	Operation
Buffer		$x = x$
Single-Shift		$x' = x+1$
Dual-Shift		$x'' = x+2$
Self-Shift		$x''' = 2x$
Self-Single-Shift		$x^\# = 2x+1$
Self-Dual-Shift		$x^\wedge = 2x+2$

Figure 2.5: 1-qutrit ternary shift gates [12].

2.8.4 Ternary Feynman gate

Figure 2.6 (a) shows the symbol for the ternary representation of the Feynman gate [14]. A 2×2 Feynman gate has 2 inputs and 2 outputs. The first output restores the first input and the second output provides the EXORed value of the two inputs. Figure 2.6 (b) shows the truth table [14].

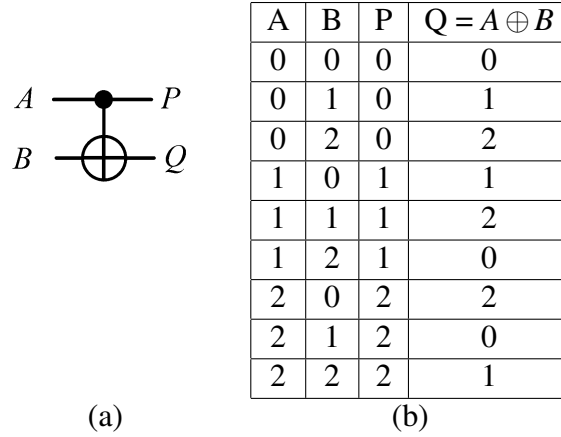


Figure 2.6: (a) The commonly-used symbol of the ternary 2-qutrit Feynman gate, and(b) the truth table [14].

The notations of the ternary Feynman gate that are used in this thesis are shown in Figure 2.7 [15].

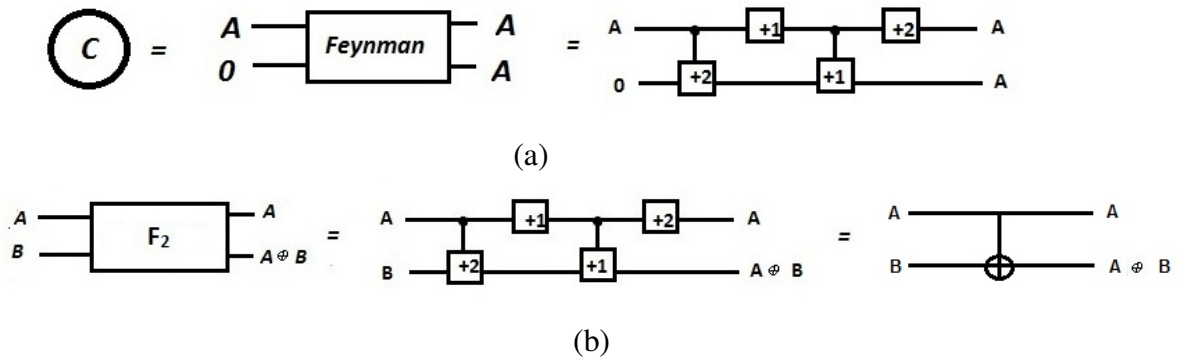


Figure 2.7: (a) The 2-qutrit copy Feynman gate, and (b) the 2-qutrit basic Feynman gate [15].

Implementation of the ternary Feynman gate using M-S gates

Figure 2.7 (b) shows the ternary M-S gate realization of the 2×2 ternary Feynman gate. The operation of the ternary Feynman gate can be described using the principle of the M-S gate. Input B of the ternary Feynman gate is Z-transformed to generate the desired output, and this Z-transformation of the input B is controlled by input A . If input $A = 0$, neither

of the M-S gates connected to B will be activated and hence no Z-transformation will be applied. Therefore, if $A = 0$ and $B \in \{0, 1, 2\}$, then $Q = 0 \oplus B = B$. Similarly if $A = 1$, the second M-S gate connected to B will be activated and output $Q = B \oplus 1$ will be generated by applying $Z(+1)$ transformation on input B where $B \in \{0, 1, 2\}$. Finally if $A = 2$, the first M-S gate connected to B will be activated and output $Q = B \oplus 2$ will be generated by applying the $Z(+2)$ transformation on input B [14]. As we can see in 2.7 (b) we require 4 gates to implement the Feynman gate, thus its cost is said to be 4 [24].

2.8.5 Ternary Toffoli gate

The ternary Toffoli gate has three inputs and three outputs. The first two outputs restore the first two inputs while the last output generates the result. If the inputs of a ternary Toffoli gate are A, B and C , the outputs would be $P = A, Q = B, R = AB \oplus C$. The ternary Toffoli gate is shown in Figure 2.8 (a) while Figure 2.8 (b) shows its symbol and Figure 2.8 (c) shows the truth table [14].

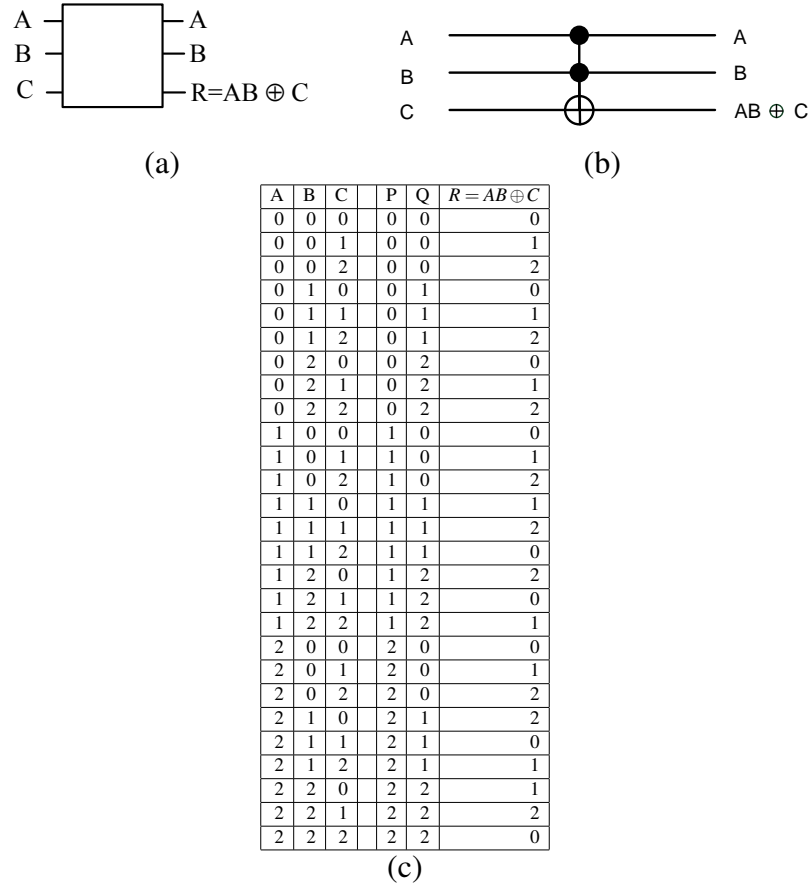


Figure 2.8: The 3-qutrit ternary Toffoli gate, (a) with I/O mapping as shown in [14], (b) its symbol, and (c) the truth table.

Implementation of the ternary Toffoli gate using M-S gates

Figure 2.9 shows the corresponding M-S gate implementation of a 3-qutrit ternary Toffoli gate. The first two inputs are the controlling inputs and the last one is the controlled input. The two controlling inputs change the value of a constant input to trigger the Z-transformation that changes the controlled input. The output corresponding to the controlled input is the Z-transformation of that input only when the values of the two controlling inputs are 2. Otherwise, the controlled input will be passed unchanged to the output R. As in Figure 2.9, when $A = B = 2$, the output R will be the Z-transformation of C where

$Z = +1$; $R = C$ otherwise. The M-S gate implementation also shows how the values of controlling inputs are used to change a constant input 0 to control the Z-transformation on the controlled input C [14]. The cost of this realization is 5 [14].

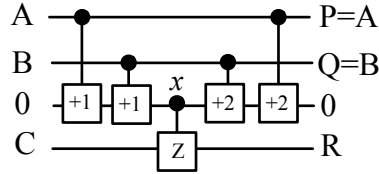


Figure 2.9: The realization of a 3-qutrit ternary Toffoli gate using M-S gates [14].

2.8.6 Ternary controlled-controlled gate

The ternary controlled-controlled gate was also proposed in [11]. The ternary controlled-controlled gate has two controlling inputs and one controlled output. Figure 2.10 illustrates the symbol of a ternary controlled-controlled gate [11]. From the figure it can be seen that the Z-transformation is applied on W only when the values of both of the inputs X and Y are 2, otherwise W is passed unchanged. The implementation of the ternary controlled-controlled gate is identical to the Toffoli gate. Two controlling lines are directly connected to a constant input which trigger the Z-transformation that changes the controlled input. If the values of both controlling inputs are 2, the Z-transformation will be triggered, otherwise the controlled input is passed unchanged through to the output. The difference between the Toffoli and the controlled-controlled gate is the value of the Z-transformation. In the Toffoli gate, the value of the Z-transformation that will change the controlled input is equal to $+AB$. However, in the controlled-controlled gate the value of the Z-transformation can be anything *i.e* $Z \in \{+1, +2, 12, 01, 10\}$.

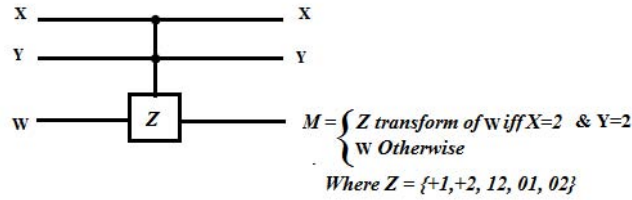


Figure 2.10: Ternary controlled-controlled gate as shown in [11].

2.8.7 3-qutrit generalized Toffoli gate (GTG)

In a 3-qutrit generalized Toffoli gate there are two controlling inputs and one controlled input. As described in Section 2.8.5, whenever the values of the two controlling inputs are 2, a Z-transformation is applied on the controlled input to generate the output. For all other combinations of the controlling inputs, the controlled input is passed unchanged. To make the operation more generalized, *i.e.* to allow the Z-transformation to be activated for other combinations of the controlling inputs (other than only 2, 2), the author of [14] proposed a generalized Toffoli gate. Therefore, the basic Toffoli gate can operate for the inputs $A = 2$ and $B = 2$ only, whereas the GTG can operate for the other values of A and B also, such as for $A = 1$ and $B = 2$. The realization of the GTG is shown in Figure 2.11.

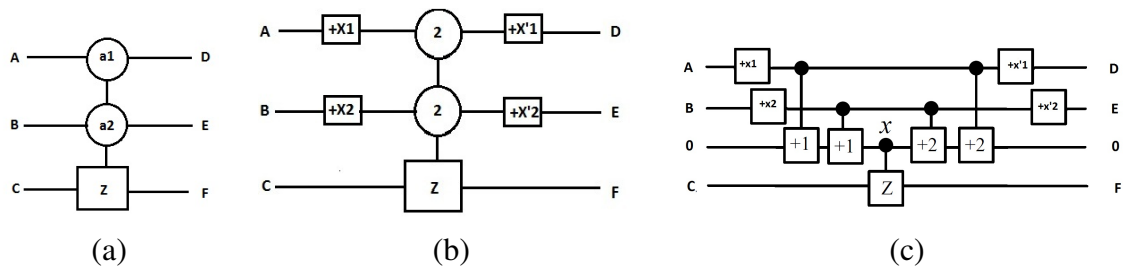


Figure 2.11: (a) Symbol of the generalized Toffoli gate, (b) its block diagram [15], and (c) its realization using the M-S gate [14].

Figure 2.11 shows how the values of the controlling inputs A and B should be shifted to trigger the Z-transformation on input C . If we assume the values of the controlling inputs are $A = a_1$ and $B = a_2$ then the idea is to shift the controlling value a_I (where I

$\in \{1, 2\}$) to 2 using the appropriate shift to activate the Z-transformation. This is illustrated in Figure 2.11(b). The X_I shift is necessary to make the value of the controlling input 2 where $+(X_I) = +(2 - a_I)$ and $I \in \{1, 2\}$. The resulting values are used to activate the Z-transformation. An inverse gate denoted by X'_I is used to restore the value of the controlling input to its initial value. The cost of this Toffoli gate is $5+2*(\text{number of non-2 controlling values})$ [15]. The non-2 controlling values refer to the controlling inputs that are not 2 and so require shifting to 2 to activate the Z-transformation.

As an example of its use, let us assume we wish to activate the (+2) Z-transformation for the input combination $A = 1$ and $B = 2$. In this case A is a non-2 controlling input. To force the value of A to be 2, we need to shift the input A by $+(2 - 1) = +1$. Input B does not need any shift since its value is already 2. The resultant $A = 2$ and $B = 2$ are used to activate the Z transformation to be applied on C and generate output $F = AB \oplus C = 2 \oplus C$.

2.8.8 Construction of ternary reversible circuits

In order to create a ternary reversible circuit, a cascade of ternary reversible gates is constructed. For example, Figure 2.12 shows how two Feynman gates can be cascaded to implement the function $a \oplus b \oplus c$.

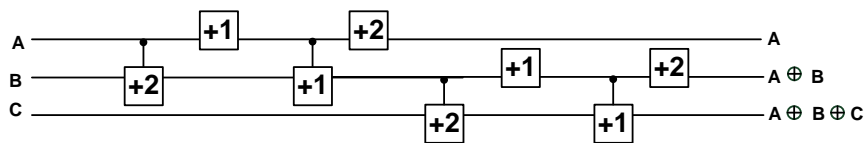


Figure 2.12: Cascading gates to construct a ternary reversible circuit implementing $a \oplus b \oplus c$.

2.9 Online Testing in Reversible Logic Circuits

Testing of a circuit is carried out to detect errors in a circuit or determine whether the circuit is error free or not. Testing holds immense importance in the implementation phase of circuits. Complex circuits may contain some ambiguous or faulty outputs that have been caused by flaws in the circuit hardware or problems in the fabrication process and testing is required to detect those flaws [8].

2.10 Online Testing

Online testability is the ability of a circuit to test a portion of the circuit while the circuit is operating [36]. Detecting a fault in operation, the point of occurrence and in some cases, attempting to recover from the fault are the major focuses of research in online testing [8]. Similar processes for binary reversible logic have been discussed in [8] and [9]. In [8] the authors discuss built-in self-test (BIST) for digital circuits. BIST refers to the design of a circuit to test itself, but not while operating, *i.e.* offline.

In [9] the author discusses the concept of self checking logic and three categories for self testable circuits [17]: fault secure, self-testing and totally self-testing digital circuits. According to the author, a fault secure digital circuit should have the characteristic that the output will not be affected by any single bit fault. A self-testing circuit refers to a digital circuit which generates outputs of some invalid pattern to represent a fault that occurs inside the circuit. A totally self-testing circuit must be both fault secure and self-testing [9]. The circuit we propose in this work falls into the self-testing digital circuit category.

2.11 Works related to online testable reversible logic gates

More than one approach to binary reversible online testing is found in the literature. The existing reversible gates like Fredkin and Toffoli gates can not incorporate online testability [36]. Hence, researchers have designed new gates with built in online testability features in [36].

2.11.1 Binary online testing using R, R1 and R2 gates (approach 1)

In [36] the authors propose three new reversible gates R, R1 and R2 which are used to construct online testable circuits. R and R1 are designed as the building blocks for arbitrary functions while R2 includes the online testable feature.

The block diagrams of the 3×3 R gate, 4×4 R1 gate and 4×4 R2 gate are shown in Figure 2.13 [36].

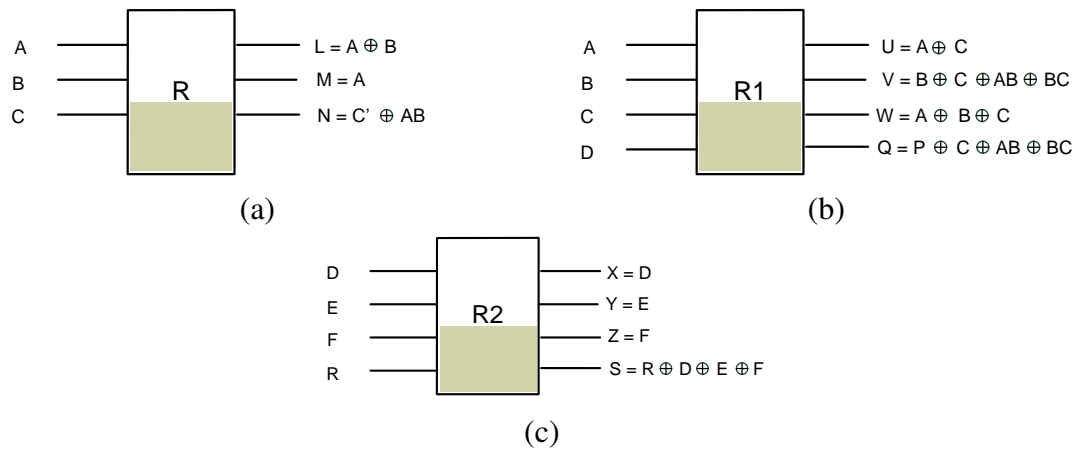


Figure 2.13: (a) The R gate, (b) R1 gate, and (c) the R2 gate [36].

The 4×4 R1 gate can be used to realize AND, OR, NAND, NOR, XNOR and XOR operations by setting different values on different inputs. The R1 gate has a parity output at

Q . The R2 gate is designed to have the online testability features integrated into it. Beside duplicating inputs, the R2 gate also generates a parity at the output S . A R1 gate is cascaded by its first three outputs with the first three inputs of a R2 gate to construct a testable logic block (TLB). R2 passes its three inputs unchanged through to the outputs. Figure 2.14 shows the construction of the testable logic block. The parities are compared to detect if there is any single bit fault in the circuit. If P is set to R' , Q being a complement of S represents a fault-free situation.

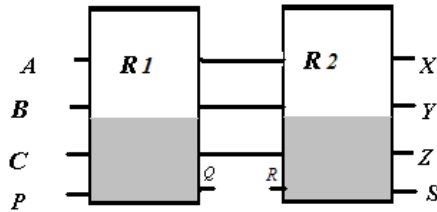


Figure 2.14: Testable logic block (TLB) [36].

Two-pair rail checker using the R gate

A rail checker circuit is used for checking the outputs of the online testable blocks and propagating the error along a larger circuit. In [36] a two-pair rail checker has been designed using R gates. Figure 2.15 illustrates the configuration of the rail checker circuit from [36]. The two-pair rail checker defines two error checking functions as follows:

$$E1 = X0Y1 + Y0X1$$

$$E2 = X0X1 + Y0Y1$$

The outputs of the functions $E1$ and $E2$ depend on the inputs $X0, Y0$ and $X1, Y1$ which are originally the outputs Q and S from two testable logic blocks (TLB). The outputs $E1$ and $E2$ are the complement of each other only if the inputs X_a and Y_a are the complement of each other, otherwise $E1 = E2$. This allows the two-pair rail checker to test the testable

logic blocks for any fault occurrence and as well propagate the error.

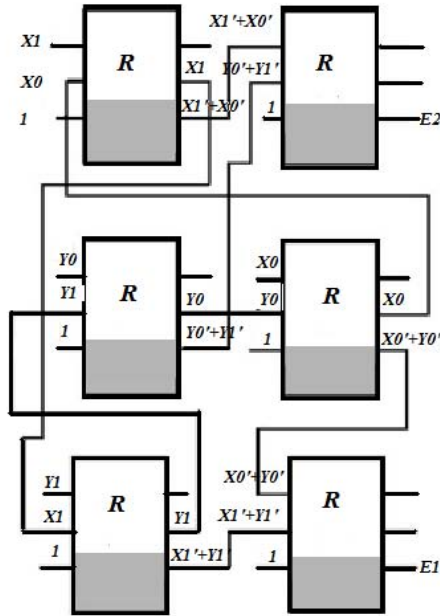


Figure 2.15: Rail checker circuit using the R gate [36].

2.11.2 Testing using a universal reversible logic gate (approach 2)

In [19], the authors proposed an approach to directly construct an online testable circuit from a given reversible circuit. The authors propose two steps for this construction. In the first step, every $n \times n$ reversible gate G in that circuit is transformed into a new $(n + 1) \times (n + 1)$ Deduced Reversible Gate DRG (G). This can be achieved by adding an extra input bit P_{ia} and the corresponding output bit P_{oa} to the reversible gate G , maintaining the original functionality of the gate. Figure 2.16 shows the configuration.

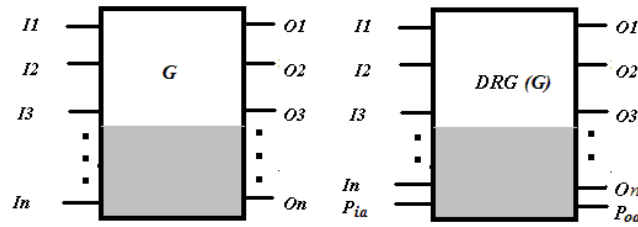


Figure 2.16: G gate and Deduced reversible logic gate DRG (G) as in [19].

In the second step, a testable reversible cell of G, TRC (G) is constructed by cascading the DRG (G) with a deduced identity gate. An identity gate is an $n \times n$ reversible gate where all the inputs are simply copied to the outputs. For instance if X is an $n \times n$ identity gate, then a deduced identity gate of X, DRG (X) can be easily constructed similarly from the reversible gate X by adding an extra input bit P_{ib} and the corresponding output bit P_{ob} . The DRG (G) and DRG (X) are cascaded by connecting the first n outputs of DRG (G) and the first n inputs of DRG (X). The new $(n + 2) \times (n + 2)$ block is called a Testable Reversible Cell, TRC (G) [19], which is the final online testable gate. The construction is shown in Figure 2.17.

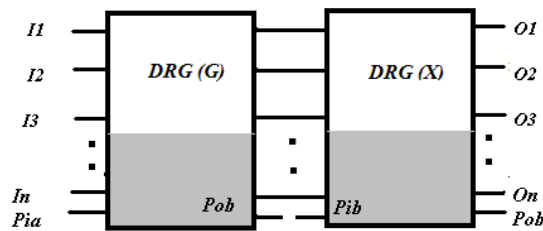


Figure 2.17: Cascaded DRGs to form a TRC [19].

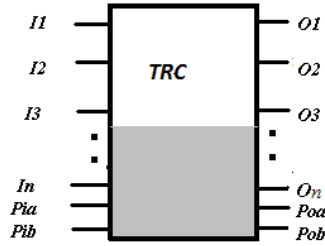


Figure 2.18: Testable Reversible Circuit (TRC) [19].

The TRC (G) has two parity outputs P_{oa} and P_{ob} as shown in Figure 2.18. If $P_{ia} = P_{ib}$, P_{oa} being the complement of P_{ob} indicates a faulty situation. Following the procedure mentioned above every reversible gate in the circuit is replaced by the corresponding TRC. If we assume that there are m TRCs in the circuit, then there would be $2m$ parity outputs in the circuit. To check the output parities a $(2m + 1) \times (2m + 1)$ test cell (TC) is constructed. The first $2m$ inputs of the TC are the parity outputs from the TRCs which are passed through to the outputs. The last input is e which is set to the value 0 or 1, and the corresponding output is $T = ((P_{oa1} \oplus P_{ob1}) + (P_{oa2} + P_{ob2}) + \dots + (P_{oam} + P_{obm}) \oplus e)$. Here P_{oa_n} and P_{ob_n} are the parity outputs of the n^{th} TRC. Figure 2.19 shows the configuration. A single bit fault is detected if $T = 1$, provided e is set to 0.

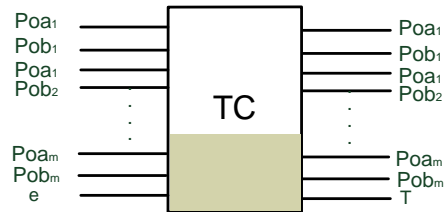


Figure 2.19: Test Cell (TC) [19].

2.12 Summary

A brief literature overview on reversible logic, MVL, TGFSOPs, ternary reversible gates and the related work on binary online testable logic gates are presented in this chapter.

Several existing gates are defined and examples of their uses are given. These logics and gates are the fundamental requirements for understanding the new proposed ternary online testable block design which we discuss in the following chapters.

Chapter 3

Ternary Online Testable Circuits

3.1 Introduction

The area of fault detection for reversible circuits is fairly new, although works such as [16, 31] and [38] are beginning to address this area. A great deal of effort has been engaged in finding ways to design and build ternary reversible gates and circuits. However modeling the faults of ternary reversible circuits and detection of the faults in real time is yet to be addressed, despite the clear significance of these areas.

Since quantum computing is also a rapidly emerging area and quantum logic operations are reversible, studying how to build online testable ternary reversible circuits may assist in the development of test methods for quantum circuits.

In this chapter we present designs for two ternary blocks: one to implement basic ternary logic functions and the other to implement online testability to detect any single bit error. Since we are working with ternary logic, a single bit error should technically be referred to as a single qutrit error; however for the sake of simplicity we use the term “single bit error” through out the rest of the thesis.

3.2 Design of the online testable ternary reversible logic block

The testable blocks that we propose are built using ternary reversible gates, and so the resulting blocks are also reversible. The M-S gate, ternary Toffoli gate and ternary Feynman gates are the building blocks that we use, and these gates are introduced in Chapter 2.

There are thousands of ternary one-place and two-place functions. However we are

most interested in those which can be easily realized in a circuit and have the ability to represent general p -valued functions [23]. Therefore, we have chosen the following ternary functions to be implemented in our block.

- Successor : $f(x) = (x + 1) \bmod 3$
- Predecessor: $f(x) = (x - 1) \bmod 3$
- Negation: $f(x) = (2 - x) \bmod 3$
- Mod-Sum: $f(x, y) = (x + y) \bmod 3$
- Mod-difference: $f(x, y) = (x - y) \bmod 3$

In our research we have found that these ternary functions can be easily represented using ternary Galois field addition and multiplication. Therefore, these functions can be represented using a Ternary Galois Field Sum of Products (TGFSOP) as shown below. Here “ \oplus ” refers to ternary addition and no operator between the variables refers to ternary multiplication.

- Successor : $f(a) = (a \oplus 1)$
- Predecessor: $f(a) = (a - 1) = \{a \oplus (-1)\}$
- Negation: $f(a) = (2a \oplus 2)$
- Mod-Sum: $f(a, b) = (a \oplus b)$
- Mod-difference: $f(a, b) = (a \oplus 2b)$

There are two major differences between the blocks proposed in [36] and our ternary reversible logic blocks. The first is that the successor operator is used instead of complement as will be shown in Section 3.2.3. The second is that, as will be shown in Section 5.2, our component consists of multiple gates rather than being an individual gate. The two blocks that compose our design are described in the following sections.

3.2.1 The TR1 block

The 4×4 TR1 block can be defined as $I = (A, B, C, P)$ and $O = (L = AB \oplus C, M = A \oplus B, N = 2AB, Q = P \oplus A \oplus B \oplus C)$, where I and O are input and output sets respectively. The block is shown in Figure 3.1.

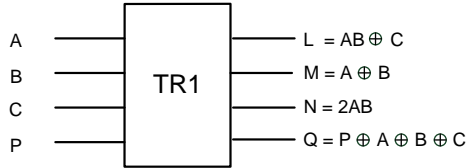


Figure 3.1: Configuration of the TR1 block.

The TR1 block has A , B , C and P as inputs where P is required for the error checking functionality. This block generates outputs $L = AB \oplus C$, $M = AB$, $N = 2AB$ and Q . Here the outputs L , M and N are sufficient to implement the discussed ternary operations and Q is used for error detection. The value of Q should be equivalent to the sum of the outputs L , M and N , and input P . The operations are independent of the input P and P is set to an arbitrary value 0, which will be used in the testability feature. The following sections describe how the inputs are chosen in order to implement the discussed ternary operations.

The AND operation in the TR1 block

The ternary AND operation (GF3 Multiplication) can be easily implemented using the TR1 block. To implement an AND operation, input C is set to 0 where A and B are set as the operands of the AND operation. L provides the desired output. Figure 3.2 shows the configuration of the TR1 block and the truth table of the ternary AND operation discussed in [15].

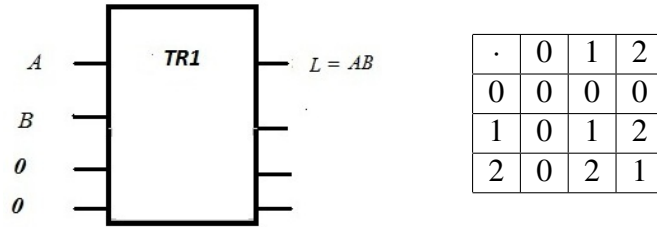


Figure 3.2: AND operation in the *TR1* block and the corresponding truth table.

The Mod-sum (EXOR) operation in the TR1 block

The ternary mod-Sum (EXOR) operation is identical to the GF3 addition and can also be easily implemented using a *TR1* block. To implement the mod-sum operation, input *C* is set to 0 where *A* and *B* are set as the operands of the mod-sum operation. *M* provides the desired output. Figure 3.3 shows the configuration and the truth table of the ternary mod-sum operation [15] respectively.

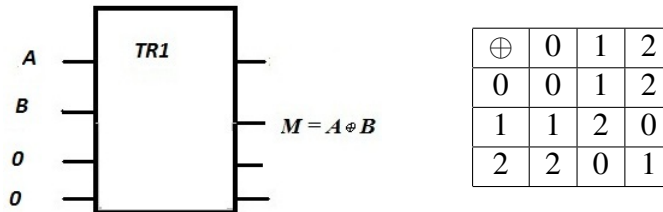


Figure 3.3: Mod-sum operation in the *TR1* block and the corresponding truth table.

The successor operation in the TR1 block

The ternary one-place operation successor (\vec{x}) is defined as $\{(x + 1) \bmod 3\}$ [23] where *x* is the ternary input. To implement the successor operation, input *C* is set to 0 and *A* and *B* are set as the operands of the successor operation *i.e.* $A = x$ and $B = 1$. The output *M* provides the desired output. Figure 3.4 shows the configuration of the *TR1* block and the corresponding truth table to implement the successor operation.

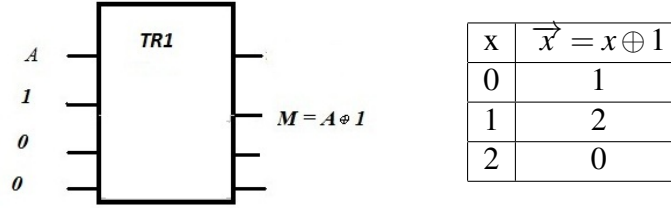


Figure 3.4: The successor operation in the *TR1* block and the corresponding truth table.

Negation/complement operation in the TR1 block

The ternary one-place operation negation/complement is defined as $\{(2 - x) \bmod 3\}$ [23] where x is the ternary input. We have found that the negation operation for ternary can be described using a TGFSOP representation as $f(x) = 2x \oplus 2$. This statement can be verified from the table in Figure 3.5.

To implement the negation operation, inputs A and C are set to 2 and B is set as the operand of the negation/complement operation *i.e* $B = x$. L provides the desired output. Figure 3.5 shows the configuration of *TR1* to implement the negation/complement operation.

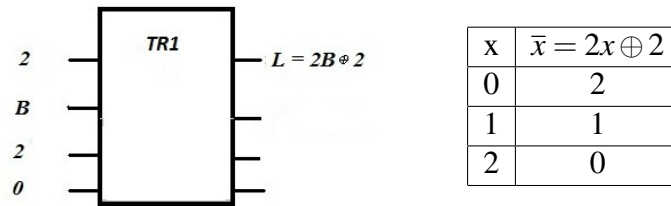


Figure 3.5: The negation/complement operation in the *TR1* block and the corresponding truth table.

Mod-difference operation in the TR1 block

The ternary mod-difference operation is defined as $\{(x - y) \bmod 3\}$ in [23] where x, y are the ternary inputs. We have discovered that the mod-difference operation can be described

using a TGFSOP representation as $f(x,y) = x \oplus 2y$. This can be verified from the table in Figure 3.6. To implement the mod-difference operation, input A is set to 2, $B = y$ and $C = x$. Since the commutative law is applicable on MVL operators [23], it can be verified that $x \oplus 2y = 2y \oplus x$. L provides the desired output. Figure 3.6 shows the configuration of $TR1$ to implement the mod-difference operation.

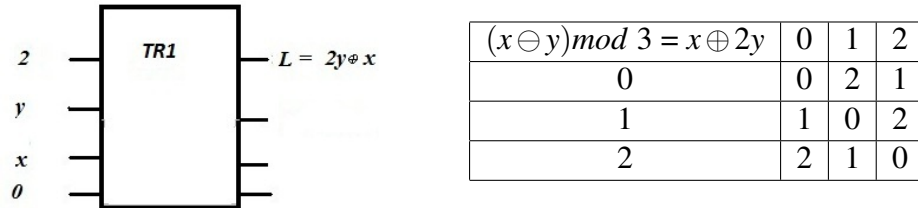


Figure 3.6: The mod-difference operation in the $TR1$ block and the corresponding truth table.

3.2.2 The $TR2$ block

While the $TR1$ block is used to implement the desired functionality the $TR2$ block incorporates the online testing features. The 4×4 $TR2$ block can be defined as $I = (D, E, F, R)$ and $O = (U = D, V = E, W = F, S = R \oplus D \oplus E \oplus F)$, where I and O are input and output sets respectively.

Outputs U, V and W are the copies of inputs D, E and F , and can be used for cascading to additional testable blocks. Input R is required for testing purposes. Output S is the EXOR of the inputs of the $TR2$ block. S is used to detect any single bit error when $TR2$ is cascaded with the $TR1$ block to form an online testable block. Figure 3.7 shows the block diagram of the $TR2$ block. In an online testable block the $TR2$ block receives the first three outputs of the $TR1$ block as inputs and generates their copies along with the error detecting output S .

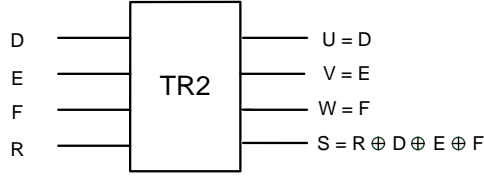


Figure 3.7: The $TR2$ block.

3.2.3 The online testable block

To construct an online testable ternary reversible block (TR), $TR1$ and $TR2$ are cascaded together. The $TR1$ and $TR2$ blocks are 4×4 blocks while the TR block that is formed by connecting $TR1$ to $TR2$ by their first three outputs and inputs is a 5×5 block. When $TR1$ and $TR2$ blocks are used to construct an online testable block, input P of the $TR1$ block and input R of the $TR2$ block must be set in such a way that $R = \vec{P}$. Since we must set R to be a successor of P , we can set $P = 0$ and $R = 1$, $P = 1$ and $R = 2$ or $P = 2$ and $R = 0$. For regular operation we have chosen to set $P = 0$ and $R = 1$. Figure 3.8 shows the configuration and Figure 3.9 shows the block diagram of TR.

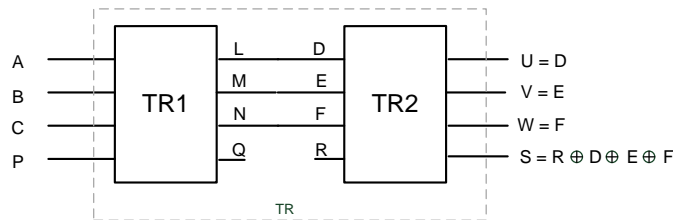


Figure 3.8: Configuration of the online testable ternary reversible block (TR).

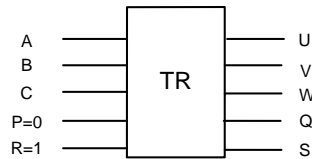


Figure 3.9: Online testable ternary reversible block (TR).

$TR1$ takes ternary logic values for implementing the desired functionality, as discussed in Section 3.2.1, at A , B , C , and P is set to 0. At the Q output, $TR1$ generates $P \oplus A \oplus B \oplus C$

where $P = 0$. $TR1$ has been constructed so that $A \oplus B \oplus C$ can be equal to $L \oplus M \oplus N$ only if no error occurs inside $TR1$. $TR2$ transfers the input values D, E, F to outputs U, V and W , where $D = L, E = M$ and $F = N$. $TR2$ also generates the error detecting output at S . The output S will be the successor of Q if no error occurs in $TR1$ and $TR2$. S is the EXOR of the inputs of $TR2$ where $R = 1$.

The error detection principle used in TR is relatively simple. The value of output S should be the successor of Q if the operation is not affected by any internal error. Since $3AB = 0$, the value of Q is

$$\begin{aligned} Q &= P \oplus L \oplus M \oplus N \\ &= P \oplus AB \oplus C \oplus A \oplus B \oplus 2AB \\ &= P \oplus A \oplus B \oplus C. \end{aligned}$$

Let us assume $X = A \oplus B \oplus C$ and $Y = D \oplus E \oplus F$. Then $Q = P \oplus X = 0 \oplus X$ and $S = R \oplus Y = 1 \oplus Y$ since $P = 0$ and $R = 1$ during regular operation. If the operations are error free, then

$$\begin{aligned} D \oplus E \oplus F &= L \oplus M \oplus N \\ &= A \oplus B \oplus C. \end{aligned}$$

Therefore, X would be equal to Y , *i.e.* $X = Y$, which results in $S = \vec{Q}$ since $R = \vec{P}$.

If any error occurs within the $TR1$ block and the fault is reflected on a single output of the $TR1$ block, then the flawed values of D, E, F will be used in calculating U, V, W and S . Therefore, $A \oplus B \oplus C$ will no longer be equal to $L \oplus M \oplus N$ and $D \oplus E \oplus F$. The result

will be the outputs Q and S not being successors, indicating a flaw. The relevant portion of the truth tables of the $TR1$ and $TR2$ blocks are shown in Table 3.1.

Table 3.1: A subset of the truth tables for (a) the $TR1$ block and (b) the $TR2$ block.

Input				Output			
A	B	C	P	L	M	N	Q
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	0	2	0	2	0	0	2
0	1	0	0	0	1	0	1
0	1	1	0	1	1	0	2
0	1	2	0	2	1	0	0
0	2	0	0	0	2	0	2
0	2	1	0	1	2	0	0
0	2	2	0	2	2	0	1
1	0	0	0	0	1	0	1
1	0	1	0	1	1	0	2
1	0	2	0	2	1	0	0
2	0	0	0	0	2	0	2
2	0	1	0	1	2	0	0
2	0	2	0	2	2	0	1
1	1	0	0	1	2	2	2
1	2	0	0	2	0	1	0
1	1	1	0	2	2	2	0
1	1	2	0	0	2	2	1
1	2	1	0	0	0	1	1
1	2	2	0	1	0	1	2
2	1	0	0	2	0	1	0
2	1	1	0	0	0	1	1
2	1	2	0	1	0	1	2
2	2	0	0	1	1	2	1
2	2	1	0	2	1	2	2
2	2	2	0	0	1	2	0

(a)

Input				Output			
D	E	F	R	U	V	W	S
0	0	0	1	0	0	0	1
1	0	0	1	1	0	0	2
2	0	0	1	2	0	0	0
0	1	0	1	0	1	0	2
1	1	0	1	1	1	0	1
2	1	0	1	2	1	0	1
0	2	0	1	0	2	0	0
1	2	0	1	1	2	0	1
2	2	0	1	2	2	0	2
0	1	0	1	0	1	0	2
1	1	0	1	1	1	0	0
2	1	0	1	2	1	0	1
0	2	0	1	0	2	0	0
1	2	0	1	1	2	0	1
2	2	0	1	2	2	0	2
1	2	2	1	1	2	2	0
2	0	1	1	2	0	1	1
2	2	2	1	2	2	2	1
0	2	2	1	0	2	2	2
0	0	1	1	0	0	1	2
1	0	1	1	1	0	1	0
2	0	1	1	2	0	1	1
0	0	1	1	0	0	1	2
1	0	1	1	1	0	1	0
1	1	2	1	1	1	2	2
2	1	2	1	2	1	2	0
0	1	2	1	0	1	2	1

(b)

For example, let $A = 0$, $B = 1$ and $C = 2$. The outputs of $TR1$ would be $L = AB \oplus C = 2$, $M = A \oplus B = 1$, $N = 2AB = 0$ and $Q = P \oplus A \oplus B \oplus C = 0$. Since $TR2$ receives the outputs L, M, N of $TR1$ as its inputs *i.e.* $D = L = 2$, $E = M = 1$, $F = N = 0$, the output S would be

$S = R \oplus D \oplus E \oplus F = 1$ since we are assuming that no error occurred in the middle of the operation. Therefore, S being the successor of Q represents the fault-free situation.

On the other hand, let us assume that, in the middle of the operation some error occurred in $TR1$ and is reflected on output N . Output N is changed to 1 due to the error. Therefore, the outputs of $TR1$ become $L = 2, M = 1, N = 1$ and $Q = 0$. Since $TR2$ will take the flawed L, M and N outputs as its inputs, $TR2$ will generate $S = 1 \oplus 2 \oplus 1 \oplus 1 = 2$. S is then no longer a successor of Q , hence the presence of an error is assumed.

The inputs and outputs for the testable block in both fault free and faulty situations are shown in Figure 3.10.

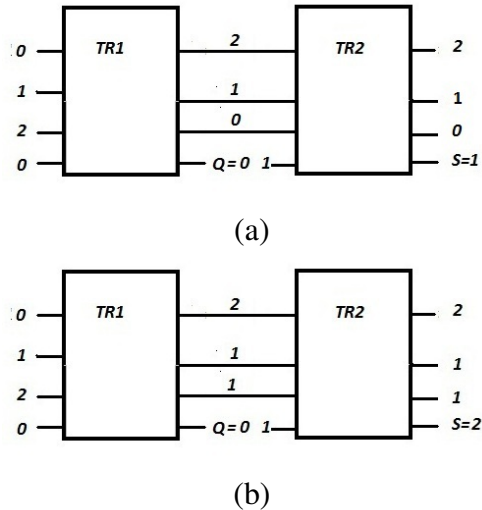


Figure 3.10: (a) Fault-free operation, and (b) faulty operation of a circuit constructed from the TR block.

This design is based on the concepts in [36] where a similar principle is used to detect a single bit error in a binary block. However in our work we are working with ternary and thus the modifications required are significant.

3.3 Summary

The blocks described in this chapter are the building blocks for any ternary online testable circuit. The TR block can be used to implement any ternary function as well as ensure testability.

Chapter 4

Ternary Two-Pair Two-Rail Checker

4.1 Introduction

As described in [26], error checking is often implemented using one of the two main techniques, parity codes and two-pair two-rail checkers. Rail checkers compare the outputs from more than one identical system. This process is also used to reduce the number of error detecting outputs. The two-pair two-rail checker receives two pairs of inputs and generates a pair of outputs that indicates if the prior operations were fault-free or faulty. A ternary two-pair two-rail checker should receive two pairs of inputs from two TR blocks, compare them and indicate in the outputs if any flaw was identified by either of those TR blocks [26]. This concept has also been used in the reversible context by other authors such as in [4] and [36].

4.2 Designing the ternary rail checker circuit

During the development of our designs, we first focused on the error checking functions for implementing the ternary two-pair two-rail checker as proposed in [36]. However, as detailed in Section 4.2.2, these functions are not suitable for detecting ternary single bit errors and hence are rejected afterwards. We detail our investigations here as background for the reader. The functions are

$$e1 = x0y1 + y0x1 \text{ and } e2 = x0x1 + y0y1$$

where $x0$ and $y0$ are the outputs of one testable block and $x1$ and $y1$ are the outputs of another testable block. Table 4.1 shows possible combinations of $x0, y0, x1$ and $y1$ when the testable blocks are fault-free. It was assumed that whenever $y0 = \vec{x0}$ and $y1 = \vec{x1}$, $e2$

will be the successor of $e1$. Hence, $e2$ being a successor of $e1$ shows that the testable blocks are fault-free.

Table 4.1: Combination of fault-free inputs and outputs for the ternary rail checker.

Inputs				Outputs	
$x0$	$y0$	$x1$	$y1$	$e1$	$e2$
0	1	0	1	0	1
0	1	1	2	1	2
0	1	2	0	2	0
1	2	0	1	1	2
1	2	1	2	1	2
1	2	2	0	1	2
2	0	0	1	2	0
2	0	1	2	1	2
2	0	2	0	0	1

4.2.1 Implementation of the ternary rail checker

A ternary version of this two-pair two-rail checker can be implemented using ternary Toffoli and Feynman gates. The Feynman gates are referred to as F2 or as copy gates, depending on their use. The implementation of such a circuit is shown in Figure 4.1.

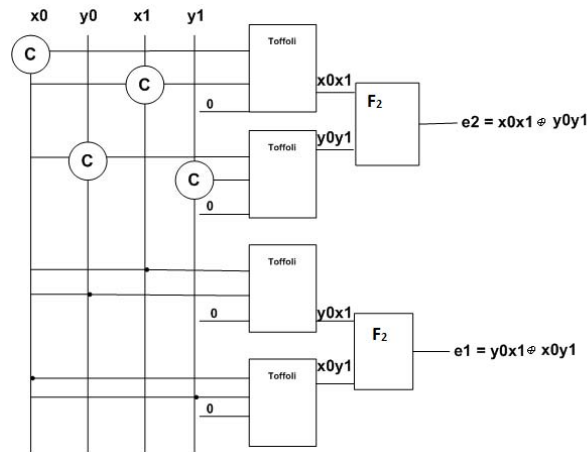


Figure 4.1: Ternary two-pair two-rail checker.

4.2.2 *Rejection of this design*

After thorough analysis of the design we have found that there are some input combinations for which e_2 should not be the successor of e_1 , although it is. Our design requires that the outputs of the rail checker should be $e_2 = \vec{e}_1$, representing the fault-free circuit, *only* for the input combinations where $y_0 = \vec{x}_0$ and $y_1 = \vec{x}_1$. However we found that sometimes $e_2 = \vec{e}_1$ even if the inputs are not successors of each other. For example, let us assume $x_0 = 1, y_0 = 0, x_1 = 1$ and $y_1 = 0$. This input combination clearly shows that the TR blocks attached to the checker are not fault-free since $y_0 \neq \vec{x}_0$ and $y_1 \neq \vec{x}_1$. However the equations mentioned in Section 4.2 generates $e_1 = 0$ and $e_2 = 1$, *i.e.* $e_2 = \vec{e}_1$ for these inputs. This result incorrectly implies that the circuit is fault-free. This illustrates that Lala's testing equations are not suitable for designing ternary rail checker and hence we rejected this design.

4.3 **New design of the ternary rail checker**

Although the above design was rejected, a two-pair two-rail checker is still necessary to detect and propagate errors in a large circuit. The new ternary rail checker is designed using ternary 1-qutrit permutative gates and ternary controlled-controlled gates. The operational principle and the implementation is discussed in the following sections.

4.3.1 *Principle of the ternary rail checker*

The purpose of the rail checker is to detect the existence of a flaw, if there is any, in the TR blocks attached to the rail checker. This is achieved by checking whether the outputs of the blocks are successors or not. The rail checker generates two outputs where one is successor to another if the attached blocks are fault-free. Since these outputs may be used afterwards

to cascade additional rail checkers, successor outputs are generated to represent the fault-free situation. Otherwise, if the rail checker detects any flaw in the attached *TR* blocks, the design guarantees that the outputs generated will never be successors. Figure 4.2 shows the block diagram of a general rail checker.

The rail checker is designed in such a way that if the inputs are successors, *i.e.* $y_0 = \overrightarrow{x_0}$ and $y_1 = \overrightarrow{x_1}$, the rail checker will generate $X_3 = 1$ and $X_4 = 2$, so that $X_4 = \overrightarrow{X_3}$, otherwise it generates outputs where $X_4 \neq \overrightarrow{X_3}$. There is no specific reason for choosing $X_3 = 1$ and $X_4 = 2$. Any two successors could have been used, for example $X_3 = 0$ and $X_4 = 1$ or $X_3 = 2$ and $X_4 = 0$.

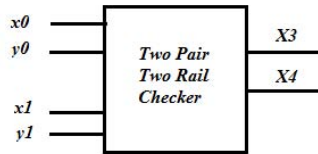


Figure 4.2: Block diagram for the new two-pair two-rail checker.

Example

Let us assume $x_0 = 1$, $y_0 = 2$, $x_1 = 0$ and $y_1 = 1$. Then the two-pair two-rail checker will generate $X_3 = 1$ and $X_4 = 2$ as its outputs. Alternatively let us assume $x_0 = 1$, $y_0 = 2$, $x_1 = 2$ and $y_1 = 1$. Then the rail checker will generate $X_3 = 0$ and $X_4 = 0$ as its outputs. Since here $y_1 \neq \overrightarrow{x_1}$, the rail checker generates $X_4 \neq \overrightarrow{X_3}$ and represents the faulty situation.

4.3.2 Elementary *E* gate

To implement the rail checker discussed above, an elementary gate (*E*) with two controlling inputs and one controlled input has been designed. The design of the *E* gate is based on the architecture of the 1-qutrit ternary comparator circuit proposed in [11]. Figure 4.3

shows the block diagram of the E gate. The behavior of the Z-transformation depends on whether the second input (y) is successor to the first input (x) or not. If the second input is a successor of the first input, the Z-transformation changes the controlled input, as previously described, to either a 1 or a 2. Otherwise, 0 is passed unchanged through to K . There are actually two E-gate designs, one with an output of 1 if $x_0 = \overrightarrow{y}_0$ and one with an output of 2 if $x_1 = \overrightarrow{y}_1$ where x_0, y_0 and x_1, y_1 are outputs from two attached TR blocks. The E-gate with an output of 1 is denoted by E_a and the E-gate with an output of 2 is denoted by E_b .

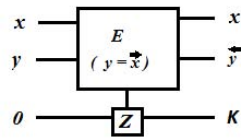


Figure 4.3: E gate.

1-qutrit permutative gates and ternary controlled-controlled gates are used to design our E gates. Table 4.2 shows all possible input combinations for any E gate, the desired output for that combination, and shifts required to generate that output.

Table 4.2: Truth table and transformation table of the E gates.

Input	Output	Shift Required	Output	Shift Required
00	0			
01	1	+1	2	+2
02	0			
10	0			
11	0			
12	1	+1	2	+2
20	1	+1	2	+2
21	0			
22	0			

As shown in Table 4.2, there are only three input combinations where a transformation is applied on the controlled input. For the input combinations for which the output

should be 1 or 2, the controlling inputs must be shifted to 2 by applying the appropriate Z-transformation ($Z(+1)$ or $Z(+2)$). The shifted values are then used to trigger the Z-transformation ($Z(+1)$ or $Z(+2)$) of the ternary controlled-controlled gate and shift the controlled input to generate $K = 1$ or $K = 2$ at the output. Figure 4.4 shows the realization of this gate.

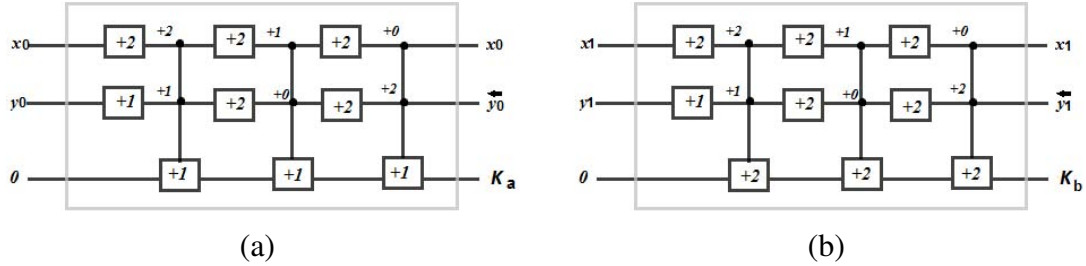


Figure 4.4: Internal structure of the E gates (a) E_a , and (b) E_b .

For example, if the input combination is $x = 0$ and $y = 1$, transformations $+2$ and $+1$ are needed to shift both the input values to 2. Thus a $(+2)$ gate is placed along the x input and a $(+1)$ gate is placed along the y input as shown in Figure 4.4 (a). The outputs of these gates are used to trigger the Z-transformation placed along the controlled input. Other transformations are applied by placing appropriate gates along x and y and the controlled input in a similar way. To clarify the example we show in Figure 4.4 the effective transformations at the controlling points. In Figure 4.4 the reader can see that the number of M-S gates required to realize an E gate is 9.

4.3.3 Architecture of the ternary rail checker circuit

The controlling inputs (x, y) of the two E-gates comprise the four inputs of the rail checker circuit and the controlled outputs K_a and K_b comprise the two outputs of the rail checker. Each controlled input is a constant input set to the value zero. One each of E_a and E_b are used to construct the ternary rail checker.

The first E-gate (E_a) receives x_0 and y_0 as its inputs and generates $K_a = 1$. This is generated by the $Z(+1)$ transformation on the controlled input, but only if $y_0 = \overrightarrow{x_0}$.

The second E-gate (E_b) that takes x_1 and y_1 as inputs generates $K_b = 2$ at the output by applying the $Z(+2)$ transformation on the controlled input in the case where $y_1 = \overrightarrow{x_1}$.

Figure 4.5 shows the block diagram of the internal architecture of the ternary rail checker.

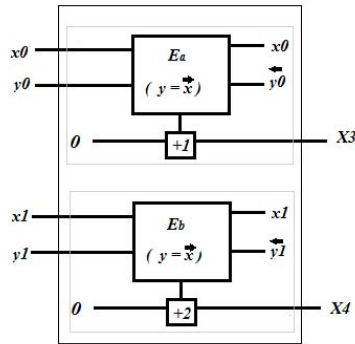


Figure 4.5: Internal architecture of the two-pair two-rail checker.

Figure 4.5 shows the internal design of the ternary rail checker. From this figure we can see that since error detecting signals X_3 and X_4 are generated out of two physically separated E-gates any single bit error in internal lines will affect one of the two outputs, but not both. As shown in Table 4.2 any single bit error in the internal lines will result in generating non-successor outputs and thus will signal the occurrence of an error.

The operation of the rail checker can be verified by examining all possible input combinations of the blocks. Let us define the state of an E-gate to be “True” if the inputs of that gate are successors of each other and “False” otherwise. Table 4.3 shows the truth table for all possible input states and the corresponding outputs for E_a and E_b .

Table 4.3: Truth table for the possible input states and the corresponding outputs of E_a and E_b .

E_a	E_b	X3	X4	Successors
True	True	1	2	Yes
True	False	1	0	No
False	True	0	2	No
False	False	0	0	No

Table 4.3 demonstrates that the rail checker generates $X3 = 1$ and $X4 = 2$, (*i.e.* $X4 = \overrightarrow{X3}$) at the output only when both E_a and E_b have the true state *i.e.* both the input sets have successive ($y_0 = \overrightarrow{x_0}$ and $y_1 = \overrightarrow{x_1}$) values. In all other cases, the rail checker circuit generates $X4 \neq \overrightarrow{X3}$. The number of M-S gates required to realize the ternary two-pair two-rail checker is 18 and the rail checker has four garbage outputs.

4.4 Sample Design

We implement the GFSOP expression $F = ab \oplus cd$ to demonstrate that the proposed blocks in this work can successfully implement a ternary GFSOP. Figure 4.6 shows the realization of function F using the proposed online testable ternary reversible blocks. Use of a variable more than once is implemented by using duplicating circuits which can also be realized by cascading multiple TR blocks. The final outputs of the second rail checker can be used for cascading if the function needs to be further extended.

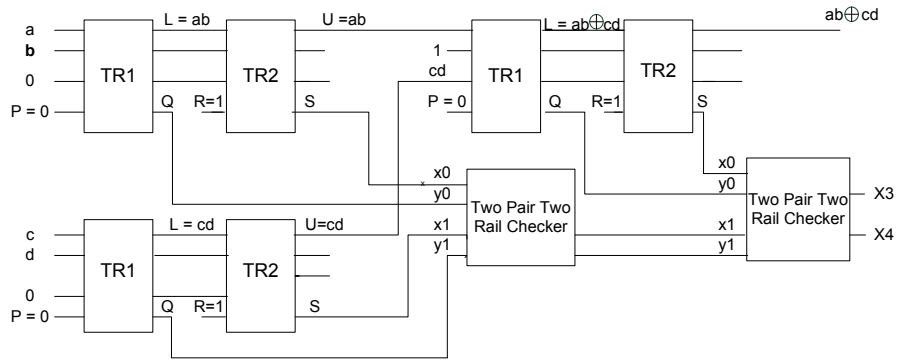


Figure 4.6: Online testable ternary reversible implementation of function $ab \oplus cd$.

4.5 Summary

No other design for a ternary rail checker has been found in the literature, thus we present this design as the first proposal for a ternary reversible rail checker. Use of a rail checker is sometimes unavoidable, as they are required for cascading and error propagating. Therefore, designing a novel ternary rail checker is a noteworthy contribution of this research.

Chapter 5

Implementation and Improvement

5.1 Introduction

In this chapter we introduce approaches for implementing the $TR1$ and $TR2$ blocks using ternary Feynman, Toffoli and 1-qutrit permutative gates. Various approaches discussed in this chapter illustrate how we improved the design in various ways from our first attempt.

5.2 Internal designs

5.2.1 The $TR1$ block

Ternary Toffoli and Feynman gates are used in the initial $TR1$ design. The internal architecture of the $TR1$ block is shown in Figure 5.1. One major highlight of the design is the distinctive use of one copy of the inputs to generate Q and the other copies for L , M and N outputs. This is to avoid propagating erroneous input to circuits for calculating both $\{L, M, N\}$ and Q , since that may cause failure in the error detection.

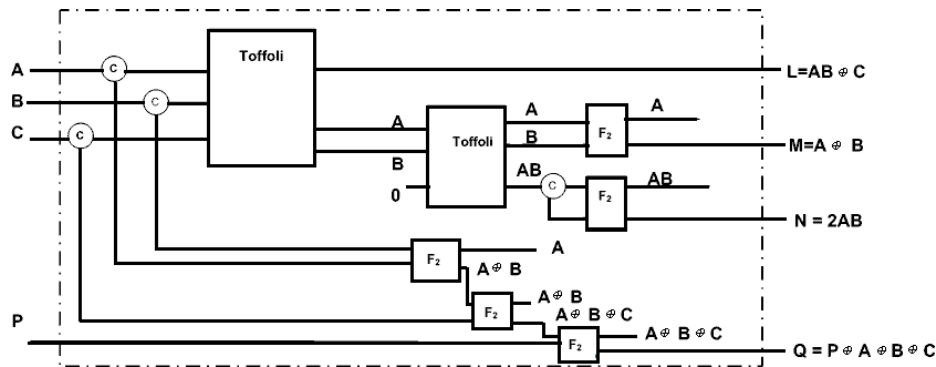


Figure 5.1: Internal diagram of the $TR1$ block.

The Feynman gates used in the $TR1$ block can be implemented by M-S gates as de-

scribed in Chapter 2. The reader can see that the output functions of the TR1 block have one common minterm, AB . We would normally use a straightforward ternary Toffoli gate to implement this, but the basic ternary Toffoli gate will not operate in the manner we desire if the inputs are changed arbitrarily. Therefore, we need to build a version of the Toffoli gate that will always provide the minterm AB even when the inputs are changed arbitrarily. Generalized Toffoli gates (GTG) can generate minterms for different input combinations, although not if the inputs are changed arbitrarily. Therefore, the following cascade of generalized Toffoli gates (GTG) is designed for this use, and is shown in Figure 5.2.

We use this cascade of GTGs to generate the minterm AB . Table 5.1 shows the truth table for two ternary inputs, A and B , and the desired output for the minterm AB , using GF3 multiplication.

Table 5.1: Truth table of AB .

A	B	AB
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	2
2	0	0
2	1	2
2	2	1

The truth table shows that the output AB is 1 only for the input combinations $\{1,1\}$ and $\{2,2\}$, the output is 2 for the combinations $\{1,2\}$ and $\{2,1\}$ and 0 for the rest of the input combinations. The GTGs are used to realize the cascade of input combinations for which the output AB should be non-zero. The controlled input C is initialized to 0. This value will be passed through to the outputs unchanged for all other input combinations.

Figure 5.2 shows the realization for the function AB . In the GTGs only control values

of 2 trigger a Z-transformation, thus shifts are needed to change the inputs for the desired functionality. The internal structure and the computation of the gate count for each GTG are discussed in Chapter 2. We use the symbol showing the values of controlling inputs of each GTG in Figure 5.2.

The cost to realize this cascade of GTGs can be easily calculated:

- the first generalized Toffoli gate has no non-2 controlled input; hence it requires $5 + 2 * (\text{number of non-2 controlled inputs}) = 5 + 2 * 0 = 5$ gates;
- the second generalized Toffoli gate has one non-2 controlled input; hence it requires $5 + 2(1) = 7$ gates; and lastly,
- the third and fourth generalized Toffoli gates require 7 and 9 M-S gates correspondingly.

Therefore, the total number of M-S gates required to realize this cascade of GTGs is 28.

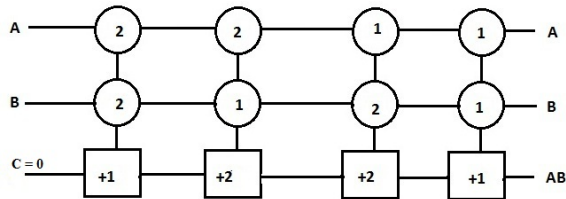


Figure 5.2: Realization of the cascade of GTGs having output AB .

Now that we have generated the minterm AB , the outputs A and B of the constructed Toffoli gate can be reused to generate outputs $A \oplus B$ and $2AB$ as shown in Figure 5.1. There are two 3×3 Toffoli gates and nine 2×2 Feynman gates in the initial $TR1$ block. The 3×3 Toffoli gates are implemented by the cascade of GTGs. As shown in Table 5.2 the number of M-S gates required for one 3×3 Toffoli gate is 28 and for one 2×2 Feynman gate is 4 [15]. Hence the total number of M-S gates required to construct the $TR1$ block is $(28 * 2) + (9 * 4) = 92$. The outputs A , AB , $A \oplus B$ and $A \oplus B \oplus C$ are the outputs which

are neither the desired outputs nor the restoration of any input. They are only required to maintain the reversibility and as such they are considered garbage outputs. The number of garbage outputs is 4 in this design. It can be argued that the first output A is not a garbage output since the initial input is A and it is passed unchanged as an output [7, 20].

5.2.2 The $TR2$ block

Three 2×2 Feynman gates are cascaded to implement the $TR2$ block as shown in Figure 5.3. The realization of the $TR2$ block is shown in Figure 5.4. In the $TR2$ block, each

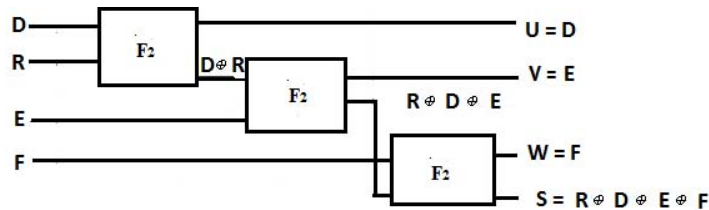


Figure 5.3: Internal diagram of the $TR2$ Block.

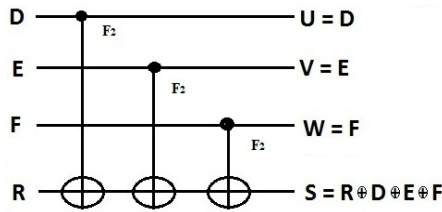


Figure 5.4: Realization of the $TR2$ Block.

Feynman gate passes the input of $TR2$ unchanged through to the output. The Feynman gates also generates the final output $S = R \oplus D \oplus E \oplus F$. If any of the inputs is changed due to an error, it will be reflected in the value of S and the error can be detected if $S \neq \vec{0}$.

Table 5.2: Cost of the ternary gates used to realize the proposed blocks.

		cost
Basic Gates	GTG with no non-2 controlled input	5
	GTG with one non-2 controlled input	7
	GTG with two non-2 controlled input	9
	1-qutrit permutative gate	1
	2×2 Feynman Gate	4
	Shift gate	1
Constructed gates	Cascade of GTGs	28

5.3 Upgraded design of the $TR1$ block

The design of the $TR1$ block can be simplified to reduce the cost (the number of M-S gates). The Feynman gate is always less costly than the Toffoli gate in terms of the number of M-S gates and so using the Toffoli gate only to generate the common minterm AB will result in cost savings. The outputs (A, B and AB) of the Toffoli gate can be reused to generate $A \oplus B$, $2AB$ and $AB \oplus C$. Output AB must be copied by a Feynman gate because of the fan-out limitation in reversible logic. One of the copies of AB is EXORED with C using a ternary Feynman gate to generate $AB \oplus C$ and the other copy is passed through a dual shift gate to generate $2AB$. Since only one Toffoli gate has been used and the remaining gates are Feynman gates and dual shift gates, the cost will reduce dramatically. Figure 5.5 shows the new design of the $TR1$ block.

One Toffoli gate, eight Feynman gates and one dual shift gate are required for the new design. The cost to realize this design is $28 + (8 * 4) + 1 = 61$. The outputs $A, A \oplus B$ and $A \oplus B \oplus C$ are considered to be garbage outputs. The number of garbage outputs is reduced to 3 in this design. Hence the cost is reduced by 33.7% and the garbage is reduced by 25%.

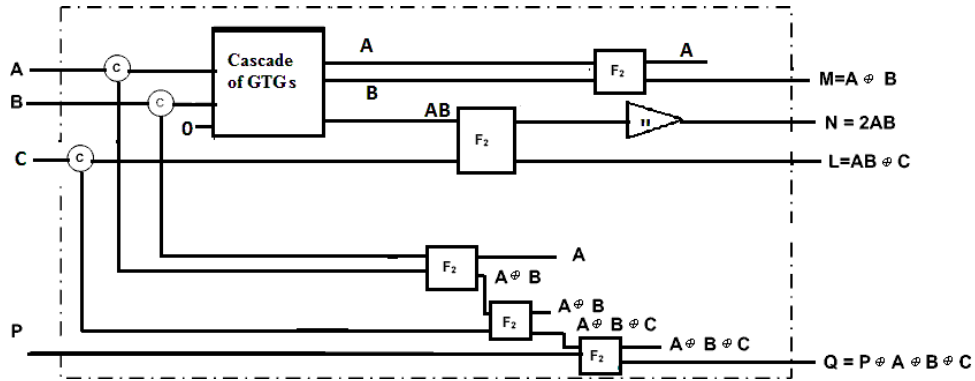


Figure 5.5: Upgraded internal design of the $TR1$ block.

We can further reduce the cost by excluding the copy gates and reorganizing the Feynman gates as shown in Figure 5.6. The cost to realize this design is $28 + (5 * 4) + 1 = 49$. The reduction of cost is 46.7% compared to the first design and 19.67% compared to the second design. However the most significant achievement of this design is the number of garbage outputs, which is zero for this design. However it should be mentioned here that when this block is used to realize a ternary circuit, some of the outputs of the TR block may become garbage outputs if they are not used in any further operation.

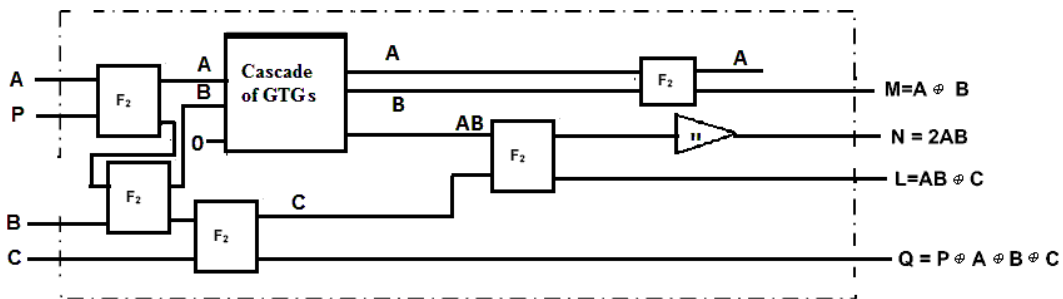


Figure 5.6: Further upgraded design of the $TR1$ block.

Figure 5.7 shows the cascade of gates to form the $TR1$ block using one cascade of GTGs and Feynman gates for the new design.

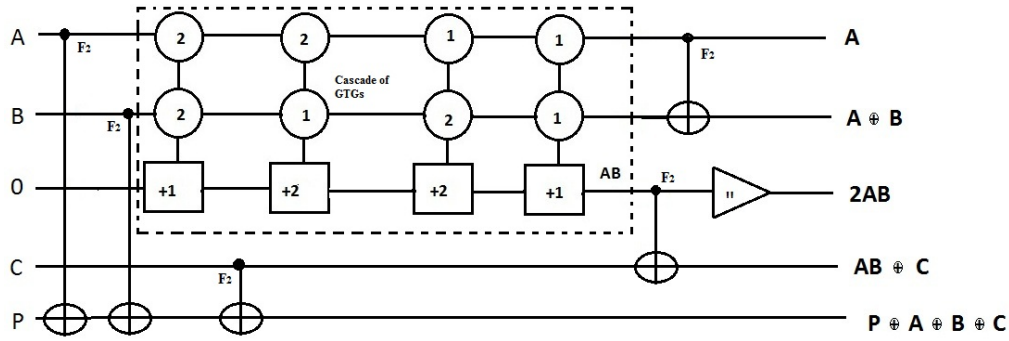


Figure 5.7: Realization of the $TR1$ block.

5.4 The TR block

The $TR1$ block and the $TR2$ block are cascaded together to construct the TR block. Figure 5.8 shows the internal diagram of the TR block and Figure 5.9 shows its realization. Since two 4×4 ternary blocks are cascaded together using their first three inputs, the resultant TR block is a 5×5 testable block. We will refer to this TR block as $4TR$ since this block has been constructed from two 4×4 blocks. The number of M-S gates required to implement the $TR1$ block is 49 and the number required for the $TR2$ block is 12. Hence, the total number of M-S gates required to implement the $4TR$ block is $(49 + 12) = 61$.

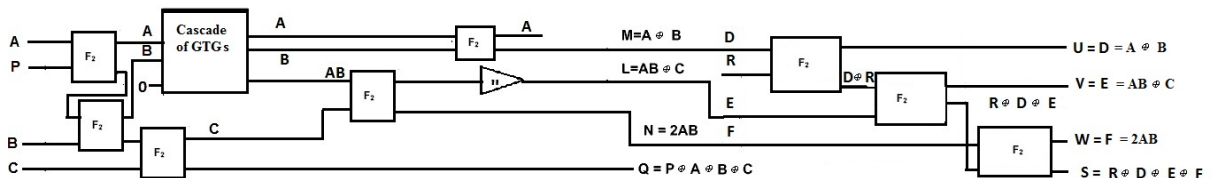


Figure 5.8: Internal diagram of the TR block.

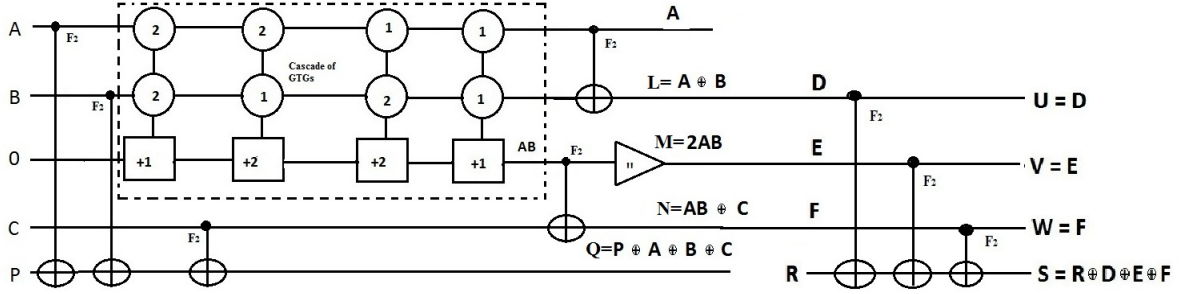


Figure 5.9: Realization of the TR block.

5.5 Limitation of the $4TR$ block

Implementing the benchmark circuits from [2] for ternary GF3 operations exposes a problem in the current $4TR$ design. Since fan out is limited to one in reversible logic, generating copies of inputs is essential. In the current design of $TR1$, generating a copy of an input is possible but the overhead cost is huge in terms of the required number of M-S gates. The following discussion presents how a copy function can be implemented using $4TR$ and the cost of this operation.

5.5.1 Implementing copy function using the $4TR$ block

In ternary GF3 logic, adding 3 to a variable leaves the variable unchanged, *e.g.*, $A \oplus 3 = A \oplus 1 \oplus 1 \oplus 1 = A$. Let us assume that a copy of the input A is required. If inputs B and C are set to 1, then we have $U = A \oplus 1$ and $V = A \oplus 1$ at the outputs of a $4TR$. If we repeat this process thrice using three TR blocks, each time providing $A = A \oplus 1$ from the previous block and $B = C = 1$, at the end of the third operation we have $U = A \oplus 1 \oplus 1 \oplus 1 = A$ and $V = A \oplus 1 \oplus 1 \oplus 1 = A$. Thus we have produced a copy of input A . Figure 5.10 shows the

configuration.

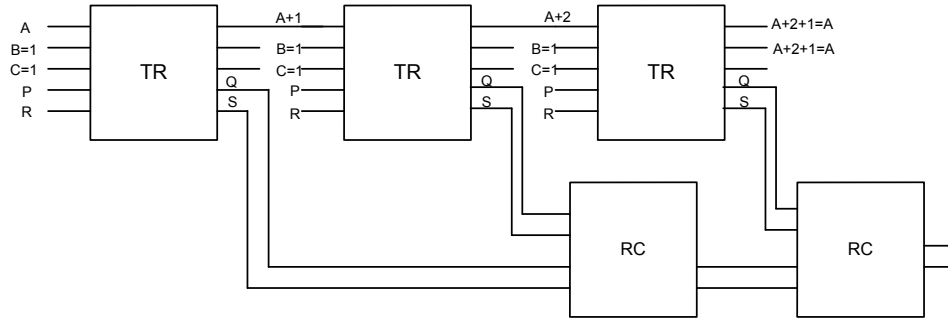


Figure 5.10: Copy operation using the $4TR$ block.

In this design five $4TR$ blocks and two Rail Checkers (RC) are required to generate a single copy. Each TR block requires 61 M-S gates and each RC requires 18 M-S gates. Therefore, the number of gates required for a single copy operation is $3 * 61 + 2 * 18 = 219$ which is huge. This inefficiency of the $4TR$ block can be improved in four different ways, as discussed in the following sections.

5.6 Reducing the cost of the copy function

The copy function can be implemented using the four approaches discussed in this section. The primary objective of each approach is to reduce the number of M-S gates required to implement a ternary function.

5.6.1 Method 1: 5×5 TR block ($5TR$)

The 4×4 $TR1$ block can be changed into a 5×5 block by adding an additional constant input 0 and an output $O = A$. The output function Q will also be changed. To distinguish between the 4×4 $TR1$ block and the 5×5 $TR1$ block in this work we refer to them as

4TR1 and 5TR1 correspondingly. The 4×4 TR2 block is addressed as 4TR2 in the rest of the work. It can be verified from Table 5.3(a) that all the properties of the 4TR1 block have been preserved in the 5TR1 block. Figure 5.11 shows the block diagram of the 5TR1 block.

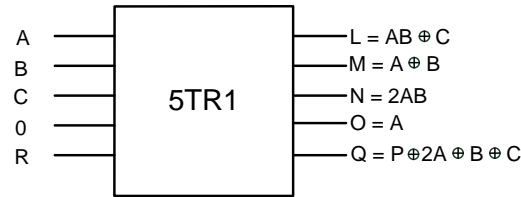


Figure 5.11: 5×5 TR1 block.

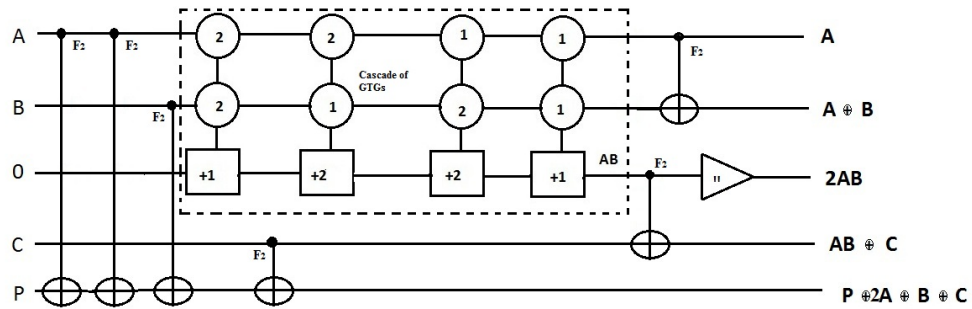


Figure 5.12: Internal design of 5×5 TR1 block.

The internal design of the 4TR1 block must be changed somewhat since the output function Q is changed to

$$\begin{aligned}
 Q &= P \oplus L \oplus M \oplus N \oplus O \\
 &= P \oplus AB \oplus C \oplus A \oplus B \oplus 2AB \oplus A \\
 &= P \oplus 2A \oplus B \oplus C
 \end{aligned}$$

One way to realize $2A$ in Q is to use two Feynman gates to add $2A$ to P , but that will increase the number of M-S gates in the 5TR1 block. Figure 5.12 shows the internal design

of *5TR1* using one additional Feynman gate. The number of M-S gates required for this design of *5TR1* is 53, however intelligent use of Feynman gates can reduce the number of M-S gates.

Table 5.3: A subset of the truth tables for (a) *5TR1* Block and (b) *5TR2* Block.

Input					Output				
A	B	C	0	P	L	M	N	O	Q
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	1
0	0	2	0	0	2	0	0	0	2
0	1	0	0	0	0	1	0	0	1
0	1	1	0	0	1	1	0	0	2
0	1	2	0	0	2	1	0	0	0
0	2	0	0	0	0	2	0	0	2
0	2	1	0	0	1	2	0	0	0
0	2	2	0	0	2	2	0	0	1
1	0	0	0	0	0	1	0	1	2
1	0	1	0	0	1	1	0	1	0
1	0	2	0	0	2	1	0	1	1
2	0	0	0	0	0	2	0	2	1
2	0	1	0	0	1	2	0	2	2
2	0	2	0	0	2	2	0	2	0
1	1	0	0	0	1	2	2	1	0
1	2	0	0	0	2	0	1	1	1
1	1	1	0	0	2	2	2	1	1
1	1	2	0	0	0	2	2	1	2
1	2	1	0	0	0	0	1	1	2
1	2	2	0	0	1	0	1	1	0
2	1	0	0	0	2	0	1	2	2
2	1	1	0	0	0	0	1	2	0
2	1	2	0	0	1	0	1	2	1
2	2	0	0	0	1	1	2	2	0
2	2	1	0	0	2	1	2	2	1
2	2	2	0	0	0	1	2	2	2

(a)

Input					Output				
D	E	F	G	R	U	V	W	X	S
0	0	0	0	1	0	0	0	0	1
1	0	0	0	1	1	0	0	0	2
2	0	0	0	1	2	0	0	0	0
0	1	0	0	1	0	1	0	0	2
1	1	0	0	1	1	1	0	0	0
2	1	0	0	1	2	1	0	0	1
0	2	0	0	1	0	2	0	0	0
1	2	0	0	1	1	2	0	0	1
2	2	0	0	1	2	2	0	0	2
0	1	0	1	1	0	1	0	1	0
1	1	0	1	1	1	1	0	1	1
2	1	0	1	1	2	1	0	1	2
0	2	0	2	1	0	2	0	2	2
1	2	0	2	1	1	2	0	2	0
2	2	0	2	1	2	2	0	2	1
1	2	2	1	1	1	2	2	1	1
2	0	1	1	1	2	0	1	1	2
2	2	2	1	1	2	2	2	1	2
0	2	2	1	1	0	2	2	1	0
0	0	1	1	1	0	0	1	1	0
1	0	1	1	1	1	0	1	1	1
2	0	1	2	1	2	0	1	2	0
0	0	1	2	1	0	0	1	2	1
1	0	1	2	1	1	0	1	2	2
1	1	2	2	1	1	1	2	2	1
2	1	2	2	1	2	1	2	2	2
0	1	2	2	1	0	1	2	2	0

(b)

The number of M-S gates can be kept unchanged at 49 by modifying the structure of the ternary Feynman gate to generate $P \oplus 2A$ instead of $P \oplus A$ for inputs P and A . This

modified Feynman gate can be used to add $2A$ to output $Q = P \oplus 2A \oplus B \oplus C$. Figure 5.13 (a) show the structure and Figure 5.13 (b) shows the symbol of the modified Feynman gate. We refer to this modified Feynman gate as the MF gate.

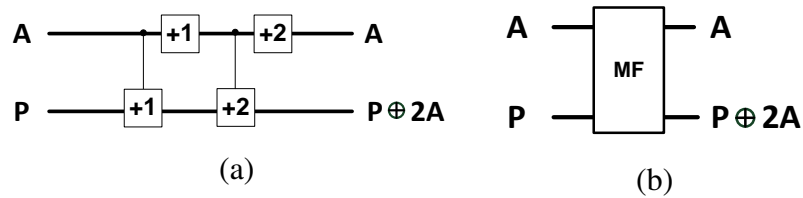


Figure 5.13: (a) Internal design of the modified Feynman (MF) gate and (b) symbol of the modified Feynman (MF) gate.

Table 5.4: Truth table for the MF gate.

Inputs		Outputs	
A	P	A	$P \oplus 2A$
0	0	0	0
0	1	0	1
0	2	0	2
1	0	1	2
1	1	1	0
1	2	1	1
2	0	2	1
2	1	2	2
2	2	2	0

The behavior of the MF gate can be verified from the truth table shown in Table 5.4. The internal structure of the *5TR1* block using a MF gate is shown in Figure 5.14. Copying can be achieved by setting A to the value we wish to copy and $B = 0$. The number of M-S gates required for this design of *5TR1* is 49 and thus the number of M-S gates remains unchanged compared to *4TR1*.

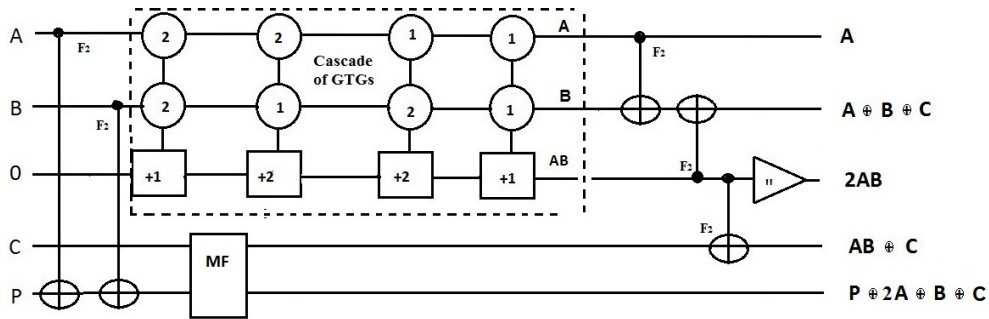


Figure 5.14: Internal design of the 5×5 $TR1$ block using the MF gate.

Table 5.3 shows the all possible input combinations and outputs of $5TR1$. Since $5TR1$ now turns into a 5×5 block, $TR2$ must also be changed into a 5×5 block to receive the new input $O = A$ from the $5TR1$ block. An extra ternary Feynman gate can be added to incorporate the new input into the error checking output. Hence, the new 5×5 $TR2$ block ($5TR2$) can be implemented using 16 M-S gates. Figures 5.15 and 5.16 show the block and internal diagrams for the $5TR2$ block.

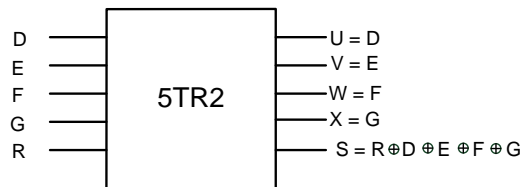


Figure 5.15: $5TR2$ block.

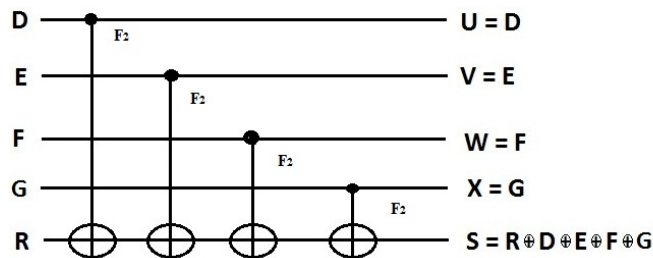


Figure 5.16: Internal design of the $5TR2$ block.

Figure 5.17 shows the new TR block consisting of $5TR1$ cascaded with $5TR2$ and addressed as the $5TR$ block. The total number of M-S gates required to construct this $5TR$ block is $(49 + 16) = 65$.

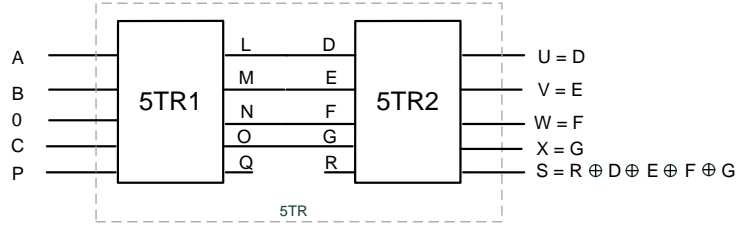


Figure 5.17: 5×5 TR (5TR) block.

5.6.2 Method 2: Combination of 4TR block and non-testable ternary Feynman gate

Another approach to implement the copy function is to use a ternary Feynman gate for the copy operation, which requires only 4 extra M-S gates for each copy operation. For all other ternary operations we will use a 4TR1 block. Although we can use a 5TR block for a copy operation, using a ternary Feynman gate will reduce the number of M-S gates dramatically. But this will also affect the testability feature since a ternary Feynman gate is not online testable. The probability of detecting a single bit fault in the circuit will be decreased. If any error occurs while copying using the non-testable ternary Feynman gates, the error will not be detected.

5.6.3 Method 3: Combination of 5TR block and online testable Copy gate (TR_c)

The limitation of Method 2 where a non-testable Feynman gate is used for the copy operation can be resolved by designing a testable block designated for copy operation. In other words, we can design an online testable block using ternary Feynman gates exclusively for the copy operation. This block can be used for generating copies of a single input but will also incorporate the online testability feature. In a 5×5 block, the maximum number of

outputs excluding the error checking output is four. Therefore four copies of a single input can be generated using the online testable copy gate. However, generating four copies will increase the number of M-S gates as well as the number of garbage outputs if only two copies are required, as in most of the cases. Thus, we have limited our design of TR_{copy} to generate only three copies of the input, although the flexibility of designing a copy gate for four copies still exists. Figures 5.18 and 5.19 show the block and internal design of the online testable copy gate. For the copy operation $5TR1$ is replaced with the new 5×5 TR_{copy} block and cascaded with a $5TR2$ block for the online testability feature. The new $5TR$ block constructed from TR_{copy} and $5TR2$ is referred as TR_c which generates three copies of a single input.

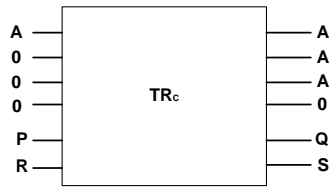


Figure 5.18: 5×5 TR_c block.

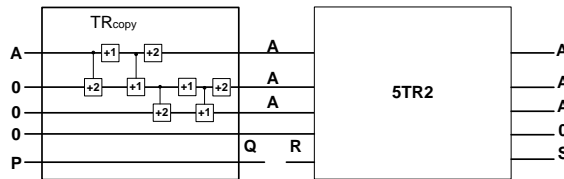


Figure 5.19: Internal design of the 5×5 TR_c block.

The operation of TR_{copy} is very simple. Two 2×2 ternary Feynman gates are used to generate the copies. Similar to the $5TR1$ block, output Q is the sum of P and the other outputs of TR_{copy} . Since in ternary Galois Field logic $3A = 0$, we have the following:

$$\begin{aligned}
Q &= P \oplus A \oplus A \oplus A \\
&= P \oplus 3A \\
&= P \oplus 0 \\
&= P
\end{aligned}$$

Hence P is directly passed through to Q . The error detection policy is identical to that used in both the $4TR$ and $5TR$ blocks. TR_{copy} requires 8 and $5TR2$ requires 12 M-S gates. Therefore, a TR_c block requires 24 M-S gates in total. Hence if a TR_c block is used to make a copy of a single input instead of using multiple $4TR$ blocks, only 24 M-S gates are required compared to 219 M-S gates. Moreover, a TR_c block generates three copies of an input which will minimize the number of copy gates required in a large function.

5.6.4 Method 4: Combination of 5TR block, TR_c and online testable Multi Copy gate (TR_{mc})

Sometimes copies of multiple variables are required. For example, two copies of A , two copies of B and two copies of C are required for the benchmark $3CyG2$ which is defined as $ab \oplus bc \oplus ca$. We need three TR_c blocks for the copy operation to be implemented for this benchmark. To avoid this situation the design of TR_{copy} can be modified to generate two copies of two different variables. Figures 5.20 and 5.21 show the configuration. We refer to this multi copy gate as $TR_{multicopy}$. 16 M-S gates are required to realize this gate. Online testability is incorporated by cascading the $TR_{multicopy}$ and the $5TR2$ block. We

refer to the new cascaded block as TR_{mc} . Therefore, a TR_{mc} block requires 32 M-S gates in total to generate one copy for each of the two variables whereas TR_C would require 48 M-S gates to perform the same operation. Although the number of M-S gates is increased in TR_{mc} , this block can reduce the number of M-S gates to 33% where copies for multiple variables are necessary. In the 3CyG2 benchmark function, we require $24 * 3 = 72$ M-S gates to implement the copy functions, whereas 56 M-S gates if one TR_C and one TR_{mc} are used. Again, in another benchmark circuit, 4CyG2, which is defined as $ab \oplus bc \oplus cd \oplus da$, $24 * 4 = 96$ M-S gates are required if TR_C is used whereas implementation using two TR_{mc} requires only 64 M-S gates. For the best result we can use a combination of TR_C and TR_{mc} blocks according to the circuit's behavior.

The operation principle of TR_{mc} is identical to that used in the $4TR$, $5TR$ and TR_C blocks. We set $P = 0$ and $R = 1$ in TR_{mc} and at the end of the operation S being the successor of Q represents the fault-free operation.

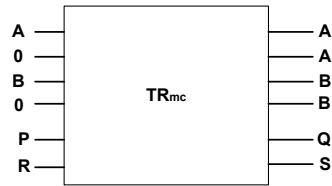


Figure 5.20: 5×5 TR_{mc} block.

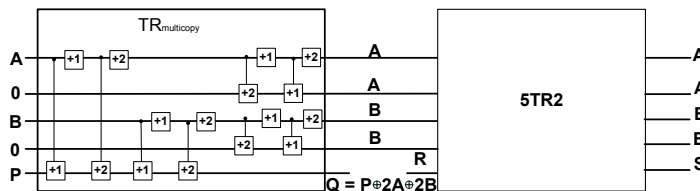


Figure 5.21: Internal design of 5×5 TR_{mc} block.

Table 5.5: Number of M-S gates required to implement the proposed blocks

Blocks	$4TR$			$5TR$			TR_c			TR_{mc}			RC
	$4TR1$	$4TR2$	Total ($4TR$)	$5TR1$	$5TR2$	Total ($5TR$)	TR_{copy}	$5TR2$	Total (TR_c)	$TR_{multicopy}$	$5TR2$	Total (TR_{mc})	
Number of M-S gates	49	12	61	49	16	65	8	16	24	16	16	32	18

5.7 Summary

In this chapter we have discussed the implementation of the $4TR1$ and $4TR2$ blocks as well as their limitation. We have also proposed here four different approaches to reduce the cost of implementing the copy function. The testable blocks designed in this work use basic ternary building blocks to incorporate the online testing feature which can construct complex and larger circuit blocks, although a priority is to keep the number of gates and garbage values at a minimum. Table 5.5 shows the number of M-S gates required to implement the proposed blocks. To keep the number of garbage values to a minimum, the garbage output of one ternary gate is used as an input to another gate. Because of the fan-out limitation of reversible logic circuits, Feynman gates are used to create copies of variables where needed.

Chapter 6

Comparison and Analysis of Different Approaches

6.1 Introduction

The methods proposed in Chapter 5 exhibit different performance for different types of ternary benchmarks. In this chapter some benchmark circuits are implemented using the proposed approaches and comparisons among them are presented in terms of the overall number of M-S gates required to implement the benchmarks. We also compare the cost of the proposed approaches with the cost of implementing the benchmarks using non-testable ternary gates.

6.2 Comparison among the proposed methods and non testable circuits

Table 6.1 presents the number of M-S gates, which we are using as our cost metric, required to realize the benchmark circuits [2] using the five design methods discussed so far. Since the concept of an online testable reversible ternary circuit is first proposed in this thesis, no other testable designs are available for comparison. Hence, the cost of implementation using the online testable blocks are compared with the cost to design the benchmark circuits with non testable ternary gates as discussed in [14]. The comparison is presented in Table 6.1. The cost includes the number of M-S gates required to implement the testable blocks as well as the rail checkers. The realization of the benchmarks *3CyG2*, *ProdG3* and *SumG3* using the five proposed methods and non-testable ternary gates are shown in Appendix A.

Table 6.1: Comparison of number of M-S gates for implementing the ternary benchmark circuits.

Benchmarks	Proposed Methods					Non-testable ternary gates	Lowest overhead
	Method 0 4TR & RC	Method 1 5TR & RC	Method 2 4TR, Feynman & RC	Method 3 5TR, TR_c & RC	Method 4 5TR, TR_c , TR_{mc} & RC		
2CyG2 ($2ab$)	61	65	61	65	65	37	164%
3CyG2 ($ab + bc + ca$)	1088	646	389	523	489	92	531%
a2bccG ($a^2 + bc + c$)	772	480	306	398	364	96	379%
ProdG2 (ab)	61	65	61	65	65	28	217%
ProdG3 (abc)	140	148	140	148	148	56	250%
SumG2 ($a \oplus b$)	61	65	61	65	65	4	1525%
SumG3 ($a \oplus b \oplus c$)	140	148	140	148	148	8	1750%

Table 6.1 shows that the fewest of M-S gates are required when using the non-testable ternary gates. This is unsurprising because incorporating testability features always adds some overhead to the circuit. However testable circuits are far more robust and fault tolerant than the non-testable circuits; hence, the tradeoff between the overhead and testability is justifiable.

In Method 0, 4TR and RC blocks are used to implement the benchmarks. In this method 4TR blocks are also used to copy variables, thus the overhead of using Method 0 is very high. For example, 1088 M-S gates are required to realize the benchmark 3CyG2. However whenever no copy is required, Method 0 uses the lowest number of gates in the testable design such as for benchmarks *ProdG2* and *SumG2*.

Method 1 uses 5TR blocks and rail checkers to realize the benchmarks. This method does not require a separate block for copying as in Methods 3 and 4, and the cascades of 5TR blocks are sufficient to implement any ternary benchmark function represented by a GF3 sum of product. It can be seen from Table 6.1 that Method 1 requires more M-S gates than the other method for benchmarks 2CyG2, 2CyG3, and a2bccG but almost equal numbers of gates for the remaining benchmarks.

Apart from the non-testable designs, Method 2 described in Section 5.6.2 requires the

lowest number of M-S gates. Unfortunately this method does not assure online testability when copies are required, since the ternary Feynman gates used in these designs are not testable. Hence, a function which requires copies will not be completely online testable if realizing using Method 2. Table 6.1 illustrates that Method 2 requires the lowest number of M-S gates for benchmarks $2CyG2$, $3CyG2$ and $a2bccG$ and equal or fewer gates as compared to other methods for the remaining benchmarks.

Method 3 described in Section 5.6.3 generates the best result whenever the circuit requires more than two copies of an input variable. The design of TR_c can be easily upgraded to generates 4 copies of a single input variable, although the upgraded design will require 8 more additional M-S gates. If the function consists of several variables which require 4 copies each, the upgraded design would be the best solution. The proposed design is the best for functions consisting of variables which require at most 3 copies each. However Method 3 requires a higher number of M-S gates than Method 4 for benchmarks $3CyG2$ and $a2bccG$, each of which requires at most 2 copies of each input variables. The cost is the same as for other methods for the remainder of the benchmarks.

Among the designs that provide complete online testability, Method 4 described in Section 5.6.4 requires the lowest number of M-S gates. This method is the most efficient since it uses both TR_c and TR_{mc} to achieve the best result. This method is most efficient if the design requires at most two copies of the input variables and also requires copies for a large number of variables. This can be easily verified from Table 6.1 where Method 4 requires a lower number of M-S gates for benchmarks $2CyG2$, $2CyG3$ and $a2bccG$ and an equal number of gates as compared to Method 3 for the remaining benchmarks.

It can also be noticed from Table 6.1 that the $4TR$ block requires the lowest number of M-S gates compared to the other testable methods for the benchmark circuits that do not require any copy operation.

6.3 Overhead analysis

In this work overhead can be approximated as the total number of extra elementary gates required to be added for the realization of the ternary online testable circuit as compared to the original gate count of the non-testable realization of the same circuit. The methods discussed in this work are the first ever proposed approaches in the area of ternary online testing and hence the overhead is compared with the binary online reversible testing approaches discussed in [27]. The cost metric used in both the binary and ternary online testing approaches is the number of elementary gates required to realize a logic function. Table 6.2 shows the comparison of average overheads among the two binary and our proposed ternary online testing approaches.

Table 6.2: Overhead comparison.

Approaches	Average overhead for testability
[36]	312.28% [27]
[4]	388.67% [27]
Our approach	688%

Although the average overhead of our approach is higher than the other two binary approaches, the following fact should be considered:

- From the last column of Table 6.1 we can see that the overhead costs for the benchmarks *SumG2* and *SumG3* are significantly higher than the other overheads. These costs are significant in increasing the average overhead of our approach. If we calculate the average overhead excluding the benchmarks *SumG2* and *SumG3*, the average overhead for our approach is lowered down to 308%. We note that the non-testable approaches for benchmarks *SumG2* and *SumG3* required less than 8 gates whereas the minimum number of gates required for a ternary online testable design is 61. Therefore, we can say that our approach is likely to minimize the overhead costs for

larger circuits.

We also briefly mention area overhead in current technologies. Nepal *et al.* in [25] discussed the area overhead for their three proposed error detection techniques on existing non-reversible binary technology. Figure 6.1 shows their chart presenting the area overheads for their parity code technique [25]. Details of these techniques are out of scope of this thesis, however the chart shows that the highest area overhead can be around 170%. Unfortunately, it is impossible to measure the area overhead for ternary online testable circuits since the technology to be used is still unknown.



Figure 6.1: Area overhead chart [25] .

6.4 Summary

Method 4 generates the lowest possible result in terms of the number of M-S gates in realizing the benchmark circuits. Where multiple copies of variable are needed Methods 2 and 3 provide the best solution. If no copy is required the 4TR block can be used to minimize the cost but using the 5TR block will provide more tolerance and robustness.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this thesis we propose designs for online testable ternary reversible logic blocks. In Chapter 3 two basic testable blocks are used in conjunction, one of which implements ternary logic functions and the other incorporates the online testability feature. As far as we are aware this is the only online testable design currently in the literature for ternary reversible logic. In Chapter 4, we also propose a design for ternary rail checker which can be used in conjunction with multiple ternary online testable blocks to implement any ternary reversible circuit. Our work is based on a single-bit fault model which means that any single error in a block can be detected and will be propagated to the outputs through the use of the rail checkers.

In Chapter 5, to implement ternary circuits we propose five online testable approaches which use combinations of $4TR$, $5TR$, TR_c , TR_{mc} and ternary Feynman gates depending on behavior of the circuit. These approaches improve on the initial design by reducing the number of M-S gates. In Chapter 6, we show comparisons of the five proposed approaches, based on the implementation cost of benchmark circuits from [2]. Fan-out is limited to one in reversible logic, and so we need to generate copies of the inputs which are required more than once in the circuit. In Chapter 5 we propose two online testable copy blocks which will preserve the online testability feature of the circuit as well as reduce the number of gates required for copy operations. We can further upgrade TR_c to generate 4 copies depending on the requirements of the function. Using TR_{mc} intelligently in larger circuits where copies of multiple inputs are required can also prove to be extremely cost-effective.

In our research we also focus on the implementation level to improve the preliminary

design. A number of different modifications to the basic blocks are also suggested, with discussion as to how each modification might best be utilized. Although the overhead is higher than for the Boolean reversible online testable circuits, the overhead is quite reasonable, and in fact our designs require zero garbage lines to realize the ternary online testable blocks.

One of the major concerns of reversible logic synthesis is to keep the number of the input constants as few as possible. In our proposed designs, we have only one constant input. The other two major concerns of reversible logic synthesis are to keep the number of garbage outputs and the length of gate cascades as small as possible. In our designs we minimize the number of garbage outputs by using the internal gates as intelligently as possible. The length of cascaded gates is also kept at a minimum by replacing $4TR$ or $5TR$ blocks with TR_c and TR_{mc} blocks, which are designated for copy operations, whenever required.

7.2 Future Works

As a completely new research area, ternary online testing is yet to be explored. Online testing includes fault detection, finding the point of occurrence of the fault and in some cases fault recovery. Fault localization is the process to identify the first failure node by backtracking the signal. Detecting the failure point is necessary to identify the root cause of the failure. Fault localization research is also going on in binary reversible logic and binary quantum circuits. However, research on fault localization in ternary reversible logic has not yet been addressed. The last aspect of testing is fault recovery which is the most difficult aspect to achieve in testing. In this thesis we focus only on fault detection in ternary reversible logic. Hence, a lot of research opportunities exist in the other fields of online testability.

For binary reversible logic, there exist different fault models. In [31], fault models for binary reversible quantum circuits are discussed, however these fault models are not clearly defined for ternary reversible logic. Our design in this thesis is based on the single bit stuck at fault model and is capable of detecting single bit errors. Designing online testable circuits for different fault models or designing a universal testability approach would be a major research concern in our future work.

In this thesis basic ternary gates (*e.g.* Toffoli, Feynman) are used to realize the proposed online testable blocks. Because of this the blocks require large numbers of constructing gates and connections among them. If it is possible to design a single online testable gate instead of a block with a similar objective, the number of constructing gates would be radically reduced. Thus our future work also includes the design of an online testable reversible ternary gate.

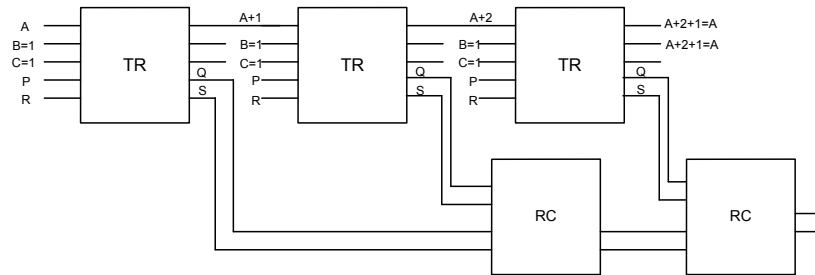
Lastly, although we have presented a comparison among the different proposed approaches, we have used hand computation to realize the benchmark circuits. Therefore, a synthesis algorithm needs to be developed for our design methodology. Since we have proposed multiple blocks, the algorithm must include a heuristic to select the lowest cost alternative block in order to implement large circuits.

Appendix A

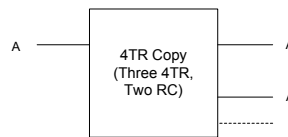
Appendix

A.1 Notation

- The copy function can be implemented using four $4TR$ blocks and three RC s as shown in Figure A.1 (a). Using these $4TR$ blocks and RC s it is possible to generate only one copy of an input. For the simplicity of the diagram we will use the notation shown in Figure A.1 (b) to represent the copy function implemented by $4TR$. 219 M-S gates are required to implement the copy function using the $4TR$ blocks.
- The error detecting outputs Q and S of the TR , $4TR$, $5TR$, TR_c and TR_{mc} blocks are represented by dashed lines. The inputs to the rail checker from a testable block and the outputs of the rail checker are also represented by dashed lines.



(a)



(b)

Figure A.1: (a) Copy using $4TR$ and (b) notation used in the diagrams.

A.2 3CyG2

The following sections illustrate the realization of the benchmark 3CyG2 using the five proposed methods and non-testable ternary gates.

A.2.1 Method 0

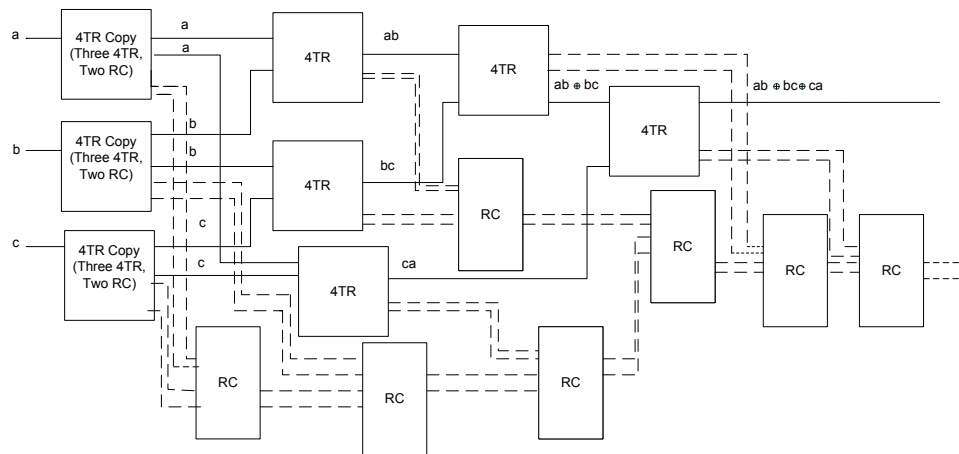


Figure A.2: Realization of the benchmark 3CyG2 using 4TR blocks.

A.2.2 Method 1

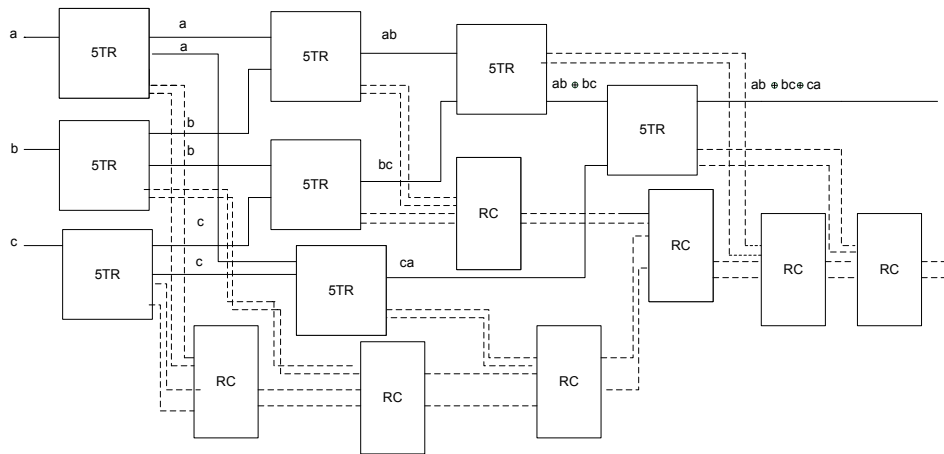


Figure A.3: Realization of the benchmark 3CyG2 using 5TR blocks.

A.2.3 Method 2

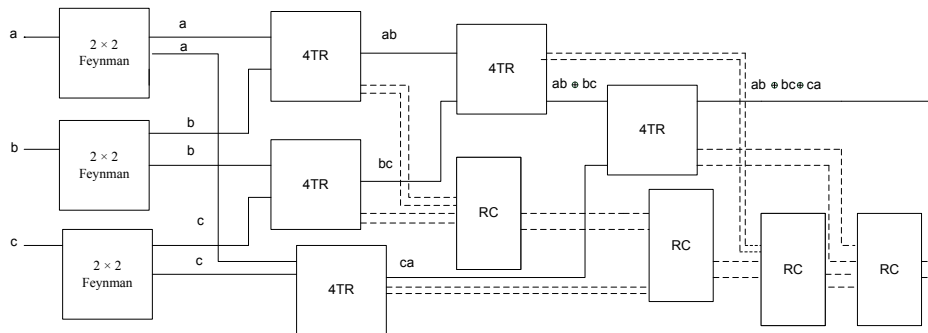


Figure A.4: Realization of the benchmark 3CyG2 using 4TR blocks and Feynman gates.

A.2.4 Method 3

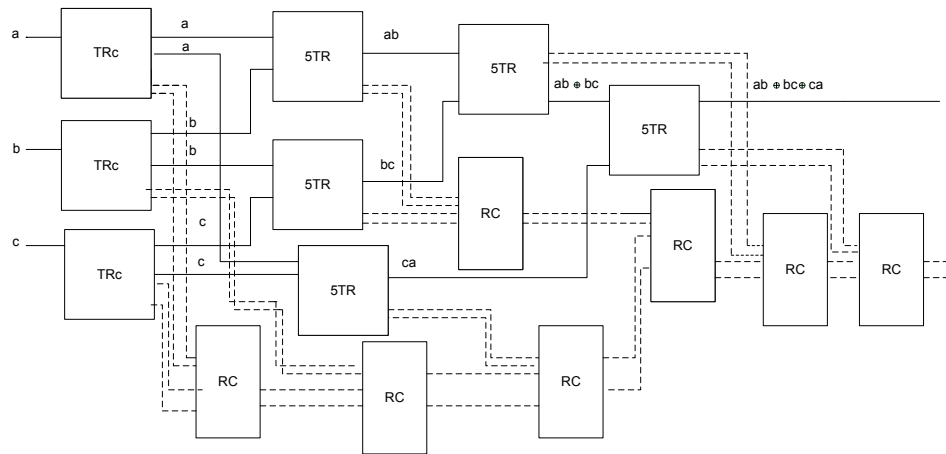


Figure A.5: Realization of the benchmark $3CyG2$ using $5TR$ and TR_c blocks.

A.2.5 Method 4

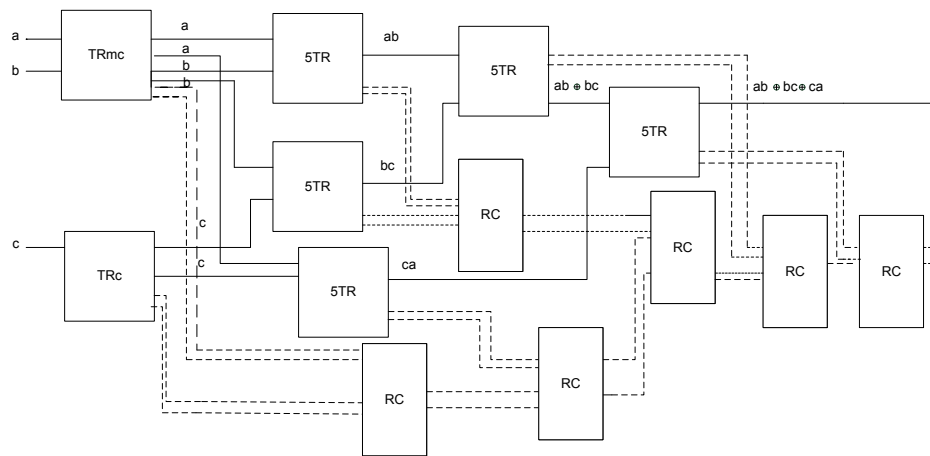


Figure A.6: Realization of the benchmark $3CyG2$ using $5TR$, TR_c and TR_{mc} blocks.

A.2.6 Non-testable ternary gates

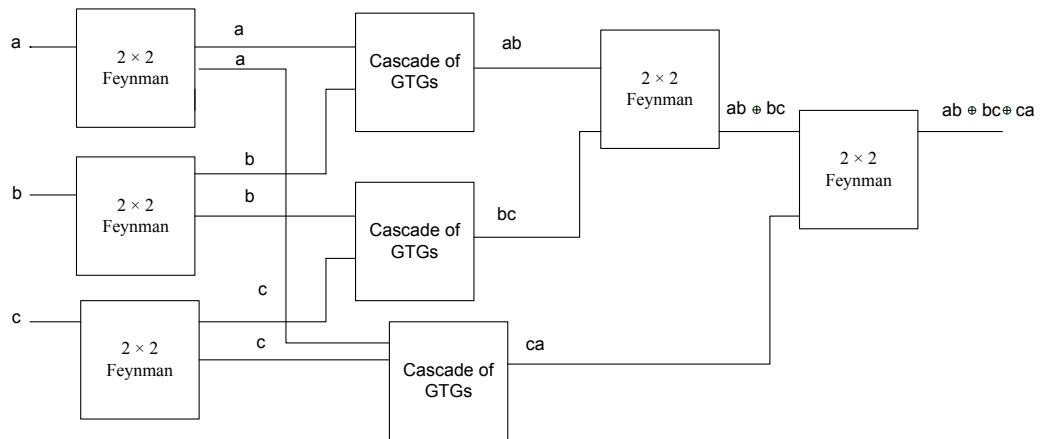


Figure A.7: Realization of the benchmark $3CyG2$ using non-testable ternary gates.

A.3 ProdG3

The following sections illustrate the realization of the benchmark $ProdG3$ using the five proposed methods and non-testable ternary gates.

A.3.1 Method 0

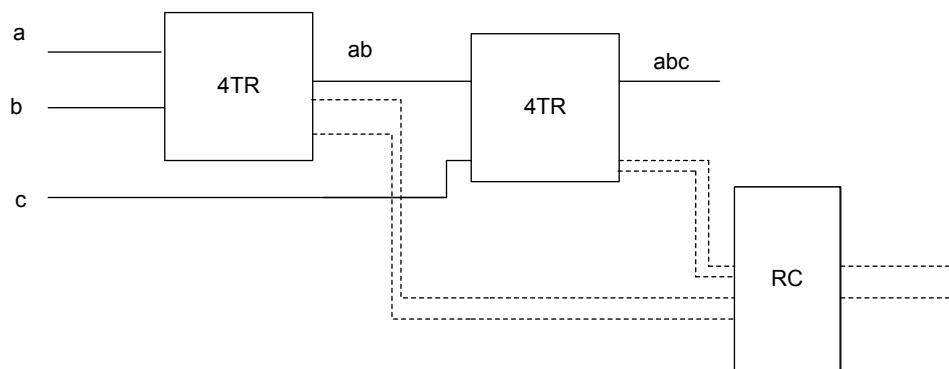


Figure A.8: Realization of the benchmark $ProdG3$ using $4TR$ blocks.

A.3.2 Method 1

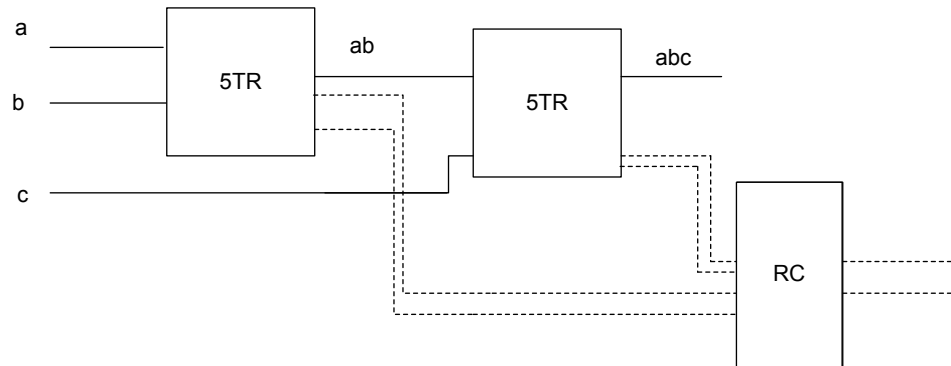


Figure A.9: Realization of the benchmark *ProdG3* using *5TR* blocks.

A.3.3 Method 2

The realization of *ProdG3* using Method 2 is identical to that using *4TR* blocks since no copy function is required.

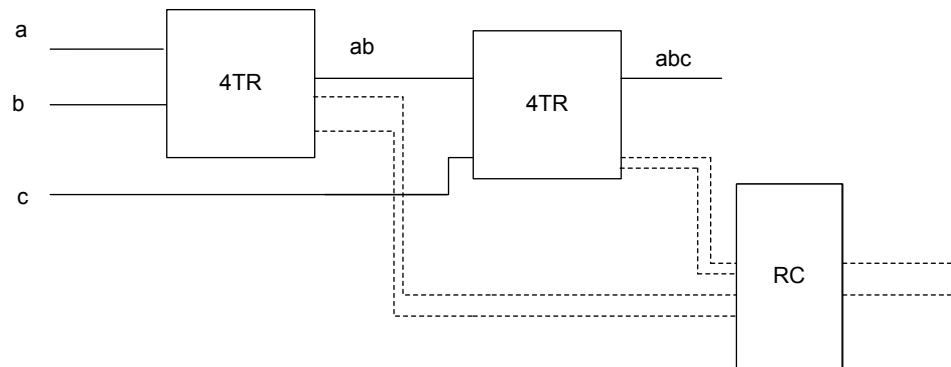


Figure A.10: Realization of the benchmark *ProdG3* using *4TR* blocks and Feynman gates.

A.3.4 Method 3

The realization of *ProdG3* using Method 3 is identical to that using *5TR* blocks since no copy function is required.

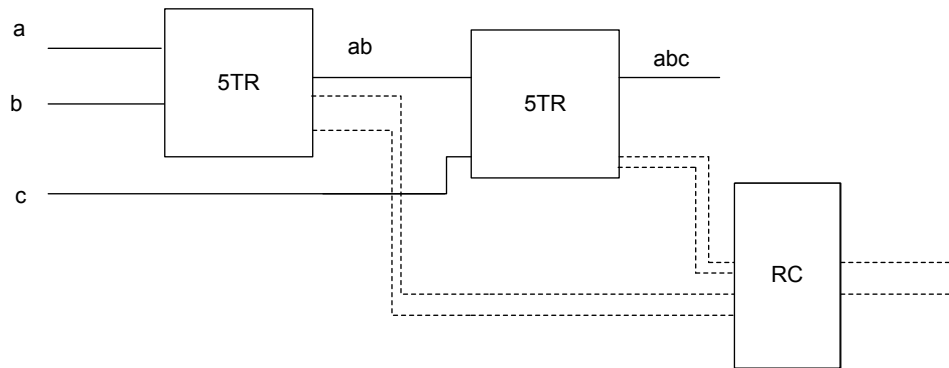


Figure A.11: Realization of the benchmark *ProdG3* using *5TR* and TR_c blocks.

A.3.5 Method 4

The realization of *ProdG3* using Method 4 is identical to that using *5TR* blocks since no copy function is required.

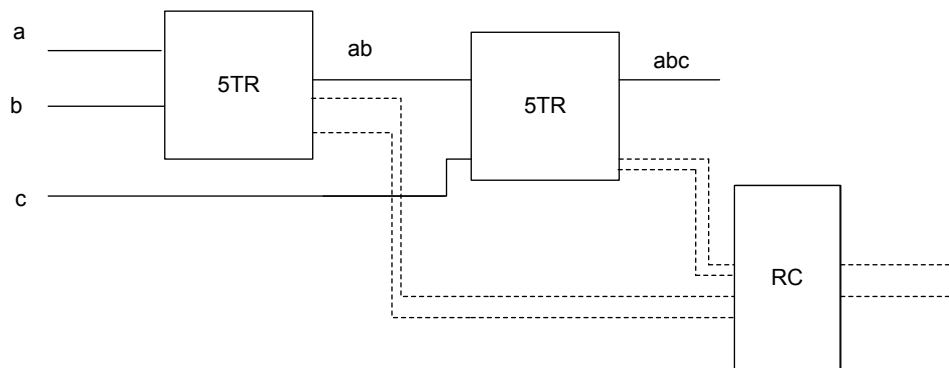


Figure A.12: Realization of the benchmark *ProdG3* using *5TR*, TR_c and TR_{mc} blocks.

A.3.6 Non-testable ternary gates

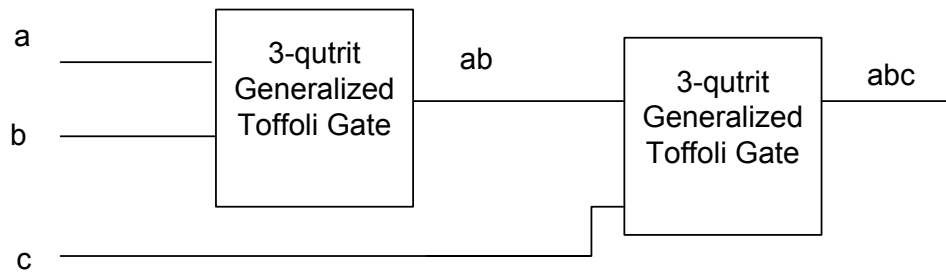


Figure A.13: Realization of the benchmark *ProdG3* using non-testable ternary gates.

A.4 SumG3

The following sections illustrate the realization of the benchmark *SumG3* using the five proposed methods and non-testable ternary gates.

A.4.1 Method 0

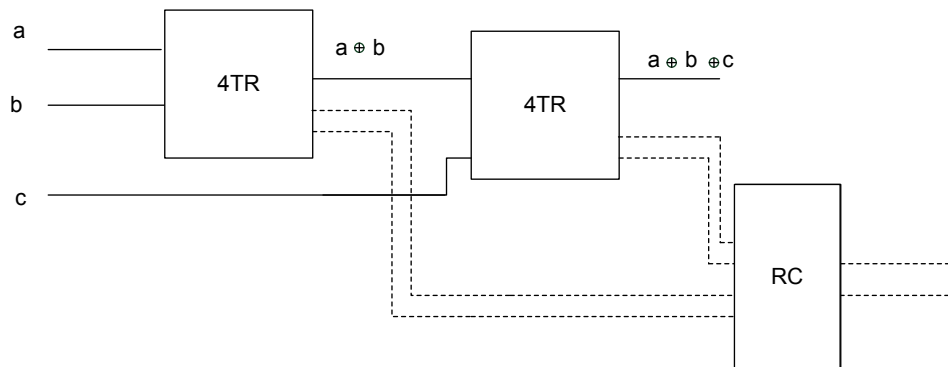


Figure A.14: Realization of the benchmark *SumG3* using *4TR* blocks.

A.4.2 Method 1

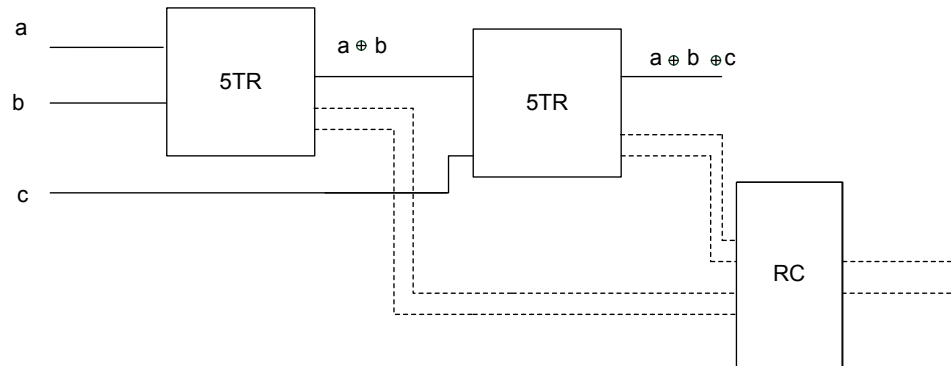


Figure A.15: Realization of the benchmark *SumG3* using 5TR blocks.

A.4.3 Method 2

The realization of *SumG3* using Method 2 is identical to that using 4TR blocks since no copy function is required.

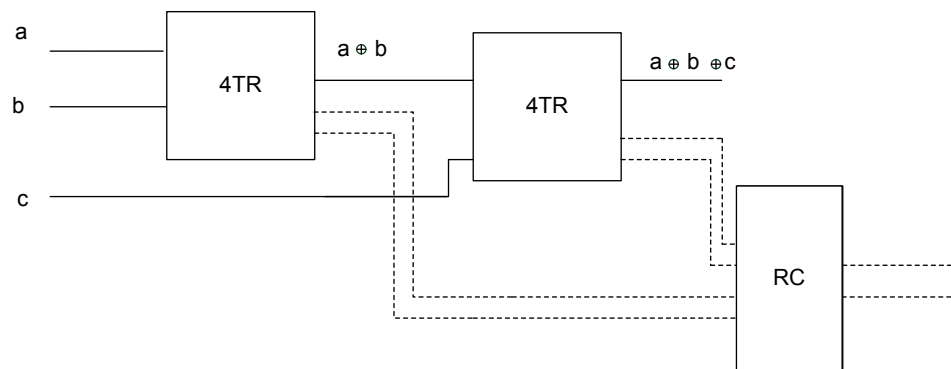


Figure A.16: Realization of the benchmark *SumG3* using 4TR blocks and Feynman gates.

A.4.4 Method 3

The realization of *SumG3* using Method 3 is identical to that using *5TR* blocks since no copy function is required.

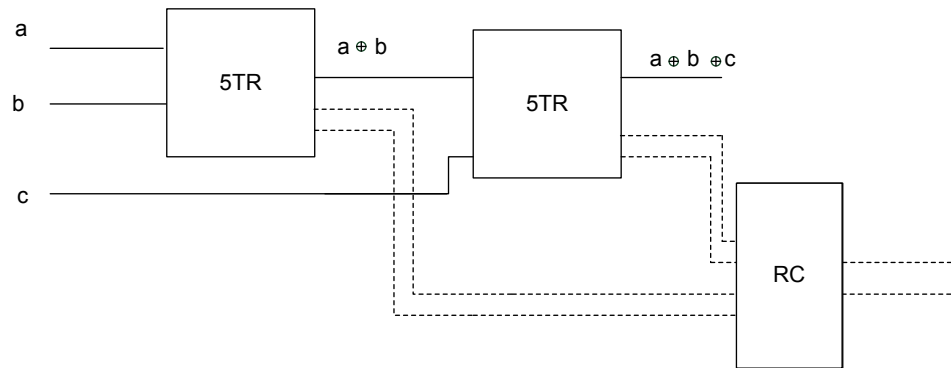


Figure A.17: Realization of the benchmark *SumG3* using *5TR* and TR_c blocks.

A.4.5 Method 4

The realization of *SumG3* using Method 4 is identical to that using *4TR* blocks since no copy function is required.

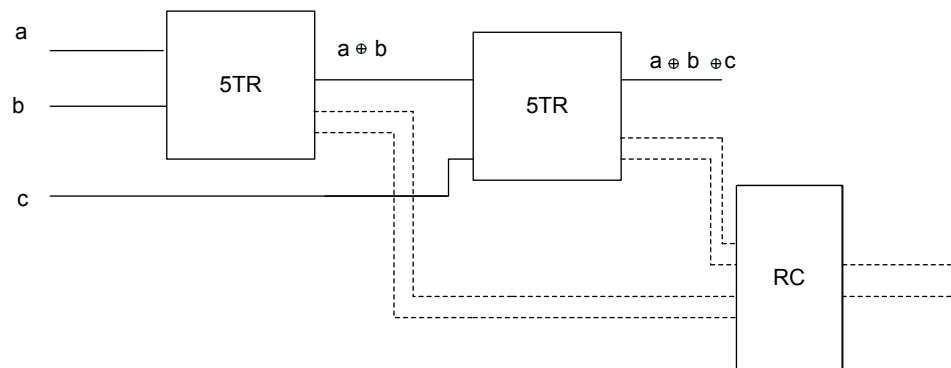


Figure A.18: Realization of the benchmark *SumG3* using *5TR*, TR_c and TR_{mc} blocks.

A.4.6 Non-testable ternary gates

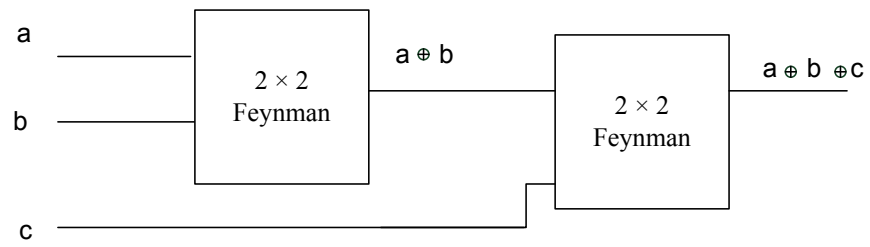


Figure A.19: Realization of the benchmark *SumG3* using non-testable ternary gates.

Bibliography

- [1] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.
- [2] N. Denler, B. Yen, M. Perkowski, and P. Kerntopf. Synthesis of reversible circuits from a subset of Muthukrishnan-Stroud quantum realizable multi-valued gates. In *Proceedings of IWLS 2004, Tamecula, California, USA*, June 2004.
- [3] G. Epstein. A summary of investigation into three and four valued logic. In *Proceedings of the 8th International Symposium on Multi Valued Logic, Rosemont, Illinois, United States*, page 257, 1978.
- [4] N. Farazmand, M. Zamani, and M. B. Tahoori. Online fault testing of reversible logic using dual rail coding. In *Proceedings of the IEEE 16th International On-Line Testing Symposium (IOLTS)*, pages 204 – 205, 2010. Corfu, 5-7 July.
- [5] R. Feynman. Quantum mechanical computers. *Optical News*, pages 11–20, 1985.
- [6] M. P. Frank. Introduction to reversible computing: Motivation, progress, and challenges. In *Proceedings of the 2nd conference on computing frontiers, Ischia, Italy*, May 4-6, 2005.
- [7] E. Fredkin and T. Toffoli. Conservative logic. *Int. Journal of Theoretical Physics*, 21:219–253, 1982.
- [8] N. Jha and S. Gupta. *Testing of Digital Systems*. The Press Syndicate of the University of Cambridge, 2003.
- [9] B. W. Johnson. *Design and Analysis of Fault-tolerant Digital Systems*. Prentice-Hall International, 1985.
- [10]] M. H. A. Khan, M. A. Perkowski, and P. Kerntopf. Multi-output galois field sum of products synthesis with new quantum cascades. In *Proceedings of the 33rd International Symposium on Multiple-Valued Logic*, pages 146–153, 2003. Tokyo, May 16-19.
- [11] M. H. A. Khan. Design of reversible/quantum ternary comparator circuits. *Engineering Letters*, 16:2:178–184, May 2008.
- [12] M. H. A. Khan. Quantum realization of ternary Toffoli gate. In *Proceedings of the 3rd International Conference on Electrical and Computer Engineering*, pages 264–266, 28-30 December, 2004. Dhaka, Bangladesh.
- [13] M. H. A. Khan and M. A. Perkowski. Genetic algorithm based synthesis of multi-output ternary functions using quantum cascade of generalized ternary gates. volume 2, pages 2194 – 2201, 19-23, June 2004.

- [14] M. H. A. Khan and M. A. Perkowski. Quantum ternary parallel adder/subtractor with partially-look-ahead carry. *Journal of Systems Architecture*, 53:453–464, 2007.
- [15] M. H. A. Khan, M. A. Perkowski, and M. R. Khan. Ternary Galois field expansions for reversible logic and Kronecker decision diagram for ternary GFSOP minimization. In *Proceedings of the 34th International Symposium on Multiple-Valued Logic*, pages 58–67, 2004. Toronto, Canada, 19-22 May.
- [16] D. K. Kole, H. Rahman, D. K. Das, and B. B. Bhattacharya. Synthesis of online testable reversible circuit. In *Proceedings of the IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 277 – 280, 2010. Vienna, 14–16 April.
- [17] P. K. Lala. *Fault-Tolerant and Fault Testable Hardware Design*. Prentice-Hall International, 2003. London, UK.
- [18] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.
- [19] S. N. Mahammad and K. Veezhinathan. Constructing online testable circuits using reversible logic. *IEEE Transaction on Instrumentation and Measurement*, 59(1):101–109, January 2010.
- [20] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, page 23(11):1497 1509, 2004.
- [21] D. Maslov and G. W. Dueck. Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 162–170, March, 2003.
- [22] D. M. Miller, D. Maslov, and G. W. Dueck. Synthesis of quantum multiple-valued circuits. In *Journal of Multiple-Valued Logic and Soft Computing*, volume 12, pages 431 – 450, 2006.
- [23] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations*. ISSN 1932-3174. The Morgan and Claypool Publishers, 2008.
- [24] A. Muthukrishnan and C. R. Stroud Jr. Multivalued logic gates for quantum computation. *Phys. Rev. A*, 62(5):1–8, 2000.
- [25] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar. Using implications for online error detection. In *Proceedings of the International Test Conference*, pages 1–10, 2008. Santa Clara, CA.
- [26] D. Nikolos. Self-testing embedded two-rail checkers. *Journal of Electronic Testing: Theory and Applications*, 12:6979.

- [27] N.M.Nayeem. Synthesis and testing of reversible toffoli circuits. Master's thesis, University of Lethbridge, 2011.
- [28] S. Palnitkar. *Verilog HDL:a guide to digital design and synthesis*. Prentice-Hall International, 2003.
- [29] B. Parhami. Fault-tolerant reversible circuits. In *In Proceedings 40th Asilomar Conf. Signals, Systems and Computers, Pacific Grove, CA*, pages 1726–1729, October, 2006.
- [30] M. A. Perkowski, A. Al-Rabadi, and P. Kerntopf. Multiple-valued quantum logic synthesis. In *Proceedings of the Int. Symposium on New Paradigm VLSI Computing*, pages 41–47, 2002. Sendai, Japan, 12-14 December.
- [31] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes. A family of logical fault models for reversible circuits. In *Proceedings of the 14th Asian Test Symposium*, pages 422–427, Los Alamitos, CA, USA, 2005. IEEE Computer Society.
- [32] M. R. Rahman and J. E. Rice. On designing a ternary reversible circuit for online testability. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 119–124, 2011. August 23–26, Victoria, Canada.
- [33] M. R. Rahman and J. E. Rice. Online testable ternary reversible circuit. In *Proceedings of the Reed-Muller Workshop*, pages 71–79, 2011. May 25–26, Tuusula, Finland.
- [34] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE transactions on computer aided design of integrated circuits and systems*, 22:6, June,2003.
- [35] T. Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming, Springer-Verlag London, UK*, pages 632–644, 1980.
- [36] D. P. Vasudevan, P. K. Lala, J. Di, and J. P. Perkerson. Reversible logic design with on-line testability. *IEEE Transaction on Instrumentation and Measurement*, 55(2):406–414, April 2006.
- [37] V. V. Zhirnov, R. K. Kavin, J. A. Hutchby, and G. I. Bourianoff. Limits to binary logic switch scaling - a gedanken model. In *Proceedings of the IEEE*, volume 91:11, pages 1934–1939, November, 2003.
- [38] J. Zhong and J. C. Muzio. Analyzing fault models for reversible logic circuits. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 2422 – 2427, 2006. Vancouver, BC.