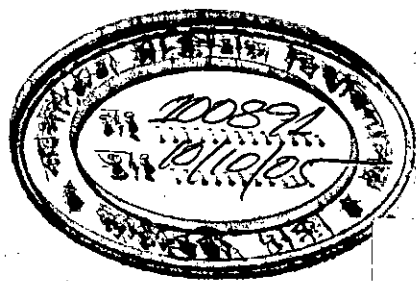


EVOLUTIONARY ALGORITHM BASED SYNTHESIS OF
MULTI-OUTPUT TERNARY FUNCTIONS USING
QUANTUM CASCADES

Md. Mujibur Rahman Khan

A Thesis Submitted to the Department of Computer Science and Engineering in the
Partial Fulfillment of the Requirements for the
Degree of
Master of Science in Engineering
(Computer Science and Engineering)

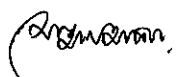


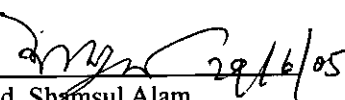
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA, BANGLADESH


JUNE 2005

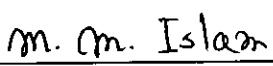
The thesis "Evolutionary Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascades", submitted by Md. Mujibur Rahman Khan, Roll No. 040305011P, Registration No. 0403224, Session April 2003, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science and Engineering (Computer Science and Engineering) and approved as to its style and contents. Examination held on June 29, 2005.

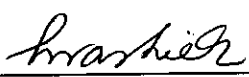
Board of Examiners

1. 

Dr. Md. Mostofa Akbar
Assistant Professor
Department of CSE
BUET, Dhaka-1000
Chairman
(Supervisor)
2. 

Dr. Md. Shamsul Alam
Professor and Head
Department of CSE
BUET, Dhaka-1000
Member
(Ex-officio)
3. 

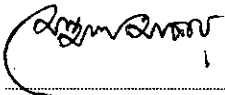
Dr. Md. Abul Kashem Mia
Professor
Department of CSE
BUET, Dhaka-1000
Member
4. 

Dr. Md. Monirul Islam
Associate Professor
Department of CSE
BUET, Dhaka-1000
Member
5. 

Dr. A. B. M. Harun Ur-Rashid
Associate Professor
Department of EEE
BUET, Dhaka-1000
Member
(External)

Declaration

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Md. Mostofa Akbar, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or Diploma.



(Dr. Md. Mostofa Akbar)

Supervisor



(Md. Mujibur Rahman Khan)

Acknowledgement

Here I would like to take the opportunity to express my greatest gratitude to the patrons of this thesis work, without whom I could never have completed this arduous task.

For me, it has been a big journey from the start to end. Needless to say, that the only thing that kept me going was the support of a number of people. First and foremost, my thesis supervisor, **Dr. Md. Mostofa Akbar**, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, “Without your unstinting support, faith in my work, there is just no way that I could have completed my thesis. You have been always there whenever I needed your help in any form. I guess no words can adequately describe what you have done for me and for my work as an advisor, and companion. I thank you for everything.”

I would also like to express my heartiest gratitude to **Dr. Mozammel H. A. Khan**, Professor and Dean, Faculty of Engineering, East West University, Dhaka for his fruitful suggestions whenever I needed. Without him, the field of Quantum Computation and Reversible Logic would remain unknown to me. He has been guiding me in doing research for more than seven years. Without his affectionate mentoring, it would not be possible for me to complete this thesis.

I would also like to thank **Dr. Md. Monirul Islam**, Associate Professor, Department of CSE, BUET, whom I visited time and again for both academic and administrative help, guidance, and advice. His constant caring supports and patience encouraged me throughout my studentship in BUET.

I must thank **Dr. Md. Shamsul Alam**, Professor and Head, Department of CSE BUET.

I would like to thank all the faculty members of this department as they have helped me directly or indirectly in various ways to complete this thesis.

I would like to thank all my course mates in BUET. Especially **Rafiq, Abid, Bipul, Monower**, and others, who took part in the group with me to discuss the lessons while completing the course works.

I would also like to thank all the staffs of Department of CSE, BUET. Especially **Mr. Syed Ehsan**, Senior Lab incharge, Dept. of CSE, BUET, who arranged me a machine on a secured corner of the Lab and ensured that nobody is using the machine or turning the switch off during my absence.

And last but not the least, I must acknowledge with due respect the constant support and patience of my family members for completing the thesis. Particularly, my wife **Syeda Shabrena Sultana** and my father **Khan Fazlay Ahmed**. I would also like to remember the inspiration of my late mother **Begum Jahanara**.

CONTENTS

DECLARATION	III
ACKNOWLEDGEMENT	IV
CONTENTS	VI
LIST OF FIGURES	IX
LIST OF TABLES.....	XI
ABSTRACT	1
CHAPTER 1	2
INTRODUCTION	2
1.1 MOTIVATION	3
1.2 BACKGROUND AND PRESENT STATE OF THE PROBLEM	5
1.3 OBJECTIVES AND FOCUS OF THE THESIS	7
1.4 ORGANIZATION OF THE THESIS	8
CHAPTER 2	9
QUANTUM COMPUTER: FUNDAMENTAL CONCEPTS	9
2.1 INTRODUCTION.....	9
2.2 QUANTUM MECHANICS	10
2.3 QUANTUM BITS.....	10
2.3.1 SINGLE QUBIT	11
2.3.2 MULTIPLE QUBITS.....	13
2.4 QUANTUM COMPUTATION	14
2.4.1 SINGLE QUBIT GATES.....	15
2.4.2 MULTIPLE QUBIT GATES	17
2.4.3 SINGLE QUBIT AND CNOT GATES ARE UNIVERSAL.....	18
2.4.4 QUANTUM CIRCUITS	21
2.4.5 CAN A QUBIT BE COPIED?.....	22
2.4.6 QUANTUM PARALLELISM.....	23
2.5 QUANTUM INFORMATION.....	24
2.6 PROSPECTS FOR QUANTUM INFORMATION PROCESSING	25
CHAPTER 3	27
QUANTUM COMPUTER: PHYSICAL REALIZATION	27
3.1 REALIZATION OF QUANTUM COMPUTER.....	27
3.2 CONDITIONS FOR QUANTUM COMPUTATION	28
3.2.1 REPRESENTATION OF QUANTUM INFORMATION	28
3.2.2 PERFORMING UNITARY TRANSFORMATION	29

3.2.3	PREPARATION OF FIDUCIAL INITIAL STATES -----	30
3.2.4	MEASUREMENT OF OUTPUT RESULTS-----	31
3.3	HARMONIC OSCILLATOR QUANTUM COMPUTER -----	32
3.3.1	THE QUANTUM HARMONIC OSCILLATOR (QHO)-----	32
3.3.2	PHYSICAL APPARATUS FOR QHO-----	34
3.3.3	THE HAMILTONIAN FOR QHO-----	35
3.3.4	QUANTUM COMPUTATION FOR QHO-----	36
3.3.5	DRAWBACKS OF QHO -----	37
3.3.6	SUMMARY OF QHO PROPERTIES -----	38
3.4	OPTICAL PHOTON QUANTUM COMPUTER (OPQC)-----	38
3.4.1	PHYSICAL APPARATUS OF AN OPQC-----	39
3.4.2	QUANTUM COMPUTATION WITH OPQC -----	41
3.4.3	DRAWBACKS OF OPQC-----	44
3.4.4	SUMMARY OF OPQC PROPERTIES-----	44
3.5	OPTICAL CAVITY QUANTUM ELECTRODYNAMICS (OCQED)-----	45
3.5.1	PHYSICAL APPARATUS FOR OCQED-----	46
3.5.2	SUMMARY OF OCQED PROPERTIES-----	47
3.6	ION TRAPS -----	48
3.6.1	PHYSICAL APPARATUS FOR ION TRAPS -----	48
3.6.2	SUMMARY OF ION TRAP PROPERTIES -----	51
3.7	NUCLEAR MAGNETIC RESONANCE (NMR)-----	52
3.7.1	SUMMARY OF NMR PROPERTIES -----	52
3.8	CHAPTER SUMMARY -----	53
CHAPTER 4 -----		55
MULTI-OUTPUT TERNARY LOGIC AND QUANTUM CASCADE: A LITERATURE		
SURVEY-----		55
4.1	REVERSIBLE LOGIC-----	55
4.1.1	MOORE'S LAW-----	55
4.1.2	ARGUMENT FOR ALTERNATIVE TECHNOLOGY-----	56
4.1.3	BINARY REVERSIBLE LOGIC-----	57
4.1.4	TERNARY REVERSIBLE LOGIC -----	62
4.1.5	SOME TERNARY REVERSIBLE GATES -----	62
4.2	GALOIS FIELD AND QUANTUM TECHNOLOGY-----	64
4.2.1	QUANTUM COMPUTATION-----	65
4.2.2	TERNARY QUANTUM COMPUTING-----	65
4.2.3	QUANTUM CIRCUIT-----	67
4.2.4	GALOIS FIELD-----	67
4.2.4.1	GF(2)-----	68
4.2.4.2	GF(3)-----	68

4.2.4.3	GF(4)	68
4.2.5	TERNARY GALOIS FIELD LOGIC	68
4.2.6	QUANTUM CASCADE (QC)	70
4.2.7	REALIZATION OF MVL USING QUANTUM CASCADE	71
4.2.8	SOME EXISTING METHODS OF REALIZING MVL USING QC	71
4.3	EVOLUTIONARY ALGORITHM	73
4.3.1	GENETIC ALGORITHMS	75
4.3.2	EVOLUTION STRATEGIES	75
4.3.3	EVOLUTIONARY PROGRAMMING	76
4.4	SUMMARY	77
CHAPTER 5		79
EA BASED SYNTHESIS OF MULTI-OUTPUT TERNARY FUNCTION USING QUANTUM CASCADES		79
5.1	INTRODUCTION	79
5.2	THE NEW 2*2 QUANTUM TERNARY GATES	79
5.3	REALIZATION OF MULTI-OUTPUT TERNARY FUNCTIONS USING THE NEW GATES	80
5.4	GTG VERSES THE NEW GATES	81
5.5	PROPOSED EVOLUTIONARY ALGORITHM	84
5.5.1	PROBLEM ENCODING	85
5.5.2	FITNESS COMPONENTS	86
5.5.3	DESCRIPTION OF THE EVOLUTIONARY ALGORITHM	91
CHAPTER 6		97
EXPERIMENTAL RESULTS AND DISCUSSION		97
6.1	INTRODUCTION	97
6.2	EXPERIMENTAL SETUP AND FINDINGS	97
6.3	CONCLUSIONS	113
CHAPTER 7		114
CONCLUSION		114
7.1	CONCLUDING WORDS	114
7.2	RECOMMENDATIONS FOR FUTURE WORK	116
APPENDIX A		118
	SOURCE CODE OF THE PROGRAM	118
APPENDIX B		144
	DESCRIPTION OF THE BENCHMARK FUNCTIONS	144
BIBLIOGRAPHY		145

List of Figures

FIG 1.1: GENERAL FORM OF EXTENDED DE VOS GATE.....	6
FIG 2.1: QUBIT REPRESENTED BY TWO ELECTRONIC LEVELS IN ATOM.....	11
FIG 2.2: BLOCH SPHERE REPRESENTATION OF A QUBIT.	12
FIG 2.3: VISUALIZATION OF HADAMARD GATE ON THE BLOCH SPHERE, ACTING ON THE INPUT STATE $(0\rangle + 1\rangle)/\sqrt{2}$	16
FIG 2.4: SINGLE BIT (LEFT) AND SINGLE QUBIT (RIGHT) LOGIC GATES.....	17
FIG 2.5: GRAPHIC AND MATRIX REPRESENTATION OF <i>CNOT</i> GATE.....	17
FIG 2.6: CONTROLLED- <i>U</i> GATE.....	18
FIG 2.7: CIRCUIT IMPLEMENTING THE TWO-LEVEL UNITARY OPERATION DEFINED BY U_x	21
FIG 2.8: CLASSICAL AND QUANTUM CIRCUIT TO COPY AN UNKNOWN BIT OR QUBIT.	22
FIG 2.9: QUANTUM CIRCUIT FOR EVALUATING $f(0)$ and $f(1)$ SIMULTANEOUSLY. U_f IS THE QUANTUM CIRCUIT WHICH TAKES INPUTS LIKE $ x, y\rangle$ TO $ x, y \oplus f(x)\rangle$	24
FIG 3.1: SKETCH OF THE FIRST FIVE SOLUTIONS OF THE SCHRÖDINGER EQUATION FOR $\psi_n(x)$	33
FIG 3.2: PARAMETRIC DOWN-CONVERSION FOR GENERATION OF SINGLE PHOTONS.....	40
FIG 3.3: SCHEMATIC OF AN OPTICAL BEAMSPLITTER. (B) IS THE INVERSE OF (A).....	41
FIG 3.4: OPTICAL CIRCUIT REPRESENTING A PHASE SHIFT BY π	43
FIG 3.5: SCHEMATIC DRAWING OF AN ION TRAP QUANTUM COMPUTER.	49
FIG 4.1: A REVERSIBLE GATE	57
FIG 4.2: MATRIX AND GRAPHIC REPRESENTATION OF NOT GATE.....	60
FIG 4.3: MATRIX AND GRAPHIC REPRESENTATION OF CONTROLLED-NOT GATE.....	61
FIG 4.4: MATRIX AND GRAPHIC REPRESENTATION OF CC-NOT GATE	62
FIG 4.5: SOME TERNARY REVERSIBLE GATES.....	63
FIG 4.6: TERNARY SHIFT OPERATIONS, GATE SYMBOLS, AND THEIR NUMBERS	63
FIG 4.7: QUANTUM CIRCUIT USING TOFFOLI GATES TO REALIZE THE FUNCTION, $F(A,B,C)$ $= [0, 1, 2, 1, 0, 2, 2, 2, 2, 1, 0, 0, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 0, 0]^T$	64
FIG 4.8: GENERALIZED TERNARY GATES (GTG).....	65
FIG 4.9: QUANTUM REALIZATION OF TERNARY SHIFT GATES	69
FIG 4.10: QUANTUM UNITARY TRANSFORMATION	70
FIG 4.11: QUANTUM CASCADE REALIZING AN ARBITRARY 3-INPUT 2-OUTPUT TERNARY FUNCTION.	72
FIG 4.12: REALIZATION OF TERNARY SWAP GATE USING GTG GATES	72
FIG 4.13: GENERALIZED MULTI-VALUED GATE OF RADIX $M+1$	73
FIG 4.14: A CANONICAL GA.....	75
FIG 4.15: A SIMPLE ES.....	76
FIG 5.1: GENERAL FORM OF THE PROPOSED GATES	79
FIG 5.2: REALIZATION OF TERNARY HALF ADDER USING THE NEW GATES	81
FIG 5.3: TERNARY HALF ADDER REALIZATION USING GTG GATES BY [36].....	82

FIG 5.4: REALIZATION OF $\langle A, B, 0, y \rangle$ AND $\langle A, B, 1, y \rangle$ GATES USING DE VOS GATES.	83
FIG 5.5: ENCODING OF THE TERNARY HALF ADDER CIRCUIT	85
FIG 5.6: CHROMOSOME REPRESENTING THE CIRCUIT IN FIGURE 5.5	86
FIG 5.7: SUB-VECTORS OF AN ARBITRARY 3-INPUT 2-OUTPUT TERNARY FUNCTION.	87
FIG 5.8: REALIZATION OF SUB-VECTORS	88
FIG 5.9: FLOWCHART OF THE PROPOSED EVOLUTIONARY ALGORITHM	91
FIG 5.10: RANKING OF THE INDIVIDUALS IN AN ARBITRARY POPULATION	93
FIG 5.11: CROSSOVER OPERATION.....	94
FIG 5.12: MUTATION OPERATION.....	95
FIG 5.11: REDUNDANT COLUMNS AND UNUSED CONSTANT LINES IN AN ARBITRARY QUANTUM CASCADE.....	95
FIG 5.14: INSERTION OF THE OFFSPRING INTO THE POPULATION WITH RANKING	96
FIG 6.1: EFFECT OF P_C AND P_M ON COST OF SOLUTION.....	98
FIG 6.2: EFFECT OF P_C ON LENGTH OF THE CIRCUIT.....	99
FIG 6.3: EFFECT OF P_C ON WIDTH OF THE CIRCUIT.....	99
FIG 6.4: EFFECT OF P_M ON LENGTH OF THE CIRCUIT	100
FIG 6.5: EFFECT OF P_M ON WIDTH OF THE CIRCUIT	100
FIG 6.6: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR ABC2.	101
FIG 6.7: CONVERGENCE OF LENGTH OF CASCADE FOR ABC2.	102
FIG 6.8: CONVERGENCE OF SCRATCHPAD WIDTH FOR ABC2.....	102
FIG 6.9: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR MUL2.....	103
FIG 6.10: CONVERGENCE OF LENGTH OF CASCADE FOR MUL2.....	104
FIG 6.11: CONVERGENCE OF SCRATCHPAD WIDTH FOR MUL2.....	104
FIG 6.12: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR A2BCC.	105
FIG 6.13: CONVERGENCE OF LENGTH OF CASCADE FOR A2BCC.	106
FIG 6.14: CONVERGENCE OF SCRATCHPAD WIDTH FOR A2BCC.....	106
FIG 6.15: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR THADD.....	107
FIG 6.16: CONVERGENCE OF LENGTH OF CASCADE FOR THADD.....	107
FIG 6.17: CONVERGENCE OF SCRATCHPAD WIDTH FOR THADD.	108
FIG 6.18: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR PROD3.....	109
FIG 6.19: CONVERGENCE OF LENGTH OF CASCADE FOR PROD3.	109
FIG 6.20. CONVERGENCE OF SCRATCHPAD WIDTH FOR PROD3.	110
FIG 6.21: CONVERGENCE OF OUTPUT VECTOR FITNESS FOR AVG2.....	110
FIG 6.22: CONVERGENCE OF LENGTH OF CASCADE FOR AVG2.	111
FIG 6.23: CONVERGENCE OF SCRATCHPAD WIDTH FOR AVG2.....	111
FIG 6.24: REALIZATION OF MUL2 USING THE PROPOSED METHOD.	112
FIG 6.25: REALIZATION OF PROD2 USING THE PROPOSED METHOD.....	112
FIG 6.26: REALIZATION OF AVG2 USING THE PROPOSED METHOD.	113

List of Tables

TABLE 4.1: TRUTH TABLE OF SOME COMMON 2-INPUT 1-OUTPUT IRREVERSIBLE GATES.	56
TABLE 4.2: THREE INPUT – THREE OUTPUT DEVICES WHICH MAPS EIGHT POSSIBLE STATES ONTO ONLY FOUR DIFFERENT STATES.	58
TABLE 4.3: TRUTH TABLE FOR NOT GATE	60
TABLE 4.4: ADDITION AND MULTIPLICATION IN $GF(2)$	68
TABLE 4.5: ADDITION AND MULTIPLICATION IN $GF(3)$	68
TABLE 4.6: ADDITION AND MULTIPLICATION IN $GF(4)$	68
TABLE 5.1: TRUTH TABLE OF TERNARY HALF ADDER FUNCTION	81
TABLE 6.1: RESULTS OBTAINED FOR DIFFERENT BENCHMARK TERNARY FUNCTIONS.....	101

Abstract

Quantum Computers, which run according to the laws of quantum mechanics, are said to be the future of today's computers. They might have exponentially more computational efficiency than any classical one. The fact that a quantum particle can be in between many states, known as entanglement of states, made Quantum Computer so powerful. Inspired by the challenge of formulating Quantum Computer, this thesis presents the synthesis of multi-output ternary quantum logic with primitive quantum gates. The main emphasis of thesis is given on showing that any logic can be realized using quantum primitive gates. It is also implied that these quantum circuits are reversible by nature. At the same time multiple-valued logic helps to reduce the complexity of the circuit when compared to binary logic.

This thesis presents a comprehensive study on the fundamentals of Quantum Computations. Then a family of quantum primitive gates is proposed. These are very simple 2-input, 2-output ternary reversible gates. These gates can be physically realized using quantum technology. Then an Evolutionary Algorithm based synthesis procedure using those primitive gates is proposed. It has been shown that a Quantum Computer capable of executing any logic function is possible to construct using the new gates only. The claim is supported by the experimental findings. The effect of different EA parameters on the solution is also examined and shown. Finally some open problems for the physicists and mathematicians are brought forward.

Chapter 1



Introduction

The synthesis of multi-valued Quantum logic is an interesting and challenging problem in Computer Science. Ternary quantum circuits have recently been introduced to reduce the size of multi-valued logic for multi-level quantum computing systems. It is implied that the quantum circuits will naturally be reversible. However, synthesizing these quantum circuits is not easy. The following are the issues related to multi-valued quantum logic synthesis:

- How the quantum cascades will be constructed to realize multi-valued, multi-output logic functions.
- How the cost of the circuit will be minimized.
- What types of gates will be used.
- How the fundamental issues regarding reversible logic will be addressed.

This thesis addresses these above mentioned issues. The primary objective of this thesis is to develop a soft computing method to synthesis multi-valued multi-output logic function. With this view, a complete synthesis process of multi-valued multi-output reversible logic using quantum cascades is presented. To be more specific, we have considered ternary reversible logic. We are also proposing a family of elementary quantum reversible gates to construct the quantum cascade. These gates are new and no such previous method exists that realizes multi-valued, multi-output using the new quantum gates. Therefore, it is not possible to directly compare the cost of the circuit. Instead, comparison with the circuit obtained using few other ternary quantum gates are done.

1.1 Motivation

A quantum mechanical phenomenon that never occurs in classical physics and which actually makes quantum computation interesting and powerful, is the superposition of states (“entanglement” of states), which means that instead of being totally in one single state, a particle can be “in between many states”. Compare this to the elements of classical computer, bits. A bit takes values zero or one, but never anything between. In a quantum computer, one is able to store both values simultaneously in one quantum bit, equally weighted or not. Moreover, in the classical computer one can store a number from zero to $2^n - 1$ in a register of length n , whereas, in quantum computation, all the values could be stored simultaneously. The computation, which is performed via **unitary transformations** in the state vector space, then applies to all these values simultaneously. Thus the power of the quantum computation lies not in the absolute speed of the hypothetical quantum computer but in the possibility to actually follow many computational paths simultaneously, as a nondeterministic automation does. Furthermore, the motivation behind this thesis can be better explained by answering the following questions:

Why Quantum Logic?

– As the semiconductor-based circuits are not capable of handling more than two states, scientists are looking for alternative technologies to realize logic functions. Quantum Logic is one of the most prominent among the alternative technologies. In fact, there are really an infinite number of states possible to a quantum bit, not just two. The “entanglement” of states, which means one unit can be at more than one logical state simultaneously as a form of superposition of states. This unique property of quantum technology made it the best choice. [Chapter 2, Chapter 3, and Section 4.2.3]

Why reversible logic circuit?

– It is implied that the quantum computers will be, by nature, reversible. The conventional irreversible logic is, now a day, rejected by the

researchers because they waste a significant amount of power as dissipated heat. Reversible circuits consume less power than conventional irreversible circuits by reducing the wastage of energy as dissipated heat. [Section 4.1]

Why multi-valued logic?

– Because the size of the logic circuit is directly dependent on the amount of information (i.e. the logical states) stored in a single unit. Binary logic can handle only two states – 0 and 1. On the other hand, multi-valued logic circuits (if constructed) are capable of storing more than two states, thus reduced in size and complexity. [Section 4.1.4]

Why EA?

– As there is no direct method to construct a quantum cascade using the new gates, we have to go for Evolutionary Algorithms (EA). Use of EA will allow us to find an appropriate combination of the new gates that realize (perhaps optimally) a multiple-valued ternary logic function. EAs are very popular Soft Computing (SC) approach for solving problems with no identified structure and high level of noise. The reasons behind this popularity are–

- A large solution space can be searched.
- The size of this search space can be moderated by parameters.
- A variety of new solutions can be produced, and
- With long enough time a solution can be obtained that is close to the optimal one.

Because of these advantages we have selected EA for synthesizing ternary functions using cascade of the new gates as the problem structure of such cascade is still undefined and the search space itself is exponentially large. [Section 4.3]

Multi-valued quantum logic synthesis methods are still very immature, though a number of works have been done (see [1]-[4], [7], [34]-[38], [40], [42]-[44]).

From these works, however, it is more or less evident that Galois Field Sum of Products (GFSOP) is a good choice for multi-valued reversible logic synthesis. In this thesis, we focus only on ternary GFSOP synthesis with cascades of quantum gates.

1.2 Background and Present State of the Problem

The unit of memory (information) for binary quantum computation is a *qubit*, the simplest quantum system that exists in a linear superposition of two basis states labeled $|0\rangle$ and $|1\rangle$. In 1996, Mattle et al [60] used the term *trit* for a ternary equivalent of qubit (however, *qutrit* is appropriate). In 1997, Chau [24] introduced the concept of a *qudit*, a d -dimensional quantum system that generalizes a qubit and has basis states $|0\rangle, |1\rangle, |2\rangle, \dots, |d-1\rangle$. Subsequently, limited work was done in multi-valued quantum logic. The work of Chau [24], Rains [22] and Ashikhmin and Knill [5], extended quantum error-correcting codes to multi-valued logic for correcting codes in single and multiple qudits. Gottesman [19] and Aharonov and Ben-Or [13] developed fault-tolerant procedures for implementing two-qudit and three-qudit analogs of universal binary gates. Burlakov [9] proposed to use correlated photon pair to represent qutrit. Since 2000 the works have got momentum.

Muthukrishnan and Stroud [7] developed multi-valued logic for multi-level quantum computing systems and showed their realizability in linear ion trap devices. However, this approach produces circuits of too large dimensions. Universality of n -qudit gates was discussed in [28], and [7] but no design algorithms were given. Picton [47] presented an approach called Universal Architecture for multi-valued reversible logic but this approach produces circuits that are far from minimum and have no relation to quantum realization.

Since 2001 Al-Rabadi et al proposed Galois Field approach to quantum logic synthesis (see [1], [2], [3], [4], and [42]). In this work Galois quantum matrices were proposed for swap and Toffoli gates, but without the proof that they can be

built from only $1*1$ and $2*2$ gates¹. Several regular structures for multi-valued quantum logic were also proposed, including cascades, but these cascades do not allow realization of powers of GFSOP and are thus non-universal. This work was based on previous works on GFSOPs and similar forms of Galois and similar logic, in which canonical expansions of Post literals and arbitrary functions were shown. However, no constructive methods for GFSOP and cascade minimization were given, nor programs were written for them. Factorized reversible cascades and complex gates (which usually yield better result) were not proposed. De Vos proposed two ternary $1*1$ gates and two ternary $2*2$ gates [6], but no synthesis method was proposed. New efficient reversible multi-valued gates were proposed in [45] and quantum realizations of multi-valued Toffoli gate in [43]. However, very little has been published on synthesis algorithms for multi-output multi-valued quantum circuits. Therefore, it is very important to look for efficient methods to synthesize *multi-output GFSOP* functions using quantum cascades. In this thesis, we concentrate on quantum cascaded realization of only ternary GFSOP functions.

The major problem of logic synthesis is that any m -input/ m -output gate ($m*m$ gate), where $m > 2$, is very difficult to realize in quantum technology ([6], [7]). Therefore it would be a better idea if the quantum circuit is constructed using $2*2$ gates (primitive gates) only. Most of the researches done so far in this field are using the gates with $m > 2$. Therefore the circuits are complex and almost impossible to realize.

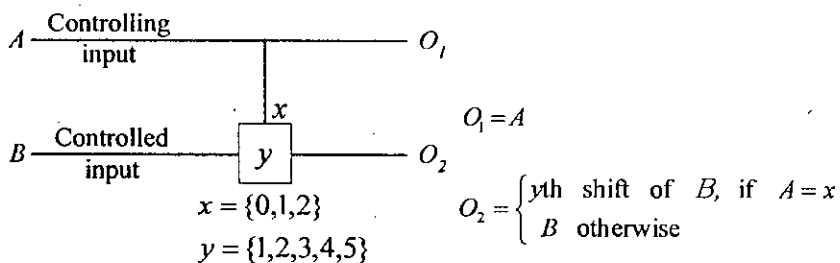


Fig 1.1: General form of Extended De Vos Gate

¹ A $m*n$ gate is one whose number of input is m and the number of output is n .

Khan et. al. proposed one complete synthesis process of ternary logic using quantum cascades (see [37]). They have used ternary Toffoli gates, Feynman gates, and swap gates as building blocks of the quantum cascade. The widths of the gates are also more than two ($m > 2$), hence the problem still remains. Although those gates can be constructed using 2×2 gates, the cost of the circuit becomes very high in terms of number of gates and the complexity of the circuit.

We are proposing a complete synthesis process to realize ternary logic using 2×2 gates directly. We are also proposing a family of extended ternary De Vos [6] gates (see Figure 1.1), those are realizable using quantum technology, to be used as building blocks of the circuit. Our primary goal is to realize any ternary reversible multi-output function using only those gates. In Figure 1.1, A and B are the two inputs. O_1 and O_2 are the two outputs while x and y are two parameters of the Gate. There are 15 possible combinations of x and y .

The main problem is there is no direct method to construct a quantum cascade using De Vos gates. The problem also remains with the new gates. Hence we have to go for Evolutionary Algorithms (EA). Use of EA will allow us to find an appropriate combination of the gates that realize a multiple-output ternary logic function. We might sometimes achieve the optimal solutions also.

1.3 Objectives and Focus of the Thesis

The main focus of this dissertation is to present a complete synthesis process of realizing ternary reversible logic using the new 2×2 primitive gates. Our ultimate goals and objectives could be summarized as follows:

- Develop a complete synthesis process to synthesize ternary multi-output function using cascade of primitive quantum gates proposed in this thesis. And eventually prove that any ternary multi-output function can be realized using the cascade of the new gates.
- Exploring the behavior of the EA parameters in solving this type of problem.

- Making guidelines for further research and future researchers in this field.

To achieve our primary goal, we have implemented the proposed synthesis process using C++ and obtained encouraging results. Implementing the gates using quantum technology and the physical realization of the circuit are beyond the scope of this thesis.

1.4 Organization of the Thesis

The thesis has been organized in different chapters, with each chapter discussing different aspects of the study. The areas covered by different chapters are briefly as follows:

- Chapter 2:** Elementary discussion on quantum mechanics and quantum computation.
- Chapter 3:** Elementary discussion on the realization technologies of quantum logic circuit.
- Chapter 4:** Provides a literature review of the different strategies that have been proposed in the context of Multiple-Valued Logic Synthesis.
- Chapter 5:** Detailed description of the proposed EA based synthesis method.
- Chapter 6:** Experimental results and discussions.
- Chapter 7:** Concluding remarks and suggestion for future research works.

Chapter 2

Quantum Computer: Fundamental Concepts

2.1 Introduction

Moor's Law has approximately held true in the decades since the 1960s. Nevertheless, most observers expect that this dream run will end some time during the first two decades of the twenty first century. Conventional approaches to the fabrication of computer technology are beginning to run up against fundamental difficulties of size. Quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller and smaller. One possible solution to the problem posed by the eventual failure of Moor's Law is to move to a different computing paradigm. One such paradigm is provided by the theory of Quantum Computation. Quantum Computation is based on the idea of using quantum mechanics to perform computations, instead of classical physics. The quantum computers offer an essential speed advantage over classical computers. This speed advantage is so significant that many researchers believe that no conceivable amount of progress in classical computation would be able to overcome the gap between the power of classical computers and the power of quantum computers.

Quantum Computation and Quantum Information is the study of the information processing tasks that can be accomplished using quantum mechanical systems. Like many simple but profound ideas it was a long time before anybody thought of doing information processing using quantum mechanical systems. To see why this is the case, we must go back in time and look in turn at each of the fields which have contributed fundamental ideas to quantum computation and quantum information – quantum mechanics, computer science, information theory, and cryptography. In the subsequent sections we discuss the different fundamental aspects of Quantum Computation briefly.

2.2 Quantum Mechanics

Quantum Mechanics is a mathematical framework or set of rules for the construction of physical theories. For example *Quantum Electrodynamics* – the physical theory that describes the interaction of atom and light can be mentioned. The relationship of quantum mechanics to specific physical theories like quantum electrodynamics is like the relationship of a computer's Operating System to specific application software. The rules of quantum mechanics are simple but even experts find them counterintuitive. Perhaps the long-standing desire of the physicists to better understand quantum mechanics set a ground for Quantum Computation and Quantum Information. One of the goals of Quantum Computation and Quantum Information is to develop tools which sharpen our intuition about quantum mechanics, and make its predictions more transparent to human minds.

Despite the intense interest, efforts to build quantum information processing systems have resulted in modest success to date. Small quantum computers, capable of doing few operations on a few qubits represent the state of the art in quantum computation. Experimental prototypes for doing quantum cryptography have been demonstrated. However, it remains a great challenge to physicists and engineers of the future to develop techniques for making large-scale quantum information processing a reality.

2.3 Quantum bits

The quantum counterpart of the classical binary digit (bit) is *qubit*. We are already familiar with the *Dirac notation* of the states of a qubit – $|0\rangle$ and $|1\rangle$. This is the standard notation for states in quantum mechanics.

2.3.1 Single qubit

The difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$. It is also possible to form *linear combination* of states, also known as *superposition*:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

The numbers α and β are complex numbers. $|0\rangle$ and $|1\rangle$ are two special cases and called *computational basis states*. The state of a qubit can also be thought as a vector in a two dimensional complex vector space. Unlike the bits, a qubit can not be examined to determine whether it is in state $|0\rangle$ or $|1\rangle$. Instead when a qubit is measured, we get either the result 0, with probability $|\alpha|^2$, or the result 1, with probability $|\beta|^2$ and naturally $|\alpha|^2 + |\beta|^2 = 1$. Thus, a qubit's state is a unit vector in a two dimensional complex vector space. If a qubit is in the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, then it will give the result 0 fifty percent of the time, and the result 1 fifty percent of the time when measured.

Many different physical systems can be used to realize qubits, some of them are mentioned below:

- i. two different polarizations of a photon,
- ii. the alignment of a nuclear spin in a uniform magnetic field,
- iii. two states of an electron orbiting a single atom (see Figure 2.1).

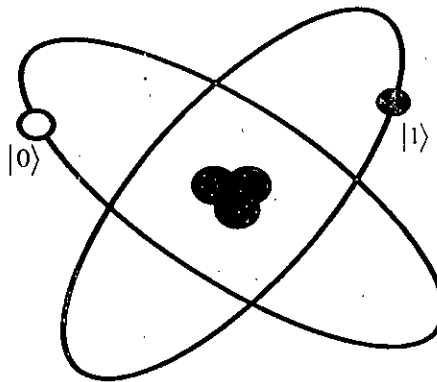


Fig 2.1: Qubit represented by two electronic levels in atom.

In the atom model, an electron can exist in either 'ground' state or 'excited' state, which we can call $|0\rangle$ and $|1\rangle$, respectively. By shining light on the atom, with appropriate energy and for an appropriate length of time, it is possible to move the electron from $|0\rangle$ to $|1\rangle$ state and vice versa. The interesting thing is that, by reducing the time of shining the atom, an electron with initial state $|0\rangle$ can be moved 'halfway' between $|0\rangle$ and $|1\rangle$, this state is often denoted as $|+\rangle$ state.

One important model to visualize the states of a qubit is *Bloch Sphere* as shown in Figure 2.2. The state of the qubit is represented by

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle.$$

The numbers θ and φ are real numbers and define a point on the unit three dimensional sphere. The Bloch sphere serves as an excellent test bed for ideas about quantum computation and quantum information.

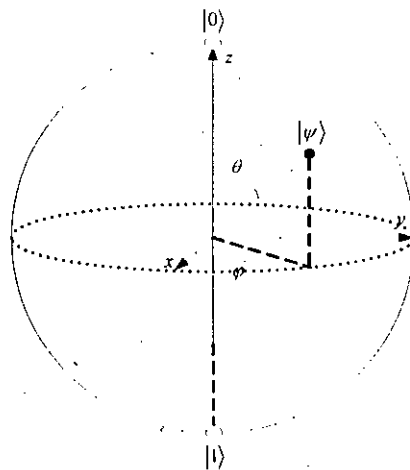


Fig 2.2: Bloch Sphere representation of a qubit.

Now the question is how much information is represented by a qubit? Paradoxically, there are an infinite number of points on the unit sphere. So, on principle, we can store an entire text of Rabindranath Tagore in the infinite binary extension of θ . However, this conclusion turns out to be misleading, because of the behavior of the qubit when observed. The measurement of a qubit will give only two states either 0 or 1. Furthermore, a measurement *changes* the state of a

qubit, collapsing it from its superposition of $|0\rangle$ and $|1\rangle$ to the specific state consistent with the measurement result. For example, if measurement of $|+\rangle$ gives 0, then the post-measurement state of the qubit will be $|0\rangle$. **Nobody knows** why this type of collapse occurs [41].

There is something conceptually important here, because when Nature evolves a closed quantum system of qubits, not performing any ‘measurement’, she apparently keeps track of all the continuous variables describing the states, like α and β . In a sense, in the state of a qubit, Nature conceals a great deal of ‘hidden information’ and the potential amount of this extra information grows exponentially with the number of qubits. Understanding this hidden *quantum information* lies at the heart of what makes quantum mechanics a powerful tool for information processing.

2.3.2 Multiple qubits

Let us consider the case of two qubits. If these were two classic bits, then there would be four possible states, 00, 01, 10, and 11. Correspondingly a two qubits system has four computational basis states denoted by $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$. A pair of qubits can also exist in a superposition of these four states, so the quantum states of two qubits involve associating a complex coefficient – sometimes called *amplitude* – with each computational basis state, such that the state vector describing the two qubits is

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle.$$

Similar to the case for a single qubit, the measurement result x ($= 00, 01, 10, \text{ or } 11$) occurs with probability $|\alpha_x|^2$, with the states of the qubits after measurement being $|x\rangle$. There exists the *normalization* condition $\sum_{x \in \{0,1\}^2} |\alpha_x|^2 = 1$ for the two qubits system. We could measure just a subset of the qubits, say the first qubit. Measuring the first qubit alone gives 0 with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$, leaving the post-measurement state

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

An important two qubit state is the *Bell state* or EPR² pair, $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$. This state is responsible for many surprises in quantum computation and quantum information. It is the key ingredient in quantum teleportation and super-dense coding. The Bell state has the property that upon measuring the first qubit, one obtains two possible results: 0 with probability $\frac{1}{2}$, leaving the post-measurement state $|\psi'\rangle = |00\rangle$, and 1 with probability $\frac{1}{2}$, leaving the post-measurement state $|\psi'\rangle = |11\rangle$. As a result, a measurement of the second qubit always give the result of the measurement of the first qubit. That indicates that the measurement outcomes are correlated.

More generally, we may consider a system with n qubits. The computational basis states of this system are of the form $|x_1 x_2 \dots x_n\rangle$, and so a quantum state of such a system is specified by 2^n amplitudes. For $n = 500$ this number is larger than the estimated number of atoms in the universe. Trying to store all those complex number is not possible on any conceivable classical computer.

2.4 Quantum Computation

Changes occurring to a quantum state can be described using the language of *Quantum Computation*. Analogous to the way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a *quantum circuit* containing wires and elementary quantum gates to carry around and to manipulate the quantum information. In this section we describe some simple quantum gates.

² Einstein-Podolsky-Rosen

2.4.1 Single qubit Gates

The only classical single-bit logic gate is the *NOT* gate. In a classical *NOT* gate the 0 and 1 states are interchanged. The quantum *NOT* gate analogously interchanges the $|0\rangle$ and $|1\rangle$ states. However, specifying the actions of the gate on the states $|0\rangle$ and $|1\rangle$ does not tell anything about what happens to superposition of states $|0\rangle$ and $|1\rangle$. In fact, the quantum *NOT* gate acts linearly, that is, it takes the state $\alpha|0\rangle + \beta|1\rangle$ to the corresponding state in which the role of $|0\rangle$ and $|1\rangle$ have been interchanged, $\alpha|1\rangle + \beta|0\rangle$. This linear behavior is a general property of quantum mechanics and very well motivated empirically.

There is a convenient way of representing the quantum *NOT* gate in matrix form, which follows directly from the linearity of quantum gates. Suppose we define a matrix X to represent the quantum *NOT* gate as follows:

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

If the quantum state $\alpha|0\rangle + \beta|1\rangle$ is written in a vector notation as:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

with the top entry corresponding to the amplitude of $|0\rangle$ and the bottom entry corresponding to the amplitude of $|1\rangle$, then the corresponding output from the *NOT* gate is:

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}.$$

So the quantum gates for single qubit can be described by two by two matrices. The appropriate condition on the matrix U representing a single qubit gate is that U be unitary [self], that is $U^\dagger U = I$. Where U^\dagger is the *adjoint* of U (obtained by transposing and then complex conjugating U), and I is the two by two identity matrix. For example, it easy to verify that $X^\dagger X = I$.

This unitarity constraint is the only constraint on quantum gates. **Any unitary matrix specifies a valid quantum gate** [41]. The unitary quantum gates are always *reversible*, since the inverse of a unitary matrix is also unitary, thus a quantum gate can always be inverted by another quantum gate.

There are a number of single qubit gates. Two very important ones are the Z gate:

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

which leaves $|0\rangle$ unchanged, and flip the sign of $|1\rangle$ to give $-|1\rangle$, and the *Hadamard* gate,

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

This gate is sometimes referred as 'square-root of *NOT*' gate. This is because this gate turns a $|0\rangle$ into $(|0\rangle+|1\rangle)/\sqrt{2}$, halfway between $|0\rangle$ and $|1\rangle$, and turns a $|1\rangle$ into $(|0\rangle-|1\rangle)/\sqrt{2}$, halfway between $|0\rangle$ and $|1\rangle$. However, H^2 is not a *NOT* gate, as simple algebra shows that $H^2=I$, and thus applying H twice to a state dose nothing to it.

The Hadamard gate is one of the most useful quantum gates and the Bloch sphere in Figure 2.3 shows its operation. The operation is a rotation of the sphere about the \hat{y} axis by 90° , followed by a reflection through the $\hat{x}-\hat{y}$ plane.

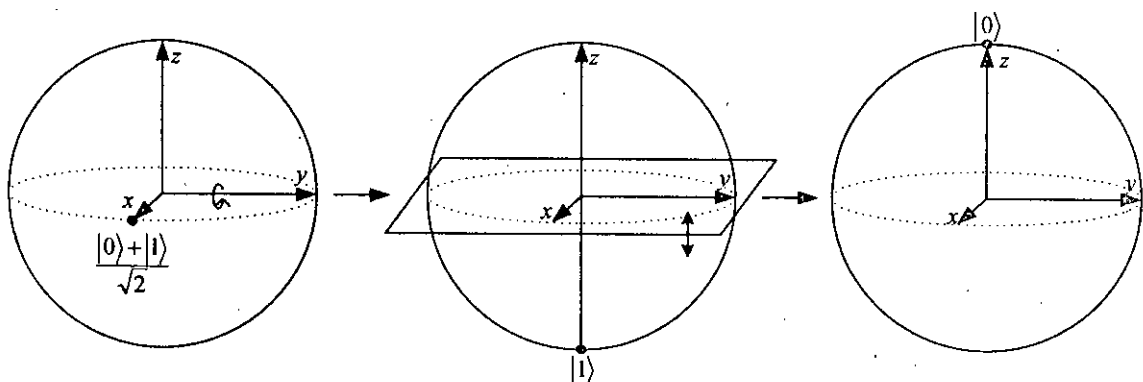


Fig 2.3: Visualization of Hadamard gate on the Bloch sphere, acting on the input state $(|0\rangle+|1\rangle)/\sqrt{2}$.

There are infinitely many single qubit gates while there is only one single bit classical gate. Some of the important single qubit gate along with the classical single bit gate are shown in Figure 2.4.

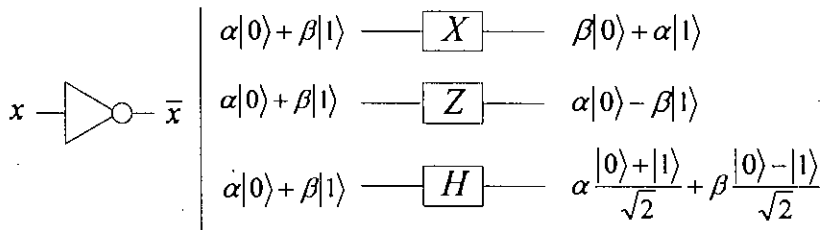


Fig 2.4: Single bit (left) and single qubit (right) logic gates.

2.4.2 Multiple qubit Gates

The prototypical multi-qubit quantum logic gate is the *Controlled-NOT* or *CNOT* gate. It is also known as binary Feynman gate. Figure 2.5 shows the graphic and matrix representation of the *CNOT* gate.

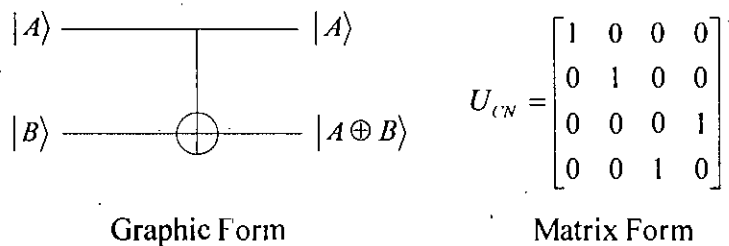


Fig 2.5: Graphic and Matrix representation of *CNOT* gate.

The gate has two input qubits, known as *control* qubit (top line) and *controlled* qubit or *target* qubit (bottom line). If the control qubit is set to 0, then the controlled qubit is left alone (pass through). If the control qubit is set to 1, then the target qubit is flipped. In equations:

$$|00\rangle \rightarrow |00\rangle; |01\rangle \rightarrow |01\rangle; |10\rangle \rightarrow |11\rangle; |11\rangle \rightarrow |10\rangle.$$

Another way of describing the *CNOT* gate is as a generalization of the classical *XOR* gate. The matrix representation of the *CNOT* gate is denoted by U_{CN} is a unitary matrix since $U_{CN}^\dagger U_{CN} = I$.

Suppose U be any unitary matrix acting on a number n of qubits, so U can be regarded as a quantum gate on those qubits. Then we can have a *Controlled- U* gate which is a natural extension of the *Controlled-NOT* gate [Figure 2.6].

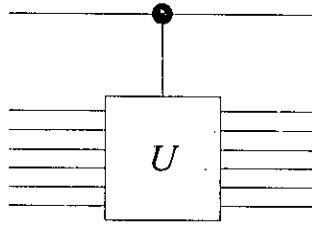


Fig: 2.6: Controlled- U gate.

Such a gate has a single *control qubit*, indicated by the line with the black dot, and n *target qubits*, indicated by the boxed U . If the controlled qubit is set to 0 then nothing happens to the target qubits. If the control qubit is set to 1 then the gate U is applied to the target qubits. The *CNOT* gate is a specific *Controlled- U* gate with $U = X$.

Of course, there are many interesting quantum gates other than the *CNOT* gate. However, in a sense the *CNOT* gate and single qubit gates are the prototypes for all other gates because of the following remarkable universality result: *Any multiple qubit logic gate can be composed from CNOT and single qubit gates*. It is the quantum binary parallel of the universality of the *NAND* gate. The following section provides a comprehensive proof of the fact.

2.4.3 Single qubit and *CNOT* Gates are Universal

Here we shall show that single qubit and *CNOT* gates can be used to implement an arbitrary unitary operation on n qubits, and therefore are universal for quantum computation.

Suppose U is a two-level unitary matrix on an n qubit quantum computer. Suppose in particular that U acts non-trivially on the space spanned by the computational basis states $|s\rangle$ and $|t\rangle$, where $s = s_1 \dots s_n$ and $t = t_1 \dots t_n$ are the binary expansions

for s and t . Let \tilde{U} be the non-trivial 2×2 unitary submatrix of U ; \tilde{U} can be thought of as a unitary operator on a single qubit.

Our immediate goal is to construct a circuit implementing U , using single qubit and *CNOT* gates only. To do this we need to make use of *Gray codes*. Suppose we have distinct binary numbers, s and t . A Gray code connecting s and t is a sequence of binary numbers starting with s and concluding with t , such that adjacent members of the list differ in exactly one bit. For instance, with $s = 101001$ and $t = 110011$ we have the gray code:

$$\begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{array}$$

Let g_1 through g_m be the elements of a Gray code connecting s and t , with $g_1 = s$ and $g_m = t$. Note that we can always find a Gray code such that $m \leq n + 1$ since s and t can differ in at most n locations.

The basic idea of the quantum circuit implementing U is to perform a sequence of gates effecting the state changes $|g_1\rangle \rightarrow |g_2\rangle \rightarrow \dots \rightarrow |g_{m-1}\rangle$, then to perform a controlled- \tilde{U} operation, with the target qubit located at the single bit where g_{m-1} and g_m differ, and then to undo the first stage, transforming $|g_{m-1}\rangle \rightarrow |g_{m-2}\rangle \rightarrow \dots \rightarrow |g_1\rangle$. The implementation of these operations are precisely described as follows. The first step is to swap the states $|g_1\rangle$ and $|g_2\rangle$. Suppose g_1 and g_2 differ at the i th digit. Then the swap is accomplished by performing a controlled bit flip on the i th qubit, conditional on the values of other qubits being identical to those in both g_1 and g_2 . Next a controlled operation to swap $|g_2\rangle$ and $|g_3\rangle$. It is continued in this fashion until the qubits $|g_{m-2}\rangle$ and $|g_{m-1}\rangle$ are swapped. The effect of this sequence of $m - 2$ operations is to achieve the operation

$$|g_1\rangle \rightarrow |g_{m-1}\rangle$$

$$|g_2\rangle \rightarrow |g_1\rangle$$

$$|g_3\rangle \rightarrow |g_2\rangle$$

.....

$$|g_{m-1}\rangle \rightarrow |g_{m-2}\rangle$$

All other computational basis states are left unchanged by this sequence of operations. Next, suppose $|g_{m-1}\rangle$ and $|g_m\rangle$ differ in the j th bit. We apply a controlled- \tilde{U} operation with the j th qubit as target, conditional on the other qubits having the same values as appear in both $|g_{m-1}\rangle$ and $|g_m\rangle$. Finally the U operation is completed by undoing the swap operations as mentioned earlier.

A simple example will illustrate the procedure further. Suppose we wish to implement the two-level unitary transformation

$$U_x = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 & 0 & c \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 & d \end{bmatrix}$$

Here, $a, b, c,$ and d are any complex numbers such that $\tilde{U} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is a unitary matrix. Notice that U_x acts non-trivially only on the states $|000\rangle$ and $|111\rangle$. We write Gray code connecting 000 and 111:

$$\begin{array}{ccc} A & B & C \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array}$$

From this we read off the required circuit, shown in Figure 2.7.

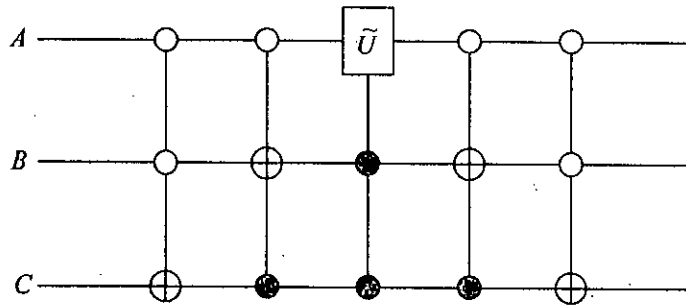


Fig 2.7: Circuit implementing the two-level unitary operation defined by U_x .

The first two gates shuffle the states so that $|000\rangle$ gets swapped with $|011\rangle$. Next the operation \tilde{U} is applied to the first qubit of the states $|011\rangle$ and $|111\rangle$, conditional on the second and third qubits being in the state $|11\rangle$. Finally, we unshuffle the states, ensuring that $|011\rangle$ gets swapped back with the state $|000\rangle$.

2.4.4 Quantum Circuits

A simple quantum circuit realizing an arbitrary function is shown in Figure 2.7. This circuit is to be read from left-to-right. Each line in the circuit represents a quantum wire in the quantum circuit. This wire does not necessarily correspond to a physical wire, it may correspond instead to the passage of time, or perhaps to a physical particle such as a photon moving from one location to another through space. It is conventional to assume that the state input to the circuit is a computational basis state, usually the states consisting of all $|0\rangle$ s.

There are a few features allowed in classical circuits that are not usually present in quantum circuits as listed below:

- Quantum Circuits do not allow any ‘loop’, that is, feedback from one part of the circuit to another. Quantum circuits are said to be *acyclic*.
- Wires can not be joined together. In classical circuits the wires can be joined and the resulting single wire contains the *OR* or *AND* of the joined wires, this is known as *FANIN*. Since this operational is not reversible, it is not available in quantum circuits.

- The inverse of *FANIN* operation, *FANOUT*, is also absent for the same reason. Besides, *FANOUT* operation requires to make copies, quantum mechanics does not allow cloning of the qubits.

Let us examine the third issue, copying a qubit, in a greater detail in the following section.

2.4.5 Can a qubit be Copied?

Consider the task of copying a classical bit. This may be done using a classical *CNOT* gate, which takes in the bit to copy (in some unknown state x) and a scratchpad bit initialized to 0, as shown in Figure 2.8(a). The output is two bits both of which are in the same state x .

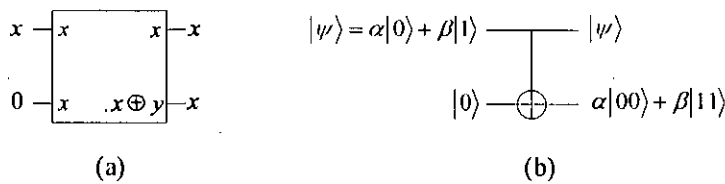


Fig 2.8: Classical and Quantum circuit to copy an unknown bit or qubit.

Now, let us try to copy a qubit in the unknown state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the same manner by using a *CNOT* gate [Figure 2.8(b)]. The input state of the two qubits may be written as

$$[\alpha|0\rangle + \beta|1\rangle]|0\rangle = \alpha|00\rangle + \beta|10\rangle.$$

The function of the *CNOT* is to negate the second qubit when the first qubit is 1, and thus the output is simply $\alpha|00\rangle + \beta|11\rangle$. Have we successfully copied $|\psi\rangle$?

That is we have created the state $|\psi\rangle|\psi\rangle$? In the case where $|\psi\rangle = |0\rangle$ or $|\psi\rangle = |1\rangle$ that is indeed what the circuit does; it is possible to use quantum circuits to copy classical information encoded as a $|0\rangle$ or a $|1\rangle$. However, for a general state $|\psi\rangle$ we see that

$$|\psi\rangle|\psi\rangle = \alpha^2|00\rangle + \alpha\beta|01\rangle + \alpha\beta|10\rangle + \beta^2|11\rangle.$$

Comparing with $\alpha|00\rangle + \beta|11\rangle$, we see that unless $\alpha\beta = 0$ the ‘copying circuit’ above does not copy the quantum state input. In fact, it turns out to be *impossible*

to make a copy of an unknown quantum state. This property that, qubits cannot be copied, is known as *no-cloning* theorem, and it is one of the major differences between classical and quantum information.

The no-cloning can be explained in another way. A qubit contains ‘hidden’ information not directly accessible to measurement. When one of the qubits of the state $\alpha|00\rangle + \beta|11\rangle$ is measured, we obtain either 0 or 1 with probabilities $|\alpha|^2$ and $|\beta|^2$. However, when one qubit is measured, the state of the other one is completely determined, and no additional information can be gained about α and β . The extra hidden information carried in the original qubit $|\psi\rangle$ was lost in the first measurement, and cannot be regained. If, however, the qubit had been copied, then the state of the other qubit still contains some of that hidden information. Therefore, a copy cannot have been created.

2.4.6 Quantum Parallelism

Quantum Parallelism is a fundamental feature of quantum computation. Heuristically, and at the risk of over-simplifying, quantum parallelism allows quantum computers to evaluate a function $f(x)$ for many different values of x simultaneously. Let us see how quantum parallelism works, and some of its limitations.

Suppose $f(x):\{0,1\} \rightarrow \{0,1\}$ is a function with one bit domain and range. A convenient way to compute this function on a quantum computer is to consider a two-qubit quantum computer which starts in the state $|x,y\rangle$. With an appropriate sequence of logic gates, it is possible to transform this state into $|x,y \oplus f(x)\rangle$ where \oplus indicates modulo-2 addition; the first register is called the ‘data’ register and the second one is the ‘target’ register. We give the transformation defined by the map $|x,y\rangle \rightarrow |x,y \oplus f(x)\rangle$ a name, U_f , and note that it is easily shown to be unitary. If $y = 0$, then the final state of the second qubit is just the value $f(x)$.

$$\begin{array}{ccc}
 \frac{|0\rangle + |1\rangle}{\sqrt{2}} & \left| \begin{array}{c} x \\ y \end{array} \right. & \begin{array}{c} x \\ y \oplus f(x) \end{array} \\
 & U_f & \\
 & & \left| \psi \right\rangle
 \end{array}$$

Fig 2.9: Quantum circuit for evaluating $f(0)$ and $f(1)$ simultaneously. U_f is the quantum circuit which takes inputs like $|x, y\rangle$ to $|x, y \oplus f(x)\rangle$.

Consider the circuit in Figure 2.9, which applies U_f to an input not in the computational basis. Instead, the data register is prepared in the superposition $(|0\rangle + |1\rangle)/\sqrt{2}$, which can be created with a Hadamard gate acting on $|0\rangle$, then U_f is applied, resulting the state:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

This is a remarkable state! The different terms contain information about $f(0)$ and $f(1)$; it is almost as if we have evaluated $f(x)$ for two values of x simultaneously, a feature known as ‘Quantum Parallelism’. Unlike classical parallelism, where multiple circuits each built to compute $f(x)$ are executed simultaneously, here a single $f(x)$ circuit is employed to evaluate the function for multiple values of x simultaneously, by exploiting the ability of a quantum computer to be in superposition of different states. This procedure can easily be generalized to functions on an arbitrary number of bits (for detailed description see [41]).

2.5 Quantum Information

The term ‘quantum information’ is used in two distinct ways in the field of quantum computation and quantum information. The first usage is a broad catch-all for all manner of operations that might be interpreted as related to information processing using quantum mechanics. This use encompasses subjects such as

quantum computation, quantum teleportation, the no-cloning theorem, and virtually all other topics in this field.

The second use of 'quantum information' is much more specialized: it refers to the study of elementary quantum information processing tasks. Quantum information theory may look like a disordered zoo to the beginner, with many apparently unrelated subjects falling under the 'quantum information theory' domain. In part, that is because the subject is still under development, and it is not yet clear how all the pieces fit together. However, a few fundamental goals can be identified uniting work on quantum information theory:

- Identify elementary classes of static resources in quantum mechanics
 - One example is the qubit. Another example is the *bit*; classical physics arises as a special case of quantum physics, so it should also be of great relevance in quantum information theory.
- Identify elementary classes of static resources in quantum mechanics
 - A simple example is *memory*, the ability to store a quantum state over some period of time. Less trivial processes are quantum information transmission between two parties, copying a quantum state, and the process of protecting quantum information processing against the effect of noise.
- Quantify resource tradeoffs incurred performing elementary dynamic processes
 - For example, what are the minimal resources required to reliably transfer quantum information between two parties using a noisy communications channel?

2.6 Prospects for Quantum Information Processing

Building quantum information processing devices is a great challenge for scientists and engineers in the third millennium. The most fundamental question is whether there is any point of principle that prohibits us from doing one or more forms of quantum information processing? Two possible obstructions suggest

themselves: that noise may place a fundamental barrier to useful quantum information processing; or that quantum mechanics may fail to be correct.

The theory of quantum error-correcting codes strongly suggests that while quantum noise is a practical problem that needs to be addressed, it does not present a fundamental problem of principle. In particular, there is a *threshold theorem* for quantum computation. The theorem roughly states that provided a level of noise in a quantum computer can be reduced below a certain constant threshold value. Quantum error-correcting codes can push it even further, essentially *ad infinitum*, for a small overhead in the complexity of the computation.

A second possibility that may preclude quantum information processing is if quantum mechanics is incorrect. Indeed, probing the validity of quantum mechanics is one of the reasons behind the interest of building quantum information processing devices. Never before we have explored the regime of Nature in which complete control has been obtained over large-scale quantum systems. And perhaps Nature may reveal some new surprises in this regime which are not adequately explained by quantum mechanics. If this occurs, it will be a momentous discovery in the history of science, and can be expected to have considerable consequences in the other areas of science and technology, as did the discovery of quantum mechanics. Until and unless such events occur, we have no way of knowing how they might affect information processing, so we can assume that quantum mechanics is a complete and correct description of the world.

A clear picture of the relative power of classical and quantum information processing is needed in order to assess their relative merits. It requires further theoretical work on the foundations of quantum computation and quantum information. It would be useful to have a clear path of interesting applications at varying levels of complexity to aid researchers aiming to experimentally realize quantum information processing.

Chapter 3

Quantum Computer: Physical Realization

Given a path of potential applications for quantum information processing, how can it be achieved in real physical system? At the small scale of a few qubits there are already several working proposals for quantum information processing devices. Mainly there are three technologies proposed for the purpose

- i. Optical,
- ii. Ion Trap, and
- iii. Nuclear Magnetic Resonance.

The different realization technologies are discussed later in this chapter in greater details. But before the detailed description of specific technologies, let us consider some important relevant issues.

3.1 Realization of Quantum Computer

To realize a quantum computer, we must not only give qubits some robust physical representation, but also select a system in which they can be made to evolve as desired. Moreover, we must be able to prepare qubits in some specified set of initial states, and to measure the final output state of the system. The challenge of quantum realization is that the basic requirements can often only be partially met. A coin has two states, and makes a good bit, but a poor qubit because it cannot remain in a superposition state for a long time.

A single nuclear spin can be a very good qubit, because superposition of being aligned with or against an external magnetic field can last a long time – even for days. But their coupling to the world is so small that they are hard to measure the orientation of single nuclei. The constraints are opposing in general: a quantum computer has to be well isolated in order to retain its quantum properties, but at the same time its qubits have to be accessible so that they can be manipulated to

perform computations and to read out the results. A realistic implementation must strike a delicate balance between these constraints. Therefore, the relevant question is not '*how to build a quantum computer*', but rather, '*how good a quantum computer can be built*'.

3.2 Conditions for Quantum Computation

The basic requirements for quantum computation are the abilities to:

- i. Robustly represent quantum information
- ii. Perform a universal family of unitary transformations
- iii. Prepare a fiducial initial state
- iv. Measure the output result

These are discussed in the subsequent sections.

3.2.1 Representation of Quantum Information

For the purpose of computation, it is crucial that the set of accessible states should be finite. The position x of a particle along a one-dimensional line is not generally a good set of states for computation, even though the particle may be in a quantum state $|x\rangle$, or even some superposition $\sum_x c_x |x\rangle$. This is because x has a continuous range of possibilities, and the Hilbert space has infinite size, so in the absence of noise the information capacity is infinite. For example, in a perfect world, the entire works of Rabindronath Tagore could be stored in the infinite number of digits in the binary fraction $x = 0.0101101000101001110\dots$. This is clearly unrealistic; instead, the presence of noise reduces the number of distinguishable states to a finite number.

It is generally desirable to have some aspect of symmetry dictate the finiteness of the state space, in order to minimize decoherence. For example, a spin-1/2 particle lives in the Hilbert space spanned by the $|\uparrow\rangle$ and $|\downarrow\rangle$ states; the spin state cannot be anything outside this two-dimensional space, and thus is a nearly ideal quantum bit when well isolated.

If the choice of representation is poor, then decoherence will result. For example, a particle in a finite square well which is just deep enough to contain two bound states would make a mediocre quantum bit, because transitions from the bound states to the continuum of unbound states would be possible. These would lead to decoherence since they could destroy qubit superposition states. For single qubits, the figure of merit is the minimum lifetime of arbitrary superposition states.

3.2.2 Performing Unitary Transformation

Closed quantum systems evolve unitarily as determined by their Hamiltonians, but to perform quantum computation one must be able to control the Hamiltonian to effect an arbitrary selection from a universal family of unitary transformations. For example, a single spin might evolve under the Hamiltonian $H = P_x(t)X + P_y(t)Y$, where $P_{\{x,y\}}$ are classically controllable parameters. By manipulating P_x and P_y appropriately, one can perform arbitrary single spin rotations.

Any unitary transform can be composed from single spin operations and *Controlled-NOT* gates, and thus realization of those two kinds of quantum logic gates are natural goals for experimental quantum computation. However, implicitly required also is the ability to address individual qubits, and to apply these gates to selected qubits or pair of qubits. This is not simple to accomplish in many physical systems. For example, in ion trap, one can direct a laser at one of many individual ions to selectively excite it, but only as long as the ions are separated by a wavelength or more.

Two important figures of merit for unitary transforms are the minimum achievable fidelity, and the maximum time required to perform elementary operations such as single spin rotations or a *controlled-NOT* gate.

3.2.3 Preparation of Fiducial Initial States

One of the most important requirements for being able to perform a useful computation, even classically, is to be able to prepare the desired input. With classical machines, establishing a definite input state is rarely a difficulty – one merely sets some switches in the desired configuration and that defines the input state. But in case of quantum systems, this can be very difficult, depending on the realization of qubits. It is only necessary to be able to produce one quantum state with high fidelity, since a unitary transformation can turn it into any other desired input state. For example being able to put n spins into the $|00\dots 0\rangle$ state is good enough.

Input state representation is a significant problem for most physical systems. For example, ion can be prepared in good input states by physically cooling them into their ground state, but this is challenging. Moreover, for physical systems where ensembles of quantum computers are involved, extra concerns arise. In nuclear magnetic resonance, each molecule can be thought of as a single quantum computer, and a large number of molecules are needed to obtain measurable signal strength. Although qubits can remain in arbitrary superposition of states for relatively long times, it is difficult to put all of the qubits in all of the molecules into the same state, because the energy difference $\hbar\omega$ between the $|0\rangle$ and $|1\rangle$ states is much smaller than kT . On the other hand, simply letting the system equilibrate establishes it in a very well-known state, the thermal one, one the density matrix $\rho \approx e^{-S/k_B T} / Z$, where Z is a normalization factor required to maintain $\text{tr}(\rho)=1$.

Two figures of merit are relevant to input state preparation: the minimum fidelity with which the initial state can be prepared in a given state ρ_{in} and the entropy of ρ_{in} . The entropy is important because, for example, it is very easy to prepare the state $\rho_{in} = I/2^n$ with high fidelity, but that is a useless state for quantum computation, since it is invariant under unitary transforms. Ideally, the input state

is a pure state, with zero entropy. Generally, input states with non-zero entropy reduce the accessibility of the answer from the output result.

3.2.4 Measurement of Output Results

For the purpose of our present discussion, let us think of measurement as process of coupling one or more qubits to a classical system such that after some interval of time, the state of the qubits is indicated by the state of a classical system. For example, a qubit state $\alpha|0\rangle + \beta|1\rangle$, represented by ground and excited states of a two-level atom, might be measured by pumping the excited state and looking for fluorescence. If an electrometer indicates that fluorescence had been detected by a photomultiplier tube, then the qubit would collapse into the $|1\rangle$ state; this would happen with probability $|\beta|^2$. Otherwise, the electrometer would detect no charge, and the qubit would collapse into the $|0\rangle$ state.

An important characteristic of measurement process for quantum computation is the wave function collapse which describes what happens when a projective measurement is performed. The output from a good quantum algorithm is a superposition state which gives a useful answer with high probability when measured. For example, one step in Shor's quantum factoring algorithm is to find an integer r from the measurement result, which is an integer close to qc/r , where q is the dimension of a Hilbert space. The output state is actually in a nearly uniform superposition of all possible values of c , but a measurement collapses this into a single, random integer, thus allowing r to be determined with high probability.

Many difficulties with measurement can be imagined; for example, inefficient photon counters and amplifier thermal noise can reduce the information obtained about measured qubit states in the scheme just described. Furthermore, projective measurements (sometimes called 'strong' measurements) are often difficult to implement. They require that the coupling between the quantum and classical systems be large, and switchable. Measurement should not occur when not

desired; otherwise they can be a decoherence process. However, strong measurements are not necessary; weak measurements which are performed continuously and never switched off are usable for quantum computation. This is made possible by completing the computation in time short compared with the measurement coupling, and by using large ensembles of quantum computers. These ensembles together give an aggregate signal which is macroscopically observable and indicative of the quantum state. Again, use of an ensemble introduces additional problems. For example, in the factoring algorithm, if the measurement output is $q\langle c \rangle / r$, the algorithm would fail because $\langle c \rangle$, the average value of c , is not necessarily an integer. Fortunately, it is possible to modify quantum algorithms to work with ensemble average readouts.

A good figure of merit for measurement capability is the signal to noise ratio (SNR). This accounts for measurement inefficiency as well as inherent signal strength available from coupling a measurement apparatus to the quantum system.

3.3 Harmonic Oscillator Quantum Computer

Before continuing on to describe a complete physical model for a realizable quantum computer, let us consider a very elementary system – the simple harmonic oscillator – and discuss why it does not serve as a good quantum computer. The formalism used in this example will also serve as a basis for studying other physical system.

3.3.1 The Quantum Harmonic Oscillator (QHO)

The harmonic oscillator is an extremely important and useful concept in the quantum description of the physical world, and a good way to begin to understand its properties is to determine the energy eigenstates of its Hamiltonian. One way to do this is simply to solve the Schrödinger equation

$$\frac{\hbar^2}{2m} \frac{d^2\psi_n(x)}{dx^2} + \frac{1}{2}m\omega^2 x^2\psi_n(x) = E\psi_n(x)$$

For $\psi_n(x)$ and the eigenenergies E , subject to $\psi(x) \rightarrow 0$ at $x = \pm\infty$, and $\int |\psi(x)|^2 = 1$; the first five solutions are sketched here in Figure 3.1. The wave functions describe the probability amplitudes that a particle in the harmonic oscillator will be found at different positions within the potential.

Although these pictures may give some intuition about what a physical system is doing in co-ordinate space, we will generally be more interested in the abstract algebraic properties of the states. Specifically, suppose $|\psi\rangle$ satisfies with energy E . Then defining operators a and a^\dagger as $a = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x + ip)$, and

$$a^\dagger = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x - ip),$$

we find that since $[H, a^\dagger] = \hbar\omega a^\dagger$,

$$H a^\dagger |\psi\rangle = ([H, a^\dagger] + a^\dagger H) |\psi\rangle = (\hbar\omega + E) a^\dagger |\psi\rangle,$$

that is, $a^\dagger |\psi\rangle$ is an eigenstate of H , with energy $E + \hbar\omega$! Similarly, $a |\psi\rangle$ is an eigenstate with energy $E - \hbar\omega$. Because of this, a and a^\dagger are called rising and lowering operators. It follows that $a^{\dagger n} |\psi\rangle$ are eigenstates for any integer n , with eigenenergies $E + n\hbar\omega$. There are thus an infinite number of energy eigenstates, whose energies are equally spaced apart, by $\hbar\omega$.

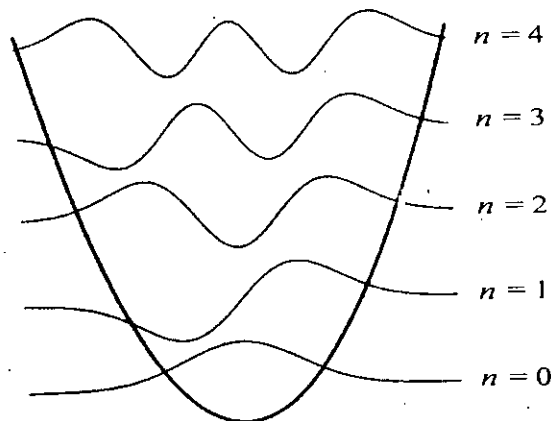


Fig 3.1: Sketch of the first five solutions of the Schrödinger equation for $\psi_n(x)$.

Moreover, since H is positive definite, there must be some $|\psi_0\rangle$ for which $a|\psi_0\rangle = 0$; this is the ground state – the eigenstate H with lowest energy. These results efficiently capture the essence of the quantum harmonic oscillator, and allow us to use a compact notation $|n\rangle$ for the eigenstates, where n is an integer, and $H|n\rangle = \hbar(n+1/2)\omega|n\rangle$.

3.3.2 Physical Apparatus for QHO

An example of simple harmonic oscillator is a particle in a parabolic potential well, $V(x) = m\omega^2 x^2 / 2$. In the classical world, this could be a mass on a spring, which oscillates back and forth as energy is transferred between the potential energy of the spring and the kinetic energy of the mass. It could also be a resonant electrical circuit, where the energy sloshes back and forth between the capacitor and the inductor. In these systems, the total energy of the system is a continuous parameter.

In the quantum domain, which is reached when the coupling to the external world becomes very small, the total energy of the system can only take on a discrete set of values. An example is given by a single mode of electromagnetic radiation trapped in a high Q cavity; the total amount of energy can only be integer multiples of $\hbar\omega$, an energy scale which is determined by the fundamental constant \hbar and the frequency of the trapped radiation, ω .

The set of discrete energy eigenstates of a simple harmonic oscillator can be labeled as $|n\rangle$, where $n = 0, 1, \dots, \infty$. The relationship to quantum computation comes by taking a finite subset of these states to represent qubits. These qubits will have lifetimes determined by the physical parameters such as the cavity quality factor Q , which can be made very large by increasing the reflectivity of the cavity walls. Moreover, unitary transformations can be applied by simply allowing the system to evolve in time. However, there are problems with this scheme, as will become clear below. We begin by studying the system Hamiltonian, and then

discuss how one might implement simple quantum logic gates such as the *Controlled-NOT*.

3.3.3 The Hamiltonian for QHO

The Hamiltonian for a particle in a one dimensional parabolic potential is

$$H = \frac{p^2}{2m} + \frac{1}{2}m\omega^2 x^2$$

where p is the particle momentum operator, m is the mass, x is the position operator, and ω is related to the potential depth. The expression for H can also be written as: $H = \hbar\omega\left(\alpha^\dagger\alpha + \frac{1}{2}\right)$, where α^\dagger and α are creation and annihilation operators, defined as

$$\alpha = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x + ip),$$

$$\alpha^\dagger = \frac{1}{\sqrt{2m\hbar\omega}}(m\omega x - ip).$$

The zero point energy $\hbar\omega/2$ contributes an unobservable overall phase factor, which can be disregarded for our present purpose. The eigenstates $|n\rangle$ of H , where $n = 0, 1, \dots$, have the properties:

$$\alpha^\dagger a|n\rangle = n|n\rangle$$

$$\alpha^\dagger|n\rangle = \sqrt{n+1}|n+1\rangle$$

$$a|n\rangle = \sqrt{n}|n-1\rangle$$

later, we will find it convenient to express interaction with a simple harmonic oscillator by introducing additional terms involving α^\dagger and α , and interaction between oscillators with term such as $a_1^\dagger a_2 + a_1 a_2^\dagger$. For now, however, we confine our attention to a single oscillator.

Time evolution of the eigenstate is given by solving the Schrödinger equation, from which we find that the state $|\psi(0)\rangle = \sum_n c_n(0)|n\rangle$ evolves in time to become

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle = \sum_n c_n e^{-in\omega t} |n\rangle.$$

We will assume for the purpose of discussion that an arbitrary state can be perfectly prepared, and that the state of the system can be protectively measured, but otherwise, there are no interactions with the external world, so that the system is perfectly closed.

3.3.4 Quantum Computation for QHO

Suppose we want to perform quantum computation with the single simple harmonic oscillator described above. What can be done? The most natural choice for representation of qubits are the energy eigenstates $|n\rangle$. This choice allows us to perform a *Controlled-NOT* gate in the following way. Recall that this transformation performs the mapping

$$\begin{aligned} |00\rangle_L &\rightarrow |00\rangle_L \\ |01\rangle_L &\rightarrow |01\rangle_L \\ |10\rangle_L &\rightarrow |11\rangle_L \\ |11\rangle_L &\rightarrow |10\rangle_L \end{aligned}$$

On two qubits states (here, the subscript L is used to clearly distinguish 'logical' states in contrast to the harmonic oscillator basis states). Let us *encode* these two qubits using the mapping

$$\begin{aligned} |00\rangle_L &\rightarrow |0\rangle \\ |01\rangle_L &\rightarrow |2\rangle \\ |10\rangle_L &\rightarrow (|4\rangle + |1\rangle)/\sqrt{2} \\ |11\rangle_L &\rightarrow (|4\rangle - |1\rangle)/\sqrt{2} \end{aligned}$$

Now suppose that at $t=0$ the system is started in a state spanned by these basis states, and we simply evolved the system to forward to time $t = \pi/\hbar\omega$. This causes the energy eigenstates to undergo the transformation $|n\rangle \rightarrow \exp(-i\pi a^\dagger a)n) = (-1)^n |n\rangle$, such that $|0\rangle$, $|2\rangle$, and $|4\rangle$ stay unchanged, but $|1\rangle \rightarrow -|1\rangle$. As a result, we obtained the desired *Controlled-NOT* gate transformation.

In general, a necessary and sufficient condition for a physical system to be able to perform a unitary transform U is simply that the time evolution operator for the

system, $T = \exp(-iHt)$, defined by its Hamiltonian H , has nearly the eigenvalue spectrum as U . In the case above, the Controlled-NOT gate was simple to implement because it only has eigenvalue +1 and -1; it was straightforward to arrange an encoding to obtain the same eigenvalues from the time evolution operator for the harmonic oscillator. The Hamiltonian for an oscillator could be perturbed to realize nearly any eigenvalue spectrum, and any number of qubits could be represented by simply mapping them into the infinite number of eigenstates of the system. This suggests that perhaps one might be able to realize an entire quantum computer in a single simple harmonic oscillator.

3.3.5 Drawbacks of QHO

Of course there are many problems with the above scenario. Clearly, one will not always know the eigenvalue spectrum of the unitary operator for a certain quantum computation, even though one may know how to construct the operator from elementary gates. In fact, for most problems addressed by quantum algorithms, knowledge of the eigenvalue spectrum is tantamount to knowledge of the solution! Another obvious problem is that the technique used above does not allow one computation to be cascaded with another, because in general, cascading two unitary transforms results in a two transform with unrelated eigenvalues.

Finally, the idea of using a single harmonic oscillator to perform quantum computation is flawed because it neglects the principle of digital representation of information. A Hilbert space of 2^n dimensions mapped into the state space of a single harmonic oscillator would have to allow for the possibility of states with energy $2^n \hbar \omega$. In contrast, the same by using Hilbert space could be obtained in two-level quantum systems, which has energy of at most $n \hbar \omega$. Similar comparisons can be made between a classical dial with 2^n settings, and a register of n classical bits. Quantum computation builds upon a digital computation, not analog computation.

The main features of the harmonic oscillator quantum computer are summarized below (each system we consider will be summarized similarly, at the end of the

corresponding section.) With this, we leave behind us the study of single oscillators, and turn next to systems of harmonic oscillators, made of photons and atoms.

3.3.6 Summary of QHO Properties

The issues related to the Harmonic Oscillator Quantum Computer can be summarized as follows:

- **Qubit Representation:** Energy levels $|0\rangle, |1\rangle, \dots, |2^n\rangle$ of a single quantum oscillator give n qubits.
- **Unitary Evolution:** Arbitrary transforms U are realized by matching their eigenvalue spectrums to that given by the Hamiltonian $H = a^\dagger a$.
- **Initial State Preparation:** Not considered.
- **Readout:** Not considered.
- **Drawbacks:** Not a digital representation. Also matching eigenvalues to realize transformations is not feasible for arbitrary U , which generally have eigenvalues.

3.4 Optical Photon Quantum Computer (OPQC)

An attractive physical system for representing a quantum bit is the optimal photon. Photons are charge less particles, and do not interact very strongly with each other, or even with most matter. They can be guided along long distances with low loss in optical fibers, delayed efficiently using phase shifters, and combined easily using beamsplitters. Photons exhibit signature quantum phenomena, such as the interference produced in two-slit experiments. Furthermore, in principle, photons can be made to interactions. There are problems with this ideal scenario; nevertheless, many things can be learned from studying the components, architecture, and drawbacks of an optimal photon quantum information processor, as we shall see in this section.

3.4.1 Physical Apparatus of an OPQC

Let us begin by considering what single photons are, how they can represent quantum states, and the experimental components useful to manipulate photons. The classical behavior of phase shifters, beamsplitters, and nonlinear optical Kerr media is described.

Photons can represent qubits in the following manner. As we saw in the discussion of the simple harmonic oscillator, the energy in an electromagnetic cavity is quantized in units of $\hbar\omega$. Each such quantum is called photon. It is possible for a cavity to contain a superposition of zero or one photon, a state which could be expressed as a qubit $c_0|0\rangle + c_1|1\rangle$, but we shall do something different. Let us consider two cavities, whose total energy is $\hbar\omega$, and take the two states of a qubit as being whether the photon is in one cavity ($|01\rangle$) or the other ($|10\rangle$). The physical state of a superposition would thus be written as $c_0|01\rangle + c_1|10\rangle$; this is known as *dual-rail* representation. The actual focus is on single photons traveling as a wavepacket through free space, rather than inside a cavity, one can imagine this as having a cavity moving along with the wavepacket. Each cavity in the qubit state will thus correspond to a different spatial mode.

One scheme for generating single photons in laboratory is by attenuating the output of a laser. A laser outputs a state known as coherent state, $|\alpha\rangle$, defined as

$$|\alpha\rangle = e^{-|\alpha|^2/2} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle,$$

where $|n\rangle$ is an n -photon energy eigenstate. It suffices to understand just that coherent states are naturally radiated from driven oscillators such as a laser when pumped high above its lasing threshold. Note that the mean energy is $\langle\alpha|n|\alpha\rangle = |\alpha|^2$. When attenuated, a coherent state just becomes a weaker coherent state, and a weaker coherent state can be made to have just one photon, with high probability.

Better synchronicity can be achieved using parametric down-conversion. This involves sending photons of frequency ω_0 into a nonlinear optical medium such as KH_2PO_4 to generate photon pairs at frequencies $\omega_1 + \omega_2 = \omega_0$. Momentum is also preserved, such that $\vec{k}_1 + \vec{k}_2 = \vec{k}_3$, so that when a single ω_2 photon is (destructively) detected, then a single ω_0 photon is known to exist (see Figure 3.2).

By coupling this to a gate, which is opened only when a single photon is detected, and by appropriately delaying outputs of multiple down-conversion sources, one can, in principle, obtain multiple single photons propagating in time synchronously, within the time resolution of the time detector and gate.

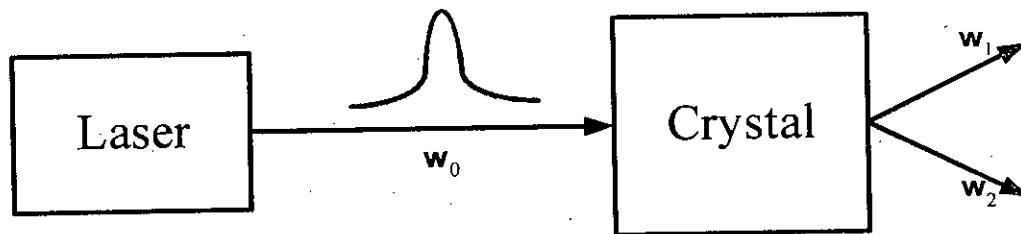


Fig 3.2: Parametric down-conversion for generation of single photons.

Three of the most experimentally accessible devices for manipulating photon states are

- i. Mirrors,
- ii. Phase shifters, and
- iii. Beamsplitters.

Mirrors with less than 0.01% loss are not unusual. A phase shifter is nothing more than a slab of transparent medium with index of refraction n different from that of free space, n_0 . The beamsplitter is nothing more than a partially silvered piece of glass which reflects a fraction R of incident light, and transmits $1-R$. In the laboratory, a beamsplitter is usually fabricated from two prisms, with a thin metallic layer sandwiched in-between, schematically drawn as in Figure 3.3. It is

convenient to define the angle θ of a beamsplitter as $\cos\theta = R$. The two inputs and two outputs of this device are related by

$$\begin{aligned} a_{out} &= a_{in} \cos\theta + b_{in} \sin\theta \\ b_{out} &= -a_{in} \sin\theta + b_{in} \cos\theta. \end{aligned}$$

Nonlinear optics provides another useful component: a material whose index of refraction n is proportional to the total intensity I of the light going through it: $n(I) = n + n_2 I$. This is known as the optical Kerr effect, and it occurs (very weakly) in materials as mundane as glass and sugar water. Experimentally, the relevant behavior is that when two beams of light of equal intensity are nearly co-propagate through a Kerr medium, each beam will experience an extra phase shift of $e^{n_2 I L / c_0}$ compared to what happens in the single beam case.

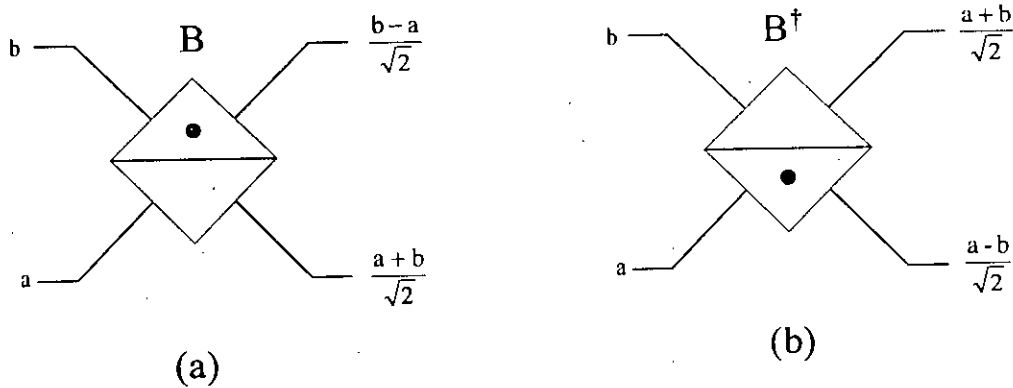


Fig 3.3: Schematic of an optical beamsplitter. (b) is the inverse of (a). [$\theta = \pi/4$]

3.4.2 Quantum Computation with OPQC

Arbitrary unitary transforms can be applied to quantum information, encoded with single photons in the $c_0|01\rangle + c_1|10\rangle$ dual-rail representation, using phase shifters, beamsplitters, and nonlinear optical Kerr media. How they work can be described by giving a quantum-mechanical Hamiltonian description of each of them.

The time evolution of a cavity mode of electromagnetic radiation is modeled quantum-mechanically by a harmonic oscillator, as was shown in the previous section. $|0\rangle$ is the vacuum state, $|1\rangle = a^\dagger|0\rangle$ is a single photon state, and in

general, $|n\rangle = \frac{a^{\dagger n}}{\sqrt{n!}}|0\rangle$ is an n -photon state, where a^\dagger is the creation operator for the mode. Free space evolution is described by the Hamiltonian

$$H = \hbar\omega a^\dagger a,$$

and applying $|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle = \sum_n c_n e^{-in\omega t}|n\rangle$, we find that the state $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ evolves in time to become $|\psi(t)\rangle = c_0|0\rangle + c_1 e^{-i\omega t}|1\rangle$. Note that the dual-rail representation is convenient because free evolution only changes $|\psi\rangle = c_0|01\rangle + c_1|10\rangle$ by an overall phase, which is undetectable. Thus, for that manifold of states, the evolution Hamiltonian is zero.

Phase Shifter: A phase shifter P acts just like normal time evolution, but at a different rate, and localized to only the modes going through it. That is because light slows down in a medium with larger index of refraction; specifically, it takes $\Delta \equiv (n - n_0)L/c_0$ more time to propagate a distance L in a medium with index of refraction n than in vacuum. For example, the action P on the vacuum state is to do nothing: $P|0\rangle = |0\rangle$, but on a single photon state, one obtains $P|1\rangle = e^{i\Delta}|1\rangle$.

P performs a useful logical operation on a dual-rail state. Placing a phase shifter in one mode retards its phase evolution with respect to another mode, which travels the same distance but without going through the shifter. For dual-rail states this transforms $c_0|01\rangle + c_1|10\rangle$ to $c_0 e^{-i\Delta/2}|01\rangle + c_1 e^{i\Delta/2}|10\rangle$, up to an irrelevant overall phase. This operation is actually nothing more than a rotation

$$R_z(\Delta) = e^{-iZ\Delta/2},$$

where we take as the logical zero $|0_L\rangle = |01\rangle$ and one $|1_L\rangle = |10\rangle$, and Z is the usual Pauli operator. One can thus think of P as resulting from time evolution under the Hamiltonian

$$H = (n_0 - n)Z,$$

where $P = \exp(-iHL/c_0)$. The following circuit in Figure 3.4 transforms a dual-

rail state by $|\psi_{out}\rangle = \begin{bmatrix} e^{i\pi} & 0 \\ 0 & 1 \end{bmatrix} |\psi_{in}\rangle$,

where the top wire represents the $|01\rangle$ mode, the bottom as the $|10\rangle$ mode, and the boxed π represents a phase shift by π .

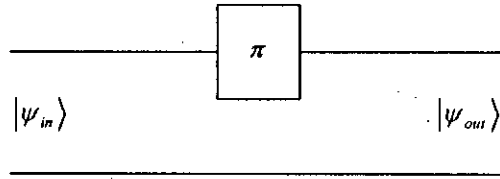


Fig 3.4: Optical circuit representing a phase shift by π .

Beamsplitter: A similar Hamiltonian description of the beamsplitter also exists. The beamsplitter acts on two modes which are described by the creation (annihilation) operators a (a^\dagger) and b (b^\dagger). The Hamiltonian is

$$H_{bs} = i\theta(ab^\dagger - a^\dagger b).$$

And the beamsplitter performs the unitary operation

$$B = \exp[-i\theta(a^\dagger b - ab^\dagger)].$$

The transformations effected by B on a and b are found to be $BaB^\dagger = a\cos\theta + b\sin\theta$ and $BbB^\dagger = -a\sin\theta + b\cos\theta$.

Nonlinear Kerr Media: The most important effect of Kerr media is the *cross Phase modulation* it provides between two modes of light. That is classically described by the n_2 term in $n(I) = n + n_2 I$, which is effectively an interaction between photons, mediated by atoms in the Kerr medium. This effect is described by the Hamiltonian

$$H_{spm} = -\chi a^\dagger ab^\dagger b.$$

Where a and b describe two modes propagating through the medium, and for a crystal of length L we obtain the unitary transform $K = e^{i\chi L a^\dagger ab^\dagger b}$.

By combining Kerr media with beamsplitter, a *Controlled-NOT* gate can be constructed in the following manner. For single photon states, we find that

$$K|00\rangle = |00\rangle$$

$$K|01\rangle = |01\rangle$$

$$K|10\rangle = |10\rangle$$

$$K|11\rangle = e^{i\chi L} |11\rangle$$

and let us take $\chi L = \pi$, such that $K|11\rangle = -|11\rangle$.

3.4.3 Drawbacks of OPQC

The single photon representation of a qubit is attractive. Single photons are relatively simple to generate and measure, and in the dual-rail representation, arbitrary single qubit operations are possible. Unfortunately, interacting photons is difficult – the best nonlinear Kerr media available are very weak, and cannot provide a cross phase modulation of π between single photon states. Moreover, there is always some absorption associated with the nonlinearity. Theoretically it can be estimated that in the best such arrangement, approximately 50 photons must be absorbed for each photon which experiences a π cross phase modulation [41].

Historically, optical classical computers were once thought to be promising replacement for electronic machines, but they ultimately failed to live up to expectations when sufficiently nonlinear optical materials were not discovered, and when their speed and parallelism advantage did not sufficiently outweigh their alignment and power disadvantages.

3.4.4 Summary of OPQC Properties

The issues related to the Optical Photon Quantum Computer can be summarized as follows:

- **Qubit Representation:** Location of single photon between two modes, $|01\rangle$ and $|10\rangle$, or polarization.

- **Unitary Evolution:** Arbitrary transforms are constructed from phase shifters (R_z rotations), beamsplitters (R_y rotations), and nonlinear Kerr media, which allow to single photons to cross phase modulate.
- **Initial State Preparation:** Create single photon states (e.g. by attenuating laser light).
- **Readout:** Detect single photons (e.g. using a photomultiplier tube).
- **Drawbacks:** Nonlinear Kerr media with large ratio of cross phase modulation strength to absorption loss are difficult to realize.

3.5 Optical Cavity Quantum Electrodynamics (OCQED)

Cavity *quantum electrodynamics* (QED) is a field of study which accesses an important regime involving coupling of single atoms to only a few optical modes. Experimentally, this is made possible by placing single atoms within optical cavities of very high Q ; because only or two electromagnetic modes exist within the cavity, and each of this has a very high electric field strength, the dipole coupling between the atom and the field is very high. Because of the high Q , photon within the cavity has an opportunity to interact many times with the atom before escaping. Theoretically, this technique present unique opportunity to control and study single quantum systems, opening many opportunities in quantum chaos, quantum feedback control, and quantum computation.

In particular, single-atom cavity QED methods offer a potential solution dilemma with the optimal quantum computer described in the previous section. Single photons can be good carriers of quantum information, but they require some other medium in order to interact with each other. Because they are bulk materials, traditional nonlinear optical Kerr media are unsatisfactory in satisfying this need. However, well isolated single atoms might not necessarily suffer from the same decoherence effects, and more over, they could also provide cross phase modulation between photons. In fact, if the state of single photons could be efficiently transferred to and single atoms, whose interactions could be controlled? This potential scenario is the topic this section.

3.5.1 Physical Apparatus for OCQED

The two main experimental components of a cavity QED system are the electromagnetic cavity and the atom. The basic physics of the cavity modes are described here:

Fabry-Perot cavity:

The main interaction involved in cavity QED is the dipolar interaction $\vec{d} \cdot \vec{E}$ between an electric dipole moment \vec{d} and an electric field \vec{E} . It is difficult to change the size of \vec{d} ; however, $|\vec{E}|$ is experimentally accessible, and one of the most important tools for realizing a very large electric field in a narrow band of frequencies and in a small volume of space, is the Fabry-Perot cavity.

In the approximation that the electric field is monochromatic and occupies a single spatial mode, it can be given a very simple quantum-mechanical description:

$$\vec{E}(r) = i \vec{\epsilon} E_0 [a e^{ikr} - a^\dagger e^{-ikr}].$$

Here, $k = \omega/c$ is the spatial frequency of the light, E_0 is the field strength, $\vec{\epsilon}$ is the polarization, and r is the position at which the field is desired. Note that the Hamiltonian governing the evolution of the field in the cavity is simply

$$H_{field} = \hbar \omega a^\dagger a,$$

and this is consistent with the semi-classic notion that the energy is the volume integral of $|\vec{E}|^2$ in the cavity.

Two-level atoms:

The electronic energy eigenstates of an atom can be very complicated, but for our purposes modeling an atom as having only two states is an excellent approximation. This two-level atom approximation can be valid because we shall be concerned with the interaction with monochromatic

light and the only relevant energy levels are those satisfying two conditions: their energy difference matches the energy of the incident photons, and symmetries do not inhibit the transition. These conditions arise from basic conservation law of energy, angular momentum, and parity. Energy conservation is no more than the condition that

$$\hbar\omega = E_2 - E_1,$$

where E_2 and E_1 are two eigenenergies of the atom. Angular momentum and parity conservation requirements can be illustrated by considering the matrix element of \hat{r} between two orbital wave functions, $\langle l_1, m_1 | \hat{r} | l_2, m_2 \rangle$. Without loss of generality, we can take \hat{r} to be in the $\hat{x} - \hat{y}$ plane, such that it can be expressed in terms of spherical harmonics as $\hat{r} = \sqrt{\frac{3}{8\pi}} [(-r_x + ir_y)Y_{1,+1} + (r_x + ir_y)Y_{1,-1}]$. In this basis, the relevant terms in $\langle l_1, m_1 | \hat{r} | l_2, m_2 \rangle$ are $\int Y_{l_1, m_1}^* Y_{1, m} Y_{l_2, m_2} d\Omega$. The first condition is the conservation of angular momentum, and the second, parity, under the dipole approximation where $\langle l_1, m_1 | \hat{r} | l_2, m_2 \rangle$ becomes relevant. These conditions are selection rules which are important in the two-level atom approximation.

3.5.2 Summary of OCQED Properties

The issues related to the Optical Cavity QED can be summarized as follows:

- **Qubit Representation:** Location of single photon between two modes, $|01\rangle$ and $|10\rangle$, or polarization.
- **Unitary Evolution:** Arbitrary transforms are constructed from phase shifters (R_z rotations), beamsplitters (R_y rotations), and a cavity QED system, comprised of a Fabry-Perot cavity containing a few atoms, to which the optical field is coupled.
- **Initial State Preparation:** Create single photon states (e.g. by attenuating laser light).
- **Readout:** Detect single photons (e.g. using a photomultiplier tube).

- **Drawbacks:** The coupling of two photons is mediated by an atom, and thus it is desirable to increase the atom-field coupling. However, coupling the photon into and out of the cavity becomes difficult, and limits cascading.

3.6 Ion Traps

Electron and nuclear spins provide potentially good representations for qubits. Since the energy difference between different spin states is typically very small compared with other energy scales (such as the kinetic energy of typical atoms at room temperature), the spin states of an atom are usually difficult to observe, and even more difficult to control. However, in carefully crafted environments, exquisite control is possible.

Such circumstances are provided by isolating and trapping small numbers of charged atoms in electromagnetic traps, then cooling the atoms until their kinetic energy is much lower than the spin energy contributions. After doing this, incident monochromatic lights can be tuned to selectively cause transitions which change certain spin states depending on other spin states. This is the essence of how trapped ions can be made to perform quantum computation.

3.6.1 Physical Apparatus for Ion Traps

An ion trap quantum computer has as its main components an electromagnetic trap with lasers and photo-detectors, and ions.

Trap geometry and lasers: The main experimental apparatus, an electromagnetic trap constructed from four cylindrical electrodes, is shown in Figure 3.5.

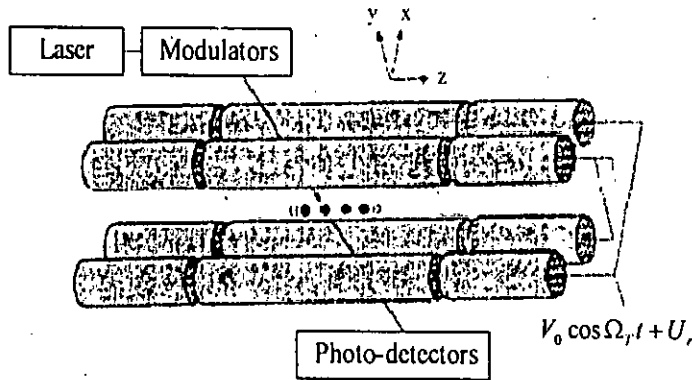


Fig 3.5: Schematic drawing of an ion trap quantum computer.

The end segments of the electrodes are biased at a different voltage U_0 than the middle, so that the ions are axially confined by a static potential $\Phi_{dc} = \kappa U_0 [z^2 - (x^2 + y^2)]/2$ along the \hat{z} axis (κ is a geometrical factor). However, a result known as *Earnshaw's Theorem* states that a charge cannot be confined in three dimensions by static potentials. Thus, to provide confinement, two of the electrodes are grounded, while the other two are driven by a fast oscillating voltage which creates a radiofrequency (RF) potential $\Phi_{rf} = (V_0 \cos \Omega_T t + U_r)(1 + (x^2 - y^2)/R^2)/2$, where R is a geometrical factor. The segments of the electrodes are capacitively coupled such that the RF potential is constant across them. The combination of Φ_{dc} and Φ_{rf} creates, on average (over Ω_T), a harmonic potential of x , y , and z , together with the Coulomb repulsion of the ions, this gives a Hamiltonian governing the motion of the N ions in the trap,

$$H = \sum_{i=1}^N \frac{M}{2} \left(\omega_x^2 x_i^2 + \omega_y^2 y_i^2 + \omega_z^2 z_i^2 + \frac{|\vec{p}_i|^2}{M^2} \right) + \sum_{i=1}^N \sum_{j>i}^N \frac{e^2}{4\pi\epsilon_0 |\vec{r}_i - \vec{r}_j|},$$

where M is the mass of each ion. Typically, $\omega_x, \omega_y \gg \omega_z$ by design, so that the ions all lie along the \hat{z} axis.

In the ion trap, the energy eigenstates represent different vibrational modes of the entire linear chain of ions moving together as a body, with mass NM . These are called the *center of mass* modes. Each $\hbar\omega_z$ quantum of vibrational energy is

called a *phonon*, and can be thought as a particle. For the phonon description to hold, certain criteria must hold. First, the coupling to the environment must be sufficiently small such that thermalization does not randomize the state of the system, and second, the width of the ion oscillation in the trap potential should be small compared to the wavelength of the incident light. This *Lamb-Dicke* criterion is conventionally expressed in terms of the *Lamb-Dicke parameter* $\eta \equiv 2\pi z_0 / \lambda$, where λ is the wavelength, and $z_0 = \sqrt{\hbar / 2NM\omega}$ is the characteristic length scale of the spacing between ions in the trap. The Lamb-Dicke criterion requires that $\eta \ll 1$; this does not strictly have to be met in order for ion traps to be useful for quantum computation, but it is desired to have that $\eta \approx 1$ at least, in order that the individual ions can be resolved by different laser beams, but without making their motional state too difficult to optically excite in order to perform logic operations.

Atomic structure: The internal atomic states relevant to the trapped ion we shall consider result from the combination F of electron spin S and nuclear spin I , giving $F = S + I$. This is formally known as the *addition of angular momenta* theory. The theory not only describes important physics for understanding atomic structure, but also is an interesting mechanism for quantum information. A single photon interacting with an atom can provide or carry away one unit of angular momentum. But there are numerous possible sources of angular momentum in an atom: orbital, electron spin, and nuclear spin. The photon cannot distinguish between different sources, and to describe what happens we must select a basis in which the total angular momentum becomes a uniquely defined property of the state.

Consider, for example, two spin- $1/2$ spins. The computational basis for this two qubit space is $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, but to span the state space we could equally well choose the basis

$$|0,0\rangle_J = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

$$|1,-1\rangle_J = |00\rangle$$

$$|1,0\rangle_J = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

$$|1,1\rangle_J = |11\rangle.$$

These special basis states are eigenstates of total momentum operator, defined by

$$j_x = (X_1 + X_2)/2, j_y = (Y_1 + Y_2)/2, j_z = (Z_1 + Z_2)/2, \text{ and } J^2 = j_x^2 + j_y^2 + j_z^2.$$

The states $|j, m_j\rangle_J$ are eigenstates of J^2 with eigenvalue $j(j+1)$, and simultaneously eigenstates of j_z , with eigenvalue m_j . These states are the natural ones selected by many physical interactions; for example, in a \hat{z} oriented magnetic field the magnetic moment μ in the Hamiltonian $\mu\beta_z$ is proportional to m_j , the component of the total angular momentum in the \hat{z} direction.

3.6.2 Summary of Ion Trap Properties

The issues related to the Ion trap Quantum Computer can be summarized as follows:

- **Qubit Representation:** Hyperfine (nuclear spin) state of an atom, and lowest level vibrational modes (phonons) of trapped ion.
- **Unitary Evolution:** Arbitrary transforms are constructed from application of laser pulses which externally manipulate the atomic state via the Jaynes-Cummings interaction. Qubits interact via a shared phonon state.
- **Initial State Preparation:** Cool the atoms (by trapping and using optical pumping) into their motional ground state, and hyperfine ground state.
- **Readout:** Measure population of hyperfine states.
- **Drawbacks:** Phonons lifetimes are short, and ions are difficult to prepare in their motional ground states.

3.7 Nuclear Magnetic Resonance (NMR)

Direct manipulation and detection of nuclear spin states using radiofrequency electromagnetic waves is a well-developed field known as *Nuclear Magnetic Resonance*. These techniques are widely used in chemistry, for example, to measure properties of liquids, solids, and gases, to determine the structure of molecules, and to image materials and even biological systems. These many applications has lead the technology of NMR to become quite sophisticated, allowing control and observation of tens to hundreds and thousands of nuclei in experiments.

However, two problems arise in using NMR for quantum computation. First, because of the smallness of the nuclear magnetic moment, a large number (more than $\approx 10^8$) of molecules must be present in order to produce a measurable induction signal. The output of an NMR measurement is an average over all the molecule's signals; can the average output of an ensemble of quantum computer be meaningful? Second, NMR is typically applied to physical systems in equilibrium at room temperature, where the spin energy $\hbar\omega$ is much less than $k_B T$. That means the initial state of the spins is nearly completely random.

Solutions to these two problems have made NMR a particularly attractive and insightful method for implementing quantum computation, despite stringent limitations which arise from thermal nature of typical systems. Many lessons can be learnt from NMR: for example, techniques for controlling realistic Hamiltonians to perform arbitrary unitary transforms, methods for characterizing and circumventing decoherence (and systematic errors), and considerations which arise in assembling components in implementing full quantum algorithms on entire systems.

3.7.1 Summary of NMR Properties

The issues related to the NMR Quantum Computer can be summarized as follows:

- **Qubit Representation:** Spin of an atomic nucleus.

- **Unitary Evolution:** Arbitrary transforms are constructed from magnetic field pulses applied to spins in a strong magnetic field. Coupling between spins are provided by chemical bonds between neighboring atoms.
- **Initial State Preparation:** Polarize the spins by placing them in a strong magnetic field, then use 'effective pure state' preparation techniques.
- **Readout:** Measure voltage signal induced by precessing magnetic moment.
- **Drawbacks:** Effective pure state preparation schemes reduce the signal exponentially in the number of qubits, unless the initial polarization is sufficiently high.

3.8 Chapter Summary

The chapter Quantum Computers: Physical Realization can be summarized as follows:

- There are four basic requirements for implementation of quantum computation:
 - i. Representation of qubits,
 - ii. Controllable unitary evolution,
 - iii. Preparation of initial qubit states, and
 - iv. Measurement of final qubit states.
- Single photons can serve as good qubits, using $|01\rangle$ and $|10\rangle$ as logical 0 and 1, but conventional nonlinear optical materials which are sufficiently strong to allow single photons to interact inevitably absorb or scatter the photons.
- Cavity-QED is a technique by which single atoms can be made to interact strongly with single photons. It provides a mechanism for using an atom to mediate interaction between single photons.
- Trapped ions can be cooled to the extent that their electronic and nuclear spin states can be controlled by applying laser pulses. By coupling spin

states through center-of-mass phonons, logic gates between different ions can be performed.

- Nuclear spins are nearly ideal qubits, and single molecules would be nearly ideal quantum computers if their spin states could only be controlled and measured. Nuclear Magnetic Resonance makes this possible using the large ensembles of molecules at room temperature, but at the expense of signal loss due to an inefficient preparation procedure.

Chapter 4

Multi-Output Ternary Logic and Quantum Cascade: A Literature Survey

Different theoretical issues regarding logic synthesis are discussed in this chapter. The Galois Field and Quantum Technology are discussed here. Quantum cascade is a good choice for realizing multi-valued reversible logic. Reversible logic and its utility are also discussed. Finally recent research works in this field are reviewed.

4.1 Reversible Logic

It is implied that the quantum computers are inherently capable of performing reversible computations. And it is also assumed that all the future computers will be reversible. As this thesis deals with the synthesis of quantum logic, we have to follow the rules and postulations of reversible logic. This Section presents the different aspects of reversible logic.

4.1.1 Moore's Law

In 1965 Moore [23] observed an exponential growth in the number of transistors per integrated circuit and predicted that this trend would continue which is well known as "Moore's Law". Through the IC producers' relentless technology advances, Moore's Law, the doubling of transistors every couple of years, has been maintained, and still holds true today. Experts expect that it will continue at least through the end of this decade.

4.1.2 Argument for Alternative Technology

The number of transistors in a processor are getting doubled every couple of years. The power consumption and heat dissipation of the Integrated Circuits are also increasing with the same pace. Another severe problem is that the capacity of semiconductor technology will soon be saturated; under this circumstances VLSI designers all over the world are trying to find alternative technology to design and realize logic circuits.

Definition 4.1: A logic gate is *irreversible* if it is not possible to determine the input combination uniquely by observing the corresponding output produced by the gate. A logic gate is *reversible* if it is possible to determine the input combination uniquely by observing the corresponding output for all the output produced by the gate.

For example, AND gate is irreversible. If the output of an AND gate is 0, then we can not exactly say what was the input combination by observing the output only; it can be any of 00, 01, or 10. Similarly OR, XOR, NAND, etc. gates are also irreversible. Table 4.1 shows the truth table of some common 2-input 1-output irreversible gates.

Input		Output				
A	B	A AND B	A OR B	A XOR B	A NAND B	A NOR B
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0

Table 4.1: Truth table of some common 2-input 1-output irreversible gates.

Figure 4.1 shows the block diagram and truth table of a popular reversible gate named Feynman gate. Note that for every output combination of the gate, there is exactly one distinct input combination in the truth table.

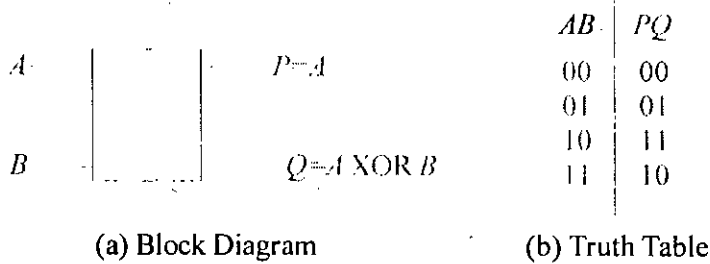


Fig 4.1: A reversible gate

Landauer [49] showed that a computational system built using traditional irreversible logic gates such as AND, OR, etc. leads inevitably to energy dissipation, regardless of the technology to realize the gates. The energy loss due to irreversible gates is negligible for current silicon technologies using adiabatic design. However, it is well known that Moor's Law will stop to function around years 2010 – 2020 and some dramatic changes will therefore have to happen in microelectronics not later than the middle of this century [26]. In that time reversible logic design will be of primary importance.

Bennett [10] showed that for power not to be dissipated in an arbitrary circuit, it is necessary that the circuit be built from reversible gates. In principle, reversible logic gates dissipate arbitrary little amount of heat and the use of reversible operations are likely to become more attractive. It should be noted that Bennett's theorem is only a necessary but not sufficient condition for the motivation of researching reversible logic. Its extreme importance lies in the technological necessity that every future technology will have to use reversible gates in order to reduce power.

4.1.3 Binary Reversible Logic

The issue of reversible logic was first investigated by Landauer in 1961[49]. The reversibility of computation became a matter of concern in the 1970s. There were two related issues, *logical reversibility* and *physical reversibility*, which were intimately connected. *Logical reversibility* refers to the ability to reconstruct the input from the output of a computation, or gate function. For instance, the AND gate is explicitly irreversible, taking two inputs to one output, while the NOT gate

is reversible (it is its own inverse). The connection to physical reversibility is usually made as follows. Since the NAND gate has only one output, one of its inputs has effectively been erased in the process, whose information has been irretrievably lost. The change in entropy that would be associated with the loss of one bit of information is $\ln 2$, which, thermodynamically, corresponds to an energy increase of $kT \ln 2$, where k is Boltzmann's constant and T is the temperature. The heat dissipated during a process is usually taken to be a sign of *physical irreversibility*, that the microscopic physical state of the system cannot be restored exactly as it was before the process took place. This is better explained by the following example, presented in [49].

Example 4.1: Consider a very small special-purpose computer, with three binary elements p , q , and r . A machine cycle replaces p by r , replaces q by r , and replaces r by pq . There are eight possible initial states, and in thermal equilibrium they will occur with equal probability. The initial and final machine states are as follows –

Before Cycle			After Cycle			Final State
p	q	r	p_1	q_1	r_1	
1	1	1	1	1	1	α
1	1	0	0	0	1	β
1	0	1	1	1	0	γ
1	0	0	0	0	0	δ
0	1	1	1	1	0	γ
0	1	0	0	0	0	δ
0	0	1	1	1	0	γ
0	0	0	0	0	0	δ

Table 4.2: Three input–three output devices which maps eight possible states onto only four different states.

There are four distinct final states, namely α , β , γ , and δ with their own frequency of occurrence. State α and β occur with a probability of $\frac{1}{8}$ each, while states γ and δ occur with a probability $\frac{3}{8}$. The initial entropy was

$$\begin{aligned}
 S_i &= k \ln W = -k \sum \rho \ln \rho \\
 &= -k \sum \frac{1}{8} \ln \frac{1}{8} = 3k \ln 2 \\
 &= 2.0794k
 \end{aligned}$$

The final entropy is

$$\begin{aligned}
 S_f &= -k \sum \rho \ln \rho \\
 &= -k \left(\frac{1}{8} \ln \frac{1}{8} + \frac{1}{8} \ln \frac{1}{8} + \frac{3}{8} \ln \frac{3}{8} + \frac{3}{8} \ln \frac{3}{8} \right) \\
 &= 1.2554k
 \end{aligned}$$

The difference $S_i - S_f$ is $0.824k$. The minimum dissipation, if the initial state has no useful information, is therefore $0.824kT$.

There were two related questions, one is whether a computation can be done in a logically reversible fashion (unlike one that uses NAND gates, for example), and the other was whether any heat needs to be dissipated during a computation. Both of these issues were quite academic however, as Feynman [51] pointed out, an actual transistor dissipates close to $10^{10}kT$ of heat, and even the DNA copying mechanism in a human cell dissipates about $100kT$ of heat per bit copied [which is understandable from a consideration of the chemical bonds that need to be broken in the process], both are far from the ideal limit of $kT \ln 2$ for irreversible computing.

That classical computation can be done reversibly with no energy dissipated per computational step was discovered by Bennett [10]. He showed this by constructing a reversible model of the Turing machine [8] and showed that any problem that can be simulated on the original irreversible machine can also be simulated with the same efficiency on the reversible model. The logical reversibility inherent in the reversible model implied that an implementation of such a machine would also be physically reversible. This started the search for physical models for reversible classical computation.

The models for reversible computation are similar to the models of classical computation, except that the number of outputs of the functions will at least be the

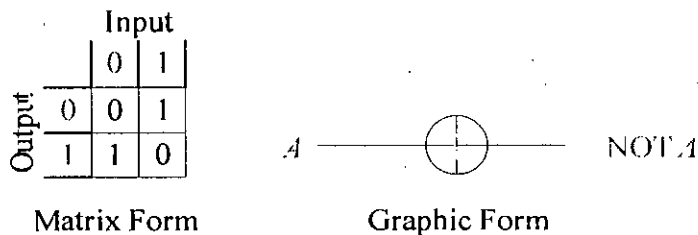
same of the number of inputs, and there must be a bijection³ between the input combinations and the output combinations of the function. Some reversible gates are presented as follows.

Since reversible logic gates are symmetric with respect to the number of inputs and outputs, we can represent them in ways other than the truth table, which emphasizes this symmetry. We are readily familiar to one reversible gate – the NOT gate, whose truth table is –

A	NOT A
0	1
1	0

Table 4.3: Truth table for NOT Gate

We could also write this in the form of a matrix, or as a graphic [Figure 4.2]. The matrix form lists the lines in the truth table in the form 0, 1. The input lines are listed horizontally on the top and the output lines are listed vertically along the side, in the same order.



Matrix Form

Graphic Form

Fig 4.2: Matrix and Graphic representation of NOT Gate

We fill in the matrix with 1's and 0's such that each horizontal or vertical line has exactly one 1, which is to be interpreted as a one-to-one mapping of the input to the output. For example, a 1 in column one, row two in the NOT means that a 0 input gets taken to a 1 output. The graphical representation to the right of the table is a condensed representation of the NOT gate. The horizontal line represents a bit, whose initial variable value, A , is listed on the left and whose final value, NOT A , is listed on the right. The operation of the NOT gate in the middle is

³ That means, there is a unique output combination corresponding to every input combination.

symbolized by the \oplus sign. A two-bit gate closely related to the NOT gate is the two-bit Controlled-NOT (or C-NOT) gate [Figure 4.3], which performs a NOT on the second bit if the first bit is 1, but otherwise has no effect.

XOR=C-NOT		Input			
		00	01	10	11
Output	00	1	0	0	0
	01	0	1	0	0
	10	0	0	0	1
	11	0	0	1	0

Matrix Form

Graphic Form

Fig 4.3: Matrix and Graphic representation of Controlled-NOT Gate

The CNOT is sometimes also called XOR, since it performs an exclusive OR operation on the two input bits and writes the output to the second bit. The graphical representation of this gate has two horizontal lines representing the two variable bits, and the conditionality of the operation is represented by the addition of a vertical line originating from the first bit and terminating with a NOT symbol on the second bit.

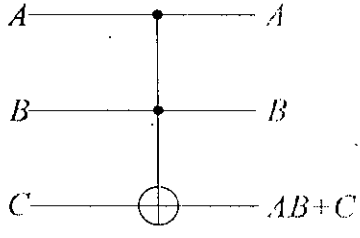
Two-bit gates are not sufficient for universal reversible computation. However, a three-bit gate is sufficient. A universal three-bit gate was identified by Toffoli [54], called the Controlled-Controlled-NOT (or CC-NOT), or simply the (binary) Toffoli gate. It applies a NOT to the third bit if the first two bits are in 11, but otherwise having no effect. The graphical representation of this conditional three-bit gate is given to the right of the table in Figure 4.4, where both A and B are checked to see if they are in 1 - denoted by the two solid circles on these bits - before performing NOT on C.

The Toffoli gate is known to be universal for reversible Boolean logic, the argument for which is based on the fact that the Toffoli gate contains the NAND gate within it. When the third bit is fixed to be 1, the Toffoli gate writes the NAND of the first two bits on the third, that is:

$$A, B, 1 \rightarrow A, B, \overline{AB}$$

CC-NOT		Input							
		000	001	010	011	100	101	110	111
Output	000	1	0	0	0	0	0	0	0
	001	0	1	0	0	0	0	0	0
	010	0	0	1	0	0	0	0	0
	011	0	0	0	1	0	0	0	0
	100	0	0	0	0	1	0	0	0
	101	0	0	0	0	0	1	0	0
	110	0	0	0	0	0	0	0	1
	111	0	0	0	0	0	0	1	0

Matrix Form



Graphic Form

Fig 4.4: Matrix and Graphic representation of CC-NOT Gate

4.1.4 Ternary Reversible Logic

Ternary quantum circuits have recently been introduced to help reduce the size of multi-valued logic for multi-level quantum computing systems. While most of the results are for binary quantum computers, the multi-valued quantum logic synthesis is very new research area. Unfortunately, previous synthesis methods produce circuits that are unnecessarily complex. One promising alternative for reducing the circuit size is to use gates that are ternary counterparts of the classical binary Feynman gates and new 2-qudit ternary controlled gates (qudit is a multiple-valued counterpart of binary quantum bit or qubit, for ternary logic it is known as qutrit).

4.1.5 Some Ternary Reversible Gates

Figure 4.5(a) shows a ternary Feynman Gate. Here A is the controlling input and B is the controlled input. P is equal to the input A and Q is GF3* sum of A and B. (note that GF3 sum is the same as modulo 3 sum). If B = 0, then Q = A, and the gate acts as a copying gate. The ternary 2*2 Feynman Gate is practically realizable [37].

* Ternary Galois Field [see Section 4.3.2.2]

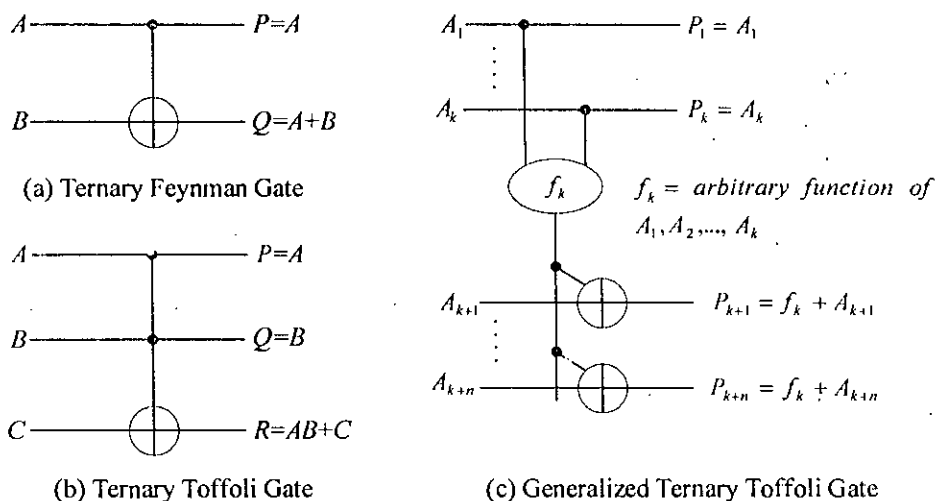


Fig 4.5: Some Ternary Reversible Gates

100801

Figure 4.5(b) shows a 3*3 Toffoli Gate. Design of Galois Field Sum of Products (GFSOP) arrays and factorized arrays is based on these gates. These arrays are the multiple-valued counterpart of well-known binary Exclusive-OR Sum of Products (ESOP) and factorized ESOP cascades. Here A and B are the controlling input and C is the controlled input. A generalized Ternary Toffoli Gate is proposed by [35] shown in Figure 4.5(c). There are k controlling inputs and n controlled inputs.

There are six ternary shift operations while binary logic has only two – no shift and NOT. Six 1*1 ternary shift gates are shown in Figure 4.6. These gates are realizable using ternary quantum Feynman primitive [35]. Two cascaded shift gates can be replaced by a single equivalent shift gate.

Gate Name	Gate Symbol with operation*	Gate Number
Buffer	$x \rightarrow \triangleleft x$	0
Single-Shift	$x \rightarrow \triangleleft / x' = x + 1$	1
Dual-Shift	$x \rightarrow \triangleleft // x'' = x + 2$	2
Self-Shift	$x \rightarrow \triangleleft /// x''' = 2x$	3
Self-Single-Shift	$x \rightarrow \triangleleft \# X'' = 2x + 1$	4
Self-Dual-Shift	$x \rightarrow \triangleleft \wedge x^{\wedge} = 2x + 2$	5

Fig 4.6: Ternary Shift operations, gate symbols, and their numbers

Realization of multiple-valued (ternary) logic function in quantum circuit using complex gates like Toffoli is not feasible. This is because in general these gates are having m inputs and m outputs where $m > 2$. Therefore it would be a better idea if the quantum circuit is constructed using 2×2 gates (primitive gates) only.

Figure 4.7 shows the realization of an arbitrary ternary function using Toffoli gates. Description of the synthesis process is beyond the scope of this thesis. (see [37])

The success in the true quantum realization of some ternary permutation gates now allows us to physically build ternary quantum computers using these gates. One very promising 2×2 primitive gate is generalized ternary gates (GTG) is shown in Figure 4.8. It was first introduced by Perkowski et. al. [43]. De-Vos Gates and Ternary Feynman Gates are special cases of this gate. They claimed that GTG can be realized using Quantum Technology such as ion traps [43]. Very recently some works are being done on synthesizing reversible ternary circuits using GTG (see [34], [36], [38], [44]).

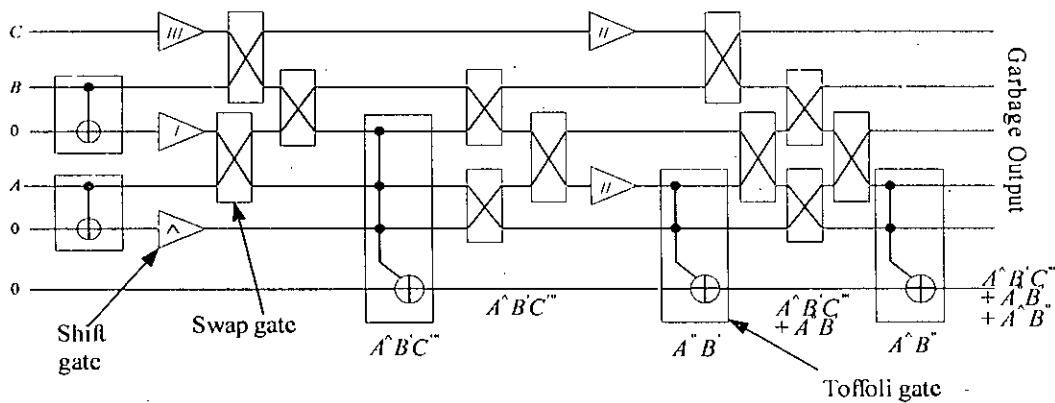


Fig 4.7: Quantum Circuit using Toffoli gates to realize the function,
 $F(A,B,C) = [0,1,2,1,0,2,2,2,2,2,1,0,0,1,2,1,1,1,1,1,1,2,2,2,0,0,0]^T$
 $= A^B C'' + A'' B' + A^B$

4.2 Galois Field and Quantum Technology

In this section we briefly discuss the theoretical background of Galois Field (GF) and Quantum technology. The mathematical foundation of reversible logic

synthesis is based on the theory of GF and the practical implementation of the reversible circuit can be done efficiently with the help of Quantum Technology.

4.2.1 Quantum Computation

The earliest formalism of quantum computation (exploiting the full power of quantum computers) was introduced by Deutsch [16] in 1985, when he defined a quantum physical analogue of a probabilistic Turing machine, but the first surprising powerful result came almost ten years later. In 1994 Peter W. Shor demonstrated how the quantum computation can be used for factoring integers into prime factors probabilistically in polynomial time [48]. This invention is naturally interesting theoretically, but also practically if a quantum computer could be really constructed, since the securities of the RSA cryptosystem and many protocols is based on the assumed non-tractability of the factoring problem.

The research on quantum computation naturally can be divided into physical and mathematical part, although the border between these parts is not clear and stable. The physical research concentrates more on the possibility of the implementation of quantum computers and quantum cryptography, while the mathematical part will be interested in the classification of quantum complexity classes and the relations between classical ones.

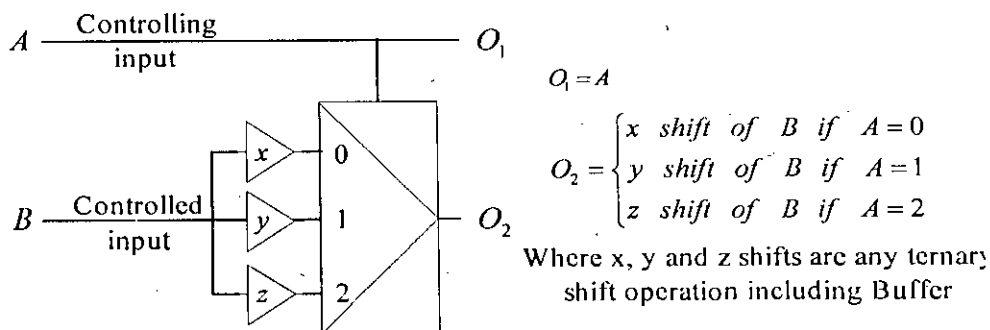


Fig 4.8: Generalized Ternary Gates (GTG)

4.2.2 Ternary Quantum Computing

In multi-valued (MV) Quantum Computing (QC), the unit of memory (information) is qudit. MV quantum logic operations manipulate qudits, which are

microscopic entities such as a photon's polarizations or an elementary particle's spins. Ternary logic values of 0, 1, and 2 are represented by a set of distinguishable different states of a qutrit. After encoding these distinguishable quantities into multiple-valued constants, qutrit states are represented by the notations $|0\rangle$, $|1\rangle$, and $|2\rangle$.

Qudits exist in a linear superposition of states. In ternary logic, the notation for the superposition is $\alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle$. These intermediate states cannot be distinguished, rather a measurement will yield that the qutrit is in one of the basis states, $|0\rangle$, $|1\rangle$, or $|2\rangle$. The probability that a measurement of a qutrit yields state $|0\rangle$ is $|\alpha|^2$, state $|1\rangle$ is $|\beta|^2$, and state $|2\rangle$ is $|\gamma|^2$. Sum of these probabilities is 1. The absolute values are required since, in general, α , β and γ are complex quantities.

Pairs of qutrits are capable of representing nine distinct states, $|00\rangle$, $|01\rangle$, $|02\rangle$, $|10\rangle$, $|11\rangle$, $|12\rangle$, $|20\rangle$, $|21\rangle$, and $|22\rangle$, as well as all possible superposition of the states. This property is known as "entanglement", and may be mathematically described using the Kronecker product (tensor product) operation \otimes , defined as –

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} x & y \\ z & v \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} x & y \\ z & v \end{bmatrix} & b \begin{bmatrix} x & y \\ z & v \end{bmatrix} \\ c \begin{bmatrix} x & y \\ z & v \end{bmatrix} & d \begin{bmatrix} x & y \\ z & v \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ax & ay & bx & by \\ az & av & bz & bv \\ cx & cy & dx & dy \\ cz & cv & dz & dv \end{bmatrix}$$

As an example, consider two qutrits with $\psi_1 = \alpha_1|0\rangle + \beta_1|1\rangle + \gamma_1|2\rangle$ and $\psi_2 = \alpha_2|0\rangle + \beta_2|1\rangle + \gamma_2|2\rangle$. When the two qutrits are considered to represent a state, that state ψ_{12} is the superposition of all possible combinations of the original qutrits, where $\psi_{12} = \psi_1 \otimes \psi_2 = \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \dots + \gamma_1\gamma_2|22\rangle$.

Superposition property allows qubit states to grow much faster in dimension than classical bits, and qudits states grow much faster than qubits states [7]. In a classical system, n bits represent 2^n distinct states, whereas n qutrits correspond

to a superposition of 3^n states. In the above formula some coefficient can be equal to zero, so there exists a constraint bounding the possible states in which the system can exist. As observed in [7] – “Allowing d to be arbitrary enables a tradeoff between the number of qudits making up the quantum computer and the number of levels in each qudit”. These all contribute to difficulty in understanding the concepts of quantum computing and creating efficient analysis, simulation, verification and synthesis algorithms for QC. Generally, however, we believe that much can be learnt from the history of Electronic Computer Aided Design as well as from MV logic theory and design, and the lessons learnt should be used to design efficient Soft Computing tools for MV quantum computing.

4.2.3 Quantum Circuit

In terms of logic operations, anything that changes a vector of qudit states can be considered as an operator. These phenomena can be modeled using the analogy of a “quantum circuit”. In a quantum circuit, wires do not carry ternary constants but correspond to 3-tuples of complex values, α , β , and γ . Quantum logic gates of the circuit map the complex values on their inputs to complex values on their outputs. Operations of quantum gates are described by matrix operations. Any quantum circuit is a composition of parallel and serial connections of blocks, from small to large. Serial connection of blocks corresponds to multiplication of their (unitary) matrices. Parallel connection corresponds to Kronecker multiplication of their matrices. So, theoretically, the analysis, simulation and verification are easy and can be based on matrix methods. Practically these are tough because the dimensions of the matrices grow exponentially.

4.2.4 Galois Field

A Galois field is a finite field with p^n elements where p is a prime integer. The set of nonzero elements of the field is a cyclic group under multiplication. Here we are showing the elements and addition and multiplication operations of the first three Galois Fields in the subsequent sections.

4.2.4.1 GF(2)

GF(2) consists of the elements 0 and 1 and is the smallest finite field. Its addition and multiplication tables are as follows:

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

Table 4.4: Addition and Multiplication in GF(2)**4.2.4.2 GF(3)**

GF(3) consists of the elements 0, 1, and 2. Its addition and multiplication tables are as follows (Table 4.5):

+	0	1	2
0	0	0	2
1	1	2	0
2	2	0	1

.	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Table 4.5: Addition and Multiplication in GF(3)**4.2.4.3 GF(4)**

The definition of GF(4) is apparently a bit different from the previous two GFs. Since 4 is a non-prime number, GF(4) is actually considered to be GF(2²). Its elements are denoted here as (0, 1, A, B). Here are the addition and multiplication tables for GF(4):

+	0	1	A	B
0	0	1	A	B
1	1	0	B	A
A	A	B	0	1
B	B	A	1	0

.	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

Table 4.6: Addition and Multiplication in GF(4)**4.2.5 Ternary Galois Field Logic**

In Galois Field Sum of Products (GFSOP) the product terms are GF products and the sums are GF sum operations. In this thesis we concentrate only on ternary GFSOPs. Ternary Galois Field (GF3) consists of the set of elements $T = \{0, 1, 2\}$

and two basic binary operations – addition (denoted by $+$) and multiplication (denoted by \cdot or absence of any operator) as defined in Table 4.5. GF3 addition and multiplication are closed, i. e., for $A, B \in T$, $A+B \in T$ and $AB \in T$. GF3 addition and multiplication are also commutative and associative, i. e., $A+B = B+A$ and $AB = BA$ (commutative), and $A+(B+C) = (A+B)+C = A+B+C$ and $A(BC) = (AB)C = ABC$ (associative). GF3 multiplication is distributive over addition, i. e., $A(B+C) = AB+AC$.

There are six reversible ternary unary operations corresponding to six possible permutations of 0, 1, and 2. These unary operations are called reversible ternary shift operations. We already have mentioned the names of these six shift operations, their operator symbols and equations, and gate symbols in Figure 4.6. Among these six shift operations only single-shift, dual-shift (both are also called Post cycles [32] and cyclic negations in [20]), and self-dual-shift (also called inverse [21]) were previously used in the context of quantum computation. All these six shift operators can be built as reversible ternary gates. Khan et. al. [34] proposed quantum realization of the ternary shift gates (except the buffer, which is quantum wire; see Fig 4.9). These realizations require two to three quantum wires.

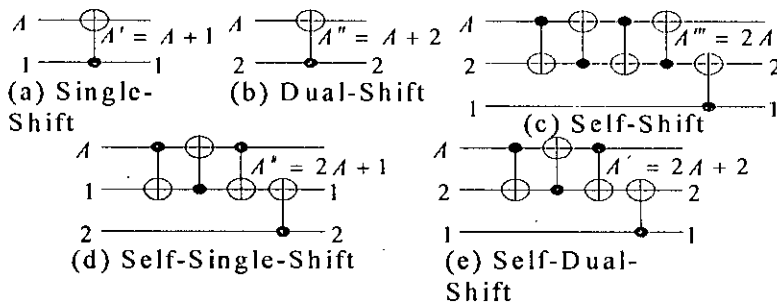


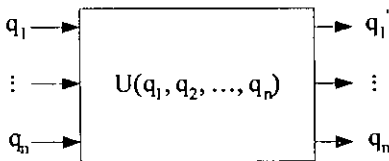
Fig 4.9: Quantum realization of ternary shift gates

The GF3 basic literal of a variable A is an element of the set $\{1, 2, A, A', A'', A''', A^{\#}, A^{\wedge}, A^2\}$. It should be noted that all ternary literals, except A^2 , are reversible. A reversible ternary literal multiplied by 2 yields another reversible ternary literal as follows: $2 \cdot 1 = 2$, $2 \cdot 2 = 1$, $2A = A'''$, $2A' = A^{\wedge}$, $2A'' = A^{\#}$, $2A''' = A$, $2A^{\#} = A''$, and $2A^{\wedge} = A'$. Again, a ternary literal may have

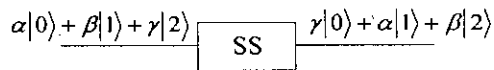
a power of only 2, since $A^3 = A$ (can be verified from Table 1), $A^4 = A^3 A = A^2$, and so on. A product term is a GF3 product of some literals. For example, AB^n is a product term. Ternary GFSOP is GF3 sum of some product terms. For example, $2 + AB^n + B^2C' + A'C^n$ is a ternary GFSOP.

4.2.6 Quantum Cascade (QC)

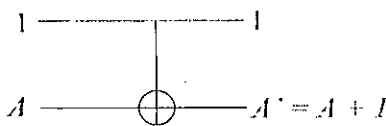
Quantum circuits are quite different than the classical logic circuits. A quantum computer processes information in form of some unitary operation carries the input qutrits to the output qutrits. This can be represented in the form of a block diagram, similar to 4.10(a). For example, the case of ternary single-shift operation is shown in figure 4.10(b) and figure 4.10(c). All the quantum gates perform in the same way. In order to realize any m -input n -output logic function, a number of $k*k$ gates ($k \geq \max \{m, n\}$) are appended one after another following a particular order. The output qutrits of the first gate are the input to the second gate, and so on. The final output is obtained at the output qutrits of the last gate. At this output level, there are k qutrits, among them n qutrits are realizing the function and the remaining $k - n$ qutrits are known as garbage output. The whole thing is known as Quantum Cascade. Figure 4.11 shows a quantum cascade realizing an arbitrary 2-input 2-output ternary function ($k = 6$).



(a) Block diagram of Quantum Unitary Transformation



$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \gamma \\ \alpha \\ \beta \end{bmatrix}$$



(c) Quantum realization of Single-Shift

(b) Block diagram and Matrix representation of the Unitary Transformation performing Single-Shift

Fig 4.10: Quantum Unitary Transformation

4.2.7 Realization of MVL Using Quantum Cascade

Multiple-valued ternary logic can be realized using Quantum Cascades. Very few works have been done so far in constructing the Quantum cascades that are capable of realizing ternary functions. The research in this field is still in its very primitive age. In most of the cases the researchers emphasized on just successfully realizing the functions, rather than making it optimal as well. In figure 4.11 the high number of garbage output gives an idea about the level of optimality achieved so far in this field. In the following article we describe some of the methods of realizing MVL using QC.

4.2.8 Some Existing Methods of Realizing MVL Using QC

Khan et al. [37] first proposed a complete method of realizing ternary logic using quantum cascade in 2004. In this literature various basic and composite ternary literals are proposed for defining TGFSOP expression. They also proposed 16 Ternary Galois Field Expansions (TGFE), like Shannon's Expansions, using these literals and three new types of Ternary Galois Field Decision Diagrams (TGFDD) using the proposed expansions. A heuristic for creating optimal Kronecker TGFDD and methods for flattening the TGFDDs for determining near-minimum TGFSOP expressions is also proposed there. Finally, they proposed a method of synthesizing multi-output TGFSOP using cascade of ternary shift gates, swap gate, and generalized Toffoli gate. They have used some sorts of local optimization technique by selecting the TGFE that generates lowest number of non-zero constants in the output vector in each level. Figure 4.11 shows the realization of one ternary multi-output function using their method. Two major drawback of the method are as follows –

- They have used generalized Toffoli gates that are not primitive ternary gates; therefore can not easily be implemented using quantum technology. However, Khan [38] shows a quantum realization of ternary Toffoli gate using primitive ternary gates.
- They have used local optimization in each level; it may easily fall into local optima.

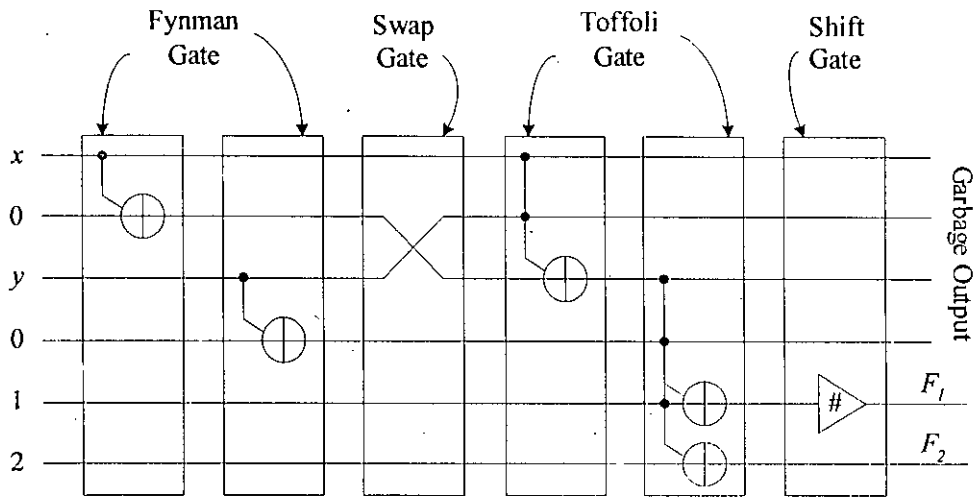


Fig 4.11: Quantum Cascade realizing an arbitrary 2-input 2-output ternary function using different types of ternary gates.

The solution of the former problem is an open problem for the quantum physicists and mathematicians while the later one can be handled by developing better heuristics.

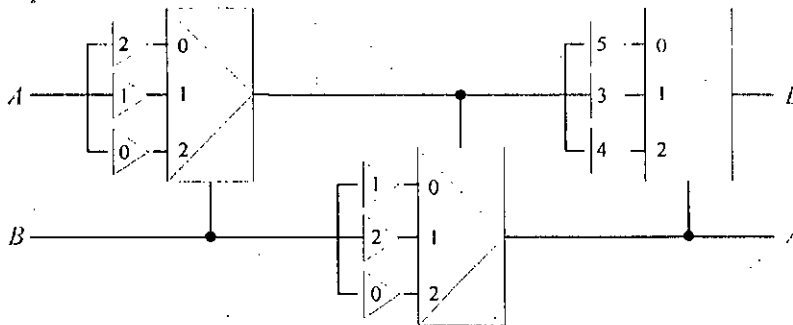


Fig 4.12: Realization of ternary Swap gate using GTG gates

Khan and Perkowski [36] proposed an EA based synthesis process of ternary logic in 2004. They have used the cascades of GTG gates to form the quantum circuit. They proposed a method for synthesizing both completely and incompletely specified ternary functions. EA to synthesize ternary functions is first proposed by them and that is why the result could not be compared to any other EA based method. But their method has produced better results than other non EA based synthesis methods. For example, the previous best design of ternary swap gate had 4 Feynman gates and one 1-qubit permutative gate; while their proposed design

requires only 3 GTG gates and it has the same symmetry as the well known design of Swap from Feynman gates in binary. Figure 4.12 shows the realization of ternary Swap gate using GTG gates.

Denler et al [44] presented a new type of realizable quantum cascade. Then they have proposed algorithms to synthesize arbitrary single-output ternary functions using those reversible cascades. The cascades use “Generalized Multi-Valued Gates” (GMVG) introduced by them, which extend the concept of GTG gates. While there are 216 GTG gates, a total of 12 ternary gates of this type (GMVG) are sufficient to realize any ternary function. Such gates are also claimed to be realizable in quantum ion trap devices. They have implemented the algorithm only for ternary logic, but its generalization to arbitrary radix is straightforward and might give better practical advantages if quantum gates with higher radices were realizable. Figure 4.13 shows the GMVG.

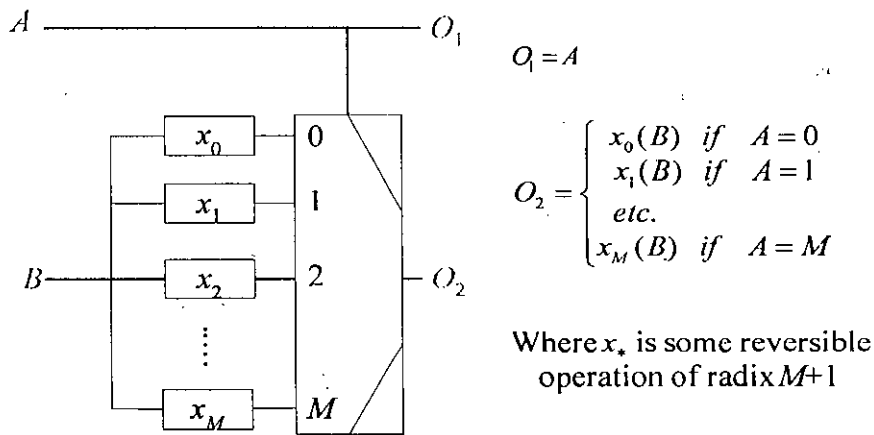


Fig 4.13: Generalized Multi-valued Gate of radix $M+1$

4.3 Evolutionary Algorithm

In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was to evolve a population of candidate solutions for a given problem, using operators inspired by natural genetic and natural selection.

EAs are computer programs that attempt to solve complex problems by mimicking the processes of Darwinian evolution [11]. In an EA a number of artificial creatures search over the space of the problem. They compete continually with each other to discover optimal areas of the search space. It is hoped that over time the most successful of these creatures will evolve to discover the optimal solution.

The artificial creatures in EAs, known as individuals, are typically represented by fixed length strings or vectors. Each individual encodes a single possible solution to the problem under consideration. EAs manipulate pools or populations of individuals. The EA is started with an initial population of size m comprising random individuals (that is, each value in every string is set using a random number generator). Every individual is then assigned a fitness value. To generate a fitness score the individual is decoded to produce a possible solution to the problem. The value of this solution is then calculated using the fitness function. Population members with high fitness scores therefore represent better solutions to the problem than individuals with lower fitness scores. Following this initial phase the main iterative cycle of the algorithm begins. Using mutation (perturbation) and recombination operators, the m individuals in the current population produce n children. The n children are assigned fitness scores. A new population of m individuals is then formed from the m individuals in the current population and the n children. This new population becomes the current population and the iterative cycle is repeated. At some point in the cycle evolutionary pressure is applied. That is, the Darwinian strategy of the survival of the fittest is employed and individuals compete against each other. This is achieved by selection based on fitness scores, with 'better fit' individuals more likely to be selected. The selection is applied either when choosing individuals to parent children or when choosing individuals to form a new population.

There have been three main independent implementation instances of EAs: GAs, developed by Holland [27] and thoroughly reviewed by Goldberg [18]; evolution strategies (ESs), developed in Germany by Rechenberg [30] and Schwefel [25]; and evolutionary programming (EP), originally developed by L. J. Fogel et al.

[31] and subsequently refined by D. B. Fogel. [15]. Each of these three algorithms has been proved capable of yielding approximately optimal solutions given complex, multimodal, non-differential, and discontinuous search spaces. Success has also been achieved for noisy and time-dependent landscapes. A simple description of each technique is given here. Finally, it is worth noting that the implementer is free to modify these algorithms.

In the subsequent sections we describe different EAs briefly with their respective general outline.

4.3.1 Genetic Algorithms

Figure 4.14 shows the canonical GA as developed by Holland [27]. The canonical GA encodes the problem within binary string individuals. Evolutionary pressure is applied in Step 3, where the stochastic technique of roulette wheel parent selection is used to pick parents for the new population.

1. A population of m random individuals is initialized.
2. Fitness scores are assigned to each individual.
3. Using roulette wheel parent selection $m/2$ pairs of parents are chosen from the current population to form a new population.
4. With probability P_c , children are formed by performing crossover on the $m/2$ pairs of parents. The children replace the parents in the new population.
5. With probability P_m , mutation is performed on the new population.
6. The new population becomes the current population.
7. If the termination conditions are satisfied exit, otherwise go to step 3.

Fig 4.14: A Canonical GA

4.3.2 Evolution Strategies

Figure 4.15 shows the ES as developed by Rechenberg [30] and Schwefel [25]. Historically ESs were designed for parameter optimization problems.

1. A current population of m individuals is randomly initialized.
2. Fitness scores are assigned to each of the m individuals.
3. n new offspring are generated by recombination from the current population.
4. The n new offspring are mutated.
5. Fitness scores are assigned to the n new offspring.
6. A new population of m individuals is selected.
7. The new population becomes the current population.
8. If the termination conditions are satisfied exit, otherwise go to step 3.

Fig 4.15: A simple ES

The encoding used in an individual is therefore a list of real numbers: these are called the object variables of the problem. Additionally, each individual contains a number of strategy parameters, these being the variances and co variances of the object variables (the co variances are optional, but when used are normally defined using the rotation angles of the covariance matrix). The strategy parameters are used to control the behavior of the mutation operator and are not required when decoding an individual.

4.3.3 Evolutionary Programming

Figure 4.16 illustrates the form of an EP scheme. EP was originally developed by L. J. Fogel et al. [31] for the evolution of finite state machines using a limited symbolic alphabet encoding. Subsequently D. B. Fogel extended the EP to encode real numbers, thus providing a tool for variable optimization [15].

Individuals in the EP comprise a string of real numbers, as in ESs. EP differs from GAs and ESs in that there is no recombination operator. Evolution is wholly dependent on the mutation operator, which uses a Gaussian probability distribution to perturb each variable. The standard deviations correspond to the

square root of a linear transform of the parents' fitness score (the user is required to parameterize this transform).

1. A current population of m individuals is randomly initialized.
2. Fitness scores are assigned to each of the m individuals.
3. The mutation operator is applied to each of the m individuals in the current population to produce m offspring.
4. Fitness scores are assigned to the m offspring.
5. A new population of size m is created from the m parents and the m offspring using tournament selection.
6. If the termination conditions are satisfied exit, otherwise go to step 3.

Fig 4.16: A simple EP scheme

To overcome parameterization problems associated with the linear transform Fogel developed meta-evolutionary programming (meta-EP) [14]. In meta-EP individuals encode both object variables and variances (one variance for each object variable). As in ESs the variances are self-adapted and used to control the Gaussian mutation operator.

4.4 Summary

We have discussed Reversible Logic, Galois Field and Quantum Computation, and Evolutionary Algorithm in this chapter. Reversible logic provides a way to construct circuits that will, theoretically, dissipate no heat – thus less power will be consumed. Binary reversible logic deals with two states namely 0 and 1, while ternary reversible logic has three states – 0, 1, and 2. The mathematical foundation of reversible logic lies on Galois Field. Specifically for ternary reversible logic, GF3 arithmetic is used. Quantum Computation can be achieved by the circuits constructed using Cascades of Quantum Gates (Quantum Cascades). Finally we

discussed the basic concepts of Evolutionary Algorithms (EA) as we have used EA for synthesizing the Quantum Cascade.

Chapter 5

EA Based Synthesis of Multi-Output Ternary Function Using Quantum Cascades

5.1 Introduction

In this chapter we first propose a family of ternary 2*2 quantum primitive gates. Then an EA based synthesis process of Multi-output Ternary function is presented. Here we propose a practical approach to synthesize directly with the new gates, but the problem is there is no direct method to construct a quantum cascade using those gates. Hence we have to go for Evolutionary Algorithms (EA). Use of EA will allow us to find an appropriate combination of the gates that realize (perhaps optimally) a multiple-valued ternary logic function.

5.2 The New 2*2 Quantum Ternary Gates

A family of 2*2 Quantum ternary gates is proposed here. These gates are the extensions of De Vos gates proposed in [6]. Muthukrishnan [7] showed that these types of gates are practically realizable in quantum ion trap. The general form of our proposed gates is shown in Figure 5.1.

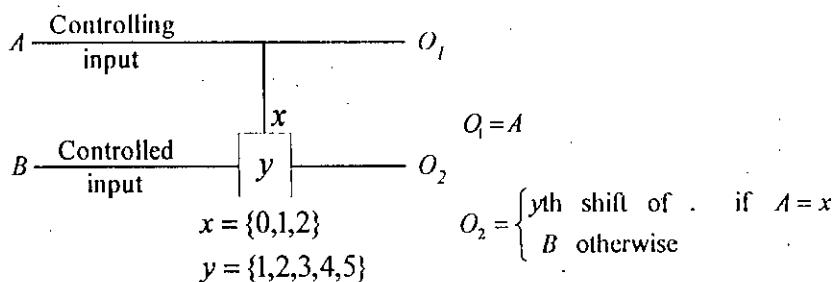


Fig 5.1: General form of the proposed Gates

The gate has two parameters – x and y . There are two input lines called controlling input and the controlled input. The controlling input passes to the output without any modification, while the controlled input is modified only if the controlling input line carries the signal equal to x ; in that case the corresponding output is the y th shift of the controlled input signal. Let us use the 4-tuple notation $\langle A, B, x, y \rangle$ to denote this new gate, where A is the controlling input, B is the controlled input, and x and y are the parameters of the gate. The parameter x can take any value from $\{0, 1, 2\}$ and y can take a value from $\{1, 2, 3, 4, 5\}$. There are fifteen different combinations of the parameters, hence 15 different gates. Among those, the gates with $(x=2, y=1)$ and $(x=2, y=2)$ are proposed by De Vos [6]. We are proposing two more values for x and three more values for y .

5.3 Realization of Multi-output Ternary Functions using the New Gates

In realization of multi-output ternary functions using our proposed gates we assumed the following:

- A Gate can be controlled either from top or from bottom,
- A limited vertical wire crossing for the controlling signal of the gates are allowed,
- Constant input signals 0, 1, and 2 are added as needed,
- Output may be realized along any primary input line or any constant input line, and
- Each of the gates forms a column where the remaining lines represent quantum wires. The columns are cascaded to realize the circuit.

Figure 5.2 shows the realization of ternary half adder circuit using the new gates. To verify the circuit, the intermediate states of the quantum wires are also shown. Each of the gates forms a quantum column and cascades of such columns construct the whole circuit. Table 5.1 shows the truth table of the Ternary Half Adder. The output functions are normally shown as a transpose form as shown in

Figure 5.2; i.e. $sum = [0,1,2,1,2,0,2,0,1]^T$ and $carry = [0,0,0,0,0,1,0,1,1]^T$ in Figure 5.2 are actually the transpose of columns Sum and Carry in Table 5.1 respectively.

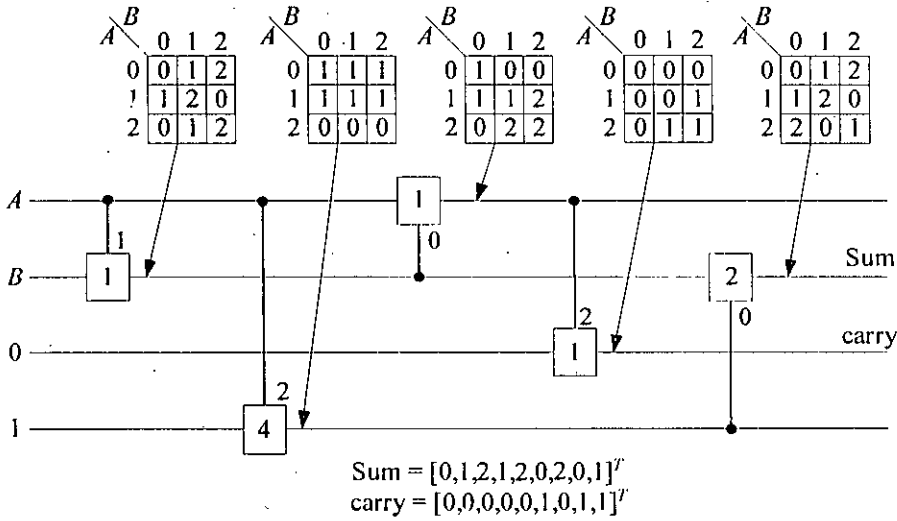


Fig 5.2: Realization of Ternary Half Adder using the new gates

<i>AB</i>	Sum	Carry
00	0	0
01	1	0
02	2	0
10	1	0
11	2	0
12	0	1
20	2	0
21	0	1
22	1	1

Table 5.1: Truth table of Ternary Half Adder function

5.4 GTG Verses the New Gates

The GTG gates are proposed by Perkowski et. Al. [43]. Since then it has become very popular among the researchers in this field. They claimed that the GTG gates can directly be constructed in linear ion trap. They referred [7] as the basis of their claim. In [7] it is shown that any *d*-valued primitive quantum gate can be constructed in linear ion trap. These gates are generally called “conditional gates” as they are capable of performing any unitary transformation on the controlled

input if all the controlling inputs are at state $|d-1\rangle$. In other words, a ternary ($d=3$) conditional quantum gate will perform any unitary transformation if all its controlling inputs are at state $|2\rangle$. De Vos [6] also stated the same.

Now the GTG gates perform different transformations depending on the different states of the controlling inputs including $|0\rangle$ and $|1\rangle$. At this moment we have not found any literature proving that the ternary transformations can be performed when the controlling states are $|0\rangle$ or $|1\rangle$. Therefore, we are not sure whether the GTG gates are directly realizable in quantum technology or not. However, later in this section, we are showing how to achieve those operations using De Vos gates.

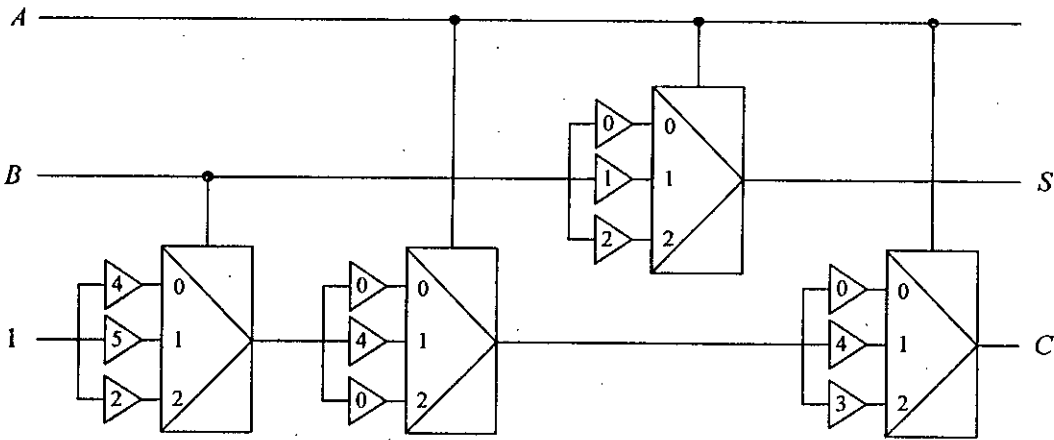


Fig 5.3: Ternary Half Adder realization using GTG gates by [36]

In case of the new gates $\langle A, B, x, y \rangle$, we know that $\langle A, B, 2, 1 \rangle$ and $\langle A, B, 2, 2 \rangle$ are nothing but De Vos gates. Now, $\langle A, B, 2, 3 \rangle$, $\langle A, B, 2, 4 \rangle$, and $\langle A, B, 2, 5 \rangle$ are direct extensions of De Vos gates and the transformations numbered 3, 4, and 5 are also unitary transformations. So according to [7] these gates can directly be constructed using quantum technology.

About the gates $\langle A, B, 0, y \rangle$ and $\langle A, B, 1, y \rangle$, we do not have any proof, neither from the quantum physicists nor from the mathematicians, that these type of gates are directly realizable in quantum technology. However, each of those gates can be constructed using three De Vos ($\langle A, B, 2, y \rangle$) gates. Figure 5.4 shows that.

The same idea can be applied for GTG gates. The third conditional transformation can be implemented directly using one $\langle A, B, 2, y \rangle$ gate. For the first and second conditional transformation, it will require six ($=3+3$) $\langle A, B, 2, y \rangle$ gates. So we can say roughly that, every GTG gate can be constructed using seven $\langle A, B, 2, y \rangle$ gates.

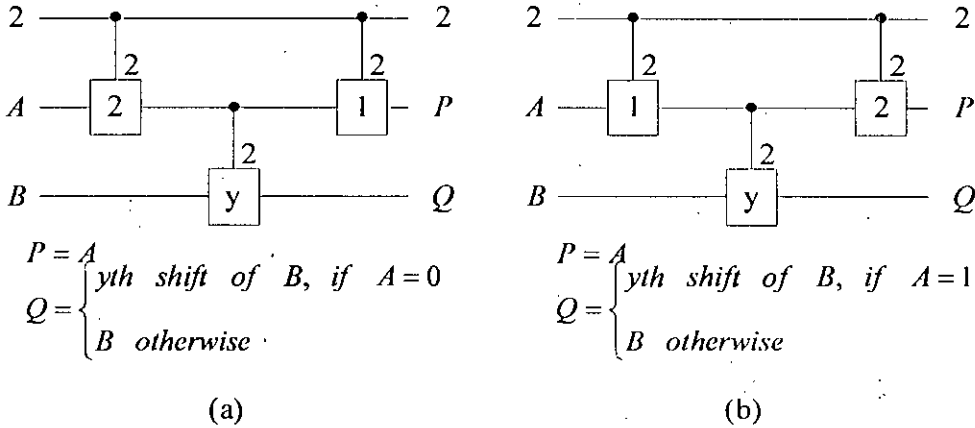


Fig 5.4: Realization of $\langle A, B, 0, y \rangle$ and $\langle A, B, 1, y \rangle$ gates using De Vos ($\langle A, B, 2, 1 \rangle$ and $\langle A, B, 2, 2 \rangle$) gates.

As we are constructing quantum circuit using $\langle A, B, x, y \rangle$ gates, we can calculate the cost of the circuits in terms of number of $\langle A, B, 2, y \rangle$ gates. Assuming that each $\langle A, B, 0, y \rangle$ and $\langle A, B, 1, y \rangle$ gate requires three $\langle A, B, 2, y \rangle$ gates. Now we can compare the cost of the quantum circuit constructed using GTG gates with quantum circuit constructed using $\langle A, B, x, y \rangle$ gates. For example the ternary half adder circuit in [36] has four GTG gates (see Figure 5.3). The total number of $\langle A, B, 2, y \rangle$ gates is –

$$(3+3+1)+(0+3+0)+(0+3+1)+(0+3+1)=18$$

Note that the 0th shift is the buffer and implemented by quantum wire only; no gate is required for that.

On the other hand the cost of the ternary half adder circuit in Figure 5.2 is –

$$(3+1+3+1+3)=11$$

It clearly indicates that the $\langle A, B, x, y \rangle$ gates are better candidate for constructing quantum circuits than the GTG gates.

5.5 Proposed Evolutionary Algorithm

We have used EA to synthesis Multi-output Ternary function because EAs are very popular Soft Computing approach for solving problems with no identified structure and high level of noise. EAs are popular because –

- A huge problem space can be searched.
- The size of the search space can be moderated by parameters.
- A variety of new solutions can be produced.
- With long enough time, a solution might be obtained that is close to the optimal one.

These advantages made us inspired to use EA since the problem structure of the cascades of the new gates is still unidentified and the search space is exponentially large. The following example explains the complexity of this type of problems. For example, Consider the Ternary Half Adder in Figure 5.2. It shows an n -input m -output function ($n=2$, $m=2$) is realized using quantum cascades of the new gates. Five columns of the gates are required here. But for any multi-output function, there is no method known to find out the required number columns, order of the columns, parameter values of the new gates, or even the controlling and controlled input prior to the synthesis. There might be any number of columns, assuming for a particular case that there are L ($=5$ in Fig. 5.2) columns and K ($=4$ in Fig. 5.2) input wires. There are 15 different combinations of x and y parameter values, two input wires can be selected from K lines in ${}^K C_2$ ways, and the controlling and controlled inputs have 2 different combinations; so a column can be constructed in $15 \times {}^K C_2 \times 2$ different ways. Again any of the columns could be placed in any position in the cascade. Hence there are $(15 \times {}^K C_2 \times 2)^L$ different cascades. There are $(15 \times 6 \times 2)^5 = 188956800000$ different combinations of a 2-variable function realized by a 5-column cascade (ignoring the fact that the number of columns required is unknown at the time of beginning of the synthesis

process). So the search space to find a proper or optimal solution is exponentially large. Hence it will not be feasible at all to use any deterministic or direct method. That is why we have chosen EA which is capable to search a huge solution space within a reasonable amount of time. In our method we have used Genetic Algorithm (GA) with real-valued encoding of the chromosome using complex data structures. Different aspects of our proposed algorithm are presented in the subsequent sections.

5.5.1 Problem Encoding

In the proposed EA we use the model of synthesizing multi-output ternary function using cascades of the new proposed gates as discussed in Section 5.3. In this model, for initial input to the EA, we added three constant signals 0, 1, and 2 for a number of times. Then after convergence of the EA we eliminate the unused constant input lines from the final circuit. We use variable length chromosome, however, we kept the maximum length 3^{n+3} for an n -variable function.

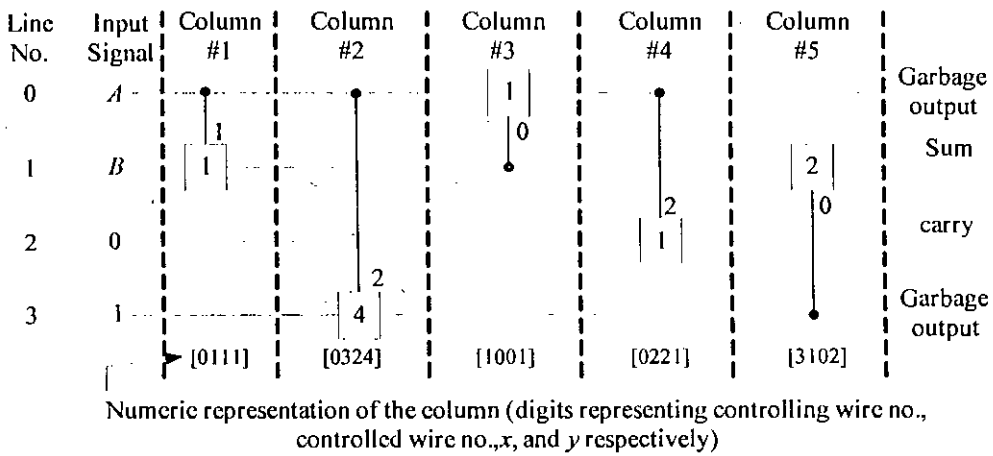


Fig 5.5: Encoding of the Ternary Half Adder circuit

Initially the chromosomes are generated with different length, and then due to mutation, the length is (possibly) changed. After convergence of the EA, some columns may be there in the circuit who do not contribute to the function output at all. These columns are also eliminated from the circuit.

The primary input lines and the constant signal lines are numbered starting from 0 as shown in Figure 5.5. Each of the columns in the circuit is represented by an ordered 4-tuple consisting of controlling wire number, controlled wire number, parameter x , and parameter y of the associated gate as shown in Figure 5.5. Using this notation the chromosome representing the circuit of Figure 5.5 is shown in Figure 5.6. Here each of the columns in the circuit is a gene in the chromosome.

0111	0324	1001	0221	3102
------	------	------	------	------

Fig 5.6: Chromosome representing the circuit in Figure 5.5

5.5.2 Fitness Components

In the proposed EA, we try to optimize the cost of the circuit by

- Reducing the number of genes in the chromosome, in other words reducing the number of columns (i.e. gates) in the circuit.
- Reducing the number of wires in the circuit (the width of the scratchpad register), i.e. increasing the number of unused constant input lines.

For this reason we used three fitness components –

- i. Output truth vector fitness,
- ii. Chromosome length fitness, and
- iii. Scratchpad width fitness.

In order to determine the output truth vector fitness we group the truth values as stated in Definition 5.1.

Definition 5.1: Given an n -input m -output ternary function f represented as m truth vectors (one vector for each output), where the locations of the truth values are designated from 0 to $3^n - 1$ for each truth vector. Every truth vector is partitioned into n types of sub-vectors, each type having a sub-vector length 3^{n-i} consisting of consecutive truth values starting from location $j3^{n-i}$ for $i = 1, 2, \dots, n$ and $j = 0, 1, \dots, (3^i - 1)$.

In this partitioning of the truth vector, i determines the length of a sub-vector and j determines the starting location of the sub-vector. For example, if $n = 3$, then for $i = 1$ the sub-vector length is $3^{n-1} = 3^{3-1} = 9$ and $j = 0, 1, (3^1 - 1 = 3^1 - 1 =) 2$. Therefore the starting locations of the sub-vectors are 0, 9, and 18. Similarly, this partitioning technique partitions the truth vectors into

Type 1: 3^{n-0} sub-vectors of length 1 starting from locations 0, 1, 2, ..., $3^n - 1$.

Type 2: 3^{n-1} sub-vectors of length 3 starting from locations 0, 3, 9, ..., $3^n - 3$.

Type 3: 3^{n-2} sub-vectors of length 9 starting from locations 0, 9, 18, ..., $3^n - 9$.

▪
▪

Type n: $3^{n-(n-1)} = 3$ sub-vectors of length 3^{n-1} starting from locations 0, 3^{n-1} , and $2 \times 3^{n-1}$.

For example, Figure 5.7 shows the three types of Sub-vectors of an arbitrary 3-input 2-output ternary function.

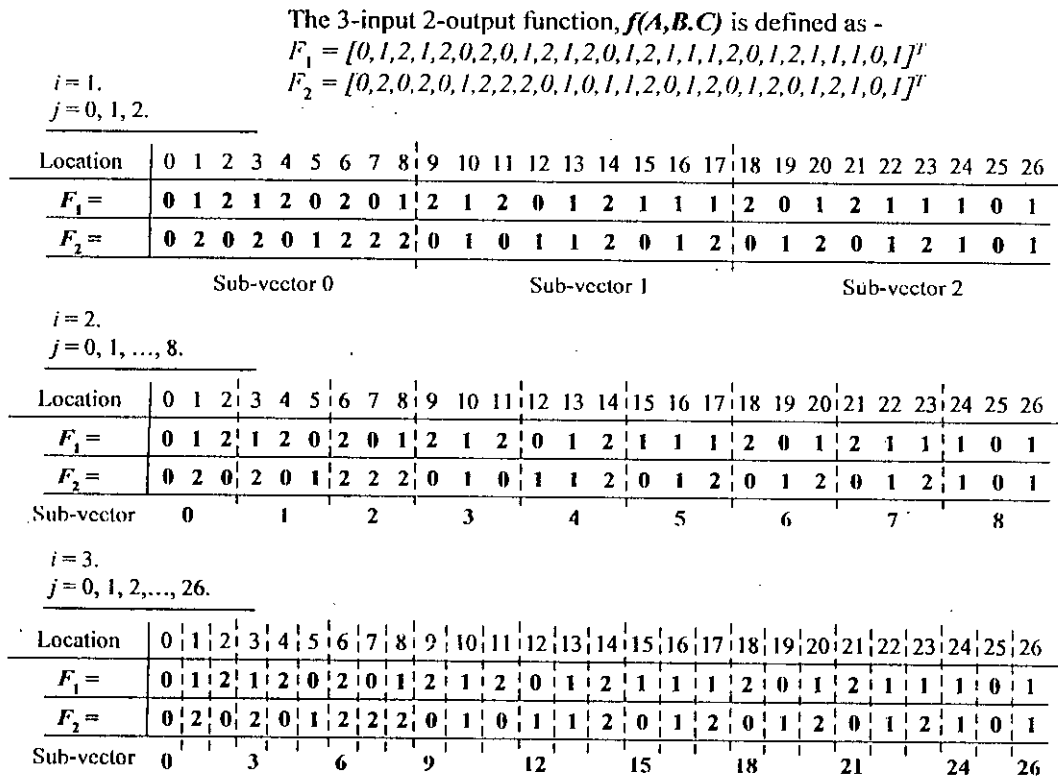


Fig 5.7: Sub-vectors of an arbitrary 3-input 2-output Ternary Function.

Sub-vector fitness: The sub-vector fitness, for a given output k ($k = 1, 2, \dots, m$), for sub-vector type i (having sub-vector length 3^{n-i}) is defined as follows:

$$S_{k,i} = \frac{Nr_{k,i}}{3^i}$$

Where, $Nr_{k,i}$ is the number of totally realized sub-vectors of Type i for the k th output along any wire.

When, for a given output k , all the sub-vectors of type i are totally realized, then $S_{k,i} = 1$.

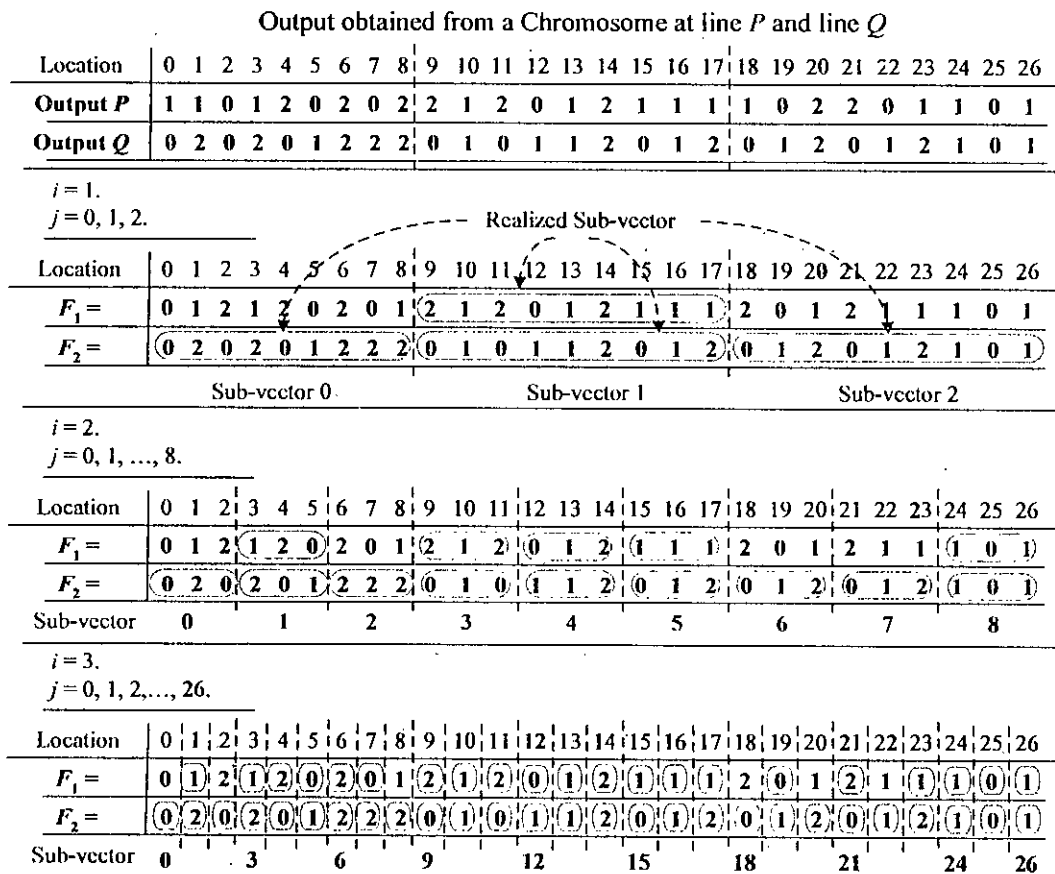


Fig 5.8: Realization of sub-vectors

For example, consider Figure 5.8; *Output P* and *Output Q* are respectively obtained at two arbitrary output lines P and Q of a quantum cascade while realizing the 3-input 2-output function $f(A,B,C)$ shown in Figure 5.7. The realized sub-vectors are shown marked in Figure 5.8. Since $n = 3$, there are three types of

sub-vectors – Type 1, Type 2, and Type 3. Let us explain Figure 5.8 for each of the types of subvectors.

Type 1: Both F_1 and F_2 are having 3 sub-vectors each. Only the Sub-vector 1 is realized for F_1 , while all three sub-vectors are realized for F_2 . Therefore,

the Sub-vector fitness of F_1 of type 1 is $S_{1,1} = \frac{1}{3^1} = \frac{1}{3}$ and that of F_2 is

$$S_{2,1} = \frac{3}{3^1} = \frac{3}{3} = 1.$$

Type 2: Both F_1 and F_2 are having 9 sub-vectors each. Five sub-vectors are realized for F_1 , while all the nine sub-vectors are realized for F_2 .

Therefore, the Sub-vector fitness of F_1 of type 2 is $S_{1,2} = \frac{5}{3^2} = \frac{5}{9}$ and that

$$\text{of } F_2 \text{ is } S_{2,2} = \frac{9}{3^2} = \frac{9}{9} = 1.$$

Type 3: In the similar way the Sub-vector fitness of F_1 of type 3 is

$$S_{1,3} = \frac{21}{3^3} = \frac{21}{27} \text{ and that of } F_2 \text{ is } S_{2,3} = \frac{27}{3^3} = \frac{27}{27} = 1.$$

Individual output truth vector fitness: Individual output truth vector fitness for output k is defined as follows:

$$O_k = p + \sum_{i=1}^n S_{k,i}$$

Where $p=1$ if output k is totally realized along any wire, 0 otherwise. When an output k is totally realized along a wire, then $O_k = 1+n$ as there are n types of sub-vectors.

For example, consider Figure 5.8 again. Earlier we have calculated the values of $S_{k,i}$. Note that F_2 is totally realized along *Output Q*. Now, individual output truth

vector fitness of F_1 is – $O_1 = 0 + \sum_{i=1}^3 S_{1,i} = \frac{1}{3} + \frac{5}{9} + \frac{21}{27} = \frac{45}{27}$, and that of F_2 is –

$$O_2 = 1 + \sum_{i=1}^3 S_{2,i} = 1 + (1+1+1) = 4, \text{ which is equal to } 1+n.$$

Output truth vector fitness: The output truth vector fitness is defined as follows:

$$O = \sum_{k=1}^m O_k$$

When all the m outputs are realized, then eventually it becomes $O = m(1+n)$. For the case of our continuing example of Figure 5.8, the Output truth vector fitness is $O = O_1 + O_2 = \frac{45}{27} + 4 = 5.667 < m(1+n) = 3(1+2) = 9$. Hence, observing the value of O , we can say that the function $f(A,B,C)$ is not realized.

To find the output truth vector fitness, we compute the resulting truth vector for all wires and then the best fit wire is selected for each of the given output.

Chromosome Length Fitness: The Chromosome Length Fitness (or cascade length fitness) is defined as follows –

$$C = \frac{L_{chr}}{L_{max}}$$

Where, L_{max} is the maximum allowable length of the chromosomes and L_{chr} is the length of the chromosome under consideration.

Scratchpad Width Fitness: The Scratchpad Width Fitness is defined as follows –

$$W = \frac{Ncu}{Nc}$$

Where, Nc is the total number of constant lines used in the synthesis and Ncu is the number of constant line those do not contribute in the realization of the function.

We rank the population using O , C , and W as primary, secondary, and tertiary key respectively. When all the m outputs are realized by any chromosome then the value of O will be $m(1+n)$. Therefore, if the output vector fitness O of any chromosome is $m(1+n)$, then the chromosome is a solution for the given function.

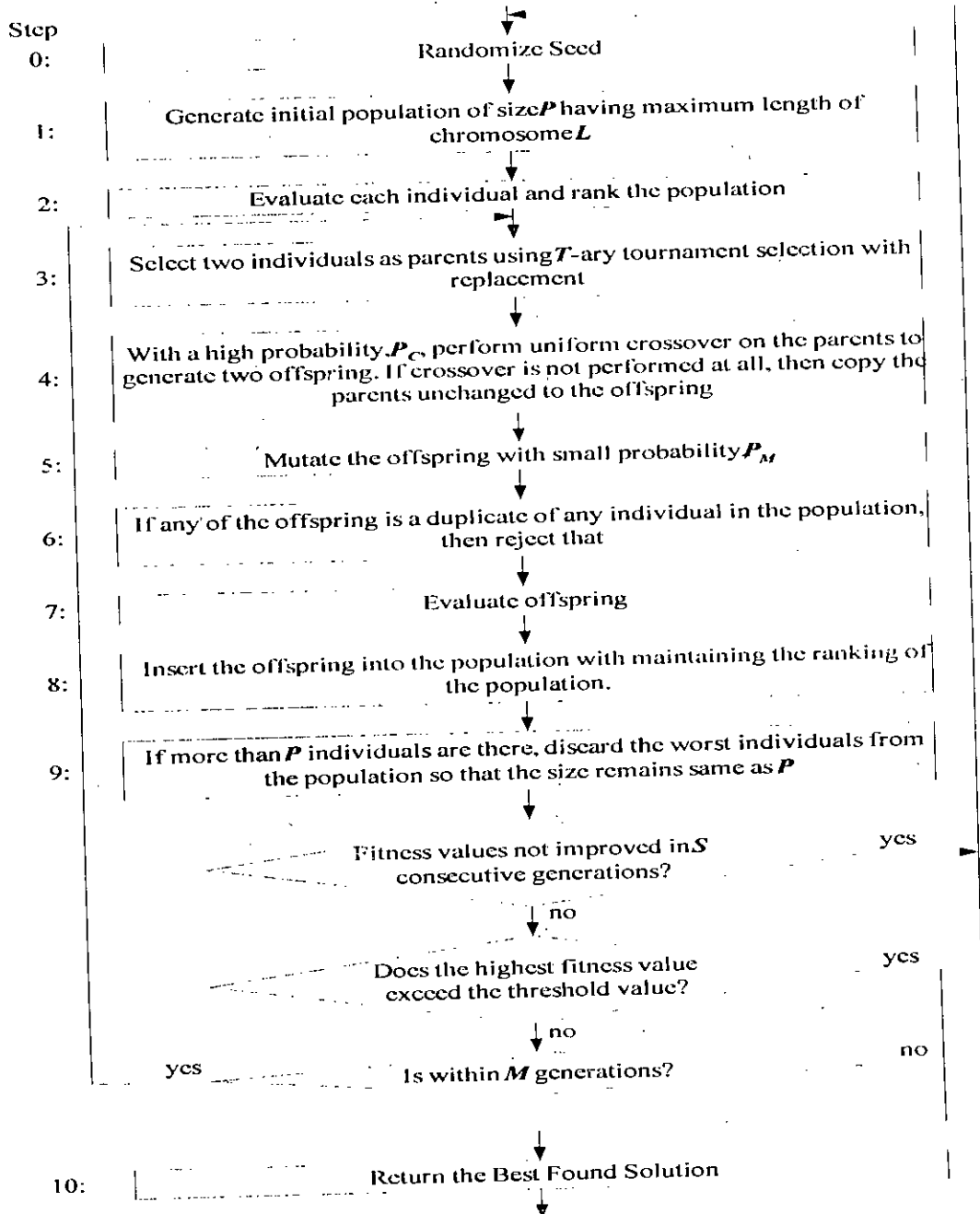


Fig 5.9: Flowchart of the proposed Evolutionary Algorithm

5.5.3 Description of the Evolutionary Algorithm

As the model of our circuit synthesis is not well structured, we want to make sure that the “so far best found” solution is not lost in the successive generations.

Therefore, we used the simple steady-state GA with T-ary tournament selection with replacement for selecting the parents, classical uniform crossover operator, mutation operator, and a problem specific repairing operation. The proposed EA is shown in Figure 5.9. For a given set of population size P , crossover probability P_C , mutation probability P_M , tournament size T , and maximum number of generations M , we repeat the EA at most R times with random seed. If for a given run of the EA, the fitness value does not improve within S consecutive generations then we do the following:

- i. Stop the run and go to the next repetition with random seed if no solution is found yet,
- ii. Stop the EA and return the best solution otherwise.
- iii. If no solution is obtained in R repetitions, then generate "Fail" and stop the EA.

The steps of our proposed EA are explained in the following sub-sections.

Step 0:

This is the initialization of the EA. At this step we reset the random number generator if it is the first repetition; for all subsequent repetition, the generator is initialized with a random seed.

Step 1:

At this step the population is initialized. A total number of P individuals are generated randomly. The length of each individual is also selected randomly between 1 to L . Each of the individuals is randomly generated according its respective length.

Step 2:

Each of the individuals is evaluated as described in Section 5.5.2. The output produced at every primary and constant line is evaluated for all the given m functions. Then we applied "marriage matching" strategy to get the best output vector fitness for each of the individuals. Then the individuals are ranked according to their respective fitness. In doing this, we considered output vector fitness O first. If more than one individual are

having the same O value, then we consider the Length fitness C . If two or more individual are having the same O value as well as same C value, then the Width fitness W is considered. For example Figure 5.10 shows the ranking of five chromosomes realizing an arbitrary function.

Chromosome	O	C	W	Rank	Chromosome	O	C	W
Chr1	4	0.6	0.5	1	Chr2	6	0.4	0.6
Chr2	6	0.4	0.6	2	Chr4	4	0.8	0.7
Chr3	3	0.7	0.5	3	Chr5	4	0.6	0.9
Chr4	4	0.8	0.7	4	Chr1	4	0.6	0.5
Chr5	4	0.6	0.9	5	Chr3	3	0.7	0.5
(a) Before Ranking					(b) After Ranking			

Fig 5.10: Ranking of the individuals in an arbitrary population

Step 3:

This step selects the parents to generate the offspring. The whole population is divided into two (perhaps non-disjoint) sets of individuals randomly. Then the two best individual from the two sets are selected as the parents. As the sets are non-disjoint, it is to be ensured that the two parents are not the same one. Actually the copies of the parents are taken to apply further operations on them keeping the population unchanged.

Step 4: (Crossover)

The crossover operation is applied in this step. We are using uniform crossover. That means for each and every gene in the parent chromosomes are to be checked for availability of crossover for that position. Since the parents can be of different lengths, we have to take some extra measure to perform the operation. Assuming that the lengths of the two parents are λ_1 and λ_2 . Also assuming that $\lambda_1 > \lambda_2$, for every gene position i ($i = 1, 2, \dots, \lambda_2$), a random probability ρ_i is generated. If $\rho_i < P_c$, then the gene pair at location i are exchanged in the parents. The newly obtained individuals are the offspring. Figure 5.11 explains the crossover operation.

$P_c = 0.8$

Parent#1	$[\lambda_1 = 5]$	2014	8221	1001	0304	2412		
Parent#2	$[\lambda_2 = 7]$	3223	1821	0124	3311	4102	8903	1011
	$\rho_i \rightarrow$	0.9	0.3	0.65	0.89	0.7		

(a) Selected parents before crossover.
The genes to be interchanged are marked.

Offspring#1	2014	1821	0124	0304	4102		
Offspring#2	3223	8221	1001	3311	2412	8903	1011

(b) Obtained offspring after crossover.

Fig 5.11: Crossover operation

Step 5: (Mutation)

At this step mutation is applied to the offspring. For each offspring, one random probability μ is generated. If $\mu < P_M$, then mutation will be applied according as the following rules –

- i. Randomly select a gene in the selected offspring.
- ii. Select one of the following mutation operation randomly and perform:
 - a. Delete the gene provided that the length of the offspring does not become zero.
 - b. Insert a randomly generated gene after this one provided that the length of chromosome does not exceed the maximum length, L.
 - c. Modify this gene randomly.

This is shown in Figure 5.12 assuming that $\mu < P_M$ for all cases.

Step 6:

If any of the offspring is the duplicate of any individual already in the population, reject the offspring.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Offspring#1	2014	1821	0124	0304	4102		
Offspring#2	3223	8221	1001	3311	2412	8903	1011

(a) Offspring before mutation.

Offspring No.	Mutation Point	Mutation Type
1	3	Modify
2	6	Delete

(b) Randomly generated mutation points and types.

Offspring#1	2014	1821	3821	0304	4102		
Offspring#2	3223	8221	1001	3311	2412	1011	

(c) Offspring after mutation.

Fig 5.12: Mutation Operation

Step 7:

Evaluate the offspring as described in Article 5.4.2. If the output vector fitness of the offspring exceeds the fitness threshold value, then eliminate redundant columns and unused constant lines from the offspring. Figure 5.11 shows redundant columns and constant lines in an arbitrary circuit.

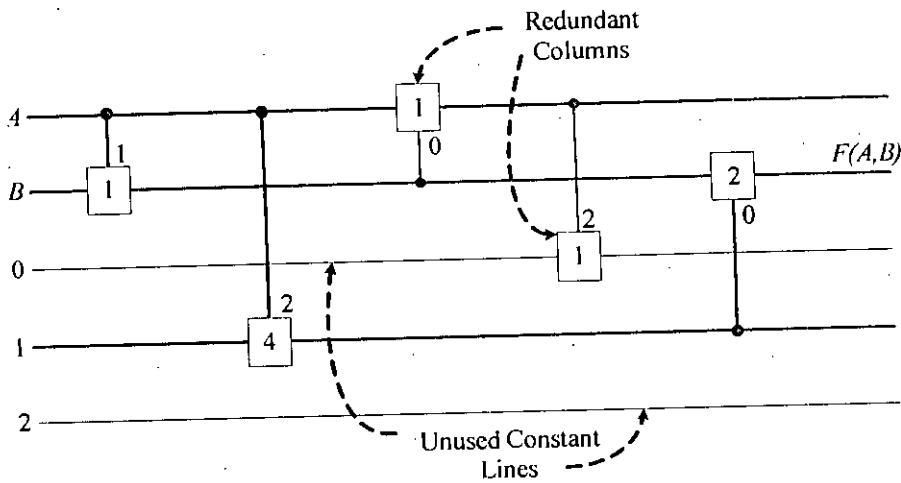


Fig 5.11: Redundant Columns and Unused Constant Lines in an arbitrary Quantum Cascade

Step 8:

Insert the offspring into the population maintaining the ranking according to their fitness. This is shown in Figure 5.14 for an arbitrary population of size 5.

Chromosome	<i>O</i>	<i>C</i>	<i>W</i>
<i>Chr1</i>	5	0.4	0.6
<i>Chr2</i>	4	0.8	0.7
<i>Chr3</i>	4	0.6	0.9
<i>Chr4</i>	4	0.6	0.5
<i>Chr5</i>	3	0.7	0.5

(a) Population before insertion

<i>Offsp1</i>	3	0.7	0.4
<i>Offsp2</i>	5	0.4	0.3

(b) Offspring

Rank	Chromosome	<i>O</i>	<i>C</i>	<i>W</i>
1	<i>Chr1</i>	6	0.4	0.6
2	<i>Offsp2</i>	5	0.4	0.3
3	<i>Chr2</i>	4	0.8	0.7
4	<i>Chr3</i>	4	0.6	0.9
5	<i>Chr4</i>	4	0.6	0.5
* 6	<i>Chr5</i>	3	0.7	0.5
* 7	<i>Offsp1</i>	3	0.7	0.4

(c) Population after insertion

* Individual to be discarded in step 9.

Fig 5.14: Insertion of the offspring into the Population with Ranking

Step 9:

Discard the worst ranked individuals from the population so that the size of population remains same. For example, in Figure 5.14(c), *Chr5* and *Offsp1* are to be discarded from the population (marked with *). Then do the following conditional actions:

- i. If the fitness is not improved in consecutive *S* generations, i.e. there is no change in the population for a long time; it indicates that the EA is in stagnation. In that case go to Step 0 to restart the whole process. Otherwise go to the next condition checking (in ii).
- ii. If the output vector fitness of the top ranked individual exceeds the threshold value defined in Definition 5.1, i.e. at least one solution has been found; then go to Step 10. Otherwise go to the next condition checking (in iii).
- iii. If the maximum number of generation limit is reached, then go to Step 0 to restart the whole process. Otherwise go to Step 3 to continue the iteration of the EA.

Step 10:

Return the solution produced by the top ranked individual.

Chapter 6

Experimental Results and Discussion

6.1 Introduction

We have presented an EA based synthesis method of multi-output ternary logic using quantum cascades. We also discussed the theoretical back ground of multi-output, multi-valued logic, reversible logic, quantum computation, Evolutionary Algorithms, etc earlier. Now in this chapter we present the experimental findings. Mainly the results are compared with the results shown in [36] as it is the only work of same type in this field done so far.

To compare the efficiency of our proposed EA based method we used two cost factors. They help us to estimate the cost of the circuit synthesized by our proposed method. These are –

1. **Length of the Cascade:** This is the total number of columns in the cascade. Each column is realized using one gate. Therefore, this is the number of gates required to realize the circuit.
2. **Scratchpad width:** This is the total number of constant input/output lines and the primary input/output lines required to realize the circuit.

It is worthless to mention that the cost of the quantum cascade is directly proportional to both of these factors. So an optimal realization is supposed to be of minimum length as well as of minimum width.

6.2 Experimental Setup and Findings

We have written C++ program to implement the proposed EA. We used the program to realize the benchmark functions given in [36]. We experimented with

the ternary half adder function with different EA parameter combinations to see the effect of the parameters on this type of problems. We used population size $P = 100, 200, 300$; crossover probability $P_C = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$; and mutation probability $P_M = 0.02, 0.04, 0.06, 0.08, 0.10$. We selected two parents using tournament selection using tournament size $T = 2$. Uniform crossover is used as [36] showed that among different classical crossover technique, uniform crossover performs better than the others.

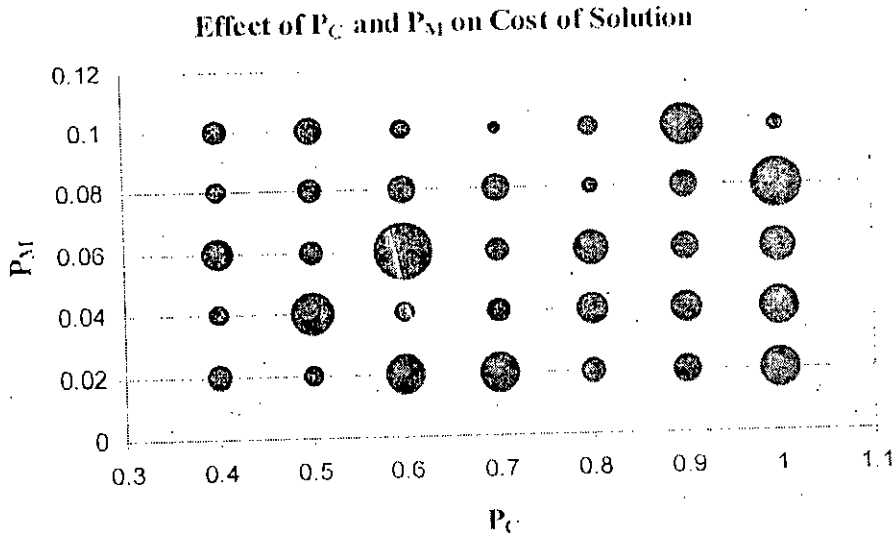


Fig 6.1: Effect of P_C and P_M on cost of Solution

Figure 6.1 shows the overall effect of the Crossover probability (P_C) and mutation probability (P_M) on the cost of the circuit. The sizes of the circles are proportional to the sum of length and width of the circuit obtained from the respective P_C - P_M combination. For every P_C - P_M combination we have averaged the cost of the circuit obtained from population size (P) of 100, 200, and 300. We can see that the EA performs better when P_C is within the range 0.6 to 0.8 and P_M is within 0.08 to 0.1. Unusual behavior shows at (0.6, 0.06) and (0.5, 0.04). Perhaps, if we allow the EA to run for much longer time, then it would no longer remain.

Figure 6.2 shows the effect of P_C on the length and width of the circuit separately. For a particular value of P_C the obtained length (and width) for $P_M = 0.02, 0.04, 0.06, 0.08, 0.1$ are averaged. Both the graphs show a tendency of

having better solution with lower values of P_C but, unusual behavior can be noticed around $P_C=0.6$, and 0.7 .

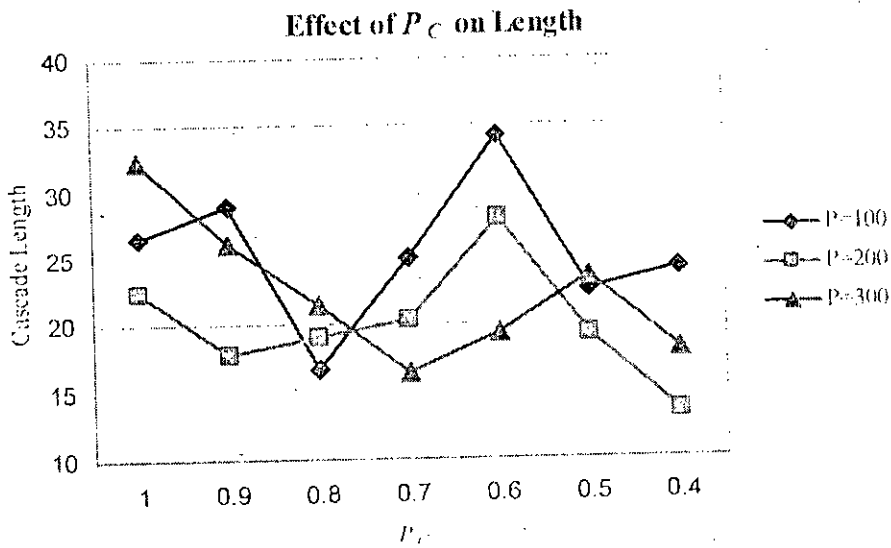


Fig 6.2: Effect of P_C on length of the circuit

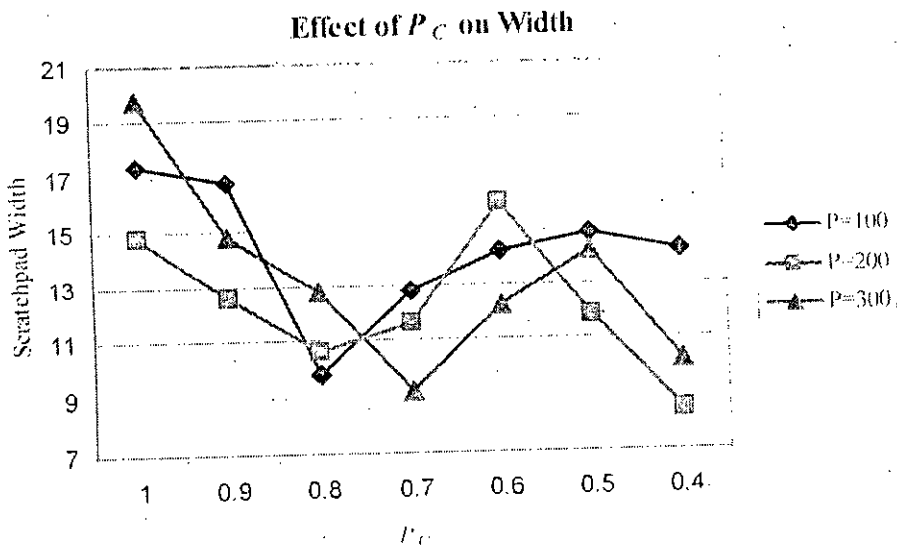


Fig 6.3: Effect of P_C width of the circuit

Figure 6.3 shows the effect of P_M on the length and width of the circuit separately. For a particular value of P_M the obtained length (and width) for $P_C = 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$ are averaged. Both the graphs shows a tendency of

having better solution with higher values of P_M but, again, unusual behavior can be noticed around $P_M=0.06$.

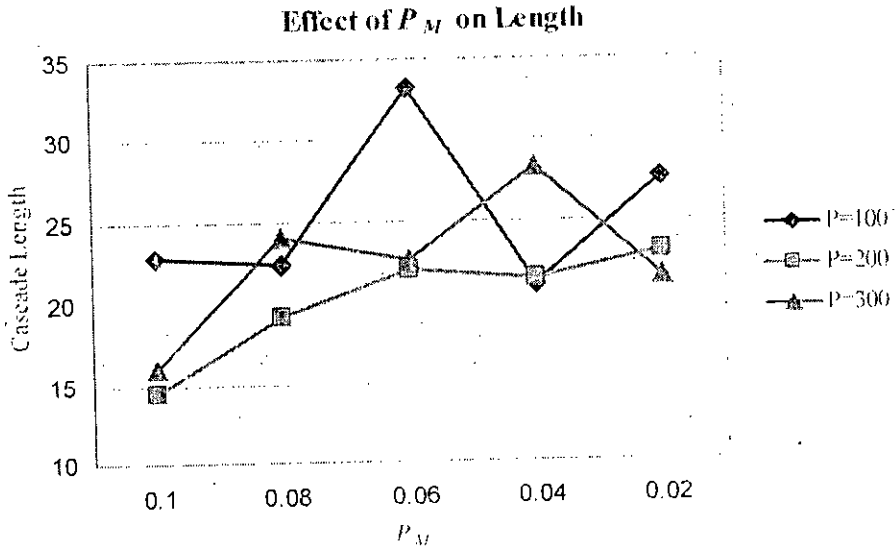


Fig 6.4: Effect of P_M on length of the circuit

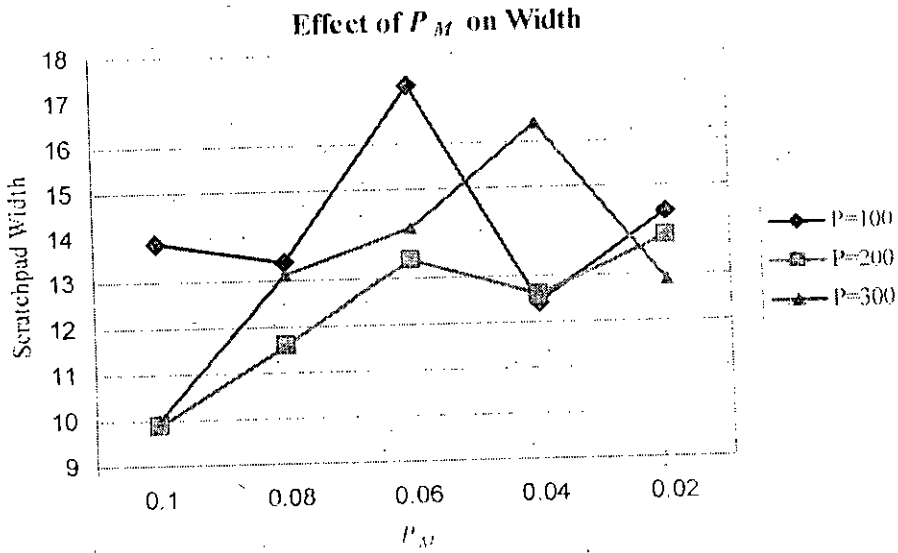


Fig 6.5: Effect of P_M on width of the circuit

Table 6.1 shows the length and width obtained for some benchmark ternary function using the proposed EA. Here initial length (also width) means the length (width) of the circuit when the EA found a solution for the first time. And the final

length (width) means the length (width) of the solution when the EA stops. Descriptions of the benchmark functions are given in Appendix B.

Function Name	Initial Length	Final Length	Initial Width	Final Width
3cy2	427	425	39	36
avg2	135	16	28	9
sum2	124	9	29	3
sum3	135	106	36	34
thadd	55	12	28	9
A2bcc	34	34	20	20
mul2	37	23	23	11
prod2	44	12	20	5
prod3	116	48	39	19

Table 6.1: Results obtained for different benchmark ternary functions.

The remaining part of this section shows the experimental findings for different ternary benchmark functions. Complete description of the ternary benchmark functions are given in Appendix B.

abc2:

This is a ternary function with 3-input and 1-output. Therefore the output vector fitness of a chromosome representing a solution is $1(3+1) = 4$.

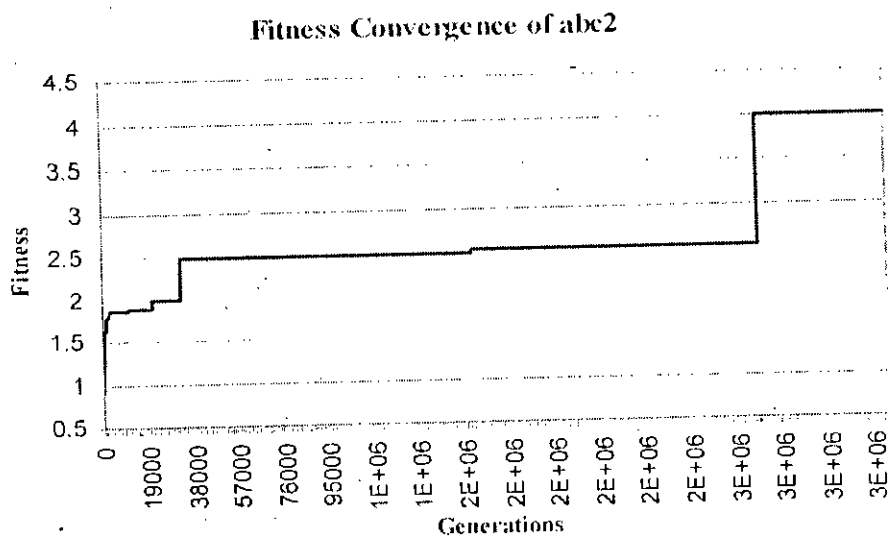


Fig 6.6: Convergence of output vector fitness for abc2.

Figure 6.6 shows the convergence of the output vector fitness. The EA starts with a very low fitness. Then there is an exponential convergence up to 325000 generations. After a long stagnation, there was a big jump and the EA achieved the fitness threshold at around 2755000 generations.

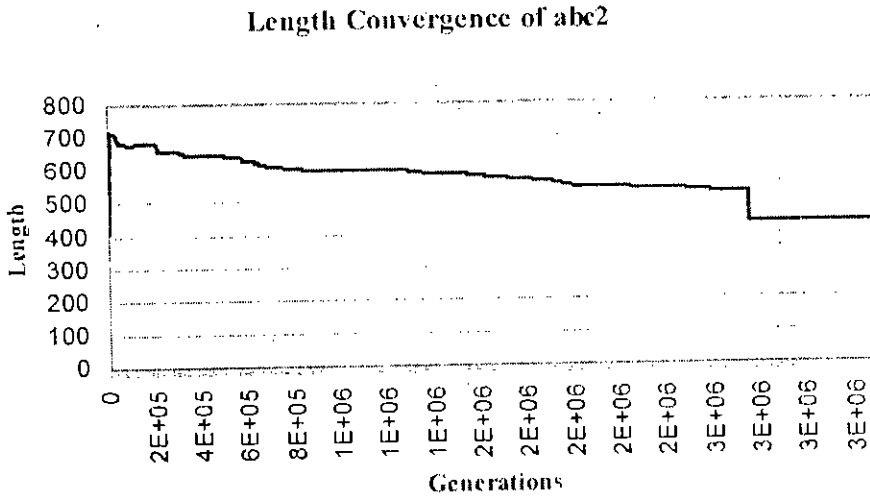


Fig 6.7: Convergence of length of cascade for abc2.

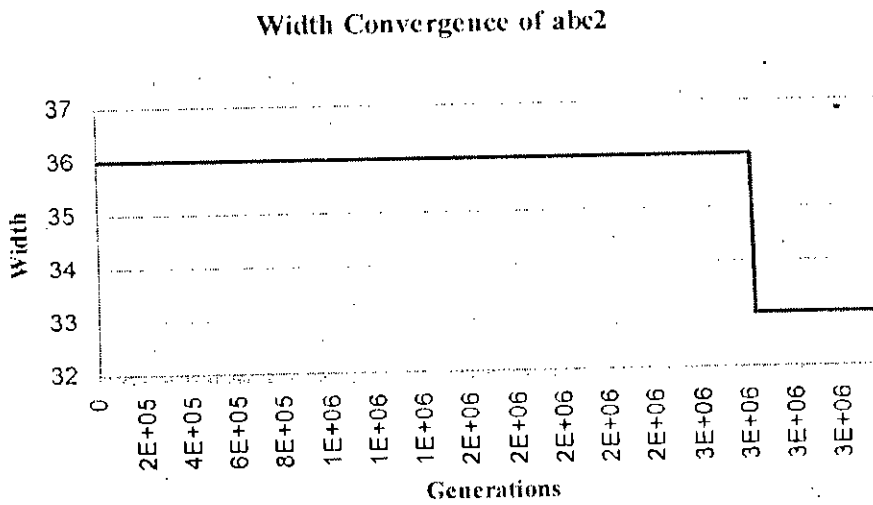


Fig 6.8: Convergence of scratchpad width for abc2.

Figure 6.7 shows the length convergence for the function. There is also a big jump at around 2755000 generations, when the EA achieved the fitness threshold. At

this point our proposed method eliminates the columns that do not have any contribution to the final output.

Figure 6.8 shows the convergence of scratchpad width for the same function. We can see that there is no significant change in the width until the fitness threshold is achieved. Again at this point our proposed method eliminates the constant lines that do not contribute to the final output.

mul2:

This is a ternary function with 2-input and 2-output. Therefore the output vector fitness of a chromosome representing a solution is $2(2+1) = 6$.

Figure 6.9 shows the convergence of the output vector fitness. The EA starts with a lower fitness. Then there is a quick convergence up to 20000 generations. After a long stagnation, there was a big jump and the EA achieved the fitness threshold at around 785000 generations.

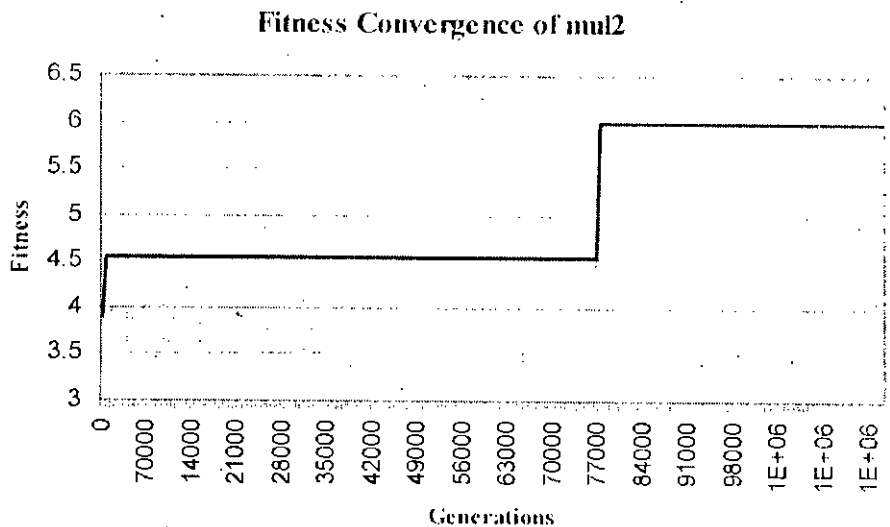


Fig 6.9: Convergence of Output Vector Fitness for mul2.

Figure 6.10 shows the length convergence for the function. There a exponential convergence for around 10000 generations. After that there was a long stagnation.

We can notice a gradual convergence after the EA have achieved the fitness threshold.

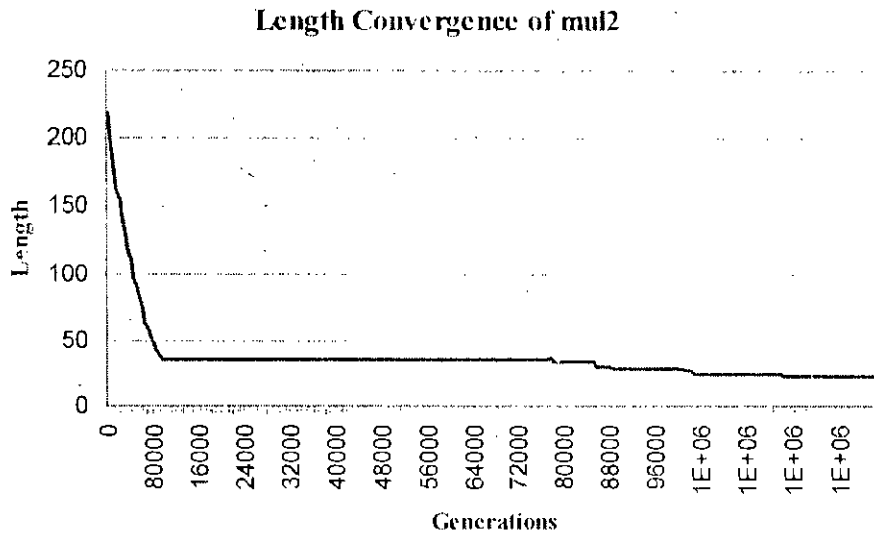


Fig 6.10: Convergence of length of cascade for mul2.

Figure 6.11 shows the convergence of scratchpad width for the same function. We can see that there is no significant change in the width until the fitness threshold is achieved. Again at this point our proposed method eliminates the constant lines that do not contribute to the final output. The width is also decreased after that in an exponential manner.

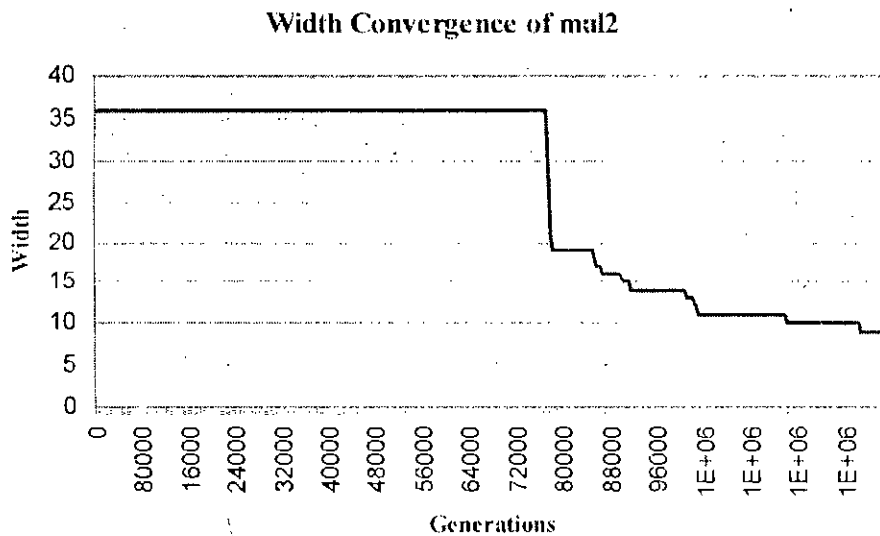


Fig 6.11: Convergence of scratchpad width for mul2.

a2bcc:

This is a ternary function with 3-input and 1-output. Therefore the output vector fitness of a chromosome representing a solution is $1(3+1) = 4$.

Figure 6.12 shows the convergence of the output vector fitness. The EA starts with a very low fitness. Then there is an exponential convergence up to 175000 generations. And then the EA achieved the fitness threshold at around after a sharp rise of the fitness.

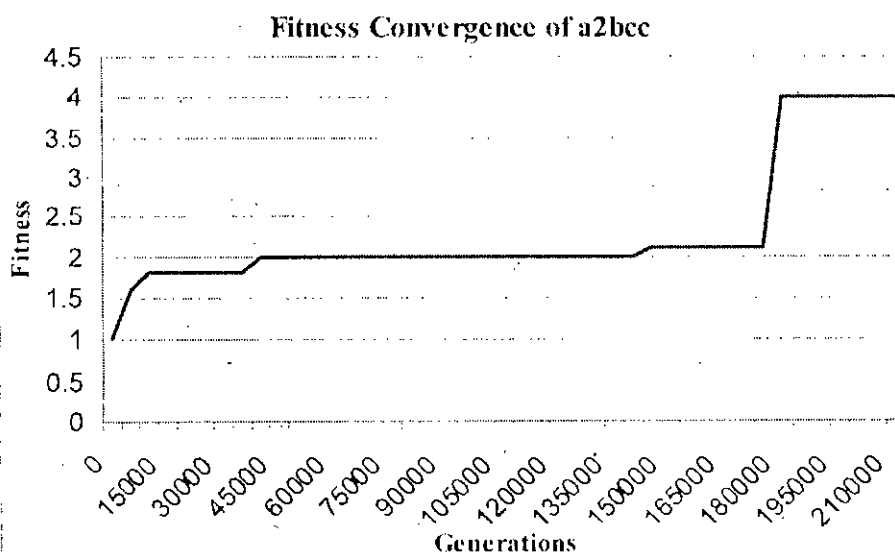


Fig 6.12: Convergence of Output Vector Fitness for a2bcc.

Figure 6.13 shows the length convergence for the function. The EA converged almost in an exponential fashion throughout the whole process. There is also a relatively big jump when the EA achieved the fitness threshold. At this point our proposed method eliminates the columns that do not have any contribution to the final output.

Figure 6.14 shows the convergence of scratchpad width for the same function. We can see that there is no significant change in the width until the fitness threshold is achieved. Again at this point our proposed method eliminates the constant lines that do not contribute to the final output.

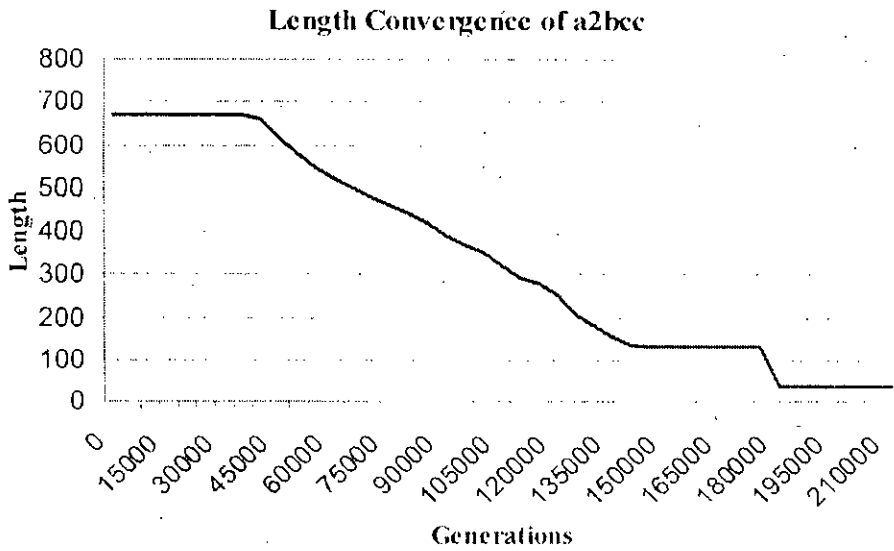


Fig 6.13: Convergence of length of cascade for a2bcc.

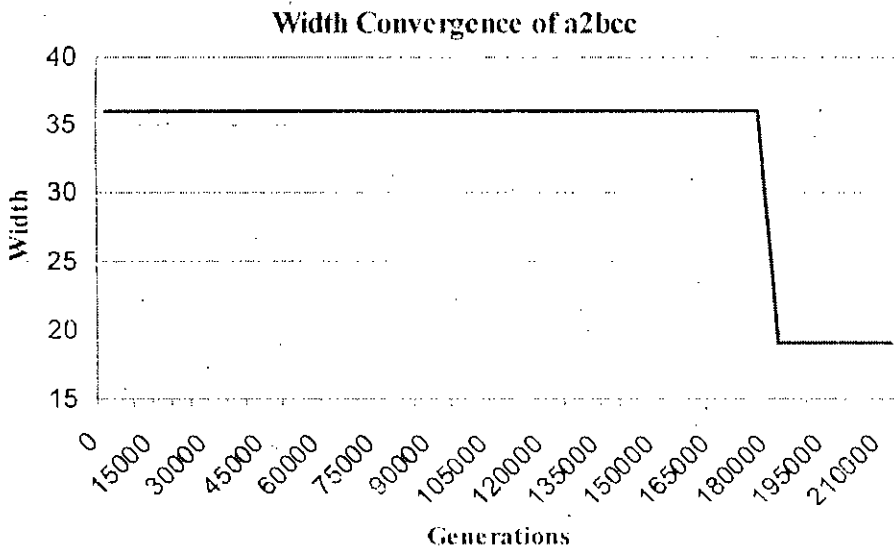


Fig 6.14: Convergence of scratchpad width for a2bcc.

thadd:

This is a ternary function with 2-input and 2-output. Therefore the output vector fitness of a chromosome representing a solution is $2(2+1) = 6$.

Figure 6.15 shows the convergence of the output vector fitness. The EA starts with a very low fitness. Then there are two almost big jumps and the EA achieved the fitness threshold at around 210000 generations.

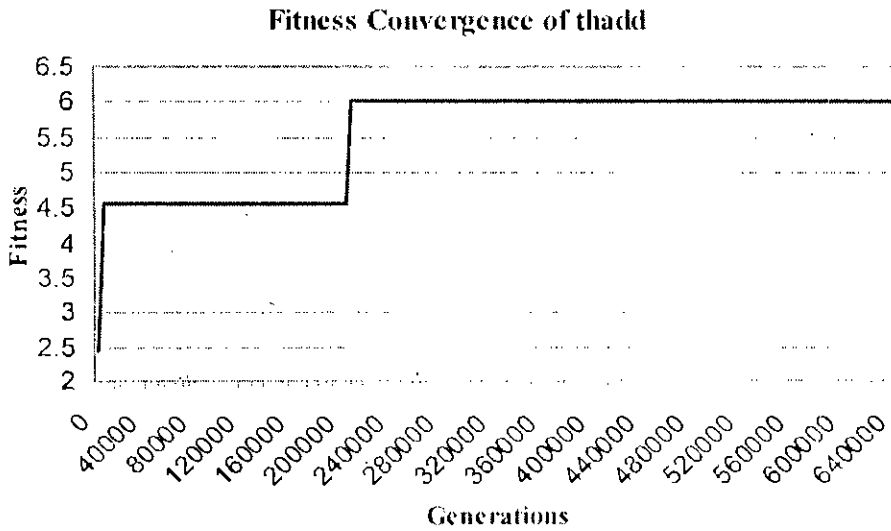


Fig 6.15: Convergence of Output Vector Fitness for thadd.

Figure 6.16 shows the length convergence for the function. After an initial negative convergence, there is an exponential convergence.

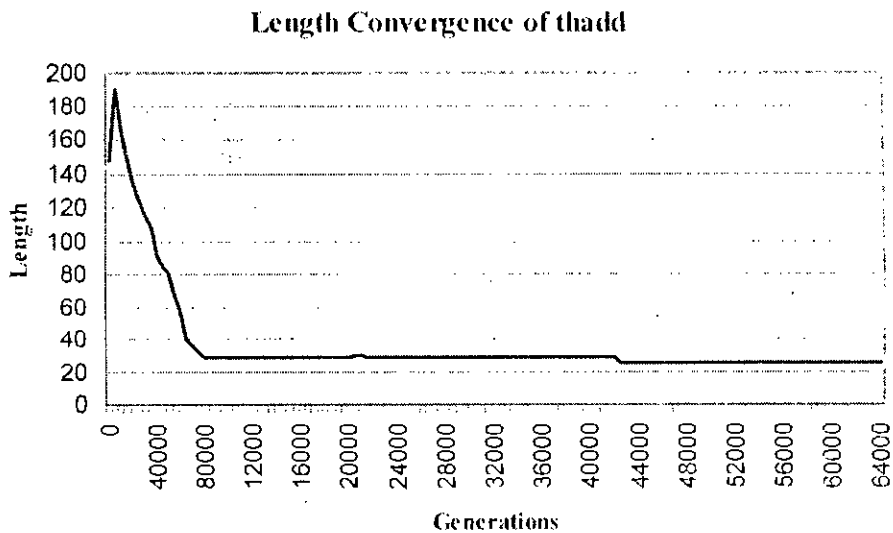


Fig 6.16: Convergence of length of cascade for thadd.

Figure 6.17 shows the convergence of scratchpad width for the same function. We can see that there is no significant change in the width until the fitness threshold is achieved. Again at this point our proposed method eliminates the constant lines that do not contribute to the final output.

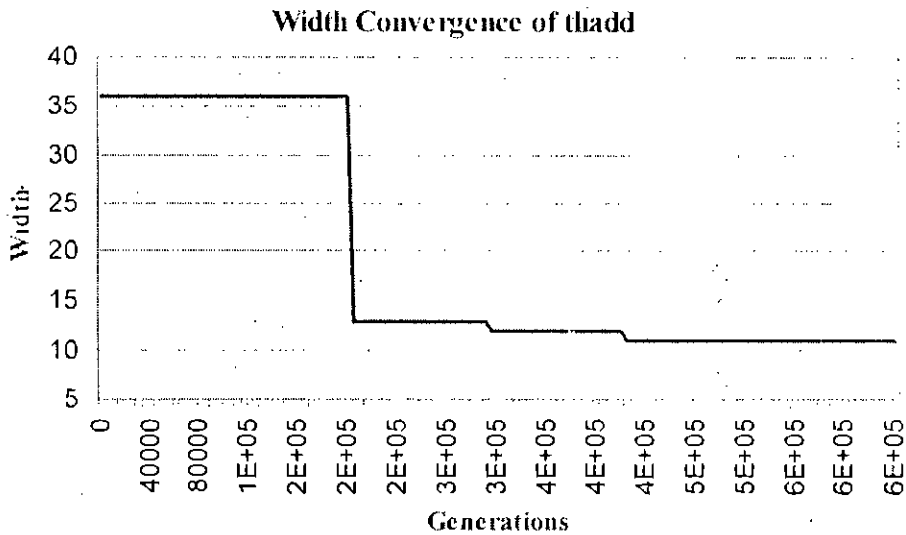


Fig 6.17: Convergence of scratchpad width for thadd.

prod3:

This is a ternary function with 3-input and 1-output. Therefore the output vector fitness of a chromosome representing a solution is $1(3+1) = 4$.

Figure 6.18 shows the convergence of the output vector fitness. Like the previously mentioned functions, the EA starts with a very low fitness, then there is an exponential convergence at primary stage, then a long stagnation, and finally it achieved the fitness threshold at around 470000 generations.

Figure 6.19 shows the length convergence for the function. There is also a big jump indicating that the EA has found a better fit chromosome with smaller length at that time.

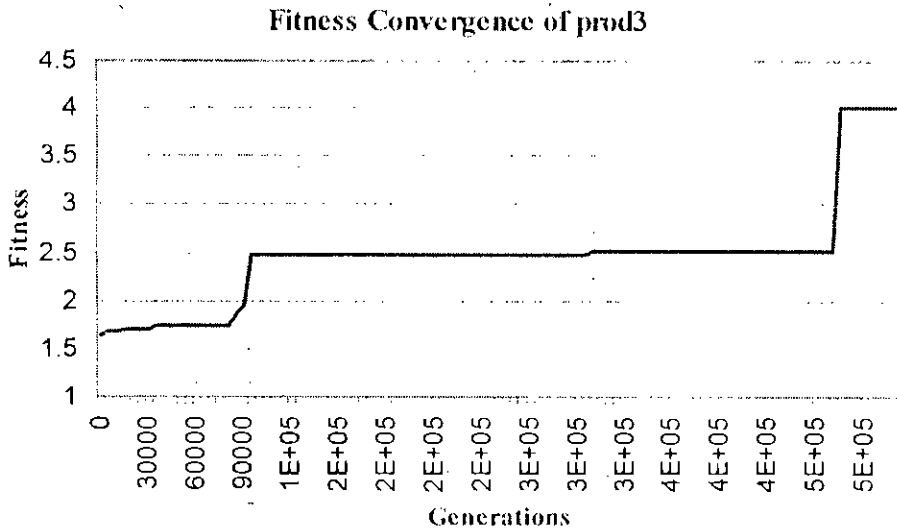


Fig 6.18: Convergence of Output Vector Fitness for prod3.

Figure 6.20 shows the convergence of scratchpad width for the same function. We can see that there is again an exponential convergence after the fitness threshold is achieved.

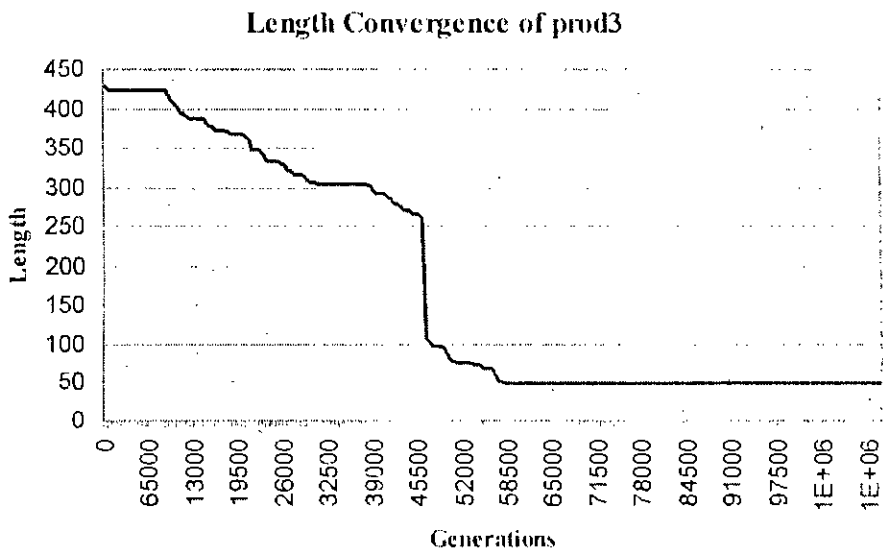


Fig 6.19: Convergence of length of cascade for prod3.

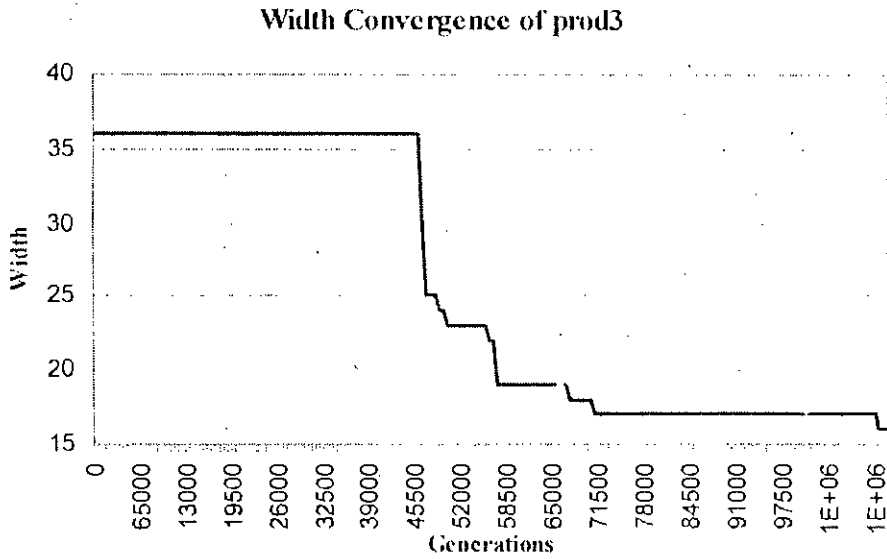


Fig 6.20: Convergence of scratchpad width for prod3.

avg2:

This is a ternary function with 2-input and 1-output. Therefore the output vector fitness of a chromosome representing a solution is $1(2+1) = 3$.

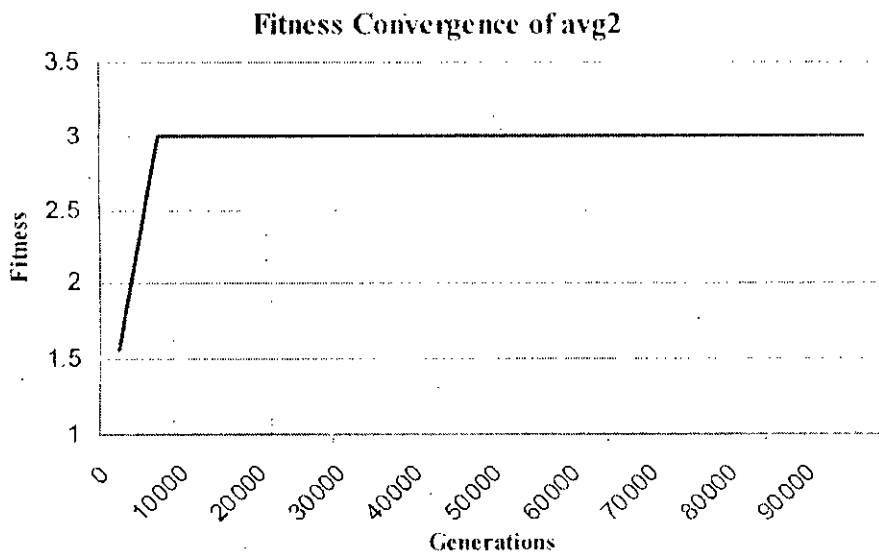


Fig 6.21: Convergence of Output Vector Fitness for avg2.

Figure 6.21 shows the convergence of the output vector fitness. The EA starts with a low fitness. Then there is a very quick convergence and reaches the fitness threshold within 7500 generations.

Figure 6.22 shows the length convergence for the function. There is also a big jump indicating that the EA has found a better fit chromosome with smaller length at that time.

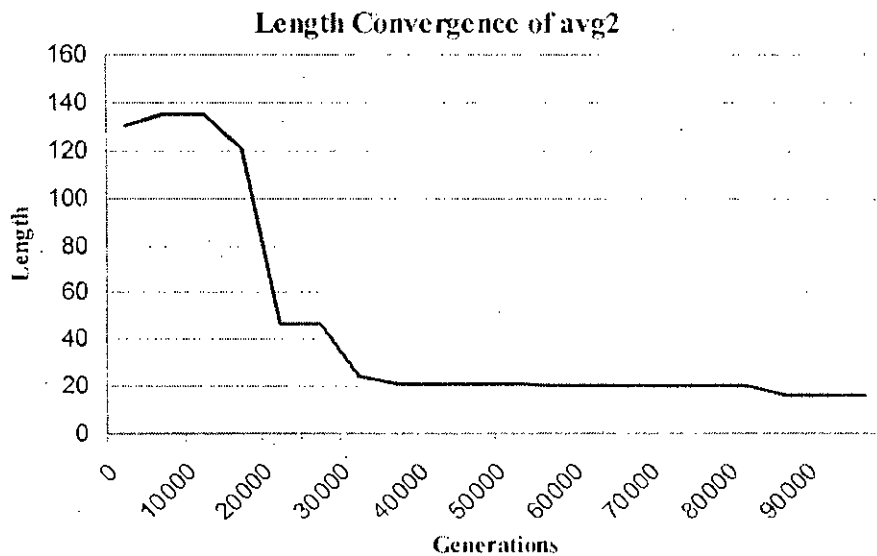


Fig 6.22: Convergence of length of cascade for avg2.

Figure 6.23 shows the convergence of scratchpad width for the same function. We can see that there is a consistent improvement throughout the whole evolution.

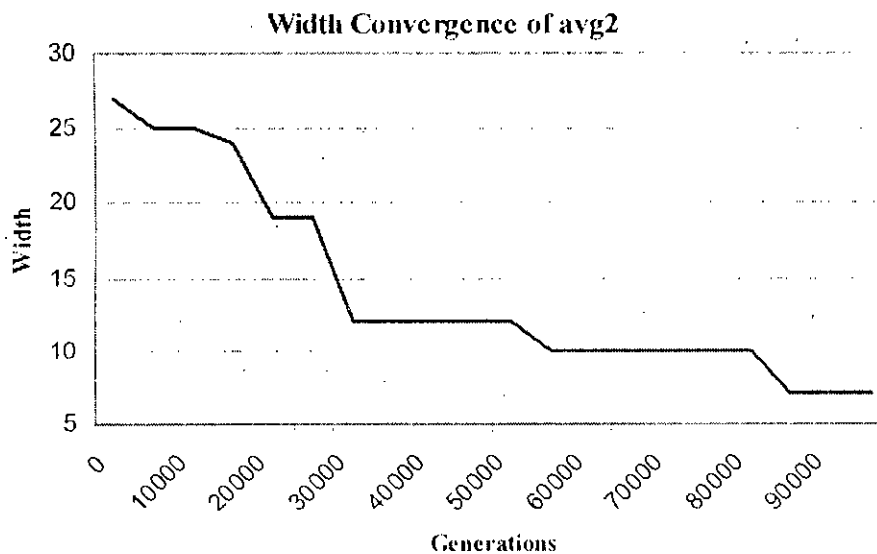


Fig 6.23: Convergence of scratchpad width for avg2.

Figure 6.24, Figure 6.25, and Figure 6.26 shows the realizations of three ternary benchmark functions obtained by the proposed method. Realizations of other circuits are a bit complex with a large number of columns, hence not shown.

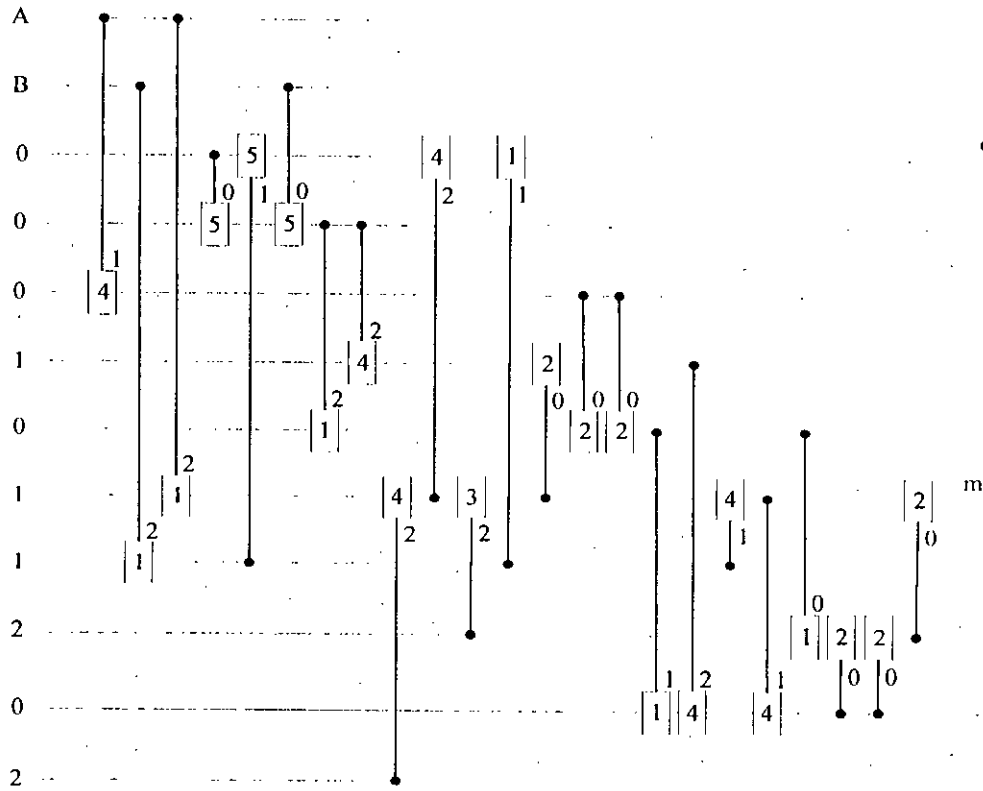


Fig 6.24: Realization of mul2 using the proposed method.

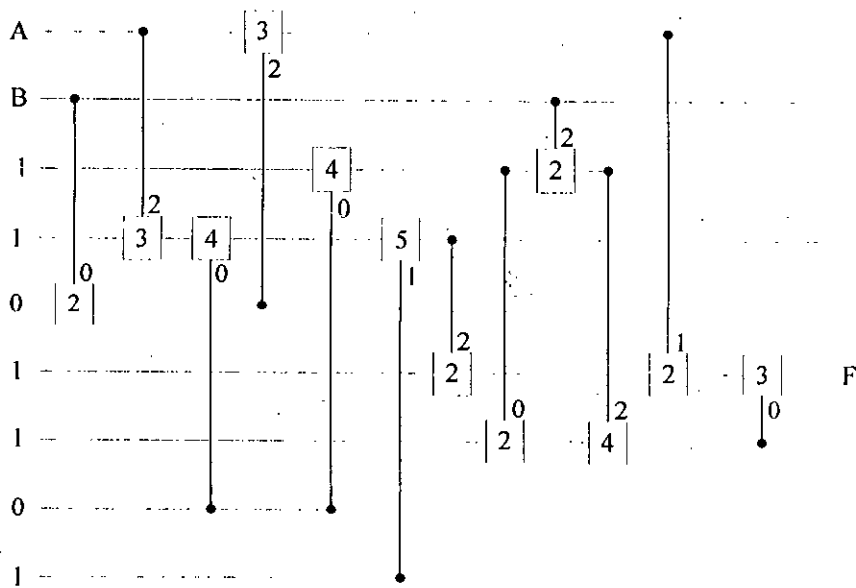


Fig 6.25: Realization of prod2 using the proposed method.

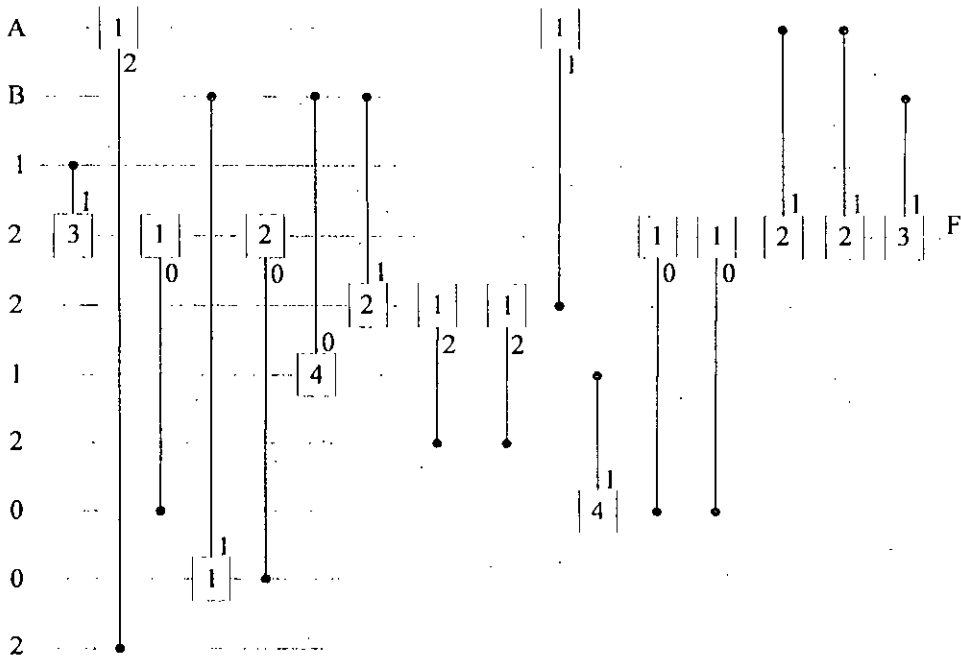


Fig 6.26: Realization of avg2 using the proposed method.

6.3 Conclusions

Experimental results obtained from our proposed EA based synthesis method is presented in this chapter. First, we have shown the effect of the EA parameters on the solution. We have shown the ranges of the parameter values for which the EA performs better. Then we have shown the results obtained for some benchmark ternary functions so that it could be compared with other methods later on.

Chapter 7

Conclusion

7.1 Concluding Words

We have presented an EA based method to synthesize multiple-output ternary functions using quantum cascades. Quantum computation is known to be the most prominent technology for future computers. The most important feature that attracts the attention of researchers is the “entanglement” of logical states. Moreover, reversible computers can be made using quantum technology that, theoretically, dissipates zero amount of heat. We have described the relevant theoretical background of Multiple-Valued Logic, Quantum Computations, Reversible Logic, and Evolutionary Algorithms.

A family of 2×2 quantum primitive gates is proposed. These gates are reversible and can directly be implemented using quantum technology. Besides, these gates are universal for ternary logic. We compared these gates with the most popular Generalized Ternary Gates. We have shown that ternary quantum circuits can be constructed in a better optimized way using the new gates. We hope that these gates will attract the interest of the researchers in this field.

We also proposed an EA based synthesis method of ternary Quantum Circuit using the new gates. Experimental results are presented and it goes in favor of our claims.

The main contributions of this thesis are summarized as follows:

1. To construct multiple-valued quantum computers, it is strongly required that the gates will be built using quantum primitives. In this thesis we have proposed a family of ternary 2×2 quantum primitive gates. This family of gates is universal, i.e. any ternary function can be realized using these gates

only. Besides, these gates can directly be realized using quantum technology. We have shown the effectiveness of using the new gate as the building blocks of ternary quantum computers. It is also shown that the new gates outperform other ternary primitive gates.

2. The most popular ternary primitive gate today is the Generalized Ternary Gates (GTG). The proposers of GTG claims that these gates can be constructed directly using quantum technology in linear ion trap. We, in this thesis, are raising a strong doubt about this claim. We have shown that all the GTG gates cannot be realized directly in linear ion trap. However, these gates can be constructed using our proposed gates.
3. An EA based synthesis of ternary logic using the new gates has been proposed. Since there is no direct method of synthesizing ternary logic using the new gates is known, and the structure of the problem is still undefined, we used EA for solving the problem. However, we believe that researchers in the field of multiple-valued logic will be interested in developing heuristic or deterministic methods of synthesizing ternary logic using the new gates.
4. An extensive study on the effect of EA parameters in solving this type of problems is carried out. Through the experimentations we have identified the range of different EA parameters for which the EA produces better solutions.
5. Other popular ternary reversible gates can be constructed using the new gates. For example, ternary Toffoli gate, Generalized Ternary Toffoli gates, ternary swap gate, etc. These non-primitive gates are widely used in ternary logic synthesis. So construction of those gates in a cost effective way is a burning question. Use of the new gates in constructing the complex gates will help us to reduce the cost of quantum circuit effectively. Therefore, realizing the non-primitive quantum gates using the new gates carries huge significance.
6. Finally, this thesis can be considered as a comprehensive collection of information relevant to Quantum Computation, Multiple-Valued Logic,

Reversible Logic, and Evolutionary Algorithms. Along with theoretical fundamentals, the recent trend of research and development in multiple-valued logic, specifically ternary logic, is discussed here. Therefore, we believe that this thesis will go a long way in research and development in this field.

7.2 Recommendations for Future Work

In order to design and develop quantum computers in an efficient way, we believe that, there are a number of areas that require further study and research. Actually this is a very new and promising research field. So there are lots of scopes to conduct research in this field. Moreover, this thesis has some limitations that can also be investigated further. Such as EA takes long time to find solutions, therefore, in order to overcome that, different strategies could be sought to make it faster.

This section essentially provides some pointers to pursue further investigation in this field.

1. In this thesis we were confined only with ternary logic. Other Multiple-Valued cases can be taken into account. The mathematical foundation of Ternary Reversible Logic lies in GF3. Other Galois Fields like GF5, GF7, etc. can be considered. Developing quantum primitive operations and logic gates under those Galois Fields can be an excellent field of research. Realizing higher values Galois Field Logic means storing more information in a single unit; thus reducing the size of circuit. On the other hand, measuring and manipulating the information in this case will become more complex. So, it requires a trade off. Through investigation in realization of Multiple Valued logic for higher values are of great importance.
2. Study can be carried on developing generalized rules and synthesis process instead of remain confined with ternary logic. General formulation of the quantum primitive operations and the logic gates would be quite interesting and challenging as well.

3. In this thesis we proposed a set of quantum primitive gates. And it is a set of universal quantum gates. But we did not investigate if there is a subset or not which is also universal. If there exist any such subset of universal gates, then with this less number of gates, perhaps logic synthesis will be much easier. Therefore, study and research can be carried to find the subset of universal gates if there is any.
4. A limited extent of interference could be allowed to apply to the Evolutionary Algorithm so that it can get rid of stagnation and converges towards a solution. For example, if the output vector fitness of the best individual remains less than but very close to the fitness threshold for a long time, then some sorts of “Genetic Engineering” to make the individual achieve the desired fitness threshold. Research can be done on developing suitable “Genetic Engineering” technique. In fact we are, at present, trying to develop one such method.
5. We have used EA based method for logic synthesis using the new gates. Application of other such strategies (especially AI techniques) for doing the same can be developed. To do that we need a better structure and formalization of the problem. This is definitely going to be quite difficult, but we believe that is not impossible. So finding deterministic or heuristic methods can be a good ground of research.
6. The proposed EA based could be applied together with other synthesis methods. For example, the quantum circuit generated by other methods can be transformed into the genotype of our proposed EA based method. Then this genotype can be used as the initial seed of the EA so that it can find a better solution afterwards. Formulating this type of hybrid methods could be done.
7. Finally, study and research could be carried to realize popular ternary gates using the new gates in a better way.

Appendix A

Source Code of the Program

File Name: evTGF.cpp

```
// EA- using ranking of individuals
#include<iostream>
#include<fstream>
#include<vector>
#include <stdlib.h>
#include <math.h>
#include <time.h>

using namespace std;

class charVect;
class char2Dvect;
class floatVect;
class float2Dvect;
class chromosome;
class population;

//*****
char getShift(char A, char B, int x, int y);
void remlt(chromosome& chr, charVect& mate);
int isn(const charVect& cv, char val, int notPos);
int MarriageMatching(float2Dvect wbp, charVect& mate);
int takeCarceOfDuplicate(const population& pop, population& offsp);
int chk_N_replace(population& pop, floatVect& popFit,
                  population& offsp, floatVect& offFit);

//*****
class Param
{
    static int inLength;      // input length - number of variable
    static int outLength;    // number of outputs
    static float Pc;         // cross over probability
    static float Pm;         // mutation probability;
    static int NCmax; // maximum number of generation in one repetition
    static int Lpop;         // population size
    static int rep;          // maximum number of repetition
    static int gen;          // maximum number of generation to wait to exhaust
    static char tC;          // crossover type; u-uniform, o-1 point, t-2 point

public:
    static void read(ifstream& fin)
    { fin>>inLength>>outLength>>Pc>>Pm>>NCmax>>Lpop>>rep>>gen>>tC;};
    static void write(ofstream& fout)
    { fout<<inLength<<endl<<outLength<<endl<<Pc<<endl<<Pm<<endl
    <<NCmax<<endl<<Lpop<<endl<<rep<<endl<<gen<<endl<<tC<<endl;};
    static void input() {cin>>inLength>>outLength>>Pc>>Pm>>NCmax>>Lpop;}
```

```

static void output() {cout<<" "<<inLength<<" "<<outLength<<" "<<Pc
<<" "<<Pm<<" "<<NCmax<<" "<<Lpop;}
static void inputGP(ifstream& fin) // Reads Global Parameters
{
    fin>>Pc>>Pm>>NCmax>>Lpop>>rep>>gen>>tC;
    switch(tC)
    {
        case 'u':
        case 'U':
            cout<<"\nUniform CrossOver..."; break;
        case 'o':
        case 'O':
            cout<<"\nOne Point CrossOver..."; break;
        case 't':
        case 'T':
            cout<<"\nTwo Point CrossOver..."; break;
        default:
            cout<<"\n!!!\V\W\UNKNOWN CrossOver...";
    }
}
static void inputFP(ifstream& fin){fin>>inLength>>outLength;}
//Reads Function specific Parameters
static void outputGP(ofstream& fout)
{fout<<"\n "<<Pc<<"\n "<<Pm<<"\n "<<NCmax<<"\n "<<Lpop<<"\n "
<<rep<<"\n "<<gen<<"\n "<<tC <<"\n/AtPc\lPm\lNCmax\lLpop\lRep\lGen\lrs'fyp";}
static void outputFP(ofstream& fout){fout<<" "<<inLength<<" "<<outLength;}
static void set(int a, int b, float c, float d, int e)
{inLength = a;outLength=b;Pc=c;Pm=d;NCmax=e;}
static void get(int& a, int& b, float& c, float& d, int& e)
{a=inLength;b=outLength;c=Pc;d=Pm;e=NCmax;}
static int inLen() {return inLength;}
static void setInLen(int l) {inLength = l;}
static int getInLen() {return inLength;}
static int outLen() {return outLength;}
static int getLpop() {return Lpop;}
static float getPc() {return Pc;}
static float getPm() {return Pm;}
static int getNCmax() {return NCmax;}
static int Rep() {return rep;}
static int Gen() {return gen;}
static char Tc() {return tC;}
static int maxCrLen() //calc & returns: max length of chromosome
{int mcl; mcl = 3*pow(3,inLength+2); return mcl;}
static int constLines() // calc & returns: number of constant input lines
{ return (inLength+outLength)*9; }
}; // End of class Param.

int Param::inLength; // input length - number of variable
int Param::outLength; // number of outputs
float Param::Pc; // cross over probability
float Param::Pm; // mutation probability;
int Param::NCmax; // maximum number of generation
int Param::Lpop; // population size
int Param::rep;
int Param::gen;
char Param::tC;

//*****
class floatVect
{

```

```

vector<float> fv;
public:
void clear() {fv.clear();}
void push_back(float c) {fv.push_back(c);}
int size()const {return fv.size();}
const float& operator[](int pos) const {return fv[pos];}
float& operator[](int pos) {return fv[pos];}
int operator==(floatVect& c)const {int i,n=fv.size();if(n!=c.size())return 0;
    for(i=0;i<n;i++){if(!(fv[i]==c.fv[i])) return 0;}return 1;}
void output();
bool empty() {return fv.empty();}
void insert(int id, float fv) {fv.insert(fv.begin()+id,fv);}
void pop_back() {fv.pop_back();}
void read(ifstream& fin)
{
    int i,n;
    float t;
    fv.clear();
    fin>>n;
    for(i=0;i<n;i++)
    {
        fin>>t;
        fv.push_back(t);
    }
}
void write(ofstream& fout)
{
    int i,n;
    n = fv.size();
    fout<<n<<"\t";
    for(i=0;i<n;i++)
        fout<<fv[i]<<endl;
}

}; // END class floatVect

//*****
class float2DVect
{
    vector<floatVect> fmat;
public:
void clear() {fmat.clear();}
void push_back(floatVect c) {fmat.push_back(c);}
int size()const {return fmat.size();}
const floatVect& operator[](int pos) const {return fmat[pos];}
floatVect& operator[](int pos) {return fmat[pos];}
int operator==(float2DVect& c)const {int i,n=fmat.size();if(n!=c.size())return 0;
    for(i=0;i<n;i++){if(!(fmat[i]==c.fmat[i])) return 0;}return 1;}
void output();
void read(ifstream& fin)
{
    int i,n;
    floatVect t;

    fmat.clear();
    fin>>n;
    for(i=0;i<n;i++)
    {
        t.read(fin);
        fmat.push_back(t);
    }
}
}

```

```

    }
    void write(ofstream& fout)
    {
        int i,n;
        n = fmat.size();
        fout<<n<<endl;
        for(i=0;i<n;i++)
        {
            fmat[i].write(fout);
            fout<<endl;
        }
    }
}; //END class float2Dvect

//*****
class gene
{
    int ctrling;
    int ctrled;
    int x;
    int y;
public:
    void input(){cin>>ctrling>>ctrled>>x>>y;}
    void output(){cout<<"["<<ctrling<<","<<ctrled<<","<<x<<","<<y<<"]";}
    void output(ofstream& outf)
        {outf<<"["<<ctrling<<","<<ctrled<<","<<x<<","<<y<<"]";}
    void randGenc()
    {
        int n,w;
        n = Param::inLen();
        w = n + Param::constLines(); // total number of input/output lines
        do{
            ctrling = rand()% w;
            ctrled = rand()% w;
        }while(ctrling==ctrled);
        x = rand() % 3;
        y = (rand() % 5) + 1;
    }
    void setGene(int a, int b, int c, int d) {ctrling=a;ctrled=b;x=c;y=d;}
    int Cing()const {return ctrling;}
    int Ced()const {return ctrled;}
    void Cing(int c) {ctrling = c;}
    void Ced(int c) {ctrled = c;}
    int X()const {return x;}
    int Y()const {return y;}
    int operator==(const gene& g)const
    {return ((ctrling==g.ctrling)&&(ctrled==g.ctrled)&&(x==g.x)&&(y==g.y));}
    void repair() {if(ctrling==ctrled) x = y = 0;}
    void read(ifstream& fin) { fin>>ctrling>>ctrled>>x>>y; }
    void write(ofstream& fout)
        { fout<<ctrling<<endl<<ctrled<<endl<<x<<endl<<y<<endl; }
}; // END class gene

//*****
class chromosome
{
    vector<gene> chrM;

```

```

public:
    void clear() {chrM.clear();}
    void push_back(gene& g) {chrM.push_back(g);}
    int size() const {return chrM.size();}
    const gene& operator[](int pos) const {return chrM[pos];}
    gene& operator[](int pos) {return chrM[pos];}
    charVect getOneOutput(charVect ipop, int insize) const;
    int getChromoOutput(char2DVect& opVect) const;
    int mutation();
    int operator==(const chromosome& c) const
        {int i,n=chrM.size();if(n!=c.size())return 0;
         for(i=0;i<n;i++){if(!(chrM[i]==c.chrM[i])) return 0;}return 1;}
    void repair() {int i,n=chrM.size();for(i=0;i<n;i++) chrM[i].repair();}
    void eliminateRedundant();
    void chromosome::output();
    void output(ofstream&);
    void unWeed(const charVect&, charVect&, charVect&);
    int erase(int id)
        {if(chrM.erase(chrM.begin()+id, chrM.end())==NULL) return 0; return 1;}
    charVect evaluate(char2DVect&, float&, float&, float&);
    void read(istream& fin)
    {
        int i, n;
        gene t;
        chrM.clear();
        fin>>n;
        for(i=0;i<n;i++)
        {
            t.read(fin);
            chrM.push_back(t);
        }
    }
    void write(ofstream& fout)
    {
        int i, n;
        n = chrM.size();
        fout<<n<<endl;
        for(i=0;i<n;i++)
            chrM[i].write(fout);
    }
}; //END class chromosome

//*****
class population
{
    vector<chromosome> popl;
    floatVect fFit;
    floatVect lFit;
    floatVect wFit;
public:
    float fSum(int i) {return fFit[i]+0.7*lFit[i]+0.3*wFit[i];}
    float fFit(int i) {return fFit[i];}
    float lFit(int i) {return lFit[i];}
    float wFit(int i) {return wFit[i];}

    void init(char2DVect& opVect);
    void output();
    void clear(){popl.clear();}
    void erase(int i) {if(i<popl.size()) popl.erase(popl.begin()+i);}
    void push_back(chromosome& cr) {popl.push_back(cr);}
};

```

```

int size()const{return popl.size();}
const chromosome& operator[](int pos) const {return popl[pos];};
chromosome& operator[](int pos) {return popl[pos];}
population& crossOverUnf();
population& crossOver1pt();
int operator==(const population& c)const
    {int i,n=popl.size();if(n!=c.size())return 0;
    for(i=0;i<n;i++){if(!(popl[i]==c.popl[i])) return 0;}return 1;}
population selectParentT_ary(int T, const floatVect& crFit);
void repairGene() {int i,n=popl.size();for(i=0;i<n;i++) popl[i].repair();};
void swap(int i, int j);
void rankPopl();
int insertWithRank(population&,int&);
void trankWorst(int);
void evaluate(char2DVect&);
void read(ifstream& fin)
{
    int i, n;
    chromosome t;
    popl.clear();
    fin>>n;
    for(i=0;i<n;i++)
    {
        t.read(fin);
        popl.push_back(t);
    }
    fFit.read(fin);
    lFit.read(fin);
    wFit.read(fin);
}
void write(ofstream& fout)
{
    int i, n;
    n = popl.size();
    fout<<n<<endl;
    for(i=0;i<n;i++)
    {
        popl[i].write(fout);
        fout<<endl;
    }
    fFit.write(fout);fout<<endl;
    lFit.write(fout);fout<<endl;
    wFit.write(fout);fout<<endl;
}
};// END class population

//*****
class charVect
{
    vector<char> cvect;
public:
    void clear() {cvect.clear();}
    void push_back(char c) {cvect.push_back(c);}
    int size()const {return cvect.size();}
    const char& operator[](int pos) const {return cvect[pos];};
    char& operator[](int pos) {return cvect[pos];};
    charVect& incComb(int n);
    int allare(char c, int n);
    int operator==(charVect& c)const

```

```

        {int i,n=cvect.size();if(n!=c.size())return 0;
        for(i=0;i<n;i++){if(!(cvect[i]==c.cvect[i])) return 0;}return 1;}
void output(char d='0')
    {int i,n=cvect.size();for(i=0;i<n;i++) cfout<<char(cvect[i]+ d)<<" ";}
floatVect evaluate(char2Dvect& opv)const;
void output(ofstream& outf, char ch = '#');
void outputChar(ofstream& outf, char ch = '#');
void outputChar(char ch = '#');
void input(ifstream& inf, char ch = '#');
int count(char ch)
{
    int i,n=cvect.size();
    int c=0;
    for(i=0;i<n;i++)
        if(cvect[i]==ch)
            c++;
    return c;
}
void read(ifstream& fin)
{
    int i, n;
    char t;
    cvect.clear();
    fin>>n;
    for(i=0;i<n;i++)
    {
        fin>>t;
        cvect.push_back(t-'0');
    }
}
void write(ofstream& fout)
{
    int i, n;
    n = cvect.size();
    fout<<n<<"\t";
    for(i=0;i<n;i++)
        fout<<(cvect[i]+'0')<<ends;
}

}; //END class charVect

//*****
class char2Dvect
{
    vector<charVect> cmat;
public:
    void clear() {cmat.clear();}
    void push_back(charVect& cv) {cmat.push_back(cv);}
    void output();
    void output(ofstream&);
    void outputVect(ofstream&);
    int size(){return cmat.size();}
    const charVect& operator[(int pos) const {return cmat[pos];};}
    charVect& operator[(int pos) {return cmat[pos];}
    int operator==(char2Dvect& c)const
        {int i,n=cmat.size();if(n!=c.size())return 0;
        for(i=0;i<n;i++){if(!(cmat[i]==c.cmat[i])) return 0;}return 1;}
    float2Dvect evaluate(char2Dvect& opv)const;
    void write(ofstream& fout)
    {
        int i, n;

```

```

        n = cmat.size();
        fout<<n<<"\n";
        for(i=0;i<n;i++)
        {
            cmat[i].write(fout);
            fout<<endl;
        }
    }; //END class char2Dvect

//*****
void floatVect::output()
{
    int i,n=fv.size();
    for(i=0;i<n;i++)
        cfout<<fv[i]<<" ";
}

void float2Dvect::output()
{
    int i,j,ni=fmat.size(),nj=fmat[0].size();
    for(i=0;i<ni;i++)
    {
        cfout<<endl;
        for(j=0;j<nj;j++)
            cfout<<fmat[i][j]<<ends;
    }
}

void char2Dvect::output()
{
    int i,j;
    for(i=0;i<cmat.size();i++)
    {
        cfout<<endl;
        for(j=0;j<cmat[i].size();j++)
            cfout<<char(cmat[i][j]+'0')<<ends;
    }
}

void char2Dvect::output(ofstream& outf)
{
    int i,j;
    for(i=0;i<cmat.size();i++)
    {
        outf<<endl;
        for(j=0;j<cmat[i].size();j++)
            outf<<char(cmat[i][j]+'0')<<ends;
    }
}

void char2Dvect::outputVect(ofstream& outf)
{
    int i;
    for(i=0;i<cmat.size();i++)
    {
        outf<<endl;
        cmat[i].output(outf);
    }
}

```



```

void population::evaluate(char2Dvect& opVect)
{
    int i, n=popl.size();
    float ff, lf, wf;
    fFit.clear();
    lFit.clear();
    wFit.clear();
    for(i=0;i<n;i++)
    {
        popl[i].evaluate(opVect,ff,lf,wf);
        fFit.push_back(ff);
        lFit.push_back(lf);
        wFit.push_back(wf);
    }
}

void population::init(char2Dvect& opVect)
{
    int popsize = Param::getLpop();
    int i,varlen = Param::getInLen();// varlen - number of variables in the function
    gene g;
    chromosome cr;
    int erlen, mcr, j;
    float ff, lf, wf;
    mcr = Param::maxCrLen();
    popl.clear();
    fFit.clear();
    lFit.clear();
    wFit.clear();
    for(i=0;i<popsize;i++)
    {
        cr.clear();
        erlen = (rand() % mcr) + 1;
        for(j=0;j<erlen;j++)
        {
            g.randGene();
            cr.push_back(g);
        }
        cr.evaluate(opVect, ff, lf, wf);
        popl.push_back(cr);
        fFit.push_back(ff);        lFit.push_back(lf);        wFit.push_back(wf);
    }
    rankPopl();
}

void population::swap(int i, int j)
{
    chromosome tempCr;
    float tff, tlf, twf;
    tempCr = popl[i];        popl[i] = popl[j]; popl[j] = tempCr;
    tff = fFit[i];        fFit[i] = fFit[j]; fFit[j] = tff;
    tlf = lFit[i];        lFit[i] = lFit[j]; lFit[j] = tlf;
    twf = wFit[i];        wFit[i] = wFit[j]; wFit[j] = twf;
}

void population::rankPopl()
{
    int n = population::size();
    int i,j;
}

```

```

for(i=0;i<n-1;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(fFit[i] < fFit[j])
            population::swap(i,j);
        else
            if(fFit[i] == fFit[j])
            {
                if(lFit[i] < lFit[j])
                    population::swap(i,j);
                else
                    if(lFit[i] == lFit[j])
                        if(wFit[i] < wFit[j])
                            population::swap(i,j);
            }
    }
}

int population::insertWithRank(population& pp, int& newBest)
{
    int i,j, n = pp.size();
    int ret = 0;
    int rlst = size();

    newBest = 0; // new best chromosome NOT found
    for(i=0;i<n;i++)
    {
        if(fSum(rlst-1) < pp.fSum(i))
        {
            for(j=0;j<rlst;j++)
            {
                if((pp.fFit[i]>fFit[j])
                    ||((pp.fFit[i]==fFit[j])&&(pp.lFit[i]>lFit[j]))
                    ||((pp.fFit[i]==fFit[j])&&(pp.lFit[i]==lFit[j])&&(pp.wFit[i]>wFit[j])))
                {
                    popl.insert(popl.begin()+j,pp.popl[i]);
                    fFit.insert(j,pp.fFit[i]);
                    lFit.insert(j,pp.lFit[i]);
                    wFit.insert(j,pp.wFit[i]);
                    popl.pop_back();
                    fFit.pop_back();
                    lFit.pop_back();
                    wFit.pop_back();
                    if(j==0)
                        newBest = 1; // new best chromosome found!!!!
                    break;
                }
            }
            ret++;
        }
    }
    return ret;
}

void population::trankWorst(int w = 2)
{
    int i;
    for(i=0;i<w;i++)

```

```

        popl.pop_back();
    }

void population::output()
{
    int i,j,crsize,popsize = popl.size();
    for(i=0;i<popsize;i++)
    {
        crsize = popl[i].size();

        if(fFit.empty())
            cfout<<endl<<crsize <<"\n";
        else
            cfout<<endl<<crsize<<"\t****"<<fFit[i] <<" "<<fFit[i]<<" "
                <<wfFit[i]<<"****\n";
        for(j=0;j<crsize;j++)
            popl[i][j].output();
    }
}

charVect chromosome::evaluate(char2Dvect& opVect, float& ffit, float& lfit, float& wfit)
// calculates the fitness components of a chromosome and returns mate
{
    const int n = Param::inLen();
    float2Dvect funcFit;
    char2Dvect crOP;
    charVect mate;
    int maxCrLn = Param::maxCrLen();
    int i;
    int ni;          // = mate.size(); this is also the number of output - m.

    {
        getChromoOutput(crOP);
        funcFit = opVect.evaluate(crOP);
        MarriageMatching(funcFit,mate);
        ffit = 0.0;
        ni = mate.size();
        for(i=0;i<ni;i++)
        {
            ffit += (funcFit[i][mate[i]]==n)?funcFit[i][mate[i]]+1:funcFit[i][mate[i]];
        }
        float thresh_ffit = (Param::inLen()+1) * Param::outLen();
        charVect gF;    //wccd-crop flags for the genes in chromosome
        charVect wF;    //wccd-crop flags for I/O lines
        int unused, cs;
        cs = unused = Param::constLines();
        if(ffit >= thresh_ffit)
        {
            unWeed(mate, wF, gF);
            for(i=n;i<cs;i++)
                if(wF[i]=='e')
                    unused--;
            // if the I/O line contributes to realize a function, it is used
        }
        else
            unused = 0;
        int crLnP = chromosome::size();
        lfit = 1 - (double(crLnP)/maxCrLn);
        wfit = double(unused)/cs;
    }
}

```

```

return matc;
}

int chromosome::getChromoOutput(char2Dvect& opvect)const
{
    charVect ipcomb;// input combination like ABC01201... for 3 var function
    charVect opcomb,t;
    ipcomb.clear();
    int insize = Param::inLen();
    int constSize = Param::constLines();
    int i;
    for(i=0;i<insize;i++)
        ipcomb.push_back(0);
    for(i=0;i<constSize;i++)
        ipcomb.push_back(i % 3);
    opvect.clear();
    do{
        opcomb = ipcomb;
        t.clear();
        t = getOneOutput(opcomb,insize);
        opvect.push_back(t);
        ipcomb.incComb(insize);
        opcomb.clear();
    }while(!ipcomb.allare(0,insize));
    if(opvect.size() != (pow(3,insize)))
        cout<<"\nDal me kuch kaala hay...";
    return opvect.size();
}

charVect chromosome::getOneOutput(charVect ipop, int insize)const
{
    int i, crsize;
    charVect opop;
    opop = ipop;
    crsize = size();
    for(i=0;i<crsize;i++)
        opop[chrn[i].Ced()] =
        getShifl(opop[chrn[i].Cing()],opop[chrn[i].Ced()],chrn[i].X(),chrn[i].Y());
    return opop;
}

charVect& charVect::incComb(int n)
{
    cvect[n-1]++;
    while((n>0)&&(cvect[n-1]>2))
    {
        cvect[n-1] -= 3;
        if(n>=2)
            cvect[n-2]++;
        n--;
    }
    return *this;
}

int charVect::allare(char c, int n)
{
    while(n>0)
    {
        if(cvect[n-1] != c)

```

```

        return 0;
    }
    n--;
}
return 1;
}

population& population::crossOverUnf()
{
    float Pcross = Param::getPc();
    int i, crlen, m, n;
    float prob;
    gene temp;
    m = popl[0].size();
    n = popl[1].size();
    crlen = (m < n) ? m : n;
    for(i=0; i < crlen; i++)
    {
        prob = float(rand() % 100)/100;
        if(prob > Pcross) continue;
        temp = popl[0][i];
        popl[0][i] = popl[1][i];
        popl[1][i] = temp;
    }
    return *this;
}

int chromosome::mutation()
{
    float Mprob = Param::getPm();
    gene temp;
    int i;
    float prob;
    int mt;
    mt = rand() % 3;
    prob = float(rand() % 100)/100;
    if(prob > Mprob) return 0;
    int crSize = chrm.size();
    if(crSize == 0)
        if(mt == 2)
            i = 0;
        else
            return 0;
    else
        i = rand() % (crSize);

    switch(mt)
    {
    case 0://delete the gene; all enjoy same probability
        if(chrm.size() > 1)
            chrm.erase(chrm.begin()+i);
        break;

    case 1://modify
        temp.randGene();
        chrm[i] = temp;
        break;

    case 2://insert
        if(chrm.size() < (Param::maxCrLen()))
        {
            temp.randGene();

```

```

        chrm.insert(chrm.begin()+i,temp);
    }
    }
    return l;
}

floatVect charVect::evaluate(char2Dvect& opv)const
{
    floatVect cc;
    int i,j,k,ctr=0,a, fail;
    float cval=0.0;
    int w = opv[0].size();
    int l = opv.size();
    int n = Param::inLen();
    for(i=0;i<w;i++)
    {
        cval = 0.0;
        for(k=0;k<n;k++)
        {
            ctr = 0;
            for(j=0;j<l;j+=pow(3,k))
            {
                fail = 0;
                for(a=j;a<j+pow(3,k);a++)
                {
                    if(opv[a][i] != cvect[a])
                    {
                        fail = 1;
                        break;
                    }
                }
                if(!fail) ctr++;
            }
            cval += ctr/pow(3,(n-k));
        }
        cc.push_back(cval);
    }
    return cc;
}

float2Dvect char2Dvect::evaluate(char2Dvect& opv)const
{
    float2Dvect ct;
    floatVect cc;
    int i,n=cmat.size();
    for(i=0;i<n;i++)
    {
        cc = cmat[i].evaluate(opv);
        ct.push_back(cc);
    }
    return ct;
}

void charVect::output(ofstream& outf, char ch)
{
    int i,n=cvect.size();
    for(i=0;i<n;i++)
    {
        if(!(i%3)) outf<<"\t";
        outf<<int(cvect[i])<<ends;
    }
}

```

```

    }
    outf<<ends<<ch;
}

void charVect::outputChar(ofstream& outf, char ch)
{
    int i,n=cvect.size();
    for(i=0;i<n;i++)
    {
        if(!(i%3)) outf<<ends;
        outf<<cvect[i];
    }
    outf<<ends<<ch;
}

void charVect::outputChar(char ch)
{
    int i,n=cvect.size();
    for(i=0;i<n;i++)
    {
        if(!(i%3)) cout<<ends;
        cout<<cvect[i];
    }
    cout<<ends<<ch;
}

void charVect::input(ifstream& inf, char ch)
{
    char cb = ch+1;//make sure cb != ch
    cvect.clear();
    inf>>cb;
    while(cb != ch)
    {
        cvect.push_back(cb - '0');
        inf>>cb;
    }
}

population population::selectParentT_ary(int T, const floatVect& crFit)
{
    population offsp;
    chromosome cr;
    int i,j,id,maxID, prevID;

    prevID = -1; // to ensure that the parents are different.
    for(i=0;i<2;i++)
    {
        maxID = Param::getLpop();
        for(j=0;j<T;j++)
        {
            id = rand()%Param::getLpop();
            if(id == prevID) continue;
            if(id < maxID)
            {
                maxID = id;
            }
        }
        cr = popl[maxID];
        prevID = maxID;
        offsp.push_back(cr);
    }
}

```

```

    }
    return offsp;
}

void chromosome::output()
{
    int i,n = chrm.size();
    for(i=0;i<n;i++)
        chrm[i].output();
}

void chromosome::output(ofstream& outf)
{
    int i,n = chrm.size();
    for(i=0;i<n;i++)
        chrm[i].output(outf);
}

void chromosome::unWeed(const charVect& mate, charVect& wFlag, charVect& gFlag)
{
    int n = Param::inLen();
    int w = n + Param::constLines();
    int m = mate.size();
    int s = size();
    int i;
    wFlag.clear();
    for(i=0;i<w;i++)
        wFlag.push_back('w');
    gFlag.clear();
    for(i=0;i<s;i++)
        gFlag.push_back('w');
    for(i=0;i<m;i++)
        wFlag[mate[i]] = 'c';
    /*****Algorithm*****/
    // w - for weed, c - for crop
    // for all i [=n,n-1,...,1]
    // if Gene[i].controlled is a crop then
    //     Gene[i] is a crop
    //     Gene[i].controlling is a crop
    // else
    //     Gene[i] is weed
    //     Gene[i].controlling as it was
    // endif
    /*****Algorithm End*****/
    for(i=s-1;i>=0;i--)
    {
        if(wFlag[chrm[i].Ced()]=='c')
        {
            gFlag[i]='c';
            wFlag[chrm[i].Cing()] = 'c';
        }
        else
        {
            gFlag[i] = 'w';
        }
    }
    for(i=0;i<s;i++)
    {
        int cicing = chrm[i].Cing();
        int ki, fl;
    }
}

```



```

        if(cicing >= n)
        {
            fl = 1;
            for(ki=0;ki<i;ki++)
            {
                if((gFlag[ki]=='c')&&(chrn[ki].Ced()==cicing))
                {
                    fl = 0;
                    break;
                }
            }
            if(!fl) continue;
            int constLineValue = (cicing - n) % 3;
            if(chrn[i].X() != constLineValue)
                gFlag[i] = 'w';
        }
    }
    for(i=s-1;i>=0;i--)
    {
        if(gFlag[i]=='w')
            chrn.erase(chrn.begin()+i);
    }
}

char getShift(char A, char B, int x, int y)
{
    if(int(A) != x)
        return B;
    return (((((v/3)+1) * B) + (y % 3)) % 3);
}

charVect Evaluate(population& popu, const char2Dvect opVect, floatVect& crFit)
// calculates the fitness of each chromosome and returns the mate
{
    const int n = Param::inLen();
    float2Dvect funcFit;
    char2Dvect crOP;
    charVect mate;
    int p, maxCrLn = Param::maxCrLen();
    const int np = popu.size();
    int i;
    int ni;// = mate.size(); this is also the number of output - m.
    float ffit, lfit, wfit,totFit;
    crFit.clear();
    for(p=0;p<np;p++)// generate the fitness table and find mates
    {
        popu[p].getChromoOutput(crOP);
        funcFit = opVect.evaluate(crOP);
        MarriageMatching(funcFit,mate);
        charVect gF;    //weed-crop flags for the genes in chromosome
        charVect wF;    //weed-crop flags for I/O lines
        popu[p].unWeed(mate, wF, gF);
        ffit = 0.0;
        ni = mate.size();
        for(i=0;i<ni;i++)
        {
            ffit +=
                (funcFit[i][mate[i]]==n)?funcFit[i][mate[i]]+1:funcFit[i][mate[i]];
        }
        int crLnP = popu[p].size();
    }
}

```

```

    lfit = 1 - (double(crLnP)/maxCrLn);
    int unused, cs;
    cs = unused = Param::constLines();
    for(i=n;i<cs;i++)
        if(wF[i]=='c')
            unused--;
        // if the I/O line contributes to realize a function, it is used
    wfit = double(unused)/cs;
    totFit = ffit + 0.7*lfit + 0.3*wfit;
    crFit.push_back(totFit);
}
return mate;
}

float max(floatVect& fv)
{
    int i,mi=0,n = fv.size();
    float m = fv[0];
    for(i=1;i<n;i++)
    {
        if(fv[i]>m)
        {
            m = fv[i];
            mi = i;
        }
    }
    return m;
}

int maxID(floatVect& fv)
{
    int i,mi=0,n = fv.size();
    float m = fv[0];
    for(i=1;i<n;i++)
    {
        if(fv[i]>m)
        {
            m = fv[i];
            mi = i;
        }
    }
    return mi;
}

float min(floatVect& fv)
{
    int i,mi=0,n = fv.size();
    float m = fv[0];
    for(i=1;i<n;i++)
    {
        if(fv[i]<m)
        {
            m = fv[i];
            mi = i;
        }
    }
    return m;
}

int minID(floatVect& fv)

```

```

{
    int i,mi=0,n = fv.size();
    float m = fv[0];
    for(i=1;i<n;i++)
    {
        if(fv[i]<m)
        {
            m = fv[i];
            mi = i;
        }
    }
    return mi;
}

int isIn(const charVect& cv, char val, int notPos)
{
    int i,n=cv.size();
    for(i=0;i<n;i++)
    {
        if(i == notPos) continue;
        if(cv[i] == val)
            return i;
    }

    return -1;
}

int MarriageMatching(float2DVect wbpj, charVect& mate)
{
    int i,j,n;
    n = wbpj.size();
    mate.clear();
    for(i=0;i<n;i++)
        mate.push_back('?');
    for(i=0;i<n;i++)
    {
        if(mate[i]!='?') continue;
        mate[i] = maxID(wbpj[i]);
        j = isIn(mate,mate[i],i);
        if(j != -1)
        {
            if(wbpj[i][mate[i]] > wbpj[j][mate[j]])
            {
                wbpj[j][mate[j]] = 0.0;
                mate[j] = '?';
                i = -1;
            }
            else
            {
                wbpj[i][mate[i]] = 0.0;
                mate[i] = '?';
                i = -1;
            }
        }
    }
    return 0;
}

char* nextPreamble(ifstream& inf, char* pr)
{

```

```

char tpr[100];
do{
  inf >> tpr[0];
  }while(tpr[0] != '.');
inf >> tpr;
if(tpr[0] == '.')
  strcpy(pr, " ");
else
  strcpy(pr,tpr);
return pr;
}

int process(char2Dvect& opVect, chromosome& maxCr,
           float& maxFit, char* functionName)
{
  ofstream resOut;

  ofstream fout;
  population pop,parent_N_offsp;
  float2Dvect funcFit;
  floatVect popFit,offFit;
  int S;
  int withInit = 1;
  int notImproved,ctr=0,rep = 0;
  int mID;
  int wfCtr;
  int replaced;
  int newBestFound;
  int solutionFound = 0;
  int maxCtr = Param::getNCmax();
  float fitnessThreshold = Param::outLen()*(Param::inLen()+1);
  ofstream tmpres;
  char convFName[50];
      char paraTemp[10];
      strcpy(convFName,functionName);
      _itoa(int(Param::getPc()*1000),paraTemp,10);
      strcat(convFName,"_");
      strcat(convFName,paraTemp);
      _itoa(int(Param::getPm()*1000),paraTemp,10);
      strcat(convFName,"_");
      strcat(convFName,paraTemp);
      _itoa(int(Param::getLpop()),paraTemp,10);
      strcat(convFName,"_");
      strcat(convFName,paraTemp);
      strcat(convFName,".conv");

  if(resume)
  { // resume process;
    cout<<"\nProgram Resuming...";
    ifstream resumeFile("progstat.sav");
    resumeFile>>solutionFound>>notImproved>>ctr>>rep>>mID
      >>wfCtr>>replaced>>maxFit;

    Param::read(resumeFile);
    popFit.read(resumeFile);
    offFit.read(resumeFile);
    funcFit.read(resumeFile);
    parent_N_offsp.read(resumeFile);
    pop.read(resumeFile);
    maxCr.read(resumeFile);
    opVect.read(resumeFile);
    resumeFile.close();
  }
}

```



```

opVect.write(resumeFile); resumeFile<<endl;
resumeFile.close();
}
if(newBestFound)
{
maxFit = pop.FFit(0);
maxCr = pop[0];
wfCtr = ctr+1;
if(!solutionFound) && (maxFit >= fitnessThreshold)
// the first time a solution is found
{
solutionFound = 1;
char resFName[50];
char paraTemp[10];
strcpy(resFName,functionName);
_itoa(int(Param::getPc()*1000),paraTemp,10);
strcat(resFName,"_");
strcat(resFName,paraTemp);
_itoa(int(Param::getPm()*1000),paraTemp,10);
strcat(resFName,"_");
strcat(resFName,paraTemp);
_itoa(int(Param::getLpop()),paraTemp,10);
strcat(resFName,"_");
strcat(resFName,paraTemp);
strcat(resFName,".rsit");
resOut.open(resFName, ios::app);
float d1,d2,d3;
charVect pmate, dg, dw;
pmate = maxCr.evaluate(opVect,d1,d2,d3);
maxCr.unWeed(pmate,dw,dg);
resOut<<"\n/** The First Found Solution...**/";
resOut<<"\n.size "<<maxCr.size()<<endl;
resOut<<"\n.solution ";
maxCr.output(resOut);
resOut<<"\nMate : "; pmate.output(resOut,0);
resOut<<"\nwFlag["<<dw.count('c')<<"/"<<dw.size()<<"]: ";
dw.outputChar(resOut,0);
resOut<<"\ngFlag["<<dg.count('c')<<"/"<<dg.size()<<"]: ";
dg.outputChar(resOut,0);
resOut<<"\n.generation "<<wfCtr;
resOut<<"\n_____ \n/** And The Final Solution **\n";
resOut.close();
}
}
if(!replaced)
notImproved++;
else
notImproved = 0;

ctr++;
//*****
// Logic for controlling this Loop
// c1 -> (rcp < R)
// c2 -> (ctr < maxCtr)
// c3 -> (notImproved >= S)
// c4 -> (maxFit < fitnessThershold)
//
// c2 c3 c4 |c1=0 |c1=1 |
//-----
// 0 0 0 | D | D |D: Done
// 0 0 1 | F | 1 |F: Fail

```

```

//      0      1      0      |      D      |      D      |1: With Init
//      0      1      1      |      F      |      1      |2: With OUT Init
//      1      0      0      |      D      |      2      |
//      1      0      1      |      F      |      2      |
//      1      1      0      |      D      |      D      |
//      1      1      1      |      F      |      1      |
//*****

int c1, c2, c3, c4;
S = (float(maxFit)*Param::Gen()*Param::inLen())/fitnessThreshold;
c1 = (rep < Param::Rep()/* R*/);
c2 = (ctr < Param::getNCmax()/*maxCtr*/);
c3 = (notImproved >= S);
c4 = (maxFit < fitnessThreshold);
//conditions for Done c3~c4+~c1~c4+~c2~c4
if((c3 && !(c4))||(!c1 && !c4)||(!c2 && !c4))
{
    maxCr = pop[0];
    return wfCtr;
}
//conditions for Fail ~c1c4
if(!c1 && c4)
{
    return 0;
}
//conditions for 1 c1c3c4+c1~c2c4
if((c1 && c3 && c4)||(!c1 && !c2 && c4))
{
    withInit = 1;
    continue;
}
//conditions for 2 c1c2~c3
if(c1 && c2 && !c3)
{
    withInit = 0;
    continue;
}
// else ... if none of above ... not possible
cout<<"\n\tc1 c2 c3 c4 \n\t"<<c1<<ends<<c2<<ends<<c3<<ends<<c4;
cout<<"\nImpossible combination FAILED!!!";
break;
}
}

int takeCareOfDuplicate(const population& pop, population& offsp)
{
    int i,j,p = Param::getI.pop();
    int rem = 0;
    for(i=0;i<2;i++)
    {
        for(j=0;j<p;j++)
        {
            if(offsp[i].size() == pop[j].size())
            if(offsp[i]==pop[j])
            {
                offsp.erase(i);
                rem++;
                break;
            }
        }
    }
}

```

```

    }
    return rem;
}

int chk_N_replace(population& pop, floatVect& popFit,
                 population& offsp, floatVect& offFit)
{
    int mID;
    if(offsp.size() == 0) return 0;
    if(offsp.size() == 1)
    {
        mID = minID(popFit);
        if(offFit[0] > popFit[mID])
        {
            pop[mID] = offsp[0];
            popFit[mID] = offFit[0];
        }
        return 1;
    }
    if(offFit[0] > offFit[1])
    {
        float tf;
        chromosome tc;
        tf = offFit[0];
        offFit[0] = offFit[1];
        offFit[1] = tf;
        tc = offsp[0];
        offsp[0] = offsp[1];
        offsp[1] = tc;
    }

    mID = minID(popFit);
    if(offFit[1] < popFit[mID]) return 0;
    if(offFit[0] > popFit[mID])
    {
        pop[mID] = offsp[0];
        popFit[mID] = offFit[0];
    }
    mID = minID(popFit);
    if(offFit[1] > popFit[mID])
    {
        pop[mID] = offsp[1];
        popFit[mID] = offFit[1];
    }
    return 1;
}

```



```

int main()
{
    cout<<"\nResume?[1/0].."; cin>>resume;
    ifstream fin;
    ofstream fout;
    char preamble[20];
    char name[20];

    charVect tev;
    char2DVect opVect;
    chromosome result;
    fin.open("param.in");
    Param::inputGP(fin);
    fin.close();
    fin.open("input.in");
    do{
        nextPreamble(fin,preamble);
    }while(strcmp(preamble,"begin"));
    while(1)
    {
        nextPreamble(fin,preamble);
        if(!strcmp(preamble,"end")) break;
        if(!strcmp(preamble,"function")) fin>>namef;
        if(!strcmp(preamble,"param"))
        {
            Param::inputFP(fin);
            opVect.clear();
            for(int i=0;i<Param::outLen();i++)
            {
                nextPreamble(fin,preamble);
                if(strcmp(preamble,"vector"))
                {cfout<<"\nERROR reading input.in - unknown file format";
                return 1;}
                tev.input(fin);
                opVect.push_back(tev);
            }
            int resF = 0;
            if(resume)
            {
                ifstream resT("progstat.sav");
                resT>>resF;
                resT.close();
            }
            char resFileName[50];
            char paraTemp[10];
            strcpy(resFileName,namef);
            _itoa(int(Param::getPc()*1000),paraTemp,10);
            strcat(resFileName,"_");
            strcat(resFileName,paraTemp);
            _itoa(int(Param::getPm()*1000),paraTemp,10);
            strcat(resFileName,"_");
            strcat(resFileName,paraTemp);
            _itoa(int(Param::getLpop()),paraTemp,10);
            strcat(resFileName,"_");
            strcat(resFileName,paraTemp);
            strcat(resFileName,".rslt");
            if(!resF)
            {
                fout.open(resFileName);
                fout<<".begin\n";
            }
        }
    }
}

```

```

        fout.close();
    }

    float fitness;
    srand(1);
    int n = process(opVect, result, fitness, namef);
    fout.open(resFileName, ios::app);
    fout<<"\n_____";
    fout<<"\n.function "<<namef;
    fout<<"\n.rem\tinput\toutput";
    fout<<"\n.param\t"<<Param::inLen()<<"\t"<<Param::outLen()<<endl;
    opVect.outputVect(fout);

if(n)
{
    float d1,d2,d3;
    charVect mate, dg, dw;
    mate = result.evaluate(opVect,d1,d2,d3);
    result.unWeed(mate,dw,dg);
    fout<<"\n.size "<<result.size()<<endl;
    fout<<"\n.solution ";
    result.output(fout);
    fout<<"\nMate : "; mate.output(fout,0);
    fout<<"\nwFlag["<<dw.count('c')<<"/"<<dw.size()<<"] : ";
    dw.outputChar(fout,0);
    fout<<"\ngFlag["<<dg.count('c')<<"/"<<dg.size()<<"] : ";
    dg.outputChar(fout,0);
    fout<<"\n.generation "<<n;
    fout<<"\n.fitness "<<fitness<<endl;
    fout<<"Param: ";
    Param::outputGP(fout);
}

else
    fout<<"\n\t\tFailed !!!";

    fout<<"\n_____";
    fout<<"\n.end";
    fout.close();
}

}
return 0;
}

```

Appendix B

Description of the Benchmark functions

prod n : input $x_0 x_1 \dots x_{n-1}$; output $y = (x_0 x_1 \dots x_{n-1}) \bmod 3$. [Output is the GF3 product of n input variables.]

sum n : input $x_0 x_1 \dots x_{n-1}$; output $y = (x_0 + x_1 + \dots + x_{n-1}) \bmod 3$. [Output is the GF3 sum of n input variables.]

ncyr: input $x_0 x_1 \dots x_{n-1}$; output $y = \left[\sum_{i=0}^{n-1} \left(\prod_{j=0}^{r-1} x_{(i+j) \bmod n} \right) \right] \bmod 3$. [A ternary GFSOP

function of n input variables, where the products consist of r input variables in cyclic order. Example: For 3cy2, $y(a, b, c) = ab + bc + ca$.]

sqsum n : input $x_0 x_1 \dots x_{n-1}$; output $y = (x_0^2 + x_1^2 + \dots + x_{n-1}^2) \bmod 3$. [Output is the GF3 sum of squares of n input variables]

avg n : input $x_0 x_1 \dots x_{n-1}$; output $y = \text{int}[(x_0 + x_1 + \dots + x_{n-1})/n] \bmod 3$. [Output is the integer part of the average of n input variables expressed as mod 3 value.]

a2bcc: input a, b, c ; output $y = (a^2 + bc + c) \bmod 3$. [An arbitrary function]

thadd: input a, b ; output $c = \text{int}[(a+b)/3]$, $s = (a+b) \bmod 3$. [Ternary half-adder]

tfadd: input a, b, c ; output $y = \text{int}[(a+b+c)/3]$, $s = (a+b+c) \bmod 3$. [Ternary full-adder]

mul2: input a, b ; output $c = \text{int}[ab/3]$, $m = ab \bmod 3$. [2-trit ternary multiplier]

mul3: input a, b, c ; output $c = \text{int}[abc/3]$, $m = abc \bmod 3$. [3-trit ternary multiplier]

mami4: input a, b, c, d ; output $y = \max(a, b)$, $z = \min(c, d)$. [The output y is the maximum of the inputs a and b ; the output z is the minimum of the inputs c and d .]

Bibliography

- [1] A. Al-Rabadi and M. Perkowski, "Multiple-Valued Galois Field S/D Trees for GFSOP Minimization and their Complexity", Proc. 31st IEEE Int. Symp. on Multiple-Valued Logic, Warsaw, Poland, May 22-24, 2001, pp. 159-166.
- [2] A. Al-Rabadi, "Synthesis and Canonical Representations of Equally Input-Output Binary and Multiple-Valued Galois Quantum Logic: Decision Trees, Decision Diagrams, Quantum Butterflies, Quantum Chrestenson Gate, Multiple-Valued Bell-Einstein-Podolsky-Rosen Basis States", Technical Report #2001/008, ECE Dept., PSU, August 2001.
- [3] A. Al-Rabadi, L. W. Casperson, M. Perkowski and X. Song, "Multiple-Valued Quantum Logic", Booklet of 11th Workshop on Post-Binary Ultra-Large-Scale Integration Systems (ULSI), Boston, Massachusetts, May 15, 2002, pp. 35-45.
- [4] A. Al-Rabadi, "Novel Methods for Reversible Logic Synthesis and Their Application to Quantum Computing", Ph. D. Thesis, PSU, Portland, Oregon, USA, October 24, 2002.
- [5] A. Ashikhmin, and E. Knill, "Non-binary quantum stabilizer codes", <http://citeseer.nj.ncc.com/ashikhmin00nonbinary.html>
- [6] A. De Vos, B. Raa, and L. Storme, "Generating the group of reversible logic gates", Journal of Physics A: Mathematical and General, Vol. 35, 2002, pp. 7063-7078.
- [7] A. Muthukrishnan, and C. R. Stroud Jr., "Multivalued Logic Gates for Quantum Computation", Physical review A, vol. 62, No. 5, 05309/1-8, 2000.
- [8] A. Turing, "On computable numbers with an application to the Entscheidungs-problem," Proc. Lond. Math. Soc. Ser. 2, 42 (1936).
- [9] A. V. Burlakov, M. V. Chekhova, O.V. Karabutova, D.N. Klyshko, and S. P. Kulik, "Polarization state of a biphoton: quantum ternary", Physical Review A, Vol. 60, R4209, 1999.
- [10] C. Bennett, "Logical Reversibility of Computation", IBM Journal of Research and Development, vol. 17, pp. 525-532, 1973.
- [11] C. Darwin, "The Origin of Species", Dent Gordon, London, 1973.

- [12] C. P. Williams, and S. H. Clearwater, "Exploration in Quantum Computing", New York: Springer-Verlag, 1998.
- [13] D. Aharonov, and M. Ben-Or, "Polynomial Simulations of Decohered Quantum Computers", (Online preprint quant-ph/9611029), 37th Annual Symp. on Foundations of Computer Science, Burlington, Vermont, October 1996, pp. 4655.
- [14] D. B. Fogel, "Evolving Artificial Intelligence", Ph.D. Thesis, University of California, San Diego, CA, 1992
- [15] D. B. Fogel, "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence", IEEE Press, Piscataway, NJ, 1995.
- [16] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", In Proceedings of the Royal Society of London, Vol A400 (1985) 97-117.
- [17] D. Deutsch, "Quantum Computational Networks", Proc. Ro. Soc. Lond. A 425, pp. 73-90, 1989.
- [18] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, 1989.
- [19] D. Gottesman, "Fault-tolerant quantum computation with higher-dimensional systems", Chaos, Solitons, Fractals, Vol. 10, No. 10, 1999, pp. 1749-1758.
- [20] D. M. Miller, and R. Drechsler, "On the Construction of Multi-Valued Decision Diagrams", Proc. 32nd IEEE Int. Symp. on Multiple-Valued Logic, Boston, Massachusetts, 2002, pp. 245-253.
- [21] E. Dubrova, and J.C. Muzio, "Generalized Reed-Muller canonical form of a multiple-valued algebra", Multiple-Valued Logic - An International Journal, 1996, pp. 65-84.
- [22] E. M. Rains, "Nonbinary Quantum Codes", IEEE Trans. on Information Theory, Vol. 45, 1999, pp. 1827-1832.
- [23] Gordon E. Moore, "Cramming more components onto integrated circuits", Electronics, Volume 38, Number 8, April 19, 1965.
- [24] H. F. Chau, "Correcting quantum errors in higher spin systems", Physical Review A, Vol. 55, R839-R841, 1997.
- [25] H.-P. Schwefel, "Numerical Optimization of Computer Models", Wiley, Chichester, 1981.
- [26] J. Birnbaum, "Computing Alternatives", Talk given at ACM97, March 3, 1997, San Jose, California.

- [27] J. H. Holland, "Adaptation in Natural and Artificial Systems", MIT Press, Cambridge, MA, 1992.
- [28] J. L. Brylinski and R. Brylinski, "Universal Quantum Gates", (Mathematics of Quantum Computation, CRC Press, 2002) LANL e-print quant-ph/010862.
- [29] K-H. Han, and J-H. Kim, "Quantum Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization", IEEE trans. Evolutionary Computation, 6(6), pp. 580 – 593, 2002.
- [30] I. Rechenberg, "Evolutionsstrategie: Optimicrung Technischer Systeme nach Prinzipien der Biologischen Evolution", Frommann-Holzboog, Stuttgart, 1973.
- [31] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial Intelligence through Simulated Evolution", Wiley, New York, 1966.
- [32] L. Macchiarulo, and P. Civera, "Ternary Decision Diagrams with Inverted Edges and Cofactors – an Application to Discrete Neural Networks Synthesis", Proc. 28th IEEE Int. Symp. on Multiple-Valued Logic, 1998, pp. 58-63.
- [33] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy, "Finding a better-than-classical Quantum AND/OR Algorithm Using Genetic Programming", Proc. of 1999 Congress on Evolutionary Computation, vol. 3, Washington DC. IEEE, Piscataway, NJ, pp. 2239-2246, 6 – 9 July 1999.
- [34] M. H. A. Khan, M. A. Perkowski, and P. Kerntopf, "Multi-Output Galois Field Sum of Products Synthesis with New Quantum Cascades", Proc. 33rd IEEE International Symposium on Multiple-Valued Logic. Tokyo, May 16-19, pp. 146-153, 2003
- [35] M. H. A. Khan, M. A. Perkowski, and M. R. Khan, "Ternary Galois Field Expansions for Reversible Logic and Kronecker Decision Diagrams for Ternary GFSOP Minimization", in Proceedings of the 34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004), Toronto, Canada, 19-22 May 2004.
- [36] M. H. A. Khan, and M. A. Perkowski, "Genetic Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascades of Generalized Ternary Gates", in Proc. of 2004 IEEE Congress on Evolutionary Computation (CEC 2004), Portland, Oregon, USA, 19-23 June 2004.
- [37] M. H. A. Khan, M. A. Perkowski, M. R. Khan, and P. Kerntopf, "Ternary GFSOP Minimization using Kronecker Decision Diagrams and Their Synthesis with Quantum Cascades", Journal of Multiple-Valued Logic and Soft Computing: Special issue to recognize T. Higuchi's contribution to Multiple-Valued VLSI Computing, 2005.

- [38] M. H. A. Khan, "Quantum Realization of Ternary Toffoli Gate", in Proc. of the 3rd International Conference on Electrical and Computer Engineering ICECE 2004, 28-30 December 2004, Dhaka, Bangladesh.
- [39] M. Hirvensalo, "Quantum Computing", Springer Verlag, 2001
- [40] M. Lukac, M. A. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim, "Evolutionary Approach to Quantum and Reversible Circuits Synthesis", *Artificial Intelligence Review*, Kluwer Academic Publishers, 20, pp. 361 – 417, 2003.
- [41] M. A. Nielsen, and I. L. Chuang, "Quantum Computation and Quantum Information", Cambridge University Press, 2000.
- [42] M. A. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Mishchenko, and M. Chrzanowska-Jeske, "Three-Dimensional Realization of Multivalued Functions Using Reversible Logic", Booklet of 10th Int. Workshop on Post-Binary Ultra-Large-Scale Integration Systems (ULSI), Warsaw, Poland, May 2001, pp. 47- 53.
- [43] M. A. Perkowski, A. Al-Rabadi, and P. Kerntopf, "Multiple-Valued Quantum Logic Synthesis", Proc. of 2002 International Symposium on New Paradigm VLSI Computing, Sendai, Japan, December 12-14, 2002, pp. 41-47.
- [44] N. Denler, B. Yen, M. A. Perkowski, and P. Kerntopf, "Synthesis of Reversible Circuits from a Sub-set of Muthukrishnan-Stroud Quantum Multi-Valued Gates", in Proc. IWLS 2004, Tamecula, California, 2-4 June 2004.
- [45] P. Kerntopf, "Maximally efficient binary and multi-valued reversible gates", Booklet of 10th Intl Workshop on Post-Binary Ultra-Large-Scale Integration Systems (ULSI), Warsaw, Poland, May 2001, pp. 55-58
- [46] P. Mazumder, and E. M. Rudnick, "Genetic Algorithms for VLSI Design, Layout & Test Automation", Pearson Education Asia, 2002.
- [47] P. Picton, "A Universal Architecture for Multiple-Valued Reversible Logic", *Multiple-Valued Logic - An International Journal*, Vol. 5, 2000, pp. 27-37.
- [48] Peter. W. Shor, "Algorithms for Quantum Computation: Discrete Log and Factoring", In Proceedings of the 35th Annual symposium on the Foundations of Computer Science (<http://xxx.lanl.gov/abs/quant-ph/9508027>).
- [49] R. Landauer, "Irreversibility and Heat Generation in the Computing Process", *IBM Journal of Research and Development*, vol. 5, No. 3, pp. 183-191, 1961.

- [50] R. P. Feynman, "Simulating Physics with Computers", *International Journal of Theoretical Physics*, Vol 21, Nos 6/7, 1982.
- [51] R. P. Feynman, "Quantum mechanical computers," *Found. Phys.*, 16 (1986), 507.
- [52] T. Bäck, H-P Schwefel, "An overview of evolutionary algorithms for parameter optimization", *Evolutionary Computation* 1(1) pp 1-23, (1993).
- [53] T. Bäck, "Evolutionary Algorithms in Theory and Practice", Oxford University Press, Oxford, 1996.
- [54] T. Toffoli, "Reversible Computing," Tech. Memo MIT/LCS/TM-151, MIT Lab. for Com. Sci. (1980).
- [55] T. Yabuki, and H. Iba, "Genetic Algorithms and Quantum Circuit design, Evolving a Simpler Teleportation Circuit", In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pp. 421 – 425, 2000.
- [56] Y. Z. Ge, L. T. Watson, and E. G. Collins, "Genetic Algorithms for Optimization on a Quantum Computer", In *Unconventional Models of Computation*, London: Springer Verlag, pp. 218 – 227, 1998.
- [57] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs (3rd edn.)", Springer, 1996.
- [58] Michael Gibbs' Home Page, [contains a good description of fundamental issues of Galois Fields] <http://members.aol.com/jmtsgibbs/galois.htm>
- [59] Website of Centre for Quantum Computation (CCC), <http://www.qubit.org/>
- [60] K. Mattle, H. Weinfurter, P.G. Kwiat, and A. Zeilinger, "Dense Coding in Experimental Quantum Communication", *Physical Review Letters*, Vol. 76, 1996, pp. 4656-4659.

