UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES

SIMULATION OF QUANTUM COMPUTERS ON CLASSICAL COMPUTERS BY USING MATHEMATICA

M.SC. THESIS
IN
PHYSIC ENGINEERING

BY
SHAKHAWAN S. ABDULLAH AL-BALAKY
OCTOBER 2012

Simulation of Quantum Computers on Classical Computers by Using Mathematica

M.Sc. Thesis

In

Physic Engineering

University of Gaziantep

Supervisor

Prof. Dr. Ramazan KOÇ

By

Shakhawan S. Abdullah AL-BALAKY

October 2012

REPUBLIC OF TURKEY

UNIVERSITY OF GAZİANTEP

GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES

NAME OF THE DEPARTMENT

Name of the thesis: Simulation of Quantum Computers on Classical Computers by Using Mathematica

Name of the student: Shakhawan Salih Abdullah
Exam date: 19/10/2012

Approval of the Graduate School of Natural and Applied Sciences

[illegible]

Assoc. Prof. Dr. Metin BEDİR
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. A. Necmeddin YAZICI
Head of Department

This is to certify that we have read this thesis and that in our consensus/majority opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Ramazan KOÇ
Supervisor

Examining Committee Members	Signature
Prof. Dr. Hayriye TÜTÜNCÜLER	[illegible]
Prof. Dr. Ramazan KOÇ	[illegible]
Asst. Prof. Dr. Abdullah KABLAN	[illegible]

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

[illegible]

Shakhawan S. A. AL-BALAKY

ABSTRACT

SIMULATION OF QUANTUM COMPUTERS ON CLASSICAL COMPUTERS BY USING MATHEMATICA

ABDULLAH, Shakhawan
M.Sc. in Physic Engineering
Supervisor: Prof. Dr. Ramazan KOÇ
October 2012
76 pages

The main objective of this study is simulation of some quantum algorithms on a classical computer using the Mathematica program. Classical computation has been thoroughly discussed at the beginning to important for achieving a significant part of our work.

The crucial part of this study is basic principles of quantum computation, particularly qubit statistics, quantum algorithms, quantum gates and quantum circuits are discussed in details. Moreover, much light has been shed on CNOT Gate, since it constitutes the core of this work. By using CNOT Gate quantum half-adder circuit and quantum full-adder Circuit, are constructed and they are both a factor for arriving at n-qubit adder quantum circuit. In this thesis as a special case, addition of 4-qubit number on a Mathematica is simulated.

One more section of this study is the use of Mathematica program for designing and obtaining the results of Quantum Circuits, particularly 4-qubit adder. Therefore, many calculating operations have been conducted for this circuit by using this program. Finally, it is logical to argue that Adder/Subtractor Algorithm is the basis of all Algorithms in the Quantum Computation science.

Key Words: Digital System, Logic Gates, Qubit, Quantum Gates, Full Adder.

ÖZ

MATHEMATICA KULLANARAK, KUANTUM KOMPUTER'IN, KLASIK BIR KOMPUTERDE SIMULASYONUNUN YAPILMASI

ABDULLAH, Shakhawan
Yüksek Lisans Fizik Müh. Bölümü
Tez Yöneticisi(leri): Prof. Dr. Ramazan KOÇ
Ekim 2012
76 sayfa

Bu tezin temel amacı, bazı kuantum algoritmaların Mathematica program kullanara klasik bilgisayarlar üzerinde simülasyonunu yapmaktır. Bu simülasyonu başarılı bir şekilde yapabilmek için, tezin başlangıcında klasik hesaplama tartışılmıştır.
Kuantum hesaplamanın temelini oluşturan, özellikle, qubitler, kuantum algoritmalar, kuantum kapılar ve kuantum devreler detaylı olarak tartışılmıştır. Simulasyon devresinin çekirdeğini oluşturan CNOT kuantum kapısının özelliklerine daha çok ışık tutulmuştur. CNOT kapısı kullanılarak yarı toplayıcı ve tam toplayıcı devreler elde edilmiş ve devre n-qubit sayıları toplayacak şekilde genelleştirilmiştir. Bu tezde özel bir durum olarak 4-qubit iki sayının toplanması Mathematica programı yardımıyla simüle edilmiştir.
Bunlara ek olarak, ilave bir bölümde Mathematica programının, quantum başta 4-qubit sayıların toplanması olmak üzere, devre tasarımında kullanımı ele alınmıştır. Böylece birçok kuantum hesaplamanın Mathematica kullanarak simulasyonun yapılabileceği basit bir yol gösterilmiştir. Sonuç olarak, pek çok quantum devrenin temelini oluşturan tam yoplama/çıkarma devresinin çalışılması ve tartışılmasının yerinde olduğu açıktır.

Anahtar Kelimeler: Dijital Sistem, Mantık Kapılar, Qubit, Kuantum kapılarte, Tam toplama devresi.

To the soul of my great father. And to my mother, sisters, brothers and my wife

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor Prof. Dr. Ramazan KOÇ for his guidance, advice, criticism, encouragements and insight throughout the writing of this research work.

My great thanks to go to Dr. Saman Salah Balaky in the UK for his kind assistance with editing the language and structure of some selected parts of my work. I also express my sincere thanks to Mr. Samir Mustafa Ahmad, a PhD student in the USA, and Hayder Mahmud , for their help and support. I am similarly indebted to all my friends and those who have helped me in one way or another.

I am particularly grateful for the constant moral support I received from my family during the course of writing and conducting this research. They were the source of inspiration and continued enthusiasm for me with this study and made me always feel positive with the future of my work.

Finally, my best acknowledgments for those hidden helpers whom I regrettably fail to remember them now and have given support on various levels.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBUL/ABBREVIATION

ABBRE.	FULL TEXT
QC	Quantum Computation
NAND	Not AND
NOR	Not OR
XOR	Exclusive OR
XNOR	Exclusive Not OR
CNOT	Control NOT
CCNOT	Control-Control NOT
CSWAP	Control SWAP
CNTS	(CNOT, NOT, TOFFOLI, and SWAP) gate library
Qubit	Quantum Bit
QMatrix	Quantum Matrix
M	Matrix
M_I	Identity Matrix
M_{SWAP}	SWAP Matrix
M_{CNOT}	Control NOT Matrix
HA	Half Adder
FA	Full Adder
C_o /C_{out}	Carry Out
C_i /C_{in}	Carry In

S	Sum
G	Garbage
CaML	Categorical Abstract Machine Language
MATLAB	Matrix Laboratory

PERSONAL INFORMATION

Name and Surname: Shakhawan S. Abdullah AL-BALAKY

Nationality: Iraqi

Birth place and date: Erbil/ 01.01.1974

Mariel status: Marred

Phone number: Iraq Mob.: +964 750 448 9550

Turkey Mob.: +90 539 599 0570

Fax: ------

Email:Shakhybarz@yahoo.com, Shakhawan74@hotmail.com

EDUCATION

	Graduate school	**Year**
Master	-----------	-------
High Diploma	Duhok University	2006-2007
Bachelor	Salahadden University	1999-2000
High School	Soran preparatory School	1994-1995

WORK EXPERIENCE

	Place	**Enrollment**
2007-Present	Soran Technical Institute /SORAN	Teaching
2001-2006	Soran Technical Institute /SORAN	Administrative
2000-2001	Syako Company/ERBIL	Accountant

PUBLICATIONS

S. A. Shakhawan, and R. KOÇ "Simulation of 4-qubit full-adder circuit by Mathematica" *International Journal of Computer Science and Network Security* (IJCSNS), under review

FOREIGN LANGUAGE

- English
- Arabic
- Kurdish
- Few of Turkish

HOBBIES

- Studying
- Reading
- Sports
- Travelling

Chapter 1

INTRODUCTION

Quantum computation is going to be a successful project, based on theory of quantum mechanics as well as mathematics, group theory, electronic engineering, quantum information theory and computer science [1]. Theoretically, it has been shown that, various problems which cannot be efficiently solved on a classical computer have efficient solution on a quantum computer [2, 3]. This property can be proven by presenting an algorithm based on superposition principles of the states [4, 5, 16, 41]. Recently Shor invented an algorithm to factor large numbers by using quantum computers much faster than any classical computer do it [5]. This type developments providing motivation to researchers and they are racing to create a practical quantum computer. Recent experiments are promising that a working model of quantum computer could be manufactured in near future [16, 42-45].

Meanwhile, let us explain basic principles of quantum computation: in a classical computer operations are done on bits, can only exist as "0" or "1". In quantum computation the rule is based on superposition. Fundamental building blocks of a quantum computer are qubits that is represented by quantum mechanical states [1-3]. According to the theory of quantum mechanics the states can be expressed as linear combinations of these quantum states, also called superposition, is the most significant property that leads to speed up of a quantum computation. Mathematically, the state of a qubit $|\psi\rangle$ can be written as $|\psi\rangle=a|0\rangle+b|1\rangle$ where a and b are related to probability of the state and $a^2 + b^2 = 1$. This implies that by performing a single operation on the state $|\psi\rangle$, two qubits can be effected at the same time. Similarly a two qubits system, $|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$ can perform operation on four qubits, three qubits system can perform operation on eight qubits and n qubits system can perform the operation on 2^n qubits. This is known as *quantum parallelism* and by a correct algorithm one can use this property to increase speed of the quantum computer exponentially when compared to a classical computer [41].

It is indicated that quantum entanglement also plays a crucial role in quantum algorithms. For instance Grover's search algorithm for two qubits case involves entanglement. Entanglement naturally appears in the Deutsch-Jozsa algorithm for single and two qubits and in the Shor algorithm. Use of entanglement in an algorithm increases speed of computation on quantum computer. Role of the entanglement in quantum computation has not yet been clarified and some questions still unanswered. In fact entanglement takes place at the origin of the quantum information theory.

If a problem can be solved by using quantum computer, it is also solved by using classical computers. Quantum algorithms provide solution of problem on quantum computer is faster than solution of the same problem on classical problem. For example Shor's algorithm based on, Quantum Fourier Transform; factorize a number in a polynomial time [4]. The same problems have been solved on a classical computer in super polynomial time. Grover's algorithm search an unsorted list including N data using $O(\sqrt{N})$ queries. The same search can be done on a classical computer using O(N) queries [5].

Consequently quantum parallelism, based on superposition and entanglement, allow possibility of performing a large number of operations in parallel. Therefore this property has led to the development of new algorithms which run by using nature of quantum mechanical devices manufactured for this purpose (quantum gates) and they cannot be run on a classical computer [26]. As it is well known, a quantum algorithm can be run by a quantum circuit to achieve its processing power.

One of the purposes of this thesis is to simulate quantum algorithms on a classical computer by using Mathematica program [40]. We also investigate behavior of quantum algorithm. Obviously, simulation of quantum computations on classical computers will improve our understanding of quantum mechanics and construction of quantum computers [7-16]. We focus our attention to simulate 4-qubits full adder circuits on Mathematica.

Various quantum algorithms can be written by using their classical algorithms too. In usual, addition algorithms generated for a quantum computer by using analogy of full adder classical algorithms [4, 22]. In this thesis we presented simulation of the four qubits full adder circuit by using a Mathematica package developed by J. L. Gómez-Muñoz and F. Delgado [40]. Although a considerable attention has been paid to present quantum algorithm for full adder circuit, the study of this problem from different point of view leads to the progress of quantum algorithm and simulation techniques. Theoretically and experimentally various quantum gates and circuits have been designed. Especially Hadamard and CNOT gates have attracted widely attentions in the construction of quantum circuits for a given algorithm [6] and our algorithms include CNOT-based quantum circuits. We also mention here, as a part of the improvement of quantum computing, it is necessary to find efficient ways to design a quantum circuit [17-33]. According to the quantum theory quantum logic circuit can be represented by a unitary matrix. Then unitary transformations of the state qubits are gives relation between input and output qubits of the circuit. These circuits are modeled by connecting one or more quantum logic gates represented by a unitary transformation matrix [41].

The theory of quantum computing leads to understanding of quantum circuits including quantum gates. Quantum computers include quantum gates represented by transformation of the quantum states. The transformation is unitary and therefore the quantum gates are reversible. Simulation of quantum computing on a classical computer can be modeled by using the unitary transformation properties of gates that can be represented by unitary matrices. As in the classical computers, quantum circuits are constructed by using quantum gates.

In this thesis simulation of a quantum full adder circuit is presented, because a full adder circuit is a fundamental unit of both quantum and classical computers. We will use a Mathematica [40] package provides a simulation of a Quantum computation.

Consequently, this thesis will describe the connection between future quantum computers and today's simulations of quantum computers. Let us begin with a demonstration of the framework we have constructed in the high-level program language Mathematica; this will probably be used for future algorithms [8].

The thesis is organized as follows. Chapter 2 provides a brief theory of quantum and classical computation. This chapter include an overview of classical and quantum computation, involve classical and quantum gates, quantum circuit and matrix representations of quantum gates. In chapter 3, we have discussed the classical and quantum addition circuit structure in detail. We have given an algorithm and a classical and quantum circuit in this chapter. Chapter 4 is presented to illustrate simulation of 4-qubit full adder quantum circuit on classical computer. The results are given in this chapter. Finally, the results are discussed and concluded in chapter5.

Chapter 2

CLASSICAL AND QUANTUM COMPUTATION

A classical computer is a science depends to bit, where each bit represents either a one or a zero; together quantum computer is a science depends to qubit. A single qubit can represent a one a zero or superposition state of the two qubit; the, for two qubit can be 4 superposition states, and three qubits in any superposition of 8 states. In general, with n qubits in quantum computer can be 2^n different states in an superposition state up to simultaneously. Quantum computer works by appointment qubits in a controlled state which represents the initial problem at hand by manipulating those qubits with a specific sequence of quantum logic gates. The quantum algorithm is obtained by sequence of gates. After measuring all states obtained at the calculation of final, the outcome can be at most n classical bits of information, if collapsing each qubit into one of the two pure states [2, 16-17].

The numbers of qubit in quantum computer is Substantial different from the same number of bit in classical computer. For example, require the storage of 2^n complex coefficients by explain n-qubit state on classical computer system. At the indeed, the qubit can be hold information more exponentially from classical computation [16].

2.1 Classical Computation

Classical computing is Consists of logic gate, any piece of hardware that described by logic gate, it is performs a logical process on one or more logical input to take out a single logical value. The circuits in classical computation consist of logic gates and wires. Information around and substituting in circuit by wires, and operations perform on information by logic gates. Moreover, memory is part of circuit; it is store the value of various bits. [8].

The classical computation includes essential Digital system. Digital system is a branch of electronics in which all the devices, components and circuitry operate only at two predefined levels of voltages, currents or any other quantity. These levels are represented by 1 and 0 and usually called high and low. In digital electronic systems every information in the world numerals, alphabets, signs, characters or commands - anything and everything - is converted to or interpreted by sequences (strings) of binary digits (bits) of 0's and 1's. A binary sequence is also referred as to binary [18].

The most important in digital systems is the binary number system; also others have will be important. Like, quantities outside digital system is represented by decimal system, also decimal system is important because it is universal used. This means that before situations are entered in a digital system, where you must convert decimal values to binary values. For example, when you punch a decimal number into your hand calculator (or computer), the circuitry inside the machine converts the decimal number to a binary value [21].

2.1.1 Digital Systems

Digital systems have such a prominent role in everyday life that we refer to the present technological period as the digital age. Digital systems are used in communication, business transaction, traffic control, space guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises [18, 22].

One characteristic of digital systems is their ability to manipulate discrete elements of information. Any set or group number that is restricted to a limited number of elements contains separate information. Examples of discrete sets are the 10 decimal digits.

Discrete elements of information are represented with groups of bits called binary codes. For example, the decimal digits 0 through 9 are represented in a digital system with a code of four bits. By using different techniques, can make a group of bits to represent the separate symbols, which are then used to develop the system in a digital format. Thus, a digital system is a system that manipulates discrete elements of information that is represented internally in binary form.

2.1.1.1 Number Systems

Actually, number system contents of the many deferent systems, such as decimal, binary, octal, and hexadecimal system, all this are used in digital technology, but decimal system is clearly

Many types of number system are used in digital technology. The most used are the decimal, binary, octal, and hexadecimal systems. Decimal system it is more common because used in all areas of daily living.

The decimal system for counting has been so widely adopted throughout our present civilization that we rarely consider the possibilities of other number systems. Nevertheless, it is not reasonable to expect a system based on the number of fingers we possess to be the most efficient number system for machine construction. The fact is that a little used but very simple system, binary system has proven that the system is more common and efficient use of the device

2.1.1.2 The Decimal System

Decimal system consists of 10 different numbers. These 10 numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9; can express any quantity using these symbols as digits. The base of decimal system is a 10, because it has consists of 10 digits, also called the base-10. The following rule is the general rule to representation decimal system:

$$a_1 10^{n-1} + a_2 10^{n-2} + \ldots + a_n$$

Where (a_1,a_2... a_n) are the value of digit number (left to right), and (n) is the number of digits to the left of the decimal point.

The radix or base of a number system is defined by different digits number. The radix or base of decimal number system is a 10. Because this system it have 10 different numbers (0, 1, 2, ... 9), any one of them can used in every position in a number. Use a several other number systems is recording by history.

2.1.1.3 The Binary System

The binary system is contains only two symbols or possible digit values, 0 and 1. The base of binary system is 2^n, this base is used to represent any binary numbers. Switches and relays are basic elements in early computers. The switch and relay are nature essential operation in binary system; that is, the switch is either on or off (on=1 and off=0). Transistors are principal elements in circuit to modern computers, similar to those used in radios and television sets. The desire for reliability led designers to use these devices so that they were essentially in one of two states, fully conducting or no conducting.

2.1.1.4 Counting in the Binary System

The same type of positional notation is used in the binary number system as in the decimal system. Table (2.1) perform list of the first 20 numbers from binary number versus decimal numbers, the decimal system uses 10^n, and the binary system 2^n. As was previously explained, the number 120 actually means $(1\times10^2) + (2 \times 10^1) + (0\times10^0)$. In the binary system, the same number (120) is represented as 1111000, meaning $(1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$.

To express the value of a binary number, therefore, $a_1 2^{n-1} + a_2 2^{n-2} + \ldots + a_n$, is represented as $a_1 a_2 \ldots a_n$, where (a=0 or a=1), and n is the digits number to the left of the binary radix point.

Table (2.1) The first 20 numbers of the decimal and binary systems

Decimal	Binary	Decimal	Binary
1	00001	11	01011
2	00010	12	01100
3	00011	13	01101
4	00100	14	01110
5	00101	15	01111
6	00110	16	10000
7	00111	17	10001
8	01000	18	10010
9	01001	19	10011
10	01010	20	10100

2.1.1.5 Binary Addition and Subtraction

Operation way to addition in a binary system is the same way the decimal system. Actually, binary arithmetic is much simpler to learn. The following operations are performing all (operation addition) in binary system.

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ with carry 1 to over

The carryovers in binary system are same operation with decimal arithmetic. Since in the binary system 1 is the largest digit, if any sum greater then 1, the operation is requires that a digit be carried over.

The easiest way of performing subtracting is to negate the subtrahend and add it to the minuend this is done by finding the 2’s complement of the subtraction and then performing the addition. For simplifying the subtraction operation and logical manipulation in digital computers using complements operation. Complements are consists of two types in binary system, there are one’s (1’s) complement and two's (2’s) complement.

The 1’s complement of a binary number is the number that results when each 0 in the number is changed to a 1 and each 1 is changed to 0. The 2’s complement of a binary number is the number results by adding 1 to the 1’s complement (i.e. 2’s complement = 1’s complement + 1). The 2’s complement can be used to form the binary subtraction. (i.e. add its 2’s complement to the subtraction and disregard the cast carry)

2.1.1.6 Converting Decimal Numbers to Binary

There are several methods for converting a decimal number to a binary number. The process of dividing the decimal number by 2 repeatedly, it is first and most obvious method, and after each division it have remainder (0 or 1), this remainder is used to indicate the coefficients of the binary number to be formed. Notice to obtain the result reading the remainder (down to up) and writing (left to right).

Decimal No. ÷ 2	Result	Remainder
120	60	0
60	30	0
30	15	0
15	7	1
7	3	1
3	1	1
1	0	1

The binary representation of 120 is therefore 1111000. Checking this result gives

$$\begin{array}{rr}
1 \times 2^6 = & 64 \\
1 \times 2^5 = & 32 \\
1 \times 2^4 = & 16 \\
1 \times 2^3 = & 8 \\
0 \times 2^2 = & 0 \\
0 \times 2^1 = & 0 \\
0 \times 2^0 = & 0 \\
\cline{2-2}
 & 120
\end{array}$$

2.1.2 Logic Gate

In [20-21, 45], majority of the digital circuits and systems irrespective of how much complex the circuit may be and performing any kind of function like mathematical operations, control operations, digital data transfer, data processing, data selection, coding, decoding, etc. at the innermost levels, there are only three basic logic operations going on. These basic logic operations are:

1. AND 2. OR and 3. NOT.

Electronic circuits assembled from diodes, resistors, transistors etc. performing these basic logic operations are called logic gates. AND and OR gates can each have two or more number of binary inputs and one binary output. NOT gate which is often also referred as inverting or complementing gate has only one input and one output. Input to logic gates are always binary variables. Binary variable means it has only two values high or low, on or off, 1 or 0. Thus each input or output can assume only one of these values. In reality, usually one value of the variable is an upper level of voltage the other value is a lower voltage.

By using logic gates can be construction digital systems. Logic Gates are electric circuits, consisting of transistors, diodes, and resistors. Logic gates process input signals and one or more logical manner. Depending on the input value (0 or 1), the logic gate will either output a value of (1→ON) or a value of (0→OFF).

Some of the following terms represent logic 0 and 1, is shown in the table (2.2).

Table (2.2) Logic symbol

Logic 0	Logic 1
Off	On
False	True
No	Yes
Low	High
Open Switch	Close Switch

Logic gates are the electronic circuits which perform the basic logic operations. They are building blocks of majority of the digital systems.

Logic gates consists of seven different gates, there are identified by their function: NOT, AND, NAND, OR, NOR, X-OR and X-NOR.

AND gate is like two or more series switches. All the switches have to be closed (ON) in order to make the lamp (output) turn on. If all inputs are not ON, the output is OFF.

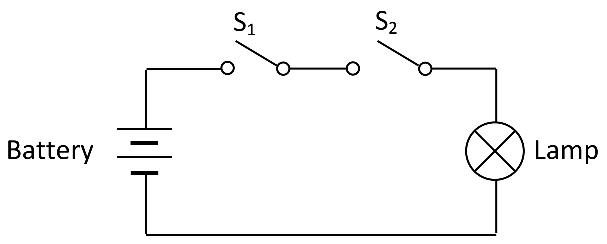


Figure (2.1) Electric circuit representation AND gate

An OR gate is like two or more parallel switches. Only one switch needs to be closed (ON) in order to make the lamp (output) turn ON.

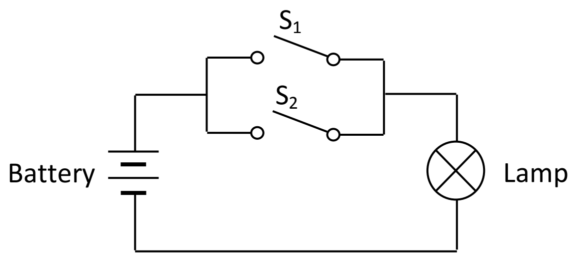


Figure (2.2) Electric circuit representation OR gate

An NOT gate is like one switch and lamp in parallel. Switch needs to be closed (ON) in order to make the (output) turn OFF, but switch is open (OFF) in order to make the (output) turn ON.

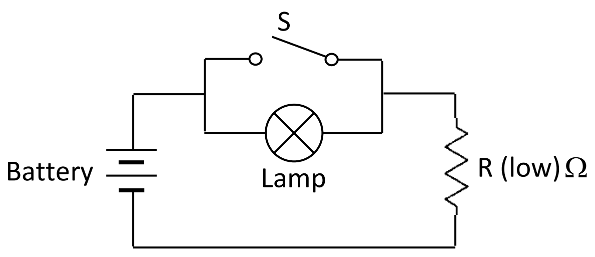


Figure (2.3) Electric circuit representation NOT gate

2.1.2.1 AND gate

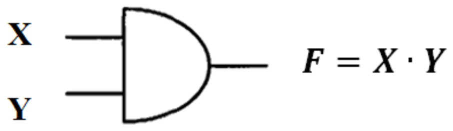


Figure (2.4) Block diagram representation AND gate

The AND Gate is an electronic circuit consists of two Inputs or more and one output, gives a “1” of the output only if all its inputs are “1”. A dot (.) is used to show the AND operation, If one input variable is *X,* the other input variable is *Y,* and the output variable is *F,* then the Boolean expression is $F = X \cdot Y$.

Table (2.3) Truth Table of 2 Input AND gate

2 Input AND gate		
X	Y	$F = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1
1= HIGH, 0=LOW		

2.1.2.2 OR gate

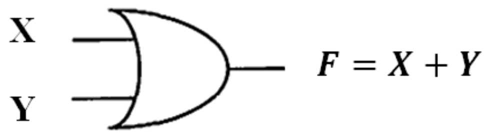


Figure (2.5) Block diagram representation OR gate

The OR Gate is an electronic circuit consists of two Inputs or more and one output, gives a "1" of the output if one or more inputs are "1". A plus (+) is used to show the OR operation, If one input variable is *X,* if the other input variable is *Y,* and if the output variable is *F,* then the Boolean expression is $F = X + Y$.

Table (2.4) Truth Table of 2 Input OR gate

2 Input OR gate		
X	Y	F=X+Y
0	0	0
0	1	1
1	0	1
1	1	1
1= HIGH, 0=LOW		

2.1.2.3 NOT gate

The OR Gate is an electronic circuit consists of only one input and one output; the property for this gate is reversed input value. It is also known as an inverter, if the input variable is called A and the output variable is called $\bar{A}$, then ($A = \bar{A}$), where $\bar{A}$ is inverse A value, if (A=0 → $\bar{A}$=1 and A=1 → $\bar{A}$=0).

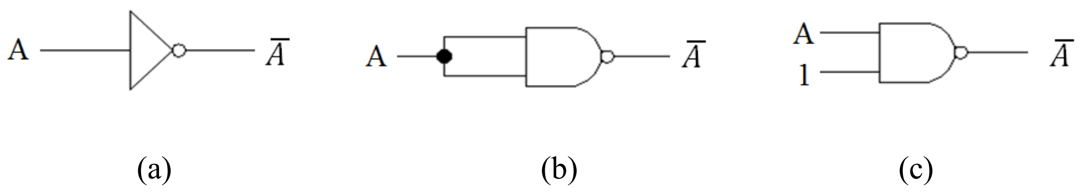


Figure (2.6) Figure (a, b and c) block diagram representation NOT gate

The figure (2.6)-b and (2.6)-c show two ways that the NAND logic gate can be configured to produce a NOT gate. Also can be done using NOR logic gates in the same way.

Table (2.5) Truth Table of NOT gate

Input	**Output**
A	$\overline{A}$
0	1
1	0
1= HIGH, 0=LOW	

2.1.2.4 NAND gate

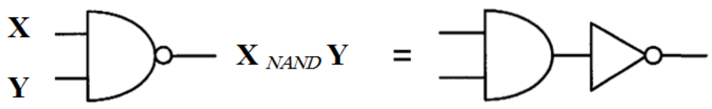


Figure (2.7) Block diagram representation NAND gate

The NAND gate structure of AND and NOT gates, It's consists only two inputs and one output, the outputs of all NAND gates are high if any of the inputs are low. The symbol of NAND gate is the same symbol of AND gate with a small circle on the output. The small circle represents inversion, The Boolean expression for the output of a 2-input NAND gate is: $F = \overline{X \cdot Y}$

.**Table (2.6)** Truth Table of NAND gate

2 Input NAND gate		
X	Y	$F = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0
1= HIGH, 0=LOW		

2.1.2.5 NOR gate

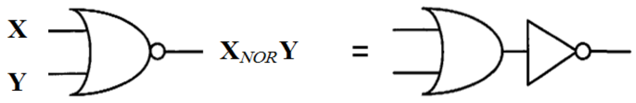


Figure (2.8) Block diagram representation NOR gate

The NOR gate structure of OR and NOT gates, It's consists only two inputs and one output, the outputs of all NOR gates are low if any of the inputs are high. The symbol of NOR gate is the same symbol of OR gate with a small circle on the output. The small circle represents inversion, The Boolean expression for the output of a 2-input NOR gate is: $F = \overline{X + Y}$

Table (2.7) Truth Table of NOR gate

2 Input NOR gate		
X	Y	$F = \overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0
1= HIGH, 0=LOW		

2.1.2.6 XOR gate

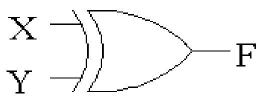


Figure (2.9) Block diagram representation XOR gate

The XOR (Exclusive-OR) gate structure of five gates (2-AND, 2-NOT and 1-OR) gates, It's consists only two inputs and one output. It will give a "1" output if both inputs are different value, but if the two inputs are same value the output give a "0".

An encircled plus sign ($\oplus$) is used to show the XOR operation, The Boolean expression for the output of XOR gate is $F = X \oplus Y$.

Table (2.8) Truth Table of XOR gate

Exclusive XOR gate		
X	Y	$F = X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0
1= HIGH, 0=LOW		

2.1.2.7 XNOR gate

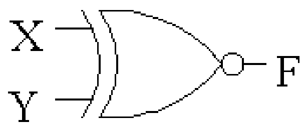


Figure (2.10) Block diagram representation XNOR gate

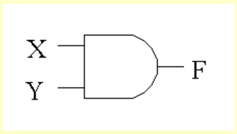
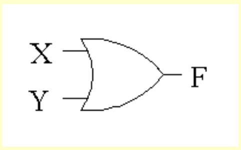
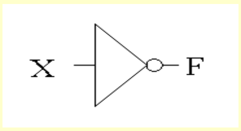
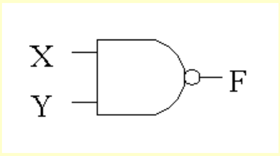
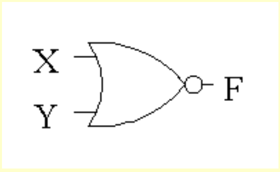
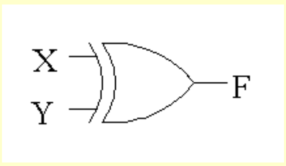
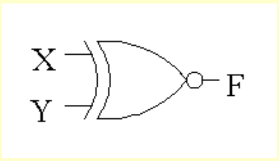
The XNOR gate circuit does the opposite to the XOR gate. It will give a "1" output if both inputs are same value, but if the two inputs are different value the output give a "0". The symbol is an XOR gate with a small circle on the output. The small circle represents inversion, The Boolean expression for the output of XNOR gate is:

$F = \overline{X \oplus Y}$.

Table (2.9) Truth Table of XNOR gate

Exclusive XNOR gate		
X	Y	$F = \overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1
1= HIGH, 0=LOW		

Table (2.10) Summary of Digital logic gates

No.	*Name*	*Graphic symbol*	*Algebraic function*	*Truth table*
1	AND		$F = X \cdot Y$	X Y | F 0 0 | 0 1 0 | 0 0 1 | 0 1 1 | 1
2	OR		$F = X + Y$	X Y | F 0 0 | 0 1 0 | 1 0 1 | 1 1 1 | 1
3	Inverter		$F = \overline{X}$	X | F 0 | 1 1 | 0
4	NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 1 0 | 1 0 1 | 1 1 1 | 0
5	NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 1 0 | 0 0 1 | 0 1 1 | 0
6	Exclusive-OR (X-OR)		$F = X \cdot \overline{Y} + \overline{X} \cdot Y$ $F = X \oplus Y$	X Y | F 0 0 | 0 1 0 | 1 0 1 | 1 1 1 | 0
7	Exclusive-NOR (X-NOR)		$F = X \cdot Y + \overline{X} \cdot \overline{Y}$ $F = \overline{X \oplus Y}$	X Y | F 0 0 | 1 1 0 | 0 0 1 | 0 1 1 | 1

2.1.3 Boolean Operators

Signs for basic operations of AND, OR and NOT are termed as Boolean operators. These are (•), (+) and $\overline{X}$ where *x* is the binary variable and the bar above it is the NOT operator also known as complement. Hence A (AND) B is normally written as $A \cdot B$ or also sometimes simply AB. In the latter case, the dot between A and B is understood.

Since in normal mathematics the dot (•) stands for product, $A \cdot B$ is also many times termed as product form. In fact Y= A (AND) B comes out to be the same as Y = A × B in mathematics. But hear (A) and (B) are the binary numbers only.

Similarly A (OR) B is normally written as A + B. In normal mathematics + sign stands for summation. Therefore this is the sum form of the variables A and B. But Y=A (OR) B in Boolean form does not always give the same result as in normal mathematics A plus B gives.

The third Boolean operation which is called the NOT operation or complement operation and is given by $Y = \overline{A}$, which is usually pronounced as A bar. That is, if A=1, Y=0 and if A=0, Y=1

2.1.4 Boolean Algebra

The algebra which deals with binary variables and the logic operators AND (•), OR (+) and NOT ($\overline{X}$) is called Boolean algebra. Boolean algebra also follows the commutative, associative and distributive laws of the common algebra.

Table (2.11) Logic Basic Rules and Boolean algebra laws

Type of laws	No.	Result
Basic Rules	1	$A \bullet 0 = 0$
	2	$A \bullet 1 = A$
	3	$A \bullet A = A$
	4	$A \bullet \overline{A} = 0$
	5	$A + 0 = A$
	6	$A + 1 = 1$
	7	$A + A = A$
	8	$A + \overline{A} = 1$
	9	$\overline{\overline{A}} = A$
Commutative laws	10	$A \bullet B = B \bullet A$
	11	$A + B = B + A$
Association laws	12	$(A \bullet B) \bullet C = A \bullet (B \bullet C) = A \bullet B \bullet C$
	13	$(A+B)+C=A+(B+C)=A+B+C$
Distribution laws	14	$A \bullet (B+C)=(A \bullet B)+(A \bullet C)$
	15	$A+(B \bullet C)=(A+B) \bullet (A+C)$
Absorption laws	16	$A \bullet (A+B)=A$
	17	$A+(A \bullet B)=A$
De Morgan's laws	18	$\overline{A \cdot B} = \overline{A} + \overline{B}$
	19	$\overline{A + B} = \overline{A} \cdot \overline{B}$

Above identities can be easily proved either by using basic rules and laws and other identities or by taking the value of each variable once 0 and then 1. Every time it will be found that left hand side of the equation is equal to right hand side.

2.2 Quantum Computation

Quantum computation and information are currently of crucial importance for computer and physical sciences, mathematics and engineering. They will probably lead to a new era of technological innovations in communication, computation and cryptography. It is argued that quantum information and quantum bits will become the forerunner of a 21st century technological breakthrough as classic information and bits were to the 20th century. As quantum physics theory is basically stochastic, randomness and uncertainty are deeply ingrained in quantum computation, quantum simulation and quantum information [23, 41].

2.2.1 Background

Since time immemorial, man has constantly been looking for tools to assist them with carrying out tasks which require calculations. Such as land computing, calculate the stresses on the rails in the bridges, or to find the shortest route between two places. It is structure that interlinks all these tasks:

$$\text{Input} \rightarrow \text{Computation} \rightarrow \text{Output}$$

The dynamical physical system can be perform the inevitably computation part of the process, evolving in time [24].

The lack of a machine model is a theoretical obstacle ahead of quantum computing. Must be developed describe and formalize the process for this new model and it must be based on applied mathematics or physics in order to more accurately give insight into the quantum computational process. The quantum computer can no longer be thought of as a tape head and an endlessly long tape. In accordance with quantum mechanics, we must look at the tape as a system described by the state function $\psi(t)$ that evolves by the passage of time to implement the calculation.

The classical computer systems perform computation by sending an electrical signal through a circuit in conjunction with the signal timing. This signal dependent on itself and it does not need interaction with any other signals to do its calculation.

It is through a gradual control of the memory evolution that a quantum computer system performs its calculations. At the beginning must be prepared a quantum computation in an initial state, which would correspond to input. This input is then transitioned to other quantum states by one of a variety of methods. The transition of input to a general quantum computation is now more than just the concatenation of its bits, because each input bit can be entangled with its neighbor and each bit is in a superposition of states.

It has been considered as essential to supply sufficient background on quantum physics to clearly explain the substance that follows, "braket" is a notation in quantum physics; it's used to describe a quantum state, this portion is used to indicate inner product of two states "braket" $\langle\varphi|\psi\rangle$. The left side of notation $\langle\varphi|$ is described the "bra", while the right side of notation $|\psi\rangle$ is described "ket". The part "ket" at the notation is used to represent a quantum state, In the situation of quantum logic, there are two distinct quantum states, "one" and "zero". These are represented by $|1\rangle$ and $|0\rangle$. For the rest of this thesis, the quantum state $|\psi\rangle$ will be used to represent which is a superposition of the two distinct quantum states [25].

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

In this case α and β represent the probability of each of the quantum states. Since the Hilbert space is a vector space, a quantum state can also be expressed in vector notation. For example, in the equation $|\psi\rangle=\alpha|0\rangle+\beta|1\rangle$, $|\psi\rangle$ could also be expressed as the vector $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$.

Due to the work of Dirac in quantum mechanics, the use of the "braket" notation seems to be common.

For instance, consider the only single input, single output, classical logic gate and the NOT gate. The NOT gate basically accepts a logic input and provides the negation of this input as the output of the circuit. Alternatively, the gate interchanges the 0 and 1 states of classical bits. The best possible way to start with is to show how a quantum NOT gate is constructed. Recall that quantum bits can represent the zero state ($|0\rangle$), the one state ($|1\rangle$), or any superposition of states in between. These states can also be expressed in vector notation. The following definitions describe how the zero and one states are expressed in vector notation.

$$|0\rangle=\begin{bmatrix}1\\0\end{bmatrix}\ and\ |1\rangle=\begin{bmatrix}0\\1\end{bmatrix}$$

To implement a quantum NOT, a gate is required that will turn the probabilities of each quantum state the other way round. As it is possible to represent quantum gates as unitary matrices, the quantum NOT gate can be represented by the following matrix:

$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

So as to ensure the functional operation of this gate, the following example is given. The quantum state $\alpha|0\rangle+\ \beta|1\rangle$written in vector notation is $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$, to compute the output of the quantum gate, a matrix-vector product is generated. For example if $\alpha=0$ and $\beta=1$, it can be seen that

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Thus, the quantum NOT gate has finally been realized. It is similarly essential to note that this quantum gate (side by side with all other quantum gates) is totally reversible. Obviously, the original quantum vector would be the result if the output of the former equation were to be replaced with the input [6, 8].

2.2.2 Properties of Quantum Computation

A computational equivalency exists between this model and a quantum computer. The computation in both cases is regarded to be generated by the time evolution of a computer memory from an initial to a final state. In a quantum computer, it is not the state of the memory but the probability of measuring a state that is propagated in time. However, this is only the elementary phase. The big difference between classical computation and quantum computation is non-existence of the following properties in classical computation.

- Superposition
- Entanglement
- Logical/physical reversibility
- Coherency
- Time independence
- Output interrogation

2.2.3 Quantum Bit

Qubit is the smallest unit of information in a quantum computer and the main part of a quantum computer, whose states are manipulated by a series of quantum logic gates. Unlike bits in classical systems, which are in one of two possible states labeled 1 and 0, a quantum bit exists in a superposition of these two states, settling on one or the other only when a measurement of the state is made [26-27].

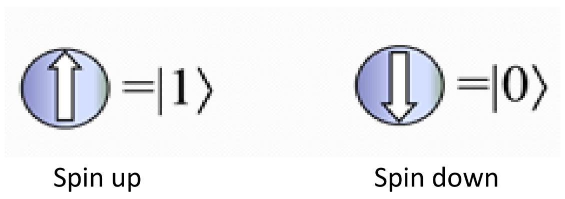


Figure (2.11) Spin Up and Spin down representation Quantum Bit

A qubit of data is represented by a single atom that is in one of two states denoted by $|0\rangle$ and $|1\rangle$.

The figure below performs the state of bit in classical computation, and state of qubit in quantum computation.

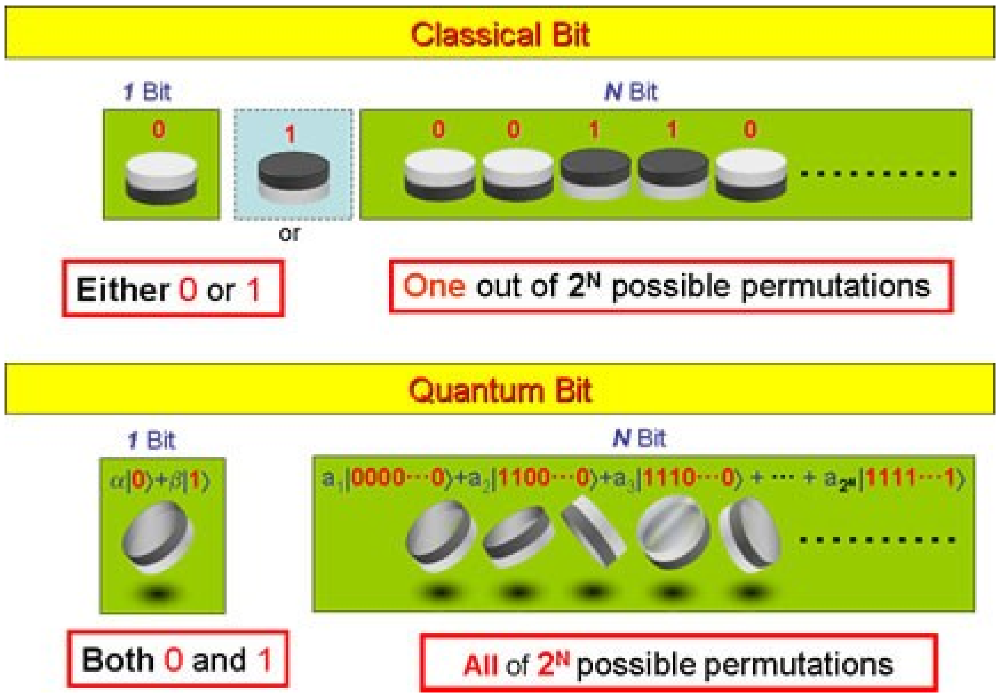

Classical Bit
1 Bit
N Bit
0
1
0 0 1 1 0
or
Either 0 or 1
One out of 2^N possible permutations
Quantum Bit
1 Bit
N Bit
α|0⟩+β|1⟩
a1|0000⋯0⟩+a2|1100⋯0⟩+a3|1110⋯0⟩ + ⋯ + a2N|1111⋯1⟩
Both 0 and 1
All of 2^N possible permutations

Figure (2.12) possible state of Classical Bit and Quantum Bit

2.2.4 Qubit States

The linear superposition of the basis states is the pure qubit state. According to this meaning the linear combination of $|0\rangle$ and$|1\rangle$ can be represented by qubit:

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

Where α and β are probability amplitudes and can generally be both complex numbers. When this qubit is measured in the standard basis, the probability of outcome$|0\rangle$is $|\alpha|^2$ and the probability of outcome$|1\rangle$is$|\beta|^2$. Because the probabilities equal to absolute squares of the amplitudes, where α and β contacted to each other by the following equation:

$$|\alpha|^2 + |\beta|^2 = 1$$

Simply because this ensures you must measure either one state or the other [16, 28].

2.2.5 Bloch Sphere

By using Bloch sphere can be visualized a possible states for a single qubit (see diagram). Represented on such a sphere, a classical bit could only be at the "North Pole" or the "South Pole", in the locations where $|0\rangle$ and $|1\rangle$ are respective The rest of the surface of the sphere is inaccessible to a classical bit, but a pure qubit state can be represented by any point on the surface. For example the pure qubit state $\frac{|0\rangle+i|1\rangle}{\sqrt{2}}$ would lie on the equator of the sphere, on the positive y axis.

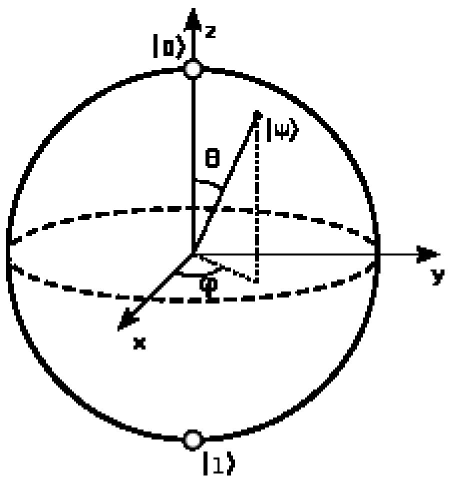


Figure (2.13) Bloch sphere representation of a qubit

The surface of the block sphere is consisting of a two-dimensional space, which represents the state space of the pure qubit states. This state space owns two local degrees of freedom. It is likely that at first look it appears that there must be four degrees of freedom, as α and β are complex numbers with two degrees of freedom for each. Nevertheless, one degree of freedom is taken away by the constraint.

$$|\alpha|^2 + |\beta|^2 = 1$$

2.2.6 Quantum Gates

In quantum computing and especially the quantum circuit model of quantum computation, a quantum gate is essential and a basic quantum circuit operating on a small number of qubits. By classical logic gates and conventional digital circuits, they can build blocks of quantum circuits.

Many classical logic gates are Dissimilar, but quantum logic gates are reversible. However, classical computing can be performed using only reversible gates. For example, the reversible Toffoli gate can implement all Boolean functions. Showing that quantum circuits are capable of implementing all operations performed by classical circuits [29, 28].

Quantum logic gates are expressed in the form of unitary matrices. The most common quantum gates operate on spaces of one or two qubits, just like the common classical logic gates operate on one or two bits. Quantum gates can be described by 2×2 or 4×4 unitary matrices, like means of matrices.

Quantum gates represent by normally Matrices, a gate which acts on k qubits is represented by a $2^k \times 2^k$ unitary matrix. The number of qubits in the input and output of the gate has to be equal. The action of the quantum gate is worked out by multiplying the matrix representing the gate with the vector which represents the quantum state [16, 30].

2.2.6.1 Hadamard gate

The Hadamard gate consists on a single qubit. The most important process of Hadamard gate is convert state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Hadamard gate represents a rotation of π about the x and z axes. The following matrix is the representing a Hadamard gate.

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

Since the rows of the matrix are orthogonal, *H* is indeed a unitary matrix.

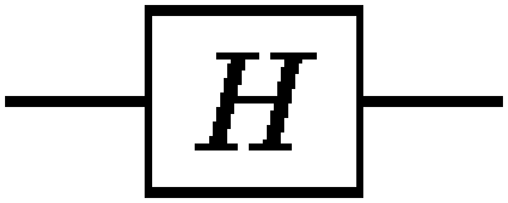


Figure (2.14) Circuit representation of Hadamard gate

2.2.6.2 Pauli-*X* gate

The Pauli-*X* gate consists on a single qubit. It is the quantum equivalent of a NOT gate. It equates to a rotation of the Bloch Sphere around the x-axis by π radians. The most important process of Pauli-*X* gate is convert state $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. The following matrix is representing a Pauli-X gate.

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

2.2.6.3 Pauli-*Y* gate

The Pauli-*Y* gate consists on a single qubit. It equates to a rotation around the Y-axis of the Bloch Sphere by π radians. It is convert state $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to -$i|0\rangle$. The following matrix is representing a Pauli-Y gate.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

2.2.6.4 Pauli-Z gate

The Pauli-*Z* gate consists on a single qubit. It equates to a rotation around the Z-axis of the Bloch Sphere by π radians. Thus, it is a special case of a phase shift gate with θ=π. It leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. The following matrix is representing a Pauli-Y gate.

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

2.2.6.5 Phase shift gates

This is a family of single-qubit gates that leave the basis state $|0\rangle$ unchanged and map $|1\rangle$ to $e^{i\theta}|1\rangle$. The probability of measuring a $|0\rangle$ or $|1\rangle$ is unchanged after applying this gate; however it modifies the phase of the quantum state. This is equivalent to tracing a horizontal circle (a line of latitude) on the Bloch Sphere by θ radians.

$$R_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

Where θ is the *phase shift*, some common examples:

θ = π, $\theta = \frac{\pi}{2}$ and $\theta = \frac{\pi}{8}$

2.2.6.6 Swap gate

The most important characteristics of this gate are the output value equal to the value of mutual input. If the input value of first line equal to (A) and the input value of second line equal to (B), then the output vale of first line equal to (B) and the output value of second line equal to (A).

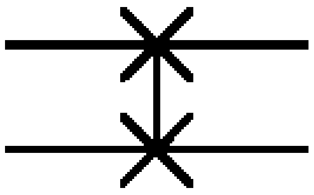


Figure (2.15) Circuit representation of SWAP gate

The swap gate swaps two qubits. It is represented by matrix bellow:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The SWAP gate synthesis and combination in three Control Not gate (CNOT), the following figure representation of SWAP gate:

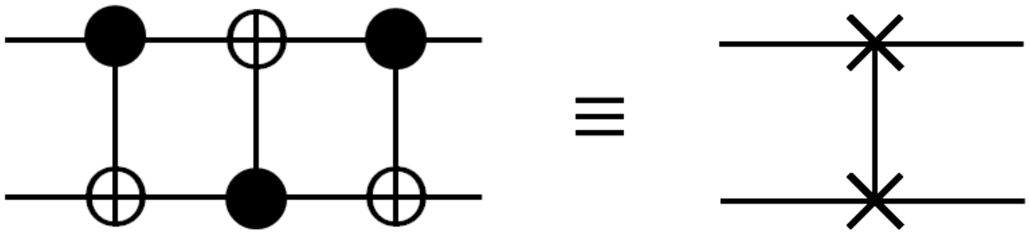


Figure (2.16) A SWAP gate is three back to back CNOT gates with control and target qubits alternating.

2.2.6.7 Controlled gates

The Controlled gate consists on a double qubits, where one or more qubits act as a control for some operation. For example, the controlled NOT gate (or CNOT) acts on 2 qubits, and performs the NOT operation on the second qubit only when the first qubit is$|1\rangle$, and otherwise leaves it unchanged.It is represented by the matrix:

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

In this gate the value of input and output of the first line is equal, but value of output in the second line is dependent on expression rule of X-OR gate.

If the input value of first line equal to (A) and the input value of second line equal to (A), then the output vale of first line equal to (B) and the output value of second line equal to (A $\oplus$ B).

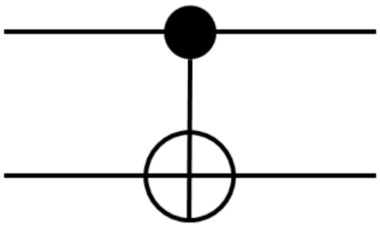


Figure (2.17) Circuit representation of controlled NOT gate

More generally if *U* is a gate that operates on single qubits with matrix representation $U = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix}$, then the *controlled-U gate* is a gate that operates on two qubits in such a way that the first qubit serves as a control. It maps the basis states as follows.

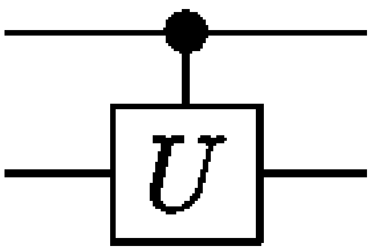


Figure (2.18) Circuit representation of controlled-*U* gate

$|00\rangle \rightarrow |00\rangle$

$|01\rangle \rightarrow |01\rangle$

$|10\rangle \rightarrow |1\rangle U|0\rangle = |1\rangle(x_{00}|0\rangle + x_{10}|1\rangle)$

$|11\rangle \rightarrow |1\rangle U|1\rangle = |1\rangle(x_{01}|0\rangle + x_{11}|1\rangle)$

The matrix representing the controlled *U* is

$$C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x_{00} & x_{01} \\ 0 & 0 & x_{10} & x_{11} \end{bmatrix}$$

When *U* is one of the Pauli matrices, σ_x,σ_y, or σ_z, the respective terms "controlled-*X*", "controlled-*Y*", or "controlled-*Z*" are sometimes used.

2.2.6.8 Toffoli gate

The block diagram of Toffoli gate is consist of three lines, the first two lines its Control line and the last line it's a NOT gate, the following figure is representation Toffoli gate.

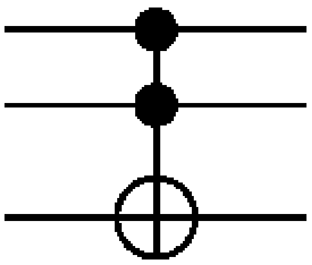


Figure (2.19) Circuit representation of Toffoli gate

The Toffoli gate, also CCNOT gate, is a 3-bit gate, which is universal for classical computation. The quantum Toffoli gate is the same gate, defined for 3 qubits, the Toffoli gate swaps three qubits. It is represented by matrix bellow:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

If the first two bits are in the state$|1\rangle$, it applies a Pauli-X on the third bit, else it does nothing. It is an example of a controlled gate. Since it is the quantum analog of a classical gate, it is completely specified by its truth table.

Table (2.12) Truth table of Toffoli gate

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

It can be also described as the gate which maps $|a, b, c\rangle$ to$|a, b, c\oplus ab\rangle$.

2.2.6.9 Fredkin gate

The block diagram of Fredkin gate is consist of three lines, the first upper line its Controlled line and the other lines it's SWAP gate, the following figure is representation Fredkin gate.

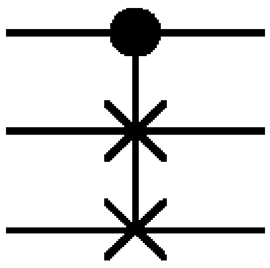


Figure (2.20) Circuit representation of Fredkin gate

The Fredkin gate (also CSWAP gate) is a 3-bit gate that performs a controlled swap, it is represented by matrix bellow:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

It is universal for classical computation. It has the useful property that the numbers of 0s and 1s are conserved throughout, which in the billiard ball model means the same number of balls are output as input. This matches agreeably to the conservation of mass in physics, and helps to show that the model is not wasteful.

Table (2.13) Truth table of Fredkin gate

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

It can be also described as the gate which maps $|a, b, c\rangle$to $|a, \bar{a}b + ac, \bar{a}c + ab\rangle$.

2.2.7 Matrix Representations

The ability of quantum gates to be represented by a transformation matrix is a special quality. A quantum gate that operates on n qubits can be represented by a $2^n \times 2^n$unitary matrix. A cascade of gates forming a quantum logic circuit can also be represented by a single matrix formed by the direct multiplication of the matrices representing the individual gates [8, 31]. As an example, the matrix below shows the matrix representation of the 3-qubit in the figure (2.21). In this example, the qubits labeled x_0 and x_2 are the control qubits, while the qubit marked x_1 is the target. This particular gate is denoted T(x_2, x_0;x_1) because x_2 represents the most important qubit and x_0 being the least important qubit. All gates mentioned in this thesis are denoted in a similar fashion.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The representation of a quantum gate with a unitary matrix gives way to many complex operations to be performed through linear algebraic methods. For instance, by using matrix-vector multiplication, it is possible to produce the output vector from aspecific input vector representing the quantum states of the qubits provided as inputs to aquantum gate.

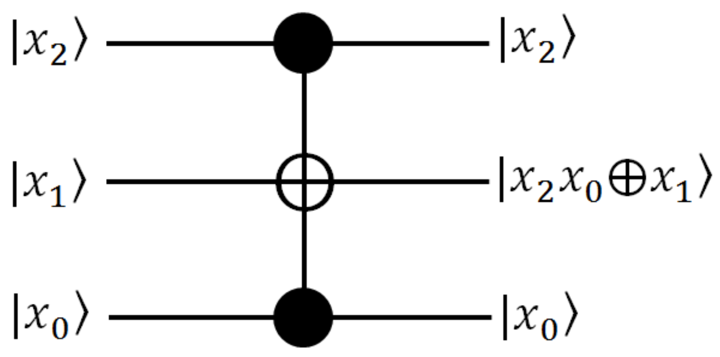


Figure (2.21) The quantum circuit representation of 3-qbit

2.2.8 Quantum Circuits

A combinational quantum-logic circuit consists of quantum gates, interconnected by quantum wires which carry qubits. It is simply a circuit which is comprised of one or more quantum gates. Multiple quantum gates may be cascaded together to form a quantum circuit [8, 32]. As each quantum gate has similar number of inputs and outputs, any cut through the circuit crosses the same number of wires. A quantum circuit can be identified as representing the sequence of quantum-logic operations on a quantum register.

The circuit needs to be read from left-to-right; however, the transformation may also be read in the opposite direction because all quantum gates are reversible. Each line in the circuit acts as a wires in the quantum circuit. This wire is not necessarily a physical wire; it may correspond instead to the passage of time, or perhaps to a physical particle such as a photon - a particle of light - 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 state consisting of all $|0\rangle$s. This rule is broken frequently in the literature on quantum computation and quantum information, but it is considered polite to inform the reader when this is the case [8, 33].

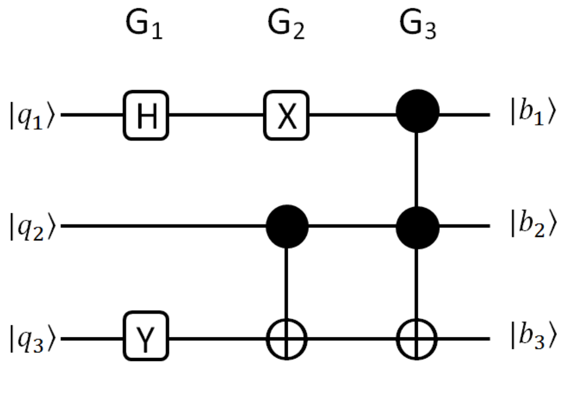


Figure (2.22) the sample of quantum circuit

As an example, the circuit in Figure 2.22 is a simple quantum circuit. In this particular circuit, the (q_1, q_2 and q_3) three qubits input represents and (b_1, b_2 and b_3) the qubit output represents.

Individual quantum gates have the special property that they can be represented by a single unitary matrix. Similarly, a quantum circuit can be represented by a single unitary matrix. This property of quantum circuits is used extensively when designing a quantum circuit simulator.

This example illustrates how a unitary matrix representing an entire quantum circuit is built. The circuit shown in Figure 2.22 will be utilized for this example. The first step in generating the representational matrix is to build the unitary matrix for each individual gate. Since the rightmost Controlled-NOT gate involves two qubits, the representational matrix will be of size $2^2 \times 2^2$, or 4×4. Since this example is considering only the $|0\rangle$ and $|1\rangle$ states, the matrix can be thought of as a permutation matrix. The matrix below is the representational matrix for this gate.

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

However, this matrix cannot be used to build the matrix representing the entire circuit. Since the entire circuit has four qubits, it is necessary to “extend” this matrix to the size of $2^3 \times 2^3$. To perform the extension, the Kronecker operation is used. The Kronecker operation is defined in [8] as the following:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

In order to properly extend the matrix, a Kronecker operation is performed on the representational matrix with an identity matrix. If the unused qubit lies above the target qubit, the identity matrix is placed on the right side of the representational matrix; otherwise it is place on the left. For example, to extend the matrix mentioned before, a Kronecker operation is performed on the left and right sides of the representational matrix.

$G_1 = H \otimes I \otimes Y;$

$$G_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$G_1 = \frac{i}{\sqrt{2}}\begin{bmatrix} 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

This process can be continued until all four gates have individual representational matrices; the matrices for the gates in the circuit in Figure (2.22) can be seen below.

$G_2 = X \otimes CNOT$

$$G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$G_3 = Toffoli\ Gate;$

$$G_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Finally, in order to generate the representational matrix for the entire circuit, the matrices are multiplied together to obtain the following product using traditional matrix multiplication.

$G = G_1 \otimes G_2 \otimes G_3;$

$$G = \frac{i}{\sqrt{2}} \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

This matrix is the representational matrix for the entire quantum circuit. This same procedure can be used to build the representational matrices for any quantum circuit.

Chapter 3

CLASSICAL AND QUANTUM FULL ADDER CIRCUIT

3.1 Background

Quantum computation based on principles of quantum mechanics. In quantum mechanics a quantum state (or qubit) can be typically obtained from the state of a two-level quantum system. As an example ground state and excited state of an atom or the vertical and horizontal polarizations of a single photon are represented as qubits. The qubits are denoted by using Dirac notation such as one of these states as $|0\rangle$ and the other as $|1\rangle$.

According to the theory of quantum mechanics the states can be written as linear combinations of these pure states, also called superposition, is the most significant property that leads to speed up of a quantum computation. In other words, the state of a qubit ψ can be written as $\psi=\alpha|0\rangle+\beta|1\rangle$ where α and α are complex numbers and $\alpha^2 + \beta^2 =1$. This implies that by performing a single operation on the state ψ, two qubits can be effected at the same time. Similarly a two qubit system can perform operation on four qubit, three qubit systems can perform operation on eight qubits and n qubit system can perform the operation on 2^n qubits. This is known as quantum parallelism and by a correct algorithm one can use this property to increase speed of the quantum computer exponentially when compared to a classical computer.

Now, we will briefly discuss various quantum gates with different functionalities and useful to construct a quantum circuit.

These are, identity I, NOT, CNOT, C^2NOT and SWAP gates. Icons of the gates are given in the figure 3.1. In the figure, the symbols •, ⊕ and | are used for control, target and contact qubits respectively. Let us summarize action of each gate:

- identity gate (I) with matrix M_I no action on the qubits. Its icon is a horizontal wire.
- NOT gate inverts the working qubit and its action is given by the matrix M_{NOT}.
- CNOT gate, which act on a qubit as follows: if the control qubit is $|1\rangle$, then the target qubit is inverted. Otherwise it remains unchanged. It is action on qubits can be obtained by using the matrix M_{CNOT}.
- SWAP gate exchanges the values of input qubits.
- C^2NOT gate is controlled-CNOT gate, also known as Toffoli gate. Its action can be described as follows: if both control qubits are $|1\rangle$, the target is inverted; otherwise it is remains the same [4].

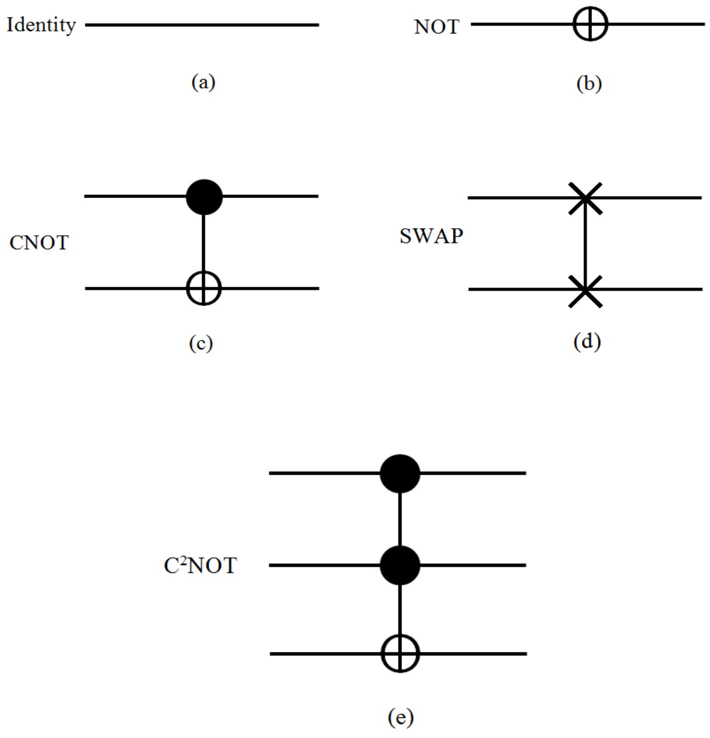
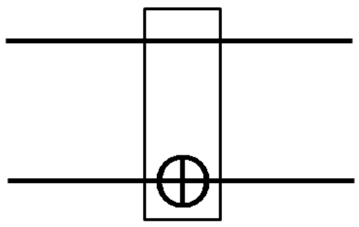


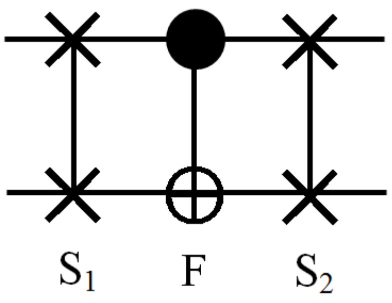
Figure (3.1) Basic quantum gates

We also mention here quantum gates are represented by unitary matrices and the circuits are also represented by unitary matrices. Such circuits are called unitary stabilizer circuits [6]. For example, in figure 3.2, NOT gate combined with identity gate. Matrix representation of combined gates can be obtained by direct product of M_I and M_{NOT}.



$$M = M_I \otimes M_{NOT} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure (3.2) A compound gate constructed from an identity and a NOT gate.



Circuit: $S_2 \circ (F \circ S_1)$

$M = M_{SWAP} \cdot (M_{CNOT} \cdot M_{SWAP})$

Figure (3.3) Cascading quantum gates to construct a quantum circuit and its QMatrix

3.2 Classical Addition

A binary Adder-Subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers. We will develop this circuit by means of a hierarchical design. The half adder design is carried out first from which we develop the full adder. Connecting *n* full adders in cascade produces a binary adder for two *n*-bit numbers. The subtraction circuit is included by providing a 2’s complement circuit [44].

The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations: 0+0=0, 0+1=1, 1+0=1, and 1+1=10. The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1. The binary sum consists of two digits. The higher significant bit of this result is called a carry. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits. A combinational circuit that performs the addition of two bits is called a half adder. Other act that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

3.2.1 The Half-Adder

The half-adder (HA) accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit. A half-adder is represented by the logic symbol in Figure (3.4.b).

From the operation of the half-adder as stated in Table 1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (Cout) is a 1 only when both A and B are 1s; therefore, Cout can be expressed as the AND of the input variables.

$$Cout = AB \quad \text{-------} \ (1)$$

Now observe that the sum output (∑) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\sum = A \oplus B \quad \text{-------} \ (2)$$

From Equations (1) and (2), the logic implementation required for the half-adder function can be developed. The output carry is produced with an AND gate with A and B on the inputs, and the sum output is generated with an Exclusive-OR gate, as shown in Figure (3.4.b). Remember that the Exclusive-OR is implemented with AND gates, an OR sate, and inverters.

Table (3.1) Half adder truth table

INPUT		OUTPUT	
A	B	∑	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
Binary digits to be added		Sum	Carry out
		XOR	AND

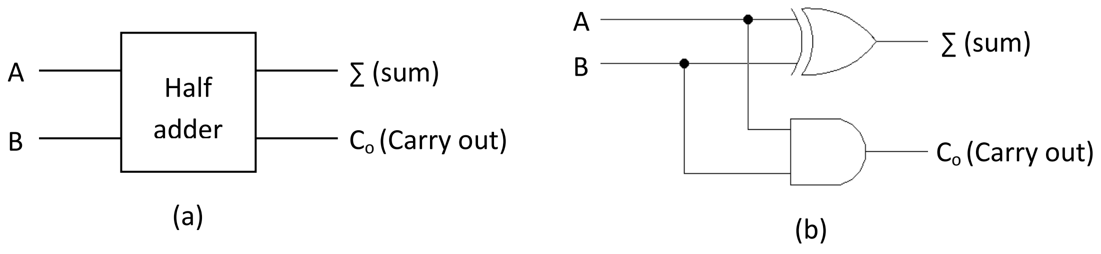


Figure (3.4) Half adder: (a) Block diagram

(b) Logic diagram

3.2.2 The Full-Adder

A classical full adder (FA) operates with an input of two addend bits, "*A*" and "*B*", and a carry bit, "C_{in}". (See Figure 3.5) In figure 4, *S* and C_{out} are the output sum and the carry-over, respectively. The sum, *S*, can be easily expressed as $S = A \oplus B \oplus C_{in}$ (where ⊕ is an addition modulo 2). One can easily obtain an expression for $\mathrm{C_{out}} = (\mathrm{A \oplus B})\mathrm{C_{in}} + \mathrm{AB}$ [34-35].

Table (3.2) Full-Adder truth table

INPUT			OUTPUT	
A	B	C_{in}	∑	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
Carry + A + B			Sum	Carry out

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, in order to construct a full adder circuit for the numbers more than 1 binary digit we connect classical full adder circuit in cascade as in the figure 3.6. The interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.

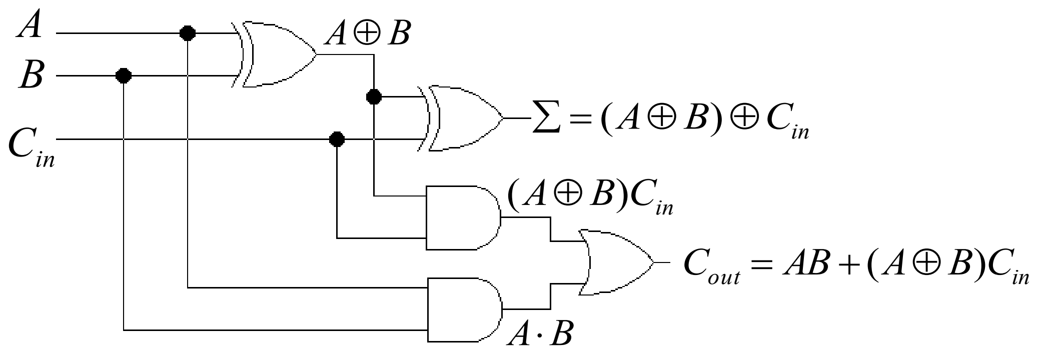

A
B
C_{in}
$A \oplus B$
$\Sigma = (A \oplus B) \oplus C_{in}$
$(A \oplus B)C_{in}$
$C_{out} = AB + (A \oplus B)C_{in}$
$A \cdot B$

Figure (3.5) Classical full-adder circuit

The augend bits of A and the addend bits of B are designated by subscript numbers from right to left , with subscript 0 denoting the least significant bit. The carries are connected in a chain through the full adders. The input carry to the adder is C_o. and it ripples through the full adders to the output carry C_4. The S outputs generate the required sum bits. An n-bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.

To demonstrate with a specific example, consider the two binary numbers A = 1011 and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:

Subscript i:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

The bits are added with full adders, starting from the least significant position (subscript 0), to form the sum bit and carry bit. The input carry C_0 in the least significant position must be 0. The value of C_{i+1} in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left. The sum bits are thus generated starting from the rightmost position and are available as soon as the corresponding previous carry bit is generated. All the carries must be generated for the correct sum bits to appear at the outputs.

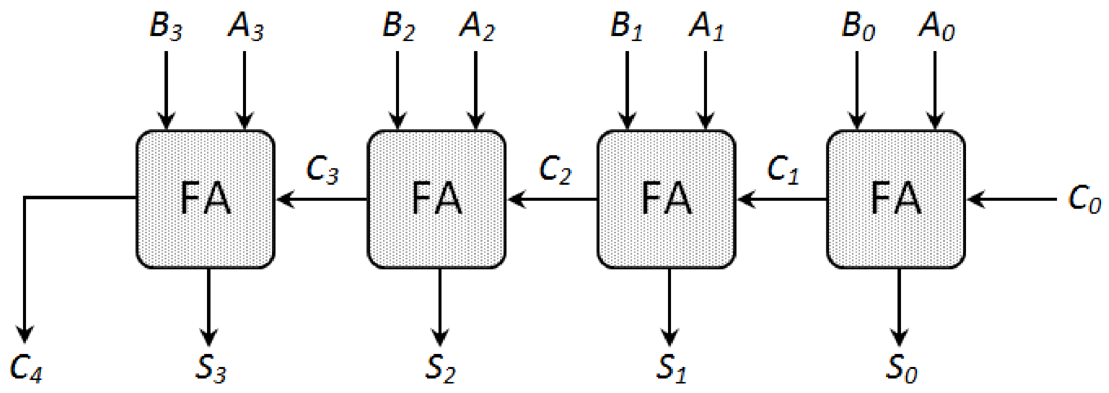


Figure (3.6) Parallel 4-bit binary Adder

The four-bit adder is a typical example of a standard component [19]. The circuit in the figure (3.6) performs calculation of two binary numbers of digits $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$. Initial carry $C_0 = 0$.

3.3 Quantum Addition

Obviously, we can’t directly implement carry-save adders with quantum gates, since the classical version of this element is clearly not unitary. It doesn’t have as many outputs as inputs, so it can’t be reversible. It’s easy to see that even adding a third output isn’t enough to make the full adder reversible. The truth table for the classical full adder (see above) has 3 inputs which map to a sum and carry of 1 and 0 respectively, and 3 inputs which map to a sum and carry of 0 and 1 respectively. Adding one bit obviously can’t distinguish among 3 values. So we have to add two outputs, making the quantum equivalent of the classical full adder a 4-input, 4-output device [36, 42].

Binary adders are a key element in any arithmetic logic unit. It is therefore important to have test reversible binary adder. This section shows a complete set of reversible adders for *1-bit* and *n-bit* binary numbers constructed from *n-bit* CNOT gates.

The following terminology is used in this paper where number denotes either a binary number. A *full-adder* is a logic circuit that takes as input two numbers A and B and a carry C_{in} and produces as output their sum S and a carry-out C_{out}. An adder that takes only A and B as input is called *half-adder.*

A *1-bit half-adder* takes two binary digits (A, B) as input and is described by the logic equations

$$Sum=A\oplus B$$

$$C_{out}=AB$$

Therefore we can construct a reversible half-adder (Figure 3.7) by using two reversible gates. This gate combination corresponds to a Peres gate.

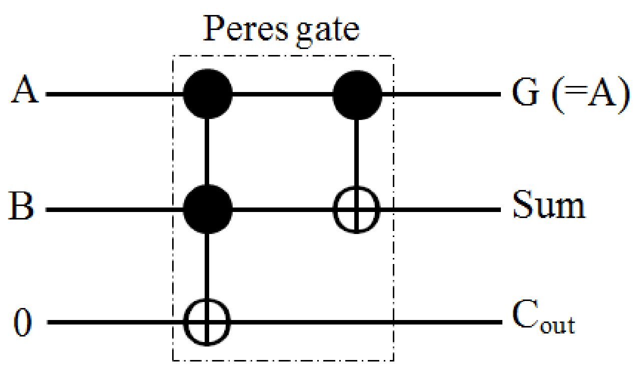


Figure (3.7) Reversible 1-bit half-adder

A 1-bit full-adder takes two binary digits (A, B) and a carry-in (C_{in}) as input is described by equations:

$$Sum = A \oplus B \oplus C_{in}$$

$$C_{out} = AB \oplus (A \oplus B)C_{in}$$

A reversible full-adder (Fig. 3.8) can be constructed from four gates (two Toffoli gates and two Feynman gates) and has two garbage bits. The gate combination can be replaced by two Peres as indicated in the figure (the input line of the second Peres gate are permuted to avoid a cross-over of lines). Between the first use of C_{in} and C_{out} is only one Toffoli gate. This property can be used to construct a fast n-bit adder by immediately propagating the carry-out [36-37].

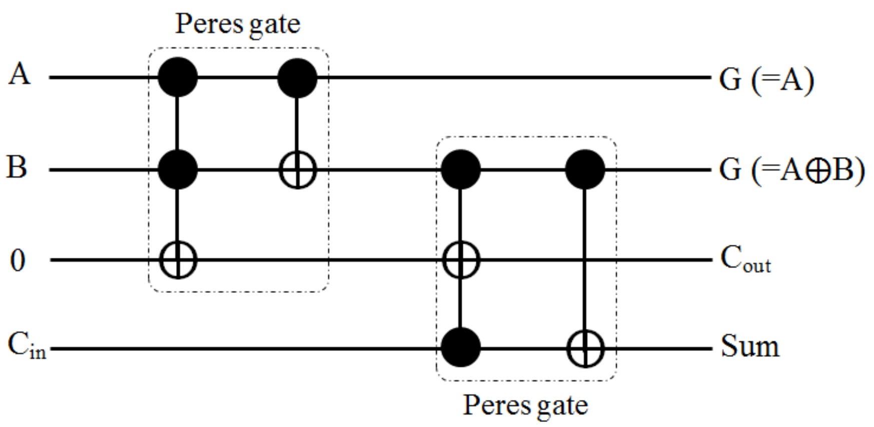


Figure (3.8) Reversible 1-qubit full-adder

Appropriate combinations of the 1-bit half and full-adder, provides a reversible *n*-bit half and full-adders.

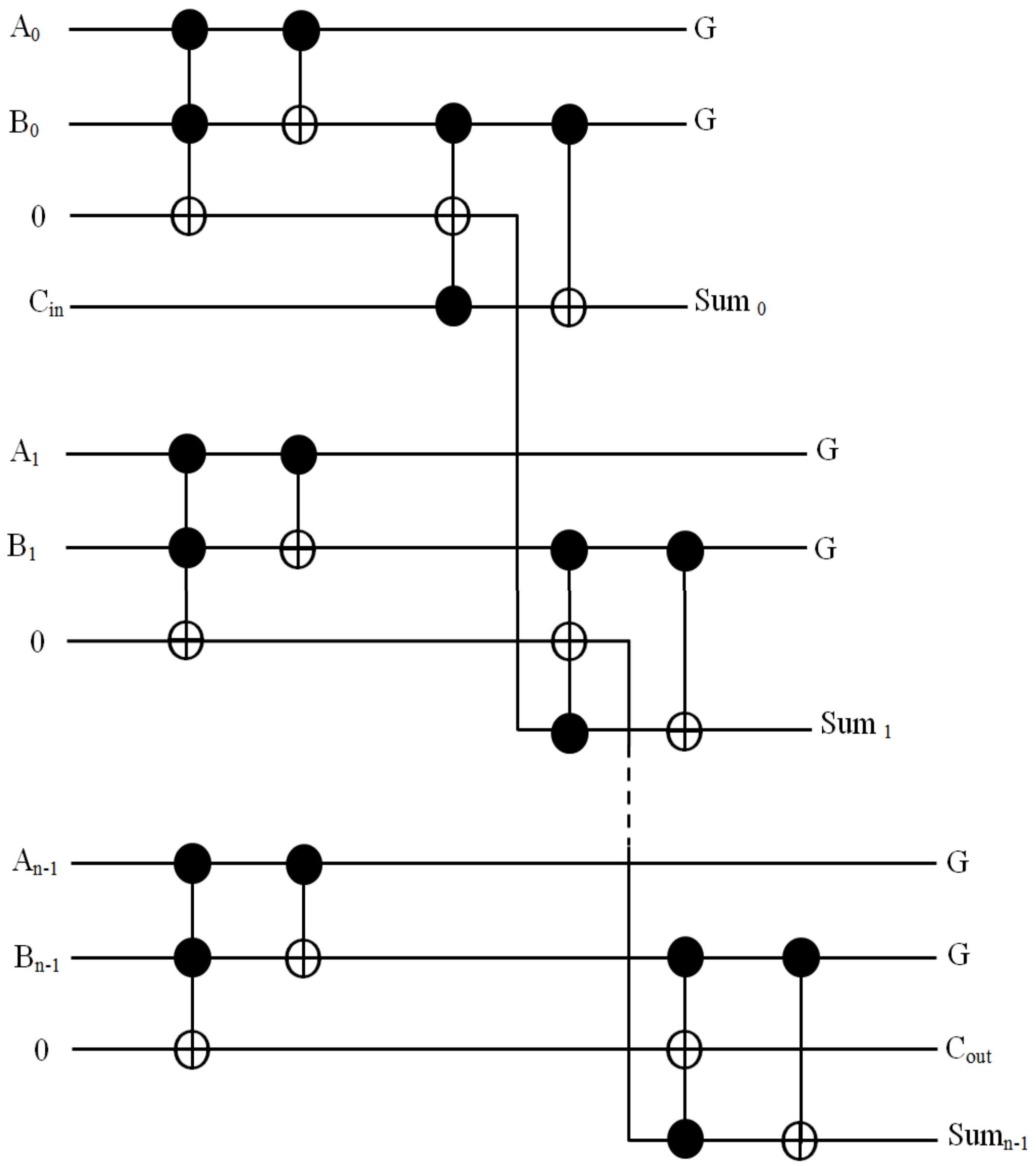


Figure (3.9) Reversible *n*-qubit full adder

Table (3.3) Input combinations that produce the same output combinations in full adder circuit (shown shaded).

Input				Output			
A	B	C_{in}	C1	S	C_{out}	G1	G2
0	0	1	0	1	0	0	0
0	1	0	0	1	0	0	1
1	0	0	0	1	0	1	0

We have shown that a reversible full adder circuit can be constructed by using CNOT and Controlled CNOT gates represented by a unitary matrix.

Chapter 4

SIMULATION OF 4-QUBITS FULL ADDER CIRCUIT BY MATHEMATICA

In fact, possible to use many different ways or different programs to simulation quantum computation, such (C/C^{++}, CaML, Java, MATLAB/Octave, Maple, Mathcad, Maxima and Mathematica..... etc.), we choose the Mathematica because it is easiest and quickest way to simulate this thesis [47].

A Mathematica for Dirac Notation, Non-commutative Algebra of Operators and commutations, Quantum Computing and Plotting of Quantum Circuits [41].

Without the ability to simulate quantum circuits, the states of qubits in quantum circuits would have to be determined by hand or through a physical quantum circuit. In order to fully utilize all that quantum circuits have to offer, it is necessary to design a circuit simulator that is both efficient and accurate [8].

4.1 Installation Program on Micro-Soft Windows:

After installation Mathematica program in [40], Files must be located in the proper file location in order to be able to load the Quantum packages with the Needs command and to have the documentation in *Mathematica*'s Documentation Center.

All the files will be located inside the directory specified by the value of *Mathematica*'s variable $UserBaseDirectory. In the computer were this document was generated, this is the value of $UserBaseDirectory (when you write $UserBaseDirectory and press the at the same time the keys {SHIFT – ENTER} in your *Mathematica*, you will get the $UserBaseDirectory in your computer, which will be different from the one shown in this example):

"C:\\Documents and Settings\\jose.luis.gomez\\Datos de programa\\Mathematica"

Like the figure (4.1) you can see the Windows Explorer opened in the $UserBaseDirectory of my computer. Remember that in your computer it will be a different location, the one that you obtain when you evaluate $UserBaseDirectory in your *Mathematica.* Important: In order to be able to see some of the folders, you might have to select "Tools", "Folder Options", "View", "Show Hidden Files and Folders"

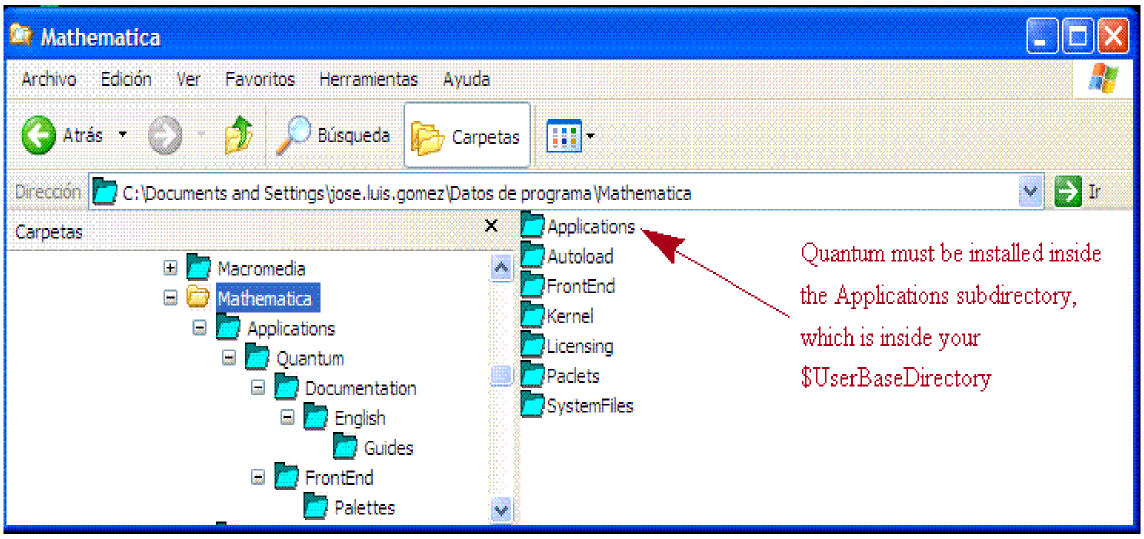
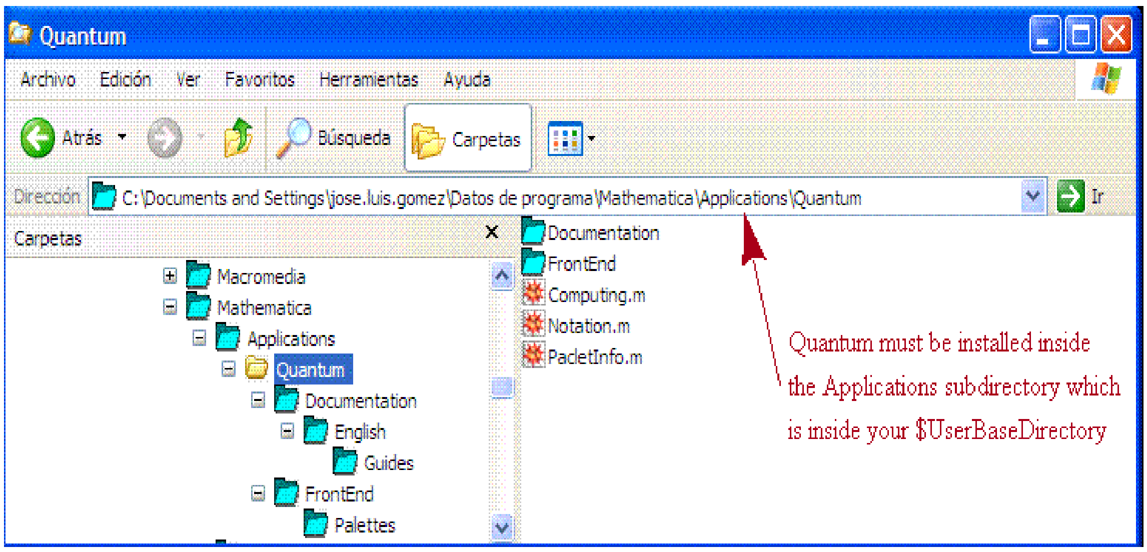

Mathematica
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás
Búsqueda
Carpetas
Dirección
C:\Documents and Settings\jose.luis.gomez\Datos de programa\Mathematica
Ir
Carpetas
Macromedia
Mathematica
Applications
Quantum
Documentation
English
Guides
FrontEnd
Palettes
Applications
Autoload
FrontEnd
Kernel
Licensing
Paclets
SystemFiles
Quantum must be installed inside the Applications subdirectory, which is inside your $UserBaseDirectory

Figure (4.1) First step shown Installation Mathematica Add-On Program

Inside your $UserBaseDirectory there must be an Applications directory, and inside the Applications directory you must unzip the file Quantum.zip, so that a Quantum directory is created. Once the unzip procedure is finished, inside the Quantum directory there must be the files Computing.m and Notation.m, which contain the programs of this Add-On, the file PacletInfo.m, which is necessary to incorporate documentation in *Mathematica*'s help system, and the directories (folders) Documentation and FrontEnd. You can see the Quantum directory in your computer like the figure (4.2):



C:\Documents and Settings\jose.luis.gomez\Datos de programa\Mathematica\Applications\Quantum

Quantum must be installed inside the Applications subdirectory which is inside your $UserBaseDirectory

Figure (4.2) Final step shown Installation Mathematica Add-On Program

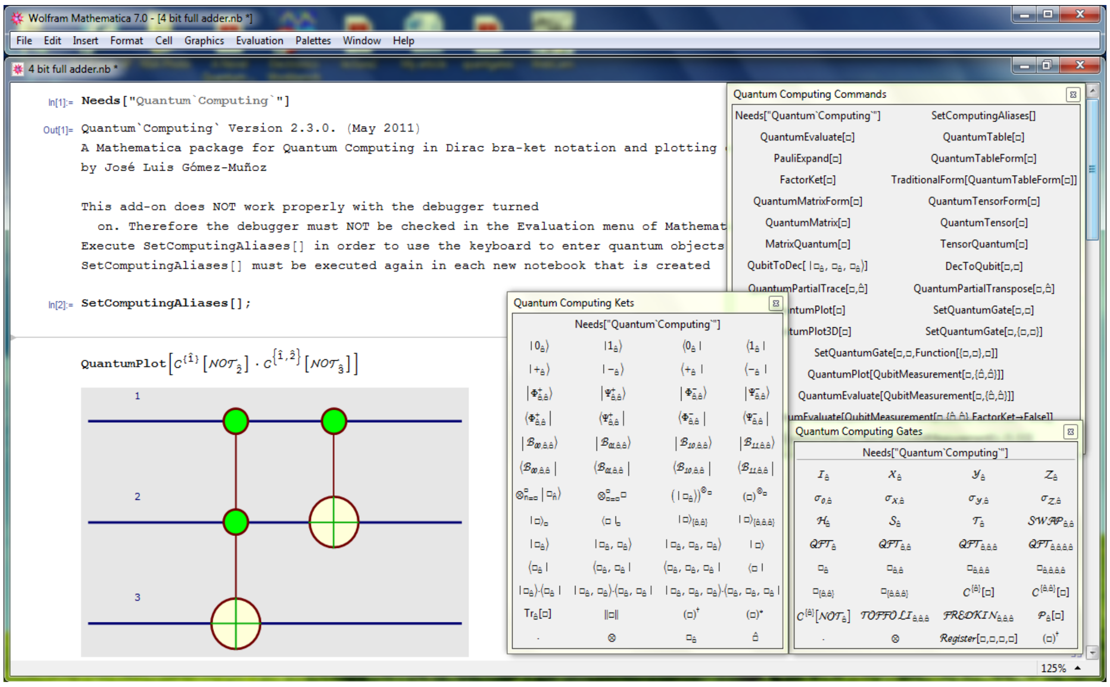
After having all the files in the proper directories you must quit and restart *Mathematica*. If the directory (folder) structure is the correct one, then you will obtain the "welcome" message after writing Needs ["Quantum`Notation`"] and pressing at the same time the keys {SHIFT – ENTER} to evaluate (the welcome message only appears the first time you execute Needs ["Quantum`Notation`"] in a fresh *Mathematica* session).

4.2 Mathematica add-on Program

This is a Mathematica add-on; this program is used to design and draw all different Quantum circuits and test their results operation. So this program is used for two purposes:

1. To draw the detailed circuit of the design
2. To test the design and make sure that the circuit working properly

The main page of the program looks like figure (4.3).



Wolfram Mathematica 7.0 - [4 bit full adder.nb *]
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
4 bit full adder.nb *
Quantum Computing Commands
Quantum Computing Kets
Quantum Computing Gates
125%

```
In[1]:= Needs["Quantum`Computing`"]

Out[1]= Quantum`Computing` Version 2.3.0. (May 2011)
        A Mathematica package for Quantum Computing in Dirac bra-ket notation and plotting
        by José Luis Gómez-Muñoz

        This add-on does NOT work properly with the debugger turned
          on. Therefore the debugger must NOT be checked in the Evaluation menu of Mathemat
        Execute SetComputingAliases[] in order to use the keyboard to enter quantum objects
        SetComputingAliases[] must be executed again in each new notebook that is created

In[2]:= SetComputingAliases[];
```

$\text{QuantumPlot}\left[C^{\{\hat{1}\}}\left[\mathcal{NOT}_{\hat{2}}\right]\cdot C^{\{\hat{1},\hat{2}\}}\left[\mathcal{NOT}_{\hat{3}}\right]\right]$

Figure (4.3) the main page of Mathematica add-on program

In [40] explain all parts of program, how to use and details explanation program.

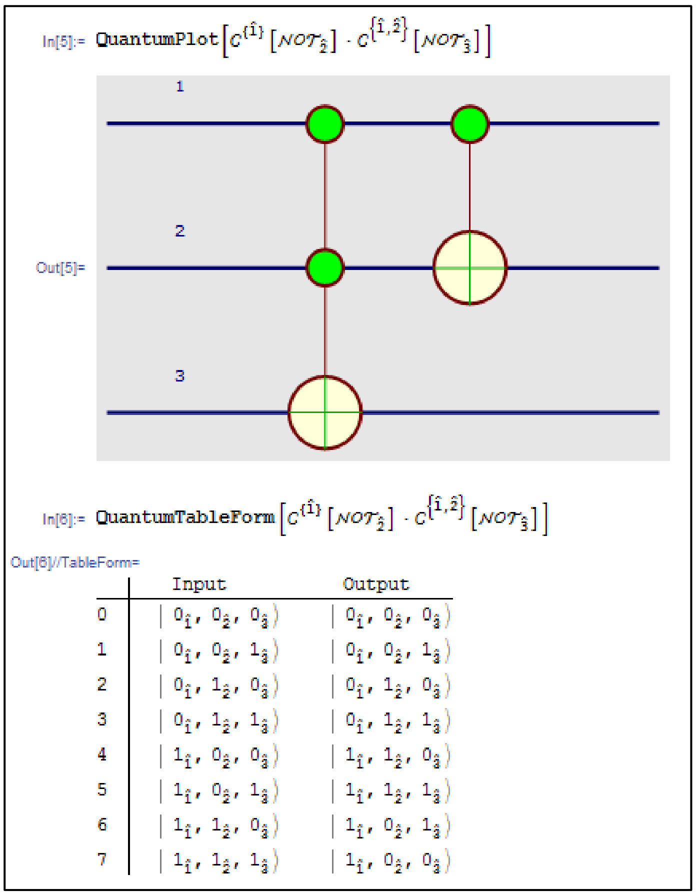
4.3 Quantum Circuit Simulator and result

Our task is now to simulate four bit quantum full adder circuit my Mathematica. A Mathematica Add-On package is presented for Dirac Notation, Noncommutative Algebra of Operators and Commutators, Quantum Computing and Plotting of Quantum Circuits [40]. In this paper we have simulated full adder circuit using the Mathematica Add-On package.

In order to fully utilize all that quantum circuits have to offer, it is necessary to design a circuit simulator that is both efficient and accurate [15]. We begin to design a half adder circuit.

4.3.1 Simulation Half-Adder

It is easy to use the program to construct a unitary circuit. Figure of the quantum circuit can be drawn by using the command **QuantumPlot[]**. Operation of the circuit on the qubits can be tabulated by using the command **QuantumTableForm[]**.



In[5]:= QuantumPlot$\left[C^{\{\hat{1}\}}[\mathcal{NOT}_{\hat{2}}] \cdot C^{\{\hat{1},\hat{2}\}}[\mathcal{NOT}_{\hat{3}}]\right]$

Out[5]=

In[6]:= QuantumTableForm$\left[C^{\{\hat{1}\}}[\mathcal{NOT}_{\hat{2}}] \cdot C^{\{\hat{1},\hat{2}\}}[\mathcal{NOT}_{\hat{3}}]\right]$

Out[6]//TableForm=

	Input	Output
0	$\|0_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}\rangle$	$\|0_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}\rangle$
1	$\|0_{\hat{1}}, 0_{\hat{2}}, 1_{\hat{3}}\rangle$	$\|0_{\hat{1}}, 0_{\hat{2}}, 1_{\hat{3}}\rangle$
2	$\|0_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}\rangle$	$\|0_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}\rangle$
3	$\|0_{\hat{1}}, 1_{\hat{2}}, 1_{\hat{3}}\rangle$	$\|0_{\hat{1}}, 1_{\hat{2}}, 1_{\hat{3}}\rangle$
4	$\|1_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}\rangle$	$\|1_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}\rangle$
5	$\|1_{\hat{1}}, 0_{\hat{2}}, 1_{\hat{3}}\rangle$	$\|1_{\hat{1}}, 1_{\hat{2}}, 1_{\hat{3}}\rangle$
6	$\|1_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}\rangle$	$\|1_{\hat{1}}, 0_{\hat{2}}, 1_{\hat{3}}\rangle$
7	$\|1_{\hat{1}}, 1_{\hat{2}}, 1_{\hat{3}}\rangle$	$\|1_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}\rangle$

Figure (4.4) Simulation Quantum Half-Adder with result by Mathematica

The half adder circuit and its operation are illustrated in figure (4.4). In the figure, lines 1, 2 and 3 represents input and output of the circuit. Synthesis of input-output relation of the circuit is summarized in table (4.1).

Table (4.1) Synthesis of input and output Quantum Half-Adder

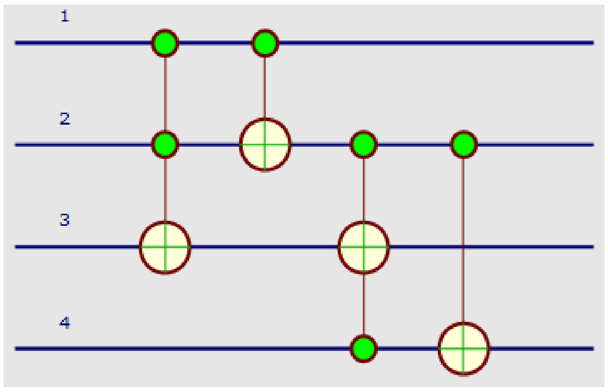
Input	Output
Line 1 = First Input bit (A)	Line 1 = Garbage
Line 2 = Second Input bit (B)	Line 2 = Sum
Line 3 = 0	Line 3 = C_{out}

4.3.2 Simulation Full-Adder

Similar to the design of half-adder circuit we can construct a full adder circuit. In the circuits input qubits are applied to lines 1 and 2. Input of the line 3 is always 0 and carry input is applied to line 4. Sum of the numbers are appears on output part of line 4 and carry appears on output line 3. Input and output relations are given in the table (4.2).

```
In[4]:= QuantumPlot[C^{2̂}[NOT_4̂] · C^{2̂,4̂}[NOT_3̂] · C^{1̂}[NOT_2̂] · C^{1̂,2̂}[NOT_3̂]]

Out[4]=
```



```
In[9]:= QuantumEvaluate[C^{2̂}[NOT_4̂] · C^{2̂,4̂}[NOT_3̂] · C^{1̂}[NOT_2̂] · C^{1̂,2̂}[NOT_3̂] · |0_1̂, 0_2̂, 0_3̂, 0_4̂⟩]

Out[9]=  |0_1̂, 0_2̂, 0_3̂, 0_4̂⟩

In[14]:= QuantumEvaluate[C^{2̂}[NOT_4̂] · C^{2̂,4̂}[NOT_3̂] · C^{1̂}[NOT_2̂] · C^{1̂,2̂}[NOT_3̂] · |0_1̂, 1_2̂, 0_3̂, 0_4̂⟩]

Out[14]=  |0_1̂, 1_2̂, 0_3̂, 1_4̂⟩

In[11]:= QuantumEvaluate[C^{2̂}[NOT_4̂] · C^{2̂,4̂}[NOT_3̂] · C^{1̂}[NOT_2̂] · C^{1̂,2̂}[NOT_3̂] · |1_1̂, 1_2̂, 0_3̂, 0_4̂⟩]

Out[11]=  |1_1̂, 0_2̂, 1_3̂, 0_4̂⟩

In[13]:= QuantumEvaluate[C^{2̂}[NOT_4̂] · C^{2̂,4̂}[NOT_3̂] · C^{1̂}[NOT_2̂] · C^{1̂,2̂}[NOT_3̂] · |1_1̂, 1_2̂, 0_3̂, 1_4̂⟩]

Out[13]=  |1_1̂, 0_2̂, 1_3̂, 1_4̂⟩
```

Figure (4.5) Simulation Quantum Full-adder with result by Mathematica

In order to evaluate action of the circuit on a given input state one can use the command **QuantumEvaluate[]**. Action of the full adder circuit on various states (qubits) is given in figure (4.5)

Table (4.2) Synthesis of input and output Quantum Full-Adder

Input	Output
Line 1 = First Input bit (A)	Line 1 = Garbage
Line 2 = Second Input bit (B)	Line 2 = Garbage
Line 3 = 0	Line 3 = C_{out}
Line 4 = C_{in}	Line 4 = Sum

Example 1:

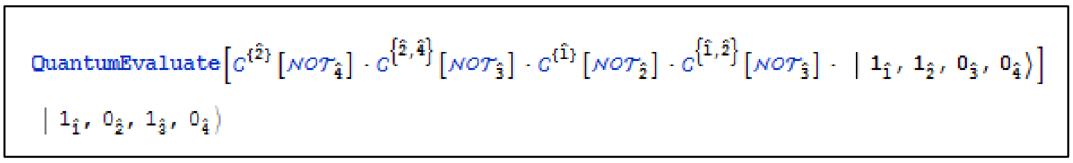
If addition two binary bit, Consider (A=1, B=1 and C_{in}=0) in quantum full-adder by using Mathematica Add-on.

1. Manually Solution:

		1	A
		1	B
+		0	C_{in}
	1	0	
	C_{out}	Sum	

2. Programing Solution:

The following Mathematica line illustrates summation of qubits (1) and (1) with (0) carry input. The sum is obtained by measuring the output 4 and carry can be determined by measuring output 3.



$$\text{QuantumEvaluate}\left[C^{\{\hat{2}\}}[\mathcal{NOT}_{\hat{4}}]\cdot C^{\{\hat{2},\hat{4}\}}[\mathcal{NOT}_{\hat{3}}]\cdot C^{\{\hat{1}\}}[\mathcal{NOT}_{\hat{2}}]\cdot C^{\{\hat{1},\hat{2}\}}[\mathcal{NOT}_{\hat{3}}]\cdot \left|1_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}, 0_{\hat{4}}\right\rangle\right]$$

$$\left|1_{\hat{1}}, 0_{\hat{2}}, 1_{\hat{3}}, 0_{\hat{4}}\right\rangle$$

Table (4.3) Input and Output result for example 1

Input	Result	Output Line
Line 1 = 1	1	garbage
Line 2 = 1	0	garbage
Line 3 = 0	1	C_{out}
Line 4 = 0	0	Sum

4.3.3 Simulation 4-Qubit Full Adder

Using the full adder circuit we can design 4 qubit quantum full adder circuits by writing the following code in Mathematica Add-On program.

The following code is writing to simulation of 4-qbit Adder plot the same figure (4.6) in mathematica Add-On program.

$$\begin{aligned}
&\texttt{QuantumPlot}\Big[C^{\{\hat{14}\}}[\mathcal{NOT}_{\hat{16}}]\cdot C^{\{\hat{14},\hat{16}\}}[\mathcal{NOT}_{\hat{15}}]\cdot C^{\{\hat{13}\}}[\mathcal{NOT}_{\hat{14}}]\cdot C^{\{\hat{13},\hat{14}\}}[\mathcal{NOT}_{\hat{15}}]\cdot\\
&\quad C^{\{\hat{10}\}}[\mathcal{NOT}_{\hat{12}}]\cdot C^{\{\hat{11}\}}[\mathcal{NOT}_{\hat{16}}]\cdot C^{\{\hat{10},\hat{12}\}}[\mathcal{NOT}_{\hat{11}}]\cdot C^{\{\hat{9}\}}[\mathcal{NOT}_{\hat{10}}]\cdot C^{\{\hat{9},\hat{10}\}}[\mathcal{NOT}_{\hat{11}}]\cdot\\
&\quad C^{\{\hat{6}\}}[\mathcal{NOT}_{\hat{8}}]\cdot C^{\{\hat{7}\}}[\mathcal{NOT}_{\hat{12}}]\cdot C^{\{\hat{6},\hat{8}\}}[\mathcal{NOT}_{\hat{7}}]\cdot C^{\{\hat{5}\}}[\mathcal{NOT}_{\hat{6}}]\cdot C^{\{\hat{5},\hat{6}\}}[\mathcal{NOT}_{\hat{7}}]\cdot\\
&\quad C^{\{\hat{2}\}}[\mathcal{NOT}_{\hat{4}}]\cdot C^{\{\hat{3}\}}[\mathcal{NOT}_{\hat{8}}]\cdot C^{\{\hat{2},\hat{4}\}}[\mathcal{NOT}_{\hat{3}}]\cdot C^{\{\hat{1}\}}[\mathcal{NOT}_{\hat{2}}]\cdot C^{\{\hat{1},\hat{2}\}}[\mathcal{NOT}_{\hat{3}}]\Big]
\end{aligned}$$

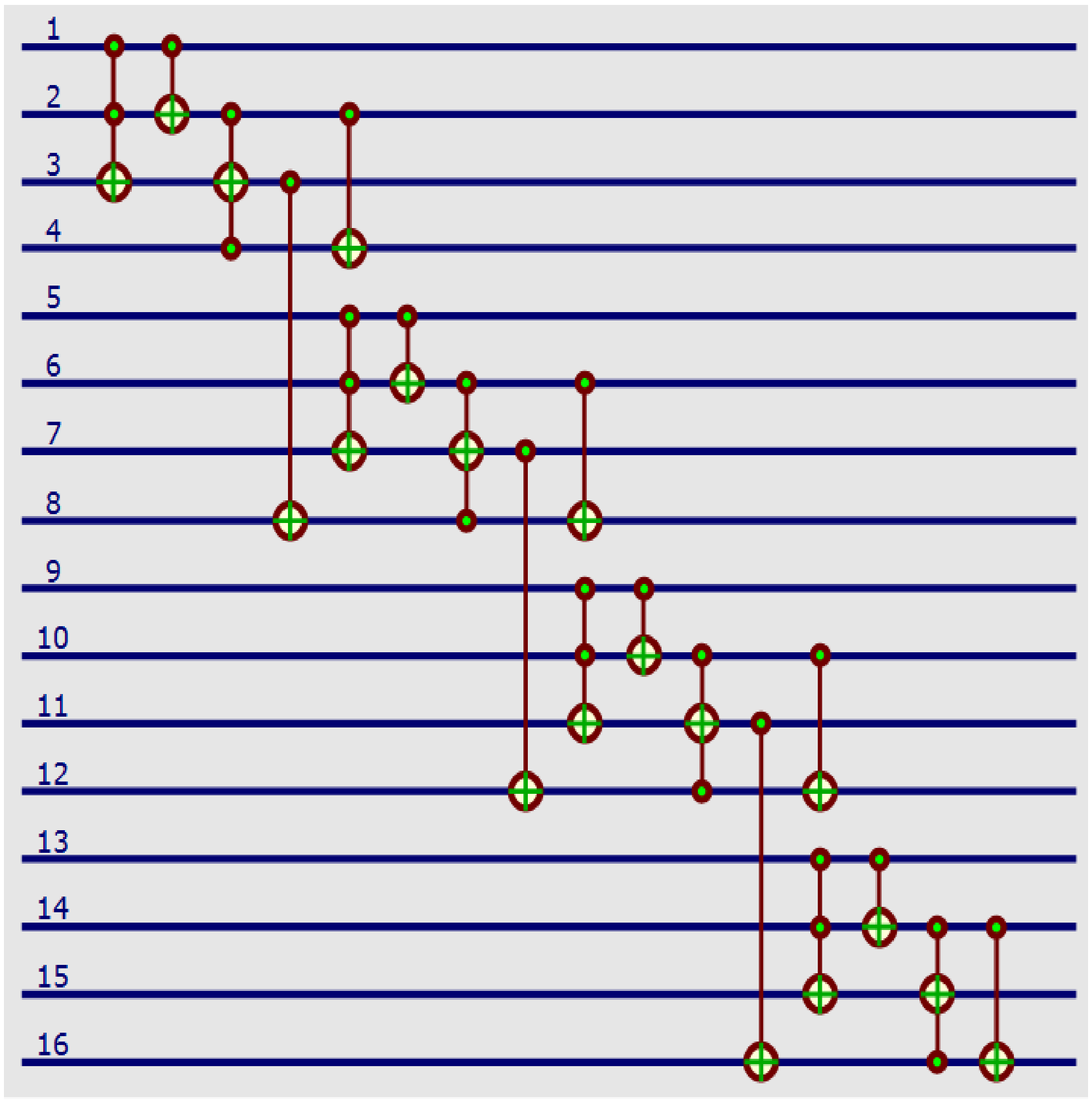


Figure (4.6) Simulation 4-qbit Adder by Mathematica

Output of the Mathematica code represented by figure of the quantum full adder circuit and it is given in figure 11. Actions of the circuit on input qubits are summarized in table (4.4).

Table (4.4) Synthesis of input and output 4-qbit Adder

Input	Output
Line 1 = First bit Input (A_0)	Line 1 = Garbage
Line 2 = Second bit Input (B_0)	Line 2 = Garbage
Line 3 = 0	Line 3 = C_{out}
Line 4 = C_{in}	Line 4 = Sum_0
Line 5 = First bit Input (A_1)	Line 5 = Garbage
Line 6 = Second bit Input (B_1)	Line 6 = Garbage
Line 7 = 0+C_{out}(Output line 3)	Line 7 = C_{out}
Line 8 = C_{in}	Line 8 = Sum_1
Line 9 = First bit Input (A_2)	Line 9 = Garbage
Line 10 = Second bit Input (B_2)	Line 10 = Garbage
Line 11 = 0+C_{out}(Output line 7)	Line 11 = C_{out}
Line 12 = C_{in}	Line 12 = Sum_2
Line 13 = First bit Input (A_3)	Line 13 = Garbage
Line 14 = Second bit Input (B_3)	Line 14 = Garbage
Line 15= 0+C_{out}(Output line 11)	Line 15 = C_{out}
Line 16 = C_{in}	Line 16 = Sum_3

Example 2:

If (1st No. = 1101) and (2nd No. = 0110),

find (1st +2nd =?).

1. Manual Solution:

1st No. = 1101 ➔ $A_0 = 1$, $A_1 = 1$, $A_2 = 0$ and $A_3 = 1$

2nd No. = 0110 ➔ $B_0 = 0$, $B_1 = 1$, $B_2 = 1$ and $B_3 = 0$

	3	2	1	0	
	1	1	0	1	A
+	0	1	1	0	B
1	0	0	1	1	Result
C_{out}	S_3	S_2	S_1	S_0	

2. Programing Solution:

The following Mathematica line illustrates summation take value of qubits. Then obtain Output result.

$$\texttt{QuantumEvaluate}\Big[C^{\{\hat{14}\}}[NOT_{\hat{16}}]\cdot C^{\{\hat{14},\hat{16}\}}[NOT_{\hat{15}}]\cdot C^{\{\hat{13}\}}[NOT_{\hat{14}}]\cdot C^{\{\hat{13},\hat{14}\}}[NOT_{\hat{15}}]\cdot$$
$$C^{\{\hat{10}\}}[NOT_{\hat{12}}]\cdot C^{\{\hat{11}\}}[NOT_{\hat{16}}]\cdot C^{\{\hat{10},\hat{12}\}}[NOT_{\hat{11}}]\cdot C^{\{\hat{9}\}}[NOT_{\hat{10}}]\cdot C^{\{\hat{9},\hat{10}\}}[NOT_{\hat{11}}]\cdot$$
$$C^{\{\hat{6}\}}[NOT_{\hat{8}}]\cdot C^{\{\hat{7}\}}[NOT_{\hat{12}}]\cdot C^{\{\hat{6},\hat{8}\}}[NOT_{\hat{7}}]\cdot C^{\{\hat{5}\}}[NOT_{\hat{6}}]\cdot C^{\{\hat{5},\hat{6}\}}[NOT_{\hat{7}}]\cdot$$
$$C^{\{\hat{2}\}}[NOT_{\hat{4}}]\cdot C^{\{\hat{3}\}}[NOT_{\hat{8}}]\cdot C^{\{\hat{2},\hat{4}\}}[NOT_{\hat{3}}]\cdot C^{\{\hat{1}\}}[NOT_{\hat{2}}]\cdot$$
$$C^{\{\hat{1},\hat{2}\}}[NOT_{\hat{3}}]\cdot \left|1_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}, 0_{\hat{4}}, 0_{\hat{5}}, 1_{\hat{6}}, 0_{\hat{7}}, 0_{\hat{8}}, 1_{\hat{9}}, 1_{\hat{10}}, 0_{\hat{11}}, 0_{\hat{12}}, 1_{\hat{13}},\right.$$
$$\left.0_{\hat{14}}, 0_{\hat{15}}, 0_{\hat{16}}\right\rangle\Big]$$

$$\left|1_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}, 1_{\hat{4}}, 0_{\hat{5}}, 1_{\hat{6}}, 0_{\hat{7}}, 1_{\hat{8}}, 1_{\hat{9}}, 0_{\hat{10}}, 1_{\hat{11}}, 0_{\hat{12}}, 1_{\hat{13}}, 1_{\hat{14}}, 1_{\hat{15}}, 0_{\hat{16}}\right\rangle$$

Table (4.5) Input and Output result for example 2

Input	Output Result	Output
Line 1 = First bit Input (A_0)	1	Line 1 = Garbage
Line 2 = Second bit Input (B_0)	1	Line 2 = Garbage
Line 3 = 0	0	Line 3 = C_{out}
Line 4 = C_{in}	1	Line 4 = Sum_0
Line 5 = First bit Input (A_1)	0	Line 5 = Garbage
Line 6 = Second bit Input (B_1)	1	Line 6 = Garbage
Line 7 = 0+C_{out}(Output line 3)	0	Line 7 = C_{out}
Line 8 = C_{in}	1	Line 8 = Sum_1
Line 9 = First bit Input (A_2)	1	Line 9 = Garbage
Line 10 = Second bit Input (B_2)	0	Line 10 = Garbage
Line 11 = 0+C_{out}(Output line 7)	1	Line 11 = C_{out}
Line 12 = C_{in}	0	Line 12 = Sum_2
Line 13 = First bit Input (A_3)	1	Line 13 = Garbage
Line 14 = Second bit Input (B_3)	1	Line 14 = Garbage
Line 15= 0+C_{out}(Output line 11)	1	Line 15 = C_{out}
Line 16 = C_{in}	0	Line 16 = Sum_3

Example 3:

If (1st No. = 1011) and (2nd No. = 1110),

find (1st +2nd =?).

1. Manually Solution:

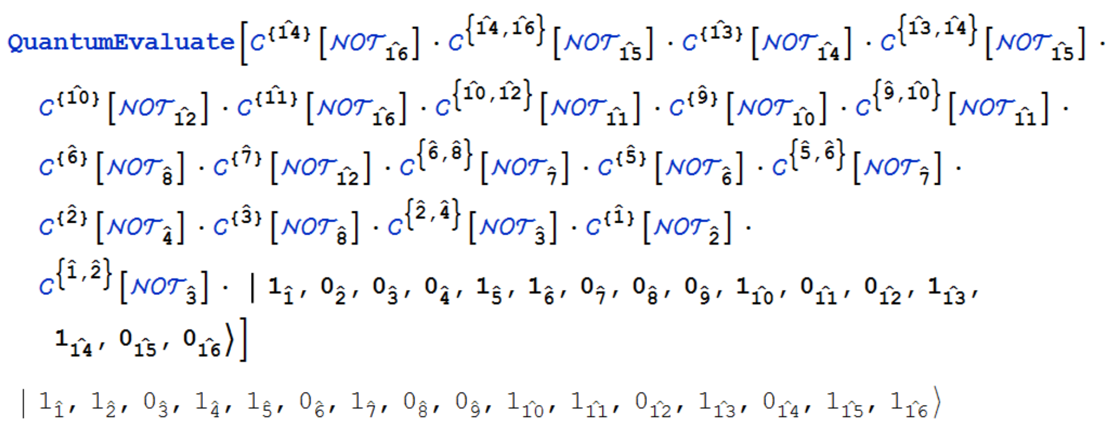
1st No. = 1011 ➔ $A_0 = 1$, $A_1 = 1$, $A_2 = 0$ and $A_3 = 1$

2nd No. = 1110 ➔ $B_0 = 0$, $B_1 = 1$, $B_2 = 1$ and $B_3 = 1$

	3	2	1	0	
	1	0	1	1	A
+	1	1	1	0	B
1	1	0	0	1	Result
C_{out}	S_3	S_2	S_1	S_0	

2. Programing Solution:

The following Mathematica line illustrates summation take value of qubits. Then obtain Output result.



$$\text{QuantumEvaluate}\Big[C^{\{\hat{14}\}}[NOT_{\hat{16}}]\cdot C^{\{\hat{14},\hat{16}\}}[NOT_{\hat{15}}]\cdot C^{\{\hat{13}\}}[NOT_{\hat{14}}]\cdot C^{\{\hat{13},\hat{14}\}}[NOT_{\hat{15}}]\cdot C^{\{\hat{10}\}}[NOT_{\hat{12}}]\cdot C^{\{\hat{11}\}}[NOT_{\hat{16}}]\cdot C^{\{\hat{10},\hat{12}\}}[NOT_{\hat{11}}]\cdot C^{\{\hat{9}\}}[NOT_{\hat{10}}]\cdot C^{\{\hat{9},\hat{10}\}}[NOT_{\hat{11}}]\cdot C^{\{\hat{6}\}}[NOT_{\hat{8}}]\cdot C^{\{\hat{7}\}}[NOT_{\hat{12}}]\cdot C^{\{\hat{6},\hat{8}\}}[NOT_{\hat{7}}]\cdot C^{\{\hat{5}\}}[NOT_{\hat{6}}]\cdot C^{\{\hat{5},\hat{6}\}}[NOT_{\hat{7}}]\cdot C^{\{\hat{2}\}}[NOT_{\hat{4}}]\cdot C^{\{\hat{3}\}}[NOT_{\hat{8}}]\cdot C^{\{\hat{2},\hat{4}\}}[NOT_{\hat{3}}]\cdot C^{\{\hat{1}\}}[NOT_{\hat{2}}]\cdot C^{\{\hat{1},\hat{2}\}}[NOT_{\hat{3}}]\cdot \left|1_{\hat{1}}, 0_{\hat{2}}, 0_{\hat{3}}, 0_{\hat{4}}, 1_{\hat{5}}, 1_{\hat{6}}, 0_{\hat{7}}, 0_{\hat{8}}, 0_{\hat{9}}, 1_{\hat{10}}, 0_{\hat{11}}, 0_{\hat{12}}, 1_{\hat{13}}, 1_{\hat{14}}, 0_{\hat{15}}, 0_{\hat{16}}\right\rangle\Big]$$

$$\left|1_{\hat{1}}, 1_{\hat{2}}, 0_{\hat{3}}, 1_{\hat{4}}, 1_{\hat{5}}, 0_{\hat{6}}, 1_{\hat{7}}, 0_{\hat{8}}, 0_{\hat{9}}, 1_{\hat{10}}, 1_{\hat{11}}, 0_{\hat{12}}, 1_{\hat{13}}, 0_{\hat{14}}, 1_{\hat{15}}, 1_{\hat{16}}\right\rangle$$

Table (4.6) Input and Output result for example 3

Input	Output Result	Output
Line 1 = First bit Input (A_0)	1	Line 1 = Garbage
Line 2 = Second bit Input (B_0)	1	Line 2 = Garbage
Line 3 = 0	0	Line 3 = C_{out}
Line 4 = C_{in}	1	Line 4 = Sum_0
Line 5 = First bit Input (A_1)	1	Line 5 = Garbage
Line 6 = Second bit Input (B_1)	0	Line 6 = Garbage
Line 7 = 0+C_{out}(Output line 3)	1	Line 7 = C_{out}
Line 8 = C_{in}	0	Line 8 = Sum_1
Line 9 = First bit Input (A_2)	0	Line 9 = Garbage
Line 10 = Second bit Input (B_2)	1	Line 10 = Garbage
Line 11 = 0+C_{out}(Output line 7)	1	Line 11 = C_{out}
Line 12 = C_{in}	0	Line 12 = Sum_2
Line 13 = First bit Input (A_3)	1	Line 13 = Garbage
Line 14 = Second bit Input (B_3)	0	Line 14 = Garbage
Line 15= 0+C_{out}(Output line 11)	1	Line 15 = C_{out}
Line 16 = C_{in}	1	Line 16 = Sum_3

Chapter 5

CONCLUSION

The most important results and achievements of this study are:

- Attentively studying Classical Computation, especially Number Systems and Digital System for obtaining Half-Adder and Full-Adder. Through Half-Adder and Full-Adder, 4-bit Parallel Adder is achieved and developed into n-bit Parallel Adder and a variety of other circuits in different areas.
- Studying the new science of Quantum Computation and all its main components. Familiarizing with all the Qubit cases, all the properties of Quantum Gates, particularly the input and output of each one of them, which help with the designing of many other different quantum circuits.
- Obtaining the output of each quantum circuit through familiarization with the input and output of quantum gates.
- Designing Quantum Half-Adder and Quantum Full-Adder and achieving the output of all the cases.
- Designing 4-qubit adder quantum circuit and providing many cases from the input cases and obtaining the output of the cases.
- Formulating all the above-mentioned processes in Quantum Computation by using Mathematic program, particularly in the area of Drawing Circuit and achieving remarkable results.

- All these will result in the development of Quantum Computation science and combining it with Mathematica program. Using this program for all the Algorithms of this science which have thus far been discovered.

  Additionally, The Mathematica add-on presented in this paper utilizes an irreducible form of output decomposition of a general controlled quantum gate with addition conditionals and is highly efficient in simulating complex quantum circuit. Another important application in which large and complex circuit need to be efficiently simulated is in the area of quantum error correction. This demonstrates a part of a general framework for simulation of quantum computers on classical computers.

REFERENCES

[1] Mades, J. E. (1999). *Quantum Computers and their impact on DoD in the 21st century*, Master thesis, Naval Postgraduate School, Monterey- California.

[2] Bellac, M. L. (2006). *A Short Introduction to Quantum Information and Quantum Computation* (English, Cambridge University Press.

[3] Khan, M. (2008). A recursive method for synthesizing quantum/reversible quaternary parallel adder/subtractor with look-ahead carry, *Journal of Systems Architecture,* **54** , 1113–1121.

[4] Grover, L. K. (1996). A Fast Quantum Mechanical Algorithm for Database Search, *Proc. 28th Annual ACM Symposium on the Theory of Computing,* USA, PA, 212-219.

[5] Shor, P. W. (1996). Polynomial Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *Proc. of the 35th Annual Symposium on Foundations of Computer Science (IEEE)*, Santa Fe, NM, 124-134.

[6] Saeedi, M., Zamani, M., Sedighi, M. (2007). Algebraic Characterization of CNOT-Based Quantum Circuits with its Applications on Logic Synthesis, *Proc. 10th IEEE Euromicro Conf. on Digital System Design Architectures*, Washington DC, USA, 339-346.

[7] Maity, S., Pal, A., Roy, T., Mandal, S., Chakrabarti, A. (2010). Design of an Efficient Quantum Circuit Simulator, *Proc. 10th IEEE International*

Symposium on Electronic System Design, Washington DC, USA, 50-55.

[8] D. Goodman, *A Quantum Circuit Simulator Based on Decision Diagrams*, Master thesis, Southern Methodist University, USA, 2007.

[9] Draper, T.G. (2000). Addition on a Quantum Computer, *arXiv:quant-ph/0008033v1*.

[10] Miquel, C., Paz, J., Perazzo, R. (1996). Factoring in a dissipative quantum computer, *American Physical Society Journals,* **54**, 2605–2613.

[11] Mohammadi, M., Eshghi, M., Haghparast, M., Bahrololoom, A. (2008). Design and Optimization of Reversible BCD Adder/Subtractor Circuit for Quantum and Nanotechnology Based Systems, *World Applied Sciences Journa,* **14**, 787-792.

[12] Berman, G., P., Doolen, G., D., Lopez, G., V., Tsifrinovich, V., I. (2001). A Quantum Full Adder for a Scalable Nuclear Spin Quantum Computer, *arXiv:quant-ph/0105133v1*.

[13] Mathematica program Installation. Available at: http://www.wolfram.com. Accessed 14.05.2012

[14] Julia-Daz, B., Burdis, J., Tabakin, F. (2006). Qdensity-A Mathematica Quantum Computer Simulation, *Computer Physics Communications,* **174**, 914–934.

[15] Nyman, P. (2008). *Representation of Quantum Algorithms with Symbolic Language and Simulation on Classical Computer*, Master Thesis, Vaxjo University, Sweden.

[16] Quantum computer. Available at: http://en.wikipedia.org/wiki/Quantum_computer. Accessed 25.08.2012.

[17] Morris, M. M. (2002) Digital Design. 3rd Edition. USA: Prentice Hall.

[18] Ali, S. N. (2003). Digital Electronics: Circuits, Systems and ICs. 3rd Edition. New Delhi: Galgotia Publication Pvt. Ltd.

[19] Thomas, L. F. (2006). Digital Fundamentals. 9th Edition. USA: Prentice Hall.

[20] Anil, K. M. (2007). *Digital Electronics*
(Principles, Devices and Applications). England : John Wiley & Sons Ltd.

[21] Thomas, C. B. (2003). Digital Computer fundamentals. 5th Edition. New York: McGraw-Hill Book Co.

[22] Robert, KD. (2005). Digital Design with CPLD Application and VHDL. 2nd Edition. New York: Thomson Delmar Learning.

[23] Wang, Y. (2012). Quantum Computation and Quantum Information, *IMS Journal of Statistical Science,* **27**, 373-394.

[24] Aharonov, D. (1998). Quantum Computation, *arXiv:quant-ph/9812037v1.*

[25] Islam, S., Islam, R. (2005). Minimization of Reversible Adder Circuits, *Asian Journal of Information Technology,* **4**, 1146-1151.

[26] Loke, T., Wang, J. (2011). An Efficient Quantum Circuit Analyser on Qubit and Qudits, *Computer Physics Communications,* **182**, 2285–2294.

[27] Quantum Bit, Available at: http://www.thefreedictionary.com/quantum+bit. Accessed 05.08.2012.

[28] Maslov, D., Dueck, G., Miller, D., Negrevergne, C. (2008). Quantum Circuit Simplification and Level Compaction, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* **27**, 436-444.

[29] Muthukrishnan, A. (1999). Classical and Quantum Logic Gates, *Seminar, Rochester Center for Quantum Information (RCQI).*

[30] Kak, S. (2006). Information Complexity of Quantum Gates, *International Journal of Theoretical Physics,* **45**, 933-941.

[31] D. Goodman, M.A. Thornton, D.Y. Feinstein, and D.M. Miller, Quantum Logic Circuit Simulation Based on the QMDD Data Structure, Proc. of the Workshop on Applications of the Reed Muller Expansion in Circuit Design and Representations and Methodology of Future Computing Technology,16, 99-105.

[32] Shende, V., Bullock, S., Markov, L. (2006). Synthesis of Quantum-Logic Circuits, *IEEE Trans. on Computer-Aided Design,* **25**, 1000 – 1010.

[34] Ronald, J., Tocci, N., Greg, M. (2007). Digital Systems: Principles and Applications. 10th Edition. USA: Prentice Hall.

[35] John, H. The Electronics Club/Logic Gates. Available at: www.kpsec.freeuk.com . Accessed 24.07.2012.

[36] Gossett, P. (1998). Quantum Carry-Save Arithmetic, *arXiv:quant-ph/9808061v2*.

[37] Kirkedal, M., Gluck, R. (2008). Optimized reversible binary-coded decimal adders, *Journal of Systems Architecture,* **54**, 697–706.

[38] Islam, S., Rahman, M., Begum, Z., Hafiz, M. (2010). Realization of a Novel Fault Tolerant Reversible Full Adder Circuit in Nanotechnology, *The International Arab Journal of Information Technology,* **7**, 317-323.

[39] Fahdil, M., Al-Azawi, A., Said, S. (2010). Operations Algorithms on Quantum Computer. *International Journal of Computer Science and Network Security (IJCSNS),* **10**, 85-95.

[40] Quantum computation –Mathematica Add-on. Available at: http://homepage.cem.itesm.mx/lgomez/quantum/. 20.08.2012.

[41] Michael, A. N., Chuang, I. L. (2000). Quantum Computation and Quantum Information, Cambridge, UK: Cambridge University.

[42] Murali, K. V., Sinha, M. N., Mahesh, T. S., Levitt, M. H., Ramanathan, K.V., Kumar, K. A. (2002). Quantum-information processing by nuclear magnetic resonance: Experimental implementation of half-adder and subtractor operations using an oriented spin-7/2 system, *American Physical Society (APS),* **66**, 1050-2947.

[43] Haghparast, M., Navi, K. (2008). A Novel Fault Tolerant Reversible Gate for Nanotechnology Based Systems, *American Journal of Applied Sciences,* **5**, 519-523.

[44] Peter, J. (2008). Digital Design (Verilog): An Embedded Systems Approach Using Verilog. Burling MA 01803, USA: Denise E. M. Penrose.

[45] Basic Gates and Functions. Available at: http://www.ee.surrey.ac.uk/Projects/Labview/gatesfunc/index.html. Accessed 02.08.2012.

[46] Quantum computation –Mathematica Add-on. Available at: http://www.quantiki.org/wiki/List_of_QC_simulators. Accessed 15.07.2012.