MODELING QUANTUM CIRCUITS

A graduate project submitted in partial fulfillment of the requirements
For the degree of Master of Science
in Computer Science

By
Margareta Vamos

December 2011
The graduate project of Margareta Vamos is approved:

______________________________  ______________
Professor John Noga           Date

______________________________  ______________
Professor Jeff Wiegley         Date

______________________________  ______________
Professor Richard Lorentz, Chair Date

California State University, Northridge
**TABLE OF CONTENTS**

Signature Page........................................................................................................................................... ii

**ABSTRACT**............................................................................................................................................... v

1. Theoretical notions...................................................................................................................................... 1
   Quantum Physics ..................................................................................................................................... 1
   Physics and the Theory of Computation ................................................................................................. 1
      Probabilistic Turing Machine .............................................................................................................. 2
      Quantum Turing Machine .................................................................................................................. 3
   Dirac Notation ....................................................................................................................................... 3
   Quantum Physical Systems ..................................................................................................................... 3
   Principles of Quantum Theory (Matrix Mechanics) .............................................................................. 4
      First principle: A quantum system may be represented by a vector ................................................. 5
      Second principle: The orientation of the vector representing a quantum system evolves in time:..... 5
      Third principle: Kets are transformed into other kets by means of operations that reveal an
      observational property ......................................................................................................................... 5
      Fourth Principle: Uncertainty Principle ............................................................................................... 6
   Qubit ......................................................................................................................................................... 6
   Quantum Entanglement .......................................................................................................................... 7
   Quantum Gates ....................................................................................................................................... 9
   Quantum Circuit .................................................................................................................................... 12
   Quantum Algorithms ............................................................................................................................. 13
   Deutsch’s Algorithm .............................................................................................................................. 14
   Deutsch-Jozsa Algorithm ....................................................................................................................... 20

2. Implementation......................................................................................................................................... 21
   Description ............................................................................................................................................. 21
   Requirements ......................................................................................................................................... 22
   Design: .................................................................................................................................................. 23
      Component Diagram .......................................................................................................................... 23
      Class Diagrams ................................................................................................................................. 24
      Activity Diagrams ............................................................................................................................... 28
   User Interface ....................................................................................................................................... 29

iii
Unit Testing.................................................................................................................................30
3. Results........................................................................................................................................32
4. Conclusions..................................................................................................................................35
Works Cited......................................................................................................................................36
ABSTRACT

MODELING QUANTUM CIRCUITS

By

Margareta Vamos

Master of Science in Computer Science

Quantum computers will be capable of new modes of computation which today’s computers are not capable of. The quantum theory of computation is the general theory of computation. The Turing model can describe completely the functioning of the existing computers. However, this model is just the classical approximation of the general quantum model.

To understand the Quantum Computation we don't need to understand quantum physics in detail. This paper presents some basic notions of quantum theory as the underlying foundation for quantum computing. The purpose of listing these notions is only to show that quantum computing has a solid theoretical foundation in quantum mechanics.

Quantum computing basic components are quantum gates that can be used to form quantum circuits. Quantum algorithms are using parallel computation based on quantum entanglement. Because of state superposition a quantum processor can use all input values at the same time. To explore the full potential of quantum parallelism, new algorithms have been expressed mathematically. Although practical quantum computing is still in its early stages, it can represent the future of computation.

The application provides means to graphically design a quantum circuit using quantum gates and to simulate a quantum algorithm represented by a quantum circuit. The user can choose a gate or a quantum gate from a list of elements. The gates have a number or a range of input and output pins, and some configurable properties. The user can drag and drop quantum gates to the project panel and connect them with quantum wires. The user will be able to run the circuit and evaluate the results.
“I cannot seriously believe in [quantum theory] because … physics should represent a reality in time and space, free from spooky action at a distance.” – Albert Einstein

1. THEORETICAL NOTIONS

QUANTUM PHYSICS

Einstein general theory of relativity and quantum physics are the two fundamental theories of contemporary physics [1]. Together, they provide the mathematical language and conceptual framework in which we express all theories in physics. Quantum computation, which is the main focus of this paper, is just one phenomenon that can be described using these theories. First, I will try to present how quantum theory describes the world as a sum of quantum physical systems; that will lead to describe the simplest physical system: the qubit. The mathematical formulations of quantum mechanics are those mathematical formalisms that permit rigorous description of quantum mechanics. The fundamental notions of the description are the quantum state and the quantum observables. Werner Heisenberg invented the Matrix Mechanics, which was the first autonomous and logically consistent formulation of quantum mechanics. Matrix Mechanics was based on algebras of infinite matrices. Although Schrödinger himself after a year proved the equivalence of his wave-mechanics and Heisenberg's matrix mechanics, the reconciliation of the two approaches is generally attributed to Paul Dirac, who was the third, and perhaps most important, person working independently in that field [2]. Paul Dirac introduced the bra-ket notation.

PHYSICS AND THE THEORY OF COMPUTATION

David Deutsch offers an analogy between Computation and Physics. One of the elements of computation is the Computer that uses Computation on a given Input. By applying a set of Rules on the Input we’ll get the Output. Similarly, in Physics we have a Physical System that is a part of nature that undergoes a Motion from an Initial State. By applying the Laws of Motion on the Initial state we’ll bring the system to the Final State. Experiment and measurement are forms of motion. They need a system to experiment on and a Measuring instrument. We find the system in an initial state, or we prepare it in the initial state. The measuring instrument will measure the outcome, which is the final state [2].

The following diagram depicts this analogy [2]:
The physical system can be used to compute functions more efficiently than by using a classical computer.

I will state for comparison the definition of a Probabilistic Turing Machine versus a Quantum Turing Machine.

**PROBABILISTIC TURING MACHINE**

A Probabilistic Turing Machine is a Nondeterministic Turing Machine which randomly chooses between the available transitions at each point according to a probability distribution. [3]

A nondeterministic Turing machine can be formally defined as a 6-tuple [3] : a finite set of states, the alphabet, the initial state, the blank state, the accepting states and a relation (function, rule) on states. The relation on states, also called the transition relation, represents the probability to reach any state. The probabilities are between 0 and 1 and the sum of the probabilities is equal to 1.
**QUANTUM TURING MACHINE**

A Quantum Turing Machine is analogous to the Nondeterministic Turing Machine but has a different transition relation [3]. The transition relation includes the use of complex numbers which are the probability amplitudes of quantum states used for computation. The probability amplitude is a complex number whose absolute value squared represents the probability to reach a state. The sum of these absolute values squared is 1.

**DIRAC NOTATION**

In Dirac’s notation the initial state, that also represents what is known is put in a ket| >. For example, | p> expresses the fact that a particle has momentum p. It could also be more explicit: | p = 2> , the particle has momentum equal to 2; If | x = 1.23> is the particle has position 1.23. | ψ> represents a system in the state ψ and is therefore called the state vector. The ket is interpreted as the initial state in some transition or event [8].

The bra represents the final state or the language in which you wish to express the content of the ket . For example, < x = .25| is the probability amplitude that a particle in state ψ will be found at position x = .25. In conventional notation we write this as ψ (x=.25), the value of the function ψ at x = .25. The absolute square of the probability amplitude, ||<x=.25| ψ>||², is the probability density and it means that a particle in state ψ will be will be found at x = .25 with this probability. Thus, we see that a bra-ket pair can represent an event, the result of an experiment. In quantum mechanics an experiment consists of two sequential observations - one that establishes the initial state (ket) and one that establishes the final state (bra). The inner products (also named brackets) are written so as to look like a bra and ket next to each other:〈ψ₁|ψ₂〉 [8].

**QUANTUM PHYSICAL SYSTEMS**

A physical system is generally described by three basic ingredients: states; observables and dynamics (or law of time evolution) [2]. A quantum state is a set of mathematical variables that fully describes a quantum system. The state of the system is represented by a single vector known as a ket. The variable name used to denote a vector (which corresponds to a pure quantum
state) is chosen to be of the form $|\psi\rangle$ (where the "\psi" can be replaced by any other symbols, letters, numbers, or even words) [4].

A system observable is a property of the system state that can be determined by some sequence of physical operations. For example, these operations might involve submitting the system to various electromagnetic fields and eventually reading a value off of some gauge. Physically meaningful observables must also satisfy transformation laws which relate observations performed by different observers in different frames of reference. Time evolution can also refer to the change in observable values. This is particularly relevant in quantum mechanics where the Schrödinger picture and Heisenberg picture are (mostly) equivalent descriptions of time evolution. The Schrödinger picture (Wave Mechanics) and the Heisenberg picture (Matrix Mechanics) are the two formulations of quantum mechanics. In the Heisenberg picture the observables change in time and the state are time independent. In the Schrödinger picture the observables are constant but states are modified in time [2].

Describing transformations of a system reduces it to its characteristic algebra. The algebra of the system is the set of all the algebraic relations among the observables of the system. The algebraic equations of the observable at one time describe the static constitution of the system. The algebraic equations of the observables at different times describe the dynamic constitution of the system [2].

The static constitution of the system can be described by a Hermitian matrix [2]. Hermitian matrix (or self-adjoint matrix) is a square matrix with complex entries that is equal to its own conjugate transpose – that is, the element in the i-th row and j-th column is equal to the complex conjugate of the element in the j-th row and i-th column, for all indices i and j:

$$a_{i,j} = \overline{a_{j,i}}.$$ 

If the conjugate transpose of a matrix $A$ is denoted by $A^\dagger$, then the Hermitian property can be written concisely as

$$A = A^\dagger$$ [9].

**Principles of Quantum Theory (Matrix Mechanics)**

In this section I will briefly enounce the basic principles of quantum theory as the underlying theory of the following sections. Quantum computation is based on these principles, but to understand the notions of quantum computations it is not necessary to have a deep understanding of quantum physics. For classical computation we don’t need to profoundly understand the semiconductor transistors and electronic signal processing. Similarly, the principles of quantum
mechanics are presented as reference. All we need to remember from this section is that they are the quantum computation matrix operations underlying principles.

**First Principle: A Quantum System May Be Represented by a Vector**

Quantum physicists call it a *ket*, denoted by the marks $| >$ delimiting some symbol(s) informing about the state of the system, for example:

$$ |\psi\rangle \equiv |\psi(E, r, p, ...)\rangle \equiv |E, r, p, ...\rangle $$

These are strictly equivalent notations of a state vector with energy E, position r, momentum p, etc. [7].

This is the first similarity between microscopic quantum system and macroscopic objects [7].

**Second Principle: The Orientation of the Vector Representing a Quantum System Evolves in Time:**

For a system of fundamental particles, Schrödinger’s equation usually appears with a factor $\hbar$ at both sides (and $i$ at the left):

$$ i\hbar \frac{d|\psi(\omega, t)\rangle}{dt} = \hbar \omega |\psi(\omega, t)\rangle $$

The factor $\hbar \omega$ multiplying the ket then represents the energy content of the system. This is Schrödinger’s time-dependent equation. So we have a second close similarity between quantum systems and macroscopic linear objects: they both obey Schrödinger's equation [7].

**Third Principle: Kets Are Transformed Into Other Kets by Means of Operations That Reveal an Observational Property**

If $\hat{U}$ is the time evolution operator $\hat{U}(t, t+dt)$

$$ \hat{U}(t, t + dt)|\psi(t)\rangle = e^{-i\omega dt}|\psi(t)\rangle = |\psi(t + dt)\rangle, $$
Û is unitary because it leaves the length of the ket unchanged, which means that the transformation is reversible.

$$\hat{U}^{-1}$$ operator is the reversed operator of $$\hat{U}$$, equivalent to the conjugate operator $$\hat{U}^*$$ in case of unitary operators, where the phase is changed by $$\omega dt$$ [7]:

$$\hat{U}^{-1}(t, t + dt)|\psi(t + dt)\rangle = e^{i\omega dt}|\psi(t + dt)\rangle = |\psi(t)\rangle$$

**Fourth Principle: Uncertainty Principle**

The Heisenberg uncertainty principle states that the more precisely one property is measured, the less precisely the other can be controlled, or determined. The product of uncertainties in position and momentum of a particle is always greater or equal than one half of the reduced Planck constant

$$\left(\frac{\hbar}{2\pi}\right) [7].$$

$$\Delta x \Delta p \geq \frac{\hbar}{2}$$

An observable is sharp if we can determine exactly its value. In other words the probability of an observable to be in a state is 1. Not all the observables of a quantum system can be sharp simultaneously. Because measurement outcomes are undetermined, quantum mechanics only give us statistical means to make predictions [2].

**Qubit**

A qubit (quantum bit) is the simplest possible quantum system; it is a Minimal Physical system, each of whose non-trivial observables is Boolean [2].

Like a bit, a qubit can have two possible values—normally a 0 or a 1. The difference is that whereas a bit must be either 0 or 1, a qubit can be 0, 1, or a superposition of both.

An important distinguishing feature between a qubit and a classical bit is that multiple qubits can exhibit quantum entanglement. A pure qubit state is a linear superposition of the basis states ($$|0>$$ and $$|1>$$). This pair of states is called the standard basis. The choice of these states is arbitrary and corresponds to the choice of a standard measurement on the qubit system. All the other states are superpositions of the basis states [14].
This means that the qubit can be represented as a linear combination of $|0\rangle$ and $|1\rangle$:

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

Where $\alpha$ and $\beta$ are probability amplitudes and can in general both be complex numbers [6].

When we measure this qubit 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 absolute squares of the amplitudes equate to probabilities, it follows that $\alpha$ and $\beta$ must be constrained by the equation

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

because this ensures you must measure either one state or the other ($|\alpha|$ is the norm of the complex number $\alpha$ ) [11]

$|0\rangle$ corresponds to $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$|1\rangle$ corresponds to $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$\alpha_0|0\rangle + \alpha_1|1\rangle$ is $\begin{pmatrix} \alpha_0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$

QUANTUM ENTANGLEMENT

A Hilbert space is a finite multi-dimensional complex vector space [9].

Consider two non-interacting systems A and B, with respective Hilbert spaces $H_A$ and $H_B$. The Hilbert space of the composite system is the tensor product

$$H_A \otimes H_B.$$ 

If the first system is in state $|\psi\rangle_A$ and the second in state $|\phi\rangle_B$, the state of the composite system is

$$|\psi\rangle_A \otimes |\phi\rangle_B.$$ 

States of the composite system which can be represented in this form are called separable states, or (in the simplest case) product states [9].
Not all states are separable states (and thus product states). Fix a basis \(\{i\}_A\) for HA and a basis \(\{j\}_B\) for HB. The most general state in \(H_A \otimes H_B\) is of the form

\[
|\psi\rangle_{AB} = \sum_{i,j} c_{ij} |i\rangle_A \otimes |j\rangle_B.
\]

This state is separable if

\[
c_{ij} = c_i^A c_j^B,
\]

Yielding

\[
|\psi\rangle_A = \sum_i c_i^A |i\rangle_A \quad \text{and} \quad |\phi\rangle_B = \sum_j c_j^B |j\rangle_B.
\]

This means that the state of the composite system is the Kronecker product of the composing states.

It is inseparable if \(c_{ij} \neq c_i^A c_j^B\). If a state is inseparable, it is called an ‘entangled state’. [4] This means that the state of the composite system is not the product of the composing states.

Entanglement is a nonlocal property that allows a set of qubits to express higher correlation than is possible in classical systems [7].

The Bell states are maximally entangled pure states \(H_A \otimes H_B\), but which cannot be separated into pure states of each \(H_A\) and \(H_B\).

For example, this is the state of a composite system of two qubits in a Bell state:

\[
|\Phi^+\rangle = \frac{1}{\sqrt{2}} (|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B).
\]

This equation can also be written:

\[
\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle).
\]

This Bell state cannot be a cross-product of any two states:

\[
|\psi\rangle = \alpha |0\rangle + \beta |1\rangle,
\]

therefore the states are entangled.

In this state, called an equal superposition, there are equal probabilities of measuring either
Entanglement is a necessary ingredient of any quantum computation that cannot be done efficiently on a classical computer \([11]\).

**Quantum Gates**

A quantum gate is a basic quantum circuit model of computation \([2]\). As classical gates operate on bits, quantum gates operate on qubits. Like a bit, a qubit can have two possible values—normally a 0 or a 1. The difference is that whereas a bit must be either 0 or 1, a qubit can be 0, 1, or a superposition of both. Examples of quantum gates are Hadamard, Pauli-X, Pauli-Y, Pauli-Z, Phase shift gates, Swap gate, Controlled gates, Toffoli, Fredkin gates and a Universal quantum gate \([6]\).

Using the Schrödinger’s equation, it can be demonstrated that all the quantum gates are reversible. This demonstration is beyond the scope of this paper. However, in the principles of quantum mechanics section above, we can read the third principle as the final state is function of the reverse of the initial state. For the same reasons, the representing matrices are unitary matrices.

For example, the Not gate is represented by:

\[
\begin{bmatrix}
0 & 1 \\
0 & 1 \\
1 & 0 \\
1 & 0
\end{bmatrix}
\]

If we look at the classical truth table for the classical not gate on 1 bit we can observe that given the final state we know which state was the initial state: (if we apply “not” to the initial state in the row we get the final state in the column)

<table>
<thead>
<tr>
<th>Initial states/ Final states</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 (False)</td>
<td>1 (True)</td>
</tr>
</tbody>
</table>
As we can see, if we know which is the final state we can tell which was the initial state. For example, if the final state is 1 (column 3) we have the truth value only in the row 3. Thus, the initial value had to be 0.

On the other hand, if we look at the truth table of any other classical gate we cannot determine the initial state from the final state. For example, let’s look at the OR gate:

The truth table is:

<table>
<thead>
<tr>
<th>Initial /Final</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

If we get the final state 1, the initial state could have been any of the states: 01, 10, 11. Therefore, the OR gate is not reversible.

So, the ‘Not’ gate is the only classical gate that is reversible; thus can be used in a quantum circuit. (If we consider the wire, which implements the identity function a gate, then the wire is also reversible)

The not gate is also referred as as Pauli X gate. It maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$.

The Pauli Y gate $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. The Pauli Y is represented by the matrix:

$$ Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} $$
The Pauli Z gate leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. The Pauli Z matrix is:

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

The Hadamard gate corresponds to the transformation matrix

$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

The CNot (Controlled Not) gate 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. Below are the representing matrix and its graphical representation.

$$\begin{array}{cccccccc}
00 & 01 & 10 & 11 \\
10 & 00 & 01 & 10 \\
01 & 10 & 00 & 11 \\
10 & 11 & 00 & 01 \\
\end{array} \quad (P, Q) = (a, a \oplus b)$$

A Toffoli gate is CCNOT gate (Controlled CNOT) is a 3 bit gate. If the first two bits are in state $|1\rangle$ it applies a Pauli X on the third bit, else does nothing.
The Universal quantum gate is a set of gates to which any operation on a quantum computer can be reduced.

QUANTUM CIRCUIT

Quantum circuits can be constructed using quantum gates. A quantum circuit can have the following components:

- An input device through which we can feed quantum data
- A set of basic gates applied sequentially or in parallel forming an acyclic directed graph
- A measurement device that can read the resulted sequences of bits [11].

An example of a simple quantum circuit is 2 Hadamard gates connected with a quantum wire.

\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} & 1 \\
1 & -1
\end{pmatrix}
\begin{pmatrix}
1 & 1 \\
1 & -1
\end{pmatrix}
= \frac{1}{2}
\begin{pmatrix}
2 & 0 \\
0 & 2
\end{pmatrix}
= \begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

This result of the product between the matrices is the identity matrix; therefore this circuit has the equivalent outcome as a quantum wire.
“Computer Science is no more about computers than astronomy is about telescopes” E.W. Dijkstra

QUANTUM ALGORITHMS

A quantum algorithm is a step-by-step procedure, where each of the steps can be performed on a quantum computer. Although all classical algorithms can also be performed on a quantum computer, the term quantum algorithm is usually used for those algorithms which seem fundamentally quantum, or use some essential feature of quantum computation such as quantum superposition or quantum entanglement [11].

All problems which can be solved on a quantum computer can be solved on a classical computer. In particular, problems which are undecidable using classical computers remain undecidable using quantum computers. What makes quantum algorithms interesting is that they might be able to solve some problems faster than classical algorithms because of using quantum parallelism.

There are only a few quantum algorithms developed today. This application will provide means to model Deutsch’s algorithm. The Deutsch-Jozsa algorithm developed in 1992 is one of the first quantum algorithms and it has mostly a didactical purpose. In the Deutsch-Jozsa problem, we are given a black box quantum computer known as an oracle that implements the function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. The function is either constant (0 on all inputs or 1 on all inputs) or balanced (returns 1 for half of the input domain and 0 for the other half); the task then is to determine if $f$ is constant or balanced by using the oracle[11].

Simon's algorithm, conceived by Daniel Simon in 1994, is a quantum algorithm which solves a black-box problem exponentially faster than any classical algorithm, including probabilistic algorithms.

Shor's algorithm, named after mathematician Peter Shor, is a quantum algorithm (for integer factorization formulated in 1994[11].

Grover's algorithm is a quantum algorithm for searching an unsorted database with $N$ entries in $O(N^{1/2})$ time and using $O(\log N)$ storage space (see big O notation). It was invented by Lov Grover in 1996[11].

The basic framework for all quantum algorithms is the following:

- The system will start with qubits in a particular classical state
- The system is put into a superposition of classical states
The system is acting on this superposition with several unitary operations
The system measures the qubits [4].

DEUTSCH’S ALGORITHM

Deutsch’s Algorithm is the simplest quantum algorithm.

There are four functions $f : \{0, 1\} \mapsto \{0, 1\}$ that might be visualized as:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Constant</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Constant</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Balanced</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Balanced</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

We call the function $f : \{0, 1\} \mapsto \{0, 1\}$ balanced if $f(0) \neq f(1)$ and constant if $f(0) = f(1)$. There are two balanced and two constant functions [11].

The problem solved by this algorithm is: given a function $f : \{0, 1\} \mapsto \{0, 1\}$ as a black box (called an oracle), where one can evaluate the input but cannot “look inside” and “see” how the function is defined, determine if the function is balanced or constant.

Using classical computing rules we need to make 2 queries to solve this problem because we need to evaluate both $f(0)$ and $f(1)$. If we had a quantum computer we could get the answer in only one query [11].

The four matrices for the four possible functions are:

- $U_f(0) = 0$ - constant
  
  $U_f(1) = 0$
\[
\begin{bmatrix}
1 & 0 \\
1 & 0
\end{bmatrix}
\]

- \( U_f(0) = 0 \) - balanced
  \( U_f(1) = 1 \)
  \[
  \begin{bmatrix}
  1 & 0 \\
  0 & 1
  \end{bmatrix}
  \]

- \( U_f(0) = 1 \) - constant
  \( U_f(1) = 1 \)
  \[
  \begin{bmatrix}
  0 & 1 \\
  0 & 1
  \end{bmatrix}
  \]

- \( U_f(0) = 1 \) - balanced
  \( U_f(1) = 0 \)
  \[
  \begin{bmatrix}
  0 & 1 \\
  1 & 0
  \end{bmatrix}
  \]

The quantum circuit that solves Deutsch’s problem in one step is the following:

As we showed earlier the states at the phase \( |a\rangle \) are:

\( |a\rangle = |0, 1\rangle \)

\[
|0\rangle \quad \text{is} \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix}
\]

\[
|1\rangle \quad \text{is} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]
At phase $|b\rangle$ we have applied the Hadamard gate to each qubit:

$|b\rangle = H|0\rangle \otimes H|1\rangle$ where $H$ is

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

$$x = H^* \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$x = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

$$y = H^* \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$y = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

At phase $|c\rangle$ we have to apply $U_f$ which can have one of the following unitary matrices ($x$ is unchanged, $y$ becomes $y \otimes f(x)$):

- $U_f(0) = 0$ - constant
  $U_f(1) = 0$
  $$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $U_f(0) = 0$ - balanced
  $U_f(1) = 1$
  $$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- $U_f(0) = 1$ - constant
  $U_f(1) = 1$
We know that:

\[x = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)\]

\[y \otimes f(x) = y = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \otimes f(1/\sqrt{2} (|0\rangle + |1\rangle))\]

At phase |c\rangle we get the following results:

\[|x\rangle|y\rangle \leftarrow U_f|x\rangle|y\rangle\]

\[= \frac{1}{2} (|0\rangle|0\rangle + f(0)\rangle) - |0\rangle|1\rangle \otimes f(0)\rangle + |1\rangle|0\rangle \otimes f(1)\rangle - |1\rangle|1\rangle \otimes f(1)\rangle\]

\[= \frac{1}{2} (|0\rangle \cdot (|f(0)\rangle - |1\rangle \otimes f(0)\rangle) + |1\rangle \cdot (|f(1)\rangle - |1\rangle \otimes f(1)\rangle))\]

\[= \frac{1}{2} ((-1)^{f(0)} |0\rangle \cdot (|0\rangle - |1\rangle) + (-1)^{f(1)} |1\rangle \cdot (|0\rangle - |1\rangle))\]

\[= \frac{1}{2} ((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle) \cdot (|0\rangle - |1\rangle)\]

\[= |c\rangle\]

\[c\text{is constant}\frac{1}{2} (\pm (|0\rangle + |1\rangle)) \cdot (|0\rangle - |1\rangle)\]

\[\text{otherwise } \frac{1}{2} (\pm (|0\rangle - |1\rangle)) \cdot (|0\rangle - |1\rangle)\]

The oracle has two qubit inputs, |x\rangle and |y\rangle, and two qubit outputs, |x\rangle and |y+f(x)\rangle, where f(x) is one of the four secret mappings we previously discussed. Given the input from the output of the two Hadamards, we can write the oracle output as the sum of these four factors:
Adding the above four terms together and simplifying yields

\[ |0\rangle \cdot |f(0)\rangle - |\overline{f}(0)\rangle + |1\rangle \cdot |f(1)\rangle - |\overline{f}(1)\rangle \] [9]

Finally, at phase \(d\rangle\) applying the Hadamard transformation to the top qubit we get:

\[ |x\rangle \leftarrow H|x\rangle \]

That is \(|x\rangle|0\rangle\) is \(f\) is constant, otherwise \(f\) is balanced.

The following images describe the transformations for all 4 functions as matrix operations.
Deutsch-Jozsa Algorithm

The Deutsch-Jozsa Algorithm is a generalization of Deutsch’s algorithm. The algorithm begins with the n+1 bit state $|0\rangle^\otimes n |1\rangle$. That is, the first n bits are each in the state $|0\rangle$ and the final bit is $|1\rangle$ [8].

When measuring the top qubit we will only get $|0\rangle$ if the function is constant, otherwise the function is balanced. In conclusion we have solved the problem in one evaluation as opposed to $2^{n-1} + 1$ evaluations needed in classical computation. This is an exponential speedup [11]!
“The meaning of the world is the separation of wish and fact” – Kurt Gödel

2. IMPLEMENTATION

DESCRIPTION:

The application provides a means to graphically design a quantum circuit using quantum gates and to simulate a quantum algorithm represented by a quantum circuit. The user can choose a gate or a quantum gate from a list of elements. The gates have a number or a range of input and output pins, and some configurable properties. The user can drag and drop quantum gates to the project panel and connect them with quantum wires. The user can run the circuit and evaluate the results.

The quantum circuit simulating applications is built on top of a classical logic circuit application. LogicCircuit – is free, open source educational software for designing and simulating digital logic circuits. The open source code is downloaded from http://www.logiccircuit.org/ [9]. The LogicCircuit application provides the framework for building classical logic circuits. It provide a list of classical gates (AND, OR, NAND, EVEN, ODD, XOR, etc.) and other circuit elements such as: ROM memory, RAM memory, probes, oscillators, LEDs, etc.

We need to add quantum gates to the existing list of circuit elements and to implement to their characteristic functionality. The quantum gates that will be added are Hadamard, Pauly-X, Pauly-Y, Pauly-Z, Phase shift gates, Swap gate, Controlled gates, Toffoli, Fredkin gates and a Universal quantum gate. Each quantum gate will have a name, a graphical representation and a unitary matrix. The gates will be connected using quantum wires. A quantum logic gate can operate on a qubit: mathematically speaking, the qubit undergoes a unitary transformation. The circuit resulted will have its own unitary matrix that will be a result of complex matrix operation between the component elements’ unitary matrices.

The application will allow combining reversible classical gates with quantum gates. The classical Not gate is the classical gate that is reversible. The application will provide visual means to show that a classical non-reversible gate cannot be connected to a quantum gate that is by definition reversible.

The application will be developed in C# using the .Net development framework and builds on top of an existing open source application.

Every quantum gate is characterized by its unitary matrix. A unitary matrix is an $n \times n$ complex matrix (a complex matrix is a matrix that has complex numbers as elements) $U$ satisfying the condition

$$U^\dagger U = UU^\dagger = I_n$$
where $I_n$ is the identity matrix in $n$ dimensions and $U^\dagger$ is the conjugate transpose (also called the Hermitian adjoint) of $U$.

The identity matrix or unit matrix of size $n$ is the $n\times n$ square matrix with ones on the main diagonal and zeros elsewhere.

$$I_1 = \begin{bmatrix} 1 \end{bmatrix}, \quad I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ldots, \quad I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

The conjugate transpose, Hermitian transpose, Hermitian conjugate, or adjoint matrix of an $m$ by $n$ matrix $A$ with complex entries is the $n$ by $m$ matrix $A^*$ obtained from $A$ by taking the transpose and then taking the complex conjugate of each entry (i.e., negating their imaginary parts but not their real parts). The conjugate transpose is formally defined by

$$(A^*)_{ij} = \overline{A_{ji}}$$

capital $A$ here suggests it is a matrix. The upper case $A$ here means for each element of matrix $A$, where the subscripts denote the $i,j$-th entry, for $1 \leq i \leq n$ and $1 \leq j \leq m$, and the over bar denotes a scalar complex conjugate. (The complex conjugate of a + bi, where $a$ and $b$ are reals, is a − bi) [11].

The logic circuit can be simulated using complex matrix operations that are operations between matrices with complex numbers as elements. The matrix operations are implemented using a free C# Complex Matrix Math Library. The open source code is downloaded from http://fekete.servebeer.com/blog/?p=55 [10]. This complex matrix class has all the needed matrix operations implemented already, except for the tensor product operation that has been added. Also there are some predefined matrices already in the class, but we need predefined unitary matrices for the quantum gates that are used. The new features were developed following the software engineering life cycle: requirement analysis, design, development, and some unit testing.

**Requirements:**

- Add quantum gates to the list of circuit elements
  - Pauli $X$,
  - Pauli $Y$,
  - Pauli $Z$,
  - Hadamard,
  - $U_f$ – universal quantum gate used in Deutsch’s algorithm, etc
- Add a IsReversible property to all the gates (all the quantum gates are reversible, only the Not classical gate is reversible)
- Reversible gates can be connected only to other reversible gates
- Create a debug view that will write the matrix operations that were executed
- Add Unitary matrices to the quantum gates
- Add functions that will operate on the unitary matrices
- Instead of entering bits we need input qubits
- At the output the probe shall read qubits

**DESIGN:**

The key features of the Logic Circuit application were already implemented. The new features were implemented to be able to create quantum circuits as well and also be able to use the existing classical circuits.

The application created dynamically a database that contains all the elements necessary to build circuits. The elements are created based on a general description that includes the graphical element description and the functionality. The quantum gates are built using the same Gate Store class such as the classical gates. The quantum gates functionality is based entirely on complex matrix operations. The gates connected in parallel will use a tensor product operation between the Hermitian matrixes that describes each gate. If the gates are connected using a serial connection the matrix operation used is the complex matrix product. The input values are transformed in corresponding matrix values when needed. Also the probe will show a value based on the matrix value on the last state before the measurement.

**COMPONENT DIAGRAM**

The component diagram is a structure diagram that shows the relationship between software components, their dependencies, communication, location and other conditions [12].

The main initial components were: the Mainframe, Logic Circuit, Item Store and Function.

I added the Matrix component that provides some predefined matrices and matrix operations on complex numbers. I needed to add the tensor product operation and predefined matrices such as Pauli X, Pauli Y, Pauli Z, Hadamard, Zero qubit, One qubit, etc.

In the ItemStore component there is a sub-component GateStore that contains all the classical gates.

These gates need to have some additional properties such as IsReversibile and Matrix.
The quantum gates added to the gate store are: Pauli X, Pauli Y, Pauli Z, Hadamard, Universal Gate, etc.

Instead of applying classical logical operation on the gates, now we shall be able to apply quantum operations. For serial components the operation will be a matrix multiplication and for parallel components a tensor product.

**CLASS DIAGRAMS**

Class diagram is a static structure diagram which describes the structure of a system at the level of classifiers (classes, interfaces, etc.)[12]. It shows some classifiers of the system, subsystem or
component, different relationships between classifiers, their attributes and operations, and constraints [13].

The matrix class that provides complex matrix operations implements the tensor product and a few predefined matrices.

The gates have properties and methods that the new quantum gates have to implement as well.

All the gates, classical or quantum, implement the Gate abstract class:
The Gate type is just one of the properties of a gate. The quantum gate types have been added to the gate types.
The gate abstract class is a part of a more complex class hierarchy, derived from the abstract class circuit, derived from the abstract class item.
The specific functions of the circuit item implement a function that implement a common abstract class.

**ACTIVITY DIAGRAMS**

Activity diagram shows sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models [12].

**Tensor (Kronecker) product of two matrices:**

With matrices this operation is usually called the *Kronecker product*, a term used to make clear that the result has a particular block structure imposed upon it, in which each element of the first matrix is replaced by the second matrix, scaled by that element [11].

For matrices \( U \) and \( V \) this is:

\[
U \otimes V = \begin{bmatrix}
  u_{1,1}V & u_{1,2}V & \cdots \\
  u_{2,1}V & u_{2,2}V \\
  \vdots & \vdots & \ddots \\
\end{bmatrix} =
\begin{bmatrix}
  u_{1,1}v_{1,1} & u_{1,1}v_{1,2} & \cdots & u_{1,2}v_{1,1} & u_{1,2}v_{1,2} & \cdots \\
  u_{1,1}v_{2,1} & u_{1,1}v_{2,2} & \cdots & u_{1,2}v_{2,1} & u_{1,2}v_{2,2} & \cdots \\
  \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
  u_{2,1}v_{1,1} & u_{2,1}v_{1,2} & \cdots & u_{2,2}v_{1,1} & u_{2,2}v_{1,2} & \cdots \\
  u_{2,1}v_{2,1} & u_{2,1}v_{2,2} & \cdots & u_{2,2}v_{2,1} & u_{2,2}v_{2,2} & \cdots \\
  \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
\end{bmatrix}.
\]
For example, the tensor product of two two-dimensional square matrices:

\[
\begin{bmatrix}
a_{1,1} & a_{1,2} \\
a_{2,1} & a_{2,2}
\end{bmatrix}
\otimes
\begin{bmatrix}
b_{1,1} & b_{1,2} \\
b_{2,1} & b_{2,2}
\end{bmatrix} =
\begin{bmatrix}
a_{1,1} & b_{1,1} & a_{1,2} & b_{1,2} \\
a_{2,1} & b_{2,1} & a_{2,2} & b_{2,2}
\end{bmatrix}
\begin{bmatrix}
a_{1,1} & a_{1,2} \\
a_{2,1} & a_{2,2}
\end{bmatrix}
= \begin{bmatrix}
a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\
a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\
a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\
a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2}
\end{bmatrix}.
\]

The following activity diagram describes the tensor product of two complex matrices algorithm.

**USER INTERFACE**

The User Interface main element is the framework. The circuit element list provides all the elements that can be added to a circuit: gates, memory, probes, or wires. The Circuit will be designed in a circuit project. The project if build correctly can be run. If it is stopped it can be edited further.
UNIT TESTING

Two examples of unit tests are provided to show the usage of the classes and the functionality that the Deutsch’s algorithm’s circuit will function.

/// A test for TensorProduct

[TestMethod()]
public void TensorProductTest()
{
    // square matrices
    Matrix A = new Matrix("0, 1; 1, 2"); // 2x2 matrix
    Matrix B = new Matrix("5, -5; 10, -10"); // 2x2 matrix
    Matrix expected = new Matrix("0,0,5,-5; 0,0,10,-10; 5,-5,10, -10; 10,-10,20,-20"); // 4x4
tensor product of A and B
    Matrix actual;
    actual = Matrix.TensorProduct(A, B);
Assert.IsTrue(expected.Equals(actual));

//non-square matrices
A = new Matrix("0, 1"); // 2x1 matrix
B = new Matrix("5, -5; 10, -10"); // 2x2 matrix
expected = new Matrix("0,0,5,-5; 0,0,10,-10"); // 4x4 tensor product of A and B

actual = Matrix.TensorProduct(A, B);
Assert.IsTrue(expected.Equals(actual));

///A test for Deutsch algorithm's circuit
///(H|0> x H|1>)*Uf*

[TestMethod()]
public void DeutschTest()
{
    double OverSqrt2 = 1/Math.Sqrt(2);
    double[,] HVals = {{OverSqrt2,OverSqrt2}, {OverSqrt2, -OverSqrt2}};
    Matrix H = new Matrix(HVals); // Hadamard matrix

    Matrix One0 = new Matrix("1;0"); // |0> matrix
    Matrix One1 = new Matrix("0;1"); // |1> matrix
    Matrix ZeroOne = new Matrix("0;1;0;0"); // |01> matrix
    Matrix I2 = new Matrix("0,1;1,0"); //case 1 balanced 0>1 1>0
    Matrix Uf1 = new Matrix("0,1,0,0; 1,0,0,0; 0,0,1,0; 0,0,0,1"); //case 2 balance 0>0 1>1
    Matrix Uf2 = new Matrix("1,0,0,0; 0,1,0,0; 0,0,0,1; 0,0,1,0"); //case 3 constant 0>0 1>0
    Matrix Uf3 = new Matrix("1,0,0,0; 0,1,0,0; 0,0,1,0; 0,0,0,1"); //case 4 constant 0>1 1>1
    Matrix Uf4 = new Matrix("0,1,0,0; 1,0,0,0; 0,0,0,1; 0,0,1,0");

    //Matrix expected = new Matrix("0,0,5,-5; 0,0,10,-10; 5,-5,10, -10; 10,-10,20,-20"); // 4x4 tensor product of A and B
    Matrix actual;
    Matrix tmp = H * One0;
    Matrix tmp2 = H * One1;
    Matrix tmp4 = Uf1 * Matrix.TensorProduct(tmp, tmp2);
    actual = Matrix.TensorProduct(H, I2) * Uf1 * Matrix.TensorProduct(H * One0, H * One1);
}
3. RESULTS

To demonstrate that the application works as designed we will build 2 quantum logical circuits by dragging and dropping the elements in the project panel. The elements are connected using
quantum wires. The elements needed to build the circuits are: constant elements, Hadamard gates, The Oracle and a probe to measure the final result. The first circuit is an implementation of Deutsch’s algorithm using an oracle that implements a balanced function, the second a circuit using an oracle with a constant function.

After ‘Running’ each of the two circuits we can see that we get different results on the probe.

The Constant Oracle shows the result 0 on the probe:

The Balanced Oracle shows the result 1 on the probe:
4. Conclusions

In conclusions this paper describes the basic elements of quantum computation and briefly explains the underlying quantum physical notions. The quantum computation is based on quantum entanglement and it is realized using quantum circuits. The quantum circuits implement quantum algorithms.

The quantum mechanical notions are pretty difficult to grasp without proper quantum physics knowledge. However, they are only listed as a reference. A computer scientist can understand easily the basic elements of quantum computing such as quantum gates and quantum circuits and the fundamental mathematical operations used to describe the transformation between states.

Understanding Quantum Algorithms is not trivial, and creating new ones it is a rather challenging task. That is why there are not many quantum algorithms developed today.

The implementation of the application is based on using open source code which is very complex and not documented. The first step was to understand this code. Open source code can be used in commercial applications only after a thorough validation process.

Implementing the new features consisted in adding new types of elements to the application and implementing new ways of interacting between components.

Finally, the results show that simulating the actual quantum circuit is not a very complicated task comparing to assimilating the underlying theory. The simulation is useful to visualize the circuit for purely academic purposes. However, it is not relevant in depicting the performance of an actual quantum system.
WORKS CITED


