A QUANTUM CIRCUIT SIMULATOR

BASED ON DECISION DIAGRAMS

Approved by:

_____
Dr. Mitchell A. Thornton
Committee Chair

_____
Dr. Hesham El-Rewini

_____
Dr. V. S. Sukumaran Nair

A QUANTUM CIRCUIT SIMULATOR

BASED ON DECISION DIAGRAMS

A Thesis Presented to the Graduate Faculty of

The School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Master of Science

with a

Major in Computer Engineering

by

David Goodman

B.S., Southern Methodist University

May 19, 2007

Goodman, David B.S., Southern Methodist University, 2006

A Quantum Circuit Simulator
Based on Decision Diagrams

Advisor: Dr. Mitchell A. Thornton

Master of Science conferred May, 19, 2007

Thesis completed April 18, 2007

The advantages of quantum computing as compared to classical computing are currently being researched; therefore, the ability to correctly simulate quantum circuits is becoming increasingly more important. A Quantum Multiple-valued Decision Diagram, or QMDD, is a structure which allows for an efficient implementation of a quantum circuit simulator. This structure can be used to create a quantum circuit simulator that is both computationally efficient as well as economic with memory usage.

In this thesis, a background of quantum computing and analyzing quantum gates and circuits is given. The concept of a QMDD is presented, and the relationship between a QMDD and a quantum circuit simulator is discussed. In addition to these topics, the basic theory behind creating a quantum circuit simulator is discussed.

This thesis proposes two different approaches for creating a quantum circuit simulator and analyzes the results generated from both approaches. The first approach utilizes explicit matrix-vector multiplication using QMDD representations of the quantum circuit as well as the input vector. The second approach utilizes implicit matrix-vector multiplication where a QMDD structure representing the quantum circuit undergoes a guided traversal based on the input vector. Both of these approaches are

compared to explicit multiplication using matrices and vectors instead of the QMDD structure. Finally, results are discussed, and the effectiveness of the two circuit simulation methods is analyzed and compared with an existing quantum circuit simulator.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

There are many people I have to thank for supporting me and influencing my work while at Southern Methodist University. First of all, I would like to thank Dr. Mitchell A. Thornton, my thesis advisor, for his guidance throughout my years as his research assistant. I owe many thanks to him for the opportunities he has given me. I appreciate the trust he has put in me even at the times I did not necessarily deserve it. I greatly value the knowledge I have acquired while working with Dr. Thornton, and I will always remember the lessons I have learned from him.

I would like to express my thanks to the other members of my thesis committee, Dr. Hesham El-Rewini, and Dr. V. S. Sukumaran Nair. I sincerely appreciate not only the time they have dedicated to review my thesis, but also the lessons I have learned from both throughout my undergraduate years at SMU. I greatly respect the work both professors do for the university, and I hope that many future students will be touched by their teaching as I have.

I have greatly enjoyed working with the students in the CAD methods research group. I have learned much from my colleagues: Lun Li, Mahsan Amoui, Poramate Ongsakorn, Jason Moore, Laura Spenner, David Feinstein, and Kelly Hawkins. I have often relied on them for assistance with my research, and I appreciate all of the help they have given me. Additionally, I would like to thank some of my other classmates,

particularly Chris Christensen, James Kang, Paul Hartin, and Samantha Oleksy, for their assistance with UPE and Tau Beta Pi. Without their help, I would have never been able to accomplish what I have this year. Also, I would like to give thanks to the School of Engineering staff, particularly Debra McDowell, Beth Minton, and Jim Dees, for all of their help in achieving my goals.

I owe much gratitude to my family for their support not only this year, but throughout the course of my education. My family has always been there to support me throughout my entire life, and I am grateful for their love and guidance. My family's support is important to me, and I hope this work and my work in the future will continue to make them proud.

I wish to thank my friends as well for their understanding. I know that my work has consumed me for the past year, and I look forward to some well-needed catching up. I look forward to some good times with some of the best people I have ever known.

Finally, I must express my greatest thanks to my girlfriend, Rachel. This entire year, we have shared in the ups and downs of my research. She has been there for me during the joyous times as well as the painful times. Without her love and support, I would have never been able to endure the late nights, lack of sleep and immense stress that has been a part of my life this year. She is someone that I share my dreams and worries with, and I know that I can always count on her to be there for me. I look forward to our future together with great anticipation. I know that the completion of this thesis means as much to her as it does to me, and I cannot thank her enough for the love that she gives me every single day. A big part of this thesis belongs to her, and I can only repay her by giving her all of my love. She is my everything. Thank you!

Chapter 1

INTRODUCTION

Recently, there has been much interest in the concept of quantum computing since many problems that are intractable with classical computers can be solved by quantum computers in polynomial time. For example, both Shor's factorization algorithm [1] and Grover's search algorithm [2] are intractable problems in the realm of classical computing; however, these problems are able to be solved in polynomial time with the use of quantum computations. In addition to the ability to solve complex problems, quantum logic operations are reversible [19]. For this reason, there is additional interest in these types of circuits because only reversible circuits can approach the theoretical limit of no heat dissipation [13] [14]; therefore, there is significant interest in developing practical quantum logic circuits because of the prospect of very low power dissipation.

The novel concept behind quantum computing is directly related to the smallest unit of data available. In classical computing, the smallest unit of data is the bit. This unit of data either holds a "one" or a "zero". These small units of data can be stored in memory, transported through logic gates, and stored in memory again. Conversely, in quantum computation, the smallest unit of data is the qubit. Just like a bit, a qubit can hold a value of "one" or "zero"; however, the thing that makes a qubit different is that it

can also hold a superposition of one and zero. In a quantum computer, these qubits (which are stored in registers) undergo evolutions in time or space. Each of these evolutions represents a quantum gate operation.

As a part of the development of quantum computing, it is necessary to find efficient ways to design the quantum circuits. A quantum logic circuit represents transformations of the state of one or more qubits over time or space. These circuits are modeled as a cascade of one or more quantum logic gates. A quantum logic gate can be represented by a unitary transformation matrix (as discussed in Chapter 2). Due to this property of quantum circuits, the transformation of a number of qubits from their initial state into a final state can be computed through the use of matrix-vector multiplication [13]. The downfall of this method is the fact that for an n-qubit circuit, the transformation matrix is of size $2^n \times 2^n$ and the vector representing the initial state of the set of qubits is of size 2^n. Due to these extreme sizes, the simulation of quantum logic circuits using a matrix-vector multiplication method is only useful for circuits with a very small number of qubits.

This thesis presents two different methods for simulating quantum circuits. Both methods utilize a structure called a Quantum Multiple-valued Decision Diagram, or QMDD. These methods eliminate the problem of exponential growth of the size of representational matrices so that very large quantum circuits can be simulated in a relatively short amount of time.

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of quantum logic, including quantum gates and matrix representations of quantum gates. Chapter 3 discusses the QMDD structure in detail. Chapter 4 details the

quantum circuit simulator that has been developed as a direct result of this research. Finally, Chapter 5 presents the computational results of the quantum circuit simulator discussed in Chapter 4 and compares these results with those of another quantum logic circuit simulator, QuIDDPro [11] [12].

Chapter 2

QUANTUM LOGIC

Quantum logic, which is analogous to digital logic in classical computing, defines the way in which quantum bits, or qubits, are transformed according to certain functions. As stated in the introduction, a special property of quantum gates and circuits is their reversibility. The reversibility of these logic gates allows for their representation by unitary matrices [21]. The work in [3] provides a very detailed background of quantum logic. Here, the basics of quantum gates and circuits are discussed.

2.1. Quantum Gates

In classical computing, a logic gate is described as a piece of hardware that performs a logical operation on one or more logical inputs to produce a single logical output. Quantum gates are analogous to digital logic gates, but there are a number of differences. Classical computing circuits consist of wires and logic gates. The wires carry information around the circuit, and the logic gates perform operations on that information. Additionally, memory may be included as part of a circuit to store the values of various bits. In order to understand the functionality of a quantum gate, some examples are introduced.

First, some background on quantum physics is provided to clarify the material to follow. The notation used to describe a quantum state is called "braket" notation. This name is used since this notation represents the inner product of two states denoted by a bracket, $\langle\varphi|\psi\rangle$. The left part, $\langle\varphi|$, is termed the "bra", while the right part, $|\psi\rangle$, is called the "ket". In quantum physics, the state of a physical system is identified by a point in the Hilbert space of the system (an abstract vector space in which distances and angles can be measured) [15]. The ket is used to represent a quantum state. In the case of quantum logic, there are two distinct quantum states, one and zero. These are represented by $|1\rangle$ and $|0\rangle$. For the remainder of this thesis, the quantum state $|\psi\rangle$ will be used to represent

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

which is a superposition of the two distinct quantum states. 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

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

The use of the "braket" notation is common because of the work of Dirac in quantum mechanics.

For example, consider the only single input, single output, classical logic gate, the NOT gate. The NOT gate simply takes a logic input and provides the negation of this input as the output of the circuit. In other words, the gate interchanges the 0 and 1 states

of classical bits. A good place to start is to show how a quantum NOT gate is constructed. Recall that quantum bits can represent the zero state (|0>), the one state (|1>), 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.

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

In order to perform a quantum NOT, a gate is needed that will reverse the probabilities of each quantum state. Since quantum gates can be represented as unitary matrices, the quantum NOT gate can be represented by the following matrix:

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

In order to verify the functional operation of this gate, the following example is presented. 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 been realized. It is also important to notice that this quantum gate (along with all other quantum gates) is completely reversible. If the output of the previous equation were to be substituted for the input, the original quantum vector would be the result.

As an example, another commonly used quantum gate is the controlled-NOT gate, or CNOT gate. This gate operates on two qubits (as opposed to one with the NOT gate). One qubit is designated the "control" while the other is designated the "target". The operation of the CNOT gate is defined as follows: when the control qubit has a value of |1> the target qubit is reversed, otherwise, the target qubit remains unchanged. In both cases, the control qubit is unchanged by the CNOT gate.

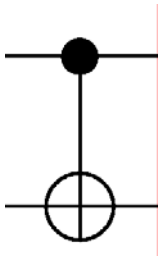


Figure 2.1. Quantum CNOT Gate

Just like logic gates, quantum gates can be represented in a diagram form. Figure 2.1 shows a graphical representation of the quantum CNOT gate. The smaller, closed circle indicates the control qubit while the larger, open circle indicates the target qubit.

Figure 2.2 shows the graphical representation of another commonly used quantum gate, the controlled-controlled-NOT or Toffoli gate.

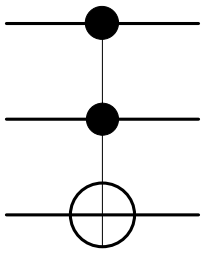


Figure 2.2. 3-qubit Toffoli Gate

The Toffoli gate is a quantum logic gate that involves three qubits. The function of the Toffoli gate is based on the two qubits designated "controls". Analogous to the functionality of the CNOT gate, the Toffoli gate will invert the "target" qubit if, and only if, both of the "control" qubits are set to |1>. Another version of the Toffoli gate is the generalized Toffoli gate. This gate utilized three or more qubits as "control" qubits. In the case of a generalized Toffoli involving *n* qubits, *n-1* qubits are controls and the remaining qubit is the target. The gate will reverse the target qubit if, and only if, all of the control qubits have a state of |1>.

The other quantum gates that are used in this thesis are the Controlled-V and Controlled-$V^{\dagger}$ gates. (Although a small set of gates are used for the examples presented in this thesis, the QMDD structure is easily extended to use other common quantum gates.) The Controlled-V gate is often referred to as the square root of NOT gate, since a cascade of two such gates produces the same transformation as a quantum NOT gate. The Controlled-V gate operates in a similar manner to the CNOT gate in that the transformation is applied to the target qubit if, and only if, the control qubit is set. In the

event that the control qubit is set, the Controlled-V gate applies the transformation, given by the following matrix, to the target qubit:

$$V = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

Similarly, the Controlled-$V^{\dagger}$ applies the following transformation if the control qubit is set to |1>:

$$V^{\dagger} = \frac{1-i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

This transformation is actually equivalent to the inverse of the V transformation matrix. The symbols for these gates are shown in Figures 2.3 and 2.4.

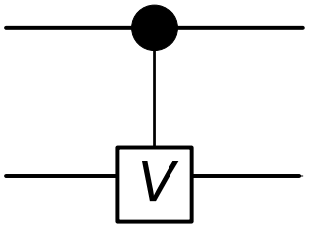


Figure 2.3. Controlled-V Gate

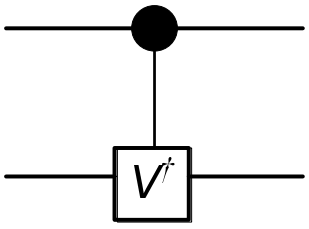


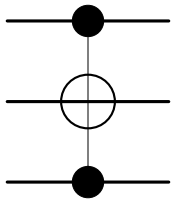
Figure 2.4. Controlled-$V^{\dagger}$ Gate

2.2. Matrix Representations

A special property of quantum gates is their ability to be represented by a transformation matrix. A quantum gate that operates on n qubits can be represented by a $2^n \times 2^n$ unitary matrix. As an example, Figure 2.5(a) shows the matrix representation of the 3-qubit Toffoli gate shown in Figure 2.5(b). 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 significant qubit. All gates mentioned in this thesis are denoted in a similar fashion.

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

(a)



(b)

Figure 2.5. Matrix Representation of a Toffoli Gate

The ability to represent a quantum gate with a unitary matrix allows for many complex operations to be performed through linear algebraic methods. For example, by using matrix-vector multiplication, it is possible to produce the output vector from a specific input vector representing the quantum states of the qubits provided as inputs to a quantum gate.

2.3. Quantum Circuits

A quantum circuit is simply a circuit comprised of one or more quantum gates. Multiple quantum gates may be cascaded together to form a quantum circuit. Quantum circuits such as the one presented in Figure 2.6 represent a transformation through time or space. The transformation is generally read from left to right; however, since all quantum gates are reversible, the transformation may also be read in the opposite direction.

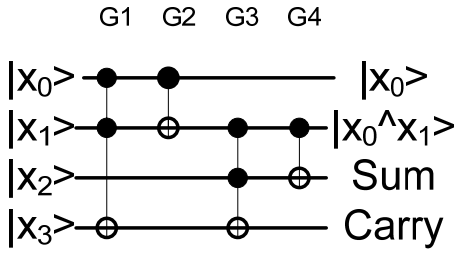


Figure 2.6. Quantum Full-Adder Circuit (rd32)

As an example, the circuit in Figure 2.6 is a quantum full-adder circuit [9]. In this particular circuit, the third qubit output represents the sum and the fourth qubit output represents the carry.

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. It is shown in Chapter 4 that representing an entire quantum circuit with a single matrix allows the circuit simulation to be performed using matrix

multiplication. Additionally, Chapter 4 demonstrates how the QMDD structure can be used to perform the same simulations without the use of matrix multiplication.

This example illustrates how a unitary matrix representing an entire quantum circuit is built. The circuit shown in Figure 2.6 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> and |1> states, the matrix can be thought of as a permutation matrix. The matrix below is the representational matrix for this gate.

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

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 $2^4 \times 2^4$. 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.

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

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

This process can be continued until all four gates have individual representational matrices. Finally, in order to generate the representational matrix for the entire circuit, the matrices are multiplied together using traditional matrix multiplication. The matrices for the gates in the circuit in Figure 2.6 can be seen below.

$$G1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

$$G2 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$G3 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

$$G4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Now the matrices can be multiplied together to obtain the following product.

$$G1 \times G2 \times G3 \times G4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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

THE QMDD STRUCTURE

The Quantum Multiple-valued Decision Diagram structure is first introduced in [4] as a means of representing a quantum or reversible circuit with a structure based on decision diagrams. The idea of storing and manipulating matrices with decision diagram structures has been previously introduced in [22].

3.1. Motivation

The main motivation for the innovation of the QMDD structure is the regular structure of the matrices that represent quantum gates and circuits. For a binary circuit, the dimensions of the representative matrices are always powers of 2; therefore, the matrices can always be partitioned into four quadrants. For example, the $2^n \times 2^n$ matrix $\mathbf{M}$ can be partitioned as

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{bmatrix}$$

where each of the partitions has dimensions $2^{n-1} \times 2^{n-1}$. Furthermore, each of the remaining partitions can be subdivided into four smaller matrices. This process can recursively continue until only single values remain. Eventually, these single values represent the weights of the four outgoing edges from a node of the decision diagram. A quality that arises in many quantum and reversible circuits is repetition within the

representational matrix. For example, in the matrix shown below, the partition of the upper-left quadrant can be further partitioned into four matrices; however, the two pairs of the newly partitioned matrices are identical.

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

The characteristic repetition within the representational matrices makes representing a quantum circuit as a decision diagram compact. Due to the repetitiveness, many of the repeated submatrices do not need to be stored more than once. This is a great help in reducing the amount of memory necessary to simulate a quantum or reversible circuit.

3.2. Design of the QMDD Structure

Many of the specific details of the Quantum Multiple-valued Decision Diagram structure are discussed in [4] and [7], but some of the important relevant aspects are summarized here. A binary decision diagram is a data structure that can be utilized to represent a Boolean function. A binary decision diagram is defined as a rooted, directed, acyclic graph consisting of decision nodes and two terminal nodes called 0-terminal and 1-terminal. In a binary decision diagram, each decision node is labeled by a Boolean variable and has two child nodes. The edge from a decision node to a child node represents an assignment of the Boolean variable to either 0 or 1. A binary decision

diagram is also referred to as "ordered" if different variables always appear in the same order on all paths from the root. Further details on binary decision diagrams can be found in [6] and [23].

As an example, Figure 3.1 shows a binary decision tree structure.

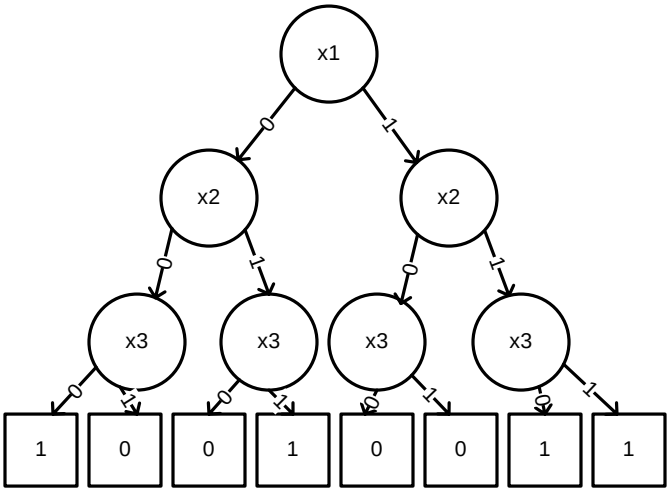


Figure 3.1. Binary Decision Tree Structure

This decision tree represents the function

$$f=\Sigma(0,3,6,7)$$

The binary decision diagram structure for this same function is shown in Figure 3.2. All of the redundant nodes have been removed.

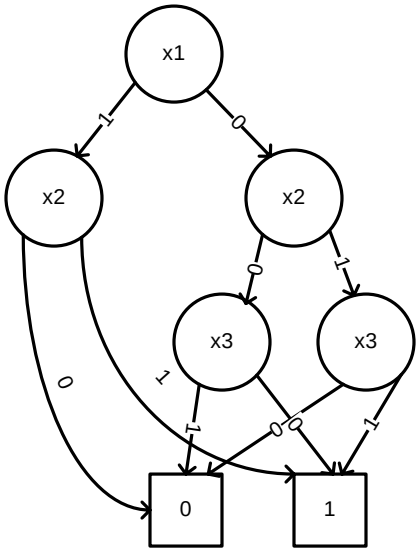


Figure 3.2. Binary Decision Diagram

Similar to a binary decision diagram, a Quantum Multiple-valued Decision Diagram is a compact structure used to represent quantum circuits. The details of such a diagram are discussed in the following sections.

3.2.1. Characteristics of a QMDD

Although the binary decision diagram is completely defined, it is necessary to create another similar structure to use with quantum and reversible circuits; this is the inspiration for the QMDD structure. There are significant differences between the QMDD structure and the binary decision diagram. Some of the differences are discussed below:

- There is only one terminal vertex. This vertex has an associated value of one and has no outgoing edges.

- Each non-terminal node has four outgoing edges (as opposed to two for a traditional binary decision diagram).
- Each edge in the QMDD has a complex-valued weight associated with it. Every edge with a weight of 0 points to the terminal node.
- No non-terminal vertex is redundant. This means that any vertices that have four outgoing edges all with the same weights and all pointing to the same vertices are only stored once to avoid repetition.
- All non-terminal vertices are normalized. This means that when looking at the outgoing edges from left to right, the first edge with a non-zero weight must have a weight of one. All edges to the left of this edge must have weights of zero.
- Finally, all non-terminal vertices are unique. This ties in with the previously stated fact that no non-terminal vertex is redundant.

As an example, Figure 3.3 shows a QMDD structure representing a single Toffoli gate with variables x_0 and x_2 as controls and x_1 as the target.

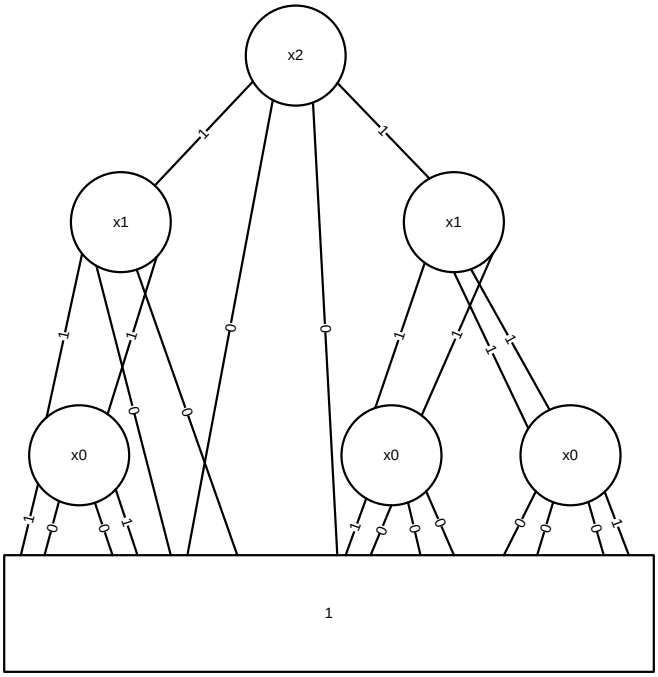


Figure 3.3. Sample QMDD Structure

This QMDD structure encompasses many of the characteristics presented in the list above. For example, there is only one terminal node, which is labeled with a 1. Also, each of the non-terminal nodes have four outgoing edges.

This QMDD structure completely represents the quantum circuit consisting of this one Toffoli gate. In order to verify that this is true, it is possible to build the representational matrix from the QMDD structure. This is done through a series of reverse Kronecker operations. Since there are three qubits, the resulting matrix will be of size 8×8. The first step in generating the matrix is to look at the initial node. Each of the four edges "points to" one of the 4×4 submatrices of the representational matrix. If any of these edges has a weight of zero, that entire submatrix is a zero matrix. For example,

after examining the initial node, the matrix can be partially completed as the matrix below shows.

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

Next, the nodes on the next level down are examined. Each of these nodes represents one of the 4×4 matrices that is yet to be filled in. A similar method is used to fill in these submatrices. Each of the edges of these nodes points to a 2×2 submatrix. The matrix below shows the representation matrix after examining the nodes at the level of variable *x1*.

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

Finally, when the lowest level of nodes is reached, the weights of the outgoing edges of each of these nodes fill in the remaining 2×2 submatrices. The final representational matrix is seen below.

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

3.2.2. Building a QMDD Structure

This section describes how a QMDD structure is built from a circuit netlist. The QMDD software package is described in detail in [4] and [7], but the process of building a QMDD structure is presented here for continuity.

In order for a QMDD structure that represents a quantum or reversible structure to be built, it is necessary to first build a QMDD structure for each individual gate. The variable order is as follows: the variables are labeled $x_0, x_1, \ldots, x_{n-1}$ from the terminal node to the start node. Additionally, each quantum logic gate is specified by the base transition matrix M. When the QMDD structure for a particular gate is being built, it is actually built from the terminal vertex working its way up to the start vertex. The procedure has three phases as described in [7] which are outlined here.

First, structures are constructed for the variables that lie "below" the target variable in the variable ordering ("below" referring to variables that are closer to the terminal vertex in the variable ordering). For each of these variables, there are four separate QMDD structures that are created. Out of these four QMDD structures, the ($i \times$

$2 + j)^{th}$ QMDD structure has a path that follows the values required to activate the controls to the terminal value $M_{i,j}$ while all other paths lead to zero. As an example of this, consider the simple quantum circuit presented in Figure 3.4.

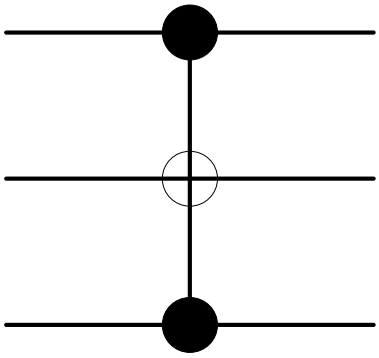


Figure 3.4. Simple Quantum Circuit

This quantum circuit consists of just one quantum gate, a Toffoli gate. Based on the description given above, the 2^{nd} QMDD structure has a path following the values required to activate the controls which then points to the terminal value in the matrix. Such a QMDD structure is given in Figure 3.5.

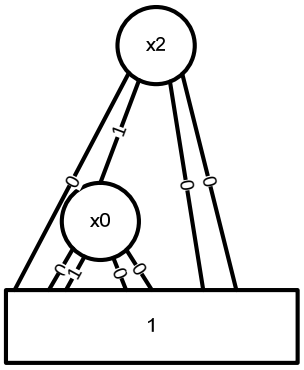


Figure 3.5. QMDD Structure for Variable Below the Target

Any variables that are not target or control variables (there are none in the above example) are taken into account through the use of identity matrices. These matrices are combined with the rest of the QMDD using Kronecker multiplication.

After these QMDD structures have been created, a single structure is created for the target variable. The source node of this QMDD structure is assigned the target variable. The edges of this source node point to the QMDD structures that were created in the previous step.

Finally, there are additional steps for the variables that lie above the target variable. Similar to those lying below the target, the upper part of the QMDD has a path that follows the values required to activate the controls, and all of the other paths lead to 0. The non-control variables are accounted for through the use of identity matrices and the Kronecker multiplication operation.

The efficiency of the QMDD building process comes from the fact that the structures are built in a single pass through the variables. The building begins from the variable associated with the terminal node and continues up through the initial node with

no backtracking. Once a QMDD structure has been created for each of the gates in the QMDD circuit, the structures are combined through the QMDD-based multiplication operation discussed in detail in [4].

3.2.3. Variable Labels

Throughout the building process, attention is given to the variable labels. This is an essential step because the variable names are important for simulation. The procedure described in the previous section gives information about how the variable labeling is performed.

Proper node labeling is necessary for correct circuit simulation. It must be ensured that when a QMDD structure is traversed from the initial node to the terminal node, each variable is encountered at most one time. Of course, some traversal paths will not include every variable. According to [16], this will only occur in the case when an edge points directly to the terminal node and has a weight of zero.

Figure 3.4 shows the complete QMDD structure for the quantum circuit in Figure 3.1.

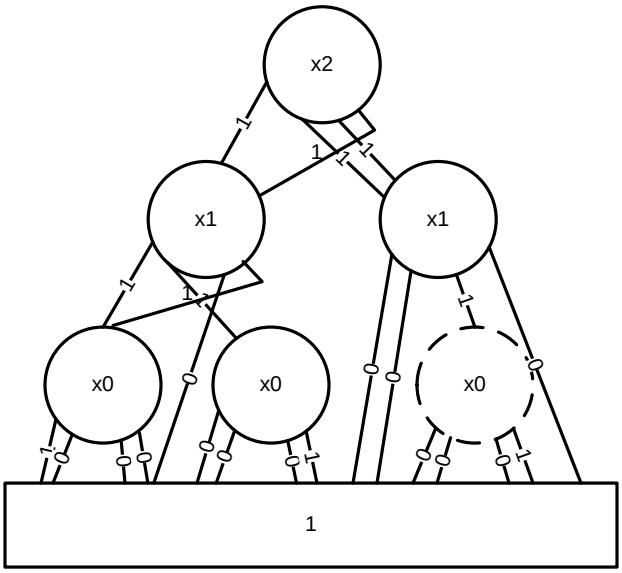


Figure 3.6. Complete QMDD Structure

In this particular quantum circuit, the QMDD structure that is built has a redundant node. This node is denoted by the circle drawn with a dashed line. This means that two different nodes have four edges with equivalent weights all pointing to the same nodes. In these cases, the redundant node is only stored in memory once, and multiple edges from a parent node point to this redundant node. An example of this can be seen in the nodes labeled with variable x_0 in Figure 3.6. Notice that the two nodes on the right side have all four edges with the exact same weights that all point to the terminal node. In this case, the node would only be stored one time. Figure 3.7 shows the QMDD structure with the redundancy removed.

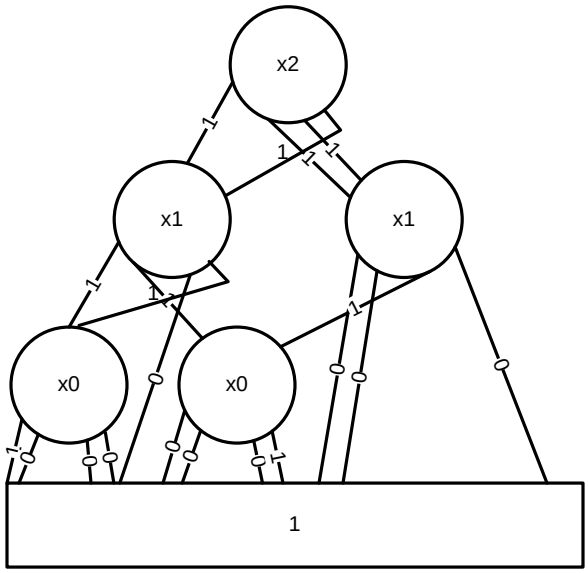


Figure 3.7. Complete QMDD Structure with Redundancy Removed

3.3. Implementation of a QMDD Package

In [7], the details of the QMDD package used for this thesis are discussed. Some important points are discussed here as they relate to results presented later in this thesis.

3.3.1 Node Storage

An important aspect of the QMDD package is the way in which the nodes are stored in memory. It is important that redundant nodes are not stored multiple times since the structure of the QMDD leads to many redundant nodes.

The choice for node storage in this particular package is the use of a unique table. A unique table is used to store references to each of the nodes. Whenever a new node is created for the QMDD structure, the unique table is first traversed to see if a redundant node has already been defined. If the node already exists in the unique table, instead of

creating a new node, the software refers to the unique table when using the node in the QMDD structure. If the node does not yet exist, it is created and then added to the unique table. By using this method, only the unique nodes are stored in memory, saving the amount of memory necessary to store redundant nodes multiple times.

3.3.2. Complex Numbers

In addition to saving memory by using a unique table, memory can also be saved by using a complex number table. Each of the outgoing edges from a QMDD node has a complex-valued weight associated with it. In order to reduce the amount of storage necessary, a complex number table is created. This table is similar to the unique table previously mentioned, except this table stores complex numbers as opposed to QMDD nodes. The operation of the table is very similar to the unique table. When a new edge is to be added, the complex number table is inspected to see if the weight of the edge has already been stored in the complex number table. If that particular number has already been stored in the table, there is no need to add it. If the number does not yet exist, it is added to the complex number table.

Additionally, to save more memory, instead of assigning the complex value to the weight of each edge, an integer is assigned to each edge weight. This integer represents an index in the complex number table. Therefore, instead of using enough memory to store a complex number for each of the edges, only a single integer is stored.

3.3.3. Operation Tables

Another time saving addition to the QMDD package is operation tables. These tables store the results of the addition, multiplication, or division of two complex numbers. Since these complex computations are relatively expensive, having an operation table can save additional computational time.

Similar to the operation of the other tables recently discussed, the operation tables are traversed any time a complex operation is performed. If the operation has not been previously performed, it is added to the operation table. Additionally, the commutative properties of complex addition and multiplication are accounted for so that no repetitive operations are performed.

3.3.4. Computed Table

Distinctly similar to an operation table, a computed table is a common feature of decision diagram packages [6]. A computed table assures that no redundant computations are performed. The computed table is used to keep track of any instances of matrix addition, matrix multiplication, or Kronecker multiplication [8] performed on two QMDD structures. Additionally, the operations themselves are recursive, so the subcomputations are stored in the table and checked before performing any operation.

3.3.5. Zero and Identity Matrices

Due to the nature of the QMDD, both zero and identity matrices occur very frequently. Methods are in place to efficiently handle both of these cases. First of all, the zero matrix case is handled where, any time a zero matrix is encountered, it is represented

as an edge with a weight of zero pointing to the terminal node. Using this method, no other precautions need to be taken for dealing with a zero matrix.

To facilitate dealing with identity matrices, a flag associated with each vertex indicates if the vertex and its descendants represent an identity matrix. If the vertex represents an identity matrix, many of the matrix operations can be simplified. For example, when performing matrix multiplication, if one of the matrices is the identity matrix, the operation becomes much simpler. The equations below demonstrate how the multiplication becomes simpler.

$$A \times I = I \times A = A$$

3.4. Conclusion

There are many other optimization details included in the implementation of the QMDD package [4] [7]. The result is a software package that represents quantum and reversible circuits efficiently. The QMDD structure and the software representing it are both integral parts of the remainder of the work discussed.

Chapter 4

QUANTUM CIRCUIT SIMULATOR

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. The background for some methods of quantum circuit simulation is provided in [11] and [12], specifically, a quantum circuit simulator called QuIDDPro. This chapter first presents the method the QuIDDPro approach uses for quantum circuit simulation, and then explores a variety of different options for simulating quantum circuits that are implemented in this work. Finally, the options are compared to QuIDDPro and to each other based on the results of a number of simulations.

4.1. Quantum Circuit Netlist Formulas

When working with quantum and reversible circuits, there are number of different ways that the circuits can be described. In previous versions of the QMDD package, a particular format has been used. This format (`tfc` files) is the same that is used by Maslov in his benchmark circuit set [9].

Another common format used to describe quantum and reversible circuits is the QASM format. The QASM format is a text-format language used to describe quantum circuits. More information on the QASM format can be found in [10].

In order to make this quantum circuit simulator a useful tool for those working in the area of quantum and reversible circuits, the simulator is compatible with both **`tfc`** formatted circuits as well as QASM formatted circuits. Since the software previously developed is compatible with the **`tfc`** format, a simple program has been written to translate a QASM specification into the **`tfc`** format. Once the translation is complete, the resulting file can be used with the existing software.

To demonstrate the differences in the file formatting, Figures 4.1 and 4.2 show examples of a file in **`tfc`** format and QASM format respectively.

```
#rd32.qasm

qubit a
qubit b
qubit c
qubit d

toffoli a,b,d
cnot a,b
toffoli b,c,d
cnot b,c
```

Figure 4.1. rd32.qasm

```
.v a,b,c,d
```

```
.i a,b,c
.o d,c
.c 0

BEGIN
t3 a,b,d
t2 a,b
t3 b,c,d
t2 b,c
END
```

Figure 4.2. rd32.tfc

These two files represent the exact same quantum circuit (rd32, or the full-adder) in both formats.

4.2. Input Vector Formation

For the purposes of these simulators, it is assumed that the input vectors are represented by an *n*-qubit register with all qubits initialized to basis or eigenstate values of |0> or |1>. In order to perform matrix-vector multiplication for simulation, the first step is that the register of eigenstate values must be transformed from Dirac notation into a vector of appropriate values. This vector is formed through the use of the Kronecker product. This operation is used to transform the Dirac-ket notation, defined in Chapter 2, into the proper column vector format [3].

One property of the Kronecker operation is that it is not commutative; it is, in fact, associative. In order to account for the lack of a commutative Kronecker product, in the simulators described in this thesis, the Kronecker operation is evaluated as a series of Kronecker products evaluated from right to left.

As an example, consider the original register values of |100>. The column vector associated with these original eigenstates is formed as shown below:

$$|100\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}$$

The column vector that is the result of these operations is the "input vector" for the quantum circuit. The input vector is used by the simulator to calculate the proper output for the circuit.

4.3. Circuit Simulation Methods

This section will discuss several different simulation methods that have been implemented. This thesis will focus on two primary methods for circuit simulation. First is the explicit multiplication based simulator. This method is compared with the other method developed in this project, the implicit multiplication based simulator. In addition to the implementation of these methods, the naïve linear algebraic approach involving matrix-vector multiplication is also implemented for comparison purposes. Finally, we describe an existing method described in [11] [12] and [20] that is also considered here for comparison purposes.

4.3.1. Linear Algebra Approach

The linear algebra approach is implemented as a part of the development of the quantum circuit simulator. The matrix-vector multiplication that occurs is actually a representation of what happens when the QMDD structures are used for quantum circuit simulation. This approach is implemented for two reasons: to check simulation outputs during development of the quantum circuit simulator and as a basis for comparison to the QMDD-based simulation methods. This approach is implemented as a matrix-vector explicit multiplication.

4.3.2. Output Conversion

With the linear algebra approach as well as the other simulation methods, a conversion of the output vector is necessary so that the results may be presented to the user in Dirac-ket notation. To handle this task, an algorithm is implemented to convert an output vector back to Dirac-ket notation. Figure 4.3 gives the pseudocode for this algorithm.

```
for(i=number of elements in output vector; i>1; i=i/2)
{
        Search the first half of the vector for a '1'
        If '1' is found
        {
                Store '0' in last available position of Dirac-ket vector
        }
        Else
        {
                Store '1' in last available position of Dirac-ket vector
                Copy second half of output vector into first half
        }
}
```

Figure 4.3. Pseudocode for Formatting Output

In general, this conversion process can be difficult since it would mathematically require formulating a Kronecker product factorization. However, in the case of this simulator it is assumed that all outputs will be given in eigenstate form. Due to this fact, the algorithm described above is sufficient to perform this translation.

A full example of this method is given here. For simplicity, a circuit consisting of a single quantum gate will be simulated. Assume that the circuit consists of a single Toffoli gate as shown in Figure 4.4.

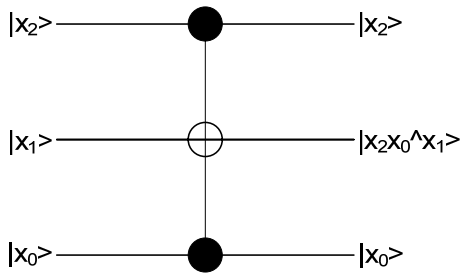


Figure 4.4. Sample Quantum Circuit for Simulation

In this case, the matrix representing the quantum circuit is given below:

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

Following the same methods as before, assume that the input vector is given by the eigenstates |101>. This can be translated into the proper input vector format through Kronecker multiplication as shown below.

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

Once this is complete, the matrix vector multiplication is performed.

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

As a final step, the algorithm to translate the output vector back to Dirac-ket notation is invoked, yielding |111>. The final step actually performs the factorization seen below.

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

The quantum circuit verifies that this is the proper output.

Utilizing matrices and vectors is a very inefficient way to perform circuit simulation. In fact, using matrices and vectors will only work for very small circuits. For any circuits larger than approximately seven qubits, the matrix and vectors become so large that there is not enough memory to handle the multiplication. This is the inspiration for using the QMDD structure to perform quantum circuit simulation. Sections 4.3.4 and 4.3.5 discuss the two methods proposed for quantum circuit simulation using the QMDD

structure, but first, the QuIDDPro simulation mechanism is introduced for comparison purposes.

4.3.3. QuIDDPro Quantum Circuit Simulator

The main structure in the QuIDDPro software package is the QuIDD, or Quantum Information Decision Diagram. QuIDDPro makes use of the decision diagram package CUDD, developed by Somenzi [17]. The decision diagrams implemented in CUDD as well as QuIDDPro are algebraic decision diagrams. More details on algebraic decision diagrams are presented in [18] and [22].

The QuIDDPro software begins simulation by creating a QuIDD for each of the gates in the quantum circuit. Next, these QuIDD structures are multiplied together to create a single QuIDD which represents the entire quantum circuit. A QuIDD structure is now created to represent the input vector. Finally, the QuIDD representing the quantum circuit is multiplied by the QuIDD representing the input vector. After the multiplication is complete, measurement functions designed specifically for QuIDDPro extract the output values.

This thesis presents alternate methods for quantum circuit simulation using the QMDD structure. The two main methods that are used in this research are described in the next two sections.

4.3.4. Explicit Multiplication Based Simulator

The first QMDD method for circuit simulation is the explicit multiplication based simulator. This simulator relies on the same basic concept as the linear algebra approach, but the use of the QMDD structure allows for an improvement in execution time.

A problem with the linear algebra approach for quantum circuit simulation is the size of the matrices. For an n-qubit circuit, the size of the representative matrix is $2^n \times 2^n$. The sizes of these matrices grow exponentially. For example, assume the circuit to be simulated has 10 qubits. This results in a representative matrix with 2^{20} elements. If the assumption is made that the matrix is storing short integers (1 byte each), this results in a matrix needing 1MB of storage. As compared to some of the benchmark circuits commonly used for quantum circuit simulation, this quantum circuit has a small number of qubits. This shows that even quantum and reversible circuits with few qubits require an excessive amount of memory to store the representational matrix.

In an effort to create a method for quantum circuit simulation that is not as memory intensive, a new method is created for circuit simulation. The QMDD structure is what allows the creation of such a method. Because of the ability of a QMDD structure to represent an entire quantum circuit without a very large representational matrix, a much larger circuit can be represented in a much smaller memory space.

In order for this method to work, the QMDDmultiply function written for the QMDD software package is used. This function is described in detail in [7], but the main idea behind it is the ability to multiply two QMDD structures and to return a QMDD structure as a result. This is equivalent to multiplying the matrices that each QMDD

represents, but by using the QMDD multiplication function, the memory storage for the matrices is not necessary.

In order to perform the necessary QMDD multiplication operations, it is first necessary to represent the column vectors representing |0> and |1> as QMDD structures. Since the column vectors for each of these eigenstates only utilize two matrix elements, only the 0- and 2-edges of a QMDD vertex are utilized. The other two edges of the QMDD vertex are set to point at NULL in order to conserve more memory. These structures are shown in Figure 4.5.

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

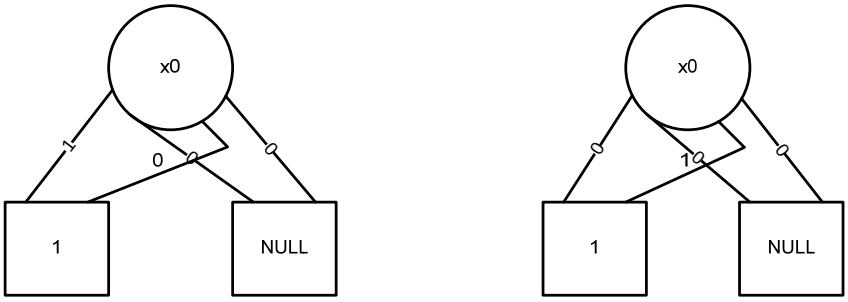


Figure 4.5. QMDD Structures for |0> and |1>

In addition to the QMDDmultiply function, there is also another function in the QMDD software package which represents a Kronecker multiplication of two QMDD structures. This function is used to combine multiple QMDD structures into a single

QMDD representing the input vector. After each of the qubits is represented as a QMDD structure, the structures are combined through Kronecker multiplication in a way similar to that mentioned in Section 4.2 of this thesis. The result of the Kronecker operations is a single QMDD structure representing the input vector.

After the QMDD representing the input vector has been formed, it is necessary to create the QMDD that represents the entire quantum circuit. A call to a function within the QMDD software package is sufficient to perform this task. At this point, there are two separate QMDD structures: one to represent the input vector and one to represent the quantum or reversible circuit.

The next step is to perform the actual multiplication of the QMDD structures. This represents the same procedure as multiplying the matrices that each of the QMDD structures represent. The actual multiplication is carried out by a function in the QMDD software package [7].

The result of this operation is another QMDD structure. This structure represents the resulting matrix after the matrix multiplication has been performed. As described above, some of the edges of the QMDD structures representing the input vector are pointing at NULL. Because of this, the resulting QMDD will represent a matrix that has some invalid values. The only part that is used for output in the resulting matrix is the first column. A function already exists in the QMDD software package that traverses a QMDD structure and builds the representational matrix. For the purposes of this simulator, this function has been modified such that it traverses the QMDD structure in a way that only the first column of the representational matrix is extracted. By only extracting the first column of the result matrix, it is not necessary to utilize storage for the

entire $2^n \times 2^n$ matrix. It is wasteful to extract the entire matrix since it consumes great quantities of memory space.

The final step of the simulation is similar to that of the matrix multiplication method. The output vector represents the result of the multiplication, but it is necessary to convert this result back to Dirac-ket notation so it is understandable to the user. The same algorithm which was used before is used to perform this conversion. The results may then be presented to the user.

The explicit multiplication based simulator is exemplified below. For simplicity, the same circuit that was used as an example for the matrix multiplication method will be used as an example here (Figure 4.4). The first step to be performed is to convert the input vector into a QMDD structure. The QMDD structure representing the eigenstate |101> is given in Figure 4.6 below:

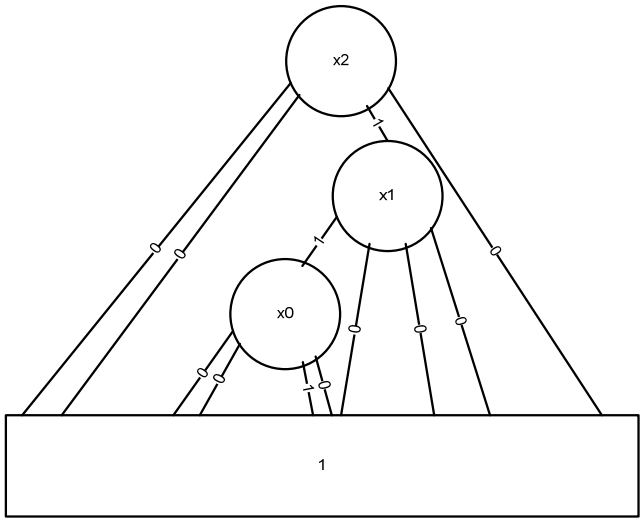


Figure 4.6. QMDD Representing Input Vector |101>

Additionally, it is necessary to build the QMDD structure which represents the entire quantum or reversible circuit. In terms of this example, the QMDD structure which represents the circuit is given in Figure 4.7 below:

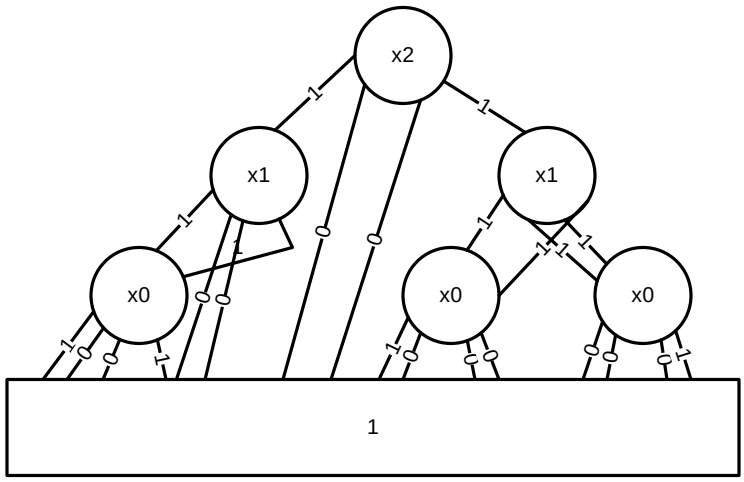


Figure 4.7. QMDD Representing Quantum Circuit

The next step is to multiply the two structures together using the function that has been previously written for the QMDD software package. In this particular case, the multiplication gives the following resulting QMDD:

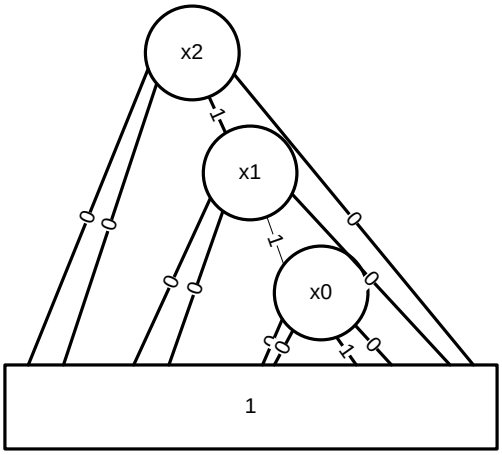


Figure 4.8. Result QMDD

Finally, the first column of the result matrix is extracted from the QMDD. This column is then translated to Dirac-ket notation, giving the resulting eigenstate of |111>, which is the same result achieved from the example computation given in section 4.3.2.

4.3.5. Implicit Multiplication Based Simulator

The other QMDD method of circuit simulation that is presented in this thesis is one that performs an implicit multiplication of QMDD structures. This method does not perform an actual multiplication, but rather it traverses the QMDD structure guided by the values of the initial state of the *n*-qubit register.

The first step of this simulation method is identical to the first step of the explicit multiplication method in that a QMDD structure which represents the entire quantum or reversible circuit is constructed. After this structure has been built, it is traversed in a

guided manner based on the input vector component values. The components of the output vector are computed during the traversal. The specific manner in which the QMDD structure is traversed is discussed next.

As an example, let *e* designate an edge which is pointing to a QMDD structure which describes a quantum circuit. Additionally, let *vector* represent an array consisting of the initial eigenstates of the qubits. Also, T(*e*) represents a function that returns a Boolean value. The function returns true if and only if the edge *e* points to the terminal node in the QMDD structure, otherwise it returns false. The recursive traversal function is described in the pseudocode below:

1. If T(*e*) is true, then the weight of *e* is stored in the i^{th} position of the output vector.

2. If T(*e*) is false, then if the value in *vector* corresponding to the variable of the current vertex = 0, the following procedure is followed:

    a. Call the recursive function with the 0^{th} edge of the current vertex.
    b. Call the recursive function with the 2^{nd} edge of the current vertex.
    Otherwise,
    c. Call the recursive function with the 1^{st} edge of the current vertex.
    d. Call the recursive function with the 3^{rd} edge of the current vertex.

Figure 4.9. Traversal Algorithm

Using either the 0 edge and the 2 edge or the 1 edge and the 3 edge is due to the structure of the matrices and the form of the resulting product of the matrices with vectors representing |0> and |1>. A square matrix with dimensions that are powers of two can be partitioned into four equal parts. When this square matrix is multiplied by a vector representing either a zero or one qubit, only half of the matrix is represented in the

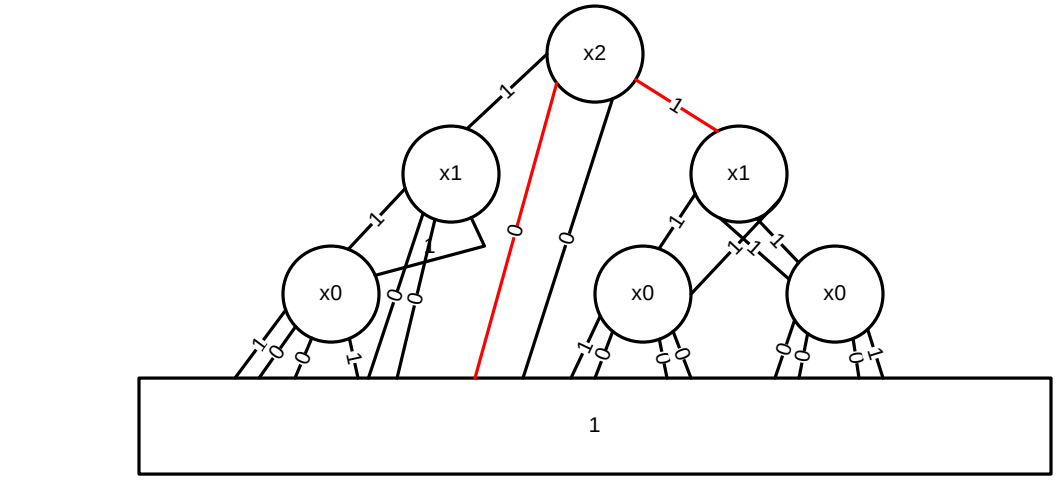
resulting vector. The example below shows how the particular edges are selected based on matrix multiplication.

$$\begin{bmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_3 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_0 \\ \mathbf{M}_2 \end{bmatrix}$$

The example below shows in detail how the matrix traversal is performed. For simplicity, the same quantum circuit will be used as that in Figure 4.4. Once again, the QMDD representing the quantum circuit must be created. Since the same quantum circuit is used as in the previous example, the QMDD in Figure 4.7 applies to this example as well.

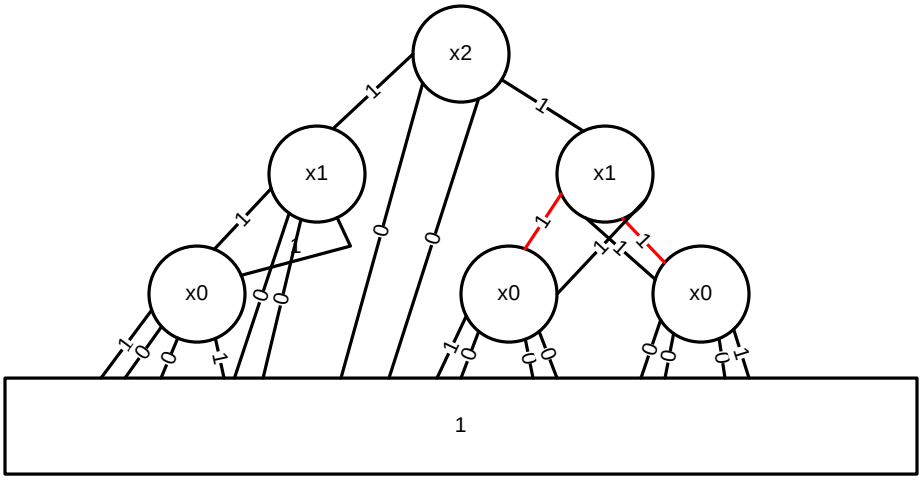
The input qubits will be the same as those used in the previous example (|101>). The algorithm given in Figure 4.9 is used here to traverse the QMDD structure. The first node that is traversed is not a terminal node. Therefore, step 2 of the algorithm is followed. The first qubit in the input is |1>, thus the 1 and 3 edges of the vertex are traversed. Figure 4.10 shows the first traversal as well as the output vector after this first step.



0	0	0	0				

Figure 4.10. Result After First Step

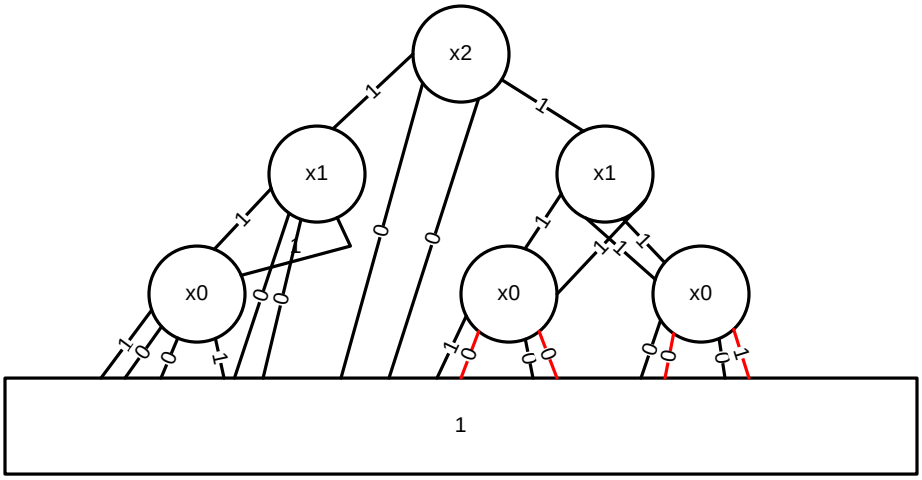
After the first step of the algorithm, the first edge which is traced points straight to the terminal node. This means that the first half of the output vector is set to the weight of that particular edge. Since that edge has a weight of 0, the first four elements of the output vector are set to 0. The other edge which is traversed points to a non-terminal node. The algorithm is recursively applied to that node now. Figure 4.11 shows the results.



0	0	0	0				

Figure 4.11. Result After Second Step

With this particular step in the algorithm, the edges leading from the vertex are non-terminal, so no new values are filled into the output vector. The input qubit for this variable is |0>, so the 0 and 2 edges are traversed. The final step is shown in Figure 4.12.



0	0	0	0	0	0	0	1

Figure 4.12. Result After Final Step

In the final step of the algorithm, the *1* and *3* edges of each of the nodes are traversed since the input qubit is |1>. Each of the edges coming out of these nodes are terminal edges, so their values are recorded in the output vector. It can be seen that the weights of these edges (from left to right) are 0, 0, 0, and 1. Thus, the output vector has all of its entries completed. Now, the output vector may be condensed into Dirac-ket notation, and the simulation is complete. This simulation method does not actually perform a multiplication (using QMDD structures or matrices); instead, traversals of the QMDD structure create the same result as the other methods.

Chapter 5

RESULTS AND CONCLUSIONS

In order to evaluate the performance of the quantum logic simulations, it is necessary to examine the results generated during the experimentation and to compare them with other simulation methods. This chapter compares the two QMDD methods discussed in Chapter 4 to determine which is the best. Additionally, these methods are also compared with the naïve linear algebra method and with the QuIDDPro simulator, a previously developed quantum circuit simulator [20].

5.1. Results of Building QMDD Structures

All of the results presented in this section have been generated using a Dell PE 2650 computer with a Dual Intel Xeon processor running at 3.2GHz with 4 GB of RAM running a Linux operating system.

When examining the results of the tests performed on each method, there are a number of observations that can be made. As mentioned in the description of the linear algebra approach, this approach is inefficient in terms of memory. The size of the representational matrix for a quantum or reversible circuit with n qubits is $2^n \times 2^n$. Thus, when the size of the circuit grows larger than approximately seven qubits, the memory demand is too high, and the simulator simply cannot complete the simulation. Based on

this, the linear algebra approach is not a valid method for quantum circuit simulation, but it is used as a basis of comparison for the QMDD-based methods.

Other observations about the circuit simulation methods can be made by looking at the results of the sample simulations. Before the simulation can be completed using a QMDD-based approach, it is necessary to build the QMDD structure which represents the quantum or reversible circuit. Table 5.1 details the experimental results generated during the building of the QMDD structure.

Circuit Name	Number of Qubits	Number of Gates	Peak Node Count to Build Circuit		Time to Build Circuit		Resultant QMDD Size
			GC on	GC off	GC on	GC off	
Ham3	3	5	50	50	<0.000	<0.000	10
3_17	3	6	53	53	<0.000	<0.000	10
rd32	4	4	52	52	<0.000	<0.000	9
mod5adders	6	21	303	303	<0.000	<0.000	38
5mod5tc	6	17	394	394	<0.000	<0.000	28
Hwb7	7	289	8746	8746	0.025	0.025	179
rd53rcmg	7	30	1220	1220	<0.000	<0.000	103
Hwb9-1541	9	1541	50000	128731	0.494	0.535	683
Hwb11	11	9314	50000	2205109	11.536	66.544	2639
0410184.nct	14	46	1156	1156	<0.000	<0.000	39
ham15a	15	132	50000	161493	0.496	0.475	26346
Cycle17_3	20	48	3040	3040	0.004	0.004	236
mod1048576 adder	40	210					

Table 5.1. Experimental Results for Building QMDD Structure

5.1.1. Results of Building QMDD Structures

Table 5.1 gives the results for a number of quantum circuits taken from Maslov's quantum circuit benchmark website [9]. The first column of this table gives the name of the quantum benchmark circuit. The second column of the table gives the number of

qubits in the benchmark circuit. The circuits are listed such that the number of qubits is given in ascending order. Next is the number of gates in the quantum circuit cascade. This information is provided to give a general sense of the size of the circuit that is being dealt with.

The next pair of columns detail the peak node count during the building of the QMDD structure which represents the circuit. Two results are given; one column represents when garbage collect is turned on and the other represents when garbage collect is turned off. The garbage collect threshold is set at 50,000 nodes, so this explains why circuits such as hwb11 have a peak node count of 50,000 when garbage collect is turned on. In most cases, the garbage collect procedure is never activated which explains why the peak node counts are the same regardless of whether or not garbage collect is turned on. In the case of some of the largest circuits, the peak node count without garbage collect is much larger than 50,000.

The next set of columns indicates the time taken to build the QMDD structure representing the quantum circuit. These results are given both with garbage collection on and off. It can be seen that for many of the circuits with a smaller number of gates, the time to build the circuit is negligible. For larger circuits, the amount of time to build the circuit generally increases as the number of quantum gates increases. (Throughout this chapter, any operations that are not complete within a time period of one hour are listed in the results tables as “timeout”.) Also, the difference when garbage collection is turned on can be seen. In almost all cases, garbage collection reduces the amount of time necessary to build the structure.

Finally, the last column indicates the resultant size of the QMDD structure. This value indicates the number of nodes in the QMDD structure after it has been completely built. This value is not dependant on garbage collection. Notice that this value is significantly smaller than the peak node count. This is due to the excessive number of intermediary nodes that are created during the building of a QMDD structure.

5.1.2. Observations on Building QMDD Structures

A number of observations can be made about the results presented in Table 5.1. First of all, notice that garbage collection can actually have a significant impact on the building of a QMDD structure. The high peak node counts of some of the QMDD structures can make the building process very inefficient. Garbage collect helps to alleviate some of the stress put on the software package by keeping the number of nodes in the unique table relatively low.

Additionally, observations can be made about the time required to build a QMDD structure. When combined with the results in the next section, Table 5.1 demonstrates that most of the time required to simulate a quantum circuit comes from the building of the structure. Once the structure is built, the simulation takes less time as compared to the building of the structure.

5.2. Quantum Circuit Simulation Results

In addition to the results already discussed, Table 5.2 presents the results of the simulation (after the QMDD structure has already been built). The results presented here focus mostly on the timing of the various simulation methods discussed.

Circuit Name	Number of Qubits	Number of Gates	Number of Input Vectors	Average CPU Runtime for Linear Algebra Approach (sec)	Average CPU Runtime for QMDD Explicit Multiplication Approach (sec) GC On	Average Resultant QMDD Size for Explicit Multiplication Approach	Peak Node Count During Explicit Multiplication Approach	Average CPU Runtime for Implicit Multiplication Approach (sec) GC On
Ham3	3	5	8	< 0.000	0.00004	11.000	20	0.00005
3_17	3	6	8	< 0.000	0.00007	12.000	21	0.00005
rd32	4	4	16	<0.000	0.00006	17.875	28	0.00006
mod5adders	6	21	64	<0.000	0.00007	45.844	51	0.00007
5mod5tc	6	17	64	<0.000	0.00006	38.000	52	0.00007
Hwb7	7	289	128	0.040	0.00014	219.500	309	0.00016
rd53rcmg	7	30	128	<0.000	0.00010	126.500	188	0.00008
Hwb9-1541	9	1541	512	timeout	0.00086	850.551	1176	0.00084
Hwb11	11	9314	2048	timeout	0.00988	3311.512	4557	0.01028
0410184.nct	14	46	16384	timeout	0.00009	74.752	105	0.00207
ham15a	15	132	328	timeout	1.00677	36617.000	49594	0.99800
Cycle17_3	20	48	1048	timeout	0.00026	255.000	277	0.12637
mod1048576 adder	40	210		timeout	timeout			timeout

Table 5.2. Simulation Results Using QMDD Structures

5.2.1. Discussion of Quantum Circuit Simulation Results

For the results presented in Table 5.2, there are a number of observations that can be made. The first three columns of the table are the same as those from Table 5.1. The same quantum circuits are used so that comparisons may more easily be made. The next column presents the number of input vectors used for simulation. For the circuits with 11 qubits or fewer, all of the possible test vectors are used. This means that every possible set of inputs are simulated. The average of the simulation times for each simulation is

presented in the table. For some of the larger circuits, only a subset of the possible input vectors are used. For example, the circuit "cycle17_3" has 20 qubits. This means that there are a total of 2^{20} possible input vectors. In order to simulate all of the possible vectors (over one million), it would take well over 36 hours. Due to this fact, a subset of 1048 vectors is used to represent all possible input vectors.

The next column represents the average CPU runtime for the linear algebra approach. For many of the small circuits, this method works in a reasonable amount of time. However, for circuits with more than 7 qubits, the simulation using the linear algebra method took over the threshold time of one hour, with most of the circuits never completing. This is due to the fact that no linear algebra techniques to simplify matrix multiplication are used. In this table, all simulations that did not complete in a time of one hour were recorded as "timeout". The results further the idea that the linear algebra method is not an efficient method for quantum circuit simulations.

The next column shows the average runtimes for the explicit multiplication method. As mentioned earlier, the majority of the simulation time is used building the QMDD structure. These results support this assertion. For the majority of the circuits, the simulation time is small. Coupled with these results, the next column shows the average number of nodes in the resultant QMDD structure. This is the size of the QMDD structure (including the original circuit QMDD) after the explicit multiplication has been performed. Since each input vector generates a distinct QMDD structure after the multiplication process, it makes sense to present this as an average. When this column and the previous column are observed together, significant observations can be made. It

can be seen that the time for simulation is directly proportional to the number of nodes in the resultant QMDD structure.

The next column indicates the peak node count during the explicit multiplication process. The explicit multiplication process manipulates the QMDD structure, and similar to when the structure is built, there are many intermediate nodes that are used during the manipulation. The peak node count given in this column represents the peak node count for all of the possible input vectors. This "peak, peak node count" is given here because there is a less than 5% variance among the peak node counts for all of the possible input vectors. This column, along with the previous column, demonstrates that the manipulations involved in performing the explicit multiplication operation generate intermediate nodes that are not a part of the final QMDD structure.

Finally, the last column indicates the amount of time necessary for the simulation using the implicit multiplication method. When these values are compared to the explicit simulation method, it is clear that the times are very close to each other. These results make deciding on a simulation method difficult. These results are discussed in the next section.

5.3. Comparison with QuIDDPro

The experimental results have been presented for the QMDD-based quantum circuit simulation methods, but it is still desired to compare these results with the results produced by QuIDDPro. Table 5.3 sums up the results of the QMDD-based methods compared with QuIDDPro.

Circuit Name	Number of Qubits	Number of Gates	Number of Input Vectors	Average CPU Runtime for QuIDDPro Simulation (sec)	Average CPU Runtime for QMDD Explicit Multiplication Approach (sec) GC On	Average CPU Runtime for Implicit Multiplication Approach (sec) GC On
Ham3	3	5	8	0.060004	0.00004	0.00005
3_17	3	6	8	0.064004	0.00007	0.00005
rd32	4	4	16	0.088006	0.00006	0.00006
mod5adders	6	21	64	0.136009	0.00007	0.00007
5mod5tc	6	17	64	0.340021	0.00006	0.00007
rd53rcmg	7	30	128	0.556034	0.00010	0.00008
0410184.nct	14	46	16384	3.040190	0.00009	0.00207
ham15a	15	132	328	4.628290	1.50277	1.49400
Cycle17_3	20	48	1048	0.812051	0.00426	0.13037

Table 5.3. Comparison of QMDD and QuIDDPro Simulators

For comparison purposes, a subset of the benchmark circuits used in the previous tables are included in this table. The first four columns are the same as those in Table 5.2. The next column shows the average runtime for the QuIDDPro quantum circuit simulator. The remaining two columns show the average runtimes for these same benchmark circuits using the two QMDD-based simulation approaches. These three columns include the time used to read in the circuit file, build the structure representing the circuit, and perform the quantum circuit simulation.

When the last three columns are compared, it can be seen that the QuIDDPro simulation takes longer than the simulation based on the QMDD structure. For each benchmark circuit, the QMDD methods are at least three times faster. In some cases, the improvement is as much as four orders of magnitude. In all cases, the QMDD based methods prove to be quicker than the QuIDDPro methods.

5.4. Conclusions

A quantum circuit simulator is a critical tool for the development of quantum or reversible circuits. The quantum circuit simulator presented in this thesis uses the Quantum Multiple-valued Decision Diagram to expedite the simulation process. As compared to other simulation processes such as QuIDDPro, this circuit simulator has shown to have faster simulation times.

Some important discoveries have been made during the development of this simulator. It is still unclear which method is the most efficient for quantum and reversible circuit simulation. As demonstrated in this thesis, the implicit and explicit multiplication methods are similar in their execution times. There are only slight differences in the result times that are produced. There is still much study that can be done to determine which of these methods is more efficient, but it is shown in this chapter that these two methods have distinct advantages over strictly using matrices to represent the quantum circuit and input vector and over the QuIDDPro simulation.

5.5. Future Work

The concept of quantum circuit simulation can still be scrutinized further in future research. One particular area that is of interest for quantum circuit simulations is the idea of dynamic variable reordering. For this thesis, all of the QMDD structures have a variable ordering such that all variables appear in order from the terminal node to the initial node. It has been shown in [24] that reordering the variables in a binary decision diagram changes the number of nodes in the diagram. More recently in [25], it has been shown that the same idea applied to the QMDD structure. Future work can be done in the

area of variable sifting to determine a variable ordering for each quantum circuit that will minimize the simulation time. When the idea of sifting is combined with quantum circuit simulation, it could allow for the efficient simulation of quantum circuits with even more qubits than those analyzed in this research.

REFERENCES

[1] P. Shor, Polynomial-time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *SIAM Journal of Computing*, vol. 26, 1997, pp. 1484-1509.

[2] L. Grover, A Fast Quantum Algorithm for Database Search, In Proceedings of the *ACM Symposium on Theory of Computing*, 1996, pp. 212-219.

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

[4] D. M. Miller and M. A. Thornton, QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits, In Proceedings of the *IEEE International Symposium on Multiple-Valued Logic*, 2006.

[5] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Form. Methods Syst. Des.*, 10(2-3): 149-169, 1997.

[6] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *27th Design Automation Conference*, June 1990.

[7] D. M. Miller, M. A. Thornton, and D. Goodman, A Decision Diagram Package for Reversible and Quantum Circuits, In Proceedings of the *IEEE World Congress on Computational Intelligence*, 2006.

[8] R. Horn and C. Johnson, **Topics in Matrix Analysis**, Chapter 4, Cambridge University Press, 1991.

[9] D. Maslov. Reversible Logic Synthesis Benchmarks Page. http://www.cs.uvic.ca/~dmaslov, 2005.

[10] I. Chuang. Quantum Circuit Viewer: qasm2circ, Massachusetts Institute of Technology, http://www.media.mit.edu/quanta/qasm2circ/.

[11] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Improving Gate-Level Simulation of Quantum Circuits," *Quantum Information Processing* vol. 2(5), October 2003, pp. 347-380.

[12] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Gate-Level Simulation of Quantum Circuits," *In Proc. of the Asia South Pacific Design Automation Conference*, pp. 295-301, January 2003.

[13] D. Goodman, M. A. Thornton, D. Y. Feinstein, D. M. Miller, "Quantum Logic Circuit Simulation Based on the QMDD Data Structure," *Reed-Muller Workshop, International Symposium on Multiple-Valued Logic*, May 16, 2007.

[14] R. Landauer, Irreversibility and Heat Generation in the Computing Process, *IBM Journal of Research Development*, vol. 5, 1961, 183.

[15] H. Weyl, **The Theory of Groups and Quantum Mechanics**. Dover Press, 1950.

[16] D. M. Miller, D. Y. Feinstein, and M. A. Thornton, "Variable Reordering and Sifting for QMDD," *IEEE Symposium on Multiple Valued Logic (ISMVL)*, May 2007.

[17] F. Somenzi. CUDD: Cu decision diagram package – release 2.4.1. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html, 2005.

[18] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Form. Methods Syst. Des.,* 10(2-3):171-206, 1997.

[19] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

[20] G. F. Viamontes, I. L. Markov, and J. P. Hayes. QuIDDPro: High-performance quantum circuit simulation. http://vlsicad.eecs.umich.edu/Quantum/qp/, 2005.

[21] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum-logic circuits. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* volume 25 issue 6, June 2006.

[22] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, Spectral Transforms for Large Boolean Functions with Application to Technology Mapping, In Proceedings of the *Design Automation Conference*, 1993, pp. 54-60.

[23] R. E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. 35, no. 8, 1986, pp. 677-691.

[24] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams", In *Proceedings of the International Conference on Computer-Aided Design*, Santa Clara, CA, Nov. 1993, pp. 42-47.

[25] D. M. Miller, D. Y. Feinstein, M. A. Thornton, Variable Reordering and Sifting for QMDD, IEEE International Symposium on Multiple Valued Logic (ISMVL), May 14-16, 2007.