𝕋ℂ𝕊-𝕋ℝ-𝔸-**07-33**

𝕋ℂ𝕊 Technical Report

An Efficient Decision Diagram Structure for Design Verification of Quantum Circuits under a Practical Restriction

by

Shigeru Yamashita, Shin-ichi Minato, and D. Michael Miller

Division of Computer Science

Report Series A

December 30, 2007

Hokkaido University
Graduate School of
Information Science and Technology

Email: minato@ist.hokudai.ac.jp

Phone: +81-011-706-7682

Fax: +81-011-706-7682

An Efficient Decision Diagram Structure for Design Verification of Quantum Circuits under a Practical Restriction

SHIGERU YAMASHITA
Graduate School of Information Science
Nara Institute of Science and Technology
Ikoma, Nara 630-0192, Japan.

SHIN-ICHI MINATO
Division of Computer Science
Hokkaido University
Sapporo 060-0814, Japan

D. MICHAEL MILLER
Faculty of Engineering
University of Victoria
Victoria, BC, V8W 3P6, Canada

December 30, 2007

(Abstract) Recently much attentions have been paid to quantum circuit design to prepare for the future "quantum computation era." Like the conventional logic synthesis, it should be important to verify and analyze the functionalities of generated quantum circuits. Therefore, we propose an efficient verification method for quantum circuits under a practical restriction. Thanks to the restriction, we can introduce an efficient verification scheme based on decision diagrams called Decision Diagrams for Matrix Functions (DDMFs). Then, we show analytically what are the advantages of our approach based on DDMFs over the previous techniques. In order to introduce DDMFs, we also introduce new concepts, quantum functions and matrix functions, which may also be interesting and useful on their own for designing quantum circuits.

1 Introduction

Recently *quantum computing* has attracted great attention by its potential abilities [11]. To realize a quantum algorithm, it is necessary to design the corresponding *quantum circuit* as small as possible. Thus, it should be very important to study quantum circuit design methods even before quantum computing is physically realized. Indeed, there have been many researches [6, 12, 7, 1, 8, 5, 13, 10] for quantum circuit design.

Typical quantum circuit design methods are based on *matrix decomposition* [13, 10] since a quantum algorithm is expressed by a matrix. They can treat any kind of quantum circuits, but they cannot treat large (hence, practical) size problems since they need to express matrices explicitly and thus they need exponential time and memory. (Note that a matrix for an n-bit quantum circuit is $2^n \times 2^n$, which will be explained later.)

There is a different approach for quantum circuit design [6, 12, 7, 1, 8, 5]. The approach is to focus on quantum circuits calculating only (classical) Boolean functions by the following observation [6]: A quantum circuit for a standard quantum algorithm essentially differs in the part of the circuit that calculates (classical) Boolean functions depending on the specification of a given problem instance (e.g., so called *oracles* in the case of Grover Search). Thus, we focus on designing a Boolean function in the model of quantum computation. Then, we may borrow many ideas from (classical) logic synthesis, especially reversible logic synthesis, and indeed there have been many researches in the conventional logic synthesis research community [6, 12, 7, 1, 8, 5]. Thus, to focus on such quantum circuits for Boolean functions, we may have a design method to handle practical size problems. This paper considers this type of quantum circuits.

Recently a paper [16] discussed an problem of the equivalence check of *general* quantum circuits and quantum states with considering so-called *phase equivalence* property of quantum states. Even for quantum circuits calculating *only* Boolean functions, it should be very important to verify and analyze the functionalities of designed circuits as the case of classical logic synthesis. For example, we may consider the following situation: One of the possible realizations of quantum computation is considered to be so called a *linear-nearest-neighbor (LNN)* architecture in which the *quantum bits* *(qubits)* are arranged on a line, and only operations to neighboring qubits are allowed. Thus, we need to modify a designed quantum circuit so that it uses only gates that operate to two adjacent qubits. In such a case, we may use some tricky transformations, and it is very convenient if we have a verification tool to confirm that the original and the modified quantum circuits are functionally equivalent.

If we consider only the classical type gates, it is enough to use the conventional technique such as Binary Decision Diagrams (BDDs) [2] for the verification. However, even if we consider quantum circuits calculating only Boolean functions, it is known that non-classical (quantum specific) gates are useful to reduce the circuit size [1, 8, 5, 4]. Thus we need to verify quantum circuits with non-classical gates. In such cases, a classical technique is not obviously enough.

As for simulating quantum circuits, there have been proposed efficient techniques using decision diagrams (called Quantum Information Decision Diagrams (QuIDDs) [15] and Quantum Multiple-valued Decision Diagrams (QMDDs) [9]) to represent matrices for quantum circuits. By using these efficient diagrams, we can express the functionalities of two quantum circuits, and then verify the equivalence of the two circuits. However, they are originally proposed to simulate *general* quantum circuits, and thus there may be a more efficient method that is suitable for verifying the functionalities of quantum circuits only for Boolean functions.

Our contribution described in this paper. Considering the above discussion, we introduce a new quantum circuit class: *Semi-Classical Quantum Circuits (SCQCs)*. Although SCQCs have a restriction, the class of SCQCs covers all the useful quantum circuits (for calculating Boolean functions) as will be explained later. Moreover, because of the restriction of SCQCs, we can express the functionalities of SCQCs very efficiently as the conventional techniques by BDDs. For that purpose, we introduce a new decision diagram structure called *a Decision Diagram for a Matrix Function (DDMF)*. Then, we show that the verification method based on DDMFs are much more efficient than the above mentioned methods based on previously known techniques. We provide an analytical comparison between DDMFs and QuIDDs, and reveal the essential difference: (1) We show that their ability to express the functionality of one quantum gate is essentially the same, but (2) we also show that our approach is much more efficient for the verification of SCQCs than a method based on known techniques. (Note that this does not mean that DDMFs are better than QuIDDs: DDMFs are only for SCQCs, whereas QuIDDs can treat all kinds of quantum circuits.) Moreover, we show by preliminary experiments that DDMFs can be used to verify SCQCs of practical size (60 inputs and 400 gates). In order to introduce DDMFs, we also introduce new concepts, *quantum functions* and *matrix functions*, which may be interesting and useful on their own for designing quantum circuits with quantum specific gates.

2 Semi-Classical Quantum Circuits and Their Representations by Decision Diagrams

This section introduces new concepts: SCQCs together with quantum functions, matrix functions and DDMFs.

2.1 Quantum States and Quantum Gates

Before introducing our new concepts, let us briefly explain the basics of quantum computation.

In quantum computation, it is assumed that we can use a *qubit* which is an abstract model of a *quantum state*. A qubit can be described as $\alpha\left|0\right\rangle + \beta\left|1\right\rangle$, where $\left|0\right\rangle$ and $\left|1\right\rangle$ are two basic states, and α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. It is convenient to use the following vectors to denote $\left|0\right\rangle$ and $\left|1\right\rangle$, respectively: $\left|0\right\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and, $\left|1\right\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. Thus, $\alpha\left|0\right\rangle + \beta\left|1\right\rangle$ can be described as a vector: $\alpha\left|0\right\rangle + \beta\left|1\right\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix}$. Accordingly, any quantum operation on a qubit can be described as a 2x2 matrix. By the laws of quantum mechanics, the matrix must be *unitary*. We call such a quantum operation a *quantum gate*. For example, the operation which transforms $\left|0\right\rangle$ and $\left|1\right\rangle$ to $\left|1\right\rangle$ and $\left|0\right\rangle$, respectively, is called a NOT gate whose matrix representation is as shown in Fig. 1.

In addition to the above NOT gates, we can also use any quantum specific unitary matrix in quantum circuits. For example, *rotation gates* denoted by $R(\theta)$ are often used in quantum computation. The matrix for the gates is as shown in Fig. 1. Although the functionality of rotation gates is not classical, they are useful to design quantum circuits for (classical) Boolean functions [1]. Another quantum specific gate called a V gate is also utilized to design quantum circuits for Boolean functions [8, 5]. The matrix for the gate is as shown in Fig. 1. This gate has the interesting property that $V^2 = NOT$.

In the following, our primitive gates are (generalized) *controlled-U gates* which are defined as follows:

Definition 1 *A controlled-U gate has (possibly many) positive and negative control bits, and one target bit. It applies a 2×2 unitary matrix U to the target qubit when the states of all the positive control bits are the states $|1\rangle$ and the states of all the negative control bits are the state $|0\rangle$. A controlled-U gate may not have a control bit. In such a case, it always applies U to the target qubit.*

See an example of a quantum circuit consisting of two controlled-NOT gates in Fig. 2. In quantum circuits, each gate works one by one from the left to the right. For the first gate, the target bit is x_3 and the symbol $\oplus$ means the NOT operation. The positive control bits are x_1 and x_2 denoted by black circles. This gate performs NOT on $|x_3\rangle$ only when both $|x_1\rangle$ and $|x_2\rangle$ are the state $|1\rangle$. Consider the second gate in the same figure. The white circles denote negative controls, which means the gate performs NOT only when both $|x_1\rangle$ and $|x_2\rangle$ are the states $|0\rangle$.

In addition to controlled-NOT gates which are essentially classical gates, we can consider any (quantum specific) unitary operation for controlled gates. For example, the functionalities of controlled gates in Figs. 3 and 4 are various (e.g., NOT, V, V^{-1}, $R(\frac{1}{2}\pi)$ and $R(\frac{1}{4}\pi)$).

2.2 Semi-Classical Quantum Circuit (SCQC)

Consider Fig. 2 again. This circuit transforms the state of the third bit $|x_3\rangle$ into $|x_3 \oplus f(x_1, x_2)\rangle$, where $f(x_1, x_2) = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$. Thus, we can use this circuit (as a part of a quantum algorithm) to calculates the Boolean function $f(x_1, x_2) = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$.

$$NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad V = \begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$$

$$R(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$

Figure 1: Unitary matrices

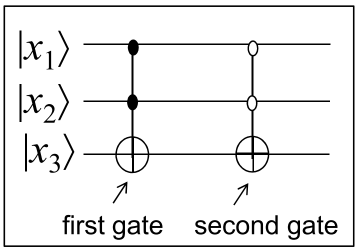

Figure 2: A quantum circuit

As mentioned before, although our goal is to construct such a quantum circuit that calculates a Boolean function, quantum specific gates (such as $R(\theta)$ and V) are useful [1, 8, 5, 4] to make the circuit size smaller. For example, the circuit as shown in Fig. 3 (reported in [5]) utilizes controlled-V and controlled-V^{-1} gates to become much smaller than the one we can do the best with only classical type gates, i.e., controlled-NOT gates. (That was confirmed by an essentially exhaustive search [5].)

In order to characterize such a quantum circuit that calculates a classical Boolean function with non-classical gates, we introduce a *Semi-Classical Quantum Circuit (SCQC)* whose definition is as follows.

Definition 2 *A Semi-Classical Quantum Circuit (SCQC) is a quantum circuit consisting of controlled-U*

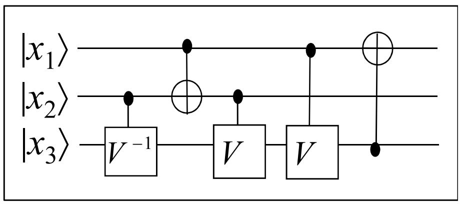

Figure 3: A quantum circuit for half-adder

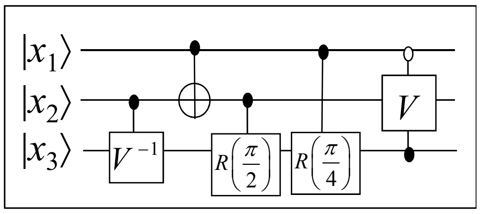

Figure 4: A non-SCQC

gates with the following restriction.

Restriction. *If all the initial input quantum states of the circuit are $|1\rangle$ or $|0\rangle$ (i.e., just classical values), the quantum states of the control qubits of any gate in the circuit should be $|1\rangle$ or $|0\rangle$ at the time when the gate is being operated.*

The circuit in Fig. 3 is an SCQC. This is because the quantum states of the control qubits of any gate are either $|1\rangle$ or $|0\rangle$ when the gate is being operated if the initial input state $|x_1\rangle$, $|x_2\rangle$ and $|x_3\rangle$ are either $|1\rangle$ or $|0\rangle$. It is not trivial to see the condition for the quantum state of the control qubit of the last gate ($|x_3\rangle$) in Fig. 3. However, by using our new concepts (explained in the next section), it is easy to verify that the state is indeed the classical value if the input states of the circuit are classical values.

On the contrary, the circuit as shown in Fig. 4 is not an SCQC. Again, by using our new concept it is easily verified that the condition is not satisfied for the quantum state of the control qubit of the last gate ($|x_3\rangle$) in Fig. 4.

Our motivation to introduce SCQCs is based on the following observations.

- Although SCQCs are in a subset of all the possible quantum circuits, quantum circuits (for calculating a Boolean function) designed by the existing methods are all SCQCs to the best of our knowledge.

- Even in the future it is very unlikely that we come up with a *tricky* design method that produces a non-SCQC to calculate (classical) Boolean functions. The reason is as follows. If the circuit is not a SCQC, there is a gate such that the quantum state of its control bit is not a simple classical value ($|0\rangle$ nor $|1\rangle$). In such a case, the quantum states of the control bit and the target bit after the gate cannot be considered separately: their states are not only non-classical values but also correlated with each other. Such a situation is called quantum *superposition* and *entanglement* [11]. Since the whole circuit should calculate a Boolean function, all of the final output quantum states should be again restored to simple classical values (i.e., $|0\rangle$ or $|1\rangle$). The reverse operations of creating quantum superposition and entanglement seems to be only one method to restore to a simple classical value. Thus, it seems nonsense to consider non-SCQC circuit when we discuss practical design method of a quantum circuit to calculate a Boolean function.

Important Note: The restriction of SCQCs means that we cannot make *entanglement* if all the initial input quantum states of a SCQC are just classical values. It is well-known that quantum computation without entanglement has no advantage over classical computation. However, this does not mean that SCQCs are meaningless by the following reason: As mentioned, a SCQC is used as a sub-circuit to calculate a Boolean function for some quantum algorithms. Thus, in the real situation where a SCQC is used as a sub-circuit, the inputs to the SCQC are not simple classical values,and so it indeed creates entanglement which should give us a quantum computation advantage. In other words, the restriction of SCQCs in the definition is considered when we suppose the inputs of SCQCs are just classical values, which may not be a real situation where SCQCs are really used.

Therefore, SCQCs should be enough if we consider to design a quantum circuit to calculate a Boolean function from the practical point of view. Moreover, the restriction of SCQCs provide us an efficient method to analyze and verify quantum circuit as we will see in Sec. 3. That is our motivation to introduce the new concept in this paper.

2.3 Quantum Functions and Matrix Functions

Before introducing our new representation of the functionalities of SCQCs, we need the following definitions.

Table 1: A truth tables for quantum, classical and matrix functions

x_1, x_2	qf_1	qf_2	mf_1	mf_2	$CM(I)$	$CM(R(\frac{1}{2}\pi))$
$0,0$	$\lvert 0\rangle$	$\lvert 1\rangle$	I	NOT	I	$R(\frac{1}{2}\pi)$
$0,1$	$V^{-1}\lvert 0\rangle$	$\lvert 0\rangle$	V^{-1}	I	I	$R(\frac{1}{2}\pi)$
$1,0$	$\lvert 0\rangle$	$\lvert 0\rangle$	I	I	I	$R(\frac{1}{2}\pi)$
$1,1$	$V^{-1}\lvert 0\rangle$	$\lvert 1\rangle$	V^{-1}	NOT	I	$R(\frac{1}{2}\pi)$

Definition 3 *A quantum function with respect to n Boolean variables $x_1, x_2, \cdots, x_n$ is a mapping from $\{0,1\}^n$ to a qubit state.*

See the third bit after the first gate in the circuit in Fig. 3 again. If the initial state of $|x_3\rangle$ is $|0\rangle$, the resultant state of the third bit can be seen as a quantum function described as $qf_1(x_1, x_2)$ in the second column of Table 1. For example, the resultant quantum state becomes $V^{-1}|0\rangle$ when $x_1 = 0, x_2 = 1$. Thus, $qf_1(0,1)$ is defined as $V^{-1}|0\rangle$ as shown in the table.

Note that a Boolean function can be seen as a special case of quantum functions. For example, the third column (qf_2) of Table 1 shows the quantum function of the resultant third qubit after the two gates of the circuit in Fig. 2 when the initial state of $|x_3\rangle$ is $|0\rangle$. This can be considered as the output of a Boolean function when $|0\rangle$ and $|1\rangle$ are considered as Boolean values 0 and 1, respectively. (As mentioned before, the circuit is considered to calculate the Boolean function: $x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$, which we consider essentially the same as (qf_2) in Table 1.)

The value of a quantum function $q(x_1, x_2, \cdots, x_n)$ can always be expressed as $mf(x_1, x_2, \cdots, x_n)|0\rangle$, where $mf(x_1, x_2, \cdots, x_n)$ is a mapping from $\{0,1\}^n$ to 2×2 unitary matrices. It is convenient to consider $mf(x_1, x_2, \cdots, x_n)$ instead of $q(x_1, x_2, \cdots, x_n)$ itself, thus we introduce the following definition.

Definition 4 *A matrix function with respect to n Boolean variables $x_1, x_2, \cdots, x_n$ is a mapping from $\{0,1\}^n$ to a 2×2 (unitary) matrix.*

The fourth and the fifth columns of Table 1 show the matrix function mf_1 and mf_2 for the quantum function qf_1 and qf_2, respectively, in the same table. In this paper, we treat a matrix function whose output values are only I or NOT as a classical Boolean function by considering that NOT and I of the matrix function correspond to 1 and 0, respectively, of the Boolean function.

Table 2: Operators $\oplus$ and $*$.

x_1, x_2	mf_1	mf_2	$mf_1 \oplus mf_2$	mf_3	f	$f * mf_3$
$0,0$	$R(\frac{1}{2}\pi)$	$R(\frac{1}{2}\pi)$	$R(\pi)$	$R(\frac{1}{2}\pi)$	1	$R(\frac{1}{2}\pi)$
$0,1$	I	I	I	I	0	I
$1,0$	I	$R(\frac{1}{4}\pi)$	$R(\frac{1}{4}\pi)$	$R(\pi)$	1	$R(\pi)$
$1,1$	$R(\frac{1}{2}\pi)$	$R(\frac{1}{4}\pi)$	$R(\frac{3}{4}\pi)$	$R(\pi)$	0	I

In other words, we represent a Boolean function by a matrix function as a special case.

We define a special type of matrix function called *constant matrix function* as follows.

Definition 5 *A matrix function $mf(x_1, x_2, \cdots, x_n)$ is called a constant matrix function if $mf(x_1, x_2, \cdots, x_n)$ are the same for all the assignments to $x_1, x_2, \cdots, x_n$. $CM(M)$ denotes a constant matrix function that always equals to the matrix M.*

The sixth and the seventh columns of Table 1 show the truth tables for constant matrix functions, $CM(I)$ and $CM(R(\frac{1}{2}\pi))$, respectively.

By using the matrix function mf_1 in the fourth column of Table 1, we can easily see how the first gate in Fig. 3 transforms the third qubit $|w\rangle$: $|w\rangle$ is transformed to $mf_1(x_1, x_2)|w\rangle$. For example, when $x_1 = 0, x_2 = 1$, $|w\rangle$ is transformed to $V^{-1}|w\rangle$.

Important Note: The above means that the representation (and so the analysis) by matrix functions works even when $|w\rangle$ is any general quantum state. Indeed, we can use a SCQC even when the input states are not simple classical values, i.e., the restriction of SCQCs does not say that SCQCs cannot be used when the inputs are not classical. (If so, we may not be able to use an SCQC for a part of a quantum algorithm.)

For matrix functions, we introduce two operators "$\oplus$" and '$*$,' which are used to construct DDMFs for a quantum circuit in the following sections.

Definition 6 *Let mf_1, mf_2 and mf_3 be matrix functions with respect to x_1 to x_n. Then $mf_1 \oplus mf_2$ is defined as a matrix function mf such that $mf(x_1, \cdots, x_n) = mf_1(x_1, \cdots, x_n) \cdot mf_2(x_1, \cdots, x_n)$ where $\cdot$ means normal matrix multiplication. Let also f be a Boolean function* *with respect to x_1 to x_n. Then $f * mf_3$ is a matrix function which equals to $mf_3(x_1, x_2, \cdots, x_n)$ when $f(x_1, x_2, \cdots, x_n) = 1$, and equals to I when $f(x_1, x_2, \cdots, x_n) = 0$.*

See examples in Table 2. Note that if mf_1 and mf_2 are considered to be Boolean functions like mf_2 in Table 1, the operator $\oplus$ corresponds to the EXOR of the two Boolean functions. Note also that if mf_3 is essentially a Boolean function like mf_2 in Table 1, the operator $*$ corresponds to the AND of the two Boolean functions.

2.4 Decision Diagrams for Matrix Functions

A matrix function for a quantum function can be expressed efficiently by using an edge-valued binary decision diagram structure, which we call a DDMF whose definition is as follows:

Definition 7 *A Decision Diagram for a Matrix Function (DDMF) is a directed acyclic graph with three types of nodes: (1) A single terminal node corresponding to the identity matrix I, (2) a root node with an incoming edge having a weighted matrix M, and (3) a set of non-terminal (internal) nodes.*

Each internal and the root node are associated with a Boolean variable x_i, and have two outgoing edges which are called 1-edge (solid line) leading to another node (the 1-child node) and 0-edge (dashed line) leading to another node (the 0-child node). Every edge has an associated matrix.

The matrix function represented by a node is defined recursively by the following three rules.

(1) The matrix function represented by the terminal node is the constant matrix function $CM(I)$.

(2) The matrix function represented by an internal node (or the root node) whose associated variable is x_i is defined as follows: $x_i * (CM(M_1) \oplus mf_1) \oplus \overline{x_i} * (CM(M_0) \oplus mf_0)$, where mf_1 and mf_0 are the matrix functions represented by the 1-child node and the 0-child node, respectively, and M_1 and M_0 are the matrices of the 1-edge and the 0-edge, respectively. (See an illustration of this structure in Fig. 5.)

(3) The root node has one incoming edge that has a matrix M. Then the matrix function represented by the whole DDMF is $CM(M) \oplus mf$, where mf is a matrix function represented by the root node.

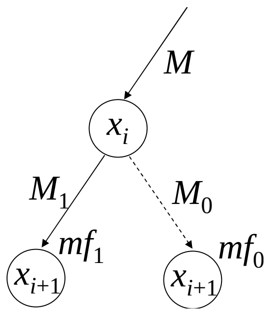

Figure 5: An internal DDMF node

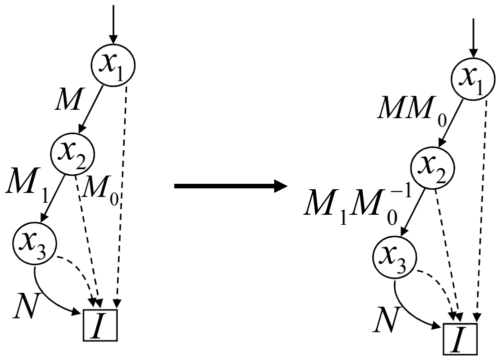

Figure 6: Conversion to the canonical form

Like conventional BDDs, we achieve the canonical form for a DDMF if we impose the following restriction on the matrices on all the edges.

Definition 8 *A (DDMF) is canonical when (1) all the matrices on 0-edges are I, (2) there are no redundant nodes: No node has 0-edge and 1-edge pointing to the same node with I as the 1-edge matrix, and (3) common sub-graphs are shared: There are no two sub-graphs.*

Any DDMF can be converted to its canonical form by using the following transformation from the terminal node to the root node: Suppose the matrices on incoming edge, 0-edge and 1-edge of a node be M, M_0 and M_1, respectively. Then, if M_0 is not I, we modify these three matrixes as follows: (1) The matrix on the incoming edge changed to be MM_0. (2) The matrix on the 1-edge changed to be $M_1M_0^{-1}$. (3) The matrix on the 0-edge changed to be I. It is easily verified that this transformation does not change the matrix function represented by the DDMF. See the example in Fig. 6 where the matrix on 0-edge of the node x_2 is converted to I. In the example, the matrices I on edges are omitted.

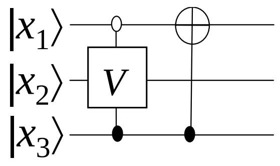

Figure 7: An SCQC (1)

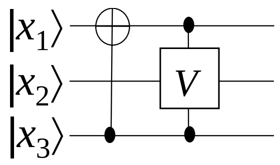

Figure 8: An SCQC (2)

Note: The concepts of *quantum functions* and *matrix functions* may be used implicitly in the design method of [1], and the decision diagram structure is similar between DDMFs and the quantum decision diagrams used in [1]. However, the quantum decision diagrams in [1] are used to represent conventional Boolean functions whereas DDMFs are used for representing matrix functions: the terminal node of a DDMF is a matrix I. Also a weight on an edge in DDMFs is generalized to any matrix. Thus, DDMFs can be considered as a generalization of quantum decision diagrams to treat matrix functions rather than Boolean functions. (As we have seen in Table 1, Boolean functions can be seen as a special case of quantum functions.)

We will use the same operators, $\oplus$ and $*$, for DDMFs as for matrix functions: (1) (DDMF for mf) = (DDMF for mf_1) $\oplus$ (DDMF for mf_2) if $mf = mf_1 \oplus mf_2$, and (2) (DDMF for mf) = (DDMF for mf_1) $*$ (DDMF for mf_2) if $mf = mf_1 * mf_2$.

3 Verification of SCQCs by using DDMFs

See two SCQCs in Fig. 7 and Fig. 8. It is easy to see that their functionality are the same. However, the problem is how to verify the equality for much larger circuits. Thanks to the introduction of DDMFs, we propose a method to verify the equality of given two n-qubit SCQCs in the following.

Step 1. We construct a DDMF to represent the matrix function that expresses the functionality for each qubit state at the end of each circuit.

Step 2. We compare two DDMFs for the corresponding qubits of the two circuits. The comparison of two DDMFs can be done in $O(1)$ time as in the case of BDDs.

Step. 1 is performed as follows. In the below, we use a notation D_i^j to express the DDMF for the i-th quantum qubit state right after the j-th gate. We also use a notation $F(D)$ to denote the matrix function (or the Boolean function in a special case) represented by a DDMF D.

Initialization. For each input x_i, we construct a D_i^0 as a DDMF for x_i. This is the DDMF for the matrix function (essentially Boolean function) which is NOT when $x_i = 1$.

Construction of the DDMFs right after the j-th gate. From the first gate to the last gate, we construct D_i^j from D_i^{j-1} as follows. If the i-th bit is not the target bit of the j-th gate, $D_i^j = D_i^{j-1}$. If the i-th bit is the target bit of the j-th gate $D_i^j = D_i^{j-1} \oplus D_{gate}$ where D_{gate} is constructed by the following two steps.

(1) For the j-th gate, let us suppose that the positive control bits be the $p_1, p_2, \cdots, p_k$-th bits, and the negative control bits be the $n_1, n_2, \cdots, n_l$-th bits. Then, by the restriction of SCQCs, all the matrix functions $F(D_m^{j-1})$ for $m = p_1, p_2, \cdots, p_k, n_1, n_2, \cdots, n_l$ are classical Boolean functions. Thus we can calculate a logical AND of them: $g = F(D_{p_1}^{j-1}) \cdot F(D_{p_2}^{j-1}) \cdots F(D_{p_k}^{j-1}) \cdot \overline{F(D_{n_1}^{j-1})} \cdot \overline{F(D_{n_2}^{j-1})} \cdots \overline{F(D_{n_l}^{j-1})}$. Note that this Boolean function can be obtained by DDMF operations since a DDMF represents a Boolean function in a special case.

(2) We construct $D_{gate} = (DDMF$ for $g)$ $*(DDMF$ for $CM(U))$, where U is a unitary matrix associated with the j-th gate.

Note that all the DDMF operations in the above should be performed efficiently by using *Apply* operations and *operation and node hash tables* as the conventional BDD operations [2].

We show an example of DDMFs for the quantum circuit as shown in Fig. 7. At the initialization step, we construct DDMFs for functions, x_1, x_2 and x_3, which are D_1^0, D_2^0 and D_3^0 as shown in Fig. 9. Then we

Table 3: A truth table for $D_{control}$

x_1, x_2, x_3	$F(D_{control})$
$0,0,0$	I
$0,0,1$	V
$0,1,0$	I
$0,1,1$	V
$1,0,0$	I
$1,0,1$	I
$1,1,0$	I
$1,1,1$	I

Table 4: A truth table for D_2^1

x_1, x_2, x_3	$F(D_2^1)$
$0,0,0$	I
$0,0,1$	V
$0,1,0$	N
$0,1,1$	VN
$1,0,0$	I
$1,0,1$	I
$1,1,0$	N
$1,1,1$	NN

construct the DDMFs for the quantum states right after the first gate. Since the target bit is the second bit for the first gate, $D_1^1 = D_1^0$, and $D_3^1 = D_3^0$. To construct D_2^1, we first calculate a Boolean function $g = \overline{F(D_1^0)} \cdot D_3^0 = \overline{x_1} \cdot x_3$. Note that this Boolean function can be obtained by DDMF operations since a DDMF represents a Boolean function as a special case. This is because the first bit and the third bit are negative and positive control, respectively. Then we construct $D_{gate} = g * (DDMF \text{ for } CM(V))$, whose matrix function is shown in Table 3. Finally, we construct $D_2^1 = D_2^0 \oplus D_{gate}$ whose matrix function is as shown in Table 4. The constructed DDMFs after the first gate are shown in Fig. 10.

4 Comparison with the previous methods

In this section we compare our method with the previous methods to show the advantage of our method.

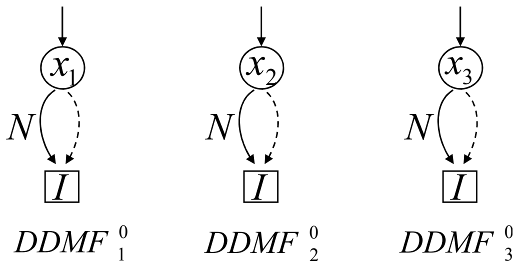

Figure 9: DDMFs after the initialization

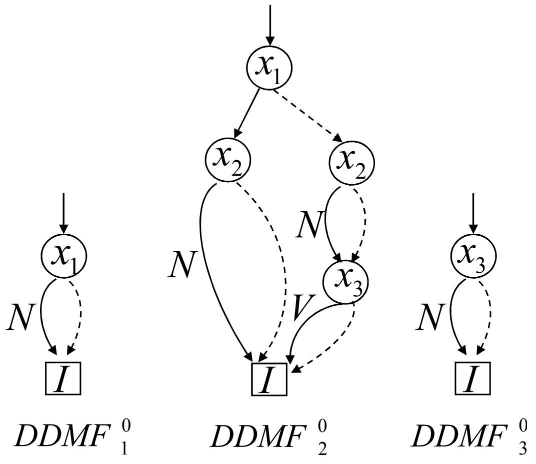

Figure 10: DDMFs after a gate

4.1 Verification method based on previous techniques

A gate (or a circuit) of n qubits can be described by a $2^n \times 2^n$ unitary matrix. For example, the unitary matrix that expresses the functionality of the last gate in Fig. 4 can be shown as in Fig. 11. (We will explain this briefly in Appendix.)

Since the same structure (sub-matrices) are often repeated in such a $2^n \times 2^n$ unitary matrices (like Fig. 11), there have been proposed data compression scheme based on decision diagram structures: QuIDDs [15] based on multi-terminal binary decision diagrams, and QMDDs [9] based on multiple-valued decision diagrams (MTBDDs) [3]. Here we explain how the data compression works for QuIDDs. QMDDs have a slightly different approach: They use multi-valued logic instead of binary logic, and the strategy of selecting decision variables is a bit different. However, it should be noted that the two approaches are essentially the same when (1) the target circuit is binary logic

$$
\begin{array}{c|cccccccc}
R_0R_1R_2 & & & & & & & & \\
000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 & 0 & 0 & 0 \\
100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
C_0C_1C_2 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111
\end{array}
$$

Figure 11: A Unitary Matrix for a 3-qubit gate

($|0\rangle$ or $|1\rangle$) valued (which is our case), and (2) the variable ordering is appropriately chosen for QuIDDs (as explained below).

A QuIDD for the matrix in Fig. 11 can be constructed as shown in Fig. 12. In a QuIDD representing a matrix, we have decision variables to specify rows (R_0, R_1, R_2) and columns (C_0, C_1, C_2) of the matrix as shown in Fig. 11. For example, the variable assignment to $R_0 = 0, R_1 = 1, R_2 = 1, C_0 = 0, C_1 = 0, C_2 = 1$) leads to the forth row and the second column element, $\frac{1-i}{2}$. We can construct a binary decision diagram where each variable assignment corresponds to one element of the matrix as shown in Fig. 12. If there are some repeated structures in a matrix, this diagram can reduce the necessary memory space to store the matrix information.

It is known that interleaving the row and the column variables (i.e., the order of $R_0, C_0, R_1, C_1, \cdots$) would be a good variable order [14]. In such a case, the variable order becomes the same as the case of QMDDs. In this paper, we also consider such a variable order.

As the conventional decision diagrams, we can implement any operations (such as addition and multiplication) between two QuIDDs based on *Apply* operations and *operation and node hash tables*. Usually, QuIDDs can reduce the necessary memory, and the necessary computational time for matrix operations for quantum circuit simulations [15].

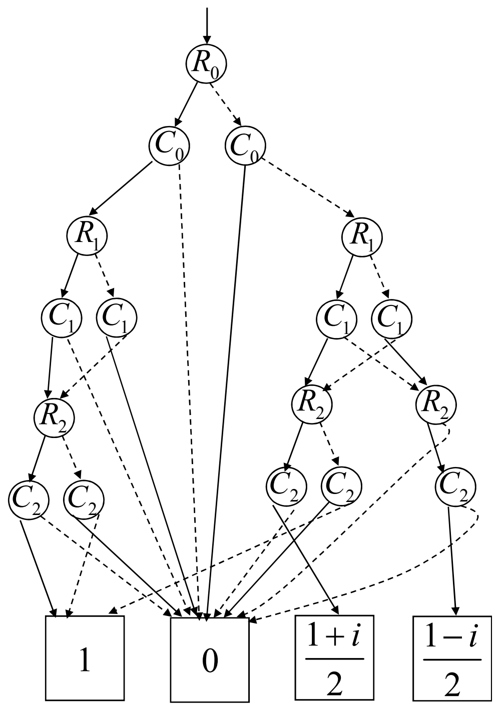

Figure 12: A QuIDD (1)

4.2 Advantages of the Proposed Approach

First we compare the number of nodes to represent the functionality of a single gate between DDMFs and QuIDDs. Let us use the last gate in Fig. 4 for our explanation. As explained before, the QuIDD shown in Fig. 12 represents the matrix corresponding to the gate. A DDMF for the same purpose can be shown as in Fig. 13. In the DDMF, V is attached on the path corresponding to the variable assignment $x_1 = 0, x_3 = 1$. This is because the gate applies V only when $x_1 =$

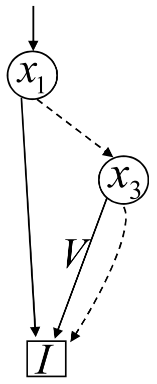

Figure 13: A DDMF

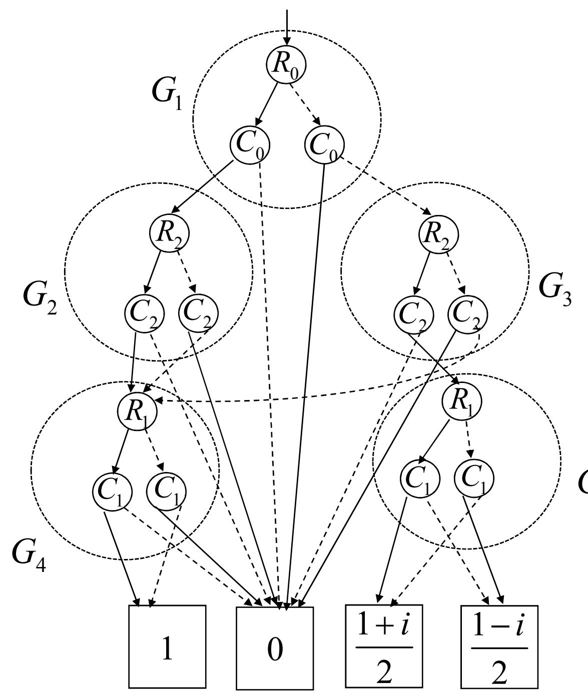

Figure 14: A QuIDD (2)

$0, x_3 = 1$.

From the two figures, DDMFs seem to be better than QuIDDs. However, this is a bit unfair comparison because the DDMF implicitly utilizes the fact that x_2 is the target bit. (Thus, x_2 does not appear in the DDMF.) On the other hand, the QuIDD does not use such knowledge. Although the explanation is omitted due to the space limitation (again see the details in a standard text), (R_1, C_1) corresponds to the input (and the output) line on x_2 in the original quantum circuit. Thus, if we know that x_2 is the target bit, we can also choose the appropriate variable order for the QuIDD such that the pairs of variables (R_1, C_1) are put on the bottom. Then, the QuIDD becomes smaller as shown in Fig. 14.

Although there still seems to be a big difference between two diagrams in Figs. 13 and 14, the essential difference is only a constant factor since we can decrease the number of nodes of the QuIDD in Fig. 14 if we consider the following two issues.

(1) If we choose the appropriate variable ordering of (R_i, C_i) as described above, the corresponding matrix can be considered as one such that there are only 2x2 matrices on the diagonal. (Again we omit the explanation.) Thus, for each group of nodes, (R_i, C_i), which is not on the bottom (e.g., G_1, G_2, G_3 in Fig. 14), the two paths corresponding to $(R_i = 1, C_i = 0)$ and $(R_i = 0, C_i = 1)$ always go to 0 terminal node. (This is because the matrix is a diagonal matrix, and thus the elements in the right upper and the left lower parts of the matrix are all 0.) Thus, we can essentially omit such paths, and then only two paths are essentially necessary for the group of nodes, which means that we can replace each group of nodes by one node. (Of course, to do so, we need more operations than the standard QuIDD operations.)

(2) If we note that the terminal nodes of a DDMF are 2x2 matrices (which has 4 elements), the last group of nodes (G_4 and G_5 in Fig. 14) should not be counted for the fair comparison.

To sum up, in this example, G_2, G_4 and G_5 should not be counted since each group leads to only the elements of a single 2x2 matrix (I or V), and G_1 and G_3 should be considered as one node; thus there is no essential difference between two diagrams. Of course, it is apparent that DDMFs are much more straight forward and easy to implement (hence should be faster) than QuIDD based approaches for our purpose. However, the above discussion makes it clear that there is only a constant factor difference.

Nevertheless, there is a good reason for us to introduce DDMFs: If we consider the verification of the two quantum circuits, the difference may become *exponential* in some cases, which will be explained below.

As mentioned in Sec. 3, when we construct D_i^j from the previous step, we always implicitly choose the appropriate variable order: We implicitly put the target bit (the i-th bit) on the bottom (more precisely, we ignore the target bit) when we calculate D_i^j. On the other hand, we cannot choose an appropriate variable ordering for the QuIDD approach since the verification by QuIDDs are performed as follows: We can verify the equality of the two quantum circuits by comparing the two QuIDDs representing the two quantum circuits. To construct the QuIDD for a circuit, we simply multiply matrices corresponding to gates in the circuit from the left to the right. This can be done by representing each matrix into a QuIDD. For the first gate, we can chose the appropriate variable order. However, if the target bits are different between the first and the second gates, we cannot choose the appropriate variable ordering when we construct the QuIDD for the second gate. This is because the same variable order should be applied for the two QuIDDs when we perform the multiplication. Thus, at least one of QuIDDs may become much larger compared to the DDMF approach. Another important observation is that the resultant QuIDD after the multiplication may be larger than the corresponding DDMF approach by the following reason. We construct each DDMF for each qubit, and thus we can implicitly choose the best variable order (i.e., putting the target bit to the bottom) for each qubit. This can be done because of the restriction of SCQCs. On the other hand, unitary matrix based approaches (such as QuIDD and QMDD based verifications) do not assume such a restriction, and thus they do not treat each qubit separately, and thus they represent the functionalities for all qubits at the same time as one unitary matrix corresponding to a part of quantum circuit. Therefore, we cannot choose the best variable if the appropriate variable order differs for different qubits. (This occurs when we multiply several matrices corresponding to quantum gates with different target bits.) Thus, in the worst case, QuIDDs becomes much bigger than DDMFs during the verification procedures. It is obvious that the necessary memory and the necessary time for *Apply* operations become less if the number of nodes is few. Thus it is apparent our approach is much more efficient than previous approaches for the purpose of the verification of SCQCs.

It should be noted that there is also an apparent advantage of DDMFs in terms of *operation and node hash tables* as follows. The variables for DDMFs during the verification is always the inputs of the circuits (i.e., x_1 to x_n). In other words, we always represent matrix functions with respect to *the inputs of the circuits.* Thus we are always working on the *the input variables of the circuits*. On the other hands, the variables for QuIDDs differs depending on the gates. More precisely, a unitary matrix corresponding to a gate (or a part of the circuit) represents a relation between the inputs of the gate (or the part of the circuit) and the outputs of that. That is, the variables for QuIDD are always *local*: The meaning of variables R_i and C_i for a QuIDD changes during the matrix multiplication. (Even though we work on the same variables R_i and C_i, the logical meaning of the variables changes depending on the corresponding quantum gates.) This apparently makes it difficult to share the previously computed results in the hash tables. Thus, it is apparent that hash tables work much better for DDMFs.

Table 5: Experimental result

♯ variables	♯ gates	♯ nodes	time (sec.)
30	100	418	0.0050
30	200	2509	0.035
30	400	16681	16.23
60	100	1568	0.017
60	200	12984	10.7664
60	400	24681	99.58

The above discussion reveals why our verification method based on DDMFs should work more efficiently than the previous approaches.

We have implemented a DDMF library by C++, and performed a preliminary experiment. Unfortunately, we did not have large SCQC benchmarks, and thus we randomly generated SCQCs and constructed DDMFs for the generated circuits. Then the average (of 10 trials) of the total number of used nodes and the CPU time (on a Linux system running at 3.0 GHz with 256 MB memory) for various settings (i.e., the numbers of inputs and the gates) are reported in Table 5. From the table, we can expect our verification method should work for quantum circuits of practical size.

5 Conclusions and Future Work

In this paper, we introduced new concepts: SCQCs together with DDMFs. As described, they should be useful for the analysis and the verification of quantum circuits with a practical restriction. It should be noted that DDMFs are provably useful even for quantum circuit design methods since DDMFs can be considered as a generalization of the data structure used in the design method in [1].

We also revealed the essential difference between DDMFs and QuIDDs for representing the functionalities of SCQCs. From our discussion, we can say that our approach is much more efficient for the verification of SCQCs than a method based on known techniques. Note that this does not mean that DDMFs are better than QuIDDs: DDMFs are only for SCQCs, whereas QuIDDs can treat all kinds of quantum circuits. In other words, in some sense, our approach stands in the middle of classical Boolean function (BDDs) and general quantum circuit specification (QuIDDs or QMDDs). As described, this standpoint can be considered as a good trade-off point if we consider to design and analyze quantum circuits from the practical view point, i.e., when we focus on sub-circuits to calculate Boolean functions for quantum algorithms. Lastly we would like to add one more issue: Since DDMFs are edge-valued decision diagrams, it should be much easier to verify *quantum phase-equivalence checking* of SCQCs by DDMFs than the method based on QuIDDs [16]. Although we cannot discuss this point due to the space limitation, such a comparison would also be interesting in future work.

In conclusion, we can expect the introduction of SCQCs and DDMFs would promote researches toward practical and efficient quantum circuit design methodology.

References

[1] A. Abdollahi and M. Pedram. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Proc. of the Design, Automation and Test in Europe (DATE'06)*, pages 317–322, 2006.

[2] R. E. Bryant. Graph-based algorithm for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):667–691, August 1986.

[3] E. M. Clarke, M. Fujita, and X. Zhao. Multi-terminal binary decision diagrams and hybrid decision diagrams. In Sasao and Fujita, editors, *Representations of Discrete Functions*, pages 93–108. Kluwer Academic Publishers, 1996.

[4] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit, 2004. Available online as http://arxiv.org/abs/quant-ph/0410184.

[5] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Quantum logic synthesis by symbolic reachability analysis. In *Proceedings of the Design Automation Conference*, pages 838–841, June 2004.

[6] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation Rules for Designing CNOT-based Quantum Circuits. In *Proceedings of the Design Automation Conference*, pages 419–424, June 2002.

[7] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli Network Synthesis with Templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817, June 2005.

[8] D. Maslov, C. Young, M. Miller, and G. Dueck. Quantum Circuit Simplification Using Templates. In *Proc. of the Design, Automation and Test in Europe (DATE'05)*, pages 1208–1213, 2005.

[9] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *Proc. of the IEEE International Symposium on Multiple-Valued Logic*, page 30, May 2006.

[10] Mikko Möttönen, Juha J. Vartiainen, Villa Bergholm, and Martti Salomaa. Quantum circuits for general multiqubit gates. *Phys. Rev. Lett.*, 93(13):130502, 2004.

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

[12] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, June 2003. Available online as http://xxx.lanl.gov/abs/quant-ph/0207001.

[13] Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. Synthesis of quantum logic circuits. *IEEE Trans. on Computer-Aided Design*, 25(6):1000–1010, 2006.

[14] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Improving gate-levelsimulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

[15] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based Simulation of Quantum Computation in the Density Matrix Representation. *Jour-*

nal of Quantum Information and Computation, 5(2):113–130, February 2005.

[16] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Equivalence checking of quantum circuits and states. In *Proceedings of the International Conference on Computer Aided Design*, pages 69–74, 2007.

*

A A Unitary Matrix for a 3-input Gate

As mentioned in Sec. 2.1, one qubit state can be described as a 2-dimensional vector. If we consider the quantum state of n qubits, the state can be described as a 2^n-dimensional vector. For example, if we consider three qubits, $|x_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$, $|x_2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$ and $|x_3\rangle = \alpha_3|0\rangle + \beta_3|1\rangle$, at the same time, the total states can be described as a $8(=2^3)$-dimensional vector $\begin{pmatrix}\alpha_1\alpha_2\alpha_3\\ \alpha_1\alpha_2\beta_3\\ \alpha_1\beta_2\alpha_3\\ \alpha_1\beta_2\beta_3\\ \beta_1\alpha_2\alpha_3\\ \beta_1\alpha_2\beta_3\\ \beta_1\beta_2\alpha_3\\ \beta_1\beta_2\beta_3\end{pmatrix}$. For example, the first element $\alpha_1\alpha_2\alpha_3$ and the second element $\alpha_1\alpha_2\beta_3$ correspond to the basis state $(|x_1\rangle|x_2\rangle|x_3\rangle =)|0\rangle|0\rangle|0\rangle$ and $|0\rangle|0\rangle|1\rangle$, respectively. This vector can be obtained from the tensor product of the three 2-dimensional vectors corresponding to $|x_1\rangle, |x_2\rangle$ and $|x_3\rangle$. See more details in a standard text such as [11].

Thus, an operation to n qubits at the same time can be described by a $2^n \times 2^n$ unitary matrix. For example, the matrix in Fig. 11 shows the unitary matrix that describes the operation by the last gate in Fig. 4. The third and the forth rows (and columns) correspond to $|0\rangle|1\rangle|0\rangle$ and $|0\rangle|1\rangle|1\rangle$, respectively, and thus we can see that this operation output the state $(\frac{1+i}{2}|0\rangle|1\rangle|0\rangle + \frac{1-i}{2}|0\rangle|1\rangle|1\rangle)$ if the input state is $(|0\rangle|1\rangle|0\rangle)$. This corresponds to the fact that the controlled-V gate applies V to $|x_3\rangle$ (i.e., $|x_3\rangle = |0\rangle$ is changed to $V|0\rangle = (\frac{1+i}{2}|0\rangle + \frac{1-i}{2}|1\rangle)$) when $|x_1\rangle|x_2\rangle = |0\rangle|1\rangle$.