An Algorithm for Synthesis of Quantum Reversible Logic Circuits Based on Decomposition

Li Dan, Xu Fanjiang, Zhao Junsuo, and Zhang Wenjun

***Abstract*—Until today, we have not found a general and effective algorithm for synthesis, especially to multi-variables quantum circuit. This is an important question needed to be resolved, because it can not only reduce the cost of manufacture quantum circuit, but also optimize many quantum algorithms. For a quantum circuit which contains *N* quantum bits, the latitude of the matrix is 2^n to realize its function. If we directly design the circuits of the matrix, the workload is huge. Want to reduce the dimensions of the matrix, but also to ensure the unitary of matrix after decomposing, so it is need to use Kronecker product.**

***Index Terms*—Decompose, unitary, kronecker product.**

I. Introduction

Quantum reversible logic circuits is an important component of quantum computer, and it consists of quantum bits, quantum logic gates and quantum lines in order to implement functions such as storage, writing ,reading and logic operation of quantum computer. Two main compositions of quantum circuits are quantum bits and quantum gates which are also called arithmetic operators. The values of Quantum bits got from measure. Multi-bit and multi-gate is expressed by kronecker. Quantum mechanics hypothesis provides a fully quantum mechanical properties of a collection, and quantum circuits synthesis use these properties to design the logic.

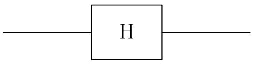
Reversible logic finds many applications, especially in the area of quantum computing. A completely specified n-input, n-output Boolean function is called reversible if it maps each input assignment to a unique output assignment and vice versa. Logic synthesis for reversible functions differs substantially from traditional logic synthesis and is currently an active area of research.

Nowadays, many kinds of reversible quantum gates have been proposed, for example, CNOT gate [1], Toffoli gate [2], and Fredkin gate [3]. How to automatically construct the quantum circuit with small cost using given quantum gates? Several approaches for reversible logic circuit synthesis have been presented. Shende [4] and Song [5] *et al.* present some algorithm of reversible logic synthesis. Shende [6] *et al.* present 3 variables synthesis algorithms. Iwama [7] *et al.* present transformation rules for CNOT-based circuits. Miller [8] and Maslov [9] *et al.* give a synthesis method based on truth table, and use template technology to simplify the circuit. Until today, we have not found a general and effective algorithm for synthesis, especially to multi-variables quantum circuit. This is an important question needed to be resolved, because it can not only reduce the cost of manufacture quantum circuit, but also optimize many quantum algorithms [10].

II. Quantum Gates

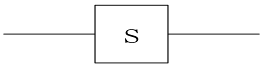
A gate of fundamental importance is the Hadamard gate. It puts a single q-bit into superposition. If a q-bit has been read, then it has collapsed to either |0> or |1>. If such a q-bit is passed through a Hadamard gate, then both possibilities become equally likely if a read is then performed. In unitary matrix form, the non-projective Hadamard gate is:

$$h = \frac{1}{\sqrt{2}}\begin{bmatrix}1\,1\\1\text{-}1\end{bmatrix}$$



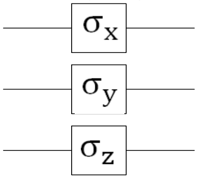
The Phase gate sends |1> to *i* |1> and fixes |0>.

$$\mathrm{ph} = \begin{bmatrix}1\,0\\0\,\mathrm{i}\end{bmatrix}$$



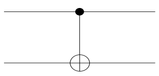
The Pauli gates are spin operators discovered by Wolfgang Pauli. They can be represented by the following matrices:

$$\sigma_x = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} \quad \sigma_y = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix} \quad \sigma_z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$$



The classical gates CNot and Toffoli can also be represented in the quantum setting and each has a related gate for wire diagrams. CNot is a 2 *q*-bit gate similar to the 3 *q*-bit gate Toffoli. To perform a not on the second bit if and only if the first is set reduces to permuting |10> and |11>.

$$c_1^0 = \begin{bmatrix}1000\\0100\\0001\\0010\end{bmatrix}$$

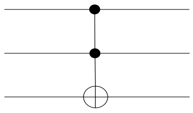


This wire diagram shows that the upper q-bit is the control and the lower bit is the target. The Toffoli gate is drawn similarly, since it has two control *q*-bits acting on one target *q*-bit [11].

Manuscript received July 9, 2013; revised December 3, 2013.

The authors are with the Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China (e-mail: lidan@iscas.ac.cn, fanjiang@iscas.ac.cn, junsuo@iscas.ac.cn, wenjun@iscas.ac.cn).

DOI: 10.7763/IJMLC.2014.V4.378



III. Universal Quantum Gates

A group of gates is called universal for quantum computing is because with this set of gates can compose a quantum circuit which can approximate arbitrary unitary operation with any precision. Toffoli gate can construct a complete set of Boolean join words because it can achieve the classic function of NAND gates. In fact, because Toffoli gate is universal for classical computation, so the quantum circuit includes classic circuit [12].

Although Toffoli gate is a universal gate of combinational circuit, they can't achieve arbitrary quantum state transformation. Therefore we need add single-bit revolving gate. It is already proved that CNOT and single-bit quantum gate is a universal set of quantum gates.

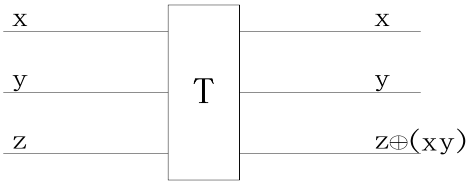


Fig. 1. Toffoli gate.

Depending on the input Toffoli gate Fig. 1 can perform several different basic logic gate operations:

1) If the input z = 0, the output is xy, perform the AND operation of x and y.
2) If the input z=0, y=1, the output is copy(x).
3) If the input x=y=1, the output is NOT(z).
4) If the input y=1, the output is zx, perform the CONT of x and y.

IV. Main Research Aspects of the Quantum Circuits

Up to now, there are two aspects with regard to the quantum circuits: one is the synthesis of quantum circuits, which use given inputs, outputs of a reversible logic function, which expresses quantum circuits functions, and use given quantum gates, according to given quantum cost standard, find the smallest or smaller quantum cost to product quantum circuits automatically. The other is building quantum circuit simulator, in order to simulate the process of quantum computing and quantum algorithm [12].

For a quantum circuit which contains N quantum bits, the latitude of the matrix is 2^n to realize its function. If we directly design the circuits of the matrix, the workload is huge. Even if use genetic algorithm, because the length of chromosome which is used to represent quantum gate sets is longer, it will inevitably reduce the effectiveness of the genetic algorithm, so within a limited time in a limited space it cannot get a final solution.

It is known that, in quantum compute, either individually basic quantum gate or combination of basic quantum gates must satisfy a basic condition that the matrix they represent must meet the unitary. Want to reduce the dimensions of the matrix, but also to ensure the unitary of matrix after decomposing, so it is need to split into Kronecker product of unitary matrix.

V. The Definition of Kronecker Product

The Kronecker product of a matrix

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

and a matrix

$$B=\begin{bmatrix} b_{11} \cdots a_{1p} \\ \vdots \ \ddots \ \vdots \\ b_{q1} \ldots a_{pq} \end{bmatrix} \text{ is } A\otimes B=\begin{bmatrix} a_{11}B \cdots a_{1n}B \\ \vdots \ \ \ddots \ \ \vdots \\ a_{m1}B \ldots a_{mn}B \end{bmatrix}$$

It is computed by multiplying the matrix B with each of the components of the matrix A. The resulting block matrix is returned as a matrix of larger dimension. The new matrix is a pm×qm matrix. For example,

$$A=\begin{bmatrix} a_{11}\ a_{12} \\ \\ a_{21}\ a_{22} \end{bmatrix}, \quad B=\begin{bmatrix} b_{11}\ b_{12} \\ \\ b_{21}\ b_{22} \end{bmatrix}$$

$$A\otimes B=\begin{bmatrix} a_{11}B\ a_{12}B \\ \\ a_{21}B\ a_{22}B \end{bmatrix}=\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

VI. Decomposition of Kronecker Product

Theorem 1. Suppose A is a nonzero matrix, and its dimension is m×n(both m and n are prime number). rank(A) $\neq$1.Then A can't be decomposed into the kronecker product of two matrixes whose line or column number is greater than 1.

Corollary 1. Suppose A is an n order square matrix (N is prime number). rank(A)$\neq$1.Then A can't be decomposed into the kronecker product of two square matrixes whose dimension is greater than 1.

Theorem 2. The rank of the invertible matrix of is equal to the order of the matrix.

Definition 1. Assume a matrix A=$(a_{ij})_{m\times n}$, arrange each line of A crosswise to get mn dimension row vector, it is called row-expansion, and it is marked rs(A). Arrange each column of A crosswise to get mn dimension column vector, it is called column-expansion, and it is marked cs(A).

$$\text{rs}(A)=\left(a_{11,}a_{12,}\ldots,a_{1n,}a_{21,}a_{22,}\ldots a_{2n,}a_{m1,}\ldots a_{mn}\right)$$

$$\text{cs}(A)=\left(a_{11,}a_{12,}\ldots,a_{1n,}a_{21,}a_{22,}\ldots a_{2n,}a_{m1,}\ldots a_{mn}\right)^{T}$$

Obviously, for matrix A, B, exist A=kB⇔rs(A)=k rs(B).

Theorem 3. If a matrix A=$(a_{ij})_{m\times n}\neq 0$, m=sp, n=tq, then A can be decomposed into the kronecker product of a s×t matrix B and a p×q matrix C ⇔ In the block matrixes of

$A=(A_{ij})_{s\times t}$, rank $\{\mathrm{rs}(A_{11}),\ldots,\mathrm{rs}(A_{1t}),\ldots,\mathrm{rs}(A_{s1}),\ldots,\mathrm{rs}(A_{st})\}=1$, and A_{ij} is a $p\times q$ matrix (i=1, 2,…, s; j=1, 2,…, t). If $K=(\mathrm{rs}(A_{11})^T,\ldots,\mathrm{rs}(A_{1t})^T,\ldots,\mathrm{rs}(A_{s1})^T,\ldots,\mathrm{rs}(A_{st})^T)$, then K is a $(pq)\times(st)$ matrix[13], [14].

VII. The Step of Decomposition

1) Get the dimension of a unitary matrix $A=(a_{ij})_{n\times n}$. If the dimension n is a prime number, According to Theorem 2 and Corollary 1 we can know that the unitary matrix A cannot be decomposed into a kronecker product of two square matrixes whose order is greater than 1. Because the dimension of a matrix of a quantum circuit which contain more than or equal to two-qubit quantum bits is $n=2^m$ (m=2, 3, 4…), the dimension of this matrix A is not a prime number for sure.
2) Get the factors of n, the factor is a_m $(a_m\neq 1$ and $a_m\neq n)$. For example, if n=8, a_1=2, a_2=4.
3) Divide matrix $A=(a_{ij})_{n\times n}$ into $b_m=(n\times n)/(a_\mathrm{m}\times a_\mathrm{m})$ sub-block matrix A_{ij} and the dimension of A_{ij} is a_m
4) Make the matrix $K=(\mathrm{rs}(A_{11})^T,\ldots,\mathrm{rs}(A_{1b_m})^T,\ldots,\mathrm{rs}(A_{b_m1})^T,\ldots,\mathrm{rs}(A_{b_mb_m})^T)$
5) Do the elementary transformation of matrix K. If K can transform into

$$D=\begin{bmatrix} d_{11}\ldots d_{1b_m}\ldots d_{b_m1}\ldots d_{b_mb_m} \\ 0\ \ldots\ 0\ \ldots\ 0\ \ldots\ 0 \\ \ldots\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\ 0\ \ldots\ 0\ \ldots\ 0\ \ldots\ 0 \end{bmatrix}$$

in which $(d_{11},\ldots,d_{1b_m},\ldots,d_{b_m1},\ldots,d_{b_mb_m})\neq 0$, matrix can be decomposed into the kronecker product of two matrixes, otherwise can't.

6) If $d_{11}\neq 0$, D can transform into

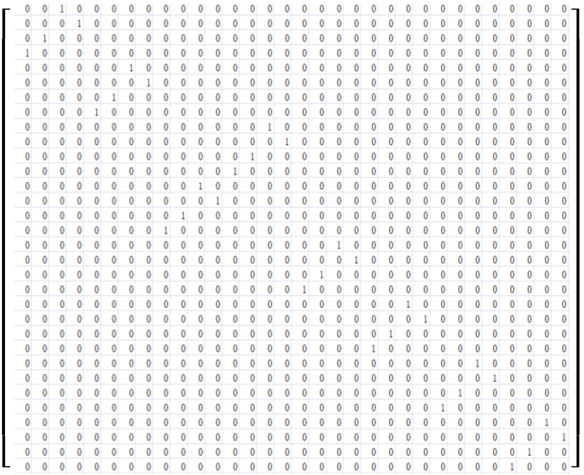
$$T=\begin{bmatrix} 1\ldots & \dfrac{d_{1b_m}}{d_{11}}\ldots & \dfrac{d_{b_m1}}{d_{11}}\ldots & \dfrac{d_{b_mb_m}}{d_{11}} \\ 0\ldots & 0\ \ldots & 0\ \ldots & 0 \\ \ldots\ldots & \ldots & \ldots & \ldots \\ 0\ldots & \ldots\ 0\ \ldots & 0\ \ldots & 0 \end{bmatrix}$$

by further elementary transformation.

Then rs $(A_{ij})^T=\dfrac{d_{1j}}{d_{11}}$ = rs $(A_{11})^T$ and $A_{ij}=\dfrac{d_{ij}}{d_{11}}A_{11}$, $(1\le i\le b_m, 1\le j\le b_m)$. Write the kronecker product of matrix $A=(A_{ij})_{b_m\times b_m}=(\dfrac{d_{ij}}{d_{11}})_{b_m\times b_m}\otimes A_{11}$.

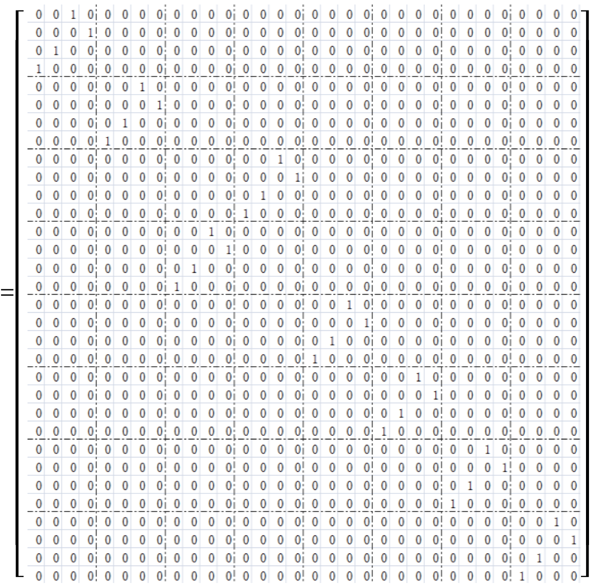
VIII. Generate Quantum Circuits

There is a matrix $A=(a_{ij})_{32\times 32}$ as shown below.



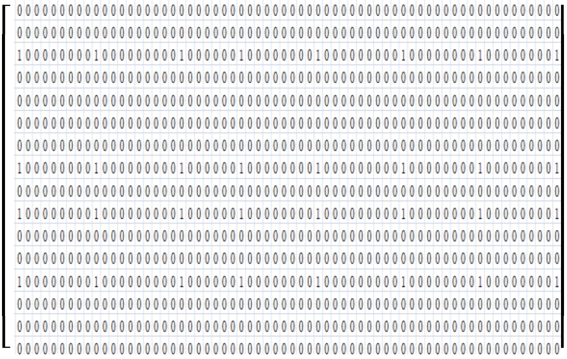
$$A=\left[\begin{array}{*{32}{c}}
0&0&1&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
0&0&0&1 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
0&1&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
1&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&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&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 \\
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&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&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&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 & 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&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&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&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 & 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&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&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&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 & 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&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&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&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 & 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&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&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&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 & 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&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&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&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 & 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&0&0 & 0&0&0&0 & 1&0&0&0
\end{array}\right]$$

Divide matrix A into (32×32)/(4×4)=8×8 sub-block matrixes.

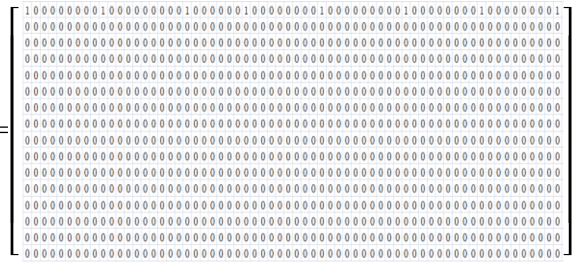


$$A=\left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc}
0&0&1&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
0&0&0&1 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
0&1&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\
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&0&0&0 & 0&0&0&0 \\ \hline
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&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 \\
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&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&0&0&0 \\ \hline
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&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 & 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&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 \\ \hline
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&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 & 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&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 \\ \hline
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&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 & 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&0&0 & 0&0&0&0 & 1&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&0 \\ \hline
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&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 & 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&0&0 & 0&0&0&0 & 1&0&0&0 & 0&0&0&0 & 0&0&0&0 \\ \hline
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&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&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 & 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&0&0 & 0&0&0&0 & 1&0&0&0 & 0&0&0&0 \\ \hline
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&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&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 & 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&0&0 & 0&0&0&0 & 1&0&0&0
\end{array}\right]$$

Make the matrix K, and does the elementary transformation of K, then K will transform into:



$$K=\left[\begin{array}{*{64}{c}}
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&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&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&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&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&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&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 \\
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&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&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&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&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 \\
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&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&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&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 \\
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&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&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&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 \\
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&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&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&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&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&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 \\
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&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&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&0&0&0&0&0&0&0&0
\end{array}\right]$$



$$\rightarrow K=\left[\begin{array}{*{64}{c}}
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&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&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&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 \\
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&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&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&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&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&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&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&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 \\
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&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&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&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&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&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&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&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 \\
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&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&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&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&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&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&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{array}\right]$$

We can get the result

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

Finally, we can get the quantum circuit as Fig. 2.

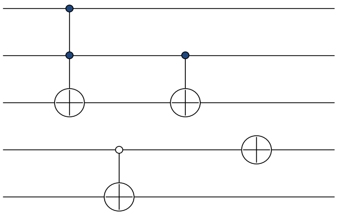


Fig. 2. The quantum circuit.

IX. Conclusion

From the example above, we can draw the conclusion that, by using the kronecker product, we can not only decompose high dimension matrix in order to reduce the complexity of quantum reversible logic circuits synthesis, but also get a much simpler circuit by using this method than get the quantum circuit synthesize directly. Furthermore, the quantum circuit which gets from this method provides better precondition for deducing count of execution time unit.

References

[1] R. Feynman, "Quantum mechanical computers," *Optic News*, 1985, pp. 11-20.

[2] T. Toffoli, J. W. de Bakker, and V. L. Jeds, "Reversible computing," *Automata, Languages and Programming*, New York, Springer, 1980.

[3] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219-253, 1982.

[4] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible logic circuit synthesis," in *Proc. the International Conference on Computer-Aided Design*, California, 2002, pp. 125-132.

[5] X. Y. Song, G. W. Yang, M. Perkowski, and Y. Wang, "Algebraic characteristics of reversible gates," *Theory of Computing Sys-tems*, vol. 39, no. 2, pp. 311-319, 2006.

[6] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Transactions on Circuits and Systems-I*, vol. 22, no. 6, pp. 723-729, 2003.

[7] K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," in *Proc. Design Automation Conference*, New Orleans, vol. 28, no. 4, pp. 419-424, 2002.

[8] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. the International Conference on Computer-Aided Design*, California, 2003, pp. 318-323.

[9] D. Maslov, G. W. Dueck, and D. M. Mille, "Toffoli network synthesis with templates," *IEEE Transactions on Circuits and Systems-I*, vol. 24, no. 6, pp. 807-817, 2005.

[10] Z. Q. Li, H. W. Chen, B. W. Xu, W. Q. Li, J. J. Wang, and W. J. Liu, *A Fast Algorithm for Synthesis of Quantum Reversible Logic Circuits*, 2010.

[11] D. C. Gajewski, *Analysis of Groups Generated by Quantum Gates*, University of Toledo, 2009.

[12] Z. Q. Li, A fast algorithm for quantum circuits.

[13] D. H. Lin, "The approach to get the decomposition of the kronecker product of matrix," 2007.

[14] D. H. Lin, "Decompostion of the kronecker product of matrix," 2006.

Li Dan was born in Liaoning, China in 1983. The author is a researcher in Institute of Software Chinese Academy of Sciences who is interested in the research of quantum computation.