Analysis and Synthesis of Quantum Circuits by Using Quantum Decision Diagrams[1]

Afshin Abdollahi *Massoud Pedram*
Department of Electrical Engineering
University of Southern California
{*afshin, pedram*}*@usc.edu*

Abstract

Quantum information processing technology is in its pioneering stage and no proficient method for synthesizing quantum circuits has been introduced so far. This paper introduces an effective analysis and synthesis framework for quantum logic circuits. The proposed synthesis algorithm and flow can generate a quantum circuit using the most basic quantum operators, i.e., the rotation and controlled-rotation primitives. The paper introduces the notion of quantum factored forms and presents a canonical and concise representation of quantum logic circuits in the form of quantum decision diagrams (QDD's), which are amenable to efficient manipulation and optimization including recursive unitary functional bi-decomposition. This paper concludes by presenting the QDD-based algorithm for automatic synthesis of quantum circuits.

1. Introduction

We are beginning to reach the fundamental limits of the materials used in the planar CMOS process [1]. Quantum computers can evolve a superposition of quantum states until the final output is obtained. Such "quantum parallelism" could potentially outstrip power of classical computers [2][3]. Certain problems for which there is no polynomial solution in classical domain can be solved in polynomial time in quantum domain (e.g., the factoring problem). Similarly, the complexity of some other problems (e.g., unstructured search and Boolean satisfiability) can be reduced by transforming them into the quantum domain [4]. Indeed, quantum circuits have the ability to perform massively parallel computations in a single time step [5][6]. Hence quantum computing has become a very attractive research area, which is expected to play an increasingly critical role in building more efficient computers [7][8]. Computer aided design of quantum circuits is at primitive stages, which motivates rigorous research aimed at developing CAD techniques and tools for quantum circuits.

In this paper we address the problem of synthesizing a general quantum operation. Exact definition of the problem is provided clearly at the end of section 2 In this paper a canonical decision diagram based representation of quantum circuits is presented and a CAD methodology and novel techniques for synthesis of quantum logic circuits based on these decision diagrams are described. The remainder of this paper is organized as follows. In section 2 fundamental aspects of quantum mechanics and in section 3 previous work on quantum circuit synthesis are reviewed. In section 4, quantum factored forms, quantum decision diagrams (QDD's), and a QDD-based quantum circuit synthesis are introduced. Conclusions are provided in section 5.

[1] This research is funded in part by the NSF QnTM program under grant no. 0524602.

2. Fundamentals of Quantum Computing

In quantum computation quantum bits (qubits), derived from the states of micro-particles such as photons, electrons or ions are used instead of classical binary bits to represent information. For example, two possible spin rotations of an electron are represented as $[1 \quad 0]^T$ and $[0 \quad 1]^T$, which are the basis states (basis vectors) of this computational quantum system [9][10].

Each particle in a quantum system is represented by a wave function inheriting the powerful concept of superposition of states. For example, the state of a particle p_1 may be represented by a wave function $\Psi_1 = \alpha_1[1 \quad 0]^T + \beta_1[0 \quad 1]^T = [\alpha_1 \quad \beta_1]^T$ where the coefficients α_1 and β_1 are in general complex and $|\alpha_1|^2+|\beta_1|^2=1$. In general, the wave function of a quantum system with n qubits represents an arbitrary superposition of 2^n states while in a classical system n bits represent only 2^n distinct states. Therefore the space of quantum systems is exponentially larger than that of the classical binary systems. Analysis (and by extension, synthesis) of quantum logic circuits is more difficult than that of the digital logic circuits because the former requires manipulation of matrices and bases in Hilbert space whereas the latter requires binary, or at most multi-valued, logic operations. Quantum operators over a set of qubits are modeled as matrix operations. As an example, for a quantum system comprising of a single particle p_1, a quantum operator (gate) is represented by a 2×2 (in general complex) unitary matrix U which transforms state $\Psi_1 = [\alpha_1 \quad \beta_1]^T$ to state $\Psi_2 = U\Psi_1$. Recall that a matrix U is unitary exactly if $UU^+=I$ where U^+ is the hermitian (complex conjugate transpose) of U. Since matrix U is unitary, the inverse of this gate is matrix U^+, which is the inverse of U. An important class of quantum operators is the *rotation operator*. A θ rotation around the X axis in Bloch sphere representation [4] is:

$$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}.$$

The following relation shows that rotation operators around X are commutative with respect to matrix multiplication:

$$R_x(\theta_1)R_x(\theta_2) = R_x(\theta_2)R_x(\theta_1) = R_x(\theta_1+\theta_2).$$

In general for an n-qubit system, a quantum operation is represented by a $2^n \times 2^n$ unitary matrix. An example of a 2-qubit gate is the controlled-U gate depicted in Figure 1. For a 2×2 unitary matrix U, the controlled-U gate works as follows: when the control signal a is $[1 \quad 0]^T$, $q=b$ and when it is $[0 \quad 1]^T$, then $q=Ub$. For both cases, $p=a$.

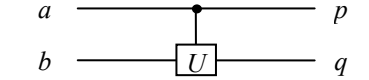

Figure 1. Schematic diagram of a controlled-*U*.

Similar to controlled-U operator, one can easily define a significant class of 2-qubit operators as the *controlled-rotation operator*. Rotation operators are elementary and easily realizable in most implementations of quantum computation [4], e.g., nuclear magnetic resonance and ion trap realizations. These factors are precisely why this paper will focus on rotation and controlled-rotation operators as the elementary building blocks for synthesis of quantum circuits. A new concise and canonical data structure, called quantum decision diagrams or QDD's, will be introduced and subsequently used for conducting quantum operations and synthesizing quantum logic circuits. More precisely, the QDD's are designed to have the ability to express the functionality of every quantum circuit composed of controlled-rotation operators assuming that all rotations are about a single *axis* and a '*binary control signal'* constraint is enforced.

3. Previous Work on Quantum Logic Synthesis

Several approaches for reversible logic circuit synthesis have been presented in [11]-[14]. These approaches resort to exhaustive search or methods such as matrix decomposition, local transformations, spectral approaches, and on adaptations of EXOR logic decomposition, Reed-Muller representations, and other classical combinational circuit design methods. Toffoli [15] provided an algorithm for implementing an arbitrary function with the "CNTS" library, comprising of controlled-NOT, NOT, Toffoli gate, and SWAP gate (see section 4). Kerntopf [16] proposed a search method to perform synthesis of small-scale circuits. In [17] a synthesis method based on manipulating the truth tables is presented. Shende et al. [18] generate a library of small optimal circuits based on branch-and-bound and exploiting the property that any sub-circuit of an optimal circuit is itself optimal. Agrawal and Jha [19] presented a RM-expansion based technique for optimizing a circuit that is mapped to reversible gates. In [20] an algorithm for synthesis of quantum circuits using reversible Davio expansion was proposed. In [21], Shende et al presented a top-down structure using the Cosine-Sine decomposition and introduced and used the quantum multiplexer for recursive implementation of quantum gates. Group theory has also been employed as a tool to analyze reversible gates [22] and investigate generators of the group of reversible gates [23]. In [24], Hung et al transform the synthesis problem into a satisfiability problem. They in fact use a SAT solver instead of employing an exhaustive search. This method is practical only for very small circuits since the reported run-time of the algorithm for optimal synthesis of a single-bit adder with 6 quantum gates is 7 hours on a 850MHz Pentium III processor running Linux. Other researchers have turned to evolutionary algorithms to reduce the CPU time [27]. It can be inferred that developing a practical synthesis algorithm for quantum circuits is extremely difficult because of the fast increase of data sizes. Indeed to-date there are no counterparts in quantum logic of such useful tools as algebraic decomposition, decision diagram based synthesis, or other standard logic synthesis techniques such as reduction to covering/coloring combinational approaches. In this paper we introduce an efficient data structure based on decision diagrams for representation, analysis and synthesis of quantum circuits and provide a synthesis approach based on the proposed decision diagrams.

4. Quantum Logic Synthesis

In this section, we will address the problem of automatically synthesizing a given Boolean function, *f*, by using $R_x(\theta)$ and controlled-$R_x(\theta)$ operators as the elementary operations (gate primitives.) In a synthesized quantum circuit, the quantum states representing binary (basis states) values $\hat{0}$ and $\hat{1}$ will be:

$$\hat{0} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T, \qquad \hat{1} = R_x(\pi)\hat{0} = \begin{bmatrix} 0 & -i \end{bmatrix}^T.$$

With this definition of $\hat{0}$ and $\hat{1}$, the basis states remain orthogonal, and hence, they can be completely distinguished with proper quantum measurements. We adopt this definition because inversion from one basis state to the other is simply obtained by a π rotation around the X axis. With these assignments, the $R_x(\pi)$ operation acts as the quantum NOT gate (since $R_x(\pi)R_x(\pi) = R_x(2\pi) = I$.) Subsequently, the controlled-NOT (CNOT) gate can be described by using the controlled-$R_x(\pi)$ operator (cf. Figure 2(i).) In addition, the Toffoli gate, also known as the 3×3 Feynman gate or the Controlled-Controlled-NOT gate, may be described by using the controlled-$R_x(\pi)$ operator (cf. Figure 2(ii).) Notice that the Boolean functions for each output of the CNOT and Toffoli gates are also shown in this figure, where '$\cdot$' and '$\oplus$' denote binary 'AND' and 'XOR' operators.

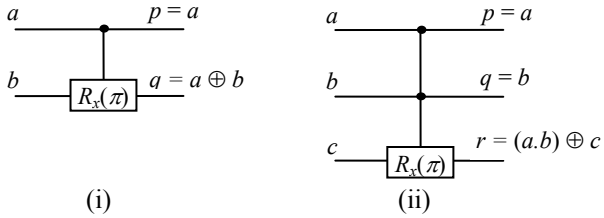

Figure 2. (i) CNOT gate (ii) Toffoli gate.

Toffoli gate can be implemented using controlled-rotation operators as demonstrated in Figure 3. In this figure only the angle of rotation is shown for controlled-rotation operators.

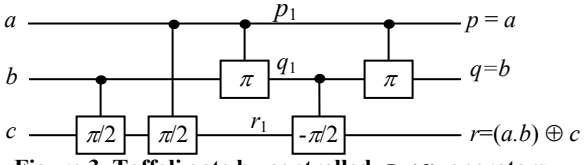

Figure 3. Toffoli gate by controlled-$R_x(\theta)$ operators.

In this paper, we focus on rotation-based quantum gates, which are directly realizable in quantum hardware [28][29]. It is critical to point out that, for all ***input basis* (*binary*) *vectors***, control inputs of the controlled-$R_x(\theta)$ operators in the circuit of Figure 3 only take $\hat{0}$ or $\hat{1}$ values. This condition, which we shall refer to as the ***binary control signal constraint***, is set as a design constraint in the synthesis process. From the viewpoint of representing quantum logic circuits, this constraint does not affect the expressive power of $R_x(\theta)$ and controlled-$R_x(\theta)$ operators. This constraint has also been adopted by other researchers in the field (cf. [24][27].) This constraint does not imply that a control signal cannot adopt a superposition value, i.e., it is possible that a control signal takes a superposition value, which happens exactly when the inputs to the circuit are non-binary. In the reminder of this paper, when we constrain a variable to assume a binary value, we only mean that if binary inputs are applied to the circuit, then constraint will be enforced. Finally, to the best of our knowledge, there is no evidence that relaxing this constraint, can improve the optimality of the synthesis result for quantum circuits.

4.1 Quantum Factored Forms

In any quantum circuit synthesized with binary control signal constraint, the first output, *p,* of any controlled-$R_x(\theta)$ operator is equal to the control input *a*. However, the second output depends on both inputs. We use the notation $q = aR_x(\theta)b$ to describe the second output *q*. With this new notation $R_x(\theta)$ can also be regarded as a two-operand operator with the following functionality: if $a = \hat{0}$, then $q = b$ else $q = R_x(\theta)b$. (The left operand, *a,* only assumes $\hat{0}$ or $\hat{1}$.)

Definition: *Quantum Factored Form*. $\hat{0}$ is a quantum factored form. Every variable is a quantum factored form. If *h* is a factored form, then $f = R_x(\theta)h$ is a quantum factored form. Moreover, if *g* and *h* are factored forms and *g* only takes $\hat{0}$ and $\hat{1}$ values, then $f = gR_x(\theta)h$ is a quantum factored form.

In a quantum circuit synthesized with $R_x(\theta)$ and controlled-$R_x(\theta)$ operators (with binary control signal constraint), any output (or internal signal) of the circuit can be described as a quantum factored form. For example, the output function *r* in Figure 3 can be described as: $r = [aR_x(\pi)b]R_x(-\pi/2)[aR_x(\pi/2)[bR_x(\pi/2)c]]$.

The following two commutative and associative relations are useful for manipulating quantum factored forms: $aR_x(\pi)b = bR_x(\pi)a$, $aR_x(\theta_1)[bR_x(\theta_2)c] = bR_x(\theta_2)[aR_x(\theta_1)c]$.

Cascade form, which is a subclass of factored forms, is defined as follows.

Definition: *Quantum Cascade Form*. $\hat{0}$ and every variable are quantum cascade forms. If *h* is a cascade form and *v* is a variable not present in *h*, then $f = vR_x(\theta)h$ is a quantum cascade form. ($f = R_x(\theta)h$ is also considered a quantum cascade form.)

A general quantum cascade form is expressed as:

$$f = R_x(\theta_0)\left[v_1R_x(\theta_1)\left[v_2R_x(\theta_2)\ldots\left[v_nR_x(\theta_n)\hat{0}\right]\right]\right]$$

Note that if $\theta_n = \pi$ then $v_nR_x(\theta_n)\hat{0} = v_n$. It can be verified that this cascade form expression can be rewritten as:

$$f = R_x(\theta_0)\left[v_{p1}R_x(\theta_{p1})\left[v_{p2}R_x(\theta_{p2})\ldots\left[v_{pn}R_x(\theta_{pn})\hat{0}\right]\right]\right]$$

where $(p_1, p_2, \ldots, p_n)$ is a permutation of $(1,2,\ldots,n)$.

The problem of realizing a function using $R_x(\theta)$ and controlled-$R_x(\theta)$ operators is equivalent to finding a quantum factored form for the function. To do this, we first introduce a graph-based data structure in the form of a decision diagram for representing quantum logic functions.

4.2 Quantum Decision Diagrams (QDD)

The concept of Reduced Ordered BDD (ROBDD) was introduced by Bryant [30], who also proved its canonicity property and also provided a set of operators for manipulating ROBDD's. From now on, we shall simply write BDD to mean ROBDD. Using complement edges can further reduce the size of the BDD [31]. Lai et al. [32] proposed Edge-Valued BDD's (EVBDD), which can represent and manipulate integer functions and can be used for functional decomposition. In this section, we describe a new decision diagram for the representation of quantum functions. In a previous work [25] decision diagrams have been used to describe a quantum circuit. The proposed structure is basically a complete tree with as many leaf nodes as the size of input space. This representation is not more efficient than tabular or any other representation. Another type of decision diagrams has been used (for a fundamentally different purpose of unitary matrix multiplication) in which the decision variables correspond to different rows and columns of matrices[26].

The input and output spaces of an *n* qubit quantum function include arbitrary superpositions of 2^n states (with norm one.) Therefore, an n-qubit function is represented by a $2^n \times 2^n$ unitary matrix i.e., an arbitrary superposition of basis binary inputs may be applied to a quantum circuit. However, based on the concept of superposition and linearity of quantum functions, the behavior of a quantum function (its functionality) can be completely described by using the results of its operation on basis victors. More precisely, given input $\Psi = \sum_{k=0}^{2^n-1}\alpha_k|k\rangle$ (where $|k\rangle$'s are the basis vector) and function U, the output can be calculated as $U\Psi = U\sum_{k=0}^{2^n-1}\alpha_k|k\rangle = \sum_{k=0}^{2^n-1}\alpha_kU|k\rangle = \sum_{k=0}^{2^n-1}\alpha_kU_k$ where U_k is equal to $U|k\rangle$. Each basis vector $|k\rangle$ (k = 0, 1, …, 2^n-1) represents a point in the *n*-dimensional Boolean space. Hence knowing the effect of a quantum function on all possible binary combinations (*n*-vectors of $\hat{0}$ and $\hat{1}$) can completely specify the functionality of the quantum circuit. ***Quantum Decision Diagrams*** are thereby introduced next in order to represent the value of quantum functions for binary basis vectors, which bear enough information to compute the functions for arbitrary quantum input states.

Definition: A QDD is a directed acyclic graph with three types of nodes: a single terminal node with value $\hat{0}$, a weighted root node, and a set of non-terminal (internal) nodes. Each internal node represents a quantum function. It is associated with a binary decision variable and has two outgoing edges: a weighted $\hat{1}$-edge (solid line) leading to another node (the $\hat{1}$-child) and a non-weighted $\hat{0}$-edge (dashed line) leading to another node (the $\hat{0}$-child.) The weights of the root node and $\hat{1}$-edges are in the form of $R_x(\theta)$ matrices. Since all the weights in a QDD are in the form of $R_x(\theta)$, the value θ is sufficient to specify the weight. We assume that $-\pi < \theta \leq \pi$. Furthermore, when the edge or root node weight is the identity matrix (i.e., $R_x(0) = I$), it will not be shown in the diagram.

Figure 4(i) shows an internal node, *f*, in a QDD with decision variable, *a*, the corresponding $\hat{0}$ and $\hat{1}$ edges, and child nodes, f_0 and f_1. This relation between the QDD nodes in this figure is as follows. If $a = \hat{1}$, then $f = R_x(\theta)f_1$ else $f = f_0$. In addition, if *f* is the weighted root node of a QDD (cf. Figure 4(ii)), then the following relation holds. If $a = \hat{1}$, then $f = R_x(\theta_r)R_x(\theta)f_1 = R_x(\theta_r + \theta)f_1$ else $f = R_x(\theta_r)f_0$.

Similar to BDD's, in QDD's isomorphic sub-graphs (nodes with the same quantum function) are merged. Additionally, if the $\hat{0}$-child and the $\hat{1}$-child of a node are the same and the weight of the $\hat{1}$-edge is $R_x(0) = I$, then that node is eliminated. Using these two reduction rules and given a total ordering on input variables, the QDD will be uniquely constructed for a quantum function.

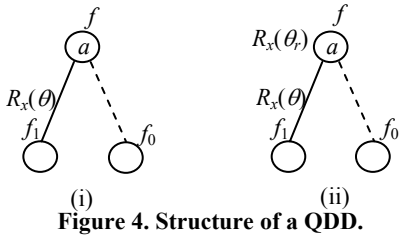

Figure 4. Structure of a QDD.

Consider a quantum function with n variables $f(v_1, v_2, ..., v_n)$. Each binary value assignment to the variables $v_1, v_2, ..., v_n$ corresponds to a path from the root to the terminal node of the QDD of f. Assuming the variable ordering $v_1<v_2<...<v_n$, the corresponding path can be identified by a top-down traversal of the QDD starting from the root node. For each node that is visited during the traversal, we select the edge corresponding to the value of its decision variable v_i. (i.e., if $v_i=\hat{1}$ select the $\hat{1}$-edge; otherwise, select the $\hat{0}$-edge) and continue with the node at the end of the selected edge until the terminal node is visited. During such a traversal for every variable v_i, only one node with decision variable v_i will be visited specifying a path from the root to the terminal node with a total number of n-1 edges. Let's denote the weight of the root node by w_0 and the weight of the selected edges by $w_1, w_2, ..., w_{n-1}$. The value of the function f for assigned values to $v_1, v_2, ..., v_n$ is: $f(v_1, v_2, ..., v_n) = w_0 w_1 ... w_{n-1} \hat{0}$. Clearly, if, during this graph traversal, a $\hat{0}$-edge is selected for variable v_i (i.e., if $v_i=\hat{0}$), then the corresponding edge weight will be $w_i=I$. We have shown that QDD's provide a concise and canonical representation for a quantum function. QDD's can be regarded as an extension of BDD's i.e., each BDD can also be regarded as a QDD (A QDD is a BDD exactly if all the weights of the QDD are either $R_x(0) = I$ or $R_x(\pi)$.) As will be shown later, the synthesis process starts with the QDD of the given logic function and decomposes the given QDD to realizable QDD's. The QDD structure has some useful properties. One important property, i.e., the *linear topology property*, is demonstrated in Figure 5. The idea is that when the $\hat{0}$-child and the $\hat{1}$-child of a node, f, are the same node, g, then that node can be directly realized by a controlled-$R_x(\theta)$ operator in terms of its child i.e., $f = aR_x(\theta)g$. As an example, Figure 5 shows the QDD's of functions q_1 and r_1 in Figure 3. The QDD's in Figure 5 are associated with functions that have a quantum cascade form representation. For example function r_1 can be represented as: $r_1 = aR_x(\pi/2)[bR_x(\pi/2)c]$ which is a cascade form.

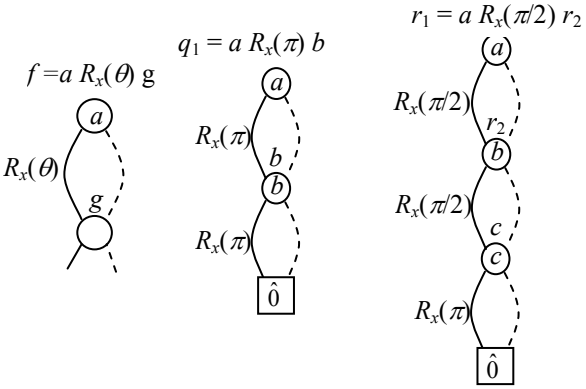

Figure 5. The linear topology property of a QDD.

Generally every QDD with a chain structure (such as QDD's in Figure 5) is associated with a cascade form and can directly be realized with the rotation and controlled-rotation operators. This property will extensively be used in the synthesis algorithm.

It is important to develop a method for applying rotation and controlled-rotation operators to QDD's. Suppose the QDD for a function, f, is given. The QDD for $h = R_x(\gamma)f$ can simply be obtained by multiplying the weight of the root node of f by $R_x(\gamma)$. To obtain $h = fR_x(\gamma)g$ for given QDD's f and g (assuming f only takes $\hat{0}$ and $\hat{1}$ values,) we use the *quantum apply operation* (q-apply), the details omitted here due to space limitation.

Given other representations for a function, the QDD can be obtained by first creating the complete binary decision tree for the function and repeatedly applying the following steps:

a) Convert the weight of $\hat{0}$-edge to I (by changing the weight of corresponding $\hat{1}$-edge and edges ending at the parent node.)
b) Merge isomorphic sub-graphs.

4.3 QDD-based Functional Decomposition

As mentioned earlier, the problem of realizing a function, f, using $R_x(\theta)$ and controlled-$R_x(\theta)$ operators is equivalent to finding a quantum factored form for the function, which can in turn be performed by recursive bi-decomposition of the given function f.

Definition: Quantum (unitary) functional bi-decomposition of f is defined as finding functions g and h and value γ such that $f = gR_x(\gamma)h$ where function g only assumes values $\hat{0}$ and $\hat{1}$.

Next we provide an algorithm for quantum unitary bi-decomposition which can be used to bi-decompose a given function f to $gR_x(\gamma)h$. Subsequently, g and h are recursively bi-decomposed, which will eventually result in a quantum factored for f. The bi-decomposition algorithm is based on the notion of *quantum linear (q-linear)* variables. In the reminder of this paper, while expressing a function as $f(v_1, v_2, ..., v_n)$, it is implicitly assumed that f depends on all variables $v_1, v_2, ..., v_n$ (i.e., f depends on all of these variables.)

Definition: For a given function $f(v_1, v_2, ..., v_{i-1}, v_i, v_{i+1}, ..., v_n)$, variable v_i is '*q-linear*' if there exists a rotation value, θ_i, such that for every value assignment to $v_1, v_2, ..., v_{i-1}, v_{i+1}, ..., v_n$: $f_{v_i} = R_x(\theta_i) f_{\bar{v}_i}$, where $f_{v_i} = f(v_1, v_2, ..., v_{i-1}, \hat{1}, v_{i+1}, ..., v_n)$ and $f_{\bar{v}_i} = f(v_1, v_2, ..., v_{i-1}, \hat{0}, v_{i+1}, ..., v_n)$.

A variable is called *q-nonlinear* exactly if it is not q-linear.

Lemma 1: Consider function $f(v_1, v_2, ..., v_n)$ with variable ordering $v_1 < v_2 < ... < v_n$. Variables $v_{k+1}, v_{k+2}, ..., v_n$ are all q-linear if and only if for each of these variables, v_i, there is exactly one node, n_i, with decision variable v_i in the QDD of function f. The weight of the $\hat{1}$-edge of n_i will be $R_x(\theta_i)$. Also no edge originating from nodes above n_j (i.e., nodes with decision variable $v_j, j<i$) will end at a node below n_i (a node with decision variable $v_j, j>i$.)

Let v_k be the lowest indexed q-nonlinear variable after which $v_{k+1}, v_{k+2}, ..., v_n$ are q-linear variables of f. From Lemma 1,

$f_{v_j} = R_x(\theta_j) f_{\bar{v}_j}$, $k+1 \le j \le n$ where θ_j is fixed independent of the input combination of $v_1, v_2, ..., v_{j-1}, v_{j+1}, ..., v_n$. Every path from the root node of the QDD to its terminal node will either go thru an internal node with decision variable v_k or it will skip any such node and directly go to the single QDD node with decision variable v_{k+1}. For the latter case, $f_{v_k} = R_x(0) f_{\bar{v}_k} = f_{\bar{v}_k}$ and for the former case, $f_{v_k} = R_x(\alpha_i) f_{\bar{v}_k}$ where there will be as many different rotation angles (e.g., α_1, α_2) for variable v_k as there are internal nodes with decision variable v_k in the QDD.

Definition: The *degree of q-nonlinearity* of variable v_k is m-1 where m denotes the number of different rotation angles α_i (including 0 if any) that $f_{v_k} = R_x(\alpha_i) f_{\bar{v}_k}$ for some $v_1, v_2, ..., v_{k-1}, v_{k+1}, ..., v_n$. For q-linear variables the degree of q-nonlinearity is zero.

Theorem 1: Consider function $f(v_1, v_2, ..., v_n)$ with variable ordering $v_1 < v_2 < ... < v_n$. Assume that $v_{k+1}, v_{k+2}, ..., v_n$ are q-linear variables of f and v_k is a q-nonlinear variable of f with degree of q-nonlinearity m-1 (i.e., for each value assignment to $v_1, v_2, ..., v_{k-1}, v_{k+1}, ..., v_n$.) Exactly one of the following m relations holds: $f_{v_k} = R_x(\alpha_1) f_{\bar{v}_k}$, …, $f_{v_k} = R_x(\alpha_m) f_{\bar{v}_k}$.) Let function g be defined as follows. If $f_{v_k} = R_x(\alpha_1) f_{\bar{v}_k}$, then $g = \hat{1}$ else $g = \hat{0}$. Function f can be bi-decomposed as: $f = g_1 R_x(\gamma) h$ where: $g_1 = v_k R_x(\pi) g$, $\gamma = (\alpha_2 - \alpha_1)/2$, $h = g_1 R_x(-\gamma) f$ and g_1 will be a function of $v_1, v_2, ..., v_k$ (i.e., g_1 will be invariant of $v_{k+1}, v_{k+2}, ..., v_n$) and v_k will be q-linear in function g_1. In addition, h will be a function of $v_1, v_2, ..., v_n$; $v_{k+1}, v_{k+2}, ..., v_n$ will be q-linear in function h; and the degree of q-nonlinearity of v_k in h will be less than or equal to m-2.

Using the proposed bi-decomposition approach f can be bi-decomposed into $f = g_1 R_x(\gamma) h$ where g_1 and h are recursively bi-decomposed until a quantum factored form is obtained. Since g_1 is independent of $v_{k+1}, v_{k+2}, ..., v_n$ and v_k in g is q-linear and degree of q-non linearity of v_k in h is at most m-2, the recursion will finally stop at terminal cases where g_1 and/or h have directly realizable QDD's, i.e., all the variables will be q-linear in the functions and hence they will have cascade forms corresponding to QDD's with a chain structure similar to QDD's in Figure 5. As a result of Lemma 1, in a function with chain structured QDD, all variables are q-linear. The algorithm, *q-factor*(*f*), uses recursive bi-decomposition in Theorem 1 to generate a quantum factored form for a function *f*.

Algorithm: *q-factor* (f)

0- If all variables are q-linear then return the corresponding cascade form for f;

1- Find the lowest indexed q-nonlinear variable, v_k, after which $v_{k+1}, v_{k+2}, ..., v_n$ are q-linear;

2- Bi-decompose f as $f = g_1 R_x(\gamma) h$ where g_1, h and γ are given in Theorem 1;

3- Return [*q-factor*(g_1)] $R_x(\gamma)$ [*q-factor*(h)];

It is important to notice that all of the above steps can be directly performed on QDD's. For example if the QDD of a function, *f*, is a chain structure, there exists a cascade form for *f* (step 0). For step 1, according to Lemma 1, identifying v_k is equivalent to identifying the lower chain-structure part of the QDD. As for step 2, according to Lemma 2, values $\alpha_1, \alpha_2, ..., \alpha_m$ can be obtained from the weights of the $\hat{1}$ -edges of nodes with decision variable v_k. Hence, $\gamma = (\alpha_2 - \alpha_1)/2$ can also be obtained. Let n_i denote the node with decision variable v_k and $\hat{1}$ -edges weight $R_x(\alpha_i)$. The QDD of g_1 can be constructed from QDD of *f* with the following method.

Starting from the QDD of *f*, change all weights to $R_x(0) = I$; create a QDD node, v_k, representing v_k. as depicted in Figure 6; redirect all edges toward n_1 to node v_k and make the weight of all such edges $R_x(\pi)$; redirect all edges toward n_2, n_3, …, n_m to node v_k and make the weight of all such edges $R_x(0)$; discard nodes n_1, n_2, …, n_m; and finally merge isomorphic sub-graphs, eliminate nodes with same $\hat{0}$ -child and the $\hat{1}$ -child if the weight of the $\hat{1}$ -edge is $R_x(0) = I$, and update weights of the QDD to make the QDD of g_1 canonical.

Having the QDD's for g_1 and *f*, the QDD of $h = g_1 R_x(-\gamma) f$ can be obtained using the q-apply operation.

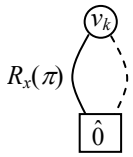

Figure 6. QDD for the node v_k.

The final factored form resulting from q-apply will be: $f = g_1 R_x(\gamma_1)\left[g_2 R_x(\gamma_2)\left[g_3 R_x(\gamma_3) \ldots \left[g_k R_x(\gamma_k)\hat{0}\right]\right]\right]$ which may also be written as:

$$f = g_{p_1} R_x(\gamma_{p_1})\left[g_{p_2} R_x(\gamma_{p_2})\left[g_{p_3} R_x(\gamma_{p_3}) \ldots \left[g_{p_k} R_x(\gamma_{p_k})\hat{0}\right]\right]\right]$$

where $(p_1, p_2, ..., p_k)$ is a permutation of $(1,2,...,k)$. Note that g_i functions should be decomposed as well using q-apply. In the following example, it is shown that different permutations on $(1,2,...,k)$ may result in different number of gates while synthesizing the circuit.

The examples in this paper demonstrate the power of the proposed synthesis approach. The q-factor algorithm is not guaranteed to be optimal; however the examples show that the results of the q-factor match the previously-generated optimal circuits (obtained by semi-exhaustive search) by previous researchers, which is one evidence for the effectiveness of the propose *automated* synthesis approach.

Example 1: In this part a four-input Toffoli gate, depicted in Figure 7 (i), will be synthesized by using the *q-factor* algorithm. Figure 7 (ii) shows the QDD of the output *s* of the Toffoli gate. Throughout the synthesis process we maintain the variable ordering *a*<*b*<*c*<*d*.

The resulting quantum circuit realization is depicted in Figure 8. The first part of the circuit (left of the dashed line) generates output *s* whereas the second part generates outputs *a*, *b* and *c*.

This realization of the 4-input Toffoli gate can be generalized for *n*-input Toffoli gates. In [33] a method for synthesizing an n-input Toffoli gate (including the 4-input gate) is provided which produces a synthesis result similar to ours. However the approach in [33] is specialized for gates similar to the n-input

Toffoli gates while our approach automatically and without assuming any prior knowledge of the function, synthesizes the circuit. The use of *q-factor* for synthesizing the *n*-input Toffoli gate *automatically* generates the circuit structures which were first reported in [33]. In our view, this fact alone demonstrates the efficacy of the proposed approach. The details can be verified by the reader.

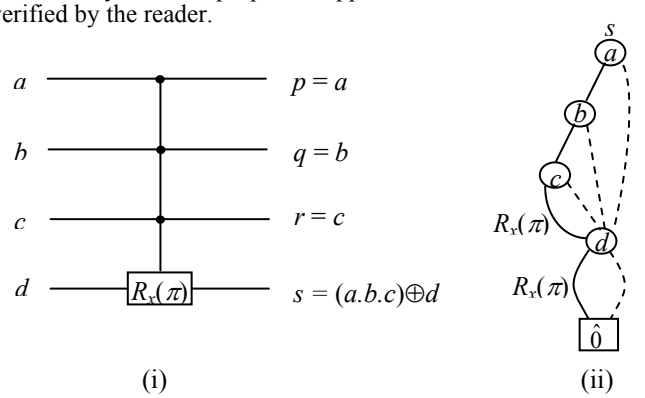

Figure 7. Four-input Toffoli gate and QDD for output, *s*.

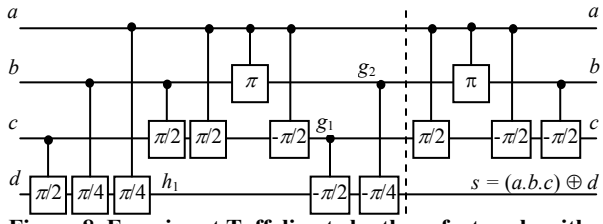

Figure 8. Four-input Toffoli gate by the q-factor algorithm.

Example 2. Consider a full adder with inputs x_1, x_2 and x_3 and outputs s (sum) and c (carry out): $s = x_1 \oplus x_2 \oplus x_3$ and $c = (x_1.x_2) + (x_1.x_3) + (x_2.x_3)$ where '+' is binary 'OR' operation. The resulting quantum circuit realization is depicted in Figure 9.

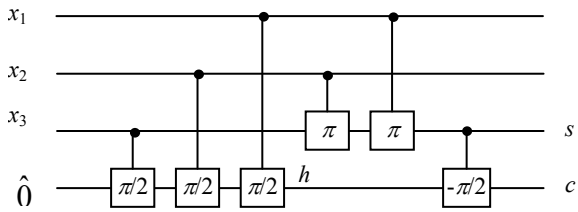

Figure 9. Quantum full adder.

The authors of [24] reported the run-time of the algorithm for optimal synthesis of a single-bit adder with 6 quantum gates as 7 hours on a 850MHz Pentium III processor running Linux. As we can observe, our method results in a circuit with the same number of quantum gates in virtually no time.

5. Conclusions

An efficient analysis and synthesis framework for quantum logic circuits was presented. We introduced the quantum factored forms, and developed a canonical and concise representation of quantum logic circuits. The focus of our approach was on the most basic quantum operators, i.e., the rotation and controlled-rotation primitives. Finally, an effective QDD-based algorithm, i.e., *q–factor* for automatic synthesis of quantum circuits was introduced. The results of applying q-factor to *n*-input Toffoli gate and full adders demonstrate the efficiency of the proposed approach.

References

[1] http://www.itrs.net/Common/2004Update/2004Update.htm

[2] R. P. Feynman, "Simulating Physics with Computers," *Int'l Journal of Theoretical Physics*, 21, 1982, pp. 467-488.

[3] D. Deutsch, "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer," *Royal Society*, A, 400, 1985, pp. 97-117.

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

[5] C. P. Williams, S. H. Clearwater, *Explorations in Quantum Computing*, Springer-Verlag, 1998.

[6] M. Hirvensalo, *Quantum Computing*, Springer Verlag, 2001.

[7] R. Landauer, "Irreversibility and Heat Generation in the Computational Process,"*IBM Journal of Research and Development*, 5, 1961, pp.183-191.

[8] R. Keyes, R. Landauer, "Minimal Energy Dissipation in Logic," *IBM Journal of Research and Development*, 14, 1970, pp. 152-157.

[9] P. A. M. Dirac, *The Principles of Quantum Mechanics*, Oxford University Press, 1st Edition, 1930.

[10] J. von Neumann, *Mathematical Foundations of Quantum Mechanics*, Princeton Univ. Press, 1950.

[11] K. Iwama, Y. Kambayashi, S. Yamashita, "Transformation Rules for Designing CNOT-Based Quantum Circuits," *DAC*, 2002, pp.419-424.

[12] A. Khlopotine, M. Perkowski, P. Kerntopf, "Reversible Logic Synthesis by Iterative Compositions," *IWLS*, 2002, pp. 261-266.

[13] D.M. Miller,"Spectral and Two-Place Decomposition Techniques in Reversible Logic," *Midwest Symp. on Circuits and Systems*, CD-ROM, 2002.

[14] M. Perkowski, et. al., "Regularity and Symmetry as a Base for Efficient Realization of Reversible Logic Circuits," *IWLS*, 2001, pp. 90-95.

[15] T. Toffoli, *Reversible Computing*, Lab. for Computer Science, MIT, Cambridge, MA, Technical Memo. MIT/LCS/TM-151, 1980.

[16] P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates," *Intl. Workshop Logic Synthesis,* 2000, pp. 261-269.

[17] D. M. Miller, D. Maslov, G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis,"*Design Automation Conf.* 2003, pp. 318-323.

[18] V. V. Shende , A. K. Prasad, I. L. Markov, J. P. Hayes, "Synthesis of Reversible Logic Circuits," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 22(6), 2003, pp. 710-722.

[19] A. Agrawal, N. K. Jha, "Synthesis of Reversible Logic," *Design Automation and Test in Europe*, 2004, pp. 21384-21385.

[20] A. Al-Rabadi, "Quantum Circuit Synthesis Using Classes of GF(3) Reversible Fast Spectral Transforms," *Int'l Symp. on Multi Valued Logic,* 2004, pp. 87-93.

[21] V. V. Shende, S. S. Bullock, I. L. Markov, "Synthesis of Quantum Logic Circuits," *A SP Design Automation Conf.*, 2005, pp. 272-275.

[22] L. Storme et al., "Group Theoretical Aspects of Reversible Logic Gates," *Journal of Universal Computer Science* 5, 1999, pp 307-321.

[23] A. De Vos et al., "Generating the Group of Reversible Logic Gates," *Journal of Physics A: Mathematical and General*,35, 2002, pp. 7063-7078.

[24] W. Hung, X. Song, G. Yang, J.Yang, M. Perkowski, "Quantum Logic Synthesis by Symbolic Reachability Analysis," *Design Automation Conference*, 2004, pp.838-841.

[25] A. Al-Rabadi, L. Casperson and M. Perkowski, Multiple-valued quantum logic, *Quantum Computers and Computing*, Vol. 3, Number 1.

[26] G.F.Viamontes, I.L.Markov and J.P. Hayes, "Improving Gate-Level Simulation of Quantum Circuits1," *Quantum Information Processing*, Vol. 2, No. 5, 2003

[27] M. Lukac et al, "Evolutionary Approach to Quantum and Reversible Circuits Synthesis," *Artificial Intelligence in Logic Design*, Kluwer Academic Publisher, 2004, pp. 361-417.

[28] J. I. Cirac, P. Zoller, "Quantum Computation with Cold Trapped Ions," *Physical Review*, 74, Issue 20, 1995, pp. 4091-4094.

[29] C. Monroe, et. al., "Simplified Quantum Logic with Trapped Ions," *Physical Review* A, 55, Issue 4, 1997, pp. 2489-2491.

[30] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. 35, 1986, pp. 677-691.

[31] K. Brace, R. Rudell, and R. Bryant, "Efficient Implementation of a BDD Package," *Design Automation Conf.*, 1990, pp. 40-45.

[32] Y.-T. Lai, M. Pedram, and S. Vrudhula, "EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition," *IEEE Trans. on Computer-Aided Design*, 8, 1994,pp. 959-975.

[33] A. Barenco et al., "Elementary Gates for Quantum Computation," *Physical Review* A, 52, 1995, pp. 3457-3467.