Optimal Synthesis of Multiple Output Boolean Functions Using a Set of Quantum Gates by Symbolic Reachability Analysis

William N. N. Hung, Xiaoyu Song, Guowu Yang, Jin Yang, and Marek Perkowski

***Abstract*—This paper proposes an approach to optimally synthesize quantum circuits by symbolic reachability analysis, where the primary inputs and outputs are basis binary and the internal signals can be nonbinary in a multiple-valued domain. The authors present an optimal synthesis method to minimize quantum cost and some speedup methods with nonoptimal quantum cost. The methods here are applicable to small reversible functions. Unlike previous works that use permutative reversible gates, a lower level library that includes nonpermutative quantum gates is used here. The proposed approach obtains the minimum cost quantum circuits for Miller gate, half adder, and full adder, which are better than previous results. This cost is minimum for any circuit using the set of quantum gates in this paper, where the control qubit of 2-qubit gates is always basis binary. In addition, the minimum quantum cost in the same manner for Fredkin, Peres, and Toffoli gates is proven. The method can also find the best conversion from an irreversible function to a reversible circuit as a byproduct of the generality of its formulation, thus synthesizing in principle arbitrary multi-output Boolean functions with quantum gate library. This paper constitutes the first successful experience of applying formal methods and satisfiability to quantum logic synthesis.**

***Index Terms*—Formal verification, logic synthesis, model checking, quantum computing, reversible logic, satisfiability.**

I. Introduction

REVERSIBLE logic [1] plays an important role in the synthesis of quantum computing circuits [2], [3]. The synthesis of reversible logic circuits using elementary quantum gates [4], [5] is different from classical (nonreversible) logic synthesis. There are some works [6]–[9] on reversible logic synthesis using basic reversible gates (Toffoli, Fredkin [10], or Feynman gates). However, these reversible logic gates have different quantum implementation costs (e.g., the cost of Feynman is lower than Toffoli). Therefore, finding the smallest number of gates to synthesize a reversible circuit does not necessarily result in quantum implementation with the lowest cost (in terms of quantum gates).

Manuscript received November 8, 2004; revised February 22, 2005 and June 8, 2005. This paper was recommended by Associate Editor J. H. Kukula.

W. N. N. Hung is with Synplicity Inc., Sunnyvale, CA 94086 USA (e-mail: william_hung@alumni.utexas.net).

X. Song, G. Yang, and M. Perkowski are with Portland State University, Portland, OR 97207 USA.

J. Yang is with Strategic CAD Labs, Intel Corporation, Hillsboro, OR 97124 USA.

Digital Object Identifier 10.1109/TCAD.2005.858352

In this paper, we focus on synthesizing reversible circuits using quantum primitives with the lowest total cost using a library of basic 2-qubit quantum gates, which will be described in Section III. Our synthesis method can also be modified to use other libraries of gates. We chose a library of basic 2-qubit quantum gates in this paper as they allow us to better evaluate the quantum implementation costs. The circuits we synthesized include common reversible gates that can next be used at higher levels of logic synthesis. Our approach can also be used as an equivalent of "technology mapping" for quantum circuits.

We reduce the quantum logic synthesis problem to multiple-valued logic synthesis; this reduction simplifies the search space and reduces the algorithm complexity. We formulate the above quantum logic synthesis task via symbolic reachability analysis [11], [12]. We used satisfiability-based model checking to solve the problem, but other decision methods or combinatorial optimization techniques can be similarly applied here. Our method not only guarantees to find a quantum implementation (for reversible circuits) but also guarantees the lowest quantum cost in the synthesized result (for the set of circuits where the control qubit of our 2-qubit gates is always basis binary). We also introduce an automated way of adding ancilla qubits and finding their appropriate constant values in the synthesis process. Thus, even irreversible circuits can be converted to reversible circuits that in turn are synthesized by our method. In contrast to previous works, which either use permutative reversible gates to design permutative circuits or universal quantum gates to design quantum circuits, we use a subset of quantum gates to design permutative circuits.

II. Background

Given a function f, we say f is reversible if and only if there exists a function g such that $x = g(f(x))$ for all x in the domain of f. The corresponding function g (as described above) is usually referred to as f^{-1}. Given n Boolean inputs, any multiple-output Boolean function on such n Boolean inputs must have exactly n Boolean outputs so that it is reversible [2]. We use $n \times n$ to denote a reversible function with n Boolean inputs and n Boolean outputs. Given an $n \times n$ reversible function f, there are 2^n input rows and 2^n output rows in the truth table of f. The output rows must be a permutation of the input rows in the truth table of f.

In quantum computing [2], the fundamental information unit is a qubit. The state of a qubit is a superposition of 0 and 1

states, also denoted as $|0\rangle$ and $|1\rangle$, respectively. The qubit state q can be represented by

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

where α and β are both complex numbers and $|\alpha|^2 + |\beta|^2 = 1$.

The classical state of binary 0 corresponds to the case where $\alpha = 1$ and $\beta = 0$. Similarly, the classical state of binary 1 corresponds to $\alpha = 0$ and $\beta = 1$. We refer to them as basis binary 0 and basis binary 1, respectively. All other combinations of α and β are not basis binary. The quantum state of a single qubit is usually denoted by the vector

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix}.$$

Given the state of each qubit, the overall quantum state is a Kronecker product of the states of each qubit. Take two qubits for example

$$\begin{pmatrix}u_0\\ u_1\end{pmatrix} \otimes \begin{pmatrix}v_0\\ v_1\end{pmatrix} = \begin{pmatrix}u_0\begin{pmatrix}v_0\\ v_1\end{pmatrix}\\ u_1\begin{pmatrix}v_0\\ v_1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}u_0v_0\\ u_0v_1\\ u_1v_0\\ u_1v_1\end{pmatrix}. \quad (1)$$

Notice that if the individual qubits are basis binary, then the Kronecker product is simply an enumeration of all the possible binary values (truth table) of its qubits. If we can use the quantum state of multiple qubits to determine the individual state of each qubit (such as the above case), we call it a separable state. There are some cases where the quantum state cannot be separated into individual states of each qubit, i.e., we cannot describe (mathematically) the state of each qubit but we can describe the quantum state of all the qubits combined. We call such states entangled states. This idea of entangled state is called quantum entanglement, and it originated from the Einstein–Podolsky–Rosen paradox [13].

The effect of quantum gates on a quantum state can be described as vector operations, where the quantum gates are represented by unitary matrices. A unitary matrix is a $n \times n$ complex matrix M with the property

$$M \times M^+ = M^+ \times M = I$$

where I is the identity matrix and M^+ is the conjugate transpose (also known as the Hermitian adjoint) of M.

Given an n-qubit quantum gate G, we call G a permutative quantum gate if and only if the outputs of G are all basis binary when its inputs are all basis binary, i.e., G is a permutative quantum gate if and only if G implements an $n \times n$ Boolean reversible function (when its inputs are basis binary).

A generalized 2-qubit controlled U gate [5] is shown in Fig. 1. Its unitary matrix is

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & u_{00} & u_{01}\\ 0 & 0 & u_{10} & u_{11}\end{pmatrix}$$

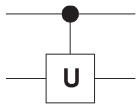


Fig. 1. Controlled-U gate.

where the four entries in the right bottom also form a (single qubit) unitary matrix U by itself

$$U = \begin{pmatrix}u_{00} & u_{01}\\ u_{10} & u_{11}\end{pmatrix}.$$

It has been shown [4], [5] that permutative quantum logic circuits can be constructed using elementary quantum, XOR, controlled-V, controlled-V^+, or NOT gates, as shown in Fig. 2. The NOT gate is also called an inverter. Its unitary matrix is

$$M_{\mathrm{NOT}} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}.$$

Quantum XOR gates are also called Feynman gates or controlled-NOT (CNOT) gates. The controlled-V gate's data output is the same as its data input (B) when its control input (A) value is 0 (FALSE). When its control value is 1 (TRUE), the data output becomes V (input) [2]

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

Similar rules apply to the controlled-V^+ gate, except that its data output becomes V^+(input), where V^+ is the Hermitian of V, i.e.,

$$\frac{1+i}{2}\begin{pmatrix}1 & -i\\ -i & 1\end{pmatrix} \times \frac{1-i}{2}\begin{pmatrix}1 & i\\ i & 1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}.$$

The quantum XOR (controlled-NOT), controlled-V, and controlled-V^+ are all special cases of the generalized controlled-U gate, where the matrix U corresponds to M_{NOT}, V, and V^+, respectively.

According to [2], the values V and V^+ are constructed such that they are the square root of NOT (i.e., inverter gate): $V \times V = V^+ \times V^+ = M_{\mathrm{NOT}}$. Hence, if the signal V (input) is passed through another controlled-V gate with its control value also equal to 1 (TRUE), the output of the second gate becomes the NOT of the input.

The quantum XOR, controlled-V, and controlled-V^+ gates are 2×2 gates. They are also called 2-qubit gates. Similarly, the NOT gate (inverter) is a 1-qubit gate. For quantum implementation, the cost of 2-qubit gates far exceeds the cost of 1-qubit gates. Hence, in a first approximation, the quantum cost of 1-qubit gates is usually ignored in the presence of 2-qubit implementations [5], [14].

In this paper, we adopt the quantum gate cost evaluation introduced in [4]. According to the method in [4], each of the 2-qubit gates (quantum XOR, controlled-V, controlled-V^+) has a quantum implementation cost of 1. In addition, when both quantum XOR and controlled-V (or controlled-V^+) are operating on the same two qubits in a

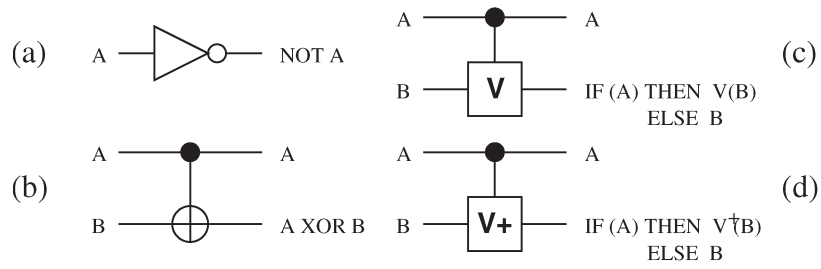


Fig. 2. Elementary quantum logic gates.

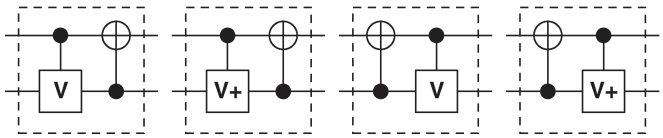


Fig. 3. Merged 2-qubit gates.

symmetric pattern (shown in Fig. 3), their total cost is considered as 1 as well. A more accurate cost function can be created for a particular quantum technology such as nuclear magnetic resonance (NMR) [15], but for simplicity and comparison to previous work we will use here the cost function from [4].

Given a reversible function, the quantum logic synthesis task considered in this paper is to synthesize the function using the above elementary quantum logic gates with the minimum cost. Various heuristic methods have been applied to find low-cost quantum implementations (using the elementary gates) for the functionality of the Fredkin [4], Toffoli [16], and Peres [17] gates. Yet, nobody has been able to prove that they have the lowest quantum cost implementation (based on the cost evaluation criteria given above).

We can perform the above quantum logic synthesis task through reachability analysis. Symbolic reachability analysis is a well-known technique in formal verification [11]. Its basic idea is to find all the reachable states of a finite state machine (FSM). Using symbolic representation, we can check if an invariant (property) is true for all reachable states. This technique is used in invariant checking [11], where the state space is traversed exhaustively against an invariant. Since the state space tends to be large for practical systems, recent symbolic reachability analysis techniques use various methods, such as binary decision diagram (BDD) [18], [19] or satisfiability (SAT), to avoid enumerating every system state while preserving the completeness of the reachability analysis. We use state-of-the-art SAT-based bounded model checking [12] to check invariants. If the invariant is false, it can automatically generate a counter-example. We can find the shortest counter-example in this way by starting with a zero bound and gradually incrementing the bound. If the invariant is true and given enough time, this method can also check that the bound is sufficiently large and establish the proof. SAT-based model checking has been successfully deployed in the industry [20]–[22].

III. Symbolic Formulation

We consider each "quantum wire" of the quantum circuit as a superposition of $|1\rangle$ and $|0\rangle$, denoted as 1 and 0, respectively. We are interested in synthesizing quantum circuits with basis binary inputs (1 and 0). The values of these signals are modified after passing through elementary gates (Fig. 2). There are six possible output values when we apply binary (1 and 0) inputs to one of those elementary gates: 0, 1, V_0, V_1, V_0^+, V_1^+, where V_0 represents V(input) when the input is 0, and similarly for V_1, V_0^+, V_1^+, i.e.,

$$V_0 = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1+i}{2}\begin{pmatrix} 1 \\ -i \end{pmatrix}$$

$$V_1 = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1+i}{2}\begin{pmatrix} -i \\ 1 \end{pmatrix}$$

$$V_0^+ = \frac{1-i}{2}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1-i}{2}\begin{pmatrix} 1 \\ i \end{pmatrix}$$

$$V_1^+ = \frac{1-i}{2}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1-i}{2}\begin{pmatrix} i \\ 1 \end{pmatrix}.$$

These six possible values are used as input values to gates in subsequent stages. We want to synthesize our circuit such that the "control" input of controlled-NOT (quantum XOR), controlled-V, or controlled-V^+ is always basis binary (0s and 1s), i.e., their input values cannot be V_0 or V_1, etc.

We impose the above restriction because a nonbinary value at the control input of the controlled-NOT, controlled-V, or controlled-V^+ gate can generate an entangled quantum state. For example, if we have V_0 at both control and data inputs of the controlled-V gate, the unitary matrix multiplied by the Kronecker product (of the inputs) becomes

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix} \times \begin{pmatrix} 0.5i \\ 0.5 \\ 0.5 \\ -0.5i \end{pmatrix} = \begin{pmatrix} 0.5i \\ 0.5 \\ 0 \\ 0.5 - 0.5i \end{pmatrix}.$$

The vector result cannot be separated into two individual qubit states using (1). The u_1u_0 entry from (1) is 0, which requires $u_1 = u_0 = 0$. It contradicts with the other entries of the vector. This is an entangled quantum state. Similar scenarios exist for controlled-V^+. The controlled-NOT also has similar examples [23]. For the rest of this paper, we focus on synthesizing quantum circuits using our set of quantum gates (NOT, controlled-NOT, controlled-V, and controlled-V^+), where the control input of the 2-qubit gates is always basis binary. However, the same approach can be used to synthesize circuits using other libraries of quantum gates as long as it can be reduced to a multiple-valued logic problem.

Based on the unitary matrices in Section II, we can see that if the input of the NOT gate is not basis binary, namely V_0, V_1, V_0^+, or V_1^+, its corresponding output is V_1, V_0, V_1^+, or V_0^+, respectively. Given a basis binary 1 on the control input of the controlled-NOT gate, the data input and the data output exhibit the same property (above) as the NOT gate. Also, as shown in Section II, given the six possible values (0, 1, V_0, V_1, V_0^+ or V_1^+) at the data input of the controlled-V or controlled-V^+, their corresponding data output has the same set of six possible values. Hence, the input/output of every quantum gate in the circuit can be represented using the above six values.

If we look at the complex matrix representation of V_0, V_1, V_0^+, and V_1^+, we can deduce that $V_0 = V_1^+$

$$V_0 = \frac{1+i}{2}\begin{pmatrix} 1 \\ -i \end{pmatrix} = \begin{pmatrix} 0.5+0.5i \\ 0.5-0.5i \end{pmatrix}$$

$$V_1^+ = \frac{1-i}{2}\begin{pmatrix} i \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5+0.5i \\ 0.5-0.5i \end{pmatrix}$$

and $V_1 = V_0^+$

$$V_1 = \frac{1+i}{2}\begin{pmatrix} -i \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5-0.5i \\ 0.5+0.5i \end{pmatrix}$$

$$V_0^+ = \frac{1-i}{2}\begin{pmatrix} 1 \\ i \end{pmatrix} = \begin{pmatrix} 0.5-0.5i \\ 0.5+0.5i \end{pmatrix}.$$

Thus, it suffices to represent signals in the circuit using four values: 0, 1, V_0, V_1. In this way, we reduce the problem of quantum circuit synthesis (which would normally use unitary matrices and Hilbert space to represent signals) to a simpler synthesis problem in mixed binary/quaternary algebra. This is a general approach to efficiently synthesize a subclass of quantum circuits. It can be applied to gates other than the 2-qubit gates introduced above.

Theorem 1: For any deterministic quantum circuit (with n qubits, $n > 0$) that produces basis binary outputs for basis binary inputs, its unitary matrix is canonical, i.e., there is only one unitary matrix that represents the function of this circuit. This is a permutation matrix.

Proof: We prove the theorem in four steps. Step 1): There are $2^n!$ distinct $n \times n$ binary reversible logic functions. Step 2): When all n qubits are basis binary, their Kronecker product has one entry equal to 1 while all the other entries are equal to 0. Step 3): Each row or column of the unitary matrix should have only one entry equal to 1 while all the other entries are equal to 0. Step 4): The unitary matrix must be unique under the above circumstances.

Step 1) The function of this quantum circuit is a binary reversible logic function. The output entries in the truth table are permutations of the input entries for this function. The truth table has 2^n rows, i.e., 2^n distinct binary input entries (and corresponding output entries). Since the output entries are permutations of the 2^n input entries, there are $2^n!$ ways to permute them. Hence, there are $2^n!$ distinct $n \times n$ binary reversible logic functions.

Step 2) The Kronecker product of n qubits is

$$\begin{pmatrix} \alpha_1 \\ \beta_1 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} \alpha_n \\ \beta_n \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_2, \ldots, \alpha_{n-1}\alpha_n \\ \alpha_1\alpha_2, \ldots, \alpha_{n-1}\beta_n \\ \vdots \\ \beta_1\beta_2, \ldots, \beta_{n-1}\beta_n \end{pmatrix}.$$

For each qubit, $\alpha\beta$ have only two choices (10 or 01) to be basis binary. There are 2^n distinct ways for all n qubits to be basis binary. Under this circumstance, the above Kronecker product is an enumeration of the truth table patterns for α and β of each qubit. Hence, there is one entry in the Kronecker product equal to 1 while all the other entries are equal to 0.

Step 3) Let U be a unitary matrix of the n-qubit circuit. There are 2^n rows and 2^n columns in U. Let P and Q be the Kronecker product of the input and output for this circuit, respectively. We have

$$U \times P = Q. \tag{2}$$

According to Step 2), the vector P has one entry equal to 1 and all the other entries are 0. Similarly, the vector Q has one entry equal to 1 and all the other entries are 0. We use u_{ij} to denote the value of matrix U in the ith row and jth column, and p_i and q_i to denote the value of vector P and Q in the ith row, respectively.

Given $0 \leq i \leq 2^n$, suppose all the entries in the ith row of U are 0, then q_i will be 0 for all possible values of P due to (2). This is a contradiction because Q has 2^n rows and 2^n distinct values, so q_i must be 1 for one of those cases. Hence, any row of U cannot be all zeros.

Furthermore, suppose there are more than one entry that is nonzero (say columns u_{ij} and u_{ik} are nonzero), then we can have two distinct patterns of P, one with $p_j = 1$ and the other with $p_k = 1$, both being able to produce a nonzero q_i. Again, this is a contradiction because we can only have one possibility for q_i to be nonzero. Hence, every row of U must have exactly one nonzero entry. In order to produce a corresponding 1 in the vector Q, the nonzero entry in U must be 1.

Lastly, suppose we have $u_{ij} = 1$ and $u_{kj} = 1$, both in the jth column. We can pick a valuation of P with $p_j = 1$. The corresponding vector Q will have $q_i = 1$ and $q_k = 1$. This is again a contradiction since only one row of vector Q can be nonzero. Thus, every column of U must have exactly one nonzero entry (which must be 1).

Step 4) There are $2^n!$ possibilities for U to satisfy the property in Step 3), which is exactly the number of distinct permutations. Hence, to each permutation corresponds a unique unitary matrix U. This completes the Proof of Theorem 1. ■

The importance of the above theorem is that once we have specified the basis binary input/output behavior of the quantum

circuit, there is only one unitary matrix that can satisfy the specification (because it is canonical). Hence, the functional behavior of the synthesized quantum circuit, under nonbinary (complex number) input/outputs, would be deterministic, even though they were not in the original specification. This idea is especially important for the synthesis of binary reversible functions (Toffoli, Fredkin, etc.) using quantum gates. It suffices to specify the basis binary input/output behavior of the reversible function, and the synthesized quantum circuit would have identical behavior as those of classical quantum circuits for all quantum values.

Suppose we intend to synthesize an $n \times n$ reversible function R specified by its truth table with n input columns, n output columns, and 2^n rows corresponding to n output patterns using the 2-qubit quantum gates [Fig. 2(b)–(d)] described above. The synthesized result should be a cascade of L stages. Each stage consists of one of the above quantum gates. Since the function applies to n qubits and the quantum gates at each stage are 1-qubit or 2-qubit gates, the synthesized result should indicate to which qubits the gates are connected. For each stage i, we use g_i to represent the gate selection variable [Fig. 2(b)–(d)], and we use A_i and B_i to indicate the two qubits that the gate is connected to, i.e., $A_i, B_i \in \{1, \ldots, n\}$. As a naming convention, we refer to the qubit indicated by A_i [the upper qubit in Fig. 2(b)–(d)] as the control qubit, and we refer to the qubit indicated by B_i [the lower qubit in Fig. 2(b)–(d)] as the data qubit. Since the two qubits must be different, we have

$$A_i \neq B_i. \tag{3}$$

We denote the inputs of stage i as $\vec{U}_i$, where $\vec{U}_i = u_{1i}u_{2i}, \cdots, u_{ni}$. Each qubit ($u_{qi}$, $q = 1, \ldots, n$) of the stage i can have four possible values (0, 1, V_0, V_1). The output of stage i is denoted by $\vec{U}_{i+1}$, i.e.,

$$u_{q(i+1)} = \begin{cases} u_{A_i i} \oplus_Q u_{qi}, & (q = B_i) \wedge (g = \text{Fig. 1(b)}) \\ V(u_{qi}), & (q = B_i) \wedge (g = \text{Fig. 1(c)}) \wedge u_{A_i i} \\ V^+(u_{qi}), & (q = B_i) \wedge (g = \text{Fig. 1(d)}) \wedge u_{A_i i} \\ u_{qi}, & \text{otherwise.} \end{cases}$$

Note that we use $\oplus_Q$ to denote the quantum XOR operation.

Due to our restriction on the control input, the values V_0 and V_1 cannot be applied to the control input of controlled-NOT, controlled-V, or controlled-V^+ gates. We create a Boolean signal E_i to represent whether the gate has been erroneously configured (misconfigured) with the V_0 or V_1 values in the current (ith) synthesis stage or any previous synthesis stages. At the initial stage, there is no misconfiguration, and we initialize by setting

$$E_0 = 0.$$

As we move to subsequent stages, the E_{i+1} value (in stage $i + 1$) is 1 if either of the following two cases is true.

1) E_i (in the previous stage) is already 1.
2) The value of the control qubit $u_{A_i i}$ is not binary (where A_i is the control qubit).

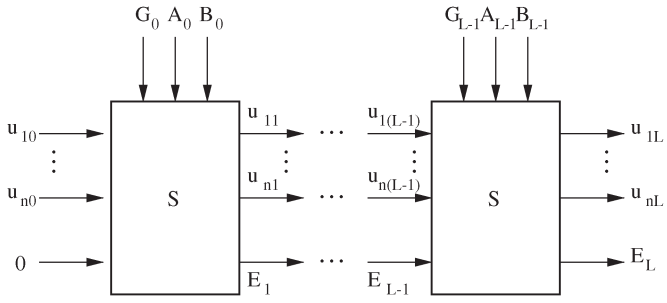


Fig. 4. L-2Syn problem.

Thus

$$E_{i+1} = E_i \vee (u_{A_i i} \notin \{0, 1\}).$$

So far, g_i had only three possible values [Fig. 2(b)–(d)]. To better reflect the quantum implementation cost, let us use a different gate selection variable G_i with seven possible values. G_i has all the three possible values of g_i, with four additional values to reflect the quantum XOR gate merged with controlled-V and controlled-V^+ gates (Fig. 3). We define the synthesis function S as

$$(\vec{U}_{i+1}, E_{i+1}) = S(G_i, A_i, B_i, \vec{U}_i, E_i). \tag{4}$$

Definition 1 (L-2Syn): The quantum logic synthesis problem for the reversible function R using 2-qubit gates as a cascade of L stages is to find a set of satisfying values to G_i, A_i, B_i (where $A_i \neq B_i$ and $i = 0, 1, \ldots, L-1$) such that $E_0 = E_L = 0$ and $\vec{U}_L = R(\vec{U}_0)$ for all possible Boolean input values of $\vec{U}_0$. Mathematically speaking, a solution to the L-2Syn problem exists if and only if

$$\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_{L-1} \exists A_{L-1} \exists B_{L-1} \cdot \left(\forall \vec{U}_0 \in \{0, 1\}^n \right.$$
$$\left. \cdot (E_0 = E_L = 0) \wedge \left(\vec{U}_L = R(\vec{U}_0) \right) \right) \wedge \left(\bigwedge_{i=0}^{L-1} A_i \neq B_i \right) \tag{5}$$

where $G_0A_0B_0, G_1A_1B_1, \ldots, G_{L-1}A_{L-1}B_{L-1}$ form a solution to the L-2Syn problem.

Fig. 4 illustrates the L-2Syn problem. Notice that we are performing $n \times n$ reversible logic synthesis here. E_0 is not an input constant to the reversible logic circuit because all the reversible gates use only qubits $1, \ldots, n$. The E_i ($i = 0, \ldots, n$) Boolean values are used to keep track of prohibited logic values, they are not a part of the reversible circuit.

Definition 2 (min-2Syn): The minimum length quantum logic synthesis problem for the reversible function R using 2-qubit gates (quantum XOR, controlled-V, controlled-V^+, or their merged versions) is to solve L-2Syn with the smallest possible number L.

Theorem 2: For any reversible function R that does not require inverters in its quantum implementation, finding its quantum logic implementation with the minimum cost is equivalent to solving the min-2Syn for R.

Proof: The min-2Syn solution consists of the smallest possible L stages where each stage has a quantum cost of 1. Thus, the minimum quantum cost is L. ■

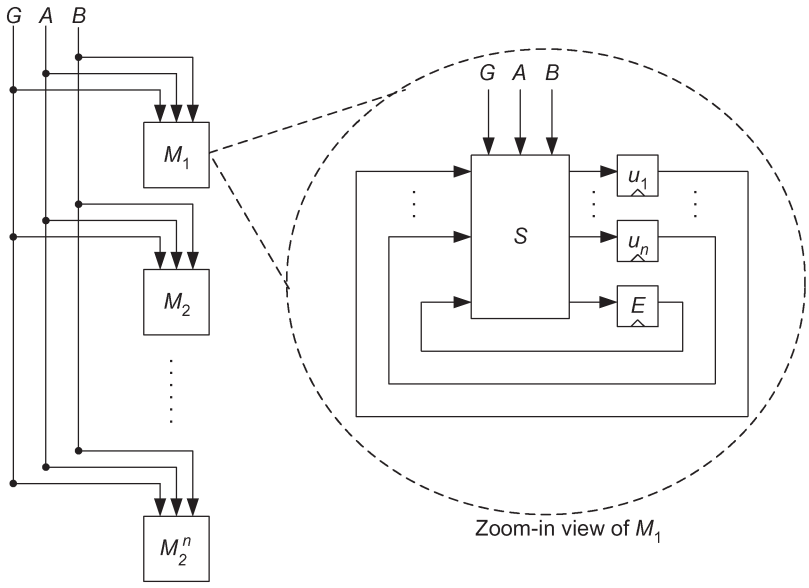


Fig. 5. FSM for reachability analysis.

So far, we have not considered inverters (1-qubit gates). Since the 1-qubit gate cost is negligible compared to 2-qubit gate costs, we can model our synthesis problem without worrying about the cost of inverters. This can be done by injecting inverters for each qubit at the inputs, outputs, and between stages. We can modify the equations mentioned in this section to arrive at a theorem similar to Theorem 2 for the minimum quantum logic implementation cost using inverters (1-qubit) or other 2-qubit gates.

IV. Reachability Analysis

Let us first formulate a solution for synthesizing reversible functions that do not require inverters. Later in this section, we will extend our formulation for any reversible function with or without inverters.

A. Invariant Checking

We have shown in Theorem 2 that finding the quantum implementation with the minimum cost of a reversible function (that does not require inverters) is equivalent to solving the min-2Syn problem.

We construct an FSM shown in Fig. 5, use a bounded model checker [12] to temporally unroll the FSM up to a specific bound, and invoke an SAT solver to find a counter-example. Our machine in Fig. 5 is in a way similar to Fig. 4, but there are some differences. Instead of cascading L instances of the S functional block in Fig. 4, we have 2^n parallel instances of FSMs $(M_1, \ldots, M_{2^n})$ in Fig. 5, as many as the number of rows in the truth table. Each FSM contains a functional block S. Three primary inputs (G, A, B) are fed to every FSM. Each machine has its own set of registers (or memory states) containing $\vec{U}$ (in terms of $u_1, \ldots, u_n$) and E.

The FSM will be initialized at time $t = 0$, and then proceeds to new states at $t = 1, 2, \ldots$. For convenience, we use $\vec{\mu}(M_h, t)$ to denote the value of the register vector $u_1, \ldots, u_n$ of machine M_h at time t, where $h = 1, \ldots, 2^n$. Similarly, we use $\varepsilon(M_h, t)$ to denote the value of the register E of machine M_h at time t. In addition, we use G_t, A_t, B_t to denote the input values at time t. As a constraint (environmental assumption), we require

$$\forall t \geq 0 \cdot (A_t \neq B_t). \tag{6}$$

From Fig. 5, we can see that the next state is computed from the current state and inputs through the combinational functional block S, i.e.,

$$(\vec{\mu}(M_h, t+1), \varepsilon(M_h, t+1)) = S\left(G_t, A_t, B_t, \vec{\mu}(M_h, t), \varepsilon(M_h, t)\right). \tag{7}$$

We initialize the E register of every machine to 0 (FALSE): $\varepsilon(M_h, 0) = 0$ for $h = 1, \ldots, 2^n$. We also initialize the $\vec{U}$ registers of every machine to their corresponding patterns in a truth table, i.e.,

$$\begin{aligned} M_1 &: \quad \vec{\mu}(M_1, 0) = 0 \ldots 00 \\ M_2 &: \quad \vec{\mu}(M_2, 0) = 0 \ldots 01 \\ &\vdots \\ M_{2^n} &: \quad \vec{\mu}(M_{2^n}, 0) = 1 \ldots 11. \end{aligned} \tag{8}$$

Given the reversible function R that we want to synthesize, we want to check the nonsynthesizeability invariant

$$inv(t) = \neg \bigwedge_{h=1}^{2^n} \left(\vec{\mu}(M_h, t) = R\left(\vec{\mu}(M_h, 0)\right)\right) \wedge \left(\varepsilon(M_h, t) = 0\right)$$

where $inv(t)$ is checked for all time $t \geq 0$.

Theorem 3: The function R is synthesizeable using 2-qubit gates if and only if there exists a counter-example (input sequence G_t, A_t, B_t for $t = 0, \ldots, L$) that satisfies (6)–(8) and violates the invariant $inv(t)$ at time $t = L$, where L is the

corresponding quantum cost using any of those seven 2-qubit gates presented in Section III.

Proof: Given a counter-example of length L, this counter-example will consist of assignments to the inputs G_t, A_t, B_t for $t = 0, \ldots, L$. The counter-example satisfies the initial condition (8), which means that all Boolean patterns (from the truth table) for $\vec{U}_{t=0}$ have been explored. The initial condition essentially states that

$$\forall \vec{U}_{t=0} \in \{0,1\}^n \cdot E_{t=0} = 0. \tag{9}$$

For any $t > 0$, the machine states $\vec{\mu}(M_h, t)$ and $\varepsilon(M_h, t)$ are computed from their initial states $\vec{\mu}(M_h, 0)$ and $\varepsilon(M_h, 0)$ and the inputs $G_{t'}$, $A_{t'}$, $B_{t'}$ for $t' = 0, \ldots, t-1$. Since our initial condition explored all possible patterns of $\vec{U}_{t=0}$, any formula of the form

$$\forall h \in \{1, \ldots, 2^n\} \cdot (\varepsilon(M_h, 0) = 0) \wedge f\left(\vec{\mu}(M_h, t), \varepsilon(M_h, t)\right)$$

can be rewritten as $\forall \vec{U}_{t=0} \in \{0,1\}^n \cdot (E_0 = 0) \wedge f(\vec{U}_t, E_t)$. We can conjunct the violated invariant $inv(t)$ with the initial condition (9) and rewrite them as

$$\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_L \exists A_L \exists B_L \cdot \forall \vec{U}_{t=0} \in \{0,1\}^n \cdot (E_{t=0} = 0) \wedge (E_{t=L} = 0) \wedge \left(\vec{U}_{t=L} = R(\vec{U}_{t=0})\right). \tag{10}$$

The existence of a counter-example is equivalent to the conjunction of formulae (6), (9), and (10). We can rewrite (6) as $\bigwedge_{t=0}^{L} A_t \neq B_t$. We can also push the conjunction inside the quantification operators to obtain

$$\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_L \exists A_L \exists B_L \cdot \left(\bigwedge_{t=0}^{L} A_t \neq B_t\right) \wedge \forall \vec{U}_{t=0} \in \{0,1\}^n \cdot (E_{t=0} = E_{t=L} = 0) \wedge \left(\vec{U}_{t=L} = R(\vec{U}_{t=0})\right). \tag{11}$$

Equation (11) characterizes the Boolean condition for the existence of a counter-example. The difference between (11) and (5) is that the existential quantification of 1inputs G_t, A_t, B_t and the constraint on input assumption $A_t \neq B_t$ ranges from 0 to L in formula (11) but only ranges from 0 to $L-1$ in (5). Now observe that the registers (E and $\vec{U}$) in our FSM (Fig. 5) depend only on the input values of the previous time cycle. Therefore, the input values G_L, A_L, B_L do not affect the existence of our counter-example at all. Hence, the existence of a counter-example is equivalent to the existence of a solution to the L-2Syn problem. ■

We have shown that synthesizing the quantum logic is equivalent to finding a counter-example to the invariant checking problem. Using bounded model checking, we can find the existence of a counter-example within the length of the bound. By starting with a small bound and gradually increasing the bound, we can find the shortest counter-example, essentially the minimum cost quantum implementation of the function R.

As mentioned in Section III, we can easily modify the above invariant checking formulation to find the minimum quantum implementation cost with inverters or other types of 2-qubit gates.

The invariant checking formulation is useful for synthesizing the quantum logic with the minimum cost as outlined above. In case the function R is not synthesizeable, as being not reversible, the model checker will prove the invariant has no counter-example (Theorem 3). However, we can easily add ancilla qubits (input constants) to transform nonreversible functions to reversible functions, thus making it synthesizeable. The next section describes an automatic approach for this transformation.

B. Synthesizing With Input Constants

Our formulations so far concentrated on synthesizing a function without additional input constants (ancilla qubits). However, some functions (e.g., irreversible functions) cannot be synthesized without input constants. For these functions, it makes sense to synthesize them with the minimum number of input constants.

We can add k input constants to the original $n \times n$ circuit, making it an $(n+k) \times (n+k)$ circuit, run it through our model checker, and see if we can get a counter-example or a proof. If we get a proof, we can increment k until we eventually get a counter-example (which should happen for finite k according to [10]). A systematic way of doing this is to start with $k = 1$ and gradually increment k until we reach a counter-example.

The invariant checking formulation with k input constants is slightly different from Section IV-A. For every input constant bit, we do not know if it should be a constant 0 or a constant 1. In order to get a counter-example (i.e., synthesize the circuit), we want to find out these constant values.

Let us look back at our machines in Fig. 5. From the figure, we have 2^n machines $(M_1, \ldots, M_{2^n})$, each with n registers. We modify this figure so that we have $n+k$ registers $(u_1, \ldots, u_{n+k})$ in each machine and each S (and δ if applicable) functional block will handle $n+k$ instead of n registers, as well as the E register.

For notational clarity, we still use $\vec{\mu}(M_i, t)$ to denote the value of the register vector $(u_1, \ldots, u_n)$ for machine M_i (where $i = 1, \ldots, 2^n$) at time t. We use $\nu_j(M_i, t)$ to denote the value of each register u_j (where $j = 1, \ldots, n+k$) of machine M_i (where $i = 1, \ldots, 2^n$) at time t. Thus, the newly introduced register values can be referred to as $\nu_{n+1}(M_i, t), \ldots, \nu_{n+k}(M_i, t)$.

Let us also introduce a new state ζ in addition to all the 2^n machines $(M_1, \ldots, M_{2^n})$. This register is initialized to 0 and then set to 1 thereafter, i.e.,

$$\zeta = \begin{cases} 0, & t = 0 \\ 1, & t > 0. \end{cases} \tag{12}$$

For those additional k registers, we want to limit their initial state to the set $\{0, 1\}$. In addition, we want to restrict the initial

state of the jth register ($j = n+1, \ldots, n+k$) at each machine to be the same

$$\bigwedge_{j=n+1}^{n+k} \bigwedge_{i=1}^{2^n} \nu_j(M_i, 0) \in \{0, 1\} \tag{13}$$

$$\bigwedge_{j=n+1}^{n+k} \zeta_t \vee (\nu_j(M_1, t) = \ldots = \nu_j(M_{2^n}, t)). \tag{14}$$

Equation (13) is possible because the symbolic model checking formulations [11] allow the initial state to be a set of values. The constraint (14) is used as an assumption that restricts the state space. Notice that we still have 2^n FSMs $(M_1, \ldots, M_{2^n})$ overall because the number of rows in the truth table for input patterns is still the same as in the case without the additional input constants.

By increasing the combinational function blocks S (and δ if applicable), we are essentially synthesizing for $(n+k) \times (n+k)$ reversible logic. Our initial state specification allows us to consider the constant 0 and constant 1 cases. The generated counter-example will contain specific values for the initial state of each bit, thus finding out the constant input values.

C. Example

Consider a classical computation unit, half adder, which takes two input bits $(n = 2)$ and outputs a sum and a carry. There are two input patterns in its truth table $(ab = 01, 10)$ that produce the same output pattern. Therefore, one needs to add a single input constant in order to separate 01 and 10 and to create a 3×3 reversible function R. We construct the 2^2 machines and the invariant according to Sections IV-A and B. A model checker can return a counter example that contains values of initial states and a sequence of input values. Most of the initial state values are already specified in (8), except for the state values of the input constant in (13). In this case, the model checker tells us that the input constant is 0, i.e.,

$$v_3(M_1, 0) = v_3(M_2, 0) = \cdots = v_3(M_{2^2}, 0) = 0.$$

Hence, the initial state values of the FSM are

$$\begin{aligned} M_1: &\quad \vec{\mu}(M_1, 0) = 000 \\ M_2: &\quad \vec{\mu}(M_2, 0) = 001 \\ M_3: &\quad \vec{\mu}(M_3, 0) = 010 \\ M_4: &\quad \vec{\mu}(M_4, 0) = 011. \end{aligned}$$

The sequence of input values in the counter-example is

$$\begin{aligned} G_0 &= V^+, & A_0 &= 2, & B_0 &= 3 \\ G_1 &= \text{XOR}, & A_1 &= 1, & B_1 &= 2 \\ G_2 &= V, & A_2 &= 2, & B_2 &= 3 \\ G_3 &= V\_\text{XOR}, & A_3 &= 1, & B_3 &= 3. \end{aligned}$$

Notice that the above input sequence satisfies the constraint (6). If we substitute these inputs back into the FSM, we will arrive at

$$\begin{aligned} M_1: &\quad \vec{\mu}(M_1, 4) = 000 \\ M_2: &\quad \vec{\mu}(M_2, 4) = 110 \\ M_3: &\quad \vec{\mu}(M_3, 4) = 010 \\ M_4: &\quad \vec{\mu}(M_4, 4) = 001. \end{aligned}$$

We compare the final state with the initial state. The initial state of $M_1, \ldots, M_4$ corresponds to the input patterns in a truth table. The final state of $M_1, \ldots, M_4$ corresponds to the output patterns in a truth table. Notice that the least significant bit (rightmost column) in the final state satisfies the carry function, and the middle bit (middle column) satisfies the summation function. The bottom bit in Fig. 10 is a garbage function $a \wedge \neg b$. Let us assume that in the circuit the top qubit corresponds to the least significant bit of our truth table, and similarly the bottom qubit corresponds to the most significant bit of our truth table. The gate types (quantum XOR, controlled-V, etc.) are already given by $G_0, \ldots, G_3$ of the counter-example. The connection of these gates to qubits are given by $A_0, B_0, \ldots, A_3, B_3$. These values directly translate to the circuit in Fig. 10.

V. Complexity and Time

Industrial experience [20], [22] suggests that the complexity of model checking is sensitive to the number of state retaining elements in the FSM. For our FSM in Fig. 5, there are $n \times 2^n$ registers, where n is the number of qubits. Each register has four possible values (0, 1, V, V^+). If we use Boolean states to encode these registers, we have $2n \times 2^n$ Boolean state elements. However, the number of qubits n tends to be small due to physical limitations. So far, the largest number [24] of qubits is 7, which is 1792 Boolean state elements. This is still manageable in the scope of industrial strength bounded model checkers [20], [22]. Nevertheless, we would like to speed up our synthesis process.

We introduce two speed up methods in this section. The first method breaks the synthesis process into two or more smaller synthesis stages. The second method constrains the location of certain gates (such as the controlled-V or controlled-V^+ gates), which reduces the search space of the algorithm.

A. Synthesis in Multiple Stages

We devised a strategy to speed up the synthesis process at the expense of a higher circuit cost. Given an $n \times n$ reversible gate to synthesize, there are 2^n cases to be enumerated. Assume, however, that we pick one of the inputs, say the first input, and consider only cases where it is 0. Then we have 2^{n-1} cases. To perform reachability analysis, we construct the same FSM as shown in Fig. 5, but check it with a different invariant $inv'(t)$

$$\neg \left[\bigwedge_{h=1}^{2^{n-1}} (\vec{\mu}(M_h, t) = R(\vec{\mu}(M_h, 0))) \wedge \bigwedge_{h=1}^{2^n} (\varepsilon(M_h, t) = 0) \right].$$

The main difference between $inv'(t)$ and $inv(t)$ is that the new invariant $inv'(t)$ checks that R is accomplished for only half of all the possible input patterns, which accounted for those cases where the first input is 0. It is easier to find a counter-example for this new invariant because only half of the cases have to be accomplished. We take a snapshot of all register states at the end of this counter-example and use it as the initial state of the FSM. We then run model checker again with our original invariant $inv(t)$. This time, since we started from a state fairly close to R, it is easier to generate a counter-example. According to Theorem 4, this method guarantees to generate the counter-example if the function that we want to synthesize is reversible.

Theorem 4: Suppose we want to synthesize a reversible function R, and suppose we have already synthesized another reversible function Q, then there exists a reversible function P such that R is equivalent to the cascade of Q and P, i.e., $R = Q \circ P$, where R, Q, and P are all $n \times n$ reversible functions.

Proof: Since Q is reversible, we have function Q^{-1} such that $Q \circ Q^{-1} = I$, where I is the identity function (outputs are equal to inputs). Hence, there exists $P = Q^{-1} \circ R$ such that $Q \circ P = Q \circ Q^{-1} \circ R = I \circ R = R$. ■

B. Constraining Search Space

The runtime complexity of model checking is due to its exhaustive nature. We can introduce more constraints to reduce the search space. For instance, we can limit the location of the data input for the controlled-V and controlled-V^+ gates to a subset of the qubits (such as the first qubit). This example will mean that (5) in Definition 1 will be changed to

$$\begin{aligned}&\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_{L-1} \exists A_{L-1} \exists B_{L-1} \\ &\quad \cdot \left(\forall \vec{U_0} \in \{0,1\}^n \cdot (E_0 = E_L = 0) \wedge \left(\vec{U_L} = R(\vec{U_0}) \right) \right) \\ &\quad \wedge \left(\bigwedge_{i=0}^{L-1} ((G_i = \text{Fig. 1(c)}) \vee (G_i = \text{Fig. 1(d)})) \Rightarrow B_i \right) \\ &\quad \wedge \left(\bigwedge_{i=0}^{L-1} A_i \neq B_i \right). \end{aligned} \tag{15}$$

Once formula (5) is changed, all subsequent logic reasoning can be adjusted for the constraint as well.

Formula (15) is just an example to limit the location of the V input to the first qubit. Similar constraints can be constructed to limit the location of the control input for the controlled-V and/or controlled-V^+ gates, or to limit the control or data inputs of the Feynman gates, etc.

VI. Experiments

We constructed our invariant checking formulations described in Section IV using NuSMV with BerkMin [25]. Our method was applied to synthesize some common quantum circuits. All experiments are conducted on a 850-MHz Pentium III processor running on Linux.

The quantum costs of several circuits are summarized in Table I. The "Prior" and "Our" columns indicate the best published quantum cost in previous literature and our synthesized quantum cost, respectively. For Miller gate [26], our synthesis result has a quantum implementation cost of 6, shown in Fig. 6. It is better than any previously published result (cost of 7) [26], [27].

TABLE I
Quantum Cost of Common Circuits

Circuit	Prior	Our	Constraint	Circuit	Time (sec)
Miller	7	6	No	Fig. 6	318.29
Fredkin	5	5	No	Fig. 7	78.02
Peres	4	4	No	Fig. 8	35.18
Toffoli	5	5	No	Fig. 9	122.52
Half-adder	6	4	No	Fig. 10	6.77
Half-adder2	N/A	4	No	Fig. 11	26.25
q4-example	N/A	5	Yes	Fig. 12	34.78
Peres-double	8	6	Yes	Fig. 14	171.27
Toffoli-double	10	7	Yes	Fig. 15	853.78

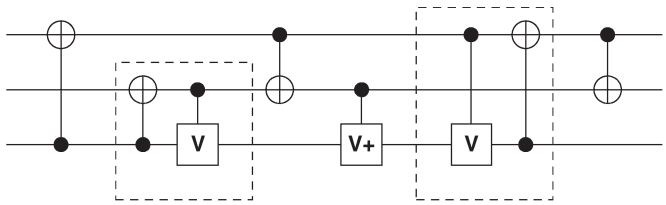


Fig. 6. Miller gate with optimum quantum cost = 6.

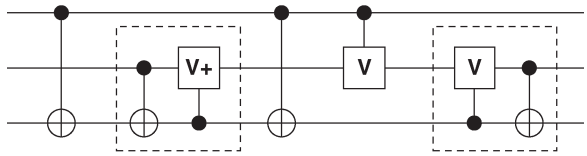


Fig. 7. Alternative Fredkin gate implementation with quantum cost = 5.

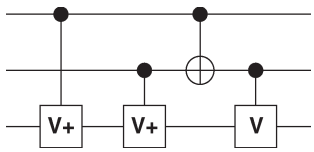


Fig. 8. Peres gate implementation with optimum quantum cost = 4.

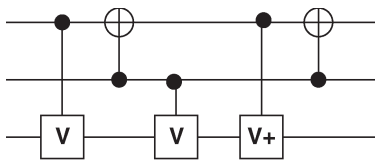


Fig. 9. Toffoli gate implementation with optimum quantum cost = 5.

For the Fredkin [10], Peres [17], and Toffoli [5], [16] gates, our synthesized results (Fig. 7–9) have the same quantum costs as reported in prior literature [4], [27]. But nobody was able to show that the cost was minimum until now. Notice also that our synthesized Fredkin circuit (Fig. 7) is different from the circuit in [4], but they are functionally equivalent (due to the canonical unitary matrix as described in Theorem 1).

We synthesized a classical half adder using input constants discussed in Section IV-B. In the past, people have been synthesizing the 2-bit adder using a Toffoli gate and a quantum XOR gate [23], [28]. Since the Toffoli gate has a minimum cost of 5 and the quantum XOR gate cost 1, the total quantum

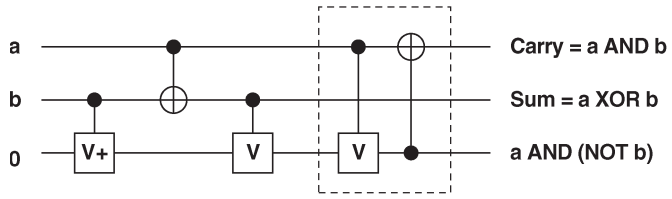


Fig. 10. Half adder with quantum cost = 4.

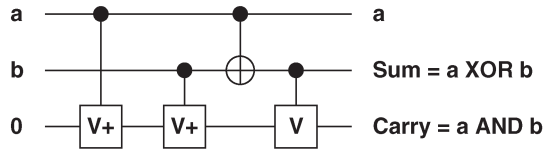


Fig. 11. Alternative half adder with quantum cost = 4.

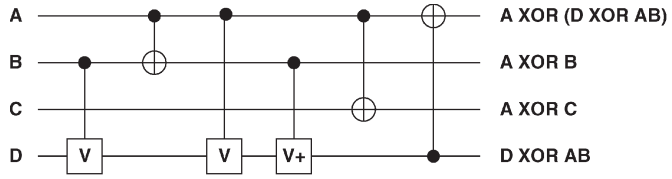


Fig. 12. q4 example.

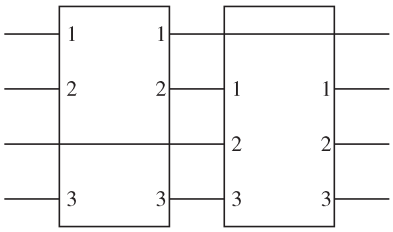


Fig. 13. Peres-double and Toffoli-double specification.

implementation cost would be 6 using that method. Our method proved that the minimum quantum cost is actually 4, as shown in Fig. 10. In fact, if we do not restrict the output of the adder to be the top two qubits, we can put one of the desired outputs on the ancilla qubit. Such an implementation is actually the Peres circuit with the last qubit input set to zero, shown in Fig. 11.

We also synthesized several 4-qubit functions using the method in Section V-B by restricting the data input/output of the controlled-V or controlled-V^+ gates to be the fourth qubit. The "q4-example" is a simple 4-qubit function shown in Fig. 12. The "Peres-double" and "Toffoli-double" functions are specified by cascading two 3-qubit Peres and Toffoli functions, respectively, in a 4-qubit manner shown in Fig. 13, where the numbers 1–3 indicates the input/output correspondence to the first, second, and third qubit of the original Peres or Toffoli functions. Since the smallest quantum cost of Peres and Toffoli gates are known to be 4 and 5, respectively, the quantum cost of having two Peres and Toffoli in a cascading manner would be 8 and 10, respectively. However, our synthesis result indicate that their quantum cost can be heuristically decreased to 6 (Fig. 14) and 7 (Fig. 15), respectively.

We synthesized the full adder using four different strategies shown in Table II. Recent papers [6], [7] used two Toffoli gates and two Feynman gates to implement a quantum cost

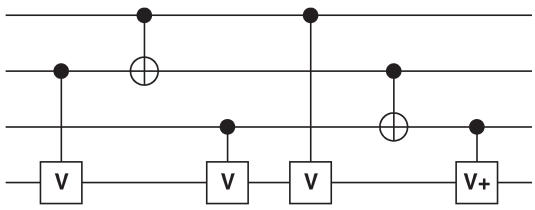


Fig. 14. Peres-double quantum cost = 6.

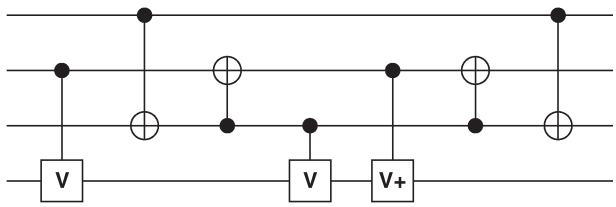


Fig. 15. Toffoli-double quantum cost = 7.

TABLE II
SYNTHESIS OF FULL ADDER

Method	Garbage input to output	Circuit	Cost	Time
Optimal	Yes	Fig. 16	6	7 hours
2-stage	Yes	Fig. 17	9	140.83 sec
Constraint	Yes	Fig. 16	6	1104.97 sec
Constraint	No	Fig. 18	6	176.09 sec

of 12. We proved that the minimum quantum cost for a full adder is 6, as shown in Fig. 16. To shorten the CPU runtime for synthesizing the full adder, we used a two-stage strategy mentioned in Section V-A and obtained an implementation with quantum cost of 9, shown in Fig. 17. The CPU runtime is significantly reduced (from 7 h to 140.83 s). Notice that the cost of this implementation can be reduced to 8 if we choose to omit the "propagate" logic (the last quantum XOR gate). We also applied the synthesis method in Section V-B by restricting the data input of the controlled-V or controlled-V^+ gates to the location of the "sum" qubit. The runtime is reduced from 7 h to 1104.97 s, and the quantum cost is the same as the original optimal method. All the top three experiments in Table II use a specification such that the useful output (sum and carry-out) does not use the same qubit as the garbage input (ancilla qubit). We remove this requirement in the last experiment of Table II and used the input/output specification in [6] and [7]. The result is shown in Fig. 18 and the synthesis took 176.09 CPU seconds.

VII. CONCLUSION

In this paper, we applied invariant checking, a formal verification technique, to the synthesis of quantum logic circuits. We reduced problems in quantum logic synthesis to those of multiple-valued logic synthesis, thus simplifying the search space and algorithm complexity. To solve the synthesis problem, we created an optimal synthesis method, a multistage synthesis method, and several constraint-related speed-up methods. Our optimal method and the multistage method are guaranteed to synthesize the circuit. We created minimum cost quantum circuits for Miller gate, half adder, and full adder, which are better than previous results. This cost is minimum for any circuit using our set of quantum gates, where the control qubit

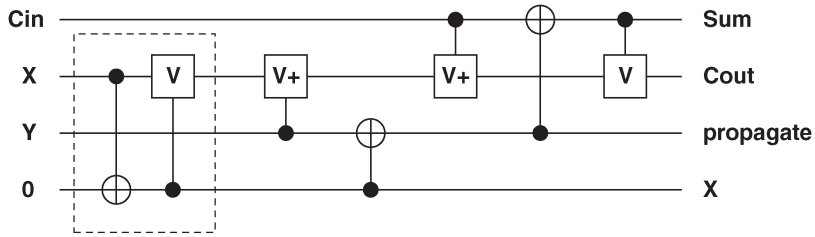


Fig. 16. Full adder with quantum cost = 6.

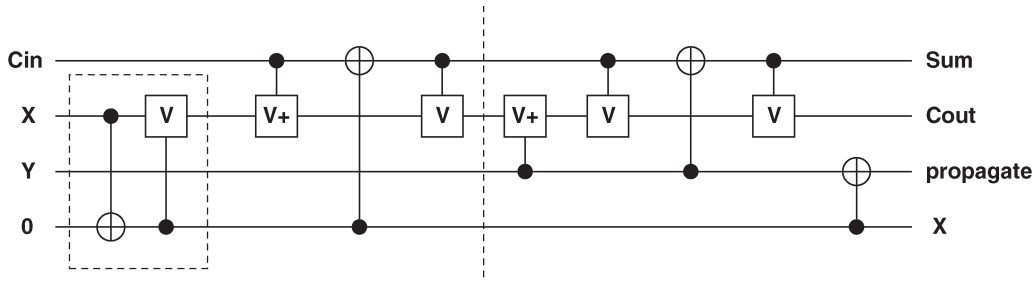


Fig. 17. Full adder with quantum cost = 9.

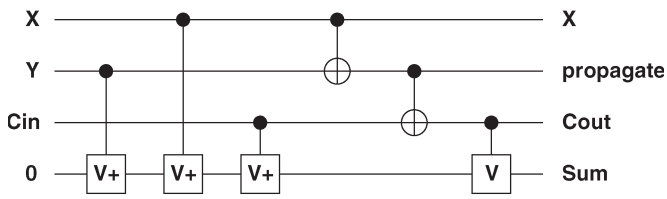


Fig. 18. Full adder with different output arrangement: quantum cost = 6.

of 2-qubit gates is always basis binary. We also proved the minimum quantum cost in the same manner for Fredkin, Peres, and Toffoli gates. In addition, we found quantum implementations with lower cost (than previous known results) for (cascaded) double Peres gates and (cascaded) double Toffoli gates. As shown in Section VI, our method can also automatically convert a nonreversible (irreversible) function to the simplest equivalent reversible function (Fig. 16) by adding and initializing the minimum number of ancilla wires. This step is missing from most reversible circuit synthesis algorithms, and the problem of minimal convertion was never discussed in the literature. We have demonstrated our method on small circuits. It can be a starting point to create such methods for larger Boolean functions. Our work is the first successful application of formal methods and satisfiability in quantum logic synthesis.

REFERENCES

[1] A. De Vos, "Reversible computing," *Prog. Quantum Electron.*, vol. 23, no. 1, pp. 1–49, Jan. 1999.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, Dec. 2000.

[3] K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," in *Proc. Design Automation Conf.*, New Orleans, LA, 2002, pp. 419–424.

[4] J. A. Smolin and D. P. DiVincenzo, "Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate," *Phys. Rev. A, Gen. Phys.*, vol. 53, no. 4, pp. 2855–2856, Apr. 1996.

[5] A. Barenco *et al.*, "Elementary gates for quantum computation," *Phys. Rev. A, Gen. Phys.*, vol. 52, no. 5, pp. 3457–3467, Nov. 1995.

[6] A. Khlopotine, M. Perkowski, and P. Kerntopf, "Reversible logic synthesis by iterative compositions," in *Proc. Int. Workshop Logic Synthesis*, New Orleans, LA, 2002, pp. 261–266.

[7] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Proc. Design Automation Conf.*, Anaheim, CA, 2003, pp. 318–323.

[8] V. V. Shende *et al.*, "Synthesis of reversible logic circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 6, pp. 710–722, Jun. 2003.

[9] X. Song *et al.*, (2005). Algebraic characteristics of reversible gates *Theory of Computing Systems* [Online]. Available: http://www.springerlink.com/link.asp?id=vh2mkff02xwm2gdb, Article In Press

[10] E. Fredkin and T. Toffoli, "Conservative logic," *Int. J. Theor. Phys.*, vol. 21, no. 3/4, pp. 219–253, 1982.

[11] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA: Kluwer, 1993.

[12] A. Biere *et al.*, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 317–320.

[13] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Phys. Rev. A, Gen. Phys.*, vol. 47, no. 10, pp. 777–780, Mar. 1935.

[14] D. Deutsch, "Quantum computational networks," *Proc. Roy. Soc. Lond. A, Math. Phys. Sci.*, no. 425, pp. 73–90, 1989.

[15] J. A. Jones and M. Mosca, "Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer," *J. Chem. Phys.*, vol. 109, no. 5, pp. 1648–1653, Aug. 1998.

[16] T. Sleator and H. Weinfurter, "Realizable universal quantum logic gates," *Phys. Rev. Lett.*, vol. 74, no. 20, pp. 4087–4090, May 1995.

[17] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A, Gen. Phys.*, vol. 32, no. 6, pp. 3266–3276, Dec. 1985.

[18] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[19] W. N. N. Hung *et al.*, "BDD minimization by scatter search," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 8, pp. 974–979, Aug. 2002.

[20] F. Copty *et al.*, "Benefits of bounded model checking at an industrial setting," in *Proc. Computer-Aided Verification*, Paris, France, 2001, pp. 436–453.

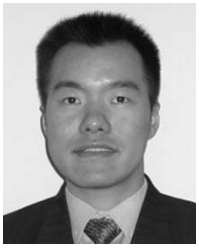
[21] W. N. N. Hung *et al.*, "Segmented channel routability via satisfiability," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 9, no. 4, pp. 517–528, Oct. 2004.

[22] W. N. N. Hung and N. Narasimhan, "Reference model based RTL verification: An integrated approach," in *Proc. IEEE Int. High Level Design Validation Test Workshop*, Sonoma Valley, CA, Nov. 2004, pp. 9–13.

[23] A. Ekert, P. Hayden, and H. Inamori, "Basic concepts in quantum computation," in *Coherent Atomic Matter Waves—Ondes de matiere coherentes*. Berlin, Germany: Springer-Verlag, Aug. 1999, pp. 659–699.

[24] L. M. K. Vandersypen *et al.*, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, Dec. 2001.

[25] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT solver," in *Proc. DATE*, Paris, France, 2002, pp. 142–149.
[26] D. M. Miller, "Spectral and two-place decomposition techniques in reversible logic," in *Proc. IEEE Midwest Symp. Circuits Systems*, Tulsa, OK, Aug. 2002, pp. II-493–II-496.
[27] G. Yang, W. N. N. Hung, X. Song, and M. Perkowski, "Majority-based reversible logic gates," *Theor. Comput. Sci.*, vol. 334, no. 1–3, pp. 259-274, Apr. 2005.
[28] J. Gruska, *Quantum Computing*. New York: McGraw-Hill, Apr. 1999.



William N. N. Hung received the B.S. and M.S. degrees in electrical and computer engineering from the University of Texas, Austin, in 1994 and 1997, respectively, and the Ph.D. degree in electrical and computer engineering from Portland State University, Portland, OR, in 2002.

From 1997 to 2004, he was a Senior Engineer at Intel Corporation, Hillsboro, OR, primarily focused on formal property verification of CPU designs. Since September 2004, he has been a Senior Staff Engineer at Synplicity Inc., Sunnyvale, CA. His research interests include logic synthesis, physical design, formal methods, satisfiability, combinatorial optimization, and quantum computing.

Dr. Hung served as a member in the Emergent Technologies Technical Committee for the IEEE Computational Intelligence Society. He also served as a member in the Program Committee of the IEEE/ACM Design Automation and Test in Europe (DATE) and in the Program Committee of the IEEE International Computer Software and Applications Conference (COMPSAC).

Xiaoyu Song received the Ph.D. degree in computer engineering from the University of Pisa, Pisa, Italy, in 1991.

From 1992 to 1999, he was on the faculty at the University of Montreal, Canada. In 1998, he was a Senior Technical Staff member at Cadence, San Jose, CA. Since 1999, he has been on the faculty at the Department of Electrical and Computer Engineering, Portland State University, Portland, OR. His research interests include synthesis, verification, and testing of high-performance digital system designs, low-power digital IC designs, timing analysis, and formal methods.

Dr. Song served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.

Guowu Yang received the B.S. degree in mathematics from the University of Science and Technology, China, in 1989, and the Ph.D. degree in electrical and computer engineering from Portland State University, Portland, OR, in 2005.

He is currently a Post-Doctoral Researcher in the Department of Computer Science, Portland State University. His research interests include quantum computing, reversible logic, hardware formal verification, floor planning, and routing.

Jin Yang received the B.S. and M.S. degrees in computer science from Peking University, Beijing, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the University of Texas, Austin, in 1997.

He was a Faculty Member at Peking University for two years before he came to the U.S. In 1995, he joined Intel, Hillsboro, OR, and is currently a Principal Engineer at Intel Strategic CAD Labs. He holds five U.S. patents. His research interests include in all aspects of formal methods, with a focus on developing practical solutions for hardware specification and verification.

Marek Perkowski received the M.S. degree in electronics in 1970 and the Ph.D. degree in automatic control in 1980 from the Technical University of Warsaw, Warsaw, Poland.

He has been on the faculty of Warsaw Technical University, The University of Minnesota, Minneapolis, and the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He has been a Visiting Faculty Member at the Technical University of Eindhoven, The Netherlands, the University of Montpellier, France, and Kyushu Institute of Technology, Japan. He worked as Summer Professor at Intel, GTE, and Sharp, and was a Consultant to several companies including Cypress Semiconductor. He is currently a Professor at Portland State University, Portland, OR. His research interests include quantum computing, automated synthesis of quantum and reversible circuits, testing of quantum circuits, and quantum computational intelligence with intelligent robotics applications.

Dr. Perkowski is the Chair of the IEEE Computer Society Technical Committee on Multiple-Valued Logic.