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).

In this paper, we focus on synthesizing reversible circuits using quantum primitives with the lowest total cost using a 43 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 combinational optimization techniques can be similarly applied here. Our 58 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). 60

We also introduce an automated way of adding ancilla qubits 63 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. 66 In contrast to previous works, which either use permutative 67 quantum gates to design permutative circuits or universal 68 quantum gates to design quantum circuits, we use a subset of 69 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 $n 	imes n$ to denote a reversible function with $n$ Boolean inputs and $n$ Boolean outputs. Given an $n 	imes n$ 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.
85 states, also denoted as $|0\rangle$ and $|1\rangle$, respectively. The qubit state
86 $q$ can be represented by
\[ q = \alpha |0\rangle + \beta |1\rangle \]
87 where $\alpha$ and $\beta$ are both complex numbers and $|\alpha|^2 + |\beta|^2 = 1$.
88 The classical state of binary 0 corresponds to the case where
89 $\alpha = 1$ and $\beta = 0$. Similarly, the classical state of binary 1 cor-
90 responds to $\alpha = 0$ and $\beta = 1$. We refer to them as basis binary
91 0 and basis binary 1, respectively. All other combinations of $\alpha$
92 and $\beta$ are not basis binary. The quantum state of a single qubit
93 is usually denoted by the vector
\[ \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \]
94 Given the state of each qubit, the overall quantum state is a
95 Kronecker product of the states of each qubit. Take two qubits
96 for example
\[ \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \otimes \begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = \begin{pmatrix} u_0 v_0 \\ u_0 v_1 \\ u_1 v_0 \\ u_1 v_1 \end{pmatrix}. \tag{1} \]
97 Notice that if the individual qubits are basis binary, then the
98 Kronecker product is simply an enumeration of all the possible
99 binary values (truth table) of its qubits. If we can use the
100 quantum state of multiple qubits to determine the individual
101 state of each qubit (such as the above case), we call it a
102 separable state. There are some cases where the quantum state
103 cannot be separated into individual states of each qubit, i.e.,
104 we cannot describe (mathematically) the state of each qubit but
105 we can describe the quantum state of all the qubits combined.
106 We call such states entangled states. This idea of entangled
107 state is called quantum entanglement, and it originated from
109 The effect of quantum gates on a quantum state can be
110 described as vector operations, where the quantum gates are
111 represented by unitary matrices. A unitary matrix is a $n \times n$
112 complex matrix $M$ with the property
\[ M \times M^+ = M^+ \times M = I \]
113 where $I$ is the identity matrix and $M^+$ is the conjugate trans-
114 pose (also known as the Hermitian adjoint) of $M$.
115 Given an $n$-qubit quantum gate $G$, we call $G$ a permutative
116 quantum gate if and only if the outputs of $G$ are all basis binary
117 when its inputs are all basis binary, i.e., $G$ is a permutative
118 quantum gate if and only if $G$ implements an $n \times n$ Boolean
119 reversible function (when its inputs are basis binary).
120 A generalized 2-qubit controlled $U$ gate [5] is shown in
121 Fig. 1. Its unitary matrix is
\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & u_{00} & u_{01} \\
0 & 0 & u_{10} & u_{11}
\end{pmatrix}
\]
where the four entries in the right bottom also form a (single 122
qubit) unitary matrix $U$ by itself
\[ U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}. \]
124 It has been shown [4], [5] that permutative quantum logic 125
circuits can be constructed using elementary quantum, XOR, 126
controlled-$V$, controlled-$V^+$, or NOT gates, as shown in Fig. 2. 127
The NOT gate is also called an inverter. Its unitary matrix is
\[ M_{\text{NOT}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \]
129 Quantum XOR gates are also called Feynman gates or 130
controlled-NOT (CNOT) gates. The controlled-$V$ gate’s data 131
output is the same as its data input (B) when its control input 132
(A) value is 0 (FALSE). When its control value is 1 (TRUE), 133
the data output becomes $V$ (input) [2]
\[ V = \frac{1 + i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}, \quad V^+ = \frac{1 - i}{2} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \]
134 Similar rules apply to the controlled-$V^+$ gate, except that its 135
data output becomes $V^+$ (input), where $V^+$ is the Hermitian of 136
$V$, i.e.,
\[ \frac{1 + i}{2} \begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix} \times \frac{1 - i}{2} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \]
137 The quantum XOR (controlled-NOT), controlled-$V$, and 138
controlled-$V^+$ are all special cases of the generalized 139
controlled-$U$ gate, where the matrix $U$ corresponds to $M_{\text{NOT}}$, 140
$V$, and $V^+$, respectively.
141 According to [2], the values $V$ and $V^+$ are constructed 142
such that they are the square root of NOT (i.e., inverter gate): 143
$V \times V = V^+ \times V^+ = M_{\text{NOT}}$. Hence, if the signal $V$ (input) 144
is passed through another controlled-$V$ gate with its control 145
value also equal to 1 (TRUE), the output of the second gate 146
becomes the NOT of the input. 147
The quantum XOR, controlled-$V$, and controlled-$V^+$ gates 148
are $2 \times 2$ gates. They are also called 2-qubit gates. Similarly, the 149
NOT gate (inverter) is a 1-qubit gate. For quantum implementa-
150tion, the cost of 2-qubit gates far exceeds the cost of 1-qubit 151
gates. Hence, in a first approximation, the quantum cost of 152
1-qubit gates is usually ignored in the presence of 2-qubit 153
implementations [5], [14].
154 In this paper, we adopt the quantum gate cost evalu-
155ation introduced in [4]. According to the method in [4], 156
each of the 2-qubit gates (quantum XOR, controlled-$V$, 157
controlled-$V^+$) has a quantum implementation cost of $1.158$
In addition, when both quantum XOR and controlled-$V$ 159
controlled-$V^+$) are operating on the same two qubits in a 160
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.,

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

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

\[
V_0^+ = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ i & i \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ i \end{pmatrix}
\]

\[
V_1^+ = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ i & i \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -i \end{pmatrix}
\]

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-\( \text{NOT} \) (quantum \( \text{XOR} \)), 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-\( \text{NOT}, \) 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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1+i}{2} & \frac{1+i}{2} & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
0.5i \\
0.5 \\
0.5 \\
-0.5i \\
0.5i \\
0.5 \\
0 \\
0.5 - 0.5i
\end{pmatrix}
\]

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 the controlled-\( V^+ \). The controlled-\( \text{NOT} \) also has similar examples.

For the rest of this paper, we focus on synthesizing quantum circuits using our set of quantum gates (\( \text{NOT}, \) controlled-\( \text{NOT}, \) 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 \( V_1, V_0, V_0^+, \) or \( V_1^+ \), respectively. Given a binary basis 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 quantum 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.

**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 2)** The Kronecker product of \( n \) qubits is

\[
\left( \begin{array}{c} \alpha_1 \\ \beta_1 \\ \vdots \\ \alpha_n \\ \beta_n \end{array} \right) \otimes \cdots \otimes \left( \begin{array}{c} \alpha_n \\ \beta_n \end{array} \right) = \left( \begin{array}{c} \alpha_1 \alpha_2 \cdots \alpha_{n-1} \alpha_n \\ \alpha_1 \beta_2 \cdots \alpha_{n-1} \beta_n \\ \vdots \\ \beta_1 \beta_2 \cdots \beta_{n-1} \beta_n \end{array} \right)
\]

For each qubit, \( \alpha \beta \) have only two choices (10 or 01) to be basis binary. There are \( 2^n \) distinct ways for all \( 275 \) \( n \) qubits to be basis binary. Under this circumstance, the above Kronecker product is an enumeration of the truth table patterns for \( \alpha \) and \( \beta \) 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 \( 283 \) for this circuit, respectively. We have

\[
U \times P = Q. \quad (2)
\]

According to Step 2), the vector \( P \) has one entry \( 285 \) equal to 1 and all the other entries are 0. Similarly, \( 286 \) 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 \( 288 \) matrix \( U \) in the \( i \)th row and \( j \)th column, and \( p_i \) and \( 289 \) \( q_i \) to denote the value of vector \( P \) and \( Q \) in the \( i \)th row, respectively.

Given \( 0 \leq i \leq 2^n \), suppose all the entries in the \( i \)th \( 291 \) row of \( U \) are 0, then \( q_i \) will be 0 for all possible \( 293 \) values of \( P \) due to (2). This is a contradiction because \( Q \) has \( 2^n \) rows and \( 2^n \) distinct values, so \( 295 \) \( q_i \) must be 1 for one of those cases. Hence, any row \( 296 \) 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 \( 300 \) 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 \( 302 \) is a contradiction because we can only have one \( 303 \) possibility for \( q_i \) to be nonzero. Hence, every row \( 304 \) of \( U \) must have exactly one nonzero entry. In order \( 305 \) to produce a corresponding 1 in the vector \( Q \), the \( 306 \) nonzero entry in \( U \) must be 1.

Lastly, suppose we have \( u_{ij} = 1 \) and \( u_{kj} = 1 \), both 308 in the \( j \)th column. We can pick a valuation of \( P \) with \( 309 \) \( p_j = 1 \). The corresponding vector \( Q \) will have \( q_i = 310 \) 1 and \( q_k = 1 \). This is again a contradiction since \( 311 \) only one row of vector \( Q \) can be nonzero. Thus, \( 312 \) every column of \( U \) must have exactly one nonzero \( 313 \) entry (which must be 1).

Step 4) There are \( 2^n! \) possibilities for \( U \) to satisfy the property of \( 315 \) (Step 3), which is exactly the number of \( 316 \) distinct permutations. Hence, to each permutation \( 317 \) corresponds a unique unitary matrix \( U \). This completes the proof of Theorem 1.

The importance of the above theorem is that once we have \( 320 \) specified the basis binary input/output behavior of the quantum circuit
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 \( 2 \)-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 \( \vec{U}_i = u_{1i}, u_{2i}, \ldots, u_{ni} \). Each qubit \( (u_{qi}, q = 1, \ldots, n) \) of the stage \( i \) can have four possible values \( 0, 1, \bar{V}_0, \bar{V}_1 \). The output of stage \( i \) is denoted by \( \vec{U}_{i+1} \), i.e.,

\[
\begin{align*}
\end{align*}
\]

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 \( (i) \) 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 \( u_{A_i} \) is not binary (where \( A_i \) is the control qubit).

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 \( G_i, A_i, B_i \) (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

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

where \( G_0, A_0, B_0, G_1, A_1, B_1, \ldots, G_{L-1}, A_{L-1}, B_{L-1} \) 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 329 their merged versions) is to solve L-2Syn with the smallest \( L \) possible number of stages where each stage has a quantum cost of 1.

**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 \( L \).
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 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.

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.

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

$$M_1 : \quad \mu(M_1,0) = 00$$

$$M_2 : \quad \mu(M_2,0) = 01$$

$$\vdots$$

$$M_{2^n} : \quad \mu(M_{2^n},0) = 11\ldots11.$$ (8)

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

$$\text{inv}(t) = \neg \bigwedge_{h=1}^{2^n} (\mu(M_h,t) = R(\mu(M_h,0)) \land \epsilon(M_h,t) = 0)$$

where $\text{inv}(t)$ 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 450 sequence $G_t, A_t, B_t$ for $t = 0, \ldots, L$) that satisfies (6)–(8) and violates the invariant $\text{inv}(t)$ at time $t = L$, where $L$ is the 452
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 $G_i, A_i, B_i$ 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

$$\forall \vec{U}_{t=0} \in \{0, 1\}^n \cdot E_{t=0} = 0.$$  \hspace{1cm} (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

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

can be rewritten as $\forall \vec{U}_{t=0} \in \{0, 1\}^n \cdot (E_0 = 0) \land f(\vec{U}_{t}, E_t)$.

We can conjunct the violated invariant $inv(t)$ with the initial condition (9) and rewrite them as

$$\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_L \exists A_L \exists B_L \cdot \forall \vec{U}_{t=0} \in \{0, 1\}^n$$

\begin{equation}
\begin{aligned}
& (E_{t=0} = 0) \land (E_{t=L} = 0) \land \left( \vec{U}_{t=L} = R(\vec{U}_{t=0}) \right). \hspace{1cm} (10)
\end{aligned}
\end{equation}

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

$$\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_L \exists A_L \exists B_L$$

\begin{equation}
\begin{aligned}
& \left( \bigwedge_{t=0}^L A_t \neq B_t \right) \land \forall \vec{U}_{t=0} \in \{0, 1\}^n \\
& \cdot (E_{t=0} = E_{t=L} = 0) \land \left( \vec{U}_{t=L} = R(\vec{U}_{t=0}) \right). \hspace{1cm} (11)
\end{aligned}
\end{equation}

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 inputs $G_i, A_i, B_i$ 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 $G_{t}, A_{t}, B_{t}$ 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 synthesizable, 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 synthesizable.

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 512 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 517 counter-example.

The invariant checking formulation with $k$ input constants is slightly different from Section IV-A. For every input constant $2^{\text{bit}}$, 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 524 figure, we have $2^n$ machines ($M_1, \ldots, M_{2^n}$), each with $n$ 525 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 $\delta$ if applicable) functional block will handle $n + k$ instead of $n$ registers, as well as the $E$ register.

For notational clarity, we still use $\vec{\mu}(M, t)$ 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 $\nu_j(M, t)$ 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 $\nu_{n+1}(M_1, t), \ldots, \nu_{n+k}(M_1, t)$. 535

Let us also introduce a new state $\zeta$ 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.,

$$\zeta = \begin{cases} 0, & t = 0 \\ 1, & t > 0. \end{cases} \hspace{1cm} (12)$$

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 to the set $\{0, 1\}$. In addition, we want to restrict the initial values of the ancilla qubits to $\{0, 1\}$. In addition, we want to restrict the initial values of the ancilla qubits to $\{0, 1\}$. In addition, we want to restrict the initial values of the ancilla qubits to $\{0, 1\}$.
state of the \( j \)th register \((j = n + 1, \ldots, n + k)\) at each machine

to be the same

\[
\bigwedge_{j=n+1}^{n+k} \bigwedge_{i=1}^{2^n} v_j(M_i, 0) \in \{0, 1\}
\]

(13)

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 \( \delta \) 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.

### 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 \((ab = 01, 10)\) to produce the same output pattern. Therefore, one needs to add a single input constant in order to separate 01 and 10 to create a \( 3 \times 3 \) reversible function \( R \). We construct the \( 2^2 \) machines and the invariant according to Sections IV-A and IV-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

\[
\begin{align*}
M_1 : \ &\bar{\mu}(M_1, 0) = 000 \\
M_2 : \ &\bar{\mu}(M_2, 0) = 001 \\
M_3 : \ &\bar{\mu}(M_3, 0) = 010 \\
M_4 : \ &\bar{\mu}(M_4, 0) = 011.
\end{align*}
\]

The sequence of input values in the counter-example is

\[
\begin{align*}
G_0 &= V^+, \quad A_0 = 2, \quad B_0 = 3 \\
G_1 &= \text{XOR}, \quad A_1 = 1, \quad B_1 = 2 \\
G_2 &= V, \quad A_2 = 2, \quad B_2 = 3 \\
G_3 &= V_{\text{XOR}}, \quad A_3 = 1, \quad B_3 = 3.
\end{align*}
\]

Notice that the above input sequence satisfies the constraint (6).

If we substitute these inputs back into the FSM, we will arrive at

\[
\begin{align*}
M_1 : \ &\bar{\mu}(M_1, 4) = 000 \\
M_2 : \ &\bar{\mu}(M_2, 4) = 110 \\
M_3 : \ &\bar{\mu}(M_3, 4) = 010 \\
M_4 : \ &\bar{\mu}(M_4, 4) = 001.
\end{align*}
\]

We compare the final state with the initial state. The initial 573 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 575 patterns in a truth table. Notice that the least significant bit 576 (rightmost column) in the final state satisfies the carry function, 577 and the middle bit (middle column) satisfies the summation 578 function. The bottom bit in Fig. 10 is a garbage function \( a \land \neg b \). 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. 582 The gate types (quantum XOR, controlled-V, etc.) are already given by \( G_0, \ldots, G_3 \) of the counter-example. The connection 584 of these gates to qubits are given by \( A_0, B_0, \ldots, A_3, B_3 \). These 585 values directly translate to the circuit in Fig. 10.

### V. Complexity and Time

Industrial experience [20], [22] suggests that the complexity 588 of model checking is sensitive to the number of state retaining 589 elements in the FSM. For our FSM in Fig. 5, there are \( n = 590 \) 2\( n \) registers, where \( n \) is the number of qubits. Each register 591 has four possible values \( 0, 1, V, V^+ \). If we use Boolean 592 states to encode these registers, we have \( 2n \times 2n \) Boolean state 593 elements. However, the number of qubits \( n \) tends to be small 594 due to physical limitations. So far, the largest number [24] of 595 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 speedup 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 601 perform reachability analysis, we construct the same FSM as shown in Fig. 5, but check it with a different invariant \( inv(t) \) in

\[
\neg \left[ \bigwedge_{h=1}^{2^{n-1}} (\bar{\mu}(M_h, t) = R(\bar{\mu}(M_h, 0))) \land \bigwedge_{h=1}^{2^n} (\varepsilon(M_h, t) = 0) \right].
\]
The main difference between \( \text{inv}'(t) \) and \( \text{inv}(t) \) is that the new invariant \( \text{inv}'(t) \) 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 \( \text{inv}(t) \). 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 \( R = Q \circ P \), 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 \( Q \circ P = Q \circ Q^{-1} \circ R = I \circ R = R \). \( \blacksquare \)

### 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

\[
\exists G_0 \exists A_0 \exists B_0, \ldots, \exists G_{L-1} \exists A_{L-1} \exists B_{L-1} \cdot \\
\left( \forall \vec{U}_0 \in \{0,1\}^n \cdot (E_0 = E_L = 0) \land \left( \vec{U}_L = R(\vec{U}_0) \right) \right) \\
\land \left( \bigwedge_{i=0}^{L-1} \left( (G_i = \text{Fig. 1(c)}) \lor (G_i = \text{Fig. 1(d)}) \Rightarrow B_i \right) \right) \\
\land \left( \bigwedge_{i=0}^{L-1} A_i \neq B_i \right). 
\]

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].

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], [26], [27]. But nobody was able to prove the cost was minimum until now. Notice also that our 644 synthesized Fredkin circuit (Fig. 7) is different from the circuit 665 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 670 discussed in Section IV-B. In the past, people have been syn-thesizing the 2-bit adder using a Toffoli gate and a quantum XOR gate [23], [28]. Since the Toffoli gate has a minimum quantum cost of 5 and the quantum XOR gate cost 1, the total quantum cost was 672.
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.

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

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

TABLE II
SYNTHESIS OF FULL ADDER

<table>
<thead>
<tr>
<th>Method</th>
<th>Garbage input to output</th>
<th>Circuit</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Yes</td>
<td>Fig. 16</td>
<td>6</td>
<td>7 hours</td>
</tr>
<tr>
<td>2-stage</td>
<td>Yes</td>
<td>Fig. 17</td>
<td>9</td>
<td>140.83 sec</td>
</tr>
<tr>
<td>Constraint</td>
<td>Yes</td>
<td>Fig. 16</td>
<td>6</td>
<td>1104.97 sec</td>
</tr>
<tr>
<td>Constraint</td>
<td>No</td>
<td>Fig. 18</td>
<td>6</td>
<td>176.09 sec</td>
</tr>
</tbody>
</table>

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 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 707 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. 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...
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 conversion 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.

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.

Fig. 19. Full adder with quantum cost = 9.

Fig. 20. Full adder with different output arrangement: quantum cost = 9.

REFERENCES


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.

Jin Yang received the B.S. and M.S. degrees in computer science from Peking University, China, 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 all aspects of formal methods, with a focus on developing practical solutions for hardware specification and verification.

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.

Xiaoyu Song received the Ph.D. degree 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.

Marek Perkowski received the M.S. degree in electronics and the Ph.D. degree in automatic control from the Technical University of Warsaw, Warsaw, Poland. He has been on the faculty of Warsaw Technical University, the University of Minnesota, 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.
AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

AQ1 = Please provide issue number in Ref. [14].
AQ2 = Please provide publisher location in Ref. [23].
AQ3 = Please provide conference location in Ref. [26].
AQ4 = Please provide publisher location in Ref. [28].
AQ5 = Please specify the field of study.
AQ6 = Please specify the year when the degrees were earned.
AQ7 = Please specify the year when the degrees were earned.
Note: Please provide photo for authors G. Yang and J. Yang.

END OF ALL QUERIES