Property Checking of Quantum Circuits Using Quantum Multiple-Valued Decision Diagrams

Julia Seiter[1], Mathias Soeken[1,2], Robert Wille[1], and Rolf Drechsler[1,2]

[1] Institute of Computer Science, University of Bremen
Group of Computer Architecture, D-28359 Bremen, Germany
{jseiter,msoeken,rwille,drechsle}@informatik.uni-bremen.de
[2] Cyber-Physical Systems
DFKI GmbH, D-28359 Bremen, Germany
rolf.drechsler@dfki.de

Abstract. For the validation and verification of quantum circuits mainly techniques based on simulation are applied. Although lots of effort has been put into the improvement of these techniques, ensuring the correctness still requires an exhaustive consideration of all input vectors. As a result, these techniques are particularly insufficient to prove a circuit to be error free.

As an alternative, we present a symbolic formal verification method that is based on Quantum Multiple-Valued Decision Diagrams (QMDDs), a data-structure allowing for a compact representation of quantum circuits. As a result, using QMDDs it is possible to check the correctness of a circuit without exhaustively considering all input patterns.

1 Introduction

Quantum computation [1] has received significant attention in recent years. Using quantum circuits many important problems such as factorization or database search can be solved quadratically or even exponentially faster in comparison to conventional technologies. As a result, much effort has been spent in the past on how to design the corresponding circuit structures. In particular synthesis received much attention (see e.g. [2–6]). But besides realizing quantum circuits for a given problem, verification and validation is an essential step that ensures whether obtained designs realize the desired functionality or not.

Considering conventional circuit design, verification has become one of the most important steps in the design flow. As a result, very powerful approaches have been developed in this domain, ranging from simulative verification (see e.g. [7–10]) to formal equivalence checking (see e.g. [11, 12]) and property checking (see e.g. [13, 14]).

For quantum computation, verification is still at the beginning. Even if first approaches in this area exist (a brief outline is provided in Sect. 3.2), they are mainly based on simulation, i.e. ensuring the correctness still requires an exhaustive consideration of the input vectors. As a result, these techniques are particularly insufficient to prove a circuit to be error-free.

In this work, we present an alternative solution to the property checking problem of quantum circuits which makes use of symbolic formal verification. More precisely, for this task we consider *Quantum Multiple-Valued Decision Diagrams* (QMDDs, [15]), which is a data-structure that allows for a compact representation of quantum circuits. Using QMDDs both a given quantum circuit and the property to be verified can be efficiently represented. Then, checking whether the circuit satisfies the considered property or not can be conducted on this data-structure. The properties themselves are thereby provided by means of a combinatorial and LTL-like language.

Experiments show that, in comparison to state-of-the-art simulation methods, the proposed approach is more robust. While simulation-based approaches work faster for failing properties (where the simulation can immediately be terminated once a counter-example is obtained), QMDDs clearly outperform simulation for holding properties where all possible input patterns need to be traversed exhaustively.

The remainder of this paper is structured as follows. The next section briefly reviews the basics on quantum circuits and the QMDD data-structure. Section 3 formally defines the considered problem, discusses related work, and introduces the general idea of the proposed solution. Afterwards, implementation details are described in Sect. 4. A summary of the experimental evaluation and conclusions are provided by means of Sect. 5 and Sect. 6, respectively.

2 Preliminaries

In order to keep the paper self-contained, this section reviews the basics on quantum circuits and the QMDD data-structure applied in this work. Due to page limitations the following descriptions are kept brief. We refer the reader to [1] and [15] for a more detailed treatment of quantum circuits and QMDDs, respectively.

2.1 Quantum Circuits

In quantum computation [1], *qubits* are the elementary information elements. A qubit is usually denoted by its state $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ which is a *superposition* of the basis states $|0\rangle$ and $|1\rangle$ with α and β being *amplitudes* such that $|a|^2 + |b|^2 = 1$. Qubits can be composed in terms of quantum registers $|\varphi_1 \dots \varphi_n\rangle = \sum_{i=0}^{2^n-1} \alpha_i|i\rangle$, where $|i\rangle$ denotes the conventional state on n bits. Quantum registers are elements in the 2^n-dimensional complex Hilbert space $\mathcal{H}$. Quantum computation can be performed using unitary matrices that are closed under $\mathcal{H}$.

Established quantum operations include e.g. $\sigma_x = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ (also known as Pauli-X or NOT operation) and $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ (also known as Hadamard operation). The first operation interchanges the amplitudes of a quantum state whereas the latter operation can be used to bring a conventional state into a superposition, e.g. $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. While these operations act on single

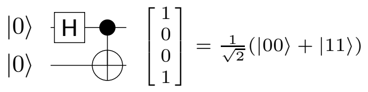

$|0\rangle$
H
$|0\rangle$
$\begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

Fig. 1. Circuit that entangles two qubits

qubits only, they can be extended to also act on quantum registers. This can either be accomplished by the parallel composition using the Kronecker product ($\otimes$) or by controlling the single-qubit operation by one further qubit. As an example, a Pauli-X operation controlled by another qubit can be represented by $\left[\begin{smallmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{smallmatrix}\right]$. Such an operation is called *controlled NOT* (CNOT).

Quantum circuits allow for a graphical representation of the composition of several quantum operations. To this end, a circuit line is drawn horizontally for each qubit. The quantum gates that represent quantum operations are drawn as a cascade from left to right. Control lines are denoted using ● while a quantum operation U is drawn using a rectangle labeled U. An exception is the Pauli-X operation that is denoted using $\bigoplus$.

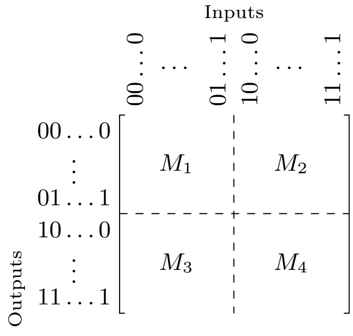
Example 1. Figure 1 shows a circuit that entangles two qubits. The first gate is a Hadamard operation on the first qubit while the second operation is a CNOT. The overall quantum operation of the circuit is $\text{CNOT} \cdot (H \otimes I)$ where $I = \left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right]$ is the 2×2 identity matrix.

2.2 Quantum Multiple-Valued Decision Diagrams

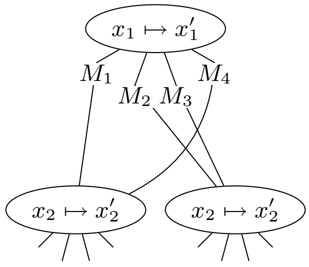
A *Quantum Multiple-valued Decision Diagram* (QMDD, [15]) is a canonical representation of a unitary matrix M and, thus, also of a quantum circuit. It is a directed acyclic graph with one root node and two terminal nodes, labeled $\boxed{0}$ and $\boxed{1}$.

Each node v in a QMDD represents a submatrix of M. In order to build a QMDD, M is divided into four submatrices. The matrix M itself is represented by the root node v_1. Each submatrix is represented by a child of v_1 so that each node in the QMDD has four child nodes. This is repeated recursively for each submatrix until they are reduced to size 1×1. Submatrices of this size are represented accordingly by one of the terminal nodes [15].

Figure 2(a) illustrates this principle by decomposing an arbitrary matrix M. On top of the matrix M, the input patterns are denoted. The corresponding output patterns are denoted to the left-hand side of M. As can be seen, each of the submatrices of M represents an input/output assignment of the very first variable x_1. For example, the submatrix M_1 represents the mapping of x_1 from 0 to 0. Figure 2(b) shows the respective submatrices represented in terms of successors of the root node. The submatrices of M are assigned to the child nodes as follows.



(a) Matrix



(b) QMDD

Fig. 2. Matrix with input output mappings and QMDD

- The upper left submatrix M_1 is the first child and represents the mapping $0 \mapsto 0$ of the input x_1 to the output x_1'.
- The upper right submatrix M_2 is the second child and represents the mapping $1 \mapsto 0$ of the input x_1 to the output x_1'.
- The lower left submatrix M_3 is the third child and represents the mapping $0 \mapsto 1$ of the input x_1 to the output x_1'.
- The lower right submatrix M_4 is the fourth child and represents the mapping $1 \mapsto 1$ of the input x_1 to the output x_1'.

In a QMDD, each edge is labeled with a weight. If a submatrix consists of 0 or 1 entries, the respective edge is annotated with the weight 1. Submatrices consisting only of 0-entries are represented by edges leading directly to $\boxed{0}$. In order to keep the QMDD readable, these edges are drawn by a 0-stub. Also, the weight 1 is often omitted as 1 defines the default weight. In case of a submatrix containing complex-valued entries, the respective edge is annotated with a complex-valued weight. More precisely, if all entries have the value w or are a multiple of w, then the edge is labeled with the weight w. The final value of an entry in the matrix is then computed by multiplying all the weights from the root node to the terminal.

These concepts are illustrated in the following example:

Example 2. Figure 3 shows a QMDD representing the circuit from Fig. 1. The corresponding matrix M is

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} .$$

By dividing M into four submatrices, the child nodes of the root node can be determined. The first and second submatrix contain the same entries and, thus, are represented by a single shared node. The third and fourth submatrix contain entries of the same value but with different signs. Consequently, they

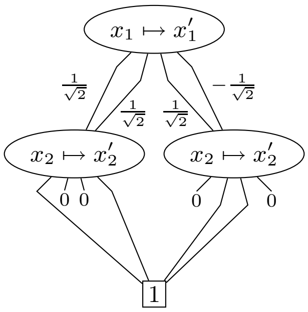


Fig. 3. QMDD representing the circuit from Fig. 1

are represented by the same node, too, but the edges leading to this node are annotated with different weights. The edges leading from the child nodes to the terminals can be determined easily by division of the submatrices. As described above, the edge weight 1 is omitted as well as the terminal $\boxed{0}$ and the edges leading to it. Instead, they lead to a 0-stub.

3 Property Checking of Quantum Circuits

This section formally defines the problem considered in this work and discusses previously introduced approaches which address this issue. Afterwards, the general idea of our solution is proposed.

3.1 Problem Formulation

In this work, we consider property checking of quantum circuits. Property checking is applied during the hardware design phase in order to check whether a designed circuit in fact satisfies the specification or not, i.e. whether the circuit has been designed correctly or not. Since a specification may be very complex, usually several properties are defined which the circuit has to satisfy. Then, these properties are individually checked. The properties describe the intended relation between the inputs and the outputs of the design or they describe requirements which have to be satisfied. If the complete specification is covered by means of properties and, additionally, a realized circuit satisfies all of them, then it has been proven that the circuit was designed correctly. Otherwise, the circuit contains design errors and has to be revised.

Formally a circuit G is considered which realizes the function $f_G : I\!B^n \rightarrow I\!B^n$ with inputs $x_1, \ldots, x_n$ and outputs $x_1', \ldots, x_n'$. The function $f_P : (f_G, X) \mapsto r$ (where X is an input pattern of G and $r \in I\!B$ is the result) evaluates the property P, i.e. f_P maps to 1 if, and only if, the circuit G with the input pattern X satisfies P. Given that, the *property checking* problem is defined by proving that $\forall X. f_P(f_G, X) = 1$ holds for a given circuit G and a given property P.

If f_P maps to 1 for all possible input patterns X of G, then the circuit G satisfies the considered property. However, the evaluation of this formula would require 2^n computations of f_G. Instead, it is often easier to determine a single counter-example X_{CEX} for which the property is not satisfied. This can be expressed as $\exists X_{\text{CEX}}.f_P(f_G, X_{\text{CEX}}) = 0$. This requires the evaluation of all 2^n input patterns only in the worst case.

As a result, property checking can be considered as the problem of determining an input pattern X_{CEX} of G so that G does not satisfy P or to prove that no such pattern exists.

3.2 Quantum Circuit Verification

The verification of conventional hardware is a well-considered field (see e.g. [13]). The approaches developed in the last decades are highly optimized and have been implemented in efficient tools. These accomplishments can be exploited when reversible circuits are considered exclusively. Since here all operations only act on Boolean values, it is sufficient to simply map these circuits to conventional gate libraries and afterwards apply the existing methods such as bounded model checking [16] or equivalence checking [17].

In contrast, quantum circuits contain non-Boolean values and often have non-Boolean outputs. As a consequence, the approaches developed so far for Boolean circuits are not applicable. Yet only few formal verification approaches for quantum circuits exist. In [18], the quantum model checker QMC has been introduced. This model checker has specifically been developed to check properties of quantum protocols. However, it is uncertain whether this approach is applicable to other more generic quantum algorithms or how the model checker behaves when used for the verification of larger systems. To the best of our knowledge, no studies in these fields have been published so far.

Instead, the majority of verification approaches for quantum circuits applies simulation techniques [19–21]. In order to simulate a circuit, several stimuli, i.e. input patterns, are generated and the respective output patterns are produced. Each of these output patterns needs to be checked against the specification in order to determine the correctness of the design. On the one hand, simulation is a very fast and inexpensive method, especially for determining a circuit's behavior when only a few cases are concerned. As a result, it can be used to ensure the correctness for several critical or common input patterns. However, a circuit with n variable inputs has 2^n possible input patterns and, thus, coverage of the entire behavior through simulation is intractable as all 2^n patterns need to be simulated resulting in an impracticably large run-time. Additionally, the operations performed by quantum circuits are usually described by complex-valued matrices. As many simulation approaches apply matrix multiplication, they have high memory requirements as well.

In order to especially address the latter problem, the simulator QuIDDPro has been introduced in [21]. QuIDDPro employs a particular data-structure, so-called *Quantum Information Decision Diagrams* (QuIDDs), to simulate quantum circuits. QuIDDs have been explicitly developed in such a way that they allow

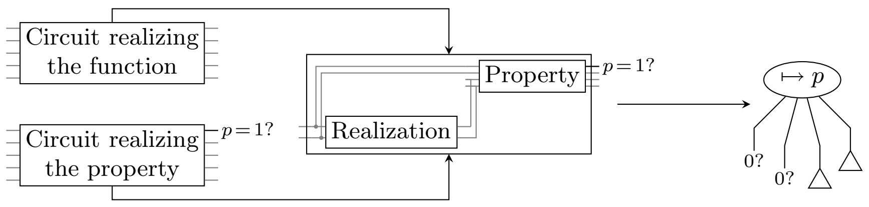

Circuit realizing the function
Circuit realizing the property
$p=1?$
Property
Realization
$p=1?$
$\mapsto p$
0?
0?

Fig. 4. Proposed verification flow

efficient quantum circuit simulation. Existing evaluations show that QuIDDPro outperforms all other known simulation approaches when only one simulation run is considered. However, QuIDDPro is naturally bounded to the number of qubits that correspond to the inputs which again leads to an exponentially growing run-time for a complete simulation.

Since simulation of conventional circuits leads to similar problems, conventional hardware is usually either entirely verified by means of formal methods or just validated by means of partial simulation for certain crucial cases. In this work, we aim for a verification of quantum circuits. That is, we introduce a formal verification approach for quantum circuits in order to provide an alternative to the existing simulation-based approaches.

3.3 General Idea

Figure 4 outlines the underlying idea of the verification flow proposed in this paper. The input to the flow are the circuit under verification and the property to be checked. The property is represented by means of the function f_P realized as a circuit.

Having this, a naïve property check could be done as follows. Each possible input combination is assigned to the inputs of the original circuit which is used to evaluate the corresponding output assignment. Then, both assignments are given as input to the circuit realizing the property (see center of Fig. 4). If the output signal f_P is 1 for each input assignment, the property holds. Although this process can be simplified by combining both circuits and connecting the inputs and outputs of the original function to the property circuit, this will not change the complexity of the verification procedure.

However, with certain modifications which are outlined in detail in the next section, both circuits can be combined and represented by a QMDD. When constructing the corresponding QMDD in a way such that the property signal f_P is located at the top-most line, possible input/output mappings of f_P will be represented by the root node of the QMDD. This easily allows to check whether there exists an input assignment such that f_P evaluates to 0, i.e. such that the property is not satisfied. In fact, it just has to be checked whether the first two successors from the root node lead to path to $\boxed{1}$ (see right-hand-side of Fig. 4). In this case a mapping to $f_P = 0$ exists, i.e. there is a input assignment which violates the property. Hence, checking whether the property holds can be done

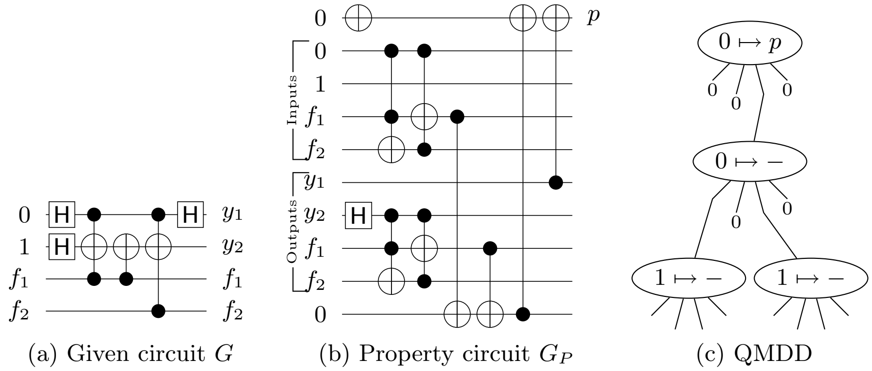


Fig. 5. Application of the proposed verification flow

with a constant number of look-ups after the QMDD has been determined. This general idea is illustrated by the following example.

Example 3. Figure 5(a) shows a generalization of the quantum circuit realizing the Deutsch algorithm [22]. Usually the oracle is the input to the Deutsch problem and the circuit solely consists of constant inputs. The generalization allows for *configuring* all possible four oracles using two additional circuit lines f_1 and f_2 which combinations lead to all truth tables 00, 01, 10, and 11 representing constant 0, identity, negation, and constant 1, respectively.

The property represented by the circuit in Fig. 5(b) will explicitly evaluate the possible functions and afterwards compare the result gathered from the original circuit on signal y_1. A (partial) QMDD representing the combined circuit is depicted in Fig. 5(c)[1]. Since the first two successors of the root node point to $\boxed{0}$, no input assignment exists which maps to $f_P = 0$. Hence, the property has been proven to be true, i.e. the circuit indeed realizes the Deutsch algorithm.

4 Implementation

This section describes the algorithm implementing the idea proposed above and illustrates the resulting verification flow by means of an example. After a brief overview of the main steps, these steps are discussed in detail.

Given are a quantum circuit G and a property P to be verified. The property P is evaluated by f_P (see Sect. 3.1) which is represented by a circuit G_P over the inputs and outputs of G. The property is satisfied if f_P always evaluates to 1, i.e. f_P is tautologous. The aim of the procedure is to obtain a QMDD which represents the result f_P. Consequently, G and G_P have to be altered in such a way that a combined circuit of the two, in the following denoted by G_C, can

[1] Note that this QMDD has been modified to handle constant inputs. This is described in detail in the next section.

be used as a basis for the QMDD Q. Since QMDDs can only represent unitary functions, this particularly includes a transformation of G_C into a unitary function realization. To this end, additional circuit lines (assuming constant inputs) need to be added. Those lines have to be explicitly handled when obtaining the result of the property check (i.e. f_P from the QMDD).

Overall, the respective algorithm executes the following steps.

1. Combine G and G_P to G_C and make G_C unitary
2. Build a QMDD Q for G_C
3. Modify the QMDD Q such that additional circuit lines (assuming constant inputs) are handled
4. Obtain the result of the property check from the root node of the QMDD Q

In the following sections, these steps are described in detail.

4.1 Combine the Circuits and Ensure Unitary

The goal of the first step is to alter both circuits G and G_P in such a way that they can be combined into one circuit G_C by concatenating G and G_P. Based on G_C, a QMDD is built in the next step.

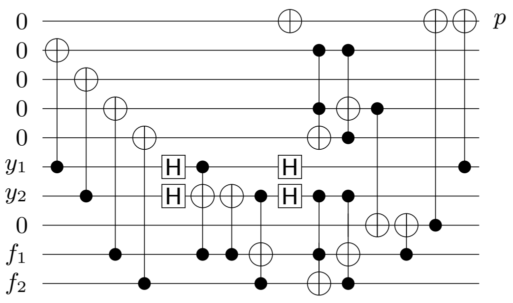
First, G and G_P have to be defined over the same set of variables and the variables have to be in the same order, because they cannot be combined into one circuit otherwise. Consequently, both circuits are checked for variables which occur exclusively in the respective circuit. These variables are then added to the other circuit in such a way that the variable ordering is the same for G and G_P.

Since from the combined circuit G_C a QMDD is supposed to be created, G_C has to represent a unitary circuit which therefore also applies for G and G_P. While it already holds for G, G_P may be non-unitary and, thus, has to be extended accordingly. Here, existing approaches for *embedding* (introduced e.g. in [23]) can be exploited. Furthermore, in order to ensure that also the combined circuit G_C is unitary, the fan-outs have to be removed which are caused by the fact that both G and G_P use the same inputs. This is done by adding additional circuit lines and gates that keep copies of the inputs. The following example illustrates this step.

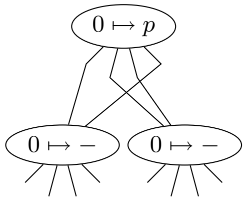
Example 4. Consider again the circuits in Figs. 5(a) and 5(b) representing a given circuit G and a property circuit G_P. As can be seen, three additional lines (assuming constant values) are necessary in order to properly embed G_P. Afterwards, both circuits are combined to G_C as shown in Fig. 6(a). For this purpose, four gates are added replacing the non-unitary fan-outs, i.e. copying the values of the inputs y_1, y_2, f_1, and f_2.

4.2 Build a QMDD from the Combined Circuit

Before building a QMDD from the combined circuit, the lines are reordered in such a way that all lines assuming a constant input value are placed at the top of



(a) Combined circuit



(b) QMDD representing the complete function

Fig. 6. Intermediate steps of the proposed verification flow

the circuit. Such a structure simplifies the succeeding steps. Afterwards, a QMDD is built representing G_C, i.e. the combined circuit of G and G_P. However, the resulting QMDD cannot be used to determine the result of the property check as it does not explicitly handle constant inputs.

Example 5. Figure 6(b) shows the QMDD representing the combined circuit G_C from Fig. 6(a). As can be seen, constant input values have not been considered yet. In fact, the root node has four outgoing edges leading to non-terminal nodes. That is, all possible input/output mappings of the first line (including output f_P) are considered. The underlying circuit however assumes a constant input 0 at this line, i.e. only the first and the third outgoing edge should be considered. Hence, the QMDD needs to be modified as described in the following step.

4.3 Modify the QMDD

Obtaining a QMDD Q which also takes constant input values into consideration requires altering the QMDD in order to avoid an incorrect result. As a consequence, each node representing a line with a constant input value has to be checked. If such a node has outgoing lines representing an input/output assignment which does not occur in the circuit, then the respective edge is replaced by an edge leading to $\boxed{0}$.

As a result from eliminating edges in the QMDD, it can occur that nodes remain whose edges all point to $\boxed{0}$. Since these nodes never appear in paths from the root node to $\boxed{1}$ and thus do not contribute to valid input/output mappings, they can be removed from the QMDD. The resulting QMDD represents the combined circuit G_C and additionally considers constant input values.

Example 6. Figure 5(c) shows the QMDD after edges and nodes have been eliminated as described above. Now, the root node has only one outgoing edge leading to a non-terminal node which corresponds to the defined constant input value.

The second and fourth edge have been eliminated through edge elimination, while the first edge has been eliminated in the course of node elimination.

4.4 Determine the Result

After the steps described above, the resulting QMDD represents all input/output mappings considering the given circuit G, the given property P, and the possibly assumed constant inputs. Of particular interest is whether there exists a mapping to $f_P = 0$. If that is the case, it has been shown that the resulting circuit does not hold the property. Since f_P is the top-most variable and, thus, represented by the root node of Q, the existence of such a mapping can be obtained by evaluating the root node and its outgoing edges. An existing mapping is indicated by the existence of the respective edge.

If the property check does not hold, then the first line maps from 0 to 0 and a counter-example can be derived from the QMDD. Such a counter-example is an input pattern for which the property is not satisfied. It can be obtained by traversing a path from the top-most node to $\boxed{1}$ which starts with the first outgoing edge of the top-most node.

Example 7. The root node of the QMDD in Fig. 5(c) indicates the result of the property check. Only the third edge leads to a non-terminal node. This edge represents the mapping $0 \mapsto 1$ for the top-most line in the combined circuit. Hence, f_P always maps to 1 and, thus, the property always holds.

5 Experimental Results

The verification flow introduced in the previous sections has been implemented and evaluated by verifying several instances of the Grover search and the Deutsch algorithm. The realizations for both algorithms have been generalized as described in Sect. 3.3. The corresponding properties were automatically synthesized. All operations concerning QMDDs, including the construction, were provided as a C-library [15]. The experiments have been conducted on a 2.3 GHz Intel Core i5 with 2GB main memory running Linux as a virtual machine.

The results of the experiments are shown in Table 1. They were compared to those obtained by the simulator QuIDDPro [21] which was applied to the same circuits as the proposed verification procedure. In addition to that, we distinguished between holding and failing property checks. For satisfying properties, the circuits have been completely simulated using QuIDDPro. For failing properties the simulation was only performed until a counter-example was determined.

The table is structured as follows. In the first column, the name of the circuit which was verified is given. In the second and third column, the number of lines and gates is denoted, split into the number of the combined and the original circuits to be checked. The remaining columns show the run-times of QuIDDPro and the QMDD-based verification flow for holding and failing properties.

In case of satisfying properties, the QMDD-based verification approach is faster than QuIDDPro except for one test case. For the smaller circuits, the

Table 1. Results for passing property checks

Circuit	Lines	Gates	Holding properties QuIDDPro	QMDD	Failing properties QuIDDPro	QMDD
Grover2	6 (5)	26 (19)	0.07	**0.01**	**0.03**	1.65
Grover3	8 (7)	111 (83)	0.72	**0.03**	**0.11**	1.67
Grover4	10 (9)	117 (107)	**2.08**	2.83	**0.12**	1.72
Grover5	12 (11)	144 (132)	5.60	**4.48**	**0.16**	1.94
Grover6	14 (13)	165 (151)	10.65	**1.63**	**0.24**	1.68
Grover7	16 (15)	263 (247)	33.27	**1.70**	**0.36**	1.66
Grover8	18 (17)	229 (211)	79.98	**3.03**	**0.28**	8.00
Grover9	20 (19)	182 (162)	timeout	**16.98**	**0.16**	18.00
Grover10	22 (21)	201 (179)	timeout	**230.48**	**0.19**	244.69
Deutsch	12 (6)	21 (8)	0.04	**0.01**	**0.02**	1.66
Deutsch-Josza	23 (22)	75 (22)	0.14	**0.03**	**0.07**	1.67

run-times are similar. However, as the number of lines and gates increases, the QMDD-based approach becomes much faster than QuIDDPro. In particular, the QMDD-based approach could still be applied for the circuits *Grover9* and *Grover10*, while the simulation with QuIDDPro does not finish. This occurs due to the complete simulation which has to be performed by QuIDDPro.

For failing property checks, the results are reversed. Whereas QuIDDPro performs very good, the run-time of the QMDD-based approach is nearly identical to that of the holding property checks. Since QuIDDPro does not need to completely simulate the circuits, but terminates the simulation as soon as a counterexample is determined, the run-time can be reduced significantly. In contrast, the QMDD is built completely for all possible input/output mappings instead of particular simulation patterns.

Although the simulation performs much faster for failing property checks, the proposed verification flow is more robust. As a result, the run-time of the verification does not depend on the outcome of the verification whereas simulation can lead to a timeout.

6 Conclusions

In this work, a new verification approach for quantum circuits has been presented. We described how QMDDs can be applied in order to prove correctness of a design and evaluated our approach by verifying realizations of Grover's algorithm and Deutsch's algorithm. In contrast to QuIDDPro, one of the best known quantum circuit simulators, the run-time of our approach is not dependent on the result of the property check and can prove correctness of a design much faster than the simulator.

Acknowledgments. We would like to sincerely thank D. Michael Miller for providing us with an implementation of the QMDD package introduced in [15] and for many helpful discussions. This work was supported by the German Research Foundation (DFG) (DR 287/20-1) and the German Academic Exchange Service (DAAD).

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, New York (2000)
2. Hung, W.N.N., Song, X., Yang, G., Yang, J., Perkowski, M.A.: Quantum logic synthesis by symbolic reachability analysis. In: Malik, S., Fix, L., Kahng, A.B. (eds.) Design Automation Conference, pp. 838–841. ACM (June 2004)
3. Shende, V.V., Bullock, S.S., Markov, I.L.: Synthesis of quantum logic circuits. In: Tang, T. (ed.) Asia and South Pacific Design Automation Conference, pp. 272–275. ACM Press (January 2005)
4. Große, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact Synthesis of Elementary Quantum Gate Circuits for Reversible Functions with Don't Cares. In: Int'l Symp. on Multiple-Valued Logic, pp. 214–219 (May 2008)
5. Maslov, D., Dueck, G.W., Miller, D.M., Negrevergne, C.: Quantum Circuit Simplification and Level Compaction. IEEE Trans. on CAD 27(3), 436–444 (2008)
6. Soeken, M., Wille, R., Dueck, G.W., Drechsler, R.: Window optimization of reversible and quantum circuits. In: Int'l Symp. on Design and Diagnostics of Electronic Circuits and Systems, pp. 341–345 (April 2010)
7. Yuan, J., Shultz, K., Pixley, C., Miller, H., Aziz, A.: Modeling design constraints and biasing in simulation using BDDs. In: Int'l Conf. on Computer-Aided Design, pp. 584–590 (November 1999)
8. Bergeron, J.: Writing Testbenches Using SystemVerilog. Springer (2006)
9. Yuan, J., Pixley, C., Aziz, A.: Constraint-Based Verification. Springer (January 2006)
10. Wille, R., Große, D., Haedicke, F., Drechsler, R.: SMT-based Stimuli Generation in the SystemC Verification Library. In: Forum on Specification & Design Languages (September 2009)
11. Brand, D.: Verification of large synthesized designs. In: Lightner, M.R., Jess, J.A.G. (eds.) Int'l Conf. on Computer-Aided Design, pp. 534–537. IEEE Computer Society (1993)
12. Disch, S., Scholl, C.: Combinational Equivalence Checking Using Incremental SAT Solving, Output Ordering, and Resets. In: Asia and South Pacific Design Automation Conference, pp. 938–943 (2007)
13. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
14. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
15. Miller, D.M., Thornton, M.A.: QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits. In: Int'l Symp. on Multiple-Valued Logic, p. 30. IEEE Computer Society (May 2006)
16. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 117–148 (2003)
17. Wille, R., Große, D., Miller, D.M., Drechsler, R.: Equivalence Checking of Reversible Circuits. In: Int'l Symp. on Multiple-Valued Logic, pp. 324–330. IEEE Computer Society (May 2009)
18. Gay, S.J., Nagarajan, R., Papanikolaou, N.: QMC: A Model Checker for Quantum Systems. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 543–547. Springer, Heidelberg (2008)

19. Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Phys. Rev. A 70, 052328 (2004)
20. Vidal, G.: Efficient Classical Simulation of Slightly Entangled Quantum Computations. Phys. Rev. Letters 91, 147902 (2003)
21. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Quantum Circuit Simulation. Springer, Heidelberg (2009)
22. Deutsch, D.: Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. Royal Society London A 400(1818) (July 1985)
23. Miller, D.M., Wille, R., Dueck, G.: Synthesizing Reversible Circuits for Irreversible Functions. In: EUROMICRO Symp. on Digital System Design, pp. 749–756 (2009)